PROGRAMAÇÃO CONCORRENTE E DISTRIBUÍDA

Docker Compose

Aula 05 — Orquestrando múltiplos containers com um só comando

Prof. Newton S. B. Miyoshi

newton.miyoshi@baraodemaua.br

Centro Universitário Barão de Mauá — Ribeirão Preto

PARTE 1

O Problema

Por que gerenciar containers um a um não escala.

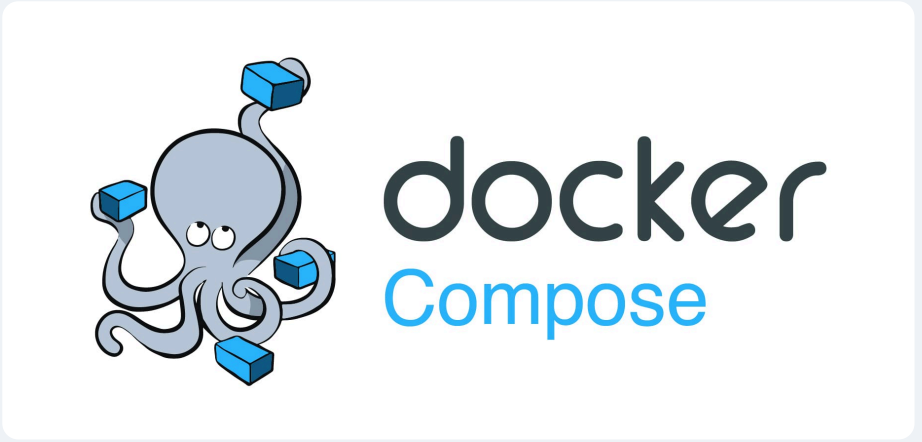
Cada mudança, um ciclo manual

- Situação real: a cada alteração na imagem é preciso parar o container, fazer o build e subir o container de novo.
- Em desenvolvimento, esse processo se repete várias vezes.
- Uma aplicação real raramente usa 1 único container
  - ex.: banco de dados, aplicação web, servidor de e-mail, proxy, cache...
- Cada aplicação em seu próprio container → `docker run` repetido para cada uma delas.

Docker Compose

Limitação: o cliente `docker` (CLI) é um wrapper para a REST API do Docker — cada comando afeta um container por vez.

Solução: o Docker Compose — com um único comando é possível configurar e rodar múltiplos containers, com muito mais possibilidades de configuração.



Usado em todos os ambientes

- Produção, staging, teste e desenvolvimento.
- Fluxo de integração contínua.
- Configuração de rotinas de testes automatizados.
- Configuração de ambiente e variáveis.

Anatomia do `compose.yaml`

services

Cada bloco descreve um container: imagem, build, portas, variáveis, volumes...

volumes

Volumes nomeados, compartilhados entre serviços e persistentes entre reinícios.

networks

Por padrão o Compose já cria uma rede e coloca todos os serviços nela — cada serviço vira um hostname nessa rede.

`services` — configurações principais

- `image` / `build` — imagem a rodar (ou como construir).
- `ports` — mapeia `"HOST:CONTAINER"`.
- `environment` — variáveis de ambiente.
- `volumes` — o que persistir ou montar.
- `depends_on` — ordem de inicialização.

```yaml
services:
  web:
    image: nginx:alpine
    ports:
      - "8080:80"
    depends_on:
      - db
    environment:
      - DB_HOST=db

  db:
    image: postgres:16
    environment:
      - POSTGRES_PASSWORD=secret
    ports:
      - "5432:5432"
```

Em `ports`, a porta da esquerda é do seu computador; a da direita, a que a app escuta dentro do container. O serviço `web` fala com o `db` pelo hostname `db` — resolvido pela rede interna do Compose.

Instalação

- Windows e Mac já vêm com o Compose pré-instalado no Docker Desktop.
- Linux exige passos adicionais — docs.docker.com/compose/install

PARTE 2

Exemplo prático: Django

Do `Dockerfile` ao `docker compose up`.

Django com Docker Compose — `Dockerfile`

```
FROM python:3-slim
RUN mkdir /code
COPY . /code
WORKDIR /code/
RUN pip install django django-crispy-forms crispy-bootstrap5
```

Django com Docker Compose — `compose.yaml`

Estrutura do projeto

```
cafecompao/
├── cafecompao/
├── padarias/
├── prototipo/
├── static/
├── templates/
├── compose.yaml
├── Dockerfile
├── manage.py
└── requirements.txt
```

```yaml
services:
  django:
    build: .
    working_dir: /app
    ports:
      - "8000:8000"
    volumes:
      - .:/app:z
    command: sh -c "python manage.py migrate &&
      python manage.py runserver 0.0.0.0:8000"
```

Build da imagem — `docker compose build`

```
$ docker compose build
[+] Building 4.9s (8/10)
 => [django internal] load build definition from Dockerfile
 => [django internal] load metadata for docker.io/library/python:3-alpine
 => [django internal] load .dockerignore
 => CACHED [django 1/5] FROM docker.io/library/python:3-alpine
 => [django 2/5] COPY . /app
 => [django 3/5] WORKDIR /app
 => [django 4/5] RUN pip install django Pillow django-crispy-forms crispy-bootstrap5
```

Rodar as migrações

```
$ docker compose run django python manage.py migrate
[+] Running 1/0
 ✓ Container cafecompao_db  Started
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, padarias, sessions
Running migrations:
  Applying padarias.0001_initial... OK
  Applying padarias.0006_padaria_cestas_endereco... OK
```

Subir o servidor — `docker compose up`

```
$ docker compose up
[+] Running 1/1
 ✓ Container cafecompao-django-1  Recreated
Attaching to django-1
django-1  | Watching for file changes with StatReloader
django-1  | [06/Sep/2024 04:21:58] "GET / HTTP/1.1" 200 3034
django-1  | [06/Sep/2024 04:22:01] "GET /padarias HTTP/1.1" 200 1944
```

ATIVIDADE

Coloque o Docker Compose para rodar

Instruções no GitHub da disciplina.

PARTE 3

Múltiplos serviços

Django + PostgreSQL num só `compose.yaml`.

Django + PostgreSQL

- Um container para o Django.
- Um container para o PostgreSQL.
- Mapeamento de portas entre host e containers.

`compose.yaml` — volumes e serviço db

```yaml
volumes:
  bd_data: {}

services:
  db:
    image: postgres
    container_name: cafecompao_db
    ports:
      - "5432:5432"
    volumes:
      - bd_data:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=cafecompao
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_PORT=5432
```

`compose.yaml` — serviço `django`

```yaml
django:
  build:
    context: .
    dockerfile: ./Dockerfile
  image: newtonsbm/cafecompao_exemplo:latest
  container_name: cafecompao_app
  depends_on:
    - db
  volumes:
    - .:/code
  ports:
    - "3000:8000"
  command: python manage.py runserver 0.0.0.0:8000
```

`depends_on` garante que o `db` suba antes do `django`.

`settings.py` — configurando o Postgres

```python
# Database
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'cafecompao',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'db',
        'PORT': '5432',
    }
}
```

`HOST` é o nome do serviço no `compose.yaml` (`db`), não `localhost` — os containers se enxergam pela rede interna do Compose.

PARTE 4

Hands On: WordPress + MySQL

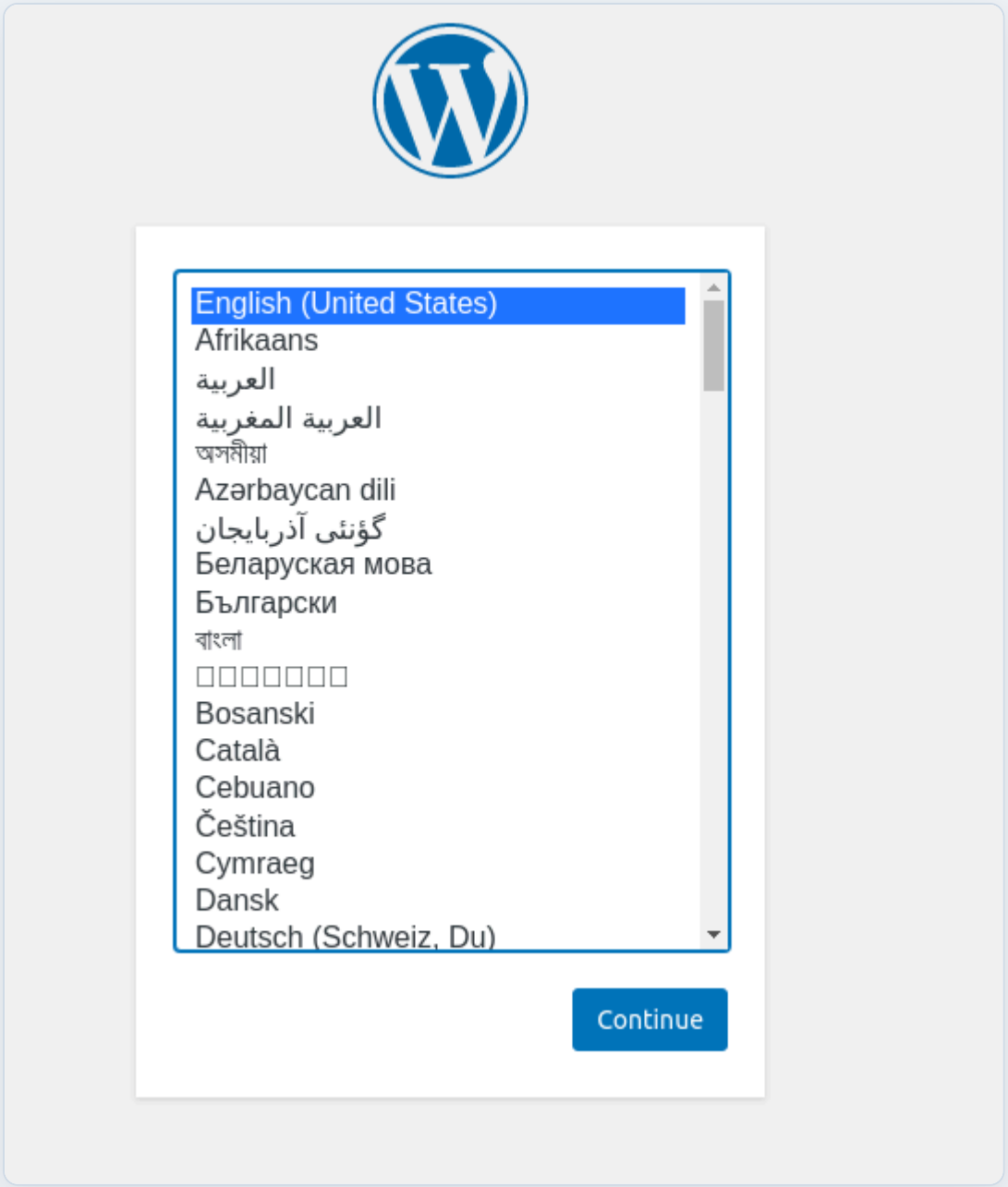
Dois serviços prontos, sem escrever uma linha de Dockerfile.

Hands On: WordPress + MySQL

docker-compose up

```
$ docker-compose up
Creating network "compose_wordpress_default" with the default driver
Creating volume "compose_wordpress_db_data" with the default driver
Pulling db (mysql:5.7)...
Status: Downloaded newer image for mysql:5.7
Pulling wordpress (wordpress:latest)...
Status: Downloaded newer image for wordpress:latest
Creating compose_wordpress_db_1 ... done
Creating compose_wordpress_wordpress_1 ... done
db_1         | Entrypoint script for MySQL Server 5.7.31 started.
wordpress_1  | WordPress not found in /var/www/html - copying now...
```

Assistente de instalação do WordPress



compose.yml — WordPress + MySQL

```yaml
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
```

PARTE 5

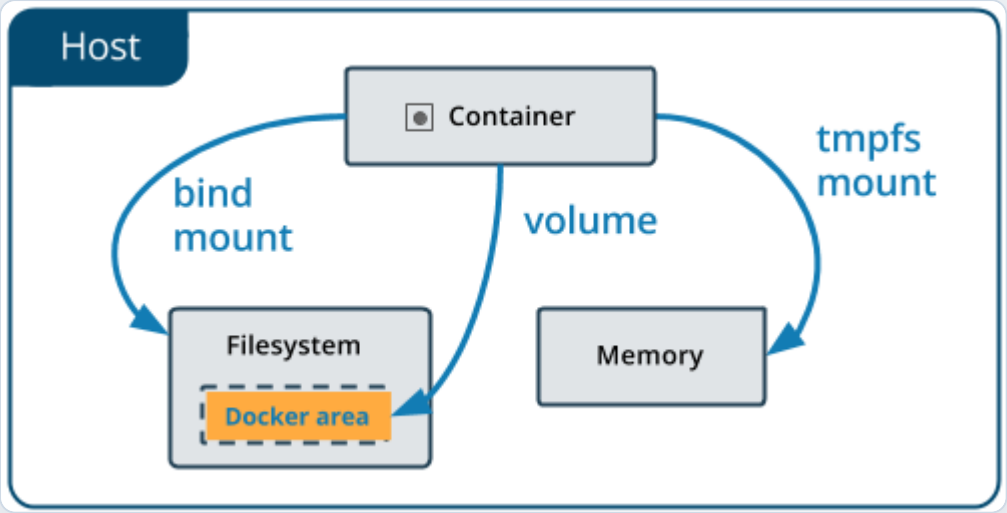
Volumes

Persistindo dados além do ciclo de vida do container.

Volumes

- Mecanismo para persistir dados gerados e usados pelos containers.
- Alternativa: bind mounts.
- Volumes são mais fáceis de fazer backup e migrar do que bind mounts.
- O cliente Docker consegue gerenciar volumes facilmente.
- Volumes podem ser compartilhados entre vários containers.

Bind mount × Volume × tmpfs



Gerenciando volumes

```
$ docker volume create my-vol
$ docker volume ls
$ docker volume rm my-vol
```

docker volume inspect

```
$ docker volume inspect my-vol
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
        "Name": "my-vol",
        "Options": {},
        "Scope": "local"
    }
]
```

Usando um volume num container — `--mount`

```
$ docker run -d \
  --name devtest \
  --mount source=myvol2,target=/app \
  nginx:latest
```

`compose.yml` — volumes nomeados e bind mount

```yaml
services:
  web:
    image: nginx:alpine
    volumes:
      - type: volume
        source: mydata
        target: /data
        volume:
          nocopy: true
      - type: bind
        source: ./static
        target: /opt/app/static

  db:
    image: postgres:latest
    volumes:
      - "dbdata:/var/lib/postgresql/data"

volumes:
  mydata:
  dbdata:
```

Sintaxe longa (`type` / `source` / `target`) permite opções extras; a curta (`origem:destino`) é só um atalho.

ATIVIDADE

Implementar Docker Compose

Implemente a composição de serviços de uma aplicação utilizando Docker Compose.

> **INSPIRAÇÃO**
> Explore exemplos prontos em github.com/docker/awesome-compose.

Suba os arquivos no repositório da disciplina.

MATERIAL EM VÍDEO

YOUTUBE Docker Compose — O guia definitivo youtube.com/watch?v=uH0QCxn9Czo

Referências bibliográficas

- DOCKER. Dockerfile reference. docs.docker.com/engine/reference/builder
- DOCKER. Dockerfile best practices. docs.docker.com/develop/develop-images/dockerfile_best-practices