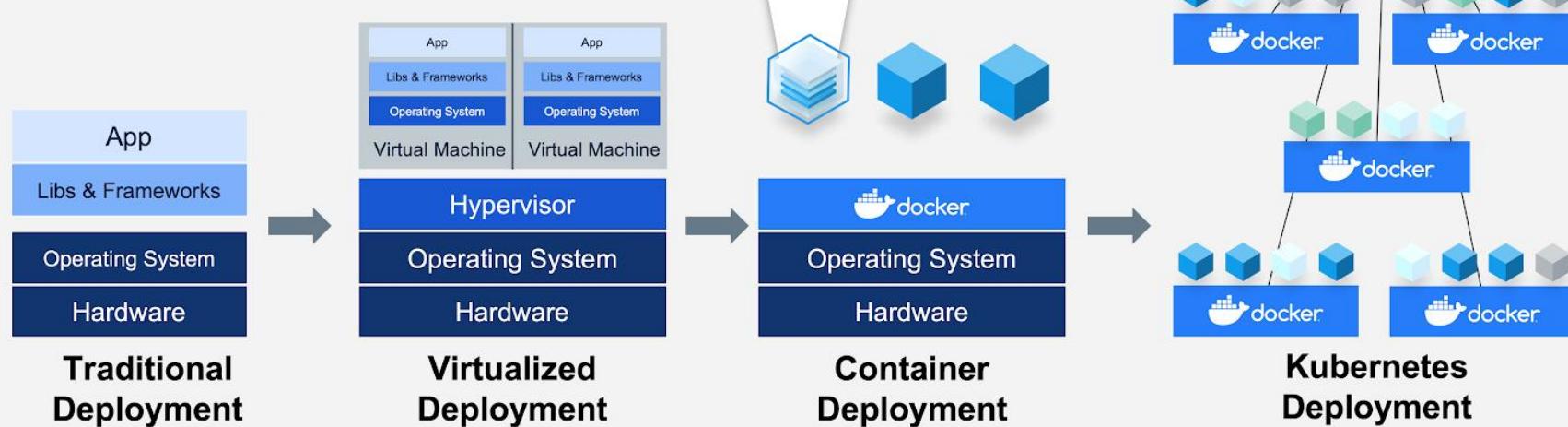


Kubernetes & Docker work together to build & run containerized applications



À quoi sert Kubernetes, et quel est son « terrain de jeu » ?

Utilité principale

Kubernetes est un **orchestrateur de conteneurs**.

Son rôle est de **déployer, exécuter, superviser et faire évoluer** des applications conteneurisées (Docker ou compatibles) **de manière automatique et fiable**, à l'échelle d'un cluster de machines.

Kubernetes ne *remplace pas* Docker :

- **Docker** = construire et exécuter un conteneur
 - **Kubernetes** = orchestrer **des centaines ou milliers de conteneurs, sur plusieurs machines**
-

Les problèmes concrets que Kubernetes résout

Sans Kubernetes, en production, tu dois gérer manuellement :

- Où lancer les conteneurs
- Que faire si un conteneur plante
- Comment répartir la charge
- Comment faire une montée en charge (scaling)
- Comment mettre à jour sans coupure
- Comment gérer la configuration et les secrets

Kubernetes **automatise tout ça**.

Son « terrain de jeu » (là où Kubernetes a du sens)

1. Environnements multi-machines (cluster)

- Plusieurs VM ou serveurs physiques
- Kubernetes décide **où** placer chaque conteneur

Typique :

- Proxmox + plusieurs VM
 - Cloud (AWS, Azure, GCP)
 - On-premise entreprise
-

2. Applications découpées en services

Kubernetes est fait pour :

- micro-services
- front / back / API / workers
- bases de données **externes ou gérées à part**

Exemple :

- Frontend (Nginx)
 - Backend (API Java / Python)
 - Workers (traitements async)
 - Cache (Redis)
 - DB hors cluster ou StatefulSet maîtrisé
-

3. Applications qui doivent tenir dans le temps

Kubernetes excelle quand :

- l'application tourne 24/7
- les pannes sont inacceptables
- les mises à jour doivent être transparentes

Fonctions clés :

- **Auto-healing** (redémarrage automatique)
 - **Rolling update** (mise à jour sans coupure)
 - **Rollback** (retour arrière immédiat)
 - **Autoscaling** (HPA)
-

4. Équipes DevOps / CI-CD

Kubernetes est **le standard industriel** pour :

- GitLab CI / GitHub Actions
 - Déploiements déclaratifs (YAML)
 - GitOps (ArgoCD, Flux)
-

Ce que Kubernetes n'est PAS

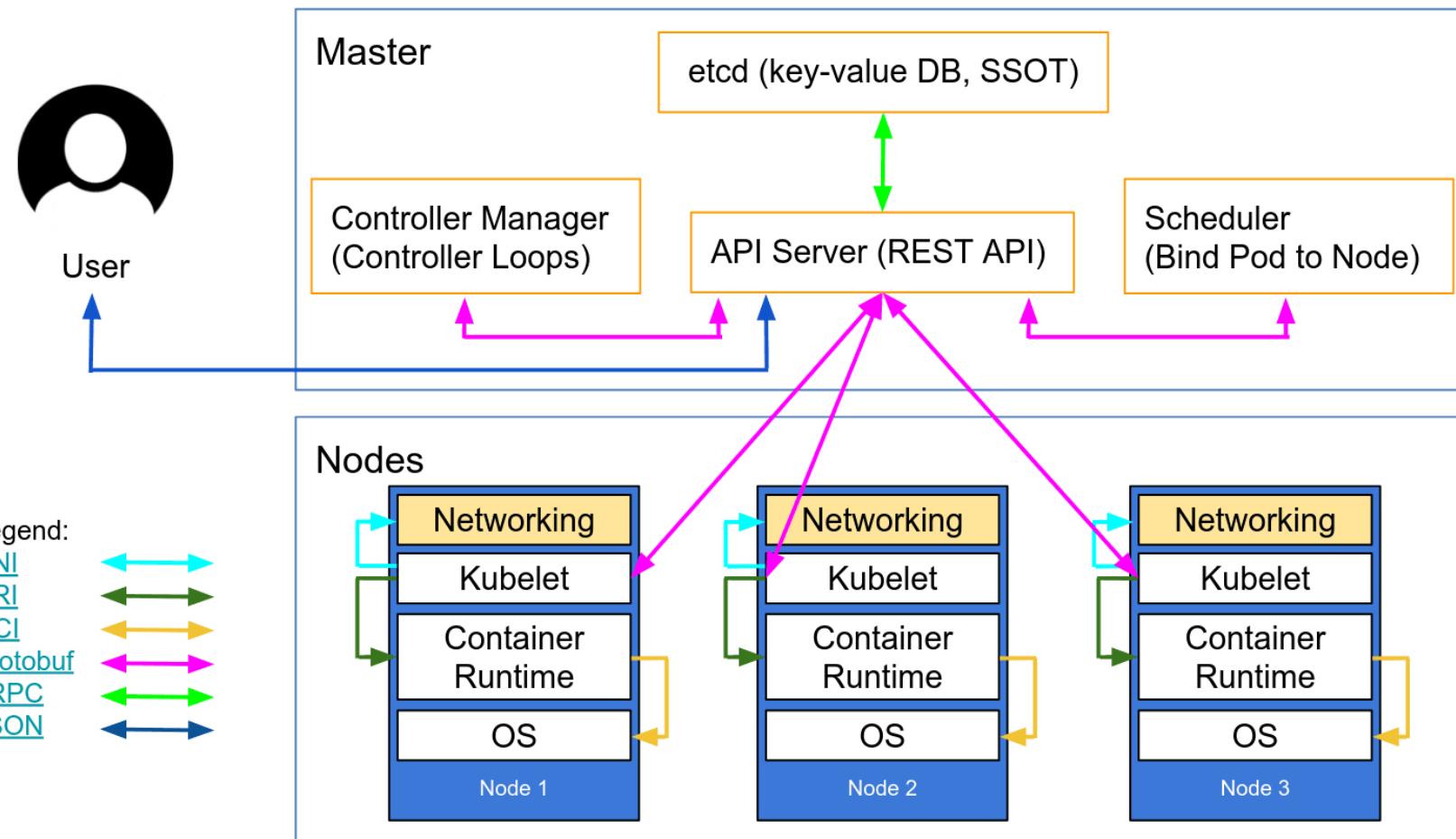
Important pour cadrer

Pas utile si :

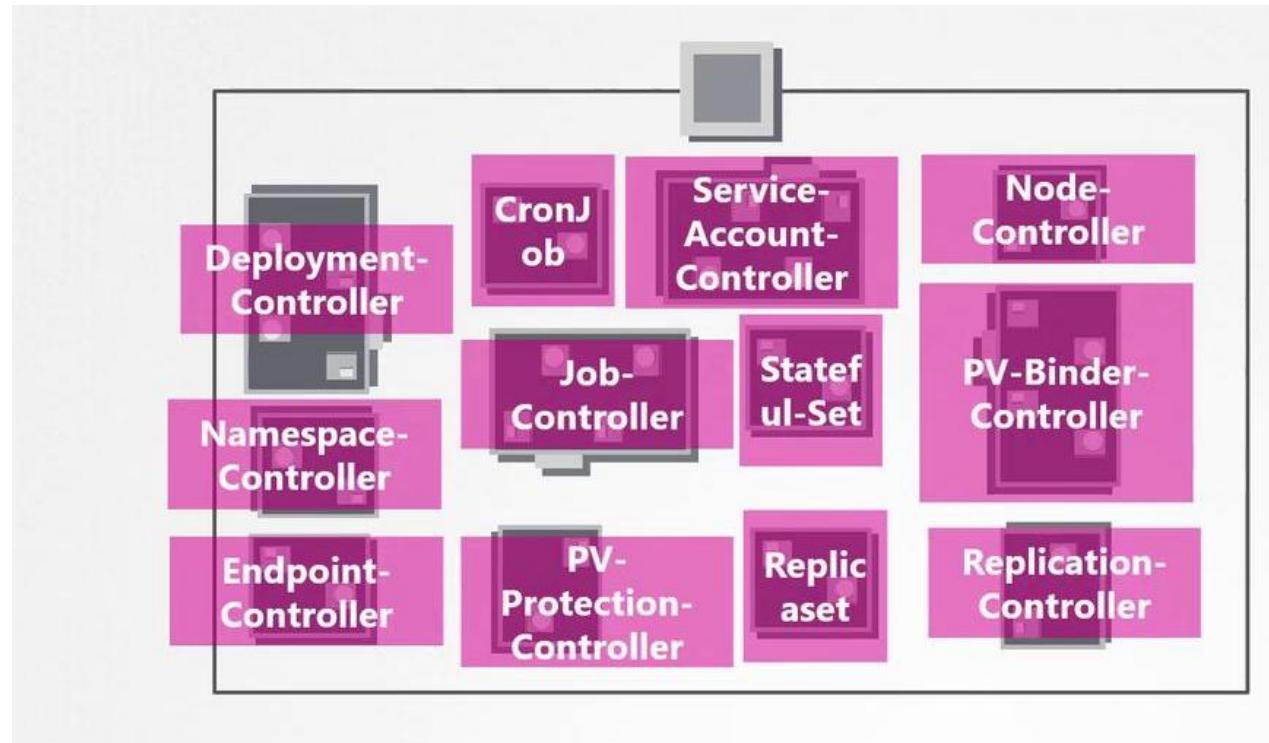
- 1 seule machine

- 2–3 conteneurs simples
- pas de besoin HA
- pas de scaling

Dans ce cas : **Docker Compose suffit largement**



Controller Manager (kube-controller-manager)



Le cerveau logique du cluster

Ce composant lance **plein de contrôleurs**, chacun surveillant un état précis.

Principe fondamental

Observer → comparer → corriger

Contrôleurs principaux

- Deployment controller

- **ReplicaSet controller**
- **Node controller**
- **Job / CronJob controller**
- **Endpoint controller**
- **Service controller**

Exemple concret

On demande :

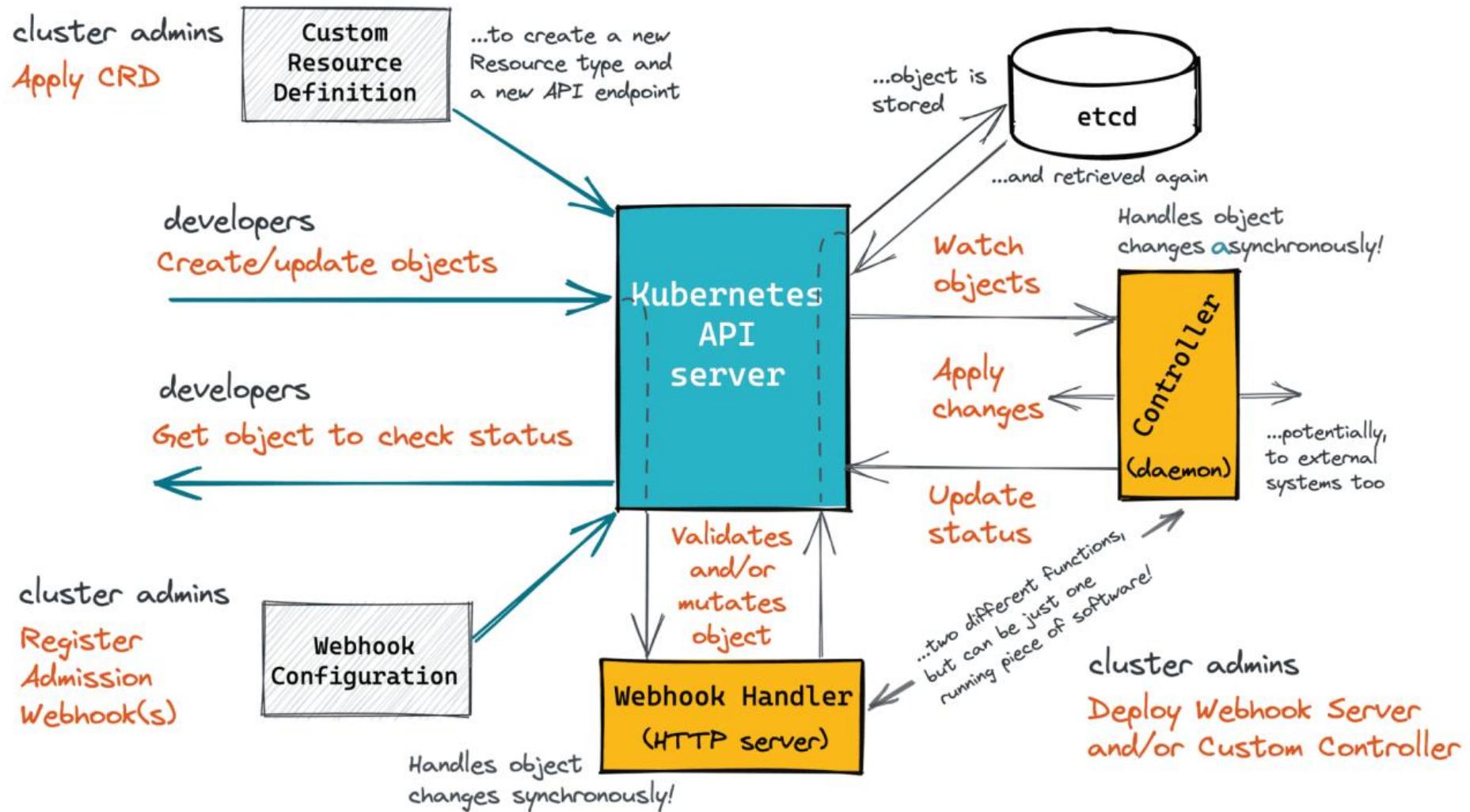
replicas: 3

➡ Controller voit :

- état réel = 2 pods
- état désiré = 3 pods
Il crée 1 Pod supplémentaire (via scheduler)

Kubernetes fonctionne **en boucle de réconciliation permanente**

API Server (kube-apiserver)



Point d'entrée unique du cluster

Rôle

- Reçoit **toutes les requêtes** :
 - kubectl
 - CI/CD
 - nodes
 - controllers
- Valide, authentifie et autorise
- Écrit / lit l'état du cluster dans **etcd**

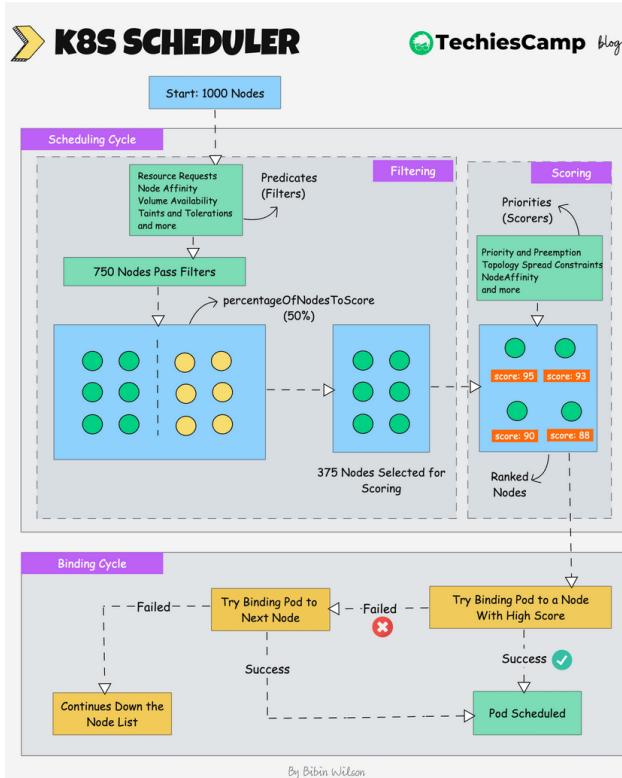
Il gère :

- Pods
- Deployments
- Services
- Secrets / ConfigMaps
- Nodes
- Volumes
- RBAC

Règle d'or

Tout passe par l'API Server. Rien ne contourne l'API.

Scheduler (kube-scheduler) :



Le placeur de Pods

Rôle

- Décide **sur quel node** un Pod doit tourner
- Ne lance rien lui-même
- Il **assigne**

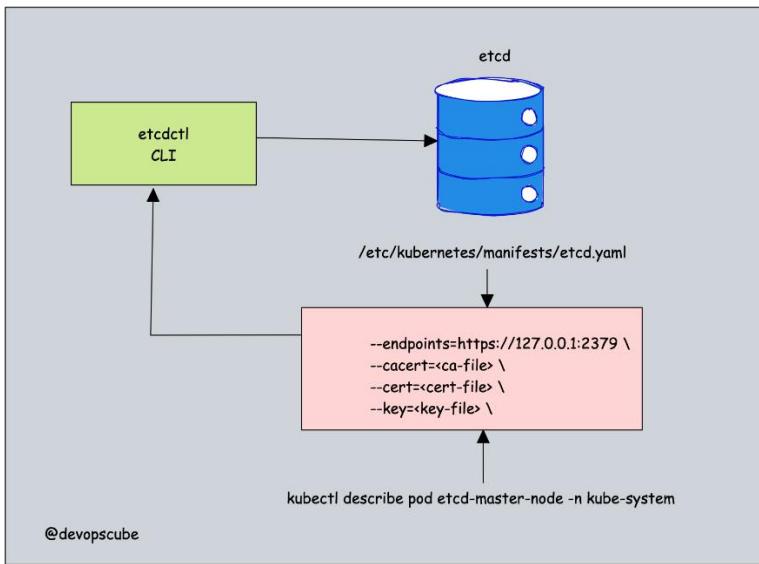
Il prend en compte :

- Ressources CPU / RAM
- Affinités / anti-affinités
- Taints / tolerations
- Contraintes réseau / stockage
- Labels des nodes

Flux logique

1. Pod créé (sans node)
2. Scheduler analyse le cluster
3. Choisit le *meilleur* node
4. Écrit la décision via l'API Server

etcd (datastore du cluster):



La mémoire du cluster

Rôle

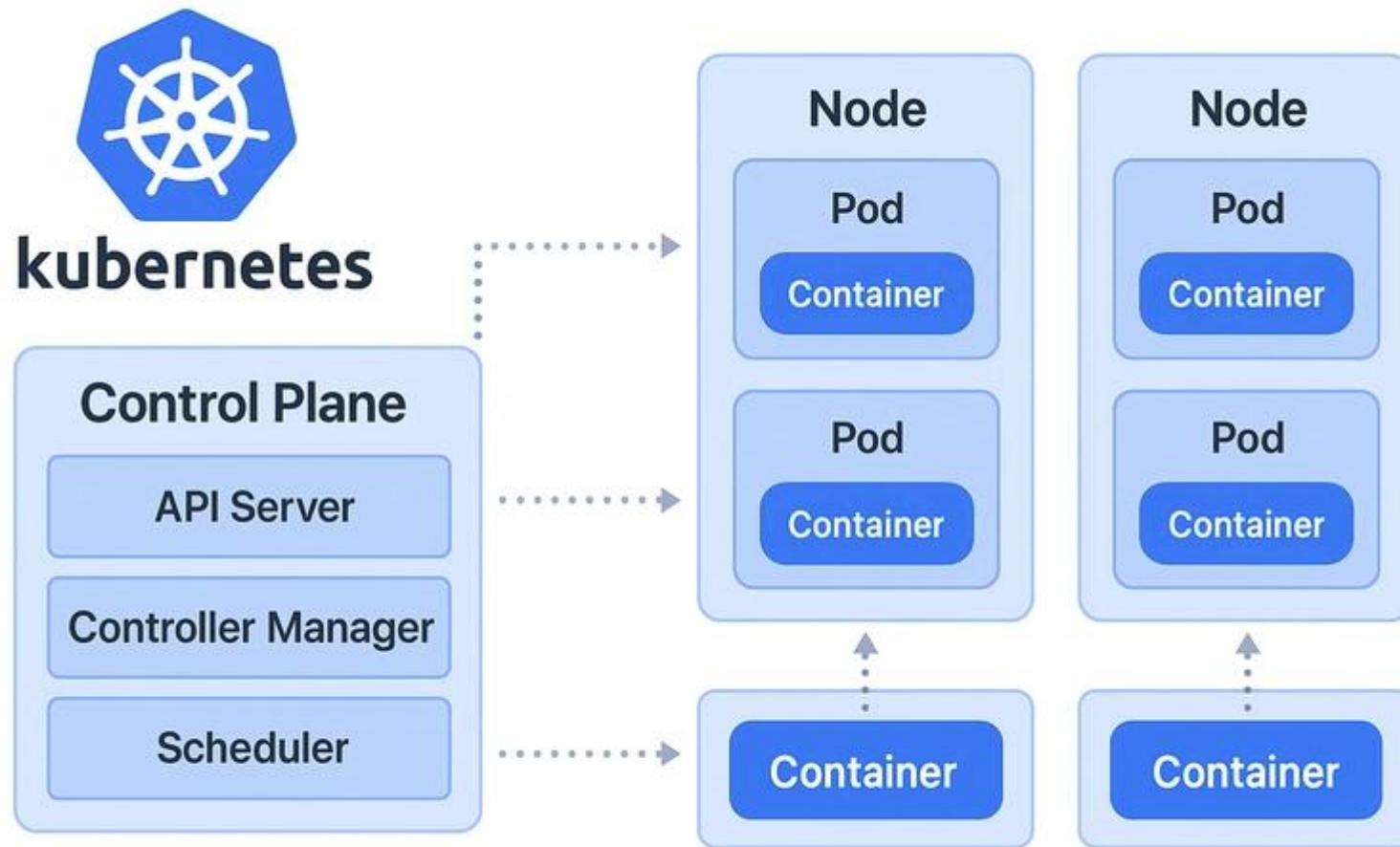
- Base clé/valeur distribuée
- Stocke **l'état désiré et réel** du cluster

Contenu typique :

- Définition des Pods
- Réplicas attendus
- ConfigMaps
- Secrets

- États des nodes
- Verrous internes

Si etcd est perdu : Le cluster **oublie tout**



Le kubelet : l'agent d'exécution du node

Dans **Kubernetes**, le **kubelet** est le composant clé présent sur chaque node (worker ou master).

C'est **l'agent local** qui fait réellement tourner les **Pods** et rend compte de leur état.

Rôle fondamental

Le kubelet transforme les décisions du control plane en actions concrètes sur la machine.

Il n'orchestre rien (ça, c'est le master), il **exécute**.

À quoi sert le kubelet exactement ?

1. Récupérer les ordres depuis l'API Server

Le kubelet :

- s'enregistre auprès du cluster
- **observe les Pods qui lui sont assignés**
- interroge régulièrement l'API Server

Exemple :

"Ce node doit exécuter ces 3 Pods."

2. Créer et maintenir les Pods

Pour chaque Pod assigné, le kubelet :

- télécharge les images

- crée les conteneurs
- monte les volumes
- configure le réseau
- démarre les conteneurs

Il s'appuie sur le **runtime de conteneurs** (containerd, CRI-O...).

3. Surveiller en permanence l'état des conteneurs

Le kubelet :

- vérifie que les conteneurs tournent
- redémarre si nécessaire
- surveille les ressources
- applique les **probes**

Types de probes :

- **Liveness** → le conteneur est-il vivant ?
 - **Readiness** → peut-il recevoir du trafic ?
 - **Startup** → a-t-il démarré correctement ?
-

4. Remonter l'état réel au cluster

Le kubelet **publie en continu** :

- état des Pods

- erreurs
- métriques
- statut du node (Ready / NotReady)

Ces infos sont envoyées à l'API Server

Les controllers réagissent si nécessaire

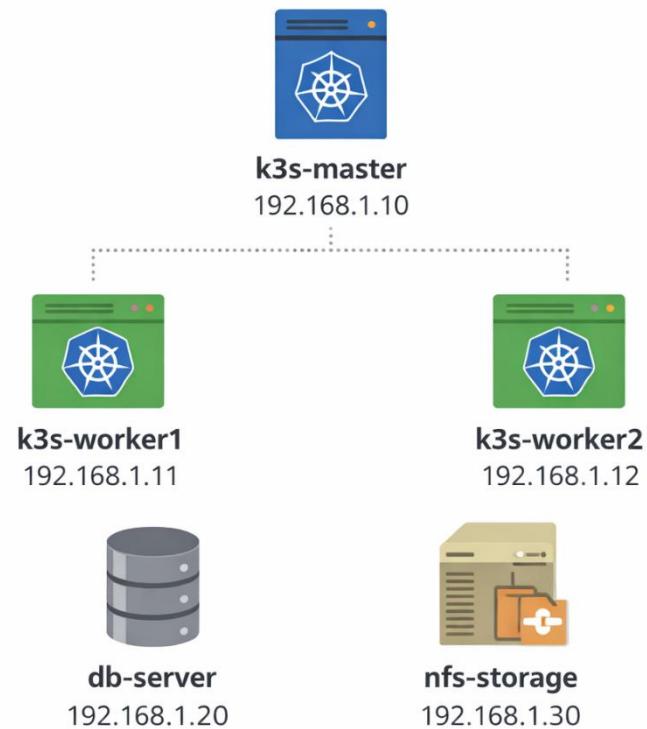
Le kubelet ne fait PAS

Il ne décide pas :

- où placer un Pod
- combien de Pods lancer
- quand scaler

Il ne communique pas directement avec les autres nodes

Il obéit strictement à l'API Server



Rôle	Nom machine	Hostname	Adresse IP	Ressources minimales (CPU / RAM)	Services installés / utiles
Kubernetes Control Plane	k3s-master	k3sm.afpadevops.lab	192.168.156.10	2 vCPU / 4 Go RAM	k3s-server (API Server, Scheduler, Controller Manager, etcd), containerd, coredns, traefik (ou nginx ingress), metrics-server
Kubernetes Worker	k3s-worker1	k3sw.afpadevops.lab	192.168.156.11	2 vCPU / 4 Go RAM	k3s-agent, containerd, workloads applicatifs (WordPress, API, etc.), CNI
Kubernetes Worker	k3s-worker2	k3sw.afpadevops.lab	192.168.156.12	2 vCPU / 4 Go RAM	k3s-agent, containerd, workloads applicatifs, CNI
Base de données externe	db-mariadb	db-mariadb.afpadevops.lab	192.168.1.130	2 vCPU / 4 Go RAM (mini)	MariaDB Server, mariadb-client, cron (backups)
Stockage partagé	nfs-storage	nfs-storage.afpadevops.lab	192.168.1.120	2 vCPU / 2 Go RAM (mini)	nfs-kernel-server, exports NFS (/exports/wp-content, /exports/backups)

Flux réseau principaux (à documenter)

Source	Destination	Port	Usage
k3s-master workers		TCP 6443	API Kubernetes
workers	db-mariadb	TCP 3306	Accès MariaDB
workers	nfs-storage	TCP/UDP 2049	Volumes RWX
clients	ingress	TCP 80/443	Accès WordPress

1 / Environnement :

Storage NFS :

✗ OS : **Debian 13**

✗ Disque dédié : **/dev/sdb** (vide)

✗ Réseau k3s : 192.168.1.0/24

✗ Rôle : **stockage NFS pour Kubernetes** (RWX)

Objectif :

/dev/sdb formaté et monté sur /srv/nfs

1-1/ partitionner sdb

lsblk

Partitionner le disque

fdisk /dev/sdb

Séquence :

n # new

p # primary

1 # partition 1

<ENTER>

<ENTER>

w

On doit voir sdb1 via la commande lsblk

On formate en ext4 :

mkfs.ext4 /dev/sdb1

```
root@k3snfs:~# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda      8:0    0   32G  0 disk
└─sda1   8:1    0 30,3G  0 part /
└─sda2   8:2    0   1K  0 part
└─sda5   8:5    0  1,7G  0 part [SWAP]
sdb      8:16   0   32G  0 disk
└─sdb1   8:17   0   32G  0 part
sr0     11:0    1 783M  0 rom
root@k3snfs:~# mkfs.ext4 /dev/sdb1
mke2fs 1.47.2 (1-Jan-2025)
Discarding device blocks: done
Creating filesystem with 8388352 4k blocks and 2097152 inodes
Filesystem UUID: 0819ea41-dc7b-461a-abdb-d0f52e35df21
Superblock backups stored on blocks:
            32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
            4096000, 7962624

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

root@k3snfs:~# |
```

```
#!/bin/bash
set -euo pipefail

#####
# CONFIGURATION (À ADAPTER)
#####

# Partition déjà existante (NE PAS FORMATER)
PARTITION="/dev/sdb1"

# Type de FS attendu
FS_TYPE="ext4"

# Point de montage
MOUNT_POINT="/srv/nfs"

# Réseau autorisé à monter les exports
ALLOWED_CIDR="192.168.1.0/24"

# Exports NFS à publier : "nom_logique:chemin_relatif"
```

```
EXPORTS=(  
    "wp-content:wp-content"  
    "backups:backups"  
    "shared:shared"  
)  
  
# Options NFS côté serveur  
# no_root_squash utile pour Kubernetes sur un réseau de confiance (lab)  
EXPORT_OPTIONS="rw,sync,no_subtree_check,no_root_squash"  
  
# Gestion idempotente de /etc/exports  
EXPORTS_FILE="/etc/exports"  
EXPORTS_TAG="# managed-by-setup-nfs (noformat)"  
  
#####  
# OUTILS  
#####  
log() { echo "[INFO] $*"; }  
warn() { echo "[WARN] $*" >&2; }  
die() { echo "[ERROR] $*" >&2; exit 1; }
```

```
require_root() {
    [[ "${EUID}" -eq 0 ]] || die "Ce script doit être exécuté en root"
}
```

```
pkg_install() {
    local pkg="$1"
    if ! dpkg -l | awk '{print $2}' | grep -qx "$pkg"; then
        log "Installation paquet: $pkg"
        apt update
        apt install -y "$pkg"
    else
        log "Paquet déjà installé: $pkg"
    fi
}
```

```
ensure_dir() {
    local d="$1"
    if [[ ! -d "$d" ]]; then
        log "Création du répertoire: $d"
```

```

mkdir -p "$d"
else
    log "Répertoire déjà présent: $d"
fi
}

mount_is_active() {
    mountpoint -q "$MOUNT_POINT"
}

fstab_has_mountpoint() {
    awk 'NF && $1 !~ /^#/ /etc/fstab | awk '{print $2}' | grep -qx "$MOUNT_POINT"
}

fstab_remove_mountpoint() {
    local tmp
    tmp=$(mktemp)
    awk -v mnt="$MOUNT_POINT" '
/^#/ { print; next }
NF==0 { print; next }
'
}
```

```

$2==mnt { next }

{ print }

' /etc/fstab > "$tmp"
cat "$tmp" > /etc/fstab
rm -f "$tmp"

}

validate_partition() {

[[ -b "$PARTITION" ]] || die "Partition introuvable: $PARTITION"

local fstype
fstype=$(blkid -s TYPE -o value "$PARTITION" 2>/dev/null || true)
[[ -n "$fstype" ]] || die "Aucun FS détecté sur $PARTITION (blkid vide)."

if [[ "$fstype" != "$FS_TYPE" ]]; then
    warn "FS détecté sur $PARTITION: $fstype (attendu: $FS_TYPE)"
    warn "Je continue quand même (tu peux corriger FS_TYPE si besoin)."
else
    log "FS détecté: $fstype (OK)"
fi

```

```
}

fstab_upsert() {

local uuid

uuid=$(blkid -s UUID -o value "$PARTITION" 2>/dev/null || true)

[[ -n "$uuid" ]] || die "Impossible de lire l'UUID de $PARTITION."

local line="UUID=${uuid} ${MOUNT_POINT} ${FS_TYPE} defaults 0 2"

if fstab_has_mountpoint; then

log "Entrée fstab existante pour $MOUNT_POINT -> remplacement"

fstab_remove_mountpoint

else

log "Aucune entrée fstab pour $MOUNT_POINT -> ajout"

fi

echo "$line" >> /etc/fstab

# systemd peut cacher l'ancienne version

systemctl daemon-reload
```

```
}
```

```
mount_partition() {
    ensure_dir "$MOUNT_POINT"
```

```
    if mount_is_active; then
        log "Déjà monté: $MOUNT_POINT"
        return 0
    fi
```

```
    log "Montage via fstab: $MOUNT_POINT"
    mount "$MOUNT_POINT" || die "Échec du montage sur $MOUNT_POINT (voir: dmesg | tail -n 50)"
}
```

```
setup_exports_dirs() {
    for entry in "${EXPORTS[@]}"; do
        IFS=: read -r name relpath <<< "$entry"
        [[ -n "$name" && -n "$relpath" ]] || die "Entrée EXPORTS invalide: $entry"
        ensure_dir "${MOUNT_POINT}/${relpath}"
    done
```

```
}
```

```
ensure_exports_block() {
    [[ -f "$EXPORTS_FILE" ]] || touch "$EXPORTS_FILE"

    if ! grep -qF "${EXPORTS_TAG} - begin" "$EXPORTS_FILE"; then
        log "Ajout du bloc géré dans $EXPORTS_FILE"
        cat >> "$EXPORTS_FILE" <<EOF

```

```
    ${EXPORTS_TAG} - begin
    ${EXPORTS_TAG} - end
EOF
fi
}
```

```
rewrite_exports_block() {
    log "Mise à jour idempotente du bloc exports"

    local tmp
    tmp=$(mktemp)"

```

```

# On conserve tout sauf l'intérieur du bloc géré

awk -v begin="${EXPORTS_TAG} - begin" -v end="${EXPORTS_TAG} - end" '
$0 == begin { print; inblock=1; next }
$0 == end  { inblock=0; print; next }
inblock==1 { next }
{ print }
' "$EXPORTS_FILE" > "$tmp"

# Réécriture avec injection des exports juste après "begin"

:> "$EXPORTS_FILE"

while IFS= read -r line; do
echo "$line" >> "$EXPORTS_FILE"

if [[ "$line" == "${EXPORTS_TAG} - begin" ]]; then
for entry in "${EXPORTS[@]}"; do
IFS=: read -r name relpath <<< "$entry"
echo "${MOUNT_POINT}/${relpath} ${ALLOWED_CIDR}(${EXPORT_OPTIONS})" >> "$EXPORTS_FILE"
done
done
done < "$tmp"

```

```
rm -f "$tmp"  
}
```

```
apply_exports() {  
    log "Application des exports"  
    exportfs -ra  
    exportfs -v  
}
```

```
setup_nfs_service() {  
    pkg_install "nfs-kernel-server"  
  
    log "Activation/démarrage nfs-server"  
    systemctl enable --now nfs-server  
  
    log "Vérification écoute NFS (2049)"  
    if ss -lntp | grep -E ':2049\b' >/dev/null; then  
        log "NFS écoute sur 2049 (OK)"  
    else
```

```
    warn "Port 2049 non détecté. Vérifie: systemctl status nfs-server / firewall"
fi
}
```

```
#####
#####
```

```
# MAIN
```

```
#####
#####
```

```
require_root
```

```
log "Validation de la partition (sans formatage)"
```

```
validate_partition
```

```
log "Configuration fstab (idempotent)"
```

```
fstab_upsert
```

```
log "Montage du disque"
```

```
mount_partition
```

```
log "Création des dossiers exportés"
```

```
setup_exports_dirs
```

```
log "Installation + démarrage du serveur NFS"
setup_nfs_service

log "Configuration /etc(exports"
ensure_exports_block
rewrite_exports_block
apply_exports

log "Serveur NFS prêt."
log "Partition : ${PARTITION} (FS attendu: ${FS_TYPE})"
log "Montage : ${MOUNT_POINT}"
log "Réseau : ${ALLOWED_CIDR}"
for entry in "${EXPORTS[@]}"; do
IFS=: read -r name relpath <<< "$entry"
log "Export : ${MOUNT_POINT}/${relpath}"
done
```

```
Invite de commandes      X  tech@k3snfs: ~      X  +  ▾
```

```
[INFO] Activation/démarrage nfs-server
[INFO] Vérification écoute NFS (2049)
[INFO] NFS écoute sur 2049 (OK)
[INFO] Configuration /etc(exports
[INFO] Ajout du bloc géré dans /etc(exports
[INFO] Mise à jour idempotente du bloc exports
[INFO] Application des exports
/srv/nfs/wp-content
    192.168.1.0/24(sync,wdelay,hide,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
/srv/nfs/backups
    192.168.1.0/24(sync,wdelay,hide,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
/srv/nfs/shared
    192.168.1.0/24(sync,wdelay,hide,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
[INFO] Serveur NFS prêt.
[INFO] Partition : /dev/sdb1 (FS attendu: ext4)
[INFO] Montage   : /srv/nfs
[INFO] Réseau    : 192.168.1.0/24
[INFO] Export     : /srv/nfs/wp-content
[INFO] Export     : /srv/nfs/backups
[INFO] Export     : /srv/nfs/shared
root@k3snfs:~# mount | grep /srv/nfs
/dev/sdb1 on /srv/nfs type ext4 (rw,relatime)
root@k3snfs:~# exportfs -v
/srv/nfs/wp-content
    192.168.1.0/24(sync,wdelay,hide,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
/srv/nfs/backups
    192.168.1.0/24(sync,wdelay,hide,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
/srv/nfs/shared
    192.168.1.0/24(sync,wdelay,hide,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
root@k3snfs:~# ss -lntp | grep 2049
LISTEN 0        4096          0.0.0.0:2049          0.0.0.0:*
LISTEN 0        4096          [::]:2049            [::]:*
root@k3snfs:~# |
```

commandes de contrôle

mount | grep /srv/nfs

exportfs -v

ss -lntp | grep 2049

Tableau récapitulatif NFS

Paramètres globaux

Élément	Valeur
Serveur NFS (VM)	K3snfs
IP serveur NFS	192.168.1.120
Point de montage du disque (serveur) /srv/nfs (monté depuis /dev/sdb1)	
Réseau autorisé	192.168.1.0/24
Port NFS	TCP 2049
Version recommandée	NFSv4.1 (vers=4.1)
Options export (serveur)	rw,sync,no_subtree_check,no_root_squash
Options mount (clients)	rw,sync,_netdev,hard,proto=tcp,vers=4.1,timeo=600,retrans=2

Exports NFS et montages clients

Usage	Export NFS (serveur)	Cible NFS (client)	Point de montage (client)	Notes
WordPress – contenu persistant	/srv/nfs/wp-content	192.168.1.120:/srv/nfs/wp-content	/mnt/wp-content	RWX requis côté K8s (NFS OK). Pour pods : PV/PVC NFS.
Backups (dumps DB, archives)	/srv/nfs/backups	192.168.1.120:/srv/nfs/backups	/mnt/backups	Sert aux sauvegardes cluster/app (jobs, scripts).
Partage commun (optionnel)	/srv/nfs/shared	192.168.1.120:/srv/nfs/shared	/mnt/shared	Pour données partagées entre apps (à limiter en prod).

Cibles Kubernetes (prévues)

Élément Kubernetes Export utilisé	Exemple de chemin monté dans le Pod	Remarque	
PVC wp-content	/srv/nfs/wp-content	/var/www/html/wp-content	WordPress nécessite persistance (uploads/plugins/themes).
PVC backups	/srv/nfs/backups	/backup	Pour jobs de sauvegarde (cronjob).

Préparation de la DB

Apt install mariadb-server, mais versionnable via compose c'est mieux

Préparation du master k3s

modifier /etc/hosts (ou dns)

```
GNU nano 8.4                               /etc/hosts *
127.0.0.1      localhost
127.0.1.1      k3sm
192.168.1.123  k3sm
192.168.1.124  k3sw1
192.168.1.125  k3sw2
```

Désactiver le swap (obligatoire)

```
sudo swapoff -a
```

```
sudo sed -i.bak '/sswap\s/s/^/#/' /etc/fstab
```

C. Modules kernel utiles

```
sudo tee /etc/modules-load.d/k8s.conf >/dev/null <<'EOF'
```

```
overlay
```

```
br_netfilter
```

EOF

```
sudo modprobe overlay
```

```
sudo modprobe br_netfilter
```

```
root@k3sm:~# cat <<'EOF' | tee /etc/modules-load.d/k8s.conf > /dev/null
overlay
br_netfilter
EOF
root@k3sm:~# modprobe overlay
root@k3sm:~# modprobe br_netfilter
root@k3sm:~# lsmod | grep -E 'overlay|br_netfilter'
br_netfilter      36864  0
bridge           389120  1 br_netfilter
overlay          217088  0
root@k3sm:~# |
```

Pour vérifier :

```
lsmod | grep -E 'overlay|br_netfilter'
```

Sysctl réseau (forward + bridge)

```
sudo tee /etc/sysctl.d/99-kubernetes-cri.conf >/dev/null <<'EOF'
```

```
net.bridge.bridge-nf-call-iptables = 1
```

```
net.bridge.bridge-nf-call-ip6tables = 1
```

```
net.ipv4.ip_forward      = 1  
EOF
```

Pour vérifier :

```
sudo sysctl -system
```

```
kernel.pid_max = 4194304  
net.bridge.bridge-nf-call-iptables = 1  
net.bridge.bridge-nf-call-ip6tables = 1  
net.ipv4.ip_forward = 1
```

SUR TOUS LES MEMBRES MASTER ET WORKERS

//////////

Pourquoi on touche aux sysctl réseau ?

Kubernetes repose sur :

- des **interfaces virtuelles** (veth, bridges)
- du **NAT**
- des **tables iptables / nftables**
- du **routage entre réseaux** (Pods ↔ Services ↔ Nodes)

Par défaut, Linux **ne fait pas tout ça** sans autorisation explicite.

1 net.ipv4.ip_forward = 1

👉 À quoi ça sert

Autorise le **routage IP** entre interfaces.

Sans ça :

- une machine Linux **ne route pas** les paquets
- elle agit comme un simple hôte, pas comme un routeur

🔴 Sans ce paramètre

- un Pod peut parler à son Node
- mais **ne peut pas atteindre** :
 - d'autres Pods
 - des Services
 - l'extérieur

🟢 Avec ce paramètre

- le Node peut router :
 - Pod → Pod
 - Pod → Service
 - Pod → Internet
 - Pod → DB externe (MariaDB)

👉 **Indispensable** pour Kubernetes.

2 net.bridge.bridge-nf-call-iptables = 1

👉 À quoi ça sert

Force le trafic **passant par un bridge Linux** à être inspecté par **iptables**.

Pourquoi c'est crucial

- Les Pods sont connectés via des **bridges virtuels**
- Sans ce paramètre :
 - le trafic Pod↔Pod **bypassé iptables**
 - les règles Kubernetes **ne s'appliquent pas**

🔴 Sans ce paramètre

- Services Kubernetes (ClusterIP) peuvent ne pas fonctionner
- NetworkPolicy inefficaces
- comportements réseau “fantômes”

🟢 Avec ce paramètre

- iptables voit le trafic bridge
- kube-proxy peut :
 - faire le NAT
 - appliquer les règles de Service
 - bloquer/autoriser via NetworkPolicy

3 net.bridge.bridge-nf-call-ip6tables = 1

👉 À quoi ça sert

Même chose que le précédent, mais pour **IPv6**.

Pourquoi on l'active même si on n'utilise pas IPv6

- Kubernetes peut activer IPv6 partiellement
- Certains plugins réseau s'appuient dessus
- Évite des bugs subtils

👉 Bonne pratique “safe by default”.

Installer le master

```
apt install sudo && apt install curl
```

```
root@k3sm:~# curl -sfL https://get.k3s.io | sudo sh -s - server --node-ip 192.168.1.123 --advertise-address 192.168.1.123 --tls-san 192.168.1.123 --write-kubeconfig-mode 644
[INFO] Finding release for channel stable
[INFO] Using v1.34.3+k3s1 as release
[INFO] Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.34.3+k3s1/sha256sum-amd64.txt
[INFO] Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.34.3+k3s1/k3s
[INFO] Verifying binary download
[INFO] Installing k3s to /usr/local/bin/k3s
[INFO] Skipping installation of SELinux RPM
[INFO] Creating /usr/local/bin/kubectl symlink to k3s
[INFO] Creating /usr/local/bin/crictl symlink to k3s
[INFO] Creating /usr/local/bin/ctr symlink to k3s
[INFO] Creating killall script /usr/local/bin/k3s-killall.sh
[INFO] Creating uninstall script /usr/local/bin/k3s-uninstall.sh
[INFO] env: Creating environment file /etc/systemd/system/k3s.service.env
```

```
curl -sfL https://get.k3s.io | sudo sh -s - server \
--node-ip 192.168.1.123 \
--advertise-address 192.168.1.123 \
--tls-san 192.168.1.123 \
--write-kubeconfig-mode 644
```

Vérifier : systemctl status k3s --no-pager

```
[INFO] Host iptables-save/iptables-restore tools not found
[INFO] Host ip6tables-save/ip6tables-restore tools not found
[INFO] systemd: Starting k3s
root@k3sm:~# systemctl status k3s --no-pager
● k3s.service - Lightweight Kubernetes
   Loaded: loaded (/etc/systemd/system/k3s.service; enabled; preset: enabled)
   Active: active (running) since Mon 2026-02-02 13:21:18 CET; 9min ago
     Invocation: 590ec0f61bd6416d87f9567a1c1f6871
       Docs: https://k3s.io
    Process: 1188 ExecStartPre=/sbin/modprobe br_netfilter (code=exited, status=0/SUCCESS)
    Process: 1189 ExecStartPre=/sbin/modprobe overlay (code=exited, status=0/SUCCESS)
   Main PID: 1191 (k3s-server)
      Tasks: 101
     Memory: 1.6G (peak: 1.6G)
        CPU: 1min 49.187s
      CGroup: /system.slice/k3s.service
              └─1191 "/usr/local/bin/k3s server"
                  ├─1260 "containerd"
                  ├─1803 /var/lib/rancher/k3s/data/ffd82c56f7aadb3263c7708547a6027705a0eb9a638534465002ba309cae9990/bin/cont...
                  ├─1878 /var/lib/rancher/k3s/data/ffd82c56f7aadb3263c7708547a6027705a0eb9a638534465002ba309cae9990/bin/cont...
                  ├─1895 /var/lib/rancher/k3s/data/ffd82c56f7aadb3263c7708547a6027705a0eb9a638534465002ba309cae9990/bin/cont...
```

kubectl get nodes -o wide

```
root@k3sm:~# kubectl get nodes -o wide
NAME      STATUS    ROLES          AGE   VERSION
k3sm      Ready     control-plane  11m   v1.34.3+k3s1
3+deb13-amd64  containerd://2.1.5-k3s1
root@k3sm:~#
```

////

1 curl -sfL https://get.k3s.io

Rôle

Télécharge le **script officiel d'installation de k3s**.

Détail des options curl

- -s → *silent* (pas de barre de progression)
- -f → *fail* (erreur si HTTP ≠ 200)
- -L → suit les redirections

👉 Ce script :

- détecte l'OS
- télécharge le binaire k3s
- crée les services systemd
- configure containerd
- démarre k3s

2 | sudo sh -s -

Rôle

Passe le script téléchargé à **sh via stdin**.

Détail

- sudo → exécution root
- sh → interprète le script
- -s → lit depuis stdin
- - → fin des options

Ça évite d'avoir un fichier temporaire.

3 server

Rôle

Indique à k3s **quel rôle installer**.

- server → **control plane** (API, scheduler, etcd)
- agent → worker

Ici on installe le **master**.

Si on ne met rien :

- k3s installe un **server** par défaut

Mais être explicite est préférable

4 --node-ip 192.168.1.123

Rôle

Force l'adresse IP utilisée par ce nœud.

Pourquoi c'est important

- Une VM peut avoir :
 - plusieurs interfaces
 - IPv6
 - une IP NAT + une IP LAN

Sans cette option :

- k3s choisit "la meilleure IP"
- parfois... la mauvaise

Concrètement

- IP annoncée aux autres nœuds
- IP vue dans kubectl get nodes -o wide

Toujours la forcer explicitement

5 --advertise-address 192.168.1.123

Rôle

Adresse sur laquelle le **serveur Kubernetes s'annonce** aux autres nœuds.

Différence avec --node-ip

- --node-ip → identité du nœud

- --advertise-address → **adresse du control plane**

C'est cette IP que les **workers utilisent** pour joindre l'API server.

Sans ça :

- l'API peut s'annoncer sur :
 - 127.0.0.1
 - une IP interne Docker
 - une mauvaise interface

En cluster multi-nœuds : **indispensable**.

💡 --tls-san 192.168.1.123

Rôle

Ajoute cette IP comme **Subject Alternative Name (SAN)** dans le certificat TLS de l'API Kubernetes.

Les certificats TLS Kubernetes vérifient :

- le **nom DNS**
- ou l'**IP**

Si l'IP n'est pas dans le SAN :

- erreur TLS
- x509: certificate is not valid for...

Cas concrets

- accès à l'API via IP

- accès depuis un autre nœud
- accès depuis un poste externe

👉 Toujours mettre :

- l'IP du master
- éventuellement le hostname

Ex :

```
--tls-san 192.168.1.123 --tls-san k3sm
```

7 --write-kubeconfig-mode 644

Rôle

Définit les **permissions du kubeconfig** généré.

Par défaut :

- /etc/rancher/k3s/k3s.yaml
- permissions restrictives (root only)

Avec 644 :

- lisible par tous les utilisateurs

Pourquoi c'est utile

- exécuter kubectl sans sudo
- copier le kubeconfig vers un autre poste

//////////

Il n'y a plus qu'à récupérer le token qui permettra de joindre les workers : cat /var/lib/rancher/k3s/server/node-token

```
root@k3sm:~# cat /var/lib/rancher/k3s/server/node-token
K10174eaebc91154c58bb3d08d405214be1fa25d1c33400749073f7948a6a09ddec : :server:3ba1ae1a8e4840bf8470c0ad7
```

2) Joindre les workers

Modifiez /etc/hosts comme pour le manager

Et mm config initiale :

```
swapoff -a
```

```
sed -i.bak '/\sswap\s/s/^/#/' /etc/fstab
```

```
tee /etc/modules-load.d/k8s.conf >/dev/null <<'EOF'
```

```
overlay
```

```
br_netfilter
```

```
EOF
```

```
modprobe overlay
```

```
modprobe br_netfilter
```

```
tee /etc/sysctl.d/99-kubernetes-cri.conf >/dev/null <<'EOF'
```

```
net.bridge.bridge-nf-call-iptables = 1
```

```
net.bridge.bridge-nf-call-ip6tables = 1
```

```
net.ipv4.ip_forward = 1
```

```
EOF
```

```
Apt install curl sudo nfs-common
```

```
--
```

```
Puis pour le worker en .124:
```

```
curl -sfL https://get.k3s.io | sudo K3S_URL="https://192.168.1.123:6443"
```

```
K3S_TOKEN="K10174eaebc91154c58bb3d08d405214be1fa25d1c3XXXX49073f7948a6a09ddec::server:3ba1ae1a8e4840bf8470c0ad7acc3X62" sh -s - agent  
--node-ip 192.168.1.124
```

```
root@k3sw1:~# curl -sfL https://get.k3s.io | sudo K3S_URL="https://192.168.1.123:6443" K3S_TOKEN="K10174eaebc91154c5b3d08d405214be1fa25d1c33400749073f7948a6a09ddec::server:3ba1ae1a8e4840bf8470c0ad7acc3362" sh -s - agent --node-ip 192.168.1.124
[INFO] Finding release for channel stable
[INFO] Using v1.34.3+k3s1 as release
[INFO] Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.34.3+k3s1/sha256sum-amd64.txt
[INFO] Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.34.3+k3s1/k3s
[INFO] Verifying binary download
[INFO] Installing k3s to /usr/local/bin/k3s
[INFO] Skipping installation of SELinux RPM
[INFO] Creating /usr/local/bin/kubectl symlink to k3s
[INFO] Creating /usr/local/bin/crictl symlink to k3s
[INFO] Creating /usr/local/bin/ctr symlink to k3s
[INFO] Creating killall script /usr/local/bin/k3s-killall.sh
[INFO] Creating uninstall script /usr/local/bin/k3s-agent-uninstall.sh
[INFO] env: Creating environment file /etc/systemd/system/k3s-agent.service.env
[INFO] systemd: Creating service file /etc/systemd/system/k3s-agent.service
[INFO] systemd: Enabling k3s-agent unit
Created symlink '/etc/systemd/system/multi-user.target.wants/k3s-agent.service' → '/etc/systemd/system/k3s-agent.service'.
[INFO] Host iptables-save/iptables-restore tools not found
[INFO] Host ip6tables-save/ip6tables-restore tools not found
[INFO] systemd: Starting k3s-agent
```

Idem pour le worker en .125

```
root@k3sw2:~# curl -sfL https://get.k3s.io | sudo K3S_URL="https://192.168.1.123:6443" \
  K3S_TOKEN="K10174eaebc91154c58bb3d08d405214be1fa25d1c33400749073f7948a6a09ddec::server:3ba1ae1a8e4840bf8470c0ad72" sh -s - agent \
  --node-ip 192.168.1.125
[INFO] Finding release for channel stable
[INFO] Using v1.34.3+k3s1 as release
[INFO] Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.34.3+k3s1/sha256sum-amd64.txt
[INFO] Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.34.3+k3s1/k3s
|
```

Et on vérifie en retournant sur le manager : kubectl get nodes -o wide

k3sm nodes										
NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME	
k3sm	Ready	control-plane	38m	v1.34.3+k3s1	192.168.1.123	<none>	Debian GNU/Linux 13 (trixie)	6.12.63+deb13-amd64	containerd://2.1.	
k3sw1	Ready	<none>	7m15s	v1.34.3+k3s1	192.168.1.124	<none>	Debian GNU/Linux 13 (trixie)	6.12.63+deb13-amd64	containerd://2.1.	
k3sw2	Ready	<none>	2m35s	v1.34.3+k3s1	192.168.1.125	<none>	Debian GNU/Linux 13 (trixie)	6.12.63+deb13-amd64	containerd://2.1.	
k3s1										

Il est temps de se poser la question : que viens t'on d'installer : et comparer K3S à K8S

Principe clé à garder en tête (k3s vs Kubernetes “vanilla”)

k3s regroupe plusieurs composants Kubernetes dans un seul binaire (k3s), alors que Kubernetes “classique” les déploie comme processus distincts.

Donc avec k3s :

- peu de services systemd
- beaucoup de composants Kubernetes logiques

■ Nœud MASTER – k3sm (k3s server)

Service systemd Rôle

k3s.service Service principal qui embarque plusieurs composants Kubernetes

containerd.service Runtime de conteneurs (lancé/embarqué par k3s)

Un seul service visible, mais il contient beaucoup de choses.

✿ Composants Kubernetes réellement actifs (équivalent k8s)

1 kube-apiserver

- **Cœur du cluster**
- Reçoit toutes les requêtes (kubectl, workers, controllers)
- Exposé sur :6443

Sans lui : **cluster sourd, muet, ... mort**

2 kube-scheduler

- Décide **sur quel worker** un Pod doit être lancé
- Se base sur :
 - ressources (CPU/RAM)
 - contraintes (labels, taints)
 - affinités

Le scheduler **ne lance rien**, il décide seulement.

3 kube-controller-manager

- Boucle de contrôle du cluster
- Vérifie que l'état réel = état désiré

Exemples :

- un Pod meurt → il le recrée
- un Node disparaît → il le marque NotReady
- un ReplicaSet manque un Pod → il en recrée un

etcd (intégré dans k3s)

- Base clé/valeur du cluster
- Stocke :
 - objets Kubernetes (Pods, Services, PVC, Secrets...)
- Dans k3s :
 - etcd est **intégré et embarqué**
 - stockage local sur le master

C'est le "cerveau mémoire" du cluster

Cloud Controller (minimal / stub)

- Présent mais très réduit
 - k3s n'est pas cloud-native par défaut
-

Addons inclus sur le master (via k3s)

Addon	Namespace	Fonction
Traefik	kube-system	Ingress Controller
CoreDNS	kube-system	DNS interne
Flannel	kube-system	Réseau Pod (CNI)

Addon	Namespace	Fonction
kube-proxy	kube-system	Services (iptables)
local-path-provisioner	kube-system	Storage local (RWO uniquement)

Ces addons sont **déployés par le master**, mais **s'exécutent sur tous les nœuds nécessaires**.

Nœuds WORKERS – k3sw1, k3sw2 (k3s agent)

Services système (systemd)

Service systemd Rôle

k3s-agent.service Agent Kubernetes du nœud

containerd.service Runtime conteneur (embarqué par k3s)

Composants Kubernetes actifs sur les workers

kubelet

- Agent local Kubernetes
- Reçoit les ordres du control plane
- Démarré/arrête les Pods
- Surveille leur état

C'est le **chef d'orchestre local**.

2 containerd

- Lance réellement les conteneurs
 - Tire les images
 - Gère les namespaces cgroups
-

3 kube-proxy

- Applique les règles réseau (iptables)
 - Permet :
 - Services ClusterIP
 - NodePort
-

4 Flannel (CNI)

- Gère le réseau Pod ↔ Pod
- VXLAN entre nœuds

Flannel est un **plugin CNI** qui fournit :

- **un réseau IP unique pour tous les Pods**
- **indépendant des nœuds physiques**
- avec un routage transparent entre nœuds

Il crée le **Pod Network**.

5 Traefik (selon scheduling)

- Traefik peut tourner :
 - sur un worker
 - ou sur le master
- Selon le scheduler

👉 Ce n'est pas "lié" au master.

⌚ Comparatif synthétique MASTER vs WORKER

Composant	Master	Worker
kube-apiserver	✓	✗
kube-scheduler	✓	✗
kube-controller-manager	✓	✗
etcd	✓	✗
kubelet	✓ (local)	✓
containerd	✓	✓
kube-proxy	✓	✓

Composant	Master	Worker
Flannel (CNI)	✓	✓
Traefik	✓	✓ (selon pods)
CoreDNS	✓	✗ (souvent master)

Le master décide et stocke, les workers exécutent.

Kubernetes est un système déclaratif piloté par des boucles de contrôle.

Commandes utiles pour illustrer

```
# services système
```

```
systemctl status k3s
```

```
systemctl status k3s-agent
```

```
# composants k8s
```

```
kubectl get pods -n kube-system
```

```
kubectl get nodes -o wide
```

```
# voir où tourne quoi
```

```
kubectl get pods -n kube-system -o wide
```

Ok, on déploie un nginx tout simple pour voir ce que ça donne :
ça va nous permettre de valider que :

- le cluster fonctionne
- le scheduling est OK
- le réseau Pod ↔ Service ↔ Node fonctionne
- Traefik est bien là

Nous pouvons déployer nginx de plusieurs façons, en déclarant différents objets :

Pod (unité de base)

Rôle

- Un ou plusieurs conteneurs
- **Aucune haute dispo**
- **Aucune auto-réparation**

Utile en debug/test par exemple

Replicaset

Rôle

- Garantit un **nombre de Pods**
- Crée/supprime des Pods

Quand l'utiliser

- quasi jamais directement

Deployment l'embarque déjà

Deployment (le standard) :

Rôle

- Gère ReplicaSet
- Rolling update
- Rollback
- Scaling

Cas d'usage

- **99 % des apps stateless** (Nginx, WordPress, API...)

Nous allons donc créer un répertoire sur le manager qui va accueillir nos fichiers :

Ce répertoire est gitté pour le versionning et cloné localement (en prod)

```
/opt/k8s/  
|   |-- nginx/  
|       |-- deployment.yaml  
|       |-- service.yaml  
|       `-- ingress.yaml
```

```
mkdir -p /opt/k8s/nginx
```

```
cd /opt/k3s/nginx/
```

Et on commence par le fichier deployment.yaml

(.yaml = .yml mais dans la doc officielle kubernetes c'est .yaml ... côté docker .yml mais se .yamlise)

nginx-deployment.yaml

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: nginx
```

```
  labels:
```

```
    app: nginx
```

```
spec:
```

```
  replicas: 1
```

```
  selector:
```

```
    matchLabels:
```

```
      app: nginx
```

```
  template:
```

```
    metadata:
```

```
labels:  
  app: nginx  
  
spec:  
  containers:  
    - name: nginx  
      image: nginx:1.25-alpine  
      ports:  
        - containerPort: 80
```

Et nous pouvons lancer ce déploiement :

```
kubectl apply -f nginx-deployment.yaml
```

À ce stade le pod fonctionne et vous pouvez le vérifier avec ces commandes :

```
kubectl get pods -o wide  
kubectl describe pod <nom_du_pod>
```

```

root@k3sm:~# mkdir -p /opt/k3s/nginx
root@k3sm:~# cd /opt/k3s/nginx/
root@k3sm:/opt/k3s/nginx# nano nginx-deployment.yaml
root@k3sm:/opt/k3s/nginx# kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx created
root@k3sm:/opt/k3s/nginx# kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE      IP          NODE   NOMINATED NODE   READINESS GATES
nginx-7c89c7f547-vqbgd   1/1     Running   0          49s    10.42.1.3   k3sw1  <none>        <none>
root@k3sm:/opt/k3s/nginx# kubectl describe pod nginx-7c89c7f547-vqbgd
Name:           nginx-7c89c7f547-vqbgd
Namespace:      default
Priority:       0
Service Account: default
Node:          k3sw1/192.168.1.124

```

Avant d'aller plus loin, déchifrons ce fichier que l'on vient d'utiliser :

Voici le détail **ligne par ligne**, avec le “pourquoi” et les implications.

apiVersion: apps/v1

- **Version d'API Kubernetes** utilisée pour cet objet.
 - apps/v1 = API stable pour les contrôleurs type Deployment/StatefulSet/DaemonSet.
 - Si tu mets une mauvaise apiVersion → Kubernetes refuse le manifest.
-

kind: Deployment

- **Type d'objet**.
- Deployment = contrôleur “stateless” qui gère :
 - ReplicaSet

- Pods
 - rolling update / rollback
 - Concrètement : tu ne crées pas “un Pod”, tu crées une **intention** (“je veux N Pods”).
-

metadata:

- Métagreffées de l'objet (comme l'en-tête).
 - Sert à l'identifier et à le classer.
-

name: nginx

- **Nom unique** du Deployment dans le namespace.
 - Utilisé partout : kubectl get deploy nginx, logs, labels, etc.
 - Doit respecter des règles DNS (minuscules, tirets...).
-

labels:

- **Étiquettes** (clé/valeur) sur l'objet Deployment lui-même.
- Ne sert pas directement au routage, mais utile pour :
 - filtrer
 - organiser
 - policies
 - outils (monitoring, GitOps...)

app: nginx

- Un label “standard” : app=nginx
 - Convention fréquente : regrouper toutes les ressources liées à une app.
-

spec:

- La **spécification** : l'état désiré.
 - C'est le cœur déclaratif.
-

replicas: 1

- Nombre de Pods désirés.
 - Ici : **1 instance** de Nginx.
 - Si tu mets 3 :
 - Kubernetes maintient **3 Pods identiques**
 - (mais attention aux volumes partagés si un jour tu en ajoutes)
-

selector:

- Très important : dit au Deployment **quels Pods il “possède”**.
- Il doit correspondre aux labels des Pods du template.

👉 Si le selector ne correspond pas au template → comportement cassé ou refus selon cas.

matchLabels:

- Forme simple de selector (exact match).
 - Il existe aussi matchExpressions (plus avancé).
-

app: nginx

- Le Deployment gère les Pods qui ont app=nginx.
-

template:

- Modèle de Pod : ce que le Deployment va créer.
 - C'est littéralement un objet Pod "emballé" dans un contrôleur.
-

metadata:

- Métagdonnées du **Pod** (pas du Deployment).
-

labels:

- Labels du Pod.
- Crucial car :
 - doit matcher le selector
 - sert aux Services pour sélectionner les Pods

app: nginx

- Label appliqué aux Pods.
 - Ici il matche bien le selector.
-

spec:

- Spécification du Pod : conteneurs, volumes, réseau, sécurité...
-

containers:

- Liste des conteneurs dans le Pod.
 - Un Pod peut contenir :
 - 1 conteneur (cas simple)
 - plusieurs (sidecar : logs, proxy, etc.)
-

- name: nginx

- Nom du conteneur **dans le Pod**.
 - Utile si plusieurs conteneurs : kubectl logs pod -c nginx
-

image: nginx:1.25-alpine

- Image OCI à exécuter.

- nginx = repository
- 1.25-alpine = tag (version + base alpine)

⚠ Point “prod” :

- un tag peut être modifié (rare mais possible)
 - en prod stricte, on pin sur un digest @sha256:...
-

ports:

- Déclare les ports exposés par le conteneur.
 - C'est surtout **informatif** et utilisé par certains outils.
 - Ça **n'ouvre pas** automatiquement l'accès extérieur.
-

- containerPort: 80

- Port sur lequel Nginx écoute **dans le conteneur**.
 - Sert ensuite à :
 - Services (targetPort: 80)
 - probes
 - documentation
-

🧠 **Ce que ce manifest crée réellement**

Quand tu appliques ce YAML :

1. Kubernetes crée un **Deployment**
 2. Le Deployment crée un **ReplicaSet**
 3. Le ReplicaSet crée **1 Pod**
 4. Le Pod lance **1 conteneur Nginx** qui écoute sur **80**
-

⚠ Ce que ce manifest NE fait PAS

- il n'expose pas Nginx à l'extérieur (pas de Service / Ingress)
- il ne met pas de volumes (pas de persistance)
- il ne met pas de healthchecks (probes)
- il ne met pas de limites CPU/RAM

Ok ok, on va utiliser un objet de type « service » pour exposer notre nginx

Nous allons déclarer un service de type **clusterIP**, mais celui-ci n'expose pas réellement à l'extérieur mais reste interne au cluster. La base pour vérifier que CNI/DNS fonctionnent notamment.

nginx-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
```

et on applique : kubectl apply -f nginx-svc.yaml

Ce service accessible en interne au cluster sera utile lors de l'exposition via traefik

Pour visualiser, nous allons plutôt dans un premier temps utiliser un service nodeport qui permettra d'exposer nginx via un port, et le service sera accessible sur n'importe quelle ip du cluster sur le port en question.
nginx-svc-nodeport.yaml

```
apiVersion: v1
kind: Service
```

```
metadata:  
  name: nginx-nodeport  
  
spec:  
  type: NodePort  
  
  selector:  
    app: nginx  
  
  ports:  
    - port: 80  
      targetPort: 80  
      nodePort: 30080
```

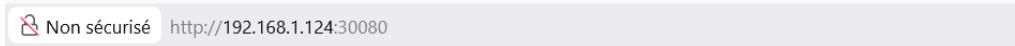
port correspond au port du SERVICE dans le cluster (ex 80)

targetport : port réel du conteneur (ex 8080)

nodeport :port exposé (ex 30080 ATTENTION doit être dans la plage 30000–32767)

et nous pouvons l'appliquer : kubectl apply -f nginx-svc-nodeport.yaml

Puis se rendre sur son navigateur : l'ip d'un nœud :le port



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

On est déjà pas mal mais peut-on faire mieux 😊

Nous allons utiliser l'ingress traefik (déjà installé dans K3S) pour accéder à notre app

nginx-ingress.yaml

```
apiVersion: networking.k8s.io/v1
```

```
kind: Ingress
```

```
metadata:
```

```
  name: nginx
```

```
spec:
```

```
  rules:
```

```
    - host: nginx.afpadevops.lab
```

```
      http:
```

```
paths:  
- path: /  
  pathType: Prefix  
backend:  
  service:  
    name: nginx  
  port:  
    number: 80
```

Comme d'hab, on applique : kubectl apply -f nginx-ingress.yaml

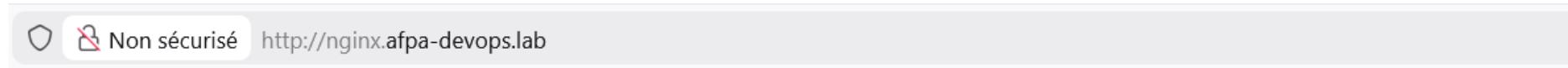
On vérifie que l'ingress est bien vu : kubectl get ingress -A

```
root@k3sm:/opt/k3s/nginx# kubectl get ingress  
NAME      CLASS      HOSTS          ADDRESS          PORTS      AGE  
nginx     traefik    nginx.afpa-devops.lab  192.168.1.123,192.168.1.124,192.168.1.125  80        6m27s
```

ajoutez l'entrée dans /etc/hosts (ou "C:\Windows\System32\drivers\etc\hosts")

192.168.1.123 nginx.afpadevops.lab

Et testez depuis votre navigateur :



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

On passe au **déploiement WordPress**

1 Volume NFS côté Kubernetes

- PersistentVolume (PV)
- PersistentVolumeClaim (PVC)

2 Test du volume avec un Pod simple

- Valider RWX
- Valider permissions

3 Déploiement WordPress

- Deployment
- Montage du PVC sur /var/www/html/wp-content

4 Service + Ingress Traefik

- Exposition HTTP
- Puis DB externe MariaDB

Il nous faut un serveur NFS avec partage monté :

✗ Serveur NFS : 192.168.1.120

✗ Export NFS : /srv/nfs/wp-content

✗ Accès RWX

Création du PV **PersistentVolume**

Un **PV** représente un disque réel de l'infrastructure, abstrait pour Kubernetes.

Il est :

- **fourni par l'admin infra**
- **global au cluster**
- **indépendant des applications**

App → demande du stockage

demande :

taille

access mode

storageClass

Kubernetes cherche un PV compatible

liaison 1 PV ↔ 1 PVC

Utilisation par un Pod

Pod → PVC → PV → stockage réel

Le Pod **ne connaît pas** :

- le serveur NFS
- le chemin réel
- le protocole

➔ **abstraction totale.**

Dans /opt/k3s on crée un répertoire « pv » pour **PersistentVolume**

/opt/k3s/pv/pv-nfs-content20go.yaml

apiVersion: v1

kind: PersistentVolume

```
metadata:  
  name: pv-nfs-content  
  labels:  
    type: nfs  
spec:  
  capacity:  
    storage: 20Gi  
  accessModes:  
    - ReadWriteMany  
  persistentVolumeReclaimPolicy: Retain  
  storageClassName: nfs-static  
  nfs:  
    server: 192.168.1.120  
    path: /srv/nfs/wp-content  
  mountOptions:  
    - vers=3  
    - rw  
    - hard  
    - timeo=600  
    - retrans=5
```

Et on applique : kubectl apply -f pv-nfs-content20go.yaml

Et on vérifie : kubectl get pv

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	VOLUMEATTRIBUTESCLASS	REASON	A
pv-nfs-content	20Gi	RWX	Retain	Available		nfs-static	<unset>		2

Il faut un status « available » sinon problème !

Dans /opt/k3s, on crée un répertoire pour wordpress et on se glisse dedans

On va commencer par créer un namespace (isolation logique) dédié wordpress et y mettre **PVC + app**, tout en gardant en tête que **le PV est cluster-wide** (pas dans un namespace).

00-namespace-wordpress.yaml

apiVersion: v1

kind: Namespace

metadata:

 name: wordpress

Appliquer :

kubectl apply -f 00-namespace-wordpress.yaml

Vérifier :

kubectl get ns wordpress

```
00-namespace-wordpress.yaml
root@k3sm:/opt/k3s/wordpress# kubectl apply -f 00-namespace-wordpress.yaml
namespace/wordpress created
root@k3sm:/opt/k3s/wordpress# kubectl get ns wordpress
NAME      STATUS   AGE
wordpress  Active   10s
```

On va réclamer de l'espace dans le PV en créant un PVC

01-pvc-content.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-wp-content
  namespace: wordpress
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 20Gi
```

```
storageClassName: nfs-static
```

Bien sur on applique et on vérifie :

```
root@k3sm:/opt/k3s/wordpress# kubectl apply -f 01-pvc-wp-content.yaml
persistentvolumeclaim/pvc-wp-content created
root@k3sm:/opt/k3s/wordpress# kubectl get pvc -n wordpress
NAME           STATUS    VOLUME          CAPACITY   ACCESS MODES  STORAGECLASS  VOLUME ATTRIBUTESCLASS AGE
pvc-wp-content Bound    pv-nfs-content  20Gi      RWX          nfs-static   <unset>          6s
root@k3sm:/opt/k3s/wordpress#
```

On crée notre déploiement :

```
02-wordpress-deploy.yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: wordpress
```

```
  namespace: wordpress
```

```
  labels:
```

```
    app: wordpress
```

```
spec:
```

```
  replicas: 1
```

```
selector:  
matchLabels:  
    app: wordpress  
  
template:  
metadata:  
labels:  
    app: wordpress  
  
spec:  
containers:  
    - name: wordpress  
      image: wordpress:6.5-apache  
      ports:  
        - containerPort: 80  
      env:  
        # --- Connexion MariaDB externe ---  
        - name: WORDPRESS_DB_HOST  
          value: "192.168.1.130:3306"  # <-- IP/port de ta VM MariaDB  
        - name: WORDPRESS_DB_NAME  
          value: "wordpress"      # <-- MODIFIE  
        - name: WORDPRESS_DB_USER
```

```
    value: "wp_user"      # <-- MODIFIE
- name: WORDPRESS_DB_PASSWORD
  value: "P@ssw0rd34"    # <-- MODIFIE (idéalement Secret ensuite)
```

```
# Bonnes pratiques minimales: probes
```

```
readinessProbe:
```

```
  httpGet:
    path: /
    port: 80
    initialDelaySeconds: 10
    periodSeconds: 10
```

```
livenessProbe:
```

```
  httpGet:
    path: /
    port: 80
    initialDelaySeconds: 30
    periodSeconds: 20
```

```
volumeMounts:
```

```
  - name: wp-content
```

```
mountPath: /var/www/html/wp-content
```

```
volumes:
```

```
- name: wp-content
```

```
  persistentVolumeClaim:
```

```
    claimName: pvc-wp-content
```

On crée le service (en clusterIP que traefik va cibler) :

03-wordpress-svc.yaml

```
apiVersion: v1
```

```
kind: Service
```

```
metadata:
```

```
  name: wordpress
```

```
  namespace: wordpress
```

```
  labels:
```

```
    app: wordpress
```

```
spec:  
  type: ClusterIP  
  selector:  
    app: wordpress  
  ports:  
    - name: http  
      port: 80  
      targetPort: 80
```

Puis on gère l'ingress

04-wordpress-ingress.yaml

```
apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
  name: wordpress  
  namespace: wordpress  
spec:
```

```
rules:  
  - host: wordpress.afpa-devops.lab  
    http:  
      paths:  
        - path: /  
          pathType: Prefix  
        backend:  
          service:  
            name: wordpress  
          port:  
            number: 80
```

Voilà, on applique nos fichiers dans le bon ordre :

```
kubectl apply -f /opt/k3s/wordpress/02-wordpress-deploy.yaml  
kubectl apply -f /opt/k3s/wordpress/03-wordpress-svc.yaml  
kubectl apply -f /opt/k3s/wordpress/04-wordpress-ingress.yaml
```

et on vérifie :

```
kubectl get pods -n wordpress -o wide  
kubectl get svc -n wordpress
```

```
kubectl get ingress -n wordpress
```

Avant de tester il nous faut ajouter l'entrée fournie à traefik dans le fichier hosts

```
192.168.1.123 wordpress.afpa-devops.lab
```

Et également initialiser la DB pour wordpress

Attention, ce n'est pas DANS K3S mais sur la vm db !! il doit y avoir correspondance avec les infos renseignées dans 02-wordpress-deploy.yaml

Via un script ou commandes (mysql -u root)

```
init-wordpress-db.sql
```

-- Création de la base WordPress

```
CREATE DATABASE IF NOT EXISTS wordpress
```

```
CHARACTER SET utf8mb4
```

```
COLLATE utf8mb4_unicode_ci;
```

-- Création de l'utilisateur WordPress

```
CREATE USER IF NOT EXISTS 'wp_user'@'%'
```

```
IDENTIFIED BY 'P@ssw0rd34';
```

-- Droits nécessaires pour WordPress

```
GRANT ALL PRIVILEGES
```

```
ON wordpress.*
```

```
TO 'wp_user'@'%';
```

```
-- Application des privilèges
```

```
FLUSH PRIVILEGES;
```

Pour exécuter le script : mysql -u root -p < /root/init-wordpress-db.sql

Et permettre à la db de recevoir des connections externes :

Dans /etc/mysql/mariadb.conf.d/50-server.cnf :

```
bind-address = 0.0.0.0
```

Redémarrer le serveur pour prise en compte de la conf :

```
systemctl restart mariadb
```

et vérifier : ss -lntp | grep 3306

```
root@k3sdb:~# ss -lntp | grep 3306
LISTEN 0      80          0.0.0.0:3306      0.0.0.0:*      users:(("mariadb",pid=2152,fd=26))
```

```
GNU nano 8.4                               /etc/mysql/mariadb.conf.d/50-server.cnf

#
# These groups are read by MariaDB server.
# Use it for options that only the server (but not clients) should see

# this is read by the standalone daemon and embedded servers
[server]

# this is only for the mariadb daemon
[mariadb]

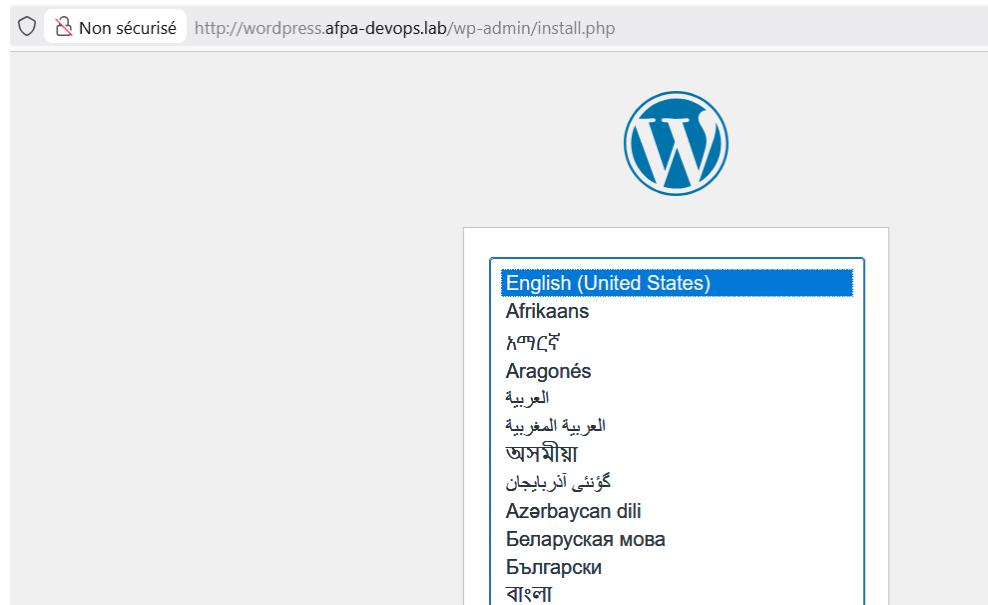
#
# * Basic Settings
#

#user          = mysql
pid-file      = /run/mysqld/mysqld.pid
basedir        = /usr
#datadir       = /var/lib/mysql
#tmpdir        = /tmp

# Broken reverse DNS slows down connections considerably and name resolve is
# safe to skip if there are no "host by domain name" access grants
#skip-name-resolve

# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address   = 0.0.0.0|
```

Et go navigateur



A screenshot of the WordPress dashboard. The top navigation bar includes icons for WordPress, home, DevopsKub, refresh (6), comments (0), and create. The main menu on the left is titled 'Tableau de bord' and lists: Accueil, Mises à jour, Articles, Médias, Pages, Commentaires, Apparence, Extensions, and Comptes. A message in the center says 'WordPress 6.9 est disponible ! Veuillez mettre à jour maintenant.' Below it, the title 'Tableau de bord' is followed by a large black box containing the text 'Bienvenue sur WordPress !' and 'En savoir plus sur la version 6.5.5'.

DB et stockage sont bien pris en compte

