PostgreSQL

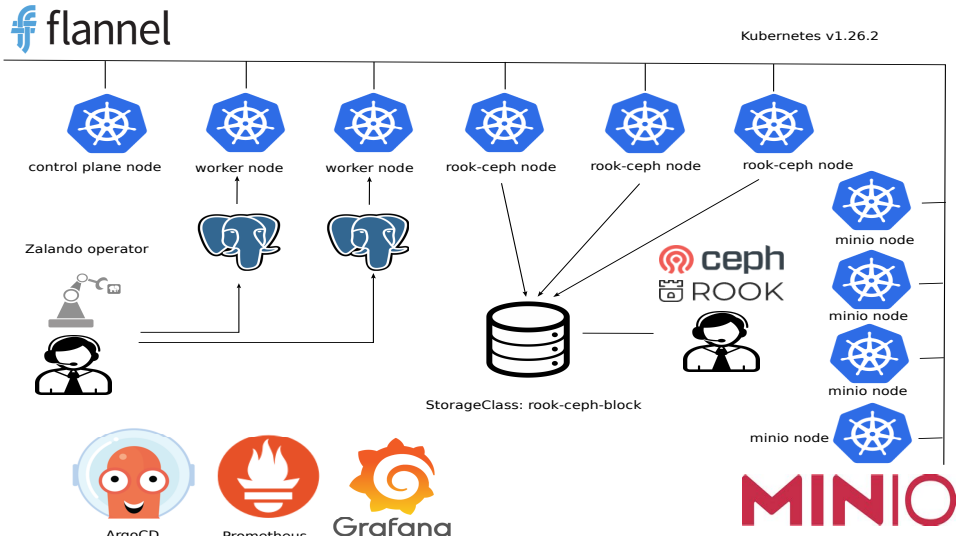
Déploiement de l'opérateur PostgreSQL de Zalando

Simon ELBAZ (selbaz@linagora.com / ossa@linagora.com)

27 février 2023

Architecture

Architecture



Installation

Versions utilisées

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

Dimensionnement des serveurs

Dimensionnement du control plane :

- 8 CPU
- 8 Go RAM

Dimensionnement des workers :

- 2 CPU
- 2 Go RAM

Prérequis matériels

- etcd est la base de données clé-valeurs 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 apt-get install -y ca-certificates curl
sudo apt-get install -y apt-transport-https
sudo apt-get update
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg \
| sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.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.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
```

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

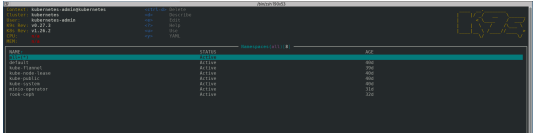
Un outil pratique de visualisation d'un cluster kubernetes est : **k9s** (`https://k9scli.io/`)
Le lien suivant[1] permet de télécharger l'archive incluant le binaire.

1.

```
https://github.com/derailed/k9s/releases/download/v0.27.3/k9s_Linux_amd64.tar.gz
```

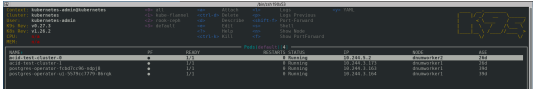
Liste des namespaces

```
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
```



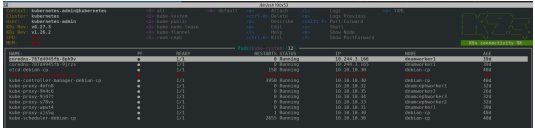
Pods du namespace default

```
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
```



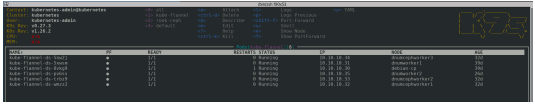
Pods du namespace kube-system

```
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
```



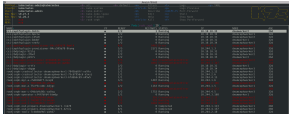
Pods du namespace kube-flannel

```
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-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-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.
Continuing without it: configmaps "kube-proxy" is forbidden: User "system:bootstrap:6pia7c"
cannot get resource "configmaps" in API group "" in the namespace "kube-system"
[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.
```

Lancement du join sur chacun des workers

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
```

Stockage

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

```
~/rook$ git diff
diff --git a/deploy/examples/cluster.yaml b/deploy/examples/cluster.yaml
index 9bd59ec97..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-jdg4p                    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
```

Déploiement PostgreSQL

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

 # 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 [2] décrit les commandes à appliquer.

2.

```
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 entrer 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

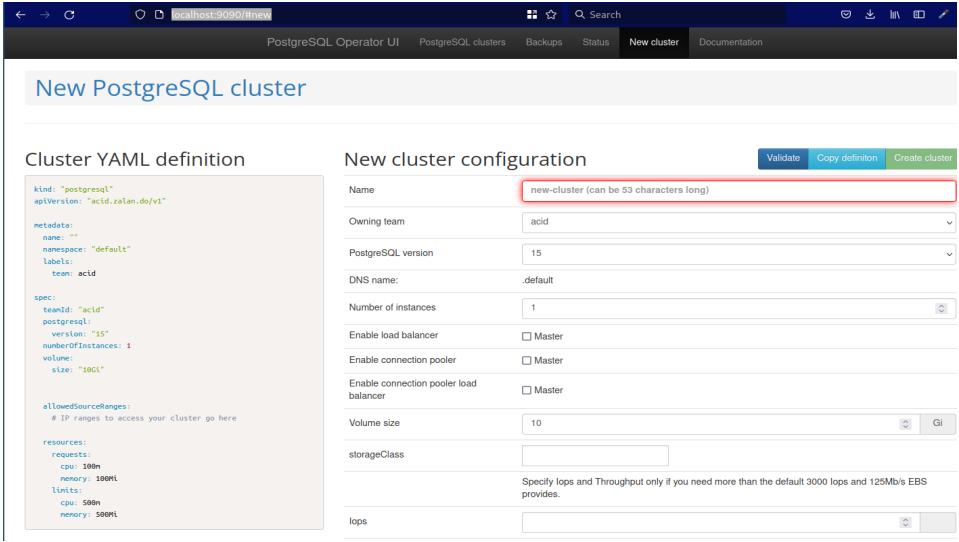
Accès à l'interface web

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



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 load-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          6m49s   master
acid-test-cluster-1   1/1     Running   0          6m12s   replica
$ 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
```

Stockage S3 - Minio

Déploiement de krew

L'opérateur Minio s'appuie sur le gestionnaire de paquets **krew**.

```
(
  set -x; cd "$(mktemp -d)" &
  OS="$(uname | tr '[:upper:]' '[:lower:]')" &
  ARCH="$(uname -m | sed -e 's/x86_64/amd64/' -e 's/\(arm\)\(64\)\?.*/\1\2/' -e 's/aarch64$/arm64/')" &
  KREW="krew-${OS}_${ARCH}" &
  curl -fsSLO "https://github.com/kubernetes-sigs/krew/releases/latest/download/${KREW}.tar.gz" &
  tar zxvf "${KREW}.tar.gz" &
  ./"${KREW}" install krew
)
```

Déploiement de krew

Ajout du répertoire des binaires du paquet krew dans **.bashrc** ou **.zshrc** :

```
export PATH="${KREW_ROOT:-$HOME/.krew}/bin:$PATH"
```

Redémarrer le shell.

Pour vérifier le déploiement correct de krew, lancer la commande suivante :

```
kubectl krew
```

Déploiement de l'opérateur Minio

Le déploiement de Minio s'appuie sur l'opérateur Minio.
Son déploiement est décrit dans le lien suivant
`https://operator.min.io/#architecture`.

```
kubectl krew update
kubectl krew install minio
```

Vérification de l'état de l'opérateur Minio

```
$ kubectl get pods -n minio-operator
NAME                              READY   STATUS    RESTARTS           AGE
console-56f9795d5c-59fsx          1/1     Running   1 (33d ago)        82d
minio-operator-7cd6784f59-5c52w   1/1     Running   5 (23h ago)        17d
minio-operator-7cd6784f59-m7h8x   1/1     Running   2320 (3d16h ago)   82d
```

Accès à la console Minio

La commande suivante ouvre un accès de type proxy à la console Minio :

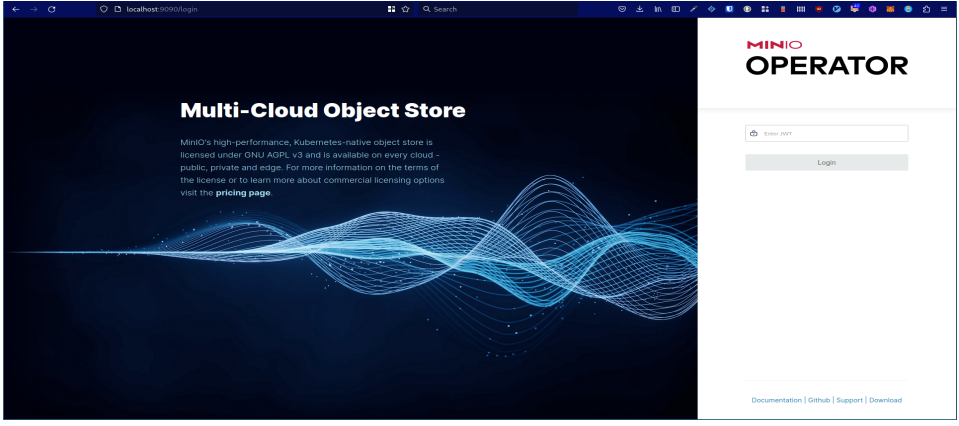
```
$ kubectl minio proxy -n minio-operator
Starting port forward of the Console UI.

To connect open a browser and go to http://localhost:9090

Current JWT to login: *****

Forwarding from 0.0.0.0:9090 -> 9090
Handling connection for 9090
```

Depuis le terminal de l'utilisateur, lancer la commande suivante :

```
$ ssh -L 9090:localhost:9090 dgfip-k8s
```

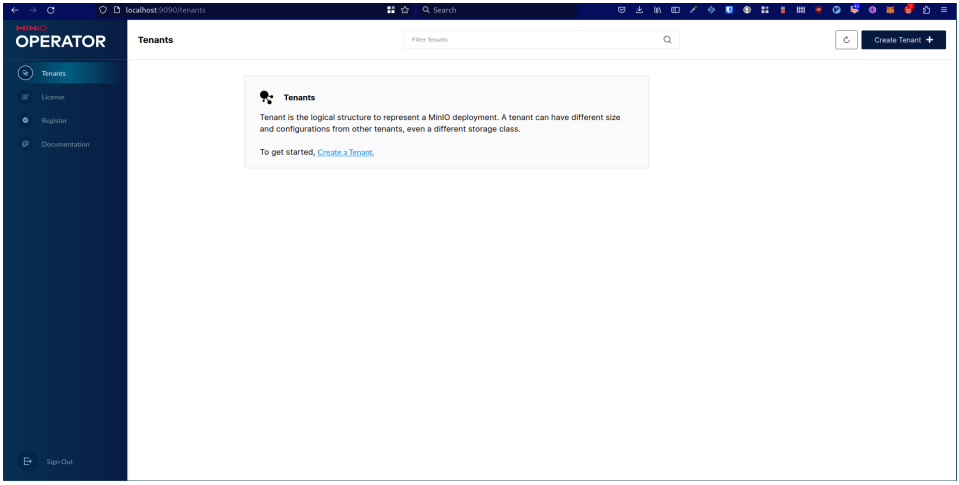
Accès à la console Minio

Dans le champ *Enter JWT*, renseigner la valeur du token JWT renvoyé par la commande précédente :

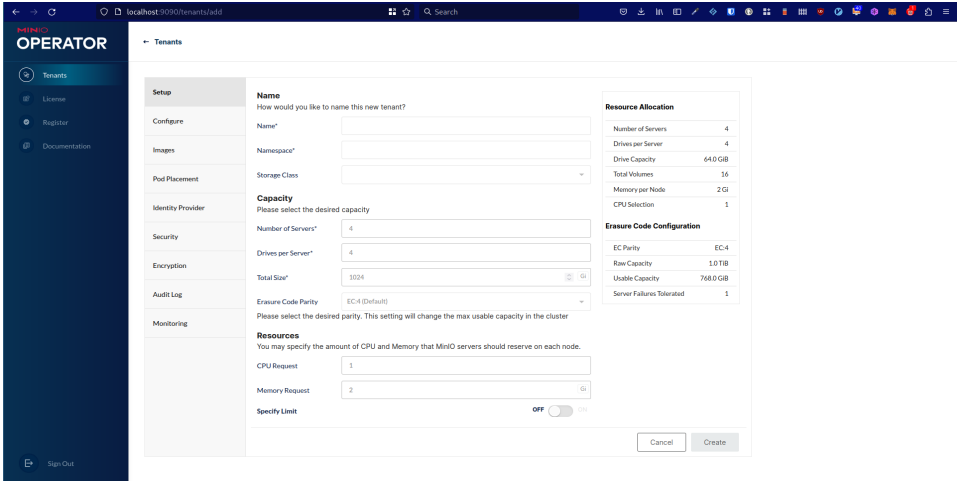


Dashboard Minio

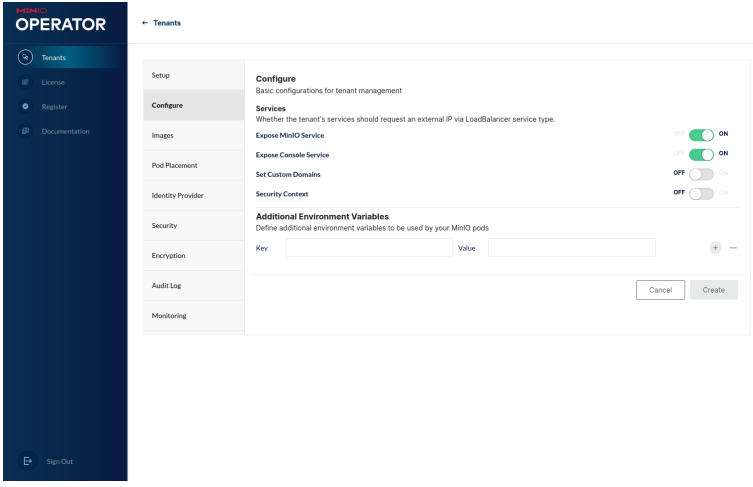
Le tableau de bord de Minio ressemble à ceci :



Création d'un tenant - Setup - Minio



Création d'un tenant - Configure - Minio



Marquage des nœuds Minio

Depuis le control plane, lancer les commandes suivantes pour marquer les nœuds :

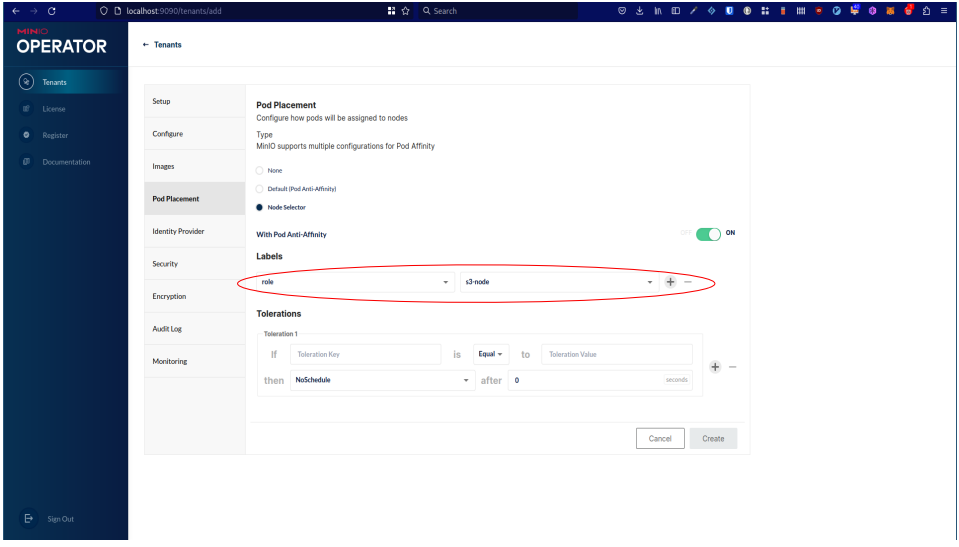
```
$ kubectl label nodes dnumminioworker1 role=s3-node
node/dnumminioworker1 labeled
$ kubectl label nodes dnumminioworker2 role=s3-node
node/dnumminioworker2 labeled
$ kubectl label nodes dnumminioworker3 role=s3-node
node/dnumminioworker3 labeled
$ kubectl label nodes dnumminioworker4 role=s3-node
node/dnumminioworker4 labeled
```

Marquage des nœuds Minio

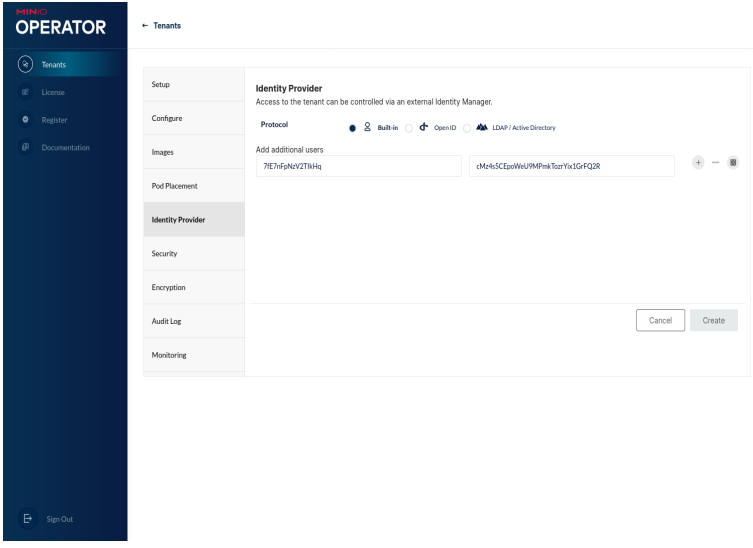
Affichage du label des nœuds :

```
$ kubectl get nodes --show-labels
NAME                STATUS   ROLES           AGE    VERSION   LABELS
…
dnumminioworker1    Ready    <none>          6h2m   v1.27.2   …,role=s3-node
dnumminioworker2    Ready    <none>          142m   v1.27.2   …,role=s3-node
dnumminioworker3    Ready    <none>          109m   v1.27.2   …,role=s3-node
dnumminioworker4    Ready    <none>          83m    v1.27.2   …,role=s3-node
…
```

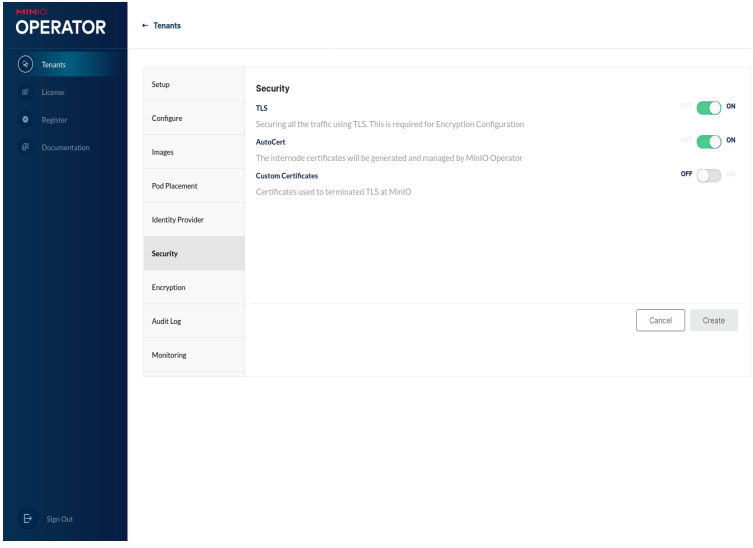
Création d'un tenant - Pod placement - Minio



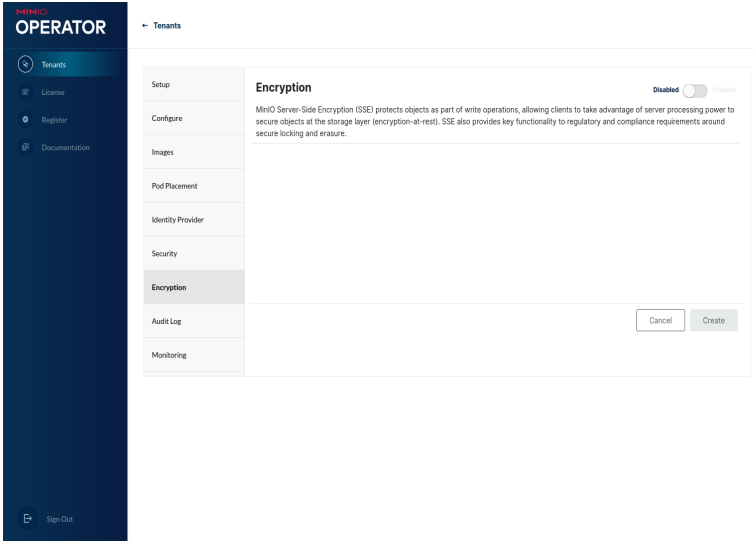
Création d'un tenant - Choix d'un fournisseur d'identité - Minio



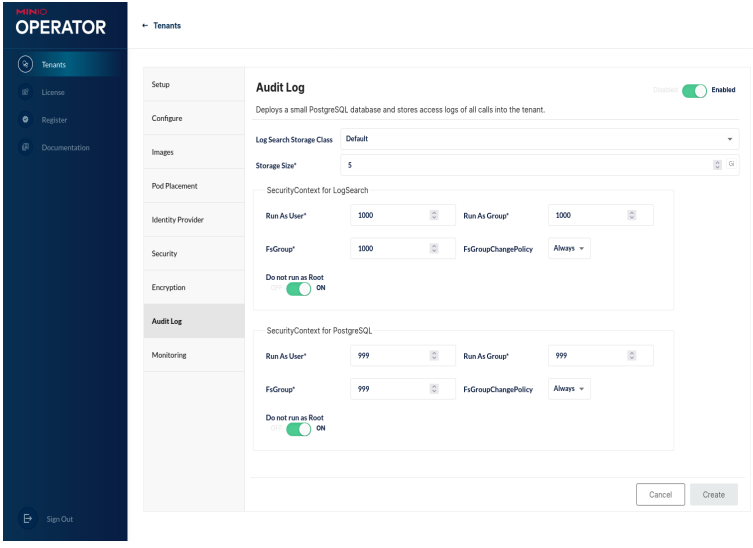
Création d'un tenant - Sécurité - Minio



Création d'un tenant - Chiffrement - Minio



Création d'un tenant - Log d'audit - Minio



Création d'un tenant - Supervision - Minio

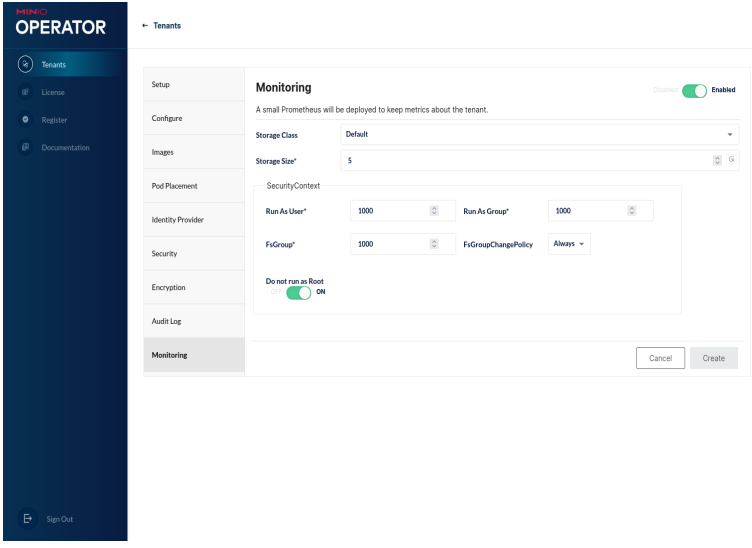


Image Docker utilisée par l'opérateur PostgreSQL

L'image déployée par l'opérateur PostgreSQL de Zalando s'appuie sur Spilo
`https://github.com/zalando/spilo`

Cette information se trouve dans le script
manifests/complete-postgres-manifest.yaml :

```
dockerImage: ghcr.io/zalando/spilo-15:3.0-p1
```

Sécurité - Support d'OpenShift

Utilisation de container en mode rootless dans OpenShift

- L'URL suivante décrit la problèmatique d'utilisation de container dans un environnement rootless [3]
- Cette page fournit également un lien intéressant sur la sécurisation d'un cluster Kubernetes en se basant sur les **Pod Security Policies**.
- Le lien est le suivant : `https://docs.bitnami.com/tutorials/secure-kubernetes-cluster-psp/`.

3.

```
https://docs.bitnami.com/tutorials/running-non-root-containers-on-openshift
```

Pod Security Policies

Comme indiqué dans `https://kubernetes.io/docs/concepts/security/pod-security-policy/`, les PSP sont dépréciés.
Ils sont maintenant remplacés par **Pod Security Admission** `https://kubernetes.io/docs/concepts/security/pod-security-admission/`.
La norme *Pod Security Admission* définit la notion de *Security Context* décrite dans le lien suivant : `https://kubernetes.io/docs/tasks/configure-pod-container/security-context/` C'est cette notion qui va permet d'approcher au plus près les condictions de run d'un cluster Openshift

Opérateur - Security Context

Par défaut, l'opérateur PostgreSQL de Zalando applique les mesures de sécurité suivantes.
Extrait de manifests/postgres-operator.yaml :

```
securityContext:
    runAsUser: 1000
    runAsNonRoot: true
    readOnlyRootFilesystem: true
    allowPrivilegeEscalation: false
```

- Il ne tourne pas avec un identifiant privilégié ni le compte root
- Il s'appuie sur des filesystems en lecture seule

Pods PostgreSQL - Security Context

Il est possible d'affecter :

- un utilisateur non privilégié
- un groupe non privilégié au pod.
- un groupe de filesystem défini

Extrait de manifests/complete-postgres-manifest.yaml :

```
spiloRunAsUser: 101
spiloRunAsGroup: 103
spiloFSGroup: 103
```

Déploiement continu

Installation d'ArgoCD

Le lien suivant décrit l'installation d'ArgoCD :

`https://argo-cd.readthedocs.io/en/stable/#quick-start`

```
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

Installation de la CLI ArgoCD

Le lien suivant [4] décrit l'installation de la CLI ArgoCD.

Les commandes d'installation sont :

```
curl -sSL -o argocd-linux-amd64 https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64
sudo install -m 555 argocd-linux-amd64 /usr/local/bin/argocd
rm argocd-linux-amd64
```

4.

https://argo-cd.readthedocs.io/en/stable/cli_installation/#download-with-curl

Activation de l'accès au serveur d'API d'ArgoCD

Le lien suivant[5] décrit les différentes méthodes d'accès au serveur d'API d'ArgoCD :

Les commandes d'installation sont :

```
$ kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
service/argocd-server patched
```

5. https://argo-cd.readthedocs.io/en/stable/getting_started/#3-access-the-argo-cd-api-server

Build de l'image Docker Spilo

Le Dockerfile définissant l'image Spilo est disponible à l'URL suivante : `https://github.com/zalando/spilo`

La méthode de génération de l'image est décrite dans : `https://github.com/zalando/spilo#how-to-build-this-docker-image`

```
$:~/spilo/postgres-appliance$ docker build --tag dnum-test .
[+] Building 9762.1s (33/33) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 3.01kB
 => [internal] load .dockerignore
 => => transferring context: 2B
…
 => => exporting layers
 => => writing image sha256:52afd69aff9414333220ec408283a8ff20e2162e703c6ab5afd5090d9d62e4e0
 => => naming to docker.io/library/dnum-test
```

Le build dure un peu moins de **2h43min**.

Login avec la CLI d'ArgoCD

La commande suivante permet de récupérer le mot de passe initial de l'admin d'ArgoCD :

```
linagora@debian-cp:~$ argocd admin initial-password -n argocd
******************

 This password must be only used for first time login. We strongly recommend you
 update the password using `argocd account update-password`.
```

Login avec la CLI d'ArgoCD

```
$ kubectl get svc -n argocd
NAME                                       TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller           ClusterIP      10.109.27.242    <none>        7000/TCP,8080/TCP            6d16h
argocd-dex-server                          ClusterIP      10.100.63.110    <none>        5556/TCP,5557/TCP,5558/TCP   6d16h
argocd-metrics                             ClusterIP      10.102.19.237    <none>        8082/TCP                     6d16h
argocd-notifications-controller-metrics    ClusterIP      10.102.70.229    <none>        9001/TCP                     6d16h
argocd-redis                               ClusterIP      10.104.253.209   <none>        6379/TCP                     6d16h
argocd-repo-server                         ClusterIP      10.111.28.234    <none>        8081/TCP,8084/TCP            6d16h
argocd-server                              LoadBalancer   10.109.47.144    <pending>     80:30235/TCP,443:30885/TCP   6d16h
argocd-server-metrics                      ClusterIP      10.98.167.31     <none>        8083/TCP                     6d16h
linagora@debian-cp:~$ argocd login 10.109.47.144
WARNING: server certificate had error: x509: cannot validate certificate for 10.109.47.144 because it doesn't contain
any IP SANs. Proceed insecurely (y/n)? y
Username: admin
Password:
'admin:login' logged in successfully
Context '10.109.47.144' updated
```

Ajout de l'application postgres-operator dans ArgoCD

- ArgoCD se met à l'écoute des changements d'un projet de déploiement dans un dépôt.
- L'étape suivante "abonne" ArgoCD au dépôt Github sur lequel est enregistré le déploiement de l'opérateur postgres
- La commande qui lie ArgoCD au dépôt de l'opérateur est la suivante :

```
argocd app create postgres-operator \
--repo https://github.com/simonelbaz/postgres-operator.git \
--path manifests \
--dest-server https://kubernetes.default.svc \
--revision poc-argocd \
--dest-namespace default
```

Ajout de l'application postgres-operator dans ArgoCD

- Cette commande est décrite en détail dans la documentation officielle [6]
- Pour information, le dépôt ci-dessus est un fork du dépôt officiel
- L'ensemble des modifications réalisées pour le POC sont tracées dans la branche *poc-argocd*

6.

```
https://argo-cd.readthedocs.io/en/stable/user-guide/commands/
argocd_app_create/
```

Ajout de l'application postgres-operator dans ArgoCD

- Pour vérifier que l'application a correctement été ajoutée dans ArgoCD :

```
$ argocd app list
NAME                      CLUSTER                         NAMESPACE  PROJECT  STATUS   HEALTH   SYNCPOLICY
argocd/postgres-operator  https://kubernetes.default.svc  default    default  Unknown  Healthy  <none>
```

- Selon le lien suivant `https://argo-cd.readthedocs.io/en/stable/user-guide/tool_detection/`, le script kustomization.yaml est automatiquement détecté par ArgoCD

Kustomize & ArgoCD

- ArgoCD est capable de s'interfacer avec Kustomize `https://kustomize.io/`.
- Zalando met à disposition un script Yaml **manifests/kustomization.yaml**

```
$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- configmap.yaml
- operator-service-account-rbac.yaml
- postgres-operator.yaml
- api-service.yaml
```

- Ce dernier facilite le déploiement de l'opérateur dans l'environnement k8s
- Une page synthétisant l'utilisation de Kustomize est disponible à l'URL suivante : `https://kubectl.docs.kubernetes.io/guides/introduction/kustomize/`

Intégration CI/CD & ArgoCD

- Le lien suivant décrit l'intégration d'un outil de CI/CD et ArgoCD.
  `https://argo-cd.readthedocs.io/en/stable/user-guide/ci_automation/`
- Globalement la mise à jour d'un cluster k8s par l'intermédiaire d'ArgoCD se déroule en 2 phases :
  - récupération du dépôt git
  - la phase de patch
  - la phase de sync

Phase de patch & ArgoCD

- La 1re étape de la phase de patch est de récupérer le dépôt du projet :

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

- Il devient possible de patcher avec la commande **kustomize** ou la commande **kubectl**
  - Patch avec kustomize (nécessite l'installation de kustomize) :

Phase de patch avec kustomize

- L'URL suivante décrit l'installation de kustomize : `https://kubectl.docs.kubernetes.io/installation/kustomize/binaries/`
- Pour installer kustomize, merci de lancer la commande suivante :

```
curl -s "https://raw.githubusercontent.com/kubernetes-sigs/kustomize/master/hack/install_kustomize.sh"  | bash
```

- kustomize peut servir à l'édition de kustomization.yaml.
- Dans le cadre du POC, cela n'a pas été nécessaire

Phase de patch avec kubectl

- On liste dans un 1er temps l'ensemble des ConfigMaps :

```
$ kubectl get configmaps
NAME                 DATA   AGE
kube-root-ca.crt     1      98d
postgres-operator    59     97d
```

- Pour lister la ConfigMap qui définit l'image **spilo** déployée :

```
$ kubectl get configmap postgres-operator -o yaml
apiVersion: v1
data:
  api_port: "8080"
…
  docker_image: ghcr.io/zalando/spilo-15:2.1-p9
…
kind: ConfigMap
metadata:
  creationTimestamp: "2023-03-16T22:20:28Z"
  name: postgres-operator
  namespace: default
…
```

Phase de patch avec kubectl

La version de l'image spilo est modifiée :

```
kubectl patch --local -f configmap.yaml \
-p '"data":"docker_image":"ghcr.io/zalando/spilo-15:3.0-p1"' \
-o yaml > configmap.yaml.new
mv configmap.yaml.new configmap.yaml
```

Phase de sync & ArgoCD

- Une fois la modification réalisée, il est nécessaire de la pusher vers le dépôt sur lequel se synchronise ArgoCD

```
git add . -m "Mise à jour de l'image"
git push
```

- Lancer ensuite les commandes suivantes dans le pipeline de la CI :

```
export ARGOCD_SERVER=argocd.mycompany.com
export ARGOCD_AUTH_TOKEN=<JWT token generated from project>
curl -sSL -o /usr/local/bin/argocd https://$ARGOCD_SERVER/download/argocd-linux-amd64
argocd app sync nom application
argocd app wait nom application
```

ArgoCD - Etat du cluster avant le lancement du sync

```
$ git log | head -1
commit 90ad8c7aed92c6430bcb36ab23228a2d57c2715d
$ argocd app get argocd/postgres-operator
Name:               argocd/postgres-operator
Project:            default
Server:             https://kubernetes.default.svc
Namespace:          default
URL:                https://10.109.47.144/applications/postgres-operator
Repo:               https://github.com/simonelbaz/postgres-operator.git
Target:             poc-argocd
Path:               manifests
SyncWindow:         Sync Allowed
Sync Policy:        <none>
Sync Status:        OutOfSync from poc-argocd (90ad8c7)
Health Status:      Healthy

GROUP                       KIND                NAMESPACE  NAME                STATUS     HEALTH   HOOK  MESSAGE
                            ConfigMap           default    postgres-operator   OutOfSync
                            Service             default    postgres-operator   OutOfSync  Healthy
                            ServiceAccount      default    postgres-operator   OutOfSync
acid.zalan.do               postgresql          default    acid-test-cluster   OutOfSync
apps                        Deployment          default    postgres-operator   OutOfSync  Healthy
rbac.authorization.k8s.io   ClusterRole                    postgres-operator   OutOfSync
rbac.authorization.k8s.io   ClusterRole                    postgres-pod        OutOfSync
rbac.authorization.k8s.io   ClusterRoleBinding             postgres-operator   OutOfSync
```

ArgoCD - Etat du cluster avant le lancement du sync

- L'information renvoyée par la commande **argocd get** est cohérente.
- Il y a bien un décalage entre l'état du cluster et le dernier commit de la branche poc-argocd
- Vérification de la version de l'image spilo déployée :

```
$ kubectl describe pod acid-test-cluster-1 | grep spilo
Labels:           application=spilo
    Image:          ghcr.io/zalando/spilo-15:2.1-p9
      KUBERNETES_ROLE_LABEL:      spilo-role
      KUBERNETES_LABELS:          "application":"spilo"
```

ArgoCD - Lancement de la synchronisation

```
$ argocd app sync postgres-operator
TIMESTAMP                  GROUP                                KIND          NAMESPACE                  NAME    STATUS   HEALTH        HOOK  MESSAGE
2023-06-23T16:49:22+02:00  rbac.authorization.k8s.io  ClusterRoleBinding                    postgres-operator  OutOfSync
2023-06-23T16:49:22+02:00                             ConfigMap             default         postgres-operator  OutOfSync
2023-06-23T16:49:22+02:00                             Service               default         postgres-operator  OutOfSync  Healthy
2023-06-23T16:49:22+02:00                             ServiceAccount        default         postgres-operator  OutOfSync
2023-06-23T16:49:22+02:00  acid.zalan.do              postgresql            default         acid-test-cluster  OutOfSync
2023-06-23T16:49:22+02:00  apps                       Deployment            default         postgres-operator  OutOfSync  Healthy
2023-06-23T16:49:22+02:00  rbac.authorization.k8s.io  ClusterRole                           postgres-operator  OutOfSync
2023-06-23T16:49:22+02:00  rbac.authorization.k8s.io  ClusterRole                           postgres-pod  OutOfSync
2023-06-23T16:49:52+02:00         ServiceAccount     default     postgres-operator     Synced
2023-06-23T16:49:52+02:00         ConfigMap          default     postgres-operator     Synced
2023-06-23T16:49:53+02:00  rbac.authorization.k8s.io  ClusterRole                     postgres-operator    Synced
2023-06-23T16:49:53+02:00  rbac.authorization.k8s.io  ClusterRole                     postgres-pod    Synced
2023-06-23T16:49:55+02:00  rbac.authorization.k8s.io  ClusterRoleBinding              postgres-operator    Synced
2023-06-23T16:49:57+02:00         Service     default     postgres-operator     Synced  Healthy
2023-06-23T16:50:00+02:00  apps  Deployment     default     postgres-operator     Synced  Progressing
2023-06-23T16:50:01+02:00  acid.zalan.do              postgresql            default         acid-test-cluster  OutOfSync                  postgresql.acid.zalan.do/acid-test-cluster con
figured. Warning: resource postgresqls/acid-test-cluster is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on
 resources created declaratively by either  create --save-config or  apply. The missing annotation will be patched automatically.
2023-06-23T16:50:01+02:00                             ConfigMap             default         postgres-operator  Synced                  configmap/postgres-operator configured. Warnin
g: resource configmaps/postgres-operator is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources creat
ed declaratively by either  create --save-config or  apply. The missing annotation will be patched automatically.
2023-06-23T16:50:01+02:00  rbac.authorization.k8s.io  ClusterRole           default         postgres-pod   Running   Synced         clusterrole.rbac.authorization.k8s.io/postgres
-pod reconciled. reconciliation required update. clusterrole.rbac.authorization.k8s.io/postgres-pod configured. Warning: resource clusterroles/postgres-pod is missing the kubectl.kubernetes.
io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declaratively by either  create --save-config or  apply. The missing ann
otation will be patched automatically.
2023-06-23T16:50:01+02:00  rbac.authorization.k8s.io  ClusterRole           default         postgres-operator  Running   Synced         clusterrole.rbac.authorization.k8s.io/postgres
-operator reconciled. reconciliation required update. clusterrole.rbac.authorization.k8s.io/postgres-operator configured. Warning: resource clusterroles/postgres-operator is missing the kube
ctl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declaratively by either  create --save-config or  apply.
 The missing annotation will be patched automatically.
2023-06-23T16:50:01+02:00  rbac.authorization.k8s.io  ClusterRoleBinding    default         postgres-operator  Running   Synced         clusterrolebinding.rbac.authorization.k8s.io/p
ostgres-operator reconciled. reconciliation required update. clusterrolebinding.rbac.authorization.k8s.io/postgres-operator configured. Warning: resource clusterrolebindings/postgres-operato
r is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declaratively by either  create --sa
ve-config or  apply. The missing annotation will be patched automatically.
2023-06-23T16:50:01+02:00                             ServiceAccount        default         postgres-operator  Synced                  serviceaccount/postgres-operator configured. W
arning: resource serviceaccounts/postgres-operator is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resou
rces created declaratively by either  create --save-config or  apply. The missing annotation will be patched automatically.
2023-06-23T16:50:01+02:00                             Service               default         postgres-operator  Synced  Healthy         service/postgres-operator configured. Warning:
 resource services/postgres-operator is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created d
eclaratively by either  create --save-config or  apply. The missing annotation will be patched automatically.
2023-06-23T16:50:01+02:00  apps                       Deployment            default         postgres-operator  Synced  Progressing     deployment.apps/postgres-operator configured.
Warning: resource deployments/postgres-operator is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resource
s created declaratively by either  create --save-config or  apply. The missing annotation will be patched automatically.
2023-06-23T16:50:02+02:00  apps       Deployment     default     postgres-operator     Synced  Healthy         deployment.apps/postgres-operator configured. Warning: resource deploym
ents/postgres-operator is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declaratively b
y either  create --save-config or  apply. The missing annotation will be patched automatically.
2023-06-23T16:50:02+02:00  acid.zalan.do  postgresql     default     acid-test-cluster     Synced                  postgresql.acid.zalan.do/acid-test-cluster configured. Warning: resourc
e postgresqls/acid-test-cluster is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declar
atively by either  create --save-config or  apply. The missing annotation will be patched automatically.
_
```

ArgoCD - Lancement de la synchronisation

```
Name:               argocd/postgres-operator
Project:            default
Server:             https://kubernetes.default.svc
Namespace:          default
URL:                https://10.109.47.144/applications/postgres-operator
Repo:               https://github.com/simonelbaz/postgres-operator.git
Target:             poc-argocd
Path:               manifests
SyncWindow:         Sync Allowed
Sync Policy:        <none>
Sync Status:        Synced to poc-argocd (90ad8c7)
Health Status:      Healthy

Operation:          Sync
Sync Revision:      90ad8c7aed92c6430bcb36ab23228a2d57c2715d
Phase:              Succeeded
Start:              2023-06-23 16:49:21 +0200 CEST
Finished:           2023-06-23 16:50:00 +0200 CEST
Duration:           39s
Message:            successfully synced (all tasks run)

GROUP                       KIND                NAMESPACE  NAME               STATUS   HEALTH   HOOK  MESSAGE
                            ServiceAccount      default    postgres-operator  Synced                 serviceaccount/postgres-operator configured. Warning: resource serviceaccounts/postgres-o
perator is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declaratively by either  creat
e --save-config or  apply. The missing annotation will be patched automatically.
                            ConfigMap           default    postgres-operator  Synced                 configmap/postgres-operator configured. Warning: resource configmaps/postgres-operator is
 missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declaratively by either  create --save-c
onfig or  apply. The missing annotation will be patched automatically.
rbac.authorization.k8s.io   ClusterRole          default    postgres-pod       Running  Synced        clusterrole.rbac.authorization.k8s.io/postgres-pod reconciled. reconciliation required up
date. clusterrole.rbac.authorization.k8s.io/postgres-pod configured. Warning: resource clusterroles/postgres-pod is missing the kubectl.kubernetes.io/last-applied-configuration annotation wh
ich is required by  apply.  apply should only be used on resources created declaratively by either  create --save-config or  apply. The missing annotation will be patched automatically.
rbac.authorization.k8s.io   ClusterRole          default    postgres-operator  Running  Synced        clusterrole.rbac.authorization.k8s.io/postgres-operator reconciled. reconciliation requir
ed update. clusterrole.rbac.authorization.k8s.io/postgres-operator configured. Warning: resource clusterroles/postgres-operator is missing the kubectl.kubernetes.io/last-applied-configuratio
n annotation which is required by  apply.  apply should only be used on resources created declaratively by either  create --save-config or  apply. The missing annotation will be patched auto
matically.
rbac.authorization.k8s.io   ClusterRoleBinding  default    postgres-operator  Running  Synced        clusterrolebinding.rbac.authorization.k8s.io/postgres-operator reconciled. reconciliation
 required update. clusterrolebinding.rbac.authorization.k8s.io/postgres-operator configured. Warning: resource clusterrolebindings/postgres-operator is missing the kubectl.kubernetes.io/last
-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declaratively by either  create --save-config or  apply. The missing annotation
 will be patched automatically.
                            Service             default    postgres-operator  Synced   Healthy       service/postgres-operator configured. Warning: resource services/postgres-operator is mis
sing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declaratively by either  create --save-confi
g or  apply. The missing annotation will be patched automatically.
apps                        Deployment          default    postgres-operator  Synced   Healthy       deployment.apps/postgres-operator configured. Warning: resource deployments/postgres-oper
ator is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declaratively by either  create -
-save-config or  apply. The missing annotation will be patched automatically.
acid.zalan.do               postgresql          default    acid-test-cluster  Synced                 postgresql.acid.zalan.do/acid-test-cluster configured. Warning: resource postgresqls/acid
-test-cluster is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by  apply.  apply should only be used on resources created declaratively by either
 create --save-config or  apply. The missing annotation will be patched automatically.
rbac.authorization.k8s.io   ClusterRole                  postgres-operator  Synced
rbac.authorization.k8s.io   ClusterRole                  postgres-pod       Synced
rbac.authorization.k8s.io   ClusterRoleBinding           postgres-operator  Synced
```

Installation d'argo-rollouts

```
$ kubectl create namespace argo-rollouts
namespace/argo-rollouts created
$ kubectl apply -n argo-rollouts -f https://github.com/argoproj/argo-rollouts/releases/latest/download/install.yaml
customresourcedefinition.apiextensions.k8s.io/analysisruns.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/analysistemplates.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/clusteranalysistemplates.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/experiments.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/rollouts.argoproj.io created
serviceaccount/argo-rollouts created
clusterrole.rbac.authorization.k8s.io/argo-rollouts created
clusterrole.rbac.authorization.k8s.io/argo-rollouts-aggregate-to-admin created
clusterrole.rbac.authorization.k8s.io/argo-rollouts-aggregate-to-edit created
clusterrole.rbac.authorization.k8s.io/argo-rollouts-aggregate-to-view created
clusterrolebinding.rbac.authorization.k8s.io/argo-rollouts created
configmap/argo-rollouts-config created
secret/argo-rollouts-notification-secret created
service/argo-rollouts-metrics created
deployment.apps/argo-rollouts created
```

Utilisation d'Argo Rollouts

- Dans le cadre du POC, il n'a pas été nécessaire d'utiliser Argo Rollouts.
- Argo CD a été suffisant pour "pusher" la nouvelle image Docker **spilo**

Réplicasets

L'opérateur PostgreSQL met en place des réplicats set :

```
$ kubectl get rs
NAME                                DESIRED   CURRENT   READY   AGE
postgres-operator-fcbd7cc96         1         1         1       101d
postgres-operator-ui-5579cc7779     1         1         1       101d
```

Détail des réplicasets

La commande suivante permet d'obtenir plus d'informations sur le réplicaset postgres-operator :

```
$ kubectl describe rs/postgres-operator-fcbd7cc96
Name:           postgres-operator-fcbd7cc96
Namespace:      default
Selector:       name=postgres-operator,pod-template-hash=fcbd7cc96
Labels:         name=postgres-operator
                pod-template-hash=fcbd7cc96
Annotations:    deployment.kubernetes.io/desired-replicas: 1
                deployment.kubernetes.io/max-replicas: 1
                deployment.kubernetes.io/revision: 1
Controlled By:  Deployment/postgres-operator
Replicas:       1 current / 1 desired
Pods Status:    1 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:           name=postgres-operator
                    pod-template-hash=fcbd7cc96
  Service Account:  postgres-operator
  Containers:
   postgres-operator:
    Image:      registry.opensource.zalan.do/acid/postgres-operator:v1.9.0
    Port:       <none>
    Host Port:  <none>
    Limits:
      cpu:     500m
      memory:  500Mi
    Requests:
      cpu:     100m
      memory:  250Mi
    Environment:
      CONFIG_MAP_NAME:  postgres-operator
    Mounts:             <none>
  Volumes:              <none>
Events:                 <none>
```

StatefulSets

- La définition officielle des StatefulSets est disponible à l'URL suivante [7].
- Les Pods portants l'image Spilo sont déployés en se basant sur des statefulsets :

```
$ kubectl get statefulsets
NAME                READY   AGE
acid-test-cluster   2/2     88d
```

7.

```
https://kubernetes.io/docs/concepts/workloads/controllers/
statefulset/
```

Détails des statefulSets

```
$ kubectl describe statefulset acid-test-cluster
Name:               acid-test-cluster
Namespace:          default
CreationTimestamp:  Wed, 29 Mar 2022 19:51:12 +0200
Selector:           application=spilo,cluster-name=acid-test-cluster
Labels:             application=spilo
                    cluster-name=acid-test-cluster
                    team=acid
Annotations:        <none>
Replicas:           2 desired | 2 total
\testOfUpdate Strategy:    OnDelete}
Pods Status:        2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:           application=spilo
                    cluster-name=acid-test-cluster
                    team=acid
   Service Account:  postgres-pod
   Init Containers:
    date:
     Image:      busybox
     Port:       <none>
     Host Port:  <none>
     Command:
      /bin/date
     Environment:  <none>
     Mounts:       <none>
   Containers:
    postgres:
     \testOf{Image:      ghcr.io/zalando/spilo-15:2.1-p9}
     Ports:      8008/TCP, 5432/TCP, 8080/TCP
     Host Ports:  0/TCP, 0/TCP, 0/TCP
     Limits:
       cpu:     500m
       memory:  500Mi
     Requests:
       cpu:     100m
       memory:  100Mi
     Environment:
       SCOPE:                          acid-test-cluster
       PGROOT:                         /home/postgres/pgdata/pgroot
       POD_IP:                          (v1:status.podIP)
       POD_NAMESPACE:                   (v1:metadata.namespace)
       PGUSER_SUPERUSER:               postgres
       KUBERNETES_SCOPE_LABEL:         cluster-name
       KUBERNETES_ROLE_LABEL:          spilo-role
       PGPASSWORD_SUPERUSER:           <set to the key 'password' in secret 'postgres.acid-test-cluster.credentials.postgresql.acid.zalan.do'> Optional: false
       PGUSER_STANDBY:                 standby
       PGPASSWORD_STANDBY:             <set to the key 'password' in secret 'standby.acid-test-cluster.credentials.postgresql.acid.zalan.do'> Optional: false
       PAM_OAUTH2:                     https://info.example.com/oauth2/tokeninfo?access_token= uid realm=/employees
       HUMAN_ROLE:                     zalandos
       ENABLE_WAL_PATH_COMPAT:         true
       PGVERSION:                      15
       KUBERNETES_LABELS:              {"application":"spilo"}
       SPILO_CONFIGURATION:            {"postgresql":{"parameters":{"shared_buffers":"32MB"}},
        \labels:
       DCS_ENABLE_KUBERNETES_API:  true
     Mounts:
       /dev/shm from dshm (rw)
       /home/postgres/pgdata from pgdata (rw)
       /tmp/empty from empty (rw)
   Volumes:
    dshm:
     Type:       EmptyDir (a temporary directory that shares a pod's lifetime)
     Medium:     Memory
     SizeLimit:  <unset>
    empty:
     Type:       EmptyDir (a temporary directory that shares a pod's lifetime)
     Medium:
     SizeLimit:  <unset>
  Volume Claims:
    Name:          pgdata
    StorageClass:  rook-ceph-block
    Labels:        application=spilo
                   cluster-name=acid-test-cluster
                   team=acid
    Annotations:   <none>
    Capacity:      5Gi
    Access Modes:  [ReadWriteOnce]
Events:            <none>
```

Détails des statefulSets

- Comme indiqué dans les détails du statefulset, la politique de mise à jour est OnDelete.
- D'après la documentation officielle, cela signifie que la mise à jour des pods passe par une suppression des pods.

Traitement des statefulsets par Argo

- A priori, le traitement des statefulsets peut avoir des limitations [8]
- Cette limitation a été corrigée

8.

```
https://argo-cd.readthedocs.io/en/stable/faq/#why-is-my-application-stuck-in-progressing-state
```

Mise à jour des images Spilo - Upgrade mineur

- Le script de création du cluster mis à disposition par le dépôt de l'opérateur PostgreSQL est : **manifests/complete-postgres-manifest.yaml**
- Il va être copié dans un répertoire dédié (**deployment**) pour gérer sa synchronisation avec ArgoCD

```
argocd app create postgres-deployment \
--repo https://github.com/simonelbaz/postgres-operator.git \
--path deployment \
--dest-server https://kubernetes.default.svc \
--revision poc-argocd \
--dest-namespace default
```

- Le script est modifié avec la nouvelle version de l'image spilo :

```
$ cat deployment/complete-postgres-manifest.yaml  | grep dockerImage
  dockerImage: ghcr.io/zalando/spilo-15:3.0-p1
```

Mise à jour des images Spilo - Upgrade mineur

```
$ argocd app sync postgres-deployment
TIMESTAMP                  GROUP                KIND   NAMESPACE                  NAME    STATUS    HOOK  MESSAGE
2023-06-26T18:44:36+02:00  acid.zalan.do  postgresql     default     acid-test-cluster  OutOfSync
2023-06-26T18:45:07+02:00  acid.zalan.do  postgresql     default     acid-test-cluster  OutOfSync
postgresql.acid.zalan.do/acid-test-cluster configured
2023-06-26T18:45:09+02:00  acid.zalan.do  postgresql     default     acid-test-cluster    Synced
postgresql.acid.zalan.do/acid-test-cluster configured

Name:               argocd/postgres-deployment
Project:            default
Server:             https://kubernetes.default.svc
Namespace:          default
URL:                https://10.109.47.144/applications/postgres-deployment
Repo:               https://github.com/simonelbaz/postgres-operator.git
Target:             poc-argocd
Path:               deployment
SyncWindow:         Sync Allowed
Sync Policy:        <none>
Sync Status:        Synced to poc-argocd (cfb2094)
Health Status:      Healthy

Operation:          Sync
Sync Revision:      cfb20940760cf96ced4a5872f161c280457c9784
Phase:              Succeeded
Start:              2023-06-26 18:44:34 +0200 CEST
Finished:           2023-06-26 18:45:06 +0200 CEST
Duration:           32s
Message:            successfully synced (all tasks run)

GROUP          KIND        NAMESPACE  NAME               STATUS  HEALTH  HOOK  MESSAGE
acid.zalan.do  postgresql  default    acid-test-cluster  Synced                postgresql.acid.zalan.do/acid-test-cluster configured
```

Accès client à la base de données

- Le lien suivant décrit comment se logger à la base de données PostgreSQL [9]
- Récupération du pod master :

```
# get name of master pod of acid-minimal-cluster
export PGMASTER=$(kubectl get pods \
-o jsonpath=.items..metadata.name \
-l application=spilo,cluster-name=acid-minimal-cluster,spilo-role=master \
-n default)

# set up port forward
kubectl port-forward $PGMASTER 6432:5432 -n default
```

9. https://github.com/zalando/postgres-operator/blob/master/docs/user.md#connect-to-postgresql

Accès client à la base de données

- Connexion depuis un tunnel SSH :

```
export PGPASSWORD=$(kubectl get secret \
postgres.acid-minimal-cluster.credentials.postgresql.acid.zalan.do \
-o 'jsonpath=.data.password' | base64 -d)
export PGSSLMODE=require
psql -U postgres -h localhost -p 6432
```

Point-in-time recovery PITR

Vidéo partagée *7-Postgres_operator_pitr.mkv*

Failover - Changement du nombre de pods

Vidéo partagée *6-Postgres_operator_failover_podnumberchange.mkv*

Mise à jour mineure de PostgreSQL

Vidéo partagée *6-Postgres_operator_minorupgrade.mkv*

Mise à jour majeure de PostgreSQL

Vidéo partagée *6-Postgres_operator_major_upgrade.mkv*

Supervision

Supervision du cluster k8s - Prometheus

- Le cluster k8s nécessite une supervision.
- Prometheus est une solution de supervision open source répandue
- Elle possède un opérateur **kube-prometheus** dont le déploiement est décrit depuis le site Github

Déploiement de kube-prometheus

- Le lien suivant décrit les commandes de déploiement [10]

10.

```
https://github.com/prometheus-operator/kube-prometheus#quickstart
```

Déploiement de kube-prometheus

- Les commandes suivantes décrivent le déploiement de l'opérateur kube-prometheus

```
$ git clone https://github.com/prometheus-operator/kube-prometheus.git
$ kubectl apply --server-side -f manifests/setup
customresourcedefinition.apiextensions.k8s.io/alertmanagerconfigs.monitoring.coreos.com serverside-applied
…
$ kubectl wait \
  --for condition=Established \
  --all CustomResourceDefinition \
  --namespace=monitoring
customresourcedefinition.apiextensions.k8s.io/alertmanagerconfigs.monitoring.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/alertmanagers.monitoring.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/analysisruns.argoproj.io condition met
…
 $ kubectl apply -f manifests/
…
```

- la dernière commande s'est terminée en erreur et a nécessité un double lancement

DaemonSets - Prometheus

- Le déploiement des nodes exporter de Prometheus est instancié sous forme de DaemonSets
- Liste des daemonsets

```
$ kubectl get daemonsets -n monitoring
NAME            DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
node-exporter   10        10        10      10           10          kubernetes.io/os=linux   110m
```

Accès à l'UI de Prometheus

- le lien suivant décrit le mode opératoire pour accéder à l'UI de Prometheus [11]
- Depuis le nœud control plane, ouvrir un port forward :

```
$ kubectl --namespace monitoring port-forward svc/prometheus-k8s 9090
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::1]:9090 -> 9090
```

- 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 :

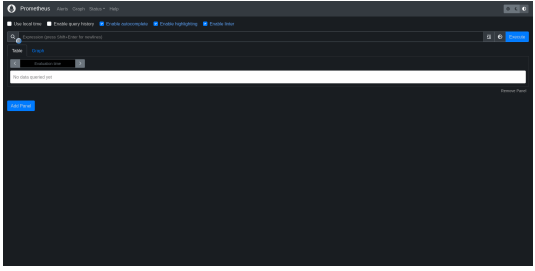
```
ssh -L 9095:10.106.57.137:9090 dgfip-k8s
```

- Lancer le navigateur pour accéder à l'URL `http://localhost:9095/`

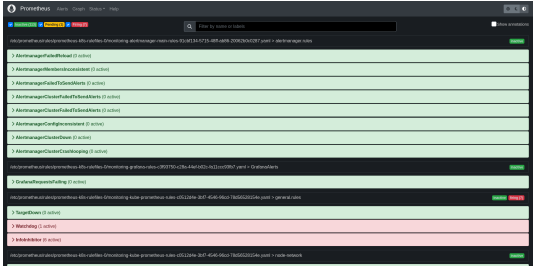
11.

```
https://github.com/prometheus-operator/kube-prometheus/blob/main/docs/access-ui.md
```

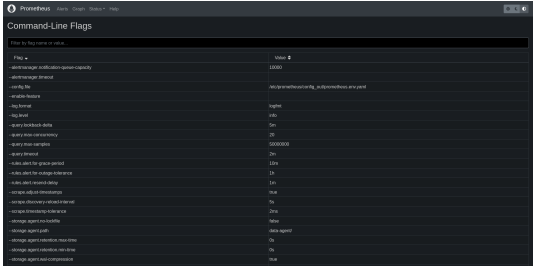
Ecran d'accueil de l'UI de Prometheus



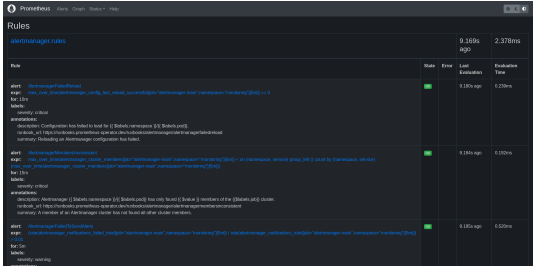
Ecran des alertes de l'UI de Prometheus



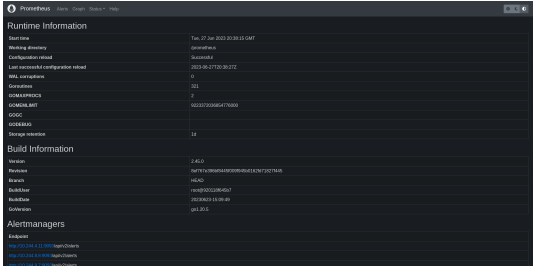
Flags de la ligne de commandes - Prometheus



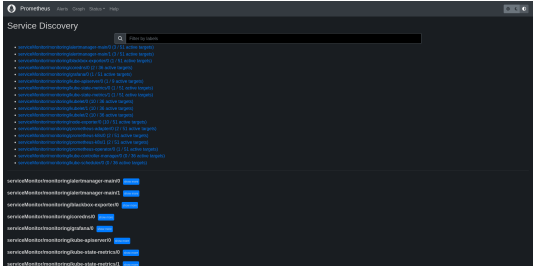
Rules - Prometheus



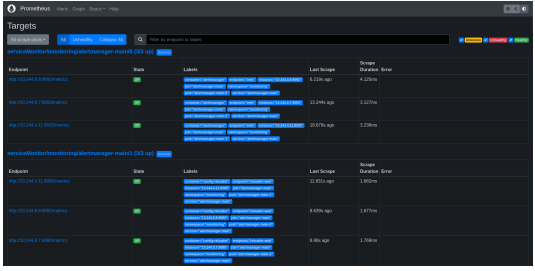
Runtime - Prometheus



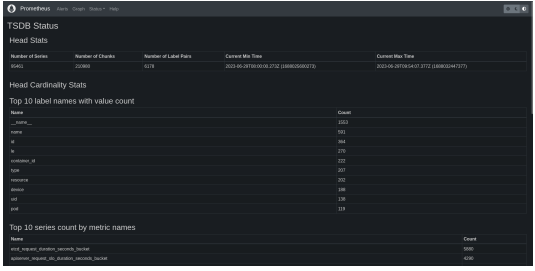
Service Discovery - Prometheus



Targets - Prometheus



TSDB Status - Prometheus



Déploiement du node exporter de PostgreSQL

- Le précédent déploiement permet de monitorer le cluster k8s
- Il est maintenant temps de monitorer la base de données PostgreSQL
- La communauté Prometheus met à disposition l'exporter PostgreSQL à l'URL suivante :
  `https://github.com/prometheus-community/postgres_exporter`
- Il est possible de déployer l'exporter postgres sur un pod de type **sidecar**
- Par simplicité du POC, l'image Docker mise à disposition par la communauté a été employée

Déploiement du node exporter de PostgreSQL

- Les commandes de déploiement de l'exporter PostgreSQL sont décrites dans l'URL suivante : `https://github.com/prometheus-community/postgres_exporter#quick-start`

Lancement de l'exporter PostgreSQL

Dans un 1er temps, on met en place le classique port forward depuis le control plane :

```
$ export PGMASTER=$(kubectl get pods \
-o jsonpath=.items..metadata.name \
-l application=spilo,cluster-name=acid-test-cluster,spilo-role=master \
-n default)
$ kubectl port-forward $PGMASTER 6432:5432 -n default
Forwarding from 127.0.0.1:6432 -> 5432
Forwarding from [::1]:6432 -> 5432
```

Dans un 2e temps, le Docker est lancé depuis la VM dédiée :

```
sudo docker run --net=host  -p 9187:9187 \
-e DATA_SOURCE_NAME="postgresql://postgres:*******@localhost:6432/postgres?sslmode=require" \
-v /home/linagora/postgres_exporter.yml:/home/linagora/postgres_exporter.yml \
quay.io/prometheuscommunity/postgres-exporter \
--config.file="/home/linagora/postgres_exporter.yml" \
--log.level="debug"
```

Accès à l'UI de l'exporter Postgres

- Ouvrir un tunnel SSH vers le Docker exporter Postgres :
  ```
  ssh -L 9187:localhost:9187 dgfip-prometheus
  ```
- Lancer le navigateur pour accéder à l'URL `http://localhost:9187/`

UI de l'exporter Postgres

Extrait de métriques remontées par l'exporter PostgreSQL

```
# HELP pg_statio_user_tables_heap_blocks_read Number of disk blocks read from this table
# TYPE pg_statio_user_tables_heap_blocks_read counter
pg_statio_user_tables_heap_blocks_readdatname="postgres",relname="job",schemaname="cron" 0
pg_statio_user_tables_heap_blocks_readdatname="postgres",relname="job_run_details",schemaname="cron" 0
pg_statio_user_tables_heap_blocks_readdatname="postgres",relname="postgres_log",schemaname="public" 0
# HELP pg_statio_user_tables_idx_blocks_hit Number of buffer hits in all indexes on this table
# TYPE pg_statio_user_tables_idx_blocks_hit counter
pg_statio_user_tables_idx_blocks_hitdatname="postgres",relname="job",schemaname="cron" 0
pg_statio_user_tables_idx_blocks_hitdatname="postgres",relname="job_run_details",schemaname="cron" 0
pg_statio_user_tables_idx_blocks_hitdatname="postgres",relname="postgres_log",schemaname="public" 0
# HELP pg_statio_user_tables_idx_blocks_read Number of disk blocks read from all indexes on this table
# TYPE pg_statio_user_tables_idx_blocks_read counter
pg_statio_user_tables_idx_blocks_readdatname="postgres",relname="job",schemaname="cron" 0
pg_statio_user_tables_idx_blocks_readdatname="postgres",relname="job_run_details",schemaname="cron" 0
pg_statio_user_tables_idx_blocks_readdatname="postgres",relname="postgres_log",schemaname="public" 0
# HELP pg_statio_user_tables_tidx_blocks_hit Number of buffer hits in this table's TOAST table indexes (if any)
# TYPE pg_statio_user_tables_tidx_blocks_hit counter
pg_statio_user_tables_tidx_blocks_hitdatname="postgres",relname="job",schemaname="cron" 0
pg_statio_user_tables_tidx_blocks_hitdatname="postgres",relname="job_run_details",schemaname="cron" 0
pg_statio_user_tables_tidx_blocks_hitdatname="postgres",relname="postgres_log",schemaname="public" 0
# HELP pg_statio_user_tables_tidx_blocks_read Number of disk blocks read from this table's TOAST table indexes (if any)
# TYPE pg_statio_user_tables_tidx_blocks_read counter
pg_statio_user_tables_tidx_blocks_readdatname="postgres",relname="job",schemaname="cron" 0
pg_statio_user_tables_tidx_blocks_readdatname="postgres",relname="job_run_details",schemaname="cron" 0
pg_statio_user_tables_tidx_blocks_readdatname="postgres",relname="postgres_log",schemaname="public" 0
```

Sommaire

Sommaire

Fin du document