MANAGING VARIABLES AND INCLUSIONS

Overview	
Goal	To describe variable scope and precedence, manage variables and facts in a play, and manage inclusions.
Objectives	• Manage variables in Ansible projects • Manage Facts in Playbooks • Include variables and tasks from external files into a playbook
Sections	• Managing Variables (and Guided Exercise) • Managing Facts (and Guided Exercise) • Managing Inclusions (and Guided Exercise)
Lab	• Lab: Managing Variables and Inclusions

Ansible supports variables that can be used to store values that can be reused throughout files in an entire Ansible project. This can help simplify creation and maintenance of a project and reduce the incidence of errors.
Variables provide a convenient way to manage dynamic values for a given environment in your Ansible project. Some examples of values that variables might contain include
• Users to create
• Packages to install
• Services to restart
• Files to remove
• Archives to retrieve from the Internet

Defining Variables
Variables can be defined in a bewildering variety of places in an Ansible project. However, this can be simplified to three basic scope levels:
• Global scope: Variables set from the command line or Ansible configuration
• Play scope: Variables set in the play and related structures
• Host scope: Variables set on host groups and individual hosts by the inventory, fact gathering, or registered tasks
If the same variable name is defined at more than one level, the higher wins. So variables defined by the inventory are overridden by variables defined by the playbook, which are overridden by variables defined on the command line.
A detailed discussion of variable precedence is available in the Ansible documentation, a link to which is provided in the References at the end of this section.

Variables in Playbooks
Defining Variables in Playbooks:-
When writing playbooks, administrators can use their own variables and call them in a task. For example, a variable web_package can be defined with a value of httpd and called by the yum module in order to install the httpd package.
Playbook variables can be defined in multiple ways. One of the simplest is to place it in a vars block at the beginning of a playbook:

```
- hosts: all
  vars:
    user: joe
    home: /home/joe
```

It is also possible to define playbook variables in external files. In this case, instead of using vars, the vars_files directive may be used, followed by a list of external variable files relative to the playbook that should be read:

```
- hosts: all
  vars_files:
    - vars/users.yml
```

Registered Variables

Administrators can capture the output of a command by using the register statement. The output is saved into a variable that could be used later for either debugging purposes or in order to achieve something else, such as a configuration based on a command's output.
The following playbook demonstrates how to capture the output of a command for debugging purposes:

```
---
- hosts: dev
  become: yes
  tasks:
    - name: install pkg and show the output
      yum:
        name: httpd
        state: installed
      register: install_result

    - debug: var=install_result
```

Guided Exercise: Managing Variables

#cat **variables.yml**

```
---
- hosts: dev
  become: yes
  vars:
    web_pkg: httpd
    firewall_pkg: firewalld
    web_service: httpd
    firewall_service: firewalld
    python_pkg: python-httplib2
    rule: http
  tasks:
    - name: Required packages are installed and up to date
      yum:
        name:
          - "{{ web_pkg }}"
          - "{{ firewall_pkg }}"
          - "{{ python_pkg }}"
        state: latest

    - name: The {{ firewall_service }} service is started and enabled
      service:
        name: "{{ firewall_service }}"
        enabled: true
        state: started
    - name: webserivce start
      service:
        name: "{{ web_service }}"
        enabled: true
        state: started
    - name: Web content is in place
      copy:
        content: "Robo robo"
        dest: /var/www/html/index.html
    - name: The firewall port for {{ rule }} is open
      firewalld:
        service: "{{ rule }}"
        permanent: true
        immediate: true
        state: enabled

- hosts: localhost
  become: false
  tasks:
    - name: Ensure the webserver is reachable
      uri:
        url: http://robo2
        status_code: 200
```

Check the syntax error on playbook:-

ansible-playbook --syntax-check variables.yml

ansible-playbook variables.yml

```
SUDO password:

PLAY [dev] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [robo2]

TASK [Required packages are installed and up to date] **************************
changed: [robo2]

TASK [The firewalld service is started and enabled] ****************************
changed: [robo2]

TASK [webserivce start] ********************************************************
changed: [robo2]

TASK [Web content is in place] *************************************************
changed: [robo2]

TASK [The firewall port for http is open] **************************************
changed: [robo2]

PLAY [localhost] ***************************************************************

TASK [Gathering Facts] *********************************************************
ok: [robo]

TASK [Ensure the webserver is reachable] ***************************************
ok: [robo]

PLAY RECAP *********************************************************************
robo                       : ok=2    changed=0    unreachable=0    failed=0
robo2                      : ok=6    changed=5    unreachable=0    failed=0
```

Naming Variables

Variables have names which consist of a string that must start with a letter and can only contain letters, numbers, and underscores.

Consider the following table that shows the difference between invalid and valid variable names:

Ansible variable names	
Invalid variable names	**Valid variable names**
`web server`	`web_server`
`remote.file`	`remote_file`
`1st file`	`file_1` or `file1`
`remoteserver$1`	`remote_server_1` or `remote_server1`