1

5

# Ansible changed_when and failed_when examples

UPDATED ON: JANUARY 5, 2021 • SARAV AK

92% 0

In this post, we are going to see how to use conditional statements of Ansible such as **when**, **changed_when**, **failed_when** and where to use them appropriately and how it works. By these conditional modules, Ansible provides a way for us to define when should ansible run a certain task or consider the executed task as Success or failure.

Long Story Short, These modules give us a way to make ansible do something when a certain condition is met or satisfied

let us cover each conditional statements one by one with examples.

We presume that you have all the basic knowledge of Ansible. If not, we highly recommend you to refer the following articles and come back.

Ansible Basics: What is Ansible, Ad hoc commands and Playbooks

Ansible In Action: How setup your own ansible infrastructure using Vagrant and run your playbook

These articles can help you to get started with Ansible.

.nd more ansible playbook examples here

## Table of Contents

1. The Ansible when Statement
2. How to use ansible when Statement
    2.1. Example 1: Shutdown the Debian flavoured servers
    2.2. Example 2: Install HTTPD when httpd is not installed already
    2.3. Example 3: [Multiple Conditions in Single when statement] Shutdown only CentOS-6 and Debian-7
3. The Ansible failed_when and changed_when Statements
4. How to use ansible changed_when Statement
    4.1. Example 1: Start the HTTPD (or) Apache Server which is already started
5. Example 2: Install Dependencies via PHP Composer
6. How to use ansible failed_when Statement
    6.1. Example 1: System Requirement / Prerequisite check before Installation

## The Ansible when Statement

Ansible when statement is more like if statement in any given programming language. It evaluates if a condition is met or satisfied.

Consider yourself having any of the following requirements

1. You want to run a task only in a specific box
2. You want to run a task based on the output of another task
3. You want to skip or proceed with an installation when the OS version is Linux or Ubuntu
4. You want to perform some disk clean up tasks when a threshold is reached
5. You want to control the execution when a certain value is reached in Loop

More and more. These are all the few test cases or real-time usage scenarios, I could think of. I m certain, there would be more explored and unexplored possibilities for these conditional tements.

## How to use ansible when Statement

We are going to provide various examples of how to use ansible when statement , You can choose to read whichever example you would like to read

### Example 1: Shutdown the Debian flavoured servers

In the following playbook, we have used a when statement and a command to execute. The Command will only execute when the defined condition is satisfied, which is exactly *when the Operating system of the host is Debian*

```
tasks:
  - name: "shut down Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_os_family == "Debian"
```

## Example 2: Install HTTPD when httpd is not installed already

92%

Being aware that shell would return **not found** error message when the given command is not installed. We are using it as our search keyword to determine the installation of httpd.In this example, we are first making sure whether “Apache httpd” is already installed by running the httpd command over the shell. The output of the command is saved into a register variable named “`validatedhttpd`”

When we run the following playbook, The first task will run the command httpd and the result of the command will be stored in “`validatehttpd`” register variable.

Then the second task, which is to install httpd using yum module. Will first execute the when condition we have specified and see if it is TRUE.

Therefore, If there is a “not found” error in the register variable. The HTTPD will be installed. If there is anything else. HTTPD will not be installed and the task will be skipped.

```
---
- hosts: web
  tasks:
  - name: "Determine if the HTTPD is installed"
    register: validatehttpd
    shell: httpd

  - name: Ensure Apache is at the Latest version
    become: yes
    become_user: root
    yum:
      name: httpd
      state: latest
    when: 'not found' in validatehttpd.stdout
```

## Example 3: [Multiple Conditions in Single when statement] Shutdown only CentOS-6 and Debian-7

Our requirement here is to shut down only CentOS6 and Debian7 version flavoured systems. So our conditional statement should be as followsAs said earlier,  when is more like an if statement so it should support the multiple conditions in a single validation.

```
(ansible_distribution == "CentOS" and ansible_distribution_major_version ==
(ansible_distribution == "Debian" and ansible_distribution_major_version ==
```

> Here we have used parenthesis ( ) for a grouping

The Playbook is given below

```
---
- hosts: all
  tasks:
  - name: "shut down CentOS 6 and Debian 7 systems"
    command: /sbin/shutdown -t now
    when: (ansible_distribution == "CentOS" and ansible_distribution_major_
          (ansible_distribution == "Debian" and ansible_distribution_major_
```

If our requirement is just to validate, If the OS distribution is **CentOS** and version is **6** , We could write it like this

```
tasks:
  - name: "shut down CentOS 6 systems"
    command: /sbin/shutdown -t now
    when:
      - ansible_distribution == "CentOS"
      - ansible_distribution_major_version == "6"
```

What we do here is mentioning our conditions in a list and *ALL SHOULD BE TRUE* for this task to run

1

5

## ɾhe Ansible failed_when and changed_when Statements

nsible failed_when and changed_when statements are similar to ansible when statement. The ɔnly difference is that It will mark the task as **failed** or **Success**[changed], when the condition ɹefined, is met or satisfied.

92% : primary purpose of the `failed_when` and `changed_when` statements are to determine whether the task is actually successful or failure

Consider you are running a `command` or `shell` module with some complex script or a simple command.

> Ansible would report it as `changed` as long as the command (or) script give ZERO return code.

But how would you decide whether the command or script ran successfully? or met your needs?

For example, When you are starting a Weblogic (or) Tomcat servers using some shell script or command. Ansible would invoke the script and consider it as done (or) changed. But you would never know whether its actually true until you re-validate with `wait_for` or `debug` module

Let us see some real time practical examples for both failed_when and changed_when statements in the upcoming section

1

5

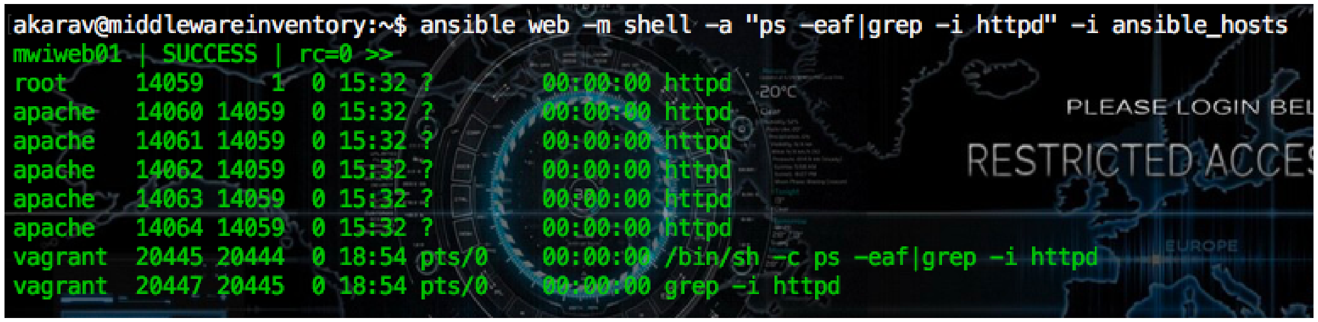
## How to use ansible changed_when Statement

### Example 1: Start the HTTPD (or) Apache Server which is already started

92%

Having said that, Let's start with our trialWe are going to start the HTTPD (or) Apache Server which is already running. Ideally, If it is already running it should not report as **changed**

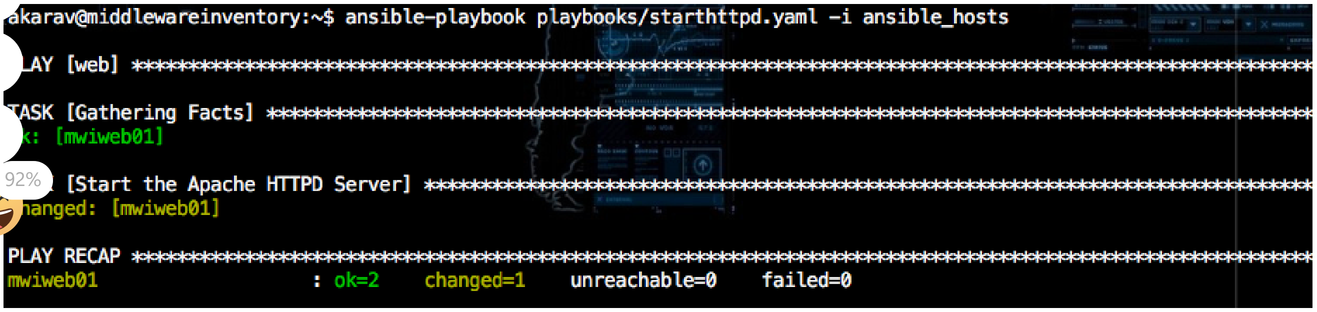
**Step 1:** Making sure it is already running by invoking the `ps -eaf|grep -i httpd` command as `ad-hoc`



```
akarav@middlewareinventory:~$ ansible web -m shell -a "ps -eaf|grep -i httpd" -i ansible_hosts
mwiweb01 | SUCCESS | rc=0 >>
root      14059     1  0 15:32 ?        00:00:00 httpd
apache    14060 14059  0 15:32 ?        00:00:00 httpd
apache    14061 14059  0 15:32 ?        00:00:00 httpd
apache    14062 14059  0 15:32 ?        00:00:00 httpd
apache    14063 14059  0 15:32 ?        00:00:00 httpd
apache    14064 14059  0 15:32 ?        00:00:00 httpd
vagrant   20445 20444  0 18:54 pts/0    00:00:00 /bin/sh -c ps -eaf|grep -i httpd
vagrant   20447 20445  0 18:54 pts/0    00:00:00 grep -i httpd
```

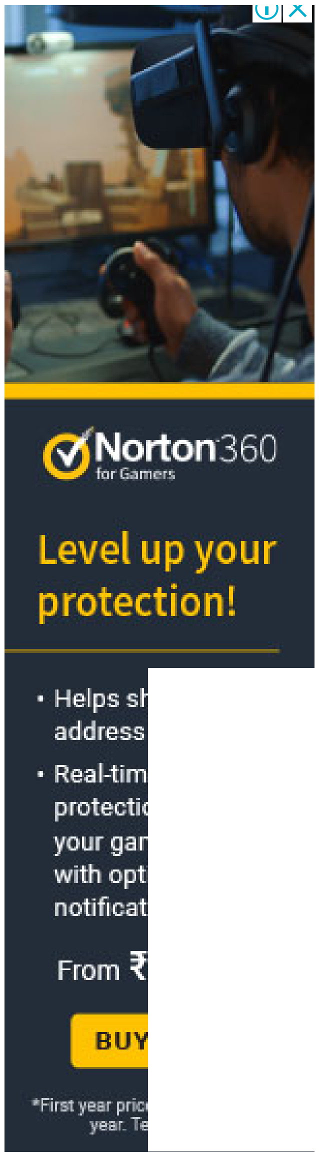
**Step 2:** Use the following playbook with the task to start the HTTPD server

```
---
- hosts: web
  tasks:
   - name: "Start the Apache HTTPD Server"
     become: true
     become_user: root
     shell: "httpd -k start"
```



```
akarav@middlewareinventory:~$ ansible-playbook playbooks/starthttpd.yaml -i ansible_hosts

LAY [web] ************************************************************************************************

TASK [Gathering Facts] ***********************************************************************************
k: [mwiweb01]

  [Start the Apache HTTPD Server] ************************************************************************
hanged: [mwiweb01]

PLAY RECAP ***********************************************************************************************
mwiweb01                   : ok=2    changed=1    unreachable=0    failed=0
```

As you could see in the preceding execution output snapshot, The task to start the httpd server has been marked as changed despite it did not actually start the apache and it was already running.



**tep 2a:** Modified Playbook with Debug Enabled

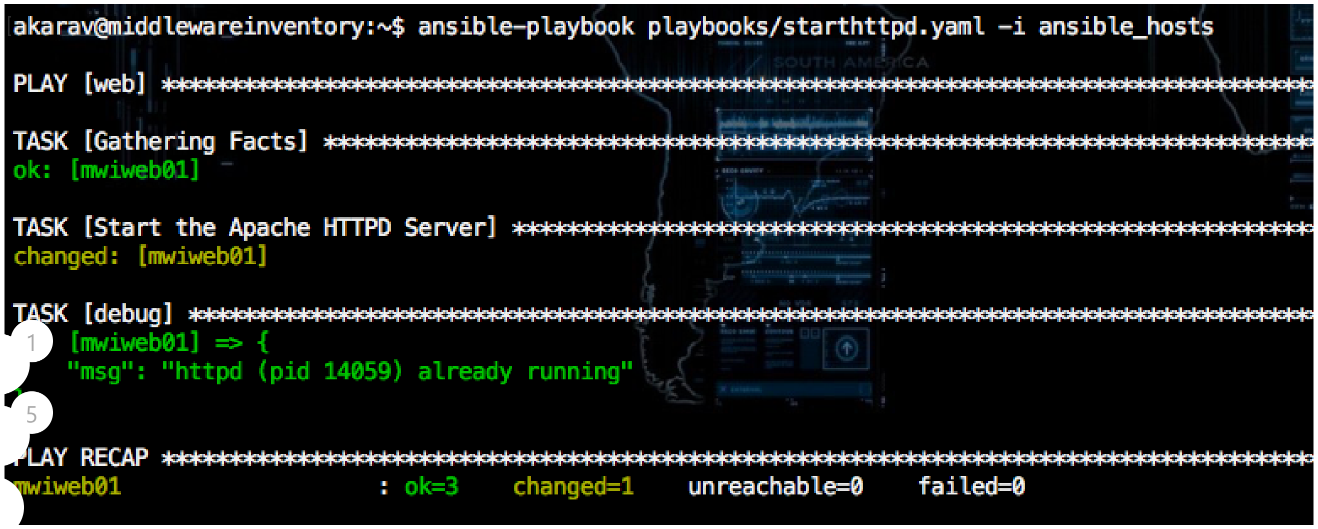
To know the truth, you should modify the playbook with a debug and register as follows

```
---
- hosts: web
  tasks:
    - name: "Start the Apache HTTPD Server"
      become: true
      become_user: root
      register: starthttpdout
      shell: "httpd -k start"


    - debug:
        msg: "{{starthttpdout.stdout}}"
```

The execution output of the preceding playbook is given below

**SEARCH**

Search ...



```
akarav@middlewareinventory:~$ ansible-playbook playbooks/starthttpd.yaml -i ansible_hosts

PLAY [web] ********************************************************************************

TASK [Gathering Facts] ********************************************************************
ok: [mwiweb01]

TASK [Start the Apache HTTPD Server] ******************************************************
changed: [mwiweb01]

TASK [debug] ******************************************************************************
ok: [mwiweb01] => {
    "msg": "httpd (pid 14059) already running"
}

PLAY RECAP ********************************************************************************
mwiweb01                   : ok=3    changed=1    unreachable=0    failed=0
```

Hope the execution output is already self-explanatory.  You could see the ansible considering the task as changed when it has not actually started it

Here comes the `changed_when` to explicitly tell ansible when to consider the task as successful (or) changed

Let us re modify our same Playbook with `changed_when`

**Step 3:** The Modified Playbook with changed_when

```
---
- hosts: web
  tasks:
  - name: "Start the Apache HTTPD Server"
    become: true
    become_user: root
    register: starthttpdout
    shell: "httpd -k start"
```

## CATEGORIES

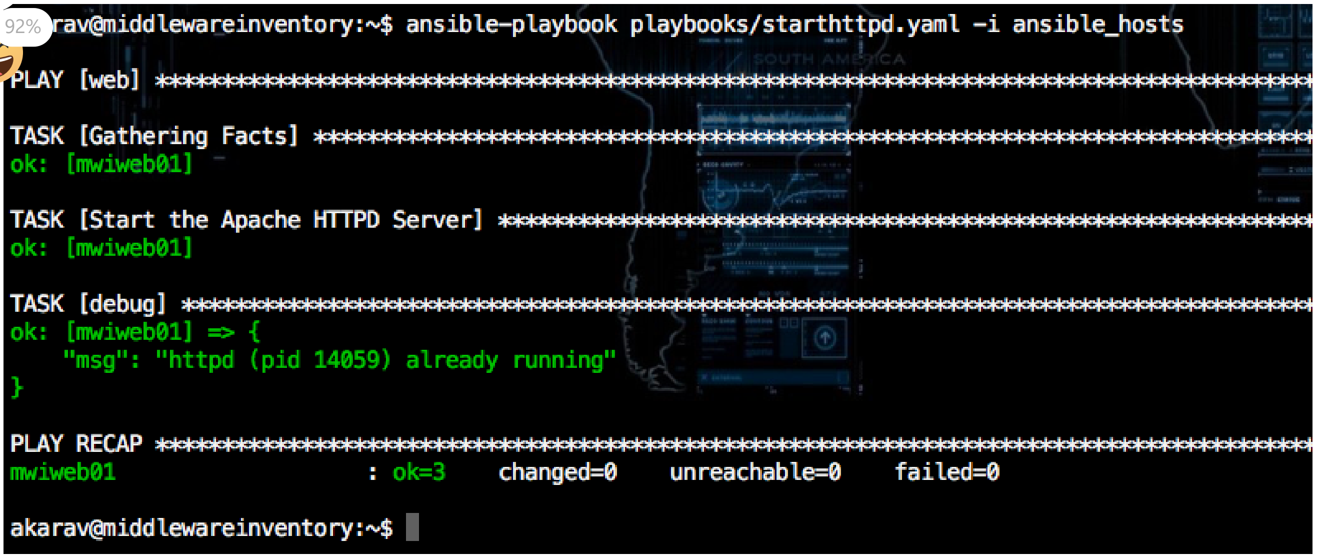
- ActiveMQ (3)
- Ansible (47)
- Apache (8)
- AWS (11)
- Best Practice (5)
- BitBucket (1)
- Cloud (1)
- Database (1)
- Development (3)
- Docker (11)
- EFK (1)
- Elastic Search (1)
- F5-Big-IP (6)
- FluentD (2)
- GCP – GoogleCloud (1)
- Graphite (1)
- IBM Websphere (1)
- IHS and Apache (10)
- IIS (4)
- JavaScript (3)
- Jenkins (2)
- Joomla (1)
- Kubernetes (4)
- Middleware (2)
- Network Troubleshooting (1)
- Networking (1)
- NodeJS (2)
- Office365 (1)

```
      changed_when: "'already running' not in starthttpdout.stdout"

    - debug:
        msg: "{{starthttpdout.stdout}}"
```

e have just added a single line to our previous version of playbook.

changed_when: "'already running' is not in starthttpdout.stdout"

he Execution Output of our new playbook is given below



```
rav@middlewareinventory:~$ ansible-playbook playbooks/starthttpd.yaml -i ansible_hosts

PLAY [web] ************************************************************************************

TASK [Gathering Facts] ************************************************************************
ok: [mwiweb01]

TASK [Start the Apache HTTPD Server] **********************************************************
ok: [mwiweb01]

TASK [debug] **********************************************************************************
ok: [mwiweb01] => {
    "msg": "httpd (pid 14059) already running"
}

PLAY RECAP ************************************************************************************
mwiweb01                   : ok=3    changed=0    unreachable=0    failed=0

akarav@middlewareinventory:~$
```

Now you can notice that the task is GREEN, not YELLOW. In other words, It is **unchanged**

- OpenSource (2)
- Oracle DB (2)
- Oracle Weblogic (2)
- Packer (2)
- Perl (1)
- PHP (1)
- PowerShell (1)
- problem/solution (8)
- Python Flask (1)
- python_scripts (7)
- Shell Scripting (3)
- SOAP Web Services (1)
- Splunk (1)
- StatsD (1)
- Terraform (4)
- tomcat (6)
- tools (4)
- Traefik (1)
- Troubleshooting (3)
- Uncategorized (4)
- Unix (5)
- Vagrant (4)
- VueJS (2)
- Vulnerabilities & Fixes (21)
- weblogic (50)
- Webserver (1)
- Websphere (7)
- Windows Server (4)
- wlst (7)
- wordpress (2)
- wsadmin scripts (1)

## Example 2:  Install Dependencies via PHP Composer

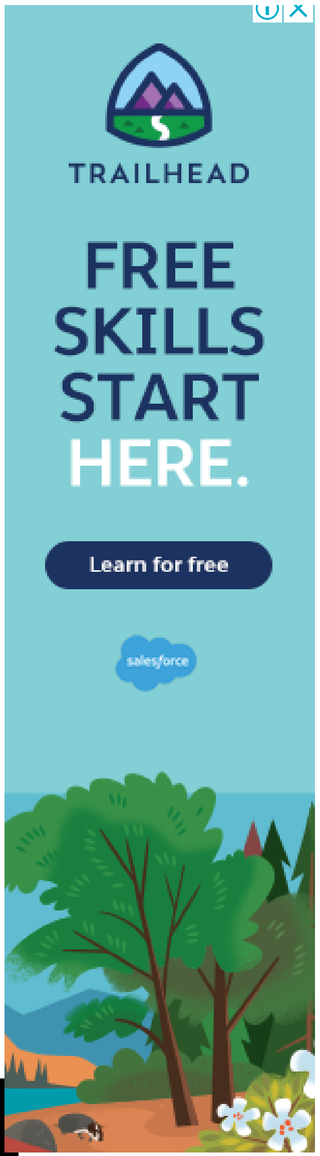
When using PHP Composer as a command to install project dependencies, it's useful to know when Composer installed something, or when nothing changed. Here's an example:

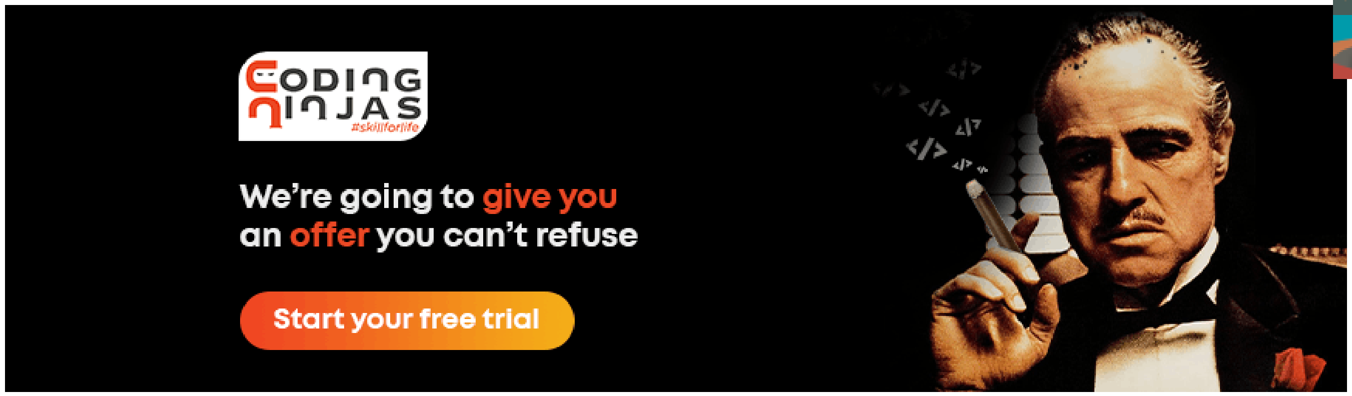
```
- name: Install dependencies via Composer.
  command: "/usr/local/bin/composer global require phpunit/phpunit --prefer
  register: composer
  changed_when: "'Nothing to install or update' not in composer.stdout"
```

92%

ou can see we used `register` to store the results of the command, then we checked whether a certain string was in the registered variable's stdout.

Only when Composer doesn't do anything it will print "Nothing to install or update", so we use that string to determine or to tell Ansible if the task resulted in a change.





## How to use ansible failed_when Statement

### Example 1: System Requirement / Prerequisite check before Installation

This one is a more real-time scenario every one of us might have come across, During the installation of software, in midway, the installation wizard will fail stating that there is no ough memory (or) the minimum system requirements to install that specific software is not m t

s we all know, Every Software needs some minimum system requirements. When they are not met, The installation would fail.

n our case, We are going to take weblogic application server installation as an example.

s per oracle recommendation for weblogic 12c to function properly and for hassle-free stallation, The system must meet the following requirement

- 2 GB of Physical Memory ( RAM)
- Minimum 4 GB of Disk space in Domain Directory `/opt`
- Minimum 1 GB of disk space in `/tmp` directory

Now we are going to perform a quick pre-requisite check using ansible **failed_when** and determine whether the system requirement specified above are met

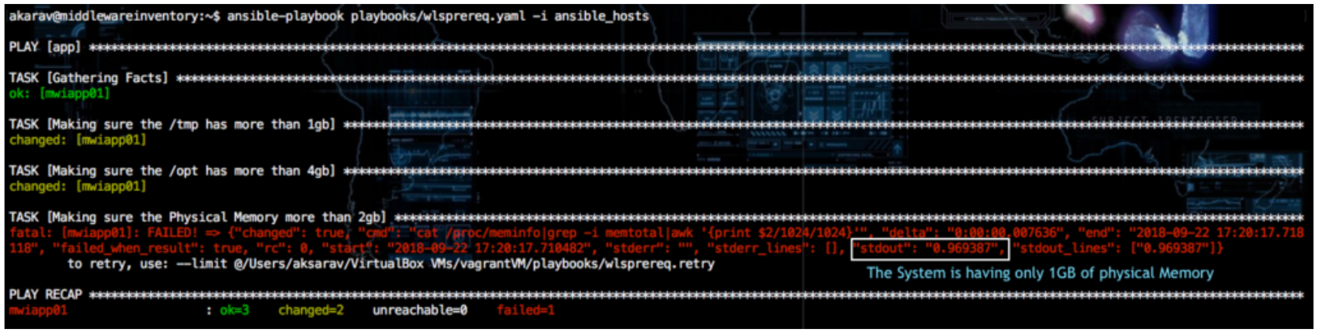
Consider the following playbook

```
---
- hosts: app
  tasks:
    - name: Making sure the /tmp has more than 1gb
      shell: "df -h /tmp|grep -v Filesystem|awk '{print $4}'|cut -d G -f1"
      register: tmpspace
      failed_when: "tmpspace.stdout|float < 1"

    - name: Making sure the /opt has more than 4gb
      shell: "df -h /opt|grep -v Filesystem|awk '{print $4}'|cut -d G -f1"
      register: tmpspace
      failed_when: "tmpspace.stdout|float < 4"

    - name: Making sure the Physical Memory more than 2gb
      shell: "cat /proc/meminfo|grep -i memtotal|awk '{print $2/1024/1024}'"
      register: memory
      failed_when: "memory.stdout|float < 2"
```

The playbook has been created exactly to validate if the system is meeting the oracle recommended System Requirements.

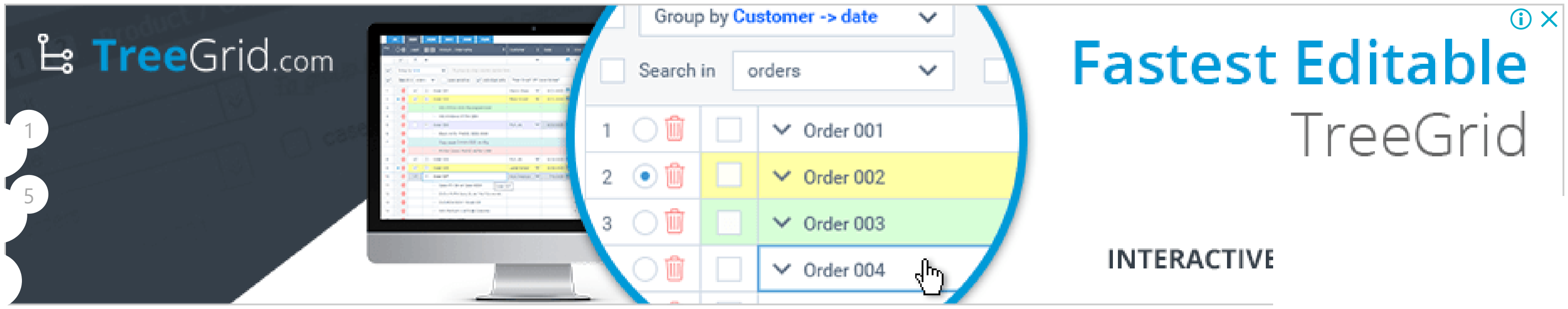
The tasks are programmed to fail when they are not meeting the requirements. This is done using failed_when and a simple math.

Execution Output of the playbook is given below



```
akarav@middlewareinventory:~$ ansible-playbook playbooks/wlsprereq.yaml -i ansible_hosts

PLAY [app] ************************************************************************************************************

TASK [Gathering Facts] ************************************************************************************************
ok: [mwiapp01]

TASK [Making sure the /tmp has more than 1gb] *************************************************************************
changed: [mwiapp01]

TASK [Making sure the /opt has more than 4gb] *************************************************************************
changed: [mwiapp01]

TASK [Making sure the Physical Memory more than 2gb] ******************************************************************
fatal: [mwiapp01]: FAILED! => {"changed": true, "cmd": "cat /proc/meminfo|grep -i memtotal|awk '{print $2/1024/1024}'", "delta": "0:00:00.007636", "end": "2018-09-22 17:20:17.718
118", "failed_when_result": true, "rc": 0, "start": "2018-09-22 17:20:17.710482", "stderr": "", "stderr_lines": [], "stdout": "0.969387", "stdout_lines": ["0.969387"]}
        to retry, use: --limit @/Users/aksarav/VirtualBox VMs/vagrantVM/playbooks/wlsprereq.retry

PLAY RECAP ************************************************************************************************************
mwiapp01                   : ok=3    changed=2    unreachable=0    failed=1
```

as shown in the preceding snapshot, The final requirement, The Physical memory is not enough (or) not meeting our requirement of 2gb. You can notice that the system is built with 1gb in the `stdout` of the error message



We come to a conclusion, as we have already discussed about all three conditional statements like when, failed_when and changed_when in detail with examples

Hope it is useful and make sense.