Mission 4 : Sauvegarde et Sécurisation des Données

L'objectif principal de cette mission était de mettre en place une solution de sauvegarde efficace et sécurisée. Plusieurs méthodes existent, comme les sauvegardes manuelles, automatisées ou les solutions de stockage en réseau. Ici, nous avons opté pour une solution basée sur **rsync** et une **authentification SSH par clés** sans passphrase, permettant d'automatiser et de sécuriser les transferts de fichiers entre les différentes machines.

Utilité sauvegarde rsync

> **Pourquoi utiliser une sauvegarde basé sur rsync ?**
>
> Il existe plusieurs méthodes de sauvegarde, parmi lesquelles les sauvegardes manuelles, les sauvegardes automatisées et les solutions de stockage en réseau. L'objectif principal est de garantir une redondance des données tout en assurant une restauration rapide et efficace. Dans ce contexte, une solution basée sur rsync et une authentification SSH par clés sans passphrase permet d'automatiser et de sécuriser les transferts de fichiers entre les différentes machines, réduisant ainsi les interventions manuelles et les risques d'erreurs. Cette approche garantit une continuité de service tout en limitant les temps d'arrêt et en assurant une récupération rapide des fichiers en cas de besoin.

Création de l'utilisateur backup2 et backup1

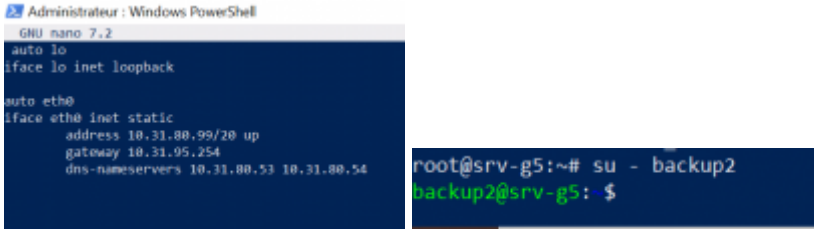
L'utilisateur **backup2** a été créé sur le conteneur de sauvegarde et tous les conteneurs visé par une sauvegarde cela est impératif, pour une sauvegarde sans intervention humaine.

```
adduser backup2 // pour le conteneur backup2
adduser backup1 // pour le conteneur backup1
```

Ensuite creer le conteneur backup2 pour ce faire copier le conteneur template

```
lxc-copy -n template -N backup2
```

Le conteneur **backup2** a été configuré avec l'adresse IP **10.31.80.99** et **10.31.80.98** pour le **backup1** dans le fichier **/etc/interfaces/network** et sert à centraliser les fichiers sauvegardés



Génération des clés SSH

Une authentification par clé SSH permet à **backup2** et **backup1** d'établir une connexion sécurisée sans demander de mot de passe. La génération de la clé se fait avec :

```
ssh-keygen -t rsa -b 4096
```

La commande suivante permet d'afficher la clé publique en entier et ainsi d'éviter des erreurs de clés incomplètes ce qui avait été mon problème .

```
cat  id_rsa.pub
```

Distribution de la clé SSH

La clé publique générée (**id_rsa.pub**) a été copiée sur les conteneurs concernés (web2,ns2,ns1,ns3 et web) en l'ajoutant au fichier `~/.ssh/authorized_keys` de chaque cible .Si le fichier .ssh n'existe pas il faut le créer .

```
Mkdir .ssh && nano authorized_keys
```

Une fois créer y placer la clé publique.

Chemin De La Clé

Vérification des connexions SSH

Après avoir déployé les clés, il faut vérifier que l'authentification SSH fonctionne correctement sans demande de mot de passe :

```
ssh backup1@10.31.81.80
```

Si la connexion réussit sans demande de mot de passe, alors la configuration est correcte.

Cette authentification SSH sans passphrase permet ainsi d'automatiser totalement les échanges de données entre les serveurs, garantissant une **synchronisation fluide et sécurisée** des fichiers de sauvegarde. On l'a fait avec l'utilistaeur de ma binôme le **backup1** pour se connecter au serveur **web**

Script de lancement de la synchronisation

Nous devons maintenant créer un script BASH permettant de synchroniser tous les répertoires à sauvegarder sur tous nos serveurs avec un répertoire dans /home/backup/ de la machine de sauvegarde.

Mais cela dépendra de la méthode , nous avons opté pour la méthode **PULL** sur cette méthode le script est exécuté sur la machine de sauvegarde (backup1 ou 2) .
Le script devra :
-Synchroniser les répertoires
-Créer un fichier de log permettant de garder une trace des sauvegardes
-Un fichier de log différent devra être créé chaque jour.
Notre fichier s'appellera **sauvegarde.sh**

```
#!/bin/bash

LOGFILE="backuplog_$(date +%d-%m-%Y).log"
rsync -azv --rsync-path="sudo rsync" -e ssh backup1@10.31.80.53:/etc/bind
/home/backup1/ns1 >> $LOGFILE
rsync -azv --rsync-path="sudo rsync" -e ssh backup1@10.31.80.54:/etc/bind
/home/backup1/ns3 >> $LOGFILE
rsync -azv --rsync-path="sudo rsync" -e ssh backup1@10.31.80.80:/etc/apache2
/home/backup1/web >> $LOGFILE
rsync -azv --rsync-path="sudo rsync" -e ssh
backup1@10.31.80.80:/var/log/apache2 /home/backup1/web >> $LOGFILE
rsync -azv --rsync-path="sudo rsync" -e ssh
backup1@10.31.80.80:/var/www/html /home/backup1/web >> $LOGFILE
rsync -azv --rsync-path="sudo rsync" -e ssh
backup1@10.31.80.53:/var/log/syslog /home/backup1/ns1 >> $LOGFILE
rsync -azv --rsync-path="sudo rsync" -e ssh
backup1@10.31.80.54:/var/log/syslog /home/backup1/ns3 >> $LOGFILE
rsync -azv --rsync-path="sudo rsync" -e ssh backup1@10.31.80.1:/etc/rc.local
/home/backup1/rc.local >> $LOGFILE
rsync -azv --rsync-path="sudo rsync" -e ssh
backup1@10.31.80.1:/etc/resolv.conf /home/backup1/resolv.conf >> $LOGFILE
rsync -azv --rsync-path="sudo rsync" -e ssh
backup1@10.31.80.1:/home/htdocs/m2l.org /home/backup1/web >> $LOGFILE
```

Nous avons sauvegardé :
- les fichiers de configuration d'apache2 du conteneur web
- les fichiers de logs d'apache2
- les fichiers de logs des serveurs DNS
- les fichiers de logs de Bind
- les bases de données du serveur web
- Fichiers de configuration du serveur

Ce script sera dans un fichier de sauvegarde dans le conteneur backup1 en utilisant l'utilisateur **backup1** avec la commande :

```
su - backup1 // se connecter à l'utilisateur
```

Ajout des droits de l'Utilisateur backup1

Nous devons configurer les droits dans le fichier **sudoers** qui définit les privilèges d'accès des utilisateurs pour exécuter des commandes avec des droits élevés .
L'utilisateur backup1 peut exécuter la commande **/usr/bin/rsync** sans avoir besoin d'entrer de mot de passe **(NOPASSWD)**.
NB: cela se fait dans tous les conteneurs **backup1 ns1 ns2** ect mais en étant le **root**

```
backup1 ALL=NOPASSWD: /usr/bin/rsync
```

```
# User privilege specification
root    ALL=(ALL:ALL) ALL
backup1 ALL=NOPASSWD: /usr/bin/rsync

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

Script De Sauvegarde pour les Bases de Données su Serveur Web

Nous devons aussi faire un script de sauvegarde des bases de données qui se trouvent dans le serveur Web, elles se trouvent dans le **var/www/html** donc faire une redirection vers ce dossier . On devra se connecter en tant que l'utilisateur **backup1**.

```
backup1@web:~$ cat database.sh
#!/bin/bash

LOGFILE="databaselog_$(date +d%-%m-%Y).log"
mysqldump --all-databases -u root -pdrowssap > /var/www/html/$LOGFILE
echo $LOGFILE
backup1@web:~$
```

```
#!/bin/bash

LOGFILE="databaselog_$(date +d%-%m-%Y).log"
mysqldump --all-databases -u root -pdrowssap > /var/www/html/$LOGFILE //
Sauvegarde toutes les bases de données présentes sur le serveur.
echo $LOGFILE
```

Synchronisation Automatisée avec Cron

Cron est un planificateur de tâches sous Unix/Linux qui permet d’exécuter automatiquement des commandes ou des scripts à des horaires prédéfinis. Il est très utilisé pour automatiser des tâches répétitives comme :
- La sauvegarde des fichiers
- La mise à jour des bases de données
- L’exécution de scripts de maintenance
- L’envoi d’e-mails automatiques

Un démon nommé cron lit les fichiers qui sont dans une crontab. Elle se trouve dans **/etc/crontab**
Chaque utilisateur possède une crontab personnelle et pour l’éditer, il faut taper :

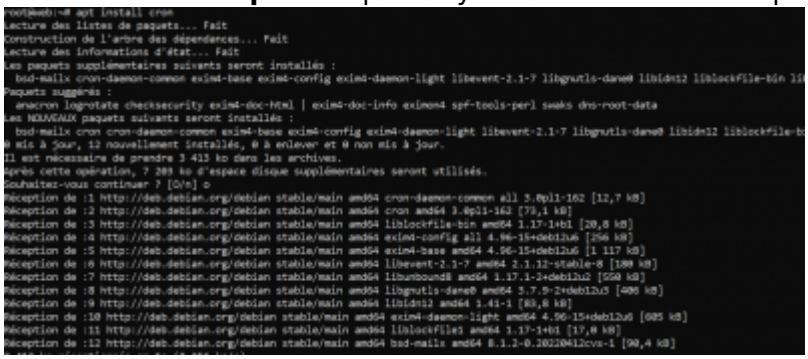
```
crontab -e
```

a)Structure d'une crontab

```
# Example of job definition:
# .---------------- minute (0 - 59)
# | .------------- hour (0 - 23)
# | | .---------- day of month (1 - 31)
# | | | .------- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .---- day of week (0-6) (Sunday=0) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * commande que l'utilisateur doit exécuter
```

b) Installation de Cron

Premièrement nous devons installer **Cron** dans tous les conteneurs plus précisemment dans l'utilisateur **backup1** afin que la synchronisation soit complète et fonctionne .



c) Arès Installation de Cron

Après avoir installé **Cron** , on doit se rendre dans le le conteneur **backup1** et l'utilisateur **backup1** . Puis nous devons nous devons nous rendre dans le dossier **/etc/crontab** et taper la commande **crontab -e**

d) Aperçu de backup1 et de Toutes Ses Sauvegardes

```
backup1@backup1:~$ ls -l
total 400
-rw-r--r-- 1 backup1 backup1   3183  6 févr. 17:51 backuplog_06-02-2025.log
-rw-r--r-- 1 backup1 backup1 319731  7 févr. 17:58 backuplog_07-02-2025.log
-rw-r--r-- 1 backup1 backup1   1159  8 févr. 02:00 backuplog_08-02-2025.log
-rw-r--r-- 1 backup1 backup1   1150  9 févr. 02:00 backuplog_09-02-2025.log
-rw-r--r-- 1 backup1 backup1   1153 10 févr. 02:00 backuplog_10-02-2025.log
-rw-r--r-- 1 backup1 backup1     83  7 févr. 15:43 database.sh
drwxr-xr-x 3 backup1 backup1   4096 10 févr. 02:00 ns1
-rw-r----- 1 backup1 backup1  30940  7 févr. 10:33 ns2
drwxr-xr-x 3 backup1 backup1   4096 10 févr. 02:00 ns3
-rwxr-xr-x 1 backup1 backup1    185  6 févr. 11:33 rc.local
-rw-r--r-- 1 backup1 backup1     19 29 janv. 15:36 resolv.conf
-rwxr-xr-x 1 backup1 backup1   1106  7 févr. 17:40 sauvegarde.sh
-rwxr-xr-x 1 backup1 backup1    185  6 févr. 11:33 srv-g5
drwxr-xr-x 4 backup1 backup1   4096  7 févr. 14:36 web
```

d) Explication de #!/bin/bash

Le **#!/bin/bash**, appelé shebang, est une ligne placée au début d'un script shell qui indique quel interpréteur utiliser pour exécuter le script.

```
#! // C'est le shebang, un marqueur qui signale au système d'exploitation quel interpréteur utiliser.
/bin/bash //C'est le chemin vers l'interpréteur Bash (Bourne Again Shell), qui est utilisé pour exécuter le script.
```