Ansible Control Node, Inventory & Managed Nodes

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

Session - 9 Agenda:

1. Ansible Variables

Variables In Ansible:

Variable: A variable is a value that can change, depending on conditions or on information passed to the program. Variables are used to store information to be referenced and manipulated in a computer program.

Why do we require variables?

It is sometime very difficult and hectic task to use any complicate or complex repetitive value. So, we store it in a variable and call the variable in the program.

Type of Variables in Ansible:

Ansible variables can be classified into two categories:

a) *System defined (built-in) variables.*
b) User defined variables.

(a) System defined (built-in) variables.

inventory_hostname, hostvars, groups, group_names etc.

(b) User defined variables.

var=1.7320508
nehraclasses=20

Variable Name:

Variable name should fulfill the following conditions:

- variable names can have letters (alphabets), numbers & underscores.
- variable names should always start with letters not with number(s).
- There shouldn't be any spaces or - present in the variable names.
- Variable names cannot have special characters in them.
- Acceptable variable names are:
  - abc
  - xyz
  - ABC
  - abc123
  - abc_123

- Invalid variable names are:
  - nehra classes
  - nehra-classes
  - Nehr@Classes
  - 20NehraClasses

Places in Ansible where we can define the variables:

1) Playbooks
2) Command Line
3) Inventory Files

1) Playbook Variables:

Variables which are defined in a playbook are called playbook variables.

Using variables in playbooks:

a) How to use system defined variables (built-in variables) in playbooks.

```
# mkdir playbooks

# cd playbooks

# vim system_variables.yml
---
- hosts: localhost
  tasks:
    - name: Ansible Buit-in Variable Playbook
      debug:
         msg: "{{ inventory_hostname }}"
...
```

Let's check the playbook for errors.

```
# ansible-playbook system_variables.yml --syntax-check
```

Now, we can run this playbook.

```
# ansible-playbook system_variables.yml
```

Similary, we can use other system defined variables in playbooks.

```
# vim system_variables2.yml
```

Put other built-in variables in the msg field and check.

b) How to use user defined variables in playbooks.

```
# vim hello.yml
---
- hosts: RHEL
  vars:
      salutations: Hello Everyone
  tasks:
   - name: Example of user defined variables.
     debug:
        msg: "{{ salutations }}"
```

Let's check the playbook for errors.

```
# ansible-playbook hello.yml --syntax-check
```

Now, we can run this playbook.

```
# ansible-playbook hello.yml
```

Types of Ansible Playbook Variables:

a) String type variable
b) Listing/Array type variable
c) Mapping type variable

a) String type variable:

```
BestLinuxChannel: NehraClasses
package: vsftpd
```

b) Listing/Array type variable:

```
var:
 - httpd
 - vsftpd
 - nfs
```

c) Mapping (Dictionary) type variable:

```
users:
  - name: nehra
    pw: redhat
    uid: 1004
  - name: vikas
    pw: redhat
    uid: 1005
```

a) String type variables examples.

```
# vim string_variables.yml
---
- name: Example of string type playbook variables.
  hosts: RHEL
  vars:
     apache: httpd
     apache_root: /var/www/html
  tasks:
      - name: Installing Apache HTTPD Server.
        yum:
           name: "{{ apache }}"
           state: installed
      - name: Starting & enabling the httpd service.
        service:
           name: "{{ apache }}"
           state: started
           enabled: true
      - name: Creating "{{ apache_root }}" directory.
        file:
           path: "{{ apache_root }}"
           state: directory
      - name: Creating index.html page inside "{{ apache_root }}"
        copy:
           content: "Welcome To Nehra Classes"
           dest: "{{ apache_root }}/index.html"
...
```

Let's check the playbook for errors.

```
# ansible-playbook string_variables.yml --syntax-check
```

Now, we can run this playbook.

```
# ansible-playbook string_variables.yml
```

It requires sudo, root or admin privileges to push the tasks.

```
# ansible-playbook string_variables.yml -b
```

There is no need to mention -b option while we execute the playbooks with root privileged.

We can permanently set it in the playbook file as well if needed.

```
# vim string_variables.yml
---
- name: Example of string type playbook variables.
  hosts: RHEL
  become: true
  vars:
     apache: httpd
     apache_root: /var/www/html
  tasks:
      - name: Installing Apache HTTPD Server.
        yum:
           name: "{{ apache }}"
           state: installed
      - name: Starting & enabling the httpd service.
        service:
           name: "{{ apache }}"
           state: started
           enabled: true
      - name: Creating "{{ apache_root }}" directory.
        file:
           path: "{{ apache_root }}"
           state: directory
      - name: Creating index.html page inside "{{ apache_root }}"
        copy:
           content: "Welcome To Nehra Classes"
           dest: "{{ apache_root }}/index.html"
...
```

Now, we can run this playbook.

```
# ansible-playbook string_variables.yml --syntax-check
```

Now, we can run this playbook without mentioning -b there.

```
# ansible-playbook string_variables.yml
```

If there is requirement of a lot of variables in your environment, it will cause complexities in your playbooks. The size of the playbook will increase depending on the number of variables used. And you have to manually mention the variables in other playbooks if same variables are needed there.

So, there is a solution of these above-mentioned problems. We can create a separate playbook containing all of the variables and import them in primary playbooks when needed.

Importing Variables in Ansible Playbook:

Let's import variables from other playbooks to primary playbook. First let's create primary file and mention vars_file option there.

```
# cp string_variables.yml string_variables2.yml

# vim string_variables2.yml
---
- name: Example of string type playbook variables.
  hosts: RHEL
  become: true
  vars_files:
      - /home/vikasnehra/playbooks/myvars.yml
   tasks:
      - name: Installing Apache HTTPD Server.
        yum:
           name: "{{ apache }}"
           state: installed
      - name: Starting & enabling the httpd service.
        service:
           name: "{{ apache }}"
           state: started
           enabled: true
      - name: Creating "{{ apache_root }}" directory.
        file:
            path: "{{ apache_root }}"
            state: directory
      - name: Creating index.html page inside "{{ apache_root }}"
        copy:
            content: "Welcome To Nehra Classes"
            dest: "{{ apache_root }}/index.html"
...

# vim /home/vikasnehra/playbooks/myvars.yml
apache: httpd
apache_root: /var/www/html/
```

Let's check the playbook for errors.
ansible-playbook string_variables2.yml --syntax-check

Now, we can run this playbook.
ansible-playbook string_variables2.yml

---

b) List/Array type variables examples.

Let's create a playbook having a list of packages to be installed on the node machines.

```
# vim list_variables.yml
---
- name: Example of listing type playbook variables.
  hosts: RHEL
  become: true
  vars:
    mypkgs:
      - ftp
      - vsftpd
      - zsh
tasks:
   - name: Installing the packages.
     yum:
        name: "{{ mypkgs }}"
        state: installed
...
```

Let's check the playbook for errors.
ansible-playbook list_variables.yml --syntax-check

Now, we can run this playbook.
ansible-playbook list_variables.yml

We can use arrays as well if needed. Let's check the availability of telnet package first.
ansible RHEL -m command -a 'rpm -qi telnet'

Let's install telnet package in the node machines using array variables in the playbook.

```
# vim array_variables.yml
- name: Example of listing type playbook variables.
  hosts: RHEL
  become: true
  vars:
    mypkgs:
      - ftp
      - vsftpd
      - zsh
      - telnet
      - httpd
 tasks:
    - name: Installing the packages.
      yum:
         name: "{{ mypkgs [3] }}"
         state: installed
...
```

Let's check the playbook for errors.

```
# ansible-playbook array_variables.yml --syntax-check
```

Now, we can run this playbook.

```
# ansible-playbook array_variables.yml -v
```

Let's check the availability of telnet package again.

```
# ansible RHEL -m command -a 'rpm -qi telnet'
```

---

c) Mapping (Dictionary) type variables examples.

Mapping type variable are used in complex playbook to automate the tasks on the node machines, like using variables with loops.

```
# vim mapping_variables.yml
---
- hosts: RHEL
  become: yes
  tasks:
   - name: Create new users.
     user:
       name: '{{ item.name }}'
       uid: '{{ item.uid }}'
```

```
    state: present
  loop:
    - name: vikas
      uid: 1020
    - name: amit
      uid: 1030
    - name: greta
      uid: 1040
```

Let's check this playbook for errors.
ansible-playbook mapping_variables.yml --syntax-check

Now, we can run this playbook.
ansible-playbook mapping_variables.yml

Verify the changes:
ansible RHEL -m command -a 'id -a vikas'
ansible RHEL -m command -a 'id -a amit'
ansible RHEL -m command -a 'id -a greta'

Prompt for user input with vars_prompt:

When running a playbook, there may be information that you need to collect at runtime. This may be sensitive information, such as passwords. At other times, this is information that can only be provided by the end user at runtime, such as the password to use for the root user when bootstrapping a system.

You can collect this information from the end user by specifying a vars_prompt section in your playbook. When you run the playbook, it will ask the questions you've specified and record the answers, ready to be used as variables.

vim variableswithprompt.yml

```
---
 - name: Prompt for user input
   hosts: RHEL
   vars_prompt:
     - name: user_name
       prompt: Please enter your username.
       private: no
```

```
  - name: password
    prompt: Please enter your password.
    private: yes
 tasks:
  - debug:
      msg: "The username is {{ user_name }} and password is {{ password }}"
```

Let's check this playbook for errors.
ansible-playbook variableswithprompt.yml --syntax-check

Now, we can run this playbook.
ansible-playbook variableswithprompt.yml

Thanks