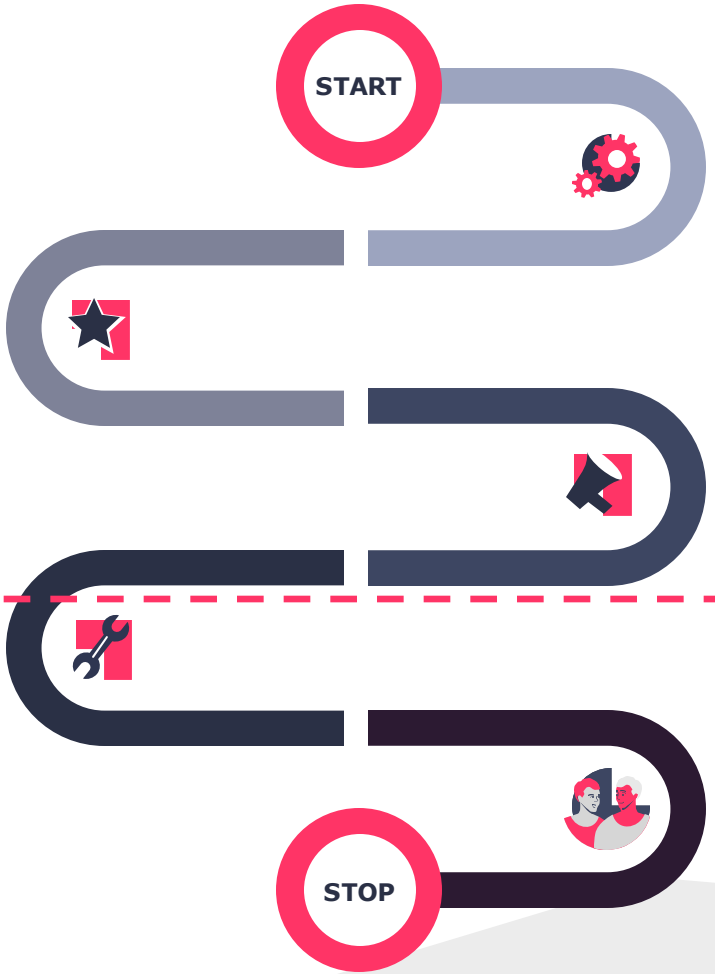
Développement et service cloud

DEV844

Améliorer l'efficacité des interactions entre une marque et ses clients

Module



Bases et notions
Découvrir les bases du cloud

Concepts
Découverte de conceptes de base d'archi cloud

Kubernetes
Découvertes et mise en place d'un cluster Kubernetes

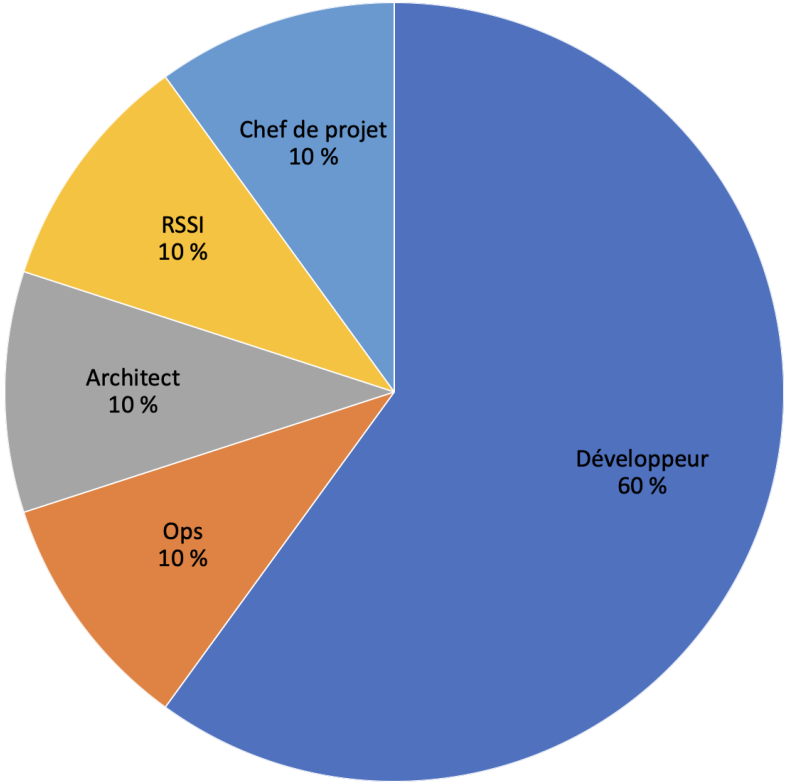
Cloud

Intégration continue
Rappels et mise en place sur de l'intégration continue

CI/CD

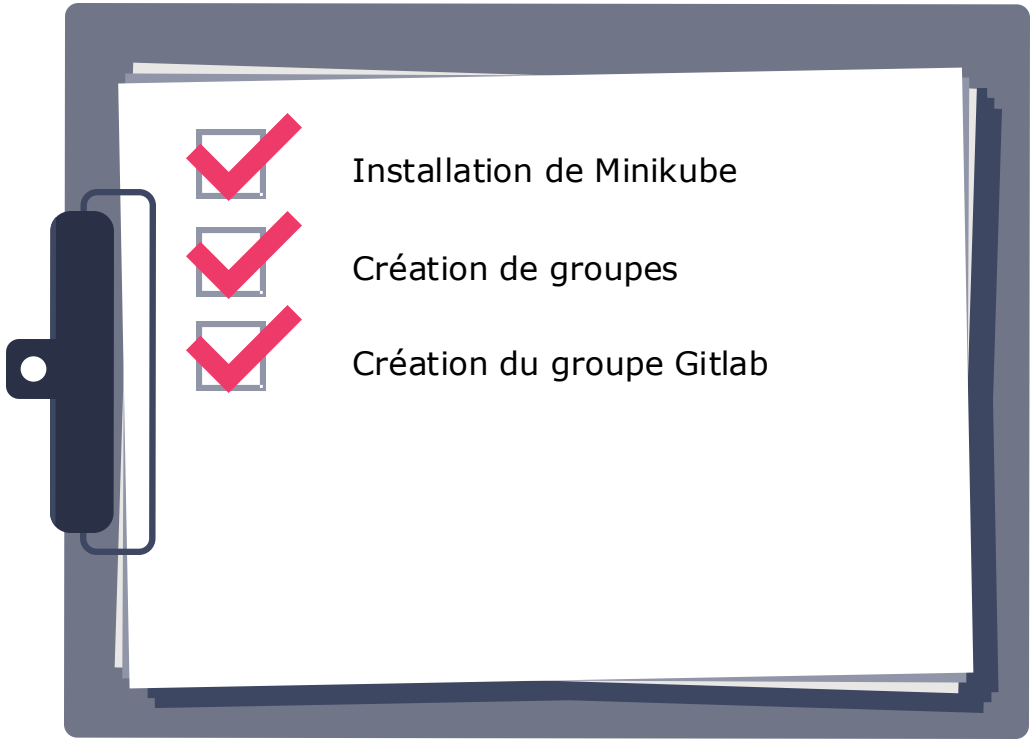
GitOps
Déploiement continue suivant des principes de GitOps

Votre rôle de full stack



Des compétences 360 font la différence entre un développeur et un ingénieur !

Avant de commencer



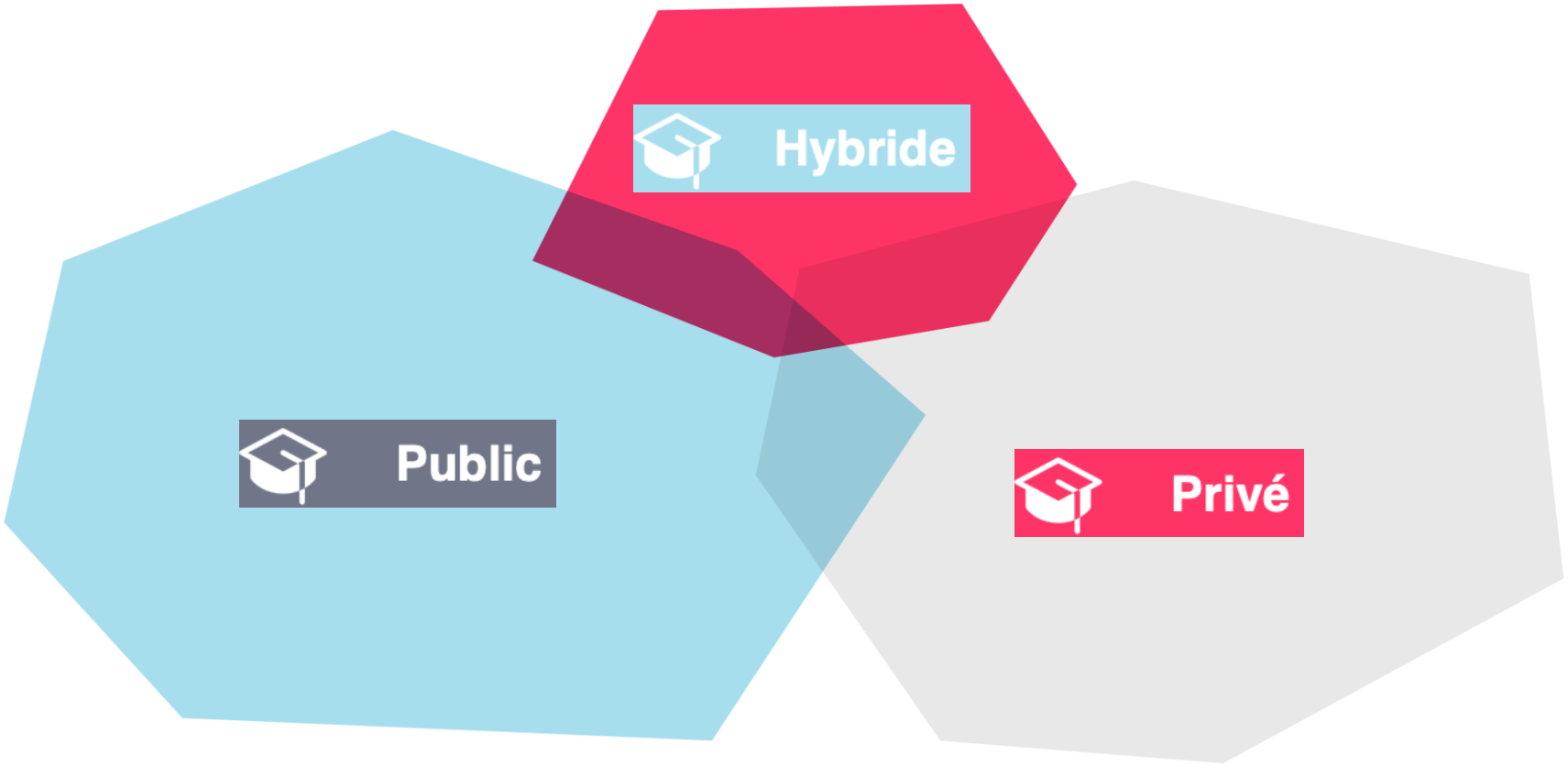
robi**nn**.meyssonnier@gmail.com

Création des groupes

Introduction au cloud computing

Lexique et notions

Les différents types de cloud



Cloud public

Pay as you go
On ne paie que ce que l'on consomme

Services managés
Fournis des services managés par la plateforme (+ chère mais présente des avantages / continents)

Accessible
Une infra en quelques cliques

Contrats de services
Des SLA souvent proches de la dispo à 100%

Cloud privé

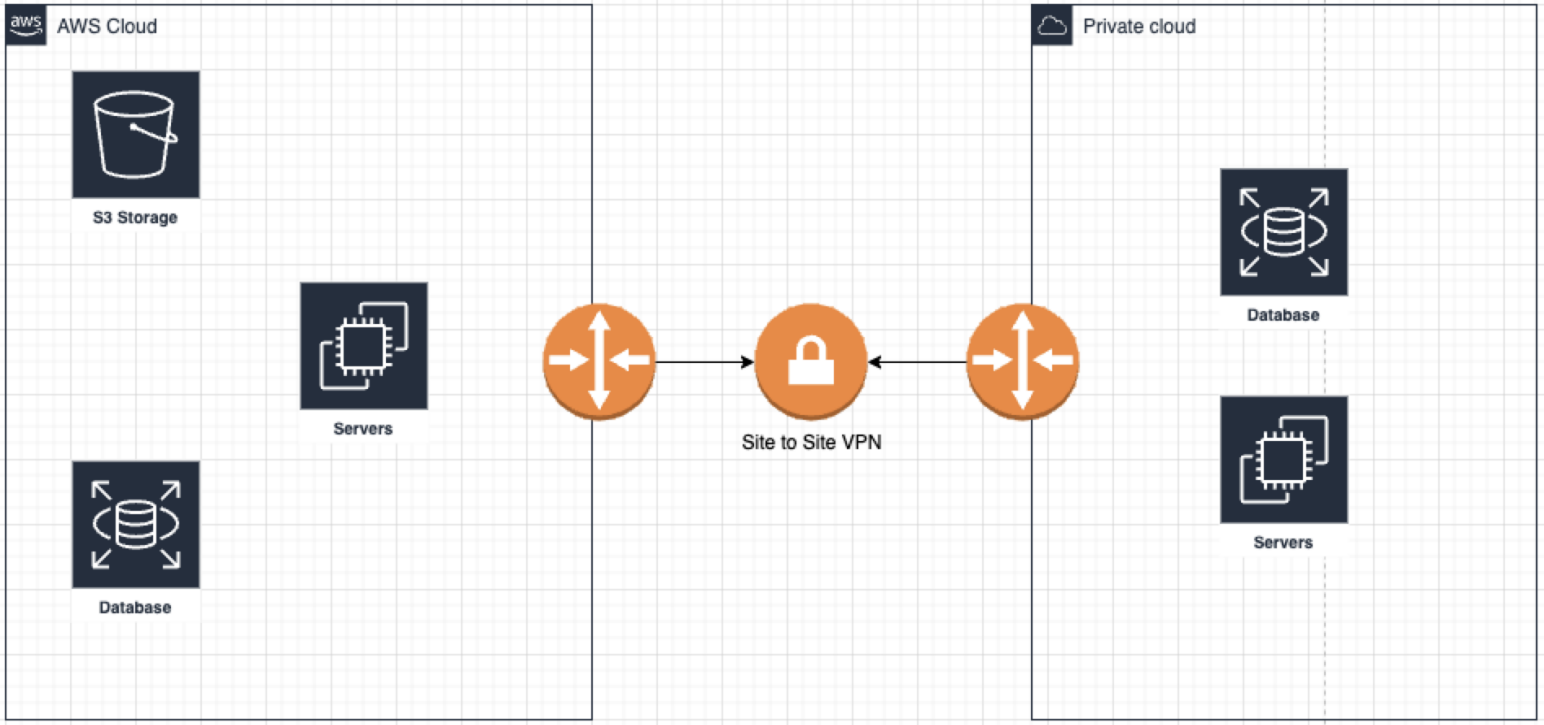
Privé dédié

L'ensemble de l'infrastructure est hébergé et porté en interne.

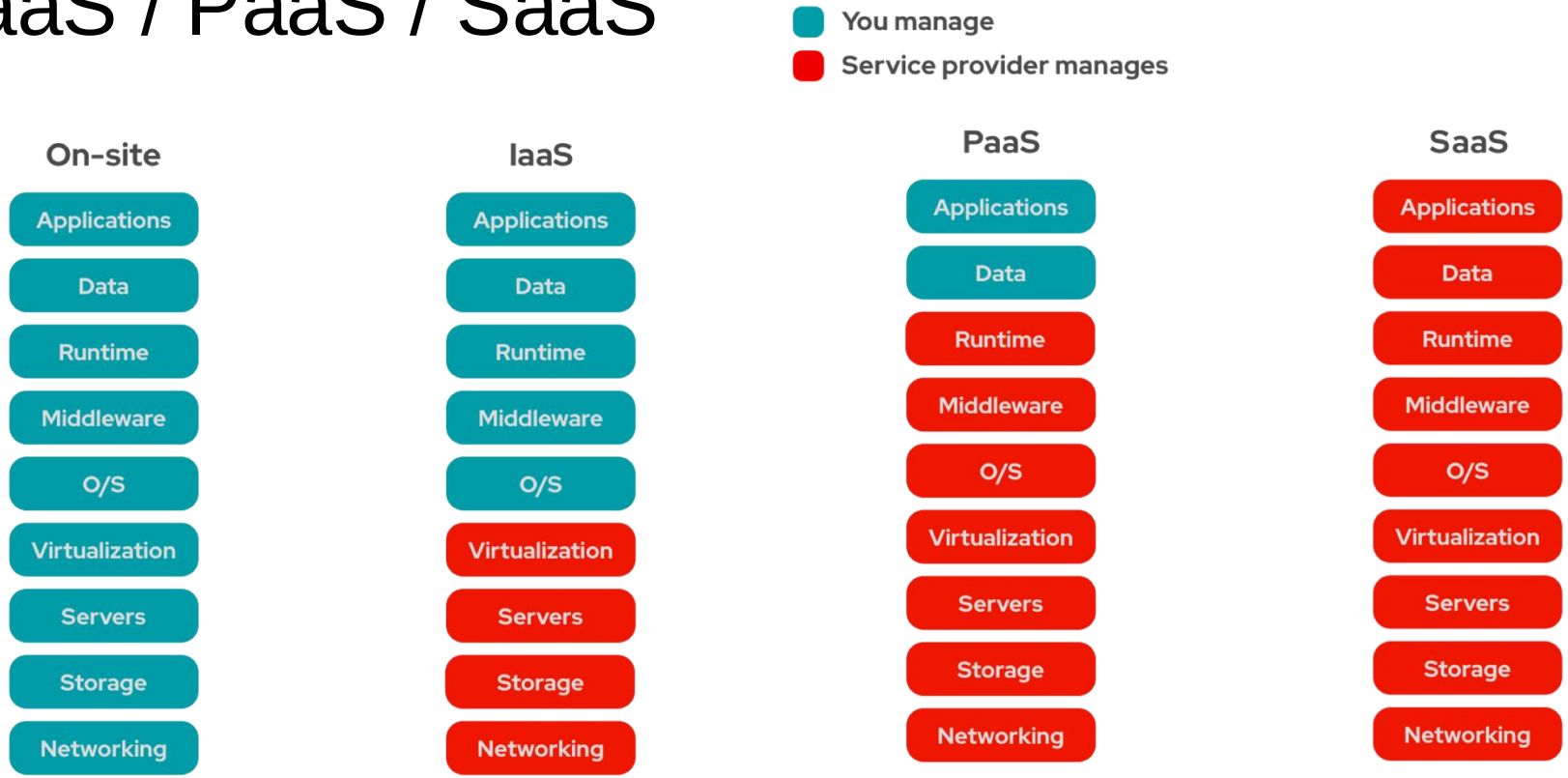
Privé géré

Un prestataire peut fournir la gestion du réseau, des hyperviseurs, bâtiments…

Cloud hybride



IaaS / PaaS / SaaS



Concepts et notions

Concepts de base d’architecture

Concepts de base

Résilience

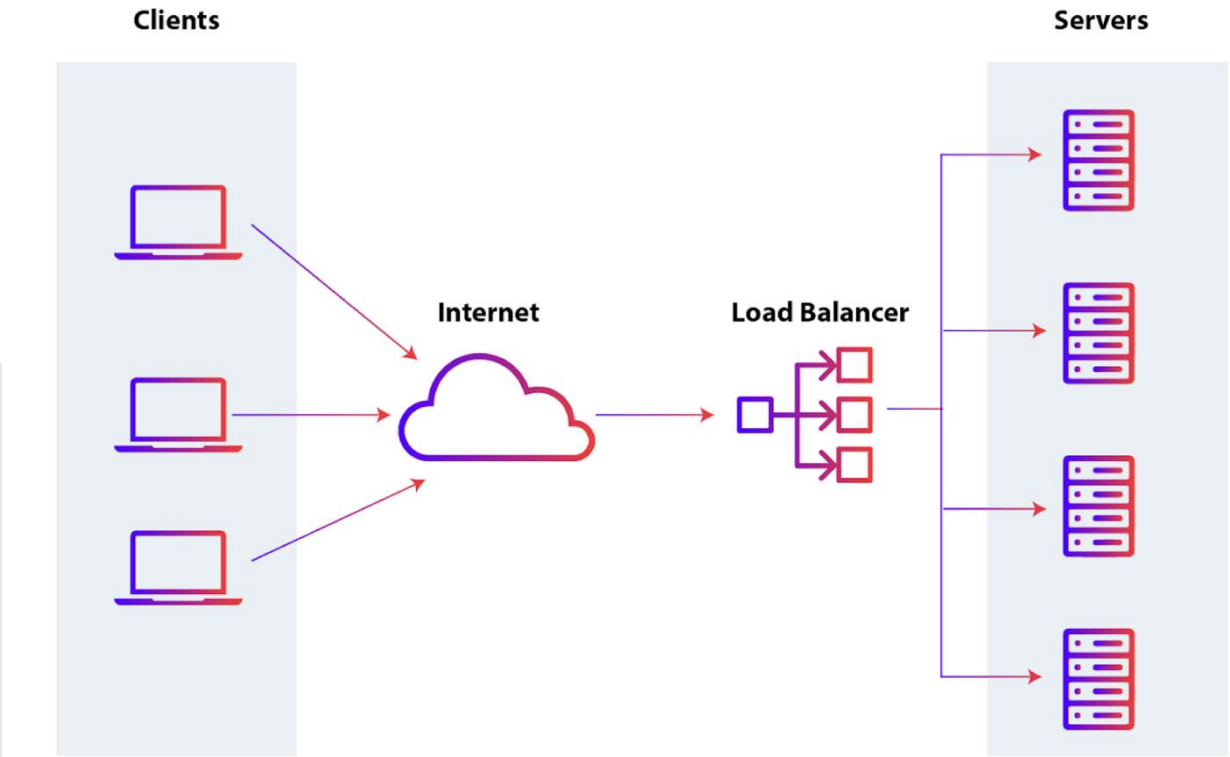
Scaling

Industrialisation

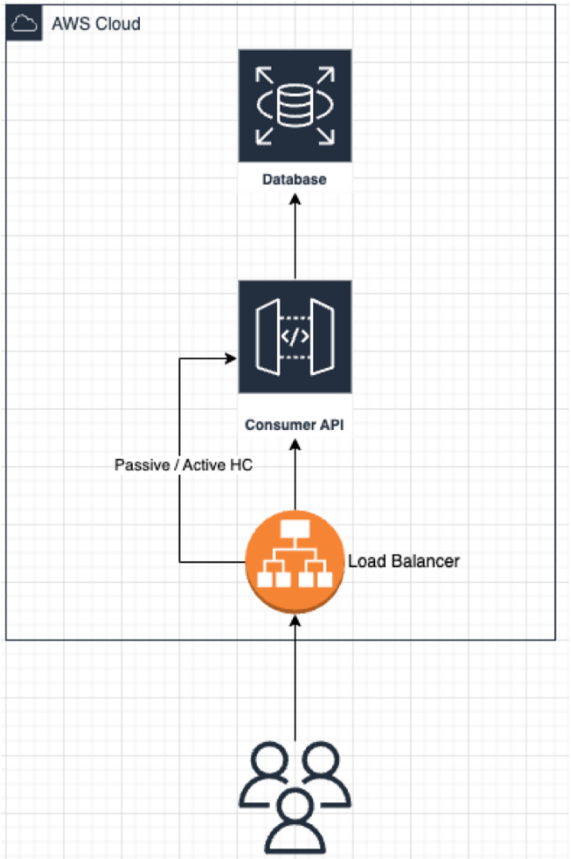
Services managés

Security by design

Load balancing



Résilience - Health check



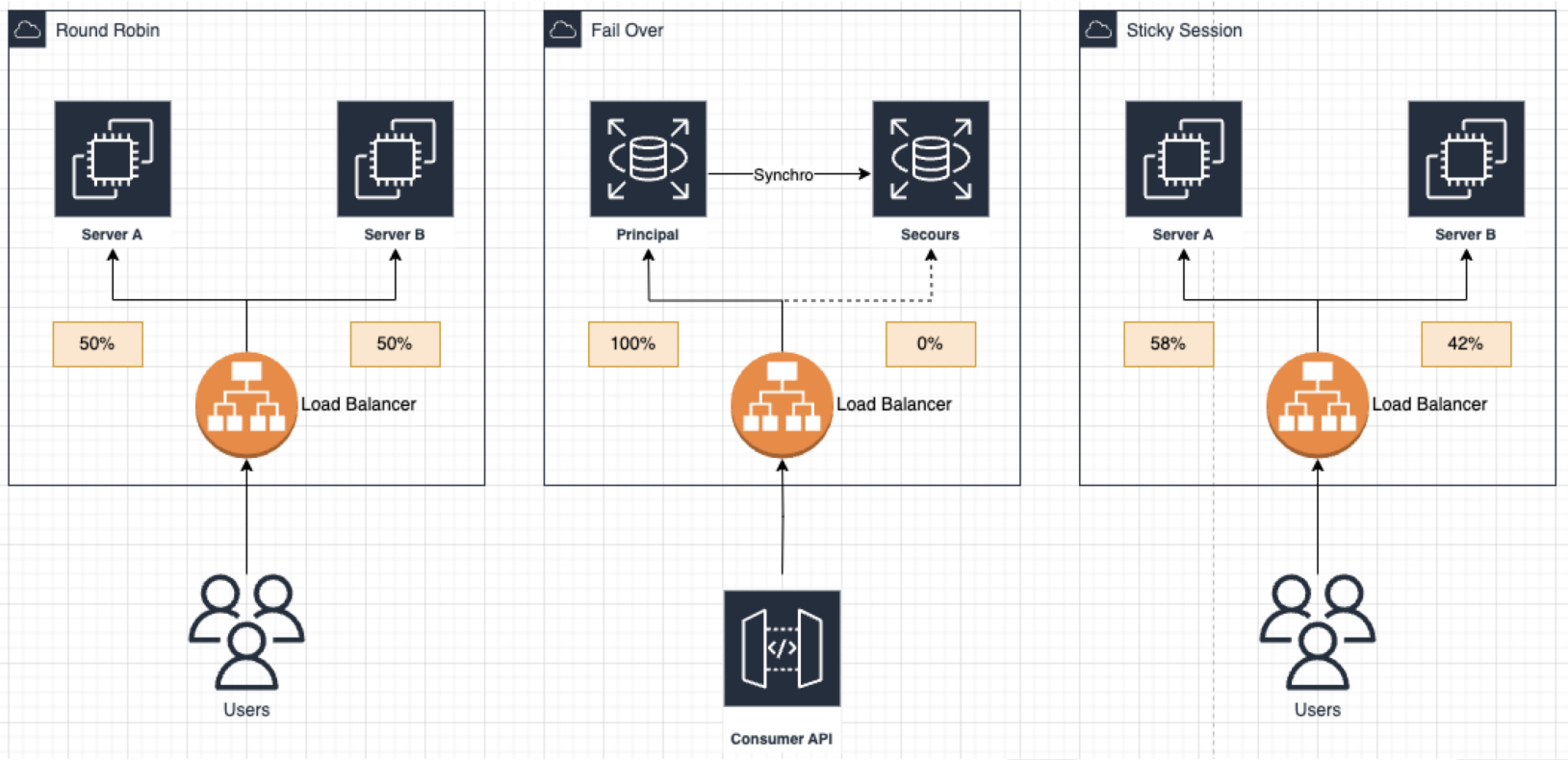
Health check actif
On appel un intervalle régulier une route donné, si le retour HTTP est OK tout va bien sinon on arrête de servir l'upstream

Health check passif
Analyse le traffic entre le load balancer et l'upstream, si trop d'erreurs se produisent, on arrête de servir l'upstream

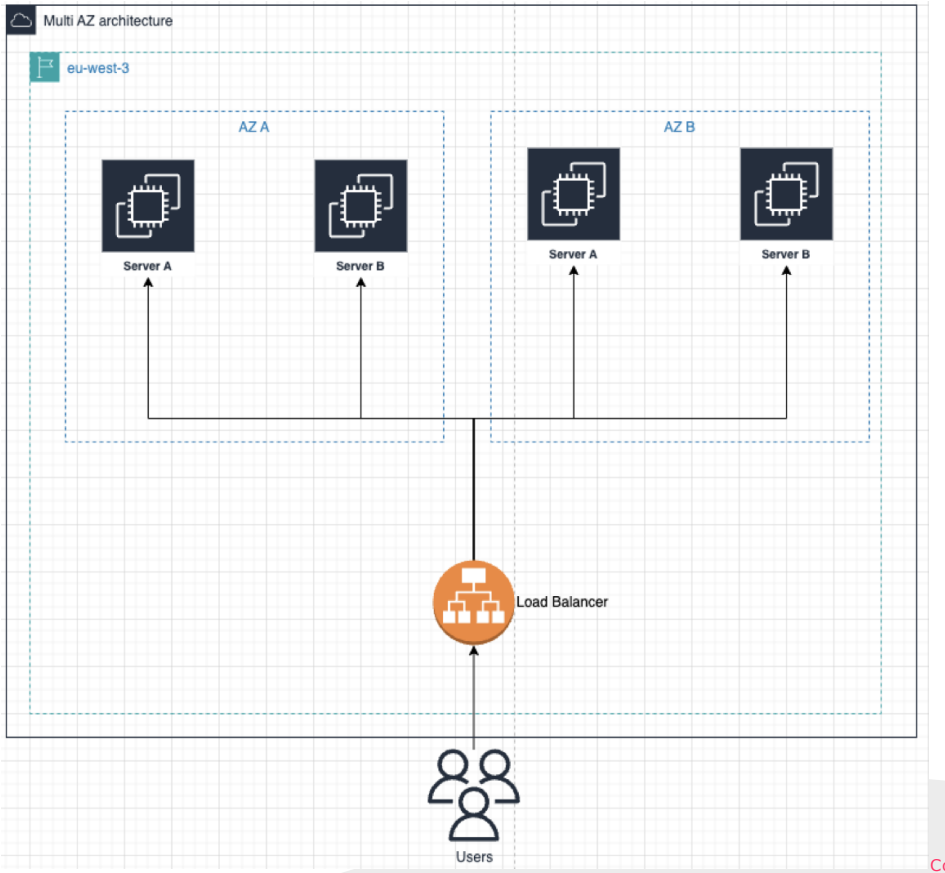
Rôle d'un health check actif
Un health check actif est généralement une route d'API qui, lorsqu'elle est appelée, remonte l'état du service.
Et dans le body de la réponse nous avons le détail du statut des composants essentiel de l'application

Objectif du healthcheck
Dans une architecture résiliante, nous avons obligatoirement plusieurs serveurs / applications redondées.
Le HC permet donc au load balancer de tester chaque élément et de ne plus servir ceux en défaut pour assurer la qualité de la réponse.

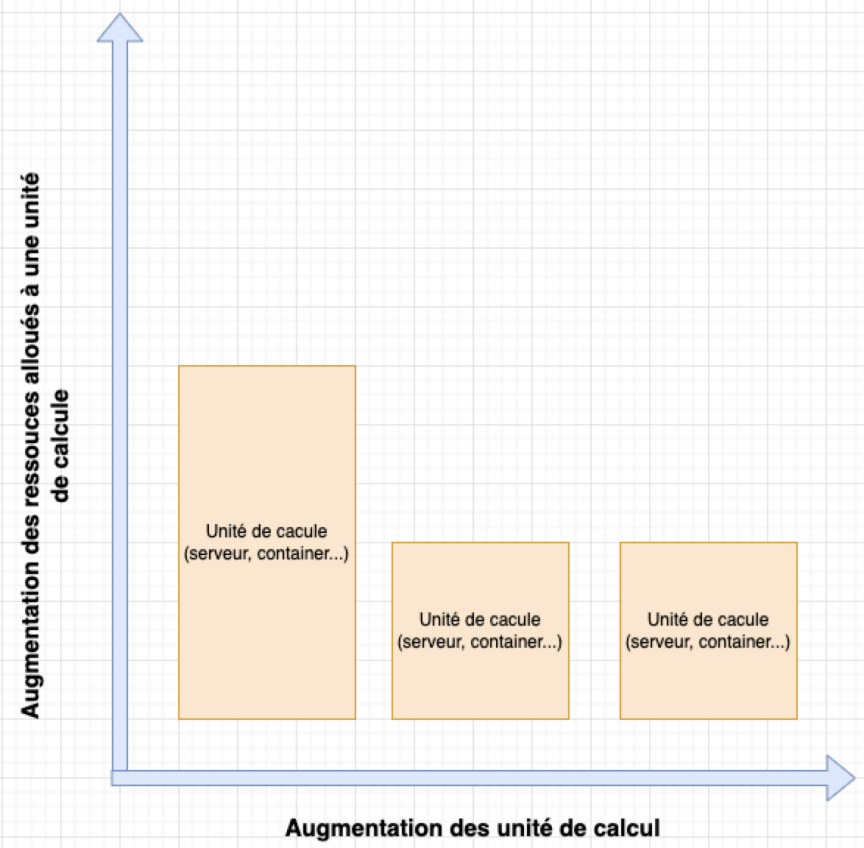
Résilience - Mise en place



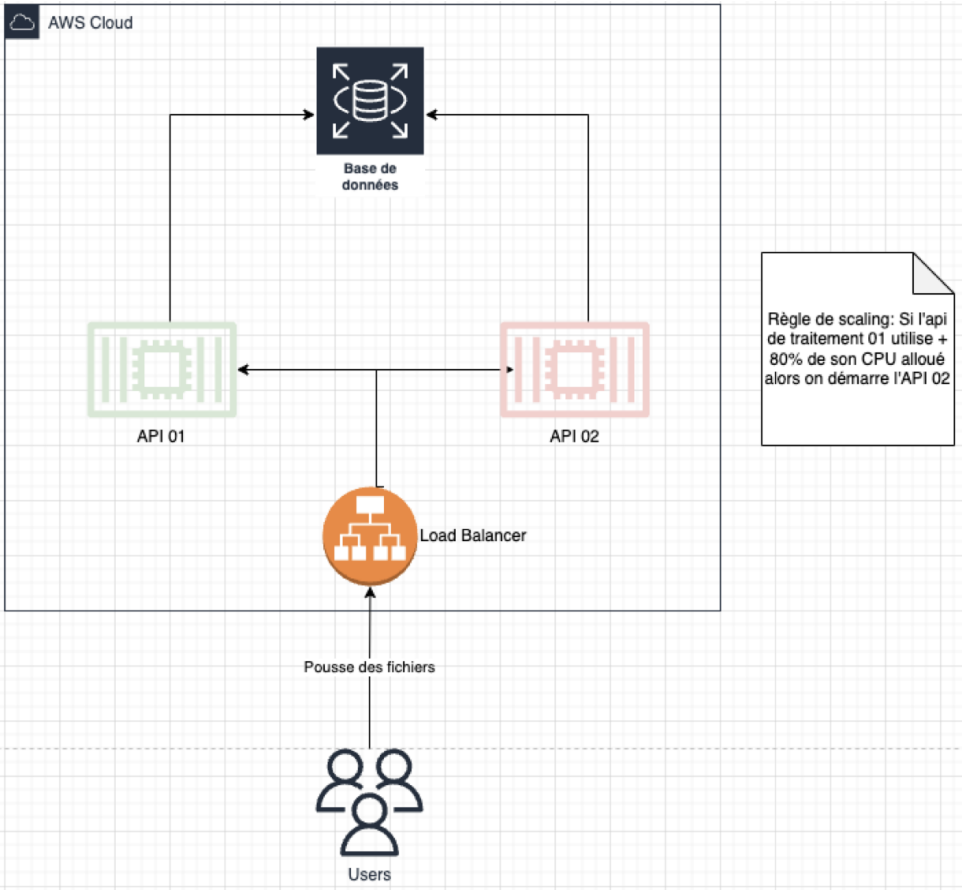
Résilience - Multi AZ (Zone de disponibilité)



Scaling - Principe de base



Scaling – Mise en place

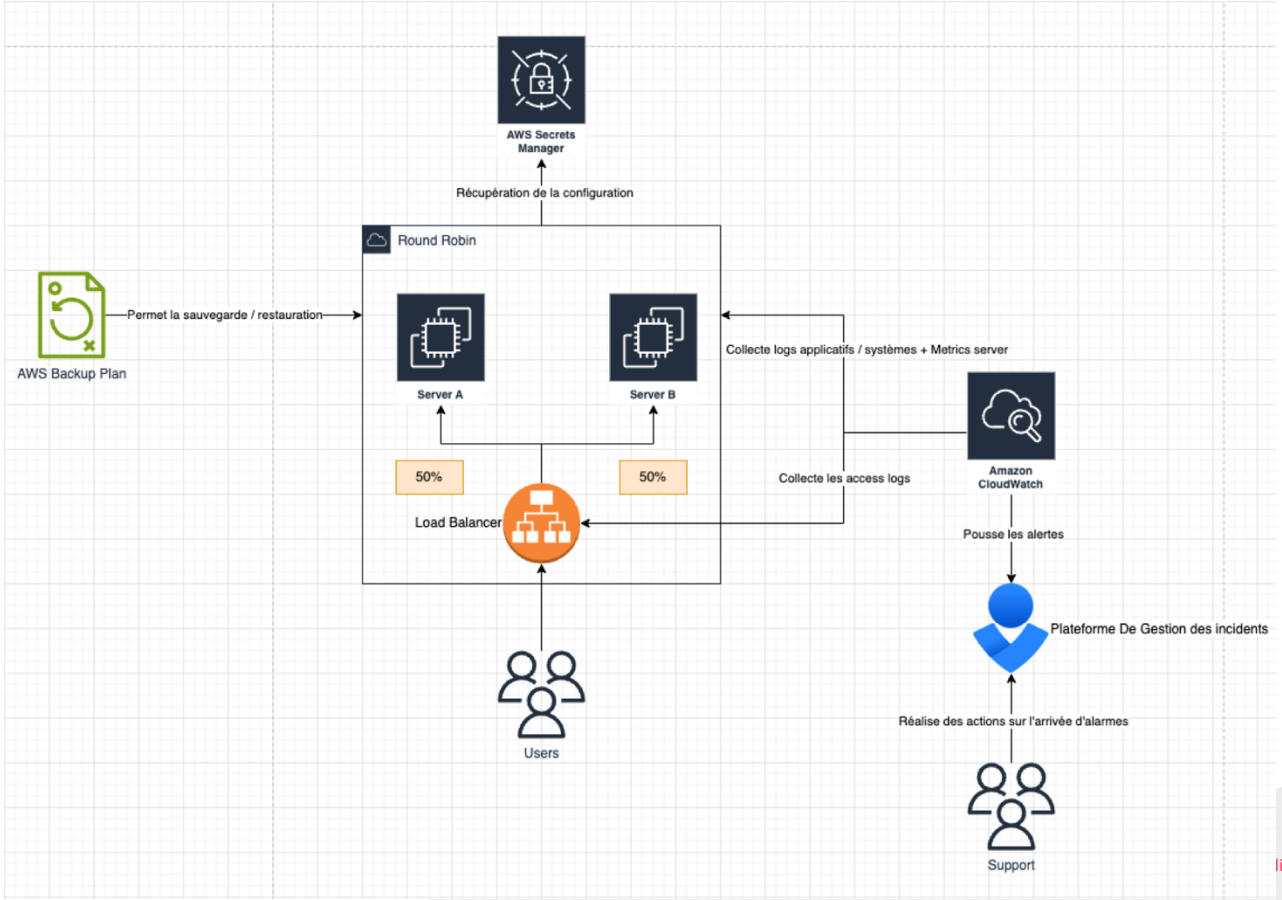


Dans notre exemple nous avons deux unités de calculs.
L'API 01 est toujours allumé, si l'utilisation de son CPU > 80%, on démarre l'unité 02 pour maintenir une qualité de traitement optimale des données.

Techniquement le scaling fonctionne grâce à un opérateur. Peu importe ce qu'il faut scale (containers, server…). Nous avons toujours besoin d'un opérateur.

L'important est la stratégie de Scaling.

Industrialisation



RBAC

Roles

Ops

Dev

Associe des permissions à des rôles

Permissions

Qui donnent à accès ou non a des ressources

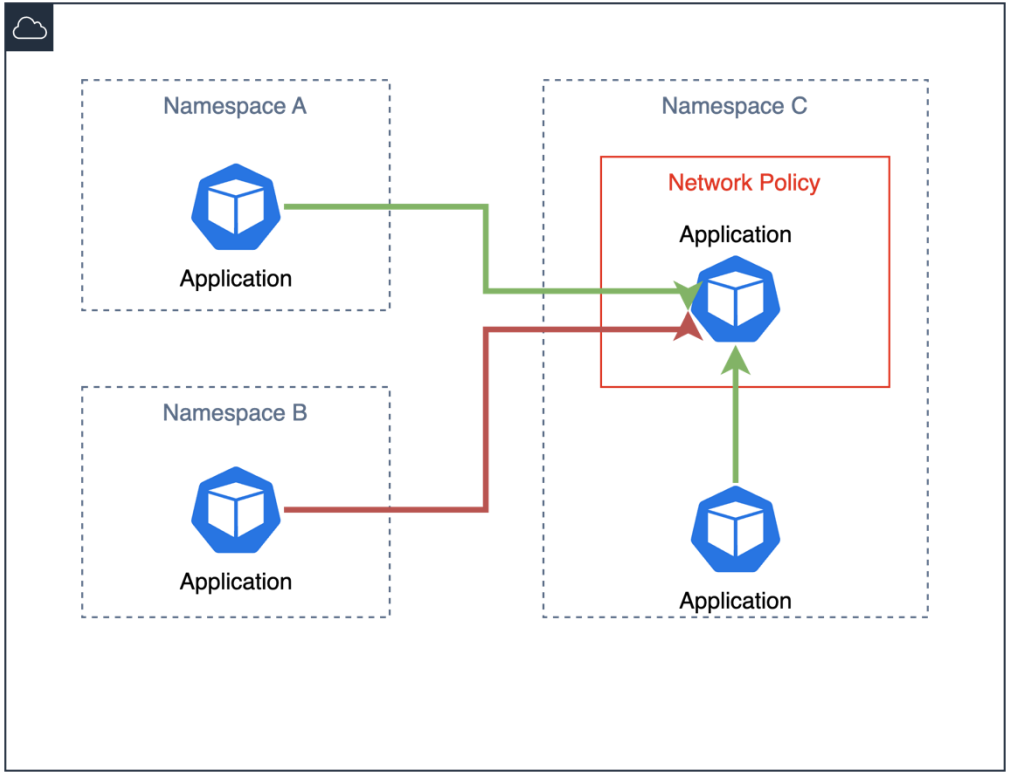
Accès

Se connecter à une bdd

Modifier une configuration

Relancer une application

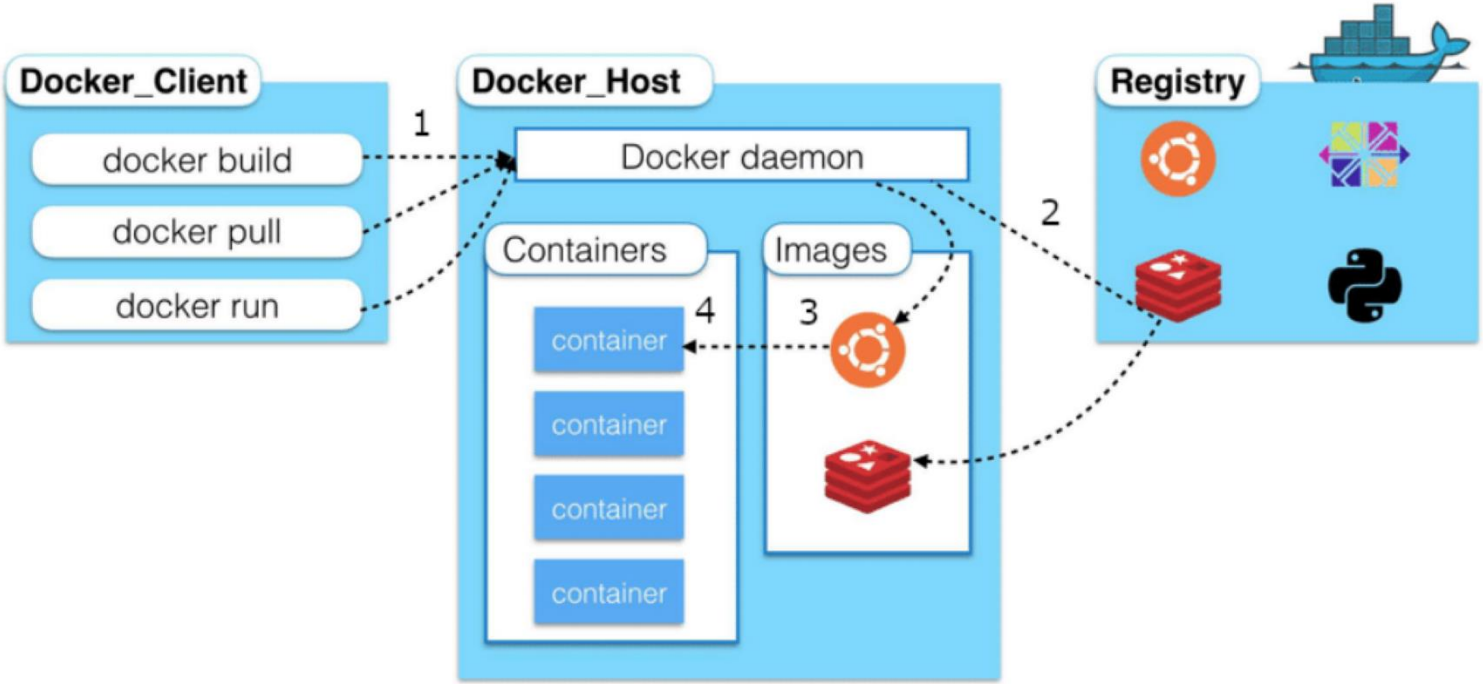
Politiques réseau



Kubernetes

Introduction à Kubernetes

Rappel : Docker

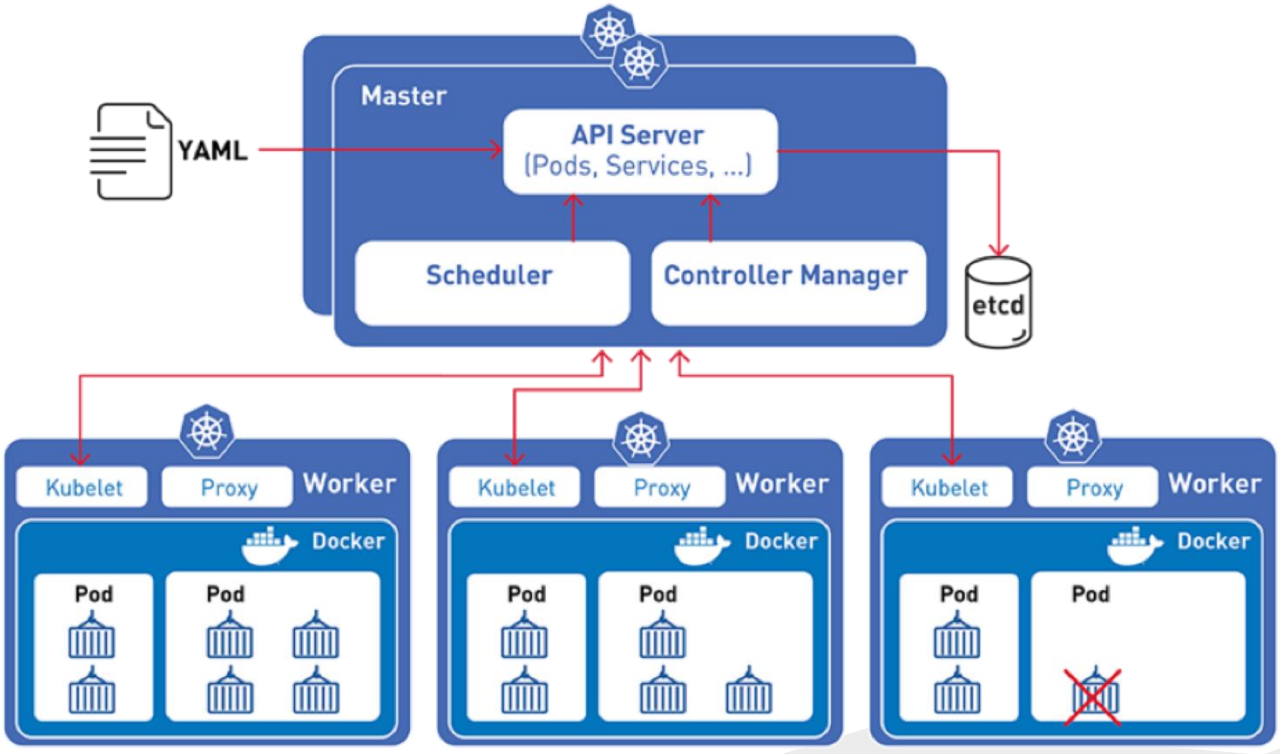


Gérer des conteneurs à grande échelle

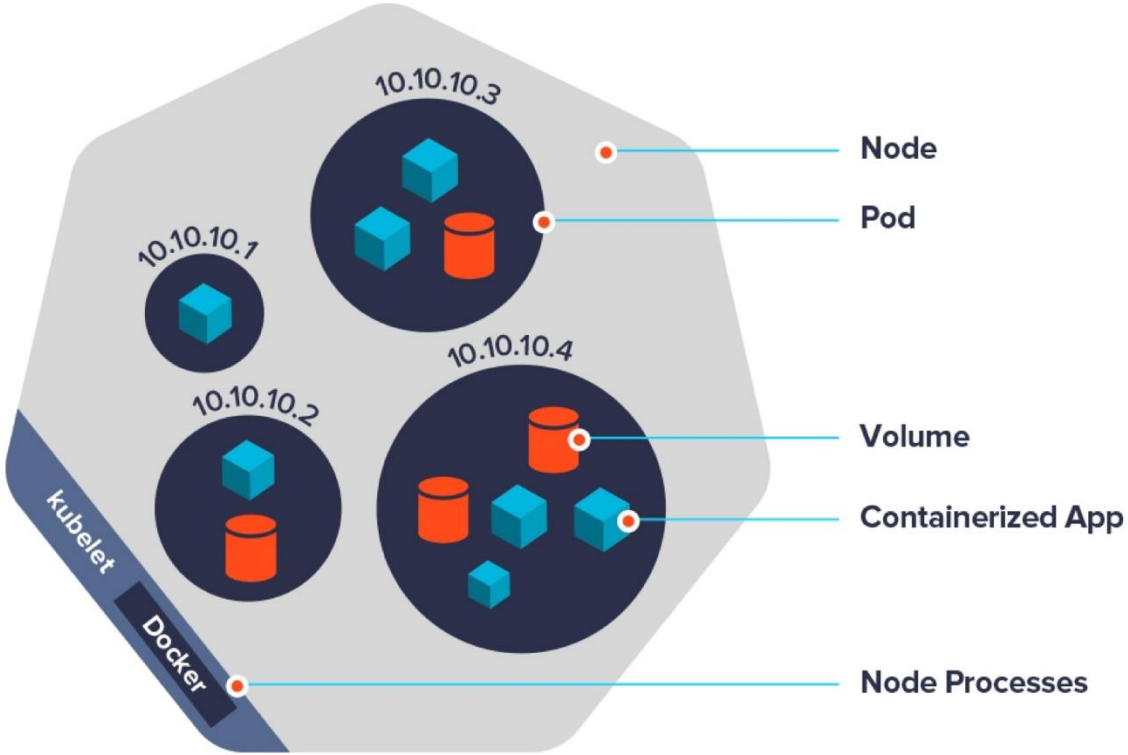
Gérer un grand nombre de conteneurs peut-être un défi. Quelles pourraient être les problématiques ?

- Orchestrer ces conteneurs
- Gérer la scalabilité des conteneurs
- Gérer la haute disponibilité
- Gérer les échecs
- Déployer ce parc de containers sur plusieurs serveurs
- L’inventaire
- Gestion des données / Volumes

Kubernetes, le meilleur ami du DevOps



Node : Mon précieux

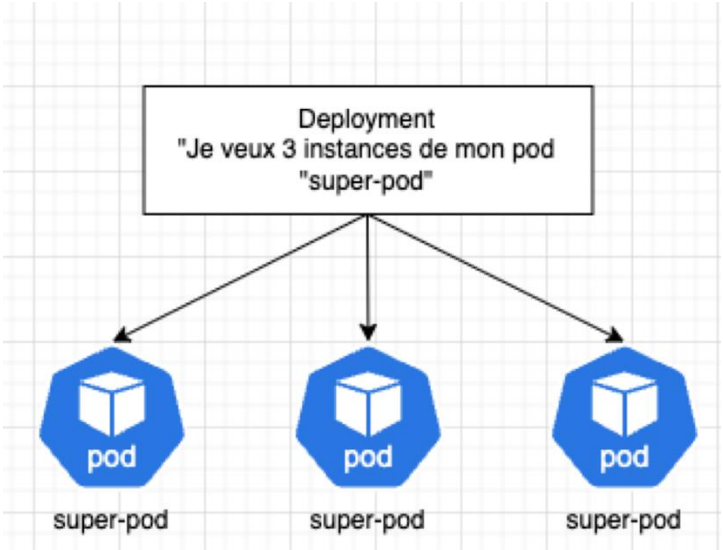


Pod : La plus petite unité de ressource

epsi-pod.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: super-pod
spec:
  containers:
    - name: container-1
      image: appli-epsi-1:latest
      ports:
        containerPort: 8080
      volumeMounts:
        - name: volume
          mountPath: /data
    - name: container-2
      image: appli-epsi-2:latest
      volumeMounts:
        - name: volume
          mountPath: /data
  volumes:
    - name: volume
      emptyDir: {}
```

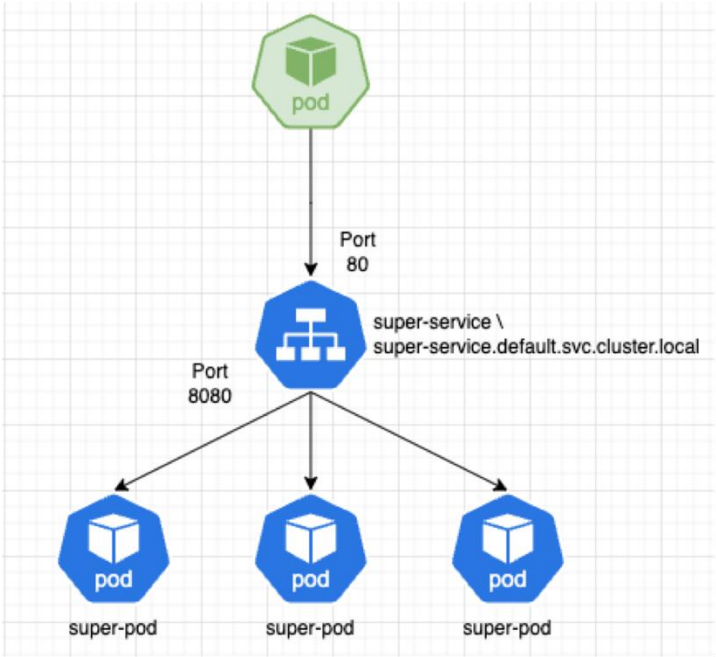

Deployment : L’orchestration des pods



epsi-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: super-pod
  template:
    metadata:
      labels:
        app: super-pod
    spec:
      containers:
        - name: container-1
          image: appli-epsi-1:latest
          ports:
            - containerPort: 8080
          volumeMounts:
            - name: volume
              mountPath: /data
        - name: container-2
          image: appli-epsi-2:latest
          volumeMounts:
            - name: volume
              mountPath: /data
      volumes:
        - name: volume
          emptyDir: {}
```

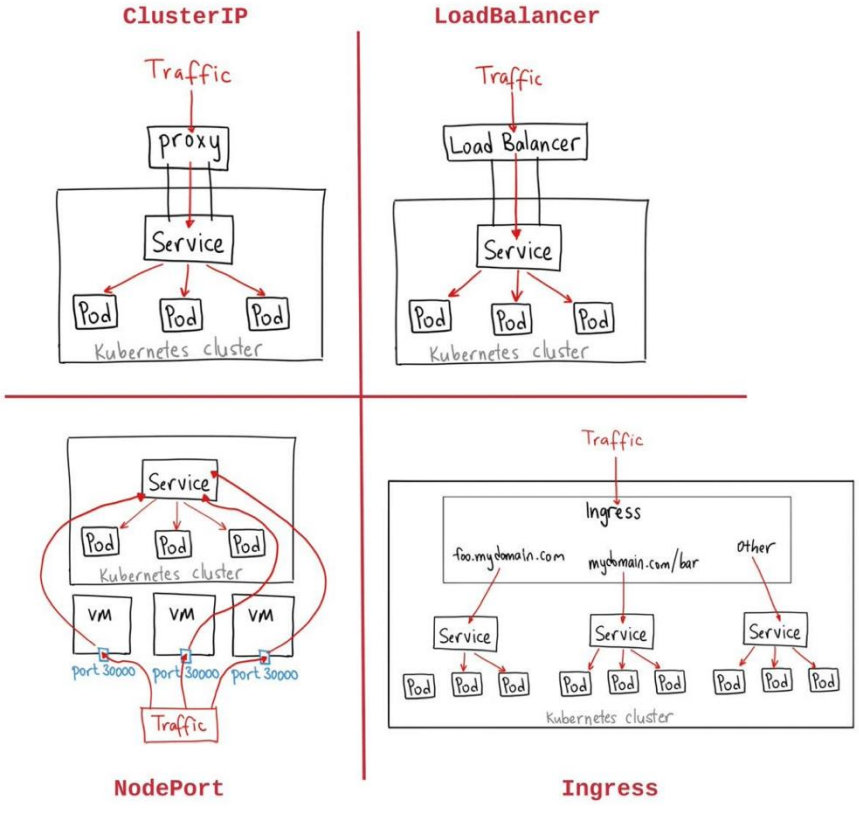
Service : La communication entre pods



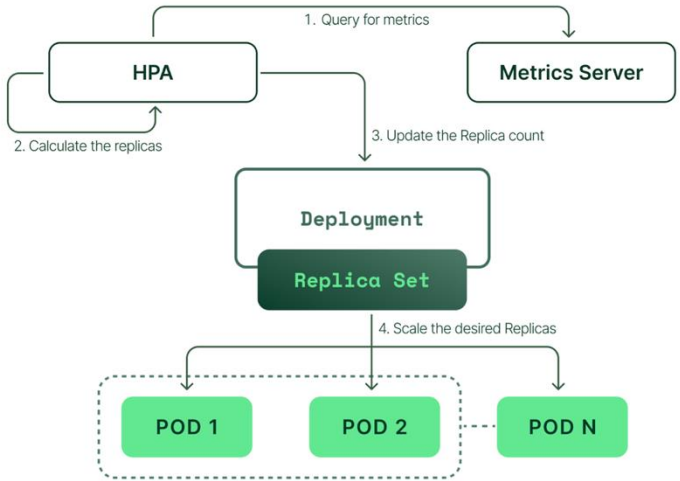
epsi-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: super-service
spec:
  selector:
    app: super-pod
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: ClusterIP
```


Service : Les différents types d’exposition



HPA : Le scaling horizontal

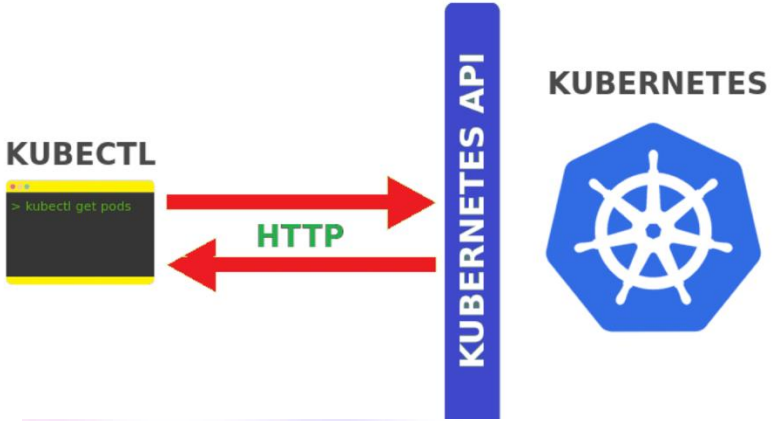


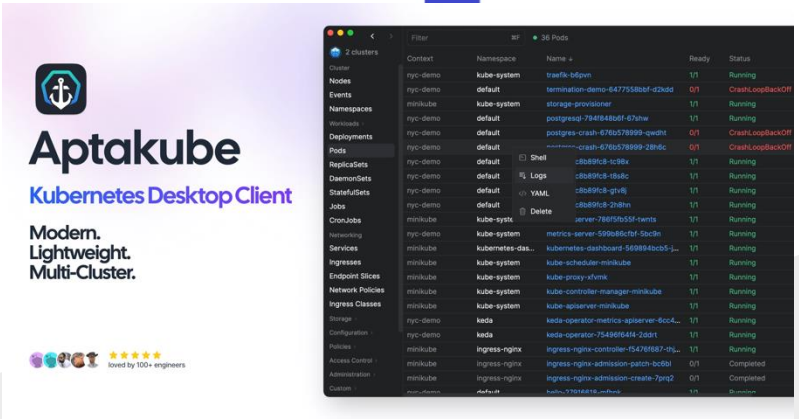
epsi-hpa.yml

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: super-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mon-deployment
  minReplicas: 2
  maxReplicas: 4
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```


kubectl : le client Kubenetes

Commande	Description
kubectl get nodes -o=wide	Permet de lister les noeuds et de récuperer leurs status
kubectl get pods	Listes les pods et leurs
kubectl get deployment	Liste les déploiements et leurs status
kubectl describe deployment	Liste les déploiements en donnant des informations étendues sur chacun
kubectl delete deployment <nom>	Détruit un déploiement
kubectl apply -f ressource.yml	Créé / met à jour des ressources à partir d'un fichier yml
kubectl delete -f ressource.yml	Détruit des ressources à partir d'un fichier yml





Au boulot

Projet road to k8s expert

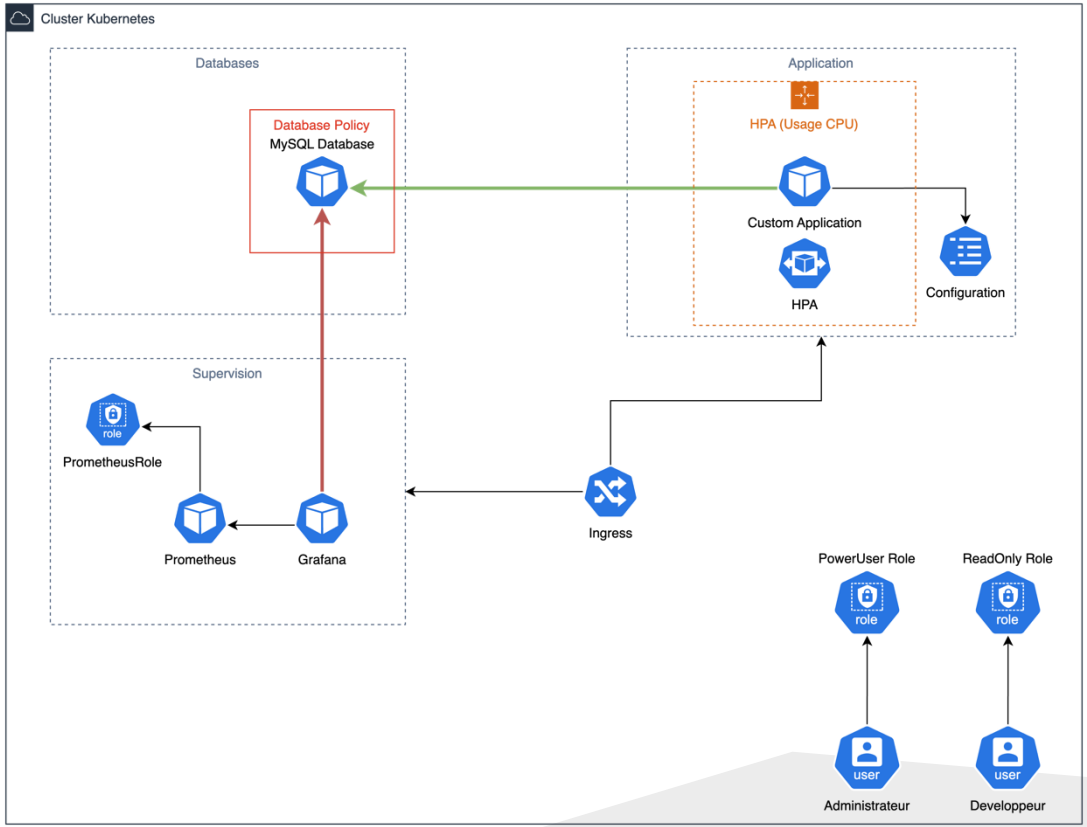
Context

Vous êtes consultant. Une entreprise souhaite déployer une application web critique sur un cluster Kubernetes. L'architecture doit garantir la séparation des composants sensibles et assurer un bon niveau de sécurité réseau tout en permettant une montée en charge efficace. Le cluster doit intégrer un outil de supervision et monitoring.

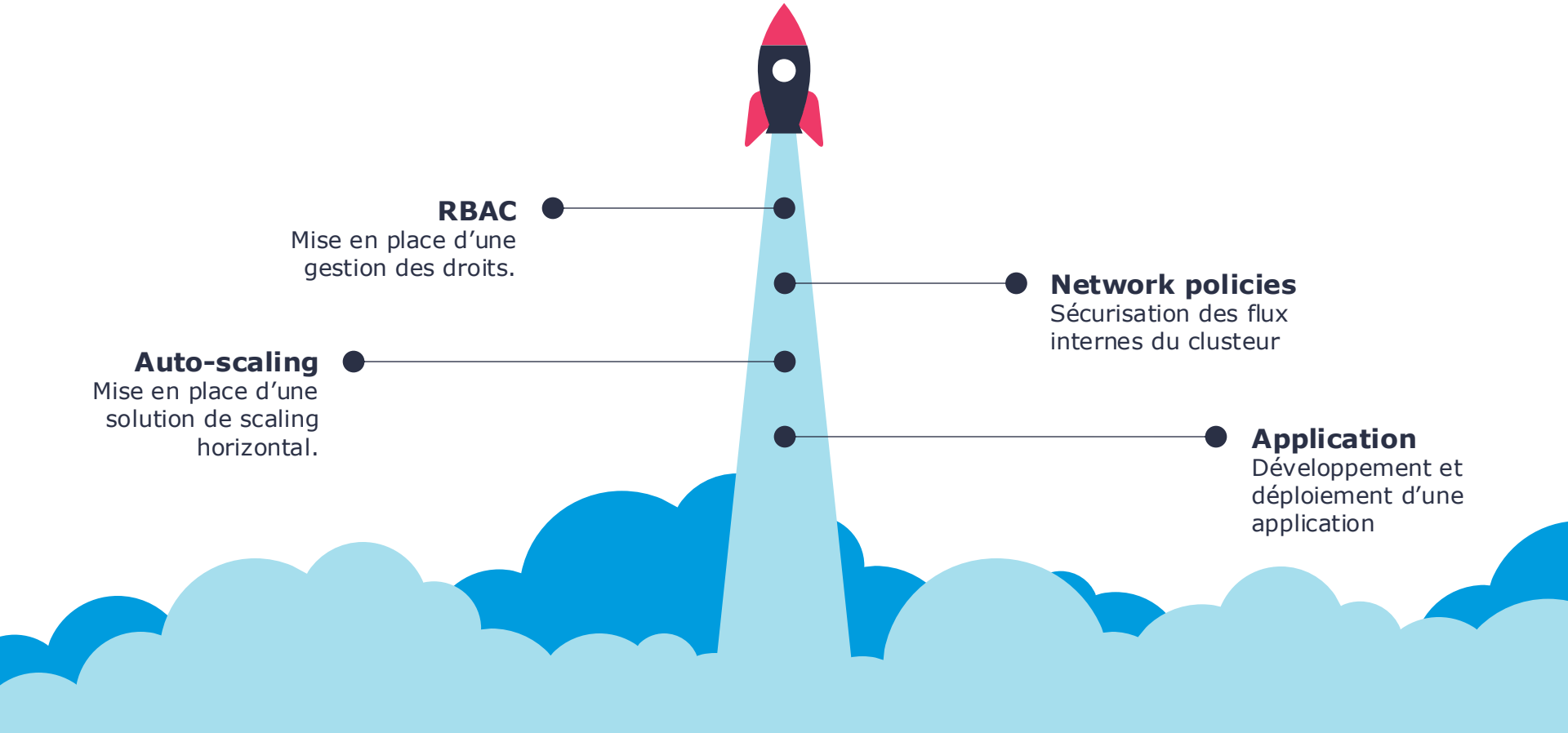
Vous êtes chargés de mettre en place cette architecture en suivant le schéma fourni.

Afin d'être certain du fonctionnement de la scalabilité, le client vous demande de démontrer son fonctionnement.

HLD fournit par le client



Objectifs



RBAC
Mise en place d'une gestion des droits.

Network policies
Sécurisation des flux internes du clusteur

Auto-scaling
Mise en place d'une solution de scaling horizontal.

Application
Développement et déploiement d'une application

Livrables

Il est recommandé de faire une simple API avec deux endpoints :

- Une lecture en bdd
- Une boucle de 3min qui fait monter le cpu

01

Application de POC

Développer une application de PoC qui fait un appel en bdd, et qui démontre le fonctionnement du HPA. Elle doit récupérer sa config via un configmap

02

Configuration du cluster

Tous les fichiers de configuration YML du cluster organisés dans un GIT.

La mise en place d'un dashboard grafana sera un plus.

03

Une démonstration

Une démonstration de ce qui a été mise en place doit être faite au client.

Des questions vous seront posés tel qu'un client pourrait le faire.

