**Exercise - Run an Ansible playbook in Azure Cloud Shell**

In this exercise, you run an Ansible playbook that configures user accounts on Linux VMs running on Azure. By the end, you'll have two Linux VMs that are configured in the exact same way.

During the process, you:

* Walk through a basic Ansible playbook so that you understand what each piece does.
* Create the Ansible playbook locally, and then upload the playbook to Azure Cloud Shell.
* Run the Ansible playbook on your Linux VMs.
* Verify that your VMs were configured as you expect.
* Run the playbook a second time to see idempotency in action.

**Define your requirements**

The first step when working with any configuration management tool is to define your requirements. You then map those requirements to configuration code.

Let's join Tim and Andy as they define the requirements for their prototype.

**Tim:** Before we run anything in Azure Pipelines, I'd like to work through the process by hand. What's the easiest way to get started?

**Andy:** Although we could install Ansible on our development systems, we can also use Cloud Shell. Ansible comes pre-configured on Cloud Shell, so there's nothing to set up. What's something that you commonly need to configure?

**Tim:** We run services on VMs that perform billing and other functions. We run these services under service accounts.

**Andy:** That sounds like a great place to start. Tell me more about what you need.

*Tim moves to the whiteboard.*

**Tim:** A service account is a user account that provides a security context for services. On Linux, a service account typically:

* Provides no home directory for the user.
* Provides no login shell.
* Provides no password or way to sign in as that user.

I say we define two users for our test. Let's call them *testuser1* and *testuser2*.

**Define the basic configuration**

Tim and Andy discover documentation for the [user](https://docs.ansible.com/ansible/latest/modules/user_module.html) module. The user module enables you to manage user accounts, including service accounts.

They define this configuration in Ansible:

**yml**

---

- hosts: all

tasks:

- name: Add service accounts

user:

name: "testuser1"

comment: service account

create\_home: no

shell: /usr/sbin/nologin

state: present

The hosts part specifies to run the playbook on all hosts in the inventory.

The tasks part defines each module to apply. This configuration defines one module, the user module.

The user module defines the user account named *testuser1*. This module specifies:

* A descriptive comment for the user account.
* Not to create the home directory for the user.
* The user's login shell. The /usr/sbin/nologin prevents login.
* That the account should exist. You can use the absent state to remove the user from the system.

**Use the loop block to add a second user**

Notice that this configuration creates just one user: *testuser1*. Although you can add a second task that performs the same function, you can use the loop keyword to repeat the task multiple times.

Tim and Andy update their configuration, like this:

**yml**

---

- hosts: all

tasks:

- name: Add service accounts

user:

name: "{{ item }}"

comment: service account

create\_home: no

shell: /usr/sbin/nologin

state: present

loop:

- testuser1

- testuser2

The loop block applies the configuration for each item in the list. The name attribute uses the {{ }} syntax to interpolate, or replace, the value with the current element in the list.

**Activate privilege escalation as the root user**

Next, Tim and Andy need to consider under which user account the configuration will run. When they created their VMs on Azure, they specified *azureuser* as the administrator name. But to configure user accounts, they need to activate privilege escalation as the *root* user.

To do so, they need to add the become directive to their configuration, like this:

**yml**

---

- hosts: all

become: yes

tasks:

- name: Add service accounts

user:

name: "{{ item }}"

comment: service account

create\_home: no

shell: /usr/sbin/nologin

state: present

loop:

- testuser1

- testuser2

A related setting, not shown here, is become\_user. The default value of this setting is root. When this configuration runs, Ansible activates privilege escalation as the *root* user.

**Note**

The meaning of become can vary based on the kind of system you're working with. But on Linux systems, you can think of become as applying the configuration with sudo privileges.

**Create the users playbook**

Now that you've verified that your VMs are connectable through Ansible, here you apply a playbook that configures service accounts on your VMs.

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

Screenshot of adding a file in Visual Studio Code

1. Add these contents to *users.yml* and then save the file:

**yml**

---

- hosts: all

become: yes

tasks:

- name: Add service accounts

user:

name: "{{ item }}"

comment: service account

create\_home: no

shell: /usr/sbin/nologin

state: present

loop:

- testuser1

- testuser2

1. Upload *users.yml* to Cloud Shell. To do so:
   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 *users.yml*.

**Run the users playbook on your VMs**

Run the users playbook on your VMs from your Cloud Shell session.

1. In Visual Studio Code, switch to your Cloud Shell session.
2. Run the following ls command to verify that *users.yml* exists in your Cloud Shell.

**Bash**

ls users.yml

1. Run the following ansible-playbook command to apply your playbook:

**Bash**

ansible-playbook \

--inventory azure\_rm.yml \

--user azureuser \

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

users.yml

The --user argument specifies the user to connect as. Here, you specify your administrator user, *azureuser*. Recall that the playbook activates privilege escalation as the *root* user.

You see from the output that both user accounts were added to each VM in your inventory:

**Output**

PLAY [all] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [vm2\_867a]

ok: [vm1\_1bbf]

TASK [Add service accounts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [vm2\_867a] => (item=testuser1)

changed: [vm1\_1bbf] => (item=testuser1)

changed: [vm2\_867a] => (item=testuser2)

changed: [vm1\_1bbf] => (item=testuser2)

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vm1\_1bbf : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

vm2\_867a : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

**Verify the configuration**

The output shows that the Ansible playbook succeeded, but let's double-check the configuration.

It can be useful to verify the result as you develop your configuration code, to ensure that your code correctly expresses your needs. But after you have a working configuration in place, you don't need to verify the results each time.

On Linux, you can use the getent passwd command to get information about a user. One way to run this command is over SSH.

As an optional step, run these commands from Cloud Shell:

**Azure CLI**

IP\_ADDRESS=$(az vm list-ip-addresses \

--resource-group learn-ansible-rg \

--name vm1 \

--query [0].virtualMachine.network.publicIpAddresses[0].ipAddress \

--output tsv)

ssh -i ~/.ssh/ansible\_rsa azureuser@$IP\_ADDRESS "/usr/bin/getent passwd testuser1"

You see this:

**Output**

testuser1:x:1001:1001:service account:/home/testuser1:/usr/sbin/nologin

Here's how to read each field:

**Output**

testuser1:x:1001:1001:service account:/home/testuser1:/usr/sbin/nologin

| | | | | | |

| | | | | | Login shell

| | | | | Home directory

| | | | Description

| | | Group ID

| | User ID

| Encrypted password

Username

You see that the account is set up as you specified. Although the getent passwd command shows */home/testuser1* as the home directory, the directory doesn't actually exist on the VM. That's because you specified create\_home: no.

This command tests only one VM. But let's say you have dozens or hundreds of VMs that you want to verify. You can use Ansible to run the same command on each of your VMs.

To do that, run the following ansible command:

**Bash**

ansible \

--inventory azure\_rm.yml \

--user azureuser \

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

--args "/usr/bin/getent passwd testuser1" \

tag\_Ansible\_mslearn

The --args argument specifies the command to run on each VM.

You see that both VMs are configured identically with the *testuser1* service account.

**Output**

vm2\_867a | CHANGED | rc=0 >>

testuser1:x:1001:1001:service account:/home/testuser1:/usr/sbin/nologin

vm1\_1bbf | CHANGED | rc=0 >>

testuser1:x:1001:1001:service account:/home/testuser1:/usr/sbin/nologin

**Run the Ansible playbook a second time**

Ansible is idempotent. That means you can run it as many times as you'd like. Ansible applies configuration changes only when your playbook changes, or when the state of your systems doesn't match what's in your playbook.

Run the following ansible-playbook command to apply your playbook a second time:

**Bash**

ansible-playbook \

--inventory azure\_rm.yml \

--user azureuser \

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

users.yml

You will see this output:

**Output**

PLAY [all] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [vm2\_867a]

ok: [vm1\_1bbf]

TASK [Add service accounts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [vm2\_867a] => (item=testuser1)

ok: [vm1\_1bbf] => (item=testuser1)

ok: [vm2\_867a] => (item=testuser2)

ok: [vm1\_1bbf] => (item=testuser2)

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vm1\_1bbf : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

vm2\_867a : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

As you might expect, Ansible doesn't apply any changes. The changed=0 part of the output tells you this.

That's because your Ansible playbook didn't change, and you haven't modified any of the user accounts on your VMs.

Later, you'll see the benefit of this behavior when you configure your infrastructure in Azure Pipelines by running the same Ansible playbook.

Keep your Cloud Shell session in Visual Studio Code open for the next exercise.