From Zero to Docker

Training | 2019.05.16 | Mário Dagot, Jorge Dias

Docker is an open platform for developing, shipping, and running applications. Through the course of this training we will guide you to the most common feature and use cases of docker. Take this as an introduction and an opportunity to dive into the docker world.

AGENDA

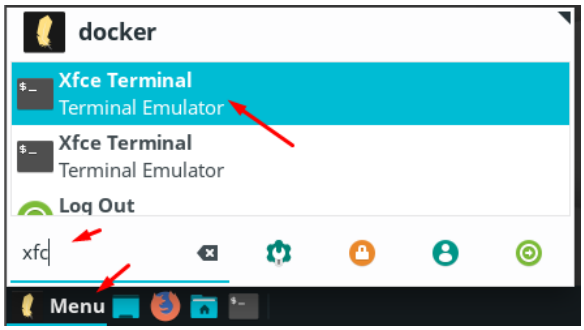
- 01 - Install Vim and Terminator and VSCode
- 02 - Install Docker CE for Ubuntu
- 03 - Hello from Busybox
- 04 - Webapp with Docker
- 05a - Webapp with Docker - My first Dockerfile – Nginx
- 05b.1 - Webapp with Docker - My first Dockerfile - Dotnet Core
- 05b.2 - Webapp with Docker - My first Dockerfile MultiStage - Dotnet Core
- 06 - Save and Restore and Push to Docker Hub
- **07a - Webapp with database integration - My first network – SpringBoot**
- 07b - Webapp with database integration - My first docker-compose – SpringBoot

07A - WEBAPP WITH DATABASE INTEGRATION - MY FIRST NETWORK – SPRINGBOOT

Objective

1. Pull a spring web app with database integration:
    a. https://github.com/spring-guides/gs-accessing-data-mysql.git
    b. https://spring.io/guides/gs/accessing-data-mysql/
2. Start the spring application using a docker container
3. Start the database engine using the docker-compose
4. Make the two containers communicate with each other

Step by Step

Open a terminal windows	
Clone the spring repository	docker ~ > workspace cd ~/workspace docker ~ > workspace git clone https://github.com/spring-guides/gs-accessing-data-mysql.git Cloning into 'gs-accessing-data-mysql'... remote: Enumerating objects: 41, done. remote: Counting objects: 100% (41/41), done. remote: Compressing objects: 100% (20/20), done. remote: Total 743 (delta 23), reused 38 (delta 21), pack-reused 702 Receiving objects: 100% (743/743), 314.42 KiB | 999.00 KiB/s, done. Resolving deltas: 100% (524/524), done. docker ~ cd ~/workspace/gs-accessing-data-mysql/complete/ docker ~ > workspace > gs-accessing-data-mysql > complete ls -l total 48 -rw-rw-r-- 1 docker docker 871 mai 9 15:55 build.gradle -rw-rw-r-- 1 docker docker 236 mai 9 15:55 docker-compose.yml drwxrwxr-x 3 docker docker 4096 mai 9 15:55 gradle -rwxrwxr-x 1 docker docker 5046 mai 9 15:55 gradlew -rw-rw-r-- 1 docker docker 2314 mai 9 15:55 gradlew.bat -rwxrwxr-x 1 docker docker 7059 mai 9 15:55 mvnw -rwxrwxr-x 1 docker docker 5007 mai 9 15:55 mvnw.cmd -rw-rw-r-- 1 docker docker 1776 mai 9 15:55 pom.xml drwxrwxr-x 3 docker docker 4096 mai 9 15:55 src
Build the java app using maven. With -v `pwd`:/app we are mounting our root folder on the /app folder of the container. With -v `pwd`/.m2:/root/.m2 we are mounting our root .m2	docker ~ > workspace > gs-accessing-data-mysql > complete docker run --rm -v `pwd`:/app -v `pwd`/.m2:/root/.m2 -w /app maven:latest mvn clean install [INFO] Scanning for projects... Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter-parent/2.1.4.RELEASE/spring-boot-starter-parent-2.1.4.RELEASE.pom Downloaded from central: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-

folder on /root/.m2 – this will allow us to save all the maven downloaded dependencies on our host. We can later on use it and avoid downloading them from the internet for each build. Remember, containers are ephemeral, if we don't save state (the maven dependencies) once the container is gone they will also be gone.	starter-parent/2.1.4.RELEASE/spring-boot-starter-parent-2.1.4.RELEASE.pom (6.6 kB at 4.8 kB/s) Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot- (…) Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.4/maven-shared-utils-0.4.jar (155 kB at 664 kB/s) [INFO] Installing /app/target/gs-mysql-data-0.1.0.jar to /root/.m2/repository/org/springframework/gs-mysql-data/0.1.0/gs-mysql-data-0.1.0.jar [INFO] Installing /app/pom.xml to /root/.m2/repository/org/springframework/gs-mysql-data/0.1.0/gs-mysql-data-0.1.0.pom [INFO] ---------- [INFO] BUILD SUCCESS [INFO] ---------- [INFO] Total time: 58.438 s [INFO] Finished at: 2019-05-11T14:40:51Z [INFO] ---------- docker > ~ > workspace > gs-accessing-data-mysql > complete
Let's start the java application The startup should have failed. Did we forget something? Looking at the output we can see database connectivity errors. Let's start the database first.	docker > … > workspace-solutions > gs-accessing-data-mysql-complete > complete > docker run --rm -v `pwd`:/app -v `pwd`/.m2:/root/.m2 -w /app maven:latest java -jar /app/target/gs-mysql-data-0.1.0.jar . ____ _ __ _ _ /\\ / ___'_ __ _ _(_)_ __ __ _ \ \ \ \ (( )___ | '_ | '_| | '_ \/ _` | \ \ \ \ \\/ ___)| |_)| | | | || (_| |)))) ' |____| .__|_| |_|_| |___, | / / / / =========|_|==========|___/=/_/_/_/ :: Spring Boot :: (v2.1.4.RELEASE) 2019-05-15 06:46:56.934 INFO 1 --- [main] hello.Application : Starting Application v0.1.0 on 0a561d56d5a5 with PID 1 (/app/target/gs-mysql-data-0.1.0.jar started by root in /app) 2019-05-15 06:46:56.962 INFO 1 --- [main] hello.Application : No active profile set, falling back to default profiles: default 2019-05-15 06:46:59.603 INFO 1 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data repositories in DEFAULT mode.

	(…) at org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator.initiateService(JdbcEnvironmentInitiator.java:137) ~[hibernate-core-5.3.9.Final.jar!/:5.3.9.Final] at org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator.initiateService(JdbcEnvironmentInitiator.java:35) ~[hibernate-core-5.3.9.Final.jar!/:5.3.9.Final] at org.hibernate.boot.registry.internal.StandardServiceRegistryImpl.initiateService(StandardServiceRegistryImpl.java:94) ~[hibernate-core-5.3.9.Final.jar!/:5.3.9.Final] at org.hibernate.service.internal.AbstractServiceRegistryImpl.createService(AbstractServiceRegistryImpl.java:263) ~[hibernate-core-5.3.9.Final.jar!/:5.3.9.Final] ... 41 common frames omitted
To start the database, the spring demo already provides a ready docker-compose file. We will discuss about docker-compose on another session.	docker ~ > workspace > gs-accessing-data-mysql > complete > cat docker-compose.yml mysql: image: mysql ports: - "3306:3306" environment: - MYSQL_USER=springuser - MYSQL_PASSWORD=ThePassword - MYSQL_DATABASE=db_example - MYSQL_ROOT_PASSWORD=root volumes: - "./conf.d:/etc/mysql/conf.d:ro"
Let's start the database If we do an ls -lrt we can see a fodler named conf.d. It stores the database state.	docker ~ > workspace > gs-accessing-data-mysql > complete > docker-compose up -d Recreating complete_mysql_1 ... done -rw-rw-r-- 1 docker docker 237 mai 15 07:52 docker-compose.ori.yml drwxrwxr-x 9 docker docker 4096 mai 15 07:52 . docker ~ > workspace > gs-accessing-data-mysql > complete > ls -lrt total 68 -rw-rw-r-- 1 docker docker 871 mai 9 14:30 build.gradle drwxrwxr-x 3 docker docker 4096 mai 9 14:30 src -rw-rw-r-- 1 docker docker 1776 mai 9 14:30 pom.xml

```
-rwxrwxr-x 1 docker docker 5007 mai  9 14:30 mvnw.cmd
-rwxrwxr-x 1 docker docker 7059 mai  9 14:30 mvnw
-rw-rw-r-- 1 docker docker 2314 mai  9 14:30 gradlew.bat
-rwxrwxr-x 1 docker docker 5046 mai  9 14:30 gradlew
drwxrwxr-x 3 docker docker 4096 mai  9 14:30 gradle
drwxr-xr-x 2 docker docker 4096 mai  9 14:36 conf.d
-rw-rw-r-- 1 docker docker  561 mai 11 22:08 docker-compose.yml
drwxr-xr-x 6 root   root   4096 mai 15 07:42 target
```

But we can do better and create our Dockerfile to bundle the code, build it and produce a runnable container all-in-one.

Did you notice the COPY of .m2 to /root/.m2?

Just a trick to speed up the building process. Instead of, for each build have maven download all the libraries we use the already dependencies we cached previously and provide them in the docker file.

Try removing that COPY line from your docker file and build the image again. Compare the output. Maven is very verbose. It's easy to notice that there's more lots of things being downloads and the build time increased.

```
docker  ~ > workspace > gs-accessing-data-mysql > complete  vi Dockerfile

docker  ~ > workspace > gs-accessing-data-mysql > complete  cat Dockerfile
FROM maven:latest AS build
WORKDIR /app
COPY .m2 /root/.m2
COPY . /app
RUN mvn package

FROM openjdk:8-jre AS runtime
WORKDIR /app
COPY --from=build /app/target/*.jar /app
ENTRYPOINT ["java", "-jar", "/app/gs-mysql-data-0.1.0.jar"]


docker  ~ > workspace > gs-accessing-data-mysql > complete  docker build --
tag myjavaapp .
Sending build context to Docker daemon  137.4MB
Step 1/9 : FROM maven:latest AS build
 ---> cafa0008b735
Step 2/9 : WORKDIR /app
 ---> Running in 906077d11222
Removing intermediate container 906077d11222
 ---> 4d46ccb63799
Step 3/9 : COPY .m2 /root/.m2
 ---> 1bfff1cfc7a0
Step 4/9 : COPY . /app
 ---> 25f2c0ef46fa
```

```
Step 5/9 : RUN mvn package
 ---> Running in c335d7c8dfdc
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------< org.springframework:gs-mysql-data >------------------
[INFO] Building gs-mysql-data 0.1.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ gs-
mysql-data ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ gs-mysql-data
---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:testResources (default-testResources) @
gs-mysql-data ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /app/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ gs-
mysql-data ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ gs-mysql-data ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:3.1.1:jar (default-jar) @ gs-mysql-data ---
[INFO] Building jar: /app/target/gs-mysql-data-0.1.0.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.1.4.RELEASE:repackage (repackage) @ gs-
mysql-data ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

	`[INFO] Total time:  4.841 s`
	`[INFO] Finished at: 2019-05-11T14:45:46Z`
	`[INFO] ------------------------------------------------------------------------`
	`Removing intermediate container c335d7c8dfdc`
	` ---> 97b3e4387f11`
	`Step 6/9 : FROM openjdk:8-jre AS runtime`
	` ---> b5ee13f1fc07`
	`Step 7/9 : WORKDIR /app`
	` ---> Running in 361db0e2586d`
	`Removing intermediate container 361db0e2586d`
	` ---> cc5112251f44`
	`Step 8/9 : COPY --from=build /app/target/*.jar /app`
	` ---> c6e7c70d8023`
	`Step 9/9 : ENTRYPOINT ["java", "-jar", "/app/gs-mysql-data-0.1.0.jar"]`
	` ---> Running in b9313e482e46`
	`Removing intermediate container b9313e482e46`
	` ---> 6ddda53b19fe`
	`Successfully built 6ddda53b19fe`
	`Successfully tagged myjavaapp:latest`

Let's now run our newly created webapp.
 	
Look at the containers. Database and Webapp running.
 	
All is set.
	`docker ~ > workspace > gs-accessing-data-mysql > complete > docker run -d -p 8080:8080 myjavaapp`
	`da26541922789112523fd3d9fd36ad7d211b8a8f26c2d6164869bfe8bcf88dde`
	`docker ~ > workspace > gs-accessing-data-mysql > complete > docker ps`
	`CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES`
	`da2654192278  myjavaapp  "java -jar /app/gs-m…"  4 seconds ago  Up 3 seconds  0.0.0.0:8080->8080/tcp  priceless_chebyshev`
	`aaa3a48ae1d1  mysql  "docker-entrypoint.s…"  2 minutes ago  Up 2 minutes  0.0.0.0:3306->3306/tcp, 33060/tcp  complete_mysql_1`

Let's just do an HTTP request to the end point and see the result. An error?! What happened?!	`docker ~ > workspace > gs-accessing-data-mysql > complete > curl http://localhost:8080/demo/all` `curl: (7) Failed to connect to localhost port 8080: Connection refused`
Let's check the logs, maybe there's a clue.	`docker ~ > workspace > gs-accessing-data-mysql > complete > docker logs priceless_chebyshev`

The webapp is not able to communicate with the database.

Is it down?

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.4.RELEASE)


2019-05-09 17:26:12.735  INFO 1 --- [           main] hello.Application
: Starting Application v0.1.0 on da2654192278 with PID 1 (/app/gs-mysql-data-
0.1.0.jar started by root in /app)
2019-05-09 17:26:12.739  INFO 1 --- [           main] hello.Application
: No active profile set, falling back to default profiles: default
2019-05-09 17:26:15.220  INFO 1 --- [           main]
.s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data
repositories in DEFAULT mode.
2019-05-09 17:26:15.348  INFO 1 --- [           main]
.s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository
scanning in 105ms. Found 1 repository interfaces.
2019-05-09 17:26:16.285  INFO 1 --- [           main]
trationDelegate$BeanPostProcessorChecker : Bean
'org.springframework.transaction.annotation.
ProxyTransactionManagementConfiguration' of type
[org.springframework.transaction.annotation.
ProxyTransactionManagementConfiguration$$EnhancerBySpringCGLIB$$88d37e78] is not
eligible for getting processed by all BeanPostProcessors (for example: not
eligible for auto-proxying)
2019-05-09 17:26:17.176  INFO 1 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080
(http)
2019-05-09 17:26:17.264  INFO 1 --- [           main]
o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2019-05-09 17:26:17.265  INFO 1 --- [           main]
org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache
Tomcat/9.0.17]
2019-05-09 17:26:17.456  INFO 1 --- [           main]
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded
WebApplicationContext
2019-05-09 17:26:17.457  INFO 1 --- [           main]
o.s.web.context.ContextLoader            : Root WebApplicationContext:
initialization completed in 4567 ms
2019-05-09 17:26:17.982  INFO 1 --- [           main]
com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
```

	2019-05-09 17:26:19.270 ERROR 1 --- [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Exception during pool initialization. com.mysql.cj.jdbc.exceptions.CommunicationsException: Communications link failure The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server. (…) 2019-05-09 17:26:19.284 WARN 1 --- [main] o.s.b.a.orm.jpa.DatabaseLookup : Unable to determine jdbc url from datasource org.springframework.jdbc.support.MetaDataAccessException: Could not get Connection for extracting meta-data; nested exception is org.springframework.jdbc.CannotGetJdbcConnectionException: Failed to obtain JDBC Connection; nested exception is com.mysql.cj.jdbc.exceptions.CommunicationsException: Communications link failure The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server. (…)
Let's troubleshoot the mysql container. We can see that the container is running and listening to port 33060.	docker ~ > workspace > gs-accessing-data-mysql > complete docker logs complete_mysql_1 --tail 5 2019-05-11T14:55:24.999766Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.16) starting as process 1 2019-05-11T14:55:25.499334Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed. 2019-05-11T14:55:25.501365Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory. 2019-05-11T14:55:25.529046Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.16' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL. 2019-05-11T14:55:25.627992Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Socket: '/var/run/mysqld/mysqlx.sock' bind-address: '::' port: 33060

Look at the containers running and ports being exposed.

All is good, from the host's point of view.

Note that, exposing ports are just a way to make the host communicate with the container. Containers are isolated from each other.

If we need containers to communicate with each other we need to make them part of the same docker network.

```
docker  ~ > workspace > gs-accessing-data-mysql > complete  docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                                NAMES
39eb9860f056        mysql               "docker-entrypoint.s…"   2 minutes ago
Up 2 minutes        0.0.0.0:3306->3306/tcp, 33060/tcp   complete_mysql_1
159f37d89f40        mywebsite           "nginx -g 'daemon of…"   About an hour
ago   Up About an hour    0.0.0.0:8080->80/tcp                mywebsite
```

Let's list networks and create a new one to make the two services talk to each other. It also serves to isolate containers from one another – just because they are containers doesn't mean all containers should talk with each other – security must be taken into account at all times.

```
docker  ~ > workspace > gs-accessing-data-mysql > complete  docker network
ls
NETWORK ID          NAME                DRIVER              SCOPE
a1e787aca383        bridge              bridge              local
3100162171e4        host                host                local
b5e9ed62461f        none                null                local
docker  ~ > workspace > gs-accessing-data-mysql > complete  docker network
create myjavaapp_network
d322d01295344931f4a54a7053d3bfabe26feeb555f9ae7196b33efc2dab915a
docker  ~ > workspace > gs-accessing-data-mysql > complete  docker inspect
myjavaapp_network
[
    {
        "Name": "myjavaapp_network",
        "Id":
"d322d01295344931f4a54a7053d3bfabe26feeb555f9ae7196b33efc2dab915a",
        "Created": "2019-05-11T16:00:18.210888632+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
```



```
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Let's join the MySQL container to our new network.

Never mind the docker-compose details. We will get to them later.

Important now:
MySQL will be the name the hostname of the container on the network
We said that it will use an external network – a network not managed by the docker-compose

```
docker > ~ > workspace > gs-accessing-data-mysql > complete > vi docker-compose-with-network.yml
docker > ~ > workspace > gs-accessing-data-mysql > complete > cat docker-compose-with-network.yml
version: '3.2'
services:
 mysql:
  image: mysql
  ports:
    - "3306:3306"
  environment:
    - MYSQL_USER=springuser
    - MYSQL_PASSWORD=ThePassword
    - MYSQL_DATABASE=db_example
    - MYSQL_ROOT_PASSWORD=root
  volumes:
    - "./conf.d:/etc/mysql/conf.d:ro"
networks:
  default:
    external:
      name: myjavaapp_network
```


	docker ~ > workspace > gs-accessing-data-mysql > complete > docker-compose -f docker-compose-with-network.yml up -d Creating complete_mysql_1 ... done docker ~ > workspace > gs-accessing-data-mysql > complete > docker-compose -f docker-compose-with-network.yml ps Name Command State Ports ---------- complete_mysql_1 docker-entrypoint.sh mysqld Up 0.0.0.0:3306->3306/tcp, 33060/tcp
Let's fix the hostname the webapp will use to connect to the database. Standard stuff on a spring boot app. We can then running container and try again.	docker ~ > workspace > gs-accessing-data-mysql > complete > cat src/main/resources/application.properties spring.jpa.hibernate.ddl-auto=create spring.datasource.url=jdbc:mysql://localhost:3306/db_example spring.datasource.username=springuser spring.datasource.password=ThePassword docker ~ > workspace > gs-accessing-data-mysql > complete > docker ps CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES aaa3a48ae1d1 mysql "docker-entrypoint.s…" 4 minutes ago Up 4 minutes 0.0.0.0:3306->3306/tcp, 33060/tcp complete_mysql_1 docker ~ > workspace > gs-accessing-data-mysql > complete > docker run --rm -p 8080:8080 myjavaapp 3534d2db98461b418be95116fa41db713c1e1a9a46253f51474fca9335a32e71
Run the myjavaapp docker image. This time use the --net to specify the network he should connect to.	docker ~ > workspace > gs-accessing-data-mysql > complete > docker run --rm -d -p 8080:8080 --net myjavaapp_network myjavaapp 82c6e48523c48c2df8f5aec4403ccebd121bb45b27151cd3bba13d3c9ed0b60e
Let's play around. Add some users and check if they were saved. You can also stop the containers and start them again. The information is now on the database and should not be lost.	docker ~ > workspace > gs-accessing-data-mysql > complete > curl http://localhost:8080/demo/all [] docker ~ > workspace > gs-accessing-data-mysql > complete > curl 'http://localhost:8080/demo/add?name=Mario&email=mario@gmail.com' Saved docker ~ > workspace > gs-accessing-data-mysql > complete > curl 'http://localhost:8080/demo/add?name=Mario2&email=mario2@gmail.com'

	docker ~ > workspace > gs-accessing-data-mysql > complete curl http://localhost:8080/demo/all [{"id":1,"name":"Mario","email":"mario@gmail.com"},{"id":2,"name":"Mario2", "email":"mario2@gmail.com"}]
Let's shut everything down	docker ~ > workspace > gs-accessing-data-mysql-complete > complete docker ps -a CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 30dc29dcbf60 myjavaapp "java -jar /app/gs-m…" 17 minutes ago Up 17 minutes 0.0.0.0:8080->8080/tcp compassionate_tesla ad0fe9337a8b mysql "docker-entrypoint.s…" 19 minutes ago Up 19 minutes 0.0.0.0:3306->3306/tcp, 33060/tcp complete_mysql_1 (failed reverse-i-search)`': docker container rm -f har^Core_wiles docker ~ > workspace > gs-accessing-data-mysql > complete docker-compose -f docker-compose-with-network.yml down Stopping complete_mysql_1 ... done Removing complete_mysql_1 ... done Network myjavaapp_network is external, skipping docker ~ > workspace > gs-accessing-data-mysql > complete docker container stop compassionate_tesla compassionate_tesla docker ~ > workspace > gs-accessing-data-mysql > complete docker ps -a CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES docker ~ > workspace > gs-accessing-data-mysql > complete

Lessons learned

Using the very useful spring training and sources – How to create a web app with database integration – from here:

a. https://github.com/spring-guides/gs-accessing-data-mysql.git
b. https://spring.io/guides/gs/accessing-data-mysql/

We learned how to build and run a spring boot application with java and maven without having them installed on our system. All using a pre made maven image (which already contained java and maven).

We learned how to start a docker-compose application. For this particular example it was used to spin up MySQL container. State was also saved locally on the host to avoid losing state.

We learned how to list and create docker networks. That allowed our two containers to communicate with each other.

Revision History

Version	Date	Author	Description
1.0	2019.05.01	Mário Dagot, Jorge Dias	Initial Version