Approx time to complete this assignment: 12 hours

Generating Ansible Inventory File (dmusers.yaml)

I created a Bash script that generates an Ansible inventory file named `dmusers.yaml`. The inventory file defines user attributes for multiple hosts, allowing for centralized management of user accounts and attributes across different systems.

The Bash script accomplishes the following tasks:

1. **Defining Hosts:** The script starts by defining the hosts for which user attributes will be managed using Ansible. In this example, the hosts are `saclass(all hosts except FreeBSD)`, `rocky(all rocky linux machines)`, and `debian (all debian machines)`.
2. **Users:** Next, the script enumerates user accounts on the local system. The user information (*UID, GECOS, home directory, shell, group, and additional groups*) is collected for each user.
3. **Generating Ansible Tasks:** For each user, the script generates Ansible tasks to update user attributes. Each task includes the following attributes:
    - `name`: A descriptive name for the task.
    - `user`: An Ansible module used to manage user attributes.
    - `uid`: The User ID (UID) of the user.
    - `comment`: The GECOS (user information) field.
    - `home`: The user's home directory.
    - `group`: The primary group of the user.
    - `groups`: Additional groups to which the user belongs.
    - `shell`: The user's login shell.
    - `create_home`: Indicates whether the user's home directory should be created if it doesn't exist.
4. **Host-Specific User Updates:** Special handling is provided for the `"snir8112"` user, as it has different group attributes on the `rocky` and `debian` hosts. For all rocky linux machines, snir8112 is a part of wheel group and debian machines, snir8112 is a part of sudo group.

Generating umask.yaml file

The Ansible playbook accomplishes the following tasks:

1. **Host Selection:** The playbook is designed to target hosts belonging to the `saclass` group. The `become: yes` directive indicates that privilege escalation to the root user will be used to execute tasks requiring elevated permissions.

2. **Copy Umask Configuration:** The playbook includes a task named "Copy umask.sh to /etc/profile.d/." This task uses the `copy` module to transfer a file named `umask.sh` from a source location to the `/etc/profile.d/` directory on the remote hosts. The `umask.sh` file presumably contains specific `umask` configurations.
3. **File Ownership and Permissions:** The task ensures that the copied `umask.sh` file is owned by the `root` user and belongs to the `root` group. It sets the file's mode to "0644," which allows read access for everyone and write access for the owner while restricting execute permissions.

Generating webcheck.yaml

This Ansible playbook is designed to manage web services on two different hosts: `machinec` and `machined`. It includes tasks for ensuring the Apache2 service is running and up to date on the `machinec` host, as well as the HTTPD service on the `machined` host. Additionally, it checks whether the services are enabled to start on boot and provides their update status.

Host Configuration:

For `machinec`:

- Privilege escalation to the root user is enabled using `become: yes`.

Tasks for `machinec`:

1. **Ensure Apache2 Service is Running:**
   - The `service` module is used to ensure that the Apache2 service is running.
   - The result is registered in the `apache_service` variable.
2. **Check if Apache2 Service is Enabled on Boot:**
   - The `command` module checks whether the Apache2 service is enabled to start on boot.
   - The result is registered in the `apache_enabled` variable.
   - The task is marked as not "changed" using `changed_when: false`.
3. **Update APT Package Cache:**
   - The `apt` module is used to update the APT package cache.
   - Privilege escalation to the root user is required.
4. **Check the Available Apache2 Version:**
   - The `apt` module is used to check for the available version of Apache2.
   - The result is registered in the `apache_update` variable.
5. **Display Apache2 Service Status:**
   - The `debug` module displays the current status of the Apache2 service.
6. **Display If Apache2 Service is Enabled on Boot:**

    - The `debug` module displays whether the Apache2 service is enabled to start on boot.
7. **Display Message if Apache2 is Not Up to Date:**
    - The `fail` module displays a message if Apache2 is not up to date based on the presence of certain text in `apache_update.stdout`.
8. **Display Message if Apache2 is Up to Date:**
    - The `debug` module displays a message if Apache2 is already at the latest version.

For `machined`:

Privilege escalation to the root user is enabled using `become: yes`.

Tasks for `machined`:

1. **Ensure HTTPD Service is Running:**
    - The `service` module is used to ensure that the HTTPD service is running.
    - The result is registered in the `httpd_service` variable.
    - The task is marked as not "changed" using `changed_when: false`.
2. **Check if HTTPD Service is Enabled on Boot:**
    - The `command` module checks whether the HTTPD service is enabled to start on boot.
    - The result is registered in the `httpd_enabled` variable.
    - The task is marked as not "changed" using `changed_when: false`.
3. **Display HTTPD Service Status:**
    - The `debug` module displays the current status of the HTTPD service.
4. **Display If HTTPD Service is Enabled on Boot:**
    - The `debug` module displays whether the HTTPD service is enabled to start on boot.
5. **Check for Available Updates for HTTPD:**
    - The `command` module checks for available updates for the HTTPD package.
    - The result is registered in the `updates_output` variable.
    - The task is marked as not "changed" and errors are ignored using `changed_when: false` and `failed_when: false`, respectively.
6. **Check for Available Packages for HTTPD:**
    - The `command` module checks for available packages for the HTTPD package.
    - The result is registered in the `available_output` variable.
    - The task is marked as not "changed" and errors are ignored using `changed_when: false` and `failed_when: false`, respectively.
7. **Display Update Status for HTTPD:**

- The `block` structure is used to display the update status of the HTTPD package. If updates are available, it provides feedback to indicate whether the package is up to date or not.