**WHOUTREACH-181**

**POC: Publish data to event streams using Apache NiFi**

Refer the following ibm documentation : <https://www.ibm.com/cloud/blog/connecting-ibm-event-streams-to-apache-nifi>

Create an Event Streams instance on IBM Cloud

1. Navigate to the **IBM Cloud Catalog**.
2. Select **Event Streams**.
3. Click **Create** to create a service instance
4. Click the **+** (Add) button to create a topic.
5. For this project, call the topic <<topic name>>, and accept the defaults

Use the **Service credentials** tab on the left side of the screen to create a new set of credentials that your application will use to access the service. Once the credentials are created, note the values for the user and password fields, along with the servers listed in the kafka\_brokers\_sasl section.

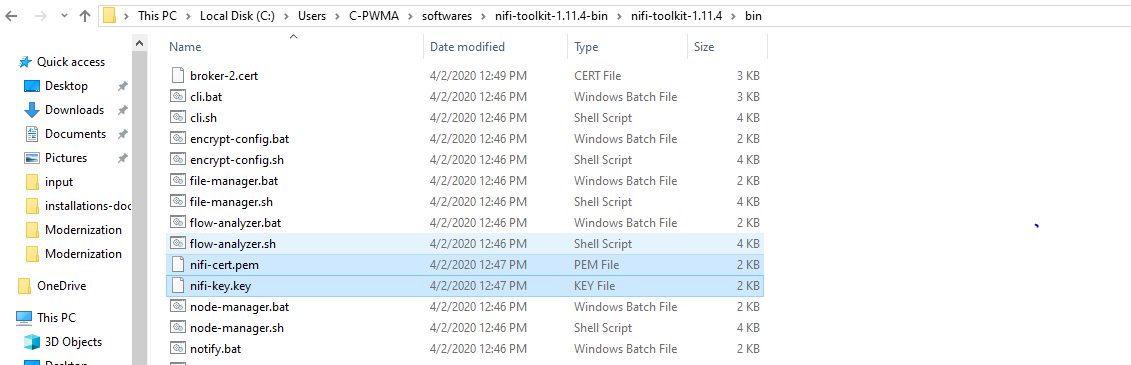
Install Nifi tool-kit : [nifi-toolkit-1.11.4-bin.zip](https://www.apache.org/dyn/closer.lua?path=/nifi/1.11.4/nifi-toolkit-1.11.4-bin.zip) from <https://nifi.apache.org/download.html>

Go to bin folder of Nifi-toolkit : C:\Users\C-PWMA\softwares\nifi-toolkit-1.11.4-bin\nifi-toolkit-1.11.4\bin

Open command prompt and execute the following command

tls-toolkit.bat standalone -n \*.nifi.svc.cluster.local -f C:/Users/C-PWMA/softwares/nifi-1.11.4-bin/nifi-1.11.4/conf/nifi.properties --trustStorePassword "<own pswd1>" --keyStorePassword "<own pswd2>”

It will generate two files :



Install Cygwin from <https://www.faqforge.com/windows/windows-10/how-to-install-cygwin-on-windows-10/>

Open Cygwin terminal go to bin folder of nifi-toolkit:



Execute the following command from Cygwin terminal:

openssl s\_client -connect broker-2-<<hostname>>.eventstreams.cloud.ibm.com:9093 -servername broker-2-<<hostname>>.eventstreams.cloud.ibm.com </dev/null \

        | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > broker-2.cert && \

    keytool -import -noprompt -trustcacerts \

        -alias kafka-broker -file broker-2.cert \

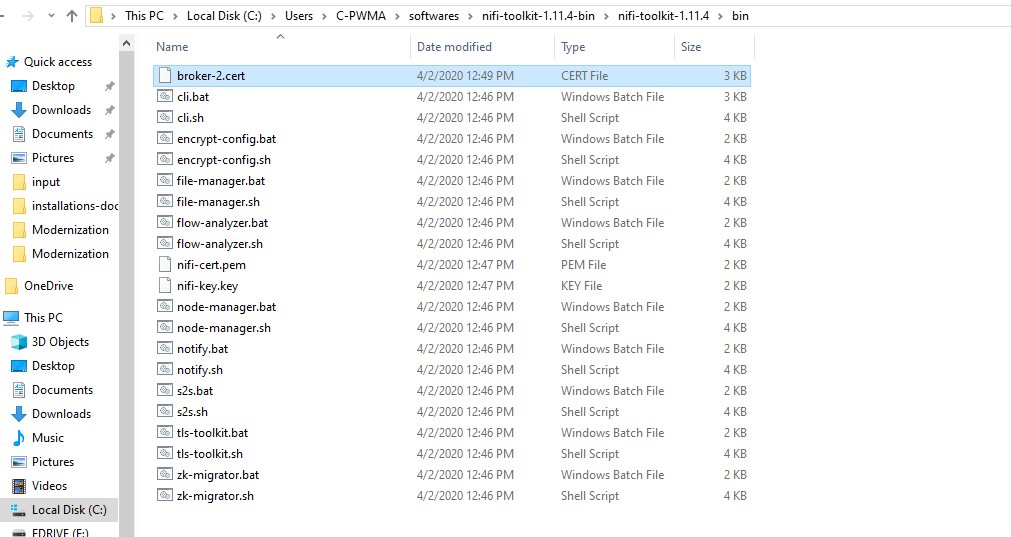
        -keystore /opt/nifi/nifi-current/\\*.<<nifi service - namespace>>.svc.cluster.local/truststore.jks -storepass "securePasswordTwo"

<<hostname>> your kafka brokers from eventstreams credentials

<<nifi service -namespace>> your namespace in CDT account.

<<securePasswordTwo>> you gave in the step-2

It will generate broker-2.cert file in your local directory



Go to your nifi folder in your local upto /conf path and execute the following commands from the Cygwin terminal

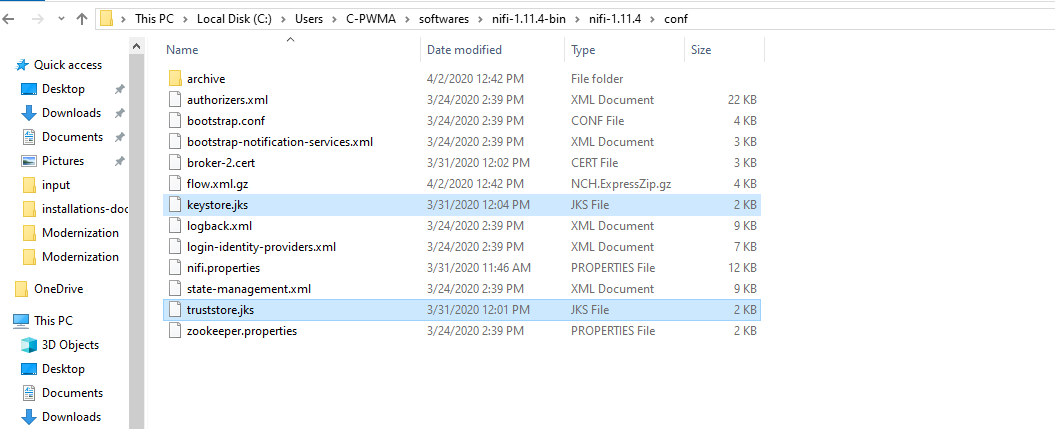
C:/Users/C-PWMA/softwares/nifi-1.11.4-bin/nifi-1.11.4/conf/

Commands:

keytool -import -file broker-2.cert -alias cacert -keystore truststore.jks -storepass "pswd you gave in step2(own pswd1)"

keytool -import -file broker-2.cert -alias cacert -keystore keystore.jks -storepass "pswd u gave in step2(own pswd2)"

these commands will generate two file in /conf folder in your local

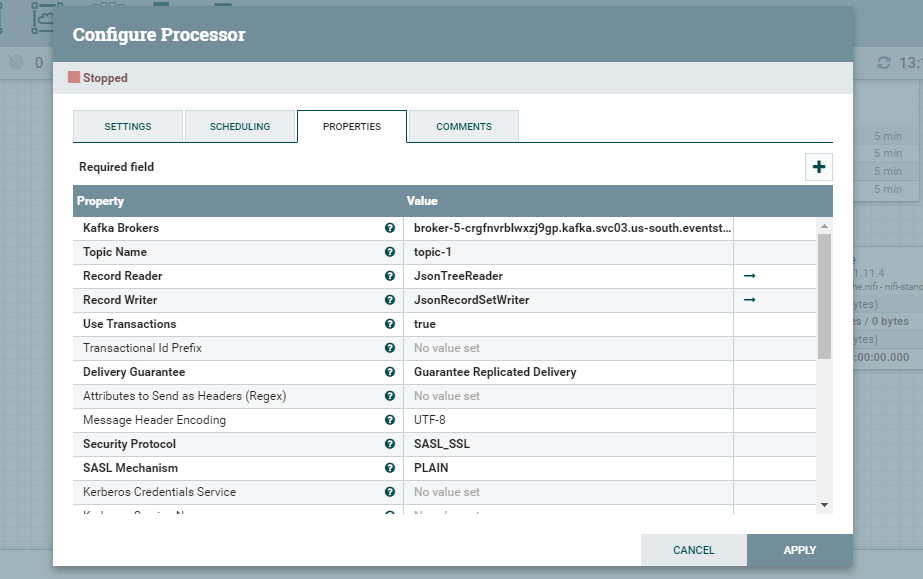


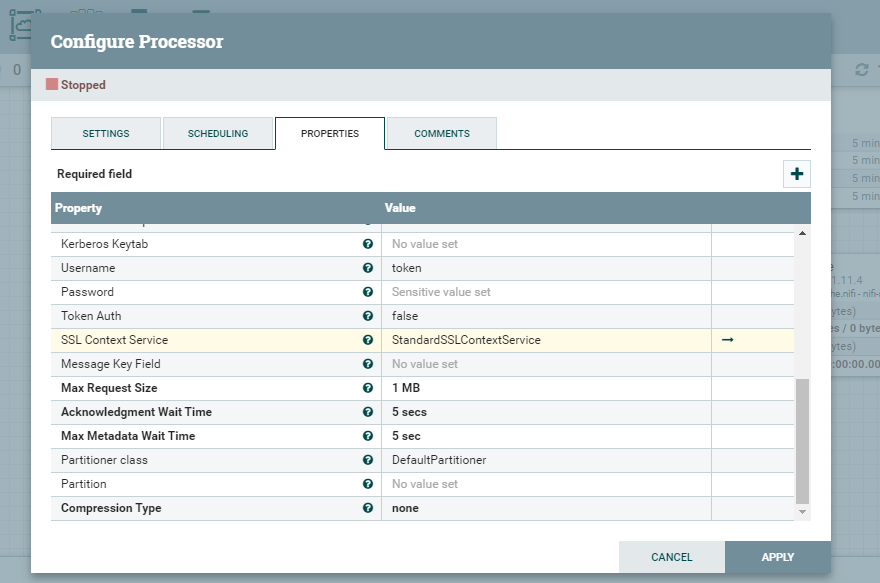
3. Configure an Apache NiFi Kafka consumer or producer

To start consuming or publishing events, add a [ConsumeKafkaRecord](https://nifi.apache.org/docs/nifi-docs/components/nifi-docs/components/org.apache.nifi/nifi-kafka-2-0-nar/1.9.0/org.apache.nifi.processors.kafka.pubsub.ConsumeKafka_2_0/index.html" \t "_blank) and  [PublishKafkaRecord](https://nifi.apache.org/docs/nifi-docs/components/nifi-docs/components/org.apache.nifi/nifi-kafka-2-0-nar/1.9.0/org.apache.nifi.processors.kafka.pubsub.PublishKafka_2_0/index.html) NiFi processor and change the following configurations.

1. Enter the comma separated list of Kafka Brokers from Service credentials.
2. Enter your Topic Name(s) for the topics that you want to consume from or publish to.
3. Select and configure a Record Reader and Record Writer.
4. Choose SASL\_SSL as the Security Protocol.
5. Enter the Username and Password retrieved from Service credentials.
6. Choose a Group ID.
7. Create a new SSL Context Service.

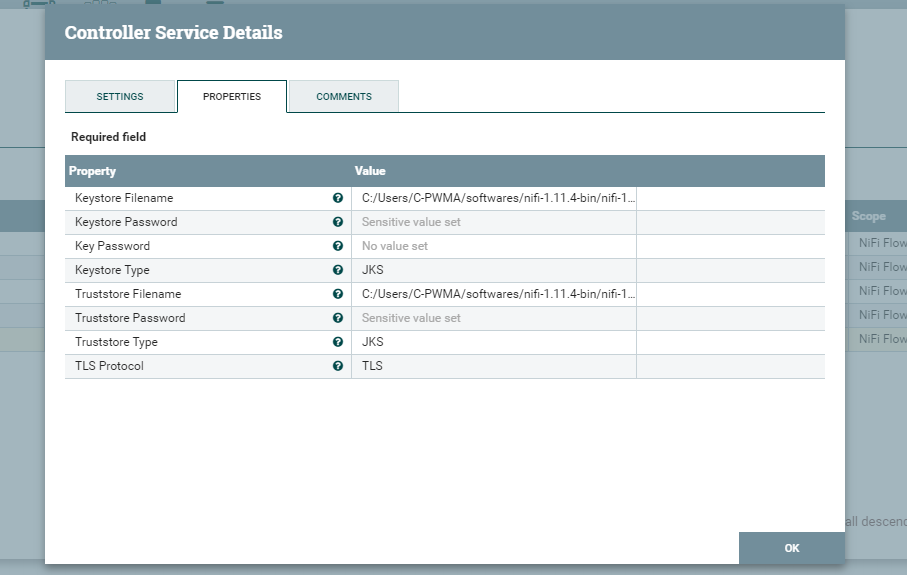
Once complete, click the arrow next to your SSL Context Service to go to the controllers page and start to configure your SSL Context Service.



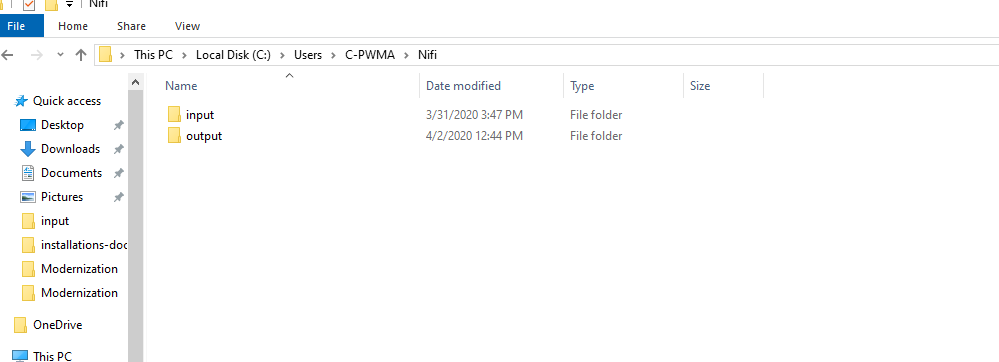


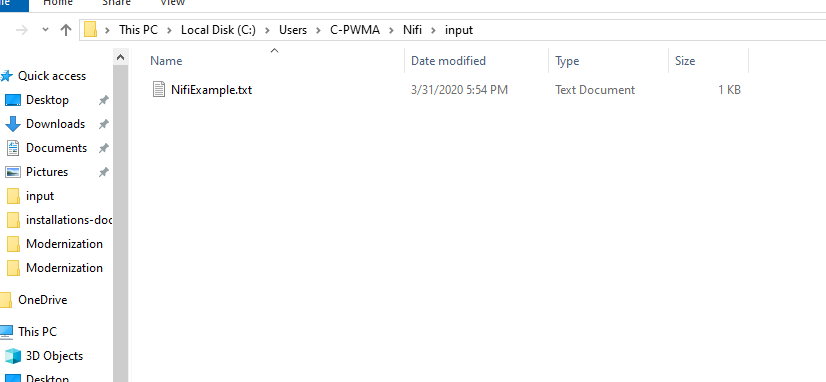
4. Configure the connected SSLContextService

The [SSL Context Service](https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-ssl-context-service-nar/1.9.2/org.apache.nifi.ssl.StandardSSLContextService/index.html) is what connects your Truststore and Keystore to Apache NiFi. Enter in the configurations to specify where the Keystore and Truststore filenames are located as well as their corresponding passwords. The image below shows example configurations to configure the SSLContextService:

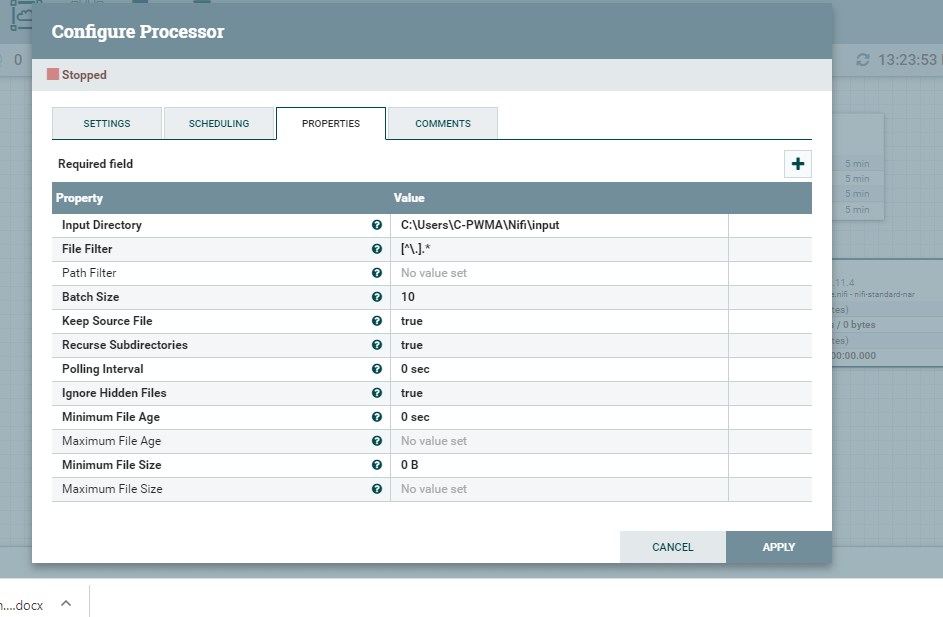


Create a sample input folder in your local and add a file that contains json data and create a output folder



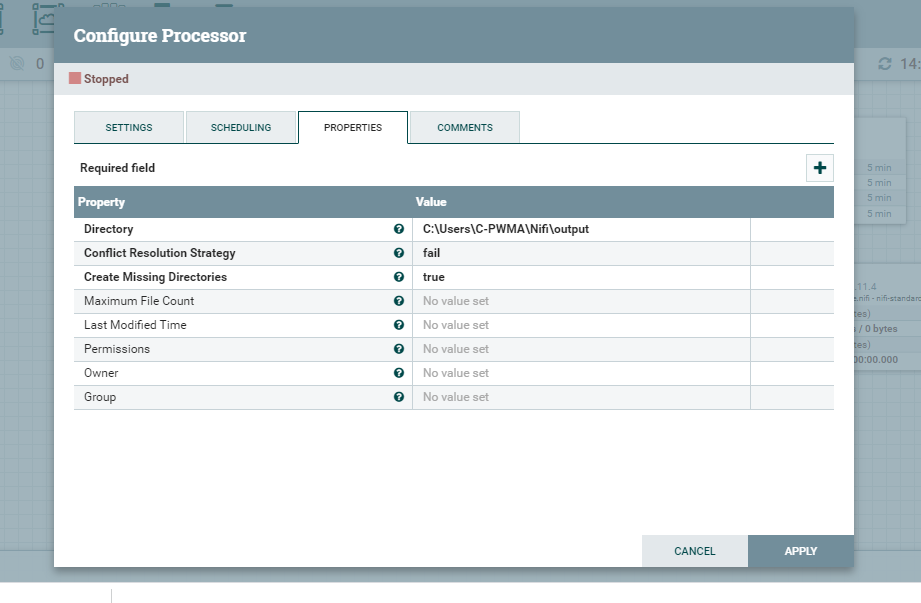


Select GetFile processor and configure it with input folder directory



Connect GetFile processor to PublishKafkaRecord processor

Configure PutFile Processor with output folder directory :



Connect PutFile Processor to ConsumeKafkaRecord processor.

Start all the processors and check in the Output folder whether we are able to read the file,Publish to eventstreams topic and consume it.

**Consume using Java and Springboot**

Write a springboot application for consuming : Refer the following tutorial:

<https://developer.ibm.com/technologies/java/tutorials/use-spring-kafka-to-access-an-event-streams-service/>

Write a sample java program to consume:

<dependency>  
<groupId>org.apache.kafka</groupId>  
<artifactId>kafka-clients</artifactId>  
<version>2.4.1</version>  
</dependency>

package com.example.cf;

​

import java.util.ArrayList;

import java.util.List;

import java.util.Properties;

​

import org.apache.kafka.clients.consumer.ConsumerRecord;

import org.apache.kafka.clients.consumer.ConsumerRecords;

import org.apache.kafka.clients.consumer.KafkaConsumer;

import org.omg.CORBA.RepositoryIdHelper;

​

import com.google.gson.JsonObject;

​

public class Consumer {

public static JsonObject main(JsonObject args) {

JsonObject response = new JsonObject();

Properties props = new Properties();

props.put("bootstrap.servers",

"broker-2-xgf5bmdv54bh7y43.kafka.svc03.us-south.eventstreams.cloud.ibm.com:9093,broker-5-xgf5bmdv54bh7y43.kafka.svc03.us-south.eventstreams.cloud.ibm.com:9093,broker-3-xgf5bmdv54bh7y43.kafka.svc03.us-south.eventstreams.cloud.ibm.com:9093,broker-4-xgf5bmdv54bh7y43.kafka.svc03.us-south.eventstreams.cloud.ibm.com:9093,broker-1-xgf5bmdv54bh7y43.kafka.svc03.us-south.eventstreams.cloud.ibm.com:9093,broker-0-xgf5bmdv54bh7y43.kafka.svc03.us-south.eventstreams.cloud.ibm.com:9093");

props.put("sasl.mechanism", "PLAIN");

props.put("security.protocol", "SASL\_SSL");

// props.put("ssl.protocol","TLSvl.2");

props.put("sasl.jaas.config",

"org.apache.kafka.common.security.plain.PlainLoginModule required username=token password= iNFme54dbODJ4LGprXigKCX8ykmABNFvG8zYMfkEleIX;");

props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

props.put("group.id", "test-group");

KafkaConsumer kafkaConsumer = new KafkaConsumer(props);

List topics = new ArrayList();

topics.add("topic-name");

kafkaConsumer.subscribe(topics);

int pollTimeout = 1000;

try {

ConsumerRecords<String, String> records = kafkaConsumer.poll(1000);

for (ConsumerRecord<String, String> record : records) {

System.out.println(record.offset() + ": " + record.value());

}

​

response.addProperty("message", "Consumer successfully received messge from Event Stream.");

} catch (Exception e) {

response.addProperty("message", e.getMessage());

e.printStackTrace();

}

​

return response;

​

}

​

}