PostgreSQL

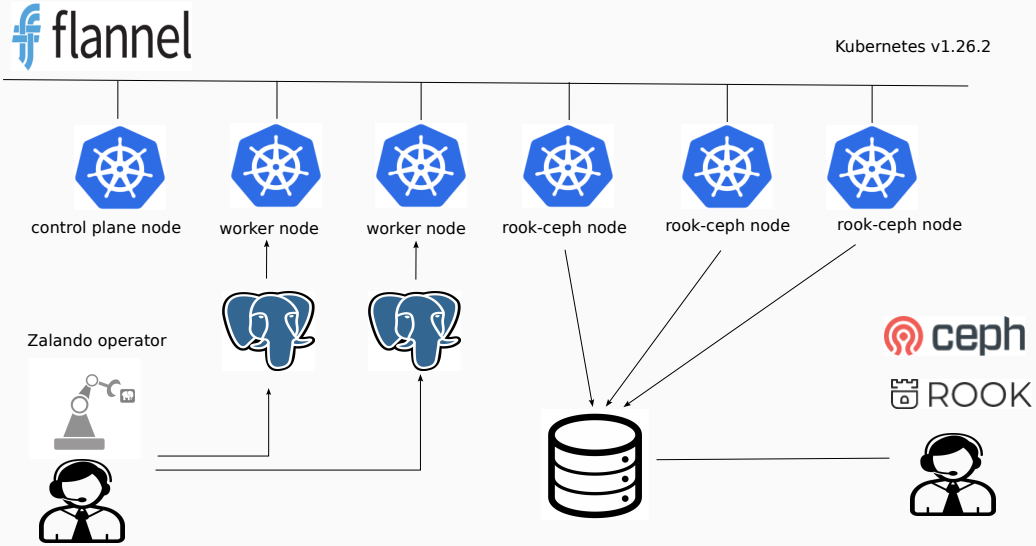
Déploiement de l'opérateur PostgreSQL

Simon ELBAZ (selbaz@linagora.com)

27 février 2023

Installation

Architecture



Versions utilisées

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

Prérequis matériels

- etcd est la base de données clé-valuers centrale utilisée par Kubernetes
- etcd utilise de manière intensive les disques à disposition
- Pour une stabilité accrue du cluster, il est préférable d'utiliser des disques de type **SSD**

Déploiement du nœud control plane

- 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/`

Désactivation permanente de la mémoire swap

Le process kubelet ne démarre pas en cas de mémoire swap activée.
Pour désactiver l'utilisation de la swap, merci d'utiliser la commande suivante :

```
swapoff -a
```

Pour persister cet état et faire en sorte que la mémoire swap ne soit pas activée au prochain reboot, supprimer ou mettre en commentaires la ligne suivante dans */etc/fstab* :

```
$ sudo cat /etc/fstab
/dev/mapper/dnumworker1--vg-root /               ext4    errors=remount-ro 0       1
# /boot was on /dev/sda1 during installation
UUID=ddd6fd9d-6ac3-4510-9156-22984bc82b67 /boot           ext2    defaults        0       2
#/dev/mapper/dnumworker1--vg-swap_1 none            swap    sw              0       0
/dev/sr0        /media/cdrom0   udf,iso9660 user,noauto     0       0
```

Installation du runtine container *containerd*

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
```

Paramétrage de containerd

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
```

Paramétrage de containerd

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 1er 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 :

I0315 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.n6u8pbm7yj16nnr8 \
      --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.n6u8pbm7yj16nnr8 \
      --discovery-token-ca-cert-hash sha256:f6d45602ea75c7659dc91f661d19e97e6817e2847e4e5d0047880b871317a145
```

Paramétrage de *kubectl*

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
```

Déploiement de l'addon CoreDNS

Comme indiqué précédemment, les addons CoreDNS et Kube-Proxy n'ont pas été déployés au 1er 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
```

Choix de la couche réseau - Container Network Interface

Il existe différentes addons Kubernetes implémentant l'interface CNI.
Ces addons sont listés dans l'URL suivante :
`https://kubernetes.io/docs/concepts/cluster-administration/addons/`
Pour le POC, l'addon sélectionné est Flannel car il semble être le plus simple et le plus basique des addons 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 :

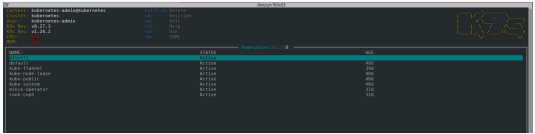
```
kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
```

Installation de k9s

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_`

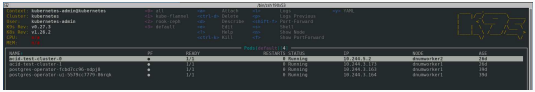
Liste des namespaces

```
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
```



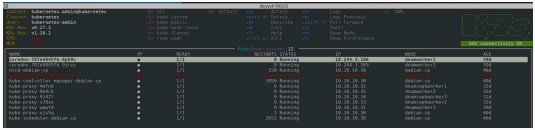
Pods du namespace default

```
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
```



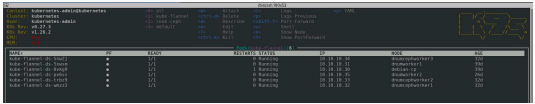
Pods du namespace kube-system

```
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
```



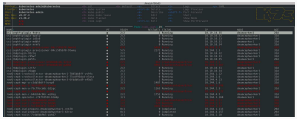
Pods du namespace kube-flannel

```
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
```



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-mddkf                                   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-84cc595b78-9twnq            5/5     Running                 2171               33d
csi-rbdplugin-92zlq                                      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-n5xsz 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-w4xvw                         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-wz8sg                         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
```



Déploiement du nœud worker

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

- le runtime containerd de Docker
- les commandes kubectl, 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 sha256:73bf45619ae0051d4ff810328d1dadc18e6a5966c95d3c4ec76275b89a934595
```

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. Continuin
[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.

Déploiement du stockage - Rook Ceph

- 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 formattés)
  - des partitions brutes (sans filesystems formattés)
  - 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       >eSO50t-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
```

Prérequis au déploiement de l'opérateur - Rook Ceph

- 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 pour marquer les nœuds :

```
$ 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
```

Affichage du label des nœuds :

```
$ 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 labelisés

```diff
~/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 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
NAME                                                READY   STATUS                  RESTARTS          AGE
csi-cephfsplugin-9nbts                              2/2     Running                 1 (63d ago)       63d
csi-cephfsplugin-bpxlw                              2/2     Running                 0                 69d
csi-cephfsplugin-jd5x8                              2/2     Running                 0                 69d
csi-cephfsplugin-mddkf                              2/2     Running                 0                 69d
csi-cephfsplugin-nrmfz                              2/2     Running                 0                 69d
csi-cephfsplugin-provisioner-84cc595b78-9mml4       5/5     Running                 6523 (28d ago)    69d
csi-cephfsplugin-provisioner-84cc595b78-9twnq       5/5     Running                 3908 (30d ago)    69d
csi-rbdplugin-92zlq                                 2/2     Running                 0                 69d
csi-rbdplugin-c95w7                                 2/2     Running                 0                 69d
csi-rbdplugin-pk57s                                 2/2     Running                 1 (63d ago)       63d
csi-rbdplugin-provisioner-6f6b6b8cd6-4c8jd          5/5     Terminating             2919 (29d ago)    69d
csi-rbdplugin-provisioner-6f6b6b8cd6-d4t56          0/5     Pending                 0                 4d10h
csi-rbdplugin-provisioner-6f6b6b8cd6-gw6bm          1/5     CreateContainerError    4465              69d
csi-rbdplugin-srtfz                                 2/2     Running                 0                 69d
csi-rbdplugin-v6gqm                                 2/2     Running                 0                 69d
rook-ceph-crashcollector-dnumcephworker1-7845bb8ff-vs9fx    1/1     Running                 0                 68d
rook-ceph-crashcollector-dnumcephworker2-75cdf95dcd-ljkqd   0/1     Pending                 0                 4d10h
rook-ceph-crashcollector-dnumcephworker2-75cdf95dcd-n5xsz   1/1     Terminating             0                 69d
rook-ceph-crashcollector-dnumcephworker3-6fddb6cd9-x45w5    1/1     Running                 2                 68d
rook-ceph-mgr-a-c5db58dff-fpp7z                     2/3     CrashLoopBackOff        146 (28d ago)     30d
rook-ceph-mgr-a-c5db58dff-hvsp9                     2/3     Terminating             3115 (30d ago)    69d
rook-ceph-mgr-b-7bbfd88c8b-jdq4p                    0/3     Pending                 0                 4d10h
rook-ceph-mgr-b-7bbfd88c8b-wh4ww                    2/3     Terminating             2283 (28d ago)    58d
rook-ceph-mon-a-75cf9ccddc-b2jgc                    2/2     Running                 1500 (31d ago)    69d
rook-ceph-mon-c-64dcb4c86c-wz8sg                    2/2     Running                 1808 (28d ago)    69d
rook-ceph-operator-cf4f7dfd4-6tm6p                  1/1     Running                 0                 68d
rook-ceph-osd-0-57d9b8db4d-d6dhr                    1/2     Terminating             731 (28d ago)     68d
rook-ceph-osd-0-57d9b8db4d-vmtjp                    0/2     Pending                 0                 4d10h
rook-ceph-osd-1-74698f77fd-6n2mh                    1/2     Running                 716 (30d ago)     68d
rook-ceph-osd-2-5cc486467c-lhm47                    1/2     Running                 1172 (28d ago)    68d
rook-ceph-osd-prepare-dnumcephworker1-rnk78         0/1     Completed               0                 57d
rook-ceph-osd-prepare-dnumcephworker3-42rxv         0/1     Completed               0                 57d
rook-ceph-tools-7c4b8bb9b5-8tf8r                    0/1     Pending                 0                 4d10h
rook-ceph-tools-7c4b8bb9b5-pxk67                    1/1     Terminating             0                 68d
```

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

Le storageclass déployé a pour nom **rook-ceph-block**.

```
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
```

Sélection des nœuds PostgreSQL

De manière similaire à l'opérateur Rook Ceph, il est possible de sélectionner les nœuds portant le pod PostgreSQL en se basant sur les labels Kubernetes.

Marquage des nœuds PostgreSQL

Les commandes ci-dessous marquent les nœuds destinés à porter les pods PostgreSQL :

```
$ kubectl label nodes dnumworker1 postgres-operator=enabled
node/dnumworker1 labeled
$ kubectl label nodes dnumworker2 postgres-operator=enabled
node/dnumworker2 labeled
$ kubectl get nodes --show-labels
NAME              STATUS   ROLES           LABELS
dnumworker1   Ready    <none>          kubernetes.io/hostname=dnumworker1,kubernetes.io/os=linux,postgres-operator=enabled
dnumworker2   Ready    <none>          kubernetes.io/hostname=dnumworker2,kubernetes.io/os=linux,postgres-operator=enabled
```

Répartitions des pods PostgreSQL sur les nœuds worker et choix du storageClass

```
$ git diff
diff --git a/manifests/complete-postgres-manifest.yaml b/manifests/complete-postgres-manifest.yaml
index 8d197a75..56b32c34 100644
--- a/manifests/complete-postgres-manifest.yaml
+++ b/manifests/complete-postgres-manifest.yaml
@@ -57,7 +57,7 @@ spec:

   volume:
     size: 1Gi
-#    storageClass: my-sc
+    storageClass: rook-ceph-block
 #    iops: 1000  # for EBS gp3
 #    throughput: 250  # in MB/s for EBS gp3
 #    selector:
@@ -203,14 +203,14 @@ spec:

 # Add node affinity support by allowing postgres pods to schedule only on nodes that
 # have label: "postgres-operator:enabled" set.
-#  nodeAffinity:
-#    requiredDuringSchedulingIgnoredDuringExecution:
-#      nodeSelectorTerms:
-#        - matchExpressions:
-#            - key: postgres-operator
-#              operator: In
-#              values:
-#                - enabled
+  nodeAffinity:
+    requiredDuringSchedulingIgnoredDuringExecution:
+      nodeSelectorTerms:
+        - matchExpressions:
+            - key: postgres-operator
+              operator: In
+              values:
+                - enabled

 # Enables change data capture streams for defined database tables
 #  streams:
```

Déploiement de l'opérateur PostgreSQL de Zalando

Le storage class est maintenant déployé.

Il devient possible d'appliquer l'opérateur PostgreSQL.

Le lien suivant décrit les commandes à appliquer :
https://github.com/zalando/postgres-operator/blob/master/docs/quickstart.md#deployment-options

Clonage du dépôt de l'opérateur

```
git clone https://github.com/zalando/postgres-operator.git
cd postgres-operator
```

Application des différents manifestes

```
kubectl create -f manifests/configmap.yaml  # configuration
kubectl create -f manifests/operator-service-account-rbac.yaml  # identity and permissions
kubectl create -f manifests/postgres-operator.yaml  # deployment
kubectl create -f manifests/api-service.yaml  # operator API to be used by UI
```

Pour information, il existe également des chart Helm pour facilier le déploiement.

Accès à l'interface web

Pour activer l'accès à l'interface web de l'opérateur PostgreSQL, veuillez la commande suivante sur le nœud control plane :

```
$ kubectl port-forward svc/postgres-operator-ui 8081:80
Forwarding from 127.0.0.1:8081 -> 8081
Forwarding from [::1]:8081 -> 8081
```

Elle redirige le flux TCP du port 80 du control plane vers le port TCP 8081 du service postgres-operator-ui

Accès à l'interface web

Pour accéder à l'interface web de l'opérateur PostgreSQL depuis le PC de l'utilisateur, il est possible de passer par une redirection SSH :

```
ssh -L 9090:10.106.57.137:80 dgfip-k8s
```

Lancer le navigateur pour accéder à l'URL `http://localhost:9090/#new`

Interface web de l'opérateur PostgreSQL

localhost:9090/#new

Search

PostgreSQL Operator UI | PostgreSQL clusters | Backups | Status | New cluster | Documentation

New PostgreSQL cluster

Cluster YAML definition

```yaml
kind: "postgresql"
apiVersion: "acid.zalan.do/v1"

metadata:
  name: ""
  namespace: "default"
  labels:
    team: acid

spec:
  teamId: "acid"
  postgresql:
    version: "15"
  numberOfInstances: 1
  volume:
    size: "10Gi"


  allowedSourceRanges:
    # IP ranges to access your cluster go here

  resources:
    requests:
      cpu: 100m
      memory: 100Mi
    limits:
      cpu: 500m
      memory: 500Mi
```

New cluster configuration

Validate | Copy definiton | Create cluster

Name	new-cluster (can be 53 characters long)
Owning team	acid
PostgreSQL version	15
DNS name:	.default
Number of instances	1
Enable load balancer	☐ Master
Enable connection pooler	☐ Master
Enable connection pooler load balancer	☐ Master
Volume size	10 Gi
storageClass	
	Specify Iops and Throughput only if you need more than the default 3000 Iops and 125Mb/s EBS provides.
Iops	

Fonctionnalités proposées par l'interface web de l'opérateur PostgreSQL

L'UI permet de :

- choisir la version PostgreSQL (jusquà la version 15 actuellement)
- le nombre d'instances
- activation du load-balancer
- activation du pool de connexions à la base
- activation du locad-balancer pour le pool de connexions à la base
- taille du volume persistent alloué à la base de données
- choix du storageClass
- performances IO
- choix des ressources (demandées et limites) CPU et RAM allouées

Utilisation de la commande en ligne pour la création d'un cluster PostgreSQL

- Les fonctionnalités proposées par l'UI sont également disponibles par la commande en ligne.
- Le manifeste *manifests/complete-postgresql-manifest.yaml* permet de préciser l'ensemble des paramètres proposés par l'UI.
- Pour appliquer ce manifeste *manifests/complete-postgresql-manifest.yaml*, la commande suivante est lancée sur le nœud :

```
kubectl create -f manifests/complete-postgresql-manifest.yaml
```

Vérification de l"état du cluster PostgreSQL

```
$ kubectl get pods -l application=spilo -L spilo-role
NAME                  READY   STATUS    RESTARTS   AGE   SPILO-ROLE
acid-test-cluster-0   1/1     Running   0          12m
acid-test-cluster-1   1/1     Running   0          10m
$ kubectl get postgresql
NAME                TEAM   VERSION   PODS   VOLUME   CPU-REQUEST   MEMORY-REQUEST   AGE   STATUS
acid-test-cluster   acid   15        2      1Gi      10m           100Mi            68d   Running
$ kubectl get pods -l application=spilo -L spilo-role
NAME                  READY   STATUS    RESTARTS   AGE   SPILO-ROLE
acid-test-cluster-0   1/1     Running   0          13m
acid-test-cluster-1   1/1     Running   0          10m
$ kubectl get svc -l application=spilo -L spilo-role
NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE   SPILO-ROLE
acid-test-cluster          ClusterIP   10.103.247.68   <none>        5432/TCP   68d   master
acid-test-cluster-config   ClusterIP   None            <none>        <none>     68d
acid-test-cluster-repl     ClusterIP   10.100.95.205   <none>        5432/TCP   68d   replica
```

Bibliographie

Webographie

Sommaire

Sommaire

Conclusion