Lesson Plan

Understanding Kube-Proxy

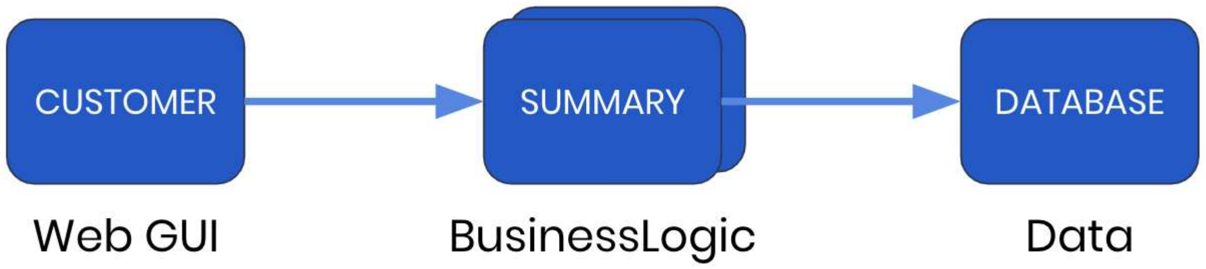
Summary

1. Lab Setup
2. Examine Services
3. Kube-Proxy Cluster IP Implementation
4. Kube-Proxy NodePort Implementation

Lab Setup

This lab assumes you already have the "Yet Another Online Bank" (yaobank) installed, as created in the “Installing the Sample Application” module in Week 1. If you don’t already have it installed then go back and do so now.

For reference, this is architecture of YAOBank:



If you are not already on host1, you can enter host1 by using the multipass shell command.

```
multipass shell host1
```

Examine Kubernetes Services

Let's take a look at the services and pods in the yaobank kubernetes namespace.

List the services:

```
kubectl get svc -n yaobank
```

Example output:

```
NAME       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
database   ClusterIP   198.19.33.51    <none>        2379/TCP       24m
summary    ClusterIP   198.19.36.113   <none>        80/TCP         24m
customer   NodePort    198.19.45.149   <none>        80:30180/TCP   24m
```

We should have three services deployed. One NodePort service (used for the customer front end when accessed from outside the cluster) and two ClusterIP services (used for the summary and database when accessed inside the cluster).

Now let's look at the endpoints for each of the services:

```
kubectl get endpoints -n yaobank
```

Example output:

```
NAME       ENDPOINTS                          AGE
database   198.19.21.69:2379                  25m
summary    198.19.21.1:80,198.19.22.131:80    25m
customer   198.19.22.132:80                   25m
```

Now list the pods:

```
kubectl get pods -n yaobank -o wide
```

Example output:

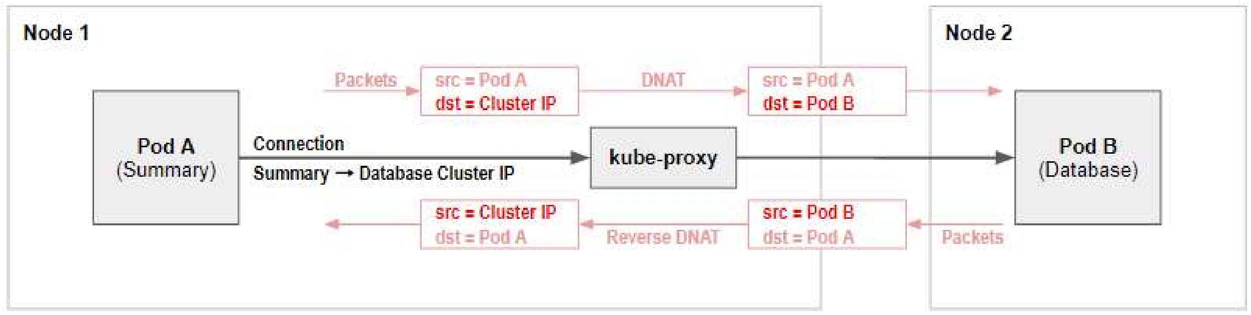
```
NAME                            READY   STATUS    RESTARTS   AGE   IP
NODE      NOMINATED NODE   READINESS GATES
database-6c5db58d95-x9s7m   1/1     Running   0          25m   198.19.21.69
node2     <none>           <none>
summary-85c56b76d7-j2kc9    1/1     Running   0          25m   198.19.21.1
control   <none>           <none>
summary-85c56b76d7-smjfn    1/1     Running   0          25m
198.19.22.131   node1     <none>           <none>
customer-574bd6cc75-2blnv   1/1     Running   0          25m
198.19.22.132   node1     <none>           <none>
```

You can see that the IP addresses listed as the service endpoints in the previous step are the IP addresses of the pods backing each service, as expected. Each service is backed by one or more pods spread across the worker nodes in our cluster.

Kube-Proxy Cluster IP Implementation

In this cluster we are currently running kube-proxy in its default, and most commonly used, iptables mode (not IPVS mode).

To explore the iptables rules kube-proxy programs into the kernel to implement Cluster IP based services, let's look at the Database service. The Summary pods use this service to connect to the Database pods. Kube-proxy uses DNAT to map the Cluster IP to the chosen backing pod.



Get service endpoints

Find the service endpoints for summary ClusterIP service:

```
kubectl get endpoints -n yaobank summary
```

Example output:

```
NAME      ENDPOINTS                           AGE
summary   198.19.21.1:80,198.19.22.131:80    27m
```

The summary service has two endpoints (198.19.21.1 on port 80, and 198.19.22.131 on port 80, in this example output). Starting from the KUBE-SERVICES iptables chain, we will traverse each chain until you get to the rule directing traffic to these endpoint IP addresses.

SSH into a node

Let’s SSH into node1 so we can explore the iptables rules that kube-proxy has set up on that node. (It will have set up similar rules on every other node on the cluster.)

```
ssh node1
```

Examine the KUBE-SERVICE chain

To make it easier to manage large numbers of iptables rules, groups of iptables rules can be grouped together into iptables chains. Kube-proxy puts its top level rules into a KUBE-SERVICES chain.

Let’s take a look at that chain now:

```
sudo iptables -v --numeric --table nat --list KUBE-SERVICES
```

Example output:

```
Chain KUBE-SERVICES (2 references)
 pkts bytes target     prot opt in     out     source
destination
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.36.113        /* yaobank/summary:http cluster IP */ tcp dpt:80
    2   120 KUBE-SVC-OIQIZJVJK6E34BR4  tcp  --  *      *       0.0.0.0/0
198.19.36.113        /* yaobank/summary:http cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  udp  --  *      *      !198.19.16.0/20
198.19.32.10         /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
    2   158 KUBE-SVC-TCOU7JCQXEZGVUNU  udp  --  *      *       0.0.0.0/0
198.19.32.10         /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
    7   420 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.32.1          /* default/kubernetes:https cluster IP */ tcp dpt:443
    7   420 KUBE-SVC-NPX46M4PTMTKRN6Y  tcp  --  *      *       0.0.0.0/0
198.19.32.1          /* default/kubernetes:https cluster IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.46.45         /* calico-system/calico-typha:calico-typha cluster IP
*/ tcp dpt:5473
    0     0 KUBE-SVC-RK657RLKDNVNU64O  tcp  --  *      *       0.0.0.0/0
198.19.46.45         /* calico-system/calico-typha:calico-typha cluster IP
*/ tcp dpt:5473
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.46.209        /* kube-system/traefik:https cluster IP */ tcp dpt:443
    0     0 KUBE-SVC-IKNZCF5XJQBTG3KZ  tcp  --  *      *       0.0.0.0/0
198.19.46.209        /* kube-system/traefik:https cluster IP */ tcp dpt:443
    0     0 KUBE-FW-IKNZCF5XJQBTG3KZ  tcp  --  *      *       0.0.0.0/0
198.19.0.3           /* kube-system/traefik:https loadbalancer IP */ tcp
dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.33.51         /* yaobank/database:http cluster IP */ tcp dpt:2379
    0     0 KUBE-SVC-AE2X4VPDA5SRYCA6  tcp  --  *      *       0.0.0.0/0
198.19.33.51         /* yaobank/database:http cluster IP */ tcp dpt:2379
```

```
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.45.149        /* yaobank/customer:http cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-PX5FENG4GZJTCELT  tcp  --  *      *       0.0.0.0/0
198.19.45.149        /* yaobank/customer:http cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.32.10         /* kube-system/kube-dns:dns-tcp cluster IP */ tcp
dpt:53
    0     0 KUBE-SVC-ERIFXISQEP7F7OF4  tcp  --  *      *       0.0.0.0/0
198.19.32.10         /* kube-system/kube-dns:dns-tcp cluster IP */ tcp
dpt:53
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.32.10         /* kube-system/kube-dns:metrics cluster IP */ tcp
dpt:9153
    0     0 KUBE-SVC-JD5MR3NA4I4DYORP  tcp  --  *      *       0.0.0.0/0
198.19.32.10         /* kube-system/kube-dns:metrics cluster IP */ tcp
dpt:9153
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.38.163        /* kube-system/metrics-server: cluster IP */ tcp
dpt:443
    0     0 KUBE-SVC-LC5QY66VUV2HJ6WZ  tcp  --  *      *       0.0.0.0/0
198.19.38.163        /* kube-system/metrics-server: cluster IP */ tcp
dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.38.41         /* kube-system/traefik-prometheus:metrics cluster IP
*/ tcp dpt:9100
    0     0 KUBE-SVC-W3ST5H65YH2QID6S  tcp  --  *      *       0.0.0.0/0
198.19.38.41         /* kube-system/traefik-prometheus:metrics cluster IP
*/ tcp dpt:9100
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.46.209        /* kube-system/traefik:http cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-X3WUOHPTYIG7AA3Q  tcp  --  *      *       0.0.0.0/0
198.19.46.209        /* kube-system/traefik:http cluster IP */ tcp dpt:80
    0     0 KUBE-FW-X3WUOHPTYIG7AA3Q  tcp  --  *      *       0.0.0.0/0
198.19.0.3           /* kube-system/traefik:http loadbalancer IP */ tcp
dpt:80
  931 55938 KUBE-NODEPORTS  all  --  *      *       0.0.0.0/0
0.0.0.0/0            /* kubernetes service nodeports; NOTE: this must be
the last rule in this chain */ ADDRTYPE match dst-type LOCAL
```

Each iptables chain consists of a list of rules that are executed in order until a rule matches. The key columns/elements to note in this output are:

- target - which chain iptables will jump to if the rule matches
- prot - the protocol match criteria
- source, and destination - the source and destination IP address match criteria
- the comments that kube-proxy includes
- the additional match criteria at the end of each rule - e.g dpt:80 that specifies the destination port match

You can see this chain includes rules to jump to service specific chains, one for each service.

KUBE-SERVICES -> KUBE-SVC-XXXXXXXXXXXXXXXX

Let's look more closely at the rules for the summary service.

```
sudo iptables -v --numeric --table nat --list KUBE-SERVICES | grep -E
summary
```

```
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !198.19.16.0/20
198.19.36.113        /* yaobank/summary:http cluster IP */ tcp dpt:80
    2   120 KUBE-SVC-OIQIZJVJK6E34BR4  tcp  --  *      *       0.0.0.0/0
198.19.36.113        /* yaobank/summary:http cluster IP */ tcp dpt:80
```

The second rule directs traffic destined for the summary service clusterIP (198.19.36.113 in the example output) to the chain that load balances the service (KUBE-SVC-XXXXXXXXXXXXXXXX).

KUBE-SVC-XXXXXXXXXXXXXXXX -> KUBE-SEP-XXXXXXXXXXXXXXXX

kube-proxy in iptables mode uses a randomized equal cost selection algorithm to load balance traffic between pods. We currently have two summary pods, so it should have rules in place that load balance equally across both pods.

Let's examine how this load balancing works using the chain name returned from our previous command. (Remember your chain name may be different than this example.)

```
sudo iptables -v --numeric --table nat --list KUBE-SVC-OIQIZJVJK6E34BR4
```

Example output:

```
Chain KUBE-SVC-OIQIZJVJK6E34BR4 (1 references)
 pkts bytes target     prot opt in     out     source
destination
    2   120 KUBE-SEP-Q6MJJR7VDMWJNZBE  all  --  *      *       0.0.0.0/0
0.0.0.0/0            /* yaobank/summary:http */ statistic mode random
probability 0.500000
```

```
00000
    0     0 KUBE-SEP-MR2OHPODPKVEKHD4  all  --  *      *       0.0.0.0/0
0.0.0.0/0            /* yaobank/summary:http */
```

Notice that kube-proxy is using the iptables statistic module to set the probability for a packet to be randomly matched.

The first rule directs traffic destined for the summary service to a chain that delivers packets to the first service endpoint (KUBE-SEP-Q6MJJR7VDMWJNZBE) with a probability of 0.50000000000. The second rule unconditionally directs to the second service endpoint chain (KUBE-SEP-MR2OHPODPKVEKHD4). The result is that traffic is load balanced across the service endpoints equally (on average).

If there were 3 service endpoints then the first chain matches would be probability 0.33333333, the second probability 0.5, and the last unconditional. The result of this is that each service endpoint receives a third of the traffic (on average).

And so on for any number of services!

KUBE-SEP-XXXXXXXXXXXXXXXX -> summary pod

Let's look at one of the service endpoint chains. (Remember your chain names may be different than this example.)

```
sudo iptables -v --numeric --table nat --list KUBE-SEP-MR2OHPODPKVEKHD4
```

Example output:

```
Chain KUBE-SEP-MR2OHPODPKVEKHD4 (1 references)
 pkts bytes target     prot opt in     out     source
destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       198.19.22.131
0.0.0.0/0            /* yaobank/summary:http */
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0
0.0.0.0/0            /* yaobank/summary:http */ tcp to:198.19.22.131:80
```

The second rule performs the DNAT that changes the destination IP from the service's clusterIP to the IP address of the service endpoint backing pod (198.19.22.131 in this example). After this, standard Linux routing can handle forwarding the packet like it would any other packet.

Recap

You've just traced the kube-proxy iptables rules used to load balance traffic to summary pods exposed as a service of type ClusterIP.

In summary, for a packet being sent to a clusterIP:

- The KUBE-SERVICES chain matches on the clusterIP and jumps to the corresponding KUBE-SVC-XXXXXXXXXXXXXXXX chain.
- The KUBE-SVC-XXXXXXXXXXXXXXXX chain load balances the packet to a random service endpoint KUBE-SEP-XXXXXXXXXXXXXXXX chain.
- The KUBE-SEP-XXXXXXXXXXXXXXXX chain DNATs the packet so it will get routed to the service endpoint (backing pod).

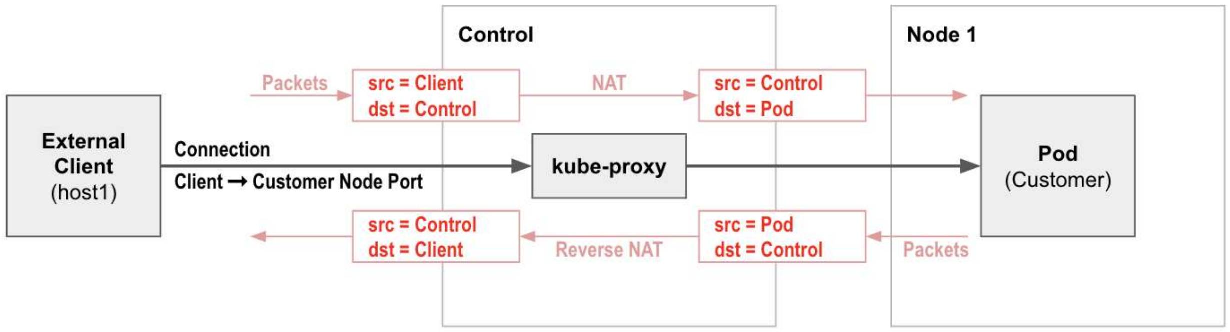
Finally, let's return to host1 to take a look at NodePorts in the next section by using the exit command.

```
exit
```

Kube-Proxy NodePort Implementation

Let's explore the iptables rules that implement the customer service with a node port.

To explore the iptables rules kube-proxy programs into the kernel to implement Node Port based services, let's look at the Customer service. External clients use this service to connect to the Customer pods. Kube-proxy uses NAT to map the Node Port to the chosen backing pod, and the source IP to the node IP of the ingress node, so that it can reverse the NAT for return packets. (If it didn't change the source IP then return packets would go directly back to the client, without the node that did the NAT having a chance to reverse the NAT, and as a result the client would not recognize the packets as being part of the connection it made to the Node Port).



Get service endpoints

Find the service endpoints for customer NodePort service.

```
kubectl get endpoints -n yaobank customer
```

Example output:

```
NAME        ENDPOINTS           AGE
customer    198.19.22.132:80    39m
```

The customer service has one endpoint (198.19.22.132 on port 80 in this example output). Starting from the KUBE-SERVICES iptables chain, we will traverse each chain until you get to the rule directing traffic to this endpoint IP address.

KUBE-SERVICES -> KUBE-NODEPORTS

Lets re-enter the node1 instance to look at iptables once again with this new information.

```
ssh node1
```

The KUBE-SERVICE chain handles the matching for service types ClusterIP and LoadBalancer. At the end of KUBE-SERVICE chain, another custom chain KUBE-NODEPORTS will handle traffic for service type NodePort.

```
sudo iptables -v --numeric --table nat --list KUBE-SERVICES | grep
KUBE-NODEPORTS
```

Example output:

```
 1216 73038 KUBE-NODEPORTS  all  --  *      *       0.0.0.0/0
0.0.0.0/0            /* kubernetes service nodeports; NOTE: this must be
the last rule in this chain */ ADDRTYPE match dst-type LOCAL
```

“match dst-type LOCAL” matches any packet with a local host IP as the destination. I.e. any address that is assigned to one of the host's interfaces.

KUBE-NODEPORTS -> KUBE-SVC-XXXXXXXXXXXXXXXX

```
sudo iptables -v --numeric --table nat --list KUBE-NODEPORTS
```

Example output:

```
Chain KUBE-NODEPORTS (1 references)
 pkts bytes target     prot opt in     out     source
destination
    0     0 KUBE-MARK-MASQ  tcp  --  *      *       0.0.0.0/0
0.0.0.0/0            /* kube-system/traefik:https */ tcp dpt:31397
    0     0 KUBE-SVC-IKNZCF5XJQBTG3KZ  tcp  --  *      *       0.0.0.0/0
0.0.0.0/0            /* kube-system/traefik:https */ tcp dpt:31397
    0     0 KUBE-MARK-MASQ  tcp  --  *      *       0.0.0.0/0
0.0.0.0/0            /* yaobank/customer:http */ tcp dpt:30180
    0     0 KUBE-SVC-PX5FENG4GZJTCELT  tcp  --  *      *       0.0.0.0/0
```

```
0.0.0.0/0            /* yaobank/customer:http */ tcp dpt:30180
    0     0 KUBE-MARK-MASQ  tcp  --  *      *       0.0.0.0/0
0.0.0.0/0            /* kube-system/traefik:http */ tcp dpt:32196
    0     0 KUBE-SVC-X3WUOHPTYIG7AA3Q  tcp  --  *      *       0.0.0.0/0
0.0.0.0/0            /* kube-system/traefik:http */ tcp dpt:32196
```

The second rule directs traffic destined for the customer service to the chain that load balances the service (KUBE-SVC-PX5FENG4GZJTCELT). tcp dpt:30180 matches any packet with the destination port of tcp 30180 (the node port of the customer service).

KUBE-SVC-XXXXXXXXXXXXXXXX -> KUBE-SEP-XXXXXXXXXXXXXXXX

(Remember your chain name may be different than this example.)

```
sudo iptables -v --numeric --table nat --list KUBE-SVC-PX5FENG4GZJTCELT
```

Example output:

```
Chain KUBE-SVC-PX5FENG4GZJTCELT (2 references)
 pkts bytes target     prot opt in     out     source
destination
    0     0 KUBE-SEP-UBXKSM3V2OSEF4IL  all  --  *      *       0.0.0.0/0
0.0.0.0/0            /* yaobank/customer:http */
```

As we only have a single backing pod for the customer service, there is no load balancing to do, so there is a single rule that directs all traffic to the chain that delivers the packet to the service endpoint (KUBE-SEP-UBXKSM3V2OSEF4IL).

KUBE-SEP-XXXXXXXXXXXXXXXX -> customer endpoint

(Remember your chain name may be different than this example.)

```
sudo iptables -v --numeric --table nat --list KUBE-SEP-UBXKSM3V2OSEF4IL
```

Example output:

```
Chain KUBE-SEP-UBXKSM3V2OSEF4IL (1 references)
 pkts bytes target     prot opt in     out     source
destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       198.19.22.132
0.0.0.0/0            /* yaobank/customer:http */
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0
0.0.0.0/0            /* yaobank/customer:http */ tcp to:198.19.22.132:80
```

This rule delivers the packet to the customer service endpoint.

The second rule performs the DNAT that changes the destination IP from the service's clusterIP to the IP address of the service endpoint backing pod (198.19.22.132 in this example). After this, standard Linux routing can handle forwarding the packet like it would any other packet.

Recap

You've just traced the kube-proxy iptables rules used to load balance traffic to customer pods exposed as a service of type NodePort.

In summary, for a packet being sent to a NodePort:

The end of the KUBE-SERVICES chain jumps to the KUBE-NODEPORTS chain

- The KUBE-NODEPORTS chaing matches on the NodePort and jumps to the corresponding KUBE-SVC-XXXXXXXXXXXXXXXX chain.
- The KUBE-SVC-XXXXXXXXXXXXXXXX chain load balances the packet to a random service endpoint KUBE-SEP-XXXXXXXXXXXXXXXX chain.
- The KUBE-SEP-XXXXXXXXXXXXXXXX chain DNATs the packet so it will get routed to the service endpoint (backing pod).

Exit back to host1

That’s it! We're done with this module. Exit the ssh session to node1 and return to host1.

```
exit
```