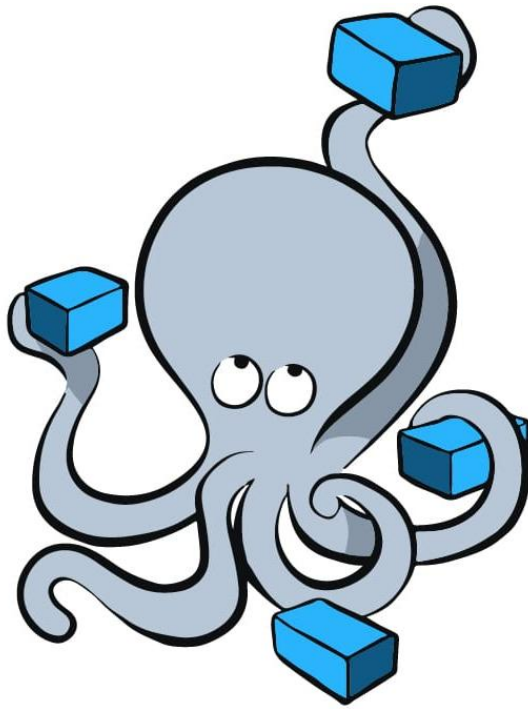


Docker Compose et Volume



docker

Compose et Volume

Bureau E204

Plan du cours

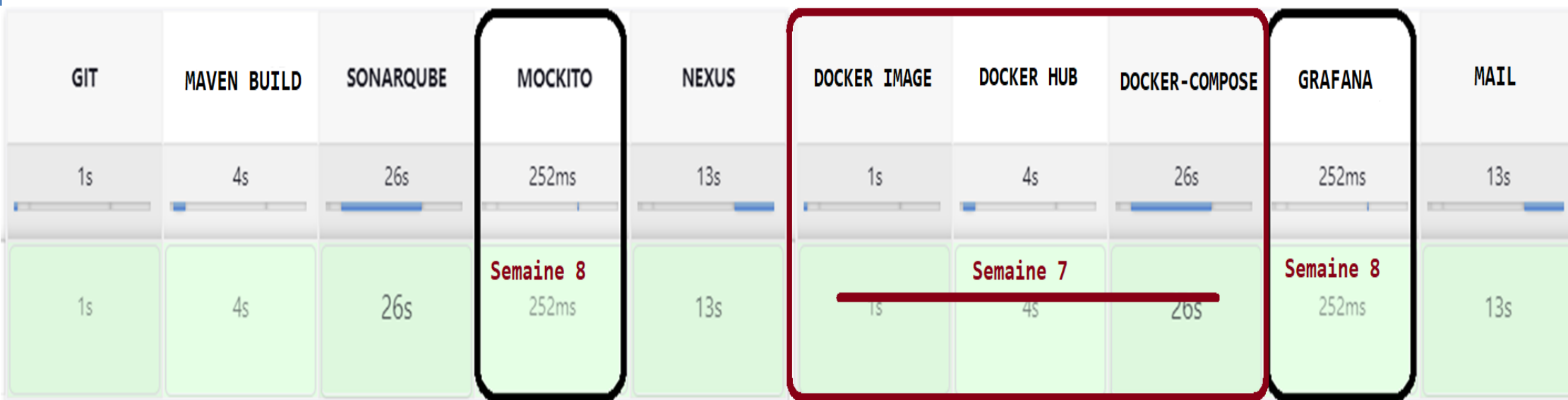
- Introduction
- Docker
- Docker Compose
- Docker Volume
- Docker Compose et Jenkins

Introduction

- Notre application Spring Boot codée, compilée et testée (unitairement et qualitativement) doit être intégrée dans une chaîne DevOps complète (CI/CD).
- La chaîne d'intégration continue (CI) a été réalisée grâce à Jenkins via la création d'un pipeline.
- Dans ce cours on va s'intéresser **à la chaîne CD (Continuos delivery and deployment)**

Introduction

Projet DevOps Final :



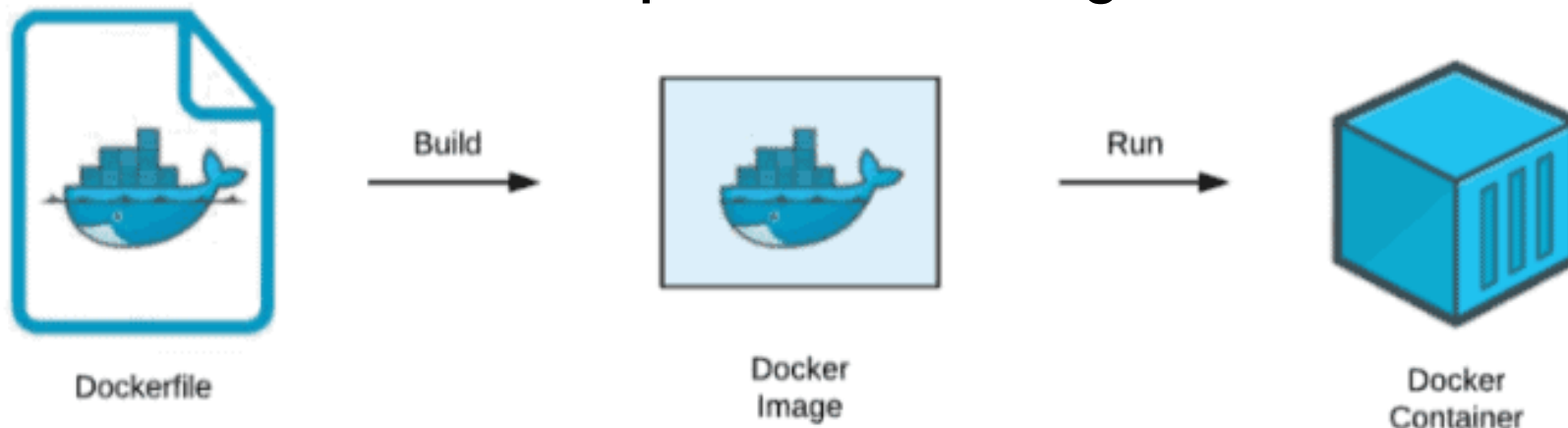
Introduction

- L'objectif de la partie CD (déploiement et livraison continu) est de placer notre application dans un environnement donné : **UAT (User Acceptance Tests), Qualification, Pré-Production, Production** et de la surveiller.
- Ces environnements peuvent être :
 - ✓ Une machine physique
 - ✓ Une machine virtuelle
 - ✓ Un conteneur Docker

Introduction

Nous avons vu que nous pouvons isoler chaque application à l'intérieur d'une image où nous pouvons définir son environnement dans un Dockerfile. Puis, avec un simple “docker build” et “docker run”, notre application sera accessible via le port que nous avons exposé:

- **docker build -t <image_name> .**
- **docker run -p 8080:8080 <image_name>**

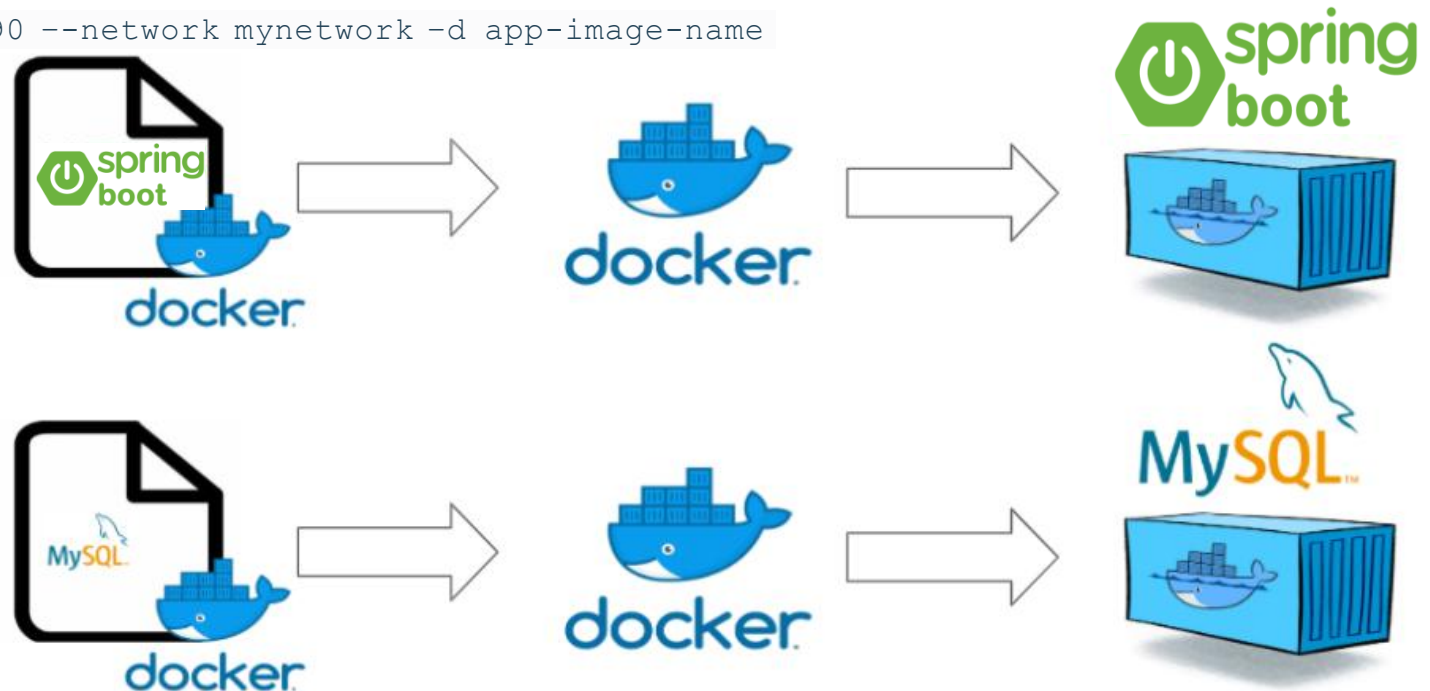


Introduction

L'application a besoin de se connecter à un serveur base de données.

→ Pour que ces deux-là puissent communiquer ensemble, il faut les mettre sous le même réseau et lancer la base de données avant le démarrage de l'application.

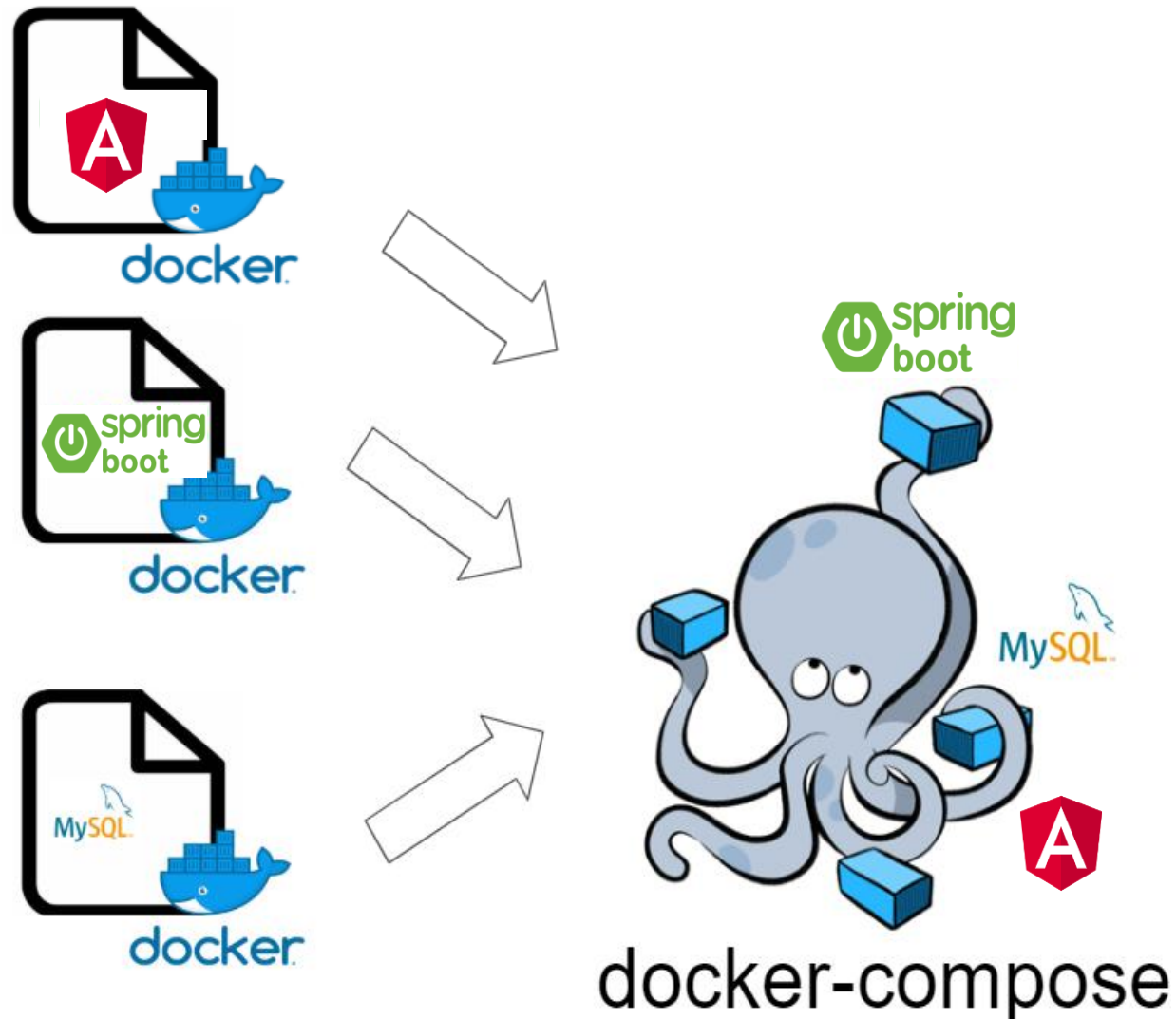
```
docker run -p 9090:9090 --network mynetwork -d app-image-name
```



```
docker run --name mysqldb --network mynetwork -e MYSQL_ROOT_PASSWORD=my-secret-pw -v /home/mysql/data:/var/lib/mysql -d mysql:8
```

Introduction

→ Et là, il nous faut docker compose.



Docker Compose

- **Docker Compose est un outil permettant de définir et d'exécuter des applications Docker multi-conteneurs.**
- Dans cette logique, chaque partie de l'application (code, base de données, serveur web, ...) sera hébergée par un conteneur.
- Cet outil repose sur le langage YAML pour décrire l'architecture physique de l'application. YAML est utilisé pour coder les fichiers de configuration.
- Le fichier Docker-Compose comporte la **version**, les **services** (REQUIS), les **réseaux**, les **volumes**, les **configurations** et les **secrets**.
- Après la configuration du fichier YAML, une seule commande à exécuter pour créer et démarrer tous les services.

Docker Compose

- L'utilisation de Docker Compose se résume à un processus en trois étapes :
 1. Définir l'environnement de votre application à l'aide d'un « **Dockerfile** » afin qu'il puisse être reproduit partout.
 2. Définir les services qui composent votre application dans « **docker-compose.yml** » afin qu'ils puissent être exécutés ensemble dans un environnement isolé.
 3. Exécuter la commande « **docker compose up** », c'est la commande pour lancer votre application entière.

Docker Compose - Example

timesheet-devops C:\Work\workspace

> .idea

> .mvn

> .settings

src

main

java

resources

static

templates

application.properties

test

target

.classpath

.gitignore

.project

docker-compose.yml

Dockerfile

HELP.md

mvnw

mvnw.cmd

pom.xml

External Libraries

Scratches and Consoles

```
1 version: "3.8"
2
3 >> services:
4 > mysql:
5     image: mysql:5.7
6     restart: unless-stopped
7     environment:
8         - MYSQL_ROOT_PASSWORD=
9         - MYSQL_DATABASE=timesheet_db
10    ports:
11        - 3306:3306
12    volumes:
13        - db:/var/lib/mysql
14 > app-timesheet:
15     depends_on:
16         - mysql
17     image: mouradhassini/timesheet-devops:1.0.0
18     restart: on-failure
19     ports:
20         - 8082:8082
21     environment:
22         SPRING_APPLICATION_JSON: '{
23             "spring.datasource.url" : "jdbc:mysql://mysql:3306/timesheet_db?createDatabaseIfNotExist=true",
24             "spring.datasource.username" : "root",
25             "spring.datasource.password" : null,
26             "spring.jpa.properties.hibernate.dialect" : "org.hibernate.dialect.MySQL5InnoDBDialect",
27             "spring.jpa.hibernate.ddl-auto" : "update"
28         }'
29     stdin_open: true
30     tty: true
31
32 volumes:
33     db :
```

Installation Docker Compose

Docker compose est normalement déjà installé, vérifier avec la commande :

`docker compose version`

Les 3 fonctions principales

Les 3 fonctions principales de docker-compose sont :

- Comment lancer un docker-compose? (se mettre dans le dossier contenant le fichier docker-compose.yml) :
docker compose up -d
- Comment vérifier les logs des conteneurs qui ont été lancé?
docker compose logs
- Comment arrêter un docker compose ?
docker compose down

Exemple

Cet exemple montre comment lancer deux conteneurs (Nexus et Sonar) en utilisant Docker-Compose.

Ce TP n'est pas à faire. Cela va surcharger votre VM avec 2 autres conteneurs.

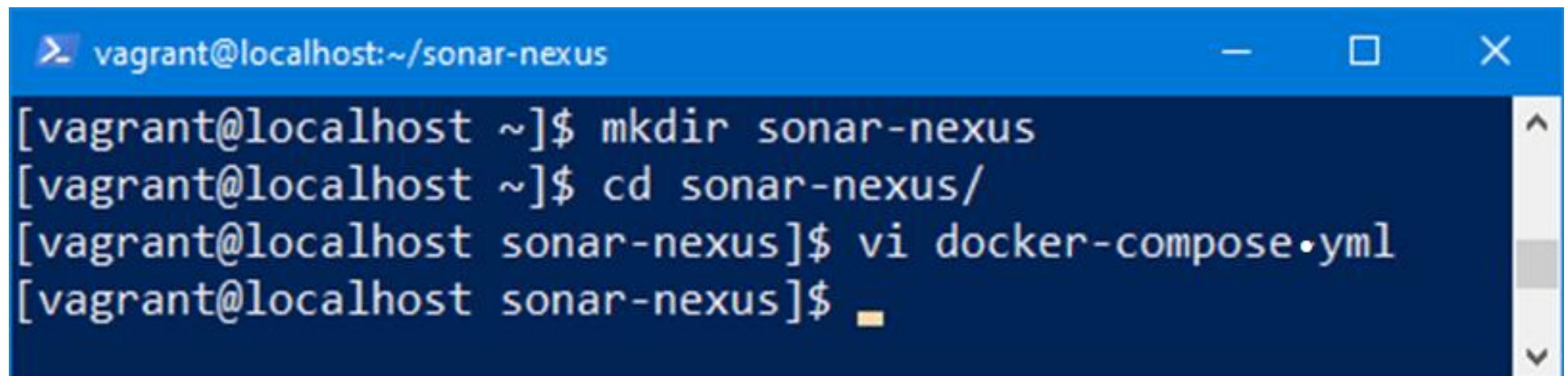
C'est juste pour vous expliquer Docker-Compose.

Vous allez vous en inspirer pour créer votre propre fichier Docker-Compose, par la suite, qui fera tourner votre application Spring boot (Conteneur 1) et une base de données MySQL (Conteneur 2).

Solution ci-dessous :

Docker Compose - Exécuter des conteneurs ensemble

- Pour utiliser « Docker-compose », nous allons configurer les deux images 'Sonarqube' et 'Nexus' afin de les lancer simultanément.
- J'ai créé un dossier nommé «sonar-nexus» et ensuite je suis allé dans ce répertoire
- J'ai créé le fichier YAML en utilisant l'éditeur de texte vi :

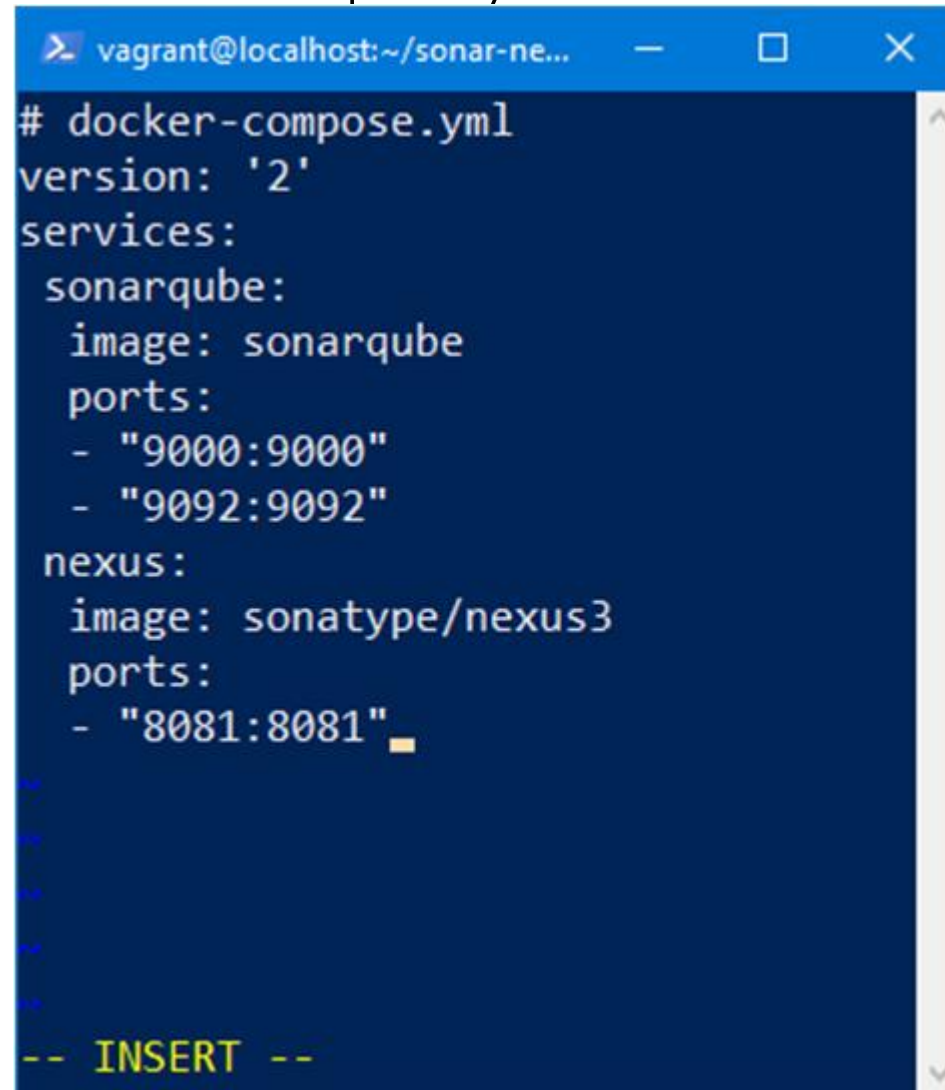


```
vagrant@localhost:~/sonar-nexus

[vagrant@localhost ~]$ mkdir sonar-nexus
[vagrant@localhost ~]$ cd sonar-nexus/
[vagrant@localhost sonar-nexus]$ vi docker-compose.yml
[vagrant@localhost sonar-nexus]$
```

Docker Compose - Exécuter des conteneurs ensemble

- Contenu du fichier « docker-compose.yml »:

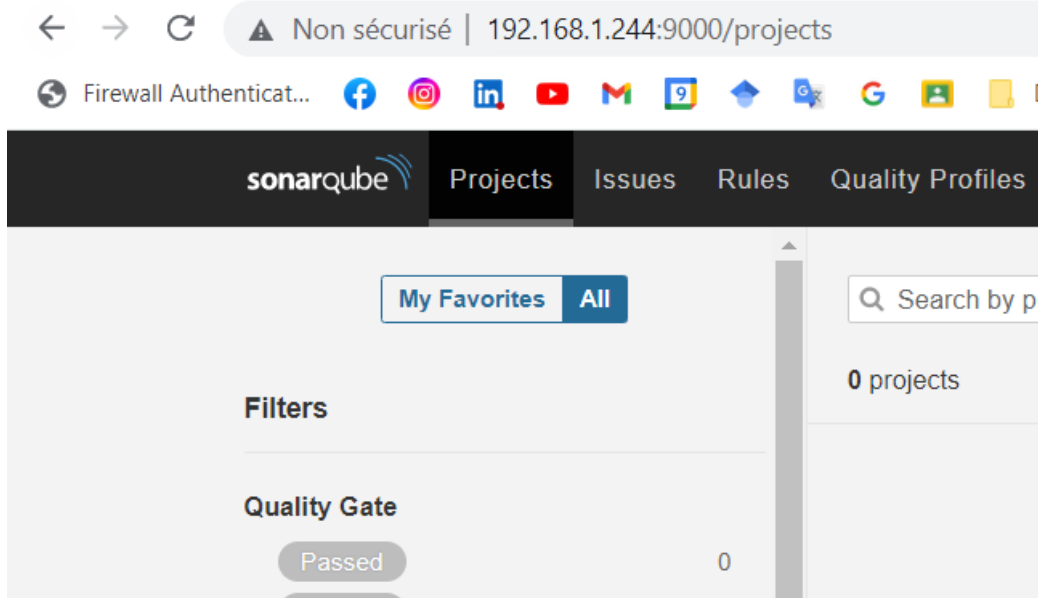
A screenshot of a terminal window with a blue title bar. The title bar text is 'vagrant@localhost: ~/sonar-ne...'. The terminal content shows a YAML configuration for Docker Compose. It starts with a comment '# docker-compose.yml', followed by 'version: '2'', and a 'services:' section. Under 'services:', there are two services: 'sonarqube' and 'nexus'. 'sonarqube' has 'image: sonarqube' and 'ports:' with two entries: '- "9000:9000"' and '- "9092:9092"'. 'nexus' has 'image: sonatype/nexus3' and 'ports:' with one entry: '- "8081:8081"'. At the bottom, there is a yellow text prompt '-- INSERT --'.

```
vagrant@localhost: ~/sonar-ne...  
# docker-compose.yml  
version: '2'  
services:  
  sonarqube:  
    image: sonarqube  
    ports:  
      - "9000:9000"  
      - "9092:9092"  
  nexus:  
    image: sonatype/nexus3  
    ports:  
      - "8081:8081"  
-- INSERT --
```

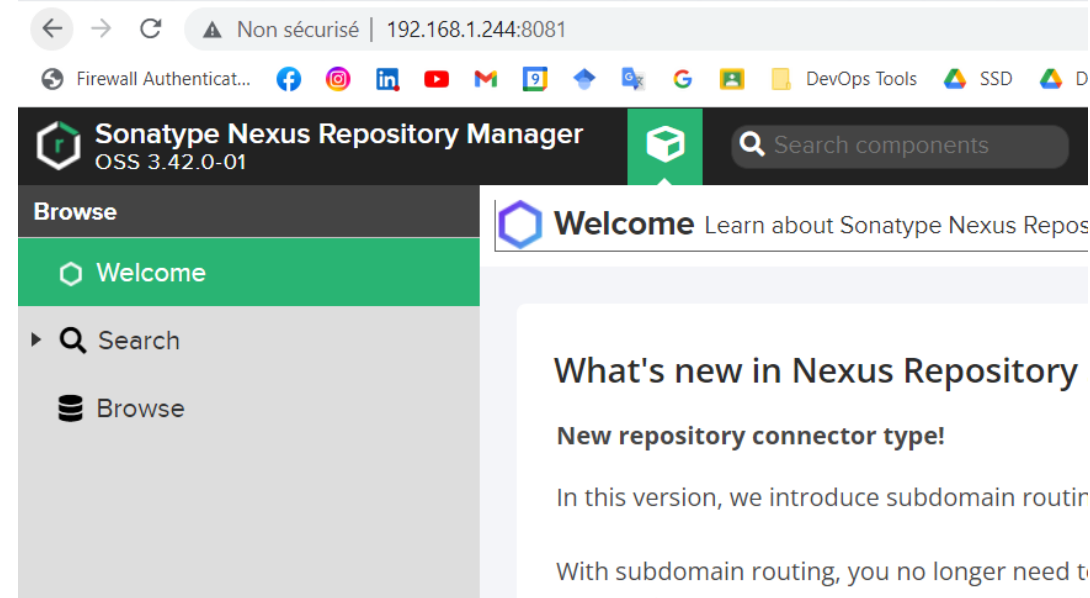

Docker Compose - Exécuter des conteneurs ensemble

- La commande suivante pour créer les conteneurs

```
Sélection vagrant@localhost:~/sonar-nexus  
[vagrant@localhost sonar-nexus]$ docker-compose up  
Creating network "sonar-nexus_default" with the default driver  
Creating sonar-nexus_nexus_1 ... done  
Creating sonar-nexus_sonarqube_1 ... done  
Attaching to sonar-nexus_sonarqube_1, sonar-nexus_nexus_1  
sonarqube_1 | 2022.10.10 23:11:50 INFO app[][o.s.a.AppFileSystem] Cleaning or creating temp directory /opt/sonarqube  
/temp  
sonarqube_1 | 2022.10.10 23:11:50 INFO app[][o.s.a.es.EsSettings] Elasticsearch listening on [HTTP: 127.0.0.1:9001,  
TCP: 127.0.0.1:34250]
```



SonarQube



Nexus

Docker Compose - Exécuter des conteneurs ensemble

- Comment vérifier les logs des conteneurs qui ont été lancés ?

docker compose logs

```
[root@localhost SonarAndNexus]# docker-compose logs
Attaching to sonarandnexus_nexus_1, sonarandnexus_sonarqube_1
sonarqube_1 | 2022.10.09 21:19:48 INFO app[][o.s.a.AppFileSystem] Cleaning or cre
sonarqube_1 | 2022.10.09 21:19:48 INFO app[][o.s.a.es.EsSettings] Elasticsearch
sonarqube_1 | 2022.10.09 21:19:48 INFO app[][o.s.a.ProcessLauncherImpl] Launch p
elasticsearch]: /opt/sonarqube/elasticsearch/bin/elasticsearch
sonarqube_1 | 2022.10.09 21:19:48 INFO app[][o.s.a.SchedulerImpl] Waiting for Ela
sonarqube_1 | warning: no-jdk distributions that do not bundle a JDK are deprecate
sonarqube_1 | 2022.10.09 21:19:58 INFO es[][o.e.n.Node] version[7.16.2], pid[40]
:42:46.604893745Z], OS[Linux/3.10.0-1160.76.1.el7.x86_64/amd64], JVM[Eclipse Adopt
sonarqube_1 | 2022.10.09 21:19:58 INFO es[][o.e.n.Node] JVM home [/opt/java/openj
nexus_1 | 2022-10-09 21:20:09,885+0000 INFO [FelixStartLevel] *SYSTEM org.son
nexus_1 | 2022-10-09 21:20:11,535+0000 WARN [CM Event Dispatcher (Fire Config
s not writeable: file:/opt/sonatype/nexus/etc/karaf/jmx.acl.cfg
nexus_1 | 2022-10-09 21:20:12,130+0000 WARN [CM Event Dispatcher (Fire Config
tall - File is not writeable: file:/opt/sonatype/nexus/etc/karaf/org.apache.karaf.
nexus_1 | 2022-10-09 21:20:12,142+0000 WARN [CM Event Dispatcher (Fire Config
```

Docker Compose - Exécuter des conteneurs ensemble

- Comment arrêter un docker compose ?

docker compose down

```
[root@localhost SonarAndNexus]# docker-compose down
Stopping sonarandnexus_nexus_1      ... done
Stopping sonarandnexus_sonarqube_1  ... done
Removing sonarandnexus_nexus_1      ... done
Removing sonarandnexus_sonarqube_1  ... done
Removing network sonarandnexus_default
```

Docker Compose - Exécuter des conteneurs ensemble

- Une fois que nous arrêtons l'exécution du « Docker-compose » et nous le démarrons une autre fois, nous devons **refaire** la configuration.
- En fait, la configuration est stockée dans le conteneur. Mais, si nous le supprimons, nous supprimons aussi les données de configuration.

Comment palier à ce problème ?

→ Docker Volume.

Docker Volume

- Les volumes sont le mécanisme privilégié pour la persistance des données générées et utilisées par les conteneurs Docker.
- Les volumes permettent de garder en mémoire des données de manière permanente.
- Le volume est une fonctionnalité très intéressante dans Docker. Il rend l'utilisation des conteneurs encore plus attrayante.
- Avec des volumes bien configurés, il est possible de réutiliser certaines données dans un autre conteneur, de les exporter ailleurs ou de les importer.

Docker Volume – Configuration dans Docker-Compose

```
# docker-compose.yml
version: '2'
services:
  sonarqube:
    image: sonarqube:8.9.7-community
    ports:
      - "9000:9000"
      - "9092:9092"
    volumes:
      - 'SonarQube_data:/opt/SonarQube/data'
      - 'SonarQube_extensions:/opt/SonarQube/extensions'
      - 'SonarQube_logs:/opt/SonarQube/logs'
  nexus:
    image: sonatype/nexus3
    ports:
      - "8081:8081"
    volumes:
      - 'nexus-data:/nexus-data'
volumes:
  SonarQube_data:
  SonarQube_extensions:
  SonarQube_logs:
  nexus-data:
```

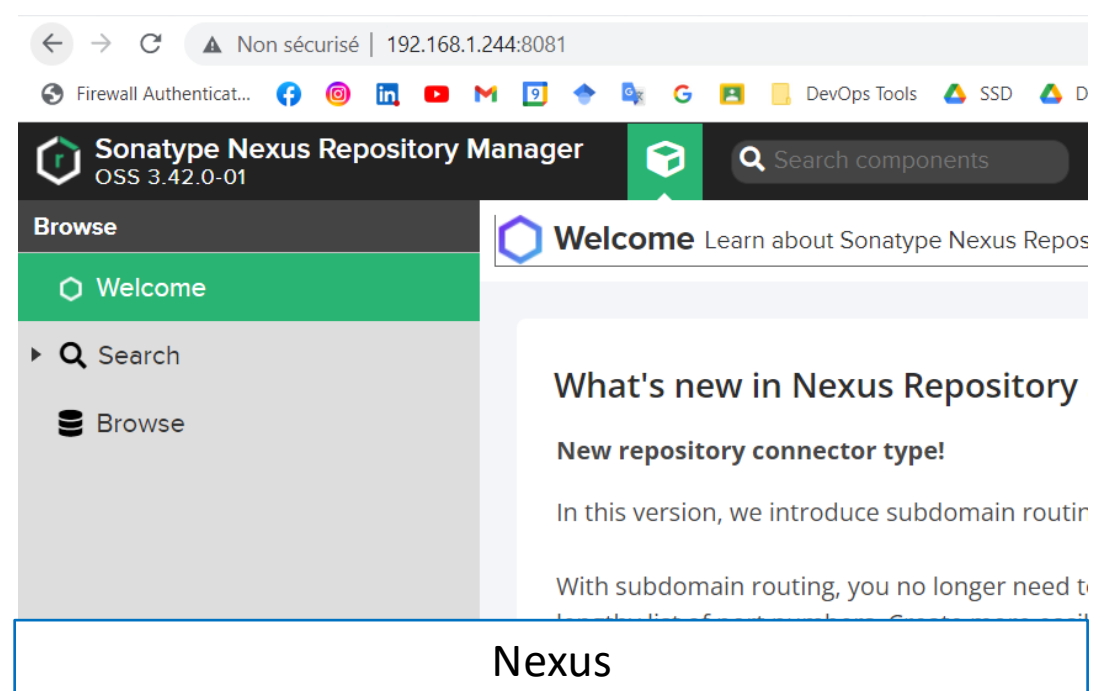
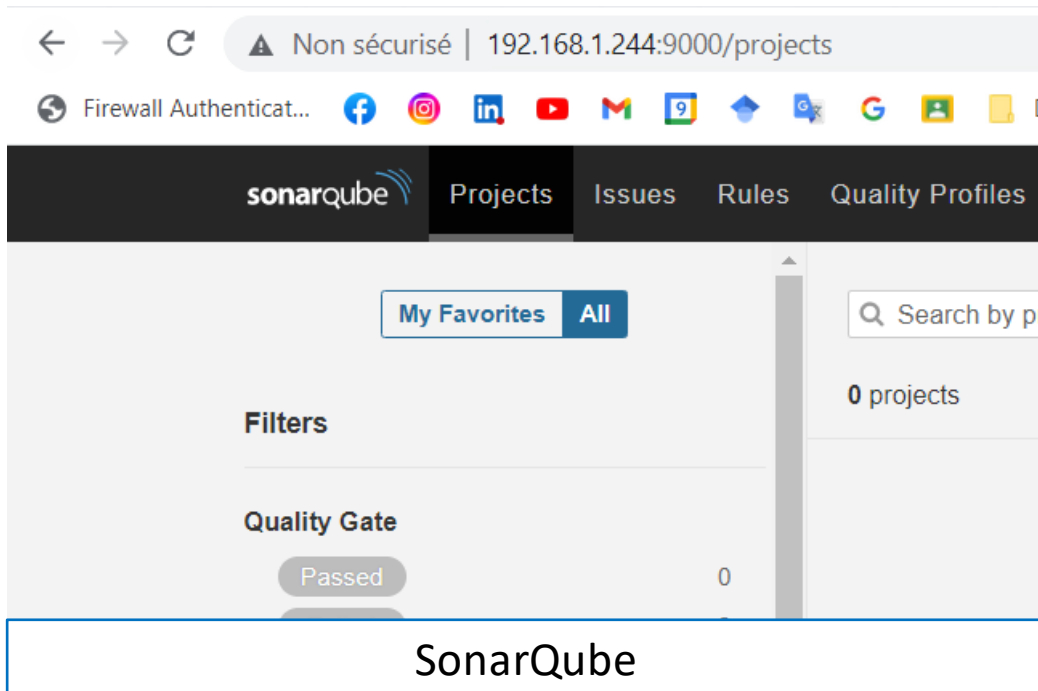
Les espaces de stockage réservés pour 'SonarQube'

L'espace de stockage réservé pour 'Nexus'

Déclaration des espaces de stockage

Docker Volume – Configuration dans Docker-Compose

```
[root@localhost SonarAndNexus]# docker-compose up
Creating volume "sonarandnexus_SonarQube_data" with default driver
Creating volume "sonarandnexus_SonarQube_extensions" with default driver
Creating volume "sonarandnexus_SonarQube_logs" with default driver
Removing sonarandnexus_sonarqube_1
sonarandnexus_nexus_1 is up-to-date
Recreating a13b6c82d625_sonarandnexus_sonarqube_1 ... done
```



Projet DevOps : Docker et Jenkins

1- Créer un **Dockerfile** dans votre projet achat (partie Spring) pour permettre la création de l'image. Vous pouvez créer ce fichier **à la racine de votre projet achat** et vous pouvez le pusher sur votre propre branche.

Exemple sur le projet timesheet-devops à adapter à votre projet achat. Mettez la bonne image java. Choisissez de dockerhub la version openjdk11. Essayer de récupérer le livrable de Nexus (ce n'est pas obligatoire). Exposez le port de votre application Spring Boot :

```
FROM openjdk:8-jdk-alpine
```

```
EXPOSE 8082
```

```
ADD target/timesheet-devops-1.0.jar timesheet-devops-1.0.jar
```

```
ENTRYPOINT ["java", "-jar", "/timesheet-devops-1.0.jar"]
```


Projet DevOps : Docker et Jenkins

2- Ajouter dans Jenkins le « stage » pour **créer** l'image de votre appliaiton (Partie Spring)

```
stage('Building image') {  
    steps{  
  
        « A Compléter ... »  
  
    }  
}
```

- Indications à adapter à votre projet achat (voir cours 2- Docker):
docker build -t mouradhassini/timesheet-devops:1.0.0 .
(pourquoi le point (.) das la comande ci-dessus ?)

Projet DevOps : Docker et Jenkins

3- Ajouter dans Jenkins le « stage » pour **déposer** l'image à déployer (Partie Spring) dans « **DockerHub** »

```
stage('Deploy Image') {  
    steps{
```

« A Compléter ... »

```
    }  
}
```

- Indications à adapter à votre projet achat (voir cours 2- Docker):

```
sh '''
```

```
docker login -u mouradhassini -p pwd
```

```
docker push mouradhassini/timesheet-devops:1.0.0
```

```
'''
```

(Vous pouvez ajouter des credentials dans Jenkins pour ne pas mettre le password dans la commande)

Projet DevOps : Docker et Jenkins

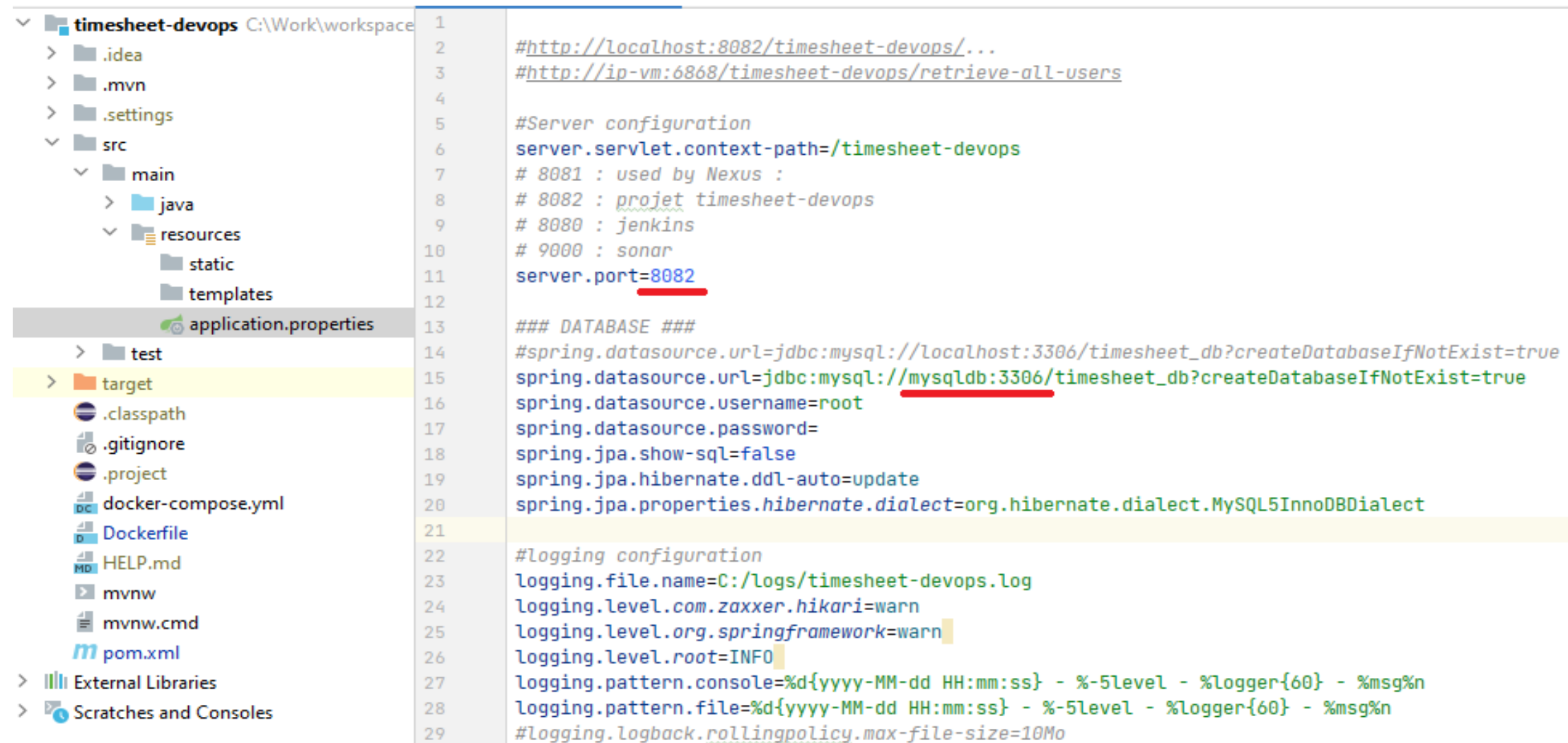
4- Créer un fichier **docker-compose.yml** (à la racine de votre projet achat par exemple) pour faire tourner votre application achat (Backend avec une base de données MySQL (inspirez vous de l'exemple ci-dessus). 2 Services sont à créer dans docker-compose.yml.

Voir exemple de **docker-compose.yml ci-dessus page 11** (à adapter à votre projet).

Attention : le fichier **application.properties** de votre application Spring Boot achat doit être mis à jour, pour pointer sur la bonne url de la base de données. Voir exemple de contenu page suivante :

Projet DevOps : Docker et Jenkins

Exemple de `application.properties` à adapter à votre application Spring Boot achat :



```
1
2 #http://localhost:8082/timesheet-devops/...
3 #http://ip-vm:6868/timesheet-devops/retrieve-all-users
4
5 #Server configuration
6 server.servlet.context-path=/timesheet-devops
7 # 8081 : used by Nexus :
8 # 8082 : projet timesheet-devops
9 # 8080 : jenkins
10 # 9000 : sonar
11 server.port=8082
12
13 ### DATABASE ###
14 #spring.datasource.url=jdbc:mysql://localhost:3306/timesheet_db?createDatabaseIfNotExist=true
15 spring.datasource.url=jdbc:mysql://mysql:3306/timesheet_db?createDatabaseIfNotExist=true
16 spring.datasource.username=root
17 spring.datasource.password=
18 spring.jpa.show-sql=false
19 spring.jpa.hibernate.ddl-auto=update
20 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
21
22 #logging configuration
23 logging.file.name=C:/logs/timesheet-devops.log
24 logging.level.com.zaxxer.hikari=warn
25 logging.level.org.springframework=warn
26 logging.level.root=INFO
27 logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} - %-5level - %logger{60} - %msg%n
28 logging.pattern.file=%d{yyyy-MM-dd HH:mm:ss} - %-5level - %logger{60} - %msg%n
29 #logging.logback.rollingpolicy.max-file-size=10Mo
```

Projet DevOps : Docker et Jenkins

4-bis : le fichier docker-compose.yml contiendra 3 services si vous allez créer un conteneur pour la partie Frontend aussi (**option ARCTIC uniquement**).

Projet DevOps : Docker et Jenkins

5- Ajouter le « stage » nécessaire pour lancer le fichier « Docker-compose » automatiquement avec l'orchestrateur Jenkins.

Indication : **docker compose up**

Comment faire pour éviter que le pipeline ne soit bloqué à cette étape et ne donne pas la main pour continuer avec les étapes suivantes du pipeline (grafana/prometheus qu'on verra la semaine prochaine, mail récapitulatif, ...) ?

DevOps project: to see logs

```
vagrant@vagrant:~$ docker ps
CONTAINER ID        IMAGE                                     COMMAND                  CREATED            STATUS              PORTS
20288758667b       mouradhassini/timesheet-devops:1.0.0  "java -jar /timeshee..." 26 minutes ago    Up 26 minutes      0.0.0.0:8082->8082/tcp, :::8082->8082/tcp
407318770c9       mysql:5.7                               "docker-entrypoint.s..." 26 minutes ago    Up 26 minutes      0.0.0.0:3306->3306/tcp, :::3306->3306/tcp
3905d2cf645c       grafana/grafana                         "/run.sh"               4 days ago        Up 2 hours         0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
d538a0c81f50       prom/prometheus                        "/bin/prometheus --c..." 4 days ago        Up About an hour    0.0.0.0:9090->9090/tcp, :::9090->9090/tcp
b8d789df0c0e       sonatype/nexus3                        "/opt/sonatype/nexus..." 2 weeks ago       Up About an hour    0.0.0.0:8081->8081/tcp, :::8081->8081/tcp
f87df4171155       sonarqube                               "/opt/sonarqube/dock..." 3 weeks ago       Up About an hour    0.0.0.0:9000->9000/tcp, :::9000->9000/tcp

vagrant@vagrant:~$ docker logs 20288758667b

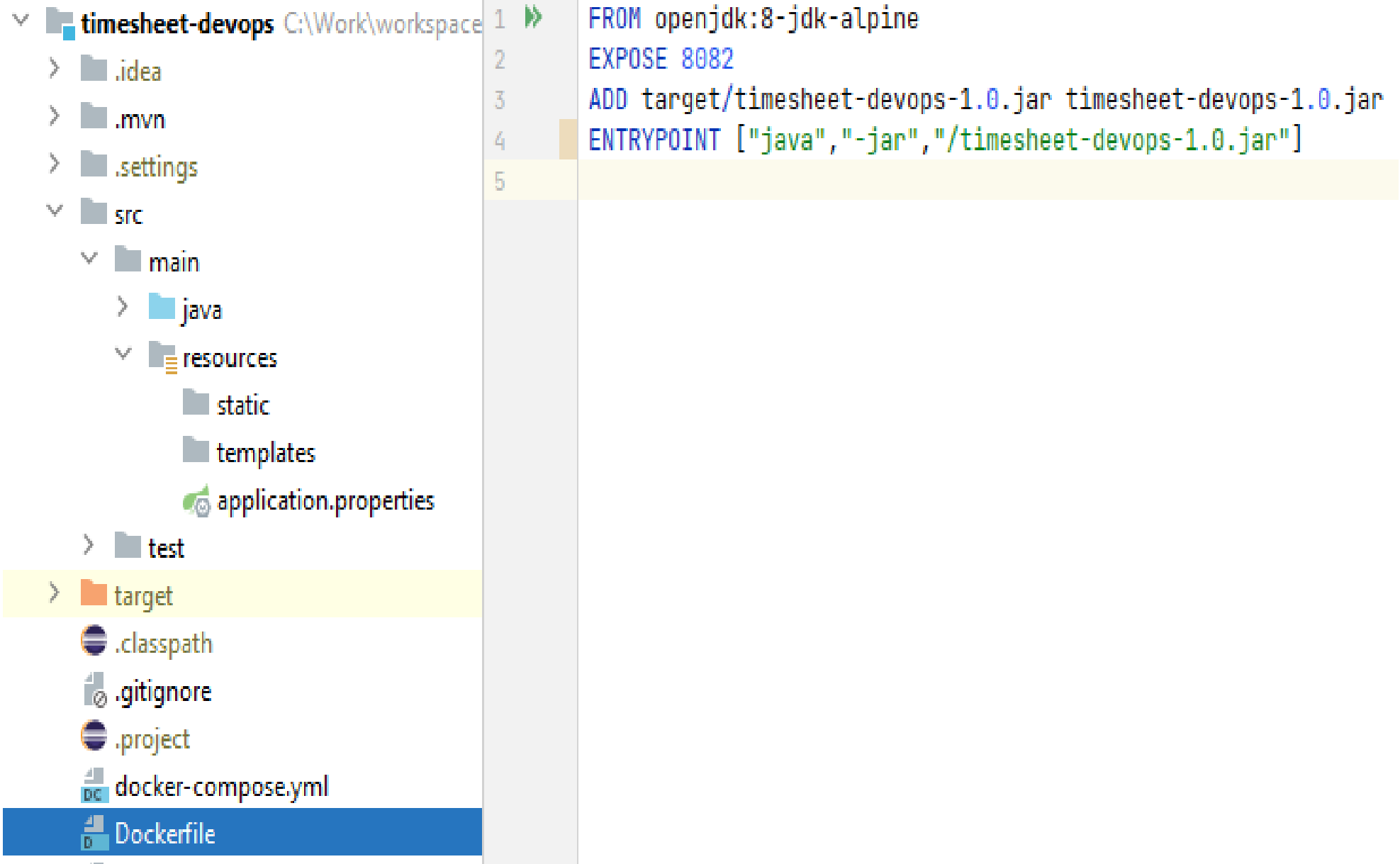
  ____ _
 / ___ \| | | |
/ /___ \| |_| |
 \___ \|____|_|_|
  =====|_|=====|_|_/=///_/

:: Spring Boot ::                (v2.5.4)

2023-11-02 21:36:04 - INFO - tn.esprit.spring.TimesheetDevopsApplication - Starting TimesheetDevopsApplication v1.0 using Java 1.8.0_212 on 20288758667b v
2023-11-02 21:36:04 - INFO - tn.esprit.spring.TimesheetDevopsApplication - No active profile set, falling back to default profiles: default
2023-11-02 21:36:09 - INFO - org.apache.catalina.core.StandardService - Starting service [Tomcat]
2023-11-02 21:36:09 - INFO - org.apache.catalina.core.StandardEngine - Starting Servlet engine: [Apache Tomcat/9.0.52]
2023-11-02 21:36:09 - INFO - o.a.c.c.C.[Tomcat].[localhost].[/timesheet-devops] - Initializing Spring embedded WebApplicationContext
2023-11-02 21:36:10 - INFO - org.hibernate.jpa.internal.util.LogHelper - HHH000204: Processing PersistenceUnitInfo [name: default]
2023-11-02 21:36:10 - INFO - org.hibernate.Version - HHH000412: Hibernate ORM core version 5.4.32.Final
2023-11-02 21:36:10 - INFO - org.hibernate.annotations.common.Version - HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2023-11-02 21:36:13 - INFO - org.hibernate.dialect.Dialect - HHH000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
2023-11-02 21:36:17 - INFO - o.h.e.transaction.jta.platform.internal.JtaPlatformInitiator - HHH000490: Using JtaPlatform implementation: [org.hibernate.er
2023-11-02 21:36:19 - WARN - o.s.b.a.orm.jpa.JpaBaseConfiguration$JpaWebConfiguration - spring.jpa.open-in-view is enabled by default. Therefore, database
spring.jpa.open-in-view to disable this warning
2023-11-02 21:36:20 - INFO - tn.esprit.spring.TimesheetDevopsApplication - Started TimesheetDevopsApplication in 18.314 seconds (JVM running for 24.682)
2023-11-02 21:38:52 - INFO - o.a.c.c.C.[Tomcat].[localhost].[/timesheet-devops] - Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-11-02 21:56:49 - WARN - o.s.web.servlet.mvc.support.DefaultHandlerExceptionResolver - Resolved [org.springframework.web.HttpRequestMethodNotSupportedException]

vagrant@vagrant:~$
```

DevOps project: a working config



The screenshot shows an IDE interface with a project structure on the left and a Dockerfile on the right.

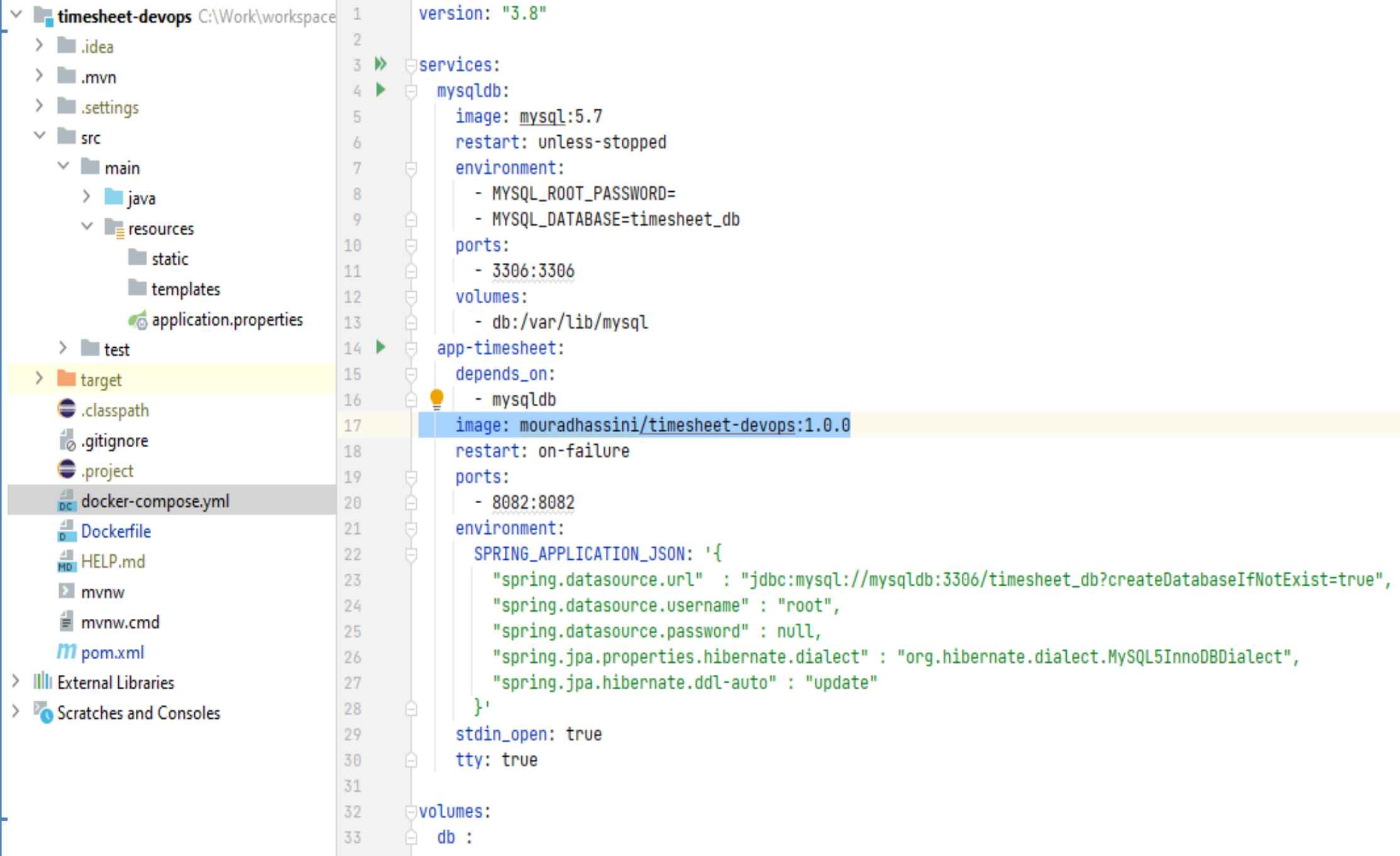
Project Structure (Left Panel):

- timesheet-devops C:\Work\workspace
 - .idea
 - .mvn
 - .settings
 - src
 - main
 - java
 - resources
 - static
 - templates
 - application.properties
 - test
 - target
 - .classpath
 - .gitignore
 - .project
 - docker-compose.yml
 - Dockerfile

Dockerfile (Right Panel):

```
1 FROM openjdk:8-jdk-alpine
2 EXPOSE 8082
3 ADD target/timesheet-devops-1.0.jar timesheet-devops-1.0.jar
4 ENTRYPOINT ["java","-jar","/timesheet-devops-1.0.jar"]
5
```

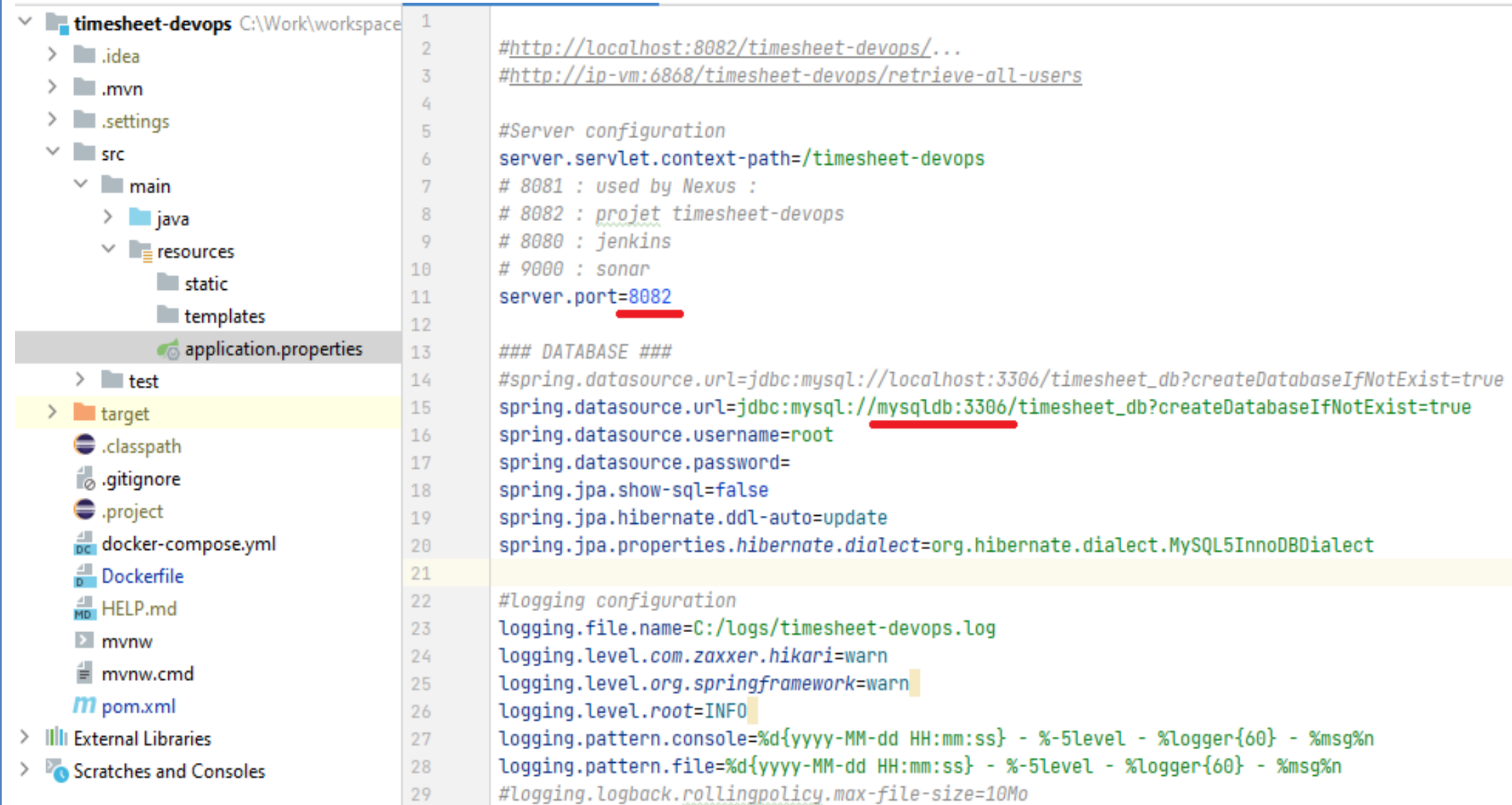

DevOps project: a working config



The screenshot displays an IDE interface with a project structure on the left and a code editor on the right. The project structure shows a directory named 'timesheet-devops' containing various files and subdirectories, including 'src/main/resources', 'test', 'target', and 'docker-compose.yml'. The code editor shows the content of 'docker-compose.yml', which defines two services: 'mysql' and 'app-timesheet'.

```
1 version: "3.8"
2
3 services:
4   mysql:
5     image: mysql:5.7
6     restart: unless-stopped
7     environment:
8       - MYSQL_ROOT_PASSWORD=
9       - MYSQL_DATABASE=timesheet_db
10    ports:
11      - 3306:3306
12    volumes:
13      - db:/var/lib/mysql
14  app-timesheet:
15    depends_on:
16      - mysql
17    image: mouradhassini/timesheet-devops:1.0.0
18    restart: on-failure
19    ports:
20      - 8082:8082
21    environment:
22      SPRING_APPLICATION_JSON: '{
23        "spring.datasource.url" : "jdbc:mysql://mysql:3306/timesheet_db?createDatabaseIfNotExist=true",
24        "spring.datasource.username" : "root",
25        "spring.datasource.password" : null,
26        "spring.jpa.properties.hibernate.dialect" : "org.hibernate.dialect.MySQL5InnoDBDialect",
27        "spring.jpa.hibernate.ddl-auto" : "update"
28      }'
29    stdin_open: true
30    tty: true
31
32 volumes:
33   db :
```

DevOps project: a working config



The screenshot displays an IDE interface with a project structure on the left and a configuration file on the right.

Project Structure (Left):

- timesheet-devops C:\Work\workspace
 - .idea
 - .mvn
 - .settings
 - src
 - main
 - java
 - resources
 - static
 - templates
 - test
 - target
 - .classpath
 - .gitignore
 - .project
 - docker-compose.yml
 - Dockerfile
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
 - External Libraries
 - Scratches and Consoles

Configuration File (Right):

```
1 #http://localhost:8082/timesheet-devops/...
2 #http://ip-vm:6868/timesheet-devops/retrieve-all-users
3
4
5 #Server configuration
6 server.servlet.context-path=/timesheet-devops
7 # 8081 : used by Nexus :
8 # 8082 : projet timesheet-devops
9 # 8080 : jenkins
10 # 9000 : sonar
11 server.port=8082
12
13 ### DATABASE ###
14 #spring.datasource.url=jdbc:mysql://localhost:3306/timesheet_db?createDatabaseIfNotExist=true
15 spring.datasource.url=jdbc:mysql://mysql:3306/timesheet_db?createDatabaseIfNotExist=true
16 spring.datasource.username=root
17 spring.datasource.password=
18 spring.jpa.show-sql=false
19 spring.jpa.hibernate.ddl-auto=update
20 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
21
22 #logging configuration
23 logging.file.name=C:/logs/timesheet-devops.log
24 logging.level.com.zaxxer.hikari=warn
25 logging.level.org.springframework=warn
26 logging.level.root=INFO
27 logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} - %-5level - %logger{60} - %msg%n
28 logging.pattern.file=%d{yyyy-MM-dd HH:mm:ss} - %-5level - %logger{60} - %msg%n
29 #logging.logback.rollingpolicy.max-file-size=10Mo
```

DevOps project: a working config

The screenshot displays a REST client interface with the following elements:

- Overview Tab:** Shows the request method as **POST** and the URL as `http://192.168.171.222`.
- Request Configuration:**
 - Method:** **POST** (highlighted with a red box).
 - URL:** `http://192.168.171.222:8082/timesheet-devops/user/add-user` (highlighted with a red box).
 - Body Tab:** Selected, showing the request body (highlighted with a red box).
 - Body Type:** **raw** (highlighted with a red box).
 - Content Type:** **JSON** (highlighted with a red box).
- Request Body:** A text area containing the JSON payload (highlighted with a red box):

```
1 {  
2   "id": 1,  
3   "lastName": null,  
4   "dateNaissance": null,  
5   "role": null  
6 }
```
- Response Section:**
 - Status:** 200 OK
 - Time:** 1646 ms
 - Size:** 221 B
 - Save as example** button.
- Response Body:** The response is displayed in the **Pretty** view, showing the JSON object:

```
1 {  
2   "id": 1,  
3   "lastName": null,  
4   "dateNaissance": null,  
5   "role": null  
6 }
```

DevOps project: a working config

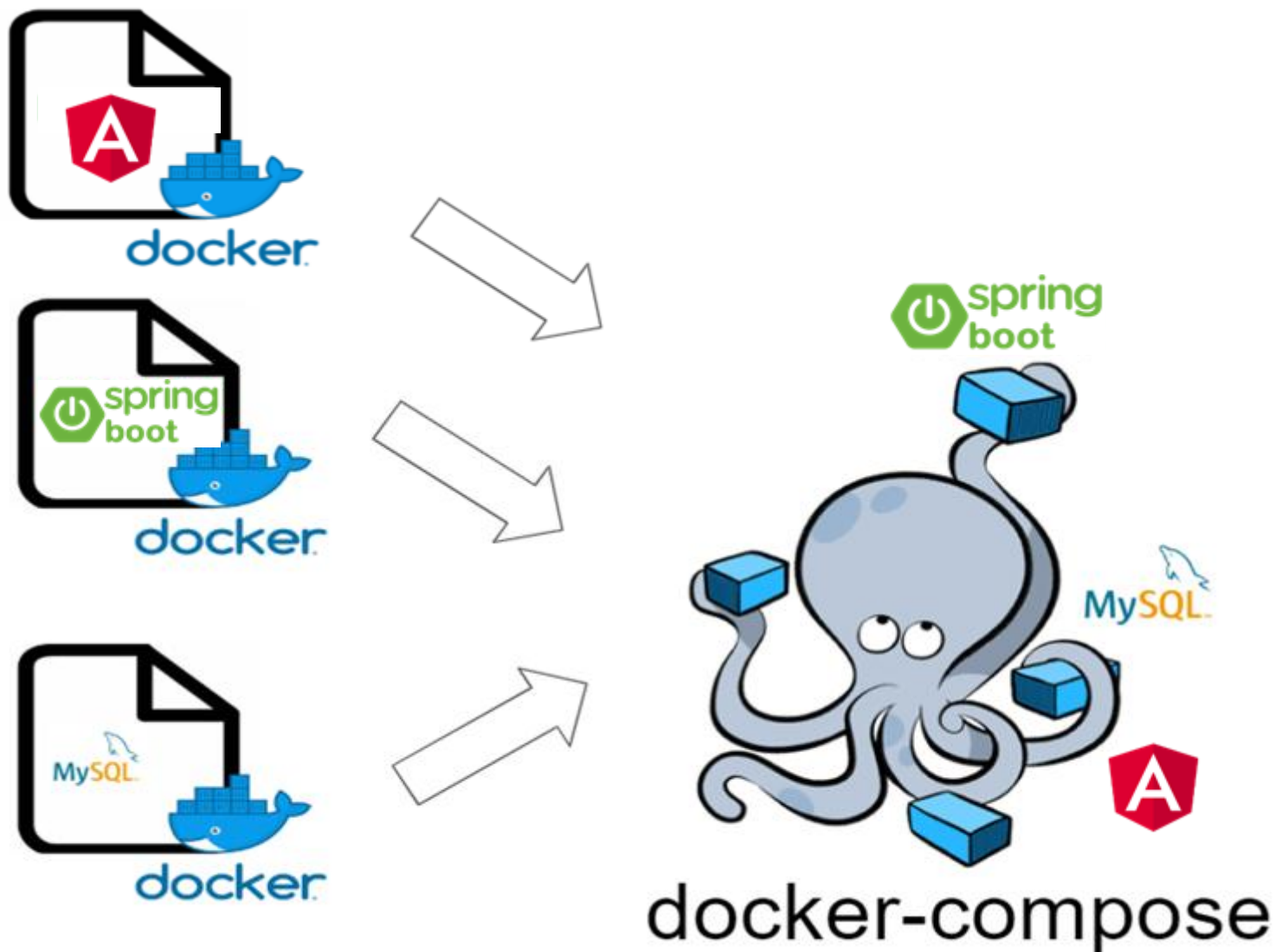
The screenshot displays a REST client interface with the following components:

- Request Section:**
 - Method: **POST**
 - URL: `http://192.168.171.222:8082/timesheet-devops/user/add-user`
 - Buttons: **Send** and a dropdown arrow.
 - Tabs: Params, Authorization, Headers (9), **Body** (selected), Pre-request Script, Tests, Settings.
 - Body Type: **JSON** (selected from a dropdown menu that also includes none, form-data, x-www-form-urlencoded, raw, binary, and GraphQL).
- Request Body:**

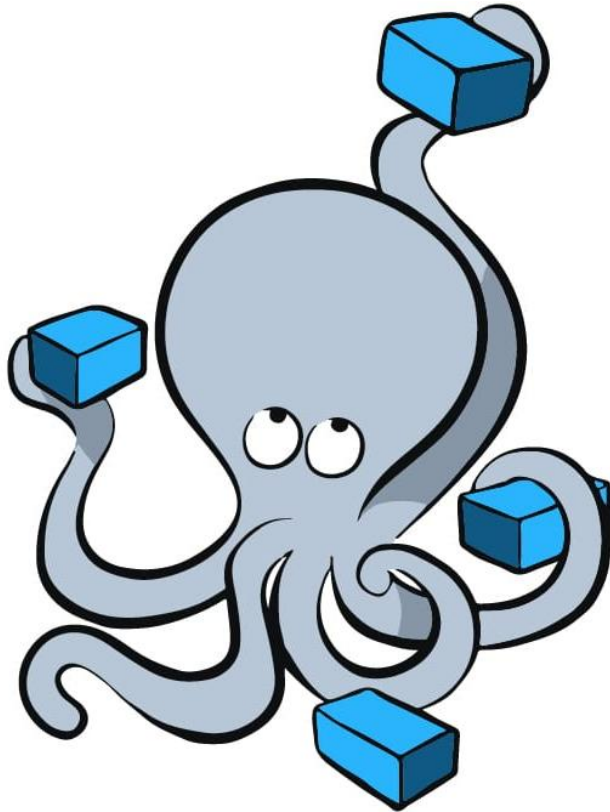
```
1 {
2   "lastName": "ASKRI",
3   "dateNaissance": "2000-01-21",
4   "role": "ADMINISTRATEUR"
5 }
```
- Response Section:**
 - Tabs: **Body** (selected), Cookies, Headers (5), Test Results.
 - Status: **200 OK**, Time: 223 ms, Size: 263 B.
 - Buttons: Save as example, and a menu icon.
 - Format: **Pretty** (selected from a dropdown menu that also includes Raw, Preview, and Visualize).
- Response Body:**

```
1 {
2   "id": 2,
3   "lastName": "ASKRI",
4   "dateNaissance": "2000-01-21T00:00:00.000+00:00",
5   "role": "ADMINISTRATEUR"
6 }
```

Projet DevOps : Docker et Jenkins



Docker Compose



docker
Compose