Conduite de Projet – 2019/2020

Industrialisation dans le développement logiciel

Guillaume Kulakowski
Architecte Technique en solutions open-source @CGI

https://twitter.com/llaumgui

https://www.linkedin.com/in/guillaumekulakowski

guillaume@kulakowski.fr

https://blog.kulakowski.fr

CGI

UNIVERSITÉ DE MONTPELLIER

Agenda

1. Présentation.
2. Plan.
3. Code : Les VCS.
4. Build, Test & Release : Plateforme d'Intégration Continue.
5. Deploy : Déploiement Continu.
6. Operate.
7. Monitor.
8. Outils divers.

1 - Présentation

- Présentation de l'intervenant.
- Présentation du cours.

Le DevOps nouveau nom et nouvelle référence

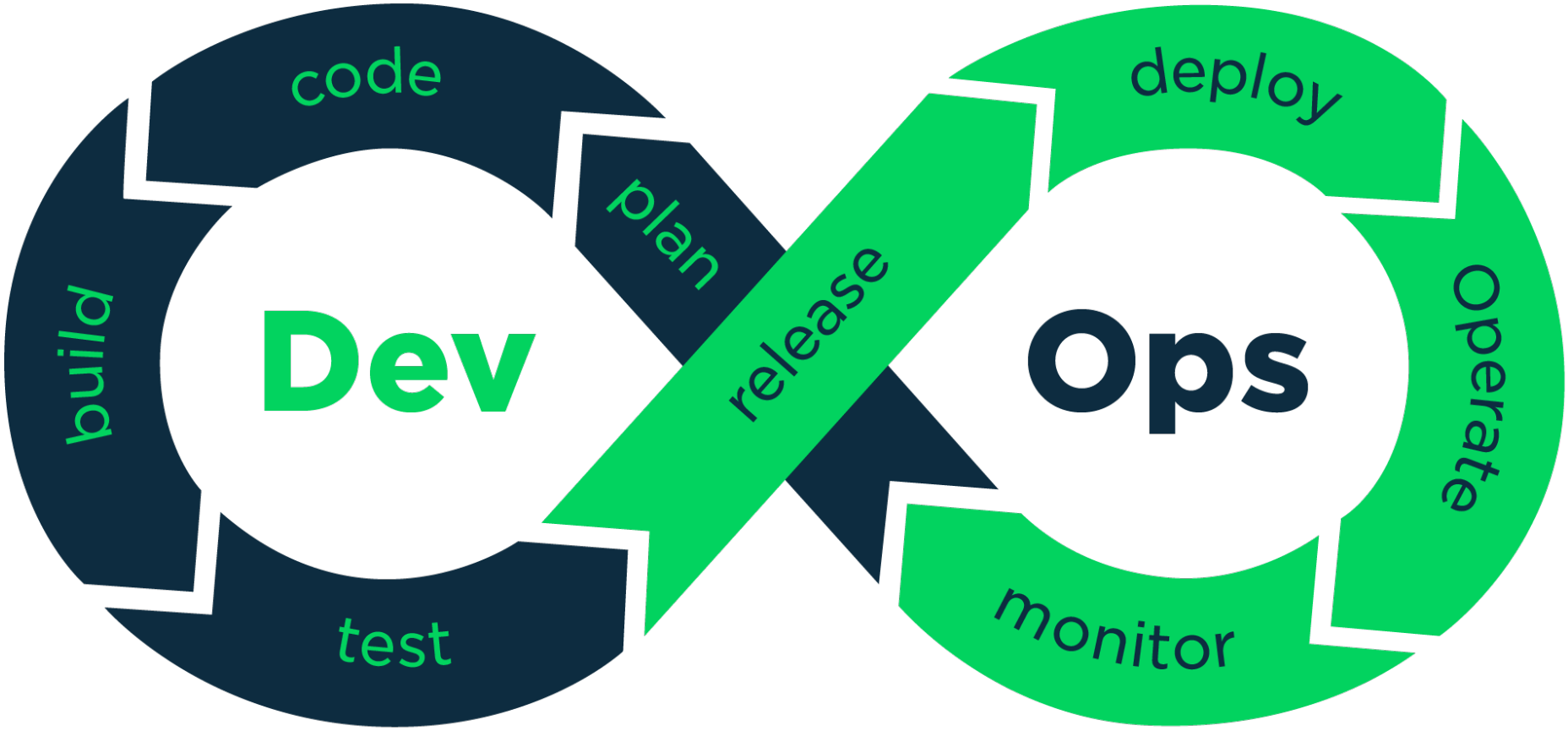
Le devops est un mouvement visant à l'alignement de l'ensemble des équipes du système d'information sur un objectif commun, à commencer par les équipes de dev ou dev engineers chargés de faire évoluer le système d'information et les ops ou ops engineers responsables des infrastructures (exploitants, administrateurs système, réseau, bases de données,...).

Ce qui peut être résumé par : travailler ensemble pour produire de la valeur pour l'entreprise.

©Wikipédia

Le DevOps nouveau nom et nouvelle référence dans l'industrialisation du développement logiciel

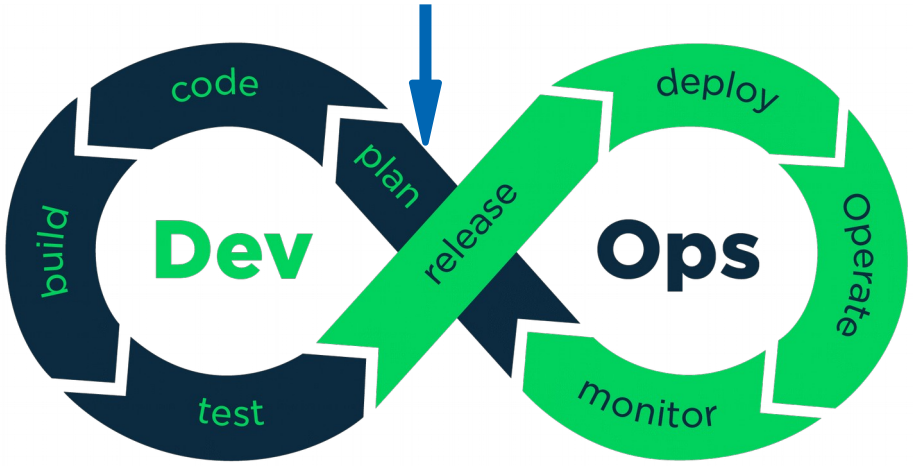
Une représentation qui semble faire l'unanimité



2 – Plan

- Définition.
- Exemple.

Le cycle DevOps

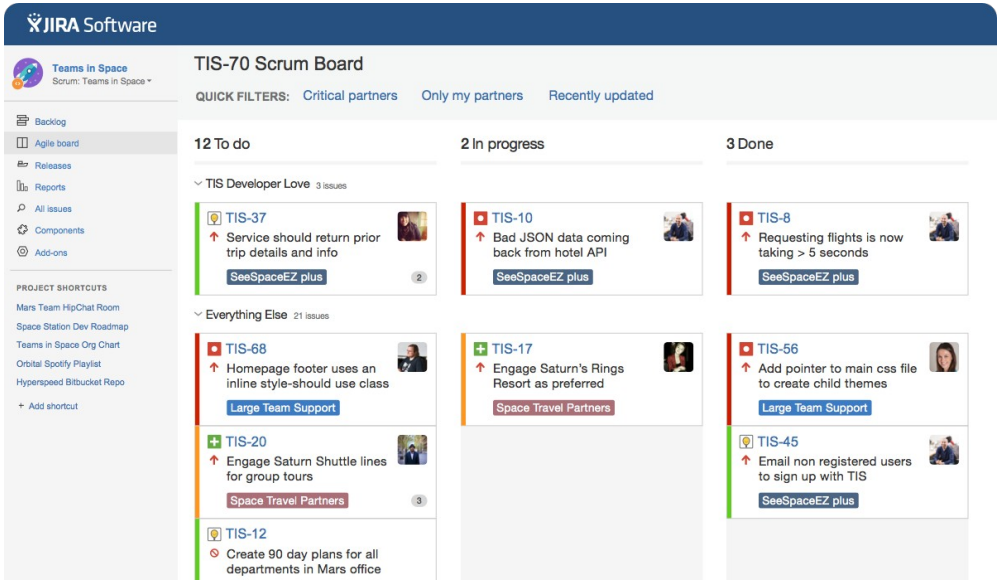


Jira

Les outils de planification

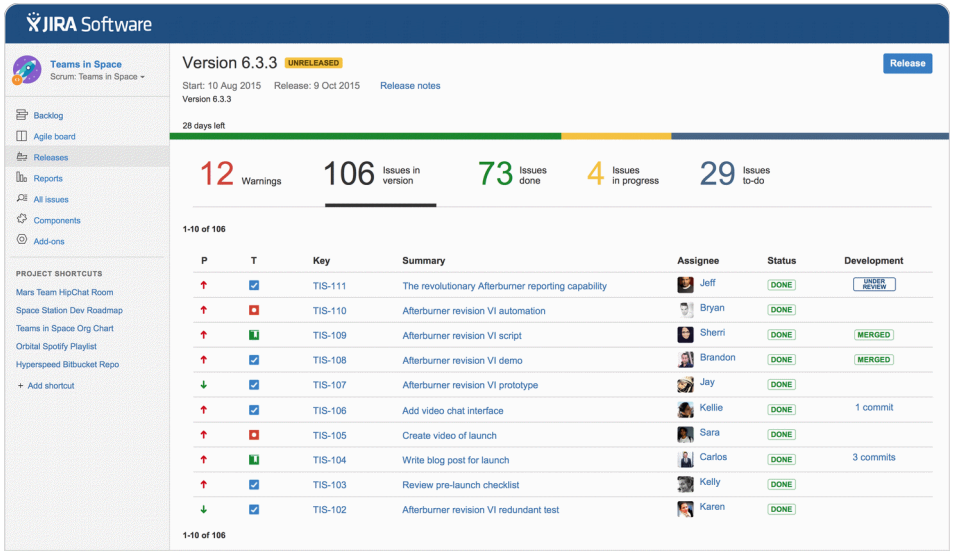
Planifier est composé en fait en deux parties : « **Définir** » et « **planifier** ». Cette étape concerne la valeur commerciale et les exigences des applications. Plus précisément, les activités "Plan" comprennent :

- Indicateurs, objets, et retour d'expérience de production
- Exigences
- Mesures commerciales
- Métriques de mise à jour et de mise en production
- Plan, timing et analyse de rentabilisation des releases
- Politique de sécurité



Une utilisation classique : *bugtracker*

- Malgré tout notre outillage, il est p~~ossible~~robable qu'un (ou plusieurs) bug(s) ai(en)t échappé(s) à notre vigilance.
- L'outil de planning sert alors au ticketing pour renseigner les incidents.
- Chaque outil possède sont workflow.
- Dans le cadre d'une anomalie, il ne faut pas oublier que la première étape est toujours la **qualification** :
  - Est-ce vraiment une anomalie ?
  - Est-ce une évolution ?
  - Est-ce une anomalie en garantie ?
  - Etc...



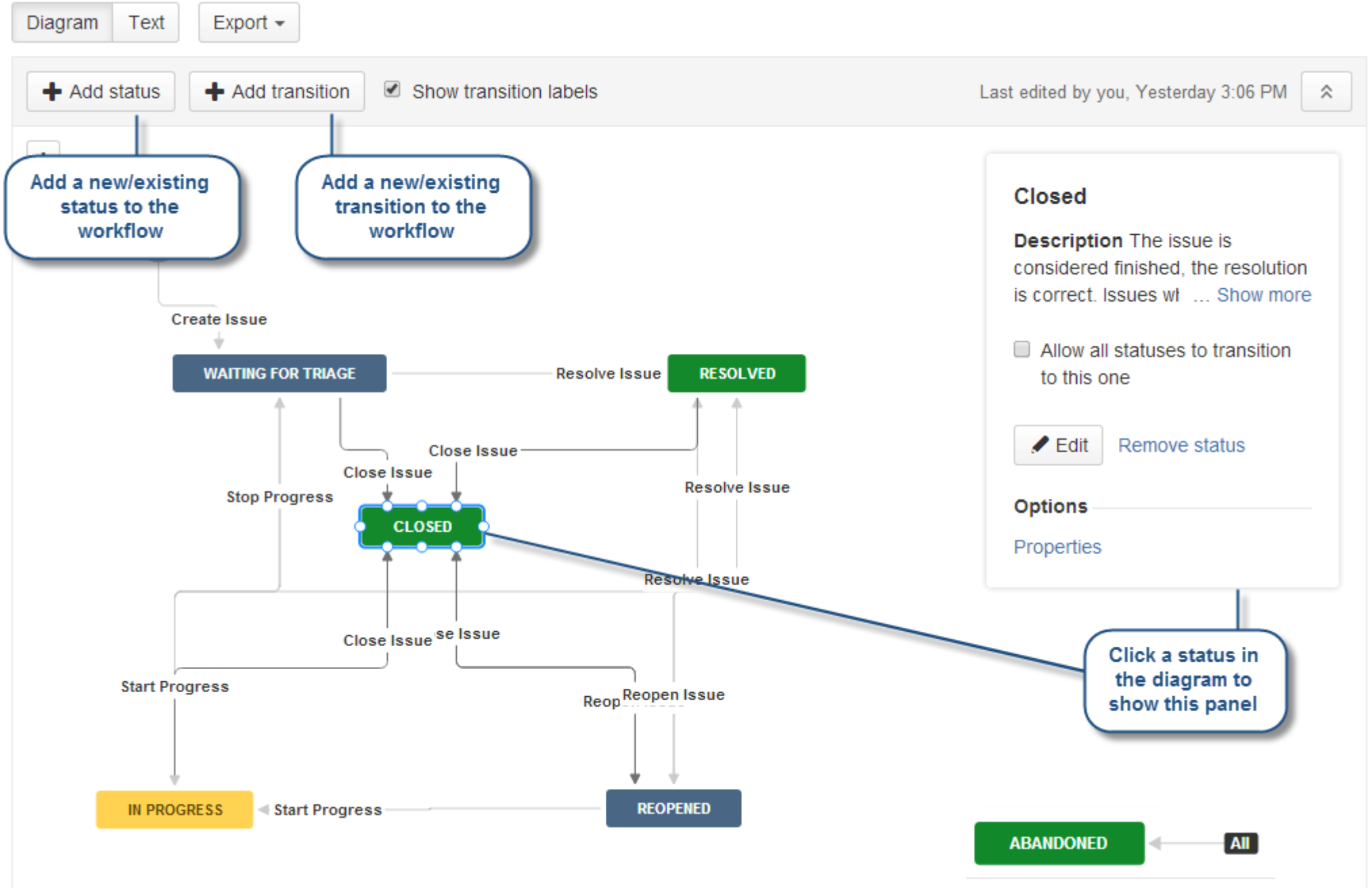
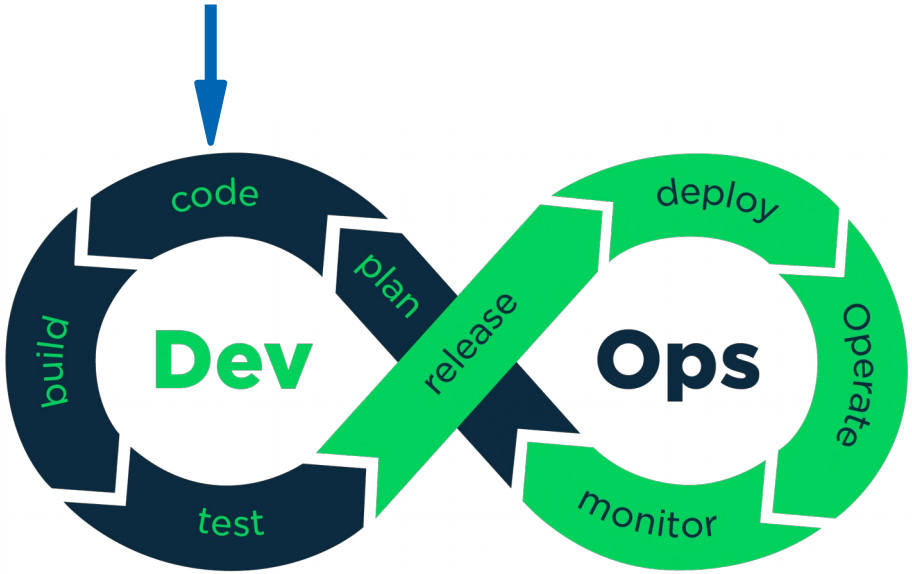


Diagram
Text
Export
Add status
Add transition
Show transition labels
Last edited by you, Yesterday 3:06 PM
Add a new/existing status to the workflow
Add a new/existing transition to the workflow
Create Issue
WAITING FOR TRIAGE
Resolve Issue
RESOLVED
Close Issue
Close Issue
Stop Progress
Resolve Issue
CLOSED
Resolve Issue
Close Issue
se Issue
Start Progress
Reop
Reopen Issue
IN PROGRESS
Start Progress
REOPENED
ABANDONED
All
Closed
Description The issue is considered finished, the resolution is correct. Issues wl
... Show more
Allow all statuses to transition to this one
Edit
Remove status
Options
Properties
Click a status in the diagram to show this panel

3 – Code: Les VCS

- Définition.
- Les différents types de VCS.
- Focus sur GIT.
- Les workflows.

Le cycle DevOps



Définition

Un **logiciel de gestion de versions** (ou **VCS** en anglais, pour ***V****ersion* ***C****ontrol* ***S****ystem*) est un logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes.

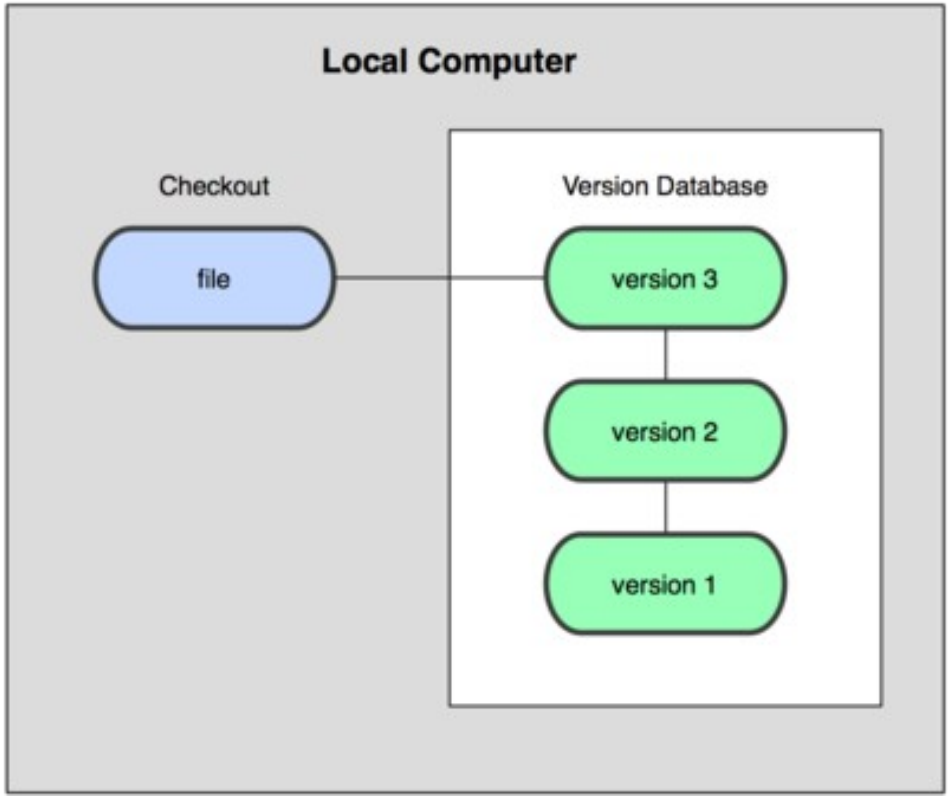
Il existe aussi des logiciels et services de gestion de versions **décentralisé** (distribué) (ou **DVCS** en anglais, pour ***D****istributed* ***V****ersion* ***C****ontrol* ***S****ystem*). Git et Mercurial sont deux exemples de logiciel de gestion de versions décentralisé et sont disponibles sur la plupart des systèmes Unix et Windows.

Les logiciels de gestion de versions sont utilisés notamment en développement logiciel pour conserver le code source relatif aux différentes versions d'un logiciel. *(© Wikipedia)*

Répond à des besoins

- **Partager** des sources en équipe.
- **Partager** des sources avec tout le monde (open source).
- **Historique** des sources.
- **Tuiler**, c'est à dire gérer plusieurs phase (branches) ou versions d'un projet en parallèle.

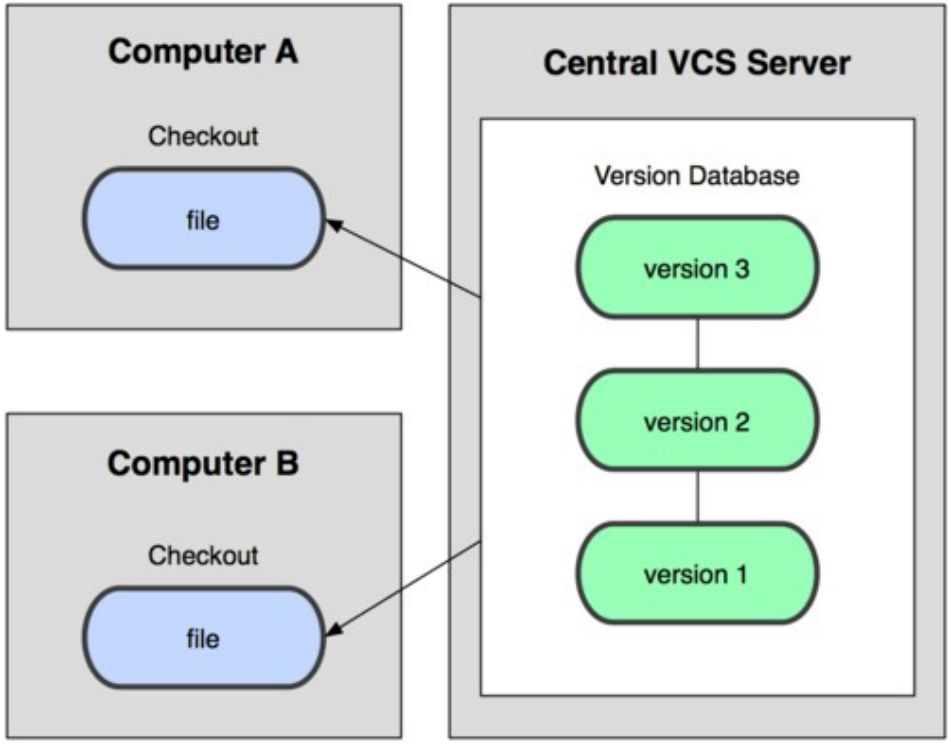
Type #1 : Gestion de versions locale



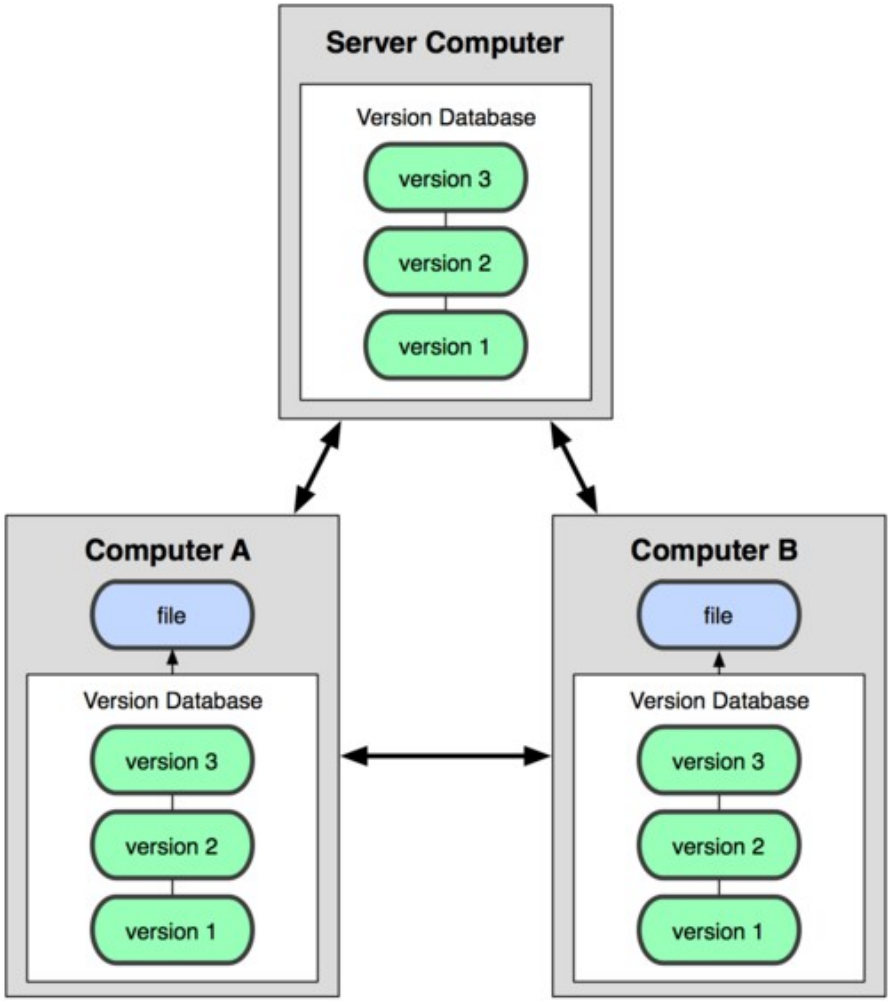
- Exemple : RCS
- Système plus répandu (heureusement) de nos jours.
- Ce schéma ressemble à ce que vous propose vos IDE avec l'historique local.

Type #2 : Gestion de versions centralisée

- VSC.
- Exemple SVN.
- Tout passe par le serveur central : SPOF.
- Les versions locales (et donc poussées) sont anarchiques et incohérentes (possibilité d'*updater* une arborescence partielle).



Type #3 : Gestion de versions décentralisée



- DVSC.
- Exemple GIT, Hg, Bz, etc.
- Serveur local clone du distant (rapidité).
- Les versions locales (et donc poussées) sont cohérentes car on ne peut pousser si le dépôt n'est pas en *phase*.
- Puissant donc complexe à utiliser.
- Vraie gestion de branche.

Focus sur GIT : Historique

- Une histoire liée au noyau **Linux** et à **Linus Torvalds**.
- En **2002**, le projet du noyau Linux commence à utiliser un DVCS propriétaire : BitKeeper.
- En **2005**, clash entre la communauté & BitKeeper qui passe sur un modèle payant.
- Suite à ça, toujours en **2005**, La réponse de la communauté & de Linus fut la **création de GIT**.

Focus sur GIT : ADN

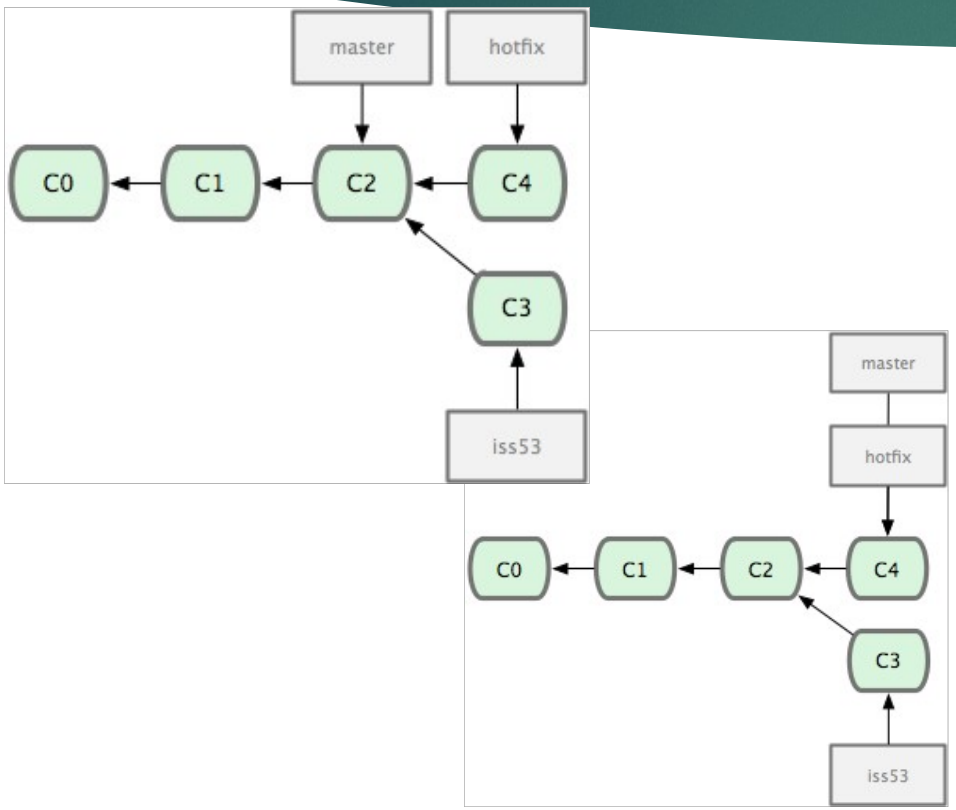
Le cahier des charges :

- **Vitesse**.
- Conception **simple**.
- Support pour les développements non linéaires (**milliers de branches** parallèles).
- Complètement **distribué**.
- Capacité à **gérer efficacement des projets d'envergure** tels que le noyau Linux (vitesse et compacité des données).

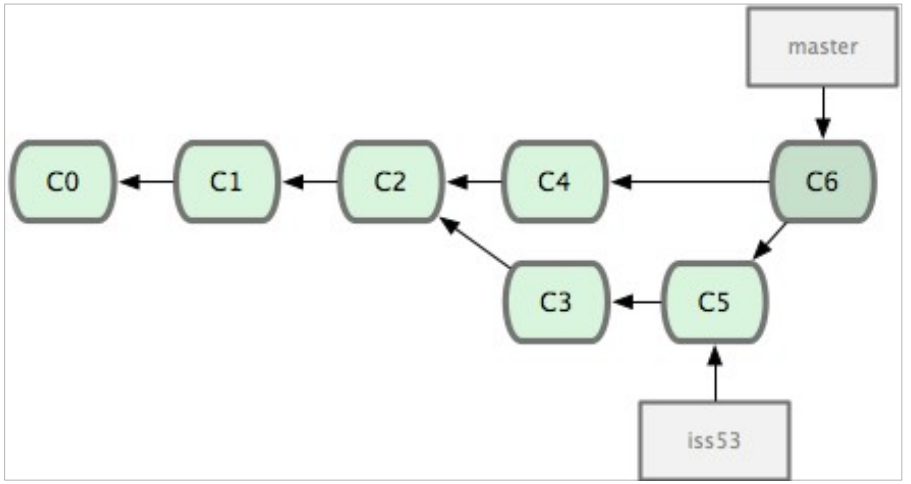
Focus sur GIT : les killer features

- **rebase :** introduire les commit d'une autre branche.
- **reset :** annuler des commits non poussés.
- **revert :** retour en arrière.
- **amend :** amender le dernier commit.
- **bissect :** localiser le commit foireux par dichotomie.
- **stash :** remiser votre code.
- **squash :** fusion de commits.

2 types de merges



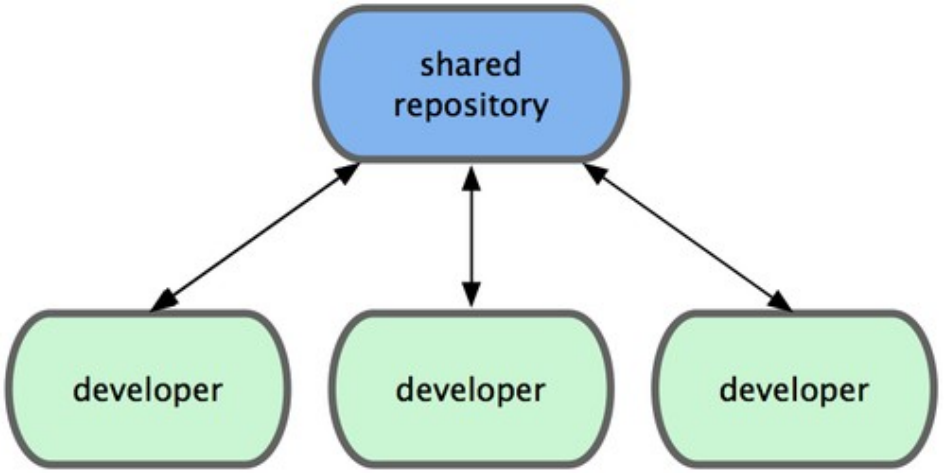
Le merge dit "fast forward" est une simple avance rapide, un déplacement de pointeur.



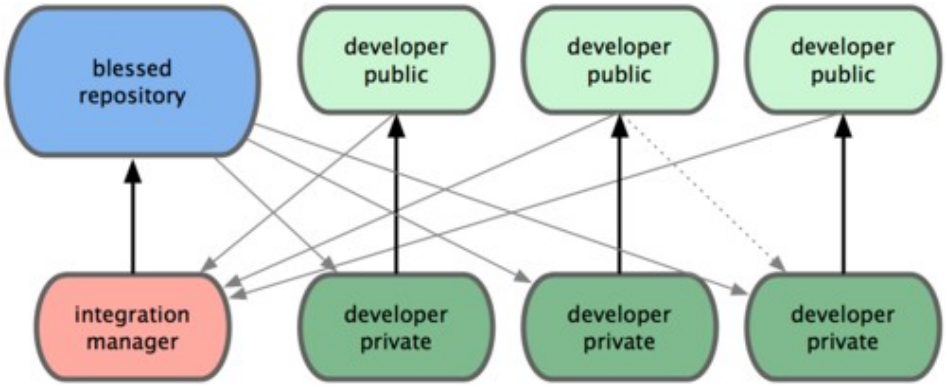
Le merge classique avec création d'un commit de merge.

Workflow #1 : Mode distribué

- Le plus utilisé car le plus simple.
- Proche du VSC.
- Tous les développeurs *push* sur le même dépôt distant.



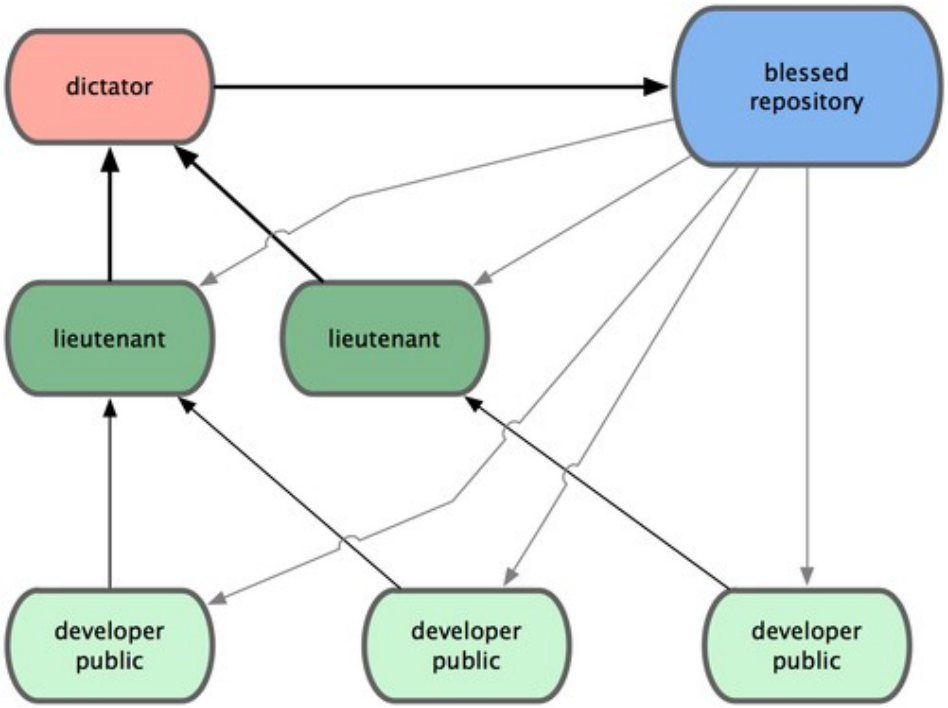
Workflow #2 : Gestionnaire d'intégration



- Approche proche du principe de *Pull Request* de GitHub.
- 1 gardien du temple vérifie les *PR* avant de les *merger* dans le dépôt officiel.

Workflow #3 : Dictateur

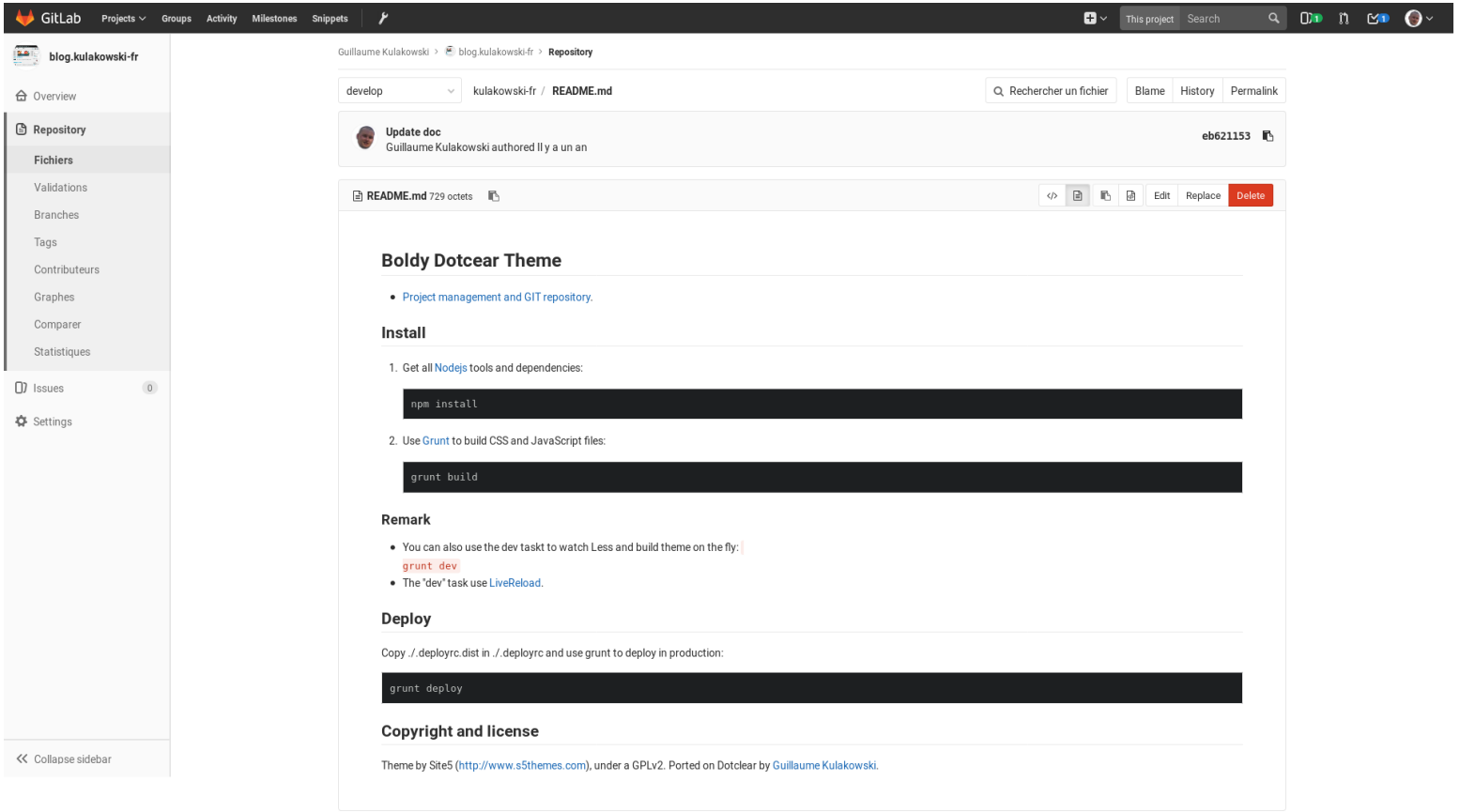
- La méthode la plus efficace pour gérer les gros projets subdivisés en plusieurs sous-projets eux même subdivisés en d'autres projets…
- Exemple Linux.



Workflow #4 : GIT-Flow

- Plus une méthode de gestion de branches qu'un workflow.
- Utilisation de branches sémantiques :
  - **master :** code de production. On ne développe rien sur cette branche.
  - **develop :** code prêt pour le *merge* pour la nouvelle release. On peut également corriger, améliorer de petites fonctionnalités.
  - **feature/xxx :** développement des nouvelles fonctionnalités. Une fois terminé, on *merge* les changements dans *develop*.
  - **release/vx.y.z :** Préparation d'une nouvelle release. Un *merge* sera alors effectué dans master.
  - **hotfix :** permet de réparer tout de suite un bug critique en production.

Gérer votre projet et votre VCS via une forge : Exemple de GitLab

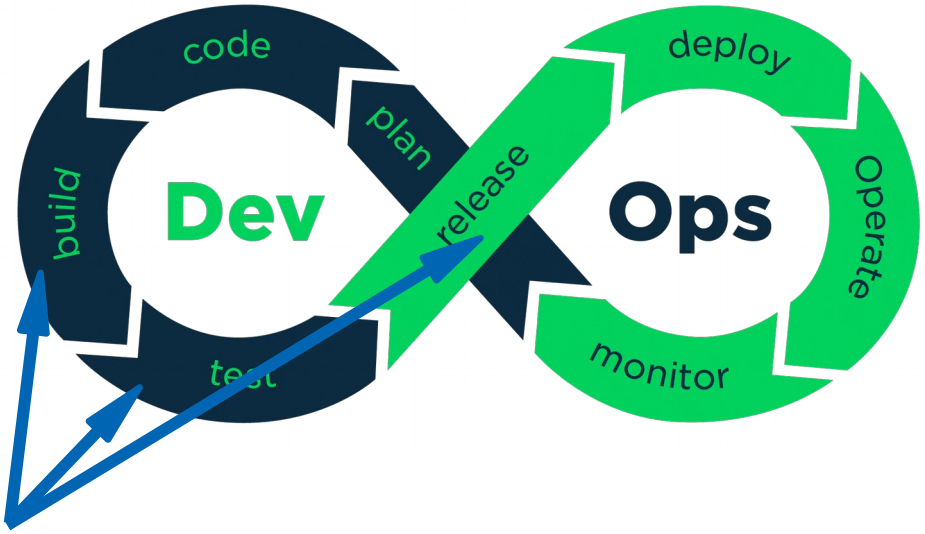


Aller plus loin...

4 – Build, Test & Release Plateforme d'Intégration Continue

- Définition
- Exemple
- Outil de qualimétrie
- Métriques

Le cycle DevOps



Jenkins

maven

COMPOSER

sonarqube

Définition

L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

Le concept a pour la première fois été mentionné par Grady Booch et se réfère généralement à la pratique de l'extreme programming. Le principal but de cette pratique est de détecter les problèmes d'intégration au plus tôt lors du développement. De plus, elle permet d'automatiser l'exécution des suites de tests et de voir l'évolution du développement du logiciel.

L'intégration continue est de plus en plus utilisée en entreprise afin d'améliorer la qualité du code et du produit final.*(© Wikipedia)*

Concrètement ?

Un PIC ça sert à quoi ?

- A construire (build, compilation, etc.).
- A générer la documentation.
- A lancer des **tests de qualité** (QA, qualimétrie).
- A lancer des **tests d'intégration** (TI) automatisés.
- A lancer des **tests de non-régression** (TNR) automatisés.
- A lancer des **tests de unitaires** (TU) automatisés.
- A construire un **paquet** (et même à le déployer).
- A lancer des **outils divers** (tests, sondes, etc.).
- Etc…

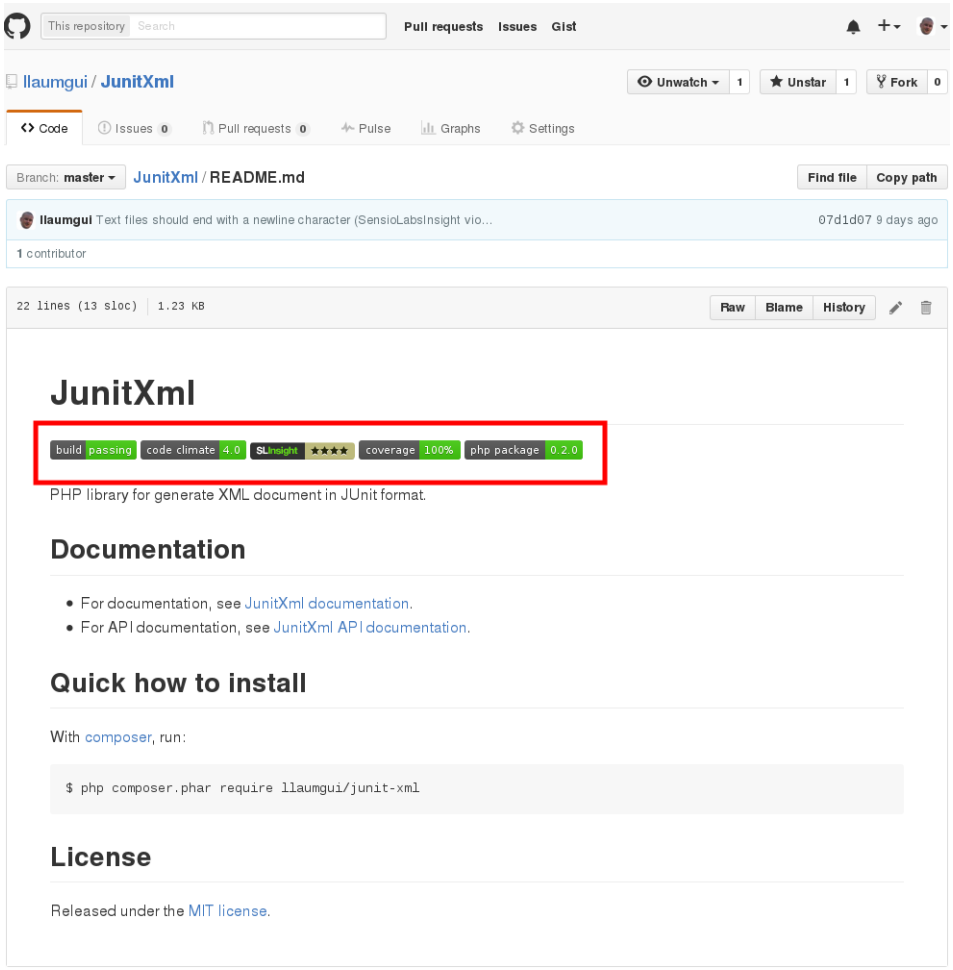
Le rôle central de l'ordonnanceur

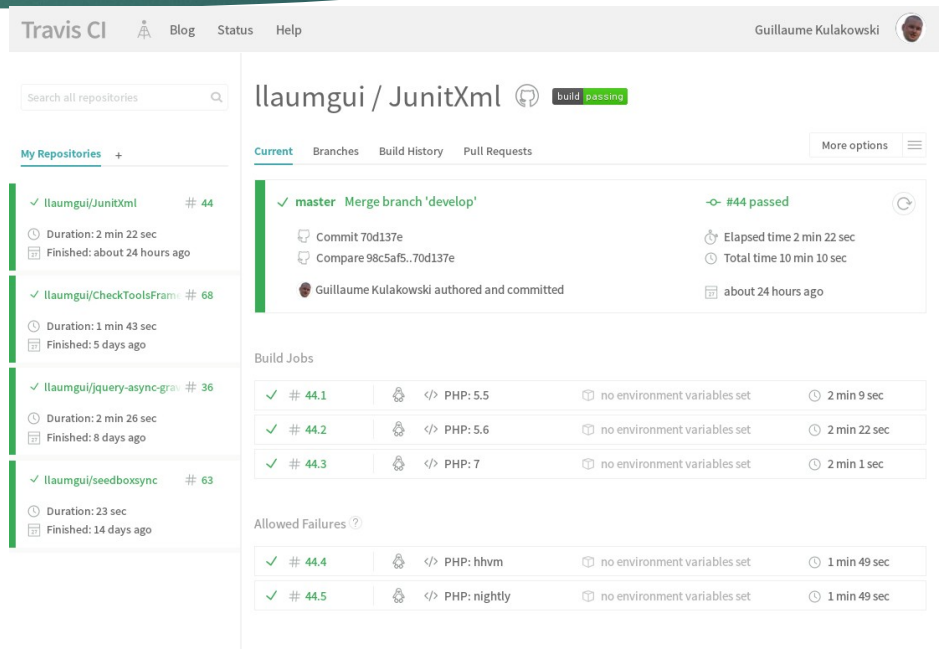
Comme le montre le choix du logo de Jenkins, une PIC ne fait pas grande chose, elle se base sur un ordonnanceur pour faire le travail.

- ANT / Maven / Gradle : ordonnanceur JAVA.
- Phing / Composer : ordonnanceur PHP.
- Etc...

C'est eux qui au travers de leur langage de script vont faire les appels aux outils (tests, qualimétrie, etc.).

Exemple de PIC #1 : Travis





✓ master Merge branch 'develop'

#44.2 passed

Commit 70d137e

Compare 98c5af5..70d137e

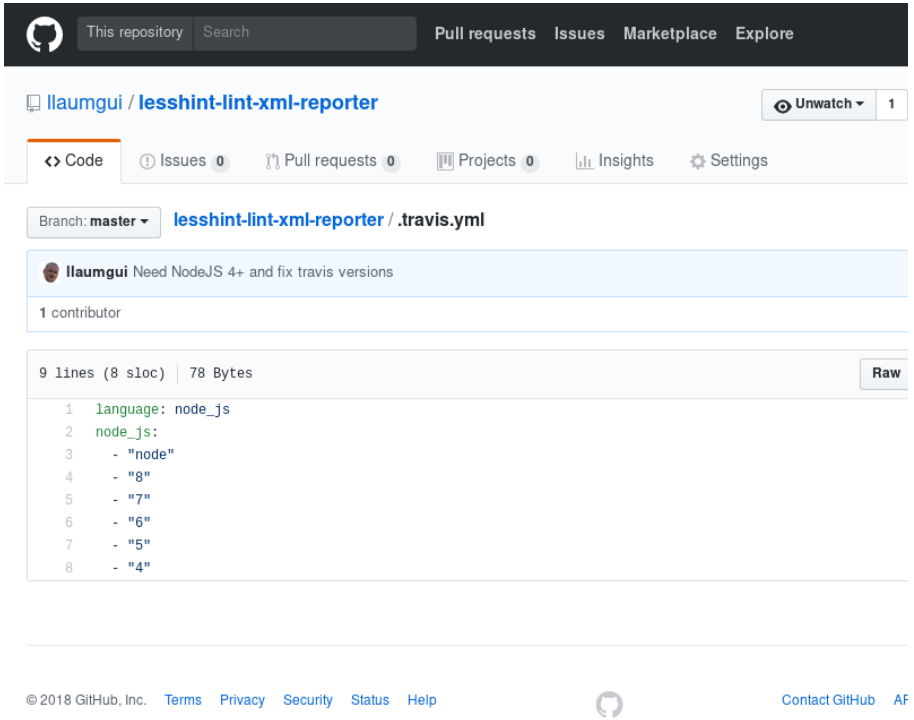
Elapsed time 2 min 22 sec

about 24 hours ago

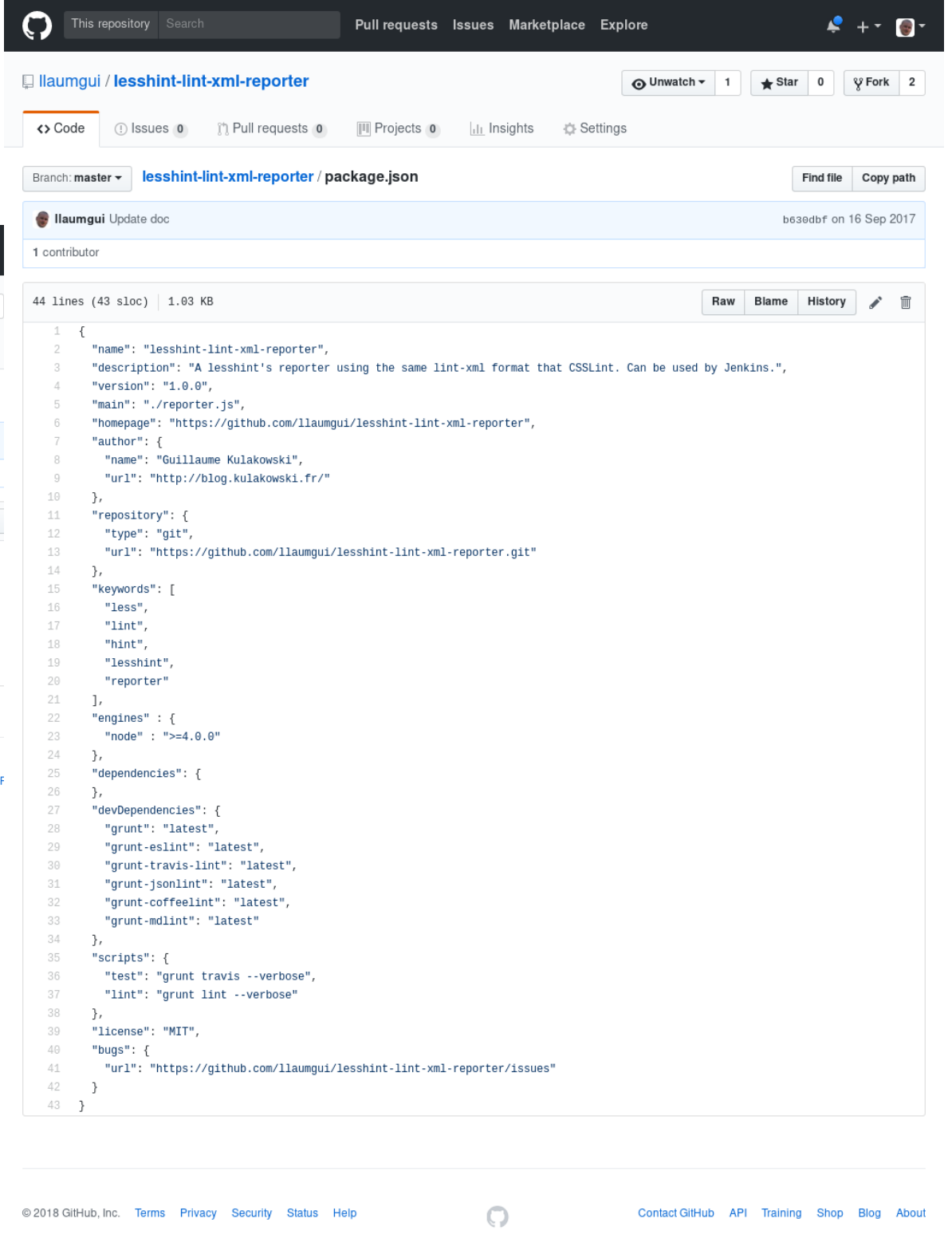
Guillaume Kulakowski authored and committed

Remove log | Raw log

```
1  Using worker: worker-linux-docker-9ec0601a.prod.travis-ci.org:travis-linux-1
2
3  Build system information                                                    system_info
67
68 $ export DEBIAN_FRONTEND=noninteractive                                     fix_CVE-2015-7547
108 $ git clone --depth=50 --branch=master https://github.com/llaumgui/JunitXml.git   git.checkout   0.50s
119
120 This job is running on container-based infrastructure, which does not allow use of 'sudo', setuid and setgid
    executables.
121 If you require sudo, add 'sudo: required' to your .travis.yml
122 See https://docs.travis-ci.com/user/workers/container-based-infrastructure/ for details.
123
124 Setting environment variables from repository settings
125 $ export CODECLIMATE_REPO_TOKEN=[secure]
126 $ export GH_TOKEN=[secure]
127
128 $ phpenv global 5.6 2>/dev/null                                                0.03s
129
130 $ php --version
131 PHP 5.6.5 (cli) (built: Feb 12 2015 01:41:10)
132 Copyright (c) 1997-2014 The PHP Group
133 Zend Engine v2.6.0, Copyright (c) 1998-2014 Zend Technologies
134     with Zend OPcache v7.0.4-dev, Copyright (c) 1999-2014, by Zend Technologies
135     with Xdebug v2.2.7, Copyright (c) 2002-2015, by Derick Rethans
136 $ composer --version
137 Warning: This development build of composer is over 30 days old. It is recommended to update it by running "/home/travis
    /.phpenv/versions/5.6/bin/composer self-update" to get the latest version.
138 Composer version 1.0-dev (1d8f05f1dd0e390f253f79ea86cd505178360019) 2015-02-11 11:31:57
139
140 $ composer self-update                                                  before_install   1.70s
145 $ composer install --prefer-source                                     before_script    93.51s
248 $ ./vendor/bin/phpunit --configuration phpunit.xml.dist --coverage-text --coverage-clover build/logs/clover.xml   2.03s
    --coverage-html build/coverage
249 PHPUnit 4.8.23 by Sebastian Bergmann and contributors.
250
251 ....
252
253 Time: 1.54 seconds, Memory: 9.75Mb                                                          Top
254
255 OK (4 tests, 18 assertions)
256
257 Generating code coverage report in Clover XML format ... done
258
259 Generating code coverage report in HTML format ... done
260
261
262 Code Coverage Report:
263   2016-03-01 19:42:21
264
265  Summary:
266   Classes: 100.00% (5/5)
267   Methods: 100.00% (18/18)
268   Lines:   100.00% (74/74)
269
270 \Llaumgui\JunitXml::JunitXmlTestCase
271   Methods: 100.00% (1/1)   Lines: 100.00% (2/2)
272 \Llaumgui\JunitXml::JunitXmlTestElement
273   Methods: 100.00% (1/1)   Lines: 100.00% (3/3)
274 \Llaumgui\JunitXml::JunitXmlTestSuite
275   Methods: 100.00% (1/1)   Lines: 100.00% (22/22)
276 \Llaumgui\JunitXml::JunitXmlTestSuites
277   Methods: 100.00% (1/1)   Lines: 100.00% (3/3)
278 \Llaumgui\JunitXml::JunitXmlValidation
279   Methods: 100.00% (14/14)   Lines: 100.00% (44/44)
280
281
282 The command "./vendor/bin/phpunit --configuration phpunit.xml.dist --coverage-text --coverage-clover build/logs
    /clover.xml --coverage-html build/coverage" exited with 0.
283 $ ./vendor/bin/phpcs                                                                        0.57s
284
285
286 The command "./vendor/bin/phpcs" exited with 0.
287 $ if  [ $TRAVIS_PHP_VERSION = '5.6' ] && [ $TRAVIS_BRANCH = 'master' ] && [                 after_script.1   0.20s
290 $ if  [ $TRAVIS_PHP_VERSION = '5.6' ] && [ $TRAVIS_BRANCH = 'master' ] && [                 after_script.2   8.26s
393
394 Done. Your build exited with 0.
```



https://github.com/llaumgui/lesshint-lint-xml-reporter



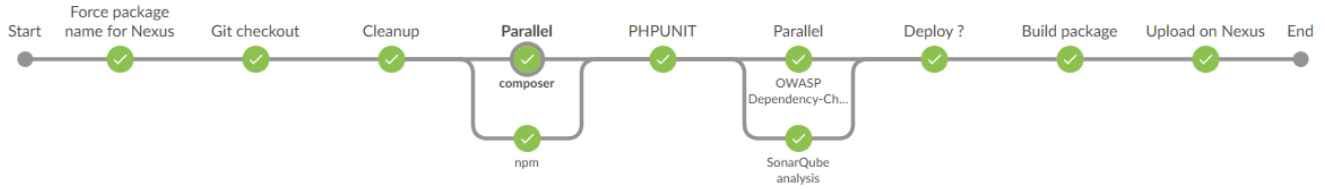
SmoovenGO / Front web 906

Pipeline | Modifications | Tests | Artefacts | Déconnexion

Branche: develop
Commit: 1fae342

11m 39s
an hour ago

Modifications par remy.bannes
Branch indexing



Parallel / composer - <1s

- Shell Script — 1m 11s
- Shell Script — 9s

```
[application] Running shell script
+ composer71 ci
> vendor/bin/phpcs --config-set installed_paths vendor/escapestudios/symfony2-coding-standard
Using config file: /var/jenkins_home/workspace/smoovengo_front_web_develop-ODMHGB2A656HXWMIOC4E3YURWVEZG66WUXYRWYZWU2ERZUIRIR2Q/application/vendor/squizlabs/php_codesniffer/CodeSniffer.conf

Config value "installed_paths" added successfully
> vendor/bin/phpcs --standard=Symfony --ignore="*/tests/*,*/data/*,*/Tests/*" --report=checkstyle --report-file=../logs/checkstyle.xml src
Time: 7.16 secs; Memory: 20Mb

Script vendor/bin/phpcs --standard=Symfony --ignore="*/tests/*,*/data/*,*/Tests/*" --report=checkstyle --report-file=../logs/checkstyle.xml src handling the phpcs event returned with error code 2
```

- Publier les résultats de l'analyse Checkstyle — 1s
- injected_BRANCH="develop" composer71 sami — Shell Script — 2s
- Publish HTML reports — <1s

Métrique : outil de type « Checkstyle »

- Exemple : Checkstyle (JAVA), PHPCS (PHP), etc.
- Contrôle la cohérence du style d'un code source :
  - Choix des noms de classe et fonction,
  - Indentation,
  - Nom des fichiers,
  - Namespacing,
  - Etc...

Métrique : duplication de code

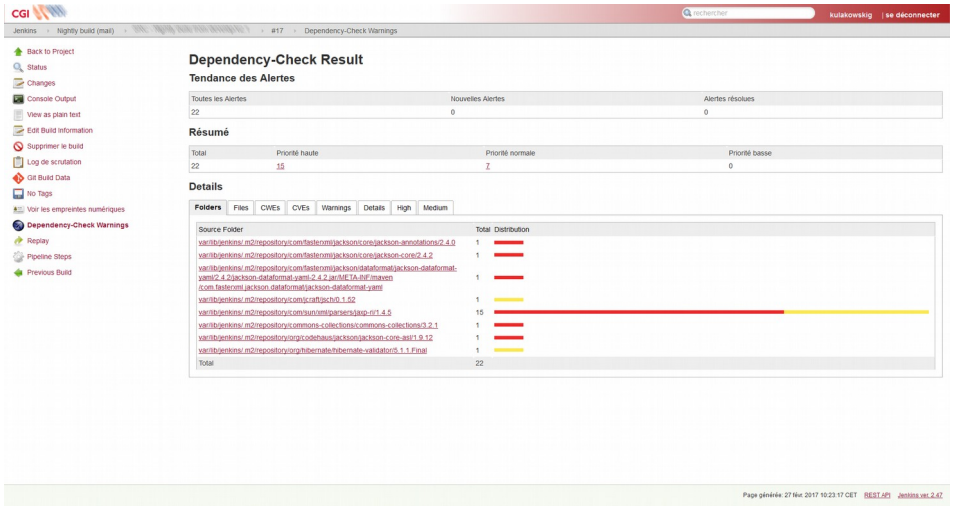
- Exemple PHPCPD, PMD
- Permet de détecter les copier/coller et les manques de factorisation au sein d'un projet.

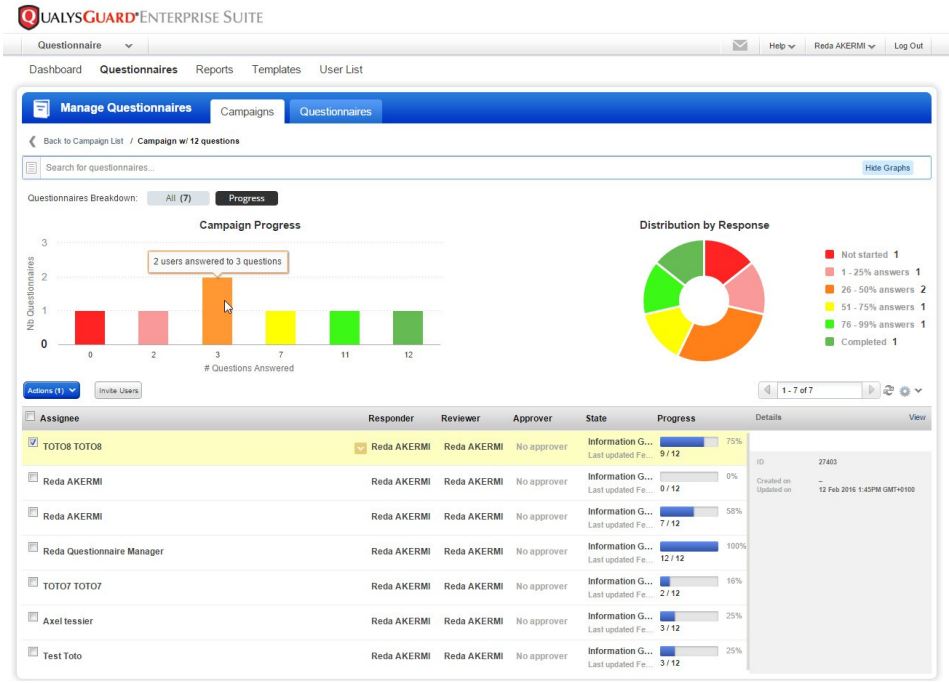
Métrique : complexité

- Exemple : PMD, PHPMD, Pdepend, etc.
- Détection de code mort, fonctions et/ou variable inutilisé.
- Notion de complexité cyclomatique.
- Longueur méthodes, classes, etc...
- Etc...

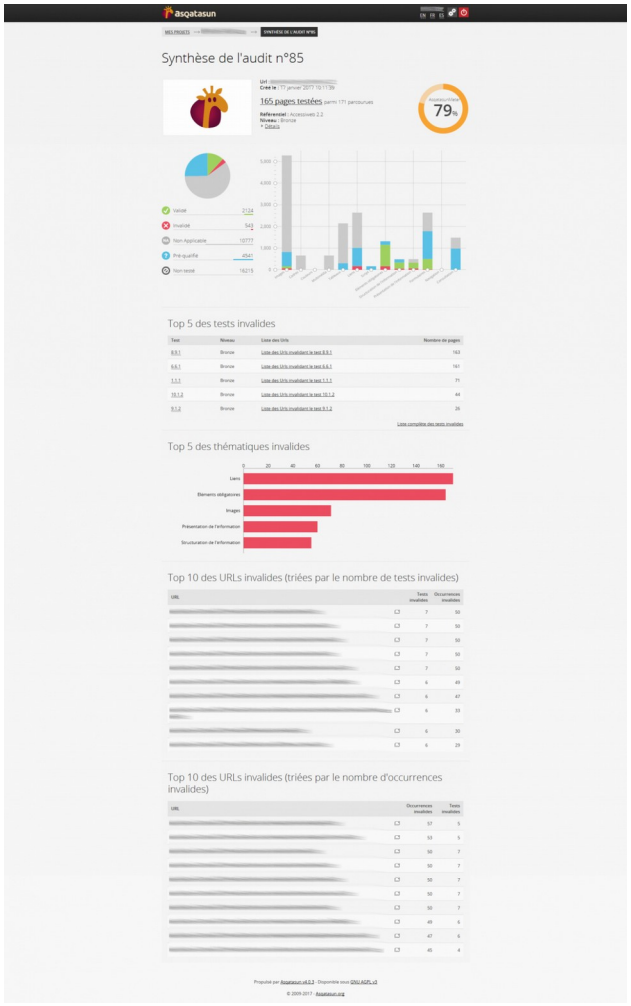
```
// Cyclomatic Complexity = 12
class Foo {
1   public function example()  {
2       if ($a == $b)  {
3           if ($a1 == $b1) {
                fiddle();
4           } else if ($a2 == $b2) {
                fiddle();
            }  else {
                fiddle();
            }
5       } else if ($c == $d) {
6           while ($c == $d) {
                fiddle();
            }
7        } else if ($e == $f) {
8           for ($n = 0; $n < $h; $n++) {
                fiddle();
            }
        } else{
            switch ($z) {
9               case 1:
                    fiddle();
                    break;
10              case 2:
                    fiddle();
                    break;
11              case 3:
                    fiddle();
                    break;
12              default:
                    fiddle();
                    break;
            }
        }
    }
}
```

Métrique : sécurité





Métrique : accessibilité



Métrique : performance (ex : Gatling)

Métrique : divers

- Ligne de code (phploc).
- Performance Web (YSlow).
- Bien sûr tous les outils de TU / TI / TNR.

Documentation

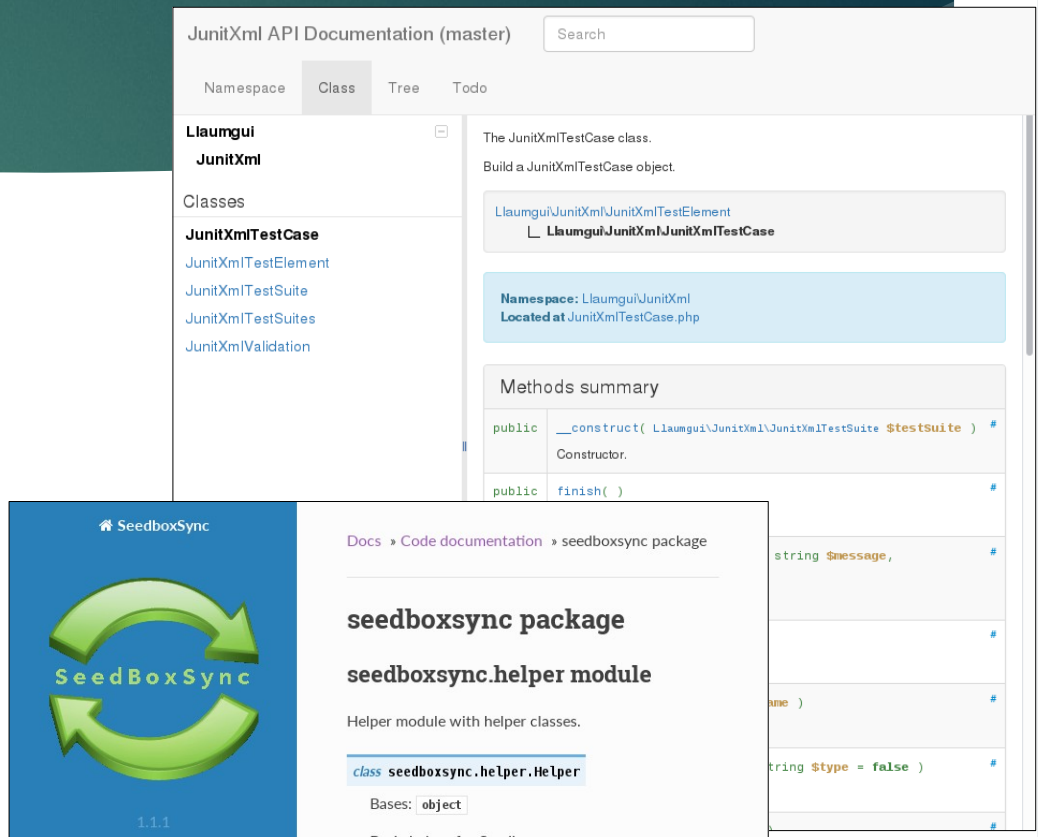
Des outils permettent de construire des mini-sites très facilement faisant office de documentation Parmi eux, citons :

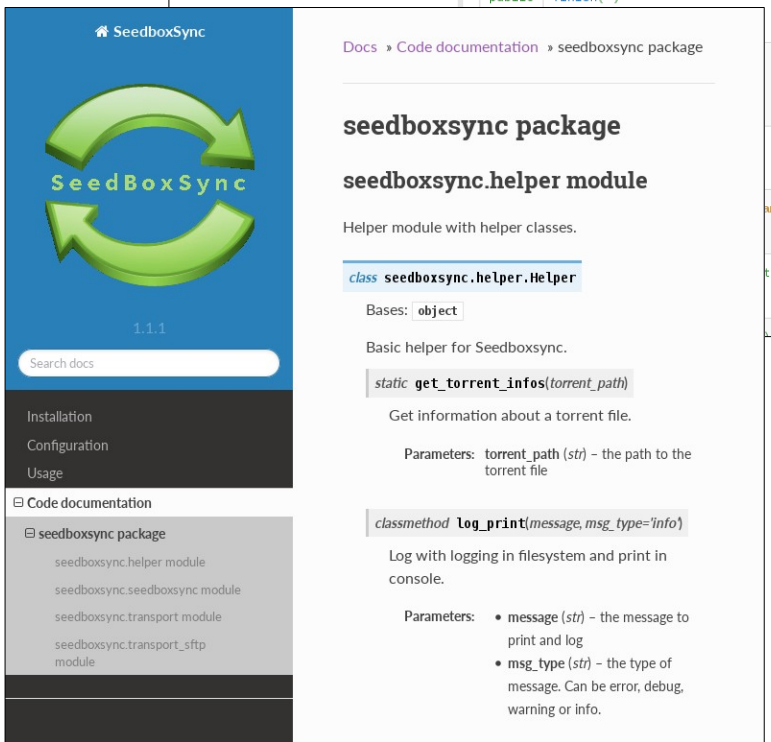
- Couscous (PHP).
- Sphinx (Python).

API documentation

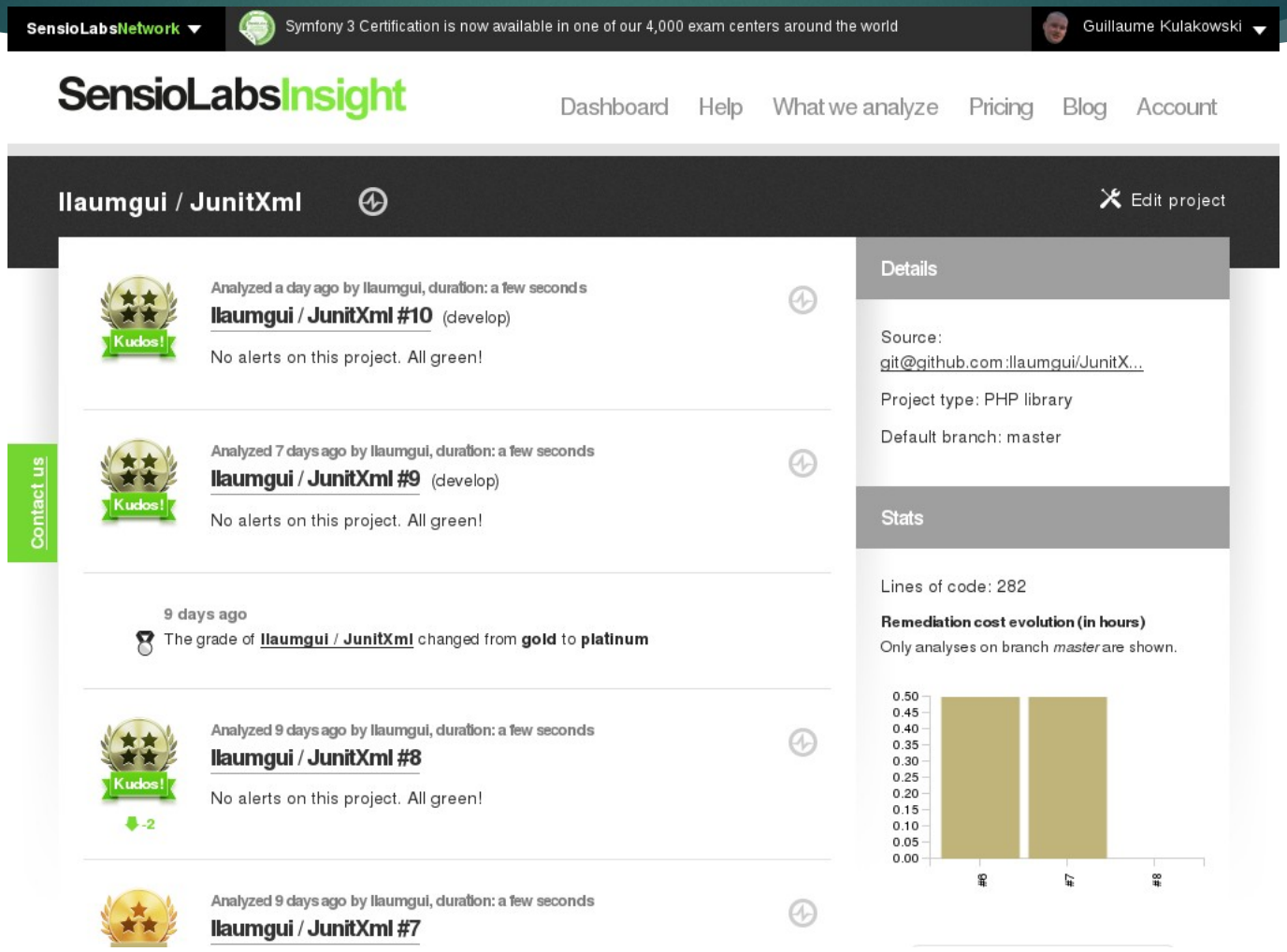
Des outils permettent de construire la documentation de l'API. Parmi eux, citons :

- apigen, phpDocumentor, Sami (PHP).
- Sphinx (Python).
- Javadoc (Java).
- Doxygen (multiple langages).

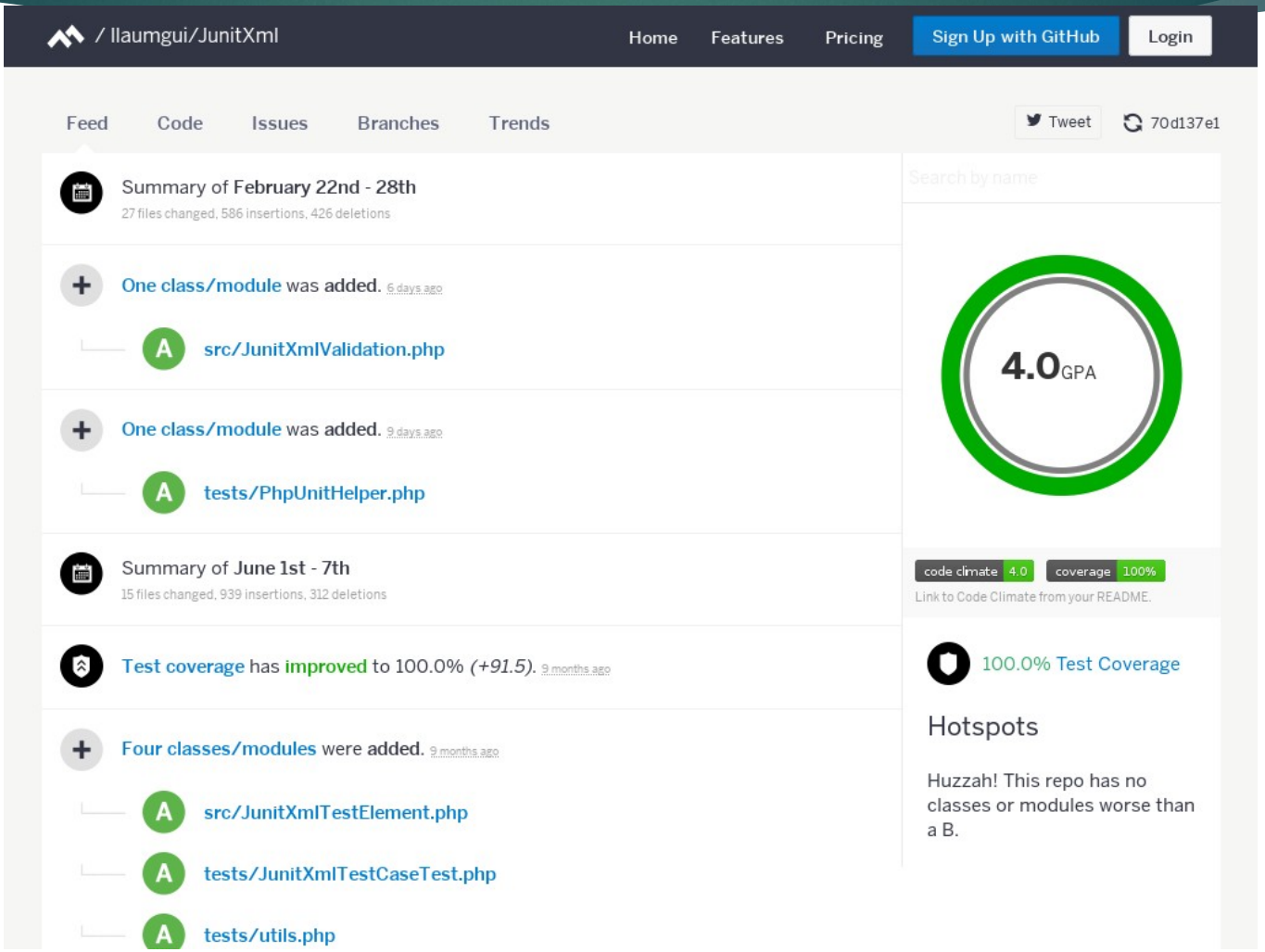




Exemple d'outil de qualimétrie #1 : SensioLabsInsight



Exemple d'outil de qualimétrie #2 : CodeClimate



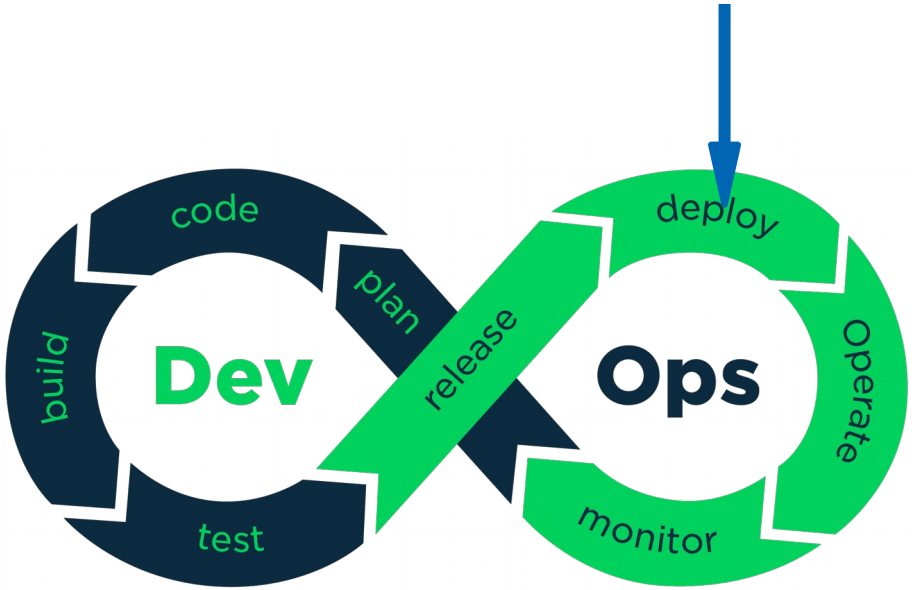
L'expert technique / Lead developer

- Son rôle est primordiale,
- Il doit suivre tous ces indicateurs,
- Il doit faire des revues de code.

5 – Deploy
Déploiement Continu

- Définition
- Exemple

Le cycle DevOps



Jenkins

maven

Deployer

Définition

Le déploiement continu, prolongement de l'intégration continue, est une pratique visant à réduire le plus possible le temps de cycle, le temps écoulé entre l'écriture d'une nouvelle ligne de code et l'utilisation réelle de ce même code par des utilisateurs finaux.

L'équipe s'appuie sur une infrastructure qui automatise l'ensemble des étapes de déploiement (ou "mise en production"), de sorte qu'après chaque intégration qui se solde par des tests passant avec succès, l'application en production est mise à jour *(© institut-agile)*.

Discours d'un autre temps...

L'outillage du DevOps

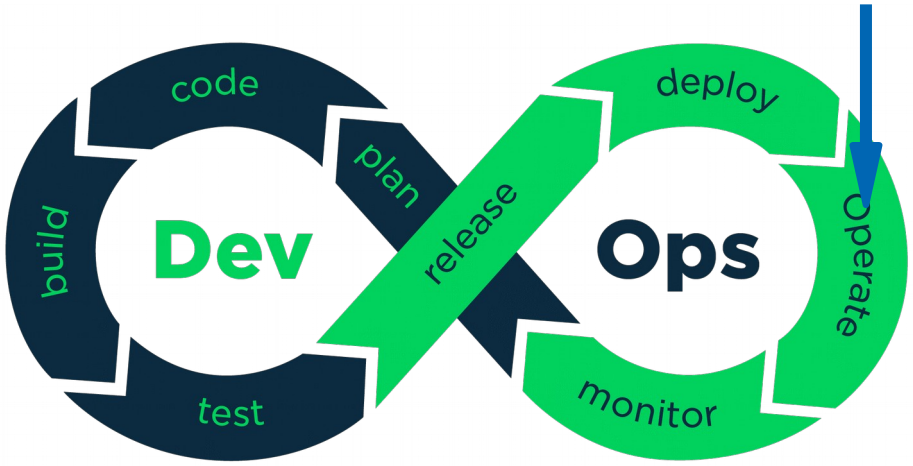
- Sans passer par des concepts de conteneurs (ex : Docker) qui permettent de pousser une infrastructure complète, des outils comme Capistrano, Mina ou Deployer permettent de pousser du code en production.
- Pousse sur **différents serveurs en même temps** (plus de facteur humain).
- Effectue des taches post-update (ex : vidage de cache *warmup*, *updater*, etc.).
- Autorise le *rollback*.
- Autorise le *staging*.
- Peut-être connecté à une PIC (la boucle est bouclée).

L'outillage du DevOps

6 – Operate

- Cloud

Le cycle DevOps



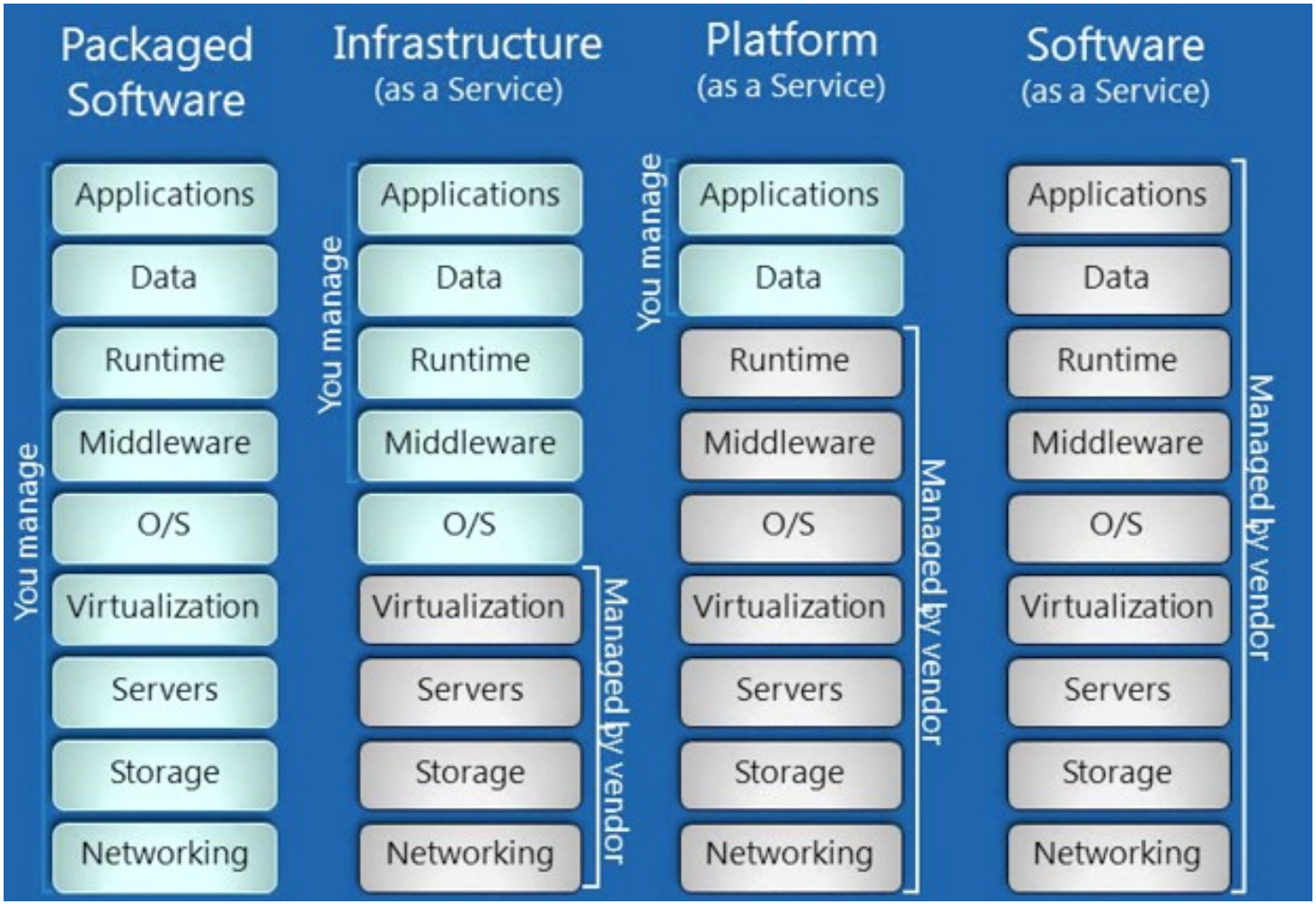
vmware

amazon web services™

heroku

Microsoft Azure

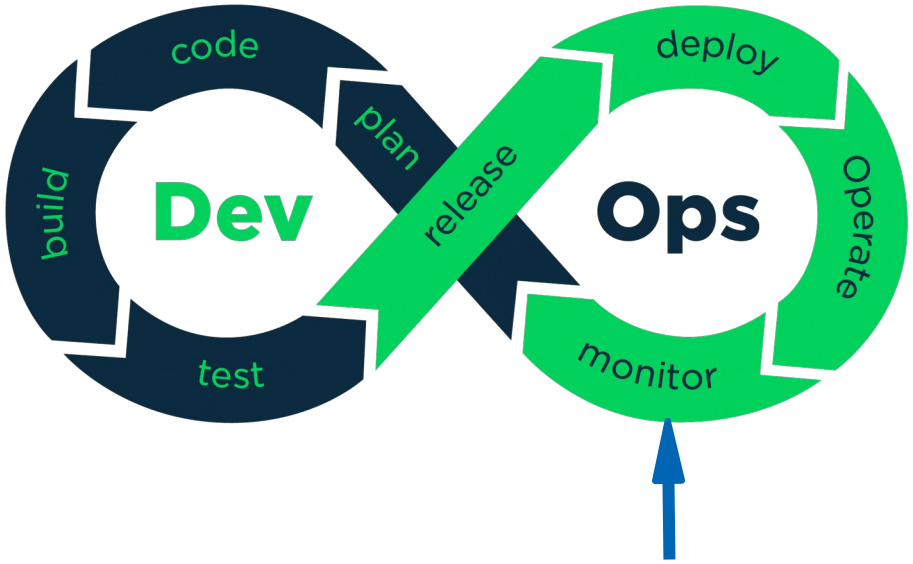
Cloud


Packaged Software
Infrastructure (as a Service)
Platform (as a Service)
Software (as a Service)
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
You manage
Managed by vendor

7 – Monitor

- Définition
- Exemple

Le cycle DevOps



graylog

dynatrace

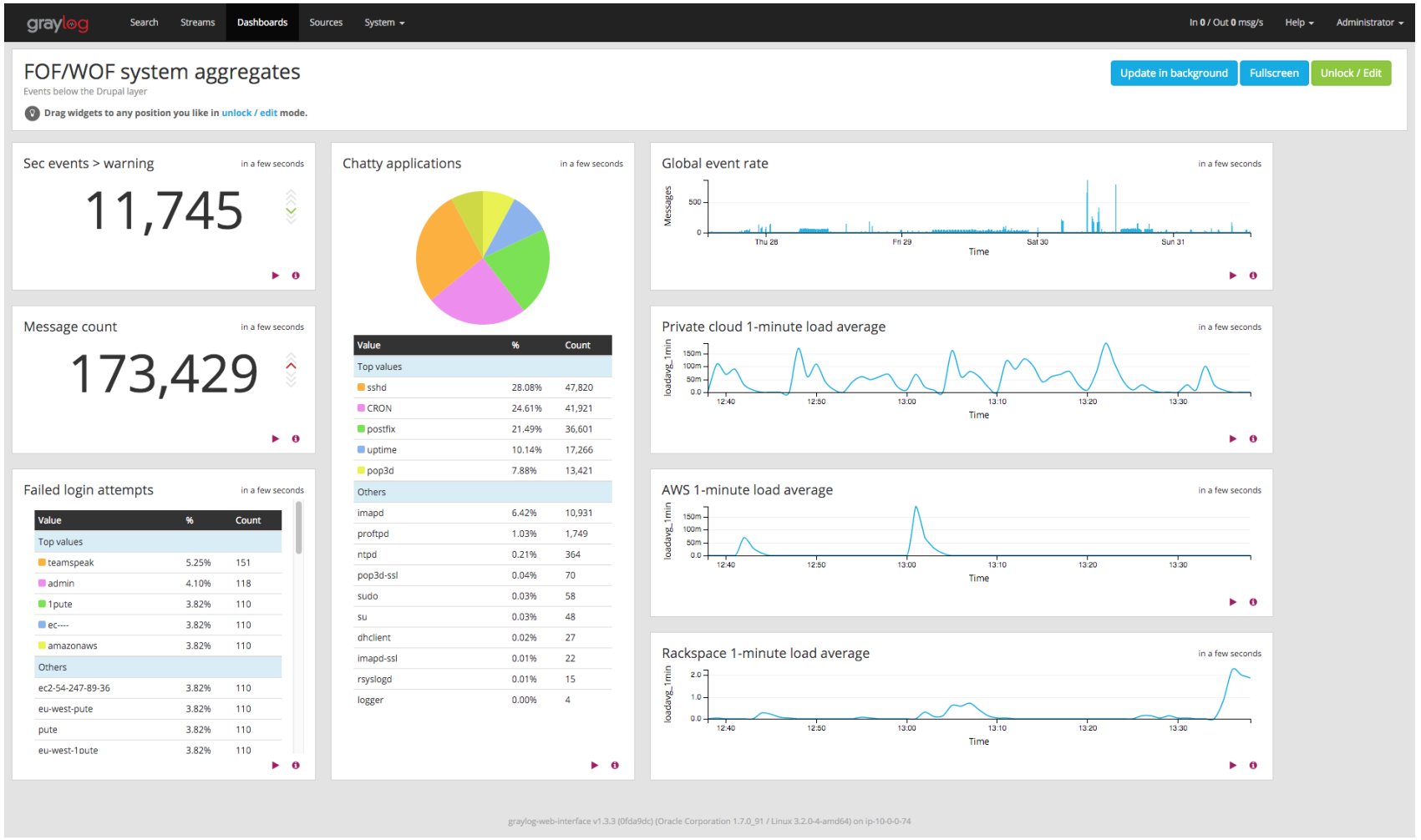
New Relic.

graphite projects

blackfire

elastic

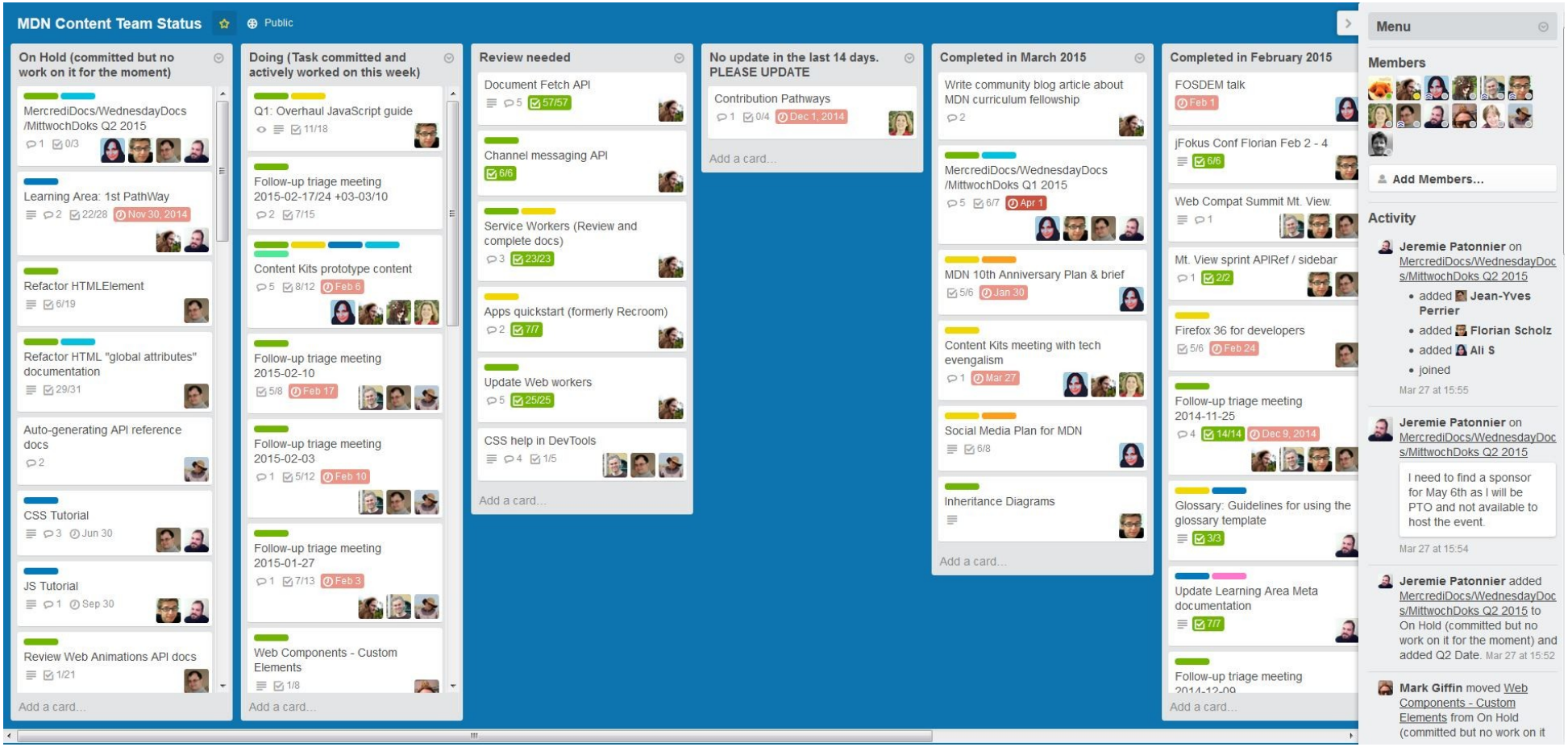
Exemple de dashboard GrayLog



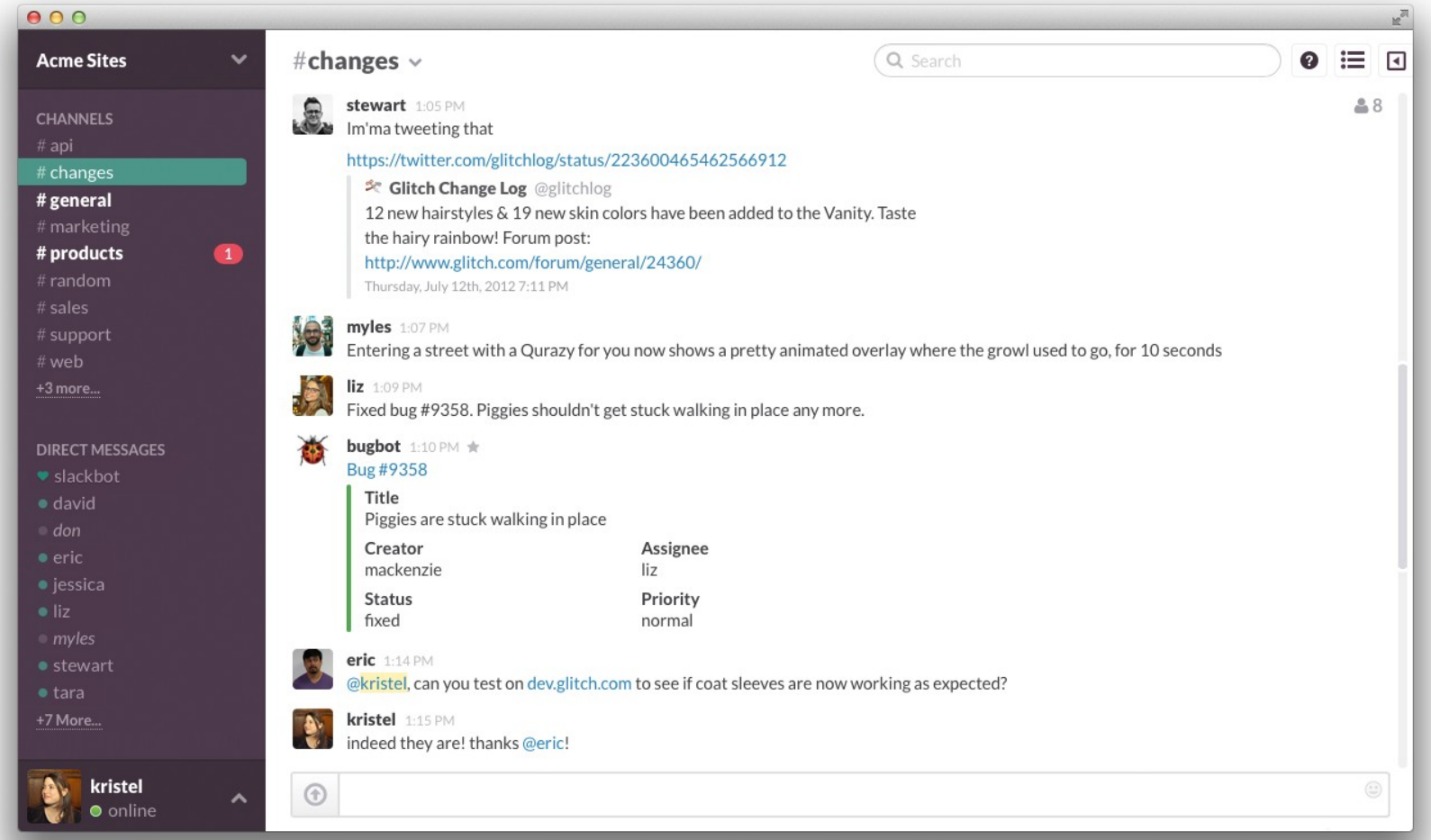
8 – Outils divers

- Kanban
- Communication

Trello: Kanban agile



Slack : communication entre équipes



Merci à vous !