Article
Michael Braam · Aug 17, 2022 12m read

Leveraging the external messaging API in InterSystems IRIS

Being interoperable is more and more important nowadays. InterSystems IRIS 2022.1 comes with a new messaging API to communicate with event streaming platforms like Kafka, AWS SQS/SNS, JMS and RabbitMQ.

This article shows how you can connect to Kafka and AWS SQS easily.
We start with a brief discussion of the basic concepts and terms of event streaming platforms.

Event streaming platforms purpose and common terms

Event streaming platforms like Kafka or AWS SQS are capable to consume a unbound stream of events in a very high frequency and can react to events. Consumers read the data from streams for further processing. They are often used in IoT environments.

Common terms in this arena are:

- **Topic/Queue**, the place where data is stored
- **Producer**, creates and sends data (events, messages) to a topic or queue
- **Consumer**, reads events/messages from one or more topics or queues
- **Publish/Subscribe**, producers send data to a queue/topic (publish), consumers subscribe to a topic/queue and get automatically notified if new data arrives
- **Polling**, consumers have to actively poll a topic/queue for new data

Why are they used?

- Decoupling of producers and consumers
- Highly scalable for real time data

Do I really need them? As a InterSystems IRIS developer probably not, but you are not alone...

The external messaging API

The new API classes are located in the ***%External.Messaging*** package. It contains generic ***Client-***, ***Settings-*** and ***Message*** classes. The specialized classes for Kafka, AWS SQS/SNS, JMS, RabbitMQ are subclasses of these generic classes.

The basic communication flow is:

1. Create a settings object for your target platform. This is also responsible for the authentication against the target platform.
2. Create a specific client object and pass the settings object to it
3. Create a message object and send it to the target.

The following sections demonstrate how you can communicate with Kafka and AWS SQS (Simple Queue Service).

Interacting with Kafka

Let's start with a Kafka example. First we create a class which leverages the new %External Messaging API to create a topic, send and receive a message to and from Kafka.

It first creates a Kafka settings object

```
set tSettings = ##class(%External.Messaging.KafkaSettings).%New()
set tSettings.servers = $$$KAFKASERVER
set tSettings.groupId = "iris-consumer"
```

After setting the Kafka server address it sets a Kafka group id.

With these settings a Kafka client object is created:

```
set tClient = ##class(%External.Messaging.Client
).CreateKafkaClient(tSettings.ToJSON(),.tSc)
```

It then creates a topic by invoking the CreateTopic() method of the Kafka client:

```
Set tSC = tClient.CreateTopic(pTopicName,tNumberOfPartitions,tReplicationFactor)
```

Below is the full code sample:

```
Include Kafka.Settings

Class Kafka.api [ Abstract ]
{

ClassMethod CreateTopic(pTopicName As %String) As %Status
{
    #dim tSc as %Status = $$$OK
    try {
        set tSettings = ##class(%External.Messaging.KafkaSettings).%New()
        set tSettings.servers = $$$KAFKASERVER
        set tSettings.groupId = "iris-consumer"
        set tClient = ##class(%External.Messaging.Client
).CreateKafkaClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)
        Set tNumberOfPartitions = 1
        Set tReplicationFactor = 1
        Set
 tSC = tClient.CreateTopic(pTopicName,tNumberOfPartitions,tReplicationFactor)
        $$$ThrowOnError(tSC)
        $$$ThrowOnError(tClient.Close())
        }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}



}
```

After creating a topic we can send and receive messages from Kafka. The code is similiar to the above code

```
ClassMethod SendMessage(pMessage As %String, pTopic As %String) As %Status
{
    #dim tSettings as %External.Messaging.KafkaSettings
    #dim tClient as %External.Messaging.KafkaClient
    #dim tMessage as %External.Messaging.KafkaMessage
    #dim tSc as %Status = $$$OK
    try {
        set tSettings = ##class(%External.Messaging.KafkaSettings).%New()
        set tSettings.servers = $$$KAFKASERVER
        set tSettings.groupId = "iris-consumer"
        set tMessage = ##class(%External.Messaging.KafkaMessage).%New()
        set tMessage.topic = pTopic
        set tMessage.value = pMessage
        set tClient = ##class(%External.Messaging.Client
).CreateKafkaClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)
        Set producerSettings =
"{""key.serializer"":""org.apache.kafka.common.serialization.StringSerializer""}"
        $$$ThrowOnError(tClient.UpdateProducerConfig(producerSettings))
        $$$ThrowOnError(tClient.SendMessage(tMessage))
        $$$ThrowOnError(tClient.Close())

    }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}

ClassMethod ReceiveMessage(pTopicName As %String, pGroupId As %String = "iris-
consumer", Output pMessages) As %Status
{
    #dim tSettings as %External.Messaging.KafkaSettings
    #dim tClient as %External.Messaging.KafkaClient
    #dim tMessage as %External.Messaging.KafkaMessage
    #dim tSc as %Status = $$$OK
    try {
        set tSettings = ##class(%External.Messaging.KafkaSettings).%New()
        set tSettings.servers = $$$KAFKASERVER
        set tSettings.groupId = pGroupId

        set tClient = ##class(%External.Messaging.Client
).CreateKafkaClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)
        Set producerSettings =
"{""key.serializer"":""org.apache.kafka.common.serialization.StringSerializer""}"
        $$$ThrowOnError(tClient.UpdateProducerConfig(producerSettings))
        $$$ThrowOnError(tClient.ReceiveMessage(pTopicName, .pMessages))
        $$$ThrowOnError(tClient.Close())
        }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
```

```
}
```

Let's try it. I have a Kafka instance running and first we create a topic ***community*** with the ***CreateTopic*** method above:

```
EVENTS>set tSc = ##class(Kafka.api).CreateTopic("community")
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

EVENTS>write tSc
1
```

Please ignore the log4j warnings here. The method returns a status code OK. So the topic was created. Next let's send a message to this topic. To verify that the message is sent to the topic, I have a generic Kafka consumer running. This consumer listens to the topic ***community***:

```
braam@ubuntu:~/Downloads/kafka_2.13-3.1.0$ bin/kafka-console-consumer.sh --topic community  --bootstrap-server localhost:9092
```

So let's send a message to this topic. I'll send a JSON-String to it, but basically you can send any message format to a topic.

```
EVENTS>set tMessage = {"content":"Hello InterSystems Developer Community!"}

EVENTS>set tSc = ##class(Kafka.api).SendMessage(tMessage.%ToJSON(),"community")
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

EVENTS>write tSc
1
```

Let's check if the consumer received the message:

```
braam@ubuntu:~/Downloads/kafka_2.13-3.1.0$ bin/kafka-console-consumer.sh --topic community  --bootstrap-server localhost:9092
{"content":"Hello InterSystems Developer Community!"}
```

The message was successfully received by the consumer.

Receiving messages and deleting topics is similiar to the above sample. Below is the full sample implementation. The include file ***Kafka.settings*** only contains a macro definition: ***#define KAFKASERVER <Kafka server location and port>***.

```
Include Kafka.Settings

Class Kafka.api [ Abstract ]
{

ClassMethod CreateTopic(pTopicName As %String) As %Status
{
    #dim tSc as %Status = $$$OK
    try {
        set tSettings = ##class(%External.Messaging.KafkaSettings).%New()
        set tSettings.servers = $$$KAFKASERVER
        set tSettings.groupId = "iris-consumer"
```

```
        set tClient = ##class(%External.Messaging.Client
).CreateKafkaClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)
        Set tNumberOfPartitions = 1
        Set tReplicationFactor = 1
        Set
 tSC = tClient.CreateTopic(pTopicName,tNumberOfPartitions,tReplicationFactor)
        $$$ThrowOnError(tSC)
        $$$ThrowOnError(tClient.Close())
        }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}

ClassMethod DeleteTopic(pTopicName As %String) As %Status
{
    #dim tSc as %Status = $$$OK
    try {
        set tSettings = ##class(%External.Messaging.KafkaSettings).%New()
        set tSettings.servers = $$$KAFKASERVER
        set tSettings.groupId = "iris-consumer"
        set tClient = ##class(%External.Messaging.Client
).CreateKafkaClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)
        Set tNumberOfPartitions = 1
        Set tReplicationFactor = 1
        Set tSC = tClient.DeleteTopic(pTopicName)
        $$$ThrowOnError(tSC)
        $$$ThrowOnError(tClient.Close())
        }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}

ClassMethod SendMessage(pMessage As %String, pTopic As %String) As %Status
{
    #dim tSettings as %External.Messaging.KafkaSettings
    #dim tClient as %External.Messaging.KafkaClient
    #dim tMessage as %External.Messaging.KafkaMessage
    #dim tSc as %Status = $$$OK
    try {
        set tSettings = ##class(%External.Messaging.KafkaSettings).%New()
        set tSettings.servers = $$$KAFKASERVER
        set tSettings.groupId = "iris-consumer"
        set tMessage = ##class(%External.Messaging.KafkaMessage).%New()
        set tMessage.topic = pTopic
        set tMessage.value = pMessage
        set tClient = ##class(%External.Messaging.Client
).CreateKafkaClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)
        Set producerSettings =
"{""key.serializer"":""org.apache.kafka.common.serialization.StringSerializer""}"
        $$$ThrowOnError(tClient.UpdateProducerConfig(producerSettings))
```

```
        $$$ThrowOnError(tClient.SendMessage(tMessage))
        $$$ThrowOnError(tClient.Close())

    }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}

ClassMethod ReceiveMessage(pTopicName As %String, pGroupId As %String = "iris-
consumer", Output pMessages) As %Status
{
    #dim tSettings as %External.Messaging.KafkaSettings
    #dim tClient as %External.Messaging.KafkaClient
    #dim tMessage as %External.Messaging.KafkaMessage
    #dim tSc as %Status = $$$OK
    try {
        set tSettings = ##class(%External.Messaging.KafkaSettings).%New()
        set tSettings.servers = $$$KAFKASERVER
        set tSettings.groupId = pGroupId

        set tClient = ##class(%External.Messaging.Client
).CreateKafkaClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)
        Set producerSettings =
"{""key.serializer"":""org.apache.kafka.common.serialization.StringSerializer""}"
        $$$ThrowOnError(tClient.UpdateProducerConfig(producerSettings))
        $$$ThrowOnError(tClient.ReceiveMessage(pTopicName, .pMessages))
        $$$ThrowOnError(tClient.Close())
        }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}

}
```

Interacting with AWS SQS

How would you communicate with AWS SQS (Simple Queue Service)?
The basic procedure is pretty similiar. But AWS requires authentication and AWS doesn't use the term topic. They talk about queues. You can send a message to a queue and consumers can receive messages from one or more queues.

Similar to my api class above I've created something for AWS SQS.

```
Class AWS.SQS.api [ Abstract ]
{

ClassMethod SendMessage(pMessage As %String, pQueue As %String) As %Status
```

```
{
    #dim tSettings as %External.Messaging.SQSSettings
    #dim tMessage as %External.Messaging.SQSMessage
    #dim tClient as %External.Messaging.SQSClient
    #dim tSc as %Status = $$$OK
    try {
        $$$ThrowOnError(##class(AWS.Utils).GetCredentials(.tCredentials))
        set tSettings = ##class(%External.Messaging.SQSSettings).%New()
        set tSettings.accessKey = tCredentials("aws_access_key_id")
        set tSettings.secretKey = tCredentials("aws_secret_access_key")
        set tSettings.sessionToken = tCredentials("aws_session_token")
        set tSettings.region = "eu-central-1"
        set tMessage = ##class(%External.Messaging.SQSMessage).%New()
        set tMessage.body = pMessage
        set tMessage.queue = pQueue

        set tClient = ##class(%External.Messaging.Client
).CreateSQSClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)
        $$$ThrowOnError(tClient.SendMessage(tMessage))

    }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}

ClassMethod ReceiveMessage(pQueueName As %String, Output pMessages) As %Status
{
    #dim tSettings as %External.Messaging.SQSSettings
    #dim tClient as %External.Messaging.SQSClient
    #dim tSc as %Status = $$$OK
    try {
        $$$ThrowOnError(##class(AWS.Utils).GetCredentials(.tCredentials))
        set tSettings = ##class(%External.Messaging.SQSSettings).%New()
        set tSettings.accessKey = tCredentials("aws_access_key_id")
        set tSettings.secretKey = tCredentials("aws_secret_access_key")
        set tSettings.sessionToken = tCredentials("aws_session_token")
        set tSettings.region = "eu-central-1"
        set tClient = ##class(%External.Messaging.Client
).CreateSQSClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)
        $$$ThrowOnError(tClient.ReceiveMessage(pQueueName, .pMessages))

    }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}

ClassMethod DeleteMessage(pQueueName As %String, pReceiptHandle As %String) As
%Status
{
    #dim tSettings as %External.Messaging.SQSSettings
    #dim tClient as %External.Messaging.SQSClient
```

```
    #dim tSc as %Status = $$$OK
    try {
        $$$ThrowOnError(##class(AWS.Utils).GetCredentials(.tCredentials))
        set tSettings = ##class(%External.Messaging.SQSSettings).%New()
        set tSettings.accessKey = tCredentials("aws_access_key_id")
        set tSettings.secretKey = tCredentials("aws_secret_access_key")
        set tSettings.sessionToken = tCredentials("aws_session_token")
        set tSettings.region = "eu-central-1"
                set tClient = ##class(
%External.Messaging.Client).CreateSQSClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)
        $$$ThrowOnError(tClient.DeleteMessage(pQueueName, pReceiptHandle))

    }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}

ClassMethod CreateQueue(pQueueName As %String) As %Status
{
    #dim tSettings as %External.Messaging.SQSSettings
    #dim tClient as %External.Messaging.SQSClient
    #dim tSQSSettings as %External.Messaging.SQSQueueSettings
    #dim tSc as %Status = $$$OK
    try {
        $$$ThrowOnError(##class(AWS.Utils).GetCredentials(.tCredentials))
        set tSettings = ##class(%External.Messaging.SQSSettings).%New()
        set tSettings.accessKey = tCredentials("aws_access_key_id")
        set tSettings.secretKey = tCredentials("aws_secret_access_key")
        set tSettings.sessionToken = tCredentials("aws_session_token")
        set tSettings.region = "eu-central-1"
        set tClient = ##class(%External.Messaging.Client
).CreateSQSClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)

        set tSQSSettings = ##class(%External.Messaging.SQSQueueSettings).%New()

        $$$ThrowOnError(tClient.CreateQueue(pQueueName,tSQSSettings))


    }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}

ClassMethod DeleteQueue(pQueueName As %String) As %Status
{
    #dim tSettings as %External.Messaging.SQSSettings
    #dim tClient as %External.Messaging.SQSClient
    #dim tSQSSettings as %External.Messaging.SQSQueueSettings
    #dim tSc as %Status = $$$OK
    try {
        $$$ThrowOnError(##class(AWS.Utils).GetCredentials(.tCredentials))
```

```
        set tSettings = ##class(%External.Messaging.SQSSettings).%New()
        set tSettings.accessKey = tCredentials("aws_access_key_id")
        set tSettings.secretKey = tCredentials("aws_secret_access_key")
        set tSettings.sessionToken = tCredentials("aws_session_token")
        set tSettings.region = "eu-central-1"
        set tClient = ##class(%External.Messaging.Client
).CreateSQSClient(tSettings.ToJSON(),.tSc)
        $$$ThrowOnError(tSc)

        $$$ThrowOnError(tClient.DeleteQueue(pQueueName))


    }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}

}
```

It contains methods for creating and deleting queues and sending and receiving messages to and from a queue.

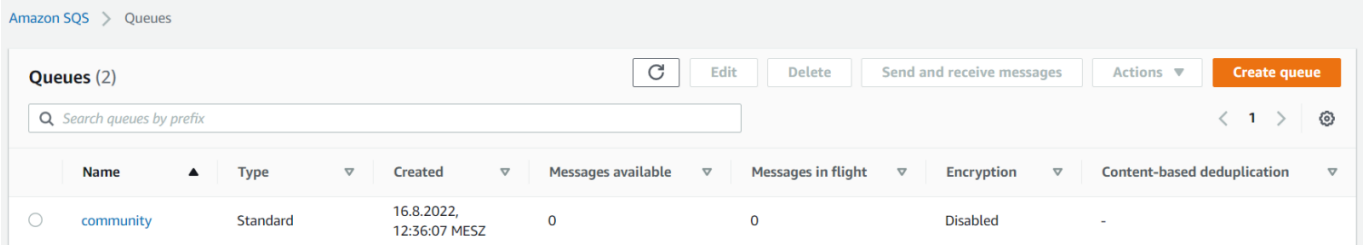
One of the key points here is how to authenticate. I didn't want to have my credentials in my code. So I created a little helper method to retrieve the credentials from my local credentials file and return it as subscripted array to use it in my api methods:

```
ClassMethod GetCredentials(Output pCredentials) As %Status
{
    #dim tSc as %Status = $$$OK
    set tFilename = "/dur/.aws/credentials"
    try {
        set tCredentialsFile = ##class(%Stream.FileCharacter).%New()
        $$$ThrowOnError(tCredentialsFile.LinkToFile(tFilename))
        // first read the header
        set tBuffer = tCredentialsFile.ReadLine()
        for i=1:1:3 {
            set tBuffer = tCredentialsFile.ReadLine()
            set pCredentials($piece(tBuffer," =",1)) = $tr($piece(tBuffer,"= ",2),
$c(13))
        }
    }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}
```

To complete this article, let's create a queue ***community*** in the AWS region "eu-central-1" (Frankfurt, Germany).

```
EVENTS>set tSc = ##class(AWS.SQS.api).CreateQueue("community")

EVENTS>write tSc
1
```

The queue has been successfully created and is visible in the AWS console for my account:

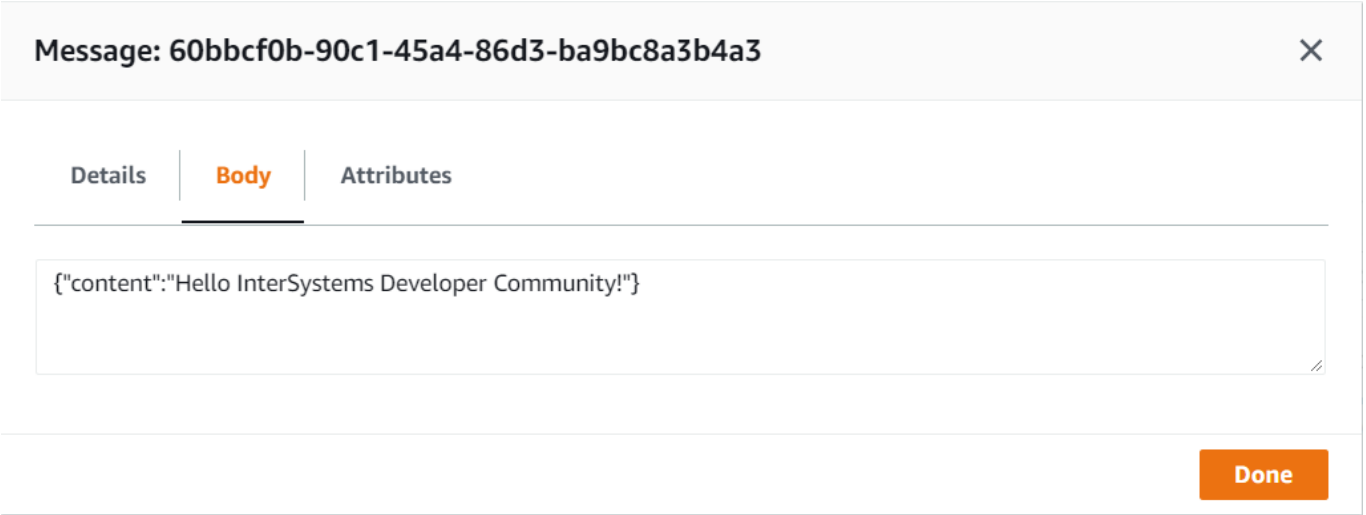


Next, let's send a message to this queue:

```
EVENTS>set tMessage = {"Content:":"Hello InterSystems developper community!"}

EVENTS>set tSc = ##class(AWS.SQS.api).SendMessage(tMessage.%ToJSON(),"community")

EVENTS>write tSc
1
```

The method call returns 1. So the message has been successfully sent.

Finally let's poll the queue from the AWS console:



The message has been successfully delivered to the queue.

Conclusion

The external messaging api in InterSystems IRIS 2022.1 makes it really simple to communicate with event streaming platforms.
Hope you find this useful.

#API #AWS #Deployment #InterSystems IRIS #InterSystems IRIS for Health