POS TECH

FULL STACK DEVELOPMENT

FASE 2 | AULA 03 -

ORQUESTRAÇÃO DE CONTÊINERES

SUMÁRIO

O QUE VEM POR AÍ?

Nesta aula, você aprenderá como orquestrar containers e como escrever um arquivo .yml, que fará a criação dos containers de forma automatizada, baseada nas instruções escritas no arquivo. Além disso, você também aprenderá como os comandos funcionam e entenderá minimamente o conceito de docker-compose e as suas funcionalidades. Vamos também rodar uma aplicação Wordpress que utiliza um banco MySQL e testar o acesso via Web.

HANDS ON

Vamos escrever nosso primeiro arquivo docker-compose.yml pelo VSCode e rodá-lo via terminal do WSL, criando os containers de forma automatizada e acessando-os via Web.

SAIBA MAIS

Sabemos que o Docker Compose gerencia múltiplos containers e que tudo é escrito e definido via arquivo docker-compose.yml. Mas também é possível criar múltiplos containers de maneiras diferentes com Docker Compose. É possível criar um serviço com o Compose apontando um arquivo Dockerfile.

A seguir, você aprenderá mais sobre os comandos do Docker Compose e entenderá em detalhes a responsabilidade de cada passo do arquivo e qual a sua funcionalidade:

- **version:** define as atualizações da sintaxe e steps aceitos no arquivo docker-compose.yml.
- **network:** cria uma rede específica para os containers e os serviços podem apontar para essa rede se precisarem se comunicar.
- **volume:** cria um disco para ser utilizado por containers já criados. Podem conter mais de um volume e são criados dentro do bloco volume, com o nome do volume seguido de dois pontos (:).

Existem diversas instruções dentro do bloco services. As principais e mais usadas são as seguintes:

- **depends_on:** define que a aplicação “x” só poderá ser iniciada depois que a aplicação “y” estiver no ar.
- **image:** aponta uma imagem que será utilizada no serviço.
- **ports:** porta utilizada para o serviço. Nesse campo, temos portaDockerHost:portaContainer. A portaContainer é a porta que estará acessível entre serviços (Containers), já a portaDockerHost é a porta usada externamente via Web. Por exemplo: se definir o ports 8002:3306, significa que outros containers devem se conectar com o banco chamando a porta 3306 e, se precisar acessar externamente (local), o banco estará acessível pela porta 8002 (localhost:8002).

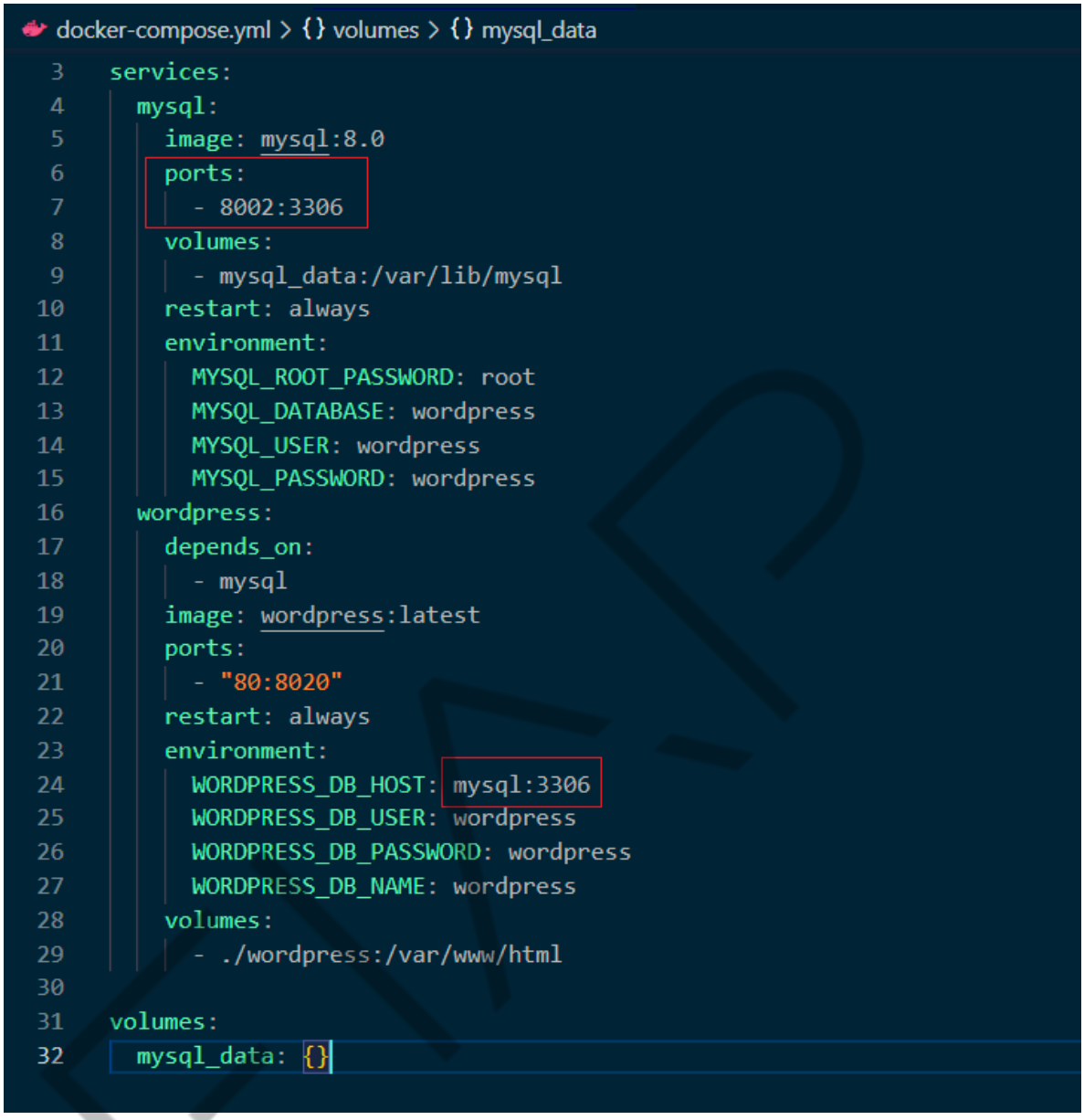


Figura 1 – Ports Docker Compose
Fonte: Elaborado pelo autor (2023)

- **restart**: são políticas que definem quando o container pode ser reiniciado. Entre as políticas, é possível definir:
  - **no:** nesse caso os containers não serão reiniciados.
  - **on-failure:** reinicia o container em caso de falha. Uma falha é caracterizada se a saída for diferente de zero.
  - **always:** nessa opção, o container será reiniciado sempre que ele parar de funcionar.

- **unless-stopped:** reinicia sempre o container, a menos que ele seja parado de forma intencional. É usado para deixar um serviço “X” desligado.

Cada serviço pode ter sua política de restart específica:

- **environment:** usado para setar variáveis de ambiente. Muitos serviços podem deixar variáveis vazias para que sejam definidos os valores externos. Dessa forma, o environment dá os valores dessas variáveis. É possível utilizar env_file ao invés de environment. O **env_file** pode apontar pra um arquivo .env qualquer dentro do host.
- **volumes:** apontam para um volume criado. É usado para apontar pastas específicas dentro de um volume.
- **command e entrypoint:** é possível alterar o start principal responsável pela execução do container. Ao utilizar esse comando, o CMD ou ENTRYPOINT presente na imagem ou Dockerfile serão substituídos pelo comando ou entrypoint do compose.

Os comandos mais empregados para utilização do docker-compose são:

- **docker compose up:** cria e inicia os containers.
- **docker compose build:** realiza apenas a etapa de build das imagens que serão utilizadas. Facilita muito na hora do “up”.
- **docker compose logs:** visualiza os logs dos containers.
- **docker compose restart:** reinicia os containers.
- **docker compose ps:** lista os containers.
- **docker compose scale**: permite aumentar o número de réplicas de um container.
- **docker compose start:** inicia os containers.
- **docker compose stop:** paralisa os containers.
- **docker compose down:** paralisa e remove todos os containers e seus componentes, como rede, imagem e volume.

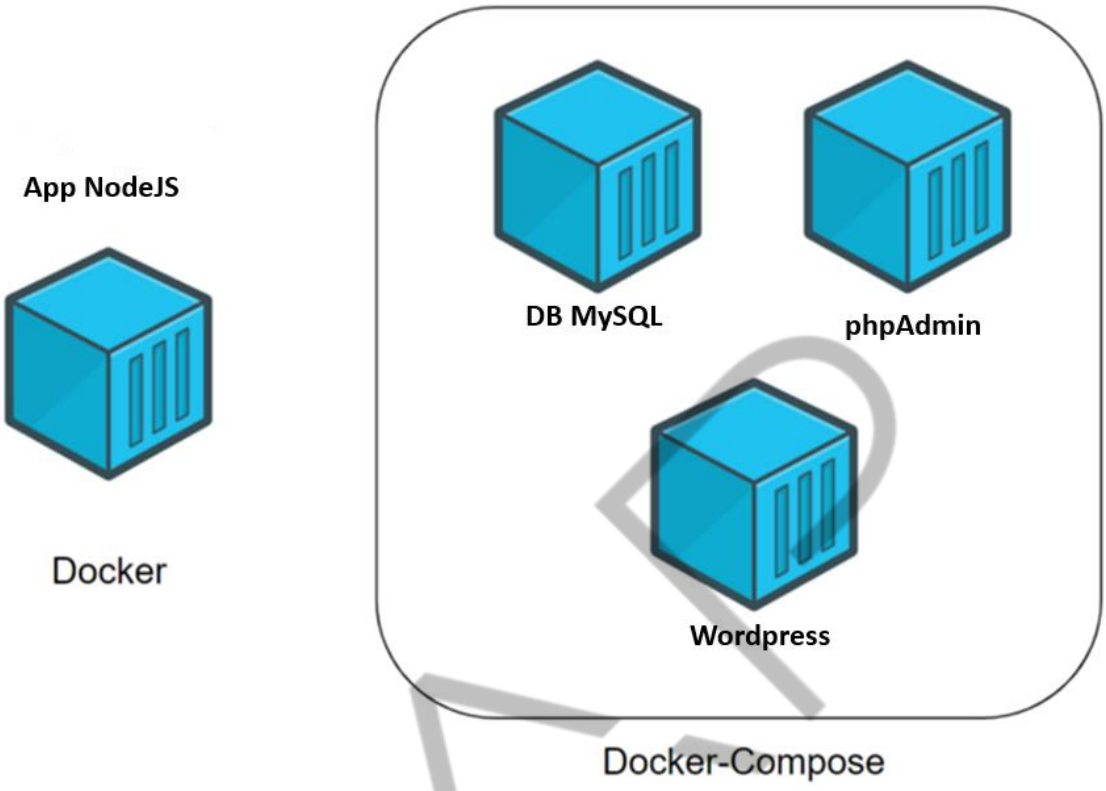
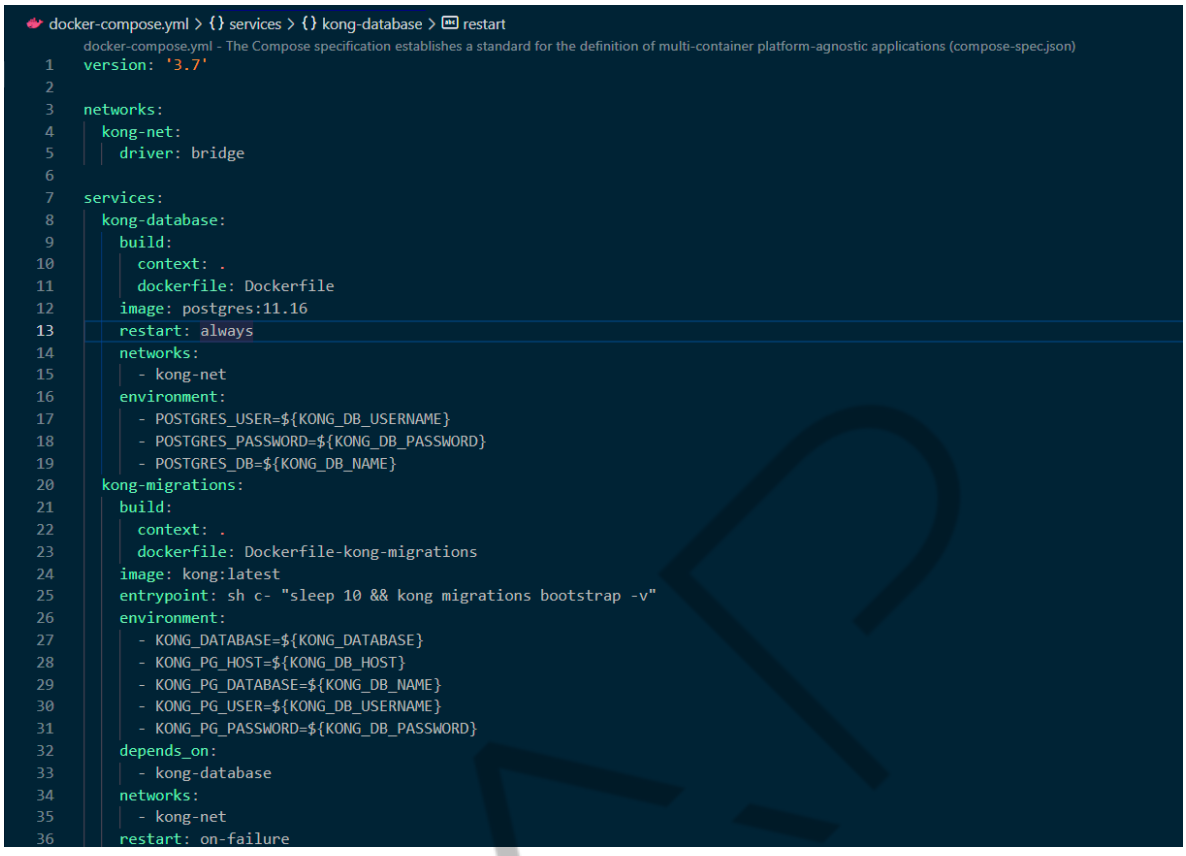


Figura 2 – Diferença entre Docker e Docker Compose
Fonte: Elaborado pelo autor (2023)

Os serviços em questão são: banco de dados Postgres (pego de um Dockerfile criado localmente), as migrations executadas no banco para atualizar as tabelas, e o serviço Kong, que possui um api gateway com versão open source, chamada “Konga”. Ela executa um comando que prepara o front end criado pela comunidade, facilitando a criação de entradas no Kong Api Gateway.

Por fim, nesse exemplo utilizaremos os comandos .env, Dockerfiles e arquivo .sql para criar um usuário dentro do banco, gerando uma instrução não vista chamada “links”. Ela será responsável por “linkar” um container com o outro, além da instrução build, que é encarregada de “buildar” a imagem do Dockerfile. Temos ainda o contexto, que define o local onde o Dockerfile criado estará. Neste caso, como ele está na raiz, utilizaremos um ponto.



```yaml
kong:
  build:
    context: .
    dockerfile: Dockerfile-kong
  image: kong:latest
  environment:
    - KONG_DATABASE=${KONG_DATABASE}
    - KONG_PG_HOST=${KONG_DB_HOST}
    - KONG_PG_DATABASE=${KONG_DB_NAME}
    - KONG_PG_USER=${KONG_DB_USERNAME}
    - KONG_PG_PASSWORD=${KONG_DB_PASSWORD}
    - KONG_PROXY_ACCESS_LOG=${KONG_PROXY_ACCESS_LOG}
    - KONG_ADMIN_ACCESS_LOG=${KONG_ADMIN_ACCESS_LOG}
    - KONG_PROXY_ERROR_LOG=${KONG_PROXY_ERROR_LOG}
    - KONG_ADMIN_ERROR_LOG=${KONG_ADMIN_ERROR_LOG}
  restart: on-failure
  ports:
    - $KONG_PROXY_PORT:8000
    - $KONG_PROXY_SSL_PORT:8443
  depends_on:
    - kong-database
    - kong-migrations
  networks:
    - kong-net
kong-prepare:
  image: pantsel/konga:next
  command: "-c prepare -a postgres -u postgresql://kong:kong@kong-database:5432/konga"
  networks:
    - kong-net
  restart: on-failure
  links:
    - kong-database
  depends_on:
    - kong-database
konga:
  image: pantsel/konga:next
  environment:
    - TOKEN_SECRET=${KONGA_TOKEN_SECRET}
    - DB_ADAPTER=${KONGA_DATABASE}
    - DB_HOST=${KONGA_DB_HOST}
    - DB_PORT=${KONGA_DB_PORT}
    - DB_DATABASE=${KONGA_DB_NAME}
    - DB_USER=${KONGA_DB_USERNAME}
    - DB_PASSWORD=${KONGA_DB_PASSWORD}
    - NODE_ENV=${KONGA_NODE_ENV}
  restart: always
  ports:
    - ${KONGA_PORT}:${KONGA_PORT}
  depends_on:
    - kong-database
    - konga-prepare
  networks:
    - kong-net
```

Figura 3 – docker-compose.yml Kong API Gateway
Fonte: Elaborado pelo autor (2023)

O QUE VOCÊ VIU NESTA AULA?

Nesta aula, você aprendeu como gerenciar múltiplos containers com Docker Compose, como criar um arquivo .yml, os seus comandos mais usados e detalhes do que cada um faz e como se comporta com o docker compose.

Você também viu uma imagem avançada do docker compose e entendeu como e onde as imagens podem ser salvas. Viu também como a rede de um container funciona e se integra ao nosso computador. Depois, criamos nosso primeiro volume e o montamos no container. Por fim, vimos mais sobre os logs de um container, tudo executado via WSL Ubuntu.

Se você ficou com alguma dúvida ou se quer aprender mais sobre o assunto desta aula, acesse a comunidade do curso no Discord e assista às videoaulas. Aproveite ao máximo a sua jornada do conhecimento. Até mais!

REFERÊNCIAS

Compose Docker Specification. Docs.docker.com. [s.d]. Disponível em: <https://docs.docker.com/compose/compose-file/develop/>. Acesso em: 08 jan. 2024.

Docker Compose Restart Policies. Baeldung. [s.d]. Disponível em: <https://www.baeldung.com/ops/docker-compose-restart-policies>. Acesso em: 08 jan. 2024.

Dominando o Docker Compose. BLOG.4LINUX.COM.BR. [s.d]. Disponível em: <https://blog.4linux.com.br/docker-compose-explicado/>. Acesso em: 08 jan. 2024.

PALAVRAS-CHAVE

Container. Docker Compose. Linux. WSL. YAML.