z/TPF publish to data streaming platform driver for Java readme

NOTE: Before using this information and the product it supports, read the general information under "Notices" in this document.

CONTENTS

1.0 Introduction

Apache Kafka is a distributed streaming platform and is designed to publish and consume streams of data such as z/TPF business events. To publish z/TPF business events to a Kafka topic without z/TPF support for Java, you might send z/TPF business events to intermediate systems. The intermediate systems would receive events from the z/TPF system and use native Kafka APIs to publish the events to Kafka topics on a distributed Kafka cluster.

While this model is able to publish z/TPF business events to a Kafka topic, it uses intermediate systems to get the events from z/TPF to Kafka. These intermediate systems are another set of components with their own risks and must be managed in terms of availability, reliability, and scalability. In addition, these intermediate systems must be maintained and kept up-to-date, similar to other systems in your enterprise.

By using the Apache Kafka Java packages on z/TPF, z/TPF can communicate directly with the distributed Kafka cluster without requiring any intermediate systems between z/TPF and the Kafka cluster. A direct-connect solution between z/TPF and the Kafka cluster means you can skip the intermediate systems, resulting in a more reliable solution with fewer components to manage and maintain.

The z/TPF publish to data streaming platform driver for Java demonstrates one example of how you can use the Apache Kafka Java package to publish z/TPF business events directly from your z/TPF system to a Kafka topic. The z/TPF publish to data streaming driver for Java provides a working example that creates signal events (a type of z/TPF business event), formats the event as a JSON document, and calls a Java service running on z/TPF to publish the JSON-formatted event to a Kafka topic on a distributed Kafka cluster. This document describes how to install, build, and run this driver on your z/TPF system.

For more information about creating Java application services and calling those services from your z/TPF applications, see the z/TPF product documentation in IBM Knowledge Center (https://www.ibm.com/support/knowledgecenter/SSB23S).

For more information on Apache Kafka, see the Apache Kafka website (http://kafka.apache.org/).

1.1 Driver components and architecture

The z/TPF publish to data streaming platform driver for Java contains the following core components, which represent a traditional z/TPF application, business event dispatch processing, and a publishing service:

- The QKFK driver represents a traditional z/TPF C/C++ application that is creating business events. In this case, the driver creates random flight status information (airline code, flight number, origin, destination, status, etc.) and uses the tpf_bev_signal() API to send the flight status as a QKFK_signal signal event. Depending on the driver mode, the QKFK driver displays the flight status information (Visual mode) or creates signal events in a tight loop (Load mode).

- The QKFK_signal event is defined by the business event specification and associated artifacts (business event dispatch adapter, DFDL schema files, and event custom programs). The business event dispatch adapter uses an event custom format program (QKFF) and a custom adapter program (QKFT) to format and transmit the signal event. The QKFF program formats the event as a JSON document, sets the topic name to "flightStatus", and creates a key using the airline code and flight number from the signal event data. The QKFT program uses the tpf_srvcInvoke() API call the KafkaPublish service to publish the JSON-formatted event to the Kafka topic.

While this event defines and processes a flight status signal event created by the QKFK driver, it could represent any signal event or data event created by your z/TPF system.

- The KafkaPublish service is a REST service that is written in Java and uses open source Apache Kafka Producer APIs to publish messages to topics in a Kafka cluster. For the QKFK_signal signal event, the KafkaPublish uses the topic and key created by the QKFF program to publish the JSON-formatted signal event to the Kafka topic. The KafkaPublish service is packaged in the KafkaApp service application and is deployed as part of the kafka JAM.

  Even though this driver calls the KafkaPublish service from business event dispatch processing, any z/TPF application program could use the tpf_srvcInvoke() API to call the KafkaPublish service and publish messages to Kafka topics.

2.0 Change history

2018Oct15 Initial version
2023Oct12 Incorporate Maven tooling enhancements

3.0 Prerequisites

The following list provides the required release levels:

- z/TPF (PUT 14 or later) with z/TPF support for Java (APAR PJ43892) installed. For more information about installing, building, and configuring z/TPF support for Java, see the z/TPF product documentation in IBM Knowledge Center (https://www.ibm.com/support/knowledgecenter/SSB23S).

- Business events must be configured and enabled on your z/TPF system. For For more information about configuring and enabling business event processing, see the z/TPF product documentation in IBM Knowledge Center (https://www.ibm.com/support/knowledgecenter/SSB23S).

The following build tools are required:

- maketpf utility

4.0 Installing the driver

1) Use FTP to transfer the tar file (QKFK.tar.gz) to your Linux on IBM Z build system. This file can be placed in any directory as a holding location, for example,

```
     /tmp/ztpftar
```

2) Create a root directory to hold the unpacked files, for example, /ztpfdrvs

3) Extract the source code from the tar file by entering the following commands:

```
     cd /ztpfdrvs
     tar -xvzf /tmp/ztpftar/QKFK.tar.gz
```

   The project source files are extracted in the following directory structure:

```
     qkfk/kafka.pom.xml
     qkfk/qkf2.cpp
     qkfk/qkf2.mak
     qkfk/qkff.cpp
     qkfk/qkff.mak
     qkfk/qkfj.mak
     qkfk/qkfk.cntl
     qkfk/qkfk.cpp
     qkfk/qkfk.loadfile
     qkfk/qkfk.mak
     qkfk/qkft.cpp
     qkfk/qkft.mak
     qkfk/qkfk_drv.h
     qkfk/qkfk_flightStatus.h
     qkfk/qkfk_functions.cpp
     qkfk/qkfk_kafka.h
     qkfk/fdes/kafka.jam.xml
     qkfk/fdes/KafkaPublish.srvc.json
     qkfk/fdes/KafkaRequest.gen.dfdl.xsd
     qkfk/fdes/kafkaServices.swagger.json
     qkfk/fdes/QKFK_Dispatch.evda.xml
     qkfk/fdes/QKFK_flightStatus.user.dfdl.xsd
     qkfk/fdes/QKFK_signal.se.dfdl.xsd
     qkfk/fdes/QKFK_signal.se.evspec.xml
     qkfk/maven/dependencies.txt
     qkfk/maven/duplicates.txt
     qkfk/producers/flightStatus.properties
     qkfk/src/main/java/com/kafka/KProducer.java
     qkfk/src/main/java/com/kafka/app/KafkaApp.java
     qkfk/src/main/java/com/kafka/models/KafkaRequest.java
     qkfk/src/main/java/com/kafka/rest/KafkaHandler.java
```

4) Create a maketpf.cfg file with the following contents:

```
     APPL_ROOT := /ztpfdrvs
     TPF_ROOT := /ztpf
     LOADTPF_IP:=ftp://<user>@<host>
     TPF_BSS_NAME := BSS
     #TPF_SS_NAME :=
     #USER_VERSION_CODE :=
```

   a) Set APPL_ROOT to the directory that contains the driver source code that was extracted.
   b) Set TPF_ROOT to the directory that contains the z/TPF source code.
   c) Set LOADTPF_IP to the correct user/host of your z/TPF system.
   d) Set TPF_BSS_NAME to the basic subsystem name of your z/TPF system. By default, this value is set to BSS.
   e) Optional: Set TPF_SS_NAME to the subsystem name.
   f) Optional: Set USER_VERSION_CODE to any 2-character string. The 2-character string that you set is appended to the shared objects that are built. By default, this value is set to null.

   For details about these variables, enter man maketpf.cfg on your Linux on Z build system.

5) Build the USRSTUB program and online program attribute table (IPAT) after you add the QKFK driver control file to your user control file.

   a) Add the following line to your user control file (base/cntl/usr.cntl):

```
       include qkfk/qkfk.cntl
```

   b) Build the USRSTUB program to generate stubs for all user programs using the following command:

```
        maketpf USRSTUB -f
```

   c) Rebuild IPAT to incorporate the changes you made in the usr.cntl file:

```
        maketpf ipat -f
```

   d) Load the IPAT that was built in step 5c to your z/TPF system.

6) If Apache Maven on your Linux on IBM Z build system is configured to use a local repository, verify that all dependency files required by this driver are installed in the local repository and download any missing dependencies. For a list of dependencies required by this driver, see /ztpfdrvs/qkfk/maven/dependencies.txt.

7) Run the maketpf utility with the accompanied control file (qkfk.cntl) to

assemble, compile, and link the driver programs:

```
      bldtpf /ztpfdrvs/qkfk/qkfk.cntl
```

8) Modify the producer properties file for the flightStatus topic. When publishing to the flightStatus topic, the Kafka producer uses properties defined in the qkfk/producers/flightStatus.properties properties file. Change the bootstrap.servers property to list the IP addresses and port numbers for your kafka brokers. Instructions for setting up the Kafka brokers are in Section 6.

9) Use the standard load procedure to transfer and load the driver shared objects, jar files (Java programs), and common deployment files that are required for the QKFK driver to the z/TPF system:

```
      loadtpf -s qkfkload /ztpfdrvs/qkfk/qkfk.cntl  /ztpfdrvs/qkfk/qkfk.loadfile
```

10) Use the standard procedure to activate these loadsets on the z/TPF system.

11) Update the test driver program to start the QKFK driver.

   a) Update base/rt/cvzz.asm (or the tool that runs driver programs) to make an entry for the QKFK driver. The QKFK shared object is the main entry point for the QKFK driver.

   b) Build and load the updated CVZZ program to the z/TPF system.

For more information about program management, including how to build and load programs to the z/TPF system, see the z/TPF product documentation in IBM Knowledge Center (https://www.ibm.com/support/knowledgecenter/SSB23S).

5.0 Configuring the z/TPF system

To deploy the DFDL schemas, service descriptors, openAPI (swagger) documents, and the JAM descriptor, enter the ZTEST QKFK command with INIT parameter specified. This command calls the ZMDES DEPLOY command for all common deployment files that are required for this driver. For example:

```
User:   ZTEST QKFK INIT

System: QKFK0003I 10.27.14 STARTING INIT REQUEST
        CSMP0099I 10.27.14 000000-B ZMDES DEPLOY FILE-KAFKA.JAM.XML
        MDES0008I 14.58.49 DEPLOY IS COMPLETE ON PROCESSOR B FOR
        FILE-/sys/tpf_pbfiles/tpf-fdes/kafka.jam.xml
        ...
        QKFK0004I 10.27.14 INIT REQUEST COMPLETE
```

Note: An CSMP0099I and MDES0008I messages will be received for each common deployment file that is in the qkfk/fdes source directory.

6.0 Setting up the Kafka environment

1) Download and install the Apache Kafka open source software from kafka.apache.org onto a system of your choice (for example, Linux or Windows).

2) Start a zookeeper server on your remote system. Kakfa provides a script, zookeeper-server-start.sh, that you can use to start a zookeeper server.

3) Start and configure your Kafka broker

    a) After starting the zookeeper server, you can start a Kafka brokers on your remote system. Kafka provides a script, kafka-server-start.sh, that you can use to start a Kafka broker.

    b) Create the flightStatus topic in your Kafka environment. Kafka provides a script, kafka-topics.sh, in the /kafka/bin directory that you can use to create a topic.

4) Start a consumer for the flightStatus topic on your remote system. Kafka provides a script, kafka-console-consumer.sh, that you can use to start a consumer for the flightStatus topic.

7.0 Starting the kafka JAM

Enter the ZJAMC START command to start the pricing server in the JAM. For example:

```
    User:   ZJAMC START N-kafka

    System: JAMC0134I 14.28.39 THE JVM FOR JAM kafka IS STARTED, PID 1092157451.
            JAMC0002I 14.28.39 JAM kafka IS ACTIVE.
```

8.0 Running the QKFK driver

Before you start the driver, check the following conditions:

o Ensure that the kafka JAM is running by entering the ZJAMC DISPLAY command. For example:

```
    User:   ZJAMC DISPLAY N-kafka

    System: JAMC0007I 15.25.25 START OF ZJAMC DETAILED DISPLAY
            JAM NAME          #JVMS #THDS SHARED CLASS CACHE        STATE
            kafka                 1     4 N/A                       ACTIVE

            DEPLOYMENT DESCRIPTOR FILE NAME
            /sys/tpf_pbfiles/tpf-fdes/kafka.jam.xml

            JVM PID      JVM STATE       JVM LOG FILE DIRECTORY
            1946746928   ACTIVE          /tpfjam/kafka/1946746928
            END OF DISPLAY+
```

o Ensure that the QKFK_signal signal event is deployed and signal events are enabled by entering the ZBEVF DISPLAY EVENT command. For example:

```
    User:   ZBEVF DISPLAY EVENT QKFK_signal

    System: BEVF0046I 15.28.11 DISPLAY OF EVENT MESSAGE SIGNAL EVENT QKFK_signal IN
            FILE-/sys/tpf_pbfiles/tpf-fdes/QKFK_signal.se.evspec.xml
            STATUS:                 SIGNAL EVENTS ENABLED ON PROCESSOR B
            DEPLOYED:               YES _
            EVENT NAME:             QKFK_signal
            APPLICTN ENRICHMNT PGM:
            DISPATCH ENRICHMNT PGM:
            PERSISTENCE:            NO
            PRIORITY:               9
            EXPIRY TIME:            -1
            DISPATCH QUEUE NAME:    IBEV.UNORDERED.DISPATCH.QUEUE
            ERROR QUEUE NAME:
            EVENT ERROR PROGRAM:
            DISPATCH ADAPTER:       QKFK_Dispatch
            EVENT MSG FORMAT FILE:  QKFK_signal.se.dfdl.xsd
            END OF DISPLAY+
```

1) Start the driver in one of the following ways:

   o To start the driver in visual mode, enter ZTEST QKFK START. Every few seconds a flight status signal event is generated and displayed on the console. For example:

```
      User:   ZTEST QKFK START

      System: QKFK0001I 10.29.49 THE QKFK DRIVER IS STARTED IN VISUAL MODE
```

      Example console output when running in visual mode:

```
        System: QKFK0011I 14.45.58 EVENT DATA FOR THE QKFK_signal SIGNAL EVENT
```

```
----------------------------------------------------------------------------
                  Flight:    NW4695 from EWR to AVP
                  Departure: 2018-09-26 at 21:52, Gate A41
                  Arrival:   2018-09-27 at 03:27, Gate C33
                  Status:    Gate Closed
                  Comment:   0 Bytes

----------------------------------------------------------------------------
```

      Example message received and displayed by the Kafka consumer (formatted for readability):

```
        Consumer: {"Event":
                    {
                      "EventHeader":{
                        "size":77,
                        "structID":"C5C8",
                        "version":1,
                        "ECBCtxFlag":0,
                        "UsrCtxFlag":0,
                        "eventName":"QKFK_signal",
                        "eventType":1,
                        "ssuName":"HPN",
                        "eventTime":"2018-09-24T18:45:58.655",
                        "fractionalMicSec":70,
                        "interceptName":"QKF2 Generated Event"
                      },
                      "EventData":{
                        "airline":"NW",
                        "flight_number":4695,
                        "orig":"EWR",
                        "dest":"AVP",
                        "departGate":"A41",
                        "arriveGate":"C33",
                        "departure_date":"2018-09-26 at 21:52",
                        "arrival_date":"2018-09-27 at 03:27",
                        "status":"Gate Closed",
                        "commentSize":0
                      }
                    }
                  }
```

   o To start the driver in load test mode, enter ZTEST QKFK START MODE-1. In load test mode, signal event information is not displayed and requests to signal events are created in a tight loop. For example:

```
      User:   ZTEST QKFK START MODE-1

      System: QKFK0001I 10.30.55 THE QKFK DRIVER IS STARTED IN LOAD MODE
```

2) To display a summary of the driver status, enter ZTEST QKFK STATUS. For example:

```
      User:   ZTEST QKFK STATUS

      System: QKFK0010I 15.24.00 QKFK DRIVER STATUS
              ***************************************************************
              IS#   ECB#          SUCCESS/FAIL           RATE
              ---------------------------------------------------------------
               1      0              1498/0            10/sec
              ---------------------------------------------------------------
                1 ECB(s)             1498/0            10/sec
              ---------------------------------------------------------------
              Throttle:     100000 us                          Mode:    LOAD
              Time Running: 00:02:31                   Comment Size:       0
              ***************************************************************
              END OF DISPLAY
```

3) To stop the driver, enter ZTEST QKFK STOP. For example:

```
      User:   ZTEST QKFK STOP

      System: QKFK0007I 10.30.08 STOPPING THE QKFK DRIVER
              QKFK0008I 10.30.08 THE QKFK DRIVER IS STOPPED
```

For information about additional parameters for the QKFK driver command, including how to run multiple QKFK driver ECBs, how to add a delay between service requests, and how to specify a comment size to alter the size of the signal event, enter ZTEST QKFK HELP.

9.0 Optional adjustments

1) By default, the kafka JAM is configured to start one JVM with four application threads and is able to run with a maximum of 256 1-MB frames for 64-bit heap; that is, a value of 256 for the MAXXMMES parameter in keypoint A. These are minimal settings that demonstrate how your z/TPF system can publish messages to a Kafka cluster. The following settings can be changed to provide improved application performance and scalability.

   If you make these changes, ensure that there are enough 1 MB frames allocated to accommodate each JVM that uses the amount of 64-bit ECB heap defined by the MAXXMMES parameter.

   o Optional: To run the kafka JAM with more JVMs or application threads, change the value of the <NumberJVMs> or <NumberThreadsPerJVM> elements in the /ztpfdrvs/qkfk/fdes/kafka.jam.xml file and load the updated kafka JAM descriptor to the z/TPF system.

   o Optional: To provide more ECB heap to the JVMs, increase the MAXXMMES parameter value in keypoint A to 600 MB or greater.

2) The KafkaApp in the kafka JAM creates a Kafka producer by using the properties defined in the qkfk/producers/flightStatus.properties properties file. Kafka provides several properties to tune the Kafka producer based on your workload, like buffer sizes and linger time.

If you alter any properties in the properties file, load the updated properties file to your z/TPF system, and stop and start or recycle the kafka JAM. For more information on producer properties, see the Apache Kafka documentation (http://kafka.apache.org).

3) Section 6 describes how to start a single Kafka broker. For reliability and scalability, multiple Kafka brokers can be started on separate distributed systems.

If you want to add more Kafka brokers to your environment, create a new server.properties file in the /kafka/config directory on your distributed system. In the new properties file, define the new Kafka broker with a new broker ID, a unique IP address and port combination, and a different log.dirs directory. You also can edit other server properties such as log retention time and the default number of partitions per topic.
After the updates are complete, you can use the script provided with Kafka, kafka-server-start.sh, to start another Kafka broker.

10.0 Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which

has
been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including
in some cases, payment of a fee.

10.1 Trademarks

10.2 Warranty

This package is provided on an "as is" basis. There are no warranties, express or
implied, including the implied warranties of merchantability and fitness for a particular
purpose. IBM has no obligation to provide service, defect correction, or any maintenance
for the package. IBM has no obligation to supply any updates or enhancements for the
package to you even if such are or later become available.