**Exercise - Configure Ansible in Azure Cloud Shell**

In this exercise, you create an Ansible configuration file and the inventory file. Then you run an Ansible module that verifies that Ansible can connect to your VMs.

You start by creating these files locally. You then upload the files to Azure Cloud Shell so that you can run Ansible. By storing these files locally, you build out a Git repository that you later push to GitHub and then connect to Azure Pipelines.

**Create the Ansible configuration file**

Ansible reads configuration settings from *~/ansible.cfg*. Here, you add settings that:

* Hide warnings that you don't yet need to understand.
* Disable host key checking, so that you're not prompted to verify the authenticity of your servers.

1. In Visual Studio Code, select **New File** from the files pane. Name the file *ansible.cfg*.

Screenshot of adding a file in Visual Studio Code

1. Add these contents to *ansible.cfg*, and then save the file:

**ini**

[defaults]

deprecation\_warnings = False

host\_key\_checking = False

**Create the inventory file**

Specify your machine inventory. Recall that the inventory specifies the list of nodes under management. In this module, you define a dynamic inventory to enable Ansible to discover your nodes at runtime.

1. In Visual Studio Code, select **New File** from the files pane. Name the file *azure\_rm.yml*.

Screenshot of adding a file in Visual Studio Code

1. Add these contents to *azure\_rm.yml*, and then save the file:

**yml**

plugin: azure\_rm

include\_vm\_resource\_groups:

- learn-ansible-rg

auth\_source: auto

keyed\_groups:

- prefix: tag

key: tags

**Note**

When using a dynamic inventory with Azure, you must ensure that this file ends with the name *azure\_rm.yml* or *azure\_rm.yaml*.

**Upload files to Cloud Shell**

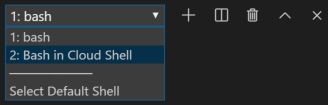
Upload your Ansible configuration file and your inventory file to the home directory in Cloud Shell.

Remember that we use Cloud Shell in this module because it comes with Ansible already set up for you.

1. In Visual Studio Code, select F1 or select **View** > **Command Palette** to access the command palette.
2. In the command palette, enter *Azure: Upload to Cloud Shell*.
3. Select *ansible.cfg*.
4. Repeat the process to upload *azure\_rm.yml*.

**Verify the files in Cloud Shell**

1. In Visual Studio Code, switch from your Bash session to your Cloud Shell session. To do so, select **Bash in Cloud Shell** in the terminal.



1. Run the following ls command to verify that your Ansible configuration file and your inventory file were successfully uploaded to Cloud Shell:

**Bash**

ls azure\_rm.yml ansible.cfg

You see both files listed.

1. Run the following ansible-config command to verify your configuration settings:

**Bash**

ansible-config view

You see this:

**Output**

[defaults]

deprecation\_warnings = False

host\_key\_checking = False

1. Run the following ansible-inventory command to verify that Ansible can discover your inventory.

**Bash**

ansible-inventory --inventory azure\_rm.yml --graph

Your output resembles this:

**Output**

@all:

|--@tag\_Ansible\_mslearn:

| |--vm1\_1bbf

| |--vm2\_867a

|--@ungrouped:

**Run the ping module on your VMs**

Although you typically write *playbooks* to express your desired configurations, you can also run Ansible modules directly.

To verify that Ansible can apply configuration changes to your inventory, run the ping module directly. This ensures that your VMs are discoverable and that Ansible can connect to each VM.

Unlike what the name suggests, the ping module doesn't connect over the ICMP protocol. Rather, it connects over SSH, just like every other Ansible module. Think of the ping module as a way to verify that Ansible can connect, and that Python is correctly installed on each node. You don't typically use this module in a playbook.

To run the ping module directly, run this ansible command from Cloud Shell:

**Bash**

ansible \

--inventory azure\_rm.yml \

--user azureuser \

--private-key ~/.ssh/ansible\_rsa \

--module-name ping \

tag\_Ansible\_mslearn

The tag\_Ansible\_mslearn argument relates to the keyed\_groups section in your inventory file. Recall that when you created your VMs, you provided the --tags Ansible=mslearn argument, to create a tag named "Ansible" whose value is "mslearn". In the tag\_Ansible\_mslearn argument:

* tag maps to the feature you're grouping by. Here, you're grouping by resource tag.
* Ansible maps to the tag name.
* mslearn maps to the tag's value.

You see this JSON output:

**JSON**

vm2\_867a | SUCCESS => {

"ansible\_facts": {

"discovered\_interpreter\_python": "/usr/bin/python"

},

"changed": false,

"ping": "pong"

}

vm1\_1bbf | SUCCESS => {

"ansible\_facts": {

"discovered\_interpreter\_python": "/usr/bin/python"

},

"changed": false,

"ping": "pong"

}

You see "SUCCESS" and "pong" in the output, which tell you that the command succeeded.

The discovered\_interpreter\_python part shows the path where Python is found on each node. The Ubuntu image on Azure comes with Python installed. If Python were not to be installed, you'd see an error message that reminds you to install Python on each node.

Your Cloud Shell session is all set up to run Ansible on your VMs. Keep your Cloud Shell session in Visual Studio Code open for the next exercise.