Learning Objectives

Learners will be able to...

- **Manage groups**
- **Add and remove users**
- **Manage passwords**
- **Manage admin privileges**

Managing Groups

The Group Module

The `ansible.builtin.group` module is used to manage groups on a host.

The task below is an example of how to create group:

```
- name: Ensure group "foo" exists
  ansible.builtin.group:
    name: foo
    state: present
```

- `name` specifies the managed group
- `state` specifies the presence of the group on the host:
  - `state` `present` (default) ensures the group exists
  - `state` `absent` removes the group

It is possible to set a group's gid (group ID) using the `gid` parameter:

```
- name: Ensure group "foo" exists
  ansible.builtin.group:
    name: foo
    state: present
    gid: 2000
```

While `ansible.builtin.group` can be used to add and/or remove groups, managing users within those groups requires the `user` module.

Managing Users

The User Module

The `ansible.builtin.user` module allows you to create and/or modify users.

The task below is an example showing how to create the user `cardib`:

```
- name: ensure the user 'cardib' exists
  ansible.builtin.user:
    name: cardib
    state: present
    system: false
```

- `state` can be `present` to ensure the user exists or `absent` to remove them.
- `system` specifies account type:
  - `system false` - (default) is a user account
  - `system true` - is a system account for services

Below is an example of adding the user created in the previous example to the 'admin' group:

```
- name: Add cardib to group admin
  ansible.builtin.user:
    name: cardib
    group: admin
```

- `group` (string) or `groups` (list) can be used to add the user to specific groups.

It's important to note that in the example shown above, as the user module adds `cardib` to the `admin` group, it also removes the user from any group to which it previously belonged.

warning

Unintended Consequences

As mentioned previously, Ansible is by default programming for state, including the state of the user's membership to groups. Different groups may have different rights on the server; if a user is unintentionally removed from a group and consequently loses sudo rights, they may be unable to run Ansible with admin rights on the host system.

To avoid removing a user from any pre-existing group membership, the `append` parameter can be used.
- `append: true` will only add users to the groups specified.
- `append: false` (default) will replace the group(s) a user belongs to.

Managing Users (continued)

Home Directory and Shell

Ansible will create a home directory for the user if it doesn't exists; to prevent this use the `create_home` parameter and set it to`create_home: no`

The `home` parameter allows us to specify the home directory's path and the `shell` parameter sets the path to shell for the user.

> info
>
> ### Shell Parameter Paths
>
> The shell can default to different paths depending on the operating system. See this link for more on the user module's shell parameter.

Expire

To create a temporary user, the `expires` parameter can be used. It accepts expiry time in epoch; since Ansible 2.6, using a negative value will remove expiry time.

```
- name: Adding user with account expiring at end of 2025
  ansible.builtin.user:
    name: offset
    shell: /bin/bash
    groups: migos
    expires: 1767250799
```

Remove

When `state: absent`, Ansible will remove user from the system but will leave the user home folder untouched. To remove directories associated with the user use `remove: yes` with `state: absent`:

```
- name: Remove the user 'cardib'
  ansible.builtin.user:
    name: cardib
    state: absent
    remove: yes
```

Managing Passwords

The User Module: Password Management

`ansible.builtin.user` can also be used to manage passwords and ssh keys.

The Password Parameter

The `password` parameter can be used to set user password. Since Ansible playbooks are written in YAML, they are plain text that can easily be viewed or edited. Therefore, the hashed password (masked version) should be passed here and not the plaintext password.

```
- name: user 'cardib' with password
  ansible.builtin.user:
    name: cardib
    state: present
    password:
'$6$abc$7JkzWNO0fUbALkqI26avMCt6mdHxHwxPztgnpifwpHxTq3LzQzTHAWAV
JpqQblVzRSVFC7JfxlhUjgLAto9d2/'
```

Password Policy

The user module also contains parameters and return values to manage password policy. The return values `password_expire_max` and `password_expire_min` can specify number of days between password changes (though both are only supported on Linux).

An important parameter is `update_password`, which can have 2 values:
* `always` - always updates user's password with every Ansible run.
* `on_create` - set password only on user creation.

Managing SSH Keys

The User Module: Generating SSH Keys

Using the user module, Ansible can generate SSH keys for users using the `generate_ssh_key` parameter:

```
- name: create SSH keys for the user
  ansible.builtin.user:
    name: cardib
    state: present
    generate_ssh_key: true
    ssh_key_type: ecdsa
    ssh_key_file: .ssh/id_rsa
```

If `generate_ssh_key` is set to `true`, the `ssh_key_file` return value will contain the corresponding private key in the generated path; alternatively, use `ssh_public_key` to return a public key.

Add an Existing Public Key

The authorize key module can be used to manage SSH authorized keys for users on a remote host.

The following examples manage SSH access for the user `cardib` using the `ansible.posix.authorized_key` module.

```
- name: Set authorized key taken from file
  ansible.posix.authorized_key:
    user: cardib
    state: present
    key: "ssh-rsa A......"
```

In this example, the `key` parameter is used with a specified string. Alternative, `key` can load from a file system:

```
- name: Set authorized key taken from file
  ansible.posix.authorized_key:
    user: cardib
    state: present
    key: "{{ lookup('file', '/home/cardib/.ssh/id_rsa.pub') }}"
```