MQ V9.1.2 – Improved REST Messaging performance on z/OS

Tony Sharkey
Published on 27/03/2019 / *Updated on 01/04/2019*

A number of customers are considering whether the Messaging REST API is a viable alternative to using a client connection.

One concern might be the relatively high cost on z/OS of servicing a request using the mqweb server, despite the majority of the cost being eligible for offload to specialty processors.

Of course the side effect of high cost, or the number of CPU cycles required to complete a request, is a high response time (or high latency).

From V9.1.2, REST messaging now supports pooled connections to IBM MQ queue managers. This support can make a significant reduction in both cost and latency when using REST messaging.

What do I need to do to use pooled connections

Beyond installing the MQ v9.1.2 code, nothing!

By default, the `mqweb.xml` file contains some new variables:

```
<variable name="mqRestMessagingMaxPoolSize" value="20"/>
<variable name="mqRestMessagingFullPoolBehavior" value="overflow"/>
```

These attributes are described in the IBM MQ Knowledge Center.

How much difference might using pooled connections make?

When we looked at the costs associated with using the Messaging REST APIs, there were 2 areas of significance.

1. The TLS handshake at the start of the connection.
2. The need for the mqweb server to start a connection to the queue manager for each instance.

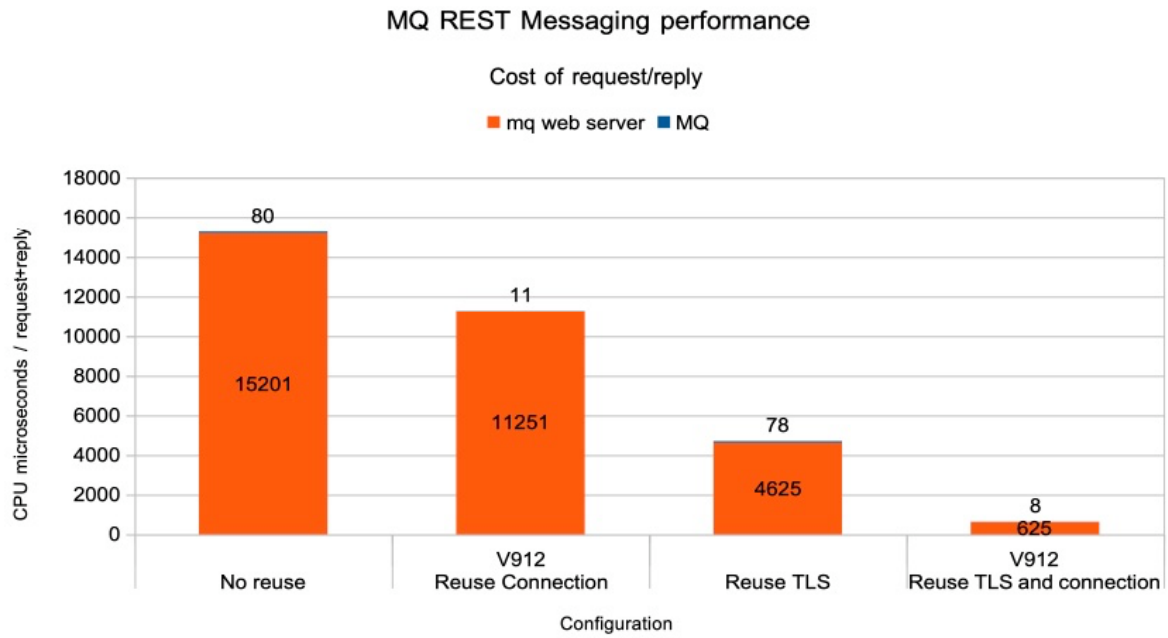
Depending upon the way your application is connecting to the Liberty server, you may be able to re-use the TLS connection – which can make a significant difference in the costs.

Even if you are unable to re-use the TLS connection, you can still benefit from using the pooled connections to your queue manager.

In our tests, we configured a set of clients running on a distributed client machine that would make a request for data consisting of a 1KB reply payload from a z/OS queue manager. For these measurements we used 4 configurations:

1. Baseline, with no re-use of TLS connections nor pooled connections to queue manager.
2. Re-use pooled connections to queue manager.
3. Re-use TLS connection.
4. Re-use TLS connection and pooled connection to queue manager.

The following chart shows the impact to the cost on z/OS of the 4 configurations.



REST Messaging on z/OS – Transaction cost with 10 clients making request of 10 bytes and receiving reply of 1KB payload

The costs shown are in CPU microseconds, i.e. the "no reuse" configuration cost **15.2 CPU milliseconds** per request in the mqweb server and **80 CPU microseconds** in the MQ queue manager address space.

Being able to re-use the TLS connection made the largest difference – in our measurements this equated to approximately **11 CPU milliseconds**.

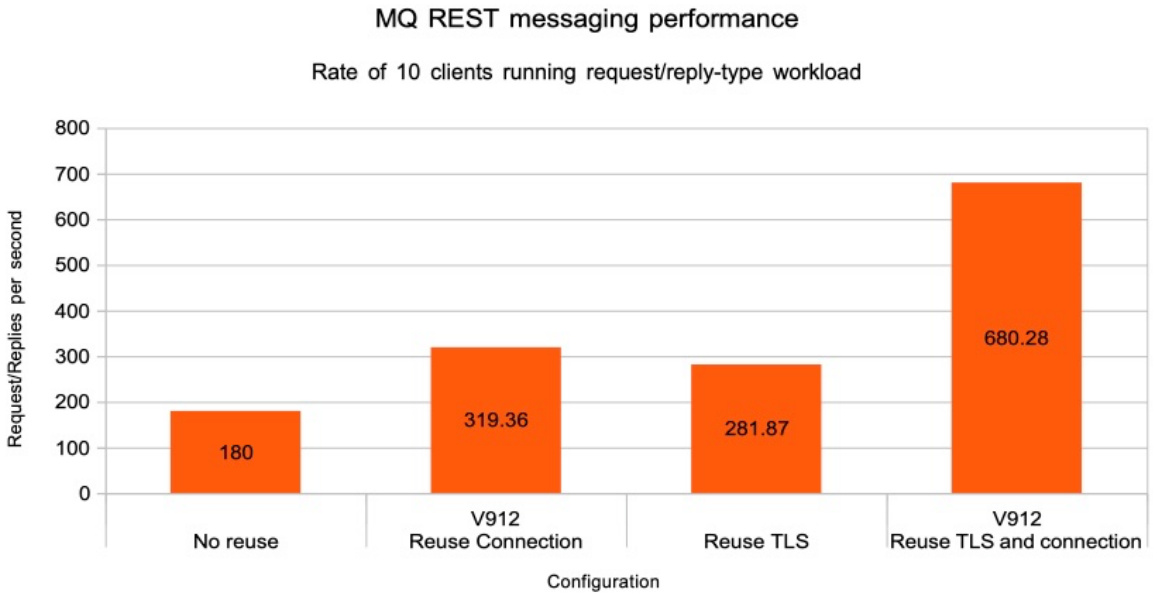
The ability to re-use the connection to the queue manager however also results in a significant cost reduction, in our measurements this was of the order of **4 CPU milliseconds** per request/reply in the mqweb server address space.

In addition, there is a small reduction in the costs in the MQ queue manager due to a number of reasons:

- Re-using the connection – the MQCONN is a relatively expensive API.
- As we were running with class(3) accounting enabled, each new connection would create a new SMF type 116 record. This meant that for a workload of 10,000 requests,

reusing the MQ connection reduced the number of SMF 116 records from 10,000 to less than 20.

Re-use, both of the TLS connection and the MQ queue manager connection, makes a difference to the achievable transaction rate too, as indicated in the following chart:



REST Messaging on z/OS – Transaction rate achieved with 10 clients making request of 10 bytes and receiving reply of 1KB payload

Being able to re-use both the TLS connection and the MQ connection saw a **3.75 times improvement in throughput**, which coupled with a reduction in cost to **just 4% of the initial configuration**, may make a huge difference when deciding whether MQ REST messaging is appropriate for your use!