Tester des microservices avec des contrats axés sur le consommateur

30 minutes

Découvrez comment tester des microservices Java avec des contrats axés sur le consommateur dans Open Liberty.

Prérequis :

Java SDK 17+ (https://developer.ibm.com/languages/java/semeru-runtimes/downloads)

Git (https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)

Maven (https://maven.apache.org/download.cgi)

Ou, Exécutez dans le cloud (https://openliberty.skillsnetwork.site/courses/course-v1:IBM+GPXX05LMEN+v1/auto_enroll?next=/courses/course-v1:IBM+GPXX05LMEN+v1/courseware/guided_project/guided_project)

Table des matières

Ce que vous apprendrez

Avec une architecture basée sur des microservices, vous avez besoin de tests robustes pour garantir que les microservices qui dépendent les uns des autres sont en mesure de communiquer efficacement. En règle générale, pour éviter plusieurs points de défaillance à différents points d'intégration, une combinaison de tests unitaires, d'intégration et de bout en bout est utilisée. Bien que les tests unitaires soient rapides, ils sont moins fiables car ils s'exécutent de manière isolée et reposent généralement sur des données fictives.

Les tests d'intégration résolvent ce problème en effectuant des tests sur des services réels en cours d'exécution. Cependant, ils ont tendance à être lents car les tests dépendent d'autres microservices et sont moins fiables car ils sont sujets à des modifications externes.

En règle générale, les tests de bout en bout sont plus fiables, car ils vérifient les fonctionnalités du point de vue de l'utilisateur. Cependant, un composant d'interface utilisateur graphique (GUI) est souvent nécessaire pour effectuer des tests de bout en bout, et les composants d'interface utilisateur graphique s'appuient sur des logiciels tiers, tels que Selenium, qui nécessitent beaucoup de temps de calcul et de ressources.

Qu'est-ce qu'un test contractuel ?

Les tests de contrat comblent les lacunes entre ces différentes méthodologies de test. Les tests de contrat sont une technique permettant de tester un point d'intégration en isolant chaque microservice et en vérifiant si les requêtes et réponses HTTP transmises par le microservice sont conformes à une compréhension commune documentée dans un contrat. De cette façon, les tests de contrat garantissent que les microservices peuvent communiquer entre eux.

Pact (https://docs.pact.io/) est un outil de test de contrat open source permettant de tester les requêtes HTTP, les réponses et les intégrations de messages à l'aide de tests de contrat.

Pact Broker (https://docs.pact.io/pact_broker/docker_images) est une application permettant de partager les contrats Pact et les résultats de vérification. Pact Broker est également un élément important pour intégrer Pact dans les pipelines d'intégration continue et de livraison continue (CI/CD).

Les deux microservices avec lesquels vous allez interagir sont appelés `system` et `inventory` . Le `system` microservice renvoie les propriétés système JVM de son hôte. Le `inventory` microservice récupère des propriétés spécifiques du `system` microservice.

Vous apprendrez à utiliser le framework Pact pour écrire des tests de contrat pour le `inventory` microservice qui seront ensuite vérifiés par le `system` microservice.

Prérequis supplémentaires

Avant de commencer, vous devez installer Docker s'il n'est pas déjà installé. Pour obtenir des instructions d'installation, reportez-vous à la **documentation officielle de Docker (https://docs.docker.com/get-docker/)** . Vous déploierez Pact Broker dans un conteneur Docker.

Démarrez votre démon Docker avant de continuer.

Commencer

Le moyen le plus rapide de suivre ce guide est de cloner le **référentiel Git (https://github.com/openliberty/guide-contract-testing.git)** et d'utiliser les projets fournis à l'intérieur :

```
git clone https://github.com/openliberty/guide-contract-testing.git
cd guide-contract-testing
```

Le `start` répertoire contient le projet de départ sur lequel vous allez construire.

Le `finish` répertoire contient le projet terminé que vous allez construire.

Avant de commencer, assurez-vous d'avoir tous les prérequis nécessaires .

Démarrage du Pact Broker

Exécutez la commande suivante pour démarrer Pact Broker :

```
docker-compose -f "pact-broker/docker-compose.yml" up -d --build
```

Lorsque Pact Broker est en cours d'exécution, vous verrez le résultat suivant :

```
Creating pact-broker_postgres_1 ... done
Creating pact-broker_pact-broker_1 ... done
```

Aller à lahttp://localhost:9292/ (http://localhost:9292/)URL pour confirmer que vous pouvez accéder à l'interface utilisateur (UI) du Pact Broker.

Vous pouvez vous référer à la documentation officielle de Pact Broker (https://docs.pact.io/pact_broker/docker_images/pactfoundation) pour plus d'informations sur les composants du fichier Docker Compose.

Implémentation des tests de pacte dans le service d'inventaire

Accédez au `start/inventory` répertoire pour commencer.

Lorsque vous exécutez Open Liberty en **mode dev (https://openliberty.io/docs/latest/development-mode.html)** , le mode dev écoute les modifications de fichiers et recompile et déploie automatiquement vos mises à jour chaque fois que vous enregistrez une nouvelle modification. Exécutez l'objectif suivant pour démarrer Open Liberty en mode dev :

```
mvn liberty:dev
```

Après avoir vu le message suivant, votre instance Liberty est prête en mode développement :

```
************************************************ ************
* Liberty fonctionne en mode développement.
```

Le mode Dev conserve votre session de ligne de commande pour écouter les modifications de fichier. Ouvrez une autre session de ligne de commande pour continuer ou ouvrez le projet dans votre éditeur.

Revenir au `start` répertoire.

Créez le fichier de classe InventoryPactIT.

```
inventory/src/test/java/io/openliberty/guides/inventory/InventoryPactIT.java
```

La `InventoryPactIT` classe contient un `PactProviderRule` fournisseur fictif qui imite les réponses HTTP du `system` microservice. L' `@Pact` annotation prend le nom du microservice comme paramètre, ce qui permet de différencier plus facilement les microservices les uns des autres lorsque vous avez plusieurs applications.

La `createPactServer()` méthode définit les réponses minimales attendues pour un point de terminaison spécifique, appelé interaction. Pour chaque interaction, la requête et la réponse attendues sont enregistrées

auprès du service fictif à l'aide de l' `@PactVerification` annotation.

Le test envoie une requête réelle avec la `getUrl()` méthode du fournisseur de simulation. Le fournisseur de simulation compare la requête réelle avec la requête attendue et confirme si la comparaison est réussie. Enfin, la `assertEquals()` méthode confirme que la réponse est correcte.

Remplacez le fichier de projet Maven d'inventaire.

```
inventory/pom.xml
```

Le framework Pact fournit un `Maven` plug-in qui peut être ajouté à la section build du `pom.xml` fichier. L' `serviceProvider` élément définit l'URL du point de terminaison du `system` microservice et le `pactFileDirectory` répertoire dans lequel vous souhaitez stocker le fichier pact. La `pact-jvm-consumer-junit` dépendance fournit la classe de test de base que vous pouvez utiliser avec JUnit pour créer des tests unitaires.

Après avoir créé la `InventoryPactIT.java` classe et remplacé le `pom.xml` fichier, Open Liberty recharge automatiquement sa configuration.

The contract between the `inventory` and `system` microservices is known as a pact. Each pact is a collection of interactions. In this guide, those interactions are defined in the `InventoryPactIT` class.

Press the `enter/return` key to run the tests and generate the pact file from the command-line session where you started the `inventory` microservice.

When completed, you'll see a similar output to the following example:

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running io.openliberty.guides.inventory.InventoryPactIT
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.631 s - in io.openlil
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
```

When you integrate the Pact framework in a CI/CD build pipeline, you can use the `mvn failsafe:integration-test` goal to generate the pact file. The Maven failsafe plug-in provides a lifecycle phase for running integration tests that run after unit tests. By default, it looks for classes that are suffixed with `IT`, which stands for Integration Test. You can refer to the Maven failsafe plug-in documentation (https://maven.apache.org/surefire/maven-failsafe-plugin/) for more information.

The generated pact file is named `Inventory-System.json` and is located in the `inventory/target/pacts` directory. The pact file contains the defined interactions in JSON format:

```
{
...
"interactions": [
{
      "description": "a request for server name",
      "request": {
        "method": "GET",
        "path": "/system/properties/key/wlp.server.name"
      },
      "response": {
        "status": 200,
        "headers": {
          "Content-Type": "application/json"
        },
        "body": [
          {
            "wlp.server.name": "defaultServer"
          }
        ]
      },
      "providerStates": [
        {
          "name": "wlp.server.name is defaultServer"
        }
      ]
    }
...
  ]
}
```

Now, navigate to the `start/inventory` directory again.

Publish the generated pact file to the Pact Broker by running the following command:

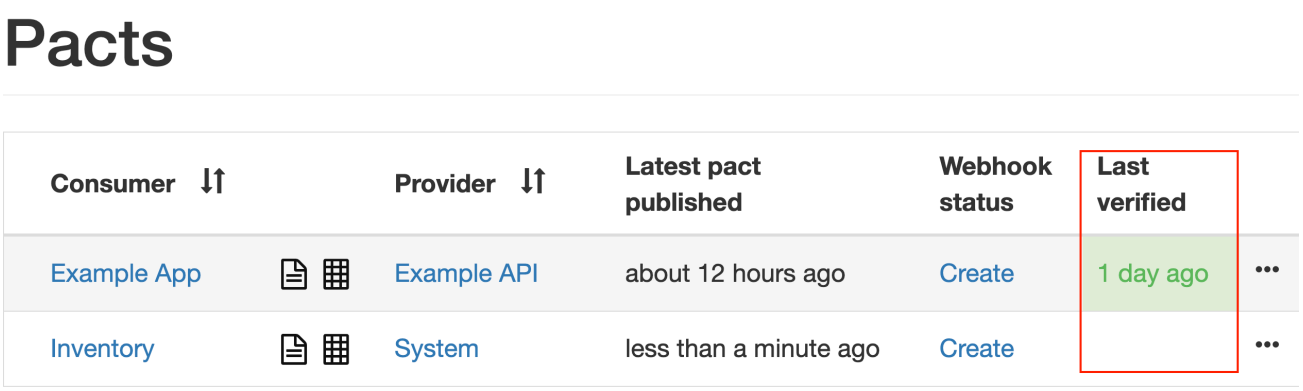
```
mvn pact:publish
```

After the file is published, you'll see a similar output to the following example:

```
--- maven:4.1.21:publish (default-cli) @ inventory ---
Publishing 'Inventory-System.json' with tags 'open-liberty-pact' ... OK
```

Verifying the pact in the Pact Broker

Refresh the Pact Broker webpage at the http://localhost:9292/ (http://localhost:9292/) URL to verify that a new entry exists. The last verified column doesn't show a timestamp because the `system` microservice hasn't verified the pact yet.



You can see detailed insights about each interaction by going to the http://localhost:9292/pacts/provider/System/consumer/Inventory/latest (http://localhost:9292/pacts/provider/System/consumer/Inventory/latest) URL, as shown in the following image:

A pact between Inventory and System

Requests from Inventory to System

- A request for server name given wlp.server.name is defaultServer
- A request for the version given version is 1.1
- A request to check for the default directory given default directory is true
- A request with an invalid property given invalid property

Interactions

Given **wlp.server.name is defaultServer**, upon receiving **a request for server name** from Inventory, with

```
{
  "method": "GET",
  "path": "/system/properties/key/wlp.server.name"
}
```

System will respond with:

```
{
  "status": 200,
  "headers": {
    "Content-Type": "application/json"
  },
  "body": [
    {
      "wlp.server.name": "defaultServer"
    }
  ]
}
```

Implementing pact testing in the system service

Open another command-line session and navigate to the `start/system` directory.

Start Open Liberty in dev mode for the `system` microservice:

```
mvn liberty:dev
```

After you see the following message, your Liberty instance is ready in dev mode:

```
************************************************************
*    Liberty is running in dev mode.
```

Open another command-line session and navigate back to the `start` directory to continue.

Create the SystemBrokerIT class file.

```
system/src/test/java/it/io/openliberty/guides/system/SystemBrokerIT.java
```

The connection information for the Pact Broker is provided with the `@PactBroker` annotation. The dependency also provides a JUnit5 Invocation Context Provider with the `pactVerificationTestTemplate()` method to generate a test for each of the interactions.

The `pact.verifier.publishResults` property is set to `true` so that the results are sent to the Pact Broker after the tests are completed.

The test target is defined in the `PactVerificationContext` context to point to the running endpoint of the `system` microservice.

The `@State` annotation must match the `given()` parameter that was provided in the `inventory` test class so that Pact can identify which test case to run against which endpoint.

Replace the system Maven project file.

```
system/pom.xml
```

The `system` microservice uses the `junit5` pact provider dependency to connect to the Pact Broker and verify the pact file. Ideally, in a CI/CD build pipeline, the `pact.provider.version` element is dynamically set to the build number so that you can identify where a breaking change is introduced.

After you create the `SystemBrokerIT.java` class and replace the `pom.xml` file, Open Liberty automatically reloads its configuration.

Verifying the contract

In the command-line session where you started the `system` microservice, press the `enter/return` key to run the tests to verify the pact file. When you integrate the Pact framework into a CI/CD build pipeline, you can use the `mvn failsafe:integration-test` goal to verify the pact file from the Pact Broker.

The tests fail with the following errors:

```
[ERROR] Failures:
[ERROR]   SystemBrokerIT.pactVerificationTestTemplate:28 Pact between Inventory (1.0-SNAPSHOT
Failures:

1) Verifying a pact between Inventory and System - a request for the version has a matching bo

    1.1) body: $.system.properties.version Expected "1.x" (String) to be a decimal number


[INFO]
[ERROR] Tests run: 4, Failures: 1, Errors: 0, Skipped: 0
```

The test from the `system` microservice fails because the `inventory` microservice was expecting a decimal, `1.1` , for the value of the `system.properties.version` property, but it received a string, `"1.1"` .

Correct the value of the `system.properties.version` property to a decimal.

Replace the SystemResource class file.

```
system/src/main/java/io/openliberty/guides/system/SystemResource.java
```

Press the `enter/return` key to rerun the tests from the command-line session where you started the `system` microservice.

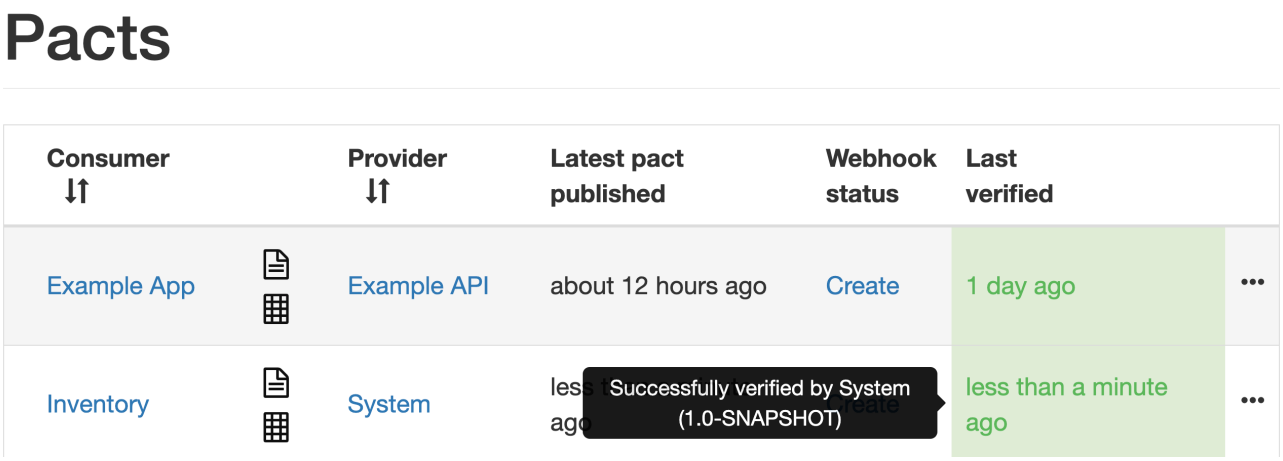
If the tests are successful, you'll see a similar output to the following example:

```
...
Verifying a pact between pact between Inventory (1.0-SNAPSHOT) and System

  Notices:
    1) The pact at http://localhost:9292/pacts/provider/System/consumer/Inventory/pact-versio

  [from Pact Broker http://localhost:9292/pacts/provider/System/consumer/Inventory/pact-versi
  Given version is 1.1
  a request for the version
    returns a response which
      has status code 200 (OK)
      has a matching body (OK)
[main] INFO au.com.dius.pact.provider.DefaultVerificationReporter - Published verification re
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.835 s - in it.io.ope
...
```

After the tests are complete, refresh the Pact Broker webpage at the http://localhost:9292/ (http://localhost:9292/) URL to confirm that the last verified column now shows a timestamp:



The pact file that's created by the `inventory` microservice was successfully verified by the `system` microservice through the Pact Broker. This ensures that responses from the `system` microservice meet the expectations of the `inventory` microservice.

Tearing down the environment

When you are done checking out the service, exit dev mode by pressing `CTRL+C` in the command-line sessions where you ran the Liberty instances for the `system` and `inventory` microservices.

Navigate back to the `/guide-contract-testing` directory and run the following commands to remove the Pact Broker:

```
docker-compose -f "pact-broker/docker-compose.yml" down
docker rmi postgres:16.2
docker rmi pactfoundation/pact-broker:latest
docker volume rm pact-broker_postgres-volume
```

Bon travail ! Et maintenant ?

Nice work! You implemented contract testing in Java microservices by using Pact and verified the contract with the Pact Broker.

Qu'avez-vous pensé de ce guide ?

Qu’est-ce qui pourrait améliorer ce guide ?

Soulever un problème (https://github.com/OpenLiberty/guide-contract-testing/issues) pour partager vos commentaires

Créez une demande d'extraction (https://github.com/OpenLiberty/guide-contract-testing/pulls) pour contribuer à ce guide

Besoin d'aide ?

Poser une question sur Stack Overflow (https://stackoverflow.com/questions/tagged/open-liberty)

Vous aimez Open Liberty ? Ajoutez notre dépôt sur GitHub.

☆ Star

Et ensuite ?

Tester une application MicroProfile ou Jakarta EE

20 minutes (/guides/microshed-testing.html)

Tester les microservices Java réactifs

20 minutes (/guides/reactive-service-testing.html)

Tester les microservices avec le conteneur géré par Arquillian

15 minutes (/guides/arquillian-managed.html)

Learn more about the Pact framework.

Go to the Pact website. (https://pact.io/)