Tutoriel sur les microservices avec JHipster :: Démarrage avec un monolithe

Plan

Recupération des fichiers

```
mkdir -p ~/github/mastering-microservices/
git clone https://github.com/mastering-microservices/tutorial.git
```

Création de l'application de base

```
mkdir -p ~/github/mastering-microservices/online-store
cd  ~/github/mastering-microservices/online-store
jhipster
```

```
? Which *type* of application would you like to create? Monolithic application
(recommended for simple projects)
? What is the base name of your application? store
? What is your default Java package name? com.mycompany.store
? Do you want to use the JHipster Registry to configure, monitor and scale your
application? No
? Which *type* of authentication would you like to use? JWT authentication
(stateless, with a token)
? Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB,
PostgreSQL, Oracle, MSSQL)
? Which *production* database would you like to use? MySQL
? Which *development* database would you like to use? H2 with disk-based
persistence
? Do you want to use the Spring cache abstraction? Yes, with the Ehcache
implementation (local cache, for a single node)
? Do you want to use Hibernate 2nd level cache? Yes
? Would you like to use Maven or Gradle for building the backend? Gradle
? Which other technologies would you like to use?
? Which *Framework* would you like to use for the client? Angular 6
? Would you like to enable *SASS* support using the LibSass stylesheet
```

```
preprocessor? Yes
? Would you like to enable internationalization support? Yes
? Please choose the native language of the application English
? Please choose additional languages to install French
? Besides JUnit and Jest, which testing frameworks would you like to use?
Gatling, Cucumber, Protractor
? Would you like to install other generators from the JHipster Marketplace? Yes
? Which other modules would you like to use? (generator-jhipster-docker-2.5.0)
Additional Docker support: Docker Hub, Local SMTP Server, NGinx, (generator-
jhipster-swagger-cli-3.0.1) JHipster module to generate swagger client code from
a swagger definition
```

> Remarque: Jetez un coup d'oeil aux sous-générateurs disponibles (tous ne sont pas compatibles avec la version courante de JHipster ou bien avec le frontend choisi). https://www.jhipster.tech/modules/marketplace/#/list

Lancez l'application en profil `dev` .

```
./gradlew
```

Loggez vous en utilisateur `admin` `admin` et parcourez les différents sous-menus d'administration (dont l'API Swagger via Swagger UI).

```
open http://localhost:8080
```

Génération des entités de l'application store

Visualisez le schéma du service monolithique avec le JDL Studio. L'image est ici.

Générez les sources (frontend et backend) relatives aux entités et à leurs relations.

```
cd ~/github/mastering-microservices/online-store
jhipster import-jdl ../tutorial/online-store.jh
```

Lancez l'application en profil `dev` .

```
./gradlew
```

Ouvrez l'application dans un browser avec le rafraissement automatique en cas de modification des sources du frontend

```
yarn start
```

Ouvrez l'application dans un browser

```
open http://localhost:8080
```

Loggez vous en utilisateur `admin` `admin` et parcourez l'API Swagger (A)

```
open http://localhost:8080
```

Lancement des tests générés

Pour le backend

```
cd ~/github/mastering-microservices/online-store
./gradlew test
```

Pour le frontend

```
yarn test
```

Pour les tests end-to-end, lancez le backend depuis un terminal

```
cd ~/github/mastering-microservices/online-store
./gradlew
```

Depuis un autre terminal, lancez le test e2e

```
cd ~/github/mastering-microservices/online-store
yarn e2e
```

Analyse de la qualité du code

Lancez un container SonarQube

```
cd ~/github/mastering-microservices/online-store
docker-compose -f src/main/docker/sonar.yml up -d
docker-compose -f src/main/docker/sonar.yml logs -f
^C
```

Attendez qu'il soit démarré et prêt au service.

Lancez l'analyseur SonarQube

```
./gradlew sonarqube
```

Visualisez le rapport de l'analyseur SonarQube

```
open http://localhost:9000
open http://localhost:9000/dashboard?id=com.mycompany.store%3Astore
```

Mise en place du CI/CD

Installez et lancez un serveur Jenkins

```
mkdir -p ~/github/mastering-microservices/jenkins
wget http://ftp-chi.osuosl.org/pub/jenkins/war-stable/2.150.1/jenkins.war
java -jar jenkins.war --httpPort=8989
```

Ouvrez la page de configuration du serveur Jenkins

```
open https://localhost:8989
```

Configurez le serveur Jenkins. Le clé admin se trouve dans la trace de la console de l'application Jenkins. Vous pouvez ajouter des addons comme Blue Ocean, ...

Générez le pipeline Jenkinsfile pour le projet online-store

```
cd  ~/github/mastering-microservices/online-store
jhipster ci-cd
cat Jenkinsfile
```

Poussez le projet `online-store` vers un dépot Git (public ou privé) préalablement créé.

```
GITHUB_USERNAME=moncomptegithub
git remote add origin git@github.com:$GITHUB_USERNAME/online-store.git
git push -u origin master
```

Ajoutez le pipeline `Jenkinsfile` pour le projet `online-store` .

Utilisation de l'API REST avec cURL

Executez les commandes suivantes pour invoquer les opérations de l'API du service.

```
# Installation
sudo apt-get install curl jq
sudo npm install -g jwt-cli

# Content-Type
ACCEPT_JSON="Accept: application/json"
ACCEPT_CSV="Accept: text/csv"
```

```
CONTENT_JSON="Content-Type: application/json"
CONTENT_CSV="Content-Type: text/csv"

# LOCAL
PORT=8080
URL=http://localhost:$PORT

# PROD
#PORT=443
#URL=https://store.mycompany.com:$PORT
#URL=https://microservice-tutorial-store.herukoapp.com:$PORT

# Doc
URL_APIDOC=${URL}/v2/api-docs

# Operations
# CURL="curl --verbose"
CURL="curl -k --verbose"
GET="${CURL} -X GET --header \""$ACCEPT_JSON"\""
POST="${CURL} -X POST --header \""$ACCEPT_JSON"\""
PUT="${CURL} -X PUT --header \""$ACCEPT_JSON"\""
DELETE="${CURL} -X DELETE --header \""$ACCEPT_JSON"\""
OPTIONS="${CURL} -X OPTIONS --header \""$ACCEPT_JSON"\""
HEAD="${CURL} -X HEAD --header \""$ACCEPT_JSON"\""

# ====================================
# Get OpenAPI2.0 specification of the API
# ------------------------------------
${GET} ${URL_APIDOC} > swagger.json

# ====================================
# Authentication operations
# ------------------------------------
USERNAME=user
PASSWORD=user

AUTH_JSON="{ \"username\": \"${USERNAME}\", \"password\": \"${PASSWORD}\" }"

# Get the Bearer token for the user
rm $USERNAME.token.json
${POST}  --header "$CONTENT_JSON" -d "$AUTH_JSON" ${URL}/api/authenticate > $USERNAM
TOKEN=$(jq -r '.id_token' $USERNAME.token.json)
AUTH="Authorization: Bearer $TOKEN"

# Decode JWT Token for fun !
echo "Decode JWT Token $TOKEN"
jwt $TOKEN

# Get account info
${GET} --header "$AUTH" ${URL}/api/account

# ====================================
# Operations on Product resource
# ------------------------------------
```

```
# Get all products
${GET} ${URL}/api/products
# --> 401 Unauthorized
${GET} --header "$AUTH" ${URL}/api/products
# --> 200
${GET} --header "$AUTH" ${URL}/api/products/1

# Add one new product
PRODUCT='{
  "name": "TEST",
  "description": "This is a test",
  "price": 5,
  "size": "XXL",
  "productCategory": {
    "id": 1,
    "name": "TEST",
    "description": "TESTTEST"
  }
}'
${POST} --header "$AUTH" --header "$CONTENT_JSON" ${URL}/api/products -d "$PRODUCT"

# Get the id of the created product
# TODO with jq ".id"
PRODUCT_ID=3

# Get all products
${GET} ${URL}/api/products

# Update one existing product
CHANGE='{
  "price": 1500
}'
${PUT} --header "$AUTH" --header "$CONTENT_JSON" ${URL}/api/products/${PRODUCT_ID} -
# --> 400

# Update one existing product
CHANGE='{
  "name": "NEWTEST"
  "price": 1500,
  "size": "S",
}'
${PUT} --header "$AUTH" --header "$CONTENT_JSON" ${URL}/api/products/${PRODUCT_ID} -
# --> according the PUT meaning, other properties should not be updated by the opera

# Get the updated product
${GET} --header "$AUTH" --header "$CONTENT_JSON" ${URL}/api/products/${PRODUCT_ID}

# Get all products
${GET} --header "$AUTH" --header "$CONTENT_JSON" ${URL}/api/products

# Remove the updated product
${DELETE} --header "$AUTH" --header "$CONTENT_JSON" ${URL}/api/products/${PRODUCT_ID
```

Géneration et utilisation de l'API REST

Le descripteur du service est disponible ici http://localhost:8080/v2/api-docs . Il est généré à partir des annotations des classes `Resource` du paquetage `com.mycompany.store.web.rest` et des classes Entity ou DTO (en fonction de la directive de génération `dto`). Vous pouvez compléter les annotations avec les annotations Swagger pour améliorer l'information de votre API.

Installez `swagger-codegen` (pour plus d'information).

```
mkdir -p ~/github/mastering-microservices/swagger-codegen
cd ~/github/mastering-microservices/swagger-codegen
wget https://oss.sonatype.org/content/repositories/releases/io/swagger/swagger-codeg
mv swagger-codegen-cli-2.2.1.jar swagger-codegen.jar
java -jar swagger-codegen.jar help
```

> Available languages: [android, aspnet5, async-scala, cwiki, csharp, cpprest, dart, flash, python-flask, go, groovy, java, jaxrs, jaxrs-cxf, jaxrs-resteasy, jaxrs-spec, inflector, javascript, javascript-closure-angular, jmeter, nancyfx, nodejs-server, objc, perl, php, python, qt5cpp, ruby, scala, scalatra, silex-PHP, sinatra, rails5, slim, spring, dynamic-html, html, html2, swagger, swagger-yaml, swift, tizen, typescript-angular2, typescript-angular, typescript-node, typescript-fetch, akka-scala, CsharpDotNet2, clojure, haskell, lumen, go-server]

```
codegen() {
  mkdir -p $1
  (cd $1;  java -jar ../swagger-codegen.jar generate -i ../swagger.json -l $1)
}
```

Récupérez la définition Swagger. Vous pouvez installer l'outil `jq` (https://stedolan.github.io/jq/download/) pour formatter le document `swagger.json` .

```
cd ~/github/mastering-microservices/swagger-codegen
wget http://localhost:8080/v2/api-docs -O swagger.json
jq '.' swagger.json
```

> Remarque: il existe un plugin Swagger Codegen pour Maven pour intégrer la génération des documents, des clients et des serveurs dans le build.

Génération de la documentation HTML

```
codegen html2
(cd html2; open index.html)
```

Génération de clients

```
# Client bash basé sur cURL
# codegen bash; (cd bash; tree .)
```

```
codegen typescript-angular; (cd typescript-angular; tree .)

codegen python; (cd python; tree .)

codegen cpprest; (cd cpprest; tree .)

codegen php; (cd php; tree .)
```

Génération de squelettes de serveurs

```
codegen python-flask; (cd python-flask; tree .)

codegen nodejs-server; (cd nodejs-server; tree .)

codegen spring; (cd spring; tree .)

codegen go-server; (cd go-server; tree .)
```

> Remarque: la génération du squelette du serveur Pistache C++ n'est disponible que via

Génération d'un plan de charge pour l'injecteur Apache JMeter

```
codegen jmeter; (cd jmeter; ls -al)
```

> Rien pour l'injecteur de charge Gatling !

Lancement l'application store en mode (ie profil) `prod`

Construisez le WAR de l'application

```
cd ~/github/mastering-microservices/online-store
./gradlew bootWar -Pprod
ls -al build/libs/*.war
```

Depuis un autre terminal, lancez la composition docker-compose qui comporte seulement le gestionnaire de base de données' (`store-mysql`).

```
cd ~/github/mastering-microservices/online-store
docker-compose -f src/main/docker/mysql.yml up -d
```

Depuis un autre terminal, visualisez la console du service

```
cd ~/github/mastering-microservices/online-store
docker-compose -f src/main/docker/mysql.yml logs -f
```

Lancez l'application

```
java -jar build/libs/store-0.0.1-SNAPSHOT.war
```

Finalement, détruisez le gestionnaire de base de données' (`store-mysql`).

```
cd  ~/github/mastering-microservices/online-store
docker-compose -f src/main/docker/mysql.yml down
```

Lancement l'application store en mode (ie profil) `prod` sur Heroku

Cont

```
jhipster heroku
```

Depuis la console Heroku, vérifiez que l'application `tuto-store` est créée.

```
./gradlew bootWar -x test -Pprod
```

> `-x test` n'exécute pas les tests

```
heroku login -i

heroku addons
heroku plugins

heroku plugins:install java

heroku deploy:jar --jar build/libs/store-0.0.1-SNAPSHOT.war --app tuto-store

heroku open  --app tuto-store
open https://tuto-store.herokuapp.com

heroku logs --tail --app tuto-store
# Failed to connect to mysql

heroku logs --tail --app tuto-store
```

Ajoutez le addon MySQL

```
open https://dashboard.heroku.com/apps/tuto-store/settings

heroku addons:open jawsdb  --app tuto-store

heroku config --app tuto-store
heroku config:get JAWSDB_URL --app tuto-store
```

Ajoutez les propriétés du add-on MySQL à configurer dans l'application

```
heroku config:set \
   JDBC_DATABASE_URL=jdbc:mysql://xxxxxxxx.xxxxxxxx.eu-west-1.rds.amazonaws.com:3306
   JDBC_DATABASE_USERNAME=uuuuuuuuuu  \
   JDBC_DATABASE_PASSWORD=pppppppppp  \
 --app tuto-store

#heroku config:unset PORT --app tuto-store

heroku logs --tail --app tuto-store
```

Attendez que le serveur soit prêt

```
2018-12-14T11:42:05.268557+00:00 app[web.1]: ----------------------------------
----------------------
2018-12-14T11:42:05.268559+00:00 app[web.1]: Application 'store' is running!
Access URLs:
2018-12-14T11:42:05.268560+00:00 app[web.1]: Local:
http://localhost:52037
2018-12-14T11:42:05.268560+00:00 app[web.1]: External:  http://172.17.78.18:52037
2018-12-14T11:42:05.268562+00:00 app[web.1]: Profile(s):        [prod, heroku]
2018-12-14T11:42:05.268563+00:00 app[web.1]: ----------------------------------
----------------------
```

Affichez la page du service quand celui ci est lancé.

```
heroku open  --app tuto-store
```

Quelques commandes supplémentaires avec Heroku:

```
heroku apps

heroku apps:info --app tuto-store

heroku ps --app tuto-store

heroku ps:scale web=2 --app tuto-store

heroku releases --app tuto-store
```

```
heroku drains --app tuto-store

heroku maintenance:on --app tuto-store

heroku maintenance:off --app tuto-store

heroku maintenance:on --app tuto-store

heroku stack --app tuto-store

heroku status
```

Lancement l'application store en mode (ie profil) `prod` dans un container

Construisez l'image du conteneur

```
./gradlew bootWar -Pprod buildDocker
docker images | grep store
```

Lancez la composition docker-compose qui comporte l'application (`store-app`) et le gestionnaire de base de données (`store-mysql`).

```
docker-compose -f src/main/docker/app.yml up -d
```

Depuis un autre terminal, visualisez les consoles des 2 services

```
docker-compose -f src/main/docker/app.yml logs -f
```

Arrêtez le service `store-app` de la composition

```
docker-compose -f src/main/docker/app.yml stop store
```

Redémarrez le service `store-app` de la composition

```
docker-compose -f src/main/docker/app.yml start store
```

Arrêtez le service `store-mysql` de la composition

```
docker-compose -f src/main/docker/app.yml stop mysql
```

> Que se passe t'il ?

Redémarrez le service `store-mysql` de la composition

```
docker-compose -f src/main/docker/app.yml start mysql
```

> Que se passe t'il ?

Détruisez la composition.

> Remarque : les données rendues persistances dans les conteneurs sont définitivement perdues.

```
docker-compose -f src/main/docker/app.yml down
```

Lancement de l'injecteur de charge avec Gatling(optionnel)

Suivez la section "Performance tests" de https://www.jhipster.tech/running-tests/

```
cd  ~/github/mastering-microservices/
wget https://repo1.maven.org/maven2/io/gatling/highcharts/gatling-charts-highcharts-
unzip gatling-charts-highcharts-bundle-2.3.1-bundle.zip
```bash

Lancez le recorder depuis un terminal
```bash
GATLING_HOME=~/github/mastering-microservices/gatling-charts-highcharts-bundle-2.3.1
cd  ~/github/mastering-microservices/online-store
cd src/test/gatling
tree .
$GATLING_HOME/bin/recorder.sh

$GATLING_HOME/bin/gatling.sh
```

Lancez l'application à tester (de préférence, la version de production en modifiant auparavant les loggers positionnés en dessous de WARN).

```
cd  ~/github/mastering-microservices/online-store
docker-compose -f src/main/docker/app.yml up -d
docker-compose -f src/main/docker/app.yml logs -f
```

Lancez l'injecteur depuis un autre terminal. Répondez aux questions.

```
GATLING_HOME=~/github/mastering-microservices/gatling-charts-highcharts-bundle-2.3.1
cd  ~/github/mastering-microservices/online-store
cd src/test/gatling
$GATLING_HOME/bin/gatling.sh
```

Ouvrez le rapport HTML qui est généré dans le répertoire `$GATLING_HOME/results` .

Génération de l'application Ionic (optionnel)

Installez Ionic et le sous-générateur pour Ionic Plus d'information.

```
npm install -g ionic
npm install -g generator-jhipster-ionic
```

```
yo jhipster-ionic
```

```
Welcome to the Ionic Module for JHipster! v3.2.0

? What do you want to name your Ionic application? store-ion
? Enter the directory where your JHipster app is located: online-store

Creating Ionic app with command: ionic start store-ion oktadeveloper/jhipster
✔ Preparing directory ./store-ion - done!
✔ Looking up starter - done!
✔ Downloading and extracting oktadeveloper/jhipster starter - done!
? Integrate your new app with Cordova to target native iOS and Android? Yes
> ionic integrations enable cordova --quiet
✔ Downloading integration cordova - done!
✔ Copying integrations files to project - done!
[OK] Integration cordova added!
```

Ajoutez les cartes pour les entités `ProductCategory` et `Product`.

```
cd store-ion
yo jhipster-ionic:entity ProductCategory
yo jhipster-ionic:entity Product
```

Lancez l'application dans un navigateur

```
ionic serve
```