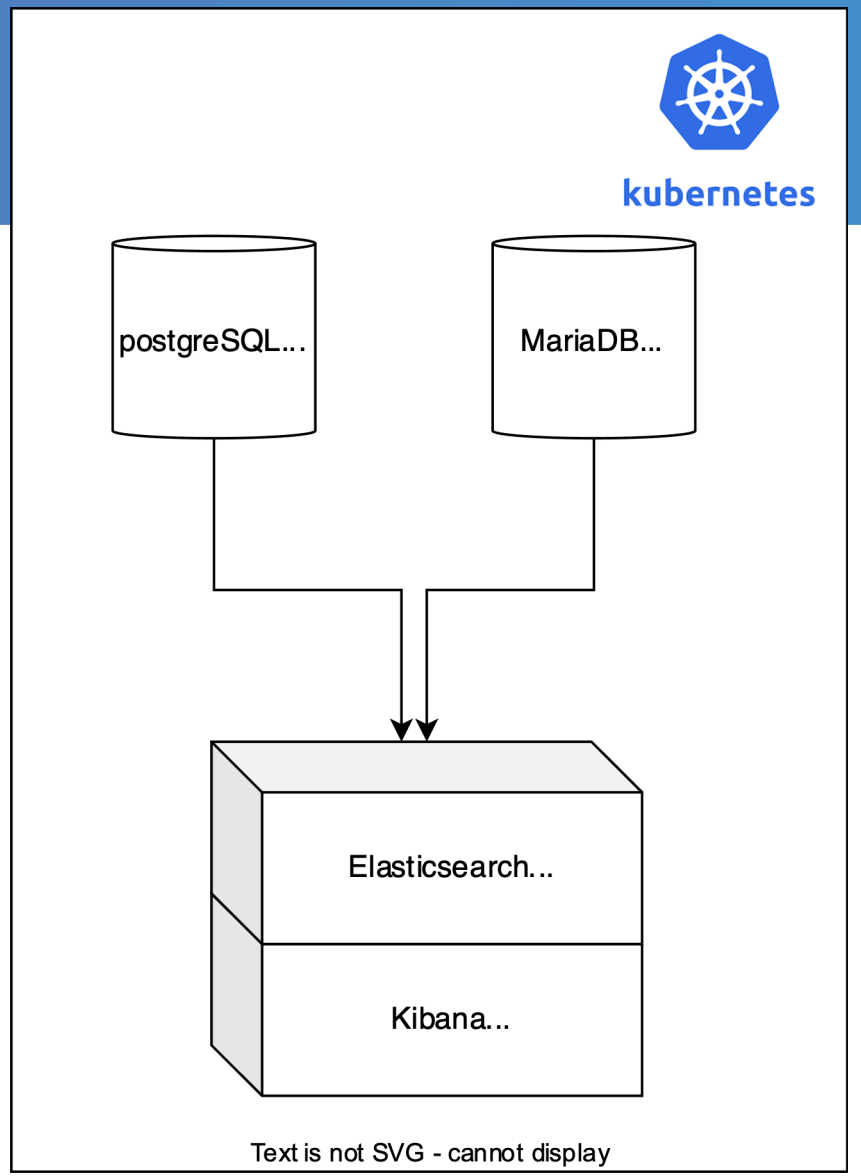
Exercice Kubernetes

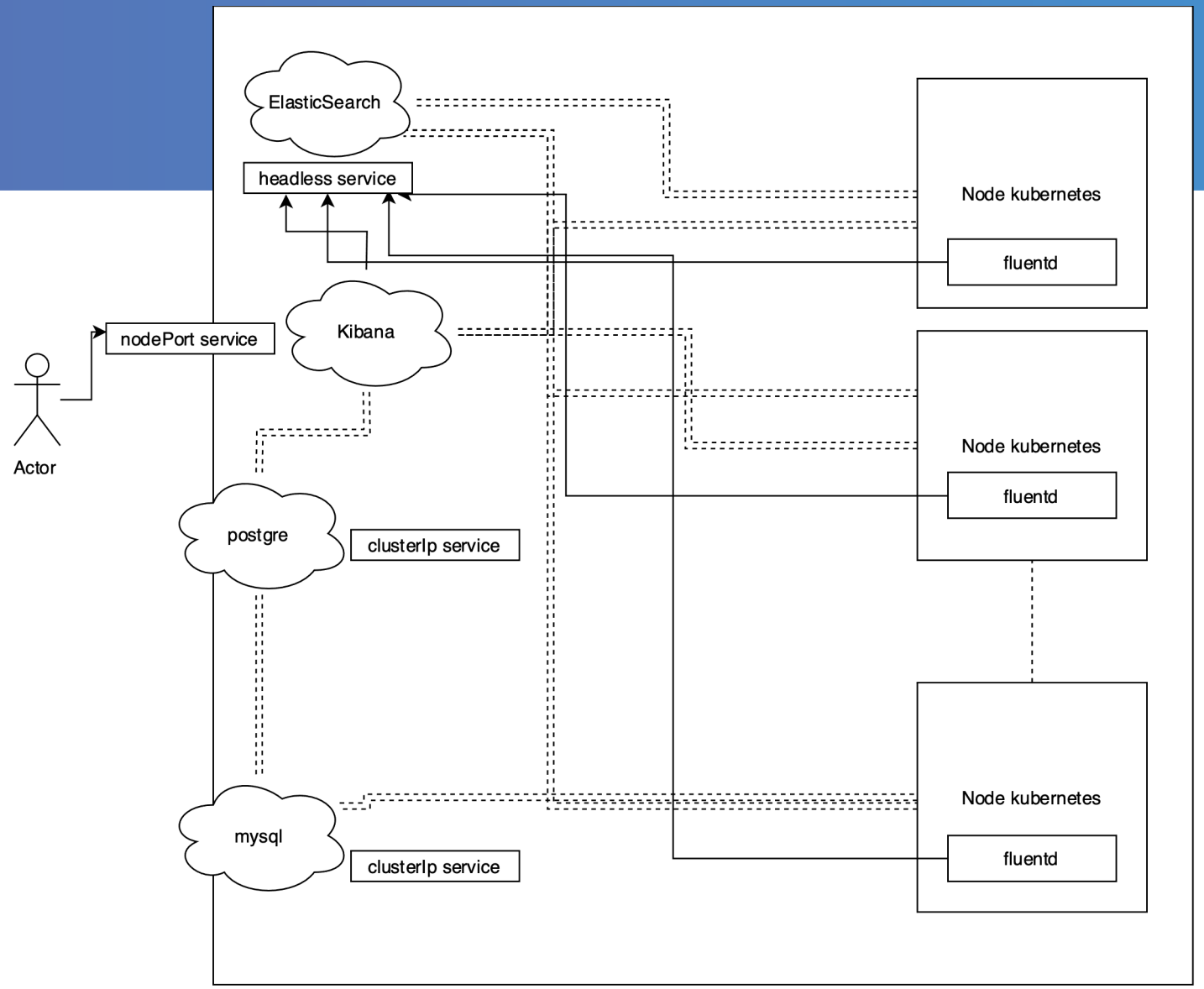
Ihab ABADI / UTOPIOS

Exercice kubernetes

- On souhaite mettre en place une architecture qui permet de récupérer les différents logs des pods d'un cluster Kubernetes et les envoyer vers un cluster elasticsearch.
- Les logs peuvent être analyser et visualiser à l'aide d'une instance kibana.
- Par exemple les logs de base de données mysql et postgres sur un cluster kubernetes.



Composition du cluster kubernetes



Etape 1 – Création du namespace

- Dans un premier temps et pour isoler nos applications dans notre cluster kubernetes, nous procéderons à la création d'une ressource de type namespace
- Ressource : create-namespace.yml

Etape 2 – Création d'un service pour elasticsearch

- Un cluster ElasticSearch possède son propre mécanisme d'auto-discovery, pour automatiser le mappage des pods (master, et node), on peut créer un service type headless.
- Dans le cas ou on utilise un ElasticSearch single-node, un service de type clusterIp suffit.
- Ressources : es_headless_service.yml

Etape 3 - Création d'un déploiement pour ElasticSearch (StateFulSet)

- Elastisearch a besoin d'une persistance stable même lors de la création de nouveaux pods par le schudle, pour répondre à ce besoin on utilisera un statefulSet au lieu d'un déploiement.
- Le fonctionnement est presque le même mais à l'inverse des déploiements stateFulSet permet de maintenir la même identité pour chaque pod.
- Pour ce dépoilement, nous utiliserons une image elasticsearch:7.16.1
- Nous procéderons également d'un changement permission avant le démarrage du conteneur, ainsi qu'au montage des volumes.
- Kubernetes démarre les conteneurs en Root, nous changerons à la fois l'utilisateur et le groupe en 1000:1000
- Nous procéderons également à l'augmentation de mmap ainsi qu'au nombre de fichiers de descriptions ouverts.
- Les volumes sont de de type storageClass standard.
- Ressource : es_statefulset.yml

Etape 4 – Service kibana

- Nous procéderons à la création d'un service de type nodeport pour accéder à notre application kibana de l'exéterieur du cluster.
- Ressource : kibana_service.yml
- Dans le cadre d'utilisation de minikube pour les tests, on expose le port nodePort

Etape 5 – Déploiement kibana

- Nous créerons un déploiement pour notre application kibana.
- Nous utiliserons une image docker.elastic.co/kibana/kibana:7.16.1
- Ressource: kibana_deployment.yml

Etape 6 – Application fluentd

- Fluentd est une application qui permet de collecter des données sur les différents nos nœuds de notre clutser et les envoyer vers une api tiers.
- Dans notre cas fluentd récupérera les différents logs des conteneurs pour les envoyer vers notre elasticsearch.
- Comme fluentd doit être sur chaque nœuds du cluster, nous utiliserons un daemonSet pour s'assurer de l'existance d'au moins une instance par nœud.
- Pour pouvoir accéder aux ressources de Kubernetes sur chaque nœuds et collecter nos logs, fluentd a besoin d'accéder à notre api rest Kubernetes.

Etape 6 – 1 – Service account

- Pour que fluentd puisse utiliser l'api rest kubernetes, nous créerons une ressource ServiceAccount.
- Ressource : fluentd_service_account.yml

Etape 6 – 2 Création des rôles

- Un service account est associé à un ensemble de roles (ClusterRole) pour limiter l'accès aux ressources du cluster.
- Nous autoriserons l'accès aux ressources pods et namespaces.
- Nous autoriserons les actions de lecture et de surveillance.
- Ressource: fluentd_cluster_role.yml

Etape 6 – 3 Association rôles et service account

- Pour associer notre serviceAccount avec notre clusterRole On utilise un clusterRoleBinding.
- Ressource: fluentd_cluster_role_binding.yml

Etape 6 – 4 – Création du daemonSet

- Nous créerons notre daemonSet associé à notre serviceAccount de l'étape 6 – 1.
- Grâce à la tolération, on peut spécifier si une instance sera également mise en place sur le master kubernetes.
- Nous utiliserons l'image: image: fluent/fluentd-kubernetes-daemonset:v1-debian-elasticsearch
- Fluentd récupèrera les logs sur les nœuds /var/logs, ainsi que les logs des conteneurs /var/lib/docker/containers
- Ressource: fluentd_daemon_set.yml

Etape 7 Création des ressources pour postgres

- Pour déployer notre application postgres sur notre cluster Kubernetes, nous aurons besoin de :
  - Un service de type ClusterIp pour faire communiquer notre base de données avec nos applications, « pour le test on utilisera un service de type nodePort ».
  - Un déploiement statefulset avec une image postgres.
  - Une persistance au niveau de notre cluster.
  - Nous utiliserons des volumes de types secrets pour partager les secrets d'accès.
- Ressources :
  - postgres-statefulset.yml
  - postgres-service.yml
  - postgres-secrets.yml

Etape 8 Création des ressources pour mysql

- Pour déployer notre application mysql sur notre cluster Kubernetes, nous aurons besoin de :
  - Un service de type ClusterIp pour faire communiquer notre base de données avec nos applications, « pour le test on utilisera un service de type nodePort ».
  - Un déploiement statefulset avec une image mysql.
  - Une persistance au niveau de notre cluster.
  - Nous utiliserons des volumes de types secrets pour partager les secrets d'accès.
- Ressources :
  - mysql-statefulset.yml
  - mysql-service.yml
  - mysql-secrets.yml