

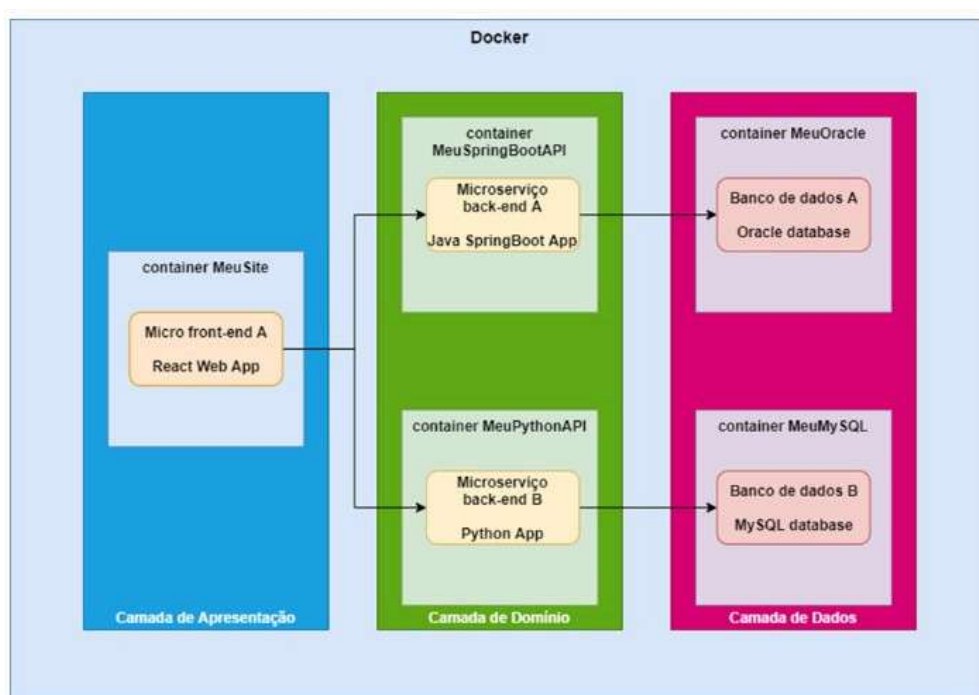
Subindo uma solução completa

Finalmente chegamos para falar do conjunto de soluções que usaremos como modelo para aprofundar no estudo do Docker.

A primeira coisa a se fazer é baixar o repositório que preparei, neste link: <https://gitlab.com/evertton.juniti/descomplica>

No próprio repositório tem várias instruções de como clonar o repositório e como criar algumas coisas, mas vamos com calma.

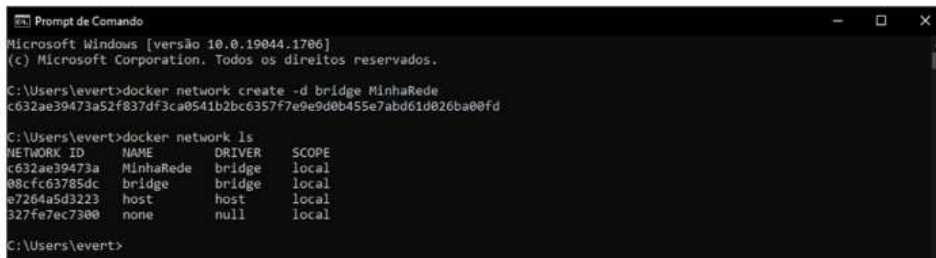
Seguiremos o seguinte desenho:



Passaremos por alguns conceitos para conseguir subir essa solução toda, mas a primeira coisa a fazermos é criar uma “sub-rede” no Docker. Já vimos o potencial de “sub-rede” e a comunicação com o uso de portas e é justamente isso que faremos com essa solução.

Vamos primeiramente criar uma “sub-rede”, para isso podemos usar o seguinte comando:

```
docker network create -d bridge MinhaRede
```



```
Prompt de Comando
Microsoft Windows [versão 10.0.19044.1706]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\event>docker network create -d bridge MinhaRede
c632ae39473a52f837df3ca0541b2bc6357f7e9e9d0b455e7abd61d026ba00fd

C:\Users\event>docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
c632ae39473a        MinhaRede           bridge              local
08cfc63785dc        bridge              bridge              local
e7264a5d3223        host                host                local
327fe7ec7300        none                null                local
C:\Users\event>
```

“docker network” é o comando do Docker para lidarmos com tudo de redes, o comando “create” indica que estamos criando uma rede (uma “sub-rede” no caso), o parâmetro “-d bridge” indica o DRIVER que usaremos que é o tipo bridge que permite a comunicação livre entre containers e o último parâmetro “MinhaRede” é um apelido para a nossa rede.

Recapitulando esse conceito, estamos criando uma “sub-rede” para que as APIs que iremos criar consigam se comunicar com os bancos de dados. Se não fizermos isso a API não “enxergará” esses bancos de dados.

Observação: em ambiente produtivo há outros mecanismos para fazer os containers enxergarem uns aos outros, mas para a prática que estamos montando aqui que será tudo em um único computador (o seu) criamos este artifício para facilitar toda a configuração.

“sub-rede” criada, então agora podemos subir as soluções. Como front-end vai falar com API e a API por sua vez vai falar com o banco de dados, é necessário que essas dependências subam primeiro para que as aplicações não dêem problema na hora de serem executadas. Por este motivo subiremos primeiramente os bancos de dados.

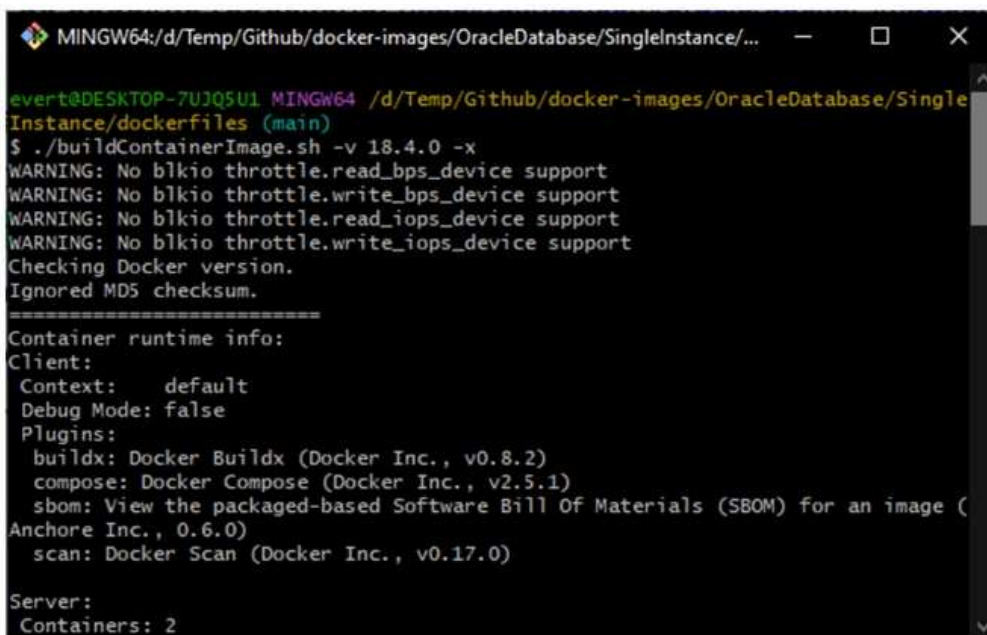
Vamos subir primeiro o banco de dados Oracle! As instruções completas estão no link: https://gitlab.com/everton.juniti/descomplica/-/tree/main/cicd_database/oracle

Neste momento irei resumir os passos, então primeiramente iremos fazer o build de uma imagem do Oracle. O Oracle possui imagens oficiais, porém não são públicas e por este motivo há a necessidade de clonarmos o repositório oficial do Oracle e escolher uma das versões para fazermos o build. Então neste momento, devemos fazer o clone do seguinte repositório: <https://github.com/oracle/docker-images.git>

No local onde você fez o clone do repositório em sua máquina, entre no conjunto de pastas “docker-images\OracleDatabase\SingleInstance\dockerfiles” e abra um terminal bash do Git (o Git Bash mesmo), depois execute o seguinte comando:

```
./buildContainerImage.sh -v 18.4.0 -x
```

Dica: digite o comando mesmo, se você copiar o comando e colar no bash, vai dar um erro e não vai fazer o build! Vai levar bastante tempo para a imagem ser criada, neste momento você terá que ser um pouco paciente.



```
MINGW64:/d/Temp/Github/docker-images/OracleDatabase/SingleInstance/...
evert@DESKTOP-7UJQ5U1 MINGW64 /d/Temp/Github/docker-images/OracleDatabase/SingleInstance/dockerfiles (main)
$ ./buildContainerImage.sh -v 18.4.0 -x
WARNING: No blkio throttle.read_bps_device support
WARNING: No blkio throttle.write_bps_device support
WARNING: No blkio throttle.read_iops_device support
WARNING: No blkio throttle.write_iops_device support
Checking Docker version.
Ignored MD5 checksum.
=====
Container runtime info:
Client:
Context:    default
Debug Mode: false
Plugins:
  buildx: Docker Buildx (Docker Inc., v0.8.2)
  compose: Docker Compose (Docker Inc., v2.5.1)
  sbom: View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc., 0.6.0)
  scan: Docker Scan (Docker Inc., v0.17.0)
Server:
Containers: 2
```

Depois que terminar o build, você verá que uma imagem nova apareceu em seu repositório local chamado “oracle/database”:

```
Selecionar Prompt de Comando

NETWORK ID NAME DRIVER SCOPE
c632ae39473a MinhaRede bridge local
08cfc63785dc bridge bridge local
e7264a5d3223 host host local
327fe7ec7300 none null local

C:\Users\event>docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE
reactsite latest ec3bb7898461 10 hours ago 879MB
springbootapi latest b9f5eb1c6f73 11 hours ago 449MB
pythonapi latest 98412dbee704 11 hours ago 198MB
nginx latest 0e901e68141f 9 days ago 142MB
mysql latest b2500a44757f 13 days ago 524MB
oracle/database 18.4.0-xe 8075b76e952e 2 weeks ago 6.03GB
registry.gitlab.com/gitlab-org/gitlab-runner/gitlab-runner-helper x86_64-f761588f 91bb0ad5f485 2 weeks ago 66.9MB
gitlab/gitlab-runner latest 89944ac4ab2c 5 weeks ago 691MB
mcr.microsoft.com/mssql/server 2019-latest d78e982c2f2b 4 months ago 1.48GB
docker 19.03.12 81f5749c9058 23 months ago 211MB

C:\Users\event>
```

Antes de subir o container, prepare um conjunto de pastas para usarmos volume no container. Assim guardaremos todos os dados do banco de dados em um local específico. Crie o conjunto de pastas “oracle\oradata” e “oracle\setup” no local de sua preferência. No caso da pasta “oracle\setup”, inclua nela o arquivo “script_inicial.sql” do repositório.

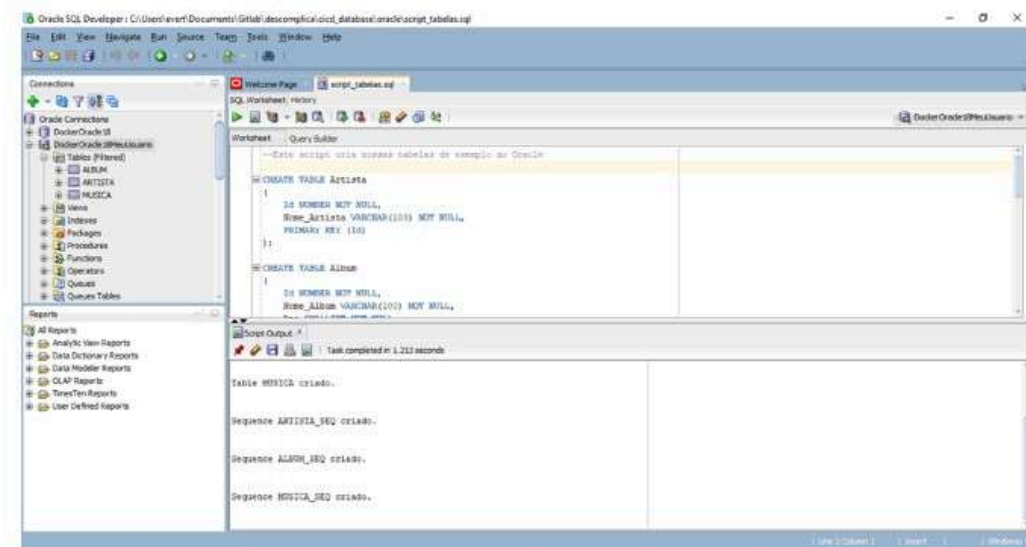
Em seguida agora sim você poderá executar o seguinte comando para subir um banco de dados Oracle:

```
docker run --name MeuOracle --network MinhaRede -v
/D/docker/volumes/oracle/oradata:/opt/oracle/oradata -v
/D/docker/volumes/oracle/setup:/opt/oracle/scripts/setup -p 1521:1521 -p 5500:5500
-e ORACLE_PWD=MinhaSenha -e ORACLE_CHARACTERSET=AL32UTF8 -d
oracle/database:18.4.0-xe
```

Os detalhes de cada parâmetro estão nas instruções do link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_database/oracle

Na primeira vez que subir o Oracle, levará bastante tempo para ele fazer a configuração inicial (aqui levou de 30 a 40 minutos).

Agora vamos nos conectar com o Oracle SQL Developer, as instruções de como instalar a ferramenta e configurar a conexão com o usuário criado no “script_inicial.sql” chamado MeuUsuario estão no link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_database/oracle. Vamos executar o script “script_tabelas.sql” que está neste repositório, isso criará nossas tabelas de exemplo:



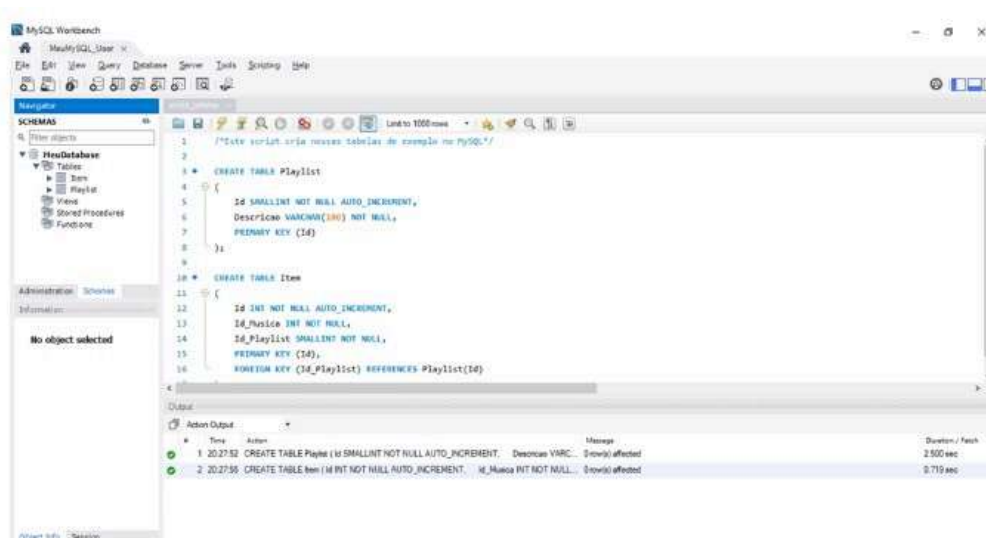
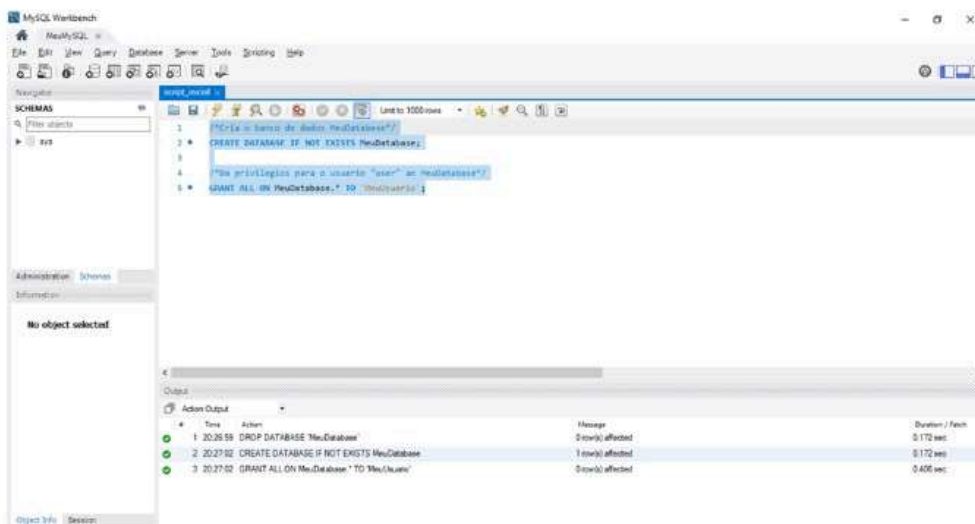
Agora vamos subir o banco de dados MySQL! As instruções completas estão no link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_database/mysql

Aqui iremos resumir, então crie uma pasta chamada “mysql” em um local que você deseje, depois podemos executar o seguinte comando para subir um container:

```
docker run --name MeuMySQL --network MinhaRede -v  
/D/docker/volumes/mysql:/var/lib/mysql -p 3306:3306 -e  
MYSQL_ROOT_PASSWORD=Minha@senha -e MYSQL_USER=MeuUsuario -e  
MYSQL_PASSWORD=MinhaSenha -d mysql:latest
```

Os detalhes de cada parâmetro estão nas instruções do link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_database/mysql

Agora vamos nos conectar com o MySQL Workbench as instruções de como instalar a ferramenta e configurar a conexão com o usuário root e com o usuário customizado criado chamado e MeuUsuario estão no link: https://gitlab.com/everton.juniti/descomplica/-/tree/main/cicd_database/mysql. Vamos executar o script “script_inicial.sql” primeiramente com o usuário root para criar o schema novo e dar permissão ao usuário MeuUsuario que criamos e logo em seguida executaremos o script “script_tabelas.sql” que está neste repositório também só que com o usuário MeuUsuario, isso criará nossas tabelas de exemplo:



Agora vamos subir nossos back-ends! Primeiro vou começar com a API em Java SpringBoot que se conecta ao banco de dados Oracle. As instruções completas

estão

no

link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_backend/springbootapi

Neste caso já há código pronto! Iremos fazer o build do projeto para criar uma imagem desta aplicação customizada e depois subir um container. Vamos olhar o Dockerfile desta solução em Java:

```
#
# Build stage
#
FROM maven:3.8.5-openjdk-17-slim AS build
COPY src /home/app/src
RUN rm -rf /home/app/src/main/resources/application.properties
COPY ./applicationProd.properties
/home/app/src/main/resources/application.properties
COPY pom.xml /home/app
RUN mvn -f /home/app/pom.xml clean package

#
# Package stage
#
FROM openjdk:17-slim
COPY --from=build /home/app/target/springbootapi-1.0.0.jar
/usr/local/lib/springbootapi.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/usr/local/lib/springbootapi.jar"]
```

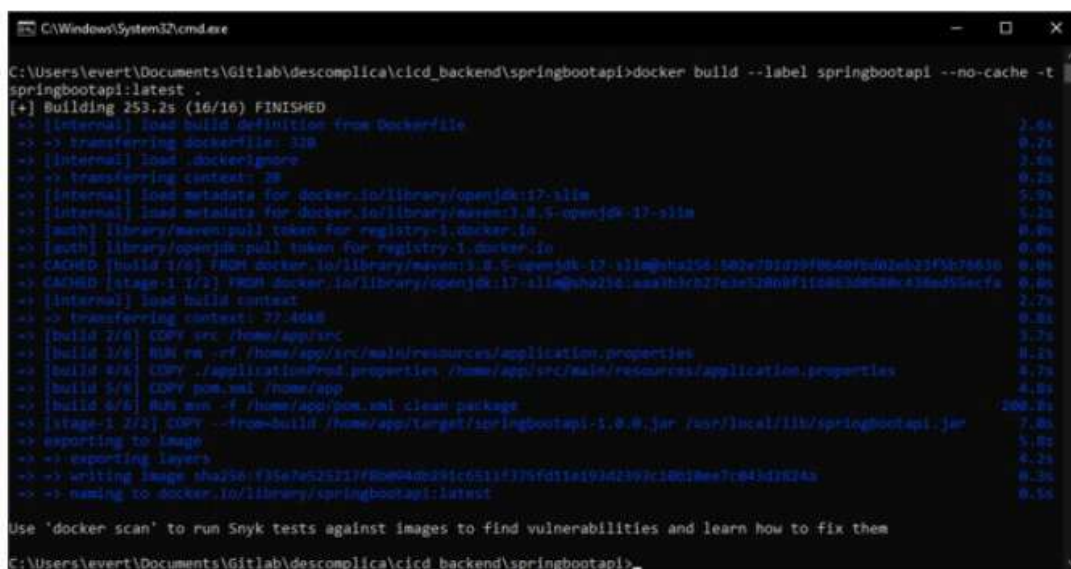
Resumidamente, o Dockerfile instrui o uso de uma imagem base do Java especificamente para efetuar a compilação do nosso código em java, neste caso a imagem é a “maven:3.8.5-openjdk-17-slim”. Como o projeto do Java utiliza o repositório do Maven para obter as bibliotecas necessárias para o projeto, usamos essa imagem base do Maven com Java para compilar o projeto. Após a compilação do projeto e geração do arquivo “.jar”, temos a parte final que usa uma imagem base “openjdk:17-slim” especificamente para executar nosso projeto compilado quando um container for criado à partir desta imagem. É exposta a porta 8080 como porta do container para receber as requisições, note que o comando para executar nossa aplicação quando o container é iniciado é o seguinte: “java -jar /usr/local/lib/springbootapi.jar”.

O detalhamento de cada linha deste Dockerfile se encontra no link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_backend/springbootapi

Podemos usar o seguinte comando para fazer o build da nossa aplicação API Java SpringBoot:

```
docker build --label springbootapi --no-cache -t springbootapi:latest .
```

Os detalhes de cada parâmetro estão nas instruções do link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_backend/springbootapi



```
C:\Windows\System32\cmd.exe
C:\Users\ever\Documents\Gitlab\descomplica\cicd_backend\springbootapi>docker build --label springbootapi --no-cache -t
springbootapi:latest .
[+] Building 253.2s (16/16) FINISHED
=> [internal] load build definition from Dockerfile                                2.0s
=> -- transferring dockerfile: 32B                                              0.2s
=> [internal] load .dockerignore                                                 0.0s
=> -- transferring context: 2B                                                  0.2s
=> [internal] load metadata for docker.io/library/openjdk:17-slim              5.0s
=> [internal] load metadata for docker.io/library/maven:3.8.5-openjdk-17-slim    1.2s
=> [auth] library/maven:pull token for registry-1.docker.io                   0.0s
=> [auth] library/openjdk:pull token for registry-1.docker.io                  0.0s
=> CACHED [build 1/6] FROM docker.io/library/maven:3.8.5-openjdk-17-slim@sha256:501e731239f0649fd02eb23f3b7663b  0.0s
=> CACHED [stage-1 1/2] FROM docker.io/library/openjdk:17-slim@sha256:aa5b3b627eae528b9f108c3d858ac33ed55ecfa  0.0s
=> [internal] load build context                                                2.7s
=> -- transferring context: 77.40kB                                             0.3s
=> [build 2/6] COPY src /home/app/src                                           3.7s
=> [build 3/6] RUN rm -rf /home/app/src/main/resources/application.properties    0.2s
=> [build 4/6] COPY ../applicationProd.properties /home/app/src/main/resources/application.properties  0.7s
=> [build 5/6] COPY pom.xml /home/app                                           4.0s
=> [build 6/6] RUN mvn -f /home/app/pom.xml clean package                      266.8s
=> [stage-1 2/2] COPY --from=build /home/app/target/springbootapi-1.0.0.jar /usr/local/lib/springbootapi.jar  7.8s
=> exporting to image                                                         5.0s
=> -- exporting layers                                                         4.2s
=> -- writing image sha256:f31e7e525717f80ee4ab261c6511f33f5fd1a19ad239c1801ee7c64bd824a    0.3s
=> -- naming to docker.io/library/springbootapi:latest                       0.5s
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
C:\Users\ever\Documents\Gitlab\descomplica\cicd_backend\springbootapi>
```

E por fim podemos criar um container com o seguinte comando:

```
docker run --name MeuSpringBootAPI --network MinhaRede -p 8080:8080 -d
springbootapi:latest
```

Os detalhes de cada parâmetro estão nas instruções do link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_backend/springbootapi

Agora vamos subir nossa segunda aplicação back-end, nosso API Python FastAPI que conecta ao banco de dados MySQL. As instruções completas estão no link: https://gitlab.com/everton.juniti/descomplica/-/tree/main/cicd_backend/pythonapi

Neste caso já há código pronto! Iremos fazer o build do projeto para criar uma imagem desta aplicação customizada e depois subir um container. Vamos olhar o Dockerfile desta solução em Python:

```
FROM python:3.6-slim

WORKDIR /code
COPY ./requirements.txt /code/requirements.txt

RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

COPY ./app /code/app
RUN rm -rf /code/app/data/databaseurl.py
COPY ./databaseurlprod.py /code/app/data/databaseurl.py

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

No caso do Python, não há necessidade de se compilar o código, então este Dockerfile apenas indica o uso da imagem base “python:3.6-slim” e demais códigos que trazem para dentro da imagem todas as bibliotecas usadas no projeto e seus códigos-fonte. É exposta a porta 8000 como porta do container para receber as requisições, note que o comando para executar nossa aplicação quando o container é iniciado é o seguinte: “uvicorn app.main:app --host 0.0.0.0 --port 8000”.

O detalhamento de cada linha deste Dockerfile se encontra no link: https://gitlab.com/everton.juniti/descomplica/-/tree/main/cicd_backend/pythonapi

Podemos usar o seguinte comando para fazer o build da nossa aplicação API Python FastAPI:

```
docker build --label pythonapi --no-cache -t pythonapi:latest .
```

Os detalhes de cada parâmetro estão nas instruções do link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_backend/pythonapi

```
C:\Windows\System32\cmd.exe
Microsoft Windows [versão 10.0.19044.1706]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\everton\Documents\Gitlab\descomplica\cicd_backend\pythonapi>docker build --label pythonapi --no-cache -t pythonapi:latest .
[+] Building 112.2s (13/13) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 193B
=> [internal] load .dockerignore
=> => transferring context: 128B
=> [internal] load metadata for docker.io/library/python:3.8-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load build context
=> => transferring context: 12.34kB
=> [1/7] FROM docker.io/library/python:3.8-slim@sha256:2cfabcf27958e8a55f78888884d91fe527498f9e32a724a6f9703b5f9b
=> CACHED [2/7] WORKDIR /code
=> [3/7] COPY ./requirements.txt /code/requirements.txt
=> [4/7] RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
=> [5/7] COPY ./app /code/app
=> [6/7] RUN rm -rf /code/app/data/databasaur1.py
=> [7/7] COPY ./databaseorigprod.py /code/app/data/databaseur1.py
=> exporting to image
=> => exporting layers
=> => writing image sha256:8eb22c8b997ee64e3121c8b3d5ee21be177edbee52292acfb920c3a308c8a
=> => naming to docker.io/library/pythonapi:latest

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

C:\Users\everton\Documents\Gitlab\descomplica\cicd_backend\pythonapi>
```

E por fim podemos criar um container com o seguinte comando:

```
docker run --name MeuPythonAPI --network MinhaRede -p 8000:8000 -d pythonapi:latest
```

Os detalhes de cada parâmetro estão nas instruções do link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_backend/pythonapi

E por último vamos dar uma olhada em nosso site feito com React, que conecta nas duas APIs que subimos, em Java e em Python! As instruções completas estão no link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_frontend/meusite

Vamos dar uma olhada no Dockerfile:

```

FROM node:slim
WORKDIR /app
ENV PATH /app/node_modules/.bin:$PATH
COPY ./src /app/src
COPY ./public /app/public
COPY package.json /app/package.json
RUN npm install
RUN npm install react-scripts@3.3.1 -g
EXPOSE 3000
CMD ["npm", "start"]

```

O detalhamento de cada linha deste Dockerfile se encontra no link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_frontend/meusite

Podemos usar o seguinte comando para fazer o build da nossa aplicação React:

`docker build --label reactsite --no-cache -t reactsite:latest .`

Os detalhes de cada parâmetro estão nas instruções do link: https://gitlab.com/evertton.juniti/descomplica/-/tree/main/cicd_frontend/meusite

```

C:\Windows\System32\cmd.exe
C:\Users\event\Documents\Gitlab\descomplica\cicd_frontend\meusite>docker build --label reactsite --no-cache -t reactsite:latest .
[+] Building 294.2s (13/13) FINISHED
-> [internal] load build definition from Dockerfile
-> => transferring dockerfile: 283B
-> [internal] load .dockerignore
-> => transferring context: 53B
-> [internal] load metadata for docker.io/library/node:slim
-> [auth] library/node:pull token for registry-1.docker.io
-> [2/7] FROM docker.io/library/node:slim@sha256:5acef02b02100a0d00a1c9e50a5e774ed7a8f858e10be4885b2c87dd40cf7
-> resolve docker.io/library/node:slim@sha256:5acef02b02100a0d00a1c9e50a5e774ed7a8f858e10be4885b2c87dd40cf7
-> sha256:5acef02b02100a0d00a1c9e50a5e774ed7a8f858e10be4885b2c87dd40cf7 1.21kB / 1.21kB
-> sha256:c0ff14cf40be13a23419aeaf23bd4b85267d405eb17549788174e0e78a534010 1.37kB / 1.37kB
-> sha256:ec8b32c6719e40034f009b3f481c540b4930425ed094e0c27cc180072a5500d 7.01kB / 7.01kB
-> sha256:ec18023b0f0a0e000b055f05e035b495fd0037b4082cc095a4ed0d0d00e75 45.73MB / 45.73MB
-> sha256:00a18190737daca180d03ff0fed04d40f3725fd75396adabb2efaeef032304e 2.70MB / 2.70MB
-> sha256:009c4b0d1d00255eb5d1c2f4cd82ee0fbb0a154953742877dd0c4520ea4fae 451B / 451B
-> extracting sha256:ec18023b0f0a0e000b055f05e035b495fd0037b4082cc095a4ed0d0d00e75
-> extracting sha256:00a18190737daca180d03ff0fed04d40f3725fd75396adabb2efaeef032304e
-> extracting sha256:009c4b0d1d00255eb5d1c2f4cd82ee0fbb0a154953742877dd0c4520ea4fae
-> [internal] load build context
-> => transferring context: 38.00kB
-> [3/7] WORKDIR /app
-> [3/7] COPY ./src /app/src
-> [4/7] COPY ./public /app/public
-> [5/7] COPY package.json /app/package.json
-> [6/7] RUN npm install
-> [7/7] RUN npm install react-scripts@3.3.1 -g
-> exporting to image
-> exporting layers
-> writing image sha256:3a9cf9a9b0b0c72c94d0e0c4b7b0c0011b02d0110e203434b02c2fd7a0f
-> naming to docker.io/library/reactsite:latest
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

```

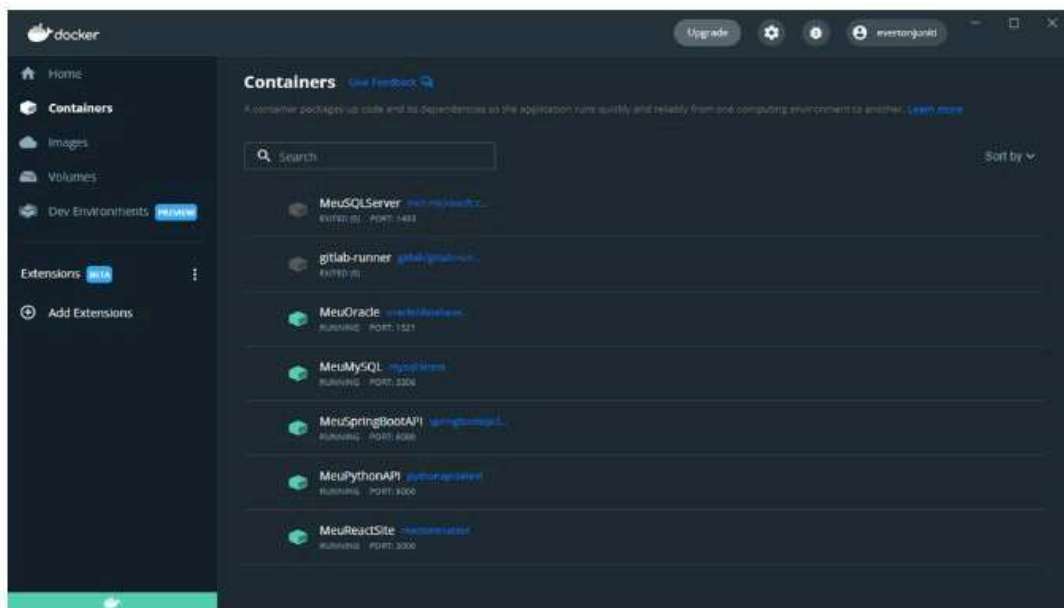
E por fim podemos criar um container com o seguinte comando:

```
docker run --name MeuReactSite --network MinhaRede -p 3000:3000 -d  
reactsite:latest
```

Os detalhes de cada parâmetro estão nas instruções do link: https://gitlab.com/evertongjuniti/descomplica/-/tree/main/cicd_frontend/meusite

Depois de tudo isso, podemos ver que todos os containers estão de pé: os bancos de dados Oracle (na porta 1521) e MySQL (na porta 3306), as APIs Java (na porta 8080) e Python (na porta 8000) e nosso site React (na porta 3000).

O site se comunica com as APIs e as APIs se comunicam com os bancos de dados!



Podemos usar o seguinte comando para verificar que todos esses containers estão na nossa “sub-rede” chamada “MinhaRede”:

```
docker network inspect MinhaRede
```

```
Prompt de Comando
C:\Users\event>docker network inspect MinhaRede
[
  {
    "Name": "MinhaRede",
    "Id": "c632ae39473a52f837df3ca0541b2bc6357f7e9e9d0b455e7abd61d026ba00fd",
    "Created": "2022-06-06T22:23:30.911906953Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "08cd1ff7efefda40c366f5ea0baa8b512f62142245f60fca264fbd9df609a82b": {
        "Name": "MeuMySQL",
        "EndpointID": "abc9ecec19591396122611f4dc0b45446618e29c1360157fdd00f688ca988087",
        "MacAddress": "02:42:ac:12:00:03",
        "IPv4Address": "172.18.0.3/16",
        "IPv6Address": ""
      },
      "68dc56005f5abfb4326b9389e9994c37129bc3c01a3d0409ae315d344668dcf3": {
        "Name": "MeuReactSite",
        "EndpointID": "dc34818c05e56511db63c2eeb0e27a1db5cc32d7309a7d65cee81f0ea1d887e3",
        "MacAddress": "02:42:ac:12:00:06",
        "IPv4Address": "172.18.0.6/16",
        "IPv6Address": ""
      }
    }
  }
]
```

```
Prompt de Comando
    },
    "91b8139063b848e839826bba7dc437319bcf46ca5563154dfbf78cf41a0d8817": {
      "Name": "MeuPythonAPI",
      "EndpointID": "bf55f27c8182ef44f8993e73c11a5616d37221f0831fb0af55fd234b2c428a93",
      "MacAddress": "02:42:ac:12:00:05",
      "IPv4Address": "172.18.0.5/16",
      "IPv6Address": ""
    },
    "dca3d07b37e9bb5af2f5f175dca066d6c0efefb7b5cab0f7d5193ced4271e1": {
      "Name": "MeuOracle",
      "EndpointID": "6c6094dd96abeab15bef9b31f949db54cc07506328faa2ed0e3989e51825c72",
      "MacAddress": "02:42:ac:12:00:02",
      "IPv4Address": "172.18.0.2/16",
      "IPv6Address": ""
    },
    "e5fa9d61df46e8ac71939546aa03d5e233414ecf37efc9150ec79c1e8b858d4d": {
      "Name": "MeuSpringBootAPI",
      "EndpointID": "dd98663243379fbf1f8cc4f882e0c763a1fd46346a6c4b66483ea295cef451e9",
      "MacAddress": "02:42:ac:12:00:04",
      "IPv4Address": "172.18.0.4/16",
      "IPv6Address": ""
    }
  },
  "Options": {},
  "Labels": {}
}
C:\Users\event>
```

Atividade Extra

Para se aprofundar no assunto desta aula leia no detalhe o conteúdo todo do repositório: “Descomplica” no GitLab, de Everton Juniti Ogura.

Link do repositório: <https://gitlab.com/everton.juniti/descomplica>

Referência Bibliográfica

- OGURA, Everton J. Repositório do GitLab, projeto Descomplica. Disponível em <https://gitlab.com/everton.juniti/descomplica>. Acesso em 06 de junho de 2022.

Ir para exercício