How to Install and Configure Ansible on Ubuntu 18.04

Posted July 13, 2018 37.6k CONFIGURATION MANAGEMENT ANSIBLE UBUNTU UBUNTU 18.04

By: Stephen Rees-Carter By: Mark Drake

Not using **Ubuntu 18.04**? Choose a different version:

CentOS 7 ›

Ubuntu 16.04 ›

Ubuntu 14.04 ›

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 requires a much smaller overhead to get started.

In this guide, we will discuss how to install Ansible on an Ubuntu 18.04 server and go over some basics of how to use the software.

How Does Ansible Work?

Ansible works by configuring client machines from a computer that has the Ansible components installed and configured.

It communicates over normal SSH channels 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. This means that any computer that you can administer through SSH, you can also administer through Ansible.

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 hosts either through command line tools or its configuration scripts, which are known as Playbooks.

Prerequisites

To follow this tutorial, you will need:

- Two or more Ubuntu 18.04 servers. One of these will be used as your *Ansible server*, while the remainder will be used as your *Ansible hosts*. Each should have a non-**root** user with `sudo` privileges and a basic firewall configured. You can set this up by following our Initial Server Setup Guide for Ubuntu 18.04. Please note that the examples throughout this guide specify three Ansible hosts, but the commands and configurations shown can be adjusted for any number of clients.
- SSH keys generated for the non-**root** user on your Ansible server. To do this, follow Step 1 of our guide on How to Set Up SSH Keys on Ubuntu 18.04. For the purposes of this tutorial, you can save the key pair to the default location (`~/.ssh/id_rsa`) and you do not need to password-protect it.

Step 1 — Installing Ansible

To begin using Ansible as a means of managing your various servers, you need to install the Ansible software on at least one machine.

To get the latest version of Ansible for Ubuntu, you can add the project's PPA (personal package archive) to your system. Before doing this, though, you should first update your package index and install the `software-properties-common` package. This software will make it easier to manage this and other independent software repositories:

```
$ sudo apt update
$ sudo apt install software-properties-common
```

Then add the Ansible PPA by typing the following command:

```
$ sudo apt-add-repository ppa:ansible/ansible
```

Press `ENTER` to accept the PPA addition.

Next, refresh your system's package index once again so that it is aware of the packages available in the PPA:

```
$ sudo apt update
```

Following this update, you can install the Ansible software:

```
$ sudo apt install ansible
```

Your Ansible server now has all of the software required to administer your hosts.

Step 2 — Configuring SSH Access to the Ansible Hosts

As mentioned previously, Ansible primarily communicates with client computers through SSH. While it certainly has the ability to handle password-based SSH authentication, using SSH keys can help to keep things simple.

On your Ansible server, use the `cat` command to print the contents of your non-root user's SSH public key file to the terminal's output:

```
$ cat ~/.ssh/id_rsa.pub
```

Copy the resulting output to your clipboard, then open a new terminal and connect to one of your Ansible hosts using SSH:

```
$ ssh sammy@ansible_host_ip
```

Switch to the client machine's **root** user:

```
$ su -
```

As the **root** user, open the `authorized_keys` within the `~/.ssh` directory:

```
# nano ~/.ssh/authorized_keys
```

In the file, paste your Ansible server user's SSH key, then save the file and close the editor (press `CTRL + X`, `Y`, then `ENTER`). Then run the `exit` command to return to the host's non-**root** user:

```
# exit
```

Lastly, because Ansible uses a python interpreter located at `/usr/bin/python` to run its modules, you'll need to install Python 2 on the host in order for Ansible to communicate with it. Run the following commands to update the host's package index and install the `python` package:

```
$ sudo apt update
$ sudo apt install python
```

Following this, you can run the `exit` command once again to close the connection to the client:

```
$ exit
```

Repeat this process for each server you intend to control with your Ansible server. Next, we'll configure the Ansible server to connect to these hosts using Ansible's `hosts` file.

Step 3 — Setting Up 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 `sudo` privileges, like this:

```
$ sudo nano /etc/ansible/hosts
```

Inside the file, you will see a number of example configurations that have been commented out (with a `#` preceding each line). These examples won't actually work for us since the hosts listed in each one are made up. We will, however, keep these examples in the file to help us with configuration if we 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 like this:

```
[group_name]
alias ansible_ssh_host=your_server_ip
```

In this example, `group_name` is an organizational tag that lets you refer to any servers listed under it with one word, while `alias` is just a name to refer to one specific server.

So, in our scenario, we are imagining that we have three servers we are going to control with Ansible. At this point, these servers are accessible from the Ansible server by typing:

```
$ ssh root@ansible_host_ip
```

You should not be prompted for a password if you have set this up correctly. For the purpose of demonstration, we will assume that our hosts' IP addresses are `203.0.113.1`, `203.0.113.2`, and `203.0.113.3`. We will set this up so that we can refer to these individually as `host1`, `host2`, and `host3`, or as a group with the name `servers`.

This is the block that we should add to our `hosts` file to accomplish this:

/etc/ansible/hosts

```
[servers]
host1 ansible_ssh_host=203.0.113.1
host2 ansible_ssh_host=203.0.113.2
host3 ansible_ssh_host=203.0.113.3
```

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

With our current settings, if we tried to connect to any of these hosts with Ansible, the command would fail (assuming you are not operating as the root user). This is because your SSH key is embedded for the **root** user on the remote systems and Ansible will by default try to connect as your current user. A connection attempt will get this error:

Output

```
host1 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh.",
```

```
    "unreachable": true
}
```

On the Ansible server, we're using a user called **sammy**. Ansible will try to connect to each host with `ssh sammy@server`. This will not work if the **sammy** user is not on the remote system as well.

We can create a file that tells all of the servers in the "servers" group to connect as the **root** user.

To do this, we will create a directory in the Ansible configuration structure called `group_vars`. Within this folder, we can create YAML-formatted files for each group we want to configure:

```
$ sudo mkdir /etc/ansible/group_vars
$ sudo nano /etc/ansible/group_vars/servers
```

We can put our configuration in here. YAML files start with "---", so make sure you don't forget that part.

/etc/ansible/group_vars/servers

```
---
ansible_ssh_user: root
```

Save and close this file when you are finished.

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 named after their alias under a directory at `/etc/ansible/host_vars`.

Step 4 — 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
```

Ping 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 `all` means all hosts. We could just as easily specify a group:

```
$ ansible -m ping servers
```

We could also specify an individual host:

```
$ ansible -m ping host1
```

We can specify multiple hosts by separating them with colons:

```
$ ansible -m ping host1:host2
```

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 ping module doesn't really take any arguments, but we can try another command to see how that works. We pass arguments into a script by typing `-a`.

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
```

Shell 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
```

With that, your Ansible server configured and you can successfully communicate and control your hosts.

Conclusion

In this tutorial, we have configured Ansible and verified that it can communicate with each host. We have also used the `ansible` command to execute simple tasks remotely.

Although this is useful, we have not covered the most powerful feature of Ansible in this article: Playbooks. Ansible Playbooks are a powerful, simple way to manage server configurations and multi-machine deployments. For an introduction to Playbooks, see this guide. Additionally, we encourage you to check out the official Ansible documentation to learn more about the tool.

By: Stephen Rees-Carter By: Mark Drake

Related Tutorials

How to Manage Multistage Environments with Ansible

How To Install Elasticsearch, Logstash, and Kibana (Elastic Stack) on Ubuntu 18.04

How To Test Ansible Roles with Molecule on Ubuntu 18.04

How To Create a Kubernetes 1.11 Cluster Using Kubeadm on Ubuntu 18.04

Navigator's Guide: Modular Infrastructure Configuration

3 Comments

Leave a comment...

Log In to Comment

ebostran10312010 *October 12, 2018*

0 hello dear people
how can i get here your*server*ip
(alias ansible*ssh*host = your*server*ip)
Find
Thnx

chanoch *October 22, 2018*

0 My experience was that I had to softlink python3 to python for this to work on Ubuntu 18.04 or ansible would complain that it couldn't find /usr/bin/python on each droplet:

```
root@my-droplet$sudo ln -s /usr/bin/python3 /usr/bin/python
```

I also found that I needed to enter the pass phrase for the ssh cert once per server. After receiving the reply from the first server, I entered the pass phrase even though there was no prompt and this seemed to make it work.

It is probably better to configure the certificate at playbook level rather than globally in any case.

harireddy090 *November 9, 2018*

0 Getting error while doing Ansible-Playbook

host1 | FAILED! => {
"changed": false,
"module*stderr": "Shared connection to 172.31.8.238 closed.*\r\n",
*"module*stdout": "/bin/sh: 1: /usr/bin/python: not found\r\n",
"msg": "MODULE FAILURE\nSee stdout/stderr for the exact error",
"rc": 127
}

already i have python version Python 2.7.15rc1