**Introduction**

Configuration management systems are designed to make controlling large numbers of servers easy for administrators and operations teams. They allow you to control many different systems in an automated way from one central location. While there are many popular configuration management systems available for Linux systems, such as Chef and Puppet, these are often more complex than many people want or need. **Ansible** is a great alternative to these options because it has a much smaller overhead to get started.

Ansible works by configuring client machines from an computer with Ansible components installed and configured. It communicates over normal SSH channels in order to retrieve information from remote machines, issue commands, and copy files. Because of this, an Ansible system does not require any additional software to be installed on the client computers. This is one way that Ansible simplifies the administration of servers. Any server that has an SSH port exposed can be brought under Ansible's configuration umbrella, regardless of what stage it is at in its life cycle.

Ansible takes on a modular approach, making it easy to extend to use the functionalities of the main system to deal with specific scenarios. Modules can be written in any language and communicate in standard JSON. Configuration files are mainly written in the YAML data serialization format due to its expressive nature and its similarity to popular markup languages. Ansible can interact with clients through either command line tools or through its configuration scripts called Playbooks.

In this guide, you'll install Ansible on a CentOS 7 server and learn some basics of how to use the software.

Prerequisites

To follow this tutorial, you will need:

* One CentOS 7 server. Follow the steps in [Initial Server Setup with CentOS 7](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-centos-7) to create a non-root user, and make sure you can connect to the server without a password.

Step 1 — Installing Ansible

To begin exploring Ansible as a means of managing our various servers, we need to install the Ansible software on at least one machine.

To get Ansible for CentOS 7, first ensure that the CentOS 7 EPEL repository is installed:

* sudo yum install epel-release

Once the repository is installed, install Ansible with yum:

* sudo yum install ansible

We now have all of the software required to administer our servers through Ansible.

Step 2 — Configuring Ansible Hosts

Ansible keeps track of all of the servers that it knows about through a "hosts" file. We need to set up this file first before we can begin to communicate with our other computers.

Open the file with root privileges like this:

* sudo vi /etc/ansible/hosts

You will see a file that has a lot of example configurations commented out. Keep these examples in the file to help you learn Ansible's configuration if you want to implement more complex scenarios in the future.

The hosts file is fairly flexible and can be configured in a few different ways. The syntax we are going to use though looks something like this:

Example hosts file

[group\_name]

alias ansible\_ssh\_host=your\_server\_ip

The group\_name is an organizational tag that lets you refer to any servers listed under it with one word. The alias is just a name to refer to that server.

Imagine you have three servers you want to control with Ansible. Ansible communicates with client computers through SSH, so each server you want to manage should be accessible from the Ansible server by typing:

* ssh root@your\_server\_ip

You should not be prompted for a password. While Ansible certainly has the ability to handle password-based SSH authentication, SSH keys help keep things simple. You can follow the tutorial [How To Use SSH Keys with DigitalOcean Droplets](https://www.digitalocean.com/community/tutorials/how-to-use-ssh-keys-with-digitalocean-droplets) to set up SSH keys on each host if you haven't already.

We will assume that our servers' IP addresses are 192.0.2.1, 192.0.2.2, and 192.0.2.3. Let's set this up so that we can refer to these individually as host1, host2, and host3, or as a group as servers. To configure this, you would add this block to your hosts file:

/etc/ansible/hosts

[servers]

host1 ansible\_ssh\_host=192.0.2.1

host2 ansible\_ssh\_host=192.0.2.2

host3 ansible\_ssh\_host=192.0.2.3

Hosts can be in multiple groups and groups can configure parameters for all of their members. Let's try this out now.

Ansible will, by default, try to connect to remote hosts using your current username. If that user doesn't exist on the remote system, a connection attempt will result in this error:

Ansible connection error

host1 | UNREACHABLE! => {

"changed": false,

"msg": "Failed to connect to the host via ssh.",

"unreachable": true

}

Let's specifically tell Ansible that it should connect to servers in the "servers" group with the **sammy** user. Create a directory in the Ansible configuration structure called group\_vars.

* sudo mkdir /etc/ansible/group\_vars

Within this folder, we can create YAML-formatted files for each group we want to configure:

* sudo nano /etc/ansible/group\_vars/servers

Add this code to the file:

/etc/ansible/group\_vars/servers

---

ansible\_ssh\_user: sammy

YAML files start with "---", so make sure you don't forget that part.

Save and close this file when you are finished. Now Ansible will always use the **sammy** user for the servers group, regardless of the current user.

If you want to specify configuration details for every server, regardless of group association, you can put those details in a file at /etc/ansible/group\_vars/all. Individual hosts can be configured by creating files under a directory at /etc/ansible/host\_vars.

Step 3 — Using Simple Ansible Commands

Now that we have our hosts set up and enough configuration details to allow us to successfully connect to our hosts, we can try out our very first command.

Ping all of the servers you configured by typing:

* ansible -m ping all

Ansible will return output like this:

Output

host1 | SUCCESS => {

"changed": false,

"ping": "pong"

}

host3 | SUCCESS => {

"changed": false,

"ping": "pong"

}

host2 | SUCCESS => {

"changed": false,

"ping": "pong"

}

This is a basic test to make sure that Ansible has a connection to all of its hosts.

The -m ping portion of the command is an instruction to Ansible to use the "ping" module. These are basically commands that you can run on your remote hosts. The ping module operates in many ways like the normal ping utility in Linux, but instead it checks for Ansible connectivity.

The all portion means "all hosts." You could just as easily specify a group:

* ansible -m ping servers

You can also specify an individual host:

* ansible -m ping host1

You can specify multiple hosts by separating them with colons:

* ansible -m ping host1:host2

The shell module lets us send a terminal command to the remote host and retrieve the results. For instance, to find out the memory usage on our host1 machine, we could use:

* ansible -m shell -a 'free -m' host1

As you can see, you pass arguments into a script by using the -a switch. Here's what the output might look like:

Output

host1 | SUCCESS | rc=0 >>

total used free shared buffers cached

Mem: 3954 227 3726 0 14 93

-/+ buffers/cache: 119 3834

Swap: 0 0 0