
The MQ-Ansible collection helps simplify the installation and configuration of IBM MQ. With this collection, you can automate tasks, streamline deployments, and ensure consistency across environments. This is particularly useful in complex enterprise environments where manual configuration can be time-consuming and prone to errors.

In this blog, we will explore a foundational IBM MQ set up that you can do from scratch using the MQ-Ansible collection: a Sender-Receiver configuration. This architecture allows you to enable communication between two separate queue managers. It's a basic yet essential setup that forms the foundation of many enterprise messaging architectures. You can find the playbooks and files mentioned this blog here.

Step 1: Install the MQ-Ansible Collection

First, you'll need to have the MQ-Ansible collection installed. You can do this by running the following command:

ansible-galaxy collection install ibm_messaging.ibmmq

Ensure you have Ansible installed and configured on your system, with the minimum version to use ansible galaxy.

Step 2: Define Your Inventory File

Create an inventory file that lists the servers where you'll be deploying MQ. For example:

### inventory.ini

[sender]

sender ansible_host=<YOUR_HOSTNAME> ansible_ssh_user:<YOUR_USER>

[receiver]

receiver ansible_host=<YOUR_HOSTNAME> ansible_ssh_user:<YOUR_USER>

Step 3: Copy across SSH keys

To avoid writing passwords to your inventory file, make sure you have copied your local (where you are running ansible) ssh keys to your target hosts.

Step 4: Use the demo playbooks

For this demo, there are 2 playbooks. setup.yml is for installing MQ, using the MQ-Ansible roles, and sdr-rcvr.yml for creating the SENDER and RECEIVER queue managers with their respective configuration. They are both run with the demo.yml playbook.

The MQSC files provided are the following:

#### sender-config.mqsc

* Define transmit queue (make sure name matches target QM)

DEFINE QLOCAL(RECEIVER) USAGE (XMITQ)

* Define sender channel

DEFINE CHANNEL(SDR.TO.RCVR) CHLTYPE(SDR) CONNAME('<RCVR.QMGR.IP>(1414)') XMITQ(RECEIVER)

* Define remote queue

DEFINE QREMOTE(REMOTE.Q) RNAME(RCVR.TARGET.Q) RQMNAME(RECEIVER)

* Start channel

START CHANNEL(SDR.TO.RCVR)

#### receiver-config.mqsc

* Define and start receiver-side listener

DEFINE LISTENER('RCVR.LISTENER.TCP') TRPTYPE(TCP) PORT(1414) CONTROL(QMGR) REPLACE

START LISTENER('RCVR.LISTENER.TCP') IGNSTATE(YES)

* Use a different dead letter queue, for undeliverable messages

DEFINE QLOCAL('DEV.DEAD.LETTER.QUEUE') REPLACE

ALTER QMGR DEADQ('DEV.DEAD.LETTER.QUEUE')

* Define target queue in receiver:

DEFINE QLOCAL('RCVR.TARGET.Q') REPLACE

* Define receiver channel

DEFINE CHANNEL(SDR.TO.RCVR) CHLTYPE(RCVR)

* Turning security off - for demo purposes

ALTER QMGR CHLAUTH(DISABLED)

ALTER QMGR CONNAUTH(' ')

REFRESH SECURITY TYPE(CONNAUTH)

The <RCVR.QMGR.IP> placeholder in sender-config.mqsc will be replaced with the receiver's IP address with the configuration playbook sdr-rcvr.yml.

Step 5: Run the Playbooks

ansible-playbook demo.yml -i ./inventory.ini -e 'ibmMqLicence=accept'

Step 6: Verify the Configuration

Log into the target machines and check that the queue managers have been created, and that the channel is running. Here we display the current channel statuses of the SENDER queue manager.

runmqsc sender

display chs(*)

Output:

AMQ8417I: Display Channel Status details.

CHANNEL(SDR.TO.RCVR) CHLTYPE(SDR)

CONNAME(XXXXXX(1414)) CURRENT

RQMNAME(RECEIVER) STATUS(RUNNING)

SUBSTATE(MQGET) XMITQ(RECEIVER)

Now you can send messages to your remote queue manager. An example using the amqsput sample application:

On the Sender host

cd /opt/mqm/samp/bin

./amqsput REMOTE.Q SENDER

<send messages>

You can get or browse the messages on the receiver end with your favorite application.

Why Automate with MQ-Ansible Collection?

The MQ-Ansible collection eliminates the manual effort involved in setting up IBM MQ. It ensures that your configurations are consistent, repeatable, and easy to scale. Whether you're managing a simple Sender-Receiver setup or a complex clustered architecture, automation helps reduce errors and save time.

If you would like see a live demo of this blog, sign up to the February MQ-Ansible community call here.