

PostgreSQL

Déploiement de l'opérateur PostgreSQL

Simon ELBAZ (selbaz@linagora.com)

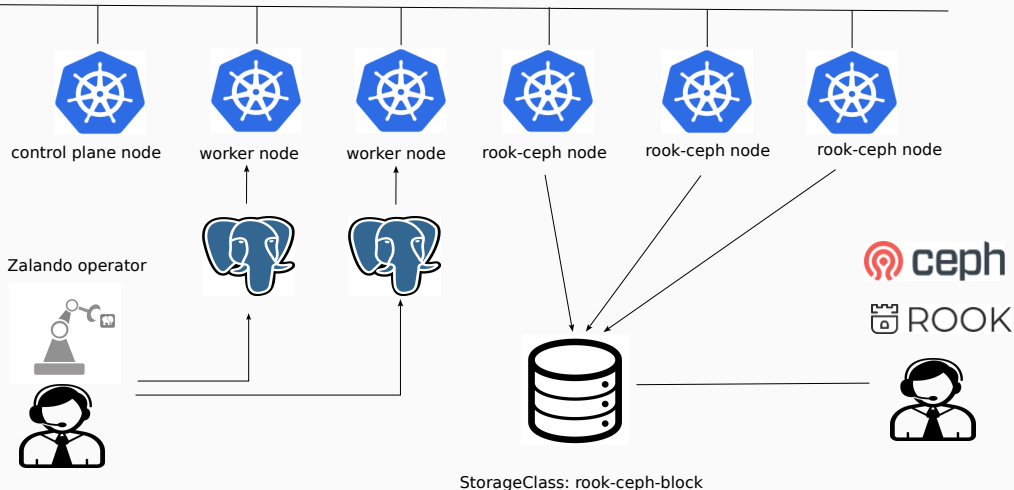
27 février 2023

Installation

Architecture



Kubernetes v1.26.2



- OS de déploiement : Debian 11 - Bullseye
- Versions de Kubernetes : 1.26.x

- Kubernetes s'appuie sur un élément essentiel qui est le *container runtime*.
- La méthode de déploiement du container runtime s'appuie la méthode décrite dans le lien : <https://docs.docker.com/engine/install/debian/>

Mise à jour de l'index du paquet *apt* et installation des paquets nécessaires à l'utilisation des dépôts avec le protocole HTTPS :

```
sudo apt-get update
```

```
sudo apt-get install \  
    ca-certificates \  
    curl \  
    gnupg
```

Ajout de la clef GPG officielle de Docker

```
sudo install -m 0755 -d /etc/apt/keyrings  
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg  
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

Ajout du dépôt de Docker

```
echo \  
"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian \  
"${(. /etc/os-release & echo "$VERSION_CODENAME")}" stable" | \  
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```


Installation de Docker Engine

```
sudo apt-get update  
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Installation de kubectl, kubeadm et kubelet

```
sudo curl -fsSLo /etc/apt/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | \
sudo tee /etc/apt/sources.list.d/kubernetes.list
```

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

Activation des modules kernel *overlay* et *br_netfilter*

```
linagora@debian-cp:/etc/modules-load.d$ cat k8s.conf
overlay
br_netfilter
linagora@debian-cp:/etc/modules-load.d$ pwd
/etc/modules-load.d
```

Activation des fonctions *bridge/iptables* et *forward* du kernel

```
linagora@debian-cp:/etc/sysctl.d$ cat k8s.conf
inet.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
linagora@debian-cp:/etc/sysctl.d$ pwd
/etc/sysctl.d
```

Génération du paramétrage par défaut de containerd :

```
root@debian-cp:~# containerd config default dump > /etc/containerd/config.toml.dmp
```

Modifier la valeur à **true** pour le paramètre **SystemdCgroup** :

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
  BinaryName = ""
  CriuImagePath = ""
  CriuPath = ""
  CriuWorkPath = ""
  IoGid = 0
  IoUid = 0
  NoNewKeyring = false
  NoPivotRoot = false
  Root = ""
  ShimCgroup = ""
  SystemdCgroup = true
```

Remplacer le paramétrage actuel par le paramétrage modifié :

```
root@debian-cp:~# cp /etc/containerd/config.toml /etc/containerd/config.toml.bak
root@debian-cp:~# cat /etc/containerd/config.toml.dmp > /etc/containerd/config.toml
root@debian-cp:~# systemctl restart containerd
```

Initialisation du cluster Kubernetes

En tant que root, lancer la commande suivante :

```
# kubeadm init --control-plane-endpoint 10.10.10.30 \  
--skip-phases=addon/coredns,addon/kube-proxy \  
--v=5 \  
--pod-network-cidr="10.244.0.0/16"
```

Si les phases *addon/coredns* et *addon/kube-proxy* ne sont pas évitées au 1^{er} lancement de kubeadm, l'erreur suivante est générée :

```
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key error execution phase  
addon/coredns : unable to fetch CoreDNS current installed version and ConfigMap. : rpc error : code = Unknown desc = malformed  
header : missing HTTP content-type To see the stack trace of this error execute with -v=5 or higher
```

Initialisation du cluster Kubernetes

Le résultat de la commande d'init est le suivant :

```
10315 01 :06 :38.342010 34405 kubeletfinalize.go :134] [kubelet-finalize] Restarting the kubelet to enable client certificate rotation
```

Your Kubernetes control-plane has initialized successfully !

To start using your cluster, you need to run the following as a regular user :

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Alternatively, if you are the root user, you can run :

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at :

<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

You can now join any number of control-plane nodes by copying certificate authorities and service account keys on each node and then running the following as root :

```
kubeadm join 10.10.10.30:6443 --token 6pia7c.n6u8pbm7yjl6nnr8 \
--discovery-token-ca-cert-hash sha256:f6d45602ea75c7659dc91f661d19e97e6817e2847e4e5d0047880b871317a145 \
--control-plane
```

Then you can join any number of worker nodes by running the following on each as root :

```
kubeadm join 10.10.10.30:6443 --token 6pia7c.n6u8pbm7yjl6nnr8 \
--discovery-token-ca-cert-hash sha256:f6d45602ea75c7659dc91f661d19e97e6817e2847e4e5d0047880b871317a145
```


L'utilisation de *kubectl* nécessite l'action suivante :

```
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Comme indiqué précédemment, les addons CoreDNS et Kube-Proxy n'ont pas été déployés au 1^{er} lancement de kubeadm.

CoreDNS peut maintenant être déployé sans erreur :

```
linagora@debian-cp:~$ sudo kubeadm init phase addon coredns  
[addons] Applied essential addon: CoreDNS
```

Déploiement de l'addon Kube-Proxy

```
linagora@debian-cp:~$ sudo kubeadm init phase addon kube-proxy  
[addons] Applied essential addon: kube-proxy
```

Il existe différentes add-ons Kubernetes implémentant l'interface CNI.

Ces add-ons sont listés dans l'URL suivante :

<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

Pour le POC, l'add-on sélectionné est Flannel car il semble être le plus simple et le plus basique des add-ons CNI.

Déploiement de l'addon *Flannel*

L'addon Flannel s'installe de plusieurs manières

(<https://github.com/flannel-io/flannel#deploying-flannel-manually>).

La méthode utilisée pour le POC est kubectl :

```
kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
```

Un outil pratique de visualisation d'un cluster kubernetes est : **k9s**

(<https://k9scli.io/>)

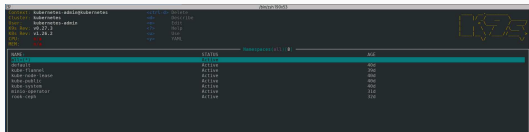
Le lien suivant permet de télécharger l'archive incluant le binaire :

https://github.com/derailed/k9s/releases/download/v0.27.3/k9s_Linux_

Liste des namespaces

```
linagora@debian-cp:~$ kubectl get namespaces
```

NAME	STATUS	AGE
default	Active	40d
kube-flannel	Active	39d
kube-node-lease	Active	40d
kube-public	Active	40d
kube-system	Active	40d
minio-operator	Active	32d
rook-ceph	Active	32d



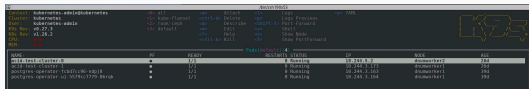
The screenshot shows a terminal window with the command `kubectl get namespaces` executed. The output is a table with three columns: NAME, STATUS, and AGE. The namespaces listed are default, kube-flannel, kube-node-lease, kube-public, kube-system, minio-operator, and rook-ceph, all with a status of 'Active' and ages ranging from 32d to 40d. A red horizontal bar highlights the header row. In the background, there is a faint logo that reads 'K8S'.

NAME	STATUS	AGE
default	Active	40d
kube-flannel	Active	39d
kube-node-lease	Active	40d
kube-public	Active	40d
kube-system	Active	40d
minio-operator	Active	32d
rook-ceph	Active	32d

Pods du namespace default

```
linagora@debian-cp:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
acid-test-cluster-0	1/1	Running	0	27d
acid-test-cluster-1	1/1	Running	0	27d
postgres-operator-fcbd7cc96-ndpj8	1/1	Running	0	40d
postgres-operator-ui-5579cc7779-86rqk	1/1	Running	0	40d



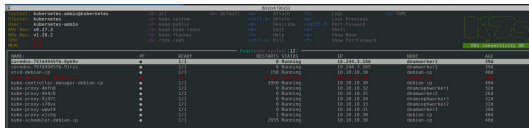
The screenshot shows a terminal window with the command `kubectl get pods` and its output. The output is a table with columns: NAME, READY, STATUS, RESTARTS, and AGE. The pods listed are acid-test-cluster-0, acid-test-cluster-1, postgres-operator-fcbd7cc96-ndpj8, and postgres-operator-ui-5579cc7779-86rqk. All are in a 'Running' state. To the right of the terminal output, there is a small network diagram showing three nodes connected in a triangle.

NAME	READY	STATUS	RESTARTS	IP	NODE	AGE
acid-test-cluster-0	1/1	Running	0	10.244.0.2	nodeworker2	28d
acid-test-cluster-1	1/1	Running	0	10.244.0.12	nodeworker1	28d
postgres-operator-fcbd7cc96-ndpj8	1/1	Running	0	10.244.0.169	nodeworker1	18d
postgres-operator-ui-5579cc7779-86rqk	1/1	Running	0	10.244.0.164	nodeworker1	18d

Pods du namespace kube-system

```
linagora@debian-cp:~$ kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
coredns-787d4945fb-8ph9v	1/1	Running	0	40d
coredns-787d4945fb-9jrzs	1/1	Running	0	40d
etcd-debian-cp	1/1	Running	158	41d
kube-apiserver-debian-cp	0/1	Running	4968 (13m ago)	41d
kube-controller-manager-debian-cp	1/1	Running	4161 (8m26s ago)	41d
kube-proxy-4mfn8	1/1	Running	0	33d
kube-proxy-9h4c6	1/1	Running	0	27d
kube-proxy-9j47t	1/1	Running	0	33d
kube-proxy-s78vx	1/1	Running	0	33d
kube-proxy-wpwt4	1/1	Running	0	40d
kube-proxy-xjs5q	1/1	Running	1 (33d ago)	41d
kube-scheduler-debian-cp	1/1	Running	2848 (6m20s ago)	41d

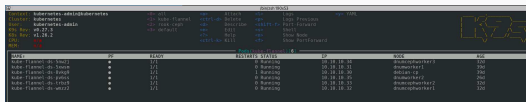


NAME	READY	STATUS	RESTARTS	IP	NODE	AGE
coredns-787d4945fb-8ph9v	1/1	Running	0	10.244.1.100	debian-cp	40d
coredns-787d4945fb-9jrzs	1/1	Running	0	10.244.1.101	debian-cp	40d
etcd-debian-cp	1/1	Running	158	10.244.1.102	debian-cp	41d
kube-apiserver-debian-cp	0/1	Running	4968	10.244.1.103	debian-cp	41d
kube-controller-manager-debian-cp	1/1	Running	4161	10.244.1.104	debian-cp	41d
kube-proxy-4mfn8	1/1	Running	0	10.244.1.105	debian-cp	33d
kube-proxy-9h4c6	1/1	Running	0	10.244.1.106	debian-cp	27d
kube-proxy-9j47t	1/1	Running	0	10.244.1.107	debian-cp	33d
kube-proxy-s78vx	1/1	Running	0	10.244.1.108	debian-cp	33d
kube-proxy-wpwt4	1/1	Running	0	10.244.1.109	debian-cp	40d
kube-proxy-xjs5q	1/1	Running	1	10.244.1.110	debian-cp	41d
kube-scheduler-debian-cp	1/1	Running	2848	10.244.1.111	debian-cp	41d

Pods du namespace kube-flannel

```
linagora@debian-cp:~$ kubectl get pods -n kube-flannel
```

NAME	READY	STATUS	RESTARTS	AGE
kube-flannel-ds-5nw2j	1/1	Running	0	33d
kube-flannel-ds-5xwsm	1/1	Running	0	40d
kube-flannel-ds-8vkg9	1/1	Running	1 (33d ago)	40d
kube-flannel-ds-pv6ss	1/1	Running	0	27d
kube-flannel-ds-trbz9	1/1	Running	0	33d
kube-flannel-ds-wmzz2	1/1	Running	0	33d



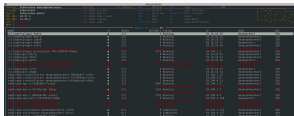
The screenshot shows a terminal window with the command `kubectl get pods -n kube-flannel` and its output. Below the command output, there is a detailed view of the pods, showing their names, IP addresses, ready status, restarts, states, and the nodes they are running on.

NAME	IP	READY	RESTARTS	STATES	IP	NODE	AGE
kube-flannel-ds-5nw2j	10.18.18.10	1/1	0	Running	10.18.18.10	dnsmaster2	33d
kube-flannel-ds-5xwsm	10.18.18.11	1/1	0	Running	10.18.18.11	dnsmaster1	40d
kube-flannel-ds-8vkg9	10.18.18.12	1/1	1	Running	10.18.18.12	dnsmaster1	40d
kube-flannel-ds-pv6ss	10.18.18.13	1/1	0	Running	10.18.18.13	dnsmaster2	27d
kube-flannel-ds-trbz9	10.18.18.14	1/1	0	Running	10.18.18.14	dnsmaster2	33d
kube-flannel-ds-wmzz2	10.18.18.15	1/1	0	Running	10.18.18.15	dnsmaster1	33d

Pods du namespace rook-ceph

```
linagora@debian-cp:~$ kubectl get pods -n rook-ceph
```

NAME	READY	STATUS	RESTARTS	AGE
csi-cephfsplugin-9nbts	2/2	Running	1 (27d ago)	27d
csi-cephfsplugin-bpxlw	2/2	Running	0	33d
csi-cephfsplugin-jd5x8	2/2	Running	0	33d
csi-cephfsplugin-mdkfk	2/2	Running	0	33d
csi-cephfsplugin-nrmfz	2/2	Running	0	33d
csi-cephfsplugin-provisioner-84cc595b78-9mml4	5/5	Running	6008 (2m44s ago)	33d
csi-cephfsplugin-provisioner-84cc595b78-9twng	5/5	Running	2171	33d
csi-rbdplugin-92z1q	2/2	Running	0	33d
csi-rbdplugin-c95w7	2/2	Running	0	33d
csi-rbdplugin-pk57s	2/2	Running	1 (27d ago)	27d
csi-rbdplugin-provisioner-6f6b6b8cd6-4c8jd	1/5	CreateContainerError	1344	33d
csi-rbdplugin-provisioner-6f6b6b8cd6-gw6bm	1/5	CreateContainerError	4465	33d
csi-rbdplugin-srtfz	2/2	Running	0	33d
csi-rbdplugin-v6gqm	2/2	Running	0	33d
rook-ceph-crashcollector-dnumcephworker1-7845bb8ff-vs9fx	1/1	Running	0	32d
rook-ceph-crashcollector-dnumcephworker2-75cdf95dcd-n5xsx	1/1	Running	0	33d
rook-ceph-crashcollector-dnumcephworker3-6fddb6cd9-x45w5	1/1	Running	1 (8d ago)	32d
rook-ceph-mgr-a-c5db58dff-hvsp9	3/3	Running	1487 (6d6h ago)	33d
rook-ceph-mgr-b-7bbfd88c8b-wh4ww	2/3	CreateContainerError	944	22d
rook-ceph-mon-a-75cf9ccddc-b2jgc	2/2	Running	1163	33d
rook-ceph-mon-b-78d6586d5-qss4z	1/2	CreateContainerError	701 (19d ago)	19d
rook-ceph-mon-c-64dcb4c86c-ws8sg	2/2	Running	1755	33d
rook-ceph-operator-cf4f7dfd4-6tm6p	1/1	Running	0	32d
rook-ceph-osd-0-57d9b8db4d-d6dhr	1/2	CreateContainerError	484	32d
rook-ceph-osd-1-74698f77fd-6n2mh	1/2	Running	529	32d
rook-ceph-osd-2-5cc486467c-lhm47	1/2	Running	1116 (49m ago)	32d
rook-ceph-osd-prepare-dnumcephworker1-rnk78	0/1	Completed	0	21d
rook-ceph-osd-prepare-dnumcephworker3-42rxv	0/1	Completed	0	21d
rook-ceph-tools-7c4b8bb9b5-pxk67	1/1	Running	0	33d



Sur chacun des 2 workers, il est nécessaire de déployer :

- le runtime containerd de Docker
- les commandes kubectrl, kubeadm et kubelet
- l'activation des modules kernel overlay et br_netfilter
- l'activation des fonctions bridge/iptables et forward du kernel
- le paramétrage de containerd

Ajout du nœud worker dans le cluster k8s - join

L'opération qui permet au nœud worker de rejoindre le cluster s'appelle le join.

La syntaxe de cette commande est obtenue en lançant la commande suivante sur le control plane avec l'utilisateur root :

```
# kubeadm token create --print-join-command
kubeadm join 10.10.10.30:6443 \
--token ilfbgc.8xco4svm5pnxkfbj \
--discovery-token-ca-cert-hash sha256:73bf45619ae0051d4ff810328d1dadcd18e6a5966c95d3c4ec76275b89a934595
```

Lancement du join sur chacun des workers

Sur chacun des workers, le lancement de la commande join produit le résultat suivant :

```
# kubeadm join 10.10.10.30:6443 \
--token 6pia7c.n6u8pbm7yjl6nnr8 \
--discovery-token-ca-cert-hash sha256:f6d45602ea75c7659dc91f661d19e97e6817e2847e4e5d0047880b871317a145
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
W0315 16:31:41.445771      6266 configset.go:78] Warning: No kubeproxy.config.k8s.io/v1alpha1 config is loaded. Continuing...
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...
```

This node has joined the cluster:

- * Certificate signing request was sent to apiserver and a response was received.
- * The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

La commande suivante permet de vérifier le résultat du join :

```
$ kubectl get nodes
NAME           STATUS    ROLES    AGE   VERSION
debian-cp      NotReady control-plane 15h   v1.26.2
dnumworker1    NotReady <none>    53s   v1.26.2
```

Terminologie du stockage dans k8s

- Le stockage permanent des données s'appuie les volumes persistants (PV)
(<https://kubernetes.io/docs/concepts/storage/persistent-volumes/>)
- Un PV est un espace de stockage mis à disposition par k8s.
- Il peut être alloué manuellement ou dynamiquement par l'intermédiaire des storage class
(<https://kubernetes.io/docs/concepts/storage/storage-classes/>)
- Les PV sont l'équivalent d'un node dans un cluster.
- Les persistentVolumeClaim (PVC) sont l'équivalent d'un pod.

- Le storage class sur lequel s'appuie l'opérateur PostgreSQL est Ceph
- L'opérateur k8s **Rook Ceph** facilite le déploiement de Ceph
- Le déploiement s'appuie sur le lien
`https://rook.io/docs/rook/v1.9/quickstart.html`
- La version de l'opérateur utilisée est la v1.9
- Elle supporte les versions k8s v1.17+

Prérequis au déploiement de l'opérateur - Rook Ceph

- Le déploiement de l'opérateur scanne l'ensemble des noeuds de stockage pour vérifier la présence de :
 - des devices bruts (sans partitions ou filesystems formatés)
 - des partitions brutes (sans filesystems formatés)
 - les volumes physiques initialisés par LVM

L'exemple ci-dessous indique comment vérifier la disponibilité d'espace pour l'opérateur Rook Ceph :

```
lsblk -f
```

NAME	FSTYPE	LABEL	UUID	MOUNTPOINT
vda				
-vda1	LVM2_member		>eS050t-GkUV-YKTH-WsGq-hNJY-eKNf-3i07IB	
-ubuntu--vg-root	ext4		c2366f76-6e21-4f10-a8f3-6776212e2fe4	/
-ubuntu--vg-swap_1	swap		9492a3dc-ad75-47cd-9596-678e8cf17ff9	[SWAP]
vdb				

- Dans l'exemple précédent, si la colonne *FSTYPE* est renseignée, cela indique la présence d'un filesystem
- La partition vdb n'est pas formatée avec un filesystem : elle est donc utilisable par l'opérateur Rook Ceph
- Le paquet **lvm2** est une dépendance importante de Rook Ceph

Sélection des nœuds sur lesquels Ceph sera déployé

L'opérateur Rook Ceph offre la possibilité de sélectionner les nœuds sur lesquels le stockage Ceph est déployé.

Pour cela, il s'appuie sur la notion de label.

Dans le cadre du POC, les 3 nœuds suivants sont sélectionnés pour porter le stockage :

- dnumcephworker1
- dnumcephworker2
- dnumcephworker3

Affectation des labels sur les nœuds de stockage

Depuis le control plane, lancer les commandes suivantes :

```
$ kubectl label nodes dnumcephworker1 role=storage-node
```

```
node/dnumcephworker1 labeled
```

```
$ kubectl label nodes dnumcephworker2 role=storage-node
```

```
node/dnumcephworker2 labeled
```

```
$ kubectl label nodes dnumcephworker3 role=storage-node
```

```
node/dnumcephworker3 labeled
```

```
$ kubectl get nodes --show-labels
```

NAME	STATUS	ROLES	LABELS
dnumcephworker1	Ready	<none>	kubernetes.io/hostname=dnumcephworker1, kubernetes.io/os=linux, role=storage-node
dnumcephworker2	Ready	<none>	kubernetes.io/hostname=dnumcephworker2, kubernetes.io/os=linux, role=storage-node
dnumcephworker3	Ready	<none>	kubernetes.io/hostname=dnumcephworker3, kubernetes.io/os=linux, role=storage-node

Paramétrage pour la répartition du stockage Ceph sur les nœuds labélisés

```
~/rook$ git diff
diff --git a/deploy/examples/cluster.yaml b/deploy/examples/cluster.yaml
index 9bd50ec97..fef3f777f 100644
--- a/deploy/examples/cluster.yaml
+++ b/deploy/examples/cluster.yaml
@@ -154,22 +154,22 @@ spec:
   # To control where various services will be scheduled by kubernetes, use the placement configuration sections below.
   # The example under 'all' would have all services scheduled on kubernetes nodes labeled with 'role=storage-node' and
   # tolerate taints with a key of 'storage-node'.
-  # placement:
-  #   all:
-  #     nodeAffinity:
-  #       requiredDuringSchedulingIgnoredDuringExecution:
-  #         nodeSelectorTerms:
-  #           - matchExpressions:
-  #             - key: role
-  #               operator: In
-  #               values:
-  #                 - storage-node
-  #     podAffinity:
-  #     podAntiAffinity:
-  #     topologySpreadConstraints:
-  #     tolerations:
-  #       - key: storage-node
-  #         operator: Exists
+  placement:
+    all:
+      nodeAffinity:
+        requiredDuringSchedulingIgnoredDuringExecution:
+          nodeSelectorTerms:
+            - matchExpressions:
+              - key: role
+                operator: In
+                values:
+                  - storage-node
+      podAffinity:
+      podAntiAffinity:
+      topologySpreadConstraints:
+      tolerations:
+        - key: storage-node
+          operator: Exists
+  # The above placement information can also be specified for mon, osd, and mgr components
+  # mon:
+  # Monitor deployments may contain an anti-affinity rule for avoiding monitor
```

Paramétrage pour la répartition du stockage Ceph sur les nœuds labelisés

La directive *nodeSelectorTerms* permet de sélectionner les noeuds portant la storageclass Ceph

```
+         nodeSelectorTerms:
...
+         - storage-node
+     podAffinity:
+     podAntiAffinity:
```

Déploiement de l'opérateur Rook Ceph

Comme indiqué dans le lien

<https://rook.io/docs/rook/v1.9/quickstart.html>, l'application des commandes ci-dessous amorce le déploiement de l'opérateur :

```
$ git clone --single-branch --branch v1.9.2 https://github.com/rook/rook.git
cd rook/deploy/examples
kubectl create -f crds.yaml -f common.yaml -f operator.yaml
kubectl create -f cluster.yaml
```

- Une fois le cluster est opérationnel, il devient possible de créer :
 - stockage bloc
 - stockage objet
 - stockage fichier

Vérification de l'opérateur Rook Ceph

```
# verify the rook-ceph-operator is in the 'Running' state before proceeding  
kubectl -n rook-ceph get pod
```


Le contrôleur d'admission (Admission Controller) - Rook Ceph

- Il est recommandé de déployer le contrôleur d'admission : il permet de vérifier que Rook est correctement paramétré grâce aux réglages des Customer Resources (CR)
- L'Admission Controller intercepte les requêtes à destination de l'API k8s avant l'objet persistant après les phases d'authentification et d'autorisation
- Pour installer l'Admission Controller, lancer les requêtes suivantes :

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.7.1/cert-manager.yaml
```

Affichage des storage class déployés

```
linagora@debian-cp:~$ kubectl get storageclass
```

NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION	AGE
local-storage	kubernetes.io/no-provisioner	Delete	WaitForFirstConsumer	false	12d
rook-ceph-block	rook-ceph.rbd.csi.ceph.com	Delete	Immediate	true	5d23h

WIP Le storage class est maintenant déployé.

TODO

Bibliographie

Webographie

Sommaire

Installation

Conclusion
