Saltstack - Le sel de la pile

Matthieu Herrb

Josy - 16 septembre 2022

https://homepages.laas.fr/matthieu/talks/saltstack-josy.pdf

Licence

Ce document est sous licence

Creative Commons Paternité - Partage dans les mêmes conditions 4.0 International.

Le texte complet de cette licence est disponible à l'adresse :

http://creativecommons.org/licenses/by-sa/4.0/

Agenda

1 Introduction

2 Présentation de SaltStack

3 Salt au LAAS

4 Conclusion

Agenda

1 Introduction

2 Présentation de SaltStack

3 Salt au LAAS

4 Conclusion

À propos de l'auteur

Matthieu Herrb

- Ingénieur de Recherche au LAAS du CNRS à Toulouse depuis plus de 30 ans
- Systèmes Unix, BSD, Linux
- CSSI du LAAS et membre de la coordiation régionale SSI Occitanie Ouest (aka DR14)
- Membre du CP du réseau Capitoul
- Contributeur au logiciel libre (X.Org, OpenBSD)
- Membre du CA du FAI associatif Tetaneutral.net

À propos de cette présentation

Cette préstation essaye de :

1. présenter les fonctionnalités (et les limites) de Saltstack,
2. un retour sur comment il est utilisé dans mon laboratoire.

Ce n'est pas :

- une présentation marketing de Saltstack,
- un test comparatif de Saltstack vs X ou Y,
- un cours sur Saltstack,
- une recommendation sur le choix d'un outil ou d'une méthode.

Agenda

1 Introduction

2 Présentation de SaltStack

3 Salt au LAAS

4 Conclusion

Généralités

SaltStack [https://saltproject.io/] c'est :

- un système de gestion de configuration
  - maintient des *états* sur des *nœuds*
- un système d'exécution distribuée
  - exécute des commandes et récupère de l'information sur des *nœuds*

Marketing :

- simplicité
- passage à l'échelle de miliers de nœuds
- protocole standard sécurisé et chiffré
- performance

Racheté par VMware en 2021

Salt en quelques mots

- Client/Serveur
- Description en YAML
- Templates via Jinja2
- Extensions en python

Salt - principe

- Les nœuds sont des *minions* (*sous-fifres*)
- les *grains* décrivent le minion (en YAML):
  - configuration système
  - grains définis par l'admin
- L'état désiré est décrit (en YAML) à l'aide de *formules*
- Les formules utilisent des *modules*
  - écrits en python
  - abstraction des spécificités des systèmes
- Le *pillier* permet de fournir des paramètres aux formules et aux états (en YAML).
- Les *reacteurs* définissent des actions pour réagir aux événements liés à l'exécution. (en YAML + python)

Ciblage

En mode push : spécification des minions (machines cibles)

- `*` : tous les client connus
- par noms
- par groupes de noms (`nodegroups`)
- par les grains
- composé : expression sur les éléments ci-dessus

Quelques exemples

Exécution distribuée

```
master# salt '*' cmd.run 'reboot'
master# salt '*' system.reboot
```

Gestion de configuration

```
master# salt '*' pkg.upgrade sudo
master# salt -G applis:robotique state.apply install-ros
```

Mode pull

```
minion# salt-call state.apply test=True
```

Exemple de formule : serveur redis

```
Redis package:
  pkg.installed:
    - name: redis-server

/etc/redis/redis.conf:
  file.line:
    - mode: replace
    - content: requirepass {{ pillar['redis_password'] }}
    - match: 'requirepass '

redis:
  service.running:
    - enable: True
    - watch:
      - file: /etc/redis/redis.conf
```

Exemple de formule : serveur redis (suite)

- le YAML du transparent précédent dans un fichier, soit :
  - `redis-server.sls`
  - `redis-server/init.sls`
- → déclare un état *redis-server*
- Applique cet état:
  - `salt` *`serveur`* `state.apply redis-server`
  - en listant cet état dans l'état global qui s'appliquera à *serveur* dans `top.sls`

```
base:
  'laas:redis-server':
    - match: grain
    - redis-server
```

Sécurité

Authentification Mutuelle des minions et du maître

- bi-clés
- approbation préalable des minions par le maître
- chiffrement des échanges
- ports dédiés : 4505 et 4506

Notion d'ACL pour les non `root`

- permet de contrôler les droits d'exécution des états et l'accès aux ressources
- permet d'utiliser une authentification externe (LDAP)

Mécanisme d'accès à des données chiffrées (PGP)

- pour les données dans le pillier
- déchiffrées par clé privée sur le minion

Agenda

1 Introduction

2 Présentation de SaltStack

3 Salt au LAAS

4 Conclusion

Salt au LAAS

- 2013 : Réunion Capitoul gestion de configuration: Ansible / Puppet
- 2014 : Evaluation de puppet
- 2015 : Comment mettre son grain de sel partout, Aurélien Minet, JRES
- 2016 : Choix de saltstack
- présenté à Capitoul par Julien Libourel en avril 2018
- depuis, utilisé pour déploiement et maintien à jour du parc Linux (CentOS/Rocky et Ubuntu)
  - des postes de travail Linux (environ 360)
  - les VM serveur Linux (125)

Approche Bottom-Up:

- décrire des états individuels d'abord,
- puis factoriser.

Démarche générale

Appliquée sur les serveurs comme pour les postes de travail :

- installation minimale avec Salt
- déploiement et maintient de la config

postes de travail : grains génériques pour déterminer un profil d'installation

serveurs : ciblage par nom et description de l'état individuel du serveur

Postes de travail

- config mirroir LAAS des dépots de paquets
- installation des paquets
- configuration LDAP + Kerberos + client NFS / automount
- paramétrage IPv6
- déploiement clés SSH sysadmin pour accès admin
- configuration syslog centralisé
- configuration client OCSinventory
- install et configuration LiveNavigator pour les portables
- ...

Serveurs

- Un état par serveur (ciblé par le nom)
- Niveaux d'automatisation variables
- Migration incrémentale de la configuration vers Salt
- Configuration réseau statique

En pratique

- un master (dans une VM CentOS)
- essentiellement mode pull (via cron sur les minions)
- un réacteur pour envoi de rapports par mail
- l'ensemble des formules et du pillier sous git (sauf secrets)
- mais (pour l'instant) :
  - pas vraiment d'environnement de développement séparé
  - ni déploiement automatique via CI.

Exemples

- Config ssh
- Déploiement d'OpenSMTPd

Quelques difficultés

- Existant très hétérogène
  travail en parallèle pour standardiser les configurations
- Plus d'une façon de faire chaque chose
- Qualité / couverture inégales des formules / states existants
  $\rightarrow$ pas mal de développements/ adaptations maison.
- Contribuer des formules / states / modules génériques est dur
- Relativement gourmand en ressources
- Fonctionne moyennement avec double pile IPv4 / IPv6
- Support des nouvelles versions de système (Ubuntu 22.04, Rocky Linux 9) tarde à venir

Agenda

1 Introduction

2 Présentation de SaltStack

3 Salt au LAAS

4 Conclusion

Conclusion

- L'harmonisation et l'automatisation de la gestion des configuration sont vitales.
- Saltstack répond à nos besoins.
- L'exécution distribuée permet des actions rapides.
- Nécessite de la pratique pour en tirer tous les bénéfices.
- Investir dans le développement de formules et de modules maison.
- Ne pas vouloir en faire trop d'un coup.