ps-plugin-KafkaTools

KafkaTools provides the following functionality that allows publishing to and consuming from topics on Kafka Servers.

Smart Services

- Publish To Kafka
- Consume From Kafka

Performance and Scalability Considerations

Because of the nature of Kafka which is designed to handle large volumes of messages, Appian strongly recommends to consider the performance and scalability impacts of consuming large volumes of messages from Kafka into Appian.

To implement this plugin successfully in an application, you must:

1. Use the Transaction Manager to throttle the processing of the messages
2. Only consume messages that are relevant to your use cases. This can be done by creating dedicated Kafka topics for Appian and implementing message filters
3. Conduct performance testing with a production-like volume of messages and with the final process models and rules designed to handle the messages
4. Tune the configuration of the polling intervals, sleeping intervals, number of consumers and the Transaction Manager settings to not overload Appian and to leave resources available for end-user activities

Installation Instructions

1. Install Transaction Manager (https://community.appian.com/b/appmarket/posts/transaction-manager)
    1. Follow instructions to create new job_type(s) and process model(s) to handle kafaka topics, could be one or multiple depending on requirements
2. For Appian Cloud environments, install the Kafka Tools plugin from the Appian App Market
3. For self-managed environments, copy the *ps-plugin-KafkaTools-X.jar* file into your *$APPIAN_HOME/data/_admin/plugins* directory. The plug-in will be installed and available automatically.
4. Create Third-Party Credentials (explained below)
    1. name: kafkaTools
    2. fields:
        - username
        - password
        - truststorepwd
        - keystorepwd
        - privatekeypwd

    - clientid
    - clientsecret
    - tokenurl
5. Give Third-Party Credentials access to KafkaTools plugin

Appian Secure Credentials Store

This plug-in uses a Third-Party Credentials Store to maintain the credentials for authentication with a Kafka Server. The configuration is available in the Admin Console under Third Party Credentials. The field names to be created are:

- username: username to use with the LoginModule. This is only required when using SASL or SASL_SSL.
- password: password to use with the LoginModule. This is only required when using SASL or SASL_SSL.
- truststorepwd: password of the TrustStore containing the Kafka server certificate. This is only required when using SSL or SASL_SSL.
- keystorepwd: password of the KeyStore containing the client private key to use to authenticate with the Kafka server. This is only required when using SSL.
- privatekeypwd: password of the private key to use to authenticate with the Kafka server. This is only required when using SSL.
- clientid: The client id of the oauth provider to use to authenticate with the Kafka server. This is only required when using OAUTHBEARER.
- clientsecret: The client secret of the oauth provider to use to authenticate with the Kafka server. This is only required when using OAUTHBEARER.
- tokenurl: The token url of the oauth provider use to authenticate with the Kafka server. This is only required when using OAUTHBEARER.

It is recommended that you mask the value of the passwords. Once the entry is created, the name will be used as a required input parameter to Kafka Tools Smart Service nodes.

Note: you must have the plug-in installed prior to creating the credentials. You must add the plug-in to the *Plug-Ins List* to allow KafkaTools to access the credentials.

If your server does not require authentication, you must still create an entry in the Third Party Credentials.

Smart Services

The nodes for these Smart Services are available under the *Integration Services* group and *Connectivity Services* subgroup of the Appian Process Modeler.

Publish To Kafka

This Smart Service implements Kafka Producer and sends a single payload to a single Kafka topic per execution.

Input Parameters

- **Secure Credentials Store Key:** use the name created in the Admin Console under Third Party Credentials (see above)
- **Servers:** provide a comma-separated list of *server name:port* values
- **Topic:** provide the name of the kafka server topic
- **Payload:** text value of the payload
- **Security Protocol:** provide the value of the security protocol to be used with the Kafka Server
- **SASL Mechanism:** if the security protocol is provided as "SASL_SSL", then this parameter must be provided, use OAUTHBEARER for oauth auth
- **TrustStore:** if the security protocol is provided as SSL or SASL_SSL, this parameter is an Appian document containing the TrustStore in JKS format
- **KeyStore:** if the security protocol is provided as SSL, this parameter is an Appian document containing the KeyStore in JKS format

Output Parameters

- **success:** returns *true* if smart service completed successfully and *false* if an exception has occurred
- **errorMessage:** returns *null* if smart service completed successfully and the exception message if an exception has occurred

Known Issues

Currently only supports text based payloads

Consume From Kafka

This Smart Service implements Kafka Consumer and subscribes to one topic on a Kafka server. It saves the payloads to a database table that is provided in the configuration of the node. The node runs for a configurable number of minutes and iterates between polling the Kafka server and sleeping.

Input Parameters

- **Servers:** provide a comma-separated list of *server name:port* values
- **Datasource Name:** provide the value that is specified in the *Data Source* dropdown in any of the Data Stores in Designer
- **TransactionTableName:** name of the database table where the Kafka records are saved into. Set to the default value when using the Transaction Manager to process the Kafka records
- **Security Protocol:** provide the value of the security protocol to be used with the Kafka Server
- **Secure Credentials Store Key:** use the name created in the Admin Console under Third Party Credentials (see above)
- **SASL Mechanism:** if the security protocol is provided as "SASL_SSL", then this parameter must be provided
- **TrustStore:** if the security protocol is provided as SSL or SASL_SSL, this parameter is an Appian document containing the TrustStore in JKS format
- **KeyStore:** if the security protocol is provided as SSL, this parameter is an Appian document containing the KeyStore in JKS format
- **Runtime In Minutes:** number of minutes that the Smart Service run for. Any value above 58 minutes will be defaulted to 58 minutes as Smart Service nodes timeout after 60 minutes and result in a process error

- **Group Id:** provide the value for the Group Id to be used to subscribe to the Kafka topic
- **NumConsumers:** provide the number of consumers to use to consume records
- **Topic:** provide the value for the Topic to subscribe on the Kafka server
- **Job Type Id:** provide the integer of the transaction job type these messages will be assigned to. Refer to the Transaction Manager documentation for more details
- **Polling Interval in Ms:** provide the number of milliseconds for the Kafka Consumer to poll a topic per iteration
- **Sleeping Interval In Ms:** provide the number of milliseconds for the thread to sleep between polling iterations
- **Session Timeout:** provide the number of milliseconds for the session timeout (default value is 30000)
- **Auto Offset Reset Config:** provide the auto offset reset config (default value is latest)
- **Deserializer Class Name:** full name of the class to be used with Kafka Consumer (default is text-based deserializer)
- **Message Filter:** jsonpath filter used to filter out message before processing in Appian. JsonPath documentation can be found here (https://goessner.net/articles/JsonPath/). Use this site (https://jsonpath.herokuapp.com/) for testing.

Output Parameters

- **success:** returns *true* if smart service completed successfully and *false* if an exception has occurred
- **errorMessage:** returns *null* if smart service completed successfully and the exception message if an exception has occurred

Payload Handler Process Models

These models will be configured an assigned through the transaction manager job types. See documentation for Transaction Manager Transaction Manager (https://community.appian.com/b/appmarket/posts/transaction-manager)