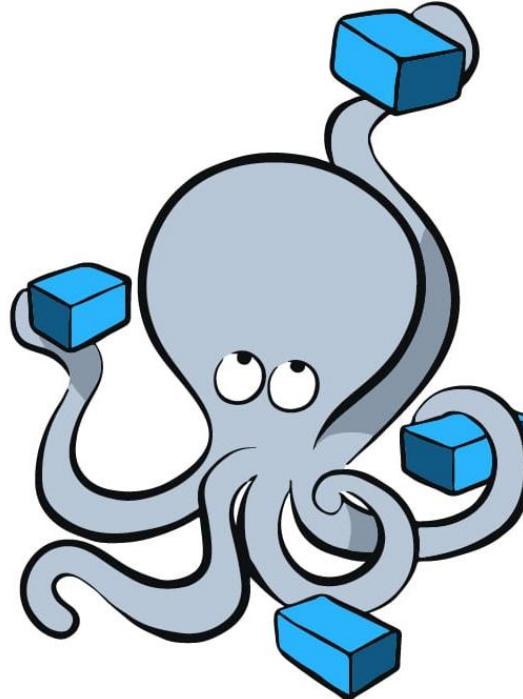


Docker Compose



docker
Compose

Plan du cours

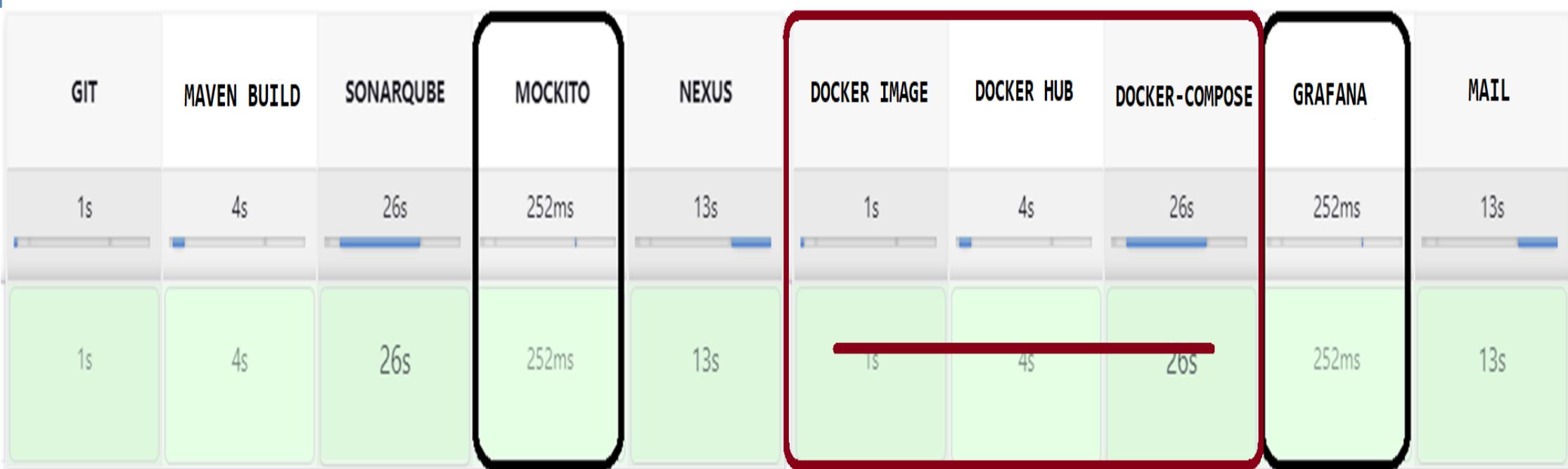
- Introduction
- Docker
- Docker Compose
- Docker Compose et Jenkins

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 (Continuous delivery and deployment)**

Introduction

Projet DevOps Final :



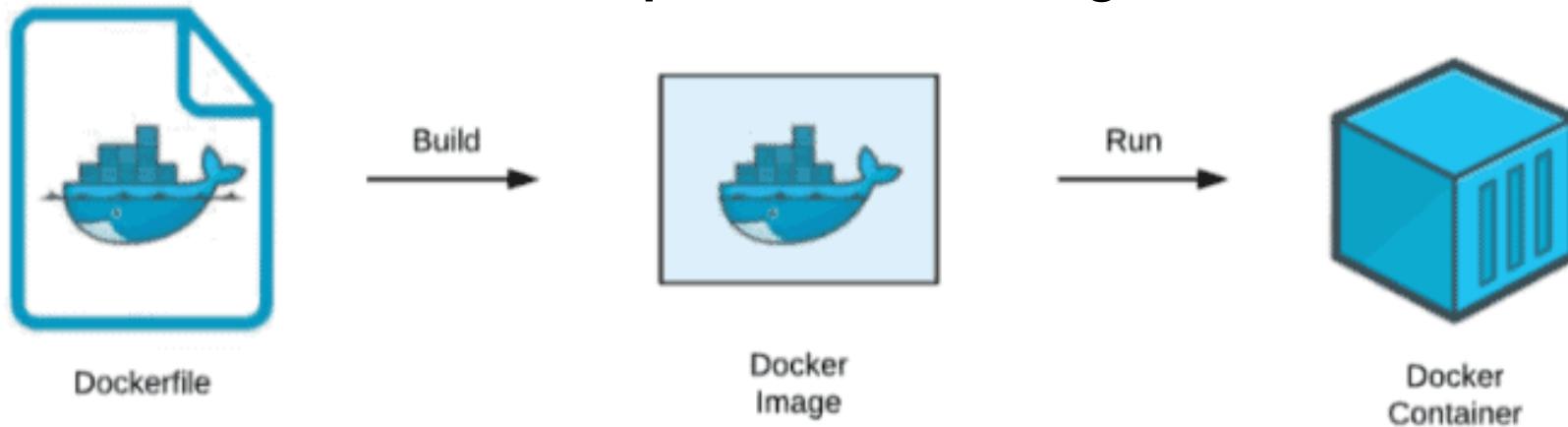
Introduction

- L'objectif de la partie CD (déploiement et livraison continu) est de livrer 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>**

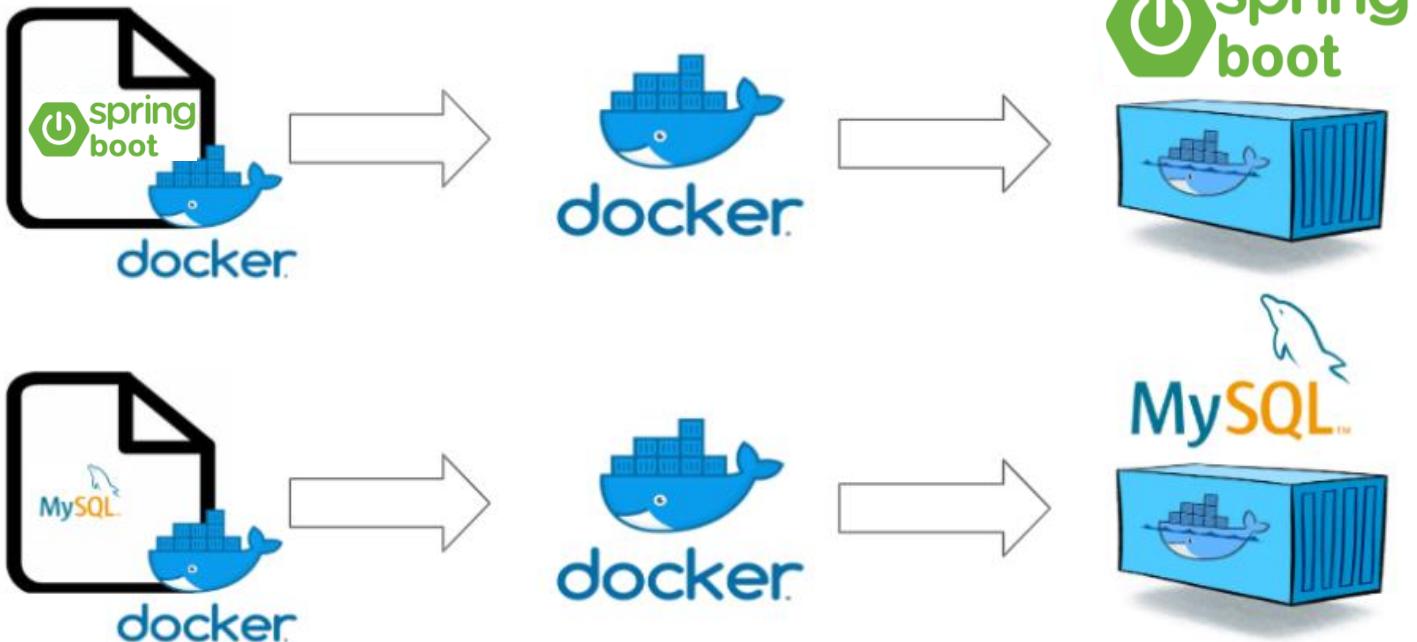


Introduction

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.

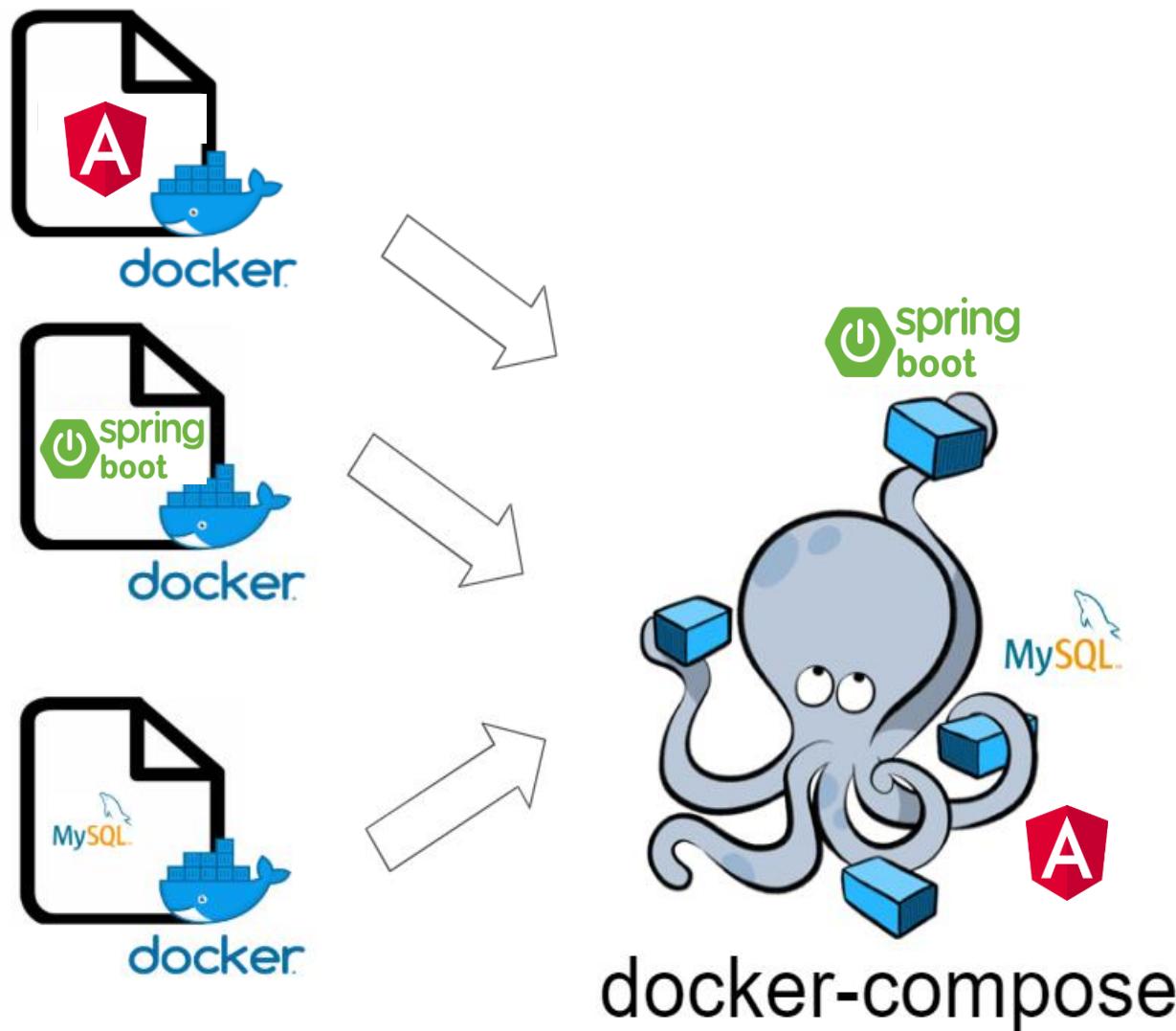
```
docker run -p 9090:9090 --network mynetwork -d app-image-name
```



```
docker run --name mysql -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 est 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 - Exemple

The screenshot shows a file browser interface with a tree view of project files on the left and the contents of the selected file on the right.

Project Structure:

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

Content of docker-compose.yml:

```
version: "3.8"
services:
  mysqlDb:
    image: mysql:5.7
    restart: unless-stopped
    environment:
      - MYSQL_ROOT_PASSWORD=
      - MYSQL_DATABASE=timesheet_db
    ports:
      - 3306:3306
    volumes:
      - db:/var/lib/mysql
  app-timesheet:
    depends_on:
      - mysqlDb
    image: mouradkhassini/timesheet-devops:1.0.0
    restart: on-failure
    ports:
      - 8082:8082
    environment:
      SPRING_APPLICATION_JSON: '{
        "spring.datasource.url" : "jdbc:mysql://mysqlDb:3306/timesheet_db?createDatabaseIfNotExist=true",
        "spring.datasource.username" : "root",
        "spring.datasource.password" : null,
        "spring.jpa.properties.hibernate.dialect" : "org.hibernate.dialect.MySQL5InnoDBDialect",
        "spring.jpa.hibernate.ddl-auto" : "update"
      }'
    stdin_open: true
    tty: true
volumes:
  db :
```

Or build : ./

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

Projet DevOps : Docker et Jenkins

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

Exemple sur le projet timesheet-devops à adapter à votre projet tp-foyer. Mettez la bonne image java. Choisissez de dockerhub la version openjdk11. Essayez 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 application (Partie Spring)

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

- Indications à adapter à votre projet tp-foyer (voir cours 2- Docker):
docker build -t mouradhassini/timesheet-devops:1.0.0 .
(pourquoi le point (.) dans la commande 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 tp-foyer (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 tp-foyer par exemple) pour faire tourner votre application tp-foyer (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 tp-foyer 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 tp-foyer:

The screenshot shows a file browser interface with the following project structure:

- 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

The `application.properties` file content is as follows:

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

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.

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 sans continuer avec les étapes suivantes du pipeline ?

DevOps project: to see logs

DevOps project: a working config

The image shows a file explorer interface with two panes. The left pane displays the project structure:

- 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**

DevOps project: a working config

The screenshot shows a Java IDE interface with a project tree on the left and a code editor on the right.

Project Structure:

- 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

Code Editor Content (docker-compose.yml):

```
version: "3.8"

services:
  mysqlDb:
    image: mysql:5.7
    restart: unless-stopped
    environment:
      - MYSQL_ROOT_PASSWORD=
      - MYSQL_DATABASE=timesheet_db
    ports:
      - 3306:3306
    volumes:
      - db:/var/lib/mysql
  app-timesheet:
    depends_on:
      - mysqlDb
    image: mouradkhassini/timesheet-devops:1.0.0
    restart: on-failure
    ports:
      - 8082:8082
    environment:
      SPRING_APPLICATION_JSON: '{
        "spring.datasource.url" : "jdbc:mysql://mysqlDb:3306/timesheet_db?createDatabaseIfNotExist=true",
        "spring.datasource.username" : "root",
        "spring.datasource.password" : null,
        "spring.jpa.properties.hibernate.dialect" : "org.hibernate.dialect.MySQL5InnoDBDialect",
        "spring.jpa.hibernate.ddl-auto" : "update"
      }'
    stdin_open: true
    tty: true
  volumes:
    db :
```

DevOps project: a working config

The screenshot shows a file tree for a Java project named "timesheet-devops" located at C:\Work\workspace. The "application.properties" file is open in the editor. The code in the file is as follows:

```
1 #http://localhost:8082/timesheet-devops/...
2 #http://ip-vm:6868/timesheet-devops/retrieve-all-users
3
4 #Server configuration
5 server.servlet.context-path=/timesheet-devops
6 # 8081 : used by Nexus :
7 # 8082 : projet timesheet-devops
8 # 8080 : jenkins
9 # 9000 : sonar
10
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://mysqlDb: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%
28 logging.pattern.file=%d{yyyy-MM-dd HH:mm:ss} - %-5level - %logger{60} - %msg%
29 #logging.logback.rollingpolicy.max-file-size=10Mo
```

URL Swagger :

http://192.168.50.4:8089/tpoyer/swagger-ui/index.html#/

The screenshot shows the Swagger UI interface for a REST API. At the top, the URL `http://192.168.50.4:8089/tpoyer/swagger-ui/index.html#/` is displayed in a browser-like address bar, with the entire URL highlighted by a red box. Below the address bar, the Swagger logo and the text "Supported by SMARTBEAR" are visible. To the right, a navigation bar shows the path `/tpoyer/v3/api-docs`. The main content area is titled "universite-rest-controller" and lists three API endpoints:

- PUT /universite/modify-universite**
- POST /universite/add-universite**
- GET /universite/retrieve-universite/{universite-id}**

DevOps project: a working config

The screenshot shows the Postman application interface. A red box highlights the 'POST' method and the URL 'http://192.168.171.222:8082/timesheet-devops/user/add-user'. Another red box highlights the 'Body' tab, which is set to 'raw' and 'JSON'. A third red box highlights the JSON response body, which contains the following data:

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

DevOps project: a working config

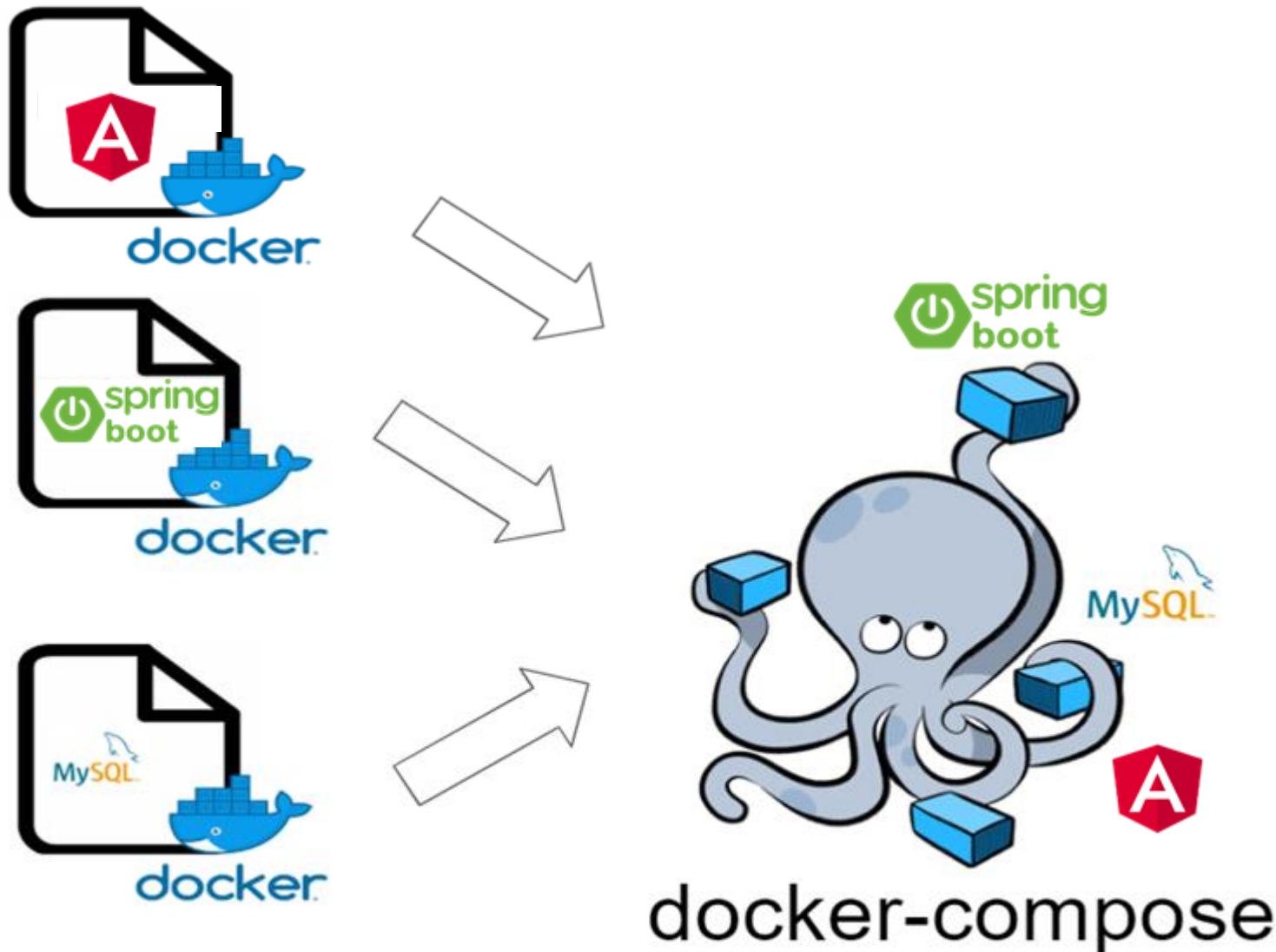
The screenshot shows a Postman interface for a POST request to `http://192.168.171.222:8082/timesheet-devops/user/add-user`. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "lastName": "ASKRI",  
3   "dateNaissance": "2000-01-21",  
4   "role": "ADMINISTRATEUR"  
5 }
```

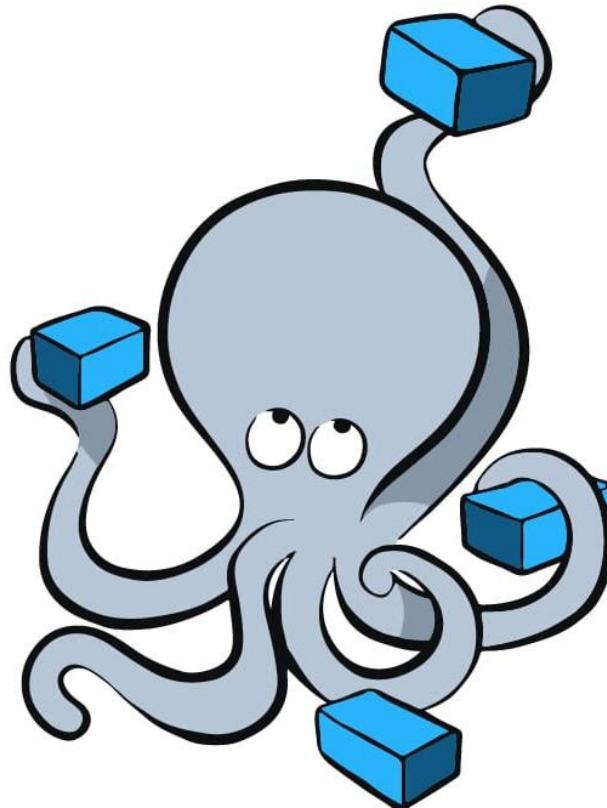
The response status is 200 OK with a time of 223 ms and a size of 263 B. The response body is:

```
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