Big Book of MLOps

Jamie Ralph

2024-09-24

Table of contents

Preface

MLOps is a big field. I work in MLOps and spend a lot of time learning things. Unfortunately my brain can only store so much information, and it's easy to forget stuff.

This book is a repository of the knowledge I've learned since starting in MLOps. The content can roughly be divided into these categories:

- Short summaries of topics with links to articles that explain things nicely
- Notes I've taken from video courses (I find these the most time consuming and attention hungry. Taking notes mean I only sit through them once)
- Useful hints from technical books I've read

Networking in Kubernetes

Notes from Certified Kubernetes Administrator (CKA) with Practice Tests

Pre-requisites on networking

How can computers talk to each other on a network?

- In a simple network, two computers (let's call them A and B) can exchange information over a network
- This communication is transmitted via a **network switch**, a piece of equipment that can connect IT devies. Network siwtches can vary in speed (e.g gigabytes per second).
- A computer sends information to the switch via an **interface**. This interface can be a piece of hardware or software depending on the situation, but is essentially a point of connection between the device and the network
- On linux, running `ip link` will print a list of interfaces in the terminal
- If we assume the network has an ip address of 192.165.1.0, we could add ip addresses to A and B with `ip addr add`, e.g. `ip addr add 192.165.1.10/24 dev eth0` on A and `ip addr add 192.165.1.11/24 dev eth0` on B. Note: Here, **dev** stands for **device** and eth0 is the first ethernet interface on the system.
- This would mean that A and B can now exchange packets with each other. Packets are small segments of a larger piece of information being sent over the network, which are recombined by the device that receives them.

Routing

- A router helps connect different networks
- The router is visible to each network with a different ip address
- Networks are configured with **gateways** which connects two different networks
- The `route` command in Linux will print the routing table
- Routes can be added using `ip route add`
- Let's assume this network setup:
  - A and B exist on network 192.165.1.0 - we'll call this network 1
  - C and D exist on network 192.165.2.0 - we'll call this network 2

  - A router is connected to network 1 via the ip 192.165.1.1, and connected to network 2 via the ip 192.165.2.1

- We can connect device A to network 2 by running `ip route add 192.165.2.0/24 via 192.165.1.1`
- Running `route` shows that the router is now a gateway to network 2
- You can set default routes instead of adding an entry for every single network - `ip route add default via 192.165.1.1` - default is sometimes seen as '0.0.0.0'
- Linux servers can act as hosts themselves but packet forwarding between interfaces needs to be enabled. This can be a security threat if one interface connects to a public network and the other to a private network.

Domain Name Systems (DNS)

- In a small simple network (let's use A and B again, connected via a switch), we can give names to each device. I can add an entry in the `/etc/hosts` file of A:

```
192.165.1.11    myname
```

- I can now run `ping myname` to check connectivity to computer B. However, A will not actually check that B's host name is myname. And this tasks would quickly become impossible as the network grew
- An internal DNS server solves the problem - it is a server containing a single source of truth
- If we assume the DNS server's ip address is 192.165.1.100, an entry in A's `/etc/resolv.conf` file tells it where to resolve domain names:

```
nameserver  192.165.1.100
```

- A DNS functions like an internet phonebook. Domain names are linked to IP addresses - it means humans don't need to memorise long IP addresses
- Domain names are strings pointing to a specific web services
- A domain name is usually comprised of several elements - e.g. www.google.com can be broken down to:
  - .com - top level domain (other examples are .edu, .org, .io) - can be a sign of the intent of the server e.g. .edu is for educational institutions, .org for non-profit
  - google - the second level domain
  - www - a subdomain (other google examples could be mail or maps)
- There are DNS servers on the internet that are searched to find the ip address of the server that is hosting the web applications
- ip addresses can be cached by browsers to speed up subsequent requests

- DNS can contain records:
  - A records - map ip addresses to hostnames
  - AAAA (quad A) records - map IPv6 to hostnames
  - CNAME (canonical name) - map one name to another e.g. if flowers.example.com had a CNAME record with a value of example.com, a lookup of "flowers.example.com" provies the ip address of "example.com", which is the canonical name

Network Namespaces in Linux

A Linux installation generally shares a single set of network interfaces and routing table entries, but you can create isolated networks within that server. This article by Scott Lowe goes into some detail about how to set up a network namespace on Ubuntu.

The CKA course on Udemy goes into more detail about creating a virtual network with namespaces and virtual switches:

- There are different options for the network switch including Linux Bridge and Open vSwitch - below we'll use Linux Bridge
- Run the command `ip link add v-net-0 type bridge` on the host - to create a new interface (run `ip link` to see it) - you'll also need to turn it up using `ip link set dev v-net-0 up`
- Next, to connect a namespace to the switch, you'll need a virtual cable. Let's assume we are connecting namespace *red* to the virtual switch:
  - Use `ip link add veth-red type veth peer name veth-red-br` to create the cable
  - Use `ip link set veth-red` netns red
  - Use `ip link set veth-red-br` master v-net-0
  - Finally, assign ip addresses to the namespace interface E.g. `ip -n red addr add 192.165.15.1/24 dev veth-red`
- To allow the virtual network to reach networks outside itself:
  - we add an ip address to the virtual interface e.g. `ip addr add 192.165.15.5/24 dev v-net-0`
  - in a given namespace (e.g. red) - add a gateway. `ip netns exec red ip route add 192.165.1.0/24 via 192.165.15.5` - our host is now the gateway (also do this for default)
  - add a rule in iptables on host to mask requests coming from the virtual network with its own ip address - `iptable -t nat -A POSTROUTING -s 192.165.15.0/24 -j MASQUERADE`

- Allowing requests to be made to a service in the virtual network from outside the host would require port forwarding with iptables
- NOTE: There were some omissions from the video - fortunately the Q&A for the video had the solution:

```
# Add a bridge interface
# It's like an interface for the host and a switch for namespaces!
ip link add v-net-0 type bridge
# Bring it up
ip link set dev v-net-0 up
# Add namespaces
ip netns add red
ip netns add blue
ip netns add green
ip netns add yellow
# Create a virtual cable for namespace and bridge
ip link add veth-red type veth peer name veth-red-br
ip link add veth-blue type veth peer name veth-blue-br
ip link add veth-green type veth peer name veth-green-br
ip link add veth-yellow type veth peer name veth-yellow-br
# Connect one end of virtual cable to namespace red
ip link set veth-red netns red
ip link set veth-blue netns blue
ip link set veth-green netns green
ip link set veth-yellow netns yellow
# Connect virtual cable to the bridge
ip link set veth-red-br master v-net-0
ip link set veth-blue-br master v-net-0
ip link set veth-green-br master v-net-0
ip link set veth-yellow-br master v-net-0
# Assign IPs to these interfaces
ip -n red addr add 192.168.15.2/24 dev veth-red
ip -n blue addr add 192.168.15.3/24 dev veth-blue
ip -n green addr add 192.168.15.4/24 dev veth-green
ip -n yellow addr add 192.168.15.5/24 dev veth-yellow
# Bring interfaces up
ip -n red link set veth-red up
ip -n blue link set veth-blue up
ip -n green link set veth-green up
ip -n yellow link set veth-yellow up
# Bring interfaces (bridge) up
ip link set veth-red-br up
```

```
ip link set veth-blue-br up
ip link set veth-green-br up
ip link set veth-yellow-br up
# Bring loopback interfaces up (for 192.168.15.0)
ip -n red link set lo up
ip -n blue link set lo up
ip -n green link set lo up
ip -n yellow link set lo up
```

Container Network Interfaces

- CNI plugins are required for implementing the Kubernetes network model
- CNI is a Cloud Native Computing Foundation (CNCF) project which defines specifications for network plugins, specifically dealing with connectivity of containers and removing allocated resources when a container is deleted
- An example plugin is the bridge plugin which connects a container to a virtual switch (the bridge) in the host network. A veth pair is generated which connects the container to the bridge. The container end is assigned an IP address. An IP address can also be assigned to the bridge itself, making it a gateway for the container.
- To check network interface configured for cluster connectivity on a node, `ip address` and check "inet" value. MAC address can be found in the same place
- To find the bridge created by the CRI on a node, run `ip address` or `ip address show type bridge`
- The state of an interface can also be found using `ip address`
- To find the default gateway on host, use `ip route`
- To find which port kube-scheduler is listening on, use `netstat -npl | grep -i scheduler`:
  - n = don't resolve name, p = display pid info, l = display listening server sockets
- To get etcd ports and count connections, `netstat -npa | grep etcd | grep <port-number> | wc -l`
  - a = display all sockets

Pod Networking

- K8s does not come with in-built pod networking
- The K8s network model requires that:
  - pods have their own private network namespace (containers can reference each other via localhost)
  - pods have IP addresses

  - pods can communicate with other pods on the same node
  - pods can communicate with other pods on different nodes without proxies or address translation (NAT)

- The ip address range applied to pods can be found by looking at the pod logs of the CNI plugin and looking for "ipalloc-range"

Service Networking

- A service is an abstraction in K8s which enables the exposure of multiple pods via a single address. These pods are usually determined with label selectors.
- Services are assigned an IP address from a pre-determined range, which should differ from the range defined for pods
- Routing rules are configured using the kube-proxy component. The process for new services or endpoints (using internal only ClusterIP) looks like this:
  - kube-apiserver communicates the change to the kube-proxy
  - kube-proxy creates rules via NAT inside a node - these are mappings from service IP and port to pod IP and port
- Newly created DNAT rules can be viewed in */var/log/kube-proxy.log* files

Notes from online articles

Proxies and Reverse Proxies

Proxies (also called forward proxies) are intermediary servers that intercept client requests. A proxy might be implemented in an organisation to do things like block restricted content. Meanwhile, reverse proxies sit in front of servers and accept requests coming over the internet. They can provide functions such as load balancing and SSL encryption/decryption. The article What is a reverse proxy? by CloudFlare provides a good overview of these concepts. NGINX is a popular reverse proxy - this introduction to NGINX provides a nice practical demonstration of its abilities.

Resources

Some of the other resources I used to understand networking basics:

- What is a network switch? by Juniper Networks
- What is computer networking? by AWS
- What is a network gateway? by NordLayer

- What is a DNS CNAME record? by CloudFlare