Introduction to DevOps

Key Takeaways

Different IT Roles in Software Development & Delivery Process

Software Development

- software **programmed by developers**
- in different programming languages, like Java, Python, JavaScript
- new functionality & bugfixes

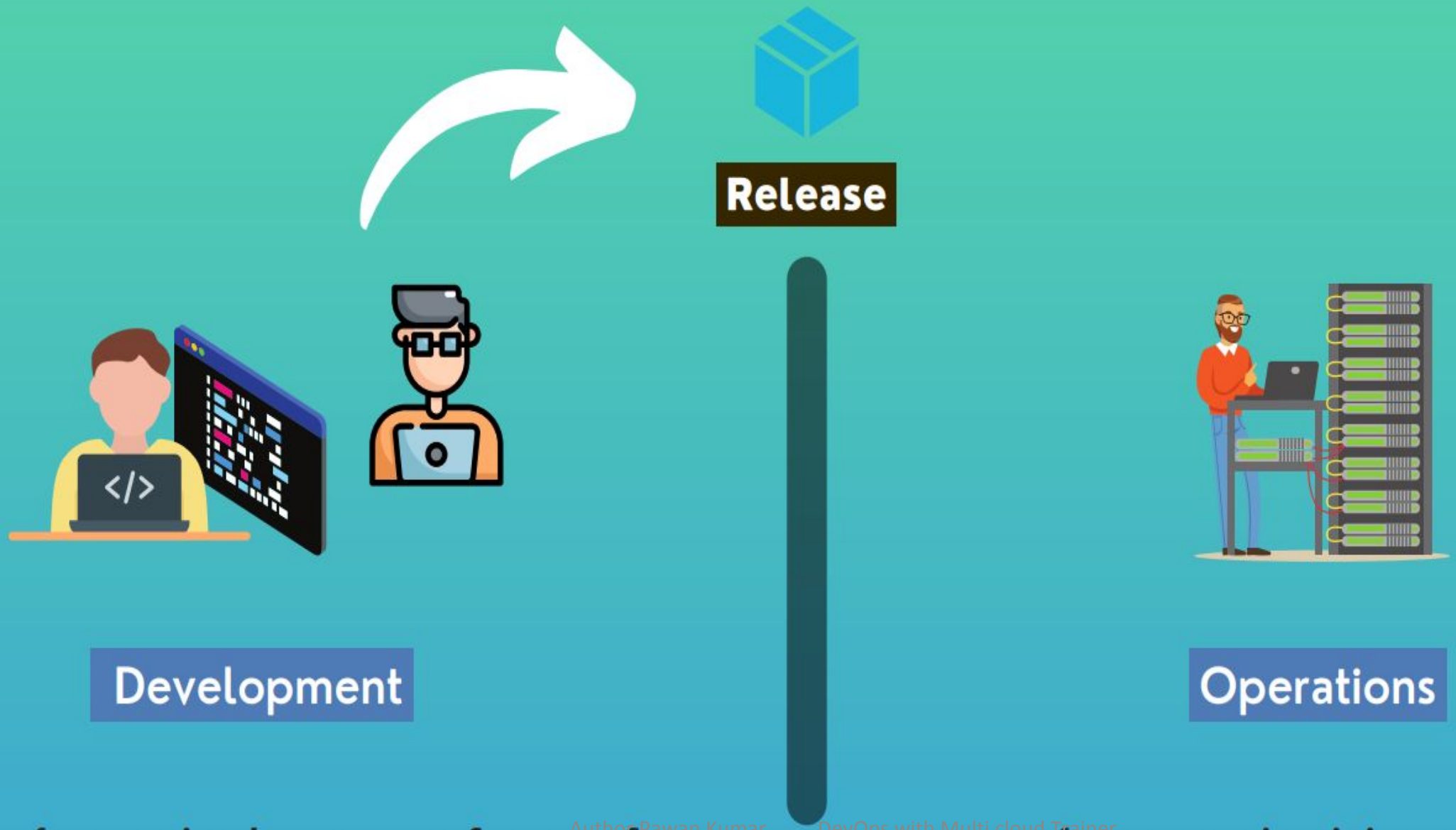
Software Testing

- **test** new features & old functionality
- done by developers & dedicated testers
- manual & automated testing

Operations

- **build** application
- **deploy** on servers
- **upgrade** existing software
- run software in production
- done by operations team

Traditional: Development vs Operations - 1



- focus on **implement new features fast**
- focus on **maintaining stability**

Traditional: Development vs Operations - 2

Different Responsibilities

Different Technical Knowledge

Different Toolsets

Development

- programming languages
- test frameworks
- databases
- version control

Operations

- OS, mostly Linux
- command-line
- scripting
- monitoring tools

Traditional: Development vs Operations - 3

Miscommunication and Lack of Collaboration

- Deployment requires configurations & environment needs to be prepared, so communication is important
- In reality **silos between those 2 departments**

Conflict of Interest

- **DEV Focus** on releasing new features fast
- **OPS Focus** on maintaining systems' stability

Manual Work & Checklists

- **Manual checks**: does new change affect systems' stability or security
- **Manual deployment**
- **Manual configurations** of deployment environment

Slows down the release process!!

Solution: DevOps Culture - 1

Simplest Definition:

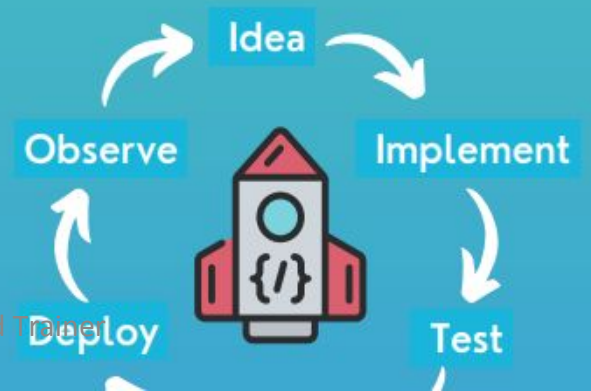
Intersection of Development & Operations

- DevOps was just a way of working between **DEV**'s and **OP**'s
- Common language to communicate

Better Definition:

DevOps is about making the process of continuous delivery fast and with minimal errors

- DevOps tries to **remove all these roadblocks** and things that slow down the release process
- Instead of manual inefficient processes helps create fully **streamlined processes for release cycles**

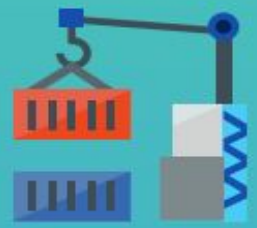


Solution: DevOps Culture - 2

- Different companies implemented DevOps in different ways
- But, gradually it got a more concrete form with common patterns

Source Code Management

Continuous Integration/Deployment



Container Orchestration

Planning & Collaboration

Package Management

Treat Infrastructure as Code

Cloud

DevOps Engineer

DevOps as a separate Role

- DevOps evolved into an **actual role**: "DevOps Engineer"
- Where either Developers, Operations or a separate role is responsible for DevOps

Developers doing DevOps

Doing only DevOps

Operations doing DevOps

- **DevOps tools**: Set of technologies used to implement the DevOps Principles became DevOps technologies
- DevOps Engineer responsible for creating a streamlined fully automated release process

DevOps Tasks and Responsibilities - 1

Need **some know-how from DEV and OPS team**

Additional DevOps specific skills and know-how

DevOps Tasks and Responsibilities - 2

- DevOps tools that are known today:
- Replacing human tasks with automation tools

Source Code Management

Git

Package Management

Docker

Infrastructure as Code

Terraform, Ansible

Continuous Integration/Delivery

Jenkins, Gitlab

Container Orchestration

Kubernetes

Cloud

AWS, Azure, Google Cloud

Continuous Monitoring

Prometheus

- At the core of DevOps: **CI/CD Pipeline for an automated release process**

Commit Code → TEST → BUILD → PUSH → DEPLOY

High Level Overview & Big Picture of DevOps

Throughout Bootcamp: Zoom in into each part and understand it in detail and be able to implement the whole DevOps process

Operating Systems & Linux Basics

Key Takeaways

Author Pawan Kumar DevOps with Multi cloud Trainer

Introduction to Operating Systems

Author Pawan Kumar DevOps with Multi cloud Trainer

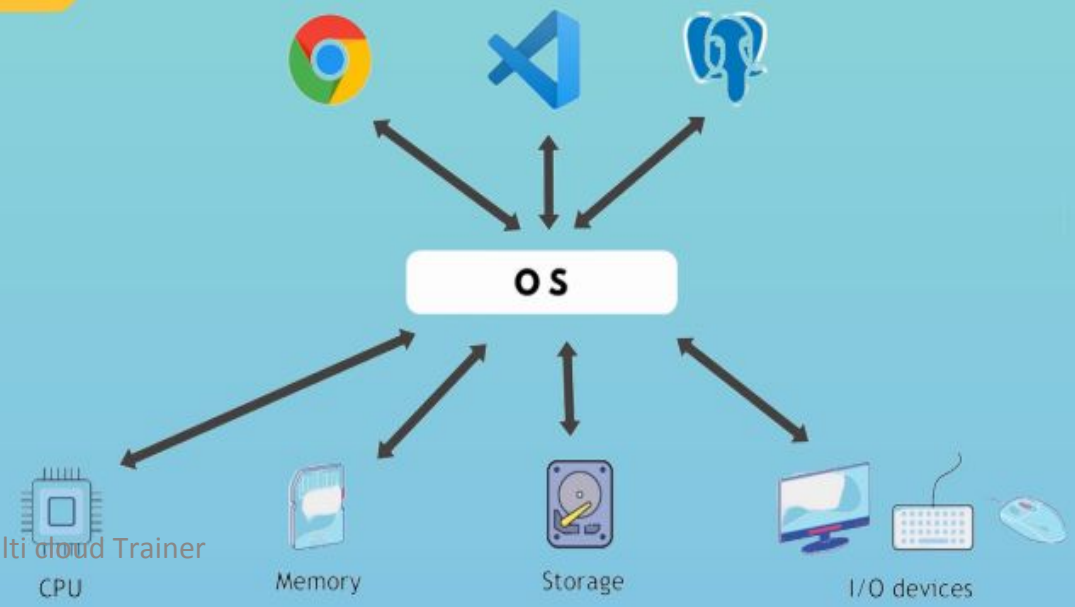
What is an Operating System?

OS is a software managing...

- computer hardware,
- software resources
- and provides common services for computer programs

OS as abstraction layer between applications and hardware

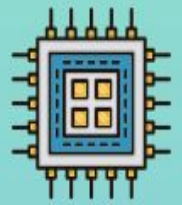
- Instead of applications (like browser) interacting with the computer hardware directly, they **can use the OS as abstraction layer between the two**
- Messy, if every app had to talk directly to the hardware parts
- Apps, like browser can't be installed directly on the hardware



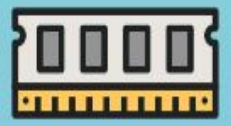
Tasks of an Operating System - 1

1) Resource Allocation and Management

OS manages resources among applications:



- **Process Management (CPU):**
  - 1 CPU can only process 1 task at a time
  - OS **decides which process gets the processor** (CPU), when and for how much time and allocates the processor to the process
  - CPU is switching so fast that you don't notice it



- **Memory Management (RAM):**
  - Allocating working memory (RAM = Rapid Access Memory) to applications

- **Storage Management (Hard Drive):**
  - Also called "secondary memory" - the hard drive
  - **Persisting data long-term**, like files, browser configurations, games, pictures, videos etc.

Tasks of an Operating System - 2

2) File Management

- Files are stored in **structured** way
- A file system is organized into directories

Folders

- On Unix system: **tree file system**
- On Windows OS: **multiple root folders**

3) Device Management

- **Manages device communication** via their respective drivers
- Activities like:
  - Keeping track of all devices
  - Deciding which process gets the device, when and how long
  - ...

OS

Tasks of an Operating System - 3

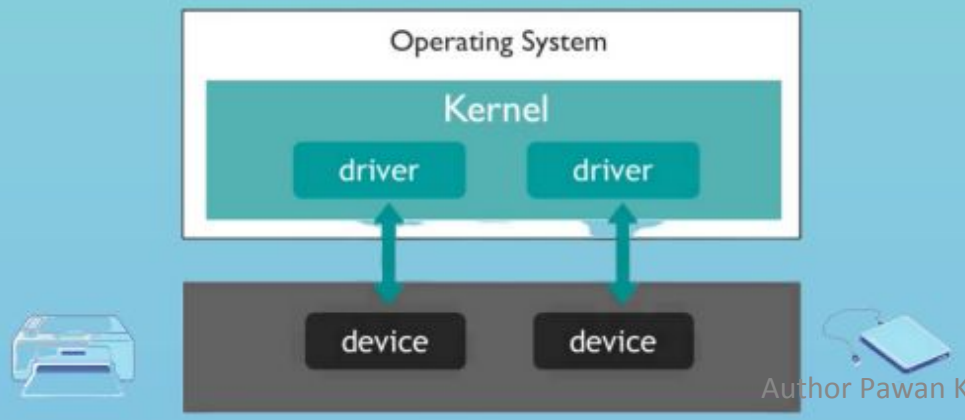
Other important tasks

- **Security:**
  - Managing Users and Permissions
  - Each user has its own space and permissions

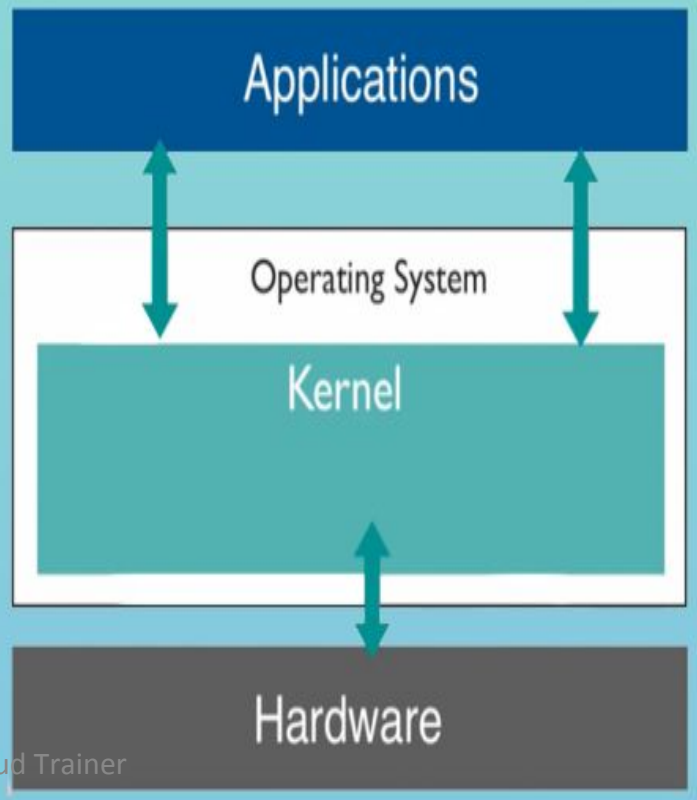
- **Networking**
  - Ports and IP addresses
  - Transmitting outgoing data from all application ports onto the network, and forwarding arriving network packets to processes

Kernel

- **Heart** of every OS
- The core program that provides basic services for all other parts of the OS
- Consists of device drivers, dispatcher, scheduler, file system etc.
- One of the first programs loaded on startup
- **Controls all hardware resources via device drivers**



- Kernel starts the process for app
- Allocates resources to app
- Cleans up the resources when app shuts down

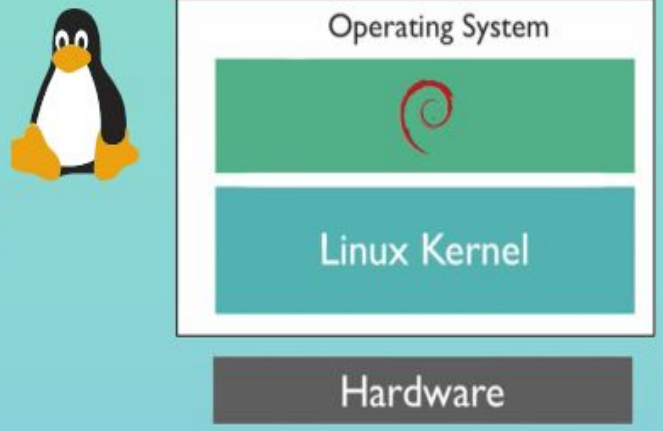


Operating System Components - 2

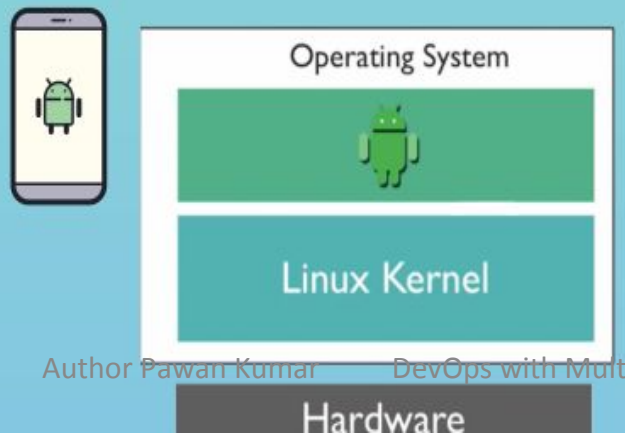
Application Layer

- **On top of Kernel is the application layer**

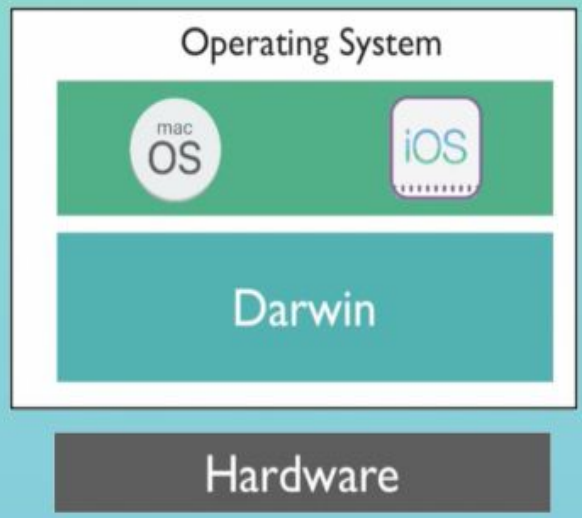
- For example different Linux distributions:
  - Ubuntu, Mint, CentOS, Debian



- Different application layers, but based on same Linux kernel
- Android is also based on Linux kernel
- Linux Kernel most widely used



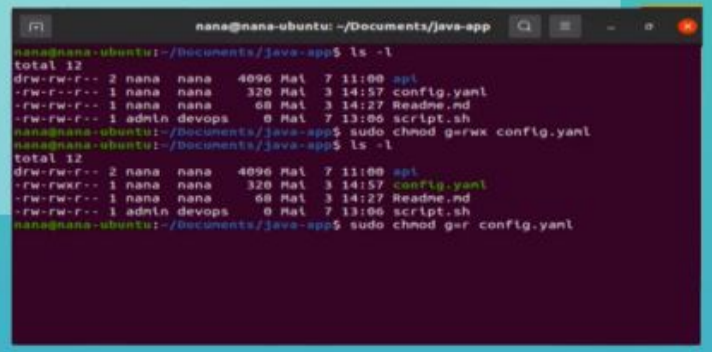
- MacOS and iOS is based on a different Kernel



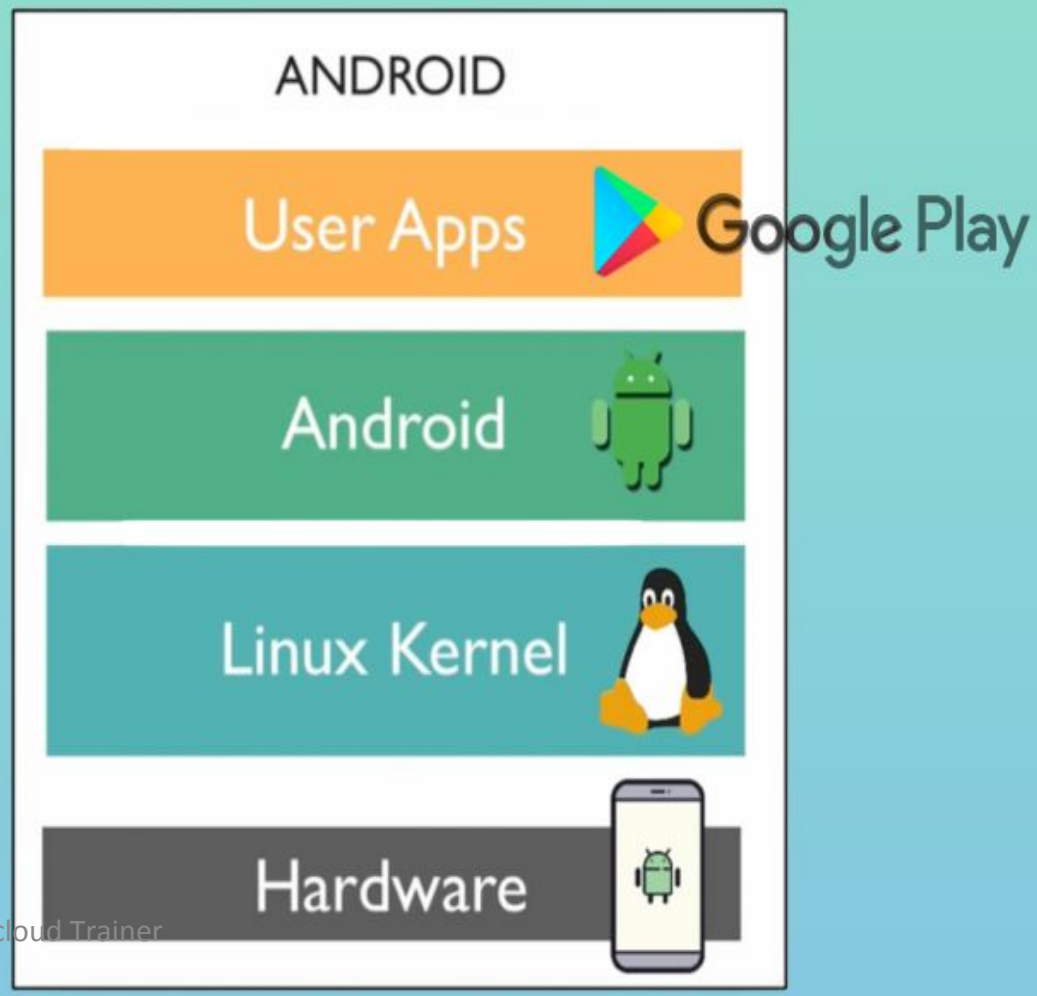
Operating System Components - 3

How to **interact** with Kernel

- Graphical User Interface
- Command Line Interface



Example: Layers of Android phone



3 Main Operating Systems

Linux

Windows

MacOS

- Each OS has many versions
- But kernel stays the same!

Client OS vs Server OS

Client OS

- For personal computers with GUI and I/OS devices

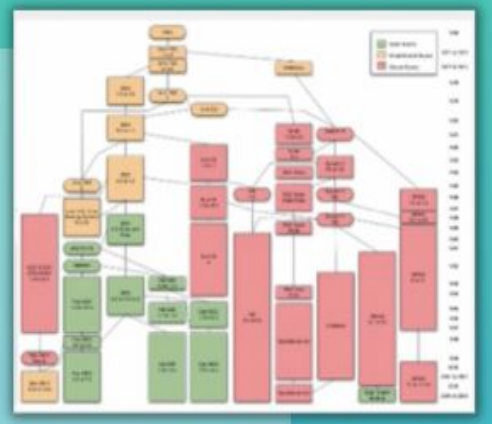
Server OS

- Linux and Windows have server distributions
- But Linux most widely used
- More **light-weight** and performant
- No GUI or other user applications

MacOS vs Linux

- MacOS and Linux: Command Line, File structure etc. **similar**
- Whereas Windows is completely different

UNIX 1970



- Codebase for many different OS
- Developed independent of Linux
- Many OS built on top of Unix
- MacOS Kernel is based on Unix
- To keep them compatible, standards were introduced
- POSIX (Portable Operating System Interface) is the most popular standard

Linux 1991

- Linux was **developed in parallel** to Unix based operating systems
- No source code of UNIX
- Created by Linus Torvalds
- Clone of UNIX or also called "**unix like**"
- Linux and MacOS **both POSIX compliant**

Why learn Linux as a DevOps Engineer

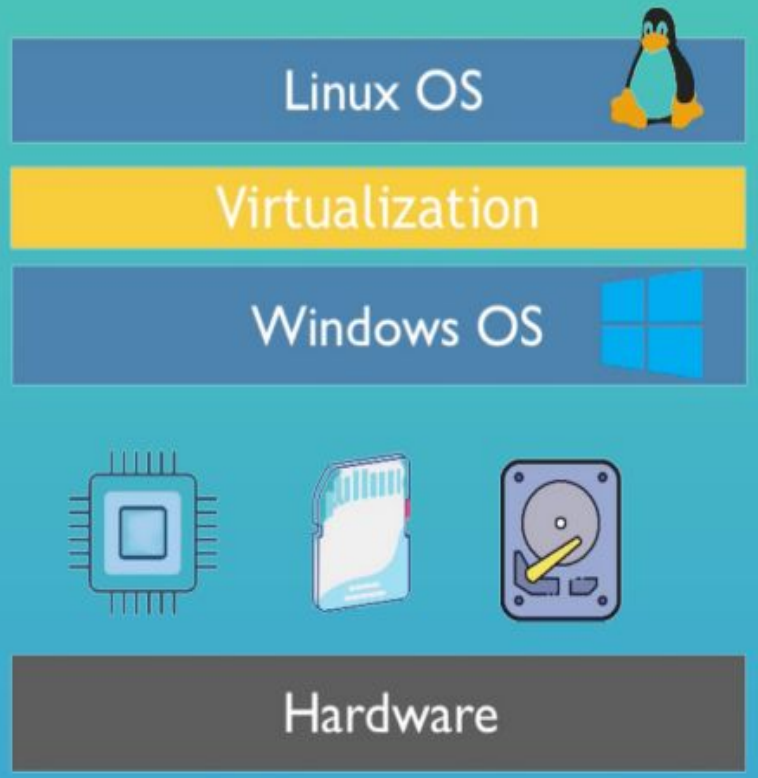
- Linux is mostly used OS for servers!
- Knowing Linux is a must for DevOps Engineers
  - you need to work with servers
  - installing and configuring servers
  - Linux native technologies

Introduction to Virtualization & Virtual Machines

Author Pawan Kumar DevOps with Multi cloud Trainer

What is Virtualization and a Virtual Machine - 1

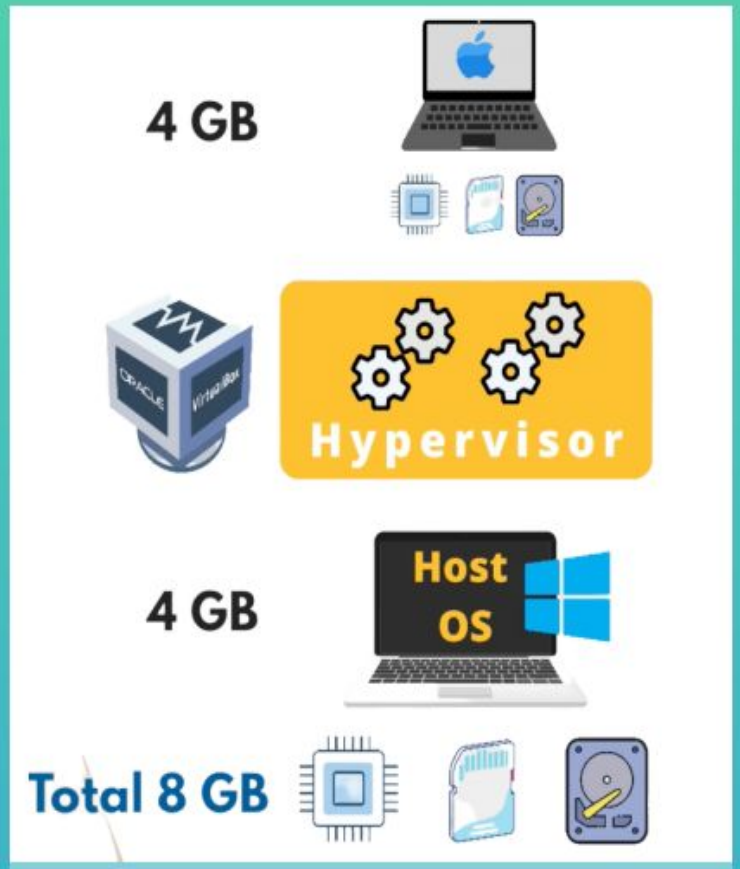
- Virtual Machine is commonly shortened to just "VM"
- VM is a virtual computer running on top of another host computer



How it works - Virtualization

- Virtualization is the process of creating a software-based, or **"virtual" version of a computer**, with dedicated amounts of CPU, memory, and storage that are "borrowed" from a physical host computer
- Makes it possible that **any OS can run on top of any other physical host machine**
- The VM is partitioned from the rest of the system, meaning it's **completely isolated** and can't interfere with the host computer's primary OS

What is Virtualization and a Virtual Machine - 2



- The **hardware resources are shared**
- So you can only give resources you actually have

Hypervisor

- The essential component in the virtualization stack is a piece of software called a hypervisor
- One of the most popular hypervisor is open-source **Oracle VM VirtualBox**

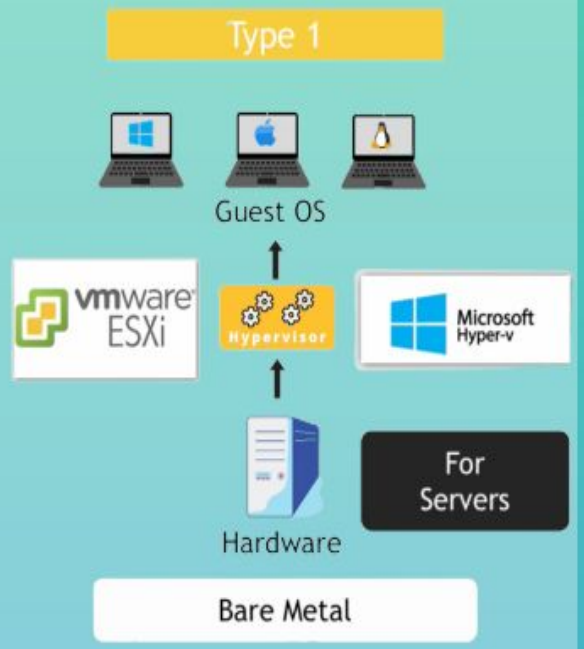
Host OS vs Guest OS

- **Host OS** = runs directly on the hardware
- **Guest OS** = runs on the virtual machine

Hypervisor Types - 1

Type 1 - Native or Bare Metal

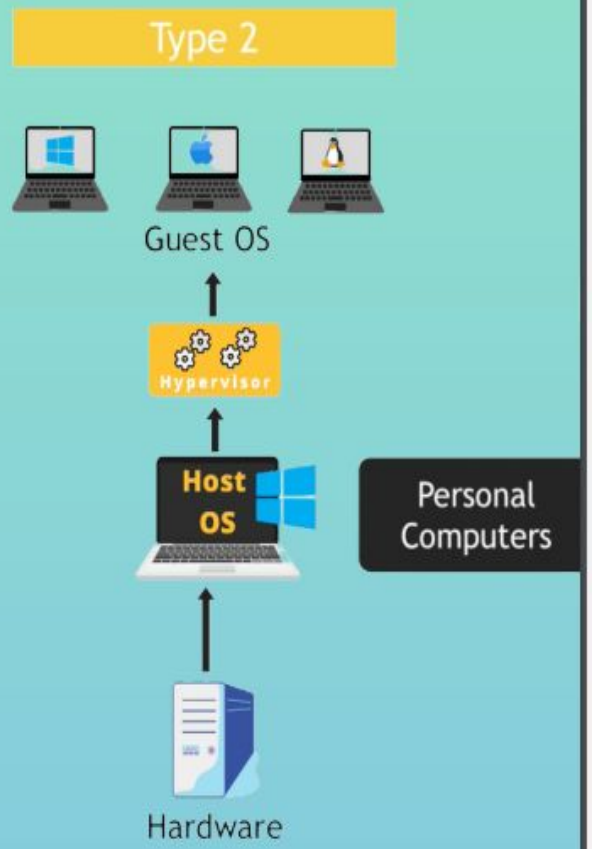
- That **run directly on the host's hardware** to control the hardware, and to monitor the guest OS. The guest OS runs on a separate level above the hypervisor



- Examples of this classic implementation of VM architecture are Oracle VM, Microsoft Hyper-V, VMware ESXi.

Type 2 - Hosted

- Designed to **run within a traditional OS**
- A hosted hypervisor adds a distinct software layer on top of the host OS, and the guest OS becomes a third software level above the hardware



- Examples: Oracle VM VirtualBox, VMware Server, Microsoft Virtual PC, KVM, QEMU and Parallels.

Hypervisor Types - 2

Type 1 - Use Cases

- **Efficient usage of hardware resources (e.g. cloud provider):**
  - Use all the resources of a performant big server
  - Users can choose any resource combinations

Type 2 - Use Cases

- **Learn and experiment**:
  - You don't need to buy a new computer for that
  - You don't endanger your main OS
- **Test your app on different OS**
- **Backing up your existing OS**

Why companies adopt Virtualization

Main benefit: Instead of OS being tightly coupled to the hardware, Virtualization gives an **abstraction** layer, with the following benefits:

- **Security**: Secure very easily
- **Agility and speed:** Spinning up a VM is easy and quick, compared to setting up an entire new server
- **Cost savings**: Efficient usage of hardware resources
- **Portable**: OS as a portable file (VMI - Virtual Machine Image)

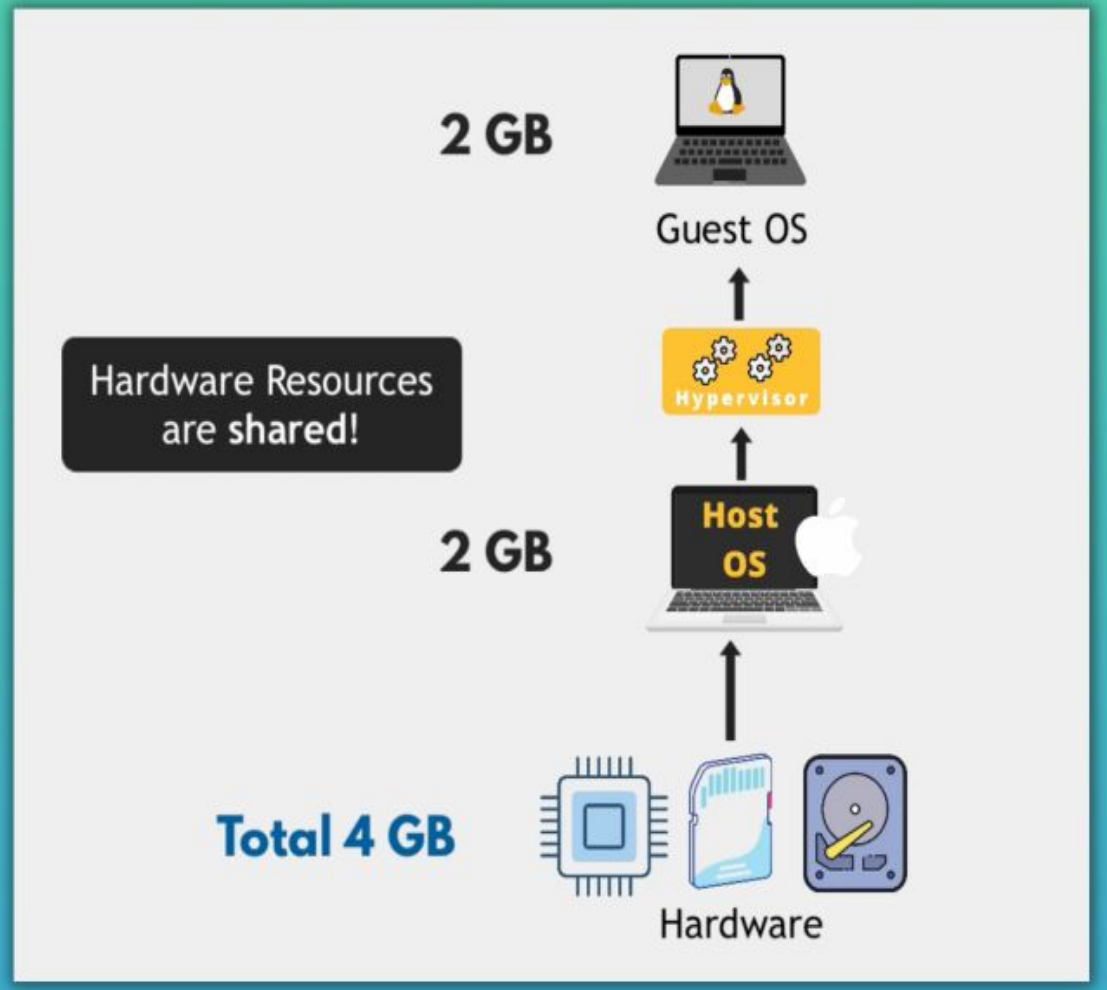
VMI:

- Includes OS and all applications on it
- You can have backups of your entire OS

Setup Linux VM

1) Install VirtualBox Hypervisor

2) Setup Linux Ubuntu Virtual Machine

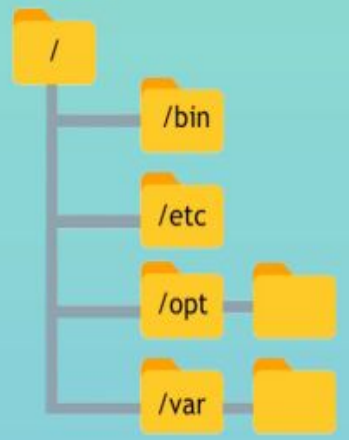


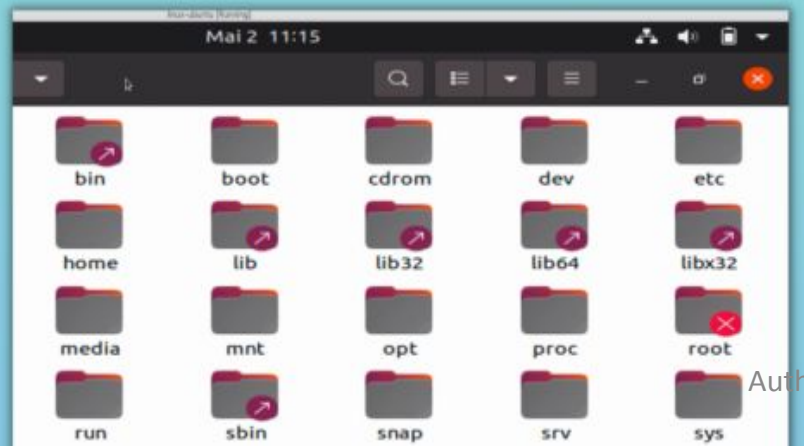
Keep in mind: shard resources

Linux File System

Linux File System - 1

- File system is used to handle the data management of the storage
- The Linux file system has a **hierarchical tree structure**
- With **1 root folder**





Windows in comparison

- Has multiple root folders
- C = internal hard drive

Removable Disks | Local Disk | Data

Linux File System - 2

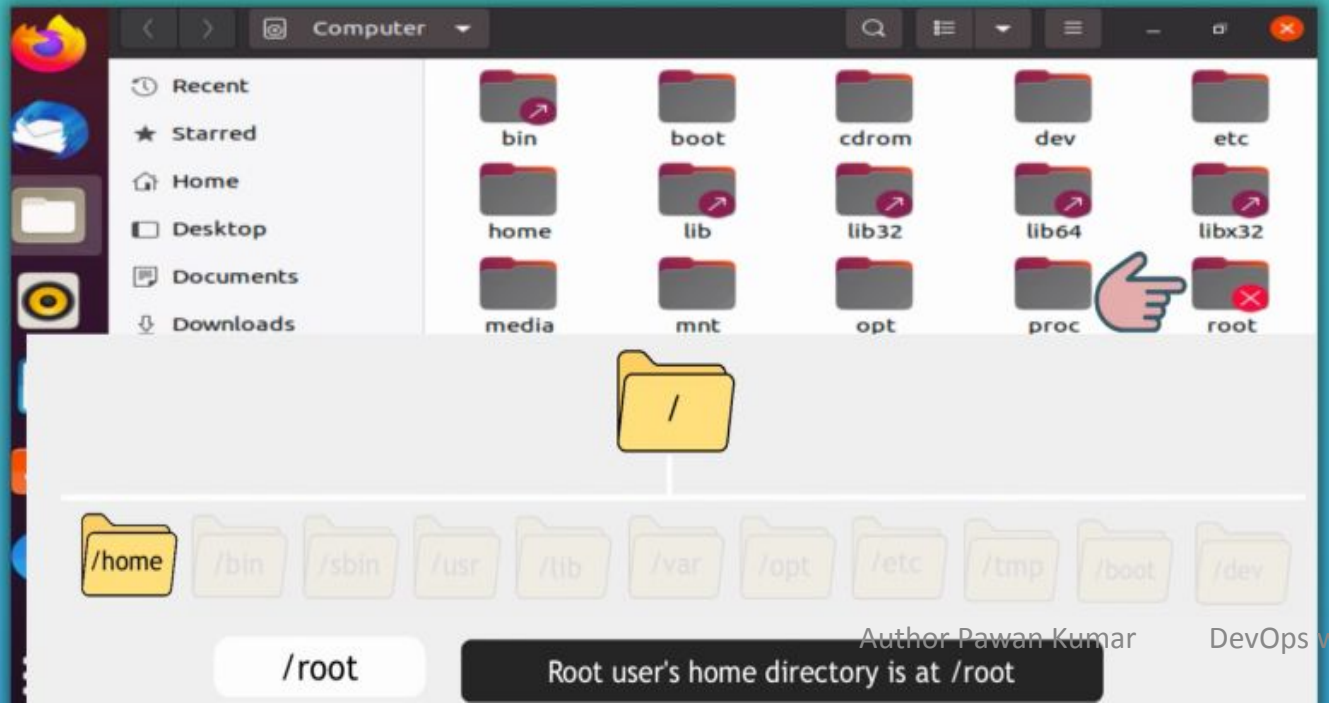
EVERYTHING in Linux is a FILE

- **Everything in the system is represented by a file descriptor**
- Text documents, pictures etc
- Commands, like pwd, ls etc
- Devices like printer, keyboard, usb
- Even directories. Linux makes no difference between a file and a directory, since a directory is just a file containing names of other files

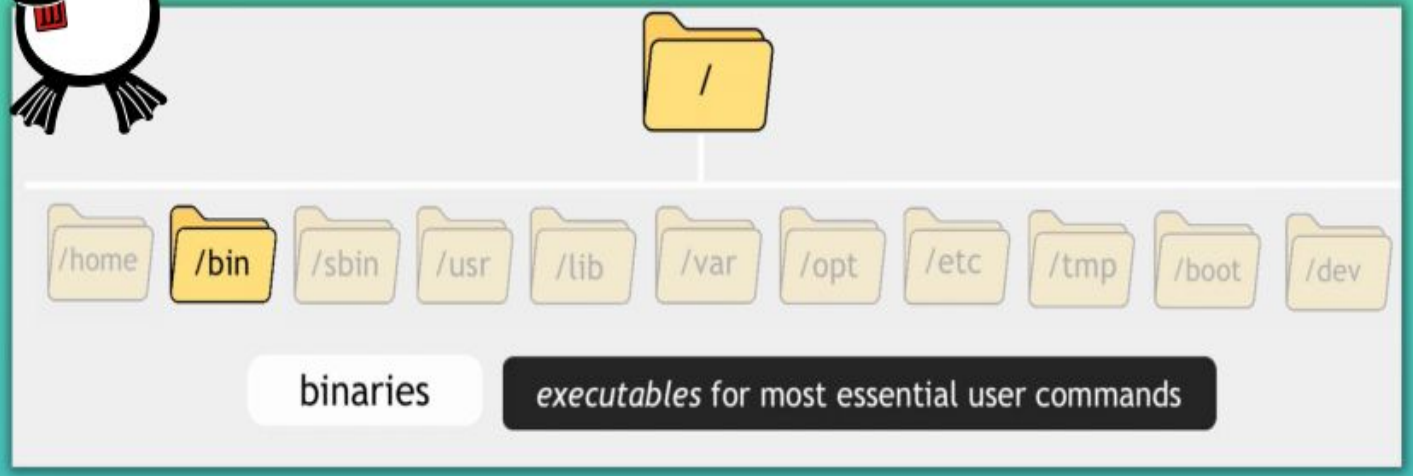
Linux File Structure - 1

Users on OS

- Possible to have multiple user accounts on 1 computer
- **Each user has its own space**
- Each user can have own configurations

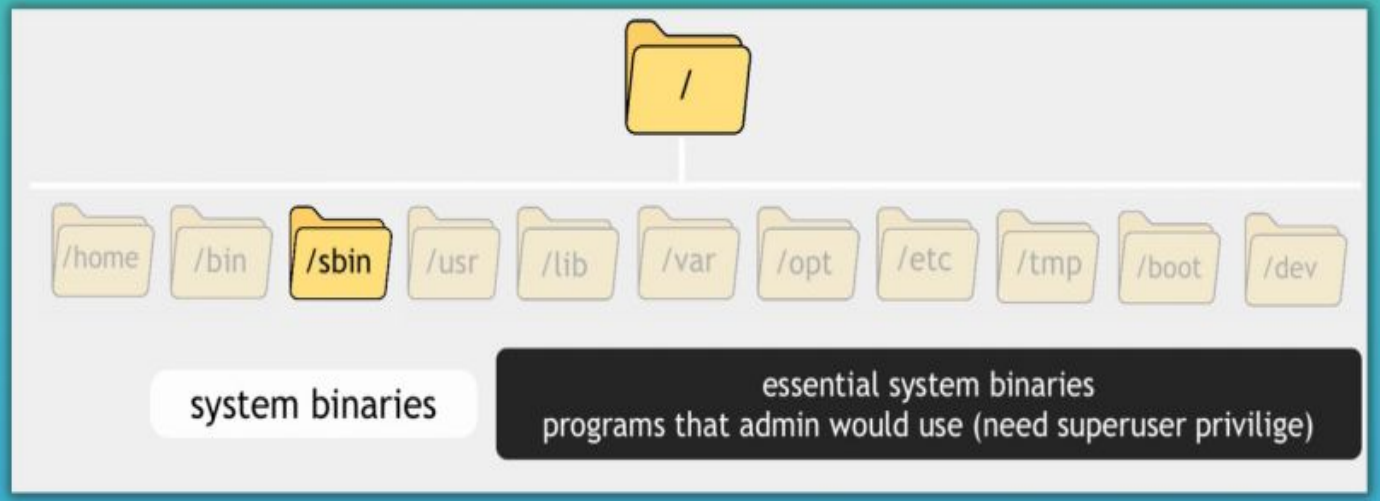


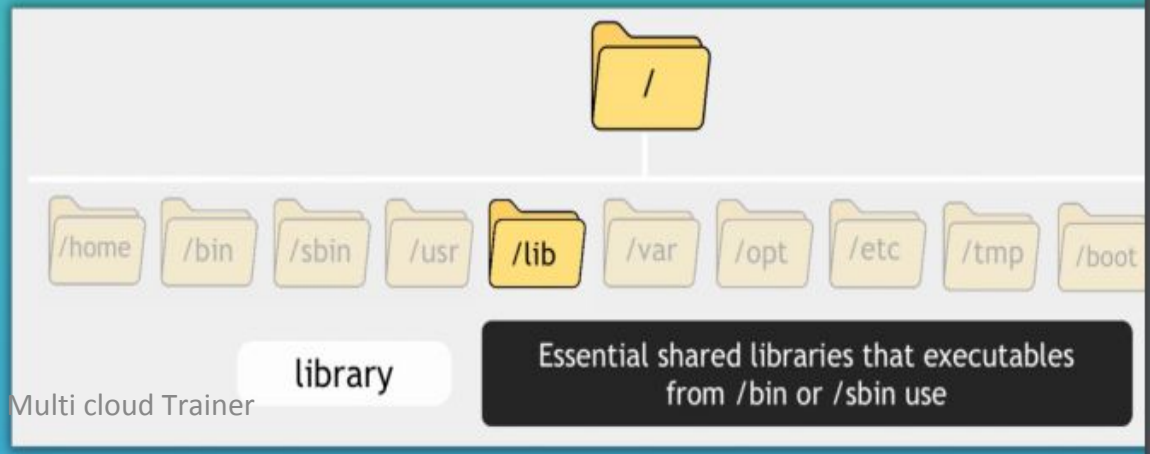
Linux File Structure - 2



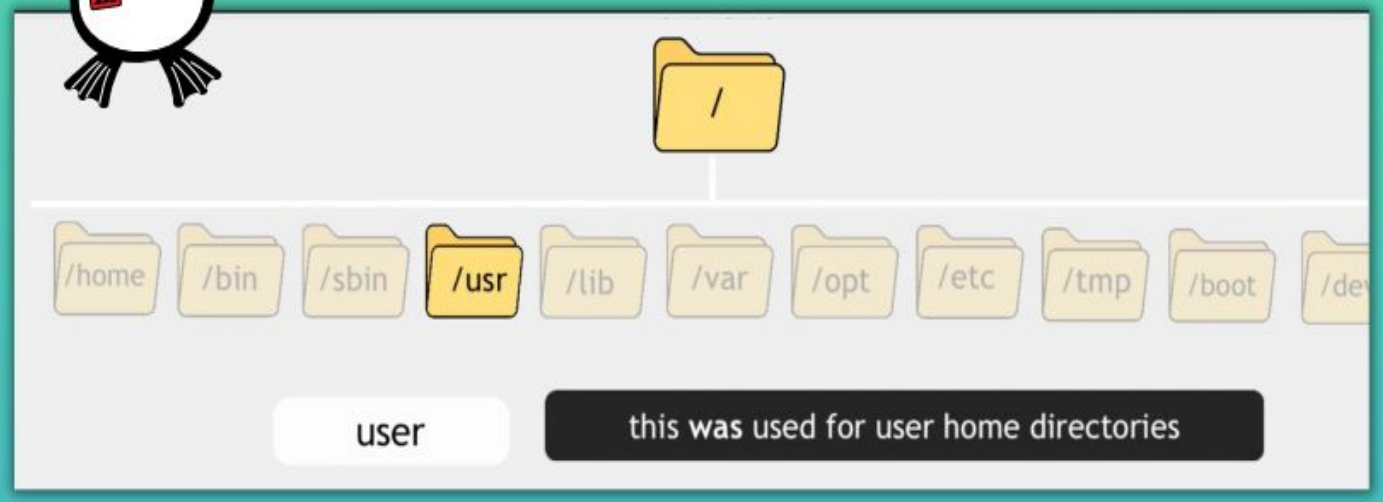
Binary

- A binary is a computer-readable format





Linux File Structure - 3



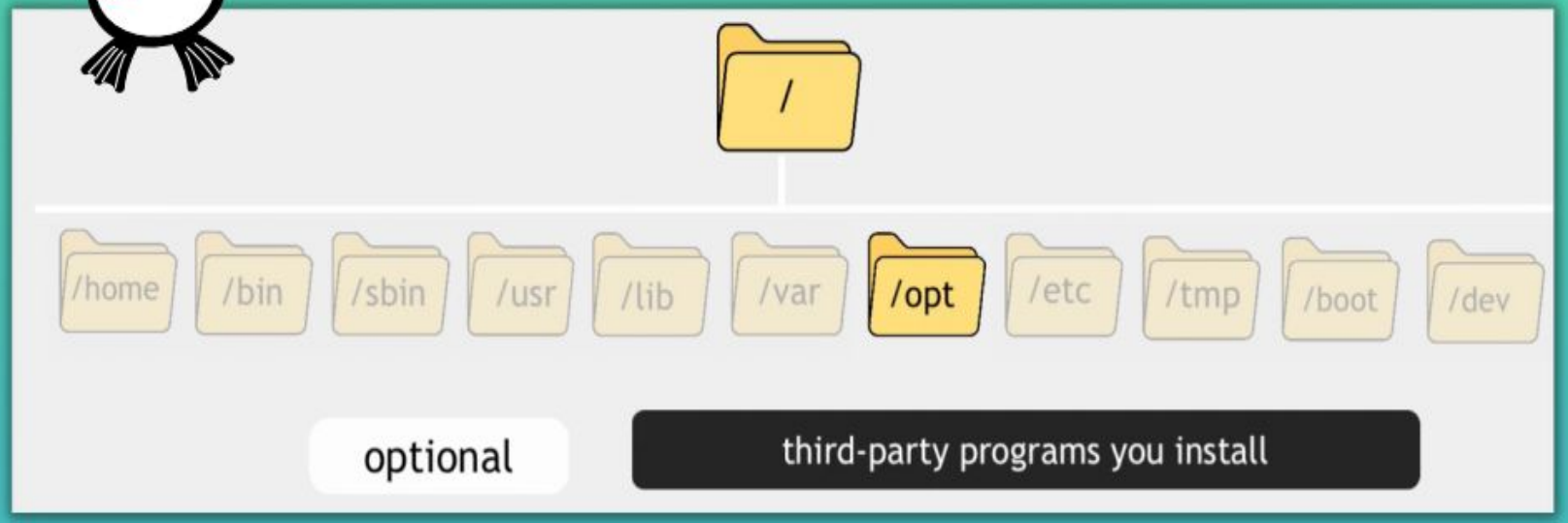
Why?

- Historic reasons
- Because of storage limitations it was split to root binary folders and user binary folders

/usr/local

- **Programs that YOU install** on the computer
- Third-party applications like docker, minikube, java, ...
- Programs installed here, will be **available for all users on the computer**
- Inside /usr/local: App installation will be split again into different folders..

Linux File Structure - 4



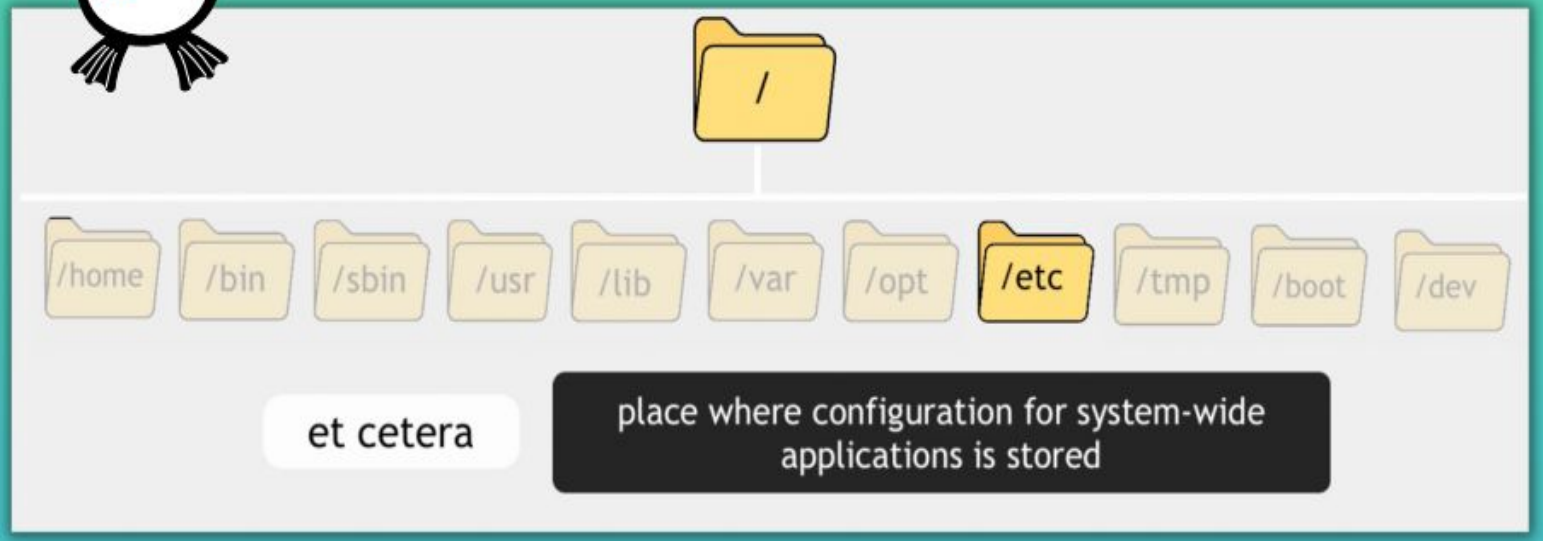
/usr/local

- Programs, which split its components

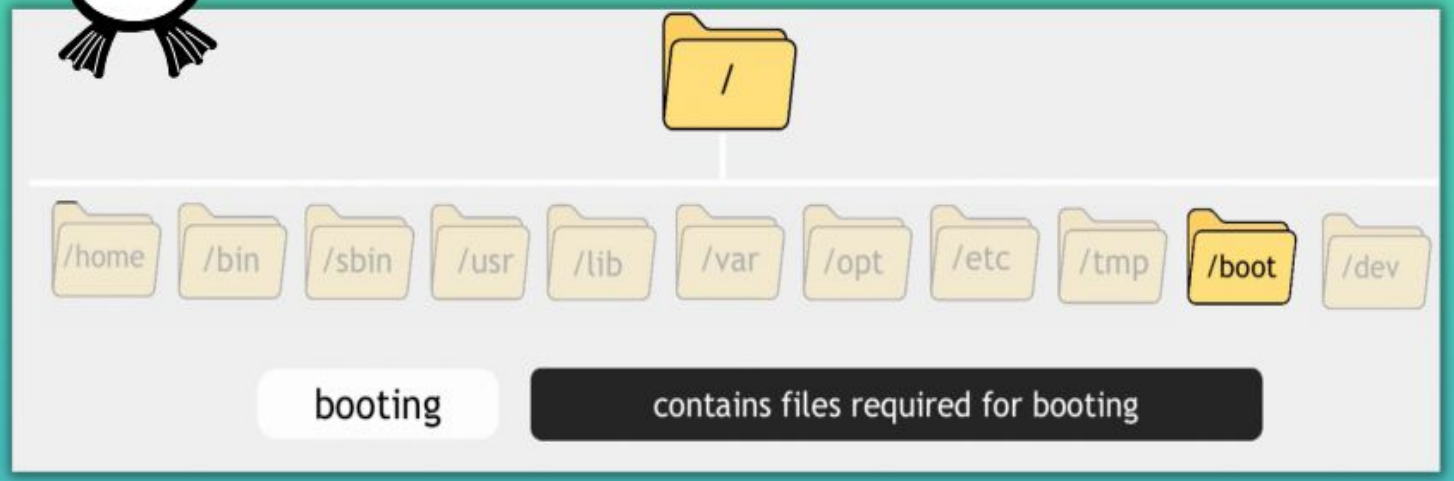
/opt

- Programs, which **NOT** split its components

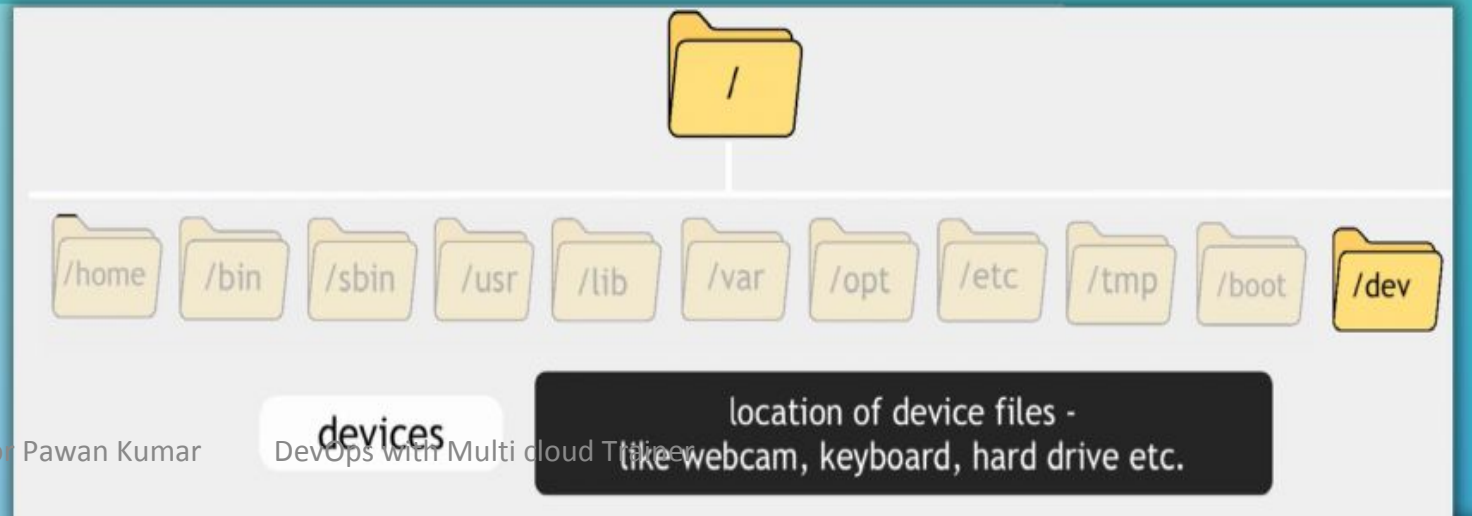
Linux File Structure - 5



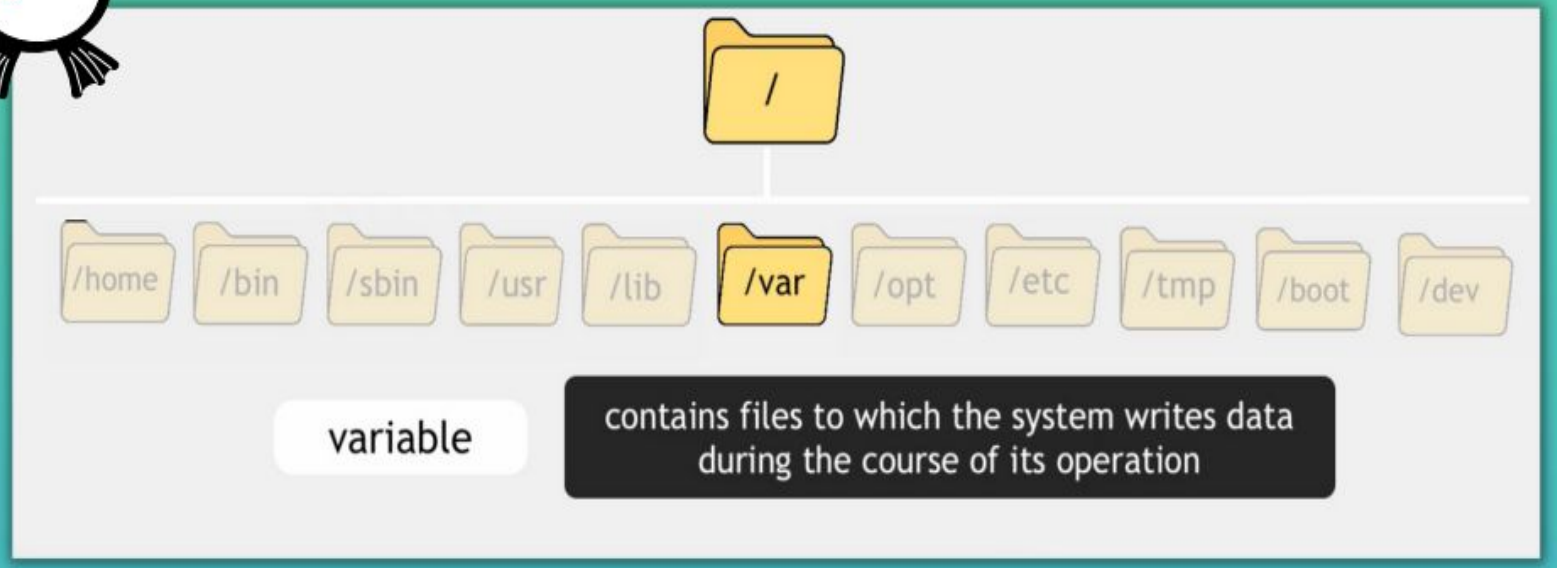
Linux File Structure - 6



- Apps and Drivers will access this, NOT the user
- Files that the system needs to interact with the device



Linux File Structure - 7



/var/log

- Contains log files

/var/cache

- Contains cached data from application programs

Linux File Structure - 8

/media

- Contains subdirectories, where **removable media devices** inserted into the computer are mounted
- E.g. when you insert a CD. A directory will automatically be created and you can access the contents of the CD inside the directory

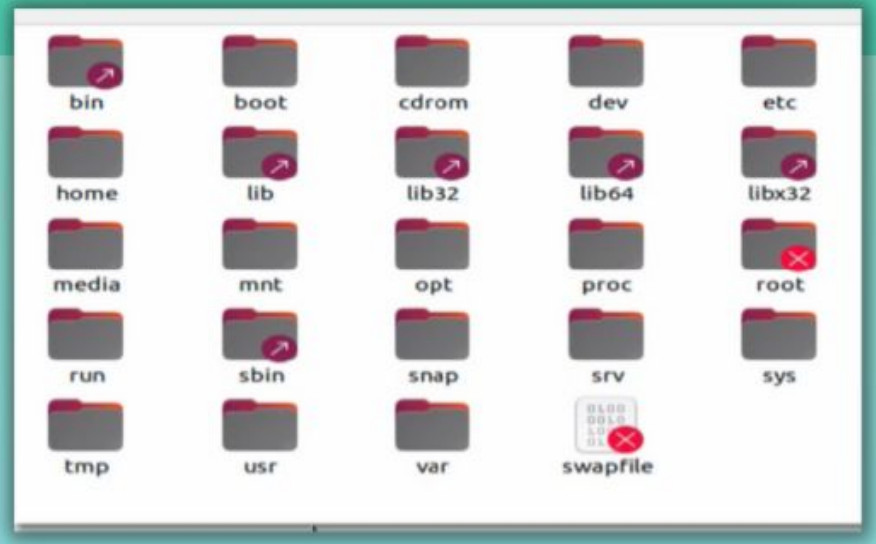
/mnt

- Temporary mount points
- Historically, system administrators mounted temporary file systems here

Linux File Structure - 9

Interactions with these root folders

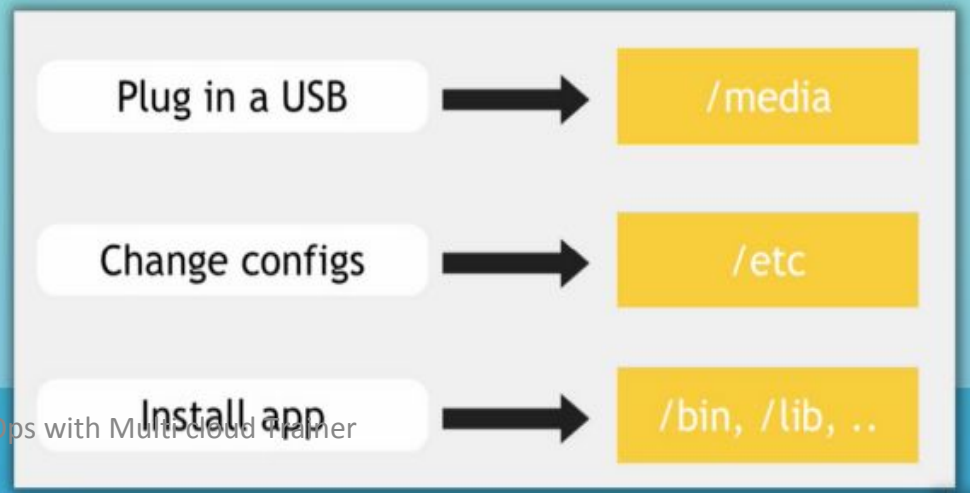
- Usually **you are not interacting with these root folders**



You install apps with package manager

Interacting with Operating System

- Installer/Package Manager is handling this
- OS is handling this



ubuntu@ip-172-31-8-244:~$
username@computername:home dir ($ normal user # root user)

CLI vs. Terminal

- **CLI** = Command Line Interface, where users **type in commands** and see the results printed on the screen

- **Terminal** = the **GUI window** that you see on the screen. It takes commands and shows output

clear
PWd
cd
mkdir
ls
ll or ls –la
touch
rm
Cd ..
rm –rf <foldername>
cd / (root folder)
cd ~ (home Directory of normal user)
cd usr/local/bin
cd ../../..

mv (rename)
Mv (file-name) (new-name)

cp (copy)
Cp –r (dirname) (new_dirname)
All the files from this will be copied

ls (foldername)

ls –R code

history
ctrl+r search history

ctrl+r reverse search history

ctrl+c kill process
ctrl+shift+v paste on terminal
ctrl+shift+c copy on terminal

history 10 will show history of last 10 used command
ls –la
ls –a

cat <file>
Will show content of file

uname -a
will display distribution details

cat /etc/os-release

lscpu
Architecture of cpu …end to end details about cpu

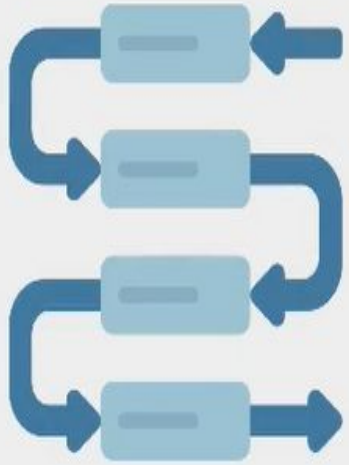
#lsmem
Will display memory details

sudo adduser admin01
ls /home
sudo ls /home/admin01
sudo ls –a /home/admin
#sudo addgroup devOps
su – admin01
Switch between the user

What is a software package?

Software Package

- A compressed archive, containing all required files
- Apps usually have dependencies



What is a Package Manager?

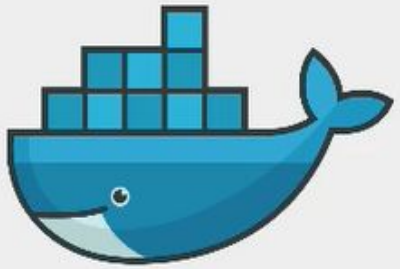
Package Manager

- downloads, installs or updates existing software from a repository
- ensures the integrity and authenticity of the package
- manages and resolves all required dependencies
- knows where to put all the files in the Linux file system

```
# sudo apt search openjdk

# java                                  # will see list of suggestions

# sudo apt install openjdk-17-jre-headless

# java –version

# sudo apt remove openjdk-17-jre-headless
```

1 central place to
install, upgrade, configure, remove software

With Package Manager you can install all the software we need for the Bootcamp



APT-GET

you can achieve the same, if you use additional command options

search command not available

APT

more user friendly, like progress bar

fewer, but sufficient command options in a more organized way

Where are these files stored?

Repository

storage location,
containing thousands of programs

- Package Manager fetches the packages from these repositories
- **Always update the package index before uppgrading or installing new packages**

sudo apt update

Updating package index

- pulls the latest changes from the APT repositories
- The APT package index is basically a database
- Holding records of available packages from the repositories

ls /etc/apt

ls /etc/apt/sources.list

Sources.list file

You need to know alternative ways to install software

Reasons, why you need alternative ways

- There are packages, that are **not available** in these official repositories
- Available, but **not the latest version**

Software Packages are verified, before adding to repository

Verification Process takes time

You need to know alternative ways to install software

Reasons, why you need alternative ways

- There are packages, that are **not available** in these official repositories
- Available, but **not the latest version**
- Browsers, Code Editors etc. are not available

You need to know alternative ways to install software

1) Alternative:
Ubuntu Software Center

2) Alternative:
Snap Package Manager

3) Alternative:
Add Repository to official list of repos
(add-apt-repository)

2) Alternative: Snap Package Manager

- Initial release in 2014
- Many still use the term "snappy", which it was called before
- Snap is a software packaging and deployment system
- For OS using the Linux kernel

How is Snap different?

Snap

- A snap is a bundle of an app and its dependencies

Snap Store

- provides a place to upload snaps, and for users to browse and install the software

Snapcraft

- is the command and framework used to build and publish snaps

Difference and which to use?

Snap

APT

Snap	APT
Self-contained - dependencies contained in the package	Dependencies are shared
Supports universal Linux packages (package type .snap)	Only for specific Linux distributions (package type .deb)
Automatic Updates	Manual Updates

3) Alternative: Add Repository to official list of repos (add-apt-repository)

1) Add repository to APT sources (into */etc/apt/sources.list*)

2) Install package as usual

Add Repository

- When installing relatively new applications
- Which are not in official repositories yet

Official Repository

Add the Docker repository to APT sources:

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bio
```

Add Repository

- When installing relatively new applications
- Which are not in official repositories yet

Official Repository

- Repository will be added to */etc/apt/sources.list*
- The next "apt install <package name>" will also look into this new added repository

Add Repository

PPA = Personal Package Archive

- PPAs are provided by the community
- Anybody can create this PPA - private repository to distribute the software
- Usually used by developers to provide updates more quickly than in the official Ubuntu repositories

Some common ways of installing software

Apt - Package Manager

- use when possible

Alternatives:

Snap - Package Manager

Add Repository

Ubuntu Software Center

Debian based

- Ubuntu
- Debian
- Mint

APT

APT-GET

Red Had based

- RHEL
- CentOS
- Fedora

YUM

APT

YUM

Similar concept:

- Package Manager uses official repositories
- Downloads packages, resolves dependencies etc.

Difference of Repos

- More or newer versions of packages

Introduction to Vi & Vim

Use Cases to use text editor in CLI

- Small modifications can be faster, especially when you are currently working in the CLI
- Faster to create and edit at the same time
- Supports multiple formats
- When working on a remote server

Introduction to Vi & Vim

Use Cases to use text editor in CLI

- Git CLI - writing the Git Commit Message

- Display Kubernetes Configuration Files

- Quickly editing one line or character in a file

Note: sometimes vim editor will not be there in the server. So we have to install

To test

```
# vim class.txt
```

If not there

```
# sudo apt install vim
```

Ready to use

Command Mode

- This is the **default** mode
- You **can't edit the text**
- Whatever you type is **interpreted as a command**
- Navigate, Search, Delete, Undo etc.

Insert Mode

- Allows you to **enter text**

i= insert the text or write the text
wq! = save and exit
q! = only exit without save
Or shortcut is
:x save and exit
Can create new file as well

Vim test.txt will create new test.txt file

- dd= to delete the line but in command mode
- D 10 d to delete 10 lines once
- U= undo the changes
- Shift + A= end of the line and insert mode too
- Type 0 (zero)= to reach beginning of the line
- 10 G = jump to line 10
- /search /nginx to search something n=jump to next search N=jump to previous
- %s/old/new will replace older string value to new one

Overview

- User Accounts
- Groups & Ownership & File Permissions
- Linux Commands for Managing Users and their permissions

3 User Categories

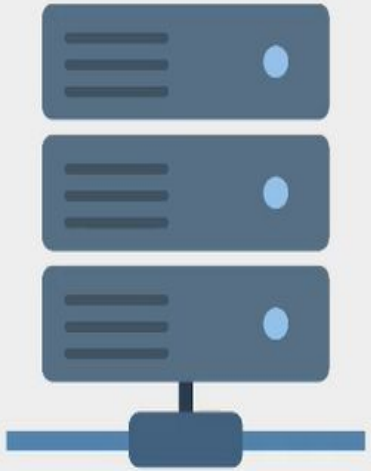
Superuser Account

User Account

Service Account

- Relevant for Linux Server Distros
- Each service will get its own user
- Best Practice for Security
- Don't run services with root user!

3 User Categories



- Always **1 Root User per computer**

- Can have **multiple** regular users and service users

How does this work?

- Admins add users to the system

- All computers are connected to this system

- When someone tries to login, OS checks it with the system

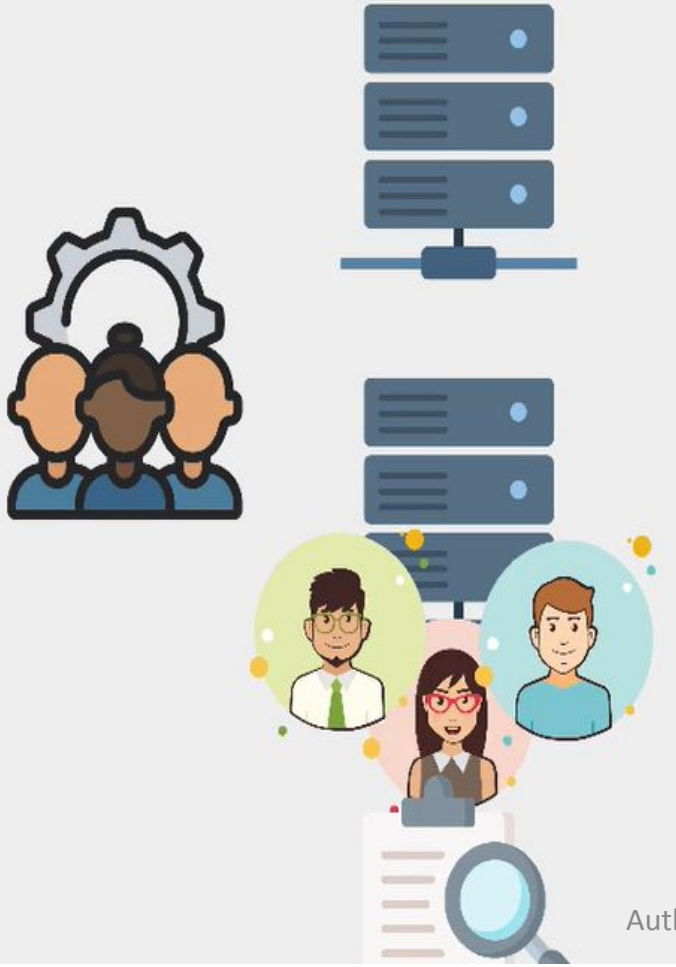
Multiple Users on a Server

- For Linux having Multi-User is important for Servers
- Usually teams administer a server

Why not just use a shared user?

Multiple Users on a Server

Why having a User for each team member is important



- They need a **non root user**

Permissions per team member

Traceability - who did what on the system?

How to manage Permissions

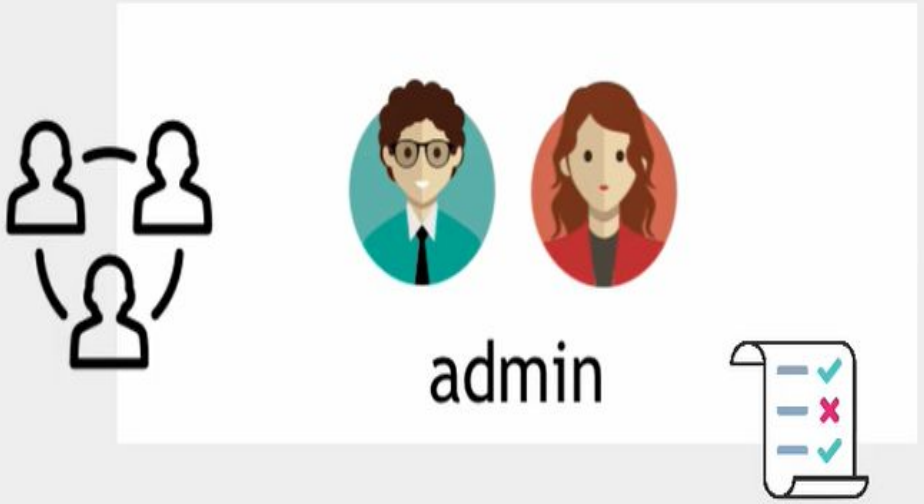
User Level

- Give permissions to User directly

Group Level

- Group Users into Linux Groups
- Give permissions to the Group

How to manage Permissions



- Users are added to the Group
- Permissions for that Group

```
# Cat /etc/passwd      #user information
#mahesh:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
Username : password : UID : GID : HOMEDIR: shell

# sudo passwd <username>                                    # to change password
# su - <username>                                           # switch between the user
# sudo groupadd DevOps                                      # group will be added
# cat /etc/group                                            #will show list of group
# deluser or userdel                                        #delete the user
# sudo usermod –g <group> <user>                            #will become primary group for user
# sudo usermod  -G <group> <user>   #will be added to primary and secondary
group
# sudo usermod –aG <group> <user> #will append group from existing group
#groups       #will display user belongs to group
#groups <user> #groups user belongs to
#sudo gpasswd –d <user> <group>
```

Different User & Group Commands

adduser	*addgroup*	*useradd*	*groupadd*

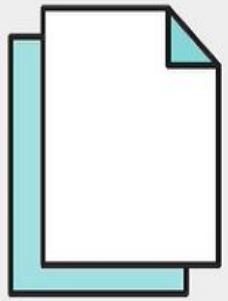
- Interactive
- More User Friendly

- You need to provide the infos yourself
- **Low-level** utilities

Easier to use!

- Like it chooses conformant UID and GUID values for you
- Creates a home directory with skeletal config automatically
- Or asks for input in interactive mode

File Permissions and Ownership



User Permissions

- User Permissions are related to reading, writing and executing files

```
# sudo chown <old>:<new> file.txt            #will change ownership from old to new
# ls –l /etc/                        #list of permission in etc
# sudo chgrp devops file.txt    #devops group will be the owner

#sudo chmod –x <folder>    #will take execute permission from all the owners
# ll                                #to check the permission

Owner          group                other
O                 g                      o

#sudo chmod g-w <folder>   # group will not have any write permission
#sudo chmod g+w <folder>    #write permission will be added
Other way of changing the permission
# sudo chmod g=rwx <filename>       #all permissions will be assigned
# sudo chmod 777 <file>                    #all permissions to file


Same ways can be changed for hidden file or folder as well
```

Absolute(Numeric) Mode

Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write + Execute	rwx

```
-rw-rw-r-- 1 admin devops    0 Mai  7 13:06 test.txt
```

drwxrwxr-x

File Type:

- `-` regular file
- `d` directory
- `c` character device file
- `l` symbolic link

etc.

d r w x r w x r - x

Owner

r Read

w Write

x Execute

- No permission

▶ Execute a script/executable

```
-rw-rw-r-- 1 admin devops    0 Mai  7 13:06 test.txt
```

- **r** Read
- **w** Write
- **x** Execute
- **-** No permission

- All other users, who are not the file owner or don't belong to the group owner

3 ways to set permissions

1) Symbolic Mode

+ add

- remove

r Read

w Write

x Execute

- No Permission

2) Set permission

= set the permission and override the permissions set earlier

3) Numeric Mode

4 Read

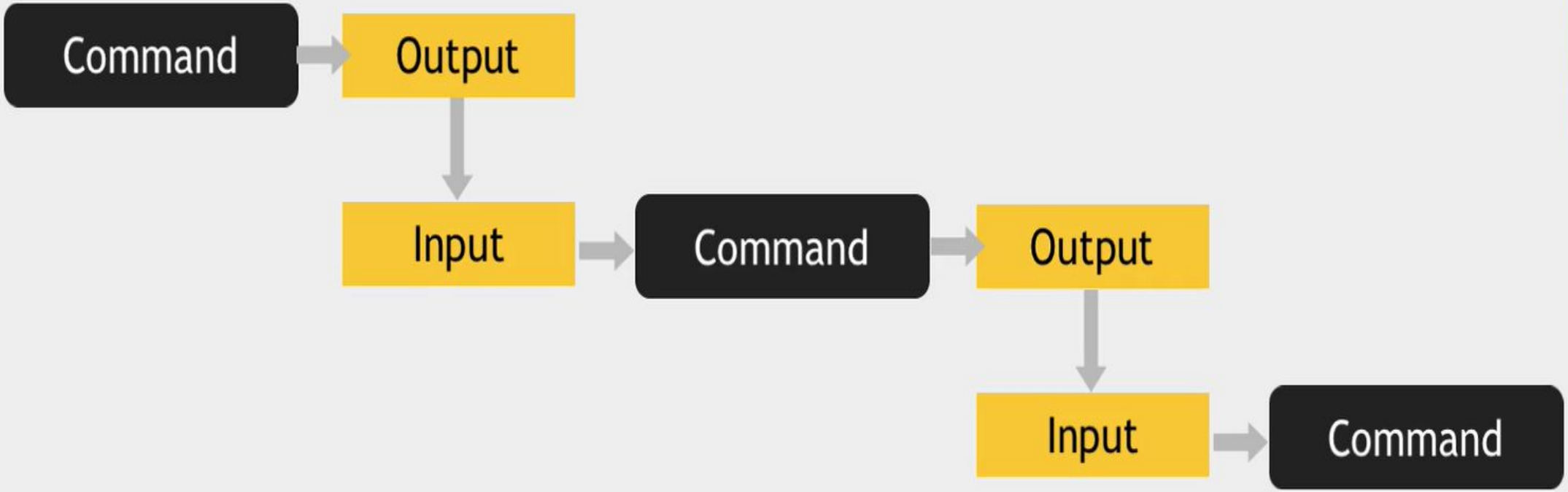
2 Write

1 Execute

0 No Permission

Input and Output

- The output from one program can become the input of another command



Piping concept

"pipe" command: |

- Pipes the output of the previous command as an input to the next command

less

- Displays the contents of a file or a command output, **one page at a time**
- Allows you to navigate forward and backward through the file
- Mostly **used for opening large files**, as *less* doesn't read the entire file, which results in faster load times

cat /var/log/syslog # will display log in terminal window which server is generating

cat /var/log/syslog | less # less will give page by page view of the content

Space bar jump to next page
P=previous page
Q= quit the program

history | less

Grep

grep

- Stands for: Globally Search for Regular Expression and Print out
- Searches for a particular pattern of characters and displays all lines that contain that pattern

Redirection

- > character is the redirect operator
- Takes the output from the previous command and sends it to a file that you give

```
# history | grep sudo                                    # will list out history of sudo

# history | grep sudo | less                            # will list out the history and will pipe to next


# ls /usr/bin/ | grep java

# history | grep sudo > sudo-commands.txt       # will have list out history of sudo in sudo-commands.txt file

#cat sudo-commands.txt > commands.txt          # will print the results of cat in commands.txt

#cat newfile >> commands.txt

#clear; sleep 5 ; echo “welcome to the world of devOps ”
```

Essential Commands in Linux

Pipes (|)

Redirects (>)

less

grep

Overview

- What is a Shell? What is Bash?

DEMO: Concepts & Syntax of Bash Scripting

- Variables
- Conditional Statements
- Passing Arguments to Script
- Read User Input
- Loops
- Functions

Introduction to Shell Scripting

Configured a server

- **useradd** tim
- **groupadd** devops
- **mkdir** project
- **touch** file.txt
- **chmod** 750 /path
- **sudo apt** docker
- docker run ...

Execute manually one by one

- Run same commands in the same order on another server

Introduction to Shell Scripting

Introduction to Shell Scripting

- Write commands in a file
- Execute that file

File is moveable

Introduction to Shell Scripting

- Such file is called a **shell script**
- Shell Scripts have a **.sh file extension**

Shell vs. sh vs. Bash

- **Shell** = The **program that interprets and executes the various commands** that we type in the terminal

Translates our command that the OS Kernel can understand

Shell vs. sh vs. Bash

- **Shell** = The **program that interprets and executes the various commands** that we type in the terminal

Different Shell Implementations

- **sh** (Bourne **Sh**ell) - */bin/sh*
- **Bash** (Bourne **a**gain **sh**ell) - */bin/bash*
  - improved version of sh
  - **default shell program** for most UNIX like systems

Shell vs. sh vs. Bash

- Shell and Bash terms are often used interchangeable

Bash Scripting

Bash Commands

- Bash is a shell program
- Bash is a programming language

Shebang

- All shell script files have the same .sh file extension

How does OS know which shell to use?

SH	BASH	ZSH

We need to tell the OS!

#!/bin/sh	#!/bin/bash	#!/bin/zsh

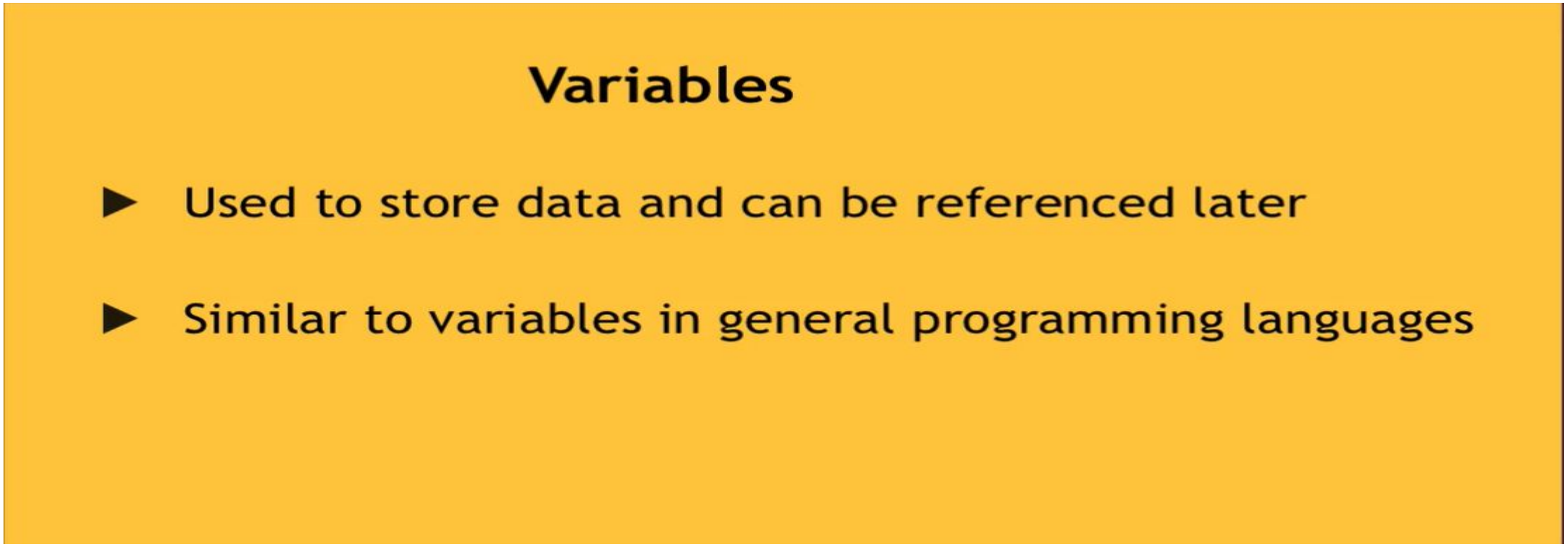
Why is it called "shebang"?

- Because of the first 2 characters: "#!"
- **#** = in musical notation, also called "sharp"
- **!** = also called "bang"
- Shebang became a shortening of sharp-bang

```
# touch  setup.sh
# vim setup.sh


#!/bin/sh                                   # will execute on any flavour of linux
#!/bin/bash                                 # will excute on bash …if bash is not installed then
it will show error

Echo “welcome to DevOps”

./setup.sh                                          # to execute the shell script
Ls –l setup.sh                                      # will execute only if we have
execute permission

# sudo chmod u+x setup.sh
```

Variabels

```
file_name=config.yaml
echo “use file $config.yaml to configure”
```



Conditionals

- Allow you to alter the control flow of the program
- E.g. Execute a command only when a certain condition is true

- If
- Else
- fi

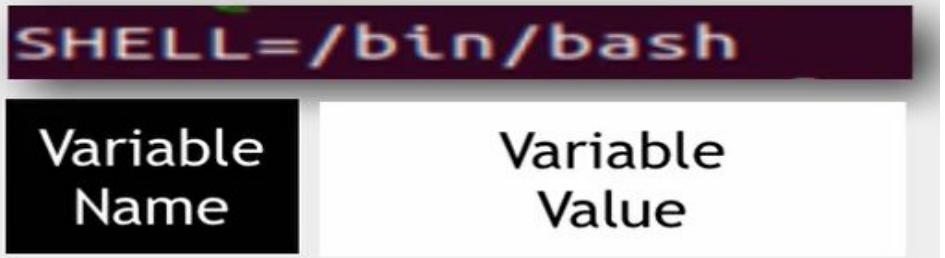
```
#!/bin/sh

echo “setup and configure server”

file_name=config.yaml

If [  -d “config” ]
Then
        echo “reading config directory contents”
        config_files=$(ls config)
else
        echo “config dir not found. Creating one”
         mkdir config
fi

        echo “using file $file_name to configure something”
        echo “here are the all config files: $config_files”
```

Font	Shell	Editor	Browser
Big	bash	Vim	Chrome
Small	sh	Nano	Firefox

- Each user has its own environment
- Each user can configure its own environment/account by setting preferences
- These OS configurations should be isolated from other user environments

Where does OS store all these configurations?

Environment Variables



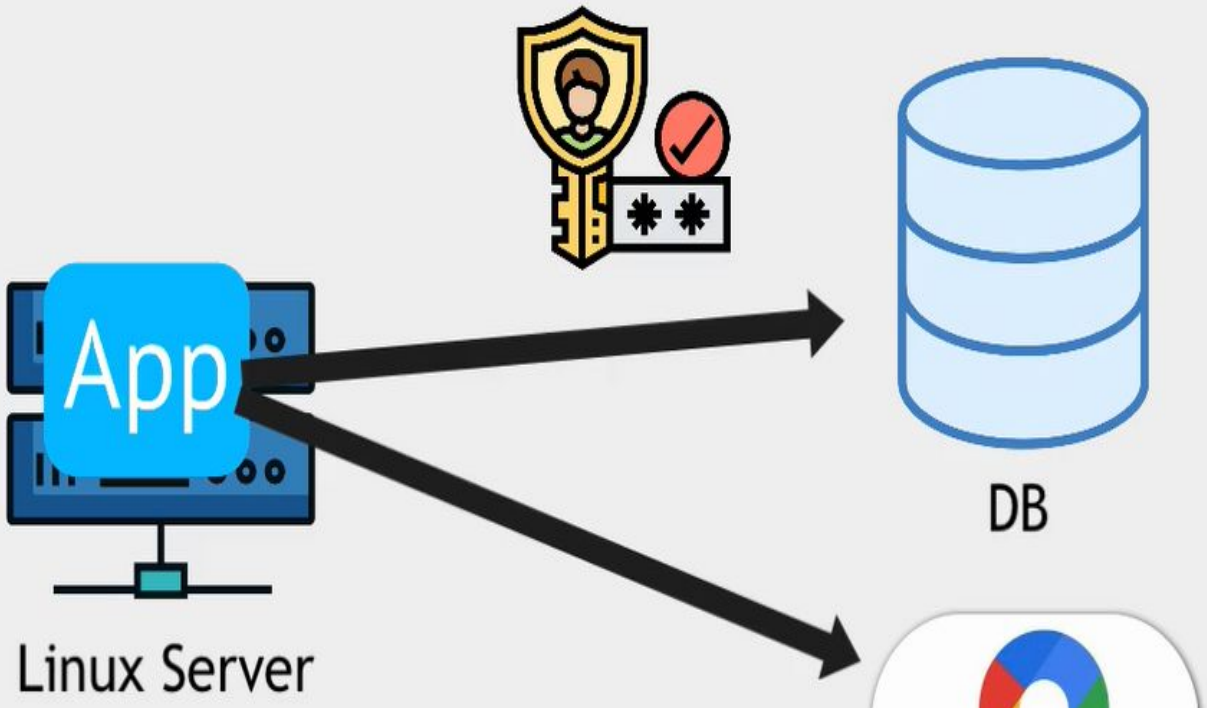
Stored variable value only for specific

 # printenv

Will list out value of environment defined

- KEY=value pairs
- Variables store information
- Environment Variables are **available for the whole environment**
- By convention, names are defined in UPPER CASE

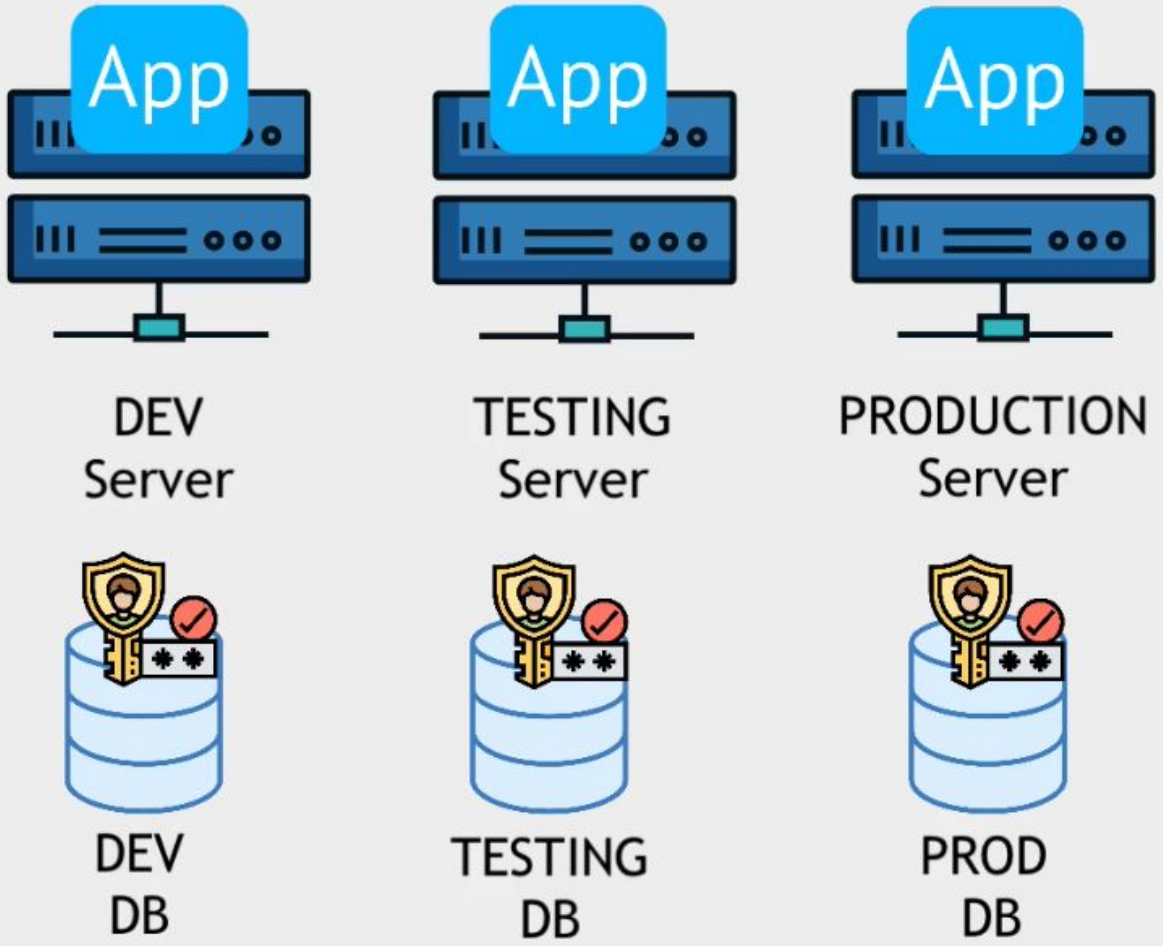
Use Case: Sensitive data for application




```
import mysql.connector

mydb = mysql.connector.connect(
  host="htt  /pr  atabase:5432",
  user="mysql  me",
  password="s  ssword"
)

print(mydb)
```

- App needs some credentials
- Not secure!

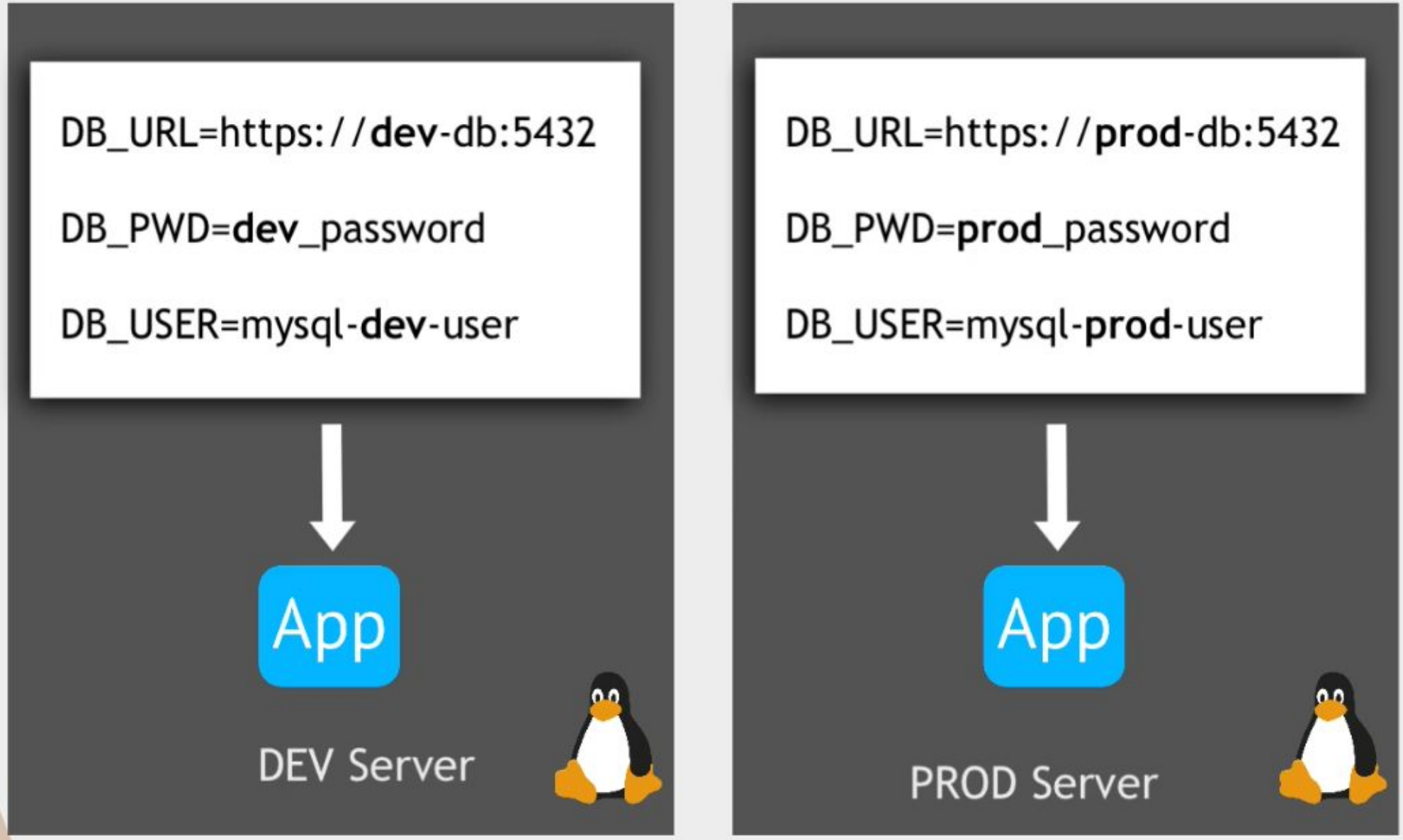
Use Case: Make application more flexibel




```
import mysql.connector

mydb = mysql.connector.connect(
  host="https://prod-database:5432",
  user="mysqlusername",
  password="secretpassword"
)
```

- Do not hardcode these connection values (Db URL, Db User, Db Pwd)

- Each DB will have its own credentials


DB_URL=https://dev-db:5432
DB_PWD=dev_password
DB_USER=mysql-dev-user
App
DEV Server
DB_URL=https://prod-db:5432
DB_PWD=prod_password
DB_USER=mysql-prod-user
App
PROD Server

```
# printenv
# printenv USER                          # will print the env variable
# printenv | grep USER                   # if we have more than one variable defined with same name
# echo $USER

How to Defined your own variables
# export DB_USERNAME=name
# export DB_PASSWORD=value
# export DB_NAME=name
# printenv | grep DB                              # will get the list of variable defind

Note= we can change the value of variables once we defind them
Removing the variables
# unset <name of env variables>
Note = value defined In the terminal is gone once we close terminal…available only for current
session

So permanent save
# Vim .bash.rc                          #for
 # vim zshell.rc                         #for zshell
# source bash.rc
# printenv | grep DB
```

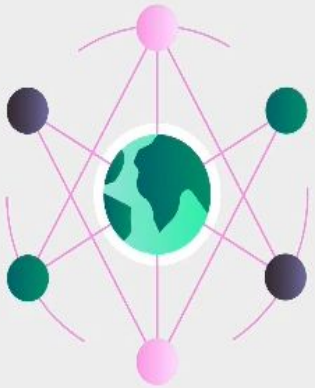
source ~/.bashrc =
Load the new env vars into the current shell session

/etc/enviroment

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
usr/local/games:/snap/bin"
```

PATH Environment Variable

- List of directories to executable files, separated by :
- Tells the shell **which directories to search for the executable** in response to our executed command



Networking

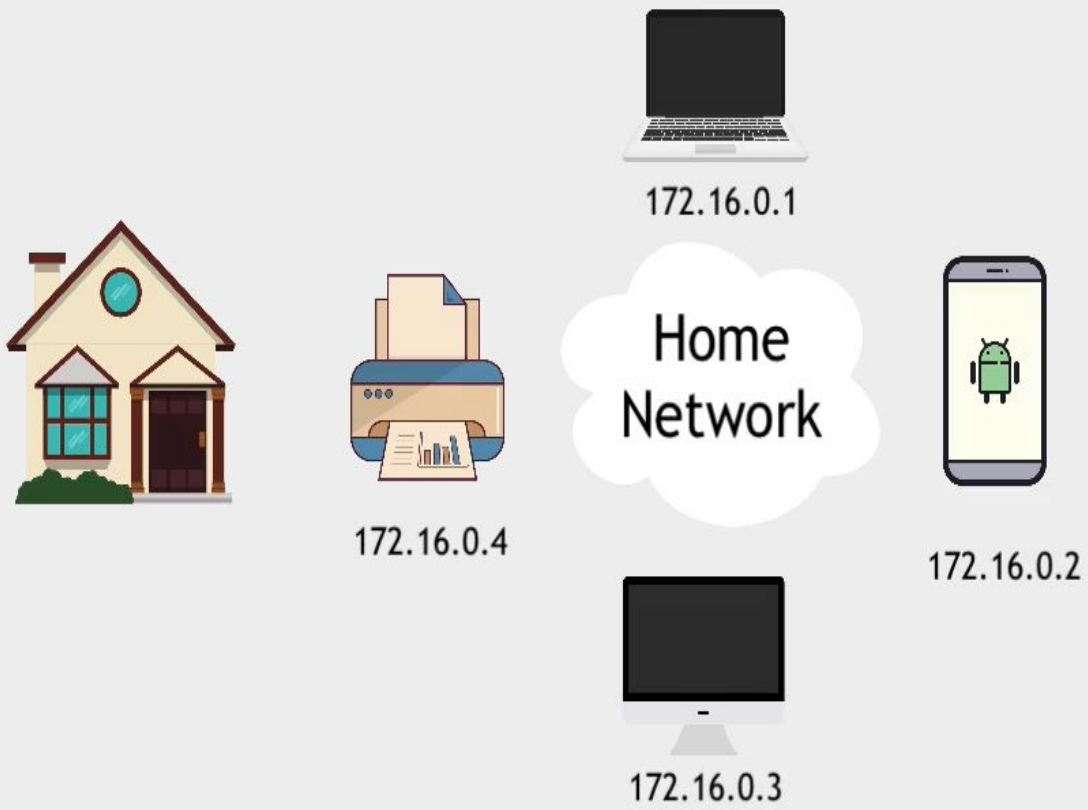
How does computer networks work?

How does computer connect to internet?

What is an IP address and port?

What is a DNS?

LAN - Local Area Network

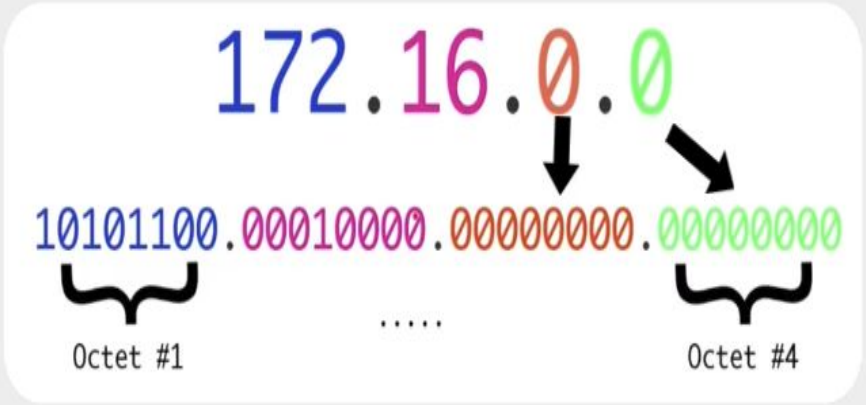


- Collection of devices connected together in one physical location
- Each device has a unique IP address

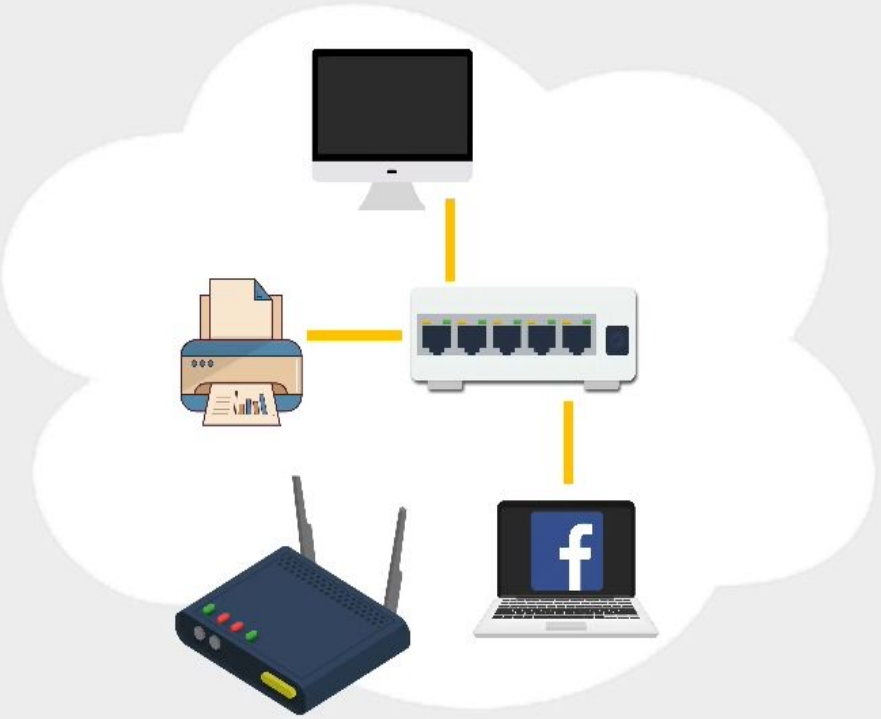
IP - Internet Protocol

- Devices communicate via these IP addresses

IP - Internet Protocol



- 32 bit value
- 1 bit = 1 or 0
- IP addresses can range from 0.0.0.0 to 255.255.255.255
- 00000000 => 0
- 11111111 => 255



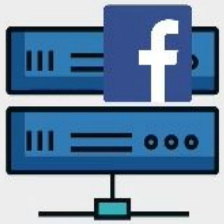
Switch

- Sits within the LAN
- Faciliates the connection between all the devices within the LAN

Router

- Sits between LAN and outside networks (WAN)

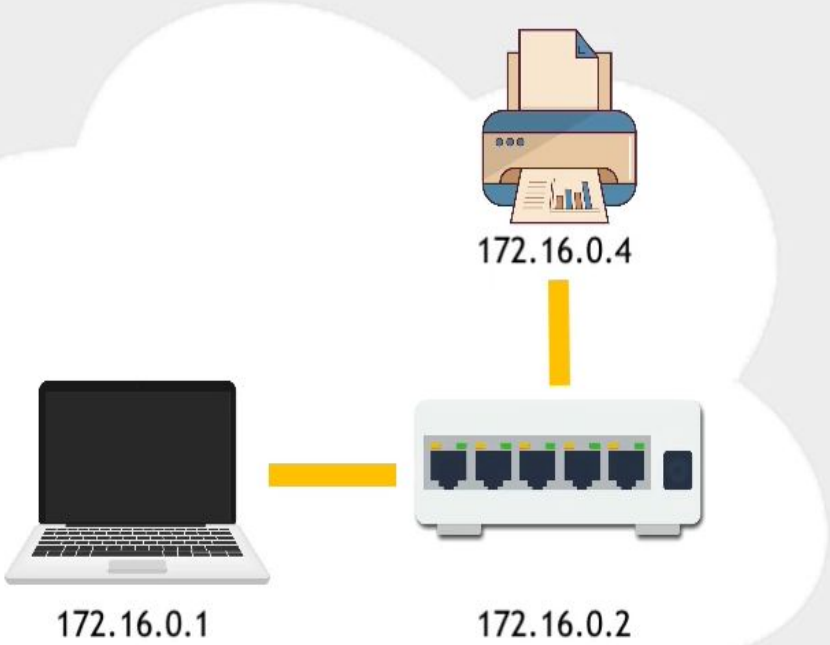
WAN = Wide Area network



- Connects devices on LAN and WAN
- Allows networked devices to access the Internet

Gateway = IP address of Router

Subnet



- It knows because of IP address of target device
- Devices in the LAN belong to **same IP address range**

Subnet = logical subdivision of an IP network

Subnetting = process of dividing a network into two or more networks

Subnet

- Example of IP address range: 192.168.0.0 255.255.255.0

1) IP address 2) Subnet Mask

255.255.0.0 - means that 16 bits are fixed

255.255.255.0 - means 24 bits are fixed

- Value 255 fixates the Octet
- Value 0 means free range

CIDR Block

- Subnet Mask dictates how many bits are fixed

255.255.0.0 - means that 16 bits are fixed

255.255.255.0 - means 24 bits are fixed

CIDR = Classless Inter-Domain Routing

- Shorthand

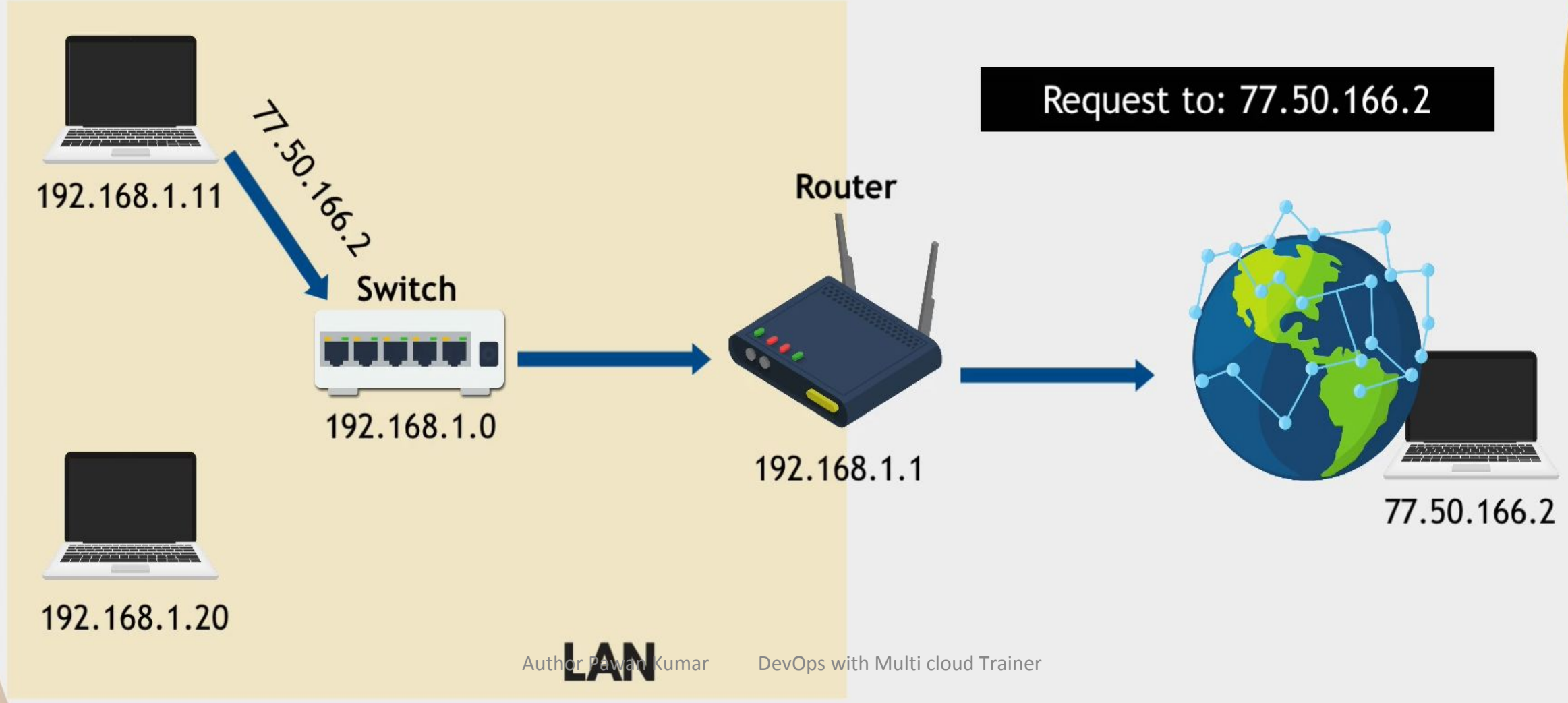
192.168.0.0/16 or 192.168.0.0/24

- /16 bits are fixed
- /24 bits are fixed

Recap - Example

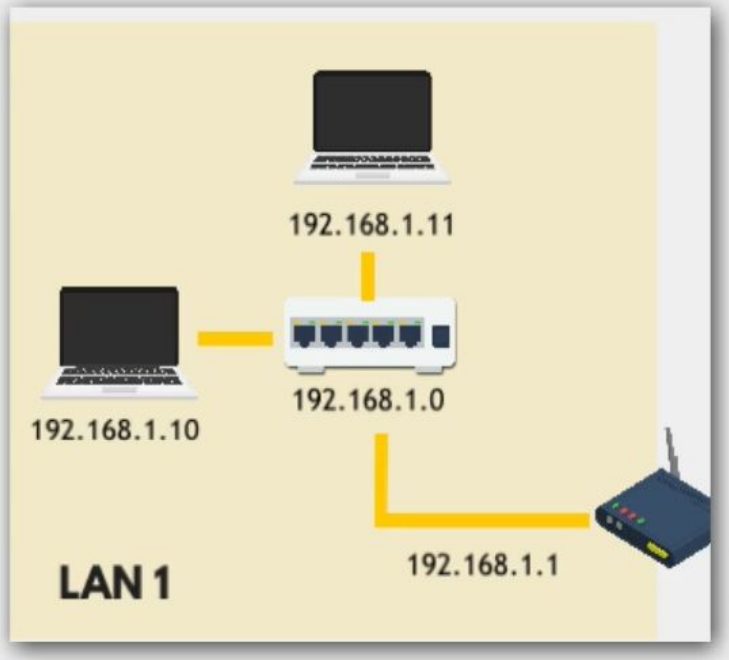
192.168.0.0 255.255.0.0

1) IP address 2) Subnet Mask



NAT

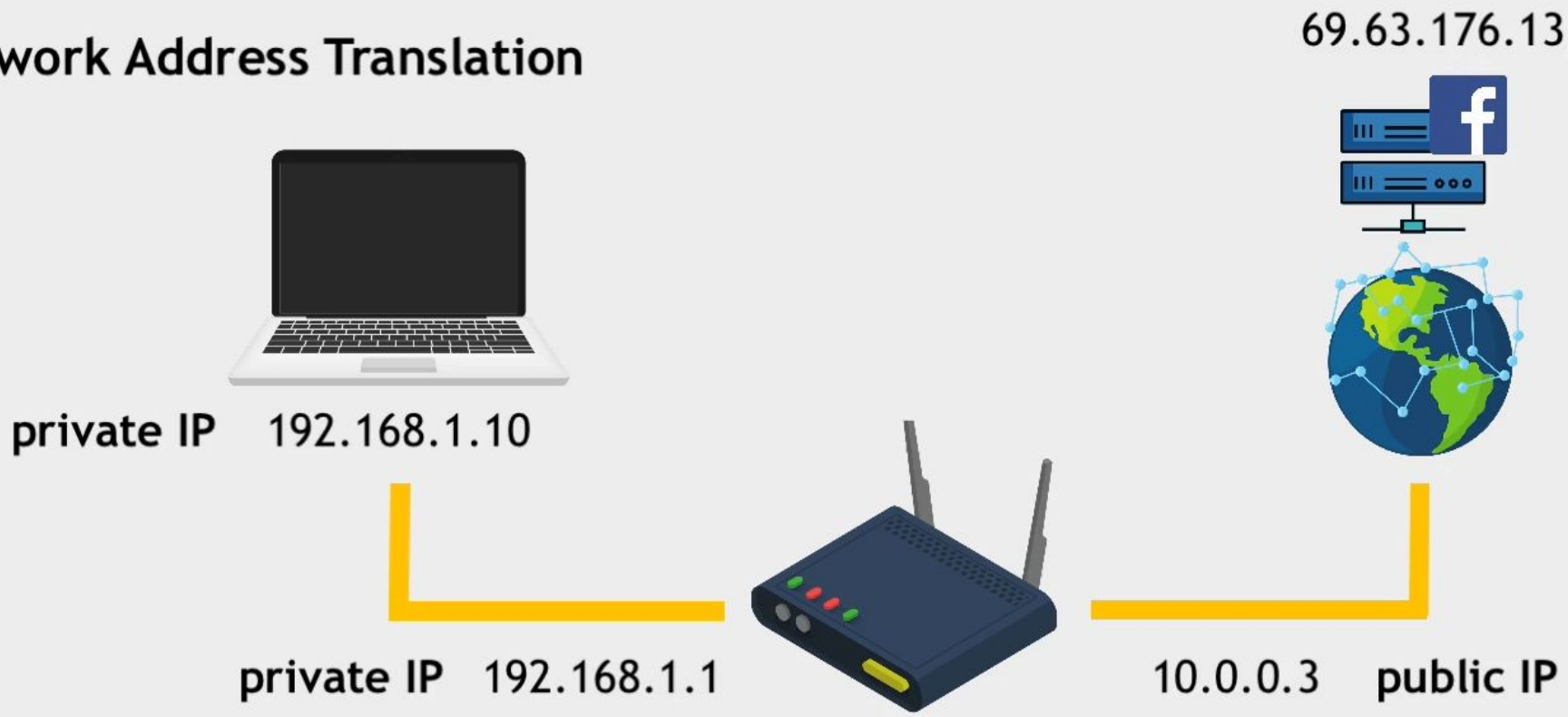
- IP Address range chosen by an administrator
- Each device gets a unique IP from that range



How to make sure that IP addresses don't overlap?

NAT

- Your Laptop's private IP address is replaced by the router
- **Network Address Translation**

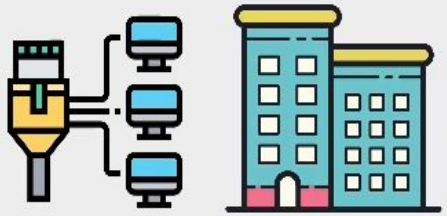


NAT = key functionality of a router

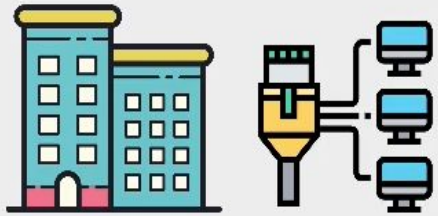
NAT

Benefits of NAT:

- **Security** and Protection of devices within LAN
- **Reuse** IP addresses



192.168.0.0 255.255.0.0

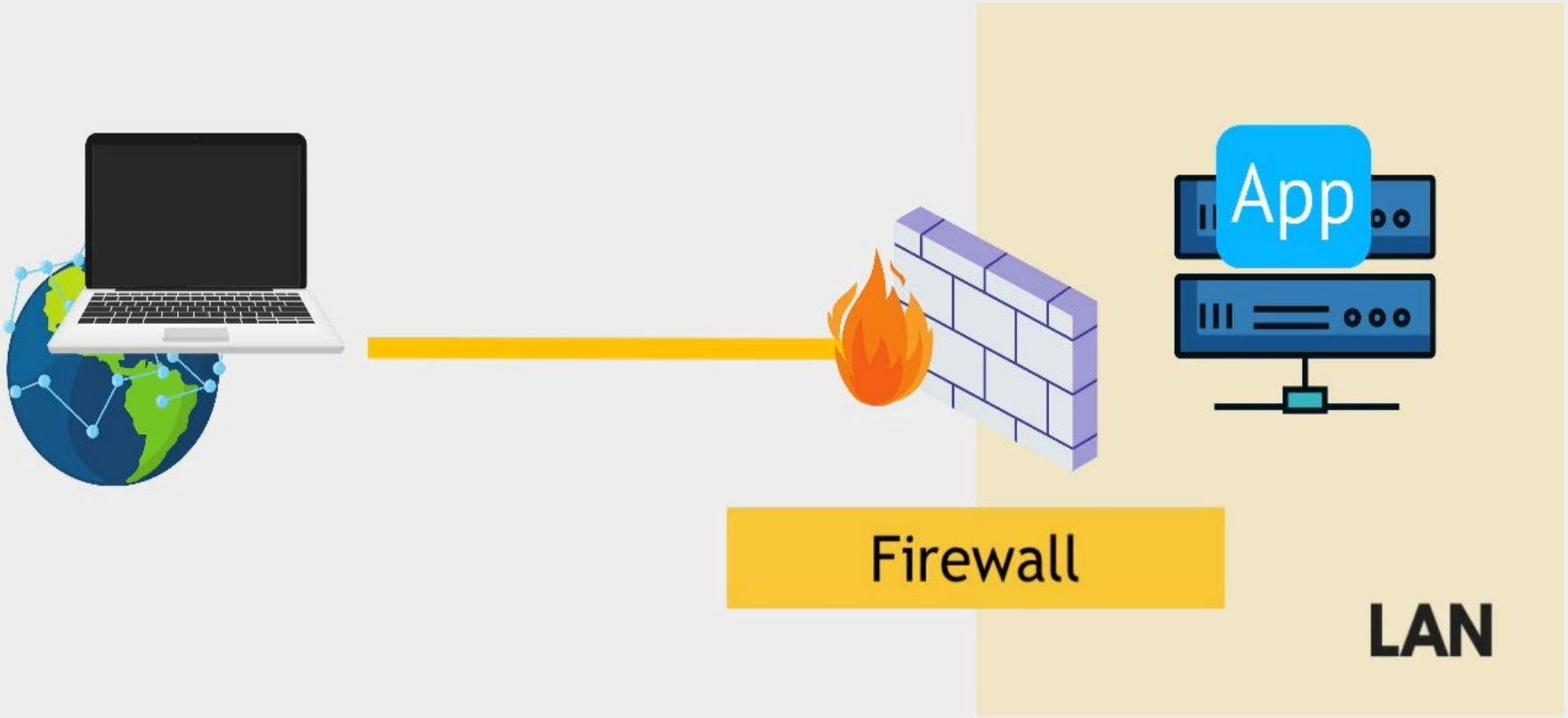


192.168.0.0 255.255.0.0

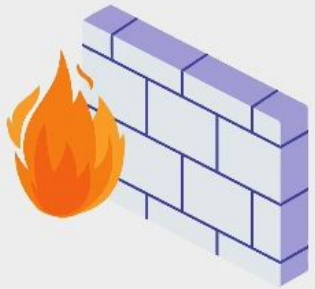
- Same IP address range within their LAN is OK

Firewall

- By default, the server is not accessible from outside the LAN



Firewall



- A system that **prevent unauthorized access** from entering a private network
- Using **Firewall Rules** you can define, which requests are allowed

Firewall Rules

- Which IP address in your network is accessible
- Which IP address can access your server
- You can e.g. allow any device access your server

Firewalls

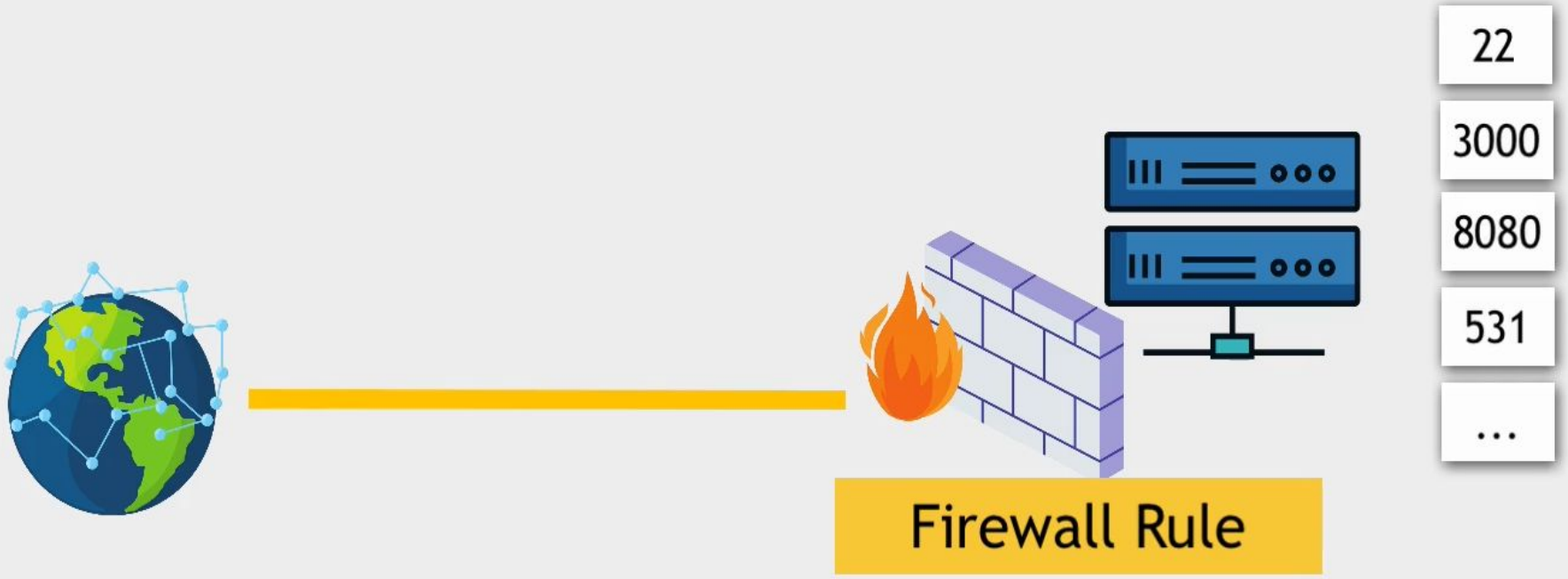
app-server-firewall

Inbound

Type	Protocol	Port Range	Sources
SSH	TCP	22	178.191.162.12
Custom	TCP	3000	All IPv4
Custom	TCP	8081	All IPv4

Port

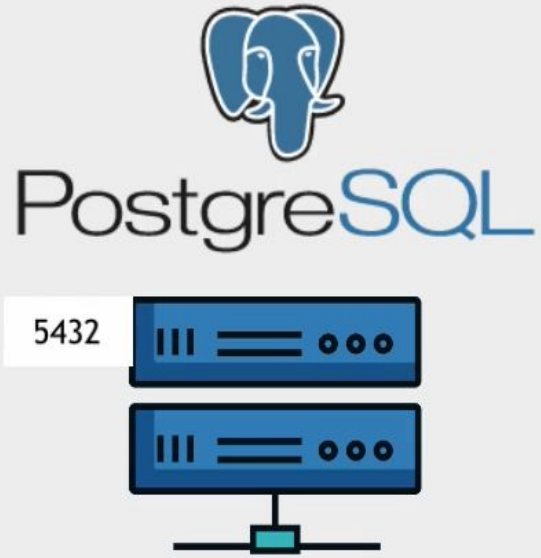
- Every device has a set of ports
- You can allow specific ports (doors)
- You can allow specific ports (doors) AND specific IP address (guests)



Port

- Different applications listen on specific ports
- **Standard ports** for many applications

Port 80	Web Servers
Port 3306	Mysql DB
Port 5432	PostgresQL DB



- For every application you need a port!
- **Each port is unique** on a device

```
Description:

Web server failed to start. Port 8080 was already in use.
```

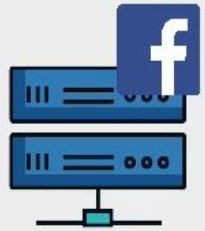
DNS - Domain Name Service

You type:
www.facebook.com

And not:
https://69.63.176.13

69.63.176.13

69.63.181.15



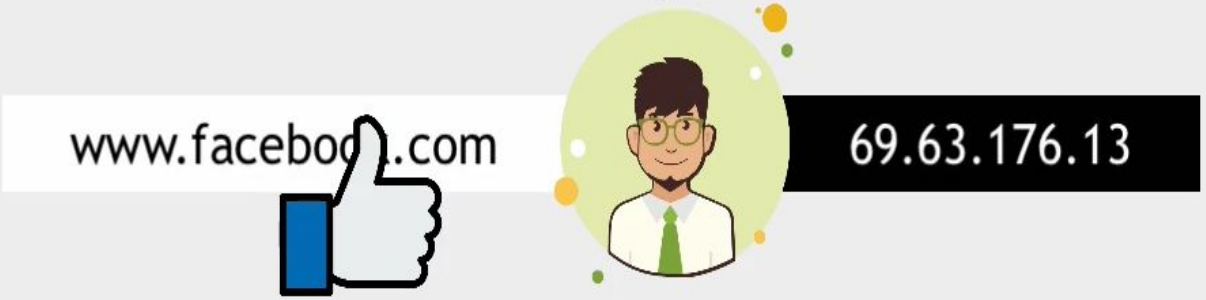
...

Why do we use names instead of IP addresses?

DNS - Domain Name Service

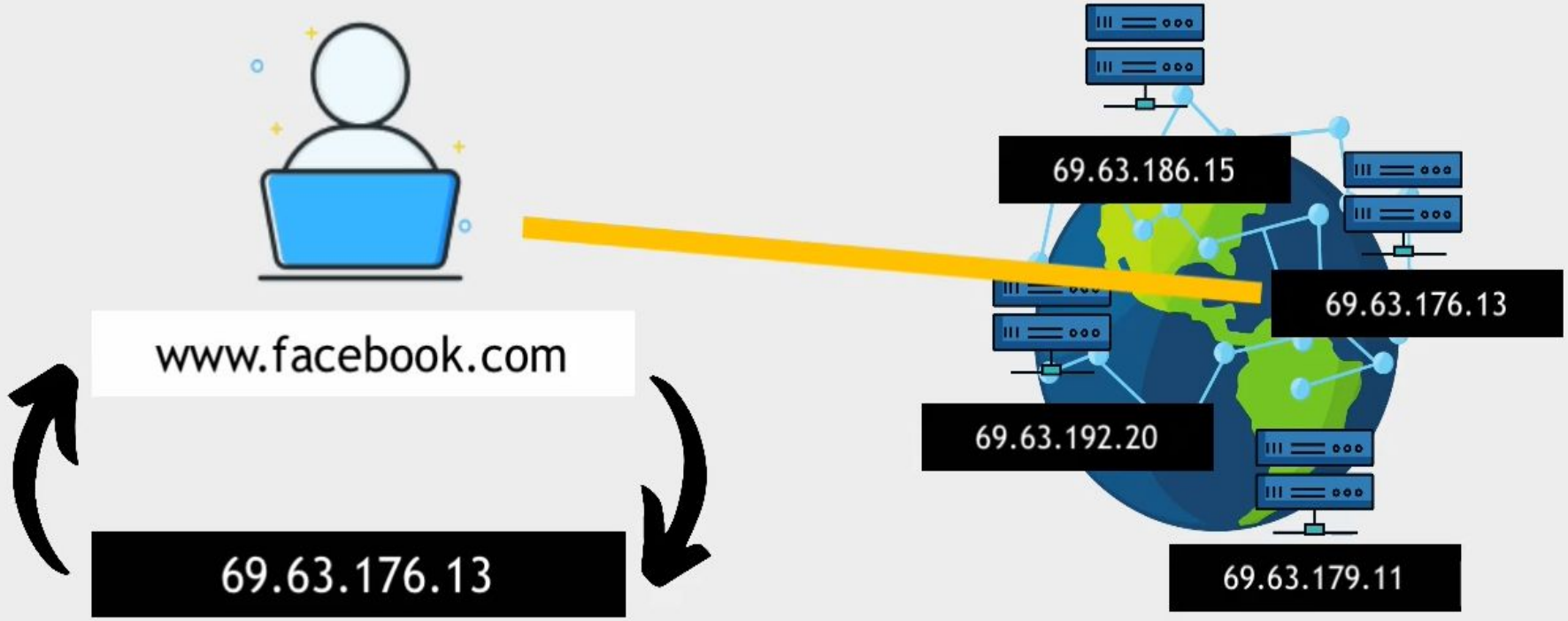
Why do we use names instead of IP addresses?

- Humans are better in remembering words and names instead of numbers



Mapping IP addresses to Names

DNS - Domain Name Service



DNS = translates domain names to IP addresses

Domain Names

www.google.com

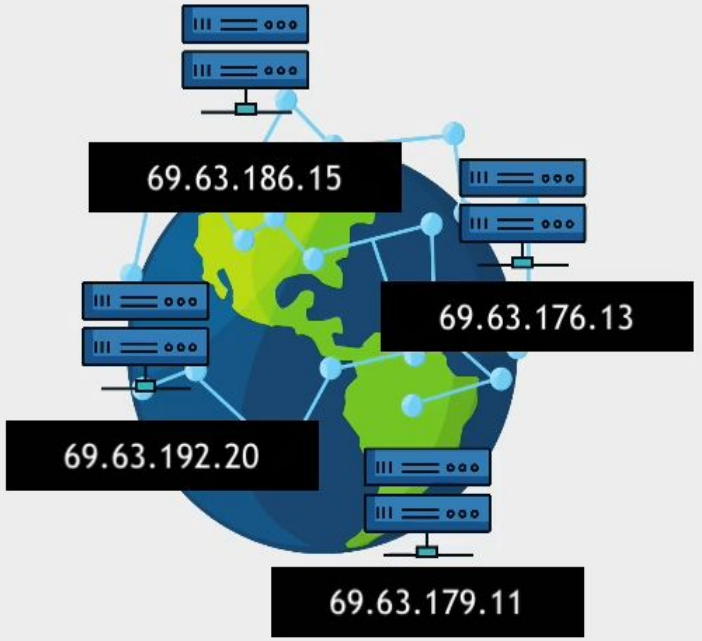
www.mit.edu

www.cncf.io

www.fairtrade.net

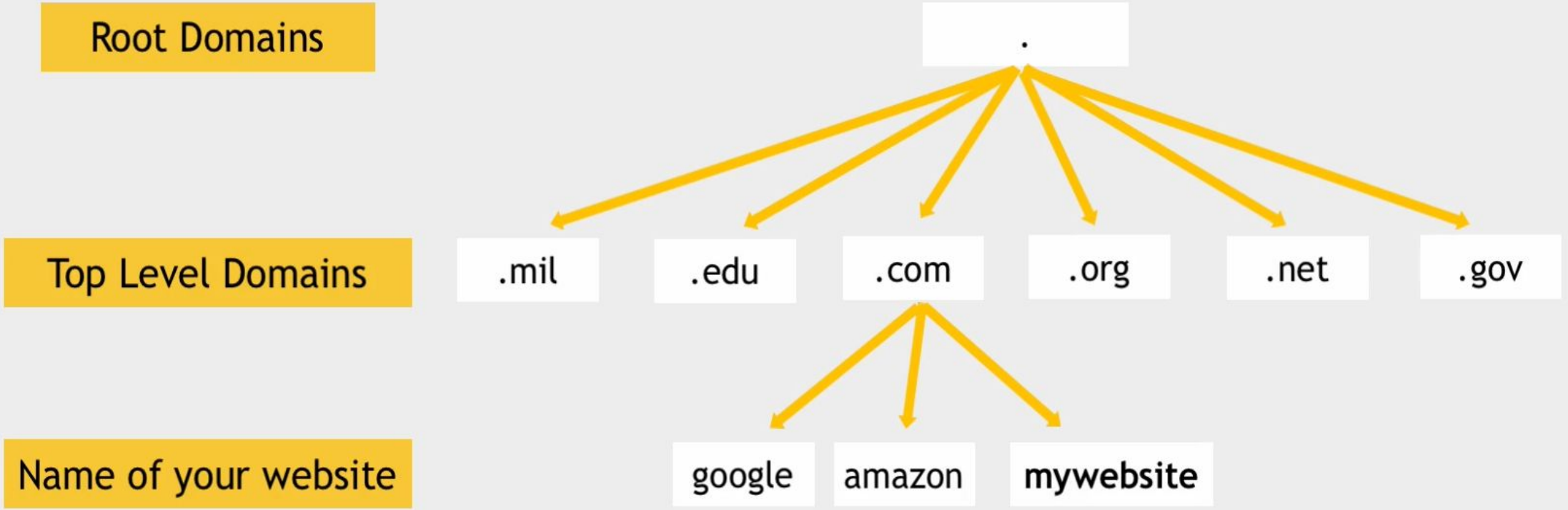
www.marines.mil

www.nyc.gov



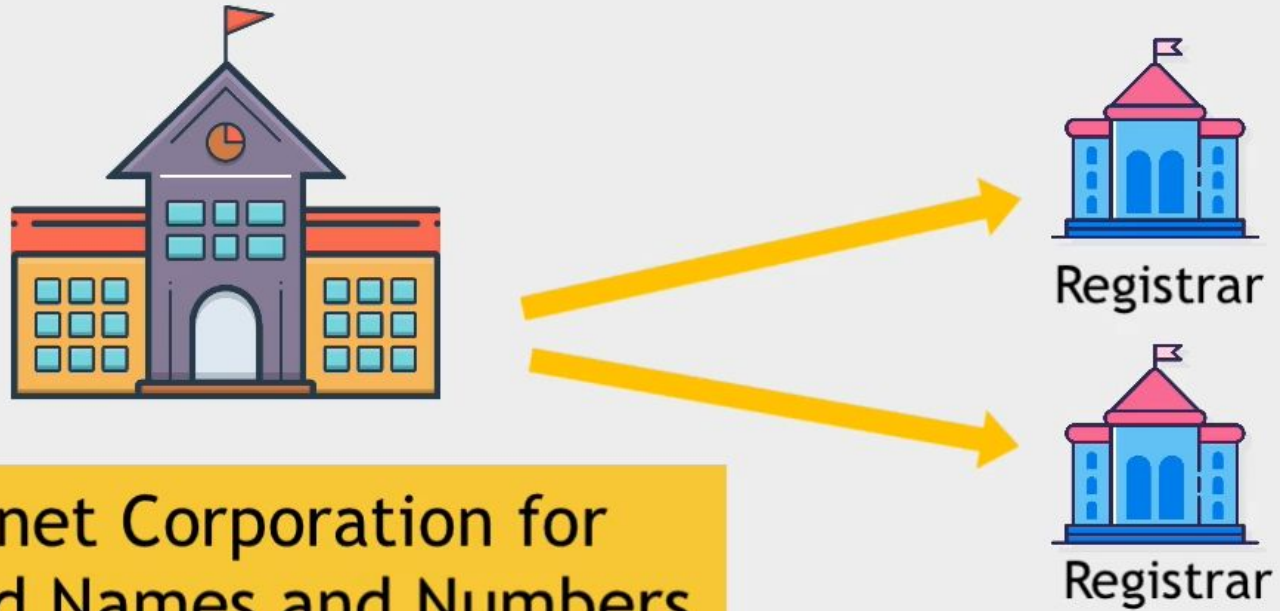
How does DNS manage all these IP addresses

Domain Names



- You can buy a domain name, e.g. mywebsite.com

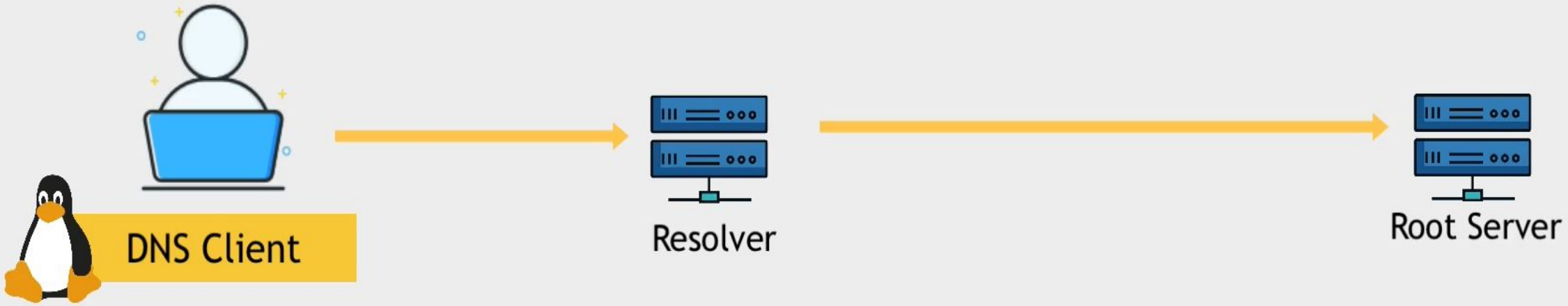
Domain Names



Internet Corporation for Assigned Names and Numbers (ICANN)

- Manages the TLD development and architecture of the internet domain space
- **Authorizes Domain Name Registrars**, which register and assign domain names

How does the DNS Resolution work?



ROOT SERVER

- Requests for top level domains
- Placed around the world

How does the DNS Resolution work?


Resolver

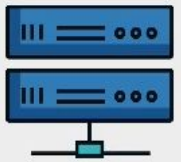

Name Server

"here IP address of facebook.com"

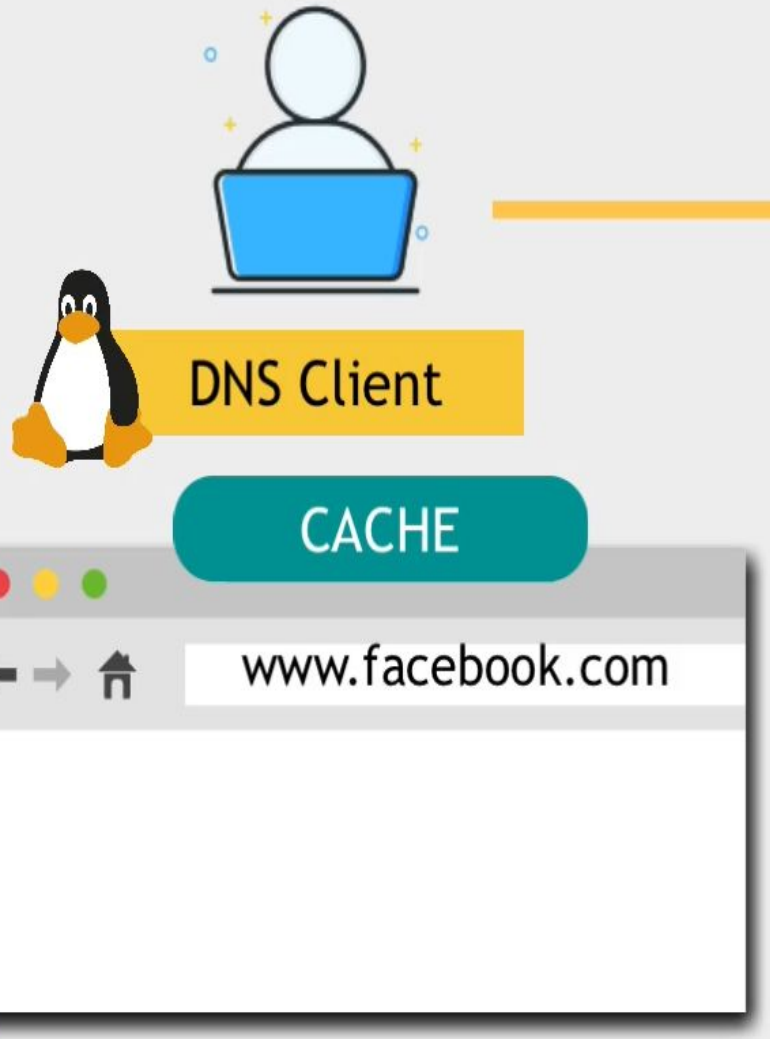
AUTHORITATIVE NAME SERVER

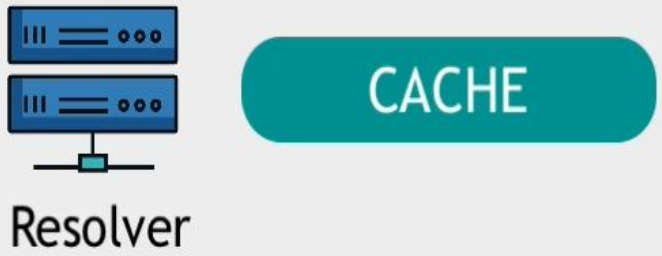
- Responsible for knowing everything about the domain, including IP address


TLD Server (.com)


Root Server

How does the DNS Resolution work?





- DNS Entries are **cached** for efficiency

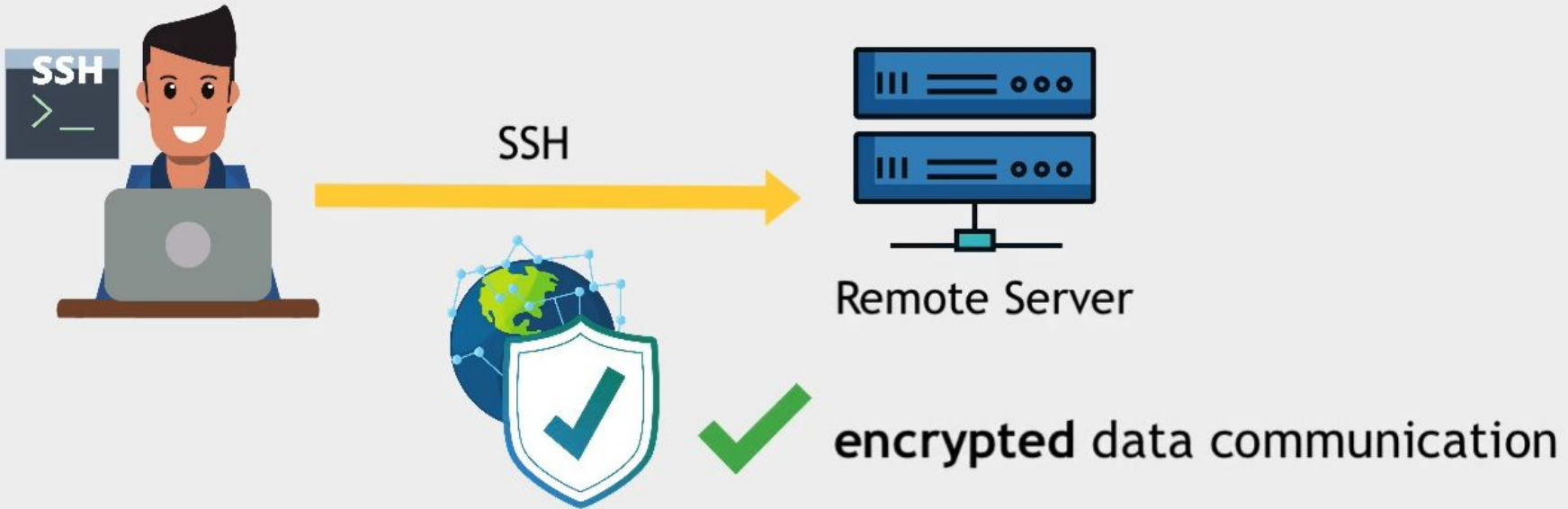
Networking Commands

ifconfig	# ip information, subnet
netstat	# active network connections
ps aux	#Running processes
nslookup	#nslookup google.com # can get ipaddress of any domain name
ping	# ping google.com # service on a application is accessible or not

Introduction to SSH

SSH - Secure Shell

- SSH is a network protocol that gives users a **secure way to access a computer over the internet**
- SSH refers also to the suite of utilities that implement that protocol

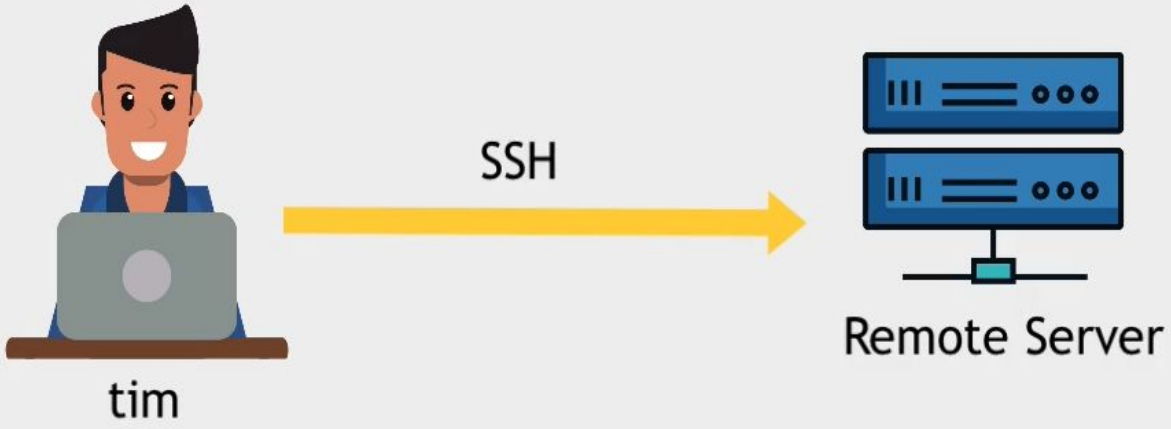


Password Authentication

2 ways to authenticate:

1) Username & Password

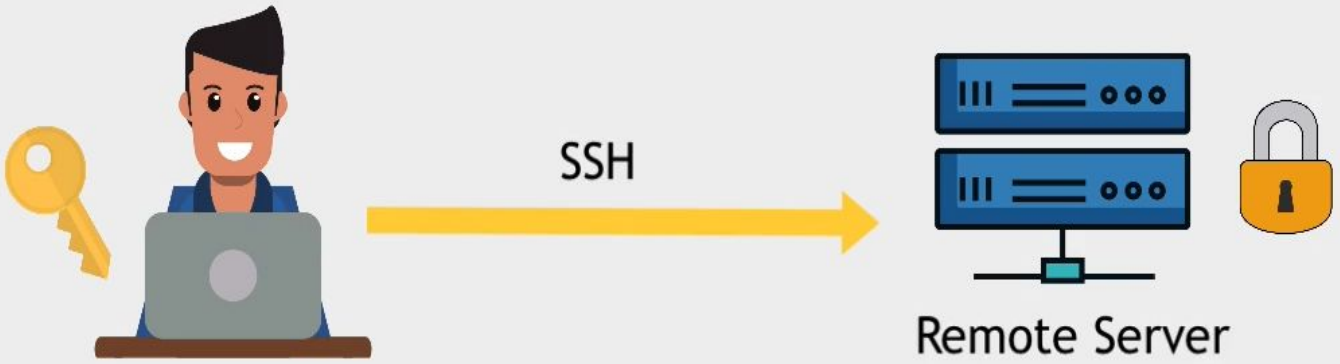
2) SSH Key Pair



SSH Keys Authentication

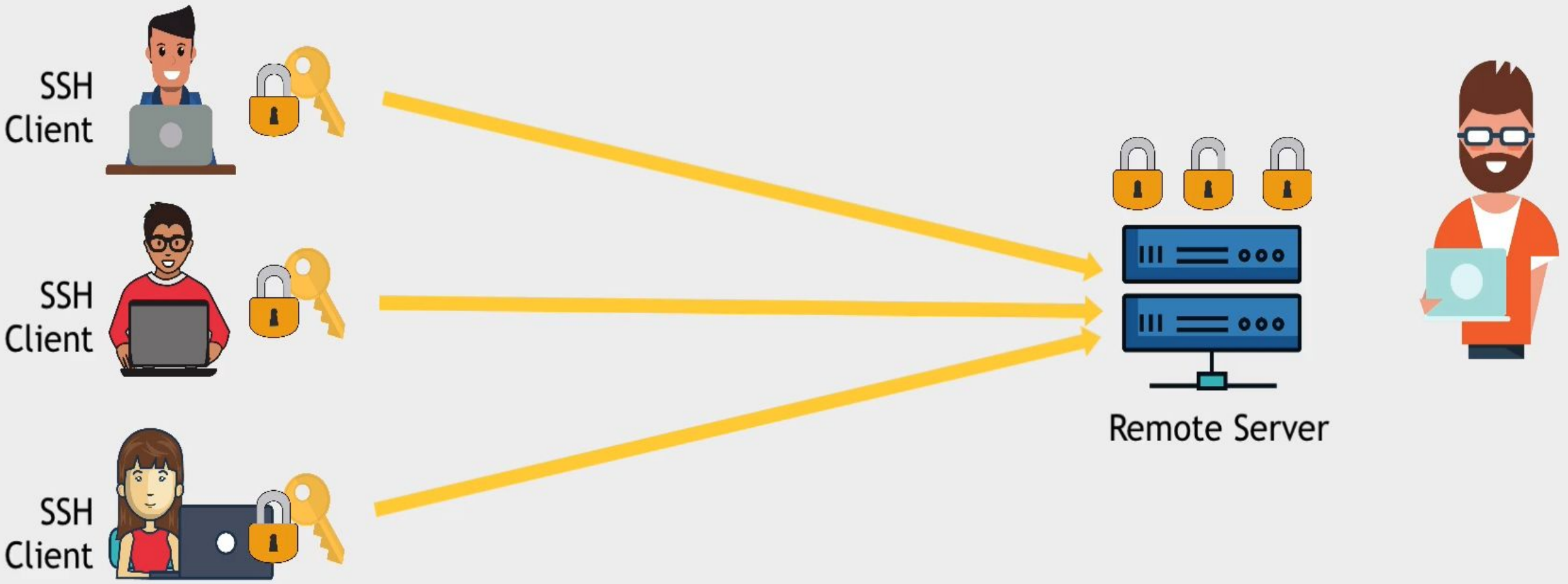
2) SSH Key Pair

- Client creates an SSH Key Pair **Key Pair = Private Key + Public Key**
- **Private Key** = Secret key. Is stored securely on the client machine
- **Public Key** = Public. Can be shared, e.g. with the remote server



- Client can "unlock" the public key with his private key

SSH Keys Authentication



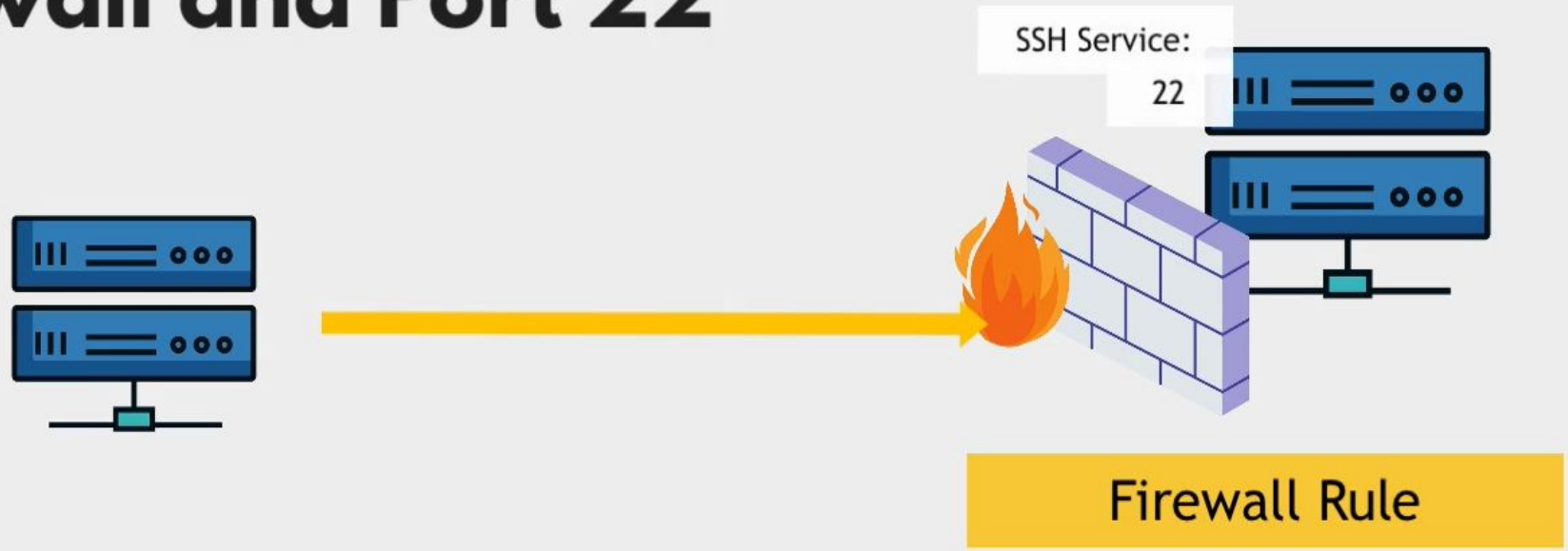
✓ If public key of a person is not registered on the remote server, he/she CANNOT connect to it

SSH for Services

- Services, like Jenkins often need to connect to another server via SSH

- Create Jenkins User on application server
- Create SSH Key Pair on Jenkins server
- Add public SSH key to authorized_keys on application server

Firewall and Port 22



- SSH Service runs by default on machine
- By default, SSH Server listens on **port 22**

- SSH is powerful and needs to be **restricted to specific IP addresses**