Salt to Ansible Migration Guide

Jude Nagurney



Contents

Copyright . . . 5
Introduction . . . 6
- Audience . . . 6
- Acknowledgements . . . 7
- Disclaimers . . . 7
  - Software Versions . . . 7
  - AI Usage . . . 7
  - Release History . . . 8

Migration Guiding Principles . . . 9
- Reduce External Dependencies . . . 9
- Check Data as Much as Possible . . . 9
- Avoid Refactoring While Migrating . . . 9

Major Salt/Ansible Differences . . . 10
- Ansible Provisions Hosts Via SSH . . . 10
- Ansible Is YAML-First While Salt is Jinja2-First . . . 10
- Ansible Calls Function Calls With ansible.builtin.import_tasks Instead of Jinja2 Macros . . . 10
- Main Ansible Source is Under tasks/main.yml Instead of Init.sls . . . 10
- Ansible Runs Tasks Sequentially . . . 10
- Ansible Names Are Descriptive Only . . . 11
- Ansible Uses host_vars and group_var Files Instead of pillar files . . . 11
- Ansible Uses A Global ansible_facts Map Instead Of Grains . . . 11
- Ansible Uses a lookup() Function Instead of external pillar . . . 12

Migration Workflow . . . 13
- Scrub the Salt-Specific Tickets and Repos . . . 13
- Inventory All the Hosts . . . 13
- Survey the Current Salt Usage . . . 14
- Expect Emergency Salt Fixes . . . 14
- Configure SSH Access To The Hosts . . . 14
- Create ansible-vault Keys . . . 15
- Migrate Pillar Data . . . 16
  - Build up a JSON Schema . . . 16
  - Migrate ext_pillar Data to inventory/host_data Files . . . 18
  - Construct the Ansible site.yml File . . . 18
- Analyze Salt States . . . 19
- Port Salt init.sls States to Ansible Tasks . . . 20
- Create a argument_specs.yml File for the Role . . . 20
- Port Salt MACRO.sls States to Ansible Tasks . . . 20
- Test Out the New Role Code . . . 22
- Work the Salt-specific Edge Case Tickets . . . 23
- Use Ansible in Production . . . 23
- Post Migration Tasks . . . 23

Ansible File Layout . . . 24
Jinja2 Migration Guide . . . 28
- Jinja2 Variable Assignment . . . 28
- Jinja2 Conditional Blocks . . . 29

Jinja2 Loops . . . 31
Jinja2 Macros . . . 33
Salt Prerequisites Migration Guide . . . 34
watch_in/watch . . . 34
onlyif/unless . . . 35
require/require_in . . . 36
order: last . . . 36
Specific Salt States Migration Guide . . . 37
Running in no-op Mode . . . 38
cmd.run (Single Command) . . . 39
cmd.run (Shell Commands) . . . 40
cmd.run (wget or curl Commands) . . . 41
file.directory . . . 42
file.absent . . . 43
file.exists . . . 44
file.managed (without source) . . . 45
file.managed (With a 'salt://' URI Source) . . . 46
file.managed (With a Path Source) . . . 47
file.managed (With Contents) . . . 48
file.managed (With contents_pillar) . . . 49
file.managed (With Template) . . . 50
file.missing . . . 52
file.recurse . . . 53
file.replace . . . 54
file.touch . . . 55
file.symlink . . . 56
pkg.installed . . . 57
service.running . . . 58
sysctl.present . . . 59
Pillar Access . . . 60
Grains Access . . . 61
user.present . . . 62
git.cloned . . . 63
git.latest . . . 64
Known Unknowns . . . 65
Porting Custom States . . . 65
Porting Custom Facts . . . 65
Appendix A: Salt Introduction . . . 66
Salt Is Agent Based . . . 66
Salt Also Uses YAML . . . 66
Salt Uses Jinja2 More Aggressively Than Ansible . . . 66
Salt Is More Holistic Than Ansible . . . 67
Salt Is More 'Batteries Included' Than Ansible . . . 67
Salt Also Uses YAML For Host Data . . . 67
Salt Uses 'grains' Instead Of ansible_facts . . . 68
Salt Uses GPG For Encrypting Data Instead of ansible-vault . . . 68
Salt Isn't Necessarily Sequentially . . . 68
Salt Runs Host-By-Host Instead of Role-By-Role . . . 68

The Concept of 'roles' Isn't Baked Into Salt 68
Appendix A(I): A ChatGPT Prompt 69

Copyright

Copyright © 2026 Jude Nagurney

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the author, except for brief quotations used in reviews or scholarly works.

All product names and trademarks are the property of their respective owners and are used for identification purposes only.

Second edition, 2026

Introduction

I'm writing this in the second half of 2025 soon after retiring with 35 years in software development. I used Salt extensively for the last 7 years, and before that I had been using Puppet for 7 years. I was often involved in tasks that eventually became known as DevOps, focusing on configuration management at smaller companies.

I became disillusioned with Salt after VMWare acquired SaltStack and then Broadcom acquired VMWare. Now in late 2025, Salt isn't getting much attention or support from its current owner. Many of the core Salt developers have left the company, and the cadence of bug fixes and releases seem to be slowing down. Also before retiring, I looked at the job market and found almost nobody is looking for people with Salt experience while at least some people were still looking for Ansible experience.

With that in mind, I decided to learn Ansible by porting the Salt states for my home lab over to Ansible. I kept notes along the way, and thought other developers might benefit if I published what I learned.

Please let me know ~~if~~ when you find errors in the text, so I can fix them in future editions. Any errors are my own fault.

If you want to support me more than paid for this book (if you paid for it at all), you can 'buy me some Ceylon tea' with the QR code below. If you're a software developer, working on a migration project covered in this book, and you have some extra unused training allowance dollars going to waste, maybe send some of them my way. This book should save you at least a day of work, so maybe send me half a day's pay per developer on you team my way. Those tea tariffs won't pay themselves.



Enjoy.

Audience

I wrote this book for three groups of readers.

The first group is Salt users who want to quickly get up to speed with Ansible because they're porting over their existing Salt states to Ansible tasks for whatever reason. Maybe they just don't want to be pigeon-holed into knowing one provisioning system. Maybe their company is forcing a migration to Ansible as one of its number one priorities this quarter. Maybe Salt's current owner is doing something new and innovative to drive away users.

The second group is users who don't know Salt and have inherited some Salt states that they need to port over to Ansible. If you aren't already familiar with Salt, check out Appendix A.

The third group has a size of one - myself. I'm writing this to prove to myself that I understand how I migrated my states well enough to document it according to the Feynman technique.

I assume all the groups have a good understanding of Salt, YAML, the Jinja2 templating system, and at least enough programming experience to understand Jinja2 variables, conditionals, and looping constructs. I'm also assuming readers have access to the Salt and Ansible documentation sites so they can look up more info on the many Salt and Ansible basics I'll be glossing over.

The purpose of the book is to show you a migration path, not to teach you Salt or Ansible from scratch.

I assume you're using git for version control, and you have some system in place for tracking issues and bugs.

Acknowledgements

The cover photo is from https://pxhere.com/en/photo/777522. According to the site it has a CC0 Public Domain license, but didn't include the name of the photographer. Thanks to whoever took the picture, and thanks for putting it in the public domain.

I used Obsidian 1.10.6 for editing the book's markup and generating the PDF, and Dia 0.97.3 for the diagrams.

Disclaimers

Software Versions During my migration work, Salt was at version 3007.10 and Ansible was at version 2.18.6. The content below may be obsolete if there are newer versions of these tools available by the time you're reading this. *Sic transit gloria mundi*[1]. #### Limitations

These features of the Salt ecosystem aren't covered in this book: - Salt reactors - Salt mines - Custom Salt facts - Custom Salt states - External pillar plugins other than salt.pillar.file_tree - Salt or Ansible for Windows (all my hosts are using Debian or Ubuntu) - SELinux-specific issues (my hosts are all in permissive mode) - Porting Salt-specific unit tests over to Ansible. - Scaling Ansible to large networks. I'm assuming the number of hosts in your network is probably less than a hundred.

Owners of this book should receive free updates whenever there are new releases that cover these topics.

AI Usage I didn't 'vibe code' my Ansible migration project, but I did use chatGPT a fair amount, especially for finding initial Salt->Ansible equivalents, and asking for explanations for ansible-playbook and ansible-lint error messages.

I'm a baseline human circa late 2025, with no neural implants. I have an analog artificial lens in my left eye after a cataract surgery. If I cover my right eye everything is clear, but if I cover other eye, everything looks like an old, sepia tone photo. That's a little freaky, but it shouldn't impact the book's content.

[1] "thus passes the glory of the world"

Release History

Version	Date	Notes
0.5	2026-01-15	(limited internal release)
1.0	2026-01-19	Initial public release, primarily to check if LeanPub displays PDF table of contents. (It doesn't)
2.0	2026-03-19	Vale suggested grammar and spelling fixes.Updated Configure SSH Access To The Hosts section

Migration Guiding Principles

Here are the principles I tried to adhere to while working on the migration. These hold for pretty much any migration project, and aren't really specific to Salt or Ansible.

Reduce External Dependencies

Early on I decided to try to migrate as many of the Salt states as possible using standard Ansible `ansible.builtin.*` roles as much as possible. I avoided picking community-built roles from the `galaxy.ansible.com` site since I didn't know Ansible well enough to know how to choose well-written and well-maintained roles.

I also didn't want to introduce unneeded dependencies on my Ansible roles if I didn't really want to have to maintain. I also wasn't sure how much of a hassle it would be to find out when new versions of the community-built roles were available.

In general, it's better to avoid dependencies as much as possible.

Check Data as Much as Possible

One of the big pain points in Salt was configuring the pillar data files correctly for new hosts. Salt had no native means of validating pillar data other than Jinja2 checks. Folks writing the pillar files often introduced indentation or spelling errors in the pillar files that Salt would not flag as errors.

Ansible's support for JSON Schema for data validating and the argument_specs.yml support for validating role arguments help flag these sorts of errors much sooner. I incorporated these checks into the migration process below, and they pointed out some typos and indentation issues during development.

I was also pleased to see that Ansible had a pretty nice linter called ansible-lint, which I incorporated into my workflow. I managed to avoid having to turn off any of the ansible-lint checks.

Avoid Refactoring While Migrating

Your current Salt states have probably been in use for some time and you've been testing them in production on every state.apply call.

With that in mind, I tried to limit refactoring the provisioning code as I migrated it, sticking with a one-for-one migration of Salt state to Ansible task as much as possible, so I didn't end up losing any of the hard-earned provisioning steps. The only documentation for the provisioning might be the existing Salt code.

That said, if I saw the current Salt states aren't "battening down the hatches" as tightly as they could be (say forgetting to set a mode on a file), I went ahead and added the missing provisioning (especially since ansible-lint would complain about things like missing file modes).

Major Salt/Ansible Differences

Here are the major differences I ran into while learning Ansible as a Salt user. This section assumes you already know Salt. If you don't, maybe skip over this section and check out Appendix A Salt Introduction instead.

Ansible Provisions Hosts Via SSH

Ansible provisions a host by ssh'ing into it and running commands. There isn't a dedicated Ansible 'minion' service that needs to be running on each host.

This means there needs to be an ssh route to each of the servers you want to provision with Ansible, possibly bouncing through bastion hosts to get to the final host.

Ansible Is YAML-First While Salt is Jinja2-First

Ansible restricts the use of Jinja2 compared with Salt. Ansible forces Jinja2 to be in the value side of Ansible key-value pairs.

Instead of skipping over states by wrapping the state in a Jinja2 conditional, in Ansible you add a 'when' attribute to a task.

Ansible doesn't support looping over tasks the way to can loop over states in Salt. Instead, Ansible adds kludgy looping attributes to its tasks.

Ansible can't assign variables with Jinja2 in the task files. Instead you can assign variables with the `ansible.builtin.set_fact` task or with a `vars` map added to a task. As a result of this, Ansible tasks don't import a Salt-style map file like map.jinja to assign variables.

See the Jinja2 Migration Guide section for more details on the `when`, `loop`, and `var` attributes.

Ansible Calls Function Calls With ansible.builtin.import_tasks Instead of Jinja2 Macros

The biggest loss I felt when moving to Ansible was that I could no longer define and call Jinja2 macros. You can work around this by using the Ansible `ansible.builtin.import_tasks` task, but that seems pretty clunky compared to calling a macro. See the Jinja2 Macros section below.

Main Ansible Source is Under tasks/main.yml Instead of Init.sls

The main source for a role is under the `roles/ROLENAME/tasks/main.yml` file instead of Salt's `init.sls` files. As mentioned in Ansible Is YAML-First While Salt is Jinja2-First, there is nothing in Ansible like Salt's imported map.jinja files because of Ansible's limited use of Jinja2.

Ansible Runs Tasks Sequentially

Ansible runs tasks in a playbook one right after the other in the order they appear in the YAML file. As a result, Ansible has nothing like the Salt ordering constraints such as `require`, `require_in`, `watch`, and `watch_in`, or `order`.

Instead Ansible tasks can have a `notify: NotificationName` attribute, and you need to define a listening task in the `roles/ROLENAME/handlers/main.yml` file. You also need to spell out if Ansible should restart or reload services when responding to a notification. See watch_in/watch in the Salt Prerequisites Migration Guide.

You can define a dependencies map in the `role/ROLENAME/meta/main.yml` file. This forces the main.yml tasks for each task mentioned in the dependencies map to run before the role's tasks.

Below is an example dependency map for the role `ROLENAME` that forces Ansible to run the tasks from `roles/fail2ban/tasks/main.yml` and `roles/postgresql/tasks/main.yml` before the `roles/ROLENAME/tasks/main.yml` tasks.

```
---

dependencies:
- role: fail2ban
- role: postgresql
```

Ansible Names Are Descriptive Only

Salt commonly uses the name of a state as the state's target, such as the name of a service that should start, or the name of the file that should have its contents updated.

Ansible uses the name of a task for documentation purposes only, and the task uses an attribute like 'name' or 'src' to specify the target.

A happy side-effect is that with Ansible you don't have to ensure the task names are unique across all the tasks the way to have ensure uniqueness across all the Salt states. You don't have to add a loop index value to a Salt state title in a loop to ensure it has a unique name.

Ansible Uses host_vars and group_var Files Instead of pillar files

Salt typically stores its host-specific pillar data under the `/srv/pillar` directory on the saltmaster. Ansible stores the same sort of data under the `inventory/host_vars/` directory. Ansible also stores group-specific data under the `inventory/group_vars` directory. These are YAML-based file like the pillar files. Most of the time you should be able to move the pillar files to the `host_vars` directory without changing anything. One exception is when the Salt files include GPG encrypted blocks, you'll need to re-encrypt the data with the `ansible-vault` command.

Ansible surfaces the data in the `host_vars` and `group_vars` files as global variables. If you have a `inventory/host_vars/some_host.yml` file with the contents below, Ansible surfaces a global variable `some_role.some_key` with value `some_value` available when running playbooks against the `some_host` host. You can also refer to the variables as arrays, so `some_role['another_key']` has the value `another_value`. Ansible has no match for the Salt `salt['pillar.get']('some_key')` function call. Also note Ansible uses dotted notation instead of ':' separated values for nested data.

```
some_role:
some_key: some_value
anotherkey: another_value
```

Also see the Pillar access section in the Specific Salt States Migration Guide.

Ansible Uses A Global ansible_facts Map Instead Of Grains

Ansible has a global ansible_facts map variable that holds the same sort of host-specific values as Salt's `salt['grains.get']('grain_name')` function surfaces.

See the Grains access section in the Migration Pattern Handbook.

Ansible Uses a lookup() Function Instead of external pillar

Ansible doesn't have a direct match to Salt's external pillar concept, but you can simulate it with Ansible's powerful lookup() function.

Also see the file.managed (with contents_pillar) in the Migration Pattern Handbook.