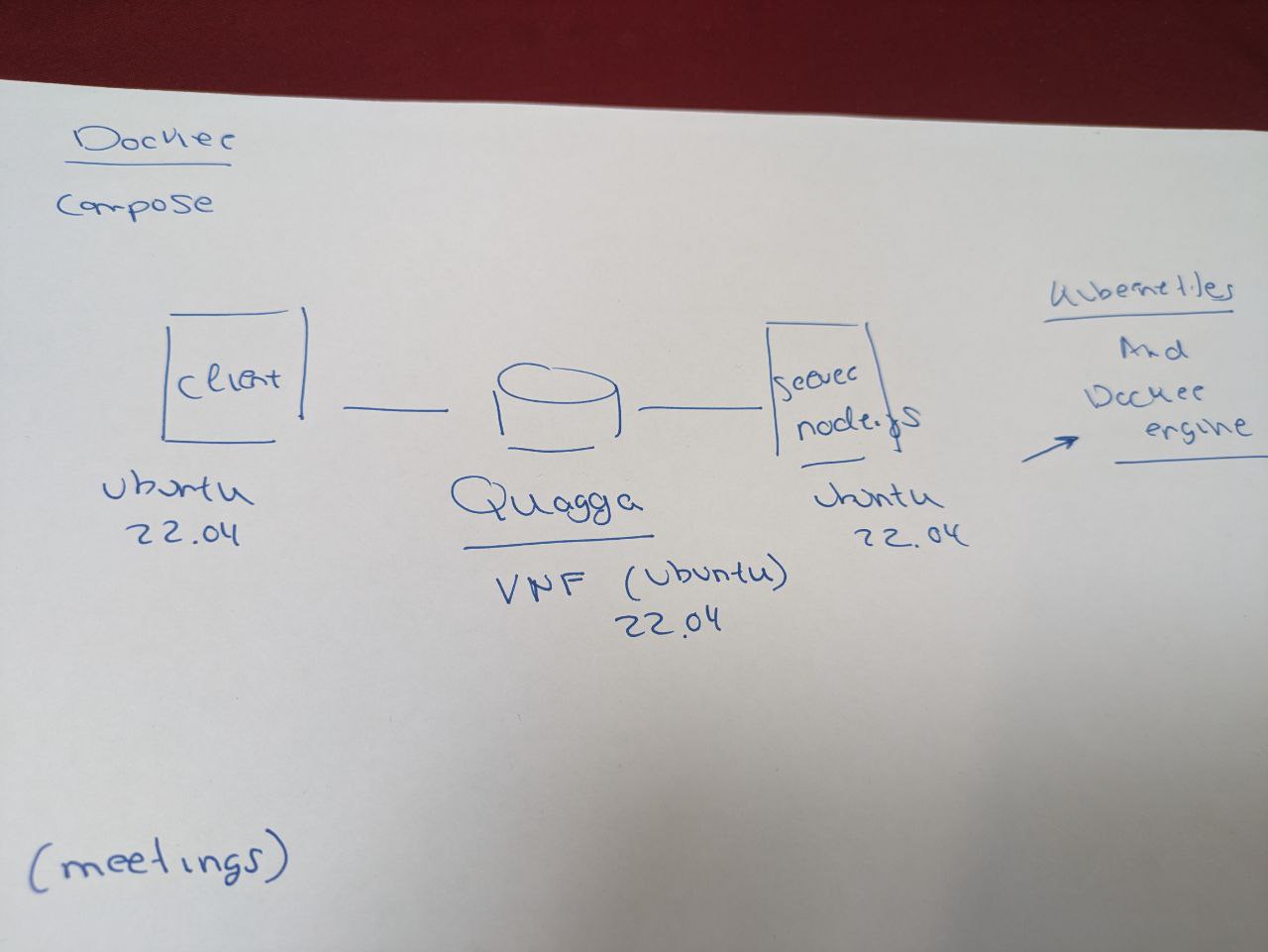
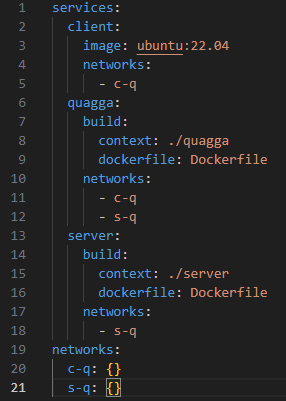
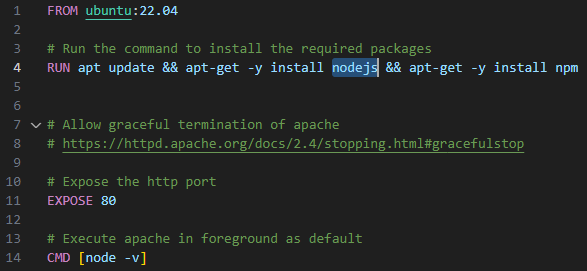
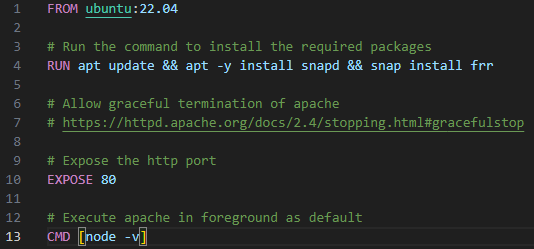
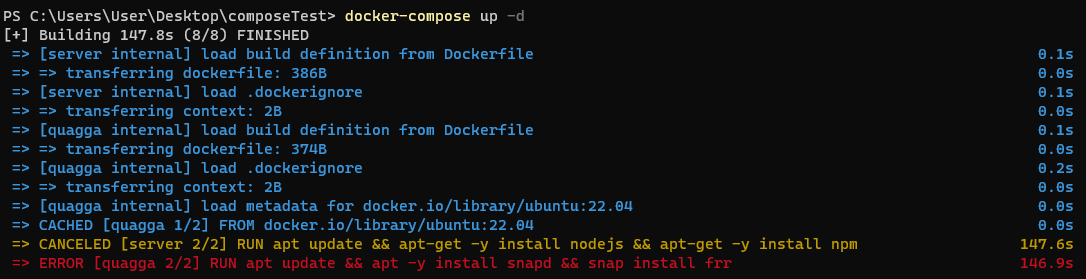
The aim is creating a virtual environment to create a monitoring system or a way to analyse the state of the system.





 server







It seemed that quagga was not available for Ubuntu 22.04 and the installation process it is very time expensive. For this reason, it is taken as hypothesis to switch the image of Ubuntu with other which are lighter.

Comparison between quagga, frr and a new base image.

Quagga is a routing software package that provides TCP/IP based routing services, using protocols such as RIP, OSPF, IS-IS and BGP-4. It also supports IPv6 routing protocols. A system with Quagga installed acts as a dedicated router. With Quagga, your machine exchanges routing information with other routers using routing protocols. Quagga uses this information to update the kernel routing table so that the right data goes to the right place.

It is made from a collection of several daemons that work together to build the routing table. There may be several protocol-specific routing daemons and zebra the kernel routing manager.

Each protocol has its own daemon in charge of managing that routing table. Finally, one more daemon that manages the kernel routing table. It is easy to add a new routing protocol daemons to the entire routing system without affecting any other software. You need to run only the protocol daemon associated with routing protocols in use. Thus, user may run a specific daemon and send routing reports to a central routing console. Each daemon has it’s own configuration file and terminal interface. When you configure a static route, it must be done in zebra configuration file.

FRRouting is a free and open source IRP suite for Linux and Unix platform with its roots planted in Quagga. It offers some improvement for each protocol offered by quagga

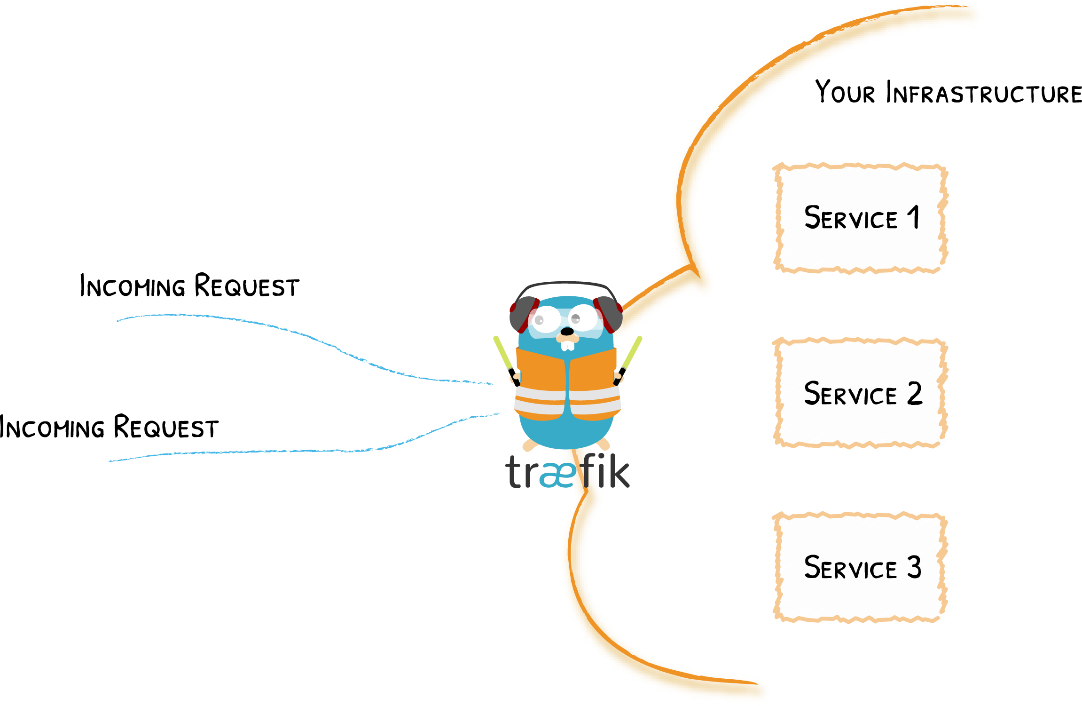
Traefik is an [open-source](https://github.com/traefik/traefik) Edge Router that makes publishing your services a fun and easy experience. It receives requests on behalf of your system and finds out which components are responsible for handling them. With Traefik, there is no need to maintain and synchronize a separate configuration file: everything happens automatically, in real time (no restarts, no connection interruptions).

Traefik is based on the concept of EntryPoints, Routers, Middlewares and Services.

The main features include dynamic configuration, automatic service discovery, and support for multiple backends and protocols.

1. [EntryPoints](https://doc.traefik.io/traefik/routing/entrypoints/): EntryPoints are the network entry points into Traefik. They define the port which will receive the packets, and whether to listen for TCP or UDP.
2. [Routers](https://doc.traefik.io/traefik/routing/routers/): A router is in charge of connecting incoming requests to the services that can handle them.
3. [Middlewares](https://doc.traefik.io/traefik/middlewares/overview/): Attached to the routers, middlewares can modify the requests or responses before they are sent to your service
4. [Services](https://doc.traefik.io/traefik/routing/services/): Services are responsible for configuring how to reach the actual services that will eventually handle the incoming requests.

Traefik is an Edge Router (or reverse Proxy), it means that it's the door to your platform, and that it intercepts and routes every incoming request: it knows all the logic and every [rule](https://doc.traefik.io/traefik/routing/routers/#rule) that determine which services handle which requests



when a service is deployed, Traefik detects it immediately and updates the routing rules in real time. Similarly, when a service is removed from the infrastructure, the corresponding route is deleted accordingly.

This suites as the best option traefik:

<https://doc.traefik.io/traefik/getting-started/quick-start/>

https://doc.traefik.io/traefik/user-guides/docker-compose/basic-example/

Comparison between Ubuntu with node.js, alpine+node and apache.

I have to agree with the others, I always use official images if I can and I always select the Alpine version if available. Why use Alpine? Two reasons:

1. Much small images. Ubuntu is 188MB alone. Then you add your app on top of that probably exceeding 200MB. Alpine Linux is only 4MB! After adding my Python runtime and code most of my images are only 52MB. Compare that will almost 200MB of Ubuntu. Smaller images are smaller upload/download and take up less disk space.
2. Smaller attack surface! When you start from Ubuntu, you are adding lots of other services that may be running that are not needed and could be exploited. Alpine Linux has very little to attack so it is inherently more secure.

You don’t need an entire generic OS… all you need is a runtime for your app and nothing more. I always start with Alpine Linux when I’m building my own containers from scratch.

How to use node:

<https://github.com/optimized/docker-node/blob/master/Dockerfile>

https://github.com/nodejs/docker-node/blob/main/README.md#how-to-use-this-image

<https://github.com/nodejs/docker-node>

Building Ubuntu 22.04 Image

Build image (from project root directory):

docker build -t frr-ubuntu22:latest -f docker/ubuntu-ci/Dockerfile .

Running Full Topotest:

docker run --init -it --privileged --name frr -v /lib/modules:/lib/modules \

frr-ubuntu22:latest bash -c 'cd ~/frr/tests/topotests ; sudo pytest -nauto --dist=loadfile'

Extract results from the above run into *run-results* dir and analyze:

tests/topotest/analyze.py -C frr -Ar run-results

Start the container:

docker run -d --init --privileged --name frr-ubuntu22 --mount type=bind,source=/lib/modules,target=/lib/modules frr-ubuntu22:latest

Running a topotest (when the docker host is Ubuntu):

docker exec frr-ubuntu22 bash -c 'cd ~/frr/tests/topotests/ospf-topo1 ; sudo pytest test\_ospf\_topo1.py'

Starting an interactive bash session:

docker exec -it frr-ubuntu22 bash

Stopping an removing a container:

docker stop frr-ubuntu22 ; docker rm frr-ubuntu22

Removing the built image:

docker rmi frr-ubuntu22:latest