

Kubernetes NodePort and iptables rules

Jul 8, 2020 · 10 min read · [3 Comments](#) ·  [Kubernetes](#)

I have been working on kubernetes over the last few months and having fun learning about the underlying systems. When I started using [kubernetes services](#), I wanted to learn about the iptables rules that kube-proxy creates to enable them, however, I didn't exactly know where to start. While there are some really good posts explaining [kubernetes networking](#) and how the concept of [services works](#), I couldn't easily find a starting point to explore these iptables except reading the code or trying to make sense of the output of `iptables-save` command (which I can't).

In this post, I share some of what I have learned by digging a little deeper into the iptables rules for NodePort type services and share answers to the following questions I came across while working on kube-proxy:

- What happens when a non-kubernetes process starts using a port that's allocated as a NodePort to a service?
- Does the service endpoint continue to route traffic to pods if kube-proxy (configured in iptables mode) process dies on the node?

Some background

Kubernetes allows a [few ways to expose applications](#) to the world outside the kubernetes cluster through the concept of [services](#). In our setup at work, we expose a few of our apps through [NodePort](#) type services. Also, [kube-proxy](#) is the kubernetes component that powers the [services concept](#) and we run it in iptables mode (default).

A little about NodePort

[NodePort](#) is one of the ways of exposing [Kubernetes Services](#) to the world outside the kubernetes cluster. As per Kubernetes [docs](#),

If you set the type field to NodePort, the Kubernetes control plane allocates a port from a range specified by `--service-node-port-range` flag (default: 30000-32767). Each node proxies that port (the same port number on every Node) into your Service.

This means that if we have a `service` with NodePort `30000`, a request to `<kubernetes-node-ip>:30000` will get routed to our app. Under normal circumstances kube-proxy [binds and listens](#) on all NodePorts to ensure these ports stay reserved and no other processes can use them.

err...a non-kubernetes is using the NodePort!!

A few weeks back at work, one of our kubernetes nodes was rebooted and as it was added back to the cluster, we saw errors in kube-proxy logs - `bind: address already in use`:

```
I0707 20:57:38.648179    1 proxier.go:701] Syncing iptables rules
E0707 20:57:38.679876    1 proxier.go:1072] can't open "nodePort for default/azure-vote-front:" (:30450/tcp), skipping
```

This error meant that a port allocated as a NodePort to a `service` was already in use by another process. We found out that there was a non-kubernetes process using this NodePort as a client port to connect to a remote server. This happened due to a race condition post node reboot. This other process started using the NodePort before kube-proxy could bind and listen on the port to reserve it.

Will the NodePort on this node still route traffic to the pods?

This was the immediate question that popped up in our heads as the NodePort allocated to a `service` was now being used by a non-kubernetes process. One of my colleagues, [Alex Dudko](#) pointed out that kube-proxy creates iptables rules in the **PREROUTING** chain in **nat** table.

Because of these rules, the answer to the above question is - **Yes!** Traffic sent to the NodePort on this node will still correctly be routed to the backend pods it targets.

The NodePort would also continue to work if another process was listening on the port as a server and not just using it as a client. Though, would our `HTTP GET` requests reach this server? More on this later.

Kube-proxy iptables rules

In iptables mode, kube-proxy creates iptables rules for kubernetes services which ensure that the request to the `service` gets routed (and load balanced) to the appropriate pods.

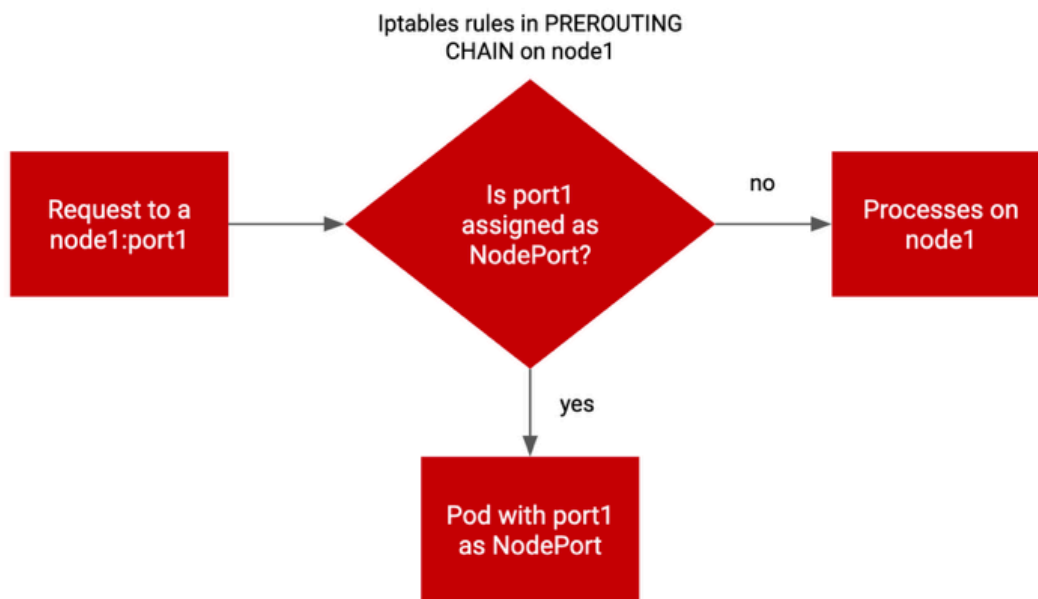
These iptables rules also help answer the second question mentioned above. As long as these iptables rules exist, requests to `services` will get routed to the appropriate pods even if kube-proxy process dies on the node. Endpoints for new `services` won't work from this node, however, since kube-proxy process won't create the iptables rules for it.

Rules in the PREROUTING chain

As outlined in this [flow chart](#), there are rules in the **PREROUTING** chain of the **nat** table for kubernetes services. As per [iptables rules evaluation order](#), rules in the **PREROUTING** chain are the first ones to be consulted as a packet enters the linux kernel's networking stack.

Rules created by kube-proxy in the **PREROUTING** chain help determine if the packet is meant for a local socket on the node or if it should be forwarded to a pod. It is these rules that ensure that the request to `<kubernetes-node-ip>:<NodePort>` continue to get routed to pods even if the NodePort is in use by another process.

Below is an over-simplified diagram that demonstrates this:



In rest of this post, I walk through a setup that reproduces the above scenario and examine the iptables rules that make this work.

Setting up a Kubernetes cluster

I created a kubernetes cluster using [Azure Kubernetes Service](#).

Creating pods and a NodePort Service

From the [AKS docs](#), I also created a [deployment for an app with a frontend and a backend](#). I updated the `azure-vote-front` to **type: NodePort** service.

- **azure-vote-front** pod in the cluster

```

$ kubectl get po -l app=azure-vote-front --show-labels -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE
azure-vote-front-5bc759676c-x9bwg   1/1     Running   0           24d   10.244.0.11   aks-nodepool1-22391620-vmss000000
  
```

- **azure-vote-front** service targeting the above pod

```

$ kubectl get svc azure-vote-front -o wide
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE   SELECTOR
azure-vote-front  NodePort    10.0.237.3   <none>        80:30450/TCP     24d   app=azure-vote-front
  
```

The above output shows that the port **30450** is assigned as the NodePort to our service. And we can send HTTP requests to the app from host network:

```
$ curl -IX GET localhost:30450
HTTP/1.1 200 OK
Server: nginx/1.13.7
Date: Wed, 08 Jul 2020 00:02:08 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 950
Connection: keep-alive

$ curl -sSL localhost:30450 | grep title
<title>Azure Voting App</title>
```

Examining iptables rules for a NodePort service

Note: AKS doesn't allow ssh'ing into an AKS worker node directly, instead, a pod needs to be spun up to ssh into the worker node.

Instructions [here](#).

Let's look at the iptables rules for the **azure-vote-front** service.

After ssh'ing into the kubernetes worker node, let's look at the **PREROUTING** chain in the **nat** table. (**PREROUTING** chain exists in **raw**, **nat** and **mangle** tables, however, kube-proxy only creates **PREROUTING** chain rules in **nat** table)

```
$ sudo iptables -t nat -L PREROUTING | column -t
Chain          PREROUTING  (policy ACCEPT)
target         prot      opt    source      destination
KUBE-SERVICES  all       --     anywhere    anywhere    /*      kubernetes  service  portals */
DOCKER        all       --     anywhere    anywhere    ADDRTYPE match  dst-type LOCAL
```

There's a **KUBE-SERVICES** chain in the target that's created by kube-proxy. Let's list the rules in that chain.

```
$ sudo iptables -t nat -L KUBE-SERVICES -n | column -t
Chain          KUBE-SERVICES  (2 references)
target         prot      opt    source      destination
KUBE-MARK-MASQ tcp       --     !10.244.0.0/16  10.0.202.188 /* kube-system/healthmodel-replicaset-serv
KUBE-SVC-KA44REDDIFNMY4W2 tcp       --     0.0.0.0/0      10.0.202.188 /* kube-system/healthmodel-replicaset-serv
KUBE-MARK-MASQ udp       --     !10.244.0.0/16  10.0.0.10 /* kube-system/kube-dns:dns
KUBE-SVC-TC0U7JCQXEZGVUNU udp       --     0.0.0.0/0      10.0.0.10 /* kube-system/kube-dns:dns
KUBE-MARK-MASQ tcp       --     !10.244.0.0/16  10.0.0.10 /* kube-system/kube-dns:dns-tcp
KUBE-SVC-ERIFXISQEP7F70F4 tcp       --     0.0.0.0/0      10.0.0.10 /* kube-system/kube-dns:dns-tcp
KUBE-MARK-MASQ tcp       --     !10.244.0.0/16  10.0.190.160 /* kube-system/kubernetes-dashboard:
KUBE-SVC-XGLOHA7QRQ3V22RZ tcp       --     0.0.0.0/0      10.0.190.160 /* kube-system/kubernetes-dashboard:
KUBE-MARK-MASQ tcp       --     !10.244.0.0/16  10.0.195.6 /* kube-system/metrics-server:
KUBE-SVC-LC5QY66VUV2HJ6WZ tcp       --     0.0.0.0/0      10.0.195.6 /* kube-system/metrics-server:
KUBE-MARK-MASQ tcp       --     !10.244.0.0/16  10.0.129.119 /* default/azure-vote-back:
KUBE-SVC-JQ4VBJ2YW022DDZW tcp       --     0.0.0.0/0      10.0.129.119 /* default/azure-vote-back:
KUBE-MARK-MASQ tcp       --     !10.244.0.0/16  10.0.237.3 /* default/azure-vote-front:
KUBE-SVC-GHSLGKVXVBRM4GZX tcp       --     0.0.0.0/0      10.0.237.3 /* default/azure-vote-front:
KUBE-MARK-MASQ tcp       --     !10.244.0.0/16  10.0.0.1 /* default/kubernetes:https
KUBE-SVC-NPX46M4PTMTKR6Y tcp       --     0.0.0.0/0      10.0.0.1 /* default/kubernetes:https
KUBE-NODEPORTS all       --     0.0.0.0/0      0.0.0.0/0 /* kubernetes
```

The last target in the **KUBE-SERVICES** chain is the **KUBE-NODEPORTS** chain. Since the service we created is of type **NodePort**, let's list the rules in **KUBE-NODEPORTS** chain.

```
$ sudo iptables -t nat -L KUBE-NODEPORTS -n | column -t
Chain          KUBE-NODEPORTS  (1 references)
target         prot      opt    source      destination
KUBE-MARK-MASQ tcp       --     0.0.0.0/0      0.0.0.0/0 /* default/azure-vote-front: */ tcp dpt:30450
KUBE-SVC-GHSLGKVXVBRM4GZX tcp       --     0.0.0.0/0      0.0.0.0/0 /* default/azure-vote-front: */ tcp dpt:30450
```

We can see that the above targets are for packets destined to our NodePort **30450**. The comments also show the namespace/pod name - **default/azure-vote-front**. For requests originating from outside the cluster and destined to our app running as a pod, the **KUBE-MARK-MASQ** rule marks the packet to be altered later in the **POSTROUTING** chain to use SNAT (source network address translation) to rewrite the source IP as the node IP (so that other hosts outside the pod network can reply back).

Since **KUBE-MARK-MASQ** target is to **MASQUERADE** packets later, let's follow the **KUBE-SVC-GHSLGKVXVBRM4GZX** chain to further examine our service.

```
$ sudo iptables -t nat -L KUBE-SVC-GHSLGKVVXVBRM4GZX -n | column -t
Chain          KUBE-SVC-GHSLGKVVXVBRM4GZX  (2 references)
target         prot          opt    source        destination
KUBE-SEP-QXDNOBCCLOXLV7LV  all          --    0.0.0.0/0      0.0.0.0/0

$ sudo iptables -t nat -L KUBE-SEP-QXDNOBCCLOXLV7LV -n | column -t
Chain          KUBE-SEP-QXDNOBCCLOXLV7LV  (1 references)
target         prot          opt    source        destination
KUBE-MARK-MASQ  all          --    10.244.0.11    0.0.0.0/0
DNAT           tcp          --    0.0.0.0/0      0.0.0.0/0      tcp to:10.244.0.11:80
```

Here we see the **DNAT (Destination Network Address Translation)** target is used to rewrite the destination of the packet destined to port **30450** to our pod **10.244.0.11:80**. We can verify the **podIP** and **containerPort** of our pod as follows:

```
$ kubectl get po azure-vote-front-5bc759676c-x9bwg -o json | jq -r '[.status.podIP, .spec.containers[0].ports[0].containerPort]'
["10.244.0.11", 80]
```

Verify kube-proxy is listening on NodePort

Under normal circumstances kube-proxy binds and listens on all NodePorts to ensure these ports stay reserved and no other processes can use them. We can verify this on the above kubernetes node:

```
$ sudo lsof -i:30450
COMMAND  PID USER  FD  TYPE  DEVICE SIZE/OFF NODE NAME
hyperkube 11558 root   9u  IPv6  95841832      0t0  TCP *:30450 (LISTEN)

$ ps -aef | grep -v grep | grep 11558
root      11558 11539  0 Jul02 ?        00:06:37 /hyperkube kube-proxy --kubeconfig=/var/lib/kubelet/kubeconfig --clust
```

kube-proxy is listening on NodePort **30450**.

Create a non-kubernetes process use the NodePort

Now let's kill kube-proxy process and start a server that listens on this NodePort instead.

```
$ kill -9 11558
$ python -m SimpleHTTPServer 30450 &
[1] 123578

$ sudo lsof -i:30450
COMMAND  PID  USER   FD  TYPE  DEVICE SIZE/OFF NODE NAME
python   123578 azureuser 3u  IPv4  95854834      0t0  TCP *:30450 (LISTEN)

$ ps -aef | grep -v grep | grep 123578
azureus+ 123578  85107  0 00:49 pts/5    00:00:00 python -m SimpleHTTPServer 30450
```

Where will the requests to NodePort be routed - pod or the non-kubernetes process?

While the python process is listening on port **30450**, also allocated as a **NodePort** to our kubernetes service, the iptables rules in the **PREROUTING** chain will route all requests to port **30450** to our pod. We can verify this as below:

```
$ curl -I -XGET localhost:30450
HTTP/1.1 200 OK
Server: nginx/1.13.7
Date: Wed, 08 Jul 2020 00:53:09 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 950
Connection: keep-alive
```

As we send a **GET** request to port **30450**, we receive a response from an nginx server hosting **Azure Voting App**, like we saw when [kube-proxy was listening on port 30450](#).

```
$ curl -sSL localhost:30450 | grep title
<title>Azure Voting App</title>
```

Summary

- When kube-proxy is used in iptables mode, routing requests to **services** continues to work for existing services even when the kube-proxy process dies on the node
- Kube-proxy binds and listens (on all k8s nodes) to all ports allocated as NodePorts to ensure these ports stay reserved and no other processes can use them

- Even if a process starts using NodePort, iptables rules (because they are in **PREROUTING** chain) ensure that the traffic sent to the `NodePort` gets routed to the pods

References

- [@thockin's](#) tweet, [describing kube-proxy iptables rules](#) in a [flow chart](#), is an excellent way to holistically follow a packet's path through various iptables rules before it reaches the destined pod.
- [Kubernetes Networking Demystified: A Brief Guide](#) is a great post explaining how kubernetes services and kubernetes networking work.

[kubernetes](#)[iptables](#)[networking](#)

Would you like to receive an email when I publish a new blog post?

NEXT

[How a Kubernetes Pod Gets an IP Address](#)

Related

- [How a Kubernetes Pod Gets an IP Address](#)
- [Understanding the Kubernetes Scale subresource](#)

3 Comments

Login

G


Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

Share

BestNewestOldest




Marc

2 years ago

When my master goes down in my 2-node master+worker cluster requests to my worker fail 50% of the time. The worker is perfectly healthy but something is blindly sending traffic round-robin to a pod on the master node which no longer exists. Is this because of the IP tables? Is there a way for the worker to actively check the pods it's balancing?

0Reply



sosogh

3 years ago

Thank you for your article about why the kube-proxy process listening on nodeport.

0Reply

R

Rajat bajaj

4 years ago

Awesome. More details on kube-proxy iptables rules are here <https://letslearn24x7.blogs...>

0Reply