S5.B.01 Phase 4
Déploiement de services

Maxence Lagourgue

13 janvier 2026

Table des matières

I Introduction

I.A Outils

Dans cette partie, les outils utilisés seront :

- Rancher pour la gestion des clusters
- RKE2 pour la mise en œuvre Kubernetes des nœuds de travail
- k3s pour le cluster Rancher
- kubectl pour la gestion des ressources
- Helm pour la gestion des applications

Plus tard, si nous avons le temps, nous utiliserons Ansible pour automiser la chaîne de production Rancher.

I.B Machines

Les machines utilisées au cours de ce projet seront :

- applicatif ⇒ k3s cluster + Rancher Server
- K8SA2 (k8s1) ⇒ RKE2 cluster + Master, Etcd, Worker Nodes
- K8SB2 (k8s2) ⇒ RKE2 cluster + Worker nodes + Backup
- K8SC2 (k8s3) ⇒ ???

I.C Configuration générale

I.C.1 Nom de domaines

Dans toutes les VMs impliquées dans le cluster Kubernetes, la configuration suivante sera définie.

Listing I.1 – /etc/hosts

```
10.0.1.3        rancher.rancher
10.0.1.4        master.rancher
10.0.1.5        worker.rancher
10.0.1.6        gitlab.rancher nailloux.gitlab.com nailloux.registry.com
```

I.C.2 Contrôle des ressources

Pour monitorer les ressources des VMs, nous ne pouvons pas nous servir des indications données par Proxmox VM car le Guest Agent est désactivé. Nous aurons donc de mauvaises indications pour la RAM par exemple. Je conseille donc :

```
wget https://github.com/fastfetch-cli/fastfetch/releases/download/2.56.1/fastfetch-linux-amd64.deb && dpkg -i fastfetch-linux-amd64.deb
```

II Installation de Rancher dans un cluster k3s

Pour utiliser Rancher, plusieurs méthodes d'installation s'offrent à nous. L'une avec docker, l'autre en tant que noeud Kubernetes. Les autres installations reposent sur l'utilisation d'un Cloud Provider ainsi que Terraform donc inutile dans notre cas.
Exemple de tutorial : Tutorial Rancher 2025
Faire le gitlab en tant qu'application kubernetes/rancher.

II.A Installation de k3s

Listing II.1 – Installation de k3s

```
curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION="v1.34.3+k3s1" sh -s - server --resolv-conf /run/systemd/resolve/resolv.conf
```

Il est possible de définir les paramètres suivants mais ils sont susceptibles de générer des bugs

```
--bind-address 10.0.1.3 --advertise-address 10.0.1.3 --node-ip 10.0.1.3
```

II.A.1 Accès distant au cluster (Optionnel)

<IP_OF_LINUX_MACHINE> est l'IP de la machine distante sur laquelle se trouve le cluster.

```
scp root@<IP_OF_LINUX_MACHINE>:/etc/rancher/k3s/k3s.yaml ~/.kube/config
```

```
nano ~/.kube/config
```

II.A.2 Installation d'helm

```
sudo apt-get install curl gpg apt-transport-https --yes

curl -fsSL https://packages.buildkite.com/helm-linux/helm-debian/gpgkey | gpg --dearmor |
sudo tee /usr/share/keyrings/helm.gpg > /dev/null

echo "deb [signed-by=/usr/share/keyrings/helm.gpg]
https://packages.buildkite.com/helm-linux/helm-debian/any/ any main" | sudo tee
/etc/apt/sources.list.d/helm-stable-debian.list

sudo apt-get update
sudo apt-get install helm
```

II.A.3 Installation de kubectl

```
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.34/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg

sudo chmod 644 /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.34/deb/ /' | sudo tee
/etc/apt/sources.list.d/kubernetes.list
sudo chmod 644 /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update
sudo apt-get install -y kubectl
```

II.A.4 Installation de Calicoctl

```
curl -L https://github.com/projectcalico/calico/releases/download/v3.30.4/calicoctl-linux-
    amd64 -o calicoctl

chmod +x ./calicoctl

mv ./calicoctl /usr/bin/calicoctl
```

II.A.5 Création d'un Déploiement Rancher

<Hostname> correspond au nom de domaine utilisé pour contacter le pod rancher.

```
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml



kubectl create namespace cattle-system
kubectl config set-context --current --namespace=cattle-system

kubectl apply -f
https://github.com/cert-manager/cert-manager/releases/download/v1.19.2/cert-manager.crds.yaml

helm repo add rancher-latest https://releases.rancher.com/server-charts/latest

helm repo add jetstack https://charts.jetstack.io

helm repo update

helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --create-namespace
```

```
helm install rancher rancher-latest/rancher \
  --namespace cattle-system \
  --set hostname=rancher.rancher \
  --set replicas=1 \
  --set bootstrapPassword=testpassword \
  --set additionalTrustedCAs=true
```

Il faut maintenant attendre car l'installation nécessite quelques minutes. On peut vérifier avec `kubectl␣get␣pods␣-n␣cattle-system`
Une fois fait, on se connecte à la page et on récupère le mot de passe :

```
kubectl get secret --namespace cattle-system bootstrap-secret -o
go-template='{{.data.bootstrapPassword|base64decode}}{{"\n"}}'
```

On définit un nouveau mot de passe qui est `qJHiA@wwaagi46U`.

II.A.6 Pour arrêter/pauser Rancher

```
kubectl scale --replicas=0 deployment/rancher -n cattle-system
```

Cela permet d'arrêter temporairement le pod Rancher.

II.A.7 Ajout d'un certificat Root

```
kubectl -n cattle-system create secret generic tls-ca-additional --from-file=ca-additional
    .pem=./ca-additional.pem
```

II.B Réinitialisation du cluster

Pour revenir à l'état 0 du cluster, il est possible de :

```
rm -rf /var/lib/rancher/k3s/server/db/etcd
/usr/local/bin/k3s-killall.sh
systemctl restart k3s.service
```

III Création d'un cluster

III.A Enrôlement de machines

III.A.1 Variables d'environnement

Parce que rancher nécessite des variables d'environnements qui ne sont pas forcément définis de base :

```
echo "CRI_CONFIG_FILE=/var/lib/rancher/rke2/agent/etc/crictl.yaml
CONTAINERD_ADDRESS=unix:///run/k3s/containerd/containerd.sock
KUBECONFIG=/etc/rancher/rke2/rke2.yaml" >> /etc/environment

echo "export PATH=$PATH:/var/lib/rancher/rke2/bin" > /etc/profile.d/rancher.sh
```

III.A.2 Ajout d'une mâchine Master

Pour cela catil faut aller dans la section **Clusters** ⇒ **<Cluster>** ⇒ **Registration**

Listing III.1 – Machine master

```
curl --insecure -fL https://rancher.rancher/system-agent-install.sh | sudo  sh -s - --
    server https://rancher.rancher --label 'cattle.io/os=linux' --token
    b6rh9zgx8m8jgwg6q6nm7h5frb7d6v87wgt8scppljfvqbvjt85kqk --ca-checksum
    b5ede295e2fdd2b453ae8cee700b60185f393914281b4bc90008c1e3f4eb8e5a --etcd --controlplane
    --worker --address 10.0.1.4 --internal-address 10.0.1.4
```

III.A.3 Ajout d'une machine Worker only

Listing III.2 – Machine worker

```
curl --insecure -fL https://rancher.rancher/system-agent-install.sh | sudo  sh -s - --
    server https://rancher.rancher --label 'cattle.io/os=linux' --token
    tvg69x5vkm9szzlzsj6qqkx7wggzrgvt2grc755nth29h2ncjbgthz --ca-checksum
    fdc9c50ea58442994213e96883b6a5ca39227fc7d4116e60fa1026c123f56583 --worker --address
    10.0.1.5 --internal-address 10.0.1.5
```

Si souci, il faut aller voir la section Troubleshooting.

IV Replication du Noeud Master

```
In the context of RKE2/K3s (the tech stack you are currently using), "replicating" a
    master node means adding additional Control Plane + Etcd nodes to form a High
    Availability (HA) cluster.

Since you are working across different subnets, the process requires a specific
    configuration to ensure the nodes can communicate and that the certificates are valid
    for your cross-subnet IPs.
1. Requirements for Master Replication

    An odd number of masters: To maintain "Quorum" (the ability to make decisions), you
        need 3, 5, or 7 master nodes. 3 is the standard for HA.

    The Shared Token: You need the token from your first master (found at /var/lib/rancher
        /rke2/server/node-token or /var/lib/rancher/k3s/server/node-token).

    A Fixed Registration Address: As discussed, this should be a DNS name pointing to your
         master IPs.

2. Configure the First Master (Initial Node)

If you already installed the first master, ensure its /etc/rancher/rke2/config.yaml (or
    k3s) has the tls-san parameter. This ensures the SSL certificates allow the "
    Registration DNS" you'll use.
YAML

token: my-shared-secret-token
tls-san:
  - k8s-api.example.com  # The DNS name for your masters
  - 10.0.1.3             # IP of Master 1
  - 10.0.2.3             # IP of Master 2 (Subnet B)

3. Join the Second Master (The Replication)

On your server in the second subnet, install the same version of RKE2/K3s, but point it to
     the first master.

The config.yaml for Master #2:
YAML

server: https://k8s-api.example.com:9345  # Point to the first node
token: my-shared-secret-token             # Must match node #1
```

```
tls-san:
  - k8s-api.example.com
# Important: Ensure this node knows where the "real" DNS is
resolv-conf: "/run/systemd/resolve/resolv.conf"

4. Handling the "No Load Balancer" Problem

Since you cannot use a load balancer across your subnets, you must handle the Initial Join
     carefully.

    Direct Join: For the very first join of Master #2, you can temporarily point the
        server parameter directly to the IP of Master #1: server: https://10.0.1.3:9345.

    DNS Update: Once Master #2 is up, add its IP to your k8s-api.example.com DNS record.

    Master #3: When you join the third master, point its server address to the DNS name.
        It will now be able to reach either Master #1 or Master #2 to join.

5. Verification

Once the replication is complete, check that all masters are participating in the etcd
    cluster and the control plane:
Bash

# Check node status
kubectl get nodes

# Check etcd health (RKE2 example)
/var/lib/rancher/rke2/bin/kubectl \
  --kubeconfig /etc/rancher/rke2/rke2.yaml \
  get componentstatuses

Summary of HA across Subnets

    Etcd: Will automatically replicate data from Master 1 to Master 2 over the network.

    API Server: Will be active on both nodes.

    Workers: Will automatically switch between Master 1 and Master 2 if one subnet goes
        offline.

Would you like the specific shell commands to extract the node-token from your first
    master to prepare for the join?
```

V Deploiement de l'application

Tout d'abord, avant de déployer quoi que ce soit il faut convertir le docker-compose.yml en fichier de déploiement Kubernetes, un manifest.

```
kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/master/
    deploy/local-path-storage.yaml

kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storageclass.
    kubernetes.io/is-default-class":"true"}}}'
```

VI Déploiement de composants annexes

Dans le même temps que le déploiement de l'application du club photo nailloux, l'infrastructure de l'entreprise nécessite d'autres services, qui seront détaillés et déployés ici.
En particulier un Gitlab et un Registre Docker.

VI.A Gitlab

VI.A.1 Phase 1 : Creation des volumes

Dans la continuité des volumes docker, il existe ce qu'on appelle les **Persistent Volume Claims**.

- gitlab-etc 2Gi /etc/gitlab Configuration (gitlab.rb)
- gitlab-opt 20Gi+ /var/opt/gitlab Database and Git Repositories
- gitlab-log 5Gi /var/log/gitlab Logs

VI.A.2 Phase 2 : Creation de la ConfigMap

Pour définir le nom de domaine qu'on va utiliser pour accéder au serveur Gitlab, on créé une **ConfigMap** avec la procédure suivante :
Dans Rancher : Cluster → Resources → ConfigMaps.
On créé une nouvelle configuration nommée **gitlab-config**. On définit une clé **gitlab.rb** avec comme valeur :

```
external_url 'http://nailloux.gitlab.com'
gitlab_rails['time_zone'] = 'Europe/Paris'
#gitlab_rails['initial_root_password'] = "qJHiA@wwaagi46U"
```

VI.A.3 Phase 3 : Deploiement des pods

Dans la section Workloads -> Deployment on créé un Déploiement comme ceci

VI.A.3.a Variables d'environnments

```
GITLAB_ROOT_EMAIL="admin@example.com" GITLAB_ROOT_PASSWORD="strongpassword"
```

VI.A.3.b Conteneur

Nom : Gitlab
Docker Image : gitlab/gitlab-ce :latest

VI.A.3.c Ouverture de ports

Il faut ensuite ouvrir des ports, qui impliqueront la création d'un service.

- ClusterIP, http, Port 80
- ClusterIP, ssh, 22
- ClusterIP, ssh2, 2222

VI.A.3.d Storage (Volume Mounts)

Il faut ensuite monter les volumes créés et la ConfigMap donc les pods.

- PVC gitlab-etc → /etc/gitlab
- PVC gitlab-opt → /var/opt/gitlab
- PVC gitlab-log → /var/log/gitlab
- ConfigMap gitlab-config → Key gitlab.rb mounted to /etc/gitlab/gitlab.rb

VI.A.3.e Security Context

Il faut savoir que Rancher par défaut, cloisonne l'utilisateur chargé de lancer le service. Cela mène à plusieurs problèmes de permissions si le service nécessite de lire des fichiers protégés.
C'est le cas ici avec Gitlab. Dans la section Security Context, il faut donc modifier les paramètres suivantes

- Privileged : True
- Run as User : 0 (Root)

VI.A.4 Phase 4 : Attente

GitLab prend 5 à 10 minutes pour démarrer la première fois.
Il ne faut donc surtout pas ajouter de "Health Check" car cela mettra le pod/déploiement en défaut.
On peut accéder aux logs avec

```
kubectl logs -f deployment/gitlab
```

VI.A.5 Phase 5 : Création d'une redirection de paquets avec un Ingress

Dans Service Discovery > Ingresses.
Create
Host : nailloux.gitlab.com
Path : / → Service : gitlab → Port : 80.
On peut définir le HTTPS pour l'Ingress dans la section Certificates en ajoutant un.

VI.A.6 Phase 6 : Récupération du mot de passe

Une fois que le serveur Gitlab est lancé, (environ 5 minutes), on peut récupérer le mot de passe généré par défaut comme ceci :

```
root@k8sc2:~# kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
gitlab-54d7466479-vcd9w   1/1     Running   0          18m
root@k8sc2:~# kubectl exec gitlab-54d7466479-vcd9w -- cat /etc/gitlab/
    initial_root_password
# WARNING: This password is only valid if ALL of the following are true:
```

```
    # •          You set it manually via the GITLAB_ROOT_PASSWORD environment variable
    #            OR the gitlab_rails['initial_root_password'] setting in /etc/gitlab/gitlab.
        rb
    # •          You set it BEFORE the initial database setup (typically during first
        installation)
    # •          You have NOT changed the password since then (via web UI or command line)
    #
    #          If this password doesn't work, reset the admin password using:
    #          https://docs.gitlab.com/security/reset_user_password/#reset-the-root-password

    Password: JKk4H/BOVuXOFgi899ZRsN4HBnLzmWLQAveHOkmtAyE=

    # NOTE: This file is automatically deleted after 24 hours on the next reconfigure run.
```

Pour interagir avec le pod :

```
    kubectl exec -it gitlab-54d7466479-vcd9w -- bash
```

VI.B Docker Registry

Le serveur gitlab possède déjà une option pour disposer d'un serveur Docker Registry mais on ne va pas utiliser cette méthode ici.

VI.B.1 Phase 1 : Creation des PVC

Afin de stocker les images docker (volumineuses), nous allons créer des volumes persistants (PVC).
Il faut aller dans Storage > PersistentVolumeClaims.
On créé un volume registry-data avec les paramètres suivants :

- Size : 20GiB
- Classe : local-path (créée précédemment)

VI.B.2 Phase 2 : Deploiement des pods

Dans la section Workload > Deployments puis Create.

1. Nom : docker-registry.
2. Image : registry :3
3. Stockage : Monter le PVC registry-data dans /var/lib/registry.

Listing VI.1 – Variables d'environnements :

```
  REGISTRY_HTTP_ADDR: :0.0.0.0:5000 (The internal port).

  REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /var/lib/registry.

  REGISTRY_HTTP_HEADERS_Access-Control-Allow-Origin : '[''*'']'
  REGISTRY_HTTP_HEADERS_Access-Control-Allow-Methods : '[''HEAD'', ''GET'', ''OPTIONS'', ''
      DELETE'']'
  REGISTRY_HTTP_HEADERS_Access-Control-Allow-Headers : '[''Authorization'', ''Accept'', ''
      Cache-Control'']'
```

Sera ensuite exposé le port 5000 en tcp, avec un service ClusterIP, nommé https.

VI.B.3 UI

Afin d'interfacer proprement notre serveur Docker-Registry, nous allons faire appel à une autre image docker **joxit/docker-registry-ui** qui implique un nouveau déploiement chargé d'afficher une UI.
Il faudra définir les paramètres suivants :

- Ouverture du port 80, en TCP, avec ClusterIP, nom : http
- Ouverture du port 443, en TCP, avec ClusterIP, nom : https

```
NGINX_PROXY_PASS_URL: http://docker-registry:5000
DELETE_IMAGES: true
SINGLE_REGISTRY: true
```

VI.B.4 Phase 3 : Création d'un Ingress avec HTTPS

Docker, par défaut, refuse de communiquer avec une API/Serveur n'utilisant pas HTTPS ou n'ayant pas un certificat SSL valide. Il est plutôt aisé de mettre en place un service HTTPS avec Rancher/Kubernetes au moyen des **Ingress**.

```
export KEY_FILE=key.pem
export CERT_FILE=cert.pem
export HOST=nailloux.registry.com

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout $KEY_FILE -out $CERT_FILE -
    subj "/CN=$HOST/O=$HOST" -addext "subjectAltName = DNS:$HOST"
```

Une fois le certificat généré, il suffit d'aller dans la section Secret : Create > TLS Certificate et de renseigner la clé privée et le certificat.
Maintenant, il faut aller dans **Service Discovery** -> Ingresses -> Create.
On peut renseigner les informations suivantes :

- Request Host : nailloux.registry.com
- Service : docker-registry-ui on Port 80
- HTTPS
- Labels & Annotations : "nginx.ingress.kubernetes.io/proxy-body-size" = 0 pour ne pas subir le 413 too large error.
- Certificates : Secret Name, on déroule et sélectionne celui créé précédemment et met comme Domain Name : nailloux.registry.com

VI.B.5 Phase 4 : Testing the Registry

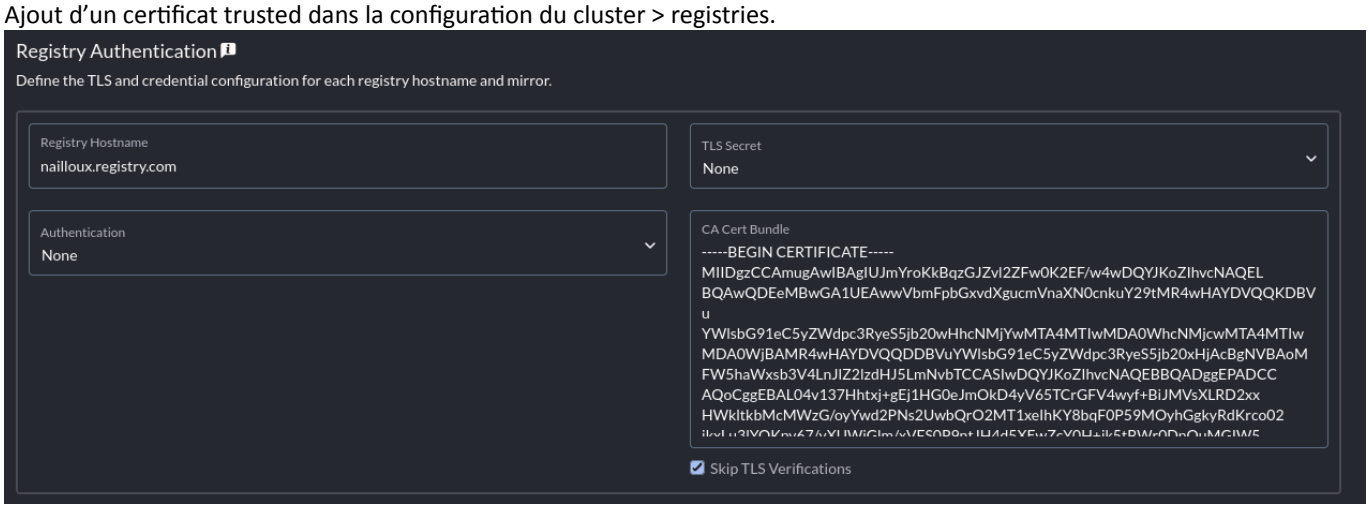
Maintenant il est possible de publier des images dans le registre en taggant les images de la manière suivante :

```
docker tag nginx:latest nailloux.registry.com/nginx:latest
docker push nailloux.registry.com/nginx:latest
```

VII Troubleshooting

VII.A Problème de certificats SSL

Ajout d'un certificat trusted dans la configuration du cluster > registries.



VII.B Cannot allocate new block due to per host block limit

```
calicoctl datastore migrate lock
calicoctl ipam check -o report.json
calicoctl ipam release --from-report report.json
calicoctl datastore migrate unlock
```

VII.C DNS

Lorsqu'on utilise systemd-resolved, le fichier /etc/resolv.conf utilise une adresse locale inutilisable par le pod **coreDNS**. Il faut donc spécifier le fichier où systemd-resolved garde ses serveurs DNS.

```
echo 'services:
  kubelet:
    extra_args:
      resolv-conf: "/run/systemd/resolve/resolv.conf"' > /etc/rancher/rke2/config.yaml
```

On peut aussi faire la méthode suivante, moins flexible, en ajoutant en dur un hôte comme on le ferait dans le fichier **/etc/hosts**. Solution à ne pas faire car on ne résoud pas réellement le souci DNS.

```
  kubectl -n cattle-system logs -l app=cattle-cluster-agent

  export KUBE_EDITOR="nano"
  kubectl edit configmap rke2-coredns-rke2-coredns -n kube-system

Corefile: |-
  .:53 {
      errors
      health {
          lameduck 10s
      }
      ready
      kubernetes  cluster.local  cluster.local in-addr.arpa ip6.arpa {
          pods insecure
          fallthrough in-addr.arpa ip6.arpa
          ttl 30
      }
      hosts {
          10.0.1.3 rancher.rancher
          fallthrough
      }
      prometheus  0.0.0.0:9153
      forward  . /etc/resolv.conf
      cache  30
      loop
      reload
      loadbalance
  }


  kubectl rollout restart deployment rke2-coredns-rke2-coredns -n kube-system
  kubectl scale deployments/cattle-cluster-agent -n cattle-system --replicas=0
  kubectl scale deployments/cattle-cluster-agent -n cattle-system --replicas=1
```