Article
Yaron Munz · Jul 12, 2022 5m read

IRIS Embedded Python with Azure Service Bus (ASB) use case

Overview

We started to use Azure Service Bus (ASB) as an enterprise messaging solution 3 years ago. It is being used to publish and consume data between many applications in the organization. Since the data flow is complex, and one application ' s data is usually needed in multi applications the " publisher " ---> " multiple subscribers " model was a great fit. The ASB usage in the organization is dozens of millions of messages per day, while IRIS platform is having around 2-3 million messages/day.

The Problem with ASB

When we started with the ASB integration, we found that the AMQP protocol is not an "out of the box" for IRIS deployment, so we were looking for an alternative solution to be able to communicate with the ASB.

The solution

We developed a local windows service (in .NET and Swagger) that was doing the actual communication to the ASB. It was installed at the same machine with IRIS. We sent the messages to ASB to this local windows service (using: localhost and doing a REST API) and that service did the rest. This solution was working fine for few years, but monitoring the traffic, debugging the "lost" messages and measuring the overall performance was very difficult. The fact that we had this local windows service as " a man in the middle " is always not the best architecture.

IRIS Embedded Python Early Access Program (EAP)

I have been asked (or volunteered I can ' remember) to participate in the Early Release Program (ERP) for the Embedded Python. This was very exciting for me, to be able to test new features, give feedback to the development team. It also felt very good to be active in participating and influencing with this product development. At this stage we started to test the Embedded Python for the ERP and decided to check if we can lavarge Embedded Python to do solve " real " problems. Taking the " IRIS to ASB direct connectivity " for this purpose was very natural.

The POC (proof of concept)

We started to code the solution and found that using the Microsoft ASB library for Python will make our life (and coding) much easier. The initial stage was to develop a function that can connect to a specific topic in ASB and received messages. This was done quite fast (1 day of development) and then we progressed to the " publish " (sending to ASB) phase.

Additional few days took us to develop the whole " envelope " for those send & receive functions: we have built the following:

- A proper incoming and outgoing " staging area " to control the flow of in/out messages
- A central mechanism to store the ASB in & out traffic to be able to have proper statistics and performance measurements
- A CSP monitoring page to enable/disable services, show statistics and give alerts on any issues

The implementation

Preliminary Configuration

Prior to the use of Microsoft ASB libraries for Python, there is a need to install them into a ..\python dedicated folder.
we choosed to use ../mge/python/ but you may use any other folder you like (initially it was an empty folder)
Those need to executed from an elevated CMD (on windows) or by using an elevated user (on Linux):
..\bin\irispip install --target ..\mgr\python asyncio
..\bin\irispip install --target ..\mgr\python azure.servicebus

Receiving messages to ASB (consuming)

We have a ClassMethod with some parameters:

- The topId - The ASB is divided to topics from which you consume messages
- subscriptionName - Azure subscription name
- connectionString - To be able to connect to the topic you are subscribed to

Note that we are using [Language = python] to indicate that this ClassMethod is written in Python (!)
ClassMethod retrieveASB(topicId As %Integer = 1, topicName As %String = "", subscriptionName As %String = "", connectionString As %String = "", debug As %Integer = 1, deadLetter As %Integer = 0) As %Integer [Language = python]

To use the ASB library, we first need to import ServiceBusClient and ServiceBusSubQueue libraries:
from azure.servicebus import ServiceBusClient,ServiceBusSubQueue

to be able to interact with IRIS (e.g. to run code) we also need to:
import iris

At this stage we can use the ASB libraries:
with ServiceBusClient.fromconnectionstring(connectionString) as client:
with client.getsubscriptionreceiver(topicName, subscriptionName, maxwaittime=1, subqueue=subQueue) as receiver:

for msg in receiver:

At this point we have a Python object "msg" (stream) where we can pass it (as a stream of course) to IRIS and store it directly in the database:
result = iris.cls("anyIRIS.Class").StoreFrom Python(stream)

Sending messages to ASB (publishing)

We have a ClassMethod with some parameters:

- topicName - The Topic we want to publish to (here we need to pass the name not the id)
- connectionString - To be able to connect to the topic you are subscribed to
- JSONmessage - the message we want to send (publish)

Note that we are using [Language = python] to indicate that this ClassMethod is written in Python (!)
ClassMethod publishASB(topicName As %String = "", connectionString As %String = "", JSONmessage As %String = "") As %Status [Language = python]

To use the ASB library, we first need to import ServiceBusClient and ServiceBusMessage libraries:
from azure.servicebus import ServiceBusClient, ServiceBusMessage

Using the ASB Libraries is then very easy:

```
try:
  result=1

  with ServiceBusClient.fromconnectionstring(connectionString) as client:

  with client.getqueuesender(topicName) as sender:

  singlemessage = ServiceBusMessage(JSONmessage)

  sender.sendmessages(singlemessage)

  except Exception as e:

  print(e)

  result=0

  return result
```

Benefits of using direct ASB connectivity

- Much faster than using the "old alternative" of the "windows local service"
- Easy to monitor, collect statistics, debug issues
- Decommissioning the "man in the middle" (windows local service) reduces a potential "point of failure"
- Ability to manage the "dead letters" for any topic automatically and to make the ASB re-send those messages to the subscriber's topic

Credits

I would like to thank @David Satorres (our senior developer for IRIS) for his huge contribution in design, coding & testing. Without his help this project wouldn't be possible.

#Azure #Best Practices #Embedded Python #InterSystems IRIS