Choosing an orchestration tool: Ansible and Salt

ANSIBLE

SALTSTACK

Vincent Boon
Opengear

Introduction

What is Orchestration, and how is it different from Automation?

- Automation involves codifying tasks like configuring a webserver, or upgrading a software package, so that they can be performed quickly, and without error.
- Orchestration involves codifying processes like deploying an application, where many tasks may need to take place on disparate devices.
- Traditionally been part of the Software and Ops world, but more and more applicable to network devices.

Introduction

opengear SMART SOLUTIONS FOR RESILIENT NETWORKS

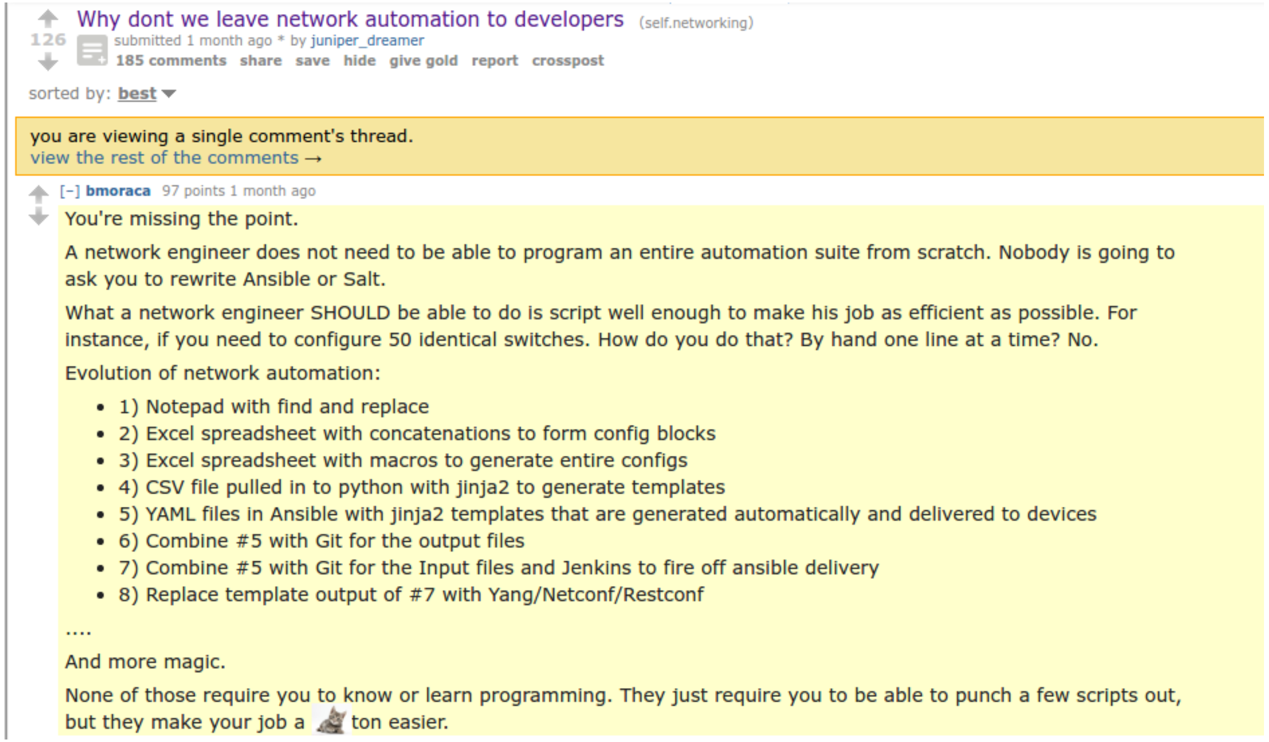
This talk is mostly going to focus on the automation component of Orchestration.

The tools discussed are capable of both; my aim today is to give you enough of an introduction that you can set aside some time to spin up a VM and try them out.

Each of the tools have their own jargon, but once you get past that, you can understand how they work, and make a choice based on your requirements.

Introduction

opengear
SMART SOLUTIONS FOR RESILIENT NETWORKS



Why dont we leave network automation to developers (self.networking)

126

submitted 1 month ago * by juniper_dreamer

185 comments share save hide give gold report crosspost

sorted by: best

you are viewing a single comment's thread.
view the rest of the comments →

[–] bmoraca 97 points 1 month ago

You're missing the point.

A network engineer does not need to be able to program an entire automation suite from scratch. Nobody is going to ask you to rewrite Ansible or Salt.

What a network engineer SHOULD be able to do is script well enough to make his job as efficient as possible. For instance, if you need to configure 50 identical switches. How do you do that? By hand one line at a time? No.

Evolution of network automation:

- 1) Notepad with find and replace
- 2) Excel spreadsheet with concatenations to form config blocks
- 3) Excel spreadsheet with macros to generate entire configs
- 4) CSV file pulled in to python with jinja2 to generate templates
- 5) YAML files in Ansible with jinja2 templates that are generated automatically and delivered to devices
- 6) Combine #5 with Git for the output files
- 7) Combine #5 with Git for the Input files and Jenkins to fire off ansible delivery
- 8) Replace template output of #7 with Yang/Netconf/Restconf

....

And more magic.

None of those require you to know or learn programming. They just require you to be able to punch a few scripts out, but they make your job a ton easier.

Changes in Network Orchestration

In 2018, Ansible gets a lot of airtime for network orchestration, more than Puppet and Chef.

Salt is also heavily promoted by companies like Cloudflare

This talk will focus on Ansible and Salt:

- What does their architecture look like.
- What comes for free vs what you pay for.
- How they use NAPALM.

What is NAPALM?

Network vendors love automation/orchestration tools

- They build a module for configuring their devices for Puppet/Chef/Ansible/Salt
- Write a whole bunch of whitepapers demonstrating its use
- Customer writes a whole bunch of configuration using the module
- Customer goes to evaluate another vendor
  - The module is different :(
- Enter NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support).

What is NAPALM?

NAPALM is a Python library that can provide a unified API to a number of different router vendors.

The napalm-ansible module provides a way to call the NAPALM API inside Ansible Playbooks

NAPALM itself is integrated inside Salt from version 2016.11.0 (Carbon) (driven by Cloudflare)

What is NAPALM?

opengear
SMART SOLUTIONS FOR RESILIENT NETWORKS

Supported Network Operating Systems are:

- Arista EOS
- Cisco IOS, IOS-XR, NX-OS
- Fortinet FortiOS
- IBM
- Juniper JunOS
- Mikrotik RouterOS
- Palo Alto NOS
- Pluribus NOS
- VyOS

What is NAPALM?

It isn’t magic:

- In general, you’re still going to be writing configuration templates for your different vendors.
- Template then gets merged into running config, and can be checked for diffs.
- Power comes from the consistent “getters” API.
- Allows “verifiers” to be written to check the bits of config you care about
- Work continues on generalized configuration templates for true cross-platform configuration, as well as Netconf and YANG support.

Ansible

Developed by Redhat, written in Python

Billed as an “masterless and agentless” automation/orchestration tool

- Uses SSH as transport, authentication is generally done using SSH keys
- Ships Python modules to the target device, which are then executed.
- When being used with ansible-napalm, transport will vary based on the device being managed.
- Can log to a variety of log services
- Integrates with Ansible Tower to provide more enterprise features
  - Acts a central server for Ansible
  - WebUI/REST API/Dashboard

Ansible

Ansible uses the concept of a playbook to define a series of steps (or “plays”) that map a series of execution steps (or tasks) to a group of hosts.

These playbooks are written in YAML.

Each task (which calls an Ansible module) should be idempotent – running it many times will give the same result, and the task definition should contain enough detail to allow it to also be used to check that the task has been carried out successfully.

Handlers can also be defined for tasks that may need to be called only once after a number of operations. For example, if a number of tasks are concerned with changing webserver configurations, then the webserver only needs to be restarted once at the end.

Ansible

In Ansible, hosts are defined inside an inventory. The inventory is often a static file, but it can be dynamic when that makes sense (for managing Docker containers, or VMs).

The inventory allows administrators to groups hosts based on their role (webservers, load-balancers, border-routers etc), as well as associating variables with individual or groups of host. These variables can be referenced inside the Playbooks to customize the particular task for the host.

Variables can also be retrieved at application time from the hosts. These variables are called “Facts”

Ansible – Free vs $$

Out of the box, Ansible is designed around the user running Ansible playbooks to push and verify configuration.

- Very basic automation of playbook scheduling is included (ansible-pull + cron)
- For more, this is where Ansible Tower comes in
  - Free Tier, then 2 ($$) Tiers that come with more features and support

Salt

Developed by SaltStack, written in Python

The architecture is generally based around a central server (salt-master), with agents called salt-minions running on the devices under management. It can be run in a masterless mode, but this is not how its normally used.

- For devices that can’t run an agent, a “proxy-minion” process can be run on a server, which then communicates with the device using its native protocols.
- Communications between the server and the minions uses the ZeroMQ message bus by default.
- All operations are scheduled and logged by the central server. Minions receive commands from the server, and run these asynchronously. Results are push back to the server via the message bus.

Salt

Salt has many types of modules, but for our example, we'll examine two: execution modules, and state modules.

Execution modules are used to perform actions

State modules use execution modules to make a device conform to the desired state. This allows the user to define a desired state for a particular segment of configuration in a declarative fashion.

Execution modules are generally run as once-off commands, while state modules are more like Ansible Playbooks. Execution modules do however return their results as JSON data, so their results can be parsed and used in other workflows

Salt Architecture

The state module uses state definitions, which are written as SLS (SaLt State) files. They can be written in many languages, but the default is YAML (like an Ansible playbook)

These are stored centrally on the master, in a file server that is referred to as the file_roots, while other data (such as variables) are stored in a data store known as the Salt Pillar.

The salt-minions retrieve these state file definitions, and other items (like variable definitions) from the Salt Pillar and the file server over the central message bus.

Salt Architecture

Like Ansible, variables can be defined locally in the state file, and can also be retrieved from the devices under management. Salt calls these variables “grains”. However, best practice with Salt is to keep variables separate from state information, by storing them in the Salt Pillar. This keeps logic (state) separate from data (variables).

Rather that specifying the devices that a state or action applies to in the state definition, Salt allows that to be specified during application time, using static data stored inside the pillar, as well as grains that are retrieved from the managed devices.

Salt

This description barely brushes the surface of what Salt can do.

It is more complex than Ansible:

- More moving parts, and options compared to Ansible Playbooks
- Steeper learning curve
- Allows more complex workflows than out-of-the-box Ansible

Salt – Free vs $$

SaltStack does have an Enterprise version that adds a number of extra features, but the OSS release allows a more sophisticated Automation and Orchestration setup than Ansible.

However, this comes at the cost of the extra effort for setup.

Conclusion

I don’t know your requirements, you do.

There are good communities around both products.

You don’t need to do everything right now

Watch some vidoes and try out some examples!

Ansible: https://www.youtube.com/watch?v=9TFlc-ekRPk

Salt: https://www.youtube.com/watch?v=AqBk5fM7qZ0

Further Information

opengear
SMART SOLUTIONS FOR RESILIENT NETWORKS

Ansible + Napalm:

https://pynet.twb-tech.com/blog/automation/napalm-ios.html

Salt + Napalm:

https://mirceaulinic.net/2016-11-17-network-orchestration-with-salt-and-napalm/