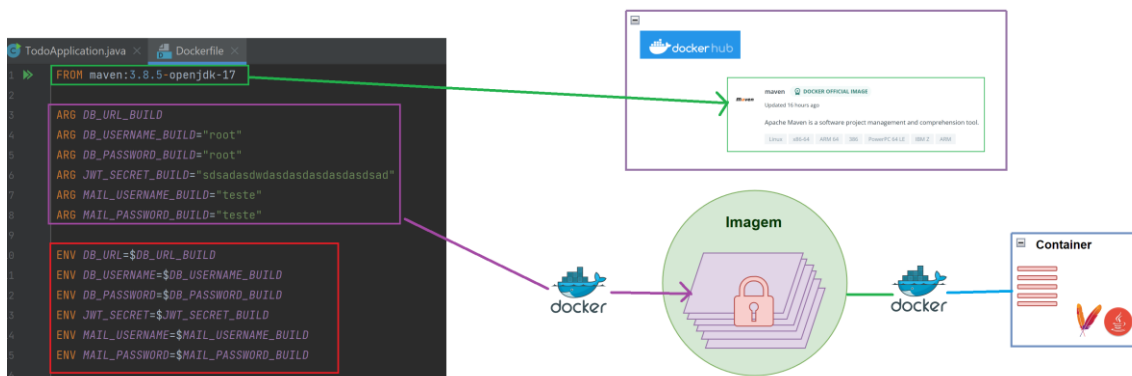




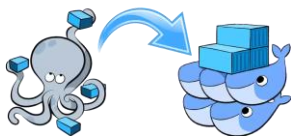
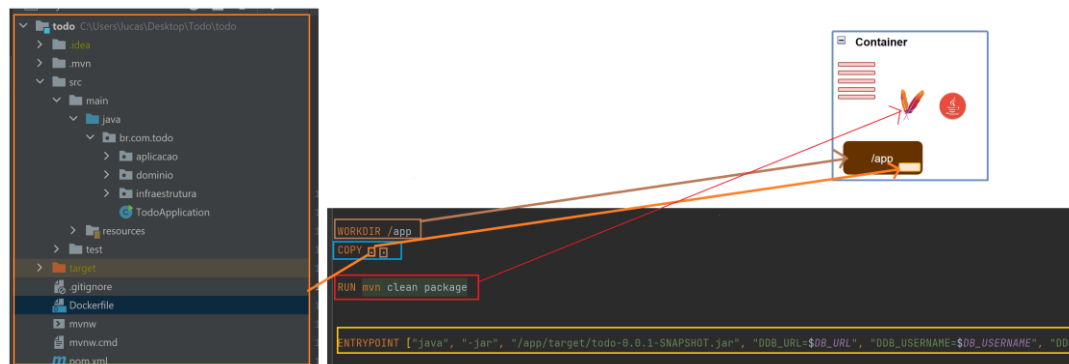
Criando um container para app java

Pra fazer esse “*hands on*” eu usei uma aplicação que desenvolvi, ela não tinha nada de **containerização**, então era minha cobaia perfeita, por enquanto ela é só uma **aplicação backend** que comunica com banco de dados, então eu tinha a noção que precisava de 2 containers.

O primeiro passo foi criar um **dockerfile** que ia gerar a imagem para criar o container da minha aplicação Java, a dockerfile começa com “**FROM**” que nada **mais é do que eu referenciar uma imagem como base dessa minha que está sendo criada**, depois defino “**ARG**” que são **variáveis de ambiente construídas durante o build da imagem** e “**ENV**” que são **variáveis de ambiente que ficam acessíveis dentro do container**:



Depois disso eu defini um **diretório de trabalho com “WORKDIR”**, é só uma **pasta dentro do container** que vai conter a **source da aplicação**. Com “**COPY**” eu **transfiro toda minha pasta da aplicação para o workdir** dentro do **container**. Então nesse **diretório** eu **faço o package do projeto** e executo ele com “**ENTRYPOINT**” passando as **variáveis de ambiente** (já que meu app não tem essas informações hard code):

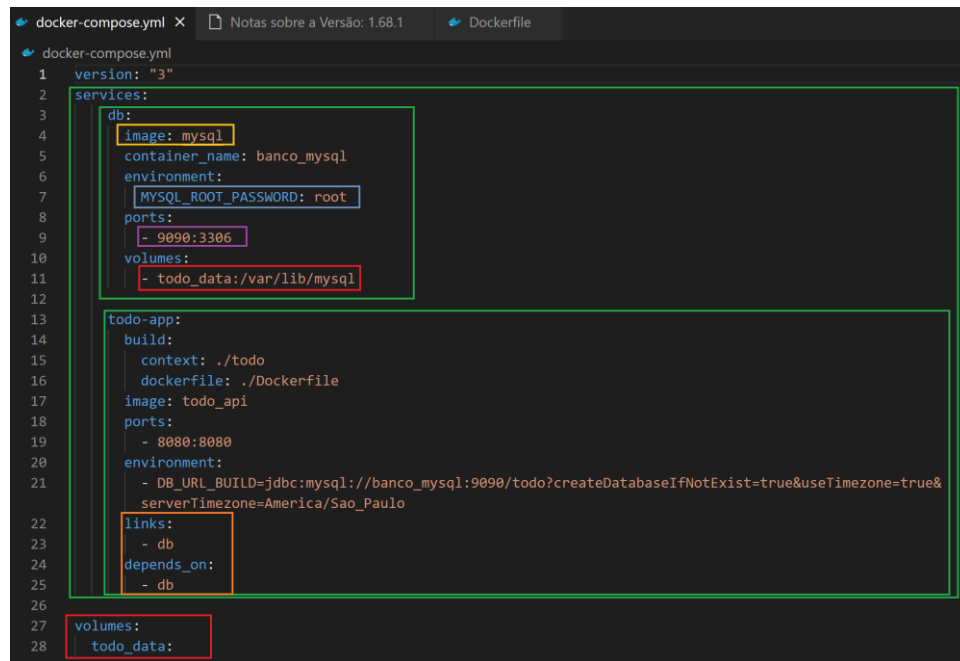


Coordenando containers com docker compose

Bom, o container da aplicação eu já tinha, agora eu preciso do container do banco de dados, e preciso que eles se **conheçam e o build seja automatizado**. Para isso existe o **docker compose**, basicamente falando ele coordena seus containers (*mas não orquestra, isso parece ser um tópico diferente*).

Obviamente o **Docker compose** utiliza um arquivo para poder coordenar os containers, esse arquivo se chama **docker-compose.yml**. Esse arquivo tem uma série de configurações possíveis, então vou listar apenas as principais.

Nesse caso, os **services** são **imagens** que eu quero **containerizar**, no caso do banco **de dados** eu usei a **imagem do Mysql**, a configuração de **environment** pode ser definida de **de acordo com a documentação**, além disso eu precisei usar um **volume** (*já que estou persistindo informações*) e precisei **mapear a porta** que ele vai expor o DB:



```
1 version: "3"
2 services:
3   db:
4     image: mysql
5     container_name: banco_mysql
6     environment:
7       MYSQL_ROOT_PASSWORD: root
8     ports:
9       - 9090:3306
10    volumes:
11      - todo_data:/var/lib/mysql
12
13   todo-app:
14     build:
15       context: ./todo
16       dockerfile: ./Dockerfile
17     image: todo_api
18     ports:
19       - 8080:8080
20     environment:
21       - DB_URL_BUILD=jdbc:mysql://banco_mysql:9090/todo?createDatabaseIfNotExist=true&useTimezone=true&serverTimezone=America/Sao_Paulo
22     links:
23       - db
24     depends_on:
25       - db
26
27 volumes:
28   todo_data:
```

Já pra criar a **imagem da minha aplicação java** eu apenas usei o **"build"** passando o caminho do meu **Dockerfile** dentro da pasta da API (**/todo/Dockerfile**).

Note que a **tag link** ta relacionada com a **imagem db**(por isso no caminho da URL eu consigo usar o nome do container do banco de dados) e também tem o **depends_on** que basicamente faz com que meu app só seja inicializado se o container de banco de dados conseguir ser iniciado.