

ESBD 4

Sistemas Distribuídos



INFORMAÇÃO,
TECNOLOGIA
& INOVAÇÃO

DOCKER

Por que Docker?



mongoDB®



PostgreSQL



Dependências

Bibliotecas

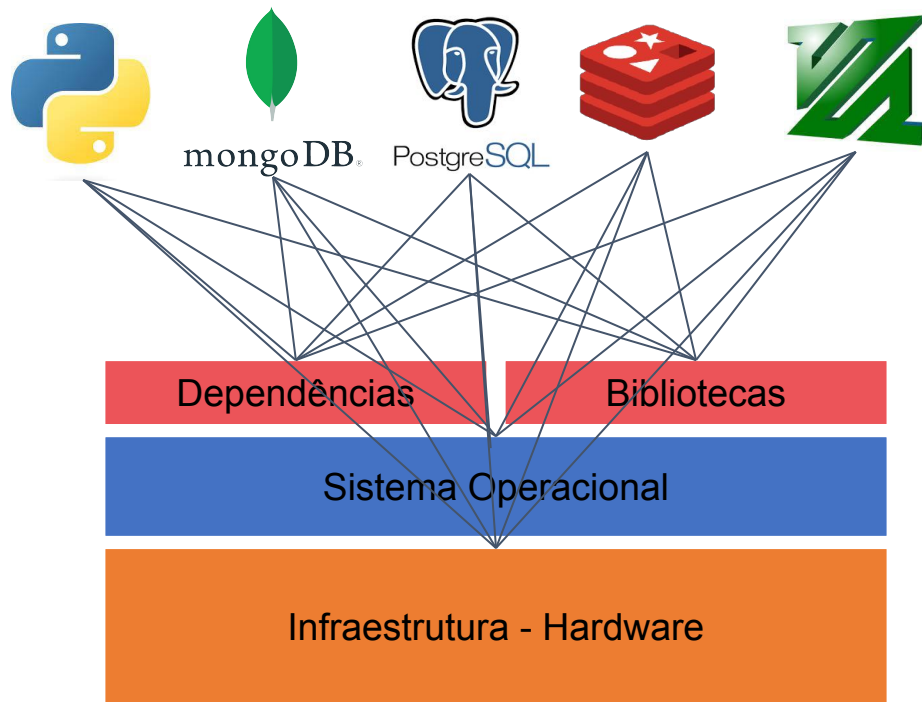
Sistema Operacional

Infraestrutura - Hardware

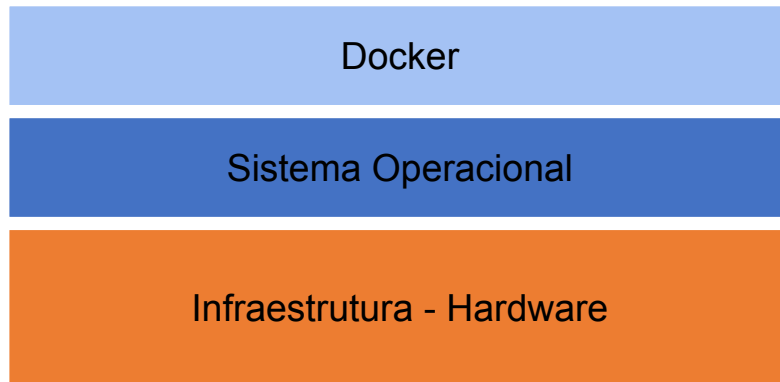
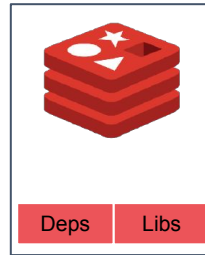


DOCKER

Por que Docker?

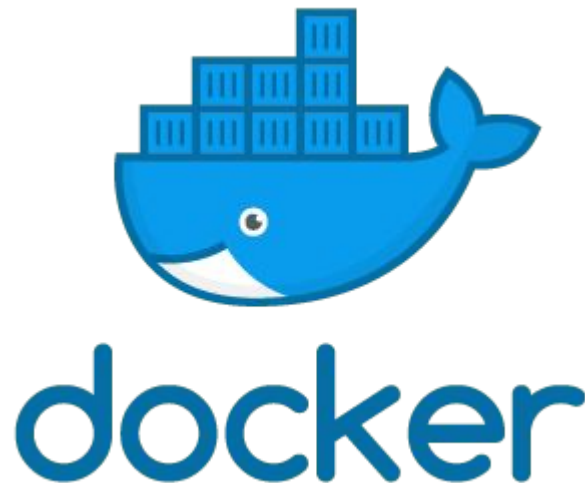


DOCKER



DOCKER

- Ambientes Isolados
- Ambientes Portáteis
- Leve
- Simples
- Versátil:
 - Desenvolvimento
 - Deploy
- Comunidade muito ativa



Docker Hub (hub.docker.com): crie uma conta



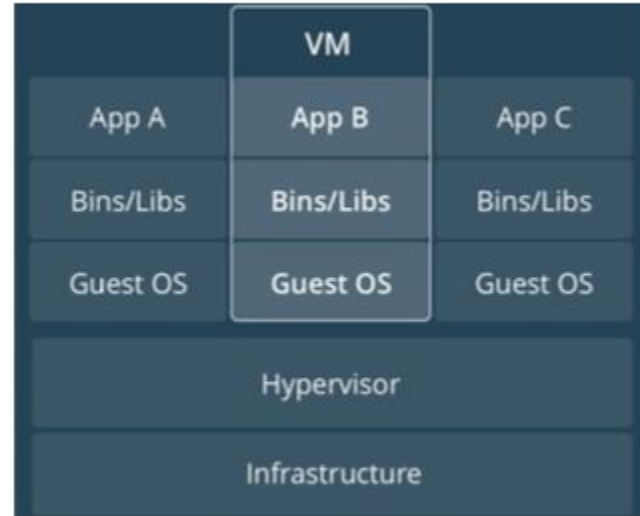
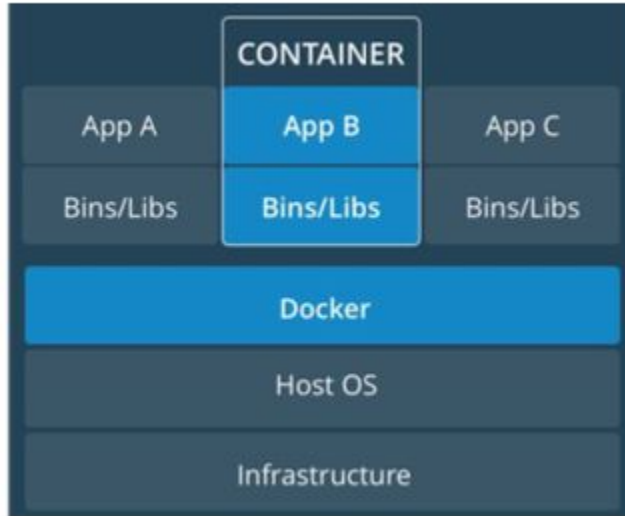
Explore Official Repositories

 nginx official	6.1K STARS	10M+ PULLS	> DETAILS
 redis official	3.8K STARS	10M+ PULLS	> DETAILS
 busybox official	1.0K STARS	10M+ PULLS	> DETAILS
 ubuntu official	6.1K STARS	10M+ PULLS	> DETAILS
 registry official	1.5K STARS	10M+ PULLS	> DETAILS
 alpine official	2.3K STARS	10M+ PULLS	> DETAILS
 mysql official	4.4K STARS	10M+ PULLS	> DETAILS
 mongo official	3.3K STARS	10M+ PULLS	> DETAILS

Source: docs.docker.com

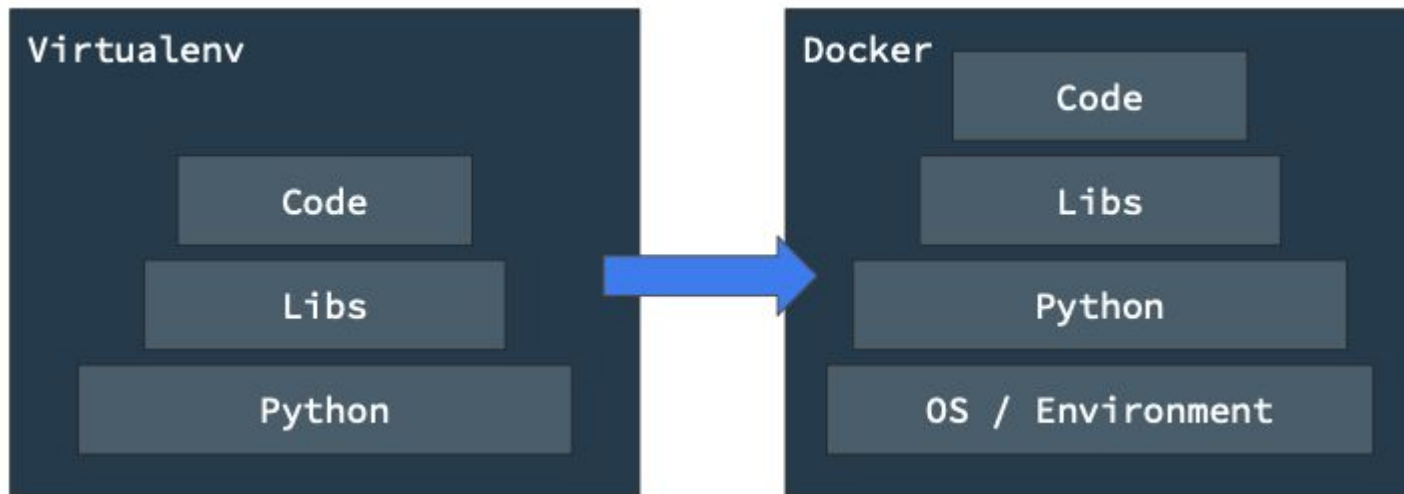


DOCKER



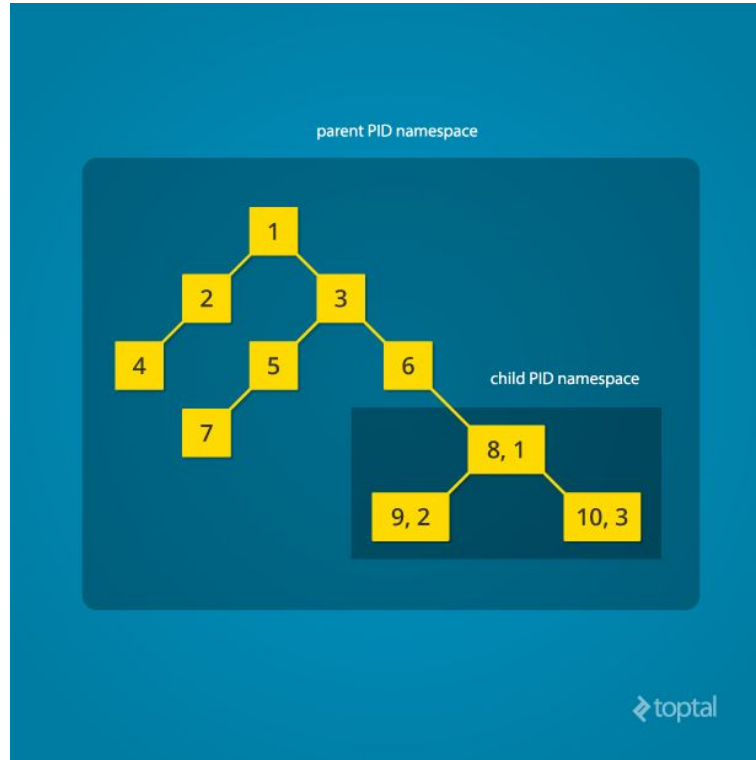
DOCKER

Paralelo ao Virtualenv



DOCKER

Kernel Namespaces



DOCKER

Vamos à prática:

<https://www.docker.com/products/docker-desktop/>

```
docker run hello-world
```

```
docker run -it debian /bin/bash
```



DOCKER

Vamos à prática:

<https://www.docker.com/products/docker-desktop/>

docker run hello-world

docker run -it debian /bin/bash

Anexar o
container no
terminal

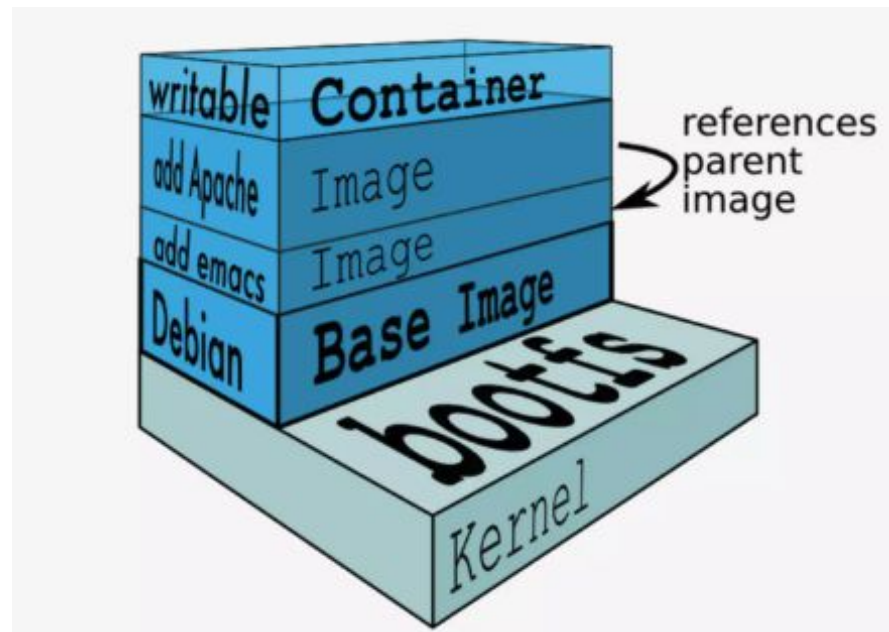
imagem,
instruções/planta
/blueprint

comando que
será executado:
"python main.py"
etc

DOCKER

Imagem vs. Container

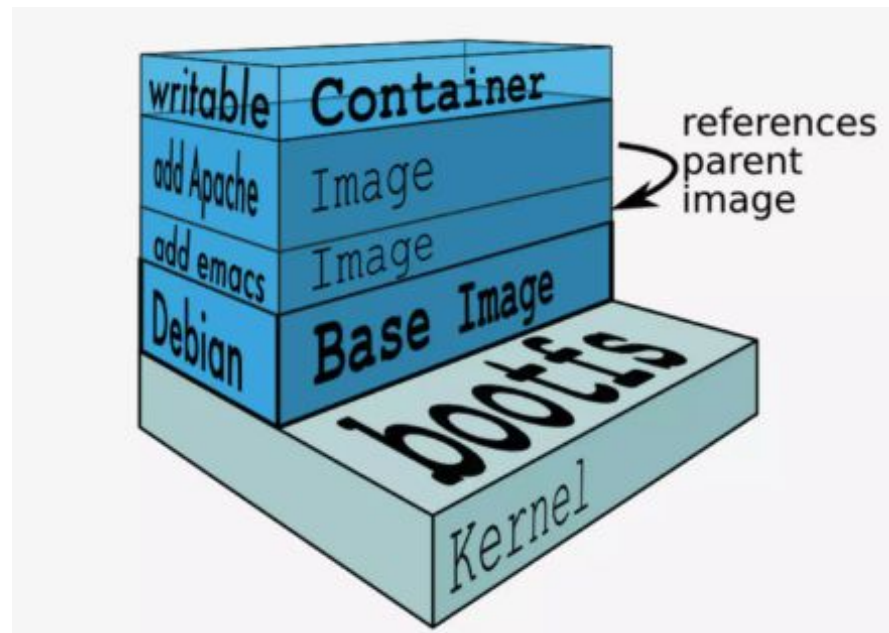
Imagem: Instruções de como um container deve ser montado. É um **template**, um **modelo** contendo as informações necessárias para construir o container. A imagem pode ser enviada de um computador para outro.



DOCKER

Imagem vs. Container

Container: Instância de uma imagem em utilização. É construído a partir de uma imagem. O container é a instância de run-time da imagem. Pode ser pausado e inicializado.

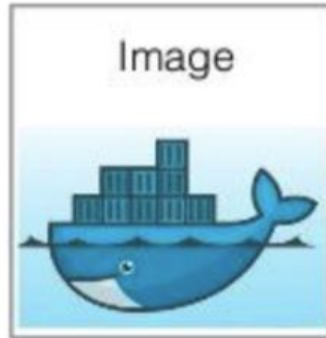


DOCKER

```
FROM ubuntu:14.04
MAINTAINER John Doe <john.doe@example.com>
RUN apt-get update && apt-get install -y python
RUN python --help
```

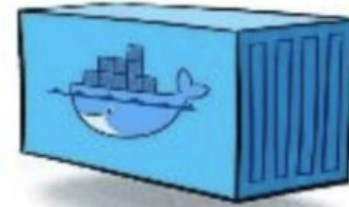
Dockerfile

build



Docker Image

run



Docker Container

DOCKER

```
Dockerfile

FROM python:3

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt

COPY . .

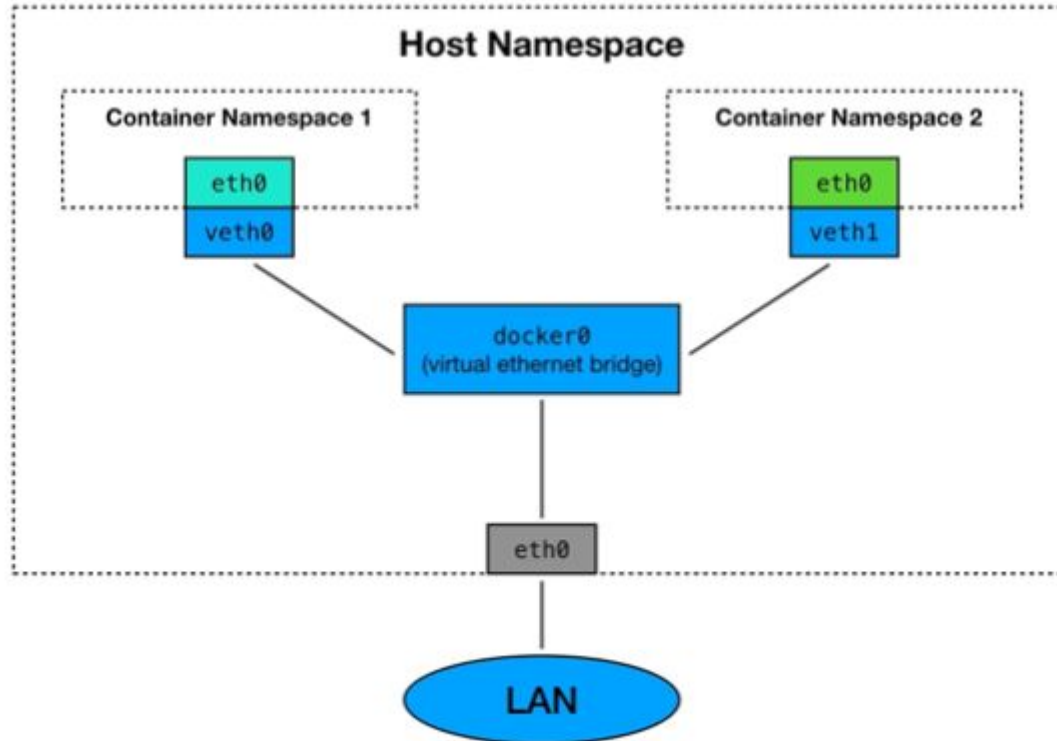
CMD [ "python", "main.py" ]
```



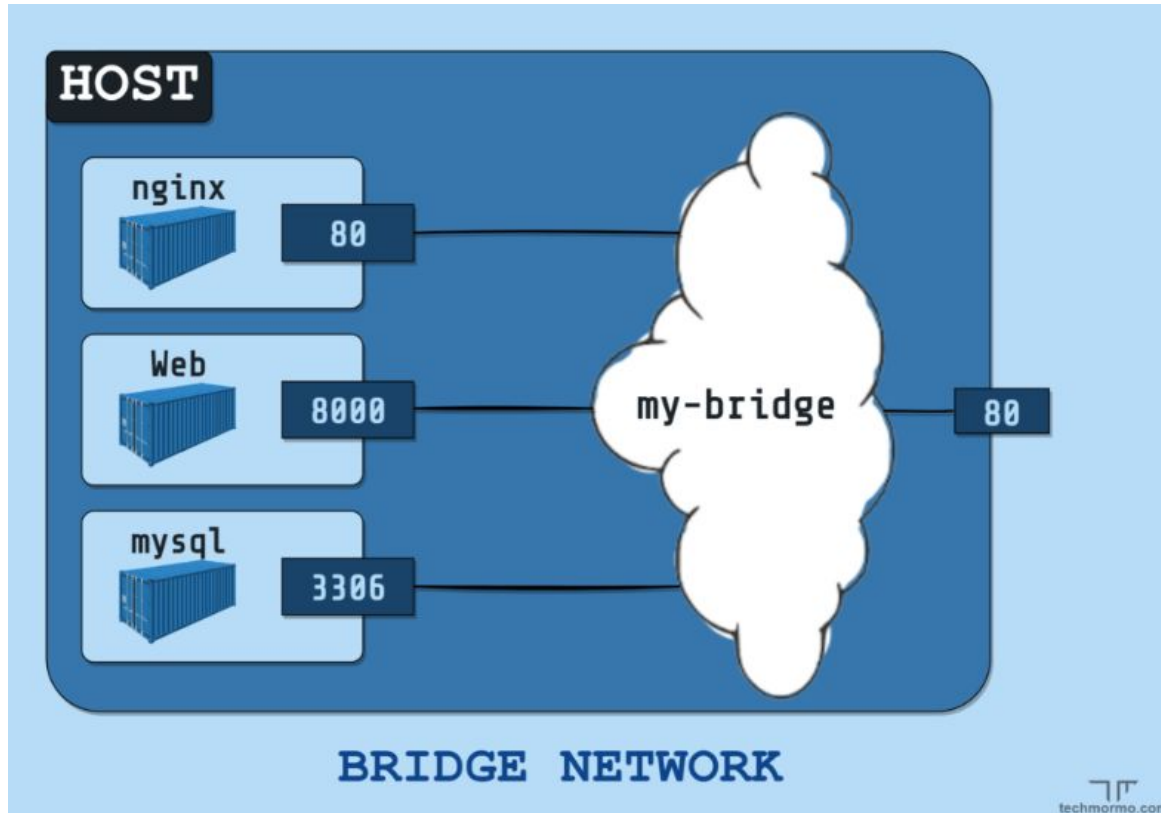
DOCKER

- docker build # constrói uma imagem a partir do Dockerfile
- docker images # lista as imagens
- docker run # roda uma imagem (cria um container)
- docker ps # lista todas os containers
- docker logs # printa o stdout de um container
- docker stop # para um container
- docker rm # remove um container
- docker rmi # remove uma imagen

DOCKER

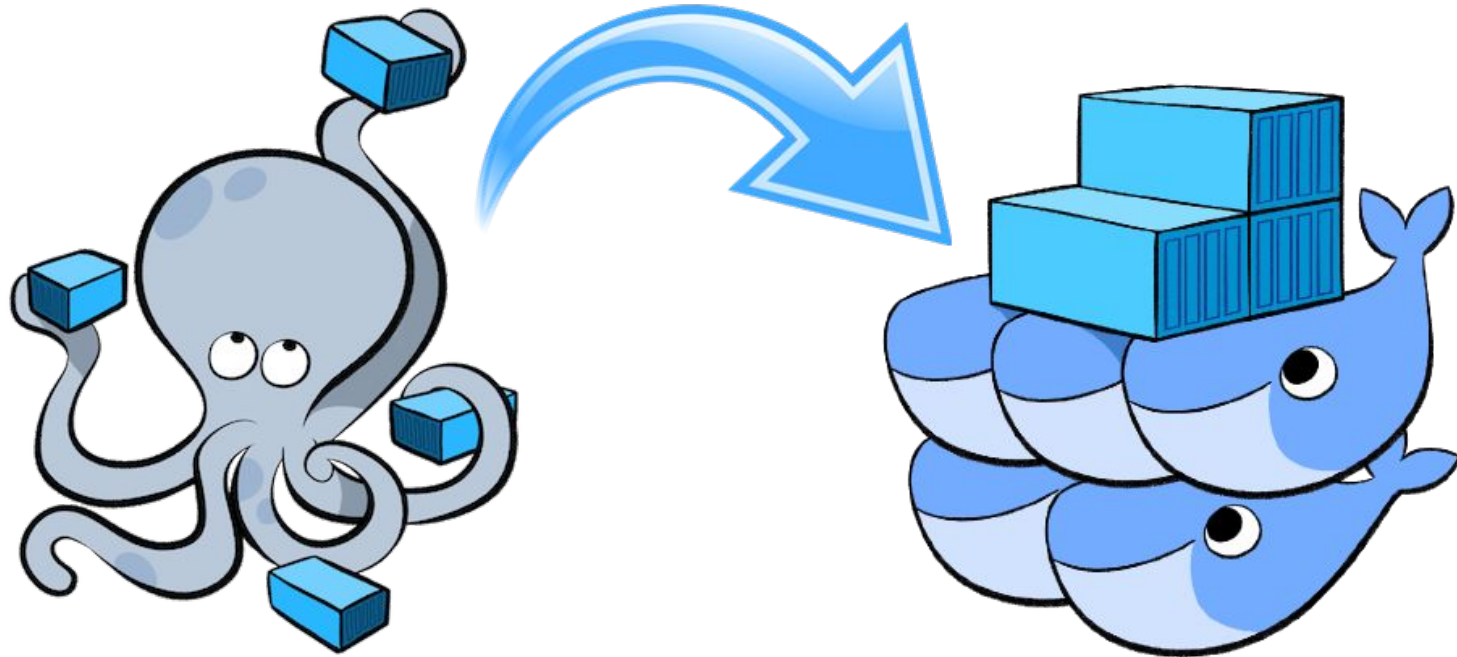


DOCKER



DOCKER

Docker Compose: Criação de múltiplos containers de forma padronizada



ZeroMQ

ØMQ:

Zero Message Queue

Biblioteca de troca de mensagens de alta performance, orientada à sockets.

Utilizada para comunicação entre partes de um mesmo sistema.



ZeroMQ

ZeroMQ:

O **zero** está para zero broker. Não possui um servidor externo. Tornando-a mais eficiente.

Suporte para diversas linguagens de programação.



ZeroMQ

ZMQ Server

```
from time import sleep

import zmq

context = zmq.Context()
socket = context.socket(zmq.REP)
socket.bind("tcp://*:5555")

msg = socket.recv()
print(msg)

sleep(1) # Work

socket.send(b"Done!")
```

ZMQ Client

```
import zmq

context = zmq.Context()
socket = context.socket(zmq.REQ)
socket.connect("tcp://localhost:5555")

socket.send(b"Do some work!")
msg = socket.recv()
print(msg)
```

ZeroMQ

Diversos padrões de comunicação (principais)

- PUSH/PULL
- REQ/REP
- PUB/SUB



Utilizados para construção de arquiteturas mais complexas. Os servidores e clientes não precisam seguir um padrão fixo: Cliente e Servidor pode ser qualquer lado da transação.

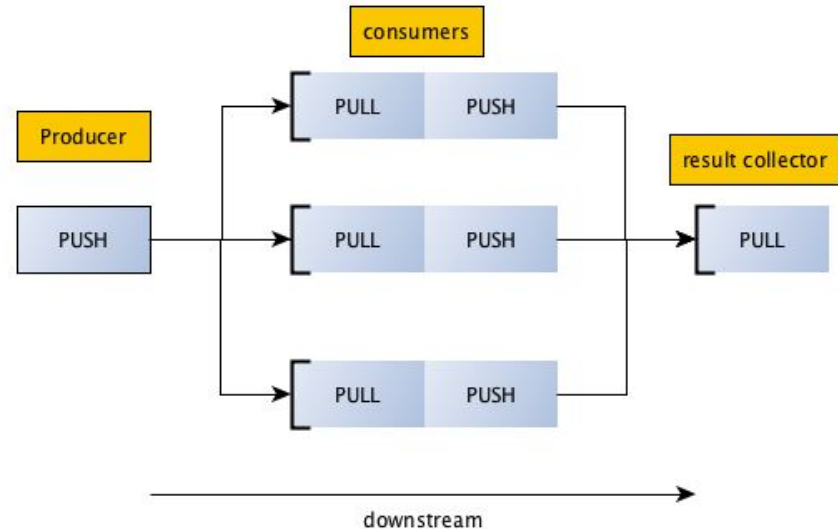
<https://zguide.zeromq.org/>



ZeroMQ - PUSH/PULL

Mensagens Unidirecionais

Distribuição de mensagens de maneira uniforme entre os clientes. Permite grande flexibilidade no fluxo de workloads e quantidade de Workers disponíveis. Filas manuseadas pelos sockets. > envio NÃO bloqueante

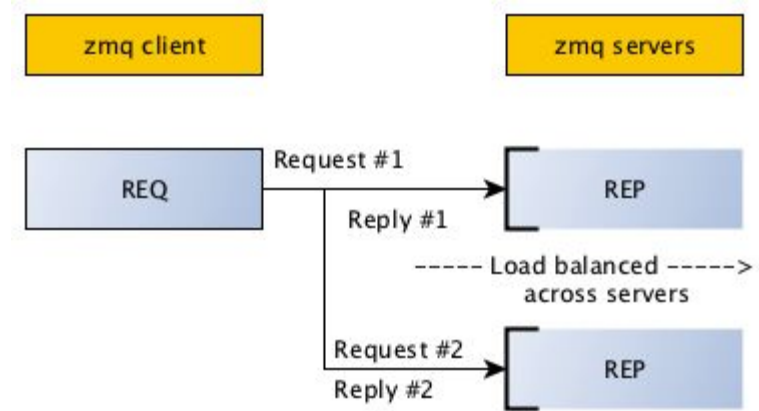


ZeroMQ - REQ/REP

Request / Reply.

Client REQ pode se conectar a múltiplos servidores, respostas são distribuídas conforme requisitadas.

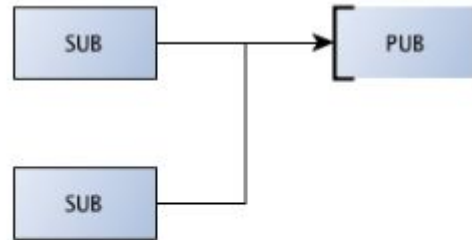
Pode pedir uma workload ou pedir para que a workload seja executada.



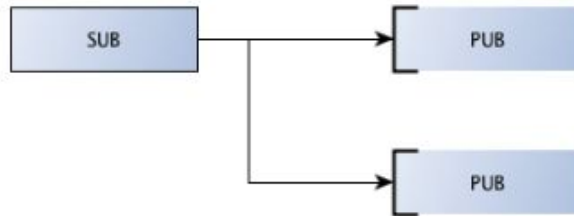
ZeroMQ - PUB/SUB

Publish / Subscribe

Completamente assíncrono, mensagens são publicadas sem a necessidade de conhecer ou sequer saber da existência de subscribers/listeners.



Scenario: #2

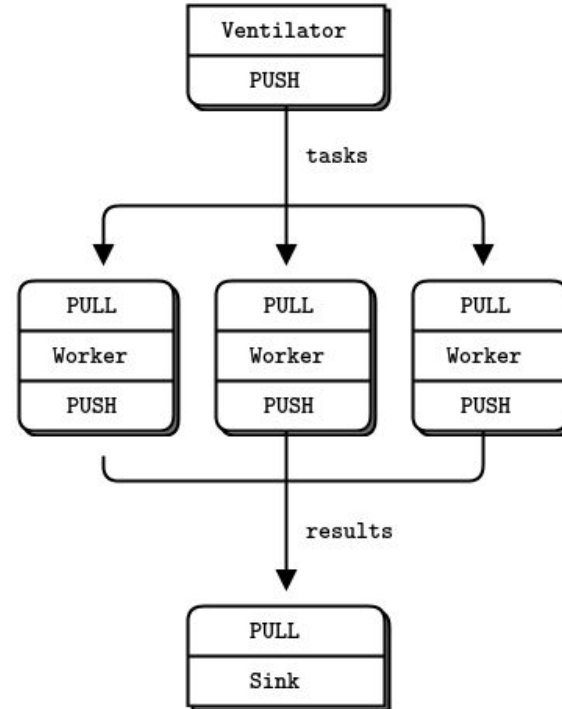


Scenario: #1

ZeroMQ - Arquiteturas

Ventilator:

Ventilator distribui todas as ordens de trabalho para todos os workers disponíveis no momento (simultaneamente). Os workers “despejam” os resultados do trabalho na Sink quando terminam.



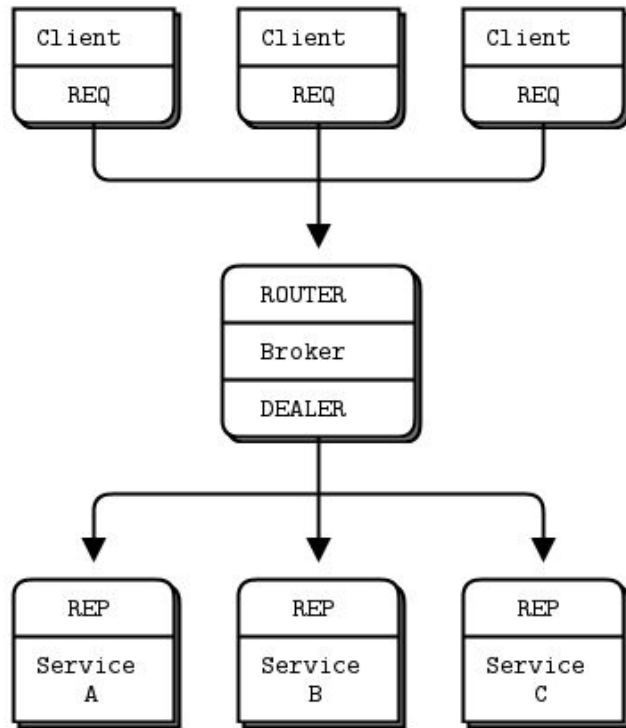
ZeroMQ - Arquiteturas

Request-Reply Broker - **SÍNCRONO**

Clientes conectam-se à Proxy e enviam uma ordem de trabalho.

Workers conectam-se à Proxy e **RECEBEM** serviço.

Workers respondem com resultado, Proxy redireciona resultado para Clientes.



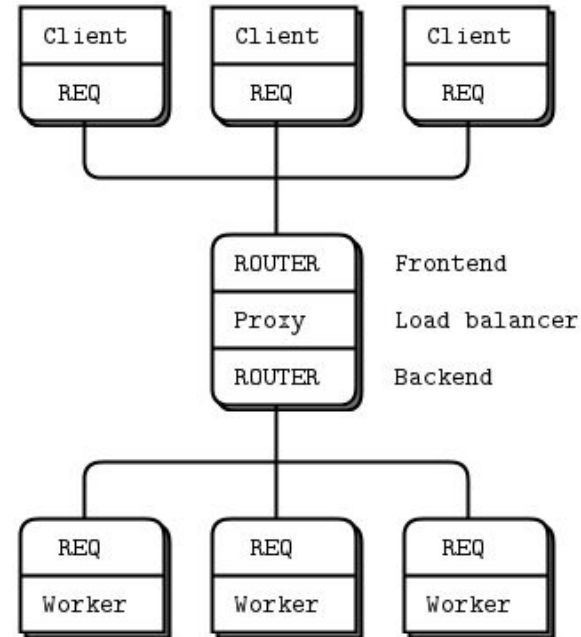
ZeroMQ - Arquiteturas

Load Balancing Broker - **ASSÍNCRONO**

Clientes conectam-se à Proxy e enviam uma ordem de trabalho. Proxy responde com confirmação.

Workers conectam-se à Proxy e **PEDEM** serviço.

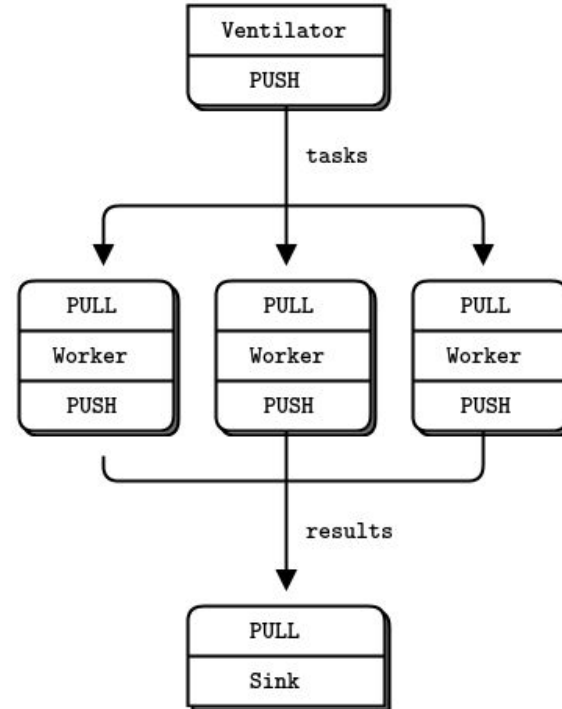
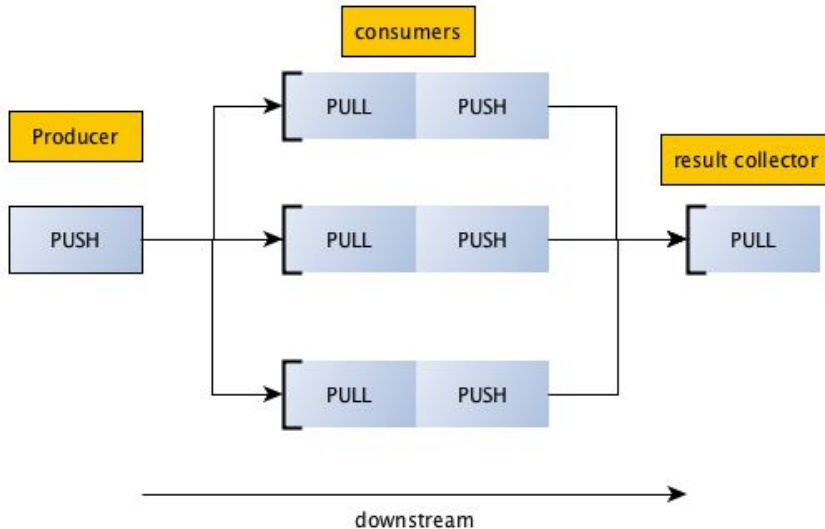
Proxy responde workers com ordem de serviço.



ZeroMQ - Arquiteturas

Vamos colocar em prática o Ventilator.

<https://t.ly/EH6hY>



Recursos

<https://learning-0mq-with-py zmq.readthedocs.io/en/latest/py zmq/basics.html>

<https://zguide.zeromq.org/>

<https://hub.docker.com/>

<https://docs.docker.com/>

<https://www.techworld-with-nana.com/>



Docker + Kubernetes

- Como escalar containers?
- Como garantir o trabalho coordenado entre os diferentes containers de uma aplicação?
- Como detectar containers com falhas e corrigir isso automaticamente?

Utilizando uma orquestração





- Também conhecido como K8s ou kube
- Desenvolvido originalmente pela Google
- Mantido pela Cloud Native Computing Foundation
- Escrito em **Go**
- Open **sourceCluster** com máquina **Master** e **Nodes**
- Criação de objetos através de arquivos no formato **YAML**
- Diversas funcionalidades para gerenciamento
- **kubectl** → ferramenta de linha de comando
- **Minikube** → ambiente de testes



gerenciamento de containers

- Orquestração
- Auto recuperação
- Reinício
- Replicação
- Escalonamento

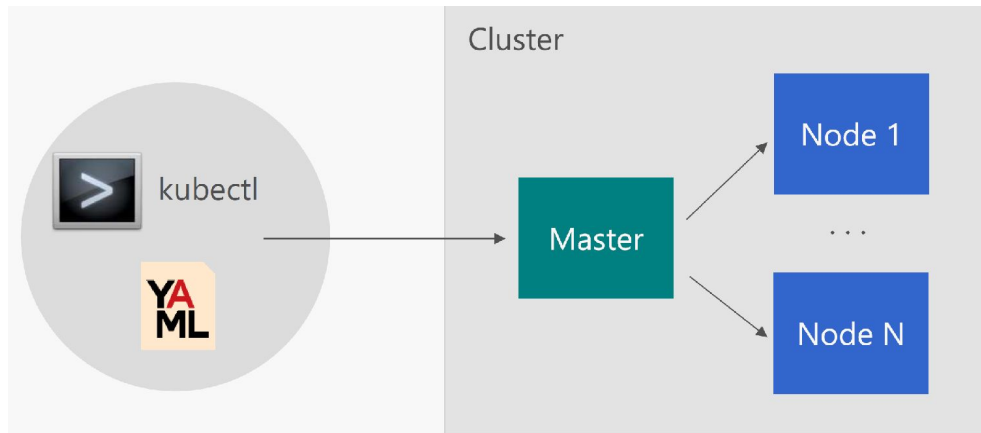


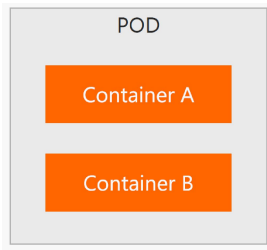
- Master

- Máquina que controla os **Nodes (Nós)**
- Responsável pelas atribuições de tarefas aos **Nodes**

- Nodes

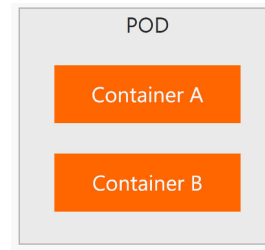
- Máquinas que realizam as tarefas atribuídas pelo **Master**





- Pod

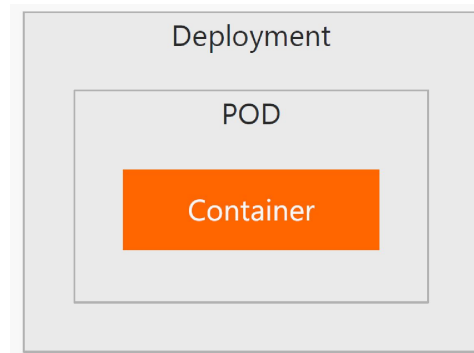
- Grupo de um ou mais containers
- implantados em um **Node (Nó)**
- Compartilham o mesmo endereço IP,
- IPC, nome do host e outros recursos

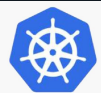




- Deployment

- Abstração de um **Pod** com recursos adicionais
- Conta com gerenciamento de estados





- Service

- Objeto mais estável (**Pods** são criados ou removidos continuamente)
- Cuidará do acesso aos **Pods**, funcionando como um **Load Balancer**

- Replication Controller

- Controla quantas cópias idênticas de um **Pod** serão executadas e em quais locais do **cluster**

- Kubelet

- Serviço que garante a inicialização e execução dos containers nos **Nodes**

