Using Ansible with IBM MQ on z/OS

Introduction

Ansible is an automation tool. In today's lab, you will use Ansible automation to create, display, alter and delete a queue on an IBM MQ for z/OS queue manager with 4 lines of commands to your PC linux terminal. In this case, your virtualized PC will act as your ansible control node and your z/OS environment will be a managed node, meaning it is acted upon by the automation triggered by the control node.

This lab demonstrates a few of the ever-growing IBM z/OS core collection capabilities on Ansible. The IBM z/OS core collection is being developed to allow seamless integration between Ansible and z/OS.

So why is ansible a good fit for IBM Z? There are other automation tools out there, but Ansible makes a lot of sense for z/OS users because it is written in YAML, which is a human readable language. This means that there is a low barrier to entry for both seasoned system programmers and new-to-IBM-Z hires. Secondly, Ansible is agentless, meaning that it does not need to be installed on your managed nodes. You only need to have it running on your control node.

In this lab we will be using Ansible running on Ubuntu. Because Linux does not run natively on Windows, Ubuntu is able to run because Windows Subsystems for Linux is enabled on our VMWare image.

System Requirements

Setup for VMWare machines

Installation Requirements:

- Windows Subsystems for Linux

- Ubuntu linux distro
- Ansible on Ubuntu
- z/OS core collection
- Python installed – at least version 2

Setup for z/OS Environment

- Unix Systems Services
- OpenSSH
- Z Open Automation
- Python on IBM Z

****LAB START****

1.) From the start window, navigate to the Ubuntu application. Ubuntu is a Linux distro we will be using because Linux does not run natively on Windows.

2.) Once in Ubuntu, enter the command pwd and hit enter. You should see that you are in /home/dorothy. You will want your own directory for yourself, so go ahead back out of the dorothy directory with a cd .. command. Then make your own directory with the command mkdir TEAMXX

Configuring SSH

2.) From the Ubuntu terminal, enter:

$ ssh-keygen -t rsa

This command generates a key pair on your workstation's Linux filesystem. You can verify that the key pair has been installed by looking in the directory ~/.ssh . You should now see a public and private key called id_rsa.pub and id_rsa, respectively.

3.) Now we need to copy that public key to the z/OS environment.

$ ssh-copy-id user@host_IP_address

3.) This will prompt you for a password. Press enter on your terminal. This recommendation is for the simplicity of the lab and not the best practice for production environments.

4.) At this point, both your workstation and z/OS environment can connect using SSH. Test this connection by entering:

ssh user@host_IP_address

The point of the ssh connection is to bypass the requirement for a password, so if a password is requested, something went wrong. Go through the process again, regenerating the keys.

Investigating the ansible file system

5.) Once you are at the command line on Ubuntu, navigate to the C directory with the command:

```
cd /mnt/c/
```

The C drive is the mount point for sharing directories and files between the Windows PC and the Linux kernel.

6.) Now, let's explore what's in the ansible directory for this lab. The 'cd' command changes the directory to the relevant place, then the 'ls' command asks linux to list what are the contents of the ansible_mq_lab directory.

```
cd ansible_mq_lab
```

```
ls
```

7.) You will see:

ansible.cfg inventory.ini group_vars

And four playbooks

mq_define_qlocal.yml mq_display_qlocal.yml

mq_alter_qlocal.yml mq_delete_qlocal.yml

8.) How do all of these files come together to let ansible work its magic? Lets take a look! The linux 'cat' command allows us to see what is in each file from the command line. From the command line, enter:

```
cat ansible.cfg
```

By default, when you install ansible on your linux environment, a config file is created in the /etc/ansible directory. However, if you choose to specify a config file locally to your playbook, ansible will look inside the playbook's directory first.

You can see that the only thing we specify in this playbook is the path for using the correct inventory file.

9.) Now, let's see what's inside our inventory file.

```
cat inventory.ini
```

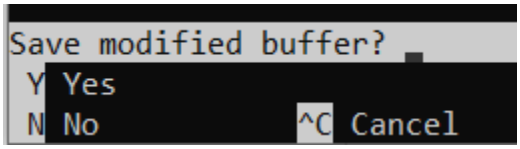
Once inside, you'll notice the inventory is the point where we specify who ansible is going to need to connect to. For our purposes, we have listed ansible's connection to the local workstation as well as its connection to a z/OS environment. Here, we also specify where ansible can find python both in our z/OS environment USS file system and on our local workstation. Ansible will need to access python in both places.

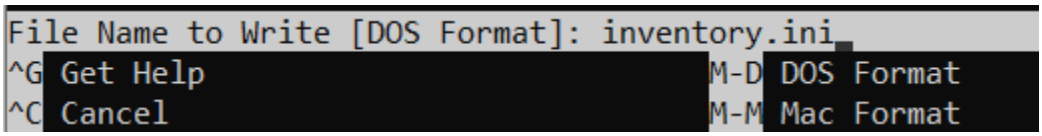
You'll also notice that for z/OS, there is a username dquincy listed. You will need to change this user name to your TEAMXX userid.

9a.) Enter `nano inventory.ini` . This command enables you to modify your inventory file within the command line interface.

9b.) Use your arrow keys to navigate your mouse to where it says `ansible_user=dquincy`. Delete the characters 'dquincy' and replace them with your appropriate team number (ex. TEAM04).

9c.) To save your newly edited inventory file, press the 'Ctrl' and 'X' keys simultaneously. This should pop up at the bottom of your command window. Select 'Y'. It will then ask you if you'd like to save the file as 'inventory.ini'. Hit the return key.





9d.) By this point, you should be back to where you started in your local_queues directory.

10.) Next, we will look at the contents of the group_vars directory and modify your queue name.

```
cd group_vars
ls
cat zos_host.yml
```

TECH TIP: The group_vars directory is recognized by ansible as being the place to store more specific variables for your specific z/OS connection. For example, if you had multiple z/OS LPARs connected to ansible, you would want them all referenced in the inventory file, but each would have its own variables resolved in its unique zos_host.yml file.

You will notice that zos_host.yml contains information about all the relevant paths once ansible connects to z/OS. For example, once ansible makes the ssh connection, it needs to know where python and ZOAU are in the z/OS USS file structure.

The zos_host.yml also specifies which queue manager we will be using and what the name of the queue(s) we will be creating are called.

Edit the queue name by using `nano zos_host.yml` . You will need to change the variable mq_queue_name to "ANSIBLE.TEAMXX.QUEUE"

```
# Queue to be altered
mq_queue_name: "ANSIBLE.TEST.QUEUE"
```

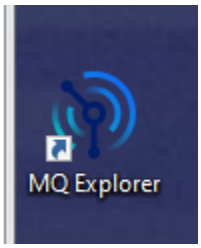
Running our MQ playbooks

11.) At this point, you should see and understand the wiring that lets ansible connect between your workstation and z/OS. So, now let's execute!

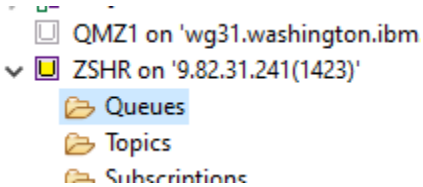
Back out of the group_vars directory with the command `cd ..` , and then let's run a playbook!

```
ansible-playbook –I inventory.ini mq_define_qlocal.yml
```

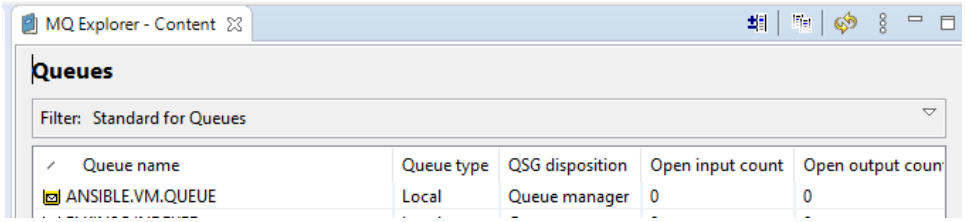
12.) When you run the command, you will get a ton of output in your terminal, but there's an easier way to see if the task was successful. Go to your desktop and select the MQ Explorer application.



Once inside the MQ Explorer application, connect to the ZSHR queue manager by right-clicking the ZSHR option in the menu panel. The right-click menu will give you the option to connect, go ahead and click it. The white box should turn yellow. Press the dropdown arrow once connected to reveal the 'Queues', 'Topics' and more for ZSHR. Click on queues, you should now see your automation-created queue there!



13.) If you don't see the queue listed, hit the refresh button in the upper right-hand corner of MQ explorer

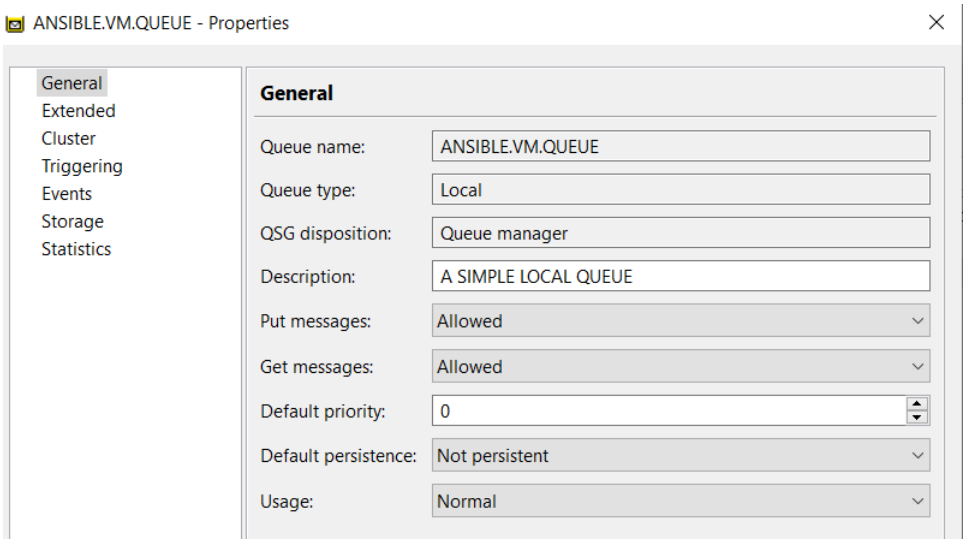


14.) Congratulations! You have now created a queue using ansible, now let's learn more about its properties using ansible. Back on your terminal, enter the command:

```
ansible-playbook -I inventory.ini mq_display_qlocal.yml
```

Here, you can see all the attributes of the queue that you previously defined. In the output, find where persistence is indicated. The attribute is called DEFPSIST. Make note of what it indicates.

You can also check whether the queue is persistent or not in MQ Explorer by looking under the properties of the queue



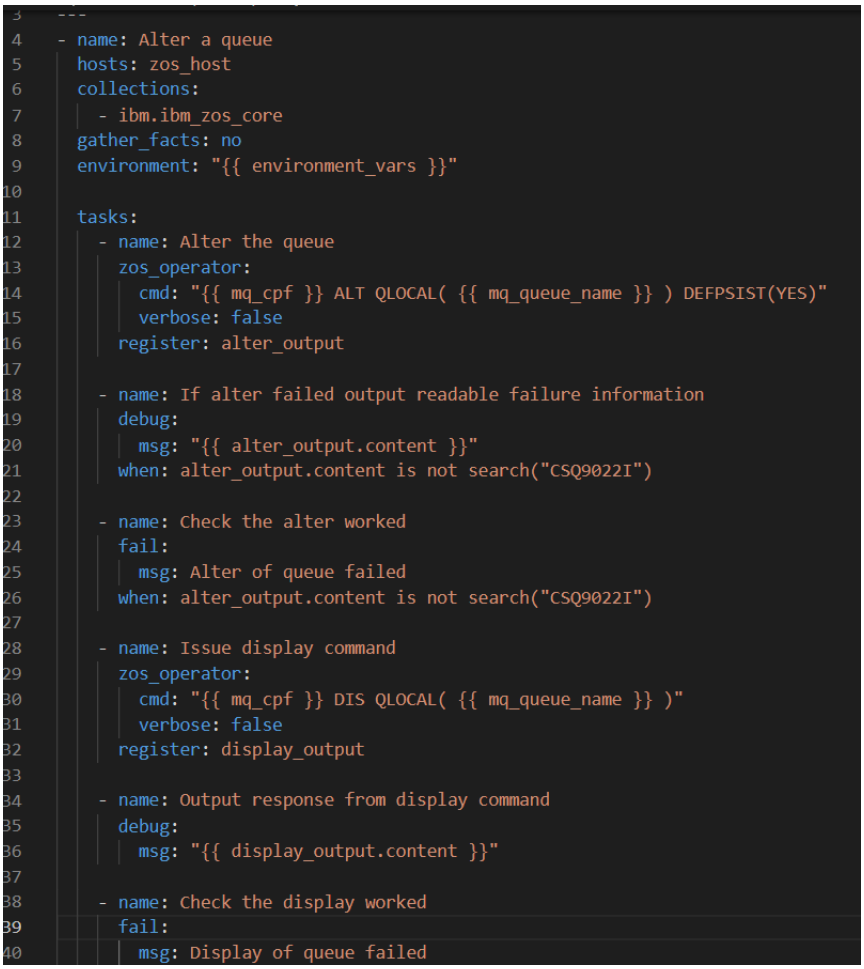
15.) Now, we'll use ansible to alter the properties. Run:

```
ansible-playbook -I inventory.ini mq_alter_qlocal.yml
```

As you might imagine, this altered our queue. Go check the Properties of the queue using MQ Explorer to see if you can spot the difference.

16.) By this point, you are probably wondering, what is making each of these playbooks work? Before executing our final playbook, let's peek inside the alter playbook to see how it is arranged and affecting z/OS.

Use the command `cat mq_alter_qlocal.yml`



```yaml
---
- name: Alter a queue
  hosts: zos_host
  collections:
    - ibm.ibm_zos_core
  gather_facts: no
  environment: "{{ environment_vars }}"

  tasks:
    - name: Alter the queue
      zos_operator:
        cmd: "{{ mq_cpf }} ALT QLOCAL( {{ mq_queue_name }} ) DEFPSIST(YES)"
        verbose: false
      register: alter_output

    - name: If alter failed output readable failure information
      debug:
        msg: "{{ alter_output.content }}"
      when: alter_output.content is not search("CSQ9022I")

    - name: Check the alter worked
      fail:
        msg: Alter of queue failed
      when: alter_output.content is not search("CSQ9022I")

    - name: Issue display command
      zos_operator:
        cmd: "{{ mq_cpf }} DIS QLOCAL( {{ mq_queue_name }} )"
        verbose: false
      register: display_output

    - name: Output response from display command
      debug:
        msg: "{{ display_output.content }}"

    - name: Check the display worked
      fail:
        msg: Display of queue failed
```

This is what a typical playbook looks like. You can see that in our play, we have 6 tasks that will be accomplished by the playbook's execution. If you are familiar with z/OS, you will recognize the format of cmd: "{{ mq_cpf }} ALT QLOCAL({{ mq_queue_name }}) DEFPSIST(YES)" .

The command is associated with the zos_operator module. The zos_operator module is one of many modules contained in the IBM core collection that enable seamless integration between ansible and z/OS.

You can find documentation for the zos_operator module here: https://ibm.github.io/z_ansible_collections_doc/ibm_zos_core/docs/source/modules/zos_operator.html

From the ansible documentation: "Ansible modules execute tasks. One or more Ansible tasks can be combined to make a play. Two or more plays can be combined to create an Ansible Playbook. Ansible Playbooks are lists of tasks that automatically execute against hosts. Groups of hosts form your Ansible inventory."

17.) At this point, you have run 3 playbooks and investigated how all the pieces work together. It's time to delete our queue! Go ahead and run one last playbook command

```
ansible-playbook -I inventory.ini mq_delete_qlocal.yml
```

Refresh the list of queues in your terminal. You should no longer see the queue you created.

****LAB END****

What else can ansible do for your MQ environment?

Underpinning each playbook we tried today, is the zos_operator module. The zos_operator module is one of the outcomes of the collaboration between ansible and z/OS developers. The commands we were able to execute today allowed us to directly connect with the z/OS command line from our workstations. What other modules are currently available in the z/OS core collection?

- zos_apf – Add or remove libraries to Authorized Program Facility (APF)
- zos_backup_restore – Backup and restore data sets and volumes
- zos_blockinfile – Manage block of multi-line textual data on z/OS
- zos_copy – Copy data to z/OS
- zos_data_set – Manage data sets
- zos_encode – Perform encoding operations.
- zos_fetch – Fetch data from z/OS
- zos_find – Find matching data sets
- zos_job_output – Display job output
- zos_job_query – Query job status
- zos_job_submit – Submit JCL
- zos_lineinfile – Manage textual data on z/OS
- zos_mount – Mount a z/OS file system.
- zos_mvs_raw – Run a z/OS program.
- zos_operator – Execute operator command
- zos_operator_action_query – Display messages requiring action
- zos_ping – Ping z/OS and check dependencies.
- zos_tso_command – Execute TSO commands

This list is growing! What automation makes sense for your environment?