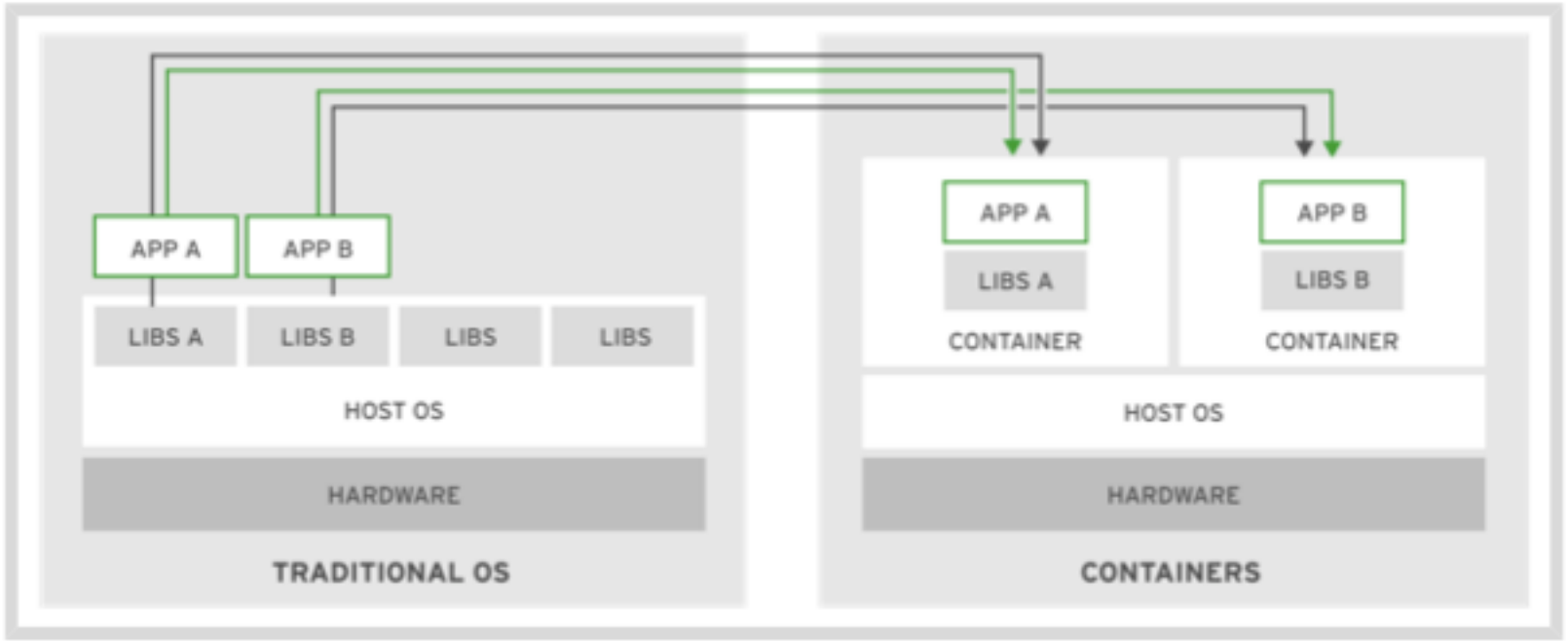
Docker – Introdução

DevOps
Mão na
Massa

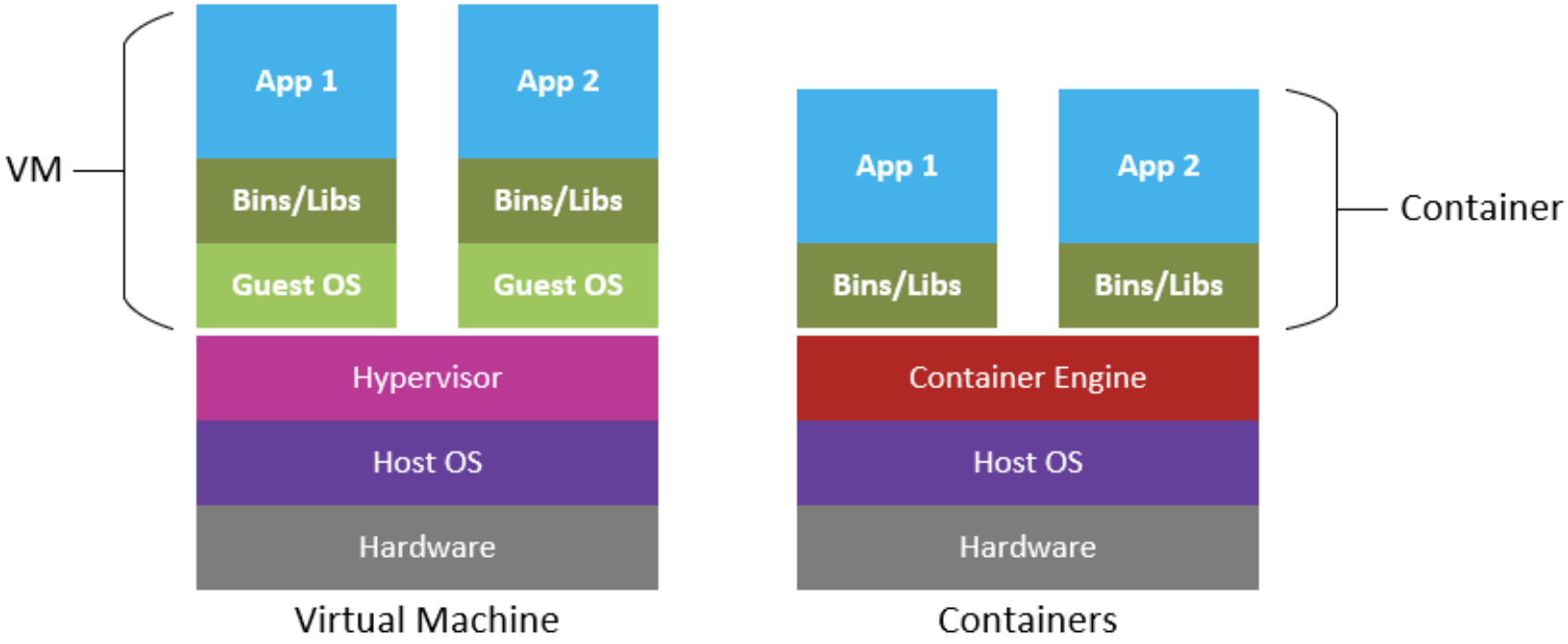
O que é e para que serve?

- Um processo executado no sistema operacional, porém isolado de todos os outros processos (cpu, memória, rede, disco, etc).
- Este isolamento permite também a separação das dependências da aplicação (versão do Java, libs de S.O, etc).
- Manutenção de muitas apps em um mesmo host dificulta a operação

Docker – Diferença entre S.O tradicional e Containers



Docker – Diferença entre VM e Containers

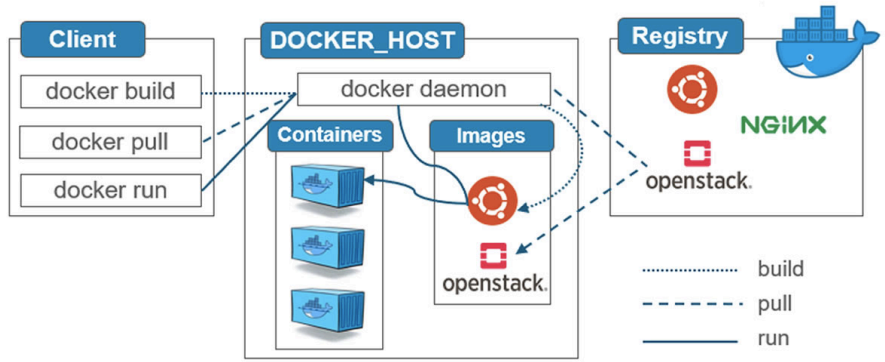


Docker – Vantagens

- **Baixo consumo de hardware**: utilização do hardware mais otimizada comparado com VM.
- **Portabilidade**: A frase “funciona na minha máquina” deixa de ser justificativa para problemas em ambientes produtivos. O mesmo container pode ser utilizado desde a máquina do Dev até PROD.
- **Reusabilidade**: É possível utilizar o mesmo container para diversos ambientes.
- **Microserviços**: Aderente a arquitetura de microserviços.

Docker – Arquitetura

- **Images**: Pacote com um sistema de arquivos com todas as suas dependências (libs de S.O, processos que serão executados, kernel, etc)
- **Container:** é um processo que executa uma imagem. A imagem é imutável mesmo após um container ser iniciado.
- **Registry**: Repositório de imagens
  - https://hub.docker.com/



Docker – conceitos básicos

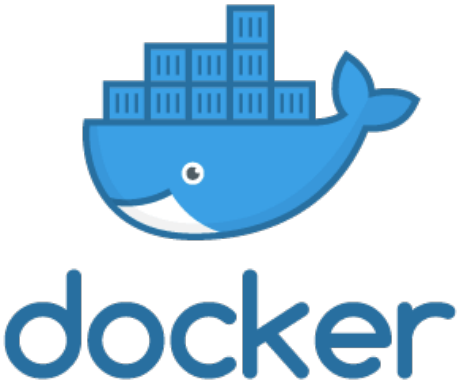
- **Dockerfile**: Arquivo texto simples com formato de script utilizado para criar uma imagem docker.
- **docker build**: Comando executado para criar uma imagem de container. Necessário criar um DockerFile.
  - **docker build -t getting-started .**
- **docker run:** Comando executado para iniciar um container.
  - **docker run -dp 3000:3000 getting-started**
- **docker pull: download da imagem docker do registry.**
  - **docker pull debian**

Dockerfile

```
FROM node:12-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "/app/src/index.js"]
```

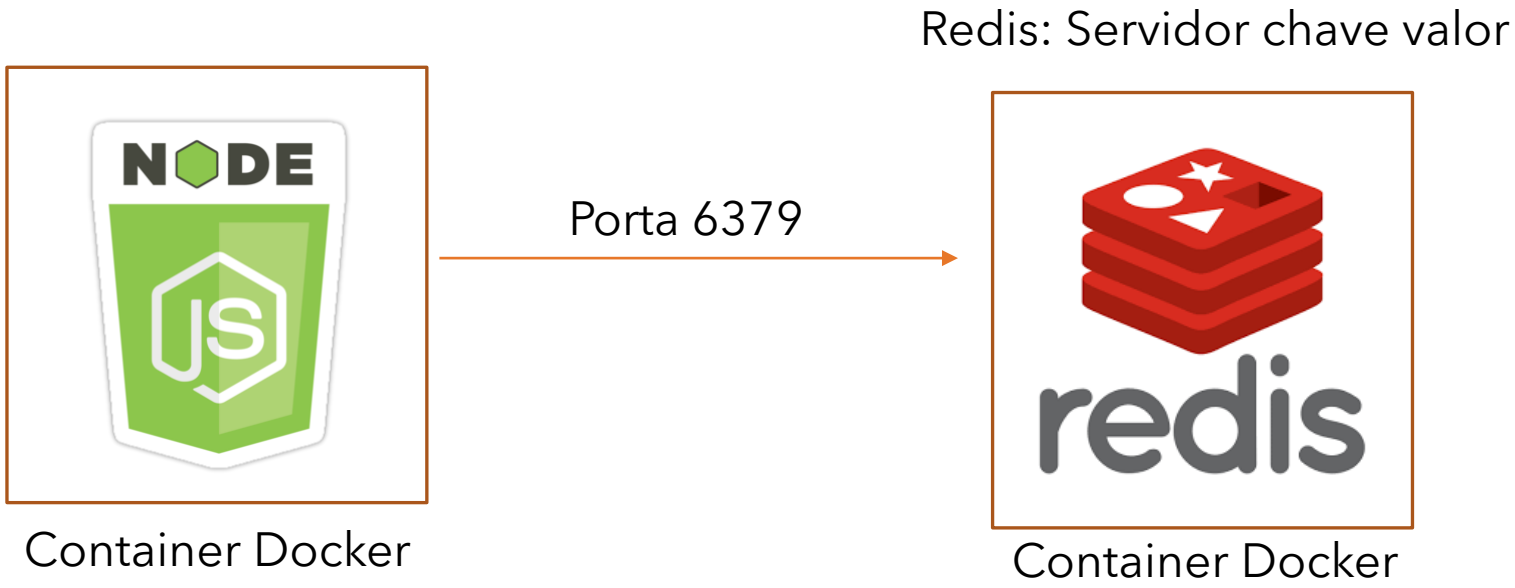
Docker – mão na massa

- Caso de uso:
  - Criar um server via Vagrant (docker-lab).
  - Instalar docker em docker-lab – utilizar script **provision.sh**
  - Utilizar duas imagens docker, uma para banco e outra para app.
  - Expor portas 8080 e 3306 para App e Banco respectivamente.
  - Conectar serviço Java (notes) com banco de dados.



Docker Mão na massa

1. Compilar aplicação Node.JS
2. Conectar NodeJS ao Redis – container apartado

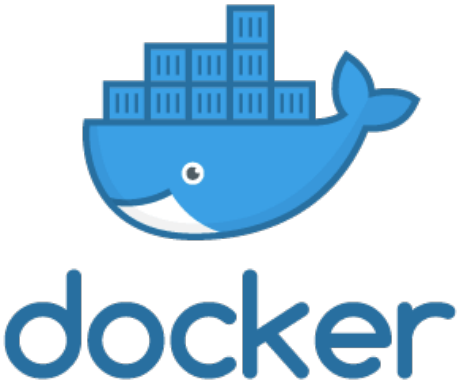


Docker – mão na massa: MariaDB

- Criar rede interna:
  - docker network create devops
- Criar diretório /root/docker/mariadb/datadir
  - diretório de persistência de dados
- Docker MariaDB:
  - docker run --net devops --name mariadb -v /root/docker/mariadb/datadir:/var/lib/mysql -e MARIADB_ROOT_PASSWORD=devopsmaonamassa –-env MARIADB_DATABASE=notes -d mariadb:latest
  - docker exec –it mariadb /bin/bash
  - mysql –uroot -pdevopsmaonamassa
  - show databases;
  - use notes;
  - show tables;

Docker – Comandos de administração

- docker ps – lista de containers (runtime)
- docker stop – para um container
- docker ps –a (exibe containers stop)
- docker rm (apaga um container)
- docker rm –f CONTAINER_ID – para e apaga container
- docker images – lista imagens docker já baixadas
- docker rmi IMAGE_NAME – Apaga imagem



Docker – mão na massa: App Java

- Docker OpenJDK:

  Dockerfile:

```
FROM openjdk:8-jdk-alpine
RUN addgroup -S notes && adduser -S notes -G notes
USER notes:notes
ARG JAR_FILE=*.jar
COPY ${JAR_FILE} easy-note.jar
COPY application.properties application.properties
ENTRYPOINT ["java","-jar","/easy-note.jar"]
```

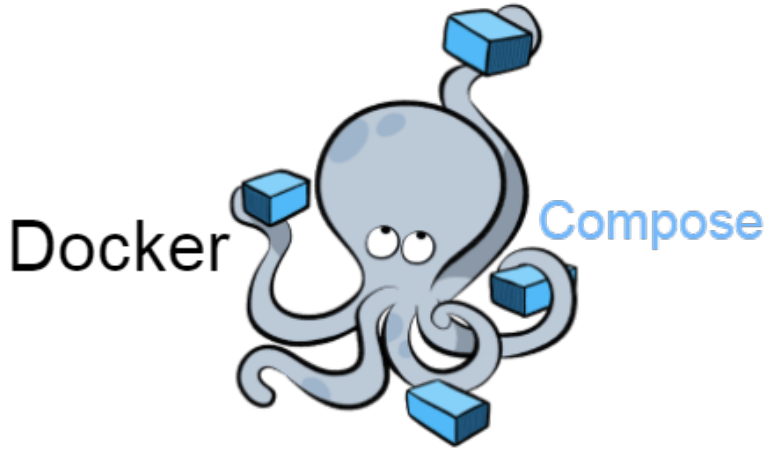
- Build da imagem:
  - **docker build -t devops/notes-docker .**
- Iniciar o container:
  - **docker run --network devops --hostname app -p 8080:8080 -d devops/notes-docker**

Docker – mão na massa: App Java

- Inserir um registro através do lab-docker:
  curl -H "Content-Type: application/json" --data @note.json http://localhost:8080/api/notes
- Recuperar todos as notas:
  - **curl http://localhost:8080/api/notes**
- Apagar um registro:
  - **curl -X DELETE -H "Content-Type: application/json" http://app01:8080/api/notes/1**

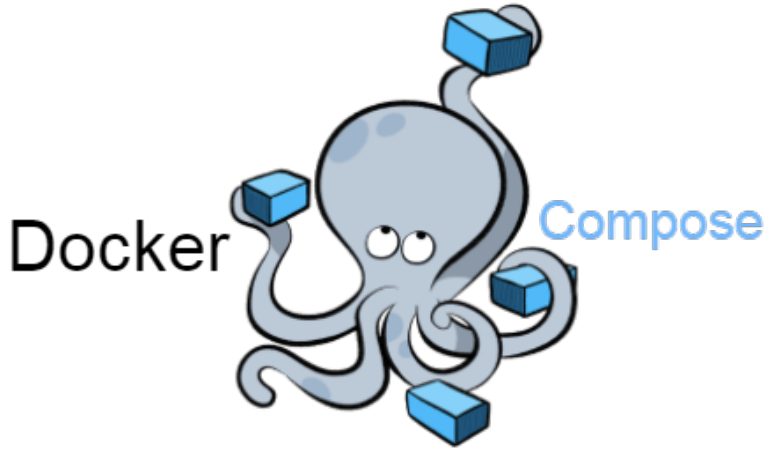
Docker Compose

- Ferramenta de definição e execução de aplicações com múltiplos containers.
- Arquivo de configuração do tipo YAML.
- Apenas um comando para subir todo o ambiente.
- Necessita do Dockerfile quando o build da imagem se faz necessário
- Se encaixa no modelo de IaC – versionamento da infraestrutura
- Automação no processo de testes (Continuos Integration)
  - **docker-compose up -d** (subida do ambiente)
  - **./run_tests** (execução dos testes)
  - **docker-compose down** (shutdown do ambiente)
- Ciclo de vida:
  - Start, stop rebuild dos serviços
  - Verificar status dos serviços
  - Monitoria de logs



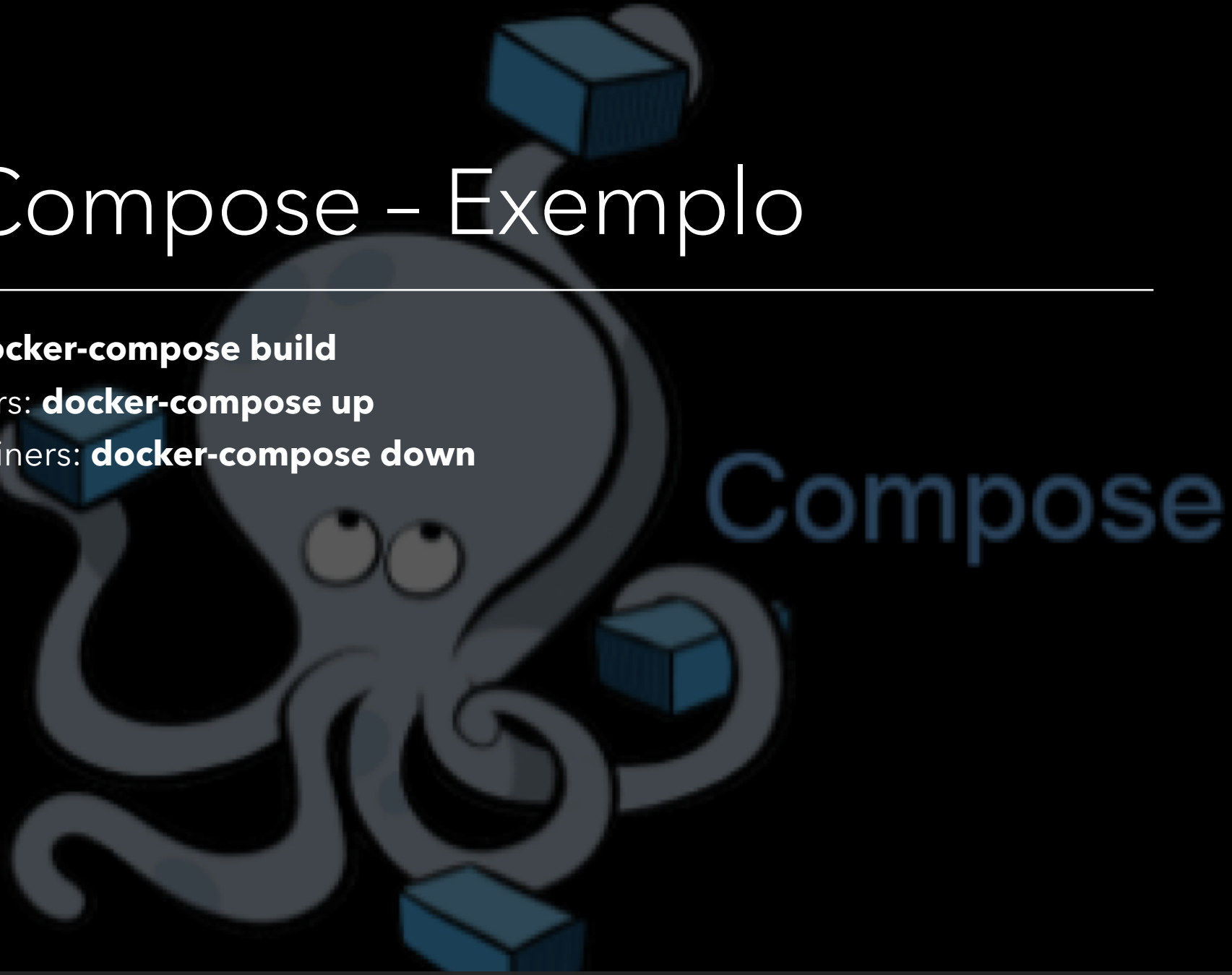
Docker Compose – Mão na massa

- Instalação:
  - **sudo curl -L "https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose**
- Mudar permissão para execução: **sudo chmod +x /usr/local/bin/docker-compose**
- Criar link simbólico (execução de qualquer diretório): **sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose**
- Validar instalação: **docker-compose --version**



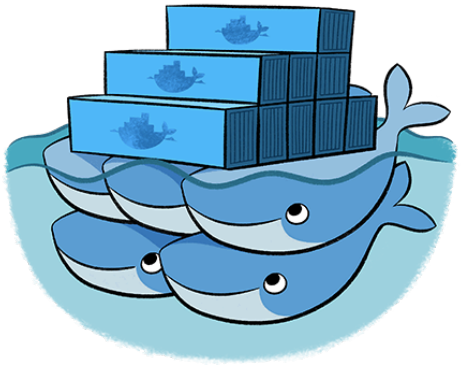
Docker Compose – Exemplo

- Build de imagem: **docker-compose build**
- Subida dos containers: **docker-compose up**
- Shutdown dos containers: **docker-compose down**



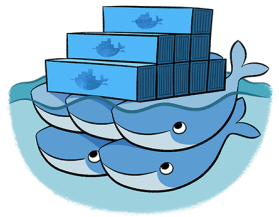
Docker Swarm - Conceitos

- Prover alta disponibilidade
- Orquestração de Containers
- Cluster nativo (built in Docker)
- Simples, fácil instalação
- Limitado comparado com outros produtos (K8s, OpenShift, etc).



Docker Swarm – Arquitetura

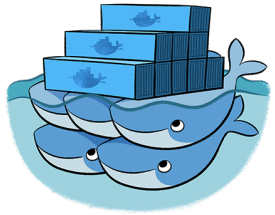
- Arquitetura
  - **Nodes**: Instancias de docker engine que participam de um cluster swarm.
  - **Manager Node**: Executam a orquestração e gerenciamento do cluster (somente um manager por cluster).
  - **Worker Node**: Recebe e executa tarefas disparadas pelo manager. Manager node também é um worker node (default).
  - **Service**: definição de tarefas para serem executadas no manager ou worker nodes.
  - **Task**: processo executado dentro do container.
  - **Load Balancer**: Expor services disponíveis para o "mundo externo"



Docker Swarm – Mão na massa

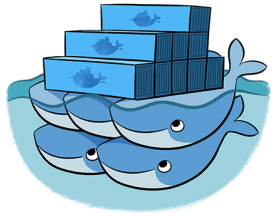
```
Vagrant.configure("2") do |config|

  config.vm.provision "shell", inline: "echo Config swarm nodes..."
  config.vm.define "manager" do |manager|
    manager.vm.box = "centos/7"
    manager.vm.hostname = "manager"
    manager.vm.provision "shell", path: "provision.sh"
    manager.vm.network "private_network", ip: "192.168.1.2"
  end

  config.vm.define "worker1" do |worker1|
    worker1.vm.box = "centos/7"
    worker1.vm.hostname = "worker1"
    worker1.vm.provision "shell", path: "provision.sh"
    worker1.vm.network "private_network", ip: "192.168.1.3"
  end

  config.vm.define "worker2" do |worker2|
    worker2.vm.box = "centos/7"
    worker2.vm.hostname = "worker2"
    worker2.vm.provision "shell", path: "provision.sh"
    worker2.vm.network "private_network", ip: "192.168.1.4"
  end

end
```

- **Obs.: Mais de um server em um mesmo Vagrantfile**
- Um host como manager, dois hosts como worker.
- Rede interna criada para comunicação do cluster.



Docker Swarm – Criando o cluster

- Executar no host manager:
  - **docker swarm init --advertise-addr 192.168.1.2**
- Executar nos hosts worker1 e worker2:
  - **docker swarm join --token <TOKEN> 192.168.1.2:2377**
- Executar no host manager:
  - **docker node ls – listar nodes do cluster**



Docker Swarm – Criando service

- Iniciar um serviço no cluster:**docker service create --name demo --publish 80:80 nginx**
- Listar service criado: **docker service ls**
- Listar detalhes do service: **docker service ps demo**
- Escalar o serviço: **docker service scale demo=3**
- Visualizar o serviço distribuído pelo cluster: **docker service ps demo**
- Abrir página do Nginx (na máquina fisica): **http://localhost:8090**