Containers

O container nada mais é do que um ambiente isolado, disposto em um servidor, que divide um único host de controle.

Caso um dos recipientes seja danificado, os demais não são afetados. Afinal, são isolados, protegidos e estão carregando seus próprios sistemas.

Cada container possui uma função e sua responsabilidade. Caso um deles sofra um dano, o funcionamento do sistema não para e a função afetada é redirecionada para um novo container.

Containers

Os containers funcionam um pouco como as VMs, mas de uma maneira muito mais específica e granular.

Em uma máquina virtual, é possível utilizar diversos recursos e ferramentas, como Apache e PHP, porém tudo roda em um mesmo sistema operacional. Em caso de pane, todas as funcionalidades são afetadas.

No caso dos containers, a ideia é que cada um faça apenas um serviço e assuma uma só responsabilidade. Ou seja, seria um rodando com Apache e outro com PHP.

Containers

Desta forma, é possível isolar os processos de cada ferramenta, garantindo que nenhuma atrapalhe o funcionamento da outra.

Para serviços web, por exemplo, os containers deixam a infraestrutura muito mais intercambiável, eficiente e flexível.

Eles isolam um único aplicativo e suas dependências – todas as bibliotecas externas de software que o aplicativo precisa executar – tanto do sistema operacional subjacente quanto de outros containers.

Containers

Todos os aplicativos em containers compartilham um único sistema operacional comum.

À primeira vista, eles podem até tornar a situação um pouco mais complexa, porém, principalmente nos servidores de produção, oferecem um ganho enorme em termos de escala e performance e, portanto, são uma ferramenta valiosa.

Containers

As ferramentas de container, incluindo o Docker, fornecem um modelo de implantação com base em imagens.

Isso facilita o compartilhamento de uma aplicação ou conjunto de serviços, incluindo todas as dependências deles em vários ambientes.

Containers

O Docker também automatiza a implantação da aplicação (ou de conjuntos de processos que constituem uma aplicação) dentro desse ambiente de container.

Essas ferramentas baseadas nos containers Linux oferecem aos usuários acesso sem precedentes a aplicações, além da habilidade de implementar com rapidez e de ter total controle sobre as versões e distribuição.

Containers

O Docker permite um uso mais eficiente dos recursos do sistema;

As instâncias de aplicativos em container usam muito menos memória do que as máquinas virtuais, elas são inicializadas e interrompidas mais rapidamente e podem ser armazenadas muito mais densamente em um hardware host. Tudo isso equivale a menos gastos com TI;

Containers

A redução de custos irá variar dependendo de quais aplicativos estão em jogo e de quão intensivos os recursos podem ser, mas os containers funcionam invariavelmente como mais eficientes que as VMs.

Também é possível economizar nos custos de licenças de software, porque é necessário muito menos instâncias do sistema operacional para executar as mesmas cargas de trabalho;

Containers

O Docker permite ciclos de entrega de software mais rápidos;

O software corporativo deve responder rapidamente a mudanças de condições. Isso significa que o escalonamento fácil atende à demanda e facilita a atualização para adicionar novos recursos conforme a necessidade do negócio;

Containers

Os containers Docker facilitam a colocação rápida de novas versões de software, com novos recursos de negócios, e a rápida reversão para uma versão anterior, se necessário.

O Docker permite a portabilidade de aplicativos;

Os containers são leves, portáteis e facilitam a construção de software em linhas de pensamento avançadas, de modo que o desenvolvedor não está tentando resolver os problemas de amanhã com os métodos de desenvolvimento de ontem.

Containers

Contêineres são ambientes isolados que permitem executar aplicativos sem interferir no sistema operacional host.

Eles incluem tudo o que é necessário para o aplicativo funcionar, como bibliotecas, dependências e configurações.

Containers

Docker é uma plataforma de virtualização que permite a criação de contêineres de software com objetivo de empacotar e distribuir aplicativos em um ambiente isolado.

Docker Compose é uma ferramenta que permite definir e executar aplicativos multi-contêineres em um ambiente isolado.

Ele usa um arquivo YAML para definir os serviços, redes e volumes do aplicativo.

Containers

É possível usar o Docker Compose para iniciar e parar um aplicativo multi-contêiner.

O arquivo YAML é usado para definir os serviços, redes e volumes do aplicativo.

Containers

Exemplo de arquivo docker-compose.yml:

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

Containers

O Docker usa imagens para empacotar e distribuir aplicativos.

Uma imagem é um pacote que contém todas as dependências necessárias para executar o aplicativo.

Quando uma imagem é executada, ela se torna um contêiner.

Containers

É possível criar uma imagem Docker a partir de um arquivo Dockerfile.

O Dockerfile é um arquivo de texto que contém as instruções para criar a imagem.

Containers

Exemplo de Dockerfile:

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y nginx
COPY index.html /var/www/html/
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Containers

Para gerar uma imagem a partir de um arquivo Dockerfile, é preciso utilizar o comando docker build.

A sintaxe do comando docker build é a seguinte:

docker build -t NOME-DA-IMAGEM:VERSÃO PATH-DO-ARQUIVO-DOCKERFILE

Containers

É possível executar um contêiner Docker a partir de uma imagem.

O comando docker run é usado para executar um contêiner.

É possível definir variáveis de ambiente e mapear portas do contêiner para a máquina host.

Containers

Exemplo de comando docker run:

docker run -d --name meu_contêiner -e VARIAVEL_DE_AMBIENTE=valor -p 8080:80 minha_imagem:latest

Containers

O -d utilizado no comando serve para fazer com que o container docker sejainicializado rodando em background.

O --name é utilizado para nomear um container docker.

O -e é utilizado para criar variáveis de ambiente.

Variáveis de ambiente são valores que podem ser usados pelo aplicativo em tempo de execução.

Elas são definidas como pares chave-valor.

Containers

Exemplo de definição de variáveis de ambiente no Docker Compose:

```
version: '3'
services:
  web:
    image: nginx
    environment:
      - VARIAVEL_DE_AMBIENTE=valor
```

Containers

O -p serve para fazer o mapeamento entre a porta que vai ser utilizada no host que o container vai subir e a porta interna do container.

O valor passado ao parâmetro -p é número da porta do host seguido de dois pontos (:) mais a porta que será utilizada internamente pelo container.

Containers

As portas permitem que os contêineres se comuniquem com o mundo externo.

É possível mapear portas do contêiner para a máquina host.

Containers

Exemplo de mapeamento de portas no Docker Compose:

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "8080:80"
```

Containers

Um outro parâmetro muito importante que pode ser utilizado é o –v, utilizado para fazer mapeamento de volumes.

Ele serve para fazer mapeamento de arquivos e diretórios entre o host e o container.

Funciona de forma parecida com o mapeamento das portas.

O valor passado ao parâmetro -v é o caminho do diretório ou arquivo do host que será mapeado seguido de dois pontos (:) mais o caminho do diretório ou arquivo que será mapeado internamente no container.

Containers

Exemplo de mapeamento de volume no Docker Compose:

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "8080:80“
    volumes:
      - ./website:/var/www/html/
      - ./apache2.conf:/etc/apache2/apache2.conf
```

Containers

É possível executar comandos no Dockerfile para configurar o ambiente do contêiner.

O comando RUN é usado para executar comandos durante a criação da imagem.

Containers

Exemplo de uso de comandos no Dockerfile:

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y python3
CMD ["python3", "/app.py"]
```

Containers

É possível executar um contêiner Docker a partir de uma imagem.

O comando docker run é usado para executar um contêiner.

Ele cria uma instância da imagem utilizada no comando e executa essa instância criada.

Containers

Exemplo de comando docker run:

```
docker run -d --name meu_contêiner -p 80:80 minha_imagem:latest
```

Containers

É possível criar um contêiner Docker a partir de uma imagem, sem executá-lo.

O comando docker create é utilizado para isso.

Ele cria uma instância da imagem utilizada no comando, mas não a executa. A instância fica com status de “Created”.

Containers

Exemplo de comando docker create:

```
docker create --name meu_contêiner -p 80:80 minha_imagem:latest
```

Containers

Para executar a imagem criada, basta utilizar o comando docker start. Para tanto, basta digitar docker start seguido do nome da imagem que está com status de criada.

Existem outros comandos importantes do docker que podemos utilizar. A seguir, falaremos de alguns deles.

Containers

docker stop NOME-DO-CONTAINER: como o próprio nome já diz, é o comando responsável por parar um container que está em execução.

docker remove NOME-DO-CONTAINER: este comando é utilizado para remover um container que não está em execução. Ou seja, que está com status diferente de UP.

Também pode ser utilizado como docker rm NOME-DO-CONTAINER

Containers

docker ps: comando utilizado para listar os container que estão em execução, com status UP.

Para listar todos os containers, independente de estarem ou não em execução, basta utilizar o comando docker ps -a

docker images: comando utilizado para listar as imagens disponíveis no host.

docker rmi ID-DA-IMAGEM: comando utilizado para remover uma imagem do host

Containers

docker cp PATH-DO-ARQUIVO-NO-HOST NOME-DO-CONTAINER:PATH-DE-DESTINO-DENTRO-DO-CONTAINER: comando utilizado para copiar um arquivo do host para dentro do container.

Também pode ser utilizado para copiar um arquivo de dentro do container para o host. Basta inverter a ordem dos arquivos.

docker inspect NOME-DO-CONTAINER: comando utilizado para exibir informações sobre um determinado container, incluindo seu endereço IP.

Containers

docker logs NOME-DO-CONATINER: comando utilizado para exibir o log de um determinado container.

Pode ser utilizado com o parâmetro -f para que o terminal continue escutando os logs ao invés de mostrar apenas os logs que já aconteceram no container.

docker volume ls: comando utilizado para listar os volumes utilizados no host.

docker volume rm: comando utilizado para remover um volume específico.

Containers

docker volume prune: comando utilizado para remover todos os volumes que não estão sendo mais utilizados no host. Este comando vai solicitar a confirmação da ação.

docker network ls: comando utilizado para listar as redes utilizadas no host.

docker network rm: comando utilizado para remover uma rede específica.

docker network prune: comando utilizado para remover todas as redes que não estão sendo utilizadas no host.

Containers

docker login HUB-DE-CONTAINERS: comando utilizado para fazer login em um hub específico.

docker pull NOME-DA-IMAGEM:VERSÃO: comando utilizado para fazer o download de uma imagem do hub para o host onde o comando está sendo utilizado.

docker push NOME-DA-IMAGEM:VERSÃO: comando utilizado para fazer um upload de uma imagem que está no host onde o comando está sendo executado para um hub específico. É preciso estar logado no hub.

Containers

docker-compose up - Iniciar os contêineres do aplicativo.

docker-compose down - Parar os contêineres do aplicativo.

docker-compose ps - Exibir os contêineres em execução.

docker-compose logs - Exibir os logs dos contêineres.

Containers

Docker e Docker Compose são ferramentas poderosas para empacotar e distribuir aplicativos em um ambiente isolado.

Aprender a usá-los pode ajudar a tornar o processo de desenvolvimento e implantação de aplicativos mais eficiente e confiável.