**Benefits of Ansible**

* Free: Ansible is an open-source tool.
* Very simple to set up and use: No special coding skills are necessary to use Ansible’s playbooks (more on playbooks later).
* Powerful: Ansible lets you model even highly complex IT workflows.
* Flexible: You can orchestrate the entire application environment no matter where it’s deployed. You can also customize it based on your needs.
* Agentless: You don’t need to install any other software or firewall ports on the client systems you want to automate. You also don’t have to set up a separate management structure.
* Efficient: Because you don’t need to install any extra software, there’s more room for application resources on your server.

**Ansible playbooks**

Ansible Playbooks offer a repeatable, reusable, simple configuration management and multi-machine deployment system, one that is well suited to deploying complex applications. If you need to execute a task with Ansible more than once, write a playbook and put it under source control. Then you can use the playbook to push out new configuration or confirm the configuration of remote systems.

Playbooks can:

declare configurations

orchestrate steps of any manual ordered process, on multiple sets of machines, in a defined order

launch tasks synchronously or [asynchronously](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_async.html#playbooks-async)

## [Playbook syntax](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html#id3)

Playbooks are expressed in YAML format with a minimum of syntax.

A playbook is composed of one or more ‘plays’ in an ordered list. The terms ‘playbook’ and ‘play’ are sports analogies. Each play executes part of the overall goal of the playbook, running one or more tasks. Each task calls an Ansible module.

**EXAMPLE 1**

---

- hosts: all

tasks:

- name: Print message

debug:

msg: Hello Ansible World

**EXAMPLE 2**

---

- hosts: all

vars:

- username: sammy

- home: /home/sammy

tasks:

- name: print variables

debug:

msg: "Username: {{ username }}, Home dir: {{ home }}"

The **vars** section of the playbook defines a list of variables that will be injected in the scope of that play. All tasks, as well as any file or template that might be included in the playbook, will have access to these variables.

## [Playbook execution](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html#id4)

A playbook runs in order from top to bottom. Within each play, tasks also run in order from top to bottom. Playbooks with multiple ‘plays’ can orchestrate multi-machine deployments, running one play on your webservers, then another play on your database servers, then a third play on your network infrastructure, and so on. At a minimum, each play defines two things:

* the managed nodes to target, using a pattern
* at least one task to execute

**EXAMPLE 3**

In this example, the first play targets the web servers; the second play targets the database servers.

**---**

**-** **name:** Update web servers

**hosts:** webservers

**remote\_user:** root

**tasks:**

**-** **name:** Ensure apache is at the latest version

**ansible.builtin.yum:**

**name:** httpd

**state:** latest

**-** **name:** Write the apache config file

**ansible.builtin.template:**

**src:** /srv/httpd.j2

**dest:** /etc/httpd.conf

**-** **name:** Update db servers

**hosts:** databases

**remote\_user:** root

**tasks:**

**-** **name:** Ensure postgresql is at the latest version

**ansible.builtin.yum:**

**name:** postgresql

**state:** latest

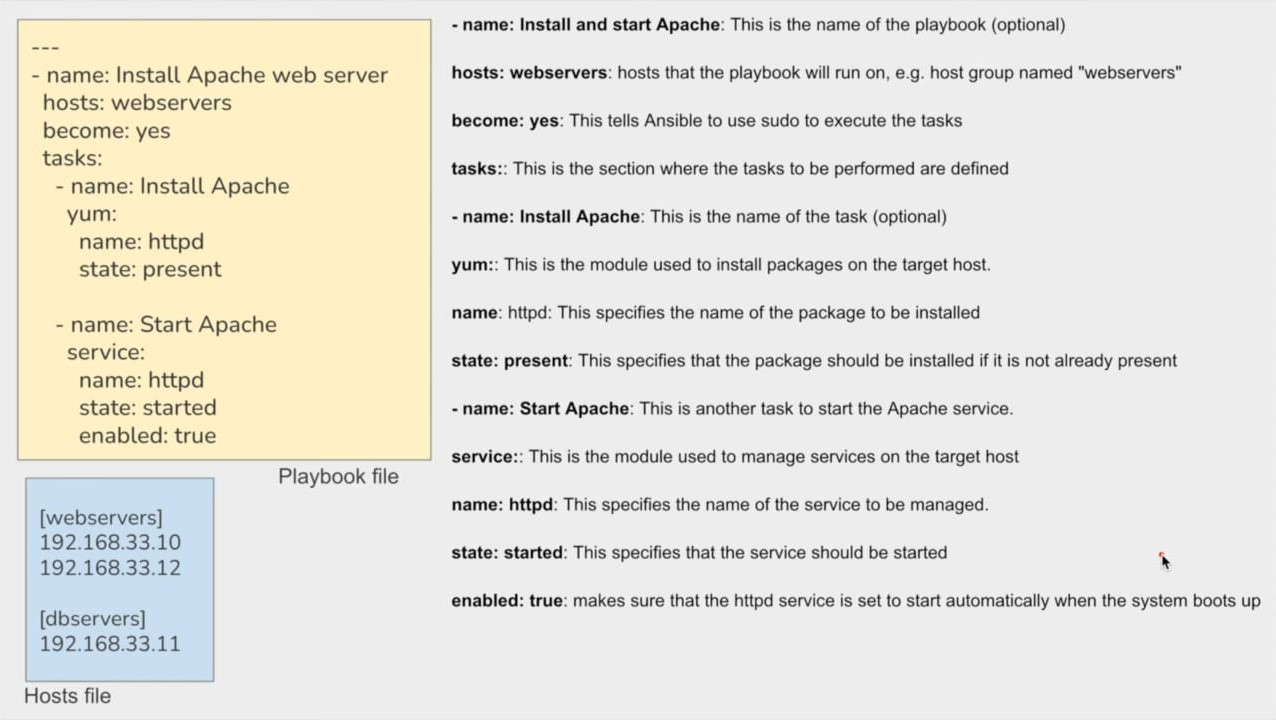
**-** **name:** Ensure that postgresql is started

**ansible.builtin.service:**

**name:** postgresql

**state:** started

Your playbook can include more than just a hosts line and tasks. For example, the playbook above sets a remote user for each play. This is the user account for the SSH connection.

**EXAMPLE 4**

## What is Ansible Inventory?

**Ansible** can work with multiple systems in your infrastructure at the same time. In order to work with multiple servers, **Ansible** needs to establish connectivity to those servers. This is done with SSH for Linux and PowerShell remoting for windows that would make **Ansible** agentless. Agentless means you don’t need to install any additional software on the Target machines to be able to work with **Ansible**. One of the major disadvantages of other automation tools is that you need to install and configure agents on the Target machines before you can invoke any kind of automation. Now the information about these Target systems is stored in a file called an inventory file. If you don’t create a new inventory file **Ansible** uses its default inventory file located at /etc/ansible/hosts.

**Ansible Modules**

Modules (also referred to as “task plugins” or “library plugins”) are discrete units of code that can be used from the command line or in a playbook task. Ansible executes each module, usually on the remote managed node, and collects return values. In Ansible 2.10 and later, most modules are hosted in collections.

**Ansible modules are standalone scripts that can be used inside an Ansible playbook. A playbook consists of a play, and a play consists of tasks. These concepts may seem confusing if you're new to Ansible, but as you begin writing and working more with playbooks, they will become familiar There are some modules that are frequently used in automating everyday tasks; those are the ones that we will cover in this article.**

**Ansible has three main files that you need to consider:**

* **Host/inventory file: Contains the entry of the nodes that need to be managed**
* **Ansible.cfg file: Located by default at /etc/ansible/ansible.cfg, it has the necessary privilege escalation options and the location of the inventory file**
* **Main file: A playbook that has modules that perform various tasks on a host listed in an inventory or host file**

## ****Module 1: Package management****

**There is a module for most popular package managers, such as DNF and APT, to enable you to install any package on a system. Functionality depends entirely on the package manager, but usually these modules can install, upgrade, downgrade, remove, and list packages. The names of relevant modules are easy to guess. For example, the DNF module is dnf\_module, the old YUM module (required for Python 2 compatibility) is yum\_module, while the APT module is apt\_module, and so on.**

**Example 1:**

- name: install the latest version of Apache and MariaDB

dnf:

name:

- httpd

- mariadb-server

state: latest

**This installs the Apache web server and the MariaDB SQL database.**

### Example 2:

- name: Install a list of packages

yum:

name:

- nginx

- postgresql

- postgresql-server

state: present

**This installs the list of packages and helps download multiple packages.**

**Module 2: Service**

**After installing a package, you need a module to start it. The service module enables you to start, stop, and reload installed packages; this comes in pretty handy.**

### Example 1:

- name: Start service foo, based on running process /usr/bin/foo

service:

name: foo

pattern: /usr/bin/foo

state: started

This starts the service **foo**.

### Example 2:

- name: Restart network service for interface eth0

service:

name: network

state: restarted

args: eth0

This restarts the network service of the interface **eth0**.

## Module 3: Copy

The copy module copies a file from the local or remote machine to a location on the remote machine.

### Example 1:

- name: Copy a new "ntp.conf file into place, backing up the original if it differs from the copied version

copy:

src: /mine/ntp.conf

dest: /etc/ntp.conf

owner: root

group: root

mode: '0644'

backup: yes

## Module 4: Debug

The debug module prints statements during execution and can be useful for debugging variables or expressions without having to halt the playbook.

### Example 1:

- name: Display all variables/facts known for a host

debug:

var: hostvars[inventory\_hostname]

verbosity: 4

This displays all the variable information for a host that is defined in the inventory file.