White paper
Cisco public

The bridge to possible

A Comprehensive Guide to Kubernetes Networking with the Intersight Kubernetes Service

Contents

Introduction

Few open-source projects have been as widely and rapidly adopted as Kubernetes (K8s), the de facto container orchestration platform. With Kubernetes, development teams can deploy, manage, and scale their containerized applications with ease, making innovations more accessible to their continuous delivery pipelines.

Cisco Intersight™ Kubernetes Service (IKS) is a fully curated, lightweight container management platform for delivering multicloud production-grade upstream Kubernetes. It simplifies the process of provisioning, securing, scaling, and managing virtualized Kubernetes clusters by providing end-to-end automation, including the integration of networking, load balancers, native dashboards, and storage provider interfaces.

Addresses and components

IKS builds a Kubernetes cluster using 100% upstream native K8s images and therefore provides the same networking as a K8s cluster built from the ground up. This document will cover the following concepts:

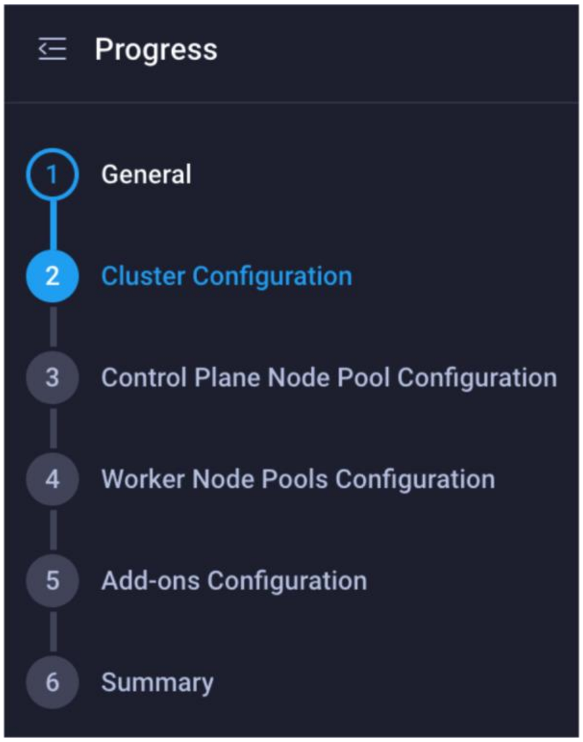
- Control plane and worker node addresses
- Pod addresses
- Cluster IP service addresses
- Load-balancer server addresses
- Kubernetes API server virtual IP address

To implement networking in each cluster, IKS uses the following components:

- CNI: Calico running in IPIP overlay mode
- L3/L4 load balancing: MetalLB running in Layer-2 mode (ARP)
- L7 ingress: NGINX

Explanation of the IKS cluster setup wizard

The IKS cluster setup wizard contains six steps. Configurations relating to cluster networking are found in steps 2, 3, and 4.



Progress

1 General

2 Cluster Configuration

3 Control Plane Node Pool Configuration

4 Worker Node Pools Configuration

5 Add-ons Configuration

6 Summary

Figure 1.
Steps in the IKS cluster creation wizard

If you have not yet built an IKS cluster in Cisco Intersight, please read through the IKS user guide below (this white paper is not designed to replace the user guide but to complement it with details on the networking configuration): https://intersight.com/help/saas/resources/intersight_kubernetes_service_user_guide

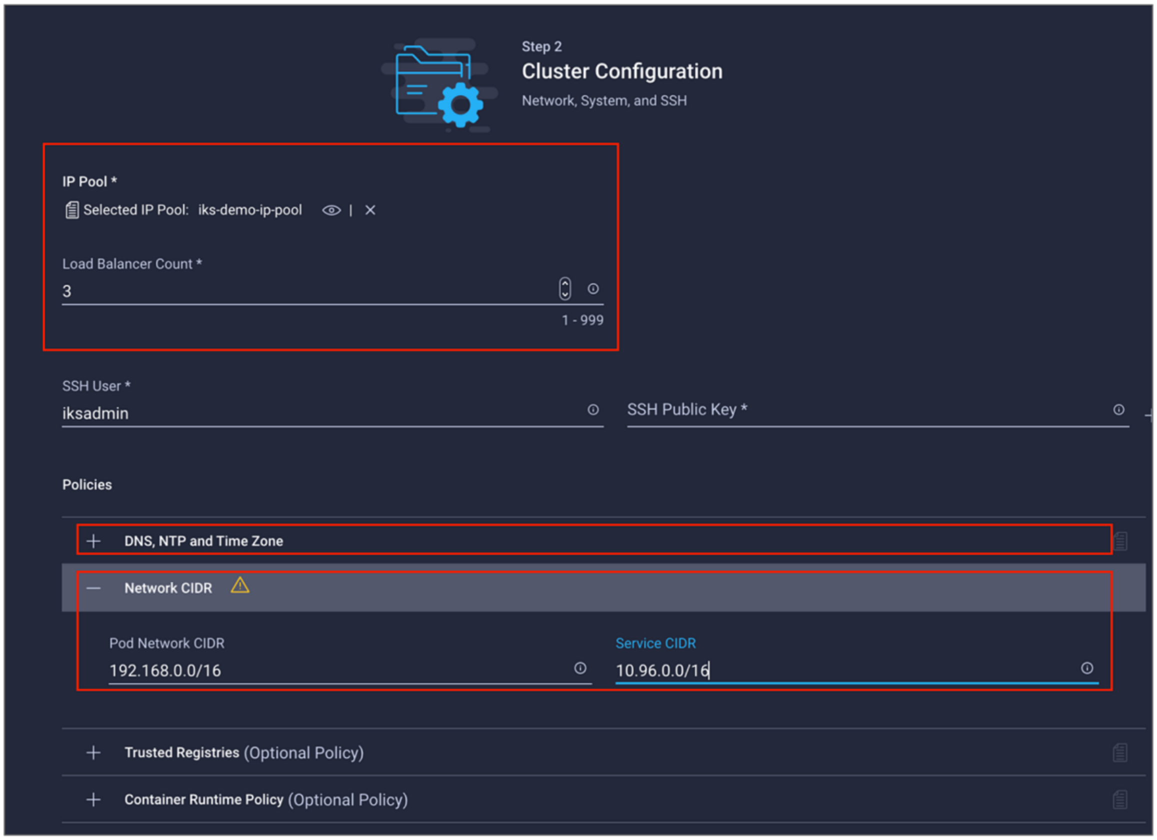
Step 1 – General

There is no network relation configuration on this page

Step 2 – Cluster Configuration

The cluster configuration page contains the following cluster-wide network related configurations:

- IP pool
- Load balancer count
- DNS server
- Pod network CIDR
- Service network CIDR



Step 2

Cluster Configuration

Network, System, and SSH

IP Pool *

Selected IP Pool: iks-demo-ip-pool

Load Balancer Count *

3

1 - 999

SSH User *

iksadmin

SSH Public Key *

Policies

+ DNS, NTP and Time Zone
— Network CIDR

Pod Network CIDR

192.168.0.0/16

Service CIDR

10.96.0.0/16

+ Trusted Registries (Optional Policy)
+ Container Runtime Policy (Optional Policy)

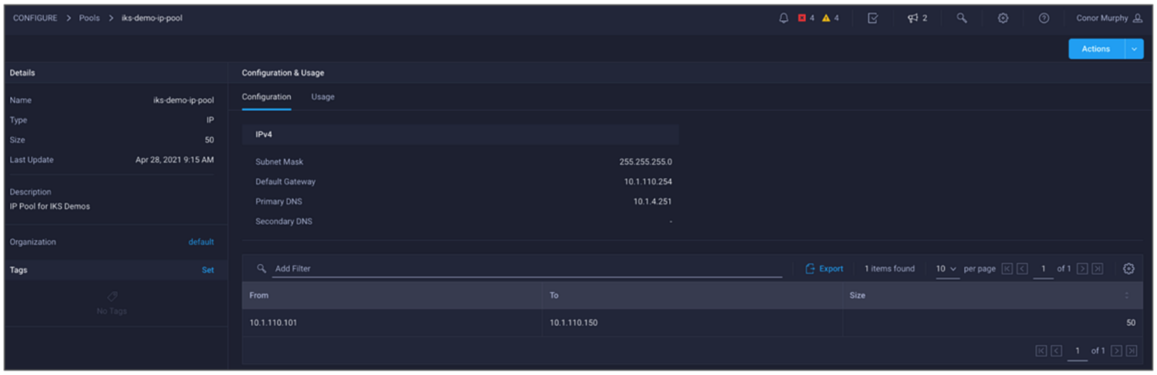
Figure 2.
The relevant fields relating to the IKS cluster-wide networking configuration

IP pool

In the context of IKS, an IP pool is used to allocate Kubernetes API Server Virtual IP, node IP addresses, and Kubernetes LoadBalancer Service (MetalLB) IP addresses.

In this step, the IP pool is only used for the LoadBalancer Service addresses. This subnet must be routable and not overlap with an existing subnet in your environment.

IP pools support IPv4 addresses.



Details

Name: iks-demo-ip-pool
Type: IP
Size: 50
Last Update: Apr 28, 2021 9:15 AM

Description
IP Pool for IKS Demos

Organization: default

Tags: Set

Configuration & Usage

IPv4

Subnet Mask: 255.255.255.0
Default Gateway: 10.1.110.254
Primary DNS: 10.1.4.251
Secondary DNS: -

Add Filter — 1 items found — 10 per page — 1 of 1

From	To	Size
10.1.110.101	10.1.110.150	50

Figure 3.
IKS IP pool. This example uses the same pool for the control plane, workers nodes, and LoadBalancer Service

Load balancer count

The load-balancer count field is used by the Kubernetes LoadBalancer Service. Use this field to specify how many IP addresses you wish to make available. These IP addresses are selected from your IP pool. When you create a new LoadBalancer service in your IKS cluster, MetalLB will assign an available IP address from the IP pool you have selected above. By default, IKS uses one IP address to provide external access to the cluster through the NGINX ingress.

DNS server

This field is used to configure the DNS server and suffix settings on the control plane and worker nodes. This will allow the nodes to perform DNS lookups. Kubernetes also provides DNS through the CoreDNS pods running on your cluster.

Pod network CIDR

Each control plane and worker node in your IKS cluster will receive its own internal subnet. This subnet is used to provide every pod an individual IP.

"By default, Calico uses an IPAM block size of 64 addresses – /26 for IPv4"

(Quoted from https://docs.projectcalico.org/networking/change-block-size)

The value that you enter in this field depends on the total number of nodes in the cluster (maximum size of control plane nodes + maximum size of worker nodes).

A minimum of 64 IP addresses must be available to each node for host assignment.

A valid size for a pod network is the maximum number of nodes in your cluster * 64. For example if you have if you have a cluster with one control node and three worker nodes, you would need a /24.

This field should not overlap with an existing subnet in your environment. Note that the pod network is non-routable outside of the cluster and can be reused for other clusters (as long as you don't build a service mesh over both clusters).

Another common error is the following: if you are running an application in a pod that is trying to access an external resource that is also in the same subnet as the pod network, you will have conflicts because the pod traffic is routed to other pods.

Service network CIDR

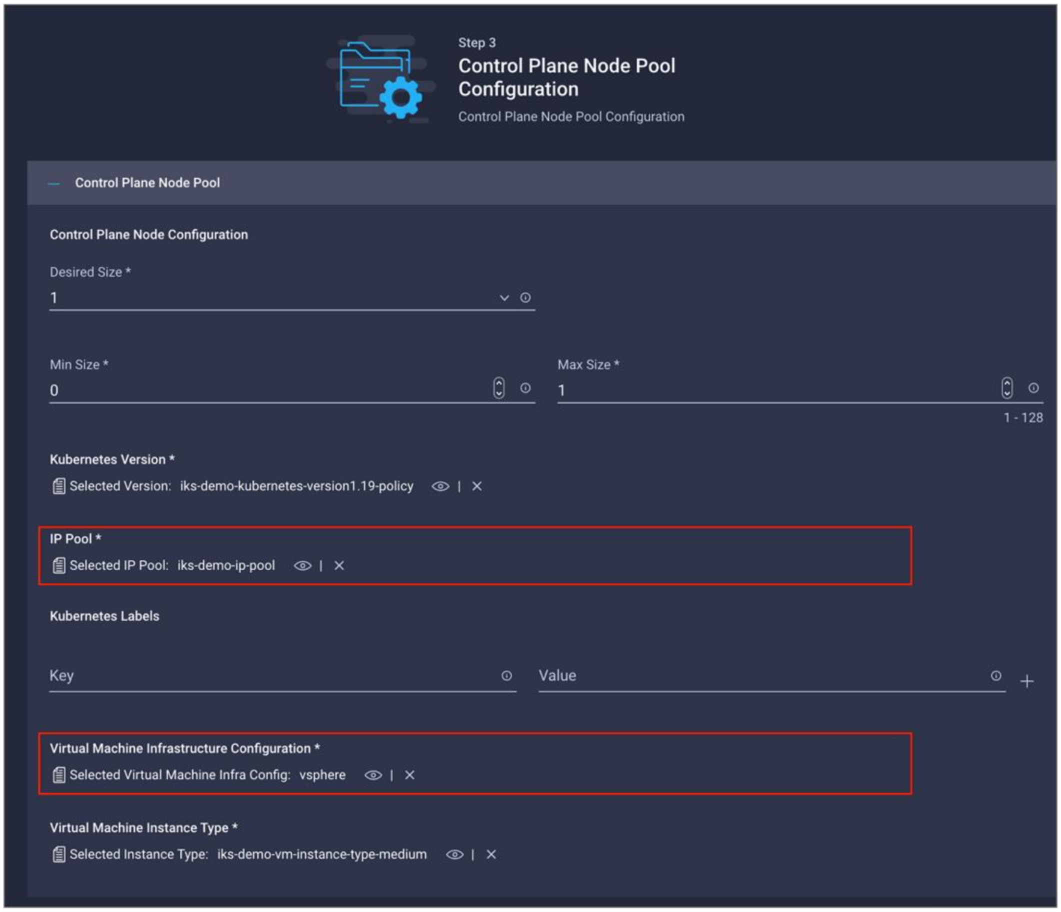
Every time a Kubernetes service (ClusterIP, NodePort, or LoadBalancer) is created, a cluster IP is assigned. This is an internal, non-routable address that is used to forward traffic in a cluster. For the service network CIDR, the validation only needs to have a minimum of /24. Any bits above /24 will be rejected. For example, /25 through /30 would fail the validation since service network CIDRs require at least 254 hosts.

A valid prefix for a service network CIDR can be between /24 (254 hosts) through /8 (16,777,214 hosts). A CIDR prefix /25 or greater will fail.

Step 3 – Control plane node pool configuration

The following configuration is networking related for the control plane nodes:

- IP pool
- Virtual machine infrastructure configuration



Step 3

Control Plane Node Pool Configuration

Control Plane Node Pool Configuration

— Control Plane Node Pool

Control Plane Node Configuration

Desired Size *
1

Min Size *
0

Max Size *
1
1 - 128

Kubernetes Version *
Selected Version: iks-demo-kubernetes-version1.19-policy

IP Pool *
Selected IP Pool: iks-demo-ip-pool

Kubernetes Labels

Key

Value

Virtual Machine Infrastructure Configuration *
Selected Virtual Machine Infra Config: vsphere

Virtual Machine Instance Type *
Selected Instance Type: iks-demo-vm-instance-type-medium

Figure 4.
The configuration of the IKS control plane nodes

IP pool

As noted before, in the context of IKS, an IP pool is used to allocate node IP addresses and Kubernetes LoadBalancer Service (MetalLB) IP addresses.

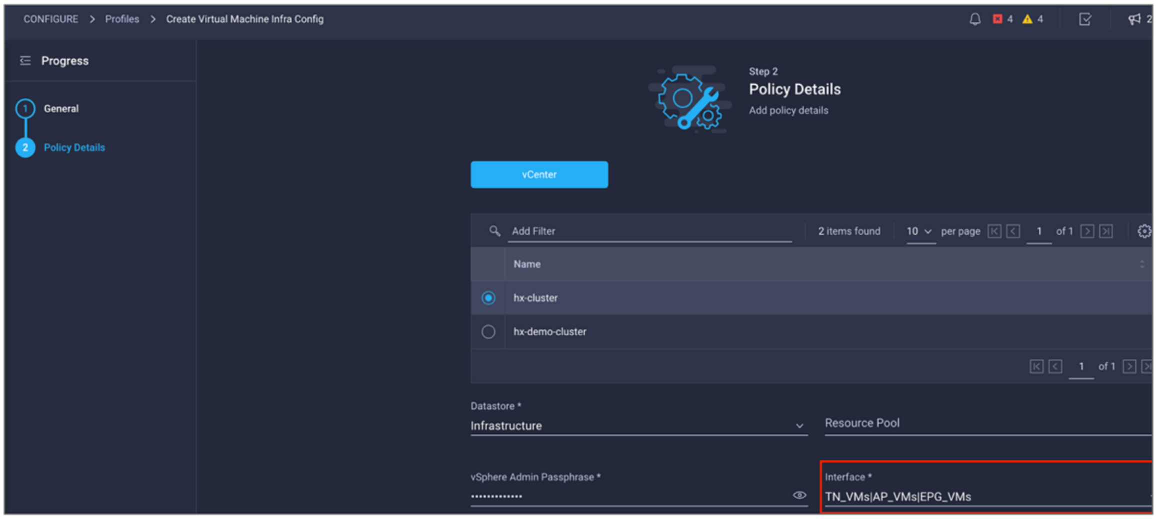
In this step, the IP pool is used to assign IP addresses to the control plane nodes. You may need to have your control plane nodes on a different subnet from your LoadBalancer IPs, hence the need to select the IP pool twice. If you have a single routable subnet, you can select the same IP pool for both the LoadBalancer Service and your node IPs.

This subnet must be routable and not overlap with an existing subnet in your environment.

IP pools support IPv4 addresses.

Virtual machine infrastructure configuration

Since IKS currently runs in a virtualized environment, you need to provide the configuration to use for the VM settings when they are deployed (for example, VMware vSphere). Specifically relating to networking is the VM adapter interface that the control plane node will use (for example, the vSphere port group).



Step 2
Policy Details
Add policy details

	Name
●	hx-cluster
○	hx-demo-cluster

Datastore * Infrastructure

Resource Pool

vSphere Admin Passphrase * ••••••••••••••

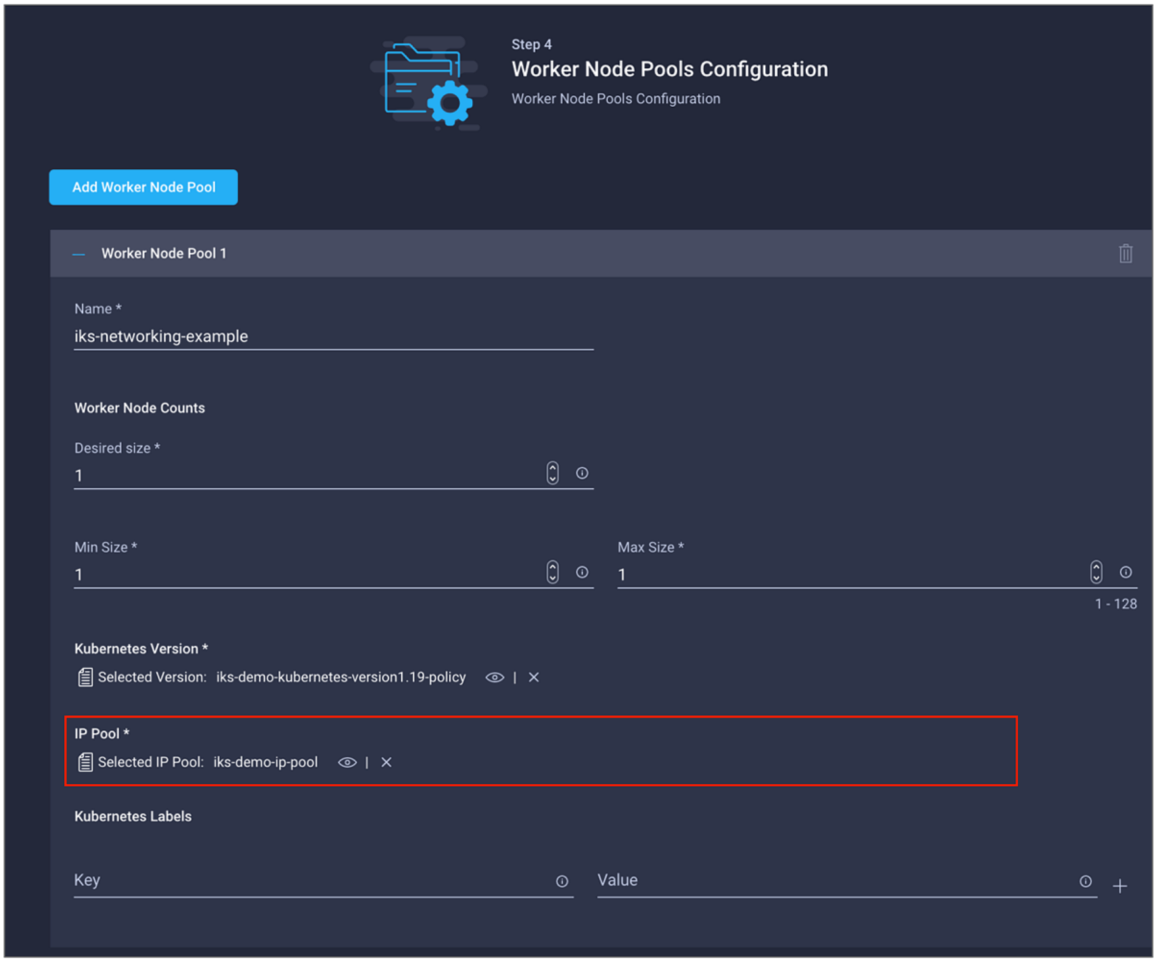
Interface * TN_VMs|AP_VMs|EPG_VMs

Figure 5.
The configuration screen for the virtual machine infrastructure configuration

Step 4 – Worker node pools configurations

The following configuration is networking-related for the worker nodes:

- IP pool



Step 4

Worker Node Pools Configuration

Worker Node Pools Configuration

Worker Node Pool 1

Name * iks-networking-example

Worker Node Counts

Desired size * 1

Min Size * 1

Max Size * 1

1 - 128

Kubernetes Version *

Selected Version: iks-demo-kubernetes-version1.19-policy

IP Pool *

Selected IP Pool: iks-demo-ip-pool

Kubernetes Labels

Figure 6.
The configuration screen for a single worker node pool

IP pool

As noted before, in the context of IKS, an IP pool is used to allocate node IP addresses and Kubernetes LoadBalancer Service (MetalLB) IP addresses.

In this step, the IP pool is used to assign IP addresses to the worker nodes. You may need to have your worker nodes on a different subnet from your control plane nodes and LoadBalancer IPs, hence the need to select the IP pool a third time. If you have a single routable subnet, you can select the same IP pool for both the LoadBalancer Service and your node IPs (control plane and worker).

This subnet must be routable and not overlap with an existing subnet in your environment.

IP pools support IPv4 addresses.

Currently IKS UI only allows specifying one InfraConfigPolicy (that is, a virtual machine infrastructure configuration) for all the node pools. If you are specifying different subnets for each node pool, you need to ensure that they are routable through the network interface specified in the InfraConfigPolicy. The network interface was specified in Figure 5, above.

Alternatively, you can specify a different InfraConfigPolicy for each node pool that uses a different subnet/IP Pool. Note that this option is only currently available through the IKS API.

Kubernetes networking concepts

The following section will give you an introduction to Kubernetes networking in general. An IKS cluster with two worker nodes has been configured with the following networking settings for the purposes of this example:

- **Node IP pool:** 10.1.110.0/24 (externally routable)
- **LoadBalancer IP pool:** 10.1.110.0/24 (externally routable)
- **Pod network CIDR:** 192.168.0.0/16 (non-routable)
- **Service network CIDR:** 10.96.0.0/16 (non-routable)

The pod subnets of 192.168.104.64/26 and 192.168.8.192/26 have been assigned for worker1 and worker2, respectively.

Container-to-container communications

Shared volume communication

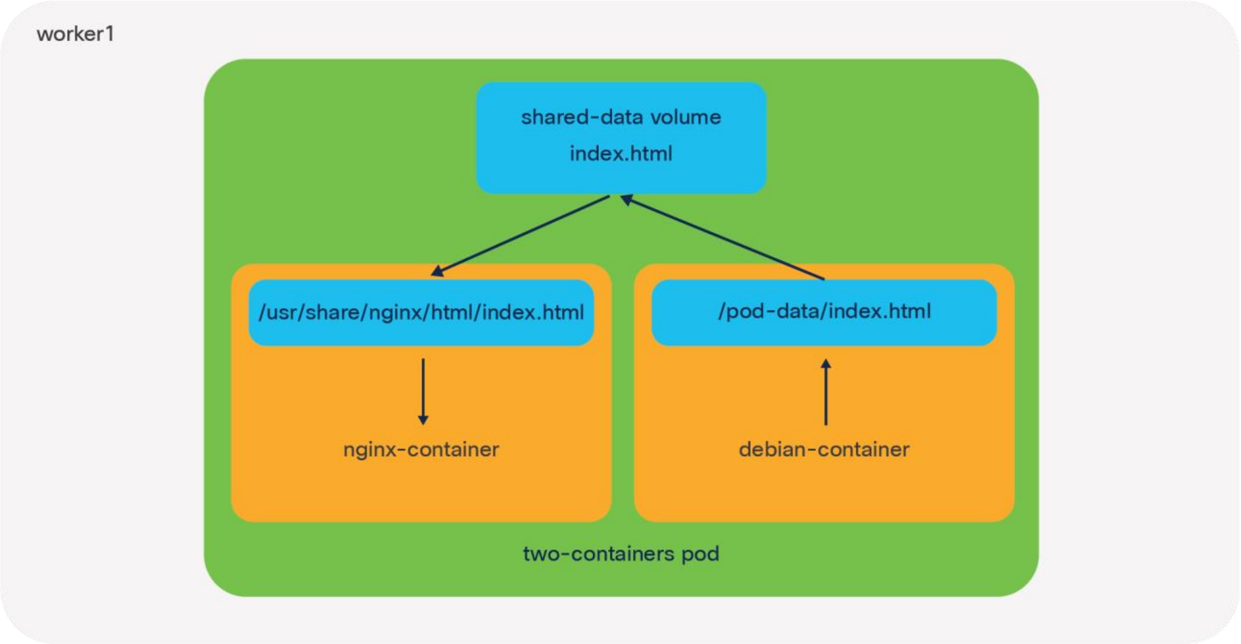


Figure 7.
An illustration of a pod running two containers on an IKS node using a shared volume for networking

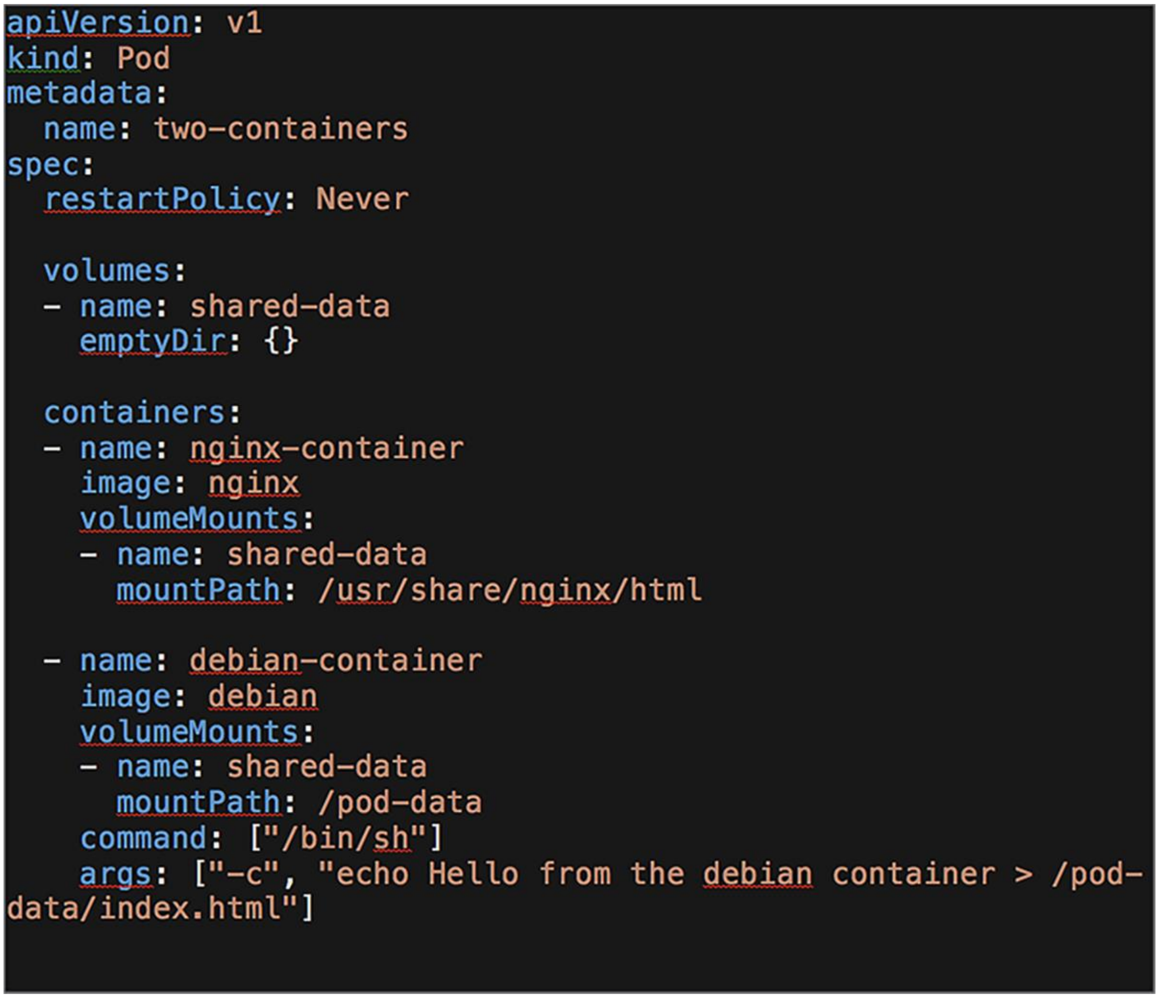
The smallest object you can deploy in Kubernetes is a pod; however, within each pod you may want to run multiple containers. A common use-case for this is a helper where a secondary container helps a primary container with tasks such as pushing and pulling data.

Container-to-container communication within a K8s pod uses either the shared file system or the local-host network interface.

You can test this by using the K8s-provided example, two-container-pod, available through this link: https://k8s.io/examples/pods/two-container-pod.yaml

When you deploy this pod, you should see two containers, **nginx-container** and **debian-container**. When the shared volume method is used, Kubernetes will create a volume in the pod that is mapped to both containers. In the **nginx-container**, files from the shared volume will map to the **/usr/share/nginx/html directory**, while in the debian-container, files will map to the **/pod-data** directory.

When a file is updated (for example, **index.html**) from the Debian container, this change will also be reflected in the NGINX container, thereby providing a mechanism for a helper (Debian) to push and pull data to and from NGINX.



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: two-containers
spec:
  restartPolicy: Never

  volumes:
  - name: shared-data
    emptyDir: {}

  containers:
  - name: nginx-container
    image: nginx
    volumeMounts:
    - name: shared-data
      mountPath: /usr/share/nginx/html

  - name: debian-container
    image: debian
    volumeMounts:
    - name: shared-data
      mountPath: /pod-data
    command: ["/bin/sh"]
    args: ["-c", "echo Hello from the debian container > /pod-
data/index.html"]
```

Figure 8.
The YAML used to deploy the shared volume communication example

Local-host communication

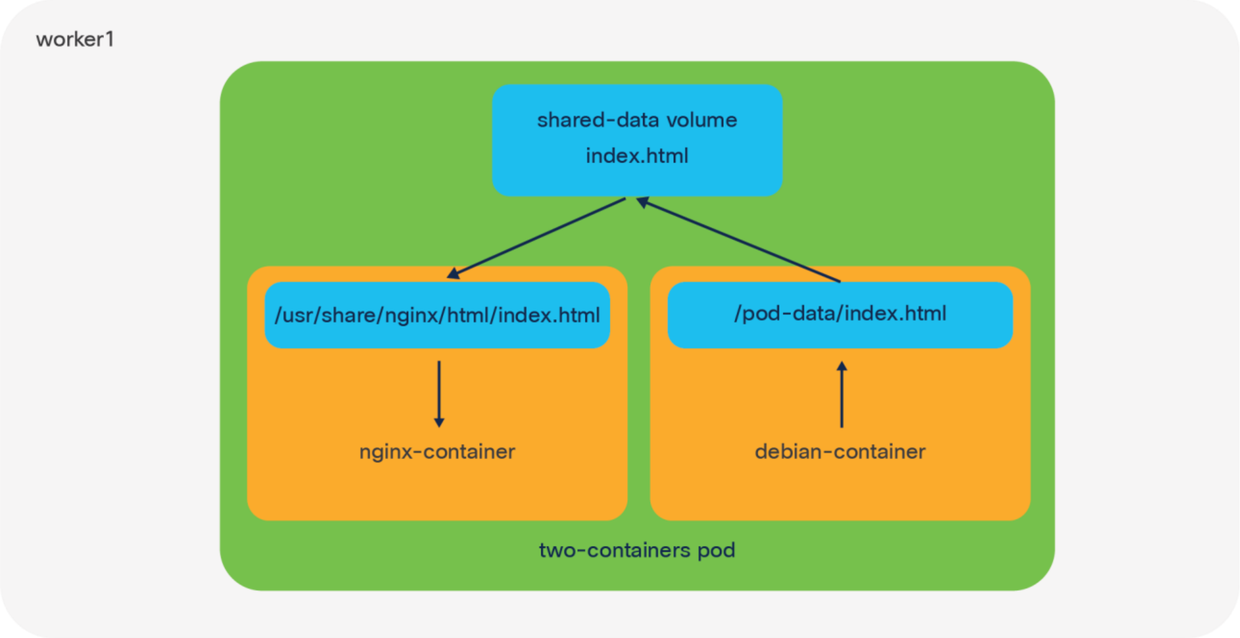


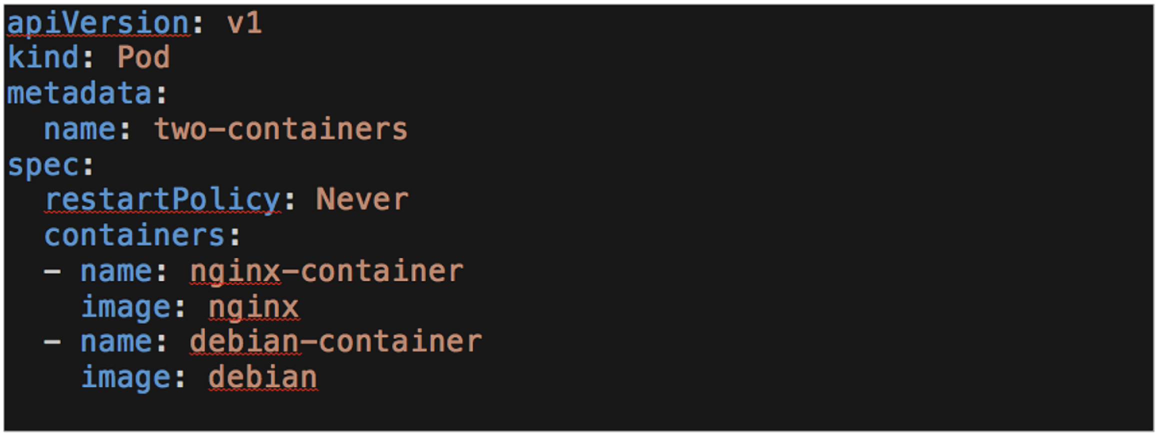
Figure 9.
An illustration of a pod running two containers on an IKS node using the local host for networking

The other method for multiple containers to communicate within a pod is through the local-host interface and the port number to which they’re listening.

You can test this again by using the K8s-provided example, two-container-pod, and modifying it slightly: https://k8s.io/examples/pods/two-container-pod.yaml

In this example NGINX is listening on port 80. If you run **curl https://localhost** from within the Debian container, you should see that the **index.html** page is served back from NGINX.

You can have multiple containers per pod in Kubernetes. This means that all containers in a pod share the same network namespace, IP address, and interfaces.



```
apiVersion: v1
kind: Pod
metadata:
  name: two-containers
spec:
  restartPolicy: Never
  containers:
  - name: nginx-container
    image: nginx
  - name: debian-container
    image: debian
```

Figure 10.
The YAML used to deploy the local-host communication example

Pod-to-pod communication on IKS

Important point: The following examples will use the Kubernetes guestbook application: https://raw.githubusercontent.com/kubernetes/examples/master/guestbook/all-in-one/guestbook-all-in-one.yaml

To follow along, deploy the Kubernetes guestbook to your IKS cluster.

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/examples/master/guestbook/all-in-
one/guestbook-all-in-one.yaml
```

This application comprises two tiers, the front-end web server (Apache) and the back-end DB (Redis). Each tier has multiple pods deployed, with the pods running across two IKS worker nodes.

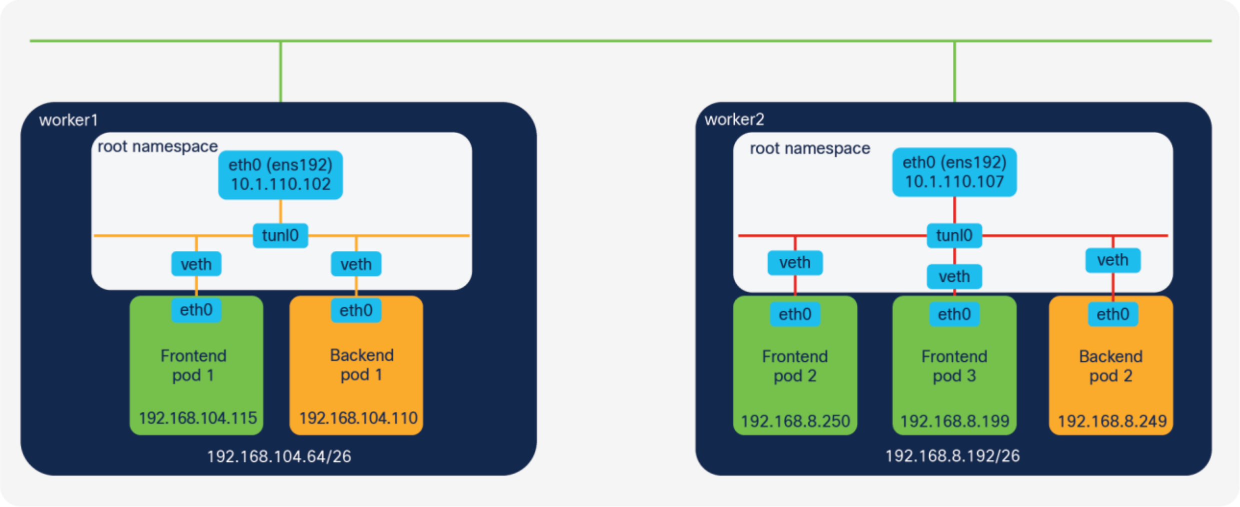


Figure 11.
The IP addresses and subnets used in the subsequent examples

Network namespaces

Kubernetes and containers rely heavily on Linux namespaces to separate resources (processes, networking, mounts, users, etc.) on a machine.

"Namespaces are a feature of the Linux kernel that partitions kernel resources such that one set of processes sees one set of resources while another set of processes sees a different set of resources. . . .

"Network namespaces virtualize the network stack. Each network interface (physical or virtual) is present in exactly 1 namespace and can be moved between namespaces.

Each namespace will have a private set of IP addresses, its own routing table, socket listing, connection tracking table, firewall, and other network-related resources."

(Quoted from https://en.wikipedia.org/wiki/Linux_namespaces.)

If you come from a networking background, the easiest way to think of a namespace is as functioning like a VRF; in Kubernetes each pod receives its own networking namespace (VRF).

Additionally, each Kubernetes node has a default or root networking namespace (VRF) that contains the external interface (for example, ens192) of the Kubernetes node.

When a pod is deployed Kubernetes first creates the pause container. This container runs a constant loop and allows for the networking namespace, and other resources used by all containers in a pod, to be available.

You can view the pause containers by SSHing into one of the Kubernetes nodes and running **sudo docker ps -a | grep pause**

Important point: Linux namespaces are different from Kubernetes namespaces. All mentions in this post are referring to the Linux network namespace.

Virtual cables and virtual Ethernet (veth) pairs

Within each pod exists an interface (for example, eth0). This interface allows connectivity outside the pods network namespace and into the root network namespace.

In the physical world, you might connect two interfaces together with a cable, for example, between a server and a switch. Kubernetes pods and nodes also have two connected interfaces. One side (for example, the eth0 interface) resides in the pod and the other side (for example, the virtual Ethernet interface) exists in the root namespace of the Kubernetes node.

Instead of a physical cable, these two interfaces are connected by a virtual cable. This is known as a virtual ethernet (veth) device pair and allows connectivity outside of the pods. See Figure 11, above, for an illustration.

Connectivity between veths

The connection from the virtual Ethernet (veth) interfaces to other pods and the external world is determined by the CNI plugin. For example, it may be a tunneled interface or a bridged interface.

Important point: Kubernetes does not manage the configuration of the pod-to-pod networking itself; rather, it outsources this configuration to another application, the Container Networking Interface (CNI) plugin.

> *"A CNI plugin is responsible for inserting a network interface into the container network namespace (e.g. one end of a veth pair) and making any necessary changes on the host (e.g. attaching the other end of the veth into a bridge). It should then assign the IP to the interface and setup the routes consistent with the IP Address Management section by invoking appropriate IPAM plugin."*

(Quoted from https://github.com/containernetworking/cni/blob/master/SPEC.md#overview-1)

Other popular plugins include Calico, Flannel, and Contiv, with each implementing the network connectivity in their own way.

Although the methods of implementing networking connectivity may differ between CNI plugins, every one of them must abide by the following requirements that Kubernetes imposes for pod-to-pod communications:

- Pods on a node can communicate with all pods on all nodes without NAT.
- Agents on a node (for example, system daemons, kubelets) can communicate with all pods on that node.
- Pods in the host network of a node can communicate with all pods on all nodes without NAT.

(Quoted from https://kubernetes.io/docs/concepts/cluster-administration/networking)

Important point: NAT is still used to provide connectivity such as source NAT on egress from pod to an external network, or destination NAT, on ingress from the internet into a pod.

The CNI plugin model

A CNI plugin is in fact an executable file that runs on each node and is located in the **/opt/cni/bin** directory. Kubernetes runs this file and passes it the basic configuration details, which can be found in **/etc/cni/net.d**.

IKS uses the Calico IP-IP encapsulation method to provide connectivity between pods. It is responsible for setting up the routing and the interfaces and assigning IP addresses to each pod.

The main components that Calico uses to configure IKS networking on each node are the Felix agent and BIRD.

Felix agent

The Felix agent is the heart of Calico networking. Felix's primary job is to program routes and ACLs on a workload host to provide desired connectivity to and from workloads on the host.

Felix also programs interface information to the kernel for outgoing endpoint traffic. Felix instructs the host to respond to ARPs for workloads with the MAC address of the host.

(Information from https://docs.projectcalico.org/reference/architecture/overview#felix)

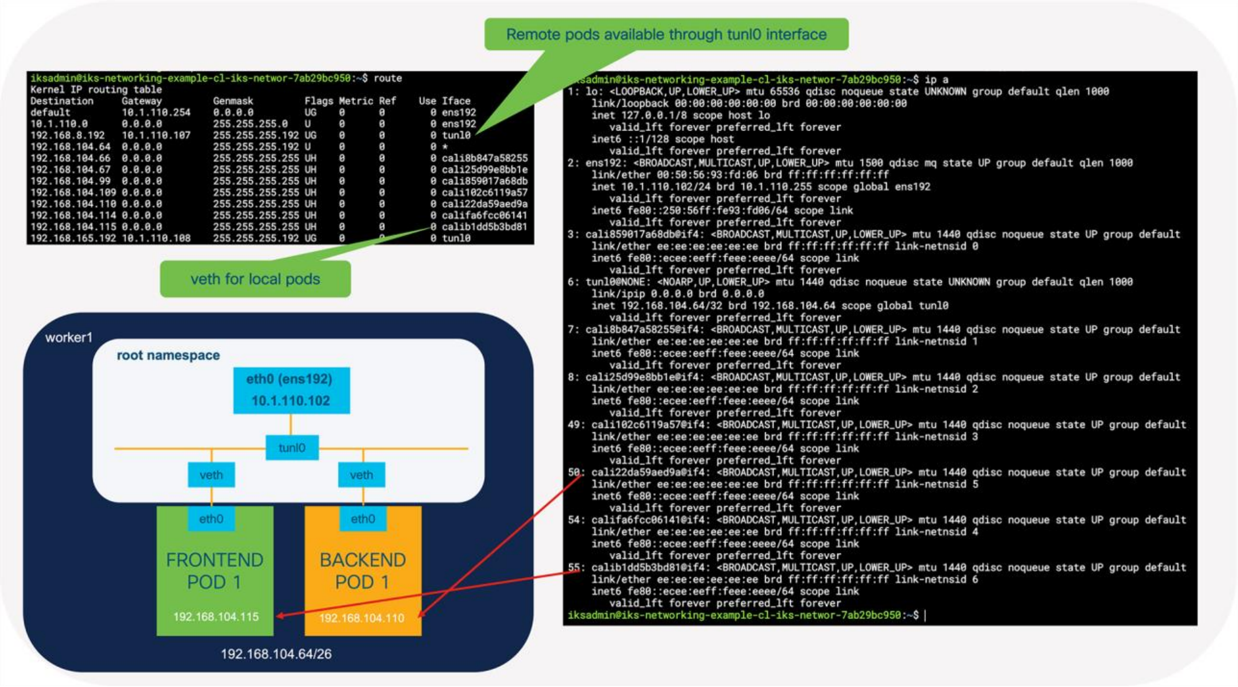
BIRD

The BIRD Internet Routing Daemon (BIRD) gets routed from Felix and distributes these routes to BGP peers on the network for inter-host routing.

When Felix inserts routes into the Linux kernel FIB, the BGP client distributes them to other nodes in the deployment.

(Information from https://docs.projectcalico.org/reference/architecture/overview#bird)

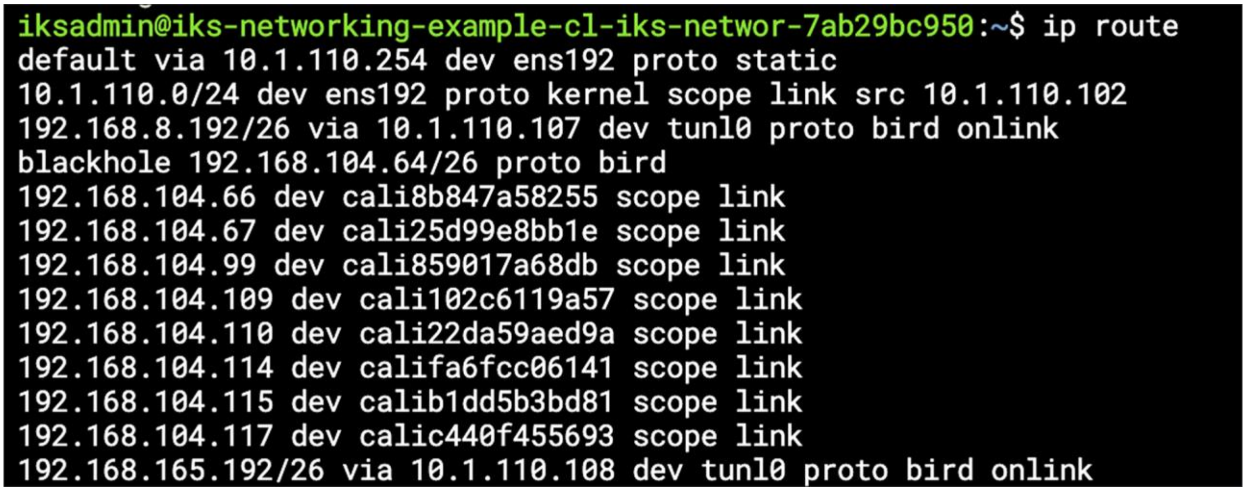
In the context of IKS, the BGP peers are the control plane and worker nodes. BIRD runs on each node within the **calico-node** pods in the **kube-system** namespace.



```
iksadmin@iks-networking-example-cl-iks-networ-7ab29bc950:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.1.110.254    0.0.0.0         UG    0      0        0 ens192
10.1.110.0      0.0.0.0         255.255.255.0   U     0      0        0 ens192
192.168.8.192   10.1.110.107    255.255.255.192 UG    0      0        0 tunl0
192.168.104.64  0.0.0.0         255.255.255.192 U     0      0        0 *
192.168.104.66  0.0.0.0         255.255.255.255 UH    0      0        0 cali8b847a58255
192.168.104.67  0.0.0.0         255.255.255.255 UH    0      0        0 cali25d99e8bb1e
192.168.104.99  0.0.0.0         255.255.255.255 UH    0      0        0 cali859017a68db
192.168.104.109 0.0.0.0         255.255.255.255 UH    0      0        0 cali102c6119a57
192.168.104.110 0.0.0.0         255.255.255.255 UH    0      0        0 cali22da59aed9a
192.168.104.114 0.0.0.0         255.255.255.255 UH    0      0        0 califa6fcc06141
192.168.104.115 0.0.0.0         255.255.255.255 UH    0      0        0 calib1dd5b3bd81
192.168.165.192 10.1.110.108    255.255.255.192 UG    0      0        0 tunl0
```

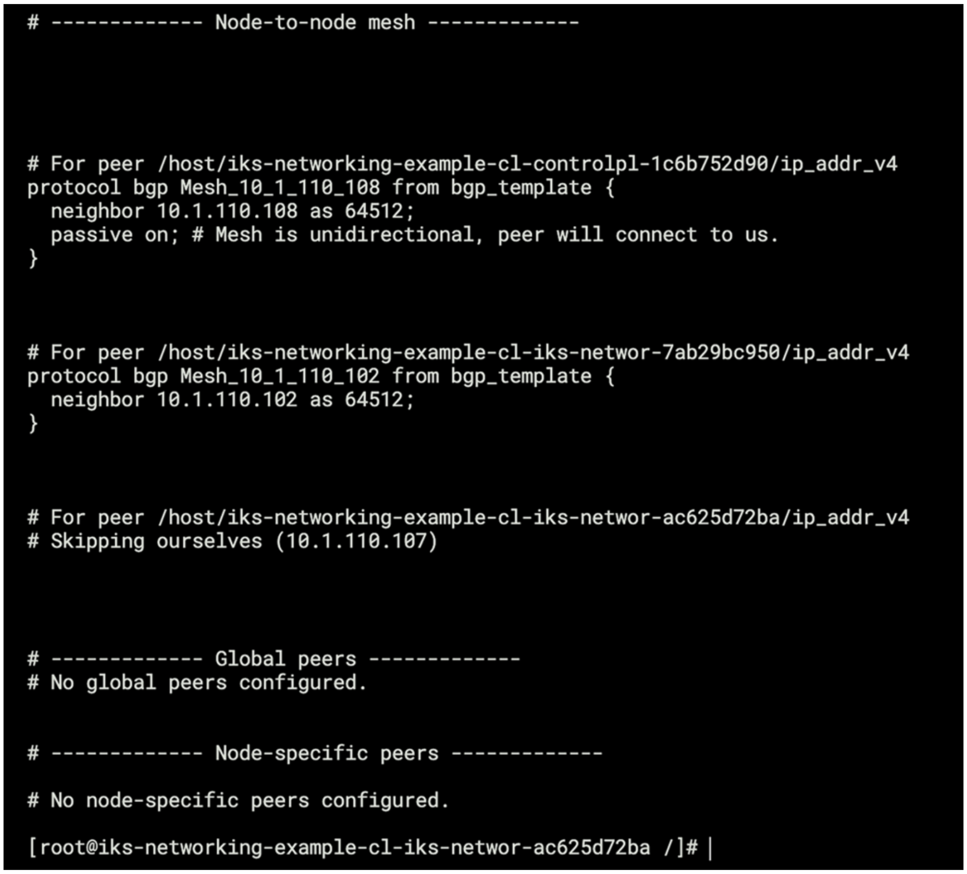
```
sadmin@iks-networking-example-cl-iks-networ-7ab29bc950:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:50:56:93:fd:06 brd ff:ff:ff:ff:ff:ff
    inet 10.1.110.102/24 brd 10.1.110.255 scope global ens192
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe93:fd06/64 scope link
       valid_lft forever preferred_lft forever
3: cali859017a68db@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1440 qdisc noqueue state UP group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::ecee:eeff:feee:eeee/64 scope link
       valid_lft forever preferred_lft forever
6: tunl0@NONE: <NOARP,UP,LOWER_UP> mtu 1440 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
    inet 192.168.104.64/32 brd 192.168.104.64 scope global tunl0
       valid_lft forever preferred_lft forever
7: cali8b847a58255@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1440 qdisc noqueue state UP group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::ecee:eeff:feee:eeee/64 scope link
       valid_lft forever preferred_lft forever
8: cali25d99e8bb1e@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1440 qdisc noqueue state UP group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 2
    inet6 fe80::ecee:eeff:feee:eeee/64 scope link
       valid_lft forever preferred_lft forever
49: cali102c6119a57@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1440 qdisc noqueue state UP group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 3
    inet6 fe80::ecee:eeff:feee:eeee/64 scope link
       valid_lft forever preferred_lft forever
50: cali22da59aed9a@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1440 qdisc noqueue state UP group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 5
    inet6 fe80::ecee:eeff:feee:eeee/64 scope link
       valid_lft forever preferred_lft forever
54: califa6fcc06141@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1440 qdisc noqueue state UP group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 4
    inet6 fe80::ecee:eeff:feee:eeee/64 scope link
       valid_lft forever preferred_lft forever
55: calib1dd5b3bd81@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1440 qdisc noqueue state UP group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 6
    inet6 fe80::ecee:eeff:feee:eeee/64 scope link
       valid_lft forever preferred_lft forever
iksadmin@iks-networking-example-cl-iks-networ-7ab29bc950:~$ |
```

Figure 12.
The interfaces and routes configured on the **worker1** node



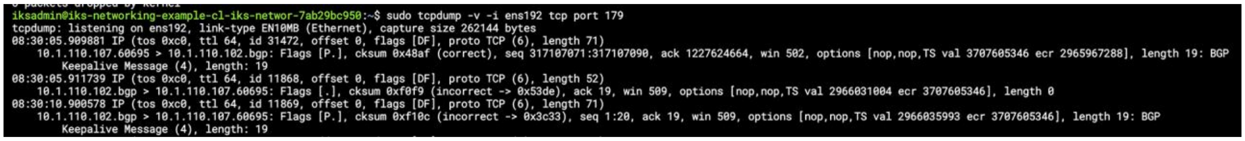
```
iksadmin@iks-networking-example-cl-iks-networ-7ab29bc950:~$ ip route
default via 10.1.110.254 dev ens192 proto static
10.1.110.0/24 dev ens192 proto kernel scope link src 10.1.110.102
192.168.8.192/26 via 10.1.110.107 dev tunl0 proto bird onlink
blackhole 192.168.104.64/26 proto bird
192.168.104.66 dev cali8b847a58255 scope link
192.168.104.67 dev cali25d99e8bb1e scope link
192.168.104.99 dev cali859017a68db scope link
192.168.104.109 dev cali102c6119a57 scope link
192.168.104.110 dev cali22da59aed9a scope link
192.168.104.114 dev califa6fcc06141 scope link
192.168.104.115 dev calib1dd5b3bd81 scope link
192.168.104.117 dev calic440f455693 scope link
192.168.165.192/26 via 10.1.110.108 dev tunl0 proto bird onlink
```

Figure 13.
The routing table on **worker1** showing the routes to **worker2** (192.168.8.192/26) installed by **BIRD**



```
# ------------- Node-to-node mesh -------------



# For peer /host/iks-networking-example-cl-controlpl-1c6b752d90/ip_addr_v4
protocol bgp Mesh_10_1_110_108 from bgp_template {
  neighbor 10.1.110.108 as 64512;
  passive on; # Mesh is unidirectional, peer will connect to us.
}


# For peer /host/iks-networking-example-cl-iks-networ-7ab29bc950/ip_addr_v4
protocol bgp Mesh_10_1_110_102 from bgp_template {
  neighbor 10.1.110.102 as 64512;
}


# For peer /host/iks-networking-example-cl-iks-networ-ac625d72ba/ip_addr_v4
# Skipping ourselves (10.1.110.107)



# ------------- Global peers -------------
# No global peers configured.


# ------------- Node-specific peers -------------

# No node-specific peers configured.

[root@iks-networking-example-cl-iks-networ-ac625d72ba /]# |
```

Figure 14.
Output from **/etc/calico/confd/config/bird.cfg** in one of the **calico-node** pods showing the BGP configuration which was automatically created by BIRD



```
0 packets dropped by kernel
iksadmin@iks-networking-example-cl-iks-networ-7ab29bc950:~$ sudo tcpdump -v -i ens192 tcp port 179
tcpdump: listening on ens192, link-type EN10MB (Ethernet), capture size 262144 bytes
08:30:05.909881 IP (tos 0xc0, ttl 64, id 31472, offset 0, flags [DF], proto TCP (6), length 71)
    10.1.110.107.60695 > 10.1.110.102.bgp: Flags [P.], cksum 0x48af (correct), seq 317107071:317107090, ack 1227624664, win 502, options [nop,nop,TS val 3707605346 ecr 2965967288], length 19: BGP
        Keepalive Message (4), length: 19
08:30:05.911739 IP (tos 0xc0, ttl 64, id 11868, offset 0, flags [DF], proto TCP (6), length 52)
    10.1.110.102.bgp > 10.1.110.107.60695: Flags [.], cksum 0xf0f9 (incorrect -> 0x53de), ack 19, win 509, options [nop,nop,TS val 2966031004 ecr 3707605346], length 0
08:30:10.900578 IP (tos 0xc0, ttl 64, id 11869, offset 0, flags [DF], proto TCP (6), length 71)
    10.1.110.102.bgp > 10.1.110.107.60695: Flags [P.], cksum 0xf10c (incorrect -> 0x3c33), seq 1:20, ack 19, win 509, options [nop,nop,TS val 2966035993 ecr 3707605346], length 19: BGP
        Keepalive Message (4), length: 19
```

Figure 15.
Output from **tcpdump** showing the BGP keepalive messages between the control plane and worker nodess

As per Figure 12, the IKS cluster contains a number of interfaces that have been created:

ens192 is the interface for external connectivity outside of the node. In the following example it has an address in the 10.1.110.0/24 subnet, which is routable in our environment. This is the subnet that was configured for the IP pool in the IKS cluster setup.

tunl0 is the interface that provides the IPIP encapsulation for remote nodes.

calixxxxx are the virtual ethernet interfaces that exist in the root namespace.

Remember, from before, that the **veth** interface connects to the **eth** interface in a pod.

Important point: As mentioned earlier, IKS implements Calico configured for IP-IP encapsulation. This is the reason for the tunneled interface (tunl0). A Kubernetes cluster with a different CNI plugin may have different interfaces, such as docker0, flannel0, or cbr0.

You'll note in the routing table that Calico has inserted some routes. The default routes direct traffic out the external interface (ens192).

The following output is from the **worker1** routing table on the example IKS cluster. This worker node has been assigned the subnet 192.168.104.64/26 for pods. As you can see, any pods on this worker are accessible through the **veth** interface, starting with **calixxxxx**.

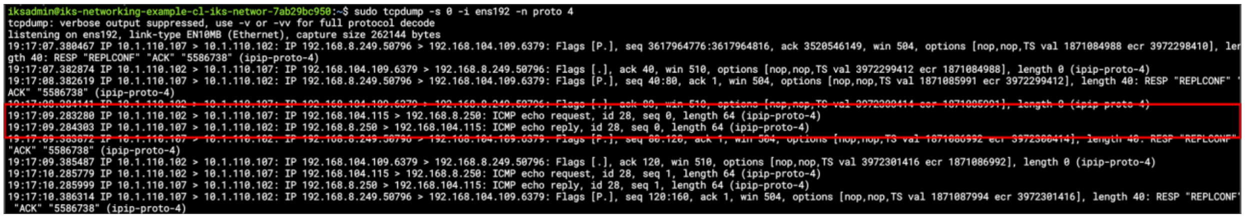
Any time traffic is sent from a pod on **worker1** to a pod on **worker2**, it is sent to the **tunl0** interface.

If pod-to-pod communication takes place on the same node, it will send packets to the veth interfaces.

Traffic between pods on different worker nodes is sent to the tunl0 interface, which will encapsulate the packets with an outer IP packet. The source and destination IP addresses for the outer packet are the external, routable addresses (10.1.110.x subnet in this example).

You can confirm the encapsulation is taking place by capturing packets from the external interface (ens192 in the example). As shown in Figure 16 below, when traffic is sent from one Frontend Pod to another Frontend Pod, the inner packets are encapsulated in an outer packet containing the external source and destination addresses of the ens192 interfaces (10.1.110.102 and 10.1.110.107).

Since the 10.1.110.0/24 subnet is routable, the packets are sent upstream and find their way from worker1 to worker2. Arriving at worker 2, they are decapsulated and sent onto the local veth interface connecting to the Frontend Pod 2.



```
iksadmin@iks-networking-example-cl-iks-networ-7ab29bc950:~$ sudo tcpdump -s 0 -i ens192 -n proto 4
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens192, link-type EN10MB (Ethernet), capture size 262144 bytes
19:17:07.380467 IP 10.1.110.107 > 10.1.110.102: IP 192.168.8.249.50796 > 192.168.104.109.6379: Flags [P.], seq 3617964776:3617964816, ack 3520546149, win 504, options [nop,nop,TS val 1871084988 ecr 3972298410], ler
gth 40: RESP "REPLCONF" "ACK" "5586738" (ipip-proto-4)
19:17:07.382874 IP 10.1.110.102 > 10.1.110.107: IP 192.168.104.109.6379 > 192.168.8.249.50796: Flags [.], ack 40, win 510, options [nop,nop,TS val 3972299412 ecr 1871084988], length 0 (ipip-proto-4)
19:17:08.382619 IP 10.1.110.107 > 10.1.110.102: IP 192.168.8.249.50796 > 192.168.104.109.6379: Flags [P.], seq 40:80, ack 1, win 504, options [nop,nop,TS val 1871085991 ecr 3972299412], length 40: RESP "REPLCONF" '
ACK" "5586738" (ipip-proto-4)
19:17:08.384141 IP 10.1.110.102 > 10.1.110.107: IP 192.168.104.109.6379 > 192.168.8.249.50796: Flags [.], ack 80, win 510, options [nop,nop,TS val 3972300414 ecr 1871085991], length 0 (ipip-proto-4)
19:17:09.283280 IP 10.1.110.102 > 10.1.110.107: IP 192.168.104.115 > 192.168.8.250: ICMP echo request, id 28, seq 0, length 64 (ipip-proto-4)
19:17:09.284303 IP 10.1.110.107 > 10.1.110.102: IP 192.168.8.250 > 192.168.104.115: ICMP echo reply, id 28, seq 0, length 64 (ipip-proto-4)
19:17:09.383872 IP 10.1.110.107 > 10.1.110.102: IP 192.168.8.249.50796 > 192.168.104.109.6379: Flags [P.], seq 80:120, ack 1, win 504, options [nop,nop,TS val 1871086992 ecr 3972300414], length 40: RESP "REPLCONF"
"ACK" "5586738" (ipip-proto-4)
19:17:09.385487 IP 10.1.110.102 > 10.1.110.107: IP 192.168.104.109.6379 > 192.168.8.249.50796: Flags [.], ack 120, win 510, options [nop,nop,TS val 3972301416 ecr 1871086992], length 0 (ipip-proto-4)
19:17:10.285779 IP 10.1.110.102 > 10.1.110.107: IP 192.168.104.115 > 192.168.8.250: ICMP echo request, id 28, seq 1, length 64 (ipip-proto-4)
19:17:10.285999 IP 10.1.110.107 > 10.1.110.102: IP 192.168.8.250 > 192.168.104.115: ICMP echo reply, id 28, seq 1, length 64 (ipip-proto-4)
19:17:10.386314 IP 10.1.110.107 > 10.1.110.102: IP 192.168.8.249.50796 > 192.168.104.109.6379: Flags [P.], seq 120:160, ack 1, win 504, options [nop,nop,TS val 1871087994 ecr 3972301416], length 40: RESP "REPLCONF"
 "ACK" "5586738" (ipip-proto-4)
```

Figure 16.
Confirmation of IP-IP encapsulation for packets sent directly from pods running on different hosts.

Kubernetes Services – tracking pods and providing external access

Although pod-to-pod communication takes place using IP in IP encapsulation, that's only part of the implementation. It is not realistic that pods will communicate directly; for example, multiple pods may all perform the same function, as is the case of the guestbook application.

The guestbook has multiple frontend pods storing and retrieving messages from multiple backend database pods.

- Should each frontend pod only ever talk to one backend pod?
- If not, should each frontend pod have to keep its own list of which backend pods are available?
- If the pod subnets are internal to the nodes only and not-routable, how can the application be accessed from an external network

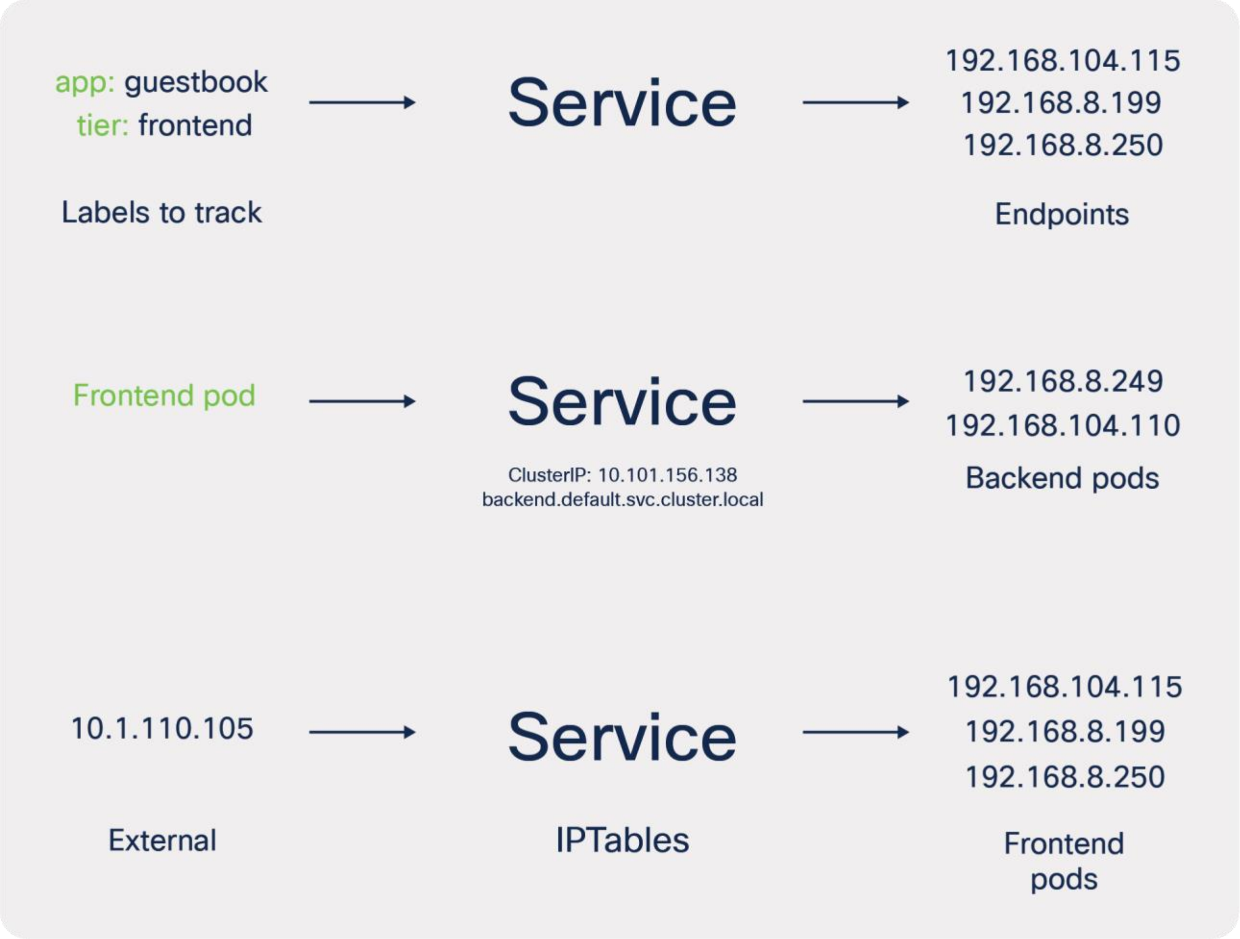
All of these points are addressed through the use of Kubernetes Services. Services are a native concept to Kubernetes, meaning they do not rely on an external plugin as is the case with the CNI for the pod and routing configuration

There are three primary services available:

- ClusterIP
- NodePort
- LoadBalancer

Kubernetes Services help with the following:

- Keeping track of pods
- Providing internal access from one pod (for example, frontend) to another (for example, backend)
- Providing L3/L4 connectivity from an external client (for example, a web browser) to a pod (for example, frontend)



Labels, selectors, and endpoints

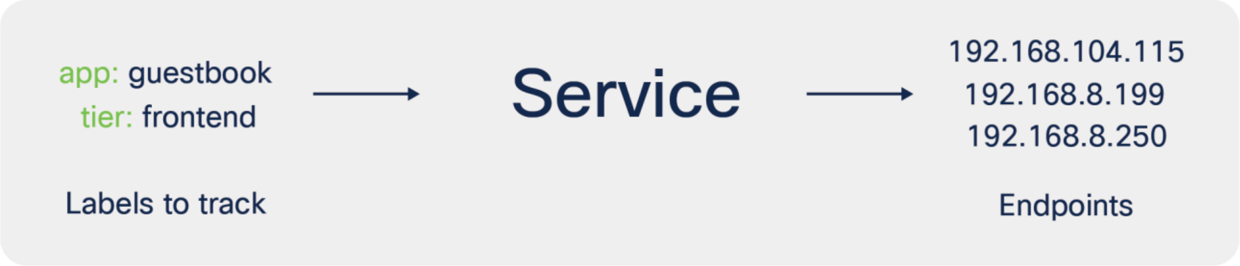
Labels and selectors are very important concepts in Kubernetes and are relevant to how a Kubernetes service tracks endpoints.

> *"Labels are key/value pairs that are attached to objects, such as pods [and] are intended to be used to specify identifying attributes of objects that are meaningful and relevant to users. Unlike names and UIDs, labels do not provide uniqueness. In general, we expect many objects to carry the same label(s)."*

> *"Via a label selector, the client/user can identify a set of objects."*

(Quoted from https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/)

Keeping track of pods



app: guestbook
tier: frontend

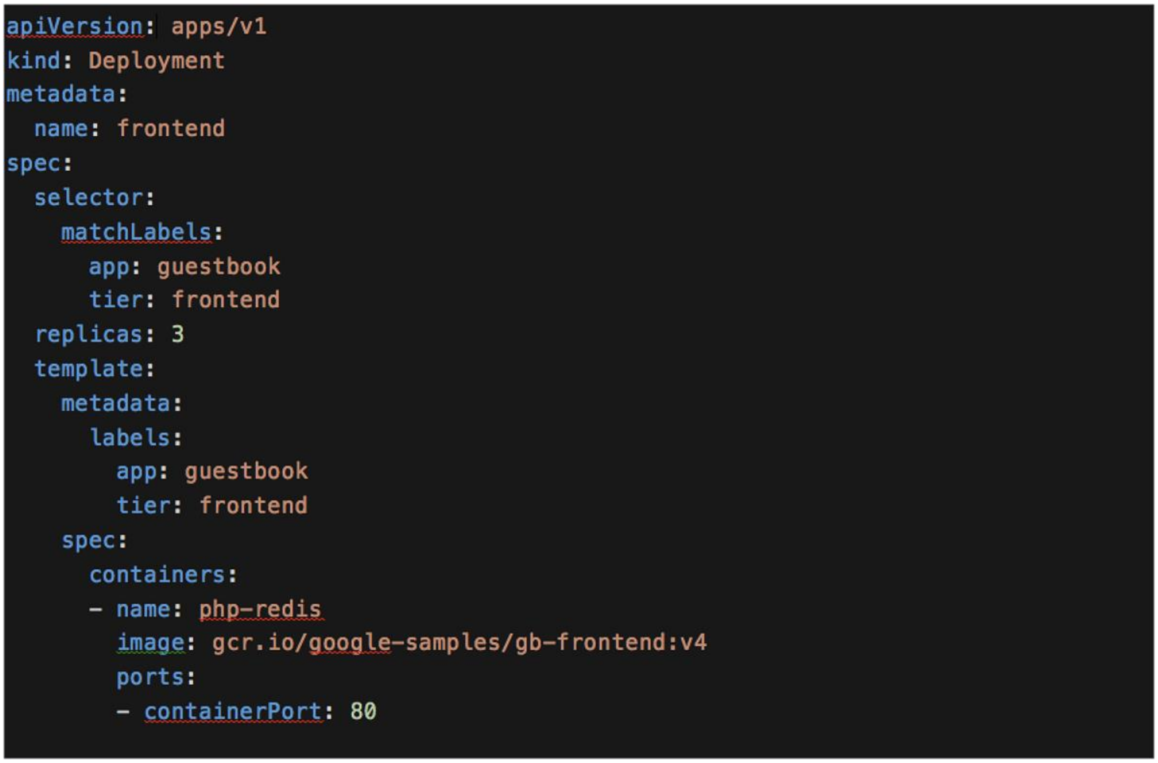
Labels to track

Service

192.168.104.115
192.168.8.199
192.168.8.250

Endpoints

To demonstrate Kubernetes services on IKS, have a look at the deployment file for the guestbook application frontend pods.

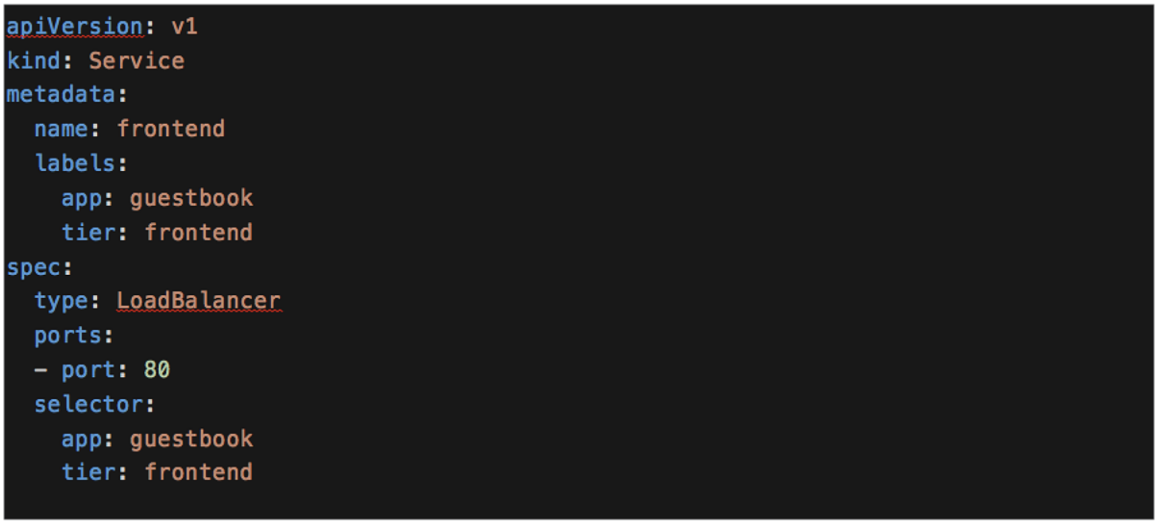


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google-samples/gb-frontend:v4
        ports:
        - containerPort: 80
```

Figure 17.
YAML definition for the guestbook frontend deployment

There are two labels, app: guestbook and tier: frontend, associated to the frontend pods that are deployed. These pods will receive an IP address from the range 192.168.x.x that we specified in the IKS cluster creation wizard (pod network CIDR).

Now look at the service description for the frontend pods. There is one service definition for each deployment (frontend and backend in this example).



```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```

Figure 18.
YAML definition for the guestbook frontend service

In this YAML the service has a selector that uses the same keys/values (**app: guestbook** and **tier: frontend**) as are configured in the deployment.

When a service is created, Kubernetes will track the IP addresses assigned to any of the pods that use the labels in the selector field. Any new pods created will automatically be tracked by Kubernetes.

As you scale up a deployment (potentially 100s or 1000s of pods deployed), Kubernetes will keep track of the internal pod endpoint IP addresses (192.168.x.x in this example).

You can use the command, **kubectl describe service <the-service-name>,** to view the service type, the labels Kubernetes is tracking, and the pod endpoints that have those same labels.

Providing internal access from one pod (for example, frontend) to another (for example, backend)

If you look at the pods or processes running on the Kubernetes nodes, you won’t find one named “Kubernetes Service.” As per the Kubernetes documentation,

“A service is an abstraction which defines a logical set of Pods and a policy by which to access them.”

(Quoted from https://kubernetes.io/docs/concepts/services-networking/service/.)

While a Kubernetes Service is a logical concept, under the covers a pod called **kube-proxy** is running on each node to implements these rules.

The kube-proxy pod watches the Kubernetes control plane for changes. Every time a new service is created, it will configure IPTables rules on the control plane and worker nodes. These rules redirect traffic from the ClusterIP (see below for details) to the IP address of the pod (192.168.x.x in the example).

Netfilter, IPTables, and connection tracking (Conntrack)

IPTables and connection tracking play a large part in the forwarding of traffic in an IKS cluster. Although you may only ever need to define Kubernetes resources such as a service or ingress, understanding how the rules are implemented can help in some scenarios, for example, when troubleshooting.

> *"IPTables is a user-space utility program that allows a system administrator to configure the IP packet filter rules of the Linux kernel firewall, implemented as different Netfilter modules."*

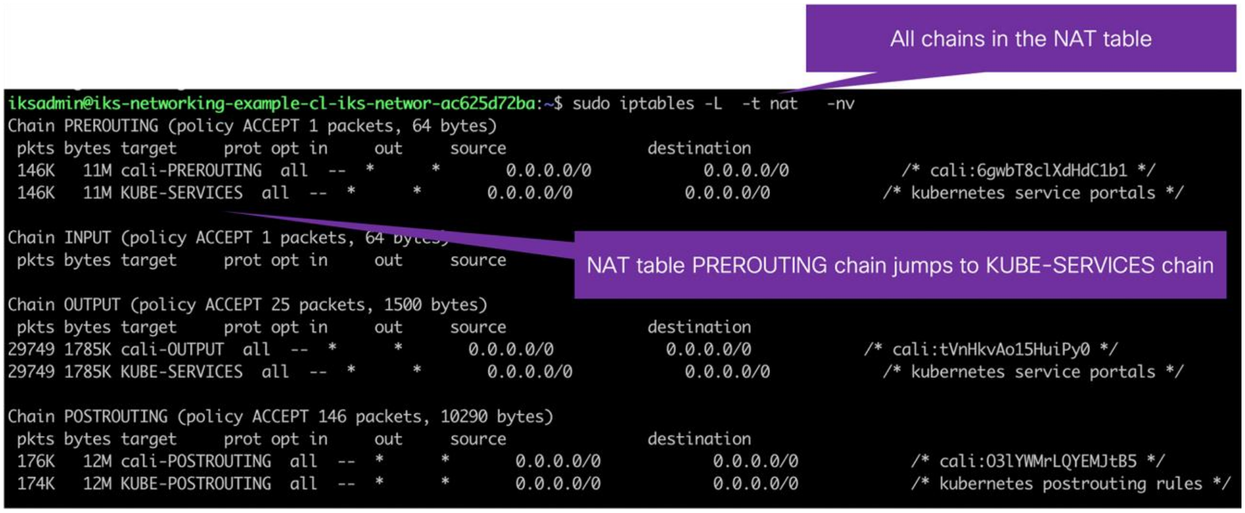
(Quoted from https://en.wikipedia.org/wiki/Iptables.)

> *"The connection tracking system stores information about the state of a connection in a memory structure that contains the source and destination IP addresses, port number pairs, protocol types, state, and timeout. With this extra information, we can define more intelligent filtering policies."*

(Quoted from https://people.netfilter.org/pablo/docs/login.pdf.)

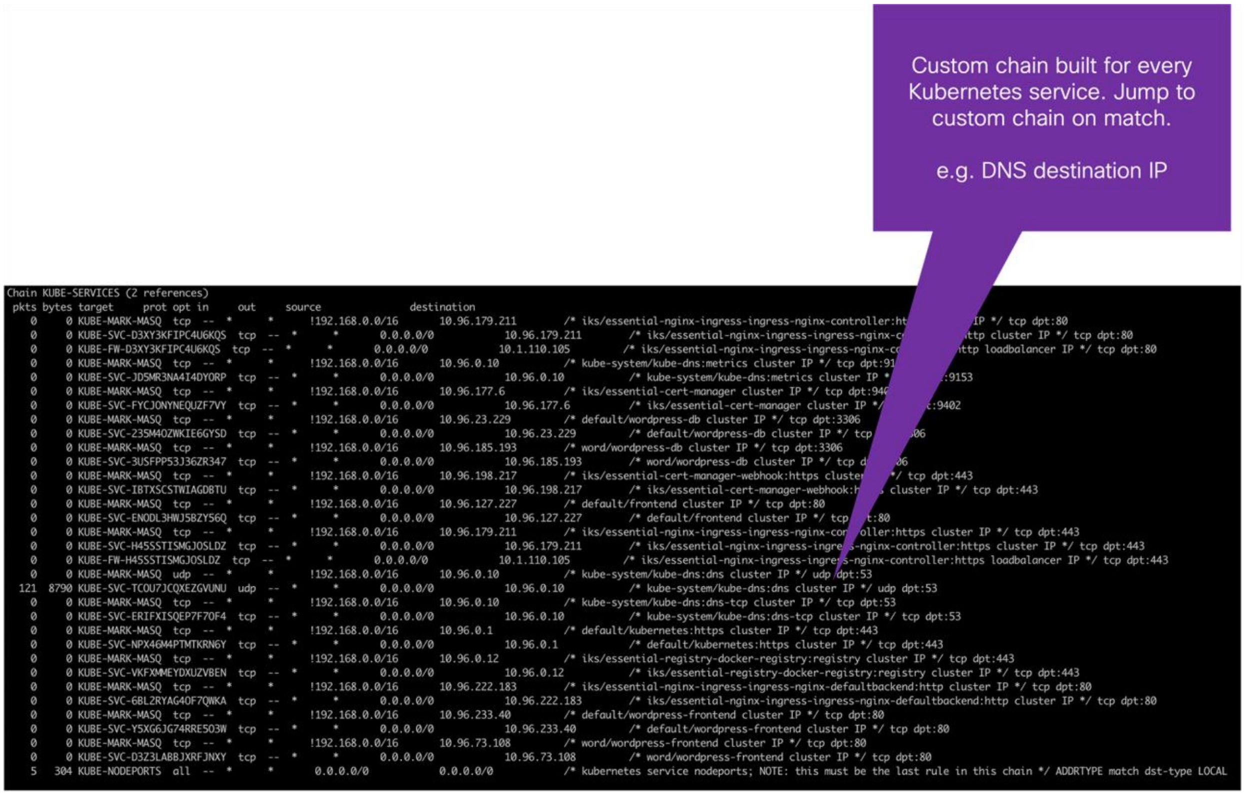
IPTables comprises **tables** with various purposes. The tables contain **chains** of rules that dictate how to treat each packet. There are four main tables and five predefined chains (see details below). Custom chains can also be created. This is shown later in more detail.

At a high level, a packet traverses the set of rules in each chain, and if a match is found, the packet can either jump to another chain in the same table, or a verdict of **accept**, **reject**, or **drop** may affect the packet. If the current rule does not match the patch, it continues to the next rule.


All chains in the NAT table
NAT table PREROUTING chain jumps to KUBE-SERVICES chain

```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L  -t nat   -nv
Chain PREROUTING (policy ACCEPT 1 packets, 64 bytes)
 pkts bytes target     prot opt in     out     source               destination
 146K   11M cali-PREROUTING  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:6gwbT8clXdHdC1b1 */
 146K   11M KUBE-SERVICES  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes service portals */

Chain INPUT (policy ACCEPT 1 packets, 64 bytes)
 pkts bytes target     prot opt in     out     source

Chain OUTPUT (policy ACCEPT 25 packets, 1500 bytes)
 pkts bytes target     prot opt in     out     source               destination
29749 1785K cali-OUTPUT  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:tVnHkvAo15HuiPy0 */
29749 1785K KUBE-SERVICES  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes service portals */

Chain POSTROUTING (policy ACCEPT 146 packets, 10290 bytes)
 pkts bytes target     prot opt in     out     source               destination
 176K   12M cali-POSTROUTING  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:O3lYWMrLQYEMJtB5 */
 174K   12M KUBE-POSTROUTING  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes postrouting rules */
```

Figure 19.
IPTables tables and chains example

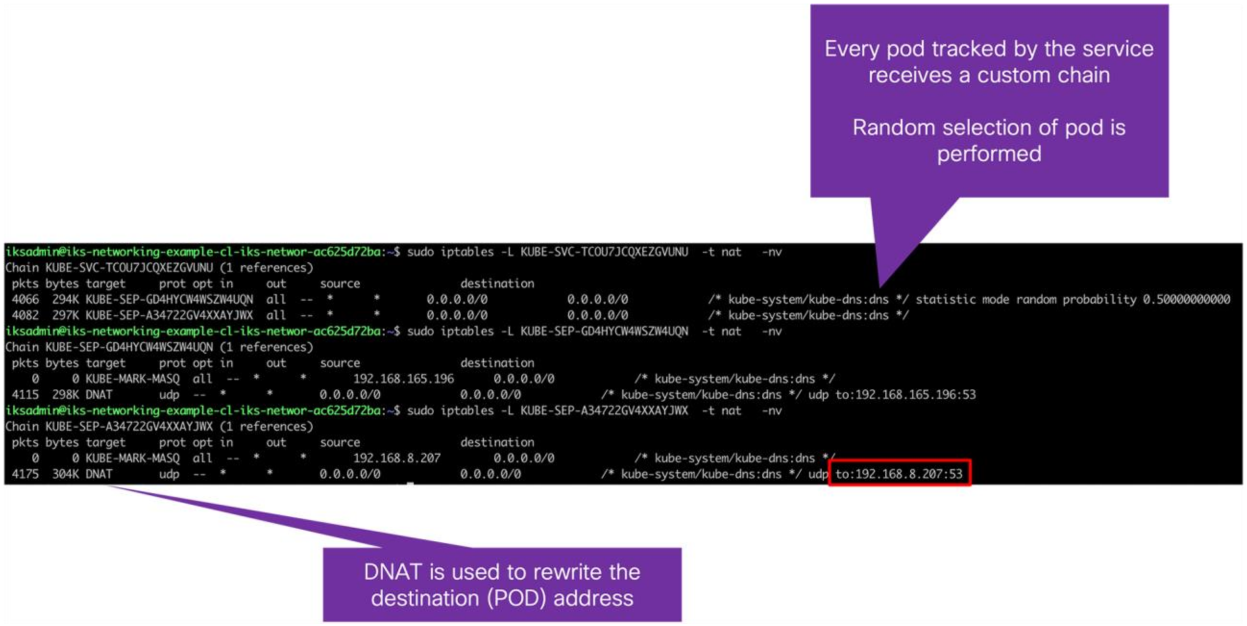


Custom chain built for every Kubernetes service. Jump to custom chain on match.

e.g. DNS destination IP

```
Chain KUBE-SERVICES (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-controller:ht        IP */ tcp dpt:80
    0     0 KUBE-SVC-D3XY3KFIPC4U6KQS  tcp  --  *      *       0.0.0.0/0            10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-c        http cluster IP */ tcp dpt:80
    0     0 KUBE-FW-D3XY3KFIPC4U6KQS  tcp  --  *      *       0.0.0.0/0            10.1.110.105         /* iks/essential-nginx-ingress-ingress-nginx-co        http loadbalancer IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.0.10           /* kube-system/kube-dns:metrics cluster IP */ tcp dpt:91
    0     0 KUBE-SVC-JD5MR3NA4I4DYORP  tcp  --  *      *       0.0.0.0/0            10.96.0.10           /* kube-system/kube-dns:metrics cluster IP *        :9153
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.177.6          /* iks/essential-cert-manager cluster IP */ tcp dpt:940
    0     0 KUBE-SVC-FYCJONYNEQUZF7VY  tcp  --  *      *       0.0.0.0/0            10.96.177.6          /* iks/essential-cert-manager cluster IP */        t:9402
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.23.229         /* default/wordpress-db cluster IP */ tcp dpt:3306
    0     0 KUBE-SVC-235M4OZWKIE6GYSD  tcp  --  *      *       0.0.0.0/0            10.96.23.229         /* default/wordpress-db cluster IP */ tcp      06
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.185.193        /* word/wordpress-db cluster IP */ tcp dpt:3306
    0     0 KUBE-SVC-3USFPP53J36ZR347  tcp  --  *      *       0.0.0.0/0            10.96.185.193        /* word/wordpress-db cluster IP */ tcp d    06
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.198.217        /* iks/essential-cert-manager-webhook:https cluste   */ tcp dpt:443
    0     0 KUBE-SVC-IBTXSCSTWIAGDBTU  tcp  --  *      *       0.0.0.0/0            10.96.198.217        /* iks/essential-cert-manager-webhook:h   s cluster IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.127.227        /* default/frontend cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-ENODL3HWJ5BZY56Q  tcp  --  *      *       0.0.0.0/0            10.96.127.227        /* default/frontend cluster IP */ tcp   t:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-con oller:https cluster IP */ tcp dpt:443
    0     0 KUBE-SVC-H45SSTISMGJOSLDZ  tcp  --  *      *       0.0.0.0/0            10.96.179.211        /* iks/essential-nginx-ingress-ingre s-nginx-controller:https cluster IP */ tcp dpt:443
    0     0 KUBE-FW-H45SSTISMGJOSLDZ  tcp  --  *      *       0.0.0.0/0            10.1.110.105         /* iks/essential-nginx-ingress-ingre s-nginx-controller:https loadbalancer IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  udp  --  *      *      !192.168.0.0/16       10.96.0.10           /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
  121  8790 KUBE-SVC-TCOU7JCQXEZGVUNU  udp  --  *      *       0.0.0.0/0            10.96.0.10           /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.0.10           /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
    0     0 KUBE-SVC-ERIFXISQEP7F7OF4  tcp  --  *      *       0.0.0.0/0            10.96.0.10           /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.0.1            /* default/kubernetes:https cluster IP */ tcp dpt:443
    0     0 KUBE-SVC-NPX46M4PTMTKRN6Y  tcp  --  *      *       0.0.0.0/0            10.96.0.1            /* default/kubernetes:https cluster IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.0.12           /* iks/essential-registry-docker-registry:registry cluster IP */ tcp dpt:443
    0     0 KUBE-SVC-VKFXMMEYDXUZVBEN  tcp  --  *      *       0.0.0.0/0            10.96.0.12           /* iks/essential-registry-docker-registry:registry cluster IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.222.183        /* iks/essential-nginx-ingress-ingress-nginx-defaultbackend:http cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-6BL2RYAG4OF7QWKA  tcp  --  *      *       0.0.0.0/0            10.96.222.183        /* iks/essential-nginx-ingress-ingress-nginx-defaultbackend:http cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.233.40         /* default/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-Y5XG6JG74RRE5O3W  tcp  --  *      *       0.0.0.0/0            10.96.233.40         /* default/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.73.108         /* word/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-D3Z3LABBJXRFJNXY  tcp  --  *      *       0.0.0.0/0            10.96.73.108         /* word/wordpress-frontend cluster IP */ tcp dpt:80
    5   304 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
```

Figure 20.
IPTables tables and chains example



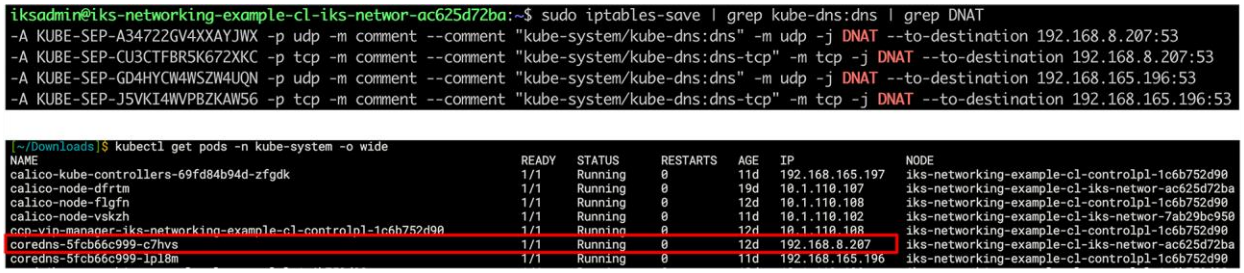
Every pod tracked by the service receives a custom chain

Random selection of pod is performed

```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SVC-TCOU7JCQXEZGVUNU  -t nat   -nv
Chain KUBE-SVC-TCOU7JCQXEZGVUNU (1 references)
 pkts bytes target     prot opt in     out     source               destination
 4066  294K KUBE-SEP-GD4HYCW4WSZW4UQN  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kube-system/kube-dns:dns */ statistic mode random probability 0.50000000000
 4082  297K KUBE-SEP-A34722GV4XXAYJWX  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kube-system/kube-dns:dns */
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SEP-GD4HYCW4WSZW4UQN  -t nat   -nv
Chain KUBE-SEP-GD4HYCW4WSZW4UQN (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       192.168.165.196      0.0.0.0/0            /* kube-system/kube-dns:dns */
 4115  298K DNAT       udp  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kube-system/kube-dns:dns */ udp to:192.168.165.196:53
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SEP-A34722GV4XXAYJWX  -t nat   -nv
Chain KUBE-SEP-A34722GV4XXAYJWX (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       192.168.8.207        0.0.0.0/0            /* kube-system/kube-dns:dns */
 4175  304K DNAT       udp  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kube-system/kube-dns:dns */ udp to:192.168.8.207:53
```

DNAT is used to rewrite the destination (POD) address

Figure 21.
IPTables tables and chains example



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables-save | grep kube-dns:dns | grep DNAT
-A KUBE-SEP-A34722GV4XXAYJWX -p udp -m comment --comment "kube-system/kube-dns:dns" -m udp -j DNAT --to-destination 192.168.8.207:53
-A KUBE-SEP-CU3CTFBR5K672XKC -p tcp -m comment --comment "kube-system/kube-dns:dns-tcp" -m tcp -j DNAT --to-destination 192.168.8.207:53
-A KUBE-SEP-GD4HYCW4WSZW4UQN -p udp -m comment --comment "kube-system/kube-dns:dns" -m udp -j DNAT --to-destination 192.168.165.196:53
-A KUBE-SEP-J5VKI4WVPBZKAW56 -p tcp -m comment --comment "kube-system/kube-dns:dns-tcp" -m tcp -j DNAT --to-destination 192.168.165.196:53
```

```
[~/Downloads]$ kubectl get pods -n kube-system -o wide
NAME                                                        READY   STATUS      RESTARTS   AGE   IP                NODE
calico-kube-controllers-69fd84b94d-zfgdk                    1/1     Running     0          11d   192.168.165.197   iks-networking-example-cl-controlpl-1c6b752d90
calico-node-dfrtm                                           1/1     Running     0          19d   10.1.110.107      iks-networking-example-cl-iks-networ-ac625d72ba
calico-node-flgfn                                           1/1     Running     0          12d   10.1.110.108      iks-networking-example-cl-controlpl-1c6b752d90
calico-node-vskzh                                           1/1     Running     0          11d   10.1.110.102      iks-networking-example-cl-iks-networ-7ab29bc950
ccp-vip-manager-iks-networking-example-cl-controlpl-1c6b752d90   1/1     Running     0          12d   10.1.110.108      iks-networking-example-cl-controlpl-1c6b752d90
coredns-5fcb66c999-c7hvs                                    1/1     Running     0          12d   192.168.8.207     iks-networking-example-cl-iks-networ-ac625d72ba
coredns-5fcb66c999-lpl8m                                    1/1     Running     0          11d   192.168.165.196   iks-networking-example-cl-controlpl-1c6b752d90
```

Figure 22.
IPTables tables and chains example

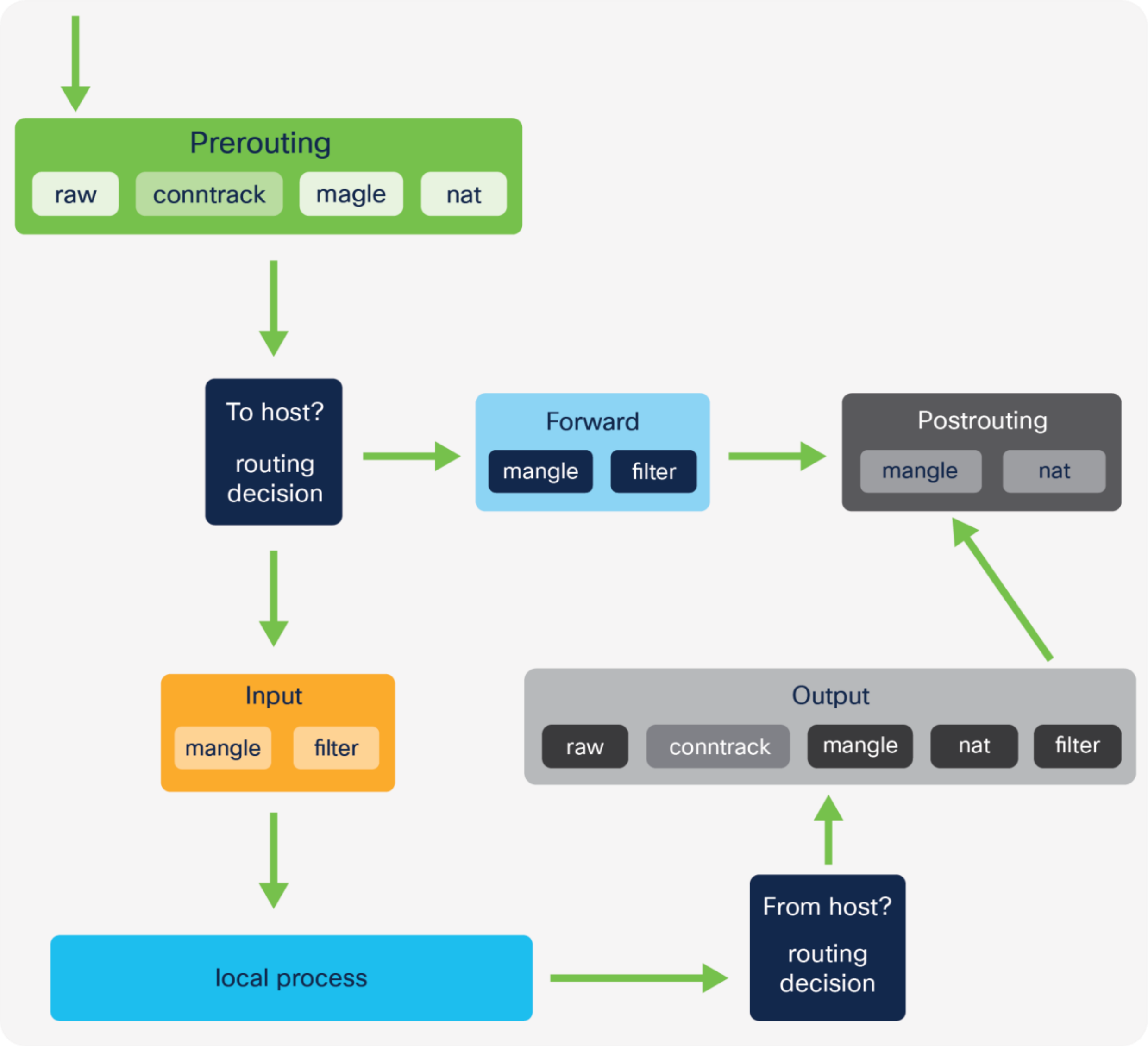


Figure 23.
IPTables tables and chains

Tables

- **Raw**
    - This table is used to store rules that mark packets to opt out of connection tracking.
- **Mangle**
    - These rules alter IP headers such as TTL and can also mark packets for further processing.
- NAT
    - Rules that implement SNAT and DNAT functions
- **Filter**
    - matches packets and takes action, for example, **accept** or **drop**

Chains

- **Prerouting**
    - First chain used after traffic is received by an interface
- **Forward**
    - Used for any packet that has been routed and is not destined for a local host process
- **Postrouting**
    - Rules that apply to packets after a routing decision has been made and it’s determined they are not for local host processes
- **Input**
    - Packets destined for the host (that is, the Kubernetes node)
    - **ip route show table local**
- **Output**
    - Packets sent from the host

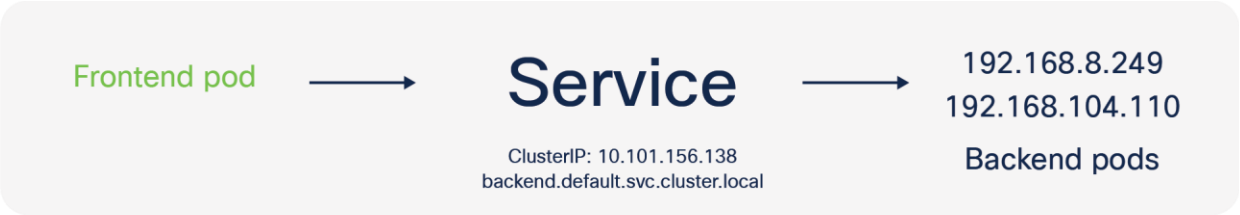
When manipulating and forwarding traffic in IKS, the **Prerouting**, **Forward**, and **Postrouting** chains are primarily used.

Viewing Kubernetes Services

Run the following command to view the Kubernetes services for your deployment.

```
kubectl get services
```

Kubernetes ClusterIP



The Kubernetes ClusterIP is an IP address assigned to a service that is internal to the Kubernetes cluster and only reachable from within the cluster. This subnet was entered in the service network CIDR field as part of the IKS cluster creation wizard.

Every time a new Kubernetes services is created a ClusterIP is assigned.

Continuing with the IKS guestbook example, each frontend and backend service has been configured with a ClusterIP (for example, 10.96.127.227). The **kube-proxy** pod configures IPTables rules to redirect any traffic destined to these ClusterIPs to one of the available pods for that service.

Services and Network Address Translation (NAT)

Network Address Translation (NAT) is used provide outbound connectivity from a pod to an external network, or internal connectivity from a network to a pod.

- Pod to external network

  The pod subnets in IKS are internal to a cluster (192.168.x.x in the example). When a packet exits a Kubernetes control plane or worker node from a pod, it requires an externally routable source address. This is implemented using source NAT (SNAT). SNAT replaces the source IP on a packet.

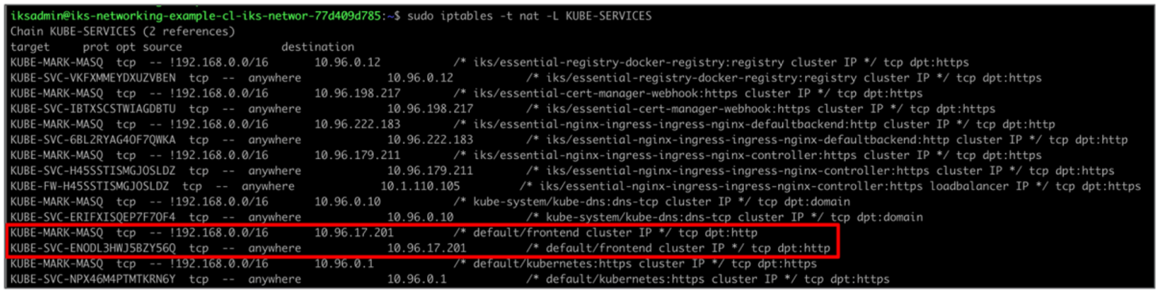
  IKS uses the IP address of the node from which the packet egresses as the source IP of the packet.

- External network to pod

  Since the pod subnet (192.168.x.x in the example) is internal to the IKS cluster, packets coming from a subnet that is not the pod network require an external IP address on which they can reach the pod. Destination NAT (DNAT) changes the destination IP address from external address (for example, a node) to the internal pod IP address. Packets from an external network to a pod also include return traffic (for example, packets in a flow between frontend and backend pods). To handle return traffic, connection tracking is implemented to maintain state and ensure that the return traffic reaches the correct destination pod.

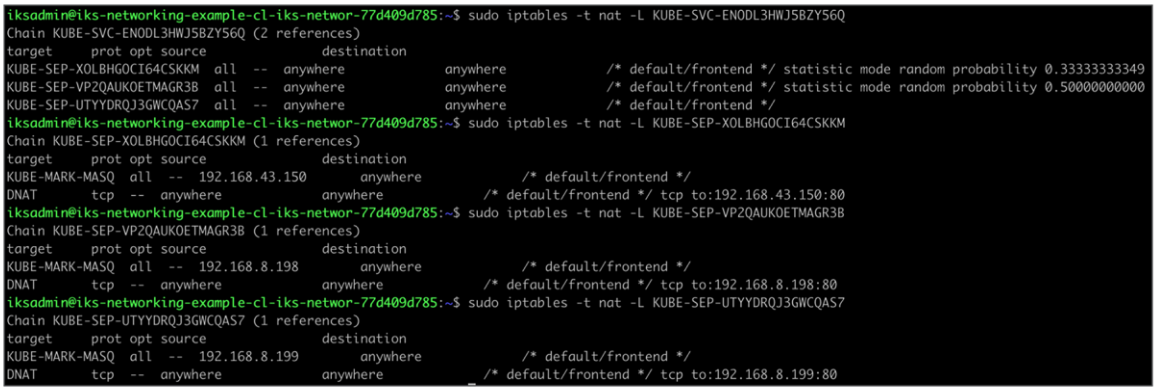
To recap, Kubernetes Services are translated into IPTables rules and provide connectivity between pods as well as inbound and outbound traffic.

You can view the NAT configuration by viewing the IPTables rules.



```
iksadmin@iks-networking-example-cl-iks-networ-77d409d785:~$ sudo iptables -t nat -L KUBE-SERVICES
Chain KUBE-SERVICES (2 references)
target     prot opt source               destination
KUBE-MARK-MASQ  tcp  -- !192.168.0.0/16       10.96.0.12           /* iks/essential-registry-docker-registry:registry cluster IP */ tcp dpt:https
KUBE-SVC-VKFXMMEYDXUZVBEN  tcp  --  anywhere             10.96.0.12           /* iks/essential-registry-docker-registry:registry cluster IP */ tcp dpt:https
KUBE-MARK-MASQ  tcp  -- !192.168.0.0/16       10.96.198.217        /* iks/essential-cert-manager-webhook:https cluster IP */ tcp dpt:https
KUBE-SVC-IBTXSCSTWIAGDBTU  tcp  --  anywhere             10.96.198.217        /* iks/essential-cert-manager-webhook:https cluster IP */ tcp dpt:https
KUBE-MARK-MASQ  tcp  -- !192.168.0.0/16       10.96.222.183        /* iks/essential-nginx-ingress-ingress-nginx-defaultbackend:http cluster IP */ tcp dpt:http
KUBE-SVC-6BL2RYAG4OF7QWKA  tcp  --  anywhere             10.96.222.183        /* iks/essential-nginx-ingress-ingress-nginx-defaultbackend:http cluster IP */ tcp dpt:http
KUBE-MARK-MASQ  tcp  -- !192.168.0.0/16       10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-controller:https cluster IP */ tcp dpt:https
KUBE-SVC-H45SSTISMGJOSLDZ  tcp  --  anywhere             10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-controller:https cluster IP */ tcp dpt:https
KUBE-FW-H45SSTISMGJOSLDZ  tcp  --  anywhere             10.1.110.105         /* iks/essential-nginx-ingress-ingress-nginx-controller:https loadbalancer IP */ tcp dpt:https
KUBE-MARK-MASQ  tcp  -- !192.168.0.0/16       10.96.0.10           /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:domain
KUBE-SVC-ERIFXISQEP7F7OF4  tcp  --  anywhere             10.96.0.10           /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:domain
KUBE-MARK-MASQ  tcp  -- !192.168.0.0/16       10.96.17.201         /* default/frontend cluster IP */ tcp dpt:http
KUBE-SVC-ENODL3HWJ5BZY56Q  tcp  --  anywhere             10.96.17.201         /* default/frontend cluster IP */ tcp dpt:http
KUBE-MARK-MASQ  tcp  -- !192.168.0.0/16       10.96.0.1            /* default/kubernetes:https cluster IP */ tcp dpt:https
KUBE-SVC-NPX46M4PTMTKRN6Y  tcp  --  anywhere             10.96.0.1            /* default/kubernetes:https cluster IP */ tcp dpt:https
```

Figure 24.
IPTables rules implemented on worker node 1 for the guestbook frontend service



```
iksadmin@iks-networking-example-cl-iks-networ-77d409d785:~$ sudo iptables -t nat -L KUBE-SVC-ENODL3HWJ5BZY56Q
Chain KUBE-SVC-ENODL3HWJ5BZY56Q (2 references)
target     prot opt source               destination
KUBE-SEP-XOLBHGOCI64CSKKM  all  --  anywhere             anywhere             /* default/frontend */ statistic mode random probability 0.33333333349
KUBE-SEP-VP2QAUKOETMAGR3B  all  --  anywhere             anywhere             /* default/frontend */ statistic mode random probability 0.50000000000
KUBE-SEP-UTYYDRQJ3GWCQAS7  all  --  anywhere             anywhere             /* default/frontend */
iksadmin@iks-networking-example-cl-iks-networ-77d409d785:~$ sudo iptables -t nat -L KUBE-SEP-XOLBHGOCI64CSKKM
Chain KUBE-SEP-XOLBHGOCI64CSKKM (1 references)
target     prot opt source               destination
KUBE-MARK-MASQ  all  --  192.168.43.150       anywhere             /* default/frontend */
DNAT       tcp  --  anywhere             anywhere             /* default/frontend */ tcp to:192.168.43.150:80
iksadmin@iks-networking-example-cl-iks-networ-77d409d785:~$ sudo iptables -t nat -L KUBE-SEP-VP2QAUKOETMAGR3B
Chain KUBE-SEP-VP2QAUKOETMAGR3B (1 references)
target     prot opt source               destination
KUBE-MARK-MASQ  all  --  192.168.8.198        anywhere             /* default/frontend */
DNAT       tcp  --  anywhere             anywhere             /* default/frontend */ tcp to:192.168.8.198:80
iksadmin@iks-networking-example-cl-iks-networ-77d409d785:~$ sudo iptables -t nat -L KUBE-SEP-UTYYDRQJ3GWCQAS7
Chain KUBE-SEP-UTYYDRQJ3GWCQAS7 (1 references)
target     prot opt source               destination
KUBE-MARK-MASQ  all  --  192.168.8.199        anywhere             /* default/frontend */
DNAT       tcp  --  anywhere             anywhere             /* default/frontend */ tcp to:192.168.8.199:80
```

Figure 25.
IPTables rules implemented on worker node 1 for the guestbook frontend service. Three frontend pods exist; therefore, three rules have been created. The pod selected to receive the traffic is chosen at random.

- **KUBE-SERVICES is the entry point for service packets. What it does is to match the destination IP:port and dispatch the packet to the corresponding KUBE-SVC-* chain.**
- **KUBE-SVC-* chain acts as a load balancer, and distributes the packet to KUBE-SEP-* chain equally. Every KUBE-SVC-* has the same number of KUBE-SEP-* chains as the number of endpoints behind it.**
- **KUBE-SEP-* chain represents a Service EndPoint. It simply does DNAT, replacing service IP:port with pod's endpoint IP:Port.**

(Quoted from https://kubernetes.io/blog/2019/03/29/kube-proxy-subtleties-debugging-an-intermittent-connection-reset/.)

DNS Services

Not only does Kubernetes assign each service a ClusterIP address, but DNS records are also automatically configured. IKS will deploy CoreDNS pods to provide internal DNS resolution for your pods and services.

Important point: The DNS server address configured as part of the IKS cluster creation wizard applies to the IKS control plane and worker nodes.

As per the following Kubernetes documentation,

> *"Kubernetes DNS schedules a DNS Pod and Service on the cluster and configures the kubelets to tell individual containers to use the DNS Service's IP to resolve DNS names."*

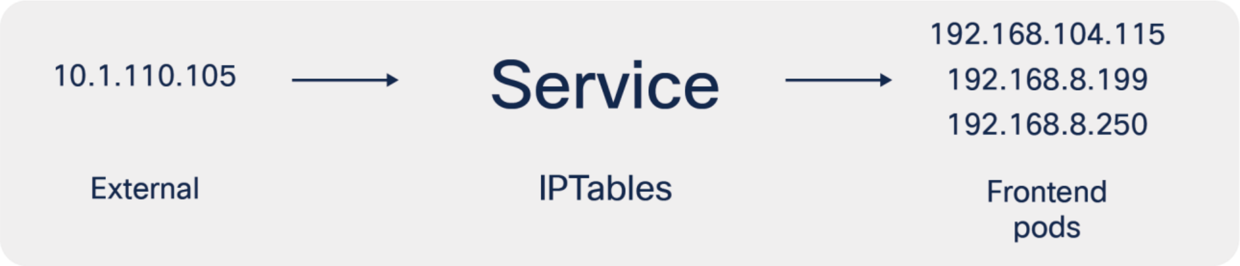
> *"Every Service defined in the cluster . . . is assigned a DNS name. By default, a client Pod's DNS search list will include the Pod's own namespace and the cluster's default domain."*

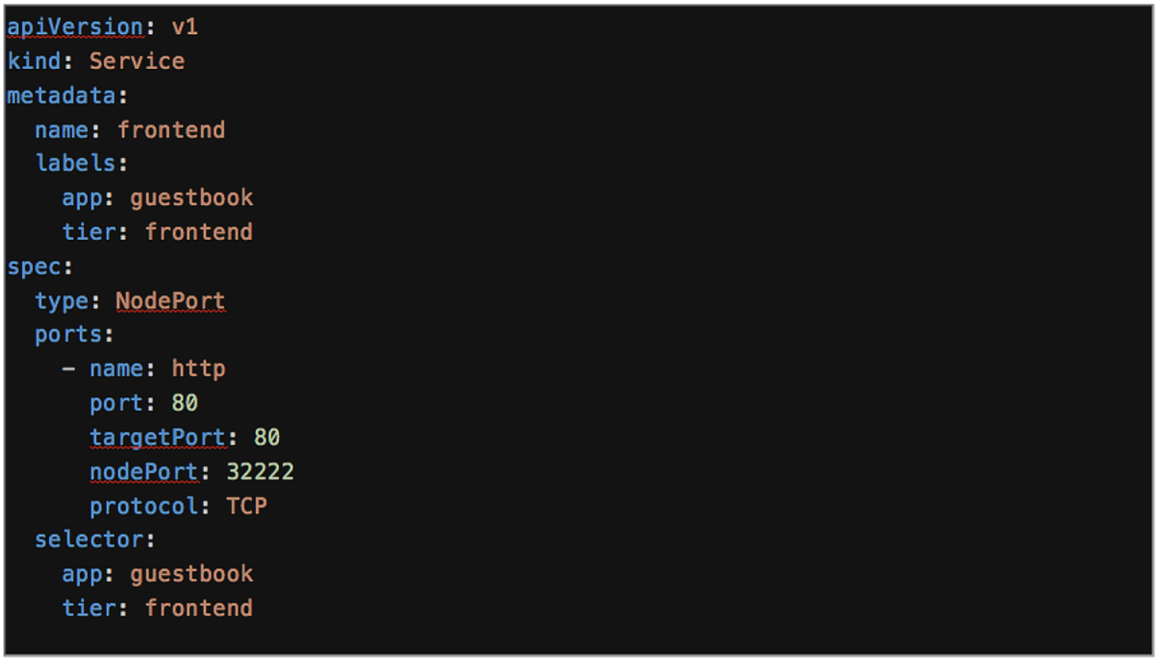
(Quoted from https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/.)

When the backend service is deployed as part of the guestbook application, not only is there an associated ClusterIP address, but there is also a DNS record created, **backend.default.svc.cluster.local**. "Default" in this case being the name of the Kubernetes namespace in which the backend pods run. Since every pod is configured to automatically use Kubernetes DNS, the address above should resolve correctly.

In the guestbook example the frontend pods can reference **backend.default.svc.cluster.local** in the application code. This will resolve to the ClusterIP address for the backend service which is then translated to one of the IP addresses of these pods (192.168.x.x).

Providing external access to the cluster – NodePort service





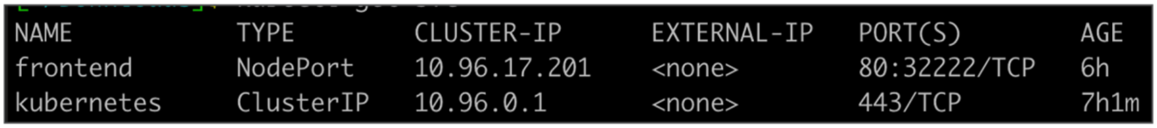
```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  type: NodePort
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 32222
      protocol: TCP
  selector:
    app: guestbook
    tier: frontend
```

Figure 26.
YAML definition for the guestbook frontend service when using a NodePort

The Kubernetes Service configurations include a field, type, which describes the type of service. This can be a **type: clusterIP** (as previously seen), **type: NodePort**, or **type: LoadBalancer**.

The NodePort service is configured by specifying a port (default is between 30000-32767) to which the external traffic is sent. A target port on which the application is listening must also be configured. For example, the guestbook application listens on port 80.

When this service has been configured, an external client can access the pods (for this service) using the IP address of any IKS cluster nodes (externally routable 10.1.110.0 in the example) and the configured NodePort.

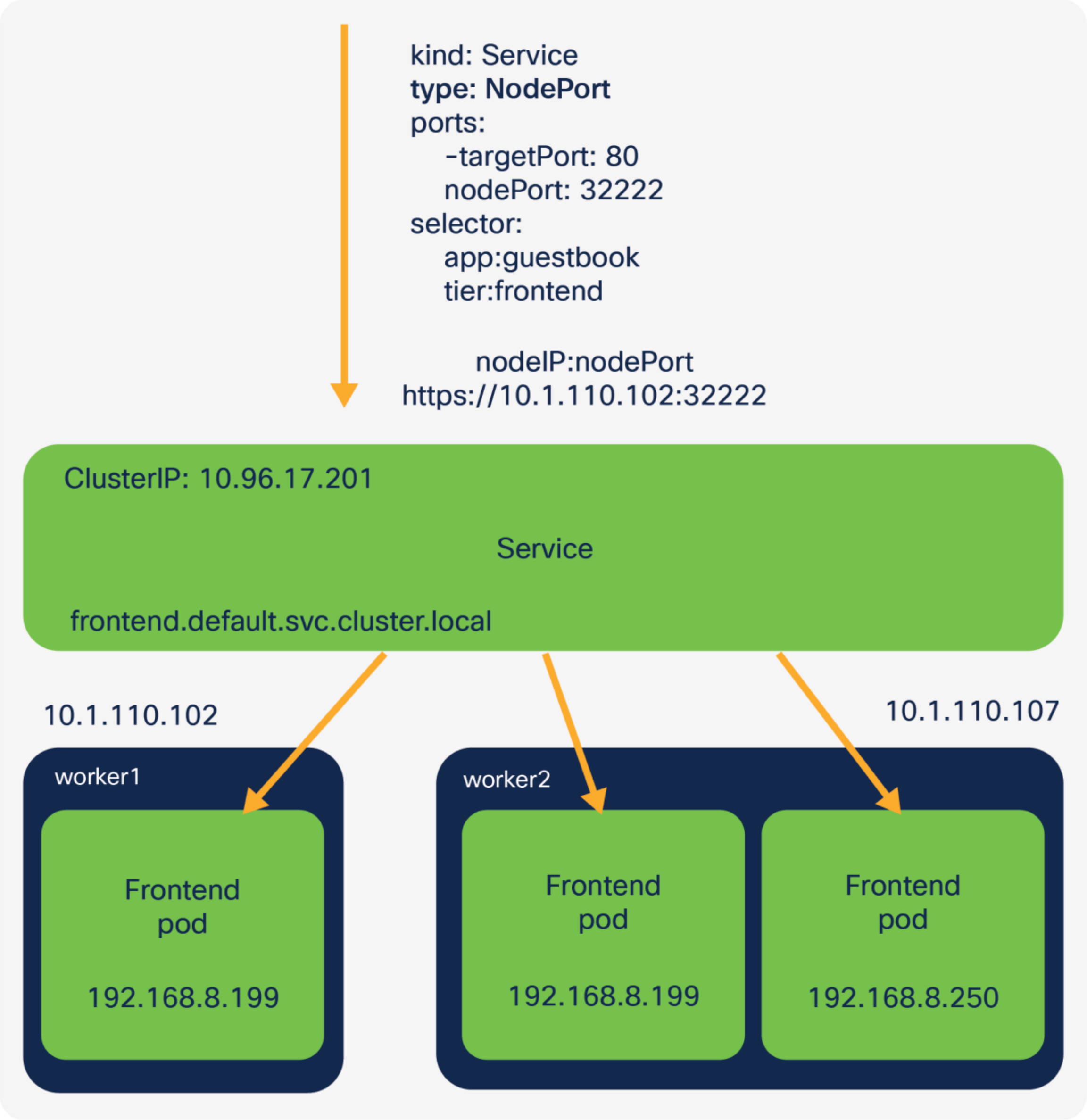


```
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
frontend     NodePort    10.96.17.201   <none>        80:32222/TCP   6h
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP        7h1m
```

Figure 27.
The output from running the **kubectl get services** command

Using the guestbook example, you can use **https://<worker-node-ip>:32222** and have access to the guestbook application through a browser.

Kubernetes will forward this traffic to one of the available pods on the specified target port (in this case, one of the frontend pods, port 80).

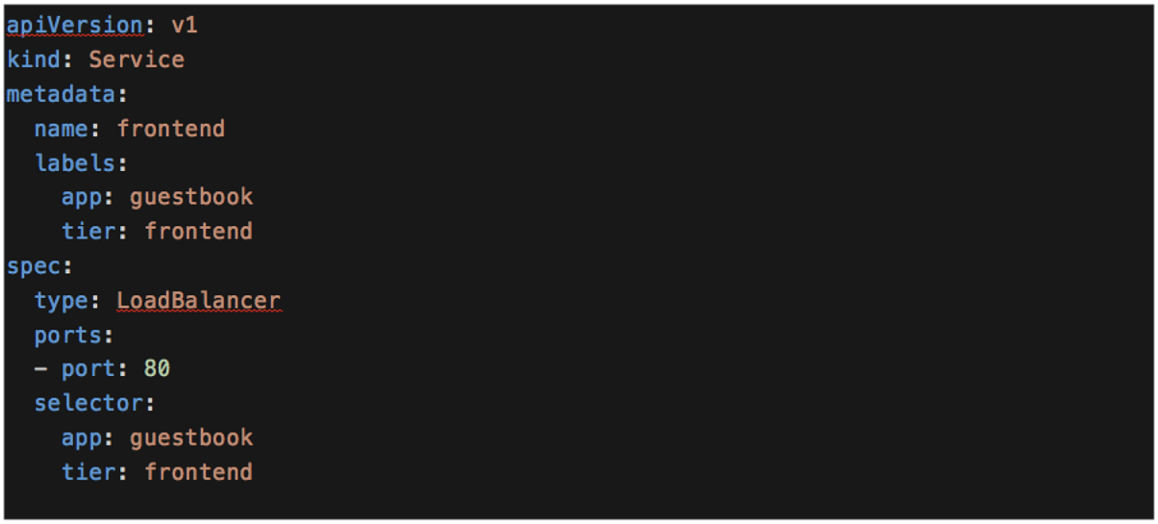


```
kind: Service
type: NodePort
ports:
   -targetPort: 80
   nodePort: 32222
selector:
   app:guestbook
   tier:frontend
```

Figure 28.
Accessing the application through the IP address of a node and the configured NodePort

Under the hood, Kubernetes has configured IPTables rules to translate the traffic from the worker node IP address: NodePort to the destination pod IP address:port.

Providing external access to the cluster - LoadBalancer service



```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```

Figure 29.
YAML definition for the guestbook frontend service when using a LoadBalancer

The Kubernetes LoadBalancer service exposes pods externally using either a public cloud provider or an on-premises load balancer.

(Information from https://kubernetes.io/docs/concepts/services-networking/service/.)

MetalLB is automatically deployed into each IKS cluster and provides L3/L4 load balancing services.

As per the following document,

"MetalLB is a load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols."

(Information from https://metallb.universe.tf/.)

The LoadBalancer service relies upon an address selected from a pool that has been configured. This was the load balancer count field in the IKS cluster creation wizard.

When a Kubernetes LoadBalancer Service is configured, MetalLB will allocate the next available IP address from the pool of addresses provided. Any traffic destined to the IP is handled by MetalLB and forwarded onto the correct pods.

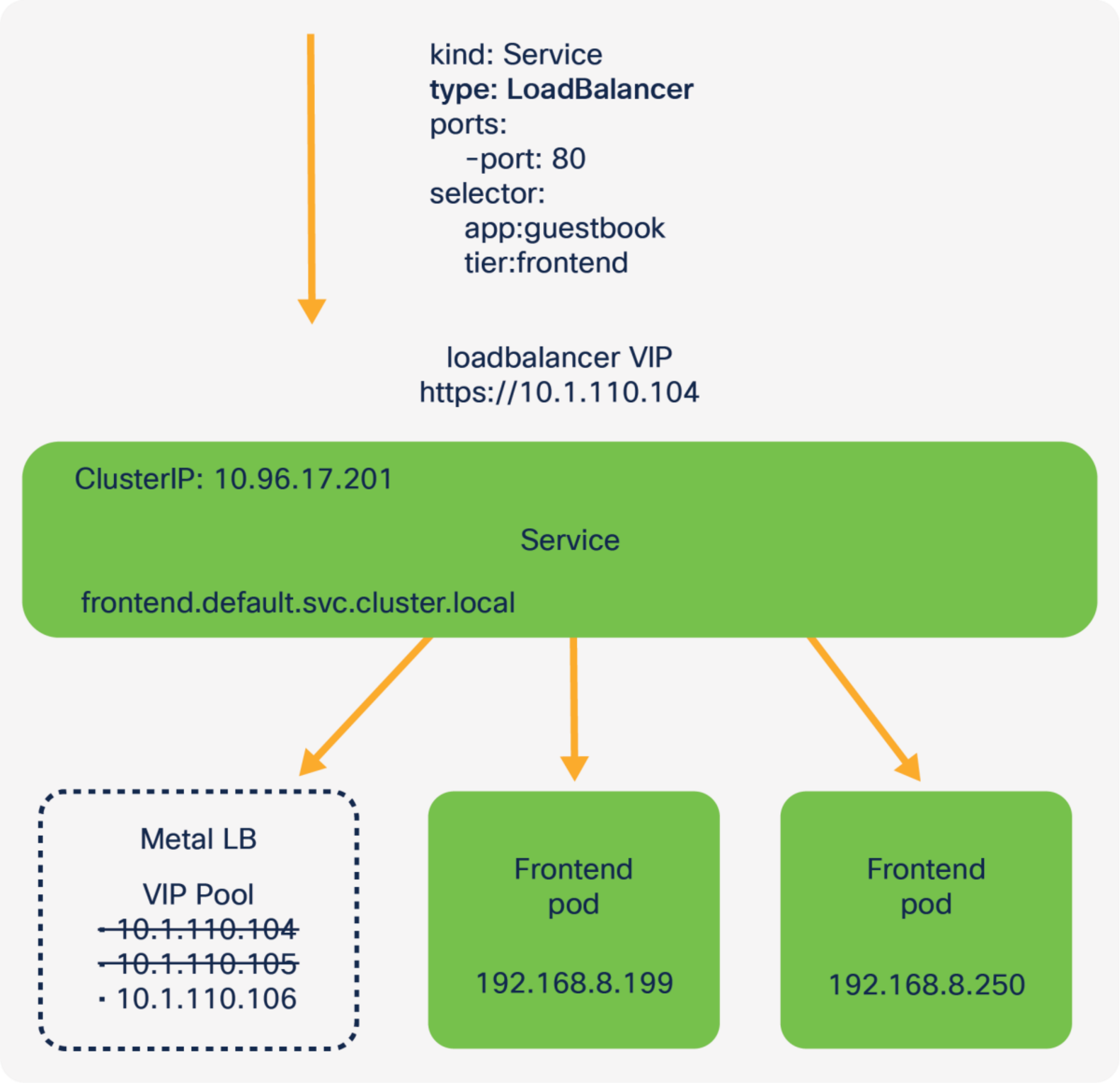


Figure 30.
Accessing the application through an IP address assigned by the MetalLB LoadBalancer

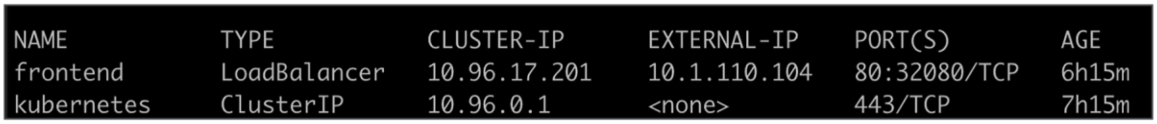
IKS uses MetalLB in Layer-2 mode (ARP/NDP).

As per the following document,

"Under the hood, MetalLB responds to ARP requests for IPv4 services, and NDP requests for IPv6. In layer 2 mode, all traffic for a service IP goes to one node. From there, kube-proxy spreads the traffic to all the service's pods."

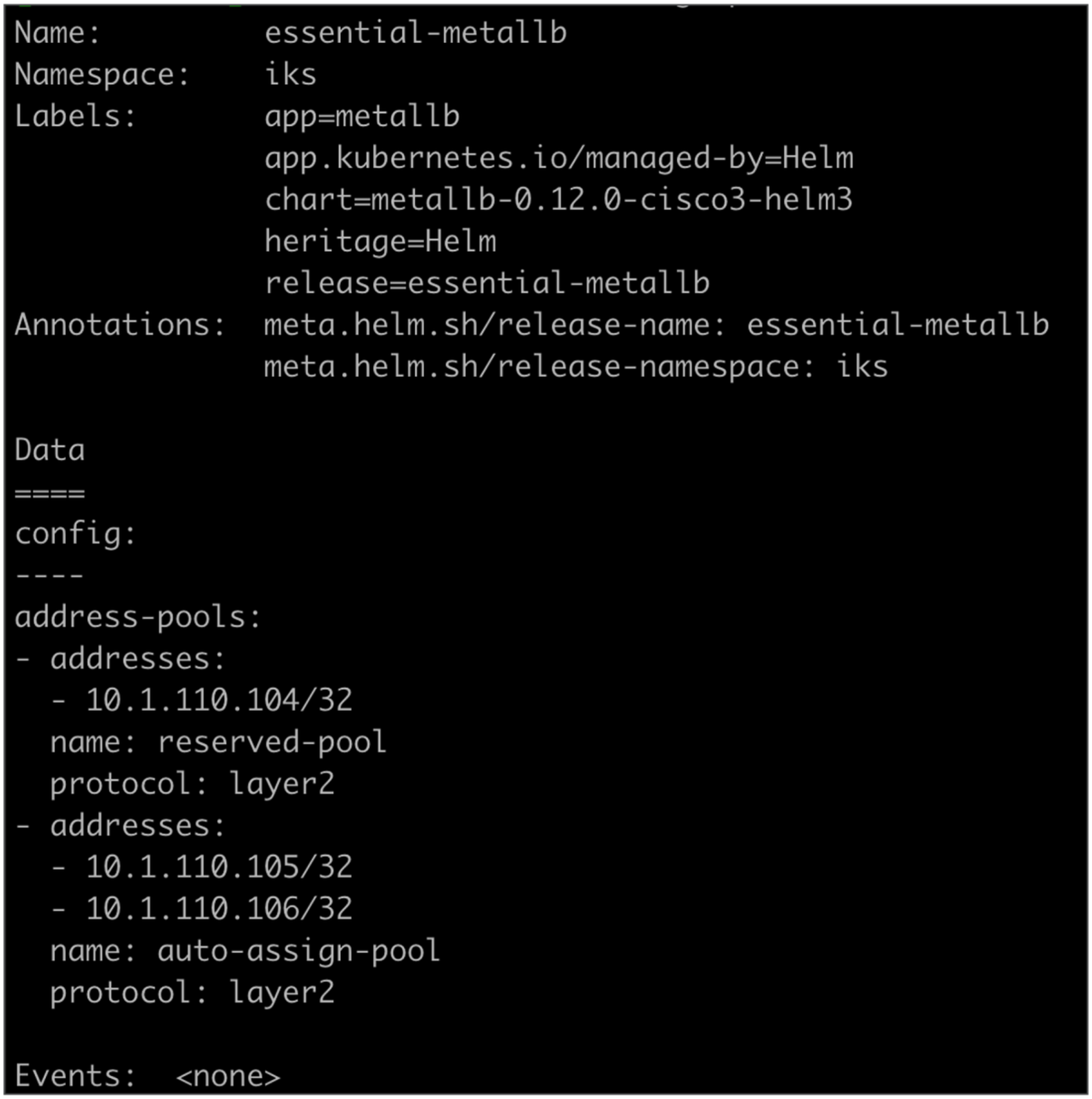
(Quoted from https://metallb.universe.tf/concepts/layer2/.)

You can verify that MetalLB is assigning IPs correctly by looking at the logs of the MetalLB pods.



```
NAME         TYPE          CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
frontend     LoadBalancer  10.96.17.201   10.1.110.104   80:32080/TCP   6h15m
kubernetes   ClusterIP     10.96.0.1      <none>         443/TCP        7h15m
```

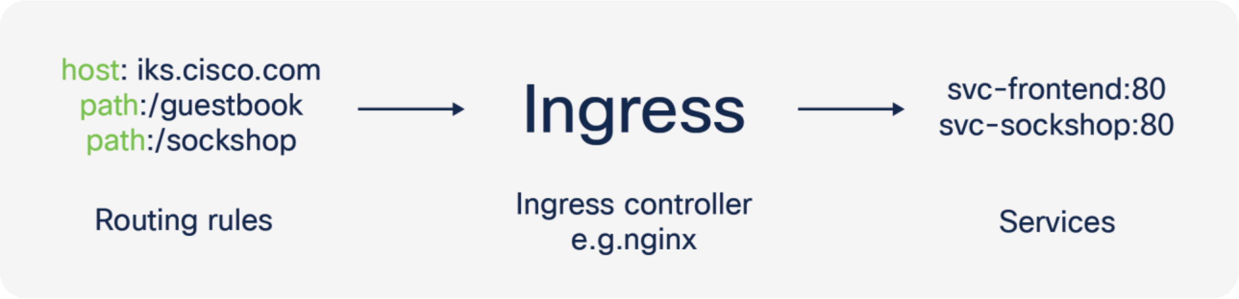
Figure 31.
Output of the **kubectl get services** command showing the frontend service with a Loadbalancer external IP



```
Name:         essential-metallb
Namespace:    iks
Labels:       app=metallb
              app.kubernetes.io/managed-by=Helm
              chart=metallb-0.12.0-cisco3-helm3
              heritage=Helm
              release=essential-metallb
Annotations:  meta.helm.sh/release-name: essential-metallb
              meta.helm.sh/release-namespace: iks

Data
====
config:
----
address-pools:
- addresses:
  - 10.1.110.104/32
  name: reserved-pool
  protocol: layer2
- addresses:
  - 10.1.110.105/32
  - 10.1.110.106/32
  name: auto-assign-pool
  protocol: layer2

Events:  <none>
```

Figure 32.
Output of the **kubectl describe configmap essential-metallb -n iks** command showing the configuration of MetalLB and available addresses

Kubernetes Ingress: rule-based routing



host: iks.cisco.com
path:/guestbook
path:/sockshop

Routing rules

Ingress

Ingress controller
e.g.nginx

svc-frontend:80
svc-sockshop:80

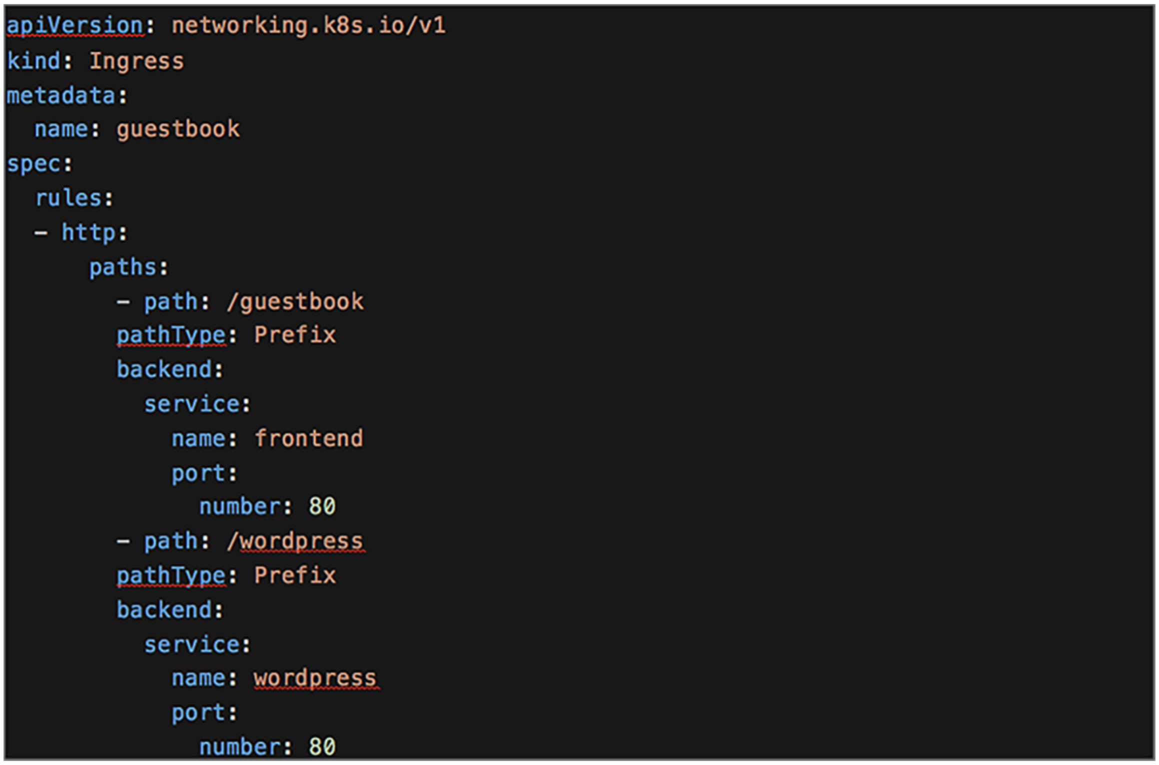
Services

A Kubernetes Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

There are a number of benefits to using an ingress:

- Centralized SSL termination
- Rule-based routing
- Reduction in IP address usage

There are a number of ways to configure a Kubernetes Ingress. A fanout is used for this example. A fanout configuration routes traffic from a single IP address to more than one service.



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: guestbook
spec:
  rules:
  - http:
      paths:
        - path: /guestbook
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
        - path: /wordpress
        pathType: Prefix
        backend:
          service:
            name: wordpress
            port:
              number: 80
```

Figure 33.
The YAML definition for a Kubernetes Ingress

Note in the YAML file above the set of rules defining two HTTP paths, one to a guestbook application and one to a different application called Wordpress.

Kubernetes Ingress controller

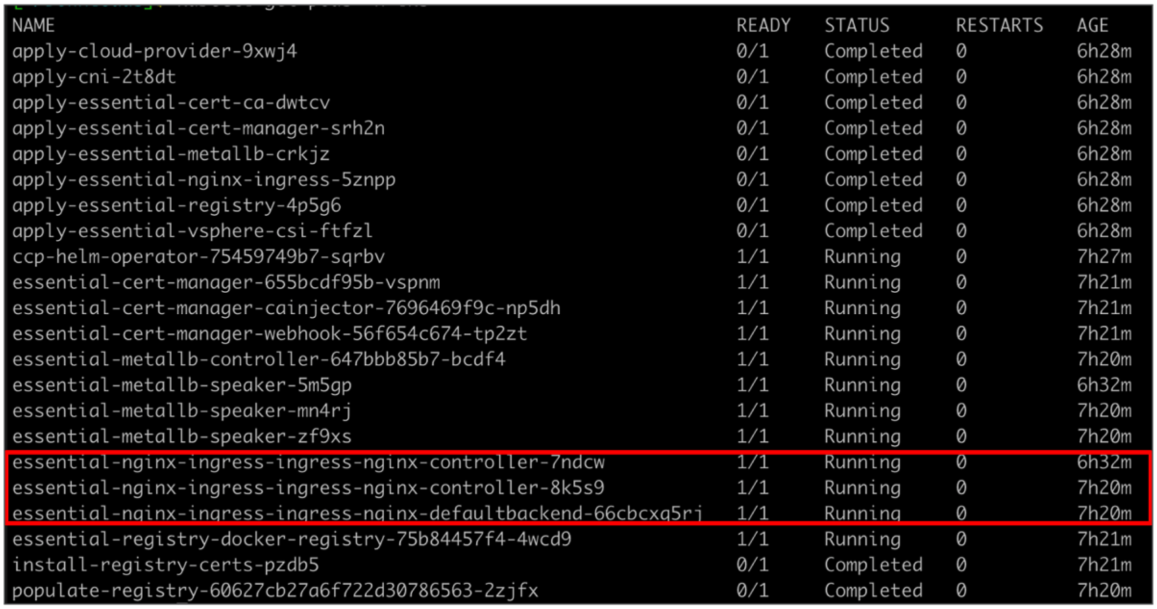
A Kubernetes Ingress itself does not provide the rule-based routing. Instead it relies on an ingress controller to perform this function.

There are many ingress controller options available. IKS automatically deploys an NGINX ingress controller to each Kubernetes cluster.

(Information from https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/.)

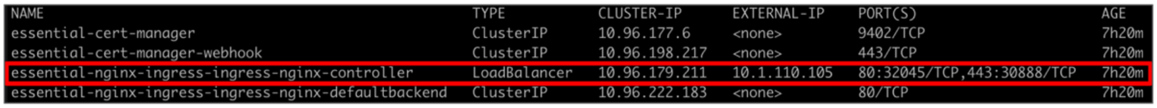
On each IKS node you should see an **nginx-ingress-controller-xxxxx** pod running.

IKS will automatically create a LoadBalancer Service to direct external traffic to the **nginx-ingress-controller-xxxxx** pods. Hence the reason for requiring at least one IP address in the load-balancer count field (cluster creation wizard).



```
NAME                                                      READY   STATUS      RESTARTS   AGE
apply-cloud-provider-9xwj4                                0/1     Completed   0          6h28m
apply-cni-2t8dt                                           0/1     Completed   0          6h28m
apply-essential-cert-ca-dwtcv                             0/1     Completed   0          6h28m
apply-essential-cert-manager-srh2n                        0/1     Completed   0          6h28m
apply-essential-metallb-crkjz                             0/1     Completed   0          6h28m
apply-essential-nginx-ingress-5znpp                       0/1     Completed   0          6h28m
apply-essential-registry-4p5g6                            0/1     Completed   0          6h28m
apply-essential-vsphere-csi-ftfzl                         0/1     Completed   0          6h28m
ccp-helm-operator-75459749b7-sqrbv                        1/1     Running     0          7h27m
essential-cert-manager-655bcdf95b-vspnm                   1/1     Running     0          7h21m
essential-cert-manager-cainjector-7696469f9c-np5dh        1/1     Running     0          7h21m
essential-cert-manager-webhook-56f654c674-tp2zt           1/1     Running     0          7h21m
essential-metallb-controller-647bbb85b7-bcdf4             1/1     Running     0          7h20m
essential-metallb-speaker-5m5gp                           1/1     Running     0          6h32m
essential-metallb-speaker-mn4rj                           1/1     Running     0          7h20m
essential-metallb-speaker-zf9xs                           1/1     Running     0          7h20m
essential-nginx-ingress-ingress-nginx-controller-7ndcw    1/1     Running     0          6h32m
essential-nginx-ingress-ingress-nginx-controller-8k5s9    1/1     Running     0          7h20m
essential-nginx-ingress-ingress-nginx-defaultbackend-66cbcxq5rj  1/1     Running     0          7h20m
essential-registry-docker-registry-75b84457f4-4wcd9       1/1     Running     0          7h21m
install-registry-certs-pzdb5                              0/1     Completed   0          7h21m
populate-registry-60627cb27a6f722d30786563-2zjfx          0/1     Completed   0          7h20m
```

Figure 34.
The output of the **kubectl get pods -n iks** command showing the NGINX ingress controllers deployed on each IKS node



```
NAME                                                   TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)                      AGE
essential-cert-manager                                 ClusterIP      10.96.177.6     <none>         9402/TCP                     7h20m
essential-cert-manager-webhook                         ClusterIP      10.96.198.217   <none>         443/TCP                      7h20m
essential-nginx-ingress-ingress-nginx-controller       LoadBalancer   10.96.179.211   10.1.110.105   80:32045/TCP,443:30888/TCP   7h20m
essential-nginx-ingress-ingress-nginx-defaultbackend   ClusterIP      10.96.222.183   <none>         80/TCP                       7h20m
```

Figure 35.
The output of the **kubectl get services -n iks** command showing the NGINX ingress controller service (LoadBalancer)

Similar to how MetalLB works for Kubernetes Services, the NGINX controller will look for any changes to the Kubernetes Ingress definition. When a new ingress is configured, the NGINX configuration (**nginx.conf** in the **nginx-ingress-controller-xxxxx** pods) is updated with the new routing rules added to the ingress YAML file.

Each ingress controller also has options to provide annotations for custom configuration of the specific controller. For example, you can find the available NGINX annotations at the following link: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/

Since the ingress controller is running in multiple pods, the LoadBalancer service provides direct external traffic to one of the available NGINX controller pods.

From there the NGINX controller will redirect based on the path, either to the guestbook frontend service or the Wordpress service. The services (IPTables rules) will in turn forward the traffic onto an available pod managed by the respective service.

Use cases for a Kubernetes Ingress

Additional to the routing rules previously described, a Kubernetes Ingress helps conserve IP addresses. When a service of type LoadBalancer is used, an externally routable address for each service configured must be assigned. While the addresses may be available on premises, in a public cloud environment there can often be a cost associated to each external IP address.

When using an ingress, a single external IP address can be assigned (for the ingress service). Each service behind the ingress can then use a ClusterIP. In this scenario the services are only accessible through the ingress and therefore don't require a public IP address.

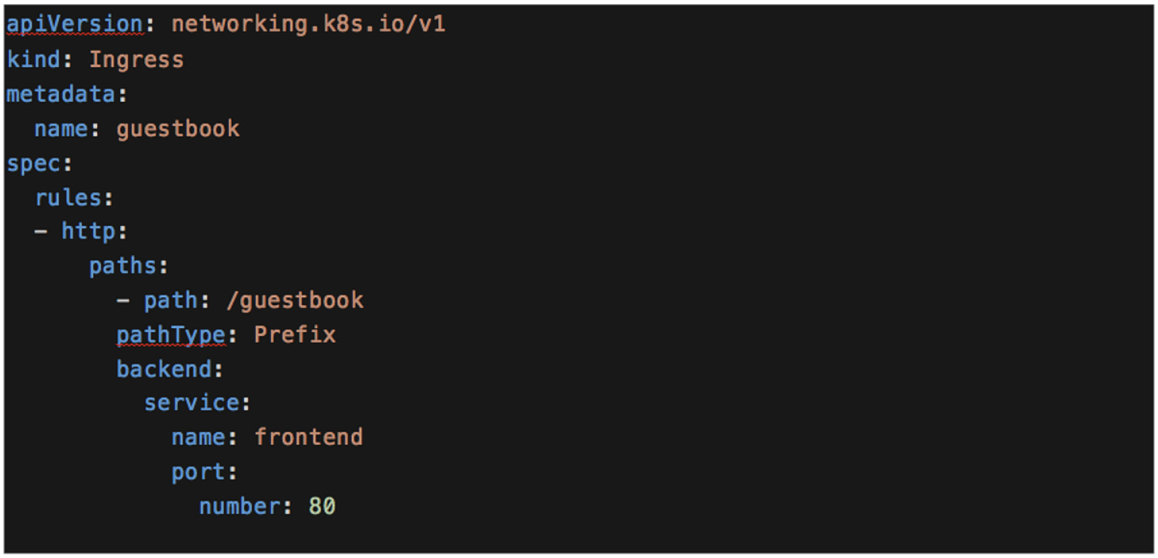
Kubernetes Ingress also provides a single ingress point for which all routing rules and TLS termination can be configured.

Ingress configuration when serving assets

Depending on the applications you're deploying, you may run into some issues while serving content through a Kubernetes Ingress. For example, serving a webpage along with assets such as images, JS, and CSS files.

The following example will use the Kubernetes guestbook application (there is also a service named Wordpress configured in the ingress examples below; however, only the guestbook is used).

Here is the Kubernetes Ingress definition. In this example the filename is **ingress.yaml**.



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: guestbook
spec:
  rules:
  - http:
      paths:
        - path: /guestbook
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
```

Figure 36.
The YAML definition for a Kubernetes Ingress

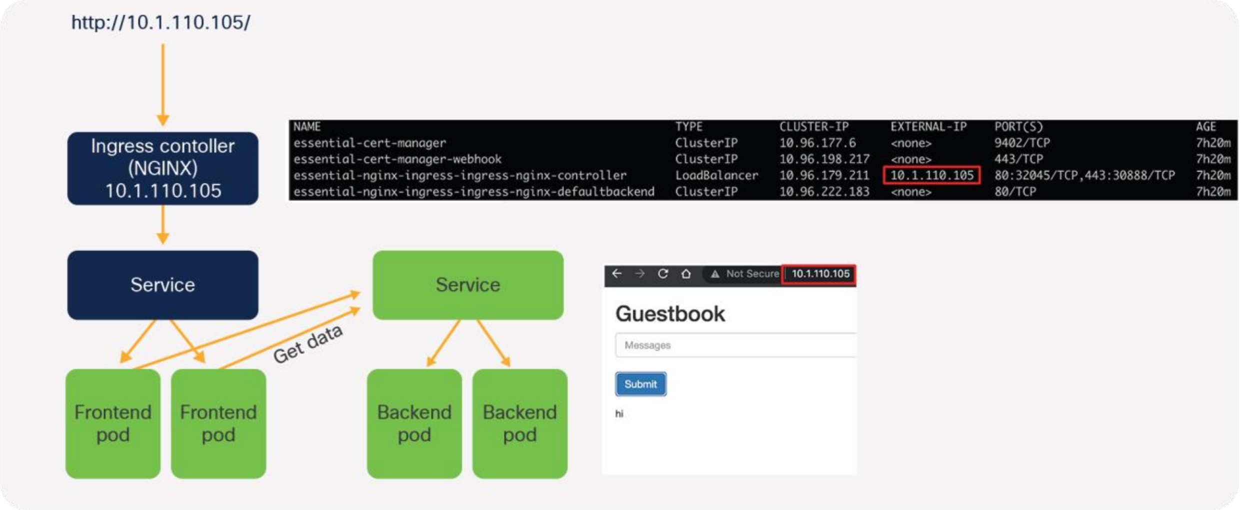
The application is deployed by applying both files.

kubectl apply -f guestbook-all-in-one.yaml

kubectl apply -f ingress.yaml

Once running, the guestbook application should be available using the IP address of the ingress controller.

kubectl -n iks get svc essential-nginx-ingress-ingress-nginx-controller -o jsonpath='{.status.loadBalancer.ingress[0].ip}



```
NAME                                              TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)                      AGE
essential-cert-manager                            ClusterIP      10.96.177.6     <none>         9402/TCP                     7h20m
essential-cert-manager-webhook                    ClusterIP      10.96.198.217   <none>         443/TCP                      7h20m
essential-nginx-ingress-ingress-nginx-controller  LoadBalancer   10.96.179.211   10.1.110.105   80:32045/TCP,443:30888/TCP   7h20m
essential-nginx-ingress-ingress-nginx-defaultbackend ClusterIP   10.96.222.183   <none>         80/TCP                       7h20m
```

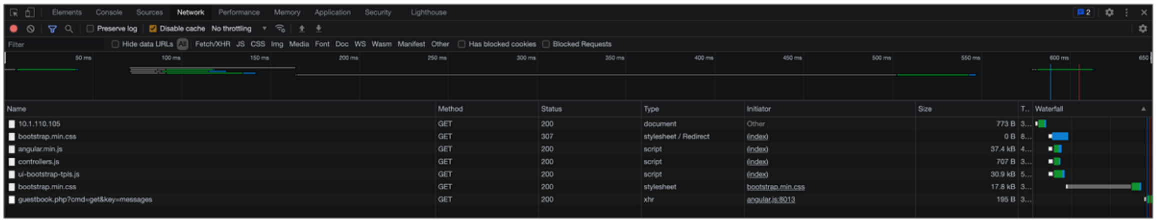
Figure 37.
The guestbook application when working correctly

You can view the files accessed by the guestbook by looking at the developer tools in your browser.

Opening the developer tools console in a browser

Chrome: https://developer.chrome.com/docs/devtools/open/

Firefox: https://developer.mozilla.org/en-US/docs/Tools



Name	Method	Status	Type	Initiator	Size	T...	Waterfall
10.1.110.105	GET	200	document	Other	773 B	3...	
bootstrap.min.css	GET	307	stylesheet / Redirect	(index)	0 B	8...	
angular.min.js	GET	200	script	(index)	37.4 kB	4...	
controllers.js	GET	200	script	(index)	707 B	3...	
ui-bootstrap-tpls.js	GET	200	script	(index)	30.9 kB	5...	
bootstrap.min.css	GET	200	stylesheet	bootstrap.min.css	17.8 kB	3...	
guestbook.php?cmd=get&key=messages	GET	200	xhr	angular.js:8013	195 B	3...	

Figure 38.
The developer tools window when the guestbook application is working correctly

In the example, the **path: /** configuration in the ingress YAML file acts as a wildcard, therefore matching all assets that are required (**index.html, controllers.js, guestbook.php**).

Important point: In Kubernetes 1.18 there are ingress enhancements (exact and prefix keywords) to provide more granular control.

(Information from https://kubernetes.io/blog/2020/04/02/improvements-to-the-ingress-api-in-kubernetes-1.18/.)

Since there's a match, the controller will forward these to the associated service, frontend.

The frontend webserver, Apache, has been configured as default and is serving content from the root directory,/, so returns the requested files.

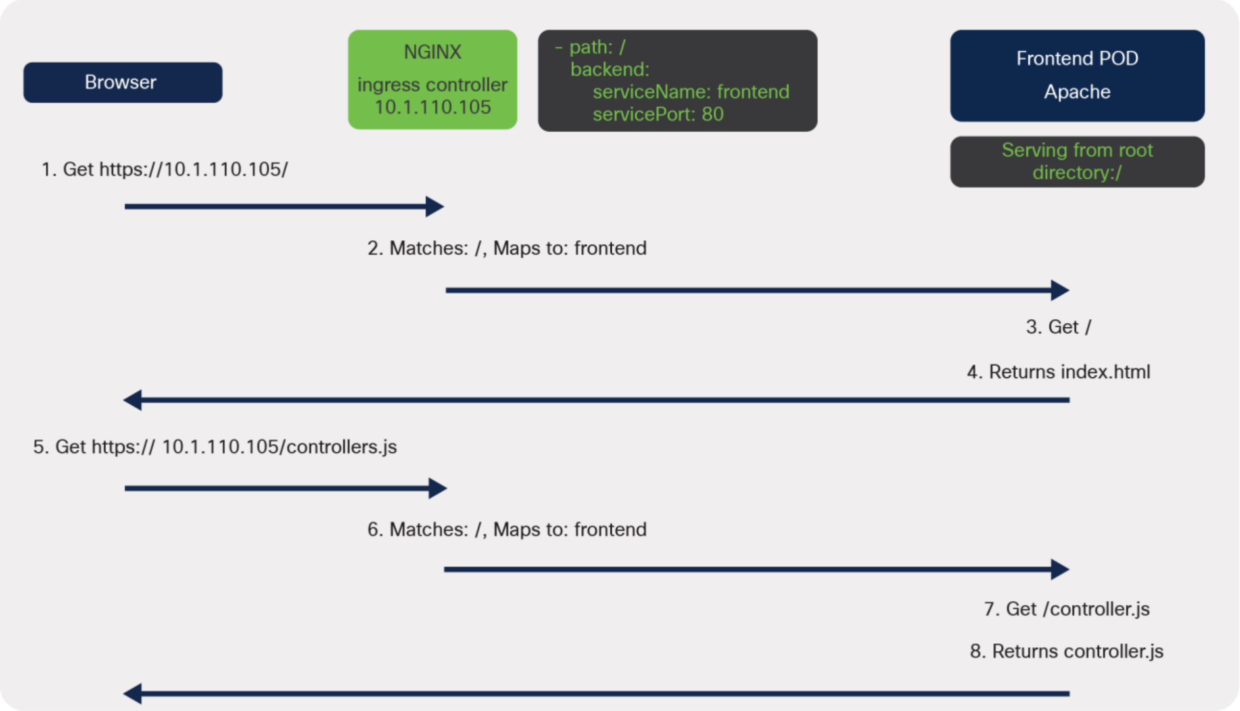
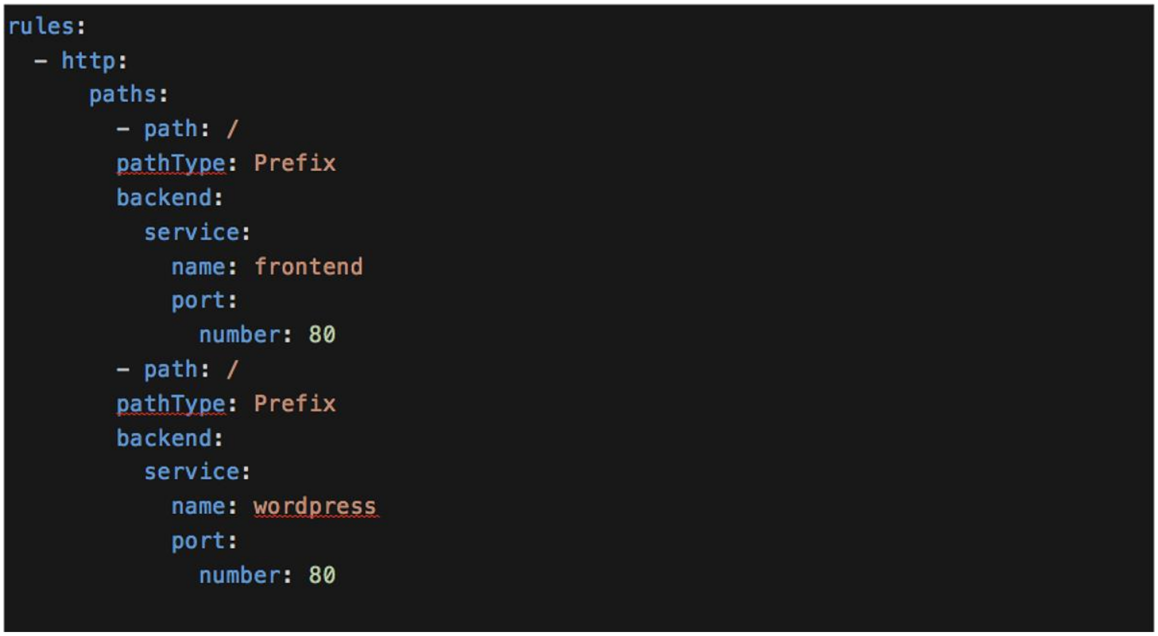


Figure 39.
The connection flow when working correctly

Single ingress for multiple applications

You may run into issues when trying to use the same root path when deploying multiple applications on the same ingress.

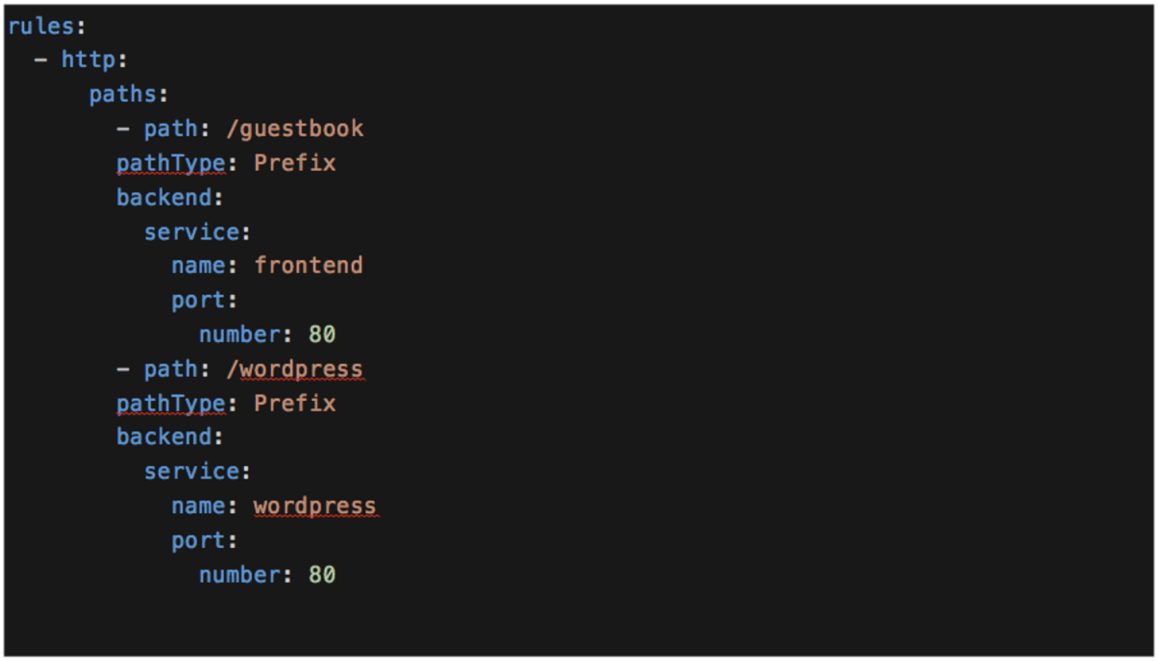
In the following example, if the same path points to two different services there is no way to determine which one is correct.



```
rules:
  - http:
      paths:
        - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
        - path: /
        pathType: Prefix
        backend:
          service:
            name: wordpress
            port:
              number: 80
```

Figure 40.
The YAML definition for a Kubernetes Ingress with the same paths

To overcome this, each service can be accessed from a unique path.



```
rules:
  - http:
      paths:
        - path: /guestbook
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
        - path: /wordpress
        pathType: Prefix
        backend:
          service:
            name: wordpress
            port:
              number: 80
```

Figure 41.
The YAML definition for a Kubernetes Ingress with differing paths

Now that a path has been configured, the application will need to be accessed via http://<ingress-controller-ip>/guestbook.

A 404 Page Not Found error should appear.

When traffic reaches the ingress controller, it matches **path: /guestbook** and is sent to the web server. The browser is trying to access files located in the **/guestbook** subdirectory; however, the web server is serving content from the root directory, /.

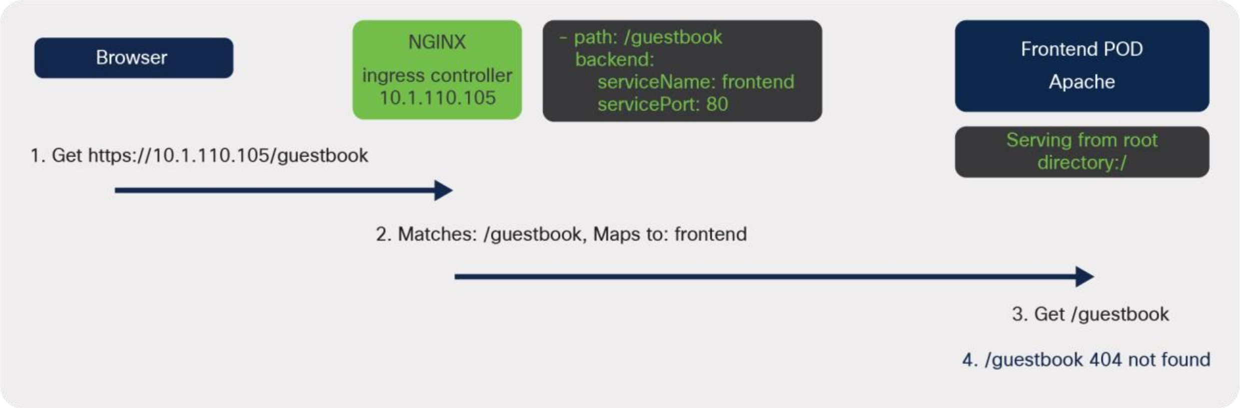
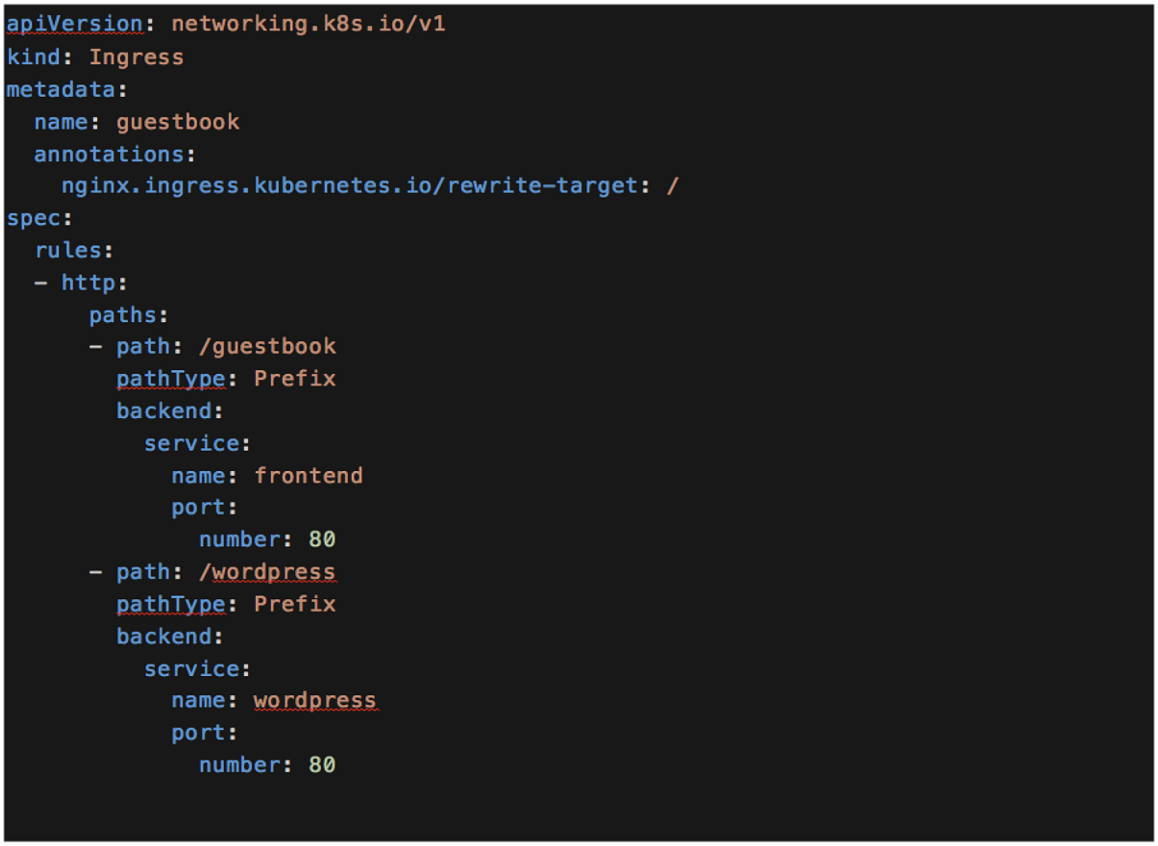


Figure 42.
A diagram of the connectivity when serving content from the root directory

Rewriting the location



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: guestbook
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /guestbook
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
      - path: /wordpress
        pathType: Prefix
        backend:
          service:
            name: wordpress
            port:
              number: 80
```

Figure 43.
YAML definition for an updated Ingress that includes the rewrite target

Kubernetes Ingress annotations can be used to overcome this initial issue. Specifically, for IKS use the **NGINX Ingress rewrite-target** annotation.

(Information from https://kubernetes.github.io/ingress-nginx/examples/rewrite/.)

The annotations in the Kubernetes Ingress allow you to include custom configurations required for an ingress controller environment. In this example, a custom NGINX configuration is added to the **nginx.conf** file on the ingress controller pods automatically.

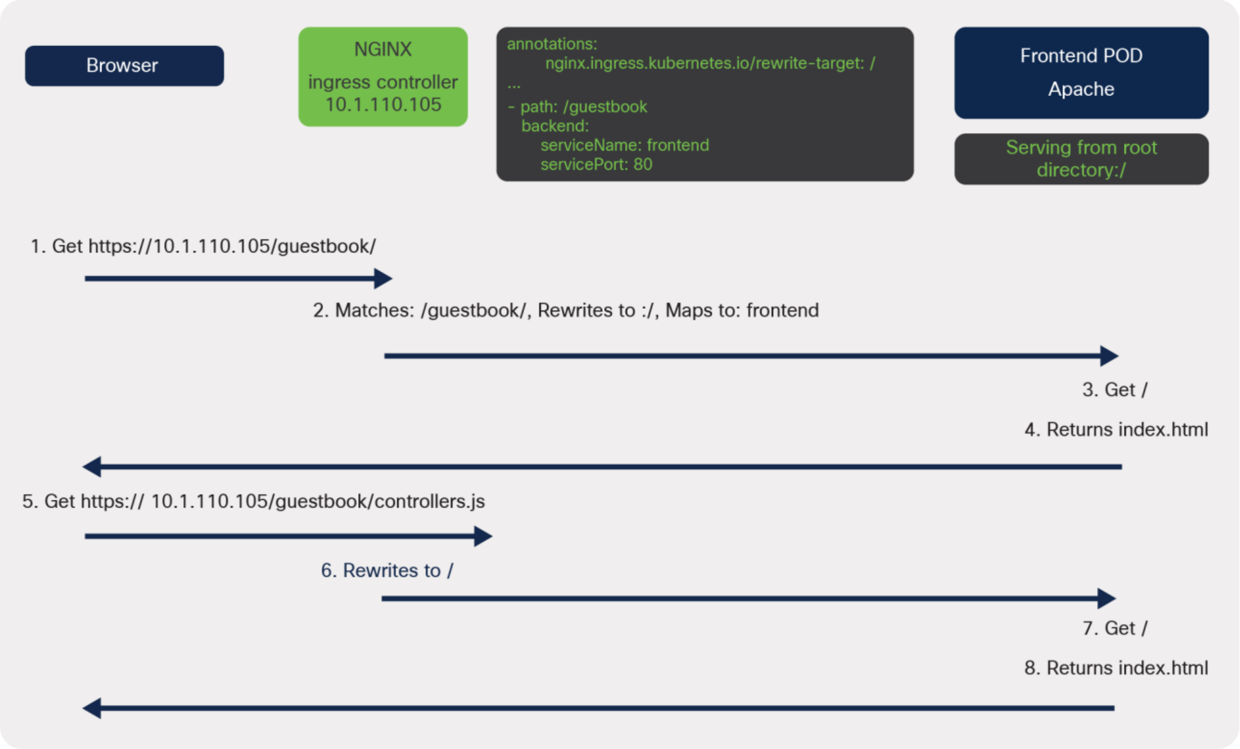
Important Point: The available annotations may differ depending on which ingress controller is used.

Using the rewrite-target annotation specifies the target URI where the traffic must be redirected.

In the example, the path, **/guestbook**, is matched. NGINX will then rewrite it to / before sending it to the web server. As it is now rewritten to the root directory, /, the **index.html** file is returned correctly, as was the case in the first scenario.

However, a problem still exists.

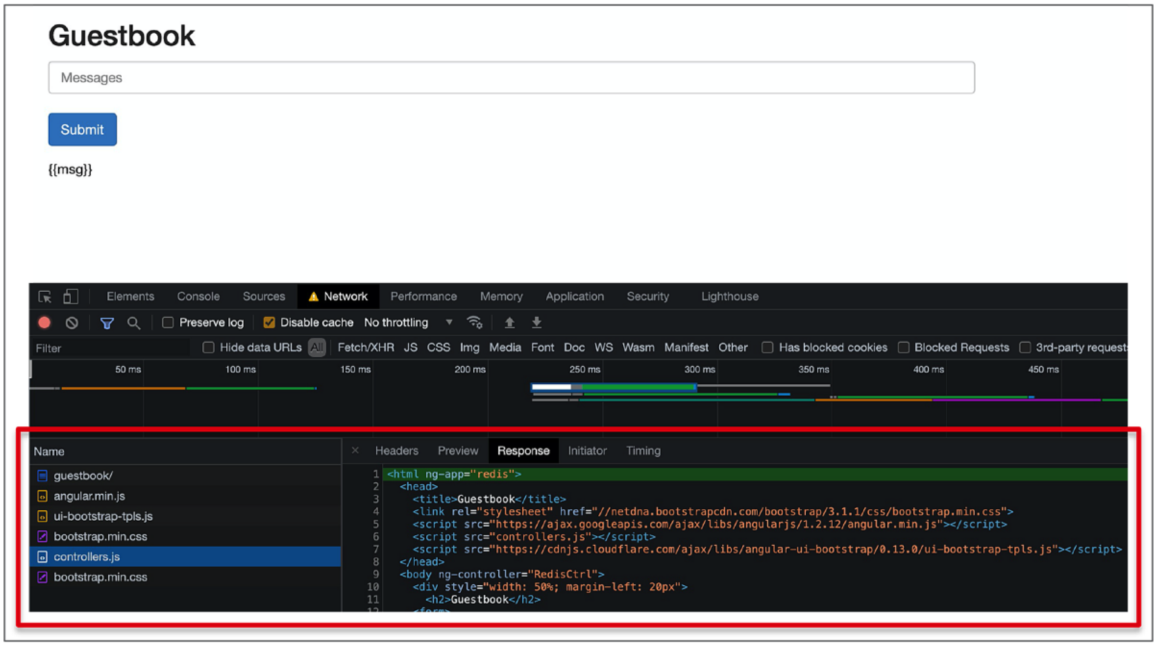
Within the **index.html** file is a reference to a few assets (CSS and JS files) that are used to build the guestbook. When the page loads it tries to download the required files, one of those being **controller.js**. Because it is rewriting to /, the index.html page is returned a second time.



```
annotations:
      nginx.ingress.kubernetes.io/rewrite-target: /
...
- path: /guestbook
   backend:
      serviceName: frontend
      servicePort: 80
```

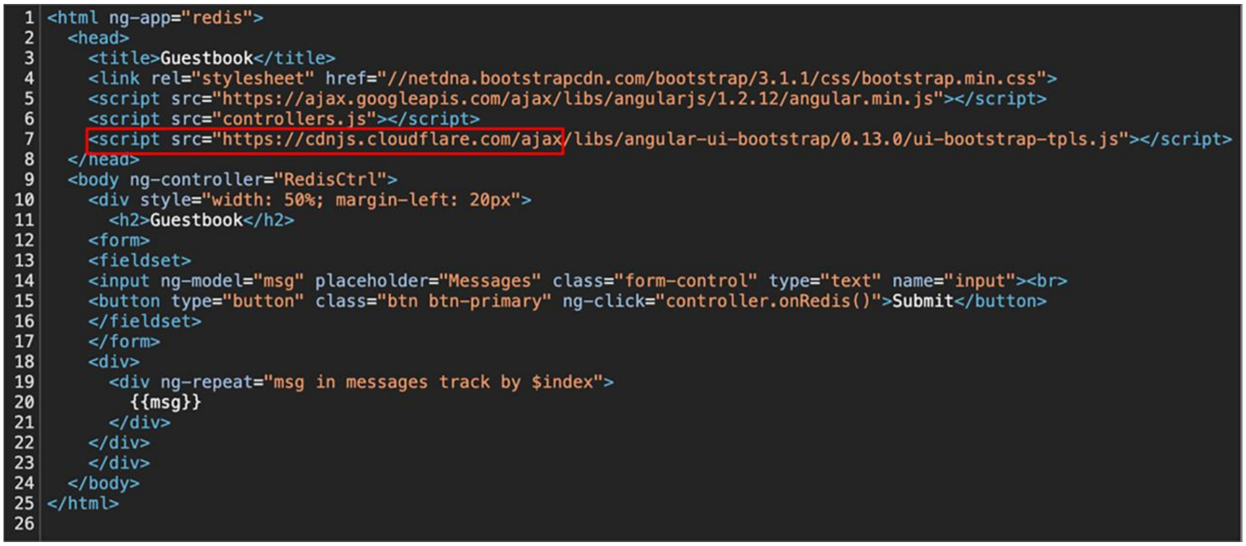
Browser
NGINX
ingress controller
10.1.110.105
Frontend POD
Apache
Serving from root directory:/
1. Get https://10.1.110.105/guestbook/
2. Matches: /guestbook/, Rewrites to :/, Maps to: frontend
3. Get /
4. Returns index.html
5. Get https:// 10.1.110.105/guestbook/controllers.js
6. Rewrites to /
7. Get /
8. Returns index.html

Figure 44.
A diagram of the connectivity when serving content from the root directory and rewriting the path at the ingress



```
<html ng-app="redis">
  <head>
    <title>Guestbook</title>
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.12/angular.min.js"></script>
    <script src="controllers.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-bootstrap/0.13.0/ui-bootstrap-tpls.js"></script>
  </head>
  <body ng-controller="RedisCtrl">
    <div style="width: 50%; margin-left: 20px">
      <h2>Guestbook</h2>
```

Figure 45.
Output from the developer console showing the response returned



```
<html ng-app="redis">
  <head>
    <title>Guestbook</title>
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.12/angular.min.js"></script>
    <script src="controllers.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-bootstrap/0.13.0/ui-bootstrap-tpls.js"></script>
  </head>
  <body ng-controller="RedisCtrl">
    <div style="width: 50%; margin-left: 20px">
      <h2>Guestbook</h2>
    <form>
    <fieldset>
    <input ng-model="msg" placeholder="Messages" class="form-control" type="text" name="input"><br>
    <button type="button" class="btn btn-primary" ng-click="controller.onRedis()">Submit</button>
    </fieldset>
    </form>
    <div>
      <div ng-repeat="msg in messages track by $index">
        {{msg}}
      </div>
    </div>
    </div>
  </body>
</html>
```

Figure 46.
Source code for the guestbook **index.html** file

Here are some options to solve this problem.

Option 1: Serve this content from a different location

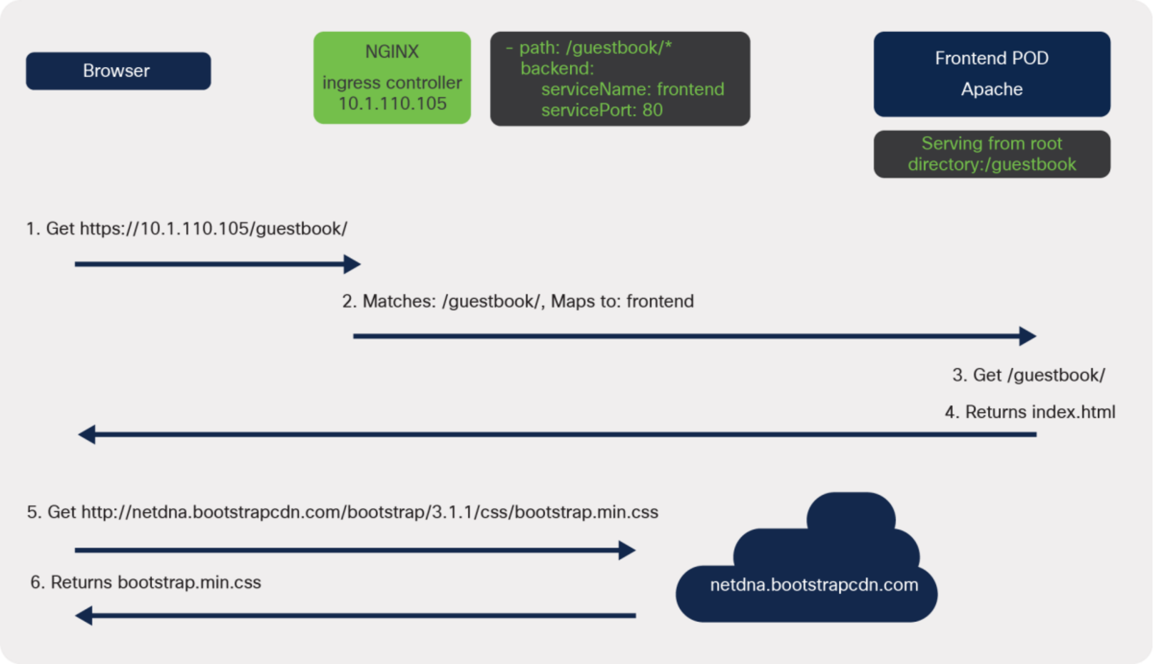

Browser
NGINX
ingress controller
10.1.110.105
- path: /guestbook/*
backend:
serviceName: frontend
servicePort: 80
Frontend POD
Apache
Serving from root directory:/guestbook
1. Get https://10.1.110.105/guestbook/
2. Matches: /guestbook/, Maps to: frontend
3. Get /guestbook/
4. Returns index.html
5. Get http://netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css
6. Returns bootstrap.min.css
netdna.bootstrapcdn.com

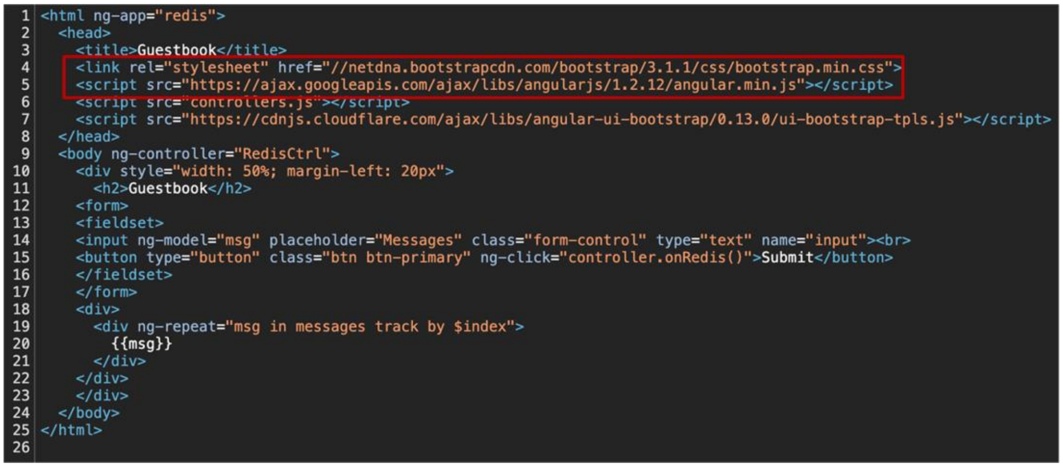
Figure 47.
Communication flow when serving content from a different location

The first option is to separate the static content from the main HTML files and host them externally.

An example of this in the guestbook application is the **bootstrap.min.css** file. Bootstrap is a popular frontend open-source toolkit, is hosted on a Content Delivery Network (CDN), and is available for anyone to use.

When the page loads, the bootstrap CSS file is downloaded directly from the CDN. This means the ingress or web server rules are not used to serve this file.

A drawback of this approach is having to manage the files that are hosted somewhere else.



```
<html ng-app="redis">
  <head>
    <title>Guestbook</title>
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.12/angular.min.js"></script>
    <script src="controllers.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-bootstrap/0.13.0/ui-bootstrap-tpls.js"></script>
  </head>
  <body ng-controller="RedisCtrl">
    <div style="width: 50%; margin-left: 20px">
      <h2>Guestbook</h2>
    <form>
    <fieldset>
    <input ng-model="msg" placeholder="Messages" class="form-control" type="text" name="input"><br>
    <button type="button" class="btn btn-primary" ng-click="controller.onRedis()">Submit</button>
    </fieldset>
    </form>
    <div>
      <div ng-repeat="msg in messages track by $index">
        {{msg}}
      </div>
    </div>
    </div>
  </body>
</html>
```

Figure 48.
Source code for the guestbook **index.html** file showing the use of a CDN for the bootstrap CSS file

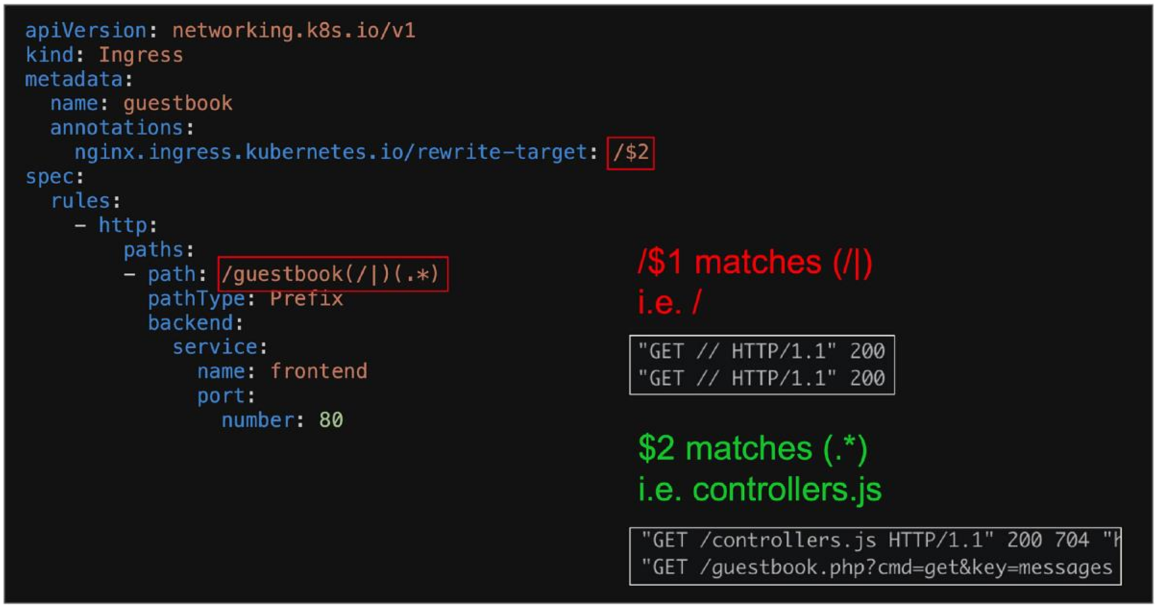
Option 2: Modify the Kubernetes Ingress to include a capture group

> *"In Version 0.22.0 and beyond, any substrings within the request URI that need to be passed to the rewritten path must explicitly be defined in a capture group."*

(Quoted from https://kubernetes.github.io/ingress-nginx/examples/rewrite/#rewrite-target)

The second option is to modify the Kubernetes Ingress to include the assets (e.g. **controllers.js**) when rewriting the target. This is achieved through the use of a capture group. In Figure 49 there are two capture groups, defined by the parentheses, which contain regex to match text. In the example in Figure 49, any text matching the first capture group, **(/|)**, can be accessed with **$1**. Any text matching the second capture group, **(.*)**, can be accessed by referencing **$2**.

To serve the static assets (e.g. **controllers.js**), the path should be rewritten to include the text in the second capture group. The regular expression, (.*), used in this capture group will match any characters after /**guestbook**/, for example /**guestbook/controllers.js**



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: guestbook
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
    - http:
        paths:
        - path: /guestbook(/|)(.*)
          pathType: Prefix
          backend:
            service:
              name: frontend
              port:
                number: 80
```

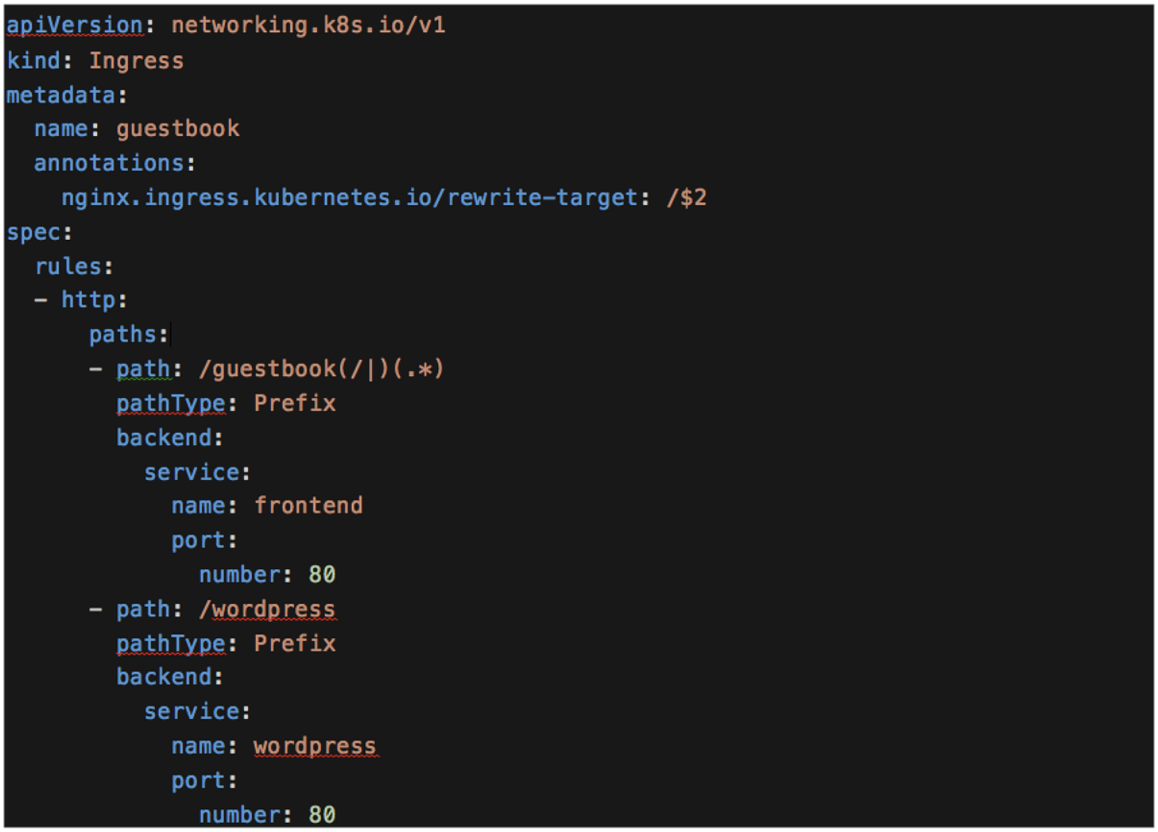
/$1 matches (/|)
i.e. /

```
"GET // HTTP/1.1" 200
"GET // HTTP/1.1" 200
```

$2 matches (.*)
i.e. controllers.js

```
"GET /controllers.js HTTP/1.1" 200 704 "
"GET /guestbook.php?cmd=get&key=messages
```

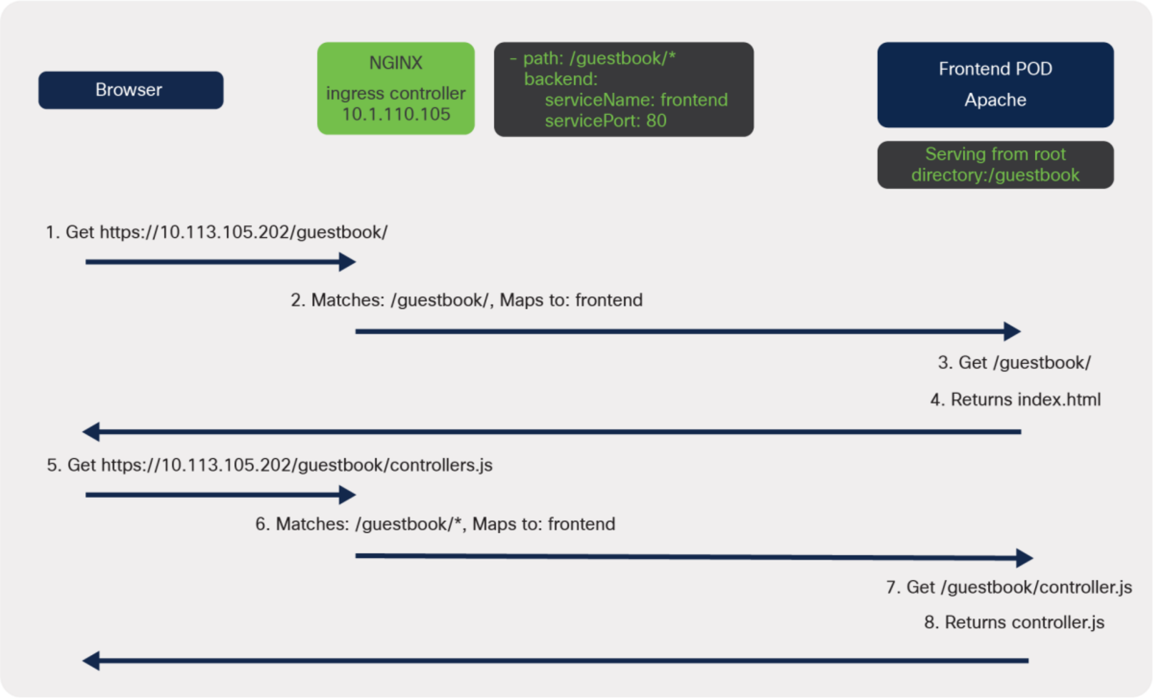
Figure 49.
The modified Kubernetes Ingress which includes a capture group



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: guestbook
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
  - http:
      paths:
      - path: /guestbook(/|)(.*)
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
      - path: /wordpress
        pathType: Prefix
        backend:
          service:
            name: wordpress
            port:
              number: 80
```

Figure 50.
Ingress YAML definition including the rewrite capture groups

Option 3: Change the Apache serving directory



Browser

NGINX
ingress controller
10.1.110.105

```
- path: /guestbook/*
  backend:
     serviceName: frontend
     servicePort: 80
```

Frontend POD
Apache

Serving from root
directory:/guestbook

1. Get https://10.113.105.202/guestbook/

2. Matches: /guestbook/, Maps to: frontend

3. Get /guestbook/

4. Returns index.html

5. Get https://10.113.105.202/guestbook/controllers.js

6. Matches: /guestbook/*, Maps to: frontend

7. Get /guestbook/controller.js

8. Returns controller.js

Figure 51.
A diagram of the communication when modifying the web server to serve from a subdirectory

A third option is to modify the configuration on the web server, in this case Apache.

From the testing performed so far it is evident that a subdirectory, **/guestbook**, is required. When this was configured the web server returned a 404. This was caused by Apache serving content from the root directory, /. It was not able to find the guestbook subdirectory files.

To solve this problem you must tell Apache to stop serving content from root, /, and instead make **/guestbook** the root directory.

In the **apache2.conf** configuration file you need to modify the **DocumentRoot** so it reads:

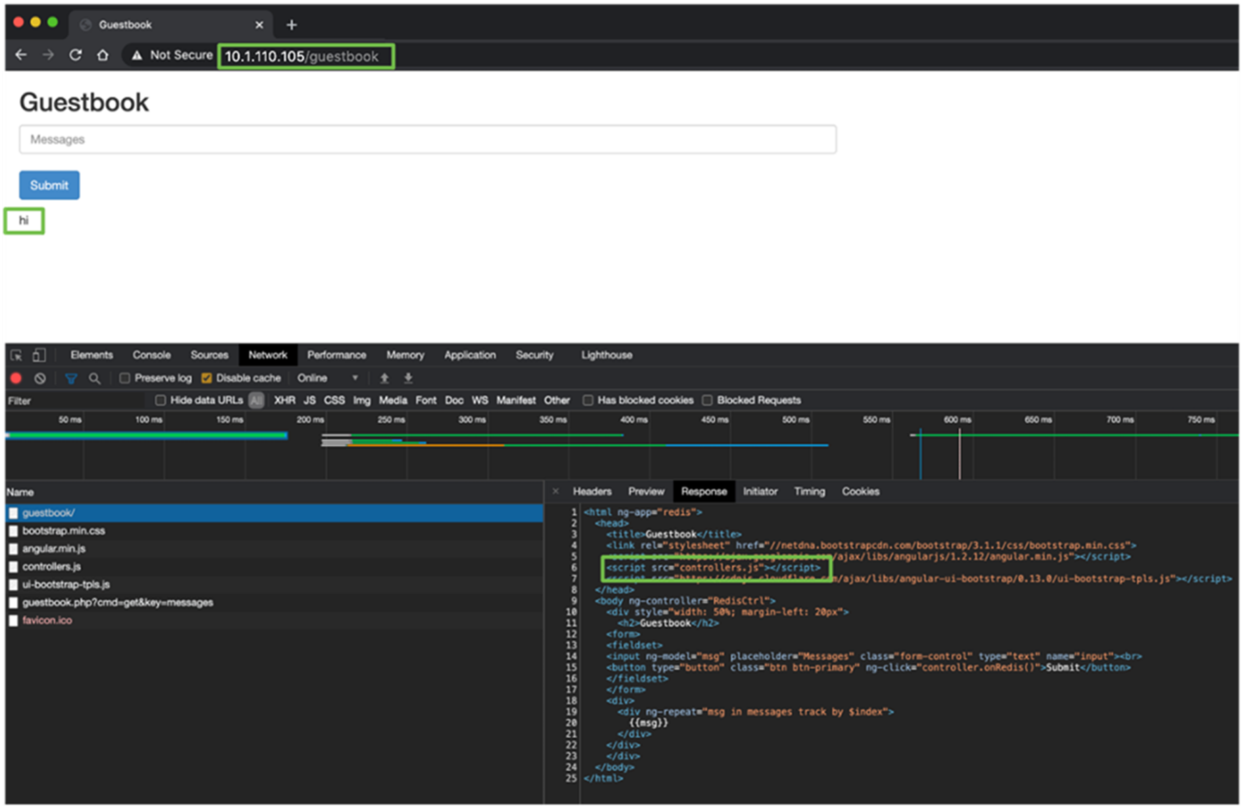
DocumentRoot /var/www/html/guestbook

On Ubuntu you should find **apache2.conf** in the **/etc/apache2/** directory.

Update this file and remove the rewrite-target annotation from the ingress because it is no longer serving content from the root directory. Also move all the guestbook files (**controllers.js, guestbook.php, index.html**) to a new folder so the new path is **/var/www/html/guestbook**. Restart Apache if needed.

If everything has worked the guestbook application should be loaded successfully and all static assets loaded correctly.

Important point: Apache was used as the web server in the guestbook example.
You may need to modify different files depending on the web server you have deployed.



```
<html ng-app="redis">
  <head>
    <title>Guestbook</title>
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.12/angular.min.js"></script>
    <script src="controllers.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-bootstrap/0.13.0/ui-bootstrap-tpls.js"></script>
  </head>
  <body ng-controller="RedisCtrl">
    <div style="width: 50%; margin-left: 20px">
      <h2>Guestbook</h2>
    <form>
    <fieldset>
    <input ng-model="msg" placeholder="Messages" class="form-control" type="text" name="input"><br>
    <button type="button" class="btn btn-primary" ng-click="controller.onRedis()">Submit</button>
    </fieldset>
    </form>
    <div>
      <div ng-repeat="msg in messages track by $index">
        {{msg}}
      </div>
    </div>
    </div>
  </body>
</html>
```

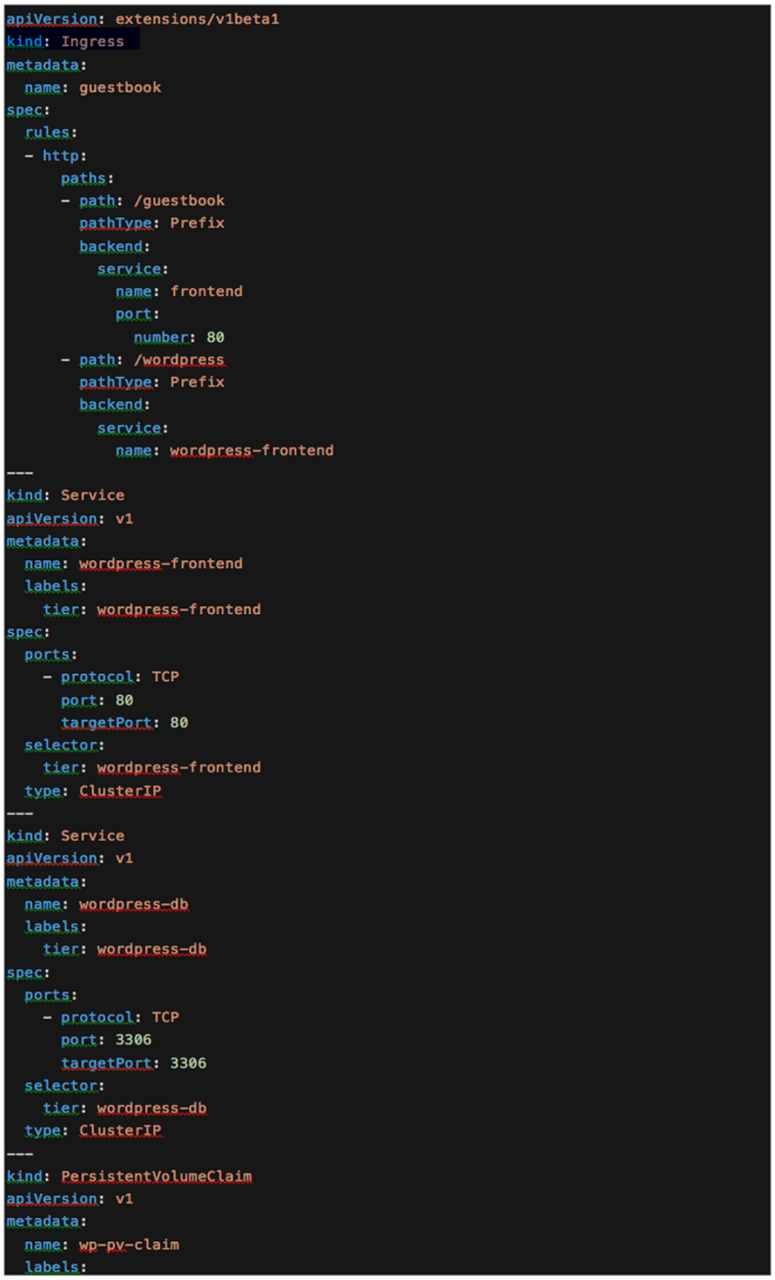
Figure 52.
Output from guestbook when the ingress path is working correctly

Another example – Wordpress

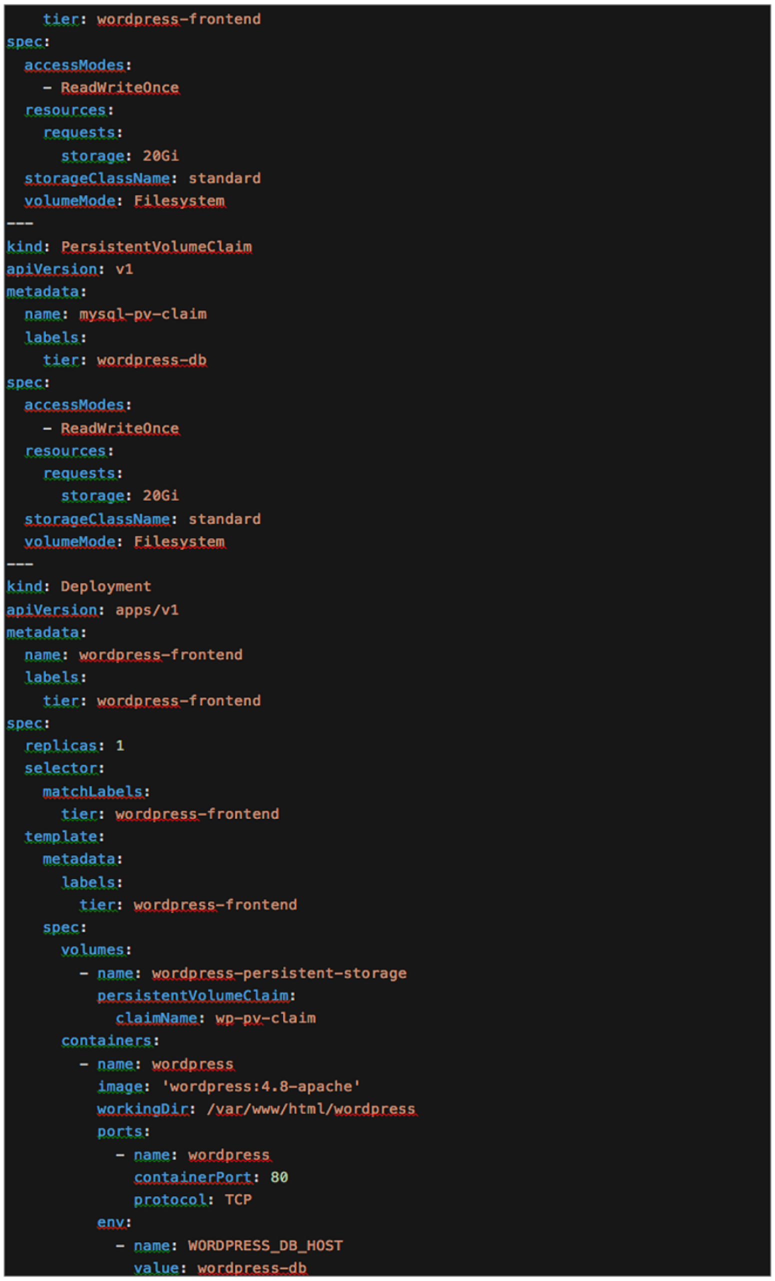
In some cases you may not need to update the web server or code, for example if deploying a Wordpress site.

Wordpress is a popular opensource content management system that was originally created for blogging. It's made up of a PHP frontend with a MySQL database. In this example it is deployed in Kubernetes as two pods, one frontend (Apache with PHP) and one DB (MySQL).

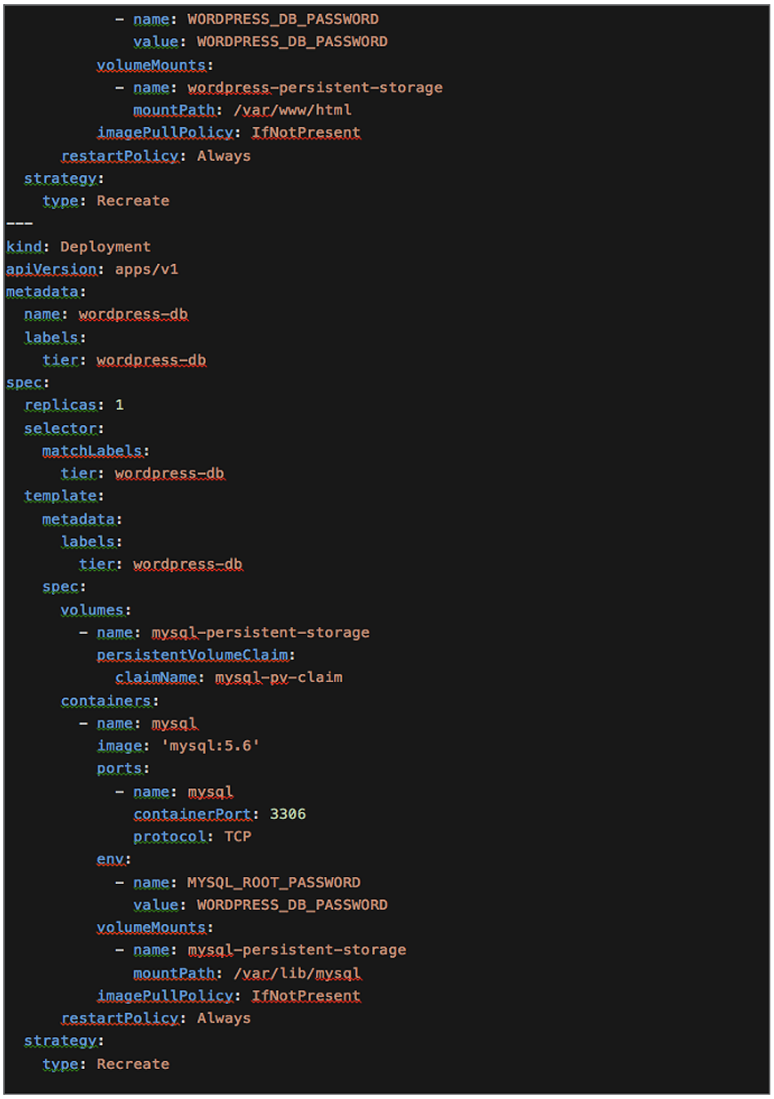
To configure Wordpress behind a Kubernetes Ingress you can change the working directory for the Wordpress frontend deployment. In Figure 53 below you can see that the **workingDir** is **/var/www/html/wordpress**



```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: guestbook
spec:
  rules:
  - http:
      paths:
      - path: /guestbook
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
      - path: /wordpress
        pathType: Prefix
        backend:
          service:
            name: wordpress-frontend
---
kind: Service
apiVersion: v1
metadata:
  name: wordpress-frontend
  labels:
    tier: wordpress-frontend
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  selector:
    tier: wordpress-frontend
  type: ClusterIP
---
kind: Service
apiVersion: v1
metadata:
  name: wordpress-db
  labels:
    tier: wordpress-db
spec:
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
  selector:
    tier: wordpress-db
  type: ClusterIP
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: wp-pv-claim
  labels:
```



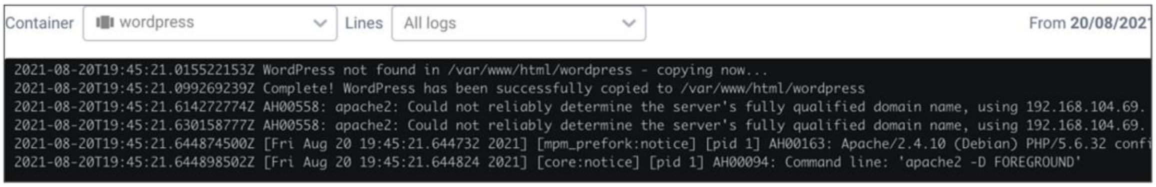
```
    tier: wordpress-frontend
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  storageClassName: standard
  volumeMode: Filesystem
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mysql-pv-claim
  labels:
    tier: wordpress-db
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  storageClassName: standard
  volumeMode: Filesystem
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: wordpress-frontend
  labels:
    tier: wordpress-frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      tier: wordpress-frontend
  template:
    metadata:
      labels:
        tier: wordpress-frontend
    spec:
      volumes:
        - name: wordpress-persistent-storage
          persistentVolumeClaim:
            claimName: wp-pv-claim
      containers:
        - name: wordpress
          image: 'wordpress:4.8-apache'
          workingDir: /var/www/html/wordpress
          ports:
            - name: wordpress
              containerPort: 80
              protocol: TCP
          env:
            - name: WORDPRESS_DB_HOST
              value: wordpress-db
```



```
            - name: WORDPRESS_DB_PASSWORD
              value: WORDPRESS_DB_PASSWORD
          volumeMounts:
            - name: wordpress-persistent-storage
              mountPath: /var/www/html
          imagePullPolicy: IfNotPresent
      restartPolicy: Always
  strategy:
    type: Recreate
---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: wordpress-db
  labels:
    tier: wordpress-db
spec:
  replicas: 1
  selector:
    matchLabels:
      tier: wordpress-db
  template:
    metadata:
      labels:
        tier: wordpress-db
    spec:
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim
      containers:
        - name: mysql
          image: 'mysql:5.6'
          ports:
            - name: mysql
              containerPort: 3306
              protocol: TCP
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: WORDPRESS_DB_PASSWORD
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
          imagePullPolicy: IfNotPresent
      restartPolicy: Always
  strategy:
    type: Recreate
```

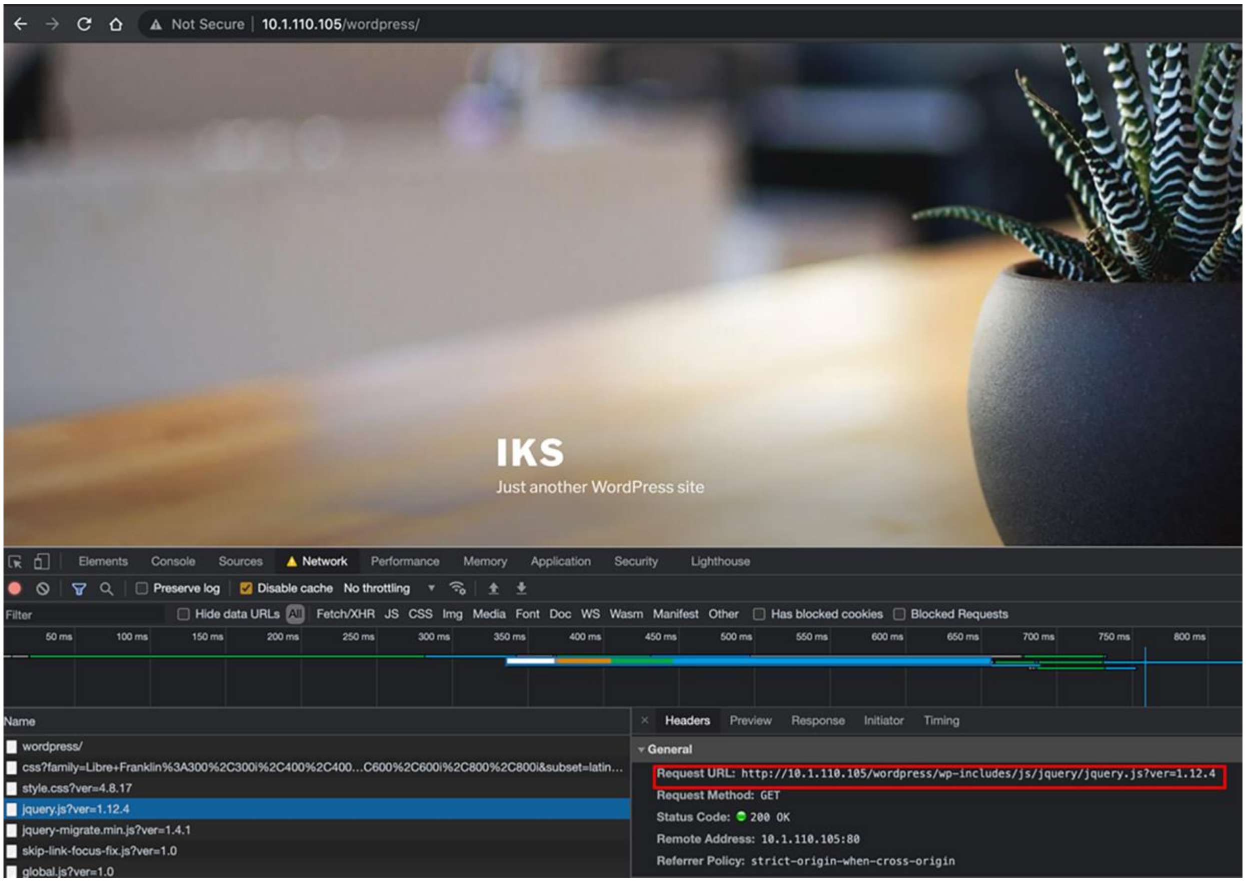
Figure 53.
YAML for the all-in-one Wordpress deployment

When the configuration is applied, the deployment scripts will install Wordpress into the current working directory **(/var/www/html/wordpress)**.



```
2021-08-20T19:45:21.015522153Z WordPress not found in /var/www/html/wordpress - copying now...
2021-08-20T19:45:21.099269239Z Complete! WordPress has been successfully copied to /var/www/html/wordpress
2021-08-20T19:45:21.614272774Z AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.104.69.
2021-08-20T19:45:21.630158777Z AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.104.69.
2021-08-20T19:45:21.644874500Z [Fri Aug 20 19:45:21.644732 2021] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.10 (Debian) PHP/5.6.32 confi
2021-08-20T19:45:21.644898502Z [Fri Aug 20 19:45:21.644824 2021] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
```

Figure 54.
Log files from the initial Wordpress deployment showing the Wordpress files are copied to the new subdirectory



Name:
- wordpress/
- css?family=Libre+Franklin%3A300%2C300i%2C400%2C400...C600%2C600i%2C800%2C800i&subset=latin...
- style.css?ver=4.8.17
- jquery.js?ver=1.12.4
- jquery-migrate.min.js?ver=1.4.1
- skip-link-focus-fix.js?ver=1.0
- global.js?ver=1.0

General

```
Request URL: http://10.1.110.105/wordpress/wp-includes/js/jquery/jquery.js?ver=1.12.4
Request Method: GET
Status Code: ● 200 OK
Remote Address: 10.1.110.105:80
Referrer Policy: strict-origin-when-cross-origin
```

Figure 55.
Developer tools output from the Wordpress deployment behind an ingress showing the Wordpress site is accessible from the subdirectory

Putting it all together – an end-to-end flow

This section will trace the packet flow using the Wordpress all-in-one deployment from Figure 53.

Important point: The following configuration and patterns have been used to capture **some data flows for this example. This is not a complete output but aims to provide guidance on tracing flows.**

Test setup

- IPTables was configured to log to **/var/log/messages**
- The following IPTables logging rules were configured.
    - The log prefix is used when printing the output table and contains the chain and table information. See Figure 56 below for an example.

```
#### HANDSHAKE ####
sudo iptables -t mangle -I cali-PREROUTING 1 -p tcp -m state --state NEW -j LOG --log-prefix
"_CALI_PRT_NEW_CONN mangle " --log-level 1


sudo iptables -t raw -I cali-PREROUTING 1 -p tcp --tcp-flags SYN,ACK SYN,ACK -j LOG --log-
prefix "_CALI_PRT_CONN_EST raw " --log-level 1
sudo iptables -t raw -I cali-PREROUTING 2 -p tcp --tcp-flags FIN,ACK FIN,ACK -j LOG --log-
prefix "_CALI_PRT_CONN_CLOSED raw " --log-level 1


#### RAW ####
sudo iptables -t raw -I cali-PREROUTING 3 -j LOG --log-prefix "_CALI_PRT raw " --log-level 1


#### MANGLE ####
sudo iptables -t mangle -I cali-PREROUTING 2 -j LOG --log-prefix "_CALI_PRT_CSTATE mangle "
--log-level 1


#### FILTER ####
sudo iptables -t filter -I cali-from-hep-forward 1 -j LOG --log-prefix "_CALI_FWD_TO_HEP
filter " --log-level 1
sudo iptables -t filter -I cali-from-hep-forward 1 -j LOG --log-prefix "_CALI_FWD_FROM_HEP
filter " --log-level 1
sudo iptables -t filter -I cali-to-wl-dispatch 1 -j LOG --log-prefix "_CALI_FWD_TO_POD
filter " --log-level 1
sudo iptables -t filter -I cali-from-wl-dispatch 1 -j LOG --log-prefix "_CALI_FWD_FROM_POD
filter " --log-level 1
sudo iptables -t filter -I cali-tw-cali11aba968e90 1 -j LOG --log-prefix
"_CALI_FWD_AT_POD_WP filter " --log-level 1
sudo iptables -t filter -I cali-tw-cali545edbfd5b2 1 -j LOG --log-prefix
"_CALI_FWD_AT_POD_DB filter " --log-level 1
```

```
sudo iptables -t filter -I cali-from-hep-forward 1 -j LOG --log-prefix "_CALI_INPUT filter " --log-level 1

sudo iptables -t filter -I cali-from-hep-forward 1 -j LOG --log-prefix "_CALI_OUTPUT filter " --log-level 1

sudo iptables -t filter -I KUBE-FORWARD 1 -j LOG --log-prefix "_KUBE_FORWARD_CSTATE filter " --log-level 1


#### NAT ####


sudo iptables -t nat -I cali-PREROUTING 1 -j LOG --log-prefix "_CALI_PRT nat " --log-level 1

sudo iptables -t nat -I KUBE-NODEPORTS 1 -j LOG --log-prefix "_KUBE_NODEPORTS_NGINX nat " --log-level 1

sudo iptables -t nat -I KUBE-SVC-D3XY3KFIPC4U6KQS 1 -j LOG --log-prefix "_KUBE_SVC_NGINX nat " --log-level 1

sudo iptables -t nat -I KUBE-SEP-AH3PTBOUGNT5BH3N 2 -j LOG --log-prefix "_KUBE_NGINX_WKR_1_DNAT nat " --log-level 1

sudo iptables -t nat -I KUBE-SEP-JKWKVBBQCO4U3ZLS 2 -j LOG --log-prefix "_KUBE_NGINX_WKR_2_DNAT nat " --log-level 1

sudo iptables -t nat -I KUBE-SVC-Y5XG6JG74RRE5O3W 1 -j LOG --log-prefix "_KUBE_SVC_WP_FRONTEND nat " --log-level 1

sudo iptables -t nat -I KUBE-SEP-QAOWPL7S7I7Y6Q3X 2 -j LOG --log-prefix "_KUBE_WP_FRNTEND_WKR_2_DNAT nat " --log-level 1

sudo iptables -t nat -I KUBE-SVC-235M4OZWKIE6GYSD 1 -j LOG --log-prefix "_KUBE_SVC_WP_DB nat " --log-level 1

sudo iptables -t nat -I KUBE-SEP-WAAL3DOSOAXDLALS 2 -j LOG --log-prefix "_KUBE_WP_DB_WKR_2_DNAT nat " --log-level 1

sudo iptables -t nat -I KUBE-SVC-TCOU7JCQXEZGVUNU 1 -j LOG --log-prefix "_KUBE_SVC_DNS nat " --log-level 1

sudo iptables -t nat -I KUBE-SEP-GD4HYCW4WSZW4UQN 2 -j LOG --log-prefix "_KUBE_DNS_CNTL_DNAT nat " --log-level 1

sudo iptables -t nat -I KUBE-SEP-A34722GV4XXAYJWX 2 -j LOG --log-prefix "_KUBE_DNS_WKR_2_DNAT nat " --log-level 1

sudo iptables -t nat -I cali-OUTPUT 1 -j LOG --log-prefix "_CALI_OUTPUT nat " --log-level 1

sudo iptables -t nat -I cali-nat-outgoing 1 -j LOG --log-prefix "_CALI_NAT_OUTGOING_NAT nat " --log-level 1

sudo iptables -t nat -I KUBE-POSTROUTING 3 -j LOG --log-prefix "_KUBE_POSTROUTING_NAT nat " --log-level 1
```

- A grep patterns file was created to filter the required output logs,

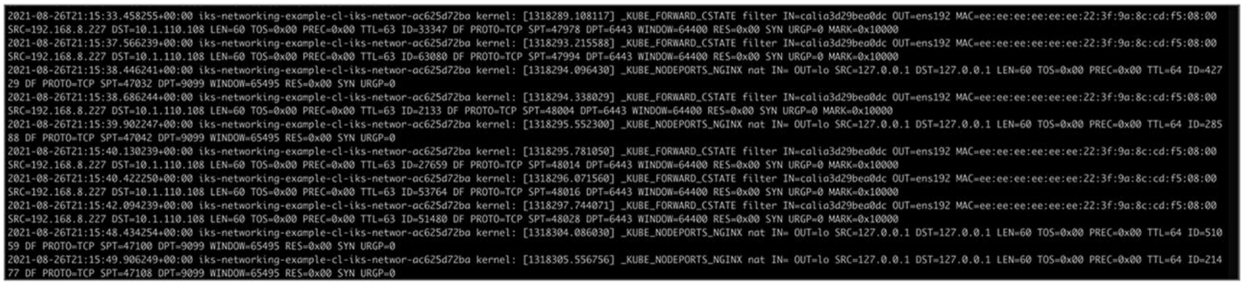
```
systemd\|rsyslogd\|kubelet\|named\|=6443\|sshd\|SRC=127.0.0.1\|192.168.104.67\|192.168.8.232\|192.168.8.228\|192.168.8.231\|192.168.8.195\|192.168.8.227\|DPT=10254\|DPT=8181\|DPT=8080\|SPT=22\|DPT=22\|SPT=443\|DPT=443\|SPT=179\|DPT=179\|SPT=123\|DPT=123
```

- A SED patterns file was used to convert the IP addresses into more relevant names,

```
s/MAC=.*SRC/SRC/g
s/10.1.0.89/client_browser/g
s/10.1.110.108/control_ens192/g
s/10.1.110.102/worker_1_ens192/g
s/10.1.110.107/worker_2_ens192/g
s/192.168.165.192/control_tunnel0/g
s/192.168.104.64/worker_1_tunnel0/g
s/192.168.8.192/worker_2_tunnel0/g
s/192.168.104.65/nginx_pod_worker_1/g
s/192.168.8.193/nginx_pod_worker_2/g
s/10.1.110.105/ingress_loadbalancer_ip/g
s/192.168.8.222/wordpress_frontend/g
s/192.168.8.223/wordpress_db/g
s/192.168.8.207/kube_dns/g
s/192.168.165.196/kube_dns/g
s/10.96.23.229/wordpress_db_svc/g
s/10.96.0.10/kube-dns-svc/g
s/10.96.179.211/nginx_svc_cluster_ip/g
s/cali7394c99cdd4/nginx_worker1_veth/g
s/cali21ac0ad37cb/nginx_worker2_veth/g
s/cali8d8a05feee4/coredns_worker_1_veth/g
s/cali148b7202262/coredns_worker_2_veth/g
s/cali11aba968e90/wordpress_frontend_veth/g
s/cali545edbfd5b2/wordpress_db_veth/g
```

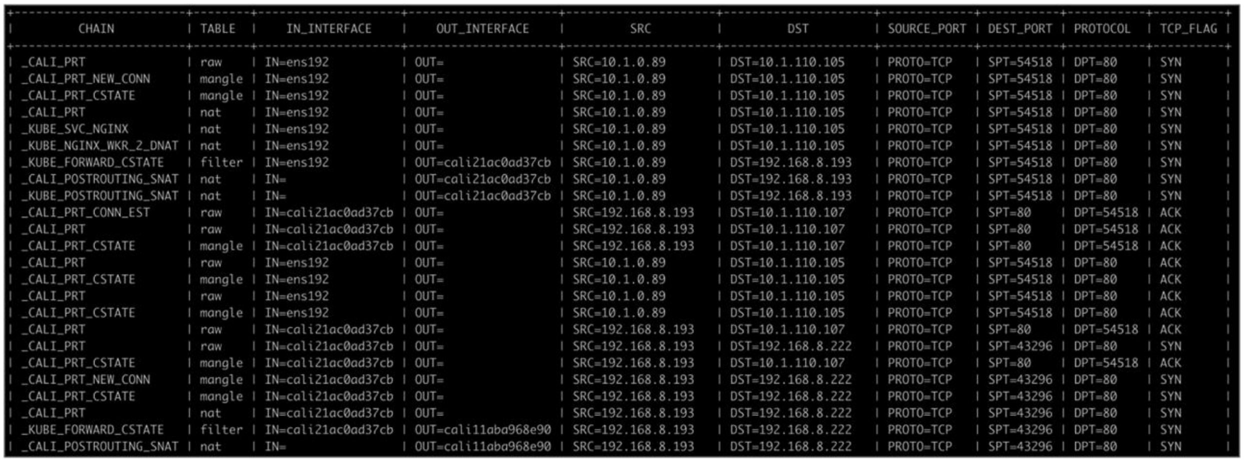
- The output was cut into columns and added to a table to achieve the end result.

```
cat /var/log/messages | grep -v -f grep_patterns_to_remove | sed -f sed_patterns | cut -d '
' -f 5,6,7,8,9,10,17,18,19,22 | showtable -d " " -
titles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,PROTOCOL,SOURCE_PORT,DEST_PORT,TCP_FLA
G | sed 's/^ *//g' | more
```



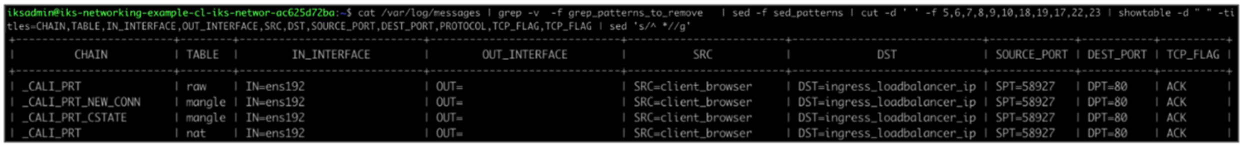
```
2021-08-26T21:15:33.458255+00:00 iks-networking-example-cl-iks-networ-ac625d72ba kernel: [1318289.108117] _KUBE_FORWARD_CSTATE filter IN=calia3d29bea0dc OUT=ens192 MAC=ee:ee:ee:ee:ee:ee:22:3f:9a:8c:cd:f5:08:00
SRC=192.168.8.227 DST=10.1.110.108 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=33347 DF PROTO=TCP SPT=47978 DPT=6443 WINDOW=64400 RES=0x00 SYN URGP=0 MARK=0x10000
2021-08-26T21:15:37.566239+00:00 iks-networking-example-cl-iks-networ-ac625d72ba kernel: [1318293.215588] _KUBE_FORWARD_CSTATE filter IN=calia3d29bea0dc OUT=ens192 MAC=ee:ee:ee:ee:ee:ee:22:3f:9a:8c:cd:f5:08:00
SRC=192.168.8.227 DST=10.1.110.108 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=63080 DF PROTO=TCP SPT=47994 DPT=6443 WINDOW=64400 RES=0x00 SYN URGP=0 MARK=0x10000
2021-08-26T21:15:38.446241+00:00 iks-networking-example-cl-iks-networ-ac625d72ba kernel: [1318294.096430] _KUBE_NODEPORTS_NGINX nat IN= OUT=lo SRC=127.0.0.1 DST=127.0.0.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=427
29 DF PROTO=TCP SPT=47032 DPT=9099 WINDOW=65495 RES=0x00 SYN URGP=0
2021-08-26T21:15:38.686244+00:00 iks-networking-example-cl-iks-networ-ac625d72ba kernel: [1318294.338029] _KUBE_FORWARD_CSTATE filter IN=calia3d29bea0dc OUT=ens192 MAC=ee:ee:ee:ee:ee:ee:22:3f:9a:8c:cd:f5:08:00
SRC=192.168.8.227 DST=10.1.110.108 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=2133 DF PROTO=TCP SPT=48004 DPT=6443 WINDOW=64400 RES=0x00 SYN URGP=0 MARK=0x10000
2021-08-26T21:15:39.902247+00:00 iks-networking-example-cl-iks-networ-ac625d72ba kernel: [1318295.552300] _KUBE_NODEPORTS_NGINX nat IN= OUT=lo SRC=127.0.0.1 DST=127.0.0.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=285
88 DF PROTO=TCP SPT=47042 DPT=9099 WINDOW=65495 RES=0x00 SYN URGP=0
2021-08-26T21:15:40.130239+00:00 iks-networking-example-cl-iks-networ-ac625d72ba kernel: [1318295.781050] _KUBE_FORWARD_CSTATE filter IN=calia3d29bea0dc OUT=ens192 MAC=ee:ee:ee:ee:ee:ee:22:3f:9a:8c:cd:f5:08:00
SRC=192.168.8.227 DST=10.1.110.108 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=27659 DF PROTO=TCP SPT=48014 DPT=6443 WINDOW=64400 RES=0x00 SYN URGP=0 MARK=0x10000
2021-08-26T21:15:40.422250+00:00 iks-networking-example-cl-iks-networ-ac625d72ba kernel: [1318296.071560] _KUBE_FORWARD_CSTATE filter IN=calia3d29bea0dc OUT=ens192 MAC=ee:ee:ee:ee:ee:ee:22:3f:9a:8c:cd:f5:08:00
SRC=192.168.8.227 DST=10.1.110.108 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=53764 DF PROTO=TCP SPT=48016 DPT=6443 WINDOW=64400 RES=0x00 SYN URGP=0 MARK=0x10000
2021-08-26T21:15:42.094239+00:00 iks-networking-example-cl-iks-networ-ac625d72ba kernel: [1318297.744071] _KUBE_FORWARD_CSTATE filter IN=calia3d29bea0dc OUT=ens192 MAC=ee:ee:ee:ee:ee:ee:22:3f:9a:8c:cd:f5:08:00
SRC=192.168.8.227 DST=10.1.110.108 LEN=60 TOS=0x00 PREC=0x00 TTL=63 ID=51480 DF PROTO=TCP SPT=48028 DPT=6443 WINDOW=64400 RES=0x00 SYN URGP=0 MARK=0x10000
2021-08-26T21:15:48.434254+00:00 iks-networking-example-cl-iks-networ-ac625d72ba kernel: [1318304.086030] _KUBE_NODEPORTS_NGINX nat IN= OUT=lo SRC=127.0.0.1 DST=127.0.0.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=510
59 DF PROTO=TCP SPT=47100 DPT=9099 WINDOW=65495 RES=0x00 SYN URGP=0
2021-08-26T21:15:49.906249+00:00 iks-networking-example-cl-iks-networ-ac625d72ba kernel: [1318305.556756] _KUBE_NODEPORTS_NGINX nat IN= OUT=lo SRC=127.0.0.1 DST=127.0.0.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=214
77 DF PROTO=TCP SPT=47108 DPT=9099 WINDOW=65495 RES=0x00 SYN URGP=0
```

Figure 56.
Example of **/var/log/messages**



```
+------------------------+--------+--------------------+---------------------+-----------------------+-----------------------+-------------+-----------+-----------+----------+
|         CHAIN          | TABLE  |    IN_INTERFACE    |    OUT_INTERFACE    |          SRC          |          DST          | SOURCE_PORT | DEST_PORT | PROTOCOL  | TCP_FLAG |
+------------------------+--------+--------------------+---------------------+-----------------------+-----------------------+-------------+-----------+-----------+----------+
| _CALI_PRT              | raw    | IN=ens192          | OUT=                | SRC=10.1.0.89         | DST=10.1.110.105      | PROTO=TCP   | SPT=54518 | DPT=80    | SYN      |
| _CALI_PRT_NEW_CONN     | mangle | IN=ens192          | OUT=                | SRC=10.1.0.89         | DST=10.1.110.105      | PROTO=TCP   | SPT=54518 | DPT=80    | SYN      |
| _CALI_PRT_CSTATE       | mangle | IN=ens192          | OUT=                | SRC=10.1.0.89         | DST=10.1.110.105      | PROTO=TCP   | SPT=54518 | DPT=80    | SYN      |
| _CALI_PRT              | nat    | IN=ens192          | OUT=                | SRC=10.1.0.89         | DST=10.1.110.105      | PROTO=TCP   | SPT=54518 | DPT=80    | SYN      |
| _KUBE_SVC_NGINX        | nat    | IN=ens192          | OUT=                | SRC=10.1.0.89         | DST=10.1.110.105      | PROTO=TCP   | SPT=54518 | DPT=80    | SYN      |
| _KUBE_NGINX_WKR_2_DNAT | nat    | IN=ens192          | OUT=                | SRC=10.1.0.89         | DST=10.1.110.105      | PROTO=TCP   | SPT=54518 | DPT=80    | SYN      |
| _KUBE_FORWARD_CSTATE   | filter | IN=ens192          | OUT=cali21ac0ad37cb | SRC=10.1.0.89         | DST=192.168.8.193     | PROTO=TCP   | SPT=54518 | DPT=80    | SYN      |
| _CALI_POSTROUTING_SNAT | nat    | IN=                | OUT=cali21ac0ad37cb | SRC=10.1.0.89         | DST=192.168.8.193     | PROTO=TCP   | SPT=54518 | DPT=80    | SYN      |
| _KUBE_POSTROUTING_SNAT | nat    | IN=                | OUT=cali21ac0ad37cb | SRC=10.1.0.89         | DST=192.168.8.193     | PROTO=TCP   | SPT=54518 | DPT=80    | SYN      |
| _CALI_PRT_CONN_EST     | raw    | IN=cali21ac0ad37cb | OUT=                | SRC=192.168.8.193     | DST=10.1.110.107      | PROTO=TCP   | SPT=80    | DPT=54518 | ACK      |
| _CALI_PRT              | raw    | IN=cali21ac0ad37cb | OUT=                | SRC=192.168.8.193     | DST=10.1.110.107      | PROTO=TCP   | SPT=80    | DPT=54518 | ACK      |
| _CALI_PRT_CSTATE       | mangle | IN=cali21ac0ad37cb | OUT=                | SRC=192.168.8.193     | DST=10.1.110.107      | PROTO=TCP   | SPT=80    | DPT=54518 | ACK      |
| _CALI_PRT              | raw    | IN=ens192          | OUT=                | SRC=10.1.0.89         | DST=10.1.110.105      | PROTO=TCP   | SPT=54518 | DPT=80    | ACK      |
| _CALI_PRT_CSTATE       | mangle | IN=ens192          | OUT=                | SRC=10.1.0.89         | DST=10.1.110.105      | PROTO=TCP   | SPT=54518 | DPT=80    | ACK      |
| _CALI_PRT              | raw    | IN=ens192          | OUT=                | SRC=10.1.0.89         | DST=10.1.110.105      | PROTO=TCP   | SPT=54518 | DPT=80    | ACK      |
| _CALI_PRT_CSTATE       | mangle | IN=ens192          | OUT=                | SRC=10.1.0.89         | DST=10.1.110.105      | PROTO=TCP   | SPT=54518 | DPT=80    | ACK      |
| _CALI_PRT              | raw    | IN=cali21ac0ad37cb | OUT=                | SRC=192.168.8.193     | DST=10.1.110.107      | PROTO=TCP   | SPT=80    | DPT=54518 | ACK      |
| _CALI_PRT              | raw    | IN=cali21ac0ad37cb | OUT=                | SRC=192.168.8.193     | DST=192.168.8.222     | PROTO=TCP   | SPT=43296 | DPT=80    | SYN      |
| _CALI_PRT_CSTATE       | mangle | IN=cali21ac0ad37cb | OUT=                | SRC=192.168.8.193     | DST=10.1.110.107      | PROTO=TCP   | SPT=80    | DPT=54518 | ACK      |
| _CALI_PRT_NEW_CONN     | mangle | IN=cali21ac0ad37cb | OUT=                | SRC=192.168.8.193     | DST=192.168.8.222     | PROTO=TCP   | SPT=43296 | DPT=80    | SYN      |
| _CALI_PRT_CSTATE       | mangle | IN=cali21ac0ad37cb | OUT=                | SRC=192.168.8.193     | DST=192.168.8.222     | PROTO=TCP   | SPT=43296 | DPT=80    | SYN      |
| _CALI_PRT              | nat    | IN=cali21ac0ad37cb | OUT=                | SRC=192.168.8.193     | DST=192.168.8.222     | PROTO=TCP   | SPT=43296 | DPT=80    | SYN      |
| _KUBE_FORWARD_CSTATE   | filter | IN=cali21ac0ad37cb | OUT=cali11aba968e90 | SRC=192.168.8.193     | DST=192.168.8.222     | PROTO=TCP   | SPT=43296 | DPT=80    | SYN      |
| _CALI_POSTROUTING_SNAT | nat    | IN=                | OUT=cali11aba968e90 | SRC=192.168.8.193     | DST=192.168.8.222     | PROTO=TCP   | SPT=43296 | DPT=80    | SYN      |
```

Figure 57.
Example of output without SED replacement



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+------------------------+--------+--------------------+---------------------+-----------------------+-----------------------------+-------------+-----------+----------+
|         CHAIN          | TABLE  |    IN_INTERFACE    |    OUT_INTERFACE    |          SRC          |             DST             | SOURCE_PORT | DEST_PORT | TCP_FLAG |
+------------------------+--------+--------------------+---------------------+-----------------------+-----------------------------+-------------+-----------+----------+
| _CALI_PRT              | raw    | IN=ens192          | OUT=                | SRC=client_browser    | DST=ingress_loadbalancer_ip | SPT=58927   | DPT=80    | ACK      |
| _CALI_PRT_NEW_CONN     | mangle | IN=ens192          | OUT=                | SRC=client_browser    | DST=ingress_loadbalancer_ip | SPT=58927   | DPT=80    | ACK      |
| _CALI_PRT_CSTATE       | mangle | IN=ens192          | OUT=                | SRC=client_browser    | DST=ingress_loadbalancer_ip | SPT=58927   | DPT=80    | ACK      |
| _CALI_PRT              | nat    | IN=ens192          | OUT=                | SRC=client_browser    | DST=ingress_loadbalancer_ip | SPT=58927   | DPT=80    | ACK      |
```

Figure 58.
Example of output with SED replacement

High-level flow

- Client browser connects to Wordpress (https://10.1.110.105/wordpress).

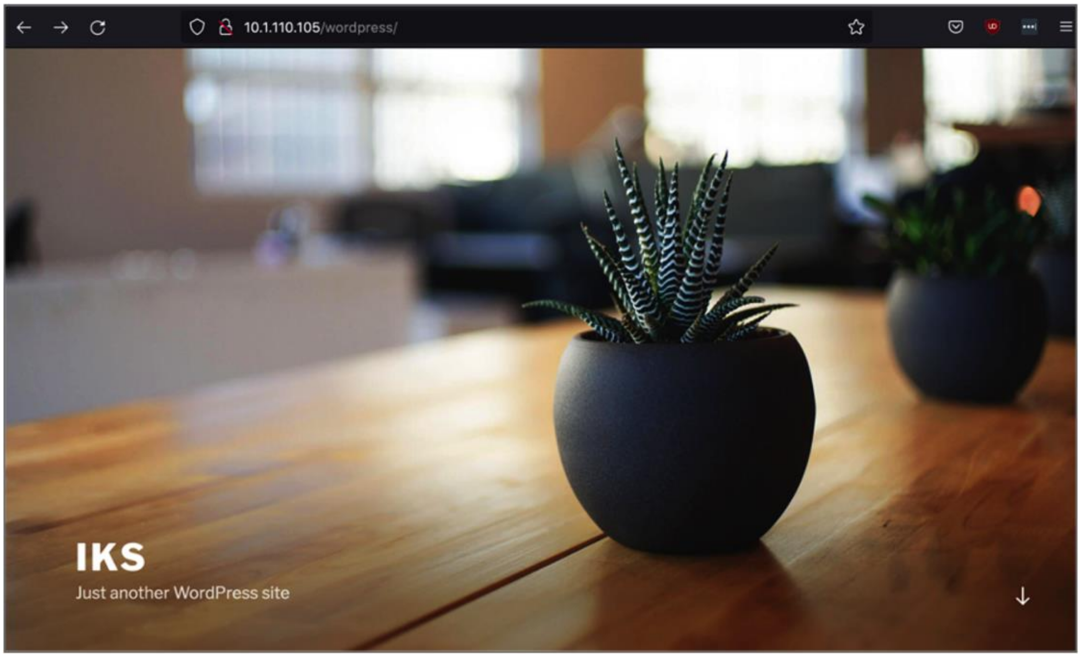


Figure 59.
Accessing the Wordpress frontend

- ARP sends a request for IKS NGINX IKS LoadBalancer IP (10.1.110.105).

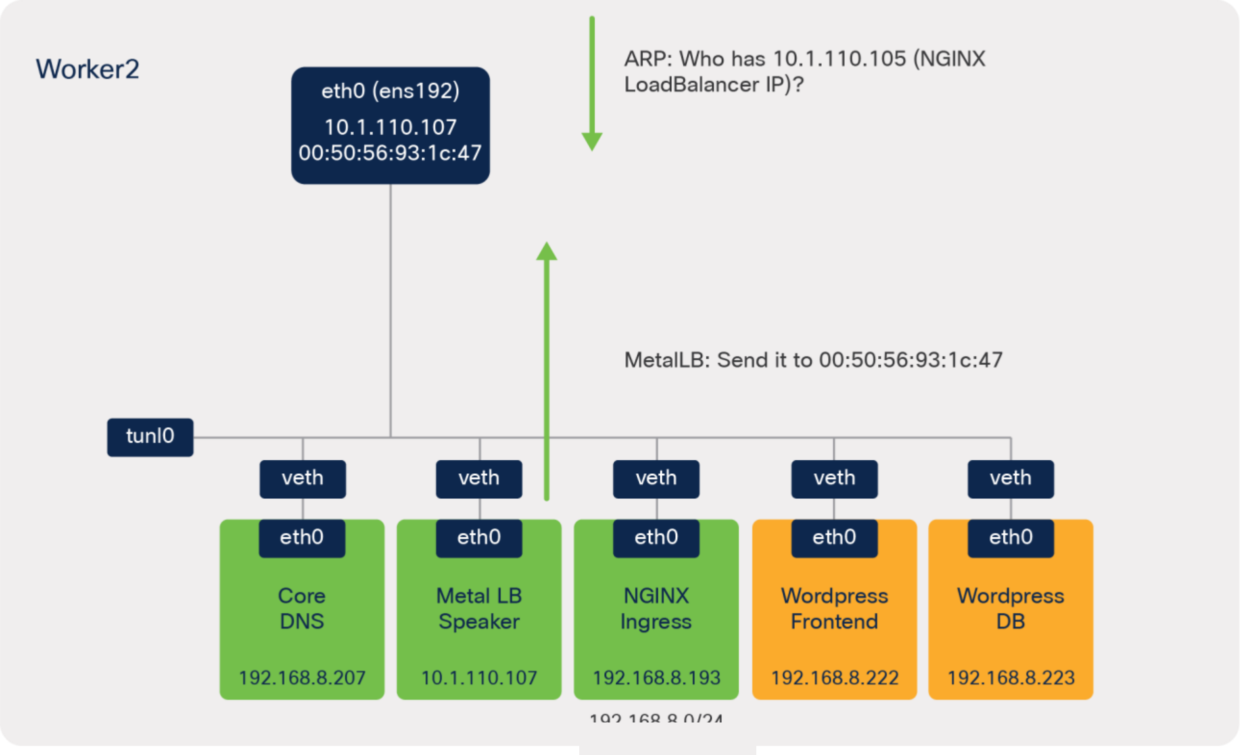
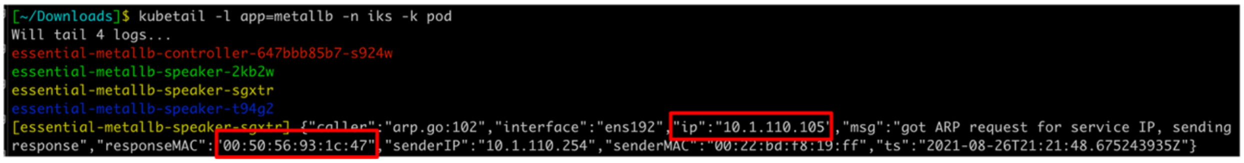
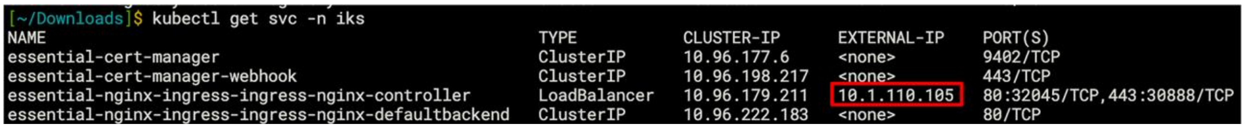


Figure 60.
ARP request and response



```
[~/Downloads]$ kubetail -l app=metallb -n iks -k pod
Will tail 4 logs...
essential-metallb-controller-647bbb85b7-s924w
essential-metallb-speaker-2kb2w
essential-metallb-speaker-sgxtr
essential-metallb-speaker-t94g2
[essential-metallb-speaker-sgxtr] {"caller":"arp.go:102","interface":"ens192","ip":"10.1.110.105","msg":"got ARP request for service IP, sending
response","responseMAC":"00:50:56:93:1c:47","senderIP":"10.1.110.254","senderMAC":"00:22:bd:f8:19:ff","ts":"2021-08-26T21:21:48.675243935Z"}
```

Figure 61.
Output from the **metallb** logs



```
[~/Downloads]$ kubectl get svc -n iks
NAME                                         TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)
essential-cert-manager                       ClusterIP      10.96.177.6     <none>         9402/TCP
essential-cert-manager-webhook               ClusterIP      10.96.198.217   <none>         443/TCP
essential-nginx-ingress-ingress-nginx-controller   LoadBalancer   10.96.179.211   10.1.110.105   80:32045/TCP,443:30888/TCP
essential-nginx-ingress-ingress-nginx-defaultbackend   ClusterIP   10.96.222.183   <none>     80/TCP
```

Figure 62.
Output from the **kubectl get svc -n iks** command

- MetalLB speaker (worker 2) responds – send it to **00:50:56:93:1c:47** (MAC of ens192 on worker 2).
- The IPTables **Raw, Mangle,** and **NAT** tables of the **Prerouting** chain are used.
    - Connection tracking is used to check the state as part of the **Mangle** table.

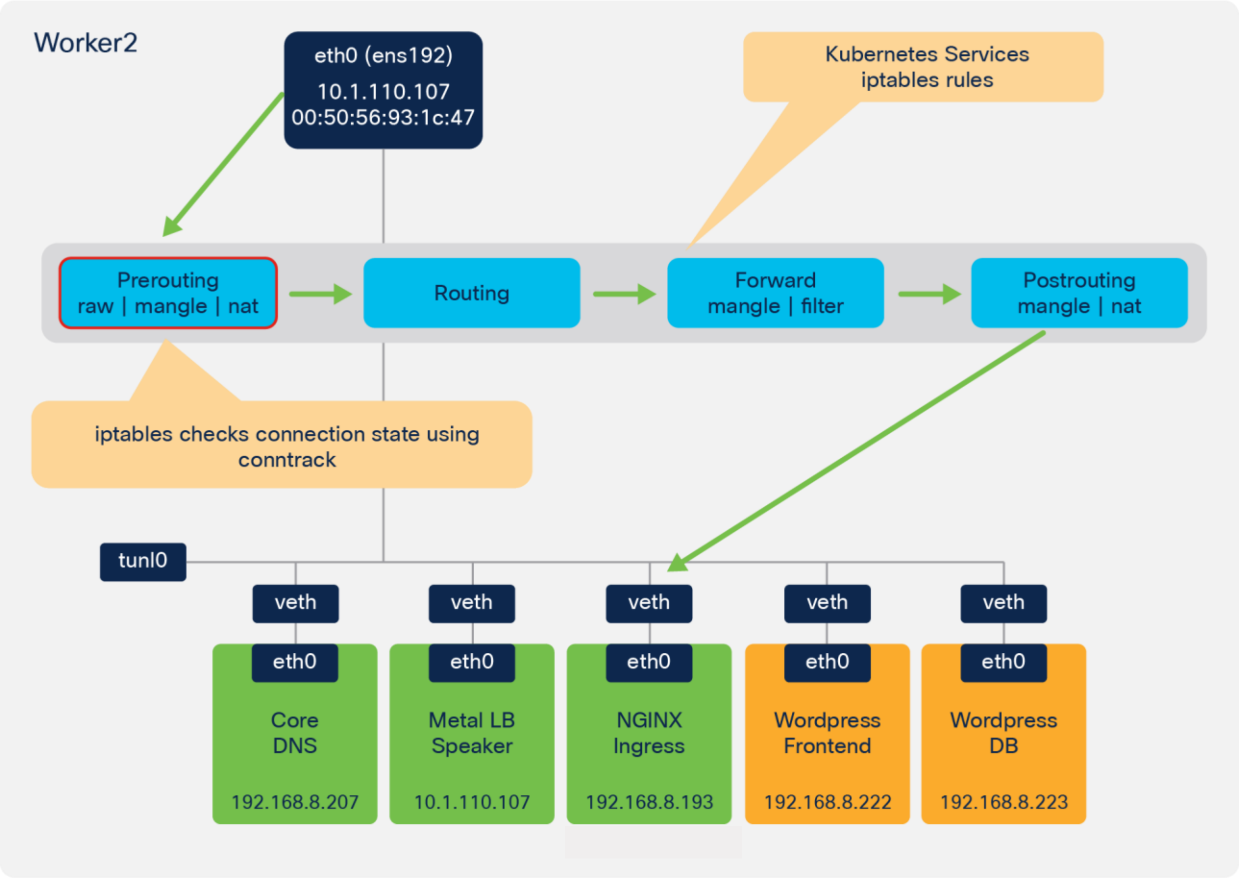
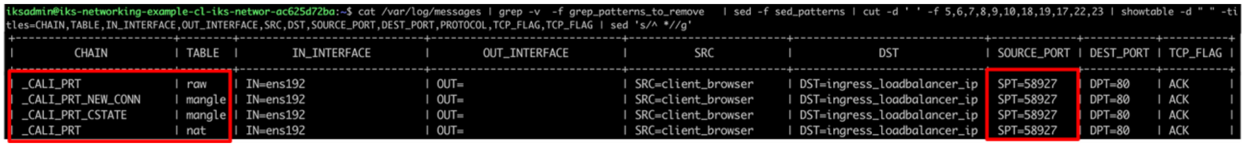
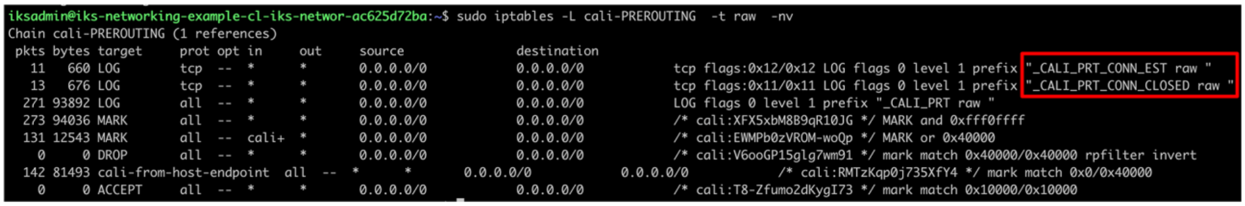


Figure 63.
Client browser to NGINX Ingress **– Prerouting chain**



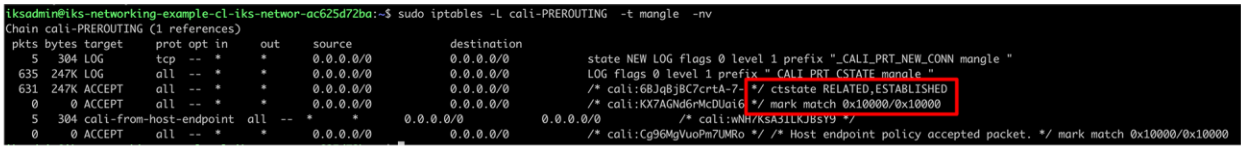
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+-----------------------+--------+---------------------------+---------------------------+----------------------+-----------------------------+-------------+----------+----------+
|        CHAIN          | TABLE  |        IN_INTERFACE       |       OUT_INTERFACE       |         SRC          |             DST             | SOURCE_PORT | DEST_PORT | TCP_FLAG |
+-----------------------+--------+---------------------------+---------------------------+----------------------+-----------------------------+-------------+----------+----------+
| _CALI_PRT             | raw    | IN=ens192                 | OUT=                      | SRC=client_browser   | DST=ingress_loadbalancer_ip  SPT=58927   | DPT=80    | ACK      |
| _CALI_PRT_NEW_CONN    | mangle | IN=ens192                 | OUT=                      | SRC=client_browser   | DST=ingress_loadbalancer_ip  SPT=58927   | DPT=80    | ACK      |
| _CALI_PRT_CSTATE      | mangle | IN=ens192                 | OUT=                      | SRC=client_browser   | DST=ingress_loadbalancer_ip  SPT=58927   | DPT=80    | ACK      |
| _CALI_PRT             | nat    | IN=ens192                 | OUT=                      | SRC=client_browser   | DST=ingress_loadbalancer_ip  SPT=58927   | DPT=80    | ACK      |
```

Figure 64.
Logging from the client browser to NGINX Ingress **– Prerouting chain**



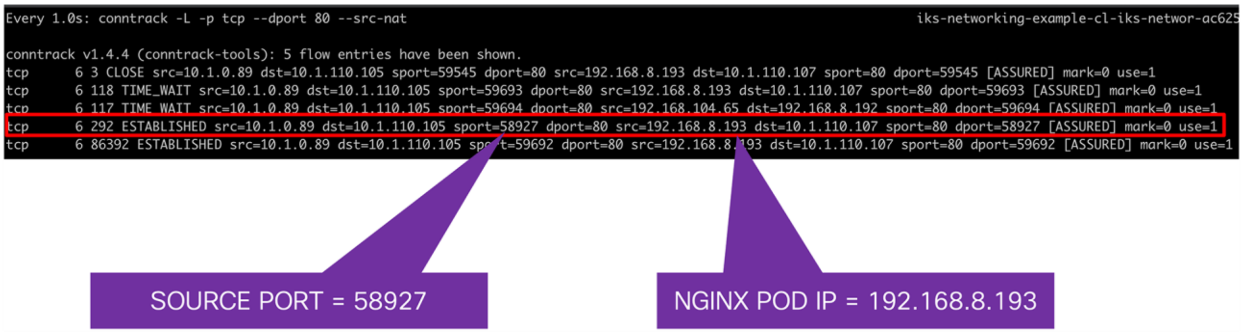
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L cali-PREROUTING  -t raw  -nv
Chain cali-PREROUTING (1 references)
 pkts bytes target     prot opt in     out     source               destination
   11   660 LOG        tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp flags:0x12/0x12 LOG flags 0 level 1 prefix "_CALI_PRT_CONN_EST raw "
   13   676 LOG        tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp flags:0x11/0x11 LOG flags 0 level 1 prefix "_CALI_PRT_CONN_CLOSED raw "
  271 93892 LOG        all  --  *      *       0.0.0.0/0            0.0.0.0/0            LOG flags 0 level 1 prefix "_CALI_PRT raw "
  273 94036 MARK       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:XFX5xbM8B9qR10JG */ MARK and 0xfff0ffff
  131 12543 MARK       all  --  cali+  *       0.0.0.0/0            0.0.0.0/0            /* cali:EWMPb0zVROM-woQp */ MARK or 0x40000
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:V6ooGP15glg7wm91 */ mark match 0x40000/0x40000 rpfilter invert
  142 81493 cali-from-host-endpoint  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:RMTzKqp0j735XfY4 */ mark match 0x0/0x40000
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:T8-Zfumo2dKygI73 */ mark match 0x10000/0x10000
```

Figure 65.
Logging from the client browser to NGINX Ingress **– Prerouting chain**



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L cali-PREROUTING  -t mangle  -nv
Chain cali-PREROUTING (1 references)
 pkts bytes target     prot opt in     out     source               destination
    5   304 LOG        tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            state NEW LOG flags 0 level 1 prefix "_CALI_PRT_NEW_CONN mangle "
  635  247K LOG        all  --  *      *       0.0.0.0/0            0.0.0.0/0            LOG flags 0 level 1 prefix " CALI PRT CSTATE mangle "
  631  247K ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:6BJqBjBC7crtA-7- */ ctstate RELATED,ESTABLISHED
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:KX7AGNd6rMcDUai6 */ mark match 0x10000/0x10000
    5   304 cali-from-host-endpoint  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:wNH7KsA3ILKJBsY9 */
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:Cg96MgVuoPm7UMRo */ /* Host endpoint policy accepted packet. */ mark match 0x10000/0x10000
```

Figure 66.
Logging from the client browser to NGINX Ingress showing connection tracking integration **– Prerouting chain**



```
Every 1.0s: conntrack -L -p tcp --dport 80 --src-nat                                        iks-networking-example-cl-iks-networ-ac625

conntrack v1.4.4 (conntrack-tools): 5 flow entries have been shown.
tcp      6 3 CLOSE src=10.1.0.89 dst=10.1.110.105 sport=59545 dport=80 src=192.168.8.193 dst=10.1.110.107 sport=80 dport=59545 [ASSURED] mark=0 use=1
tcp      6 118 TIME_WAIT src=10.1.0.89 dst=10.1.110.105 sport=59693 dport=80 src=192.168.8.193 dst=10.1.110.107 sport=80 dport=59693 [ASSURED] mark=0 use=1
tcp      6 117 TIME_WAIT src=10.1.0.89 dst=10.1.110.105 sport=59694 dport=80 src=192.168.104.65 dst=192.168.8.192 sport=80 dport=59694 [ASSURED] mark=0 use=1
tcp      6 292 ESTABLISHED src=10.1.0.89 dst=10.1.110.105 sport=58927 dport=80 src=192.168.8.193 dst=10.1.110.107 sport=80 dport=58927 [ASSURED] mark=0 use=1
tcp      6 86392 ESTABLISHED src=10.1.0.89 dst=10.1.110.105 sport=59692 dport=80 src=192.168.8.193 dst=10.1.110.107 sport=80 dport=59692 [ASSURED] mark=0 use=1
```

SOURCE PORT = 58927

NGINX POD IP = 192.168.8.193

Figure 67.
Connection tracking output

- DNAT is used to rewrite the destination (pod) address.
- Since there are two NGINX ingress pods (one on each worker node), Kubernetes selects one at random.
    - In this example, the local ingress pod on worker 2 is selected.

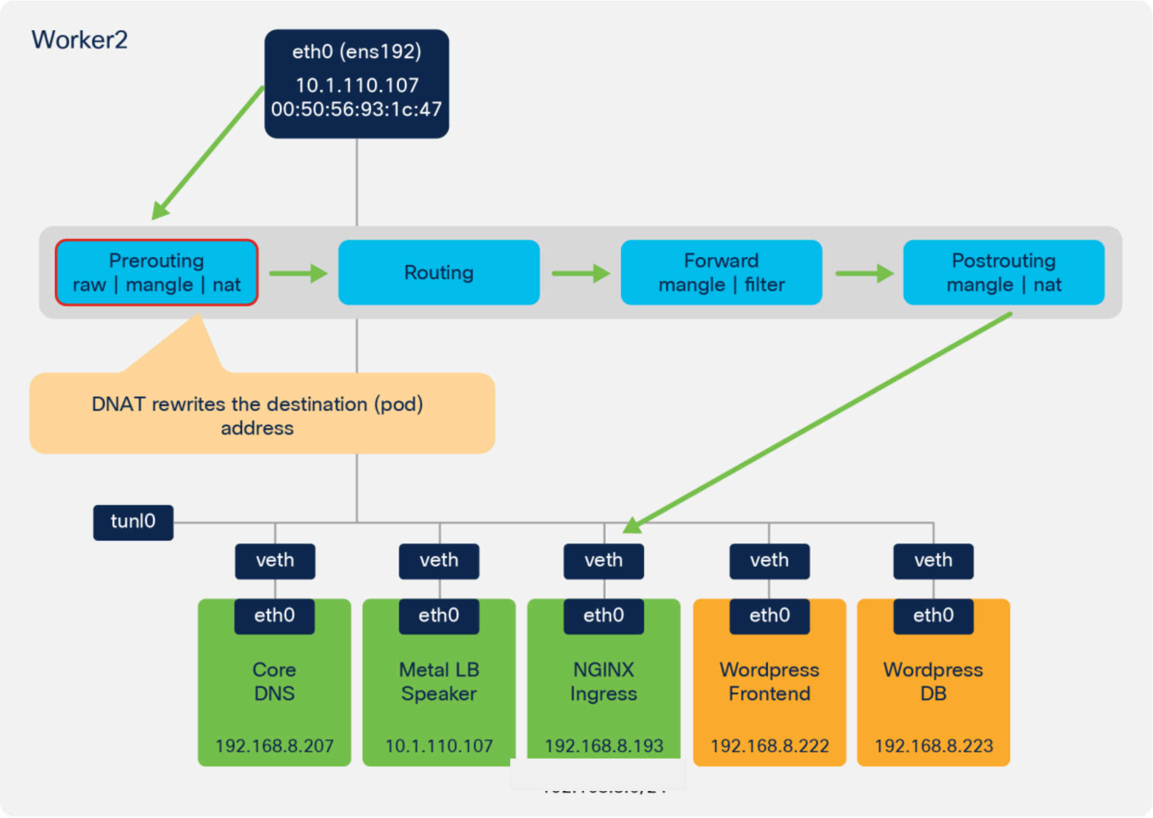
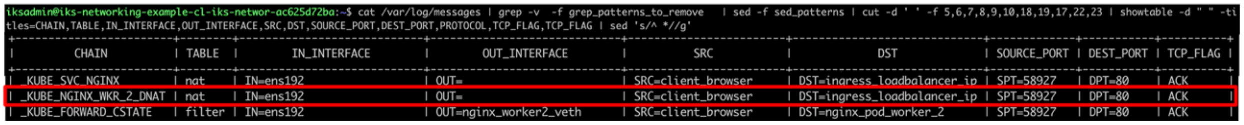
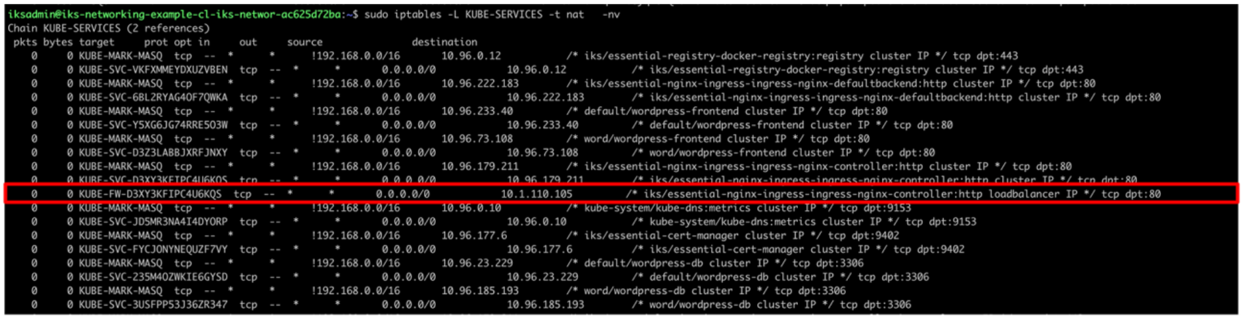


Figure 68.
Prerouting DNAT



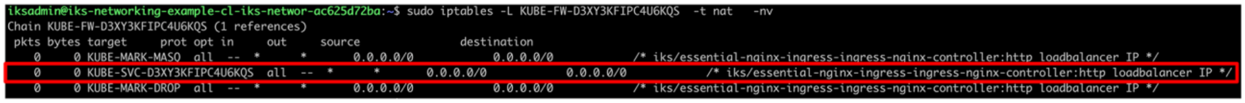
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -titles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+------------------------+--------+---------------------------+---------------------------+---------------------+---------------------------+-------------+-----------+----------+
|         CHAIN          | TABLE  |        IN_INTERFACE       |       OUT_INTERFACE       |         SRC         |            DST            | SOURCE_PORT | DEST_PORT | TCP_FLAG |
+------------------------+--------+---------------------------+---------------------------+---------------------+---------------------------+-------------+-----------+----------+
| _KUBE_SVC_NGINX        | nat    | IN=ens192                 | OUT=                      | SRC=client_browser  | DST=ingress_loadbalancer_ip | SPT=58927 | DPT=80    | ACK      |
| _KUBE_NGINX_WKR_2_DNAT | nat    | IN=ens192                 | OUT=                      | SRC=client_browser  | DST=ingress_loadbalancer_ip | SPT=58927 | DPT=80    | ACK      |
| _KUBE_FORWARD_CSTATE   | filter | IN=ens192                 | OUT=nginx_worker2_veth    | SRC=client_browser  | DST=nginx_pod_worker_2    | SPT=58927   | DPT=80    | ACK      |
```

Figure 69.
Logging showing **Prerouting** DNAT



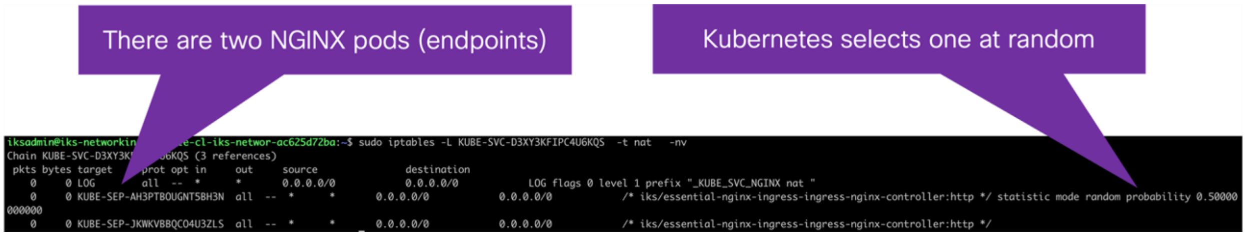
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SERVICES -t nat   -nv
Chain KUBE-SERVICES (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.0.12           /* iks/essential-registry-docker-registry:registry cluster IP */ tcp dpt:443
    0     0 KUBE-SVC-VKFXMMEYDXUZVBEN  tcp  --  *      *       0.0.0.0/0            10.96.0.12           /* iks/essential-registry-docker-registry:registry cluster IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.222.183        /* iks/essential-nginx-ingress-ingress-nginx-defaultbackend:http cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-6BL2RYAG4OF7QWKA  tcp  --  *      *       0.0.0.0/0            10.96.222.183        /* iks/essential-nginx-ingress-ingress-nginx-defaultbackend:http cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.233.40         /* default/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-Y5XG6JG74RRE5O3W  tcp  --  *      *       0.0.0.0/0            10.96.233.40         /* default/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.73.108         /* word/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-D3Z3LABBJXRFJNXY  tcp  --  *      *       0.0.0.0/0            10.96.73.108         /* word/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-controller:http cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-D3XY3KFIPC4U6KQS  tcp  --  *      *       0.0.0.0/0            10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-controller:http cluster IP */ tcp dpt:80
    0     0 KUBE-FW-D3XY3KFIPC4U6KQS  tcp  --  *      *       0.0.0.0/0            10.1.110.105         /* iks/essential-nginx-ingress-ingress-nginx-controller:http loadbalancer IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.0.10           /* kube-system/kube-dns:metrics cluster IP */ tcp dpt:9153
    0     0 KUBE-SVC-JD5MR3NA4I4DYORP  tcp  --  *      *       0.0.0.0/0            10.96.0.10           /* kube-system/kube-dns:metrics cluster IP */ tcp dpt:9153
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.177.6          /* iks/essential-cert-manager cluster IP */ tcp dpt:9402
    0     0 KUBE-SVC-FYCJONYNEQUZF7VY  tcp  --  *      *       0.0.0.0/0            10.96.177.6          /* iks/essential-cert-manager cluster IP */ tcp dpt:9402
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.23.229         /* default/wordpress-db cluster IP */ tcp dpt:3306
    0     0 KUBE-SVC-235M4OZWKIE6GYSD  tcp  --  *      *       0.0.0.0/0            10.96.23.229         /* default/wordpress-db cluster IP */ tcp dpt:3306
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.185.193        /* word/wordpress-db cluster IP */ tcp dpt:3306
    0     0 KUBE-SVC-3USFPP53J36ZR347  tcp  --  *      *       0.0.0.0/0            10.96.185.193        /* word/wordpress-db cluster IP */ tcp dpt:3306
```

Figure 70.
Output from IPTables showing the rule for NGINX



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-FW-D3XY3KFIPC4U6KQS  -t nat   -nv
Chain KUBE-FW-D3XY3KFIPC4U6KQS (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http loadbalancer IP */
    0     0 KUBE-SVC-D3XY3KFIPC4U6KQS  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http loadbalancer IP */
    0     0 KUBE-MARK-DROP  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http loadbalancer IP */
```

Figure 71.
Output from IPTables showing the rule for NGINX

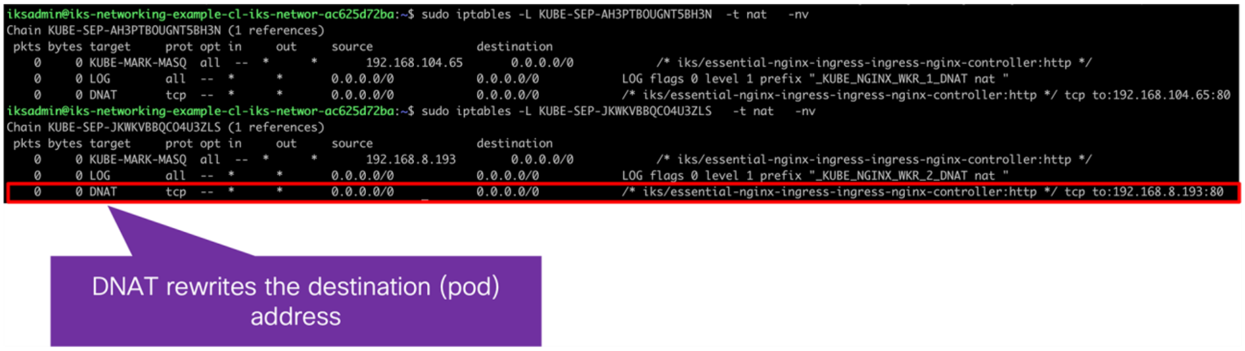


There are two NGINX pods (endpoints)

Kubernetes selects one at random

```
iksadmin@iks-networkin    e-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SVC-D3XY3KFIPC4U6KQS  -t nat   -nv
Chain KUBE-SVC-D3XY3K    6KQS (3 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 LOG        all  --  *      *       0.0.0.0/0            0.0.0.0/0            LOG flags 0 level 1 prefix "_KUBE_SVC_NGINX nat "
    0     0 KUBE-SEP-AH3PTBOUGNT5BH3N  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http */ statistic mode random probability 0.50000
000000
    0     0 KUBE-SEP-JKWKVBBQCO4U3ZLS  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http */
```

Figure 72.
Output from IPTables showing the NGINX pods



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SEP-AH3PTBOUGNT5BH3N  -t nat   -nv
Chain KUBE-SEP-AH3PTBOUGNT5BH3N (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       192.168.104.65       0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http */
    0     0 LOG        all  --  *      *       0.0.0.0/0            0.0.0.0/0            LOG flags 0 level 1 prefix "_KUBE_NGINX_WKR_1_DNAT nat "
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http */ tcp to:192.168.104.65:80
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SEP-JKWKVBBQCO4U3ZLS   -t nat   -nv
Chain KUBE-SEP-JKWKVBBQCO4U3ZLS (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       192.168.8.193        0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http */
    0     0 LOG        all  --  *      *       0.0.0.0/0            0.0.0.0/0            LOG flags 0 level 1 prefix "_KUBE_NGINX_WKR_2_DNAT nat "
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http */ tcp to:192.168.8.193:80
```

DNAT rewrites the destination (pod) address

Figure 73.
Output from IPTables showing the DNAT rule for the NGINX pod

- A forwarding decision is made based on routing information. Since the NGINX destination pod is local, the traffic is dispatched to the relevant Calico veth interface on worker 2.
    - The Calico IPTables rules accept the traffic destined to the pod.

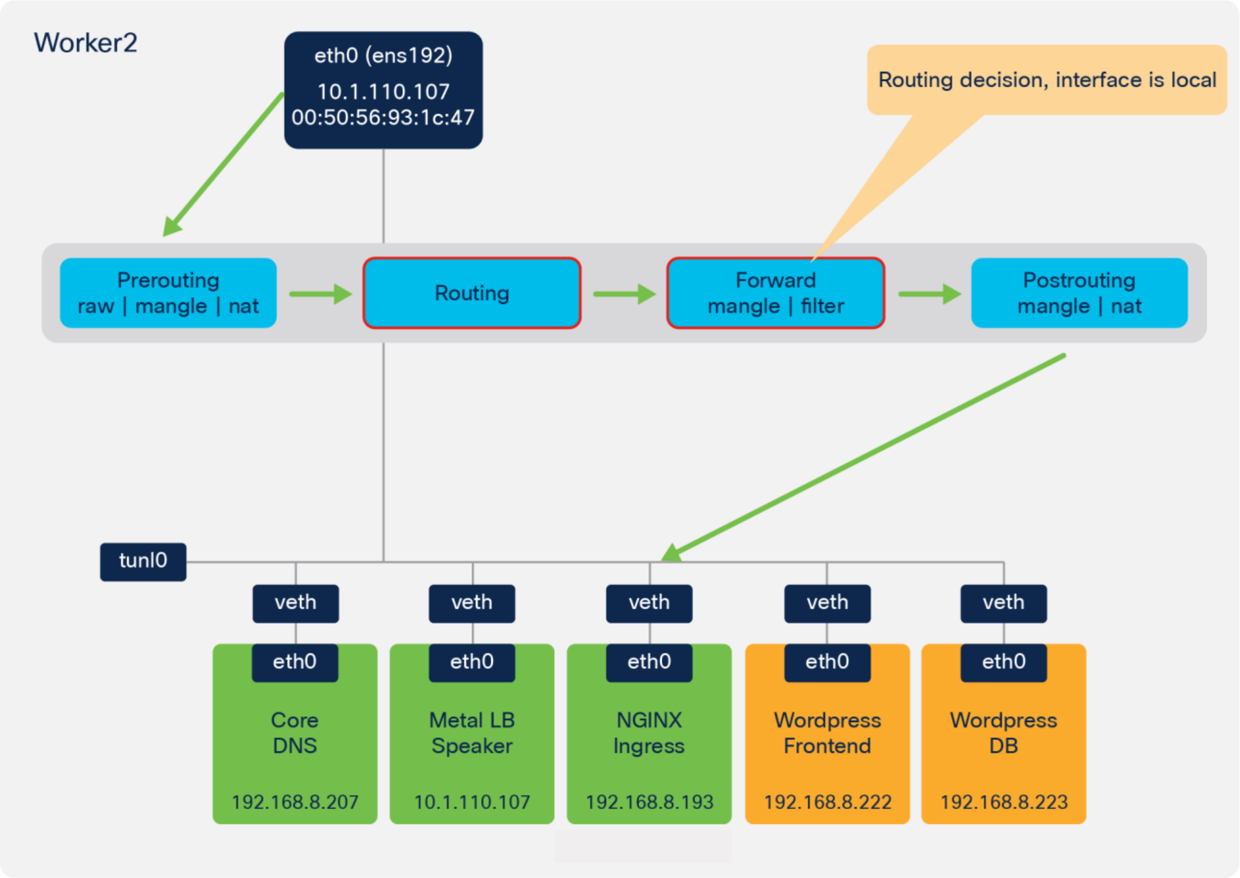
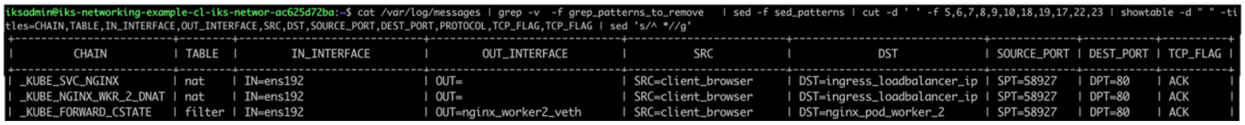
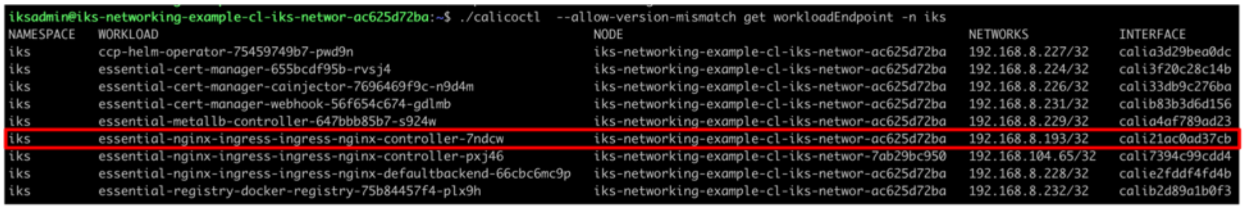


Figure 74.
Client browser to NGINX ingress – **routing** and **forward**



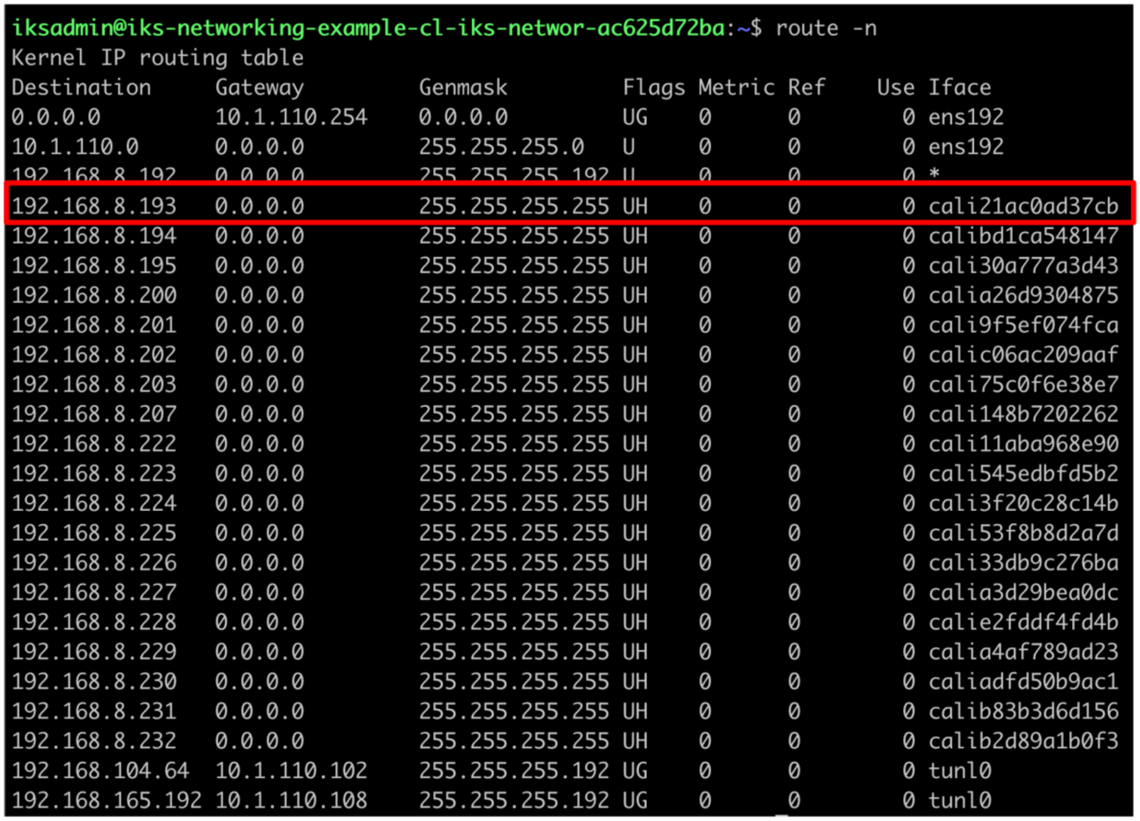
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+-----------------------+--------+----------------------------+----------------------------+-----------------------+----------------------------+-------------+-----------+----------+
|         CHAIN         | TABLE  |        IN_INTERFACE        |       OUT_INTERFACE        |          SRC          |            DST             | SOURCE_PORT | DEST_PORT | TCP_FLAG |
+-----------------------+--------+----------------------------+----------------------------+-----------------------+----------------------------+-------------+-----------+----------+
| _KUBE_SVC_NGINX       | nat    | IN=ens192                  | OUT=                       | SRC=client_browser    | DST=ingress_loadbalancer_ip | SPT=58927  | DPT=80    | ACK      |
| _KUBE_NGINX_WKR_2_DNAT | nat   | IN=ens192                  | OUT=                       | SRC=client_browser    | DST=ingress_loadbalancer_ip | SPT=58927  | DPT=80    | ACK      |
| _KUBE_FORWARD_CSTATE  | filter | IN=ens192                  | OUT=nginx_worker2_veth     | SRC=client_browser    | DST=nginx_pod_worker_2     | SPT=58927   | DPT=80    | ACK      |
```

Figure 75.
Logging from the client browser to NGINX Ingress **- routing and forward chain**



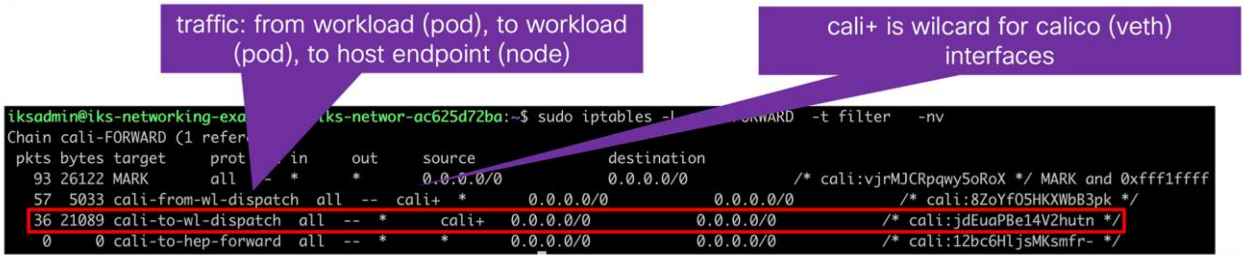
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ ./calicoctl  --allow-version-mismatch get workloadEndpoint -n iks
NAMESPACE   WORKLOAD                                                NODE                                           NETWORKS           INTERFACE
iks         ccp-helm-operator-75459749b7-pwd9n                      iks-networking-example-cl-iks-networ-ac625d72ba   192.168.8.227/32   calia3d29bea0dc
iks         essential-cert-manager-655bcdf95b-rvsj4                 iks-networking-example-cl-iks-networ-ac625d72ba   192.168.8.224/32   cali3f20c28c14b
iks         essential-cert-manager-cainjector-7696469f9c-n9d4m      iks-networking-example-cl-iks-networ-ac625d72ba   192.168.8.226/32   cali33db9c276ba
iks         essential-cert-manager-webhook-56f654c674-gdlmb         iks-networking-example-cl-iks-networ-ac625d72ba   192.168.8.231/32   calib83b3d6d156
iks         essential-metallb-controller-647bbb85b7-s924w           iks-networking-example-cl-iks-networ-ac625d72ba   192.168.8.229/32   calia4af789ad23
iks         essential-nginx-ingress-ingress-nginx-controller-7ndcw  iks-networking-example-cl-iks-networ-ac625d72ba   192.168.8.193/32   cali21ac0ad37cb
iks         essential-nginx-ingress-ingress-nginx-controller-pxj46  iks-networking-example-cl-iks-networ-7ab29bc950   192.168.104.65/32  cali7394c99cdd4
iks         essential-nginx-ingress-ingress-nginx-defaultbackend-66cbc6mc9p   iks-networking-example-cl-iks-networ-ac625d72ba   192.168.8.228/32   calie2fddf4fd4b
iks         essential-registry-docker-registry-75b84457f4-plx9h     iks-networking-example-cl-iks-networ-ac625d72ba   192.168.8.232/32   calib2d89a1b0f3
```

Figure 76.
Output from **calicoctl** tool showing Kubernetes pod, node, and **veth** mapping



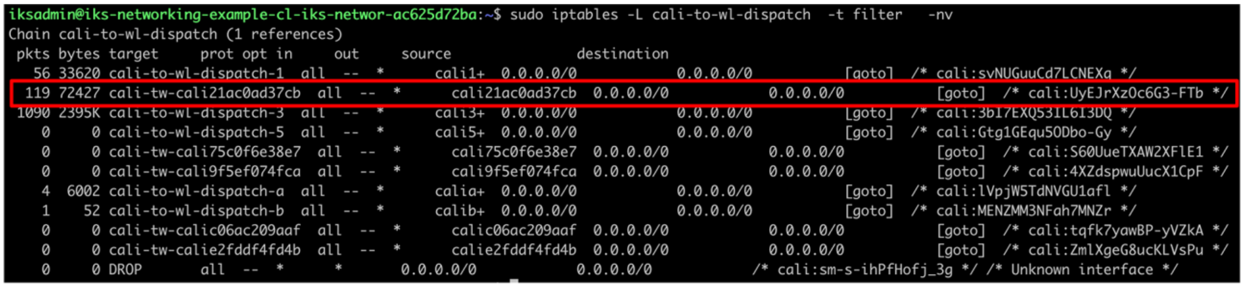
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.1.110.254    0.0.0.0         UG    0      0        0 ens192
10.1.110.0      0.0.0.0         255.255.255.0   U     0      0        0 ens192
192.168.8.192   0.0.0.0         255.255.255.192 U     0      0        0 *
192.168.8.193   0.0.0.0         255.255.255.255 UH    0      0        0 cali21ac0ad37cb
192.168.8.194   0.0.0.0         255.255.255.255 UH    0      0        0 calibd1ca548147
192.168.8.195   0.0.0.0         255.255.255.255 UH    0      0        0 cali30a777a3d43
192.168.8.200   0.0.0.0         255.255.255.255 UH    0      0        0 calia26d9304875
192.168.8.201   0.0.0.0         255.255.255.255 UH    0      0        0 cali9f5ef074fca
192.168.8.202   0.0.0.0         255.255.255.255 UH    0      0        0 calic06ac209aaf
192.168.8.203   0.0.0.0         255.255.255.255 UH    0      0        0 cali75c0f6e38e7
192.168.8.207   0.0.0.0         255.255.255.255 UH    0      0        0 cali148b7202262
192.168.8.222   0.0.0.0         255.255.255.255 UH    0      0        0 cali11aba968e90
192.168.8.223   0.0.0.0         255.255.255.255 UH    0      0        0 cali545edbfd5b2
192.168.8.224   0.0.0.0         255.255.255.255 UH    0      0        0 cali3f20c28c14b
192.168.8.225   0.0.0.0         255.255.255.255 UH    0      0        0 cali53f8b8d2a7d
192.168.8.226   0.0.0.0         255.255.255.255 UH    0      0        0 cali33db9c276ba
192.168.8.227   0.0.0.0         255.255.255.255 UH    0      0        0 calia3d29bea0dc
192.168.8.228   0.0.0.0         255.255.255.255 UH    0      0        0 calie2fddf4fd4b
192.168.8.229   0.0.0.0         255.255.255.255 UH    0      0        0 calia4af789ad23
192.168.8.230   0.0.0.0         255.255.255.255 UH    0      0        0 caliadfd50b9ac1
192.168.8.231   0.0.0.0         255.255.255.255 UH    0      0        0 calib83b3d6d156
192.168.8.232   0.0.0.0         255.255.255.255 UH    0      0        0 calib2d89a1b0f3
192.168.104.64  10.1.110.102    255.255.255.192 UG    0      0        0 tunl0
192.168.165.192 10.1.110.108    255.255.255.192 UG    0      0        0 tunl0
```

Figure 77.
Output from routing table showing local **veth** interface


traffic: from workload (pod), to workload (pod), to host endpoint (node)
cali+ is wilcard for calico (veth) interfaces

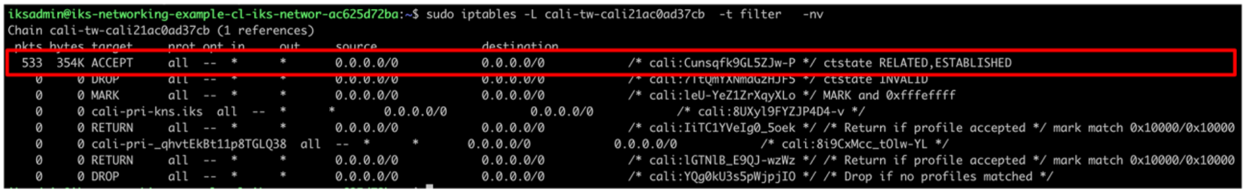
```
iksadmin@iks-networking-exa      iks-networ-ac625d72ba:~$ sudo iptables -L      ORWARD  -t filter   -nv
Chain cali-FORWARD (1 refer
 pkts bytes target     prot   in     out     source               destination
   93 26122 MARK       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:vjrMJCRpqwy5oRoX */ MARK and 0xfff1ffff
   57  5033 cali-from-wl-dispatch  all  --  cali+  *       0.0.0.0/0            0.0.0.0/0            /* cali:8ZoYfO5HKXWbB3pk */
   36 21089 cali-to-wl-dispatch  all  --  *      cali+   0.0.0.0/0            0.0.0.0/0            /* cali:jdEuaPBe14V2hutn */
    0     0 cali-to-hep-forward  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:12bc6HljsMKsmfr- */
```

Figure 78.
IPTables rules for **Calico** local **veth** interfaces



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L cali-to-wl-dispatch  -t filter   -nv
Chain cali-to-wl-dispatch (1 references)
 pkts bytes target     prot opt in     out     source               destination
   56 33620 cali-to-wl-dispatch-1  all  --  *      cali1+  0.0.0.0/0            0.0.0.0/0            [goto]  /* cali:svNUGuuCd7LCNEXq */
  119 72427 cali-tw-cali21ac0ad37cb  all  --  *      cali21ac0ad37cb  0.0.0.0/0            0.0.0.0/0            [goto]  /* cali:UyEJrXzOc6G3-FTb */
 1090 2395K cali-to-wl-dispatch-3  all  --  *      cali3+  0.0.0.0/0            0.0.0.0/0            [goto]  /* cali:3bI7EXQ53IL6I3DQ */
    0     0 cali-to-wl-dispatch-5  all  --  *      cali5+  0.0.0.0/0            0.0.0.0/0            [goto]  /* cali:Gtg1GEqu5ODbo-Gy */
    0     0 cali-tw-cali75c0f6e38e7  all  --  *      cali75c0f6e38e7  0.0.0.0/0            0.0.0.0/0            [goto]  /* cali:S60UueTXAW2XFlE1 */
    0     0 cali-tw-cali9f5ef074fca  all  --  *      cali9f5ef074fca  0.0.0.0/0            0.0.0.0/0            [goto]  /* cali:4XZdspwuUucX1CpF */
    4  6002 cali-to-wl-dispatch-a  all  --  *      calia+  0.0.0.0/0            0.0.0.0/0            [goto]  /* cali:lVpjW5TdNVGU1afl */
    1    52 cali-to-wl-dispatch-b  all  --  *      calib+  0.0.0.0/0            0.0.0.0/0            [goto]  /* cali:MENZMM3NFah7MNZr */
    0     0 cali-tw-calic06ac209aaf  all  --  *      calic06ac209aaf  0.0.0.0/0            0.0.0.0/0            [goto]  /* cali:tqfk7yawBP-yVZkA */
    0     0 cali-tw-calie2fddf4fd4b  all  --  *      calie2fddf4fd4b  0.0.0.0/0            0.0.0.0/0            [goto]  /* cali:ZmlXgeG8ucKLVsPu */
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:sm-s-ihPfHofj_3g */ /* Unknown interface */
```

Figure 79.
IPTables rules for **Calico** local **veth** interfaces



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L cali-tw-cali21ac0ad37cb  -t filter   -nv
Chain cali-tw-cali21ac0ad37cb (1 references)
 pkts bytes target     prot opt in     out     source               destination
  533  354K ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:Cunsqfk9GL5ZJw-P */ ctstate RELATED,ESTABLISHED
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:7TtQmYXNmaGzHJF5 */ ctstate INVALID
    0     0 MARK       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:leU-YeZ1ZrXqyXLo */ MARK and 0xfffeffff
    0     0 cali-pri-kns.iks  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:8UXyl9FYZJP4D4-v */
    0     0 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:IiTC1YVeIg0_5oek */ /* Return if profile accepted */ mark match 0x10000/0x10000
    0     0 cali-pri-_qhvtEkBt11p8TGLQ38  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:8i9CxMcc_tOlw-YL */
    0     0 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:lGTNlB_E9QJ-wzWz */ /* Return if profile accepted */ mark match 0x10000/0x10000
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:YQg0kU3s5pWjpjIO */ /* Drop if no profiles matched */
```

Figure 80.
IPTables rules for **Calico** local **veth** interfaces

- Traffic traverses the **NAT** table in the **Postrouting** chain and sent to the veth interface on the NGINX worker 2 pod.

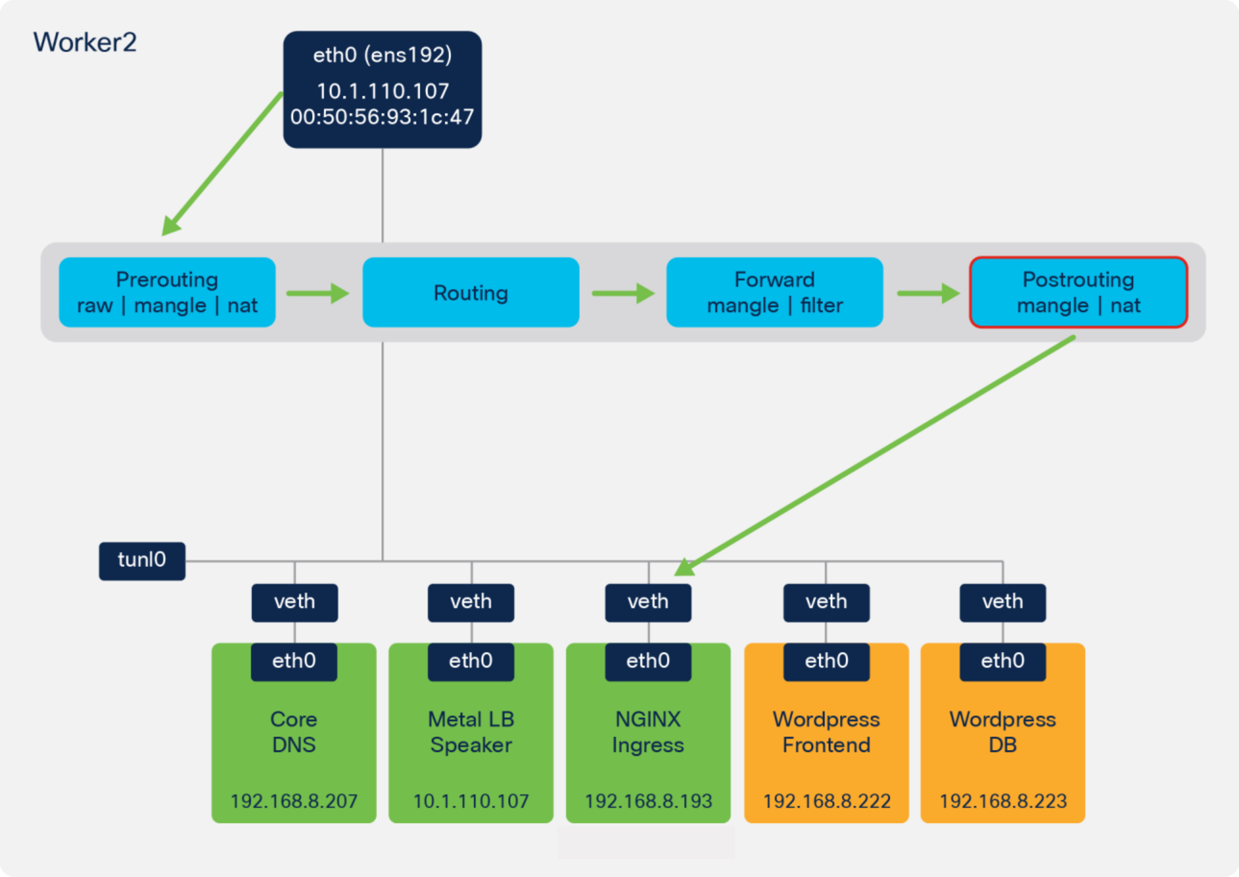
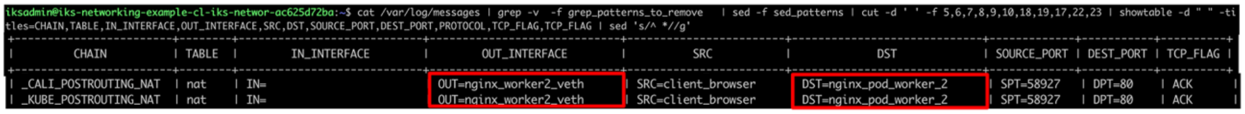
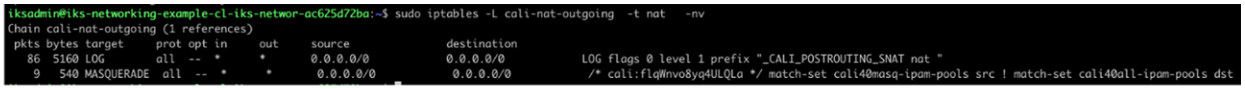


Figure 81.
Postrouting to NGINX ingress



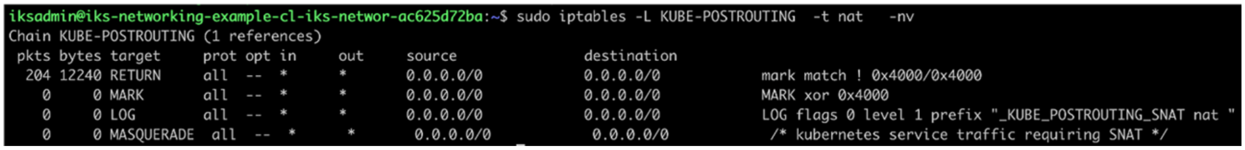
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+-----------------------+--------+-----------------------------+-----------------------------+-------------------------+------------------------------+--------------+------------+-----------+
|         CHAIN         | TABLE  |         IN_INTERFACE        |        OUT_INTERFACE        |           SRC           |             DST              | SOURCE_PORT  | DEST_PORT  | TCP_FLAG  |
+-----------------------+--------+-----------------------------+-----------------------------+-------------------------+------------------------------+--------------+------------+-----------+
| _CALI_POSTROUTING_NAT | nat    | IN=                         | OUT=nginx_worker2_veth      | SRC=client_browser      | DST=nginx_pod_worker_2       | SPT=58927    | DPT=80     | ACK       |
| _KUBE_POSTROUTING_NAT | nat    | IN=                         | OUT=nginx_worker2_veth      | SRC=client_browser      | DST=nginx_pod_worker_2       | SPT=58927    | DPT=80     | ACK       |
```

Figure 82.
Logging showing NGINX Ingress **- postrouting chain**



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L cali-nat-outgoing  -t nat   -nv
Chain cali-nat-outgoing (1 references)
 pkts bytes target     prot opt in     out     source               destination
   86  5160 LOG        all  --  *      *       0.0.0.0/0            0.0.0.0/0            LOG flags 0 level 1 prefix "_CALI_POSTROUTING_SNAT nat "
    9   540 MASQUERADE  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:flqWnvo8yq4ULQLa */ match-set cali40masq-ipam-pools src ! match-set cali40all-ipam-pools dst
```

Figure 83.
IPTables rules showing the **postrouting** chain



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-POSTROUTING  -t nat   -nv
Chain KUBE-POSTROUTING (1 references)
 pkts bytes target     prot opt in     out     source               destination
  204 12240 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0            mark match ! 0x4000/0x4000
    0     0 MARK       all  --  *      *       0.0.0.0/0            0.0.0.0/0            MARK xor 0x4000
    0     0 LOG        all  --  *      *       0.0.0.0/0            0.0.0.0/0            LOG flags 0 level 1 prefix "_KUBE_POSTROUTING_SNAT nat "
    0     0 MASQUERADE  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes service traffic requiring SNAT */
```

Figure 84.
IPTables rules showing the **postrouting** chain

- Traffic is sent from the NGINX pod to the Wordpress frontend pod based on the NGINX ingress configuration (stored within the **nginx.conf** file on the ingress controller pods).

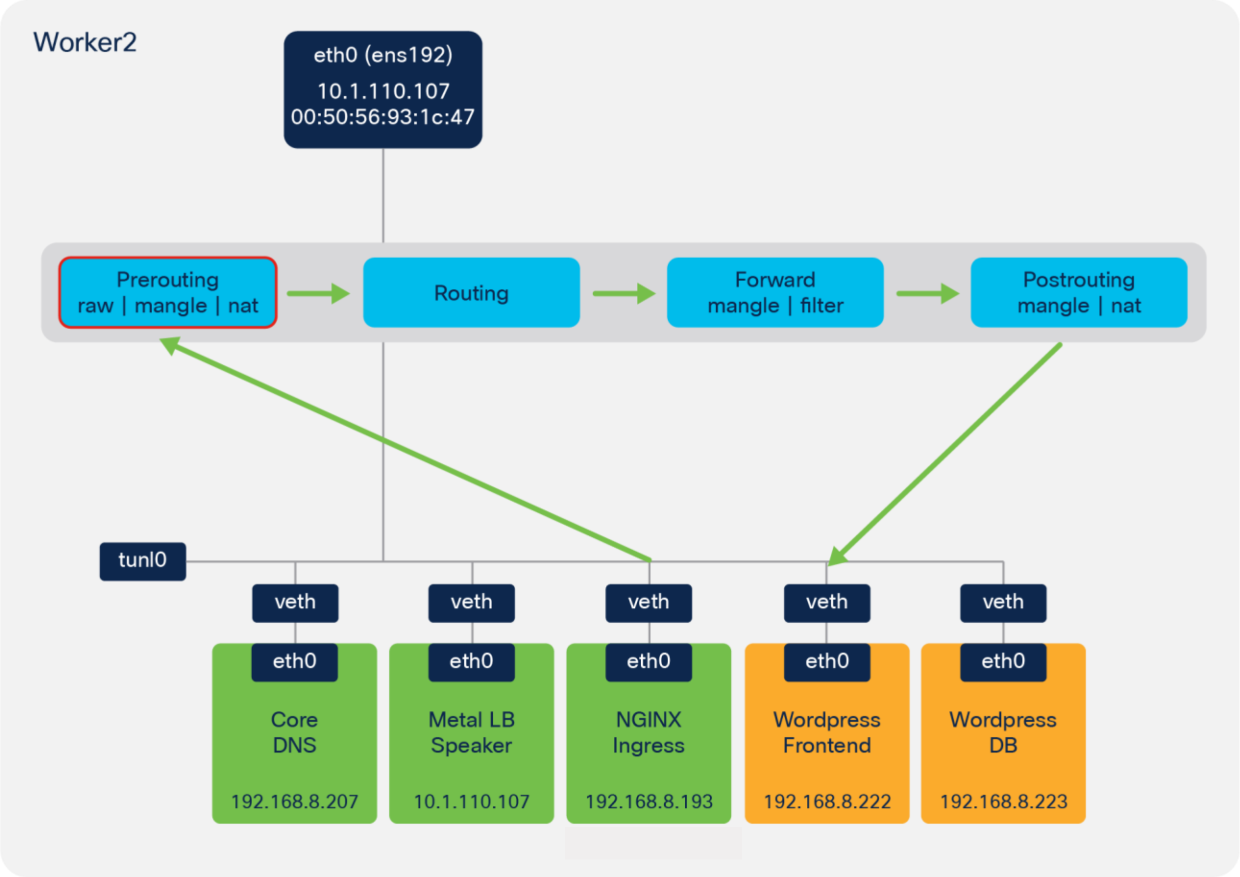
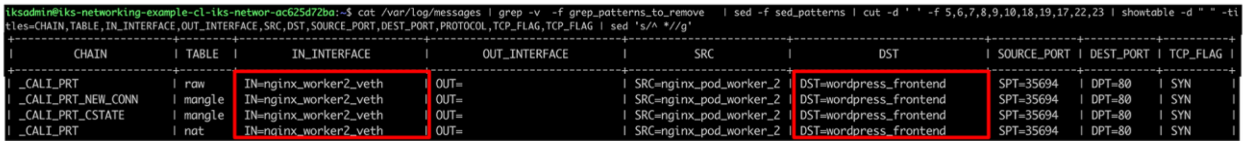
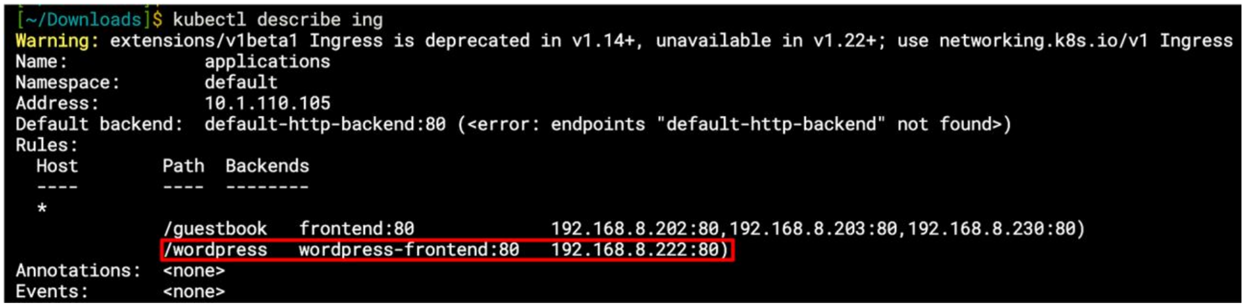


Figure 85.
NGINX ingress to Wordpress frontend



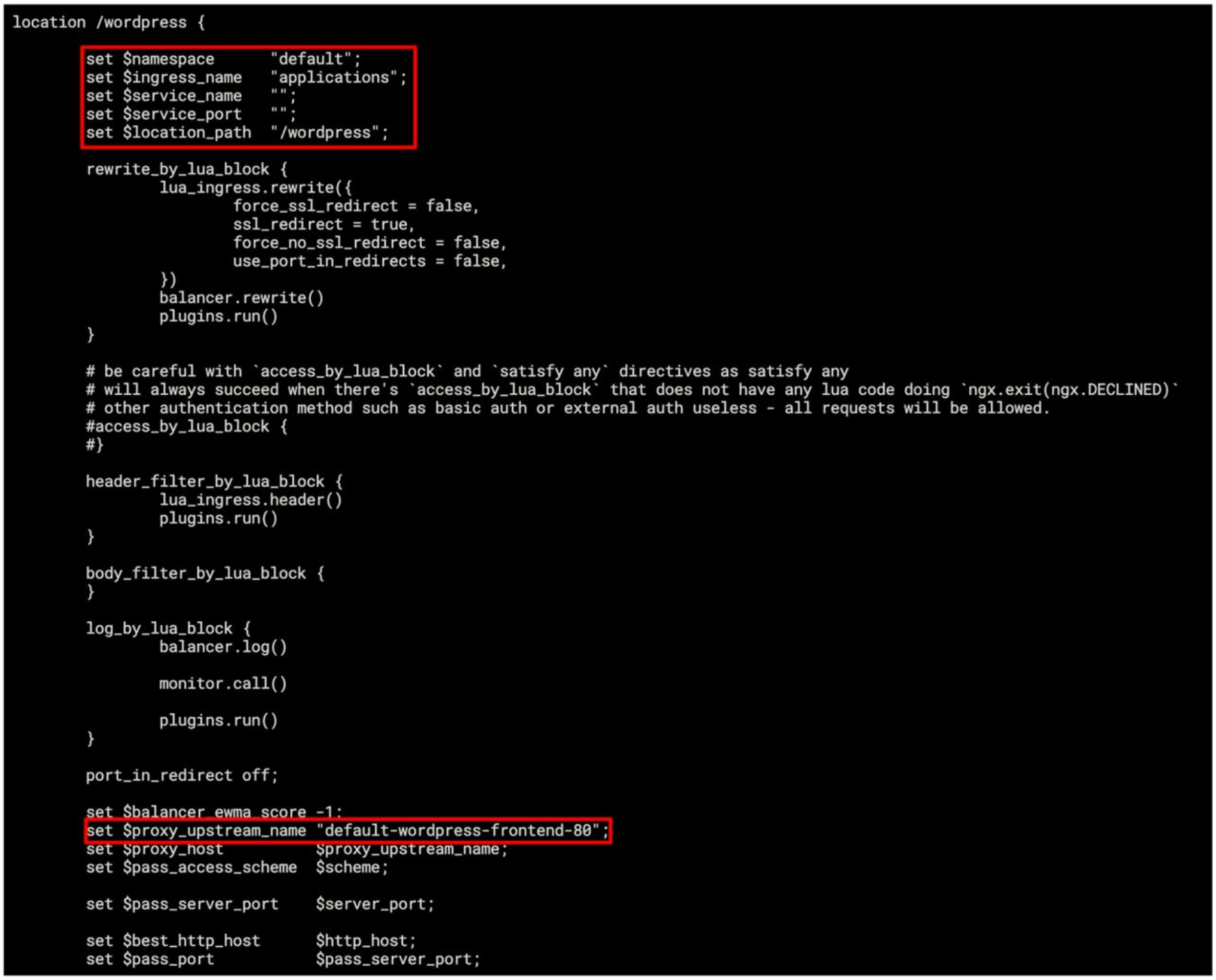
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+-----------------------+--------+--------------------------+--------------------------+-----------------------+---------------------------+-------------+-----------+----------+
|         CHAIN         | TABLE  |       IN_INTERFACE       |       OUT_INTERFACE      |          SRC          |            DST            | SOURCE_PORT | DEST_PORT | TCP_FLAG |
+-----------------------+--------+--------------------------+--------------------------+-----------------------+---------------------------+-------------+-----------+----------+
| _CALI_PRT             | raw    | IN=nginx_worker2_veth    | OUT=                     | SRC=nginx_pod_worker_2 | DST=wordpress_frontend     SPT=35694   | DPT=80    | SYN      |
| _CALI_PRT_NEW_CONN    | mangle | IN=nginx_worker2_veth    | OUT=                     | SRC=nginx_pod_worker_2 | DST=wordpress_frontend     SPT=35694   | DPT=80    | SYN      |
| _CALI_PRT_CSTATE      | mangle | IN=nginx_worker2_veth    | OUT=                     | SRC=nginx_pod_worker_2 | DST=wordpress_frontend     SPT=35694   | DPT=80    | SYN      |
| _CALI_PRT             | nat    | IN=nginx_worker2_veth    | OUT=                     | SRC=nginx_pod_worker_2 | DST=wordpress_frontend     SPT=35694   | DPT=80    | SYN      |
```

Figure 86.
Logging showing NGINX ingress to Wordpress frontend – **prerouting chain**



```
[~/Downloads]$ kubectl describe ing
Warning: extensions/v1beta1 Ingress is deprecated in v1.14+, unavailable in v1.22+; use networking.k8s.io/v1 Ingress
Name:             applications
Namespace:        default
Address:          10.1.110.105
Default backend:  default-http-backend:80 (<error: endpoints "default-http-backend" not found>)
Rules:
  Host        Path  Backends
  ----        ----  --------
  *
              /guestbook   frontend:80             192.168.8.202:80,192.168.8.203:80,192.168.8.230:80)
              /wordpress   wordpress-frontend:80   192.168.8.222:80)
Annotations:  <none>
Events:       <none>
```

Figure 87.
Output from **kubectl describe ingress** command



```
location /wordpress {

        set $namespace      "default";
        set $ingress_name   "applications";
        set $service_name   "";
        set $service_port   "";
        set $location_path  "/wordpress";

        rewrite_by_lua_block {
                lua_ingress.rewrite({
                        force_ssl_redirect = false,
                        ssl_redirect = true,
                        force_no_ssl_redirect = false,
                        use_port_in_redirects = false,
                })
                balancer.rewrite()
                plugins.run()
        }

        # be careful with `access_by_lua_block` and `satisfy any` directives as satisfy any
        # will always succeed when there's `access_by_lua_block` that does not have any lua code doing `ngx.exit(ngx.DECLINED)`
        # other authentication method such as basic auth or external auth useless - all requests will be allowed.
        #access_by_lua_block {
        #}

        header_filter_by_lua_block {
                lua_ingress.header()
                plugins.run()
        }

        body_filter_by_lua_block {
        }

        log_by_lua_block {
                balancer.log()

                monitor.call()

                plugins.run()
        }

        port_in_redirect off;

        set $balancer_ewma_score -1;
        set $proxy_upstream_name "default-wordpress-frontend-80";
        set $proxy_host          $proxy_upstream_name;
        set $pass_access_scheme  $scheme;

        set $pass_server_port    $server_port;

        set $best_http_host      $http_host;
        set $pass_port           $pass_server_port;
```

Figure 88.
nginx.conf showing the wordpress ingress rules

- Traffic is forwarded from the NGINX pod veth interface to the veth interface on the Wordpress frontend pod.

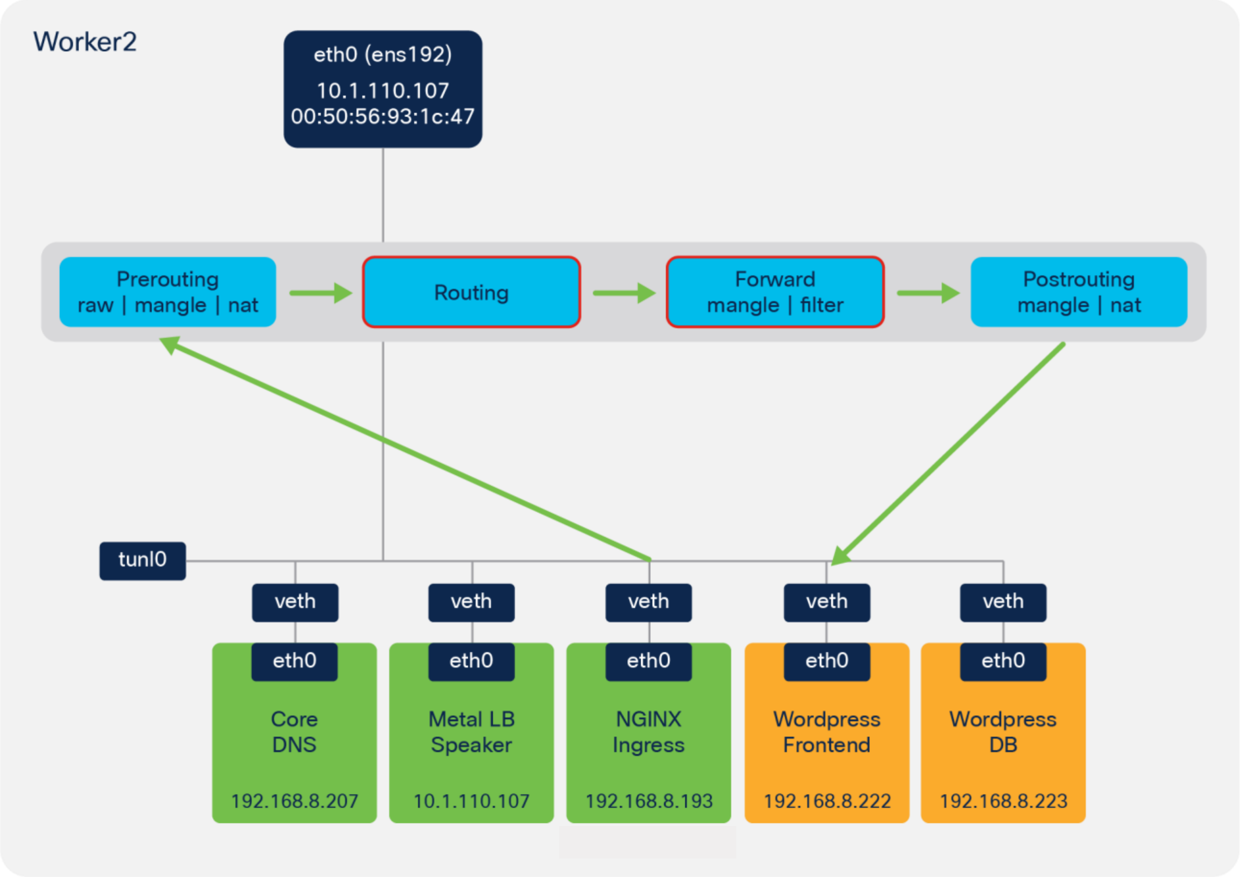
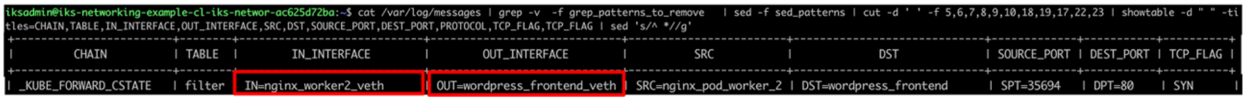


Figure 89.
NGINX ingress to Wordpress frontend – **routing** and **forward**



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+-----------------------+--------+-------------------------+-------------------------+----------------------+------------------------+-------------+-----------+----------+
|         CHAIN         | TABLE  |       IN_INTERFACE      |      OUT_INTERFACE      |         SRC          |          DST           | SOURCE_PORT | DEST_PORT | TCP_FLAG |
+-----------------------+--------+-------------------------+-------------------------+----------------------+------------------------+-------------+-----------+----------+
| _KUBE_FORWARD_CSTATE  | filter | IN=nginx_worker2_veth   | OUT=wordpress_frontend_veth | SRC=nginx_pod_worker_2 | DST=wordpress_frontend | SPT=35694   | DPT=80    | SYN      |
```

Figure 90.
Logging NGINX ingress to Wordpress frontend – **routing** and **forward chains**

- Traffic traverses the IPTables **NAT** table in the **Postrouting** chain to forward the traffic onto the Wordpress frontend **veth** interface.

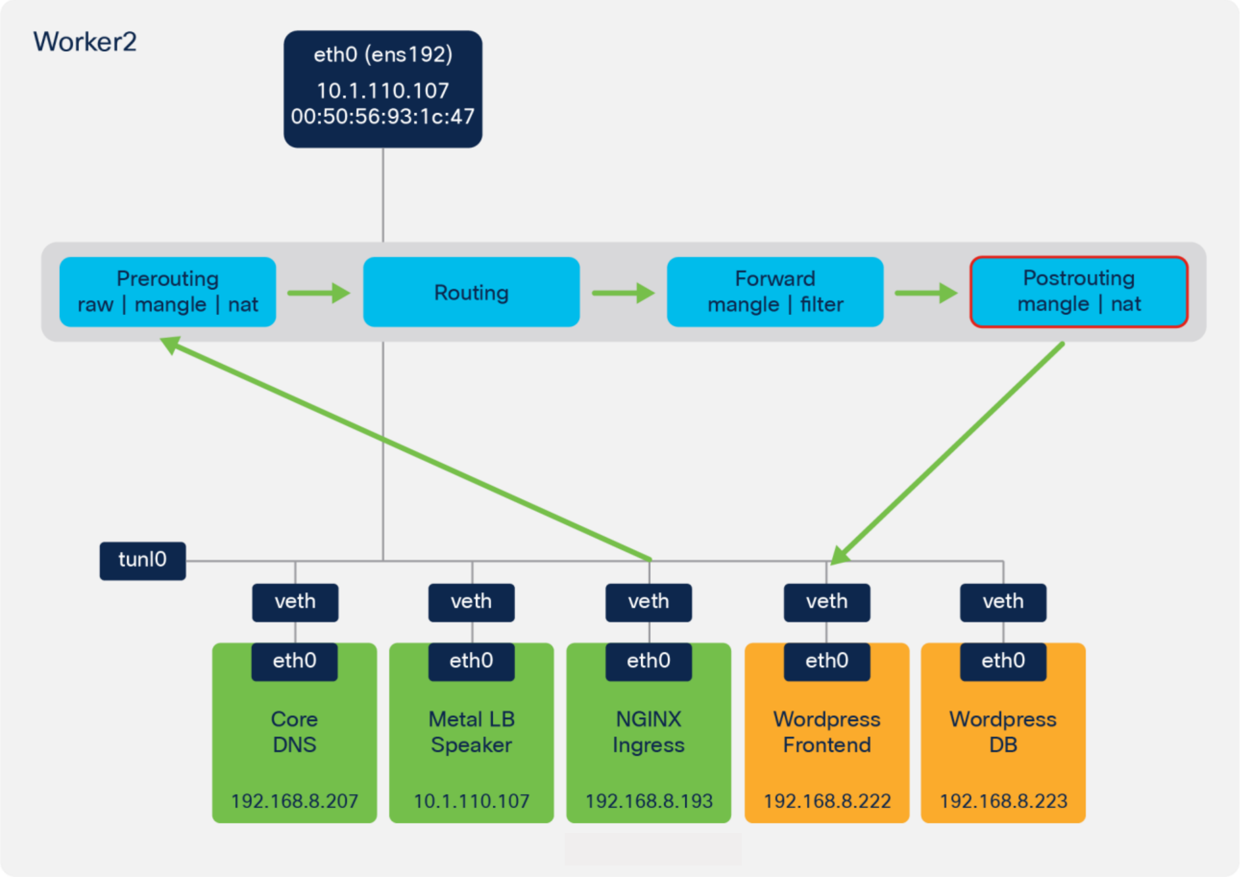
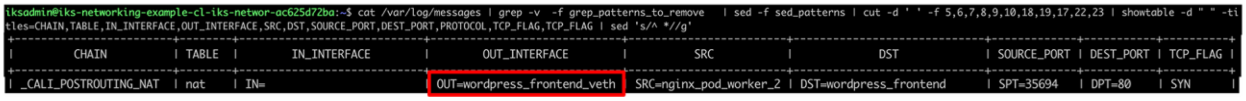


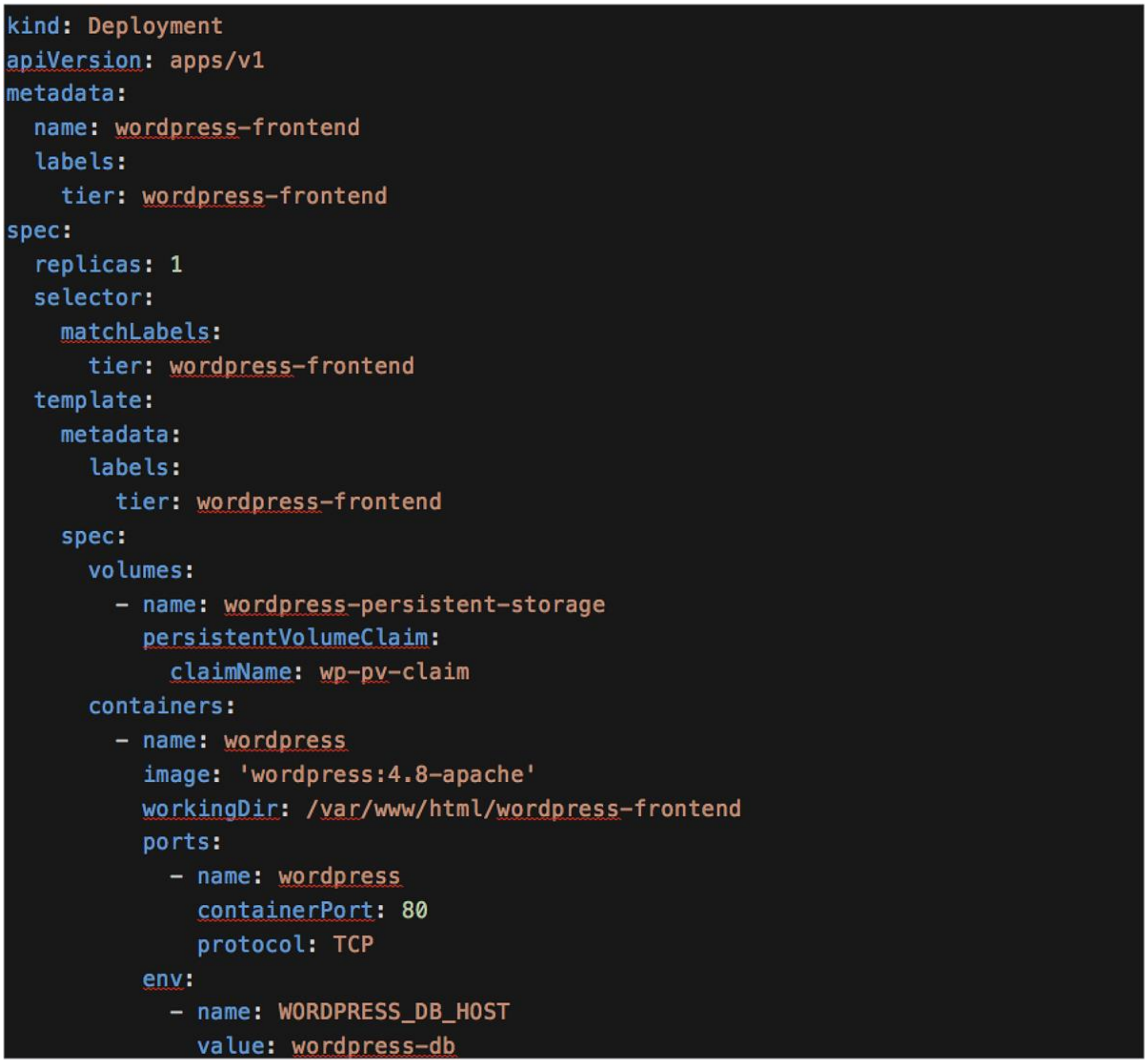
Figure 91.
NGINX ingress to Wordpress frontend – **postrouting**



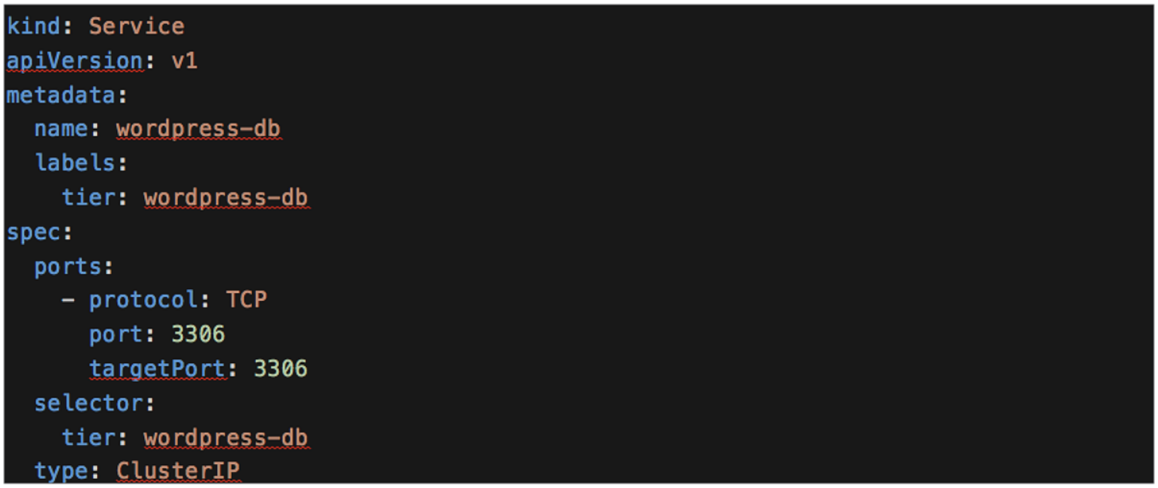
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+-----------------------+--------+----------------------------+----------------------------+-----------------------+----------------------------+-------------+-----------+----------+
|         CHAIN         | TABLE  |        IN_INTERFACE        |        OUT_INTERFACE       |          SRC          |            DST             | SOURCE_PORT | DEST_PORT | TCP_FLAG |
+-----------------------+--------+----------------------------+----------------------------+-----------------------+----------------------------+-------------+-----------+----------+
| _CALI_POSTROUTING_NAT | nat    | IN=                        | OUT=wordpress_frontend_veth | SRC=nginx_pod_worker_2 | DST=wordpress_frontend     | SPT=35694   | DPT=80    | SYN      |
```

Figure 92.
Logging NGINX ingress to Wordpress frontend – **postrouting chain**

- The Wordpress frontend sends traffic to the Wordpress DB service.
- The Wordpress frontend deployment has the DB host (**WORDPRESS_DB_HOST**) configured, pointing to the DB Kubernetes Service, **wordpress-db**
- The pod resolves the service name to the service IP address by sending the DNS query to a **Kube-DNS** pod.



```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: wordpress-frontend
  labels:
    tier: wordpress-frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      tier: wordpress-frontend
  template:
    metadata:
      labels:
        tier: wordpress-frontend
    spec:
      volumes:
        - name: wordpress-persistent-storage
          persistentVolumeClaim:
            claimName: wp-pv-claim
      containers:
        - name: wordpress
          image: 'wordpress:4.8-apache'
          workingDir: /var/www/html/wordpress-frontend
          ports:
            - name: wordpress
              containerPort: 80
              protocol: TCP
          env:
            - name: WORDPRESS_DB_HOST
              value: wordpress-db
```



```yaml
kind: Service
apiVersion: v1
metadata:
  name: wordpress-db
  labels:
    tier: wordpress-db
spec:
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
  selector:
    tier: wordpress-db
  type: ClusterIP
```

Figure 93.
YAML description of Wordpress frontend deployment and Wordpress DB service

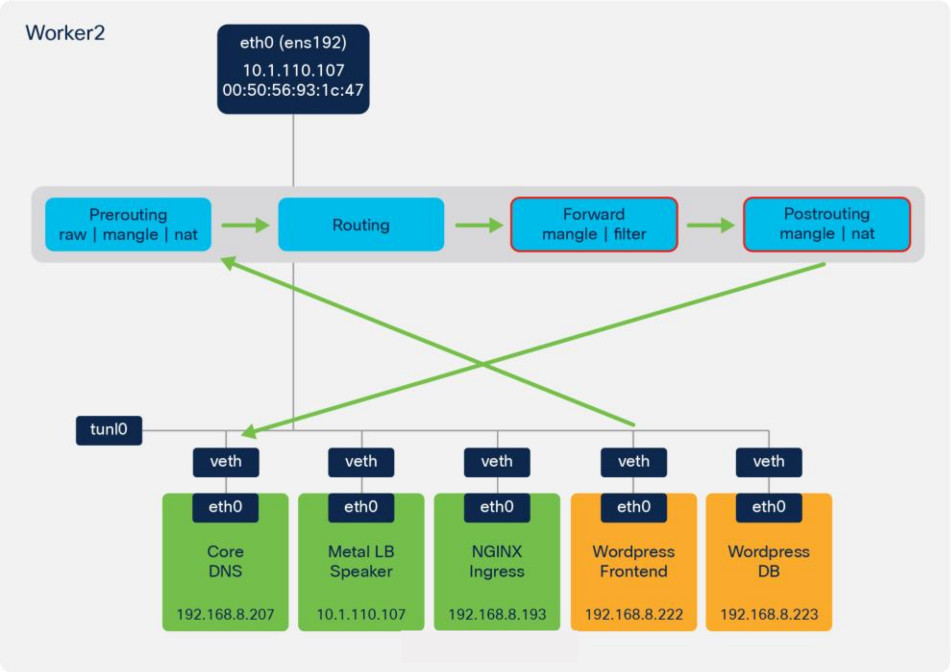
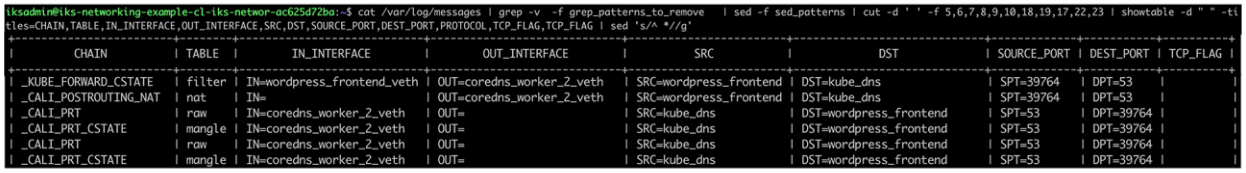
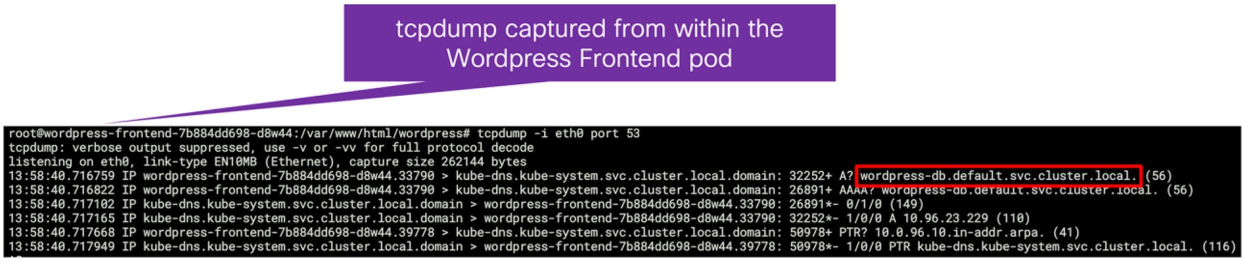


Figure 94.
Wordpress frontend to CoreDNS to resolve Wordpress DB service



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+------------------------+--------+----------------------------+-----------------------------+------------------------+----------------------------+-------------+-----------+----------+
|         CHAIN          | TABLE  |        IN_INTERFACE        |        OUT_INTERFACE        |          SRC           |            DST             | SOURCE_PORT | DEST_PORT | TCP_FLAG |
+------------------------+--------+----------------------------+-----------------------------+------------------------+----------------------------+-------------+-----------+----------+
| _KUBE_FORWARD_CSTATE   | filter | IN=wordpress_frontend_veth | OUT=coredns_worker_2_veth   | SRC=wordpress_frontend | DST=kube_dns               | SPT=39764   | DPT=53    |          |
| _CALI_POSTROUTING_NAT  | nat    | IN=                        | OUT=coredns_worker_2_veth   | SRC=wordpress_frontend | DST=kube_dns               | SPT=39764   | DPT=53    |          |
| _CALI_PRT              | raw    | IN=coredns_worker_2_veth   | OUT=                        | SRC=kube_dns           | DST=wordpress_frontend     | SPT=53      | DPT=39764 |          |
| _CALI_PRT_CSTATE       | mangle | IN=coredns_worker_2_veth   | OUT=                        | SRC=kube_dns           | DST=wordpress_frontend     | SPT=53      | DPT=39764 |          |
| _CALI_PRT              | raw    | IN=coredns_worker_2_veth   | OUT=                        | SRC=kube_dns           | DST=wordpress_frontend     | SPT=53      | DPT=39764 |          |
| _CALI_PRT_CSTATE       | mangle | IN=coredns_worker_2_veth   | OUT=                        | SRC=kube_dns           | DST=wordpress_frontend     | SPT=53      | DPT=39764 |          |
```

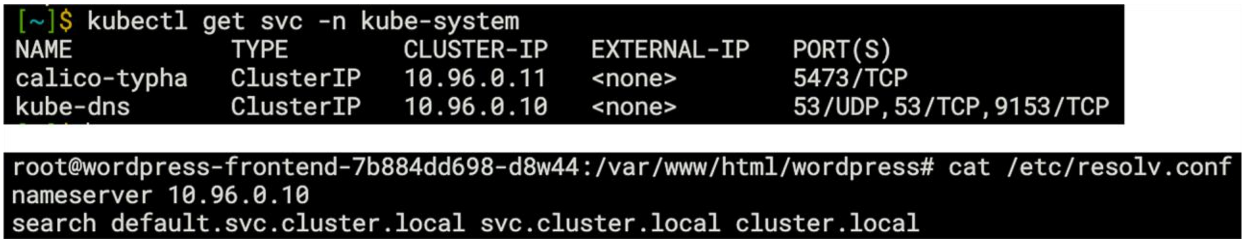
Figure 95.
Logging showing Wordpress frontend to CoreDNS to resolve Wordpress DB service – **forward** and **postrouting chains**



tcpdump captured from within the Wordpress Frontend pod

```
root@wordpress-frontend-7b884dd698-d8w44:/var/www/html/wordpress# tcpdump -i eth0 port 53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:58:40.716759 IP wordpress-frontend-7b884dd698-d8w44.33790 > kube-dns.kube-system.svc.cluster.local.domain: 32252+ A? wordpress-db.default.svc.cluster.local. (56)
13:58:40.716822 IP wordpress-frontend-7b884dd698-d8w44.33790 > kube-dns.kube-system.svc.cluster.local.domain: 26891+ AAAA? wordpress-db.default.svc.cluster.local. (56)
13:58:40.717102 IP kube-dns.kube-system.svc.cluster.local.domain > wordpress-frontend-7b884dd698-d8w44.33790: 26891*- 0/1/0 (149)
13:58:40.717165 IP kube-dns.kube-system.svc.cluster.local.domain > wordpress-frontend-7b884dd698-d8w44.33790: 32252*- 1/0/0 A 10.96.23.229 (110)
13:58:40.717668 IP wordpress-frontend-7b884dd698-d8w44.39778 > kube-dns.kube-system.svc.cluster.local.domain: 50978+ PTR? 10.0.96.10.in-addr.arpa. (41)
13:58:40.717949 IP kube-dns.kube-system.svc.cluster.local.domain > wordpress-frontend-7b884dd698-d8w44.39778: 50978*- 1/0/0 PTR kube-dns.kube-system.svc.cluster.local. (116)
```

Figure 96.
tcpdump from within the Wordpress frontend pod showing the DNS requests for the **wordpress-db** service



```
[~]$ kubectl get svc -n kube-system
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
calico-typha  ClusterIP   10.96.0.11   <none>        5473/TCP
kube-dns      ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP
```

```
root@wordpress-frontend-7b884dd698-d8w44:/var/www/html/wordpress# cat /etc/resolv.conf
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
```

Figure 97.
Output from within the Wordpress frontend pod showing the configured nameserver as the **kube-dns** service

- Traffic traverses the IPTables **NAT** table in the **Prerouting** chain.

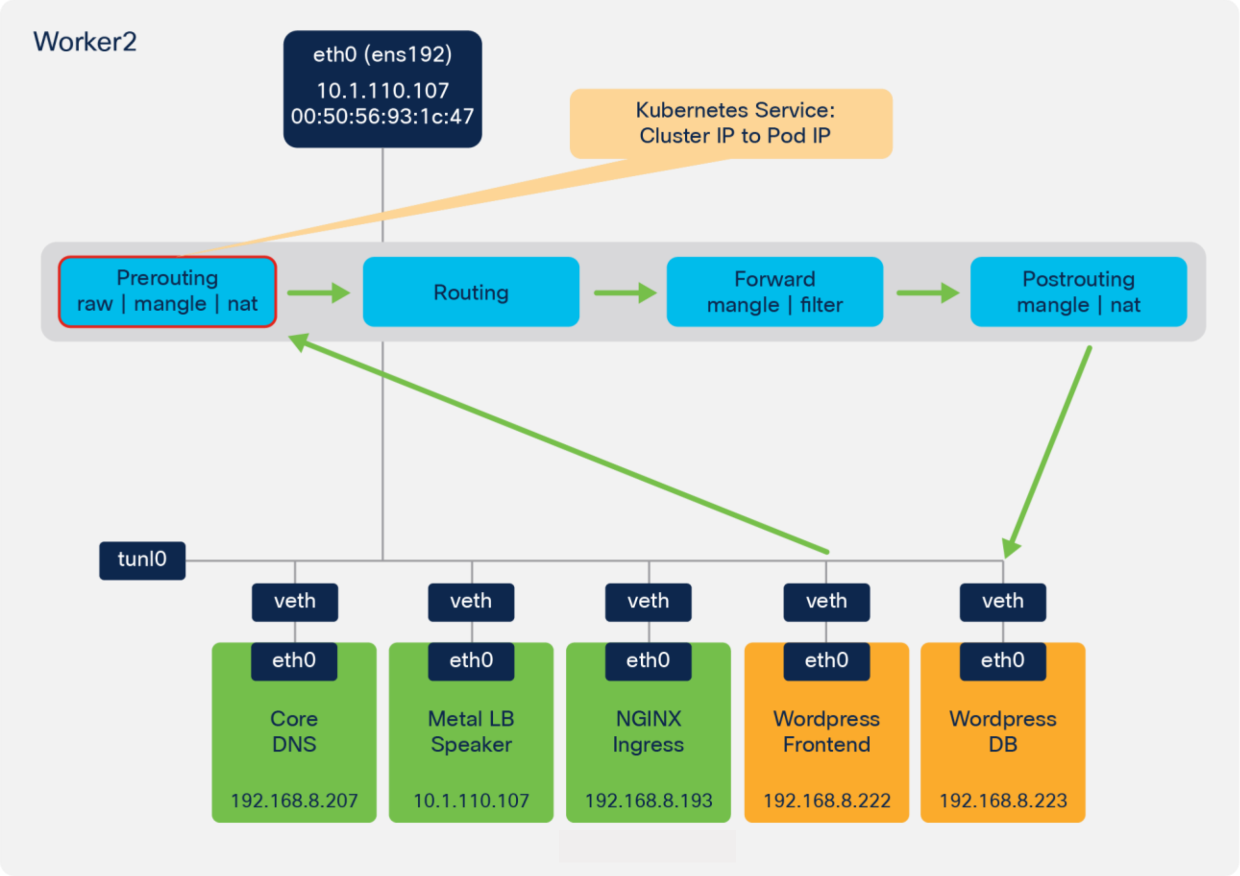
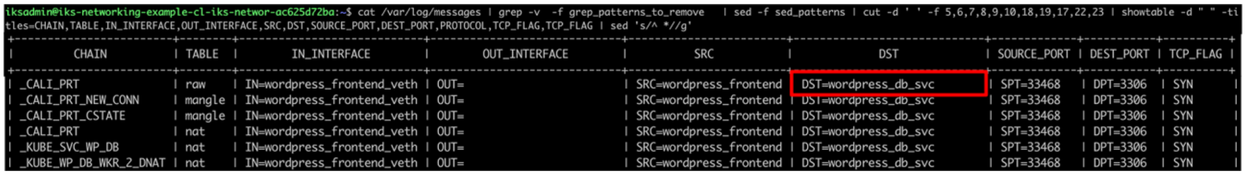
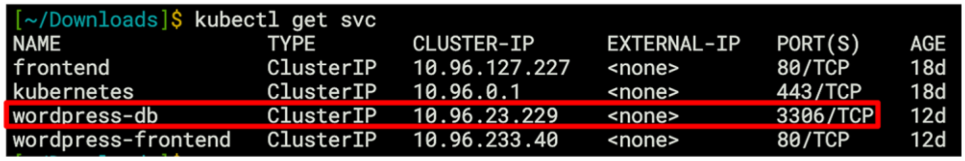


Figure 98.
Wordpress frontend to Wordpress DB – **prerouting**



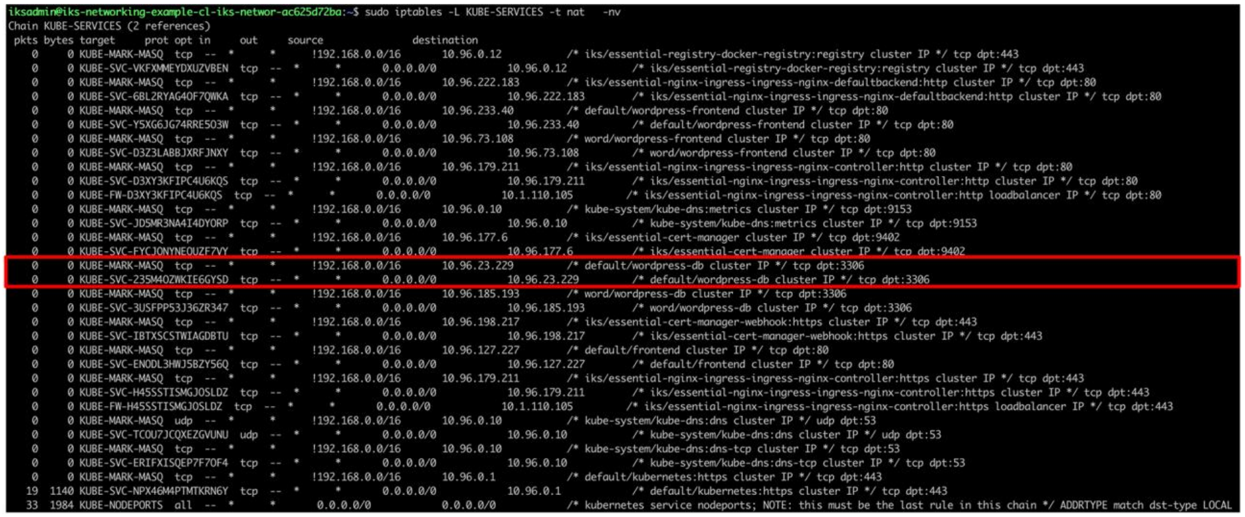
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+-----------------------+--------+---------------------------+---------------------------+-----------------------+----------------------------+-------------+-----------+----------+
|         CHAIN         | TABLE  |       IN_INTERFACE        |       OUT_INTERFACE       |          SRC          |            DST             | SOURCE_PORT | DEST_PORT | TCP_FLAG |
+-----------------------+--------+---------------------------+---------------------------+-----------------------+----------------------------+-------------+-----------+----------+
| _CALI_PRT             | raw    | IN=wordpress_frontend_veth | OUT=                     | SRC=wordpress_frontend  DST=wordpress_db_svc          | SPT=33468   | DPT=3306  | SYN      |
| _CALI_PRT_NEW_CONN    | mangle | IN=wordpress_frontend_veth | OUT=                     | SRC=wordpress_frontend | DST=wordpress_db_svc         | SPT=33468   | DPT=3306  | SYN      |
| _CALI_PRT_CSTATE      | mangle | IN=wordpress_frontend_veth | OUT=                     | SRC=wordpress_frontend | DST=wordpress_db_svc         | SPT=33468   | DPT=3306  | SYN      |
| _CALI_PRT             | nat    | IN=wordpress_frontend_veth | OUT=                     | SRC=wordpress_frontend | DST=wordpress_db_svc         | SPT=33468   | DPT=3306  | SYN      |
| _KUBE_SVC_WP_DB       | nat    | IN=wordpress_frontend_veth | OUT=                     | SRC=wordpress_frontend | DST=wordpress_db_svc         | SPT=33468   | DPT=3306  | SYN      |
| _KUBE_WP_DB_WKR_2_DNAT | nat   | IN=wordpress_frontend_veth | OUT=                     | SRC=wordpress_frontend | DST=wordpress_db_svc         | SPT=33468   | DPT=3306  | SYN      |
```

Figure 99.
Logging the Wordpress frontend to Wordpress DB – **prerouting chain**



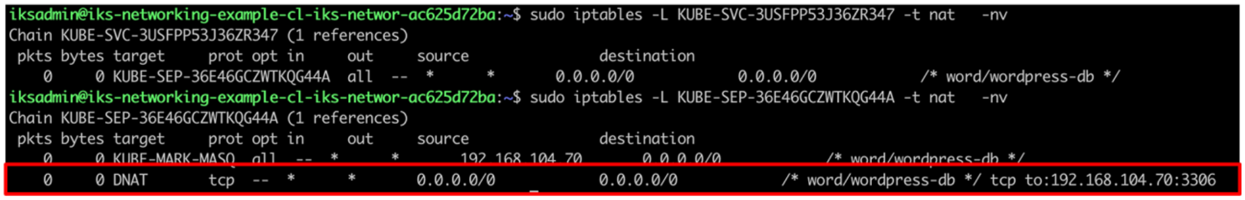
```
[~/Downloads]$ kubectl get svc
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
frontend             ClusterIP   10.96.127.227   <none>        80/TCP     18d
kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP    18d
wordpress-db         ClusterIP   10.96.23.229    <none>        3306/TCP   12d
wordpress-frontend   ClusterIP   10.96.233.40    <none>        80/TCP     12d
```

Figure 100.
Output from the **kubectl get service** command showing the **wordpress-db** port



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SERVICES -t nat   -nv
Chain KUBE-SERVICES (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.0.12           /* iks/essential-registry-docker-registry:registry cluster IP */ tcp dpt:443
    0     0 KUBE-SVC-VKFXMMEYDXUZVBEN  tcp  --  *      *       0.0.0.0/0            10.96.0.12           /* iks/essential-registry-docker-registry:registry cluster IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.222.183        /* iks/essential-nginx-ingress-ingress-nginx-defaultbackend:http cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-6BL2RYAG4OF7QWKA  tcp  --  *      *       0.0.0.0/0            10.96.222.183        /* iks/essential-nginx-ingress-ingress-nginx-defaultbackend:http cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.233.40         /* default/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-Y5XG6JG74RRE5O3W  tcp  --  *      *       0.0.0.0/0            10.96.233.40         /* default/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.73.108         /* word/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-D3Z3LABBJXRFJNXY  tcp  --  *      *       0.0.0.0/0            10.96.73.108         /* word/wordpress-frontend cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-controller:http cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-D3XY3KFIPC4U6KQS  tcp  --  *      *       0.0.0.0/0            10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-controller:http cluster IP */ tcp dpt:80
    0     0 KUBE-FW-D3XY3KFIPC4U6KQS  tcp  --  *      *       0.0.0.0/0            10.1.110.105         /* iks/essential-nginx-ingress-ingress-nginx-controller:http loadbalancer IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.0.10           /* kube-system/kube-dns:metrics cluster IP */ tcp dpt:9153
    0     0 KUBE-SVC-JD5MR3NA4I4DYORP  tcp  --  *      *       0.0.0.0/0            10.96.0.10           /* kube-system/kube-dns:metrics cluster IP */ tcp dpt:9153
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.177.6          /* iks/essential-cert-manager cluster IP */ tcp dpt:9402
    0     0 KUBE-SVC-FYCJONYNEOUZF7VY  tcp  --  *      *       0.0.0.0/0            10.96.177.6          /* iks/essential-cert-manager cluster IP */ tcp dpt:9402
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.23.229         /* default/wordpress-db cluster IP */ tcp dpt:3306
    0     0 KUBE-SVC-235M4OZWKIE6GYSD  tcp  --  *      *       0.0.0.0/0            10.96.23.229         /* default/wordpress-db cluster IP */ tcp dpt:3306
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.185.193        /* word/wordpress-db cluster IP */ tcp dpt:3306
    0     0 KUBE-SVC-3USFPP53J36ZR347  tcp  --  *      *       0.0.0.0/0            10.96.185.193        /* word/wordpress-db cluster IP */ tcp dpt:3306
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.198.217        /* iks/essential-cert-manager-webhook:https cluster IP */ tcp dpt:443
    0     0 KUBE-SVC-IBTXSCSTWIAGDBTU  tcp  --  *      *       0.0.0.0/0            10.96.198.217        /* iks/essential-cert-manager-webhook:https cluster IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.127.227        /* default/frontend cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-ENODL3HWJ5BZY56Q  tcp  --  *      *       0.0.0.0/0            10.96.127.227        /* default/frontend cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-controller:https cluster IP */ tcp dpt:443
    0     0 KUBE-SVC-H45SSTISMGJOSLDZ  tcp  --  *      *       0.0.0.0/0            10.96.179.211        /* iks/essential-nginx-ingress-ingress-nginx-controller:https cluster IP */ tcp dpt:443
    0     0 KUBE-FW-H45SSTISMGJOSLDZ  tcp  --  *      *       0.0.0.0/0            10.1.110.105         /* iks/essential-nginx-ingress-ingress-nginx-controller:https loadbalancer IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  udp  --  *      *      !192.168.0.0/16       10.96.0.10           /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
    0     0 KUBE-SVC-TCOU7JCQXEZGVUNU  udp  --  *      *       0.0.0.0/0            10.96.0.10           /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.0.10           /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
    0     0 KUBE-SVC-ERIFXISQEP7F7OF4  tcp  --  *      *       0.0.0.0/0            10.96.0.10           /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !192.168.0.0/16       10.96.0.1            /* default/kubernetes:https cluster IP */ tcp dpt:443
   19  1140 KUBE-SVC-NPX46M4PTMTKRN6Y  tcp  --  *      *       0.0.0.0/0            10.96.0.1            /* default/kubernetes:https cluster IP */ tcp dpt:443
   33  1984 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
```

Figure 101.
IPTables rules to direct traffic to the **wordpress-db** port



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SVC-3USFPP53J36ZR347 -t nat   -nv
Chain KUBE-SVC-3USFPP53J36ZR347 (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-SEP-36E46GCZWTKQG44A  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* word/wordpress-db */
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SEP-36E46GCZWTKQG44A -t nat   -nv
Chain KUBE-SEP-36E46GCZWTKQG44A (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       192.168.104.70       0.0.0.0/0            /* word/wordpress-db */
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            /* word/wordpress-db */ tcp to:192.168.104.70:3306
```

Figure 102.
IPTables rules to direct traffic to the **wordpress-db** port

- Using the IPTables **Filter** and **NAT** tables in the **Forward** and **Postrouting** chains, the traffic is sent to the veth interface on the Wordpress DB pod.

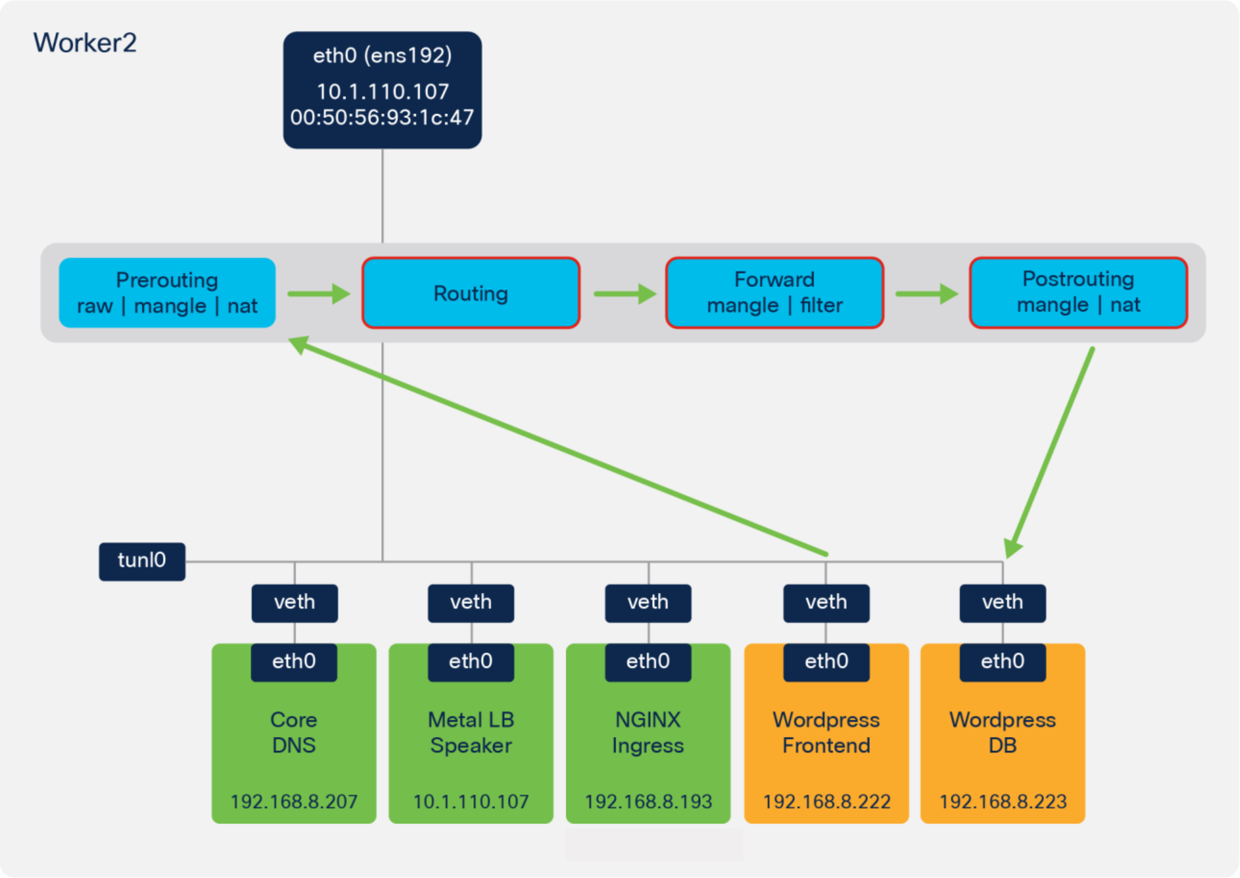
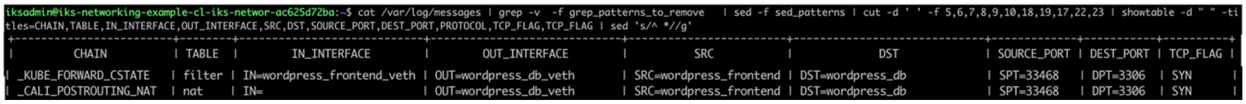
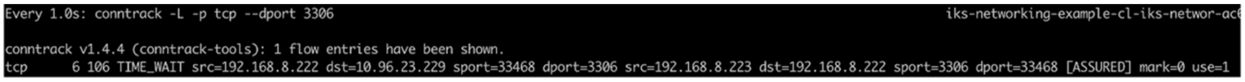


Figure 103.
Wordpress frontend to Wordpress DB – **routing**, **forward**, and **postrouting**



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+-------------------------+--------+---------------------------+---------------------------+-------------------------+-----------------------------+-------------+-----------+----------+
|          CHAIN          | TABLE  |       IN_INTERFACE        |       OUT_INTERFACE       |           SRC           |             DST             | SOURCE_PORT | DEST_PORT | TCP_FLAG |
| _KUBE_FORWARD_CSTATE    | filter | IN=wordpress_frontend_veth | OUT=wordpress_db_veth     | SRC=wordpress_frontend | DST=wordpress_db             | SPT=33468   | DPT=3306  | SYN      |
| _CALI_POSTROUTING_NAT   | nat    | IN=                       | OUT=wordpress_db_veth     | SRC=wordpress_frontend | DST=wordpress_db             | SPT=33468   | DPT=3306  | SYN      |
```

Figure 104.
Logging from Wordpress frontend to Wordpress DB – **routing**, **forward**, and **postrouting chains**



```
Every 1.0s: conntrack -L -p tcp --dport 3306                                                                           iks-networking-example-cl-iks-networ-ac

conntrack v1.4.4 (conntrack-tools): 1 flow entries have been shown.
tcp      6 106 TIME_WAIT src=192.168.8.222 dst=10.96.23.229 sport=33468 dport=3306 src=192.168.8.223 dst=192.168.8.222 sport=3306 dport=33468 [ASSURED] mark=0 use=1
```

Figure 105.
Output from connection tracking

All the traffic in the example so far has been located on the same worker node. In many cases pods may be running across multiple nodes in the Kubernetes cluster. When this occurs, the traffic will leave one node and enter a second node where the workload is running. As previously described, IKS uses Calico for container networking and implements IP-IP tunneling.

Figure 106 provides an example of the forwarding behaviour when traffic is sent between Kubernetes nodes in IKS.

- In the example, the client traffic has reached Worker 2, based on the ARP response from MetalLB.
- There are two NGINX pods and, through the IPTables rules, it is determined that the NGINX pod on Worker 1 is used for the connection.
- Based on a route lookup, the traffic is sent to the **tunl0** interface and encapsulated (IP in IP encapsulation).
- Traffic is sent out the **ens192** interface to the NGINX pod on Worker 1.

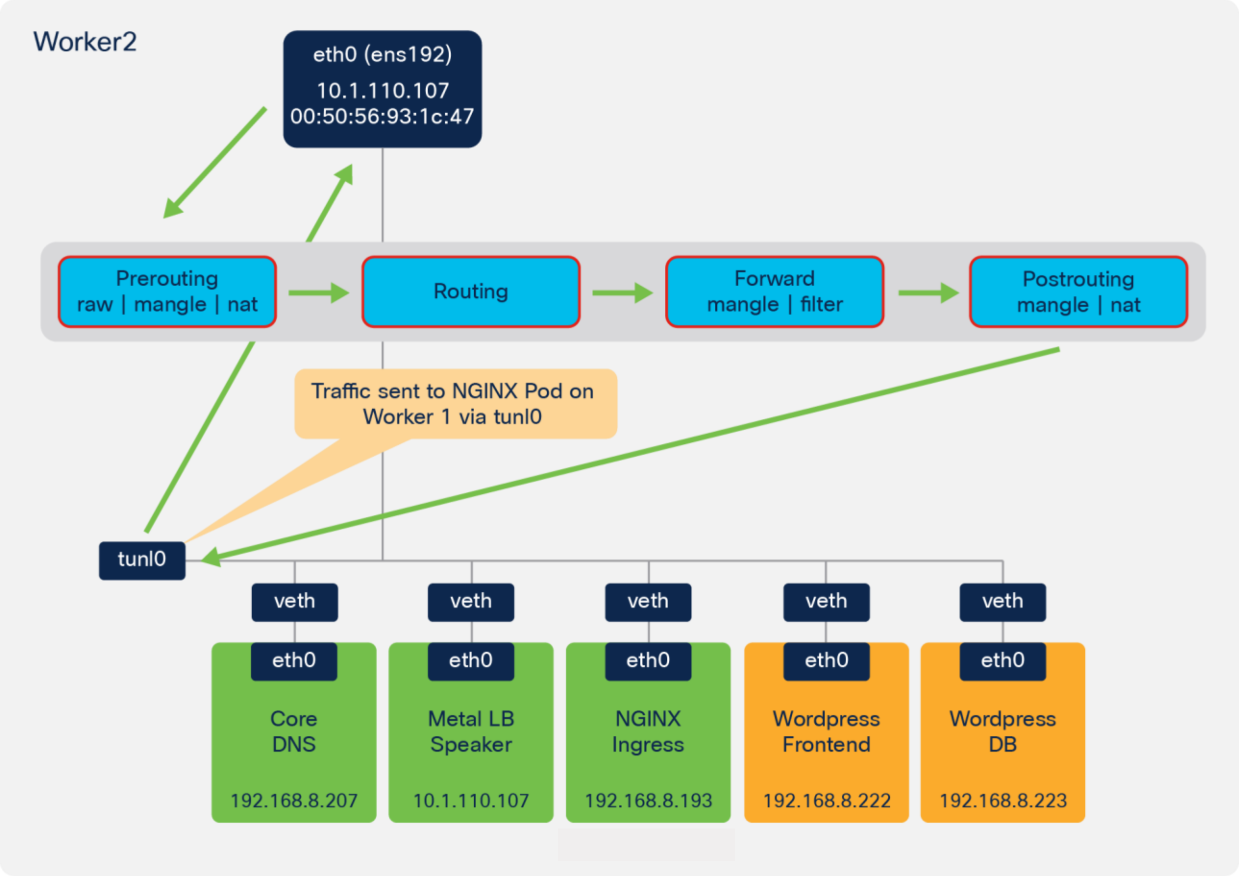
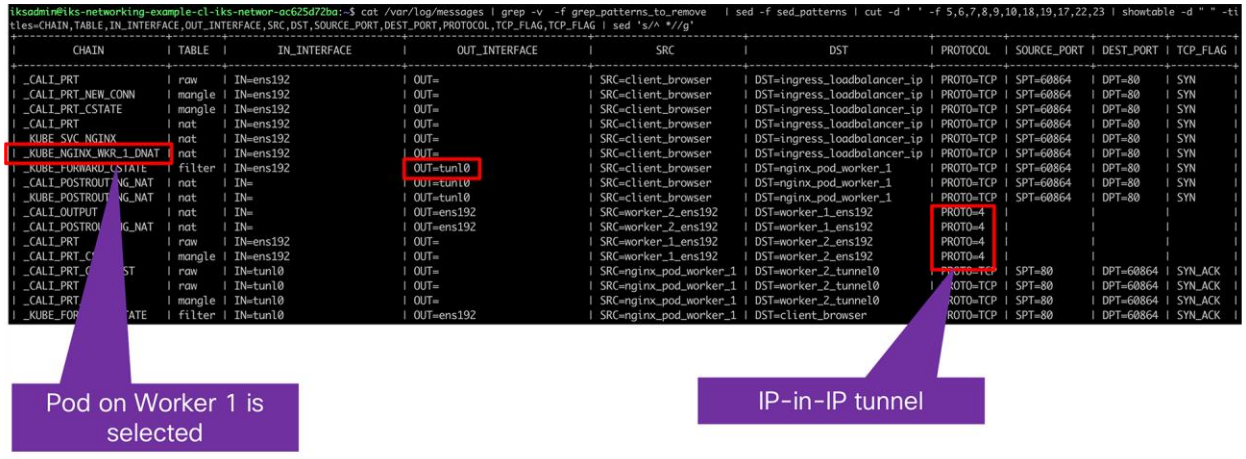
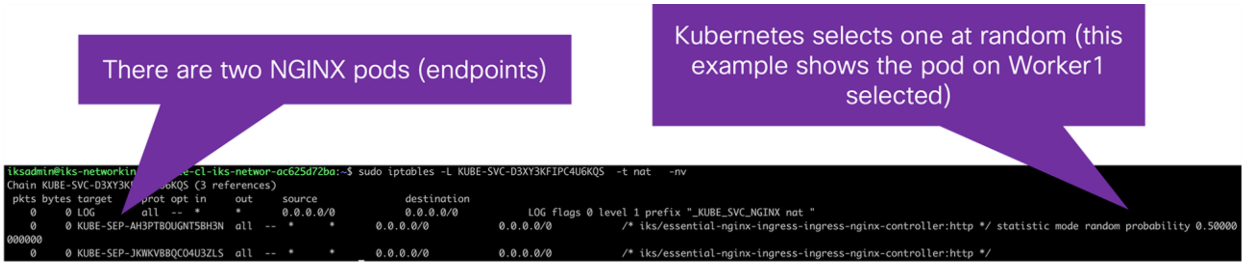


Figure 106.
Example flow showing tunnelled traffic from worker 2 to worker 1



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ cat /var/log/messages | grep -v  -f grep_patterns_to_remove   | sed -f sed_patterns | cut -d ' ' -f 5,6,7,8,9,10,18,19,17,22,23 | showtable -d " " -ti
tles=CHAIN,TABLE,IN_INTERFACE,OUT_INTERFACE,SRC,DST,SOURCE_PORT,DEST_PORT,PROTOCOL,TCP_FLAG,TCP_FLAG | sed 's/^ *//g'
+--------------------------+--------+---------------------------+---------------------------+-------------------------+-----------------------------+-----------+-------------+-----------+----------+
|          CHAIN           | TABLE  |        IN_INTERFACE       |       OUT_INTERFACE       |           SRC           |             DST             | PROTOCOL  | SOURCE_PORT | DEST_PORT | TCP_FLAG |
+--------------------------+--------+---------------------------+---------------------------+-------------------------+-----------------------------+-----------+-------------+-----------+----------+
| _CALI_PRT                | raw    | IN=ens192                 | OUT=                      | SRC=client_browser      | DST=ingress_loadbalancer_ip | PROTO=TCP | SPT=60864   | DPT=80    | SYN      |
| _CALI_PRT_NEW_CONN       | mangle | IN=ens192                 | OUT=                      | SRC=client_browser      | DST=ingress_loadbalancer_ip | PROTO=TCP | SPT=60864   | DPT=80    | SYN      |
| _CALI_PRT_CSTATE         | mangle | IN=ens192                 | OUT=                      | SRC=client_browser      | DST=ingress_loadbalancer_ip | PROTO=TCP | SPT=60864   | DPT=80    | SYN      |
| _CALI_PRT                | nat    | IN=ens192                 | OUT=                      | SRC=client_browser      | DST=ingress_loadbalancer_ip | PROTO=TCP | SPT=60864   | DPT=80    | SYN      |
| _KUBE_SVC_NGINX          | nat    | IN=ens192                 | OUT=                      | SRC=client_browser      | DST=ingress_loadbalancer_ip | PROTO=TCP | SPT=60864   | DPT=80    | SYN      |
| _KUBE_NGINX_WKR_1_DNAT   | nat    | IN=ens192                 | OUT=                      | SRC=client_browser      | DST=ingress_loadbalancer_ip | PROTO=TCP | SPT=60864   | DPT=80    | SYN      |
| _KUBE_FORWARD_CSTATE     | filter | IN=ens192                 | OUT=tunl0                 | SRC=client_browser      | DST=nginx_pod_worker_1      | PROTO=TCP | SPT=60864   | DPT=80    | SYN      |
| _CALI_POSTROUTING_NAT    | nat    | IN=                       | OUT=tunl0                 | SRC=client_browser      | DST=nginx_pod_worker_1      | PROTO=TCP | SPT=60864   | DPT=80    | SYN      |
| _KUBE_POSTROUTING_NAT    | nat    | IN=                       | OUT=tunl0                 | SRC=client_browser      | DST=nginx_pod_worker_1      | PROTO=TCP | SPT=60864   | DPT=80    | SYN      |
| _CALI_OUTPUT             | nat    | IN=                       | OUT=ens192                | SRC=worker_2_ens192     | DST=worker_1_ens192         | PROTO=4   |             |           |          |
| _CALI_POSTROUTING_NAT    | nat    | IN=                       | OUT=ens192                | SRC=worker_2_ens192     | DST=worker_1_ens192         | PROTO=4   |             |           |          |
| _CALI_PRT                | raw    | IN=ens192                 | OUT=                      | SRC=worker_1_ens192     | DST=worker_2_ens192         | PROTO=4   |             |           |          |
| _CALI_PRT_C              | mangle | IN=ens192                 | OUT=                      | SRC=worker_1_ens192     | DST=worker_2_ens192         | PROTO=4   |             |           |          |
| _CALI_PRT_C       ST     | raw    | IN=tunl0                  | OUT=                      | SRC=nginx_pod_worker_1  | DST=worker_2_tunnel0        | PROTO=TCP | SPT=80      | DPT=60864 | SYN_ACK  |
| _CALI_PRT                | raw    | IN=tunl0                  | OUT=                      | SRC=nginx_pod_worker_1  | DST=worker_2_tunnel0        | PROTO=TCP | SPT=80      | DPT=60864 | SYN_ACK  |
| _CALI_PRT_               | mangle | IN=tunl0                  | OUT=                      | SRC=nginx_pod_worker_1  | DST=worker_2_tunnel0        | PROTO=TCP | SPT=80      | DPT=60864 | SYN_ACK  |
| _KUBE_FOR        ATE     | filter | IN=tunl0                  | OUT=ens192                | SRC=nginx_pod_worker_1  | DST=client_browser          | PROTO=TCP | SPT=80      | DPT=60864 | SYN_ACK  |
```

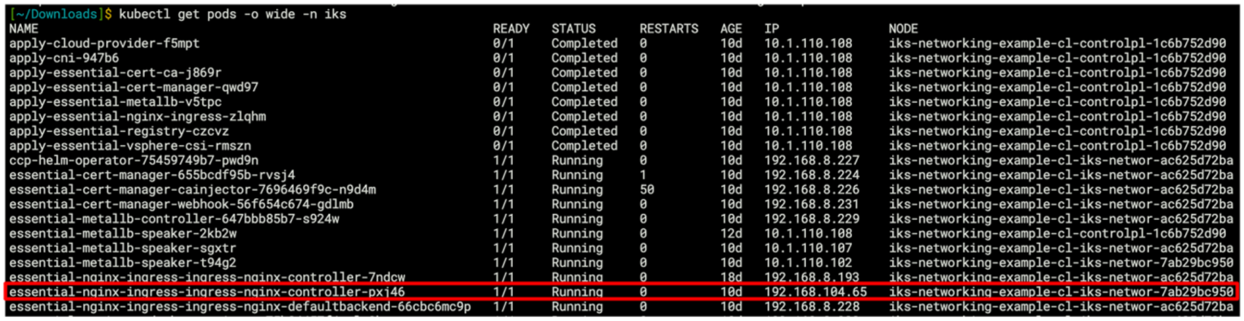
Figure 107.
Logging from IPTables showing showing traffic is directed to NGINX on worker 1



There are two NGINX pods (endpoints)
Kubernetes selects one at random (this example shows the pod on Worker1 selected)

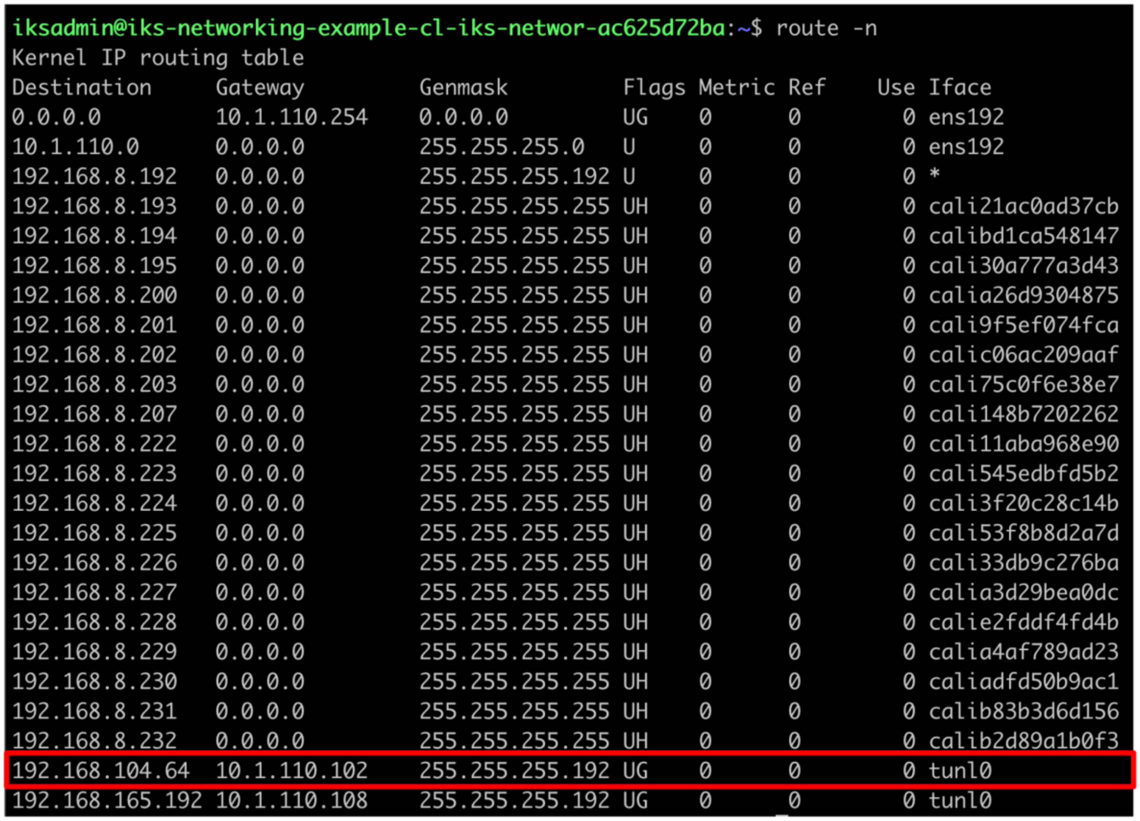
```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ sudo iptables -L KUBE-SVC-D3XY3KFIPC4U6KQS  -t nat   -nv
Chain KUBE-SVC-D3XY3KFIPC4U6KQS (3 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 LOG        all  --  *      *       0.0.0.0/0            0.0.0.0/0            LOG flags 0 level 1 prefix "_KUBE_SVC_NGINX nat "
    0     0 KUBE-SEP-AH3PTBOUGNT5BH3N  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http */ statistic mode random probability 0.50000
000000
    0     0 KUBE-SEP-JKWKVBBQCO4U3ZLS  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* iks/essential-nginx-ingress-ingress-nginx-controller:http */
```

Figure 108.
IPTables rules showing two NGINX pods



```
[~/Downloads]$ kubectl get pods -o wide -n iks
NAME                                                     READY   STATUS      RESTARTS   AGE   IP              NODE
apply-cloud-provider-f5mpt                               0/1     Completed   0          10d   10.1.110.108    iks-networking-example-cl-controlpl-1c6b752d90
apply-cni-947b6                                          0/1     Completed   0          10d   10.1.110.108    iks-networking-example-cl-controlpl-1c6b752d90
apply-essential-cert-ca-j869r                            0/1     Completed   0          10d   10.1.110.108    iks-networking-example-cl-controlpl-1c6b752d90
apply-essential-cert-manager-qwd97                       0/1     Completed   0          10d   10.1.110.108    iks-networking-example-cl-controlpl-1c6b752d90
apply-essential-metallb-v5tpc                            0/1     Completed   0          10d   10.1.110.108    iks-networking-example-cl-controlpl-1c6b752d90
apply-essential-nginx-ingress-zlqhm                      0/1     Completed   0          10d   10.1.110.108    iks-networking-example-cl-controlpl-1c6b752d90
apply-essential-registry-czcvz                           0/1     Completed   0          10d   10.1.110.108    iks-networking-example-cl-controlpl-1c6b752d90
apply-essential-vsphere-csi-rmszn                        0/1     Completed   0          10d   10.1.110.108    iks-networking-example-cl-controlpl-1c6b752d90
ccp-helm-operator-75459749b7-pwd9n                       1/1     Running     0          10d   192.168.8.227   iks-networking-example-cl-iks-networ-ac625d72ba
essential-cert-manager-655bcdf95b-rvsj4                  1/1     Running     1          10d   192.168.8.224   iks-networking-example-cl-iks-networ-ac625d72ba
essential-cert-manager-cainjector-7696469f9c-n9d4m       1/1     Running     50         10d   192.168.8.226   iks-networking-example-cl-iks-networ-ac625d72ba
essential-cert-manager-webhook-56f654c674-gdlmb          1/1     Running     0          10d   192.168.8.231   iks-networking-example-cl-iks-networ-ac625d72ba
essential-metallb-controller-647bbb85b7-s924w            1/1     Running     0          10d   192.168.8.229   iks-networking-example-cl-iks-networ-ac625d72ba
essential-metallb-speaker-2kb2w                          1/1     Running     0          12d   10.1.110.108    iks-networking-example-cl-controlpl-1c6b752d90
essential-metallb-speaker-sgxtr                          1/1     Running     0          10d   10.1.110.107    iks-networking-example-cl-iks-networ-ac625d72ba
essential-metallb-speaker-t94g2                          1/1     Running     0          10d   10.1.110.102    iks-networking-example-cl-iks-networ-7ab29bc950
essential-nginx-ingress-ingress-nginx-controller-7ndcw   1/1     Running     0          18d   192.168.8.193   iks-networking-example-cl-iks-networ-ac625d72ba
essential-nginx-ingress-ingress-nginx-controller-pxj46   1/1     Running     0          10d   192.168.104.65  iks-networking-example-cl-iks-networ-7ab29bc950
essential-nginx-ingress-ingress-nginx-defaultbackend-66cbc6mc9p   1/1     Running     0          10d   192.168.8.228   iks-networking-example-cl-iks-networ-ac625d72ba
```

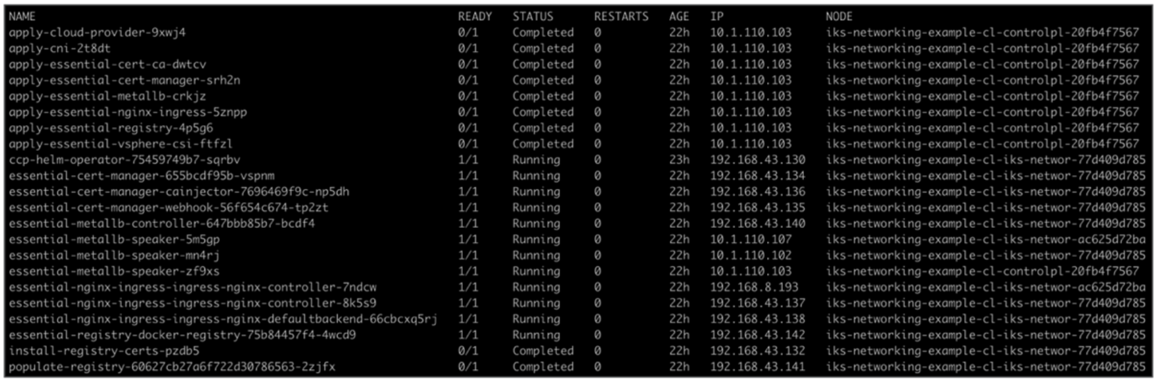
Figure 109.
Output from **kubectl get pods -o wide -n iks** command showing the selected NGINX pod is running on worker 1



```
iksadmin@iks-networking-example-cl-iks-networ-ac625d72ba:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.1.110.254    0.0.0.0         UG    0      0        0 ens192
10.1.110.0      0.0.0.0         255.255.255.0   U     0      0        0 ens192
192.168.8.192   0.0.0.0         255.255.255.192 U     0      0        0 *
192.168.8.193   0.0.0.0         255.255.255.255 UH    0      0        0 cali21ac0ad37cb
192.168.8.194   0.0.0.0         255.255.255.255 UH    0      0        0 calibd1ca548147
192.168.8.195   0.0.0.0         255.255.255.255 UH    0      0        0 cali30a777a3d43
192.168.8.200   0.0.0.0         255.255.255.255 UH    0      0        0 calia26d9304875
192.168.8.201   0.0.0.0         255.255.255.255 UH    0      0        0 cali9f5ef074fca
192.168.8.202   0.0.0.0         255.255.255.255 UH    0      0        0 calic06ac209aaf
192.168.8.203   0.0.0.0         255.255.255.255 UH    0      0        0 cali75c0f6e38e7
192.168.8.207   0.0.0.0         255.255.255.255 UH    0      0        0 cali148b7202262
192.168.8.222   0.0.0.0         255.255.255.255 UH    0      0        0 cali11aba968e90
192.168.8.223   0.0.0.0         255.255.255.255 UH    0      0        0 cali545edbfd5b2
192.168.8.224   0.0.0.0         255.255.255.255 UH    0      0        0 cali3f20c28c14b
192.168.8.225   0.0.0.0         255.255.255.255 UH    0      0        0 cali53f8b8d2a7d
192.168.8.226   0.0.0.0         255.255.255.255 UH    0      0        0 cali33db9c276ba
192.168.8.227   0.0.0.0         255.255.255.255 UH    0      0        0 calia3d29bea0dc
192.168.8.228   0.0.0.0         255.255.255.255 UH    0      0        0 calie2fddf4fd4b
192.168.8.229   0.0.0.0         255.255.255.255 UH    0      0        0 calia4af789ad23
192.168.8.230   0.0.0.0         255.255.255.255 UH    0      0        0 caliadfd50b9ac1
192.168.8.231   0.0.0.0         255.255.255.255 UH    0      0        0 calib83b3d6d156
192.168.8.232   0.0.0.0         255.255.255.255 UH    0      0        0 calib2d89a1b0f3
192.168.104.64  10.1.110.102    255.255.255.192 UG    0      0        0 tunl0
192.168.165.192 10.1.110.108    255.255.255.192 UG    0      0        0 tunl0
```

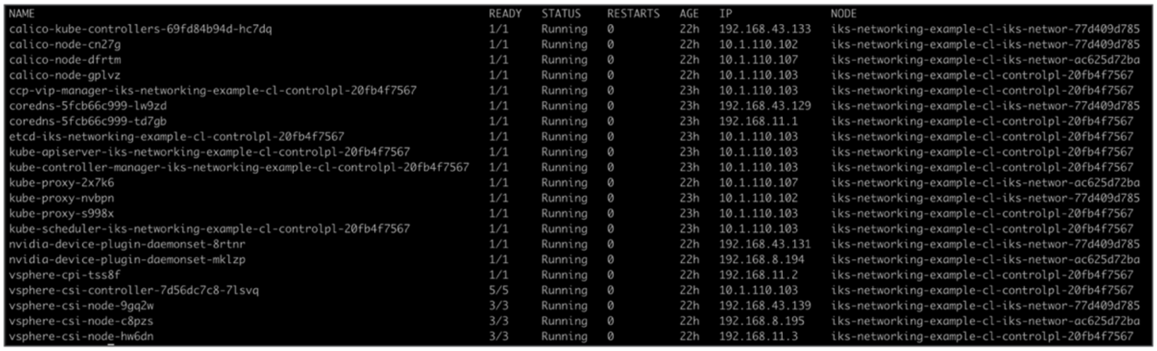
Figure 110.
Output from the routing table on worker 2 indicating the traffic for worker 1 should use the **tunl0** interface

Troubleshooting IKS networking connectivity



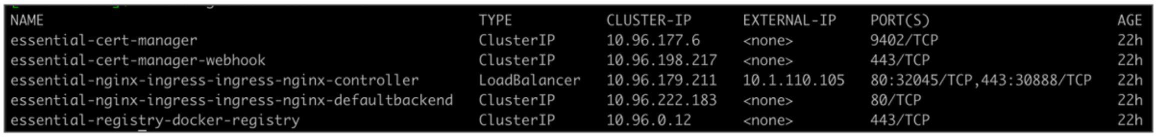
```
NAME                                                              READY   STATUS     RESTARTS   AGE   IP              NODE
apply-cloud-provider-9xwj4                                        0/1     Completed  0          22h   10.1.110.103    iks-networking-example-cl-controlpl-20fb4f7567
apply-cni-2t8dt                                                   0/1     Completed  0          22h   10.1.110.103    iks-networking-example-cl-controlpl-20fb4f7567
apply-essential-cert-ca-dwtcv                                     0/1     Completed  0          22h   10.1.110.103    iks-networking-example-cl-controlpl-20fb4f7567
apply-essential-cert-manager-srh2n                                0/1     Completed  0          22h   10.1.110.103    iks-networking-example-cl-controlpl-20fb4f7567
apply-essential-metallb-crkjz                                     0/1     Completed  0          22h   10.1.110.103    iks-networking-example-cl-controlpl-20fb4f7567
apply-essential-nginx-ingress-5znpp                               0/1     Completed  0          22h   10.1.110.103    iks-networking-example-cl-controlpl-20fb4f7567
apply-essential-registry-4p5g6                                    0/1     Completed  0          22h   10.1.110.103    iks-networking-example-cl-controlpl-20fb4f7567
apply-essential-vsphere-csi-ftfzl                                 0/1     Completed  0          22h   10.1.110.103    iks-networking-example-cl-controlpl-20fb4f7567
ccp-helm-operator-75459749b7-sqrbv                                1/1     Running    0          23h   192.168.43.130  iks-networking-example-cl-iks-networ-77d409d785
essential-cert-manager-655bcdf95b-vspnm                           1/1     Running    0          22h   192.168.43.134  iks-networking-example-cl-iks-networ-77d409d785
essential-cert-manager-cainjector-7696469f9c-np5dh                1/1     Running    0          22h   192.168.43.136  iks-networking-example-cl-iks-networ-77d409d785
essential-cert-manager-webhook-56f654c674-tp2zt                   1/1     Running    0          22h   192.168.43.135  iks-networking-example-cl-iks-networ-77d409d785
essential-metallb-controller-647bbb85b7-bcdf4                     1/1     Running    0          22h   192.168.43.140  iks-networking-example-cl-iks-networ-77d409d785
essential-metallb-speaker-5m5gp                                   1/1     Running    0          22h   10.1.110.107    iks-networking-example-cl-iks-networ-ac625d72ba
essential-metallb-speaker-mn4rj                                   1/1     Running    0          22h   10.1.110.102    iks-networking-example-cl-iks-networ-77d409d785
essential-metallb-speaker-zf9xs                                   1/1     Running    0          22h   10.1.110.103    iks-networking-example-cl-controlpl-20fb4f7567
essential-nginx-ingress-ingress-nginx-controller-7ndcw            1/1     Running    0          22h   192.168.8.193   iks-networking-example-cl-iks-networ-ac625d72ba
essential-nginx-ingress-ingress-nginx-controller-8k5s9            1/1     Running    0          22h   192.168.43.137  iks-networking-example-cl-iks-networ-77d409d785
essential-nginx-ingress-ingress-nginx-defaultbackend-66cbcxq5rj   1/1     Running    0          22h   192.168.43.138  iks-networking-example-cl-iks-networ-77d409d785
essential-registry-docker-registry-75b84457f4-4wcd9               1/1     Running    0          22h   192.168.43.142  iks-networking-example-cl-iks-networ-77d409d785
install-registry-certs-pzdb5                                      0/1     Completed  0          22h   192.168.43.132  iks-networking-example-cl-iks-networ-77d409d785
populate-registry-60627cb27a6f722d30786563-2zjfx                  0/1     Completed  0          22h   192.168.43.141  iks-networking-example-cl-iks-networ-77d409d785
```

Figure 111.
Output from the **kubectl get pods -n iks** command



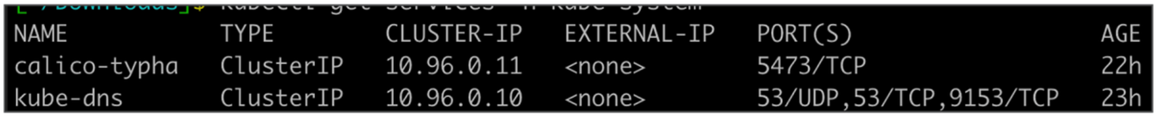
```
NAME                                                                    READY   STATUS    RESTARTS   AGE   IP               NODE
calico-kube-controllers-69fd84b94d-hc7dq                                1/1     Running   0          22h   192.168.43.133   iks-networking-example-cl-iks-networ-77d409d785
calico-node-cn27g                                                       1/1     Running   0          22h   10.1.110.102     iks-networking-example-cl-iks-networ-77d409d785
calico-node-dfrtm                                                       1/1     Running   0          22h   10.1.110.107     iks-networking-example-cl-iks-networ-ac625d72ba
calico-node-gplvz                                                       1/1     Running   0          22h   10.1.110.103     iks-networking-example-cl-controlpl-20fb4f7567
ccp-vip-manager-iks-networking-example-cl-controlpl-20fb4f7567          1/1     Running   0          23h   10.1.110.103     iks-networking-example-cl-controlpl-20fb4f7567
coredns-5fcb66c999-lw9zd                                                1/1     Running   0          23h   192.168.43.129   iks-networking-example-cl-iks-networ-77d409d785
coredns-5fcb66c999-td7gb                                                1/1     Running   0          23h   192.168.11.1     iks-networking-example-cl-controlpl-20fb4f7567
etcd-iks-networking-example-cl-controlpl-20fb4f7567                     1/1     Running   0          23h   10.1.110.103     iks-networking-example-cl-controlpl-20fb4f7567
kube-apiserver-iks-networking-example-cl-controlpl-20fb4f7567           1/1     Running   0          23h   10.1.110.103     iks-networking-example-cl-controlpl-20fb4f7567
kube-controller-manager-iks-networking-example-cl-controlpl-20fb4f7567  1/1     Running   0          23h   10.1.110.103     iks-networking-example-cl-controlpl-20fb4f7567
kube-proxy-2x7k6                                                        1/1     Running   0          22h   10.1.110.107     iks-networking-example-cl-iks-networ-ac625d72ba
kube-proxy-nvbpn                                                        1/1     Running   0          23h   10.1.110.102     iks-networking-example-cl-iks-networ-77d409d785
kube-proxy-s998x                                                        1/1     Running   0          23h   10.1.110.103     iks-networking-example-cl-controlpl-20fb4f7567
kube-scheduler-iks-networking-example-cl-controlpl-20fb4f7567           1/1     Running   0          23h   10.1.110.103     iks-networking-example-cl-controlpl-20fb4f7567
nvidia-device-plugin-daemonset-8rtnr                                    1/1     Running   0          22h   192.168.43.131   iks-networking-example-cl-iks-networ-77d409d785
nvidia-device-plugin-daemonset-mklzp                                    1/1     Running   0          22h   192.168.8.194    iks-networking-example-cl-iks-networ-ac625d72ba
vsphere-cpi-tss8f                                                       1/1     Running   0          22h   192.168.11.2     iks-networking-example-cl-controlpl-20fb4f7567
vsphere-csi-controller-7d56dc7c8-7lsvq                                  5/5     Running   0          22h   10.1.110.103     iks-networking-example-cl-controlpl-20fb4f7567
vsphere-csi-node-9gq2w                                                  3/3     Running   0          22h   192.168.43.139   iks-networking-example-cl-iks-networ-77d409d785
vsphere-csi-node-c8pzs                                                  3/3     Running   0          22h   192.168.8.195    iks-networking-example-cl-iks-networ-ac625d72ba
vsphere-csi-node-hw6dn                                                  3/3     Running   0          22h   192.168.11.3     iks-networking-example-cl-controlpl-20fb4f7567
```

Figure 112.
Output from the **kubectl get pods -n kube-system** command



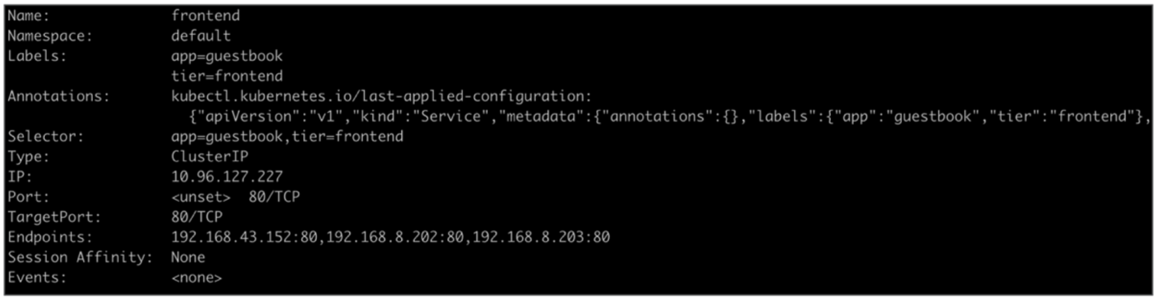
```
NAME                                                 TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)                      AGE
essential-cert-manager                               ClusterIP      10.96.177.6     <none>         9402/TCP                     22h
essential-cert-manager-webhook                       ClusterIP      10.96.198.217   <none>         443/TCP                      22h
essential-nginx-ingress-ingress-nginx-controller     LoadBalancer   10.96.179.211   10.1.110.105   80:32045/TCP,443:30888/TCP   22h
essential-nginx-ingress-ingress-nginx-defaultbackend ClusterIP      10.96.222.183   <none>         80/TCP                       22h
essential-registry-docker-registry                   ClusterIP      10.96.0.12      <none>         443/TCP                      22h
```

Figure 113.
Output from the **kubectl get services -n iks** command



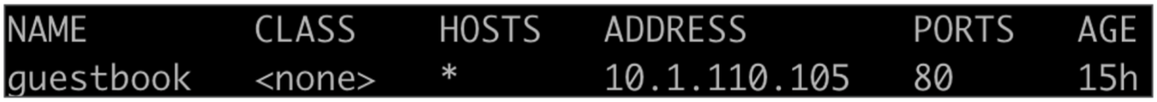
```
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
calico-typha  ClusterIP   10.96.0.11   <none>        5473/TCP                 22h
kube-dns      ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   23h
```

Figure 114.
Output from the **kubectl get services -n kube-system** command



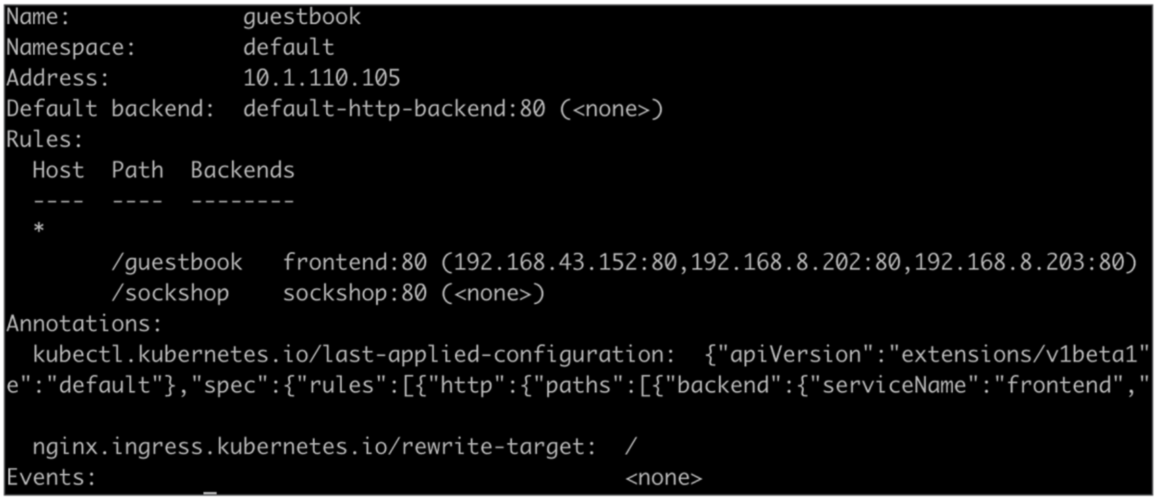
```
Name:              frontend
Namespace:         default
Labels:            app=guestbook
                   tier=frontend
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"app":"guestbook","tier":"frontend"},
Selector:          app=guestbook,tier=frontend
Type:              ClusterIP
IP:                10.96.127.227
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.43.152:80,192.168.8.202:80,192.168.8.203:80
Session Affinity:  None
Events:            <none>
```

Figure 115.
Output from the **kubectl get describe service frontend** command



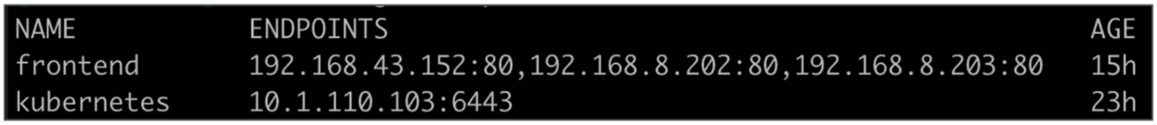
```
NAME        CLASS    HOSTS   ADDRESS        PORTS   AGE
guestbook   <none>   *       10.1.110.105   80      15h
```

Figure 116.
Output from the **kubectl get ingress** command



```
Name:             guestbook
Namespace:        default
Address:          10.1.110.105
Default backend:  default-http-backend:80 (<none>)
Rules:
  Host  Path  Backends
  ----  ----  --------
  *
        /guestbook   frontend:80 (192.168.43.152:80,192.168.8.202:80,192.168.8.203:80)
        /sockshop    sockshop:80 (<none>)
Annotations:
  kubectl.kubernetes.io/last-applied-configuration:  {"apiVersion":"extensions/v1beta1"
e":"default"},"spec":{"rules":[{"http":{"paths":[{"backend":{"serviceName":"frontend","

  nginx.ingress.kubernetes.io/rewrite-target:  /
Events:                                        <none>
```

Figure 117.
Output from the **kubectl describe ingress guestbook** command



```
NAME          ENDPOINTS                                            AGE
frontend      192.168.43.152:80,192.168.8.202:80,192.168.8.203:80  15h
kubernetes    10.1.110.103:6443                                    23h
```

Figure 118.
Output from the **kubectl get endpoints** command

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **https://www.cisco.com/go/offices**.

Printed in USA C11-2655904-00 11/21