



adaptTo()

APACHE SLING & FRIENDS TECH MEETUP
10-12 SEPTEMBER 2018

Example code at
<https://tinyurl.com/potsdam2018>




Karate, the black belt of HTTP API testing?

Bertrand Delacretaz :: @bdelacretaz :: grep.codeconsult.ch

Principal Scientist, Adobe Basel

Sling PMC Member, Member of the Board of Directors, Apache Software Foundation

Karate?



Karate

Web-Services Testing Made **Simple.**

maven central **0.8.0.RC8** Build Status release **v0.8.0** support slack

Hello World

```
Scenario: create and retrieve a cat

Given url 'http://myhost.com/v1/cats'
And request { name: 'Billie' }
When method post
Then status 201
And match response == { id: '#notnull', name: 'Billie' }

Given path response.id
When method get
Then status 200
```

Annotations:

- JSON is 'native' to the syntax
- Intuitive DSL for HTTP
- Payload assertion in one line
- Second HTTP call using response data

<https://github.com/intuit/karate>
Efficient (and fun?) way of testing HTTP services.

Created by Peter Thomas of Intuit.

V0.1.2 in February 2017

Current 0.8.0, moving away from Cucumber

Example code at
<https://tinyurl.com/potsdam2018> !

Great docs & demos!

Index

Start	Maven Gradle Quickstart Folder Structure Naming Conventions Script Structure
Run	JUnit TestNG Command Line IDE Support Tags / Grouping Parallel Execution Java API
Report	Configuration Environment Switching Reports JUnit HTML Report Logging
Types	JSON XML JavaScript Functions Reading Files Type / String Conversion Floats and Integers Embedded Expressions JsonPath XPath Karate Expressions
Variables	def text table yaml string json xml xmlstring copy
Actions	assert print replace get set remove configure call callonce eval read() karate API
HTTP	url path request method status soap action
Request	param header cookie form field multipart file multipart field multipart entity params headers cookies form fields multipart files multipart fields
Response	response responseStatus responseHeaders responseCookies responseTime requestTimeStamp
Assert	match == match != match contains match contains only match contains any match !contains match each Fuzzy Matching contains short-cuts Schema Validation
Re-Use	Calling Other *.feature Files Data Driven Features Calling JavaScript Functions Calling Java Code Commonly Needed Utilities Data Driven Scenarios
Advanced	Polling Conditional Logic Before / After Hooks HTTP Basic Auth Header Manipulation GraphQL
More	Mock Servlet Test Doubles Performance Testing Karate UI Karate vs REST-assured Karate vs Cucumber Examples and Demos

Karate Demo

This is a sample [Spring Boot](#) web-application that exposes some functionality as web-service end-points. And includes a set of Karate examples that test these services as well as demonstrate various Karate features and best-practices.

Example	Demonstrates
greeting.feature	Simple GET requests and multiple scenarios in a test.
headers.feature	Multiple examples of header management including dynamic setting of headers for each request using a JS file (classpath:headers.js). Also demonstrates handling of cookies, and path / query parameters. There are also examples of how to set up a re-usable *.feature file for per-request secure / auth headers after a sign-in. An OAuth 2 sample is also available.
sign-in.feature	HTML form POST example. Typically you use the response to get an authentication token that can be used to build headers for subsequent requests. This example also demonstrates getting past an end-point protected against CSRF .
cats.feature	Great example of embedded-expressions (or JSON / XML templating). Also shows how to set the Accept header for getting XML from the server.
kittens.feature	Reading a complex payload expected response from a file . You can do the same for request payloads as well.
read-files.feature	The above example reads a file with embedded expressions and this one reads normal JSON and XML for use in a match . Also a good example of using the set and table keywords to build JSON (or XML) payloads from scratch.
graphql.feature	GraphQL example showing how easy it is to prepare queries and deal with the highly dynamic and deeply nested JSON responses - by focusing only on the parts you are interested in
upload.feature	Multi-part file-upload example, as well as comparing the binary content of a download. Also shows how to assert for expected response headers . Plus an example of how to call custom Java code from Karate.
dogs.feature	How to easily use Java interop to make a JDBC / database call from a Karate test. Here is the utility-class code - which depends on Spring JDBC : Dbutils.java . The same approach can

Excerpts from <https://github.com/intuit/karate>



VERY readable tests - black belt!

Scenario: create and retrieve a cat

Given url 'http://myhost.com/v1/cats'

And request { name: 'Billie' }

When method **post**

Then status 201

And match response == { id: '#notnull', name: 'Billie' }

Given path **response.id**

When method **get**

Then status 200

JSON is 'native'
to the syntax

Intuitive DSL
for HTTP

Payload
assertion in
one line

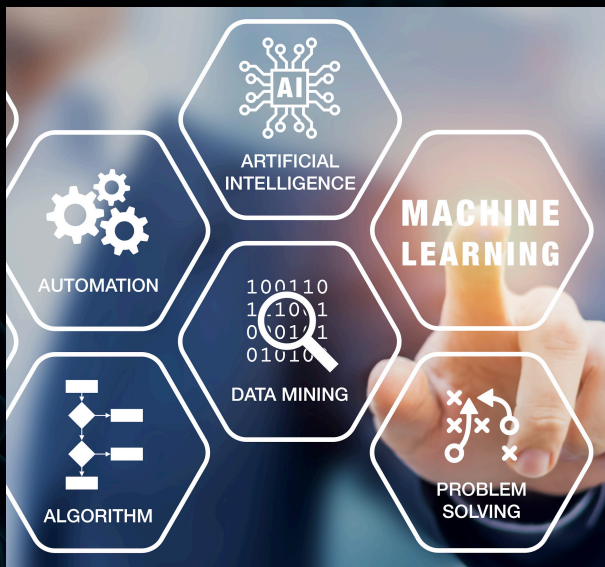
Second HTTP
call using
response data

“Laser focused on making things as simple as possible”

Example from <https://github.com/intuit/karate>

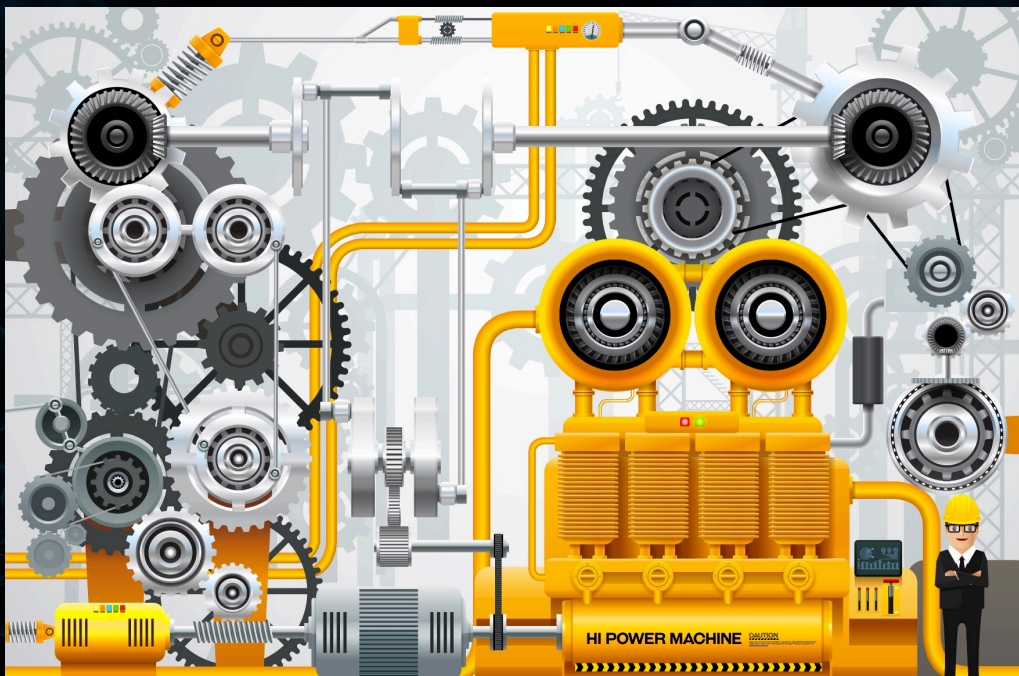
Karate, the black belt of HTTP API testing? - Bertrand Delacretaz - adaptTo() 2018

HTTP testing in a world of services...



Clients
(usually machines)

HTTP
service
API



Opaque Services

THIS is the “sacred boundary” that needs to be
VERY SERIOUSLY tested and documented.

Karate tests as HTTP API reference documentation?

```
# -----  
Scenario: Create a resource, update, read back, delete
```

```
# -----  
  
# Create a resource  
Given path testFolderPath, '*'  
And form field f1 = 'v1A' + testID  
And form field f2 = 'v2A'  
When method POST  
Then status 201
```

```
# The Location header provides the path where the resource was created  
* def resourcePath = responseHeaders['Location'][0]
```

```
# Read back  
Given path resourcePath + '.json'  
When method GET  
Then status 200  
  
And match response.f1 == 'v1A' + testID  
And match response.f2 == 'v2A'
```



The screenshot shows the Apache Sling documentation page for 'Manipulating Content - The SlingPostServlet (servlets.post)'. The page is part of the 'Documentation' section, specifically under 'Bundles'. It lists various links for getting started, development, and tutorials. The main content area is titled 'Manipulating Content - The SlingPostServlet (servlets.post)' and contains a list of links for multiple ways to modify content, including content creation, removal, copying, moving, and importing. It also includes a section for 'Special Parameters' and 'Response format'. The page is branded with the Sling and Apache logos.

A great complement to
overview + concepts docs !

Testing Apache Sling with Karate

and demonstrating major Karate features

Test Feature: basic structure

```
# -----  
@importcontent @postservlet  
Feature: Import content using the Sling POST Servlet  
# -----  
  
Background:  
* url baseUrl  
  
# Use admin credentials for all requests  
* configure headers = call read('classpath:util/basic-auth-header.js')  
  
# Sling instance ready?  
* eval karate.call('classpath:util/sling-ready.feature')  
  
* def testID = '' + java.util.UUID.randomUUID()  
* def testFolderPath = 'importContentTest/' + testID  
  
# -----  
Scenario: Create the parent folder, import JSON content, verify and delete  
# -----
```

<- Tags for test selection

<- Tests are “features”

<- Background section for setup

<- Call javascript or other features

<- Use Java code to compute
unique test paths

<- We’re ready to run Scenarios

Sling PostServlet Scenario

```
# -----  
Scenario: Create a resource, update, read back, delete  
# -----  
  
# Create a resource  
Given path testFolderPath, '*'  
And form field f1 = 'v1A' + testID  
And form field f2 = 'v2A'  
When method POST  
Then status 201  
  
# The Location header provides the path where the resource was created  
* def resourcePath = responseHeaders['Location'][0]  
  
# Read back  
Given path resourcePath + '.json'  
When method GET  
Then status 200  
And match response.f1 == 'v1A' + testID  
And match response.f2 == 'v2A'
```

```
# Overwrite one field and add a new one  
Given path resourcePath  
And form field f2 = 'v2B'  
And form field f3 = 'v3B'  
When method POST  
Then status 200  
  
# Read modified resource back  
Given path resourcePath + '.json'  
When method GET  
Then status 200  
And match response.f1 == 'v1A' + testID  
And match response.f2 == 'v2B'  
And match response.f3 == 'v3B'
```

CAN U RD THS ?

Define JSON structures, verify (match) responses

```
# Import content
Given path parentFolder
And form field :operation = 'import'
And form field :contentType = 'json'
And form field :name = testID
And form field :content = newContent
When method POST
Then status 201

# Verify imported content
Given path parentFolder, testID + ".json"
When method GET
Then status 200
And match $ == newContent
And match $.p2[2] == testID
```

```
* def newContent =
"""
{
  'jcr:primaryType' : 'nt:unstructured',
  p1 : '#(testID)',
  p2: [ 'a', 'b', '#(testID)' ]
}
"""
```

Testing the “import” operation of the SlingPostServlet

Uploading an image

```
# Create a resource
```

```
Given path testFolderPath + "/*"
```

```
And multipart field file = read("classpath:images/testimage.jpg")
```

```
And multipart field name = filename
```

```
When method POST
```

```
Then status 201
```

```
# Read metadata back and verify
```

```
Given path imagePath + '.tidy.5.json'
```

```
When method GET
```

```
Then status 200
```

```
And match response.jcr:primaryType == 'nt:unstructured'
```

```
And match response.name == filename
```

```
And match response.file == expectedFile
```

Schema-like
matching, see
next slide

Schema-like matching

```
* def expectedFile =
  """
  {
    "jcr:primaryType" : "nt:resource",
    "jcr:mimeType" : "application/octet-stream",
    "jcr:lastModifiedBy" : #string,
    "jcr:lastModified" : #string,
    ":jcr:data" : "10102",
    "jcr:uuid" : #uuid
  }
  """
```

Used in:

And match response.file == expectedFile

Marker	
#ignore	
#null	
#nonnull	
#present	
#notpresent	
#array	
#object	

#boolean	
#number	
#string	
#uuid	
#regex STR	
#? EXPR	
#[NUM] EXPR	
#(EXPR)	

Matching the image itself!

```
# Read the image itself back and verify
Given path imagePath + '/file/jcr:data'
When method GET
Then status 200
And match header Content-Type == 'application/octet-stream'
And match response == read("classpath:images/testimage.jpg")
And match header Content-Length == "10102"
```

“Laser focused on making things as simple as possible”

It's not much code!

```
# The actual tests
src/test/java/sling/postservlet/importContent.feature
src/test/java/sling/postservlet/createContent.feature
src/test/java/sling/setup/initialContent.feature
src/test/java/sling/filestorage/uploadImage.feature

# JUnit tests "proxy"
src/test/java/sling/SlingTest.java (0 ALOC)

# Gatling performance tests "proxy"
src/test/scala/sling/PostServletSimulation.scala (~30 ALOC)

# Test utilities (all small)
src/test/java/util/cleanup-paths.js
src/test/java/util/basic-auth-header.js
src/test/java/util/sling-ready.feature
src/test/java/util/cleanup-path.feature
src/test/java/logBaseURL.js

# Test content
src/test/java/images/testimage.jpg

# Tests configuration
src/test/java/karate-config.js
src/test/java/logback-test.xml
```

Interlude: Running the Tests

aka “live demo”

Running Karate tests

Using Maven via trivial JUnit proxy tests, selected by tags:

```
1 Scenarios (1 passed)
29 Steps (29 passed)
0m0.138s
Karate version: 0.8.0
html report: (paste into browser to view)

-----
file:/Users/bertrand/workspace/apache/sling/sling-whiteboard/karate-http-testing/target/Content.html
```

```
1 Scenarios (1 passed)
25 Steps (25 passed)
0m0.171s
Karate version: 0.8.0
html report: (paste into browser to view)

-----
file:/Users/bertrand/workspace/apache/sling/sling-whiteboard/karate-http-testing/target/Content.html
```



Test Suite Navigation

Results :

of failed tests: 1/35

Scenario: Create a resource, update, read back, delete

```
Test 1 : * url baseUrl 0.005969
Test 2 : * configure headers = call read('classpath:util/basic-auth-header.js') 0.00105
Test 3 : * eval karate.call('classpath:util/sling-ready.feature') 0.01489
Test 4 : * call (Check access to the Sling instance under test) [Check access to HTTP root] classpath:util/sling-ready.feature 0
Test 5 : * url baseUrl 0
Test 6 : * configure headers = call read('classpath:util/basic-auth-header.js') 0
Test 7 : Given path '/json' 0
Test 8 : When method GET 0
Test 9 : Then status 200 0
Test 10 : * json response.json = 'java.util.UUID.randomUUID()' 0.000907
Test 11 : * json response.folderPath = 'createContentTest' + testID 0.008103
Test 12 : * json response.testFolderName = 'testFolderName' + testID 0.001932
Test 13 : * json response.eld1 = 'v1A' + testID 0.002438
Test 14 : * json response.eld2 = 'v2A' + testID 0.001496
Test 15 : * json response.eld3 = 'v3B' + testID 0.016712
Test 16 : * json response.eld4 = 'v4B' + testID 0.000084
Test 17 : * json response.eld5 = 'v5A' + testID 0.000051
Test 18 : * json response.eld6 = 'v6A' + testID 0.002028
Test 19 : * json response.eld7 = 'v7A' + testID 0.006173
Test 20 : * json response.eld8 = 'v8A' + testID 0.000063
Test 21 : * json response.eld9 = 'v9A' + testID 0.000305
Test 22 : * json response.eld10 = 'v10A' + testID 0.000027
Test 23 : * json response.eld11 = 'v11A' + testID 0.000058
Test 24 : * json response.eld12 = 'v12A' + testID 0.001992
Test 25 : * json response.eld13 = 'v13A' + testID 0.001606
Test 26 : * json response.eld14 = 'v14A' + testID 0.010848
Test 27 : * json response.eld15 = 'v15A' + testID 0.000113
Test 28 : Given path resourcePath + '.json' 0.001853
Test 29 : When method GET 0.013098
```

Karate UI, basic but useful.

Variable	Type	Value
toDelete	js[]	["/uploadImageT...
responseTime	num	7
imagePath	str	/uploadImageTes...
requestMethod	str	GET
requestParams	null	
requestUri	str	http://localhost:8...

When method GET

View raw Request/Response

```
responseStatus : 200
responseHeaders : {
  "Date": [
    "Wed, 05 Sep 2018 15:06:40 GMT"
  ],
  "X-Content-Type-Options": [
    "nosniff"
  ],
  "X-Frame-Options": [
    "SAMEORIGIN"
  ]
}
```

HTML test reports with debug info,
Gatling performance reports

More Fun & Useful Features

It's not over yet

Reuse Karate features for Gatling performance tests

```
class PostServletSimulation extends Simulation {

  // Declare path patterns used by our Karate tests, to group similar
  // requests in the Gatling report. Optionally add delays to specific
  // HTTP request methods.
  val protocol = karateProtocol(
    "/createContentTest/{folder}/{testResource}" -> pauseFor("post" -> 0),
    "/createContentTest/{folder}" -> pauseFor("post" -> 0),
    ...excerpted for slides...
  )

  // Which Karate features do we want to test?
  val createContent = scenario("create")
    .exec(
      karateFeature("classpath:sling/postservlet/createContent.feature")
    )
  val importContent = scenario("import")
    .exec(
      karateFeature("classpath:sling/postservlet/importContent.feature")
    )

  // Define Gatling load models
  setUp(
    createContent.inject(rampUsers(75) over (5 seconds)).protocols(protocol),
    importContent.inject(rampUsers(125) over (3 seconds)).protocols(protocol),
  )
}
```



Scenario Outlines for data-driven testing

```
# -----  
Scenario Outline: Validate Initial Content
```

```
# -----  
Given path '<contentPath>' + '.json'  
When method GET  
Then status 200  
And match $.<jsonPath> == "<value>"
```

```
# -----  
# Data values used by the Scenario outline  
# -----
```

Examples:

```
| contentPath | jsonPath | value |
```

```
# Sling starter content
```

```
| starter/css | jcr:primaryType | sling:Folder |  
| starter/css/bundle.css | jcr:createdBy | admin |  
| starter/index.html/jcr:content | jcr:mimeType | text/html |
```

```
# OSGi console
```

```
| system/console/bundles | data[0].symbolicName | org.apache.felix.framework |
```

```
# Empty path means root
```

```
| | sling:target | /starter/index.html |  
| | sling:resourceType | sling:redirect |
```

Validating initial content in
a default Sling instance

Test Doubles: mock (main or auxiliary) services

```
# -----  
Scenario: pathMatches('/cats/{id}')
```

```
* def result = cats[pathParams.id]  
* eval if(!result) abortWithStatus(404)  
* def responseHeaders = { 'Content-Type': 'application/json'  
* def response = result
```

```
# -----  
Scenario: pathMatches('/cats') && methodIs('post')
```

```
* def cat = request  
* def id = uuid()  
* set cat.id = id  
* eval cats[id] = cat  
* def response = cat  
* def responseStatus = 201
```

<https://github.com/bdelacretaz/karate-mini-mocks>

```
# -----  
Scenario: pathMatches('/cats/{id}') && acceptContains('text/xml')
```

```
* def result = cats[pathParams.id]  
* eval if(!result) abortWithStatus(404)  
* def responseHeaders = { 'Content-Type': 'text/xml' }  
* def response = <cat><id>#{result.id}</id><name>#{result.name}</name></cat>
```

Scenarios for
mocking HTTP
services,
similar syntax.

Not much code for the Test Doubles either

Mocked server + tests

src/test/java/server/server.feature

src/test/java/client/client.feature

JUnit test "proxies"

src/test/java/server/TestBase.java

src/test/java/client/ClientTest.java

Tests configuration

src/test/java/logback-test.xml

src/test/java/karate-config.js

```
public class TestBase {  
  
    private static FeatureServer server;  
  
    @BeforeClass  
    public static void setup() {  
        File file = FileUtils  
            .getFileRelativeTo(TestBase.class, "server.feature");  
        server = FeatureServer  
            .start(file, 0, false, null);  
        System.setProperty("karate.env", "mock");  
        System.setProperty(  
            "mock.server.url",  
            "http://localhost:" + server.getPort());  
    }  
  
    @AfterClass  
    public static void cleanup() {  
        if(server != null) {  
            server.stop();  
            server = null;  
        }  
    }  
}
```

Coda

Now it's over. Almost.



adaptTo()



Adobe



Karate

Web-Services Testing Made Simple.

maven central 0.8.0.RC8 Build Status release v0.8.0 support slack

Hello World

Scenario: create and retrieve a cat

Given url 'http://myhost.com/v1/cats'

And request { name: 'Billie' }

When method post

Then status 201

And match response == { id: '#notnull', name: 'Billie' }

Given path response.id

When method get

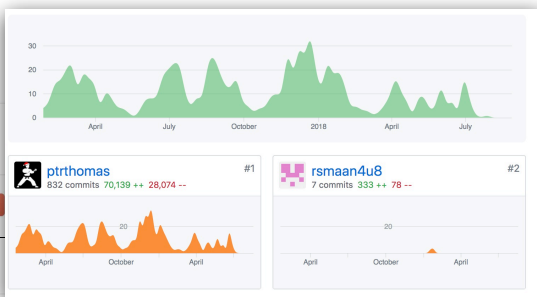
Then status 200

JSON is 'native'
to the syntax

Intuitive DSL
for HTTP

Payload
assertion in
one line

Second HTTP
call using
response data



Powerful, efficient, fun
and READABLE tests!
Open for contributions
Good enough to
document HTTP APIs?

Example code at
<https://tinyurl.com/potsdam2018> !

<https://github.com/intuit/karate>

Karate, the black belt of HTTP API testing? - Bertrand Delacretaz - @bdelacretaz - adaptTo() 2018

