Big Book of MLOps

Jamie Ralph

2024-09-24

Table of contents

Preface

My journey into MLOps started in late 2023 when I needed to deploy an R Shiny app at the company I was working for. Up to that point there had been no easy way to do this, but luckily for me the MLOps team had put together a Helm chart for deploying containerised applications. I wasn't part of the data science team, but MLOps very kindly helped me get my app deployed. At the time I had no idea what Kubernetes was or how Helm charts worked - I just containerised my app with Docker, filled in some YAML files, and there it was. I was given access to this mystical platform called 'ArgoCD' where I could look at little tiles that represented things I didn't understand. The only thing I knew for certain was that the little green heart was good, and the broken red heart was bad.

Around 6 months later I was given the chance to transition over to the MLOps team. I jumped at the chance and never regretted my decision, but I quickly found the learning curve to be steep (somewhere around 89 degrees?) and was overwhelmed with where to begin. A few months into the job I started a google doc - I called it "the big book of MLOPs" (O'Reilly got there first). This was my place to write down notes from the tutorials/courses/books etc. that I was using to gain the knowledge I needed for my job. I decided these notes should sit in a more public forum because a) they could help someone else and b) if my notes could be improved, someone can open a PR and improve them, and c) the human brain can only hold so much information - writing things down is much more effective than trying to remember everything.

I hope you might find these notes useful, especially if you're just starting out in MLOps or looking to move into the field.

1 Networking in Kubernetes

1.1 Networking basics

1.1.1 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 (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` for A and `ip addr add 192.165.1.11/24 dev eth0`. 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.

1.1.2 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.

1.1.3 Domain Name Systems (DNS) in a simple internal network

- 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
```

DNS on the internet

- 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

1.1.4 Proxies

- A proxy server is a digital intermediary, routing internet traffic between users and online resources, ensuring secure and controlled data exchange. This is sometimes called a "forward proxy"
- Instead of the client request going straight to the destination via the ISP, it is routed through the proxy. The proxy can mask the original IP address and intercept the information that is returned. It can additionally scan for malicious information.
- Reverse proxies are different - they sit in front of web servers to prevent direct communication to the destination server. The difference between a forward and reverse proxy is subtle but important. A simplified way to sum it up would be to say that a forward proxy sits in front of a client and ensures that no origin server ever communicates directly with that specific client. On the other hand, a reverse proxy sits in front of an origin server and ensures that no client ever communicates directly with that origin server.
- Benefits of reverse proxies include:
  - Load balancing
  - Protection from attacks - because the origin IP address is never revealed, attacks like DOS are harder to carry out
  - Caching - local proxies accessing websites elsewhere can cache results for future users (e.g. user in the UK accessing a site hosted in the USA - cached data can now be served to other UK users for faster access)
  - SSL encryption and decryption can be performed by the reverse proxy, reducing the computational load on the origin server

1.2 Resources

Some of the resources I used in this chapter include:

- Certified Kubernetes Administrator (CKA) with Practice Tests
- 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