UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MGL7460-90 HIVER 2020

REALISATION ET MAINTENANCE DE LOGICIELS

PROJET INDIVIDUEL

PRÉSENTÉ PAR

MAJED ABIDI

7 FÉVRIER 2020

Équipe Enseignante :

Professeur : Sébastien Mosser (UQAM)

Laboratoires : Jean-Philippe Gélinas (SFL, Lévis), Jonatan Cloutier (SFL, Montréal)

Table des matières

1. Introduction

Il y a plus d'une dizaine d'années, le framework Java de tests unitaires JUnit avait connu une forte évolution avec l'introduction de JUnit 4. A partir troisième trimestre 2017 une nouvelle ère approche avec JUnit 5. L'équipe JUnit a livré en début avril 2017 une quatrième version *milestone* de JUnit 5 avec une architecture complètement différente des versions précédentes et comprenant une documentation complète.

Dans cet article, je présenterai l'architecture de JUnit 5, la qualité du code source ainsi une analyse sur la stabilité de l'équipe de développement.

Je présenterai ensuite un bref état des lieux sur le support de JUnit 5 par les outils de développement les plus populaires et je terminerai par les méthodes de test mises en place dans le projet.

2. Cadre d'analyse de maintenance

2.1. Dimension "Équipe de développement"

Pour faire l'analyse sur la dimension de l'équipe de développement nous avons utilisé l'outil Gitstat. Cet outil collecte les informations à partir de git log sur tous les commits poussé par personne ou par groupe. Il utilise une comparaison de sous-chaine très simple avec l'adresse email trouvée dans les balises git Author, signée par, acquittée, testée par et rapportée par.

Project name:
junit5
Generated:
2020-04-10 19:23:15 (in 6023 seconds)
Generator:
GitStats (version)
Report Period:
2015-06-04 09:33:07 to 2020-04-10 11:17:47
Age:
1773 days, 1120 active days (63.17%)
Total Files:
1418
Total Lines of Code:
113435 (249670 added, 136235 removed)
Total Commits:
6192 (average 5.5 commits per active day, 3.5 per all days)
Authors:
153

Figure 1 : Nombre total des commits, auteurs et ligne de code

2.1.1.Qui sont les développeurs principaux du projet ?

List of Authors

Author	Commits (%)	+ lines	- lines	First commit	Last commit	Age	Active days	# by commits
Sam Brannen	2564 (41.41%)	154228	85572	2015-10-23	2020-04-07	1628 days, 1:58:10	673	1
Marc Philipp	1651 (26.66%)	114805	53422	2015-06-04	2020-04-10	1772 days, 1:44:40	549	2
Christian Stein	672 (10.85%)	44707	25964	2016-11-15	2020-04-04	1236 days, 2:21:09	381	3
Johannes Link	578 (9.33%)	33361	19706	2015-10-26	2017-11-10	746 days, 14:56:45	110	4
Matthias Merdes	225 (3.63%)	13528	7443	2015-10-28	2020-02-24	1579 days, 17:51:07	78	5
Stefan Bechtold	71 (1.15%)	11447	8474	2015-10-27	2017-01-11	441 days, 6:49:58	27	6
Stefan Birkner	43 (0.69%)	1398	1054	2016-03-09	2017-10-21	590 days, 12:49:31	23	7
JUnit Team	38 (0.61%)	13121	14816	2015-12-02	2020-01-02	1491 days, 22:21:38	17	8
Juliette de Rancourt	28 (0.45%)	2108	663	2019-05-20	2020-03-29	314 days, 2:32:15	14	9
Jonathan Bluett-Duncan	14 (0.23%)	926	283	2016-08-22	2017-09-12	386 days, 0:07:28	9	10
lutovich	13 (0.21%)	4248	155	2016-05-02	2018-01-11	618 days, 20:15:48	6	11
Jens Schauder	12 (0.19%)	321	213	2015-12-19	2016-02-12	55 days, 6:11:06	2	12
Jonathan Leitschuh	10 (0.16%)	1123	206	2017-06-29	2020-01-15	930 days, 11:24:22	10	13
Steve Moyer	9 (0.15%)	374	129	2016-05-13	2017-02-18	281 days, 0:31:34	8	14
Giorgos Gaganis	9 (0.15%)	507	190	2017-02-26	2018-05-25	453 days, 0:43:15	9	15
aalmiray	8 (0.13%)	470	356	2016-05-05	2016-05-12	7 days, 1:01:34	2	16
Edward Thomson	8 (0.13%)	668	242	2019-03-05	2019-03-16	10 days, 18:34:29	6	17
Nicolai Parlog	7 (0.11%)	677	90	2016-07-09	2018-08-29	780 days, 11:58:44	7	18
Jendrik Johannes	7 (0.11%)	13	18	2019-10-15	2019-10-18	3 days, 3:46:46	3	19
Thomas Heilbronner	5 (0.08%)	1990	122	2016-04-11	2017-12-23	621 days, 1:59:03	5	20

Figure 2 : Liste des autheurs par nombre de commits/Lines de code

Author of Year

Year	Author	Commits (%)	Next top 5
2020	Marc Philipp	90 (54.22% of 166)	Sam Brannen, Christian Stein, Juliette de Rancourt, Matthias Merdes
2019	Sam Brannen	301 (38.39% of 784)	Marc Philipp, Christian Stein, Juliette de Rancourt, Edward Thomson
2018	Sam Brannen	615 (49.40% of 1245)	Marc Philipp, Christian Stein, Matthias Merdes, Giorgos Gaganis
2017	Sam Brannen	503 (42.45% of 1185)	Marc Philipp, Christian Stein, Matthias Merdes, Stefan Birkner
2016	Sam Brannen	794 (42.99% of 1847)	Marc Philipp, Johannes Link, Matthias Merdes, Stefan Bechtold
2015	Johannes Link	330 (34.20% of 965)	Sam Brannen, Marc Philipp, Matthias Merdes, Stefan Bechtold

Figure 3 : Top 5 des autheurs par année

L'analyse des deux figures on se basant sur le pourcentage du nombre des commits poussé par personne nous permet de conclure que les developpeurs principaux sont les suivant :

1- Sam Branen
2- Marc Philip
3- Johanne Link
4- Matthias Merdes
5- Stefan Bechtold

2.1.2. L'équipe de développement est-elle stable ?

Selon les informations collectées par Gitstats, les développeurs/ingénieurs principaux de JUnit5 sont les mêmes depuis le lancement du projet jusqu'aujourd'hui avec un taux de participation presque égal entre les membres de l'équipe, cela nous permet de conclure que l'équipe est stable.

2.1.3. Comment est répartie la paternité du code source dans l'équipe ?

À la suite d'une analyse manuelle d'un certain nombre de commits par développeur, j'ai constaté qu'il n'existe pas une répartition du code source dans l'équipe.
Tous les développeurs ont touché à presque tous les différents modules de la solution Junit5, ce qui nous démontre que <<tout fait tout >>.

2.1.4. Comment les développeurs communiquent entre eux ?

Les canaux principaux de communication entre les développeurs sont StackOverflow et Gitter.

2.2. Dimension "Architecture logicielle"

2.2.1. Quels sont les composants principaux de l'application ?

- JUnit Platform : Fondation pour le lancement de frameworks de test sur la JVM.
- JUnit Jupiter : Une nouvelle API et un mécanisme d'extension pour écrire des tests Unitaire sur JUnit5.
- JUnit Vintage : C'est un TestEngine pour exécuter des tests existant sur JUnit3&4

2.2.2. Comment est gérée la modularité de l'application ?

JUnit 5 est devisé en 3 modules et se présentait sous forme d'une architecture modulaire :

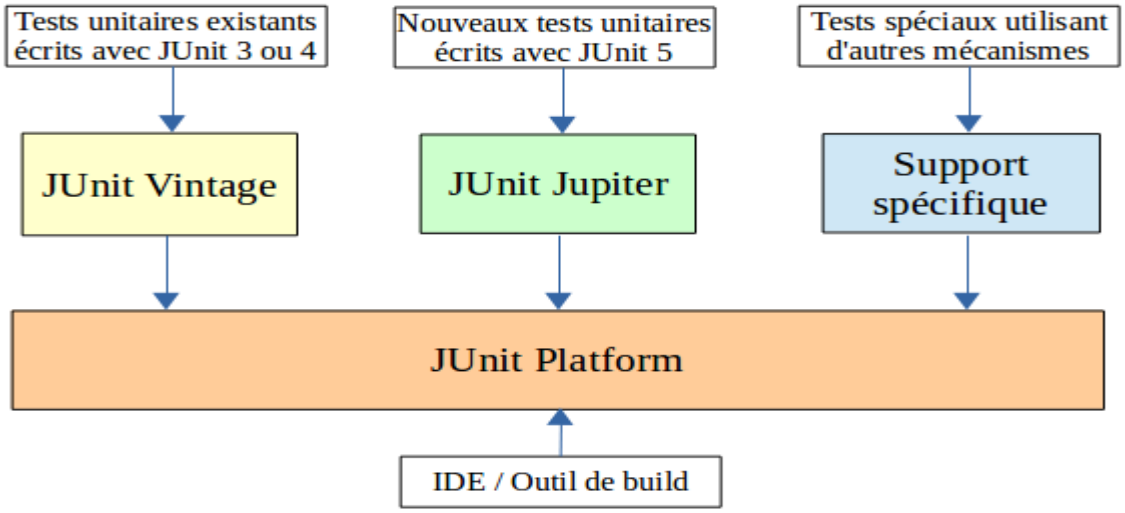


Figure 4 : Diagramme de modularité

Il s'agit d'une conception en plusieurs blocs modulaire. Chaque module correspond à un sous-ensemble de fonctionnalités reliées :

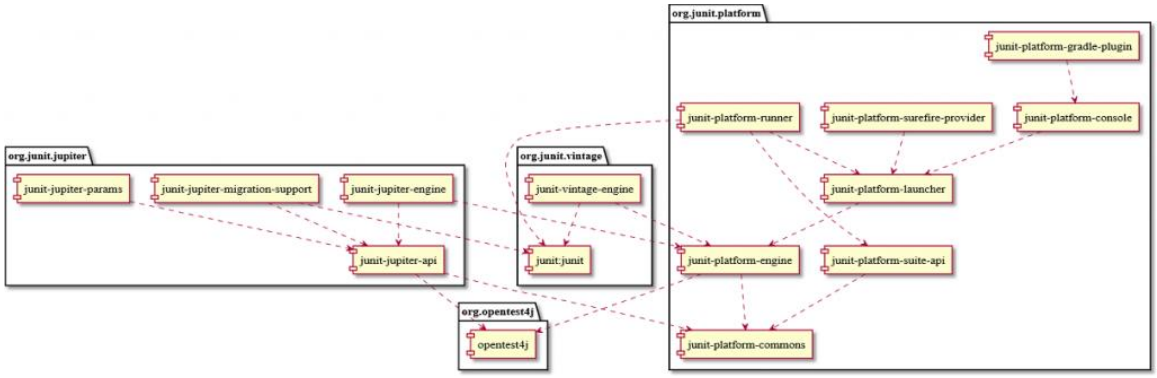


Figure 5 : Diagramme de de dépendance

Modularité :

La Solution Junit5 est bien séparée en 3 modules, Junit vintage et Junit Jupiter qui communiquent ensemble à l'aide de l'API Junit plateforme, cette architecture permet facilement à remplacer et à réutiliser ces modules.

Couplage :

L'objectif de cette architecture est de séparer les responsabilités des tests, d'exécution et d'extensions. Elle doit aussi permettre de faciliter l'intégration d'autres Framework de tests dans JUnit.

Le diagramme de dépendance/ package (Figure 5) ci-dessus nous démontre que le couplage entre les différents modules est faible, un changement dans une classe n'aura aucun impact sur les classes qui l'utilisent.
Par conséquent, un couplage faible permet de faire des changements de code sans impacter le reste de la solution.

Cohésion :

La cohésion est l'étroitesse du lien entre les différentes fonctionnalités d'une même classe.

En analysant la hiérarchie des classes de la solution Junit5 (Source : https://junit.org/junit5/docs/current/api/overview-tree.html) on constate qu'un très grand nombre de fonctionnalités de certaines classes sont essentielles à leurs fonctionnements, ce qui nous démontre que la cohésion est forte.

Conclusion :

On cherche toujours à réutiliser des composantes logicielles pour ne pas avoir à réécrire constamment du code qui remplit les mêmes fonctions.
Grace au couplage faible de JUnit5, sa forte cohésion et sa bonne modularité, on favorise la création des composantes réutilisables.

1.1.1. Comment est documentée l'architecture de l'application ?

La documentation de l'architecture de Junit5 sur leur sites officiel, Github, forums et blogs des développeurs n'est pas très détaillé, il manque surtout les diagrammes de classes, diagrammes des cas d'utilisation, ainsi le choix de l'architecture.

1.2. Dimension "Code source" et tests

1.2.1. Comment qualifiez-vous la qualité du code source ?

J'ai cloné le projet sur mon Github, afin que je puisse l'analyser avec SonarCloud puisque l'outil n'a pas d'analyseur statique du code ainsi que je ne peux pas l'analyser directement sur le repository Junit-tTam sans être membre de l'équipe.

L'outil qui sera utiliser pour faire l'analyse du code sera SonarCloud

https://sonarcloud.io/dashboard?id=MajedTP_Junitlab

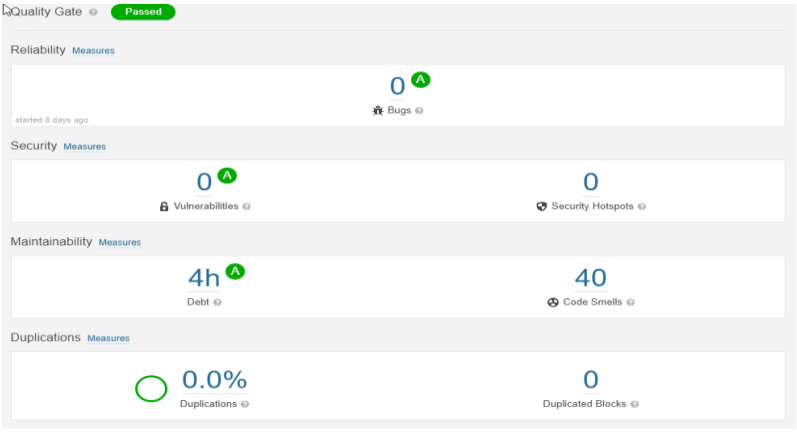


Figure 6 : Résultat d'analyse de Sonar Cloud

Pour évaluer profondément la qualité logiciel de la solution Junit, j'ai eu recours aux métriques logicielles largement utilisé par l'outil SonarCloud et suggérer par les normes de génie logiciel. Ces métriques capturent différentes caractéristiques des classes telles que le couplage, la cohésion, la complexité, l'héritage et la taille. Certaines de ces métriques ont été utilisées pour évaluer des classes logicielles, nous les désignerons par métriques logicielles. D'autres ont servi à évaluer les classes de tests unitaires, nous les désignerons par métriques de test. Nous exposons, dans cette section, les métriques qui seront utiliser et analyser dans le cas de Junit5.

- Métriques de taille :
  - LOC : compte le nombre de lignes d'instructions dans la classe mesurée

- Métriques de complexité :
  - WMC : somme les complexités cyclomatiques de toutes les méthodes de la classe mesurée
  - RFC : le nombre de ses méthodes ainsi que le nombre de méthodes (des autres classes) appelées par la classe en réponse à un message

- Métriques d'héritage :
  - DIT : compte le nombre de classes qu'il y'a entre la classe mesurée et la racine de sa hiérarchie d'héritage
  - NOC : Elle compte le nombre de classes immédiatement dérivées de la classe mesurée

- Métriques de test :
  - TLOC : compte le nombre de lignes de code d'une classe test JUnit
  - TASSERT : compte le nombre d'assertions
  - TNOO : compte le nombre de méthodes dans une classe test

Pour être certain de mes résultats j'ai analysé Junit5 avec outil autre que SonarCloud.

Cet outil s'appelle Embold et il est de même principe de Sonar (Lien et rapport sans dans le fichier readme). Les deux outils nous démontre qu'il y'a aucune violation des métriques cité ci-dessus.
https://docs.embold.io/gamma-score/#gamma-rating-system

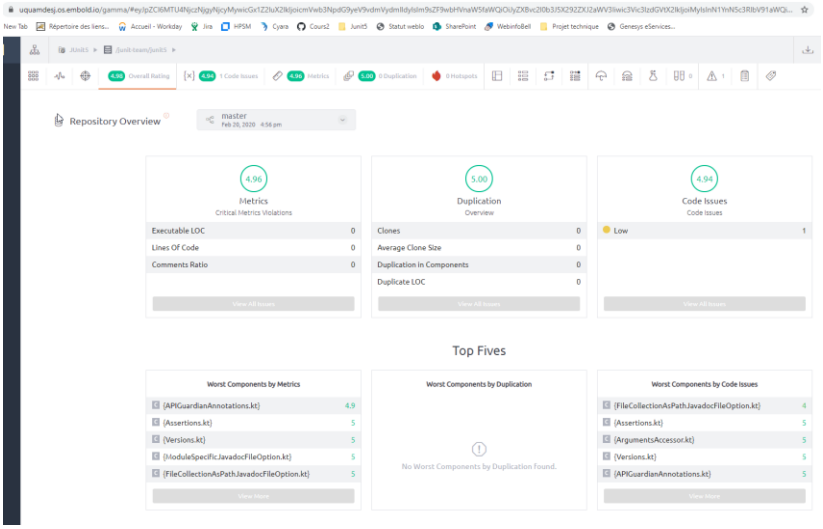
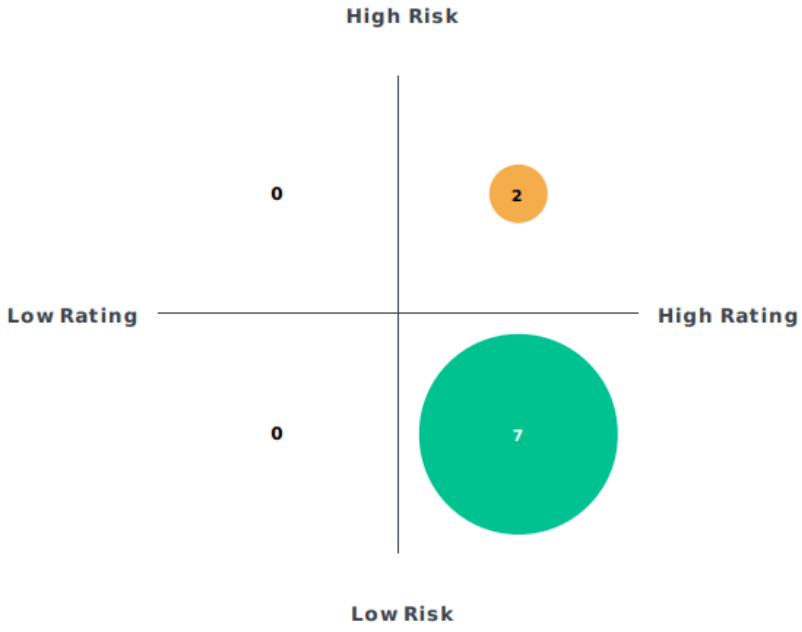


Figure 8 : Analyse Embold



Category	Count
Fragile Components (Low Rating & High Risk)	0
Potential-Risk Components I (High Risk, But High Rating)	2
Potential-Risk Components II (Low Rating, But Low Risk)	0
Stable Components (Low Risk & High Rating)	7

Figure 9 : Les risques du code source

En générale, je trouve que ce produit est très bien fait, la qualité du code est très bonne et ça respecte bien les normes ISO de génie logiciel.

1.2.2. Quels outils de construction (p.-ex. make, maven) sont utilisés?

L'outil de construction principal utilisé est Gradle https://scans.gradle.com/s/6723pxshyiv2y/tests/by-project.

Gradle est un système de gestion de build basé sur Groovy conçu spécifiquement pour la construction de projet basés sur java.

1.2.3.Quel modèle de branche est utilisé dans le développement du projet ?

Le modèle de branche utilisé dans ce projet est le modèle Supportingbranches de Github, qui est composée principalement de 5 branches :

- Master
- Develop
- Expriment
- Release Branches
- Hotfix branches (Issues et Refactor dans le cas de JUnit5)

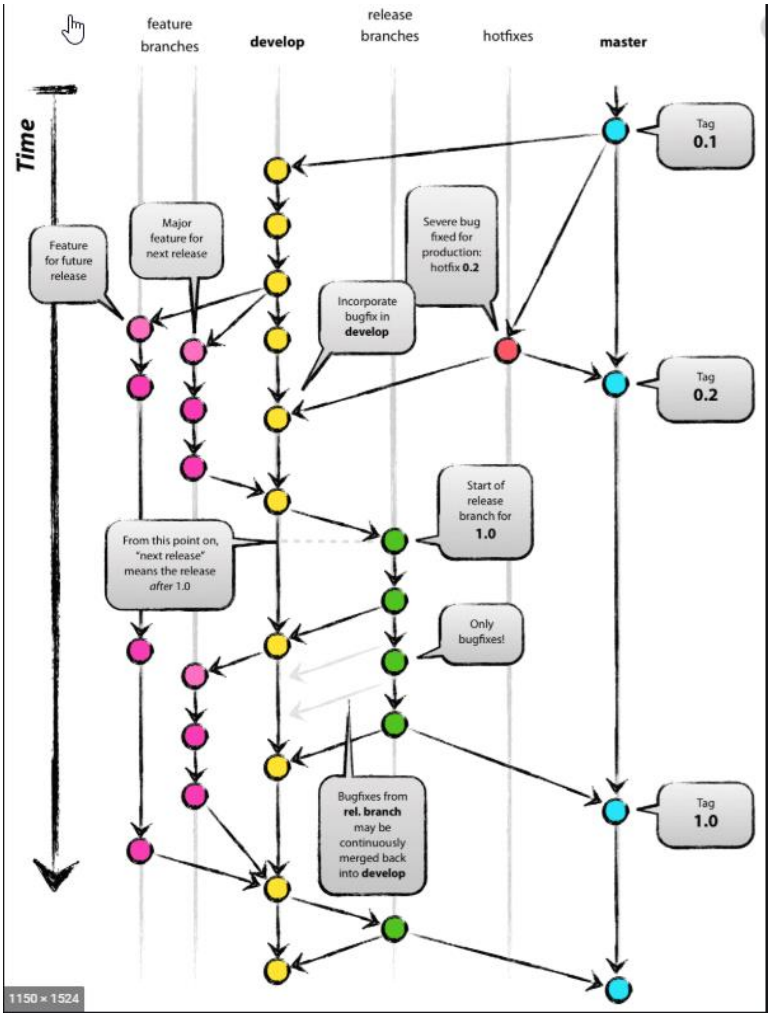


Figure 10 : Modèle de branche GIT

https://nvie.com/posts/a-successful-git-branching-model/

1.3. Dimension "Déploiement & Livraison"

1.3.1. Quels outils d'intégration / déploiement continus sont mis en place ?

L'outil d'intégration / Déploiement continus utiliser est Azure Pipelines.

Azure Pipelines est un service cloud que vous pouvez utiliser pour créer et tester automatiquement votre projet et le mettre à la disposition d'autres utilisateurs. Il fonctionne avec à peu près n'importe quel langage ou type de projet.

Azure Pipelines combine l'intégration continue (CI) et la livraison continue (CD) pour tester et créer votre code en permanence et de manière cohérente et l'expédier à n'importe quelle cible.

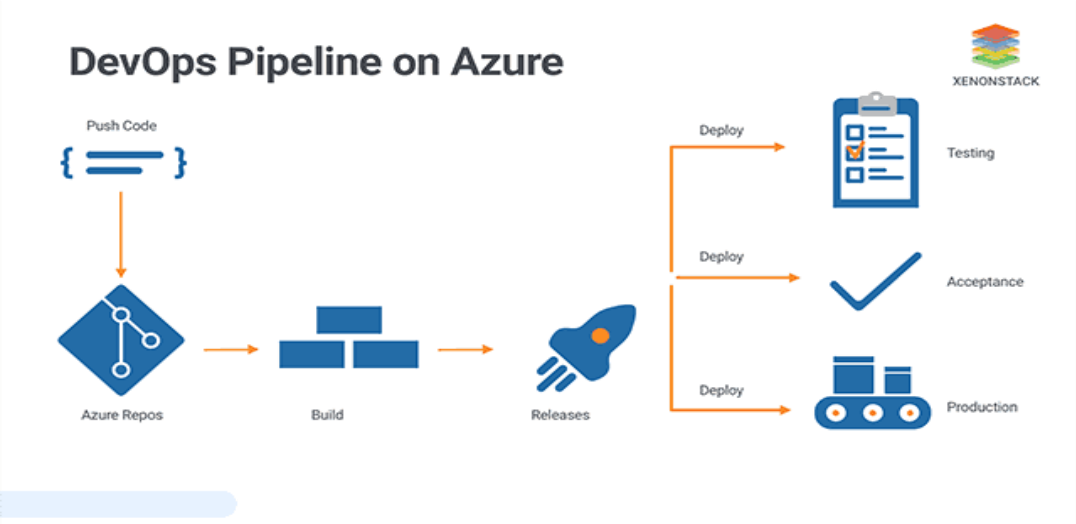


Figure7 : DevOps Pipeline on Azure

Pourquoi Azure Pipelines :

- Se déploie sur différents types de cibles en même temps
- S'intègre à GitHub
- S'intègre aux déploiements Azure
- Construit sur des machines Windows, Linux ou Mac
- Fonctionne avec des projets open-source.

1.3.2. Comment sont gérées les dépendances logicielles dans le projet ?

JUnit 5 utilise des fonctionnalités de Java 8 donc pour l'utiliser, il est nécessaire d'avoir une version 8 ou ultérieure de Java. La version 1.0 de JUnit 5 est diffusée en septembre 2017.
JUnit 5 est livré sous la forme de plusieurs jars qu'il faut ajouter au classpath en fonction des besoins. Le plus simple est d'utiliser Maven.

Group ID	Version	Artefact ID
org.unit.jupiter	5.0.0	junit-jupiter-api API pour l'écriture des tests avec JUnit Jupiter junit-jupiter-engine Implémentation du moteur d'exécution des tests JUnit Jupiter junit-jupiter-params Support des tests paramétrés avec JUnit Jupiter.
org.junit.platfom	1.0.0	junit-platform-commons Utilitaires à usage interne de JUnit junit-platform-console Support pour la découverte et l'exécution des tests JUnit dans la console junit-platform-console-standalone Jar exécutable qui contient toutes les dépendances pour exécuter les tests dans une console junit-platform-engine API publique pour les moteurs d'exécution des tests junit-platform-gradle-plugin Support pour la découverte et l'exécution des tests JUnit avec Gradle junit-platform-launcher Support pour la découverte et l'exécution des tests JUnit avec des IDE et des outils de build junit-platform-runner Implémentation d'un Runner pour exécuter des tests JUnit 5 dans un environnement JUnit 4 junit-platform-suite-api Support pour l'exécution des suites de tests junit-platform-surefire-provider Support pour la découverte et l'exécution des tests JUnit avec le plugin Surefire de Maven
org.junit.vintage	4.12.0	junit-vintage-engine Implémentation d'un moteur d'exécution des tests écrits avec JUnit 3 et 4 dans la plateforme JUnit 5

Sources :

https://blogs.apache.org/netbeans/entry/junit-5-apache-ant-and

https://fr.slideshare.net/sbrannen/junit-5-evolution-and-innovation-springone-platform-2019

https://sonarcloud.io/dashboard?id=MajedTP_Junitlab

https://blog.codefx.org/design/architecture/junit-5-architecture-jupiter/

https://junit.org/junit5/docs/current/user-guide/

https://scans.gradle.com/s/bl3pw4mrbgsao/tests/by-project

https://github.com/junit-team/junit5

https://www.codeflow.site/fr/article/junit-5-preview

https://blog.ippon.fr/2017/05/15/junit-5/

https://www.infoq.com/presentations/junit-5-new/