Instruction.md

Lab Kubernetes

Dans ce Lab Kubernetes, nosus allons apprendre tous les concepts de Kubernetes que nous avons vu avec le déploiement d'une application concrête. Il s'agira de Wordpress avec sa Base de donnée MysQL.

Mise en place de l'environnement

Choisissez un environnement Lab qui vous convient Utilisez le Vagantfile mise à disposition pour

Exploration de Kubernetes et manipulation de base

Tout ce qui sera demandé devra être fait en ligne de commande avec l'outil `kubectl` 2 manière de

- Lister les noeuds de votre cluster. Combien de master avez-vous et combien de worker nodes ?

▼ Correction

```
kubectl get nodes
```

- Combien de namespaces avez vous dans votre cluster ? Listez les

▼ Correction

```
kubectl get namespaces
```

- Lister les Pods de tous les namespaces

▼ Correction

```
kubectl get pods –A
```

- lancer un Pod du nom de webserver à l'aide de l'image `nginx:latest` de la manière impérative et visualiser sur quel noeud il a été déloyé
- Dans quel namespaces ?

- QUel est son état ?

▼ Correction

```
kubectl run webserver --image nginx
kubectl get pods -o wide
# cette commande affiche le nom des namespaces également
kubectl get pods -o wide -A
# describe
kubectl describe pod webserver
```

il a été déployé dans le namespaces par défaut.

▼ Correction
(Source https://kubernetes.io/fr/docs/concepts/overview/working-with-objects/namespaces/) Kubernetes démarre avec quatre namespaces initiaux:

- default Le namespace par défaut pour les objets sans mention d'autre namespace
- kube-system Le namespace pour les objets créés par Kubernetes lui-même
- kube-public Ce namespace est créé automatiquement et est visible par tous les utilisateurs (y compris ceux qui ne sont pas authentifiés). Ce namespace est principalement réservé à l'utilisation du cluster, au cas où certaines ressources devraient être disponibles publiquement dans l'ensemble du cluster. L'aspect public de ce namespace n'est qu'une convention, pas une exigence.
- kube-node-lease Ce namespace contient les objets de bail associés à chaque nœud, ce qui améliore les performances des pulsations du nœud à mesure que le cluster évolue.

Lorsque vous emettez des requètes Kubectl, il se connecte au Cluster avec des setting par défaut du Context. Il est possible de changer de manière permanent ce namespace par défaut en modifiant votre context de connection.

```
kubectl config set-context --current --namespace=<insert-namespace-name-here>
```

- Créer un namespace `k8s-lab` et lancez un Pod du nom de `tools` image:busybox exécutant la commande `sleep 1000`

▼ Correction

```
kubectl create ns k8s-lab
kubectl run tools --image busybox -n k8s-lab -- sleep 1000
```

- Tentez de créer un pod du même nom avec une image de votre choix dans le même namespace précédent `k8s-lab`
- Arrivez-vous à le faire ? Si non pourquoi ?

▼ Correction

Dans un namespace, le nom d'une ressource doit être unique, un peu comme dans un espace de nom DNS

- Faite une description du pod webserver et une description du pod tools

▼ Correction

```
kubectl describe pods webserver
kubectl describe pods tools -n k8s-lab
```

- Quels sont les IP des Pods précédents ?
- Accédez à l'intérieur du Pod `tools` et emmettez une requete vers le Pod `Webserver` : un Ping, curl...

▼ Correction

```
kubectl exec -it -n k8s-lab tool -- sh
/ # ping 192.168.118.29
PING 192.168.118.29 (192.168.118.29): 56 data bytes
64 bytes from 192.168.118.29: seq=0 ttl=63 time=0.268 ms
64 bytes from 192.168.118.29: seq=1 ttl=63 time=0.063 ms
```

- Si le Ping passe, pourquoi selon vous c'est passé et soi non pour quoi ? est-ce que le fait que les ressources soient dans differents namespaces empêchent les communications ?

▼ Correction

Non, les namespaces ne limitent pas les communications

- Comment regarder le positionnement des Pods ?

▼ Correction

Soit en faisant le Describe comme précédament ==> il y'a une section `Node: k8s-worker-1/192.168.56.12` Soit en affichant les pods avec l'option `-o wide`

```
kubectl get pods -o wide
```

Exploration des autres ressources

- Créez un déploiement de 3 pods nginx, le nom du déploiement sera `frontend`

▼ Correction

```
kubectl create deployment frontend --image nginx --replicas 3
```

je peux modifier le deploiement en faisant `kubectl edit deployment frontend`

- quelles sont les ressources qui ont été créés par cette commande précédente ?

▼ Correction
Lorsqu'un déploiement est créé, en dehors de la ressource deployment elle même, derrière une ressource Replicaset du même nom dérivé est créé

```
kubectl get deployment
kubectl get replicaset
NAME                    DESIRED   CURRENT   READY   AGE
frontend-7f9649877f     3         3         3       2m40s
```

Ce replicat va engendrer le nombre de pods exprimé dont les noms sont à leur tour dérivés du nom replicaset

```
kubectl get pods
NAME                           READY   STATUS             RESTARTS        AGE
frontend-7f9649877f-29c6c      1/1     Running            0               5m36
frontend-7f9649877f-fq5v7      1/1     Running            0               5m36
frontend-7f9649877f-pkjst      1/1     Running            0               5m36
```

- Comment allez-vous accéder à ce déploiement ?

▼ Correction
Un déploiement dans Kubernetes n'a du sens que pour le Control plane. Derrière le déploiement en réalité c'est des Pods/Conteneurs qui seront tangibles aux quels les utilisateurs devront accéder. Donc de ce fait pour accéder au déploiement créé, il suffira en fait d'accéder aux Pods/conteneurs individuels créés, et donc par leurs IPs. avec l'option `-o wide` ou le Describe vous pouvez obtenur leurs IPs. Bien évidemment, c'est fastidieux car il faut manuellement les chercher. Pour rendre facile l'accès de ces 3 Pods/conteneurs créés nous aurons besoin d'un object Service qui va configurer une sorte de loadbalancer entre les 3 pods. Désormais donc, au lieu de chercher les iP des 3 conteneurs, il nous suffira d'accéder au endpoint du service et le tour est joué.

- Créer un service du nom de `frontend-http` de type ClusterIP qui expose le Déploiement `frontend` au port 80

```
kubectl expose deployment frontend --port=80 --target-port=80 --name=frontend-
```

- listez les services

▼ Correction

```
kubectl get service
```

- tester l'accès à votre déploiement avec un Pod que vous aller nommer `client` dont l'image est `curlimages/curl`

▼ Correction

```
kubectl run client --rm  -it --image curlimages/curl -- sh
$ curl http://frontend-http
```

```
vagrant@k8s-master-1:~$ kubectl get all -n kube-system
NAME                                           READY   STATUS    RESTARTS
pod/calico-kube-controllers-566654d67d-lrgpn   1/1     Running   0
pod/calico-node-gk6qg                          1/1     Running   5 (20h ago)
pod/calico-node-ljmmc                          1/1     Running   6 (20h ago)
pod/calico-node-qg7h7                          1/1     Running   5 (20h ago)
pod/coredns-565d847f94-6lfhq                   1/1     Running   6 (20h ago)
pod/coredns-565d847f94-bnqlv                   1/1     Running   6 (20h ago)
pod/etcd-k8s-master-1                          1/1     Running   11 (20h ago)
pod/kube-apiserver-k8s-master-1                1/1     Running   17 (20h ago)
pod/kube-controller-manager-k8s-master-1       1/1     Running   2 (20h ago)
pod/kube-proxy-g9s6l                           1/1     Running   5 (20h ago)
pod/kube-proxy-m29bk                           1/1     Running   5 (20h ago)
pod/kube-proxy-z7kn6                           1/1     Running   6 (20h ago)
pod/kube-scheduler-k8s-master-1                1/1     Running   0

# Dans notre environnement Kubernetes, il y'a un deploiment du nom de coredns
# Quand une ressources Service est créé avec son IP l'entré DNS correspondante
# à chaque Pod créés le fichier /ets/resolv.conf. et c'est de cette manière qu

NAME               TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
service/kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/T

NAME                         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABL
daemonset.apps/calico-node   3         3         3       3            3
daemonset.apps/kube-proxy    3         3         3       3            3

NAME                                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/calico-kube-controllers   1/1     1            1           40d
deployment.apps/coredns                   2/2     2            2           40d

NAME                                                 DESIRED   CURRENT   READY
replicaset.apps/calico-kube-controllers-566654d67d   1         1         1
replicaset.apps/coredns-565d847f94                   2         2         2
vagrant@k8s-master-1:~$ kubectl run client --rm  -it --image curlimages/curl -
If you don't see a command prompt, try pressing enter.
/ $ cat /etc/resolv.conf
search default.svc.cluster.local svc.cluster.local cluster.local
nameserver 10.96.0.10
options ndots:5
/ $
```

Vous remarquerez que nous avons maintenant juste besoin du nom du service qui joue d'office le role de nom DNS de mon déploiement. Lorsque le service est créé, Kube-proxy, Kubelet... mettent tout en oeuvre pour enregister un nom de domain du service et dans le conteneur/Pod le fichier /etc/resolv.conf indique le service DNs à contacter pour la résolution.

Deploiement des ressources Kubernetes avec YAML

Dans cette partie tout ce qu'on fera sera fait avec des manifest YAML Kubernetes

- Créez un déploiement du nom de `robots` , contenant 2 pods, l'image est busybox et la commande qu'il doit exécuter est le script suivant `while true; do tr -dc A-Za-z0-9 </dev/urandom| head -c 13 ; echo '' && sleep 1; done`

▼ Correction

Pour générer les fichier YAML correspondant, il existe une astuce simple qui est de combiner l'utilisation de kubectl avec l'option `--dry-run=client -o yaml` l'option `--dry-run=client` indique à l'PAI server que nous somme entrain de faire des simulations et qu'il ne devra pas créer la ressource l'otion `- o yaml` est de dire que nous voulons un output de l'objet au format YAML.

```
# Etape 1 => créer un fichier du script
$ cat << _EOF > script.sh
#!/bin/sh
while true; do tr -dc A-Za-z0-9 </dev/urandom| head -c 13 ; echo '' && sleep 1
_EOF
$ kubectl create deployment robots --replicas 2  --dry-run=client -o yaml --im
$ cat robots.yml
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: robots
  name: robots
spec:
  replicas: 2
  selector:
    matchLabels:
      app: robots
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: robots
```

```
      spec:
        containers:
        - command:
          - /bin/sh
          - -c
          - |-
            #!/bin/sh
            while true; do tr -dc A-Za-z0-9 </dev/urandom| head -c 13 ; echo ''
          image: busybox
          name: busybox
          resources: {}
  status: {}
```

Pour déployer donc nous ferons

```
kubectl apply -f robots.yml
```

Normalement si le déploiement est bien fait, vous pouvez voir les logs des pods

```
$ kubectl logs -f robots-577dbd8d5d-25mdb
F1buq9n5dhonu
QpcV5rcnf5yqm
eSn3raQVvBf0T
TA92AZM9Asjqr
```

- quel est l'avantage de déployer de cette manière ?

▼ Correction
Avec le Apply c'est la manière déclarative de déployer et c'est grâce à cela que nous pouvons faire du IaC => l'état **désiré** de notre Cluster est sauvegardé et toute modification autres que le Apply n'est pas accepé. Vous pouvez également utiliser kubectl apply pour mettre à jour une ressource existante en modifiant les paramètres de configuration dans le fichier manifeste donné et le Reapply.

- Reprenez la partie du Lab qui parle du dploiement de frontend avec la creation de service et faite le d'une manière déclarative.

▼ Correction
Pour avoir les infirmation sur les option d'une resources Kubernetes on peut se servir de `kubectl explain` ```bash kubectl explain deployment kubectl explain deployment --recursive ```

Mise en place d'un déploiement Wordpress avec persistance de données

Cette partie du Lab, nous allons reconstituer les manifestes nécessaires pour déployer une application wordpress avec sa base de données.

Frontend => https://github.com/kubernetes/examples/blob/master/mysql-wordpress-pd/mysql-deployment.yaml Backend => https://github.com/kubernetes/examples/blob/master/mysql-wordpress-pd/wordpress-deployment.yaml

- Créez un secret générique du nom de `mysql-password` qui devra contenir la clé/valeur suivante : `passwd => "Mettez un mot de passe quelconque"`

▼ Correction

```
kubectl create secret generic my-secret --from-literal=passwd=MonPassroot01 --
```

le fichier YAML généré est le suivant. Remarquez que c'est par ce que le type est secret que la valeur a été encodé en Base64.

```
apiVersion: v1
data:
  passwd: TW9uUGFzc3Jvb3QwMQ==
kind: Secret
metadata:
  creationTimestamp: null
  name: my-secret
```

Si vous exécuté cette commande par exemple vous verrai le password en clair. Au temps vous dire que les type Secret ne sont pas véritablement un moyen de sécuriser son mot de passe.

```
echo "TW9uUGFzc3Jvb3QwMQ==" | base64 --decode
MonPassroot01
```

ou bien ceci

```
$ kubectl -n mynamespace get secrets mysql-password \
    -o 'jsonpath={.data.passwd}' | base64 -d
SuperSecretPassword
```

```
apiVersion: v1
data:
  password: cm9vdHJvb3QwMQ==
kind: Secret
metadata:
  creationTimestamp: null
```

```
    name: mysql-pass
---
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1 #  for k8s versions before 1.9.0 use apps/v1beta2  and bef
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
      - image: wordpress:4.8-apache
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
```

```
            value: wordpress-mysql
          - name: WORDPRESS_DB_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql-pass
                key: password
          ports:
          - containerPort: 80
            name: wordpress
          volumeMounts:
          - name: wordpress-persistent-storage
            mountPath: /var/www/html
        volumes:
        - name: wordpress-persistent-storage
          persistentVolumeClaim:
            claimName: wp-pv-claim



apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
  clusterIP: None
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1 # for k8s versions before 1.9.0 use apps/v1beta2  and befc
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
```

```
    selector:
      matchLabels:
        app: wordpress
        tier: mysql
    strategy:
      type: Recreate
    template:
      metadata:
        labels:
          app: wordpress
          tier: mysql
      spec:
        containers:
        - image: mysql:5.6
          name: mysql
          env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql-pass
                key: password
          livenessProbe:
            tcpSocket:
              port: 3306
          ports:
          - containerPort: 3306
            name: mysql
          volumeMounts:
          - name: mysql-persistent-storage
            mountPath: /var/lib/mysql
        volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim
```

```
$ kubectl apply -f front-wordpress.yml
secret/mysql-pass created
service/wordpress created
persistentvolumeclaim/wp-pv-claim created
deployment.apps/wordpress created

$ kubectl apply -f back-wordpress.yml
service/wordpress-mysql created
persistentvolumeclaim/mysql-pv-claim created
deployment.apps/wordpress-mysql created

$ kubectl get deployment,pods,svc,secret,pvc,pv,sc
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wordpress         1/1     1            1           35s
deployment.apps/wordpress-mysql   1/1     1            1           2m37s
```

```
NAME                                  READY   STATUS    RESTARTS   AGE
pod/pvlab                             1/1     Running   0          57m
pod/wordpress-5994d99f46-fddjd        1/1     Running   0          35s
pod/wordpress-mysql-7fc5cb7ccc-fkckl  1/1     Running   0          2m37s

NAME                      TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)
service/kubernetes        ClusterIP      10.4.16.1     <none>        443/TCP
service/wordpress         LoadBalancer   10.4.30.247   34.71.81.82   80:32020/
service/wordpress-mysql   ClusterIP      None          <none>        3306/TCP

NAME                         TYPE                                  DATA   AGE
secret/default-token-r4p9c   kubernetes.io/service-account-token   3      12h
secret/mysql-pass            Opaque                                1      36s

NAME                                    STATUS   VOLUME
persistentvolumeclaim/mysql-pv-claim    Bound    pvc-4d312cf3-609b-45c4-8e24-8d
persistentvolumeclaim/pvc1              Bound    pvc-5bd5facd-2532-4789-bc25-f3
persistentvolumeclaim/wp-pv-claim       Bound    pvc-26903e6a-48aa-438f-8f86-8d

NAME                                                        CAPACITY   ACCESS
persistentvolume/pvc-26903e6a-48aa-438f-8f86-8de264c38e67   20Gi       RWO
persistentvolume/pvc-4d312cf3-609b-45c4-8e24-8de879dc2da3   20Gi       RWO
persistentvolume/pvc-5bd5facd-2532-4789-bc25-f37337b004ec   3Gi        RWO

NAME                                             PROVISIONER             RECLA
storageclass.storage.k8s.io/premium-rwo          pd.csi.storage.gke.io   Delet
storageclass.storage.k8s.io/standard (default)   kubernetes.io/gce-pd    Delet
storageclass.storage.k8s.io/standard-rwo         pd.csi.storage.gke.io   Delet
```

On accède avec l'IP public du loadBalancer 34.71.81.82 => Ok succès de connexion

Autres notions variées

- Créer un déployement du nom de `custom-web` de 3 replicas. Le Pod déployé doit avoir de deux conteneurs partageant le même espace de stockage temporaire `/usr/share/nginx/html/`
    - les deux conteneurs doivent monté un volume de type `EmptyDir: {}` mappé sur `/usr/share/nginx/html/`
    - Le premier conteneur est Nginx
    - le second conteneur est busybox qui exécute un script boucle infini qui écrira dans le fichier `/usr/share/nginx/html/index.html` les informations clée suivante : le nom de l'hôte sur lequel le Pod est exécuté
    - Appliquez le déploiement
    - Créer un service de type `ClusterIp` qui expose ce déploiement et constaté qui que un curl depuis un conteneur de test affiche ce que le busybox écrit

- Modifier le déploiement pour configurer une antiafinité entre eux de sorte que deux pods ne se trouvent pas sur le même node
- Appliquez à nouveau le déploiement et constatez bien que l'antiaffinité est respecté
- Modifier le déploiement pour limiter les ressources RAM et CPU des conteneurs Nginx
- Appliquez à nouveau le déploiement et constatez bien que les limites sont respecté

▼ Correction

```
---
apiVersion: v1
kind: Service
metadata:
  name: custom-web
  labels:
    app: custom-web
spec:
  ports:
    - port: 80
  selector:
    app: custom-web
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: custom-web
  name: custom-web
spec:
  replicas: 4
  selector:
    matchLabels:
      app: custom-web
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: custom-web
    spec:
      containers:
      - command:
        - /bin/sh
        - -c
        - |-
          #!/bin/sh
```

```
              while true; do tr -dc A-Za-z0-9 </dev/urandom| head -c 13 >> /
          # echo Blah > /usr/share/nginx/html/index.html
          #echo '${NODE_NAME}' >> /usr/share/nginx/html/index.html
        image: busybox
        name: busybox
        env:
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        volumeMounts:
        - name: htmldir
          mountPath: /usr/share/nginx/html/
      - image: nginx
        name: nginx
        volumeMounts:
        - name: htmldir
          mountPath: /usr/share/nginx/html/
      volumes:
      - name: htmldir
        emptyDir: {}
```

```
$ kubectl apply -f custom-web.yml
service/custom-web created
deployment.apps/custom-web configured

$ kubectl logs custom-web-58f86dc44-8ghgq -c nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to per
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-def
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/cc
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/10/05 14:42:33 [notice] 1#1: using the "epoll" event method
2022/10/05 14:42:33 [notice] 1#1: nginx/1.23.1
2022/10/05 14:42:33 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-
2022/10/05 14:42:33 [notice] 1#1: OS: Linux 5.10.133+
2022/10/05 14:42:33 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/10/05 14:42:33 [notice] 1#1: start worker processes
2022/10/05 14:42:33 [notice] 1#1: start worker process 31
2022/10/05 14:42:33 [notice] 1#1: start worker process 32
10.128.0.13 - - [05/Oct/2022:14:45:46 +0000] "GET / HTTP/1.1" 200 2960 "-" "Mo
10.128.0.13 - - [05/Oct/2022:14:45:46 +0000] "GET /favicon.ico HTTP/1.1" 404 5
2022/10/05 14:45:46 [error] 31#31: *1 open() "/usr/share/nginx/html/favicon.ic
```

Comprendre un fichier YAML

Dans cette partie il sera question de voir ensemble comment lire un fichier YAML

https://raw.githubusercontent.com/elastic/beats/8.4/deploy/kubernetes/filebeat-kubernetes.yaml

Télécharger ce fichier YAML et expliquer le section par section

La correction se trouve ci-après

▼ Correction

```
# les --- que vous voyez représentent à chaque le début d'un Fichier YAML. Dor
# Une astuce pour bien lire ce genre de fichier est de chercher à voir quel ty
# une fois que vous avez répéré de quoi il s'agit je vous recommande de commer
# Parce que Kubernetes est un orchestrateur de conteneurs et l'objectif d'util
# Donc commencer par le DaemonSet nous donne déjà une idée de ce que ce manife
# Liser les commentaires dans les manifestes dans l'ordre suivant :
#      1. manifest de DaemonSet ==> Allons y
#      2. Manifest de ServiceAccount ==> Allons y
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: filebeat-config
  namespace: kube-system
  labels:
    k8s-app: filebeat
data:
  filebeat.yml: |-          # Lorsqu'un configMap débute de la sorte ==> alors
    filebeat.inputs:
    - type: container
      paths:
        - /var/log/containers/*.log
      processors:
        - add_kubernetes_metadata:
            host: ${NODE_NAME}
            matchers:
            - logs_path:
                logs_path: "/var/log/containers/"

    # To enable hints based autodiscover, remove `filebeat.inputs` configurati
    #filebeat.autodiscover:
    #  providers:
    #    - type: kubernetes
    #      node: ${NODE_NAME}
    #      hints.enabled: true
    #      hints.default_config:
    #        type: container
    #        paths:
    #          - /var/log/containers/*${data.kubernetes.container.id}.log

    processors:
```

```
      - add_cloud_metadata:
      - add_host_metadata:

    cloud.id: ${ELASTIC_CLOUD_ID}
    cloud.auth: ${ELASTIC_CLOUD_AUTH}

    output.elasticsearch:
      hosts: ['${ELASTICSEARCH_HOST:elasticsearch}:${ELASTICSEARCH_PORT:9200}'
      username: ${ELASTICSEARCH_USERNAME}
      password: ${ELASTICSEARCH_PASSWORD}
---
apiVersion: apps/v1
kind: DaemonSet # Il s'agit d'un DaemonSet
metadata:
  name: filebeat
  namespace: kube-system # le pods sera déployer dans le namespace kube-system
  labels:
    k8s-app: filebeat   # Ce DaemonSet portera ce label filebeat
spec:
  selector:
    matchLabels:
      k8s-app: filebeat # Le Selector du DaemonSet est les Template qui porter
  template:         # Début du template
    metadata:
      labels:
        k8s-app: filebeat   # Le label auquel fait référence DaemonSet dans le
    spec:
      serviceAccountName: filebeat      # Le service account filebeat est le c
      terminationGracePeriodSeconds: 30     # L'application à l'intérieur du P
      hostNetwork: true     # Ce pod va s'exécuter dans le réseau hôte du node
      dnsPolicy: ClusterFirstWithHostNet # Ceci doit être explicitement spécif
      containers:
      - name: filebeat  # le nom de notre conteneur
        image: docker.elastic.co/beats/filebeat:8.4.3   # l'image qui sera tél
        args: [
          "-c", "/etc/filebeat.yml",
          "-e",
        ]
        env:                                    # On charge dans le Pod les
        - name: ELASTICSEARCH_HOST
          value: elasticsearch
        - name: ELASTICSEARCH_PORT
          value: "9200"
        - name: ELASTICSEARCH_USERNAME
          value: elastic
        - name: ELASTICSEARCH_PASSWORD
          value: changeme
        - name: ELASTIC_CLOUD_ID
          value:
        - name: ELASTIC_CLOUD_AUTH
          value:
        - name: NODE_NAME
          valueFrom:
```

```
            fieldRef:
              fieldPath: spec.nodeName
        securityContext:        # Le Pod s'exécutera avec les droit Root
          runAsUser: 0
          # If using Red Hat OpenShift uncomment this:
          #privileged: true
        resources:      # Le Pod s'exécutera avec une limitation de resources
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 100Mi
        volumeMounts:           # Des volumes sont déclarés
        - name: config          # Ce volument n'est pas un volume de donnée qu
          mountPath: /etc/filebeat.yml
          readOnly: true
          subPath: filebeat.yml
        - name: data            # Volume de données monté
          mountPath: /usr/share/filebeat/data
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
        - name: varlog
          mountPath: /var/log
          readOnly: true
      volumes:                # Les volumes montés sont déclarés ici.
      - name: config          # Ce volumes est particulier. En fait c'est un
        configMap:
          defaultMode: 0640
          name: filebeat-config
      - name: varlibdockercontainers    # On monte un Directory du Node /var/l
        hostPath:
          path: /var/lib/docker/containers
      - name: varlog                    # Pareil également On monte un Director
        hostPath:
          path: /var/log
      # data folder stores a registry of read status for all files, so we don'
      - name: data
        hostPath:                     # Le Pod persistera certaines données da
          # When filebeat runs as non-root user, this directory needs to be wr
          path: /var/lib/filebeat-data
          type: DirectoryOrCreate
---
apiVersion: rbac.authorization.k8s.io/v1  # Le ClusterRoles plus bas "filebeat
kind: ClusterRoleBinding
metadata:
  name: filebeat
subjects:
- kind: ServiceAccount
  name: filebeat
  namespace: kube-system
roleRef:
  kind: ClusterRole
```

```
    name: filebeat
    apiGroup: rbac.authorization.k8s.io
  ---
  apiVersion: rbac.authorization.k8s.io/v1   # Le roles plus bas "filebeat" sera
  kind: RoleBinding
  metadata:
    name: filebeat
    namespace: kube-system
  subjects:
    - kind: ServiceAccount
      name: filebeat
      namespace: kube-system
  roleRef:
    kind: Role
    name: filebeat
    apiGroup: rbac.authorization.k8s.io
  ---
  apiVersion: rbac.authorization.k8s.io/v1   # Le roles plus bas "filebeat-kubea
  kind: RoleBinding
  metadata:
    name: filebeat-kubeadm-config
    namespace: kube-system
  subjects:
    - kind: ServiceAccount
      name: filebeat
      namespace: kube-system
  roleRef:
    kind: Role
    name: filebeat-kubeadm-config
    apiGroup: rbac.authorization.k8s.io
  ---
  apiVersion: rbac.authorization.k8s.io/v1    # Definition d'un ClusterRole. La
  kind: ClusterRole
  metadata:
    name: filebeat
    labels:
      k8s-app: filebeat
  rules:
  - apiGroups: [""] # "" indicates the core API group
    resources:
    - namespaces
    - pods
    - nodes
    verbs:
    - get
    - watch
    - list
  - apiGroups: ["apps"]
    resources:
      - replicasets
    verbs: ["get", "list", "watch"]
  - apiGroups: ["batch"]
    resources:
```

```
    - jobs
  verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1        # Voir L'explication du role s
kind: Role
metadata:
  name: filebeat
  # should be the namespace where filebeat is running
  namespace: kube-system
  labels:
    k8s-app: filebeat
rules:
  - apiGroups:
      - coordination.k8s.io
    resources:
      - leases
    verbs: ["get", "create", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1        # La définition d'un Role. qui
kind: Role
metadata:
  name: filebeat-kubeadm-config
  namespace: kube-system
  labels:
    k8s-app: filebeat
rules:                                      # La règle de ce Role c'est qu'il
  - apiGroups: [""]
    resources:
      - configmaps
    resourceNames:
      - kubeadm-config
    verbs: ["get"]
---
apiVersion: v1
kind: ServiceAccount                # Aussi simple que ça, on crée un compte de
metadata:
  name: filebeat
  namespace: kube-system
  labels:
    k8s-app: filebeat
---
```

Déployer une application microservice

L'application ecommerce de

PV, PVC, SC Lab

▼ Correction

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  capacity:
    storage: 10Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName:
  local:
    path: /opt
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - k8s-worker-1
          - k8s-worker-2
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: pvlab
spec:
  containers:
    - name: pvlab-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: wew-storage
  volumes:
    - name: wew-storage
```

```
      persistentVolumeClaim:
        claimName: pvc1
```

```
vagrant@k8s-master-1:~$ kubectl apply -f pv1.yml
persistentvolume/pv1 created
vagrant@k8s-master-1:~$ kubectl get pv
NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
els-pv-volume   10Gi       RWO            Retain           Released    default
pv1             10Gi       RWO            Delete           Available
vagrant@k8s-master-1:~$ vim pvc1.yml
vagrant@k8s-master-1:~$ kubectl apply -f pvc1.yml
persistentvolumeclaim/pvc1 created
vagrant@k8s-master-1:~$ kubectl get pv,pvc
NAME                              CAPACITY   ACCESS MODES   RECLAIM POLICY   ST
persistentvolume/els-pv-volume    10Gi       RWO            Retain           Re
persistentvolume/pv1              10Gi       RWO            Delete           Bo

NAME                          STATUS   VOLUME   CAPACITY   ACCESS MODES   STORA
persistentvolumeclaim/pvc1    Bound    pv1      10Gi       RWO
vagrant@k8s-master-1:~$
```

```
vagrant@k8s-master-1:~$ kubectl get pv
NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM
els-pv-volume   10Gi       RWO            Retain           Released   default/
pv1             10Gi       RWO            Delete           Bound      default/
vagrant@k8s-master-1:~$ kubectl get pvc
NAME   STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvc1   Bound    pv1      10Gi       RWO                           47s
vagrant@k8s-master-1:~$ vim pvlab-pod.ymal
vagrant@k8s-master-1:~$ kubectl apply -f pvlab-pod.ymal
pod/pvlab created
vagrant@k8s-master-1:~$ kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
elastic1                   1/1     Running   0          4h39m
frontend-86d67d9884-5f6zr  1/1     Running   0          117m
frontend-86d67d9884-dftl2  1/1     Running   0          117m
nginx-pod                  1/1     Running   0          4h38m
pvlab                      1/1     Running   0          45s
robots-68548b4489-7hrrn    1/1     Running   0          52m
robots-68548b4489-tqvjw    1/1     Running   0          52m
web1                       1/1     Running   0          4h51m
webserver                  1/1     Running   0          135m
vagrant@k8s-master-1:~$ kubectl exec -it pvlab -- sh
# ls /usr/share/nginx/html
VBoxGuestAdditions-6.1.34  cni  containerd
# touch file-from-pod-pvlab
# touch /usr/share/nginx/html/file-from-pod-pvlab
#
```

```
## sur le node on voit bien que le ficher aparait
vagrant@k8s-worker-2:~$ ls /opt/
VBoxGuestAdditions-6.1.34  cni  containerd  file-from-pod-pvlab
vagrant@k8s-worker-2:~$
```

Storage Class

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
spec:
  storageClassName: premium-rwo
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: pvlab
spec:
  containers:
    - name: pvlab-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: wew-storage
  volumes:
    - name: wew-storage
      persistentVolumeClaim:
        claimName: pvc1
```

```
$ kubectl get sc premium-rwo -o yaml
```

Résultat de la commande :

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    components.gke.io/component-name: pdcsi
    components.gke.io/component-version: 0.11.8
    components.gke.io/layer: addon
  creationTimestamp: "2022-10-05T01:31:56Z"
  labels:
    addonmanager.kubernetes.io/mode: EnsureExists
    k8s-app: gcp-compute-persistent-disk-csi-driver
  name: premium-rwo
  resourceVersion: "608"
  uid: ddf2b6d2-ac71-4764-9658-509badd8a59b
parameters:
  type: pd-ssd
provisioner: pd.csi.storage.gke.io
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

Observons le fait que création de PVC et d'un Pod va engendrer la création du PV

```
$ kubectl apply -f pvc1.yml
# Nous avons créé le PVC

$ kubectl get sc,pv,pvc
NAME                                                 PROVISIONER             RECLA
storageclass.storage.k8s.io/premium-rwo              pd.csi.storage.gke.io   Delet
storageclass.storage.k8s.io/standard (default)       kubernetes.io/gce-pd    Delet
storageclass.storage.k8s.io/standard-rwo             pd.csi.storage.gke.io   Delet

NAME                           STATUS    VOLUME   CAPACITY   ACCESS MODES   STOR
persistentvolumeclaim/pvc1     Pending                                        prem

# On voit que le PVC est en mode pending parce qu'il attend d'être consommé pa

$ kubectl apply -f pod.yml

# On remarquera que quand le Pod devient Up il y'a des changements : création

$ kubectl get pods,sc,pv,pvc
NAME        READY   STATUS    RESTARTS   AGE
pod/pvlab   1/1     Running   0          25s
```

```
NAME                                                  PROVISIONER             RECLA
storageclass.storage.k8s.io/premium-rwo               pd.csi.storage.gke.io   Delet
storageclass.storage.k8s.io/standard (default)        kubernetes.io/gce-pd    Delet
storageclass.storage.k8s.io/standard-rwo              pd.csi.storage.gke.io   Delet

NAME                                                          CAPACITY   ACCESS
persistentvolume/pvc-5bd5facd-2532-4789-bc25-f37337b004ec     3Gi        RWO

NAME                           STATUS   VOLUME
persistentvolumeclaim/pvc1     Bound    pvc-5bd5facd-2532-4789-bc25-f37337b004ec
```