2nd Edition

Ansible for DevOps

Server and configuration management for humans

Jeff Geerling

Ansible for DevOps

Server and configuration management for humans

Jeff Geerling

This book is available at https://leanpub.com/ansible-for-devops

This version was published on 2025-05-25

ISBN 978-0-9863934-3-3

Tweet This Book!

Please help Jeff Geerling by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just purchased @Ansible4DevOps by @geerlingguy on @leanpub - https://leanpub.com/ansible-for-devops #ansible

The suggested hashtag for this book is #ansible.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#ansible

This book is dedicated to my wife, Natalie, and my children.

Editing by Margie Newman and Katherine Geerling.

Contents

Foreword

Over the last few years, Ansible has rapidly become one of the most popular IT automation tools in the world. We've seen the open source community expand from the beginning of the project in early 2012 to over 1200 individual contributors today. Ansible's modular architecture and broad applicability to a variety of automation and orchestration problems created a perfect storm for hundreds of thousands of users worldwide.

Ansible is a general purpose IT automation platform, and it can be used for a variety of purposes. From configuration management: enforcing declared state across your infrastructure, to procedural application deployment, to broad multi-component and multi-system orchestration of complicated interconnected systems. It is agentless, so it can coexist with legacy tools, and it's easy to install, configure, and maintain.

Ansible had its beginnings in 2012, when Michael DeHaan, the project's founder, took inspiration from several tools he had written prior, along with some hands-on experience with the state of configuration management at the time, and launched the project in February of 2012. Some of Ansible's unique attributes like its module-based architecture and agentless approach quickly attracted attention in the open source world.

In 2013, Said Ziouani, Michael DeHaan, and I launched Ansible, Inc. We wanted to harness the growing adoption of Ansible in the open source world, and create products to fill the gaps in the IT automation space as we saw them. The existing tools were complicated, error-prone, and hard to learn. Ansible gave users across an IT organization a low barrier of entry into automation, and it could be deployed incrementally, solving as few or as many problems as the team needed without a big shift in methodology.

This book is about using Ansible in a DevOps environment. I'm not going to try to define what DevOps is or isn't, or who's doing it or not. My personal interpretation of the idea is that DevOps is meant to shorten the distance between the developers writing the code, and the operators running the application. Now, I don't believe adding a new “DevOps” team in between existing development and operations teams achieves that objective! (Oops, now I’m trying for a definition, aren’t I?)

Well, definitions aside, one of the first steps towards a DevOps environment is choosing tools that can be consumed by both developers and operations engineers. Ansible is one of those tools: you don’t have to be a software developer to use it, and the playbooks that you write can easily be self-documenting. There have been a lot of attempts at “write once, run anywhere” models of application development and deployment, but I think Ansible comes the closest to providing a common language that’s useful across teams and across clouds and different datacenters.

The author of this book, Jeff, has been a long-time supporter, contributor, and advocate of Ansible, and he’s maintained a massive collection of impressive Ansible roles in Galaxy, the public role-sharing service maintained by Ansible, Inc. Jeff has used Ansible extensively in his professional career, and is eminently qualified to write the end-to-end book on Ansible in a DevOps environment.

As you read this book, I hope you enjoy your journey into IT automation as much as we have. Be well, do good work, and automate everything.

Tim Gerla Ansible, Inc. Co-Founder & CTO

Preface

Growing up, I had access to a world that not many kids ever get to enter. At the local radio stations where my dad was chief engineer, I was fortunate to see networks and IT infrastructure up close: Novell servers and old Mac and Windows workstations in the '90s; Microsoft and Linux-based servers; and everything in between. Best of all, he brought home decommissioned servers and copies of Linux burned to CD.

I began working with Linux and small-scale infrastructures before I started high school, and my passion for infrastructure grew as I built a Cat5 wired network and a small rack of networking equipment for a local grade school. When I started developing full-time, what was once a hobby became a necessary part of my job, so I invested more time in managing infrastructure efficiently. Over the past ten years, I've gone from manually booting and configuring physical and virtual servers; to using relatively complex shell scripts to provision and configure servers; to using configuration management tools to manage thousands of cloud servers.

When I began converting my infrastructure to code, some of the best tools for testing, provisioning, and managing my servers were still in their infancy, but they have since matured into fully-featured, robust tools that I use every day. Vagrant is an excellent tool for managing local virtual machines to mimic real-world infrastructure locally (or in the cloud), and Ansible — the subject of this book — is an excellent tool for provisioning servers, managing their configuration, and deploying applications, even on my local workstation!

These tools are still improving, and I'm excited for what the future holds. The time I invest in learning new infrastructure tools well will be helpful for years to come.

In these pages, I'll share with you all I've learned about Ansible: my favorite tool for server provisioning, configuration management, and application deployment. I hope you enjoy reading this book as much as I did writing it!

— Jeff Geerling, 2015

Second Edition

I've published 23 major revisions to the book since the original 1.0 release in 2015. After major rewrites (and three new chapters) in 2019 and 2020 to reflect Ansible's changing architecture, I decided to publish the new content as a '2nd edition'.

I will continue to publish revisions in the future, to keep this book relevant for as long as possible! Please visit the book's website, at www.ansiblefordevops.com, for the latest updates, or to subscribe to be notified of Ansible and book news!

— Jeff Geerling, 2020

Who is this book for?

Many of the developers and sysadmins I work with are at least moderately comfortable administering a Linux server via SSH, and manage between 1-100 servers, whether bare metal, virtualized, or using containers.

Some of these people have a little experience with configuration management tools (usually with Puppet or Chef), and maybe a little experience with deployments and continuous integration using tools like Jenkins, Capistrano, or Fabric. I am writing this book for these friends who, I think, are representative of most people who have heard of and/or are beginning to use Ansible.

If you are interested in both development and operations, and have at least a passing familiarity with managing a server via the command line, this book should provide you with an intermediate- to expert-level understanding of Ansible and how you can use it to manage your infrastructure.

Typographic conventions

Ansible uses a simple syntax (YAML) and simple command-line tools (using common POSIX conventions) for all its powerful abilities. Code samples and commands will be highlighted throughout the book either inline (for example: `ansible [command]`), or in a code block (with or without line numbers) like:

```
---
# This is the beginning of a YAML file.
```

Some lines of YAML and other code examples require more than 70 characters per line, resulting in the code wrapping to a new line. Wrapping code is indicated by a \ at the end of the line of code. For example:

```
# The line of code wraps due to the extremely long URL.
wget http://www.example.com/really/really/really/long/path/in/the/url/c\
auses/the/line/to/wrap
```

When using the code, don't copy the \ character, and make sure you don't use a newline between the first line with the trailing \ and the next line.

Links to pertinent resources and websites are added inline, like the following link to Ansible[1], and can be viewed directly by clicking on them in eBook formats, or by following the URL in the footnotes.

Sometimes, asides are added to highlight further information about a specific topic:

Informational asides will provide extra information.

Warning asides will warn about common pitfalls and how to avoid them.

Tip asides will give tips for deepening your understanding or optimizing your use of Ansible.

When displaying commands run in a terminal session, if the commands are run under your normal/non-root user account, the commands will be prefixed by the dollar sign ($). If the commands are run as the root user, they will be prefixed with the pound sign (#).

[1]https://www.ansible.com/

Please help improve this book!

New revisions of this book are published on a regular basis (see current book publication stats below). If you think a particular section needs improvement or find something missing, please post an issue in the Ansible for DevOps issue queue[2] (on GitHub) or contact me via Twitter (@geerlingguy[3]).

All known issues with Ansible for DevOps will be aggregated on the book's online Errata[4] page.

Current Published Book Version Information

- **Current book version**: 2.3
- **Current Ansible version as of last publication**: 11.6.0 (core 2.18.6)
- **Current Date as of last publication**: May 25, 2025

About the Author

Jeff Geerling is a developer who has worked in programming and reliability engineering for companies with anywhere between one to thousands of servers. He also manages many virtual servers for services offered by Midwestern Mac, LLC and has been using Ansible to manage infrastructure since early 2013.

[2]https://github.com/geerlingguy/ansible-for-devops/issues
[3]https://twitter.com/geerlingguy
[4]https://www.ansiblefordevops.com/errata

Introduction

In the beginning, there were sysadmins

Since the beginning of networked computing, deploying and managing servers reliably and efficiently has been a challenge. Historically, system administrators were walled off from the developers and users who interact with the systems they administer, and they managed servers by hand, installing software, changing configurations, and administering services on individual servers.

As data centers grew, and hosted applications became more complex, administrators realized they couldn't scale their manual systems management as fast as the applications they were enabling. That's why server provisioning and configuration management tools came to flourish.

Server virtualization brought large-scale infrastructure management to the fore, and the number of servers managed by one admin (or by a small team of admins), has grown by an order of magnitude. Instead of deploying, patching, and destroying every server by hand, admins now are expected to bring up new servers, either automatically or with minimal intervention. Large-scale IT deployments now may involve hundreds or thousands of servers; in many of the largest environments, server provisioning, configuration, and decommissioning are fully automated.

Modern infrastructure management

As the systems that run applications become an ever more complex and integral part of the software they run, application developers themselves have begun to integrate their work more fully with operations personnel. In many companies, development and operations work is integrated. Indeed, this integration is a requirement for modern test-driven application design.

As a software developer by trade, and a sysadmin by necessity, I have seen the power in uniting development and operations—more commonly referred to now as DevOps or Site Reliability Engineering. When developers begin to think of infrastructure as *part of their application*, stability and performance become normative. When sysadmins (most of whom have intermediate to advanced knowledge of the applications and languages being used on servers they manage) work tightly with developers, development velocity is improved, and more time is spent doing 'fun' activities like performance tuning, experimentation, and getting things done, and less time putting out fires.

> *DevOps* is a loaded word; some people argue using the word to identify both the *movement* of development and operations working more closely to automate infrastructure-related processes, and the *personnel* who skew slightly more towards the system administration side of the equation, dilutes the word's meaning. I think the word has come to be a rallying cry for the employees who are dragging their startups, small businesses, and enterprises into a new era of infrastructure growth and stability. I'm not too concerned that the term has become more of a catch-all for modern infrastructure management. My advice: spend less time arguing over the definition of the word, and more time making it mean something *to you*.

Ansible and Red Hat

Ansible was released in 2012 by Michael DeHaan (@laserllama[5] on Twitter), a developer who has been working with configuration management and infrastructure orchestration in one form or another for many years. Through his work with Puppet Labs and Red Hat (where he worked on Cobbler[6], a configuration management tool, Func, a tool for communicating commands to remote servers, and some other projects[7]), he experienced the trials and tribulations of many different organizations and individual sysadmins on their quest to simplify and automate their infrastructure management operations.

Additionally, Michael found many shops were using separate tools[8] for configuration management (Puppet, Chef, cfengine), server deployment (Capistrano, Fabric), and

[5]https://twitter.com/laserllama
[6]https://cobbler.github.io/
[7]https://web.archive.org/web/20240223204426/https://www.ansible.com/blog/2013/12/08/the-origins-of-ansible#expand
[8]https://highscalability.com/ansible-a-simple-model-driven-configuration-management-and-c/

ad-hoc task execution (Func, plain SSH), and wanted to see if there was a better way. Ansible wraps up all three of these features into one tool, and does it in a way that's actually *simpler* and more consistent than any of the other task-specific tools!

Ansible aims to be:

1. **Clear** - Ansible uses a simple syntax (YAML) and is easy for anyone (developers, sysadmins, managers) to understand. APIs are simple and sensible.
2. **Fast** - Fast to learn, fast to set up—especially considering you don't need to install extra agents or daemons on all your servers!
3. **Complete** - Ansible does three things in one, and does them very well. Ansible's 'batteries included' approach means you have everything you need in one complete package.
4. **Efficient** - No extra software on your servers means more resources for your applications. Also, since Ansible modules work via JSON, Ansible is extensible with modules written in a programming language you already know.
5. **Secure** - Ansible uses SSH, and requires no extra open ports or potentially-vulnerable daemons on your servers.

Ansible also has a lighter side that gives the project a little personality. As an example, Ansible's major releases used to be named after Led Zeppelin songs (e.g. 2.0 was named after 1973's "Over the Hills and Far Away", 1.x releases were named after Van Halen songs). Additionally, Ansible uses `cowsay`, if installed, to wrap output in an ASCII cow's speech bubble (this behavior can be disabled in Ansible's configuration).

Ansible, Inc.[9] was founded by Saïd Ziouani, Michael DeHaan, and Tim Gerla, and acquired by Red Hat in 2015. The Ansible team oversees core Ansible development and development of the Red Hat Ansible Automation Platform[10] for organizations using Ansible. Hundreds of individual developers have contributed patches to Ansible, and Ansible is the most starred infrastructure management tool on GitHub (with over 64,000 stars as of this writing).

In October 2015, Red Hat acquired Ansible, Inc., and has proven itself to be a good steward and promoter of Ansible. I see no indication of this changing in the future.

[9]https://www.ansible.com/
[10]https://www.redhat.com/en/technologies/management/ansible

Ansible Examples

There are many Ansible examples (playbooks, roles, infrastructure, configuration, etc.) throughout this book. Most of the examples are in the Ansible for DevOps GitHub repository[11], so you can browse the code in its final state while you're reading the book. Some of the line numbering may not match the book *exactly* (especially if you're reading an older version of the book!), but I will try my best to keep everything synchronized over time.

Other resources

We'll explore all aspects of using Ansible to provision and manage your infrastructure in this book, but there's no substitute for the wealth of documentation and community interaction that make Ansible great. Check out the links below to find out more about Ansible and discover the community:

- Ansible Documentation[12] - Covers all Ansible options in depth. There are few open source projects with documentation as clear and thorough.
- Ansible Glossary[13] - If there's ever a term in this book you don't seem to fully understand, check the glossary.
- The Bullhorn[14] - Ansible's official newsletter.
- Ansible Mailing List[15] - Discuss Ansible and submit questions with Ansible's community via this Google group.
- Ansible on GitHub[16] - The official Ansible code repository, where the magic happens.
- Ansible Example Playbooks on GitHub[17] - Many examples for common server configurations.
- Getting Started with Ansible[18] - A simple guide to Ansible's community and resources.

[11]https://github.com/geerlingguy/ansible-for-devops
[12]https://docs.ansible.com/ansible/
[13]https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html
[14]https://us19.campaign-archive.com/home/?u=56d874e027110e35dea0e03c1&id=d6635f5420
[15]https://groups.google.com/forum/#!forum/ansible-project
[16]https://github.com/ansible/ansible
[17]https://github.com/ansible/ansible-examples
[18]https://www.ansible.com/resources/get-started

- Ansible Blog[19]

I'd like to especially highlight Ansible's documentation (the first resource listed above); one of Ansible's greatest strengths is its well-written and extremely relevant documentation, containing a large number of relevant examples and continuously-updated guides. Very few projects—open source or not—have documentation as thorough, yet easy-to-read. This book is meant as a supplement to, not a replacement for, Ansible's documentation!

[19]https://www.ansible.com/blog

Chapter 1 - Getting Started with Ansible

Ansible and Infrastructure Management

On snowflakes and shell scripts

Many developers and system administrators manage servers by logging into them via SSH, making changes, and logging off. Some of these changes would be documented, some would not. If an admin needed to make the same change to many servers (for example, changing one value in a config file), the admin would manually log into *each* server and repeatedly make this change.

If there were only one or two changes in the course of a server's lifetime, and if the server were extremely simple (running only one process, with one configuration, and a very simple firewall), *and* if every change were thoroughly documented, this process wouldn't be a problem.

But for almost every company in existence, servers are more complex—most run tens, sometimes hundreds of different applications or application containers. Most servers have complicated firewalls and dozens of tweaked configuration files. And even with change documentation, the manual process usually results in some servers or some steps being forgotten.

If the admins at these companies wanted to set up a new server *exactly* like one that is currently running, they would need to spend a good deal of time going through all of the installed packages, documenting configurations, versions, and settings; and they would spend a lot of unnecessary time manually reinstalling, updating, and tweaking everything to get the new server to run close to how the old server did.

Some admins may use shell scripts to try to reach some level of sanity, but I've yet to see a complex shell script that handles all edge cases correctly while synchronizing multiple servers' configuration and deploying new code.

Configuration management

Lucky for you, there are tools to help you avoid having these *snowflake servers*—servers that are uniquely configured and impossible to recreate from scratch because they were hand-configured without documentation. Tools like CFEngine[20], Puppet[21] and Chef[22] became very popular in the mid-to-late 2000s.

But there's a reason why many developers and sysadmins stick to shell scripting and command-line configuration: it's simple and easy-to-use, and they've had years of experience using bash and command-line tools. Why throw all that out the window and learn a new configuration language and methodology?

Enter Ansible. Ansible was built (and continues to be improved) by developers and sysadmins who know the command line—and want to make a tool that helps them manage their servers exactly the same as they have in the past, but in a repeatable and centrally managed way. Ansible also has other tricks up its sleeve, making it a true Swiss Army knife for people involved in DevOps (not just the operations side).

One of Ansible's greatest strengths is its ability to run regular shell commands verbatim, so you can take existing scripts and commands and work on converting them into idempotent playbooks as time allows. For someone (like me) who was comfortable with the command line, but never became proficient in more complicated tools like Puppet or Chef (which both required at least a *slight* understanding of Ruby and/or a custom language just to get started), Ansible was a breath of fresh air.

Ansible works by pushing changes out to all your servers (by default), and requires no extra software to be installed on your servers (thus no extra memory footprint, and no extra daemon to manage), unlike most other configuration management tools.

[20]http://cfengine.com/
[21]http://puppetlabs.com/
[22]http://www.getchef.com/chef/

Idempotence is the ability to run an operation which produces the same result whether run once or multiple times (source[23]).

An important feature of a configuration management tool is its ability to ensure the same configuration is maintained whether you run it once or a thousand times. Many shell scripts have unintended consequences if run more than once, but Ansible deploys the same configuration to a server over and over again without making any changes after the first deployment.

In fact, almost every aspect of Ansible modules and commands is idempotent, and for those that aren't, Ansible allows you to define when the given command should be run, and what constitutes a changed or failed command, so you can easily maintain an idempotent configuration on all your servers.

Installing Ansible

Ansible's only real dependency is Python. Once Python is installed, the simplest way to get Ansible running is to use `pip`, a simple package manager for Python.

If you're on a Mac, installing Ansible is a piece of cake:

1. Check if `pip` is installed (`which pip`). If not, install it: `sudo easy_install pip`
2. Install Ansible: `pip install ansible`

You could also install Ansible via Homebrew[24] with `brew install ansible`. Either way (`pip` or `brew`) is fine, but make sure you update Ansible using the same system with which it was installed!

If you're running Windows, it will take a little extra work to set everything up. Typically, people run Ansible inside the Windows Subsystem for Linux. For detailed instructions setting up Ansible under the WSL, see Appendix A - Using Ansible on Windows workstations.

If you're running Linux, chances are you already have Ansible's dependencies installed, but we'll cover the most common installation methods.

[23]http://en.wikipedia.org/wiki/Idempotence#Computer_science_meaning
[24]http://brew.sh/

If you have `python-pip` and `python-devel` (`python-dev` on Debian/Ubuntu) installed, use `pip` to install Ansible (this assumes you also have the 'Development Tools' package installed, so you have `gcc`, `make`, etc. available):

```
$ pip install ansible
```

Using pip allows you to upgrade Ansible with `pip install --upgrade ansible`.

Fedora/Red Hat Enterprise Linux

The easiest way to install Ansible on an RPM-based OS is to use the official dnf package. If you're running Red Hat Enterprise Linux (RHEL) or CentOS/Rocky/Alma Linux, you need to install EPEL's RPM before you install Ansible (see the info section below for instructions):

```
$ dnf -y install ansible
```

On RPM-based systems, `python-pip` and `ansible` are available via the EPEL repository[25]. If you run the command `dnf repolist | grep epel` (to see if the EPEL repo is already available) and there are no results, you need to install it with the following commands:

```
# If you're on RHEL/CentOS 6:
$ rpm -ivh http://dl.fedoraproject.org/pub/epel/6/x86_64/\
epel-release-6-8.noarch.rpm
# If you're on RHEL/CentOS 7:
$ yum install epel-release
# If you're on RHEL 8+/Fedora:
$ dnf install epel-release
```

Debian/Ubuntu

The easiest way to install Ansible on a Debian or Ubuntu system is to use the official apt package.

[25] https://fedoraproject.org/wiki/EPEL

```
$ sudo apt-add-repository -y ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install -y ansible
```

If you get an error like "sudo: add-apt-repository: command not found", you're probably missing the `software-properties-common` package. Install it with the command:

Once

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

Ansible is installed, make sure it's working properly by entering `ansible --version` on the command line. You should see the currently-installed version information:

```
$ ansible --version
ansible [core 2.14.6]
  ...
  python version = 3.10.11
  jinja version = 3.1.2
  libyaml = True
```

What about Python 3? If you have both Python 2 and Python 3 installed, and `pip` is aliased to an older Python 2 version of `pip`, you should consider installing Python 3 and `pip3`, and using that version instead. Ansible is fully compatible with Python 3, and unless you're running on a very old system that doesn't have Python 3 available for it, you should use Python 3.

Creating a basic inventory file

Ansible uses an inventory file (basically, a list of servers) to communicate with your servers. Like a hosts file (at `/etc/hosts`) that matches IP addresses to domain names, an Ansible inventory file matches servers (IP addresses or domain names) to groups. Inventory files can do a lot more, but for now, we'll just create a simple file with one server. Create a file named `hosts.ini` in a test project folder:

```
$ mkdir test-project
$ cd test-project
$ touch hosts.ini
```

Inventory file names do not have to follow any particular naming convention. I often use the file name `hosts.ini` for Ansible's default 'ini-style' syntax, but I also sometimes call the file `inventory` (with no file extension).

Edit this hosts file with nano, vim, or whatever editor you'd like. Put the following into the file:

```
[example]
www.example.com
```

...where `example` is the group of servers you're managing and `www.example.com` is the domain name (or IP address) of a server in that group. If you're not using port 22 for SSH on this server, you will need to add it to the address, like `www.example.com:2222`, since Ansible defaults to port 22 and won't get this value from your ssh config file.

This first example assumes you have a server set up that you can test with; if you don't already have a spare server somewhere that you can connect to, you might want to create a small VM using DigitalOcean, Amazon Web Services, Linode, or some other service that bills by the hour. That way you have a full server environment to work with when learning Ansible—and when you're finished testing, delete the server and you'll only be billed a few pennies!

Replace the `www.example.com` in the above example with the name or IP address of your server.

You can also place your inventory in Ansible's global inventory file, `/etc/ansible/hosts`, and any playbook will default to that if no other inventory is specified. However, that file requires `sudo` permissions and it is usually better to maintain inventory alongside your Ansible projects.

Running your first Ad-Hoc Ansible command

Now that you've installed Ansible and created an inventory file, it's time to run a command to see if everything works! Enter the following in the terminal (we'll do something safe so it doesn't make any changes on the server):

```
$ ansible -i hosts.ini example -m ping -u [username]
```

…where `[username]` is the user you use to log into the server. If everything worked, you should see a message that shows `www.example.com | SUCCESS >>`, then the result of your ping. If it didn't work, run the command again with `-vvvv` on the end to see verbose output. Chances are you don't have SSH keys configured properly—if you login with `ssh username@www.example.com` and that works, the above Ansible command should work, too.

Ansible assumes you're using passwordless (key-based) login for SSH (e.g. you login by entering `ssh username@example.com` and don't have to type a password). If you're still logging into your remote servers with a username and password, or if you need a primer on Linux remote authentication and security best practices, please read Chapter 11 - Server Security and Ansible. If you insist on using passwords, add the `--ask-pass` (`-k`) flag to Ansible commands (you may also need to install the `sshpass` package for this to work). This entire book is written assuming passwordless authentication, so you'll need to keep this in mind every time you run a command or playbook.

Need a primer on SSH key-based authentication? Please read through Ubuntu's community documentation on SSH/OpenSSH/Keys[26].

Let's run a more useful command:

[26]https://help.ubuntu.com/community/SSH/OpenSSH/Keys

```
$ ansible -i hosts.ini example -a "free -h" -u [username]
```

In this example, we quickly see memory usage (in a human-readable format) on all the servers (for now, just one) in the `example` group. Commands like this are helpful for quickly finding a server that has a value out of a normal range. I often use commands like `free -h` (to see memory statistics), `df -h` (to see disk usage statistics), and the like to make sure none of my servers is behaving erratically. While it's good to track these details in an external tool like Nagios[27], Munin[28], or Cacti[29], it's also nice to check these stats on all your servers with one simple command and one terminal window!

Summary

That's it! You've just learned about configuration management and Ansible, installed it, told it about your server, and ran a couple commands on that server through Ansible. If you're not impressed yet, that's okay—you've only seen the *tip* of the iceberg.

```
 _______________________________________
/ A doctor can bury his mistakes but an \
| architect can only advise his clients |
\ to plant vines. (Frank Lloyd Wright)  /
 ---------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

[27]http://www.nagios.org/
[28]http://munin-monitoring.org/
[29]http://www.cacti.net/

Chapter 2 - Local Infrastructure Development: Ansible and Vagrant

Prototyping and testing with local virtual machines

Ansible works well with any server to which you can connect—remote *or* local. For speedier testing and development of Ansible playbooks, and for testing in general, it's a very good idea to work locally. Local development and testing of infrastructure is both safer and faster than doing it on remote/live machines—especially in production environments!

> In the past decade, test-driven development (TDD), in one form or another, has become the norm for much of the software industry. Infrastructure development hasn't been as organized until recently, and best practices dictate that infrastructure (which is becoming more and more important to the software that runs on it) should be thoroughly tested as well.
>
> Changes to software are tested either manually or in some automated fashion; there are now systems that integrate both with Ansible and with other deployment and configuration management tools, to allow some amount of infrastructure testing as well. Even if it's just testing a configuration change locally before applying it to production, that approach is a thousand times better than what, in the software development world, would be called 'cowboy coding'—working directly in a production environment, not documenting or encapsulating changes in code, and not having a way to roll back to a previous version.

The past decade has seen the growth of many virtualization tools that allow for flexible and very powerful infrastructure emulation, all from your local workstation!

It's empowering to be able to play around with a config file, or to tweak the order of a server update to perfection, over and over again, with no fear of breaking an important server. If you use a local virtual machine, there's no downtime for a server rebuild; just re-run the provisioning on a new VM, and you're back up and running in minutes—with no one the wiser.

Vagrant[30], a server provisioning tool, and VirtualBox[31], a local virtualization environment, make a potent combination for testing infrastructure and individual server configurations locally. Both applications are free and open source, and work well on Mac, Linux, or Windows hosts.

We're going to set up Vagrant and VirtualBox for easy testing with Ansible to provision a new server.

Your first local server: Setting up Vagrant

To get started with your first local virtual server, you need to download and install Vagrant and VirtualBox, and set up a simple Vagrantfile, which will describe the virtual server.

1. Download and install Vagrant and VirtualBox (whichever version is appropriate for your OS): - Download Vagrant[32] - Download VirtualBox[33] (when installing, make sure the command line tools are installed, so Vagrant works with it)
2. Create a new folder somewhere on your hard drive where you will keep your Vagrantfile and provisioning instructions.
3. Open a Terminal or PowerShell window, then navigate to the folder you just created.
4. Add a Rocky Linux 8.x 64-bit 'box' using the `vagrant box add`[34] command: `vagrant box add geerlingguy/rockylinux8` (note: HashiCorp's Vagrant Cloud[35] has a comprehensive list of different pre-made Linux boxes. Also, check out the 'official' Vagrant Ubuntu boxes in Vagrant's Boxes documentation[36].

[30]https://www.vagrantup.com
[31]https://www.virtualbox.org/
[32]https://www.vagrantup.com/downloads.html
[33]https://www.virtualbox.org/wiki/Downloads
[34]https://www.vagrantup.com/docs/boxes.html
[35]https://app.vagrantup.com/boxes/search
[36]https://www.vagrantup.com/docs/boxes.html

5. Create a default virtual server configuration using the box you just downloaded: `vagrant init geerlingguy/rockylinux8`
6. Boot your Rocky Linux server: `vagrant up`

Vagrant downloaded a pre-built 64-bit Rocky Linux 8 virtual machine image (you can build your own[37] virtual machine 'boxes', if you so desire), loaded the image into VirtualBox with the configuration defined in the default Vagrantfile (which is now in the folder you created earlier), and booted the virtual machine.

Managing this virtual server is extremely easy: `vagrant halt` will shut down the VM, `vagrant up` will bring it back up, and `vagrant destroy` will completely delete the machine from VirtualBox. A simple `vagrant up` again will re-create it from the base box you originally downloaded.

Now that you have a running server, you can use it just like you would any other server, and you can connect via SSH. To connect, enter `vagrant ssh` from the folder where the Vagrantfile is located. If you want to connect manually, or connect from another application, enter `vagrant ssh-config` to get the required SSH details.

Using Ansible with Vagrant

Vagrant's ability to bring up preconfigured boxes is convenient on its own, but you could do similar things with the same efficiency using VirtualBox's (or VMWare's, or Parallels') GUI. Vagrant has some other tricks up its sleeve:

- **Network interface management**[38]: You can forward ports to a VM, share the public network connection, or use private networking for inter-VM and host-only communication.
- **Shared folder management**[39]: Vagrant sets up shares between your host machine and VMs using NFS or (much slower) native folder sharing in VirtualBox.
- **Multi-machine management**[40]: Vagrant is able to configure and control multiple VMs within one Vagrantfile. This is important because, as stated in the

[37]https://www.vagrantup.com/docs/providers/virtualbox/boxes.html
[38]https://www.vagrantup.com/docs/networking
[39]https://www.vagrantup.com/docs/synced-folders
[40]https://www.vagrantup.com/docs/multi-machine

documentation, "Historically, running complex environments was done by flattening them onto a single machine. The problem with that is that it is an inaccurate model of the production setup, which behaves far differently."
- **Provisioning**[41]: When running `vagrant up` the first time, Vagrant automatically *provisions* the newly-minted VM using whatever provisioner you have configured in the Vagrantfile. You can also run `vagrant provision` after the VM has been created to explicitly run the provisioner again.

It's this last feature that is most important for us. Ansible is one of many provisioners integrated with Vagrant (others include basic shell scripts, Chef, Docker, Puppet, and Salt). When you call `vagrant provision` (or `vagrant up` the first time), Vagrant passes off the VM to Ansible, and tells Ansible to run a defined Ansible playbook. We'll get into the details of Ansible playbooks later, but for now, we're going to edit our Vagrantfile to use Ansible to provision our virtual machine.

Open the Vagrantfile that was created when we used the `vagrant init` command earlier. Add the following lines just before the final 'end' (Vagrantfiles use Ruby syntax, in case you're wondering):

```
# Provisioning configuration for Ansible.
config.vm.provision "ansible" do |ansible|
  ansible.playbook = "playbook.yml"
end
```

This is a very basic configuration to get you started using Ansible with Vagrant. There are many other Ansible options[42] you can use once we get deeper into using Ansible. For now, we just want to set up a very basic playbook—a simple file you create to tell Ansible how to configure your VM.

Your first Ansible playbook

Let's create the Ansible `playbook.yml` file now. Create an empty text file in the same folder as your Vagrantfile, and put in the following contents:

[41] https://www.vagrantup.com/docs/provisioning
[42] https://www.vagrantup.com/docs/provisioning/ansible_intro

```
---
- hosts: all
  become: yes

  tasks:
  - name: Ensure chrony (for time synchronization) is installed.
    dnf:
      name: chrony
      state: present

  - name: Ensure chrony is running.
    service:
      name: chronyd
      state: started
      enabled: yes
```

I'll get into what this playbook is doing in a minute. For now, let's run the playbook on our VM. Make sure you're in the same directory as the Vagrantfile and new playbook.yml file, and enter `vagrant provision`. You should see status messages for each of the 'tasks' you defined, and then a recap showing what Ansible did on your VM—something like the following:

```
PLAY RECAP **********************************************************
default              : ok=3    changed=0    unreachable=0    failed=0
```

Ansible just took the simple playbook you defined, parsed the YAML syntax, and ran a bunch of commands via SSH to configure the server as you specified. Let's go through the playbook, step by step:

```
---
```

This first line is a marker showing that the rest of the document will be formatted in YAML (read an introduction to YAML[43]).

[43]https://yaml.org/spec/1.2.2/

```
- hosts: all
```

This line tells Ansible to which hosts this playbook applies. `all` works here, since Vagrant is invisibly using its own Ansible inventory file (instead of using a manually-created `hosts.ini` file), which just defines the Vagrant VM.

```
  become: yes
```

Since we need privileged access to install chrony and modify system configuration, this line tells Ansible to use `sudo` for all the tasks in the playbook (you're telling Ansible to 'become' the root user with `sudo`, or an equivalent).

```
  tasks:
```

All the tasks after this line will be run on all hosts (or, in our case, our one VM).

```
  - name: Ensure chrony (for time synchronization) is installed.
    dnf:
      name: chrony
      state: present
```

This command is the equivalent of running `dnf install chrony`, but is much more intelligent; it will check if chrony is installed, and, if not, install it. This is the equivalent of the following shell script:

```
if ! rpm -qa | grep -qw chrony; then
    dnf install -y chrony
fi
```

However, the above script is still not quite as robust as Ansible's `dnf` command. What if some other package with `chrony` in its name is installed, but not `chrony`? This script would require extra tweaking and complexity to match the simple Ansible dnf command, especially after we explore the dnf module more intimately (or the `apt` module for Debian-flavored Linux, or `package` for OS-agnostic package installation).

```
  - name: Ensure chrony is running.
    service:
      name: chronyd
      state: started
      enabled: yes
```

This final task both checks and ensures that the `chronyd` service is started and running, and sets it to start at system boot. A shell script with the same effect would be:

```
# Start chronyd if it's not already running.
if ps aux | grep -q "[c]hronyd"
then
    echo "chronyd is running." > /dev/null
else
    systemctl start chronyd.service > /dev/null
    echo "Started chronyd."
fi
# Make sure chronyd is enabled on system startup.
systemctl enable chronyd.service
```

You can see how things start getting complex in the land of shell scripts! And this shell script is still not as robust as what you get with Ansible. To maintain idempotency and handle error conditions, you'll have to do even more work with basic shell scripts than you do with Ansible.

We could be more terse (and demonstrate Ansible's powerful simplicity) ignoring Ansible's self-documenting `name` parameter and shorthand `key=value` syntax, resulting in the following playbook:

```yaml
---
- hosts: all
  become: yes
  tasks:
  - dnf: name=chrony state=present
  - service: name=chronyd state=started enabled=yes
```

Just as with code and configuration files, documentation in Ansible (e.g. using the `name` parameter and/or adding comments to the YAML for complicated tasks) is not absolutely necessary. However, I'm a firm believer in thorough (but concise) documentation, so I always document what my tasks will do by providing a `name` for each one. This also helps when you're running the playbooks, so you can see what's going on in a human-readable format.

Cleaning Up

Once you're finished experimenting with the Rocky Linux Vagrant VM, you can remove it from your system by running `vagrant destroy`. If you want to rebuild the VM again, run `vagrant up`. If you're like me, you'll soon be building and rebuilding hundreds of VMs and containers per week using Vagrant and Ansible!

Summary

Your workstation is on the path to becoming an "infrastructure-in-a-box," and you can now ensure your infrastructure is as well-tested as the code that runs on top of it. With one small example, you've got a glimpse at the simple-yet-powerful Ansible playbook. We'll dive deeper into Ansible playbooks later, and we'll also explore Vagrant a little more as we go.

```
 ______________________________________
/ I have not failed, I've just found   \
| 10,000 ways that won't work. (Thomas |
\ Edison)                              /
 --------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```