A Python CLI Application for Instance(s) Configuration with Files

An Ansible configuration Approach

Lokesh Kola
lokeshkola2000@gmail.com

Instructor: *Nikolaos Kakouros, KTH*
Date: February 1, 2023

1 Introduction

The application's main aim is to take the instance names as parameters and then upload the attached files to instances. Then execute some shell commands for Compiling a C program, executing, and storing the result in a file. The operating system used for implementation was Ubuntu 22.04.

Internally ansible is responsible for dealing with all given instances at a time. Ansible is an open-source IT automation tool that automates provisioning, configuration management, application deployment, orchestration, and many other manual IT processes. Ansible will connect to the instances through SSH. Here in this application python ansible module was used for implementation. Then the expected tasks were performed successfully and the test cases were also passed with the localhost as an instance.

The tasks, of which the application is capable are:

1. **Task-1 :** Creating Temporary Directory on Instances
2. **Task-2 :** Uploading C Program File and Makefile(consists of command for compiling C Program) to Instances
3. **Task-3 :** Executing C Program by using Makefile
4. **Task-4 :** Moving the C Executable file to the current working directory
5. **Task-5 :** Deleting the Temporary Directory
6. **Task-6 :** Giving Execution permissions
7. **Task-7 :** Executes C Executable file and stores output to a file

The test cases for tasks are:

1. **For Task-1 :** Checking whether the directory was created or not.
2. **For Task-2 :** Checking both files uploaded or not.
3. **For Task-3 :** After Execution of makefile check for the executable exists or not.
4. **For Task-4 :** Checking the executable file in the current working directory.
5. **For Task-5 :** Checking the directory deleted or not
6. **For Task-6 :** Checking the permission set to 764 or not.
7. **For Task-7 :** Checking the output file created or not.

2 Implementation

Initially, an ssh key should be generated by using one of the algorithms among rsa, dsa, ecdsa, and ed25519 with the command ssh-keygen. Then need to copy the public key to the instances(localhost) then ansible can connect to instances without any password.

Then coming to implementation, initially, all the required classes were instantiated. Those classes were

1. Data Loader
2. Inventory Manager
3. Variable Manager
4. Task Queue Manager(expects the objects created above)

Then the playbook was implemented in a dictionary. All the tasks and sub-commands of tasks were also formed in a dictionary. The playbook is shown in below Figure 1.

```
63  # Creating a Playbook with Tasks and stores as Dictionary
64  play_book = dict(
65      name="Compiles and Install a C Program File on Remote Instances", # Name of the Play_Book
66      hosts=hosts_list, # Setting Hosts list to hosts(similar to "all")
67      gather_facts='yes', # Gathering all the tasks
68      tasks=[
69          #dict(action=dict(module='debug', args=dict(msg="Creating Temporary directory on Remote Instances"))),
70          dict(action=dict(module='shell', args='mkdir Remote_Instance_dir')), #Creating Temporary Directory on Instances
71          #dict(action=dict(module='debug', args=dict(msg="Uploading C Program File"))),
72          dict(action=dict(module='copy', args=dict(src= 'matinv.c',dest='Remote_Instance_dir/'))), #Uploading C Program File to Instances
73          #dict(action=dict(module='debug', args=dict(msg="Uploading Makefile(which has command for compilation)"))),
74          dict(action=dict(module='copy', args=dict(src= 'Makefile',dest='Remote_Instance_dir/'))), #Uploading Makefile to Instances
75          #dict(action=dict(module='debug', args=dict(msg="Running Makefile (Compiling C Program)"))),
76          dict(action=dict(module='shell', args='make -C Remote_Instance_dir')), #Executing Makefile
77          #dict(action=dict(module='debug', args=dict(msg="Installing C Executable File"))),
78          dict(action=dict(module='copy', args=dict(remote_src=True, src= 'Remote_Instance_dir/mat_inv',dest='.'))), #Moving Executable to ouside of temporary
79          #dict(action=dict(module='debug', args=dict(msg="Deleting Temporary directory"))),
80          dict(action=dict(module='shell', args='rm -r Remote_Instance_dir')), #Deleting the Temporary Directory
81          #dict(action=dict(module='debug', args=dict(msg="Giving Execute Permission"))),
82          dict(action=dict(module='shell', args='chmod u+x mat_inv')), #Giving Execute Permission for Executable.
83          #dict(action=dict(module='debug', args=dict(msg="Executing C Program"))),
84          dict(action=dict(module='shell', args='./mat_inv -n 1024 -I fast -P 1 > output.txt')), #Executing Executable and storing output to a file on Instanc
85          #dict(action=dict(module='debug', args=dict(msg="SUCCESS"))),
86      ]
87  )
```

Figure 1: Play-Book with Tasks

So dictionary contains keys and their values. Initially given a name for the playbook and assigned hosts to all hosts(list of hosts). Then gather-facts was set to yes which makes the ansible module gather all the tasks first and then move for execution.

Then all the tasks were implemented in another dictionary. Each task was also a dictionary and parameters with modules were also a sub-dictionary. The tasks with their test cases were explained below.

1. **Task-1 :** task 1 was creating the temporary directory and this task was done by using a shell command with a shell module as an action. The shell command was ”mkdir Remote-Instance-dir”.

   Test cases for Task-1 : After creating the temporary directory. The task is to get the information of ls with grep of the directory name and register the output to a variable. Then checking the directory name in the output by using assert.

2. **Task-2 :** task 2 was copying the C program file and Makefile to the temporary directory and this task was done by using a source, and destination with a copy module as an action.

   Test cases for Task-2 : After uploading the both files to instances. The task is to get the information of ls in the temporary directory with grep of file names and register the output to a variable. Then check the file names in the output by using assert.

3. **Task-3 :** task 3 was executing the makefile, which consists of a shell command for compiling a C program, and this task was done by using a shell command with a shell module as an action. The shell command was ”make -C Remote-Instance-dir”.

   Test cases for Task-3 : After executing the makefile the C executable file will be created. The task is to get the information of ls in the temporary directory with grep of executable file names and register the output to a variable. Then check the file name in the output by using assert.

4. **Task-4 :** task 4 was copying the C executable file from out of the temporary directory and this task was done by using a remote source, and destination parameters with a copy module as an action.

   Test cases for Task-4 : After transferring the executable file from out of the temporary directory. The task is to get the information of ls with grep of the executable file name and register the output to a variable. Then checking the directory name in the output by using assert.

5. **Task-5 :** task 5 was deleting the temporary directory and this task was done by using a shell command with a shell module as an action. The shell command was ”rm -r Remote-Instance-dir”.

   Test cases for Task-5 : After deleting the temporary directory. The task is to get the information of ls and register the output to a variable. Then checking the directory name is not in the output by using assert.

6. **Task-6 :** task 6 was providing execution permissions for the executable file. This task was done by using a shell command with a shell module as an action. The shell command was ”stat -c ”%a” mat-inv”’.

   Test cases for Task-6 : After providing the execution permission. The test is to get the information on the permissions of the executable file and register the output to a variable. Then check the file permissions ”764” in the output by using assert.

7. **Task-7 :** task 7 was executing the C executable and storing the output in a file. This task was done by using a shell command with a shell module as an action. The shell command was ”./mat-inv -n 1024 -I fast -P 1 ¿ output.txt”.

   Test cases for Task-7 : After execution the output is stored in a file. The task is to get the information of ls with grep of the file name and register the output to a variable. Then check the output file name in the output by using assert.

The Test cases were shown in below Figure 2.

```
63    # Creating a Playbook with Tasks and stores as Dictionary
64    play_book = dict(
65        name="Compiles and Install a C Program File on Remote Instances", # Name of the Play_Book
66        hosts=hosts_list, # Setting Hosts list to hosts(similar to "all")
67        gather_facts='yes', # Gathering all the tasks
68        tasks=[
69            #dict(action=dict(module='debug', args=dict(msg="Creating Temporary directory on Remote Instances"))),
70            dict(action=dict(module='shell', args='mkdir Remote_Instance_dir')), #Creating Temporary Directory on Instances
71            #dict(action=dict(module='debug', args=dict(msg="Uploading C Program File"))),
72            dict(action=dict(module='copy', args=dict(src= 'matinv.c',dest='Remote_Instance_dir/'))), #Uploading C Program File to Instances
73            #dict(action=dict(module='debug', args=dict(msg="Uploading Makefile(which has command for compilation)"))),
74            dict(action=dict(module='copy', args=dict(src= 'Makefile',dest='Remote_Instance_dir/'))), #Uploading Makefile to Instances
75            #dict(action=dict(module='debug', args=dict(msg="Running Makefile (Compiling C Program)"))),
76            dict(action=dict(module='shell', args='make -C Remote_Instance_dir')), #Executing Makefile
77            #dict(action=dict(module='debug', args=dict(msg="Installing C Executable File"))),
78            dict(action=dict(module='copy', args=dict(remote_src=True, src= 'Remote_Instance_dir/mat_inv',dest='.'))), #Moving Executable to ouside of temporary
79            #dict(action=dict(module='debug', args=dict(msg="Deleting Temporary directory"))),
80            dict(action=dict(module='shell', args='rm -r Remote_Instance_dir')), #Deleting the Temporary Directory
81            #dict(action=dict(module='debug', args=dict(msg="Giving Execute Permission"))),
82            dict(action=dict(module='shell', args='chmod u+x mat_inv')), #Giving Execute Permission for Executable.
83            #dict(action=dict(module='debug', args=dict(msg="Executing C Program"))),
84            dict(action=dict(module='shell', args='./mat_inv -n 1024 -I fast -P 1 > output.txt')), #Executing Executable and storing output to a file on Instanc
85            #dict(action=dict(module='debug', args=dict(msg="SUCCESS"))),
86        ]
87    )
```

Figure 2: Play-Book with Tasks and their Test cases

Then loading the playbooks to Play class. The play object is given to the Task Queue Manager object for execution. As the custom callback was not used and the default callback shows the output on the terminal. The Task Queue Manager object will return an integer value by using that we can say that whether all tasks were executed successfully or not. If the return value is 0 then everything is fine else some tasks failed in execution. Eventually, cleaned up all the forked processes and temporary files in objects of the data loader and task queue manager.