**Etape 1 du plan – Compréhension et capabilité structurelle**

Idée Générale de l’infrastructure

* Surface de contact WAN avec une VM Bitnami préconfigurée

Son support garantie des mises à jour et offre donc une bonne interface de contact WAN

* Des VM configurées manuellement pour les accès authentifiés

Leur liberté de réglage permettra le déploiement personnalisé d’une plateforme privée pour des utilisateurs authentifiés

* Accès sécurisé aux bases de données et partages de documents

Des solutions iSCSI et NFS pour gérer les accès sécurisés et redondants aux bases de données et aux partages. Ces solutions auront eux-mêmes leur propre résilience

* Rôle de la machine hôte physique

Porter la virtualisation imbriquée via VirtualBox dans un premier temps, puis Workstation en migration. Cette étape permettra d’effectuer et de documenter les tests de migration. La machine hôte effectuera les tout premiers tests du PC client avant de laisser ce rôle à un autre ordinateur quelconque.

* Composants critiques à déployer

Load balancing et monitoring

C’est la répartition des charges (load balancing) et la surveillance continue (monitoring) de l’infrastructure pour vite y détecter les anomalies

Tests de failover

Ce sont les tests de basculement pour s’assurer que la redondance fonctionne comme prévu en cas de panne d’un composant

Sécurité et mises à jour

On planifie des notifications et des alertes pour veiller à garder les composants à jour et ainsi leur appliquer les correctifs dès qu’ils sont disponibles

L’axe de la robustesse

* Tests approfondis

L’implémentation des composants critiques ne suffit pas. Il faut les tester rigoureusement avant la mise en production en tenant compte des fluctuations dues aux pics d’activité

* Sécurité renforcée

Même si l'architecture est segmentée et sécurisée, il faut intégrer des audits de sécurité réguliers, mettre en place des mises à jour automatiques ou programmées et monitorer la sécurité (IDS/IPS). Il ne faut pas traquer ces informations manuellement. L’administration manuelle est l’acte de configurer l’automatisation et de veiller sur son bon fonctionnement. Attention ! L’intelligence artificielle nous prouve tous les jours que les données dites automatisées peuvent être tronquées et ou falsifiées. C’est là le travail véritable que celui de maîtriser les processus, pour ne pas dire *« surveiller l’Automate »*.

* Plan de reprise après sinistre (DRP)

Proposer une stratégie de sauvegarde et de restauration complète incluant un plan de continuité d'activité qui fait face aux incidents majeurs

* Surveillance et monitoring en temps réel

Intégrer des outils de monitoring et de logging pour suivre l'état de l'infrastructure, anticiper les problèmes et réagir rapidement en cas d'incident. Les scénarii d’intervention doivent être prêts à l’avance pour éviter toute panique

* Automatisation et gestion des configurations

Garantir la reproductibilité du déploiement avec Ansible, Puppet et Docker Compose

* Documentation détaillée

Une documentation complète et à jour peut être pensé au regard de la complexité du travail informatique et des potentialités dynamiques du web pour les « data mémoire ».

Concernant le clustering et son positionnement dans le projet

L’approche consiste à réserver le clustering et la redondance globale pour la fin du projet

* Points de réussites cumulatifs

En implémentant d'abord des éléments unitaires stables et opérationnels, je vais minimiser les risques en cumulant des réussites fiables et autonomes, une par une, pour les intégrer toutes dans le clustering global.

* Disponibilité des ressources

Cette approche vaut d’avancer utilement en attendant d'avoir les ressources suffisantes : matériel, tests fonctionnels et retours d’expérience

* Vue d'ensemble du maillage final

Une fois les composants de base testés et validés, j’aurai une meilleure vision globale de l’architecture et des interactions des éléments. Ce discernement me facilitera l’optimisation du maillage en cluster

* Continuité de service hors du cluster

Il faut concevoir chaque élément de manière qu’il puisse fonctionner avec autonomie tout en étant intégré au cluster afin de garantir une continuité en cas de défaillance du cluster principal en lui-même. Cette approche anticipe les plans de sinistres et les rend palpables

Bilan des installations et des environnements de test

* Environnements existants

- VM Bitnami (préconfigurée pour Drupal)

- VM Ubuntu avec Drupal déployé sur Docker

🡪 Etude des avantages et inconvénients pour une meilleur disposition dans le cluster

* Etape actuelle : comparaison technique entre Ubuntu et Debian

- 1 VM Debian avec Drupal déployé sur Docker

- 2 VM Ubuntu et 2 VM Debian pour une installations natives sur leurs OS

- 2 VM Ubuntu et 2 VM Debian dédiées aux composants avancés tels que Bind et HAProxy

🡪 recherche des nuances Debian/Ubuntu, même minimes, pour maîtriser leur positionnent

🡪 recherche des nuances LAMP/LEMP, même minimes, pour maîtriser leur positionnent

🡪 renoncement au déploiement de Drupal sur Windows. Drupal est optimisé pour Linux.

Les VM Windows pourraient néanmoins est déployées en tant que postes Clients dans le réseaux Lan pour effectuer des tests avancés qui simulent une infiltration de l’intérieur, entre via WSL pour renforcer le test (machine & AI in the middle : référence à « man in the middle »). Cette d’expérimentation dépendra des ressources effectives de l’hôte une fois en action

Utilisation de solutions DNS et choix de clustering

* DNS sur VM Linux

D’avoir renoncé au rôle actif de Windows dans le cluster, j’ai choisi Bind pour mon système DNS. Je reste ainsi homogène avec un service qui s’intègre bien sur toutes les distributions majeures de Linux.

* Clustering avec Proxmox

J’ai opter pour une solution de clustering basée sur Proxmox pour : 1- centraliser la gestion des machines virtuelles ; 2- orchestrer les ressources ; 3- assurer la résilience.

* Distribution des rôles

- Une VM rooter robuste (type PFSens) : protéger du flux massif initial (avec sa propre redondance)  
🡪 Bien que disposant de leurs propres pare-feu les composants se situent déjà sur un Lan

- Une VM de filtrage administratif robuste (type AdGuard) : filtrer les robots (avec sa propre redondance)  
🡪 Bien que disposant de leurs propres pare-feu les composants se situent déjà sur un Lan

- Une VM pour le reverse proxy et load balancing (avec sa propre redondance)  
🡪 Nginx ou HAProxy pour gérer le trafic WAN et rediriger les requêtes vers Drupal

- Une VM pour : Monitoring, Logs et Gestion Centralisée dont backups (avec sa propre redondance)  
Celle-ci pourrait intégrer la surveillance, la journalisation et la gestion des sauvegardes

- Une VM pour le service DNS et quelques fonctionnalités moins majeurs (avec sa propre redondance)  
Le reste des fonctionnalités autres que « le site Drupal » seront réparties rationnellement

- Une VM pour les bases de données (avec sa propre redondance)

Les BDD seront développées avec pluralité pour clarifier la pensée du système de données

Le reste des fonctionnalités autres que « le site Drupal » seront réparties rationnellement

- Une VM de stockage intermédiaire vers le Front et de bases analytiques (avec sa redondance)

Les BDD ne sont pas faites pour être consultées couramment (ressources du site et data science)

Les BDD ne proposent que des accès via iSCSI et le partage NFS (sauvegardes régulières)

- Une VM pour les backup et les archives (avec sa propre redondance)

Le véritable système d’archivage à longue durée de versionning et lutte contre les sinistres

- Toutes les VM du site Drupal en lui-même (avec leur propres redondances)

Le véritable système d’archivage à longue durée de versionning et lutte contre les sinistres

Identification des besoins génériques de l'infrastructure

A présent je suis capable de définir les composants principaux du système d’information pour mieux justifier ce en quoi la formule d’isolation choisie alourdit certes le système mais en assure réellement la sécurité et lui donne même, à bien des égards, une efficacité accrue

* Synthèse du besoin en termes de « composants »

1- Serveur DNS (Bind sur VM Linux)

2- Clustering (avec Proxmox)

3- Reverse proxy / Load Balancer (ex. Nginx ou HAProxy)

4- Environnements applicatifs pour Drupal (installations natives, Docker, Bitnami)

5- Base de données et solutions de stockage (iSCSI, NFS)

6- Monitoring et gestion centralisée (monitoring, logs, backups)

7- Serveur de gestion centralisée (pilotage via une VM dédiée)

* Liste des services associés à chaque composant

- Détailler chaque composant en termes d'actions à réaliser

1- commandes à exécuter dans le Terminal

2- Les lignes à écrire et ou modifier

3- Les scripts à créer et ou modifier

4- Les fichiers de configuration à paramétrer

5- Les manipulations via d’interfaces graphiques d’utilisateur le cas échéant

6- Les chemins, logs et notes explicatives, pour les mots clés et les syntaxes spécifiques

7- Des notes de synthèse pour valider la configuration et faciliter sa maintenance

Le tout peut ainsi être revu avec simplicité et organisé sous forme de checklist dans un document de référence pour guider la mise en production.

* Directive de production des tutoriels

- La directive consiste à s’appuyer sur cette vue d’ensemble pour revenir aux pratiques unitaires du client, l’utilisateur du site web, afin de relier chacune de ses interactions ainsi reclarifiée à un service performant offert par « l’Infrastructure Drupal déployée » et non pas le « site vitrine, seul ».

- En commençant par l'installation des VM, la configuration des réseaux (WAN, NAT, Host Only) et des pare-feu, je pose d’abord une base solide qui garantira de ne pas rencontrer de "problèmes gris" par la suite.

- Les solutions de routage tels que PFSens et AdGuard à envisager ne doivent pas être pensées comme une économie des mesures de protection unitaires de chaque VM, de chaque conteneur Docker. Définir clairement, systématiquement, les interfaces réseau et les règles de pare-feu à l’issue de chaque déploiement pour inclure ce développement au déploiement, comme une gymnastique rituelle, préserve la sécurité de l'ensemble des services et évite que des flux non autorisés ne perturbent le système. De fait, il n’est pas impossible que les premières barrières soient franchies, bien que robustes et efficaces. Il n’est pas impossible non plus que ces premières barrières soient sinistrées. L’infrastructure doit tenir le temps que les alternatives s’appliquent. La protection propre de chaque composant sera ainsi poussée plus en avant, plus tard, au stade des clusterings, afin d’*« incarner en termes de maillage la véritable redondance du système de sécurité »*

Les étapes de professionnalisation

La stratégie consiste à faire migrer progressivement cette infrastructure de VirtualBox vers Workstation (voire ESXi) pour évoluer vers une solution professionnelle en bare-metal.

* En route vers Proxmox en bare-metal

Migration des VM de VirtualBox vers Workstation

Migrer vers Workstation permettra d'obtenir une meilleure gestion des ressources, une prise en charge avancée de la virtualisation imbriquée et une interface plus robuste pour un environnement de développement professionnel d’emblée pleinement compatibles avec les solutions en bare-métal comme ESXi qui fait partie du même ensemble de produits Vmware que Workstation. La mise en production sur le cloud dédié ou communautaire ne devrait alors souffrir d’aucune contrariété puisque ce sont les mêmes types de structurations

* Exportation/Importation OVF/OVA

Depuis VirtualBox il est aisé d'exporter chaque machines virtuelles au format OVF/OVA. Ce format très bien reconnu par VMware Workstation. Il n’y a donc pas de problème de migration entre les hyperviseurs de type 2

* Adaptations mineures

Une fois importées dans Workstation, il peut être nécessaire d'ajuster quelques paramètres (comme les contrôleurs de disque, les pilotes réseau, etc.) pour que la VM fonctionne de la même manière qu'avant. A ce stade avancé de ce processus général, il sera toujours temps de faire les tutoriel concernés

* Test et validation

Il est recommandé de tester minutieusement chaque VM après migration pour s'assurer que toutes les configurations (réseau, stockage, pare-feu, etc.) fonctionnent correctement.

* En attendant …
* VirtualBox

- Il est gratuit, simple, idéal pour les environnements de développement et les tests.

- Il faut être très rigoureux sur les étapes de validation de la virtualisation imbriquée pour l‘installation de Proxmox dans une VM. Le résultat final de l’infrastructure sera plus délicat à manipuler qu’une solution payante mais il est évéré temporaire. Donc il n’y a as de problème à ce niveau. La stratégie qui consiste à finaliser chaque VM individuellement en dehors du cluster pour bénéficier pleinement de potentialité de VirtualBox garantir une certitude unitaire vient également de cette réalité.

Préparation de l'Environnement sur VirtualBox

* Installation de VirtualBox  
  Télécharger la dernière version de VirtualBox et l’installer pas à pas
* Activation de la virtualisation imbriquée  
  Pour les VM qui exécuteront Proxmox et celles où Docker sera installé, vérifier que cette option est activée dans les paramètres de VirtualBox

**Accès → VirtualBox → sélectionner la VM → Configuration → Système → Processeur et cocher « Activer VT-x/AMD-V » ainsi que « Activer la virtualisation imbriquée », si disponible**

* Ressources matérielles suffisantes

Veillez à disposer d'assez de RAM, CPU et stockage pour héberger vos VM. Il ne faut pas espérer tester une telle infrastructure complète avec un ordinateur dont le processeur n’est pas au moins l’équivalent d’un Intel 5 qui dispose d’au moins 16Go de RAM et de 8 cœurs au minimum. Cette configuration minimale rencontrerait déjà des difficultés significatives même si en utilisant les éléments par blocs partiels cela reste possible. Ce serait néanmoins une affaire de confort considérable à ne ps négliger avant de s’engager. Un équivalent de Intel 7 doté de 32Go de RAM et 16 cœurs (Virtual CPU) n’assure déjà pas 100 % du confort nécessaire à un tel projet (je parle de confort). A bon entendeur …

Installation une VM initiale de contrôle provisoire dans VirtualBox

Création d'une VM de Base sur Linux pour jouer les rôles d’interactions clients et autres essais de communication sur le réseau privé (LAN)

* Ouvrir VirtualBox et cliquer sur "Nouvelle" :

Choisir un nom explicite (ex. VM-Drupal-App-01)

Sélectionner le type et la version (par exemple, Linux - Ubuntu 64 bits ou Debian 64 bits)

* Configuration des ressources :

Mémoire RAM : donner une quantité suffisante (par exemple, 4 Go ou plus selon vos besoins)

Disque dur virtuel : de 20 à 40 Go voir plus selon les ressources le rôle dans le système

* Rappel : configuration de la virtualisation imbriquée (VM Proxmox et Docker)

Dans Configuration → Système → Processeur, cochez « Activer VT-x/AMD-V » et « Activer la virtualisation imbriquée ».

* Configuration des Interfaces Réseau dans VirtualBox

Chaque VM devra disposer de 4 plusieurs adaptateurs réseau

Adaptateur 1 (WAN)

Mode : « Accès par pont » (bridge) pour simuler un réseau public et un accès direct à l'extérieur

Adaptateur 2 (NAT)

Mode : « NAT » : translation d'adresse : accès sécurisé via l'hôte sans être directement visible

Adaptateurs 3 & 4 (Host Only)

Mode : « Réseau interne » : local, privé et isolés du reste d’internet sans accès direct

* Appliquer la configuration et démarrer la VM

Données génériques de configuration des Réseaux et des Pare-feu

RAPPEL : exemple de fichier Netplan pour le réseau sur Ubuntu

yaml

version: 2

renderer: networkd

ethernets:

enp0s3:

dhcp4: no

addresses: [192.168.1.10/24] # WAN (Adaptateur 1)

gateway4: 192.168.1.1

nameservers:

addresses: [8.8.8.8,8.8.4.4]

enp0s8:

dhcp4: yes # NAT (Adaptateur 2)

enp0s9:

dhcp4: no

addresses: [10.0.0.10/24] # Host Only 1 (Adaptateur 3)

enp0s10:

dhcp4: no

addresses: [10.0.1.10/24] # Host Only 2 (Adaptateur 4)

* Appliquer la configuration

sudo netplan apply

Configuration du Pare-feu (UFW)

* Installer et activer UFW

sudo apt update && sudo apt install ufw

sudo ufw enable

* Définir les règles de base (par exemple, pour SSH, HTTP, HTTPS)

sudo ufw allow ssh

sudo ufw allow in on enp0s3 to any port 80 proto tcp

sudo ufw allow in on enp0s3 to any port 443 proto tcp

sudo ufw default deny incoming

sudo ufw default allow outgoing

* Vérifier les règles

sudo ufw status verbose

* La configuration des réseaux et des pare-feu suit les mêmes principes pour quèsement toutes les VM de ce projet
* Il faut néanmoins adapter les noms des interfaces réseau en fonction de VirtualBox (par exemple, enp0s3, enp0s8, etc.)

Installation de Proxmox dans une VM VirtualBox

Installer Proxmox en VM dans VirtualBox (pour la phase de développement)

* Télécharger l'image ISO de Proxmox VE

Télécharger l'ISO de Proxmox VE sur le site officiel de Proxmox.

* Créer une VM dédiée pour cet hyperviseur de type 1

Penser à sélectionner un OS compatible (Proxmox VE est basé sur Debian)

Activer la virtualisation imbriquée est activée

* Configurer la VM pour démarrer depuis l'ISO de Proxmox

Dans les paramètres de la VM, sous « Stockage », ajoutez l'ISO comme lecteur optique

Démarrer la VM et suivez l'installation de Proxmox

Configurer le Réseau et autres premières nécessités

Composant → VM Proxmox VE (pour la virtualisation imbriquée) → Checklist

| # | Action | Type et chemin | Détail | Commande / GUI | Logs / Note |
| --- | --- | --- | --- | --- | --- |
| 1 | Créer la VM Proxmox | GUI | Dans VirtualBox → Nouvelle Nom : VM‑Proxmox‑Dev Type : Linux, Version : Debian 64 bits (Proxmox VE est basé sur Debian) | Choisir type "Linux" et "Debian (64-bit)" |  |
| 2 | Allouer ressources | GUI | CPU : 2-4 cœurs, RAM : 4-8 Go, HDD : 32-64 Go (selon disponibilité) | Configurer dans VirtualBox lors de la création de la VM | Adapter selon ressources hôte |
| 3 | Activer Virtualisation imbriquée | GUI | Dans la configuration de la VM → Système → Processeur Cocher « Activer VT-x/AMD-V » et « Virtualisation imbriquée » | Accéder aux paramètres de la VM → Système → Processeur | Vérifier dans BIOS et VirtualBox |
| 4 | Configurer Interfaces Réseau | GUI | Comme pour la VM Ubuntu, configurer 4 NIC : • Adaptateur 1 : Bridge (WAN) • Adaptateur 2 : NAT • Adaptateur 3 & 4 : Host-Only ou Réseau Interne | Paramétrer dans Configuration → Réseau pour VM Proxmox | Adapter le nom des interfaces plus tard (vérifier via ip a dans Proxmox) |
| 5 | Attacher l'ISO Proxmox VE | GUI | Télécharger l'image ISO de Proxmox VE depuis le site officiel Attacher l'ISO dans le contrôleur de stockage de la VM | Dans VirtualBox → Configuration → Stockage → Ajouter l’ISO Proxmox VE |  |
| 6 | Installer Proxmox VE | GUI | Démarrer la VM avec l’ISO attaché Suivre l’installateur Proxmox (choix de partitionnement, configuration réseau initiale, etc.) | Suivre l’interface de l’installateur (graphique ou textuel) | Noter la version installée et paramètres de base |
| 7 | Configuration initiale post-installation | Terminal / GUI | Accéder à l’interface web de Proxmox via l’adresse IP configurée Effectuer les réglages de base (création de stockage, pool de ressources, etc.) | Ouvrir un navigateur, se connecter à https://<IP-Proxmox>:8006 | Vérifier les logs via l’interface Proxmox (système et cluster) |
| 8 | Documentation et vérification | Note de synthèse | Documenter la configuration (IPs, ressources allouées, réglages réseau et de virtualisation imbriquée) | Créer un fichier /docs/VM-Proxmox-Dev.md | Noter toute anomalie ou ajustement à réaliser |

Proxmox dispose de son propre système de logs dans /var/log/pve. Ces données sont également accessibles via l’interface web

🡪 Prendre note de toute anomalie et documenter les réglages appliqués.

Configuration Réseau dans Proxmox VE

* Fichier de Configuration Réseau

Etant basé sur Debian, Proxmox VE utilise généralement le fichier /etc/network/interfaces pour définir la configuration réseau

# /etc/network/interfaces

# Interface WAN (Adaptateur 1 - Bridge)

auto vmbr0

iface vmbr0 inet static

address 192.168.1.100

netmask 255.255.255.0

gateway 192.168.1.1

bridge\_ports enp1s0 # Remplacer "enp1s0" par l'interface physique de l'hôte

bridge\_stp off

bridge\_fd 0

# Interface NAT (Adaptateur 2)

auto vmbr1

iface vmbr1 inet dhcp

bridge\_ports none

bridge\_stp off

bridge\_fd 0

# Interface Host Only 1 (Adaptateur 3)

auto vmbr2

iface vmbr2 inet static

address 10.0.0.1

netmask 255.255.255.0

bridge\_ports none

bridge\_stp off

bridge\_fd 0

# Interface Host Only 2 (Adaptateur 4)

auto vmbr3

iface vmbr3 inet static

address 10.0.1.1

netmask 255.255.255.0

bridge\_ports none

bridge\_stp off

bridge\_fd 0

* Explications

- vmbr0 (WAN) est configuré en statique avec une adresse sur mon réseau public.

🡪 Le paramètre bridge\_ports doit être associé à la NIC de l'hôte. Proxmox agira alors comme un pont pour les VM connectées à vmbr0

🡪 Penser à vérifier les correspondances entre les cartes réseau

ip a

- vmbr1 (NAT) est configuré en DHCP (vous pouvez aussi le configurer statiquement si nécessaire) mais sans pont physique

- vmbr2 et vmbr3 (Host Only) sont utilisés pour la communication interne entre les VM, avec des adresses définies en réseau privé

* Application et Validation

sudo systemctl restart networking

un certain nombre de problèmes peuvent venir des modification à chaud. Il faut donc surveiller les logs

sudo journalctl -u networking

* Nouvelle vérification des Interfaces

ip a

🡪 Vérifier que chaque carte statique a bien l’adresse manuellement attribuée. Les autres doivent apparaître en DHCP

🡪 Tester la connectivité WAN en faisant un ping su toutes les passerelles concernées par le réseau

ping -c 4 192.168.1.1

🡪Tester le DNS et les réseaux Host Only (si les résolveurs ont été configurés)

ping -c 4 8.8.8.8

🡪 Utilisez traceroute pour observer le chemin des paquets, par exemple vers une adresse externe :

sudo apt install traceroute

traceroute 8.8.8.8

🡪 Vérifier les Pare-feu de Proxmox

Proxmox intègre un système de pare-feu au niveau du cluster ET de chaque VM

Accès : 🡪 GUI web 🡪 section "Firewall" de chaque nœud/VM

Accès : 🡪 sudo pve-firewall status

🡪 Etape suivante → Création et installation de la VM Ubuntu de base pour les fonctionnalités dédiées

Composant → VM Base Ubuntu pour des fonctionnalités dédiées (Ubuntu Server 24.04) → Checklist

| # | Action | Type et chemin | Détail | Commande / GUI | Logs / Note |
| --- | --- | --- | --- | --- | --- |
| 1 | Créer la VM | GUI | Dans VirtualBox → **Nouvelle** | Nom : VM‑Drupal‑Base Type : Linux / Ubuntu (64‑bit) |  |
| 2 | Allouer ressources | GUI | CPU=2 cores, RAM=4 Go, HDD dynamique=40 Go |  |  |
| 3 | Activer nested VT-x | GUI | Settings → Système → Processeur → cocher « Activer VT‑x/AMD‑V » + « Virtualisation imbriquée » |  | Vérifier BIOS Hôte |
| 4 | Configurer 4 NICs | GUI | Settings → Réseau : • Adapter1 → Bridge • Adapter2 → NAT • Adapter3 → Host‑Only (vboxnet0) • Adapter4 → Host‑Only (vboxnet1) |  |  |
| 5 | Attacher ISO Ubuntu | GUI | Settings → Stockage → Ajouter ISO Ubuntu Server 24.04 |  |  |
| 6 | Installer Ubuntu | GUI | Démarrer VM → installer Ubuntu Server Création utilisateur, SSH, partitionnement par défaut |  | Installer OpenSSH Server |
| 7 | Setup réseau | Fichier config | Éditer /etc/netplan/01-netcfg.yaml | Voir bloc YAML ci‑dessous | Appliquer avec sudo netplan apply |
| 8 | Setup pare‑feu | Terminal | Installer et configurer UFW | ```bash |  |
|  |  |  |  | sudo apt update && sudo apt install ufw -y |  |
|  |  |  |  | sudo ufw allow ssh |  |
|  |  |  |  | sudo ufw allow in on enp0s3 to any port 80,443 proto tcp |  |
|  |  |  |  | sudo ufw default deny incoming |  |
|  |  |  |  | sudo ufw enable |  |
| ``` |  | Logs : /var/log/ufw.log |  |  |  |
| 9 | Vérifier connectivité | Terminal | Ping vers passerelle, DNS et Host‑Only peers | ```bash |  |
|  |  |  |  | ping -c3 192.168.1.1 |  |
|  |  |  |  | ping -c3 10.0.0.1 |  |

### Bloc YAML — `/etc/netplan/01-netcfg.yaml`

```yaml

network:

version: 2

renderer: networkd

ethernets:

enp0s3:

dhcp4: no

addresses: [192.168.1.10/24]

gateway4: 192.168.1.1

nameservers: { addresses: [8.8.8.8,8.8.4.4] }

enp0s8:

dhcp4: yes

enp0s9:

dhcp4: no

addresses: [10.0.0.10/24]

enp0s10:

dhcp4: no

addresses: [10.0.1.10/24]

étape suivante → Création et installation de la VM ?

**Etape 2 du plan –** Le déploiement des VM dédiées aux applicatifs directes de Drupal (Disons « Drupal VMs »)

Composant → Environnement Drupal via Docker → Checklist

| # | Action | Type et chemin | Détail | Commande / GUI / Contenu de Script ou fichier config |
| --- | --- | --- | --- | --- |
| 1 | Créer une VM Linux dédiée | GUI | Utiliser VirtualBox ou Proxmox pour créer une VM Ubuntu ou Debian dédiée | Comme précédemment  Nom : VM‑Drupal‑Docker, ressources adaptées |
| 2 | Installer Docker | Terminal | Installer Docker Engine sur la VM Linux | ```bash |
|  |  |  |  | sudo apt update |
|  |  |  | La phase de réglage de l’environnement n’est pas détaillée dans cette checklist. Il ne faut donc pas | sudo apt install docker.io -y |
|  |  |  | oublier les dépendances, les signatures de fichiers et les clés d’authentification | sudo systemctl enable docker |
|  |  |  |  | sudo systemctl start docker |
| ``` | Vérifier l’installation |  |  | docker --version |
| 3 | Installer Docker Compose | Terminal / Script | Installer Docker Compose pour orchestrer les services Drupal | ```bash |
|  |  |  |  | sudo apt install docker-compose -y |
| ``` | Vérifier l’installation |  |  | docker-compose --version |
| 4 | Créer un dossier projet Drupal | GUI / Terminal | Créer un répertoire pour héberger la configuration Docker et les données Drupal | ```bash |
|  |  |  |  | mkdir -p ~/projects/drupal-docker |
| 5 | Rédiger le fichier  docker-compose.yml | GUI / Terminal projects/drupal-docker | Définir les services nécessaires :  Web (Apache/Nginx + PHP), Base de données, etc | Exemple de configuration ci‑dessous |
|  | Rédiger toutes les notes nécessaires | GUI / Terminal projects/drupal-docker | Bâtir la mémoire avec redondance | mkdir -p ~/projects/drupal-docker/docs/ docker-compose-drupal.md |
| 6 | Démarrer les services Docker | Terminal | Lancer la stack Docker avec docker-compose puis vérifier que le conteneur a bien démarré | docker-compose up -d  docker ps |
| 7 | Tester la connectivité et l’accès Drupal | Terminal / Navigateur | Accéder à l’interface web Drupal via l’adresse définie dans la configuration réseau | Ouvrir navigateur sur `http://<IP-VM>:<port>`  Vérifier les Logs Docker et les logs web situés dans les conteneurs |
| 8 | Rédiger toutes les notes nécessaires | GUI / Terminal projects/drupal-docker | Bâtir la mémoire avec redondance | Créer un fichier `/docs/VM-Drupal-Docker.md`  synthèse incluant IP, ports, versions, et tout ajustement nécessaire |
|  |  |  |  |  |

### Exemple de fichier docker-compose.yml pour Drupal

```yaml

version: '3.8'

services:

drupal:

image: drupal:11-apache

container\_name: drupal\_app

ports:

- "8080:80"

volumes:

- drupal\_data:/var/www/html

environment:

- DRUPAL\_DB\_HOST=db

- DRUPAL\_DB\_USER=drupaluser

- DRUPAL\_DB\_PASSWORD=drupalpass

- DRUPAL\_DB\_NAME=drupaldb

depends\_on:

- db

db:

image: mariadb:10.5

container\_name: drupal\_db

restart: always

environment:

- MYSQL\_ROOT\_PASSWORD=rootpass

- MYSQL\_DATABASE=drupaldb

- MYSQL\_USER=drupaluser

- MYSQL\_PASSWORD=drupalpass

volumes:

- db\_data:/var/lib/mysql

volumes:

drupal\_data:

db\_data:

Composant → Environnement natif de Drupal 11 sur Debian → Checklist

| # | Action | Type et chemin | Détail | Commande / GUI / Fichier |
| --- | --- | --- | --- | --- |
| 1 | Mettre à jour le système | Terminal | Actualiser les paquets et dépendances sur Debian | sudo apt update && sudo apt upgrade -y |
| 2 | Installer Apache ou Nginx | Terminal | Installer Apache (ou Nginx) pour héberger Drupal | sudo apt install apache2 -y |
| 3 | Installer PHP et extensions | Terminal | Installer PHP 8.1 (ou vers. compatible Drupal 11) et ses modules requis. !! l’exemple est en une seule ligne !! | sudo apt install php php-cli php-fpm php-mysql php-xml php-gd php-json php-curl -y |
| 4 | Installer MariaDB ou MySQL | Terminal | Installer MariaDB pour la base de données | sudo apt install mariadb-server -y |

| # | Action | Type et chemin | Détail | Commande / GUI / Fichier |
| --- | --- | --- | --- | --- |
| 5 | Télécharger Drupal 11 | Terminal | Télécharger la version stable de Drupal 11 depuis drupal.org | wget https://ftp.drupal.org/files/projects/drupal-11.x.tar.gz |
|  | Vérifier somme de contrôle | Terminal  Site officiel de Drupal | ? | ? |
| 6 | Extraire Drupal | Terminal | Extraire l’archive dans le répertoire web (ex. /var/www/html) | sudo tar -xzvf drupal-11.x.tar.gz -C /var/www/html |
| 7 | Configurer les permissions | Terminal | Définir les droits d’accès pour le serveur web | sudo chown -R www-data:www-data /var/www/html/drupal sudo chmod -R 755 /var/www/html/drupal |
| 8 | Configurer Apache (ou Nginx) | Fichier de config - IDEM Ubuntu  Seul le chemin change  /etc/apache2/sites-available/ | Créer ou modifier un fichier de configuration  Ex. /etc/apache2/sites-available/drupal.conf) | Voir exemple de configuration ci-dessous |
| Activer le site | sudo a2ensite drupal.conf |
| Terminal | Recharger Apache | sudo systemctl reload apache2 |
| 9 | Configurer la base de données | Terminal | Utiliser les commandes spécifiques à MariaDB pour se  se connecter à la BDD MariaDB | **?** |
| Créer une base de données | CREATE DATABASE drupaldb; |
| Créer un utilisateur pour Drupal | CREATE USER 'drupaluser'@'localhost' IDENTIFIED BY 'drupalpass'; |
| Régler les permissions | GRANT ALL PRIVILEGES ON drupaldb.\* TO 'drupaluser'@'localhost'; |
| Valider les permissions | FLUSH PRIVILEGES; |
| 10 | Lancer l’installation via le navigateur | GUI  Ex. http://votre-ip/drupal | Observer les logs d’installation | Suivre l’assistant d’installation de Drupal |

Composant → Environnement natif de Drupal 11 sur Ubuntu → Checklist

| # | Action | Type et chemin | Détail | Commande / GUI / Fichier |
| --- | --- | --- | --- | --- |
| 5 | Télécharger Drupal 11 | Terminal  Site officiel de Drupal | Télécharger la version stable de Drupal 11 depuis drupal.org | wget <https://ftp.drupal.org/files/projects/drupal-11.x.tar.gz> |
|  | Vérifier la somme de contrôle du fichier | Terminal  Site officiel de Drupal | ? | ? |
| 6 | Extraire Drupal | Terminal | Extraire l’archive dans le répertoire de votre serveur web | tar -xzvf drupal-11.x.tar.gz -C /var/www/html |
| 7 | Configurer les permissions | Terminal | Définir les droits d’accès pour que le serveur web puisse écrire | sudo chown -R www-data:www-data /var/www/html/drupal sudo chmod -R 755 /var/www/html/drupal |
| 8 | Configurer Apache (ou Nginx) | Fichier de config | Créer ou modifier un fichier de configuration (ex. /etc/apache2/sites-available/drupal.conf) | Voir exemple ci-dessous |
|  | Activer le site | Terminal | ? | a2ensite drupal.conf |
|  | Recharger Apache | Terminal | ? | ? |
| 9 | Configurer la base de données MariaDB | Terminal / Script | Utiliser les commandes spécifiques à MariaDB pour se  se connecter à la BDD MariaDB | **?** |
| Créer une base de données | CREATE DATABASE drupaldb; |
| Créer un utilisateur pour Drupal | CREATE USER 'drupaluser'@'localhost' IDENTIFIED BY 'drupalpass'; |
| Régler les permissions | GRANT ALL PRIVILEGES ON drupaldb.\* TO 'drupaluser'@'localhost'; |
| Valider les permissions | FLUSH PRIVILEGES; |
| 10 | Lancer l’installation via le navigateur | GUI : Accéder à l’URL du site  Ex. `http://votre-ip/drupal` | Suivre les logs d’installation dans le dossier `/sites/default/files` | Suivre l’assistant d’installation de Drupal |

### 3. Exemple de Configuration Apache pour Drupal sur Ubuntu

```apache

<VirtualHost \*:80>

ServerName drupal.example.com

DocumentRoot /var/www/html/drupal

<Directory /var/www/html/drupal>

Options Indexes FollowSymLinks

AllowOverride All

Require all granted

</Directory>

ErrorLog ${APACHE\_LOG\_DIR}/drupal\_error.log

CustomLog ${APACHE\_LOG\_DIR}/drupal\_access.log combined

</VirtualHost>

Points de Divergence Debian / Ubuntu

* Emplacements de fichiers de configuration  
  La structure est très similaire sur Ubuntu et Debian pour Apache, PHP et MariaDB.  
  Toutefois, vérifiez que les chemins (ex. /etc/apache2/) et les commandes spécifiques (gestion de services avec systemctl ou service) soient bien ceux de votre version de Debian, car ils peuvent varier légèrement selon la version.
* Gestion des paquets  
  La commande apt fonctionne de manière identique sur Ubuntu et Debian, mais certains paquets ou versions pourraient différer selon la distribution et sa version.
* Interface graphique et administration  
  Sur Ubuntu, certaines distributions (comme Ubuntu Server) peuvent disposer de configurations légèrement différentes dans Netplan pour la gestion réseau, alors que Debian utilise généralement /etc/network/interfaces (sauf si vous avez opté pour une version récente avec Netplan). Adaptez-vous à l’outil de configuration réseau utilisé sur votre distribution.

Composant → Reverse Proxy / Load Balancer (ex. Nginx) → Checklist

| # | Action | Type | Détail | Commande / GUI / Fichier de config |
| --- | --- | --- | --- | --- |
| 1 | Installer Nginx sur une VM dédiée (Linux) | Terminal | Installer Nginx sur la VM qui jouera le rôle de reverse proxy / load balancer | sudo apt update && sudo apt install nginx -y |
|  | Vérifier | Terminal | S’assurer que l’installation c’est bien passée | systemctl status nginx |
| 2 | Configurer le fichier de site pour Nginx | Fichier de config | Créer ou modifier le fichier de configuration du site dans /etc/nginx/sites-available/ | Créer /etc/nginx/sites-available/drupal\_proxy  Voir exemple de config ci‑dessous |
| 3 | Configurer la répartition de charge | Fichier de config | Définir les upstreams pour répartir le trafic entre vos serveurs Drupal | Inclure la directive upstream dans le fichier config |
| 4 | Activer la configuration du site | Terminal / GUI | Créer un lien symbolique dans /etc/nginx/sites-enabled/ puis tester la config | sudo ln -s /etc/nginx/sites-available/drupal\_proxy /etc/nginx/sites-enabled/  sudo nginx -t |
|  | Relire les logs de Nginx | Terminal  /var/log/nginx/ | ? | ? |
| 5 | Recharger Nginx | Terminal | Appliquer la nouvelle configuration de Nginx | sudo systemctl reload nginx |
| 6 | Tester l’accès via le reverse proxy | Navigateur / Terminal | Vérifier l’accès au site Drupal via l’adresse du reverse proxy | Accéder à http://<IP-reverse-proxy> dans un navigateur ou via curl |
|  | Vérifier les logs d’accès | Terminal  /var/log/nginx/access.log | ? | ? |
| 7 | Documenter la configuration | Terminal / GUI  Note de synthèse | Consigner la configuration détaillée, adresses IP, ports, et règles appliquées | Créer un fichier /docs/Nginx-Drupal-Proxy.md |

Objectif

🡪 Assurer la répartition de la charge entre vos différentes instances Drupal(containerisées et/ou installées nativement).

🡪 Sécuriser et centraliser le point d’entrée du trafic WAN vers vos serveurs d’applications.

* Exemple de Configuration Nginx pour Drupal (reverse proxy)

🡪 /etc/nginx/sites-available/drupal\_proxy

nginx

Copier

# Définir les serveurs backend Drupal

upstream drupal\_backend {

server 10.0.0.10:80; # Serveur Drupal 1 (peut être Docker ou natif)

server 10.0.0.11:80; # Serveur Drupal 2 (exemple)

# Ajouter d'autres serveurs si nécessaire

}

server {

listen 80;

server\_name drupal.example.com;

# Rediriger tout le trafic vers le backend Drupal

location / {

proxy\_pass http://drupal\_backend;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

}

# Fichiers logs personnalisés

access\_log /var/log/nginx/drupal\_access.log;

error\_log /var/log/nginx/drupal\_error.log;

}

* Sécurité et SSL

configurer SSL/TLS (certificats Let's Encrypt ou autre) 🡪 écoute du port 443 et utilisation de ssl\_certificate et ssl\_certificate\_key dans le bloc serveur.

* Scalabilité et ajustements

ajuster la directive upstream avec des paramètres de répartition de charge (ex. least\_conn, ip\_hash, etc.) selon la charge et le comportement des serveurs Drupal.

* Monitoring

Intégrer des outils de monitoring et configurer la journalisation pour détecter rapidement tout problème de connectivité ou de charge.

**Etape 3 : le serveur dédié aux bases de données et au stockage**

Composant → Serveur Base de Données & Stockage (iSCSI, NFS, sauvegardes) → Checklist

| # | Action | Type et chemin | Détail | Commande / GUI / Fichier de config |
| --- | --- | --- | --- | --- |
| 1 | Créer une VM dédiée pour BD & Stockage | GUI | Créer une VM nommée, par exemple, VM-DB-Storage avec ressources adaptées | Paramétrer comme pour les autres VM |
| 2 | Installer le serveur de de BDD et vérifier | Terminal | Installer MariaDB (ou MySQL) | sudo apt update && sudo apt install mariadb-server -y  sudo systemctl status mariadb |
| 3 | Sécuriser l’installation de la base de données | Terminal / Script | Exécuter mysql\_secure\_installation pour définir le mot de passe root et désactiver l'accès anonyme | sudo mysql\_secure\_installation |
| 4 | Créer la base de données pour Drupal | Terminal / Script | Créer une base Drupal | mysql -u root -p OU mariadb -u root -p  CREATE DATABASE drupaldb; |
| Créer un utilisateur spécifique pour Drupal  Attribuer à l’utilisateur les privilèges de son accès | CREATE USER 'drupaluser'@'%' IDENTIFIED BY 'drupalpass';  GRANT ALL PRIVILEGES ON drupaldb.\* TO 'drupaluser'@'%'; |
| Valider les privilèges | FLUSH PRIVILEGES; |
| 5 | Installer NFS | Terminal / GUI | Installer le serveur NFS pour le partage de fichiers (ex. pour stocker les fichiers Drupal persistants) | sudo apt install nfs-kernel-server -y |
| Configurer NFS puis vérifier | Fichier de config  éditer `/etc/exports` | Créer un répertoire à partager  Ex. `/srv/nfs/drupal\_files` | exportfs -v  consulter les log 🡪 `/var/log/syslog` |
| 6 | Installer iSCSI | Terminal / GUI | Installer l’outil iSCSI target pour offrir un stockage block (utile pour la BD ou backups) | sudo apt install tgt -y |
| Configurer iSCSI | Fichier de config  `/etc/tgt/conf.d/drupal.conf | Définir les cibles iSCSI souhaitées | consulter les log 🡪 `/var/log/tgt/` ou `journalctl` |
| 7 | Créer les scripts de sauvegarde | Script  /usr/local/bin/backup\_drupal.sh | Mettre en place des scripts de sauvegarde Ex. via `mysqldump` pour la BD et `rsync` pour NFS | Voir exemples et détails dans les bulles d’information |
| Activer les scripts | Terminal | planifier l’exécution des scripts de sauvegardes dans la crontab (tâches cron) | crontab -e` |
| 8 | Tester la connectivité et la redondance | Terminal - ajuster l’étape précédente en cas d’erreur | Vérifier l’accès à la base de données depuis une autre VM et tester l’accès aux partages NFS/iSCSI | mysql -h <VM-DB-Storage\_IP> -u drupaluser -p  mount -t nfs <VM-DB-Storage\_IP>:/srv/nfs/drupal\_files /mnt/test |
| 9 | Documenter chaque action | Fichier  /docs/VM-DB-Storage.md | Être rigoureux sur les emplacements de chaque fichier, les configurations de réussite et les alertes signifiantes | Créer autant de sous dossier que nécessaire pour structurer  Inclure captures d’écran, logs, et commentaires |

Configuration iSCSI

L’iSCSI sert à proposer un stockage en mode bloc aux clients (par exemple, pour héberger des données sensibles ou même la base de données). Ici, nous allons utiliser le paquet tgt qui est souvent utilisé sur Debian/Ubuntu pour créer des cibles iSCSI.

* Installer l’outil iSCSI target

sudo apt update 🡪 sudo apt install tgt -y

* Préparer le stockage

🡪 Créez un fichier qui servira de disque virtuel (par exemple 10 Go) :

sudo dd if=/dev/zero of=/srv/iscsi/drupal\_disk.img bs=1M count=10240

🡪 Vérifiez que le dossier /srv/iscsi/ existe et créez-le s’il manque :

sudo mkdir -p /srv/iscsi 🡪 sudo chown -R root:root /srv/iscsi

* Configurer la cible iSCSI

🡪 Créez un fichier de configuration pour Drupal, par exemple /etc/tgt/conf.d/drupal.conf avec le contenu suivant :

<target iqn.2025-03.com.example:drupal.disk>

backing-store /srv/iscsi/drupal\_disk.img

initiator-address 10.0.0.0/24

# Très important : on peut ajouter d'autres règles, par exemple pour limiter l'accès à certaines IP

</target>

🡪 Unités de bulles explicatives à réaliser

- iqn.2025-03.com.example:drupal.disk : l’identifiant iSCSI (à réajuster avec notre domaine et date)

- backing-store : chemin vers le fichier créé

- initiator-address : limite l’accès aux IP du réseau (ici le sous-réseau 10.0.0.0/24 par exemple)

* Redémarrer et vérifier la cible iSCSI

🡪 Redémarrez le service tgt pour prendre en compte la nouvelle configuration 🡪 sudo systemctl restart tgt

🡪 Vérifiez que la cible est active 🡪 sudo tgtadm --mode target --op show 🡪 on devrait voir la cible définie avec son IQN et le backing store

* Tester depuis un client

🡪 Sur une VM cliente (par exemple, une VM Drupal), installer l’outil initiator iSCSI et lancez une découverte

sudo apt install open-iscsi -y 🡪 sudo iscsiadm -m Discovery -t sendtargets -p <IP-VM-DB-Storage> 🡪 Se connecter à la cible 🡪

sudo iscsiadm -m node -T iqn.2025-03.com.example:drupal.disk -p <IP-VM-DB-Storage> --login 🡪 le disque doit apparaît (par exemple, via lsblk)

Options avancées iscsi

* Authentification CHAP

L'authentification CHAP sert à renforcer la sécurité en s'assurant que seuls les initiateurs autorisés se connectent à votre cible iSCSI

🡪 Configuration côté cible en ajoutant les paramètres CHAP dans le fichier de configuration (ex. /etc/tgt/conf.d/drupal.conf)  
<target iqn.2025-03.com.example:drupal.disk>

backing-store /srv/iscsi/drupal\_disk.img

initiator-address 10.0.0.0/24

incominguser yourusername yourpassword

outgoinguser yourusername yourpassword

</target>

🡪 Unités de bulles explicatives à réaliser

- incominguser définit les identifiants que le client devra présenter lors de la connexion.

- outgoinguser peut être utilisé pour l’authentification réciproque, si nécessaire.

* Optimisation des performances et gestion des LUNs

Pour améliorer les performances, on peut ajuster quelques paramètres de plus

🡪 Utilisation de plusieurs LUNs  
On peut créer plusieurs cibles ou plusieurs LUNs. Par exemple, un LUN pour la base de données et un autre pour les fichiers Drupal

🡪 Paramètres de performance  
On peut aussi ajuster la taille des blocs (block size) ou configurer des options du noyau (si la distribution et la version du paquet le permettent)  
<target iqn.2025-03.com.example:drupal.disk>

backing-store /srv/iscsi/drupal\_disk.img,blocksize=4096

initiator-address 10.0.0.0/24

incominguser yourusername yourpassword

</target>

Attention ! La disponibilité de l'option blocksize dépend de la version de tgt et de la distribution.

🡪 Monitoring et logs iSCSI :  
On peut configurer le niveau de log pour tgt en modifiant la configuration globale (souvent dans /etc/default/tgt ou les options de démarrage du service) afin de surveiller les connexions et identifier d'éventuels goulots d'étranglement.

🡪 Intégration d'une Solution de Rotation des Sauvegardes

Pour éviter que les sauvegardes ne saturent l'espace disque, il est essentiel de mettre en place une rotation automatique.

* Utiliser une approche basée sur un script pour automatiser la gestion du risque de saturation par la sauvegarde

On peut modifier nos scripts de sauvegarde pour supprimer les anciennes sauvegardes datant de X jours avant d’en créer de nouvelles.

Par exemple, dans nos scripts de sauvegarde de la base de données (/usr/local/bin/backup\_drupal\_db.sh) on peut ajouter une commande pour supprimer les fichiers de plus de 7 jours

#!/bin/bash

# Variables

DB\_USER="drupaluser"

DB\_PASS="drupalpass"

DB\_NAME="drupaldb"

BACKUP\_DIR="/var/backups/drupal\_db"

DATE=$(date +%F)

RETENTION\_DAYS=7

# Créer le répertoire de sauvegarde s'il n'existe pas

mkdir -p $BACKUP\_DIR

# Supprimer les sauvegardes plus anciennes que RETENTION\_DAYS jours

find $BACKUP\_DIR -type f -name "\*.sql.gz" -mtime +$RETENTION\_DAYS -exec rm {} \;

# Effectuer la sauvegarde avec mysqldump et compresser

mysqldump -u $DB\_USER -p$DB\_PASS $DB\_NAME | gzip > $BACKUP\_DIR/drupaldb\_$DATE.sql.gz

* Utiliser logrotate pour les sauvegardes

Bien que logrotate soit principalement utilisé pour la rotation des logs, vous pouvez adapter une configuration similaire pour vos sauvegardes. Créez un fichier de configuration, par exemple /etc/logrotate.d/drupal\_backups

/var/backups/drupal\_db/drupaldb\_\*.sql.gz {

daily

missingok

rotate 7

compress

delaycompress

notifempty

create 640 root adm

}

🡪 Unités de bulles explicatives à réaliser

- daily : Rotation quotidienne

- rotate 7 : Conserver 7 sauvegardes

- compress et delaycompress : Compresser les fichiers après un jour. Utile pour les sauvegardes récentes si vous préférez un accès rapide.

- notifempty : Ne pas effectuer de rotation si le fichier est vide

* Planification via Cron

On s’assure que nos scripts sont exécutés via cron, comme décrit précédemment, et que le système de rotation (que ce soit intégré dans le script ou via logrotate) est testé pour vérifier que l'espace disque est bien géré sur le long terme.

Mise en place des scripts de sauvegarde automatisées

On doit absolument sauvegarder la BDD et les fichiers critiques (par exemple, le contenu Drupal) pour restaurer rapidement en cas de problème

* Sauvegarde de la base de données avec mysqldump

🡪 Créez un script, par exemple /usr/local/bin/backup\_drupal\_db.sh

#!/bin/bash

# Variables

DB\_USER="drupaluser"

DB\_PASS="drupalpass"

DB\_NAME="drupaldb"

BACKUP\_DIR="/var/backups/drupal\_db"

DATE=$(date +%F)

# Créer le répertoire de sauvegarde s'il n'existe pas

mkdir -p $BACKUP\_DIR

# Effectuer la sauvegarde avec mysqldump

mysqldump -u $DB\_USER -p$DB\_PASS $DB\_NAME > $BACKUP\_DIR/drupaldb\_$DATE.sql

# Optionnel : compresser la sauvegarde

gzip $BACKUP\_DIR/drupaldb\_$DATE.sql

🡪 Rendre le script exécutable en lui accordant les permission d’exécution (par défaut ces permissions sont accordées au système au niveau global de root)

sudo chmod +x /usr/local/bin/backup\_drupal\_db.sh

Testez manuellement le script pour vérifier qu'il crée bien le fichier de sauvegarde dans /var/backups/drupal\_db

* Sauvegarde des fichiers Drupal avec rsync

🡪 Créez un script, par exemple /usr/local/bin/backup\_drupal\_files.sh

#!/bin/bash

# Variables

DRUPAL\_DIR="/var/www/html/drupal"

BACKUP\_DIR="/var/backups/drupal\_files"

DATE=$(date +%F)

# Créer le répertoire de sauvegarde s'il n'existe pas

mkdir -p $BACKUP\_DIR

# Sauvegarder avec rsync

rsync -av --delete $DRUPAL\_DIR $BACKUP\_DIR/drupal\_$DATE

🡪 Rendre le script exécutable

sudo chmod +x /usr/local/bin/backup\_drupal\_files.sh

Testez le script pour confirmer que les fichiers Drupal sont correctement synchronisés dans le répertoire de sauvegarde

* Automatisation avec Cron

🡪 Éditer la crontab pour exécuter ces scripts à une fréquence régulière (par exemple, quotidiennement)

sudo crontab -e

🡪 Ajoutez les lignes suivantes

# Sauvegarde de la base de données tous les jours à 2h du matin

0 2 \* \* \* /usr/local/bin/backup\_drupal\_db.sh

# Sauvegarde des fichiers Drupal tous les jours à 3h du matin

0 3 \* \* \* /usr/local/bin/backup\_drupal\_files.sh

* Vérification et Documentation

🡪 Vérifier les logs après exécution pour s’assurer que les sauvegardes s’effectuent correctement  
/var/backups/drupal\_db et /var/backups/drupal\_files

Documentation (par exemple dans /docs/DB-Storage-Configuration.md)  
Noter les chemins des sauvegardes, la fréquence d’exécution et toutes les spécificités (compression, rotation, etc)

Ne pas oublier de noter également comment restaurer la sauvegarde 🡪 exemple 🡪

gunzip < drupaldb\_<DATE>.sql.gz | mysql -u drupaluser -p drupaldb (revoir ça en détail avec des bulles explicatives )

Composant → Monitoring & Gestion Centralisée → Checklist

| # | Action | Type et chemin | Détail | Commande / GUI / Fichier |
| --- | --- | --- | --- | --- |
| 1 | Créer une VM dédiée et finir sa configuration réseau/pare-feu | GUI | VM nommée VM‑Monitoring, Ubuntu Server 24.04, 4 vCPU, 8 Go RAM, 50 Go HDD | VirtualBox / Proxmox  sudo apt update |
| 2 | Installer Prometheus | Terminal | Collecte métriques | sudo apt install prometheus prometheus-node-exporter -y |
| ``` | Vérifier l’installation dans les fichiers de configuration et log | Fichier config | S’assurer qu’il n’y a aucun problème gris | /etc/prometheus/prometheus.yml /var/log/prometheus/prometheus.log |
| 3 | Configurer Prometheus puis  Vérifier son configuration | Fichier config  /etc/prometheus/prometheus.yml | Ajouter toutes les cibles (VM IPs) dans scrape\_configs | sudo systemctl status prometheus |
| 4 | Installer Grafana | Terminal | Visualisation des dashboards | sudo apt install grafana -y  sudo systemctl enable --now grafana-server |
| ``` | Vérifier l’installation dans les fichiers de configuration et log | Fichier config | S’assurer qu’il n’y a aucun problème gris | /etc/grafana/grafana.ini /var/log/grafana/grafana.log |
| 5 | Configurer datasource Grafana | GUI → http://<IP‑Monitoring>:3000 (admin/admin) | Créer des dashboards basiques pour tester  (Node Exporter Full) | Grafana UI → Add Data Source → Prometheus |
| 6 | Installer Filebeat sur Monitoring | Terminal | Aggrégation de logs | sudo apt install filebeat -y |
|  | Vérifier l’installation dans les fichiers de configuration et log | Fichier config | S’assurer qu’il n’y a aucun problème gris | /etc/filebeat/filebeat.yml /var/log/filebeat/filebeat |
| 7 | Installer Elasticsearch | Terminal | Stockage logs | sudo apt install elasticsearch -y  sudo systemctl enable --now elasticsearch |
| ``` | Vérifier l’installation dans les fichiers de configuration et log | Fichier config | S’assurer qu’il n’y a aucun problème gris | /etc/elasticsearch/elasticsearch.yml /var/log/elasticsearch |
| 8 | Installer Kibana | Terminal | Visualisation logs | sudo apt install kibana -y  sudo systemctl enable --now kibana |
| ``` | Vérifier l’installation dans les fichiers de configuration et log | Fichier config | S’assurer qu’il n’y a aucun problème gris | /etc/kibana/kibana.yml /var/log/kibana |
| 9 | Configurer Filebeat  !!!!! SUR CHAQUE VM !!!!! | Fichier config  /etc/filebeat/filebeat.yml | Pointer vers Elasticsearch central 🡪 output.elasticsearch.hosts: ["<IP‑Monitoring>:9200"] | sudo filebeat test output |
| 10 | Configurer Alertmanager | Terminal / Fichier config  Edit /etc/alertmanager/config.yml | Alerting via Prometheus  /var/log/alertmanager | sudo apt install prometheus-alertmanager -y |
|  | Créer alert rules Prometheus | Fichier config  Edit /etc/prometheus/rules.yml | Ex : CPU > 85% 5m | Reload config via API |
| 12 | Créer le script de sauvegarde | /usr/local/bin/backup\_monitoring.sh | Sauvegarde Elasticsearch indices et Prometheus TSDB | Logs de cron 🡪 /var/log/syslog |
| 13 | Tester l’accès & les dashboards | Navigateur | Vérifier métriques et logs | Grafana: http://<IP‑Monitoring>:3000 Kibana: http://<IP‑Monitoring>:5601 |
| 14 | Documenter & valider | /docs/Centralized-Monitoring.md | Création VM ; Setup Réseau & Firewall ; Déploiement logiciel ; accès, procédures de restauration | Être capable de tout reproduire  Avoir de la redondance dans les notes |

🡪 Objectifs

1- Collecter métriques (CPU, RAM, I/O, réseau) de toutes les VM

2- Centraliser les logs système et applicatifs

3- Fournir des dashboards en temps‑réel, des alertes et des historiques

4- Assurer la pérennité des données grâce aux sauvegardes automatisées

🡪 Script de sauvegarde (/usr/local/bin/backup\_monitoring.sh)

#!/bin/bash

BACKUP\_DIR="/var/backups/monitoring"

DATE=$(date +%F)

mkdir -p $BACKUP\_DIR

# Backup Prometheus TSDB

cp -r /var/lib/prometheus/data $BACKUP\_DIR/prometheus\_$DATE

# Snapshot Elasticsearch indices

curl -X PUT "localhost:9200/\_snapshot/backup\_repo/snapshot\_$DATE?wait\_for\_completion=true"

🡪 Cron (via sudo crontab -e) :

0 4 \* \* \* /usr/local/bin/backup\_monitoring.sh >> /var/log/backup\_monitoring.log 2>&1

Alerting Avancé avec Prometheus et Alertmanager

Définition d’Alert Rules dans Prometheus

* Fichier de règles :

Créez ou éditez un fichier, par exemple /etc/prometheus/rules/alert\_rules.yml, pour définir vos alertes.

🡪 Exemple d’alerte sur l’utilisation CPU

groups:

- name: server\_alerts

rules:

- alert: HighCPUUsage

expr: avg by(instance)(rate(node\_cpu\_seconds\_total{mode!="idle"}[5m])) > 0.85

for: 5m

labels:

severity: critical

annotations:

summary: "CPU usage high on instance {{ $labels.instance }}"

description: "CPU usage is above 85% for more than 5 minutes."

🡪 Unités de bulles explicatives à réaliser

- expr : Expression PromQL qui calcule l’utilisation moyenne du CPU.

- for : Durée pendant laquelle la condition doit être vraie pour déclencher l’alerte.

- labels et annotations : Informations supplémentaires pour le contexte de l’alerte.

* Configuration d’Alertmanager

global:

resolve\_timeout: 5m

route:

group\_by: ['alertname', 'instance']

group\_wait: 30s

group\_interval: 5m

repeat\_interval: 1h

receiver: 'email-notifications'

receivers:

- name: 'email-notifications'

email\_configs:

- to: 'admin@example.com'

from: 'alertmanager@example.com'

smarthost: 'smtp.example.com:587'

auth\_username: 'alertmanager@example.com'

auth\_identity: 'alertmanager@example.com'

auth\_password: 'votre\_mot\_de\_passe'

🡪 Unités de bulles explicatives à réaliser

- route : Permet de regrouper et d’acheminer les alertes vers le receiver défini

- receivers : Ici, un exemple d’envoi d’e-mail. Vous pouvez ajouter des configurations pour Slack, PagerDuty, etc

🡪 Test et Validation

- Reload de Prometheus et Alertmanager après modification

curl -X POST http://<IP-monitoring>:9090/-/reload

curl -X POST http://<IP-monitoring>:9093/-/reload

🡪 Simulation d’alerte :  
Utiliser la console Prometheus pour visualiser et simuler l’apparition des alertes

vérifier qu’elles sont correctement routées par Alertmanager

* Custom Dashboards dans Grafana

🡪 Création et Personnalisation

Accès à l’interface Grafana 🡪 http://<IP-monitoring>:3000 et connectez-vous (par défaut, admin/admin)

Ajout d’une Data Source 🡪 Vérifiez que Prometheus est configuré comme source de données

Création d’un Dashboard 🡪 Cliquez sur "Create" → "Dashboard", puis "Add new panel". Vous pouvez personnaliser chaque panneau

Type de graphique 🡪 Choisissez entre Graph, Singlestat, Table, etc

Personnalisation des axes, couleurs, légendes 🡪 Utilisez l'éditeur visuel pour adapter la présentation

🡪 Requête PromQL 🡪 Exemple : pour afficher l'utilisation CPU

avg(rate(node\_cpu\_seconds\_total{mode!="idle"}[5m])) by (instance)

🡪 Nommer le dashboard (ex. "Vue Globale Serveurs") et le sauvegarder pour une consultation continue

* Variables, alerting et droits d’accès

🡪 Variables  
Créez des variables (ex. $instance) afin de filtrer dynamiquement les données dans les panneaux

🡪 Alerting intégré aux Panels  
Grafana peut ajouter des alertes directement sur certains panneaux. Il faut configurer les seuils critiques de leurs déclenchements

🡪 Partage et permissions  
Configurer les rôles et les autorisations d’accès du dashboard

* Validation et Documentation

🡪 Test de navigation  
Parcourez les dashboards, testez l’interactivité (filtres, drill-down) et ajustez la configuration au besoin

Noter la configuration et l’objectif de chaque dashboard dans un document de synthèse, par exemple dans /docs/Grafana-Dashboards.md

* Restauration de Prometheus TSDB

Faire la sauvegarde préalable. Comme indiqué dans le script de sauvegarde, les données Prometheus sont copiées dans un répertoire dédié

🡪 Arrêtez Prometheus

sudo systemctl stop prometheus

🡪 Copiez les données sauvegardées dans le répertoire de données de Prometheus

cp -r /var/backups/monitoring/prometheus\_<DATE>/\* /var/lib/prometheus/data/

🡪 Redémarrer Prometheus

sudo systemctl start Prometheus

* Restauration d’Elasticsearch (Snapshots)

Elasticsearch peut restaurer un snapshot via son API

🡪 Vérifier le snapshot

curl -X GET "localhost:9200/\_snapshot/backup\_repo/snapshot\_<DATE>?pretty"

🡪 Restaurer le snapshot

curl -X POST "localhost:9200/\_snapshot/backup\_repo/snapshot\_<DATE>/\_restore?wait\_for\_completion=true"

🡪 Valider  
Contrôlez que les indices sont restaurés avec succès via l’API Elasticsearch ou Kibana

* Documentation et Tests

🡪 Rédiger un manuel de restauration  
Documenter précisément chaque étape dans un fichier, avec les commandes et les prérequis, par exemple /docs/Restoration-Procedures.md

🡪 Test de restauration périodique  
Prévoir des tests de restauration pour valider que la procédure fonctionne en conditions réelles, en notant les éventuels ajustements à faire

Troubleshooting

Pour chaque problème rencontré, ajoutez une entrée structurée :

**Grille de Troubleshooting Enrichie (GTE)**

| Projet | Composant | Étape | Raison du Contexte | Observations Préalables | Résultats Bruts | Analyses  Des Tests | Correctifs Appliqués | Points Non Résolus | Synthèse |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (ID ou Nom du Projet, ex. "Drupal-HA-Infra") | (Ex. "VM-DB", "Nginx", "Grafana") | (Ex. "Test de connectivité", "Chargement dashboard") | Décrire le contexte qui engendre le test (ex. "Déploiement initial, mise à jour de config, comportement inattendu") | Notes sur l’état initial, hypothèses, environnement de test | Sortie brute de la commande, logs d’erreur, captures d’écran | Interprétation des résultats, identification des anomalies | Liste des commandes/modifications appliquées pour corriger | a. Description b. Bilan de Connaissance de la Raison (BCR) c. Recommandations d. Liens utiles e. Planification f. Responsabilités g. Synthèse des actions en attente | Récapitulatif des points et état final du troubleshooting |

**Un exemple d'enregistrement**

| Projet | Composant | Étape | Raison du Contexte | Observations Préalables | Résultats Bruts | Analyses des Tests | Correctifs Appliqués | Points Non Résolus | Synthèse |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Drupal-HA-Infra | VM-DB | Test de connectivité | Après déploiement initial, connexion BD intermittente | Connexion intermittente, pings variables | ping 10.0.0.10: 0% perte vs. 30% perte | Potentiel problème de routage ou surcharge réseau | Ajustement des routes, vérification des pare-feu, redémarrage du service réseau | a. Description : Intermittence malgré la configuration correcte b. BCR : Peut-être une surcharge sur le réseau interne c. Recommandations : Monitorer avec iPerf d. Liens : [Documentation réseau](https://exemple.com) e. Planification : Test en heure creuse f. Responsabilités : Équipe Réseau g. Synthèse : Problème à suivre sur une semaine | Connectivité stabilisée après redémarrage ; surveiller l’évolution |

🡪 Unités de bulles explicatives à réaliser

1. id\_site : en mode de gestion calculé, cet id unique est commun à tous les enregistrements des BDD de l’infrastructure
2. projet : identification du projet concerné (utile quand l’infrastructure commencera à mutualiser plusieurs projets)
3. composant : Le nom et l’identifiant du composant concerné (VM, serveur, service, etc.)
4. étape : L’étape du processus ou du test spécifique (ex. connectivité, performance, sécurité, etc.)
5. id\_trouble : cet id est générique au sein d’une liste fine qui recense le scope des problèmes en produisant des unités (tags veille technologique)
6. nom : appellation générique littéraire et expressive basée sur l’arborescence de l’ id\_trouble pour nourrir les moteurs de recherche
7. numéro : au format suivant : truble\_YYYY-MM-DD\_incrementedNumber
8. raison du Contexte : pourquoi ce test ou ce rapport est-il réalisé ? (ex. "Déploiement initial", "Mise à jour de configuration", "Incident de production", etc.)
9. Observations Préalables : Noter les observations initiales, hypothèses, et conditions avant de lancer le test
10. résultats Bruts : Pour chaque Action à lister, Enregistrer directement la sortie des tests, logs, captures d’écran ou données collectées
11. analyses des Tests : Interpréter les résultats bruts et identifier la cause racine du problème
12. correctifs Appliqués : Documenter précisément les actions correctives (commandes exécutées, fichiers modifiés, etc.)
13. points Non Résolus : Structurez en sous-parties pour détailler tout ce qui reste à investiguer ou corriger
14. synthèse : Récapituler l’état final, l’impact sur l’infrastructure et les recommandations pour le suivi

Dictionnaire des entités du troubleshooting page 1/2

1- Table « site »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1.1- cluster | 1.2- node | 1.3- operatingSystem | 1.4- virtualisation | 1.5- site\_description | 1.6- siteDates | 1.7- id\_site (unique) |

Nb: dates = planning, … We shall find here any little ligature and fork, as nested virtualization makes these realities complex, with mandatory fill up needs before new records

2- Table « project »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2.1- jobs | 2.2- works | 2.3- tasks | 2.4- targetMesures | 2.5- situations | 2.6- projectDates | 2.7- id\_project (unique) |

Nb: dates = planning, … targetMesures = we shall be able to find in here the project management conduction needed previsions and updates to build the components and steps

3- Table « component »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3.1- macroState | 3.2- mainStructure | 3.3- rôle | 3.4- functions | 3.5- activeTechnology | 3.6- componentDates | 3.7- id\_tech\* (unique) |

Nb: dates = planning, … We shall always find the architecture deployment, development and maintenance targets BUT ALSO STEPS in “targetMesures”

4- Table « step »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4.1- id\_tasks | 4.2- stepUpperTrees | 4.3- stepDownTrees | 4.4- wikiStep | 4.5- stepTechWatch | 4.6- stepDates | 4.7- id\_step (unique) |

Nb: dates = planning, … Nb: wikiStep = active and passive alerts and notifications to remember closure needs and to forbid actions until the algorithm step needs are filled

5- Table « trouble »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5.1- id\_step | 5.2- troubleUpperTrees | 5.3- troubleDownTrees | 5.4- wikiTrouble | 5.5- troubleTechWatch | 5.6- troubleDates | 5.7- id\_trouble (unique) |

Nb: dates = planning, … Nb: wikiTrouble = active and passive alerts and notifications to remember closure needs and to forbid actions until the algorithm step needs are filled

6- Table « names »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6.1- findThisTarget | 6.2- namesUpperTrees | 6.3- nameDownTrees | 6.4- wikiNames | 6.5- standByChange | 6.6- nameDates | 6.7- id\_names (unique) |

Nb: dates = planning, … A name is not bound to be unique. We try so, but we are also feeding the engine. A name shall always appear with steps and components differences

Nb: findThisTarget is the relation table where any trees can be fallowed to understand the unicity of one named target record. The wikiNames table has no vocation for this. It groups together all processing data liked to the planning and starts from there any notes, thoughts to remember, messages for further user, etc. for this raison, $wiki\* is not the default place you will find solutions or trouble definitions. We will always try to put each precise information in the most appropriated table. **🡪 Nb: “6.2- names”** is the place to record official translations

7- Table « numbers »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 7.1- pattern | 7.2- patternUpperTrees | 7.3- patternDownTrees | 7.4- wikiNumber | 7.5- patternTechWatch | 7.6- numberDates | 7.7- id\_numbers (un\*) |

Nb: dates = planning, … / We Build dynamically useful troubleshooting patterns to prepare steps by taking profit of the older records and we give direct references for the simulation issues. We can create empty templates for technological watch to do and save them to schedule notifications. Empty numberPatterns can come after the watch, before simulations. So, we try to feed solutions even before we meet them troubles. This is also a powerful migration method for other troubleshooting databases and a renovation solution for our own memory.

Dictionnaire des entités du troubleshooting page 2/2

8- Table « raison »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 8.1- findTheStep | 8.2- findThetrouble | 8.3- findTheRecord | 8.4- wikiRaison | 8.5- raisonTechWatch | 8.6- raisonDates | 8.7- id\_raison (unique) |

Nb: dates = planning, … Raisons are the start points of the built of identities. We try the catch the schemes of the will until we can end with the key names that open processes. Views can be dynamic but it should be better for the engine performance to rely on dedicated tables already set to produce dynamically specific views. Then, when we will need a singular one, any legitim technician will be able to create the views with very simple requests. So, “raison” should already filter those tables by step, trouble, problem, action AND RECORDS. Finaly, we need a “record pass” that will work as a gate authorization. We declare first what we want to do by creating or integrating a “raison” and we receive a record number. Then, we use that code to work on the database. One raison declaration is one request. That will help us to prepare and focus on what is important and not use the database for fun. We will use exercises databases with wrong data to practice.

9- Table « problem »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 9.1- findTheRecord | 9.2- problemUpperTrees | 9.3- problemDownTrees | 9.4- wikiProblem | 9.5- problemTechWatch | 9.6- problemDates | 9.7- id\_problèm (un\*) |

Nb: dates = planning, … Nb: wikiProblem = active and passive alerts and notifications to remember closures needs and to forbid actions until the problem wiki needs are filled. A problem is an issue when you apply an official solution to a well identified trouble. This could never be the same definition as the ticketing processes one because we are using daily the devices ourselves. If an official solution doesn’t work redundantly, the solution is wrong ! We need to build a tree for the referred raisons, troubles, and steps to declare a salvation action that will help us with the planning, the notifications and the alerts. Then, we need to create an fork to keep the same global configuration in an experimental lab, and eradicate the issue with persistent new official solution. Finaly we will update the database with the new official solution. For that raison, a problem will always be the simple fact that we encounter an issue while an official solution has just been applied. We must remember that we try to provide solutions for troubles that we haven’t even met yet. Some of them can be mistaken, some of them can become deprecated with time and provoke errors messages

10- Table « actions »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10.1- findTheRecord | 10.2- actionUpperTrees | 10.3- actionDownTrees | 10.4- wikiAction | 10.5- actionOwnHistory | 10.6- actionDates | 10.7- id\_action (unique) |

Nb: dates = planning, … Nb: wikiAction = active and passive alerts and notifications to remember closure needs and to forbid actions until the algorithm step needs are filled

11- Table « analyses »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11.1- findItsTopicRecord | 11.2- analyseUpperTrees | 11.3- analyseDownTrees | 11.4- wikiAnalyse | 11.5- subjectTechWatch | 11.6- analyseDates | 11.7- id\_analyse (u\*) |

Nb: dates = planning, … Nb: wikiAnalyse = active and passive alerts and notifications to remember closures needs and to forbid actions until the algorithm step needs are filled

subjectTechWatch is not giving concrete solutions for our issues. It is opening usefully knowledge. The topic itself and the main points it will cover will be related to the concrete solutions we actually have.

12- Table « solutions »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 12.1- findItsSolvedCases | 12.2- solutionUpperTree | 12.3- solutionDetails | 12.4- wikiSolution | 12.5- solutionTechWatch | 12.6- solutionDates | 12.7- id\_solution (u\*) |

Nb: dates = planning, … Nb: wikiSolution = active and passive alerts and notifications to remember closure needs and to forbid actions until the algorithm step needs are filled

13- Table « unsolved »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 13.1- | 13.2- unsolvedUpperTrees | 13.3- unsolvedDownTrees | 13.4- wikiUnsolved | 13.5- unsolvedTechWatch | 13.6- unsolvedDates | 13.7- id\_unsolved (u\*) |

Nb: dates = planning, … Nb: wikiStillTrouble = active and passive alerts and notifications to remember closure needs and to forbid actions until the algorithm step needs are filled

14- Table « protocol\_retained »

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 14.1- editProtocols | 14.2- protocolUpperTrees | 14.3- protocolDownTrees | 14.4- wikiProtocol | 14.5- protocol Files | 14.6- publicationDates | 14.7- id\_protocol (u\*) |

Nb: dates = planning, … Nb: wikiProtocol = active and passive alerts and notifications to remember closures needs and to forbid actions until the algorithm step needs are filled

Dernière étape : Documentation Globale, Tests d’Intégration et Validation Finale

Composant → Checklist Finale

| # | Action | Type | Détail | Commande / GUI / Fichier |
| --- | --- | --- | --- | --- |
| 1 | Centraliser la documentation globale | Documentation  /docs sur un serveur dédié de l’infra | Rassembler tous les documents (notes de synthèse, guides d’installation, procédures de restauration, configurations) | Connecté le serveur à un repository Git |
| 2 | Réaliser des tests d’intégration | /docs/tests\_integration.md | Tester la connectivité entre chaque composant (VM, réseaux, reverse proxy, BD, monitoring, stockage) | scripts de test (ping, traceroute, curl) |
| 3 | Valider le fonctionnement de chaque service | Captures d’écran, rapports d’erreur, etc.  interfaces web et logs | Vérifier le fonctionnement des applications (Drupal, Grafana, Prometheus, Elasticsearch, Alertmanager, Filebeat, etc.) | Checklist évoluée |
| 4 | Tester la redondance et la haute disponibilité | /docs/tests\_ha.md | Simuler la défaillance de certains composants (Exemple : arrêt d’une VM, coupure d’un lien, …) et observer la réaction du système | Stopper temporairement une VM pour vérifier le failover |
| 5 | Vérifier la sécurité globale de l’infrastructure | Audit de sécurité | Exécuter des scans de vulnérabilité et vérifier la configuration des pare-feu, accès restreints, certificats SSL/TLS | Utiliser des outils comme nmap, Nessus, ou OpenVAS |
| 6 | Valider les sauvegardes et la restauration | Tests et Simulations  /docs/Restoration-Procedures.md | Restaurer de manière simulée la base de données, les fichiers Drupal et les indices de monitoring à partir des sauvegardes | Faire des restaurations manuelles :  🡪 mysqldump, snapshot ES, … |
| 7 | Rassembler le fichier final (dont guide de déploiement) | Valider le guide avec l’équipe technique | Créer un guide complet qui servira de référence pour la mise en production, la maintenance et les évolutions futures | Compiler tous les documents dans un guide (ex. PDF, wiki interne) |
| 8 | Préparer la migration vers la production | Planification : Inclure le calendrier, les étapes clés, les points de contrôle | Établir le plan de migration (test sur environnement de staging, validation avec l’équipe, plan de basculement) | Documenter le plan de migration dans /docs/Migration-Plan.md |
| 9 | Revue finale et validation par les parties prenantes | Réunion / Validation  Noter chaque détail et perfectionner | Organiser une revue de l’infrastructure avec les parties prenantes (administrateurs, techniciens, responsables sécurité) | Avancer sur le terrain des livraisons |

🡪 Objectifs

- Centraliser la documentation de chaque composant et procédure

- Valider l’intégration de tous les services de l’infrastructure

- Tester le fonctionnement global en condition quasi‑production

- Préparer la mise en production en identifiant et en corrigeant les éventuels points faibles

🡪 Tests réguliers  
Même après la validation initiale il faut planifier des tests réguliers (automatisés si possible) pour surveiller la santé de l’infrastructure

🡪 Plan de migration et de retour en arrière  
Toujours avoir un plan de reprise en cas de défaillance lors de la migration vers la production (procédures de restauration et rollback plan)

1. Utilisation de Prometheus et Grafana :

* Prometheus est principalement conçu pour collecter et stocker des métriques, mais il peut être configuré pour récupérer des informations depuis une base de données externe en utilisant des *exporters* personnalisés ou des scrapers. Cependant, Prometheus n'est pas directement conçu pour récupérer des données structurées provenant de bases de données relationnelles comme MariaDB. Cela dit, il existe des moyens détournés pour y parvenir, comme l'utilisation d'un exporter SQL ou de connecteurs spécifiques.
* Grafana, par contre, est bien plus flexible pour ce type de projet. Il te permet de connecter différents types de sources de données, y compris MariaDB, via des plugins ou des connecteurs natifs. Tu peux ainsi créer des dashboards dynamiques où les données changent en fonction des filtres appliqués à ta base de données.

2. Intégration avec MariaDB (ta base de données) :

* Grafana propose un connecteur MariaDB natif via son plugin MySQL, car MariaDB est une fork de MySQL. En connectant Grafana à ta base de données MariaDB, tu peux définir des requêtes SQL qui seront exécutées pour récupérer les données. Ces requêtes peuvent être paramétrées dynamiquement via des variables que tu définis dans Grafana. Cela te permet de filtrer les données en temps réel selon les besoins (par exemple, par date, par type de problème, etc.).
* Cela signifie que tu peux avoir des graphiques et des tableaux de bord qui affichent des informations personnalisées, comme les logs de troubleshooting ou des métriques extraites directement de ta base de données, et ces informations s’ajusteront en fonction des filtres que tu appliques dans Grafana.

3. Création de dashboards dynamiques :

* Tu peux créer des variables dans Grafana qui serviront de filtres dynamiques. Ces variables peuvent être liées à des champs dans ta base de données MariaDB. Par exemple, tu peux avoir une variable "Type de problème" qui se remplit automatiquement avec les types de problèmes que tu as enregistrés dans ta base de données.
* Ensuite, tu peux lier cette variable à une requête SQL dans Grafana pour qu’en sélectionnant un type de problème, les graphiques ou les tableaux de données s'ajustent en temps réel pour ne montrer que les résultats pertinents à cette sélection.

4. Processus d'intégration :

1. Connecter Grafana à MariaDB : Dans Grafana, tu vas dans les paramètres des sources de données et choisis MariaDB/MySQL. Tu y entreras les informations de connexion à ta base de données (hôte, utilisateur, mot de passe, etc.).
2. Créer des requêtes SQL personnalisées : Une fois la connexion établie, tu peux créer des requêtes SQL personnalisées qui récupèrent les données de ta base de données en fonction de ce que tu souhaites afficher dans ton dashboard.
3. Ajouter des variables dynamiques : Tu peux ensuite ajouter des variables de filtrage pour permettre aux utilisateurs de filtrer les résultats directement depuis l'interface de Grafana.
4. Création de panels : Enfin, tu crées des panels dans ton dashboard qui afficheront les données récupérées en fonction des requêtes et des filtres appliqués.

5. Exemple pratique :

* Supposons que tu veux afficher une liste des erreurs de logs en fonction de la date. Tu peux créer une variable pour sélectionner les dates (par exemple, un calendrier ou une liste déroulante), et ta requête SQL dans Grafana pourrait ressembler à ça :

sql

Copier

SELECT error\_message, count(\*) AS occurrences

FROM troubleshooting\_logs

WHERE log\_date BETWEEN $\_\_from AND $\_\_to

GROUP BY error\_message

* Le \_\_$from et \_\_$to sont des variables intégrées dans Grafana qui représenteront les dates sélectionnées par l'utilisateur. Cela permet de filtrer dynamiquement les données de ta base MariaDB en fonction de la période choisie.

**1. Comprendre les logs et leur gestion actuelle :**

* Effectivement, les logs des processus système (comme ceux de **syslog**, **apache**, **nginx**, etc.) sont souvent enregistrés dans des fichiers journaux, et selon les configurations de rotation (comme logrotate sur Linux), ces fichiers sont régulièrement archivés ou effacés pour éviter qu'ils ne consomment trop d'espace disque.
* Si tu ne fais rien de spécifique pour sauvegarder certains logs, une fois qu'ils sont archivés ou effacés, tu perds effectivement les données, sauf si tu as un système de sauvegarde en place.

**2. Sauvegarde automatisée des logs dans une base de données :**

Pour automatiser la sauvegarde de certains logs dans un format structuré et bien défini, voici les étapes possibles :

* **Étape 1 : Identifier les logs à sauvegarder** Tu peux définir un ensemble de critères pour identifier quels logs spécifiques tu souhaites sauvegarder. Par exemple, certains logs de processus peuvent contenir des erreurs critiques, des avertissements, ou des événements particuliers qui méritent d'être archivés.
* **Étape 2 : Capture des logs en temps réel** Pour capturer ces logs en temps réel et les structurer dans une base de données, tu pourrais utiliser un outil comme **Filebeat** (qui fait partie de la suite Elastic) ou un script personnalisé. Ces outils peuvent lire les fichiers de logs en temps réel et envoyer les événements à un autre système, par exemple une base de données.
* **Filebeat** peut être configuré pour surveiller des fichiers logs et envoyer les événements capturés dans des systèmes comme **Elasticsearch** ou directement dans une base de données SQL (par exemple MariaDB) via des modules d'output ou un processus intermédiaire comme **Logstash**.
* **Étape 3 : Structurer et insérer les logs dans la base de données** Une fois que tu as capturé un log, tu dois pouvoir le traiter et l'insérer dans ta base de données. Voici un exemple de la façon dont tu pourrais structurer ta base de données pour un cas de troubleshooting :
  + **Table des logs** :

sql

Copier

CREATE TABLE troubleshooting\_logs (

id INT AUTO\_INCREMENT PRIMARY KEY,

timestamp DATETIME NOT NULL,

log\_level VARCHAR(10),

process\_name VARCHAR(100),

message TEXT,

error\_code VARCHAR(50),

other\_relevant\_data TEXT

);

* + - Tu pourrais avoir des tables spécifiques pour chaque type de log, ou une table générale comme celle-ci où chaque champ est défini pour collecter les informations pertinentes (par exemple, le niveau de log, le message, l'erreur, etc.).
  + **Automatisation avec des scripts** : Si tu préfères ne pas utiliser un outil comme Filebeat, tu peux écrire un **script Bash** ou **Python** qui surveille les fichiers de log, filtre les entrées pertinentes (par exemple, les erreurs ou les événements critiques), puis insère ces données dans ta base de données en temps réel.
  + Exemple simple en Python (en utilisant MySQLdb ou PyMySQL pour MariaDB) :

python

Copier

import time

import mysql.connector

# Connexion à la base de données

db = mysql.connector.connect(

host="localhost",

user="root",

password="password",

database="troubleshooting\_db"

)

cursor = db.cursor()

# Fonction pour lire les logs et les insérer

def process\_logs():

with open("/var/log/syslog", "r") as log\_file:

for line in log\_file:

if "ERROR" in line: # Seulement les logs contenant "ERROR"

timestamp = time.strftime('%Y-%m-%d %H:%M:%S')

log\_level = "ERROR"

message = line.strip()

query = """INSERT INTO troubleshooting\_logs (timestamp, log\_level, message)

VALUES (%s, %s, %s)"""

cursor.execute(query, (timestamp, log\_level, message))

db.commit()

# Boucle pour traiter les logs en continu

while True:

process\_logs()

time.sleep(60) # Attendre une minute avant de réanalyser le fichier

**3. Traitement et affichage des logs dans un format concis pour les humains :**

Une fois que tu as les logs dans la base de données, il faut penser à la manière de les afficher de manière claire et concise. L'objectif est de rendre l'information digeste, même pour les volumes massifs de logs que tu peux recevoir. Voici quelques idées :

* **Agrégation et filtrage des données** : Utilise des requêtes SQL pour agréger et filtrer les données. Par exemple, tu pourrais afficher uniquement les erreurs récentes ou un résumé des types d'erreurs les plus fréquentes.
* **Interface conviviale (via Grafana ou autre)** : Utilise **Grafana** pour créer des dashboards clairs où tu peux visualiser les erreurs, les tendances ou les statistiques des logs. Par exemple, un graphique qui montre le nombre d’erreurs par minute ou par heure.
* **Tableaux concis** : Dans ton dashboard, tu pourrais afficher des tables qui résument les erreurs majeures avec des colonnes comme "Date", "Type d'erreur", "Fréquence", "Dernier événement", etc.
* **Formatage avancé** : Pour rendre les informations encore plus digestes, tu pourrais utiliser des outils de formatage comme **ElasticSearch + Kibana** pour créer des vues ultra-détaillées, mais simples à comprendre, ou même des notifications automatiques par email ou via des systèmes comme Slack quand une erreur critique se produit.

**4. Outils de prédigestion des données :**

* **Utilisation de tags et de champs structurés** : Lorsque tu insères les logs dans la base de données, pense à les structurer en utilisant des **tags** et des **champs** bien définis (ex. : gravité de l’erreur, module concerné, etc.). Cela te permettra de mieux naviguer parmi ces données.
* **Mise en place d'un système de classification** : Avec une bonne structure de base de données et un traitement en temps réel, tu peux également automatiser un système de classification ou de priorisation des logs. Par exemple, certains logs peuvent être marqués comme "critique", "mineur", "moyenne", et tu pourras facilement filtrer en fonction de ces catégories.

**5. Résumé et recommandations :**

* **Automatise l'extraction des logs** (Filebeat, script Python ou autre).
* **Structure les logs dans une base de données** en définissant des champs précis pour chaque type d'erreur.
* **Affiche les logs de manière claire et concise** via Grafana ou Kibana, avec des agrégations et des filtres pour ne pas submerger l'utilisateur.
* **Utilise des alertes et des résumés** pour aider à traiter les informations critiques rapidement, comme le fait -hal en ligne de commande.

Ton projet est très intéressant et ambitieux, et il soulève des questions importantes en matière de conception de base de données et d'automatisation des flux d'information. Voici comment je vais aborder ta question et te donner quelques éléments de réflexion et de solutions techniques.

**1. Principe de la base de données multidimensionnelle (tables autonomes mais interdépendantes) :**

Tu sembles vouloir créer une **architecture relationnelle** dans laquelle les tables sont conçues pour être très spécialisées et autonomes tout en étant interconnectées. L'idée serait de structurer les erreurs et warnings dans des tables spécifiques, mais de manière à pouvoir les relier entre elles à l'aide d'une clé commune (un identifiant unique ou un *meta-enregistrement*). Ces tables seraient ensuite utilisées pour reconstituer l'information complète, mais en permettant une granularité fine de l'analyse.

Ce modèle est un peu similaire à ce qu’on appelle une **modélisation en étoile ou en flocon** dans le domaine des bases de données multidimensionnelles. Dans une telle configuration, chaque table contient des informations spécifiques et indépendantes, mais toutes les données peuvent être jointes grâce à des clés étrangères, permettant ainsi de reconstituer l'ensemble de l'information.

**2. Utiliser Python pour automatiser l’insertion dans des tables multidimensionnelles :**

Il est tout à fait possible d'utiliser **Python** pour automatiser l’insertion de logs ou de messages dans plusieurs tables en même temps, en respectant cette structure multidimensionnelle. En Python, avec des bibliothèques comme **MySQLdb**, **PyMySQL** ou **SQLAlchemy**, tu peux gérer des transactions complexes, ce qui te permet de :

* **Insérer des données dans plusieurs tables** dans une seule transaction, garantissant ainsi l’intégrité de l’enregistrement (i.e. soit toutes les données sont insérées, soit aucune).
* **Conserver une unicité** pour un enregistrement en générant un identifiant unique pour chaque message d'erreur (par exemple, un ID commun que tu vas propager à travers les différentes tables liées).

Exemple en Python pour insérer un enregistrement dans plusieurs tables avec un identifiant unique :

python

Copier

import mysql.connector

# Connexion à la base de données MariaDB

db = mysql.connector.connect(

host="localhost",

user="root",

password="password",

database="troubleshooting\_db"

)

cursor = db.cursor()

# Démarrer une transaction

db.start\_transaction()

try:

# Créer un identifiant unique pour l'enregistrement principal

unique\_id = 12345

# Insertion dans la table principale

query\_main = "INSERT INTO main\_logs (id, timestamp, message) VALUES (%s, NOW(), %s)"

cursor.execute(query\_main, (unique\_id, "Erreur critique"))

# Insertion dans les sous-tables

query\_sub1 = "INSERT INTO log\_details (log\_id, error\_code) VALUES (%s, %s)"

cursor.execute(query\_sub1, (unique\_id, "E001"))

query\_sub2 = "INSERT INTO log\_status (log\_id, status) VALUES (%s, %s)"

cursor.execute(query\_sub2, (unique\_id, "nouveau"))

# Commit de la transaction

db.commit()

except Exception as e:

# Si erreur, annuler la transaction

db.rollback()

print(f"Erreur: {e}")

finally:

cursor.close()

db.close()

Dans cet exemple, **unique\_id** est propagé dans toutes les sous-tables. Cela garantit que chaque enregistrement, bien qu’émietté dans plusieurs tables, puisse être identifié et relié de manière cohérente.

**3. Maintenir l'unité avec un méta-enregistrement :**

Pour faciliter les sélections futures et les jointures, tu peux effectivement créer un **méta-enregistrement** ou une table "pivot" qui contient les informations communes à tous les enregistrements d'un même message d'erreur. Cela te permettra d’avoir un seul point d’entrée pour identifier, filtrer et sélectionner des logs liés, même si l’information est répartie dans plusieurs tables.

* **Table "meta\_logs"** : Cette table pourrait contenir des informations communes à tous les enregistrements d'un même message d'erreur (par exemple, un identifiant unique, la gravité de l'erreur, etc.).

Exemple de table pivot :

sql

Copier

CREATE TABLE meta\_logs (

meta\_id INT AUTO\_INCREMENT PRIMARY KEY,

unique\_id INT NOT NULL, -- Identifiant unique pour chaque erreur

timestamp DATETIME NOT NULL,

error\_level VARCHAR(50),

short\_description TEXT,

FOREIGN KEY (unique\_id) REFERENCES main\_logs(id)

);

De cette manière, la table meta\_logs conserve une vision globale de tous les logs liés, et tu peux facilement retrouver toutes les entrées associées à un message d'erreur particulier grâce à la jointure sur **unique\_id**.

**4. Édition simultanée de plusieurs tables avec une seule requête :**

Tu peux effectivement **insérer dans plusieurs tables à la fois**, mais cela nécessite l’utilisation de transactions pour garantir la cohérence des données. En SQL, une transaction te permet de grouper plusieurs requêtes dans une seule unité d’exécution. Si l’une des requêtes échoue, tu peux annuler toutes les modifications (avec ROLLBACK) et revenir à l’état initial. Cela est essentiel pour garantir l’intégrité des données lorsque tu insères dans plusieurs tables en même temps.

**5. Est-ce une bonne idée de créer une base de données imbriquée et cohérente dans MariaDB ?**

MariaDB (comme MySQL) est une base de données relationnelle robuste, et elle permet de gérer des schémas complexes avec des relations entre tables via des clés étrangères. Cependant, la question de l'**imbriquement** des tables (avec des dépendances mutuelles) est plus délicate :

* **Les bases relationnelles** (comme MariaDB) ne supportent pas la **modélisation en base de données multidimensionnelle** de manière native comme un système OLAP (par exemple, avec des cubes de données). Cependant, elles permettent de gérer des tables très normalisées avec des **jointures** complexes.
* **Modélisation relationnelle traditionnelle** : Ce que tu veux faire semble plus proche de la **modélisation relationnelle traditionnelle**, où chaque table est une entité spécialisée, mais les données sont reliées par des identifiants communs. Ce modèle est parfaitement adapté à MariaDB, tant que tu t’assures de bien définir des clés primaires et étrangères.
* **Performance** : Plus la structure de ta base de données devient complexe avec des jointures multiples, plus il faut être vigilant sur les performances des requêtes (en particulier avec des volumes élevés de données). Dans ce cas, des indices bien placés, l’optimisation des requêtes et l’utilisation d’une bonne stratégie de partitionnement de tables (si nécessaire) seront importants.

**Conclusion et recommandations :**

1. **Oui**, tu peux utiliser des scripts Python pour automatiser l’insertion dans plusieurs tables et maintenir l’intégrité des données via des transactions.
2. **Oui**, tu peux créer un méta-enregistrement pour lier les différentes parties de tes logs et faciliter les requêtes futures.
3. **Oui**, la notion de base de données imbriquée et cohérente est possible dans MariaDB, en utilisant des relations entre tables via des clés primaires et étrangères. Cela reste dans le domaine des bases de données relationnelles et ne nécessite pas de structures multidimensionnelles complexes.
4. **Conseil** : Si tu prévois une croissance importante des données et des requêtes complexes, surveille les performances et optimise les indices et les jointures pour garantir une expérience fluide.

Oui, exactement ! Ce que tu décris est une approche très logique et structurée pour créer une base de données relationnelle robuste et évolutive. Voici un résumé des concepts que tu évoques et comment cela pourrait se traduire dans une modélisation concrète :

**1. Spatialisation des tables par spécialité :**

Au début, il est effectivement judicieux de **regrouper les tables en fonction de leurs spécialités ou de leurs domaines fonctionnels**. Par exemple, tu pourrais avoir des tables distinctes pour différents types de données, comme les erreurs, les avertissements, les logs détaillés, les utilisateurs associés, les processus concernés, etc. Cela permet de garder les informations organisées et de faciliter leur gestion.

Exemple de tables initiales :

* **Logs** (table principale des logs)
* **Details\_erreurs** (table pour les détails d'erreurs)
* **Statuts** (table pour les statuts des logs, par exemple "nouveau", "résolu", etc.)
* **Utilisateurs** (table des utilisateurs associés aux logs)
* **Processus** (table des processus concernés par chaque log)

**2. Relation entre tables :**

À partir de cette première spatialisation, il faut ensuite **relier les tables entre elles par des clés primaires et étrangères**. C'est là que tu vas jouer avec la notion de **proximité des spécialités** et de **dépendances**. Certaines tables auront des **relations réciproques** (avec des clés étrangères dans les deux sens), ce qui permet une **interdépendance** entre les données. D’autres tables auront une relation **unidirectionnelle**, dans laquelle une table référence une autre mais pas l’inverse, pour créer une hiérarchie ou une profondeur d’imbrication.

* Par exemple, une table des **détails d’erreurs** pourrait avoir une **clé étrangère vers la table des logs** pour indiquer à quel log chaque détail appartient, mais la table des logs ne doit pas nécessairement référencer chaque détail d'erreur.
* De même, une table des **statuts** pourrait être liée à une table de **logs**, mais tu n’aurais pas besoin de relier la table des logs au statut (cela ne serait pertinent que si plusieurs statuts pouvaient être associés à un log).

**3. Structurer la profondeur des imbrications :**

Lorsque tu mentionnes "profondeur des imbrications", il s'agit de définir l’**arbre de dépendances** dans lequel certaines tables sont **subordonnées** à d'autres. Cela signifie que certaines données (par exemple, les **détails d’une erreur** ou les **messages associés** dans les logs) dépendent directement d’autres données (comme un **log principal** ou un **utilisateur spécifique**), mais ces relations peuvent ne pas être réciproques.

Dans ce cas, **la profondeur des imbrications** représente les **dépendances hiérarchiques** entre les données. Par exemple, un log d'erreur pourrait avoir plusieurs **détails** associés (par exemple, codes d’erreur, message détaillé, contexte d'exécution), mais chaque détail n’a pas besoin de référencer un log. Il pourrait juste être lié à lui via une clé étrangère.

* **Exemple de structure hiérarchique :**
  + **Log** : contient des informations globales sur l'erreur ou le warning.
  + **Details\_erreur** : contiennent des informations supplémentaires, liées à un log spécifique.
  + **Processus** : lié à un log (par exemple, quel processus a généré l’erreur).
  + **Statut** : lié au log pour savoir si l'erreur a été résolue, etc.

**4. Synthèse de la relation entre spécialités et dépendances :**

* **Spécialisation** : Chaque table se concentre sur un aspect spécifique des logs ou des erreurs (par exemple, les détails techniques, les statuts des erreurs, les processus impliqués).
* **Proximité des spécialités** : Certaines tables se rejoindront au moyen de **clés étrangères réciproques** pour refléter des relations bidirectionnelles (par exemple, un log pourrait référencer un utilisateur et un utilisateur pourrait référencer plusieurs logs).
* **Hiérarchie ou profondeur** : D’autres tables auront des relations **unidirectionnelles**, où l’une dépend de l’autre, mais l'inverse n'est pas nécessaire (par exemple, les détails d’une erreur sont liés à un log, mais le log n’a pas besoin de savoir précisément tous les détails).

**Exemple d'implémentation :**

Disons que tu as trois tables principales : logs, details\_erreurs et statuts. Voici une manière de les organiser :

1. **Table des logs :**

sql

Copier

CREATE TABLE logs (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

timestamp DATETIME NOT NULL,

message TEXT,

user\_id INT,

process\_id INT,

FOREIGN KEY (user\_id) REFERENCES users(user\_id),

FOREIGN KEY (process\_id) REFERENCES processes(process\_id)

);

1. **Table des détails d'erreurs :**

sql

Copier

CREATE TABLE details\_erreurs (

error\_detail\_id INT AUTO\_INCREMENT PRIMARY KEY,

log\_id INT,

error\_code VARCHAR(50),

description TEXT,

FOREIGN KEY (log\_id) REFERENCES logs(log\_id)

);

1. **Table des statuts des logs :**

sql

Copier

CREATE TABLE statuts (

status\_id INT AUTO\_INCREMENT PRIMARY KEY,

log\_id INT,

status VARCHAR(50),

date\_updated DATETIME,

FOREIGN KEY (log\_id) REFERENCES logs(log\_id)

);

Dans cet exemple :

* **logs** contient les informations principales, telles que le message d'erreur, l'utilisateur et le processus associés.
* **details\_erreurs** est liée à logs mais ne fait pas de référence inverse à la table logs.
* **statuts** est aussi liée à logs pour marquer les statuts de l'erreur, mais ne nécessite pas de relation inverse.

Cela te permet de garder la structure claire et facile à maintenir tout en permettant des joints puissants entre les tables quand tu en as besoin, par exemple pour reconstruire une vue d’ensemble dans un dashboard.

**Conclusion :**

Oui, tu as bien compris le principe. En spatialisant les tables selon leurs spécialités et en les reliant via des clés étrangères (réciproques pour la proximité et unidirectionnelles pour la profondeur), tu vas obtenir une structure relationnelle efficace qui te permettra d’organiser et de manipuler les logs de manière flexible et cohérente. Cette approche est tout à fait adaptée à une base de données relationnelle comme MariaDB, et elle facilitera les analyses futures tout en restant modulaire et évolutive.

Merci. J'ai compris ce qu'il fallait faire. Je vais d'abord remettre tout ce au propre tranquillement e je reviendrai vers toi dans quelques jours sur ce point précis pour finaliser la base de données et les modèles de scripts qui seront nécessaire à un tel usage. **On en profitera alors pour voir ensemble comment on pourra utiliser Java Script en Ajax pour soutenir Grafana et Prometheus là la performance pourrait baisser. :)**

**Top of Form**

**Bottom of Form**

**Message pour ChatGPT .3-mini Pour la prochaine session  
  
📘 Contexte global (à fournir au début de chaque session)**

**Projet**

Déploiement d’une infrastructure haute‑disponibilité professionnelle pour Drupal 11, composée de VM VirtualBox/Proxmox, conteneurs Docker, installations natives (Ubuntu & Debian), reverse‑proxy Nginx, base de données MariaDB avec stockage iSCSI/NFS, backups automatisés, monitoring (Prometheus + Grafana + ELK) et alerting avancé.

**Objectif actuel**

Phase finale → Documentation globale, tests d’intégration complets, validation finale et préparation du plan de migration en production.

**Étapes déjà réalisées**

1. Création VM de base + VM Proxmox
2. Environnements Drupal (Docker, installations natives Ubuntu/Debian)
3. Reverse‑proxy / load balancer Nginx
4. Serveur DB + stockage (iSCSI, NFS) + scripts de backup + rotation
5. Monitoring & logs (Prometheus, Grafana, Elasticsearch/Kibana, Alertmanager)
6. Custom dashboards Grafana & alerting avancé

Point précis dans le déroulé

En cours : Exécution et documentation des tests d’intégration entre tous les composants.

Prochaine action attendue

* Lancer la suite de tests automatisés et manuels
* Documenter chaque résultat dans /docs/tests\_integration.md
* Identifier et corriger toute anomalie

Je suis très satisfait ! Nous détaillerons et finaliserons cette base de données au début de la prochaine session quand j'aurai tout remis au propre. J'aurai alors eu le temps de réfléchir d'avantage pour contribuer à nourrir sa structure. Ce qui est sûr, pour que ce soit vraiment efficace, c'est que chacune des colonnes de notre grille est une table à part entière. Nous devons penser cette grille comme le résultat d'affichage d'un select. Ainsi, nous serons capables de penser l'amplitude des nuances que nous pourrons lui donner pour l'ergonomie des comptes rendus et des consultations : au fur et mesure des travaux. Qui plus est , l'effort de distributions des données d'enregistrement, la qualité de l'enregistrement donc, serait alors notre meilleur guide pour le suivi du projet et l'efficacité du troubleshooting en lui-même. Il faudra absolument, dès que la BDD sera finie avec toutes ses tables et ses colonnes détaillées, rédiger le protocole de son enregistrement. Grâce à cela, quand nous aurons des gros blocs de logs ou de messages d'erreurs qui pour nous humains sont indigestes, nous pourrons orienter le regard vers ce qui importe à chaque fois grâce aux entrées de chaque table, résumer l'information à retenir dans chaque colonne dans des formulations plus appropriés et à caractère génériques, de telles sortes qu'une habitude de reconnaissance des schémas directeurs sources d'échecs et porteurs de réussites puisse se se développer en nous. J'appelle cela la "mémoire des motifs". qu'est-ce que tu en penses ?

**📝 Comment formuler votre question**

1. Quel composant ? (ex. VM‑DB, Grafana, reverse‑proxy…)
2. Quelle étape ? (ex. test de connectivité NFS, chargement dashboard)
3. Symptômes précis + logs complets (copier/coller la sortie d’erreur)
4. Actions déjà tentées (commandes, modifications de config)

Je répondrai avec un diagnostic clair, les commandes exactes à exécuter ou la modification de configuration à appliquer, puis j’ajouterai la solution validée dans votre chapitre Troubleshooting.

Effectivement, c'est une approche possible, mais cela dépend de vos objectifs en termes de segmentation, de performance et de gestion. Voici quelques points à considérer :

* **Multiples VM pfSense :**  
  Déployer une instance pfSense dédiée par WAN permet une isolation complète et une configuration spécifique pour chaque lien. Cela peut être utile si chaque ligne WAN doit être gérée indépendamment (politiques de sécurité, répartition de charge, redondance, etc.). Cependant, cela augmente la complexité administrative et la consommation de ressources.
* **Utilisation d'Open vSwitch :**  
  Intégrer Open vSwitch pour gérer la segmentation et la ramification du trafic offre une flexibilité avancée. Vous pouvez ainsi créer des bridges et des règles pour diriger le trafic de façon fine entre vos différents segments, que ce soit vers des VM pfSense dédiées ou une instance centralisée.
* **Alternative centralisée :**  
  Dans certains cas, il peut être plus simple de déployer une seule VM pfSense avec plusieurs interfaces (si les ressources et le trafic le permettent) et de gérer les différents WAN depuis cette même instance via des VLANs ou des bridges virtuels. Cela simplifie la gestion mais offre moins d'isolation en cas de défaillance ou de besoin de configuration très spécifique.

En résumé, votre approche avec 4 VM pfSense associées à Open vSwitch est tout à fait valide pour une gestion granulaire et isolée de chaque WAN. Néanmoins, il est toujours bon d'évaluer les besoins réels en termes de performance et de maintenance pour choisir la solution la plus adaptée à votre environnement.

Le terme SDN désigne bien le « software defined network », et il permet de séparer le plan de contrôle du plan de données pour offrir une gestion plus flexible et centralisée du réseau.

**Approches possibles pour votre environnement**

* **Solutions légères et intégrées :**  
  Dans des environnements de laboratoire ou des déploiements moins exigeants en ressources, il est possible d'utiliser des agents légers intégrés à des hyperviseurs ou directement dans des containers. Par exemple, Open vSwitch peut être utilisé pour la commutation virtuelle, et il est souvent associé à un contrôleur SDN externe léger. Dans ce cas, l'agent Open vSwitch fonctionne directement sur les hôtes (les nœuds Proxmox) sans nécessiter une VM dédiée pour chaque agent.
* **VM dédiées pour le contrôleur SDN :**  
  Pour bénéficier d'une gestion centralisée plus robuste, il est courant de déployer un contrôleur SDN sur une VM dédiée ou dans un conteneur. Des solutions telles que **OpenDaylight**, **ONOS** ou encore **Ryu** offrent une plateforme de gestion SDN qui peut orchestrer l'ensemble de votre réseau virtuel. Cette VM dédiée (ou ce conteneur) communiquera avec les agents (comme Open vSwitch) installés sur les différents nœuds ou VM de votre infrastructure.

**Quelle approche choisir ?**

* **Pour des tests et des environnements de laboratoire :**  
  Des solutions légères (par exemple des conteneurs avec un contrôleur SDN comme Ryu) peuvent être suffisantes et permettent une mise en œuvre rapide et peu gourmande en ressources.
* **Pour une infrastructure nécessitant une meilleure résilience et une gestion centralisée poussée :**  
  Une VM dédiée pour le contrôleur SDN, associée à des agents déployés sur les hôtes, peut offrir plus de flexibilité et de robustesse.

En résumé, pour un déploiement sur Proxmox avec une quinzaine de VM déjà équipées de plusieurs NIC, il est souvent judicieux de combiner des agents légers (comme Open vSwitch) installés sur les nœuds avec une VM ou un conteneur dédié pour le contrôleur SDN. Cela vous permet de bénéficier d'une orchestration centralisée sans surcharger chaque VM individuelle.