Using AWS CloudWatch to monitor IBM MQ queue managers

Mark E Taylor |Aug 25 2016 Updated

In this blog entry I'm going to show how you can monitor MQ queue managers using Amazon's CloudWatch service.

**Keeping the demo simple**

For the purposes of this demonstration, I stayed away from running any actual code on AWS. That avoided having to deploy programs into images, or configure much security. The queue manager runs on my local Linux workstation, as does the grafana service. The data is pushed to, and retrieved from, CloudWatch running in AWS but that's all. In reality, someone using CloudWatch is likely to want to include the monitoring collection program in an EC2 image to run alongside the queue manager in that instance. And that in turn will require a security role to be configured with that instance.

**Security configuration**

For my simple setup, I had to have accessible credentials so that I could write to CloudWatch. Since the MQ service runs under the mqm account, I put a copy of a credentials file containing the AWS\_ACCESS\_KEY and AWS\_SECRET\_KEY into ~mqm/.aws/credentials. That file may also contain the user's default AWS region (eg "us-west-2") for connection; if not provided there, the region can be given as a parameter on the **mq\_aws** command line. That same file was also put under the home directory of the grafana userid so it can use CloudWatch as a data source.

**The MQ program**

All of the MQ programs for monitoring are in the same [github repository](https://github.com/ibm-messaging/mq-golang). Cloning that repository gives copies of all of the MQ source code, configuration scripts, and example Grafana dashboards. You will need to get a copy of an error-logging package using

 go get -u github.com/Sirupsen/logrus

To build the CloudWatch collector, **mq\_aws**, you also need the AWS client packages:

 go get -u github.com/aws/aws-sdk-go/service

You can then build the collection program with

 go build -o mq\_aws cmd/mq\_aws/\*.go

Just as for the other monitoring programs, configure the queue manager through the MQSC and shell scripts in that directory, so that the main program is started as an MQ Service.

**Looking at CloudWatch data**

Once the monitoring program is running correctly (you can look for errors in /var/mqm/errors/aws.out if you keep the same MQSC script), the metrics are sent to CloudWatch.

When you log into AWS and go to the CloudWatch Management Console you can see them in the Custom Metrics section, under the IBM/MQ namespace:



From there, metrics can be selected. You can see two filters, one for the queue-related metrics (which of course also includes the queue manager name as a field) and one for the queue-manager-wide metrics:



Selecting some of these metrics allows creation of a dashboard in CloudWatch.



**Using Grafana**

The CloudWatch dashboard is not very sophisticated graphically, but it does at least allow graphs to be created and shown. Better as a visualisation interface is Grafana. And for that, I created a dashboard similar to that used for all the other databases in this series of articles. The biggest difference is that there does not seem to be a way to directly query using wildcards for the queue or queue manager names ("dimensions" in CloudWatch terms); using Grafana templates might be a way to deal with that. The queries used in my sample dashboard explicitly name the queues to be monitored. Also, refresh times are set to be much less frequent because of the rates used in CloudWatch, and the pricing model that charges for queries and updates.



**Summary**

This series of articles has shown how MQ V9 can feed data into a variety of monitoring solutions, including those commonly-used for cloud deployments of queue managers.

I would welcome feedback on these tools. Please leave any feedback here, or in the [GitHub issue tracker](https://github.com/ibm-messaging/mq-golang/issues), whether bugs, enhancements, or thoughts on the value of the monitoring.