

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 un 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**

CI/CD

- Cloud Build
- Cloud Deploy
- Container Registry
- Artifact Registry**
- Dépôts sources

Dépôts
Paramètres



Dépôts

+ CRÉER UN DÉPÔT

SUPPRIMER

INSTRUCTIONS DE CONFIGURATION

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

← Créer un dépôt

Nom *
docker-registry

Format

- Docker
- Maven
- npm
- Python
- Apt **BÉTA**
- Yum **BÉTA**
- Kubeflow Pipelines **BÉTA**

Type d'emplacement

- Région
- Multirégions

Région *
europe-west1 (Belgique)

Description

Libellés

+ AJOUTER UNE ÉTIQUETTE

Chiffrement

- Clé de chiffrement gérée par Google
Aucune configuration requise
- Clé de chiffrement gérée par le client (CMEK)
Gestion via Google Cloud Key Management Service

CRÉER ANNULER

Votre registre devrait apparaître dans la liste après quelques minutes.

Dépôts

+ CRÉER UN DÉPÔT

SUPPRIMER

INSTRUCTIONS DE CONFIGURATION

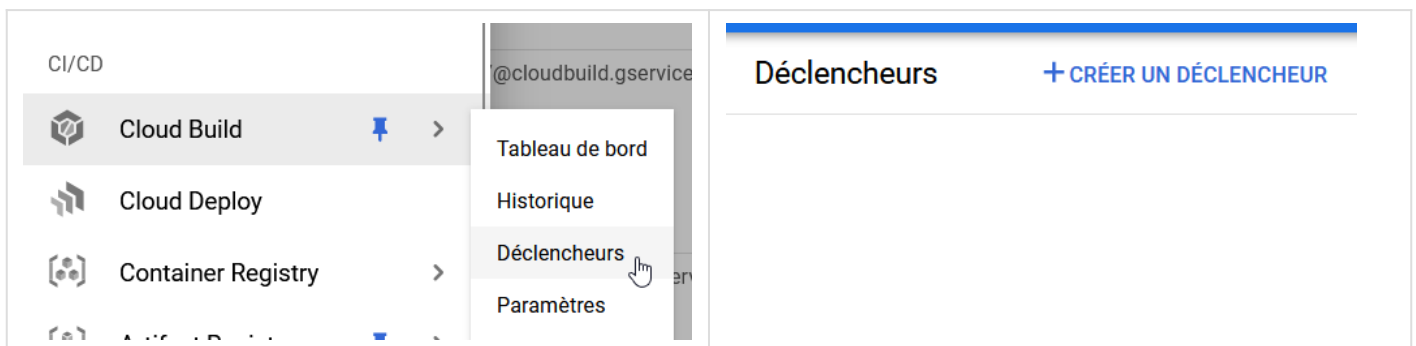
Filtre Saisissez le nom ou la valeur de la propriété

<input type="checkbox"/>	Nom ↑	Format	Zone	Description	Libellés	Stratégie de version ?	Chiffrement ?	Clé de chiffrement	Création	Mis à jour
<input type="checkbox"/>	boissi-registry	Docker	europe-west9 (Paris)				Clé gérée par Google	—	il y a 1 jour	il y a 3 heures

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.

← Créer un déclencheur

Nom *
my-ci
Doit être unique dans la région du projet

Région *
global (non régional)

Description
This is my handsome CI/CD that deploy my kubernetes

Balises ?

Événement

Événement de dépôt qui appelle un déclencheur

- Déployer sur une branche
- Transférer le nouveau tag
- Demande d'extraction
Non disponible pour Cloud Source Repositories

Ou en réponse à :

- Appel manuel
- Message Pub/Sub
- Événement de webhook

- 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 laissez par défaut **cloudbuild.yml**.

Source

Dépôt *

Nouuu/Boissibook (Application GitHub)

Sélectionnez le dépôt sur lequel surveiller les événements et à cloner lorsque le déclencheur est appelé

Branche *

^main\$

Utilisez une expression régulière correspondant à une branche spécifique [En savoir plus](#)

Inverser l'expression régulière

Correspond à la branche suivante : main

✓ AFFICHER LES FILTRES PAR FICHIERS INCLUS ET IGNORÉS

Configuration

Type

Détection automatique

Un fichier cloudbuild.yaml ou Dockerfile sera détecté dans le dépôt

Fichier de configuration Cloud Build (YAML ou JSON)

Dockerfile

Buildpacks

Emplacement

Dépôt

Nouuu/Boissibook (Application GitHub)

Intégré

Écrire un fichier YAML intégré

Emplacement du fichier de configuration Cloud Build *

/ cloudbuild.yaml

Indiquez le chemin d'accès d'un fichier de configuration Cloud Build dans le dépôt Git. [En savoir plus](#)

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

Options avancées

Variables de substitution

Les substitutions permettent de réutiliser un fichier cloudbuild.yaml avec des valeurs de variables différentes. Utilisez la manipulation de chaînes bash pour combiner des variables et des liaisons, et ainsi accéder aux données arbitraires dans la charge utile JSON du webhook. [En savoir plus](#)

Variable 1 * _APP_NAME	Valeur 1 boissibook-app
Variable 2 * _CLUSTER_NAME	Valeur 2 boissi-cluster
Variable 3 * _CLUSTER_ZONE	Valeur 3 europe-west1-b
Variable 4 * _DATASOURCE_DBNAME	Valeur 4 boissibook
Variable 5 * _DATASOURCE_HOST	Valeur 5 [REDACTED]
Variable 6 * _DATASOURCE_PASSWORD	Valeur 6 [REDACTED]
Variable 7 * _DATASOURCE_USERNAME	Valeur 7 [REDACTED]
Variable 8 * _GCP_REGISTRY_NAME	Valeur 8 boissi-registry
Variable 9 * _GCP_REGISTRY_URL	Valeur 9 europe-west1-docker.pkg.dev
Variable 10 * _IMAGE_NAME	Valeur 10 boissibook
Variable 11 * _IMAGE_TAG	Valeur 11 latest
Variable 12 * _JPA_HIBERNATE_DDL_AUTO	Valeur 12 [REDACTED]
Variable 13 * _SWAGGER_ULENABLED	Valeur 13 [REDACTED]

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 administration > 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 :

<ul style="list-style-type: none">• 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	<div><p>Compte principal 637@cloudbuild.gserviceaccount.com Projet Boissibook</p><table><tr><td>Rôle</td><td>Condition</td><td></td></tr><tr><td>Administrateur Artifact Registry</td><td>Ajouter une condition</td><td></td></tr><tr><td colspan="3">Accès administrateur permettant de créer et de gérer les dépôts.</td></tr><tr><td>Rôle</td><td>Condition</td><td></td></tr><tr><td>Compte de service Cloud ...</td><td>Ajouter une condition</td><td></td></tr><tr><td colspan="3">Peut créer des constructions.</td></tr><tr><td>Rôle</td><td>Condition</td><td></td></tr><tr><td>Développeur sur Kubernetes Engi...</td><td>Ajouter une condition</td><td></td></tr><tr><td colspan="3">Accès complet aux objets API Kubernetes dans les clusters Kubernetes.</td></tr><tr><td>Rôle</td><td>Condition</td><td></td></tr><tr><td>Éditeur</td><td>Ajouter une condition</td><td></td></tr><tr><td colspan="3">Consultez, créez, modifiez et supprimez la plupart des ressources Google Cloud. Consultez la liste des autorisations associées.</td></tr></table><p>+ AJOUTER UN AUTRE RÔLE</p><p>ENREGISTRER TESTER LES MODIFICATIONS ⓘ ANNULER</p></div>	Rôle	Condition		Administrateur Artifact Registry	Ajouter une condition		Accès administrateur permettant de créer et de gérer les dépôts.			Rôle	Condition		Compte de service Cloud ...	Ajouter une condition		Peut créer des constructions.			Rôle	Condition		Développeur sur Kubernetes Engi...	Ajouter une condition		Accès complet aux objets API Kubernetes dans les clusters Kubernetes.			Rôle	Condition		Éditeur	Ajouter une condition		Consultez, créez, modifiez et supprimez la plupart des ressources Google Cloud. Consultez la liste des autorisations associées.		
Rôle	Condition																																				
Administrateur Artifact Registry	Ajouter une condition																																				
Accès administrateur permettant de créer et de gérer les dépôts.																																					
Rôle	Condition																																				
Compte de service Cloud ...	Ajouter une condition																																				
Peut créer des constructions.																																					
Rôle	Condition																																				
Développeur sur Kubernetes Engi...	Ajouter une condition																																				
Accès complet aux objets API Kubernetes dans les clusters Kubernetes.																																					
Rôle	Condition																																				
Éditeur	Ajouter une condition																																				
Consultez, créez, modifiez et supprimez la plupart des ressources Google Cloud. Consultez la liste des autorisations associées.																																					

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 étrangement à celles configurés sur **GCP** dans notre déclencheur **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.

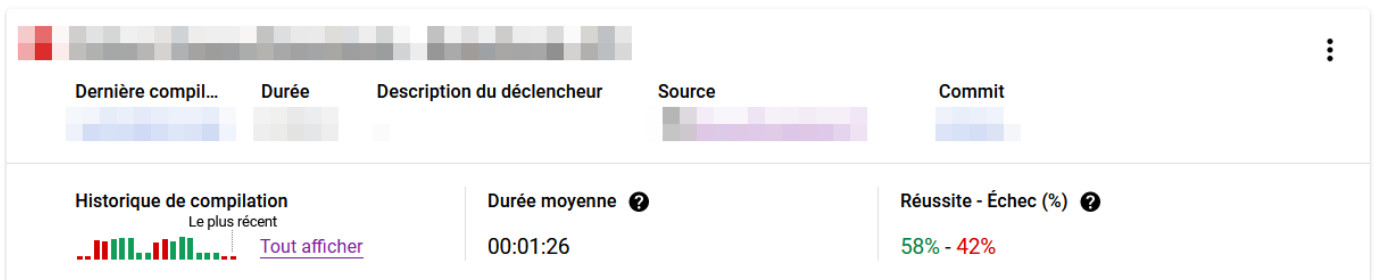
Step 6

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 ciblée (dans notre exemple **main**), puis d'aller observer sur le tableau de bord **Cloud Build** le déroulement de votre workflow.



Succès

Date et heure de

Déclencheur

Source

Branche

Commit

Étapes	Durée	JOURNAL DES BUILDS	DÉTAILS DE L'EXÉCUTION	ARTEFACTS DE COMPILATION
✓ Résumé de la compilat... 3 étapes	00:00:29	Filtre Filtrer les journaux ↻ 🔗		
✓ 0: gcr.io/cloud-builders/do... bash -c docker pull europe...	00:00:13	▶ i 2022-05-15T14:05:38.189886504Z Step #2: Running: gcloud container clusters get-credentials --pr...		
✓ 1: gcr.io/cloud-builders/gc... bash -c sed -i 's/%_DATAS...	00:00:00	i 2022-05-15T14:05:38.943567236Z Step #2: Fetching cluster endpoint and auth data.		
✓ 2: gcr.io/cloud-builders/ku... apply -f k8s/	00:00:08	i 2022-05-15T14:05:39.092301984Z Step #2: kubeconfig entry generated for boissi-cluster.		
		i 2022-05-15T14:05:39.213728979Z Step #2: Running: kubectl apply -f k8s/		
		i 2022-05-15T14:05:42.441532364Z Step #2: deployment.apps/boissibook created		
		i 2022-05-15T14:05:42.660375645Z Step #2: horizontalpodautoscaler.autoscaling/boissibook created		
		i 2022-05-15T14:05:42.893462277Z Step #2: service/boissibook created		
		i 2022-05-15T14:05:43.104489701Z Finished Step #2		
		i 2022-05-15T14:05:43.141100865Z PUSH		
		▶ i 2022-05-15T14:05:43.141132913Z Pushing europe-west9-docker.pkg.dev/boissibook/boissi-registry/b...		
		▶ i 2022-05-15T14:05:43.650438809Z The push refers to repository [europe-west9-docker.pkg.dev/boiss...		
		i 2022-05-15T14:05:44.138737660Z fc737f4bcd49: Preparing		
		i 2022-05-15T14:05:44.138746929Z 5dd80c3ea3bc: Preparing		
		i 2022-05-15T14:05:44.138748106Z 34f7184834b2: Preparing		
		i 2022-05-15T14:05:44.138748651Z 5836ece05bfd: Preparing		
		i 2022-05-15T14:05:44.138749227Z 72e830a4dff5: Preparing		
		i 2022-05-15T14:05:45.180800706Z 5836ece05bfd: Layer already exists		
		i 2022-05-15T14:05:45.186964256Z 72e830a4dff5: Layer already exists		
		i 2022-05-15T14:05:45.199390561Z fc737f4bcd49: Layer already exists		
		i 2022-05-15T14:05:45.202497005Z 34f7184834b2: Layer already exists		
		i 2022-05-15T14:05:45.206105197Z 5dd80c3ea3bc: Layer already exists		
		▶ i 2022-05-15T14:05:46.310297854Z latest: digest: sha256:f2e0fea7b93a85c5faab153451fb198578a520958...		
		i 2022-05-15T14:05:46.496424441Z DONE		

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

Charges de travail ↻ ACTUALISER + DÉPLOYER 🗑 SUPPRIMER 🔗 OPÉRATIONS

Cluster: boissi-cluster ▼ Espace de noms: default, kube-node-lease, ... ▼ RÉINITIALISER SAVE

Les charges de travail sont des unités de calcul déployables qui peuvent être créées et gérées dans un cluster.

APERÇU OPTIMISATION DES COÛTS

Filtre Est un objet système : Faux Filtrer les charges de travail ✕ ? ☰

<input type="checkbox"/>	Nom ↑	État	Type	Pods	Espace de noms	Cluster
<input type="checkbox"/>	boissibook	✓ OK	Deployment	1/1	default	boissi-cluster

Services et entrées

[ACTUALISER](#)[+ CRÉER UN INGRESS](#)[SUPPRIMER](#)

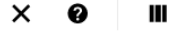
Cluster
boissi-cluster

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



<input type="checkbox"/>	Nom ↑	État	Type	Points de terminaison	Pods	Espace de noms	Clusters
<input type="checkbox"/>	boissibook	OK	Équilibreur de charge externe	:8080	1/1	default	boissi-cluster



Revision #5

Created 2022-05-15 17:32:53 UTC by Noé Larrieu-Lacoste

Updated 2022-05-30 09:30:33 UTC by Noé Larrieu-Lacoste