LIGNIERE
Timothée
BTS2 SIO
Date de création
16/07/23
Date de maj
16/04/24

Saltstack

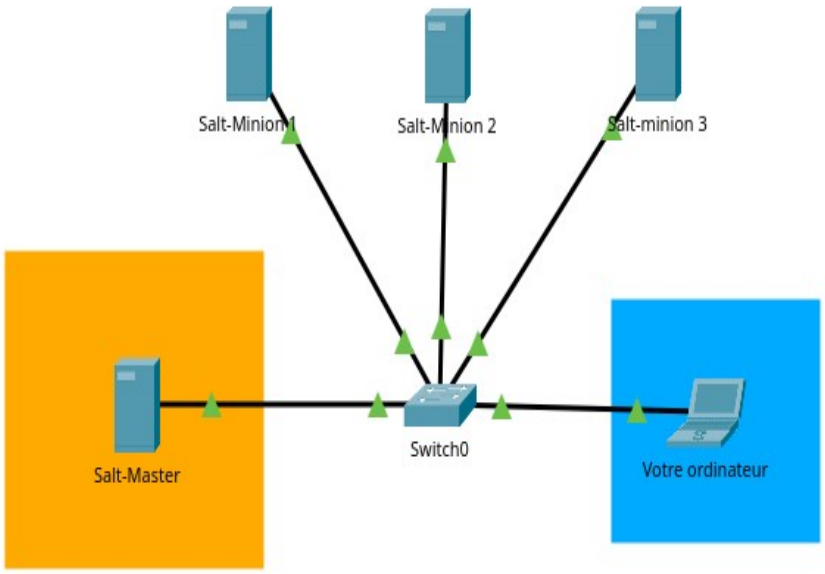
Sommaire

1 - Problématique

Il peut être fastidieux d'installer plusieurs fois le même logiciel sur plusieurs machines, surtout dans des cas d'infrastructure où l'on peut déjà compter 10 instances.

Salt (Saltstack) permet de résoudre cette problématique dans une interaction MASTER – SLAVES entre plusieurs machines. Le Master contient toutes les consignes d'installation ou de configuration qui sont ensuite exécutés par les Slaves.

2 - Schéma logique de la solution



3 - Notice d'installation

L'installation suivante s'est déroulée sur **Ubuntu 22, en cas de problème d'installation avec votre version de Linux, se référer à https://docs.saltproject.io/salt/install-guide/en/latest/topics/install-by-operating-system/index.html**

Afin de savoir plus facilement sur quelle machine vous êtes lors de l'installation, je vous invite à modifier le nom de vos machines :

```
nano /etc/hostname
```

Le nom affiché sera changé au prochain redémarrage de la machine

3.1 – Mise en place du Salt Master (MASTER)

Le Master peut être considéré comme un poste admin vis-à-vis de son utilité dans l'infrastructure de salt, faites donc en sorte qu'il soit protégé au préalable, les minions (Slaves) auront uniquement besoin d'accéder aux **ports 4505 et 4506** du Master.

Installer salt-master avec la commande suivante :

```
apt update
apt install salt-master
```

Puis modifier le fichier de configuration pour le réglage basique pour qu'il puisse fonctionner :

```
nano /etc/salt/master
```

Modifier les lignes en dessous de « File Server Settings » à la ligne 656 :

```
#####      File Server settings      #####
######################################
# Salt runs a lightweight file server written in zeromq to deliver files to
# minions. This file server is built into the master daemon and does not
# require a dedicated port.

# The file server works on environments passed to the master, each environment
# can have multiple root directories, the subdirectories in the multiple file
# roots cannot match, otherwise the downloaded files will not be able to be
# reliably ensured. A base environment is required to house the top file.
# Example:
# file_roots:
#   base:
#     - /srv/salt/
#   dev:
#     - /srv/salt/dev/services
#     - /srv/salt/dev/states
#   prod:
#     - /srv/salt/prod/services
#     - /srv/salt/prod/states
#
file_roots:
  base:
    - /srv/salt
#
```

Modifier aussi les lignes en dessous de « Pillar settings » à la ligne 842 :

```
#####         Pillar settings        #####
######################################
# Salt Pillars allow for the building of global data that can be made selectively
# available to different minions based on minion grain filtering. The Salt
# Pillar is laid out in the same fashion as the file server, with environments,
# a top file and sls files. However, pillar data does not need to be in the
# highstate format, and is generally just key/value pairs.
pillar_roots:
  base:
    - /srv/pillar
#
#ext_pillar:
#  - hiera: /etc/hiera.yaml
#  - cmd_yaml: cat /etc/salt/yaml
```

Vous pouvez désormais allumer salt-master et activé son service :

```
systemctl enable salt-master
systemctl start salt-master
```

Créez les fichiers que nous avons activés dans le /etc/salt/master, ainsi qu'un fichier qui vous servira plus tard lors de l'utilisation :

```
mkdir /srv/salt /srv/pillar
touch /srv/salt/top.sls /srv/pillar/top.slsl
```

Tout est désormais en ordre pour votre salt-master, nous allons désormais passer aux salt-minions.

3.2 – Mise en place d’un Salt Minion (SLAVE)

Pour commencer la mise en place, installer salt-minion :

```
apt install salt-minion
```

Dès la mise en route du salt-minion, il cherchera à se connecter à un salt-master en le cherchant via un serveur DNS par l’adresse « salt ». Vous pouvez configurer un DNS qui renverra en effet au salt-master, mais nous pouvons aussi indiquer au salt-minion l’adresse exacte qu’il doit chercher :

```
nano /etc/salt/minion
```

Puis modifier la ligne 16 pour qu’elle ressemble à ceci :

```
# Set the location of the salt master server. If the master server cannot be
# resolved, then the minion will fail to start.
master: ADRESSE DE VOTRE SALT MASTER
```

Je vous recommande de modifier aussi l’ID de chacun de vos minions pour pouvoir les distinguer sur le master :

```
nano /etc/salt/minion_id
```

Vous pouvez désormais l’activer, elle cherchera ensuite à se connecter au salt-master, nous allons donc passer à la connexion des deux machines et vérification du bon fonctionnement après cela :

```
systemctl enable salt-minion
systemctl start salt-minion
```

3.3 – Connexion d’un minion au master

Je n’utiliserais qu’un seul minion pour cette étape, cependant, elle peut tout à fait être réalisée avec plusieurs minions en même temps.

Aucun port n’a besoin d’être ouvert sur les minions pour le fonctionnement de salt . Cependant, le master a besoin d’avoir des ports ouverts pour chacun de ses minions, **ces ports sont 4505 et 4506**

Sur le minion, vérifier s'il se connecte sans problème au master :

```
systemctl status salt-minion
```

Dans le cas où il n’y arrive pas, vous devriez avoir un retour similaire à celui-ci :

```
● salt-minion.service - The Salt Minion
     Loaded: loaded (/lib/systemd/system/salt-minion.service; enabled; vendor preset: e>
     Active: active (running) since Tue 2023-08-01 16:03:03 UTC; 5h 43min ago
       Docs: man:salt-minion(1)
             file:///usr/share/doc/salt/html/contents.html
             https://docs.saltproject.io/en/latest/contents.html
   Main PID: 636 (salt-minion)
      Tasks: 8 (limit: 4662)
     Memory: 81.3M
        CPU: 3.825s
     CGroup: /system.slice/salt-minion.service
             ├─636 /usr/bin/python3 /usr/bin/salt-minion
             ├─819 /usr/bin/python3 /usr/bin/salt-minion
             └─828 /usr/bin/python3 /usr/bin/salt-minion

Aug 01 21:38:44 salt-minion salt-minion[819]: [ERROR   ] Error while bringing up minion>
Aug 01 21:39:36 salt-minion salt-minion[819]: [ERROR   ] Error while bringing up minion>
Aug 01 21:40:28 salt-minion salt-minion[819]: [ERROR   ] Error while bringing up minion>
Aug 01 21:41:21 salt-minion salt-minion[819]: [ERROR   ] Error while bringing up minion>
Aug 01 21:42:13 salt-minion salt-minion[819]: [ERROR   ] Error while bringing up minion>
Aug 01 21:43:05 salt-minion salt-minion[819]: [ERROR   ] Error while bringing up minion>
Aug 01 21:43:57 salt-minion salt-minion[819]: [ERROR   ] Error while bringing up minion>
Aug 01 21:44:49 salt-minion salt-minion[819]: [ERROR   ] Error while bringing up minion>
Aug 01 21:45:41 salt-minion salt-minion[819]: [ERROR   ] Error while bringing up minion>
Aug 01 21:46:33 salt-minion salt-minion[819]: [ERROR   ] Error while bringing up minion>
```

Les deux erreurs possibles face à ce retour sont soit :

- L’adresse du salt-master qui n’a pas été correctement configuré au niveau du minion (Dans ce cas là revoyez L’installation de votre minion)
- Un pare-feu bloque la connexion minion → master

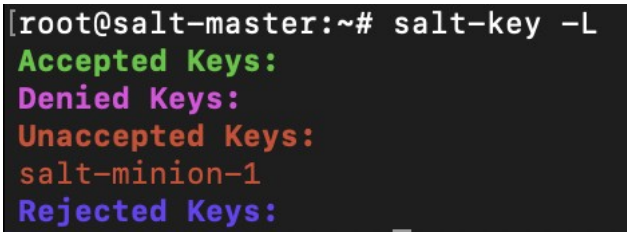
Lorsque vous penserez avoir réglé le problème, utiliser la commande suivante :

```
systemctl restart salt-minion
```

Sur le master, utiliser la commande suivante pour vérifier si le master reçoit bien les demandes de connexion du minion :

```
salt-key -L
```

Si c'est le cas, vous devriez avoir le retour suivant :



On peut donc voir les Clés acceptées, Refusés, Non-Acceptés et les Rejetés.

La commande suivante permet d'accepter toutes les clés dans « Unaccepted Keys » :

```
salt-key -A
```

Une fois ceci fait, tout est en place pour l'utilisation de SaltStack du côté des configurations, ceci dit, il y a encore beaucoup à faire selon vos besoins.

4 - Notice d’utilisation

4.1 – Les bases de SaltStack

SaltStack est un logiciel qui vous demandera d’aller très souvent sur son wiki afin d’obtenir le résultat que vous désirez, et certains termes peuvent être flous lors de l’apprentissage de cet outil ; voici donc quelques définitions basiques des éléments les plus utilisés :

Salt-Master : Chef d’orchestre des Salt-minions, il envoie ses consignes par des **States** et envoie des données spécifiques par ses **Pillars.**

Salt-Minion : Esclave du Salt-Master, il suit les consignes qu’on lui donne selon son ID, il envoie des **Grains** en cas de conditions particulières pour le Salt-Master

States : Consignes de gestion envoyé aux Salt-Minion, ils gèrent la configuration de chaque minion, ils peuvent donc servir de référence pour comprendre la disposition des fichiers dans une machine (Si ces fichiers sont gérés par un State).

Pillar : Données envoyées aux minions, c’est en général dans ces fichiers que l’on va positionner les mots de passes pour contrôler quelles machines y a accès.

Grains : Données très spécifiques liés à chaque minion, elles ne sont pas sur le master, ils contiennent par exemple la version de l’OS, l’architecture du CPU, les paramètres du kernel et tout simplement le nom d’hôte.

4.2 – Première utilisation

Nous allons faire un state très basique afin de comprendre le fonctionnement, tout se passe donc sur le salt-master;

Dans notre cas, nous allons installer le package « tree » sur un minion, ce package permet d'avoir une commande similaire à dir, mais plus complète, car elle montre les sous-dossiers et fichiers de ces dossiers.

Dirigez-vous sur le dossier contenant vos states puis créé un fichier en format sls, je vais personnellement l'appeler « tools » car je compte y positionner toutes les consignes pour des outils que je juge nécessaire pour mon confort :

```
cd /srv/salt
nano tools.sls
```

Insérer le contenu suivant dans ce fichier :

```
Tools Installation:
  pkg.installed:
    - pkgs:
      - tree
```

« Tools Installation » est un ID représentant donc cette consigne, il peut être changé selon vos besoins.

« pkg.installed » permet de vérifier si un paquet a été installé, si ce n'est pas le cas, il l'installera.

« - pkgs : » n'est pas vraiment nécessaire, car il n'y a qu'un seul paquet à installer actuellement, mais elle permet normalement une liste de paquets à vérifier.

« - tree » est le paquet que l'on cherche à installer

Il est capital de respecter les deux espaces à chaque branche, si cela n'est pas le cas, la consigne ne fonctionnera pas.

Ensuite, nous allons modifier le top.sls pour que ces consignes s'appliquent à tous les minions:

```
nano /srv/salt/top.sls
```

Insérer ceci :

```
base:

 "*":
  - tools
```

Explications du top.sls actuel :

« base : » Applique-les states de l'environnement de base (Il n'y a pas besoin d'y toucher la majorité du temps, un seul en haut du fichier suffit)

« "*" : » Indique que tous les minions sont concernés par les states suivants

« - tools » Notre fichier tools.sls que nous avons créés pour l'installation du paquet tree

Avec un seul minion, il est difficile de montrer toutes les possibilités, voici donc un fichier top.sls plus complet pour une infrastructure plus complexe afin de vous montrer un exemple :

```
base:
  "SQL-server":
    - mariadb
    - backup_python
    - sending_backups_to_OOS

  "SQL*":
    - mariadb
    - mariadb-redundancy
#    - slave_user_mariadb
#    - wordpress_user

  "LAB-server":
    - nfs-server
    - docker_prerequisites

  "MON-server":
    - prometheus
    - alertmanager
    - grafana

  "WP*":
    - wordpress
    - nfs-client
    - NFS-syncs
    - wordpress_server_several_db

  "HAP-server":
    - haproxy


  "*":
#    - apt_update
    - node_exporter
    - tools
```

Vous pouvez donc voir qu'avec un nombre plus important de minions et de states à utiliser ; le top peut être très pratique pour savoir quels éléments sont sur quelles instances.

Ensuite, pour appliquer les states définir dans le top.sls, faites la commande suivante :

```
salt “*“ state.apply
```

Vous pouvez remplacer l’astérisque par un ID précis d’un minion afin d’éviter de perdre du temps, cela devient véritablement un problème dans certains cas, car les minions ne donnent un retour que lorsqu’ils ont terminé d’exécuté leur states ou lorsqu’ils ont une erreur sûr un state.

Si tout s’est passé comme il faut, vous devriez avoir le retour suivant :

```
root@salt-master:/srv/salt# salt "*" state.apply
salt-minion-1:
----------
          ID: Tools Installation
    Function: pkg.installed
      Result: True
     Comment: The following packages were installed/updated: tree
     Started: 16:18:16.247819
    Duration: 20399.141 ms
     Changes:
              ----------
              tree:
                  ----------
                  new:
                      2.0.2-1
                  old:

Summary for salt-minion-1
------------
Succeeded: 1 (changed=1)
Failed:    0
------------
Total states run:     1
Total run time:  20.399 s
```

Félicitations, vous avez réussi à utiliser SaltStack.

4.3 – Commandes récurrentes de state

Il m'est impossible de présenter toutes les consignes possibles dans un state, cependant voici celles qui selon mon avis reviennent le plus souvent (Néanmoins je n'inclus pas tous les paramètres de chaque commande, car certains ne sont pas toujours utiles):

RAPPEL : Les ID de consignes peuvent être changé sans problème, la seule obligation est qu'il n'y ait jamais 2 ID identiques.

file.directory : Permet de s'assurer de l'existence d'un dossier en le créant s'il n'existe pas, voici son utilisation la plus courante :

```
Create Directory: #ID de la consigne
 file.directory:
  - name: /directory #Emplacement du dossier à vérifier
  - user: user #Propriétaire du dossier
  - group: group #Groupe propriétaire du dossier
  - mode: 744 #Autorisations du dossier
```

group.present : S'assure de l'existence d'un groupe d'utilisateur

```
Create Group: #ID de la consigne
 group.present:
  - name: group #Nom du groupe
```

user.present : Identique à group.present mais pour les utilisateurs

```
Create User: #ID de la consigne
 user.present:
  - name: user #Nom de l'utilisateur
```

pkg.installed :Permets d'installer des paquets de logiciel

```
Install Package: #ID de la consigne
 pkg.installed:
  - pkgs: #Déclare que l'on souhaite installer plusieurs paquets (ça ne pose pas de problème si il n'y en a qu'un seul au final)
    - paquet 1 #Nom d'un paquet (par exemple Apache2, tree, net-tools...)
    - paquet 2
```

cmd.run : Dans certains cas, il n'existe pas de consigne salt possible pour une action, dans ce cas-là, on utilise cette consigne

```
Run command: #ID de la consigne
  cmd.run:
    - name: "commande" #La commande que vous voulez exécuté, elle doit être obligatoirement entre guillemets
```

file.managed : Vérifie si un fichier est bien présent à un emplacement, et le télécharge si ce n'est pas le cas

```
Verify file: #ID de la consigne
  file.managed:
    - name: /directory/file #Emplacement du fichier à verifier
    - source: salt://directory/file #Importe le fichier depuis le salt-master en prenant comme racine /srv/salt
    - mode: 744 #Sécurité du fichier
```

service.running : S'assure du bon fonctionnement d'un service

```
Verify service is running: #ID de la consigne
  service.running:
    - name: service-name #Nom du service à vérifier
    - enable: True #Active le service au démarrage de la machine
    - reload: True #Redémarre le service pour appliquer des changements
```

4.4 – Utilisation des pillars

RAPPEL : Les pillars sont des données spécifiques attribuées aux minions selon le top.sls des pillars, leurs utilités sont multiples, mais on s'en sort généralement pour les mots de passe et adresses IP.

Pour commencer, nous allons créer un pillar nommé test.sls et lui donner des informations que le minion devra restituer.

```
nano /srv/pillar/test.sls
```

Dans ce pillar, nous allons mettre le contenu suivant

```
information: Hello world
```

Puis dans le top.sls positionner dans /srv/pillar, rajouter ceci :

```
base:
  ”*”:
    - test
```

Désormais, rafraîchisser les pillars pour les attribuer aux minions :

```
salt ”*” saltutil.pillar_refresh
```

Enfin créé un test.sls dans le dossier /srv/salt avec le contenu suivant :

```
Pillar verification:
  cmd.run:
    - name: echo {{pillar["information"]}} #La récupération de données provenant de pillars est similaire aux dictionnaires du langage Python
```

Rajouter une ligne pour ce test dans le top.sls des states, si vous avez fait la première utilisation des states le top.sls devrait maintenant ressemblé à ça :

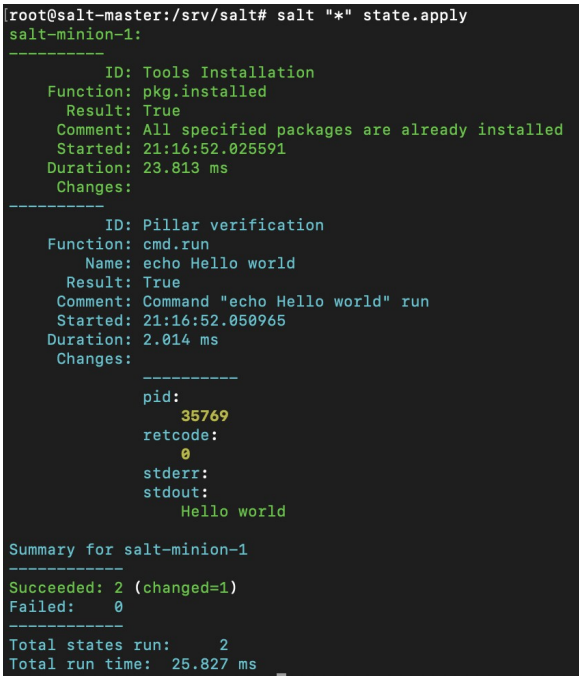
```
base:

  "*":
    - tools
    - test
```

Appliquer les states et observer le résultat :

```
salt ”*” state.apply
```

Si tout s’est bien passé, vous avez le message suivant :



Félicitations, le state a bien réussi à récupérer l’information dans le pillar.

4.4 Addendum – Boucle pour variables à multiples valeurs

Dans certains cas, un logiciel a besoin de l'adresse IP de chaque machine avec qui il est censé communiqué, et modifier manuellement le fichier gérant ces adresses peut être fastidieux. On peut utiliser Jinja dans un state dans ces cas-là.

Jinja est un langage de programmation utilisable dans les states, voici un exemple d'application de boucle dans un state selon le nombre de valeurs dans un pillar :

Pillar utilisé :

```
EveryIP:

  - name: SQL-server
    IP: 127.0.0.1

  - name: LAB-server
    IP: 127.0.0.1

  - name: WP-server
    IP: 127.0.0.1

  - name: MON-server
    IP: 127.0.0.1
```

Boucle dans un state profitant de ce pillar, les termes participant à Jinja sous en gras :

{% for machines in salt['pillar.get']('EveryIP') %} # La boucle se répétera pour chaque valeur à enregistrer présente dans le dictionnaire « EveryIP »
- targets: **["{{ machines["IP"] }}"]** #Récuperation des valeurs mis sous la clé « IP » dans le pillar
labels:
hostname: "**{{ machines["name"] }}**" #Récupération des valeurs mis sous la clé « name » dans le pillar
{% endfor %}

Cependant, cette utilisation n'est pas nécessaire à petite échelle, mais elle peut le devenir pour le confort ou simplement un fichier devenant trop important pour être modifié sans perdre du temps.

5 - Annexes

Wiki de SaltStack : https://docs.saltproject.io/en/latest/contents.html

5.1 – Fiche de recette

Fiche de recette

Vérification de l'opérationnalité de la solution mise en œuvre : *SaltStack*
Description du test : 1. *Test de connexion entre master et minions par commande salt* 2. *Test de consigne*
Résultats Attendus : 1. *Retour positif du test* 2. *Renvoi correct de la consigne*

Réception Globale : SaltStack **Date:** **Auteurs:Timothée LIGNIERE**
Reçu : ☐ Reçu avec réserve : ☐ …… Refusé : ☐…… Commentaire :

Recette étape par étape *	****** (pour chaque étape, vous devez élaborer dans un fichier distinct un scénario détaillé à faire appliquer au « client » venant valider votre solution)***

Réception Etape 1: Test de connexion en faisant salt « * » test.ping , ce qui devrait
Reçu : ☐ Reçu avec réserve : ☐…… Refusé : ☐…… Commentaire :

Réception Etape 2 : Test de consignes avec le state fournis dans 4.2 Première Utilisation
Reçu : ☐ Reçu avec réserve : ☐…… Refusé : ☐…… Commentaire :