Meteor project

Tests

June 2017

Sommaire

[1. Introduction 4](#_Toc486491867)

[2. Context 4](#_Toc486491868)

[3. What to test 4](#_Toc486491869)

[4. Tools 4](#_Toc486491870)

[5. Implementation 4](#_Toc486491871)

[5.1. Items 4](#_Toc486491872)

[5.2. Structure Svn 4](#_Toc486491873)

[5.3. Tests 4](#_Toc486491874)

[5.3.1. En local 4](#_Toc486491875)

[5.3.2. Sur le server d’integration 4](#_Toc486491876)

[5.3.3. Sur la plateform validation 4](#_Toc486491877)

Information

BonitaSoft

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Email** |
|  | Lionel Palacin |  |
|  | Pierre-Yves Monnet |  |
|  | Pierrick Voulet |  |

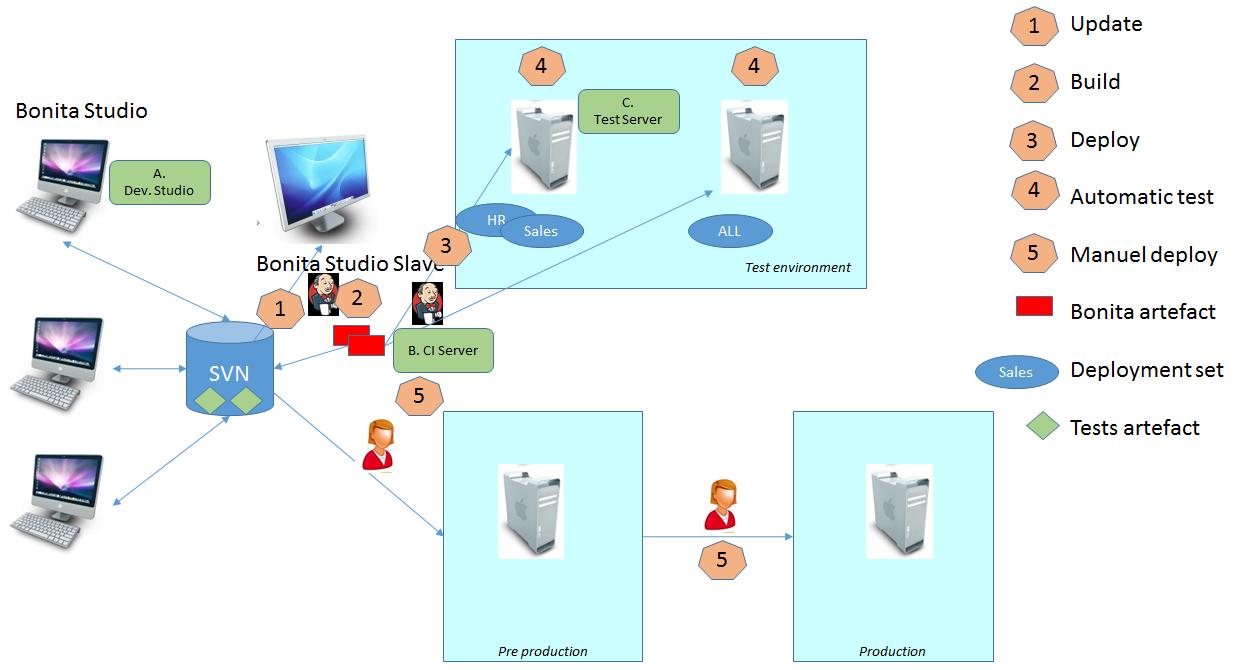
Content

# Introduction

This document groups the big picture to automate the test. It gives the direction we want to use

# Context

There is multiple context to run tests, according the CI description



***A. Dev studio***:

the developer, on its BonitaStudio would start some local tests

***B. CI Server:***

Integration server would run tests. For example, some CI scheduler (like Bamboo) can detect when a new artefact is committed in SVN and then can pilot the Bonita Studio Slave to build the artefact immediately and run any test related to this artefact. Jenkins is more based on a schedule timer. In that situation, the CI can start the test, but a Bonita Server is needed. The Bonita Server is then a server in the test environment. That implied to run the test on the CI to know on which test server to start the test

***C. test server:***

A variant of the previous one, except that the test is started directly on the server itself, inside the Bonita Server. Then, the test does not need to have a server in configuration (it’s itself).

There are advantage and concern:

* for the performance point of view, the server run the test, so it may have implied the performance
* some customer will not setup a CI environment, so it’s important to give a solution out of the CI
* For analysis, it’s important to have a user interface and start manually the test (before automate it for example)

# What to test

There is different artefact to test, and there are two different tests:

* Unit Test: a test with a input test game, and an expected result in term of data and performance. Note the performance is more mandatory on the Test Server
* Load Test: a test loading the server (create 10000 cases in a process). The result is needed, but performances are important too

## Process

To test a process, the best way is to create cases, with different set of data to parse all the different flow. A cover test is then nice (did I cover all the different path on my process? )

Human task as to be consider: the test must be able to execute this type of task.

## Form

A form is display on a browser, and must be verify: is all the data is displayed correctly? Is the dynamic of the form run (example, user check a box, and then additional information appears, of user give a bad period, a warning message should appears and it must not be possible to valid the form)

## Rest Api Extension

The Rest API Extension has to be verify with mockup, and with real data too.

## Permission

Permission has to be verify for each items: is Walter Bates can access this REST API ? But Jan Fisher don’t ? Same for processes, Page, Profile and Application.

## Page

Verify the page, with the same logical : is all the REST API in a page give a result ?

# Tools

Here the list of the different tools, and where they must run

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1. **Dev Studio** | 1. **CI Server** | 1. **Test Server** |
| **Meteor / Process** | Yes | Yes | Yes |
| **Groovy Scenario** | Yes | Yes | Yes |
| **Eclipse Scenario** | Yes | Yes | Yes |
| **Rest Api Extension JUNIT** | Yes | No | No |
| **Rest Api Extension** | Yes | Yes | Yes |
| **Permission** | No | Yes | Yes |
| **Page** | Yes | Yes | Yes |

# Implementation

The main idea is to have a common interface to run each test. If the test respect this interface, then we can after build the engine to be able to run it on different environment.

This common interface should include a way to save it, to be able to save it in the SVN database.

Multiple team can work, and share the same SVN, the same CI, and the same test environment. Here a realistic scenario:

* 4 different team (Blue, Red, Orange, Green) work with Bonita
* They use the same SVN, the same CI server
* On the deployment, 2 different servers are used. Blue, Red deploy process on Server 1, tenant 1. Orange use server 1, tenant 101. Green use the server 2
* On preproduction, there are 1 server, 3 Tenants (Blue and Red use the same Tenant)

So tests must be able to deploy an test on different server different tenant

The description envelop has:

* A Unique Name for the test
* include all needed information – in case a a Eclipse Scenario, the JAVA part has to be used

The deployment set has to register the different test. For example, team Orange will create 4 tests, and in the Deployment Set, specify

* the different artefact to deploy on Server 1, tenant 101
* the tests to run

## Test Envelop

A test Envelop is a ZIP file containing

* A page.properties with the following attributes
  + Name
  + Version
  + Type [ProcessScenario, GroovyScenario, EclipseScenario, RestApiScenario]
  + Parameters : utile a l’execution du test.
* According the type, the content

## Structure Svn

<to be describe>

## Engine to run the tests

Each test should have derived from the MeteorRobot Java Class. In this class, a getAPiAccessor() is accessible

MeterRobot : peut etre changer le name ? Il faudrait trouver un nom « testenvelop » ou «testframe » ou « junitbonita » ?

Cette classe est une classe mere. Tous les tests doivent deriver de cette classe. Elle est instanciée par le moteur, et celui-ci cree la connection au serveur Bonita.

Afin de respecter les outils JUNIT, elle doit etre capable de retourner les informations necessaires.

### Localy

Idee : faire tourner dans Eclipse, meme les ProcessScenario et GroovyScenario. En fait, il faudrait que les classes Java de meteor soit disponibles dans eclipse.

Moteur : ici le moteur est un program Java fait sous Eclipse qui va se connecter au localhost :8080. Ce moteur doit aussi etre capable d’aller chercher les tests dans SVN ?

### CI Server

Un programme doit etre capable de:

* Lire le deploiement set et faire les deploiements
* Lire les tests associés au deploiement, et executer chacun des testes. Dans ce cas, le deploiment set donne le server Bonita

### Test Environement

Meteor est capable de faire cela. Il doit aussi pouvoir être piloté par REST API pour permettre a un scheduler de le lancer.

Meteor doit pouvoir également être capable de se référencer dans le scheduler Bonita pour s’exécuter toutes les nuits, et pouvoir fournir le rapport d’exécution.

Il faut qu’il puisse également exécuter les test Eclipse

# Meteor 3

* Analyse un / des caseid pour generer le scenario de test (analyse des contrats)
* Pouvoir deployer/ executer le scenario avec un job jenkins
* Remarque charge
* Suite à notre meeting de ce jour sur ta contribution Meteor, voici une propage pour avancer.
* Problèmes de la page actuelle:
* - mix de personas: technique et moins technique
* - ergonomie très loin du niveau attendu
* - spectre trop large: fonctionnel, perf, custom…
* - environnement de la page à améliorer (objectif, ce que c’est, ce que ce n’est pas…)
* Proposition pour avancer:
* - refaire une nouvelle page from scratch comme cela tu ne perturbes pas la communauté avec la page existante
* - réduire le périmètre au test fonctionnel
* - rendre la page « clic clic only », pas de code
* - ajoute l’export « as a Jenkins job »
* Solution envisagée:
* - créer l’API qui va bien pour récupérer un enregistrement (contrats d’instance et de tâche) d’un case donné correspondant à un enregistrement fait par un utilisateur sur la plateforme
* - permettre dans la page de créer un scénario de test composé de l’import de différents cases depuis:
* \* soit un case id directement
* \* soit par sélection d’un process par son nom & version puis sélection du case grâce à certaines méta-données (date de création, contrat d’instantiation…)
* - permettre depuis la page de jouer le case ? Quel serait l’intérêt?
* - permettre depuis la page d’exporter le test fonctionnel en tant que job Jenkins qui exposerait juste le paramètre de « sleep » avant d’exécuter une tâche
* Questions / doutes à creuser:
* - persistenceId qui bougent pour un BDM?
* Proposition de texte pour cette nouvelle page:
* - title: à définir si tu veux des noms « fun » mais dans ce cas, c’est cool d’expliquer le choix du nom…
* - sub-title: « Jenkins job: process test generator »
* - goal:  graphically create functional process tests and export them as Jenkins jobs
* - how to use it: download and install this page in one of your non-production Bonita Runtime

HEADQUARTERS

PARIS, FRANCE

76 boulevard de la République

92100 Boulogne-Billancourt

EMEA, ASIA & LATIN AMERICA

GRENOBLE, FRANCE

32, rue Gustave Eiffel  
38000 Grenoble

NORTH AMERICA

SAN FRANCISCO, USA

44 Tehama Street  
San Francisco, CA 94105

NEW YORK, USA

33 Nassau Avenue

Brooklyn NY 11222