Variables

Variable in playbooks are **very similar** to using variables in any programming language. It helps you to use and assign a value to a variable and use that anywhere in the playbook. One can put conditions around the value of the variables and accordingly use them in the playbook.

Example

```
- hosts : <your hosts>
vars:
tomcat_port : 8080
```

In the above example, we have defined a variable name **tomcat_port** and assigned the value 8080 to that variable and can use that in your playbook wherever needed.

Now taking a reference from the example shared. The following code is from one of the roles (install-tomcat) −

```
block:
   - name: Install Tomcat artifacts
      action: >
      yum name = "demo-tomcat-1" state = present
      register: Output

   always:
      - debug:
         msg:
            - "Install Tomcat artifacts task ended with message:
{{Output}}"
            - "Installed Tomcat artifacts - {{Output.changed}}"
```

Here, the output is the variable used.

Let us walk through all the keywords used in the above code −

- **block** − Ansible syntax to execute a given block.
- **name** − Relevant name of the block - this is used in logging and helps in debugging that which all blocks were successfully executed.
- **action** − The code next to action tag is the task to be executed. The action again is a Ansible keyword used in yaml.
- **register** − The output of the action is registered using the register keyword and Output is the variable name which holds the action output.
- **always** − Again a Ansible keyword , it states that below will always be executed.
- **msg** − Displays the message.

Usage of variable - {{Output}}

This will read the value of variable Output. Also as it is used in the msg tab, it will print the value of the output variable.

Additionally, you can use the sub properties of the variable as well. Like in the case checking {{Output.changed}} whether the output got changed and accordingly use it.

Exception Handling in Playbooks

Exception handling in Ansible is similar to exception handling in any programming language. An example of the exception handling in playbook is shown below.

```
tasks:
   - name: Name of the task to be executed
      block:
         - debug: msg = 'Just a debug message , relevant for
logging'
         - command: <the command to execute>

      rescue:
         - debug: msg = 'There was an exception.. '
         - command: <Rescue mechanism for the above exception
occurred)

      always:
         - debug: msg = "this will execute in all scenarios. Always
will get logged"
```

Following is the syntax for exception handling.

- **rescue** and **always** are the keywords specific to exception handling.
- Block is where the code is written (anything to be executed on the Unix machine).
- If the command written inside the block feature fails, then the execution reaches rescue block and it gets executed. In case there is no error in the command under block feature, then rescue will not be executed.
- **Always** gets executed in all cases.
- So if we compare the same with java, then it is similar to try, catch and finally block.
- Here, **Block** is similar to **try block** where you write the code to be executed and **rescue** is similar to **catch block** and **always** is similar to **finally**.

Loops

Below is the example to demonstrate the usage of Loops in Ansible.

The tasks is to copy the set of all the war files from one directory to tomcat webapps folder.

Most of the commands used in the example below are already covered before. Here, we will concentrate on the usage of loops.

Initially in the 'shell' command we have done ls *.war. So, it will list all the war files in the directory.

Output of that command is taken in a variable named output.

To loop, the 'with_items' syntax is being used.

with_items: "{{output.stdout_lines}}" --> output.stdout_lines gives us the line by line output and then we loop on the output with the with_items command of Ansible.

Attaching the example output just to make one understand how we used the stdout_lines in the with_items command.

```
---
#Testing
- hosts: tomcat-node
   tasks:
      - name: Install Apache
      shell: "ls *.war"
      register: output
      args:
         chdir: /opt/ansible/tomcat/demo/webapps

      - file:
         src: '/opt/ansible/tomcat/demo/webapps/{{ item }}'
         dest: '/users/demo/vivek/{{ item }}'
         state: link
      with_items: "{{output.stdout_lines}}"
```

Blocks

The playbook in totality is broken into blocks. The smallest piece of steps to execute is written in block. Writing the specific instruction in blocks helps to segregate functionality and handle it with exception handling if needed.

Example of blocks is covered in variable usage,exception handling and loops above.

Conditionals

Conditionals are used where one needs to run a specific step based on a condition.

```
---
#Tsting
- hosts: all
   vars:
      test1: "Hello Vivek"
   tasks:
      - name: Testing Ansible variable
      debug:
```

```
msg: "Equals"
when: test1 == "Hello Vivek"
```

In this case, Equals will be printed as the test1 variable is equal as mentioned in the when condition. **when** can be used with a logical OR and logical AND condition as in all the programming languages.

Just change the value of test1 variable from Hello Vivek to say Hello World and see the output.