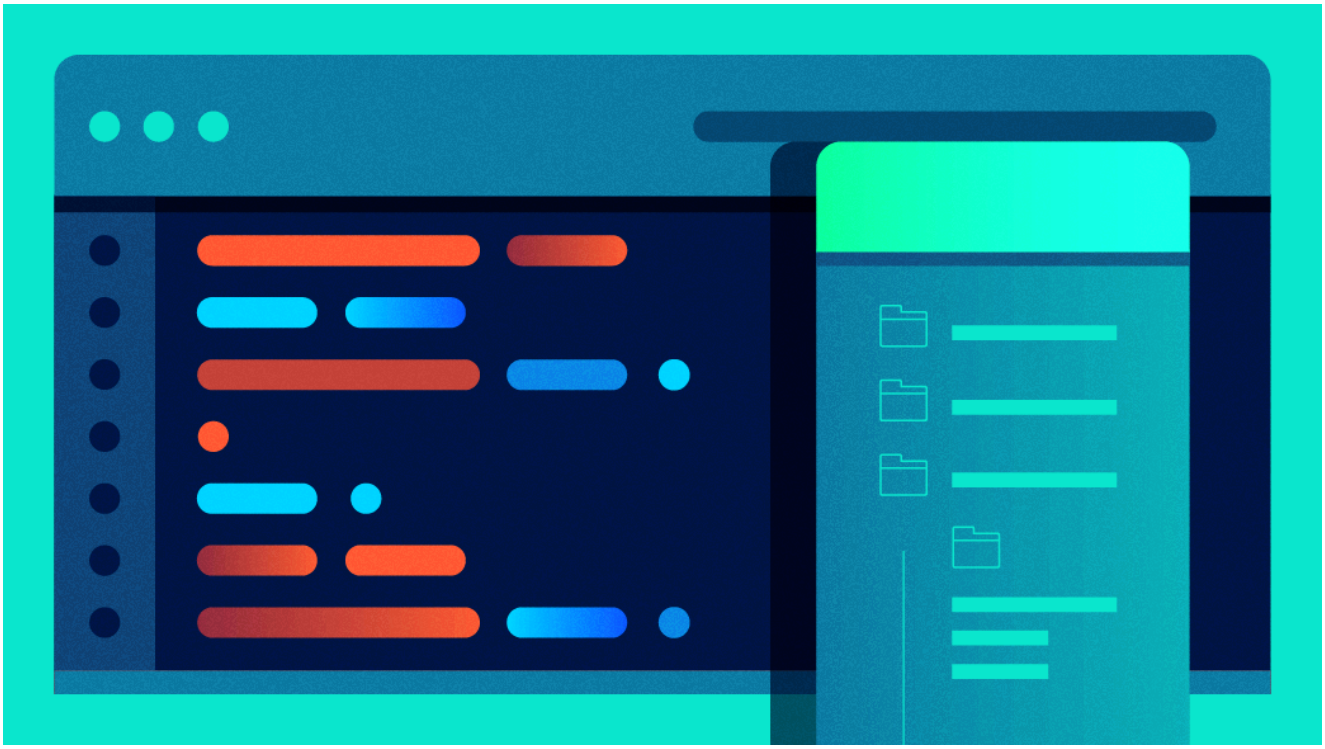
10 Ansible modules you need to know

See examples and learn the most important modules for automating everyday tasks with Ansible.

By Shashank Hegde

September 11, 2019 | 6 Comments | 7 min read

489 readers like this.



Ansible is an open source IT configuration management and automation platform. It uses human-readable YAML templates so users can program repetitive tasks to happen automatically without having to learn an advanced programming language.

Ansible is agentless, which means the nodes it manages do not require any software to be installed on them. This eliminates potential security vulnerabilities and makes overall management smoother.

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.

More on Ansible

A quickstart guide to Ansible

Ansible cheat sheet

Free online course: Ansible essentials

Download and install Ansible

eBook: The automated enterprise

eBook: Ansible for DevOps

Free Ansible eBooks

Latest Ansible articles

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, the Slackpkg module is slackpkg_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.

[Advance your automation expertise. Get the Ansible checklist: 5 reasons to migrate to Red Hat Ansible Automation Platform 2]

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/bi
  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 th
  copy:
    src: /mine/ntp.conf
    dest: /etc/ntp.conf
    owner: root
    group: root
    mode: '0644'
    backup: yes
```

Example 2:

```
- name: Copy file with owner and permission, using symbolic
  copy:
    src: /srv/myfiles/foo.conf
    dest: /etc/foo.conf
    owner: foo
    group: foo
    mode: u=rw,g=r,o=r
```

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.

Example 2:

```
- name: Write some content in a file /tmp/foo.txt
  copy:
    dest: /tmp/foo.txt
    content: |
      Good Morning!
      Awesome sunshine today.
    register: display_file_content
- name: Debug display_file_content
    debug:
      var: display_file_content
      verbosity: 2
```

This registers the content of the copy module output and displays it only when you specify verbosity as 2. For example:

```
ansible-playbook demo.yaml -vv
```

Module 5: File

The file module manages the file and its properties.

It sets attributes of files, symlinks, or directories.

It also removes files, symlinks, or directories.

Example 1:

```
- name: Change file ownership, group and permissions
  file:
    path: /etc/foo.conf
    owner: foo
    group: foo
    mode: '0644'
```

This creates a file named **foo.conf** and sets the permission to **0644**.

Example 2:

```
- name: Create a directory if it does not exist
  file:
    path: /etc/some_directory
    state: directory
    mode: '0755'
```

This creates a directory named **some_directory** and sets the permission to **0755**.

Module 6: Lineinfile

The lineinfile module manages lines in a text file.

It ensures a particular line is in a file or replaces an existing line using a back-referenced regular expression.

It's primarily useful when you want to change just a single line in a file.

Example 1:

```
- name: Ensure SELinux is set to enforcing mode
  lineinfile:
    path: /etc/selinux/config
    regexp: '^SELINUX='
    line: SELINUX=enforcing
```

This sets the value of **SELINUX=enforcing**.

Example 2:

```
- name: Add a line to a file if the file does not exist, wi
  lineinfile:
    path: /etc/resolv.conf
    line: 192.168.1.99 foo.lab.net foo
    create: yes
```

This adds an entry for the IP and hostname in the **resolv.conf** file.

Module 7: Git

The git module manages git checkouts of repositories to deploy files or software.

Example 1:

```
# Example Create git archive from repo
- git:
    repo: https://github.com/ansible/ansible-examples.git
    dest: /src/ansible-examples
    archive: /tmp/ansible-examples.zip
```

Example 2:

```
- git:
    repo: https://github.com/ansible/ansible-examples.git
    dest: /src/ansible-examples
    separate_git_dir: /src/ansible-examples.git
```

This clones a repo with a separate Git directory.

Module 8: Cli_command

The cli_command module, first available in Ansible 2.7, provides a platform-agnostic way of pushing text-based configurations to network devices over the **network_cli connection** plugin.

Example 1:

```
- name: commit with comment
  cli_config:
    config: set system host-name foo
    commit_comment: this is a test
```

This sets the hostname for a switch and exits with a commit message.

Example 2:

```
- name: configurable backup path
  cli_config:
    config: "{{ lookup('template', 'basic/config.j2') }}"
    backup: yes
    backup_options:
      filename: backup.cfg
      dir_path: /home/user
```

This backs up a config to a different destination file.

Module 9: Archive

The archive module creates a compressed archive of one or more files. By default, it assumes the compression source exists on the target.

Example 1:

```
- name: Compress directory /path/to/foo/ into /path/to/foo.
  archive:
```

```
    path: /path/to/foo
    dest: /path/to/foo.tgz
```

Example 2:

```
- name: Create a bz2 archive of multiple files, rooted at /
  archive:
    path:
    - /path/to/foo
    - /path/wong/foo
    dest: /path/file.tar.bz2
    format: bz2
```

Module 10: Command

One of the most basic but useful modules, the command module takes the command name followed by a list of space-delimited arguments.

Example 1:

```
- name: return motd to registered var
  command: cat /etc/motd
  register: mymotd
```

Example 2:

```
- name: Change the working directory to somedir/ and run th
  command: /usr/bin/make_database.sh db_user db_name
  become: yes
  become_user: db_owner
  args:
    chdir: somedir/
    creates: /path/to/database
```

Conclusion

There are tons of modules available in Ansible, but these ten are the most basic and powerful ones you can use for an automation job. As your requirements change, you can learn about other useful modules by entering **ansible-doc <module-name>** on the command line or refer to the official documentation.

What to read next

5 ops tasks to do with Ansible

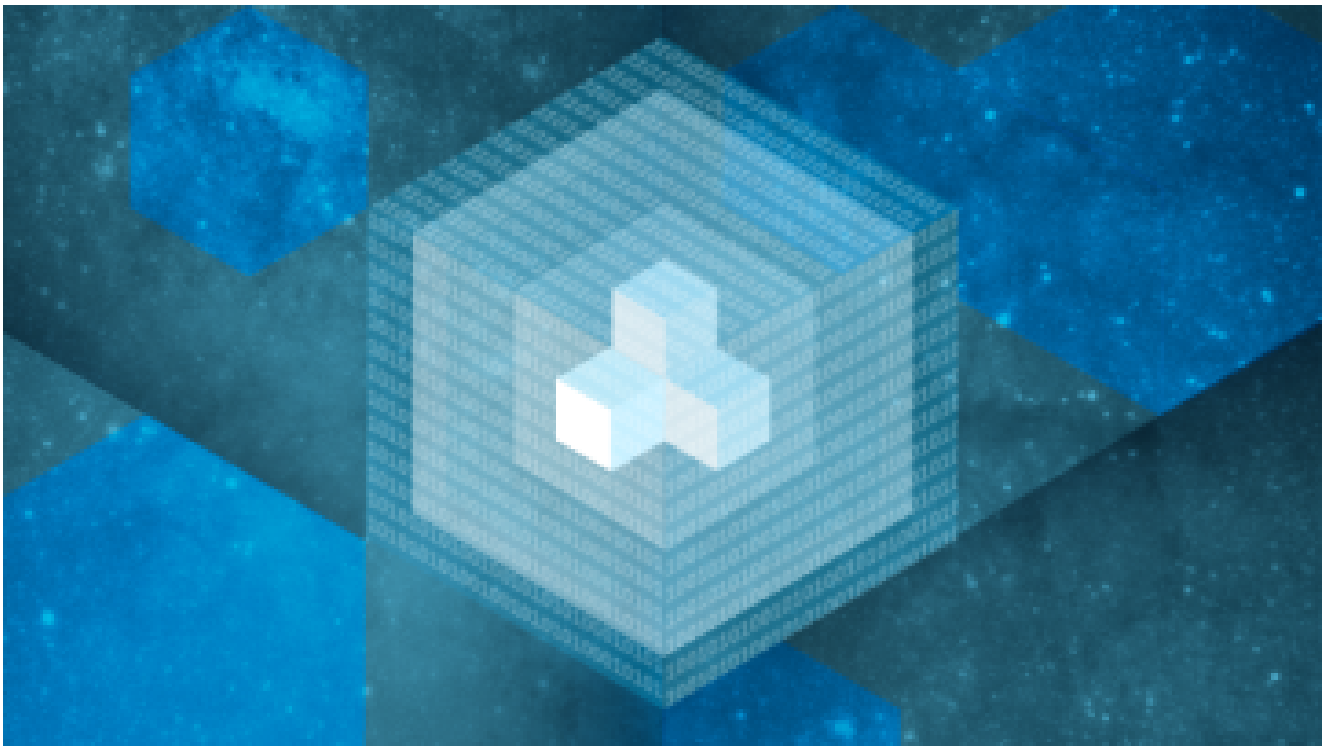
Less DevOps, more OpsDev.

Mark Phillips

A project manager's guide to Ansible

Ansible is best known for IT automation, but it can streamline operations across the entire organization.

Rich Butkevic



What you need to know about Ansible modules

Learn how and when to develop custom modules for Ansible.

 Jairo da Silva Junior

Tags: ANSIBLE

Shashank Hegde

I work as a solutions engineer at Red Hat and my day-to-day work involves working with OpenShift and Ansible. I'm highly passionate about open source, cloud, security and networking technologies.

More about me

6 Comments

These comments are closed.

Marco Bravo | September 11, 2019

No readers like this yet.

Thank you for the recap of those Ansible modules. It's a good list.

Adam Miller | September 11, 2019

No readers like this yet.

Hi! Ansible upstream contributor here, I am one of the maintainers for the yum and dnf modules.

First off, great write up!

Second, as of Ansible 2.8 you can now use the yum module for either yum or dnf (there's an action plugin associated with it to handle the magic). Just wanted to mention that, happy automating!

Ricky Latupeirissa | September 11, 2019

No readers like this yet.

Why would you use the specific package management modules in stead of the package module? package is good for 95% of all use cases. I also would say that I use template more than file. I would also use command only as a last resort.

Justin W. Wheeler | September 24, 2019

No readers like this yet.

+1 for the package module. It is not that interoperable between APT and RPM distributions, but it makes it easier to write the same tasks once for CentOS/Fedora/SUSE hosts.

John Call | September 11, 2019

No readers like this yet.

Thanks for the primer article. I think, however, that Module 6's Example 2 should replace /etc/resolv.conf with /etc/hosts

Justin W. Wheeler | September 24, 2019

No readers like this yet.

The lineinfile module example sets SELinux to enforcing, but I also wanted to point out the SELinux module exists and is a better way to set and manage SELinux: https://docs.ansible.com/ansible/latest/modules/selinux_module.html

Related Content

Automate OpenStack using Ansible

How I use Ansible to add a feature to my Linux KDE desktop

Deploy applications using Foreman ACD

A note on advertising: Opensource.com does not sell advertising on the site or in any of its newsletters.