Choosing an orchestration tool: Ansible and Salt

ANSIBLE

SALTSTACK

Ken Wilson
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.

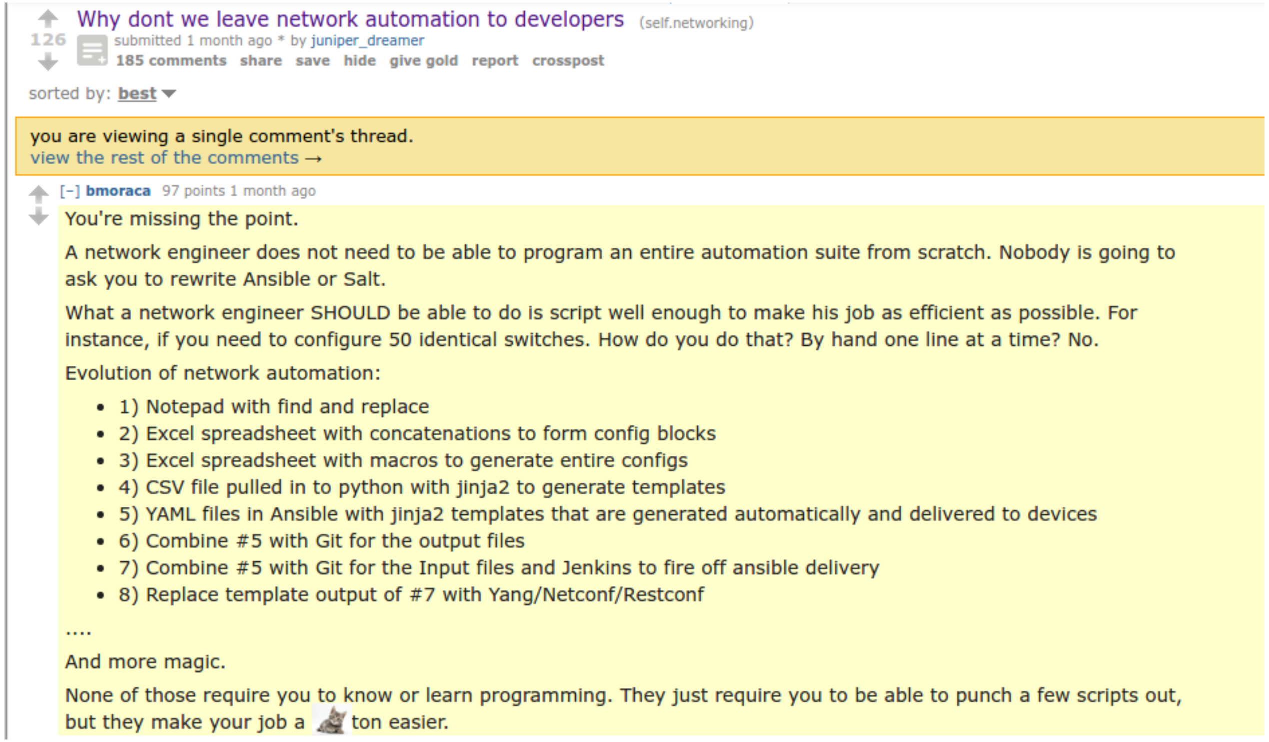
Introduction

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 has its own jargon, but at their heart, they work the same way, and they don’t require you to be a developer to use them.

Introduction



126

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

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.

2016-2017

AusNOG 2016 – Central Orchestration of Network Infrastructure – NetOps meets DevOps

- Presentation covered: Puppet, Chef, and a little bit of Ansible

Comments from attendees

- Attendee from a small regional ISP: “Wow, I’ve seen these tools used for server maintenance, but not for network gear. This would really simplify my life!”
- Attendee from G**g**: “Doesn’t everyone do this?”
- Another Attendee: “What about Salt! Everyone forgets Salt”

Changes in Network Orchestration

In 2017, 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.
- Configuration examples for Network Orchestration/Automation using NAPALM
  - 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?

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

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 - Example

Example – Applying templated configuration to two Cisco Devices

Requirements:

- Devices must have SSH and SCP enabled

Source is available at:

https://github.com/kenwilson/central-orchestration-ausnog2017/

Config Apply Playbook and Template

```
---

- name: Apply basic config
  hosts: cisco-cpe

  tasks:
    - name: Generate local configuration for hosts
      local_action: template dest={{ playbook_dir }}/gen-config/initial-
{{ hostname }}.conf src={{ playbook_dir }}/source/initial.j2

    - name: Gen .diff file (apply change)
      napalm_install_config:
            hostname: "{{ host }}"
            username: "{{ username }}"
            password: "{{ password }}"
            dev_os: "{{ dev_os }}"
            config_file: gen-config/initial-{{ hostname }}.conf
            commit_changes: True
            replace_config: True
            diff_file: initial-{{ hostname }}.diff
            optional_args: {'auto_rollback_on_error': False}
```

```
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname {{ hostname }}
!
boot-start-marker
boot-end-marker
!
enable secret 5 $1$WBBw$3WDO6jJHSwFtuw4Wk38Zv/
enable password notdefault
!
no aaa new-model
no ip routing
!
!
no ip cef
!
!
!
!
!
```

Get Facts Playbook and Inventory

```yaml
---

- name: Get Device Facts
  hosts: cisco-cpe

  tasks:
    - name: Get Facts
      napalm_get_facts:
            hostname: "{{ host }}"
            username: "{{ username }}"
            password: "{{ password }}"
            dev_os: "{{ dev_os }}"
            filter: 'facts,interfaces,interfaces_ip,mac_address_table'
      register: results

    - name: print data
      debug: var=result
```

```ini
# Cisco CPE devices
[cisco-cpe]
r1 host=192.168.82.76 hostname=r1-cpe.opengear.com
r2 host=192.168.82.77 hostname=r2-cpe.opengear.com

# Default variables for Cisco devices managed via NAPALM
[cisco-cpe:vars]
username=admin
password=default
dev_os=ios
ansible_connection=loca
```

Run Output

```
kenw@kenw-srv:~/src/central-orchestration-ausnog2017/ansible-napalm-ios$ ssh
admin@192.168.82.77
Password:

r2-cpe.opengear.com#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
r2-cpe.opengear.com(config)#hostname r2-cpe-modified.opengear.com
% Hostname "R2-CPE-MODIFIED.            " is not a legal LAT node name, Using
"CISCO_CC0008"
r2-cpe-modified.open(config)#end
r2-cpe-modified.opengear.com#exit
Connection to 192.168.82.77 closed.
kenw@kenw-srv:~/src/central-orchestration-ausnog2017/ansible-napalm-ios$ ansible-
playbook config.play -i inventory

PLAY [Apply basic config]
*************************************************************************************************

TASK [Gathering Facts]
****************************************************************************************************
**
ok: [r1]
ok: [r2]

TASK [Generate local configuration for hosts]
******************************************************************************

ok: [r1 -> localhost]
ok: [r2 -> localhost]
```

```
TASK [Gen .diff file (apply change)]
**************************************************************************************
ok: [r1]
changed: [r2]

PLAY RECAP
****************************************************************************************************
**************
r1                    : ok=3    changed=0    unreachable=0    failed=0
r2                    : ok=3    changed=1    unreachable=0    failed=0

kenw@kenw-srv:~/src/central-orchestration-ausnog2017/ansible-napalm-ios$ cat initial-
r2-cpe.opengear.com.diff
+hostname r2-cpe.opengear.com
-hostname r2-cpe-modified.opengear.com
```

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
  - Schedule Playbook runs
  - Real time job status updates
  - Job logging
  - Lots more

Salt

Developed by SaltStack, written in Python

The architecture is based around a central server (salt-master), with agents called salt-minions running on the devices under management

- 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 ZeroMQ by default.
- All operations are scheduled and logged by the central server

Salt

Salt has two types of modules: execution modules, and state modules.

Execution modules are used to perform actions

State modules use executions modules to make a device conform to the desired state.

Execution modules are generally run as once-off commands, while state modules are more like Ansible Playbooks

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 storage facility called the Salt Pillar.

The salt-minions retrieve these state file definitions, and other items (like variable definitions) from the Salt Pillar over their 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”. Salt can also store variables centrally in the Pillar.

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:

- Requires a central server, along with a fair amount of configuration
- Proxy minions require a process each, as well as configuration stored on the central server
- Higher learning curve

Salt - Example

Example – Applying templated configuration to two Cisco Devices

Requirements:

- Devices must have SSH and SCP enabled

Source is available at:

https://github.com/kenwilson/central-orchestration-ausnog2017/

Salt Proxy Minion Config

```
proxy:
  proxytype: napalm
  driver: ios
  host: 192.168.82.76
  username: admin
  password: default
  optional_args:
    auto_rollback_on_error: False

#R1 specific variables
hostname: r1.cpe.opengear.com
```

Salt State File and Template

```
proxy:
  proxytype: napalm
  driver: ios
  host: 192.168.82.76
  username: admin
  password: default
  optional_args:
    auto_rollback_on_error: False

#R1 specific variables
hostname: r1.cpe.opengear.com
```

```
full_config:
  netconfig.managed:
    - template_name: salt://router-config.j2
    - replace: True
```

Salt State File and Template

```
full_config:
  netconfig.managed:
    - template_name: salt://router-config.j2
    - replace: True
```

```
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname {{ pillar.hostname }}
!
boot-start-marker
boot-end-marker
!
enable secret 5
$1$WBBw$3WDO6jJHSwFtuw4Wk38Zv/
enable password notdefault
!
no aaa new-model
no ip routing
!
!
no ip cef
!
!
```

Salt Output

```
kenw@kenw-srv:~/src/central-orchestration-ausnog2017/salt-napalm-ios$
sudo salt r1 state.sls config
r1:
----------
          ID: full_config
    Function: netconfig.managed
      Result: True
     Comment: Configuration changed!
     Started: 17:33:50.636394
    Duration: 33202.776 ms
     Changes:
              ----------
              diff:
                  +hostname r1.cpe.opengear.com
                  -hostname r1-modified.cpe.opengear.com

Summary for r1
------------
Succeeded: 1 (changed=1)
Failed:    0
------------
Total states run:     1
Total run time:  33.203 s

kenw@kenw-srv:~/src/central-orchestration-ausnog2017/salt-napalm-ios$
sudo salt -G 'os:ios' net.load_template salt://router config.j2 replace=True
```

```
r2:
    ----------
    already_configured:
        False
    comment:
    diff:
        +hostname r2.cpe.opengear.com
        -hostname r2-cpe.opengear.com
    loaded_config:
    result:
        True
r1:
    ----------
    already_configured:
        False
    comment:
    diff:
        +hostname r1.cpe.opengear.com
        -hostname r1-cpe.opengear.com
    loaded_config:
    result:
        True
```

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, even a setup like the basic examples I’ve shown is a start

Try them out!

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/