

SONARQUBE



sonarqube 

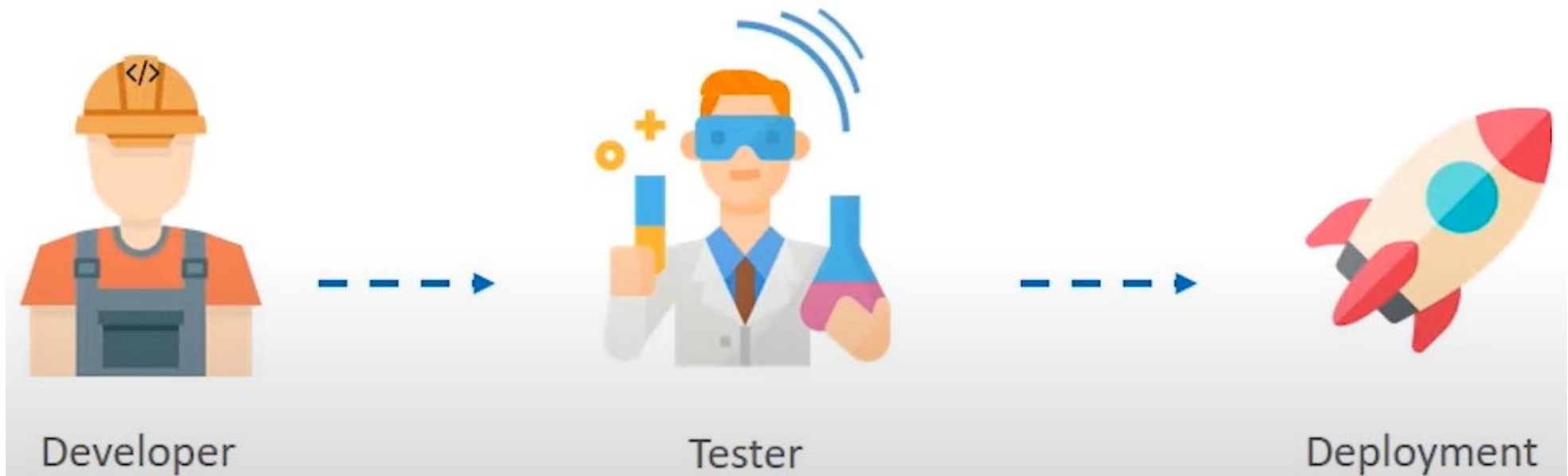
The word 'sonarqube' is written in a large, bold, lowercase sans-serif font. To the right of the text is the SonarQube icon, which consists of three concentric blue curved lines that resemble a stylized 'S' or a sonar wave.

PLAN DU COURS

- Tests dynamiques et Tests statiques
- C'est quoi SONARQUBE
- Caractéristiques de SONARQUBE
- Installation de SONARQUBE (à partir d'une Image Docker)
- Utilisation de Sonarqube
- Compréhension des résultats d'analyse de code

Tests dynamiques et Tests statiques

- Les **tests** font partie de cycle de vie du développement d'une application donnée.
- Les tests visent à s'assurer que le code qui sera déployé est de **bonne qualité, sécurisé** et ne présente pas de **bugs** (environ **30%** du temps de développement doit être consacré aux tests).

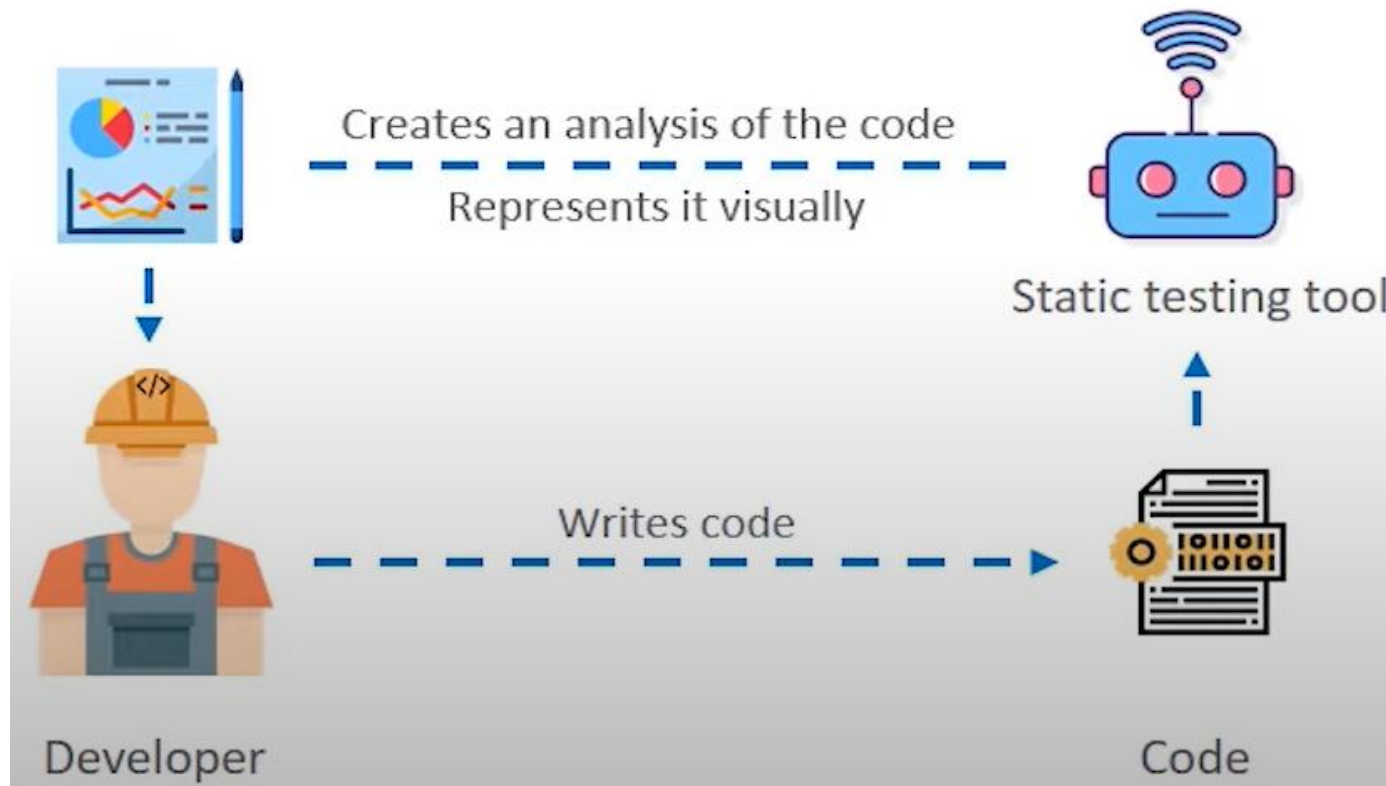


Tests dynamiques et Tests statiques

- Les **tests dynamiques** : Ces tests sont faits alors que l'application tourne, pour détecter les **dysfonctionnements** (fonctionnalités mal implémentées, DB inaccessible, ...) : Tests Unitaires (JUnit par exemple).
- Les **tests statiques** : Ces tests sont faits sur le code source, avant de l'exécuter. Il s'agit d'analyser le code pour détecter les écarts aux **bonnes pratiques de développement** (absence de logs, absence de commentaires. SONARQUBE fait ce type de tests.

SONARQUBE

- **SONARQUBE** est un outil de test statique, open source, utilisé pour analyser la qualité du code source, selon des règles prédéfinies. Il permet donc l'inspection en continue de la qualité de votre code (Code Review automatique).

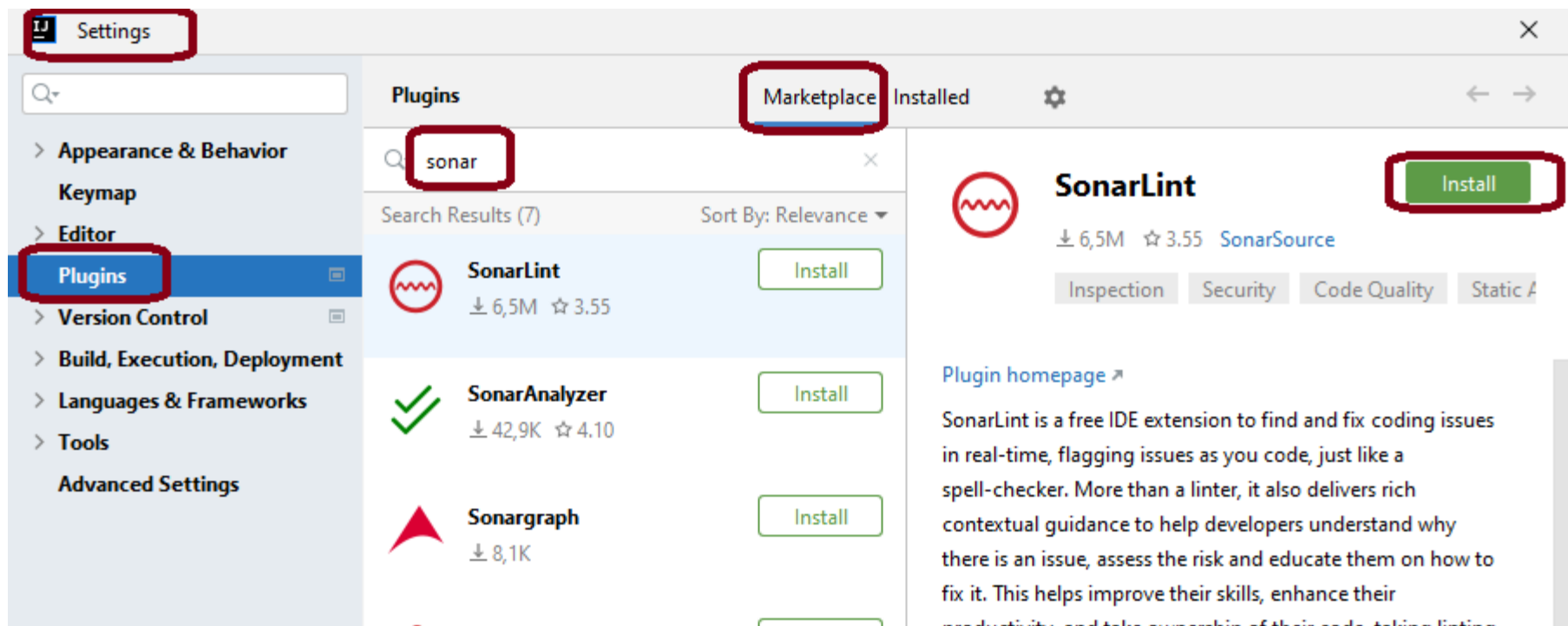


Caractéristiques de SONARQUBE

- SONARQUBE peut être utilisé avec une vingtaine de langages (Java, .Net (C#), Python, PHP, Cobol, JavaScript, ...)
- Il permet de détecter **les défauts de codage** (code jamais utilisé, dupliqué, sans commentaires, sans tests unitaires, sans gestion d'exception, non sécurisé,).
- Il nous permet de choisir **les règles à activer** lors de l'analyse de notre code.
- SONARQUBE peut être installé en mode **standalone**, ou en tant que **plugin** intégré à un IDE comme **STS** (Eclipse).

PLUGIN SONARQUBE pour IDE

- Dans IntelliJ, aller Settings -> Plugins -> MarketPlace, chercher « sonar », installer le plugin « **SonarLint** », accepter l'installation, accepter de redémarrer IntelliJ à la fin de l'installation.



PLUGIN SONARQUBE pour IntelliJ

- Ouvrir un de vos projets sur IntelliJ, bouton droit et choisir « Sonar Lint » -> Analyze, voir le résultat :



PLUGIN SONARQUBE pour IntelliJ

- Résultat :

The screenshot displays the IntelliJ IDEA interface with the SonarLint plugin active. The main editor shows the `CategorieProduitController.java` file. The SonarLint panel at the bottom left indicates that 72 issues were found across 18 files, with 5 issues specifically in `CategorieProduitController.java`. The issues listed are:

- (37, 75) Replace this persistent entity with a simple POJO or DTO object.
- (52, 78) Replace this persistent entity with a simple POJO or DTO object.
- (42, 4) This block of commented-out lines of code should be removed.
- (23, 32) Immediately return this expression instead of assigning it to the variable.
- (38, 38) Immediately return this expression instead of assigning it to the variable.

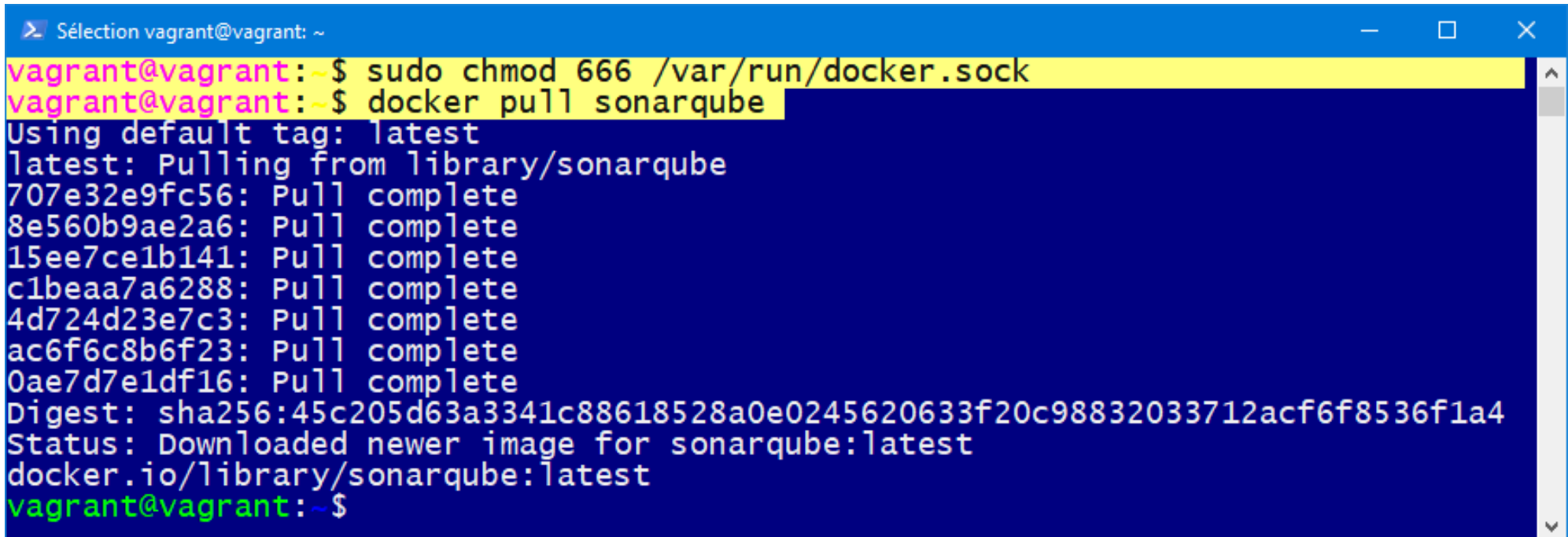
The right panel shows the details for the selected issue (23, 32):

- Rule:** Local variables should not be declared and then immediately returned.
- Locations:** java:S1488
- Why is this an issue?** Declaring a variable only to immediately return or throw it is a bad practice. Some developers argue that the practice improves code readability, because it enables them to explicitly name what is being returned. However, this variable is an internal implementation detail that is not exposed to the callers of the method.

The bottom status bar shows the SonarLint icon and the text "Analysis of 63 files done 2 minutes ago".

SONARQUBE : INSTALLATION

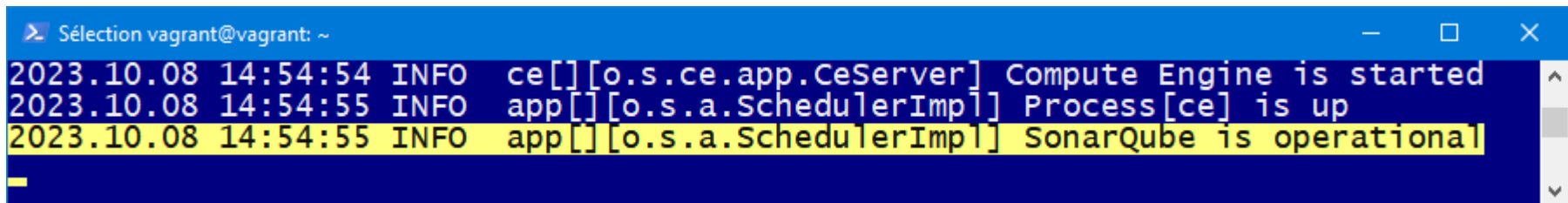
- Connectez-vous à votre VM Ubuntu et lancer un client ssh (lancer Virtual Box, lancer un Powershell : `vagrant up`, `vagrant ssh`).
- (faites un **chmod** auparavant pour éviter les problèmes de droits d'accès) :

A terminal window titled 'Sélection vagrant@vagrant: ~' with standard window controls. The terminal shows two commands being executed: 'sudo chmod 666 /var/run/docker.sock' and 'docker pull sonarqube'. The output of the second command shows the pulling of the 'latest' tag from the 'library/sonarqube' repository, with seven layers being pulled successfully. The digest is 'sha256:45c205d63a3341c88618528a0e0245620633f20c98832033712acf6f8536f1a4'. The status indicates a newer image was downloaded.

```
vagrant@vagrant:~$ sudo chmod 666 /var/run/docker.sock
vagrant@vagrant:~$ docker pull sonarqube
Using default tag: latest
latest: Pulling from library/sonarqube
707e32e9fc56: Pull complete
8e560b9ae2a6: Pull complete
15ee7ce1b141: Pull complete
c1beaa7a6288: Pull complete
4d724d23e7c3: Pull complete
ac6f6c8b6f23: Pull complete
0ae7d7e1df16: Pull complete
Digest: sha256:45c205d63a3341c88618528a0e0245620633f20c98832033712acf6f8536f1a4
Status: Downloaded newer image for sonarqube:latest
docker.io/library/sonarqube:latest
vagrant@vagrant:~$
```

SONARQUBE : LANCEMENT

- Lancer Sonarqube : `docker run -p 9000:9000 sonarqube`
- (-p pour exposer le port 9000 et pour que sonarqube soit accessible de l'extérieur)
- Cela prend du temps :



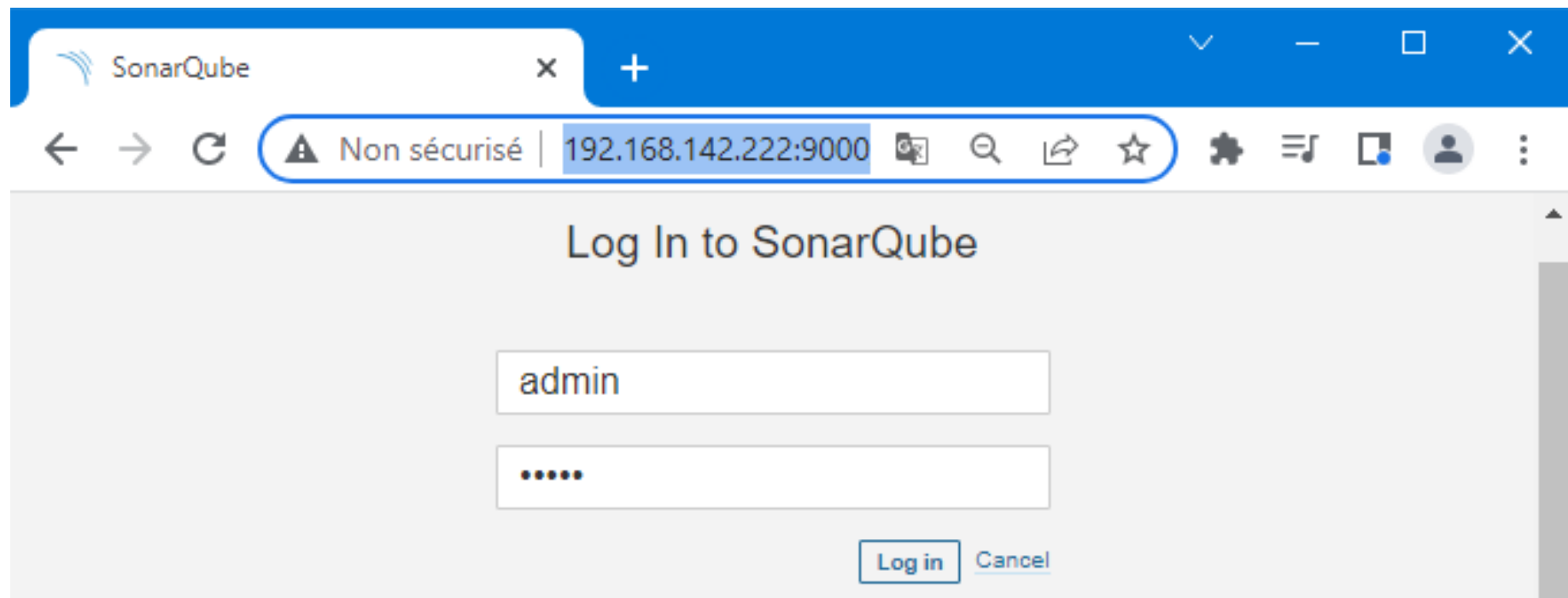
A screenshot of a terminal window titled "Sélection vagrant@vagrant: ~". The terminal displays three log lines from SonarQube. The first line is "2023.10.08 14:54:54 INFO ce[][o.s.ce.app.CeServer] Compute Engine is started". The second line is "2023.10.08 14:54:55 INFO app[][o.s.a.SchedulerImp] Process[ce] is up". The third line is "2023.10.08 14:54:55 INFO app[][o.s.a.SchedulerImp] SonarQube is operational", which is highlighted in yellow. The terminal has a dark blue background and a light blue title bar.

```
Sélection vagrant@vagrant: ~
2023.10.08 14:54:54 INFO ce[][o.s.ce.app.CeServer] Compute Engine is started
2023.10.08 14:54:55 INFO app[][o.s.a.SchedulerImp] Process[ce] is up
2023.10.08 14:54:55 INFO app[][o.s.a.SchedulerImp] SonarQube is operational
```

- C'est possible de lancer Sonar en background (option -d ou un docker start sur le conteneur).

SONARQUBE : UTILISATION

- Sur votre machine Windows, aller à l'url `http://<ip-vm>:9000` et se connecter avec `admin/admin` :



- Changer le mot de passe à sonar par exemple (`admin/sonar`).

SONARQUBE : UTILISATION

- **Attention** : Si vous arrêtez le conteneur sonarqube (CTRL S ou en arrêtant la VM), il ne faut pas lancer un `docker run` sur l'image sonarqube, car cela créera un nouveau conteneur. Il faut juste faire :

`docker start id-conteneur-sonarqube`

- Et attendre que le temps que tout se lance (Elasticsearch, Sonarqube, ...) :

```
vagrant@vagrant:~$ sudo chmod 666 /var/run/docker.sock
vagrant@vagrant:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND
f87df4171155   sonarqube                          "/opt/sonarqube/dock...
2ed331260ff0   hello-world                        "/hello"
313002c32776   mouradhassini/alpine:1.0.0        "/bin/sh"
9f903569b63c   postgres                          "docker-entrypoint.s...
cf063f7757d8   hello-world                        "/hello"
vagrant@vagrant:~$ docker start f87df
f87df
vagrant@vagrant:~$ _
```

SONARQUBE : UTILISATION

- Puisque nous n'avons pas de projets à analyser sur notre VM, nous allons utiliser Jenkins, pour récupérer un projet de Git, puis l'analyser avec Sonarqube :
- Se connecter à **Jenkins** via l'url `http://<ip-vm>:8080`
- Sur le **pipeline** Jenkins déjà créé, récupérer le code de votre projet de Github en ajoutant le stage suivant dans le script Groovy (mettez l'URL de votre projet). **Normalement, cette étape a déjà été faite la semaine dernière :**

```
stage ('GIT') {  
    steps {  
        echo "Getting Project from Git";  
        .....  
    }  
}
```

SONARQUBE : UTILISATION

- Sur le même pipeline, lancer les commandes Maven **clean** et **compile** pour compiler le code de votre projet récupéré de Git (sh « ... »).
Normalement, cette étape aussi a déjà été faite la semaine dernière :

```
stage('MVN CLEAN') {  
    steps {  
        sh '.....'  
    }  
}
```

```
stage('MVN COMPILE') {  
    steps {  
        .....  
    }  
}
```

SONARQUBE : UTILISATION

- Sur le même pipeline, lancer la commandes Maven d'analyse de code **sonar:sonar** pour analyser la qualité de votre code et envoyer le rapport au serveur Sonarqube (Mettez le mot de passe de votre Sonarqube) :

```
stage('MVN SONARQUBE') {  
    steps {  
        .....  
    }  
}
```

- **Lancer le Job via Jenkins :**

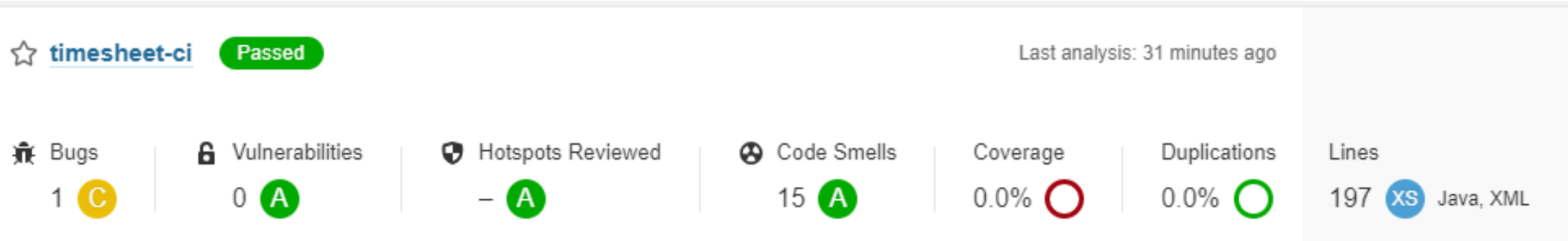
SONARQUBE : UTILISATION Dans JENKINS

The screenshot shows the Jenkins web interface for a job named 'Mohamed_ASKRI_5DS2'. The browser address bar indicates the URL '192.168.94.222:8080/job/Mohamed_ASKRI_5DS2/'. The interface includes a sidebar with navigation options: 'Full Stage View', 'Renommer', and 'Pipeline Syntax'. Below these is a 'Historique des builds' section with a 'tendance' dropdown and a 'Filter builds...' search bar. The main content area displays a table of build stages for build #11, which was executed on October 8, 2023, at 16:06. The stages are 'Git', 'Maven Clean & Compile', and 'Sonarqube'. The 'Sonarqube' stage shows a duration of 36s. The table also includes a summary row for the entire build, showing an average stage time of 2s for Git, 30s for Maven, and 50s for Sonarqube, with a total average full run time of approximately 1 minute.

Average stage times: (Average <u>full</u> run time: ~1min 0s)		Git	Maven Clean & Compile	Sonarqube
#11	oct. 08 17:06 No Changes	2s	30s	50s
#11	oct. 08 17:06 No Changes	1s	7s	36s
#10	oct. 08 17:06 No Changes	1s	7s	1min 4s

SONARQUBE : ANALYSE DES RÉSULTATS

- Aller dans <http://<ip-vm>:9000> et regarder le résultat de l'analyse de votre projet :



- Sur l'interface ci-dessus, il suffit de cliquer sur le projet « timesheet » pour accéder aux détails de toute cette analyse :
- Sonarqube détecte automatiquement le **langage (Java + XML)** dans notre cas)
- Sonarqube vérifie si le développeur a mis du **code dupliqué** dans plusieurs endroit (source d'erreur) : 0% dans notre cas




SONARQUBE : ANALYSE DES RÉSULTATS

- **Bug** : Un code qui peut engendrer un mauvais comportement de l'application (un null pointer exception par exemple), dans notre cas :

```
@Override
public User retrieveUser(String id) {
    l.info("in retrieveUser id = " + id);
    //User u = userRepository.findById(Long.parseLong(id)).orElse(null);
    //int i = 1/0;
    User u = userRepository.findById(Long.parseLong(id)).get();

    l.info("user returned : " + u);
    return u;
}
```

Call "Optional#isPresent()" before accessing the value. Why is this an issue?

 Bug ▾  Major ▾  Open ▾ Not assigned ▾ 10min effort [Comment](#)

```
l.info("user returned : " + u);
return u;
}
```

- **Vulnerability** : Faille de sécurité dans notre code.
- **Hotspots Reviewed** : Code à revoir pour être sûr que ce n'est pas une faille de sécurité.

SONARQUBE : ANALYSE DES RÉSULTATS

- Sonarqube affiche si le code a été exécuté par les outils de tests (comme JUnit). Sonarqube ne fait pas l'analyse lui-même mais se base sur d'autres outils comme JaCoCo (c'est pour cela qu'on a 0% de **Coverage** puisque JaCoCo n'est pas ajouté à notre projet).
- **Code Smells** : Ce n'est pas un bug, mais c'est un code qui peut retarder l'équipe de développement ou l'équipe de support quand ils essaient de comprendre ou modifier le code (exemple : beaucoup de commentaires ou des imports non utilisés) :

☐ This block of commented-out lines of code should be removed. Why is this an issue?
👤 Code Smell ▼ ⬆ Major ▼ ○ Open ▼ Not assigned ▼ 5min effort Comment

📄 src/.../java/tn/esprit/spring/services/UserServiceImpl.java

☐ Remove this unused import 'javax.transaction.Transactional'. Why is this an issue?
👤 Code Smell ▼ ⬇ Minor ▼ ○ Open ▼ Not assigned ▼ 2min effort Comment

SONARQUBE

Passed

All conditions passed.

New Code

Overall Code


1

 Bugs

Reliability



0

 Vulnerabilities

Security



0

 Security Hotspots 

— Reviewed

Security Review



1h 32min

Debt

15

 Code Smells

Maintainability





0.0%

Coverage on 49 Lines to cover

—

Unit Tests



0.0%

Duplications on 197 Lines

0

Duplicated Blocks