Tests fonctionnels à Résif-DC

Découverte de Behave

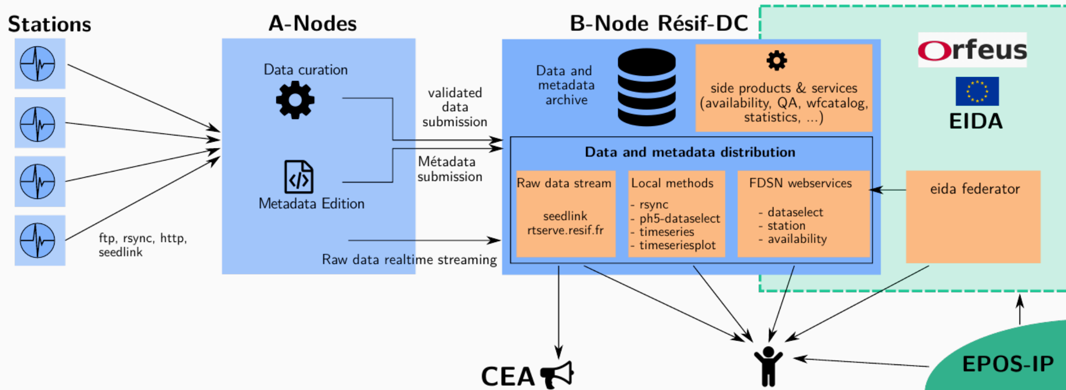
Philippe Bollard (CNRS/OSUG/Résif-DC)
philippe.bollard@univ-grenoble-alpes.fr

30/06/2021 - Séminaire rDataDev - Tests logiciels

Contexte

Résif-DC

Comment tester le bon fonctionnement de Résif-DC ?



⇒ Tester les Web Services comme le ferait un utilisateur (humain) !

Framework de test “maison” : WS-Tests

```
SERVICEURL = "http://ws.resif.fr/fdsnws/station/1/query"
tests = [{
    'id' : 'S02 - tous les réseaux RESIF contenant des canaux accéléro au format text',
    'get' : '?level=network&channel=?N?&format=text',
    'httpstatus' : 200 ,
}, {
    'id' : 'S17 - toutes les stations du réseau YR ayant des composantes N au format XML',
    'get' : '?level=station&network=YR&channel=*N&format=xml'
    'httpstatus' : 200 ,
}]

for test in tests:
    # Assemblage de la requête
    # Execution
    # Contrôle du code retour
    # Affichage : requête + OK/ERREUR
```

Analyse de l’existant

Que fait WS-Test ?

- lancer quelques requêtes (URLs) à la manière d’un humain
- vérifier le code retour ⇒ résultat OK ou pas

Évolutions souhaitées

- vérifier la conformité du contenu retourné par une requête
- syntaxe claire mais plus évoluée
- lancer un test isolémment du reste
- cas de tests plus exhaustifs

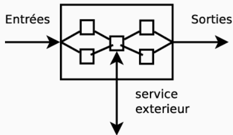
Pourquoi pas un framework de test d’API (Tavern, Apiritif) ?

- pas de réelle évolution par rapport à WS-Test
- les Web services Résif ne sont pas de vraies API REST…

Frameworks de test

Boîte blanche ou noire ?

Tests en boîte blanche



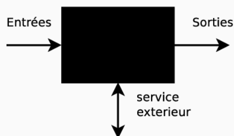
Tests unitaires

- point de vue “développeur”
- vérifier le comportement de chaque brique individuellement

WS-Tests

$\Rightarrow$ Cela ne correspond pas vraiment !

Tests en boîte noire



Tests fonctionnels

- point de vue "utilisateur"
- vérifier le comportement global de l'ensemble des briques mises en oeuvre
- valider des scénarios complexes avec plusieurs étapes
- simuler le comportement d'un utilisateur humain

WS-Tests

$\Rightarrow$ C'est ce que nous voulons faire !

Frameworks de test

Tests fonctionnels

Principaux frameworks de test fonctionnel

Selenium

- automatisation d’un navigateur Web (Firefox, Chrome)
- ne permet de tester que le comportement d’un site Web

Robot Framework

- syntaxe pseudo-Gherkin (en plus compliquée)
- plusieurs “libraries” génériques dispo (Python, Java)
- implémentation des tests = écriture d’une “library” projet

Cucumber

- description des scénarios en syntaxe Gherkin
- implémentation des “steps” en Java (ou Python/Go/Rust/Ruby/PHP/C++/...)

Cucumber pour Python

Behave

- Implémentation semi-officielle ($\Rightarrow$ recommandé)
- Facilité de partage d'infos entre les étapes grâce à une variable 'context'

Pytest-BDD

- Extension à Pytest pour les tests fonctionnels
- Difficulté de partage d'infos entre les étapes car usage des 'fixture' de PyTest (variable immutable)

Langage Gherkin

Accessible à un non-informaticien

- Syntaxe simple et claire, calquée sur les “User stories”
- Description d’un scénario en langue quasi-naturelle

Syntaxe

```
Feature <nom>
    Scenario <titre>
        Given <contexte1>
          And <contexte2>
        When <action>
        Then <résultat1>
         But <résultat2>
```

Mise en oeuvre de Behave

Premiers pas

Définion du scénario

```
Feature: showing off behave
  This is an example

  Scenario: run a simple test
     Given we have behave installed
      When we implement a test
      Then behave will test it for us!
```

Implémentation des ’steps’

```
from behave import *

@given('we have behave installed')
def step_impl(context):
    pass

@when('we implement a test')
def step_impl(context):
    assert True is not False

@then('behave will test it for us!')
def step_impl(context):
    assert context.failed is False
```

Exécution des tests

```
$ behave
Feature: showing off behave # tutorial.feature:1

  Scenario: run a simple test          # tutorial.feature:3
    Given we have behave installed     # tutorial.py:3
    When we implement a test           # tutorial.py:7
    Then behave will test it for us! # tutorial.py:11

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
3 steps passed, 0 failed, 0 skipped, 0 undefined
```

Mise en oeuvre de Behave

Aller plus loin

Composition de “steps”

Définition

```
Given some known state
  And some other known state
 When some action is taken
  And some other action is taken
 Then some outcome is observed
  But some other outcome is not observed.
```

Implémentation (etxrait)

```python
from behave import given

@given("some known state")
def step(context):
    pass

@given("some other known state")
def step(context):
    pass
```

Regrouper plusieurs scénarios

```
Feature: Multiple site support

  Background:
    Given a global administrator named "Greg"
    And a blog named "Greg's anti-tax rants"
    And a customer named "Wilson"
    And a blog named "Expensive Therapy" owned by "Wilson"

  Scenario: Wilson posts to his own blog
    Given I am logged in as Wilson
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."

  Scenario: Greg posts to a client's blog
    Given I am logged in as Greg
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."
```

Tester plusieurs cas avec un seul scénario

```
Scenario Outline: Blenders
   Given I put <thing> in a blender,
    When I switch the blender on
    Then it should transform into <other thing>

 Examples: Amphibians
   | thing          | other thing |
   | Red Tree Frog  | mush        |

 Examples: Consumer Electronics
   | thing          | other thing |
   | iPhone         | toxic waste |
   | Galaxy Nexus   | toxic waste |
```

Paramétrer une “step”

Définition

```
Given the parameter 'level' set to 'network'
```

Implémentation

```python
from behave import given

@given("the parameter '{key}' set to '{value}'")
def step(context, key, value):
    context.q_params = {}
    context.q_params[key] = value
```

Charger une valeur depuis un texte

Définition

```
Given the content
    """
    format=text
    level=network
    * * * ?N? * *
    """
```

Implémentation

```
@given("the content")
def step(context):
    context.q_content = context.text
```

Charger des valeurs depuis une table

Définition

```
Given the parameters
    | key        | value      |
    | level      | network    |
    | format     | text       |
    | startafter | 2020-10-31 |
```

Implémentation

```
@given("the parameters")
def step(context):
    context.q_params = {}
    for row in context.table:
        context.q_params[row['key']] = row['value']
```

Test en erreur

```
@then("the response should have the status code '{code}'")
def step_then_status_code(context, status_code):
    assert context.response.status_code == int(code), "Unexpected status code '%s

@when("I submit a '{query_method}' query to the webservice")
def step_when_query(context, query_method):
    if str(query_method).lower() == 'get':
        params = ['%s=%s' % (k, v) for k, v in context.q_params.items()]
        url = context.q_url + '?' + '&'.join(params)
        context.response = requests.get(url)
    elif str(query_method).lower() == 'post':
        context.response = requests.post(context.q_url, data=context.q_content)
    else:
        raise AssertionError("Unsupported '%s' query method" % query_method)
```

Et plus

- Enchaîner des traitements avant ou après une feature/scénario/step
- Déclencher un seul test en l'appelant par son nom
- Déclencher un ensemble de tests par "feature" ou tags
- Rapport d'exécution
- Intégration framework Web (Pyramid, Flask, Django)
- Intégration aux IDE (ou syntaxe Gherkin)
- Piloter Selenium (via une "fixture")
- Écriture des scénarios Gherkin dans d'autres langues que l'anglais

Conclusion

- outil simple à prendre en main
- syntaxe Gherkin facile à manipuler
- nombreuses possibilités grâce à l'implémentation des steps en Python
- un peu de travail pour "factoriser" des tests (méta-scénarios)
- écosystème Cucumber riche et qui ne se limite pas à Python ;)

Questions ?

Liens

- https://behave.readthedocs.io/
- https://cucumber.io/
- https://robotframework.org/
- https://pytest-bdd.readthedocs.io/
- https://tavern.readthedocs.io/
- https://pypi.org/project/apiritif/