TUTORIAL

How to Install and Configure Ansible on Ubuntu 18.04

Ubuntu Configuration Management Ansible Ubuntu 18.04

By Stephen Rees-Carter, Mark Drake, and Erika Heidi

Last Validated on April 16, 2020 • Originally Published on July 13, 2018 280k

English

Not using Ubuntu 18.04?
Choose a different version or distribution.

Ubuntu 18.04

Introduction

SCROLL TO TOP

Configuration management systems are designed to streamline the process of controlling large numbers of servers, 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 tools 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 offers a simple architecture that doesn't require special software to be installed on nodes, using SSH to execute the automation tasks and YAML files to define provisioning details.

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 this software.

How Does Ansible Work?

Ansible works by configuring client machines, referred to as *managed nodes*, from a computer that has the Ansible components installed and configured, which is then called the *Ansible control node*.

It communicates over normal SSH channels to retrieve information from remote systems, 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, enabling you to extend 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:

- **One Ansible Control Node**: The Ansible control node is the machine we will use to connect to and control the Ansible hosts over SSH. Your Ansible control node can either be your local machine or a server dedicated to running Ansible, though this guide assumes your control node is an Ubuntu 18.04 system. Make sure the control node has:
  - A non-root user with sudo privileges. To set this up, you can follow **Steps 2 and 3** of our Initial Server Setup Guide for Ubuntu 18.04. However, please note that if you're using a remote server as your Ansible Control node, you should follow **every step** of this guide. Doing so will configure a firewall on the server with `ufw` and enable external access to your non-root user profile, both of which will help keep the remote server secure.
  - An SSH keypair associated with this user. To set this up, you can follow **Step 1** of our guide on How to Set Up SSH Keys on Ubuntu 18.04.
- **One or more Ansible Hosts**: An Ansible host is any machine that your Ansible control node is configured to automate. This guide assumes your Ansible hosts are remote Ubuntu 18.04 servers. Make sure each Ansible host has:
  - The Ansible control node's SSH public key added to the `authorized_keys` of a system user. This user can be either **root** or a regular user with sudo privileges. To set this up, you can follow **Step 2** of How to Set Up SSH Keys on Ubuntu 18.04.

Step 1 — Installing Ansible

To begin using Ansible as a means of managing your server infrastructure, you need to install the Ansible software on the machine that will serve as the Ansible control node.

From your control node, run the following command to include the official project's PPA (personal package archive) in your system's list of sources:

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

Press `ENTER` when prompted to accept the PPA addition.

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

```
$ sudo apt update
```

Following this update, you can install the Ansible software with:

```
$ sudo apt install ansible
```

Your Ansible control node now has all of the software required to administer your hosts. Next, we will go over how to add your hosts to the control node's inventory file so that it can control them.

Step 2 — Setting Up the Inventory File

The *inventory file* contains information about the hosts you'll manage with Ansible. You can include anywhere from one to several hundred servers in your inventory file, and hosts can be organized into groups and subgroups. The inventory file is also often used to set variables that will be valid only for specific hosts or groups, in order to be used within playbooks and templates. Some variables can also affect the way a playbook is run, like the `ansible_python_interpreter` variable that we'll see in a moment.

To edit the contents of your default Ansible inventory, open the `/etc/ansible/hosts` file using your text editor of choice, on your Ansible Control Node:

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

> **Note**: some Ansible installations won't create a default inventory file. If the file doesn't exist in your system, you can create a new file at `/etc/ansible/hosts` or provide a custom inventory path using the `-i` parameter when running commands and playbooks.

The default inventory file provided by the Ansible installation contains a number of examples that you can use as references for setting up your inventory. The following example defines a group named `[servers]` with three different servers in it, each identified by a custom

alias: **server1**, **server2**, and **server3**. Be sure to replace the highlighted IPs with the IP addresses of your Ansible hosts.

/etc/ansible/hosts

```
[servers]
server1 ansible_host=203.0.113.111
server2 ansible_host=203.0.113.112
server3 ansible_host=203.0.113.113

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

The `all:vars` subgroup sets the `ansible_python_interpreter` host parameter that will be valid for all hosts included in this inventory. This parameter makes sure the remote server uses the `/usr/bin/python3` Python 3 executable instead of `/usr/bin/python` (Python 2.7), which is not present on recent Ubuntu versions.

When you're finished, save and close the file by pressing `CTRL+X` then `Y` and `ENTER` to confirm your changes.

Whenever you want to check your inventory, you can run:

```
$ ansible-inventory --list -y
```

You'll see output similar to this, but containing your own server infrastructure as defined in your inventory file:

Output

```
all:
  children:
    servers:
      hosts:
        server1:
          ansible_host: 203.0.113.111
          ansible_python_interpreter: /usr/bin/python3
        server2:
          ansible_host: 203.0.113.112
          ansible_python_interpreter: /usr/bin/python3
```

```
        server3:
          ansible_host: 203.0.113.113
          ansible_python_interpreter: /usr/bin/python3
    ungrouped: {}
```

Now that you've configured your inventory file, you have everything you need to test the connection to your Ansible hosts.

Step 3 — Testing Connection

After setting up the inventory file to include your servers, it's time to check if Ansible is able to connect to these servers and run commands via SSH.

For this guide, we will be using the Ubuntu **root** account because that's typically the only account available by default on newly created servers. If your Ansible hosts already have a regular sudo user created, you are encouraged to use that account instead.

You can use the `-u` argument to specify the remote system user. When not provided, Ansible will try to connect as your current system user on the control node.

From your local machine or Ansible control node, run:

```
$ ansible all -m ping -u root
```

This command will use Ansible's built-in `ping` module to run a connectivity test on all nodes from your default inventory, connecting as **root**. The `ping` module will test:

- if hosts are accessible;
- if you have valid SSH credentials;
- if hosts are able to run Ansible modules using Python.

You should get output similar to this:

```
Output
server1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
```

```
}
server2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
server3 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

If this is the first time you're connecting to these servers via SSH, you'll be asked to confirm the authenticity of the hosts you're connecting to via Ansible. When prompted, type `yes` and then hit `ENTER` to confirm.

Once you get a `"pong"` reply back from a host, it means you're ready to run Ansible commands and playbooks on that server.

> **Note**: If you are unable to get a successful response back from your servers, check our Ansible Cheat Sheet Guide for more information on how to run Ansible commands with different connection options.

Step 4 — Running Ad-Hoc Commands (Optional)

After confirming that your Ansible control node is able to communicate with your hosts, you can start running ad-hoc commands and playbooks on your servers.

Any command that you would normally execute on a remote server over SSH can be run with Ansible on the servers specified in your inventory file. As an example, you can check disk usage on all servers with:

```
$ ansible all -a "df -h" -u root
```

Output

```
server1 | CHANGED | rc=0 >>
Filesystem      Size  Used Avail Use% Mounted on
udev            3.9G     0  3.9G   0% /dev
tmpfs           798M  624K  798M   1% /run
```

```
/dev/vda1       155G  2.3G  153G   2% /
tmpfs           3.9G     0  3.9G   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/vda15      105M  3.6M  101M   4% /boot/efi
tmpfs           798M     0  798M   0% /run/user/0

server2 | CHANGED | rc=0 >>
Filesystem      Size  Used Avail Use% Mounted on
udev            2.0G     0  2.0G   0% /dev
tmpfs           395M  608K  394M   1% /run
/dev/vda1        78G  2.2G   76G   3% /
tmpfs           2.0G     0  2.0G   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/vda15      105M  3.6M  101M   4% /boot/efi
tmpfs           395M     0  395M   0% /run/user/0

...
```

The highlighted command `df -h` can be replaced by any command you'd like.

You can also execute Ansible modules via ad-hoc commands, similarly to what we've done before with the `ping` module for testing connection. For example, here's how we can use the `apt` module to install the latest version of `vim` on all the servers in your inventory:

```
$ ansible all -m apt -a "name=vim state=latest" -u root
```

You can also target individual hosts, as well as groups and subgroups, when running Ansible commands. For instance, this is how you would check the `uptime` of every host in the `servers` group:

```
$ ansible servers -a "uptime" -u root
```

We can specify multiple hosts by separating them with colons:

```
$ ansible server1:server2 -m ping -u root
```

For more information on how to use Ansible, including how to execute playbooks to automate server setup, you can check our Ansible Reference Guide.

Conclusion

In this guide, you've installed Ansible and set up an inventory file to execute ad-hoc commands from an Ansible Control Node.

Once you've confirmed you're able to connect and control your infrastructure from a central Ansible controller machine, you can execute any command or playbook you desire on those hosts. For fresh servers, the Initial Server Setup community playbook is a good starting point. You can also learn how to write your own playbooks with our guide Configuration Management 101: Writing Ansible Playbooks.

For more information on how to use Ansible, check out our Ansible Cheat Sheet Guide.

About the authors

Stephen Rees-Carter

has authored 9 tutorials.

Mark Drake

Technical Writer @ DigitalOcean

Erika Heidi

Dev/Ops passionate about open source, PHP, and Linux.

Still looking for an answer?

Ask a question

Search for more help

RELATED

How To Create a Kubernetes Cluster Using Kubeadm on Ubuntu 20.04

Tutorial

How To Manage Logfiles with Logrotate on Ubuntu 20.04

Tutorial

Comments

9 Comments

SCROLL TO TOP

Leave a comment...

Sign 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

Reply Report

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.

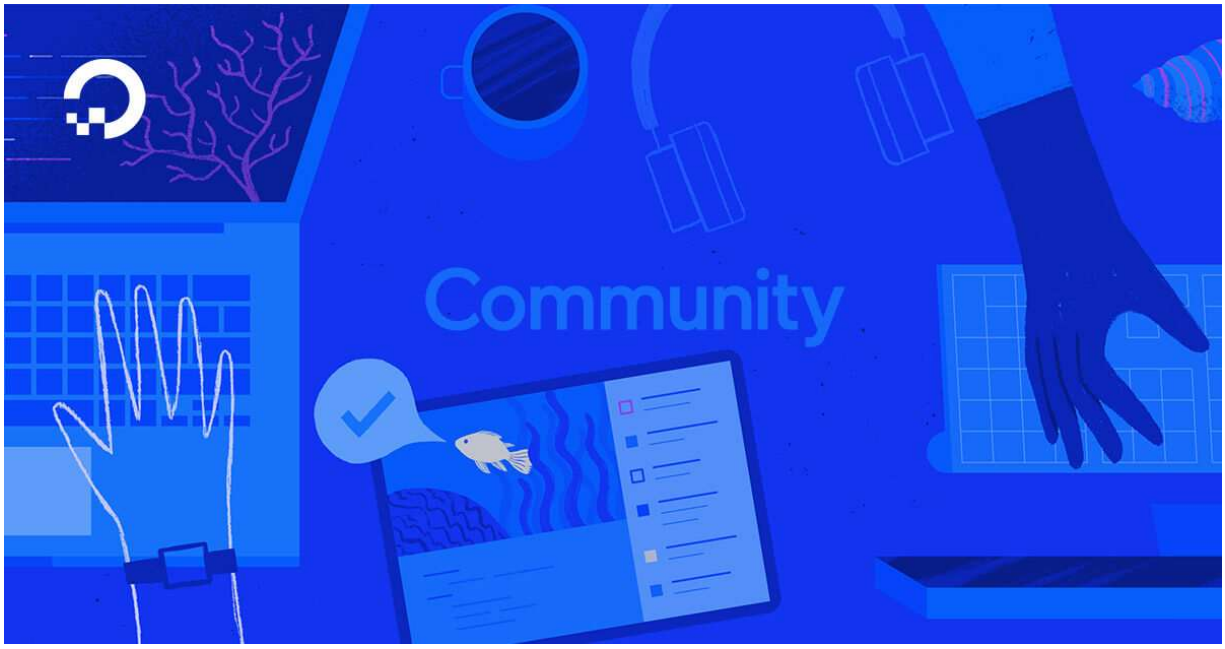
Reply Report

erikaheidi November 25, 2019

1 Hi @chanoch ! We have updated the tutorial to use Python 3. About the passphrase: if your SSH keypair uses a passphrase, you should include an extra parameter to the Ansible commands: `--ask-become-pass`. So a ping command would look like this

SCROLL TO TOP

`ansible all -m ping --ask-become-pass`. For more information on connection options, please check our Ansible Cheat Sheet Guide.



How to Use Ansible: A Reference Guide

by Erika Heidi

Ansible is a modern configuration management tool that facilitates the task of setting up and maintaining remote servers. This cheat sheet-style guide provides a quick reference to commands and practices commonly used when working with

Reply Report

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

Reply Report

Altif March 26, 2019

0

ansible*ssh*port deosn't work anymore with ansibile 2.0 and later version.

SCROLL TO TOP

Below link for reference.

https://docs.ansible.com/ansible/2.3/intro_inventory.html

```
Ansible 2.0 has deprecated the “ssh” from ansible_ssh_user, ansible_ssh_host, and ansi
```

Reply Report

mdrake March 26, 2019

1

Hello @Altif, thank you so much for your comment. I've updated the tutorial to make mention of these changes.

Reply Report

Comspots July 24, 2019

0

Hello Team,

Under one of my Ansible host, I am unable to locate authorized_keys within the ~/.ssh directory.

KIndly help

Reply Report

mdrake August 8, 2019

0

Hello @Comspots, and thank you for your comment.

If there isn't an `authorized_keys` file in your `/.ssh` directory, you can just create one by running the following:

```
$ nano ~/.ssh/authorized_keys
```

Then, you can add your Ansible server's SSH key to the new, empty file.

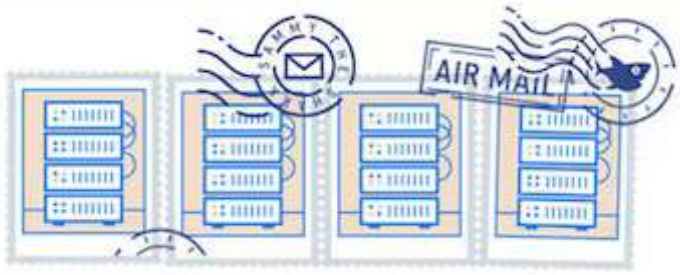
Reply Report

GrantWinney December 19, 2019

2

Thanks for this! I setup 2 Ubuntu droplets on DO and was able to get everything working by following your tutorial... worked great. I'll be reading your others, on creating playbooks and such, next.

Reply Report



GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.

HOLLIE'S HUB FOR GOOD

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

SCROLL TO TOP

Featured on Community Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

DigitalOcean Products Virtual Machines Managed Databases Managed Kubernetes Block Storage Object Storage Marketplace VPC Load Balancers

Company

About
Leadership
Blog
Careers
Partners
Referral Program
Press
Legal
Security & Trust Center

Products

Pricing
Products Overview
Droplets
Kubernetes
Managed Databases
Spaces
Marketplace
Load Balancers
Block Storage
API Documentation
Documentation
Release Notes

Community

Tutorials
Q&A
Tools and Integrations
Tags
Write for DigitalOcean
Presentation Grants
Hatch Startup Program
Shop Swag
Research Program
Open Source
Code of Conduct

Contact

Get Support
Trouble Signing In?
Sales
Report Abuse
System Status
Share your ideas

SCROLL TO TOP