IBM Event Streams
Kafka Connect for IBM MQ 構成ガイド

第1.0版（2019.07）

日本アイ・ビー・エム システムズ・エンジニアリング

はじめに

- 本書は、IBM Event Streamsが提供するKafka Connector for IBM MQについて記載しています。
- 前提とする、IBM Event Streamsの基本については下記資料をご参照ください。
- SIL「IBM Event Streams on ICP 導入・構成ガイド」
  - https://ibm.ent.box.com/s/ywgonge2xe626r9wiry2emoy5k9psc7g

内容

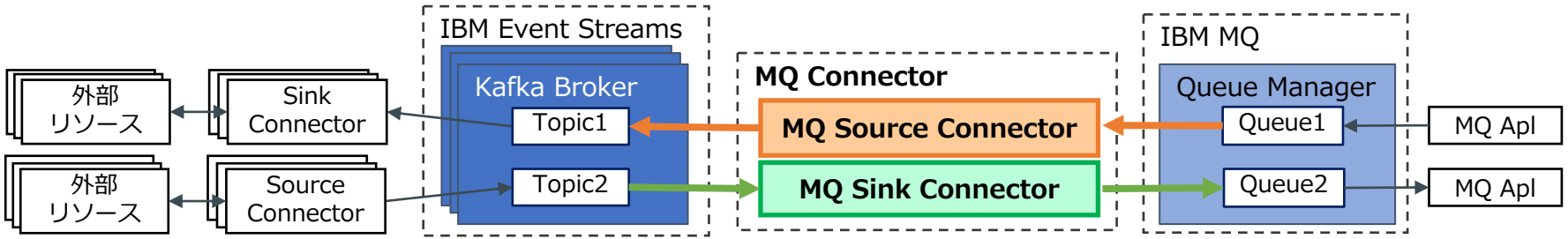
- 概説
  - Kafka Connect for IBM MQ
  - MQ Connectorの稼働形態
  - MQ Connectorの基本動作
- 構成手順
  - 構成手順
  - ①MQの構成
  - ②IESの構成
  - ③Kafka Connectの構成
  - ④MQ Connectorの構成、起動
- 参考リンク

概説

Kafka Connect for IBM MQ

- IBM Event Streamsでは、**KafkaとMQを連携する2つのConnector**を提供
  - **Kafka Connect Source Connector for IBM MQ （MQ Source Connector）**
    - MQのキューからIESのトピックにデータを転送
  - **Kafka Connect Sink Connector for IBM MQ （MQ Sink Connector）**
    - IESのトピックからMQのキューにデータを転送
  - Connectorは、IESだけでなく他のKafkaブローカーに対しても利用できる

- **IES/Kafkaを介して、様々なリソースとMQを連携することができる**
  - Kafka Connectとは、Kafkaと外部リソースを連携するためのフレームワーク
    - Source Connectorは外部リソースからKafkaへ、Sink ConnectorはKafkaから外部リソースへの連携を行う
    - ベンダー/コミュニティから様々なConnectorが提供されている
  - 他のConnectorと組み合わせることで様々なリソースとMQを連携することが可能



Kafka Connect for IBM MQ

- MQ Connectorは、JARの形態で提供
  - IESの管理画面から入手
    - サンプルのプロパティ・ファイルも付属

Connector	Connector JAR & property file
MQ Source Connector	・kafka-connect-mq-source-<version>-jar-with-dependencies.jar（最新のversionは、1.0.1） ・mq-source.properties
MQ Sink Connector	・kafka-connect-mq-sink-<version>-jar-with-dependencies.jar（最新のversionは、1.0.1） ・mq-sink.properties

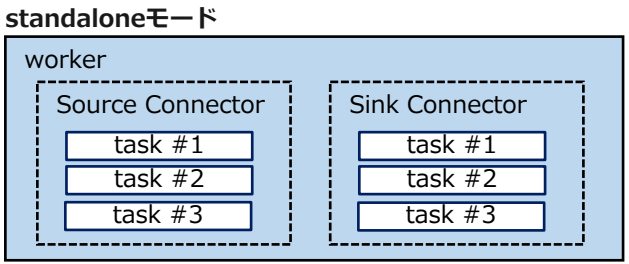
    - GitHubからも入手可能
      - ✓ MQ Source Connector : https://github.com/ibm-messaging/kafka-connect-mq-source
      - ✓ MQ Sink Connector : https://github.com/ibm-messaging/kafka-connect-mq-sink

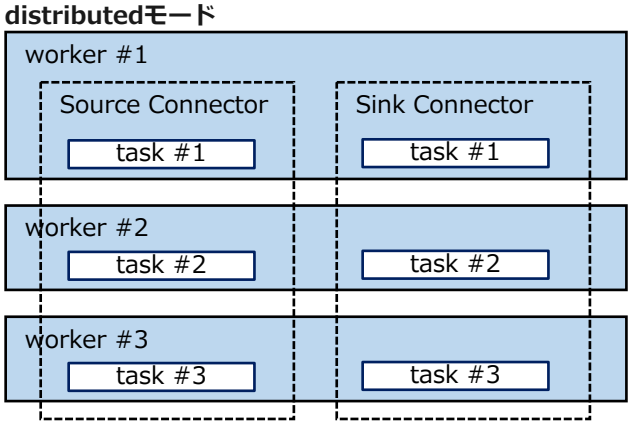
- 前提
  - IES or Kafka 2.0.0 以降
  - MQ V8.0 以降 or MQ on Cloud

MQ Connectorの稼働形態

- MQ Connectorは、Kafka Connectのワーカー・プロセス内で稼働
  - ワーカーはJavaプロセス
  - MQ Connectorはワーカー内のタスク（スレッド）として稼働
    - 複数のタスクで並列稼働させることも可
  - ワーカー内には、1つ or 複数のConnectorを稼働させることができる

- Kafka Connectワーカーの稼働モード
  - **standaloneモード**
    - 1つのワーカーが単体で稼働
  - **distributedモード**
    - 複数のワーカーでクラスタを構成して稼働
    - タスクは複数のワーカーに分散して稼働

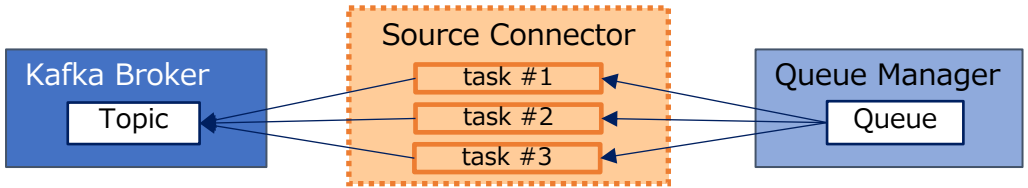




MQ Connectorの基本動作

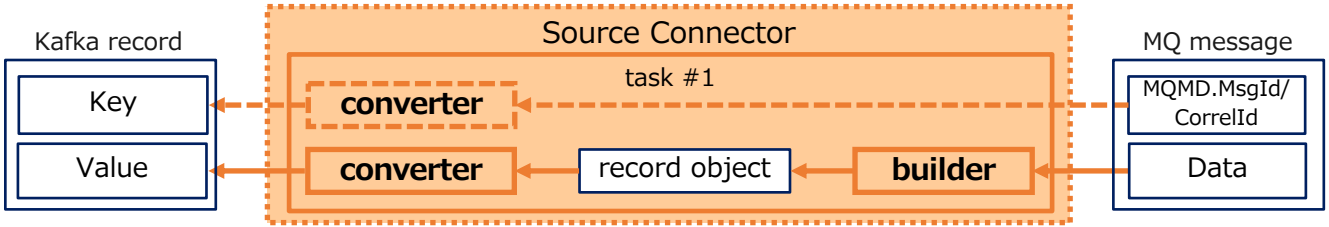
■ MQ Source Connectorの動き

- ◆ MQのキューからメッセージを受信し、受信データをKafkaのトピックにレコードとして送信
  - **MQに対して常駐型の受信アプリケーション**として稼働
    - ✓ キューマネージャーに対して、ローカル接続、クライアント接続とも可能
      - CCDT利用やTLS通信も可能
    - ✓ 複数のタスクで稼働する場合は、それぞれがキューマネージャーに接続し、メッセージを受信
      - 1つのキューに対し、複数のタスクで分散処理
      - タスク数を増やすことで並列度を上げることができる
    - ✓ JMSインターフェースでアクセス
      - JMSフォーマットではない通常のMQメッセージ（MQRFH2ヘッダーなし）も受信可能
      - セレクターの指定は不可
    - ✓ ストリングデータの場合、受信時に非UnicodeからUnicodeへコード変換することも可能
      - コード変換せずにバイナリのまま受信し、Kafkaレコードに出力することも可
  - **Kafkaに対してProducer**として稼働
    - ✓ Key値を指定する／しないを選択可（後述）
      - Key値を指定しない場合はトピックを構成する複数のパーティションに分散して送信
      - Key値を指定した場合は、Key値に基づいて選出されたパーティションに送信
      - 同一Key値を持つメッセージは同じパーティションに送信される



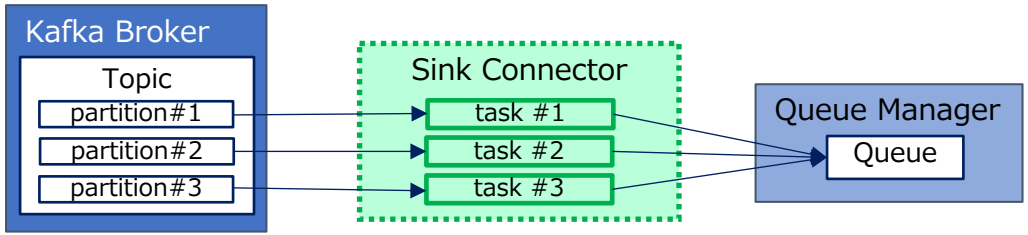
MQ Connectorの基本動作

- MQメッセージのデータ部をKafkaレコードのValueにマッピング
  - 内部では、 **builder**がデータ部をJavaオブジェクトに変換し、**converter**がValueに変換（Serialize）
    - ✓ ユーザーは、構成時にデータ部/Valueのデータ・フォーマットに応じて、適切なbuilder、converterを選択する（後述）
    - ✓ MQメッセージのフォーマットや選択するbuilderによって、Javaオブジェクトのタイプが決まる
    - ✓ Javaオブジェクトのタイプに合わせて、converterを選択する
  - MQMDのMsgId/CorrelIdをKafkaレコードのKeyにマッピングすることも可（オプション）
    - ✓ MessageID/CorrelationIDのフォーマットはバイナリ固定のため、builderの指定は不要
    - ✓ converterのみKeyのデータ・フォーマットに応じて選択



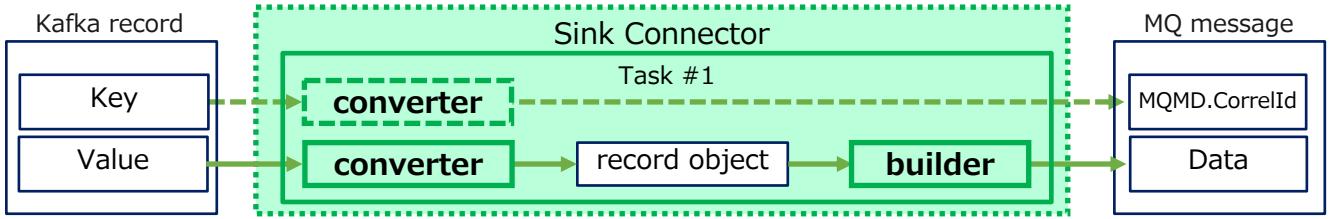
MQ Connectorの基本動作

- **MQ Sink Connectorの動き**
  - Kafkaのトピックからレコードを受信し、MQのキューにメッセージを送信
    - **Kafkaに対してConsumer**として稼働
      - ✓ 複数のタスクで稼働する場合、コンシューマー・グループIDは同一のため各タスクにパーティションが割り振られる
      - ✓ トピックのパーティション数に合わせてタスクを起動することで並列処理が可能
    - **MQに対して送信アプリケーション**として稼働
      - ✓ キューマネージャーに対して、ローカル接続、クライアント接続とも可能
        - CCDT利用やTLS通信も可能
      - ✓ 複数のワーカーで稼働する場合は、それぞれがキューマネージャーに接続し、メッセージを送信
      - ✓ JMSインターフェースでアクセス
        - JMSフォーマットではない通常のMQメッセージ（MQRFH2ヘッダーなし）を送信することも可能
      - ✓ 一部のMQMDプロパティは指定可能
        - Expiry（存続時間）、Persistence（永続性）
      - ✓ 送信時のコード変換は不可
        - ストリングデータはUnicodeで送信される
        - MQMD.CodedCharSetIdは1208



MQ Connectorの基本動作

- KafkaレコードのValueをMQメッセージのデータ部にマッピング
  - 内部では、 **converter**がValueをJavaオブジェクトに変換（ Deserialize）し、**builder**がデータ部に変換
    - ✓ ユーザは、構成時にデータ部/Valueのデータ・フォーマットに応じて、適切なbuilder、converterを選択する（後述）
  - KafkaレコードのKeyをMQMDのCorrelIdにマッピングすることも可（オプション）
    - ✓ CorrelIdのフォーマットはバイナリ固定のため、builderの指定は不要
    - ✓ converterのみKeyのデータ・フォーマットに応じて選択



MQ Connectorの基本動作

- builderはIESが提供
  - Source用はRecordBuilder、Sink用はMessageBuilder

Source用builder	用途
com.ibm.eventstreams.connect.mqsource.builders.DefaultRecordBuilder	バイナリデータ、ストリングデータを扱う場合
com.ibm.eventstreams.connect.mqsource.builders.JsonRecordBuilder	JSONデータを扱う場合

Sink用builder	用途
com.ibm.eventstreams.connect.mqsink.builders.DefaultMessageBuilder	バイナリデータ、ストリングデータを扱う場合
com.ibm.eventstreams.connect.mqsink.builders.JsonMessageBuilder	JSONデータを扱う場合
com.ibm.eventstreams.connect.mqsink.builders.ConverterMessageBuilder	

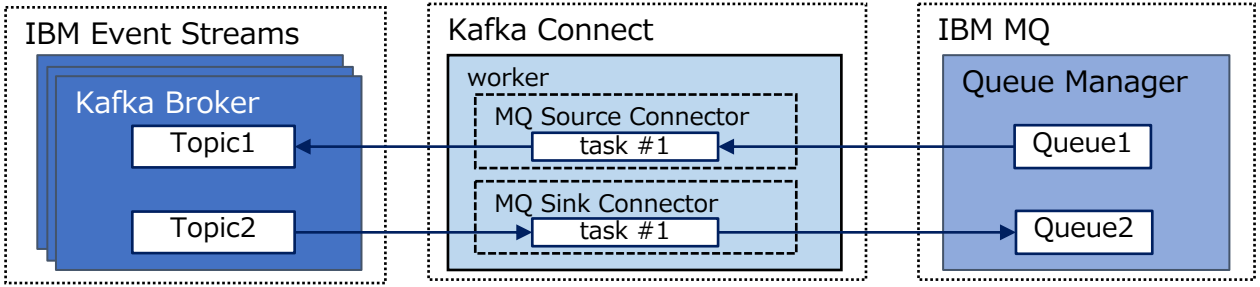
- 主要なconverterはApache Kafkaから提供
  - Source用、Sink用の区別はない
  - 構成時にKey用、Value用それぞれを指定

converter	用途
org.apache.kafka.connect.converters.ByteArrayConverter	Key/Valueをバイナリデータで扱う場合
org.apache.kafka.connect.storage.StringConverter	Key/Valueをストリングデータとして扱う場合
org.apache.kafka.connect.json.JsonConverter	Key/ValueをJSONデータとして扱う場合

構成手順

構成手順

- 構成の流れ
  - ① MQの構成
    1. キューマネージャーの構成
  - ② IESの構成
    1. 連携用トピックの作成
    2. 接続設定
  - ③ Kafka Connectの構成
    1. Apache Kafkaモジュールの用意
    2. Kafka Connectプロパティ・ファイルの設定
  - ④ MQ Connectorの構成、起動
    1. MQ Connectorモジュールの用意
    2. MQ Connectorプロパティ・ファイルの設定（standaloneモードの場合のみ）
    3. ワーカーの起動
    4. MQ Connectorの登録、起動（distributedモードの場合のみ）



①MQの構成

1. キューマネージャーの構成

- IESと連携するキュー、およびMQ Connectorが接続するために必要なオブジェクト（キュー、チャネル、リスナー等）を定義
  - MQ Connectorは、ローカル接続、クライアント接続とも可能
  - チャネル定義テーブル（CCDT）の利用も可
- 必要に応じてセキュリティ設定を実施
  - MQ Connectorは、ユーザID／パスワードの送付、TLS通信可能

- 当資料では、下記リンクの「Setting up the queue manager」の構成を前提に以降の設定を行う
  - MQ Source Connectorの場合
    - ✓ https://ibm.github.io/event-streams/connecting/mq/source/
  - MQ Sink Connectorの場合
    - ✓ https://ibm.github.io/event-streams/connecting/mq/sink/

MQ構成	設定
キューマネージャー	QM1
ポート	1414
サーバー接続チャネル	MYSVRCONN
Source用キュー	MYQSOURCE
Sink用キュー	MYQSINK
ユーザーID / PW	alice / passw0rd

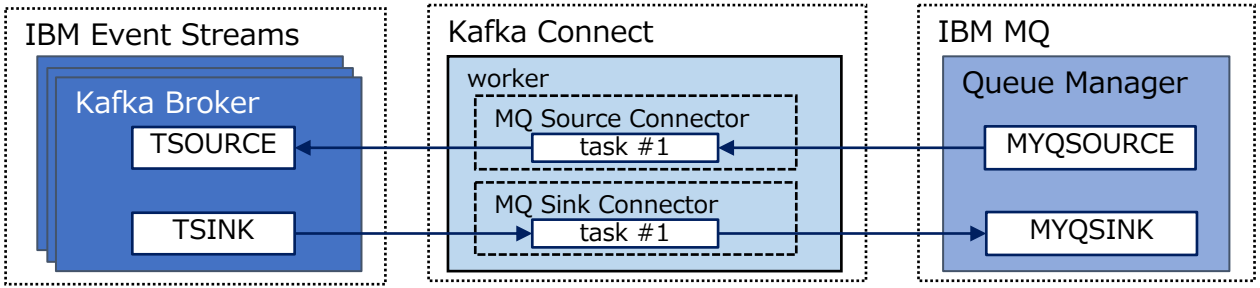
②IESの構成

1. 連携用トピックの作成

- IES上にMQと連携するトピックを作成する

- 当資料では以下の構成を前提に以降の設定を行う

IES構成	設定
Source用トピック	TSOURCE
Sink用トピック	TSINK



②IESの構成

2. 接続設定

- MQ ConnectorはIESのProducer/Consumerアプリケーションとして稼働するため、IESに接続する際に必要となる以下の操作を行う（操作手順は次ページ）

a. 接続情報の確認

- IESのブートストラップ・サーバーのIPアドレスとポート番号を確認

b. トラストストアの入手

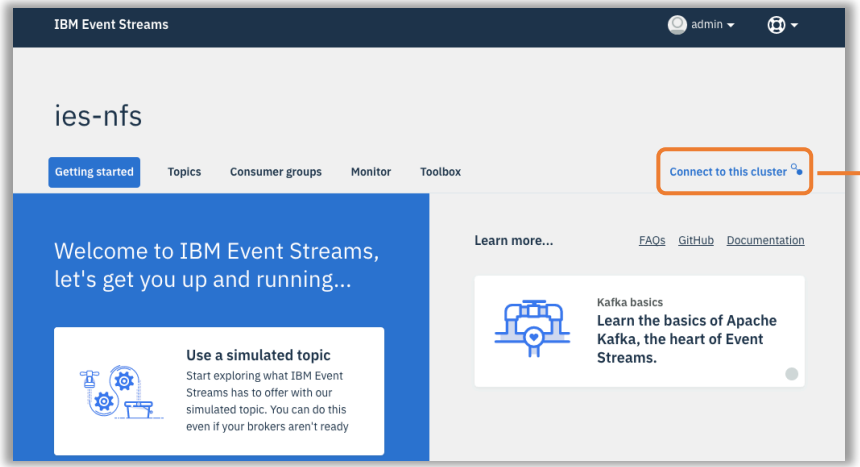
- IESとTLS通信するためのトラストストアを入手
- 入手したトラストストアはMQ Connector稼働環境の任意の箇所に配置

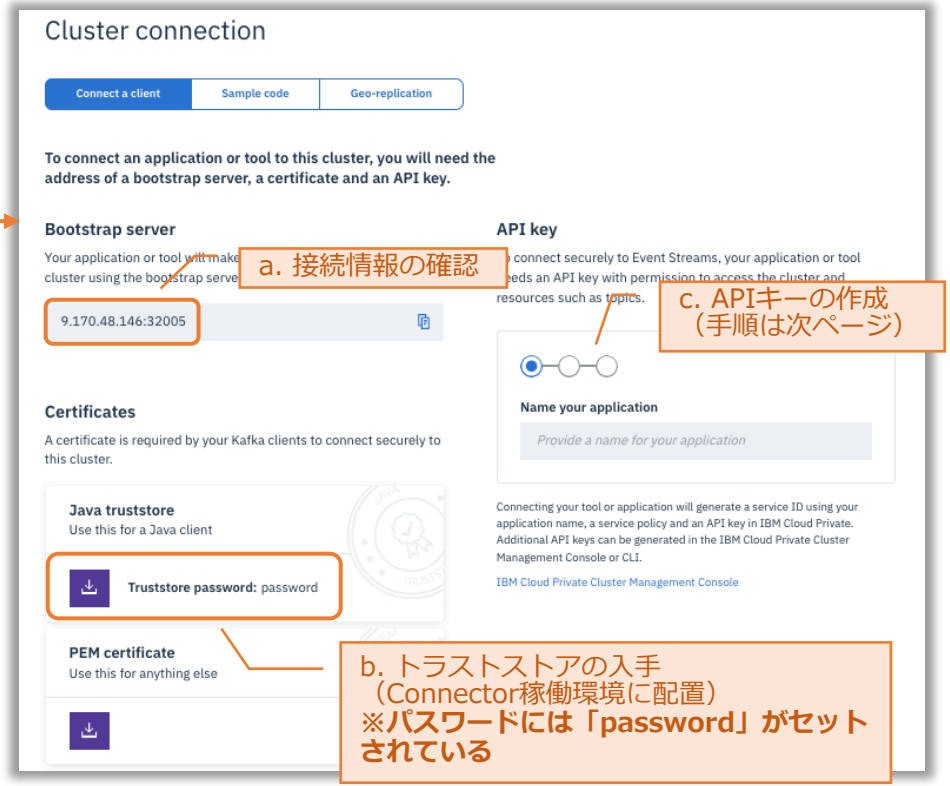
c. APIキーの作成

- IESのアクセスする際に指定するAPIキーを作成

②IESの構成

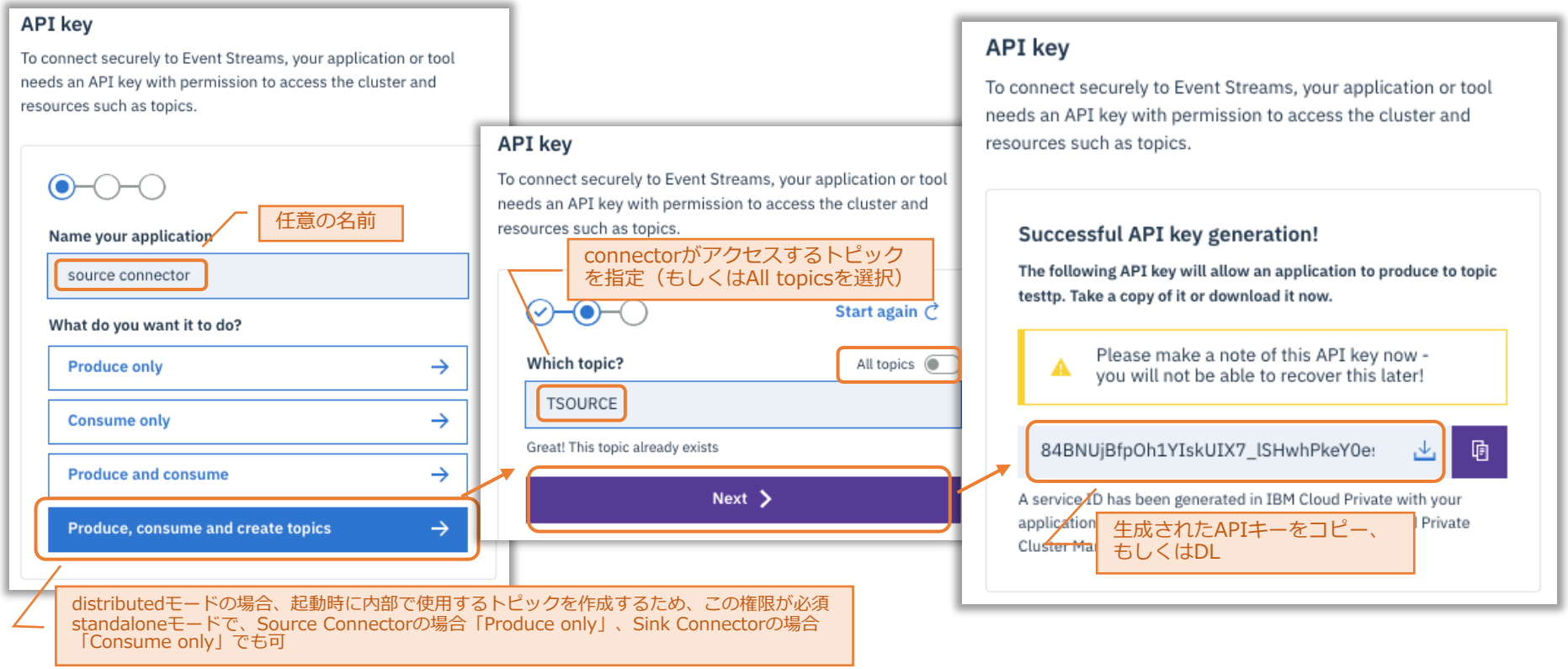
- IES管理画面から、以下の操作を実施





②IESの構成

- APIキーの作成



③Kafka Connectの構成

1. Apache Kafkaモジュールの用意
   - Apache Kafka 2.0.0以降をMQ Connector稼働環境に用意
     - 以下からモジュールをダウンロードし、任意の箇所に展開
       - ✓ https://kafka.apache.org/downloads

③Kafka Connectの構成

2. Kafka Connectプロパティ・ファイルの設定

- ③の1で展開したApache Kafkaのconfigディレクトリ下の下記ファイルをコピーし、編集
  - connect-standalone.properties　　：standaloneモードで稼働する場合
  - connect-distributed.properties　　：distributedモードで稼働する場合
  - ファイル名は任意に指定可

- 上記プロパティ・ファイルには主にIESと連携するための設定を行う
  - プロパティについては、下記リンク参照
    - ✓ https://ibm.github.io/event-streams/connecting/mq/source/
    - ✓ https://ibm.github.io/event-streams/connecting/mq/sink/

※稼働モード、およびMQ Source/Sinkで必要な設定プロパティが異なる

③Kafka Connectの構成

- standaloneモードでMQ Source Connectorを稼働させる場合の設定例

```
bootstrap.servers=9.170.48.146:32005

#セキュリティ設定
security.protocol=SASL_SSL
ssl.protocol=TLSv1.2
ssl.truststore.location=/work/es-cert.jks
ssl.truststore.password=password
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="token" password="84BNUjBfpOh1YIskUIX7_lSHwhPkeY0es6sRBI39IoFf";
ssl.endpoint.identification.algorithm=

#Source Connector用セキュリティ設定
producer.security.protocol=SASL_SSL
producer.ssl.protocol=TLSv1.2
producer.ssl.truststore.location=/work/es-cert.jks
producer.ssl.truststore.password=password
producer.sasl.mechanism=PLAIN
producer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="token"
password="84BNUjBfpOh1YIskUIX7_lSHwhPkeY0es6sRBI39IoFf";
producer.ssl.endpoint.identification.algorithm=

#デフォルトConverter
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter

#オフセット管理用ファイル
offset.storage.file.filename=/work/connect.offsets
```

②の2で確認したIESへの接続情報

②の2で入手したトラストストア

トラストストアのパスワード

usernameは”token” 固定

②の2で確認したAPIキー

Source ConnectorはProducerとして稼働するため、“producer”のプリフィックスを付加したセキュリティ関連プロパティを設定（設定値は上記と同じ値）
Sink Connectorの場合は、同様に”consumer”を付加したプロパティを設定

Converterは後述の手順でConnector毎に指定（オーバーライド）できるが、デフォルトConverterの設定は必須

standaloneモードの場合、オフセットの管理はローカルのファイルで行うため、任意のファイルを指定

③Kafka Connectの構成

- distributedモードでMQ Source Connectorを稼働させる場合の設定例

```
bootstrap.servers=9.170.48.146:32005

group.id=connect-cluster

offset.storage.topic=connect-offsets
offset.storage.replication.factor=1
config.storage.topic=connect-configs
config.storage.replication.factor=1
status.storage.topic=connect-status
status.storage.replication.factor=1

rest.port=8083

plugin.path=/work/MQConnector

security.protocol=SASL_SSL
ssl.protocol=TLSv1.2
ssl.truststore.location=/work/es-cert.jks
ssl.truststore.password=password
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="token" password="84BNUjBfpOh1YIskUIX7_lSHwhPkeY0es6sRBI39IoFf";
ssl.endpoint.identification.algorithm=

producer.security.protocol=SASL_SSL
producer.ssl.protocol=TLSv1.2
producer.ssl.truststore.location=/work/es-cert.jks
producer.ssl.truststore.password=password
producer.sasl.mechanism=PLAIN
producer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="token"
password="84BNUjBfpOh1YIskUIX7_lSHwhPkeY0es6sRBI39IoFf";
producer.ssl.endpoint.identification.algorithm=

key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
```

ConnectorのクラスターID
※Consumer Group IDとは重複しない名前を指定すること

distributedモードのConnectorが内部的に使用するトピック
任意の名前を指定可
起動時に存在しない場合は自動で作成される

REST APIのポート番号

③の2で入手したConnector JARが配置されているパスを指定

④MQ Connectorの構成、起動

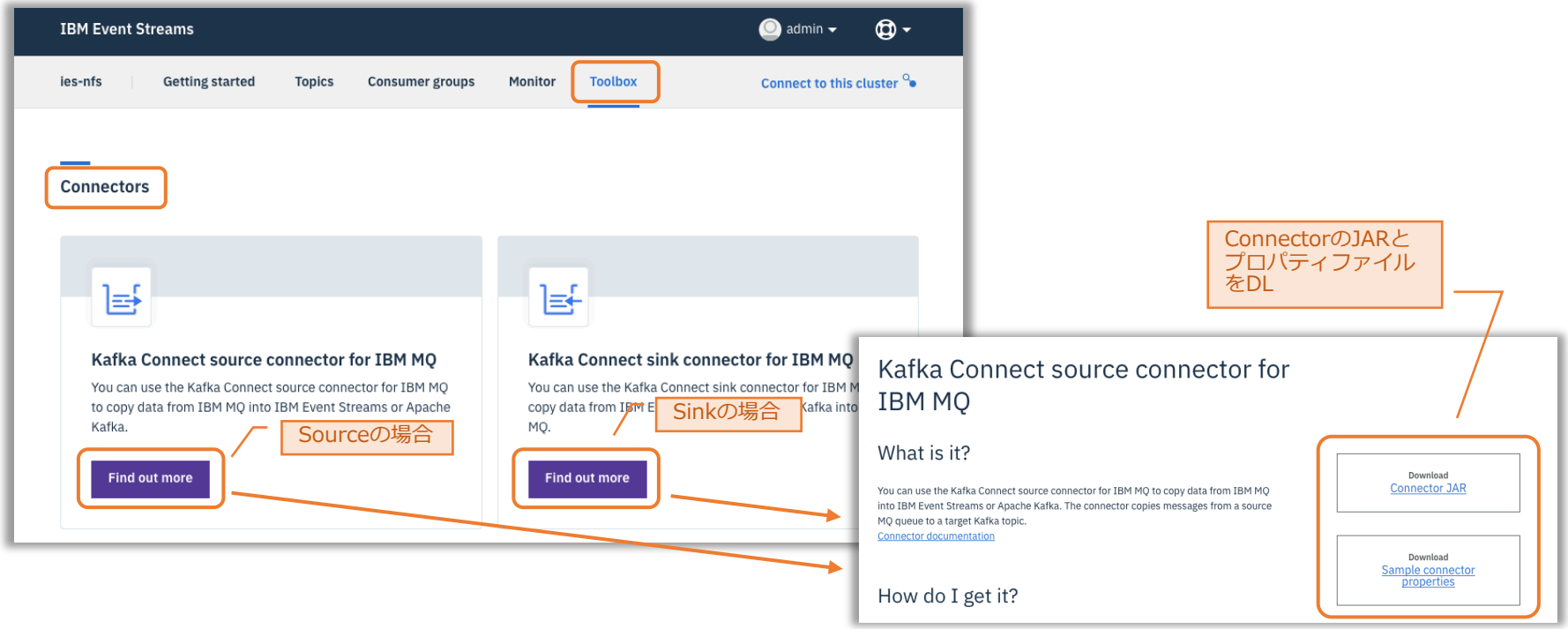
1. MQ Connectorモジュールの用意
   - IES管理画面の「Toolbox」タブから「Connectors」の対象Connectorを選択し、「Find out more」ボタンを押す（次ページ）
   - MQ ConnectorのJARファイルとプロパティ・ファイルをダウンロードし、稼働環境に配置

Connector	Connector JAR	プロパティ・ファイル
Source	kafka-connect-mq-source-<version>-jar-with-dependencies.jar	mq-source.properties
Sink	kafka-connect-mq-sink-<version>-jar-with-dependencies.jar	mq-sink.properties

   - Githubからも入手可
     - https://github.com/ibm-messaging/kafka-connect-mq-source
     - https://github.com/ibm-messaging/kafka-connect-mq-sink
     - ビルドしてJARファイル／propertiesファイルを生成

④MQ Connectorの構成、起動

- 操作手順



④MQ Connectorの構成、起動

2. MQ Connectorプロパティ・ファイルの用意

- standaloneモードで稼働させる場合のみ、以下の手順を実施
  （distributedモードの場合はプロパティ・ファイルと同等の設定をワーカー起動後にREST APIで設定する（④の4））
- ③の2の手順で入手したサンプルのmq-source.properties／mq-sink.propertiesをコピーして、編集

- 上記プロパティ・ファイルには主にMQと連携するための設定を行う
  - 「①MQの構成」で定義した内容に合わせる
  - 設定可能なプロパティはサンプルのファイルに記載されているため、必要に応じて値を設定
  - 連携対象のIES側のトピックもここで指定する

  - 扱うデータのフォーマットに応じて、builder、converterを設定
    - ✓ 設定例は後述

  - 個々のプロパティについては、ファイル内の説明、およびGithubのREADMEを参照
    - ✓ MQ Source Connector
      - https://github.com/ibm-messaging/kafka-connect-mq-source#configuration
    - ✓ MQ Sink Connector
      - https://github.com/ibm-messaging/kafka-connect-mq-sink#configuration

④MQ Connectorの構成、起動

- mq-source.propertiesの設定例

```
name=mq-source
connector.class=com.ibm.eventstreams.connect.mqsource.MQSourceConnector

tasks.max=1

topic=TSOURCE

mq.queue.manager=QM1
mq.connection.name.list=localhost(1414)
mq.channel.name=MYSVRCONN
mq.user.name=alice
mq.password=passw0rd

mq.queue=MYQSOURCE

mq.record.builder=com.ibm.eventstreams.connect.mqsource.builders.DefaultRecordBuilder
mq.message.body.jms=true
mq.record.builder.key.header=JMSMessageID
key.converter=org.apache.kafka.connect.converters.ByteArrayConverter
value.converter=org.apache.kafka.connect.converters.ByteArrayConverter
```

タスクの数

IES側のトピック名

MQへの接続情報、連携するキュー名、セキュリティ設定等を設定

Source Connectorが受信するキュー

builder、converterはデータ・フォーマットに応じて設定

④MQ Connectorの構成、起動

- mq-sink.propertiesの設定例

```
name=mq-sink
connector.class=com.ibm.eventstreams.connect.mqsink.MQSinkConnector

tasks.max=1

topics=TSINK

mq.queue.manager=QM1
mq.connection.name.list=localhost(1414)
mq.channel.name=MYSVRCONN
mq.user.name=alice
mq.password=passw0rd

mq.queue=MYQSINK

mq.persistent=true
mq.time.to.live=0

mq.message.builder=com.ibm.eventstreams.connect.mqsink.builders.DefaultMessageBuilder
mq.message.body.jms=true

key.converter=org.apache.kafka.connect.converters.ByteArrayConverter
value.converter=org.apache.kafka.connect.converters.ByteArrayConverter
```

タスクの数

IES側のトピック名

MQへの接続情報、連携するキュー名、セキュリティ設定等を設定

Sink Connectorが送信するキュー

送信するメッセージの属性の一部を指定可能

builder、converterはデータ・フォーマットに応じて設定

④MQ Connectorの構成、起動

- Source Connectorが出力するKafkaレコードの Key/Value値に影響するプロパティ

mq-source.properties	説明
mq.record.builder	扱うデータに応じて以下のいずれかのbuilderを指定（次ページ参照） ・com.ibm.eventstreams.connect.mqsource.builders.DefaultRecordBuilder ・com.ibm.eventstreams.connect.mqsource.builders.JsonRecordBuilder
mq.message.body.jms	MQメッセージがJMSフォーマット（MQRFH2ヘッダー付き）かどうかをtrue / falseで指定 デフォルトは、false
mq.record.builder.key.header	MQMDのMsgIdもしくはCorrelIdをKeyにマッピングする場合に以下をいずれかを指定 JMSMessageID / JMSCorrelationID / JMSCorrelationIDAsBytes
value.converter	builderによって生成されたJavaオブジェクトのタイプに合わせてconverterを指定（次ページ参照）
key.converter	mq.record.builder.key.headerの設定に合わせてconverterを指定

- MQメッセージのフォーマットと設定の組み合わせにおけるKey/Value値の例は次ページ参照

④MQ Connectorの構成、起動

- MQメッセージのフォーマットとプロパティ設定によって、出力されるKafkaレコードのValue値が決まる

MQメッセージのフォーマット	mq-source.propertiesの設定	出力されるKafkaレコードのValue値
Anyフォーマット	mq.message.body.jms=false mq.record.builder=DefaultRecordBuilder value.converter=ByteArrayConverter	MQメッセージのデータ部がそのままバイナリデータとしてValueにマッピング JMSメッセージの場合、MQRFH2ヘッダーも含めたデータ部がそのままValueにマッピングされる ストリングデータ/JMS TextMessageの文字コード変換は行われない
ストリング・フォーマット （MQMD.format=MQSTRのストリング・データ,JMS TextMessage）	mq.message.body.jms=true mq.record.builder=DefaultRecordBuilder value.converter=StringConverter	MQメッセージのストリングデータがValueにマッピング JMSのTextMessageの場合は、MQRFH2ヘッダー部を除いたデータのみがValueにマッピングされる ストリング・データ/JMS TextMessageの文字コードがUnicode（UTF8）以外の場合、UTF8にコード変換される
バイナリ・フォーマット （JMSのBytesMessage）	mq.message.body.jms=true mq.record.builder=DefaultRecordBuilder value.converter=ByteArrayConverter	JMSのBytesMessageのMQRFH2ヘッダー部を除いたデータのみがValueにマッピングされる
JSONフォーマット	mq.record.builder=JsonRecordBuilder （※JsonRecordBuilderの場合 mq.message.body.jmsの設定は使われない） value.converter=JsonConverter	MQメッセージのデータ部の値がJSONデータとしてValueにマッピング

※builder、converterのパッケージ名は省略（実際にはパッケージ名も含めて指定してください）

- フォーマットとプロパティ設定が適切な組み合わせでないと、実行時にエラーとなる場合もある
  - ✓ 例えば、上記表の上から2つ目のケースにおいて、value.converterをByteArrayConverterにすると以下のExceptionが発生する
    - ByteArrayConverter is not compatible with objects of type class java.lang.String

④MQ Connectorの構成、起動

- Key値は以下のプロパティの設定によって決まる

MQMD.MsgId/CorrelIdのフォーマット	mq-source.propertiesの設定	出力されるKafkaレコードのKey値
バイナリデータ	mq.record.builder.key.header=JMSMessageID key.converter=StringConverter	MQMDのMsgIdのヘキサ表記（ストリングデータ）
バイナリデータ	mq.record.builder.key.header=JMSCorrelationID key.converter=StringConverter	MQMDのCorrelIdのヘキサ表記（ストリングデータ）
バイナリデータ	mq.record.builder.key.header=JMSCorrelationIDAsBytes key.converter=ByteArrayConverter	MQMDのCorrelIdのバイナリデータ

※MsgId/CorrelIdは常にバイナリデータ

※converterのパッケージ名は省略（実際にはパッケージ名も含めて指定してください）

④MQ Connectorの構成、起動

- Sink Connectorが出力するMQメッセージのMQMD/データに影響するプロパティ

mq-sink.properties	説明
mq.message.builder	扱うデータに応じて以下のいずれかのbuilderを指定（次ページ参照） ・com.ibm.eventstreams.connect.mqsink.builders.DefaultMessageBuilder ・com.ibm.eventstreams.connect.mqsink.builders.JsonMessageBuilder ・com.ibm.eventstreams.connect.mqsink.builders.ConverterMessageBuilder
mq.message.body.jms	MQメッセージがJMSフォーマット（MQRFH2ヘッダー付き）かどうかをtrue / falseで指定 デフォルトは、false
mq.message.builder.key.header	Key値をMQMDのCorrelIdにマッピングする場合、”JMSCorrelationID”を指定 指定しない場合、CorrelIdの値は、MQCI_NONE（2進数の0）
value.converter	Valueのデータ・フォーマットに応じたconverterを指定
key.converter	Keyのデータ・フォーマットに応じたconverterを指定
mq.time.to.live	メッセージのExpiry時間を指定（ミリ秒単位）　デフォルトは0（Unlimited）
mq.persistent	メッセージの永続性を指定（true / false）　デフォルトはtrue（パーシステント）

④MQ Connectorの構成、起動

- Kafkaレコードのフォーマットとプロパティ設定によって、出力されるMQメッセージの値が決まる

KafkaレコードのValueのフォーマット	mq-sink.propertiesの設定	MQメッセージの値
Anyフォーマット	value.converter=ByteArrayConverter mq.message.body.jms=false mq.message.builder=DefaultMessageBuilder	Valueのデータがそのまま MQメッセージのデータ部にマッピング MQMD.Format：ブランク
ストリング・フォーマット	value.converter=StringConverter mq.message.body.jms=false mq.message.builder=DefaultMessageBuilder	Valueのストリングデータが MQメッセージのデータ部にマッピング MQMD.Format：MQSTR CodedCharSetId：1208
JSONフォーマット	value.converter=JsonConverter mq.message.body.jms=false mq.message.builder=JsonMessageBuilder	ValueのJSONデータが MQメッセージのデータ部にマッピング MQMD.Format：MQSTR CodedCharSetId：1208
Anyフォーマット	value.converter=ByteArrayConverter mq.message.body.jms=true mq.message.builder=DefaultMessageBuilder	MQRFH2ヘッダーが付与される（JMSフォーマット） Valueのデータがそのまま MQRFH2ヘッダーの後ろにマッピング MQMD.Format：MQHRF2（MQRFH2.Flormatはブランク） MQRFH2.Msd：jms_bytes（JMSのBytesMessage）
ストリング・フォーマット	value.converter=StringConverter mq.message.body.jms=true mq.message.builder=DefaultMessageBuilder	MQRFH2ヘッダーが付与される（JMSフォーマット） Valueのストリングデータが MQRFH2ヘッダーの後ろにマッピング MQMD.Format：MQHRF2（MQRFH2.FlormatはMQSTR） MQRFH2.Msd：jms_text（JMSのTextMessage） CodedCharSetId：1208（MQRFH2.CodedCharSetId:1208）
JSONフォーマット	value.converter=JsonConverter mq.message.body.jms=true mq.message.builder=JsonMessageBuilder	MQRFH2ヘッダーが付与される（JMSフォーマット） ValueのJSONデータが MQRFH2ヘッダーの後ろにマッピング MQMD.Format：MQHRF2（MQRFH2.FlormatはMQSTR） MQRFH2.Msd：jms_text（JMSのTextMessage） CodedCharSetId：1208（MQRFH2.CodedCharSetId:1208）

※builder、converterのパッケージ名は省略（実際にはパッケージ名も含めて指定してください）

④MQ Connectorの構成、起動

- MQメッセージのMQMD.CorrelIdは以下のプロパティの設定によって決まる

KafkaレコードのKeyのフォーマット	mq-sink.propertiesの設定	MQMDのCorrelIdの値
Anyフォーマット	mq.message.builder.key.header=JMSCorrelationID key.converter=ByteArrayConverter	Key値のバイナリデータ
ストリングフォーマット	mq.message.builder.key.header=JMSCorrelationID key.converter=StringConverter	Key値のストリング・データの16進数表記（バイナリデータ）

※converterのパッケージ名は省略（実際にはパッケージ名も含めて指定してください）

  - ✓ MQMDのCorrelIdは24バイトのため、Key値が24バイト以上の場合、超過分はトランケートされる

④MQ Connectorの構成、起動

3. ワーカーの起動

- Kafkaモジュールのbinディレクトリ下にあるシェルを利用してワーカーを起動
  - standaloneモードで起動する場合　　: connect-standalone.sh
  - distributedモードで起動する場合　　: connect-distributed.sh

- ワーカーの稼働モードによって、MQ Connectorの指定方法、および起動方法が異なる

- standaloneモードの場合
  - MQ Connectorは、ワーカー起動時に引数として指定し、ワーカーと共に起動する

- distributedモードの場合
  - ワーカーを起動した後に、MQ ConnectorをREST APIで登録し、起動する

④MQ Connectorの構成、起動

- standaloneモードの起動方法
  - CLASSPATH環境変数にMQ ConnectorのJARファイルを指定
    - ✓ 複数のConnectorを起動する場合は、それぞれのJARファイルを指定
  - 以下の構文で実行

```
# connect-standalone.sh <Kafka Connectプロパティ・ファイル> <MQ Connectorプロパティ・ファイル>
```

    - ✓ Kafka Connectプロパティ・ファイルには、③の2で用意したconnect-standalone.propertiesを指定
    - ✓ MQ Connectorプロパティ・ファイルには、④の2で用意したmq-source.properties/mq-sink.propertiesを指定
      - 複数のConnectorを起動する場合は、スペース区切りで複数のプロパティ・ファイルを指定
  - 実行例：MQ Source Connectorをstandaloneモードで起動する場合

```
# export CLASSPATH=/work/kafka-connect-mq-source-1.0.1-jar-with-dependencies.jar
# ./bin/connect-standalone.sh ./config/connect-standalone.properties /work/mq-source.properties
```

    - ✓ 実行ディレクトリは、Kafkaモジュールを展開したディレクトリとする
  - 実行後は標準出力にログが出力されるため、エラーが出ていないか確認
    - ✓ 起動に成功すると、以下のようなメッセージが出力される

```
[2019-02-19 13:46:53,451] INFO REST server listening at http://9.68.254.151:8083/, advertising URL http://9.68.254.151:8083/
(org.apache.kafka.connect.runtime.rest.RestServer:217)
[2019-02-19 13:46:53,452] INFO Kafka Connect started (org.apache.kafka.connect.runtime.Connect:55)
（省略）
[2019-02-19 13:46:53,651] INFO Created connector mq-source (org.apache.kafka.connect.cli.ConnectStandalone:104)
[2019-02-19 13:46:54,238] INFO Building records using com.ibm.eventstreams.connect.mqsource.builders.DefaultRecordBuilder
(com.ibm.eventstreams.connect.mqsource.builders.DefaultRecordBuilder:44)
[2019-02-19 13:46:54,745] INFO Connection to MQ established (com.ibm.eventstreams.connect.mqsource.JMSReader:197)
[2019-02-19 13:46:54,745] INFO WorkerSourceTask{id=mq-source-0} Source task finished initialization and start
(org.apache.kafka.connect.runtime.WorkerSourceTask:199)
```

④MQ Connectorの構成、起動

- distributedモードの起動方法
  - 以下の構文で実行

```
# connect-distributed.sh <Kafka Connectプロパティ・ファイル>
```

    - ✓ Kafka Connectプロパティ・ファイルには、③の2で用意したconnect-distributed.propertiesを指定

  - 実行例：distributedモードで起動する場合

```
# ./bin/connect-distributed.sh ./config/connect-distributed.properties
```

    - ✓ 実行ディレクトリは、Kafkaモジュールを展開したディレクトリとする

  - 実行後は標準出力にログが出力されるため、エラーが出ていないか確認
    - ✓ 起動に成功すると、以下のようなメッセージが出力される

```
[2019-06-19 07:27:19,957] INFO Started KafkaBasedLog for topic connect-configs (org.apache.kafka.connect.util.KafkaBasedLog:155)
[2019-06-19 07:27:19,958] INFO Started KafkaConfigBackingStore (org.apache.kafka.connect.storage.KafkaConfigBackingStore:253)
（省略）
[2019-06-19 07:27:24,273] INFO Starting connectors and tasks using config offset -1
(org.apache.kafka.connect.runtime.distributed.DistributedHerder:858)
[2019-06-19 07:27:24,274] INFO Finished starting connectors and tasks (org.apache.kafka.connect.runtime.distributed.DistributedHerder:868
```

  - 次の手順でMQ Connectorを登録し、起動する

④MQ Connectorの構成、起動

4. MQ Connectorの登録、起動

- distributedモードで稼働させる場合のみ、以下の手順を実施

- MQ ConnectorをREST APIで登録する
  - ポート番号　　：connect-distributed.propertiesのrest.portで指定した値（デフォルトは8083）
  - URL　　：http://<hostname>:<rest.port>/connectors
  - メソッド　　：POST
  - データ　　：JSONオブジェクト形式
    name (string)：MQ Connector名（任意）
    config (map)：MQ Connctorのプロパティ
    ※standaloneモードで稼働させる場合のmq-source.properties/
    mq-sink.propertiesに設定するプロパティと同じ（④の2参照）

- 登録が成功すると、自動的にConnectorが起動する
  - 起動に成功すると、以下のようなメッセージが出力される

```
[2019-06-19 08:58:55,735] INFO Connection to MQ established (com.ibm.eventstreams.connect.mqsink.JMSWriter:200)
[2019-06-19 08:58:55,735] INFO WorkerSinkTask{id=mq-sink-0} Sink task finished initialization and start
(org.apache.kafka.connect.runtime.WorkerSinkTask:302)
```

④MQ Connectorの構成、起動

- 実行例：MQ Sink Connectorを登録する場合

```
# curl -H "Content-Type: application/json" -X POST http://localhost:8083/connectors -d '
{
   "name": "mq-sink",
   "config": {
      "tasks.max":1,
      "connector.class":"com.ibm.eventstreams.connect.mqsink.MQSinkConnector",
      "mq.queue.manager":"QM1",
      "mq.connection.name.list":"localhost(1414)",
      "mq.channel.name":"MYSVRCONN",
      "mq.queue":"MYQSINK",
      "mq.user.name":"alice",
      "mq.password":"passw0rd",
      "mq.message.builder":"com.ibm.eventstreams.connect.mqsink.builders.DefaultMessageBuilder",
      "mq.message.body.jms":"false",
      "topics":”TSINK",
      "value.converter":"org.apache.kafka.connect.converters.ByteArrayConverter"
   }
}
'
```

④MQ Connectorの構成、起動

- 登録は以下のREST APIで確認

```
# curl http://127.0.0.1:8083/connectors
 （レスポンス）
["mq-sink"]
```

```
# curl http://127.0.0.1:8083/connectors/mq-sink
 （レスポンス）
{
  "name": "mq-sink",
  "config": {
    "connector.class": "com.ibm.eventstreams.connect.mqsink.MQSinkConnector",
    "mq.connection.name.list": "localhost(1414)",
    "mq.password": "passw0rd",
    "tasks.max": "1",
    "mq.queue.manager": "QM1",
    "topics": "testtp",
    "mq.queue": "MYQSINK",
    "mq.message.body.jms": "false",
    "mq.channel.name": "MYSVRCONN",
    "mq.user.name": "alice",
    "mq.message.builder": "com.ibm.eventstreams.connect.mqsink.builders.DefaultMessageBuilder",
    "name": "mq-sink",
    "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter"
  },
  "tasks": [{"connector": "mq-sink","task": 0}],
  "type": "sink”
}
```

④MQ Connectorの構成、起動

- ステータスは以下のREST APIで確認

```
# curl http://127.0.0.1:8083/connectors/mq-sink/status
 （レスポンス）
{
  "name": "mq-sink",
  "connector": {
    "state": "RUNNING",
    "worker_id": "9.68.254.50:8083"
  },
  "tasks": [
    {
      "state": "RUNNING",
      "id": 0,
      "worker_id": "9.68.254.50:8083"
    }
  ],
  "type": "sink"
}
```

- REST APIの詳細については以下を参照
  - https://docs.confluent.io/current/connect/references/restapi.html

④MQ Connectorの構成、起動

- MQ Connector起動後は、任意のMQアプリケーションおよびKafkaアプリケーションを使用して、MQとIESが連携できることを確認

参考リンク

- IBM Event Streams Documentation
  - https://ibm.github.io/event-streams/
  - MQ Connectorについては下記リンク参照
    - https://ibm.github.io/event-streams/connecting/mq/
- Kafka Connect for IBM MQ @GitHub
  - https://github.com/ibm-messaging/kafka-connect-mq-source
  - https://github.com/ibm-messaging/kafka-connect-mq-sink
- Apache Kafka Documentation
  - https://kafka.apache.org/documentation/
- Kafka Connect @confluent
  - https://docs.confluent.io/current/connect/index.html