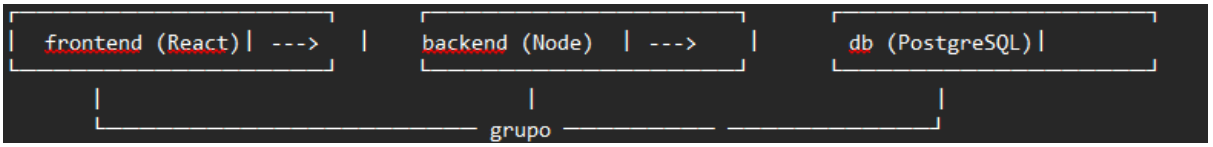
DOCKER-COMPOSE

O Docker Compose é uma ferramenta usada para definir e executar aplicações multi-contêiner no Docker. Ele permite que você descreva todos os serviços, redes e volumes necessários para rodar sua aplicação em um único arquivo de configuração (geralmente docker-compose.yml), facilitando a orquestração e a execução dos contêineres necessários para a aplicação funcionar.



OBS:Os 3 containers agrupados no mesmo grupo.
No Docker o grupo se localiza no containers, clicando no grupo se vê os containers separados em questão. É um grupo de containers onde cada container está fazendo/rodando sua aplicação.

Commands CLI Docker composse:
Docker-compose up - sobe todos os serviços/containers para a excução.
Docker-compose down - Para todos os serviços/containers que estão em execução.
Docker-composse restart - Reinicia todos os serviços/contariners que estão em execução
Todos esses se pode passar um parâmetro especifico com o nome/id do containers para aplicar so nele o comando, Exemplo:
Docker-composse up dotnet_projeto.

Montando Docker composse:
Nome do arquivo - Sempre docker-compose.yml
Version -Se inicia com a versão do Docker compose que está sendo usado, isso depende da versão do Docker que se está usando + a versão do composer geralmente 3.8 para as mais novas.
Services - Dentro desse parâmetro que vão vim todos os containers necessários para rodar as aplicações.
<Nome do container_1>: - Nesse momento se coloca o nome do container, como ele vai se chamar.
build:
context: <definir o diretório base a partir do qual o docker vai executar> . É o diretório atual.
dockerfile: <Caminho com base no contexto até o nome do dockerfile que vai ser usado na build>, geralmente é o próprio dockerfile o nome do arquivo.
args: (São variáveis passadas para o dockerfile usadas na construção da image)
-<nomeVariável>=<valor>
enviroment: (São variáveis passadas para o contêiner, (durante a execução))
-<nomeVariavelExistente>
<Nome do container_2>: - Nesse momento se coloca o nome do 2° container, como ele vai se chamar
build:
context: <definir o diretório base a partir do qual o docker vai executar> . É o diretório atual.

dockerfile: <Caminho com base no contexto até o nome do dockerfile que vai ser usado na build>, geralmente é o próprio dockerfile o nome do arquivo.

<Nome do container_3>: - Nesse momento se coloca o nome do 3° container, como ele vai se chamar

build:

context: <definir o diretório base a partir do qual o docker vai executar> . É o diretório atual.

dockerfile: <Caminho com base no contexto até o nome do dockerfile que vai ser usado na build>, geralmente é o próprio dockerfile o nome do arquivo.

<...>

OBS 1: Caso tenha uma image local ou no repositório Docker, ao invés do build, passar image e informar o nome da imagem.

Exemplo:

<Nome do container_3> - Nesse momento se coloca o nome do 3° container, como ela vai se chamar

image: "mysql"

<...>

OBS 2: Também nas versões mais recentes do Docker não precisa informar e colocar a version no topo do arquivo composse.

OBS 3: ARGS - são diferentes das variáveis geradas no dockerfile, pois essas não são visíveis dentro do container, visíveis apenas para a construção da imagem.

Exemplo:

docker-compose:

```
build:
  context: .
  args:
    - region=us-east-1
    - alice=0
```

dockerfile:

```
ARG region
ARG alice   #visível apenas para o dockerfile
ENV utf-8   #visível no container

RUN echo "Região configurada: $region"
RUN echo "Valor de Alice: $alice"
```

Apenas se as variáveis forem colocadas dentro do enviroment que são passadas para o container, durante a execução do container e após. Mas não são visíveis na build da imagem.

Exemplo:

docker-compose:

```
ARG region
build:
  context: .
  args:
    - region=us-east-1
```

```
    - alice=0
environment:
   -  region
   -  DATABASE_URL=mysql://usuario:senha@localhost:3306/meubanco
   -  DEBUG_MODE=true
```

env_file: (Permite carrega variáveis de um arquivo externo)
- .env # Carrega todas as variáveis do arquivo .env.
- mysql/env_vars # Carrega todas as variáveis do arquivo env_vars.

útil para manter as configurações separadas do código e evitar expor informações sensíveis diretamente no YAML.
ports: (Realiza o processo de publicação da porta que está em execução no container para o host>
- "8081:5984"
- "<porta host>:<porta container>"

Similar ao cli do Docker -p (--publish)

volumes: (Realiza o processo de compartilhamento de volume entre o host e o container)
Existem 2 jeitos de criar volumes, o primeiro é especificando o caminho do diretório host explicitamente similar ao bind do cli, o outro jeito é especificando o nome do volume criado no escopo global que é dessa forma o Docker que gerencia similar ao volume do cli.
Exemplo:
Volume do services:
volumes:
- <caminho do diretório que se quer compartilhar do host>:<caminho destino do container>:<modo de acesso>
- <caminho destino do container>:<modo de acesso>
- <nome do volume global>:<caminho destino do container>:<modo de acesso>

Volume escopo global gerenciado pelo Docker.
Volumes:
<nome qualquer para se dar>:
Modos de acesso são o rw (leitura e escrita) e ro (apenas leitura). Essas permissões são destinadas ao container.
Similar ao --mount do cli do docker onde se tem que definir o type, sorce e target (tst).
Se não informar o diretorio da maquina host pela criação de volume padrão, o docker cria um sozinho.
depends_on: (Informa se o serviço tem alguma depêndencia de outro)
- <nome da depêndencia/serviço>

AVISO:
Forma como o Dockerfile e o Docker Compose lidam com variáveis de ambiente e a expansão delas.
Dockerfile: Quando você usa ENV PATH=$DOTNET_ROOT:$PATH no Dockerfile, o Docker já entende que PATH deve ser configurado dessa forma e faz a expansão de variáveis de ambiente durante o processo de construção da imagem.

Docker Compose: Quando você passa variáveis de ambiente no docker-compose.yml, o Docker Compose não expande variáveis durante a construção do contêiner. Ou seja, a variável PATH=$DOTNET_ROOT:$PATH no docker-compose.yml não será expandida automaticamente. Em vez disso, o que é colocado no PATH será literalmente a string "$DOTNET_ROOT:$PATH", não o valor das variáveis.

O YML é case sensitive, tem que usar tudo em letra minúscula em TUDO.
Também ficar atento quando usar o “”, ele não é utilizado para variáveis dentro do compose e nem para passar caminhos de arquivos. PORTS, usa o “” quando for passar as portas.