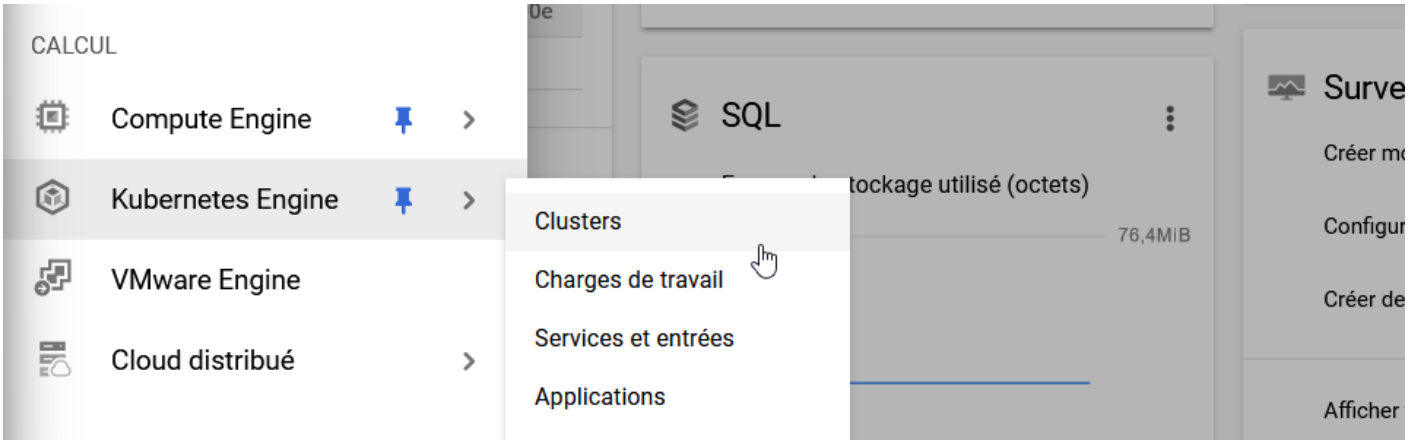
Google Cloud Platform

- Google Kubernetes Engine
- SQL Instance
- CI / CD avec GKE et Cloud Build

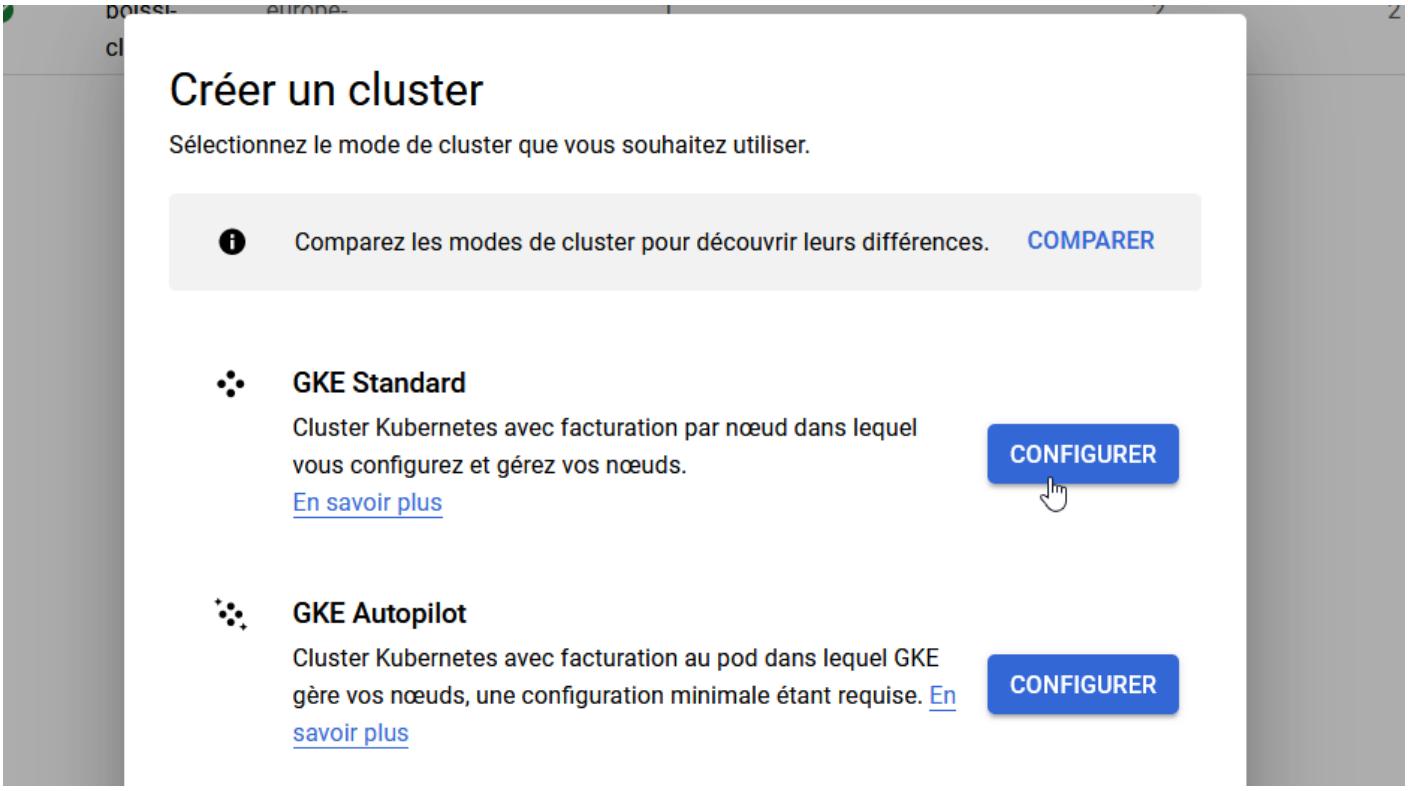
Google Kubernetes Engine

Mise en place d'un cluster GKE

Dans la section **CALCUL** de **GCP**, allez sur le service **Kubernetes Engine > Clusters**.



Une fois sur cette page, il peut vous être demandé d'activer GKE. Cela fait, cliquez sur le bouton **Créer** pour ajouter un nouveau cluster, puis choisissez le mode **GKE Standard.**



Configuration du cluster

Paramètres de base du cluster

- Donnez un nom à votre cluster
- Choisissez une zone proche de là où vous trouvez
- Choisissez une version de Kubernetes (celle proposée par défaut est très bien)

Paramètres de base du cluster

Le cluster sera créé avec le nom, la version et l'emplacement que vous indiquez ici. Une fois le cluster créé, le nom et l'emplacement ne peuvent pas être modifiés.

Pour effectuer des tests avec un cluster abordable, sélectionnez **Mon premier cluster** dans le menu **Guides de configuration des clusters**

Nom
cluster-1

Type d'emplacement
Le prix des ressources peut varier d'une région à l'autre. En savoir plus

- (•) Zonal
- ( ) Régional

Zone
europe-central2-a

[ ] Spécifiez les emplacements de nœuds par défaut

Emplacements par défaut actuels : europe-central2-a

Version du plan de contrôle

Choisissez une version disponible pour la gestion automatique de la version de votre cluster et de la fréquence de mise à niveau. Optez pour une version statique pour gérer la version de votre cluster de façon plus directe. En savoir plus.

- ( ) Version stable
- (•) Version disponible

Version disponible
Version standard (par défaut)

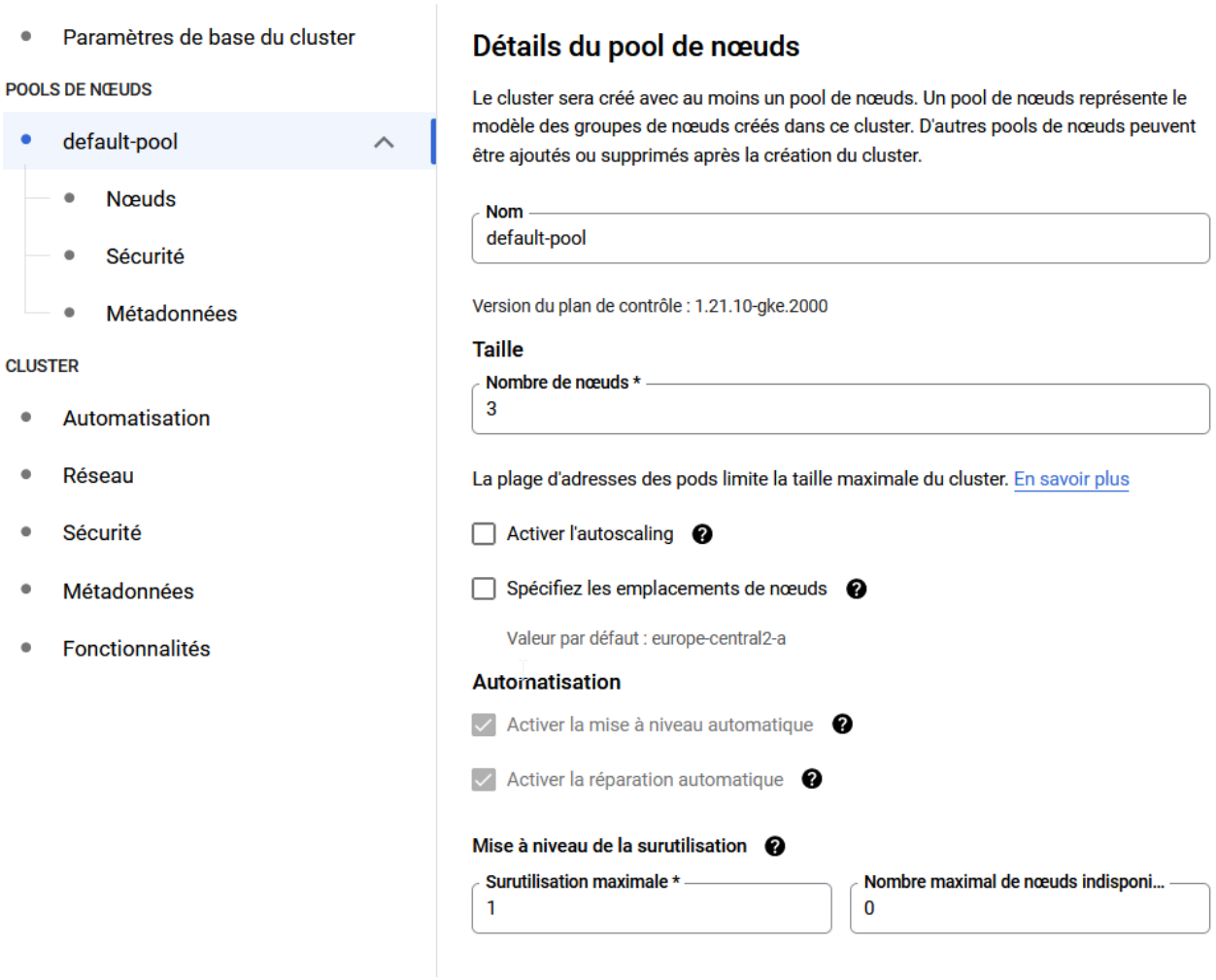
Version
1.21.10-gke.2000 (par défaut)

Ces versions ont réussi les tests de validation interne et sont considérées de qualité production, mais ne disposent pas de données historiques suffisantes pour garantir leur stabilité. Les problèmes connus ont généralement des solutions de contournement connues. Notes de version

Pools de nœuds

- Donnez un nom à votre pool
- Choisissez la taille
- Pour éviter une surfacturation, laissez décoché l'autoscaling

> Le nombre de noeuds peut être redimensionnable manuellement après la création du cluster en cas de besoin



- Côté nœuds, laissez le type d'image à **Container Optimized** pour minimiser la consommation de ressources
- Sélectionnez aussi le type de nœuds

> En dessous de 2 Go de RAM cela va être limite, surtout si vous souhaitez utiliser des annotations de demande de ressources dans vos pods car ceux présents de base prennent déjà quasiment tout.

- Taille du disque : 30 Go pour minimiser les frais
- Le reste des options peuvent être laissées par défaut

- Paramètres de base du cluster

POOLS DE NŒUDS

- default-pool
  - Nœuds
  - Sécurité
  - Métadonnées

CLUSTER

- Automatisation
- Réseau
- Sécurité
- Métadonnées
- Fonctionnalités

Nœuds

Ces paramètres de nœud seront utilisés lorsque des nœuds seront créés à l'aide de ce pool de nœuds.

Type d'image
Container-Optimized OS avec containerd (cos_containerd) (par défaut)

Pour les clusters et les pools de nœuds récemment créés avec la version 1.21.10-gke.2000 ou une version ultérieure, l'image de nœud Linux par défaut est Container-Optimized OS avec Containerd. Pour les pools de nœuds Windows utilisant la version 1.21 ou une version ultérieure, Containerd est également l'environnement d'exécution recommandé. Étant donné que Dockershim devient obsolète dans le projet Kubernetes, GKE va abandonner les images de nœuds Docker. Nous vous recommandons de migrer vers des images de nœuds containerd dès que possible. Apprenez-en plus sur les différentes images de nœuds.

Configuration de la machine

Famille de machines

USAGE GÉNÉRAL

Types de machines pour les charges de travail courantes permettant d'optimiser les coûts et la flexibilité

Série
E2

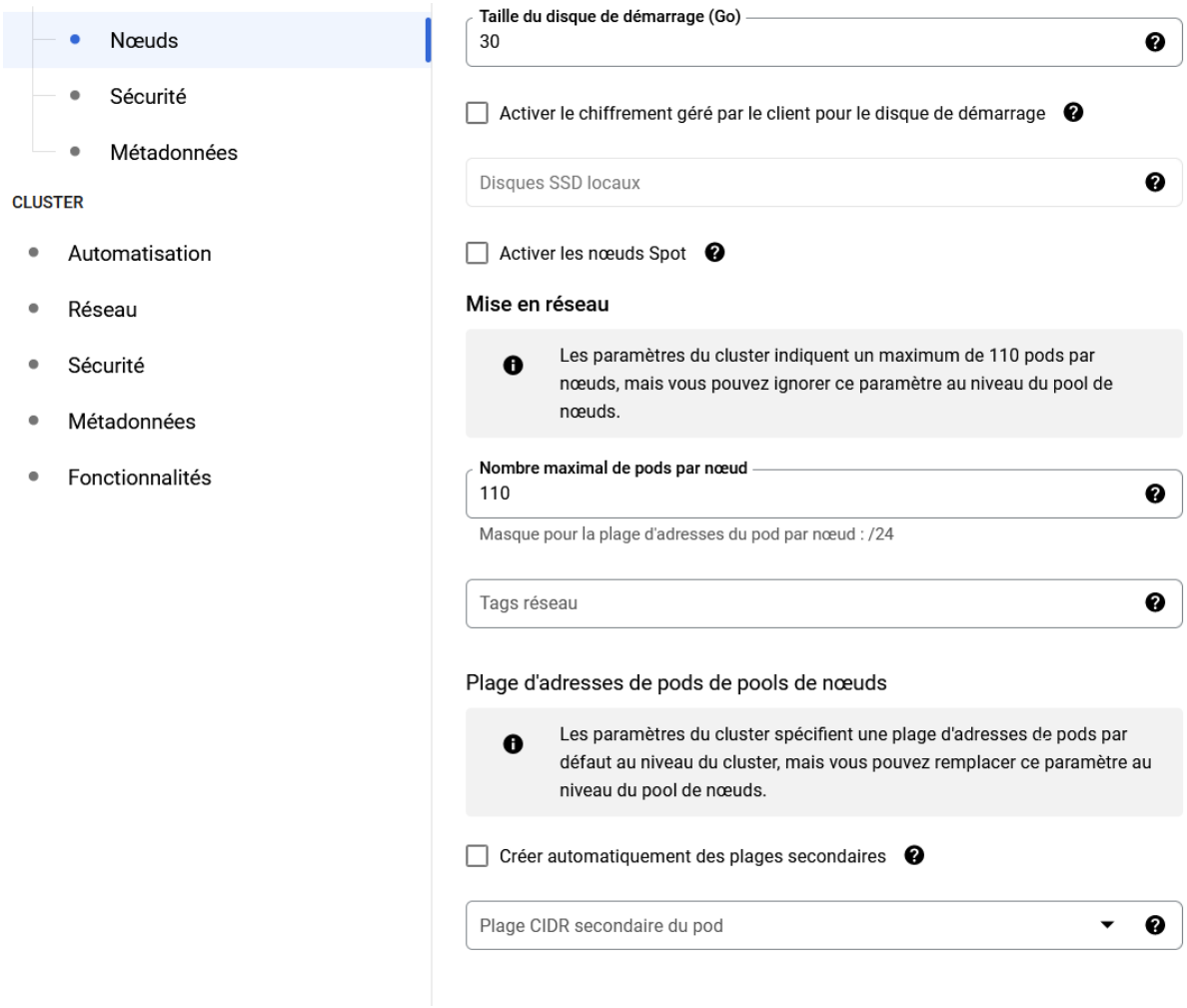
Sélection de la plate-forme de processeur en fonction de la disponibilité

Type de machine
e2-small (2 processeurs virtuels, 2 Go de mémoire)

vCPU	Memory
Un cœur partagé	2 Go

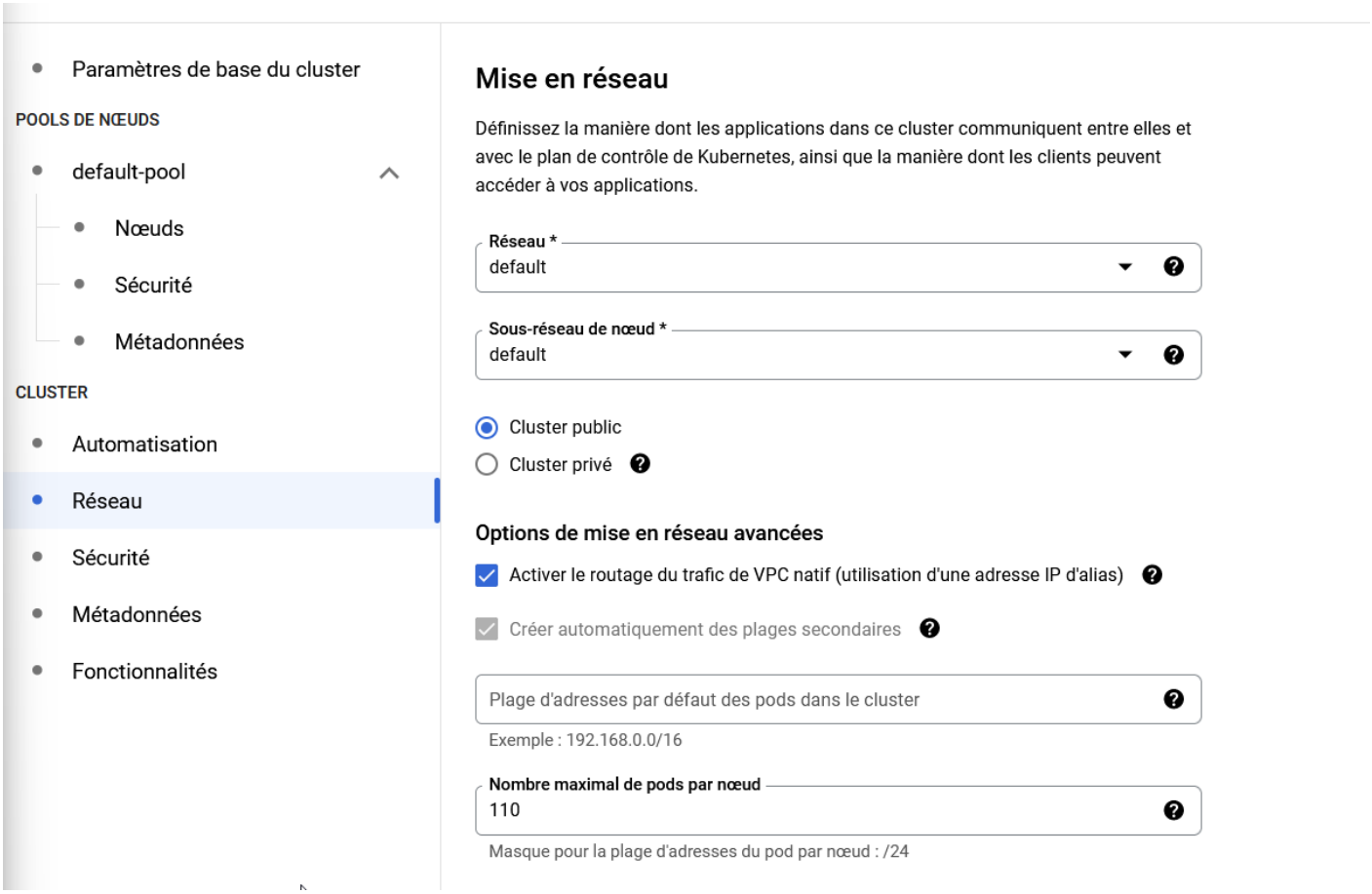
PLATE-FORME DU PROCESSEUR ET GPU

Type de disque de démarrage
Disque persistant standard



Réseau

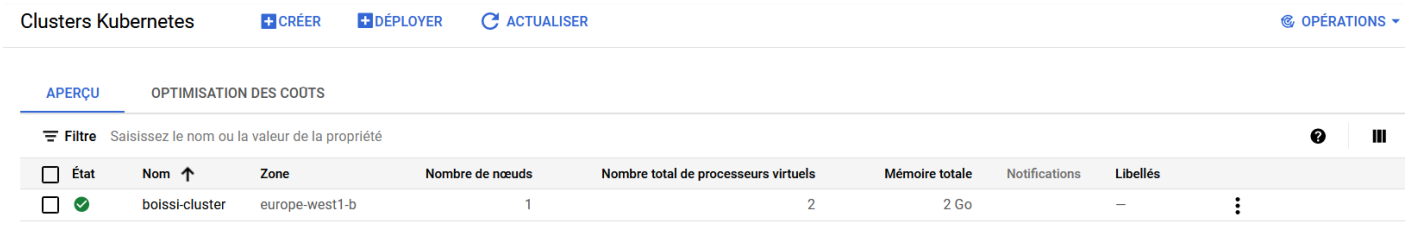
Vous pouvez définir la visibilité de votre Cluster (par défaut public) ainsi que le réseau sur lequel il se trouve.
On peut laisser par défaut.



Création

Cliquez ensuite sur **Créer** en bas de la page pour créer le cluster Kubernetes.

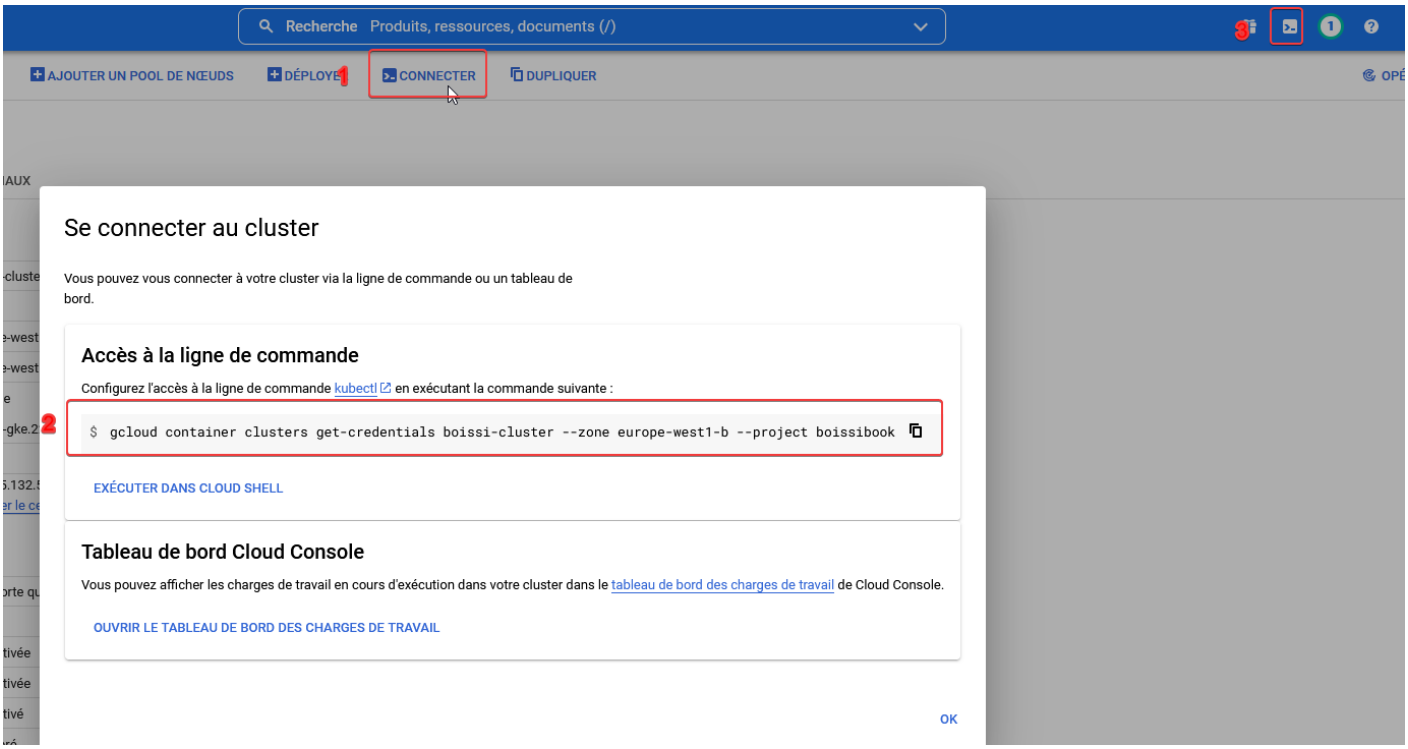
Après quelques minutes, ce dernier devrait apparaitre dans la liste des clusters avec un état **Vert** signifiant qu'il est opérationnel.

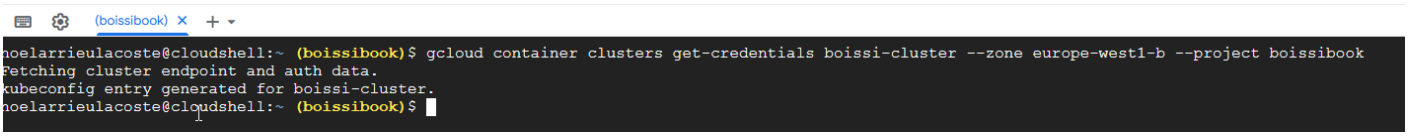


Connexion au Cluster via la console GCP

pour manipuler notre cluster, on peut s'y connecter via la console GCP.

En allant voir les détails du cluster, cliquez sur **Connecter**, puis copiez collez la commande de connexion dans votre console Cloud Shell.





On peut désormais effectuer quelques tests avec la commande **kubectl**

```
noelarrieulacoste@cloudshell:~ (boissibook)$ kubectl top nodes
NAME                                        CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
gke-boissi-cluster-pool-2-3eb7953f-9pxb     123m         13%    810Mi           59%
noelarrieulacoste@cloudshell:~ (boissibook)$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                                  READY   STATUS    RESTARTS   AGE
default       boissibook-68dfd59d4c-wd4gc                           1/1     Running   0          152m
kube-system   event-exporter-gke-5479fd58c8-pzkhs                   2/2     Running   0          179m
kube-system   fluentbit-gke-pjmmv                                   2/2     Running   0          3h6m
kube-system   gke-metrics-agent-m69r7                               1/1     Running   0          3h6m
kube-system   konnectivity-agent-6f44578758-4645b                   1/1     Running   0          3h6m
kube-system   konnectivity-agent-autoscaler-7fd5dd4f5-vgpg8         1/1     Running   0          179m
kube-system   kube-dns-56494768b7-4wzsq                             4/4     Running   0          160m
kube-system   kube-dns-autoscaler-f4d55555-r6fqb                    1/1     Running   0          179m
kube-system   kube-proxy-gke-boissi-cluster-pool-2-3eb7953f-9pxb    1/1     Running   0          3h6m
kube-system   l7-default-backend-69fb9fd9f9-fhb7s                   1/1     Running   0          179m
kube-system   metrics-server-v0.4.5-bbb794dcc-6fwgn                 2/2     Running   0          179m
kube-system   pdcsi-node-qhr8h                                      2/2     Running   0          3h6m
noelarrieulacoste@cloudshell:~ (boissibook)$
```

Configuration d'un Ingress

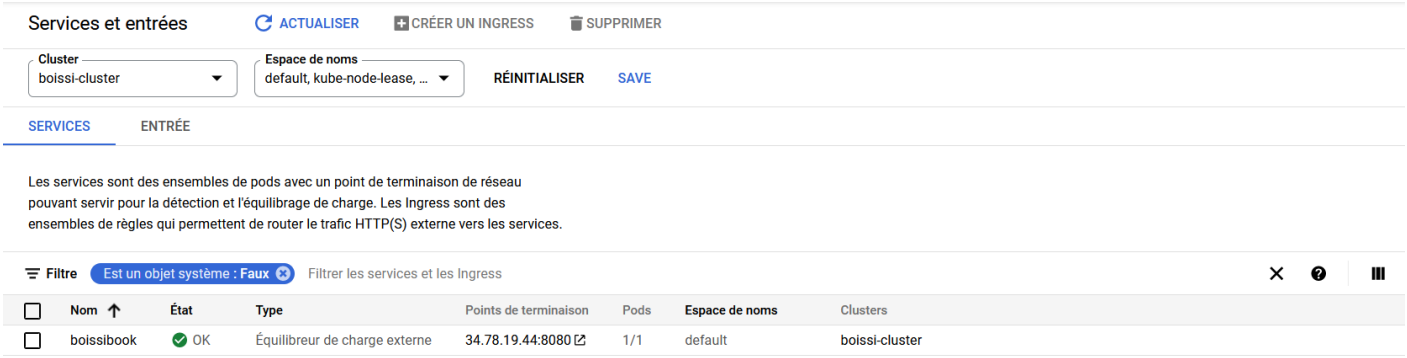
Mise en place d'un LoadBalancer

Lorsqu'une application tourne sur votre cluster, elle n'est pas directement exposée sur internet. Le mieux est de la rendre visible derrière un **LoadBalancer** autogéré par GCP.

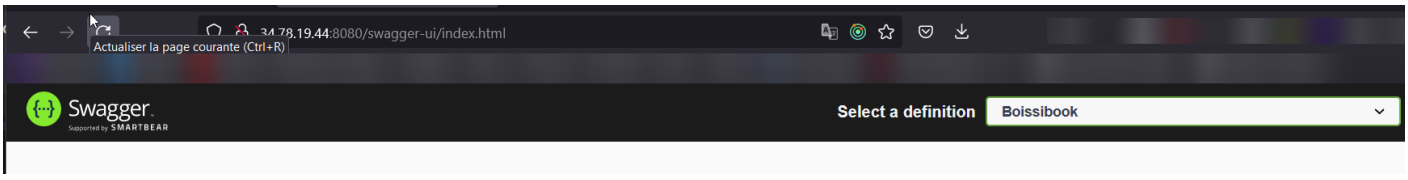
Pour cela, votre déploiement doit être exposé par service de type **LoadBalancer** :

```
kind: Service
apiVersion: v1
metadata:
  name: <deploymentName>
spec:
  selector:
    run: <deploymentName>
  ports:
    - protocol: TCP
      port: <WantedExposedPort>
      targetPort: <podExposedPort>
  type: LoadBalancer
```

En vous rendant sur **GCP > Kubernetes Engine > Services et entrées**, vous devriez pouvoir vérifier que votre **LoadBalancer** fonctionne et pointe correctement sur votre déploiement.



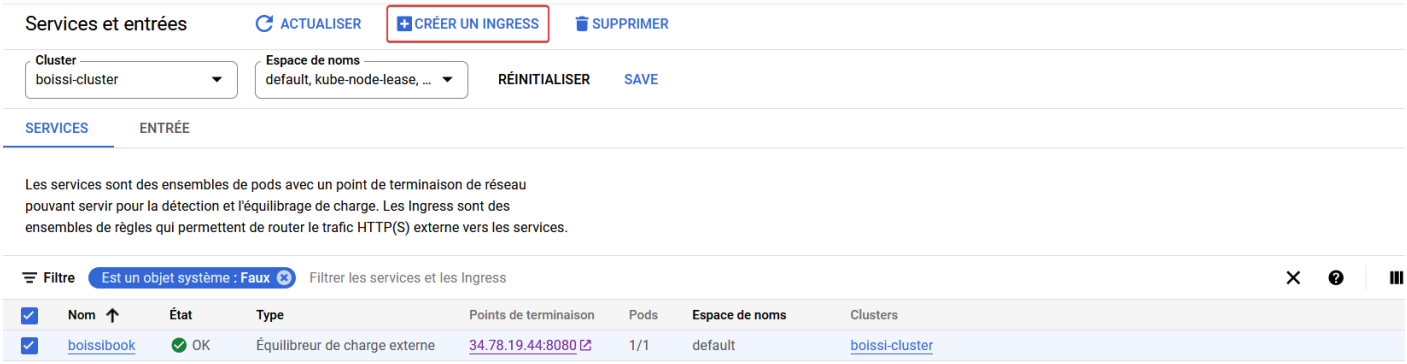
Vérifier que votre accès à votre service fonctionne correctement en vous rendant sur le **Point de terminaison**



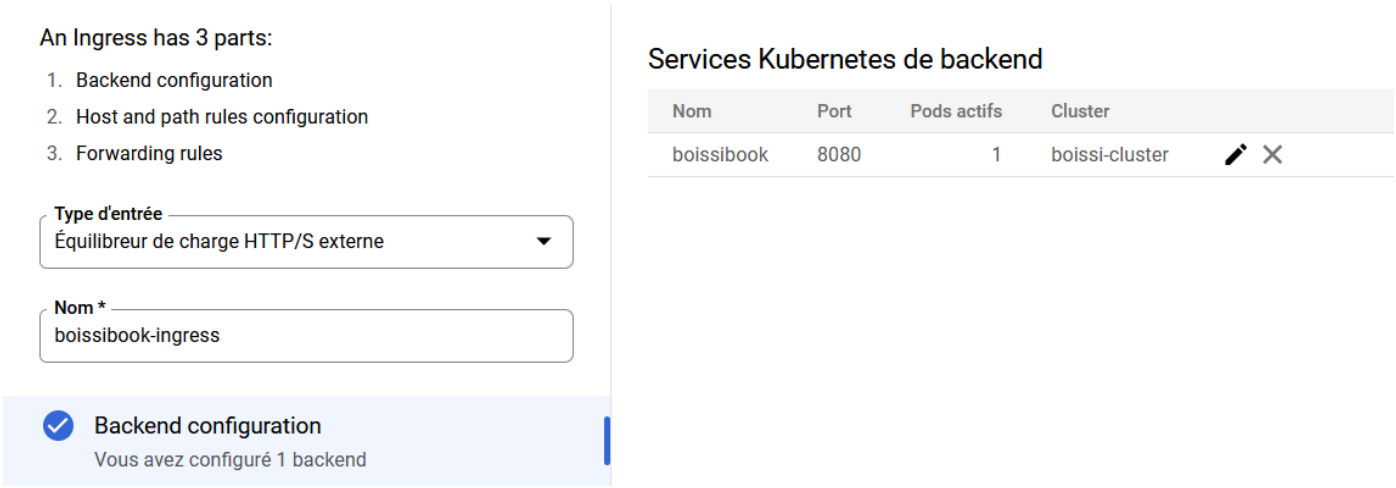
Activation d'un Ingress

Il faut maintenant activer l'exposition de ce LoadBalancer derrière un domaine et avec un certificat HTTPS.

Pour faire cela, cochez votre LoadBalancer puis cliquez sur **Créer un Ingress**

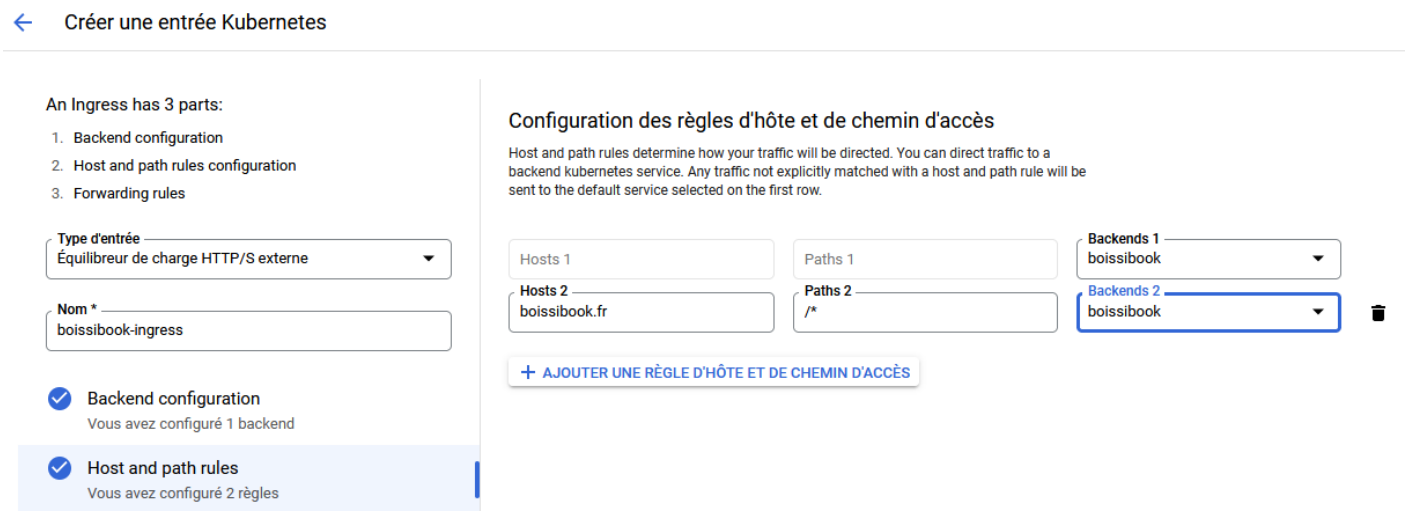


Donnez un nom à votre entrée



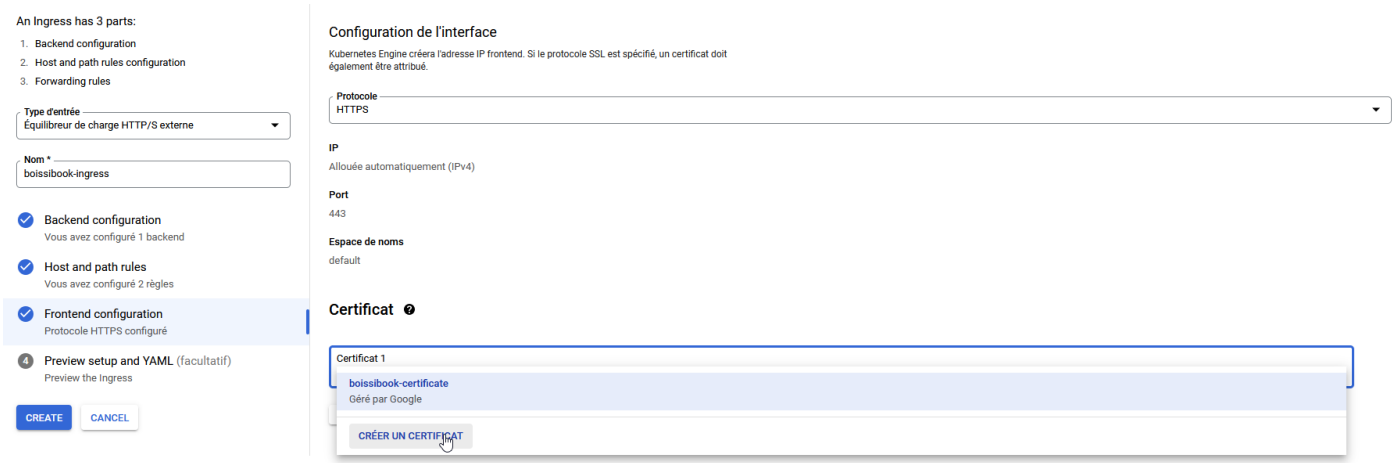
Définissez ensuite les règles d'accès à votre **LoadBalancer**, notamment le domaine par lequel y acceder

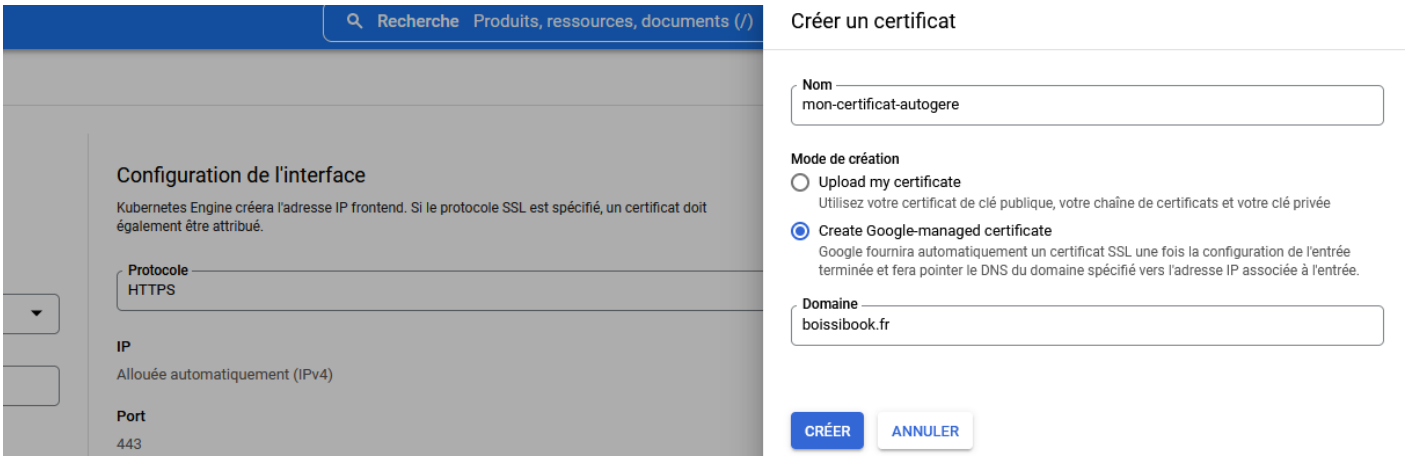
> Une fois l'IP publique de ce point d'entrée définis, il faudra bien sûr aller mettre à jour l'enregistrement DNS de type A et AAAA pour qu'il pointe sur ce dernier



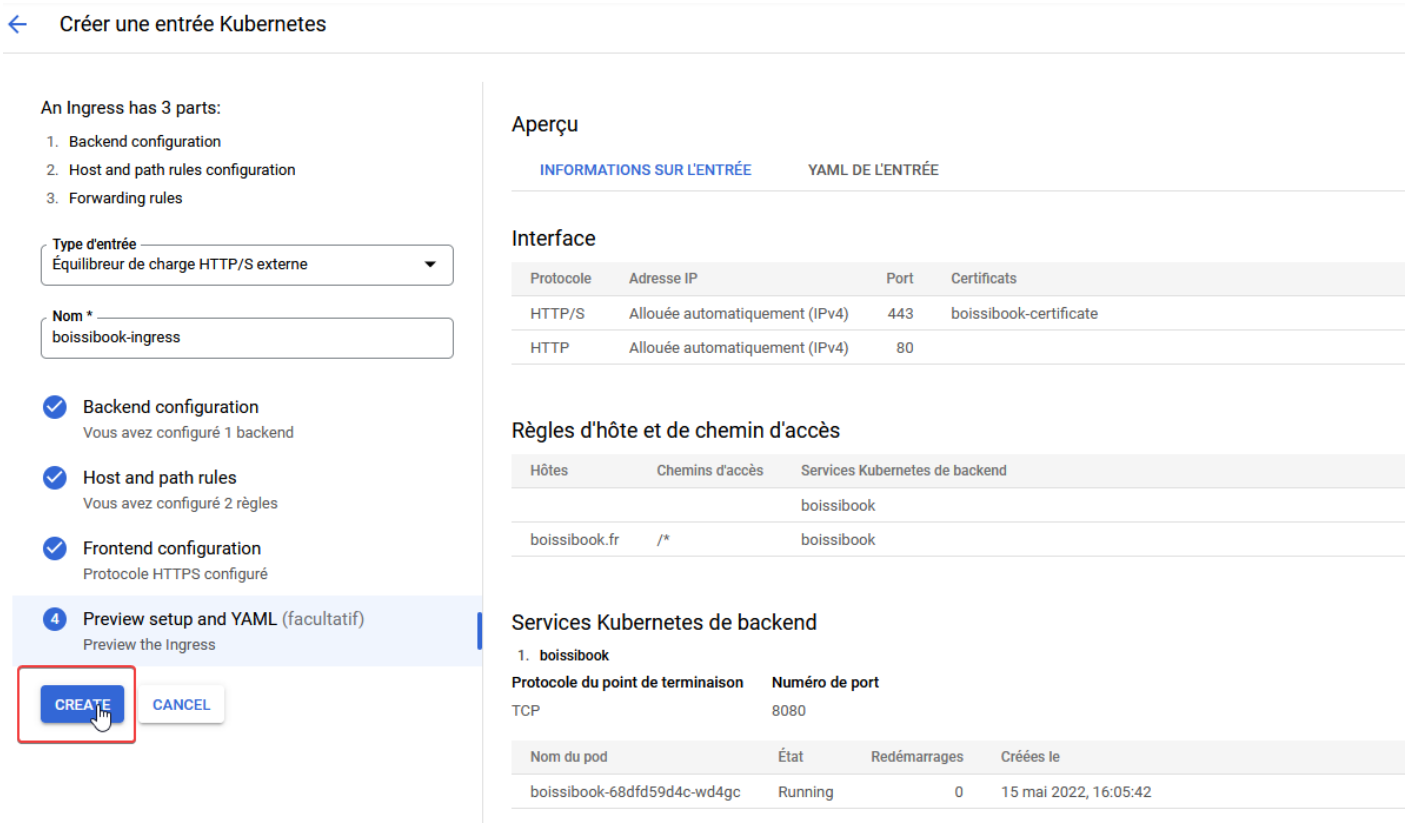
Dans la section **Frontend Configuration**, vous pouvez définir si ce dernier sera accessible en HTTP/HTTPS

Vous pouvez soit fournir un certificat existant, soit laisser GCP gérer ça pour vous.

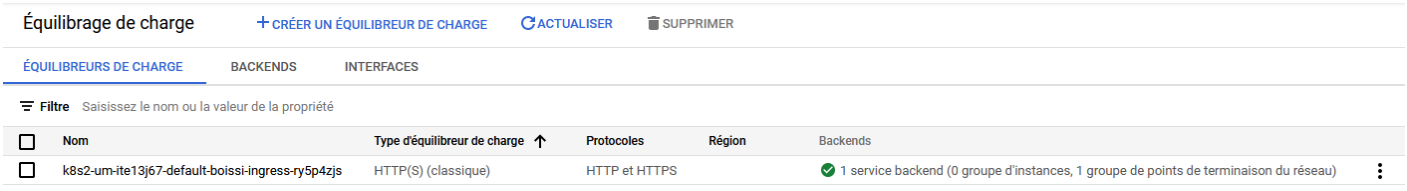




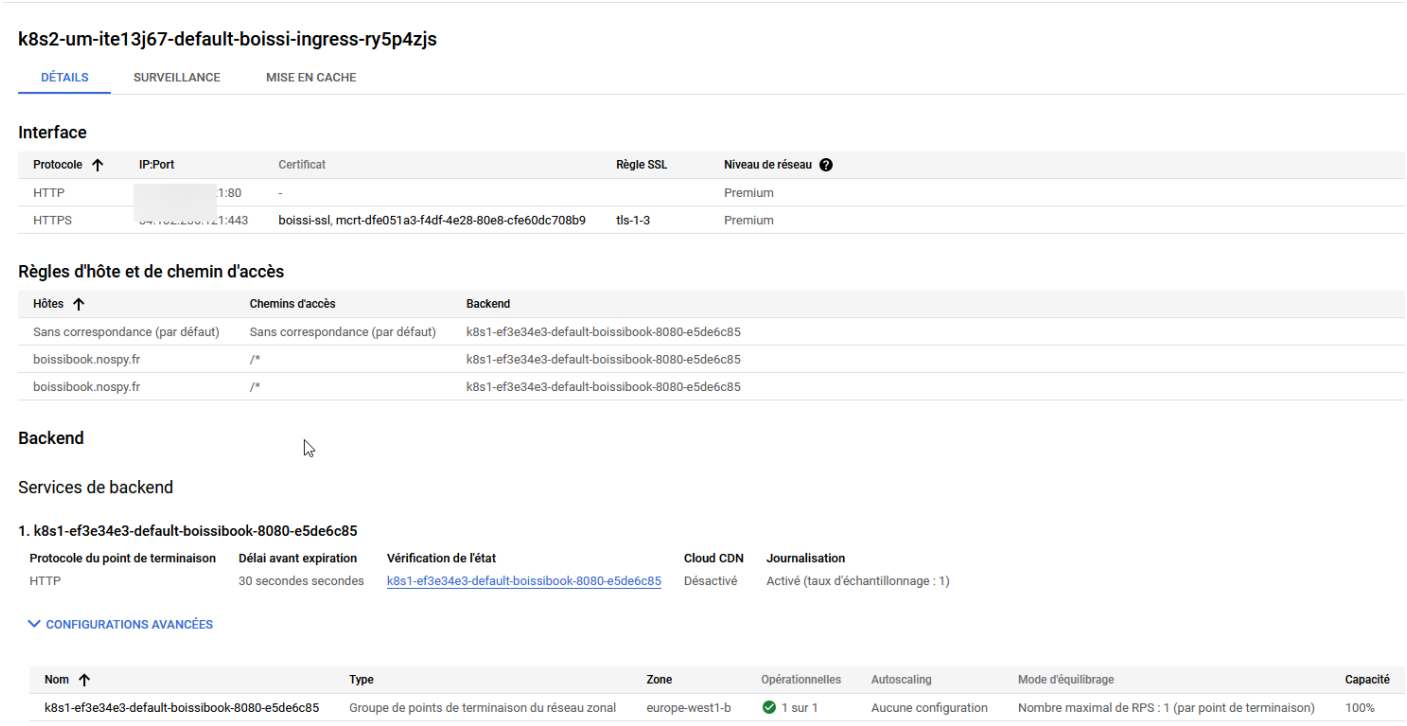
Vérifier que votre configuration est bonne, puis cliquez sur **Créer**



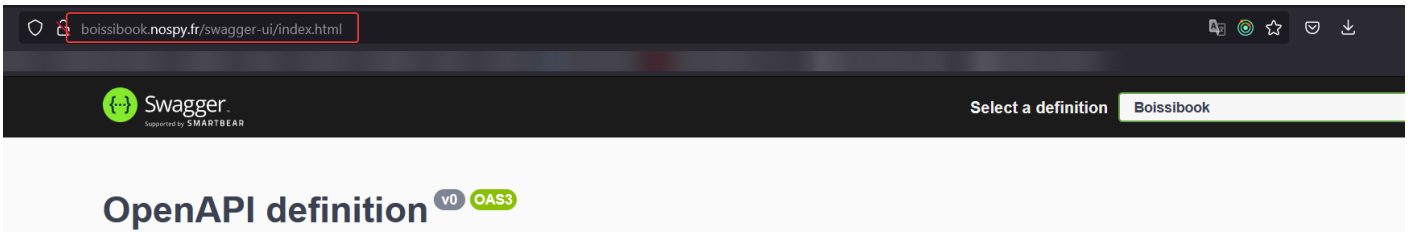
Après quelques minutes, notre déploiement Ingress devrait apparaitre dans la section **Services réseau > Équilibrage de charge**



En allant consulter les détails, on peut y retrouver l'adresse IP attribuée ainsi que d'autres informations sur son fonctionnement, notamment les certificats utilisés.



Une fois votre zone DNS mis à jour et votre cache DNS rafraichis, vous pouvez accéder à votre service via votre domaine.

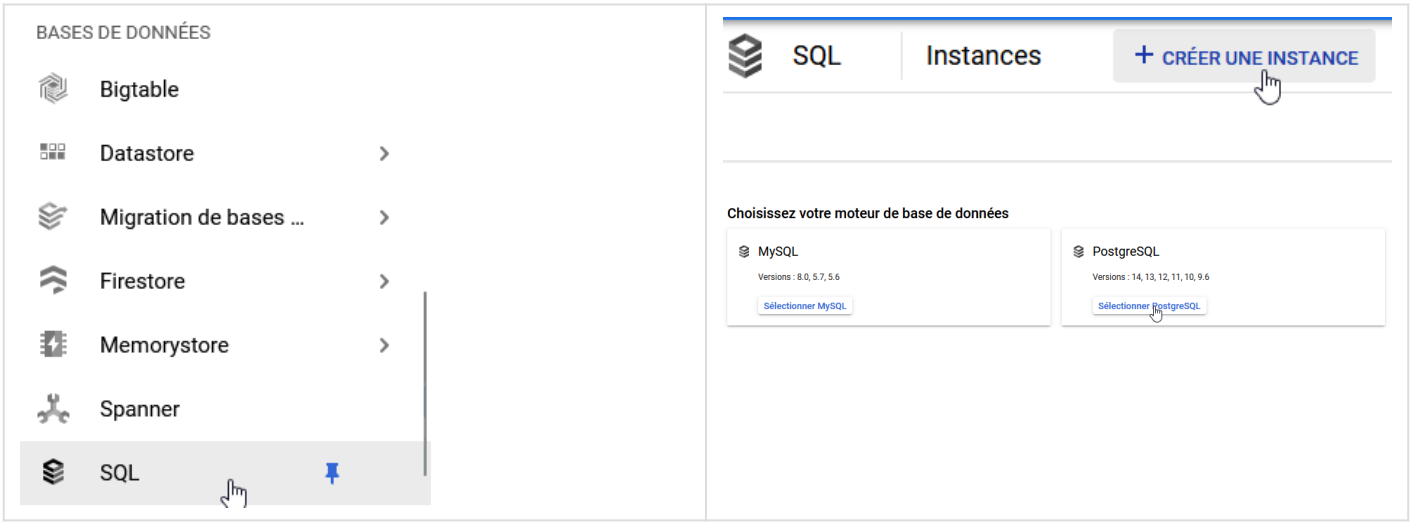


> La génération du certificat peut prendre un peu de temps (~1h)

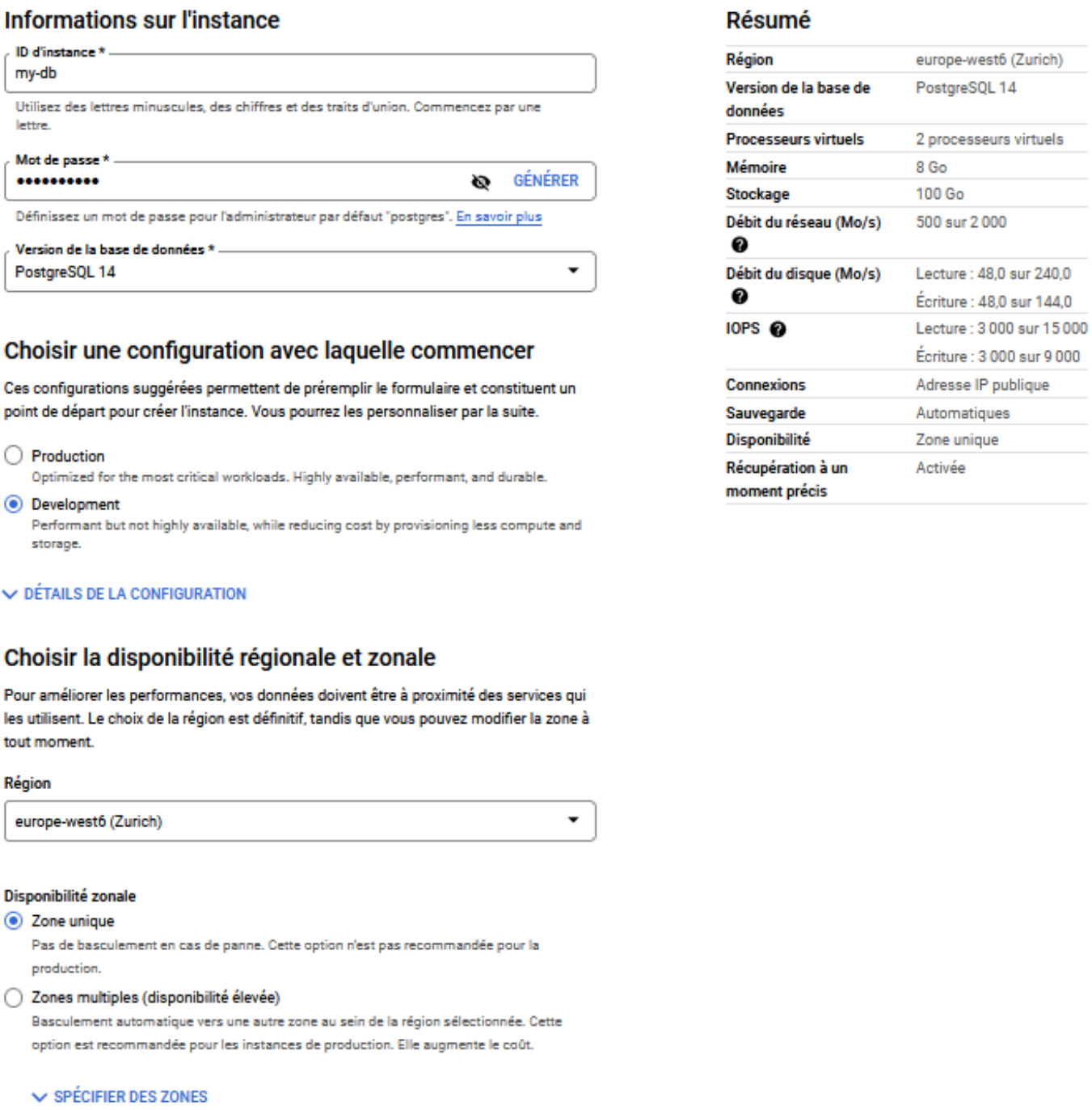
SQL Instance

Création d'une instance PostgreSQL

Rendez-vous dans la section **Bases de données > SQL**, puis cliquez sur **Créer une instance**



Remplissez les différentes informations demandées.



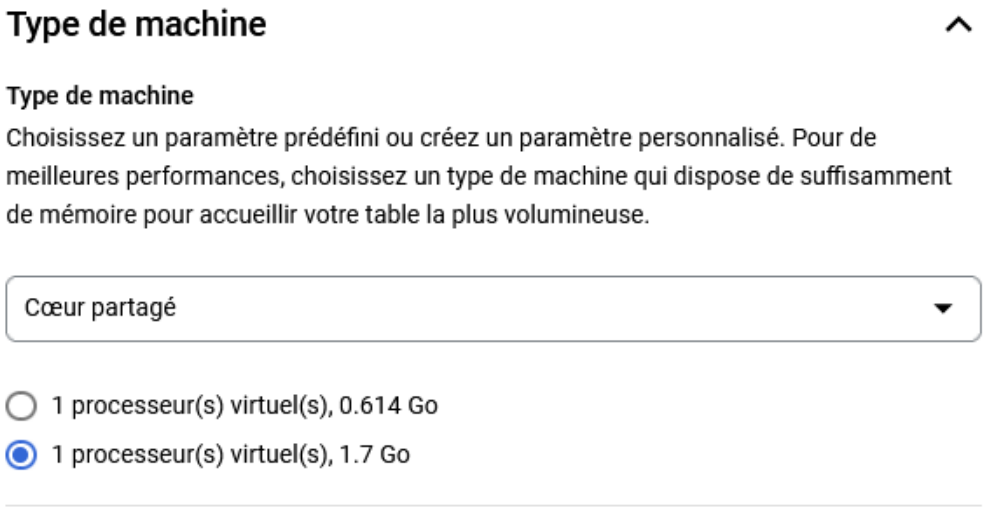
- **ID d'instance** : le nom de votre instance BDD (cela n'est pas le nom de la base)
- **Mot de passe** : Mot de passe pour l'administrateur par défaut **postgres**
- **Version de la base** : Version de la base PostgreSQL, par défaut la dernière.
- **Configuration** : Production / Développement → définit une dimension de notre base (RAM, CPU, stockage ...) que l'on pourra personnaliser ensuite.
- **Disponibilité** : La région sur laquelle sera déployée l'instance

Personnaliser l'instance

On peut personnaliser la configuration de l'instance qui de base est calibré pour une grosse utilisation et va donc engendrer des frais plus importants.

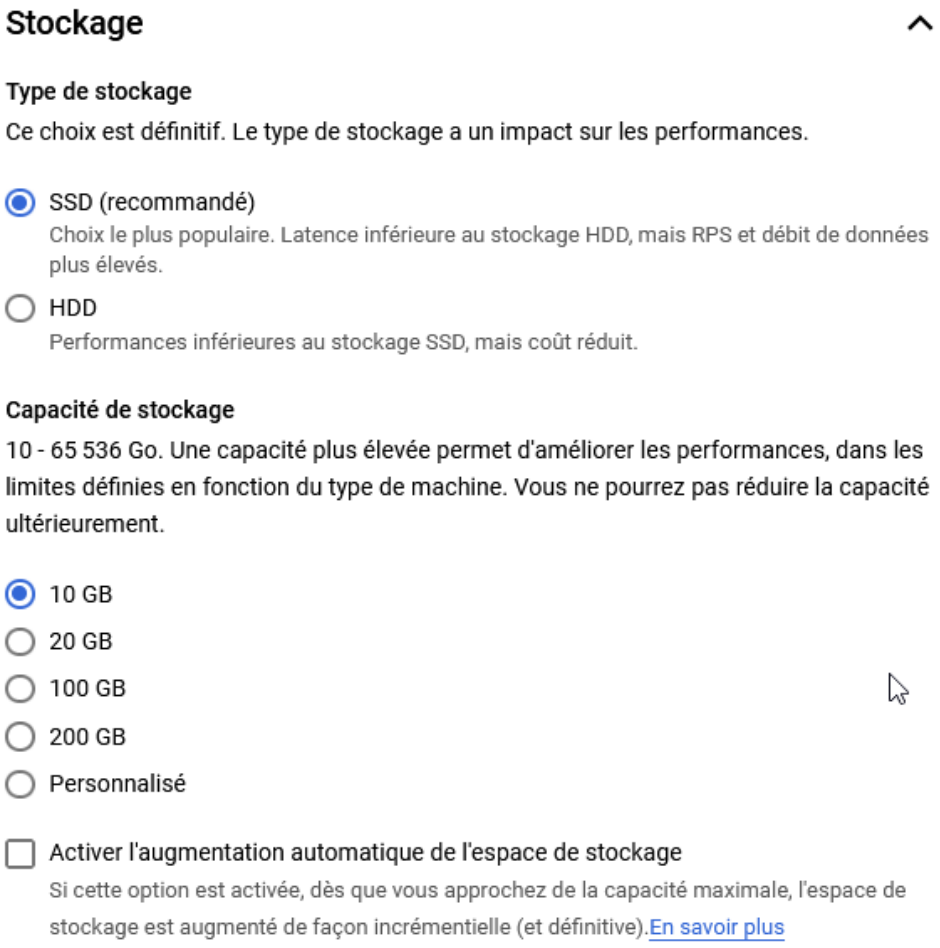
Type de machine

Choisissez un paramètre prédéfini ou créez un paramètre personnalisé.
Pour minimiser les frais, on peut configurer une instance partagée avec un processeur virtuelle (à éviter dans un environnement de production !).



Stockage

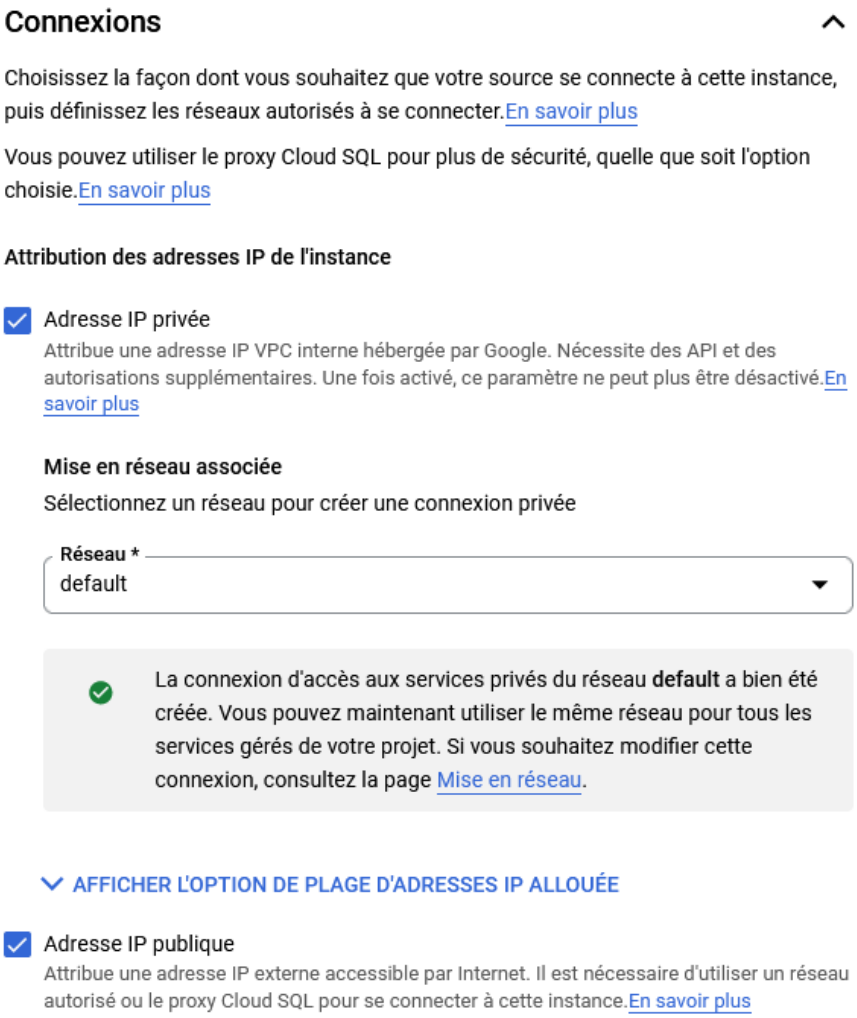
On peut redéfinir le type de stockage (HDD /SDD) ainsi que la capacité de ce dernier. On peut également activer l'augmentation automatique de l'espace de stockage en cas de saturation.
Il est tout de même recommandé de rester sur du SSD.



Connexions

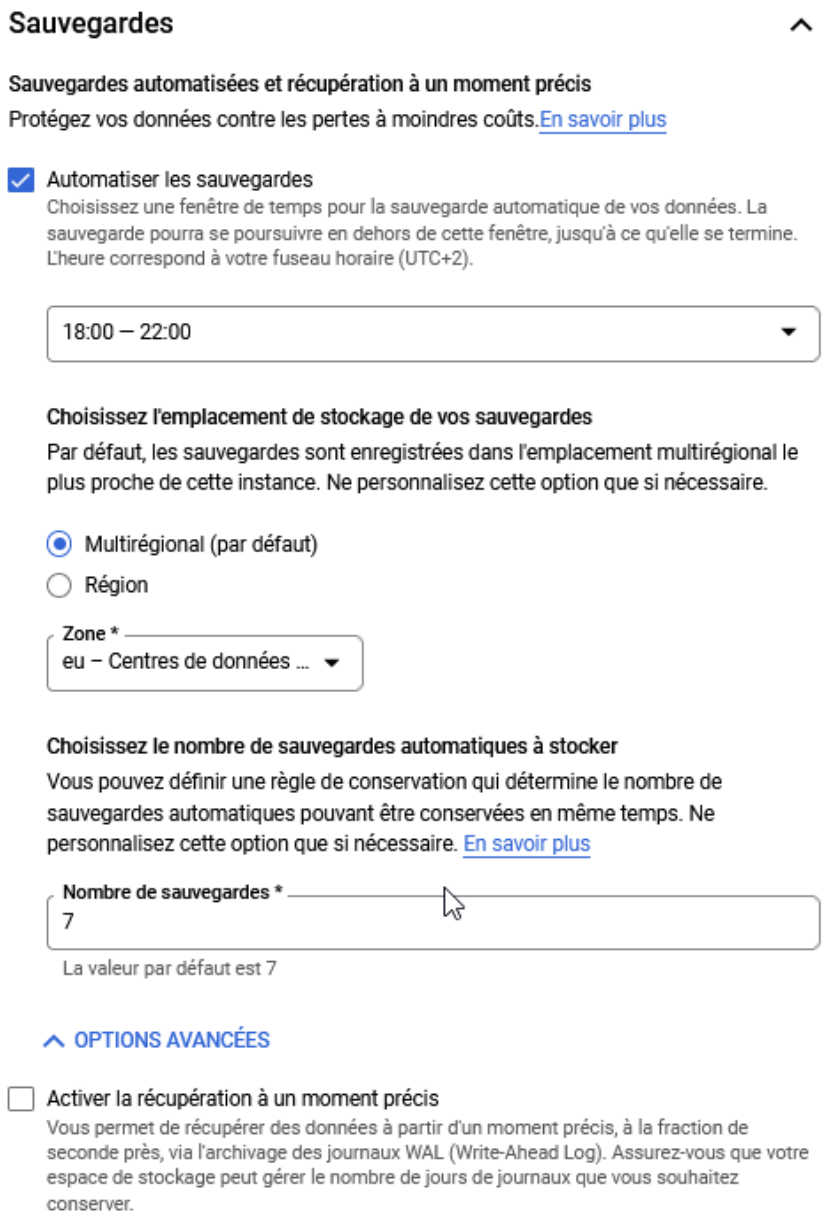
Choisissez la façon dont vous souhaitez que votre source se connecte à cette instance, puis définissez les réseaux autorisés à se connecter.

- **Privée** : Attribue une adresse IP VPC interne et accessible uniquement par d'autres instances GCP.
- **Publique :** Attribue une adresse publique par laquelle on peut se connecter de l'extérieur, il faudra tout de même autoriser manuellement les plages d'IP autorisées à s'y connecter.



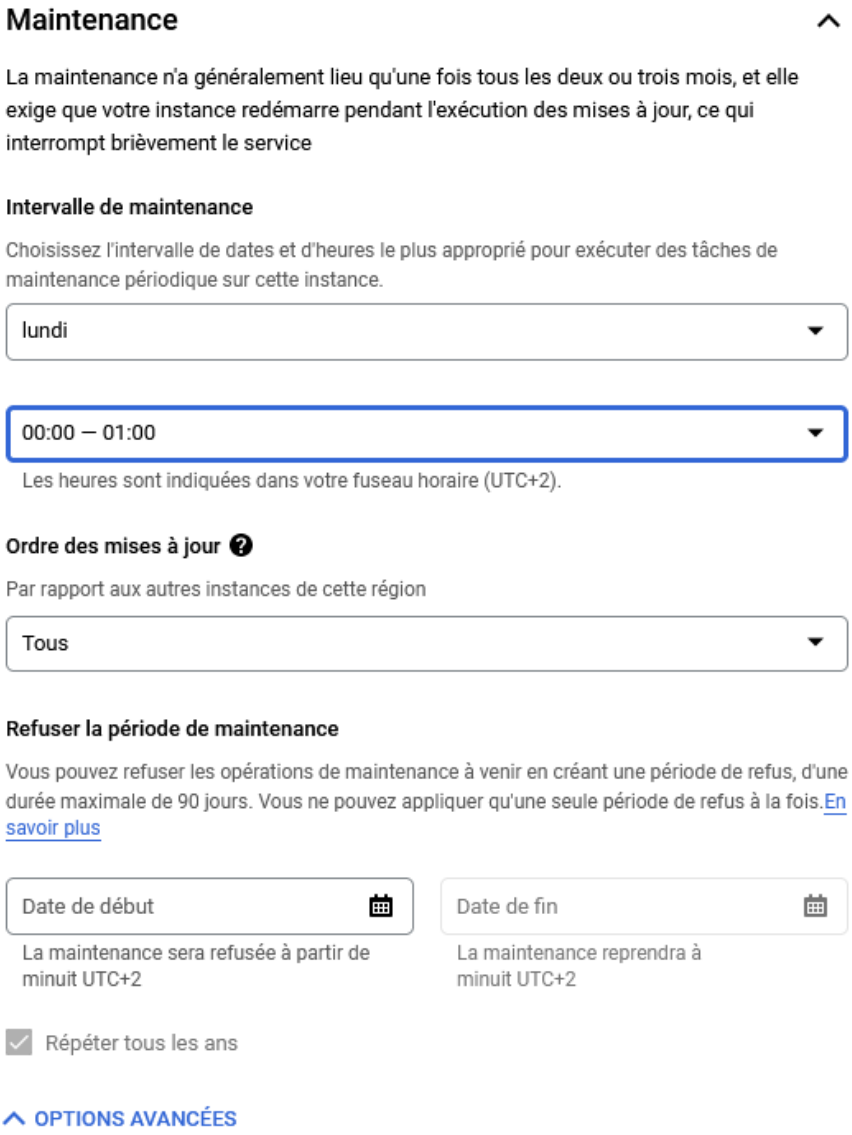
Sauvegarde

Par défaut, une sauvegarde par jour est configurée sur les sept derniers jours à une heure précise. On peut redéfinir tout ça.



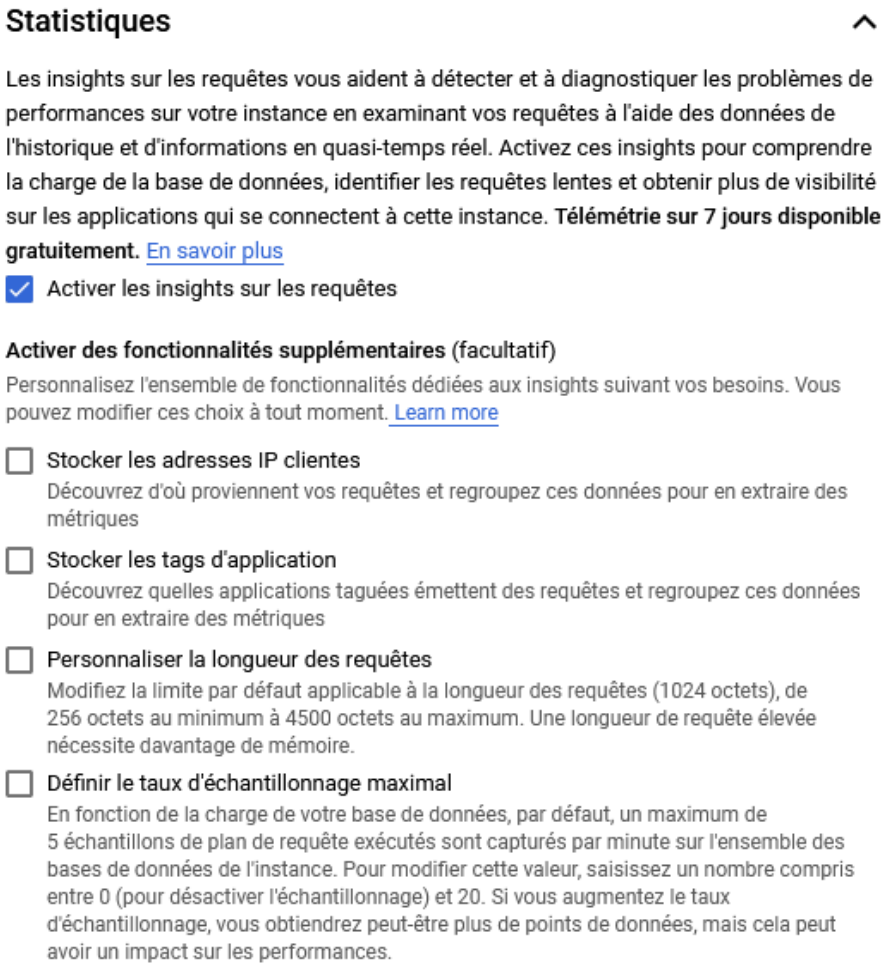
Maintenance

La maintenance est rare (2-3 mois), mais nous pouvons quand même demander à ce que cette dernière se fasse un certains jours, sur une plage horaire précise pour éviter que cela se fasse en plein moment de forte utilisation.



Statistiques

On peut activer certains insight afin de monitorer le comportement de notre base et les requêtes passants dessus.

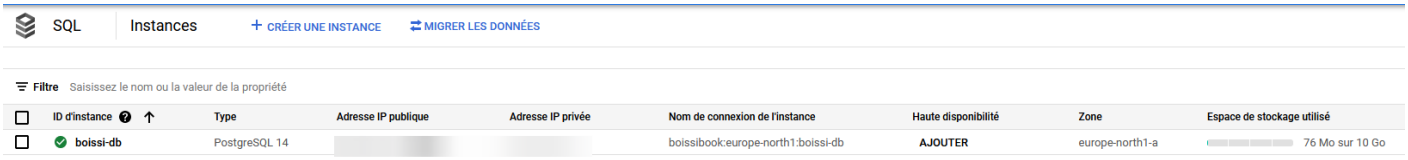


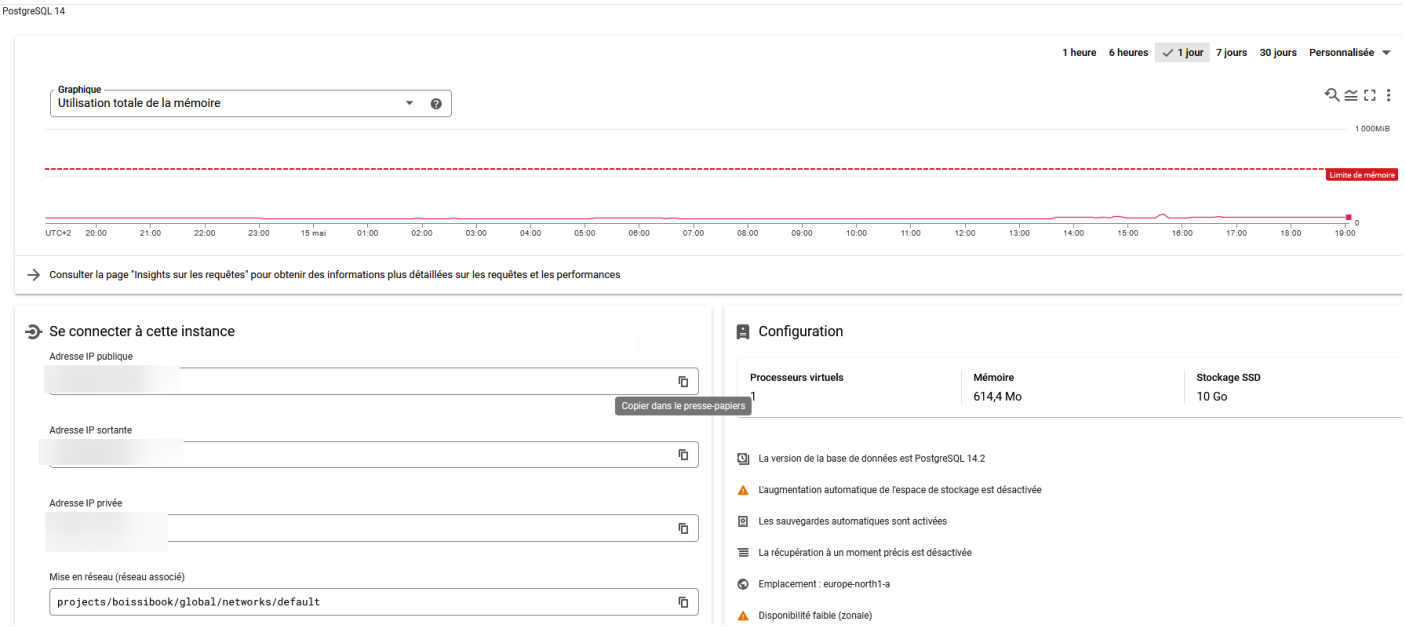
Création de l'instance

Une fois que tout est configuré à votre convenance, cliquez sur **Créer une instance** tout en bas



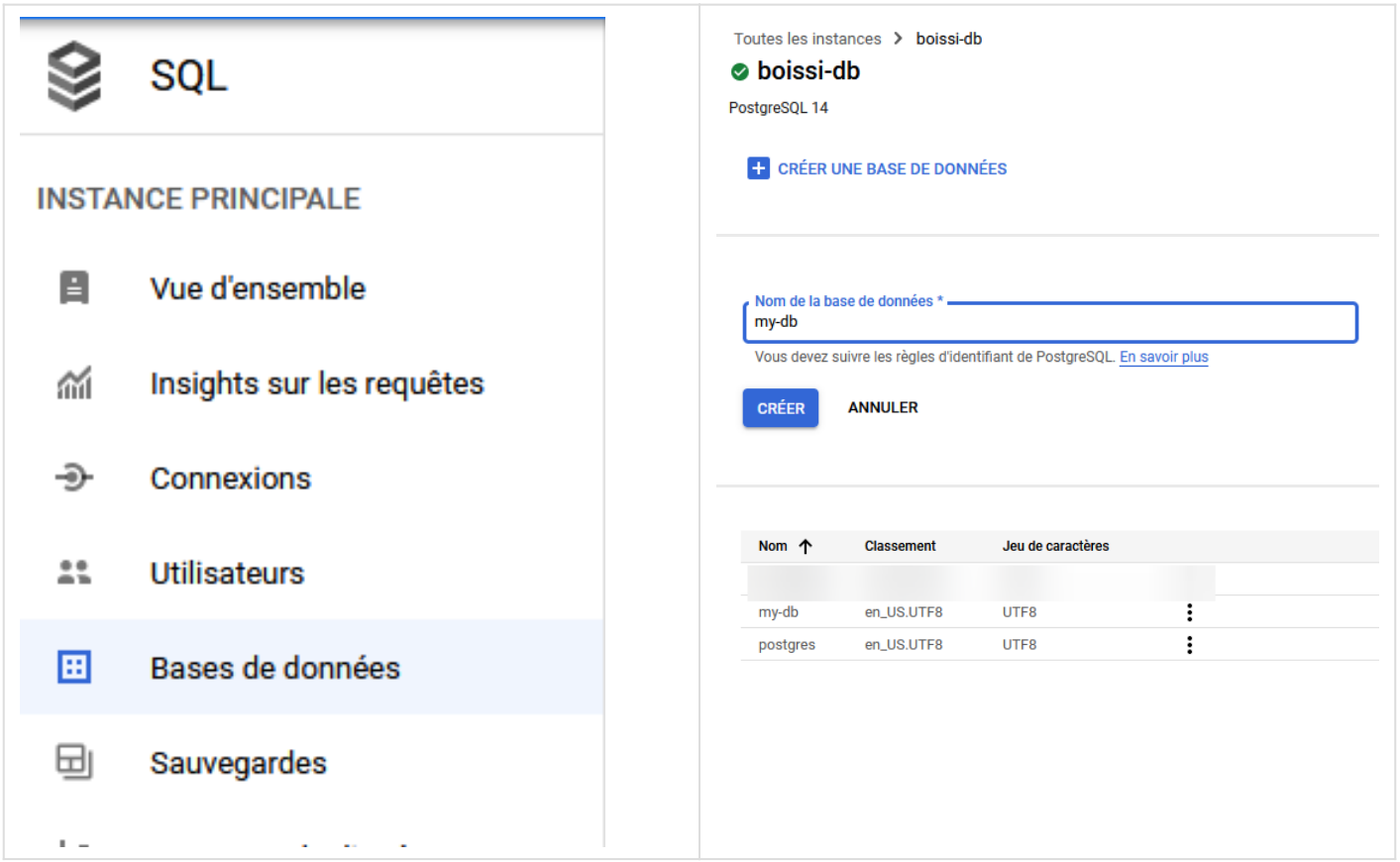
Après quelques minutes, vous devriez voir apparaitre votre instance avec tous les détails





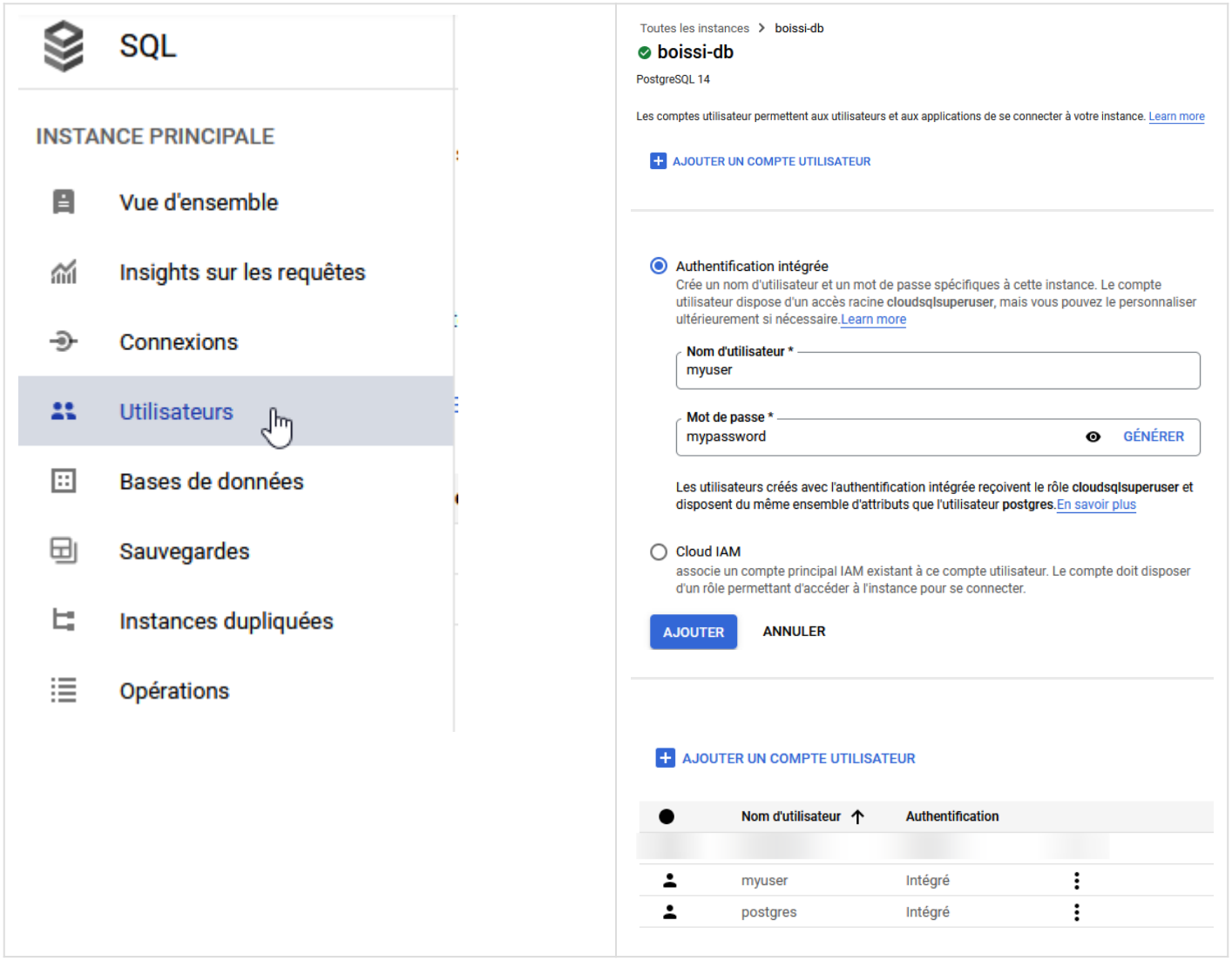
Création d'une base

Lorsque vous consultez votre instance, allez dans la section **Bases de données**, puis cliquez sur **Créer une base de données**.
Il ne vous reste plus qu'à lui donner un nom.



Ajout d'un utilisateur

Lorsque vous consultez votre instance, allez dans la section **Utilisateurs**, puis cliquez sur **Ajouter un compte utilisateur**.
Il ne vous reste plus qu'à lui donner un nom.



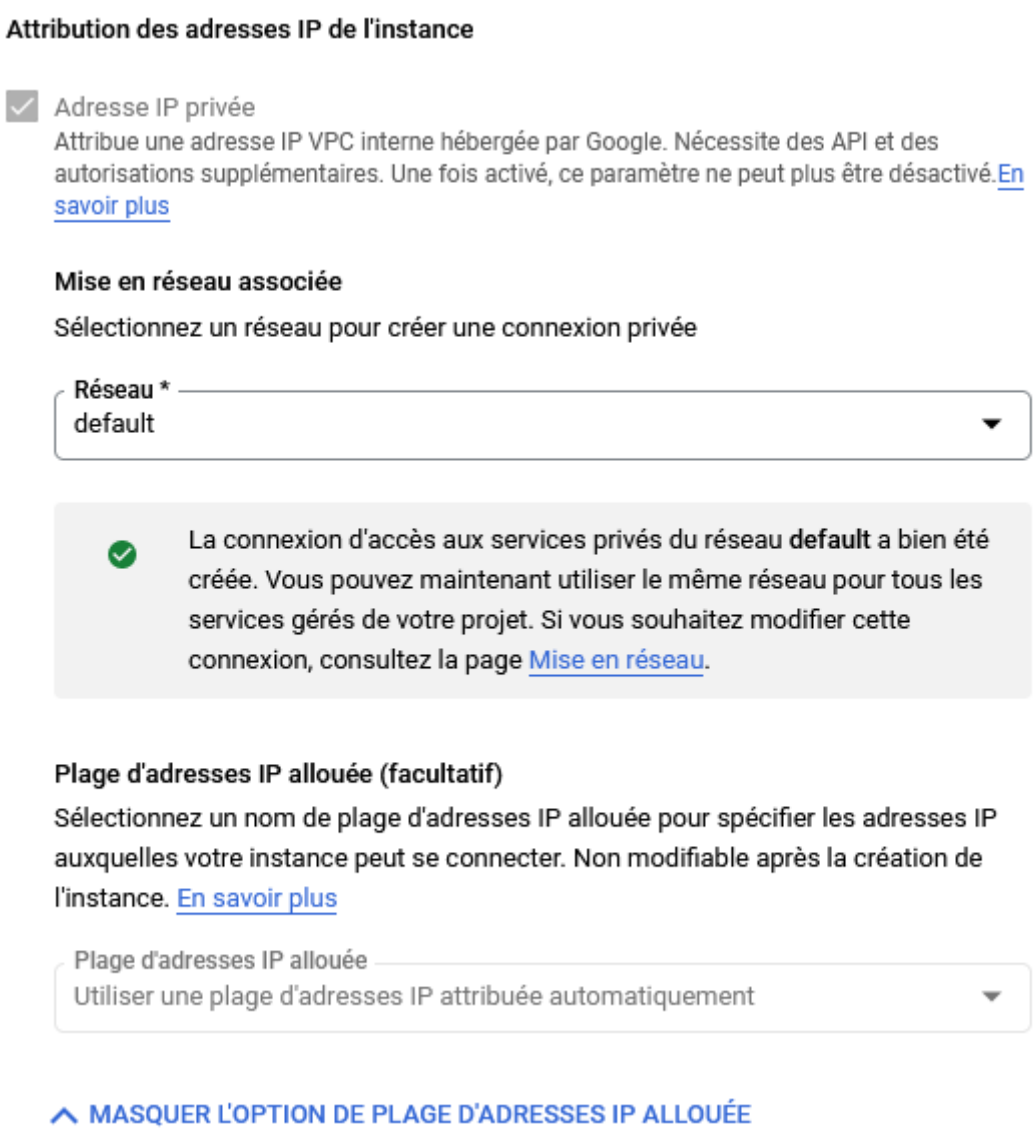
Configuration des accès

Lorsque vous consultez votre instance, allez dans la section **Connexions**.

Réseau privé

Comme dit précédemment, le réseau privé autorise uniquement une connexion sur l'adresse IP privé en interne pour les autres instances étant connecté sur le même réseau, dans cet exemple le

réseau **default**



> Une fois activé, le mode de connexion privé ne peut pas être desactivé

Réseau public

Votre base de données est accessible via l'adresse IP publique de votre instance. Elle refusera tout de même toute connexion entrante qui n'est pas autorisé explicitement par une plage IP.

Dans cet exemple, je n'autorise que deux adresses IP externes à se connecter à ma base (/32)

Adresse IP publique

Attribue une adresse IP externe accessible par Internet. Il est nécessaire d'utiliser un réseau autorisé ou le proxy Cloud SQL pour se connecter à cette instance.En savoir plus

Réseaux autorisés

Vous pouvez spécifier des plages CIDR pour permettre aux adresses IP de ces plages d'accéder à votre instance.En savoir plus

Home IP (10/32)

Modifier le réseau

Nom

N IP

Utilisez format CIDR

Réseau *

43/32

Exemple : 199.27.25.0/24

OK

AJOUTER UN RÉSEAU

CI / CD avec GKE et Cloud Build

Objectif

Nous allons voir comment configurer une intégration continue entièrement géré par **GCP** via le service **Cloud Build** de manière à gérer un déploiement continue sur ukjinn Cluster **Google Kubernetes Engine**.

Le workflow à configurer sera le suivant :

1. Push sur un dépôt distant (GitHub) sur une ou plusieurs branches définies
2. Construction d'une image docker
3. Mis à jour de l'image dans le registre géré par GCP
4. Provisioning des variables d'environnement dans nos fichiers de déploiements kubernetes
5. Mise à jour du déploiement Kubernetes sur notre cluster GKE

Pré-requis

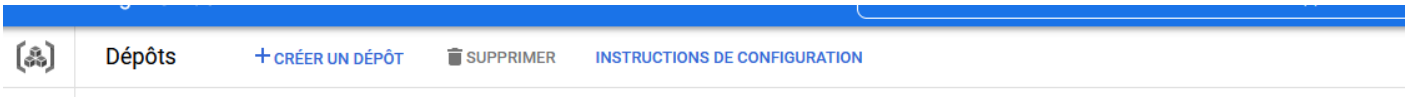
Pour effectuer ces étapes, vous devez impérativement avoir :

- Un dépôt Git (dans notre exemple sur GitHub) dont vous êtes propriétaire
- Un Dockerfile fonctionnel sur votre dépôt
- Un cluster GKE à disposition (Google Kubernetes Engine)

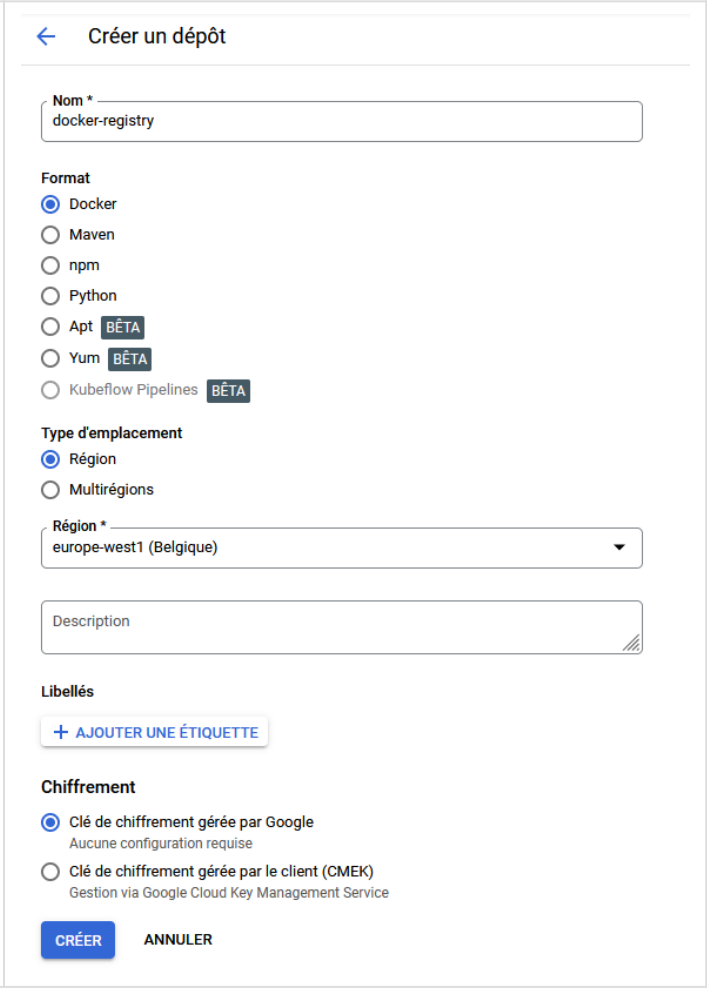
Ajout d'un registre docker sur GCP

Rendez-vous sur **Google Cloud Platform**, et dans la section **CI/CD > Artifact Registry**, puis sur **Créer un dépôt**





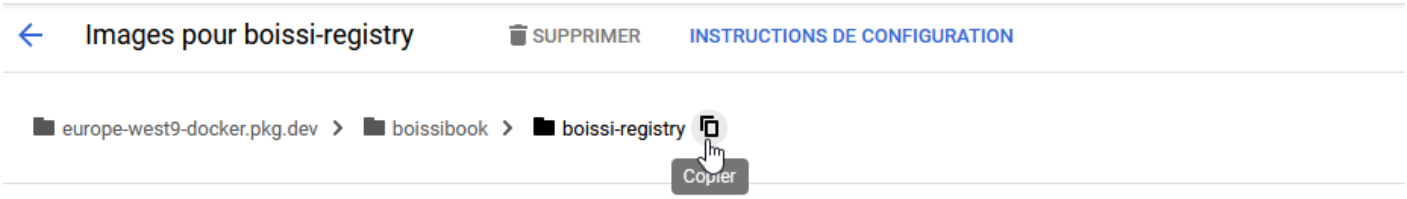
- Choisissez le type de dépôt **Docker**
- Choisissez une **région** à proximité de votre localisation
- **Créer**



Votre registre devrait apparaitre dans la liste après quelques minutes.

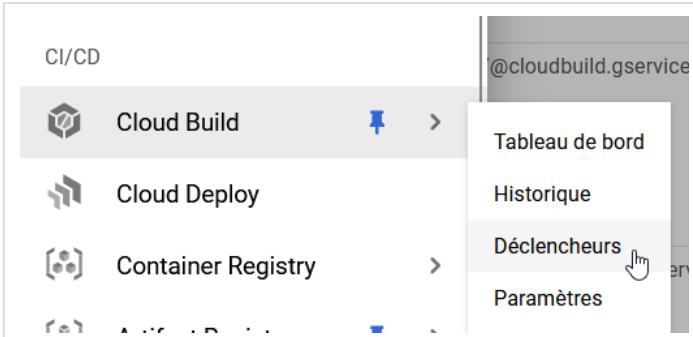


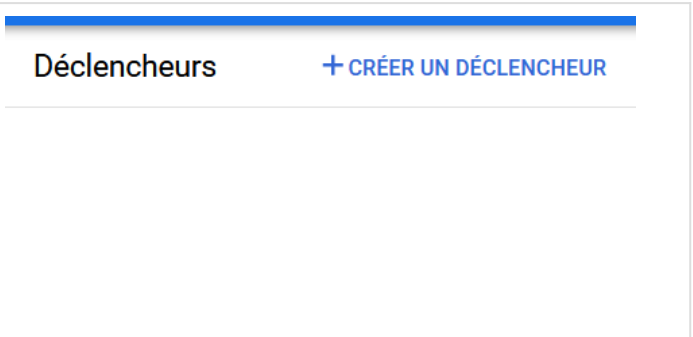
> Vous pouvez obtenir l'adresse de votre registre en allant dans les détails de ce dernier.



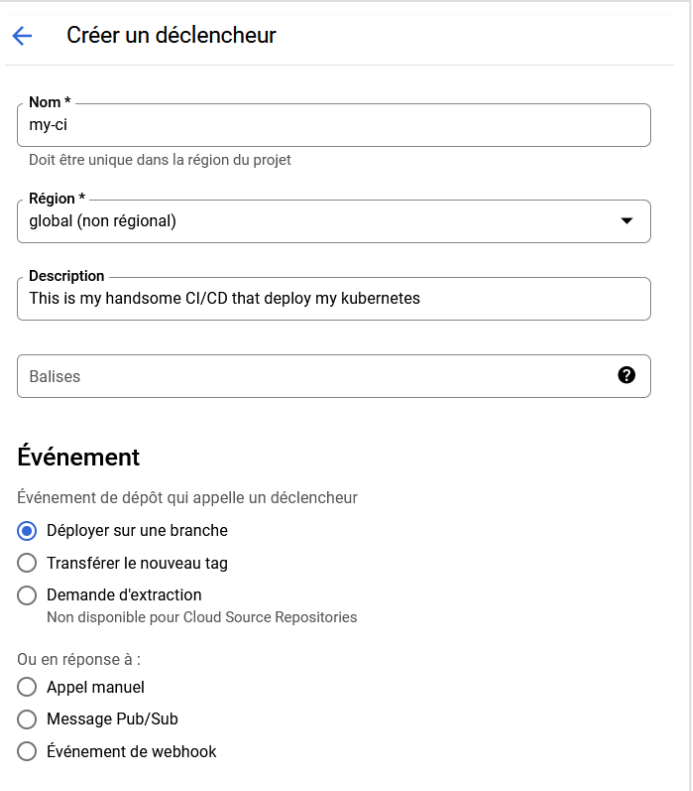
Création d'un déclencheur Cloud Build

Rendez-vous dans sur **GCP** dans la section **CI/CD > Cloud Build > Déclencheurs** et cliquez sur **Créer un déclencheur**.



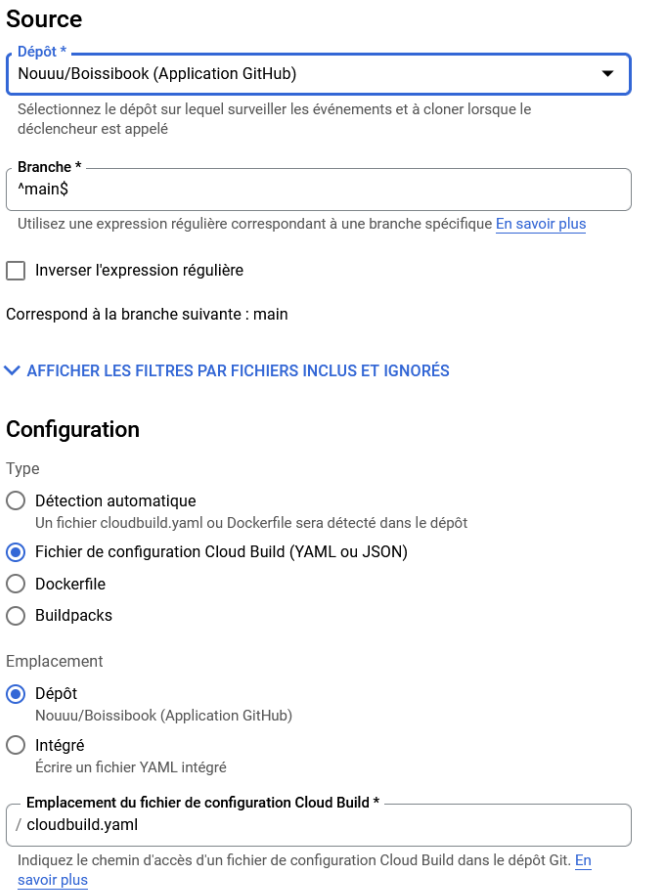


- Donnez un nom à votre déclencheur
- Choisissez une région (global, c'est très bien)
- Entrez une description
- Choisir le type de déclencheur. Déployer sur une branche signifie que le workflow se déclenchera lors d'un push sur une ou plusieurs branche(s) défini.



- Sélectionnez la source de votre dépôt. Si vous choisissez GitHub par exemple, il vous sera demandé de vous authentifier et d'autoriser **GCP** à accéder à vos dépôts.
  - Vous devez être propriétaire du dépôt pour l'ajouter.

- Choisissez la branche à cibler, vous pouvez utiliser des patterns RegEx.
- Sélectionnez dans **Configuration** fichier de configuration et laisser par défaut **cloudbuild.yml**.

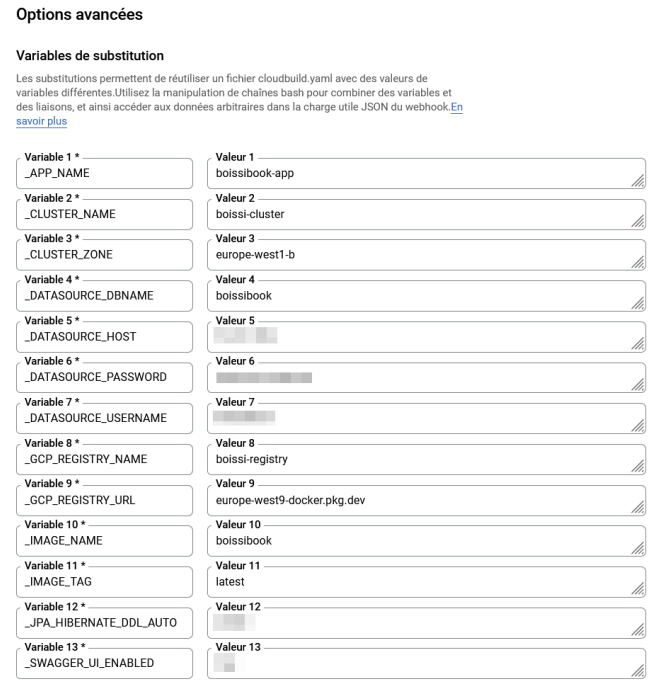


Dans les options avancées, vous pouvez ajouter des **Variables de substitution** qui serviront de variables d'environnements dans votre workflow.
Celles qui sont essentielles sont :

- **_APP_NAME** : Le nom de votre application à déployer sur kubernetes
- **_CLUSTER_NAME** : Le nom de votre cluster
- **_CLUSTER_ZONE** : La zone de votre cluster
- **_GCP_REGISTRY_NAME** : Le nom de votre registre docker créé plus haut

- **_GCP_REGISTRY_URL** : L'url de votre registre docker

- **_IMAGE_NAME** : Le nom que vous souhaitez donner à votre image docker
- **_IMAGE_TAG** : Le tag de votre image docker



Vous pouvez ensuite créer votre déclencheur.

Ajout et configuration d'un service utilisateur

Pour permettre à notre service **Cloud Build**, il faut lui créer un compte de service qui possédera les droits nécessaires pour ajouter une image dans notre registre et mettre à jour notre cluster Kubernetes.

Allez dans la section **Autres produits > IAM et adminstration > IAM**

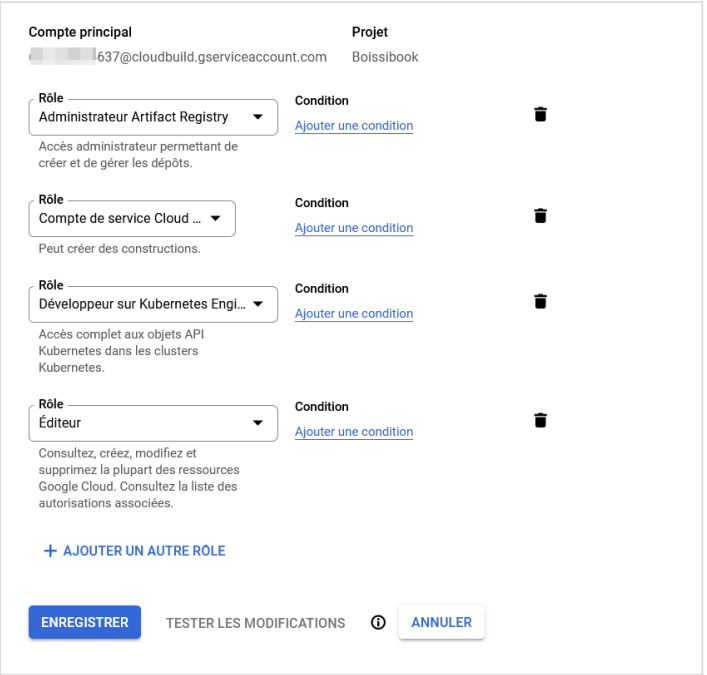
Vous devriez y trouver un utilisateur portant un nom avec ce format-là :

> ***<project-number>@cloudbuild.gserviceaccount.com***

Éditez ce rôle et mettez-lui les droits suivants :

- **Administrateur Artifact Registry** :
  Permettra d'ajouter des images à notre registre
- **Compte de service Cloud Build** :
  Droits de base permettant à ce rôle de lancer une exécution **Cloud Build**
- **Développeur sur Kubernetes Engine** :
  Donne un accès à notre cluster Kubernetes pour mettre à jour notre déploiement.

- **Éditeur** :
  Donne des accès de base à nos ressources **Google Cloud Platfom**



Fichiers de configuration sur le dépôt

Configuration Kubernetes

Sur votre dépôt, créez un dossier **k8s** et mettez-y deux fichiers :

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: <app-name>
  name: <app-name>
spec:
  replicas: 1
  selector:
    matchLabels:
      run: <app-name>
  template:
    metadata:
      labels:
        run: <app-name>
    spec:
      containers:
        - image: <gcp-registry-address>/<image-name>:<image-tag>
          imagePullPolicy: Always
          name: <app-name>
          env:
            - name: DATASOURCE_HOST
              value: "%_DATASOURCE_HOST%"
            - name: DATASOURCE_DBNAME
              value: "%_DATASOURCE_DBNAME%"
            - name: DATASOURCE_USERNAME
              value: "%_DATASOURCE_USERNAME%"
            - name: DATASOURCE_PASSWORD
              value: "%_DATASOURCE_PASSWORD%"
            - name: SWAGGER_UI_ENABLED
              value: "%_SWAGGER_UI_ENABLED%"
            - name: JPA_HIBERNATE_DDL_AUTO
              value: "%_JPA_HIBERNATE_DDL_AUTO%"
          ports:
```

```
            - containerPort: 8080
          resources:
            requests:
              cpu: "200m"
              memory: "200Mi"
            limits:
              cpu: "800m"
              memory: "400Mi"
```

service.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: <app-name>
spec:
  selector:
    run: <app-name>
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: LoadBalancer
```

> Le port 8080, les variables d'environnements ainsi que les ressources demandé sont propre à mon image docker, vous devrez adapter ces valeurs à la votre.

> Pour les variables d'environnements au format **%_ENV_VAR%**, vous remarquerez qu'elles ressemblent étrangements à calles configurés sur **GCP** dans notre déclancheur **Cloud Build** , nous verrons juste en dessous comment les substituer.

Ces fichiers représentent notre déploiement Kubernetes ainsi que le service permettant d'exposer son port via un **Load Balancer**.

Fichier workflow Cloud Build

À la racine de votre dépôt, créez le fichier **cloudbuild.yaml**, le fichier qui indiquera le workflow à suivre par **GCP Cloud Build**.

```
steps:
  #step 1
  - name: 'gcr.io/cloud-builders/docker'
    entrypoint: 'bash'
    args: [
      '-c',
      'docker pull $_GCP_REGISTRY_URL/$PROJECT_ID/$_GCP_REGISTRY_NAME/$_IMAGE_NAME:$_IMAGE_TAG
|| exit 0'
    ]
  #step 2
  - name: gcr.io/cloud-builders/docker
    args: [
      'build',
      '-t',
      '$_GCP_REGISTRY_URL/$PROJECT_ID/$_GCP_REGISTRY_NAME/$_IMAGE_NAME:$_IMAGE_TAG',
      '.'
    ]
    #step 3
  - name: gcr.io/cloud-builders/docker
    args: [
      'push',
      '$_GCP_REGISTRY_URL/$PROJECT_ID/$_GCP_REGISTRY_NAME/$_IMAGE_NAME:$_IMAGE_TAG',
    ]
  #step 4
  - name: 'gcr.io/cloud-builders/gcloud'
    entrypoint: bash
    args:
      - '-c'
      - |
        sed -i 's/%_DATASOURCE_HOST%/'${_DATASOURCE_HOST}'/g' k8s/*.yaml
        sed -i 's/%_DATASOURCE_DBNAME%/'${_DATASOURCE_DBNAME}'/g' k8s/*.yaml
        sed -i 's/%_DATASOURCE_USERNAME%/'${_DATASOURCE_USERNAME}'/g' k8s/*.yaml
        sed -i 's/%_DATASOURCE_PASSWORD%/'${_DATASOURCE_PASSWORD}'/g' k8s/*.yaml
        sed -i 's/%_SWAGGER_UI_ENABLED%/'${_SWAGGER_UI_ENABLED}'/g' k8s/*.yaml
        sed -i 's/%_JPA_HIBERNATE_DDL_AUTO%/'${_JPA_HIBERNATE_DDL_AUTO}'/g' k8s/*.yaml

  #step 5
  - name: 'gcr.io/cloud-builders/kubectl'
    args: [ 'apply', '-f', 'k8s/' ]
    env:
```

```
      - 'CLOUDSDK_COMPUTE_ZONE=$_CLUSTER_ZONE'
      - 'CLOUDSDK_CONTAINER_CLUSTER=$_CLUSTER_NAME'
  #step 6
  - name: 'gcr.io/cloud-builders/kubectl'
    args: [ 'rollout', 'restart', 'deployment/boissibook' ]
    env:
      - 'CLOUDSDK_COMPUTE_ZONE=$_CLUSTER_ZONE'
      - 'CLOUDSDK_CONTAINER_CLUSTER=$_CLUSTER_NAME'
images: [
  '$_GCP_REGISTRY_URL/$PROJECT_ID/$_GCP_REGISTRY_NAME/$_IMAGE_NAME:$_IMAGE_TAG'
]
options:
  logging: CLOUD_LOGGING_ONLY
```

Step 1
Nous essayons d'extraire la dernière image existante de l'application que nous essayons de construire, afin que notre construction soit plus rapide, car docker utilise les couches mises en cache des anciennes images pour créer de nouvelles images.
La raison de l'ajout de `||  exit 0` est au cas où l'extraction de l'image fixe renvoie une erreur (lors de l'exécution de cette version pour la première fois, il n'y aura pas d'image la plus récente à extraire du référentiel), l'ensemble du pipeline échouerait.
C'est pourquoi `||  exit 0` est ajouté pour ignorer et poursuivre la génération même si une erreur se produit à cette étape.

Step 2
Nous construisons l'image docker de notre application.

Step 3
Une fois notre image construite, nous la poussons dans notre registre avec le tag donné, afin que le registre soit à jour.

Step 4
C'est ici que nous remplaçons dans notre fichier **deployment.yaml** les fameuses variables **%_ENV_VAR%** par celles configurées dans **Cloud Build**.

Step 5
Nous appliquons tous les fichiers **yamls** de configuration qui existent dans le dossier **k8s/** de notre application.
Kubectl est un outil vraiment puissant et il créera ou mettra automatiquement à jour les configurations et ressources manquantes dans le cluster en fonction des fichiers **yamls** que vous avez fournis dans le dossier **k8s**.
<cluster-zone> est la zone de votre cluster kubernetes. Votre cluster peut être régional, mais si vous accédez au moteur Kubernetes sur Google, vous verrez le nom de zone par défaut de votre cluster. Vous devez l'ajouter, **<cluster-name>** est le nom de votre cluster que vous avez donné

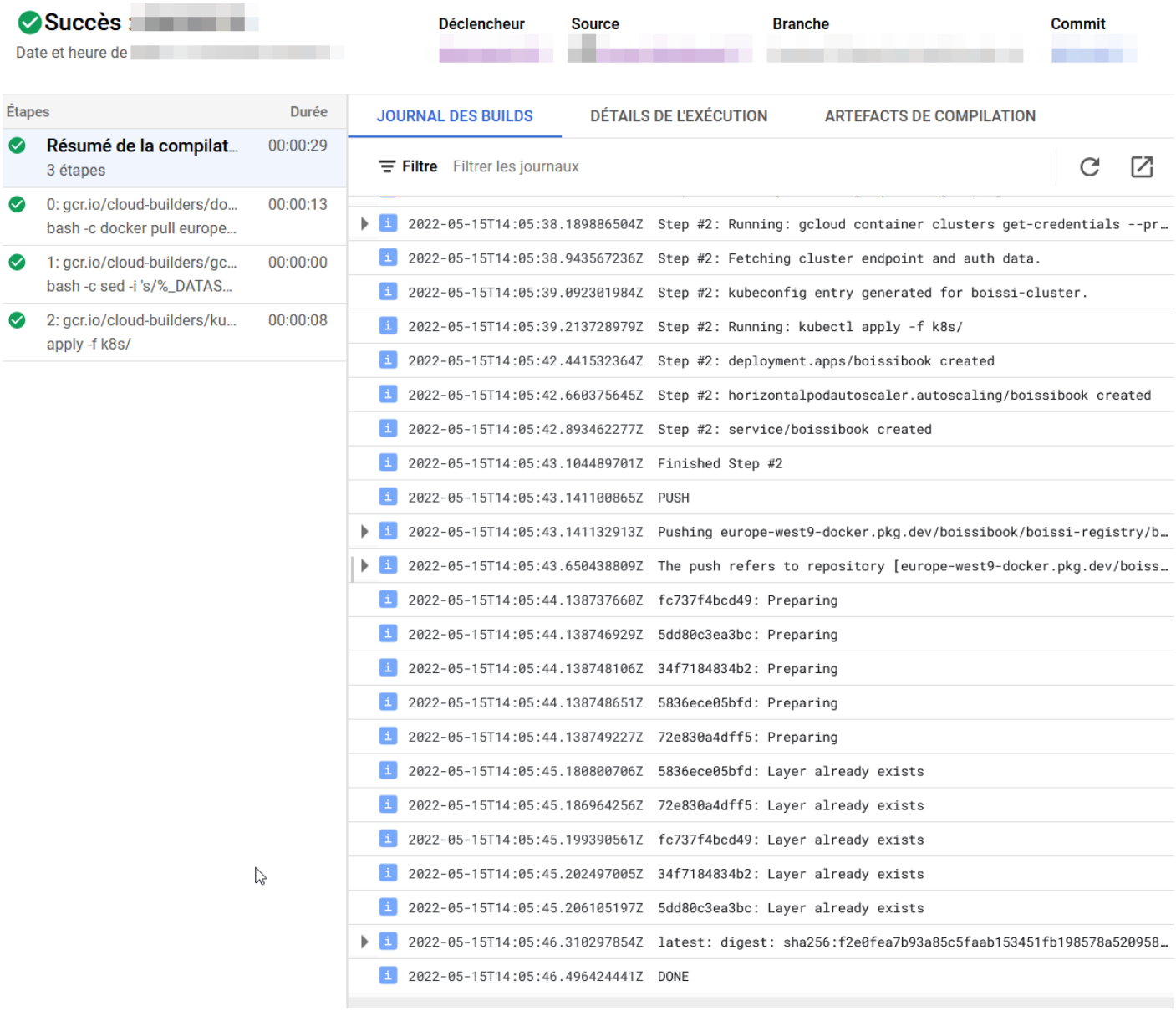
lors de la création du cluster.
Heureusement, nous les avons configurés dans nos variables d'environnements **Cloud Build**, pour nous éviter des mettre des informations trop précises dans ce fichier.

Step 6
Dans le cas où nos fichiers de déploiements ne sont pas mise à jour, mais que notre image oui, il faut que notre cluster mette à jour cette dernière pour l'utiliser.
Pour faire ça, nous redémarrons manuellement notre déploiement afin que grâce à notre configuration **imagePullPolicy: Always**, notre cluster télécharge la nouvelle image et redémarre nos **pods.**

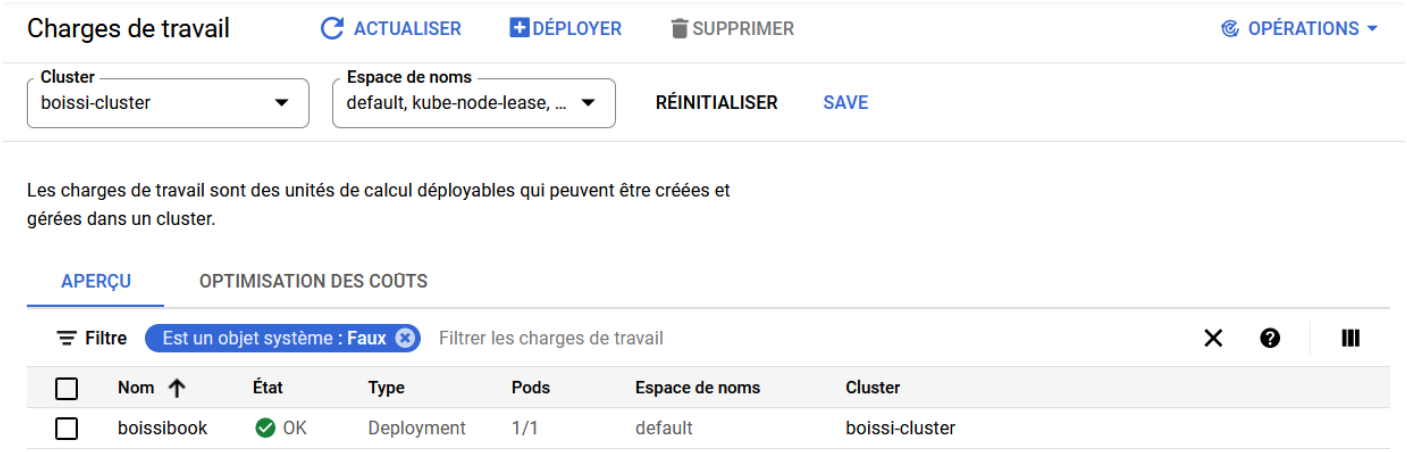
Lancement du workflow

Maintenant que tout est en place, il suffira de faire un changement et de faire un push sur la branche ciblé (dans notre exemple **main**), puis d'aller observer sur le tableau de bord **Cloud Build** le déroulement de votre workflow.





On peut aller vérifier sur notre cluster Kubernetes que notre application a bien été déployé.



Services et entrées

ACTUALISER CRÉER UN INGRESS SUPPRIMER

Cluster: boissi-cluster

Espace de noms: default, kube-node-lease, ...

RÉINITIALISER SAVE

SERVICES ENTRÉE

Les services sont des ensembles de pods avec un point de terminaison de réseau pouvant servir pour la détection et l'équilibrage de charge. Les Ingress sont des ensembles de règles qui permettent de router le trafic HTTP(S) externe vers les services.

Filtre | Est un objet système : Faux | Filtrer les services et les Ingress

Nom	État	Type	Points de terminaison	Pods	Espace de noms	Clusters
boissibook	OK	Équilibreur de charge externe	[redacted]:8080	1/1	default	boissi-cluster

