Public

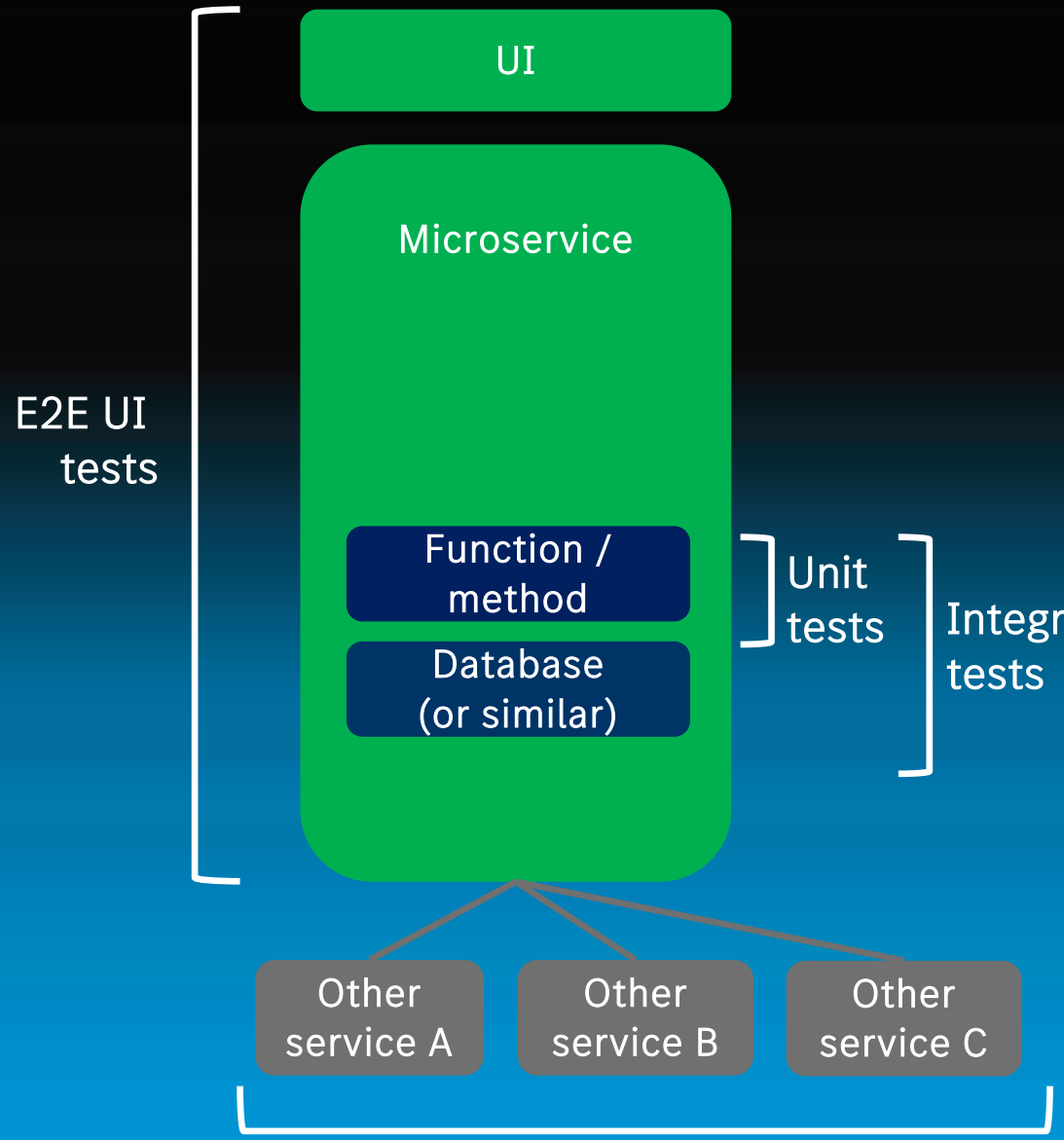
Karate und Gatling – Testautomatisierung und integrierte Lasttests

Johannes Schützner
Mercedes-Benz R&D

Java Forum Stuttgart
10. Juli 2025

Mercedes-Benz

Software Testing



Functional tests – verify business logic

Acceptance tests – verify customer requirements

Smoke tests – verify deployment

Permission tests – verify real roles

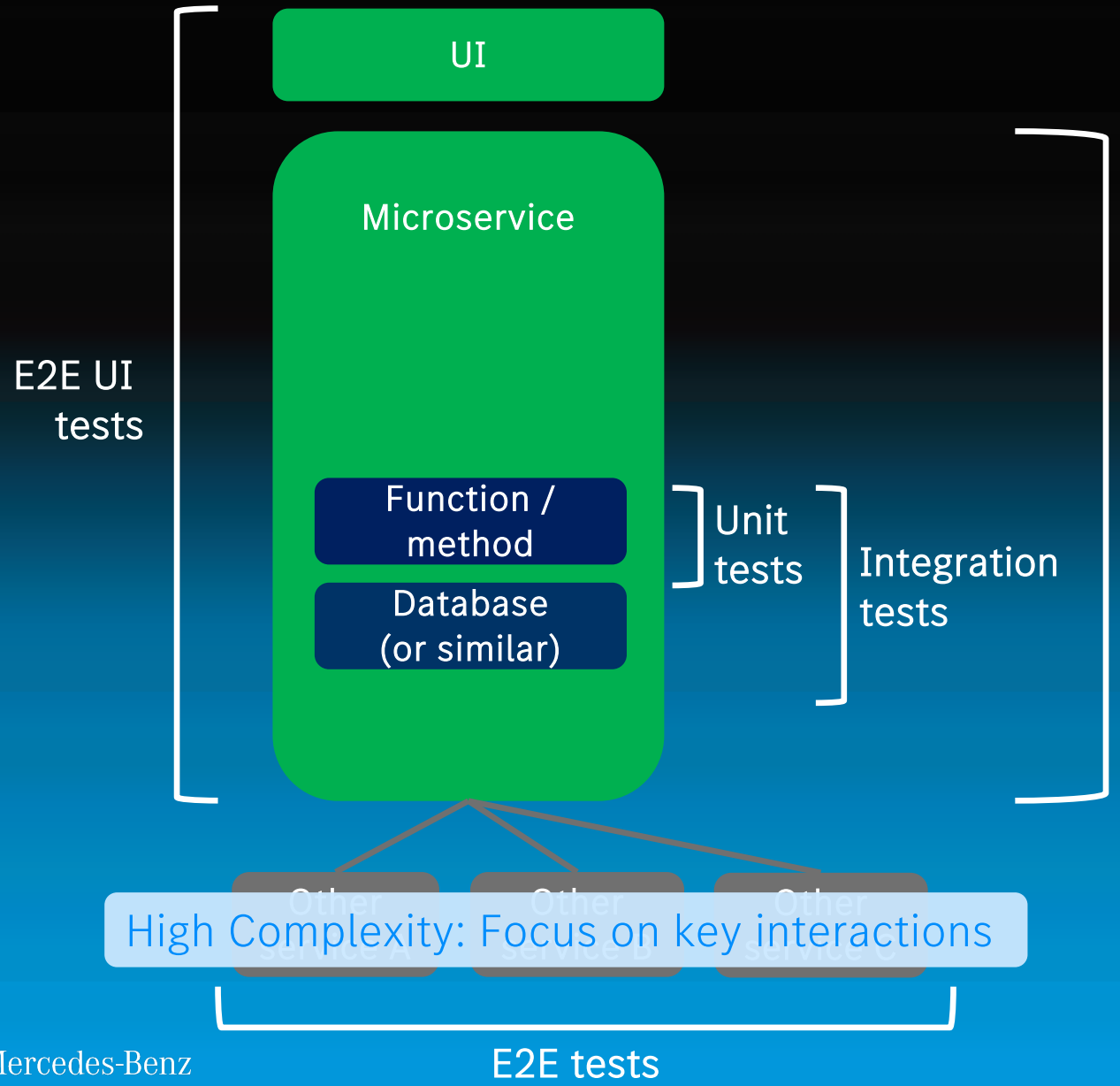
Destructive tests - verify error handling

Regression tests – verify major code changes

Load tests – verify workloads

Frequency: Manual, automated, continuous

Software Testing



Functional tests – verify business logic

Acceptance tests – verify user stories

Smoke tests – verify deployment

Permission tests – verify real roles

Destructive tests - verify error handling

Regression tests – verify major code changes

Load tests – verify workloads

Karate API tests & Gatling load tests

Unified tests for multiple purposes

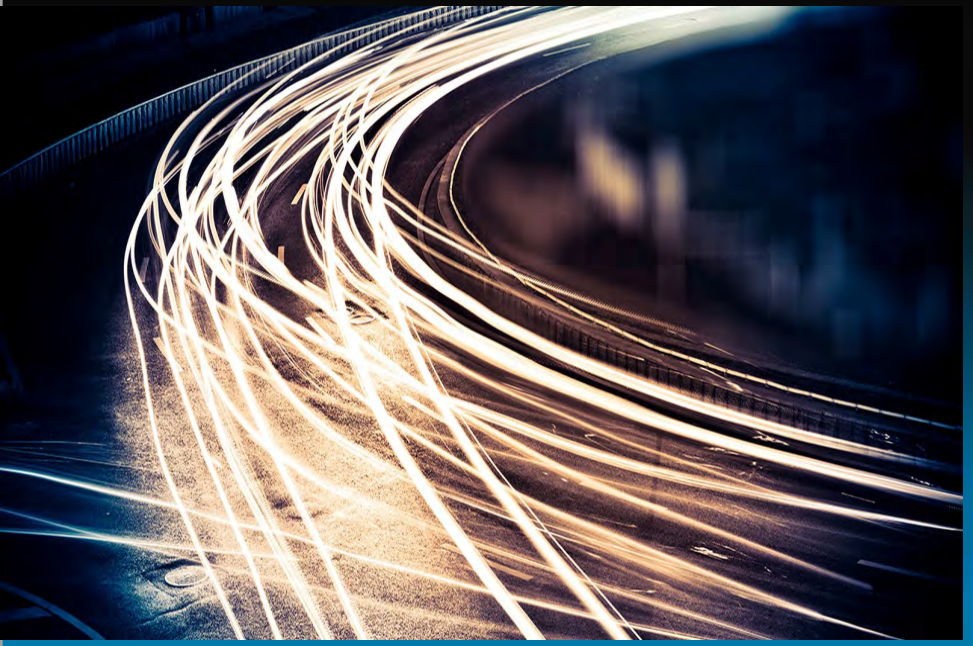
→ Outside perspective
→ Serve as fitness functions
→ Extend to BDD (Behavior-Driven Development)

Frequency: Manual, automated, continuous

Integrate in CI/CD pipelines
→ Continuous testing (CT)

Karate API Tests

Karate API Test Framework - Overview



- Open Source framework for API test automation
- Seamless integration into CI/CD pipelines
- Parallel execution to reduce runtimes
- Data-driven tests
- Built-in support for switching configs (e.g., Dev, QA, Nonprod)
- Async Kafka API testing (requires Pro license)

www.karatelabs.io

Karate API Tests - Suitable Language

```
Given url "postman-echo.com/get"
And header demo = 'sample'
When method GET
Then status 200
And match
 response.headers.demo == 'sample'
```

- Syntax is language neutral – designed for HTTP and JSON
- Scripts are plain-text and super readable
- No compilation needed, no helper code needed
- Based on Gherkin for BDD (Behaviour-Driven Development)
- Facilitates collaboration with CFOs, POs, business analysts
- Native support for reading JSON, YAML and CSV files
- Powerful result set assertions in single step

Karate API Tests - Scenario

```
Scenario: Create a cake and retrieve it

Given url 'http://somecakes.ccc/v1/cakes'
And request { type: 'cheese'}
When method POST
Then status 201
And match response == { id: '#notnull', type: 'cheese' }
```

Native JSON syntax

Single line JSON payload assertion

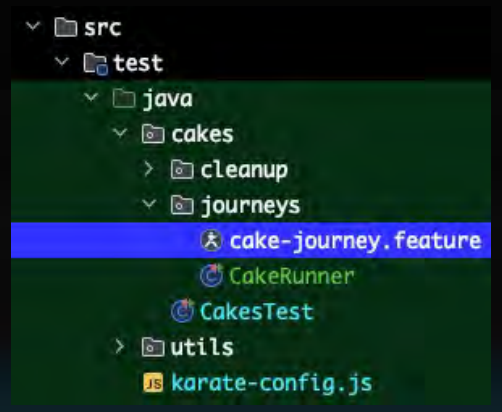
Pre-defined marker for fuzzy matching

Karate API Tests - Scenario

```
Scenario: Create a cake and retrieve it

Given url 'http://somecakes.ccc/v1/cakes'
And request { type: 'cheese'}
When method POST
Then status 201
And match response == { id: '#notnull', type: 'cheese' }

Given path '/v1/cakes', response.id
When method GET
Then status 200
And match response.type == 'cheese'
```

Use response data in subsequent calls

Karate API Tests - Feature File



karate-config.js

```
var config = {
  cakesPath : '/v1/cakes'
};
if (profile == 'dev') {
 config.serviceUrl =
 'http://dev.somecakes.ccc';
}
if (profile == 'local') {
 config.serviceUrl =
 'http://localhost:8091';
}
```

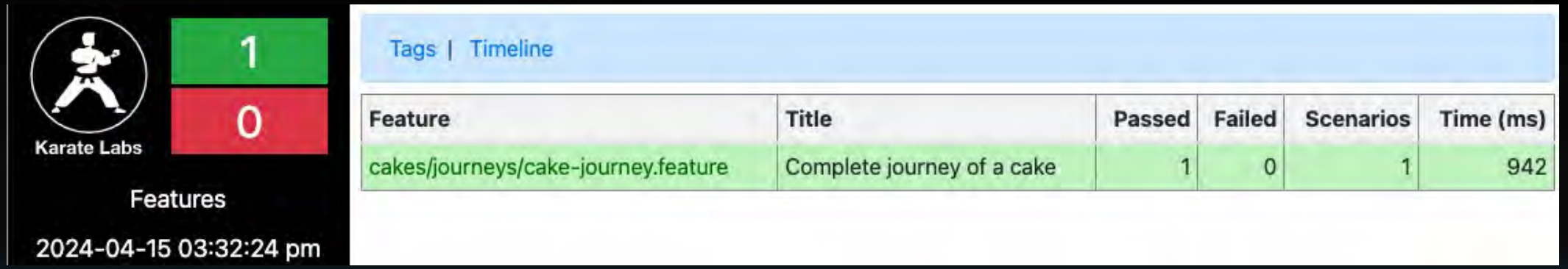
```
Feature: Complete cake journey

 Background:
   * url serviceUrl

   Scenario: Lifecycle of a cake
   Given path cakesPath
   And request { type: 'cheese'}
   When method POST
   Then status 201

   Given path cakesPath, response.id
   When method GET
   Then status 200

   Given path cakesPath, response.id
   When method DELETE
   Then status 200
```

Feature file with multiple scenarios

Preparation step using config file

Variable defined in config file

Karate API Tests - HTML Report

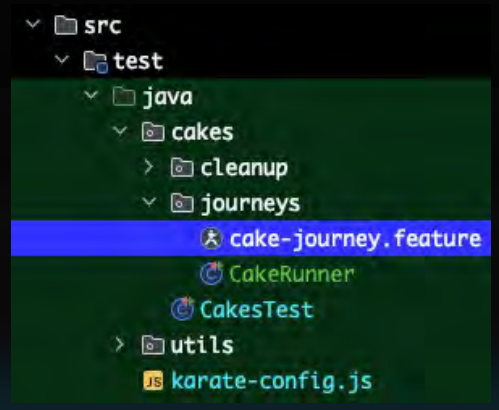


Scenario: [1:7] **Create a cake and retrieve it**		ms: 295
>>	Background:	
9	Given path cakesPath	1
10	And request { type: 'cheese'}	24
11	When method POST	235
12	Then status 201	0
13	And match response == { id: '#notnull', type: 'cheese' }	12
15	Given path cakesPath, response.id	7
16	When method GET	7
17	Then status 200	0
18	And match response.type == 'cheese'	1
20	Given path cakesPath, response.id	0
21	When method DELETE	7
22	Then status 200	0

Provides detailed timeline

Test Execution

Karate API Tests - Test Execution with Java Classes




```java
package utils;
import com.intuit.karate.junit5.Karate;

public abstract class BasicKarateTestSuite {
  protected Karate getKarateConfigurationForCurrentDir() {
    return Karate.run()
            .relativeTo(getClass())
            .reportDir("build/reports/karate-reports");
  }
```

Define test configuration

CakeRunner

```java
public class CakeRunner
 extends BasicKarateTestSuite {

 @Karate.Test
 Karate testAll() {
  return getKarateConfiguration
          ForCurrentDir();
  }
```

CakesTest

```java
package cakes;
class CakesTest extends BasicKarateTestSuite {

 @Test
 void testParallel() {
   Results results = getKarateConfigurationForCurrentDir()
                     .parallel(5);
   assertEquals(0, results.getFailCount);
 }
```

Runs all tests in this package

For JUnit integration

Configure parallel degree

Karate API Tests - Standalone Test Execution

Feature files can also be run using standalone executable:

- Single feature file:

```
java -Dkarate.config.dir=<path> -jar karate.jar cake-journey.feature
```

- All features files in a folder:

```
java -Dkarate.config.dir=<path>  -jar karate.jar cakes/journeys
```

→ More details on standalone JAR

Karate API Tests - Switching Between Configs

File karate-config.js

```
function fn() {
  var profile = karate.env; // Java system property 'karate.env'
  var config = {
    cakesPath    : 'v1/cakes'
  }

  if (profile == 'local') {
    config.baseUrl: 'http://localhost';
  }

  var token = karate.callSingle('get-token.feature', config);
  var globalHeaders = {};
  globalHeaders.Authorization = 'Bearer ' + token.response.id_token;
  karate.configure('headers', globalHeaders);

  return config;
}
```

Invoked before every scenario!

Adapt config based on stage

Globally invoked only once

Returns JSON object

Karate API Tests – Integrate in CI/CD Pipeline

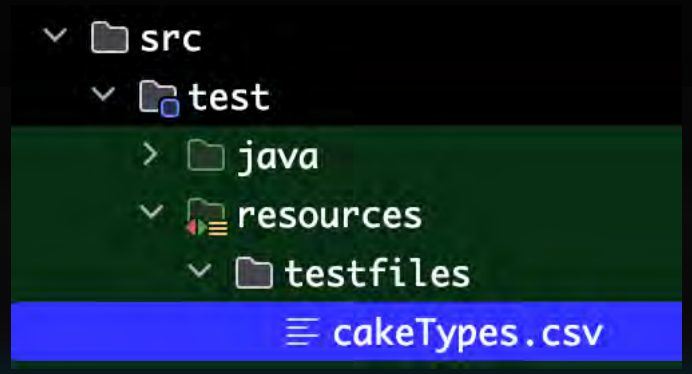
Extensive integration capabilities – examples:

- GitHub Action to run Karate tests
- Add test step as quality gate for Azure DevOps pipelines
- Execute using
  - Maven or Gradle task
  - Standalone JAR
  - Docker
  - NPM

→ Further integrations

Data-Driven Tests

Karate API Tests - Data Driven Testing



cakeTypes.csv

```
type, decoration
cheese,no
cheery,yes
linzer,no
sacher,yes
```

```
Feature: Cake factory - bake lots of cakes
  Background:
    * url  serviceUrl

  Scenario Outline: Create cakes of different types
    Given path cakesPath
    And request { type: <type> }
    When method POST
    Then status 201

    Examples:
      | read('classpath:testfiles/cakeTypes.csv') |
```

Executed for every row in csv

Be aware that the background steps are re-run for every scenario and for every row in the Examples:
→ To execute a background call only once, use ‘callonce’

Karate API Tests - Integrate Further Feature Files

```
  Scenario Outline: Create cakes of different types
    Given path cakesPath
    And request { type: <type> }
    When method POST
    Then status 201

    * def result = karate.call('generate-invoice.feature',
                               { id: response.id } )

    Examples:
      | read('classpath:testfiles/cakeTypes.csv') |
```

Call feature and pass parameters

generate-invoice.feature

```
Feature: Generate invoice for a cake
  Background:
    * url serviceUrl
  Scenario: Call endpoint that generates an invoice

  Given path invoicePath
    And request { soldCake: id}
    When method POST
    Then status 201
```

Karate API Tests - Automatic Processing of Arrays

Call feature file for every object of array:

```
Given path '/v1/cakes'
When method GET
Then status 200

* def result = karate.call('generate-invoice.feature',
                           response.cakes)
```

Response of GET request:

```
{
  "traceId": "1234",
  "cakes": [
      {
        "id": "dlqd9qd",
        "type": "chocolate cake"
      },
      {
        "id": "i2lqd9qd",
        "type": "cheese"
      }
  ]
}
```

Karate API Tests - Generate Test Data

Readily generate JSON test data:

```
Feature: Cake factory - bake lots of cakes
  Background:
    * def generateIdList = function(i)
                           { return { id: 'SomePrefix' + String(i).padStart(7, '0') } }

  Scenario: Create cakes of different types
    * def idList = karate.repeat(1000, (i) => generateIdListPut(i))

    Given path cakesPath, cakeId
    And request idList
    When method PUT
    Then status 200
```

Forms array of results

Karate API Tests - Sophisticated and Powerful Result Checks

Straightforward processing of arrays in result:

```
Given path cakesPath
And request { type: 'cheese'}
When method POST
Then status 201
* def cakeId = response.id

Given path '/v1/cakes'
When method GET
And retry until responseStatus == 200
And assert response.cakes.length >= 1
And match response.cakes[*].id
                      contains cakeId
```

Retry capability

Check size of array

Check field value of array

Response of GET request:

```
{
  "traceId": "1234",
  "cakes": [
      {
        "id": "dlqd9qd",
        "type": "chocolate cake"
      },
      {
        "id": "i2lqd9qd",
        "type": "cheese"
      }
  ]
}
```

Karate API Tests - Sophisticated and Powerful Result Checks

Identity records using built-in JSONPath evaluation:

```
Given path cakesPath
And request { type: 'cheese'}
When method POST
Then status 201
* def cakeId = response.id

Given path '/v1/cakes'
When method GET
Then status 200
* def records = karate.jsonPath(response,
                    "$..cakes[?(@.id=='" + cakeId + "')]")
* match records[0].type == "cheese"
```

Response of GET request:

```
{
  "traceId": "1234",
  "cakes": [
      {
        "id": "dlqd9qd",
        "type": "chocolate cake"
      },
      {
        "id": "i2lqd9qd",
        "type": "cheese"
      }
  ]
}
```

Karate API Tests - Sophisticated and Powerful Result Checks

Verify record that matches RegEx on string “chee” in field ‘type’ in array “cakes“:

```
Given path '/v1/cakes'
When method GET
Then status 200
* match karate.jsonPath(response,
      "$..cakes[?(@.type =~ /.*chee.*/)]")[0].id == cakeId
```

Response of GET request:

```
{
  "traceId": "1234",
  "cakes": [
      {
        "id": "dlqd9qd",
        "type": "chocolate cake"
      },
      {
        "id": "i2lqd9qd",
        "type": "cheese"
      }
  ]
}
```

Gatling Load Tests

Load Tests

Goal:

- Validate behavior of application under **specific expected load**
- Execute tests with a volume **that simulates production workload**
  → Identify potential bottlenecks before new version is deployed to production

Variations:

- Stress testing: Identify breaking points and evaluate the behavior under extreme conditions
  → Check robustness during extreme load
- Spike testing: Suddenly increasing or decreasing the workload
- Capacity testing: Incrementally increase workload to find out maximum workload that can be handled
  → Also known as breakpoint testing
- Soak testing: Make sure the application can sustain a continuous workload
  → Also known as endurance testing

→ Can serve as fitness functions for evolutionary architectures

Gatling Load Tests

- Powerful open-source load testing solution – load test as code
- Designed for continuous load testing: integrates with CI/CD pipelines
- Can re-use Karate tests as load tests
- Use Gatling for defining the load model and everything else in Karate
- **Highly efficient asynchronous actor model to mimic realistic workload**
- **Assertions** on response times, min, max, percentiles → Quality gates
- Detailed performance dashboards

gatling.io

Gatling Load Tests

```
public class CakeFactorySimulation extends Simulation {
  public CakeFactorySimulation() {
    KarateProtocolBuilder protocol =
                              karateProtocol(uri("/v1/cakes/{cakeId}").nil());

    ScenarioBuilder getCakeScenario = scenario("getCake")
        .feed(csv("testfiles/cakeIds.csv").circular())
        .exec(karateFeature("classpath:performance/cake-load-test.feature"));

    setUp(getCakeScenario
            .injectOpen(rampUsersPerSec(3).to(80).during(30),
                        nothingFor(Duration.ofSeconds(5)),
                        constantUsersPerSec(80).during(20))
            .protocols(protocol)
        .assertions(
            global().responseTime().max().lt(100),
            details("GET /v1/cakes").responseTime().percentile3().lt(75),
            global().failedRequests().percent().is(0d));
  }
}
```

Specify feeder of test data

Define feature file of scenario

Set up workload, e.g. ramping up users

Define response time assertions

cakeIds.csv

cakeId
4144-fwdq-4131
0593-fffo-9918
6163-ymfm-6161

That's all ☺

Gatling Load Tests

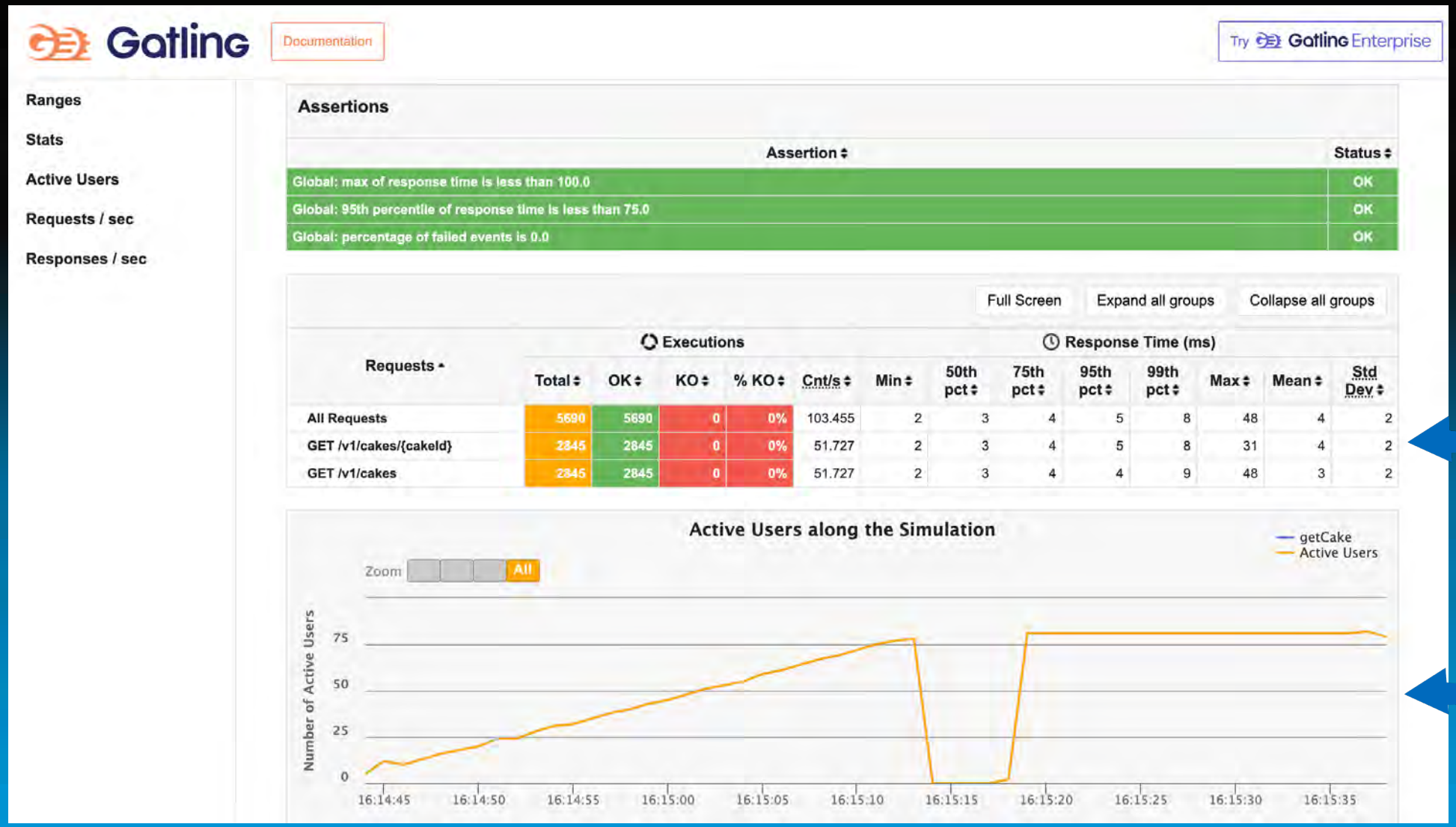
Standard Karate feature files are called by Gatling:

cake-load-test.feature

```
Feature: Load test for querying cakes
  Background:
    * url serviceUrl

    Scenario: Get requested cake and all cakes
      * def cakeId = karate.get('__gatling.cakeId',
                                'defaultId')
      Given path cakesPath, cakeId
      When method GET
      Then status 200
      And match response.type == 'cheese'

      Given path cakesPath
      When method GET
      Then status 200
```

Retrieve values from Gatling feeder

Gatling Performance Report - Automatically Generated



Response time breakdown

Evolution of workload

Summary

source: media.mercedes-benz.com

Karate API tests for real-life verification of microservices

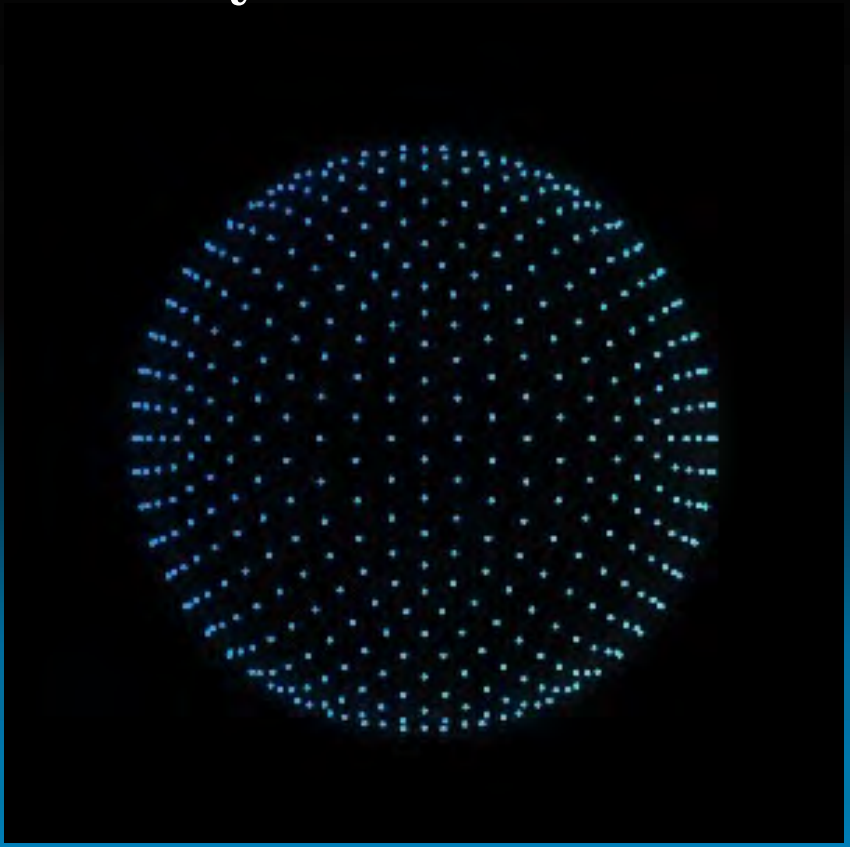
- Tests are easy to write with little efforts
- Covering various test dimensions
- Perform comprehensive checks and assertions
- Inherent parallelism cuts down runtime

Gatling is powerful open-source load testing solution

- Integrates seamlessly with Karate
- Declarative workload definitions and assertions
- *Load test as code* as fitness function

→ Enable automated CI/CD pipelines up to Prod
→ Speed up the delivery of new features with confidence

Thank you



Johannes Schützner
Research & Development
Mercedes-Benz AG

Java Forum Stuttgart 2025