

GITHUB ACTIONS

GitHub Actions es la herramienta que permite reducir la cadena de acciones necesaria para la ejecución de código, y automatizar un ciclo de vida de flujos de trabajo CI/CD de nuestros repositorios GitHub.

El 13 de noviembre de 2019, los ingenieros de Github revelaron algunas noticias: GitHub Actions está soportando CI/CD ahora, ¡y es gratis para repositorios públicos! La API soporta múltiples sistemas operativos (Linux, Windows, MacOS...) y diferentes idiomas.

Por lo tanto, GitHub Actions permite crear workflows que se puedan utilizar para compilar, testear y desplegar código. Además, da la posibilidad de crear flujos de integración y despliegue continuo dentro de nuestro repositorio.

Actions utiliza paquetes de códigos en los contenedores de Docker, los cuales se ejecutan en los servidores de GitHub y que, a su vez, son compatibles con cualquier lenguaje de programación. Esto hace que puedan funcionar con servidores locales y nubes públicas.

Introducción GitHub Actions

Van a ser varias cosas las que necesitemos para empezar a trabajar con GitHub actions:

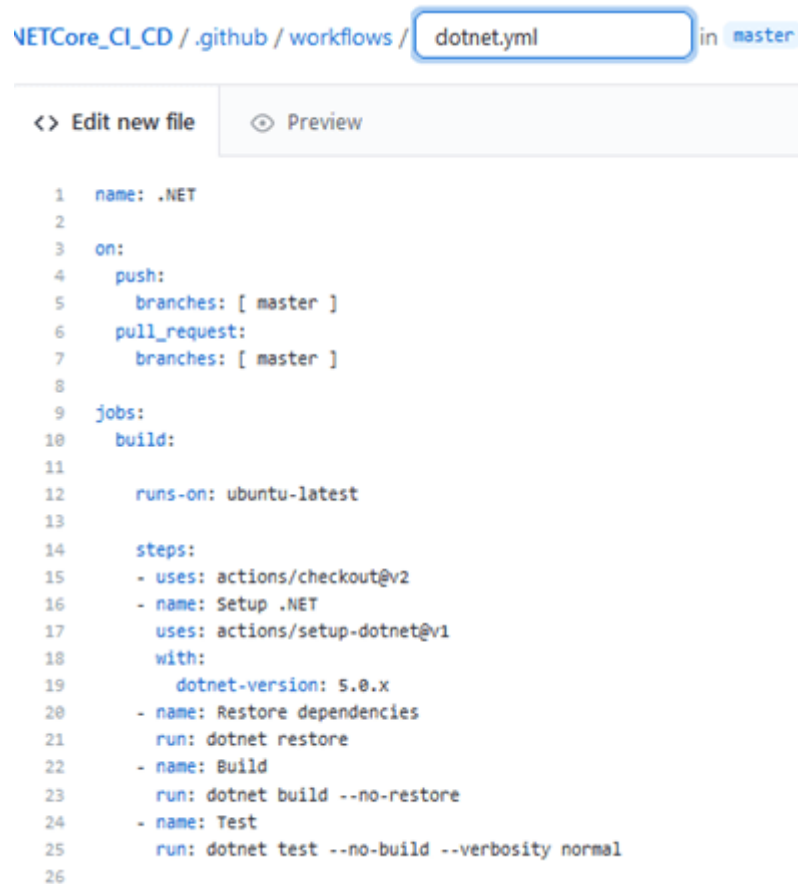
1. Una cuenta y un repositorio de GitHub, obvio, ¿no?
2. Ficheros YAML de definición de nuestros flujos de trabajo, dentro del repositorio.
3. Un agente de compilación de GitHub.

Los YAML de definición de workflows en GitHub Actions

Los ficheros [YAML](#), cuya estructura se basa en espacios y/o tabulaciones, nos permite definir nuestro flujo de CI/CD. A fin de cuentas, estos flujos son una serie de acciones que se ejecutan en una determinada secuencia, para generar nuestras aplicaciones (ya sean binarios o por ejemplo contenedores Docker) y, posteriormente, desplegarlos.

GitHub Actions define una estructura de YAML para ejecutar estas secuencias. Por convención, guardaremos estos ficheros en nuestro repositorio, bajo la

carpeta `.github/workflows` y podremos tener tantos como queramos dentro de nuestro repositorio. Un ejemplo de un fichero muy (pero que muy) básico para generar una aplicación .NET Core, podría ser esto:

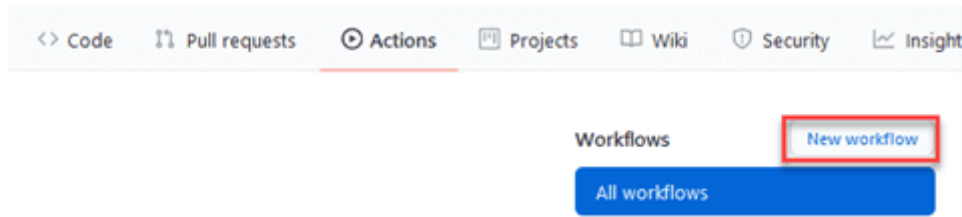


```
1 name: .NET
2
3 on:
4   push:
5     branches: [ master ]
6   pull_request:
7     branches: [ master ]
8
9 jobs:
10  build:
11
12    runs-on: ubuntu-latest
13
14    steps:
15      - uses: actions/checkout@v2
16      - name: Setup .NET
17        uses: actions/setup-dotnet@v1
18        with:
19          dotnet-version: 5.0.x
20      - name: Restore dependencies
21        run: dotnet restore
22      - name: Build
23        run: dotnet build --no-restore
24      - name: Test
25        run: dotnet test --no-build --verbosity normal
26
```

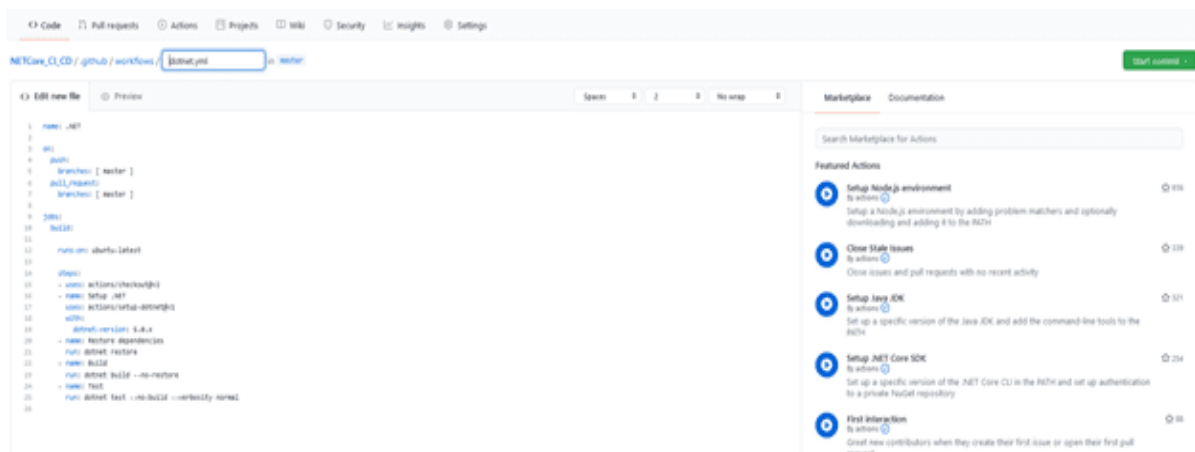
En este sencillo ejemplo vemos las siguientes partes:

1. Definimos el nombre de nuestro flujo (.NET)
2. Definimos qué eventos dentro del repositorio van a provocar la ejecución de este Workflow ([más información](#))
3. Y a continuación, definimos la colección de trabajos, en este caso solo uno: *build*. Que a su vez tiene la siguiente estructura:
 1. El agente (del que hablaremos más adelante) donde se va a ejecutar la secuencia de pasos, en este caso *Ubuntu-latest*.
 2. Las acciones típicas de una aplicación .NET Core, que son las tareas a ejecutar: Setup del framework (5.0) de .NET Core, dotnet restore de sus dependencias, dotnet build, y, por supuesto, la ejecución de test.

Tiene que estar en el directorio predeterminado estos ficheros YAML, pero el modo más sencillo de crearlos es a través del interfaz web de GitHub. Desde la home de nuestro repositorio, pulsaremos en *Actions* y a continuación en *new workflow*.



Al crear un Workflow, existen plantillas predefinidas de Actions, pero también podemos empezar uno de cero. Si escogemos la de .NET Core, que es el ejemplo anterior, se nos mostrará una ventana de edición:



Desde aquí, iremos editando el fichero y añadiendo la secuencia de tareas que queramos. Nos fijamos en la parte de la derecha. Tenemos tareas ya creadas por el propio Github o terceras partes para hacer todo lo que necesitamos: generar contenedores Docker, publicar artefactos, despliegues a cualquiera de las nubes principales.

Agentes de compilación y despliegue en GitHub Actions

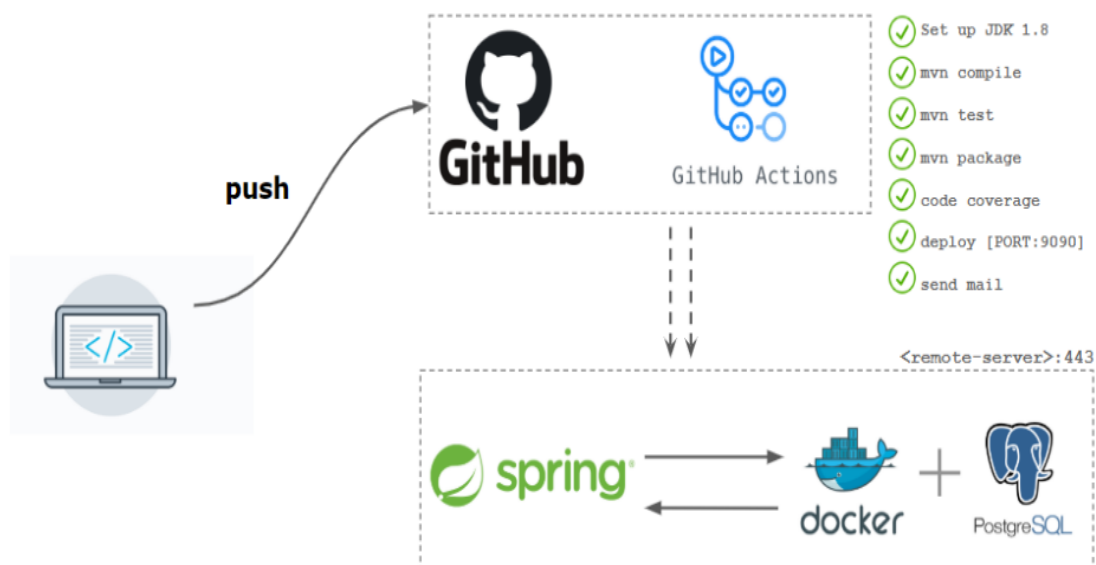
Como decíamos inicialmente, y como veíamos en el YAML, necesitamos un agente donde se ejecutan todos estos pasos. En el ejemplo, teníamos en la línea 12 esto: *runs-on: ubuntu-latest*. Esto dirá al motor de GitHub Actions, que los pasos de ese trabajo se tienen que ejecutar en una máquina Ubuntu, pero ¿de dónde sale esa máquina?

La buena noticia es que el propio Github pone a nuestra disposición, dinámicamente, diversos agentes de compilación que podemos elegir para ejecutar nuestros flujos, y estos son:

- Windows Server 2019: Windows-latest o Windows-2019

- Ubuntu 20.04: Ubuntu-latest o Ubuntu-20.04
- Ubuntu 18.04: Ubuntu-18.04
- Ubuntu 16.04: Ubuntu-16.04
- macOS Big Sur 11.0: macOS-11.0
- macOS Catalina 10.15: macOS-latest o macOS-10.15

Todos estos agentes tienen una serie de características técnicas de hardware y de software instalado, que podés consultar. Pero aún así, si esos agentes no satisfacen esas necesidades del hardware o software, podés también instalar el agente de GitHub Actions en el propios entornos.



Sitios web para ampliar más sobre GitHub Actions:

- ☐ <https://misovirtual.virtual.uniandes.edu.co/codelabs/github-actions/index.html?index=.%2F..practicas#0>
- ☐ <https://www.adictosaltrabajo.com/2020/10/30/como-crear-acciones-utilizando-github-actions/>
- ☐ <https://dzone.com/articles/automate-your-development-workflow-with-github-act-1>
- ☐ [¿Qué son la integración/distribución continuas \(CI/CD\)? \(redhat.com\)](#)

- [Github Actions tutorial español - Como usarlas de manera fácil y GRATUITA!](#)
- [¿Qué es Github Actions? #github #github actions #devops - YouTube](#)

-  Qué son las GITHUB ACTIONS español ft. Alberto Gimeno (Curso de Github) ...