

Github Actions

Sintaxis básica.

Índice de Contenidos

1. [Entorno](#)
2. [Introducción](#)
3. [Workflow. Partes y sintaxis.](#)
4. [Configuración y ejecución de un workflow](#)
5. [Construir la imagen con Docker](#)
6. [Secretos](#)
7. [Conclusiones](#)

1. Entorno

Este tutorial ha sido escrito utilizando el siguiente entorno:

- Lenovo Thinkpad L15
- Intel Core i7 64GB
- Ubuntu 20.04

2. Introducción a Github Actions

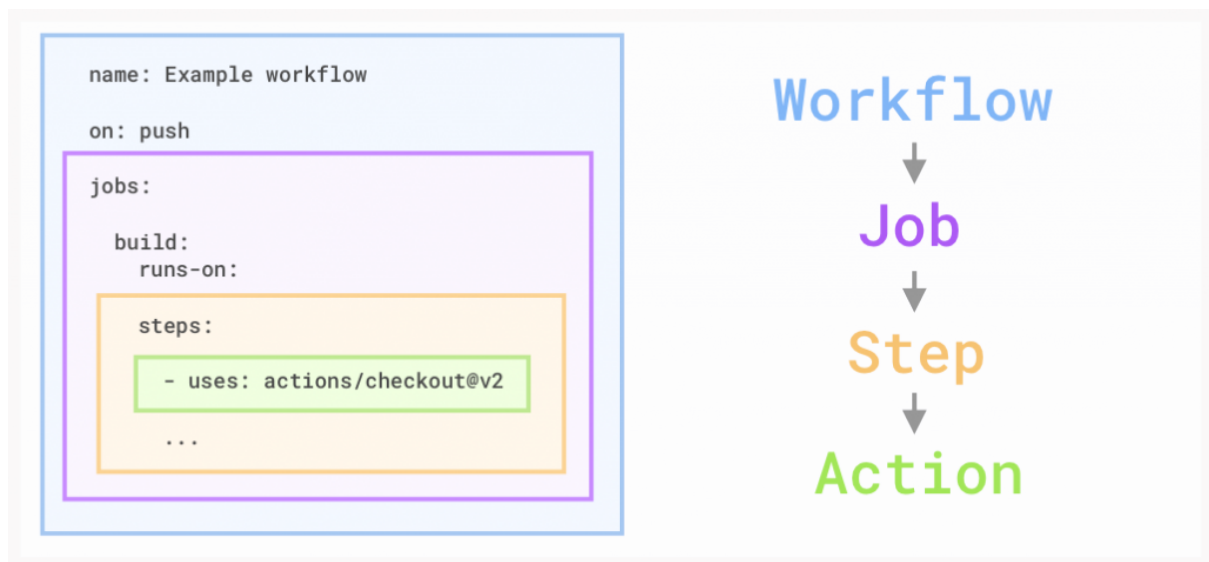
La intención en este artículo es profundizar un poco en esta tecnología de CI/CD, explicando en qué consiste, para qué surgió y cómo se está utilizando.

Como bien sabemos, la **Integración Continua** es una práctica que nos requiere añadir frecuentemente nuevo código a un repositorio compartido para detectar errores a la mayor brevedad posible. Para ello Github propuso en 2018 no solo alojar nuestro código en sus repositorios como hemos hecho siempre, sino que además la posibilidad de automatizar los distintos pasos de compilación y test de nuestros proyectos (al igual que ya tenía implementado Gitlab).

Para ello Github Actions ha creado el concepto de **workflow** el cual es el encargado principal de todo nuestro proceso o Pipeline. Se puede configurar de manera que Github reaccione a ciertos eventos (por ejemplo cuando se hace un nuevo push a una rama), automáticamente de forma periódica o por eventos externos. Especificando en dicho workflow que se analicen los componentes del proyecto, una vez terminados se mostrarán los resultados de los mismos y se podrá comprobar si el cambio en dicha rama ha producido algún error o ha ido todo bien. Este mismo es ejecutado en un **runner** o instancia en un servidor y Github te da la posibilidad de utilizar un runner hospedado por Github o añadir un host propio.

3. Workflow. Partes y sintaxis.

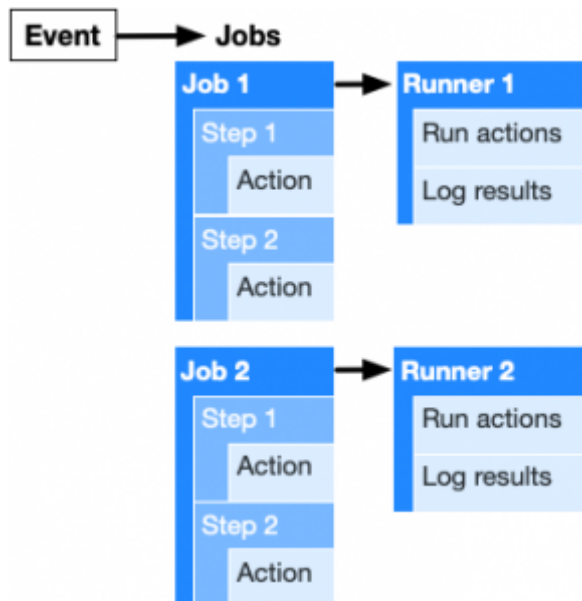
Para entender bien en qué consiste un **workflow**, el siguiente esquema muestra las partes que lo componen.



- **Workflow:** como ya hemos comentado anteriormente, es un procedimiento automatizado el cual se añade a un repositorio. Con él se puede hacer el build, test, package, release o deploy de un proyecto dentro de Github.
- **Job:** es un conjunto de **steps** que se ejecutan en runner de nuestro proceso.
- **Step:** es una tarea individual que puede ejecutar comandos dentro de un **job**. Un job está formado por uno o más steps y éstos están ejecutados sobre el mismo runner a la hora de ejecutarse el workflow.
- **Action:** Son los comandos de ejecución del proceso, ejecutados en un **step** para crear un **job**. Son el bloque de construcción más

pequeño que hay. Puedes crear tus propios actions o utilizar algunos de ellos que ya están creados por la comunidad de Github.

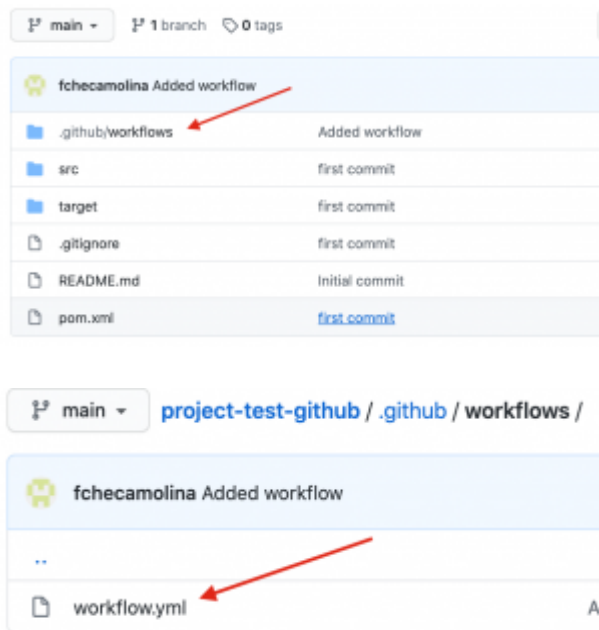
Obligatoriamente para utilizar un action en un workflow, éste debe ir incluido en un step.



Para ver un ejemplo más claro de esto se puede tomar un ejemplo de un proyecto Java con Maven muy simple de este enlace:

<https://github.com/Daniel-MundosE/project-test-github>

Lo primero de todo es crear la propia carpeta en la cual añadiremos el fichero de configuración del workflow. Esta carpeta tiene por norma que llamarse `.github/workflows/`. Aquí vamos a crear el fichero `workflow.yml` (Github Actions utiliza la sintaxis de [YAML](#) para especificar la configuración de todo el workflow).



El contenido inicial de nuestro **workflow.yml** (se le puede poner cualquier otro nombre) es el siguiente:

YAML

```
1  name: Build and test of Java Project

2  on: [push]

3  jobs:

4  build:

5      runs-on: ubuntu-latest

6      steps:

7          - uses: actions/checkout@v2

8          - name: Set up JDK 1.8
```

```
9      uses: actions/setup-java@v1

10      with:

11        java-version: 1.8

12      - name: Build with Maven

13        run: mvn -B package --file pom.xml

14

15

16
```

Entendiendo el fichero por partes:

- **“name: Build and test of Java Project”**: El nombre opcional que le das al workflow
- **“on”**: Especifica el evento que automáticamente comienza a ejecutar el fichero de workflow. El ejemplo lo ejecuta gracias al comando **push** de git sobre nuestro repositorio. Para especificar además la rama o ramas sobre las que nos gustaría que iniciase, sería añadiendo:

YAML

```
1  on: [push]

2      Branches: [master]
```

- «**jobs**»: Sección donde se pueden especificar uno o más jobs.
- «**build**»: Es el nombre que le hemos dado a nuestro primer y único job. En este caso el nombre sí es **obligatorio**.
- «**runs on: ubuntu-latest**»: Configura el workflow para que se ejecute en una instancia de la última versión de ubuntu. Se puede cambiar por otro sistema operativo si quisiéramos: *windows-latest*, *macos-11.0*, etc. [Aquí](#) se pueden ver los disponibles.
- «**steps**»: Sección donde se especifican uno o más steps de un único job.
- «**uses: actions/checkout@v2**»: La palabra clave **uses** le dice al job de obtener **v2** (versión 2, antiguamente se usaba la v1) de la acción de la comunidad de Github llamada **actions/checkout**. Éste es un action que comprueba nuestro repositorio y lo descarga en nuestro runner o instancia, permitiendo que sobre el código podamos ejecutar el resto de acciones. Es **obligatorio** añadir este action de **checkout** las veces que nuestro workflow ejecute sobre nuestro código o se haga uso de un action que hemos definido en otro fichero del repositorio.
- «**name: Set up JDK 1.8**»: Un nombre opcional que se le ha dado al action.
- «**uses: actions/setup-java@v1**»: Este action se encarga de descargar e instalar una versión específica de java (**java-version: 1.8** como podemos ver) que la comunidad de Github ya ha preparado para poder utilizarse. [Aquí](#) se pueden consultar además todas las versiones disponibles y sintaxis adicional.

- «**run: mvn -B package — file pom.xml**»: La palabra *run* le dice al job de ejecutar un comando en el runner. En este caso estamos utilizando **maven** para compilar y empaquetar nuestro proyecto.

4. Configuración y ejecución de un workflow

Ahora para lanzar el workflow y ver los resultados, hacemos **push** de nuestro proyecto a la rama en la que lo tengamos (no importa la rama pues como no hemos especificado cuál, lo va a lanzar para todas) tras haber creado la carpeta y añadido el fichero de *workflow.yml*.

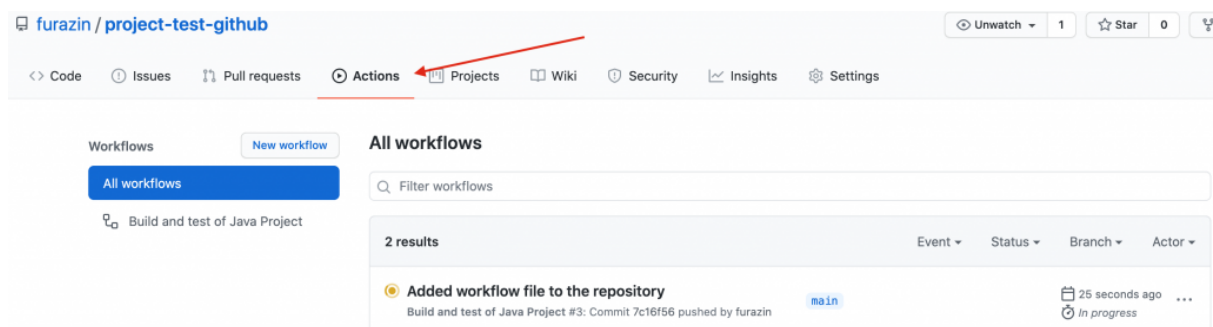
Shell

```
1  git add .

2  git commit -m "Added workflow file to the repository"

3  git push
```

Justo después, nos dirigimos a la sección de **Actions** de nuestra página principal del repositorio en Github.



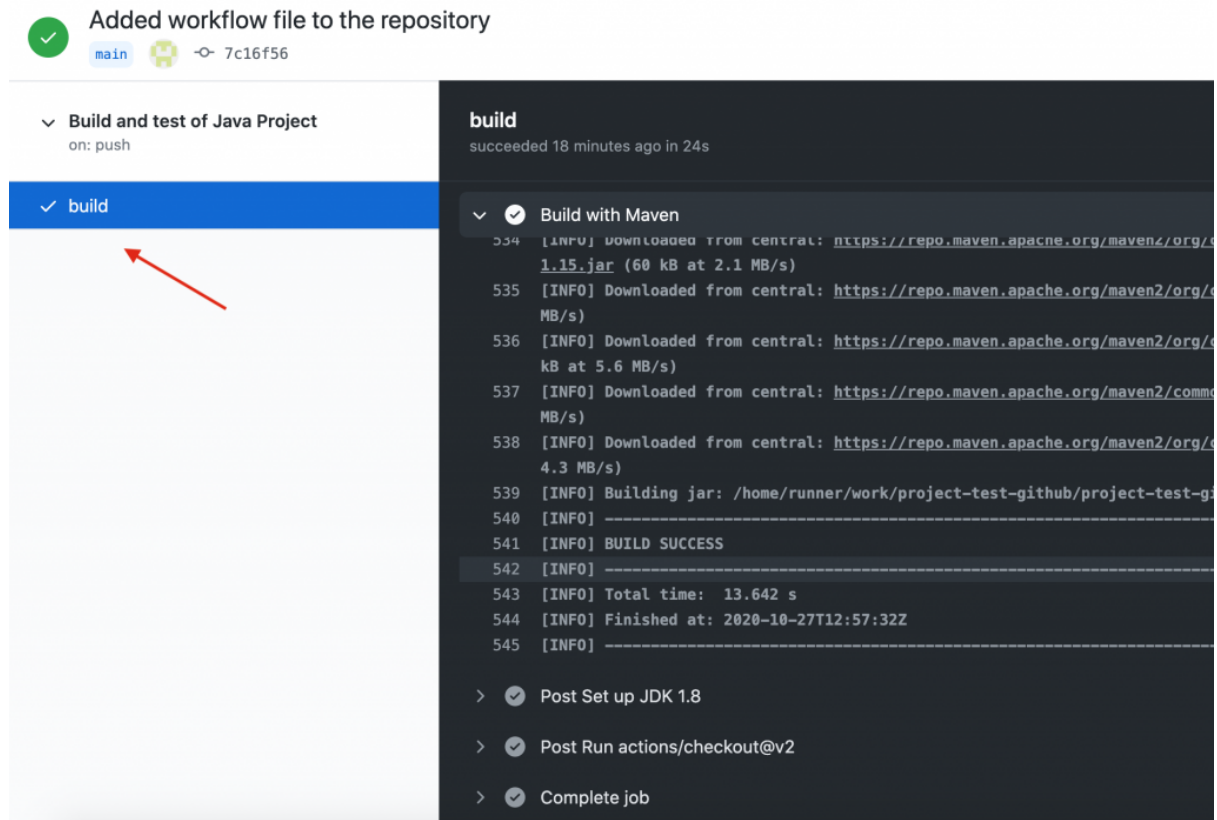
The screenshot shows the GitHub interface for the repository 'furazin/project-test-github'. The 'Actions' tab is selected in the navigation bar, indicated by a red arrow. Below the navigation bar, the 'All workflows' section is visible, showing a search bar and a list of results. The first result is 'Added workflow file to the repository', which is in progress. The workflow is titled 'Build and test of Java Project' and is associated with the commit '7c16f56 pushed by furazin'.

Como vemos, automáticamente se nos ha creado un proceso de **workflow** llamado «Added workflow file to the repository» y que se está ejecutando. El icono amarillo indica que aún no ha acabado.

Una vez termina satisfactoriamente, tiene que aparecer de la siguiente manera:



Y además se pueden consultar los logs de cada uno de los jobs que lo forman pulsando sobre él y luego seleccionando el job que queremos para ver si se ha ejecutado cada paso que hemos especificado.



Además si en nuestro proyecto hemos añadido Tests, éstos se ejecutarán y se mostrarán sus resultados también:

```
490 [INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/  
491 [INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/  
492 2.12.4.jar (37 kB at 1.7 MB/s)  
493 -----  
494 T E S T S  
495 -----  
496 Running com.furazin.projecttestgithub.ArithmeticTest  
497 Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.091 sec  
498  
499 Results :  
500  
501 Tests run: 4, Failures: 0, Errors: 0, Skipped: 0  
502
```

Si alguno de los tests fallan, el workflow nos aparecería con error y tendríamos que arreglar el test y volverlo a lanzar para que nos aparezcan todos los jobs completados con éxito.

5. Construir la imagen con Docker

Github Actions nos da la posibilidad de correr nuestro entorno en una imagen Docker, con todas las ventajas que ello tiene. Para ello vamos a añadir el siguiente fichero de **Dockerfile** a la raíz del proyecto el cual nos compila de la misma manera que hacíamos anteriormente:

YAML

```
1 FROM maven:3.6.0-jdk-8-slim AS build

2 COPY src /usr/src/app/src

3 COPY pom.xml /usr/src/app/

4 RUN mvn -B package --file /usr/src/app/pom.xml

5

6 FROM java:8

7 EXPOSE 8080

8 COPY --from=build /usr/src/app/target/project-test-github-1.0-SNAPSHOT.jar
  /usr/app/project-test-github-1.0-SNAPSHOT.jar

9 ENTRYPOINT ["java","-jar","/usr/app/project-test-github-1.0-SNAPSHOT.jar"]
```

Una vez añadido el fichero, tenemos que modificar nuestro **workflow.yml** para indicar que queremos utilizar Docker a través de un Dockerfile, quedando de la siguiente manera:

YAML

```
1 name: Build and test of Java Project

2 on: [push]

3 jobs:

4   build:

5     name: Build with Docker
```

```
6      runs-on: ubuntu-latest

7      steps:

8          - uses: actions/checkout@v2

9          - name: Building the image from the Dockerfile

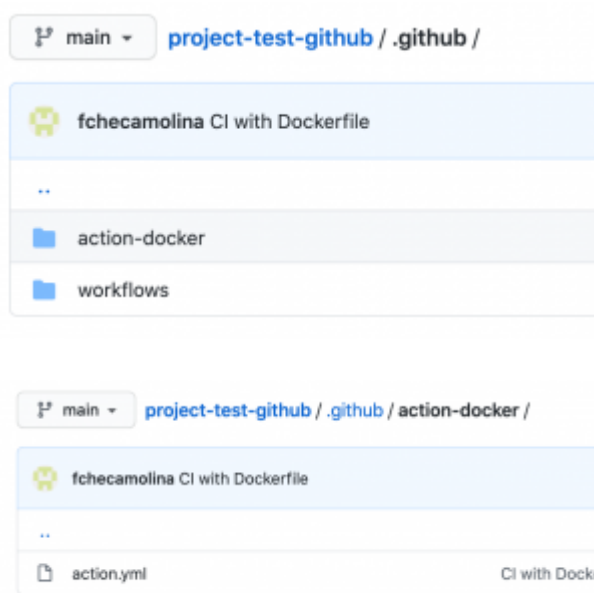
10             uses: ../.github/action-docker

11

12

13
```

Como se puede ver ha disminuido bastante el tamaño del fichero manteniendo el sistema operativo de **ubuntu-latest** que Github nos proporciona ya que cuenta con Docker instalado. Aparte del action de checkout únicamente añadimos otro nuevo action que hace referencia a una nueva carpeta que hemos creado llamada **action-docker**. En esta carpeta vamos a añadir otro YAML con el action que especifica que queremos hacer uso del Dockerfile.



Es muy **importante** saber que el YAML tiene que llamarse ***action.yml*** o ***action.yaml*** sino no nos funcionará. En este [enlace](#) se puede consultar la sintaxis de Github Actions para los ficheros YAML.

El **action.yml** para que nos ejecute nuestra imagen Docker bastaría con añadirlo como:

YAML

```
1  name: "Using Docker"

2  runs:

3      using: "docker"

4      image: "../..../Dockerfile"
```

Indicando en «**image**» donde tenemos nuestro Dockerfile nos lo utilizará para crear la imagen. Si ahora hacemos push de todo esto y no nos hemos

equivocado en nada, en los logs de nuestro workflow del apartado de **Actions** nos muestra cómo compila igual que antes en una imagen Docker.

```
Building the image from the Dockerfile
2 Building docker image
3 Dockerfile for action: '/home/runner/work/project-test-github/project-test-github/.github/action-docker/../../Dockerfile'.
4 /usr/bin/docker build -t 1e5c35:102267eac6f850e5e2ae674a50bf9349 -f "/home/runner/work/project-test-github/project-test-github/.github/action-docker/../../Dockerfile" "/home/runner/work/project-test-github/project-test-github"
5 Sending build context to Docker daemon 157.2kB
6
7 Step 1/8 : FROM maven:3.6.0-jdk-8-slim AS build
8 3.6.0-jdk-8-slim: Pulling from library/maven
9 27833a3ba0a5: Pulling fs layer
10 16d944e3d00d: Pulling fs layer
11 9019de9fce5f: Pulling fs layer
12 9b053055f644: Pulling fs layer
13 1c80aca6b8ec: Pulling fs layer
14 a63811f09e7c: Pulling fs layer
15 f88ce8d36c86: Pulling fs layer
16 a603a4761981: Pulling fs layer
17 f315d92acca3: Pulling fs layer
18 1c80aca6b8ec: Waiting
19 a63811f09e7c: Waiting
20 f88ce8d36c86: Waiting
21 a603a4761981: Waiting
22 f315d92acca3: Waiting
```

```
Building the image from the Dockerfile 43s
641 Status: Downloaded newer image for java:8
642 ----> d23bdf5b1b1b
643 Step 6/8 : EXPOSE 8080
644 ----> Running in 5699a0d906fb
645 Removing intermediate container 5699a0d906fb
646 ----> 773604476adf
647 Step 7/8 : COPY --from=build /usr/src/app/target/project-test-github-1.0-SNAPSHOT.jar /usr/app/project-test-github-1.0-SNAPSHOT.jar
648 ----> 5b863f313971
649 Step 8/8 : ENTRYPOINT ["java","-jar","/usr/app/project-test-github-1.0-SNAPSHOT.jar"]
650 ----> Running in 6130f447a4d4
651 Removing intermediate container 6130f447a4d4
652 ----> fd3e23ec30bd
653 Successfully built fd3e23ec30bd
654 Successfully tagged 1e5c35:102267eac6f850e5e2ae674a50bf9349
655 /usr/bin/docker run --name e5c35102267eac6f850e5e2ae674a50bf9349_e14cac --label 1e5c35 --workdir /github/workspace --rm -e HOME -e
GITHUB_JOB -e GITHUB_REF -e GITHUB_SHA -e GITHUB_REPOSITORY -e GITHUB_REPOSITORY_OWNER -e GITHUB_RUN_ID -e GITHUB_RUN_NUMBER -e
GITHUB_RETENTION_DAYS -e GITHUB_ACTOR -e GITHUB_WORKFLOW -e GITHUB_HEAD_REF -e GITHUB_BASE_REF -e GITHUB_EVENT_NAME -e GITHUB_SERVER_URL -
e GITHUB_API_URL -e GITHUB_GRAPHQL_URL -e GITHUB_WORKSPACE -e GITHUB_ACTION -e GITHUB_EVENT_PATH -e GITHUB_ENV -e RUNNER_OS
-e RUNNER_TOOL_CACHE -e RUNNER_TEMP -e RUNNER_WORKSPACE -e ACTIONS_RUNTIME_URL -e ACTIONS_RUNTIME_TOKEN -e ACTIONS_CACHE_URL -e
GITHUB_ACTIONS=true -e CI=true -v "/var/run/docker.sock":"/var/run/docker.sock" -v "/home/runner/work/_temp/_github_home":"/github/home" -
v "/home/runner/work/_temp/_github_workflow":"/github/workflow" -v "/home/runner/work/_temp/_runner_file_commands":"/github/file_commands"
-v "/home/runner/work/project-test-github/project-test-github":"/github/workspace" 1e5c35:102267eac6f850e5e2ae674a50bf9349
```

6. Secretos

Los secretos de Github Action nos permiten almacenar información sensible en nuestro repositorio para luego poder usarlo en nuestros procesos de

workflow. Son variables de entorno encriptadas por Github, haciendo uso de [libsodium sealed box](#) para asegurar que la información de los secretos se aseguran antes de siquiera ser parte de los repositorios de Github y por supuesto mientras se utilizan en cualquiera de los workflows.

Para añadir un secreto a nuestro repositorio:

1. Navegamos al menú principal del repositorio.
2. Debajo del nombre del repositorio, seleccionamos **Settings**.
3. En el menú de la izquierda seleccionamos **Secrets** y a continuación nos sale el botón **New secret** para añadir un nuevo secreto.

furazin / project-test-github

<> Code ⓘ Issues 🔗 Pull requests ▶ Actions 📁 Projects 📖 Wiki 🛡 Security 📊 Insights ⚙ **Settings**

Options
Manage access
Security & analysis
Branches
Webhooks
Notifications
Integrations
Deploy keys
Secrets
Actions

Secrets

[New secret](#)

Secrets are environment variables that are **encrypted** and only exposed to selected actions. Anyone with **collaborator** access to this repository can use these secrets in a workflow.

Secrets are not passed to workflows that are triggered by a pull request from a fork. [Learn more](#).

There are no secrets for this repository.

Encrypted secrets allow you to store sensitive information, such as access tokens, in your repository.

4. Añadimos nombre y valor y pulsamos sobre **Add secret**. Ahora ya nos aparece creado y listo para poder añadirlo a nuestros pipelines.

Secrets / New secret

Name

NEXUS_USER

Value

user_name

Add secret

 NEXUS_USER

Updated 1 minute ago

Update

Remove