Docker

DevOps principais práticas: DevOps visa integrar essas equipes, promovendo uma cultura de colaboração e utilizando ferramentas para acelerar as entregas sem comprometer a qualidade. O foco é que os engenheiros acompanhem todo o ciclo de vida da aplicação, desde o desenvolvimento até as operações. As principais práticas do movimento incluem:

1. Infraestrutura como código (IaC)
2. Pipelines de integração e entrega contínua
3. Virtualização e conteinerização
4. Monitoramento e feedback constante da aplicação em produção

O objetivo é unir pessoas, processos e tecnologias para entregar valor contínuo ao cliente, com foco em agilidade e recuperação rápida em caso de falhas.

O que é um Docker?

O Docker é uma plataforma de software que permite criar, implantar e executar aplicativos em contêineres. Contêineres são ambientes isolados e leves que empacotam uma aplicação e suas dependências como bibliotecas, configurações, firmwares e outros de forma que possam ser executados de maneira consistente em qualquer ambiente, seja no computador do desenvolvedor, em servidores ou em nuvem.

A principal vantagem do Docker é a portabilidade. Como o contêiner contém tudo o que a aplicação precisa para rodar, ela pode ser movida entre diferentes máquinas sem problemas relacionados a configurações ou dependências.

Além disso, o Docker facilita a automação do processo de desenvolvimento, teste e implantação de aplicativos, assim como é usado em conjunto com outras ferramentas de DevOps para criar pipelines de integração e entrega contínua.

O que é um contêiner?

Contêiner é uma unidade de software que empacota uma aplicação e todas as suas dependências, como bibliotecas, configurações e arquivos necessários, para que ela possa ser executada de forma consistente em qualquer ambiente, seja no computador do desenvolvedor, em servidores ou na nuvem. Dessa forma, facilitando o trabalho, economizando tempo e ainda garantindo que todas as máquinas e aplicações dos trabalhadores funcionem da maneira devida.

Docker imagem vs Docker contêiner

A principal diferença entre Docker Imagem e Docker Container está no fato de que a imagem é um pacote imutável e estático, enquanto o contêiner é uma instância em execução dessa imagem, ou seja, a imagem serve como um molde para criar e executar contêineres.

Docker vs Máquinas Virtuais

A principal diferença entre dockers e mv é que um Docker utiliza o mesmo sistema operacional (kernel) do host, então é uma maneira mais leve e eficiente de realizar certos trabalhos, visto que não é necessário outro sistema operacional, que ocuparia espaço de armazenamento e usaria dados. Nesse sentido, Docker é ideal quando você precisa de eficiência, agilidade e portabilidade. Ele é amplamente usado em ambientes de desenvolvimento, microserviços e DevOps.

Máquinas Virtuais são mais adequadas quando você precisa de isolamento completo ou deseja executar sistemas operacionais diferentes no mesmo host. Elas são mais utilizadas em infraestruturas tradicionais ou quando o controle total sobre o sistema operacional é necessário.

Docker no desenvolvimento de software:

O Docker transforma a forma como as equipes de desenvolvimento de software trabalham, trazendo agilidade, consistência e portabilidade aos processos de desenvolvimento, teste e implantação. Com o Docker, as equipes podem desenvolver, testar, escalar e implantar suas aplicações de forma mais eficiente, resultando em uma entrega contínua e mais rápida de valor para os clientes. Além disso, o Docker resolve o problema do "funciona na minha máquina", uma vez que os desenvolvedores podem empacotar uma aplicação com todas as suas dependências em um contêiner, que pode ser executado de forma consistente em diferentes ambientes, como máquinas de desenvolvimento, testes e produção.

Docker network

O Docker Network é um componente fundamental do Docker que permite que os contêineres se comuniquem uns com os outros e com o mundo externo. Ele fornece a infraestrutura de rede necessária para que os contêineres se conectem entre si, com o host e com outros recursos de rede. Ao usar redes Docker, é possível garantir que contêineres que precisam se comunicar compartilhem o mesmo ambiente de rede, enquanto os que não precisam de comunicação podem ser isolados.

Bridge Network (Rede Padrão)

> **Uso**: Ideal para situações em que você quer que os contêineres se comuniquem entre si, mas não necessitam de acesso direto à rede externa.

Host Network

> **Uso**: Útil quando se deseja alto desempenho de rede e não se precisa de isolamento, já que o contêiner compartilha a pilha de rede do host.

None Network

> **Uso**: É usada em cenários onde o contêiner não precisa de rede, como quando o contêiner apenas executa tarefas isoladas.

Overlay Network

> **Uso**: Usado em ambientes com Docker Swarm ou Kubernetes, quando é necessário orquestrar múltiplos contêineres distribuídos em diferentes servidores físicos.

Macvlan Network

> **Uso**: Ideal quando você precisa que os contêineres se comportem como dispositivos de rede físicos ou quando os contêineres precisam ser acessados diretamente pela rede externa.

Docker compose:

O Docker Compose é uma ferramenta para gerenciar contêineres Docker de forma simples e eficiente. Ele permite que você defina a arquitetura do seu aplicativo com múltiplos serviços em um único arquivo de configuração, facilitando o desenvolvimento, testes e implantação. Com Compose, é possível gerenciar de forma ágil e automatizada aplicações complexas que dependem de múltiplos contêineres e serviços.

Principais Conceitos do Docker Compose

> **Serviços**: Cada contêiner no seu aplicativo é um "serviço". No arquivo docker-compose.yml, você define quais contêineres serão executados, suas imagens ou instruções para construir as imagens, as portas que devem ser expostas, e os volumes e redes que devem ser conectados.
>
> **Redes**: O Compose permite definir redes para que os contêineres possam se comunicar entre si de maneira isolada e controlada.
>
> **Volumes**: Com Compose, você pode facilmente persistir dados fora dos contêineres usando volumes, para garantir que os dados não sejam perdidos quando um contêiner for removido ou reiniciado.

Principais comandos e suas funções:

docker compose build → Constrói as imagens a partir do Dockerfile.

docker compose start → Inicia os containers que já foram criados anteriormente.

docker compose stop → Para os containers, mas sem removê-los.

docker compose up -d → Constrói (se necessário) e inicia os containers em segundo plano.

docker compose os → Lista os containers que estão rodando com base no docker-compose.yml.
docker compose rm → Remove containers parados da memória, sem apagar as imagens.

docker compose down → Para e remove os containers, redes e volumes criados pelo 'up'.

docker compose logs → Exibe os logs de execução dos containers.

docker compose exec [container] bash → Executa um comando dentro de um container, como abrir o bash.

O que é Kubernetes:

É uma plataforma open-source para orquestrar e gerenciar contêineres em larga escala, automatizando o deployment, o gerenciamento e a escalabilidade de aplicativos em contêineres. Desenvolvido originalmente pelo Google, o Kubernetes foi projetado para facilitar

o gerenciamento de ambientes complexos e distribuídos, como aqueles que utilizam contêineres Docker.

Dockerfile: Um Dockerfile é um arquivo de texto simples usado para criar imagens Docker. Ele contém uma série de instruções que automatizam o processo de criação de um ambiente configurado, como sistema operacional, dependências, bibliotecas e configurações necessárias para executar uma aplicação em containers. Abaixo estão as principais funções para se escrever em um Dockerfile:

FROM

A instrução FROM é obrigatória em qualquer Dockerfile, pois define a base para a criação da imagem. Ela especifica a imagem inicial que será utilizada, como openjdk para criar uma imagem com Java, ou mesmo scratch para começar do zero, sem dependências pré-definidas. A escolha da base é crucial para determinar o ambiente inicial do container.

RUN

A instrução RUN permite executar comandos durante a construção da imagem, criando camadas que podem ser reutilizadas no cache para otimizar o processo de build. Por exemplo, comandos como apt-get update ou apt-get install instalam pacotes e configuram o ambiente, e o resultado dessas operações é incluído na imagem final. Além disso, os comandos podem ser escritos em formato shell (RUN comando) ou em forma de lista (RUN ["comando", "argumentos"]), sendo igualmente eficientes.

CMD e ENTRYPOINT

As instruções CMD e ENTRYPOINT são usadas em Dockerfiles para definir comandos que serão executados quando o container for iniciado, mas possuem comportamentos diferentes. A CMD especifica comandos ou argumentos padrão para o container, que só são executados se não forem substituídos no momento da execução. Caso vários comandos CMD sejam definidos no Dockerfile, apenas o último será considerado. Por exemplo, no Dockerfile abaixo, o comando touch outro-arquivo será o único executado, já que sobrescreve os anteriores.

Já a ENTRYPOINT é semelhante, mas seus parâmetros não podem ser sobrescritos da mesma forma que os do CMD. Isso garante que o comando especificado na ENTRYPOINT será sempre executado, mesmo que argumentos adicionais sejam passados ao iniciar o container. A diferença fundamental está na flexibilidade: CMD permite substituição, enquanto ENTRYPOINT assegura a execução do comando principal definido.

ADD e COPY

As instruções ADD e COPY em Dockerfiles servem para copiar arquivos ou diretórios da máquina host para dentro da imagem Docker, mas possuem diferenças importantes. A ADD permite copiar arquivos locais, baixar arquivos diretamente de URLs e até descomprimir arquivos tar automaticamente ao adicioná-los na imagem. Por exemplo, no comando:

ADD arquivo-host arquivo-host-transferido

O arquivo arquivo-host é transferido para dentro da imagem com o nome especificado.

Já a COPY tem uma funcionalidade mais simples: ela apenas copia arquivos ou diretórios locais para a imagem, sem suportar downloads ou descompressão automática. Sua sintaxe é similar ao ADD, como neste exemplo:

COPY arquivo-host arquivo-host-transferido

Embora o ADD tenha mais recursos, o COPY é recomendado para operações de cópia simples, garantindo maior clareza e previsibilidade no Dockerfile.

EXPOSE

A instrução EXPOSE em um Dockerfile serve para documentar a porta que o container usará, mas não a publica efetivamente. Seu objetivo é informar os desenvolvedores sobre a comunicação entre o Dockerfile e quem rodará o container. Por exemplo, ao definir EXPOSE 8080, não estamos permitindo que a porta 8080 seja acessada diretamente; estamos apenas indicando que essa porta estará disponível para uso, dependendo da configuração do ambiente onde o container for executado.

VOLUME

A instrução VOLUME é utilizada para criar um diretório dentro do container que será compartilhado com a máquina host. Essa pasta pode ser usada para armazenar arquivos que precisam ser persistidos ou acessados entre a máquina e o container. No Dockerfile, ao usar VOLUME /foo, criamos uma pasta /foo dentro do container, e todo arquivo criado ali será acessível a partir do diretório /var/lib/docker/volumes no host. Essa abordagem é essencial para persistir dados, como bancos de dados ou arquivos de log.

WORKDIR

A instrução WORKDIR no Dockerfile define o diretório de trabalho dentro do container onde as instruções como RUN, CMD, ENTRYPOINT, ADD e COPY serão executadas. Com ela, você pode organizar as tarefas de maneira eficiente, garantindo que o container inicie sempre em um diretório específico. Por exemplo, ao definir WORKDIR /pasta-qualquer, todas as operações seguintes ocorrerão nesse diretório, incluindo a cópia de arquivos da máquina host para dentro do container, facilitando a organização e execução do ambiente de trabalho dentro do container.

Além disso, existem comandos básicos para o Docker rodar da maneira devida, abaixo está um exemplos dos principais comandos mais usados:

Verificar a versão do Docker: docker --version
Verifica a versão instalada do Docker.

Verificar se o Docker está rodando: docker info
Exibe informações detalhadas sobre o estado atual do Docker.

Baixar uma imagem: docker pull <nome-da-imagem>
Baixa uma imagem específica do Docker Hub.

Criar e iniciar um container: docker run -d <nome-da-imagem>
Cria e executa um container em segundo plano a partir de uma imagem.

Listar containers em execução: docker ps
Exibe todos os containers em execução.

Listar todos os containers (inclusive os parados): docker ps -a
Exibe todos os containers, tanto os em execução quanto os parados.

Parar um container: docker stop <id-do-container>
Para um container em execução.

Remover um container: docker rm <id-do-container>
Remove um container que não está mais em uso.

Remover uma imagem: docker rmi <nome-da-imagem>
Remove uma imagem do seu sistema local.

C**onstruir uma imagem a partir de um Dockerfile**: docker build -t <nome-da-imagem> <caminho-do-Dockerfile>
Constrói uma imagem Docker a partir de um Dockerfile.

Verificar logs de um container: docker logs <id-do-container>
Exibe os logs gerados por um container.

Acessar o terminal de um container: docker exec -it <id-do-container> /bin/bash
Acessa o terminal de um container em execução.

Listar imagens locais: docker images
Exibe todas as imagens Docker armazenadas localmente.

Remover volumes não utilizados: docker volume prune
Remove volumes não utilizados por containers, liberando espaço.