>>> >>> Docker

Name: (Universidade de São Paulo)

Date: December 1, 2022

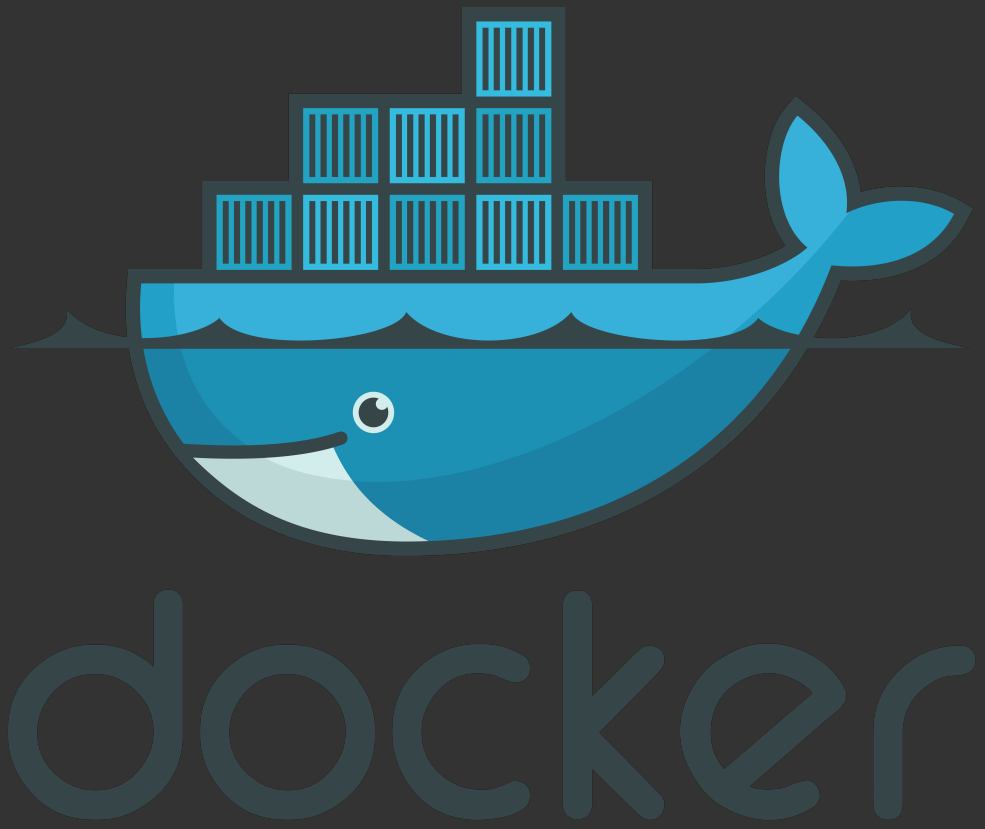


Figure: Docker logo

>>> Integrantes

* Fernanda Elimelek N° 11208155
* Gabriel Medeiros Jospin N° 11796020
* Gabriela Jie Han N° 11877423
* Guilherme Kendji Ishikawa N° 11914650
* Gustavo Tsuyoshi Ariga N° 11857215
* Henrique Tsuyoshi Yara N° 11796083

>>> Índice

>>> O que é docker?

Gerenciar a infra-estrutura:

* Ambientes "isolados" usando container;

>>> O que é docker?

Gerenciar a infra-estrutura:

* Ambientes "isolados" usando container;
* Agilizar as entregas das aplicações;

>>> O que é docker?

Gerenciar a infra-estrutura:

* Ambientes "isolados" usando container;
* Agilizar as entregas das aplicações;
* Aplicações seguras;

>>> O que é docker?

Gerenciar a infra-estrutura:

* Ambientes "isolados" usando container;
* Agilizar as entregas das aplicações;
* Aplicações seguras;
* Portabilidade;

>>> O que é um container?

* Containers:
    * Usa o *kernel* da máquina *host*;

>>> O que é um container?

* Containers:
    * Usa o *kernel* da máquina *host*;
    * Contém:
        * *Softwares*;

>>> O que é um container?

* Containers:
    * Usa o *kernel* da máquina *host*;
    * Contém:
        * *Softwares*;
        * Bibliotecas;

>>> O que é um container?

- Containers:
  - Usa o *kernel* da máquina *host*;
  - Contém:
    - *Softwares*;
    - Bibliotecas;
  - Leves (Somente o necessário);

>>> O que é um container?

- Containers:
  - Usa o *kernel* da máquina *host*;
  - Contém:
    - *Softwares*;
    - Bibliotecas;
  - Leves (Somente o necessário);
  - Execução quase igual à execução nativa;

>>> Características do container

* Portabilidade:
    * Fácil compartilhar;

>>> Características do container

* Portabilidade:
  * Fácil compartilhar;
  * Ambiente igual para todos;

>>> Características do container

* Portabilidade:
  * Fácil compartilhar;
  * Ambiente igual para todos;
  * A mesma execução para todos;

>>> Características do container

* Portabilidade:
  * Fácil compartilhar;
  * Ambiente igual para todos;
  * A mesma execução para todos;
* Alternativa viável para as máquinas virtuais (hypervisor):
  * Melhor uso da capacidade do seu computador;

>>> Características do container

* Portabilidade:
  * Fácil compartilhar;
  * Ambiente igual para todos;
  * A mesma execução para todos;
* Alternativa viável para as máquinas virtuais (hypervisor):
  * Melhor uso da capacidade do seu computador;
  * Ambientes grandes e densos;

>>> Características do container

* Portabilidade:
  * Fácil compartilhar;
  * Ambiente igual para todos;
  * A mesma execução para todos;
* Alternativa viável para as máquinas virtuais (hypervisor):
  * Melhor uso da capacidade do seu computador;
  * Ambientes grandes e densos;
  * Ambientes pequenos ou médios;

>>> Máquina virtual X Container

* Máquina Virtual
  * Virtualiza a camada de aplicação e a camada do SO;
* Container

>>> Máquina virtual X Container

* Máquina Virtual
  * Virtualiza a camada de aplicação e a camada do SO;
* Container
  * Virtualiza a camada de aplicação;

>>> Máquina virtual X Container

* Máquina Virtual
  * Virtualiza a camada de aplicação e a camada do SO;
  * Mais pesado (Na casa dos Gigabytes);
* Container
  * Virtualiza a camada de aplicação;

>>> Máquina virtual X Container

* Máquina Virtual
  * Virtualiza a camada de aplicação e a camada do SO;
  * Mais pesado (Na casa dos Gigabytes);
* Container
  * Virtualiza a camada de aplicação;
  * Mais leve (Na casa dos megabytes);

>>> Máquina virtual X Container

* Máquina Virtual
  * Virtualiza a camada de aplicação e a camada do SO;
  * Mais pesado (Na casa dos Gigabytes);
  * Demoram mais para inicializar;
* Container
  * Virtualiza a camada de aplicação;
  * Mais leve (Na casa dos megabytes);

>>> Máquina virtual X Container

* Máquina Virtual
  * Virtualiza a camada de aplicação e a camada do SO;
  * Mais pesado (Na casa dos Gigabytes);
  * Demoram mais para inicializar;
* Container
  * Virtualiza a camada de aplicação;
  * Mais leve (Na casa dos megabytes);
  * Inicializam rapidamente;

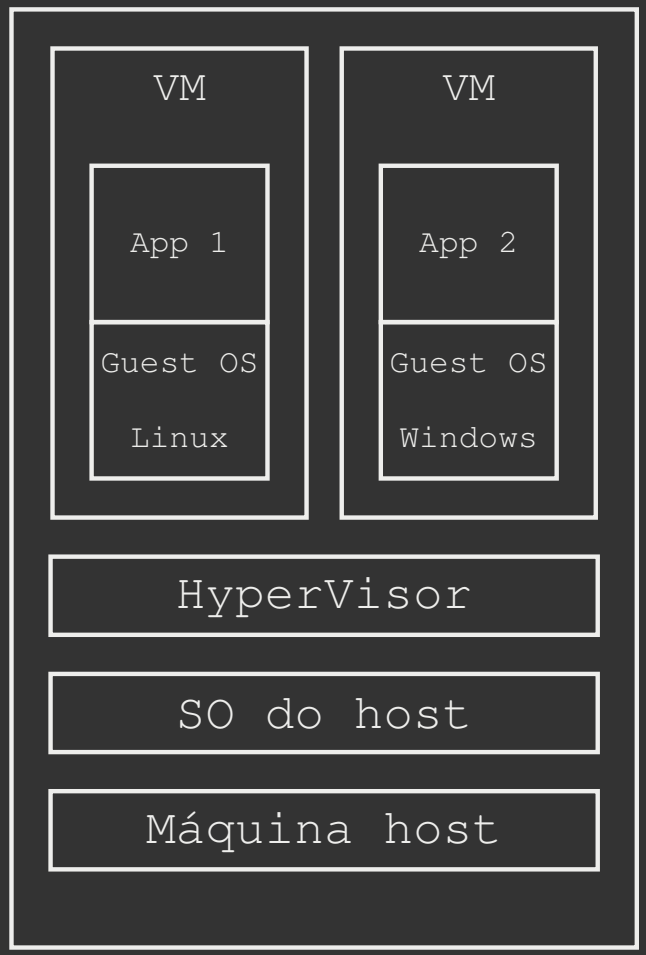
>>> Máquina virtual X Container

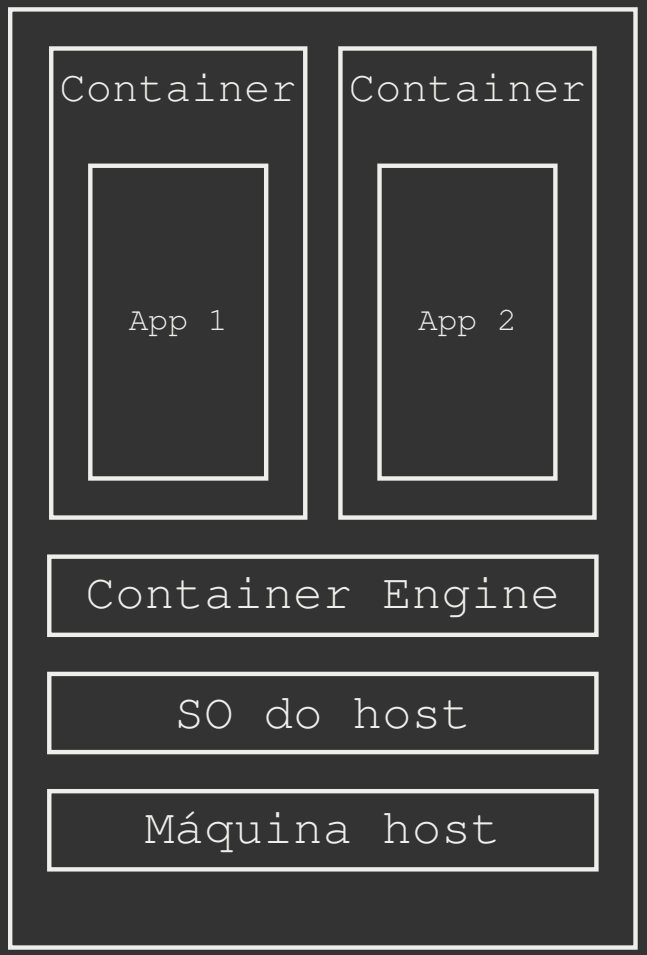
* Máquina Virtual
  * Virtualiza a camada de aplicação e a camada do SO;
  * Mais pesado (Na casa dos Gigabytes);
  * Demoram mais para inicializar;
  * Executa qualquer sistema operacional em qualquer Sistema Operacional;
* Container
  * Virtualiza a camada de aplicação;
  * Mais leve (Na casa dos megabytes);
  * Inicializam rapidamente;

>>> Máquina virtual X Container

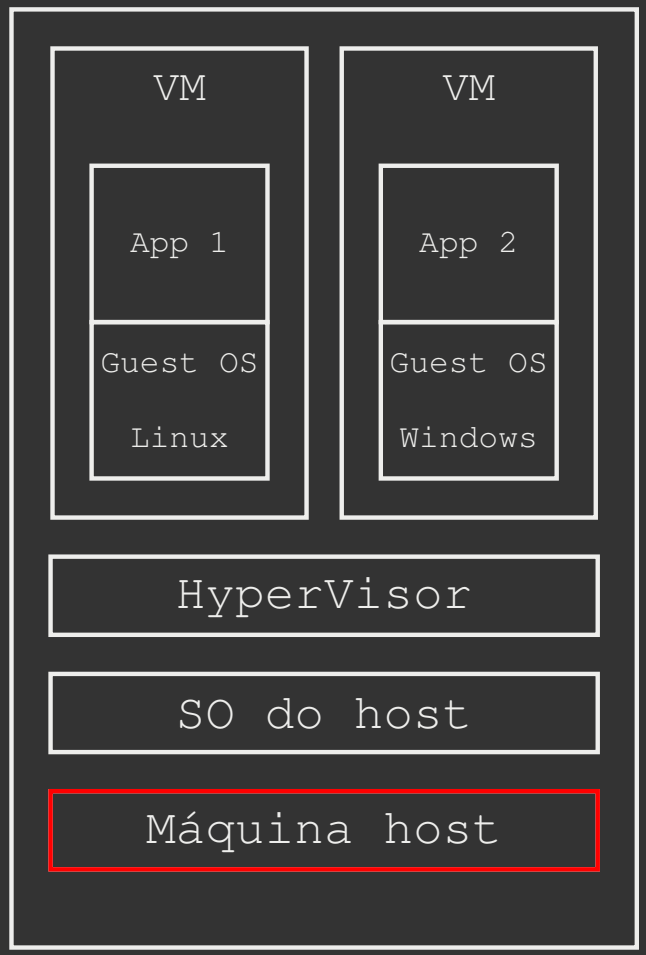
* Máquina Virtual
  * Virtualiza a camada de aplicação e a camada do SO;
  * Mais pesado (Na casa dos Gigabytes);
  * Demoram mais para inicializar;
  * Executa qualquer sistema operacional em qualquer Sistema Operacional;
* Container
  * Virtualiza a camada de aplicação;
  * Mais leve (Na casa dos megabytes);
  * Inicializam rapidamente;
  * O sistema operacional precisa ser compatível com a máquina principal ou usar uma camada de compatibilidade;

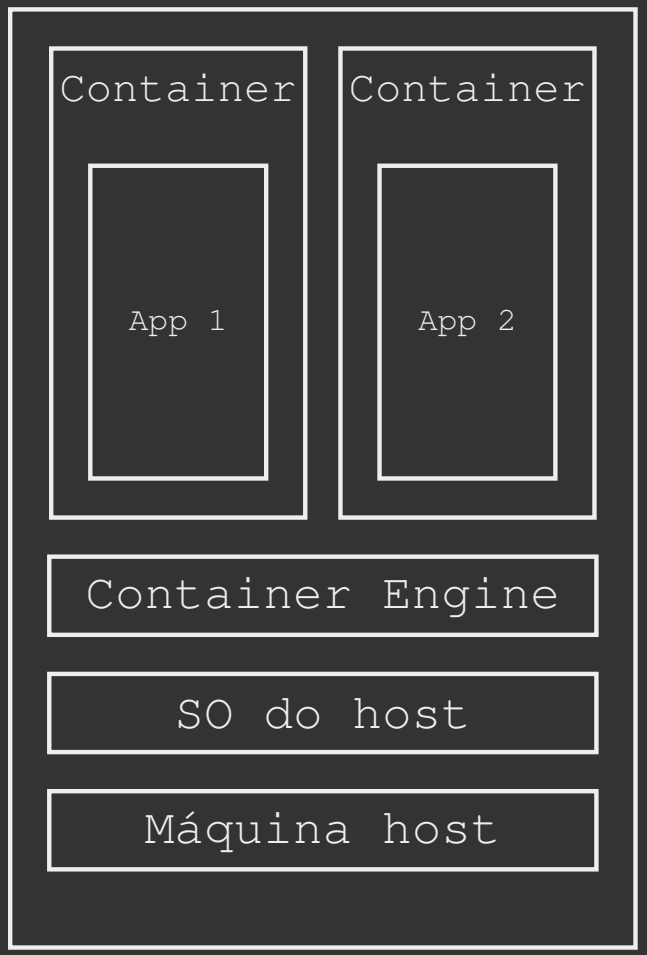
>>> Máquina virtual X Container (Imagens)



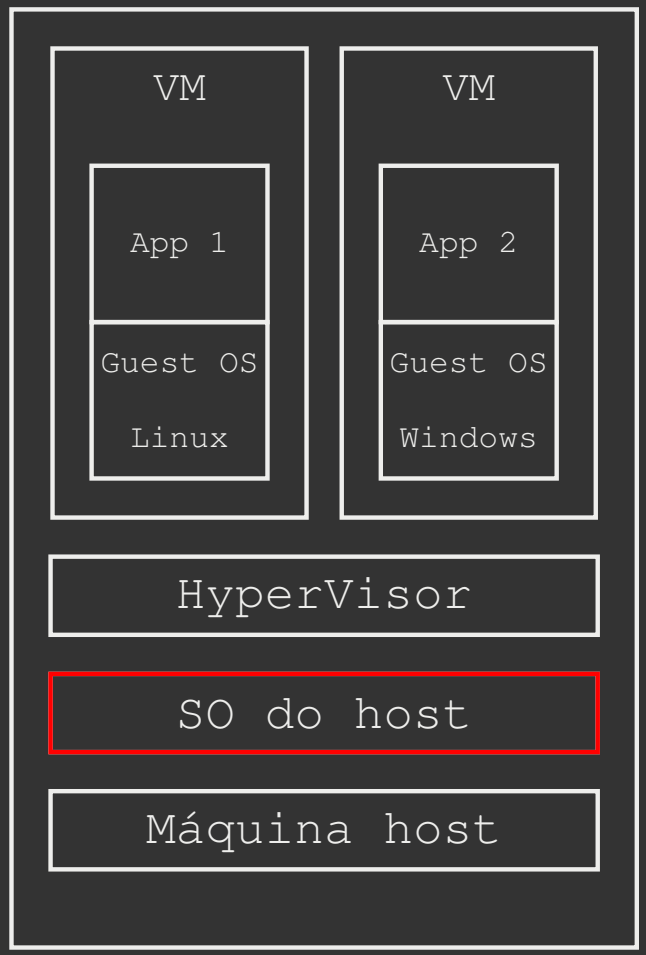


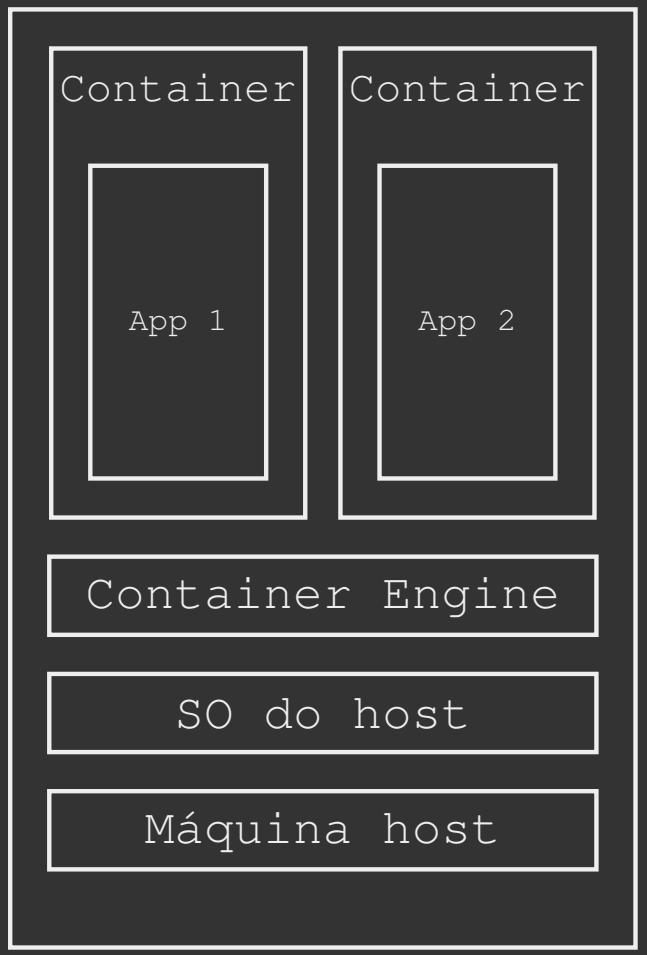
>>> Máquina virtual X Container (Imagens)



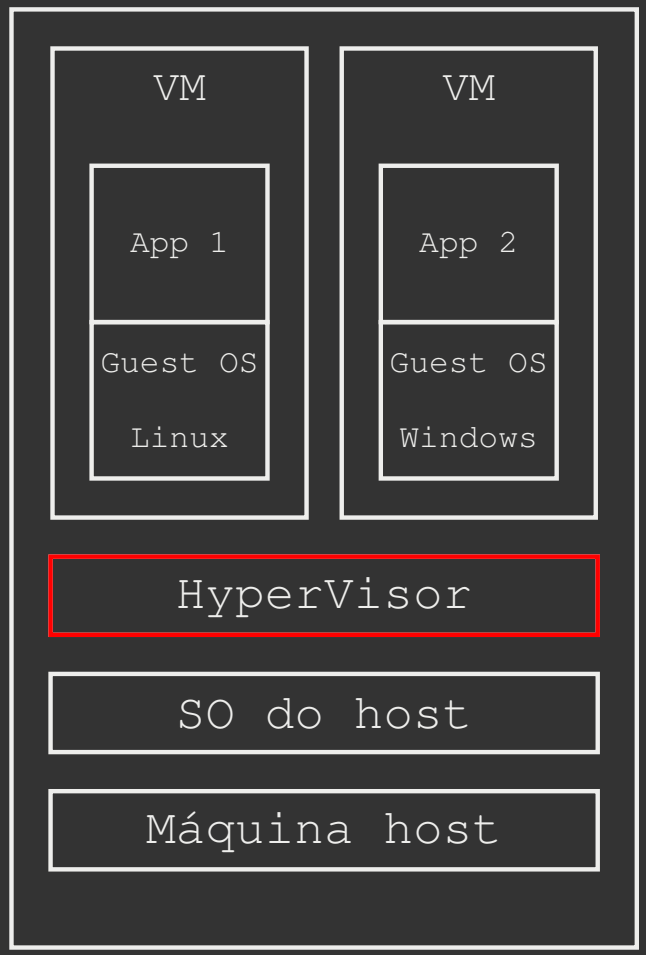


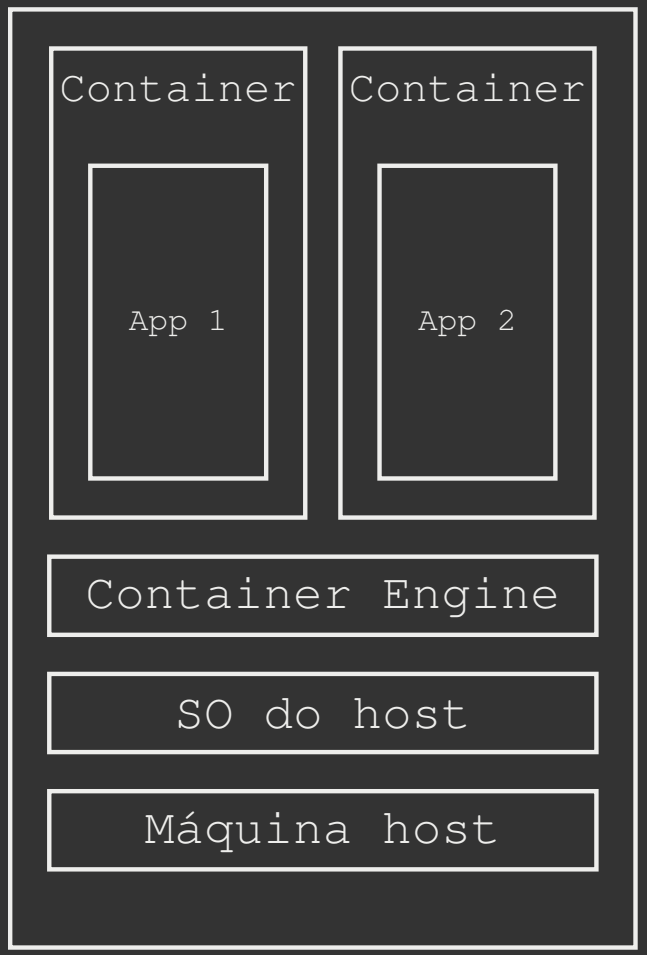
>>> Máquina virtual X Container (Imagens)



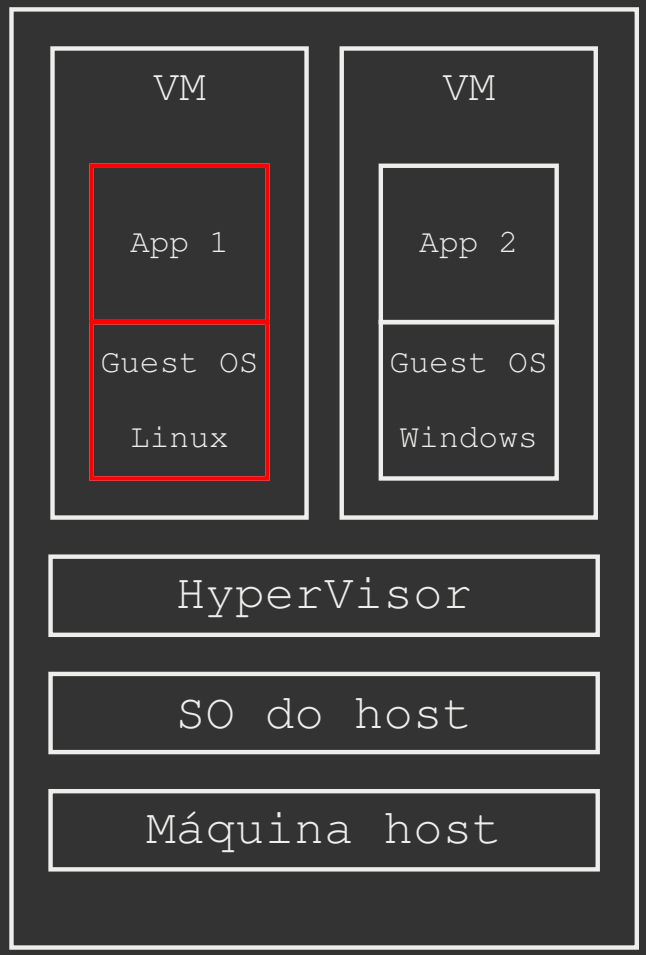


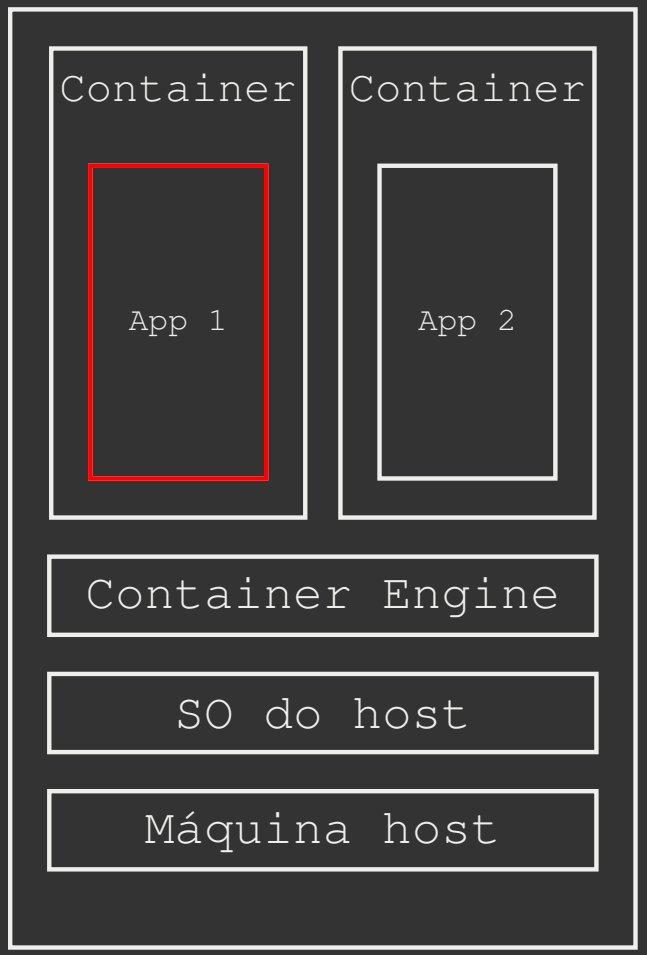
>>> Máquina virtual X Container (Imagens)





>>> Máquina virtual X Container (Imagens)





>>> Onde os containers podem ser encontrados?

* Repositórios privados:
    * Repostórios das empresas;

>>> Onde os containers podem ser encontrados?

* Repositórios privados:
    * Repostórios das empresas;
* Repositórios públicos:
    * Docker hub;

>>> Desenvolvimento e Execução - Objetivos

* Ambientes "isolados":
  * Não é necessário instalar softwares dependentes na máquina principal;

>>> Desenvolvimento e Execução - Objetivos

* Ambientes "isolados":
  * Não é necessário instalar softwares dependentes na máquina principal;
  * Evita conflitos de versão;

>>> Desenvolvimento e Execução - Objetivos

* Ambientes "isolados":
  * Não é necessário instalar softwares dependentes na máquina principal;
  * Evita conflitos de versão;
  * Abstrair a parte da infra-estrutura aplicação;

>>> Desenvolvimento e Execução - Objetivos

* Ambientes "isolados":
  * Não é necessário instalar softwares dependentes na máquina principal;
  * Evita conflitos de versão;
  * Abstrair a parte da infra-estrutura aplicação;
  * Toda a infraestrutura pode ser facilmente executada na máquinal local;

>>> Entrega - Objetivos

* Ambientes "isolados":
    * Ambiente de testes = Ambiente de produção;

>>> Entrega - Objetivos

* Ambientes "isolados":
    * Ambiente de testes = Ambiente de produção;
    * Entregas de software de forma mais rápida;
        * Não é necessário instalar dependências nas máquinas (↑ tempo);

>>> Entrega - CI/CD

* Docker é ótimo para *continuous delivery* e *continuous integration*:
  * Desenvolvedores fazem mudanças no código localmente;

>>> Entrega - CI/CD

* Docker é ótimo para *continuous delivery* e *continuous integration*:
  * Desenvolvedores fazem mudanças no código localmente;
  * No ambiente de teste são executados testes manuais e automáticos;

>>> Entrega - CI/CD

* Docker é ótimo para *continuous delivery* e *continuous integration*:
  * Desenvolvedores fazem mudanças no código localmente;
  * No ambiente de teste são executados testes manuais e automáticos;
  * Erros podem ser reproduzidos e arrumados usando docker;

>>> Entrega - CI/CD

* Docker é ótimo para *continuous delivery* e *continuous integration*:
  * Desenvolvedores fazem mudanças no código localmente;
  * No ambiente de teste são executados testes manuais e automáticos;
  * Erros podem ser reproduzidos e arrumados usando docker;
  * Então a imagem vai para o ambiente de produção;

>>> Entrega - Produção e escalabilidade

* Responsive deployment and scaling
    * Pode ser executado em:
        * Máquinas físicas;

>>> Entrega - Produção e escalabilidade

* Responsive deployment and scaling
    * Pode ser executado em:
        * Máquinas físicas;
        * Máquinas virtuais;

>>> Entrega - Produção e escalabilidade

* Responsive deployment and scaling
    * Pode ser executado em:
        * Máquinas físicas;
        * Máquinas virtuais;
        * Data centers;

>>> Entrega - Produção e escalabilidade

* Responsive deployment and scaling
    * Pode ser executado em:
        * Máquinas físicas;
        * Máquinas virtuais;
        * Data centers;
        * Nuvem;

>>> Entrega - Produção e escalabilidade

* Responsive deployment and scaling
    * Pode ser executado em:
        * Máquinas físicas;
        * Máquinas virtuais;
        * Data centers;
        * Nuvem;
        * Misto;

>>> Entrega - Produção e escalabilidade

* Responsive deployment and scaling
  * Pode ser executado em:
    * Máquinas físicas;
    * Máquinas virtuais;
    * Data centers;
    * Nuvem;
    * Misto;
  * Portabilidade → escalar os projetos ou retirar aplicações e serviços (Quase tempo real);

>>> Ciclo de vida

* Desenvolver: Aplicação e componentes isolados no container;

>>> Ciclo de vida

* Desenvolver: Aplicação e componentes isolados no container;
* Execução: Testar e distribuir containers;

>>> Ciclo de vida

* Desenvolver: Aplicação e componentes isolados no container;
* Execução: Testar e distribuir containers;
* Entrega: Ambiente de produção usando container ou serviço orquestrado;

>>> Arquitetura do docker

* Arquitetura cliente-servidor:
    * Cliente docker se comunica com o servidor do docker;

>>> Arquitetura do docker

* Arquitetura cliente-servidor:
  * Cliente docker se comunica com o servidor do docker;
  * Comunicação usando sockets UNIX ou pela interface de redes usando uma API REST;

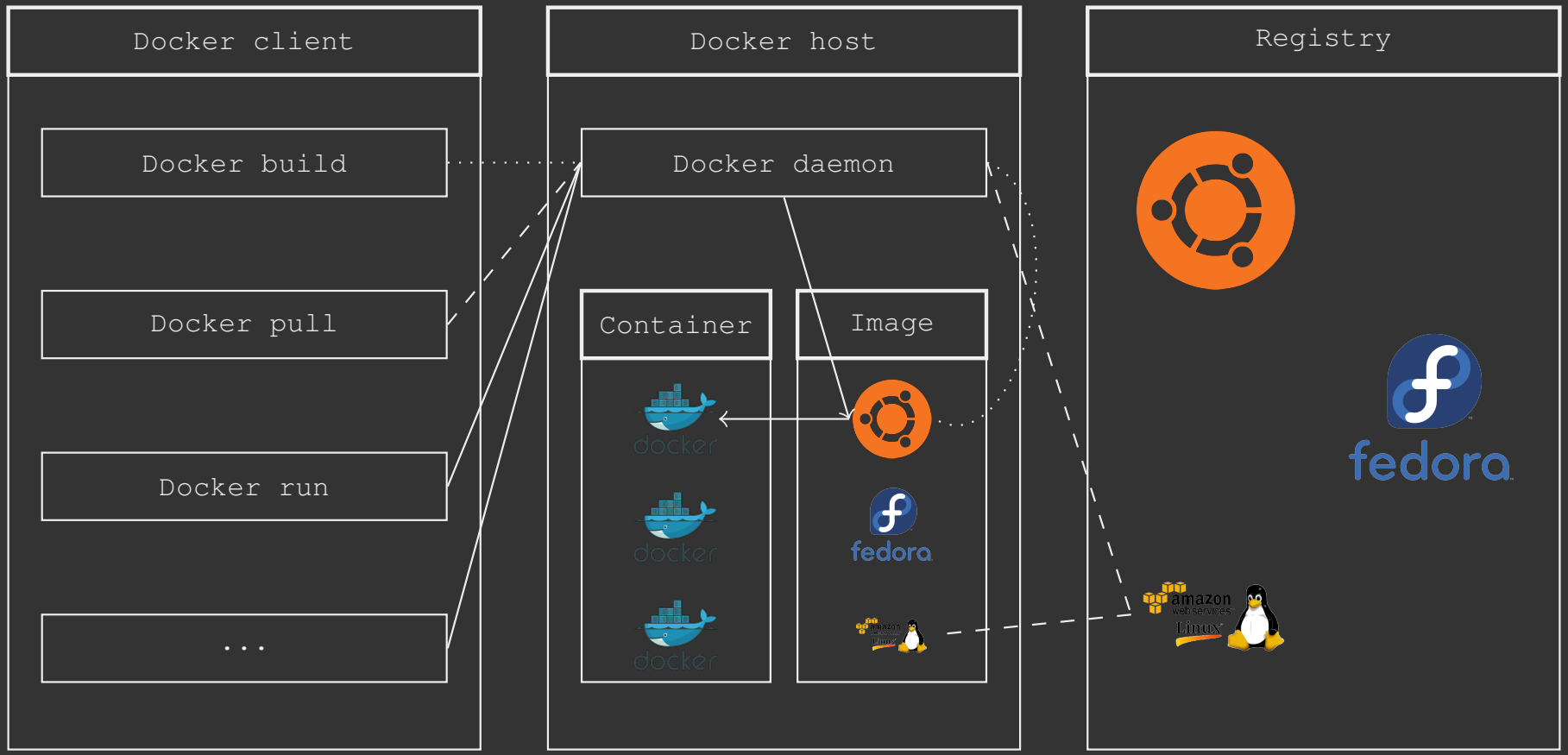
>>> Arquitetura do docker

* Arquitetura cliente-servidor:
  * Cliente docker se comunica com o servidor do docker;
  * Comunicação usando sockets UNIX ou pela interface de redes usando uma API REST;
  * O servidor é o responsável por criar imagens, rodar imagens e distribuir;

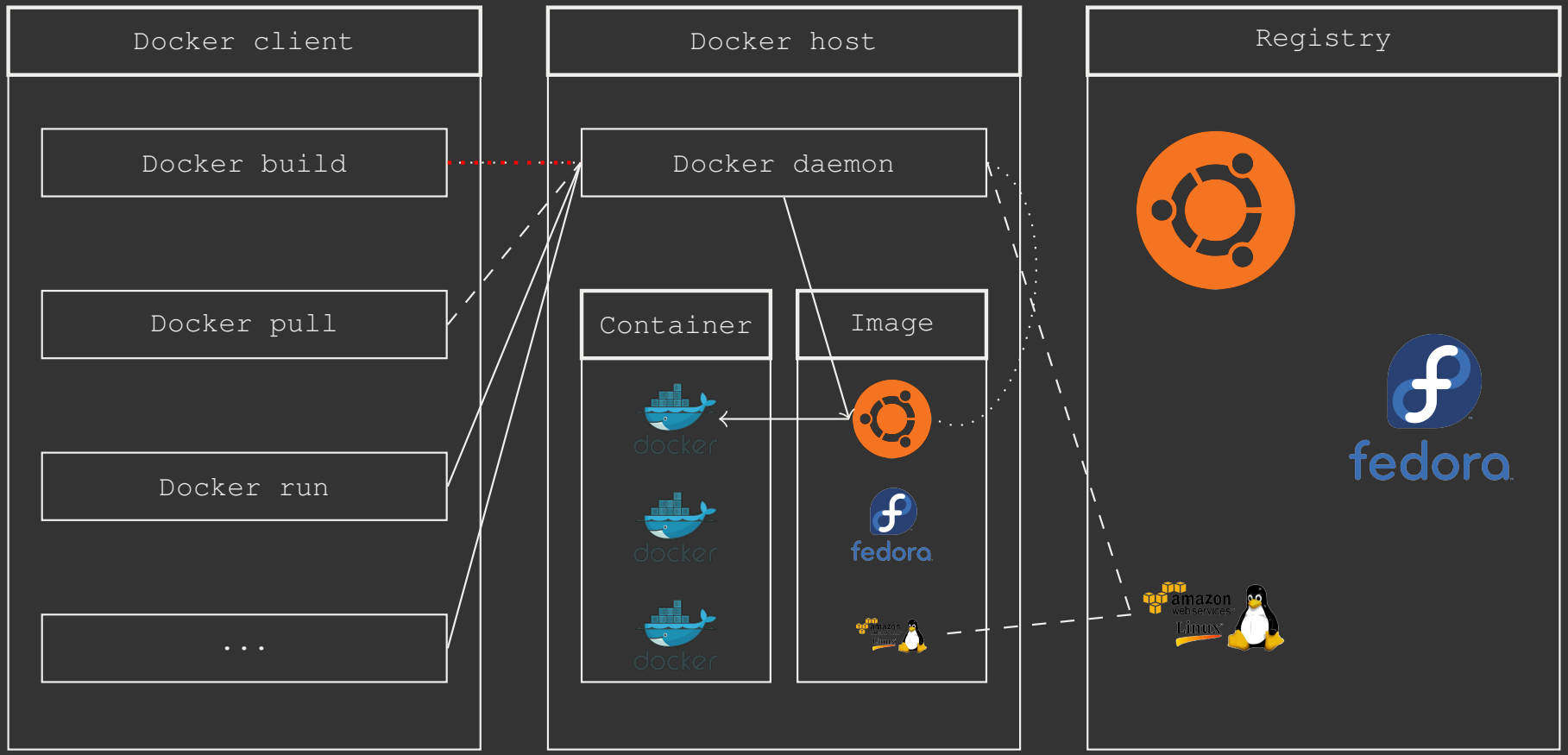
>>> Arquitetura do docker

* Arquitetura cliente-servidor:
  * Cliente docker se comunica com o servidor do docker;
  * Comunicação usando sockets UNIX ou pela interface de redes usando uma API REST;
  * O servidor é o responsável por criar imagens, rodar imagens e distribuir;
  * Podem rodar na mesma máquina ou em máquinas diferentes;

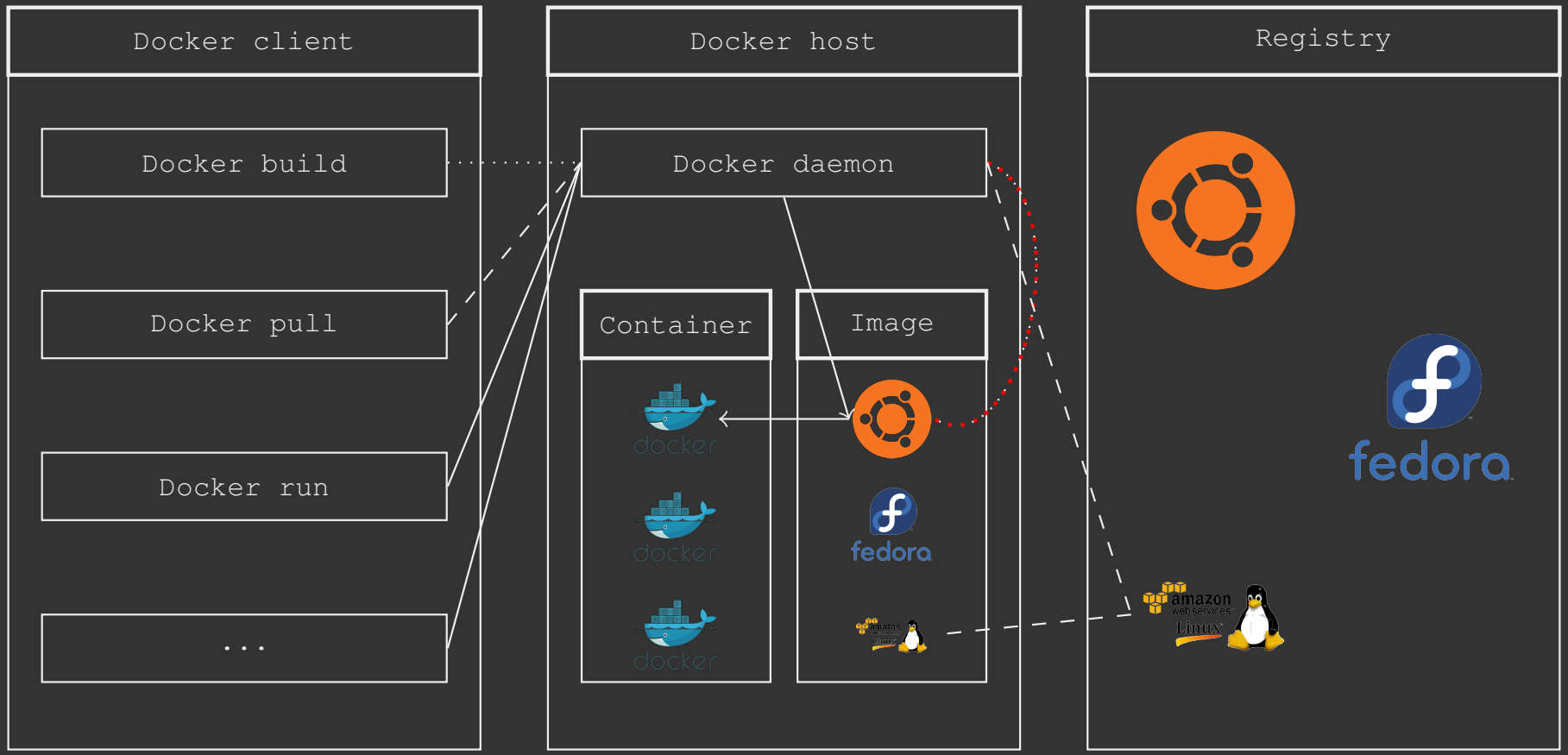
>>> Arquitetura do docker



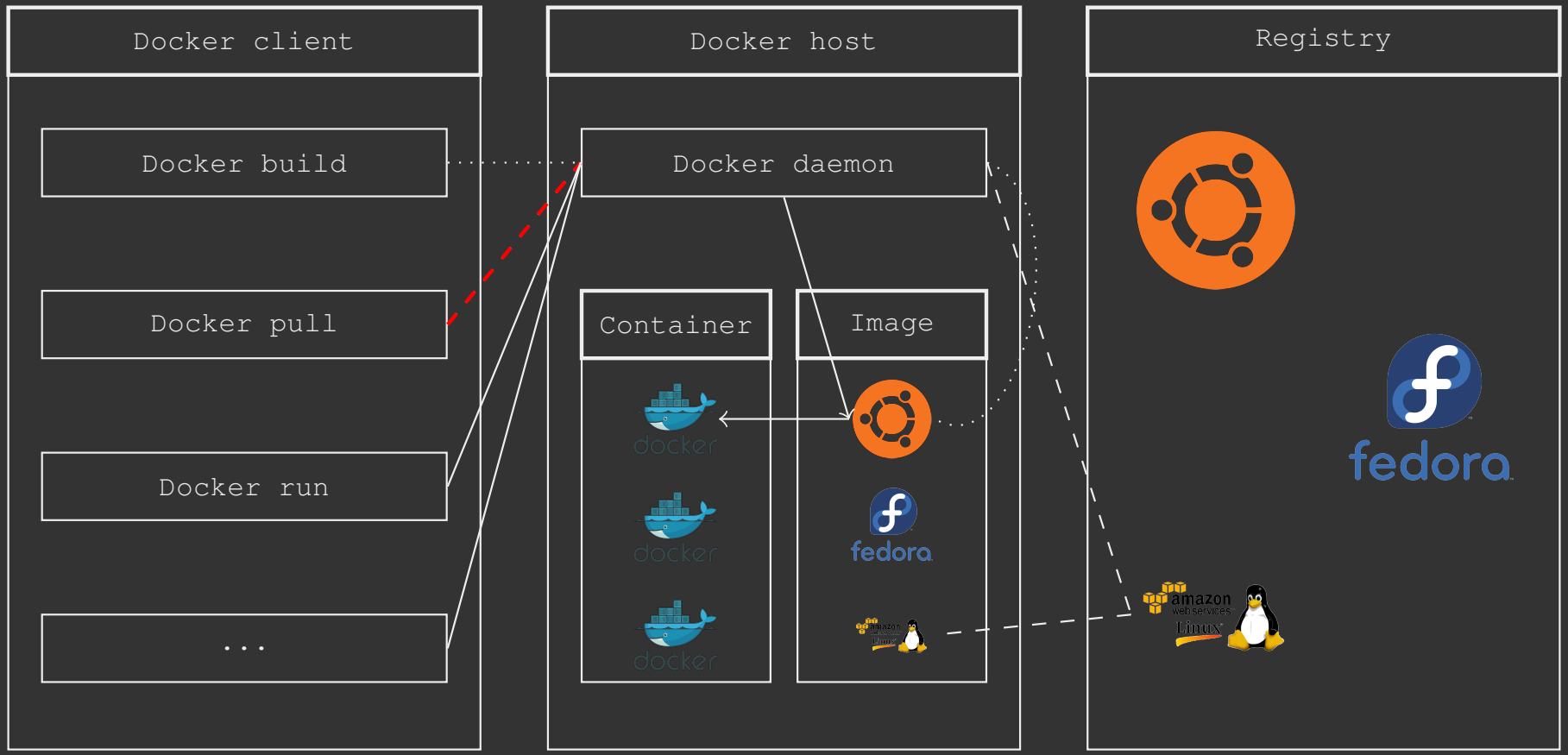
>>> Arquitetura do docker



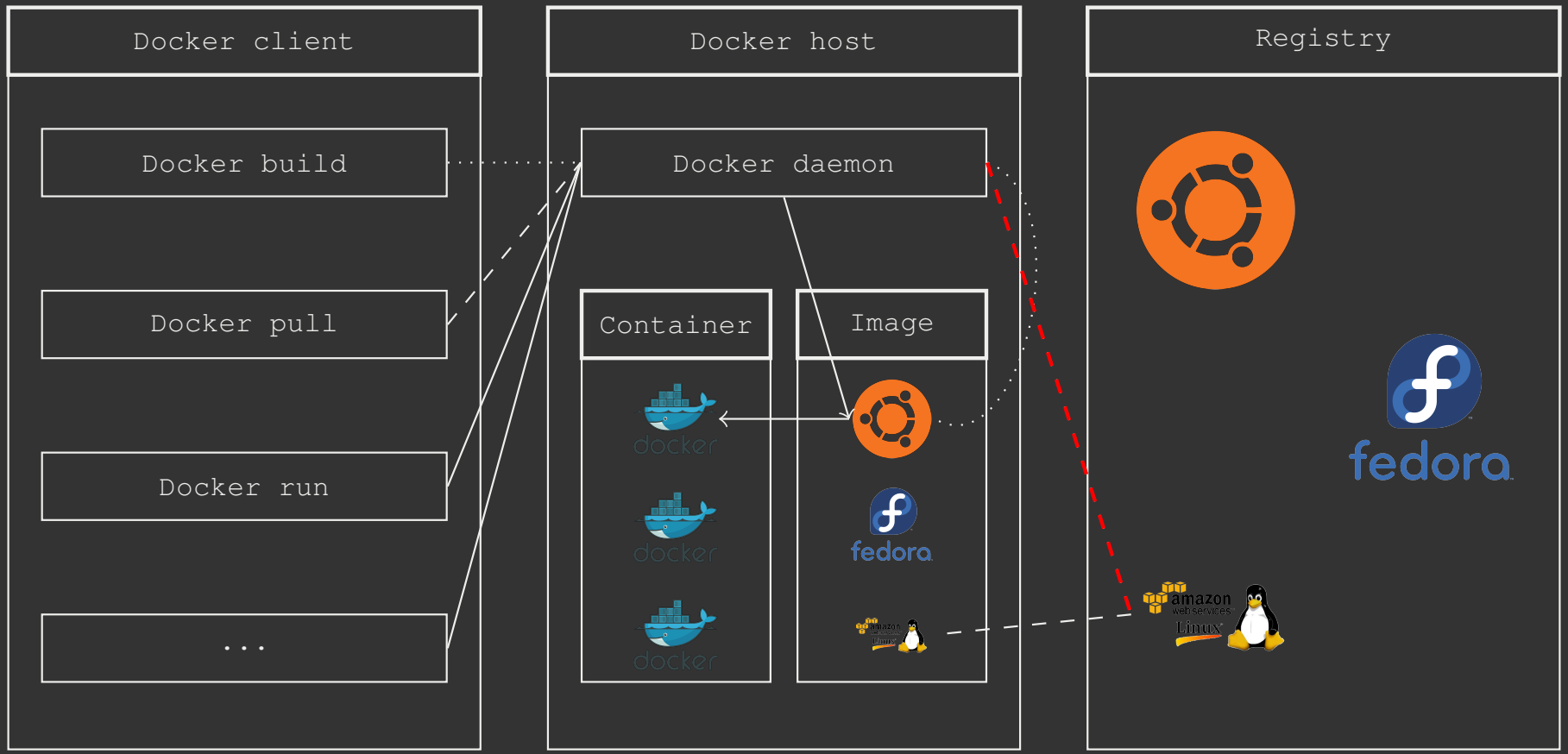
>>> Arquitetura do docker



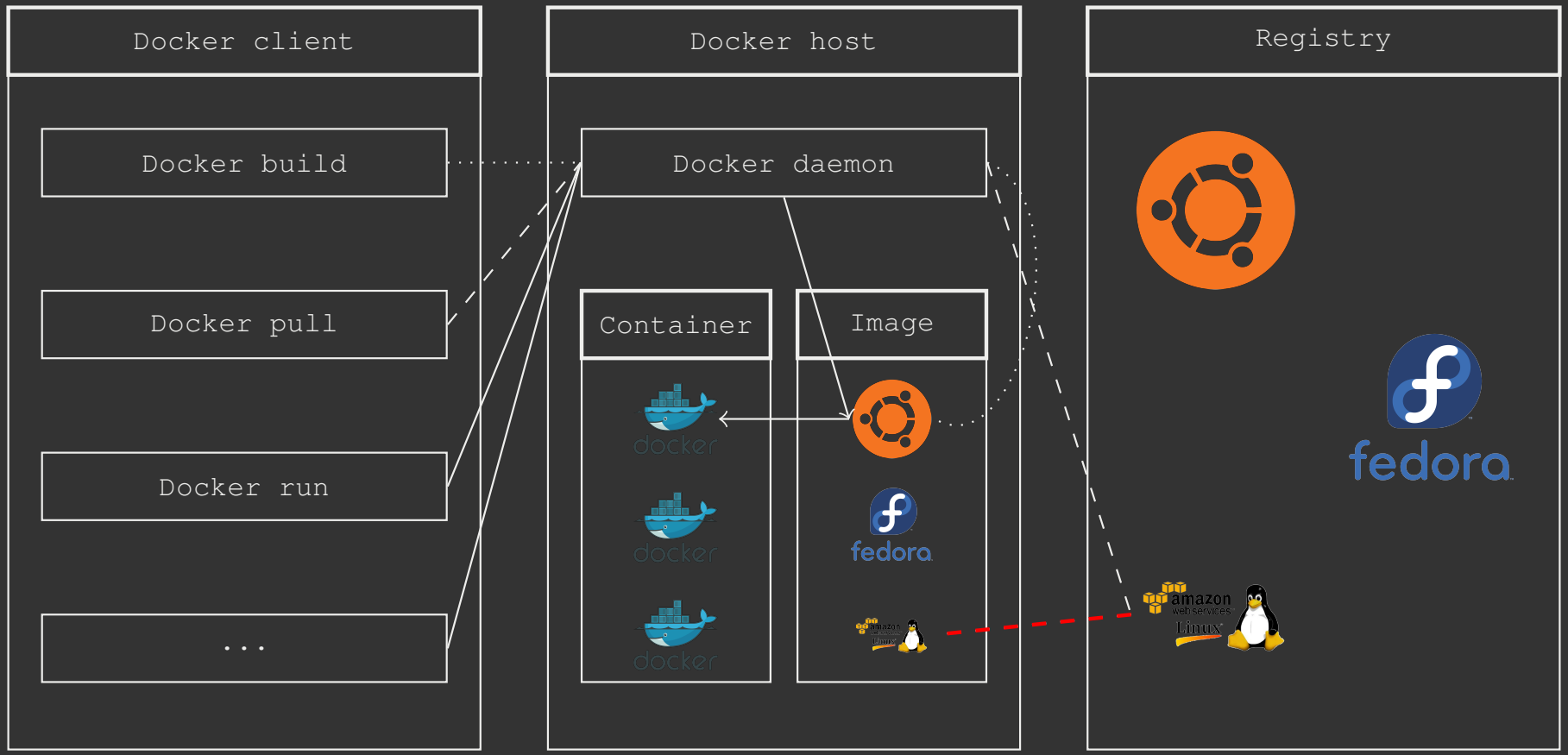
>>> Arquitetura do docker



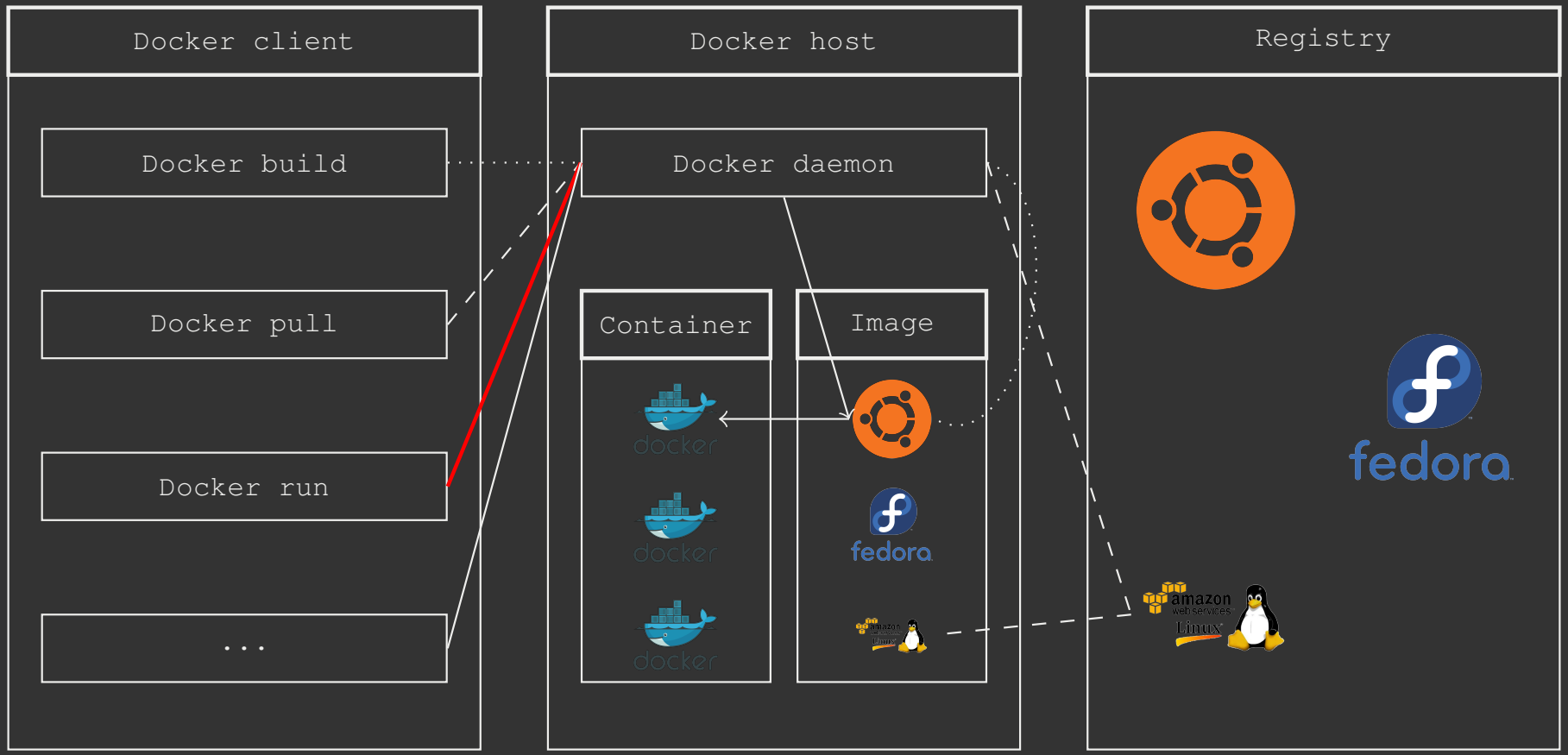
>>> Arquitetura do docker



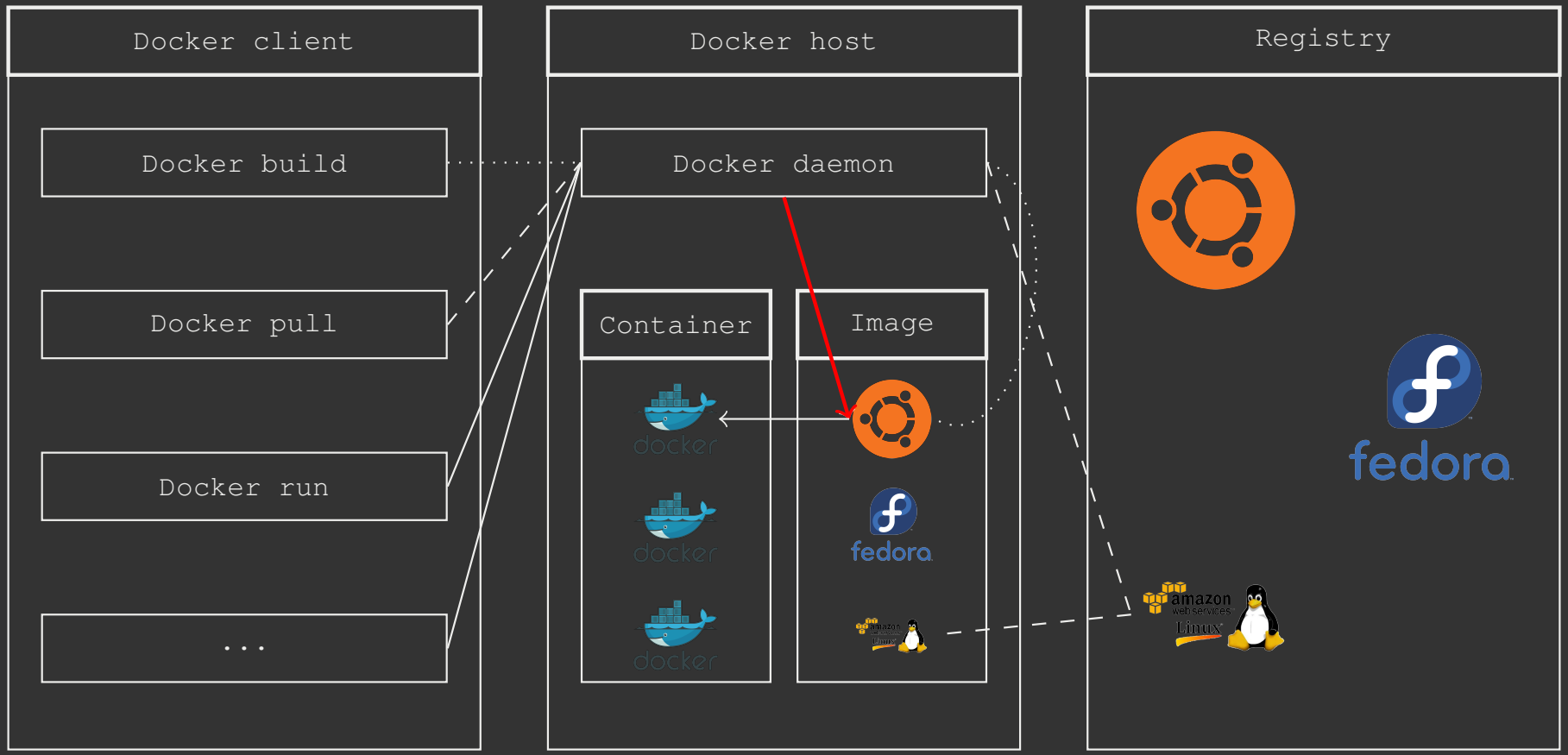
>>> Arquitetura do docker



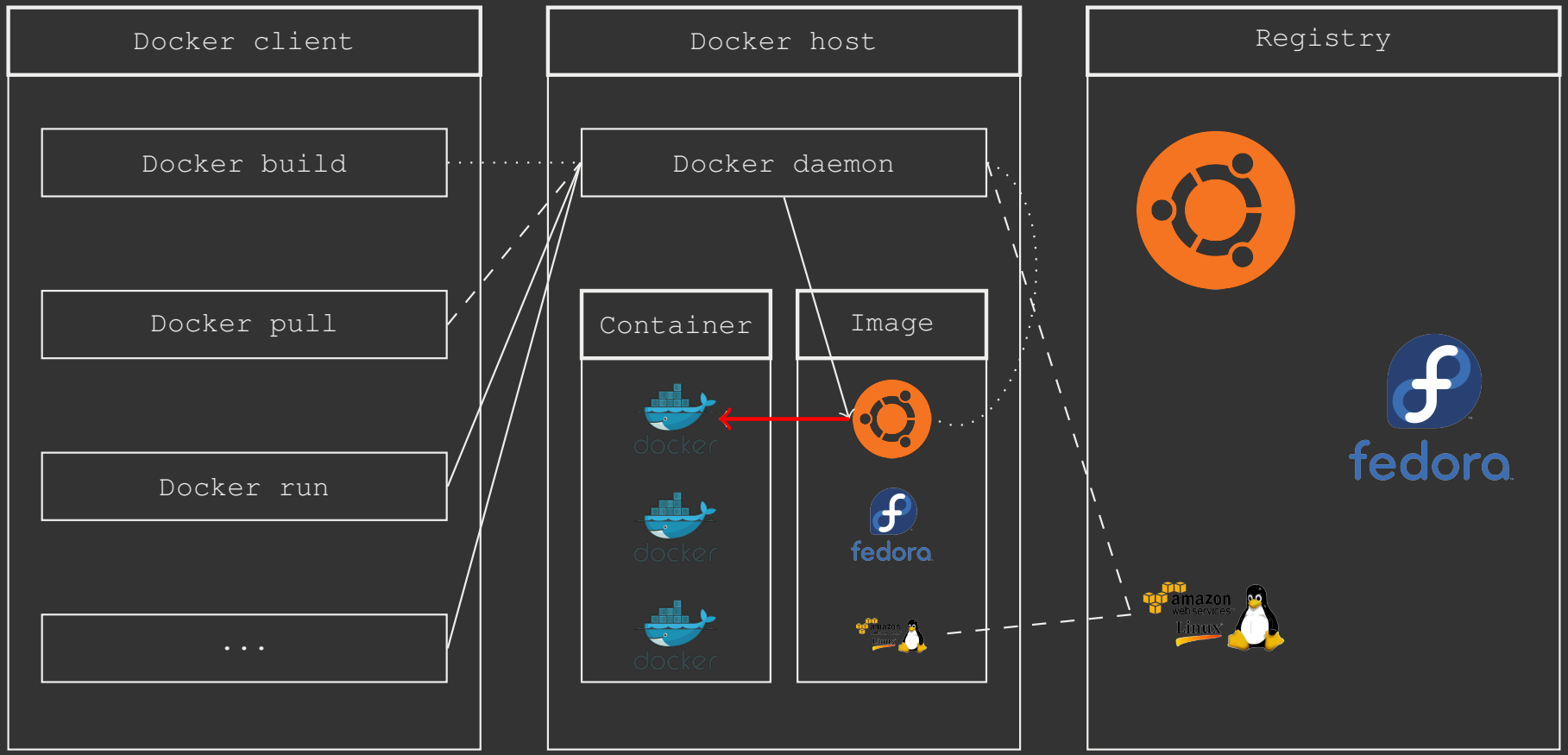
>>> Arquitetura do docker



>>> Arquitetura do docker



>>> Arquitetura do docker



>>> Docker Objects - Imagem

* É um **template**. Portanto, só é permitido ler;

>>> Docker Objects - Imagem

* É um **template**. Portanto, só é permitido ler;
* Contém instruções para criar um container de *Docker*;

>>> Docker Objects - Imagem

* É um **template**. Portanto, só é permitido ler;
* Contém instruções para criar um container de *Docker*;
* Uma **Imagem** é composta por camadas;

>>> Docker Objects - Imagem

* É um **template**. Portanto, só é permitido ler;
* Contém instruções para criar um container de *Docker*;
* Uma **Imagem** é composta por camadas;
* Camada inicial → Imagem pequena de Linux;

>>> Docker Objects - Imagem

* É um **template**. Portanto, só é permitido ler;
* Contém instruções para criar um container de *Docker*;
* Uma **Imagem** é composta por camadas;
* Camada inicial → Imagem pequena de Linux;
* Imagens existentes → Customizadas para gerar novas imagens;

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;

```
FROM python:3
```

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;

>>> Docker Objects - Imagem personalizada

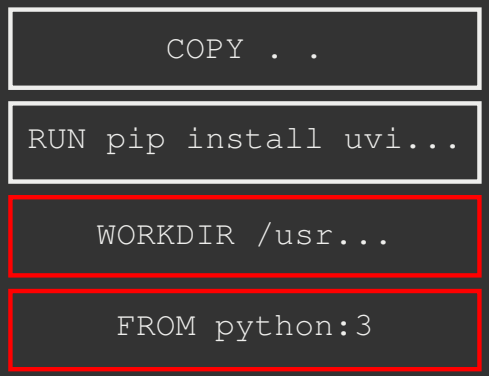
* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;

>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;



>>> Docker Objects - Imagem personalizada

* Para construir a uma imagem customizada, é preciso construir um *Dockerfile*.
* *Dockerfile*:
  * Arquivo com instruções para construir e executar uma imagem de um container;
  * Cada instrução no arquivo Dockerfile adiciona uma camada nova na imagem;

>>> Docker Objects - Imagem

* Novas customizações → Novas camadas na Imagem;

>>> Docker Objects - Imagem

* Novas customizações → Novas camadas na Imagem;
* Na construção de uma imagem, apenas as camadas que sofreram mudanças vão ser reconstruídas;

>>> Docker Objects - Imagem

* Novas customizações → Novas camadas na Imagem;
* Na construção de uma imagem, apenas as camadas que sofreram mudanças vão ser reconstruídas;
* Imagens leves, pequenas e rápidas (Comparadas com outras tecnologias);

>>> Docker Objects - Containers

* Instância de uma imagem;
  * Configurações definidas na inicialização e na imagem;

>>> Docker Objects - Containers

* Instância de uma imagem;
  * Configurações definidas na inicialização e na imagem;
  * Mudanças no container não são armazenadas;

>>> Docker Objects - Containers

* Instância de uma imagem;
  * Configurações definidas na inicialização e na imagem;
  * Mudanças no container não são armazenadas;
* Com a API do docker é possível:
  * Criar um container;

>>> Docker Objects - Containers

* Instância de uma imagem;
  * Configurações definidas na inicialização e na imagem;
  * Mudanças no container não são armazenadas;
* Com a API do docker é possível:
  * Criar um container;
  * Executar um container;

>>> Docker Objects - Containers

* Instância de uma imagem;
  * Configurações definidas na inicialização e na imagem;
  * Mudanças no container não são armazenadas;
* Com a API do docker é possível:
  * Criar um container;
  * Executar um container;
  * Parar um container;

>>> Docker Objects - Containers

* Instância de uma imagem;
  * Configurações definidas na inicialização e na imagem;
  * Mudanças no container não são armazenadas;
* Com a API do docker é possível:
  * Criar um container;
  * Executar um container;
  * Parar um container;
  * Mover um container;

>>> Docker Objects - Containers

* Instância de uma imagem;
  * Configurações definidas na inicialização e na imagem;
  * Mudanças no container não são armazenadas;
* Com a API do docker é possível:
  * Criar um container;
  * Executar um container;
  * Parar um container;
  * Mover um container;
  * Deletar um container;

>>> Docker Objects - Containers

* Instância de uma imagem;
  * Configurações definidas na inicialização e na imagem;
  * Mudanças no container não são armazenadas;
* Com a API do docker é possível:
  * Criar um container;
  * Executar um container;
  * Parar um container;
  * Mover um container;
  * Deletar um container;
* Gerenciar:
  * Redes;

>>> Docker Objects - Containers

* Instância de uma imagem;
  * Configurações definidas na inicialização e na imagem;
  * Mudanças no container não são armazenadas;
* Com a API do docker é possível:
  * Criar um container;
  * Executar um container;
  * Parar um container;
  * Mover um container;
  * Deletar um container;
* Gerenciar:
  * Redes;
  * Armazenamento;

>>> Docker Objects - Containers

* Instância de uma imagem;
  * Configurações definidas na inicialização e na imagem;
  * Mudanças no container não são armazenadas;
* Com a API do docker é possível:
  * Criar um container;
  * Executar um container;
  * Parar um container;
  * Mover um container;
  * Deletar um container;
* Gerenciar:
  * Redes;
  * Armazenamento;
  * Imagem;

>>> Docker Objects - Tecnologias

* Linguagem **Go;**

>>> Docker Objects - Tecnologias

* Linguagem **Go;**
* Muitas funcionalidades do kernel do linux:
    * namespaces: Garante que cada um dos containers tenha a capacidade de se isolar em níveis:

>>> Docker Objects - Tecnologias

* Linguagem **Go**;
* Muitas funcionalidades do kernel do linux:
  * namespaces: Garante que cada um dos containers tenha a capacidade de se isolar em níveis:
    * PID: Ter seus próprios PIDs, mas a máquina host pode ver os PIDs do container;
    * NET: Ter suas próprias interfaces de redes e portas. Responsável pela comunicação de containers;
    * MNT: Garante que um sistema de arquivos montado não consiga acessar outro sistema de arquivos montado por outro mnt;
    * IPC: Ter um SystemV IPC isolado, além de uma fila de mensagems POSIX própria;
    * UTS: Isolamento do hostname, nome do domínio, versão do SO, etc...;

>>> Docker Objects - Tecnologias

* Linguagem **Go**;
* Muitas funcionalidades do kernel do linux:
  * namespaces: Garante que cada um dos containers tenha a capacidade de se isolar em níveis:
    * PID: Ter seus próprios PIDs, mas a máquina host pode ver os PIDs do container;
    * NET: Ter suas próprias interfaces de redes e portas. Responsável pela comunicação de containers;
    * MNT: Garante que um sistema de arquivos montado não consiga acessar outro sistema de arquivos montado por outro mnt;
    * IPC: Ter um SystemV IPC isolado, além de uma fila de mensagems POSIX própria;
    * UTS: Isolamento do hostname, nome do domínio, versão do SO, etc...;
  * cgroups: Garantir o controle do consumo de processo;

>>> Docker Objects - Tecnologias

* Linguagem **Go**;
* Muitas funcionalidades do kernel do linux:
  * namespaces: Garante que cada um dos containers tenha a capacidade de se isolar em níveis:
    * PID: Ter seus próprios PIDs, mas a máquina host pode ver os PIDs do container;
    * NET: Ter suas próprias interfaces de redes e portas. Responsável pela comunicação de containers;
    * MNT: Garante que um sistema de arquivos montado não consiga acessar outro sistema de arquivos montado por outro mnt;
    * IPC: Ter um SystemV IPC isolado, além de uma fila de mensagems POSIX própria;
    * UTS: Isolamento do hostname, nome do domínio, versão do SO, etc...;
  * cgroups: Garantir o controle do consumo de processo;
  * Linux Security Modules: Permissões do que eu sou permitido fazer;

>>> Docker Objects - Container Runtimes

* As ferramentas **Container Runtimes** são ferramentas que facilitam a criação de containers de forma isolada e segura.
* Open container Initiative runtimes:
  * Native Runtimes:
    * runC: escrito em go e mantido pelo projeto moby do docker;
    * Railcar: escrito em rust, mas foi abandonado;
    * Crun: escrito em c, performatico e leve;
* Container Runtime Interface
  * containerd: um runtime de alto nível desenvolvido no projeto moby do docker, por padrão usa o runC por baixo dos panos;
  * cri-o: implementação liderada pela Redhat, feita especificamente para o kubernetes;

>>> Docker Objects - Resumo

* Daemon (dockerd): espera por chamads API do docker, e gerencia objetos como imagens, containers, redes e volumes;

>>> Docker Objects - Resumo

* Daemon (dockerd): espera por chamads API do docker, e gerencia objetos como imagens, containers, redes e volumes;
* Cliente (docker): interage com o dockerd;
    * Comandos como: docker run, o cliente envia comandos para o dockerd;

>>> Docker Objects - Resumo

* Daemon (dockerd): espera por chamads API do docker, e gerencia objetos como imagens, containers, redes e volumes;
* Cliente (docker): interage com o dockerd;
  * Comandos como: docker run, o cliente envia comandos para o dockerd;
  * O cliente docker pode se comunicar com mais de um daemon;

>>> Docker Objects - Resumo

* Daemon (dockerd): espera por chamads API do docker, e gerencia objetos como imagens, containers, redes e volumes;
* Cliente (docker): interage com o dockerd;
  * Comandos como: docker run, o cliente envia comandos para o dockerd;
  * O cliente docker pode se comunicar com mais de um daemon;
* Docker desktop: é uma aplicação fácil de ser instalada nos ambientes Mac, Windows ou Linux.

>>> Docker Objects - Resumo

* Daemon (dockerd): espera por chamads API do docker, e gerencia objetos como imagens, containers, redes e volumes;
* Cliente (docker): interage com o dockerd;
  * Comandos como: docker run, o cliente envia comandos para o dockerd;
  * O cliente docker pode se comunicar com mais de um daemon;
* Docker desktop: é uma aplicação fácil de ser instalada nos ambientes Mac, Windows ou Linux.
  * O docker desktop contém o daemon (dockerd), o docker client (docker), docker compose, docker content trust, kubernetes e credential helper;

>>> Docker Objects - Resumo

* Docker registries: Guarda as imagens de docker;
  * Docker Hub: é um registro público, por padrão o docker procura por imagens no Docker Hub;

>>> Docker Objects - Resumo

* Docker registries: Guarda as imagens de docker;
  * Docker Hub: é um registro público, por padrão o docker procura por imagens no Docker Hub;
  * É possível ter até seu registro próprio;

>>> Docker Objects - Resumo

* Docker registries: Guarda as imagens de docker;
  * Docker Hub: é um registro público, por padrão o docker procura por imagens no Docker Hub;
  * É possível ter até seu registro próprio;
  * Os comandos docker pull, docker run e docker push procuram ou enviam as imagens para o registro configurado na máquina que estão rodando;

>>> Docker Objects - Resumo

* Docker registries: Guarda as imagens de docker;
  * Docker Hub: é um registro público, por padrão o docker procura por imagens no Docker Hub;
  * É possível ter até seu registro próprio;
  * Os comandos docker pull, docker run e docker push procuram ou enviam as imagens para o registro configurado na máquina que estão rodando;
* Docker objects: Imagens, conteiners, redes, volumes, plugins, etc...