# ENTENDENDO DOCKER E CONTAINERS

By Talison Fabio

# Primeiro, algumas perguntas

- Quem trabalha com desenvolvimento?

# Primeiro, algumas perguntas

- Quem trabalha com desenvolvimento?
- Quem utiliza Docker no trabalho, ou ja utilizou?

# Primeiro, algumas perguntas

- Quem trabalha com desenvolvimento?
- Quem utiliza Docker no trabalho, ou ja utilizou?
- Quem não tem ideia do que é containers?

# Primeiro, algumas perguntas

- Quem trabalha com desenvolvimento?
- Quem utiliza Docker no trabalho, ou ja utilizou?
- Quem não tem ideia do que é containers?
- Quem tem algum conhecimento mas não avançado?

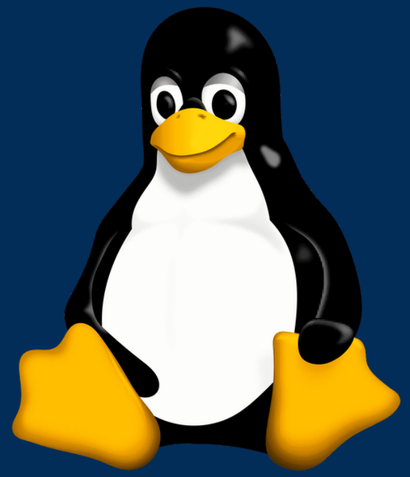
# Primeiro, algumas perguntas

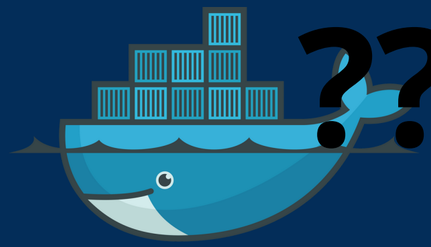
- Quem trabalha com desenvolvimento?
- Quem utiliza Docker no trabalho, ou ja utilizou?
- Quem não tem ideia do que é containers?
- Quem tem algum conhecimento mas não avançado?
- Quem se garante??

# Tecnologias que estão em todo lugar


# Tecnologias que estão em todo lugar

git



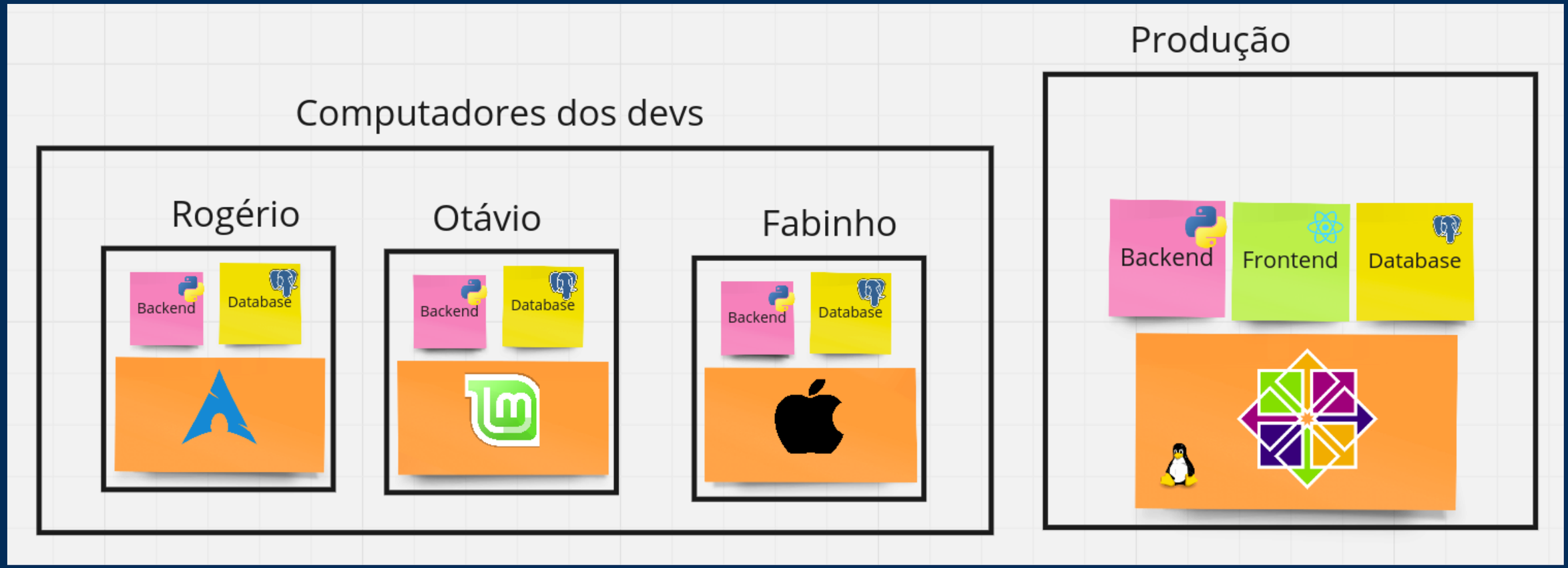


???

docker

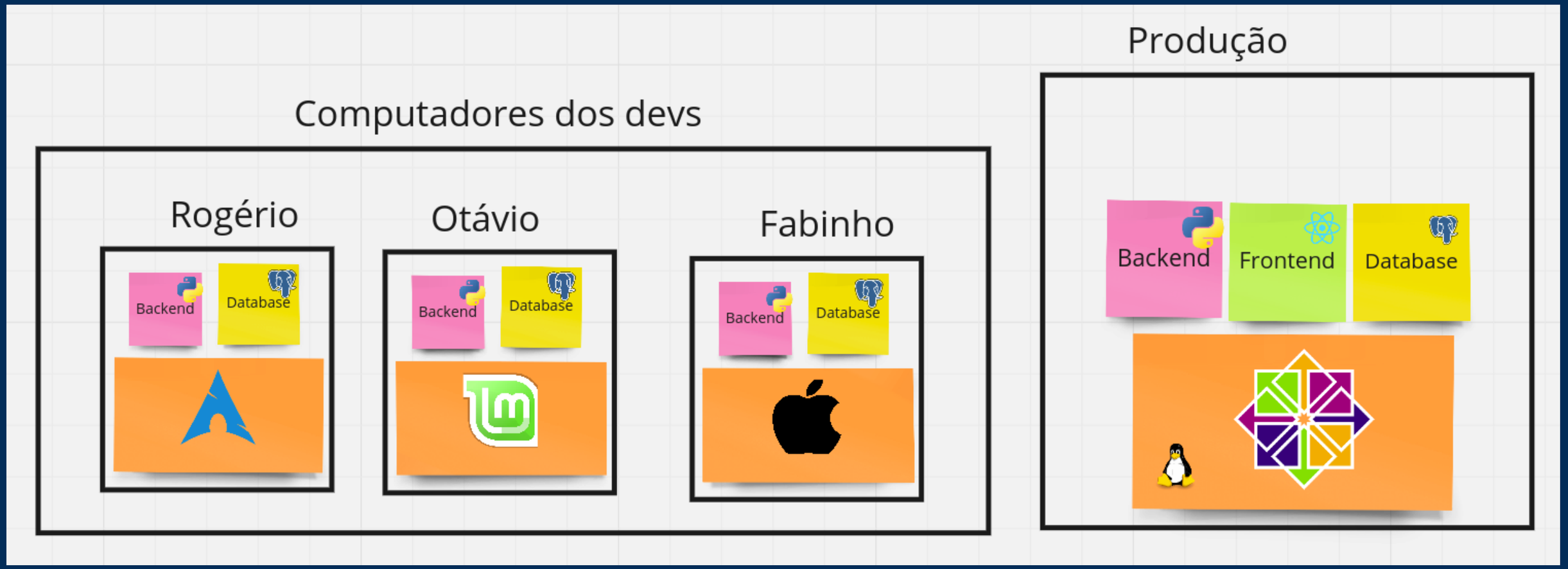
# Quais problemas containers e o docker buscam resolver?

# Historia 1: O dev novo!



# Historia 1: O dev novo!

# Historia 2: Na Minha máquina funciona!



# Historia 2: Na Minha máquina funciona!

# Historia 3: O pobi sysadmin

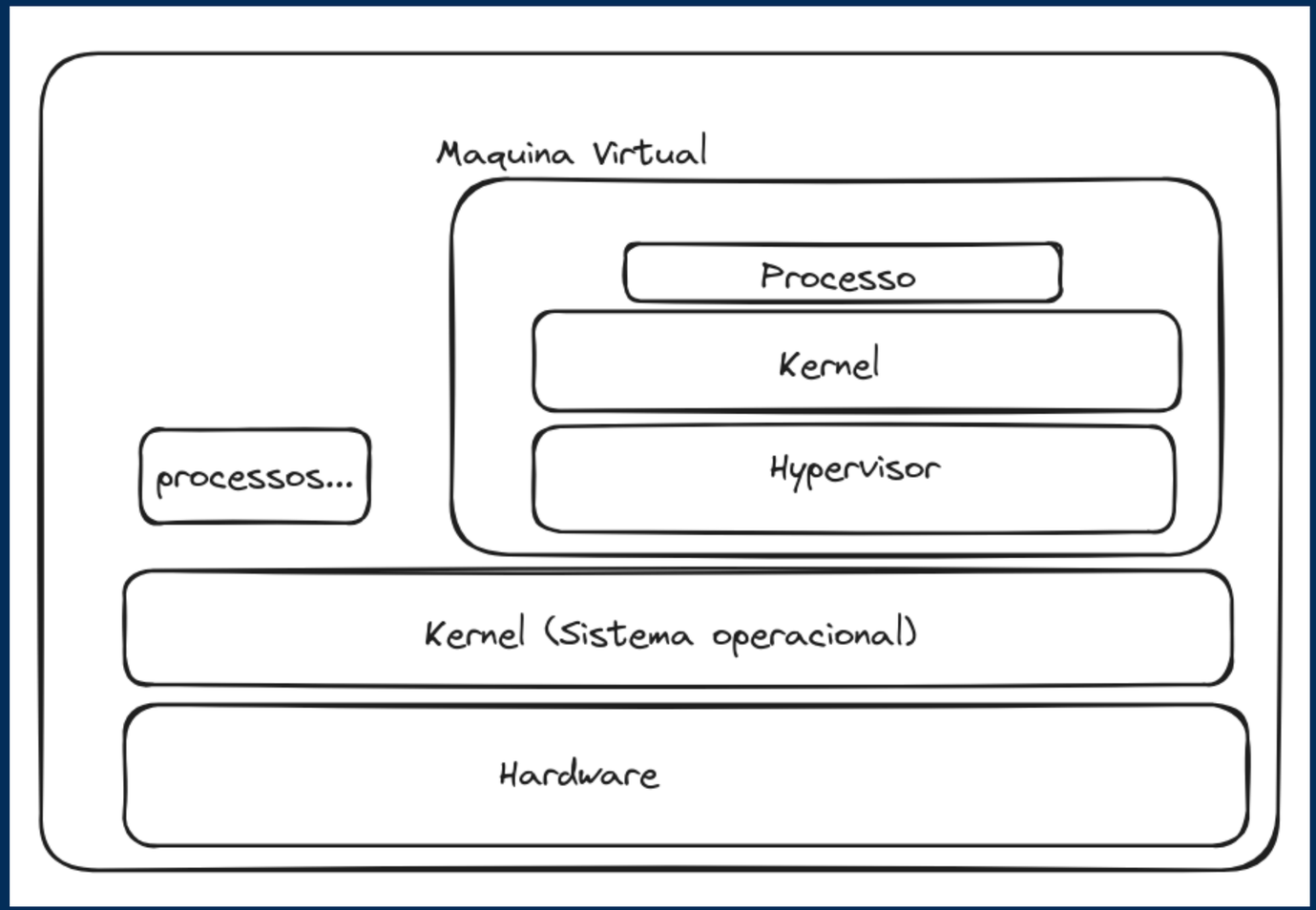
Problemas

- Ambiente não é facilmente replicável
- Ambiente não é mesmo entre dev e prod
- Dificuldade para escalonar

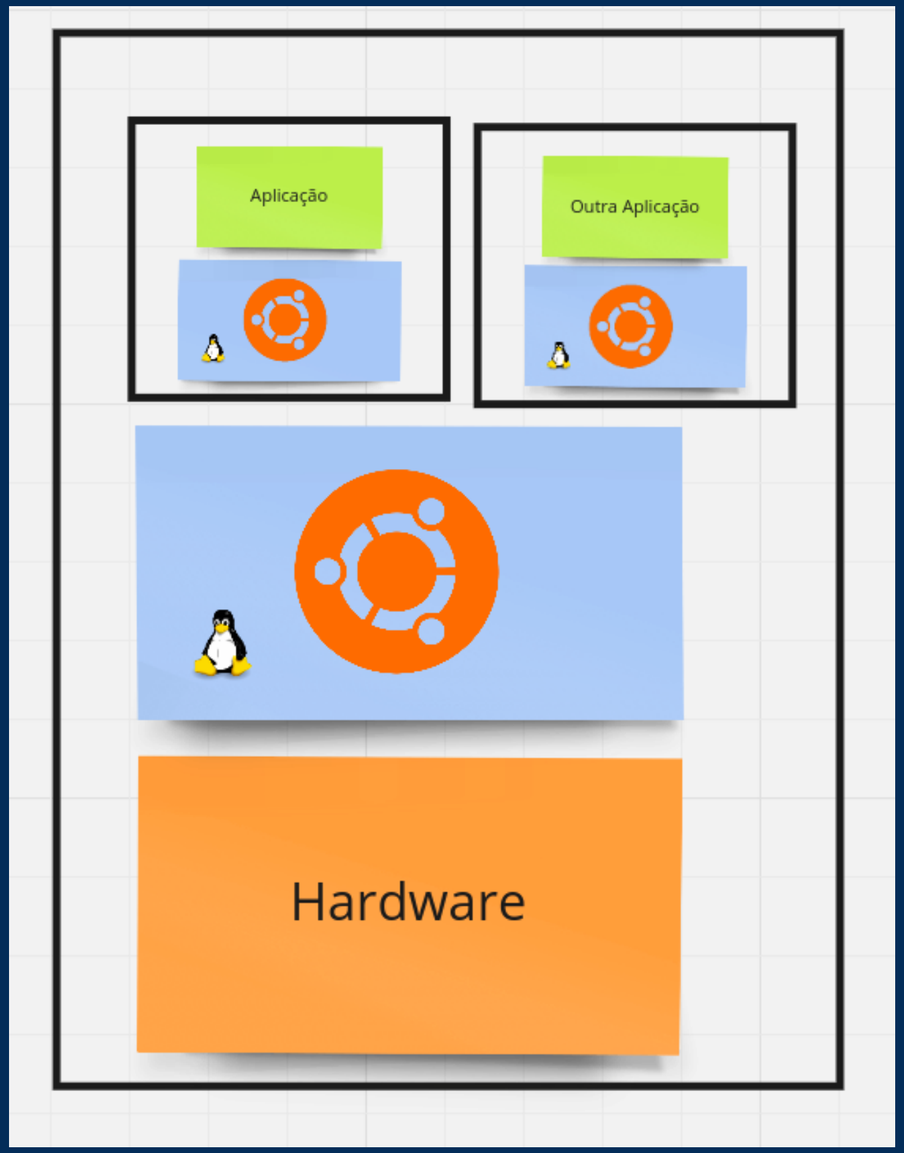
Além disso...

- Isolamento de processos
- Acesso público aos mesmos recursos de todos

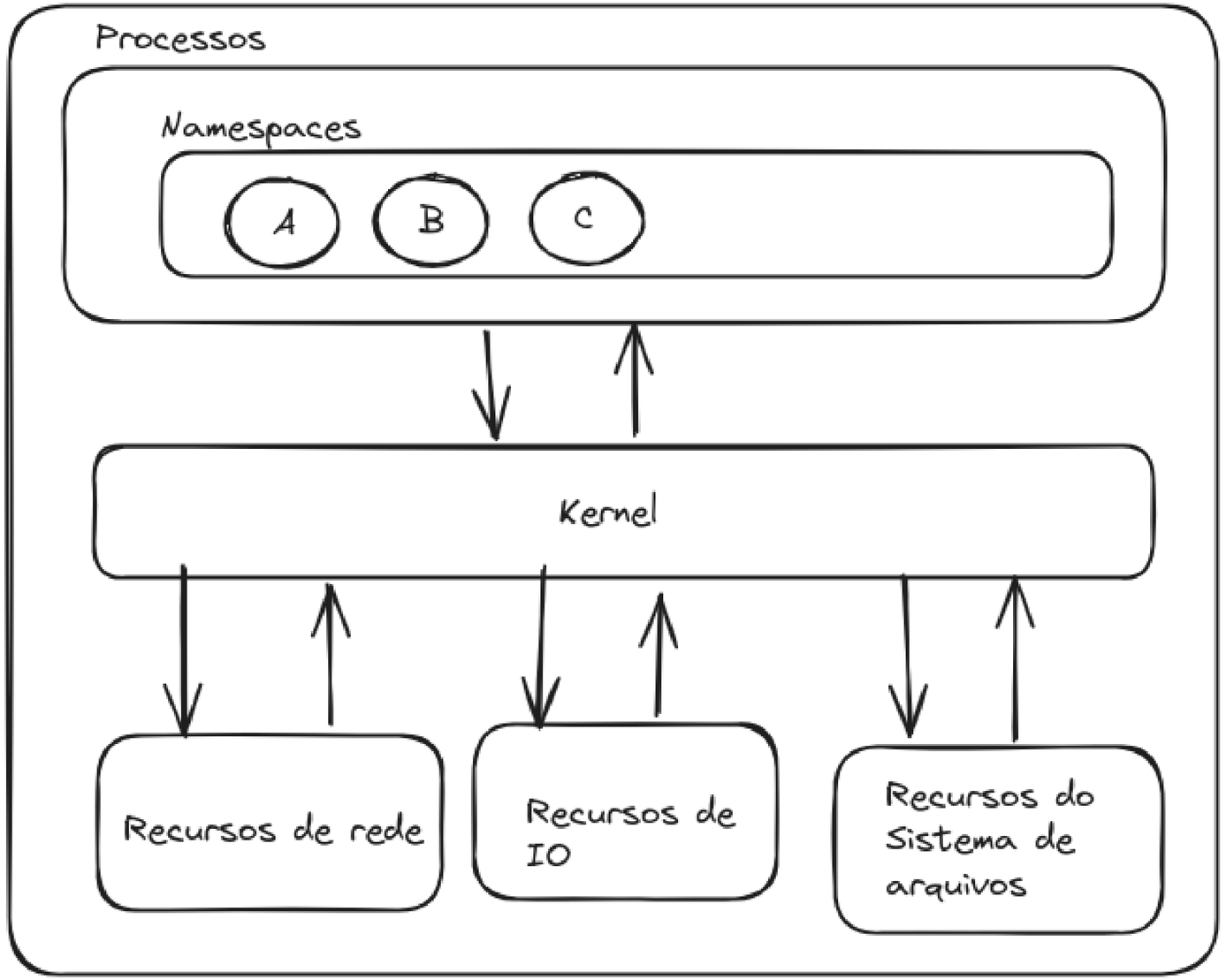
# Solução velha: Maquinas virtuais

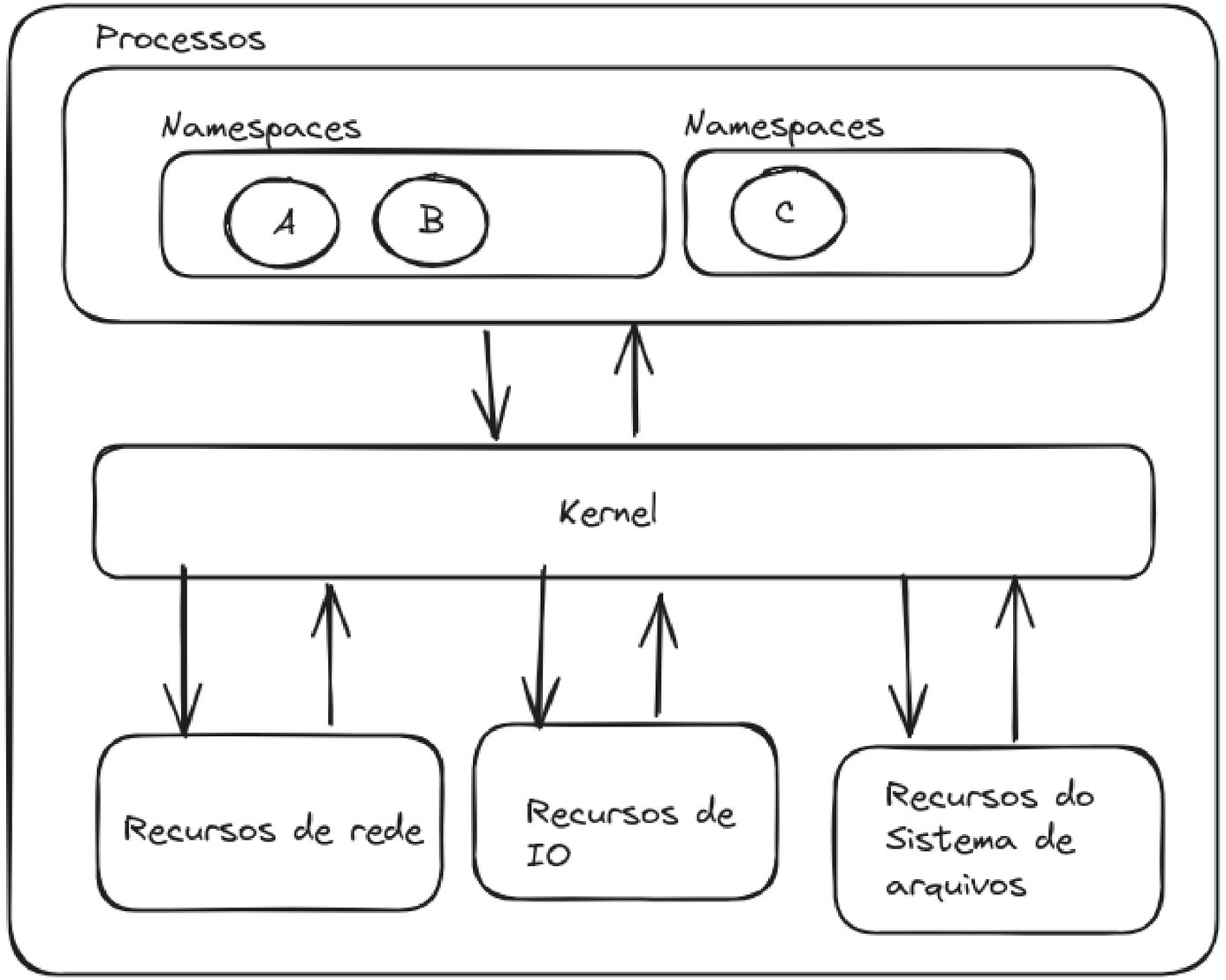


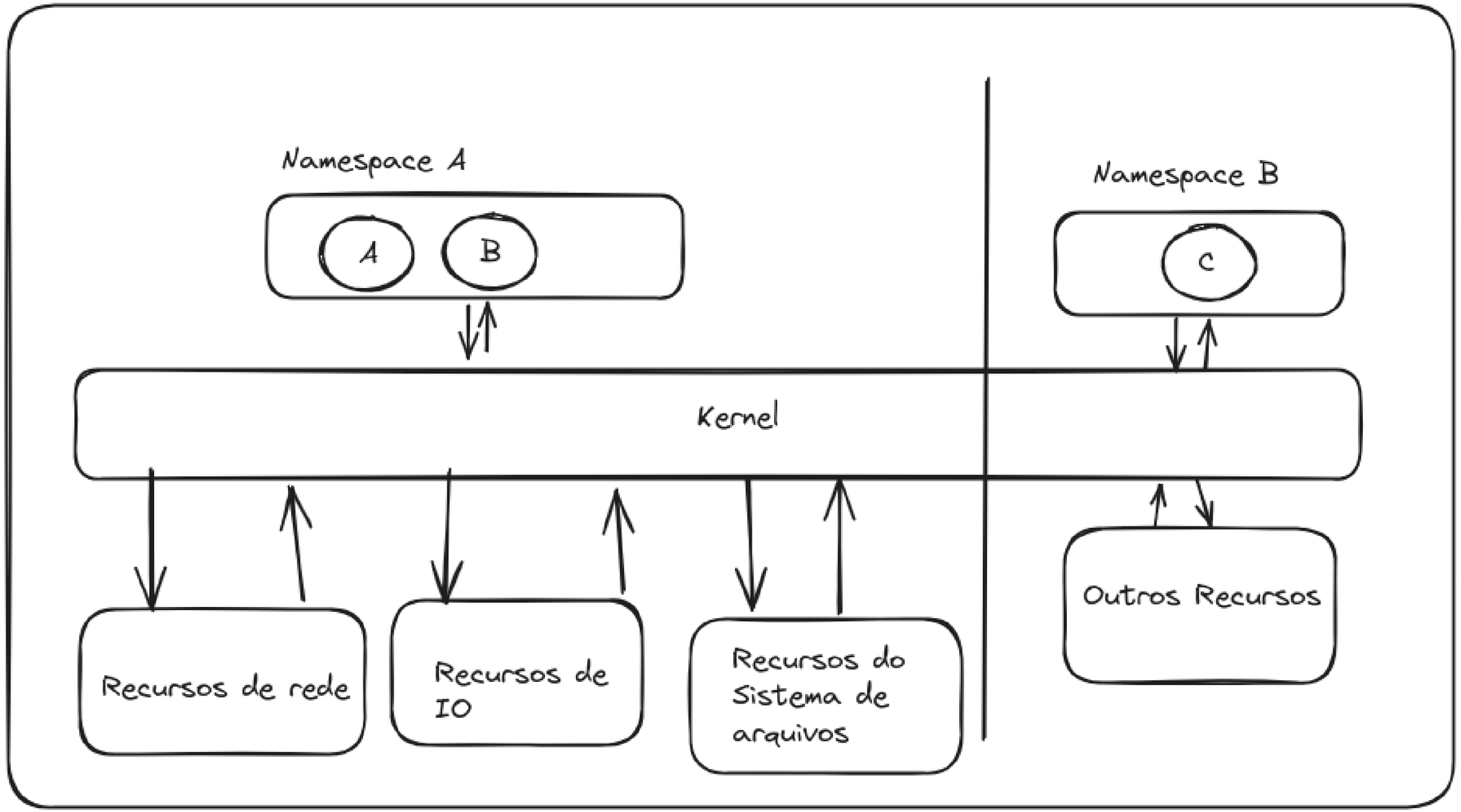
# Solução velha: Maquinas virtuais



# Precisamos pensar melhor....


Processos
Namespaces
A
B
C
Kernel
Recursos de rede
Recursos de IO
Recursos do Sistema de arquivos


Processos
Namespaces
A
B
Namespaces
C
Kernel
Recursos de rede
Recursos de IO
Recursos do Sistema de arquivos


Namespace A
A
B
Namespace B
C
Kernel
Recursos de rede
Recursos de IO
Recursos do Sistema de arquivos
Outros Recursos

# Para isso precisamos falar sobre....

**Namespaces**

**Cgroups**

# Prática

## Utilizar namespaces

- sudo unshare --fork --mount-proc --pid bash
- sudo unshare --fork --pid --mount-proc=./{{ROOT_FAKE}}/proc chroot debian_os /bin/bash

## Isolamento de Filesystem

- chroot {ROOT_FAKE} /bin/bash

# Prática

## Utilizar namespaces

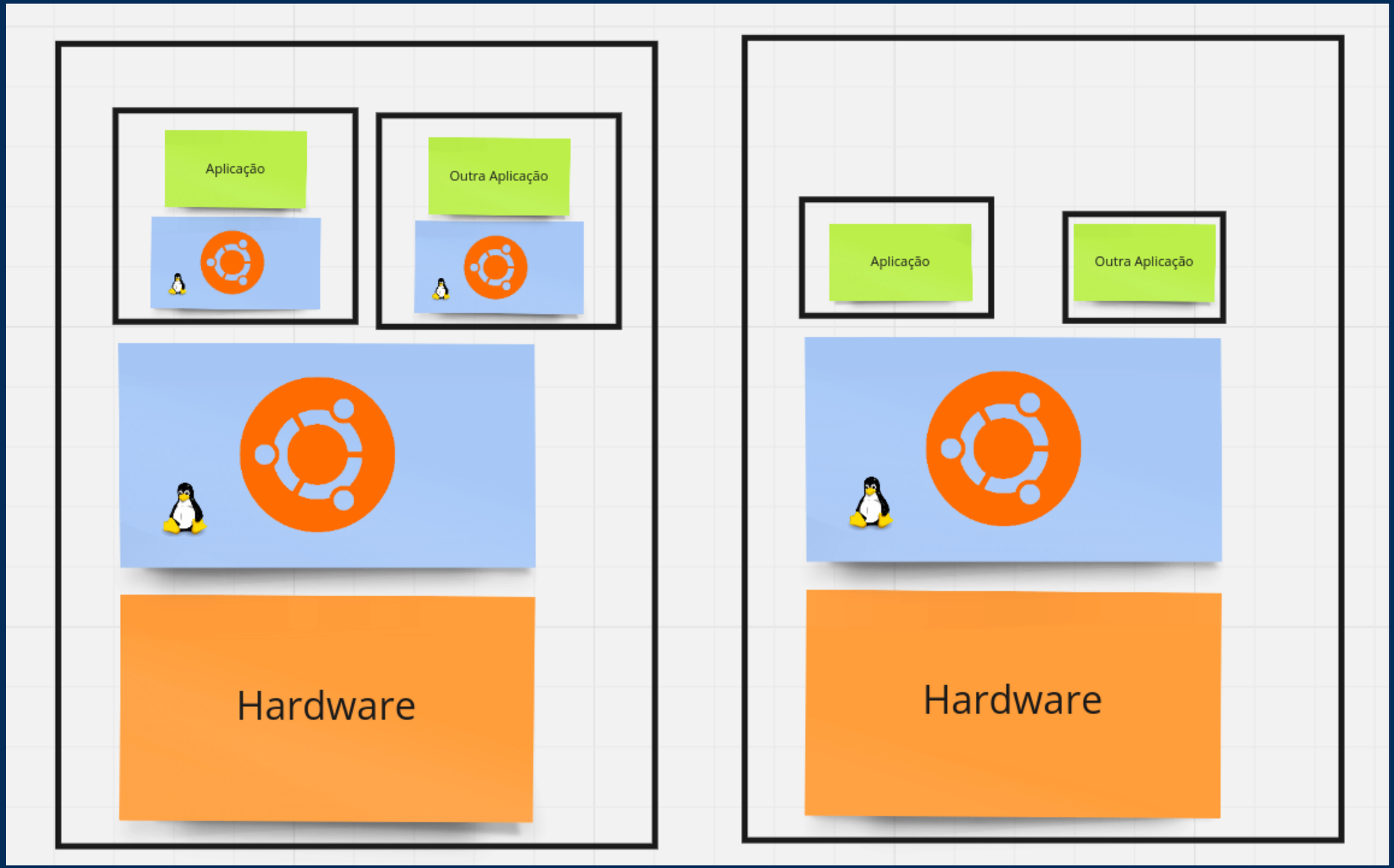
- sudo unshare --fork --mount-proc --pid bash
- sudo unshare --fork --pid --mount-proc=./{{ROOT_FAKE}}/proc chroot debian_os /bin/bash

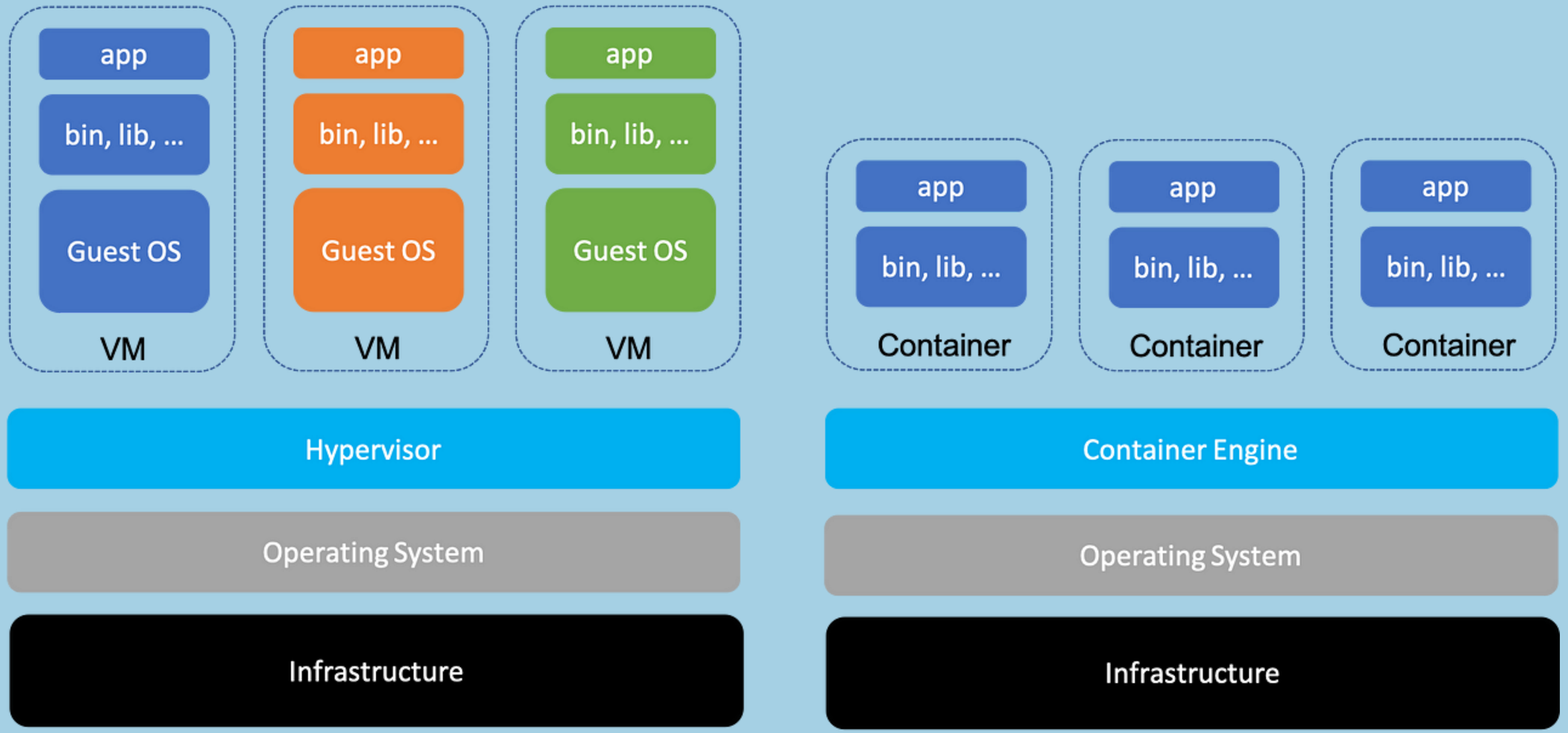
## Isolamento de Filesystem

- chroot {ROOT_FAKE} /bin/bash

## Namespaces + chroot

- sudo unshare --fork --pid --mount-proc={ROOT_FAKE}/proc chroot {ROOT_FAKE} /bin/bash


Aplicação
Outra Aplicação
Hardware
Aplicação
Outra Aplicação
Hardware



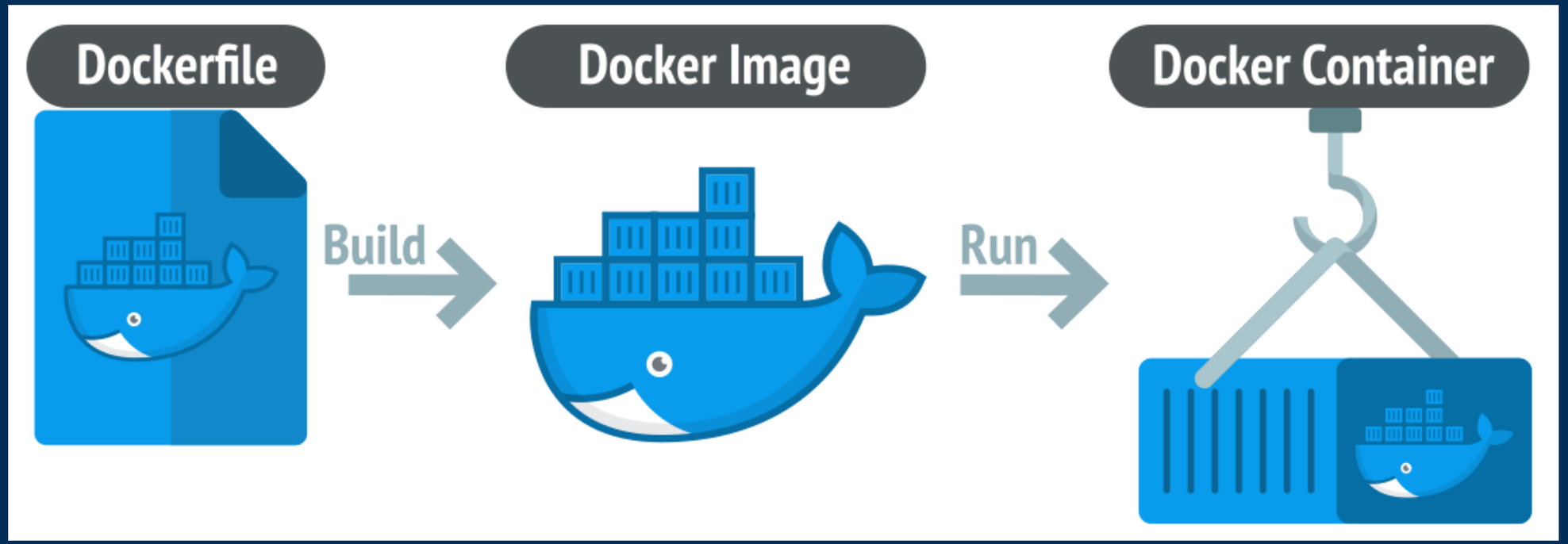

(a) Virtual Machine architecture

(b) Container architecture

# Duvidas???

# O que é Docker afinal então?

Mostrando na prática....
docker pull ubuntu
docker run -it ubuntu


Dockerfile
Build
Docker Image
Run
Docker Container

# Dockerfiles

```
FROM ubuntu

RUN apt update && apt install

COPY requiriments.txt /usr/app
COPY src/main.py /usr/app/src/

RUN pip install -r requiriments

CMD ["python", "main.py"]

```

```
FROM node:18.17.0

COPY backend/package*.json ./

RUN npm install

COPY backend/ /app

RUN npm run build

CMD ["npm", "run", "serve"]

```

## Filters (2) Clear All

**Products**

- [x] Images
- [ ] Extensions
- [ ] Plugins

**Trusted Content**

- [x] Docker Official Image
- [ ] Verified Publisher
- [ ] Sponsored OSS

**Operating Systems**

- [ ] Linux
- [ ] Windows

**Architectures**

- [ ] ARM
- [ ] ARM 64
- [ ] IBM POWER
- [ ] IBM Z
- [ ] PowerPC 64 LE
- [ ] x86
- [ ] x86-64

1 - 25 of 177 available images.

Docker Official Image × images ×

Suggested

**alpine** Docker Official Image · 1B+ · 10K+
Updated 2 months ago
A minimal Docker image based on Alpine Linux with a complete package index and only 5 ...
Linux riscv64 x86-64 ARM ARM 64 386 PowerPC 64 LE IBM Z
Pulls: 9,186,863 Last week
Learn more

**nginx** Docker Official Image · 1B+ · 10K+
Updated 14 days ago
Official build of Nginx.
Linux ARM ARM 64 386 mips64le PowerPC 64 LE IBM Z x86-64
Pulls: 14,507,753 Last week
Learn more

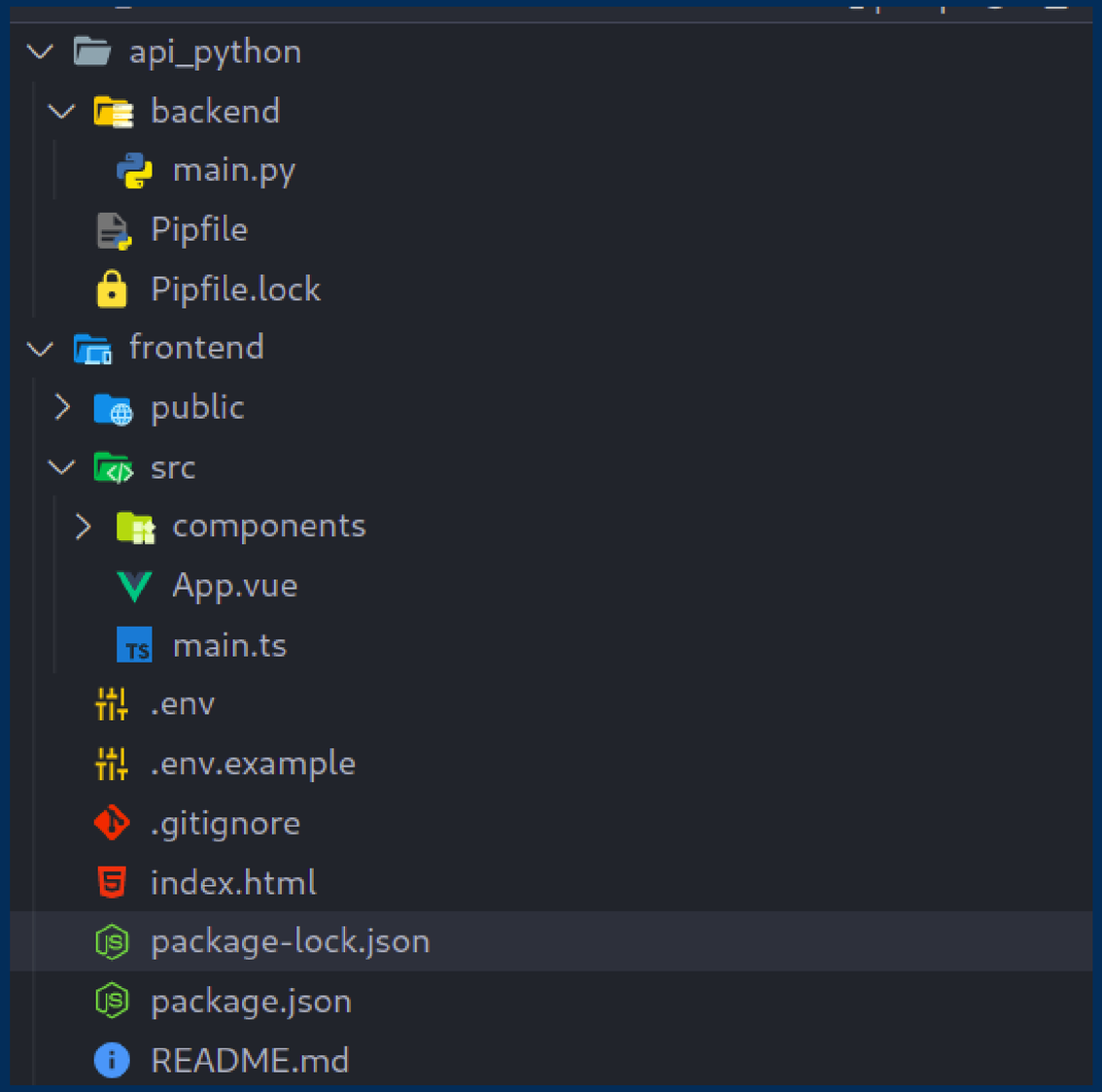
**busybox** Docker Official Image · 1B+ · 3.1K
Updated 4 months ago
Busybox base image.
Linux IBM Z x86-64 ARM ARM 64 386 mips64le PowerPC 64 LE riscv64
Pulls: 8,448,514 Last week
Learn more

**ubuntu** Docker Official Image · 1B+ · 10K+
Updated a month ago
Ubuntu is a Debian-based Linux operating system based on free software.
Linux ARM 64 PowerPC 64 LE IBM Z 386 riscv64 x86-64 ARM
Pulls: 28,080,279 Last week
Learn more

**python** Docker Official Image · 1B+ · 9.2K
Updated 13 days ago
Python is an interpreted, interactive, object-oriented, open-source programming language.
Linux Windows IBM Z mips64le x86-64 ARM ARM 64 386 PowerPC 64 LE
Pulls: 7,993,686 Last week
Learn more

**redis** Docker Official Image · 1B+ · 10K+
Updated 12 days ago
Pulls: 48,894,648 Last week

# Corrigindo uma aplicação com Docker



# Corrigindo uma aplicação com Docker

```python
...
from typing import Union

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

@app.get("/")
def welcome():
    return {"Bad request": "use /api/ na rota"}

@app.get("/api/")
def welcome():
    return {"Bem": "Vindo"}

@app.get("/api/name/{name}")
def welcome_who(name: str):
    return {"Bem-vindo": name}

```

```vue
<script setup lang="ts">
import { ref } from 'vue'

defineProps<{ msg: string }>()

const inputText = ref('Fulano');
const resultText = ref('');

const fetchData = async () => {
  try {
    const response = await fetch(`${import.meta.env.VITE_BACKEND_HOST}/api/name/${inputText.value}`);
    const data = await response.json();
    resultText.value = JSON.stringify(data);
  } catch (error) {
    console.error('Erro ao buscar dados da API:', error);
  }
}

</script>

<template>
  <h1>{{ msg }}</h1>

  <div>
    <form @submit.prevent="fetchData">
      <label>
        Digite algo:
        <input v-model="inputText" type="text" />
      </label>
      <button type="submit">Buscar</button>
    </form>
    <div>
      <p>Resultado:</p>
      <textarea rows="4" cols="50" v-model="resultText" readonly></textarea>
    </div>
  </div>
</template>      You, 14 hours ago • init
```

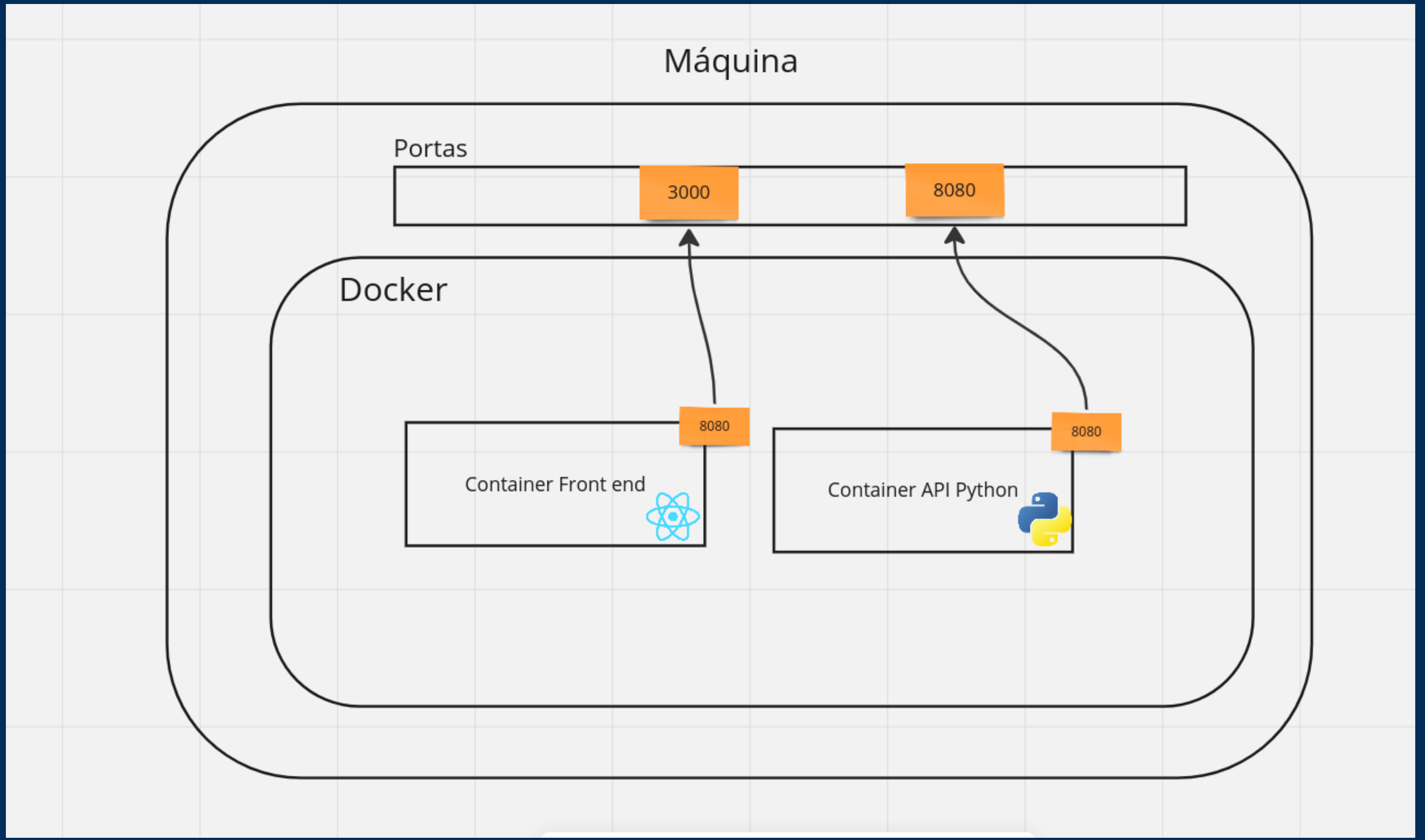
# Construindo as imagens

```dockerfile
FROM node:18.18.2-bullseye

RUN npm install -g http-server

RUN mkdir -p /usr/app
COPY package*.json /usr/app/
COPY . /usr/app/

RUN cd /usr/app/ && npm install
RUN cd /usr/app/ && npm run build

CMD [ "http-server", "/usr/app/dist" ]
```

```dockerfile
FROM python:3.9

# Aqui ele seta o pasta "atual" do container
# Ou seja, não é necessário usar 'cd /usr/app' toda vez
# e na linha 14 o backend/ da direita é na verdade '/usr/app/backend/'
WORKDIR /usr/app

RUN pip install pipenv

COPY Pipfile Pipfile
COPY Pipfile.lock Pipfile.lock
RUN pipenv install --system --deploy

COPY backend/ backend/

CMD ["uvicorn", "backend.main:app", "--host", "0.0.0.0", "--port", "8080"]
```

```
apps/basic_setup on  master [?]
❯ cd frontend/

apps/basic_setup/frontend on  master [?] via  v21.1.0
❯ ls
dist  Dockerfile  index.html  node_modules  package-lock.json  pacl

apps/basic_setup/frontend on  master [?] via  v21.1.0
❯ docker build -t frontend .
```

```
apps/basic_setup/frontend on  master [?] via  v21.1.0
❯ docker run --rm -it -p 3000:8080 frontend
```


Máquina
Portas
3000
8080
Docker
8080
Container Front end
8080
Container API Python

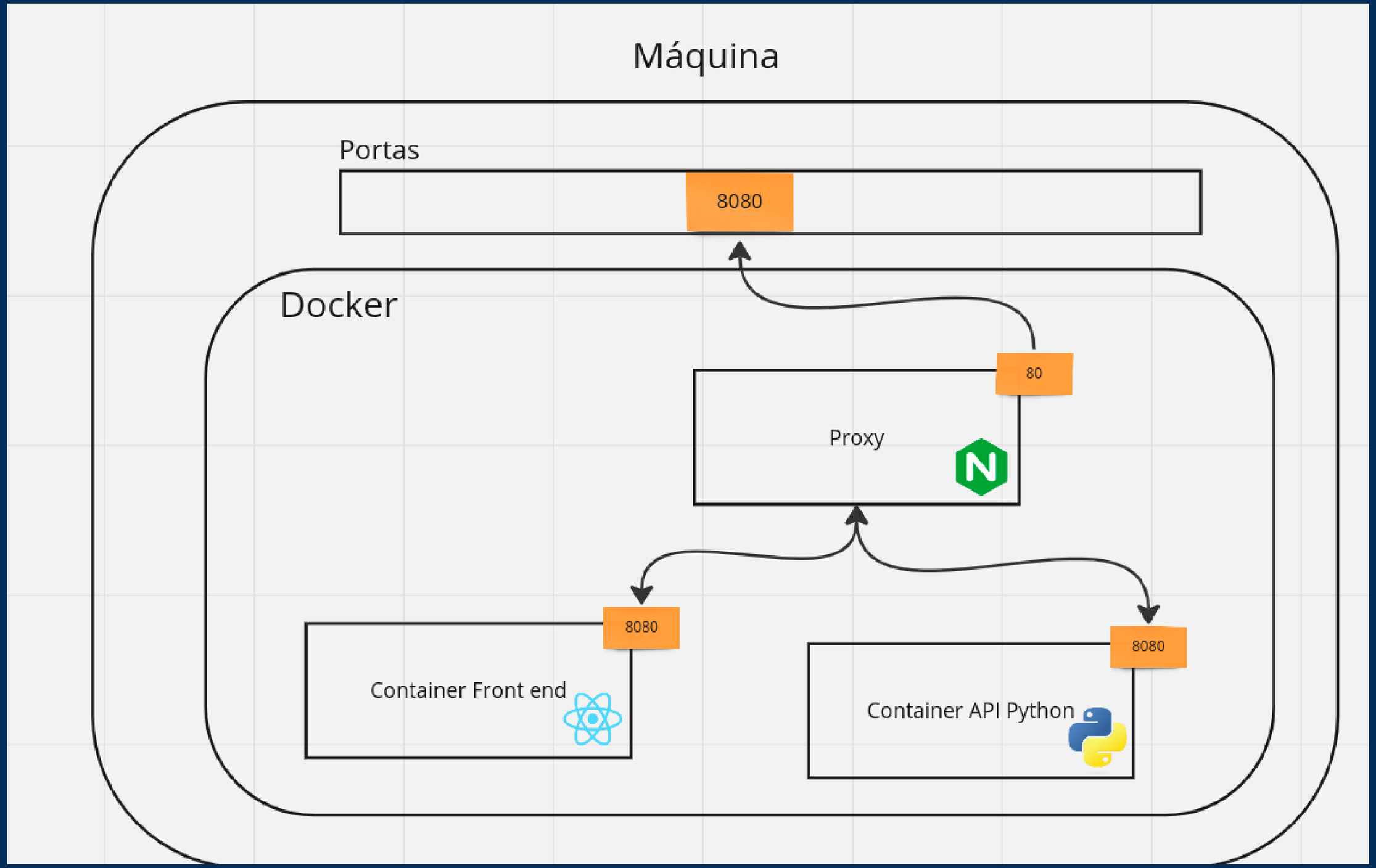
# Docker compose

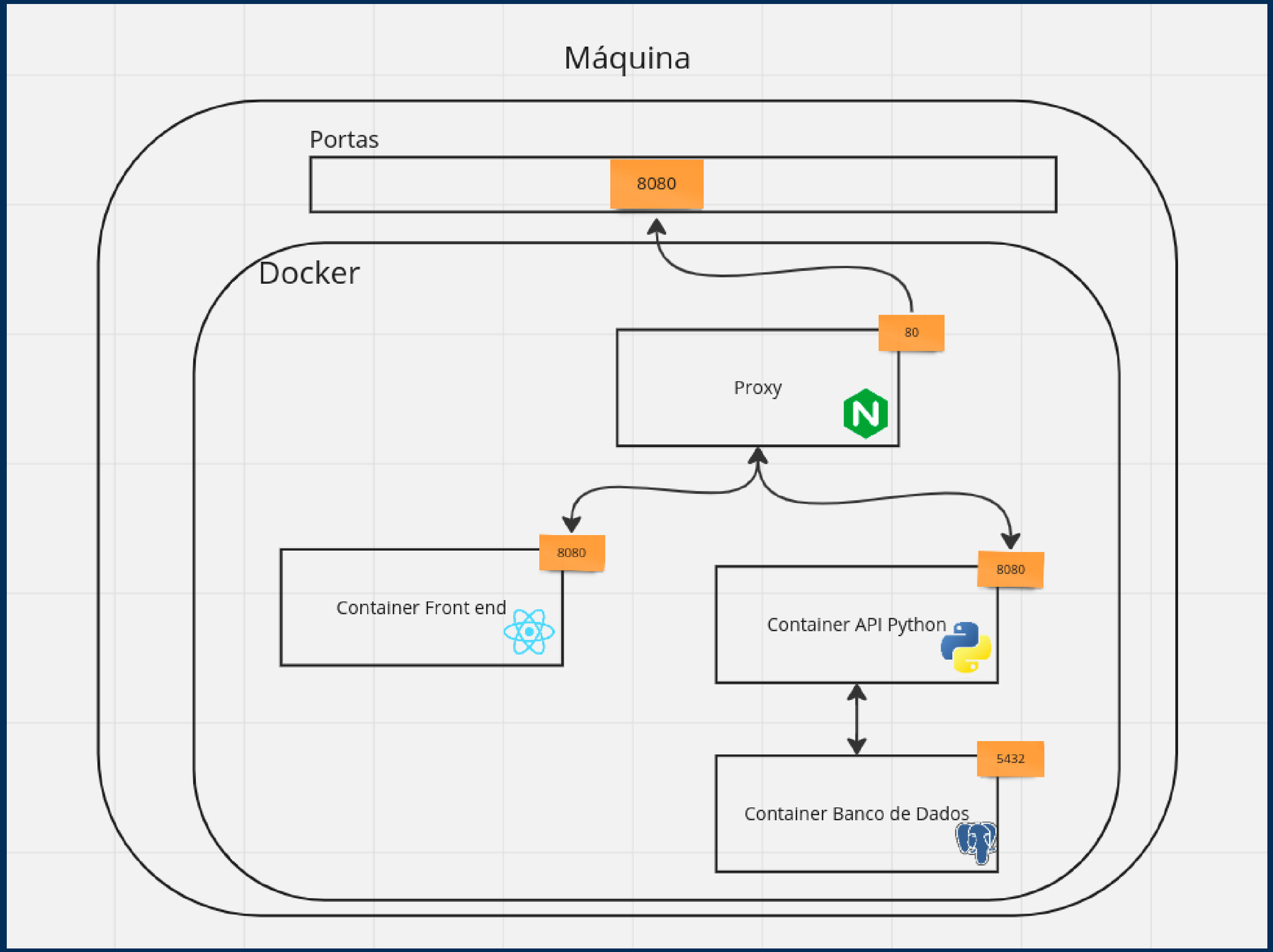
```yaml
version: '3'

services:
  backend:
    build:
      context: api_python
      dockerfile: Dockerfile
    ports:
      - '3001:8080'


  frontend:
    build:
      context: frontend
      dockerfile: Dockerfile
    ports:
      - '8080:8080'
```

# Um exemplo um pouco mais avançado

```yaml
version: '3'

services:
  gateway:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./gateway/nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - backend
      - frontend

  backend:
    build:
      context: api_python
      dockerfile: Dockerfile

  frontend:
    build:
      context: frontend
      dockerfile: Dockerfile
```

```
http {
  # Redirect requests starting with "/api/" to the backend service
  server {
    listen 80;
    server_name localhost;

    location /api/ {
      proxy_pass http://backend:8080/api/; # <=============
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    # Default route (anything else) goes to the frontend service
    location / {
      proxy_pass http://frontend:8080/; # <=============      You, 1 s
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
  }
}
```


Máquina
Portas
8080
Docker
80
Proxy
8080
Container Front end
8080
Container API Python


Máquina
Portas
8080
Docker
80
Proxy
8080
Container Front end
8080
Container API Python
5432
Container Banco de Dados

# Duvidas???

# Onde usar Docker?

# O que existe de containers além do Docker?

## Orquestação


## Alternativas

podman

## Aplicativos desktop

Flatpak

## Distribuições linux usando containers além do básico

fedora
SILVERBLUE

Vanilla

Por hoje é isso pessoal.....

# THANK YOU!

Slides e codígos usados

https://github.com/talis-fb/entendendo_containers