Implementing Continuous Integration with GitHub Actions

Introduction

GitHub is a web-based platform for version control and collaboration on software development projects. GitHub Actions is an automation and continuous integration and deployment (CI/CD) platform provided by GitHub. A runner is software that runs on a virtual machine or physical hardware and executes the actions defined in your GitHub Actions workflows. This CI/CD accelerator helps you implement a basic Talend CI process with Talend 8.*x* on the GitHub Actions technology stack using a runner.

For use in conjunction with the Talend **github_pipeline_yml.zip** archive file.

Prerequisites

Recommended training

- Talend Academy training:
  - Introduction to Data Integration
  - Introduction to Talend Cloud
- Knowledge of CI/CD
- Knowledge of GitHub and GitHub Actions runners

Software requirements

- Access to the **github_pipeline_yml.zip** file containing the **github-pipeline.yml**, **maven_settings.xml**, and **sample_pom.xml** files
- GitHub repository with admin rights on GitHub projects, also known as repositories
- Maven binaries (for version 8.0.4 and lower)
- Java (OpenJDK or OracleJDK version 11.*x*)
- Talend Cloud account to manage projects
- Talend Studio
- Talend platform license file (for on-premises and cloud)

Important: From Talend download version 8.0.4 onward, Talend CI Builder is available in the Talend repository. You must download and install the CI Builder manually if you use a lower version.

Technical prerequisites

- Minimum system requirements:

OS	CPU	RAM	SSD Disk Size
Windows/Linux/Mac	4 Cores or equivalent	4 GB (minimum)	50 GB (minimum recommended)

Environment

The following accounts are required to proceed with the setup:

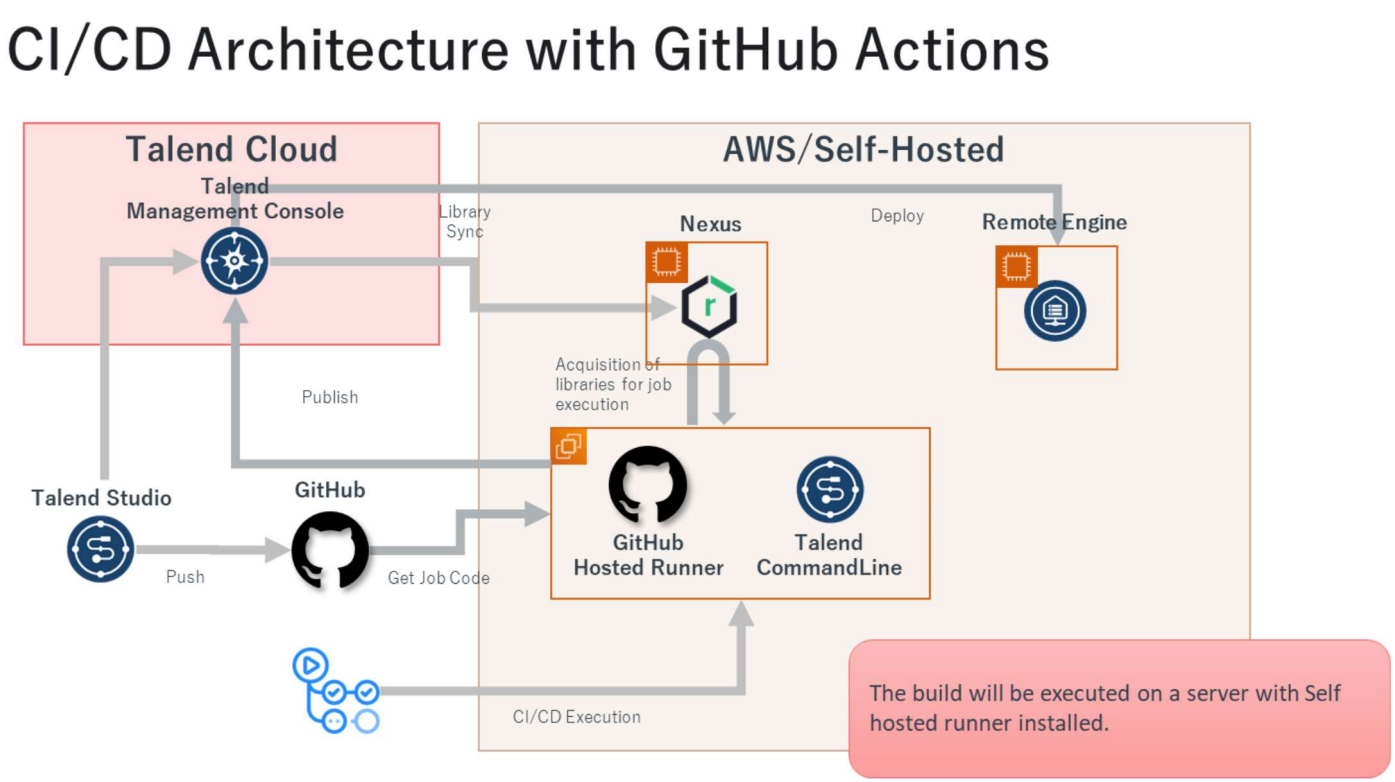
- GitHub account
- Talend Cloud account

Contents

Overview

A large variety of continuous integration (CI) server technologies are available in the CI domain. Some of the CI server technologies are old, some are new, some are mature, some are compliant with industry standards, and some have attractive user interfaces (UI). Talend CI is not coupled explicitly to any of them and can run on any application supporting Apache Maven 3.*x*, a 20+-year-old standard that emerged from a common industry practice based on Apache Ant.

The chosen CI server technology does not impact how Talend CI works with Maven. However, using this step-by-step guide focused on helping you implement CI with a GitHub Actions runner will save time.



Getting started

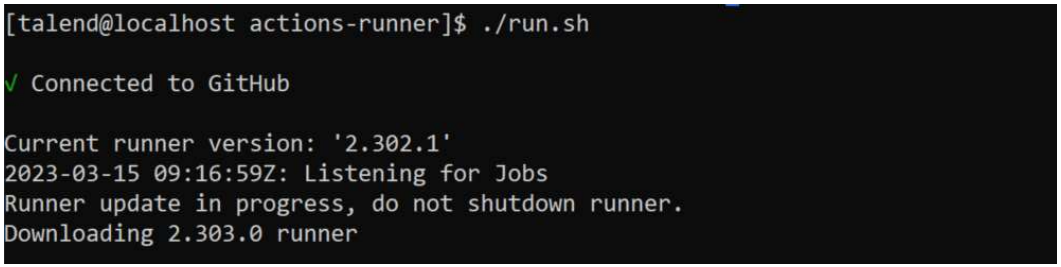
Setting up your environment

Adapt the following files to fit your environment:

- **Maven_settings.xml**: Replace the parameter values (Nexus or Artifactory URLs, hostnames, ports, credentials, and so on) and save the changes.
- **github-pipeline.yml**: Customize the Maven commands and options.

Configuring a GitHub Actions runner

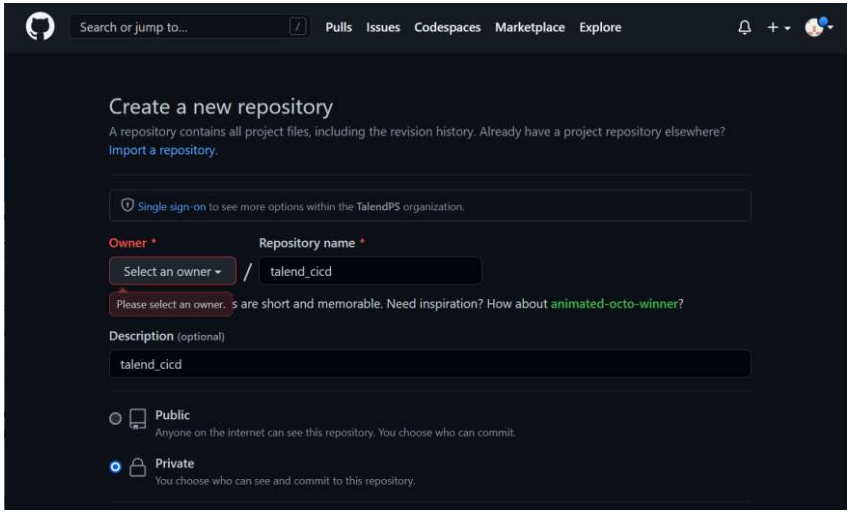
Runners are the machines that execute jobs in a GitHub Actions workflow. GitHub (Github-hosted runners) provides these machines, or you can host them yourself (self-hosted runners). This implementation accelerator uses a self-hosted runner.



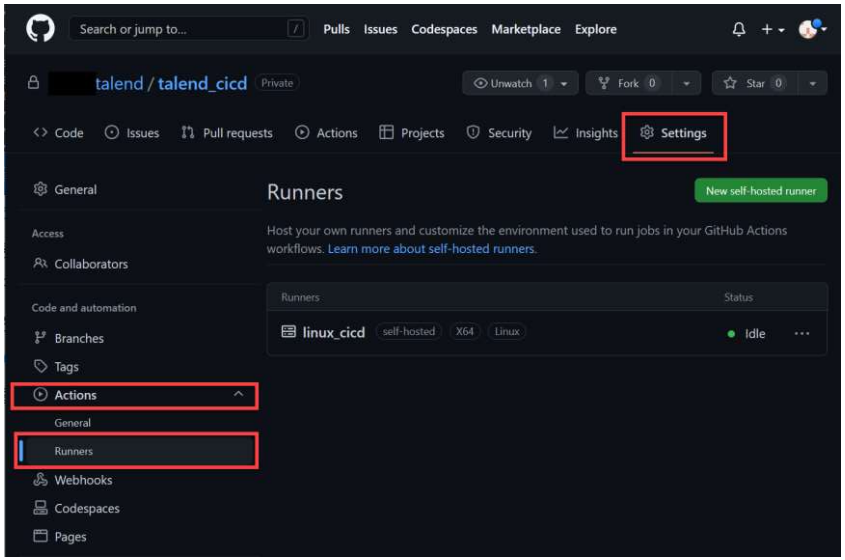
Warning: When a new version is released, a self-hosted runner automatically updates itself when a job is assigned to the runner or within a week of release if the runner hasn't been assigned any jobs. A self-hosted runner is automatically removed from GitHub if it has not connected to GitHub Actions for more than 14 days.

For more information, see the GitHub documentation page, Differences between GitHub-hosted and self-hosted runners.

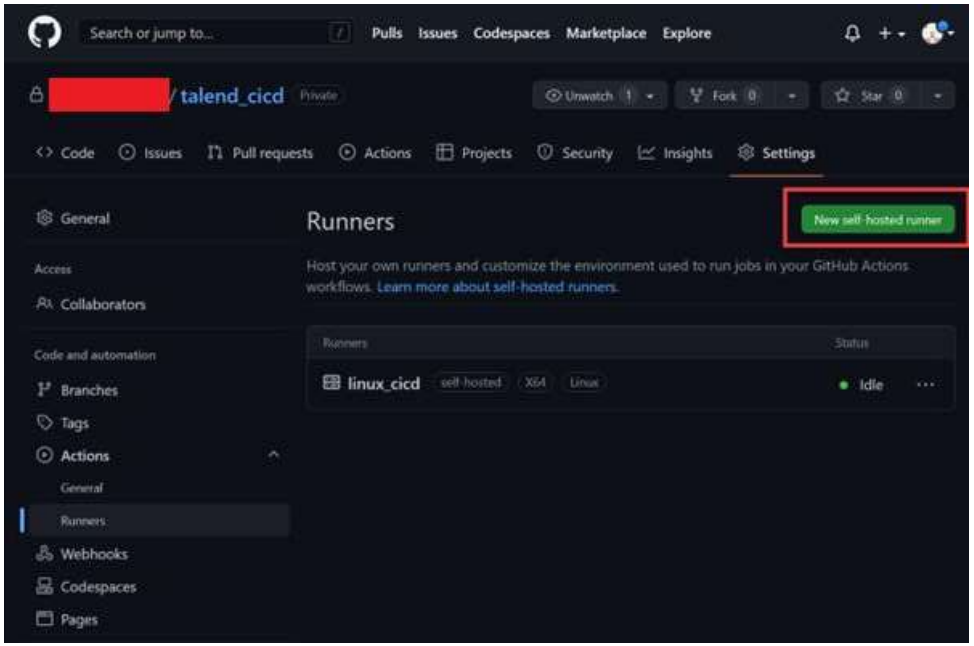
1. Navigate to the GitHub portal, create a repository, name it **talend_cicd**, and make it private.



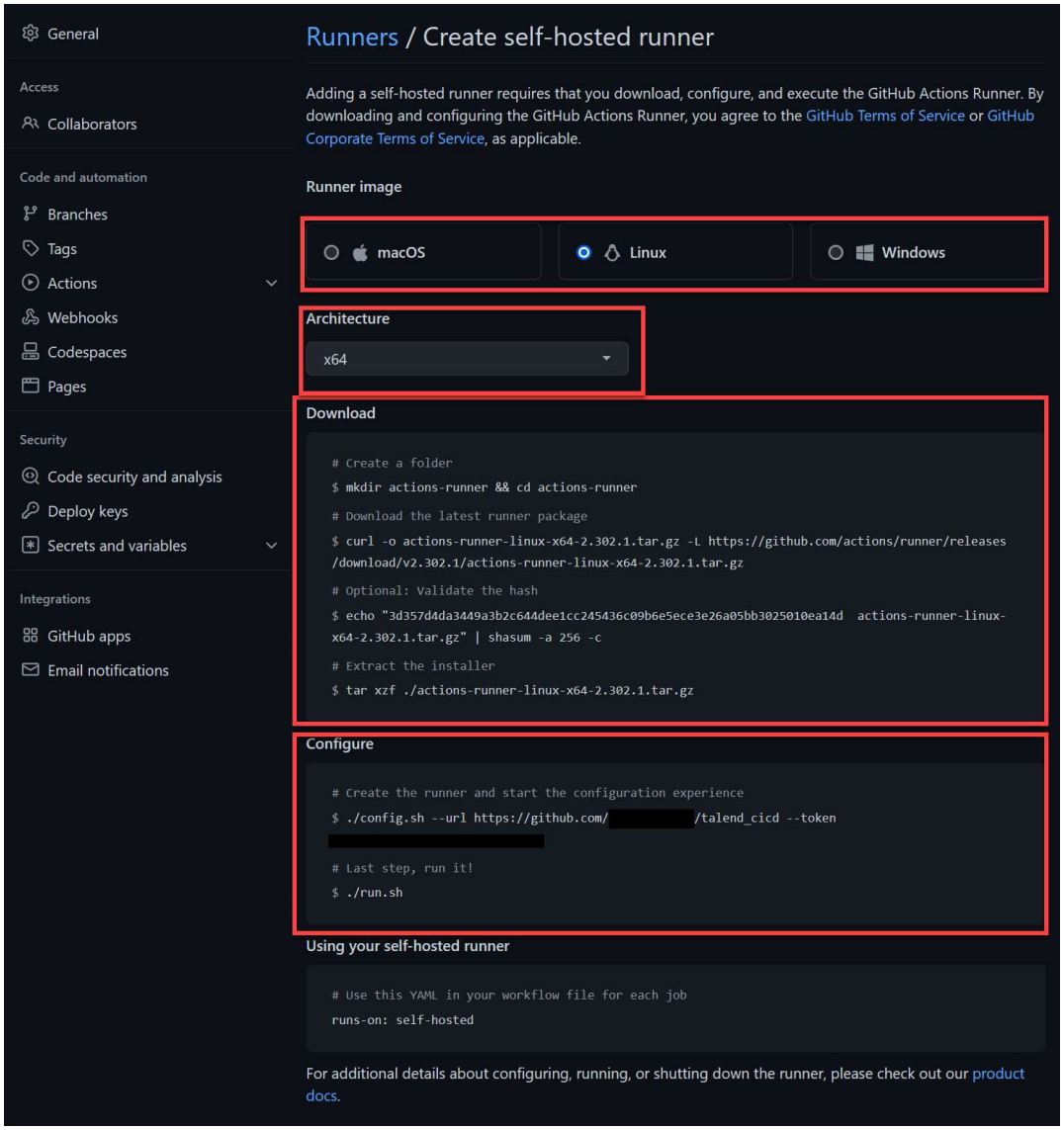
2. In the GitHub repository, select **Settings** > **Actions** > **Runners**.



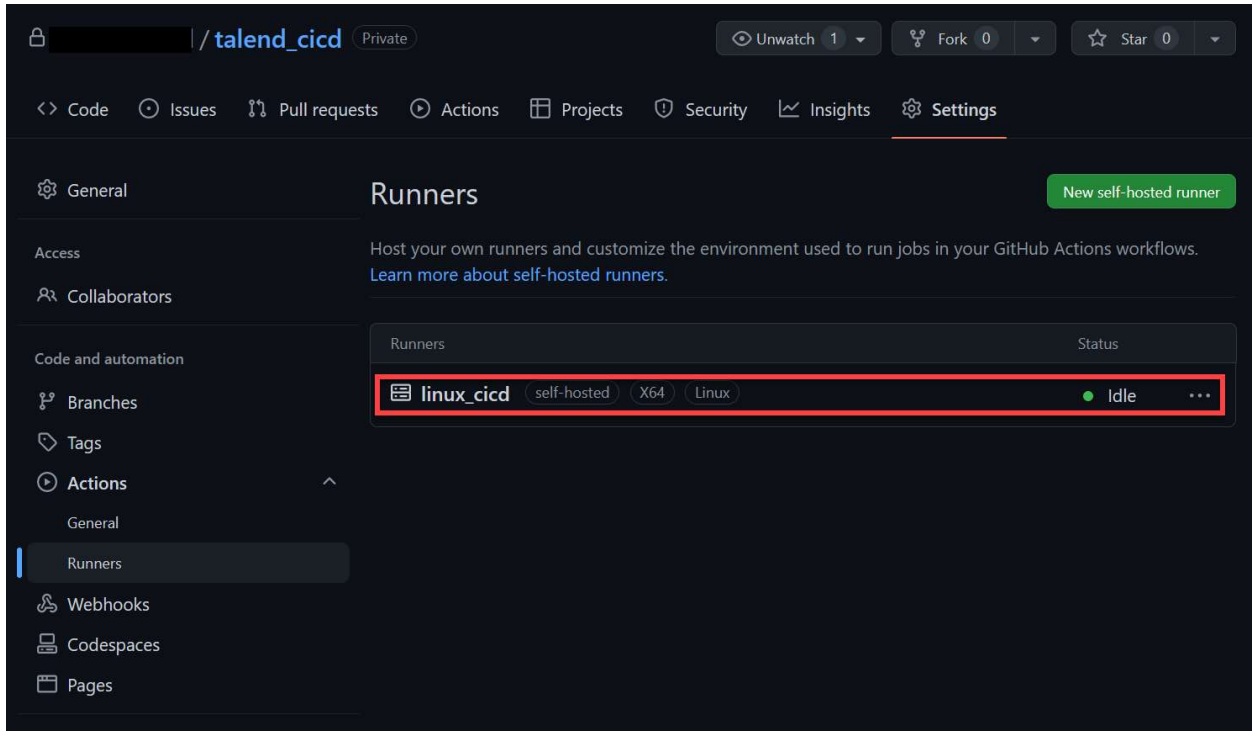
3. Click the **New self-hosted runner** button.



4. Select your **Runner image** and follow the GitHub instructions to install and configure the runner.



5. After installation, ensure the runner is visible in **Settings** > **Actions** > **Runners**. If the runner's status is **offline**, ensure that the time of the server is synchronized.

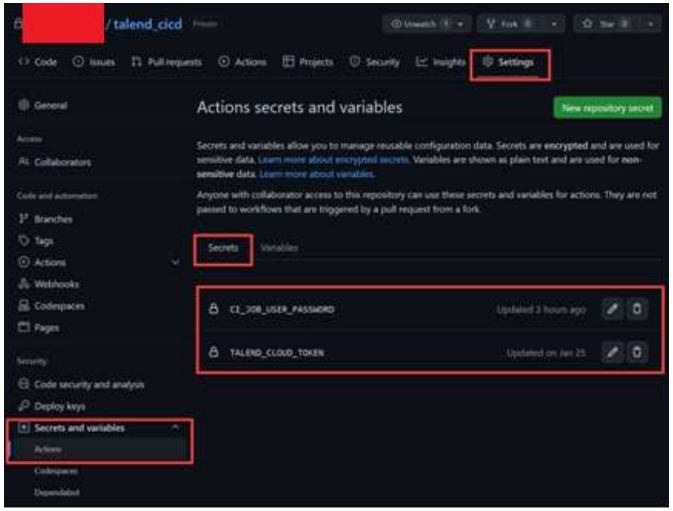


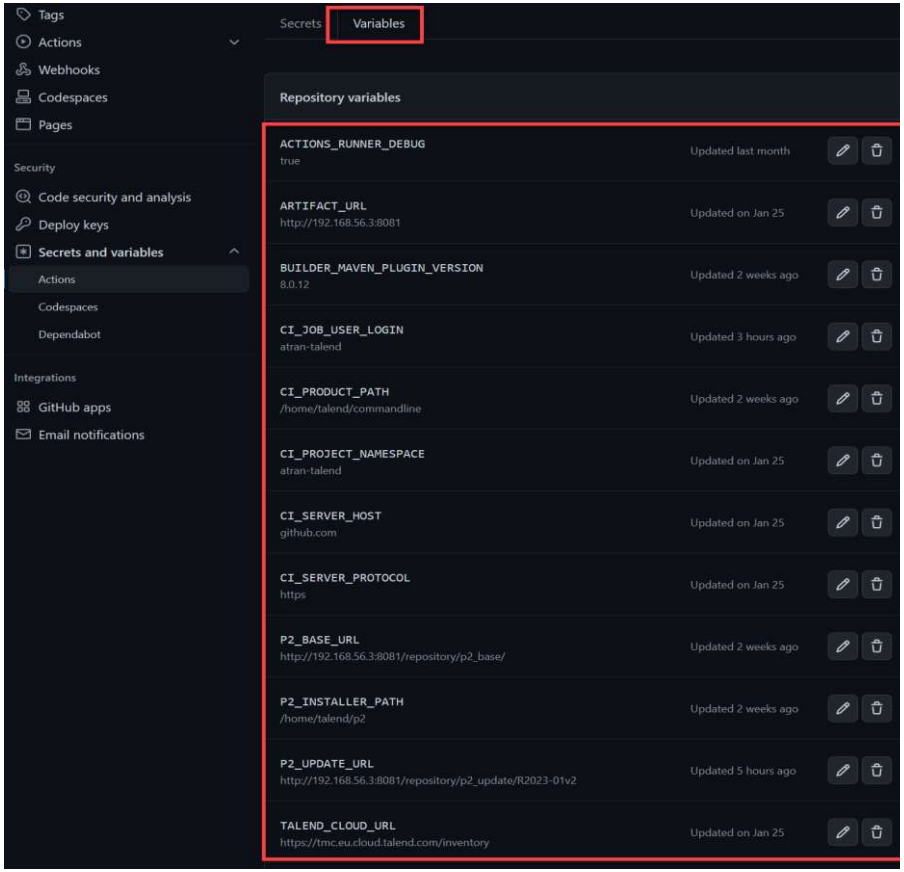
Configuring GitHub

In this section, you create variables and secrets to store values you want to control (URLs, credentials, and more), reuse them in all your project's group pipelines, and upload the required configuration files.

1. In the GitHub repository, select **Settings** > **Secrets and variables** > **Actions**.

    a. Create the global variables in **Secrets** if it contains a password or secret, or in **Variables** if it is not sensitive information.



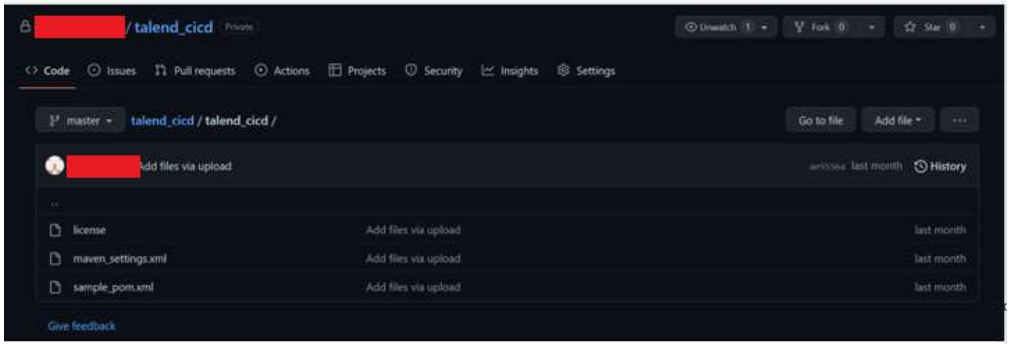


Type	Variables	Description	Example
Secrets	CI_JOB_USER_PASSWORD	Token of GitHub account – this is useful when a reference project is used (because stored in another GitHub repository). You do not need the token to read the current GitHub repository. The runner will automatically generate and use the GitHub Token in the workflow (secrets.GITHUB_TOKEN).	
Secrets	TALEND_CLOUD_TOKEN	To publish artifacts to Talend Cloud: A personal access token is necessary to authenticate your account. Mask this variable.	
Secrets	LARGE_SECRET_PASSPHRASE	The gpg tool uses the passphrase to encrypt or decrypt files.	
Variables	ACTIONS_RUNNER_DEBUG	Set to true to enable debug mode.	true
Variables	ARTIFACT_URL	To publish artifacts to an artifact repository: the URL of the artifact repository (Nexus, Artifactory, and more) where you want to deploy your artifacts.	http://192.168.56.3:8081
Variables	BUILDER_MAVEN_PLUGIN_VERSION	The version of the Talend CI Builder Maven plugin – you can update it with a monthly patch.	8.0.12
Variables	CI_JOB_USER_LOGIN	GitHub account login	

Type	Variables	Description	Example
Variables	CI_PRODUCT_PATH	Path where CommandLine will be installed.	/home/talend/commandline
Variables	CI_PROJECT_NAMESPACE	Owner of the GitHub repository	atran-talend
Variables	CI_SERVER_HOST	GitHub server host	github.com
Variables	CI_SERVER_PROTOCOL	GitHub protocol	HTTPS
Variables	P2_BASE_URL	URL of the P2 server containing the Talend CommandLine and the Maven plugins necessary to generate and deploy your artifacts.	https://update.talend.com/Studio/8/base
Variables	P2_UPDATE_URL	URL of the folder where the cumulative patches can be downloaded.	https://update.talend.com/Studio/8/updates/R2023-01v2
Variables	P2_INSTALLER_PATH	Path where p2 will be installed.	/home/talend/p2
Variables	TALEND_CLOUD_URL	To publish artifacts to Talend Cloud: URL of the Talend Cloud service where you want to deploy your artifacts.	https://tmc.eu.cloud.talend.com/inventory

2. Upload the configuration files in the Git repository. In GitHub, you can add your custom Maven files and your Talend product license or local patches in a project and then use these files in the CI/CD pipelines.

    a. Upload all the files you have adapted to your environment to the Main Branch of the **talend_cicd** repository in GitHub. In this case, the files are uploaded in a folder named **talend_cicd** in the project **TALEND_CICD**. Your TALEND_CICD project should look like this:



Notes:

- The Git clone is downloaded in your default downloads folder. You can change it using the working directory command (see the clone git repository stage in this guide's Creating a GitHub build pipeline section).

- Issues with dynamic variables are initialized dynamically based on other variables (see the set MAVEN_OPS variable for Talend stage in this guide's Creating a GitHub build pipeline section).

- As a best practice, Talend recommends storing the **maven_setting.xml** or any files containing sensitive information on the server side, not in Git or any source code repository, for security purposes. However, for technical implementation reference, the files are stored in Git.

Creating a GitHub build pipeline

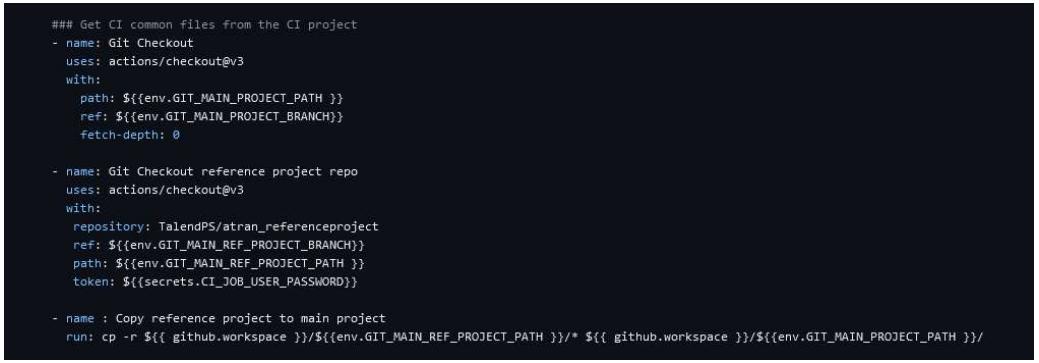
Retrieve and adapt YAML parameters to your environment for the **github-pipeline.yml** file. The pipeline starts with the inner variables definition, then has five stages:

- **set MAVEN_OPS variable for Talend**: For security reasons, the initialization of a variable based on another variable is not straightforward (the variable MAVEN_OPS depends on the variables GIT_FOLDER_NAME, P2_BASE_URL, P2_UPDATE_URL, and so on). However, as a workaround, you can use the syntax using the **echo** command followed by **>>$GITHUB_ENV**.

```
### TO DECOMMENT WHEN USING TALEND CLOUD
- name: set MAVEN_OPS variable for Talend
  working-directory: ${{runner.temp}}
  run: echo "MAVEN_OPTS=$(echo -Dinstaller.clean=true  \
          -Dlicense.path=${{runner.temp}}/${{env.GIT_FOLDER_NAME}}/${{env.CI_REPO_NAME}}/license  \
          -Dtalend.studio.p2.base=${{vars.P2_BASE_URL}}  \
          -Dtalend.studio.p2.update=${{vars.P2_UPDATE_URL}}  \
          -Dgeneration.type=local \
          -Dservice.url=${{vars.TALEND_CLOUD_URL}} \
          -Dcloud.token=${{secrets.TALEND_CLOUD_TOKEN}} \
          -Dcloud.publisher.screenshot=true \
          -Dcloud.publisher.updateFlow=true \
          -Dcloud.publisher.environment=${{env.TALEND_CLOUD_ENV}} \
          -Dcloud.publisher.workspace=${{env.TALEND_CLOUD_WORKSPACE}} \
          -Dp2Installer.path=${{vars.P2_INSTALLER_PATH}}  \
          -Dproduct.path=${{vars.CI_PRODUCT_PATH}}  \
          -Dorg.slf4j.simpleLogger.showDateTime=true   \
          -Dorg.slf4j.simpleLogger.dateTimeFormat=HH:mm:ss,SSS)" >> $GITHUB_ENV
```

- **display variables** (optional): Displays the variables used by the pipeline.

- **clone git repository**: Gets the shared files and the Talend project from the TALEND_CICD project for the current run.



```
### Get CI common files from the CI project
- name: Git Checkout
  uses: actions/checkout@v3
  with:
    path: ${{env.GIT_MAIN_PROJECT_PATH }}
    ref: ${{env.GIT_MAIN_PROJECT_BRANCH}}
    fetch-depth: 0

- name: Git Checkout reference project repo
  uses: actions/checkout@v3
  with:
   repository: TalendPS/atran_referenceproject
   ref: ${{env.GIT_MAIN_REF_PROJECT_BRANCH}}
   path: ${{env.GIT_MAIN_REF_PROJECT_PATH }}
   token: ${{secrets.CI_JOB_USER_PASSWORD}}

- name : Copy reference project to main project
  run: cp -r ${{ github.workspace }}/${{env.GIT_MAIN_REF_PROJECT_PATH }}/* ${{ github.workspace }}/${{env.GIT_MAIN_PROJECT_PATH }}/
```

Note: The first Git checkout fetches the code source from the **main** repository. No token is sent to this Git command; the workflow will use the secret **GITHUB_TOKEN** by default. The token's permissions are limited to the repository that contains the workflow.

The second Git checkout is optional. It fetches the code source from the reference project. Because the previous token can't give permission to read the reference repository, an additional token is used as input in the Git checkout command.

- **install P2 and generate poms**: For performance reasons, the POM files for your projects are not generated by default. It is mandatory to generate your POM files first to execute a CI build successfully. Talend CI Builder lets you generate POM files for your projects using the **mvn org.talend.ci:builder-maven-plugin:8.0.*X*: generateAllPoms** command. This step will also install the Talend CI CommandLine from the P2 repository at the first launch.

- **build and deploy to artifact repository**: This stage generates and builds the Jobs, runs the unit tests, and publishes the artifacts.

Configure the **github-pipeline.yml** file to generate the project items according to the Maven phase you want to achieve and the repository you want to deploy your artifacts. Some parameters and credentials come from the group variables you created in the previous section. For more information about the Talend Maven parameters you can use when configuring your pipeline, see Talend custom Maven build options.

Warning: Syntax requirements:

The value of the *PROJECT_NAME* variable must be in uppercase; otherwise, it might cause build failure. If you used special characters in the project name or have doubts about its final syntax, you can check it in the **parent pom.xml** file of your project (*talend.project.name* variable in **<*project*>/poms/pom.xml**).

The value of the *JOB_LIST* variable must be in lowercase even if the original name of your Job, Route, or Service contained uppercase, as all artifacts are in lowercase in the **.pom** files required to launch the build.

Do not manually add the **"-SNAPSHOT"** suffix to the version value. Snapshots and releases are decoupled from the version itself and must only be handled using the Studio preferences. For more information, see Changing the deployment version of each artifact at once available in Talend Help Center.

Note: If you want to deploy Jobs that use custom components, you need to use the **-Dcomponents.*** options described in Talend custom Maven build options available in Talend Help Center.

To define the Maven phase you want to achieve: make sure the phase stated in the goals line is the one you want to perform. The default goal defined in the default script provided by Talend is **deploy**. For more information on the Maven phases, see the Maven documentation, Introduction to the Build Lifecycle.

Tip: All build profiles (-Pdocker, -Pnexus, and more) are listed in the **parent pom.xml** file of your project. Some default parameters, like the docker image name or the Open JDK name, can also be overwritten in this file.

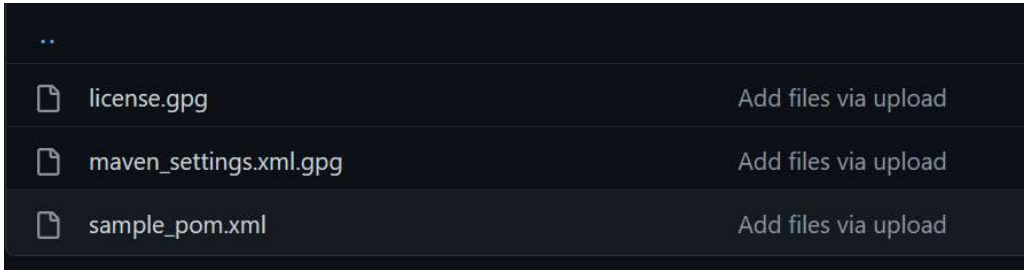
In the **github-pipeline.yml** file, the parameter **-Dinstaller.clean** is set to **true** in the **MAVEN_OPTS**. If you use permanent storage for your runner, you can set this parameter to **false** after the first successful run. This will avoid reinstalling and updating the P2 at each run.

Encrypting or decrypting configuration files containing secrets

In the GitHub project, the following files contain sensitive information. You can use the gpg tool to encrypt the information:

- **Talend license** file
- **maven_settings.xml** file contains artifact repository credentials.

When you finish your GitHub repository should contain the following files:

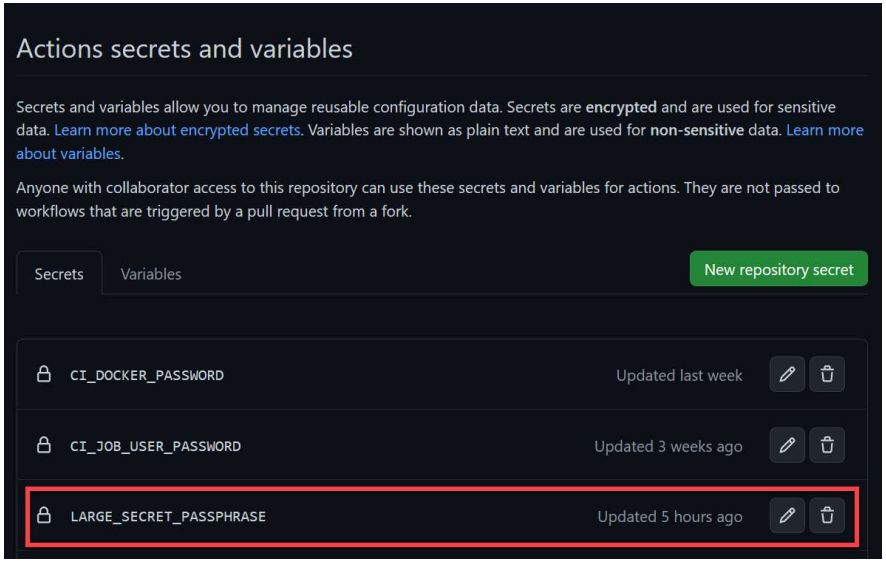


Encrypting the files - examples

The following command encrypts the license file using the gpg tool and AES256 algorithm. You will be prompted to enter a passphrase.

```
[talend@localhost talend_cicd]$ gpg --symmetric --cipher-algo AES256 license
gpg: répertoire « /home/talend/.gnupg » créé
gpg: le trousseau local « /home/talend/.gnupg/pubring.kbx » a été créé
[talend@localhost talend_cicd]$
```

You should store the passphrase under **Secrets**.

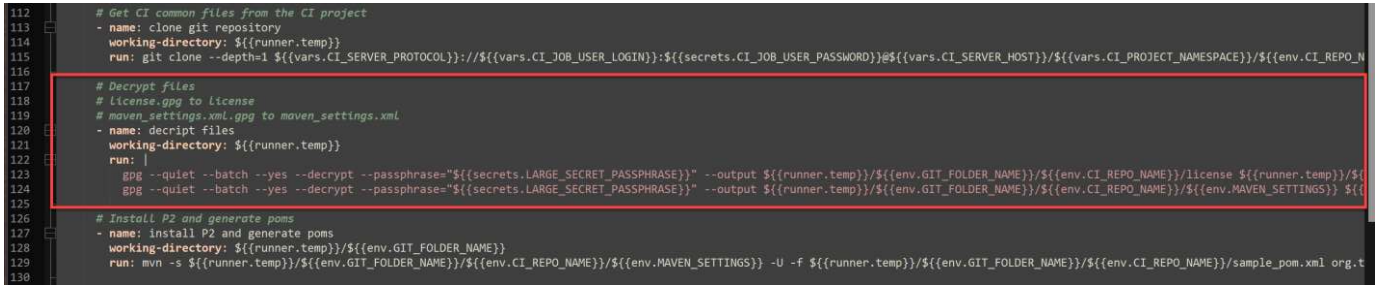



```
[..]
gpg --symmetric --cipher-algo AES256 license
gpg --symmetric --cipher-algo AES256 maven-settings.xml
[..]
```

Decrypting the files - examples

If the project contains encrypted files, use the following to decrypt the files:

```
[..]
# Decrypt files
      # license.gpg to license
      # maven_settings.xml.gpg to maven_settings.xml
      - name: decript files
        working-directory: ${{runner.temp}}
        run: |
          gpg --quiet --batch --yes --decrypt --passphrase="${{secrets.LARGE_SECRET_PASSPHRASE}}" \
            --output ${{runner.temp}}/${{env.GIT_FOLDER_NAME}}/${{env.CI_REPO_NAME}}/license \
  ${{runner.temp}}/${{env.GIT_FOLDER_NAME}}/${{env.CI_REPO_NAME}}/license.gpg
          gpg --quiet --batch --yes --decrypt --passphrase="${{secrets.LARGE_SECRET_PASSPHRASE}}" \
            --output  \ {{runner.temp}}/${{env.GIT_FOLDER_NAME}}/${{env.CI_REPO_NAME}}/${{env.MAVEN_SETTINGS}} \
  ${{runner.temp}}/${{env.GIT_FOLDER_NAME}}/${{env.CI_REPO_NAME}}/${{env.MAVEN_SETTINGS}}.gpg
[..]
```



Publishing artifacts to Talend Cloud Management Console

1. Open the **github-pipeline.yml** file and look for the **XXXX** references defined at the top of the script.

    a. Set your values for the **PROJECT_NAME**, **CI_REPO_NAME**, and **JOB_LIST** environment variables.

    b. Replace the **TMC_ENV** and **TMC_WORKSPACE** Talend Cloud environment and workspace parameters with values corresponding to your environment.

2. Save the file.

The following is an example of the parameters for a Talend project named **TALEND_CICD**, a Job named **mydemo** with a **0.1** version published in the Talend Cloud DEV environment in the Talend Cloud workspace **DEMO_WORKSPACE**. The shared CI files are stored in the **TALEND_CICD** GitHub project.

```
Env:
  GITHUB_RUNNER: "self-hosted"
  # Talend project name
  PROJECT_NAME: "TALEND_CICD"
  # Shared GitLab project for CI shared files (lowercase)
  CI_REPO_NAME: "talend_cicd"
  # Root folder of git clone as the runner.temp folder isn't empty
  GIT_FOLDER_NAME: "git_root"
  ############ JOB FILTER ####################
  #Variable corresponding to the specific artifact you want to build. Define it as a variable in the GUI so it can be
overwritten at runtime. If you don't use it, all existing artifacts in your project will be executed and published. Coma
separator if more than one.
  JOB_LIST: "jobs/process/mydemo_0.1"
  ############ TMC ############################
  TALEND_CLOUD_ENV: "atran_dev"
  TALEND_CLOUD_WORKSPACE: "atran_work"
  ############# MAVEN OPTIONS #################
[…]
  MAVEN_PLUGIN: "-Pcloud-publisher"
[…]
  - name: set MAVEN_OPS variable for Talend
        working-directory: ${{runner.temp}}
        run: echo "MAVEN_OPTS=$(echo -Dinstaller.clean=true  \
                -Dlicense.path=${{runner.temp}}/${{env.GIT_FOLDER_NAME}}/${{env.CI_REPO_NAME}}/license  \
                -Dtalend.studio.p2.base=${{vars.P2_BASE_URL}}  \
                -Dtalend.studio.p2.update=${{vars.P2_UPDATE_URL}}  \
                -Dgeneration.type=local \
                -Dservice.url=${{vars.TALEND_CLOUD_URL}} \
                -Dcloud.token=${{secrets.TALEND_CLOUD_TOKEN}} \
                -Dcloud.publisher.screenshot=true \
                -Dcloud.publisher.updateFlow=true \
                -Dcloud.publisher.environment=${{env.TALEND_CLOUD_ENV}} \
                -Dcloud.publisher.workspace=${{env.TALEND_CLOUD_WORKSPACE}} \
                -Dp2Installer.path=${{vars.P2_INSTALLER_PATH}}  \
                -Dproduct.path=${{vars.CI_PRODUCT_PATH}}  \
                -Dorg.slf4j.simpleLogger.showDateTime=true   \
                -Dorg.slf4j.simpleLogger.dateTimeFormat=HH:mm:ss,SSS)" >> $GITHUB_ENV
```

Note: The artifact published to Talend Cloud cannot exceed 400 MB.

Publishing artifacts to a remote Maven repository

1. Open the **github-pipeline.yml** file and edit the **XXXX** references defined at the top of the script.
   a. Set your values for the **PROJECT_NAME**, **CI_REPO_NAME**, and **JOB_LIST** environment variables.
   b. Comment the **TMC** and **MAVEN_OPTS** parameters for Talend Cloud Management Console.
   c. Uncomment the **MAVEN_OPTS** to publish to a remote Maven repository.
   Warning: Be sure to have defined your releases and snapshots in the servers and repositories sections of your **maven_setting.xml** file.
2. Save the file.
3. Depending on where you want to publish artifacts, edit the artifact repository URL stated in the **artifact_url** group variable you created previously.

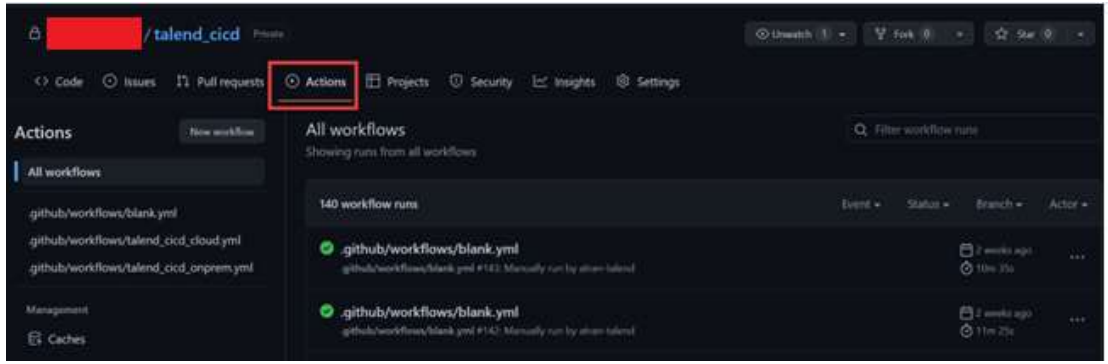
The following is an example of parameters for a project named **TALEND_CICD**, a Job named **mydemo** with a **0.1** version published in the release's repository of the Nexus artifact repository. The shared CI files are stored in the **TALEND_CICD** GitHub project.

```
 Env:
  GITHUB_RUNNER: "self-hosted"
  # Talend project name
  PROJECT_NAME: "TALEND_CICD"
  # Shared GitHub project for CI shared files (lowercase)
  CI_REPO_NAME: "talend_cicd"
  # Root folder of git clone as the runner.temp folder isn't empty
  GIT_FOLDER_NAME: "git_root"
  ############ JOB FILTER ####################
  #Variable corresponding to the specific artifact you want to build. Define it as a variable in the GUI so it can be
overwritten at runtime. If you don't use it, all existing artifacts in your project will be executed and published. Coma
separator if more than one.
  JOB_LIST: "jobs/process/mydemo_0.1"
  ############# MAVEN OPTIONS ################
[…]
  MAVEN_PLUGIN: "-Pcloud-publisher"
[…]
  - name: set MAVEN_OPS variable for Talend on-prem
        working-directory: ${{runner.temp}}
        run: echo "MAVEN_OPTS=$(echo -Dinstaller.clean=true   \
                -Dlicense.path=${{runner.temp}}/${{env.GIT_FOLDER_NAME}}/${{env.CI_REPO_NAME}}/license  \
                -Dtalend.studio.p2.base=${{vars.P2_BASE_URL}}  \
                -Dtalend.studio.p2.update=${{vars.P2_UPDATE_URL}}  \
                -Dgeneration.type=local  \
                -Dp2Installer.path=${{vars.P2_INSTALLER_PATH}}  \
                -Dproduct.path=${{vars.CI_PRODUCT_PATH}}  \
                -Dorg.slf4j.simpleLogger.showDateTime=true   \
                -Dorg.slf4j.simpleLogger.dateTimeFormat=HH:mm:ss,SSS)" >> $GITHUB_ENV
[…]
```

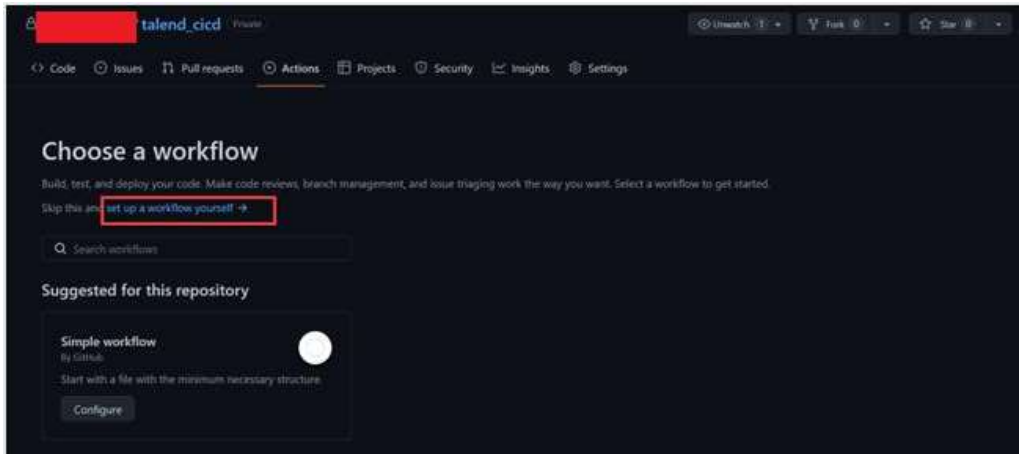
Adding the GitHub pipeline to your project

When your **github-pipeline.yml** pipeline is ready, upload it to the **TALEND_CICD** project by performing the steps:

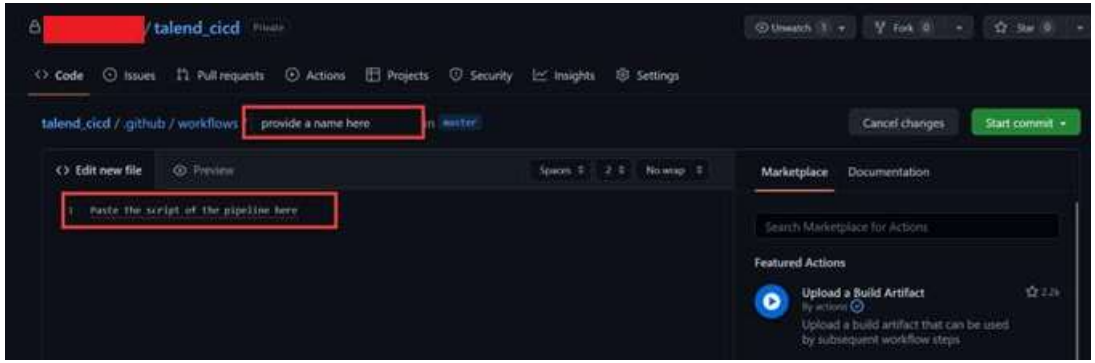
1. Click **GitHub** > **Projects** > **Actions** > **New Workflow**.



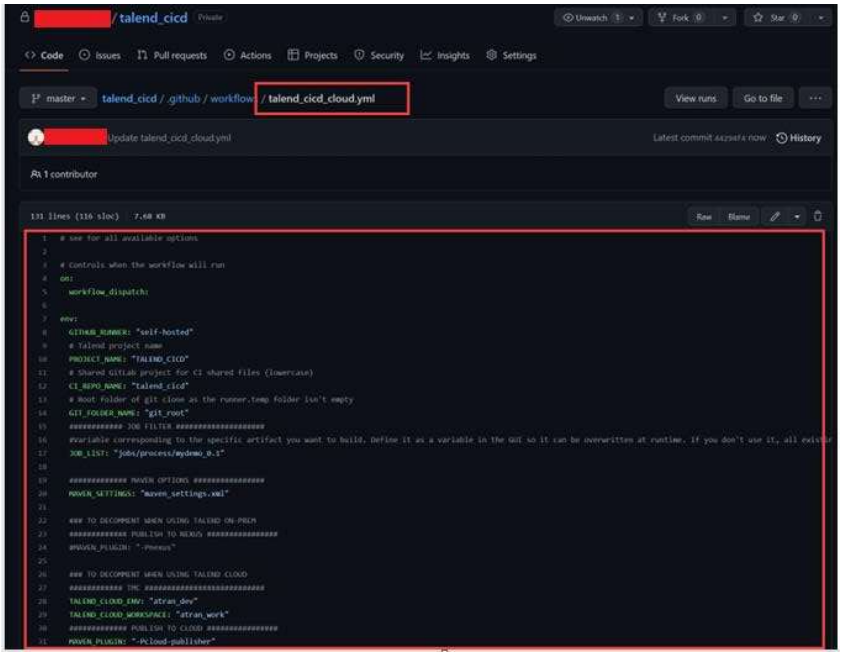
2. Click **Set up a Workflow Yourself**.



3. In the empty file, paste the script and give it a name.



4. Click the **Start commit** button to create the workflow.



Running the GitHub pipeline

In GitHub, the pipelines are triggered by a git push (whatever the branch is) or run manually. You can add restrictions to a branch (or more) in the **yml** script at the build stage using trigger on push event.

The following is an example of rules to trigger the workflow after a push on the master branch only:

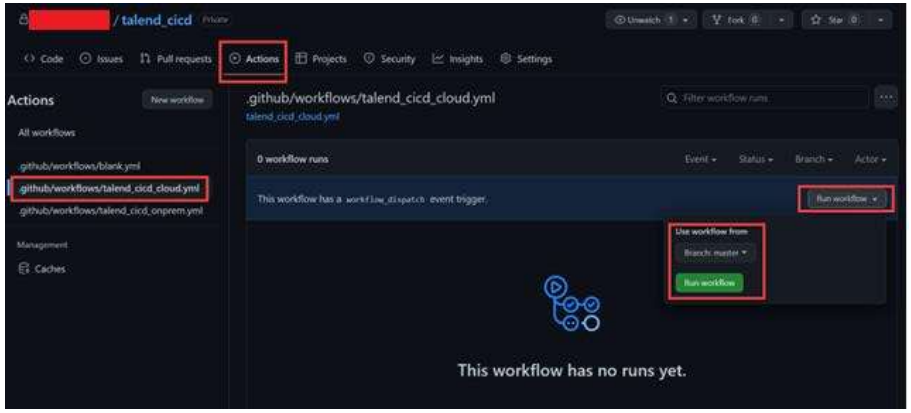
```
on:
  push:
    branches:
      - main
```

Warning: Studio libraries synchronization

Before running your pipeline, make sure you have synchronized the Studio libraries to your artifact repository defined in your **maven_settings.xml** file (Nexus or Artifactory). For more information, see Artifact repository for libraries preferences available in the Talend Help Center.

Running the build

1. Navigate to your **Talend Project GitHub** repository and select **Actions**.
2. Select the workflow you want to execute from the pane on the left.
3. Click **Run Workflow** and select the branch.
4. Click **Run Workflow** again to confirm the selected branch.



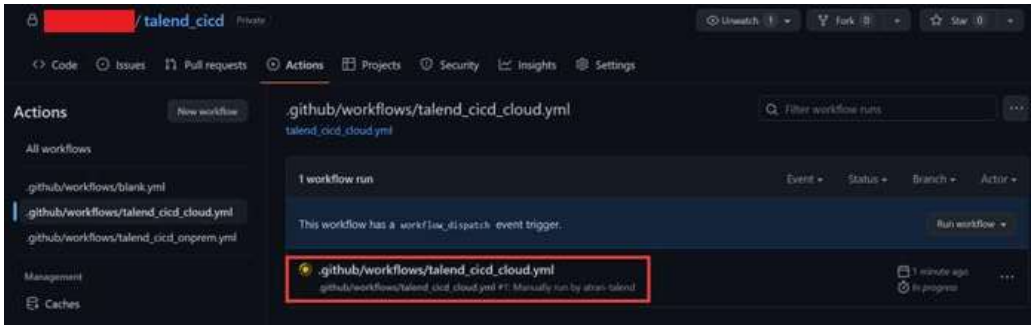
You can follow the logs by clicking the pipeline's Job (or stage).

Reviewing the results

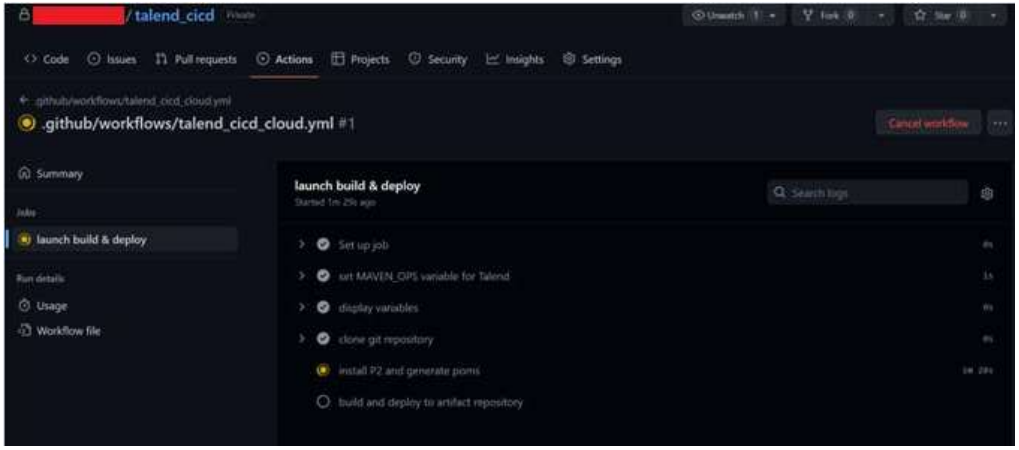
The pipeline is launched, and your project is processed according to the Maven phases defined in your script. As a best practice, use the deploy phase to generate the source code, compile it, test it, package it, and then deploy the packaged artifacts.

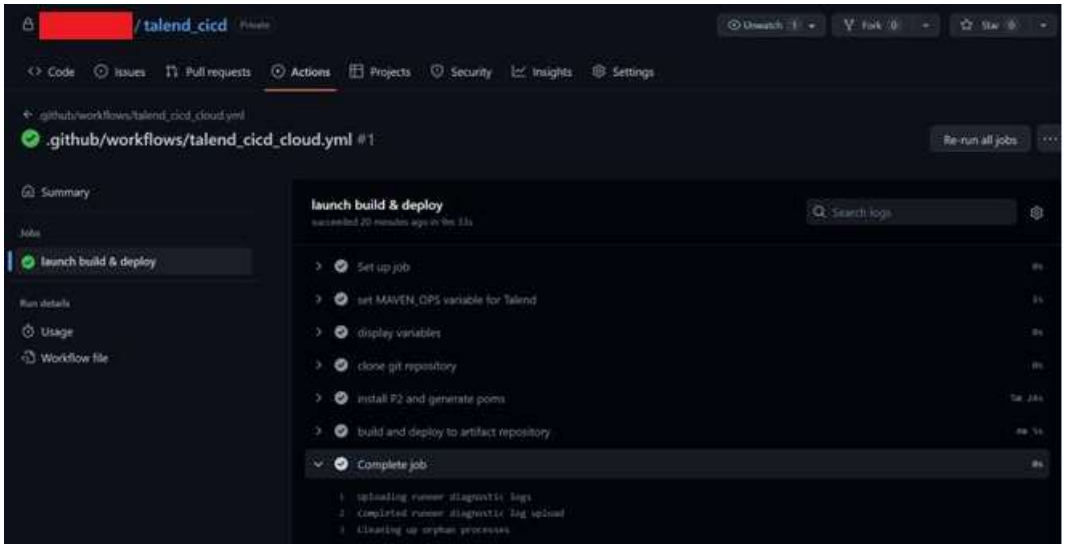
After your pipeline executes successfully, you can see the status and details of each step.

1. Go to your **Talend Project GitHub** repository and select **Actions**.
2. Select the workflow.

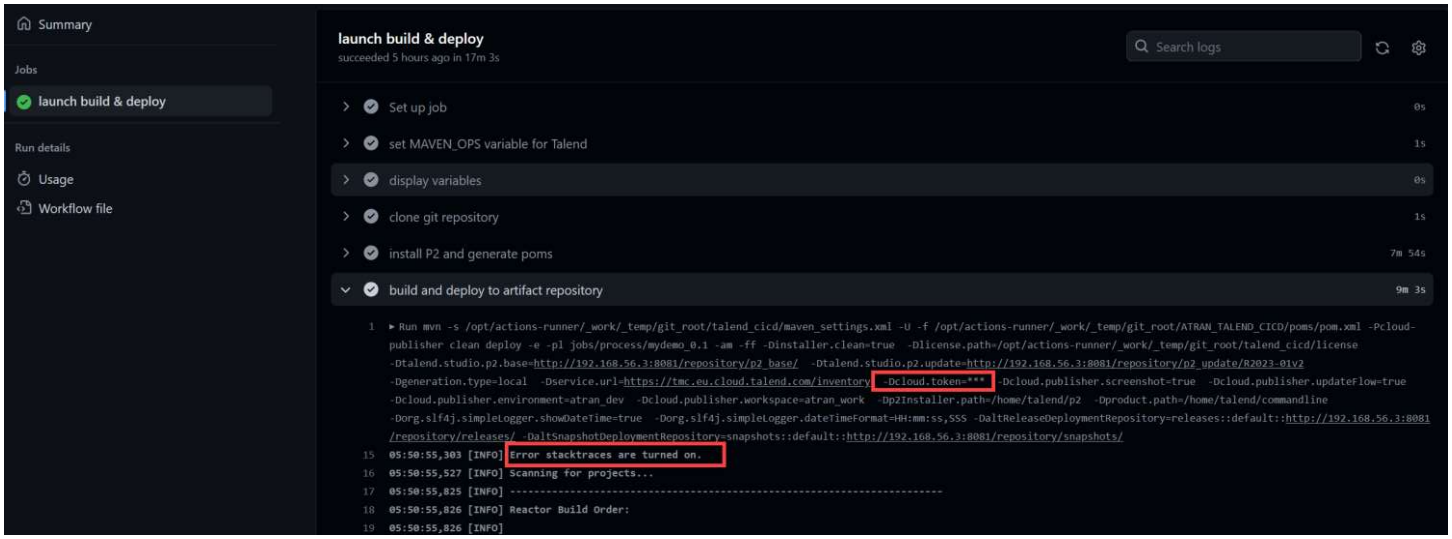


3. Click a Job (or stage) to see the details of the log.

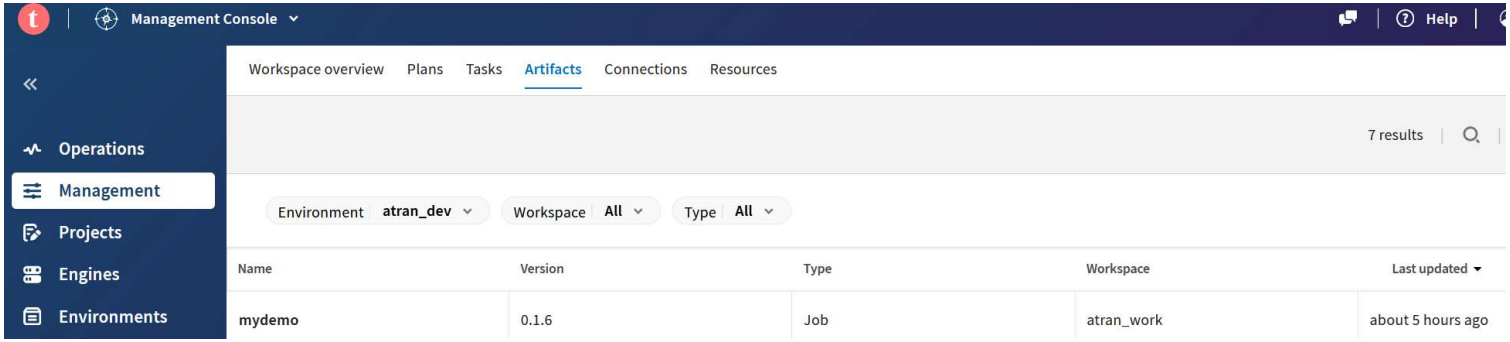




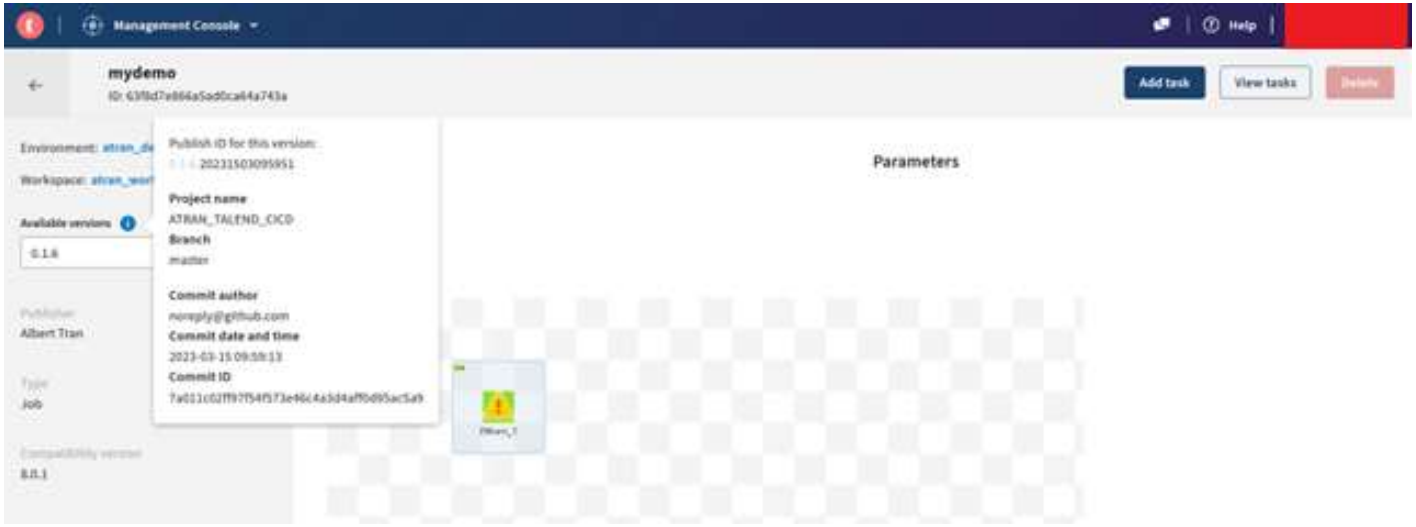
4. Notice that the secrets are obfuscated even though the debug mode is enabled. For more information, see the GitHub documentation, Enabling debug logging.



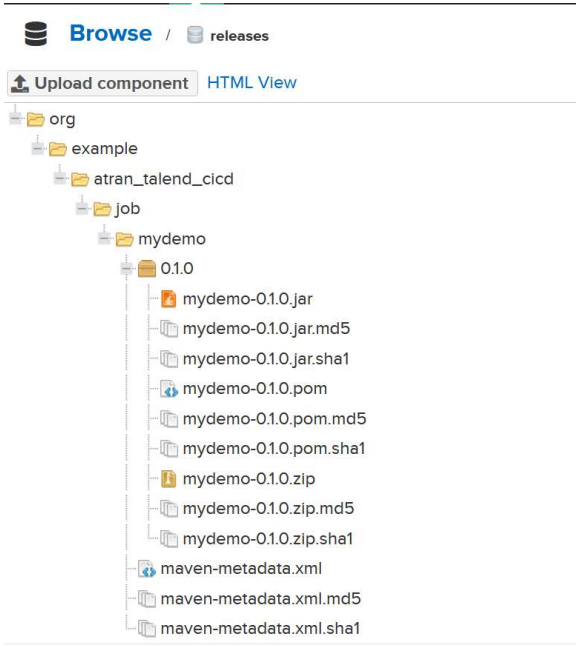
If you have chosen to publish in Talend Cloud Management Console, you can see your published task and artifact. The following is an example of the **mydemo** Job with version **0.1.0** deployed in the environment and workspace defined in the pipeline parameters:



The option to display Git information (author, commit ID, commit date) in Talend Cloud Management Console when publishing artifacts using CI builds is available from version 8.0.1 onward (available from R2022-01). This information is displayed on the artifact details page of Talend Cloud Management Console.



If you have chosen to publish in an artifact repository: Example of Job artifact with version **0.1.0** deployed in the Nexus **releases** repository with a Group ID:



Conclusion

This implementation accelerator showed you how to implement the CI/CD with a GitHub Actions runner on any use case, for hands-on purposes, or to reproduce any scenario in your environment.

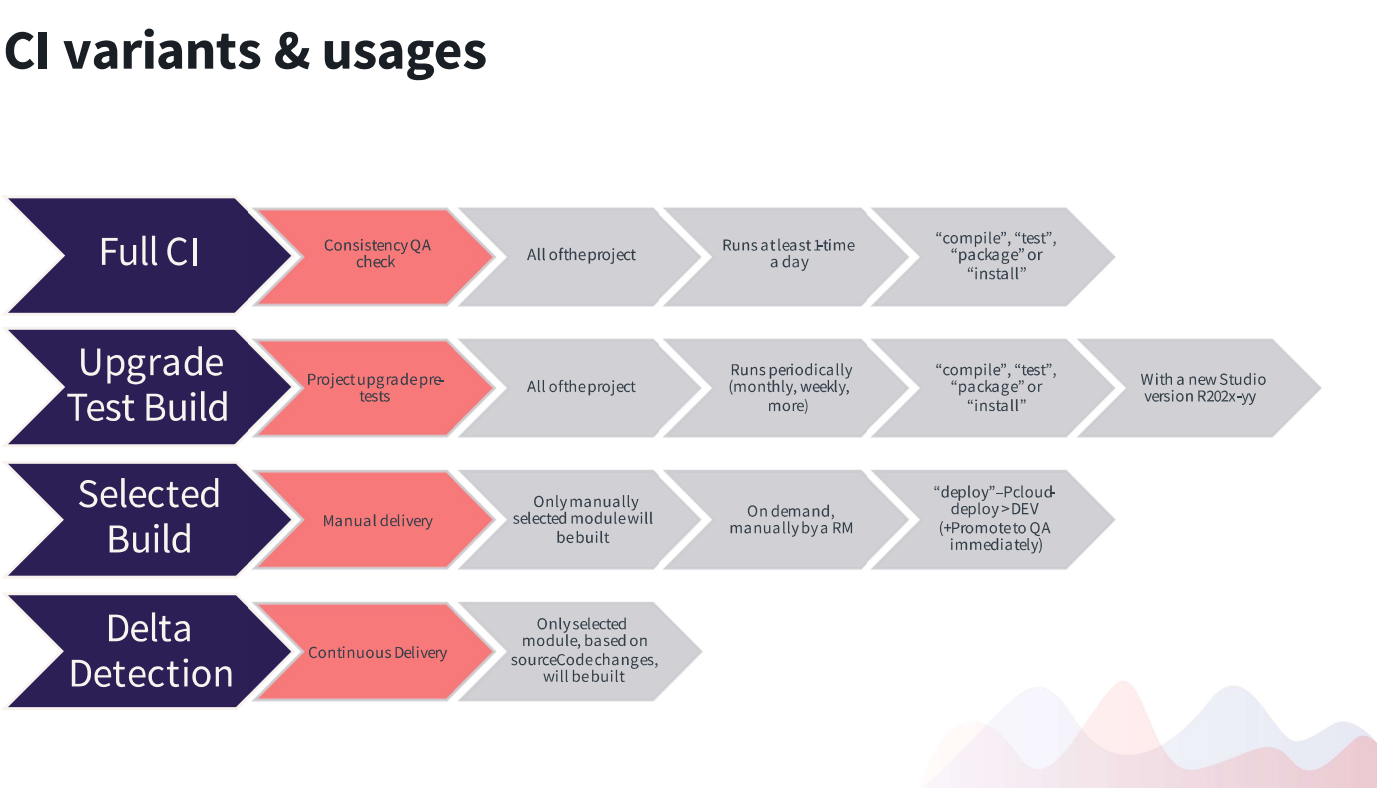
Appendix

Going further with CI

Now that you have made your first implementation of Talend CI with the technology of your choice, consider how you will use it.

Do not forget that the initial goal of CI is to increase the overall quality of your entire project by detecting any side effects that one modification may unexpectedly have on other parts of your project.

As such, you may consider the following four cases, from the most basic or default reason for implementing CI to the most advanced.



References

Websites:

- Understanding GitHub Actions
- Differences between GitHub-hosted and self-hosted runners
- Enabling debug logging
- Automatic token authentication
- Permissions for the GITHUB_TOKEN
- Welcome to Apache Maven
- Downloading Apache Maven 3.9.2
- Download Azul JDKs

Talend Help Center:

- Building and Deploying
- What is Talend Cloud Management Console?

Talend support:

- Talend Professional Services
- Talend Global Customer Support