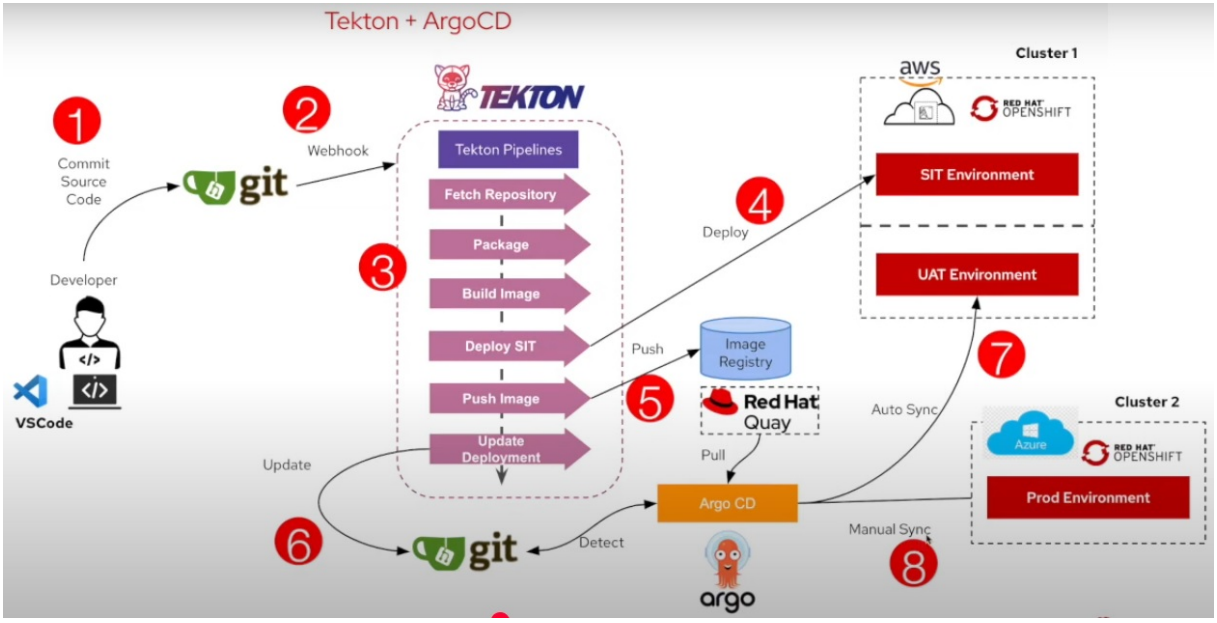
15 L'opérateur ARGOCD

15.1 L'installation de l'opérateur ArgoCD

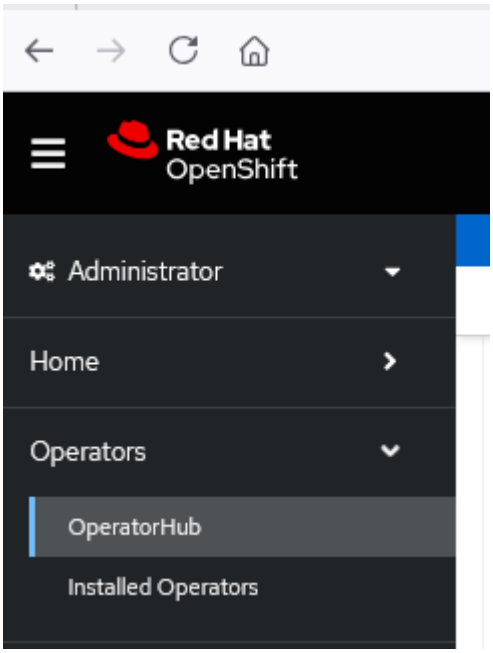
ArgoCD est une application qui permet de déployer d'autres applications.

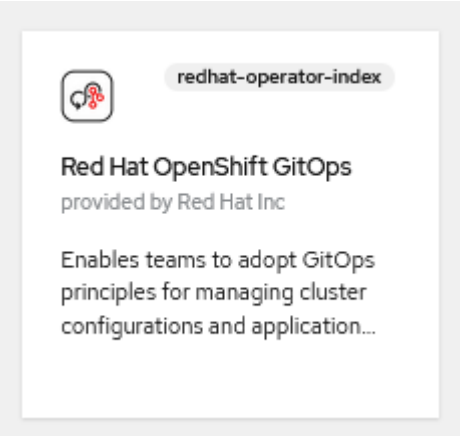
ArgoCD permet de rendre les déploiements plus efficace avec différents mécanismes dont nous n'aborderons qu'une petite partie . Il peut s'intégrer avec Tekton au niveau du dépôt git de déploiement des applications (yaml) et au niveau du registre privée REDHAT Quay. Il fournit à la production l'applicatif

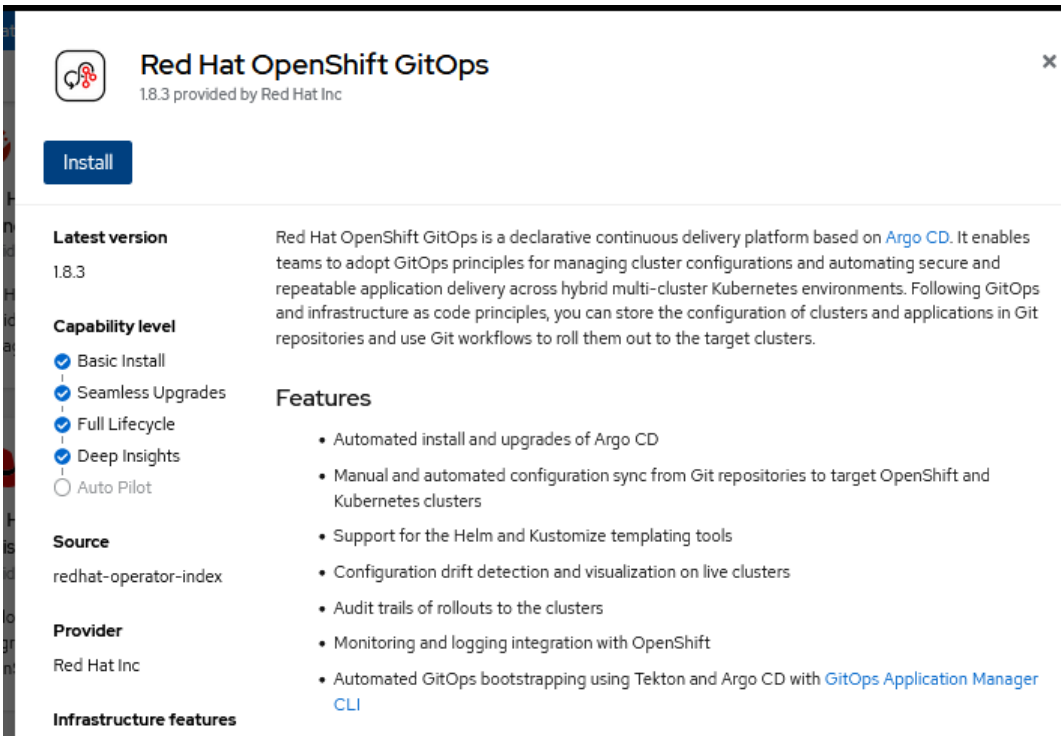


Installation de ARGOCD par les opérateurs

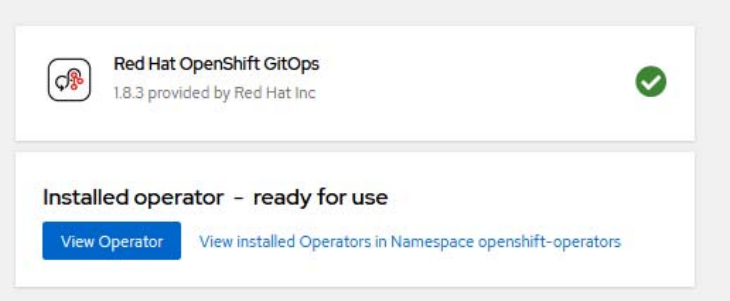
La première chose est d'installer l'opérateur baptisé RED HAT OPENSHIFT GITOPS



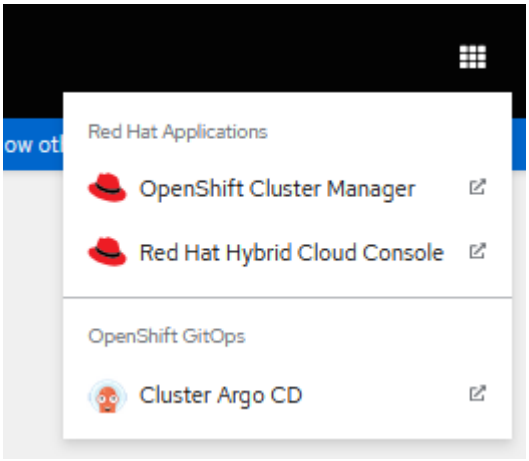




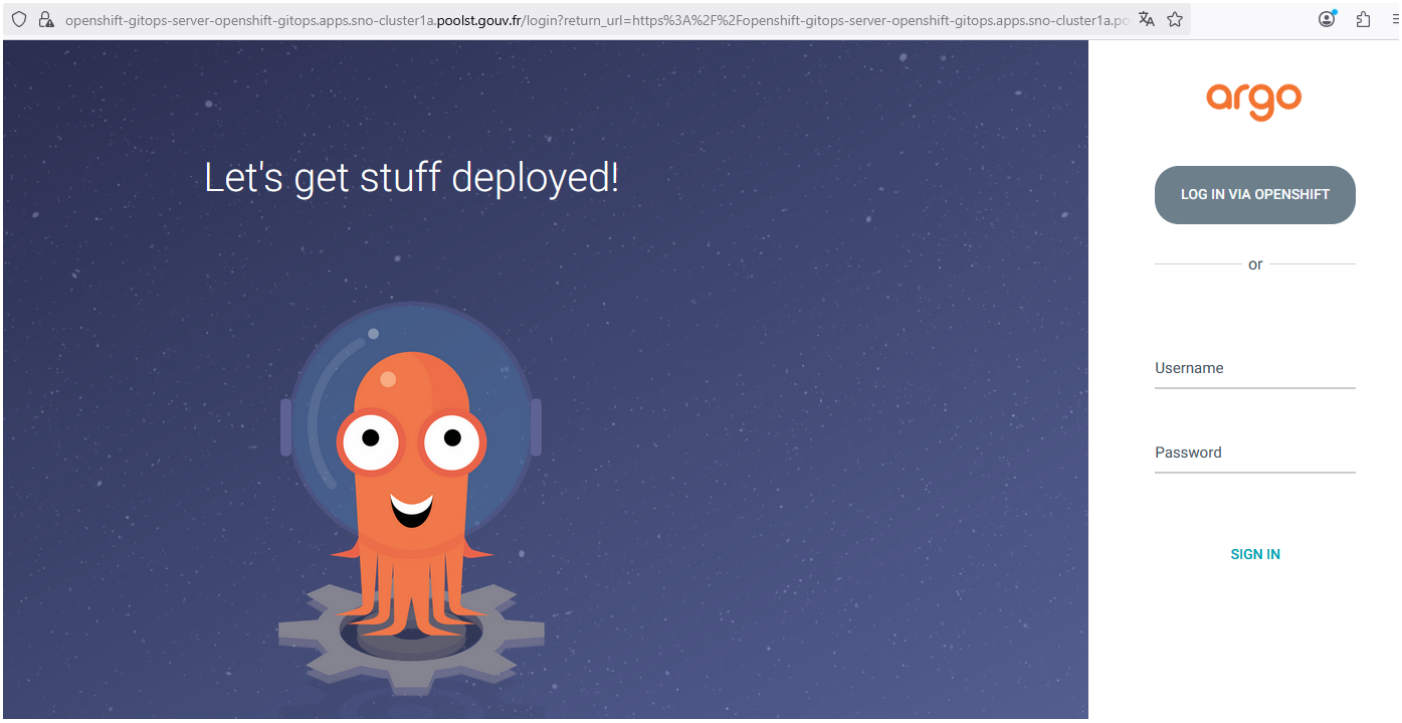
Cliquez sur install



L’opérateur est installé et cliquez sur la grille en haut dans la barre d’openshift



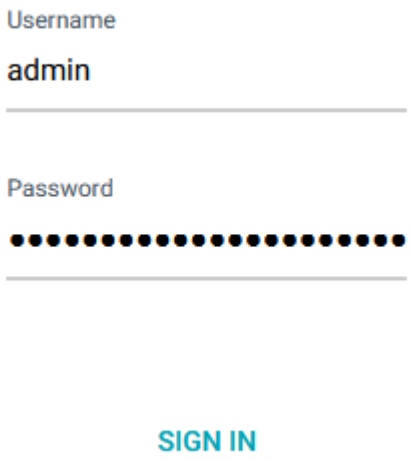
Il faut alors rentrer le login admin et le password qu'il faut déterminer



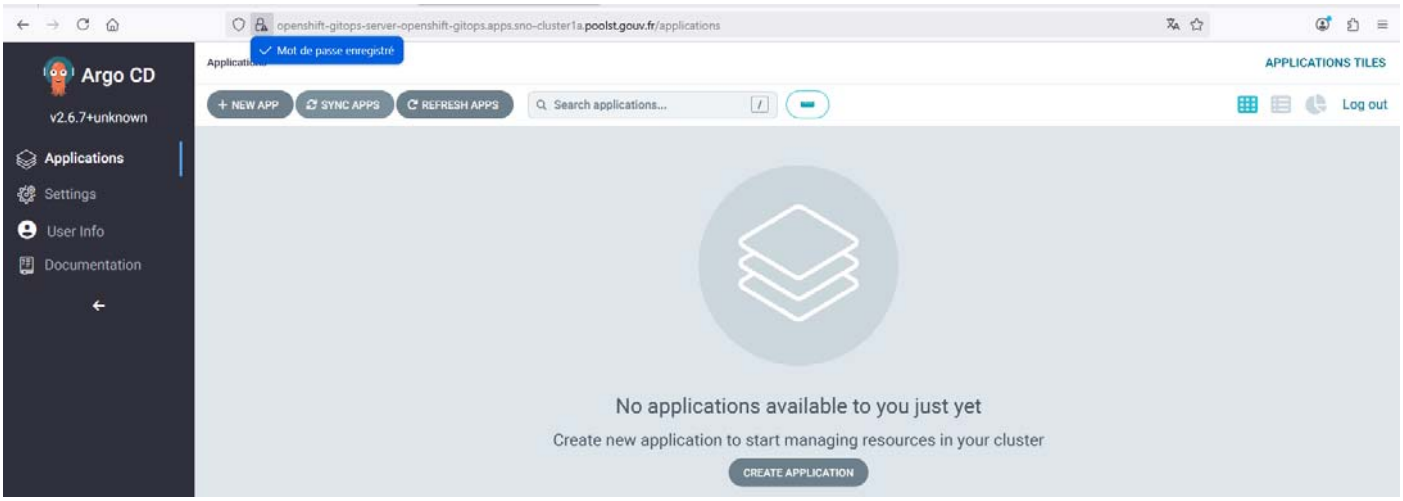
Le login par défaut est admin, on peut récupérer le mot de passe en tapant la commande suivante:

```
oc get secret/openshift-gitops-cluster -n openshift-gitops -o jsonpath='{.data.admin\.password}' | base64 -d
```

le password extrait est : 5UnYsFg7QkA8mwq3JNIXP6ZlKTBvaihd



Argo CD est presque prêt à fonctionner il faut lui donner des droits pour le cluster



ArgoCD a besoin de manipuler le cluster. Pour cela, il doit avoir les droits nécessaires :

```
kubectl create clusterrolebinding argocd-admin-binding \
 --clusterrole=cluster-admin \
 --serviceaccount=openshift-gitops:openshift-gitops-argocd-application-controller

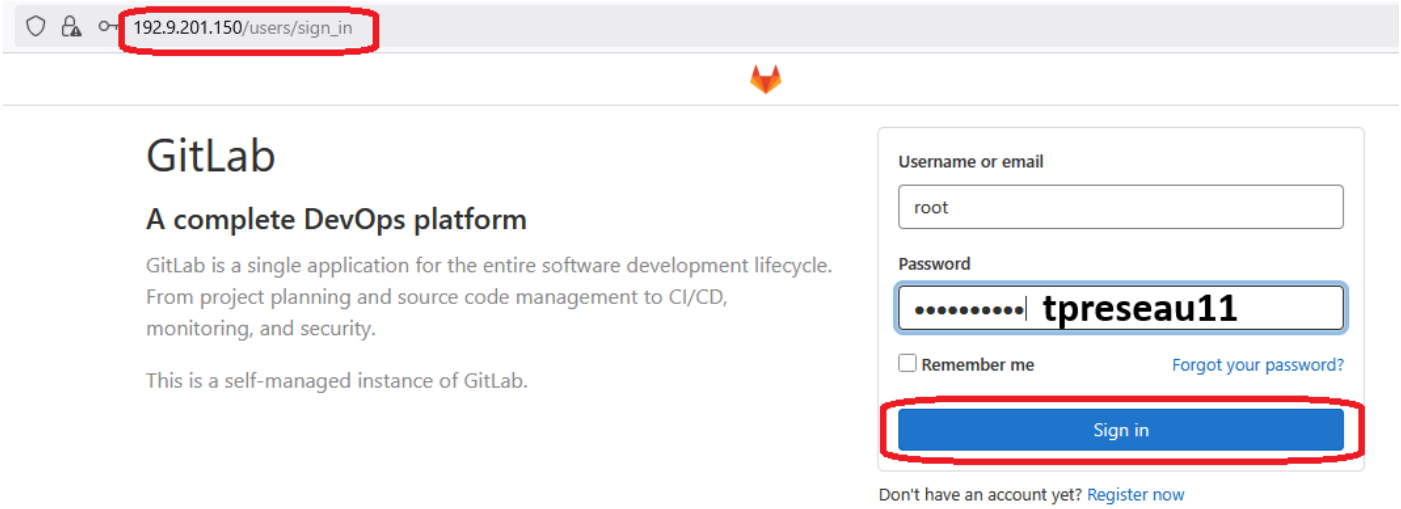
kubectl create clusterrolebinding argocd-server-admin-binding \
 --clusterrole=cluster-admin \
  --serviceaccount=openshift-gitops:openshift-gitops-argocd-server
```

On va donc déployer notre première application sur ARGOCD dans un dépot gitlab

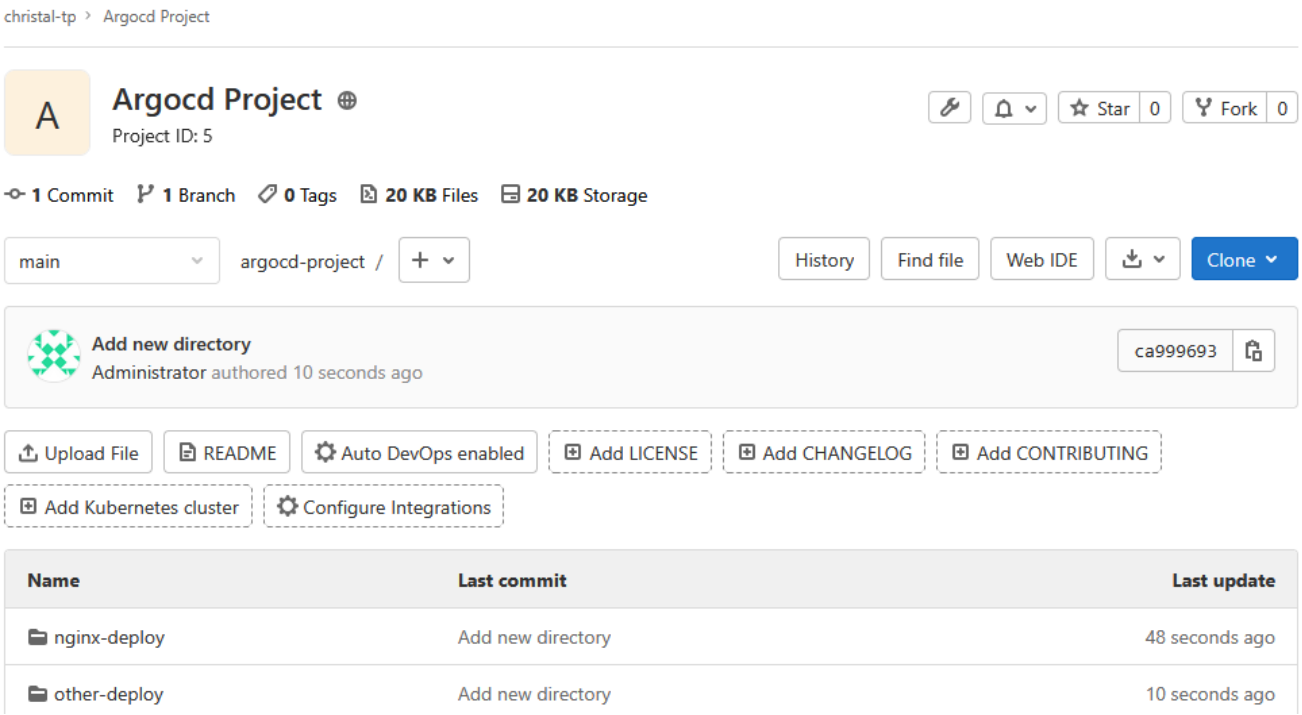
15.2 Notre application sur ARGOCD

Les dépôts GitLab n'ont pas besoin d'être créé d'une façon particulière pour être reconnus par ArgoCD. Pour pouvoir en utiliser un pour ArgoCD, on va utiliser le repo suivant:

Se positionner sur gitlab dans le dépôt



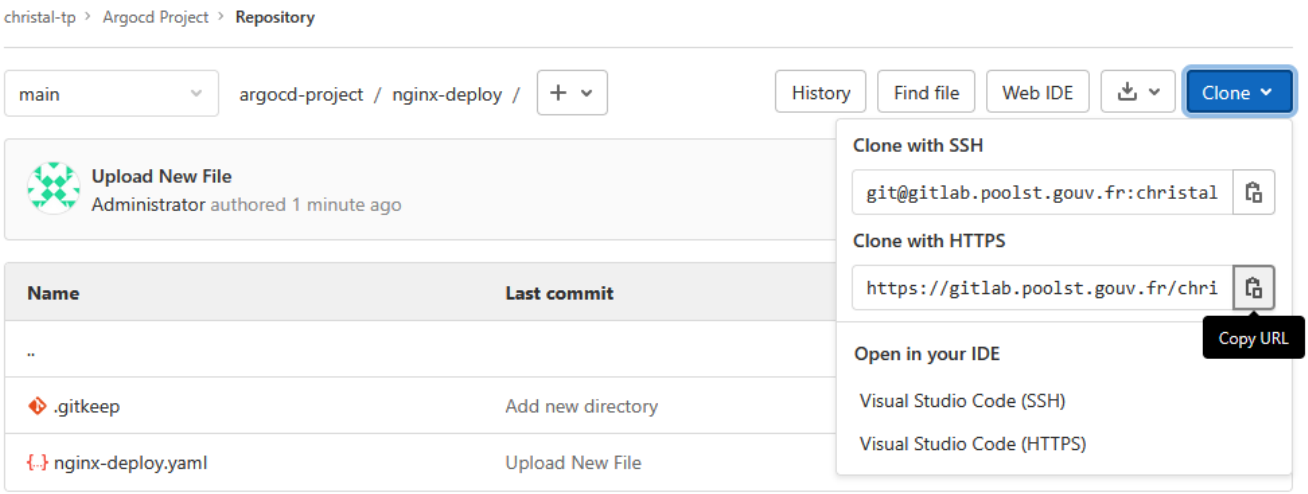
Allez dans le projet Argocd project



Puis le répertoire nginx-deploy

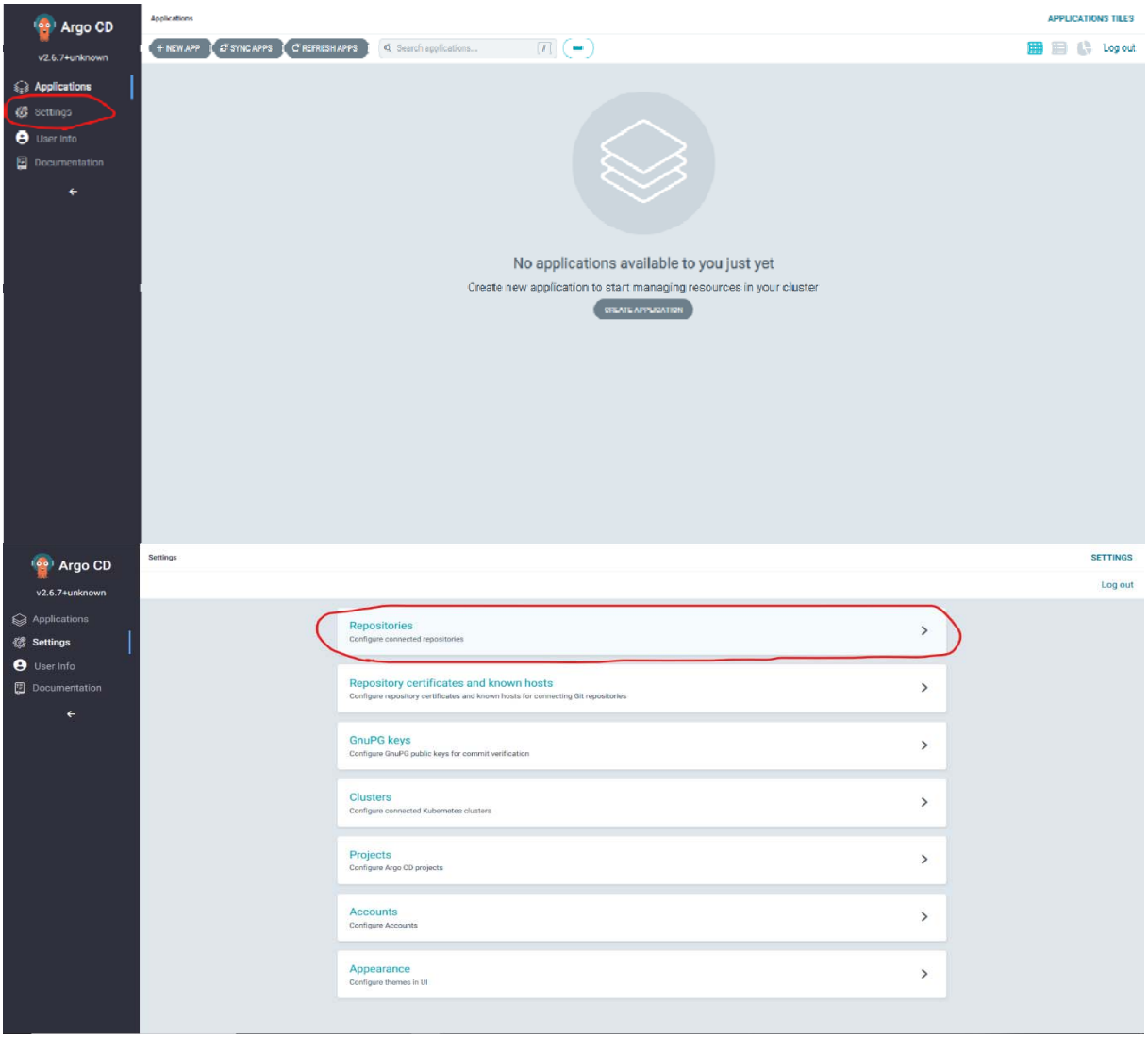
Il faut dans un premier temps récupérer le lien du dépôt correspondant à la façon dont on souhaite se connecter à celui-ci avec ArgoCD (ici HTTPS) :

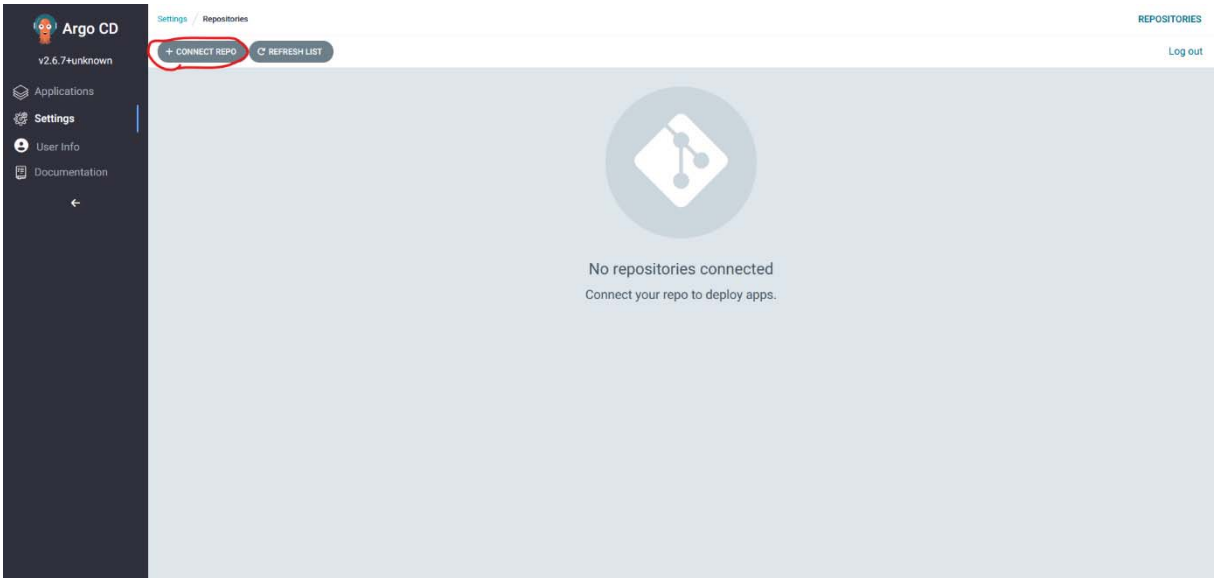
Afin qu'ArgoCD puisse accéder au dépôt, il est possible de lui donner le lien de celui-ci depuis l'interface web d'ArgoCD

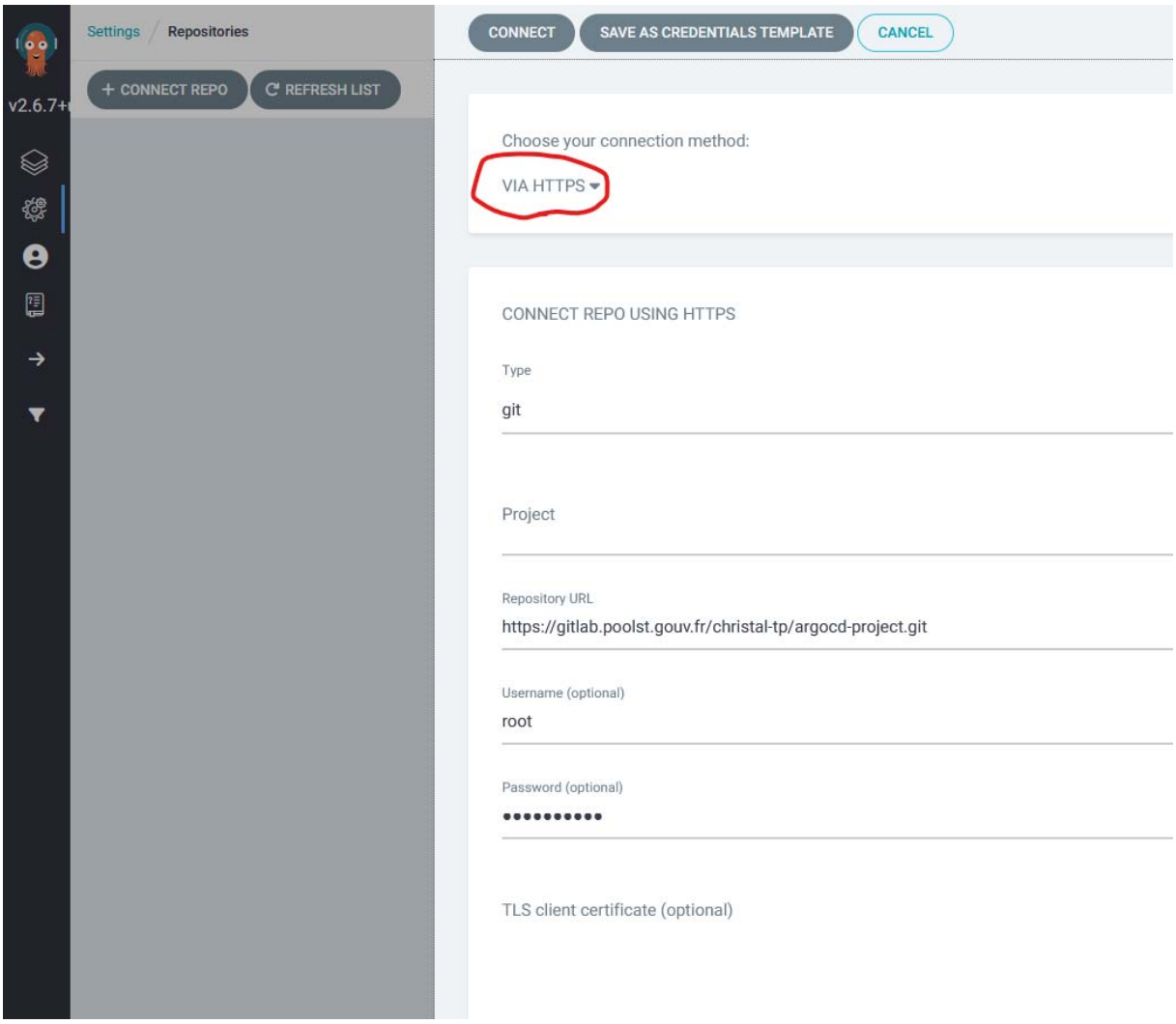


15.2.1 L'application par webcli console

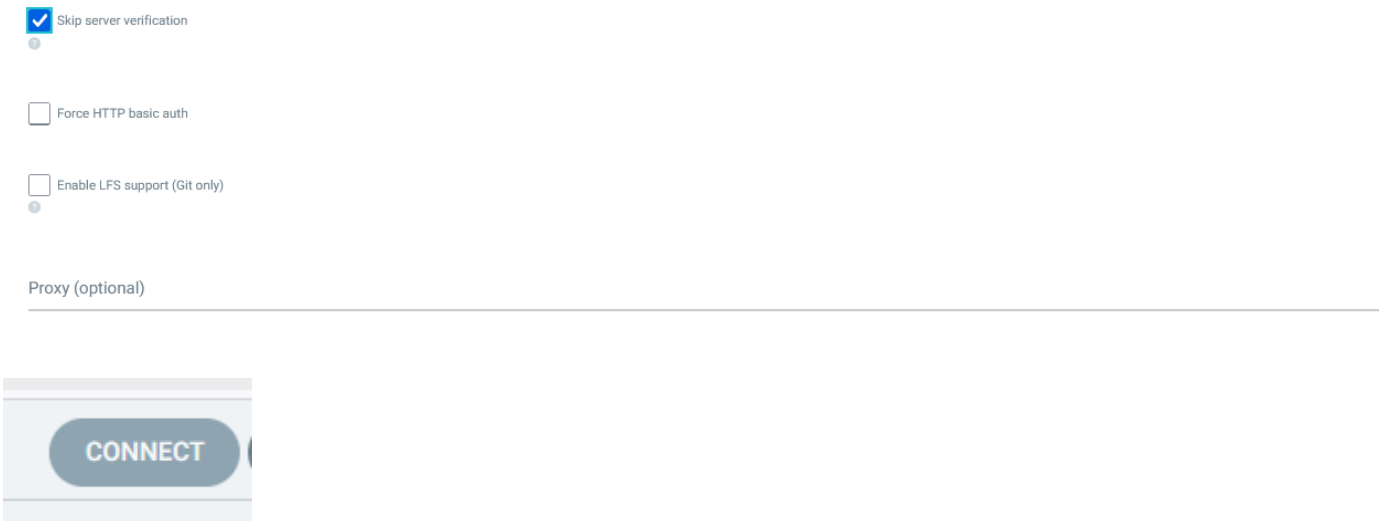
On va associer le dépôt Gitlab à Argocd

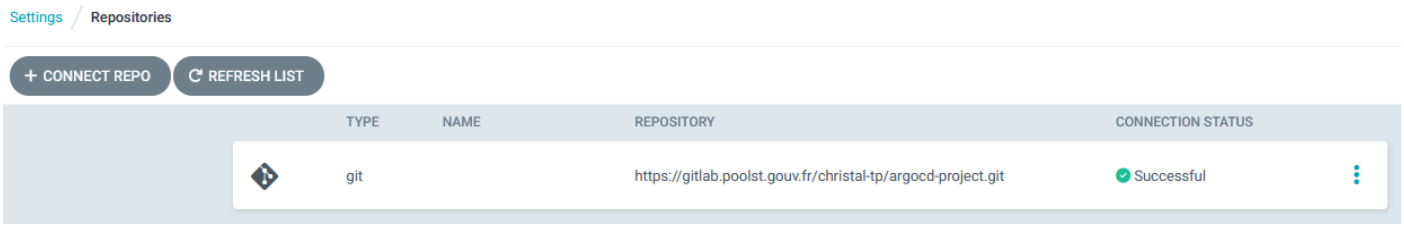






Comme la connexion choisie est en HTTPS et que le repository est public, on peut fournir des identifiants pour un compte pouvant pull le repository (le compte root ici).
Afin de ne pas avoir à vérifier l'état des certificats, il suffit de cocher la case Skip server vérification (en bas de page).

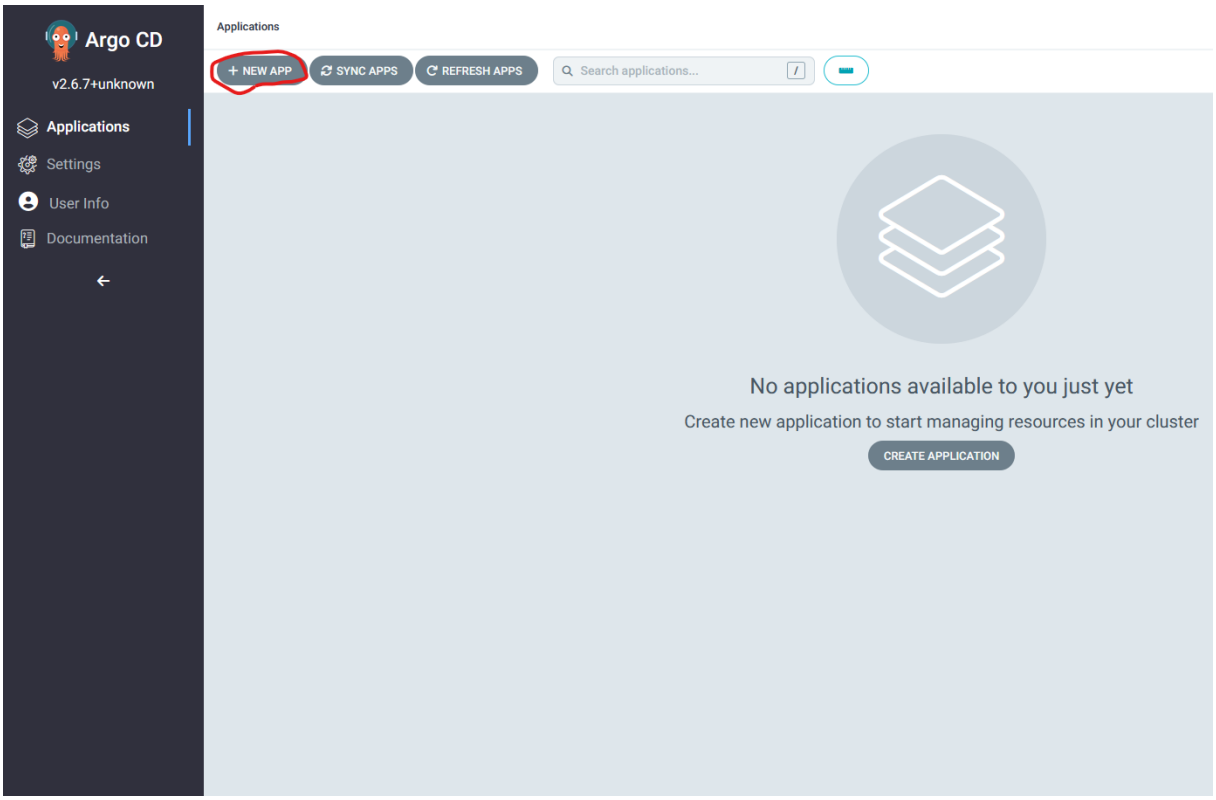




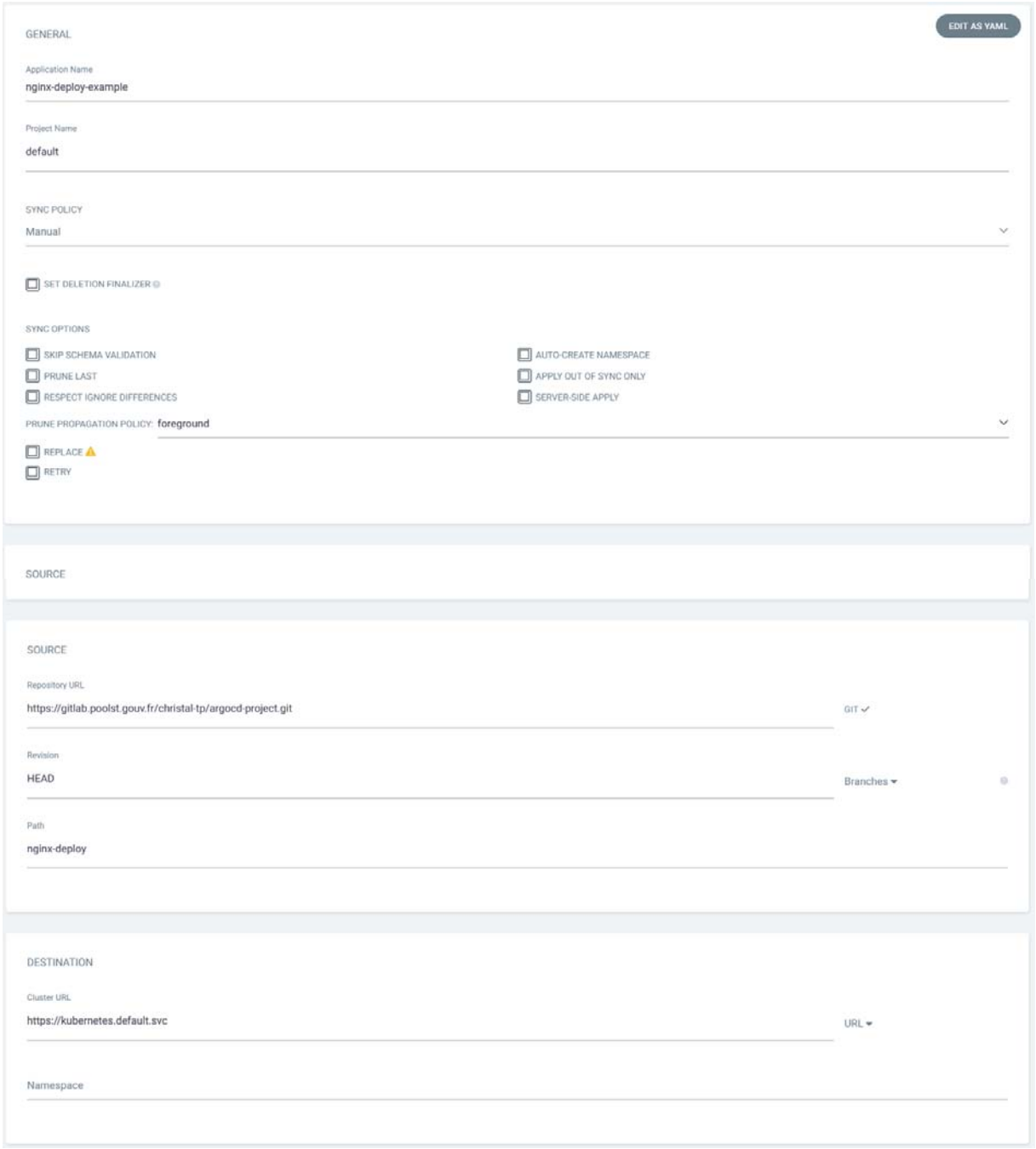
L'état de la connexion devrait être Successful

Création d'une application avec la webcli d'ArgoCD

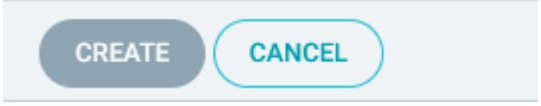
En reprenant le dépôt gitlab initialisé précédemment, il est possible de créer une application depuis l'interface graphique proposée par ArgoCD:



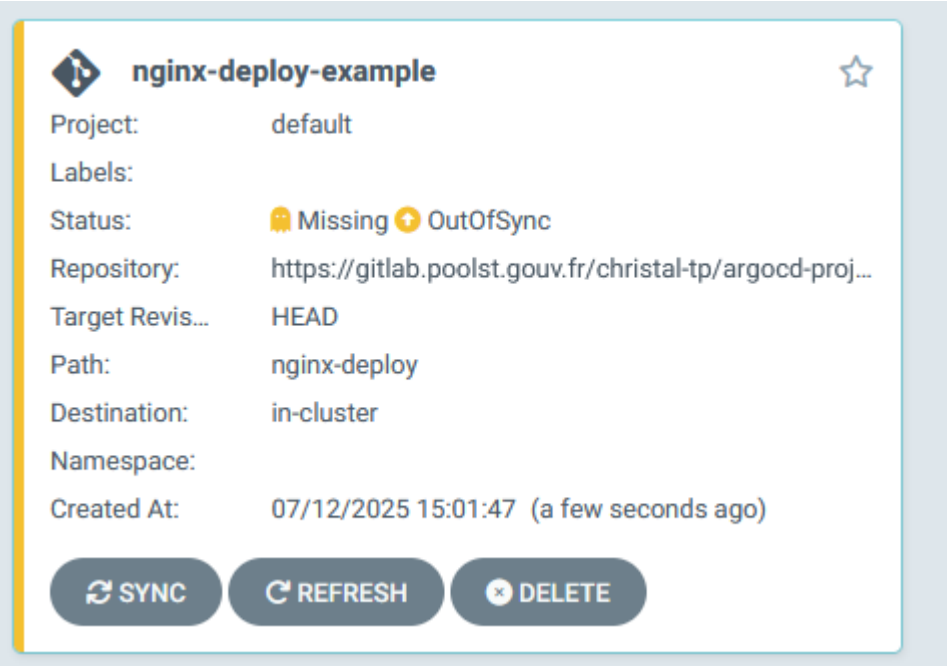
Il est possible de choisir certaines options que l'on verra plus tard dans cette fenêtre .



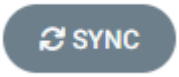
Puis, une fois toutes les informations renseignées, il suffit de cliquer sur CREATE en haut de page pour créer l'application.



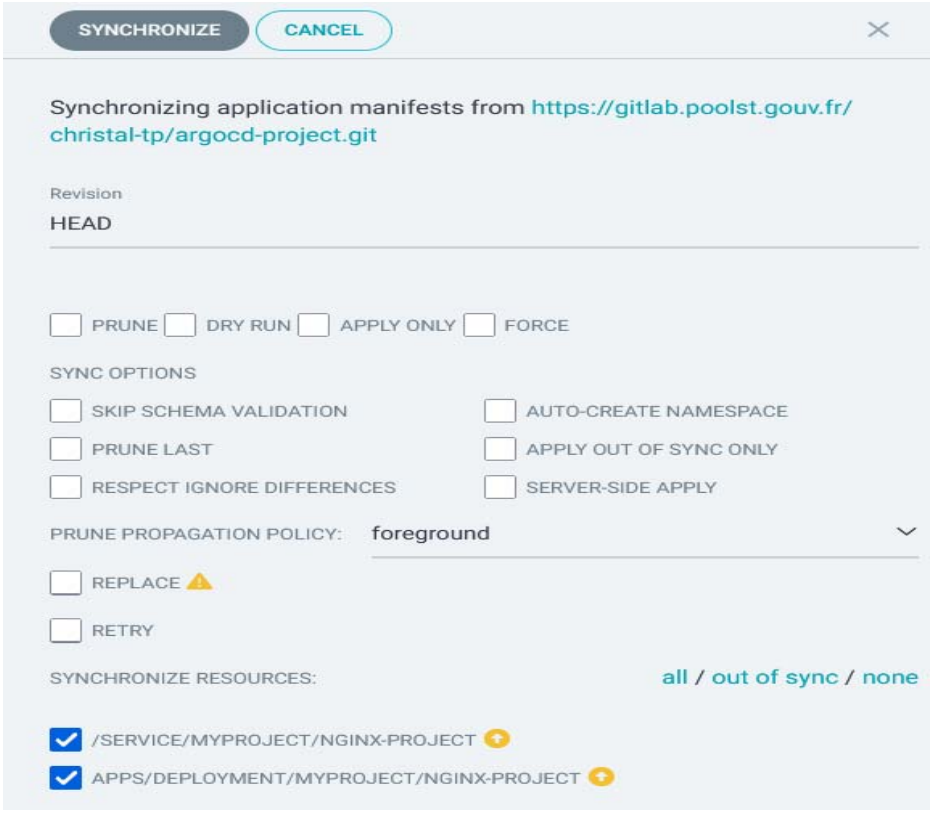
L'application créée ayant une synchronisation définie comme manuelle, ArgoCD considère que l'application n'est pas synchronisée :



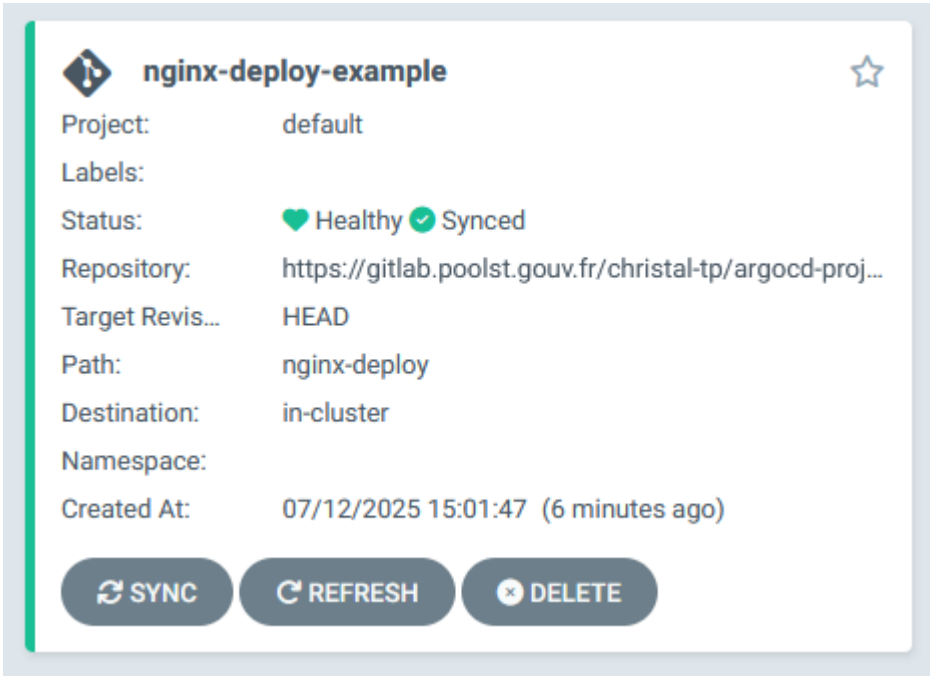
Cliquez sync pour synchroniser



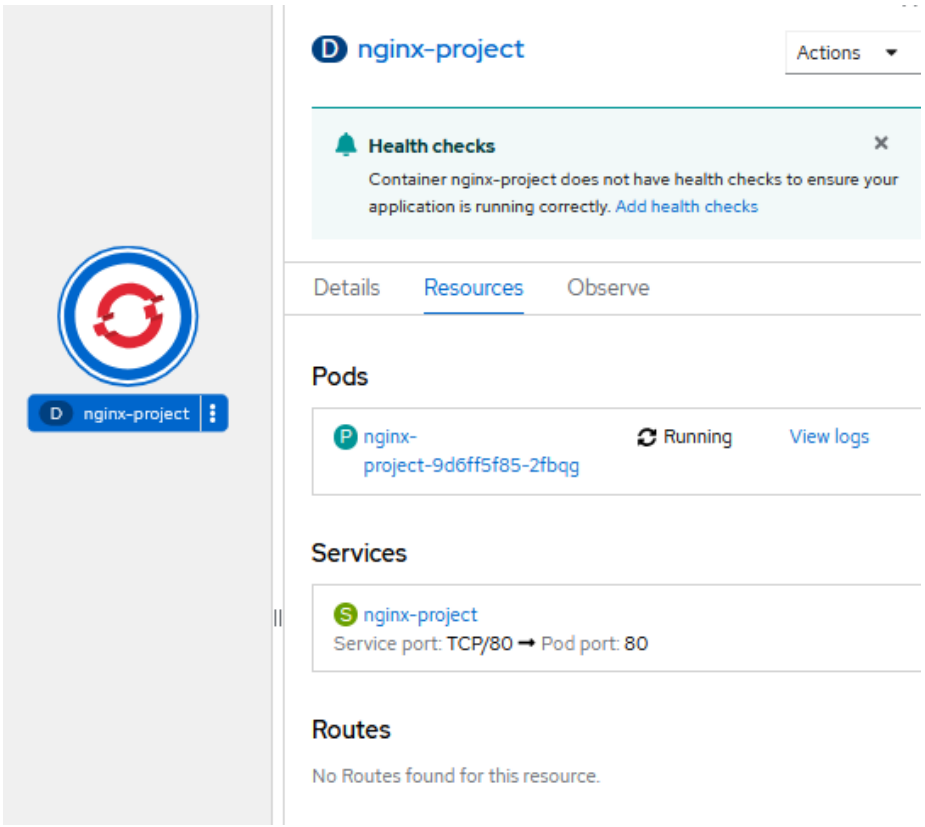
La fenêtre suivante apparait :



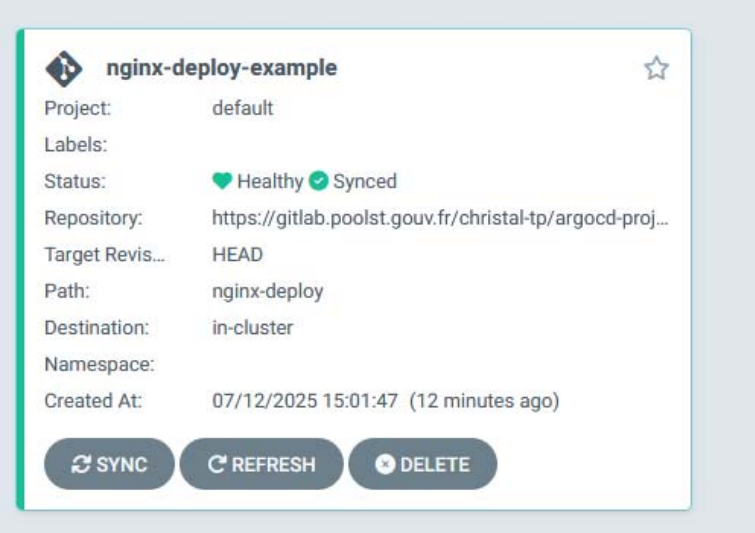
Cliquez sur synchronize. Le dépôt devient synchrone avec le cluster



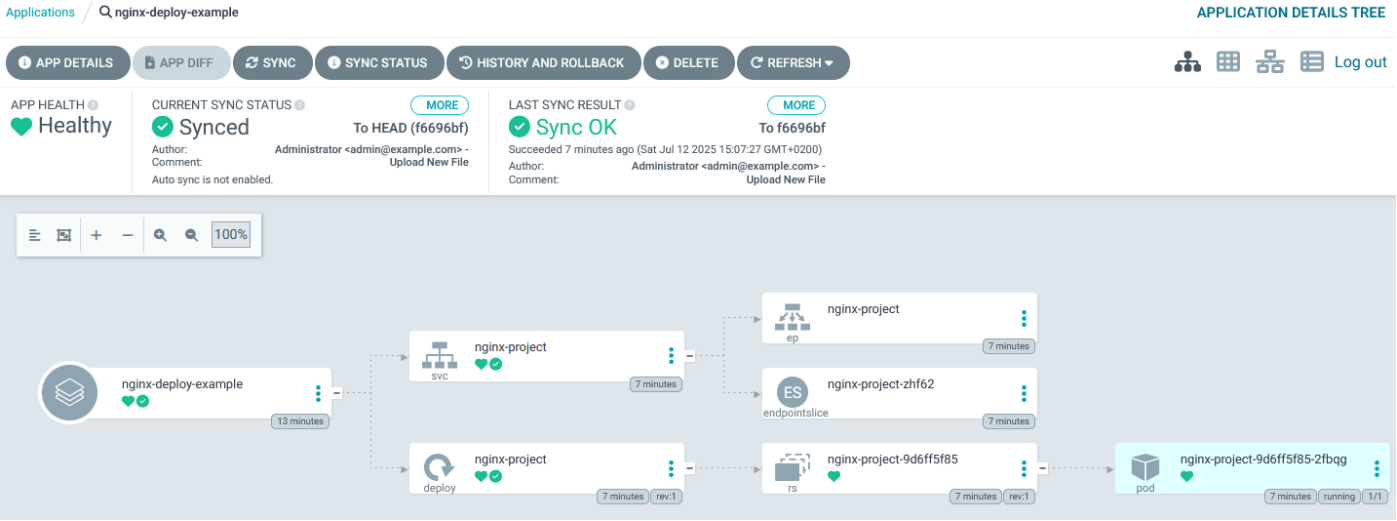
Vérifions la création de notre pod



Cliquez dans le cadre de l'application ARGOCD



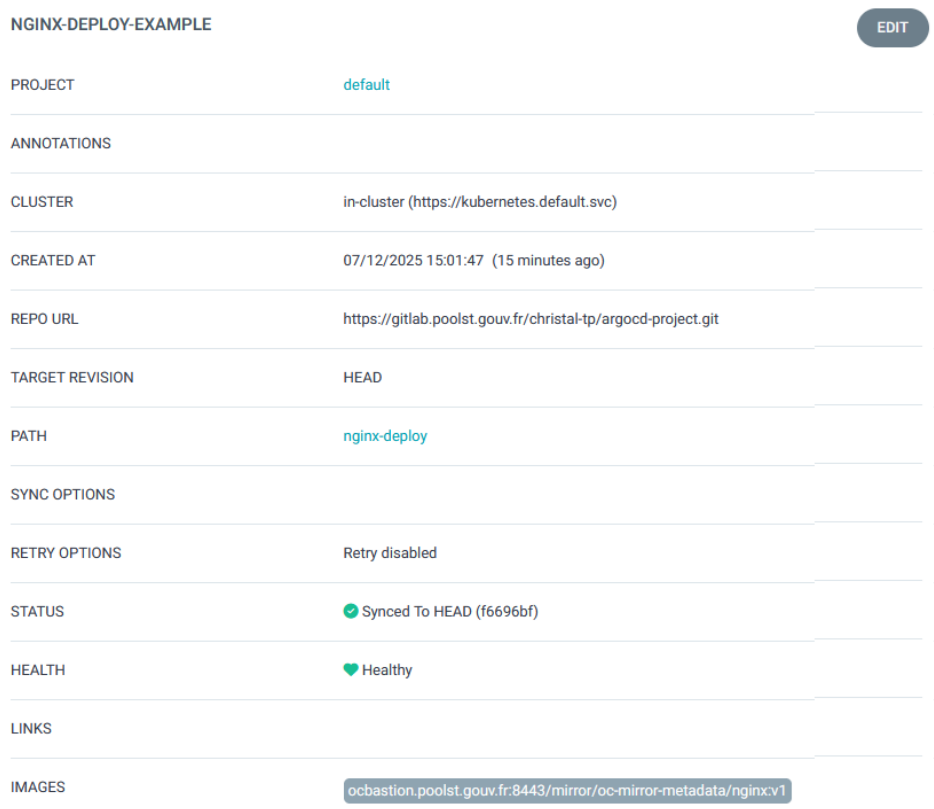
On observe l'arborescence des différents objets d'openshift



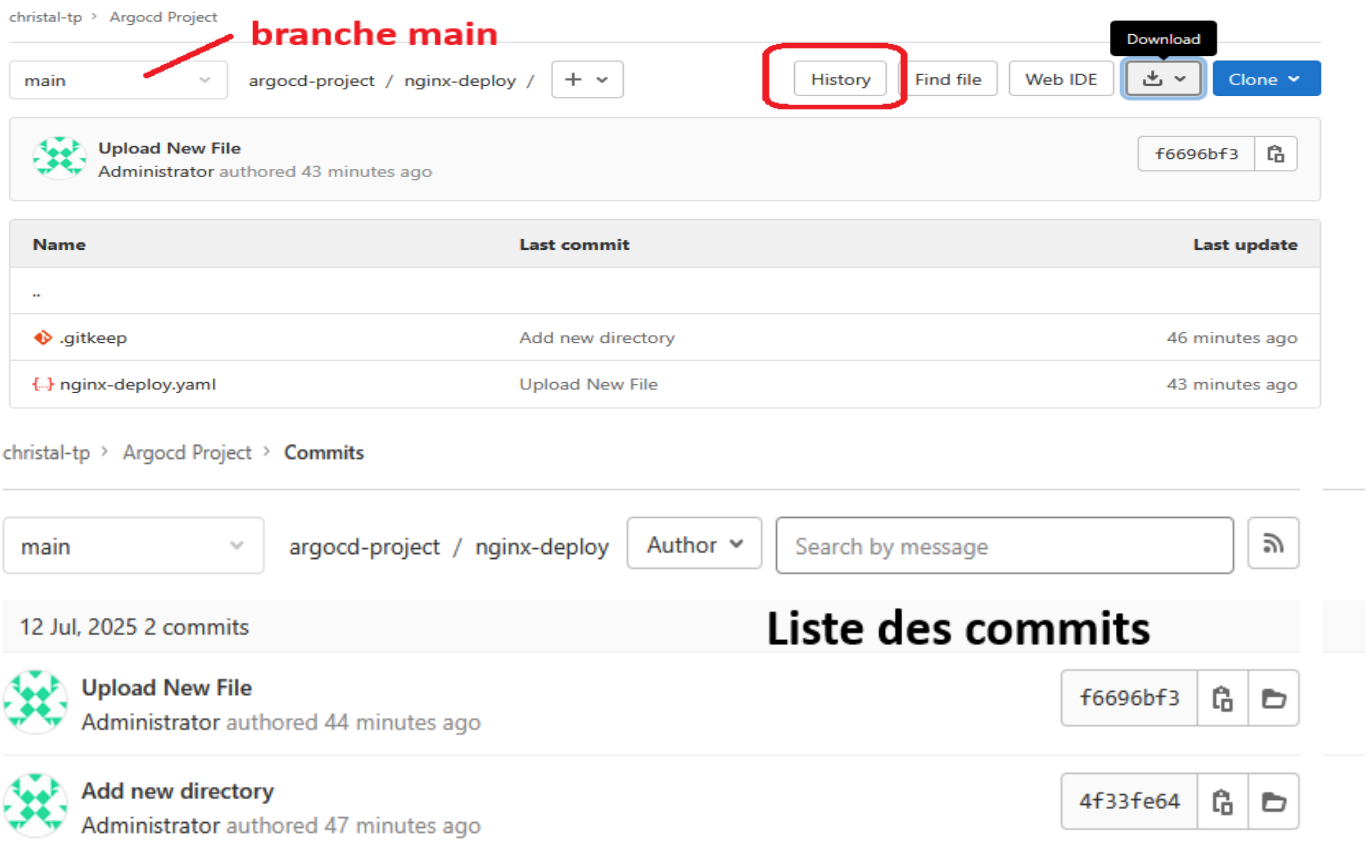
Cliquez sur APP DETAILS



On observe notre objet application de argocd que l'on peut modifier avec edit



La synchronisation se fait sur le dernier commit de la branche main de git (**head**) dont celui-ci a comme valeur f6696bf
Vérifions sur gitlab en cliquant sur history

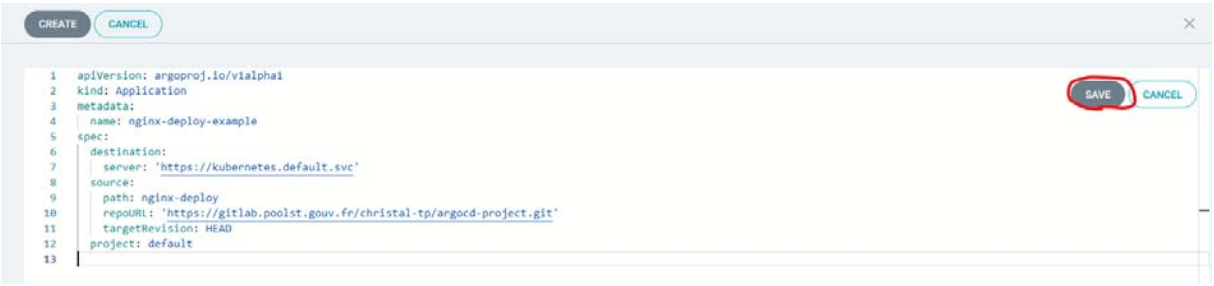


Revenons sur argocd et cliquant sur manifest

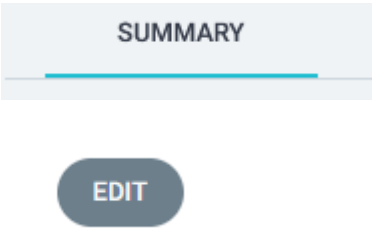


Le yaml suivant permet de recréer la même application que celle créée dans la partie précédente :

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: nginx-deploy-example
spec:
  destination:
    server: 'https://kubernetes.default.svc'
  source:
    path: nginx-deploy
    repoURL: 'https://gitlab.poolst.gouv.fr/christal-tp/argocd-project.git'
    targetRevision: HEAD
  project: default
```



Revenons dans summary puis EDIT



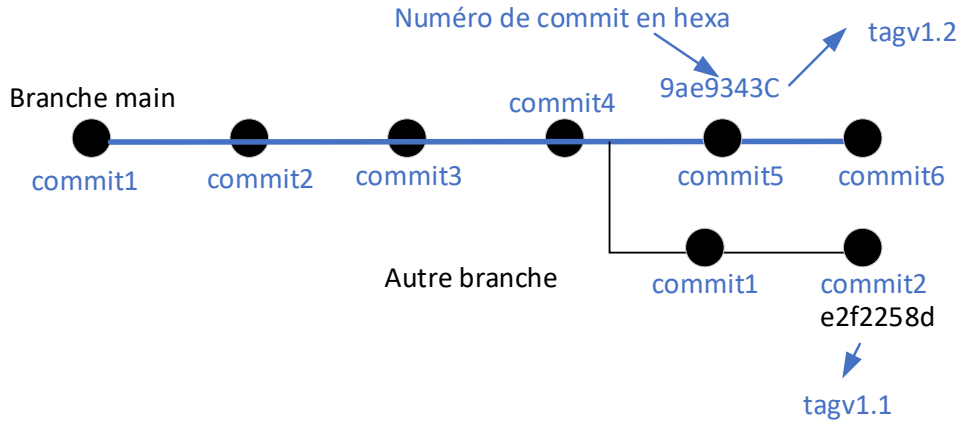
Le paramètre targetRevision nous permet de spécifier une migration du cluster par rapport au dépôt gitlab soit :

- ✓ Sur une nouvelle branche
- ✓ Soit sur un Tag



Commit : sorte de snapshot du dépot git qui traduit une différence entre deux versions du yaml

On peut associer des tags à des commits ou même créer une nouvelle branche



✍ Comment pourrait-on bloquer le déploiement de notre application sur une version bien précise ?

Modifions notre dépôt git (le target revision reste programmé à head)

christal-tp › Argocd Project › **Repository**

Edit file

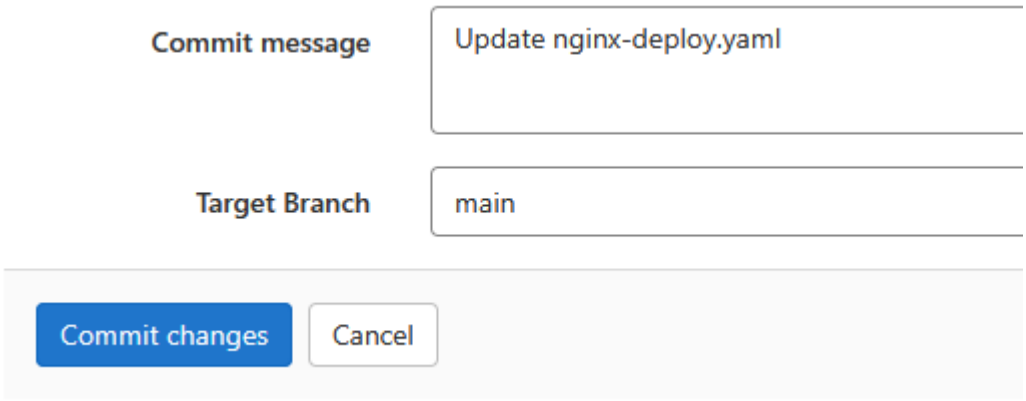
Write Preview changes

main | nginx-deploy/nginx-deploy.yaml

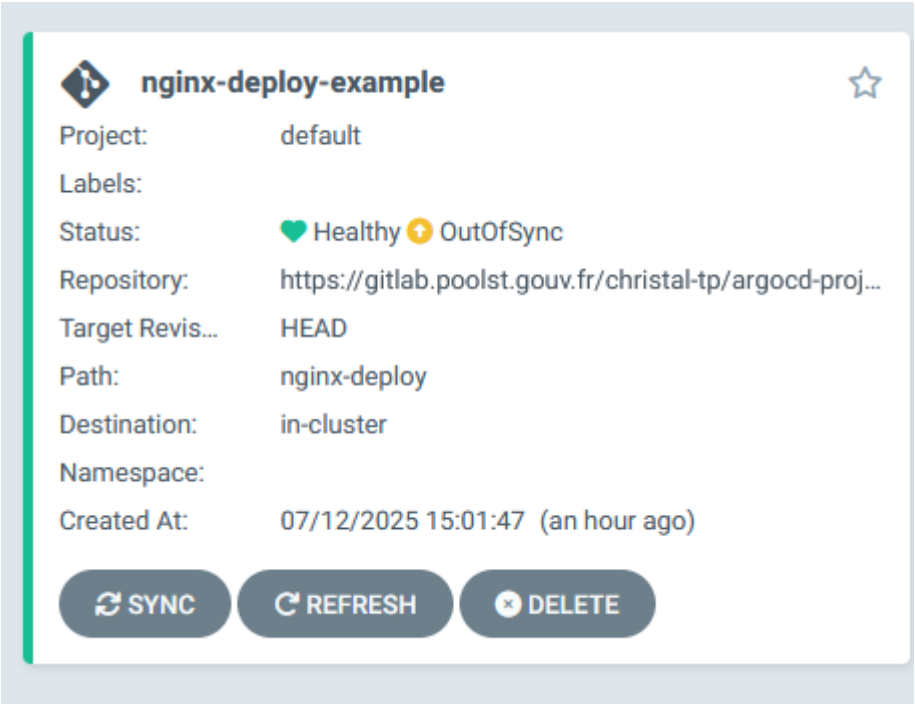
```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: "nginx-project"
5    namespace: myproject
6  spec:
7    selector:
8      matchLabels:
9        app: nginx-project
10   replicas: 2
11   template:
12     metadata:
13       labels:
14         app: nginx-project
15     spec:
16       containers:
17       - name: nginx-project
18         image: "ocbastion.poolst.gouv.fr:8443/mirror/oc-mirror-metadata/nginx:v1"
19         ports:
20         - containerPort: 80
21           protocol: TCP
22       imagePullSecrets:
23       - name: quay1
```

Modification réplicas de 2

Puis cliquez sur commit changes

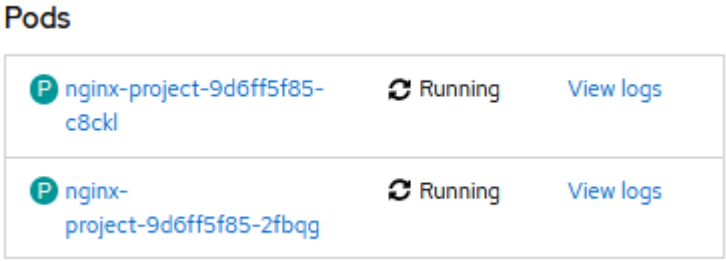


Observez ARGOCD



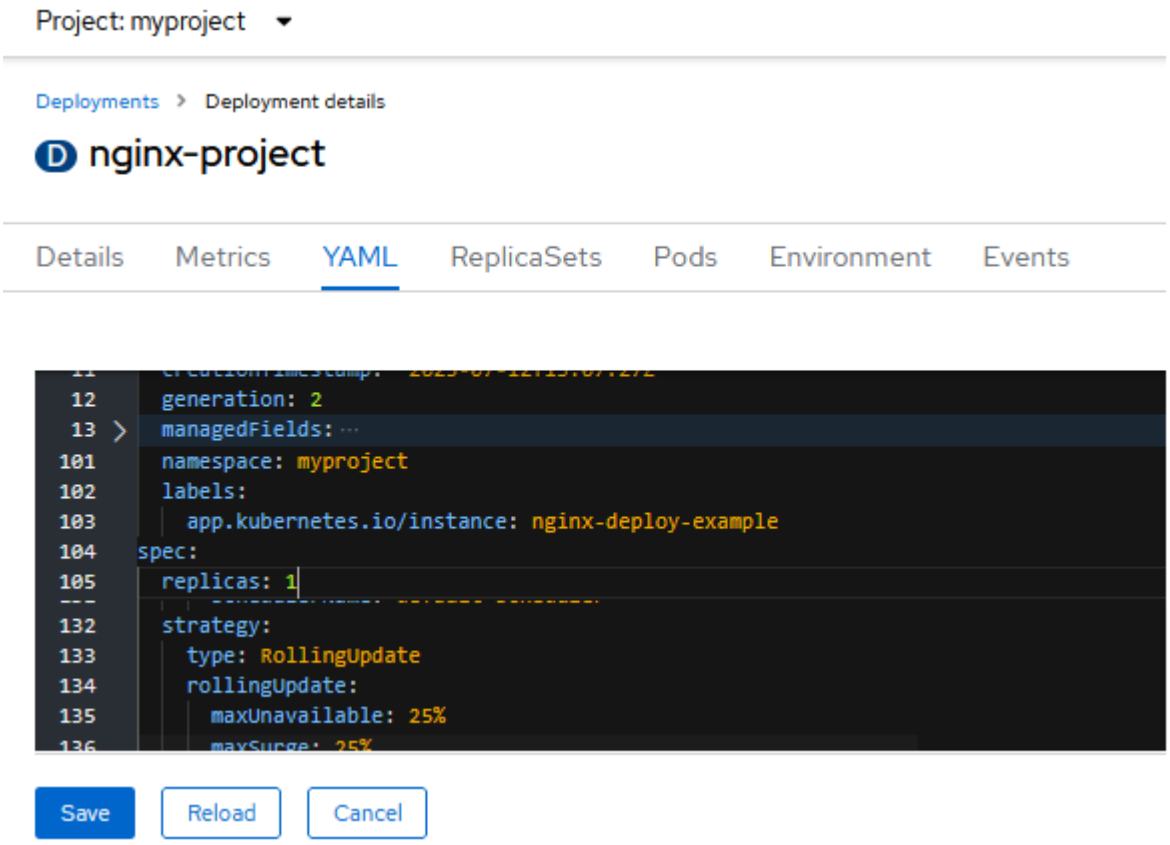
✍ Conclure

Synchroniser et observer les modifications sur notre applicatif



✍ Conclure

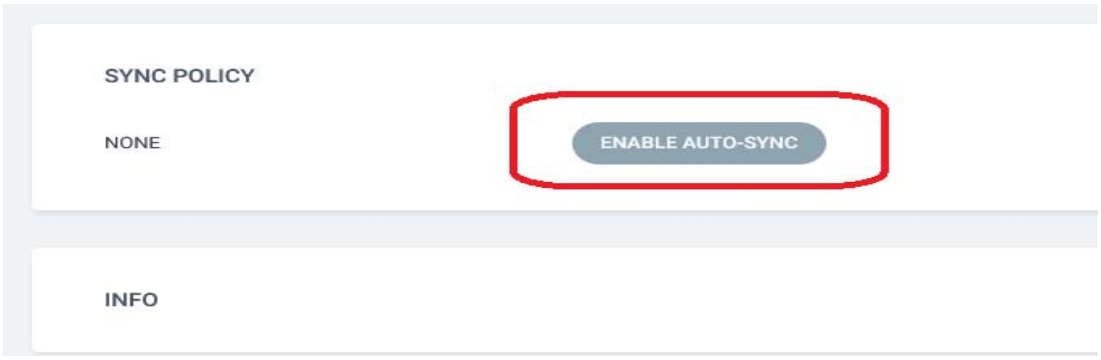
Par la suite essayez de modifier le replicaset à 1 directement dans le cluster (Web cli console de openshift) plus cliquez sur save

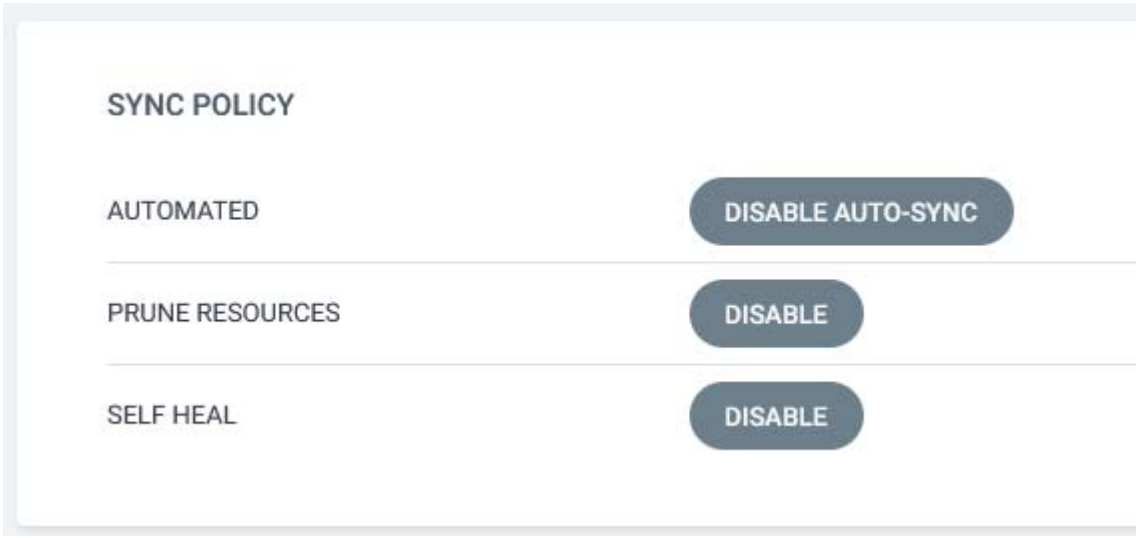


✍Conclure

✍ En resynchronisant ARGOCD que se passe t'il ?

On modifie les paramètres suivants (validez tous):





✍ Quel changement a-t-on apporté ?

✍Quel impact va avoir ce changement lors des deux modifications faites précédemment dans le dépôt git et directement dans le cluster

15.2.2 Description des champs du kind 'Application'

source

Dans le champ spec
Définit les informations concernant le dépôt et les fichiers de celui-ci sur lesquels les comparaisons avec le cluster seront faites.
source :

- repoURL <lien vers le dépôt> : Renseigne le lien vers le dépôt lié au cluster.
- targetRevision: <nom> : Renseigne le nom de la branche ou du tag que l'on souhaite prendre comme source.
- path: <chemin vers fichiers YAML> : Renseigne le chemin vers les fichiers que l'on souhaite prendre comme source au sein du dépôt. Si les fichiers se trouvent à la racine du dépôt, il faut renseigner le path /.

Si l'on souhaite avoir une application qui reste synchronisée sur la même version peut importe les changements apportés au git, il faut initialiser la valeur de targetRevision au tag correspondant. Si l'on souahite qu'ArgoCD se synchronise sur tous les changements quels qu'ils soient, il faut l'initialiser à HEAD. Et si l'on souhaite qu'ArgoCD se synchronise avec une branche spécifique, il suffit de la renseigner dans le champ targetRevision le nom de la branche.

destination

Dans le champ spec
Définit les informations concernant le cluster où vont être déployé les ressources.
destination :

- server : Renseigne l'URL vers l'API Kubernetes du cluster ou son nom symbolique.
- namespace : Renseigne le namespace du clsuter.

ignoreDifferences

Dans le champ spec
Ce champ répertorie les propriétés qu'ArgoCD va ignorer en comparant le cluster et le dépôt.
Exemple :

```
spec:
  ignoreDifferences:
    - group: "apps"
      kind: "Deployment"
      jsonPointers:
        - /spec/replicas
```

Ici, lorsqu'ArgoCD comparera l'état actuel du cluster et le dépôt git, il ignorera la valeur de replicas des deploymentdu groupe apps. Ce qui signifie que si la seule différence entre le cluster et le dépôt est une valeur de ce type, ArgoCD ne cosidérera pas le cluster comme non synchronisé.
Cependant, si le champ spec.syncPolicy.syncOptions.RespectIgnoreDifferences n'est pas initialisé à true, alors la synchronisation prendra en compte la différence de ces champs et les synchronisera.

finalizers

Dans le champ metadata.
Finalizers (réalisé lors de la suppression de l'application) :

- resources-finalizer.argocd.argoproj.io : En supprimant une application, elle entre dans un état de suppression pendant lequel toutes les ressources dépendantes sont supprimées avant de supprimer l'application. L'application reste visible pendant toute la durée de la suppression.
- resources-finalizer.argocd.argoproj.io/background (par défaut) : Supprime l'application, les objets dépendants sont supprimés par le garbage collector.

retry

Dans le champ spec.syncPolicy.
Permet de réessayer de synchroniser le cluster si celui-ci subit un premier échec.
Retry :

- limit: <int> : Nombre maximal de tentatives
- backoff :
    - duration: <int>m<int>s : Délai initial entre chaque tentative
    - maxDuration: <int>m<int>s : Délai maximal d'attente entre chaque tentative
    - factor: <int> : Facteur multiplicatif du délai après chaque tentative. Si une tentative échoue, le temps d'attente avant d'en refaire une est égale au temps d'attente attendu pour la tentative précédente multiplié par factor avec une valeur maximale de maxDuration.

Exemple :

```
spec:
  syncPolicy:
    retry:
      limit: 4
      backoff:
        duration: 5s
        maxDuration: 2m
        factor: 5
```

Les temps d'attentes pour chaque tentative seront les suivantes :

1. Tentative initiale
2. Attente initiale (5 secondes)
3. Première tentative
4. Attendre 5 secondes
5. Seconde tentative
6. Attendre 5 * 5 = 25 secondes
7. Troisième tentative
8. Attendre 2 minutes car 5 * 25 = 125 secondes or 125 secondes > 2 minutes
9. Quatrième tentative
10. Attendre 2 minute

automated

Dans le champ spec.syncPolicy
Si ce champ est null, la synchronisation sera manuelle. Sinon, automated est défini comme suit :
automated :

- prune
    - true : ArgoCD supprime les ressources qui ne sont pas définies dans le dépôt.
    - false : ArgoCD ne supprime pas les ressources qui ne sont pas définies dans le dépôt.
- selfHeal :
    - true : ArgoCD force l'aplication des ressources du dépôt dans le cluster.
    - false : ArgoCD envoie une alerte lorsque le dépôt et le cluster ne sont pas synchronisés mais n'applique pas les changements.

syncOptions

Dans le champ spec.syncPolicy.syncOptions.
Validate :

- true (par défaut) : ArgoCD valide les ressources pour vérifier qu'elles respectent les spécifications du cluster avant de les déployer.
- false : ArgoCD applique ces ressources même si elles ne sont pas en accord avec les spécifications du cluster.

PruneLast :

- true : En synchonisant le cluster avec le dépôt, ArgoCD attend que toutes les ressources soient déployées et fonctionnelles avant de supprimer les ressources non présentes dans le dépôt.
- false (par défaut) : Les ressources non présentes dans le dépôt sotn supprimées en parallèle de la synchonisation avec le dépôt.

RespectIgnoreDifferences :

- true : ArgoCD prend en compte le contenue de spec.ignoreDifferences lors de la synchronisation.
- false (par défaut) : ArgoCD prend en compte le contenue de spec.ignoreDifferences lors de la synchronisation.

CreateNamespace :

- true : Permet à ArgoCD de créer le namespace donné dans le champ spec.destination.namespace si celui-ci n'existe pas.
- false (par défaut) : Génère une erreur lors de la synchronisation si le namespace n'existe pas.

ApplyOutOfSyncOnly :

- true : Lors d'une synchronisation, applique uniquement les ressources n'étant pas synchronisés.
- false (par défaut) : Lors d'une synchronisation, applique toutes les ressources, qu'elles soient synchonrisées avec le dépôt ou non.

ServerSideApply :

- true : ArgoCD va utiliser kubectl apply --server-side pour appliquer les changements
- false (par défaut) : ArgoCD va utiliser kubectl apply pour appliquer les changements

Prune Propagation Policy (réalisé lors de la synchronisation avec le dépôt) :

- Foreground (par défaut) : Supprimes les ressources enfants (pods, ReplicaSets, ...) avant de supprimer les ressources parents (Deployment, ...).
- Background : Supprime les ressources parents puis le garbage collector supprime les ressources orphelines.
- Orphan : Supprime les ressources parent mais ne supprime pas les enfants

Replace :

- true : ArgoCD utilise kubectl replace ou kubectl create pour appliquer les changements.
  Si ServerSideApply=true et Replace=true, ArgoCD va prioriser l'option Replace et utiliser kubectl replace ou kubectl create.
- false (par défaut) : ArgoCD utilise kubectl apply pour appliquer les changements.

Exemple

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: nginx-deployment-example
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  destination:
    name: ''
    namespace: myproject
```

```
  server: 'https://kubernetes.default.svc'
source:
  path: nginx-deploy
  repoURL: 'https://gitlab.poolst.gouv.fr/christal-tp/argocd-project.git'
  targetRevision: main
sources: []
project: default
syncPolicy:
  automated:
    prune: true
    selfHeal: true
  syncOptions:
    - Validate=false
    - PruneLast=true
    - RespectIgnoreDifferences=true
    - ServerSideApply=true
    - ApplyOutOfSyncOnly=true
    - CreateNamespace=true
    - PrunePropagationPolicy=background
    - Replace=true
  retry:
    limit: 2
    backoff:
      duration: 5s
      maxDuration: 3m0s
      factor: 2
```