S5.B.01 Phase 4
Déploiement de services

Maxence Lagourgue

8 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

```
curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION="v1.24.10+k3s1" sh -s - server
--cluster-init --bind-address 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/v3.31.3/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
```

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.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 Cluster pour l'application Nailloux

III.B Cluster pour les composants divers, Git, DockerRegistry...

III.B.0.a Enregistrement

```
curl --insecure -fL https://rancher.rancher/system-agent-install.sh | sudo  sh -s - --
    server https://rancher.rancher --label 'cattle.io/os=linux' --token 272
    xjzq28nm5xp4cpjjrxt4zqrwvsmftq45xs5bx4vv7kk65mh72pv --ca-checksum
    fdc9c50ea58442994213e96883b6a5ca39227fc7d4116e60fa1026c123f56583 --etcd --controlplane
    --worker --address 10.0.1.6 --internal-address 10.0.1.6
```

III.C Enrôlement de machines

```
echo "CRI_CONFIG_FILE=/var/lib/rancher/rke2/agent/etc/crictl.yaml
CONTAINERD_ADDRESS=unix:///run/k3s/containerd/containerd.sock
#PATH=/usr/local/sbin /usr/local/bin /usr/sbin /usr/bin /sbin /bin
KUBECONFIG=/etc/rancher/rke2/rke2.yaml" >> /etc/environnment

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

Pour cela il 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
    tvg69x5vkm9szzlzsj6qqkx7wggzrgvt2grc755nth29h2ncjbgthz --ca-checksum
    fdc9c50ea58442994213e96883b6a5ca39227fc7d4116e60fa1026c123f56583 --etcd --controlplane
    --worker --address 10.0.1.4 --internal-address 10.0.1.4
```

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 DNS issue :

```
  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  . /run/systemd/resolve/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
```

IV Creation d'un Workload

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"}}}'
```

V Cluster components

V.A Gitlab

V.A.1 Phase 1 : Create the Storage (The Foundation)

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

V.A.2 Phase 2 : Create the 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.
Create a new one named gitlab-config.
Key : gitlab.rb
Value (Paste this) :

```
    external_url 'http://nailloux.gitlab.com'
    gitlab_rails['time_zone'] = 'Europe/Paris'
```

V.A.3 Phase 3 : Deploy GitLab (The Workload)

Create a Deployment with the following specific settings.

V.A.3.a Basic Info

Name : gitlab
Docker Image : gitlab/gitlab-ce :latest

V.A.3.b Networking (Ports)

Add these "Service" ports :
HTTP : Port 80 (mapped to 80 in container)
SSH : Port 2222 (mapped to 22 in container) — Note : We use 2222 because 22 is usually taken by your actual Node.

V.A.3.c Storage (Volume Mounts)

Mount your PVCs and the ConfigMap :
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

V.A.3.d Security Context (CRITICAL)

As we discovered, GitLab needs permissions to fix the folders created by local-path.
Privileged : True
Run as User : 0 (Root)

V.A.4 Phase 4 : Handle the ”Wait”

GitLab takes 5 to 10 minutes to start the first time.
Do not add Health Checks yet. If you add a ”Liveness Probe” now, Kubernetes will see the app isn’t responding (because it’s still installing) and kill it, creating a ”CrashLoopBackOff.”
Monitor the logs : “‘bash kubectl logs -f deployment/gitlab
Wait until you see the ”GitLab Reconfigured !” message and the internal Nginx starts.

V.A.5 Phase 5 : Accessing GitLab (The Ingress)

Once the logs show GitLab is up, you need to route traffic from your 10.x or 100.x IP to the pod.
Go to Service Discovery > Ingresses.
Create a new Ingress.
Host : git.yourdomain.com
Path : / → Service : gitlab → Port : 80.

V.A.6 Phase 6 : Get your Password

Once the site loads, you need the initial root password.
Use the Rancher ”Execute Shell” on the GitLab pod.

```
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: fU16xdsVRA/de4Mf/N2geVuifm8kDIqJkeB3zQVnwcE=
```

```
    # 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
```

V.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.

V.B.1 Phase 1 : Create Storage for your Images

Docker images take up a lot of space. You need a persistent place to store them so they don't disappear when the pod restarts.
Go to Storage > PersistentVolumeClaims.
Create a PVC named registry-data.
Size : At least 20Gi (depending on how many images you plan to store).
Storage Class : local-path.

V.B.2 Phase 2 : Deploy the Registry Workload

Go to Workload > Deployments and click Create.
Name : docker-registry.
Image : registry :2.
Storage :
Mount the PVC registry-data to /var/lib/registry.
Environment Variables :

```
    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'']'
```

Expose : ClusterIP, https, 5000, tcp

V.B.3 UI

use the image joxit/docker-registry-ui.

```
            - containerPort: 80
                name: http
                protocol: TCP
                _serviceType: ClusterIP
              - containerPort: 443
                name: https
                protocol: TCP
```

```
                  _serviceType: ClusterIP

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

V.B.4 Phase 3 : Expose it (The SSL Problem)

Docker refuses to push or pull from a registry over plain HTTP unless you configure every single node as an ”insecure registry.” It is much easier to set up an Ingress with SSL.
Generate a Certificate : Since you are in a private network, you can use a ”Secret” to store a self-signed cert, or use Cert-Manager if you have it.
Dans la section Secret : Create > TLS Certificate et il suffit de renseigner la clé privée et publique
Create Ingress :
Host : registry.rancher (or whatever your DNS handles).
Service : docker-registry-ui on Port 80.
Protocol : HTTPS.
Il faut ensuite aller dans Labels & Annotations et définir la valeur ”nginx.ingress.kubernetes.io/proxy-body-size” à 0 pour ne pas subir le 413 too large error.

V.B.5 Phase 4 : Testing the Registry

From your development machine (where you build your Dockerfiles) :
Tag your image : docker tag nailloux-app :v1 registry.rancher/nailloux-app :v1
Push it : docker push registry.rancher/nailloux-app :v1

VI Troubleshooting

VI.A 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.

```
 nano /etc/rancher/rke2/config.yaml
root@k8sc2:~# echo 'services:
  kubelet:
    extra_args:
      resolv-conf: "/run/resolvconf/resolv.conf"' > /etc/rancher/rke2/config.yaml
```