Intégration continue avec GitHub

Table of Contents

1. Objectifs

1.1. Une vérification systématique de l'état du code

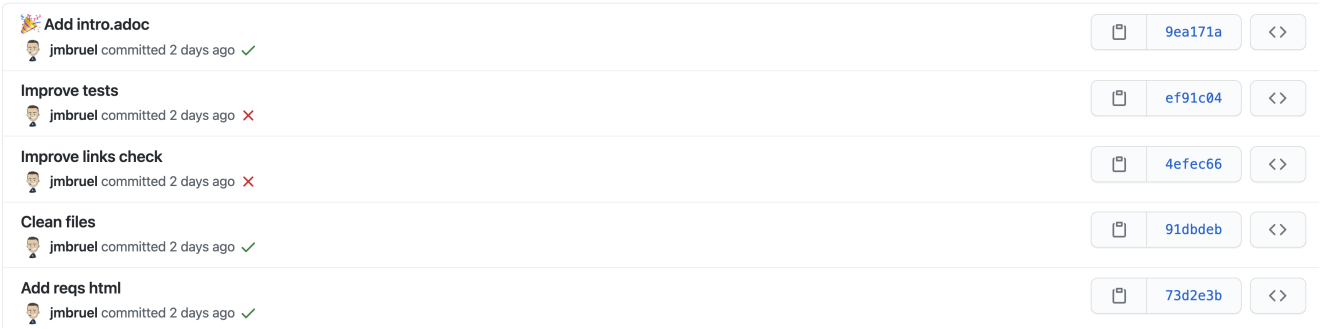

Figure 1. Résultats visibles et tracés

1.2. Environnement

On utilise ici l'intégration continue fournie avec GitHub : Actions, mais on peut coupler d'autres outils avec son compte GitHub :

- circleci -

Liste exhaustive ici : https://github.com/ligurio/awesome-ci

1.3. eXtreme Programming

> Continuous Integration is a software development practice where members of a team integrate their work frequently, [...] leading to multiple integrations per day.
>
> — Martin Fowler

1.4. Principe général

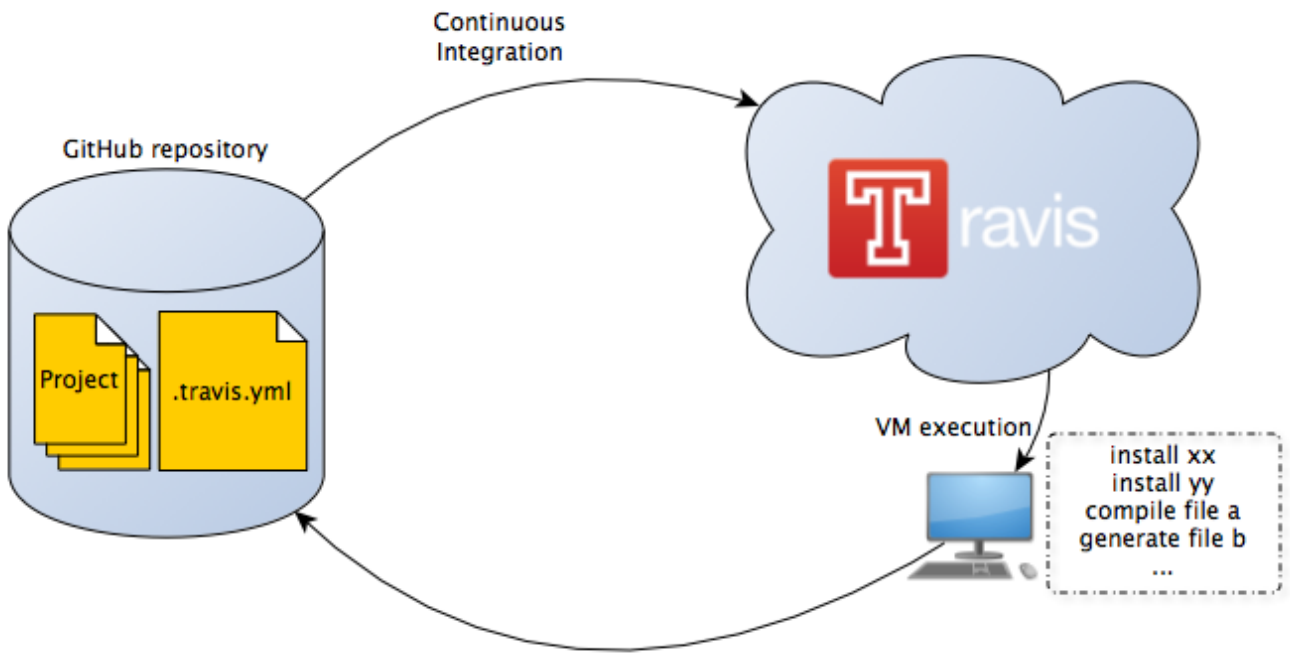

Figure 2. Exemple d'Intégration Continue (github-travis)

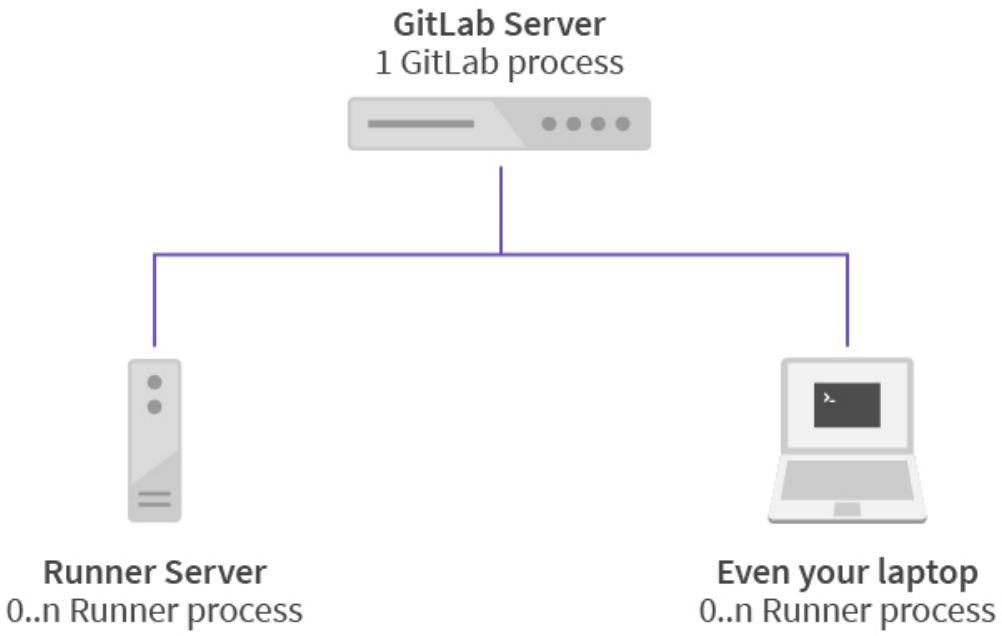

Figure 3. Architecture GitLab-CI (https://about.gitlab.com/gitlab-ci/)

1.5. YAML

YAML: **Y**AML **A**in't **M**arkup **L**anguage

Exemple de fihier `.yml`

```
---
receipt:     Oz-Ware Purchase Invoice
date:        2012-08-06
customer:
    first_name:   Dorothy
    family_name:  Gale
```

Utiliser des espaces et non des tabulations.

2. Utilisation

Un serveur d'intégration continue peut permettre de :

- Faire les opérations type git (pull, checkout, push)
- Compiler du code source ⇒ **build**
- Créer des archives
- Déployer du code sur une machine de test
- Exécuter une suite de **tests** (Junit, Audit de code source, test IHM, ...)
- Notifier des résultats (mail, RSS)
- etc.

3. Services connus

Les serveurs d'intégration les plus connus (Wikipedia en compte plus de 40!) :

- Cruise Control
- Travis CI
- Jenkins (anciennement Hudson)
- Gitlab CI

4. Pour notre environnement (GitLab)

4.1. Processus type

1. Créer à la racine du projet un fichier `.gitlab-ci.yml`
2. Y décrire ce que l'on souhaite réaliser
3. "Pousser" dans le dépôt GitLab ses modifications
4. Contrôler les résultats

4.2. Exemple (MPA2016-1B2)

```
image: node:4.2.2   ①

all_tests:
  script:           ②
   - npm install express --save
   - node ./myapp.js
```

① Nom du "runner"

② Instructions à réaliser sur la machine

Résultat avec le fichier `.gitlab-ci.yml` précédent :

```
Running with gitlab-ci-multi-runner 1.5.2 (76fdacd)
WARNING: image is not supported by selected executor and shell
Using Shell executor...
Running on algec...
Cloning repository...
Cloning into '/home/gitlab-runner/builds/8e5cecac/0/hugues/MPA2016-G1B2'...
Checking out 0bcaba58 as testingCI...
su: User not known to the underlying authentication module
$ npm install express --save
express@4.14.0 node_modules/express
├── escape-html@1.0.3
├── array-flatten@1.1.1
├── utils-merge@1.0.0
├── cookie-signature@1.0.6
├── merge-descriptors@1.0.1
├── fresh@0.3.0
├── methods@1.1.2
├── vary@1.1.0
├── path-to-regexp@0.1.7
├── encodeurl@1.0.1
├── range-parser@1.2.0
├── parseurl@1.3.1
├── content-type@1.0.2
├── etag@1.7.0
├── cookie@0.3.1
├── content-disposition@0.5.1
├── serve-static@1.11.1
├── depd@1.1.0
├── qs@6.2.0
├── on-finished@2.3.0 (ee-first@1.1.1)
├── finalhandler@0.5.0 (unpipe@1.0.0, statuses@1.3.0)
├── debug@2.2.0 (ms@0.7.1)
├── proxy-addr@1.1.2 (forwarded@0.1.0, ipaddr.js@1.1.1)
├── send@0.14.1 (destroy@1.0.4, ms@0.7.1, statuses@1.3.0, mime@1.3.4, http-errors@1.5.0)
├── accepts@1.3.3 (negotiator@0.6.1, mime-types@2.1.12)
└── type-is@1.6.13 (media-typer@0.3.0, mime-types@2.1.12)
$ node ./myapp.js
App listening on port 3000! Type in a browser localhost:3000/
```

Figure 4. Attention au code infini

Avec un fichier `.gitlab-ci.yml` plus conforme (exécution de tests) :

```
    - npm install express --save
    - node ./specs/start.js ./specs/async.spec.js
```

```
$ node ./specs/start.js ./specs/async.spec.js
Spec started

  testing parallel
    ✓ should return a Promise object (0.005 sec)
    ✓ should return correct output (0.006 sec)
    ✓ should call the function with each element in the array (0.003 sec)

  testing waterfall
    ✓ should return a Promise object (0.001 sec)
    ✓ should throw error if all items in function array are not functions (0.001 sec)
    ✓ should  throw error no arguments given (0 sec)

Executed 6 of 6 specs SUCCESS in 0.029 sec.

Creating cache test_async/master/cache.zip...
node_modules/: found 45154 matching files
Uploading cache.zip

Build succeeded
```

Figure 5. Le build est un succès

test_async: stage: test script: - npm install - node ./specs/start.js ./specs/async.spec.js

4.3. Exemple HelloWorld

1. Code java
2. Petit `main` de test (pas unitaire)
3. Compilation manuelle
4. Build ant
5. Améliorations
6. Tests

7. Eclipse
8. Intégration continue

4.3.1. Code java

Consultez le tutoriel ant : https://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html

src/HelloWorld.java

```
package org.jmb;
public class HelloWorld
{
   private String name = "";
   public String getName()
   {
      return name;
   }
   public String getMessage()
   {
      if (name == "")
      {
         return "Hello!";
      }
      else
      {
         return "Hello " + name + "!";
      }
   }
   public void setName(String name)
   {
      this.name = name;
   }
}
```

4.3.2. Petit `main` de test (pas unitaire)

src/Main.java

```
package org.jmb;


public class Main {
    public static void main(String[] args) {
          org.jmb.HelloWorld h = new org.jmb.HelloWorld();
          h.setName("JMB");
          System.out.println(h.getMessage());
    }
}
```

4.3.3. Compilation manuelle

```
$ javac -sourcepath src -d bin/ src/Main.java
$ ls bin
HelloWorld.class    Main.class
$ java -cp bin Main
Hello JMB!
```

4.3.4. Build ant

build.xml

```
<project>
    <target name="clean">
        <delete dir="bin"/>
    </target>

    <target name="build">
        <mkdir dir="bin"/>
        <javac srcdir="src" destdir="bin"/>
    </target>

    <target name="jar">
        <mkdir dir="bin/jar"/>
        <jar destfile="bin/jar/HelloWorld.jar" basedir="bin">
            <manifest>
                <attribute name="Main-Class" value="Main"/>
            </manifest>
        </jar>
    </target>

    <target name="run">
        <java jar="bin/jar/HelloWorld.jar" fork="true"/>
    </target>

</project>
```

```
$ ant clean build jar
$ ant run
Buildfile: /Users/bruel/HelloWorld/build.xml

run:
     [java] Hello JMB!

BUILD SUCCESSFUL
Total time: 0 seconds
```

4.3.5. Améliorations

build.xml (amélioré)

```
<project name="HelloWorld" basedir="." default="main">

    <property name="src.dir"     value="src"/>
    <property name="build.dir"   value="bin"/>
    <property name="classes.dir" value="${build.dir}/classes"/>
    <property name="jar.dir"     value="${build.dir}/jar"/>

    <property name="main-class"  value="Main"/>



    <target name="clean">
        <delete dir="${build.dir}"/>
    </target>

    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="${jar.dir}"/>
        <jar destfile="${jar.dir}/${ant.project.name}.jar" basedir="${classes.dir}">
            <manifest>
                <attribute name="Main-Class" value="${main-class}"/>
            </manifest>
        </jar>
    </target>

    <target name="run" depends="jar">
        <java jar="${jar.dir}/${ant.project.name}.jar" fork="true"/>
    </target>

    <target name="clean-build" depends="clean,jar"/>

    <target name="main" depends="clean,run"/>

</project>
```

```
$ ant
Buildfile: /Users/bruel/HelloWorld/build.xml

clean:
   [delete] Deleting directory /Users/bruel/HelloWorld/bin

compile:
    [mkdir] Created dir: /Users/bruel/HelloWorld/bin/classes
    [javac] Compiling 2 source files to /Users/bruel/HelloWorld/bin/classes

jar:
    [mkdir] Created dir: /Users/bruel/HelloWorld/bin/jar
      [jar] Building jar: /Users/bruel/HelloWorld/bin/jar/HelloWorld.jar

run:
     [java] Hello JMB!

main:

BUILD SUCCESSFUL
Total time: 1 second
```

4.3.6. Tests

src/TestHelloWorld.java (failing)

```
package org.jmb;
public class TestHelloWorld extends junit.framework.TestCase {

    public void testNothing() {
    }

    public void testWillAlwaysFail() {
        fail("An error message");
    }

}
```

build.xml (librairies extérieures)

```
<project name="HelloWorld" basedir="." default="main">

    <property name="src.dir"     value="src"/>
    <property name="build.dir"   value="bin"/>
    <property name="classes.dir" value="${build.dir}/classes"/>
    <property name="jar.dir"     value="${build.dir}/jar"/>

    <property name="main-class"  value="Main"/>
```

```
    <property name="lib.dir"     value="lib"/>

    <path id="classpath">
        <fileset dir="${lib.dir}" includes="**/*.jar"/>
    </path>

    <target name="clean">
        <delete dir="${build.dir}"/>
    </target>

    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"
classpathref="classpath"/>
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="${jar.dir}"/>
        <jar destfile="${jar.dir}/${ant.project.name}.jar" basedir="${classes.dir}">
            <manifest>
                <attribute name="Main-Class" value="${main-class}"/>
            </manifest>
        </jar>
    </target>

    <path id="application" location="${jar.dir}/${ant.project.name}.jar"/>

    <target name="run" depends="jar">
        <java fork="true" classname="${main-class}">
            <classpath>
                <path refid="classpath"/>
                <path refid="application"/>
                <path location="${jar.dir}/${ant.project.name}.jar"/>
            </classpath>
        </java>
    </target>

    <target name="junit" depends="jar">
        <junit printsummary="yes">
            <classpath>
                <path refid="classpath"/>
                <path refid="application"/>
            </classpath>

            <batchtest fork="yes">
                <fileset dir="${src.dir}" includes="TestHelloWorld.java"/>
            </batchtest>
        </junit>
    </target>

    <target name="clean-build" depends="clean,jar"/>
```

```
    <target name="main" depends="clean,run"/>

</project>
```

```
$ ant junit
Buildfile: /Users/bruel/HelloWorld/build.xml

compile:

jar:

junit:
    [junit] Running TestHelloWorld
    [junit] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0,003 sec
    [junit] Test TestHelloWorld FAILED

BUILD SUCCESSFUL
Total time: 1 second
```

src/TestHelloWorld.java (un vrai ce coup-ci)

```
package org.jmb;
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;


public class TestHelloWorldReal {

   private org.jmb.HelloWorld h;

   @Before
   public void setUp() throws Exception
   {
      h = new org.jmb.HelloWorld();
   }

   @Test
   public void testHelloEmpty()
   {
      assertEquals(h.getName(),"");
      assertEquals(h.getMessage(),"Hello!");
   }

   @Test
   public void testHelloWorld()
   {
      h.setName("World");
      assertEquals(h.getName(),"World");
      assertEquals(h.getMessage(),"Hello World!");
   }
}
```

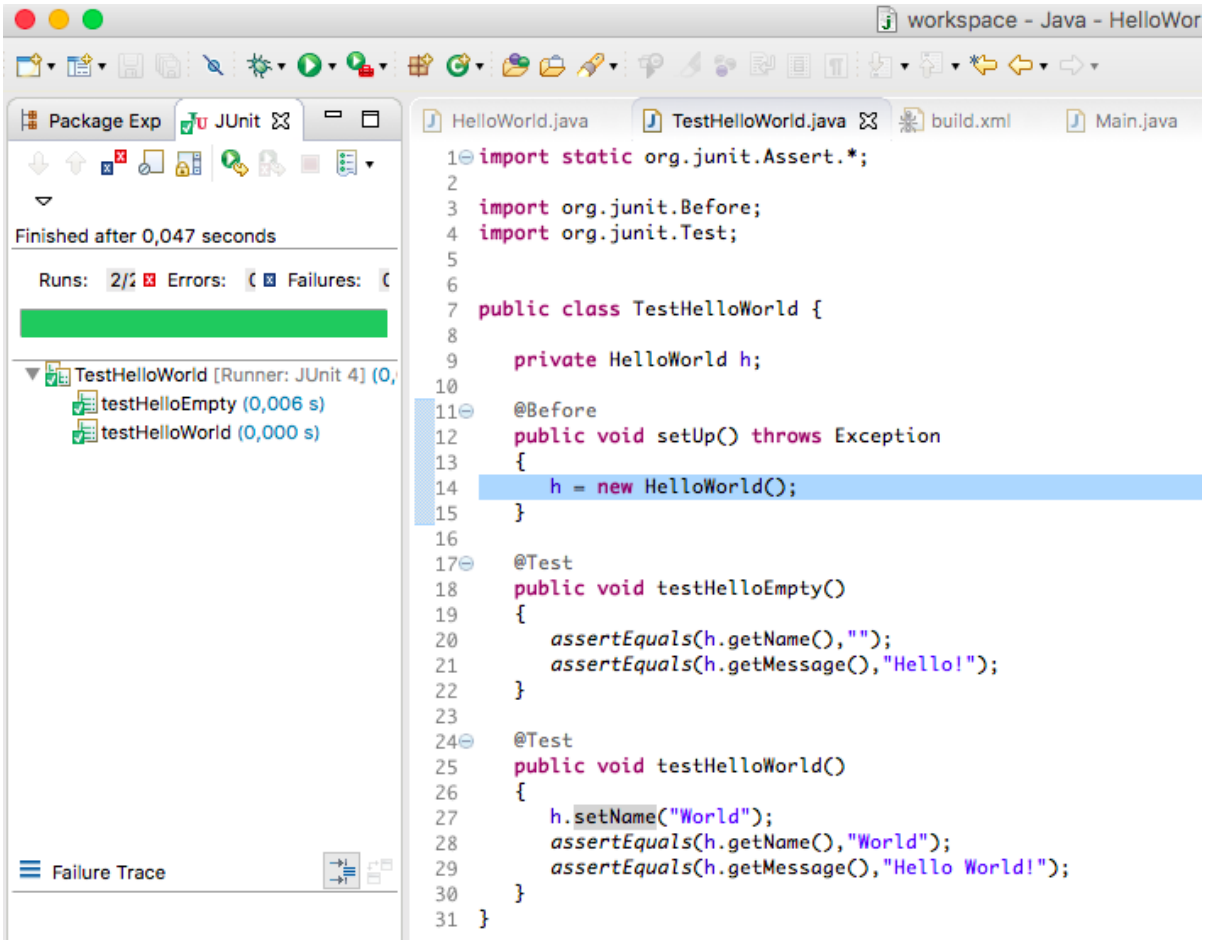

Figure 6. Un test exécuté sous eclipse

Ne pas hésiter à utiliser le plugin infinitest.

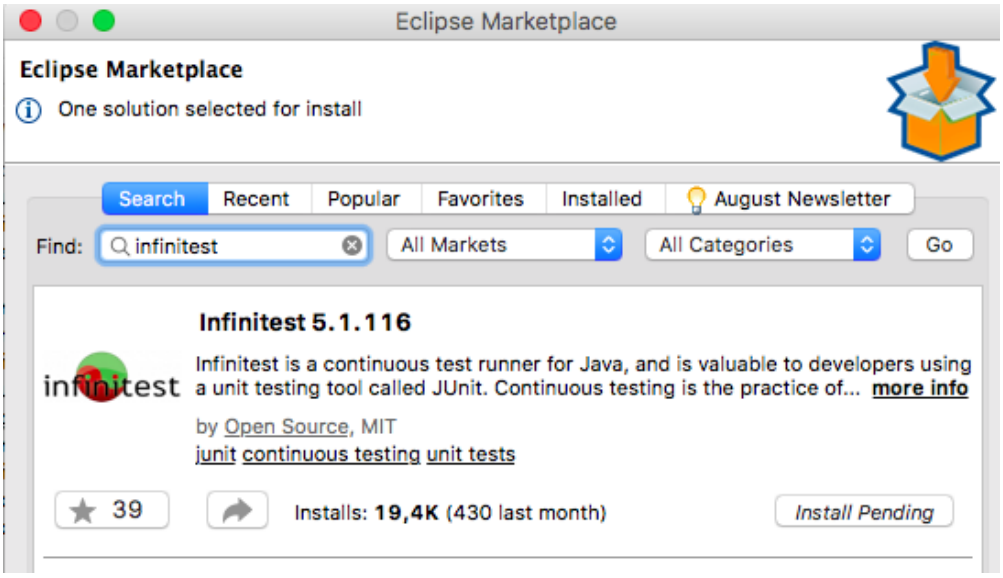

Figure 7. Utilisation d'infinitest sous eclipse

4.3.7. Eclipse

Profiter de la génération de fichier de build ant par Eclipse!

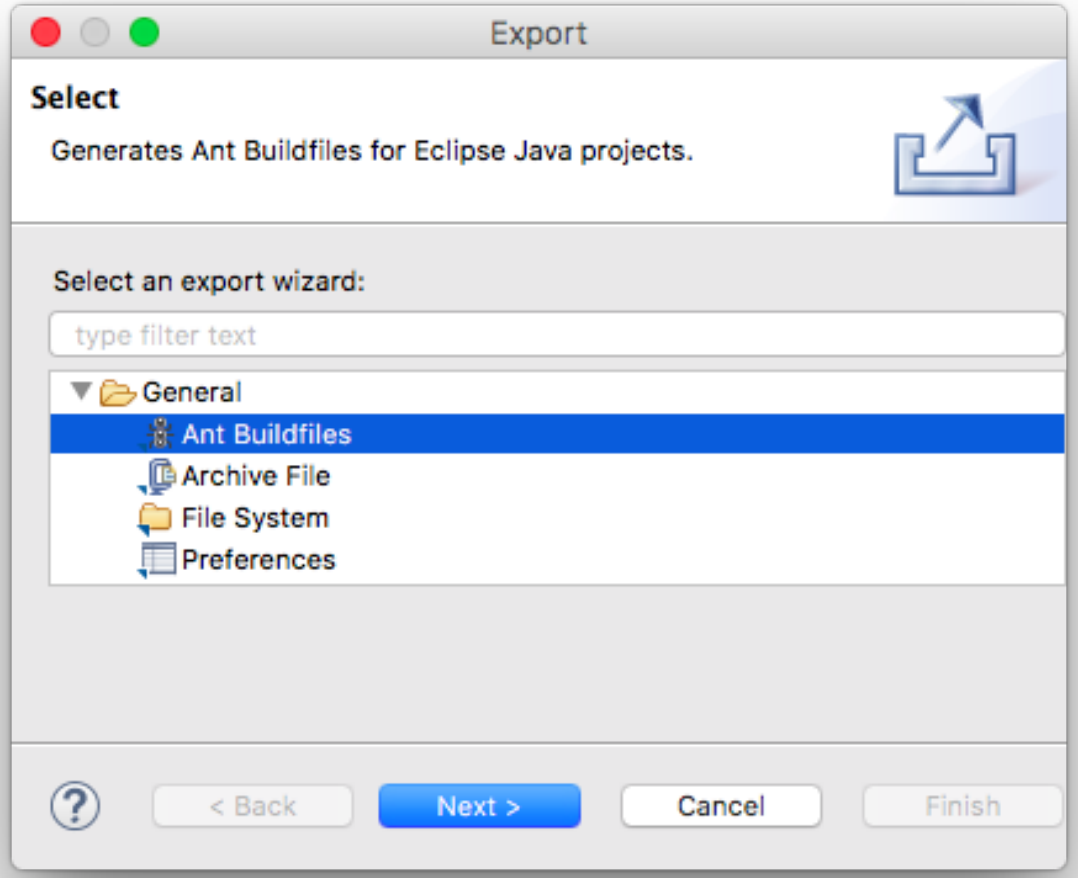

Figure 8. Export du bild ant sous eclipse

build.xml (généré par eclipse)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- WARNING: Eclipse auto-generated file.
              Any modifications will be overwritten.
              To include a user specific buildfile here, simply create one in the same
              directory with the processing instruction <?eclipse.ant.import?>
              as the first entry and export the buildfile again. --><project
basedir="." default="build" name="HelloWorld">
    <property environment="env"/>
    <property name="junit.output.dir" value="junit"/>
    <property name="debuglevel" value="source,lines,vars"/>
    <property name="target" value="1.8"/>
    <property name="source" value="1.8"/>
    <path id="JUnit 4.libraryclasspath">
        <pathelement
location="../../../.p2/pool/plugins/org.junit_4.12.0.v201504281640/junit.jar"/>
        <pathelement
location="../../../.p2/pool/plugins/org.hamcrest.core_1.3.0.v201303031735.jar"/>
    </path>
```

```
    <path id="HelloWorld.classpath">
        <pathelement location="bin"/>
        <path refid="JUnit 4.libraryclasspath"/>
    </path>
    <target name="init">
        <mkdir dir="bin"/>
        <copy includeemptydirs="false" todir="bin">
            <fileset dir="src">
                <exclude name="**/*.launch"/>
                <exclude name="**/*.java"/>
            </fileset>
        </copy>
    </target>
    <target name="clean">
        <delete dir="bin"/>
    </target>
    <target depends="clean" name="cleanall"/>
    <target depends="build-subprojects,build-project" name="build"/>
    <target name="build-subprojects"/>
    <target depends="init" name="build-project">
        <echo message="${ant.project.name}: ${ant.file}"/>
        <javac debug="true" debuglevel="${debuglevel}" destdir="bin"
includeantruntime="false" source="${source}" target="${target}">
            <src path="src"/>
            <classpath refid="HelloWorld.classpath"/>
        </javac>
    </target>
    <target description="Build all projects which reference this project. Useful to
propagate changes." name="build-refprojects"/>
    <target name="Main">
        <java classname="org.jmb.Main" failonerror="true" fork="yes">
            <classpath refid="HelloWorld.classpath"/>
        </java>
    </target>
    <target name="TestHelloWorld">
        <mkdir dir="${junit.output.dir}"/>
        <junit fork="yes" printsummary="withOutAndErr">
            <formatter type="xml"/>
            <test name="org.jmb.TestHelloWorld" todir="${junit.output.dir}"/>
            <jvmarg line="-ea"/>
            <classpath refid="HelloWorld.classpath"/>
        </junit>
    </target>
    <target name="junitreport">
        <junitreport todir="${junit.output.dir}">
            <fileset dir="${junit.output.dir}">
                <include name="TEST-*.xml"/>
            </fileset>
            <report format="frames" todir="${junit.output.dir}"/>
        </junitreport>
    </target>
```

```
</project>
```

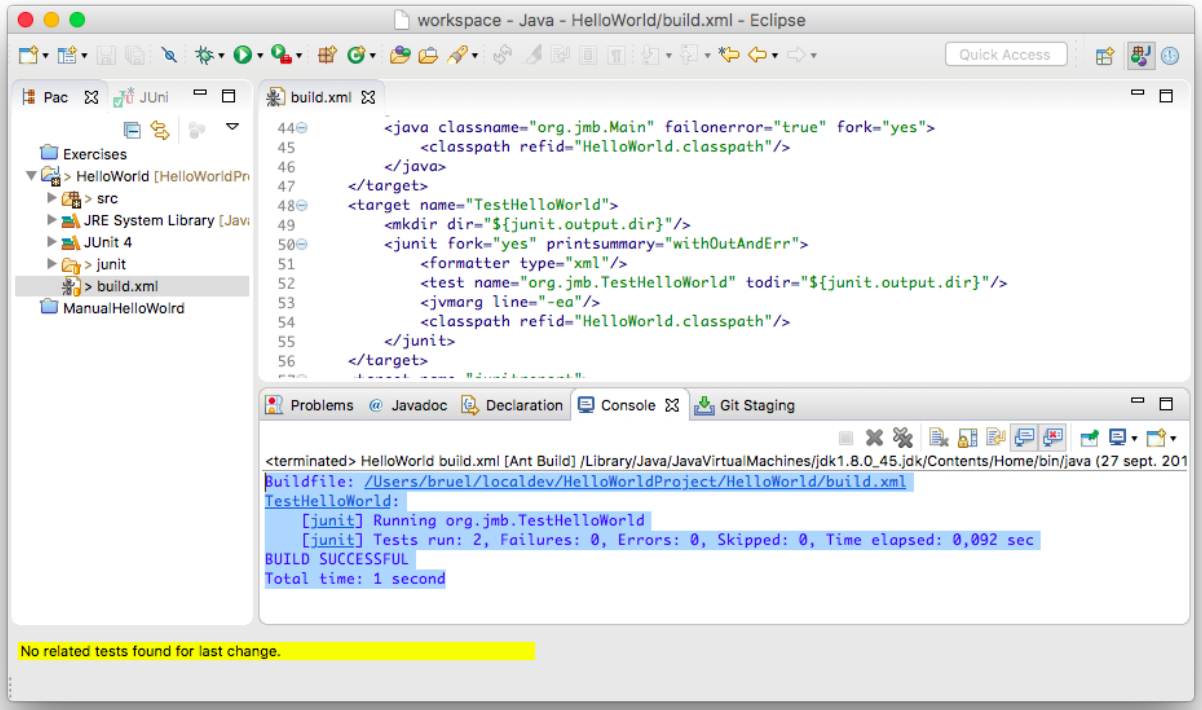

Figure 9. Run de ant sous eclipse

Autre avantage de junit, la génération de pages web (répertoire `junit`, ouvrir `index.html`).

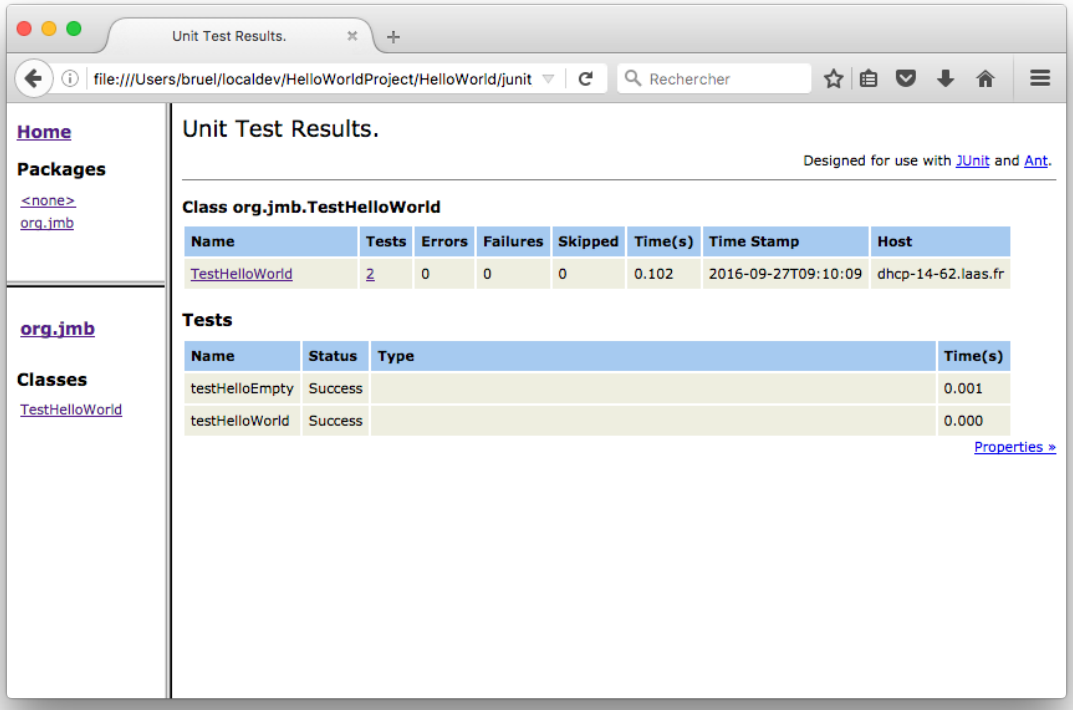

Figure 10. Présentation des résultats de test

4.3.8. Integration Continue

.gitlab-ci.yml (ajouté à la racine du projet git)

```
image: asciidoctor/docker-asciidoctor

variables:
  GIT_SSL_NO_VERIFY: "1"

stages:
  - build
  - test

html:
  stage: build
  script:
    - asciidoctor README.adoc -o index.html
  artifacts:
    paths:
    - index.html

pdf_preview:
  stage: test
  when: manual
  environment:
    name: preview/$CI_COMMIT_REF_NAME
  except:
    - /master/
  artifacts:
    paths:
    - README.pdf
    expire_in: 1 week
  script:
    - asciidoctor-pdf README.adoc
```

Il faut modifier éventuellement le `build.xml`, souvent trop lié à Eclipse.

5. Divers

5.1. Quand on ne veux pas lancer l'IC

```
git commit -m "Blabla... [ci skip]"
```

5.2. Pour vérifier la syntaxe de son fichier YAML

Possibilité de tester son fichier `gitlab-ci.yml` :

https://docs.gitlab.com/ee/ci/lint.html

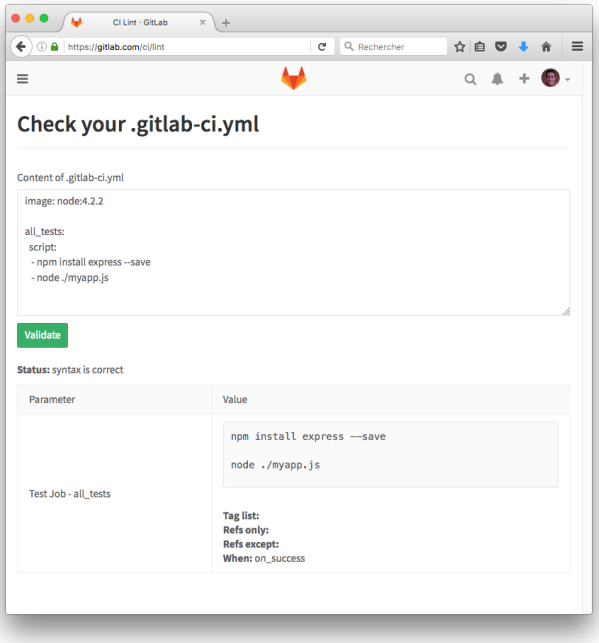

Figure 11. Validation de son code YAML

6. Liens utiles

Le site de référence

https://about.gitlab.com/gitlab-ci/

Un tuto en français sur l'IC sous GitLab (merci @npm_kader)

https://www.grafikart.fr/tutoriels/divers/gitlab-ci-docker-808