Article
Sylvain Guilbaud · Mars 18, 2022 6m de lecture

Open Exchange

Conteneurisation des passerelles .Net/Java (ou démonstration d'une intégration Kafka)

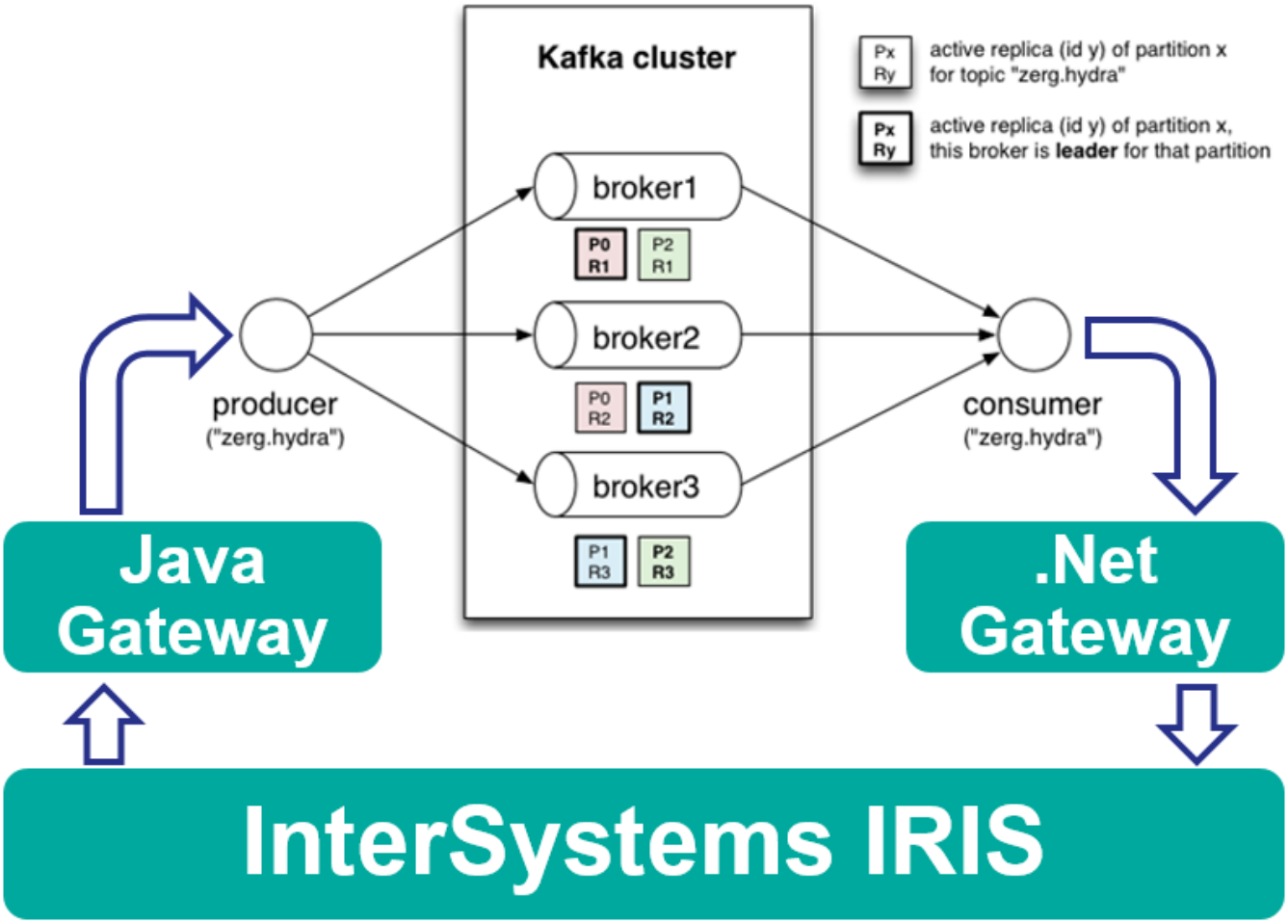
Dans cet article, je vais vous montrer comment vous pouvez facilement conteneuriser les passerelles .Net/Java.

Pour notre exemple, nous allons développer une intégration avec Apache Kafka.

Et pour interopérer avec le code Java/.Net, nous utiliserons PEX.

Architecture

Notre solution fonctionnera entièrement dans docker et ressemblera à ceci :



Passerelle Java

Tout d'abord, nous allons développer l'opération Java pour envoyer des messages dans Kafka. Le code peut être écrit dans l'IDE de votre choix et il peut ressembler à ceci.

En bref :

- Pour développer une nouvelle opération commerciale PEX, nous devons implémenter la classe abstraite `com.intersystems.enslib.pex.BusinessOperation`
- Les propriétés publiques sont les paramètres de l'hôte de l'entreprise
- La méthode `OnInit` est utilisée pour initier la connexion à Kafka et obtenir un pointeur vers InterSystems IRIS
- `OnTearDown` est utilisé pour se déconnecter de Kafka (à l'arrêt du processus)
- `OnMessage` reçoit le message dc.KafkaRequest et l'envoie à Kafka

Maintenant, plaçons-le dans Docker !

Voici notre dockerfile :

```
FROM openjdk:8 AS builder

ARG APP_HOME=/tmp/app

COPY src $APP_HOME/src

COPY --from=intersystemscommunity/jgw:latest /jgw/*.jar $APP_HOME/jgw/

WORKDIR $APP_HOME/jar/
ADD https://repo1.maven.org/maven2/org/apache/kafka/kafka-clients/2.5.0/kafka-
clients-2.5.0.jar .
ADD https://repo1.maven.org/maven2/ch/qos/logback/logback-classic/1.2.3/logback-
classic-1.2.3.jar .
ADD https://repo1.maven.org/maven2/ch/qos/logback/logback-core/1.2.3/logback-
core-1.2.3.jar .
ADD https://repo1.maven.org/maven2/org/slf4j/slf4j-api/1.7.30/slf4j-api-1.7.30.jar .

WORKDIR $APP_HOME/src

RUN javac -classpath $APP_HOME/jar/*:$APP_HOME/jgw/* dc/rmq/KafkaOperation.java && \
    jar -cvf $APP_HOME/jar/KafkaOperation.jar dc/rmq/KafkaOperation.class

FROM intersystemscommunity/jgw:latest

COPY --from=builder /tmp/app/jar/*.jar $GWDIR/
```

Allons-y ligne par ligne et voyons ce qui se passe ici (je suppose que vous connaissez les constructions docker à plusieurs niveaux) :

```
FROM openjdk:8 AS builder
```

Notre image de départ est JDK 8.

```
ARG APP_HOME=/tmp/app
COPY src $APP_HOME/src
```

Nous copions nos sources du dossier `/src` dans le dossier `/tmp/app`.

COPY --from=intersystemscommunity/jgw:latest /jgw/*.jar $APPHOME/jgw/

Nous copions les sources de la passerelle Java dans le dossier `/tmp/app/jgw`.

```
WORKDIR $APP_HOME/jar/
ADD https://repo1.maven.org/maven2/org/apache/kafka/kafka-clients/2.5.0/kafka-clients-2.5.0.jar .
ADD https://repo1.maven.org/maven2/ch/qos/logback/logback-classic/1.2.3/logback-classic-1.2.3.jar .
ADD https://repo1.maven.org/maven2/ch/qos/logback/logback-core/1.2.3/logback-core-1.2.3.jar .
ADD https://repo1.maven.org/maven2/org/slf4j/slf4j-api/1.7.30/slf4j-api-1.7.30.jar .

WORKDIR $APP_HOME/src

RUN javac -classpath $APP_HOME/jar/*:$APP_HOME/jgw/* dc/rmq/KafkaOperation.java && \
    jar -cvf $APP_HOME/jar/KafkaOperation.jar dc/rmq/KafkaOperation.class
```

Maintenant toutes les dépendances sont ajoutées et `javac`/`jar` est appelé pour compiler le fichier jar. Pour un projet concret, il est préférable d'utiliser maven ou gradle.

```
FROM intersystemscommunity/jgw:latest

COPY --from=builder /tmp/app/jar/*.jar $GWDIR/
```

Et enfin, les jars sont copiés dans l'image de base jgw (l'image de base se charge également du démarrage de la passerelle et des tâches connexes).

Passerelle .Net

Vient ensuite le service .Net qui recevra les messages de Kafka. Le code peut être écrit dans l'IDE de votre choix et il peut ressembler à ceci.

En bref :

- Pour développer un nouveau service d'entreprise PEX nous devons implémenter la classe abstraite `InterSystems.EnsLib.PEX.BusinessService`
- Les propriétés publiques sont les paramètres de l'hôte de l'entreprise
- La méthode `OnInit` est utilisée pour établir une connexion avec Kafka, s'abonner à des sujets et obtenir un pointeur vers InterSystems IRIS
- `OnTearDown` est utilisé pour se déconnecter de Kafka (à l'arrêt du processus)
- `OnMessage` consomme les messages de Kafka et envoie des messages `Ens.StringContainer` aux autres hôtes d'Interoperability

Maintenant, plaçons-le dans Docker !

Voici notre dockerfile :

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.1 AS build

ENV ISC_PACKAGE_INSTALLDIR /usr/irissys
```

```
ENV GWLIBDIR lib
ENV ISC_LIBDIR ${ISC_PACKAGE_INSTALLDIR}/dev/dotnet/bin/Core21

WORKDIR /source
COPY --from=store/intersystems/iris-
community:2020.2.0.211.0 $ISC_LIBDIR/*.nupkg $GWLIBDIR/

# copier csproj et restaurer en tant que couches distinctes
COPY *.csproj ./
RUN dotnet restore

# copier et publier l'application et les bibliothèques
COPY . .
RUN dotnet publish -c release -o /app

# étape/image finale
FROM mcr.microsoft.com/dotnet/core/runtime:2.1
WORKDIR /app
COPY --from=build /app ./

# Configurations pour démarrer le serveur passerelle
RUN cp KafkaConsumer.runtimeconfig.json IRISGatewayCore21.runtimeconfig.json && \
    cp KafkaConsumer.deps.json IRISGatewayCore21.deps.json

ENV PORT 55556

CMD dotnet IRISGatewayCore21.dll $PORT 0.0.0.0
```

Allons-y ligne par ligne :

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.1 AS build
```

Nous utilisons le SDK complet .Net Core 2.1 pour construire notre application.

```
ENV ISC_PACKAGE_INSTALLDIR /usr/irissys
ENV GWLIBDIR lib
ENV ISC_LIBDIR ${ISC_PACKAGE_INSTALLDIR}/dev/dotnet/bin/Core21

WORKDIR /source
COPY --from=store/intersystems/iris-
community:2020.2.0.211.0 $ISC_LIBDIR/*.nupkg $GWLIBDIR/
```

Copiez .Net Gateway NuGets de l'image Docker officielle d'InterSystems dans notre image de constructeur

```
# copier csproj et restaurer comme couches distinctes
COPY *.csproj ./
RUN dotnet restore

# copier et publier l'application et les bibliothèques
COPY . .
RUN dotnet publish -c release -o /app
```

Construisez notre bibliothèque.

```
# étape/image finale
FROM mcr.microsoft.com/dotnet/core/runtime:2.1
WORKDIR /app
COPY --from=build /app ./
```

Copiez les dll de la bibliothèque dans le conteneur final que nous exécuterons réellement.

```
# Configurats pour démarrer le serveur Gateway
RUN cp KafkaConsumer.runtimeconfig.json IRISGatewayCore21.runtimeconfig.json && \
    cp KafkaConsumer.deps.json IRISGatewayCore21.deps.json
```

Actuellement, la passerelle .Net doit charger toutes les dépendances au démarrage, nous lui faisons donc connaître toutes les dépendances possibles.

```
ENV PORT 55556

CMD dotnet IRISGatewayCore21.dll $PORT 0.0.0.0
```

Démarrez la passerelle sur le port 55556 en écoutant sur toutes les interfaces.

Et voilà, c'est tout !

Voici un docker-compose complet pour que tout fonctionne (y compris Kafka et Kafka UI pour voir les messages).

Pour exécuter la démo, vous dever :

1. Installer :
    - docker
    - docker-compose
    - git
2. Exécuter :

   git clone https://github.com/intersystems-community/pex-demo.git cd pex-demo docker-compose pull docker-compose up -d

Avis important : Les bibliothèques Java Gateway et .Net Gateway DOIVENT provenir de la même version que le client InterSystems IRIS.

#.NET #Docker #Interopérabilité #Java #Opérations d'entreprise #Service aux entreprises #InterSystems IRIS
Voir l'application sur InterSystems Open Exchange

URL de la source:https://fr.community.intersystems.com/post/conteneurisation-des-passerelles-netjava-ou-d%C3%A9monstration-dune-int%C3%A9gration-kafka