Documentation DataHub Développeur

D3IA

Cyril HOUISSE - Jean-Philippe LANG

Table des matières

1. 2. Prise en main

Ce chapitre décrit les actions à mener par une équipe de développeurs pour accéder aux différents services de la plateforme DataHub.

1.1 2.1 Les plateformes

Deux plateformes sont actuellemnt disponibles. Une dédiée à la recette, la seconde à la production.

1.2 2.2 Etablir votre demande d'accès

La création d'un projet sur le DataHub s'effectue au travers d'une demande sur SPS projet Assistance Produits Data.

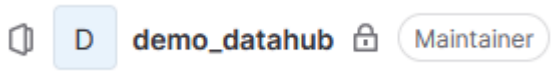
Les éléments demandés dans ce formulaire sont décrits dans cette documentation.

Par défaut, un projet se voit attribuer **8 CPUs** et **32 Go de RAM**.

1.3 2.3 Stockage des codes sources sur GitLab

Pour bénéficier du service de déploiement de la plateforme DataHub, vous devez disposer d'un projet sur le GitLab du ministère et avoir le rôle `Maintainer` ou `Owner` sur ce dernier.

A noter : les fonctionnalités d'intégration et de déploiement continus de la platforme GitLab ne sont accessibles qu'aux utilisateurs appartenant à **un groupe**.



La documentation qui suit retient le principe d'un projet GitLab unique pour l'intégration d'une part (génération de la ou des image(s) et publication dans la registry GitLab) et le déploiement d'autre part (déploiement des ressources sur le DataHub). Il est toutefois possible de séparer les éléments spécifiques au déploiement dans un projet GitLab distinct. Dans ce cas, c'est le nom de ce dernier projet qui est à fournir lors de votre demande de création de projet DataHub.

1.4 2.4 Intégrations et déploiements continus

L'option **CI/CD** (ou **Intégration et livraison continues** en français) doit être activée dans les paramètres de votre projet GitLab. Le runner préconisé est celui de GitLab, mais il est possible de mettre en place un runner dédié.

1.4.1 2.4.1 Préparation du CI (Intégration continue)

2.4.1.1 Le fichier `.gitlab-ci.yml`

L'exemple ci-dessous (pour une intégration d'application Python) donne un modèle de la structure et du contenu attendus. L'image de l'application est générée avec kaniko à partir du descripteur `docker/Dockerfile` présent dans le dépôt, et cette image générée est publiée dans la registry GitLab.

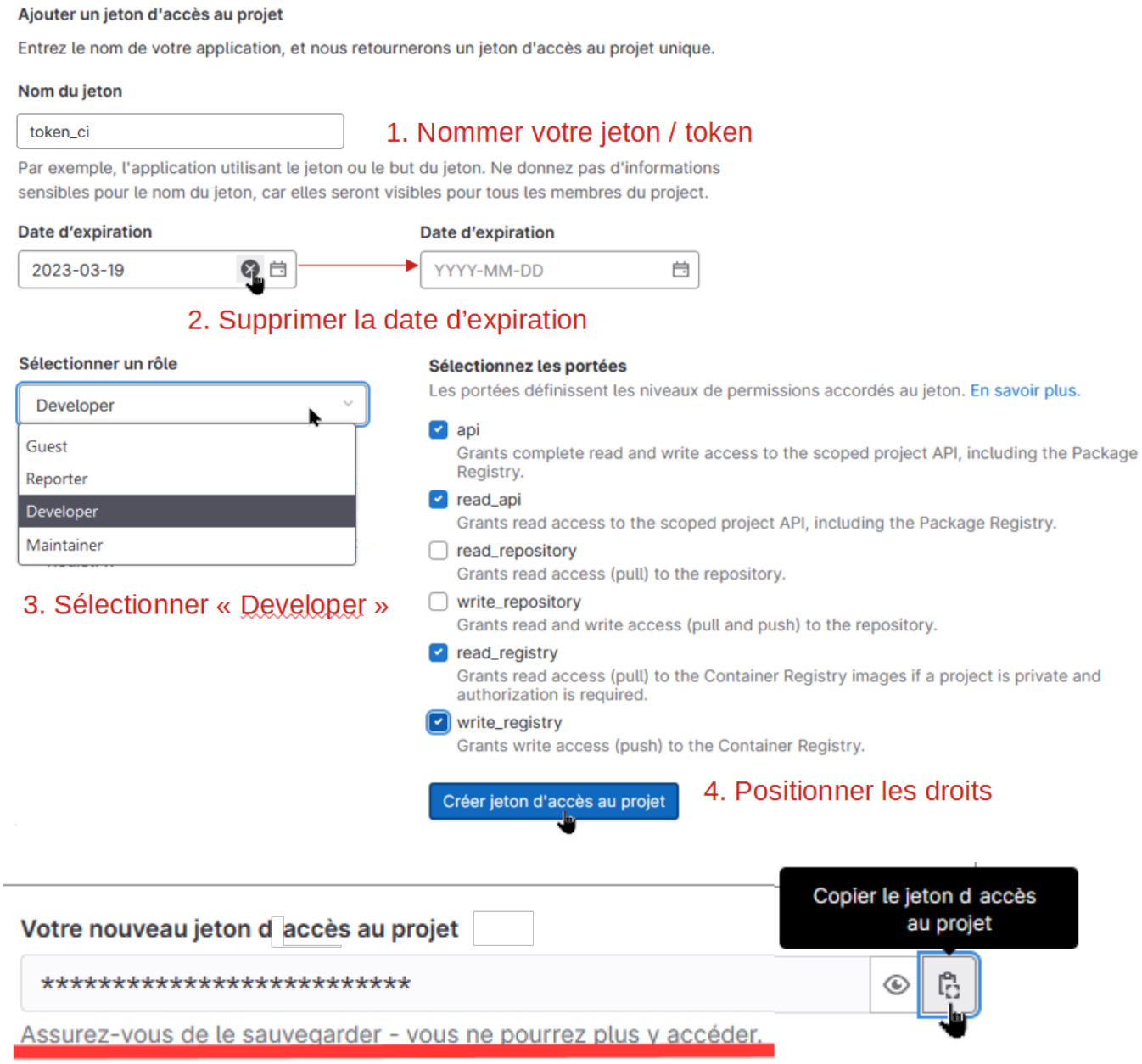
```
variables:
    REGISTRY_BASE_IMAGE: docker.io/library/python:3.8.10
    REGISTRY_FINAL_IMAGE: $REGISTRY/$REGISTRY_PROJECT/monappli:$CI_COMMIT_TAG
    REGISTRY_PROJECT: mon_appli
    REGISTRY: registry.gitlab-forge.din.developpement-durable.gouv.fr/snum/ds/gd3ia/

image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""]

before_script:
    - mkdir -p /kaniko/.docker
    - echo "{\"auths\":{\"$REGISTRY\":{\"username\":\"$REGISTRY_USER\",\"password\":\"$REGISTRY_PASSWORD\"}}}" > /kaniko/.docker/config.json

application:
  stage: build
  script:
    - |
         /kaniko/executor  --build-arg REGISTRY_BASE_IMAGE=$REGISTRY_BASE_IMAGE \
           --context $CI_PROJECT_DIR \
           --destination $REGISTRY_FINAL_IMAGE \
           --dockerfile $CI_PROJECT_DIR/docker/Dockerfile
  only:
    - tags
```

2.4.1.2 Mise en place de `REGISTRY_USER` et `REGISTRY_PASSWORD`

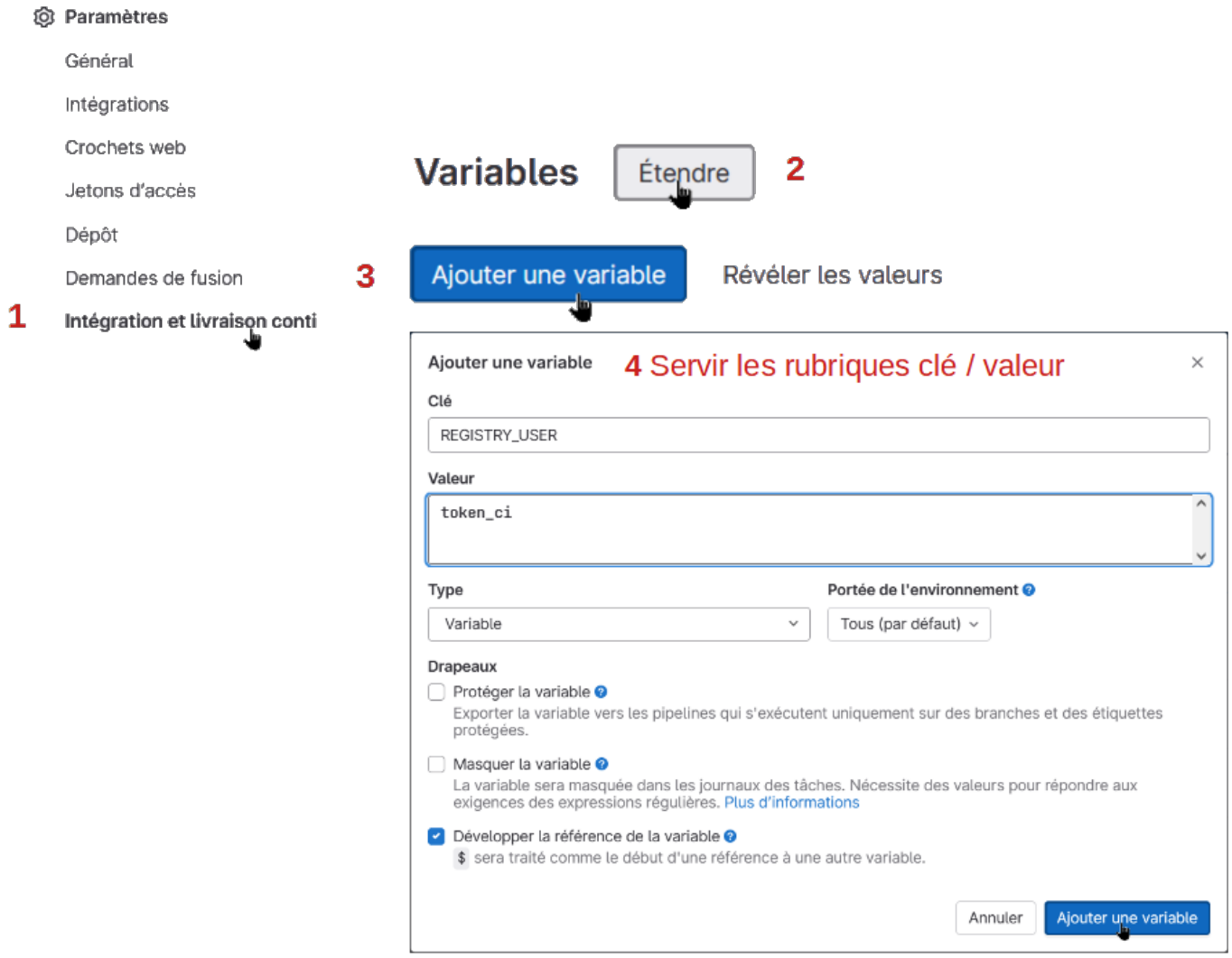
Ce script nécessite que les variables `REGISTRY_USER` et `REGISTRY_PASSWORD`, qui permettent d'accéder à la registry GitLab, soient définies. Pour définir ces variables, il suffit de créer un jeton avec les droits d'accès à la registry, et d'affecter le nom et la valeur de ce jeton respectivement aux variables CI `REGISTRY_USER` et `REGISTRY_PASSWORD`.

2.4.1.2.1 CRÉATION DU TOKEN

Dans le menu de votre projet GitLab **Settings / Access Tokens** (ou **Paramètres / Jetons d'accès** en français) :



2.4.1.2.2 CRÉATION DES VARIABLES



Renouveler les opérations numérotées **3** et **4** avec comme "Clé" `REGISTRY_PASSWORD` et comme "Valeur" la valeur du token copiée précédemment.

Type	↑ Clé	Valeur	Options	Environnements	
Variable	REGISTRY_PASSWORD	*****	Masquée, Développée	Tous (par défaut)	
Variable	REGISTRY_USER	*****	Développée	Tous (par défaut)	

1.4.2 2.4.2 Préparation du CD

Le déploiement sur le DataHub est géré au travers du répertoire `deploys` situé à la racine de votre dépôt. Ce répertoire contient la description des ressources à déployer sous forme de fichiers `.yaml` appelés descripteurs (ou manifests). Le nommage de ces fichiers descripteurs est libre, ils doivent simplement avoir l'extension `.yaml`.

En plus de la description des ressources à déployer, ce répertoire doit contenir le fichier `fleet.yaml` (nom à respecter) qui précise la cible du déploiement. Son contenu est à reprendre à l'identique de celui-ci :

```
targetRestrictions:
  - clusterGroup: prod
targets:
  - clusterGroup: prod
```

Le processus de déploiement du DataHub scrute ce répertoire `deploys`, dans la branche `main` par défaut. Une autre branche peut être choisie lors de la demande de création du projet DataHub. A chaque changement détecté dans un des fichiers `.yaml` présents, le déploiement est déclenché par le DataHub.

Pour permettre au processus de déploiement du DataHub d'accéder à vos éléments sur GitLab (dépôt de code et registry), vous devez créer deux jetons en respectant portées et rôles. Les noms des jetons sont donnés à titre d'exemple.

jetons d'accès au projet actifs (3)

Nom du jeton	Portées	Créé	Dernière Utilisation	Expire	Rôle	Action
token_harbor	api, read_api, read_registry	janv. 19, 2023	il y a 13 heures	Jamais	Developer	
token_rancher	api, read_api, read_repository	janv. 18, 2023	il y a 18 heures	Jamais	Developer	
token_ci	api, read_api, read_registry, write_registry	janv. 18, 2023	il y a une semaine	Jamais	Developer	

`token_harbor` : token permettant d'accéder à la registry GitLab pour récupérer les images de votre projet

`token_rancher` : token permettant d'accéder au dépôt, pour lire les descripteurs contenus dans le répertoire `deploys`

Ces informations (nom des tokens et valeur), ainsi que **le nom de votre projet** sont à communiquer de manière sécurisée au Groupe D3IA. Le nom de votre projet est utilisé pour créer un namespace dans lequel figureront vos pods (et uniquement les votres).

1.4.3 2.4.3 URL d'exposition

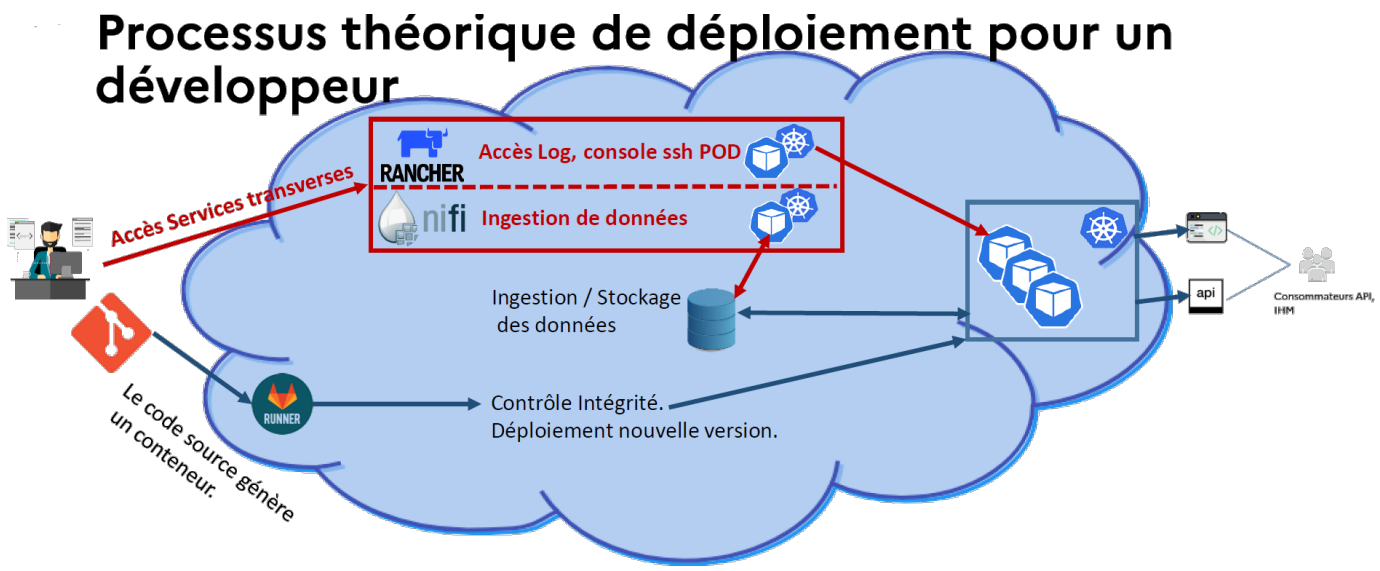
L'URL d'accès est définie dans un manifeste de type `Ingress` :

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: monapp-ingress
  namespace: mon-espace
spec:
  ingressClassName: haproxy
  rules:
  - host: **mon_url**
    http:
      paths:
      - backend:
          service:
            name: monapp-service
            port:
              number: 3000
        path: /
        pathType: Prefix
```

mon_url deviendra pour une exposition :

- sur le RIE, xxxx-namespace**.datahub.eco4.cloud.e2.rie.gouv.fr**
- internet, xxxx-namespace**.datahub.din.developpement-durable.gouv.fr** (Rappel : pour bénéficier du lien internet, il faut en faire la demande. Une action du GD3IA est nécessaire.)

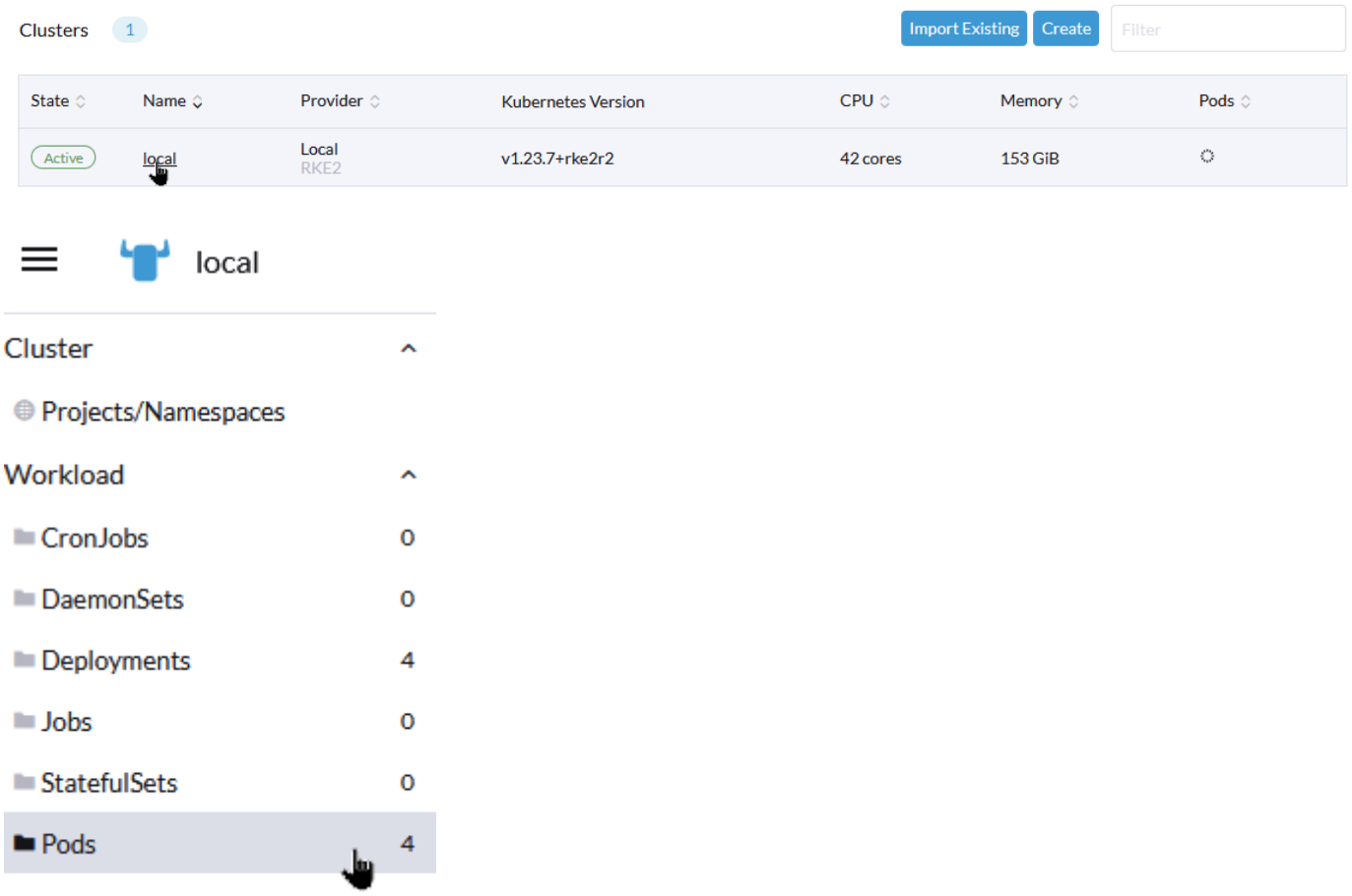
1.4.4 2.4.4 Schéma de synthèse de la chaîne de déploiement



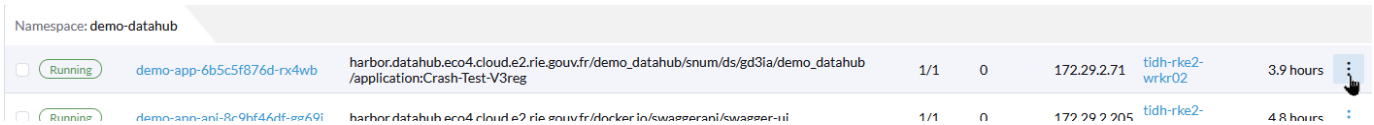
2. 3. Rancher : Maintenance et surveillance des applications conteneurisées

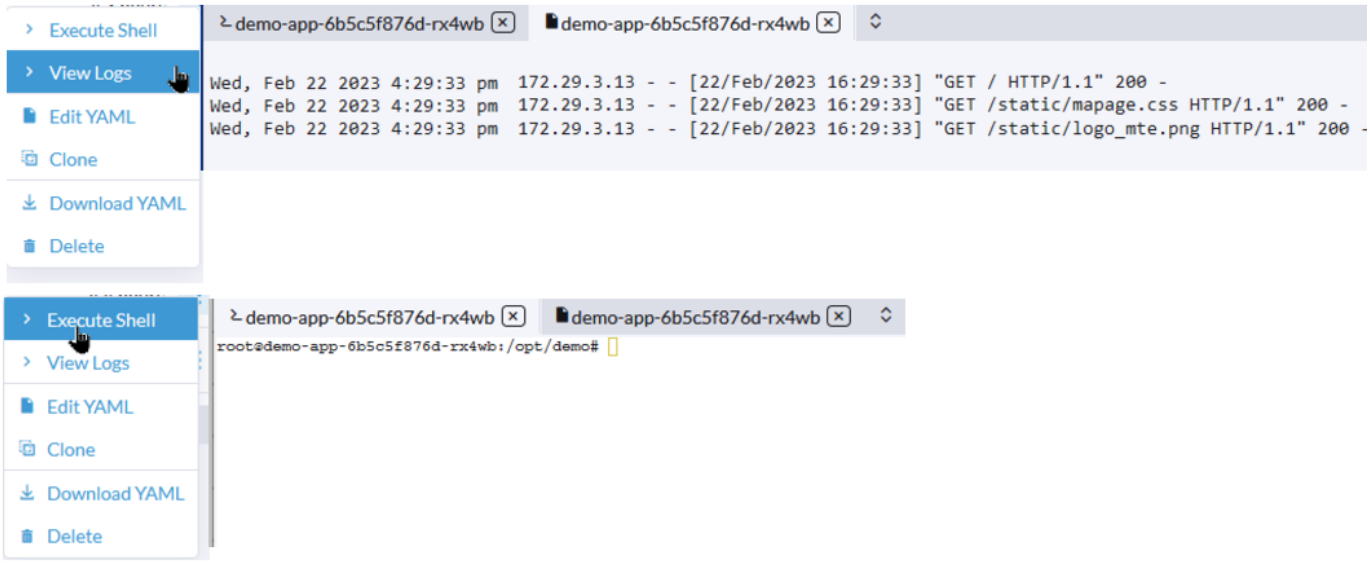
Intimement lié au service de déploiement de conteneurs, Rancher est l'outil qui vous permet d'accéder à vos pods déployés dans le DataHub. Ce service n'est pas encore Cerbérisé. Un identifiant / mot de passe vous sera communiqué par le Groupe D3IA.

Accéder à Rancher (RIE uniquement)



Cette option permet entre autres de voir les logs générés par vos pods et d'y accéder via un shell.





⚠ Toute modification des pods via un shell n'est pas pérenne. Un pod peut être reconstruit et les modifications opérées perdues.

3. Exemples d'utilisation

3.1 4. Déployer une application R Shiny dans le DataHub

- Dépôt GitLab de l'exemple : https://gitlab-forge.din.developpement-durable.gouv.fr/snum/ds/gd3ia/DataHub_Exemples/rshiny
- L'application Shiny déployée : http://appshiny.demo-datahub.datahub.eco4.cloud.e2.rie.gouv.fr/

3.1.1 4.1 Objectif

Dans cet exemple, l'objectif est de :

- construire une image d'une application R Shiny
- déployer cette image pour rendre l'application accessible sur le RIE.

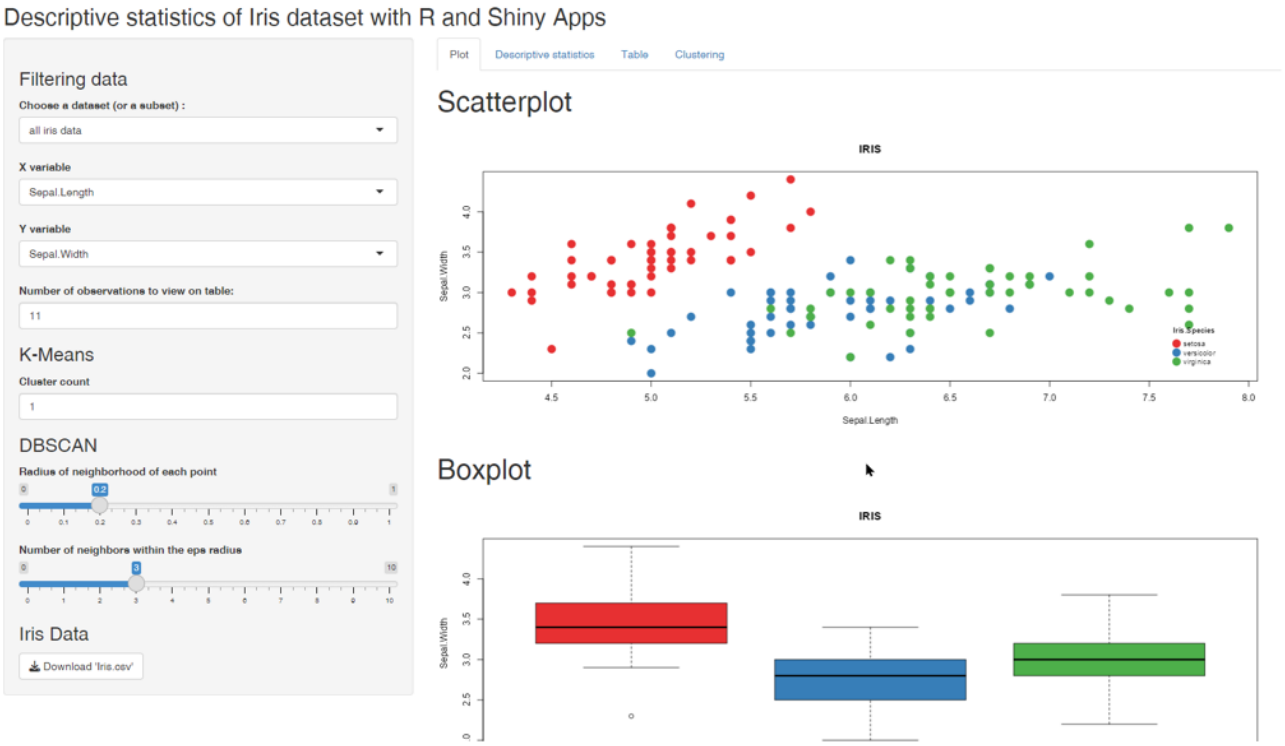
3.1.2 4.2 L'application R Shiny

L'application comprend trois fichiers stockés dans un dossier `app`. Cette structure est usuelle pour ce type d'application :

- `Global.R` reprend les librairies et variables utilisées par le projet
- `ui.R` décrit les éléments de l'interface utilisateur
- `server.R` définit la logique serveur

Cette application s'appuie sur les données Iris embarquées dans la librairie Shiny, le sujet de l'accès à des sources de données externes n'est donc pas traité dans le cadre de cet exemple.

Ci-dessous un aperçu du rendu de cette application.



3.1.3 4.3 Construction de l'image via la CI GitLab

La CI du projet est définie dans `.gitlab-ci.yml`. Dans cet exemple, elle est déclenchée lors d'un tag. Elle construit l'image de l'application (basée sur Rocker) à l'aide kaniko et publie cette image sur la registry GitLab.

3.1.4 4.4 Déploiement sur le DataHub

Le processus de déploiement du DataHub scrute le répertoire `deploys` du dépôt GitLab déclaré à la création du projet DataHub. A chaque changement détecté dans un des descripteurs (ou manifests) présents dans ce répertoire, le déploiement est déclenché par le DataHub.

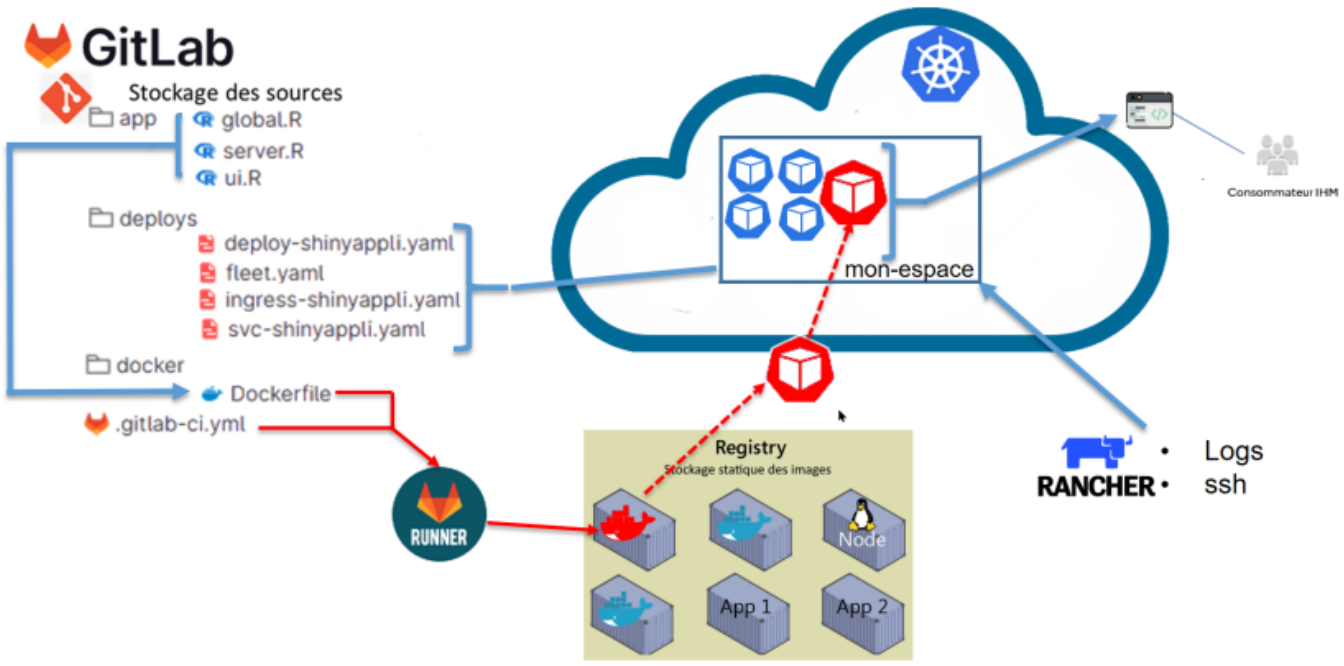
Dans l'exemple, le répertoire `deploys` contient 4 descripteurs :

- `fleet.yaml` : décrit la cible du déploiement (ce descripteur est obligatoire, le contenu du fichier de cet exemple est à reprendre en l'état)
- `deploy-shinyappli.yaml` : déploiement du container de l'image de notre application
- `svc-shinyappli.yaml` : déclaration du service associé à notre application
- `ingress-shinyappli.yaml` : l'ingress, publiant le service à l'adresse http://appshiny.demo-datahub.datahub.eco4.cloud.e2.rie.gouv.fr

En actualisant le nom de l'image référencée dans `deploy-shinyappli.yaml`, on déclenche le déploiement de la nouvelle image.

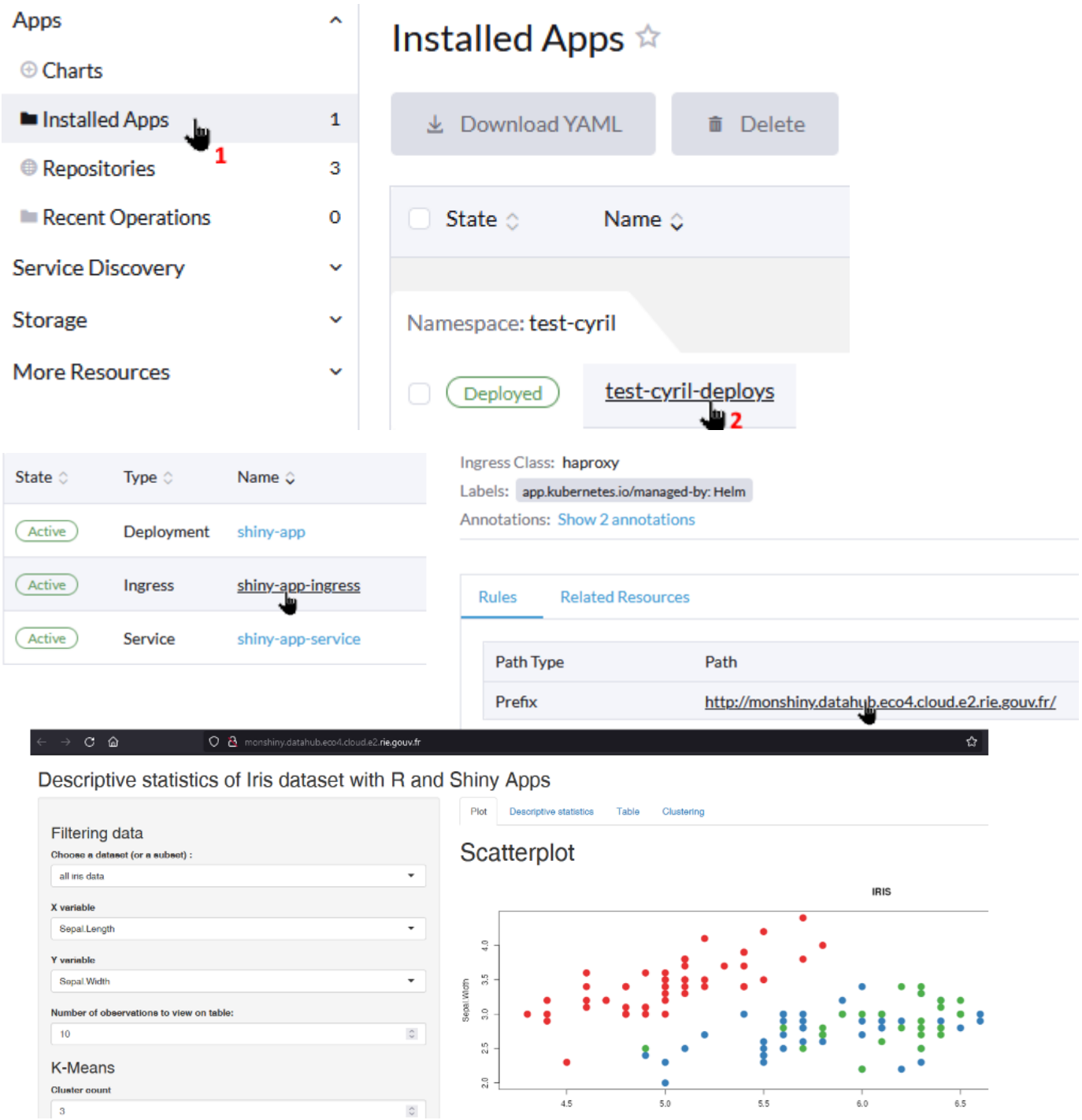
A noter : `deploy-shinyappli.yaml` référence l'image dans une registry Harbor propre au DataHub (harbor.datahub.eco4.cloud.e2.rie.gouv.fr). Cette registry ne joue qu'un rôle de proxy, le chemin vers l'image doit être identique à celui de la registry GitLab.

Schéma de synthèse de la chaîne de déploiement :



3.1.5 4.5 Rancher

L'application déployée est visible dans l'interface Rancher. Rancher vous permet notamment de gérer le pod de l'application (arrêt, redémarrage), d'accéder à ses logs et à un shell.



3.2 5. Déploiement à l'aide de helm

3.2.1 5.1 Principe de fonctionnement

Helm est un gestionnaire de paquets pour Kubernetes. Il permet d'installer, en quelques commandes, une application et ses dépendances. Le déploiement d'un chart helm dans le DataHub réside en la mise en place d'un unique manifeste `fleet.yaml` dans le dossier scruté par Rancher. Ce dernier se décompose en deux parties :

- la première section reprend les éléments propres au cluster. Destination et identification du namespace.
- la seconde, la description du chart helm et les surcharges nécessaires à son bon fonctionnement.

Ci-dessous un exemple pour le déploiement du chart "airflow".

```
# Target customization are used to determine how resources should be modified per target
# Targets are evaluated in order and the first one to match a cluster is used for that cluster.
targetRestrictions:
  - clusterGroup: recette
targets:
  - clusterGroup: recette
namespace: demo-datahub

helm:
  chart: airflow
  repo: "https://airflow.apache.org"
  releaseName: "apache-airflow"
  version: "1.11.0"
  namespace: votre-namespace
  values:
    ingress:
      enabled: true
      web:
       hosts: ["airflow.votre-namespace.datahub-recette.eco4.cloud.e2.rie.gouv.fr"]
```

3.2.2 5.2 Surcharge éventuelle des values liées au proxy

Les mentions relatives au proxy deviennent nécessaires lorsque lors de l'installation, vous devvez accéder à internet. Lors d'un `pip install` par exemple.

```
...
  values:
    ingress:
      enabled: true
      web:
       hosts: ["votrehost.votre-namespace.datahub-recette.eco4.cloud.e2.rie.gouv.fr"]
    api:
      env:
        http_proxy : "http://proxy.infra.cloud.e2.rie.gouv.fr:1080"
        https_proxy : "http://proxy.infra.cloud.e2.rie.gouv.fr:1080"
        no_proxy : "localhost,127.0.0.0/8,172.30.0.0/16,10.1.16.0/24,10.240.0.0/16,10.241.0.0/16,*.datahub-recette.eco4.cloud.e2.rie.gouv.fr"
    web:
      env:
        http_proxy : "http://proxy.infra.cloud.e2.rie.gouv.fr:1080"
        https_proxy : "http://proxy.infra.cloud.e2.rie.gouv.fr:1080"
        no_proxy : "localhost,172.30.0.0/16,127.0.0.0/8,10.1.16.0/24,10.240.0.0/16,10.241.0.0/16,*.datahub-recette.eco4.cloud.e2.rie.gouv.fr"
```

3.2.3 5.3 Application et charts helm

Il est possible de mixer une application avec un ou plusieurs charts helm, sans pour autant multiplier les projets GitLab.

4. 6. Points de vigilance

4.1 6.1 Sauvegarde du contenu des volumes

L'offre DataHub ne prévoit pas la sauvegarde des volumes déclarés à l'aide d'un manifeste de type `PersitentVolumeClaim`. Vous pouvez recourir à l'offre B3 pour effectuer cette opération.

4.2 6.2 Deployer l'image d'une application non personnalisée

Bien qu'à partir du dépôt officiel du DockerHub (https://hub.docker.com), le chemin des images génériques ne font pas figurer le dossier `library`, il est absolument nécessaire de le mentionner dans le manifeste yaml du DataHub du Ministère. Par exemple, pour nginx le lien officiel https://hub.docker.com/_/nginx deviendra dans le fichier lié au déploiement : http://harbor.datahub.eco4.cloud.e2.rie.gouv.fr/docker.io/library/nginx

5. A propos de cette documentation

5.1 Versions

Numero de version	Date	Commentaire
V05-2024	05/2024	+ Helm -NiFi + Réorganisation partielle
V10-2023	10/2023	+ Vigilance + URL d'exposition + quotas Mach.
V6-2023a	06/2023	Réorganisation
V6-2023	06/2023	Ajout Cas Usage R Shiny
V4-2023	04/2023	Développeur DataHub
V3-2023	03/2023	Cas usage Demo

5.2 Rédacteur(s)

- Cyril HOUISSE (MTE CT/SG/DNUM/MSP/DS/GD3IA)
- Jean-Philippe LANG (MTE CT/SG/DNUM/MSP/DS)

5.3 Relecteur(s)

- Guillaume PAYET (MTE CT/SG/DNUM/MSP/DS/GD3IA)
- Xavier Richard (MTE CT/SG/DNUM/MSP/DS/GD3IA)
- Jean-Philippe LANG (MTE CT/SG/DNUM/MSP/DS)

5.4 Eléments techniques

- Hébergement GitLab Pages.
- Génération des pages web MkDocs.
- Génération PDF mkdocs-with-pdf.

https://snum.gitlab-pages.din.developpement-durable.gouv.fr/ds/gd3ia/datahub_doc/