Ansible - Introduction

Ansible is simple open source IT engine which automates application deployment, intra service orchestration, cloud provisioning and many other IT tools.

Ansible is easy to deploy because it does not use any agents or custom security infrastructure.

Ansible uses playbook to describe automation jobs, and playbook uses very simple language i.e. **YAML** (It's a human-readable data serialization language & is commonly used for configuration files, but could be used in many applications where data is being stored)which is very easy for humans to understand, read and write. Hence the advantage is that even the IT infrastructure support guys can read and understand the playbook and debug if needed (YAML – It is in human readable form).

Ansible is designed for multi-tier deployment. Ansible does not manage one system at time, it models IT infrastructure by describing all of your systems are interrelated. Ansible is completely agentless which means Ansible works by connecting your nodes through ssh(by default). But if you want other method for connection like Kerberos, Ansible gives that option to you.

After connecting to your nodes, Ansible pushes small programs called as "Ansible Modules". Ansible runs that modules on your nodes and removes them when finished. Ansible manages your inventory in simple text files (These are the hosts file). Ansible uses the hosts file where one can group the hosts and can control the actions on a specific group in the playbooks.

Sample Hosts File

This is the content of hosts file −

```
#File name: hosts
#Description: Inventory file for your application. Defines machine
type abc
node to deploy specific artifacts
# Defines machine type def node to upload
metadata.

[abc-node]
#server1 ansible_host = <target machine for DU deployment>
ansible_user = <Ansible
user> ansible_connection = ssh
server1 ansible_host = <your host name> ansible_user = <your unix
user>
ansible_connection = ssh

[def-node]
#server2 ansible_host = <target machine for artifact upload>
```

```
ansible_user = <Ansible user> ansible_connection = ssh
server2 ansible_host = <host> ansible_user = <user>
ansible_connection = ssh
```

What is Configuration Management

Configuration management in terms of Ansible means that it maintains configuration of the product performance by keeping a record and updating detailed information which describes an enterprise's hardware and software.

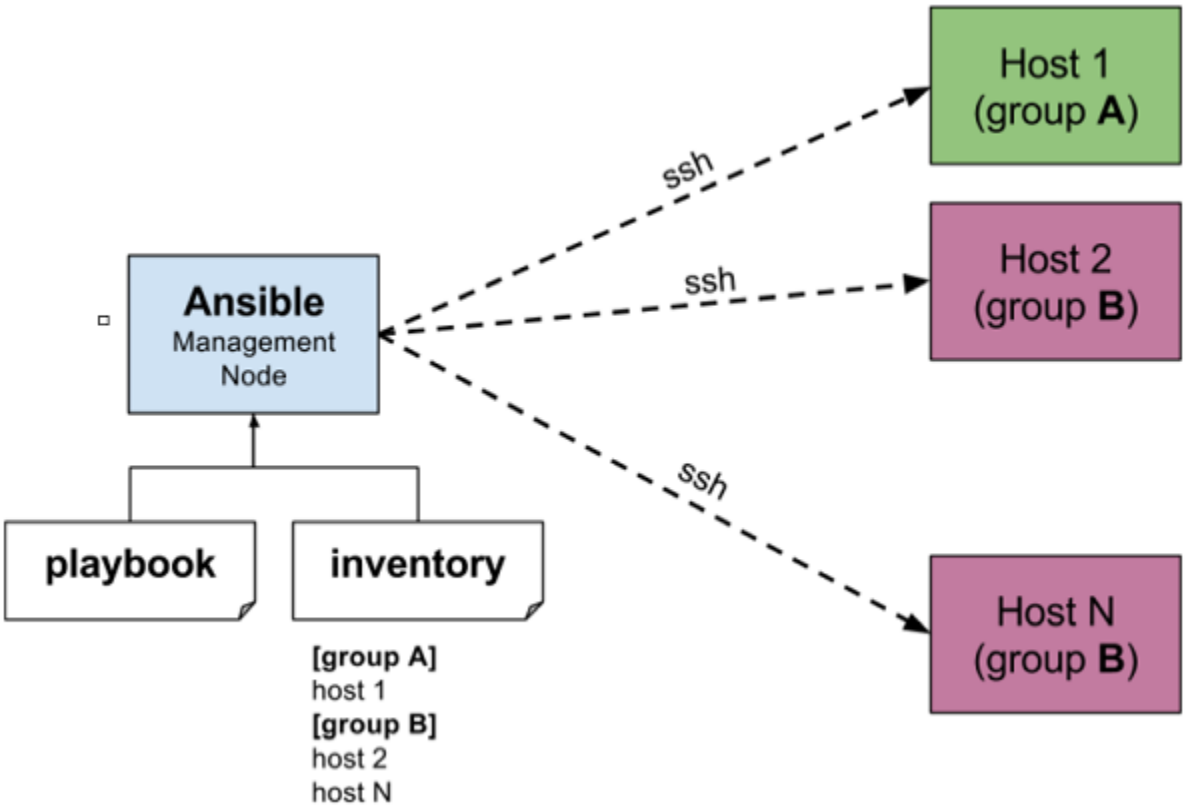
Such information typically includes the exact versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices. For e.g. If you want to install the new version of **WebLogic/WebSphere** server on all of the machines present in your enterprise, it is not feasible for you to manually go and update each and every machine.

You can install WebLogic/WebSphere in one go on all of your machines with Ansible playbooks and inventory written in the most simple way. All you have to do is list out the IP addresses of your nodes in the inventory and write a playbook to install WebLogic/WebSphere. Run the playbook from your control machine & it will be installed on all your nodes.

How Ansible Works?

The picture given below shows the working of Ansible.

Ansible works by connecting to your nodes and pushing out small programs, called "**Ansible** modules" to them. **Ansible** then executes these modules (over SSH by default), and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.



The management node in the above picture is the controlling node (managing node) which controls the entire execution of the playbook. It's the node from which you are running the installation. The inventory file provides the list of hosts where the Ansible modules needs to be run and the management node does a SSH connection and executes the small modules on the hosts machine and installs the product/software.

Beauty of Ansible is that it removes the modules once those are installed so effectively it connects to host machine , executes the instructions and if it's successfully installed removes the code which was copied on the host machine which was executed.

Ansible - Environment Setup

In this chapter, we will learn about the environment setup of Ansible.

Installation Process

Mainly, there are two types of machines when we talk about deployment −

- **Control machine** − Machine from where we can manage other machines.
- **Remote machine** − Machines which are handled/controlled by control machine.

There can be multiple remote machines which are handled by one control machine. So, for managing remote machines we have to install Ansible on control machine.

Control Machine Requirements

Ansible can be run from any machine with Python 2 (versions 2.6 or 2.7) or Python 3 (versions 3.5 and higher) installed.

Note − Windows does not support control machine.

By default, Ansible uses **ssh** to manage remote machine.

Ansible does not add any database. It does not require any daemons to start or keep it running. While managing remote machines, Ansible **does not** leave any software installed or running on them. Hence, there is no question of how to upgrade it when moving to a new version.

Ansible can be installed on control machine which have above mentioned requirements in different ways. You can install the latest release through Apt, yum, pkg, pip, OpenCSW, pacman, etc.

Installation through Apt on Ubuntu Machine

For installing Ansible you have to configure PPA on your machine. For this, you have to run the following line of code −

```
$ sudo apt-get update
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible $ sudo apt-get update
$ sudo apt-get install ansible
```

After running the above line of code, you are ready to manage remote machines through Ansible. Just run Ansible–version to check the version and just to check whether Ansible was installed properly or not.

Ansible - YAML Basics

Ansible uses YAML syntax for expressing Ansible playbooks. This chapter provides an overview of YAML. Ansible uses YAML because it is very easy for humans to understand, read and write when compared to other data formats like XML and JSON.

Every **YAML** file optionally starts with “---” and ends with “...”.

Understanding YAML

In this section, we will learn the different ways in which the YAML data is represented.

key-value pair

YAML uses simple key-value pair to represent the data. The dictionary is represented in key: value pair.

Note − There should be space between : and value.

Example: A student record

```
--- #Optional YAML start syntax
james:
   name: james john
```

```
   rollNo: 34
   div: B
   sex: male
… #Optional YAML end syntax
```

Abbreviation

You can also use abbreviation to represent dictionaries.

Example

```
James: {name: james john, rollNo: 34, div: B, sex: male}
```

Representing List

We can also represent List in YAML. Every element(member) of list should be written in a new line with same indentation starting with "- " (- and space).

Example

```
---
countries:
   - America
   - China
   - Canada
   - Iceland
…
```

Abbreviation

You can also use abbreviation to represent lists.

Example

```
Countries: ['America', 'China', 'Canada', 'Iceland']
```

List inside Dictionaries

We can use list inside dictionaries, i.e., value of key is list.

Example

```
---
james:
   name: james john
   rollNo: 34
   div: B
   sex: male
   likes:
      - maths
      - physics
      - english
…
```

List of Dictionaries

We can also make list of dictionaries.

Example

```
---
```

```
- james:
   name: james john
   rollNo: 34
      div: B
   sex: male
   likes:
      - maths
      - physics
      - english

- robert:
      name: robert richardson
      rollNo: 53
      div: B
      sex: male
   likes:
      - biology
      - chemistry
…
```

YAML uses “|” to include newlines while showing multiple lines and “>” to suppress newlines while showing multiple lines. Due to this we can read and edit large lines. In both the cases intendentation will be ignored.

We can also represent **Boolean** (True/false) values in YAML. where **boolean** values can be case insensitive.

Example

```
---
- james:
   name: james john
   rollNo: 34
   div: B
   sex: male
   likes:
      - maths
      - physics
      - english

   result:
      maths: 87
      chemistry: 45
      biology: 56
      physics: 70
      english: 80

   passed: TRUE

   messageIncludeNewLines: |
      Congratulation!!
```

```
    You passed with 79%

messageExcludeNewLines: >
    Congratulation!!
    You passed with 79%
```

Some common words related to Ansible.

Service/Server − A process on the machine that provides the service.

Machine − A physical server, vm(virtual machine) or a container.

Target machine − A machine we are about to configure with Ansible.

Task − An action(run this, delete that) etc managed by Ansible.

Playbook − The yml file where Ansible commands are written and yml is executed on a machine.

Ansible - Ad hoc Commands

Ad hoc commands are commands which can be run individually to perform quick functions. These commands need not be performed later.

For example, you have to reboot all your company servers. For this, you will run the Adhoc commands from '**/usr/bin/ansible**'.

These ad-hoc commands are not used for configuration management and deployment, because these commands are of one time usage.

ansible-playbook is used for configuration management and deployment.

Parallelism and Shell Commands

Reboot your company server in 12 parallel forks at time. For this, we need to set up SSHagent for connection.

```
$ ssh-agent bash
$ ssh-add ~/.ssh/id_rsa
```

To run reboot for all your company servers in a group, 'abc', in 12 parallel forks −

```
$ Ansible abc -a "/sbin/reboot" -f 12
```

By default, Ansible will run the above Ad-hoc commands form current user account. If you want to change this behavior, you will have to pass the username in Ad-hoc commands as follows −

```
$ Ansible abc -a "/sbin/reboot" -f 12 -u username
```

File Transfer

You can use the Ad-hoc commands for doing **SCP** (Secure Copy Protocol) lots of files in parallel on multiple machines.

Transferring file to many servers/machines

```
$ Ansible abc -m copy -a "src = /etc/yum.conf dest = /tmp/yum.conf"
```

Creating new directory

```
$ Ansible abc -m file -a "dest = /path/user1/new mode = 777 owner =
user1 group = user1 state = directory"
```

Deleting whole directory and files

```
$ Ansible abc -m file -a "dest = /path/user1/new state = absent"
```

Managing Packages

The Ad-hoc commands are available for yum and apt. Following are some Ad-hoc commands using yum.

The following command checks if yum package is installed or not, but does not update it.

```
$ Ansible abc -m yum -a "name = demo-tomcat-1 state = present"
```

The following command check the package is not installed.

```
$ Ansible abc -m yum -a "name = demo-tomcat-1 state = absent"
```

The following command checks the latest version of package is installed.

```
$ Ansible abc -m yum -a "name = demo-tomcat-1 state = latest"
```

Gathering Facts

Facts can be used for implementing conditional statements in playbook. You can find adhoc information of all your facts through the following Ad-hoc command −

```
$ Ansible all -m setup
```

Ansible - Playbooks

In this chapter, we will learn about Playbooks in Ansible.

Playbooks are the files where Ansible code is written. Playbooks are written in YAML format. YAML stands for Yet Another Markup Language. **Playbooks** are one of the core features of Ansible and tell Ansible what to execute. They are like a to-do list for Ansible that contains a list of tasks.

Playbooks contain the steps which the user wants to execute on a particular machine. Playbooks are run sequentially. Playbooks are the building blocks for all the use cases of Ansible.

Playbook Structure

Each playbook is an aggregation of one or more plays in it. Playbooks are structured using Plays. There can be more than one play inside a playbook.

The function of a play is to map a set of instructions defined against a particular host.

YAML is a strict typed language; so, extra care needs to be taken while writing the YAML files. There are different YAML editors but we will prefer to use a simple editor like

notepad++. Just open notepad++ and copy and paste the below yaml and change the language to YAML (Language → YAML).

A YAML starts with --- (3 hyphens)

Create a Playbook

Let us start by writing a sample YAML file. We will walk through each section written in a yaml file.

```
---
   name: install and configure DB
   hosts: testServer
   become: yes

   vars:
      oracle_db_port_value : 1521

   tasks:
   -name: Install the Oracle DB
      yum: <code to install the DB>

   -name: Ensure the installed service is enabled and running
   service:
      name: <your service name>
```

The above is a sample Playbook where we are trying to cover the basic syntax of a playbook. Save the above content in a file as **test.yml**. A YAML syntax needs to follow the correct indentation and one needs to be a little careful while writing the syntax.

The Different YAML Tags

Let us now go through the different YAML tags. The different tags are described below –

name

This tag specifies the name of the Ansible playbook. As in what this playbook will be doing. Any logical name can be given to the playbook.

hosts

This tag specifies the lists of hosts or host group against which we want to run the task. The hosts field/tag is mandatory. It tells Ansible on which hosts to run the listed tasks. The tasks can be run on the same machine or on a remote machine. One can run the tasks on multiple machines and hence hosts tag can have a group of hosts' entry as well.

vars

Vars tag lets you define the variables which you can use in your playbook. Usage is similar to variables in any programming language.

tasks

All playbooks should contain tasks or a list of tasks to be executed. Tasks are a list of actions one needs to perform. A tasks field contains the name of the task. This works as the help text for the user. It is not mandatory but proves useful in debugging the playbook. Each task internally links to a piece of code called a module. A module that should be executed, and arguments that are required for the module you want to execute.

Ansible - Roles

Roles provide a framework for fully independent, or interdependent collections of variables, tasks, files, templates, and modules.

In Ansible, the role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing **complex playbooks**, and it makes them easier to reuse. The breaking of playbook allows you to logically break the playbook into reusable components.

Each role is basically limited to a particular functionality or desired output, with all the necessary steps to provide that result either within that role itself or in other roles listed as dependencies.

Roles are not playbooks. Roles are small functionality which can be independently used but have to be used within playbooks. There is no way to directly execute a role. Roles have no explicit setting for which host the role will apply to.

Top-level playbooks are the bridge holding the hosts from your inventory file to roles that should be applied to those hosts.

Creating a New Role

The directory structure for roles is essential to create a new role.

Role Structure

Roles have a structured layout on the file system. The default structure can be changed but for now let us stick to defaults.

Each role is a directory tree in itself. The role name is the directory name within the /roles directory.

```
$ ansible-galaxy -h
```

Usage

```
ansible-galaxy
[delete|import|info|init|install|list|login|remove|search|setup] [-
-help] [options] ...
```

Options

- **-h, --help** − Show this help message and exit.
- **-v, --verbose** − Verbose mode (-vvv for more, -vvvv to enable connection debugging)
- **--version** − Show program's version number and exit.

Creating a Role Directory

The above command has created the role directories.

```
$ ansible-galaxy init vivekrole
ERROR! The API server (https://galaxy.ansible.com/api/) is not
responding, please try again later.

$ ansible-galaxy init --force --offline vivekrole
- vivekrole was created successfully

$ tree vivekrole/
vivekrole/
├── defaults
│   └── main.yml
├── files ├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md ├── tasks
│   └── main.yml
├── templates ├── tests │    ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
```

Not all the directories will be used in the example and we will show the use of some of them in the example.

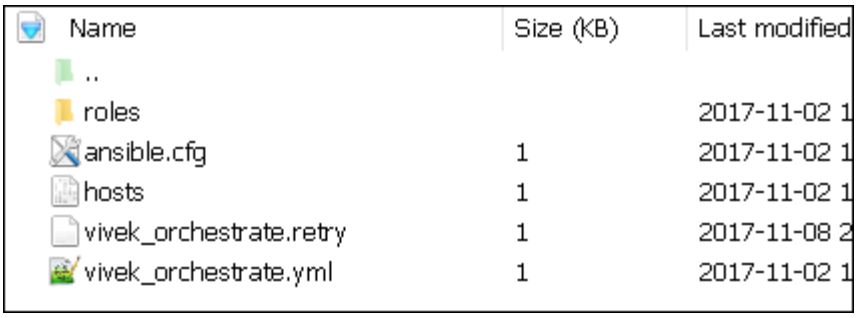
Utilizing Roles in Playbook

This is the code of the playbook we have written for demo purpose. This code is of the playbook vivek_orchestrate.yml. We have defined the hosts: **tomcat-node** and called the two roles – **install-tomcat** and **start-tomcat**.

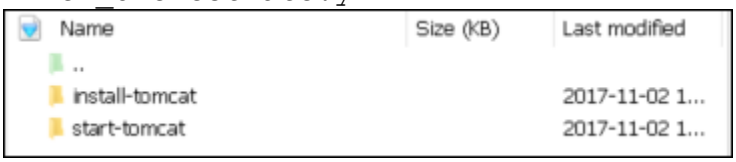
The problem statement is that we have a war which we need to deploy on a machine via Ansible.

```
---
- hosts: tomcat-node
roles:
   - {role: install-tomcat}
   - {role: start-tomcat}
```

Contents of our directory structure from where we are running the playbook.




```
$ ls
ansible.cfg  hosts  roles  vivek_orchestrate.retry
vivek_orchestrate.yml
```



There is a tasks directory under each directory and it contains a main.yml. The main.yml contents of install-tomcat are −

```
---
#Install vivek artifacts
-
   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}}"
```

The contents of main.yml of the start tomcat are −

```
#Start Tomcat
-
   block:
      - name: Start Tomcat
      command: <path of tomcat>/bin/startup.sh"
      register: output
      become: true

   always:
      - debug:
         msg:
            - "Start Tomcat task ended with message: {{output}}"
```

```
        - "Tomcat started - {{output.changed}}"
```

The advantage of breaking the playbook into roles is that anyone who wants to use the Install tomcat feature can call the Install Tomcat role.

Breaking a Playbook into a Role

If not for the roles, the content of the main.yml of the respective role can be copied in the playbook **yml** file. But to have modularity, roles were created.

Any logical entity which can be reused as a reusable function, that entity can be moved to role. The example for this is shown above

Ran the command to run the playbook.

```
-vvv option for verbose output – verbose output
$ cd vivek-playbook/
```

This is the command to run the playbook

```
$ sudo ansible-playbook -i hosts vivek_orchestrate.yml –vvv
-------------------------------------------------------------------
-------------------------------------------------------------------
----
```

Output

The generated output is as seen on the screen −

Using **/users/demo/vivek-playbook/ansible.cfg** as config file.

```
PLAYBOOK: vivek_orchestrate.yml
*****************************************************
*****************************************************
1 plays in vivek_orchestrate.yml

PLAY [tomcat-node]
*********************************************************************
***
******** ************************************************

TASK [Gathering Facts]
***********************************************
****************************
*******************************************
Tuesday 21 November 2017  13:02:05 +0530 (0:00:00.056) 0:00:00.056
******
Using module file
/usr/lib/python2.7/sitepackages/ansible/modules/system/setup.py
<localhost> ESTABLISH LOCAL CONNECTION FOR USER: root
<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'
<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo
   /root/.ansible/tmp/ansible-tmp-1511249525.88-259535494116870 `"
&&
```

```
   echo ansible-tmp-1511249525.88-259535494116870="`
   echo /root/.ansible/tmp/ansibletmp-1511249525.88-259535494116870
`" ) && sleep 0'
<localhost> PUT /tmp/tmpPEPrkd TO
   /root/.ansible/tmp/ansible-tmp-
1511249525.88259535494116870/setup.py
<localhost> EXEC /bin/sh -c 'chmod u+x
   /root/.ansible/tmp/ansible-tmp1511249525.88-259535494116870/
   /root/.ansible/tmp/ansible-tmp-
1511249525.88259535494116870/setup.py && sleep 0'
<localhost> EXEC /bin/sh -c '/usr/bin/python
   /root/.ansible/tmp/ansible-tmp1511249525.88-
259535494116870/setup.py; rm -rf
   "/root/.ansible/tmp/ansible-tmp1511249525.88-259535494116870/" >
/dev/null 2>&1 && sleep 0'
ok: [server1]
META: ran handlers

TASK [install-tomcat : Install Tomcat artifacts]
**********************************
********************************************************************
task path: /users/demo/vivek-playbook/roles/install-
tomcat/tasks/main.yml:5
Tuesday 21 November 2017  13:02:07 +0530 (0:00:01.515)
0:00:01.572 ******
Using module file
/usr/lib/python2.7/sitepackages/ansible/modules/packaging/os/yum.py
<localhost> ESTABLISH LOCAL CONNECTION FOR USER: root
<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'
<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo
   /root/.ansible/tmp/ansible-tmp-1511249527.34-40247177825302 `"
&& echo
   ansibletmp-1511249527.34-40247177825302="` echo
   /root/.ansible/tmp/ansible-tmp1511249527.34-40247177825302 `" )
&& sleep 0'
<localhost> PUT /tmp/tmpu83chg TO
   /root/.ansible/tmp/ansible-tmp-
1511249527.3440247177825302/yum.py
<localhost> EXEC /bin/sh -c 'chmod u+x
   /root/.ansible/tmp/ansible-tmp1511249527.34-40247177825302/
   /root/.ansible/tmp/ansible-tmp-
1511249527.3440247177825302/yum.py && sleep 0'
<localhost> EXEC /bin/sh -c '/usr/bin/python
   /root/.ansible/tmp/ansible-tmp1511249527.34-
40247177825302/yum.py; rm -rf
   "/root/.ansible/tmp/ansible-tmp1511249527.34-40247177825302/" >
/dev/null 2>
   &1 && sleep 0'
changed: [server1] => {
```

```
   "changed": true,
   "invocation": {
      "module_args": {
         "conf_file": null,
         "disable_gpg_check": false,
         "disablerepo": null,
         "enablerepo": null,
         "exclude": null,
         "install_repoquery": true,
         "installroot": "/",
         "list": null,
         "name": ["demo-tomcat-1"],
         "skip_broken": false,
         "state": "present",
         "update_cache": false,
         "validate_certs": true
      }
   },
   "msg": "",
   "rc": 0,
   "results": [
      "Loaded plugins: product-id,
      search-disabled-repos,
      subscriptionmanager\nThis system is not registered to Red Hat
Subscription Management.
      You can use subscription-manager to register.\nResolving
Dependencies\n-->
      Running transaction check\n--->
      Package demo-tomcat-1.noarch 0:SNAPSHOT-1 will be
installed\n--> Finished Dependency
      Resolution\n\nDependencies Resolved\n

\n================================================================
==============\n
      Package Arch Version Repository

Size\n================================================================
=====\nInstalling:\n
      demo-tomcat-1 noarch SNAPSHOT-1 demo-repo1 7.1
M\n\nTransaction

Summary\n================================================================
========\nInstall  1
      Package\n\nTotal download size: 7.1 M\nInstalled size: 7.9
M\nDownloading
         packages:\nRunning transaction
      check\nRunning transaction test\nTransaction test
succeeded\nRunning transaction\n  Installing :
      demotomcat-1-SNAPSHOT-1.noarch 1/1 \n  Verifying  :
```

```
      demo-tomcat-1-SNAPSHOT-1.noarch 1/1 \n\nInstalled:\n
      demo-tomcat-1.noarch 0:SNAPSHOT-1 \n\nComplete!\n"
   ]
}

TASK [install-tomcat : debug]
*************************************************************
*********************************************************************
********
task path: /users/demo/vivek-playbook/roles/install-
tomcat/tasks/main.yml:11
Tuesday 21 November 2017  13:02:13 +0530 (0:00:06.757) 0:00:08.329
******
ok: [server1] => {
   "changed": false,
   "msg": [
      "Install Tomcat artifacts task ended with message: {
         u'msg': u'', u'changed': True, u'results':
         [u'Loaded plugins: product-id,
         search-disabledrepos,
         subscription-manager\\nThis system is not registered to
Red Hat Subscription Management.
         You can use subscription-manager to register.\\nResolving
Dependencies\\n-->
         Running transaction check\\n--->
         Package demo-tomcat-1.noarch 0:SNAPSHOT-1 will be
installed\\n-->
         Finished Dependency Resolution\\n
         \\nDependencies

Resolved\\n\\n=============================================================
=============\\n
         Package Arch Version Repository

Size\\n=====================================================================
============
         =====\\nInstalling:\\n demo-tomcat-1 noarch SNAPSHOT-1
demo-repo1 7.1 M\\n\\nTransaction

Summary\\n=================================================================
\\nInstall  1
         Package\\n\\nTotal download size: 7.1 M\\nInstalled size:
7.9 M\\nDownloading
            packages:\\nRunning
         transaction check\\nRunning transaction test\\nTransaction
test succeeded\\nRunning
            transaction\\n
         Installing : demo-tomcat-1-SNAPSHOT-1.noarch 1/1 \\n
Verifying  :
```

```
         demo-tomcat-1-SNAPSHOT-1.noarch
         1/1 \\n\\nInstalled:\\n  demo-tomcat-1.noarch 0:SNAPSHOT-1
\\n\\nComplete!\\n'], u'rc': 0
      }",
      "Installed Tomcat artifacts - True"
   ]
}

TASK [install-tomcat : Clean DEMO environment]
*************************************
************************************************************
task path: /users/demo/vivek-playbook/roles/install-
tomcat/tasks/main.yml:19
Tuesday 21 November 2017  13:02:13 +0530 (0:00:00.057) 0:00:08.387
******
[WARNING]: when statements should not include jinja2 templating
delimiters such as {{ }} or
   {% %}. Found: {{installationOutput.changed}}

Using module file
/usr/lib/python2.7/sitepackages/ansible/modules/files/file.py
<localhost> ESTABLISH LOCAL CONNECTION FOR USER: root
<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'
<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo
   /root/.ansible/tmp/ansible-tmp-1511249534.13-128345805983963 `"
&& echo
   ansible-tmp-1511249534.13-128345805983963="` echo
   /root/.ansible/tmp/ansibletmp-1511249534.13-128345805983963 `" )
&& sleep 0'
<localhost> PUT /tmp/tmp0aXel7 TO
   /root/.ansible/tmp/ansible-tmp-
1511249534.13128345805983963/file.py
<localhost> EXEC /bin/sh -c 'chmod u+x
   /root/.ansible/tmp/ansible-tmp1511249534.13-128345805983963/
   /root/.ansible/tmp/ansible-tmp-
1511249534.13128345805983963/file.py && sleep 0'
<localhost> EXEC /bin/sh -c '/usr/bin/python
   /root/.ansible/tmp/ansible-tmp1511249534.13-
128345805983963/file.py; rm -rf
   "/root/.ansible/tmp/ansible-tmp1511249534.13-128345805983963/" >
/dev/null 2>&1
   && sleep 0'
changed: [server1] => {
   "changed": true,
      "diff": {
         "after": {
            "path": "/users/demo/DEMO",
            "state": "absent"
      },
```

```
      "before": {
         "path": "/users/demo/DEMO",
         "state": "directory"
      }
   },

   "invocation": {
      "module_args": {
         "attributes": null,
         "backup": null,
         "content": null,
         "delimiter": null,
         "diff_peek": null,
         "directory_mode": null,
         "follow": false,
         "force": false,
         "group": null,
         "mode": null,
         "original_basename": null,
         "owner": null,
         "path": "/users/demo/DEMO",
         "recurse": false,
         "regexp": null,
         "remote_src": null,
         "selevel": null,
         "serole": null,
         "setype": null,
         "seuser": null,
         "src": null,
         "state": "absent",
         "unsafe_writes": null,
         "validate": null
      }
   },
   "path": "/users/demo/DEMO",
   "state": "absent"
}

TASK [install-tomcat : debug]
*********************************************************
*********************************************************
task path: /users/demo/vivek-playbook/roles/install-
tomcat/tasks/main.yml:29
Tuesday 21 November 2017  13:02:14 +0530 (0:00:00.257)
0:00:08.645 ******
ok: [server1] => {
   "changed": false,
   "msg": [
```

```
      "Clean DEMO environment task ended with message:{u'diff':
{u'after': {u'path':
         u'/users/demo/DEMO', u'state': u'absent'},
      u'before': {u'path': u'/users/demo/DEMO', u'state':
u'directory'}}, u'state': u'absent',
         u'changed': True, u'path': u'/users/demo/DEMO'}",
      "check value  :True"
   ]
}

TASK [install-tomcat : Copy Tomcat to user home]
************************************
*************************************************
task path: /users/demo/vivek-playbook/roles/install-
tomcat/tasks/main.yml:37
Tuesday 21 November 2017  13:02:14 +0530 (0:00:00.055)
0:00:08.701 ******
[WARNING]: when statements should not include jinja2 templating
delimiters such as {{ }} or
   {% %}. Found: {{installationOutput.changed}}

Using module file
/usr/lib/python2.7/sitepackages/ansible/modules/commands/command.py
<localhost> ESTABLISH LOCAL CONNECTION FOR USER: root
<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'
<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo
   /root/.ansible/tmp/ansible-tmp-1511249534.43-41077200718443 `"
&& echo
   ansibletmp-1511249534.43-41077200718443="` echo
   /root/.ansible/tmp/ansible-tmp1511249534.43-41077200718443 `" )
&& sleep 0'
<localhost> PUT /tmp/tmp25deWs TO
   /root/.ansible/tmp/ansible-tmp-
1511249534.4341077200718443/command.py
<localhost> EXEC /bin/sh -c 'chmod u+x
   /root/.ansible/tmp/ansible-tmp1511249534.43-41077200718443/
   /root/.ansible/tmp/ansible-tmp-
1511249534.4341077200718443/command.py && sleep 0'
<localhost> EXEC /bin/sh -c '/usr/bin/python
   /root/.ansible/tmp/ansible-tmp1511249534.43-
41077200718443/command.py; rm -rf
   "/root/.ansible/tmp/ansibletmp-1511249534.43-41077200718443/" >
/dev/null 2>&1
   && sleep 0'
changed: [server1] => {
   "changed": true,
   "cmd": [
      "cp",
      "-r",
```

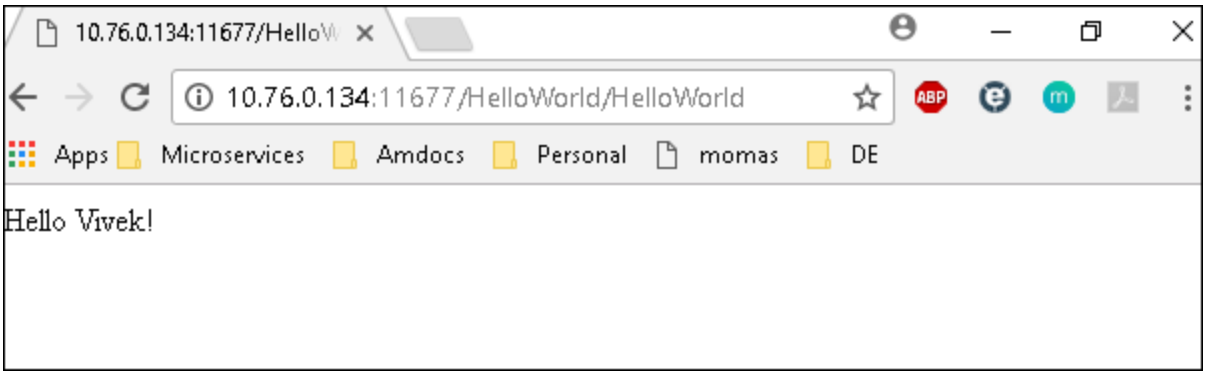
```
      "/opt/ansible/tomcat/demo",
      "/users/demo/DEMO/"
   ],
   "delta": "0:00:00.017923",
   "end": "2017-11-21 13:02:14.547633",
   "invocation": {
      "module_args": {
         "_raw_params": "cp -r /opt/ansible/tomcat/demo
/users/demo/DEMO/",
         "_uses_shell": false,
         "chdir": null,
         "creates": null,
         "executable": null,
         "removes": null,
         "warn": true
      }
   },
   "rc": 0,
   "start": "2017-11-21 13:02:14.529710",
   "stderr": "",
   "stderr_lines": [],
   "stdout": "",
   "stdout_lines": []
}

TASK [install-tomcat : debug]
*********************************************************
*********************************************************
task path: /users/demo/vivek-playbook/roles/install-
tomcat/tasks/main.yml:47
Tuesday 21 November 2017  13:02:14 +0530 (0:00:00.260)
0:00:08.961 ******
ok: [server1] => {
   "changed": false,
   "msg": "Copy Tomcat to user home task ended with message {
      'stderr_lines': [], u'changed': True, u'end': u'2017-11-21
13:02:14.547633', u'stdout':
      u'', u'cmd': [u'cp', u'-r', u'/opt/ansible/tomcat/demo',
u'/users/demo/DEMO/'], u'rc': 0,
      u'start': u'2017-11-21 13:02:14.529710', u'stderr': u'',
u'delta': u'0:00:00.017923',
      'stdout_lines': []}"
}

TASK [start-tomcat : Start Tomcat]
*********************************************
*********************************************************
task path: /users/demo/vivek-playbook/roles/start-
tomcat/tasks/main.yml:5
```

```
Tuesday 21 November 2017  13:02:14 +0530 (0:00:00.044)
0:00:09.006 ******
Using module file
/usr/lib/python2.7/sitepackages/ansible/modules/commands/command.py
<localhost> ESTABLISH LOCAL CONNECTION FOR USER: root
<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'
<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo
   /root/.ansible/tmp/ansible-tmp-1511249534.63-46501211251197 `"
&& echo
   ansibletmp-1511249534.63-46501211251197="` echo
   /root/.ansible/tmp/ansible-tmp1511249534.63-46501211251197 `" )
&& sleep 0'
<localhost> PUT /tmp/tmp9f06MQ TO
   /root/.ansible/tmp/ansible-tmp-
1511249534.6346501211251197/command.py
<localhost> EXEC /bin/sh -c 'chmod u+x
   /root/.ansible/tmp/ansible-tmp1511249534.63-46501211251197/
   /root/.ansible/tmp/ansible-tmp-
1511249534.6346501211251197/command.py && sleep 0'
<localhost> EXEC /bin/sh -c '/usr/bin/python
   /root/.ansible/tmp/ansible-tmp1511249534.63-
46501211251197/command.py; rm -rf
   "/root/.ansible/tmp/ansibletmp-1511249534.63-46501211251197/" >
/dev/null 2>&1
   && sleep 0'
changed: [server1] => {
   "changed": true,
   "cmd": [ "/users/demo/DEMO/bin/startup.sh" ],
   "delta": "0:00:00.020024",
   "end": "2017-11-21 13:02:14.741649",
   "invocation": {
      "module_args": {
         "_raw_params": "/users/demo/DEMO/bin/startup.sh",
         "_uses_shell": false,
         "chdir": null,
         "creates": null,
         "executable": null,
         "removes": null,
         "warn": true
      }
   },
   "rc": 0,
   "start": "2017-11-21 13:02:14.721625",
   "stderr": "",
   "stderr_lines": [],
   "stdout": "Tomcat started.",
   "stdout_lines": [ "Tomcat started." ]
}
```

```
TASK [start-tomcat : debug]
*********************************************
*****************************************************************
***
task path: /users/demo/vivek-playbook/roles/start-
tomcat/tasks/main.yml:10
Tuesday 21 November 2017  13:02:14 +0530 (0:00:00.150)
0:00:09.156 ******
ok: [server1] => {
   "changed": false,
   "msg": [
      "Start Tomcat task ended with message: {'
         stderr_lines': [], u'changed': True, u'end': u'2017-11-21
13:02:14.741649', u'stdout':
         u'Tomcat started.', u'cmd':
[u'/users/demo/DEMO/bin/startup.sh'], u'rc': 0, u'start':
         u'2017-11-21 13:02:14.721625', u'stderr': u'', u'delta':
u'0:00:00.020024',
         'stdout_lines': [u'Tomcat started.']}",
      "Tomcat started - True"
   ]
}
META: ran handlers
META: ran handlers

PLAY RECAP
*****************************************************************
************
**************************************************
server1  : ok = 9    changed = 4    unreachable = 0    failed = 0

Tuesday 21 November 2017  13:02:14 +0530 (0:00:00.042)
0:00:09.198 ******
===============================================================
============
install-tomcat : Install Tomcat artifacts ------------------------
------ 6.76s
/users/demo/vivek-playbook/roles/install-tomcat/tasks/main.yml:5 --
------------
Gathering Facts --------------------------------------------------
------ 1.52s
 ----------------------------------------------------------------
------------
install-tomcat : Copy Tomcat to user home ------------------------
------ 0.26s
/users/demo/vivek-playbook/roles/install-tomcat/tasks/main.yml:37 -
------------
```

```
install-tomcat : Clean DEMO environment ---------------------------
------ 0.26s
/users/demo/vivek-playbook/roles/install-tomcat/tasks/main.yml:19 -
------------

start-tomcat : Start Tomcat ---------------------------------------
------ 0.15s
/users/demo/vivek-playbook/roles/start-tomcat/tasks/main.yml:5 ----
------------

install-tomcat : debug --------------------------------------------
------ 0.06s
/users/demo/vivek-playbook/roles/install-tomcat/tasks/main.yml:11 -
------------

install-tomcat : debug --------------------------------------------
------ 0.06s
/users/demo/vivek-playbook/roles/install-tomcat/tasks/main.yml:29 -
------------

install-tomcat : debug --------------------------------------------
------ 0.04s
/users/demo/vivek-playbook/roles/install-tomcat/tasks/main.yml:47 -
------------

start-tomcat : debug ----------------------------------------------
------ 0.04s
/users/demo/vivek-playbook/roles/start-tomcat/tasks/main.yml:10 ---
------------
```

Hit the following URL and you will be directed to a page as shown below − **http://10.76.0.134:11677/HelloWorld/HelloWorld**



The deployed war just has a servlet which displays “Hello World”. The detailed output shows the time taken by each and every task because of the entry added in ansible.cfg file −

```
[defaults]
callback_whitelist = profile_tasks
```

Ansible - 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.

```
---
#Tsting
- 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.

Ansible - Advanced Execution

In this chapter, we will learn what is advanced execution with Ansible.

How to Limit Execution by Tasks

This is a very important execution strategy where one needs to execute only one execution and not the entire playbook. **For example**, suppose you only want to stop a server (in case a production issue comes) and then post applying a patch you would like to only start the server.

Here in original playbook stop and start were a part of different roles in the same playbook but this can be handled with the usage of tags. We can provide different tags to different roles (which in turn will have tasks) and hence based on the tags provided by the executor only that specified role/task gets executed. So for the above example provided, we can add tags like the following −

```
- {role: start-tomcat, tags: ['install']}}
```

The following command helps in using tags −

```
ansible-playbook -i hosts <your yaml> --tags "install" -vvv
```

With the above command, only the start-tomcat role will be called. The tag provided is case-sensitive. Ensure exact match is being passed to the command.

How to Limit Execution by Hosts

There are two ways to achieve the execution of specific steps on specific hosts. For a specific role, one defines the hosts - as to which specific hosts that specific role should be run.

Example

```
- hosts: <A>
   environment: "{{your env}}"
   pre_tasks:
      - debug: msg = "Started deployment.
```

```
      Current time is {{ansible_date_time.date}}
{{ansible_date_time.time}} "

   roles:
      - {role: <your role>, tags: ['<respective tag>']}
   post_tasks:
      - debug: msg = "Completed deployment.
      Current time is {{ansible_date_time.date}}
{{ansible_date_time.time}}"

- hosts: <B>
   pre_tasks:
      - debug: msg = "started....
      Current time is {{ansible_date_time.date}}
{{ansible_date_time.time}} "

   roles:
      - {role: <your role>, tags: ['<respective tag>']}
   post_tasks:
      - debug: msg = "Completed the task..
      Current time is {{ansible_date_time.date}}
{{ansible_date_time.time}}"
```

As per the above example, depending on the hosts provided, the respective roles will only be called. Now my hosts A and B are defined in the hosts (inventory file).

Alternate Solution

A different solution might be defining the playbook's hosts using a variable, then passing in a specific host address via **--extra-vars** −

```
# file: user.yml   (playbook)
---
- hosts: '{{ target }}'
   user: ...
playbook contd….
```

Running the Playbook

```
ansible-playbook user.yml --extra-vars "target = "<your host
variable>"
```

If {{ target }} isn't defined, the playbook does nothing. A group from the hosts file can also be passed through if need be. This does not harm if the extra vars is not provided.

Playbook targeting a single host

```
$ ansible-playbook user.yml --extra-vars "target = <your hosts
variable>" --listhosts
```

Ansible - Troubleshooting

The most common strategies for debugging Ansible playbooks are using the modules given below −

Debug and Register

These two are the modules available in Ansible. For debugging purpose, we need to use the two modules judiciously. Examples are demonstrated below.

Use Verbosity

With the Ansible command, one can provide the verbosity level. You can run the commands with verbosity level one (-v) or two (-vv).

Important Points

In this section, we will go through a few examples to understand a few concepts.

If you are not quoting an argument that starts with a variable. For example,

```
vars:
   age_path: {{vivek.name}}/demo/

{{vivek.name}}
```

This will throw an error.

Solution

```
vars:
   age_path: "{{vivek.name}}/demo/" - marked in yellow is the fix.

How to use register -> Copy this code into a yml file say test.yml
and run it
---
#Tsting
- hosts: tomcat-node
   tasks:

   - shell: /usr/bin/uptime
      register: myvar
      - name: Just debugging usage
         debug: var = myvar
```

When I run this code via the command Ansible-playbook -i hosts test.yml, I get the output as shown below.

If you see the yaml , we have registered the output of a command into a variable – **myvar** and just printed the output.

The text marked yellow, tells us about property of the variable –myvar that can be used for further flow control. This way we can find out about the properties that are exposed of a particular variable. The following debug command helps in this.

```
$ ansible-playbook -i hosts test.yml

PLAY [tomcat-node]
*****************************************************************
```

```
****************
*************************************************************
*************** ******************************

TASK [Gathering Facts]
*************************************************************
**************
*************************************************************
************** ************************
Monday 05 February 2018  17:33:14 +0530 (0:00:00.051) 0:00:00.051
*******
ok: [server1]

TASK [command]
*************************************************************
*************
*************************************************************
************* *********************************
Monday 05 February 2018  17:33:16 +0530 (0:00:01.697) 0:00:01.748
*******
changed: [server1]

TASK [Just debugging usage]
*************************************************************
*************
*************************************************************
************* *********************
Monday 05 February 2018  17:33:16 +0530 (0:00:00.226) 0:00:01.974
*******
ok: [server1] => {
   "myvar": {
      "changed": true,
      "cmd": "/usr/bin/uptime",
      "delta": "0:00:00.011306",
      "end": "2018-02-05 17:33:16.424647",
      "rc": 0,
      "start": "2018-02-05 17:33:16.413341",
      "stderr": "",
      "stderr_lines": [],
      "stdout": " 17:33:16 up 7 days, 35 min,  1 user,  load
average: 0.18, 0.15, 0.14",
      "stdout_lines": [
         " 17:33:16 up 7 days, 35 min,  1 user,  load average:
0.18, 0.15, 0.14"
      ]
   }
}
```

```
PLAY RECAP
*********************************************************************
*********
*********************************************************************
***************
 ************************************
server1 : ok = 3    changed = 1    unreachable = 0    failed = 0
```

Common Playbook Issues

In this section, we will learn about the a few common playbook issues. The issues are −

- Quoting
- Indentation

Playbook is written in yaml format and the above two are the most common issues in yaml/playbook.

Yaml does not support tab based indentation and supports space based indentation, so one needs to be careful about the same.

Note − once you are done with writing the yaml , open this site(https://editor.swagger.io/) and copy paste your yaml on the left hand side to ensure that the yaml compiles properly. This is just a tip.

Swagger qualifies errors in warning as well as error.