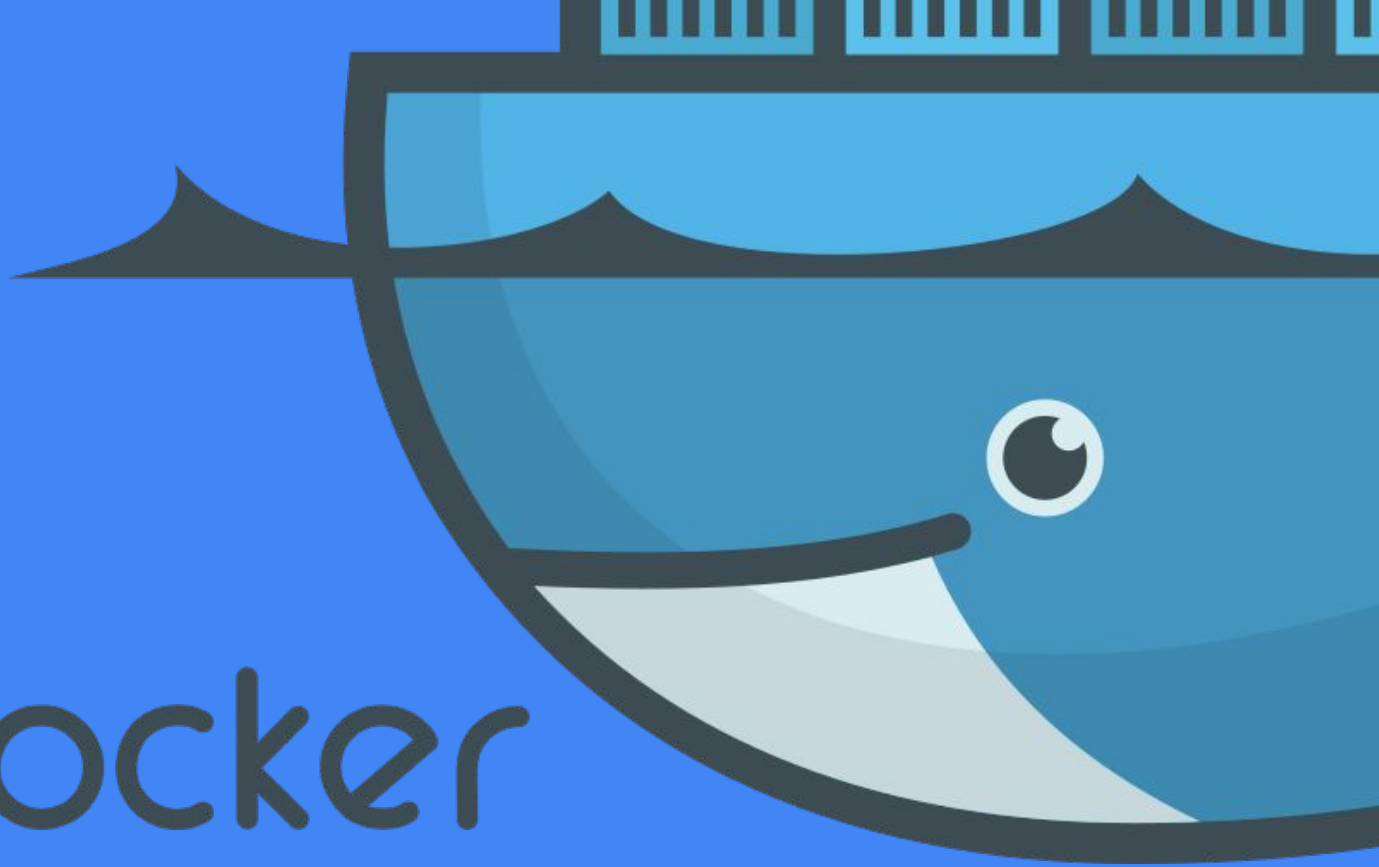
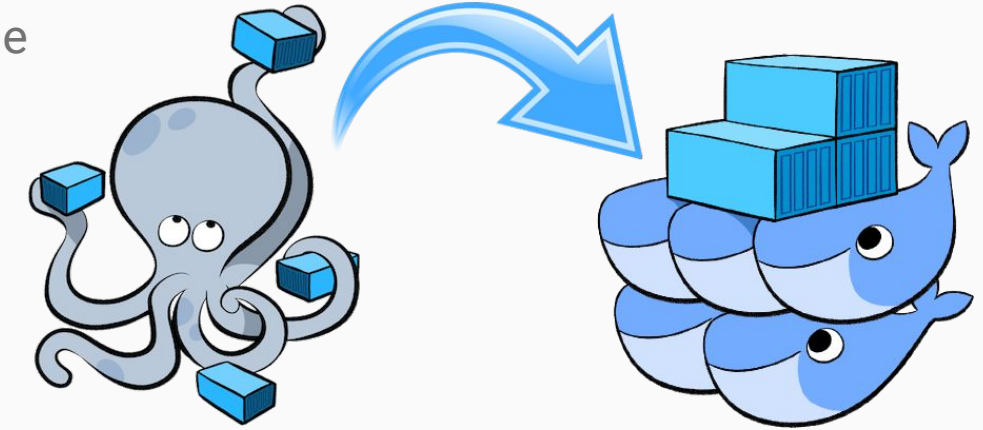


Módulo 2 - docker-compose

Por Fabio Szostak (2020)

docker-compose

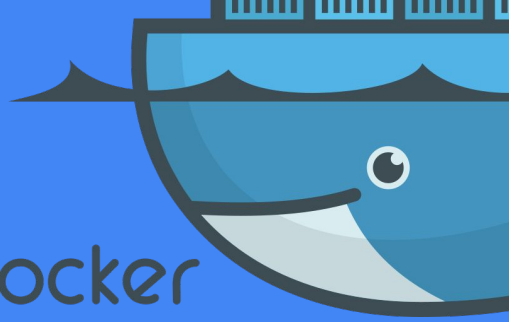
Compose é uma ferramenta para definir e executar aplicativos Docker de vários contêineres. Com o Compose, você usa um arquivo YAML para configurar os serviços do seu aplicativo. Então, com um único comando, você cria e inicia todos os serviços de sua configuração.



docker-compose

Utilizamos o docker-compose apenas para configurar o ambiente de desenvolvimento local e deixá-lo o mais próximo possível do ambiente de produção.

docker-compose



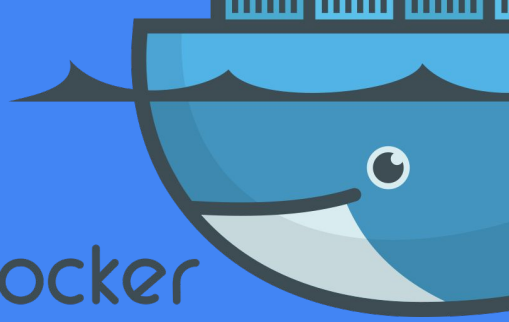
docker-compose.yml

Ao lado apenas um exemplo para executar um container de NGINX.

Podemos identificar alguns parâmetros que teriamos que passar no momento do run do container.

```
version: '3.8'

services:

  nginx:
    hostname: nginx
    image: nginx:1.19
    ports:
      - 90:80
    volumes:
      - .:/usr/share/nginx/html
```

docker-compose



.env

Variáveis podem ser especificadas dentro de um arquivo .env, o docker-compose irá ler o arquivo e essas variáveis serão utilizadas dentro do docker-compose.yml

docker-compose.yml

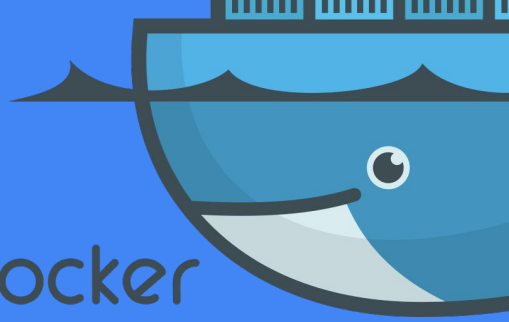
```
version: '3.8'

services:

  mysql:
    container_name: ${APP_NAME}-mysql
    hostname: mysql
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: ${DB_NAME}
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD: ${DB_PASS}
```

.env

```
APP_NAME=UnimedPortal
ENVIRONMENT=development

# Database settings
DB_HOST=mysql
DB_NAME=unimed_portal_dev
DB_USER=drupal
DB_PASS=drupal
DB_PORT=3306
```

Este recurso é útil para definir variáveis que são modificadas entre ambientes, geralmente os hosts, users, passwords, etc.

docker-compose : networks



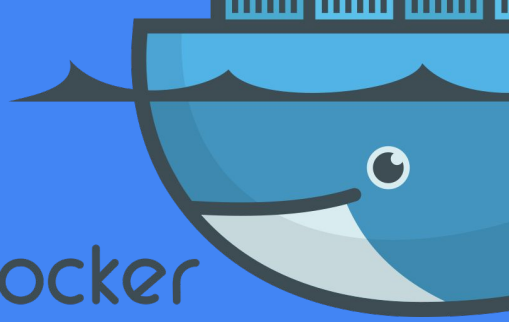
name: especifica o nome da rede a ser utilizada pelos containers.

O APP_NAME é utilizado para não gerar conflito com algum outro projeto que esteja rodando.

docker-compose.yml

```yaml
version: '3.8'

services:

  mysql:
    container_name: ${APP_NAME}-mysql
    hostname: mysql
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: ${DB_NAME}
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD: ${DB_PASS}


networks:
  default:
    name: ${APP_NAME}-network
```

docker-compose : services



container-name: nome do container, o APP_NAME é usado para não conflitar com containers de outros projetos.

restart: Significa que caso o container seja encerrado por qualquer motivo, ele será reiniciado.

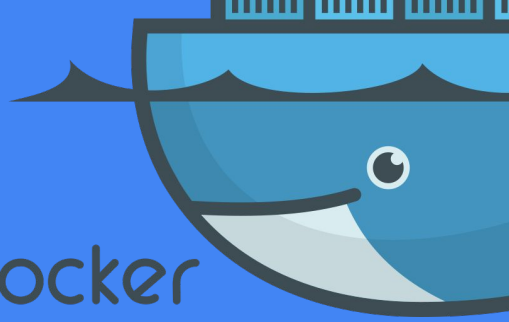
hostname: hostname do container dentro da sua rede, por default, é usado o service-id.

docker-compose.yml

```
version: '3.8'

services:
  web:
    container_name: ${APP_NAME}-web
    restart: always
    hostname: web
    build:
      dockerfile: ./docker/web/Dockerfile.local
      context: .
      args:
        environ: production
```

service-id

docker-compose : build



dockerfile: especifica qual arquivo Dockerfile será utilizado no build.

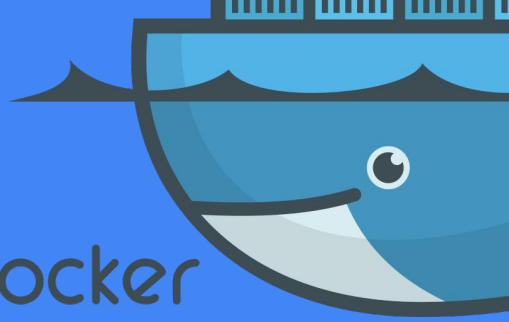
context: indica a partir de qual diretório estará disponível para os comandos COPY/ADD.

args: variáveis a serem repassadas para os ARG`s do Dockerfile durante o build.

docker-compose.yml

```
version: '3.8'

services:

  web:
    container_name: ${APP_NAME}-web
    restart: always
    hostname: web
    build:
      dockerfile: ./docker/web/Dockerfile.local
      context: .
      args:
        environ: development
```

docker-compose : ports



Mapear portas do seu localhost para portas internas do container.

http://localhost *(porta 80)*
https://localhost *(porta 443)*

Por exemplo, caso a porta locahost:80 esteja ocupada, você pode mapear a porta 8080 para container nginx:80.
http://localhost:8080

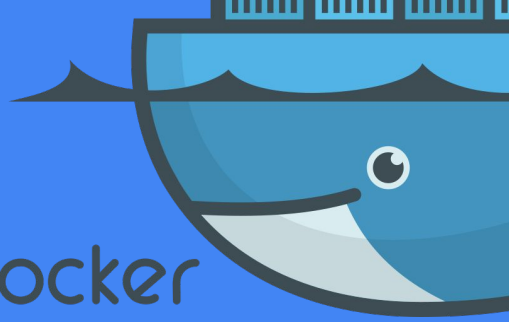
docker-compose.yml

```
version: '3.8'

services:

  nginx:
    container_name: ${APP_NAME}-nginx
    restart: always
    hostname: nginx
    build:
      dockerfile: ./docker/nginx/Dockerfile.local
      context: .
    ports:
      - 80:80
      - 443:443
```

Mapeando localhost:8080 para nginx:80

```
    ports:
      - 8080:80
```

docker-compose : environment

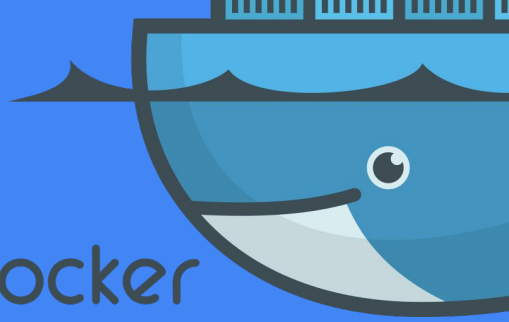


Define variáveis de ambiente a serem utilizadas pelo container.

docker-compose.yml

```yaml
version: '3.8'

services:

  mysql:
    container_name: ${APP_NAME}-mysql
    hostname: mysql
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: ${DB_NAME}
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD: ${DB_PASS}
```

docker-compose : volumes



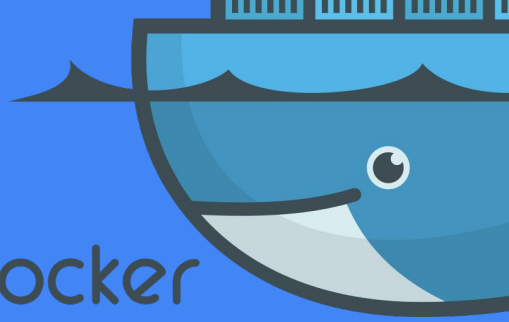
Compartilhar um diretório de repositório local com o container.

Todo volume montado sobrepõe os arquivos existentes na imagem.

docker-compose.yml

```
version: '3.8'

services:

  nginx:
    container_name: ${APP_NAME}-nginx
    restart: always
    hostname: nginx
    build:
      dockerfile: ./docker/nginx/Dockerfile.local
      context: .
    ports:
      - 80:80
      - 443:443
    volumes:
      - ./src/web:/usr/share/nginx/html
```

docker-compose : volumes



Alguns containers geram alguns diretórios que não são necessários serem mantidos na origem do compartilhamento.

Ex. node_modules/, vendor/, etc.

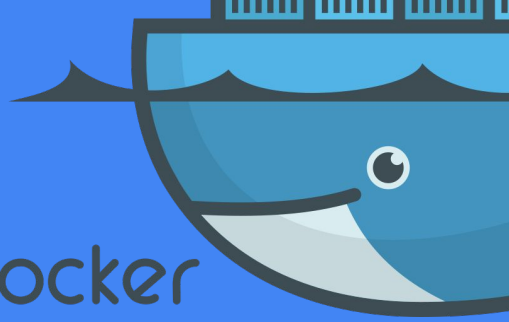
Ao inserí-lo como volume ele apenas manterá o conteúdo dentro do container e localmente estará vazio.

docker-compose.yml

```yaml
version: '3.8'

services:

  web:
    container_name: ${APP_NAME}-web
    restart: always
    hostname: web
    build:
      dockerfile: ./docker/web/Dockerfile.local
      context: .
    ports:
      - 80:80
      - 443:443
    volumes:
      - ./src/web:/app
      - /app/node_modules
```

Facilita o processo de limpeza de disco, evitando armazenar arquivos desnecessários.

docker-compose : links



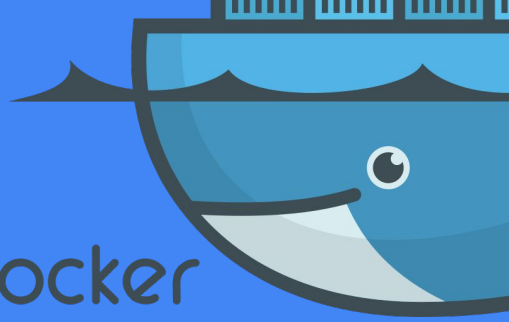
Quando um container necessita conectar-se a outro, é necessário utilizar o ***links***.

Obs.: não aguarda o container estar pronto para o uso, apenas estabelece o canal de conexão.

docker-compose.yml

```
version: '3.8'

services:

  mysql:
    image: mysql:5.7

  web:
    container_name: ${APP_NAME}-web
    restart: always
    hostname: web
    build:
      dockerfile: ./docker/web/Dockerfile.local
      context: .
    ports:
      - 80:80
      - 443:443
    volumes:
      - ./src/web:/app
    links:
      - mysql
```

docker-compose : startup



O processo de *build* das imagens pode ser executado invidualmente ou em conjunto com o comando de inicialização das instâncias dos containers.

O *build* somente é necessário ser executado após uma alteração em arquivos da imagem. Ex. Dockerfile

Gerar imagens definidas no docker-compose.yml

```
$ docker-compose build
```

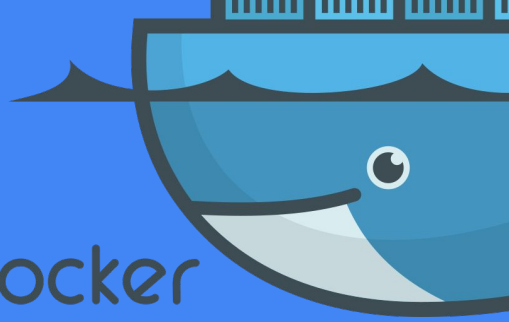
Inicializar o ambiente

```
$ docker-compose up
ou
$ docker-compose up --build
```

Parar o ambiente

```
$ docker-compose down
```

docker-compose



As instâncias mantém um cache de execução para acelerar a inicialização, é possível forçar a re-criação com o ***--force-recreate***.

Forçar re-criação dos container

```
$ docker-compose up --force-recreate
```

Se por acidente você remover um service do docker-compose.yml enquanto ele estiver em execução, irá precisar utilizar o ***--remove-orphans*** para pará-lo.

Remover uma instância que esta sendo executada e não está presente no docker-compose.yml

```
$ docker-compose up --remove-orphans
```

docker learn

Iniciando com docker-compose

https://docs.docker.com/compose/gettingstarted/