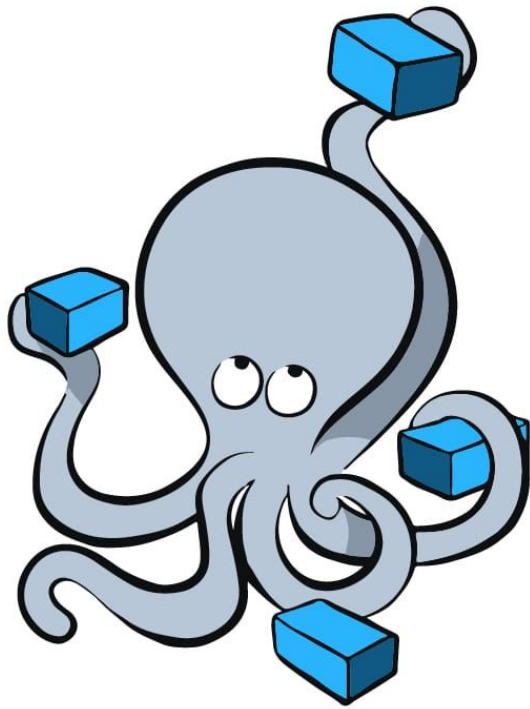


Docker Compose et Volume



docker

Compose et Volume

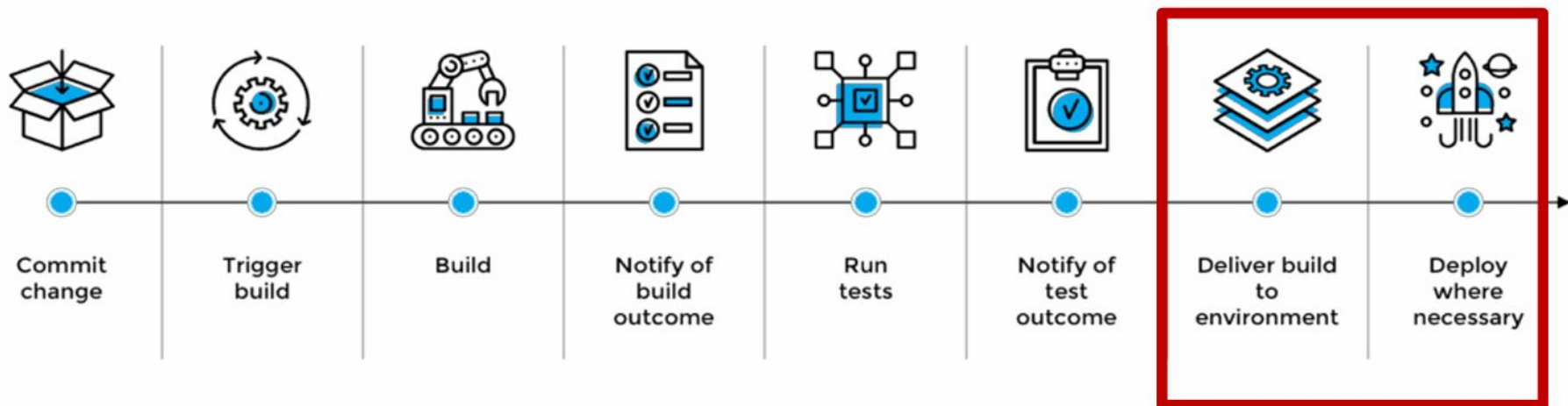
Bureau E204

Plan du cours

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

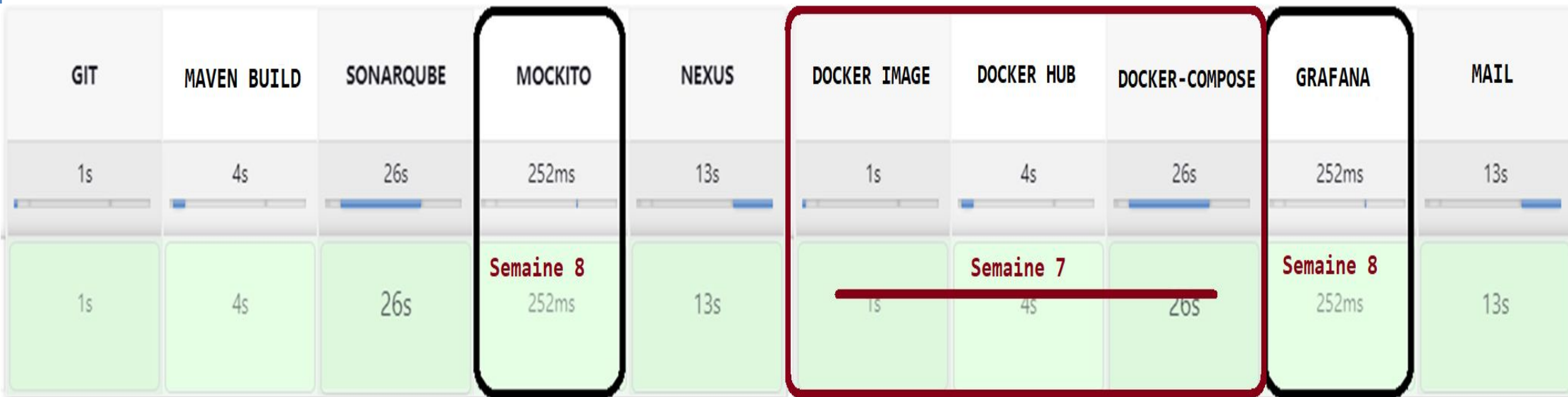
Introduction

- 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

- Qu'est ce qu'une livraison continue ?
- Qu'est ce qu'on doit livrer ?
- Où dois-je livrer le livrable ?
- Quelle est la différence entre la livraison continue et le déploiement continu ?

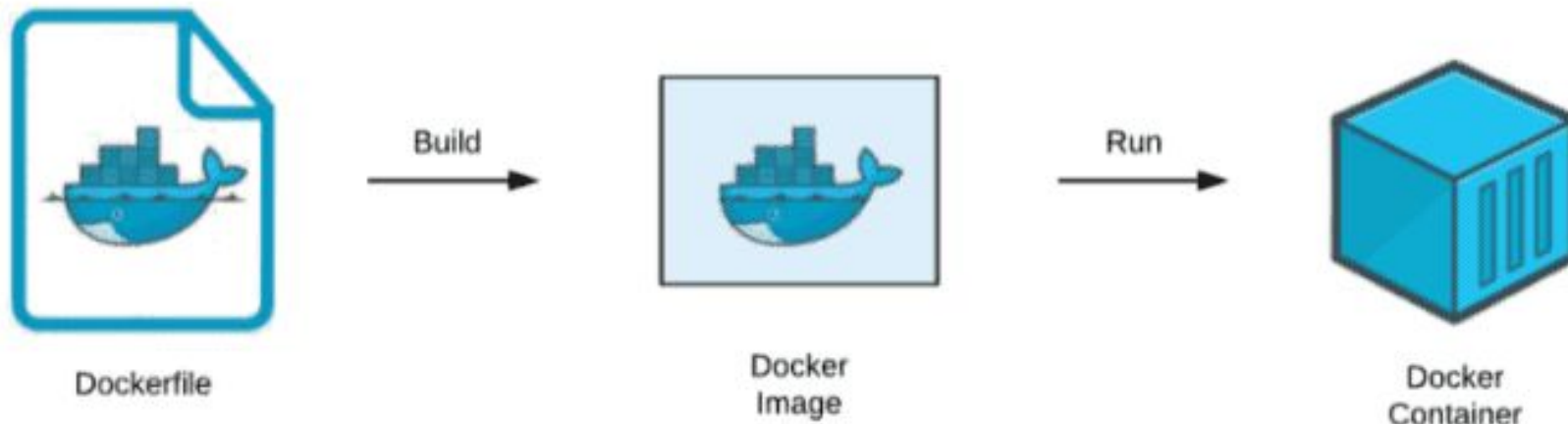
Introduction

- 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>**

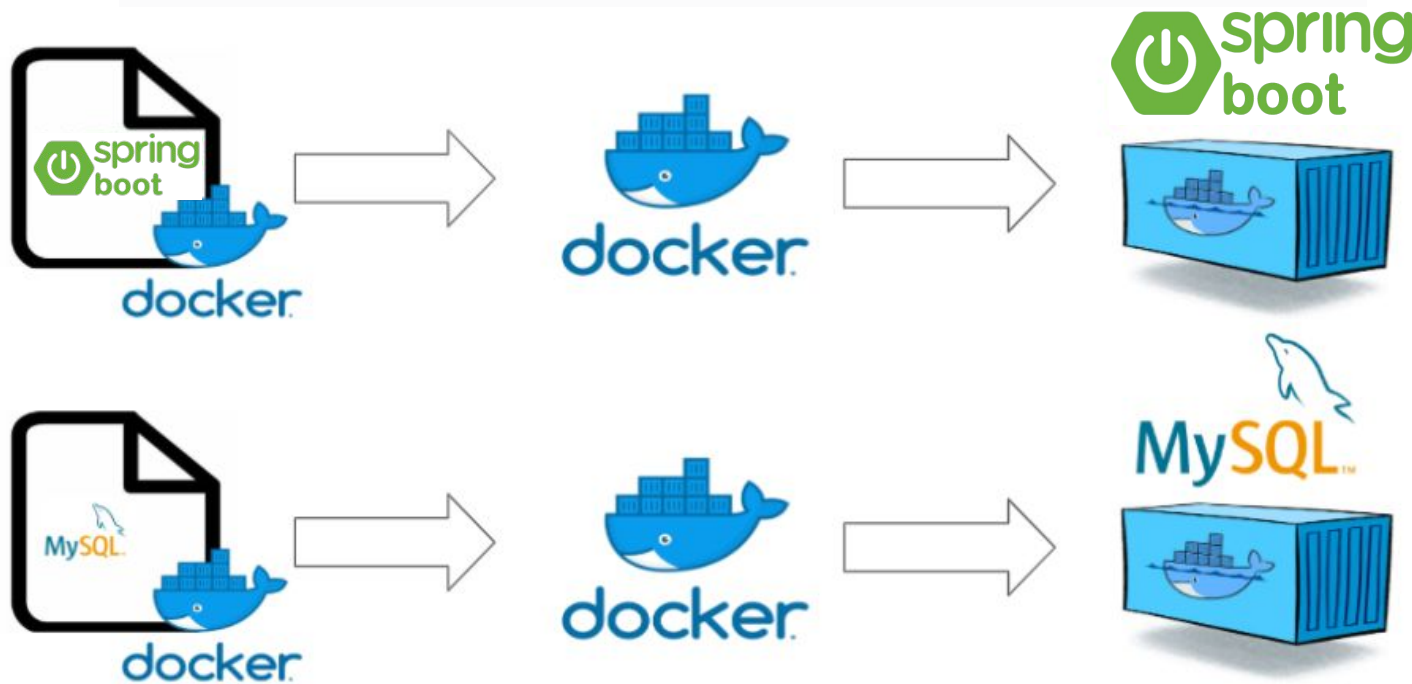


Introduction

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.

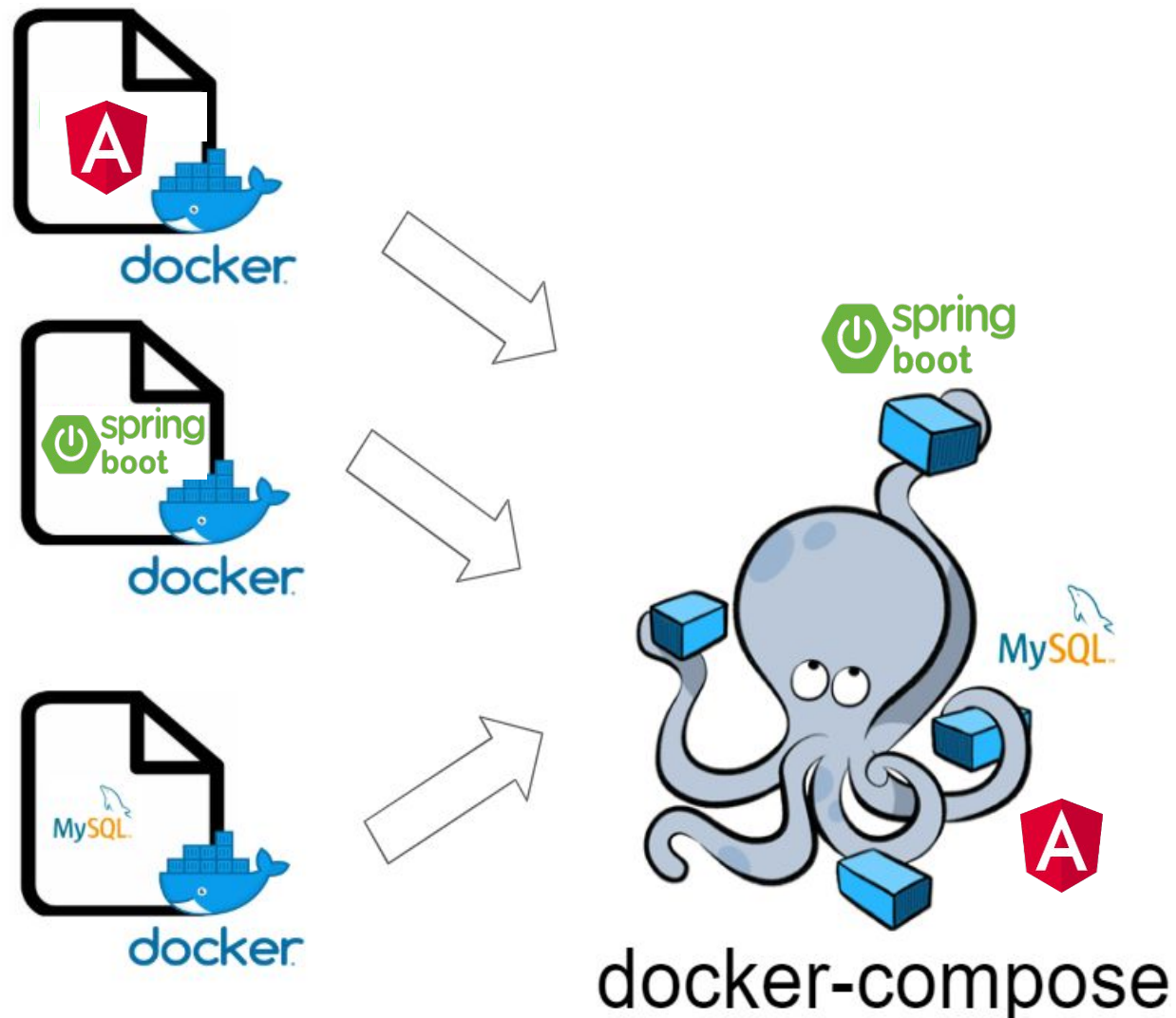
```
docker run -p 9090:9090 --network mynetwork -d app-image-name
```



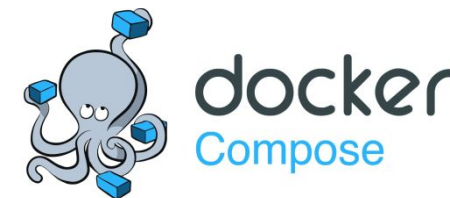
```
docker run --name mysqldb --network mynetwork -e MYSQL_ROOT_PASSWORD=my-secret-pw -v  
/home/mysql/data:/var/lib/mysql -d mysql:8
```


Introduction

- Et là, il nous faut docker compose.



Docker Compose

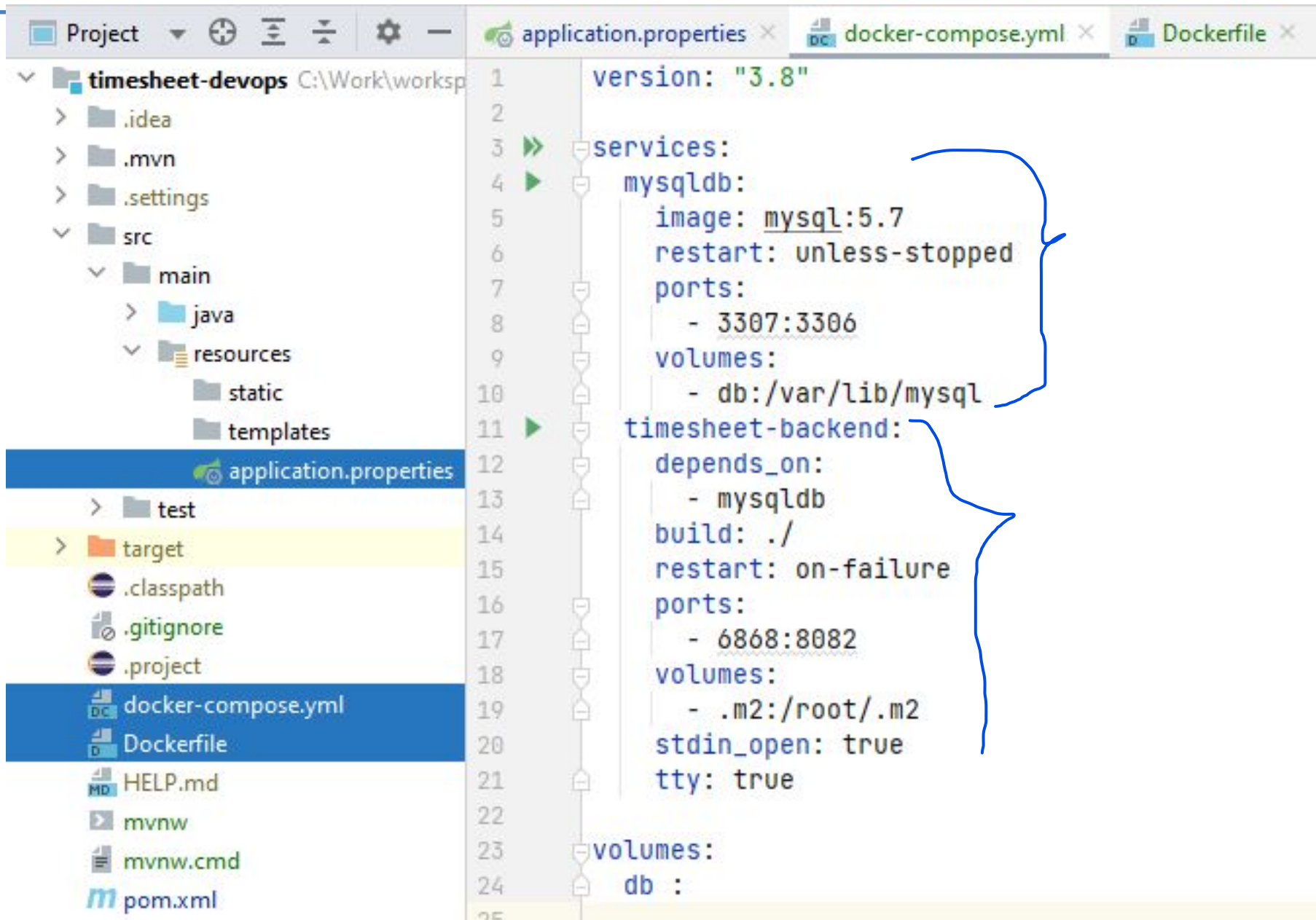


- **Docker Compose est un outil permettant de définir et d'exécuter des applications Docker multi-conteneurs.**
- Dans cette logique, chaque partie de l'application (code, base de données, serveur web, ...) sera hébergée par un conteneur.
- Cet outil repose sur le langage YAML pour décrire l'architecture 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



The screenshot shows an IDE with a project structure on the left and a Docker Compose configuration file in the center. The project structure includes a `timesheet-devops` directory with subdirectories `.idea`, `.mvn`, `.settings`, `src` (containing `main` with `java` and `resources`), `test`, `target`, `.classpath`, `.gitignore`, `.project`, `docker-compose.yml`, `Dockerfile`, `HELP.md`, `mvnw`, `mvnw.cmd`, and `pom.xml`. The `docker-compose.yml` file is open in the center, showing the following configuration:

```
1 version: "3.8"
2
3 services:
4   mysql:
5     image: mysql:5.7
6     restart: unless-stopped
7     ports:
8       - 3307:3306
9     volumes:
10      - db:/var/lib/mysql
11   timesheet-backend:
12     depends_on:
13       - mysql
14     build: ./
15     restart: on-failure
16     ports:
17       - 6868:8082
18     volumes:
19       - .m2:/root/.m2
20     stdin_open: true
21     tty: true
22
23 volumes:
24   db :
```

Blue brackets highlight the `mysql` service configuration (lines 4-10) and the `timesheet-backend` service configuration (lines 11-21). The `timesheet-backend` service is also highlighted with a yellow background.

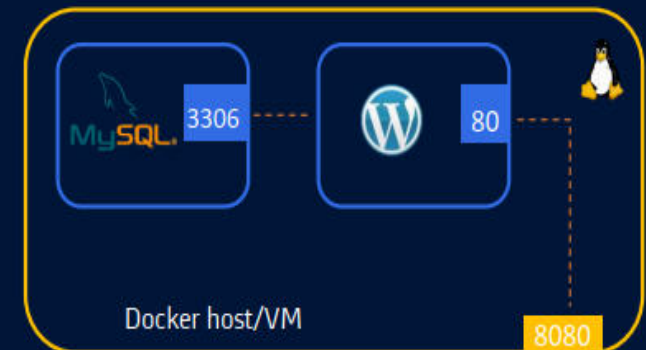
Docker Compose - Un autre Exemple

```
version: '3.3'
services:
  wordpress:
    image: wordpress
    depends_on:
      - mysql
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: mysql
      WORDPRESS_DB_NAME: wordpress
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
    volumes:
      - ./wordpress-data:/var/www/html
    networks:
      - my_net
  mysql:
    image: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - my_net
volumes:
  mysql-data:
networks:
  my_net:
```

```
wordpress:
  image: wordpress
  depends_on:
    - mysql
  ports:
    - 8080:80
  environment:
    WORDPRESS_DB_HOST: mysql
    WORDPRESS_DB_NAME: wordpress
    WORDPRESS_DB_USER: wordpress
    WORDPRESS_DB_PASSWORD: wordpress
  volumes:
    - ./wordpress-data:/var/www/html
  networks:
    - my_net
```

```
mysql:
  image: mariadb
  environment:
    MYSQL_ROOT_PASSWORD: wordpress
    MYSQL_DATABASE: wordpress
    MYSQL_USER: wordpress
    MYSQL_PASSWORD: wordpress
  volumes:
    - mysql-data:/var/lib/mysql
  networks:
    - my_net
```

```
volumes:
  mysql-data:
networks:
  my_net:
```



Installation Docker Compose (3 Commandes)

```
mkdir -p ~/.docker/cli-plugins/
```

```
curl -SL
```

```
https://github.com/docker/compose/releases/download/v2.3.3/docker-compose-linux-x86\_64 -o ~/.docker/cli-plugins/docker-compose
```

```
chmod +x ~/.docker/cli-plugins/docker-compose
```

```
docker compose version
```


Installation Docker Compose

```
vagrant@vagrant: ~  
vagrant@vagrant:~$ mkdir -p ~/.docker/cli-plugins/  
vagrant@vagrant:~$ curl -SL https://github.com/docker/compose/releases/download/  
v2.3.3/docker-compose-linux-x86_64 -o ~/.docker/cli-plugins/docker-compose  
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current  
           % Dload  % Upload   Total   Spent    Left     Speed  
0         0     0      0     0      0     0      0  --:--:-- --:--:-- --:--:--    0  
100 24.8M  100 24.8M    0     0 3085k      0  0:00:08  0:00:08 --:--:-- 3666k  
vagrant@vagrant:~$ chmod +x ~/.docker/cli-plugins/docker-compose  
vagrant@vagrant:~$ docker compose version  
Docker Compose version v2.3.3  
vagrant@vagrant:~$ _
```

Les commandes principales

Les commandes 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 stop
- Comment supprimer un docker compose ?
docker compose down

Docker compose - commandes

<code>docker compose start</code>	# Starts existing containers for a service.
<code>docker compose stop</code>	Stops running containers without removing them.
<code>docker compose pause</code>	Pauses running containers of a service.
<code>docker compose unpause</code>	Unpauses paused containers of a service.
<code>docker compose ps</code>	Lists containers.
<code>docker compose up</code>	Builds, (re)creates, starts, and attaches to containers for a service.
<code>docker compose down</code>	Stops containers and removes containers, networks, volumes, and images created by up.

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.

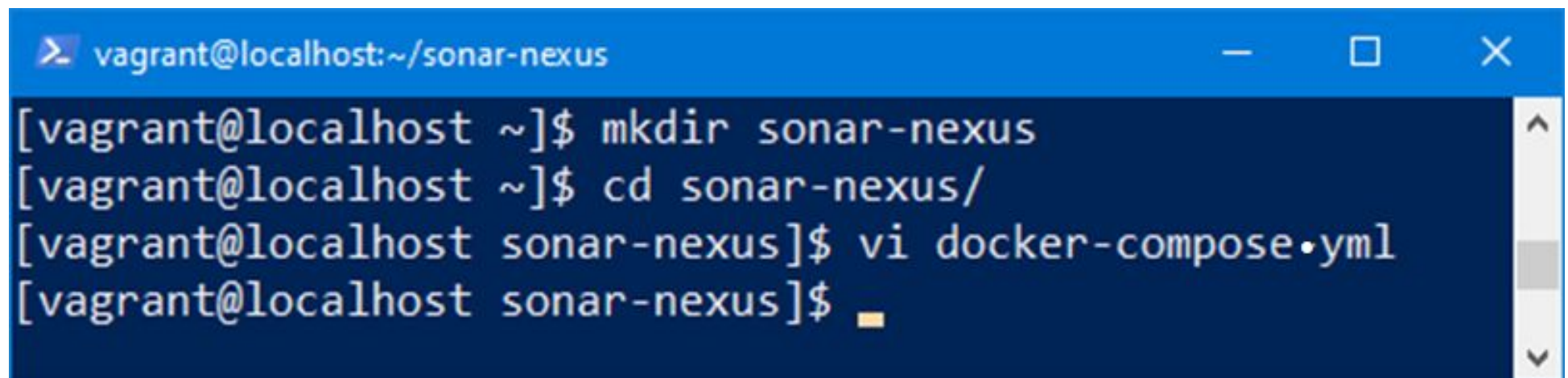
C'est juste pour vous expliquer Docker-Compose.

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

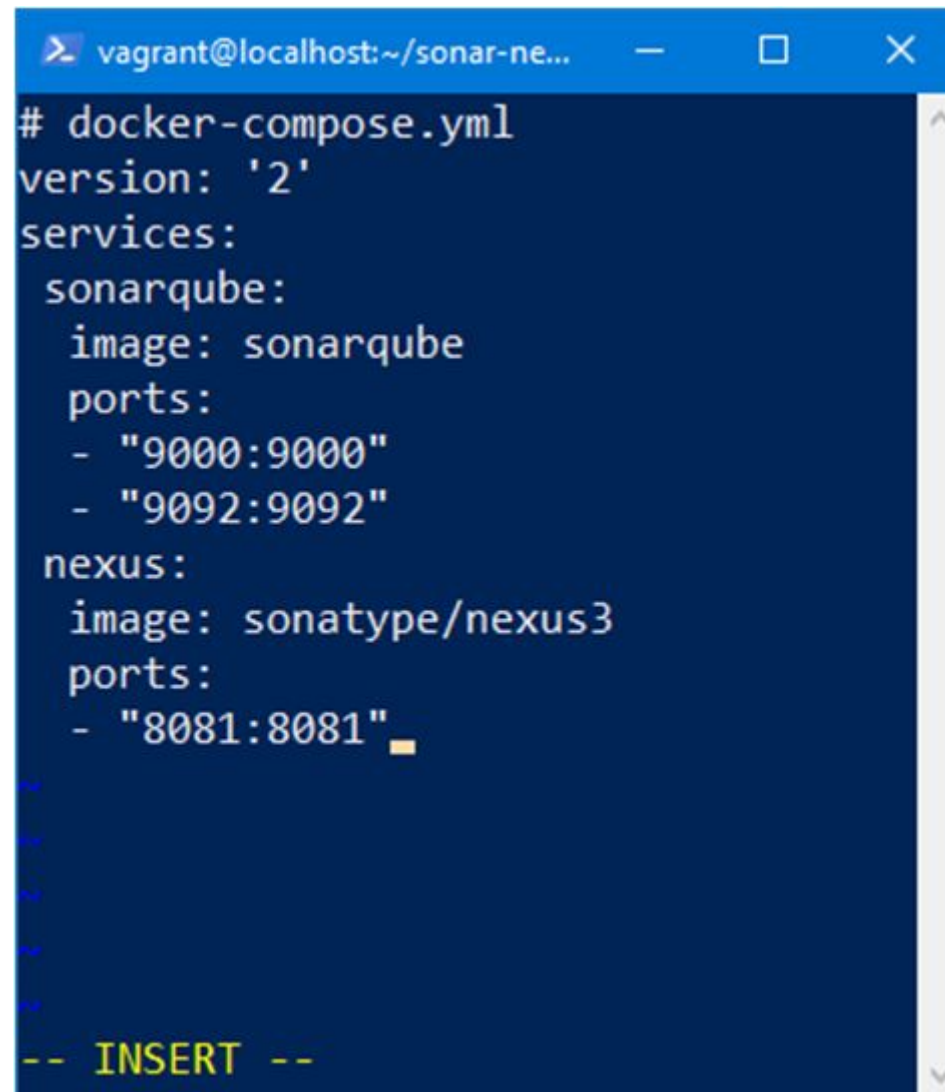
- Pour utiliser « Docker compose », nous allons configurer les deux images 'Sonarqube' et 'Nexus' afin de les lancer simultanément.
- J'ai créé un dossier nommé «sonar-nexus» et ensuite je suis allé dans ce répertoire
- J'ai créé le fichier YAML en utilisant l'éditeur de texte vi :



```
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 »:

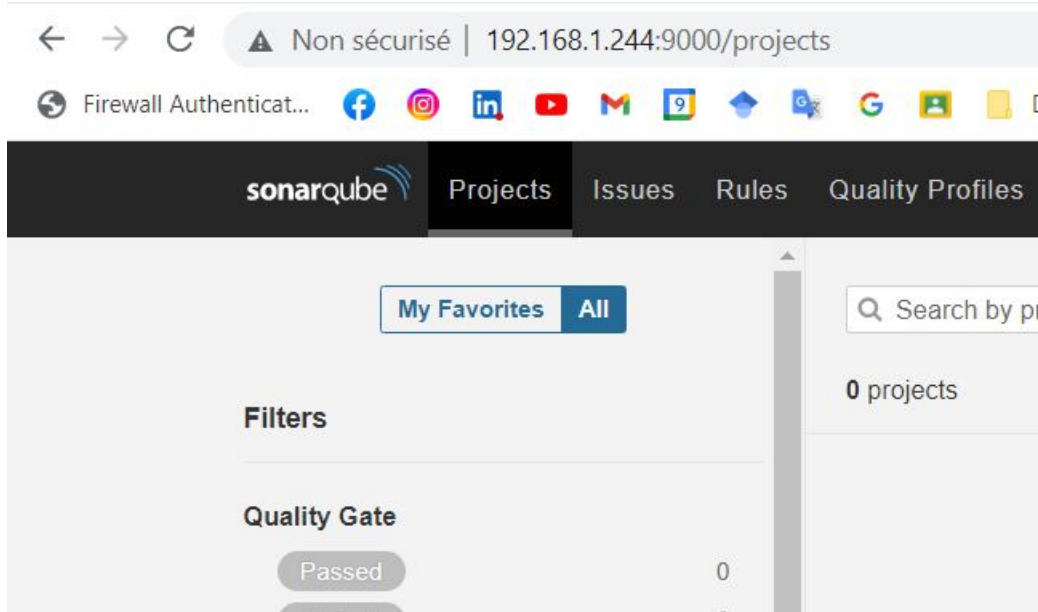
A screenshot of a terminal window with a blue title bar. The title bar text is 'vagrant@localhost: ~/sonar-ne...'. The terminal content is a YAML file for Docker Compose. It defines two services: 'sonarqube' and 'nexus'. 'sonarqube' uses the 'sonarqube' image and maps ports 9000 and 9092. 'nexus' uses the 'sonatype/nexus3' image and maps port 8081. At the bottom, there is a comment '-- INSERT --' in yellow. The terminal has a dark blue background and a vertical scrollbar on the right.

```
> 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 --
```

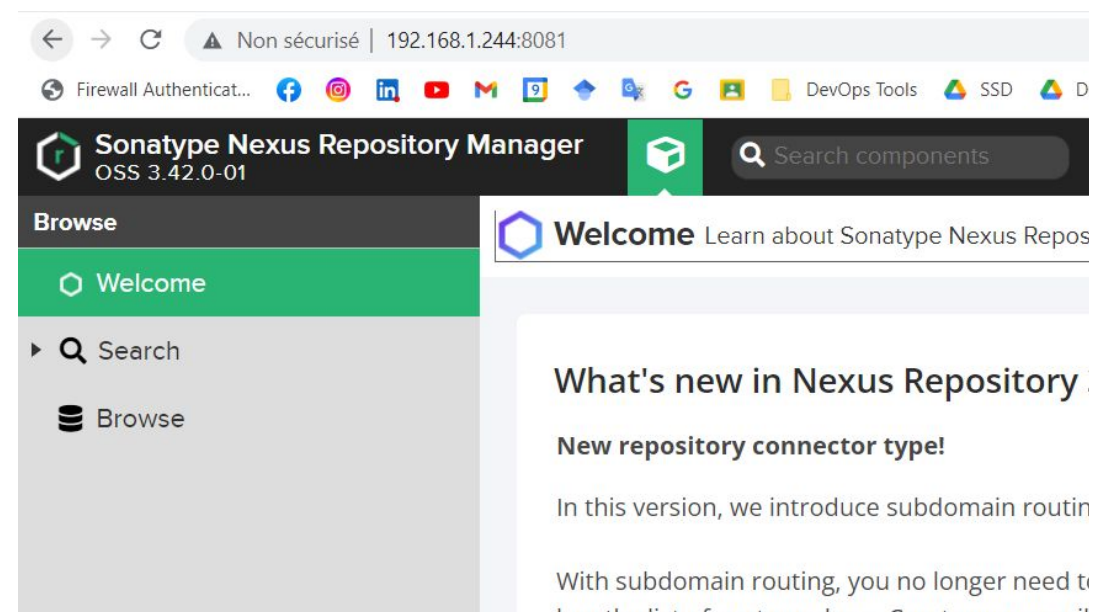
Docker Compose - Exécuter des conteneurs ensemble

- La commande suivante pour créer les conteneurs

```
vagrant@vagrant:~/sonar-nexus$ docker compose up -d
[+] Running 3/3
 ✓ Network sonar-nexus_default          Creat...          0.2s
 ✓ Container sonar-nexus-nexus-1        Cre...           0.1s
 ✓ Container sonar-nexus-sonarqube-1    Created          0.1s
Attaching to sonar-nexus-nexus-1, sonar-nexus-sonarqube-1
sonar-nexus-sonarqube-1 | 2023.10.22 02:50:16 INFO  app[][o.s.a.AppFileSystem] Cle
```



SonarQube



Nexus

Docker Compose - Exécuter des conteneurs ensemble

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

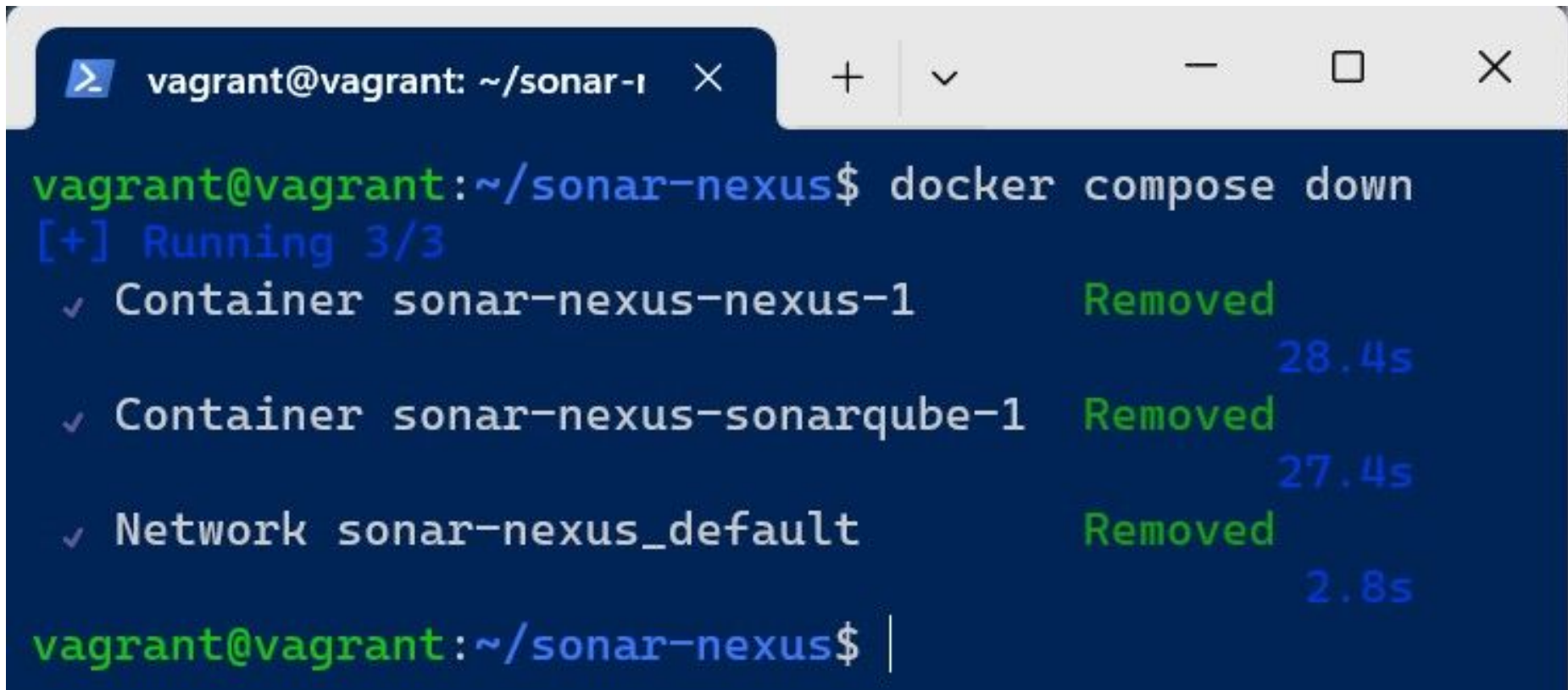
docker compose logs

```
vagrant@vagrant:~/sonar-nexus$ docker compose logs
sonar-nexus-sonarqube-1 | 2023.10.22 02:50:16 INFO app[][o.s.a.AppFileSystem] Cleaning or creating
temp directory /opt/sonarqube/temp
sonar-nexus-nexus-1 | 2023-10-22 02:50:28,488+0000 INFO [FelixStartLevel] *SYSTEM org.sonatype
.nexus.pax.logging.NexusLogActivator - start
sonar-nexus-nexus-1 | 2023-10-22 02:50:30,351+0000 INFO [FelixStartLevel] *SYSTEM org.sonatype
.nexus.features.internal.FeaturesWrapper - Fast FeaturesService starting
sonar-nexus-nexus-1 | 2023-10-22 02:50:33,933+0000 INFO [FelixStartLevel] *SYSTEM ROOT - bundl
e org.apache.felix.scr:2.1.30 (57) Starting with globalExtender setting: false
sonar-nexus-nexus-1 | 2023-10-22 02:50:33,985+0000 INFO [FelixStartLevel] *SYSTEM ROOT - bundl
e org.apache.felix.scr:2.1.30 (57) Version = 2.1.30
sonar-nexus-nexus-1 | 2023-10-22 02:50:35,181+0000 WARN [FelixStartLevel] *SYSTEM uk.org.lidal
ia.sysoutslf4j.context.SysOutOverSLF4JInitialiser - Your logging framework class org.ops4j.pax.loggi
ng.slf4j.Slf4jLogger is not known - if it needs access to the standard println methods on the consol
e you will need to register it by calling registerLoggingSystemPackage
sonar-nexus-nexus-1 | 2023-10-22 02:50:35,186+0000 INFO [FelixStartLevel] *SYSTEM uk.org.lidal
ia.sysoutslf4j.context.SysOutOverSLF4J - Package org.ops4j.pax.logging.slf4j registered; all classes
within it or subpackages of it will be allowed to print to System.out and System.err
```

Docker Compose - Exécuter des conteneurs ensemble

- Comment supprimer un docker compose ?

docker compose down

A terminal window with a dark blue background and light blue text. The window title bar shows 'vagrant@vagrant: ~/sonar-1' and standard window controls. The terminal output shows the command 'docker compose down' being executed. It reports that 3/3 components are running and then lists the removal of three items: 'Container sonar-nexus-nexus-1' (28.4s), 'Container sonar-nexus-sonarqube-1' (27.4s), and 'Network sonar-nexus_default' (2.8s). The prompt returns to 'vagrant@vagrant: ~/sonar-nexus\$'.

```
vagrant@vagrant: ~/sonar-1  
vagrant@vagrant:~/sonar-nexus$ docker compose down  
[+] Running 3/3  
✓ Container sonar-nexus-nexus-1      Removed      28.4s  
✓ Container sonar-nexus-sonarqube-1  Removed      27.4s  
✓ Network sonar-nexus_default        Removed      2.8s  
vagrant@vagrant:~/sonar-nexus$
```

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: '3.8'
services:
  nexus:
    image: sonatype/nexus3
    ports:
      - "8082:8081"
    volumes:
      - nexus-data:/nexus-data
```

L'espace de stockage réservé pour Nexus

```
sonarqube:
  image: sonarqube
  ports:
    - "9000:9000"
    - "9092:9092"
  volumes:
    - SonarQube_data:/opt/SonarQube/data
    - SonarQube_extensions:/opt/SonarQube/extensions
    - SonarQube_logs:/opt/SonarQube/logs
```

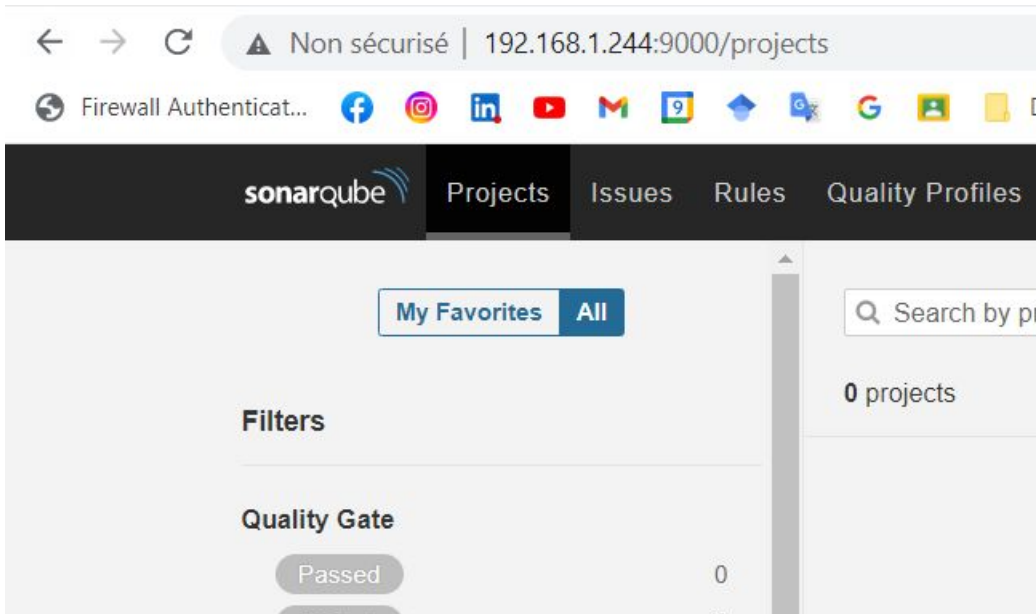
Les espaces de stockage réservés pour 'SonarQube'

```
volumes:
  nexus-data:
  SonarQube_data:
  SonarQube_extensions:
  SonarQube_logs:
```

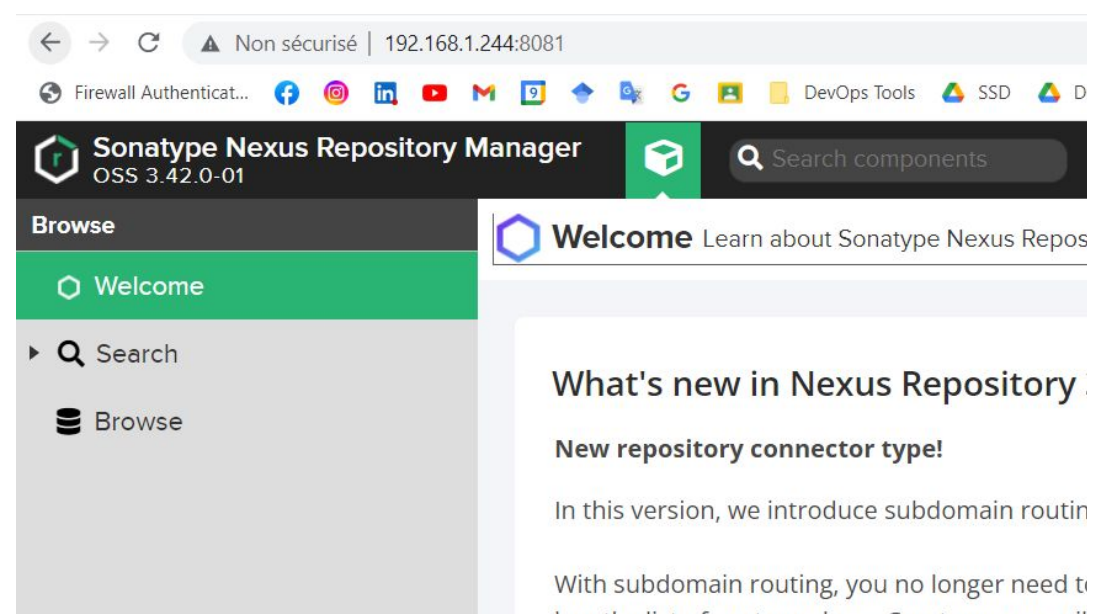
Déclaration des espaces de stockage

Docker Volume – Configuration dans Docker-Compose

```
vagrant@vagrant:~/sonar-nexus$ docker compose up
[+] Running 3/3
 ✓ Network sonar-nexus_default          Creat...          0.2s
 ✓ Container sonar-nexus-nexus-1        Cre...           0.1s
 ✓ Container sonar-nexus-sonarqube-1     Created           0.1s
Attaching to sonar-nexus-nexus-1, sonar-nexus-sonarqube-1
sonar-nexus-sonarqube-1 | 2023.10.22 02:50:16 INFO  app[][o.s.a.AppFileSystem] Cle
aning or creating temp directory /opt/sonarqube/temp
```



SonarQube



Nexus

Projet DevOps : Docker et Jenkins

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

Exemple sur le projet timesheet-devops à adapter à votre projet. Mettez la bonne image java. Choisissez de Docker Hub 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...

EXPOSE ...

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 achat (voir cours 2- Docker):

docker build -t timesheet-devops:1.0 .

(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 achat (voir cours 2- Docker):

docker login -u mouradhassini -p pwd

docker push mouradjassini/timesheetdevops:1.0

(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 par exemple) pour faire tourner votre application achat (Backend avec une base de données MySQL (inspirez vous de l'exemple ci-dessus). Deux 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 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 :



The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer displays the project structure for 'timesheet-devops' located at 'C:\Work\worksp'. The 'src/main/resources' directory is expanded, and 'application.properties' is selected. The code editor shows the following content:

```
1 #http://localhost:8082/timesheet-devops/...
2 #http://localhost: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
```


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.

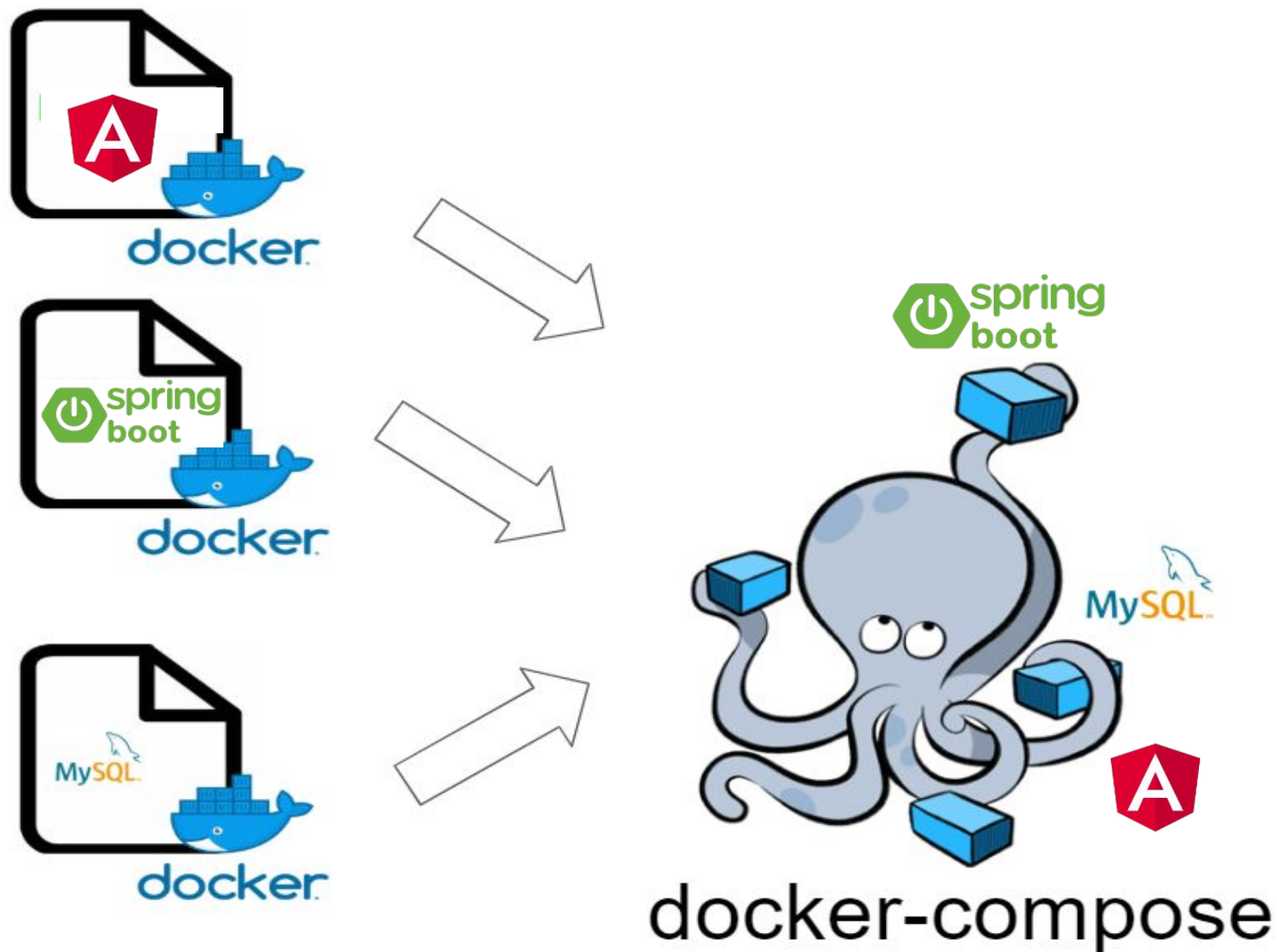
Projet DevOps : Docker et Jenkins

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 et ne donne pas la main pour continuer avec les étapes suivantes du pipeline (grafana/prometheus qu'on verra la semaine prochaine, mail récapitulatif, ...)?

Projet DevOps : Docker et Jenkins



Docker et Jenkins

- Indication : Pour automatiser la création des images « Docker » dans « Jenkins », vous pouvez Installer le plugin « Docker Pipeline »:

Installation/Mise à jour des Plugins

Préparation

- Vérification de la connexion à internet
- Vérification de la connexion à jenkins-ci.org
- Succès

Authentication Tokens API



Succès

Docker Commons



En cours d'installation



Docker Pipeline



En cours

Loading plugin extensions



Pending



[Revenir en haut de la page](#)

(vous pouvez commencer à utiliser les plugins installés dès maintenant)

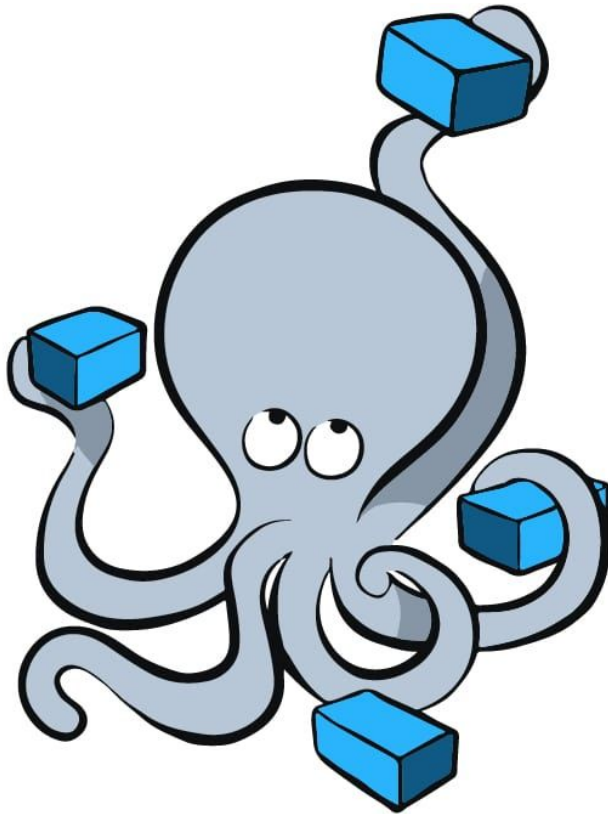


☐ Redémarrer Jenkins quand l'installation est terminée et qu'aucun job n'est en cours

Docker et Jenkins

- Indication : Lire le lien suivant pour le pipeline CD : **cliquer ici**

Docker Compose



docker
Compose