



Une entreprise ordinaire à l'agilité extraordinaire

FORMATION JAVA SPRING ANGULAR

QUALITY CODE & DEVOPS

mohamed.el-babili@fms-ea.com

33+ 628 111 476

Version : 2.5

DMAJ : 03/12/23

Module : DEV-DEVO-001

Quality Code & DevOps

01 Pourquoi et comment réaliser du Code Clean avec les outils Sonar

02 Mise en œuvre de l'intégration & déploiement continu avec Jenkins



SOMMAIRE

- ▶ Pré requis & Versions
- ▶ Introduction
- ▶ Pourquoi la qualité dans le code ?
- ▶ Comment faire du code clean ?
- ▶ Concrètement
- ▶ Historique Sonar
- ▶ La philosophie Sonar
- ▶ Les différents niveaux d'analyse
- ▶ SonarLint
- ▶ Sonar Rules
- ▶ SonarQube
- ▶ Qualités & Mesures
- ▶ DevOps
- ▶ CI/CD avec Jenkins & Docker
- ▶ Pour infos
- ▶ Pour aller plus loin
- ▶ Ressources

Pré requis & Versions

Pré requis pour suivre cette masterclass :

- Niveau avancé sur un langage (Java, Js...)
- Maîtriser terminal + notion de script
- Git/GitHub
- Notion de Test unitaire
- Maven

Liste des versions des applications utilisées

- SonarLint 7.5
- Eclipse 4.25.0 -> 2023
- IntelliJ 2021.3.3 -> 2023
- Maven 3.8.6 -> 3.9.5
- Java 11
- SonarQube 9.6.1 -> 9.9.1
- Jenkins 2.361.1 → 2.426.1
- Docker 20.10.17



Faut pas avoir peur de tomber.

Faut pas voir peur de se tromper.

C'est comme ça qu'on apprend !

Introduction

Nous verrons dans un 1^{er} temps les éléments liés à la qualité dans le code ou code Clean en introduisant les outils Sonar, comment celle-ci va avoir un impact sur la qualité d'une application et comment mesurer celle-ci à l'aide d'outils statistiques, on parlera de métriques logicielles.

Nous aborderons dans un second temps, des activités du DevOps » qui englobe un large éventail de compétences techniques et non techniques.

Nous finirons par des aspects plus large liés au métier d'architecte logiciel ainsi que des perspectives autour du cloud computing (Aws...)

Pourquoi la qualité dans le code ?

Si on **livre du code sans contrôle qualité**, il y a de fortes chances qu'on en paye le prix tôt ou tard :

- Les évolutions deviendront de + en + compliqué
- Cout de maintenance
- Apparition de bugs
- Compliqué à tester
- Failles de sécurité
- Pb de compréhension du code (on passe + de temps à lire du code)

→ Constat : QQ soit les compétences des devs et ingénieurs, la qualité n'est pas tjs au rdv !

C'est quoi la qualité au juste ?

Une **application est dite de qualité** si l'ajout d'une nouvelle fonctionnalité est constante, cad, que la même fonctionnalité ajoutée maintenant ou dans 1 an aura le même cout.

Inversement, **une appli est dite de mauvaise qualité** si l'ajout d'une nouvelle fonctionnalité est exponentiel selon le temps. Si bien qu'elle en devient difficilement maintenable et la question de l'abandonner se pose.

Les raisons sont souvent liées au code mais peuvent aussi provenir de l'architecture, de l'environnement de dev, du langage...

Comment faire du code clean ?

Si on comprend pourquoi il faut faire du code de qualité, on réalise qu'il y a nécessité d'établir des normes d'écritures de code et de les respecter. C'est la démarche de Robert C.Martin dans son livre « **Clean Code** » dont les **règles générales** sont :

- Suivre des conventions reconnues
- Keep it simple stupid (Rester le plus simple possible)
- Règle du boy scout (laissez le camp plus propre qu'à votre arrivé)
- Lors de résolution de pb, toujours chercher et trouver la cause racine
- Principe de moindre surprise (code compréhensible, manipulable par tous)
- DRY : ne vous répétez pas (ne pas dupliquer son code sauf que...)

Voir aussi les règles de design (préférez le polymorphisme aux if/else ou utiliser l'ID...)

Astuces pour la compréhensibilité (rester cohérent, utiliser des noms de variables explicites...)

Règles de nommages, règles relatives aux fonctions, aux commentaires

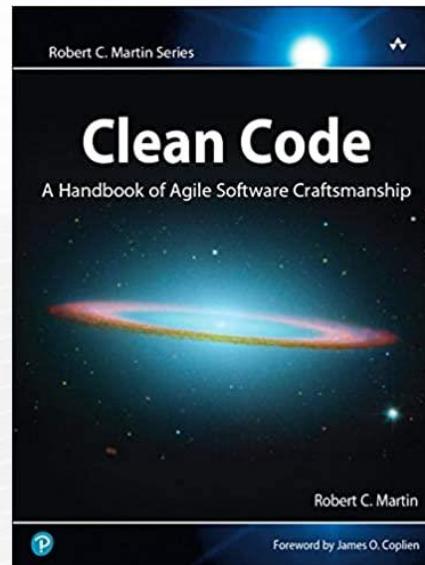
Structure du code source, objets et structure de données

Tests (un concept par test, rapides, indépendants, répétables, auto validant, utiles, lisibles, facile à lancer, code coverage)

Indicateurs de code difficile à maintenir (code smells)

Gestion des erreurs (utiliser des exceptions au lieu de renvoyer des codes d'erreurs...)

Le livre que tous les développeurs doivent lire absolument une fois dans leur vie



Concrètement

Qui & Pourquoi

Les développeurs d'une même équipe ou organisation doivent se mettre d'accord sur l'ensemble des normes de codages à appliquer. En effet, si on souhaite faciliter la lecture du code de chacun, il faut établir ces normes et tous les respecter. Les nouveaux venus seront ainsi incités dès le départ à mettre en œuvres ces bonnes pratiques.

Quoi

Il peut s'agir d'indentation du code, conventions de nommage, bon dosage de commentaires, ou encore des mauvaises pratiques (oubli d'accolade). Ces normes ne peuvent pas remplacer le rôle des TechLead ou Quality Leader qui effectuent le + souvent la revue de code afin de juger de la qualité des algorithmes par ex.

Exceptions

Cependant, celui-ci ne peut pas tout contrôler aussi il peut être intéressant d'utiliser des outils d'analyse de code automatisable, pouvant vérifier jusqu'au byte code, proposant des avertissements du compilateur, informant sur des commentaires TODO laissés à l'abandon, indiquant le nombre de lignes de code, la moyenne de complexité de code, le nombre de lignes par classe, recherchant de potentiel bug ou mauvaises pratiques connues...

COMMENT ?

Il existe de nombreux [outils](#) pour faciliter la prise de conscience des développeurs et l'amorçage de l'amélioration continue visant à progressivement apporter plus de qualité au code des applications. Nous allons présenter l'historique, l'esprit et les principaux outils Sonar.

Historique Sonar

2009 : Sonar par Sonarsource (opensource) [Sonar 3.7 LTS]

- plateforme très orienté Java qui intègre les résultats d'autre linter (Checkstyle, PMD, findbugs)
- principale préoccupation : Qualité & Sécurité d'ou la nécessité de développer leurs propres outils

2013 : SonarQube 4.0

- développement d'un produit avec leurs analyseurs présent sur un serveur, toujours au départ pour Java suivi progressivement par la couverture d'autre langage
- **Clean as you code** (nettoyer au fur et à mesure que vous codez) & **QualityGate** qui fixe les règles/normes d'écriture de codes : la qualité du code va progressivement s'affiner

2015 : SonarLint

- intégration à l'IDE de l'analyseur qui permet d'informer le dev directement et pas attendre le prochain CI (UX)
 - possibilité de pousser que du code propre à condition de rester vigilant

2017 : SonarCloud

- La solution SonarQube prend du temps à configurer/installer(serveur), de la ressource, nécessité de MAJ
- Il s'agit donc de proposer ce service directement sur le Cloud associé aux autres outils des plateformes devops

2018 : SonarQube 7.9 LTS

- Analyseur de sécurité

2021 : SonarQube 8.9 LTS

- Taint analysis visant à trouver des vulnérabilités dans le code
- Intégration continue

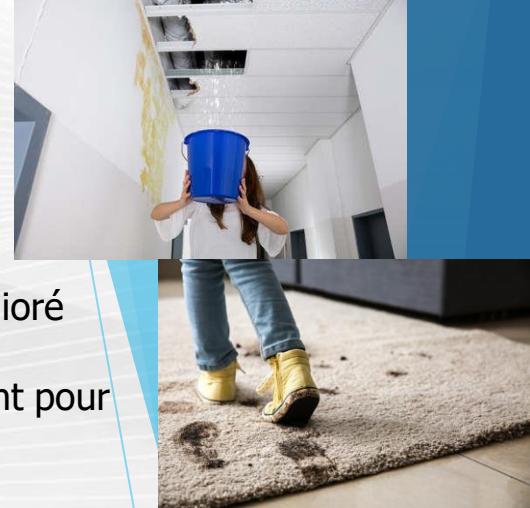
2022 : SonarQube 9.x [LTS 9.9 top]

- DevOps platform intégration (GitHub, GitLab...)

2023 : SonarQube 10.x

Philosophie Sonar

- Il est aberrant de mobiliser 2 devs pour corriger les erreurs de 10 devs en prod
- En revanche, on peut sensibiliser tous les devs pour améliorer la qualité de leur code progressivement
- Du coup, au bout de qq années, la qualité du code de l'application aura été globalement amélioré
- Une documentation permet aux développeurs de comprendre avant de corriger, c'est important pour changer les habitudes
- L'adhésion des devs est donc indispensable, l'outil doit être crédible en remontant de vrai pb, cependant il est perfectible via les forums notamment.
- UX : Expérience Utilisateur



3 piliers chez Sonar :

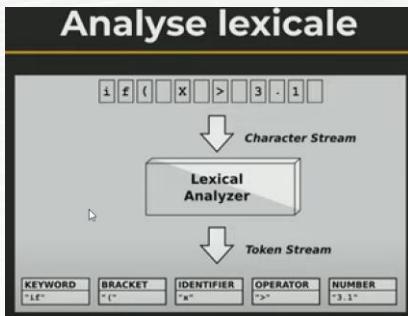
IDE : analyse en temps réel comme un correcteur orthographique

Serveur distant : décoration des demandes de changement pull/merge/change request

Bonnes pratiques : Clean as you code (le dev est responsable du code qu'il délivre)

Important : Sonar n'est pas en mesure de nous dire si une appli est de qualité, en revanche, il peut nous alerter sur ses défauts dans une certaine limite.

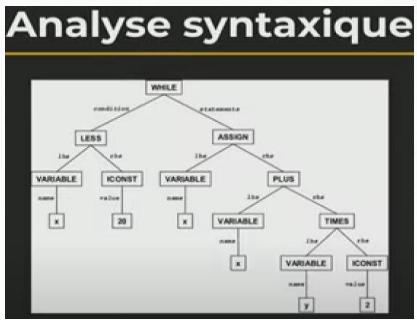
Les différents niveaux d'analyse



Apporte de la coloration

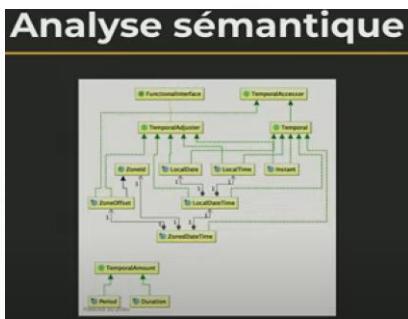
```
@Override  
public List<Article> getCart() {  
    return new ArrayList<Article>(<del>cart.values().</del>);  
}  
  
public double getTotalAmo  
double total = 0;  
for(Article article :  
    total += article.  
}  
return total;  
}
```

Replace the type specification in this constructor call with the diamond operator ("<>").
3 quick fixes available:
Replace with <>
Open description of rule java:S2293
Deactivate rule java:S2293
Press 'F2' for focus



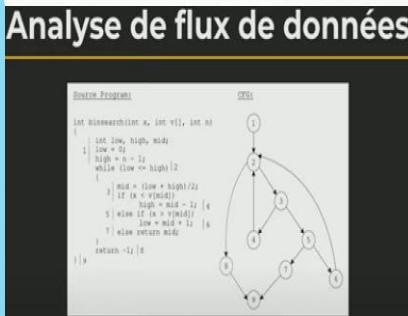
Génère l'arbre de syntaxe

Représente le code sous forme d'arbre sur la base de la structure du langage, on peut analyser ici les incohérences (code dupliqué, branche identique...)
Selon les langages, ils vont déclarés des types, des fonctions ou méthodes, structures de flux de données.



« arbre » sémantique

À partir des arbres de syntaxe, on ajoute des infos sémantiques. En effet, en faisant la résolution des symboles et en attachant des flux sémantiques, on peut dès lors déjà commencé à trouver des bugs (contrainte de nullabilité, bornes sur des types)



Contrôle flow graphe

Il s'agit d'observer comment s'exécute le code à l'aide d'un CFG pour représenter tous les chemins possibles dans un code. Permet de voir par ex si un bout de code n'est jamais appelé.

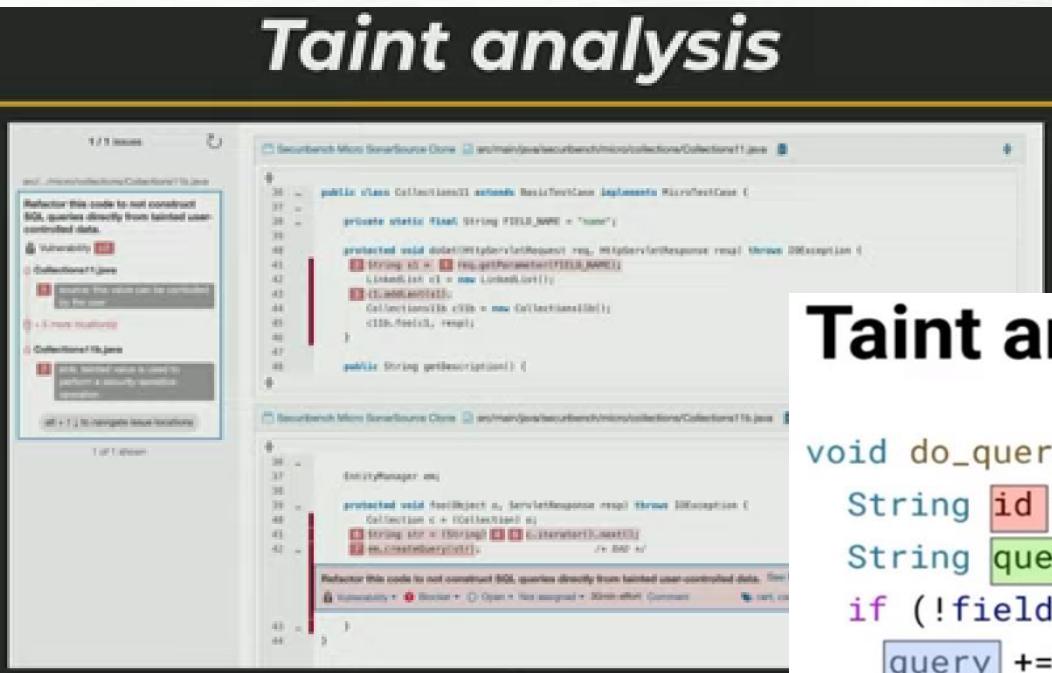
Analyse de sécurité

Toutes les étapes précédentes nous permettent d'arriver à l'étape dite d'analyse de sécurité.

A partir des contrôles flow graphe, on va essayer de trouver des chemins entre, des sources de données (par ex des paramètres http, variables d'environnement, fichiers...) potentiellement contrôlées par un attaquant, et l'exécution d'une commande ou une requête sql par exemple.

Ce sont des failles de vulnérabilité !

Taint analysis



Taint analysis: Un simple exemple

```
void do_query(String fields) {  
    String id = source();  
    String query = "SELECT ";  
    if (!fields.empty()) {  
        query += fields;  
    } else {  
        query += "name";  
    }  
    query += " FROM u WHERE id = " + id;  
    db.query(query);  
}
```

Source: input
possiblement malicieux

Sink: opération sensible

SonarLint

- On va commencer par SonarLint car il s'agit ici de pouvoir produire un maximum de code propre sans avoir à attendre le retour de l'intégration continue.
- Une fois le plug in installé dans votre Ide préféré, on a le retour direct quand on ouvre le fichier.
- De plus, il propose parfois un fix (solution)
- De plus, on peut pousser quasiment que du code propre
- Le [forum](#) est notamment là pour remonter les remarques des utilisateurs afin d'améliorer les outils

The screenshot shows the SonarLint integration within an IDE. The main window displays Java code for a JwtAuthenticationFilter class. Several code review annotations are visible, such as a yellow circle with a question mark at line 51, indicating a potential issue. The code itself handles JWT token creation and response headers.

Below the code editor, the SonarLint interface is shown. It includes a sidebar with project navigation and a bottom panel for viewing issues. The bottom panel shows:

- Found 4 issues in 1 file
- JwtAuthenticationFilter.java (4 issues):
 - (44, 10) This block of commented-out lines of code should be removed. few seconds ago
 - (18, 7) Remove this unused import 'java.util.HashMap'.
 - (19, 7) Remove this unused import 'java.util.Map'.
 - (51, 96) Replace this lambda with method reference 'GrantedAuthority::getAuthority'.
- Rule: Lambdas should be replaced with method references (Code smell, Minor, java:S1612)
- Description: Method/constructor references are commonly agreed to be, most of the time, more compact and readable than using lambdas, and are therefore preferred.
- Note: In some rare cases, when it is not clear from the context what kind of function is being described and reference would not increase the clarity, it might be fine to keep the lambda.
- Similarity: null checks can be replaced with references to the Objects::isNull and Objects::nonNull methods, casts can be replaced with SomeClass.class::cast and instanceof can be replaced with SomeClass.class::isInstance.
- Note: This rule is automatically disabled when the project's sonar.java.source is lower than 8.
- Noncompliant Code Example:

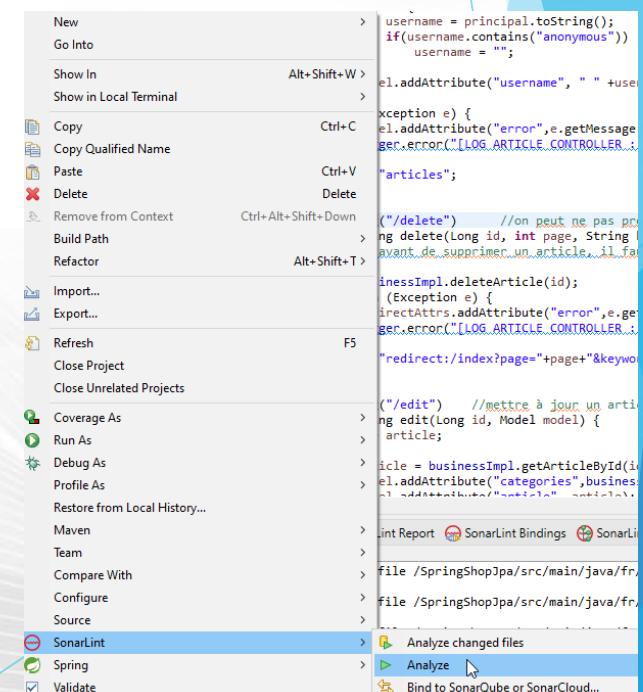
```
class A {  
    void process(List<A> list) {  
        list.stream()  
            .filter(a -> a instanceof B)  
            .map(a -> (B) a)  
            .map(b -> b.<String>.getObject())  
            .forEach(b -> { System.out.println(b); });  
    }  
}
```

SonarLint & Eclipse

The screenshot shows the Eclipse IDE interface. On the left, there is a code editor window displaying Java code for an ArticleController. The code includes annotations like @GetMapping and @PostMapping, and various Java statements. On the right, a 'SonarLint' dialog box is open, containing settings such as 'Run SonarLint automatically' and a link to bind the project to SonarQube/SonarCloud. Below the code editor, the Eclipse toolbar has several SonarLint-related icons: SonarLint Report, SonarLint Bindings, SonarLint Issue Locations, SonarLint On-The-Fly, SonarLint Security Hotspots, SonarLint Taint Vulnerabilities, and SonarLint Rule Description.

Par défaut, SonarLint s'exécute automatiquement à chaque ouverture et(ou) modification de fichier, la console affiche le résultat de l'analyse de code avec un nombre d'issue(s) ou problèmes

The screenshot shows the Eclipse Console window. It displays the output of the SonarLint analysis for the 'SpringShopJpa' project. The console logs show the processing of 44 files, finding 9 issues in ArticleController.java and 4 issues in CartController.java. It also shows the analysis of CategoryController.java and the final summary: 'Found 81 issue(s)'.



Il est possible par le clic droit sur le projet ou un fichier de (re)lancer l'analyse, par ex sur le projet dans sa totalité, cela prendra plus de temps et nous obtenons un nombre d'issues conséquent comme ci-dessus

SonarLint Report & Rule description

Une fois l'analyse fini, le rapport permet d'identifier le fichier concerné, la description du problème, des images décrivant le type et la gravité du problème

Resource	Date	Description
CartController.java		⚠️ ⓘ Complete the task associated to this TODO comment.
CartController.java		⚠️ ⓘ Complete the task associated to this TODO comment.
CartController.java		⚠️ ⓘ Call "Optional#isPresent()" before accessing the value.
CartController.java		⚠️ ⓘ Replace this persistent entity with a simple POJO or DTO object.
CategoryController.java		⚠️ ⓘ Call "Optional#isPresent()" before accessing the value.
Customer.java		⚠️ ⓘ This block of commented-out lines of code should be removed.
IBusiness.java		⚠️ ⓘ Define and throw a dedicated exception instead of using a generic one.
IBusiness.java		⚠️ ⓘ Define and throw a dedicated exception instead of using a generic one.
IBusiness.java		⚠️ ⓘ Define and throw a dedicated exception instead of using a generic one.
IBusiness.java		⚠️ ⓘ Define and throw a dedicated exception instead of using a generic one.
IBusiness.java		⚠️ ⓘ Define and throw a dedicated exception instead of using a generic one.
IBusiness.java		⚠️ ⓘ Define and throw a dedicated exception instead of using a generic one.
IBusiness.java		⚠️ ⓘ Define and throw a dedicated exception instead of using a generic one.
IBusiness.java		⚠️ ⓘ Define and throw a dedicated exception instead of using a generic one.
IBusiness.java		⚠️ ⓘ Define and throw a dedicated exception instead of using a generic one.
IBusinessImpl.java	4 hours ago	⚠️ ⓘ Replace the type specification in this constructor call with the diamond operator ("<>").
IBusinessImpl.java		⚠️ ⓘ Complete the task associated to this TODO comment.
IBusinessImpl.java		⚠️ ⓘ Remove this method and declare a constant for this value.
IBusinessImpl.java		⚠️ ⓘ Replace the type specification in this constructor call with the diamond operator ("<>").
IBusinessImpl.java		⚠️ ⓘ Replace this if-then-else statement by a single return statement.
IRunningImpl.java		⚠️ ⓘ Call "Optional#isPresent()" before accessing the value.

Un clic droit ou sélection de la ligne à scruter permet de voir la description du problème
Avec un exemple de code non conforme et la solution afin de corriger le problème.

Console SonarLint Report SonarLint Bindings SonarLint Issue Locations SonarLint On-The-Fly SonarLint Security Hotspots SonarLint Taint Vulnerabilities SonarLint Rule Description

Generic exceptions should never be thrown (java:S112)

Code smell Major

Using such generic exceptions as `Error`, `RuntimeException`, `Throwable`, and `Exception` prevents calling methods from handling true, system-generated exceptions differently than application-generated errors.

Noncompliant Code Example

```
public void foo(String bar) throws Throwable { // Noncompliant
    throw new RuntimeException("My Message"); // Noncompliant
}
```

Compliant Solution

```
public void foo(String bar) {
    throw new MyOwnRuntimeException("My Message");
}
```

Exceptions

Generic exceptions in the signatures of overriding methods are ignored, because overriding method has to follow signature of the throw declaration in the superclass. The issue will be raised on superclass declaration of the method (or not part of the analysis).

Règles & résolutions

Issues

While running an analysis, SonarQube raises an issue every time a piece of code breaks a coding rule. The set of coding rules is defined through the associated [Quality Profile](#) for each language in the project.

Issue Types

There are three types of issues:

- Bug** – A coding mistake that can lead to an error or unexpected behavior at runtime.
- Vulnerability** – A point in your code that's open to attack.
- Code Smell** – A maintainability issue that makes your code confusing and difficult to maintain.

Issue Severity

Each issue has one of five severities:

- BLOCKER**
Bug with a high probability to impact the behavior of the application in production: memory leak, unclosed JDBC connection, The code **MUST** be fixed immediately.
- Critical**
Either a bug with a low probability to impact the behavior of the application in production or an issue which represents a security flaw: empty catch block, SQL injection, ... The code **MUST** be immediately reviewed.
- MAJOR**
Quality flaw which can highly impact the developer productivity: uncovered piece of code, duplicated blocks, unused parameters, ...
- MINOR**
Quality flaw which can slightly impact the developer productivity: lines should not be too long, "switch" statements should have at least 3 cases, ...
- INFO**
Neither a bug nor a quality flaw, just a finding.

Règles à traiter de la plus importante à la moins importante, ex :

Blocker doit être traité immédiatement
En réalité, tous doivent être traité avec plus ou moins d'urgence

The screenshot shows the SonarQube IDE interface with the 'SonarLint Report' tab selected. The report lists various issues found in Java files like App.java, Dao.java, Model.java, and ProductDao.java. Each issue includes a severity icon, a timestamp, and a detailed description with a fix suggestion. For example, a 'BLOCKER' issue in App.java suggests replacing System.out.println with a logger.

source	Date	Description
App.java	51 minutes ago	Replace this use of System.out or System.err by a logger.
App.java	55 minutes ago	Use indentation to denote the code conditionally executed by this "for". [+1 location]
App.java		Remove this unused private "testProductDao" method.
App.java		This block of commented-out lines of code should be removed.
App.java		This block of commented-out lines of code should be removed. [+2 locations]
BddConnecti		Remove the declaration of thrown exception 'java.io.FileNotFoundException' which is a subclass of 'java....'
BddConnecti		Extract this nested try block into a separate method. [+1 location]
BddConnecti		Remove this assignment of "connection". [+1 location]
BddConnecti		Replace this use of System.out or System.err by a logger.
BddConnecti		Replace this use of System.out or System.err by a logger.
BddConnecti		Replace this use of System.out or System.err by a logger.
BddConnecti		Replace this use of System.out or System.err by a logger.
Controller.jav		This block of commented-out lines of code should be removed. [+4 locations]
Dao.java		The return type of this method should be an interface such as "List" rather than the implementation "Arr...
Model.java		Use a StringBuilder instead.
Model.java		Replace the type specification in this constructor call with the diamond operator ("<>").
Model.java		A "NullPointerException" could be thrown; "rows" is nullable here. [+2 locations]
Model.java		Replace the synchronized class "Vector" by an unsynchronized one such as "ArrayList" or "LinkedList".
Model.java		This block of commented-out lines of code should be removed. [+6 locations]
Model.java		Make "productDao" transient or serializable.
ProductDao.j		Complete the task associated to this TODO comment.
ProductDao.j		Complete the task associated to this TODO comment.

The screenshot shows a tooltip for a code smell: 'A conditionally executed single line should be denoted by indentation'. It provides three quick fixes: 'Show issue locations', 'Open description of rule java:S3973', and 'Deactivate rule java:S3973'. Below the tooltip, there is a 'Noncompliant Code Example' and a 'Compliant Solution' section.

A conditionally executed single line should be denoted by indentation (java:S3973)

In the absence of enclosing curly braces, the line immediately after a conditional is the one that is conditionally executed. By both convention and good practice, programmer is entirely unclear and perhaps not actually what is executed. Additionally, such code is highly likely to be confusing to maintainers.

Noncompliant Code Example

```
if (condition) // Noncompliant
    doTheThing();

doTheOtherThing();
somethingElseEntirely();

foo();
```

Compliant Solution

Les autres outils SonarLint

SonarLint on the Fly : sélectionner cet onglet et cliquer sur un fichier pour obtenir son rapport

SonarLint Issue Locations : permet de localiser les portions de codes qui posent problème (par ex l'outil indique une méthode qui doit être factorisée afin d'assurer la maintenabilité de l'appli)

SonarLint Bindings : permet de lier/connecter l'appli locale à un serveur SonarQube (ou SonarCloud) aussi les paramètres définis côté serveur (analyse personnalisée...) seront partagés avec tous les membres de l'équipe. Les autres avantages du mode connecté :

- Les problèmes supprimés comme "ne seront pas résolus" ou "faux positifs" sur le serveur n'apparaîtront pas dans SonarLint
 - Certaines langues ne sont disponibles qu'en mode connecté
 - Des notifications facultatives vous tiennent informé de l'état du portail qualité de votre projet
- Pour connecter un projet local à un projet existant sur un serveur SonarQube :
- Ide/projet/clic droit/sonarLint/bind to sonarQube...

SonarLint Security Hotspots : faille de sécurité qui ne met pas en péril l'appli, au dev de voir si correction indispensable

SonarLint Taint Vulnerabilities : faille de sécurité mettant en péril l'application, nécessite une correction immédiate

Sonar Rules : règles à respecter

The screenshot shows the Sonar Rules interface for Java static code analysis. On the left, a sidebar lists various programming languages: Secrets, ABAP, Apex, C, C++, CloudFormation, COBOL, C#, CSS, Flex, Go, HTML, Java (selected), JavaScript, Kotlin, Kubernetes, Objective C, PHP, PL/I, PL/SQL, Python, RPG, Ruby, Scala, Swift, and Terraform. The main content area has a title "Java static code analysis" with a Java icon. It displays a list of rules under the heading "All rules 649". The first rule shown is "Credentials should not be hard-coded", categorized as a Vulnerability (Blocker). The rule details explain that a hard-coded secret has been found in the code and advise revoking it and changing it in every system that uses it. It also notes that if third parties can authenticate as another system or person, they can impersonate legitimate identities and undermine trust. The "Analyze your code" button is visible. Other listed rules include "Components should not be vulnerable to intent redirection", "XML parsers should not allow inclusion of arbitrary files", "HTTP responses should not be vulnerable to session fixation", "Extracting archives should not lead to zip slip vulnerabilities", and "Dynamic code execution should not be vulnerable to injection attacks".

Compliant Solution

Using AWS Secrets Manager:

```
import software.amazon.awssdk.services.secretsmanager.SecretsManagerClient;
import software.amazon.awssdk.model.GetSecretValueRequest;
import software.amazon.awssdk.services.secretsmanager.model.GetSecretValueResponse;
import org.h2.security.SHA256;

public static void doSomething(SecretsManagerClient secretsClient, String secretName) {
    GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
        .secretId(secretName)
        .build();

    GetSecretValueResponse valueResponse = secretsClient.getSecretValue(valueRequest);
    String secret
        = valueResponse.secretString();
}
```

Using Azure Key Vault Secret:

```
import com.azure.identity.DefaultAzureCredentialBuilder;
import com.azure.security.keyvault.secrets.SecretClient;
import com.azure.security.keyvault.secrets.SecretClientBuilder;
import com.azure.security.keyvault.secrets.models.KeyVaultSecret;
import org.h2.security.SHA256;

public static void doSomething(SecretClient secretClient, String secretName) {
    KeyVaultSecret retrievedSecret = secretClient.getSecret(secretName);
    String secret
        = retrievedSecret.getValue();

    byte[] key = secret.getBytes();
    SHA256.getHMAC(key, message);
}
```

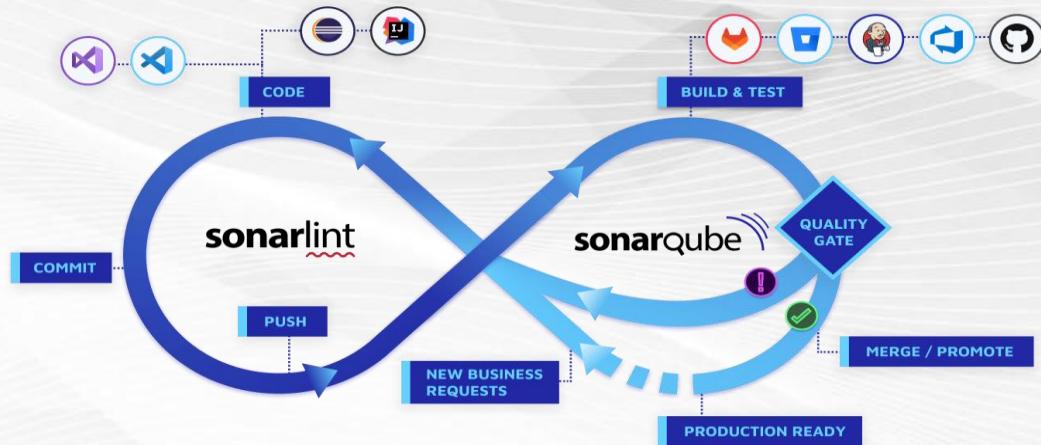
See

- [AWS - Secret Manager](#)
- [Azure - Key Vault](#)
- [GCP - Secret Manager](#)
- [Hashicorp Vault - Secret Management](#)
- [OWASP Top 10 2021 Category A7 - Identification and Authentication Failures](#)
- [OWASP Top 2017 Category A2 - Broken Authentication](#)
- [MITRE, CWE-798 - Use of Hard-coded Credentials](#)
- [MITRE, CWE-259 - Use of Hard-coded Password](#)
- [CERT, MSC03-J - Never hard code sensitive information](#)

Available In:

sonarlint | sonarcloud | sonarqube

SonarLint & SonarQube



SonarLint n'est présent que dans l'IDE hôte, il permet un retour instantané pendant la saisie de code.

SonarQube est un serveur central qui effectue une analyse complète et plus poussée que SonarLint. Il peut héberger plusieurs projets et exécuter l'analyse sur chacun. Il couvre de nombreux langages (Java, C/C++, JS/TS, Python...) Il permet une vue précise et globale de la qualité de code de votre application.

En plus des analyseurs qu'utilise SonarLint, SonarQube utilise des analyseurs tiers (Checkstyle, pmd, findbugs, stylecop), il peut donc détecter des problèmes complexes. De plus, il peut analyser la sécurité d'une appli.

SonarLint peut établir une connexion avec SonarQube. Delors, il peut importer des règles supplémentaires de SonarQube.

L'utilisation de SonarLint et SonarQube forme une solution d'analyse continue de la qualité du code qui maintient notre base de code propre

Parfois il faut prendre le temps de lire la doc !!!

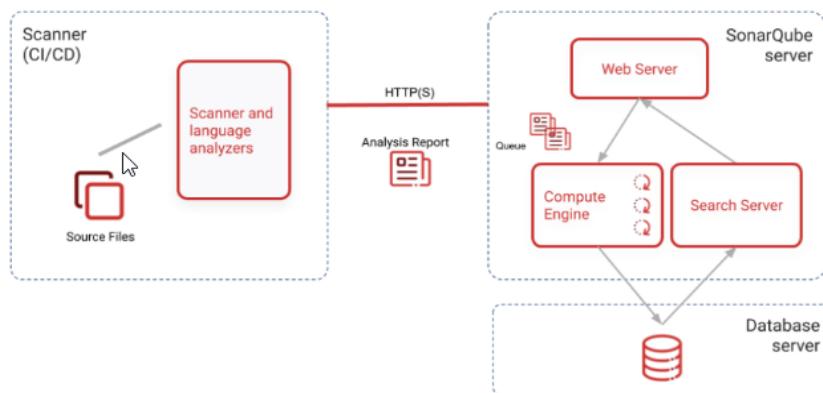
SonarQube

<https://docs.sonarsource.com/sonarqube/9.9/>

Télécharger/Installer SonarQube Community et démarrer le script

C:\Program Files\sonarqube-9.6.1.59531\bin\windows-x86-64>.\StartSonar.bat
puis ouvrir localement votre console d'admin [sonarQube](#)

Une instance SonarQube comprend trois composants :



NB : Java 11 est recommandée pour le Scanner de SonarQube

Attention, si vous utilisez plusieurs versions de Java, assurez vous que le chemin vers la version 11 apparaisse en premier (en haut) dans votre Path

IMPORTANT : En production, il est indispensable d'installer un SGBD type PostgreSQL par exemple et de créer une base de donnée associée à un utilisateur voir les pré-requis [ici](#)
De même, il faudra installer le serveur de SonarQube qui nécessite Java 17 le tout **afin de conserver les rapports d'analyses**.

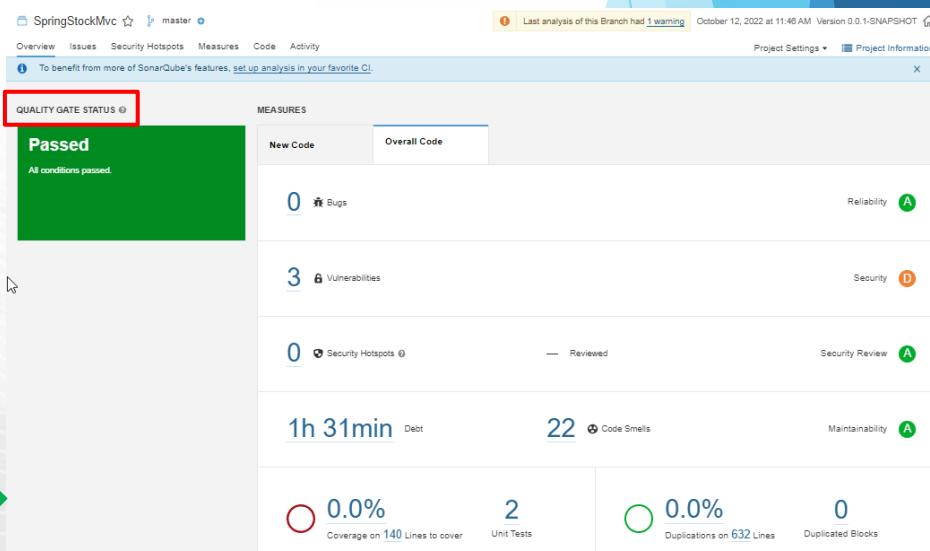
Dans notre cas, ce n'est pas nécessaire une base de donnée temporaire est créée et afin d'éviter à avoir à installer un scanner Sonar, il faut ajouter au fichier settings de maven les paramètres de <pluginGroup> et <profile> voir [ici](#)

- Une fois connecté à SonarQube (login/pwd -> admin/admin à changer)
- Créer un projet manuellement
- Sélectionner l'analyse localement + générer token / continue / maven
- Lancer l'analyse dans votre projet avec maven :
 - à l'aide des commandes générées par SonarQube, positionnez vous dans le répertoire de votre projet avant d'exécuter les commandes copier/coller:

```
mvn clean verify sonar:sonar \
-Dsonar.projectKey=sonar-project \
-Dsonar.host.url=http://localhost:9000 \
-Dsonar.login=sq_80497480731f81f8b0502f1df7380c3ba5c1ed52
```

```
[INFO] ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=api-training
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=AywU0crYKoNw_t744o-1
[INFO] Analysis total time: 8.132 s
[INFO]
[INFO] BUILD SUCCESS
```

→ Si tout va bien
Quality Gate Status : passed



Qualités & Mesures

Qualité de code : manière avec laquelle est codée par ex une fonctionnalité

Qualité logiciel : résultat final de l'application

Mesures ou Métrique logiciel : Consiste à évaluer une caractéristique spécifique d'un logiciel, telle que le nombre de lignes de code. C'est une approche quantitative qui implique l'extraction de données mesurables du code source pour évaluer la qualité du logiciel, généralement en utilisant des méthodes d'analyse statistique. En d'autres termes, c'est une manière de quantifier des aspects du logiciel pour mieux comprendre et évaluer ses performances et sa qualité.

Exemple de mesures logiciel :

- nb de if, for..
- nb de ligne de code
- nb & qualité des commentaires
- couverture de code par test unitaires
- duplication de code (factorisation)
- instabilité du code lié à plusieurs niveaux d'héritages (simplifier, utiliser interfaces et abstractions)
- trop de paramètres ou de variables dans une méthode, voir trop surcharge dans un méthode
- nb de failles de sécurités [Vulnérabilité]
- nb d'erreurs [Bugs]
- Portabilité [Code Smell]

La finalité ici reste d'identifier, via les différents indicateurs, comment améliorer la qualité de nos applications

Les classes d'évaluation			
Classe	Maintenabilité	Bugs	Sécurité
A	<=5%	0	>= 80%
B	6 à 10%	1 mineur	>= 70% à <80%
C	11 à 20%	1 majeur	>= 50% à <70%
D	21 à 50%	1 critique	>= 30% à <50%
E	>50%	1 bloqueur	< 30%

Admin Console

Issues & filters

Add at least one assertion to this test case.

Tests should include assertions [java:S2699](#)

Code Smell • **Blocker** • **Open** • **Not assigned** • **10min effort** • **0 comments**

Add at least one assertion to this test case.

```

class SpringStockMvcApplicationTests {
    @Autowired
    ImplCaddy implCaddy;

    @Test
    void contextLoads() {
        Add at least one assertion to this test case.
    }

    @Test
    void testCaddy() {
        implCaddy.addArticle(new Article((long)1,"Samsung S8",250,1,null));
        implCaddy.addArticle(new Article((long)1,"Samsung S9",250,1,null));
        implCaddy.addArticle(new Article((long)2,"Iphone 10",500,1,null));

        assertEquals(implCaddy.getTotalAmount(),1000);
    }
}

```

SpringStockMvc master

Last analysis of this Branch had **1 warning** October 12, 2022 at 11:46 AM Version 0.0.1-SNAPSHOT

Project Settings • Project Information
1 / 25 issues | 2h 1min effort

My Issues All

Bulk Change

Filters

Period: New code

Type: Bug, Vulnerability, Code Smell (0)

Severity: Blocker (1), Critical (6), Major (10), Minor (5), Info (3)

Scope, Resolution, Status, Security Category, Creation Date, Language, Rule, Tag, Directory, File, Assignee, Author

pom.xml: Remove this commented out code. (2 hours ago) L55 % unused

src/main/java/fr/ldnr/SpringStockMvcApplication.java: Remove this unused import 'fr.ldnr.entities.Article'. (2 hours ago) L20 % unused

src/main/java/fr/ldnr/entities/Category.java: Remove this unused import 'fr.ldnr.entities.Category'. (2 hours ago) L21 % unused

src/main/java/fr/ldnr/business/ImplCaddy.java: This block of commented-out lines of code should be removed. (2 hours ago) L37 % unused

src/main/java/fr/ldnr/business/ImplCaddy.java: This block of commented-out lines of code should be removed. (2 hours ago) L39 % unused

src/main/java/fr/ldnr/business/ImplCaddy.java: This block of commented-out lines of code should be removed. (2 hours ago) L78 % unused

src/main/java/fr/ldnr/dao/ArticleRepository.java: Replace the type specification in this constructor call with the diamond operator ("<>"). (2 hours ago) L18 % clumsy

src/main/java/fr/ldnr/dao/ArticleRepository.java: Replace the type specification in this constructor call with the diamond operator ("<>"). (2 hours ago) L43 % clumsy

src/main/java/fr/ldnr/dao/ArticleRepository.java: Replace this if-then-else statement by a single return statement. (2 hours ago) L55 % clumsy

src/main/java/fr/ldnr/entities/Category.java: This block of commented-out lines of code should be removed. (2 hours ago) L3 % unused

Supervisions des projets

A = 0
B = at least 1 Minor
C = at least 1 Major
D = at least 1 Critical
E = at least 1 Blocker

C'est bien tout ça mais il faut maintenant trouver un moyen d'automatiser tout le cycle de dev jusqu'en production : introduction au CI/CD

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Search for projects... Create Project

My Favorites All

3 projects

Perspective: Overall Status Sort by: Name

SpringShopJpa Passed Last analysis: 2 minutes ago

- Bugs: 4 (C)
- Vulnerabilities: 2 (D)
- Hotspots Reviewed: 0.0% (E)
- Code Smells: 60 (A)
- Coverage: 0.0%
- Duplications: 0.0%
- Lines: 704 (XS) Java, XML

SpringStockMvc Passed Last analysis: 22 hours ago

- Bugs: 0 (A)
- Vulnerabilities: 3 (D)
- Hotspots Reviewed: - (A)
- Code Smells: 22 (A)
- Coverage: 0.0%
- Duplications: 0.0%
- Lines: 632 (XS) Java, XML

test-freestyle Passed Last analysis: 5 days ago

- Bugs: 0 (A)
- Vulnerabilities: 0 (A)
- Hotspots Reviewed: - (A)
- Code Smells: 1 (A)
- Coverage: 0.0%
- Duplications: 0.0%
- Lines: 57 (XS) XML, Java

3 of 3 shown



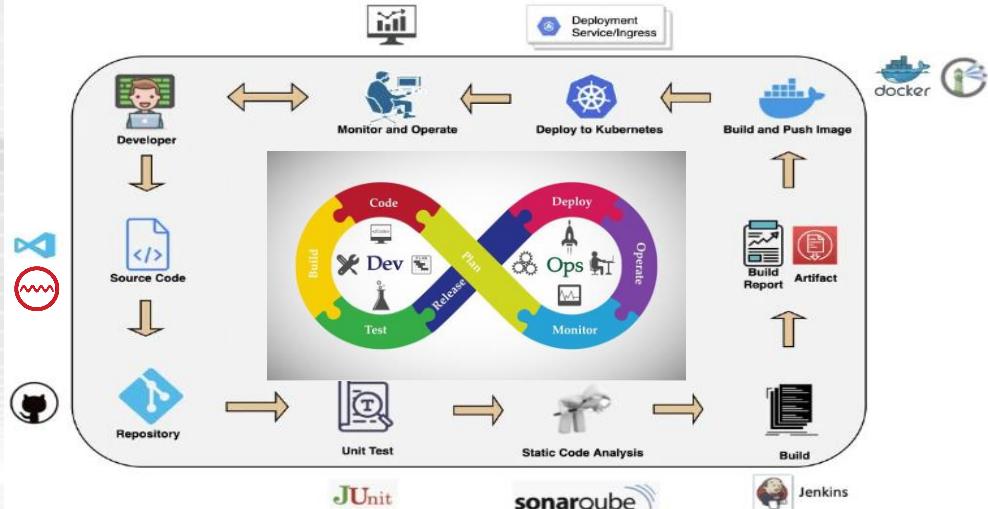
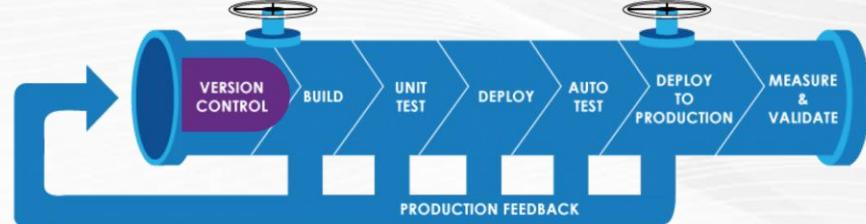
Définition : fusion entre 2 phases de conception d'une application Développement & Opérationnelle afin de proposer une application + performante à un cout/délai moindre. C'est une nouvelle manière de travailler en automatisant par ex toutes les taches secondaires.

Les activités du DevOps (techniques et non techniques) :

- Maitriser les outils d'automatisation d'intégration/déploiement continue, de versionnings, conteneurisation/orchestration, cloud computing, sécurité, surveillance, script et langage de programmation.
- Maitriser les outils de travail collaboratif, les systèmes d'exploitation et leur ligne de commande, gestion du changement dans le cycle de vie d'un logiciel, culture DevOps, Communication, Agilité, sécurité, gestion de projet et formation continue car les choses évoluent si vite, veille méthodo/techno permanente.

Pour devenir un DevOps complet, il faut beaucoup de temps et d'expérience afin d'avoir une compréhension des principes et des compétences associées. Une grande curiosité, de la patience et de la résilience car les traversés du désert (seul) sont nombreuses. Un grand gout pour la lecture des différentes documentations permettra l'économie de nombreux neurones. L'utilisation pertinente d'outil IA tel que ChatGpt peut s'avérer très utile mais il ne faut pas se reposer que sur celui-ci.

Intégration/Déploiement continue (CI/CD) avec Jenkins



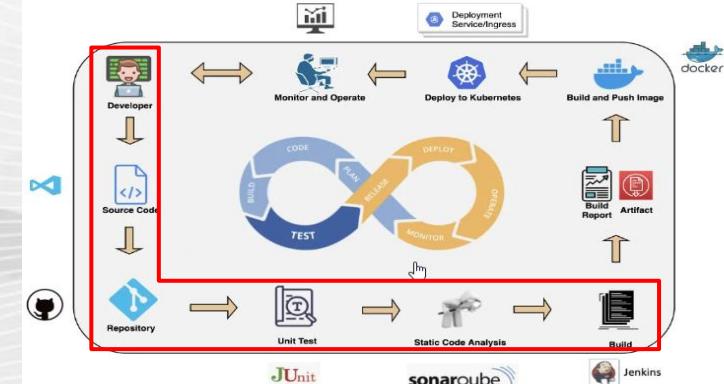
Intégration continue (CI): process permettant aux dev d'intégrer leur code sur une ressource partagée.

Déploiement continue (CD): process permettant la mise en production du travail des devs vérifié et validé.

Pipeline CI/CD : Les 2 process sont automatisés et permanent, on parlera de Job dans un pipeline

Jenkins est un outil d'automatisation des process CI/CD, open source (gratuit) et écrit en Java. Il peut créer des pipelines permettant à des devs de se connecter sur GitHub ou GitLab pour intégrer leur code, puis déployer leurs applis sur des conteneurs tels que Docker. De plus, il peut prendre en charge d'autres outils tel que le contrôle qualité à l'aide de SonarQube.

Comparatifs outils CI/CD

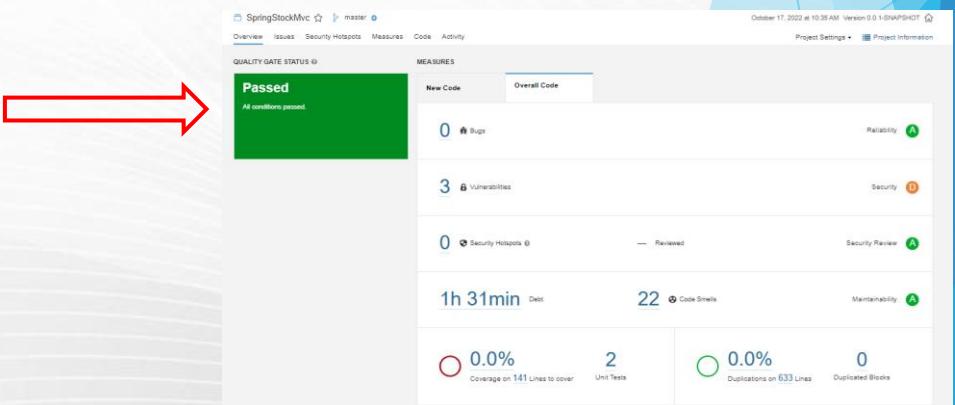


Comment mettre en place l'intégration continue avec Jenkins :

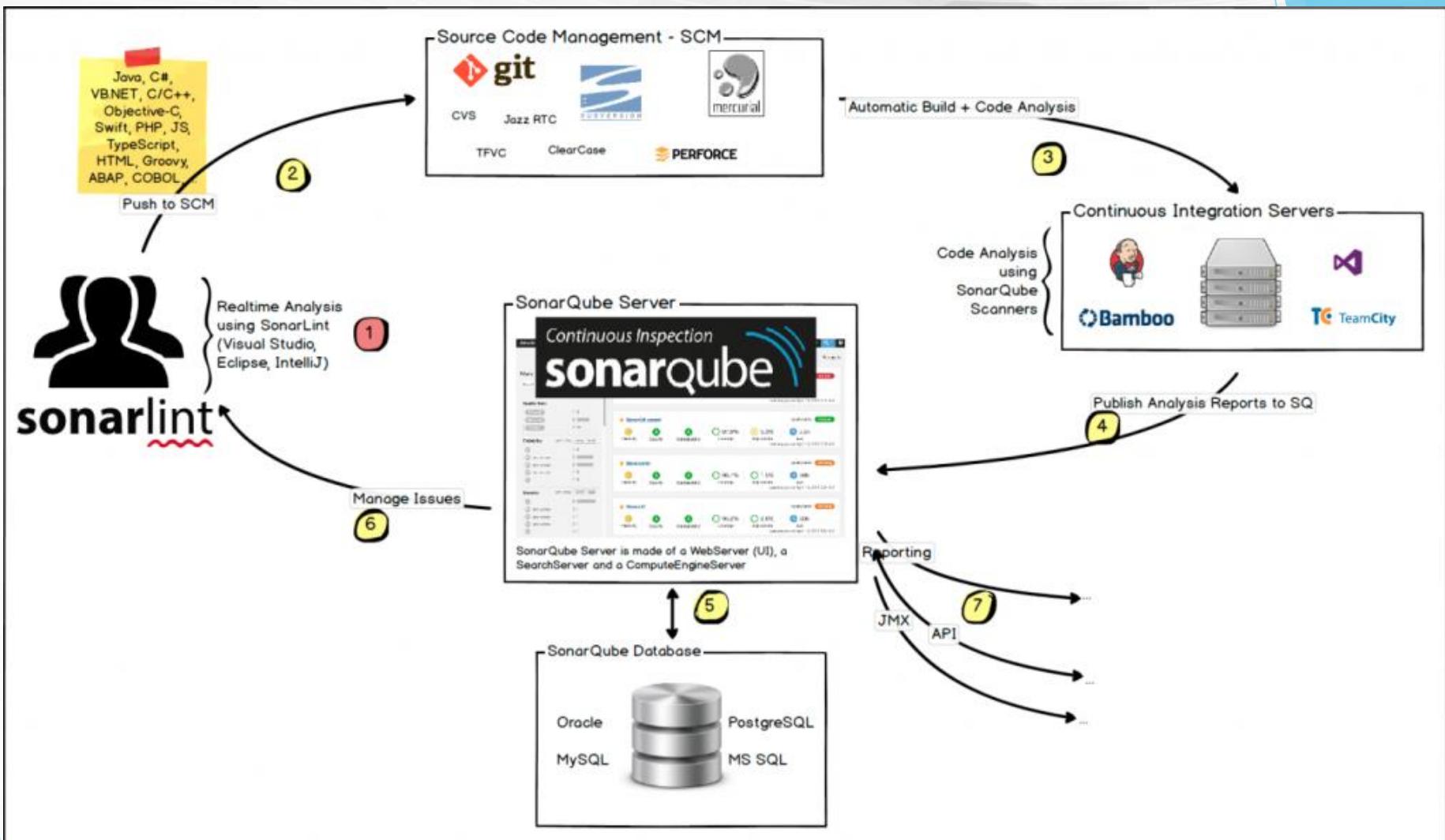
- **Télécharger/Installer Jenkins** (Msi ou War) et démarrer (**java -jar jenkins.war**) par défaut [ici](#)
- **Télécharger/Installer Sonarqube** (si c'est pas déjà fait se reporter au chapitre sur SonarQube)
- **Ajouter à Jenkins le plugin SonarQube Scanner :**
 - Administrer Jenkins / plug in manager / SonarQube Scanner : install
- **Indiquer où se trouve SonarQube :**
 - Administrer Jenkins / configurer le système / sonarQube servers / ajouter install SonarQube
 - Activer injection sonar + nom + url serveur (par défaut <http://localhost:9000/>)
- **Préciser la version de sonarQube Scanner**
 - Administrer Jenkins / Global tool Configuration / ajouter sonarQube Scanner : appliquer
- **Ajouter le chemin vers Git :**
 - Administrer Jenkins / Tools / Path ...
- **Create new job(item) / Freestyle project** : « *jenkins-spring-shop* » par ex
 - / Git / url du repo git associé (public donc pas de credentials) + branch main (pas master)
 - / build steps / Invoquer les cibles maven de haut niveau / cibles maven : *clean verify sonar:sonar*
- **Lancer le build** : tableau de bord / mes vues / sélectionner le projet à construire → visualiser la sortie de console

Il faut créer un token avec Sonar (MyAccount/security) et l'intégrer à Jenkins (administrer/credentials) afin de l'utiliser ici

```
[INFO] CPD Executor CPD calculation finished (done) | time=13ms
[INFO] Analysis report generated in 80ms, dir size=186.0 kB
[INFO] Analysis report compressed in 96ms, zip size=65.6 kB
[INFO] Analysis report uploaded in 373ms
[INFO] ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=fr.ldnr%3ASpringstockMvc
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=AYP1FdgN02eZIwLxgXN
[INFO] Analysis total time: 9.590 s
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 25.619 s
[INFO] Finished at: 2022-10-17T10:35:57+02:00
[INFO] -----
Finished: SUCCESS
```



Synthèse



Docker



Tout d'abord, un **conteneur** est un environnement permettant d'isoler des applications et leurs dépendances respectives afin d'éviter qu'elles ne créent des interférences entre elles. Il s'agit d'une alternative aux machines virtuelles (simulent un SE), les conteneurs vont puiser uniquement les ressources nécessaires à l'exécution de l'appli. Ce qui permet de simuler plusieurs systèmes dans le même SE. Un conteneur peut être stoppé dès qu'un processus n'est plus en cours d'exécution. Tout cela fait qu'ils sont + performants, flexibles, rapides, légers et simples à mettre en place.

Docker est une plateforme de conteneurisation dans laquelle on peut tester, déployer et exécuter des applications sur n'importe quel environnement (Linux, Windows, Mac, Cloud...). Il est basé sur le kernel Linux !

Docker offre un réel avantage sur les systèmes distribués (1 seule machine) S'agissant du BigData ou le cloud, il est particulièrement fiable, sûre, scalable, consomme très peu de ressources, les apps sont faciles à déployer...

Il couvre tous les écosystèmes : Java, Node, .Net, Php...

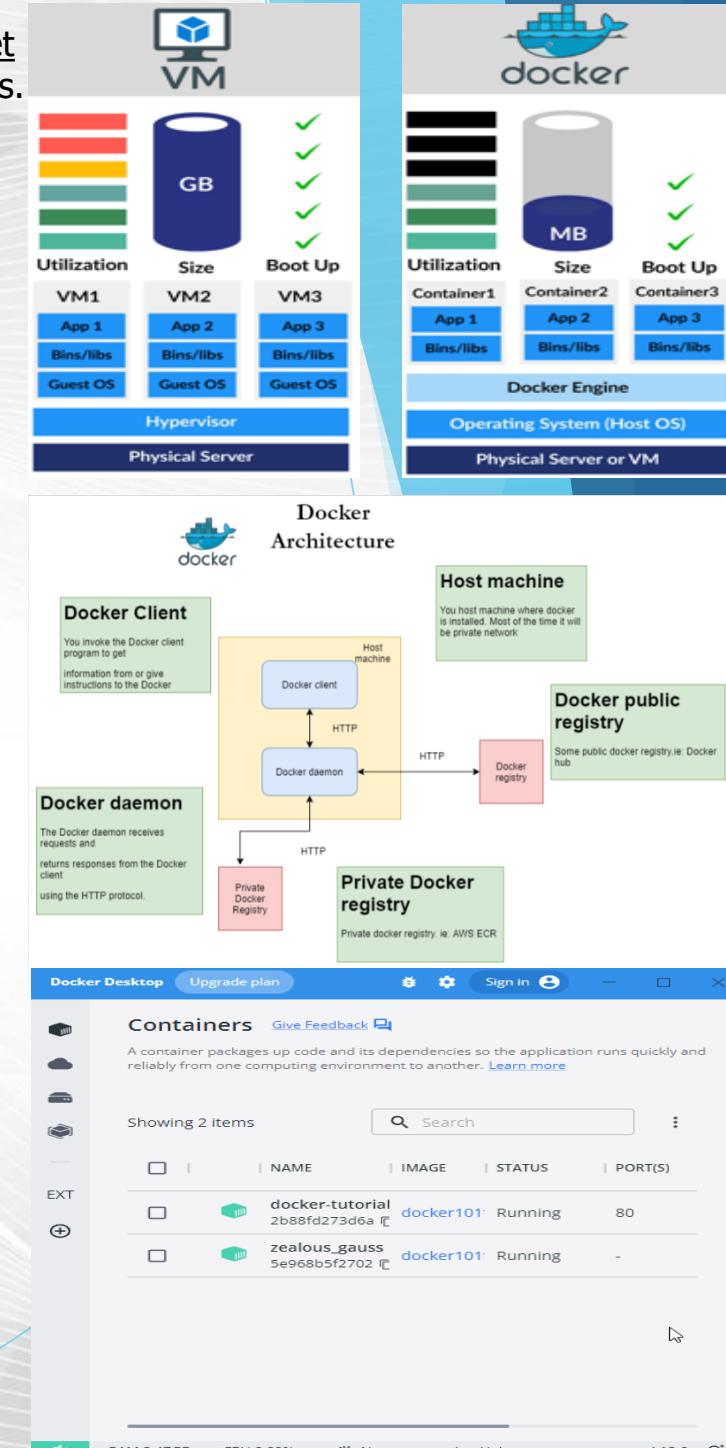
Les conteneurs Docker sont déployés sous forme d'**images**. C'est à partir de ces images que l'on crée le conteneur sur une plateforme hôte, qu'on l'exécute sur celle-ci et qu'on partage sur Docker Hub par ex.

Docker peut être lancé en mode console mais l'outil **Docker Desktop** est à privilégier.

Vous pouvez l'installer à partir [d'ici](#), installer WSL2 au passage car le noyau de Docker tourne sous Linux, il faut ajouter cet élément au SE hôte sous windows notamment.

Le **Docker Registry** est l'emplacement où l'on peut stocker et télécharger des images Docker. Il existe 2 types de DR : registres publics et privés.

Le registre par défaut de Docker est **Docker Hub** permettant d'accéder à des images et de créer des registres privés. C'est un hébergeur d'images de conteneurs.



Mise en œuvre d'un Pipeline Jenkins CI/CD

La mise en œuvre d'un pipeline Jenkins repose sur la création d'un fichier nommé **Jenkinsfile**. Il s'agit d'un fichier texte contenant toutes les étapes à suivre pour créer le pipeline. Jenkins propose un assistant pour simuler ce fichier, [sinon il faudra le réaliser et l'ajouter à la racine du projet](#).

Pour construire un pipeline Jenkins, il faut :

- Installer Docker Desktop puis Démarrer
- Installer Jenkins (windows/Msi) + démarrer
- Réaliser un projet Spring fonctionnel, ajouter le DockerFile (script permettant à Docker de créer une image) **puis pousser sur GitHub**
- Sur Jenkins/ nouveau item / pipeline :
 - GitHub project / url repo à ajouter
 - Cocher « *GitHub hook trigger for GITScm polling* »
 - Compléter le Pipeline script (slide suivant)
- Jenkins / Global tool config / Docker / install automatically : name + latest

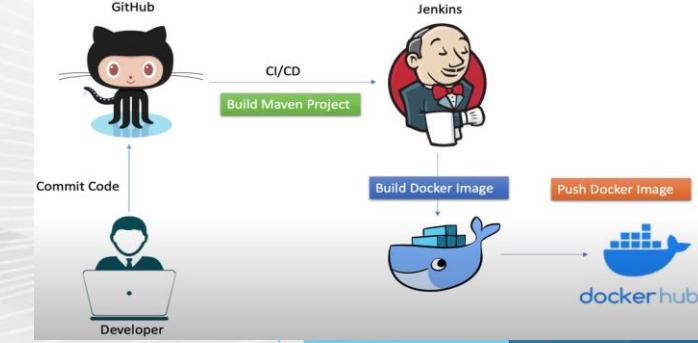
- Nous pouvons dès lors déjà tester la construction d'une image docker et lancer l'application
 - Sur Docker Desktop, vous devez avoir l'image dans la liste : **RUN**
 - Sur un navigateur, tester l'application lancée dans un conteneur Docker

- Pousser l'image docker sur DockerHub
 - Compléter la dernière étape de votre pipeline (id/pwd docker hub)

NB : L'autre manière consiste à ajouter un fichier jenkinsfile au niveau de la racine du projet directement avec le même contenu, puis Pousser sur GitHub, il faut changer dans la config du pipeline : « *Pipeline script from SCM* »...

Cette approche est à privilégier !

**Afin de bénéficier d'outils complémentaires, il est nécessaire
De customiser Jenkins en ajoutant ici et là les plugins**

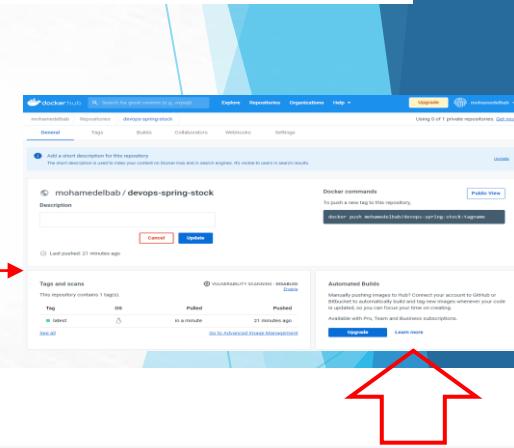


DockerFile à ajouter à la racine du projet

```
FROM openjdk:8
EXPOSE 8080
ADD target/devops-spring-app.jar devops-spring-app.jar
ENTRYPOINT ["java","-jar","/devops-spring-app.jar"]
```

Pour changer le nom du jar généré dans pom.xml

```
<finalName>devops-spring-app</finalName>
</build>
```



Stage View

Average stage times:
(Average full run time: ~1min 19s)

#17	Oct 18 11:11	No Changes
#16	Oct 17 16:15	No Changes

Declarative: Tool Install	Build Maven	Build docker image	push docker image to dockerhub
159ms	21s	7s	30s
247ms	27s	5s	1min 14s
105ms	24s	3s	1min 44s

Selectionner l'option Pipeline Script et Utiliser l'outil **Pipeline Syntax** pour générer le code du script puis copier/coller

Par exemple, s'agissant de l'étape 'build maven', selectionner **Checkout : Check out from version control**

Ajouter l'url du repo (si public, pas de credentials) + Préciser le nom de la branche principale */main ou master

```
pipeline {  
    agent any  
    tools {  
        maven "maven_3_9_5"    On peut préciser la version de maven à utiliser ou pas : "maven"  
    }  
    stages {  
        Stages correspond à un ensemble d'étapes pour créer, tester et déployer une appli, on pourra les visualiser  
        stage('Build Maven') {  
            steps {  
                checkout([$class: 'GitSCM', branches: [[name: '/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/elbabili/devops-spring-app']]])  
                bat 'mvn clean install'  
            }  
        }  
  
        stage('Build docker image'){  
            steps {  
                script {  
                    bat 'docker build -t mohamedelbab/devops-spring-app .'  
                }  
            }  
        }  
  
        stage('push docker image to dockerhub'){  
            steps {  
                script {  
                    withCredentials([string(credentialsId: 'docker-hp', variable: 'docker-hub-pwd')]) {  
                        bat 'docker login -u mohamedelbab -p %docker-hub-pwd%'  
                    }  
                    bat 'docker push mohamedelbab/devops-spring-app'  
                }  
            }  
        }  
    }  
}
```

Le **pipeline** bloc se compose de toutes les instructions pour créer, tester et livrer des logiciels. C'est le composant clé d'un pipeline Jenkins

Un **agent** est assigné pour exécuter le pipeline sur un nœud et allouer un espace de travail pour le pipeline

withCredentials: Bind credentials to variables

withCredentials ?

Secret values are masked on a best-effort basis.
guidelines.

Bindings

Secret text ?

Variable ?

docker-hub-pwd

Credentials ?

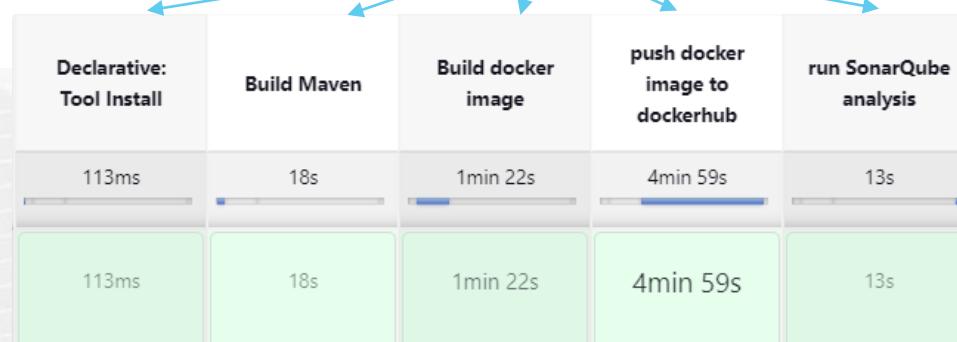
docker-hub-pwd

+ Ajouter ▾

Jenkins

Ajouter ▾

withCredentials : Bind credentials to variables 
secret text
add credentials



D'après vous, est-ce judicieux de lancer l'analyse à la fin du pipeline ?

Que faut-il rajouter à notre pipeline pour bénéficier de l'analyse SonarQube ici ?

Pour Infos

FROM openjdk:8	-> spécifie l'image de base à utiliser pour construire l'image docker, l'image 'openjdk:8' est une image officiel d'OpenJdk version 8
EXPOSE 8888	-> indique que le conteneur docker exposera le port 8888 pour l'application
ADD target/devops-spring-app.jar devops-spring-app.jar	-> ajoute le fichier JAR de l'application à l'image Docker depuis le répertoire Target
ENTRYPOINT ["java","-jar","/devops-spring-app.jar"]	-> indique la commande d'entrée qui sera exécutée au démarrage du conteneur

groupId : groupe du projet

artifactId : nom du projet

Version : version du projet

Name : ???

Nb : ArtifactId est un identifiant unique utilisé pour identifier votre projet dans le système de gestion de maven, alors que name est un nom lisible par l'homme.

mvn clean : Nettoie le répertoire **target**, supprimant les fichiers générés lors de la compilation précédente.

mvn compile : Compile les sources du projet.

mvn test : Exécute les tests unitaires du projet.

mvn package : Emballe le projet dans un fichier JAR ou WAR, selon le type de projet.

mvn install : Compile + test + package

mvn deploy : Déploie le paquet dans un référentiel distant pour le partager avec d'autres développeurs ou projets.

mvn clean install : Effectue un nettoyage, une compilation et une installation en une seule commande.

mvn clean package : Effectue un nettoyage et un empaquetage en une seule commande.

mvn clean test : Effectue un nettoyage et exécute les tests unitaires en une seule commande.

...

- En cas de modif sur un projet, ne pas oublier de pousser sur GitHub car le pipeline jenkins va démarrer de là
- Prenez l'habitude de démarrer une console en mode admin pour démarrer sonar, jenkins...
- En cas d'erreur, prenez le temps de lire les logs, le pb/solution est souvent là
- savoir trouver et killer un processus sous windows peut être un plus

Pour aller plus loin

Voir Kubernetes : Plateforme permettant d'optimiser l'orchestration des conteneurs...

Docker Compose : Docker Compose est un outil qui permet de définir et de gérer des applications multi-conteneurs Docker

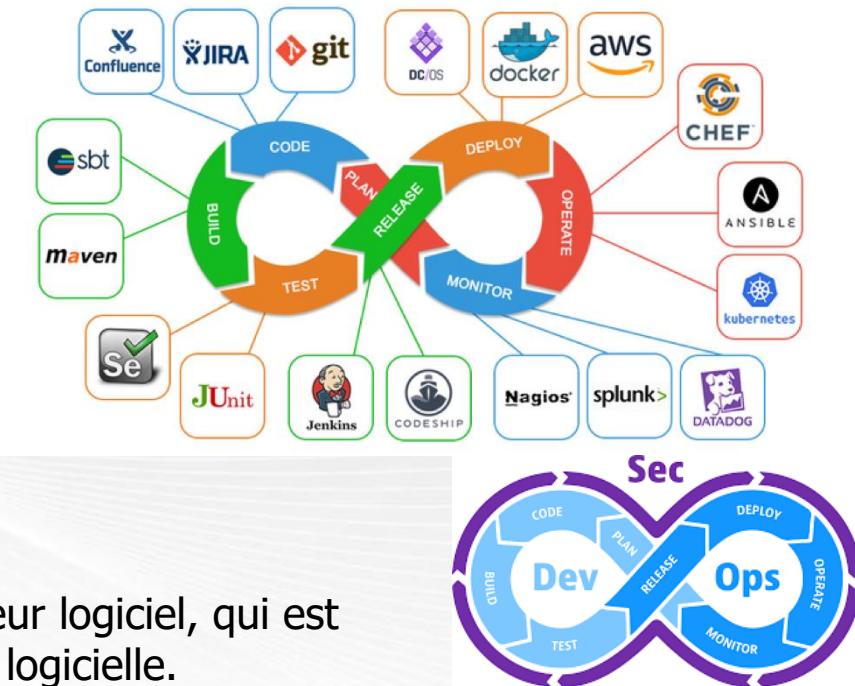
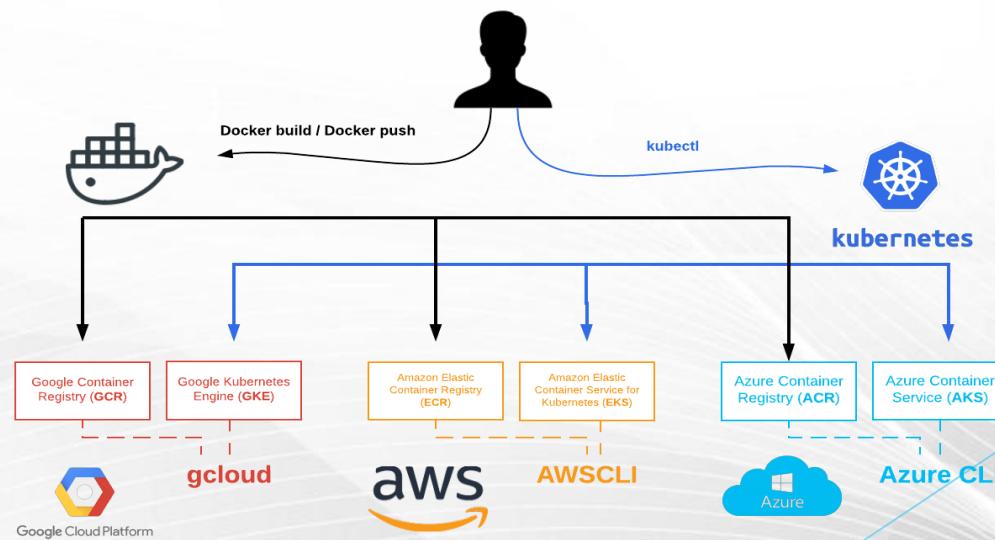
Voir aussi : **Spring Docker Compose, SonarCloud, DevSecOps/CyberSécurité**

C'est quoi le métier de l'architecte logiciel ?

L'architecte logiciel est un expert en informatique, souvent ingénieur logiciel, qui est responsable de la création et du respect du modèle d'architecture logicielle.

Il travaille en étroite collaboration avec les concepteurs, souvent leader technique, eux-mêmes garantissent que leurs équipes de devs respectent les règles établies.

Voir aussi Aws, Azure ou Google Cloud Platform, JFrog... pour comprendre pourquoi on raisonne de + en + qu'en terme d'infrastructures...



CONCLUSION

La question de la qualité dans une application est essentielle, sa mise en œuvre peut être contraignante mais apporte une réelle + value à court, moyen et long terme.

Tout commence avec la prise de conscience au « clean code » et « clean as you code »

Les outils Sonar sont relativement intuitif et complet, ils sont utilisés par environ 6 millions de devs, des forums dédiés favorisent l'amélioration continue.

Le retour sur investissement s'échelonne certes dans le temps : Risque & Productivité
Ceci dit, les retours des utilisateurs des applis sont plutôt positif

L'idée reste de construire de meilleur logiciel avec

- Un process linéaire répétable (devOps)
- Une meilleure qualité de code pour éviter de devoir le reconstruire (Clean code)
- La prise en compte permanente des impératifs de sécurité

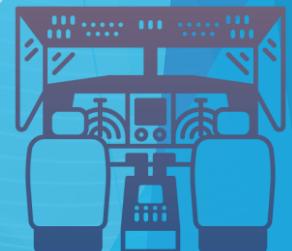
SonarQube s'intègre parfaitement au process devOps d'intégration/déploiement continu par l'automatisation des outils, on peut donc l'utiliser :

- Au lancement d'une image docker
- Dans un pipeline Jenkins
- Binding appli locale avec serveur SonarQube

Cout Sonar : 75% de la solution est opensource et gratuite, s'agissant des fonctionnalités avancées, il faudra tout de même basculer sur l'offre commercial.

Enfin, l'activité DevOps s'apparente à celle d'un pilote expérimenté dans son cockpit, il a de nombreux indicateurs qu'il a automatisés lui permettent de superviser l'ensemble des étapes CI/CD, il est près à prendre les commandes quand c'est nécessaire et gère au mieux les zones de fortes turbulences.

Il connaît parfaitement son avion et maîtrise chaque élément de son cockpit.



Ressources

<https://maven.apache.org/guides/index.html>

<https://cwiki.apache.org/confluence>

<https://docs.sonarqube.org/latest/>

<https://community.sonarsource.com/>

<https://rules.sonarsource.com/java>

<https://www.youtube.com/watch?v=AuF7d-fxRj8>

<https://damien.pobel.fr/post/clean-code/>

<https://www.invivoo.com/lart-clean-code-environnement-java/>

<https://www.qrpinternational.fr/blog/gestion-des-services-informatiques/devops-cest-quoi-definition-devops/7>

<https://www.data-transitionnumerique.com/devops-jenkins/>

<https://youtu.be/PKcGy9oPVXg>

<https://devcenter.heroku.com/articles/local-development-with-docker-compose>

<https://youtu.be/MCqHTILVGYA>

<https://youtu.be/R8ttW4wimv8>

<https://jfrog.com/>