DevOps Training

Caio Volpato

Topics

- Day 1:
  - Living in the shell
  - Managing systemd services
  - Scheduled tasks
- Day 2:
  - Network concepts and commands
  - Firewall: Concepts and how to set up
  - How to create your own Debian package to deploy your app faster
- Day 3:
  - Security best practices
  - Monitoring
  - Disk failures and Backup solutions
- Day 4:
  - Docker
- Day 5:
  - Ansible
  - Final words, how to get help, great communities and resources
  - Bonus: Raspberry pi tips and how to create an onion service for your app

Introduction:

Source:
instagram:@mateuslinux

What's DevOps?

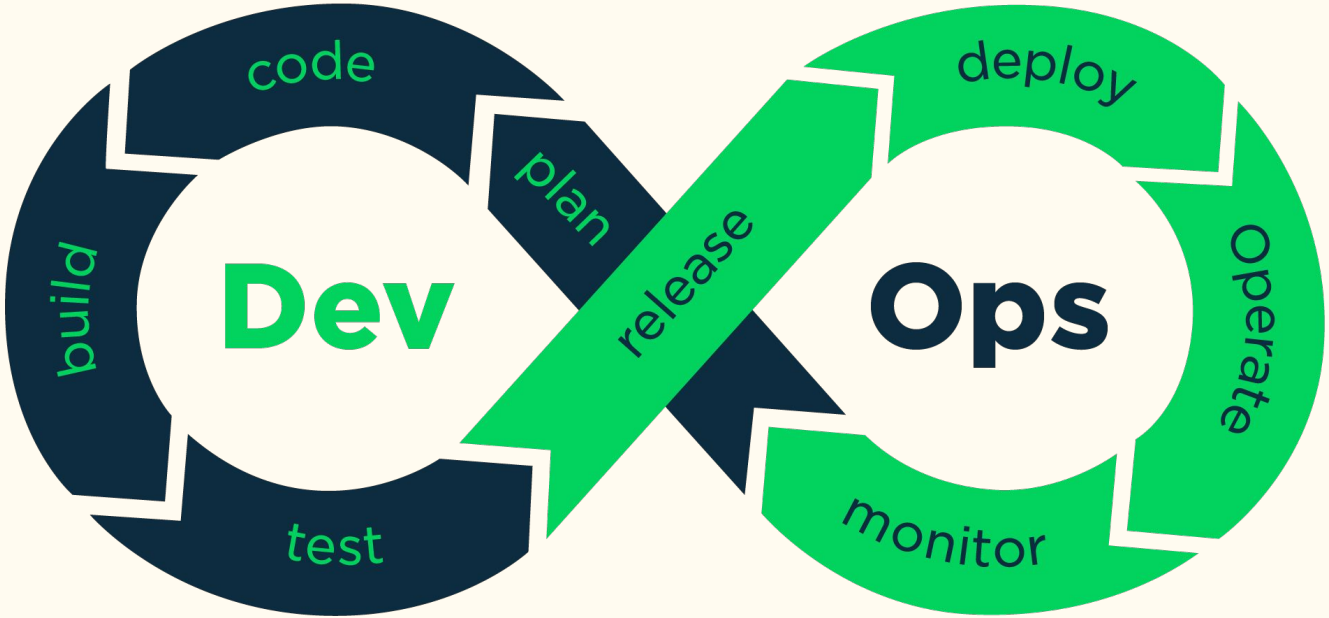
What is not DevOps? In the previous model, development and operations here totally different teams, so ops team is responsible for all the environments of the company and if anything goes wrong they will be blamed.

That's why there's so much bureaucracy, to safeguard the ops team.

DevOps aims to end or at least reduce these barriers, so Dev and Ops work together (and share responsibilities) to make all the software releases more efficient and faster.

DevOps is a culture, not a role!

DevOps diagram



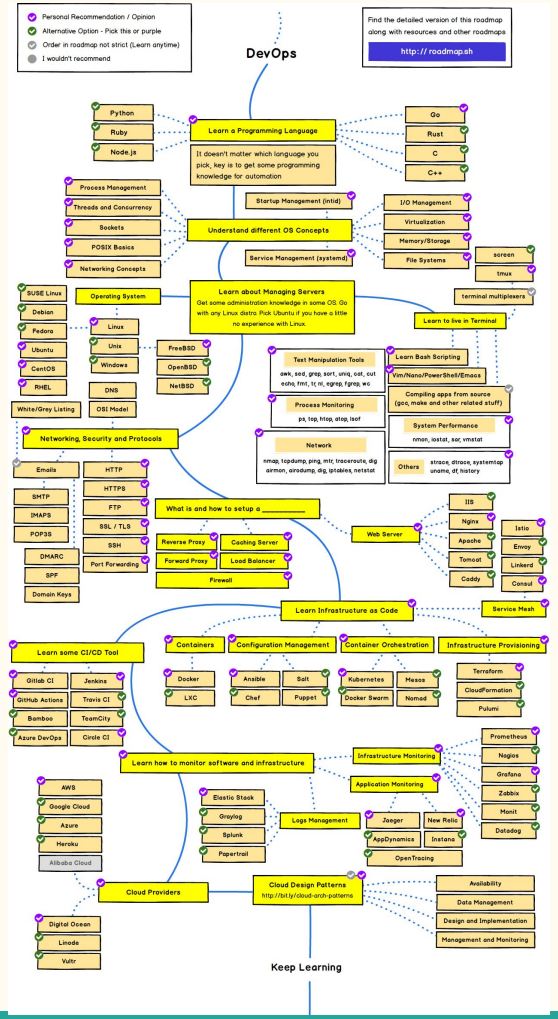
DevOps salaries

Looking for salaries in São Paulo city, as Oct/2019 according to glassdoor website in average DevOps salaries are 20% bigger than developers.

Great resources:

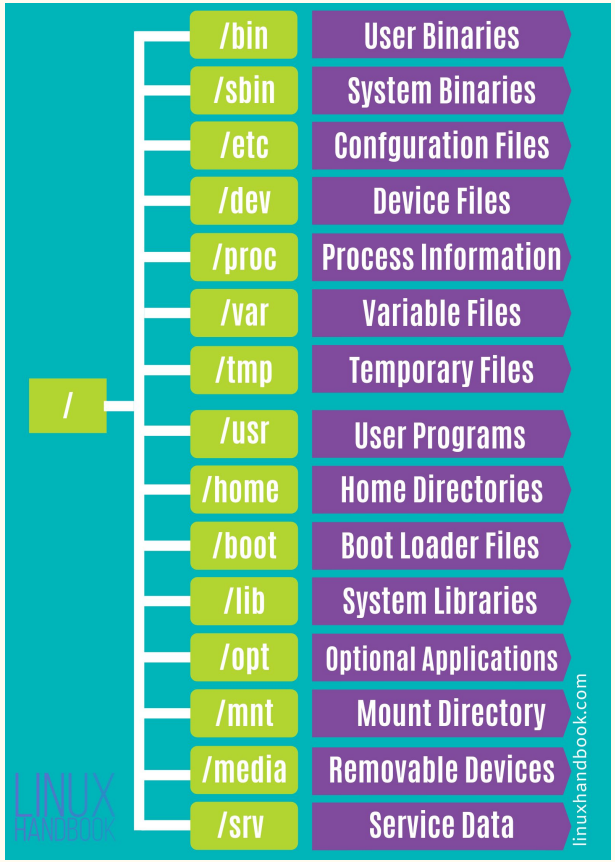
- (PT-BR) DevOps e Transformação Digital
- Patrick Debois - Trust me, we're doing DevSecOps | #FiqueEmCasaConf

Roadmap



https://roadmap.sh/devops

Living in the shell: Directory Structure



https://linuxhandbook.com/linux-directory-structure/

Living in the shell: $PATH

When typing an command in the shell, like ls, But how does the shell know where the binaries are? Does it search the whole filesystem?

The $PATH (environment) variable is an array of the directories where the shell should look for binaries.

- What's current $PATH? `echo $PATH`
- How to change $PATH: put something like:

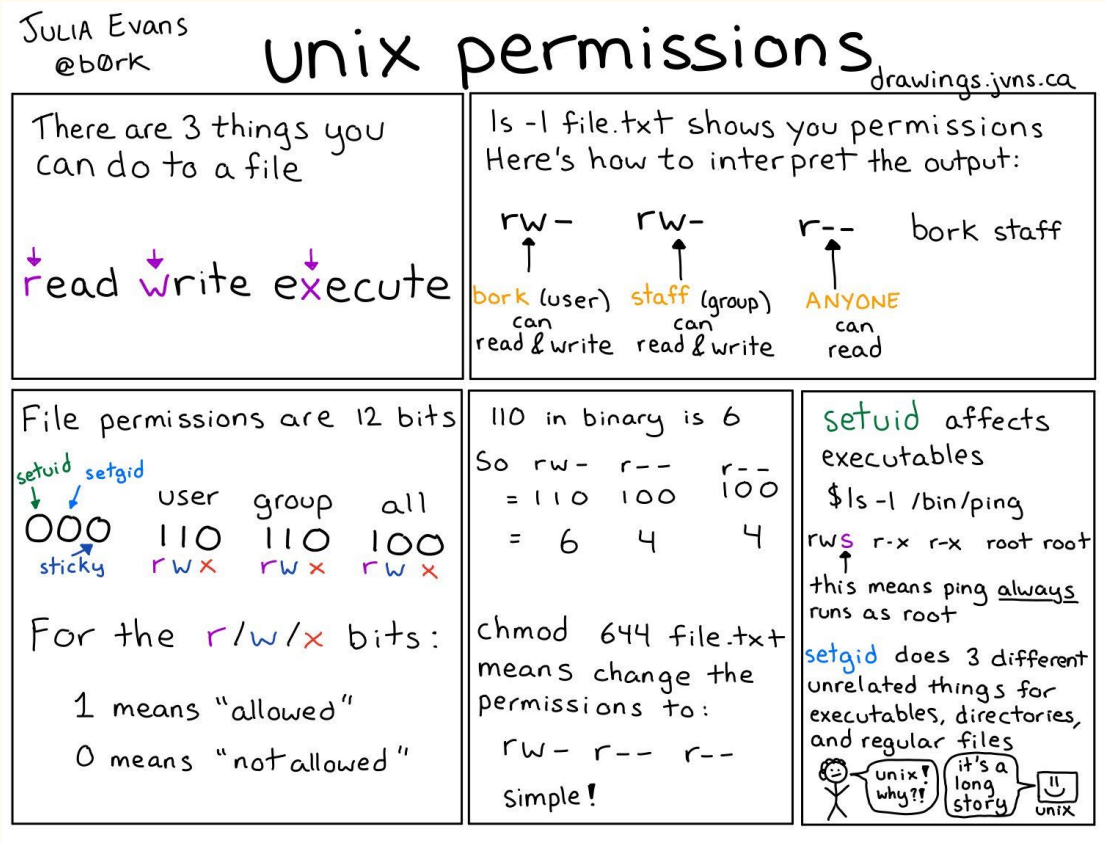
`export  PATH="/folder/path/:$PATH"`, on your `~/.bashrc` (or your shell rc) and

`source ~/.bashrc` (or restart the shell).

Explanation: this will set the PATH, putting /folder/path at the beginning and pretending the "old" $PATH

- Where the binaries are (ie ls)? `which ls` or `whereis ls`

Living in the shell: file permissions



For directories:

r = list files

w = create files

x = access & modify files

Source: @b0rg

Living in the shell: file permissions

- Changing file(s) permissions: chmod 755 file (-R folder recursively in the folder)(755 means: read+write+execute for owner and read+execute for group and others)
- Changing files owner: chown user:group file
- Display file/folder status: stat file
- Change file attributes: chattr
- File/Folder creation mask: umask, since the umask sets an mask, it's "inverted" from the usual chmod, for instance: umask 177 means chmod 600 = 777-177.

Living in the shell: man pages

In order to lookup manuals for commands, there's some options:

- Manpages: commands manual pages, example: to look ls command manual:
  - man ls
- Info pages: info man alternative
- http://cheat.sh/: look for command with example, it can be used directly, example:
  - curl cheat.sh/tar
- https://explainshell.com/: given an command/oneliner it explains what it does.

Living in the shell: text editors

Living in the shell: text editors

When using the shell, sometimes editing some files is needed:

- Nano: easier to use
- Vim: Popular option: vimtutor gives an nice tutorial.
  - https://www.openvim.com/: great vim tutorial.

Living in the shell: screen multiplexes

Sometimes we need to leave some command running and switch between them.

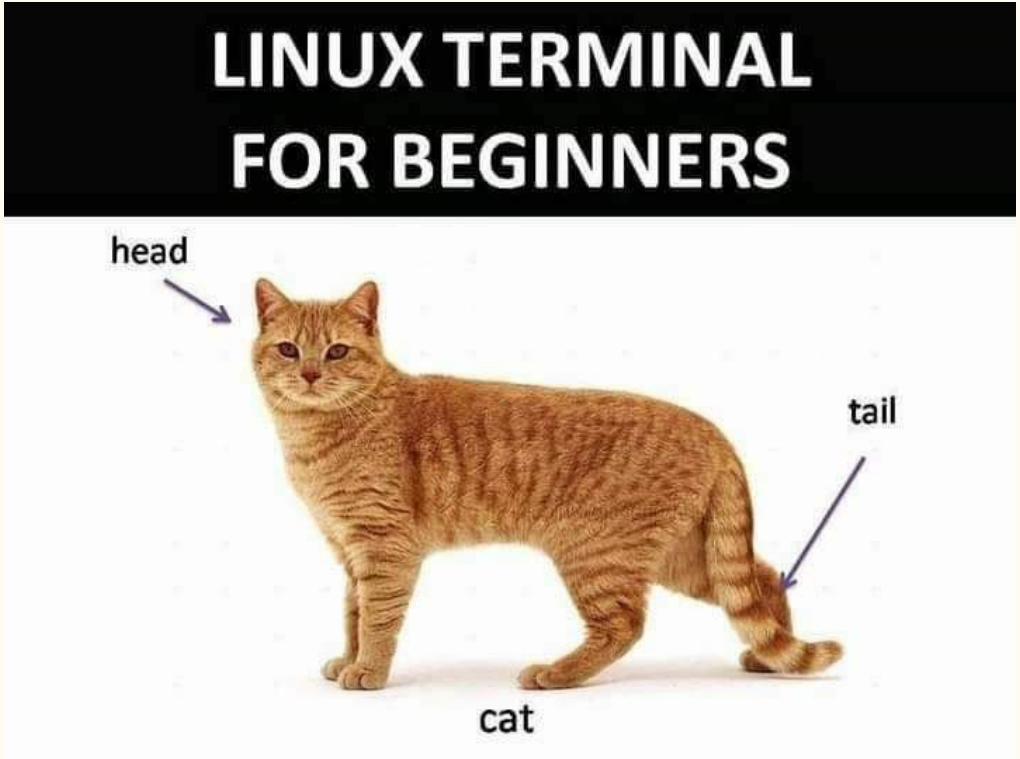
- screen: easier to use.
- tmux: more complete.
  - tmux shortcuts & cheatsheet

Living in the shell: process monitoring

When we need to monitor processes, there's some options:

- ps: list processes, typical use: ps -ef , or ps aux
- lsof: list open files.
- htop (my favorite)
- bashtop (more complete htop, with network, temperature, disks)

Living in the shell: text manipulation



source

Living in the shell: text manipulation

Sometimes we have some text files and we want to manipulate them:

- cat/less/head/tail: read the file
- wc: count how many lines the file has.
- grep: Global search a Regular Expression and Print
- awk: script language for text processing.
- sed : stream editor
- cut: removes parts of each line
- sort, uniq: sorts an file, counts unique occurences.

Living in the shell: tips and tricks

- hostnamectl: discover everything about the machine: what distro if its a vm ...
- Script / scriptreplay: record your terminal session, so you can lookup later.
- Wormhole and onionshare: awesome ways for send files
- Watch: run a command every x seconds
- Know your shell (bash): HISTTIMEFORMAT and HISTCONTROL
- Fish shell: the friendly interactive shell
- tmate : Instant terminal sharing
- Shortcuts:
  - Control+Arrow_keys to skip args
  - Control + l: clear the screen
  - Control + r : search in the history
  - Control +a , control +e: go to the start/end of the line.

Living in the shell: tips and tricks

- loops:
  - **for i in *.png; do echo $i; done**
- Disk space commands:
  - du: estimate file space usage (used on an folder)
  - df: report file system disk space usage (used for the whole disk)
  - tree: list contents of directories in a tree-like format
  - Ncdu: interactive program of du, allow navigating in the files and deleting files.
- Find: search for files in a directory hierarchy
  - find /path/to/something -mtime +30 -type f -exec rm -fv {} \;
  - Will delete all files older than 30 days.
- Magical braces:
  - convert file.{jpg,png}: expands to convert file.jpg file.png
- $(command): gives output of command:
  - echo today is: $(date -I), prints out: today is: 2020-09-22
- xargs: build and execute command lines from standard input

Living in the shell: dealing with failures

Return codes (or exit codes) is the code returned to the parent process by the executable, 0 means success, any other code means anything else.

How to get the return code? In the shell it's in **$?** Variable.

For example: `true; echo $?` Also try `false; echo $?`

Alternatively in using if based on $? value, use the logical operators:

- `command1 && command2`: will only run command2 if command1 succeed
- `Command1 || command2`: Opposite of &&, only run not succeed
- `Command1 & command2`: It will run both at the same time;
- `command1 ; command2`: like an new line in script, run command2 after command1

Living in the shell: sudo

Some operations can only be run by root user, but if other users need to run those operation we would need to give that person the root user password?

Sudo is an alternative approach: It delegates the authority to desired users to run some commands.

You can configure that users and what commands via the `sudo visudo` command.

But by default there's an sudo group that is configured to be allowed to run all commands, so just add the user to the sudo group: `sudo usermod -aG sudo user`

Living in the shell: Managing systemd services

All commands below need to be issued via sudo

- systemctl status [--all] [service]
  - Lists all running services, or if an specific service it shows it status
- systemctl start/stop, enable/disable/mask service
  - enable/disable sets if the service will be started at boot
  - mask is an stronger disable, it prevents from been started
- systemctl restart/reload service
  - reload will just send an signal (usually SIGHUP) so the program reloads it’s conf, without downtime.

Living in the shell: Managing systemd services

We can use journalctl to look for service logs:

- journalctl
- journalctl -u service --since 2019-11-25 [--no-pager]

Note: Some distros (like debian) does not keep journals from previous boots: Here's how to keep persistent it: gist

autossh example: Reverse proxy/shell

What we want to accomplish? Imagine you have an raspberry pi at home and you want to access it from the internet, but not all consumer ISPs allows port forwarding. So we will “connect” ssh in the pi to an computer with an public IP like an VPS. **This is called a reverse proxy/shell.**

There are some services that do that like ngrok and pagekite (open source), but we are going to just use ssh to create the tunnel, making it more generic.

autossh example: The tunnel it self:

Running this in the pi:

```
ssh -R 7654:localhost:22 username@vps
```

Will create an ssh tunnel, “connecting” the port 22 on the pi to the 7654 in the VPS

If you want to tunnel in the reverse direction, replace the -R with -L meaning it will connect the port in the VPS into the pi

autossh example: set up the service

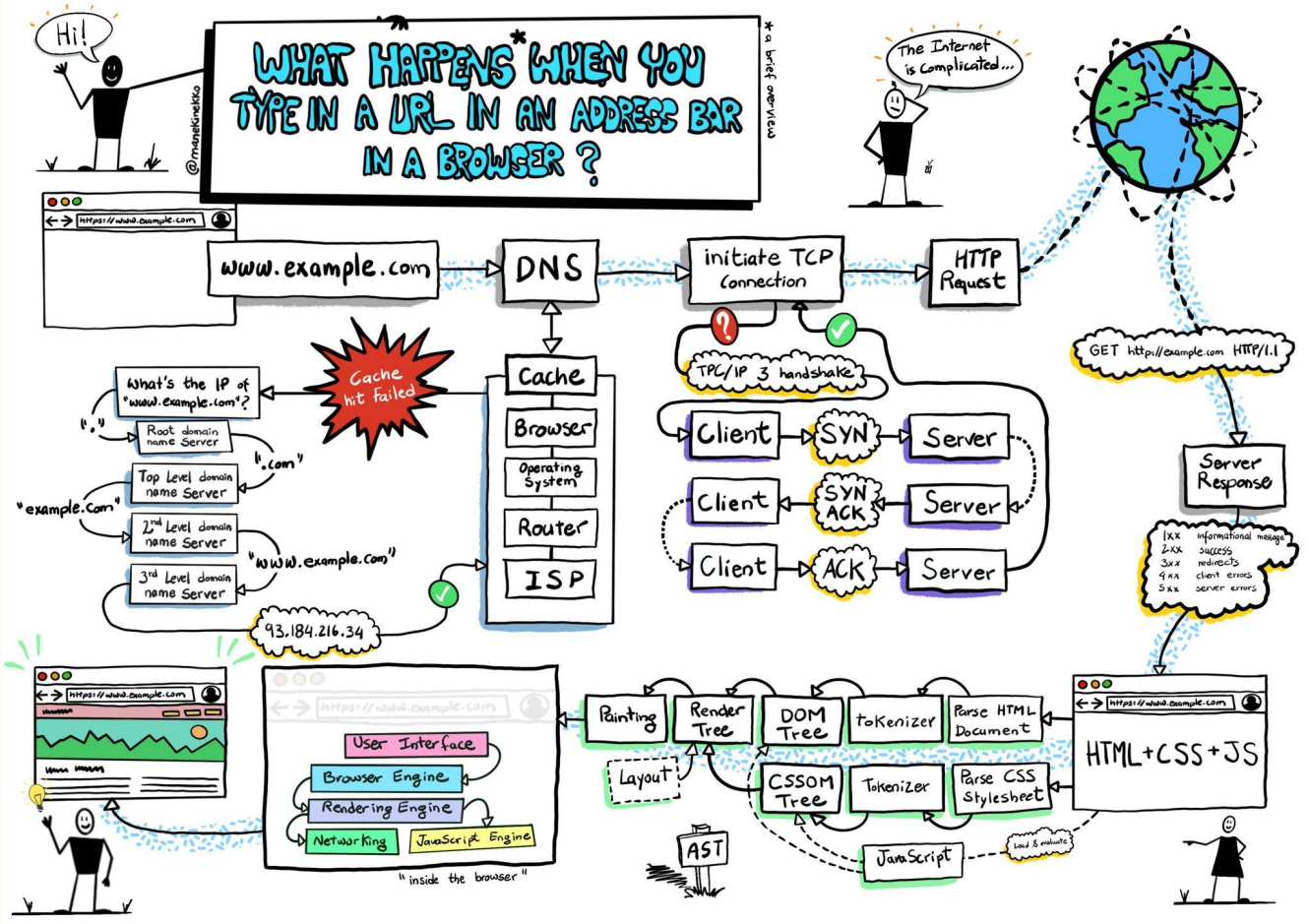
- Edit the file autossh.service, test if running the ExecStart works
- Copy it to /etc/systemd/system
- Reload systemd: sudo systemctl daemon-reload
- Enable the service: sudo systemctl enable autossh
- Start it: sudo systemctl start autossh
- Test it: ssh -J vps localhost -p 7654
  - -J will ProxyJump meaning it will proxy the connection to the VPS
  - It's the same as: ssh vps ssh localhost -p 1234

Scheduled tasks

If you want to run some tasks periodically/at certain time:

- Most important: don't forget to create **lock files**, so if your task takes longer than expected, and will start another one and so on ... taking a lot of resources...; Use the `flock` command
- Use cron, using `crontab -e` and put the cron expression followed put the task, example:
  - 5 8 * * * some_command; will run some_command daily at 8:05AM
  - crontab.guru: helps with the cron syntax
  - /etc/cron.d/file you can put an crontab directly (useful for automated scripts)
  - /etc/cron.{hourly,daily,weekly,monthly}/ are folders that run with that respective frequency: just put your script file there.
- systemd timers: systemd has timers so can schedule running systemd services, checkout the arch wiki article on systemd timers

Network



Source: @manekinekko

Living in the shell: network commands

- ss/netstat: Show network connections (netstat is being deprecated)
  - `sudo ss -tunalp` : Will print all connections
- ifconfig/ip: Configure network interfaces:
  - `ip addr` Will print all interfaces and it's addresses (similar to ifconfig)
- tcpdump: dump traffic (**with great powers come great responsibilities**)
- iftop: display bandwidth usage
- netcat (nc): TCP/IP swiss army knife (like an low level curl)
- dig/nslookup: DNS lookup
  - On linux DNS is set on `/etc/resolv.conf` file, but it's often overwritten by DHCP.
- vnstat: network traffic monitor
- nmap: Network exploration tool and security / port scanner
- arp-scan: ARP scanner: it scans an entire subnet in seconds.

Firewall: ufw

An firewall monitors the packages and based on an set of rules it takes some action.

There are some firewall options:

- iptables (most tradicional)
  - Since using iptables directly is hard there are some front ends such ufw.
- nftables: It modernizes iptables, and aims to replace it.
- OpenBSD PF: An common firewall in some BSD distributions.

After issuing your firewall rules, keep your current SSH connection open and open a new SSH connection to make sure you won't get locked out.

Firewall: ufw: Using it:

- Install: sudo apt install ufw (there's an GUI called gufw)
- Set the default policy: If an package doesn't have an rule for it, it will apply this
  - sudo ufw default deny incoming (default)
  - sudo ufw default allow outgoing (default)
- Add the rules:
  - sudo ufw allow ssh
    - ssh is and "ufw app" so there are some predefined apps/builtins.
    - Allow is desired action, it could be deny (which drops) or reject (let's clients know it's denied).
    - Another great action is limit: it rate limits it avoid brute forcing.
  - ufw insert 1 allow from **15.15.15.0/24** to any port 80**/tcp**
    - Only allows to this IP/subnet at port 80, only TCP.
    - Insert 1: Rules order matter: this will insert this rule at top.
- Enable it: sudo ufw enable

Package managers

- Package managers are like “App Stores” it allows to install/upgrade/remove software.
- This way your VLC is vulnerable, you won’t need to go to the website download the installer and upgrade it, it manages all the programs in your computer.

Common actions: (using apt: works on Debian based systems)

- apt update: download package information from all configured sources
- apt upgrade: install available upgrades of all packages currently installed
  - apt full-upgrade: will remove the package if needed to upgrade it
- apt install/remove/purge pkg_name
  - Purge will remove the package and all files it generated (usually configuration file you don’t want to delete)
- apt search/info pkg_name: search for pkg, print pkg info
- apt list [--installed]: List installed packages (awesome for backup)
- dpkg -S /path/to/file : Show which package a file belongs to.

Example: creating our own Debian package

An Debian package consists of 3 parts:

- An control file inside the DEBIAN folder: where you can define some information about the package (like the name, version, homepage ...) and it's **dependencies**.
- (optional) {post,pre}{inst,rm}: scripts files inside the DEBIAN folder that will be run after/before the installation/removal of the package.
- Your package files: your package folders is like the root the of the filesystem and all the files you put there will be extracted.

After all of those parts sorted: generate the package: `dpkg-deb --build pkg_folder`

- It's nice to integrate the pkg build process into your CI/CD pipeline.

To install it: `sudo apt install ./pkg_name.deb`

Security best practices

OWASP -- Open Web Application Security Project

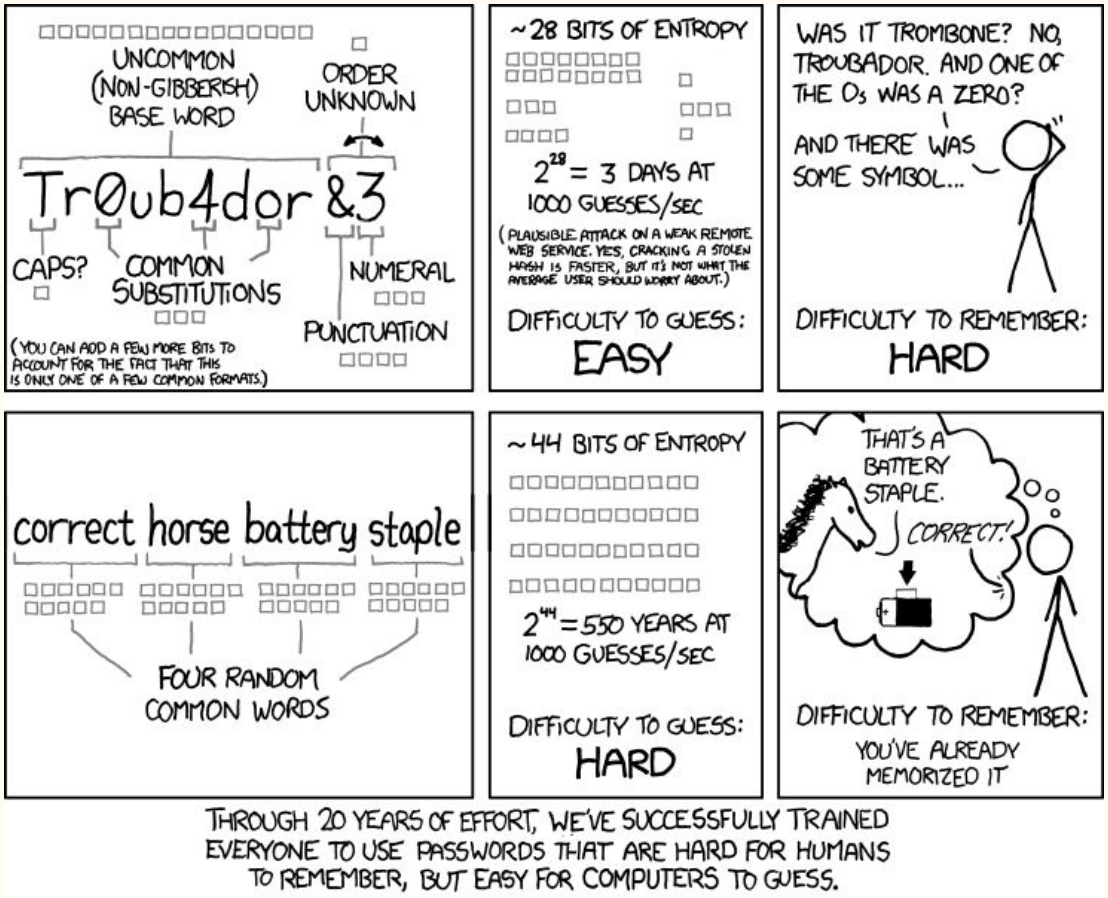
OWASP is an online community that gives a lot of recommendations on making your web application secure.

Recommended reading: Threat Modeling Cheat Sheet

Passwords:

source

Passwords: Relevant XKCD



XKCD #936:

Password Strength

Passwords

- [OC] My text (PT-BR): P@ssw0rds: The weakest link of our security
- Relevant XKCD
- A secure password requires to be: truly random, long enough (14+) and most important easy to remember.
  - Diceware method: english wordlist and PT-br wordlist.
    - Example: panoramic nectar precut smith banana handclap
- Using a well established open source Password manager such as KeePassXC (or lessPass or bitwarden) is highly encouraged, and enable 2FA or even U2F (aka yubikey)
- Passwords requirements and expiring policies are full of shit, and do more harm than good
- New (2017) Nist recommendations should be used.
- When storing the passwords your app should always try to use argon2, and validate the password strength using zxcvbn and verify if it was compromised using for instance the haveibeenpwned.com API

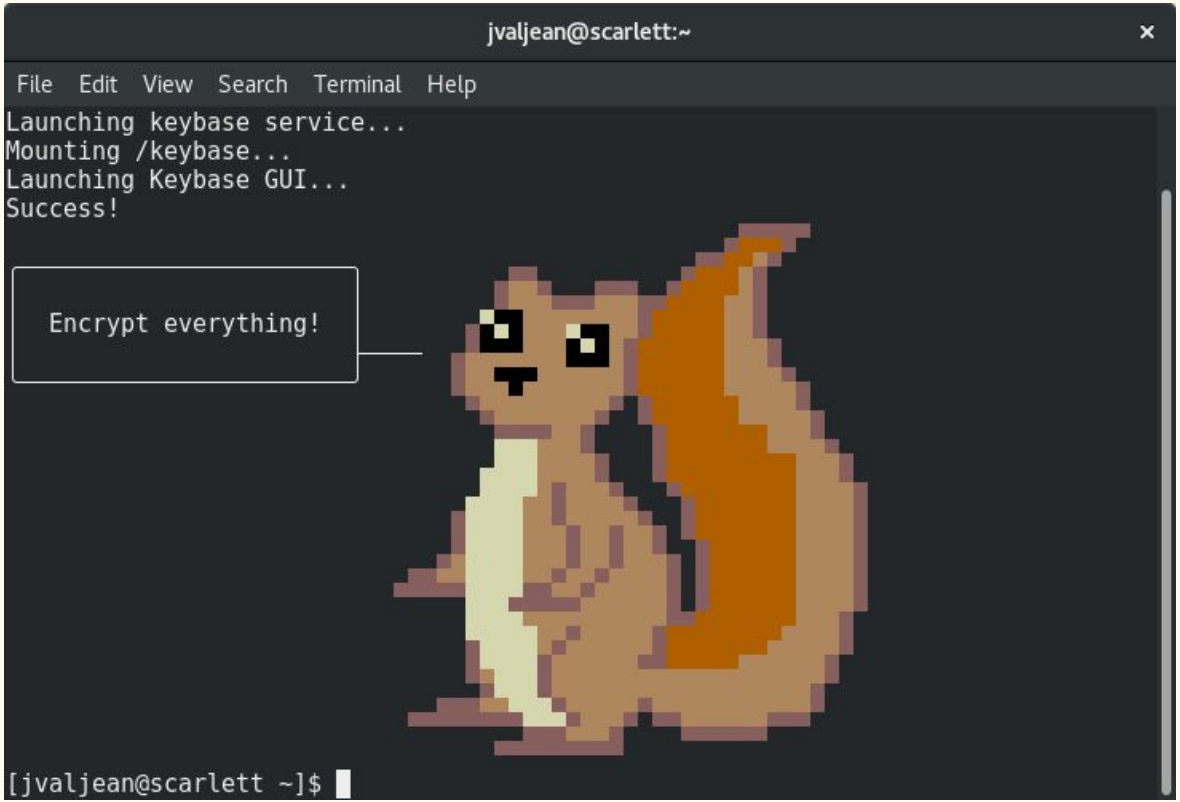
Security updates

- Have you heard these names?
  - Shellshock
  - Heartbleed
  - dirtyCOW
  - KRACK
  - POODLE and logjam
  - Spectre, meltdown and cpu.fail
- Sign Up for your Distros security announcements: Ubuntu security notices , Debian Security announces and your framework security mailing list such: Django announce.
- Unattended upgrades: it's a native mechanism to automatically install updates.
- Install the latest microcode from your CPU vendor and run spectre-meltdown-checker

Security updates: setting up unattended updates

- Install: sudo apt install unattended-upgrades
- Enable: sudo dpkg-reconfigure unattended-upgrades
- Configure: edit /etc/apt/apt.conf.d/50unattended-upgrades
  - Make sure all origins are listed: run apt-cache policy | grep release
  - Read all the file to read all the options, if instance if your server can be rebooted if needed.
- Test it:
  - sudo unattended-upgrade --debug --dry-run
  - Make sure all origins are listed
  - Logs: wait for few days, and see in the logfile: /var/log/unattended-upgrades/unattended-upgrades.log to see if it's running and updating

Disk encryption



Keybase.io: Crypto(graphy) for the masses

Disk encryption

When encrypting the disk there are two schemes: FDE (Full Disk Encryption) or FBE (File Based Encryption) which only encrypts the files, not the whole disk.

In Linux we can use the following tools to encrypt:

- eCryptfs/EncFS: It's easy and can be setup after the distro is installed, it unlocks when the user logins (uses the same password).
- LUKS: It's more tradicional and secure: asks the password at boot.
- Veracrypt: It's great, creates an "volume" to encrypt only the file within that and also works on MacOS and Windows.

Disk encryption: With LUKS

cryptsetup -v -y luksFormat

--type luks2

--pbkdf argon2id

--pbkdf-memory 320000

-i 12000

--use-random

-s 512

-c twofish-xts-plain64

-h sha512

/dev/sda1

Disk encryption: With LUKS

- To unlock:
  - Cryptsetup luksOpen /dev/sda1 volumename
- Lock:
  - umount first, then:
  - Cryptsetup luksClose volumename

SSH best practices

source

SSH best practices

- Good practices:
  - Secure cipherlist
  - rate limit in the firewall or fail2ban
  - pubkey auth only (disable password auth)
  - port knocking: The ssh port is blocked in the firewall, but after an set of port are knocked in the right sequence the port will be open only for the authorized IP and for limited time.
  - 2FA
    - Recent versions of OpenSSH supports using any U2F key as smartcard, both the client and server most support it :(
  - Access only via Onion, this way only you who knows the onion addr can connect.
- SSH tarpit
- Securing your ssh keys:
  - Generate the key in a air gapped machine and only use it in a yubikey
  - QubesOS SSH split mode

HTTPS cipherlist

By default, when your application has HTTPS, some older insecure (or null) cipher is enabled in order to work with old software.

So you need to evaluate and enforce an cipherlist that makes sense to you and it's secure.

Mozilla has an great configuration tool: Mozilla SSL Configuration Generator

Test it with: testssl.sh: /bin/bash based SSL/TLS tester

HTTP Security headers

In order to prevent some attacks suck Cross-Site Scripting (XSS), it's advised to setup some special headers that makes your app more secure.

Goto securityheaders.com and it tests your app and provide some explanation.

Ubuntu vs Debian

Ubuntu is based on Debian, but it grew so much apart that there's a lot of important differences: A newbie's newbie guide to Debian, by Helen Koike

- Debian is a non profit project (unlike Ubuntu and Fedora)
- All Debian volunteers sign the Debian social contract.
- Ubuntu has a spyware/keylogger installed by default (since 2012).
  - And threatened to sue a website which contains instructions do remove the spyware
- Debian is rock solid Stable.
  - Debian requires reboots every ~4 months while ubuntu ~2 weeks.
  - Ubuntu oftens breaks the system when updating (most release upgrade break the system)
- Most Ubuntu mirrors doesn't even have HTTPS (CVE-2019-3462)
  - Debian not only have HTTPS but also onion: onion.debian.org
- Most Debian packages are build reproducible (fundamental for trust and security)
- Debian doesn't require frequent updates (only bugfix and security)
- Ubuntu has a lot of broken packages (such as usbguard, firejail, munin)

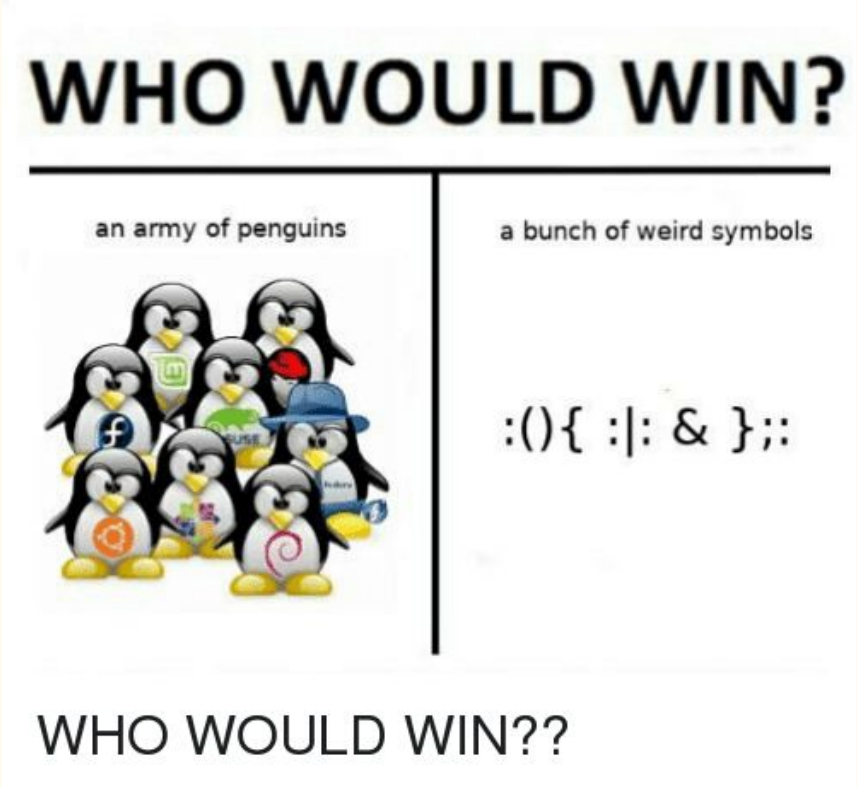
Other “GNU/Linux distros” to look for.

- Tails: The Amnesic Incognito Live System, it’s a security focused distro installed on CDs or flash drives, not leaving any trace on the computer, and routing all traffic via Tor.
- Qubes: A reasonably secure operating system. A system that has a bunch of VMs (using Xen) to compartmentalize all the activities.
  - Security Through Distrusting -- Joanna Rutkowska
- GuixSD: A distro made around the super complete and flexible guix package manager, inspired by NixOS
  - Solving the deployment crisis with GNU Guix
- OpenBSD: Not a Linux distro!!! It’s a BSD security focused system.
  - **In more than 20 years of the project it only had 2 remote holes vulnerabilities in the base install:** Why OpenBSD rocks! Runbsd.info

Great video on OpenBSD:

- An Introduction to OpenBSD is a great video that talks about OpenBSD and it's features and how to manage it.

Monitoring


WHO WOULD WIN?
an army of penguins
a bunch of weird symbols
:(){ :|: & };:
WHO WOULD WIN??

Monitoring

Monitoring consist of an system that monitors computers (or others equipments) to notify administrators to take some action.

- How Regular Expressions and a WAF DoS-ed Cloudflare
  - Official report
- Monitoring solutions: Old school Munin, prometheus and zabbix
- Notifications:
  - Healthchecks.io: every time your scheduled tasks run it pings this service, if an ping does not occur an notification is generated.
  - Uptimerobot: pings some service or port to see if up, if not notifies you.

It's interesting to use 3 services: one intranet (prometheus or zabbix), one to the outside network (healthchecks.io) and one from the outside network (uptimerobot)

Disk failures

Disk failures

In order to avoid data loss due hardware failure is highly advised to run badblocks in order to identify disk badblocks, even for new drives.

- Smartctl: S.M.A.R.T.: Self-Monitoring, Analysis and Reporting Technology
  - sudo apt install smartmontools
  - `sudo smartctl -l long /dev/sda`
  - `smartctl -H /dev/sda`
- fsck every boot: `tune2fs -c 1 /dev/sda1`
  - **`adjust sda1 , use lsblk command`**
- fsck with badblocks: `fsck.ext4 -vcck /dev/sda1`
  - `-cc that runs badblocks (read-write) it's non-destructive`
- Watchout for io latency in the monitoring
- New Hard Drive rituals
- Scrutiny is an dashboard for SMART monitoring, with real world data for failures based on the disk model.

Disk failures: Backup Solutions

3-2-1 Rule:

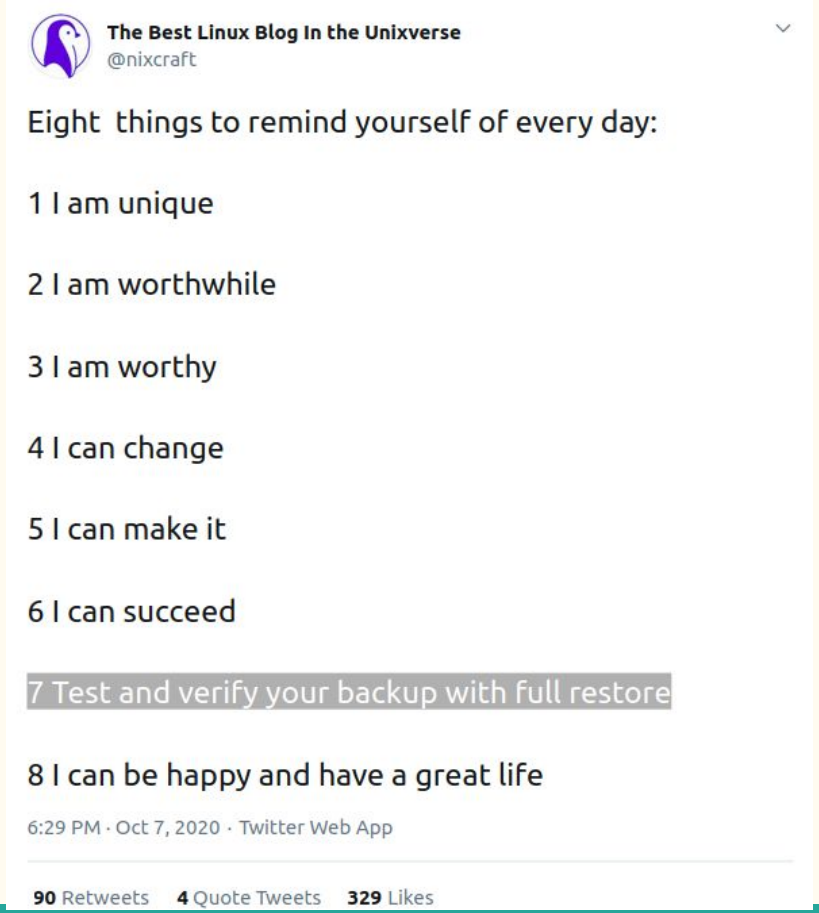
3: Your backup should have **3 copies** (1 working +2 backups).

2: Must have 2 types of media (**RAID does not counts as an backup**).

1: One offsite backup, in case of an fire or similar disasters.

The main idea of this rule is: In case of a problem in the working computer we have an fast, easy local backup and in case of disaster we have an copy to restore the data.

Disk failures: Backup Solutions

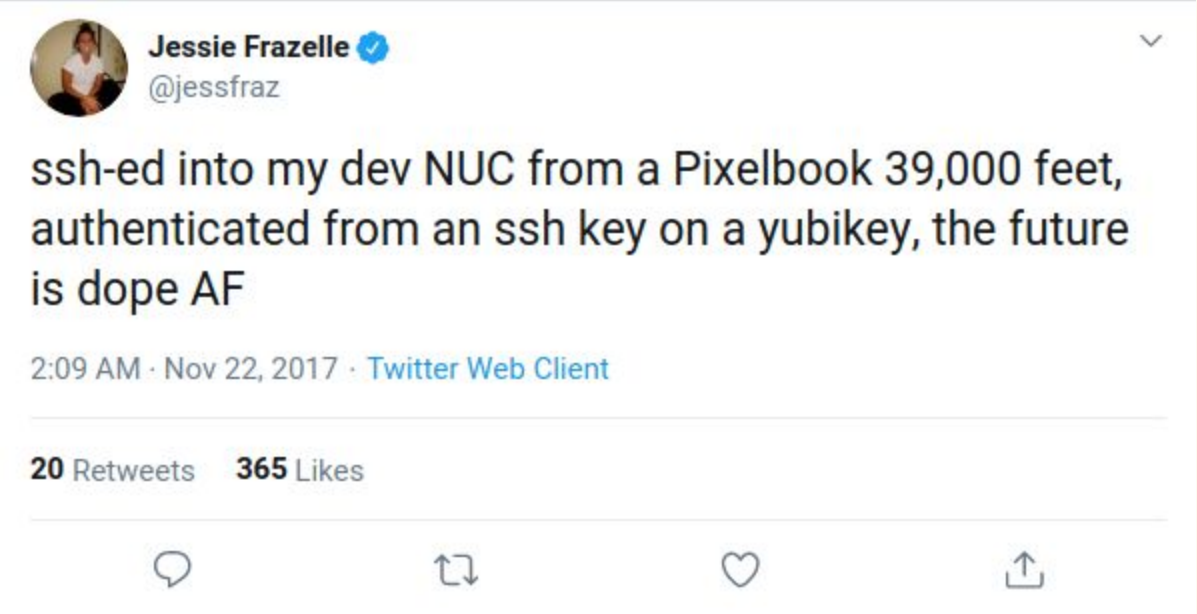


source

Disk failures: Backup Solutions

- [OC] My text (PT-BR): How I stopped worrying and started loving my backup system
- rsync: old school tool that creates **incremental** backups.
- Borgbackup (borg): **Holy grail of the backups**
  - All backups are full backups (not incremental).
  - Compressed and super space efficient (it only wastes space where the data changed).
  - It's possible to encrypt with authentication.
  - Append-only mode allow the data only to appended, not overwrite, providing protection against ransomware.

Reliable remote only access principle



source

Reliable remote only access principle

We need to be able to access all the machines reliably and without the need of physical intervention. In order to do that we can:

- Cool project: mosh mobile shell: Better “ssh” for bad connections
- Remotely unlock encrypted disks: so we don't need to type the password into the keyboard (dropbear-initramfs package).
- We need an machine to run skype:
  - Virtual screen: we don’t need a physical monitor to control the screen (xserver-xorg-video-dummy).
  - VNC server: So we can access the virtual screen remotely.
  - (GDM) Autologin so the user password is not prompted
  - Seahorse: Disable password to encrypt the vault, so no password is asked
  - Desktop autostart: skype autostart after boot.
  - Firejail: we can run 2 skypes at the same time, with 2 different accounts. (/home is mounted as different dir)

Docker: Why containers?

“It works on my machine”

The main problem docker was trying to solve was, making **reproducible** and **isolated** dev environments.

This way with docker deploying an software is easy and effortlessly and is very lightweight when compared with an VM.

What containers are made from?

Containers == Processes

Containers are just “isolated” and “restricted” processes in the host.

Containers aren’t magic: Checkout this script that manually starts an container:

What makes then possible are features available on the Linux kernel, the main ones are:

- Namespaces ~ What the container can see
- Cgroups ~ What the container can do

docker run hello-world

1. This will fetch the “hello-world” image from docker hub (because it’s not available locally)
2. Create a new container from that image and run the binary that produces the “hello world” output

Running containers:

`docker run -p 80:80 nginx:1.19.2`

This will run the nginx web server image (version 1.19.2), and “publish” port 80 to the host, so it’s accessible from the host and the outside world.

Cool options to add:

- --name webhost: gives and friendly name, not an random name like (busy_newton)
- -d: detaches the containers so it will will run at background.
- Is this an disposable container? --rm will remove the container when exits.
- When using -d how do I can access its output?
  - `docker logs [-f] webhost`
- Stop the container: `docker stop/kill webhost`
- Start it again: `docker start webhost`

How to “ssh” into an container?

But it need to change something in the running container?

Since containers are very minimal, most likely it will not have an ssh server so you can access it.

So we need to spawn a new shell (like bash) inside the running container:

```
docker exec -it container_name bash
```

If you want to create a new container and open an shell:

```
docker run -it image bash
```

Note: Some images, like alpine based doesn’t come with bash, so replace it with sh

Docker: Persistent data

I'm running an database, or my app need to store some data, What to do?

- Named volumes: it's best when it's not critical stuff (ephemeral data), docker will create an volume and give an name.
  - docker run -e MYSQL_ROOT_PASSWORD=mypass **-v mariadb_vol:/var/lib/mysql** mariadb:10.5.5
  - List the volumes and required space: docker system df -v
- Bind-mount: Use this if the data is critical and you need to back it up. Will bind-mount and folder in the container into a folder in the host, be careful with permissions, it will have the permissions of the container (use the --user on run to override this)
  - mkdir mariadb_files
  - docker run -e MYSQL_ROOT_PASSWORD=mypass **-v $PWD/mariadb_files/:/var/lib/mysql** mariadb:10.5.5

Dockerfile: Creating your app docker image

The Dockerfile defines how to build the docker image for your app.

1. What is the image base for your docker file? Search in docker hub for the language your app uses, or some distro like debian and use that as your base image (`FROM baseimage:version`).
2. What does your app requires to run? Install your dependencies (`RUN pip install ...`) , copy your source code (`COPY . .`) and set your env variable (`ENV VAR=Value`)
3. Your app needs an port open to be accessed? (`EXPOSE 5000`)
4. Your app needs to save some files persistently? (`VOLUME app/data`)
5. Final command to run your app (`CMD python app.py`)

Done! Built it and run

Dockerfile: Building and running your image

Build:

docker build -t **app_name:version** -f CustomDockerfile .

Run:

docker run -p 8888:5000 **app_name:version**

You can upload your app image to docker hub, by docker login and

docker push **your_username/app_name:version**

Docker example: Flask app

In order to illustrate how everything works, I made an basic flask API (python):

gitlab.com/caioau/flask-demo

- App itself and what it does
- Unittests and pytest
- Gitlab-ci
- Dockerfile and it’s security, how to build it and run
- Health Check in Dockerfile

All of that is written in the README of the repo

Container security 📦💥

@IanColdwater

Containers security 📦💥

There's awesome checklist to make sure your security is fine when using docker:

- By default, when not started with --privileged option, running your app is more secure than running in the host or VM. That's because by default docker limits what an container can do (via seccomp, limiting capabilities and apparmor/selinux)
- Don't run apps in containers as root (USER in the dockerfile)
- Use Slimmed-Down Base Images (less stuff is less possible vulnerabilities)
- Run docker-bench-security to check how is your security
- If you are really worried, run docker as rootless

Docker goodies

- Traefik: Is an edge router making easy and simple to publish your services.
- Selenoid: Does your app uses selenium? With selenoid is possible to run browsers on docker
    - Checkout this awesome video: (PT-BR) Selenium com Python #15 - Selenium Docker it's part of an selenium course with python

Docker: note on alpine images

There's an distro called alpine which really popular because of its size, it takes ~5MB

While using alpine based images is tempting, it comes with some limitations:

- It does not use glibc as libc, so python libs needs to be compiled during pip install:

Base image	Time to build	Image size
python:3.8-slim	30 seconds	363MB
python:3.8-alpine	1557 seconds	851MB

Source:Using Alpine can make Python Docker builds 50× slower

- Doesn't come with apt so your will need to migrate all your scripts to apk

Debian-slim is already reasonably small (~70 MB), and since docker stacks the images this space is only used once.

IMHO: Converting to alpine based image should be the least of your priorities.

Docker great resources

- Udemy course: Docker Mastery: The Complete Toolset From a Docker Captain
- Docker Curriculum: A Docker Tutorial for Beginners
- Play with Docker Classroom: Official tutorials on docker within the browser
- Talk: Jérôme Petazzoni - Creating Optimized Images for Docker and Kubernetes: How to create images using the nix package manager

Vagrant

Vagrant is an tool to easily (re-)create your environment, by provisioning VMs.

To use it:

1. Find your distro your environment is based on: vagrant boxes
2. Create your Vagrantfile: `vagrant init debian/buster64`
3. (Optional) You can edit your Vagrantfile to automatically provision your machine.
4. Start the VM: `vagrant up`
5. Access it: `vagrant ssh`, `vagrant ssh-config` will print the info about ssh.
6. Made a mess? `vagrant destroy` to throw away your VM.

Ansible: infrastructure as code

Ansible is a powerful server and configuration management tool. It's similar to Chef, Puppet or Saltstack.

The biggest advantage is that ansible is agentless (so the managed machines only need a python interpreter installed, no additional installations), it's super easy to make playbooks/roles it's just a yaml file, and the available modules that do the heavy work.

It can be used to setup a machine to install an application or manage and do all the necessary maintenance in all the servers.

The best way to test your playbook is using vagrant: That creates your environment in a VM, which can be easily re-created or destroyed.

Ansible: Concepts

- Inventory: It’s an file to list all your hosts.
- Modules: Ansible ships with some ready to use modules: Module Index
- Tasks: An single call to an module (example install some pkg).
- Playbook: It’s an set of tasks to accomplish something (install pkg, copy files ...)
- Roles: It’s and set of playbooks to accomplish all the desired provision.

Ansible: Examples

- Webserver
- Cts machine
- FPS update.yml

Ansible: Good reference:

I recommend the book: Ansible for DevOps

Using ansible-lint helps a lot, because it gives good practices and tips for your playbooks.

(PT-BR) There are 2 great posts about ansible from Aecio Pires:

- Primeiros passos com Ansible – Aécio Pires
- ansible containers: github repo with ansible roles that configures prometheus with grafana

Getting help

source

How to get help:

- /r/linux4noobs is great for beginners.
- DigitalOcean has a lot of great tutorials
- A great place deeply understand something is the archlinux wiki

Great communities:

- casahacker HC and LHC HC
- DevOps Campinas slack group
- Unicamp groups:
  - Enigma (cryptography, privacy and security study group)
  - LKCamp (Linux Kernel study group)
  - Archventure Time (Arch Linux group)

What I can do for learning:

- Host a Tor relay or bridge
- Use Linux in your computer, start with an easy distro such as Linux Mint, then when you're confident use Arch Linux or Gentoo.
  - 10 ways Linux is just better!
- Self Hosting is always fun:
  - Nextcloud instance: so you can host your files like dropbox, calendar, contacts, notes and kanban.
  - Plex server: so you can host your media.
  - Ttrss: Best way to burst out your feed bubble: follow anything RSS.
  - Jitsi web: Host your own video conferences.
  - Wireguard vpn
  - awesome-selfhosted list

Awesome resources to follow:

- Youtube:
  - NetworkChuck
  - Chris Titus Tech
  - LearnLinuxTV
  - Level1Linux
  - Luke Smith
  - Quidsup
  - LINUXtips
    - FiqueEmCasaConf
- Media.ccc.de: Chaos Computer conference recorded talks.
- Podcasts:
  - Jupiterbroadcasting: podcast network with dozen podcasts.
  - The Syscast podcast
- Julia Evans: awesome zines to learn about a lot tech things.
- Guiafoca.org: (PT-BR) awesome resource to learn about computers and GNU/Linux

(Bonus) Raspberry pi tips

Because of the SD card hosting anything in pies are risky, so don't ever host anything critical.

Recently on the pi4 it's possible to boot from USB, like an external HDD.

Another approach it to write less stuff to the card, it is not perfect (again don't ever host anything critical on pies):

- Mount the filesystem with the flag noatime
- Disable swap (and maybe use zram?)
- Run fstrim daily (to wear level)
- Mount /var/log, /var/tmp/ as tmpfs

Bonus: “Onionize” your application

Unless your application is a bank system, you can consider to allow your users to access your app via onion, so the users have complete anonymity.

For instance, facebook can be accessed via facebookcorewwwi.onion, surely all the actions that a user does in the site are logged, like the clearnet version, but having an onion can help for instance journalists to access it without their ISP or gov knowing it. It’s accessed by 1 million users monthly as of April/2016 (source).

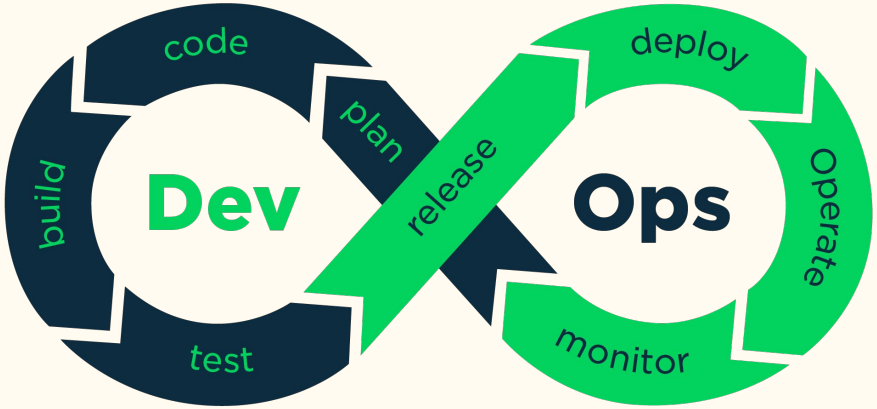
Tor can also help with censorship: Roger Dingledine - The Tor Censorship Arms Race, for instance Most of the popular Brazilian ISPs started blocking safe abortion website

Bonus: “Onionize” your application

Official documentation: https://community.torproject.org/onion-services/

1. You need to install the tor software on your server, an in the torrc create an onionservice to connect to some port in that server. Another option is to use eotk to act like an proxy
2. On your clearnet version put the onion-location header/html meta tag so when using Tor browser the clients will switch to the onion address.

Final words



Let's remind the DevOps diagram, now we know how to:

- How the Linux OS works.
- Operacional tasks such as monitoring and backups
- Release using gitlab-ci
- Deploy using docker and ansible
- Monitor using zabbix or prometheus