**Ad-hoc Commands**

Ad-hoc commands are those that need to be typed in order to do it quickly but are not saved for later. In case we need to reboot all our servers, then we will have to run the ad-hoc commands from **/usr/bin/ansible.**

These commands are one-time-usage commands and are not used for configuration management and deployment. Only Ansible playbook can be used for configuration management and deployment.

### **Ansible Playbooks Commands**

### Ansible code is written in playbooks in the YAML (Yet Another Markup Language) format. YAML is a script type language; we can use any editor to write YAML files, but notepad++ will be the easiest one.

Let’s now check out the basic syntax of the playbook with an example. A typical YAML file starts with three hyphens (**—**).

---

**name: install & configure Intellipaat DB**

**hosts: testServers**

**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: <our service name>**

We can save the above file as first .yml. We need to follow the correct indentation while writing in YAML and be a little careful about its syntax.

* **Name:**It specifies the name of our Ansible playbook. Based on what this particular playbook will do, we can give it a relevant name.
* **Host:**It specifies the list of hosts or host groups against which we will run our task. This host tag is compulsory as it communicates with Ansible and enables it to run the listed tasks. These tasks can be run either on the same machine or on remote machines, or we can run the tasks on multiple machines thereby hosts tags can enable the entry of a group of hosts.
* **Vars:**Using this tag, we can easily define the variable we have used in our playbook; its usage is similar to the variables in any of the programming languages.
* **Tasks:**Playbooks must contain tasks or a list of tasks that are to be executed. They are basically a list of actions that the playbook needs to perform. The task field contains the name of the task. Every task links to a piece of code called a module. The module and the arguments (in the module), both should be executed.

# **Ansible Shell**

Ansible shell module is designed to execute the shell commands against the target UNIX based hosts. Ansible can run except any high complexes commands with pipes, redirection. And you can also perform the shell scripts using the Ansible shell module.

The main advantage of the Ansible shell is except any high complexes commands with pipes and semicolons can be a disadvantage from the security perspective as a single mistake could cost a lot and break the system integrity.

* The Ansible shell module is designed to work only with LINUX based machines and not for the windows. For windows, you should use the**win\_shell**
* Ansible shell module can be used to execute shell scripts. Ansible has a dedicated module named script, which is used to copy the shell script from the control machine to the remote server.

# **Ansible Debug**

Ansible provides a debug module option that makes the tasks more manageable. It is a handy tool to figure out any problem areas.

Ansible version 2.1 extended the debug module with a verbosity parameter that transforms it from a print line.

**For example:** Let's create the playbook **1\_debug\_example.yml**, such as:

1. ---
2. - name: Debug Example Uptime
3. hosts: localhost
4. connection: local
6. tasks:
7. - name: Find Uptime
8. shell: /usr/bin/uptime
9. register: result
11. - name: Print debug message
12. debug:
13. var: result
14. verbosity: 2

To print a message from the Ansible playbook, as well as a value of a variable, we can use the Ansible debug module. Ansible debug module is easy to use.

**For example:** Let's execute a simple hello world playbook **2\_debug\_example.yml**, such as:

1. ---
2. - name: Debug Example - Hello World
3. hosts: localhost
4. tasks:
5. - name: Print debug message
6. debug:

The Ansible includes a debugger as a part of the strategy plugins. This debugger enables you to debug as a task. You have access to all the features of the debugger in the context of the task. You can check or set the value of variables, update module arguments, and re-run the task with the new variables and arguments to resolve the cause of the failure.

There are many ways to invoke the debugger, such as:

**Using the debugger Keyword**

The debugger keyword can be used on any block where you provide a name attribute such as a role, block, task, or play.

The debugger keyword accepts several values, such as:

**Always:** Always invokes the debugger, regardless of the outcome.

**Never:** Never invokes the debugger, regardless of the outcome.

**On\_failed:** It only invokes the debugger if a task fails.

**On\_unreachable:** It only invokes the debugger if the host was unreachable.

**On\_skipped:** It only invokes the debugger if the task is skipped.