TASK ITERATION WITH LOOPS

Using loops saves administrators from the need to write multiple tasks that use the same module. For example, instead of writing five tasks to ensure five users exist, you can write one task that iterates over a list of five users to ensure they all exist.

Ansible supports iterating a task over a set of items using the loop keyword. You can configure loops to repeat a task using each item in a list, the contents of each of the files in a list, a generated sequence of numbers, or using more complicated structures. This section covers simple loops that iterate over a list of items. Consult the documentation for more advanced looping scenarios.

Simple Loops A simple loop iterates a task over a list of items. The loop keyword is added to the task, and takes as a value the list of items over which the task should be iterated. The loop variable item holds the value used during each iteration.

Consider the following snippet that uses the service module twice in order to ensure two network services are running:

#vim withoutloop.yml

```
- name : Installing Selected Packages
  hosts : db
  tasks :
    - name : Installing vim
      yum :
        name : vim
        state : latest
    - name : Installing Docker
      yum :
        name : docker
        state : latest
    - name : Installing wget
```

```
    yum :
      name : wget
      state : latest
:wq!
```

These three tasks can be rewritten to use a simple loop so that only one task is needed to ensure to install all the three packages:

#vim loops1.yml

```
---
 - name : Installing Some Selected Packages
   hosts : db
   tasks :
     - name : Installing uptools, ypbind & LDAP Packages
       yum :
         name : "{{ item }}"
         state : latest
       loop :
         - vim
         - docker
         - wget
:wq!
```

The list used by loop can be provided by a variable. In the following example, the variables mail_pkg, mail_services contains the list of Packages & services that need to be installed and running.

#vim loops2.yml

```
---
```

```
- name : Play with variables & Loops
  hosts : db
  tasks :
  vars :
    mail_pkg :
      - postfix
      - dovecot
    mail_service :
      - postfix
      - dovecot
  tasks :
    - name : Installing Mail Services
      yum :
        name : "{{ item }}"
        state : latest
      loop : "{{ mail_pkg }}"
    - name : Ensuring Mail Services are UP
      service :
        name : "{{ item }}"
        state : started
      loop : "{{ mail_service }}"
:wq!
```

Loops over a List of Hashes or Dictionaries

The loop list does not need to be a list of simple values. In the following example, each item in the list is actually a hash or a dictionary. Each hash or dictionary in the example has two keys, name and groups, and the value of each key in the current item loop variable can be retrieved with the item.name and item.groups variables, respectively.

#vim loopshash.yml

```
---
 - name : Creating users with selected group
   hosts : db
   tasks :
    - name : Creating user
      user :
        name : "{{ item.name }}"
        state : present
        groups : "{{ item.groups }}"
      loop :
        - name : user11
          groups : aws
        - name : user12
          groups : azure
```

:wq!

Earlier-Style Loop Keywords

Before Ansible 2.5, most playbooks used a different syntax for loops. Multiple loop keywords were provided, which were prefixed with with_, followed by the name of an Ansible look-up plug-in (an advanced feature not covered in detail in this course). This syntax for looping is very common in existing playbooks, but will probably be deprecated at some point in the future.

LOOP KEYWORD

with_items :

Behaves the same as the loop keyword for simple lists, such as a list of strings or a list of hashes/dictionaries. Unlike loop, if lists of lists are provided to with_items, they are flattened into a singlelevel list. The loop variable item holds the list item used during each iteration.

with_file :

This keyword requires a list of control node file names. The loop variable item holds the content of a corresponding file from the file list during each iteration

with_sequence :

Instead of requiring a list, this keyword requires parameters to generate a list of values based on a numeric sequence. The loop variable item holds the value of one of the generated items in the generated sequence during each iteration.

An example of with_items in a playbook is shown below:

#vim loops4.yml

```
---
 - name : Creating User
   hosts : db
   tasks :
     - name : Creating Users
       user :
         name : "{{ item }}"
         state : present
         shell : /bin/sh
         createhome : yes
```

```
    home : /home/{{ item }}
  with_items :
    - user101
    - user102
    - user103
    - user104
    - user105
:wq!
```