Testing Ansible

(BECAUSE YOU REALLY SHOULD)

David Norman - deekayen

Testing Ansible

Because you really should

David N

This book is for sale at http://leanpub.com/testingansible

This version was published on 2022-05-31

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Preface

What this book covers

Who this book is for

System administrators, software developers, networking specialists, security operations workers who care about the technical problems and fine details of project implementation. Python experience is not required for this text, though experience with YAML and Ansible is presumed. You will be at an advantage if you already have experience with automating projects in tools like Gitlab Runners, Jenkins, or Travis CI.

Who this book is not for

If you're not comfortable with the basics of writing Ansible playbooks and roles in YAML, this is not the place to learn.

Other resources

Unfortunately this text will likely be outdated or materially incomplete as soon as I think I finished. Check out the "Other Tools and Programs[1]" page in the official Ansible documentation for the latest list of tools you should know about as you implement testing for your Ansible library.

- https://www.jeffgeerling.com/blog

[1] https://bit.ly/3D5W3Bo

Conventions

I think of Debian, Ubuntu, and Mint configuration as interchangeable, same as for CentOS, Red Hat Enterprise Linux, and Fedora. I realize this is not entirely true, but it serves as my reality. My primary workstation is a MacBook Pro and I won't do much to hide my bias in favor of working in macOS, though I use Linux and Windows examples for variety.

Code words, command line utilities, directories and folder names, paths, URLs, user input, variables, and database tables display in `monospace, constant width`.

Formatted and code blocks

A block of terminal input is set as follows:

```
$ sudo su -
# cat ~/Downloads/ascii_druplicon.txt > /etc/motd
# exit
```

Example Ansible playbook output is formatted as json.

```
ok: [localhost] => {
        "censored": "the output has been hidden due to the fact that 'no_log: true' was specified for \
this result",
        "changed": false
    }
```

A diff, to show how a file changes as part of a bugfix:

```
diff --git a/molecule/default/molecule.yml b/molecule/default/molecule.yml
--- a/molecule/default/molecule.yml
+++ b/molecule/default/molecule.yml
@@ -10,6 +10,7 @@ platforms:
     image: centos7
 provisioner:
   name: ansible
+  log: True
   lint:
     name: ansible-lint
 verifier:
```

Line-wrapping

Some commands, configuration, or output cannot fit on a single line of text in book format. Backslashes at the ends of line represent a soft line break for line-wrapping. For example, each of these lines with a backslash at the end represents a single line in a terminal or configuration file.

```
ruby -e "$(curl -fsSL \
https://raw.github.com/Homebrew/homebrew/go/install)"
```

```
launchctl load \
~/Library/LaunchAgents/homebrew.mxcl.percona-server.plist
```

```
SSLCipherSuite "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384 EECDH+ECDSA+SHA256 EECDH+aRSA+\
SHA384 EECDH+aRSA+SHA256 EECDH+aRSA+RC4 EECDH EDH+aRSA RC4 !aNULL !eNULL !LOW !3DES !MD5 !EXP !PSK !SR\
P !DSS +RC4 RC4"
```

Text blocking, asides, sidebars

> This indented block highlights a quote or block of unique text.

> This signifies an aside.

This icon signifies a tip or suggestion.

This icon signifies a warning.

This is an exercise. Any commands presented are good practice or to test proper configuration.

Reader feedback, errata, questions

I care about fixing errors in this text, but your mere possession of this text doesn't entitle you to my time for answering your questions. I'm much more likely to respond to questions sent to `david@dkn.email` if you have questions related to the clarity of this text. I'm not likely to respond if you have a general question about a Mac OS utility or about Ansible development unless you're sending money, too (see also GitHub Sponsors[2], Paypal[3]).

Who am I? Why did I write this book?

When my son joined a Cub Scout pack, our first meeting in the school gym was chaotic. I remember standing against the cold, cinder block wall wondering why

[2] https://github.com/sponsors/deekayen
[3] https://www.paypal.com/paypalme2/deekayen

"someone" didn't take control of the meeting and fix it. As a few months passed, that "someone" never really materialized. I later had the good fortune to enroll in Wood Badge, a leadership training course offered by the Boy Scouts of America to adult leaders. Wood Badge helped me realize sometimes **I* may be the someone I'm waiting on.*

Much of my experience in IT was spent in Drupal development, as a PHP developer. As a developer, I learned about all sorts of tools I could use to maintain high code quality, and the sort of code that would be easily shared with and used by other developers in an open-source, public space.

As the team of Ansible developers started growing in my day job, I was spending a lot of time piecing together bits of tools and automation like I had as a PHP developer. As I did so, I found myself asking again, "why doesn't someone have this all written down somehwere?!" And there it was, the "someone" again.

I struggled, in starting this book, with trying to determine content which wouldn't bore or insult my audience. In taking some advice from On Writing Well[4], what I intend to present in the following pages is simply the conclusions I've drawn in my ability to research the related topics. My motivation in this essay is merely to write for myself. If I do not entertain myself, I will not finish. I am my own editor. I write the way I converse with people in speech, which won't match collegiate writing style guides.

I will embed opinions which are controversial - others are merely my opinion on issues and have no real correct and final answer. In some cases, I may present you with incorrect or inexperienced advice, which I hope you'll give me the opportunity to amend with time and experience.

These are my notes which I share with you. I hope you find them useful.

[4] http://www.amazon.com/gp/product/B0090RVGW0/

Chapter 1

Spend any time with an experienced software developer, and they're likely to tell you about test-driven development or TDD. In short, it's a development philosophy where you write tests to confirm the features and functionality of your software project before you generate any features or code. When you have a complete suite of tests, they should all fail until you write the matching functionality to make the tests pass.

Often when software developers are given a new project, they drive right into generating new code, building something, then loading it in a web browser, compiling it, or generating an output file. Then they load website or the CSV file and check to see if it looks like what they expected. Find errors, fix them, repeat.

The problems start happening when you add a second developer. Two developers never think the same and they don't test the same, either. Rarely does a developer look at another's block of code and rave about its perfection. That's why developers of popular languages use linters.

Linters follow apply a set of standards to scan the formatting of code in a software project to look for consistent spacing, indents, function arguments, line length, capitalization, deprecated libraries, and such. Anyone who has published an Ansible role to Red Hat's Galaxy website has it scanned by the ansible-lint and yamllint utilities to help users get a quick assessment of the role's code quality. On Galaxy, the linters' review is represented with a 0-5 score.

The score on Galaxy is a representation of the author's understanding of Ansible, their attention to detail, and knowledge of best practices. When working in a group, each of these things are important for the team members to follow the same rules of grammar and syntax as their peers. It's a courtesy for people that follow the original author to easily understand what they're reading.

Hang around developers long enough, and you'll year one grouse, "this code is garbage. The best thing to do is start over!" Take these two blocks of PHP code, for example. They're equally valid to the PHP engine, however one will bring anger to the eyes of a fellow team member.

```
<?php
function recursion( $a = 'nothing' )
{
    if ( $a < 20 )
    {
        echo "$a\n";
        recursion($a + 1);
    }
}
?>
```

```
<?
function recursion($a="nothing"){
  if($a<20){
    echo "$a\n";
    recursion($a + 1);
  }
}
?>
```

When scanning each code block with `phplint`, it reports helpful warnings like `Warning: using deprecated short tag <? -- Hint: use <?php instead`. Linters also report helpful warnings for things you forgot - like inline documentation and metadata.

When you need to do something one time and quickly, Ansible's ad-hoc command lets you run at scale without worrying about sharing your work with others. Using roles in Ansible is the opposite; they're reusable blocks of tasks intended to share with others. When initializing a new Ansible role with the `ansible-galaxy` utility, the first thing many people do is delete the `meta` and `tests` folders automatically generated with the template, which are of the most important parts for helping others understand how to use your role and what it does!

One of the simplest ways to get started with testing your playbooks is built-in to Ansible already!

```
$ ansible-playbook --syntax-check -i 127.0.0.1, playbook.yml
```

The ansible-lint utility has a suite of tests which examine playbooks and roles for deprecations, formatting, commands vs shell, idioms, and metadata. For each of the linter inspections, there's a severity score applied which are classified VERY_LOW,

LOW, MEDIUM, HIGH, or VERY_HIGH. High findings are considered errors, mediums show as warnings.

Integration tests using ansible-test[5]

Setting reminders

Sometimes people just need a little reminder that they should stick to the configuration managed by Ansible. It happens - you used to edit a file with `vi`, finally got the motivation to put it under control of Ansible, and then your co-worker can't break the habit. Picture yourself opening your NTP configuration and the first thing you see is this:

```
#
# Ansible managed
#
```

Ansible offers styles for comments in "plain" (#...), C (//...), C block (/.../), Erlang (%...) and XML ():

```
{{ ansible_managed | comment }}
{{ ansible_managed | comment('c') }}
{{ ansible_managed | comment('cblock') }}
{{ ansible_managed | comment('erlang') }}
{{ ansible_managed | comment('xml') }}
{{ ansible_managed | comment('plain', prefix='###\n#', postfix='#\n###') }}
```

You can setup a template like this that updates the location of the template, the last timestamp for edits, and so forth with the `ansible_managed` variable in your templates. I suggest even if you think you only have a static file that doesn't require variable information, put it in your role's `templates` folder instead of putting a static file in the `files` folder.

To make the output of the `ansible_managed` variable more valuable, change the configuration in `ansible.cfg` to this:

[5] https://bit.ly/3F4IFxF

```
[defaults]

ansible_managed = This file is managed by Ansible.%n
  template: {file}
  date: %Y-%m-%d %H:%M:%S
  user: {uid}
  host: {host}
```

Then the output would look more like this:

```
#
# This file is managed by Ansible.
#
# template: ~/time_management/roles/ntp/templates/ntp.conf.j2
# date: 2021-10-29 22:11:14
# user: ansible
# host: henlein
#
```

`ansible_managed` practice

The Testing Ansible Examples[6] companion git repository has an example folder for `ansible_managed` for you to practice deploying a managed configuration file, including a customized `ansible.cfg` file. Run the playbook more than once - you'll find that even if you insert a timestamp in your `ansible_managed` output, Ansible is smart enough to not update the changed status of the file if the rest of the contents in the template haven't also changed. It's a perfect example of Ansible's idempotent capabilities.

etckeeper

A trick I like to use with my servers is to install `etckeeper`[7]. It serves as a tattle tale when someone edits the configuration on my systems. It can also expose files in `/etc` that people are manually editing, which you didn't think about putting under Ansible's care and feeding. Any time someone, including Ansible, updates a file in `/etc`, it automatically commits and pushes the changes to a git, bzr, mercurial, or

[6] https://bit.ly/3kp9ftz
[7] https://bit.ly/3CXgfVT

darcs repo. From there, you can have a receive hook email you with a diff of the changes.

etckeeper ansible role

I made an etckeeper role[8] for RedHat which can install etckeeper and setup a GitLab repository for receiving configuration changes from updates to `/etc`. It was setup for a very specific environment, but should serve as a good starting point for everything you need to change it to fit your environment.

[8] https://bit.ly/3bXhad3