Deploying services in the cloud

(Day 1 - Server Setup)

Agenda

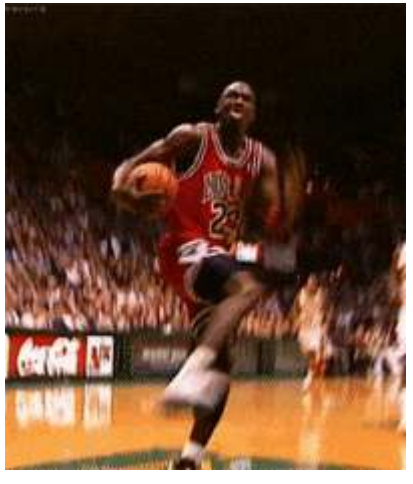
1) About this course
2) Recap of linux navigation and Vim
3) Setting up a fresh server
4) Securing a fresh server
5) Administrating user access

About this course 📜

Who is the MVP?

MVP is a well-known acronym, but it has two meanings:

1. **Most valuable player**: a term predominantly used in American sports to define an excellent contributor to a team
2. **Minimum viable product**: a business/startup term for service that has just enough features to provide a full service or baseline from which to develop.



Who is the MVP?

As a data scientist, you may think that your job involves thinking about numbers only. As far as you are concerned, your job spec does not require any thinking about servers, hosting, networking, etc. Once the dashboard is built, it is someone else's problem to think about how to get it up and running, right?

Wrong.

Learning more about the technical infrastructure that supports your daily activities and hosts your dashboards makes you better able to a) appreciate your colleagues who do all that work for you and b) allow you to develop a proof of concept for a larger service without relying on other team members.

IT professionals are full time employees for a reason. System administration, networking, cybersecurity and database administration (to name just a few fields) are deep and complex topics that you can build a full career in.

However, there is a relatively small amount of knowledge that can take you a long way. What we aim to do is harness this knowledge to create a **minimum viable product** for you as a data scientist to be able to run analytical environments, databases and dashboards that are always on and accessible to other potential users. You are then able to stand on your own two feet, without any being overly reliant on other (overworked) team members.

Then **YOU** are the MVP.

Course objectives

On completion of the course, you should feel comfortable to do the following:

- Start from scratch with a fresh Linux server
- Secure your server from common cybersecurity threats
- Create and manage users and files on your server (basic system administration)
- Deploy a database with structured access for multiple users
- Populate a database
- Deploy RStudio as an analytical environment with secured remote access

A brief overview of various roles this covers

- **System administrator**: manage health of the server, manage access and networks (bash)
- **Database adminstrator**: maintain integrity and database function (SQL)
- **Development Operations** (DevOps): deploying services for effective access (bash, Docker)

Session Breakdown: Setup

Thursday 9 May (09:00 - 10:30):

- Course introduction.
- Introduction to cloud and VPS.
- Install software for course.
- Linux basics review.
- Adding users to the server and to groups.

Thursday 9 May (11:30 - 13:00):

- Setting file and directory permissions.
- Setting up SSH keys.

Thursday 9 May (14:00 - 16:00)

- Setting up firewalls with iptables
- Access and error log tracking.
- Setting up Fail2Ban.

Session Breakdown: Setup

Friday 10 May (09:00 - 10:30):

- Setting up a database as a service.
  - Recap Docker.
  - Setting up PSQL in Docker.

Friday 10 May (11:30 - 13:00):

- Setting up a database as a service.
  - uploading data to the database.
  - adding users to the database.

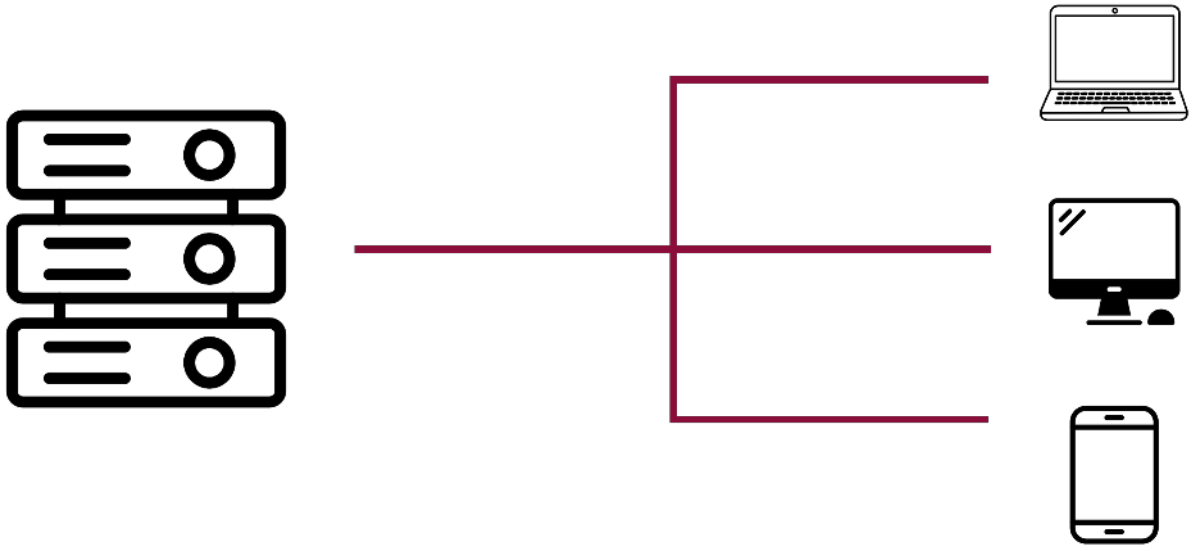
Friday 10 May (14:00 - 16:00)

- Deploying RStudio as a service.
  - Installing RStudio server.
  - Connecting RStudio to a domain using Nginx.
  - Connection RStudio to the PSQL database in the container.
- Bonus: monitoring Linux host metrics with Prometheus and Grafana.

What is a server

A server is actually just a computer. Just like a laptop, it has core components of CPU, RAM and storage. However, a server is distinct from your typical laptop in that it is built with parts that are designed to function 24/7. It is also typically set up to allow direct incoming connections more readily than a laptop.

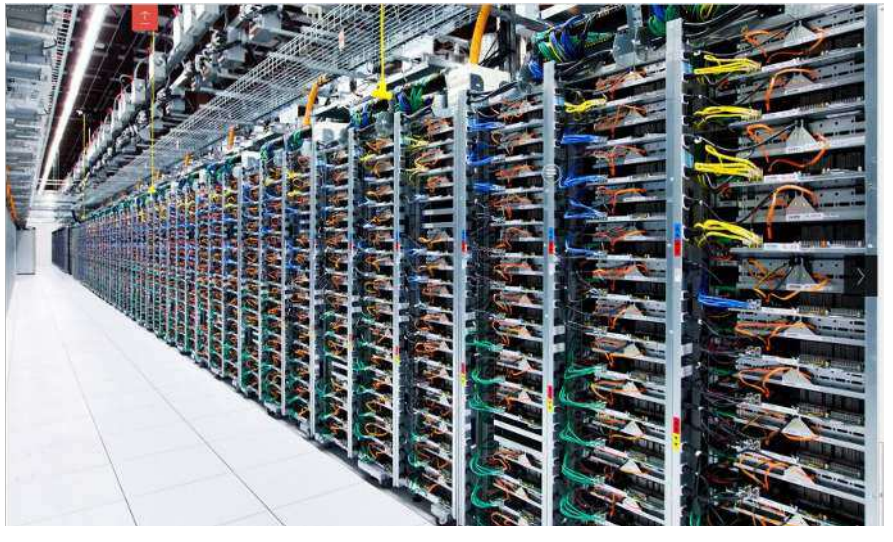
Many things are technically indistinguishable from computers as we commonly know them - mobile phones, servers, even cars and fridges. However these computers are typically named after the **purpose that they serve**. So think of a server more literally as a computer that serves you outputs to your inputs.



Cloud providers

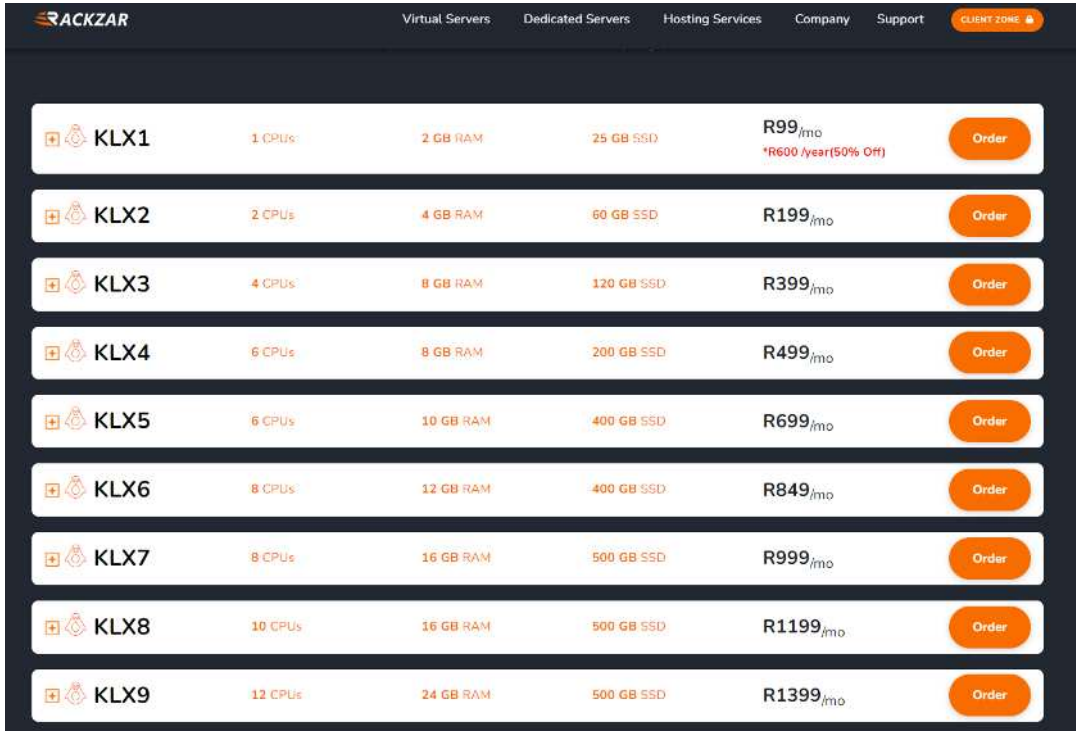
Due to the increases in processing and storage capacity and innovations in software development called a **hypervisor**, it is possible to partition a single machine to host multiple virtual machines on a single server. Of course, cloud doesn't actually mean cloud. What cloud means is remote access to what is typically a virtual machine located in a huge data centre.

Cloud services can be expensive, but they provide many benefits to a situation where you are looking to test out a a service (like a dashboard), in that they offer easily scalable and accessible computing resources. Small machines are often reasonably priced, and you are able to terminate the services at any time and only pay for the time that you used.



Your cloud server

There are many VPS providers. The one we work with most often is Rackzar. The physical servers are located in data centres in Cape Town and Johannesburg. To turn you all into MVPs, we have acquired servers for you all to use to practise the training content.

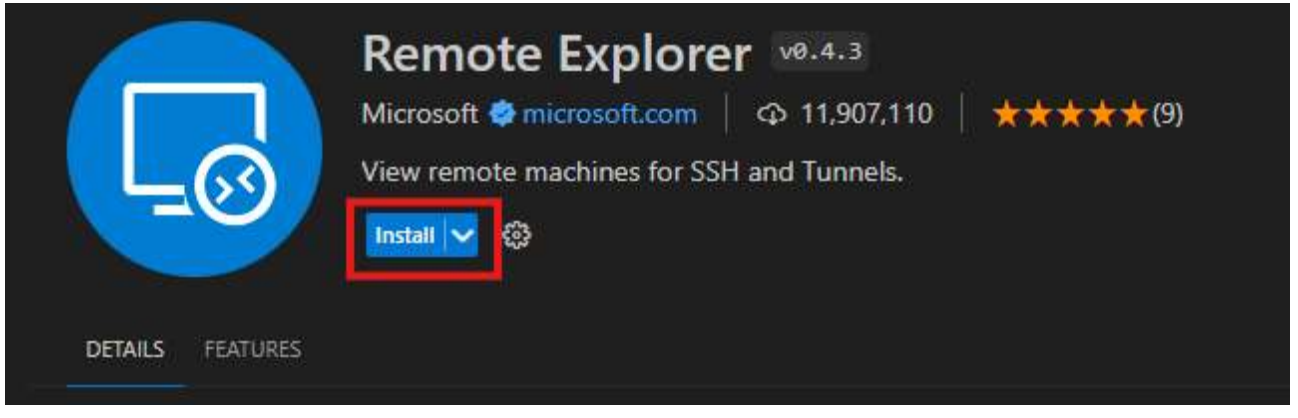


Login to your server

Setting up VS Code

We are going to use the Visual Studio Code source-code editor. The first step is to install VS Code which you have hopefully already done.

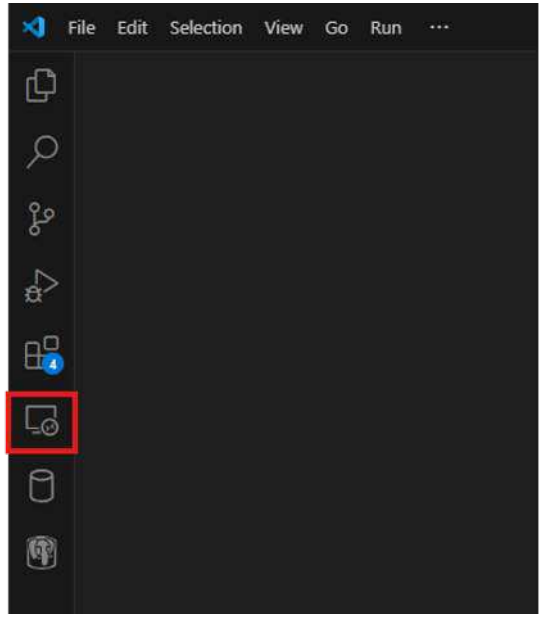
After opening VS Code we'll have to install the extension **Remote Explorer** which will help us access our VPS via SSH. SSH stands for **S**ecure **Sh**ell, which uses asymmetric cryptography to secure the content of your session. This means that if anyone is spying on your session somehow, they won't be able to make any sense out of what they see because it will be **encrypted**.

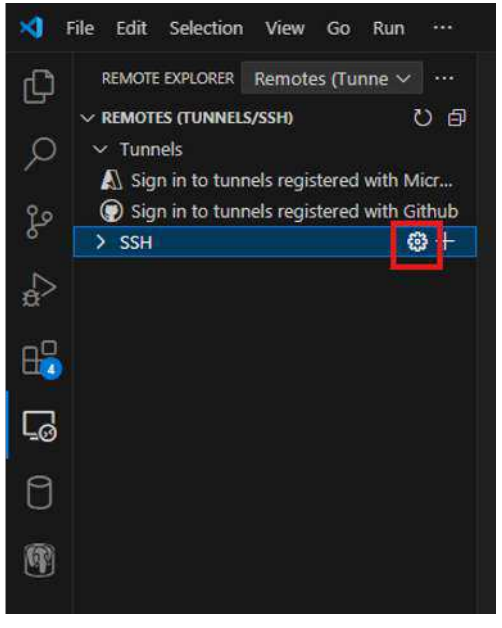


Configuring Remote SSH access

Before accessing our VPS using the Remote Explorer's SSH functionality, we have to give VS Code the details for how to access our VPS. We do this in the **configuration** file, typically known as a config file.

After installing Remote Explorer, navigate to your Remote Explorer, click on the cog next to the SSH tab (as illustrated below). This should open a menu in the top of the window where you can access your config file.





Adding details to the config file

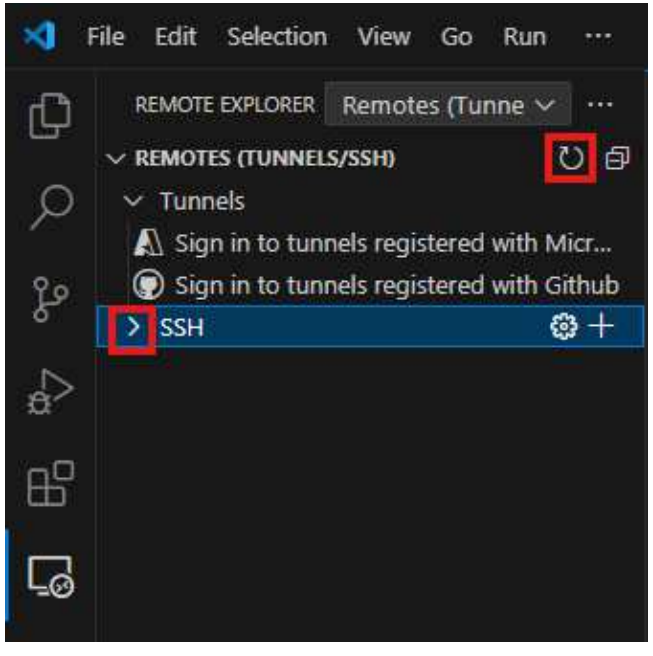
After opening the config file, you'll have to add the following information. In this case `your_server_name` is a name that you choose to refer to your server in VS Code's Remote Explorer (give it a descriptive name). End your server name off with `_root` the reason for doing this will become clear later on. `your_server_ip` is the IP-address of your VPS.

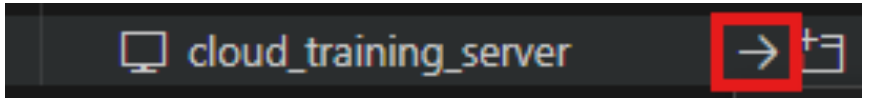
Save the config after adding the information and close it.

```
Host your_server_name_root
  HostName your_server_ip
  Port 22
  User root
```

Accessing your server

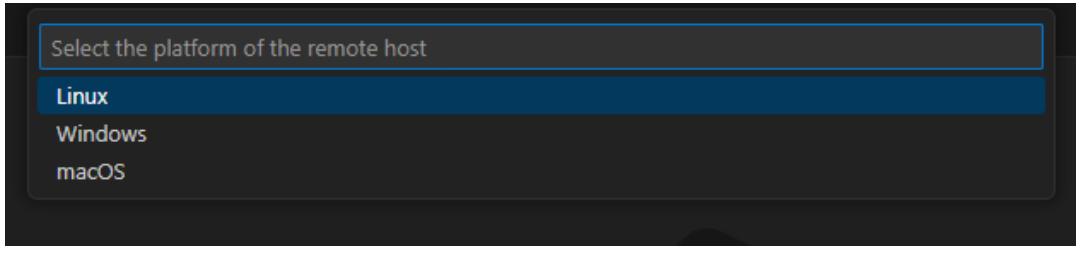
You should now be able to see your server (with the name you gave it) when you click refresh on your Remote Explorer tab and then expand the SSH tab. You can connect to your server by hovering over it and selecting to connect to it in the current session or to connect to it in a new session.



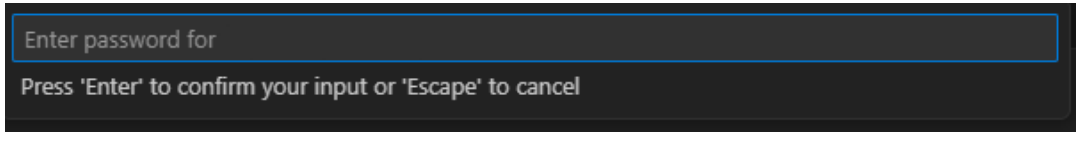


Logging in to your server

You should see options for the software of the server and choose the Linux option.

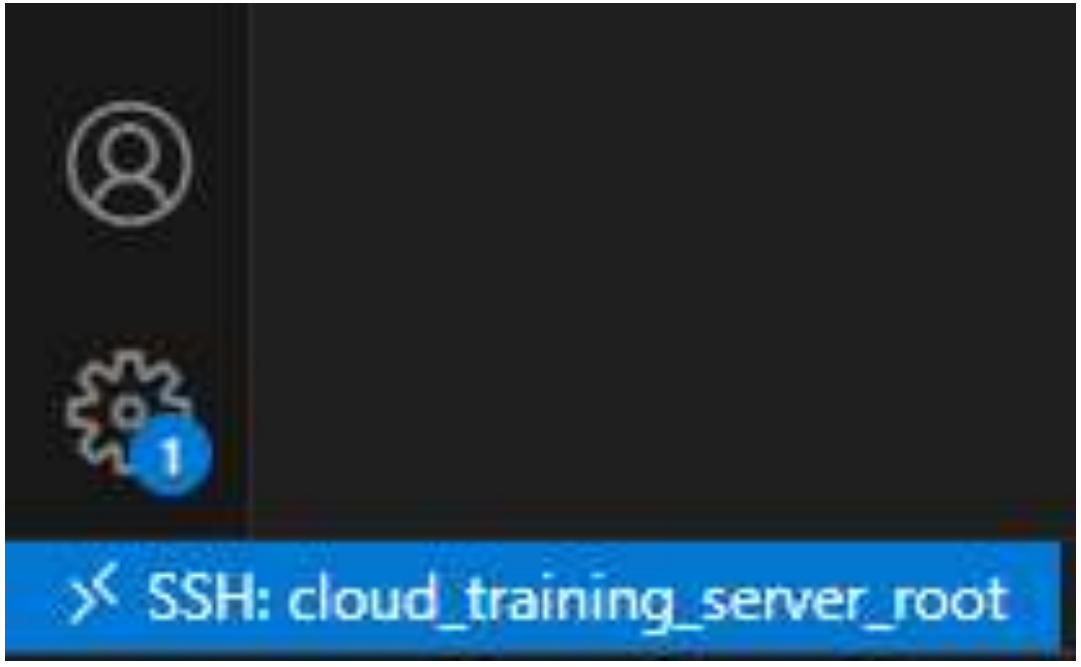


You should then be prompted for you password. Type your password and press the Enter key.



Accessing the terminal

You should now be able to see that you are connected to your server via SSH in the bottom left corner of your window.



We can access the terminal by using the shortcut CTRL + *backtick* which is usually located below the ESC key on your keyboard.

This should open your terminal for `root@your_server_name`. While VS Code has many helpful supporting functions, a terminal session is all that we need to get started on setting up our server.

Linux, Less and Vim

The command line

Whenever we talk about *black screen*, *command line* or *shell* we are essentially talking about the interface that takes input from the keyboard and sends it to the operating system (OS). Every button click on a GUI is translated into some sort of command - using the terminal just allows us to skip through the front end.

Almost all Linux distributions supply a shell program from the GNU Project called `bash`. This is going to be the primary way that we manage our server.

Try these basic commands:

```
date
```

```
free -h
```

```
cal
```

Navigating Linux

Moving around Linux can take a while to get used to, however with repetition comes familiarity. You should already have some exposure to this system from the Foundations training.

Just like any Windows system that you will be used to, all *Unix*-like systems use a folder structure that follows a tree structure. The top-most level is called the *root* directory. You can list the contents of a given directory using `tree` or `ll`

Obviously looking at files in your `home` directory doesn't take you very far. We need to be able to navigate the file system in a quick and efficient manner. The `cd` command in Linux is a powerful way to navigate the tree folder structure that is the file system.

Navigating Linux

The two main methods for traversing the tree is: (1) Absolute Paths and (2) Relative Paths:

- **Absolute Paths** begins with the root redirectory `/` and expands to the folder you are interested in: `/home/hanjo/Data`
- **Relative Paths** starts at the working directory and starts navigation from there. These paths have a special notation, a single dot (`.`) and a dot dot (`..`). The `.` notation refers to the working directory, and the `..` notation refers to the working directory's parent directory.

When changing directories in Linux, **TAB** is your best friend!

Navigate to the `/usr/bin` directory and list all the files.

Notes about filenames in Linux

Filenames in Linux are quite special and if you have worked closely with someone who works in Linux, you would have noticed some things. First and foremost:

- NEVER use a space in filenames use an underscore (`_`) instead - thank me later ;-)
  - ex. `this file name sucks.txt` where `this_is_much_better.txt`
- Filenames that start with a `.` are hidden files. The `ls` command will not list these unless you use a *parameter* `ls -a`. These files usually relate to configuration settings.
  - ex. `.bashrc`.
- CASE MATTERS, so dont ever use Capitals for folders or filenames - it gets confusing.
  - ex. `This/path/IS/different/`. from `/this/path/is/different/`
- Linux does not have any concept of "file extensions". So remember to name your files in an appropriate manner if you would like them to be readable by the correct application.
  - ex. `mypdffile` and `mypdffile.pdf` is the same

See this presentation by Dr. Anna Krystalli for further tips on file naming.

Creating folders

Apart from knowing how to navigate folders, we must also know how to create files and folders.

The basic commands for this is:

- Create folder

```
mkdir scripts
mkdir scripts data analysis
```

Viewing contents of files

To view the contents of a file, we use a program called `less`. This is an important program to use, as it allows for the viewing of a file **without running the risk of inadvertently changing some unknown but critical parameter** in our system setup config. A useful principle to live by when creating or managing any kind of software, but especially server setup is:

> I am an idiot, and I probably did something stupid that will break everything

Committing this to heart will force you to proceed with caution and write code that aims to be as idiot-proof as possible.

> The only way to write good code is to write tons of bad code first. Feeling shame about bad code stops you from getting to good code.
>
> — *Hadley Wickham*

Viewing contents of files

Lets start by looking at the users on the system:

```
less /etc/passwd
```

Navigation:

- **G** - Move to the end of the text file
- **g** - Move to the beginning of the text file
- **10g** - Move to the nth line
- **q** - Exit

Forward Search:

- **/characters** - Search forward
- **n** - Search forward
- **N** - Search backwards

Editing files

Browsing the server contents under the protection of `less` is all well and good, but if we want to get anything done we will have to start actually editing and writing files. To do this, we will use `vim`, a command-line file editor. `vim` is a critical tool to know when doing system adminstrator tasks, as you may not necessarily have access to richer tools like VS Code in times of crisis!

Basics of editing a file

⚠️ Follow my commands before typing. Do not type anything yet!

> Remember, if something bad happens just press ESC a couple of times and then exit VIM with `:q!`

```
vim owner_information.txt
```

- In VIM, every keystroke is a specific command, this type of editor is known as a *modal editor*.
  - VIM starts by going into *command mode*, which means it expects commands, NOT input text.

To type something we must go to *Insert Mode*. To do this, type `i`. You should see the following at the bottom:

```
-- INSERT --
```

Now, type the following:

```
[owner] James Scott
```

Save and exit by pressing `ESC` and `:wq`

Basics of editing a file

What happens if you happen to have made a mistake? To delete a file, use the `rm` command.

⚠️ In Linux, when you delete the file is gone forever. So be careful!

```
rm owner_information.txt
```

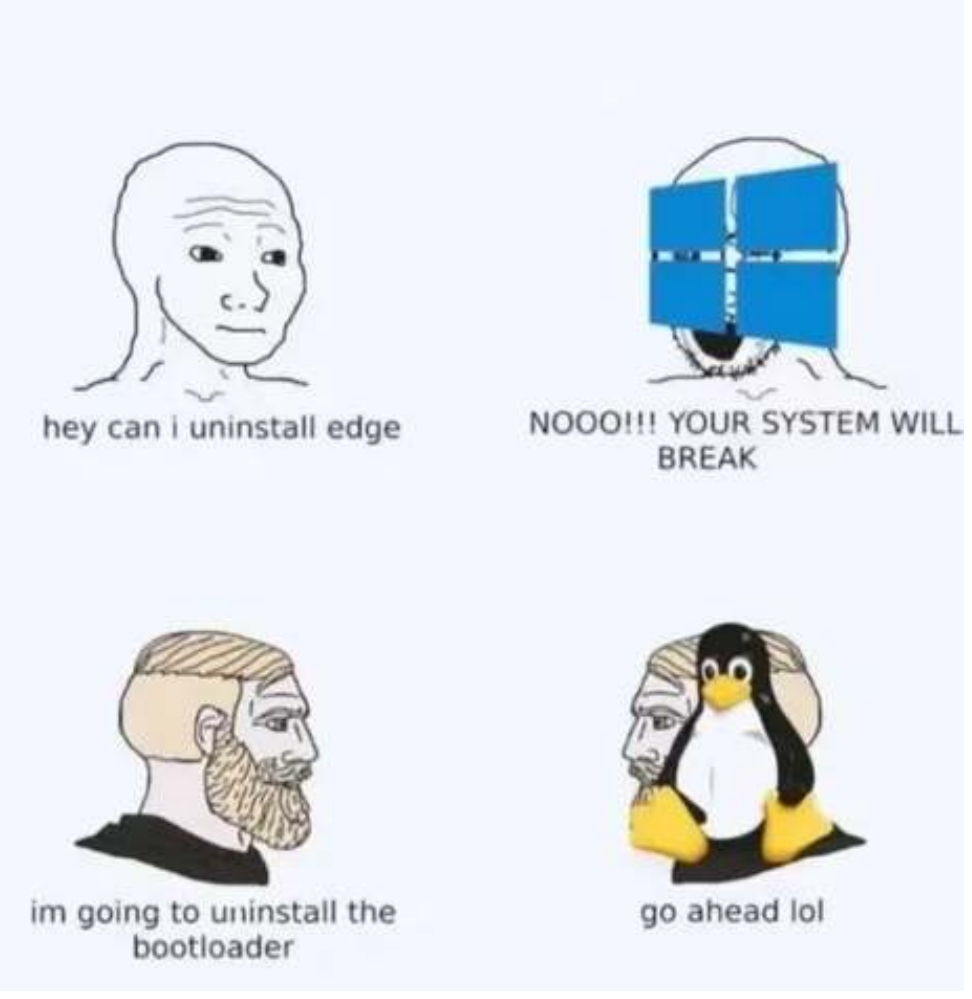
With the basics refreshed, we are now ready to get started with some sysadmin.

Creating users

The root user (#)

Root is powerful!

`root` is a superuser and `#` indicates the root user is being used at the command prompt `user@machine:location#` (normal users have a `$` instead of `#`). Working with root exclusively can compromise security or cause other bad things to happen. Linux is notoriously open and allows the user a lot of freedom.



Different type of users

Superuser

> With great power comes great responsibility!

On a Linux system a **Superuser** refers to a user who has unlimited access to the file system with privileges to run all Linux commands. The key difference compared to `root` users is that `sudo` users must preface commands they wouldn't ordinarily be able to execute with `sudo`, and must enter a password. `root` users can execute any command with no warning given.

- This responsibility is mostly given to experienced SysAdmins. The reason being there is no "take-backsies" in linux. Once a command has been executed under `sudo` (superuser do) , there is almost never a way to reverse the execution (ex. deleting a file).
- The Superuser/Root is also responsible for setting up security and thus, limiting the power to a single (or very few individuals is preferred).

Setting up a new user

It is a good idea to create a separate user with `sudo` privileges that you use for day-to-day tasks. Not only does it add some level of protection by forcing you to use `sudo` and enter your password, it also adds allows you to protect your server in the cybersecurity sense, which we will touch on later.

Use the `adduser` function to add a user called [your name]. You will have to create a password for this user, which will be used later. Make sure to create a **secure** password that is not easy to brute-force! Use an online password generator like LastPass, and ideally store this password in some sort of password manager.

You don't have to fill in the additional information, but you're welcome to do it if you feel like it.

Superuser privileges

Since we're still going to need some superuser functionality, we should grant our newly created user superuser privileges. We can do this by adding the user to the `sudo` group. This will be done using the `usermod` function, which is short for for **modify a user account**. Let's have a look at what this function does using our best friend in the whole universe, `man`.

Now that we have a grip of what `usermod` does, we can put it into practise:

- We check which groups the user is a member of using `id`
- We use the `usermod` function with the `-aG` option

Securing user access with SSH keys

An SSH key is a way of establishing a secure connection with a server. SSH keys are similar to passwords, but are often less susceptible to brute force attacks as they are longer. If you think of a password as a secret phrase you say to a guard to allow you access to your house, SSH keys are the equivalent of using a key that only you have to a locked gate.

SSH keys come in private-public key pairs. The public key can be thought of the lock in the gate that the key is fitted to. SSH security is driven by the technique **asymmetric cryptography**, as is explained here.

Generating public-private key pairs

Each user can generate their own SSH key. To create a new SSH key for our user, switch using `su [user_name]` (`su` = **switch user**). You will have to enter the password that you just generated - hopefully you have kept it somewhere.

Once you have switched, your should see your prompt shift to reflect your new user. Run `cd` to make sure that you are in your home directory. We are also going to need a directory to store our keys in. The default directory is called `.ssh/`, and it should be created in our home directory.

Once you have done that, run the following to generate your SSH key pair:

```
ssh-keygen
```

This will prompt you to provide a name for the resulting files. The default name is `id_rsa`, but you can use something more intuitive like `cloud_training_key`. Make sure to add the relative file path (`.ssh/`) to make sure that it lands in the correct place.

Authorising SSH keys

We now have a valid key that works in a lock. The only thing that remains is to fit it to the gate of our house. The way this works in practise is that when you attempt an SSH connection using your name, the `ssh` service looks for a file called `authorized_keys` in `.ssh/` in your home directory. If the private key that you supply matches a public key in this file, your access is granted. So all that remains is to add our public key to this file:

```
cat cloud_training_key.pub > authorized_keys
```

Double check that you have added the correct key to the `authorized_keys` file using `less`. You should see a comment at the end of the file that looks like `james@VM01`.

Finally, we are going to need to store our **private key** on our local laptop. The technique here is low-tech: paste the contents of the key using `cat cloud_training_key`, and copy into a text file. Save this file in a sensible place!

Logging in with SSH key

We can now reconfigure our VS Code Remote Explorer to access our server via our new user and the SSH keys we just generated. It is easier to just copy and paste the file path from your file explorer, especially on Windows.

```
Host your_server_name
  HostName your_server_ip
  Port 22
  User [user_name]
  IdentityFile "path/to/your/private/key/private_key_name"
  IdentitiesOnly yes
```

Once that is done, reattempt login using your new user.

Securing our server

We should now be all set to start running sysadmin tasks. Our first concern will be to prevent any unwanted logins from our server. The `ssh` service keeps a record of all login attempts - have a look at the file `/var/log/auth.log`. Be shocked.

Cybersecurity is another deep and complex field that can swallow up an entire career. As practitioners of MVP services, it is not worth attempting to safeguard your services against the highest-grade cyber attacks. The most watertight and easy to implement defensive solution is probably to diligently back up your code and data.

Some general principles, however, do apply. In general, the biggest risk to system security is **human error**. This means that our first port of call should be analyse our own behaviour for security concerns. Some typical human errors are:

- Creating simple passwords
- Sharing passwords
- Clicking or downloading anything from a strange/suspicious link
- Downloading unaccredited packages

Securing our server

All that said, it would be frustrating to have to set up our server over and over after falling victim to `script-kiddie` attacks.

We can implement some protocols that will help to protect our fresh server from unwanted brute-force or DoS attacks. These are:

- Disabling root login
- Disabling password login (SSH key login only)
- Enabling a **firewall**
- Enabling Fail2Ban, a software that limits the number of unsuccessful login attempts any one IP address can make.

Securing SSH

Disabling root and password logins

The configuration settings for the `ssh` service are stored in a file called `sshd_config` in the `/etc/ssh` directory. Before it starts, the `ssh` service reads this file in order to know how it should function.

Before you edit, make a copy of the file and save it as `sshd_config.dist` (in case something goes wrong while editing the original). Then, edit `sshd_config` using `vim` in the following areas so that the following three options are set to 'no' (check that the lines are not commented out):

- PasswordAuthentication no
- PermitRootLogin no
- UsePAM no

PAM stands for **P**luggable **A**uthentication **M**odules, which is a more recent addition to SSH which allows custom authentication methods to the `ssh` service. This can override other settings if left specified as **yes**.

Disabling the root user and password login

After the edits from the previous slide, we need to restart the `ssh` service.

WARNING NOTE ⚠️: If you restart the service without checking that your SSH Key login works, you might lock yourself out of the server. Do **NOT** restart the service without checking that you can login with your SSH key. ⚠️

When you are ready, restart the service with `sudo systemctl restart ssh`

DO NOT CLOSE YOUR CURRENT TERMINAL. Keep your current session open and open an additional session to check if you can log in successfully, just in case.

Test that you cannot use root to login, and test that you are not offered a chance to enter a password when not authenticating with an SSH key. This cuts out a good deal of brute-force risk. We will get back to more security measures later.

Managing permissions

Permissions

Now that we are feeling relatively sure that our new machine won't be torn to pieces by a 5 year old, we can turn our attention towards one of system administrators' biggest headaches: **permissions**.

Why are permissions important?

Imagine you are enjoying an incredible Sunday lunch with your family. You have savoured a delicious meal, and several mubimba. Nature calls, and you decide to leave your current Primus Nini on the floor in the middle of the passage. As you return, your 8 year old nephew runs down the passage and knocks your primus all over the floor. You are devastated.

A question: do you have any right to be angry with your nephew?

Permissions

These are the sorts of questions that a system administrator must think about. You cannot really be angry with your nephew, because they did not know any better and they did not act with malice. In the same way, you as a system administrator cannot grant `sudo` privileges to a Linux user with the total experience of two weeks and not expect to take the blame for the inevitable system crash that ensues when the user gets lost in the root directory.

Setting permissions refers to the selective granting of the ability to read, write or execute files or scripts for according to what is required for each user to do their job. Correct setting of permissions can offer two things:

- peace for mind for the **system administrator** that the server will not be compromised by a mistake by a user who doesn't know better
- peace of mind for the **user** that they will not compromise the server by making a mistake

Understanding permissions

There is a common saying in the Linux community:

> *Everything in Linux is a file*

This is largely true. Word documents (encoded), system configuration files and csvs are at the end of the day all just raw characters in a plain-text file. While the reality is a bit more complex, looking at things from our MVP perspective it makes it easier to understand the **3 basic permissions** as follows:

- **read (r)**: the ability for a user to see what a file contains
- **write (w)**: the ability for a user to make edits to the contents of a file
- **execute (x)**: the ability for a user, via the use of a program (like python or R) to execute the contents of the file

These permissions can be set independently for any given file. This means that a file can have any combination of these three permissions.

Understanding permissions

Only being able to grant one set of permissions per file is quite limiting. Imagine if you have a sensitive dataset on the server that should only be read by a select few. These users would then have to be granted `sudo` permissions to read the data. This is inflexible and unhelpful. Fortunately, Linux provides a way for us to assign a set of permissions to different kinds of users:

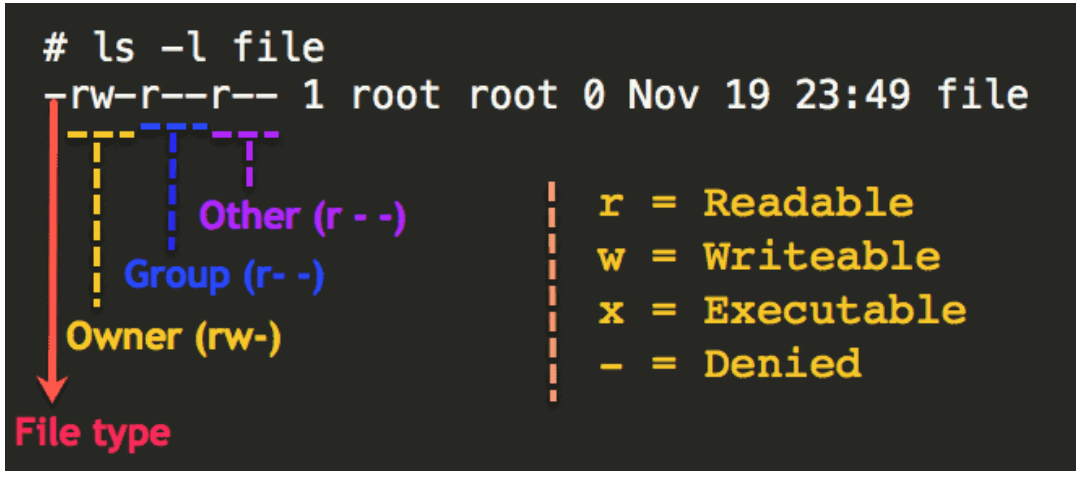
- **owner**: typically the creator of the file, the owner refers to **one user only**. Every file has an owner.
- **group**: a set of users tied together by membership of a defined group. Every file has a group assigned to it, either by default or manually after creation
- **other**: any user that is not the owner or a member of the assigned group

To test out privileges, make a test file in your home directory called `hello.sh` that looks like this:

```
#! /usr/bin/bash
echo hello
```

Understanding permissions

In general, the file permissions are reported as follows:



The `ll` command is very helpful in understanding the privileges assigned to a file. Run it on your home directory, and the relevant entry should look like this:

```
-rw-r--r--  1 james james    28 May  8 21:57 test.sh
```

What does this imply for the permissions for our new file?

Changing permissions

As the file extension implies, this is a shell script that needs to be executed. What happens if we try to execute via `./test.sh`?

As expected, permission is denied. In order to edit the permissions so that we can do so, we must make use of the `chmod` command.

`chmod` has many different specifications, but the simplest is the following:

- start with `chmod`
- add the initial of the user group that you want to edit permissions for (u,g,o)
- define if the permissions are to be added (+) or removed (-)
- add the types of permissions
- finally, specify the file or files it should be applied to

In this case, we would specify `chmod u+x test.sh`. Once this is done, retrying `./test.sh` is successful. You can also notice that the appearance of the file when executing `ll` has changed.

Using this pattern, assign permissions to `test.sh` so that the owner has read permissions only, the group has write permissions only, and other users have execute permissions only.

Application

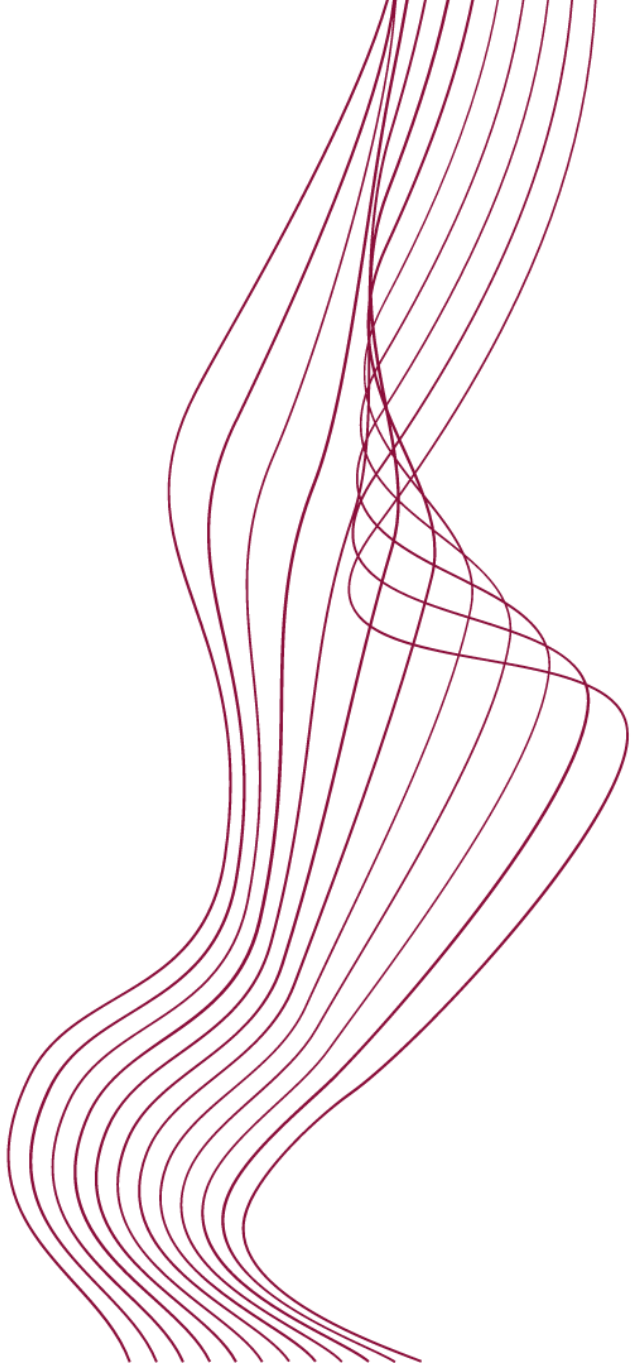
Determining privileges is best done on a case-by-case basis. Most of the time, Linux will handle permissions for you with little input needed. However, a good application of some permissions magic can be shown in the creation of a **central data directory** on the server. This directory will grant access to a data storage facility where only members of a group can **read** and **write** the data, and others can only **read**. This involves the following:

1. Create a group called `analytics` using `groupadd`
2. Create the directory `/home/data`
3. Set the ownership to `james:analytics`
4. Set the sticky group to analytics using `chmod g+s`
5. Create test users and assign them to group `analytics` to test

Securing your server

Agenda

1) What is a firewall
2) iptables as a firewall
2) Logs to monitor our system
3) Fail2Ban for protection



Firewalls 

Firewalls

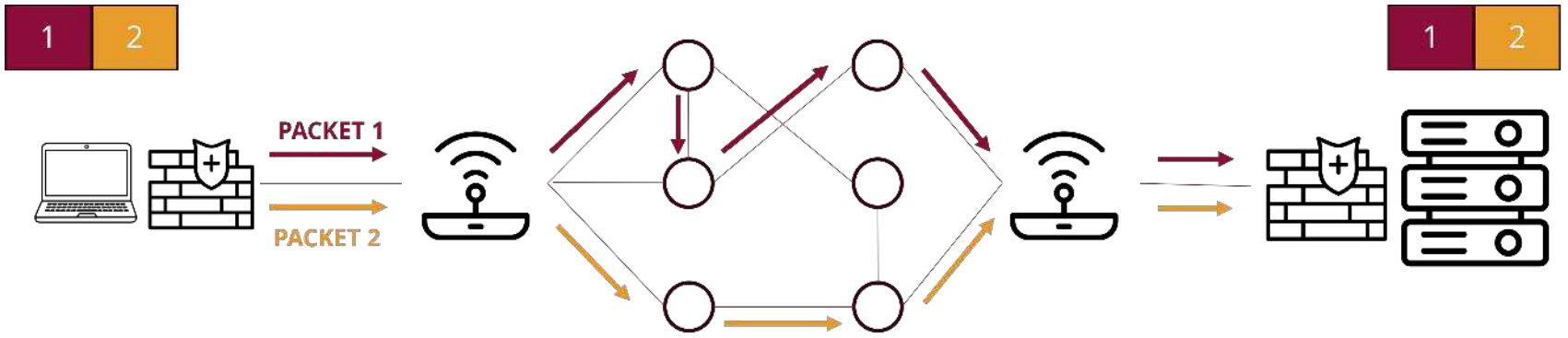
A firewall is simply a **network security system** that **monitors and controls** incoming and outgoing **network traffic** based on specified criteria.

> Technically anyone with a internet connection can send packets to our server. A firewall checks the packets against a set of rules and then decides whether to allow them to enter the system or to reject them.

What are packets?

Packets are simply smaller segments of larger messages. They increase the efficiency and reliability of transmitting data over a network.

Once these packets reach their destination the receiver reassembles them into the larger message.



71point4
NUMBERS TO LIFE

cenfri

Firewalls with `iptables`

Overview

The tool we use to set up our firewall in this course is `iptables`. It will enable us as the system administrators of our VPSs to configure the IP packet filtering rules for the Linux kernel firewall.

`iptables` organises these filters into different tables (hence then name) which contain chains of rules which determine how network packets are treated.

Chains

An `iptables` **chain** is a collection of rules that are compared, in order, against packets that share a common characteristic (such as being routed to the Linux system, as opposed to away from it). The most important built-in chains for our purposes are the INPUT, OUTPUT, and FORWARD chains (specifically in the filter table):

- The **INPUT** (incoming packets) chain is traversed by packets that are destined for the local Linux system after a routing calculation is made within the kernel (i.e., packets destined for a local socket).
- The **OUTPUT** (outgoing packets) chain is reserved for packets that are generated by the Linux system itself.
- The **FORWARD** (routed packets) chain governs packets that are routed through the Linux system (i.e., when the `iptables` firewall is used to connect one network to another and packets between the two networks must flow through the firewall).

Matches

An `iptables` **match** is a condition that must be met by a packet in order for `iptables` to process the packet according to the action specified by the rule **target**. We discuss **targets** on the next slide.

> For example, to apply a rule only to TCP packets, you can use the --protocol match.

The table below displays some of the most important `iptables` matches, but we can consult the `man` page for more information.

Option	Description
--source (-s)	Match on a source IP address or network
--destination (-d)	Match on destination IP address or network
--protocol (-p)	Match on an IP value
--in-interface (-i)	Input interface
--output-interface (-o)	Output interface
--state	Match on a set of connection states
--string	Match on a sequence of application layer data bytes
--comment	Associate up to 256 bytes of comment data with a rule within kernel memory

Targets

A target specifies the action to take should the matching criteria be met. The most important `iptables` targets are listed in the table below.

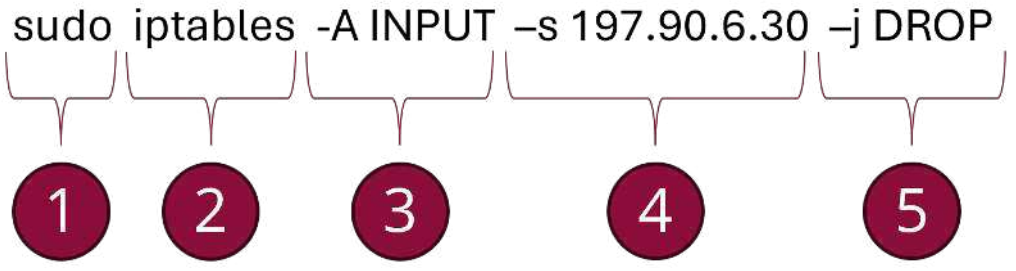
Option	Description
ACCEPT	Allows a packet to continue on its way
DROP	Drops a packet. No further processing is performed, and as far as the receiving stack is concerned, it is as though the packet was never sent.
LOG	Logs a packet to syslog
REJECT	Drops a packet and simultaneously sends an appropriate response packet (e.g., a TCP Reset packet for a TCP connection or an ICMP Port Unreachable message for a UDP packet).
RETURN	Continues processing a packet within the calling chain.

iptables

`iptables` rules are merely user defined commands that manipulate the network traffic.

Setting rules

1. Only the superuser can add rules to `iptables`.
2. Call the `iptables` utility.
3. Specify what **action** to take (Append, Delete, Replace, Check or List) and for which **chain** (in this chase INPUT).
4. Specify a **matching component** (in our case the IP address).
5. Specify the **target**.



In practice

We can view the current `iptables` rules with line numbers for each rule using the command below. Note that if we don't specify the `-t` option in `iptables` the default table (filter) is selected.

```
kiza@youngsta:~$ sudo iptables -vnL
sudo: unable to resolve host youngsta: Name or service not known
[sudo] password for kiza:
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
kiza@youngsta:~$
```

What can you see?

- Which chains are visible?
- What is the default target for each chain?

In practice

Let's define some basic rules to control the traffic flow to and from our machine in the network.

> Append a rule to the INPUT chain that accepts incoming packets if they are related to already established connections.

```
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

> Append rules to the INPUT chain that accept loop-back connections and pings (notice the use of comments).

```
sudo iptables -A INPUT -i lo -m comment --comment "Allow loopback connections" -j ACCEPT
sudo iptables -A INPUT -p icmp -m comment --comment "Allow Ping to work as expected" -j ACCEPT
```

> Allow SSH connection.

```
sudo iptables -I INPUT 4 -p tcp -s 0.0.0.0/0 --dport 22 -m comment --comment "SSH access" -j ACCEPT
```

Exercise

Let's have a look at the rules that we've defined so far.

```
kiza@youngsta:~$ sudo iptables -vnL --line-numbers
[sudo] password for kiza:
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1    10319 1200K ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
2        0     0 ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0           /* Allow loopback connections */
3       36  3024 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0           /* Allow Ping to work as expected */
4      115  6900 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:22 /* SSH access */

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
kiza@youngsta:~$
```

How can you change the SSH rule to ensure that only you can connect via SSH? Consult the `man` page and make the necessary adjustment (TIP: you can google "what's my ip" to get the IP address of your local machine).

10:00

In practice

The default policy of the INPUT chain is still set to ACCEPT. This means that if a packet traverses this chain and is not matched by any of the rules, the packet will be accepted. We need to change that to allow only the connections specified already.

```
sudo iptables -A INPUT -m comment --comment "Drop all other connections" -j DROP
```

```
kiza@youngsta:~$ sudo iptables -vnL --line-numbers
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1    15793 1795K ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
2        0     0 ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0           /* Allow loopback connections */
3       56  4612 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0           /* Allow Ping to work as expected */
4        0     0 ACCEPT     tcp  --  *      *       your_ip              0.0.0.0/0           tcp dpt:22 /* SSH access */
5     6294  937K DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0           /* Drop all other connections */

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
kiza@youngsta:~$
```

In practice

We are making great progress towards securing our server! Another important step that we need to take is to make these `iptables` rules permanent.

INSTALL `iptables-persistent`

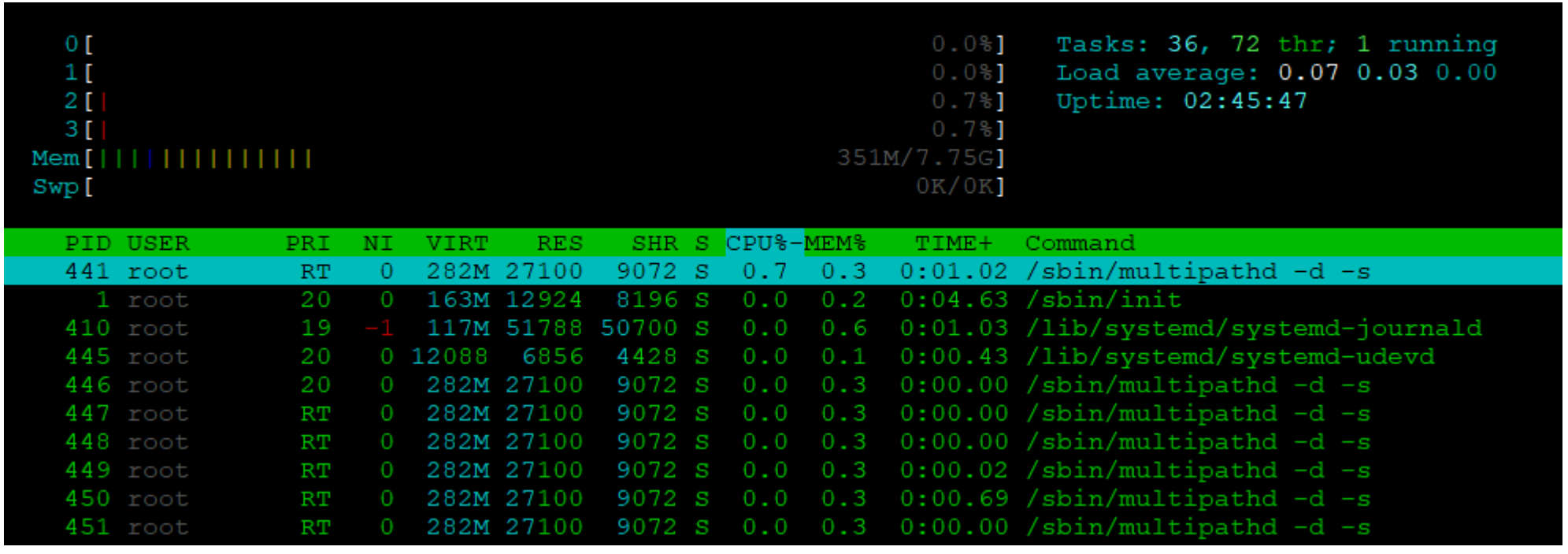
```
sudo apt-get install iptables-persistent
```

After each edit to the `iptables` rules run the following to ensure that the rules are loaded each time the server is rebooted:

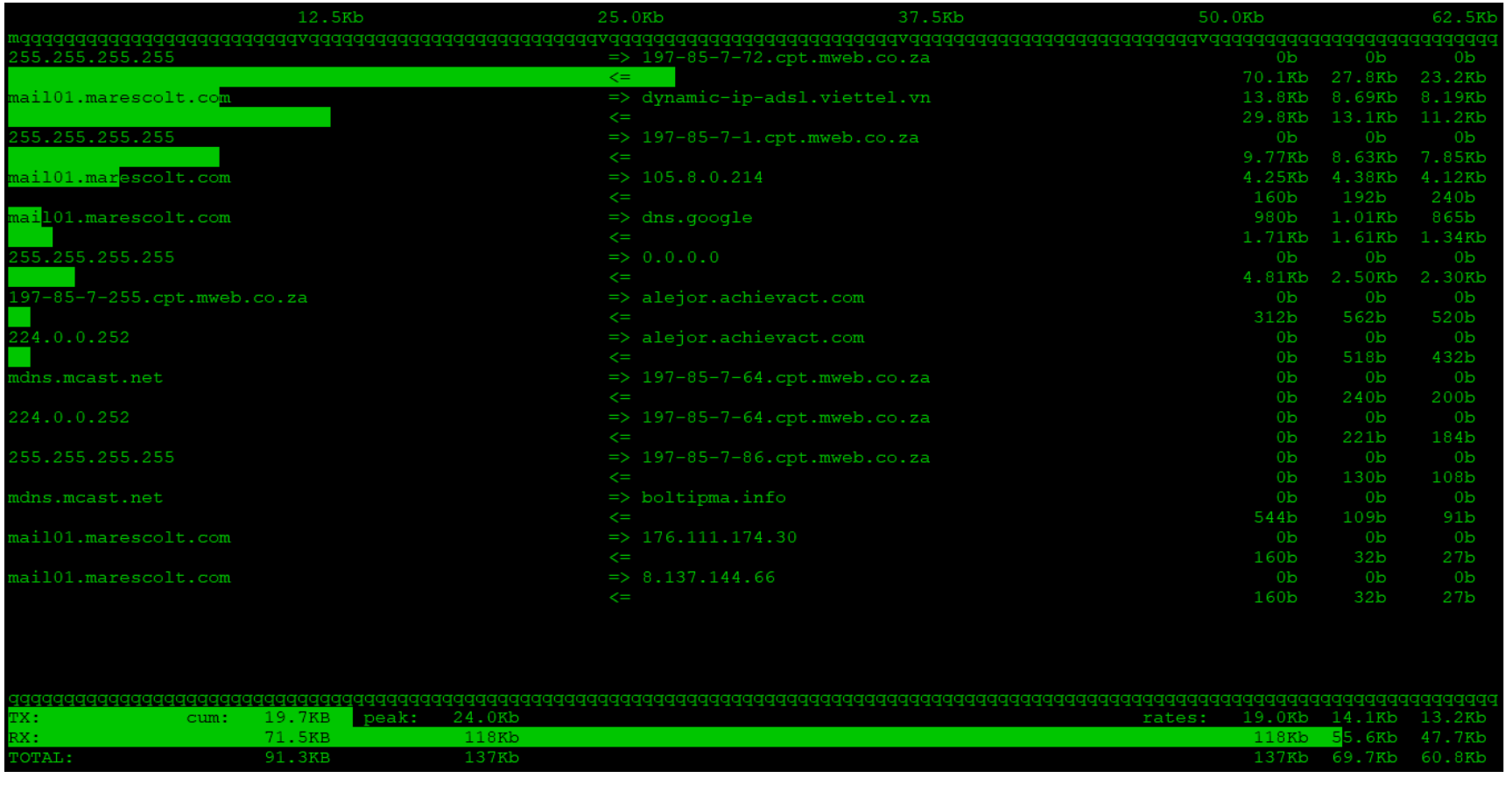
```
/sbin/iptables-save > /etc/iptables/rules.v4
/sbin/ip6tables-save > /etc/iptables/rules.v6
```

Logs to monitor our system

htop


0[0.0%]
1[0.0%]
2[| 0.7%]
3[| 0.7%]
Mem[|||||||||||| 351M/7.75G]
Swp[0K/0K]
Tasks: 36, 72 thr; 1 running
Load average: 0.07 0.03 0.00
Uptime: 02:45:47
PID USER PRI NI VIRT RES SHR S CPU%- MEM% TIME+ Command
441 root RT 0 282M 27100 9072 S 0.7 0.3 0:01.02 /sbin/multipathd -d -s
1 root 20 0 163M 12924 8196 S 0.0 0.2 0:04.63 /sbin/init
410 root 19 -1 117M 51788 50700 S 0.0 0.6 0:01.03 /lib/systemd/systemd-journald
445 root 20 0 12088 6856 4428 S 0.0 0.1 0:00.43 /lib/systemd/systemd-udevd
446 root 20 0 282M 27100 9072 S 0.0 0.3 0:00.00 /sbin/multipathd -d -s
447 root RT 0 282M 27100 9072 S 0.0 0.3 0:00.00 /sbin/multipathd -d -s
448 root RT 0 282M 27100 9072 S 0.0 0.3 0:00.00 /sbin/multipathd -d -s
449 root RT 0 282M 27100 9072 S 0.0 0.3 0:00.02 /sbin/multipathd -d -s
450 root RT 0 282M 27100 9072 S 0.0 0.3 0:00.69 /sbin/multipathd -d -s
451 root RT 0 282M 27100 9072 S 0.0 0.3 0:00.00 /sbin/multipathd -d -s

iftop


12.5Kb
25.0Kb
37.5Kb
50.0Kb
62.5Kb
255.255.255.255 => 197-85-7-72.cpt.mweb.co.za 0b 0b 0b
<= 70.1Kb 27.8Kb 23.2Kb
mail01.marescolt.com => dynamic-ip-adsl.viettel.vn 13.8Kb 8.69Kb 8.19Kb
<= 29.8Kb 13.1Kb 11.2Kb
255.255.255.255 => 197-85-7-1.cpt.mweb.co.za 0b 0b 0b
<= 9.77Kb 8.63Kb 7.85Kb
mail01.marescolt.com => 105.8.0.214 4.25Kb 4.38Kb 4.12Kb
<= 160b 192b 240b
mail01.marescolt.com => dns.google 980b 1.01Kb 865b
<= 1.71Kb 1.61Kb 1.34Kb
255.255.255.255 => 0.0.0.0 0b 0b 0b
<= 4.81Kb 2.50Kb 2.30Kb
197-85-7-255.cpt.mweb.co.za => alejor.achievact.com 0b 0b 0b
<= 312b 562b 520b
224.0.0.252 => alejor.achievact.com 0b 0b 0b
<= 0b 518b 432b
mdns.mcast.net => 197-85-7-64.cpt.mweb.co.za 0b 0b 0b
<= 0b 240b 200b
224.0.0.252 => 197-85-7-64.cpt.mweb.co.za 0b 0b 0b
<= 0b 221b 184b
255.255.255.255 => 197-85-7-86.cpt.mweb.co.za 0b 0b 0b
<= 0b 130b 108b
mdns.mcast.net => boltipma.info 0b 0b 0b
<= 544b 109b 91b
mail01.marescolt.com => 176.111.174.30 0b 0b 0b
<= 160b 32b 27b
mail01.marescolt.com => 8.137.144.66 0b 0b 0b
<= 160b 32b 27b
TX: cum: 19.7KB peak: 24.0Kb rates: 19.0Kb 14.1Kb 13.2Kb
RX: 71.5KB 118Kb 118Kb 55.6Kb 47.7Kb
TOTAL: 91.3KB 137Kb 137Kb 69.7Kb 60.8Kb

Checking the log files

System logs deal with exactly that - the Ubuntu system - as opposed to extra applications added by the user. These logs may contain information about authorizations, system daemons and system messages.

> Authorization log

Keeps track of authorization systems, such as password prompts, the sudo command and remote logins.

```
/var/log/auth.log
```

> Daemon Log

Daemons are programs that run in the background, usually without user interaction. For example, display server, SSH sessions, printing services, bluetooth, and more.

```
/var/log/daemon.log
```

Checking the log files

> Debug log

Provides debugging information from the Ubuntu system and applications.

```
/var/log/debug
```

> System logs

Contains more information about your system. If you can't find anything in the other logs, it's probably here.

```
/var/log/syslog
```

I think you can start seeing some pattern here... Some applications also create logs in `/var/log/`:

- `rstudio-server/`, `nginx/`, `letsencrypt/`

Analysing log files

There are multiple programmes to analyse web logs (such as `goaccess`), but for now, lets do some basic analysis:

First and foremost, become root - `sudo su`. Then open up `/var/log/auth.log`.

See if you can find yourself logging on: `grep "hanjo" /var/log/auth.log | less`.

Lets also have a look at the files for a bit... `tail -f /var/log/auth.log`

Jip….

Fail2Ban 

Cybersecurity

A complex topic - we are often overwhelmed with news of data breaches, which often have real implications for our data subjects. Obviously we would like to take some precautions to avoid our services being disabled. However, there are a few things we need to remember as Data Scientists running small applications in the cloud.

1. We are not full time system engineers or security experts!
2. Our concern for security should be in proportion to how confidential or important the data/services hosted by our service is.

Therefore - if you are running a dashboard based on publicly available survey data, then your most airtight security strategy is to have all your code backed up (hopefully on GitHub, but that's a story for another day).

The cloud is a huge system protected by seasoned professionals. As such, the biggest risk to our system is probably us! What does this mean?

- don't fall for phishing attacks
- don't download strange programs
- only download recommended/trusted libraries
- don't throw passwords around!
- use secure passwords!

Protecting against common cybersecurity threats

That being said, there are a significant number of what are commonly known as 'script-kiddies' who are throwing bugs all over the internet

> Brute force attack

- These hack attempts are in essence unsophisticated - all they involve is trying all possible combinations of your password in an attempt to gain access to your server.

> Denial of Service (DoS) attacks

- This involves your server being bullied by a larger computer bombarding your machine with so many requests (even unsuccessful requests) that you end up not being able to use it at all.

Fail2Ban

Fortunately, there is a relatively simple and easy to deploy service that helps you to mitigate these attacks. It is a technique that you are no doubt already familiar with from many internet login or even phone passwords. The simple solution is to only allow a user a set number of attempts to enter a password, and after that number of unsuccessful attempts block their attempts for a set amount of time.

For linux and SSH access, we are going to implement this using `Fail2Ban`, which is an open-source, free-to-use tool:

```
sudo apt install fail2ban
fail2ban-client --version
```

Fail2Ban

Fail2Ban has two components - `fail2ban-client`, which allows the user an easy interface to configuration files, and `fail2ban-service` which refers to the actual program doing the work behind the scenes.

Fail2Ban reads through **log files** and interprets them in such a way that it is able to identify machines (IP addresses) that unsuccessfully attempt to SSH in and add them to a **jail** for a certain period of time.

The default location for all things Fail2Ban is `/etc/fail2ban/`. In the `jail.d/` directory, you will find `sshd.conf`:

- `port`: which port is being targeted for connection attempts?
- `filter`: which service do we want this jail to focus on?
- `logpath`: which logs should be read?
- `maxretry`: how many unsuccessful attempts should be tolerated?
- `bantime`: how long should the IP be chucked in jail for? Measured in seconds

Lets see if there is any IPs in jail? `fail2ban-client status sshd`

- You can also "unban" someone: `fail2ban-client set {JAIL} unbanip {IP}`.

...later we will secure our cloud DB!

Fail2Ban

It might be good to ensure that your IP doesnt get blocked (this is usually good when you have a fixed IP):

You can add specific IPs you wish to ignore by adding them to the ignoreip line. This won't ban the localhost by default. Adding the ignore list may be to your benefit if you tend to frequently leverage an individual IP address:

- `vim /etc/fail2ban/jail.local`

```
[DEFAULT]
# "ignoreip" can be an IP address, a CIDR mask or a DNS host. Fail2ban will not
# ban a host which matches an address in this list. Several addresses can be
# defined using space separator.

ignoreip = 127.0.0.1/8 123.45.67.89
```

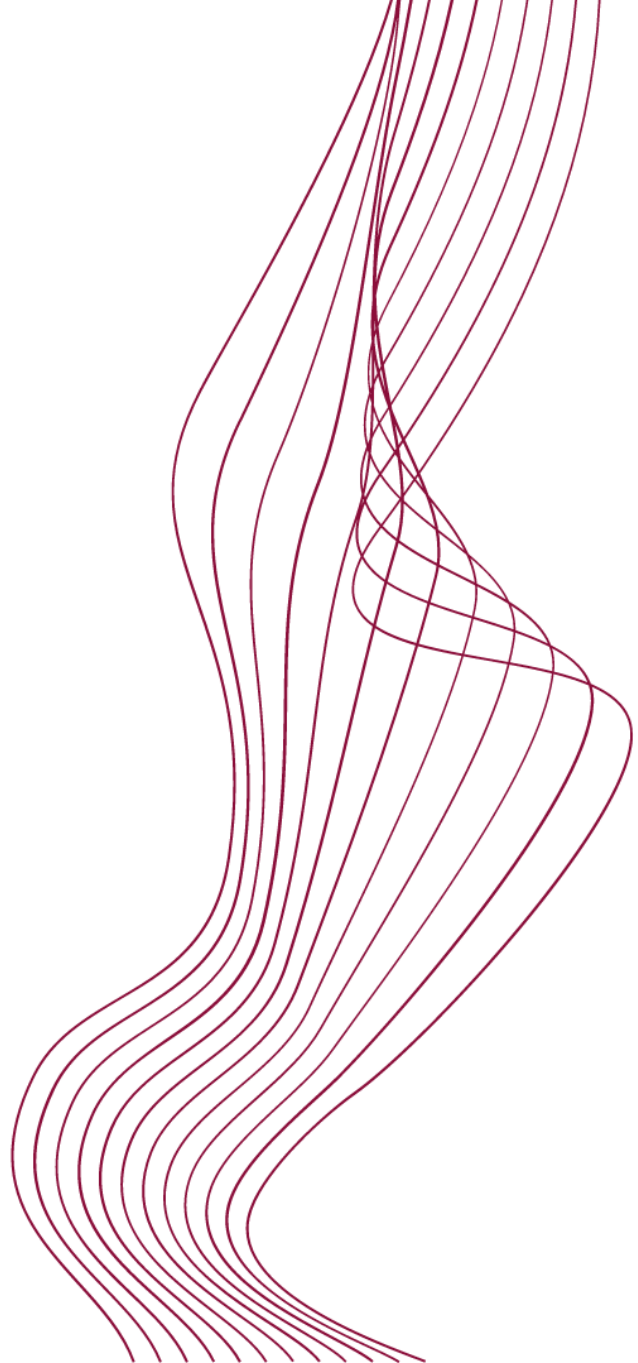
Want to whitelist IPs only for specific jails? Utilize the fail2ban-client command. Just switch `JAIL` with your jail's name, and 192.0.0.1 with the IP you intend to be whitelisted.

```
fail2ban-client set JAIL addignoreip 192.0.0.1
```

DevOps
(Day 2 - Deploying services)

Agenda

1) Agenda
2) Agenda
3) Property Market Analysis



Basics of networking

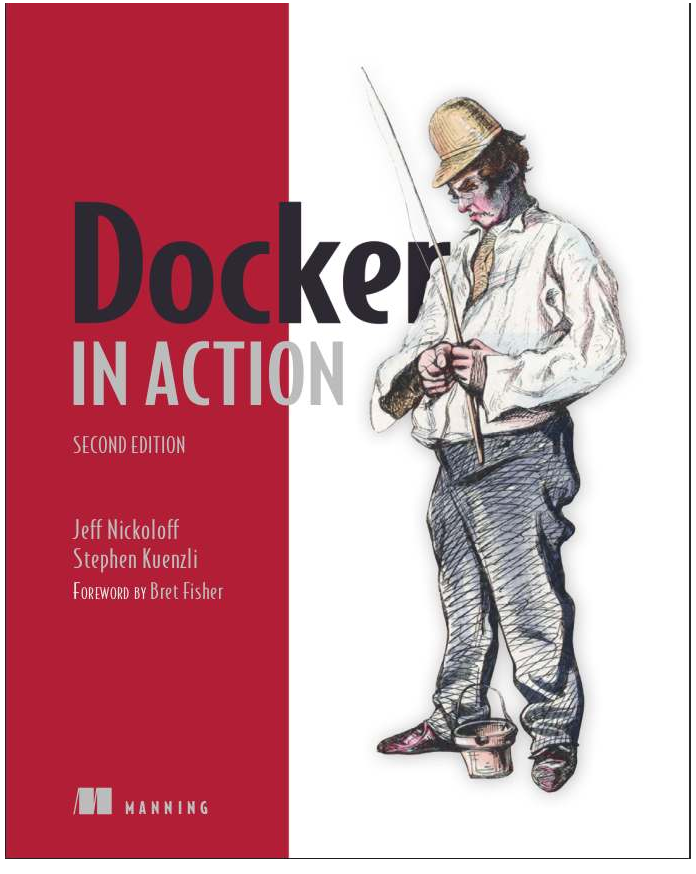
Before we dive in, it would be useful to cover some basic networking. Otherwise you will just be fumbling around in the dark.

What are IP addresses? What are ports?

Docker and DevOps

A great quote:

> A best practice is an optional investment in your product or system that should yield better outcomes in the future. Best practices enhance security, prevent conflicts, improve serviceability, or increase longevity. **Best practices often need advocates because justifying the immediate cost can be difficult.**



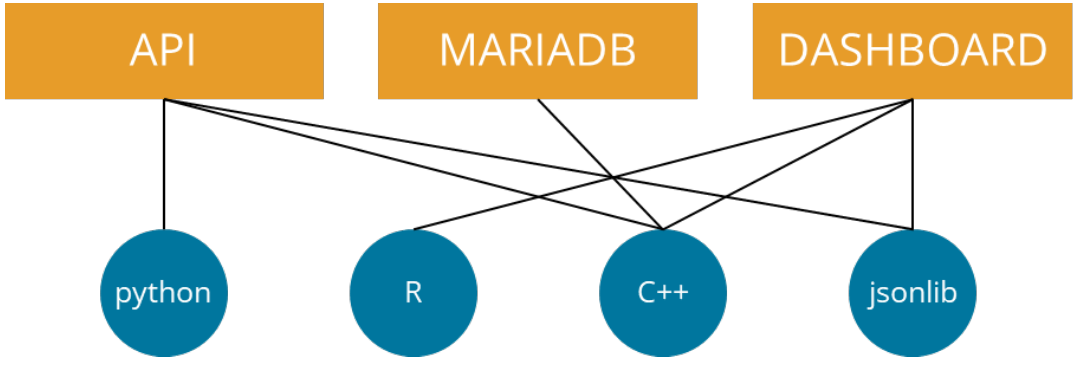
Docker and DevOps

In a nutshell, Docker is a **container engine** that allows developers like yourselves to build production-grade applications in **isolated, stable and easily portable environments**.

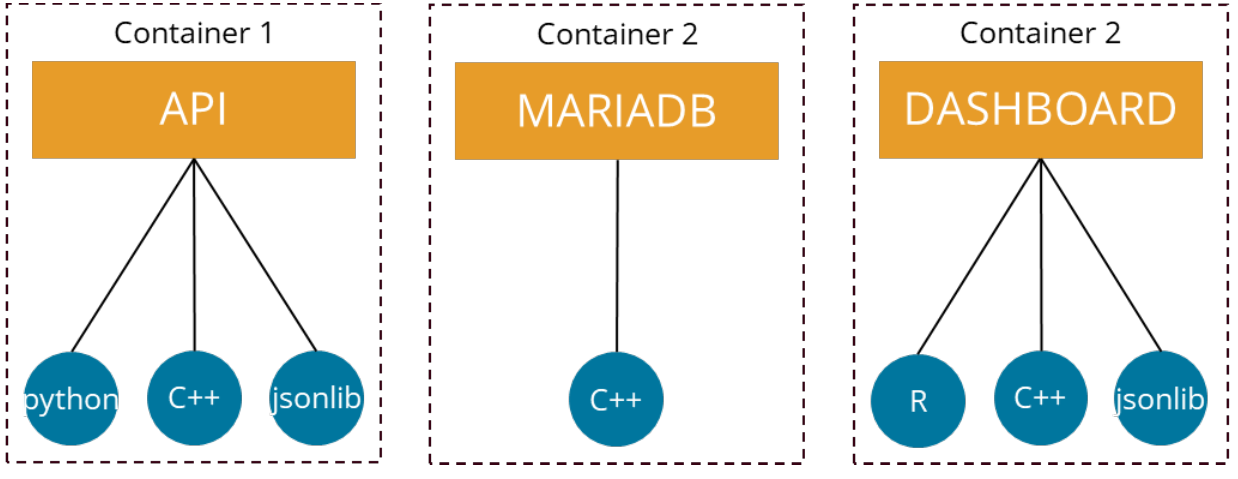
Again, in English: Docker allows you build an application **that will run anywhere*** and **will not interfere with any other program running on your machine** (*anywhere there is Docker, which is everywhere).

Why Docker?

This figure below illustrates the web of dependencies created by running multiple applications natively:



Docker cleans up this web by running each application inside a container:



Why Docker?

Docker was inspired by the adoption of a standard shipping container by the shipping industry. Creating standard dimensions for carrying goods increased the efficiency of freight and the shipping industry took off. For more details see this keynote from the founder of Docker.

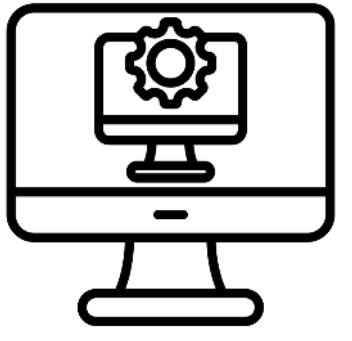
This should by now be ringing bells - ensuring that your services are run in containers means you can easily "ship" them to another service with little to no additional configuration required to get things up and running on the other side.

A quick note on Virtual Machines

Many of you will have heard of **Virtual Machines**, and you might be thinking that Docker is a Virtual Machine. Indeed, they can serve the same function as a Docker container. However there are some key differences that you should be aware of.

Technical difference - JARGON WARNING:

- **A Docker container** is a clever way of *isolating* operating system processes. Although many different containers could be running on a system, at the end all their processes are running on the same operating system and in the same kernel.
- **Virtual Machine** is just what it claims it is. It is a virtual simulation of a physical computer, which means it has its own specially partitioned and ringfenced system resources **in addition to** software.



The key difference is that **Docker containers don't use any hardware virtualization**. This makes them much more efficient that a Virtual Machine.

Quick SideQuest 👑

Solved by adding these two lines in `/etc/resolv.conf`

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

Then

```
systemctl daemon-reload
systemctl restart docker
```

Installing Docker

The online Docker user manuals, called Docker Docs, is incredibly comprehensive repository for all things Docker related, and it should be your first point of reference when you run into trouble!

You can find a guide to installing Docker here. Follow this now!

Once you're done, check the results of `docker --version` to see if it works.

With confirmation that Docker is successfully installed, it's time to spin up a Docker for the first time.

```
docker run hello-world
```

What happens?

The Docker group

You should see that your `docker run hello-world` attempt will fail with this error message:

> `docker: permission denied while trying to connect to the Docker daemon socket`

While this is likely the first time you will have run into this error, it most certainly will not be the last!

Docker controls the ability to interact with images, containers and anything else Docker related via the `docker` group. If you aren't assigned to this group, whatever docker command you run you will be met with the same fate. Use `id` to check what groups have been assigned to you, and then use `sudo usermod -aG docker $USER`

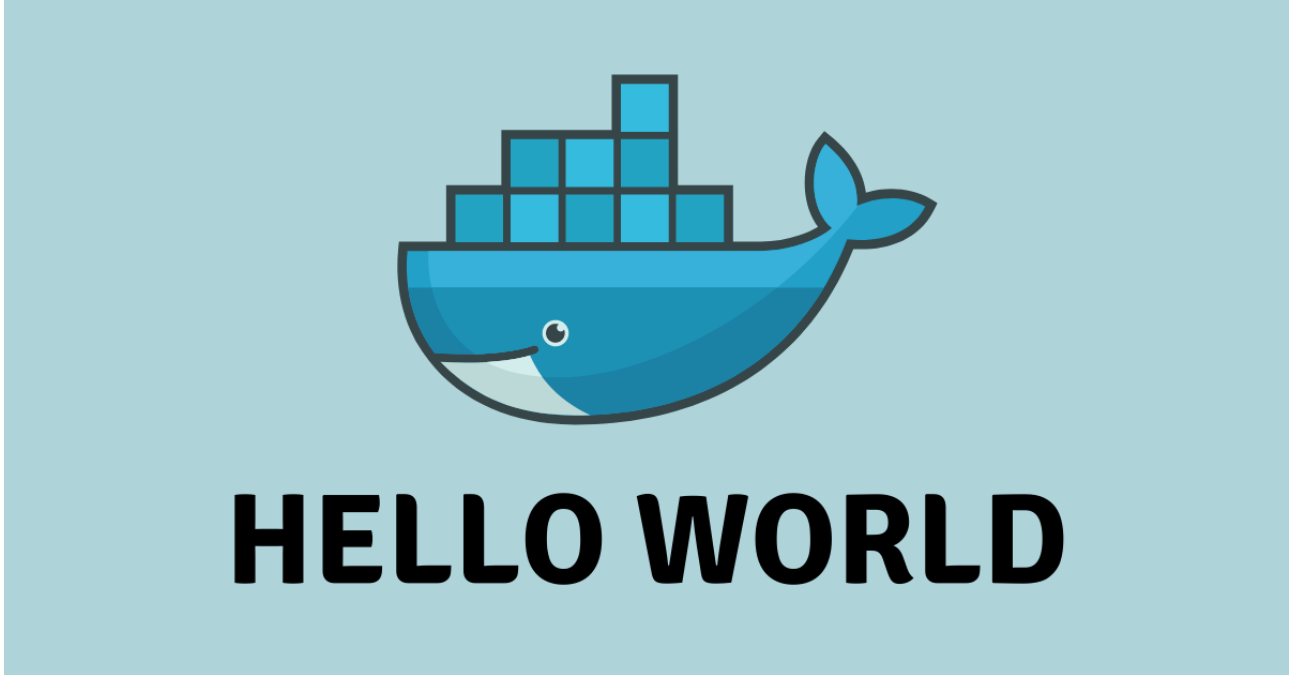
- `usermod` is the command to (surprisingly) **mod**ify a **user**'s characteristics
- `-aG` is the shorthand for **a**dd **G**roup
- `docker` is the name of the group
- `$USER` is the system variable that identifies the current user in the shell

Once you have done this, relog into your machine and check that everything has worked by repeating `id`. If you don't relog you won't see any changes, as your user information is only re-evaluated once a new shell session is started.

First Docker

Now that annoying admin is out of the way, we can actually get to running our first docker. What happens?

```
docker run hello-world
```



Understanding Docker

What has just happened here is not particularly clear, and raises more questions than answers. What makes this any different from a plain old `echo`? What is an image? What is Docker Hub? Let's try and clear this up by getting to grips with some concepts that are core to the way Docker works.

To recap: Docker allows you to create an isolated environment for an application (in this case, think database!). The easiest way to think about this is as a second computer that has completely different packages installed it, that can be upgraded or deleted entirely independently from your machine. But how is this environment defined? What packages are installed in it? What operating system is it running?

All of these questions are defined in a Docker **image**. You can think of this as a list of specifications that outline the **software setup** of a computer. Note that it does not ever mention anything about hardware allocation, since this is not a Virtual Machine!

Now then: a Docker **container** is the actual running isolated environment. Every Docker container is based on a Docker image. You can have multiple Docker containers running from the same image, but you cannot have one Docker container based on more than one image.

Fortunately, you don't have to build an image from scratch. Docker Hub is an online repository for pre-made images free for anyone to use!

PSQL container

A database is probably one of the most important and ubiquitous services supporting any application or server. Databases come in many different flavours, but since we are focusing on analytics here, we are going to be using a **relational database**, which records data in a familiar tabular format, and is very good at maintaining connections between different, but linked, datasets.

In this case, our flavour of DB is going to be **PostgreSQL** (PSQL). PSQL is a shining example of the power of open-source projects and is a common choice for many software developers as a data storage facility.

PSQL container

Fortunately for us, some kind set of strangers on the internet (sounds like a misnomer, doesn't it) has developed a PSQL container that we can just use for free to set up our own PSQL db... How nuts is that?

Let's pull this image. It's a good idea to decide on the latest version there and then and set that as the version of the container image that you are using to prevent any breaking changes in the future (although this is just a slight possibility)

```
docker pull postgres:14.11
```

PSQL container

On the Docker Docs page, they straight away give us the smallest command needed to get this container up and running. Let's adapt it slightly to make this suitable for our specific situation:

```
docker run \
  --name psql \
  -e POSTGRES_PASSWORD='hellopsql' \
  -e POSTGRES_USER='rhea' \
  -e POSTGRES_DB='warehouse' \
  -d postgres:14.11
```

Once that has run, use `docker ps` to check that the container is active. Once you verify that, use `docker exec -it psql bash` to enter the container.

Testing the DB

Once inside the container, you should see that a prompt that looks like this: `root@8141fc89950f`

This is just a slightly different looking terminal. You can do most of the things you are used to, like `ls` and `cd`.

To enter the psql terminal, we need to be a little specific: `psql -U rhea warehouse`

Structuring access for other users

This is NOT a good way to get access to the database. Later, we are going to have to grant other, non-super users access to the database. The point is defeated if we have to give them rights to the `docker` group. What we want is to be able to access the db straight from the command line. How will we do this? By mapping a port **inside** the container to one **outside** the container (i.e. on our actual server).

```
docker stop psql
docker run \
  --name psql \
  -e POSTGRES_PASSWORD='An3JAJk07CCXuXOVY8Ht' \
  -e POSTGRES_USER='rhea' \
  -e POSTGRES_DB='warehouse' \
  -p 3001:5432 \
  -d postgres:14.11
```

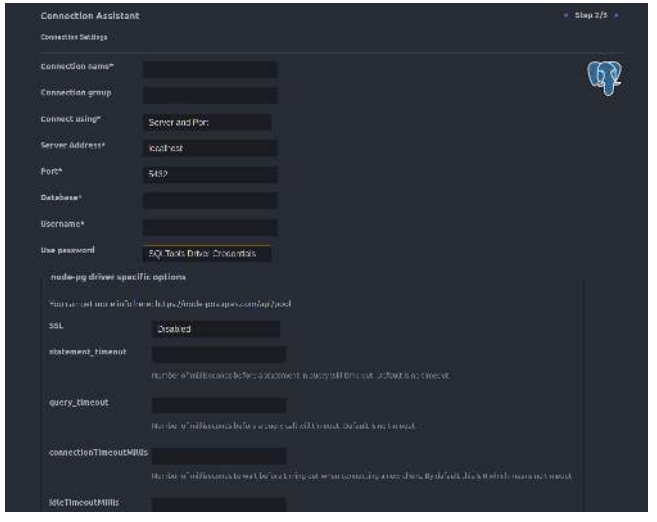
⚠️ Docker overrides iptables! So lets use a secure password!

Structuring access for other users

Now that there is a structured network connection between the container and the server, we can set up access via the terminal:

```
sudo apt install postgresql-client
psql -h localhost -p 3001 -U rhea warehouse
```

I also recommend you access it from VS Code!



71point4
NUMBERS TO LIFE

cenfri

Nginx, SSL and PSQL

Securing in PSQL

You now have a live database! But how do we go about securing our Passwords? Currently the text (aka your password is being sent in plain text) to the server. We need to implement what is called SSL. Secure Sockets Layer (SSL) is a security protocol that provides privacy, authentication, and integrity to Internet communications.

To do this we are going to need two things:

- Nginx `apt -y install nginx`
- Certbot `snap install --classic certbot && ln -s /snap/bin/certbot /usr/bin/certbot`

ANAME and CNAME

The A and CNAME records are the two common ways to map a host name (“name”) to one or more IP addresses. There are important differences between these two records.

The `A` record points a name to a specific IP. If you want `myserver.com` to point to the server 123.45.67.90 you’ll configure:

```
myserver.com.     A        123.45.67.90
```

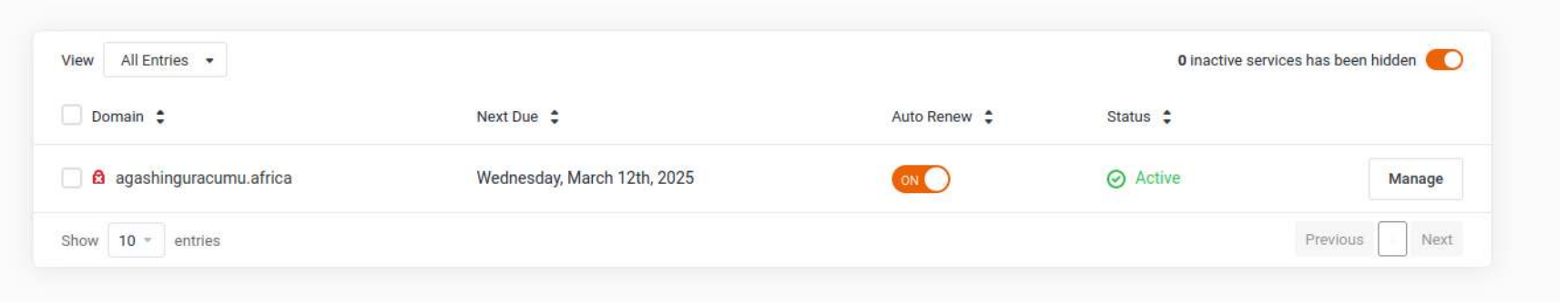
The `CNAME` (Canonical Name) record points a `ANAME` instead of to an IP. The CNAME source represents an alias for the target name and inherits its entire resolution chain.

So think of it as you have an office block with multiple offices:

> ANAME is your address, while CNAME tells you which office on the property to go to...

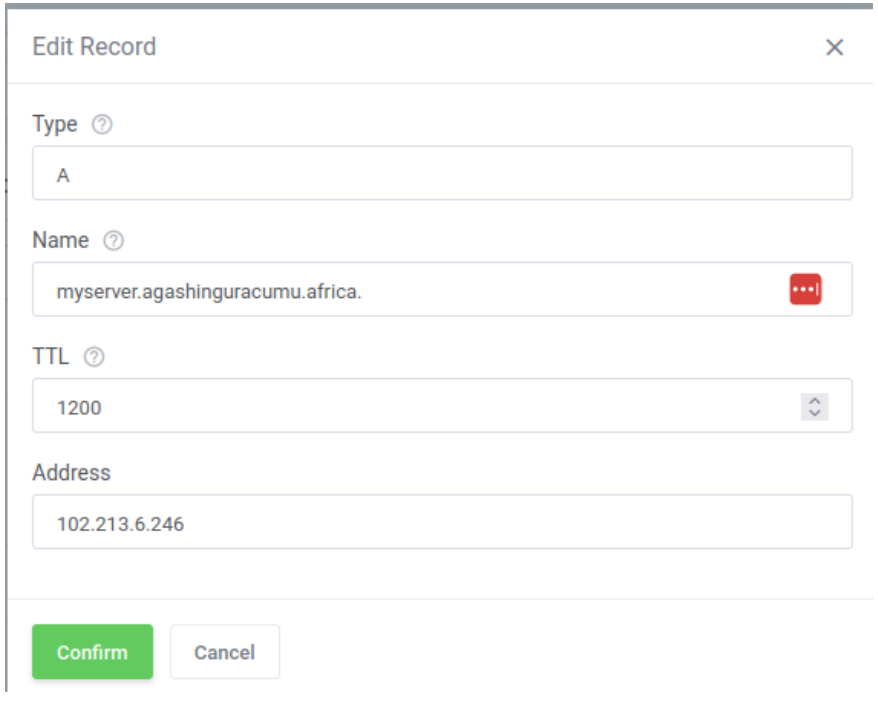
Creating ANAME and CNAME records

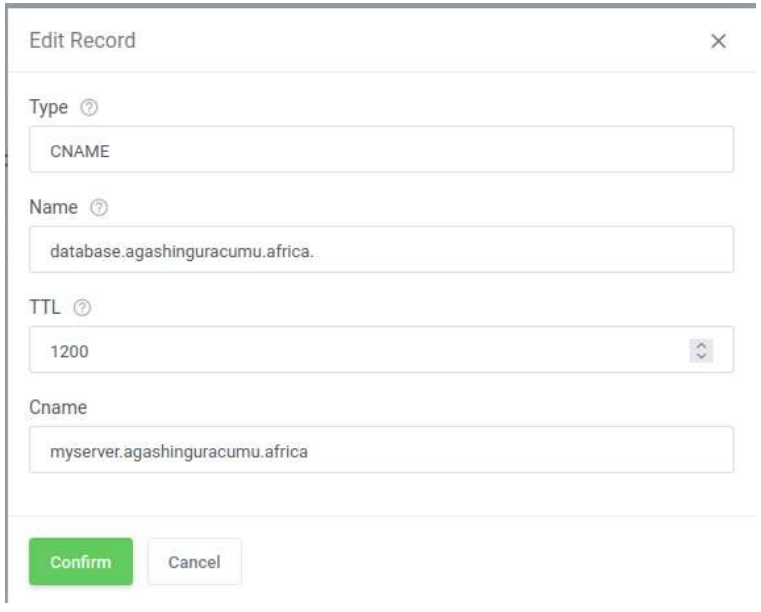
Log onto Rackzar! They are also providing us the vps services. Click on `domains` on the main home screen and go to DNS Manager:



Creating ANAME and CNAME records

Next create a new `ANAME` record pointing to your server's IP. You can find your IP by using the `ifconfig` command. Once you have done that, create the `CNAME`:





Creating ANAME and CNAME records

Time to test!!

Use the command `nslookup` to go around the room asking other people's `ANAME` and testing to see if it points to the correct server:

```
hanjo@cronus:~$ nslookup database.agashinguracumu.africa
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
database.agashinguracumu.africa canonical name = myserver.agashinguracumu.africa
.
Name:   myserver.agashinguracumu.africa
Address: 102.213.6.246
```

Nginx as reverse Proxy

> Imagine there's a cool new bar `busybar.com` at address `192.0.0.1`.

- There are 2 entrances: a VIP entrance called `vip.busybar.com` and `dance.busybar.com`.
- The bartender (Nginx) checks each guest's invitation (request) to see which party they are here for: `vip.busybar.com` or `dance.busybar.com` and guides them to the right room (port).

Besides `Nginx` acting as a bouncer/barman, it can do a million other things we do not cover in this workshop - one of them is loadbalancing!

NGINX®

Configuring Nginx

Make sure you are `root`, then go to `/etc/nginx/sites-available/`:

```
root@VM18:/etc/nginx# ll
total 72
drwxr-xr-x  8 root root 4096 May  8 22:51 ./
drwxr-xr-x 98 root root 4096 May  8 23:26 ../
drwxr-xr-x  2 root root 4096 May 30  2023 conf.d/
-rw-r--r--  1 root root 1125 May 30  2023 fastcgi.conf
-rw-r--r--  1 root root 1055 May 30  2023 fastcgi_params
-rw-r--r--  1 root root 2837 May 30  2023 koi-utf
-rw-r--r--  1 root root 2223 May 30  2023 koi-win
-rw-r--r--  1 root root 3957 May 30  2023 mime.types
drwxr-xr-x  2 root root 4096 May 30  2023 modules-available/
drwxr-xr-x  2 root root 4096 May  8 22:51 modules-enabled/
-rw-r--r--  1 root root 1447 May 30  2023 nginx.conf
-rw-r--r--  1 root root  180 May 30  2023 proxy_params
-rw-r--r--  1 root root  636 May 30  2023 scgi_params
drwxr-xr-x  2 root root 4096 May  8 22:51 sites-available/
drwxr-xr-x  2 root root 4096 May  8 22:51 sites-enabled/
drwxr-xr-x  2 root root 4096 May  8 22:51 snippets/
-rw-r--r--  1 root root  664 May 30  2023 uwsgi_params
-rw-r--r--  1 root root 3071 May 30  2023 win-utf
root@VM18:/etc/nginx# cd sites-available/
```

Configuring Nginx

Next create a file: `touch database.conf` and enter it using vim: `vim database.conf`. We are going to use this file to configure our reverse proxy:

```
server {
    server_name database.agashinguracumu.africa;

    access_log      /var/log/nginx/database_access_log.log;
    error_log       /var/log/nginx/database_error_log.log warn;

    location / {

                    proxy_set_header        Host $host;
                    proxy_set_header        X-Real-IP $remote_addr;
                    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
                    proxy_set_header        X-Forwarded-Proto $scheme;
                    proxy_set_header        Upgrade $http_upgrade;

                    proxy_http_version 1.1;

                    proxy_read_timeout  90s;
                    proxy_pass          http://127.0.0.1:3001/;
            }

}
```

Configuring Nginx

After this, we save the configuration (`:wq`) and create a symbolic link in `sites-enabled`:

```
cd /etc/nginx/sites-enabled
ln -s ../sites-available/database.conf
```

This should look something like this:

```
root@VM18:/etc/nginx/sites-available# cd ../sites-enabled/
root@VM18:/etc/nginx/sites-enabled# ln -s ../sites-available/database.conf
root@VM18:/etc/nginx/sites-enabled# ll
total 8
drwxr-xr-x 2 root root 4096 May  8 23:38 ./
drwxr-xr-x 8 root root 4096 May  8 22:51 ../
lrwxrwxrwx 1 root root   32 May  8 23:38 database.conf -> ../sites-available/database.conf
lrwxrwxrwx 1 root root   34 May  8 22:51 default -> /etc/nginx/sites-available/default
root@VM18:/etc/nginx/sites-enabled#
```

If so... then you are almost there...

Configuring iptables for Nginx to work

You are going to need to configure your firewall to allow port `80` and `443` for Nginx to work properly:

```
iptables -I INPUT 5 -p tcp -s 0.0.0.0/0 --dport 80 -m comment --comment "Nginx HTTP" -j ACCEPT
iptables -I INPUT 6 -p tcp -s 0.0.0.0/0 --dport 443 -m comment --comment "Nginx HTTPS" -j ACCEPT
```

Unleash SSL!!

Unleash SSL!!

We can now use `certbot` to create secure certificates: `certbot --nginx --register-unsafely-without-email`

```
root@VM18:/etc/nginx/sites-enabled# certbot --nginx --register-unsafely-without-email
Saving debug log to /var/log/letsencrypt/letsencrypt.log

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.4-April-3-2024.pdf. You must agree in
order to register with the ACME server. Do you agree?
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: Y
Account registered.

Which names would you like to activate HTTPS for?
We recommend selecting either all domains, or all domains in a VirtualHost/server block.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1: database.agashinguracumu.africa
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel): 1
Requesting a certificate for database.agashinguracumu.africa

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/database.agashinguracumu.africa/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/database.agashinguracumu.africa/privkey.pem
This certificate expires on 2024-08-06.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

Deploying certificate
Successfully deployed certificate for database.agashinguracumu.africa to /etc/nginx/sites-enabled/database.conf
Congratulations! You have successfully enabled HTTPS on https://database.agashinguracumu.africa

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
If you like Certbot, please consider supporting our work by:
 * Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
 * Donating to EFF:                    https://eff.org/donate-le
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
root@VM18:/etc/nginx/sites-enabled#
```

Unleash SSL!!

Last step is to ensure your traffic is directed through `Nginx` and not bypassing your firewall:

```
docker stop psql
docker run \
  --name psql \
  -e POSTGRES_PASSWORD='An3JAJk07CCXuXOVY8Ht' \
  -e POSTGRES_USER='rhea' \
  -e POSTGRES_DB='warehouse' \
  -p 127.0.0.1:3001:5432 \
  -d postgres:14.11
```

Congratulations, you now have setup a `SaaS`!