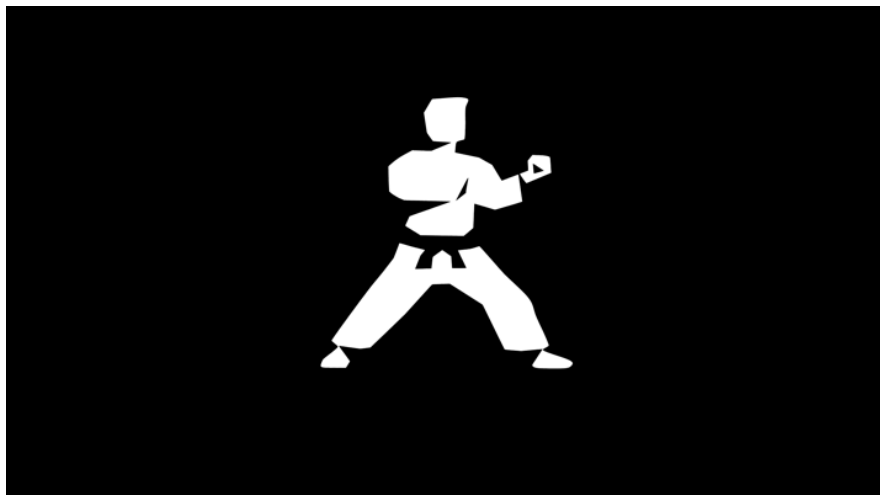




Ceinture noire en test API ? Passez au Dan supérieur avec Karate

Par Mathias DEREMER-ACCETTONE // **Back, Divers**

4 juin 2020 | Temps de lecture : 11 mn



Quand on parle de tests de non régression on pense souvent aux tests côté UI. [Selenium](#), [Cucumber](#), [Cypress](#) ou encore [TestCafé](#), les outils ne manquent pas !

Lorsqu'on souhaite réaliser ce type de tests sur des API, le choix est plus restreint mais on parvient tout de même à trouver chaussure à notre pied. Citons à titre d'exemple [Postman](#) qui, au-delà d'offrir une interface intuitive facilitant la réalisation d'appels aux endpoints de nos applications, permet de créer de véritables scénarios (retrouvez d'ailleurs notre dernier [article sur](#)

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

Après avoir comparé cette solution avec quelques leaders du marché, nous verrons quels sont les principaux atouts de Karate et comment l'utiliser.

Alors enfilez votre kimono, ça va tester !



Quelques champions dans le Dōjō ...

Selenium et Cucumber



Cette première méthode consiste à coupler le Webdriver Selenium avec Cucumber. Pas d'interface permettant d'écrire nos campagnes dans le cas présent : le développeur doit créer une série de fichiers en **Gherkin** dans lesquels seront décrites chaque feature.

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

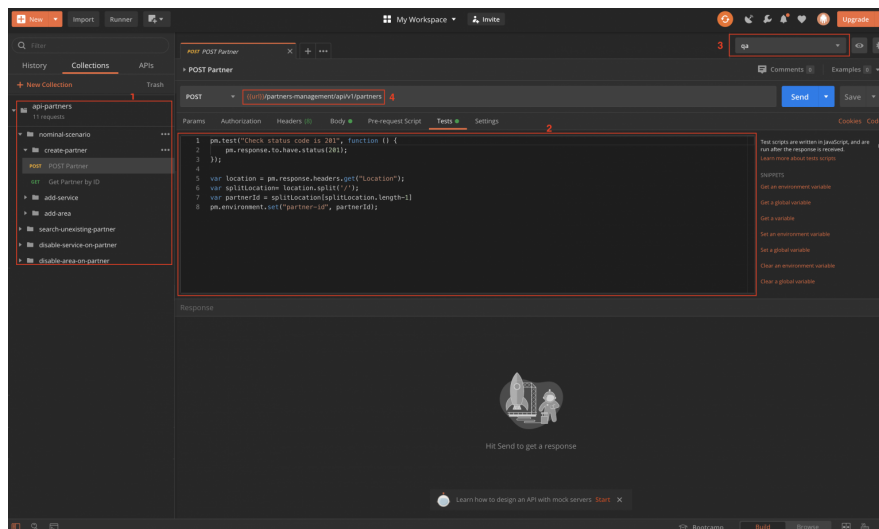
J'ai compris

Le Gherkin est un langage facilement exploitable par les développeurs, mais aussi par les autres membres d'une équipe projet ne sachant pas forcément développer (que nous appellerons "moldu" dans cet article). Ce point est donc une force puisque l'écriture des tests peut être réalisée par n'importe qui (au moins en partie). Néanmoins, au delà du Gherkin, il est également nécessaire d'écrire la "glue" entre les lignes de nos features et les actions à exécuter.

Postman



L'approche proposée par Postman est légèrement différente. Au delà d'un simple client REST doté d'une GUI, nous pouvons avec cet outil créer de véritables campagnes de tests comme le montre le screenshot suivant (1) :



Les différentes vérifications (code http de la réponse, contenu du body...) sont définies en JavaScript dans

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

En savoir plus

J'ai compris

le fait de pouvoir externaliser ses campagnes en Json (et pourquoi pas les versionner avec le code source de l'application) afin de les exécuter automatiquement par un outil de CI via **Newman** (version en ligne de commande de Postman).

Cette méthode est moins complexe que la précédente : moins de code à écrire, syntaxe relativement simple, tests des campagnes possibles via l'interface... Cependant nous sommes dépendant d'un outil tiers pour éditer nos scénarios (les Json exportés n'étant pas très "human readable").

Il ne s'agit ici que d'un échantillon des outils disponibles (nous n'avons pas parlé de Cypress ou de plateformes plus complexes comme Cerberus), mais Postman et Cucumber font sans doute partie des solutions les plus utilisées. Or vous remarquerez que dans chaque cas il y a des avantages, mais aussi quelques inconvénients.

... face à un véritable challenger

Créé par Peter Thomas au sein d'Intuit, Karate se présente sous la forme d'un framework OpenSource combinant automatisation des tests API, création de mocks et de scénarios, et tests de performances.

Le nombre de features proposées est juste hallucinant, et parmi les plus importantes :



Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

[J'ai compris](#)

- Création de tests et de scénarios pour évaluer la non régression d'API.
- Création de mocks, très utile dans le cas où on souhaite, par exemple, valider notre développement, sans être branché sur un véritable environnement (base de données, APIs tiers...)
- Support de l'asynchronisme, ce qui signifie qu'il est possible d'intégrer de façon transparente la gestion des événements provenant de files de messages.
- Evaluation des réponses dynamiques d'API GraphQL.
- Support de REST, mais aussi de SOAP, Websocket, gRPC...
- Intégration de Gatling, permettant d'évaluer la performance de votre application dans vos scénarios.
- Exécution en parallèle des tests, pouvant vous faire gagner un temps précieux dans le cas de tests d'intégrations ou End to End.
- Intégration avec JUnit 4 et 5, mais également utilisable en standalone.
- Automatisation des tests UI (Karate ne se limite pas seulement aux tests APIs !).

Mais au-delà de ces fonctionnalités, ce qui semble le plus intéressant est sans doute l'approche offerte par le framework. L'intégralité des scénarios est écrit en Gherkin (les fans de Cucumber apprécieront 😊), et le plus beau dans tout cela : les actions utilisées dans le DSL sont déjà traduites, nul besoin de Java. On fait du Gherkin et que du Gherkin ! Et c'est là un des atouts

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

sont donc capables d'écrire les tests tout comme les développeurs.

- Les scénarios peuvent facilement être embarqués avec le code source de l'application, et pourront être versionnés de la même manière.
- Contrairement à Postman, plus besoin d'un outil tiers pour écrire les scénarios (votre IDE suffit).

Mais tout comme pour l'art martial, Karate s'apprend en pratiquant alors trêve de théorie et mettons en œuvre quelques Kata.

Hajime !

Pour démontrer la force de frappe de Karate, quoi de mieux qu'une mise en pratique. Pour cela nous allons mettre en œuvre quelques tests de régressions sur le backend d'une application simpliste : la gestion d'une Todo List. L'API qui sera testée n'est donc qu'un simple CRUD écrit en Quarkus : après s'être authentifié, un utilisateur peut lister, créer, supprimer et rechercher des "todo lists" sauvegardées dans une base MongoDB.

Le code complet est disponible [ici](#).

Nos premiers scénarios

La première étape pour utiliser Karate au sein de notre projet consiste à ajouter les dépendances Maven dont nous allons avoir besoin :

```
...  
<dependency>  
  <groupId>com.intuit.karate</groupId>  
  <artifactId>karate-apache</artifactId>  
  <version>0.9.5</version>  
  <scope>test</scope>  
</dependency>  
<dependency>  
  <groupId>com.intuit.karate</groupId>
```

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

Nous créons ensuite une classe `FeatureRunner` dans le répertoire **test** du projet et dont le rôle est référencer et exécuter les scénarios que nous allons écrire :

```
public class FeatureRunner {  
  
    @Karate.Test  
    Karate testNominalScenario() {  
        return Karate.run("features/create-  
todo")  
            .tags("@nominal")  
            .relativeTo(getClass());  
    }  
}
```

Vous observerez deux choses importantes dans cette classe :

- la présence de l'annotation `@Karate.Test`, nous permettant de préciser que la méthode `testNominalScenario` doit être exécutée comme test (les habitués de Junit ne seront normalement pas dépayés)
- l'utilisation de la méthode `tags`, prenant en argument la chaîne de caractères `@nominal`. Celle-ci nous permet de filtrer les scénarios à exécuter pour tel ou tel fichier de features. Nous y reviendrons dans la suite de l'article.

Ecrivons à présent notre premier scénario. Tout comme avec Cucumber, les scénarios sont écrits en Gherkin et regroupés dans des fichiers portant l'extension `.feature`.

Le premier que nous allons créer correspond à un scénario nominal, puisqu'il consiste simplement à s'authentifier, à créer une todo, à la rechercher, puis à la supprimer (chaque étape se soldant par un succès).

```
Feature: Create new todo
```

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

```

quarkus-demo-realm/protocol/openid-connect/token'
  And request { username: 'todo-list-user',
password: 'todo-list-user' }
  And form field username = 'todo-list-user'
  And form field password = 'todo-list-user'
  And form field client_id = 'karate-quarkus-
demo'
  And form field client_secret = '19f745ce-
5452-467f-bad8-ee14184240e5'
  And form field grant_type = 'password'
  When method POST
  Then status 200
  And def accessToken = response.access_token

  # Create a todo
  Given url 'http://localhost:8080/api/todos'
  And header Authorization = 'Bearer ' +
accessToken
  And request {title: "course", description:
"Aller faire les courses", priority : 1}
  When method POST
  Then status 201

  * def location = responseHeaders['Location']
[0]

  # Search a todo
  Given url location
  And header Authorization = 'Bearer ' +
accessToken
  And header Content-Type = 'application/json'
  When method GET
  Then status 200
  And match response contains {title:
'course', priority: 1}

  # Remove a todo
  Given url location
  And header Authorization = 'Bearer ' +
accessToken
  And header Content-Type = 'application/json'
  When method DELETE
  Then status 204

```

Ici notre scénario est annoté avec `@nominal`, libellé qui est utilisé par la méthode `tags` que nous avons précédemment décrit et qui, pour rappel, permet de spécifier les groupes de scénarios à exécuter pour telle ou telle feature (cette valeur étant libre, nous aurions pu

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

des tests via notre outil d'intégration continu préféré : nous pourrions alors très bien envisager de scheduler une exécution automatique de certains cas de tests lors des commits sur feature branches, ou jouer périodiquement l'intégralité des tests depuis la branche **develop** sur un environnement dédié.

Pour en revenir au scénario, ce dernier est écrit en Gherkin. Reprenons la première étape pour en décrire la syntaxe.

```
1.  # Authentication
2.  Given url
    'http://localhost:8180/auth/realms/karate-
    quarkus-demo-realm/protocol/openid-
    connect/token'
3.  And request { username: 'todo-list-user',
    password: 'todo-list-user' }
4.  And form field username = 'todo-list-
    user'
5.  And form field password = 'todo-list-
    user'
6.  And form field client_id = 'karate-
    quarkus-demo'
7.  And form field client_secret = '19f745ce-
    5452-467f-bad8-ee14184240e5'
8.  And form field grant_type = 'password'
9.  When method POST
10. Then status 200
11. And def accessToken =
    response.access_token
```

- En premier lieu nous spécifions l'url qui sera appelée via le mot clé `Given` (ligne 2)
- Les paramètres sont passés via une succession de `And`, permettant de préciser les valeurs de chaque champ (ligne 3 à 8).
- L'appel via le verbe POST est réalisé en utilisant le mot clé `When` (ligne 9).
- On contrôle qu'en retour nous obtenons le code Http 200 (ligne 10).

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

Vous l'aurez compris, l'approche **Given When Then** est quasi identique à celle de Cucumber !

Ecrivons dans la foulée un second scénario, permettant de vérifier le bon fonctionnement de notre API lorsqu'on crée une todo avec des paramètres manquants (ici une todo sans "title" devra retourner une 400 Bad Request).

```
Scenario: Create todo without title should
return a 400 status
  # Authentication
  Given url
    'http://localhost:8180/auth/realms/karate-
    quarkus-demo-realm/protocol/openid-connect/token'
    And request { username: 'todo-list-user',
    password: 'todo-list-user' }
    And form field username = 'todo-list-user'
    And form field password = 'todo-list-user'
    And form field client_id = 'karate-quarkus-
    demo'
    And form field client_secret = '19f745ce-
    5452-467f-bad8-ee14184240e5'
    And form field grant_type = 'password'
  When method POST
  Then status 200
  And def accessToken = response.access_token

  # Create a todo
  Given url 'http://localhost:8080/api/todos'
  And header Authorization = 'Bearer ' +
  accessToken
  And request {description: "Aller faire les
  courses", priority : 1}
  When method POST
  Then status 400
```

Optimiser nos scénarios et parfaire notre technique

Vous remarquerez que plusieurs choses peuvent être améliorées dans nos scénarios :

- La phase d'authentification est toujours la même, quelque soit le scénario. Celle-ci pourrait être centralisée afin d'éviter les duplications

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

- Les URL et credentials sont en dur dans nos scénarios, les rendant inutilisables si nous changeons d'environnement (à moins d'en refactorer le code).

Factorisation du code

Tout d'abord voyons comment centraliser l'étape d'authentification et en faire une "fonction" réutilisable.

Commençons par créer une nouvelle feature dans fichier `authentification.feature` :

```
Feature: Reusable authentication feature

Scenario:
  Given url authUrl
  And request { username: '#(username)',
password: '#(password)', clientId: '#(clientId)',
clientIdSecret: '#(clientIdSecret)'}
  And form field username = username
  And form field password = password
  And form field client_id = clientId
  And form field client_secret = clientIdSecret
  And form field grant_type = 'password'
  When method POST
  Then status 200
  And def accessToken = response.access_token
```

Le code est le même que précédemment, nous avons simplement variabilisé certains paramètres comme le `username` ou le `password`.

Ajoutons ensuite le bloc suivant au début de `create-todo.feature`

```
Background:
  * def signIn = call
read('authentication.feature') { username: 'todo-
list-user', password: 'todo-list-user', clientId:
'karate-quarkus-demo', clientIdSecret: '19f745ce-
5452-467f-bad8-ee14184240e5' }
  * def accessToken = signIn.accessToken
```

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

obtenu dans une variable réutilisable dans l'ensemble de nos scénarios.

A ce stade, nous pouvons retirer les steps d'authentification de nos scénarios, cette phase étant désormais prise en charge par le bloc précédemment décrit.

Externalisation des données

A présent, voyons comment externaliser les données utilisées dans nos requêtes, au sein d'un fichier json réutilisable dans nos scénarios.

Commençons par créer un répertoire "data" dans lequel nous ajouterons ensuite le fichier contenant le json suivant :

```
{
  "title" : "course",
  "description" : "Aller faire les courses",
  "priority" : 1
}
```

Modifions le fichier `create-todo.feature` comme suit :

```
Background:
...
* def todoJson = read('data/todo.json')
```

Le bloc précédent permet de placer le contenu de `todo.json` dans la variable `todoJson`. Nous pourrons ensuite réutiliser cette variable dans nos scénarios :

```
@nominal
Scenario: Create todo should return 201
status
  # Create a todo
  Given url 'http://localhost:8080/api/todos'
  And header Authorization = 'Bearer ' +
```

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

Gestion des configurations

Comme nous l'évoquions un peu plus haut, les urls et identifiants nécessaires au déroulement de nos scénarios (et utilisés notamment lors de l'authentification) sont définis en dur. Au delà de l'aspect sécurité, cela limite la souplesse de nos features : impossible d'exécuter une campagne de tests sur des environnements différents, si il n'est pas possible de switcher de configuration dans nos scénarios.

Encore une fois Karate a la solution.

Créons un fichier `karate-config.js` dans le dossier **resources** du répertoire **test** de notre projet. Ce dernier comportera une simple fonction :

```
1.  function init() {
2.      var env = karate.env;
3.      karate.log('karate.env selected
environment was:', env);
4.      if (!env) {
5.          env = 'local';
6.      }
7.      var config = {
8.          env: env,
9.          clientId: 'karate-quarkus-demo',
10.         clientSecret: '19f745ce-5452-467f-
bad8-ee14184240e5',
11.         authUrl:
'http://localhost:8180/auth/realms/karate-
quarkus-demo-realm/protocol/openid-
connect/token',
12.         apiBaseUrl:
'http://localhost:8080/api'
13.     };
14.     if (env == 'dev') {
15.         config.clientId= 'karate-quarkus-
demo',
16.         config.clientSecret= '29f745af-5452-
487f-bad8-ee14354141a9',
17.         config.authUrl=
'http://192.168.1.17:8180/auth/realms/karate-
quarkus-demo-realm/protocol/openid-
connect/token',
```

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

```

5000);
23.     karate.configure('readTimeout', 5000);
24.     return config;
25. }

```

Nous créons une configuration par défaut (ligne 7 à 13) pouvant être surchargée en fonction d'une variable d'environnement (ligne 14 à 21).

Les paramètres définis dans ce fichier sont utilisables librement dans les features :

```

...
* def resourceUrl = apiUrl + '/todos'
...

@nominal
Scenario: Create todo should return 201 status
  # Create a todo
  Given url resourceUrl
  And header Authorization = 'Bearer ' +
accessToken
  And request todoJson
  When method POST
  Then status 201

```

Et voilà ! Il ne reste plus qu'à lancer nos tests via la commande :

```

mvn clean test -DargLine="-Dkarate.env=local" -
Dtest=FeatureRunner

```

```

mai 25, 2020 6:36:25 PM com.intuit.karate.Logger info
INFO: karate.env selected environment was:
-----
feature: classpath:com/ineat/karate/quarkus/demo/nrt/features/test.feature
scenarios: 1 | passed: 1 | failed: 0 | time: 2,245s
-----
HTML report: (paste into browser to view) | Karate version: 0.9.5
file:/Users/mathiasder/IdeaProjects/karate-quarkus-demo/target/surefire-reports/com.ineat.karate.quarkus.demo.nrt.features.test.html
-----
Process finished with exit code 0


```

Comme indiqué dans la console, un rapport html contenant le détail de l'exécution est généré.

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

 Test Suite Navigation					
# of failed tests: 0/32 (0.00%)					
# of skipped tests: 0/32 (0.00%)					
# of passed tests: 32/32 (100.00%)					
1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32				
Scenario (1.11) Create todo should return 201 status [Test 1 : 'def signIn = call read/authentication.feature' { username: 'todo-list-user', password: 'todo-list-user', clientId: 'karate-quarkus-demo', clientSecret: '19f745ce-5452-4e71-ba08-ea14184240d0' }] 1.050463 [Test 2 : 'com/intuit/karate/quarkusdemo/rh/features/authentication.feature'] 1.038336 [Test 3 : 'Given url baseUrl'] 0.000043 [Test 4 : 'And request { username: 'tusername', password: 'tpassword', clientId: 'tclientId', clientSecret: 'tclientSecret' }] 0.014658 [Test 5 : 'And form field username = username'] 0.000107 [Test 6 : 'And form field password = password'] 0.000017 [Test 7 : 'And form field client_id = clientId'] 0.000014 [Test 8 : 'And form field client_secret = clientSecret'] 0.000016 [Test 9 : 'And form field grant_type = 'password''] 0.000464 [Test 10 : 'When method POST'] 1.013583 [Test 11 : 'Then status 200'] 0.000026 [Test 12 : 'And def accessToken = response.accessToken'] 0.007181 [Test 13 : 'def accessToken = signIn.accessToken'] 0.004934 [Test 14 : 'def todoJson = read('data/todo.json')'] 0.000018 [Test 15 : 'def requestBody = json(todoJson)'] 0.000019 [Test 16 : 'Given url baseUrl'] 0.000015 [Test 17 : 'And header Authorization = 'Bearer ' + accessToken'] 0.000071 [Test 18 : 'And request todoJson'] 0.000014 [Test 19 : 'When method POST'] 0.999997 [Test 20 : 'Then status 201'] 0.000009 [Test 21 : 'def location = response.headers['Location'][0]'] 0.000008 [Test 22 : 'Given url location'] 0.000014 [Test 23 : 'And header Authorization = 'Bearer ' + accessToken'] 0.000011 [Test 24 : 'And header Content-Type = 'application/json''] 0.001186 [Test 25 : 'When method GET'] 0.115602 [Test 26 : 'Then status 200'] 0.000001 [Test 27 : 'And match response contains { title: 'course', priority: 1 }'] 0.00104 [Test 28 : 'Given url location'] 0.000001 [Test 29 : 'And header Authorization = 'Bearer ' + accessToken'] 0.000011 [Test 30 : 'And header Content-Type = 'application/json''] 0.000003 [Test 31 : 'When method DELETE'] 0.046635 [Test 32 : 'Then status 204'] 0.000019					

Des APIs plus rapides que Bruce Lee ?

Nous nous écartons un peu des tests de non régressions mais je ne pouvais pas conclure sur Karate sans vous parler rapidement de son intégration avec Gatling. Nous l'évoquions au début de cet article : il est possible de tester la performance de vos APIs avec Karate, et ce simplement en ajoutant une dépendance au projet :

```
<dependency>
  <groupId>com.intuit.karate</groupId>
  <artifactId>karate-gatling</artifactId>
  <scope>test</scope>
</dependency>
```

A partir de là, vous pouvez réutiliser vos scénarios Karate en tant que tests de performances exécutés par Gatling. Le seul code à écrire sera le modèle de chargement des virtual users, tout le reste est réalisable à partir de Karate.

Vous disposerez même de quelques options pour distribuer vos tests de charges, en multipliant le nombre de noeuds à disposition par ou en passant par des conteneurs Docker par exemple.

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

En savoir plus

J'ai compris

Victoire par Ippon ?

Karate est un outil complet qui, malgré l'absence d'interface, saura se faire une place auprès des moldus : pas de code complexe à écrire, pas de glue pour traduire les actions. Les scénarios s'écrivent en Gherkin uniquement, ce qui permet à n'importe qui de s'y mettre !

Nous venons de voir qu'une infime partie des capacités de Karate mais cette première approche aura permis de démontrer que le framework n'a rien à envier à ses adversaires. Je ne peux donc que vous conseiller de le mettre en pratique : vous verrez, l'essayer c'est l'adopter !

Liens utiles

Quelques liens utiles pour ceux d'entre vous qui souhaiteraient remonter sur le tatami et aller plus loin avec Karate.

- <https://intuit.github.io/karate/> – Site officiel du projet
- <https://github.com/intuit/karate> – Code source
- <https://cucumber.io/docs/gherkin/reference/> – Syntaxe Gherkin
- <https://github.com/intuit/karate/tree/master/karate-gatling> – Tests de performance avec Gatling
- <https://www.youtube.com/watch?v=NYIPxd5dZOU> – Présentation au Devovx 2018
- <https://blog.ineat-group.com/2020/05/exemples-dutilisation-de-postman-newman/> – Exemples d'utilisation de Postman
- <https://github.com/ineat/karate-quarkus-demo> – Code source de l'exemple

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

J'ai compris

[Java](#)[Karate](#)[Rest Api](#)[Tests](#)

[Retour](#)

Vous pouvez aussi aimer



Securisez vos APIs Spring avec Keycloak : #2 – Paramétrage d'un domaine Keycloak

Par Ludovic Dussart [// Back](#) 14 novembre 2017

| Temps de lecture : 5 mn



Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

[J'ai compris](#)

StackStorm, l'automatisation pilotée par les événements

Par Simon Steinmetz // **Divers** 2 décembre 2020

| Temps de lecture : 10 mn



INEAT Group © 2021 - Tous droits réservés. [Mentions légales](#)

Ce site utilise des cookies Google Analytics à des fins de mesure d'audience. En poursuivant votre navigation, vous acceptez l'utilisation de ces cookies.

[En savoir plus](#)

[J'ai compris](#)