



TP04 Infrastructures Cloud

Par David TEJEDA, Vincent LAGOGUE, Tom THIOULOUSE, Thomas
PEUGNET, Alexis PLESSIAS

Cozystack sur MicroK8S

Pour installer notre CP, on choisit la machine avec le plus de performance **100.79.188.4** :

On installe l'utilitaire **k9s** :

```
wget  
https://github.com/derailed/k9s/releases/download/v0.32.7/k9s\_linux\_amd64.de  
b && sudo apt install ./k9s_linux_amd64.deb
```

K9s permet d'avoir des compléments d'informations et de surveiller les ressources Kubernetes. Il permet aussi de naviguer plus facilement entre les ressources Kubernetes. De plus, l'outil est léger et ne nécessite qu'un terminal.

Déployer microk8s

On installe et on déploie **microk8s** :

```
sudo snap install microk8s --classic
sudo microk8s enable hostpath-storage
sudo microk8s enable dns

mkdir bin
echo "microk8s kubectl \${@}" > ~/bin/kubectl
echo "microk8s helm \${@}" > ~/bin/helm
chmod +x ~/bin/*

sudo usermod -aG microk8s $USER

echo "export PATH=\$HOME/bin:\$PATH" >> ~/.bashrc
mkdir ~/.kube
sudo microk8s config > /home/$USER/.kube/config
sudo chown -R $USER: /home/$USER/.kube
```

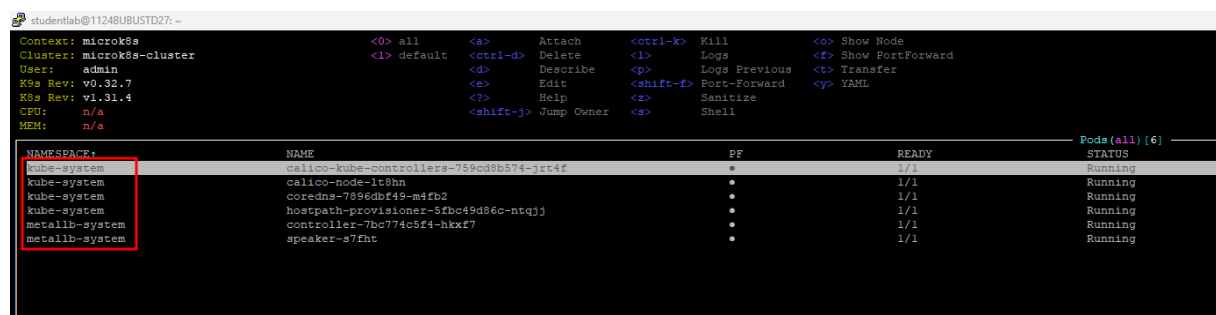
On se constate que l'accès se fait avec l'utilisateur « k9s » :

```
studentlab@1124BUBUSTD27:~/bin$ kubectl
La commande « kubectl » n'a pas été trouvée, mais peut être installée avec :
sudo snap install kubectl
studentlab@1124BUBUSTD27:~/bin$ ./kubectl
Insufficient permissions to access MicroK8s.
You can either try again with sudo or add the user studentlab to the 'microk8s' group:

    sudo usermod -a -G microk8s studentlab
    sudo chown -R studentlab ~/.kube

After this, reload the user groups either via a reboot or by running 'newgrp microk8s'.
```

Quand on lance **k9s** on remarque que nos modules ont bien été lancé aussi :



The screenshot shows the k9s terminal interface. At the top, there's a header with system information: Context: microk8s, Cluster: microk8s-cluster, User: admin, K9s Rev: v0.32.7, K8s Rev: v1.31.4, CPU: n/a, MEM: n/a. Below this is a table of running pods. The 'NAMESPACE' column is highlighted with a red box, and the 'NAME' column is also highlighted. The pods listed are:

NAMESPACE	NAME	PF	READY	STATUS
kube-system	calico-kube-controllers-759cd3b574-jrt4f	*	1/1	Running
kube-system	calico-node-1t8hn	*	1/1	Running
kube-system	coredns-7896dbf49-m4fb2	*	1/1	Running
kube-system	hostpath-provisioner-5fbc49d86c-ntqj	*	1/1	Running
metallb-system	controller-7bc774c5f4-hkxf7	*	1/1	Running
metallb-system	speaker-s7fnc	*	1/1	Running

Déployer et utiliser Cozystack

Déploiement

Un objet **NodePort** est intéressant pour utiliser **Cozystack** car il permet de rendre un service accessible à partir de n'importe quel nœud du cluster Kubernetes sur un port spécifique.

Cela permet aux utilisateurs ou outils locaux de se connecter directement au tableau de bord ou aux API de Cozystack sans avoir besoin de configurations supplémentaires comme un Ingress ou un LoadBalancer.

On configure notre environnement pour cozystack :

```
# Éléments de configuration
---
apiVersion: v1
kind: Namespace
metadata:
  name: cozy-system
  labels:
    name: cozy-system
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: cozystack
  namespace: cozy-system
data:
  bundle-name: "paas-hosted"
```

Ensuite on télécharge et on applique les changements :

```
wget https://github.com/aenix-io/cozystack/raw/v0.18.0/manifests/cozystack-
installer.yaml
kubectl apply -f config.yaml
```

Attention, il faut modifier le fichier **cozystack-installer.yaml** pour ajouter l'adresse du cluster IP :

```
studentlab@1124BUBUSTD27:~$ kubectl get services -n default
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kubernetes    ClusterIP    10.152.183.1   <none>         443/TCP    7d19h
```

On ajoute aussi le port 443 qui est utilisé par **Microk8s** pour l'administration.

```

# Source: cozy-installer/templates/cozystack.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cozystack
  namespace: cozy-system
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cozystack
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  template:
    metadata:
      labels:
        app: cozystack
    spec:
      hostNetwork: true
      serviceAccountName: cozystack
      containers:
        - name: cozystack
          image: "ghcr.io/aenix-io/cozystack/cozystack:v0.18.0"
          env:
            - name: KUBERNETES_SERVICE_HOST
              value: 10.152.183.1
            - name: KUBERNETES_SERVICE_PORT
              value: "443"
            - name: K8S_AWAIT_ELECTION_ENABLED
              value: "1"
            - name: K8S_AWAIT_ELECTION_NAME
              value: cozystack
            - name: K8S_AWAIT_ELECTION_LOCK_NAME

```

Dans le fichier `cozystack-installer.yaml`

```
kubectl apply -f cozystack-installer.yaml
```

Après quelques minutes on remarque que de nouveaux *namespaces* sont créés :

[illegible]

Utilisation de la console

On redirige le service vers le port 30000 :

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: cozy-dashboard
  labels:
    name: cozy-dashboard
---
apiVersion: v1
kind: Service
metadata:
  name: dashboard-nodeport
  namespace: cozy-dashboard
spec:
  type: NodePort
  selector:
    app.kubernetes.io/component: frontend
    app.kubernetes.io/instance: dashboard
    app.kubernetes.io/name: kubeapps
  ports:
    - protocol: TCP
      port: 80
      targetPort: http
      nodePort: 30000
```

On applique la configuration. On obtient ensuite l'interface web :



Cette interface est accessible en local et depuis le *tailscale* et le VPN de l'école. On se connecte depuis une adresse IP d'une machine du cluster pour avoir accès à l'interface.

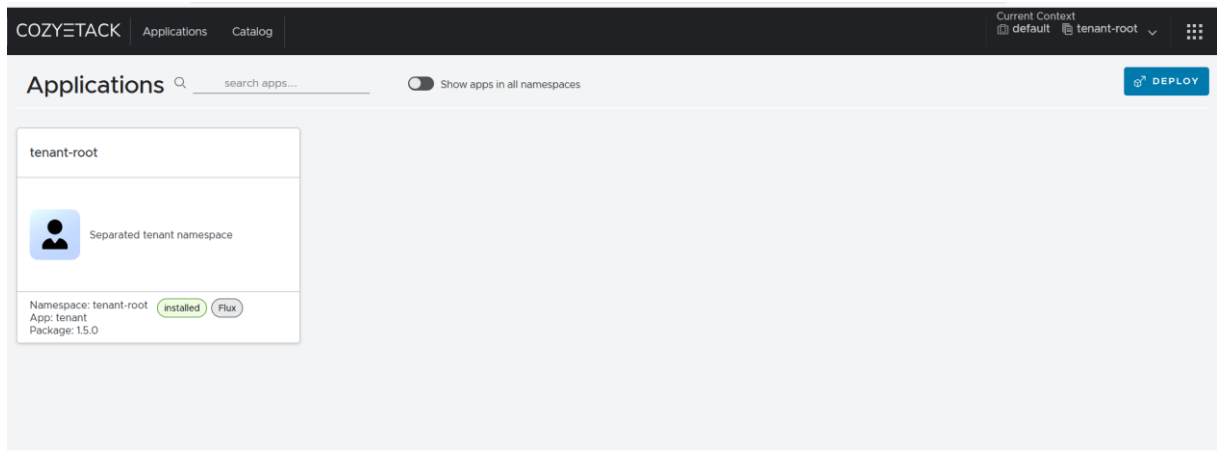
Pour récupérer le token on utilise cette commande :

```
kubectl get secret tenant-root -n tenant-root -o jsonpath='{.data.token}'  
| base64 --decode
```

On peut aussi le retrouver dans l'onglet *secrets* dans **k9s** :

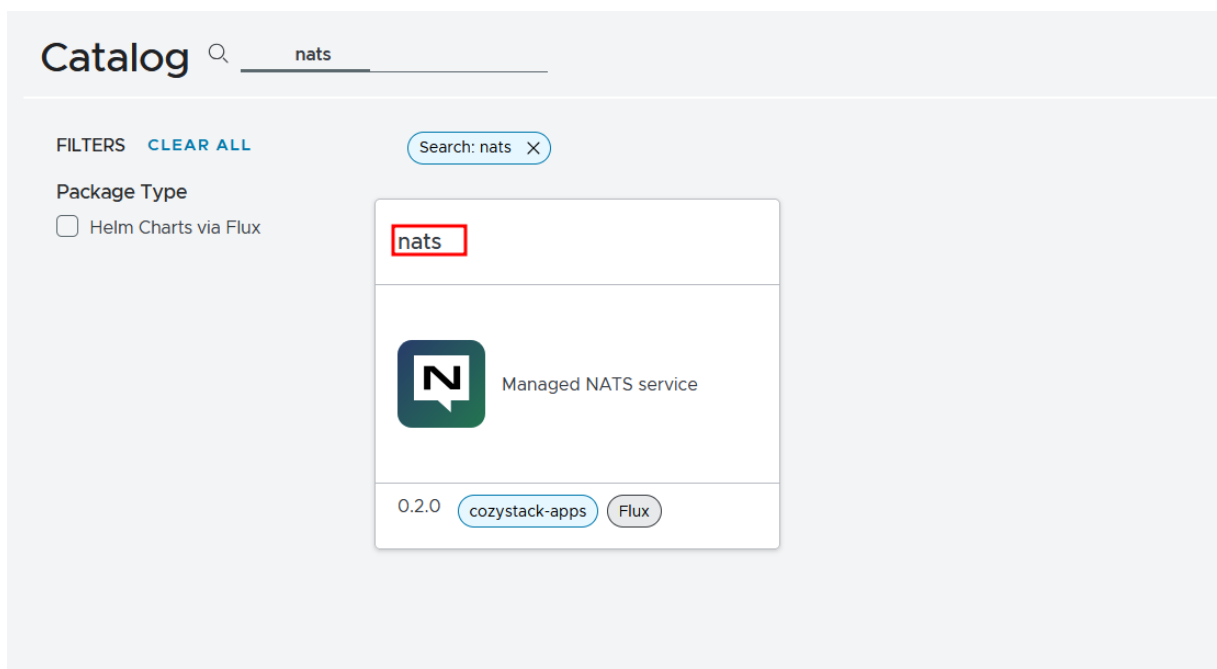
```
k9s  
:secrets
```

On sélectionne ensuite le *secret* en question et on peut récupérer le *token*.



Panneau de contrôle de CozyStack

On s'intéresse maintenant à la création d'un service managé :



Dans le catalogue on trouve et on installe le service **nats**.

The screenshot shows the configuration page for the 'nats' Helm chart. The 'App Version' is 1.4.1 and the 'Package Version' is 0.2.0. The 'Visual editor' displays a table of values:

Key	Type	Description	Default Value	Current Value
external	boolean	external	false	false
replicas	number	replicas	2	2
storageClass	string	storageClass		

A red box highlights the 'DEPLOY 0.2.0' button at the bottom left.

Le service apparait désormais dans les applications :

The screenshot shows the 'Applications' catalog in COZYETACK. Two application cards are visible:

- nats-nats**: Managed NATS service. Namespace: tenant-root. App: nats v1.4.1. Package: 0.2.0. Status: installed.
- tenant-root**: Separated tenant namespace. Namespace: tenant-root. App: tenant. Package: 1.5.0. Status: installed.




De plus dans le *namespace* **tenant-root** on observe que des *Pods* ont été créées.

NAME	IP	READY	STATUS	RESTARTS	CPU	MEM	%CPU/R	%CPU/L	%MEM/R	%MEM/L	IP	NODE	AGE
nats-nats-0		3/3	Running	0	3	17	n/a	n/a	n/a	n/a	10.1.176.105	1124bubustd27	2m32s
nats-nats-1		3/3	Running	0	3	17	n/a	n/a	n/a	n/a	10.1.176.108	1124bubustd27	2m39s
nats-nats-box-594c7fc697-jndgr		1/1	Running	0	0	0	n/a	n/a	n/a	n/a	10.1.176.105	1124bubustd27	2m29s

On crée aussi une ressource pour le service **mysql**

COZYSTACK Applications Catalog

Applications 🔍 search apps... ☐ Show apps in all namespaces

mysql-database	nats-nats	tenant-root
 Managed MariaDB service	 Managed NATS service	 Separated tenant namespace
Namespace: tenant-root App: mysql v11.0.2 Package: 0.5.2	Namespace: tenant-root App: nats v1.4.1 Package: 0.2.0	Namespace: tenant-root App: tenant Package: 1.5.0
<input checked="" type="radio"/> installed <input type="radio"/> Flux	<input checked="" type="radio"/> installed <input type="radio"/> Flux	<input checked="" type="radio"/> installed <input type="radio"/> Flux

Par défaut, le service est interne à Kubernetes. Mais on peut le rendre public via un *NodePort* comme on l'a fait pour le service CozyStack.

Intégration de Kubernetes avec Tailscale

Installation de l'opérateur Tailscale

On ajoute le dépôt Helm de *Tailscale* :

```
helm repo add tailscale https://pkgs.tailscale.com/helmcharts
helm repo update
```

On ajoute une ligne dans nos ACLs sur *Tailscale* (<https://login.tailscale.com/admin/acls/file>) :

```
"tagOwners": {
  "tag:k8s-operator": [],
  "tag:k8s": ["tag:k8s-operator"],
}
```

Dans notre interface d'administration (<https://login.tailscale.com/admin/settings/oauth>) on crée un client OAuth avec les autorisations :

```
scope auth_keys : write ;
scope devices:core : write
```

On récupère ensuite une clé et un secret qu'on exporte

```
export client_id=...
export client_secret=...
```

Ensuite on lance la commande :

```
helm upgrade \
--install \
tailscale-operator \
tailscale/tailscale-operator \
--namespace=tailscale \
--create-namespace \
--set-string oauth.clientId="$client_id" \
--set-string oauth.clientSecret="$client_secret" \
--wait
```

```
studentlab@1124BUBUSTD27:~$ tailscale status | grep operator
100.68.238.104 tailscale-operator tagged-devices linux -
```

On a bien un nouveau nœud tailscale

On supprime le NodePort :

```
kubectl delete svc -n cozy-dashboard dashboard-nodeport
```

Lorsqu'on supprime le NodePort, on perd l'accès à notre serveur. Ce dernier fonctionne toujours en local mais n'est plus accessible via des appels externes.

```
studentlab@112488857027:~$ curl http://cozy-dashboard-dashboard
<!doctype html><html lang="en"><head><script defer="defer" src="/static/js/main.b8a09575.js"></script><link href="/static/css/main.21ace40b.css" rel="stylesheet"></head><body><noscript>Yo
u need to enable JavaScript to run this app.</noscript><div id="root"></div></body></html>studentlab@112488857027:~$
```

Différence avec l'objet NodePort utilisé précédemment :

Le NodePort expose un service Kubernetes sur un port spécifique de chaque nœud du cluster. Il nécessite que le client soit sur le même réseau ou que le port soit ouvert sur le pare-feu. Alors que le SDN ne dépend pas de l'IP publique du nœud ou d'un port ouvert.

On pourrait avoir plusieurs cas d'usage pour un Service publié sur un SND :

- Accès sécurisé pour les utilisateurs distants : Permet à des utilisateurs ou développeurs de se connecter de manière sécurisée au tableau de bord sans VPN supplémentaire.
- Connexion inter-cluster : Facilite la communication entre plusieurs clusters Kubernetes répartis sur différents réseaux.
- Accès restreint : Publie un service uniquement à des clients approuvés sur le réseau Tailscale, limitant les risques de sécurité.
- Gestion simplifiée : Évite les configurations réseau complexes comme les règles de pare-feu, NAT ou ouverture de ports.

Pour exposer le tableau de bord directement sur Internet, on peut mettre en place :

- Un Ingress Controller : Configure un objet Ingress dans Kubernetes pour gérer les accès externes au service via un DNS public et HTTPS.
- Un Load Balancer : Utilise un service Kubernetes de type LoadBalancer, qui alloue une IP publique pour exposer le tableau de bord.
- Un Proxy inverse : Ajoute un outil comme NGINX, Traefik ou un autre proxy pour sécuriser et gérer les connexions entrantes.
- Certificats SSL/TLS : Pour sécuriser les connexions, ajoute un certificat via Let's Encrypt ou un autre fournisseur.

Intégration de la PKI dans le cluster Kubernetes

Installation des ressources avec Helm

Le namespace contenant cert-manager est « cozy-cert-manager ».

On installe le dépôt de Helm de wikimedia :

```
helm repo add wikimedia-charts https://helm-charts.wikimedia.org/stable
helm repo update wikimedia-charts
helm install \
cfssl-issuer-crds wikimedia-charts/cfssl-issuer-crds

helm install \
cfssl-issuer wikimedia-charts/cfssl-issuer \
--namespace cozy-cert-manager
```

Context: microk8s	<0> all	<6> default	<a>	Attach	<ctrl-x>	Kill	<0> Show Node
Cluster: microk8s-cluster	<1> cozy-cert-manager	<ctrl-d>	<d>	Delete	<i>	Logs	<f> Show PortForward
User: admin	<2> cozy-dashboard	<d>	<d>	Describe	<p>	Logs Previous	<t> Transfer
K9s Rev: v0.32.7	<3> tenant-root	<e>	<e>	Edit	<shift-f>	Port-Forward	<y> YAML
K9s Rev: v1.31.5	<4> cozy-system	<t>	<t>	Help	<z>	Sanitize	
CPU: 14%	<5> cozy-rabbitmq-operator	<shift-j>	<shift-j>	Jump Owner	<s>	Shell	
MEM: 46%							

Pods (cozy-cert-manager) [4]										
NAME	FF	READY	STATUS	RESTARTS	CPU	MEM	%CPU/R	%CPU/L	%MEM/R	%MEM/L IP
cert-manager-565f63769f-f2njk	*	1/1	Running	0	3	23	n/a	n/a	n/a	n/a 10.1.176.88
cert-manager-cainjector-56fAdf7486-62wn9	•	1/1	Running	0	1	78	n/a	n/a	n/a	n/a 10.1.176.90
cert-manager-webhook-7cc695b96-s9rkp	•	1/1	Running	0	1	11	n/a	n/a	n/a	n/a 10.1.176.89
cfssl-issuer-6bf46565f7-9qcwk	•	1/1	Running	0	0	0	0	0	0	0 10.1.176.117

On voit bien que nos pods tournent dans le NS cozy-cert-manager

Configuration de l'objet ClusterIssuer

On récupère le secret auth_key de la PKI généré lors de la création de la PKI.

On récupère aussi le certificat de la CA intermédiaire de la PKI. Ce dernier est nécessaire afin de garantir des échanges sécurisés dans notre cluster.

On crée un objet ConfigMap nommée **internal-ca-chain** contenant le certificat de la CA intermédiaire

```
kubectl -n cozy-cert-manager create configmap internal-ca-chain --from-file=ca-bundle.crt=./intermediate-ca.pem
```

On crée un patch :

```
{
  "spec": {
    "template": {
      "spec": {
        "$setElementOrder/containers": [
          {
```

```

      "name": "cfssl-issuer"
    }
  ],
  "containers": [
    {
      "name": "cfssl-issuer",
      "volumeMounts": [
        {
          "mountPath": "/etc/pki/tls/certs/",
          "name": "internal-ca-chain"
        }
      ]
    }
  ],
  "volumes": [
    {
      "configMap": {
        "name": "internal-ca-chain"
      },
      "name": "internal-ca-chain"
    }
  ]
}
}
}
}

```

Et on l'applique :

```

kubectrl -n cozy-cert-manager patch deployment cfssl-issuer --patch-file
patch.json

```

On crée un CRD :

```

---
apiVersion: cfssl-issuer.wikimedia.org/v1alpha1
kind: ClusterIssuer
metadata:

```

```
name: cfssl-internal-efrei-ca
spec:
  authSecretName: "0b2b7cda34425ec452e71d8ea0fd4676"
  bundle: false
  label: efrei
  profile: "host"
  url: https://100.127.217.69:8000
```

On applique le fichier YAML. Et on vérifie l'état de notre objet :

```
studentlab@1124BUBUSTD27:~$ kubectl get crd
NAME                                         CREATED AT
addresspools.metallb.io                    2025-01-14T15:07:41Z
alerts.notification.toolkit.fluxcd.io       2025-01-22T09:37:10Z
apprepositories.kubeapps.com                2025-01-22T09:37:22Z
backups.k8s.mariadb.com                     2025-01-22T09:38:51Z
backups.postgresql.cnpg.io                 2025-01-22T09:38:11Z
bfdprofiles.metallb.io                     2025-01-14T15:07:41Z
bgpadvertisements.metallb.io                2025-01-14T15:07:41Z
bgpconfigurations.crd.projectcalico.org     2025-01-14T14:11:30Z
bgppeers.crd.projectcalico.org              2025-01-14T14:11:30Z
bgppeers.metallb.io                        2025-01-14T15:07:41Z
bindings.rabbitmq.com                       2025-01-22T09:37:27Z
blockaffinities.crd.projectcalico.org       2025-01-14T14:11:30Z
buckets.source.toolkit.fluxcd.io            2025-01-22T09:37:10Z
caliconodestatuses.crd.projectcalico.org    2025-01-14T14:11:30Z
certificaterequests.cert-manager.io         2025-01-22T09:37:23Z
certificates.cert-manager.io                2025-01-22T09:37:23Z
challenges.acme.cert-manager.io             2025-01-22T09:37:24Z
clickhouseinstallations.clickhouse.altinity.com 2025-01-22T09:37:22Z
clickhouseinstallationtemplates.clickhouse.altinity.com 2025-01-22T09:37:22Z
clickhousekeeperinstallations.clickhouse-keeper.altinity.com 2025-01-22T09:37:23Z
clickhouseoperatorconfigurations.clickhouse.altinity.com 2025-01-22T09:37:23Z
clusterimagecatalogs.postgresql.cnpg.io     2025-01-22T09:38:11Z
clusterinformations.crd.projectcalico.org    2025-01-14T14:11:30Z
clusterissuers.cert-manager.io              2025-01-22T09:37:24Z
clusterissuers.cfssl-issuer.wikimedia.org   2025-01-27T22:04:38Z
clusters.postgresql.cnpg.io                 2025-01-22T09:38:11Z
communities.metallb.io                      2025-01-14T15:07:41Z
connections.k8s.mariadb.com                 2025-01-22T09:38:51Z
connectors.tailscale.com                    2025-01-27T20:45:48Z
```

On teste ensuite notre service en lui soumettant une requête à signer :

```
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: test-host-cert-lab-efrei
spec:
  secretName: test-host-cert-lab-efrei
  duration: 2160h # 90 jours
  renewBefore: 360h # 15 jours
  privateKey:
```

algorithm: RSA
encoding: PKCS1
size: 2048
usages:
- server auth
dnsNames:
- "test-host-cert-lab-efrei"
uris:
- spiffe://cluster.local/ns/default/sa/default
ipAddresses:
- "100.127.217.69"
issuerRef:
name: "cfssl-internal-efrei-ca"
kind: ClusterIssuer
group: cfssl-issuer.wikimedia.org

```
janv. 29 08:58:50 1124BUBUSTD25 sudo[140572]: 2025/01/29 08:58:50 [DEBUG] getting info
janv. 29 08:58:50 1124BUBUSTD25 sudo[140572]: 2025/01/29 08:58:50 [INFO] 100.79.188.4:28243 - "POST /api/v1/cfssl/info" 200
janv. 29 08:59:50 1124BUBUSTD25 sudo[140572]: 2025/01/29 08:59:50 [DEBUG] checking label
janv. 29 08:59:50 1124BUBUSTD25 sudo[140572]: 2025/01/29 08:59:50 [DEBUG] getting info
janv. 29 08:59:50 1124BUBUSTD25 sudo[140572]: 2025/01/29 08:59:50 [INFO] 100.79.188.4:29198 - "POST /api/v1/cfssl/info" 200
```

On voit sur les logs que le service a bien reçu des informations.

Enfin quand on regarde sur **k9s**, on a bien un nouveau secret :

```
Describe(cozy-cert-manager/0b2b7cda34425ec452e71d8ea0fd4676)
Name: 0b2b7cda34425ec452e71d8ea0fd4676
Namespace: cozy-cert-manager
Labels: controller.cert-manager.io/fao=true
Annotations: cert-manager.io/alt-names: test-host-cert-lab-efrei
              cert-manager.io/certificate-name: test-host-cert-lab-efrei
              cert-manager.io/common-name:
              cert-manager.io/ip-sans: 100.127.217.69
              cert-manager.io/issuer-group: cfssl-issuer.wikimedia.org
              cert-manager.io/issuer-kind: ClusterIssuer
              cert-manager.io/issuer-name: cfssl-internal-efrei-ca
              cert-manager.io/uri-sans: spiffe://cluster.local/ns/default/sa/default

Type: kubernetes.io/tls

Data
====
tls.crt: 1168 bytes
tls.key: 1679 bytes
```