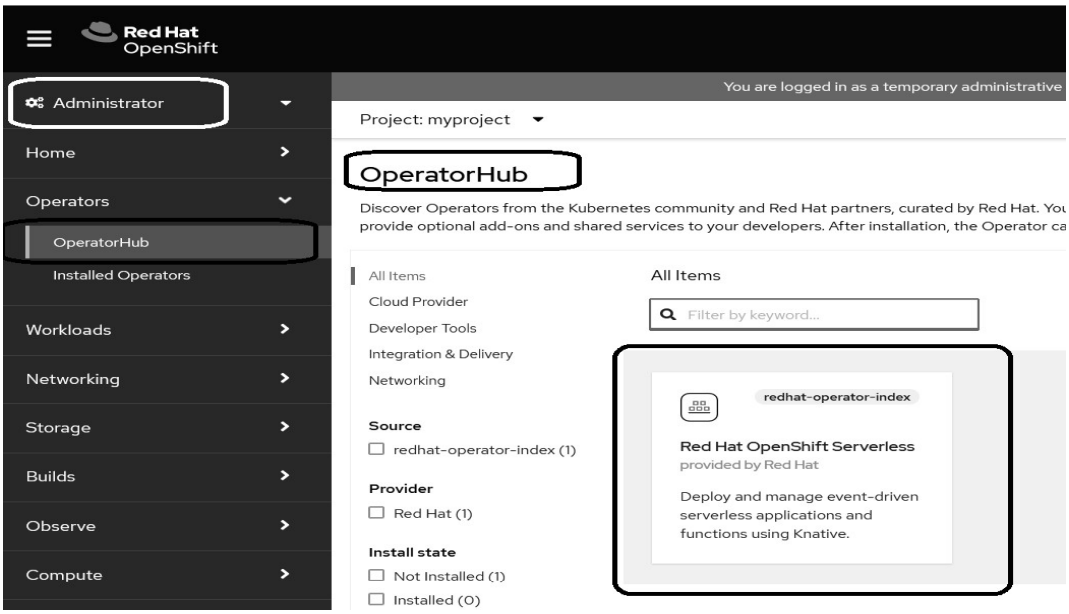
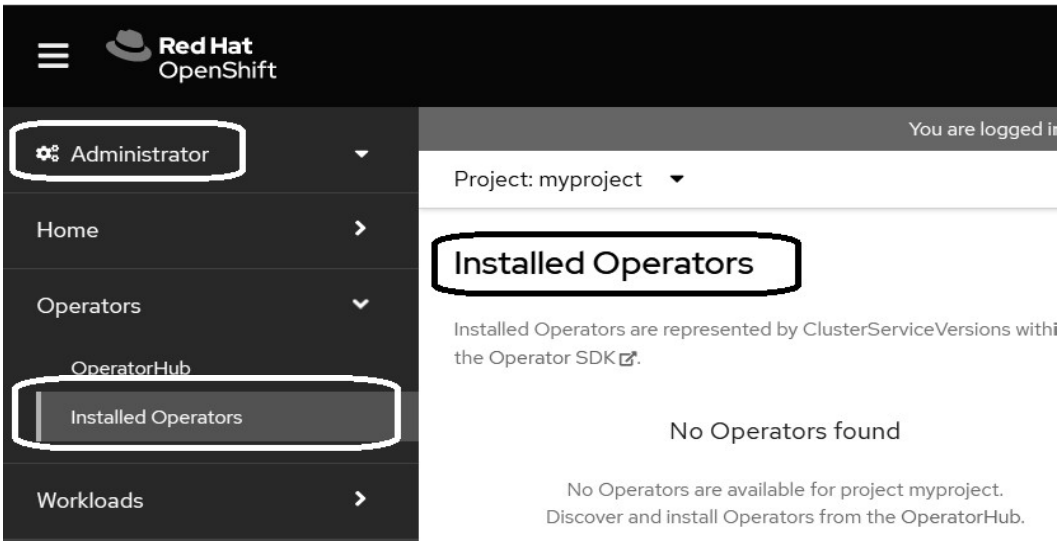
14 Les opérateurs

Les Opérateurs sont un concept majeur depuis la version 4.0 d'OpenShift. Les Opérateurs sont avant tout des opérateurs Kubernetes - pouvant fonctionner sur n'importe quelle distribution - mais OpenShift est la seule ayant été totalement architecturée autour de cette notion et offrant le plus de services pour tirer parti de cette technologie le plus facilement possible. Ainsi, OpenShift apporte la simplicité du cloud à travers **cette marketplace permettant de déployer simplement de nombreuses solutions conteneurisées,** tout en bénéficiant de tout le savoir-faire en matière d'exploitation de cette solution, grâce à la notion d'opérateur: en résumé, un "opérateur" **est un conteneur qui se déploie aux côtés de la solution qu'il gère, et intègre les bonnes pratiques d'exploitation qui sont automatisées et fournies par l'éditeur lui-même de cette solution.** Ainsi l'utilisateur consomme un service de manière simple, mais bénéficie d'une expertise nativement intégrée en simplifiant l'exploitation au maximum (installation, mises à jour, configuration, maintien en condition opérationnelle, etc.)..

On peut observer les opérateurs disponibles dans le menu suivant :



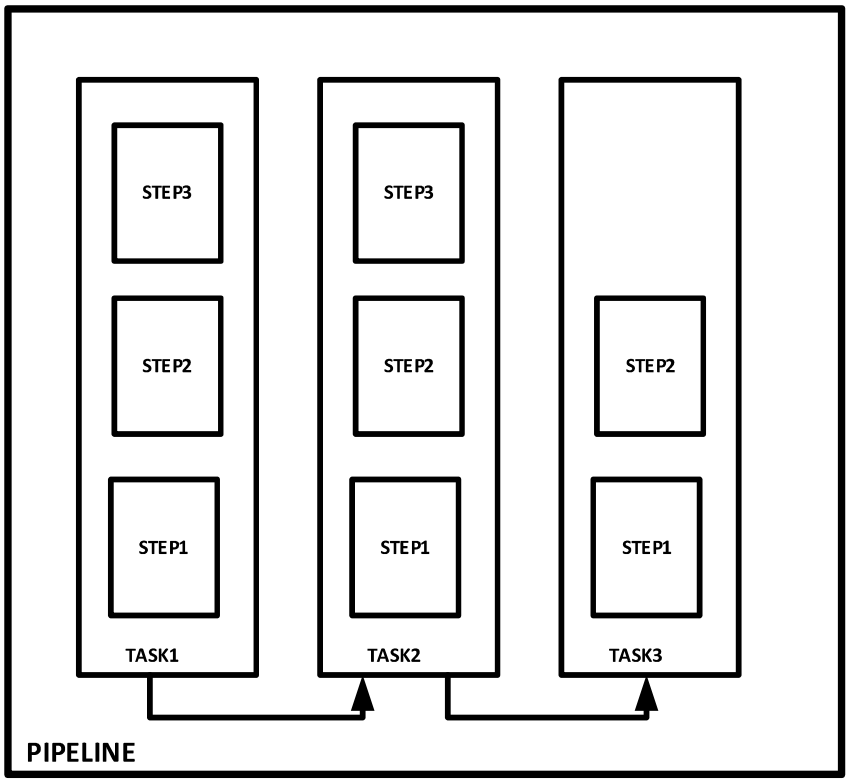
et les opérateurs installés :



Par la suite on va déployer une solution CI/CD avec l'opérateur pipeline tekton

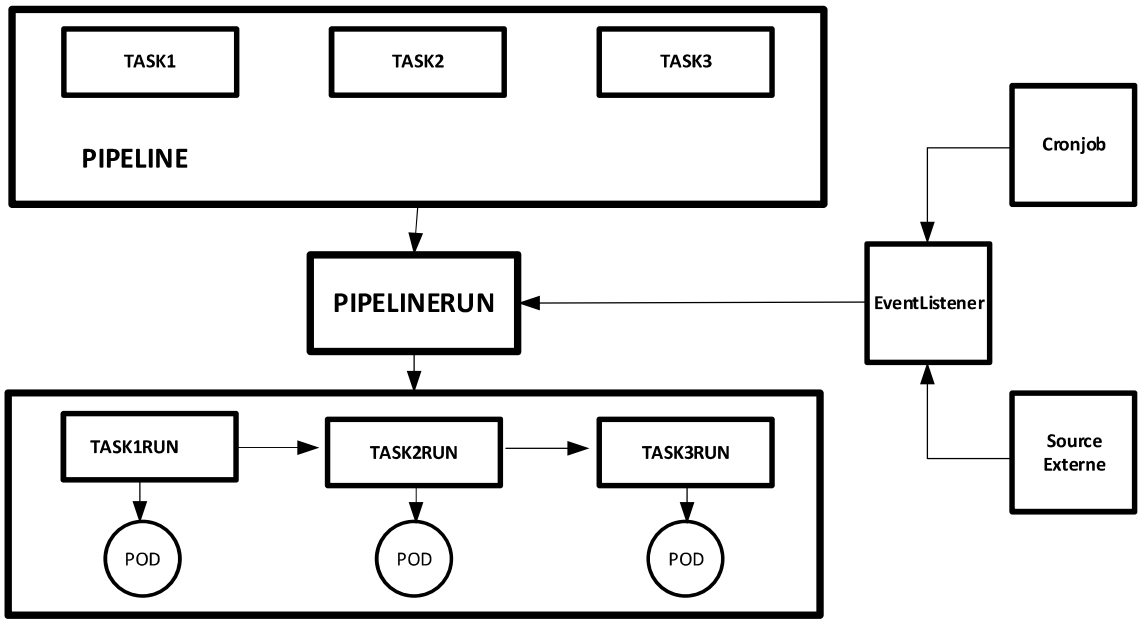
Principaux concepts

Red Hat OpenShift Pipelines est une solution CI/CD basée sur l'open source **Tekton** projet. L'objectif principal de Tekton est de permettre aux équipes DevOps de créer rapidement des pipelines pour des activités impliquant des étapes simples et reproductibles. Une caractéristique unique de Tekton qui le différencie des solutions CI/CD précédentes et que la procédure Tekton s'exécute dans un conteneur spécialement créé pour cette tâche. Cela fournit un degré d'isolation qui prend en charge l'exécution de tâches prévisibles et reproductibles et garantit que les équipes de développement n'ont pas à gérer une instance de serveur de build partagée.
Un pipeline est une séquence de taches pour représenter un flux de travail de développement logiciel (build, test, deploy) également connu **sous le nom d'intégration continue / déploiement continu (CI/CD).** Les ingénieurs DevOps cherchent toujours à automatiser ce flux de travail afin de minimiser les erreurs humaines, d'améliorer le temps de livraison des logiciels et de produire des logiciels cohérents.



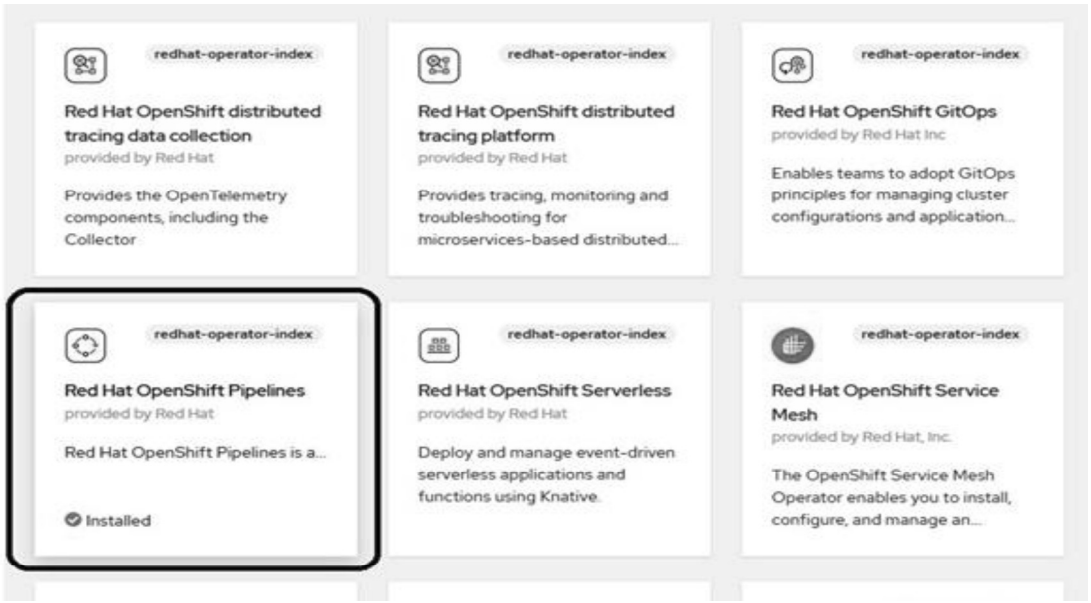
Tekton Pipeline définit un ensemble de concept, et d'objets Kubernetes (CRD) dont les principaux sont les suivants :
Step : C'est la plus petite unité d'exécution présente clans Tekton Pipeline, et correspond à un conteneur. C'est ici que nous allons pouvoir définir quelle commande doit être exécutée, avec quelle image de conteneur, ainsi que l'ensemble des options possible de passer à un conteneur dons Kubernetes. Elle fait partie de l'objet Task.
Task (CRD) : Une tache est un ensemble de Step qui s'exécute de manière séquentielle (l'une après l'autre). II est possible de définir des paramètres d'entrée, des Workspaces permettant de partager des données (entre taches...) ou d'y attacher des secrets ou configuration. II est également possible de définir des résultats, que les Step sont responsables de fournir, et qui peuvent être ensuite réutilisé (en tant que paramètres) par d'autres taches.
TaskRun (CRD) : Il s'agit de I 'exécution d'une Task. C'est sur cet objet que I 'on définit les paramètres et les données à utiliser. L'exécution va créer un Pod et monitorer son statut. PipelineRun (CRD) : II s'agit de

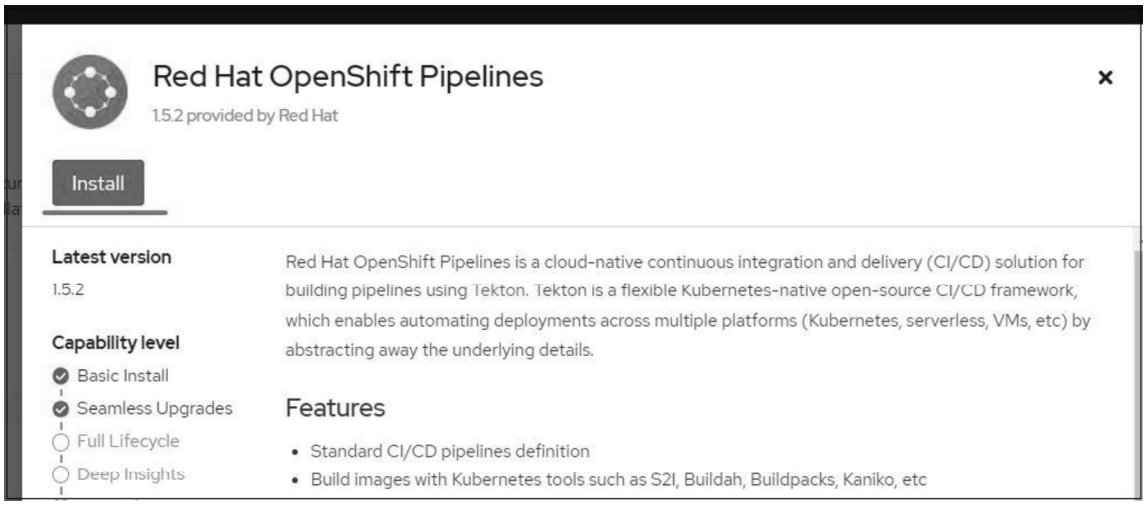
('exécution d'un Pipeline. A l 'instar d'un TaskRun, c' est sur cet objet que l'on définit les paramètres et les données à utiliser

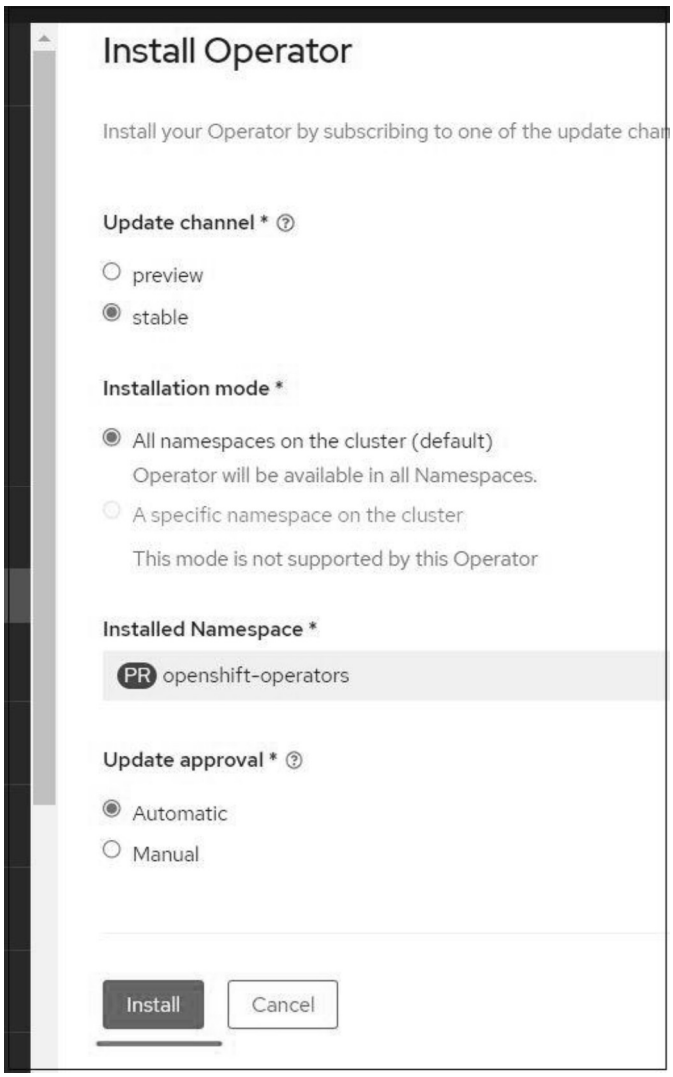


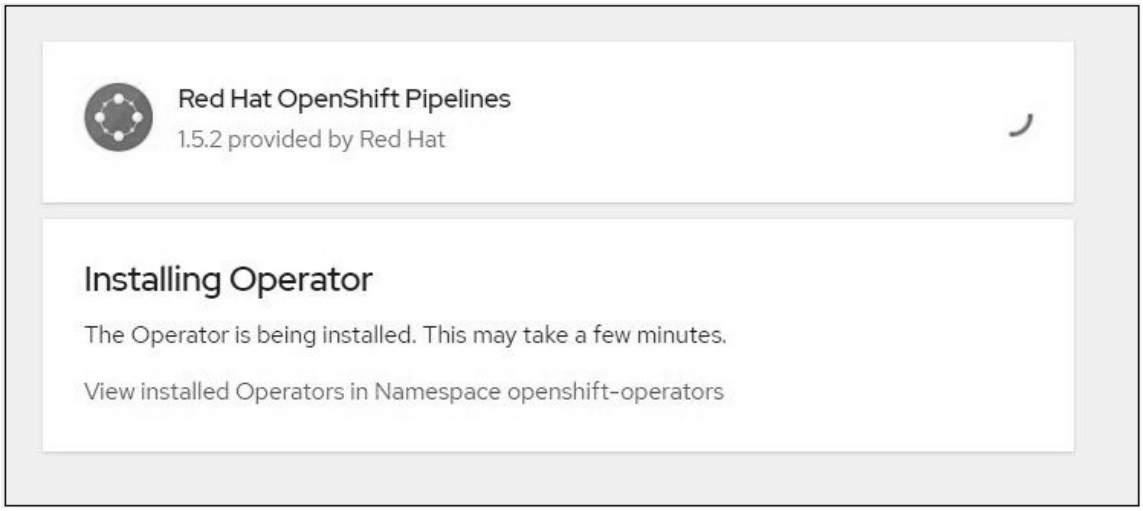
Installation de l'opérateur RED HAT OPENSHIFT PIPELINE

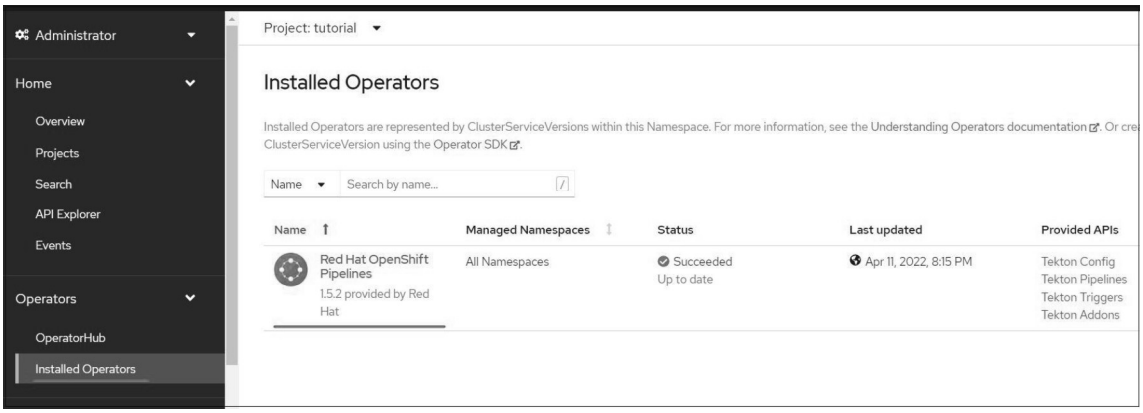
Dans la rubrique operators choisir l'opérateur RED HAT openshift Pipelines :



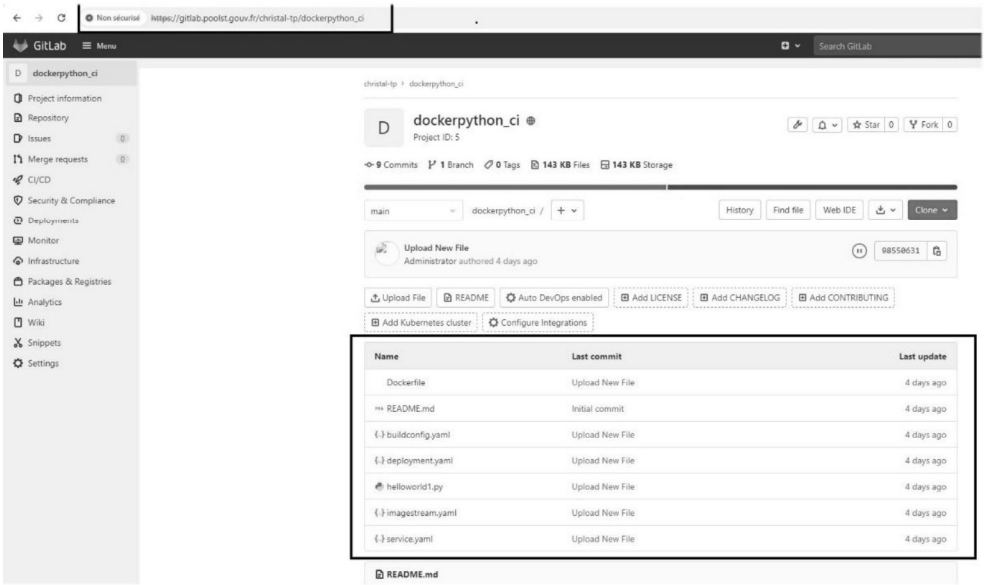

Red Hat OpenShift Pipelines
1.5.2 provided by Red Hat
Install
Latest version
1.5.2
Capability level
Basic Install
Seamless Upgrades
Full Lifecycle
Deep Insights
Red Hat OpenShift Pipelines is a cloud-native continuous integration and delivery (CI/CD) solution for building pipelines using Tekton. Tekton is a flexible Kubernetes-native open-source CI/CD framework, which enables automating deployments across multiple platforms (Kubernetes, serverless, VMs, etc) by abstracting away the underlying details.
Features
Standard CI/CD pipelines definition
Build images with Kubernetes tools such as S2I, Buildah, Buildpacks, Kaniko, etc


Install Operator
Install your Operator by subscribing to one of the update chan
Update channel *
preview
stable
Installation mode *
All namespaces on the cluster (default)
Operator will be available in all Namespaces.
A specific namespace on the cluster
This mode is not supported by this Operator
Installed Namespace *
PR openshift-operators
Update approval *
Automatic
Manual
Install
Cancel





Prendre connaissance du dépôt git dockerpython_ci qui sera la base de notre construction



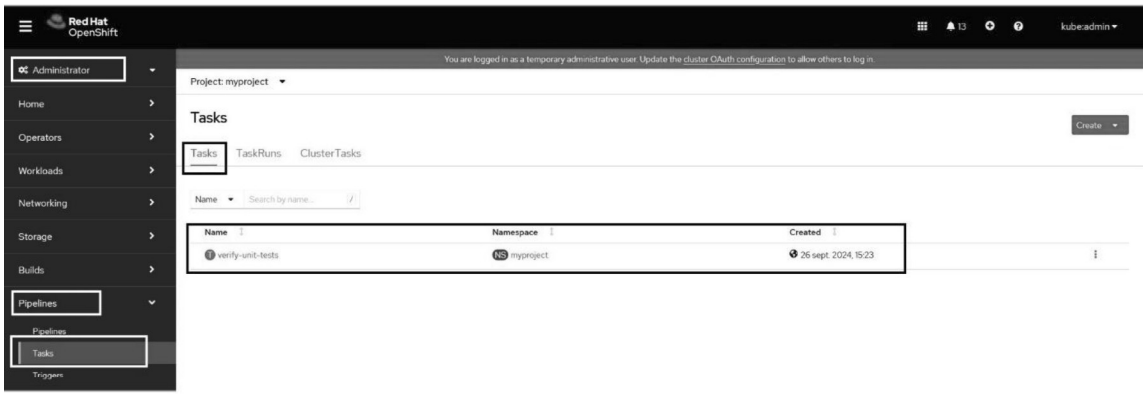
Créer une tache de test (verify-unit-tests) qui permet de vérifier si la chaine de compilation python est opérationnelle

Donnez le principe de ce test en analysant le fichier task-unit.yaml

Dans le répertoire **/root/elevex/tekton**

```
oc apply -f task-unit.yaml
```

Visualisez la création de la tache sur openshift



On peut par la suite lancez cette tâche par la commande suivante :

```
tkn task start verify-unit-tests –showlog
```

deux effets :

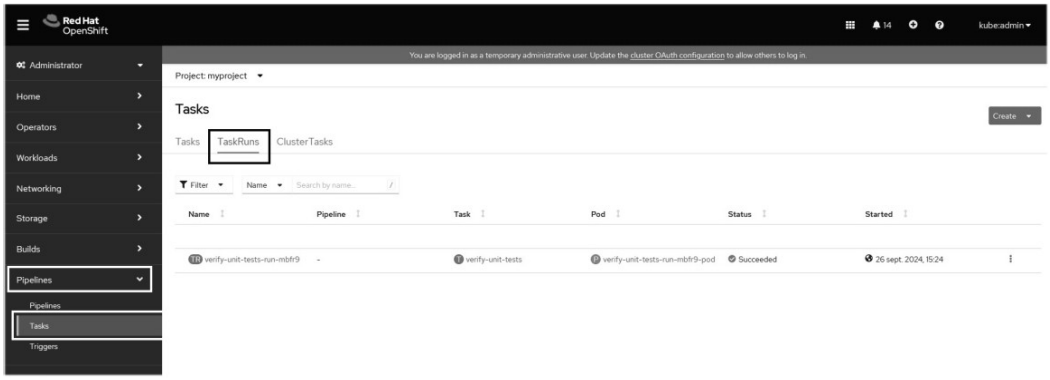
en ligne de commande

```
[root@ocbastion tekton_python]# tkn task start verify-unit-tests --showlog
TaskRun started: verify-unit-tests-run-mbfr9
Waiting for logs to be available...
[clonetestunit] testl
[clonetestunit] Cloning into '/workspace/src'...

[test-unit] Hello World Test! Built with a Docker file.

[root@ocbastion tekton_python]#
```

Sur l'IHM d'openshift:



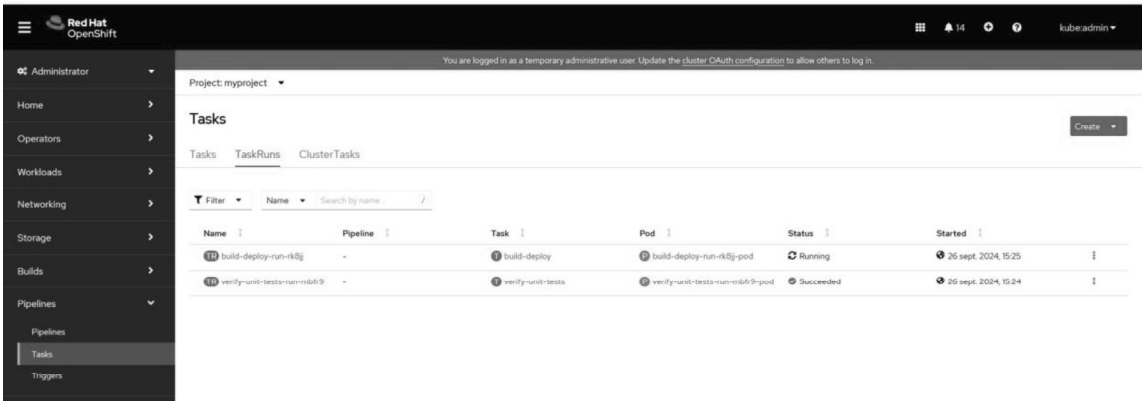
Dans le repertoire **/root/elevex/tekton/ task-deploy.yaml**
Déployez une tache de déploiement (build-deploy) qui permet de déployer le site web

```
oc apply -f task-deploy.yaml
```

Donnez le principe de ce déploiement en analysant le fichier task-unit.yaml



Lancez par la suite cette tache



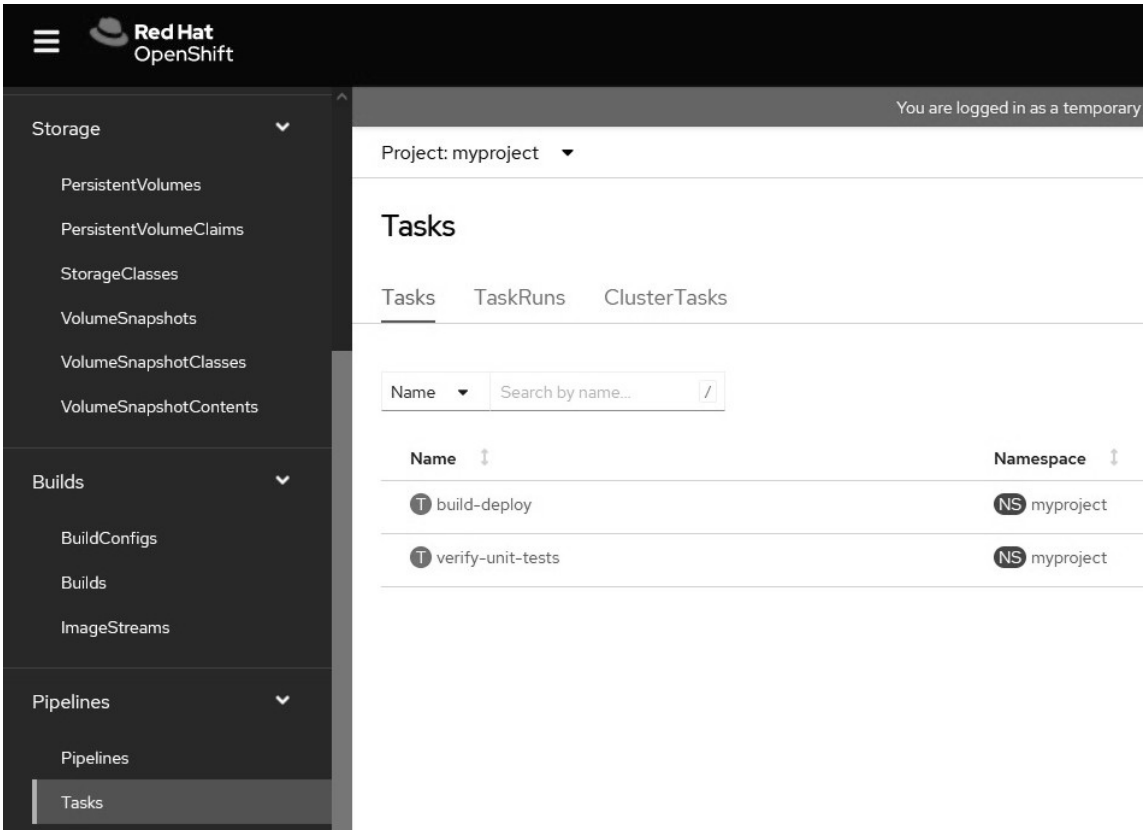
```
[root@ocbastion tekton_python]# tkn task start build-deploy --showlog
TaskRun started: build-deploy-run-rk8jj
Waiting for logs to be available...
[clonedeploy] Cloning into '/workspace/src'...
[clonedeploy] test1

[clonedeploy] /workspace/src
[clonedeploy] total 40 drwxr-sr-x. 4 root 1000680000 240 Sep 26 13:26 . drwxrwsrwx. 3 root 1000680000 17 Sep 26 13:26 .. drwxr-sr-x. 8 root 1000680000 163 Sep 26 13:26 .git -rw-r--r--. 1 root
-rw-r--r--. 1 root 1000680000 273 Sep 26 13:26 Dockerfile1 -rw-r--r--. 1 root 1000680000 16 Sep 26 13:26 README.md -rw-r--r--. 1 root 1000680000 896 Sep 26 13:26 buildconfig.yaml drwxr-sr-x. 2 root
--. 1 root 1000680000 912 Sep 26 13:26 deployment.yaml -rw-r--r--. 1 root 1000680000 625 Sep 26 13:26 helloworld.py -rw-r--r--. 1 root 1000680000 285 Sep 26 13:26 helloworld1.py -rw-r--r--. 1 root 1
yaml -rw-r--r--. 1 root 1000680000 13 Sep 26 13:26 requirements.txt -rw-r--r--. 1 root 1000680000 406 Sep 26 13:26 service.yaml
[clonedeploy] test2
[clonedeploy] /workspace/src
[clonedeploy] total 40 drwxr-sr-x. 4 root 1000680000 240 Sep 26 13:26 . drwxrwsrwx. 3 root 1000680000 17 Sep 26 13:26 .. drwxr-sr-x. 8 root 1000680000 163 Sep 26 13:26 .git -rw-r--r--. 1 root
-rw-r--r--. 1 root 1000680000 273 Sep 26 13:26 Dockerfile1 -rw-r--r--. 1 root 1000680000 16 Sep 26 13:26 README.md -rw-r--r--. 1 root 1000680000 896 Sep 26 13:26 buildconfig.yaml drwxr-sr-x. 2 root
--. 1 root 1000680000 912 Sep 26 13:26 deployment.yaml -rw-r--r--. 1 root 1000680000 625 Sep 26 13:26 helloworld.py -rw-r--r--. 1 root 1000680000 285 Sep 26 13:26 helloworld1.py -rw-r--r--. 1 root 1
yaml -rw-r--r--. 1 root 1000680000 13 Sep 26 13:26 requirements.txt -rw-r--r--. 1 root 1000680000 406 Sep 26 13:26 service.yaml
[clonedeploy] test3
[clonedeploy] imagestream.image.openshift.io/deploypython created
[clonedeploy] buildconfig.build.openshift.io/deploypython created
[clonedeploy] test4
[clonedeploy] build.build.openshift.io/deploypython-2 started
[clonedeploy] Cloning "https://gitlab.poolst.gouv.fr/christal-tp/dockerpython.git" ...
[clonedeploy]     Commit: a966c46c57c1cbfbe92d4e25e9ee4fc132610791 (Upload New File)
[clonedeploy]     Author: Administrator <admin@example.com>
[clonedeploy]     Date:   Thu Sep 26 12:52:57 2024 +0000
[clonedeploy] time="2024-09-26T13:26:08Z" level=info msg="Not using native diff for overlay, this may cause degraded performance for building images: kernel has CONFIG_OVERLAY_FS_REDIRECT_DIR
[clonedeploy] I0926 13:26:08.969873       1 defaults.go:112] Defaulting to storage driver "overlay" with options [mountopt=metacopy=on].
[clonedeploy] Caching blobs under "/var/cache/blobs".

[clonedeploy] Pulling image ocbastion.poolst.gouv.fr:8443/mirror/oc-mirror-metadata/alpine_pip3:v1 ...
[clonedeploy] Trying to pull ocbastion.poolst.gouv.fr:8443/mirror/oc-mirror-metadata/alpine_pip3:v1...
[clonedeploy] Getting image source signatures
[clonedeploy] Copying blob sha256:6fc521cd1e5f501a6dd23630851581 3eb214699c99e0b268d8a5b883fae7896c
```

Supprimez tous les éléments (le déploiement, les pods, le service , la route) liées à ces deux taches dans la rubrique developer project (projet myproject) supprimez le builds , l'imagestream et les deux taskrun

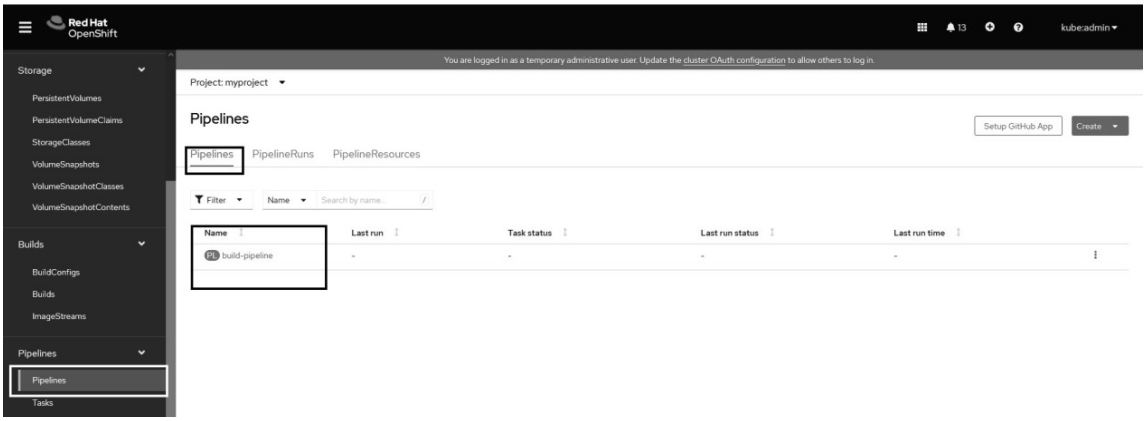
On peut associer ces deux taches de manière chronologique en créant un pipeline de tache



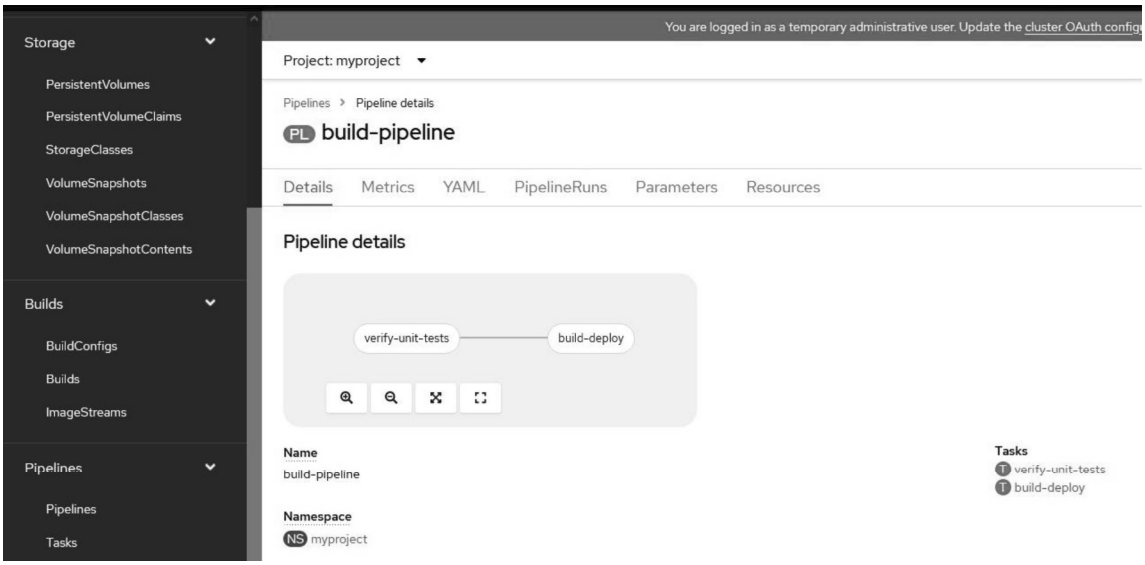
En analysant la fichier pipeline.yaml donnez quelle tache va être exécutée avant l'autre

Déployez ce pipeline dans **/root/elevex/ pipeline.yaml**

```
oc apply -f pipeline.yaml
```



On voit apparaitre la chronologie des deux taches



Lancez le démarrage du pipeline

```
tkn pipeline start build-pipeline --showlog
```

le déroulé dans les logs des différentes étapes

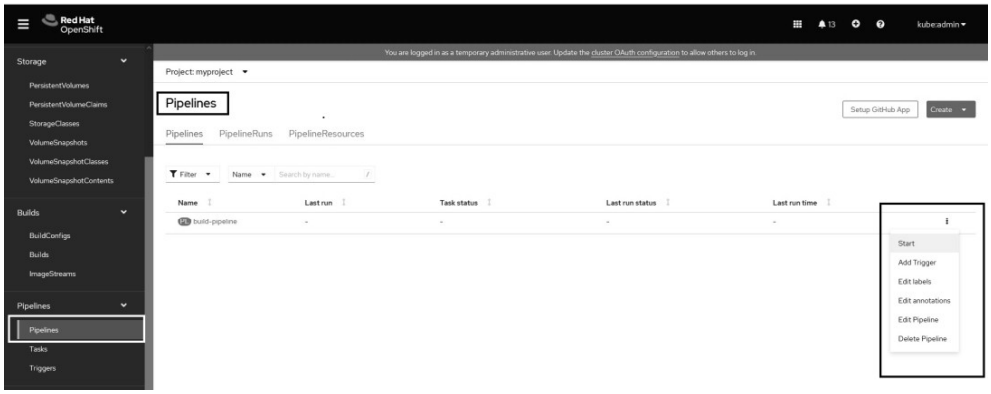
```
[verify-unit-tests : clonetestunit] test1
[verify-unit-tests : clonetestunit] Cloning into '/workspace/src'...

[verify-unit-tests : test-unit] Hello World Test! Built with a Docker file.

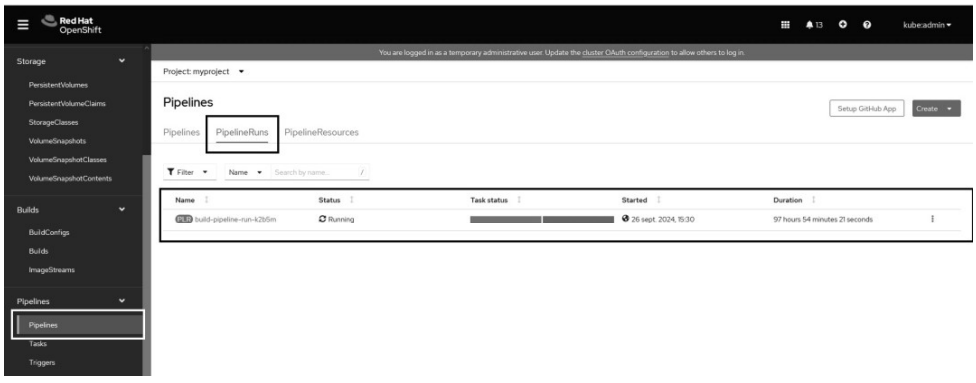
[build-deploy : clonedeploy] Cloning into '/workspace/src'...
[build-deploy : clonedeploy] test1

[build-deploy : configure-project] /workspace/src
[build-deploy : configure-project] total 40 drwxr-sr-x. 4 root 1000680000 240 Sep 26 13:30 . drwxrwsrwx. 3 root 1000680000 17 Sep 26 13:30 .
:30 Dockerfile -rw-r--r--. 1 root 1000680000 273 Sep 26 13:30 Dockerfile1 -rw-r--r--. 1 root 1000680000 16 Sep 26 13:30 README.md -rw-r--r--
:30 ci -rw-r--r--. 1 root 1000680000 912 Sep 26 13:30 deployment.yaml -rw-r--r--. 1 root 1000680000 625 Sep 26 13:30 helloworld.py -rw-r--r--
30 imagestream.yaml -rw-r--r--. 1 root 1000680000 13 Sep 26 13:30 requirements.txt -rw-r--r--. 1 root 1000680000 406 Sep 26 13:30 service.yaml
[build-deploy : configure-project] test2
[build-deploy : configure-project] /workspace/src
[build-deploy : configure-project] total 40 drwxr-sr-x. 4 root 1000680000 240 Sep 26 13:30 . drwxrwsrwx. 3 root 1000680000 17 Sep 26 13:30 .
:30 Dockerfile -rw-r--r--. 1 root 1000680000 273 Sep 26 13:30 Dockerfile1 -rw-r--r--. 1 root 1000680000 16 Sep 26 13:30 README.md -rw-r--r--
:30 ci -rw-r--r--. 1 root 1000680000 912 Sep 26 13:30 deployment.yaml -rw-r--r--. 1 root 1000680000 625 Sep 26 13:30 helloworld.py -rw-r--r--
30 imagestream.yaml -rw-r--r--. 1 root 1000680000 13 Sep 26 13:30 requirements.txt -rw-r--r--. 1 root 1000680000 406 Sep 26 13:30 service.yaml
[build-deploy : configure-project] test3
[build-deploy : configure-project] imagestream.image.openshift.io/deploypython created
[build-deploy : configure-project] buildconfig.build.openshift.io/deploypython created
[build-deploy : configure-project] test4
[build-deploy : configure-project] build.build.openshift.io/deploypython-2 started
[build-deploy : configure-project] Cloning "https://gitlab.poolst.gouv.fr/christal-tp/dockerpython.git" ...
[build-deploy : configure-project]      Commit: a966c46c57c1cbfbe92d4e25e9ee4fc132610791 (Upload New File)
[build-deploy : configure-project]      Author: Administrator <admin@example.com>
[build-deploy : configure-project]      Date:   Thu Sep 26 12:52:57 2024 +0000
[build-deploy : configure-project] time="2024-09-26T13:30:41Z" level=info msg="Not using native diff for overlay, this may cause degraded pe
[build-deploy : configure-project] I0926 13:30:41.133798       1 defaults.go:112] Defaulting to storage driver "overlay" with options [mount
[build-deploy : configure-project] Caching blobs under "/var/cache/blobs".
[build-deploy : configure-project]
[build-deploy : configure-project] Pulling image ocbastion.poolst.gouv.fr:8443/mirror/oc-mirror-metadata/alpine_pip3:v1 ...
[build-deploy : configure-project] Trying to pull ocbastion.poolst.gouv.fr:8443/mirror/oc-mirror-metadata/alpine_pip3:v1...
[build-deploy : configure-project] Getting image source signatures
[build-deploy : configure-project] Copying blob sha256:6fc521cd1e5f501a6dd236308515813eb214699c99e0b268d8a5b883fae7896c
```

on peut aussi lancez le démarrage du pipeline de tâche par openshift



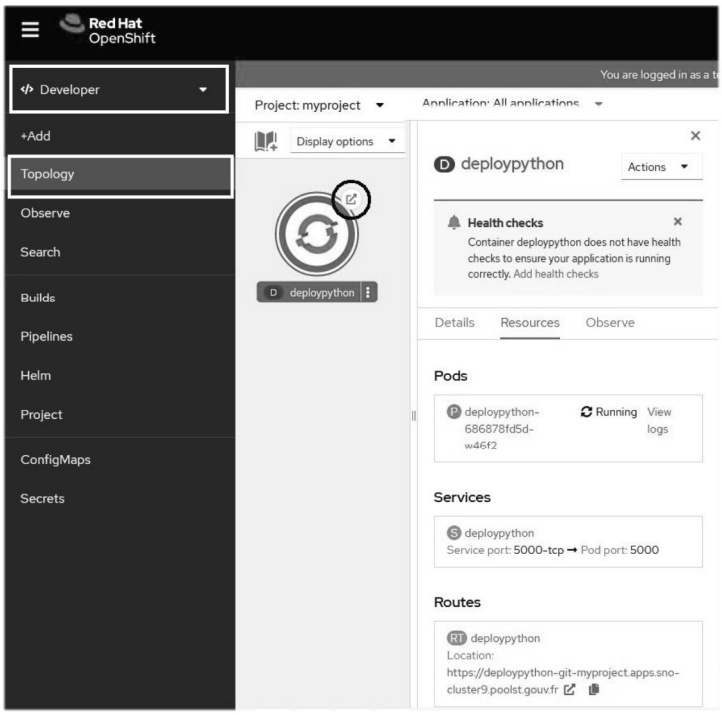
On observe la progression des deux taches

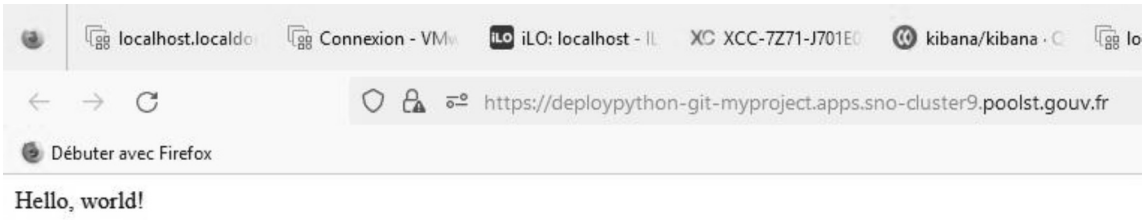


La fin des deux taches

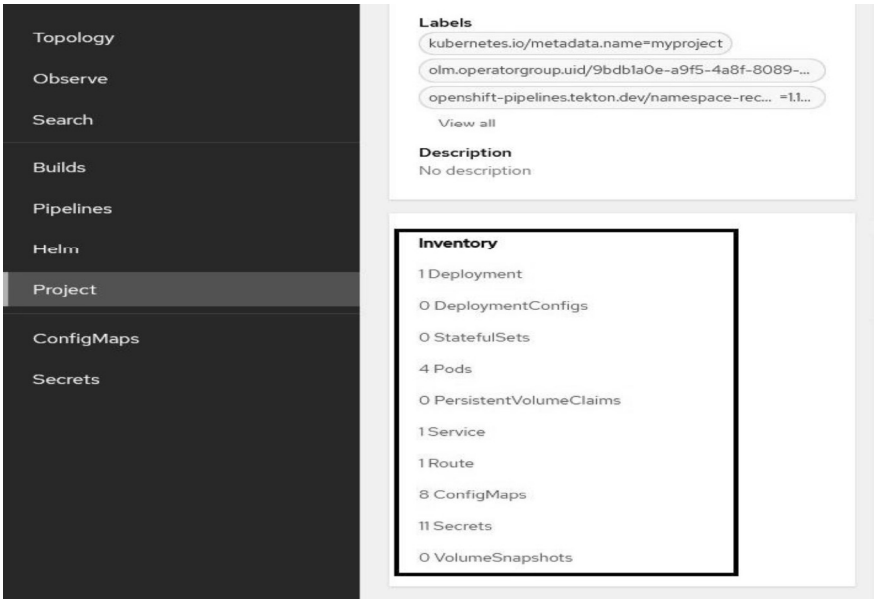


Cliquez sur la route de notre site web (flèche dans l'encadré)



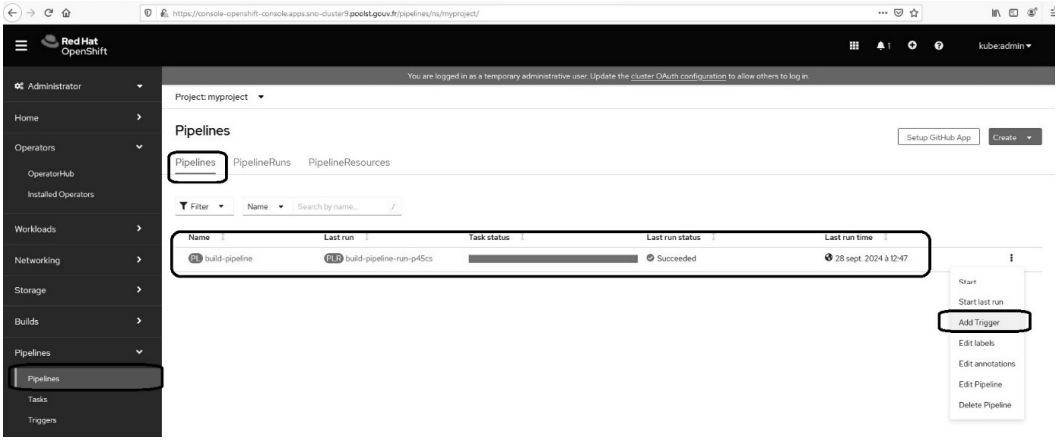


Beaucoup d'éléments dans notre projet

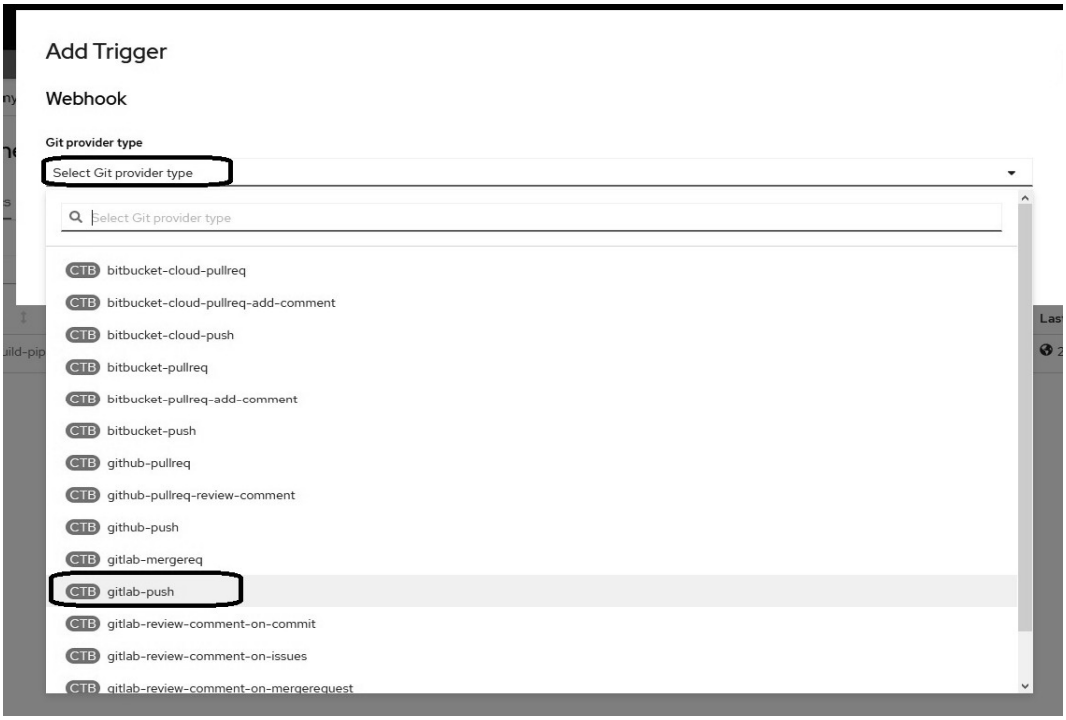


Les triggers

Automatisez la création d'images Docker à l'aide des déclencheurs OpenShift Pipeline et des webhooks Gitlab. Les déclencheurs capturent les événements externes et les traitent pour en extraire les informations clés. Une tâche PipelineRun doit s'exécuter automatiquement pour toute nouvelle modification de code dans le référentiel Git et c'est ainsi que nous pouvons réaliser l'automatisation du pipeline que nous avons créé précédemment à l'aide d'OpenShift Pipelines.
Les déclencheurs automatisent ce processus en capturant et en traitant tout événement de modification et en déclenchant une tâche PipelineRun qui déploie la nouvelle image avec les dernières modifications à partir de votre référentiel Git.
Configurer des déclencheurs de pipeline
Dans la vue Pipelines, cliquez sur l'icône des trois points verticaux en regard du pipeline, puis cliquez sur Ajouter un déclencheur.



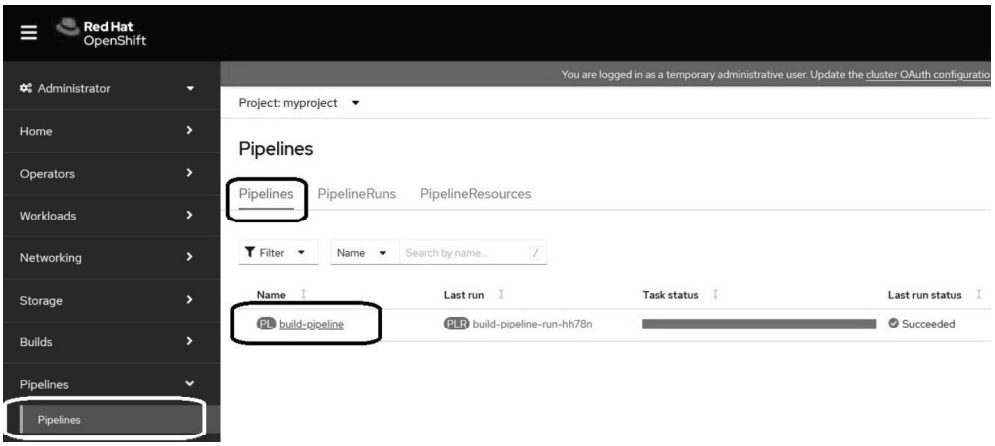
Sur la page Ajouter un déclencheur, entrez les valeurs des champs suivants :



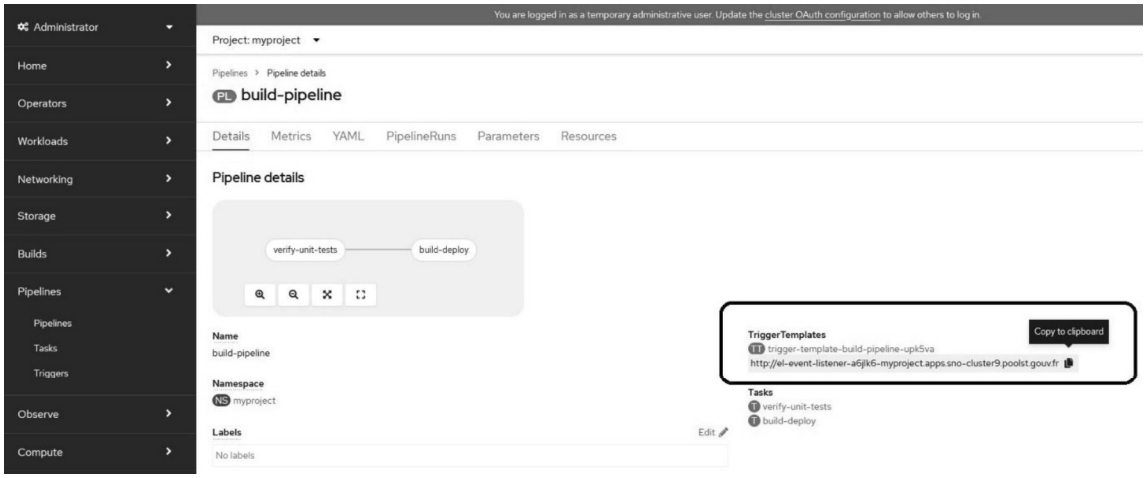
Dans la liste déroulante Type de fournisseur Git, sélectionnez github-push.



Cliquez ensuite sur Ajouter.
Sur la page Pipelines, cliquez sur le pipeline.

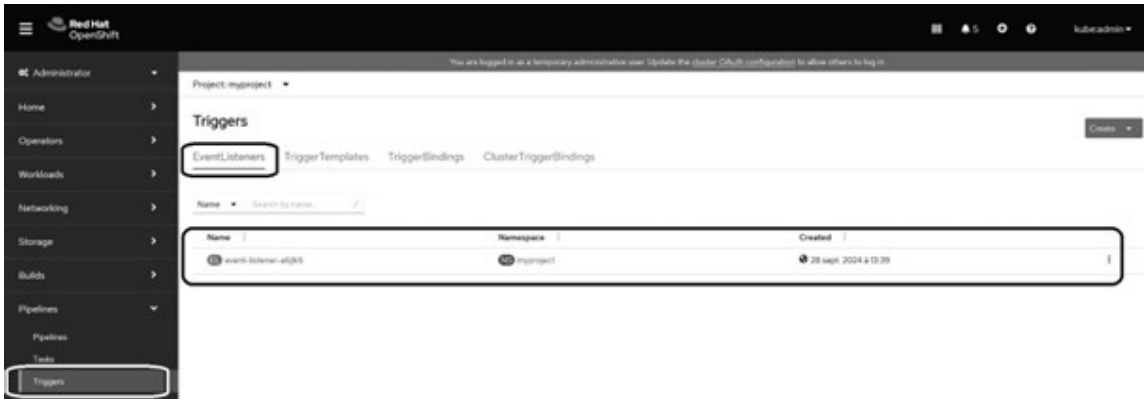


Sur la page Détails du pipeline, vous pouvez voir que l'URL de route HTTP de l'écouteur d'événements a été créée. L'écouteur d'événements est un composant du déclencheur de pipelines qui écoute les événements externes.



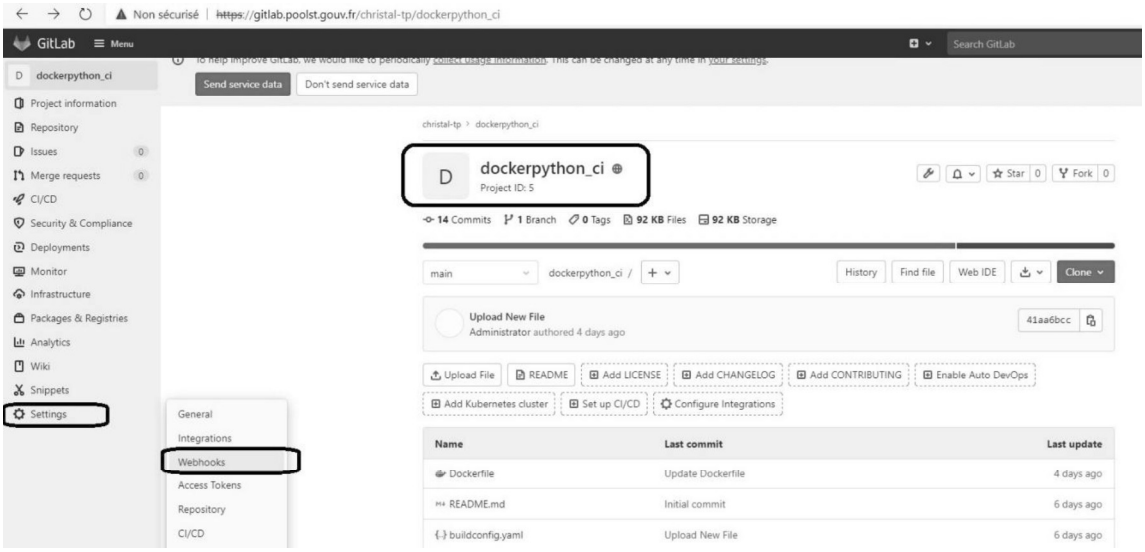
Copiez l'URL HTTP (applicable uniquement si votre cluster OpenShift prend en charge les routes HTTP, et si ce n'est pas le cas, accédez à la section Créer une route HTTPS d'écouteur d'événements et copiez l'URL HTTPS) et enregistrez-la pour une utilisation ultérieure. Cette URL (HTTP ou HTTPS, selon le cas) sera utilisée comme URL de charge utile dans la configuration des webhooks Gitlab (vous trouverez plus de détails dans les rubriques suivantes).

Un eventListener est créé dans la rubrique triggers

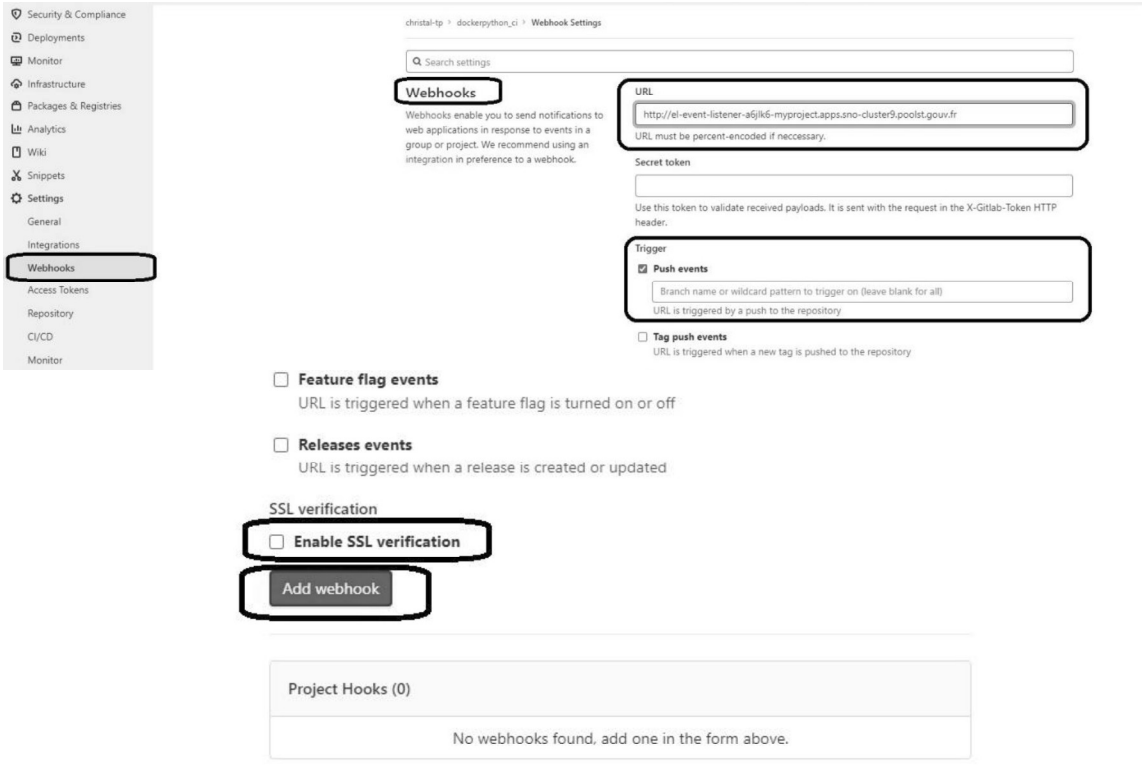


Les webhooks Gitlab permettent aux services externes d'être avertis lorsque certains événements Gitlab se produisent. Nous sommes intéressés par l'événement **git push**. Dans cette procédure, nous allons configurer les webhooks Gitlab avec l'URL HTTP ou HTTPS de l'écouteur d'événements (le cas échéant) comme URL de charge utile, de sorte que les modifications apportées au code source Gitlab seront notifiées à l'écouteur d'événements, ce qui permettra de déclencher un nouveau travail PipelineRun. Accédez à votre référentiel Gitlab dans le navigateur

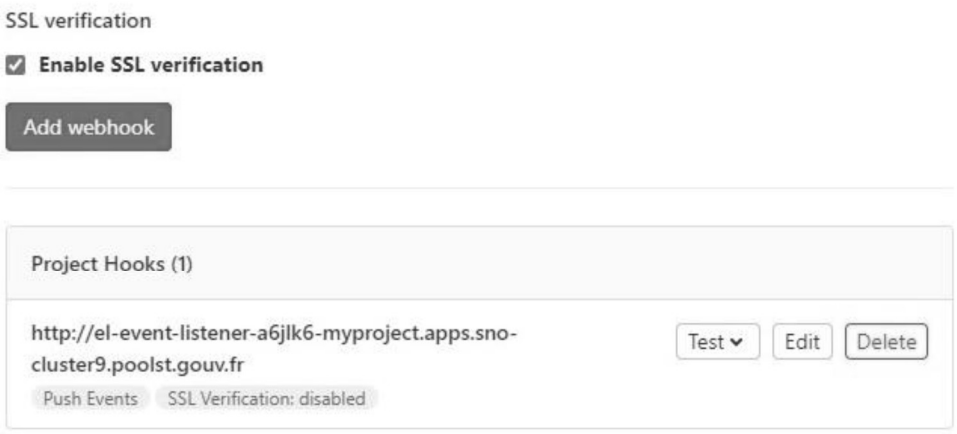
Cliquez sur l'onglet Paramètres, puis sur Webhooks.



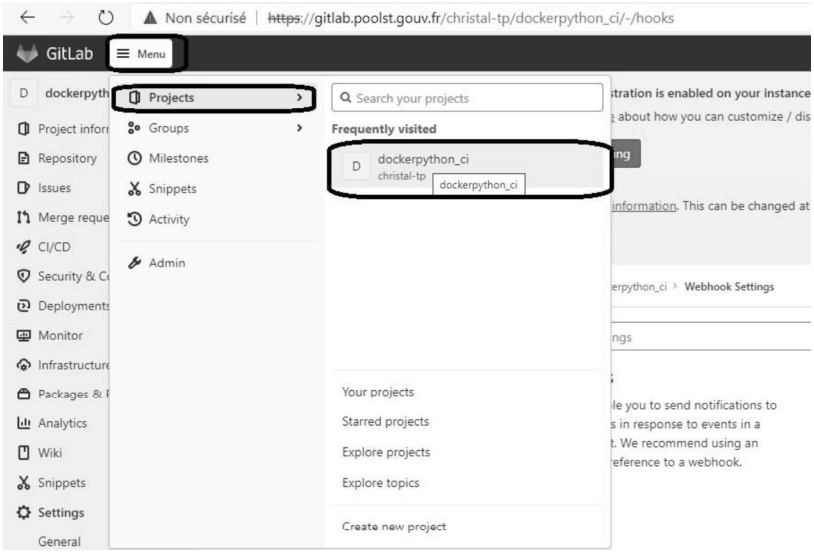
Cliquez sur Ajouter un webhook.
Sur la page Ajouter un webhook, saisissez les données nécessaires dans les champs suivants :

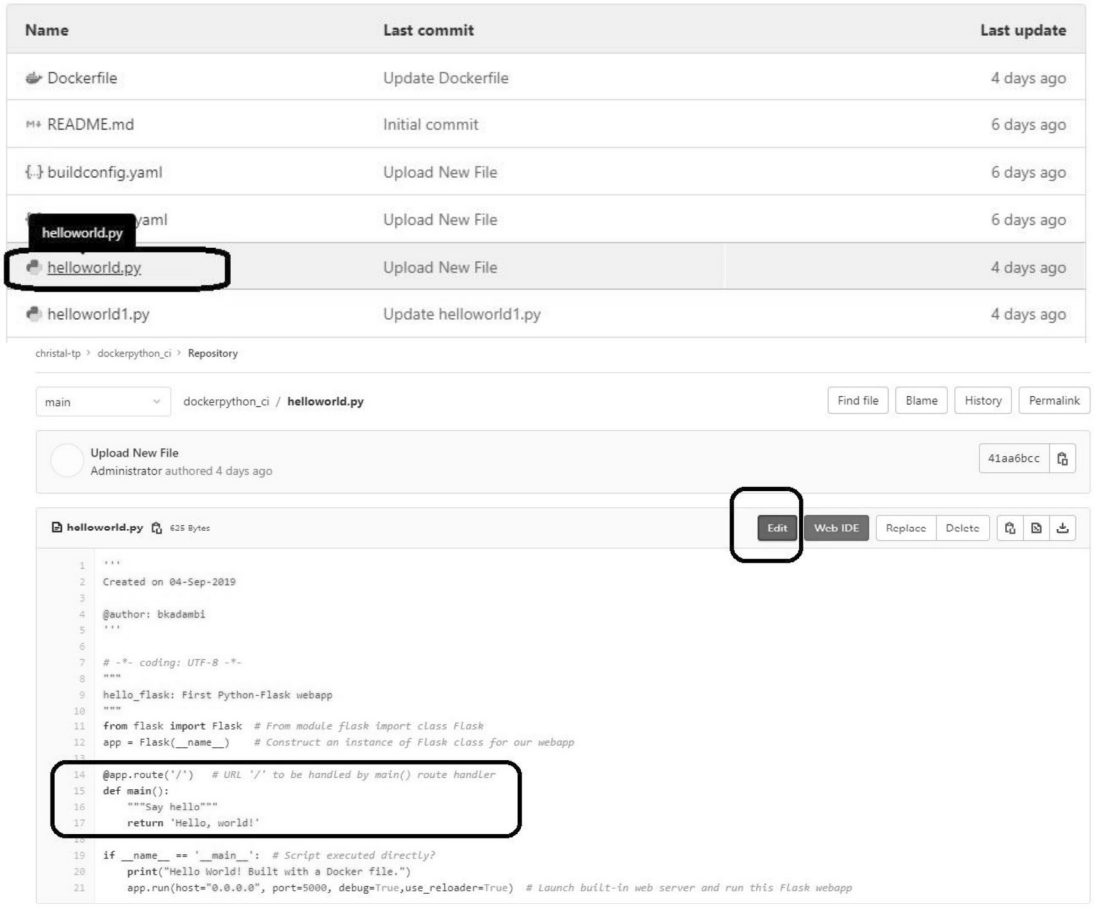


Dans le champ URL de la charge utile, entrez l'URL de route de l'écouteur d'événements HTTP ou HTTPS (le cas échéant) (dans mon cas, il s'agit d'une URL HTTPS).
Vous pouvez conserver les valeurs par défaut pour les champs restants, puis cliquer sur Ajouter un webhook.

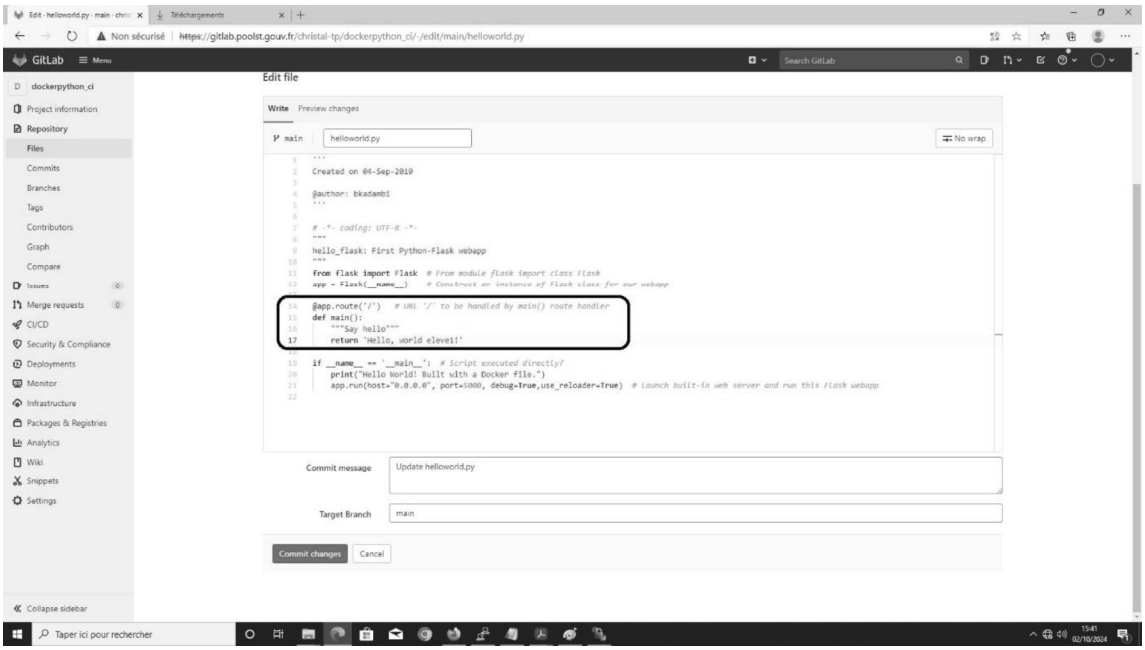


Un nouveau webhook sera créé (la dernière entrée si vous avez plusieurs webhooks). Il sera accompagné d'une coche (vous devrez peut-être actualiser la page des webhooks au cas où vous ne la verriez pas automatiquement), ce qui indique que GitHub est en mesure d'envoyer une requête ping ou de se connecter à votre cluster OpenShift à l'aide de l'URL de route HTTP ou HTTPS de l'écouteur d'événements (selon le cas pour votre cluster).
Le webhook Gitlab est maintenant connecté à votre cluster OpenShift et toute modification apportée au code source Gitlab déclenchera un nouveau travail PipelineRun.
Vérifiez si une modification du code Gitlab crée une nouvelle image Docker
Apportons une petite modification de code dans notre référentiel GitHub et vérifions s'il crée effectivement une nouvelle tâche PipelineRun, qui à son tour crée une nouvelle image Docker de notre application.
Accédez à votre référentiel source Gitlab, cliquez sur helloworld.py et modifiez-le en cliquant sur edit.



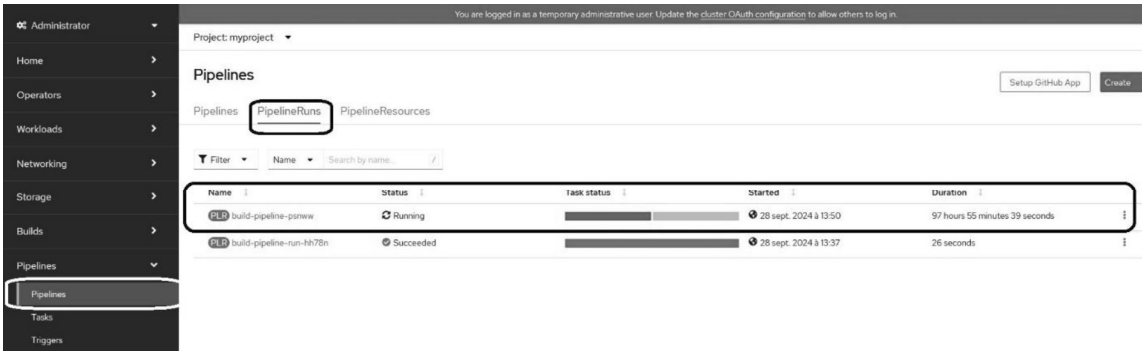


En mode d'édition, effectuez les modifications suivantes :



Modifiez le message de bienvenue et cliquez sur commit changes.

Lorsqu'il est validé avec succès, une nouvelle tâche PipelinRun est déclenchée. Accédez à votre console OpenShift, cliquez sur Pipeline, puis sur votre pipeline. Dans la vue Détails du pipeline, cliquez sur l'onglet PipelineRuns.

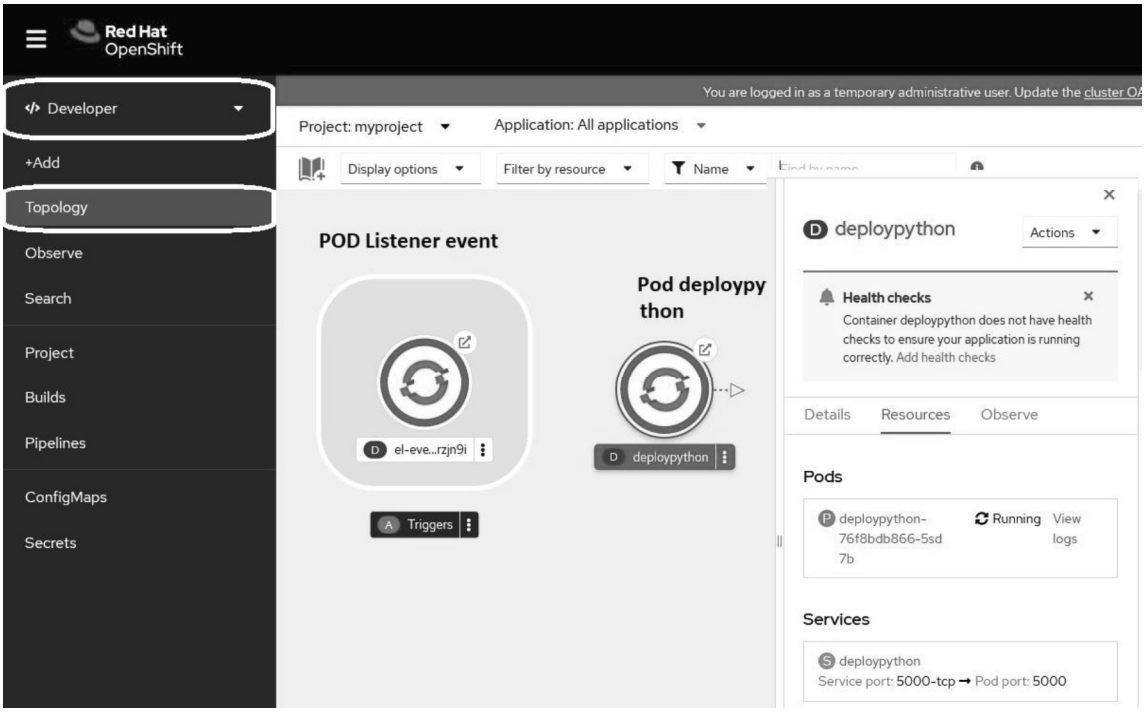


Notez qu'une nouvelle tâche PipelineRun est en cours d'exécution.

Attendez la fin de la tâche PipelineRun.



Vérifiez la création des deux pods suivants :



Cliquez sur l'URL (flèche encadrée)



Le site web fait transparaitre la modification de notre page web

