

Virtualização

Introdução Engenharia Informática

Mário Antunes

October 13, 2025

Exercícios

Laboratório Prático: Trabalhar com Docker Compose

Objetivo: Este laboratório irá guiá-lo através dos fundamentos da criação, gestão e implementação de aplicações usando o Docker (com foco em ficheiros Compose). Irá aplicar os conceitos de imagens, contentores, volumes e redes para construir e executar aplicações de serviço único e multi-serviço.

Pré-requisitos:

- Um computador com um navegador web moderno e um editor de texto.
 - Docker e Docker Compose instalados.
-

Instalar o Docker no Debian

Se estiver a usar um anfitrião Linux, siga estes passos no seu terminal para instalar a versão mais recente do Docker. Baseado nestas [instruções](#).

1. Configurar o repositório apt do Docker:

```
# Remover pacotes não oficiais do docker
sudo apt remove docker.io docker-doc \
docker-compose podman-docker containerd runc

# Atualizar o índice de pacotes e instalar pré-requisitos
sudo apt update
sudo apt install ca-certificates curl

# Adicionar a chave GPG oficial do Docker
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg \
-o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Adicionar o repositório às fontes do Apt:
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
https://download.docker.com/linux/debian \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
```

2. Instalar os pacotes Docker:

```
sudo apt install docker-ce docker-ce-cli containerd.io \
docker-buildx-plugin docker-compose-plugin
```

3. Gerir o Docker como um utilizador não-root (Recomendado):

Para executar comandos docker sem sudo, adicione o seu utilizador ao grupo docker.

```
sudo usermod -aG docker $USER
```

Importante: Tem de fazer logout e login novamente para que esta alteração tenha efeito.

Exercício 1: “Hello, World” com Docker Compose

Objetivo: Compreender a estrutura básica de um ficheiro `compose.yml` e executar uma imagem pré-construída.

1. Crie uma nova pasta para este exercício (p. ex., `ex1-helloworld`).
2. Dentro da pasta, crie um novo ficheiro chamado `compose.yml` com o seguinte conteúdo:

```
services:  
  hello:  
    image: hello-world
```

3. Abra o seu terminal nesta pasta e execute a aplicação.

```
$ docker compose up
```

4. Observe o resultado. O contentor `hello-world` irá arrancar, imprimir a sua mensagem e depois terminar.

5. Limpe o contentor criado pela execução.

```
$ docker compose down
```

Exercício 2: Construir uma Imagem de Servidor Web Personalizada

Objetivo: Usar um `Dockerfile` com o Docker Compose para criar uma imagem de aplicação autónoma.

1. Crie uma nova pasta (`ex2-build`) e uma subpasta dentro dela chamada `my-website`.
2. Dentro de `my-website`, crie um ficheiro chamado `index.html`:

```
<!DOCTYPE html>  
<html>  
  <body>  
    <h1>Esta página foi construída dentro da imagem Docker!</h1>  
  </body>  
</html>
```

3. Na raiz da pasta `ex2-build`, crie um `Dockerfile`:

```
FROM nginx:alpine  
COPY ./my-website /usr/share/nginx/html
```

4. Finalmente, crie o seu ficheiro `compose.yml`:

```
services:  
  webserver:  
    build: .  
    ports:  
      - "8080:80"
```

5. Construa e inicie o serviço. A flag `-d` executa-o em segundo plano.

```
$ docker compose up --build -d
```

6. Abra o seu navegador e navegue para `http://localhost:8080`. Deverá ver a sua página web personalizada.
-

Exercício 3: Desenvolvimento em Tempo Real com Volumes

Objetivo: Compreender como os volumes lhe permitem alterar o conteúdo do seu site sem reconstruir a imagem.

1. Crie uma nova pasta (ex3-volumes) com a mesma estrutura my-website/index.html do exercício anterior.

2. Crie um ficheiro compose.yml. Desta vez, vamos usar a imagem padrão nginx:alpine e montar a nossa pasta local como um volume. **Não é necessário Dockerfile.**

```
services:  
  webserver:  
    image: nginx:alpine  
    ports:  
      - "8080:80"  
    volumes:  
      - ./my-website:/usr/share/nginx/html
```

3. Inicie o serviço: docker compose up -d.

4. Abra o seu navegador em http://localhost:8080 para confirmar que está a funcionar.

5. **Atualização em Tempo Real:** Enquanto o contentor está a correr, **edite o ficheiro index.html** na sua máquina anfitriã. Altere o cabeçalho para <h1>Atualização em tempo real com um Volume!</h1>.

6. Guarde o ficheiro e **atualize o seu navegador**. A alteração aparece instantaneamente!

Exercício 4: Conteúdo Rico em Cache com Varnish & NGINX ⚡

Objetivo: Construir uma aplicação web de duas camadas com uma cache Varnish a servir uma página web rica a partir de um backend NGINX.

1. **Criar a Estrutura de Ficheiros:**

- Crie uma nova pasta (p. ex., ex4-varnish-cache).
- Dentro dela, crie duas subpastas: varnish e my-dynamic-website.

2. **Criar o Conteúdo Web:**

- Encontre um GIF animado divertido online e guarde-o dentro de my-dynamic-website como animation.gif.

- Dentro de my-dynamic-website, crie um ficheiro index.html para exibir o GIF:

```
<!DOCTYPE html>  
<html lang="pt">  
  <head>  
    <meta charset="UTF-8">  
    <title>Teste de Cache Varnish</title>  
    <style> body { font-family: sans-serif; text-align: center; } </style>  
  </head>  
  <body>  
    <h1>Esta página está a ser servida pela cache do Varnish!</h1>  
      
  </body>  
</html>
```

3. **Criar a Configuração do Varnish:**

- Dentro da pasta varnish, crie um ficheiro chamado default.vcl. Isto diz ao Varnish onde encontrar o servidor NGINX.

```
vcl 4.1;  
backend default {  
  .host = "nginx";  
  .port = "80";  
}
```

4. **Criar o Ficheiro Compose:**

- Na raiz da sua pasta ex4-varnish-cache, crie o compose.yml:

```

services:
  cache:
    image: varnish:stable
    volumes:
      - ./varnish:/etc/varnish
    ports:
      - "8080:80"
    depends_on:
      - nginx

  nginx:
    image: nginx:alpine
    volumes:
      - ./my-dynamic-website:/usr/share/nginx/html

```

5. Executar e Verificar:

- Inicie os serviços: `docker compose up -d`.
 - Abra o seu navegador em `http://localhost:8080`. Deverá ver a sua página web com o GIF. O ponto-chave aqui é que foi o **Varnish** que lhe serviu a página, não o NGINX diretamente.
 - **Veja a cache em ação:** Verifique os logs do NGINX para o primeiro pedido.
\$ `docker compose logs nginx`
 - Agora, atualize a página do seu navegador várias vezes. Verifique os logs do `nginx` novamente. Deverá ver **nenhum novo registo de log**, porque o Varnish está a servir o conteúdo da sua cache sem contactar o backend NGINX.
-

Exercício 5: Implementar uma Aplicação do Mundo Real

Objetivo: Aprender a ler documentação oficial e a implementar um serviço complexo auto-hospedado à sua escolha.

1. **Escolha um Serviço:** Vá a [LinuxServer.io](#) e navegue pela lista de imagens populares. Escolha uma que lhe interesse, por exemplo:

- **Jellyfin:** Um servidor de multimédia para os seus filmes e música.
- **Nextcloud:** Uma nuvem pessoal para ficheiros, contactos e calendários.
- **Home Assistant:** Uma plataforma de automação residencial de código aberto.

2. **Leia a Documentação:** Na página da imagem escolhida, encontre a secção “Docker Compose”. Leia-a com atenção, prestando especial atenção aos **volumes** e **variáveis de ambiente** necessários.

- **Volumes** (- `./config:/config`): É aqui que a configuração da aplicação será armazenada na sua máquina anfitriã.
- **Variáveis de Ambiente (PUID, PGID, TZ):** Estas são críticas. TZ define o seu fuso horário (p. ex., Europe/Lisbon). PUID e PGID garantem que os ficheiros criados pelo contentor têm a propriedade correta. Em Linux/macOS, encontre o seu ID executando o comando `id` no seu terminal. Um valor comum é `1000`.

3. **Crie o seu `compose.yml`:** Com base na documentação, crie o ficheiro. Aqui está um exemplo para o **Jellyfin**:

```

services:
  jellyfin:
    image: lscr.io/linuxserver/jellyfin:latest
    container_name: jellyfin
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/Lisbon
    volumes:
      - ./config:/config
      - ./series:/data/tvshows
      - ./filmes:/data/movies
    ports:

```

```
- "8096:8096"
restart: unless-stopped
```

4. Prepare e Implemente:

- Crie as pastas locais que definiu nos seus volumes (p. ex., `mkdir config series filmes`).
- Execute a aplicação: `docker compose up -d`.

5. Explore: Verifique a documentação para o número da porta padrão. Para o Jellyfin, é 8096. Abra o seu navegador em `http://localhost:8096` e siga o assistente de configuração para o seu novo serviço!