Managing queue manager startup and shutdown on Linux with

systemd

[Arthur Barr](javascript:;" \o "Arthur Barr) |Aug 11 2016 Updated

In this blog entry, we'll take a look at how to use [systemd](https://freedesktop.org/wiki/Software/systemd/) to start MQ queue managers at system boot time, and stop them when the system shuts down.  systemd has been widely adopted across many Linux distributions, including Red Hat Enterprise Linux V7, Ubuntu 15.04 onwards, SUSE Linux Enterprise Server V12 and many more.  systemd can be described as follows:

*"systemd is a suite of basic building blocks for a Linux system. It provides a system and service manager that runs as PID 1 and starts the rest of the system. systemd... offers on-demand starting of daemons, keeps track of processes using Linux control groups, ...and implements an elaborate transactional dependency-based service control logic."*  
- <https://freedesktop.org/wiki/Software/systemd/>

I don't claim to be a systemd expert, but these instructions came about from a need to run a queue manager at boot time, and seem to work well for me.  If you have any feedback, please add a comment to this blog entry.

**Creating a simple systemd service**

In order to run as a systemd service, you need to create a "unit" file.  The following is a simple unit file for running MQ, which should be saved in /etc/systemd/system/qm1.service

[Unit]

Description=IBM MQ V8 queue manager qm1

After=network.target

[Service]

ExecStart=/opt/mqm/bin/strmqm qm1

ExecStop=/opt/mqm/bin/endmqm -w qm1

Type=forking

User=mqm

Group=mqm

KillMode=none

LimitNOFILE=10240

LimitNPROC=4096

Update: Added LimitNPROC.

Let's break down the key parts of this file:

* ExecStart and ExecStop give the main commands to start and stop the queue manager service.
* Type=forking tells systemd that the strmqm command is going to fork to another process, so systemd shouldn't worry about the strmqm process going away.
* KillMode=none tells systemd not to try sending SIGTERM or SIGKILL signals to the MQ processes, as MQ will ignore these if they are sent.
* LimitNOFILE and LimitNPROC are needed because systemd services are not subject to the usual PAM-based limits (for example, in /etc/security/limits.conf), so we need to make sure MQ can have enough open files and processes.
* After=network.target makes sure that MQ is only started after the network stack is available.  Note, that this doesn't necessarily mean that IP addresses are available, but just the network stack is up.  This option is particularly important because it affects the shutdown sequence, and makes sure that the MQ service is stopped before the network is stopped.  See [here](https://www.freedesktop.org/wiki/Software/systemd/NetworkTarget/) for a good explanation of this.

In order to try out the service, you first need to tell systemd to reload its configuration, which you can do with the following command:

systemctl daemon-reload

Assuming you've already created a queue manager called "qm1", you can now start it as follows:

systemctl start qm1

You can then see the status of the systemd service as follows:

systemctl status qm1

This should show something like this:

● qm1.service - IBM MQ V8 queue manager qm1

Loaded: loaded (/etc/systemd/system/qm1.service; static; vendor preset: disabled)

Active: active (running) since Wed 2016-04-13 10:06:51 EDT; 3s ago

Process: 2351 ExecStart=/opt/mqm/bin/strmqm qm1 (code=exited, status=0/SUCCESS)

Main PID: 2354 (amqzxma0)

CGroup: /system.slice/qm1.service

├─2354 /opt/mqm/bin/amqzxma0 -m qm1 -u mqm

├─2359 /opt/mqm/bin/amqzfuma -m qm1

├─2364 /opt/mqm/bin/amqzmuc0 -m qm1

├─2379 /opt/mqm/bin/amqzmur0 -m qm1

├─2384 /opt/mqm/bin/amqzmuf0 -m qm1

├─2387 /opt/mqm/bin/amqrrmfa -m qm1 -t2332800 -s2592000 -p2592000 -g5184000 -c3600

├─2398 /opt/mqm/bin/amqzmgr0 -m qm1

├─2410 /opt/mqm/bin/amqfqpub -mqm1

├─2413 /opt/mqm/bin/runmqchi -m qm1 -q SYSTEM.CHANNEL.INITQ -r

├─2414 /opt/mqm/bin/amqpcsea qm1

├─2415 /opt/mqm/bin/amqzlaa0 -mqm1 -fip0

└─2418 /opt/mqm/bin/amqfcxba -m qm1

Apr 13 10:06:50 rhel-mq.novalocal systemd[1]: Starting IBM MQ V8 queue manager qm1...

Apr 13 10:06:50 rhel-mq.novalocal strmqm[2351]: WebSphere MQ queue manager 'qm1' starting.

Apr 13 10:06:50 rhel-mq.novalocal strmqm[2351]: The queue manager is associated with installation 'Installation1'.

Apr 13 10:06:50 rhel-mq.novalocal strmqm[2351]: 5 log records accessed on queue manager 'qm1' during the log replay phase.

Apr 13 10:06:50 rhel-mq.novalocal strmqm[2351]: Log replay for queue manager 'qm1' complete.

Apr 13 10:06:50 rhel-mq.novalocal strmqm[2351]: Transaction manager state recovered for queue manager 'qm1'.

Apr 13 10:06:51 rhel-mq.novalocal strmqm[2351]: WebSphere MQ queue manager 'qm1' started using V8.0.0.4.

Apr 13 10:06:51 rhel-mq.novalocal systemd[1]: Started IBM MQ V8 queue manager qm1.

You can see that systemd has identified `amqzxma0` as the main queue manager process.  You will also spot that there is a Linux [control group (cgroup)](https://en.wikipedia.org/wiki/Cgroups) for the queue manager.  The use of cgroups allows you to specify limits on memory and CPU for your queue manager.  You could of course do this without systemd, but it's helpfully done for you now.  This doesn't constrain your processes by default, but gives you the option to easily apply limits to CPU and memory in the future.  Note that you can still run MQ commands like runmqsc as normal.  If you run strmqm qm1, you will start the queue manager as normal, as your current user, in your user cgroup.  It is perhaps better to get in the habit of running systemctl start qm1` instead, to make sure you're using your configured settings, and running in the correct cgroup.

**Templated service**

If you have multiple queue managers, it would be nice to not duplicate the service unit file many times.  You can create templated services in systemd to do this.  Firstly, stop your qm1 service using the following command:

systemctl stop qm1

Next, rename your unit file to `mq@.service`, and edit the file to replace all instances of the queue manager name with "%I".  After doing a daemon-reload again, you can now start your "qm1" queue manager by running the following command:

systemctl start mq@qm1

The full name of the service created will be "mq@qm1.service", and you can use it just as before.

As it stands, you are supplying the name of the queue manager on the command line, so what about system startup?  The non-templated version is an active unit, so would get started up automatically, but with the templated version, the trick is to add an "[Install]" section to your unit file, giving the following:

[Unit]

Description=IBM MQ V8 queue manager %I

After=network.target

[Service]

ExecStart=/opt/mqm/bin/strmqm %I

ExecStop=/opt/mqm/bin/endmqm -w %I

Type=forking

User=mqm

Group=mqm

KillMode=none

LimitNOFILE=10240

LimitNPROC=4096

[Install]

WantedBy=multi-user.target

After doing a daemon-reload, you can now "enable" a new service instance with the following command:

systemctl enable mq@qm1

You can, of course, run this many times, once for each of your queue managers.  Using the "enable" command causes systemd to create symlinks on the filesystem for your particular service instances.  In this case, we've said that the "multi-user" target (kind of like the old "runlevel 3"), should "want" our queue managers to be running.  This basically means that when the system boots into a multi-user mode, the start up of our queue managers should be initiated.  They will still be subject to the "After" rule we defined earlier.

**Summary**

systemd is a powerful set of tools, and we've really only scratched the surface here.  In this blog entry, we've made the first useful step of ensuring that queue managers are hooked correctly into the lifecycle of the server it's running on.  Doing this is very important for failure recovery.  Using systemd instead of the old-style init.d scripts should help improve your server's boot time, as well as providing additional benefits such as the use of cgroups for finer-grained resource control.  It's possible to set up more sophisticated dependencies for your units, if (say) you wanted to ensure your client applications were always started after the queue manager, or you wanted to wait for a mount point to become available.  Be careful with adding too many dependencies though, as this could slow down your boot time.

I'm sure there are many of you, dear blog readers, who can recommend further changes or tweaks that helped in your environment.  Please share your thoughts in the comments.