API Performance Test

This guide provides the instructions on how to generate and execute API performance tests for a microservice.

Generating Performance Test Layer

By default, the performance test layer is generated when using a full template. The following custom configurations include the performance test layers in the generated microservice.

> **Note**
>
> Make sure to take the latest dependency version when using custom template configuration.

CRUD Microservice

The following example shows a custom configuration with the performance test layer that is added to the CRUD microservice template:

```yaml
templates:
  crud_performancetest:
    layers:
    - resource
    - persistence
    - service
    - performance_test
    model-generation:
      enable: true
      type: DTO
    key-generation:
      enable: true
    mapper-generation:
      enable: true
dependency-versions:
  ms360sdk: "5.0.4-SNAPSHOT" # The template version
```

Business Microservice Template

The following example shows a custom configuration with the performance test layer that is added to the business microservice template:

```
templates:
  business_performance:
    layers:
    - resource
    - service
    - gateway
    - performance_test
    model-generation:
      enable: true
      type: DM
    key-generation:
      enable: false
    mapper-generation:
      enable: true
dependency-versions:
  ms360sdk: "5.0.4-SNAPSHOT" # The template version
```

Batch Microservice Template

The following example shows a custom configuration with the performance test layer that is added to the batch microservice template:

```
templates:
  batch_performance:
    layers:
    - async
    - gateway
    - service
    - performance_test
    model-generation:
      enable: false
    key-generation:
      enable: false
    mapper-generation:
      enable: false
dependency-versions:
  ms360sdk: "5.0.4-SNAPSHOT" # The template version
```

Adopting Performance Test Layer

To adopt the performance test layer:

1. Update the `pom.xml` file as follows:
    a. Uncomment the executions tag to run your simulation class.
    b. Add additional executions if you add additional simulation classes.
2. Add your request JSON file in the `src/test/resources` path and declare this file in your simulation class in the `RequestBody` variable. If you are simulating a GET request, add an empty JSON file.
3. Add your `EndPoint` and `ScenarioName` to the simulation class.

   Example:

   ```
   private val RequestBody = "addSubscriberExternal_request.json"
   private val Endpoint = "/addSubscriberExternal"
   private val ScenarioName = "Test_Post_AddSubscriberExternal"
   ```

4. Uncomment assertions that you want to test in the simulation class.

   Example:

   ```
   //  global.responseTime.max.lt( 5000 ),
   //  global.successfulRequests.percent.gt( 95 ),
   //  forAll.successfulRequests.percent.is( 100 )
   //  global.responseTime.percentile4.lt( 5000 )
   ```

5. Change the log levels, if necessary, in the `logback-test.xml` file. This file resides in `src/test/resources`.

Executing Performance Tests

To run the performance tests:

1. Run the `mvn clean install` command in the performance test folder.
2. The Microservice CI pipeline is triggered. The pipeline:
    a. Runs the performance test as part of the component test stage.
    b. Generates a Gatling link to view generated performance test report.