

# Práctica guiada

## Desarrollo ágil de una aplicación web con integración continua

# Contenido

Objetivo de la práctica .....	4
1. Instalar la infraestructura necesaria .....	5
1.1 Crear una cuenta gratuita en github.com.....	5
1.2 Crear una cuenta gratuita educativa en Azure.....	6
1.3 Entorno para pruebas locales en el ordenador de cada desarrollador .....	8
1.3.1 Instalar git.....	8
1.3.2 Instalar visual studio code.....	9
1.3.3 Instalar maven .....	9
1.3.4 Descargar y probar la aplicación de ejemplo.....	9
1.4 Instalar docker.....	11
1.5 Instalar un contenedor con sonarqube .....	11
2. Crear repositorio local y proyecto en github.com .....	13
2.1 Crear repositorio git local.....	13
2.2 Crear repositorio remoto en GitHub.....	14
3. Diseñar el flujo de trabajo para la integración continua (workflow) .....	17
3.1 Trabajo build .....	21
3.1.1 Compilar con un error forzado.....	22
3.1.2 Compilar sin errores.....	23
3.2 Trabajo test .....	25
3.2.1 Realizar una prueba fallida .....	28
3.2.2 Realizar pruebas sin fallos .....	29
3.3 Trabajo qa.....	31
3.3.1 Activar un runner propio en el contenedor ubuntu-sonar.....	31
3.3.2 Definir el trabajo qa.....	35
3.3.3 Aumentar la exigencia de calidad .....	38
3.4 Trabajo deploy .....	43
3.4.1 Despliegue continuo .....	43
3.4.2 Entrega continua.....	51
4. Realizar una segunda versión de la aplicación .....	57
4.1 Planificación ágil: crear historias de usuario (issues).....	57
4.1.1 Crear un proyecto para el repositorio .....	57

4.1.2 Crear un sprint (milestone).....	59
4.1.3 Crear historias de usuario (issues).....	60
4.2 Crear una rama con git para cada issue (feature).....	64
4.2.1 Programar issue “Añadir color de fondo a la página principal” .....	65
4.2.2 Programar issue “Añadir color al título de la página principal” y resolver conflicto.....	70

## Objetivo de la práctica

Con esta práctica vamos a simular la realización de un proyecto de desarrollo de una aplicación web Java aplicando los principios de la Integración Continua y el Desarrollo Ágil.

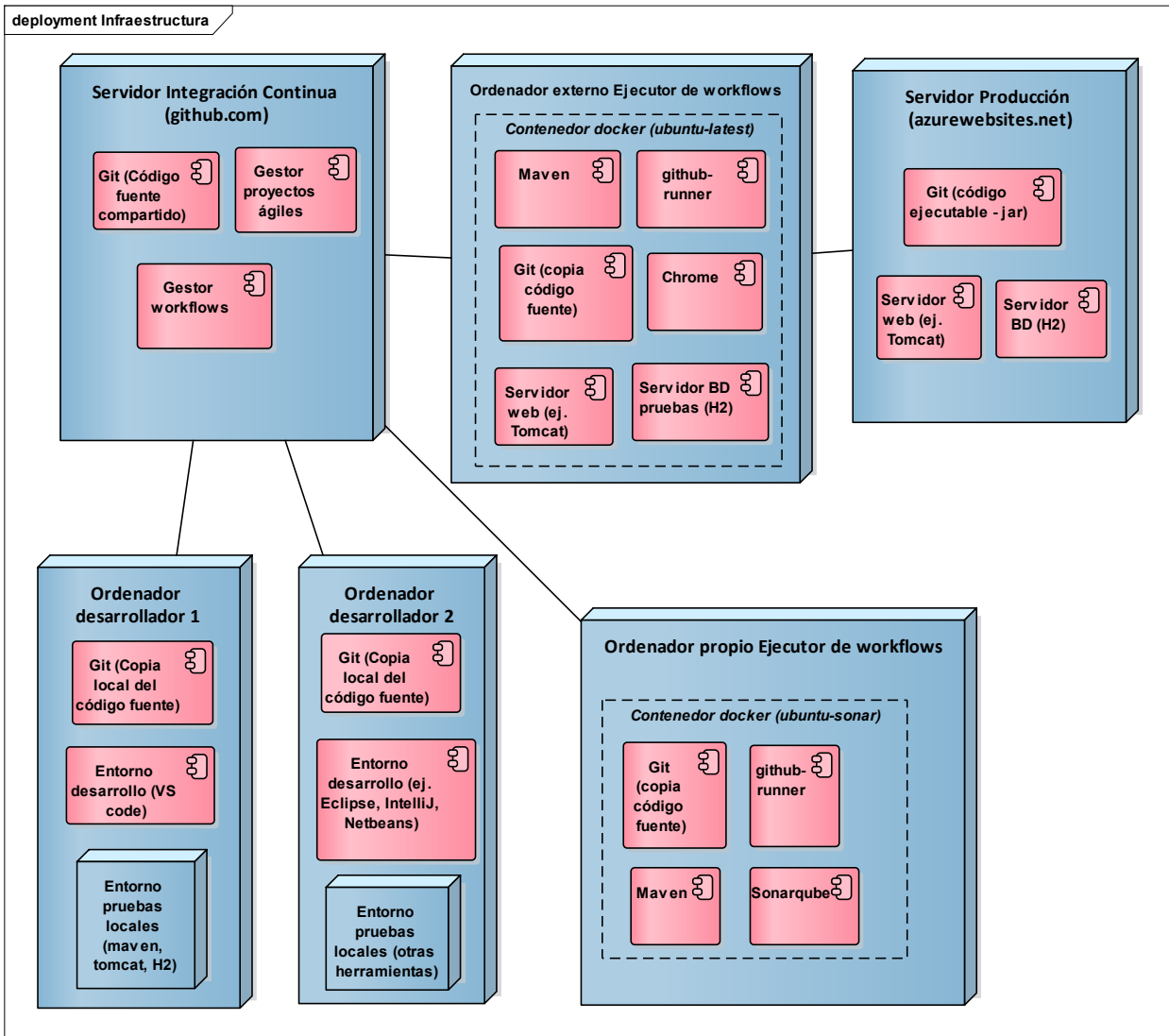
Se utilizarán las siguientes herramientas: Azure, Visual Studio Code, Maven, Git, GitHub Actions, GitHub Actions Runner, Docker, JUnit, Mockito, Selenium y Sonarqube, habituales en el contexto DevOps, en el que se enmarca este trabajo.

Se seguirán los siguientes pasos, cada uno de los cuales se explica en el apartado del documento, que tiene el mismo nombre:

- 1) Instalar la infraestructura necesaria
- 2) Crear repositorio local y proyecto en github.com
- 3) Diseñar el flujo de trabajos para la integración continua (workflow)
- 4) Realizar una segunda versión de la aplicación

# 1. Instalar la infraestructura necesaria

Se utilizará una infraestructura como la de la siguiente imagen. Es un diagrama de despliegue de UML, los cubos representan máquinas (reales o virtuales) y los otros elementos son componentes software.



Utilizaremos un servidor externo en github.com, otro externo en una nube gestionada por github, y otro en azurewebsites.net. Las otras tres máquinas, aunque podrían ser tres ordenadores diferentes, en la práctica serán nuestro ordenador.

En los siguientes subapartados se explica cómo instalar y la razón de usar cada elemento.

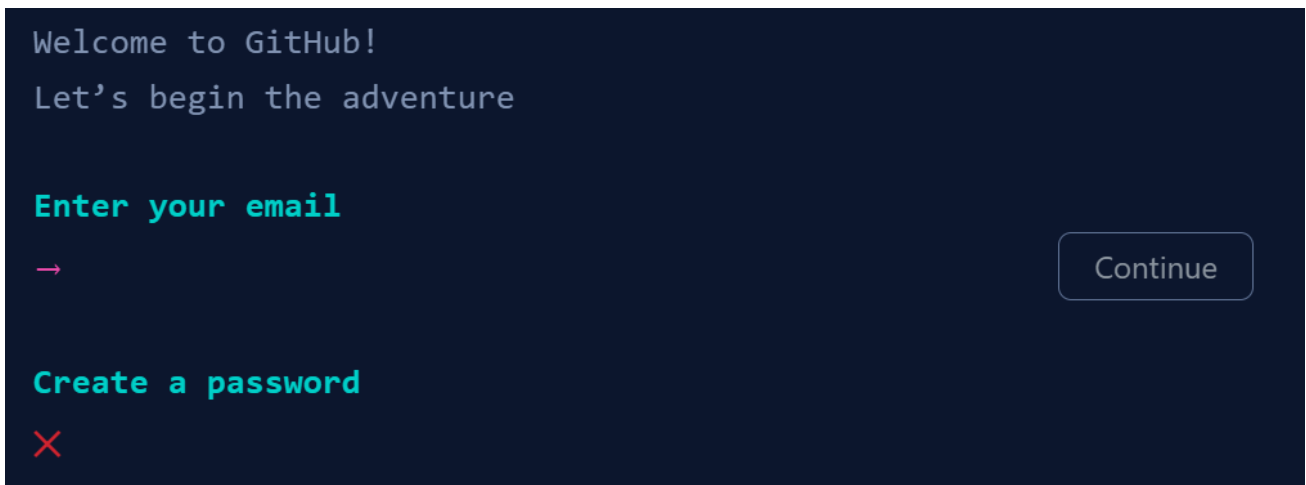
## 1.1 Crear una cuenta gratuita en github.com

En esta práctica se utilizará github.com, porque es una herramienta que ofrece tres funcionalidades en una única máquina:

- Servidor de integración continua
- Repositorio git para almacenar el código fuente del proyecto
- Tableros Kanban para la gestión ágil de un proyecto, aplicando una metodología como Scrum.

En esta práctica se utilizará github.com porque ofrece prácticamente toda la funcionalidad en la nube de forma gratuita si un proyecto es público. Y porque de esta forma el profesor podrá acceder al proyecto y evaluar el trabajo realizado por el alumno.

Para crear una cuenta en github.com hay que acceder a <https://github.com> y elegir Sign in > Create an account.



## 1.2 Crear una cuenta gratuita educativa en Azure

Si se dispone de una cuenta en GitHub se puede crear una cuenta gratuita en la nube Azure de Microsoft para desplegar nuestras aplicaciones cuyo código fuente esté en GitHub y que cualquier persona pueda utilizarlas.

Pero como miembros de la Universidad de Alcalá, podemos acogernos al acuerdo educativo que tiene Microsoft con las universidades y crearnos una cuenta educativa gratuita que no requiere una tarjeta de crédito al registrarse.

Debemos acceder a <https://azure.microsoft.com/es-es/free/students/>, y seleccionar el botón "Empezar gratis".

## Crear soluciones en la nube gratis con Azure for Students

Usa el correo electrónico de tu universidad o centro educativo para registrarte y renovar cada año que seas estudiante.

[Empezar gratis](#)[Más información acerca de los requisitos](#)

En la siguiente página debemos introducir nuestro correo de la uah:



## Iniciar sesión

Correo electrónico, teléfono o Skype

¿No tiene una cuenta? [Cree una.](#)

¿No puede acceder a su cuenta?

[Siguiente](#)

Lo importante es que la dirección de correo sea la de la Universidad de Alcalá, para que nos acepte como usuario estudiante.

Es posible que nos pida un teléfono para confirmar identidad mediante un mensaje, o un CIF, que no hace falta rellenar, y si nos pide una dirección se puede escribir, por ejemplo, "Escuela Politécnica" y Código postal 28805.

La cuenta para estudiantes se crea y se "regala" un crédito de 100 dólares para un año, suficiente para el trabajo a realizar en la asignatura.

The screenshot shows the Microsoft Azure Education portal. At the top, there is a blue header with the Microsoft Azure logo and a search bar containing the text "Buscar recursos, servicios y documentos (G+/)". Below the header, the page title is "Education | Información general". A navigation menu on the left includes "Información general", "Recursos de aprendizaje", "Roles", "Software", "Aprendizaje", and "Plantillas". The main content area is titled "Empezar Información general" and displays "Detalles de la oferta para estudiantes". This section shows "Créditos disponibles" as "100 US\$ de 100 US\$" and "Días hasta que expiren los créditos" as "366".

### 1.3 Entorno para pruebas locales en el ordenador de cada desarrollador

Cada desarrollador trabaja con una copia local del código fuente de la aplicación en su máquina, y cuando hace cambios en la aplicación, antes de enviarlos al repositorio remoto compartido en github.com, debe compilarlos y probarlos previamente en su máquina y asegurarse de que funcionan al menos a nivel local, para lo cual necesitará instalarse git, un entorno de desarrollo, maven, y también su propio servidor web y de base de datos. En este caso no será necesario instalar servidor web ni de base de datos, pues la aplicación se basa en Spring boot con una base de datos en memoria, y un servidor que Maven se encarga de gestionar.

Es importante resaltar que estos servidores locales sólo los usa el desarrollador en su máquina para sus pruebas y no están compartidos con otros desarrolladores, que probablemente tendrán algo parecido en sus respectivos ordenadores. Es habitual que cada miembro del equipo utilice una infraestructura local similar o acordada por el equipo, pero lo importante en integración continua es que todos compartan el repositorio remoto de código, donde se harán las integraciones de los cambios de la aplicación.

#### 1.3.1 Instalar git

Git se necesita para crear y gestionar el repositorio de código del proyecto, así como sus versiones. Es necesario para subir nuevas versiones del código al servidor github.com, que también tiene instalado git, y donde se almacena el repositorio remoto compartido por todos los desarrolladores.

Git también se necesita en local para descargar desde github.com las nuevas versiones del código del proyecto que haya modificado otro desarrollador.

Descargar e instalar desde <https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Instalaci%C3%B3n-de-Git>, eligiendo el sistema que corresponda. Para Windows: <https://git-scm.com/download/win>

Comprobar que está instalado, desde la consola con el comando:

```
C:\> git --version
git version 2.5.2.windows.2
```

### 1.3.2 Instalar visual studio code

Aunque se puede utilizar cualquier entorno de desarrollo para Java, en esta práctica se asume que el desarrollador va a utilizar Visual Studio Code para hacer cambios en la aplicación web. Esta herramienta tiene la ventaja de ofrecer editores para todos los tipos de archivos que habrá que editar en la práctica, y además se integra con git, de tal forma que el desarrollador sabe en cada momento en que rama está trabajando, y los cambios pendientes de actualizar en el repositorio remoto.

Se descarga desde:

<https://code.visualstudio.com/download>

### 1.3.3 Instalar maven

Con Maven se compilará y se ejecutará la aplicación, se puede descargar desde:

<https://maven.apache.org/>

Sólo hay que descargar el archivo apache-maven-bin.zip (de la última versión), descomprimirlo y añadir a la variable de entorno path del sistema operativo la dirección de la subcarpeta bin de maven.

Comprobar que está instalado, desde la consola con el comando:

```
C:\> mvn --version
Apache Maven 3.9.6
```

### 1.3.4 Descargar y probar la aplicación de ejemplo

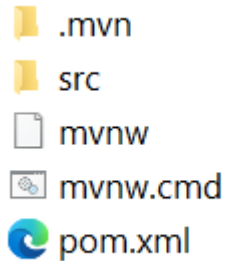
El código fuente de la aplicación a utilizar en la práctica está disponible en el campus virtual, en el archivo [moviecards.zip](#)<sup>1</sup>.

---

<sup>1</sup> La aplicación de ejemplo ha sido desarrollada como parte de un Trabajo Fin de Máster, cuya autora es Laura Cercas, que está disponible en la [biblioteca de la Universidad de Alcalá](#).

Para los ejemplos de esta práctica se descomprimirá en un ordenador Windows, en la raíz de un disco C:, y como programa de comandos se usará cmd de Windows. Se puede utilizar otro lugar del disco, otro sistema operativo y otro intérprete de comandos, como bash, disponible para todos los sistemas.

Al descomprimir se obtiene la carpeta (directorio) “moviecards” con la estructura habitual de un proyecto Java Spring Boot con Maven.



Se puede ejecutar desde la carpeta donde se encuentre el archivo pom.xml, con el comando:

```
C:\>moviecards> mvn spring-boot:run
```

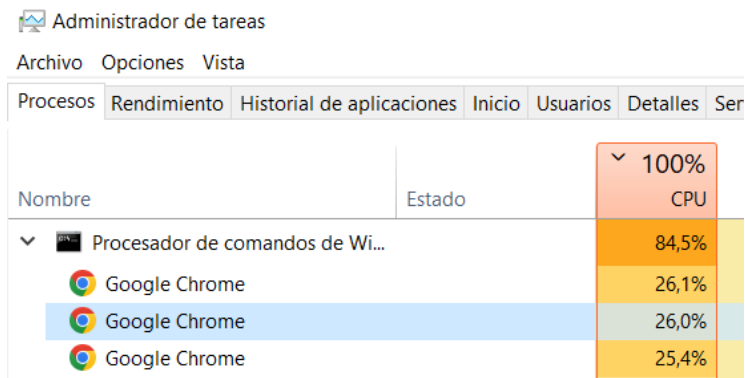
La aplicación está funcionando en el puerto indicado en el archivo de propiedades de spring boot, en src/main/resources/application.properties, que en este caso es el 8089. Por lo que puede abrirse con un navegador, usando la dirección: <http://localhost:8089/>



Para ejecutar las pruebas que se han programado en la carpeta moviecards\src\test del proyecto, sólo hay que ejecutar el siguiente comando:

```
C:\>moviecards> mvn clean verify
```

NOTA: Si se ejecuta en Windows, es posible que haya que acceder al Administrador de tareas y eliminar manualmente los procesos llamados "Google Chrome", ya que los generan las pruebas de la carpeta `src/test/java/com/lauracercas/moviecards/endtoendtest`, pero Windows no los elimina.



## 1.4 Instalar docker

Descargar de <https://docs.docker.com/get-docker/> para el SO que corresponda. En los ejemplos de esta práctica, se supone que se trabaja en un ordenador Windows, pero funcionaría igual en un ordenador Linux.

Comprobar que está instalado, desde la consola con el comando:

```
C:\> docker --version
Docker version 20.10.6, build 370c289
```

## 1.5 Instalar un contenedor con sonarqube

Como se indicaba en la figura de la infraestructura del apartado 1, necesitaremos en la práctica un contenedor con un servidor sonarqube.

Para crearlo podemos utilizar la imagen `jose.hilera/ubuntu-sonar`, disponible en [Docker Hub](https://hub.docker.com/r/josehilera/ubuntu-sonar), con el comando:

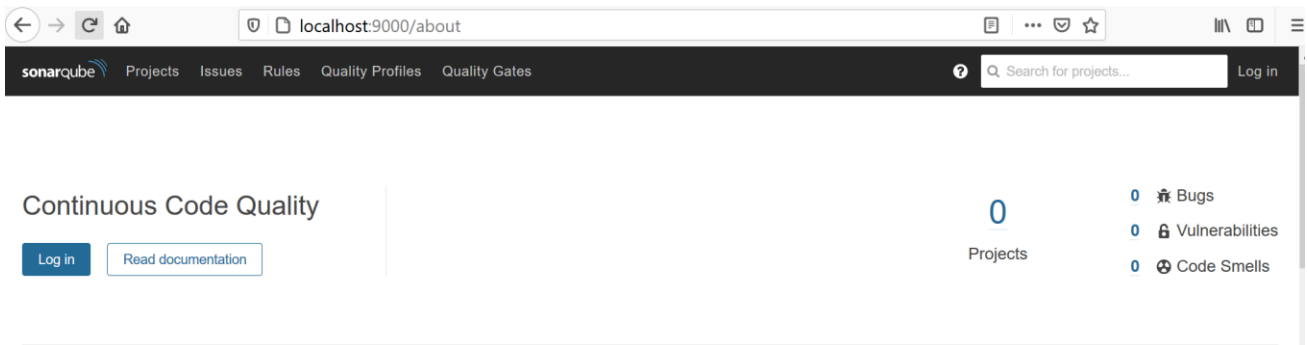
```
C:\> docker run -d --name ubuntu-sonar -p 9000:9000 josehilera/ubuntu-sonar
```

Puede comprobarse que está arrancado con el comando:

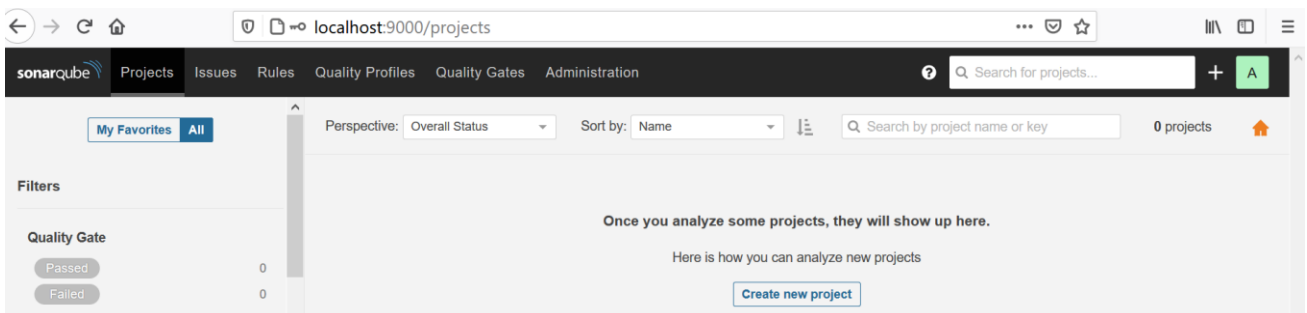
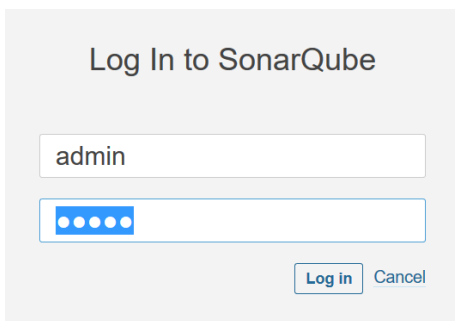
```
C:\> docker ps -a
```

El contenedor está funcionando, es una máquina Linux (Ubuntu) que ha arrancado un servidor Sonarqube en el puerto 9000.

Ahora puede accederse al servidor con un navegador, en la URL <http://localhost:9000>. Es posible que haya que esperar unos minutos hasta que funcione.



Por defecto existe un usuario “admin” con clave “admin”.



Si en algún momento se detiene el contenedor, se puede arrancar de nuevo mediante:

```
C:\> docker start ubuntu-sonar
```

Además, si queremos tener abierta otra consola del contenedor, se puede hacer mediante:

```
C:\> docker exec -it -u root ubuntu-sonar /bin/bash
```

## 2. Crear repositorio local y proyecto en github.com

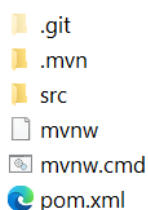
Suponemos que ya tenemos el código fuente en una carpeta llamada moviecards creada en el apartado 1.3.4. En este apartado se trata de convertir dicha carpeta en un repositorio git y subir una copia del mismo a github.com.

### 2.1 Crear repositorio git local

Para convertir la carpeta moviecards en un repositorio git para el control de versiones, y que permita subir (push) y bajar (pull) cambios en el código al repositorio remoto compartido por todos los desarrolladores del proyecto en el servidor de integración continua github.com, se debe ejecutar el comando git init dentro de la carpeta:

```
C:\moviecards> git init
Initialized empty Git repository in C:/moviecards/.git/
```

Se ha creado una carpeta .git oculta con los archivos que necesita git para gestionar versiones del repositorio.



Ahora hay que crear en la carpeta del proyecto, con un editor de texto, un archivo con el nombre .gitignore (IMPORTANTE: sin extensión y con punto delante del nombre), en el que se indicarán los archivos y carpetas completas que no deberán subirse al repositorio remoto cuando se hagan envíos con push, ya que son archivos de uso particular del desarrollador, que han de ser ignorados por git, y no subirlos al remoto.

Por ejemplo, hay que incluir la carpeta /target/, ya que contiene los resultados de la compilación de la aplicación que haga el desarrollador para sus pruebas locales, pero que no hay que subir al repositorio remoto compartido, en el que sólo debe almacenarse el código fuente de la aplicación, no los archivos resultantes de la compilación ni ejecutables. El servidor de integración continua de github.com se encargará de recompilar la aplicación si se aceptan los cambios que envíen los desarrolladores.

Otros archivos que hay que incluir en .gitignore son aquellos que genere el entorno de programación que utilice el desarrollador. Puede ocurrir que cada desarrollador de un mismo proyecto utilice en su ordenador un entorno de desarrollo diferente (Visual Studio Code, Netbeans, Eclipse, IntelliJ, etc.), que crea archivos en la carpeta del proyecto que no hay que subir al repositorio remoto, en el que el código compartido por todos los desarrolladores debe ser independiente de cualquier entorno de desarrollo. Por ejemplo, si se desarrolla localmente VS Code, seguramente existirá una carpeta .vscode, si se desarrolla con IntelliJ IDEA, existirá la carpeta .idea, etc, que habrá que ignorar.

Para evitar que se suban al repositorio remoto, el archivo .gitignore debería ser el siguiente, indicando en cada fila un archivo o carpeta que git debe ignorar al hacer push hacia el repositorio remoto.

```
Archivo C:\moviecards\.gitignore

/target

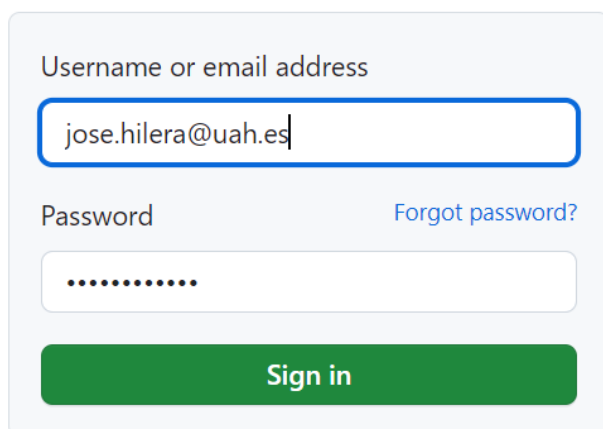
### VS Code ###
/.vscode

### IntelliJ IDEA ###
/.idea/
```

## 2.2 Crear repositorio remoto en GitHub

Hay que tener una cuenta en [github.com](https://github.com), en los ejemplos de esta práctica se utilizará la cuenta jose.hilera. Cada alumno debe utilizar la suya.

Sign in to GitHub



Username or email address

Password [Forgot password?](#)

**Sign in**


Una vez dentro de GitHub, para crear un repositorio se selecciona el botón New, indicando como nombre “moviecards”, visibilidad “Public”, **no marcar** “Add a README file”.

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).


Owner \*      Repository name \*


 josehilera /

✔ moviecards is available.

Great repository names are short and memorable. Need inspiration? How about [shiny-happiness](#) ?

Description (optional)

 **Public**  
Anyone on the internet can see this repository. You choose who can commit.

 **Private**  
You choose who can see and commit to this repository.

Initialize this repository with:

**Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Al pulsar el botón para crear el proyecto aparecen sugerencias de comandos para subir un repositorio existente: "Push an existing repository from the command line". Que podremos ir copiando y pegando en la consola de nuestro ordenador local.

Volvemos a la consola de nuestro ordenador y dentro de la carpeta moviecards ejecutamos los siguientes comandos:

```
C:\moviecards> git remote add origin https://github.com/josehilera/moviecards.git
C:\moviecards> git add .
C:\moviecards> git status
C:\moviecards> git commit -m "Commit inicial"
C:\moviecards> git push -u origin master
Username for 'https://github.com': jose.hilera@uah.es
Password ...
```

<sup>2</sup> Si no queremos tener que introducir usuario y contraseña cada vez que hagamos git push, se pueden ejecutar los siguientes comandos:

```
git config --global user.name "nombre de usuario"
git config --global user.password "contraseña"
```

Si volvemos a github.com, vemos que se ha subido una réplica del repositorio local, excepto la carpeta oculta .git, y los archivos y subcarpetas indicados en .gitignore.

The screenshot shows the GitHub interface for a repository named 'moviecards' by user 'josehilera'. The repository is public and has 1 branch (master) and 0 tags. The commit history shows an initial commit (1e2dfc2) made 1 minute ago. The commit includes the following files and folders:

File/Folder	Commit	Time
.mvn/wrapper	Commit inicial	1 minute ago
src	Commit inicial	1 minute ago
.gitignore	Commit inicial	1 minute ago
mvnw	Commit inicial	1 minute ago
mvnw.cmd	Commit inicial	1 minute ago
pom.xml	Commit inicial	1 minute ago

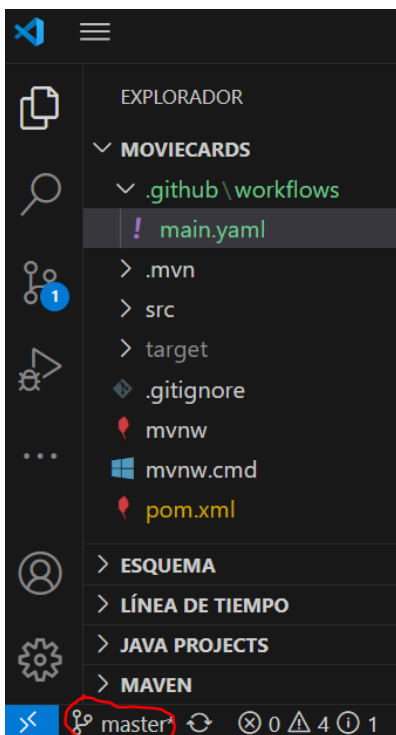
### 3. Diseñar el flujo de trabajo para la integración continua (workflow)

GitHub Actions actúa como servidor de integración continua, y en el repositorio creado todos los miembros de proyecto irán subiendo cambios.

Cada vez que se sube un cambio hay que compilar, probar y, en su caso, desplegar la aplicación a producción. Un principio de la integración continua es que el repositorio tenga todo lo necesario para recompilar y ejecutar la aplicación, que ha podido funcionar en el ordenador local, pero se trata de que funcione en uno o varios servidores web externos, bajo el control del servidor de integración.

Para que cada vez que se suba un cambio se recompile la aplicación, hay que preparar un flujo de trabajo (workflow) en un archivo con el nombre **main.yaml**<sup>3</sup> dentro de una carpeta que hay que crear con el nombre **.github\workflows\** (**IMPORTANTE: debe empezar por punto**).

Este archivo lo creamos en el repositorio local con un editor de texto o un entorno de desarrollo. Se recomienda usar Visual Studio Code, pues ofrece utilidades para sincronizarse con el repositorio remoto, y nos indica en la parte inferior izquierda en que rama del proyecto estamos trabajando en cada momento. En este caso, si abrimos la carpeta moviecards con este editor, podemos ver que estamos en la rama master:



Escribimos en el archivo main.yaml el siguiente contenido:

---

<sup>3</sup> El archivo puede tener otros nombres y extensión yaml o yml. Puede haber varios archivos yaml.

### Archivo C:\moviecards\.github\workflows\main.yaml

```
name: CI

on: push

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: empaquetado
        run: echo "empaquetado"

  test:
    needs: build
    runs-on: ubuntu-latest
    steps:
      - name: pruebas
        run: echo "pruebas"

  qa:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - name: calidad
        run: echo "calidad"

  deploy:
    needs: qa
    runs-on: ubuntu-latest
    steps:
      - name: despliegue
        run: echo "despliegue"
```

El significado del contenido es el siguiente<sup>4</sup>:

- En la sección “name” se indica el nombre del workflow, en este caso “CI”.
- En la sección “on” se define cuándo el workflow será ejecutado, en este caso cada vez que se haga un push a cualquier rama del repositorio.
- En la sección “jobs” se indican los 4 trabajos a realizar: build, test, qa, deploy.
- Para evitar que los 4 trabajos se ejecuten en paralelo, se utiliza la sección “needs” para indicar que un trabajo no se debe ejecutar hasta que haya finalizado el citado en esta sección. Cada trabajo se divide en pasos o “steps”. En este caso todos los trabajos constan sólo de un paso.
- En la sección “run-on” se indica qué runner debe ejecutar los comandos de cada trabajo. Hemos indicado “ubuntu-latest” en todos los trabajos, para establecer que todos los comandos sean ejecutados en un runner gratuito incluido en un contenedor Docker llamado “ubuntu-latest”, hospedado por GitHub en algún lugar de Internet. Otra posible opción sería indicar “self-hosted”, si dispusiéramos de un runner propio instalado en un algún ordenador propio con conexión a Internet, como veremos en un apartado posterior. Cuando se diseñen flujos de

---

<sup>4</sup> Ver <https://docs.github.com/es/actions/using-workflows/workflow-syntax-for-github-actions>

trabajo complejos, con muchos trabajos y muchos comandos, lo conveniente será utilizar nuestro propio runner, porque el runner gratuito que ofrece GitHub tiene limitaciones<sup>5</sup>.

Las actividades que realizará cada trabajo consisten en comandos, en este caso de Linux, que ejecutará la máquina en la que se ha instalado el runner asignado al trabajo. Estos comandos se indican en la sección run de cada step de cada trabajo (job).

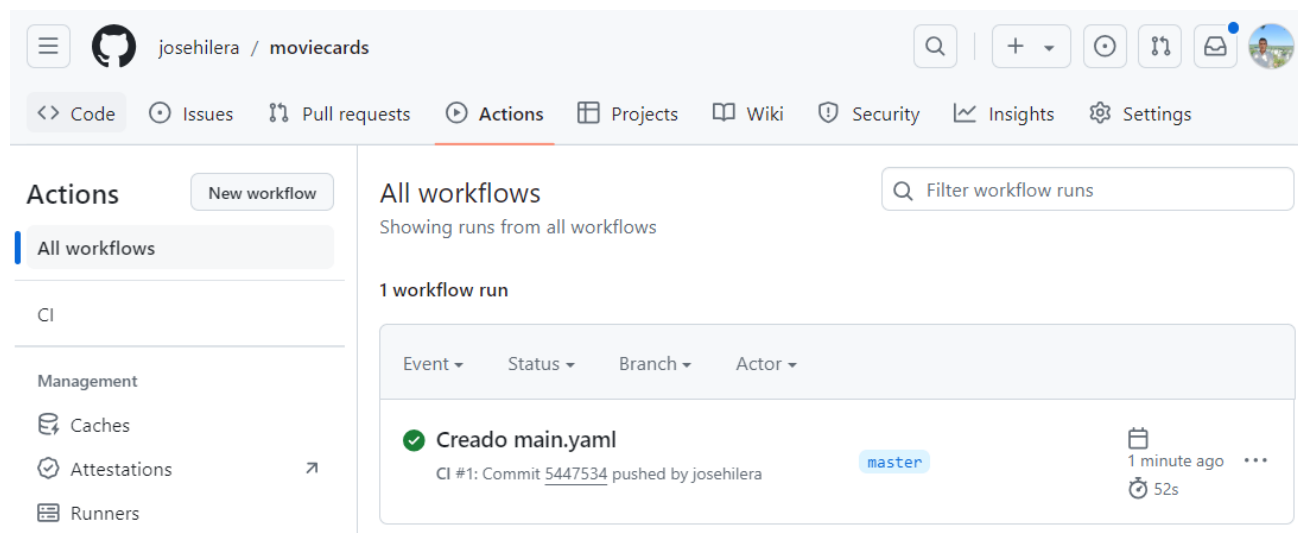
De momento, en el archivo main.yaml sólo hemos definido en los trabajos un comando echo que muestre el nombre de cada paso. En los siguientes apartados los sustituiremos por comandos que realicen de verdad el trabajo previsto en cada caso.

Los trabajos definidos en main.yaml se ejecutarán de forma automática cada vez que un desarrollador envíe cambios al repositorio remoto desde su repositorio local mediante un push.

Como acabamos de modificar el repositorio local añadiendo este nuevo archivo, podemos hacer un push contra el repositorio remoto para actualizar los cambios, y comprobaremos que se ejecutan los trabajos definidos en el archivo.

```
C:\moviecards> git add .
C:\moviecards> git status
C:\moviecards> git commit -m "Creado main.yaml"
C:\moviecards> git push
```

En la sección Actions > All workflows de GitHub podemos ver las ejecuciones de los workflows de cada push realizado sobre el repositorio, en este caso el correspondiente al commit "Creado main.yaml".



Puede comprobarse que la ejecución del workflow se marca como éxito (icono en verde).

<sup>5</sup> El runner gratuito tiene algunas limitaciones de uso, como se indica en <https://docs.github.com/es/actions/learn-github-actions/usage-limits-billing-and-administration>.

Si seleccionamos el propio workflow, aparece un diagrama con los trabajos realizados:



Todos están en verde indicando que han pasado con éxito. Si seleccionamos uno de los trabajos, por ejemplo “build”, podemos ver la consola de la máquina que ha ejecutado el trabajo

Podemos comprobar que es una máquina Ubuntu, pero no es una nuestra sino una máquina de GitHub, ya que en ese trabajo indicamos que se ejecutara en un runner de GitHub en una máquina con la última versión de Ubuntu.

En los siguientes apartados vamos a ir definiendo el detalle de los trabajos a realizar, modificando el archivo main.yaml.

### 3.1 Trabajo build

En este trabajo de construcción de la aplicación hay que realizar los siguientes pasos:

- 1) Descargar en el runner<sup>6</sup> todos los archivos con el código fuente del repositorio
- 2) Instalar JDK en el runner
- 3) Compilar y empaquetar la aplicación web en el runner
- 4) Subir al repositorio en GitHub el paquete jar generado con la aplicación compilada

En el archivo main.yaml hay que realizar los siguientes cambios.

```
Archivo C:\moviecards\github\workflows\main.yaml
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Descargar repositorio
        uses: actions/checkout@v2

      - name: Instalar JDK 11
        uses: actions/setup-java@v2
        with:
          java-version: "11"
          distribution: "adopt"

      - name: Construir con Maven
        run: mvn clean package -DskipTests

      - name: Guardar paquete generado para el trabajo de despliegue
        uses: actions/upload-artifact@v4
        with:
          name: moviecards-java
          path: "${{ github.workspace }}/target/*.jar"
```

Hemos añadido un primer paso “Descargar repositorio” para que el runner que ejecute el trabajo (ubuntu-latest) pueda acceder a los archivos del proyecto (.java, .jsp, .html, ..) para poderlos usar en los siguientes pasos. En lugar de escribir comando para ello, incluimos una sección “uses” para utilizar comandos creados por otros autores y disponibles en forma de lo que se denominan “actions”, que son similares al concepto de librerías de funciones, en este caso para GitHub Actions. De esta forma nos ahorramos tener que escribir los comandos necesarios para realizar ese paso. Estas acciones reutilizables se encuentran en el [Marketplace de GitHub](#).

En nuestro caso, hemos insertado una referencia a la acción llamada “checkout”, en su versión 2, disponible en <https://github.com/actions/checkout>.

---

<sup>6</sup> Para simplificar usaremos sólo el término “runner” cuando nos refiramos al contenedor (en este caso, ubuntu-latest) que contiene el runner.

Esta acción contiene los comandos necesarios para que el runner que ejecuta un trabajo reciba todo el contenido del repositorio de nuestro proyecto, es decir todo el código fuente de la aplicación, y lo guarda en la carpeta de trabajo que se indicó al crear el runner (por defecto, el nombre es `_work`), para poder trabajar con dichos archivos, por ejemplo, para compilarlos o para ejecutar las pruebas, como va a ser en este caso.

Se ha incluido como segundo paso una referencia la acción llamada [setup-java](#), para instalar en el runner la versión de JDK 11, para estar seguros de que el compilador que se use sea el mismo que el que indica el archivo `pom.xml` del proyecto:

```
<properties>
  <java.version>11</java.version>
</properties>
```

Para compilar y empaquetar la aplicación en un archivo `moviecards.jar`, usamos un comando Maven. En el comando `mvn` indicamos la opción `DskipTest` a `true`, para que no se realicen las pruebas unitarias, pues se realizarán en el trabajo “test”.

Por último, se usa la acción [upload-artifact](#) para que el paquete `jar` obtenido al compilar se guarde en el repositorio, porque hará falta usarlo en la fase o trabajo de despliegue, en la que habrá que enviar el paquete al servidor en el que se vaya a instalar la aplicación.

Todavía no hacemos `push` contra el repositorio remoto, lo haremos en el siguiente apartado.

### 3.1.1 Compilar con un error forzado

Vamos a ver cómo falla el trabajo de construcción de la aplicación forzando un error de compilación. Para ello modificamos con el editor el archivo `src/main/java/com/lauracercas/moviecards/MovieCardsApplication.java` en nuestro repositorio local, comentando la segunda línea de código para forzar un error de compilación.

```
//import org.springframework.boot.SpringApplication;
```

Salvamos todo y lo subimos al repositorio remoto:

```
C:\moviecards> git add .
C:\moviecards> git status
C:\moviecards> git commit -m "Error de compilación forzado"
C:\moviecards> git push
```

Vemos en `Actions > All workflows` de GitHub, que se ha ejecutado de nuevo el workflow, y que se ha producido un error en el trabajo “build”:

The screenshot shows the GitHub Actions interface for the repository 'josehilera / moviecards'. The 'Actions' tab is selected, displaying a workflow named 'CI' with a failed job 'Error de compilación forzado #4'. The job summary indicates it was triggered via push 5 minutes ago, failed, and had a total duration of 27s. A 'main.yaml' file is shown with a workflow triggered on push, containing jobs for 'build', 'test', and 'qa'. The 'build' job is marked as failed.

Ello supone que habrá que revisar los errores y corregirlos. En la consola del trabajo build se puede ver cuál ha sido el error:

The screenshot shows the console output for the failed 'build' job. The job is titled 'Construir con Maven' and failed 3 minutes ago in 18s. The output shows a compilation error: 'cannot find symbol' for the variable 'SpringApplication' in the class 'com.lauracercas.moviecards.MovieCardsApplication'. The error message is highlighted with a red circle.

### 3.1.2 Compilar sin errores

Para corregir el error, editamos de nuevo en local quitando el comentario de `src/main/java/com/lauracercas/moviecards/MovieCardsApplication.java`:







```
//import org.springframework.boot.SpringApplication;
```

Guardamos y lo volvemos a subir:




```
C:\moviecards> git add .
```

```
C:\moviecards> git status
C:\moviecards> git commit -m "Error de compilación corregido"
C:\moviecards> git push
```

Comprobamos que se ejecuta el workflow y que ya no hay ningún error.

 <b>Error de compilación corregido</b> CI #5: Commit <a href="#">38e4b67</a> pushed by josehilera	master	 2 minutes ago ...  1m 16s
 <b>Error de compilación forzado</b> CI #4: Commit <a href="#">9561404</a> pushed by josehilera	master	 13 minutes ago ...  27s

Como indicamos en main.yaml que queríamos guardar el paquete jar generado en el trabajo build ejecutado por el runner antes de que se borre al finalizar el trabajo, podemos comprobar que ese artefacto se encuentra al final de la página del workflow, en la sección Artifacts.

Artifacts		
Produced during runtime		
Name	Size	
 moviecards-java	42.5 MB	 

Podemos descargarlo: se trata de un zip en el que se incluye el archivo moviecards-0.0.1-SNAPSHOT.jar, con la aplicación empaquetada. Más adelante en la práctica veremos cómo se puede usar ese artefacto en otro trabajo posterior.

## 3.2 Trabajo test

El siguiente trabajo definido en el workflow es test, que se realiza una vez superada la compilación sin errores. En este trabajo vamos a programar varios pasos:

- 1) Descargar en el runner, todos los archivos con el código fuente del repositorio
- 2) Instalar JDK en el runner (porque se va a volver a compilar)
- 3) Instalar el runner el navegador Chrome y la librería ChromeDriver para las pruebas funcionales end to end, para poder simular el uso de la aplicación por parte de un usuario que utiliza un navegador web
- 4) Ejecutar la aplicación en el runner para poder realizar las pruebas end to end
- 5) Ejecutar en el runner todas las pruebas unitarias, de integración y end to end

Las pruebas están ya programadas en la carpeta `src/test/` del repositorio del proyecto. Son las siguientes:

- Pruebas unitarias: en la subcarpeta `unittest`
- Pruebas de integración: en la subcarpeta `integrationtest`
- Pruebas funcionales end to end: en la subcarpeta `endtoendtest`

Hay que modificar el archivo `main.yaml` cambiando el trabajo test por el código siguiente:

### Archivo C:\moviecards\.github\workflows\main.yaml

```
test:
  needs: build
  runs-on: ubuntu-latest
  steps:
    - name: Descargar repositorio
      uses: actions/checkout@v2

    - name: Instalar JDK 11
      uses: actions/setup-java@v2
      with:
        java-version: "11"
        distribution: "adopt"

    - name: Instalar Chrome y ChromeDriver para pruebas end to end
      run: |
        wget https://dl.google.com/linux/direct/google-chrome-
        stable_current_amd64.deb

        sudo dpkg -i google-chrome-stable_current_amd64.deb

        sudo apt --fix-broken install -y

        CHROMEDRIVER_VERSION=$(curl -sS
        https://chromedriver.storage.googleapis.com/LATEST_RELEASE)

        curl -L -o chromedriver.zip
        https://chromedriver.storage.googleapis.com/$CHROMEDRIVER_VERSION/chr
        omedriver_linux64.zip

        unzip chromedriver.zip

        chmod +x chromedriver

        sudo mv chromedriver /usr/local/bin/

    - name: Ejecutar la aplicación para pruebas end to end
      run: mvn spring-boot:run & sleep 60

    - name: Ejecutar las pruebas unitarias, de integración y end to end
      run: mvn clean verify
```

Debemos guardar los cambios del archivo main.yaml y subirlos a GitHub:

```
C:\moviecards> git add .
C:\moviecards> git status
C:\moviecards> git commit -m "Trabajo test creado"
C:\moviecards> git push
```

Cuando se termine de ejecutar el trabajo, que puede tardar varios minutos porque son muchas pruebas, podemos comprobar que todas las pruebas han pasado sin errores.



### 3.2.1 Realizar una prueba fallida

Para comprobar que las pruebas detectan los errores cometidos por el programador, vamos a modificar parte del código de la aplicación para que falle una de las pruebas unitarias.

Por ejemplo, podemos modificar en el archivo `src/main/java/com/lauracercas/moviecards/model/Actor.java` el siguiente código:

```
public String getName() {  
    return name;  
}
```

Por

```
public String getName() {  
    return "Hola";  
}
```

Una vez guardado el archivo `Actor.java`, hacemos push al repositorio remoto, para ver que al desencadenarse de nuevo el workflow, se interrumpe el trabajo test.

```
C:\moviecards> git add .  
C:\moviecards> git status  
C:\moviecards> git commit -m "Fallo forzado en prueba unitaria"  
C:\moviecards> git push
```

Puede comprobarse en el nuevo workflow que falla el trabajo test:

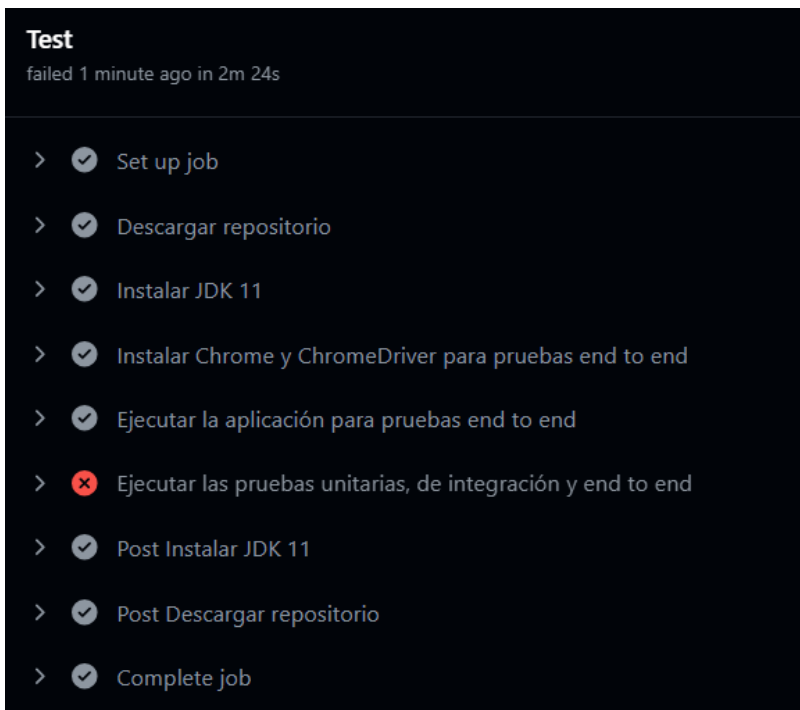
The screenshot shows a GitHub Actions workflow run titled "Fallo forzado en prueba unitaria #9". The workflow was triggered via push 15 minutes ago by user josehilera. The status is "Failure" and the total duration is 3m 22s. The workflow file is `main.yaml` with the trigger `on: push`. The workflow consists of two jobs: `build` (39s) and `Test` (2m 24s). The `Test` job failed, as indicated by the red 'x' icon.

Triggered via	Status	Total duration
push 15 minutes ago	Failure	3m 22s

Jobs:

- build (39s) - Success
- Test (2m 24s) - Failure
- qa
- deploy

Si accedemos al detalle del trabajo test, puede verse que ha fallado el paso "Ejecutar las pruebas unitarias, de integración y end to end".



En el archivo de logs, se comprueba que ha fallado la prueba testSetName.

```
[INFO] Running com.lauracercas.moviecards.unittest.model.ActorTest
[ERROR] Tests run: 5, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.004 s <<< FAILURE! - in com.lauracercas.moviecards.unittest.model.ActorTest
[ERROR] testSetName Time elapsed: 0 s <<< FAILURE!
org.opentest4j.AssertionFailedError: expected: <Sample name> but was: <Hola>
    at com.lauracercas.moviecards.unittest.model.ActorTest.testSetName(ActorTest.java:29)
```

### 3.2.2 Realizar pruebas sin fallos

Para que no falle la prueba, hay que corregir el cambio realizado en `src/main/java/com/lauracercas/moviecards/model/Actor.java`:

```
public String getName() {
    return "Hola";
}
```

Por

```
public String getName() {
    return name;
}
```

Se guarda el archivo y hay que volver a subir al repositorio remoto:

```
C:\moviecards> git add .
C:\moviecards> git status
C:\moviecards> git commit -m "Pruebas sin fallos"
C:\moviecards> git push
```

Puede comprobarse que este workflow se realiza sin problemas.

## ✓ Pruebas sin fallos #10

R

### Summary

Jobs

- ✓ build
- ✓ Test
- ✓ qa
- ✓ deploy

Run details

- Usage
- Workflow file

Triggered via push 3 minutes ago

Status	Total duration
Success	3m 40s

🌐 josehilera pushed -> bdfaae5 master

### main.yaml

on: push

```
graph LR; build[✓ build 32s] --> Test[✓ Test 2m 32s]
```

### 3.3 Trabajo qa

Para poder realizar este trabajo, necesitamos utilizar el servidor sonarqube que ya está instalado en el contenedor ubuntu-sonar creado en el apartado 1.5, y que está funcionando en <http://localhost:9000>.

Si se hubiera detenido el contenedor, se puede arrancar de nuevo mediante:

```
C:\> docker start ubuntu-sonar
```

#### 3.3.1 Activar un runner propio en el contenedor ubuntu-sonar

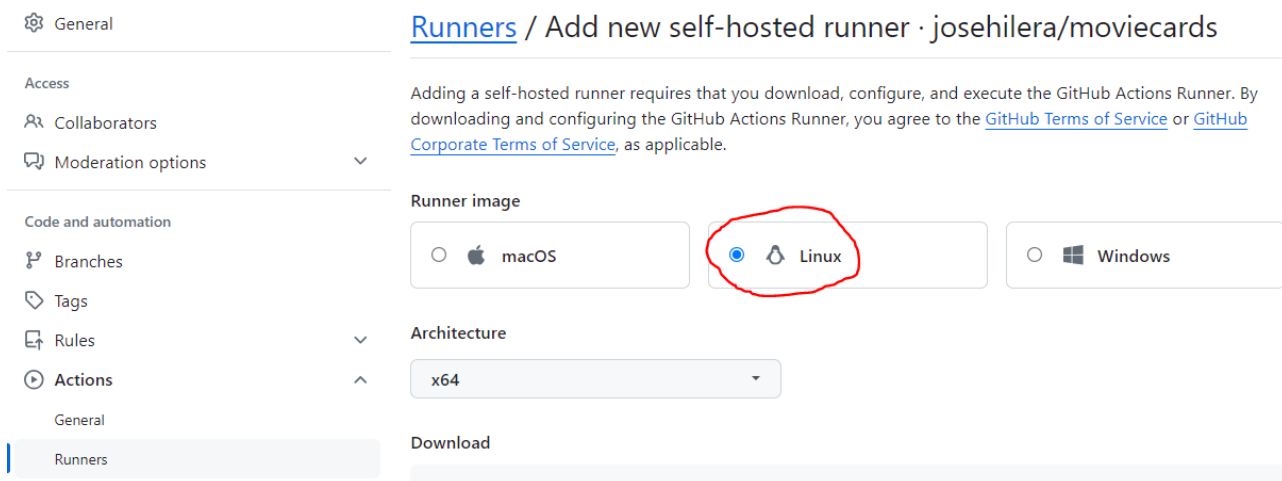
GitHub Actions utiliza la aplicación externa [actions-runner](#) para ejecutar las órdenes de los workflows asociados al repositorio. Ese servicio puede estar instalado en un ordenador Linux, Windows o Mac con conexión a internet. En esta práctica se va a instalar en el contenedor ubuntu-sonar que ya está creado y arrancado.

Para abrir una consola del contenedor, se tiene que ejecutar:

```
C:\> docker exec -it -u root ubuntu-sonar /bin/bash
```

La aplicación actions-runner se instala en el contenedor ubuntu-sonar con unos comandos que hay que copiar y pegar desde nuestro repositorio en GitHub.

Hay que acceder a la sección Settings > Actions > Runners del repositorio y seleccionar “New self-hosted runner”, y en esa pantalla la opción Linux.



En la sección Download se indican los comandos a ejecutar para descargar e instalar en el contenedor la última versión del programa de creación de runners:

```
# mkdir actions-runner && cd actions-runner

# curl -o actions-runner-linux-x64-2.317.0.tar.gz -L
https://github.com/actions/runner/releases/download/v2.317.0/actions-runner-
linux-x64-2.317.0.tar.gz
```

```
# tar xzf ./actions-runner-linux-x64-2.317.0.tar.gz
```

## Download

```
# Create a folder
$ mkdir actions-runner && cd actions-runner
# Download the latest runner package
$ curl -o actions-runner-linux-x64-2.317.0.tar.gz -L
https://github.com/actions/runner/releases/download/v2.317.0/actions-runner-linux-x64-2.317.0.tar.gz
# Optional: Validate the hash
$ echo "9e883d210df8c6028aff475475a457d380353f9d01877d51cc01a17b2a91161d actions-runner-linux-x64-
2.317.0.tar.gz" | shasum -a 256 -c
# Extract the installer
$ tar xzf ./actions-runner-linux-x64-2.317.0.tar.gz
```

En la sección Configure se indican los comandos para configurar y arrancar el runner en nuestro contenedor ubuntu-sonar.

## Configure

```
# Create the runner and start the configuration experience
$ ./config.sh --url https://github.com/josehilera/moviecards --token ADELWNXMU2ATJMRVIKKDQ73GO7J3G
# Last step, run it!
$ ./run.sh
```

**IMPORTANTE:** En la lista de comandos falta uno previo de creación de una variable de entorno, porque si no lo ejecutamos, podemos comprobar que se genera un aviso de que no se puede realizar la configuración:

```
# ./config.sh --url https://github.com/josehilera/moviecards --token
ADELWNXMU2ATJMRVIKKDQ73GO7J3G
Must not run with sudo
```

Por lo que hay que ejecutar primero:

```
# export RUNNER_ALLOW_RUNASROOT="1"
```

Y después ya se puede copiar y pegar el comando en la consola de nuestro contenedor:

```
# ./config.sh --url https://github.com/josehilera/moviecards --token
ADELWNXMU2ATJMRVIKKDQ73GO7J3G
```

```
GitHub Actions
Self-hosted runner registration

# Authentication

✓ Connected to GitHub

# Runner Registration

Enter the name of the runner group to add this runner to: [press Enter for Default]
```

Nos pide un nombre del grupo al que añadir el runner, pulsamos Enter para que asigne uno por defecto.

A continuación, nos pide un nombre para el runner. Podemos escribir cualquiera, por ejemplo “mi-runner”.

```
Enter the name of runner: [press Enter for d8ca80fef8b] mi-runner
```

Y después nos pide una etiqueta (label), pulsamos Enter para quedarnos con las etiquetas por defecto.

```
This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]
```

Finalmente, nos pide una carpeta de trabajo, pulsamos Enter para que la asigne por defecto.

```
Enter name of work folder: [press Enter for _work]
```

Una vez terminado el proceso, el runner se ha creado.



Su estado es "Offline". Para activarlo hay que ejecutar el comando run en el contenedor y queda listo para recibir trabajos que se le envíen desde GitHub.

```
# ./run.sh
```


```
✓ Connected to GitHub
Current runner version: '2.317.0'
2024-06-23 07:09:28Z: Listening for Jobs
```

En el repositorio el estado ahora es Idle.

## Runners

New self-hosted runner

Host your own runners and customize the environment used to run jobs in your GitHub Actions workflows. [Learn more about self-hosted runners.](#)

Runners	Status
 <b>mi-runner</b> self-hosted Linux X64	● Idle ...

No debemos cerrar la consola de ubuntu-sonar porque si no se detendría el runner. Si queremos trabajar con ubuntu-sonar, podemos abrir otra consola con el siguiente comando de docker desde una nueva ventana de comandos de Windows:

```
C:\> docker exec -it ubuntu-ci /bin/bash
```

Si en algún momento apagamos el contenedor, cuando lo volvamos a arrancar hay que acordarse de ejecutar de nuevo el runner que tenemos creado, mediante los comandos:

```
# export RUNNER_ALLOW_RUNASROOT="1"
# ./run.sh
```

### 3.3.2 Definir el trabajo qa

Debemos definir en el archivo main.yaml los comandos del trabajo qa para controlar la calidad del código de la aplicación moviecards<sup>7</sup>.

---

<sup>7</sup> Se podría cambiar `needs:test` por `needs:build` si quisiéramos hacer el control de la calidad del código en paralelo con la realización de las pruebas.

### Archivo C:\moviecards\.github\workflows\main.yaml

```
qa:
  needs: test
  runs-on: self-hosted
  continue-on-error: true
  steps:
    - name: Descargar repositorio
      uses: actions/checkout@v2

    - name: Instalar JDK 11
      uses: actions/setup-java@v2
      with:
        java-version: "11"
        distribution: "adopt"

    - name: Construir con Maven
      run: mvn clean package -DskipTests

    - name: Revisar la calidad con Sonarqube
      run: |
        mvn sonar:sonar -Dsonar.host.url=http://localhost:9000 -
        Dsonar.qualitygate.wait=true -Dsonar.login=admin -Dsonar.password=admin
```

Se ha incluido la sección `continue-on-error: true` para que el workflow no se aborte cuando no tenga éxito el trabajo de control de calidad del código. Esto es habitual, que la calidad de código sea importante y se revise, pero que no bloquee el proceso de entrega de la aplicación, es decir, que se intente mejorar en siguientes versiones, pero que no detenga la integración o el despliegue de la aplicación

Se ha incluido el comando `mvn clean package`, porque Sonarqube exige que los proyectos con Java estén compilados, ya que además del código fuente, también revisa la calidad de los archivos `.class`.

En el comando `mvn sonar:sonar` se han indicado como usuario y password "admin", si se han modificado en SonarQube hay que cambiarlos en el comando. Otra opción es cambiarlo en sonarqube para que la password sea "admin". Esto puede hacerse desde <http://localhost:9000> > Administration > Security > Users > Administrator (Botón rueda de configuración) > Enter a new password.

### Enter a new password

All fields marked with \* are required

Old Password \*

New Password \*

Confirm Password \*

[Change](#) [Cancel](#)

Para que Maven pueda ejecutar el comando `mvn sonar:sonar`, en el archivo `pom.xml` del proyecto debe aparecer una referencia al plugin de sonarqube para Maven. Puede comprobarse que ya está desde que se creó el proyecto, por lo que no hay que añadirlo.

```
<plugin>
  <groupId>org.sonarsource.scanner.maven</groupId>
  <artifactId>sonar-maven-plugin</artifactId>
  <version>3.10.0.2594</version>
</plugin>
```

Una vez guardados los cambios hay que hacer push sobre el repositorio remoto.

```
C:\Moviecards> git add .
C:\Moviecards> git status
C:\Moviecards> git commit -m "Creado trabajo qa"
C:\Moviecards> git push
```

Al ejecutarse el trabajo qa, puede comprobarse que pasa sin problemas.

✓ Creado trabajo qa #14 Re-run all jobs ...

Summary

Jobs

- ✓ build
- ✓ test
- ✓ qa
- ✓ deploy

Run details

main.yaml  
on: push

Triggered via push 30 minutes ago

Status	Total duration	Artifacts
Success	9m 31s	1

Build pipeline: build (33s) → test (2m 42s) → qa (5m 47s)

Puede comprobarse, que el trabajo qa se ha ejecutado en nuestro propio contenedor llamado `mi-runner`.

Summary

Jobs

- build
- test
- qa
- deploy

Run details

- Usage
- Workflow file

```

qa
succeeded now in 5m 47s

Set up job 4s
1 Current runner version: '2.317.0'
2 Runner name: 'mi-runner'
3 Runner group name: 'Default'
4 Machine name: 'd8ca80fef8b'
5 ► GITHUB_TOKEN Permissions
9 Secret source: Actions
10 Prepare workflow directory
11 Prepare all required actions
12 Getting action download info
13 Download action repository 'actions/checkout@v2' (SHA:ee0669bd1cc54295c223e0bb666b733df41de1c5)
14 Download action repository 'actions/setup-java@v2' (SHA:91d3aa4956ec4a53e477c4907347b5e3481be8c9)
15 Complete job name: qa

```

También puede comprobarse que realmente se ha ejecutado en nuestro ordenador, porque en la consola desde la que ejecutamos el runner en el apartado anterior, aparece “Running job: qa” y “Job qa completed with result: Succeeded”.

```

✓ Connected to GitHub

Current runner version: '2.317.0'
2024-06-23 07:09:28Z: Listening for Jobs
2024-06-23 07:34:09Z: Running job: qa
2024-06-23 07:40:03Z: Job qa completed with result: Succeeded

```

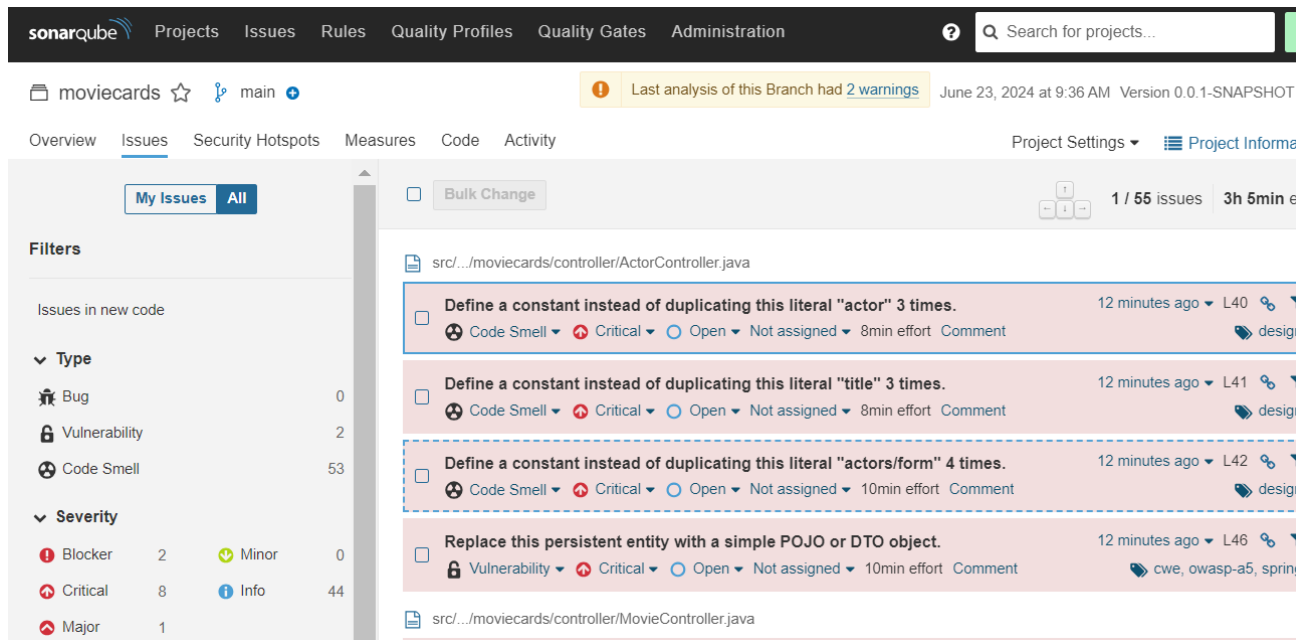
### 3.3.3 Aumentar la exigencia de calidad

Si accedemos al servidor sonarqube <http://localhost:9000> (admin/admin), podemos ver que se ha creado de forma automática un proyecto llamado moviecards y que ha pasado con éxito el control de calidad.

The screenshot shows the SonarQube web interface at `localhost:9000/projects`. The main content area displays a project named `moviecards` with a `Passed` status. The last analysis was performed 10 minutes ago. A summary table provides details for the project:

Bugs	Vulnerabili...	Hotspots Revie...	Code Sm...	Cover...	Duplicati...	Lines
A	D	A	A	0.0%	0.0%	645 XS Java, XML

Aunque en la sección Projects > Moviecards > Issues podemos ver que hay algunos problemas de calidad del código, por ejemplo 8 problemas marcados como críticos, y también 53 problemas de tipo “code smell”, aunque no suficientes para no pasar el control de calidad.



Podemos hacer que sea más exigente, y definir una regla para que no pase la prueba de calidad del código si el número de problemas de calidad críticos es mayor de una cantidad. Esto se hace en la sección:

Quality Gates > Create > Name = “control calidad” > Unlock editing > Add Condition > On Overall code > Quality Gate fails when > Critical issues is greater than 5

**Add Condition**

On New Code  On Overall Code

**Quality Gate fails when**

Critical Issues

**Operator** is greater than **Value** 5

Después hay que entrar en la ventana de la condición “control calidad” y activar el botón “Set as default”, para que se aplique a todos los proyectos.

Quality Gates ⓘ Create

Sonar way BUILT-IN

control calidad DEFAULT

control calidad Rename

Reliability Rating	is worse than	A (No bugs)
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A (No vulnerabilities)

Conditions on Overall Code

Metric	Operator	Value
Critical Issues	is greater than	5 <span>ⓘ</span> <span>✎</span>

Ahora en GitHub, en el workflow accedemos al trabajo “qa” y seleccionamos “Re-run this job” (icono con flechas en círculo) para que se ejecute de nuevo este trabajo.

**qa**  
succeeded 22 minutes ago in 5m 47s

Search logs 🔄 ⚙️

- > ✔️ Set up job 4s
- > ✔️ Descargar repositorio 1s
- > ✔️ Instalar JDK 11 54s
- > ✔️ Construir con Maven 1m 7s
- > ✔️ Revisar la calidad con Sonarqube 3m 10s
- > ✔️ Post Instalar JDK 11 0s
- > ✔️ Post Descargar repositorio 0s
- > ✔️ Complete job 0s

Vemos que aparece un error en el trabajo “qa”, pero que no se ha abortado la ejecución del workflow y ha continuado con el trabajo “deploy”. Esto ocurre porque en main.yaml indicamos `continue-on-error: true` en el trabajo “qa”.

main.yaml  
on: push

```

graph LR
    build[build 33s] --> test[test 2m 42s]
    test --> qa[qa 2m 11s]
    qa --> deploy[deploy]
  
```

En la ventana del trabajo “qa” nos indica que el error es porque ha fallado una QUALITY GATE.

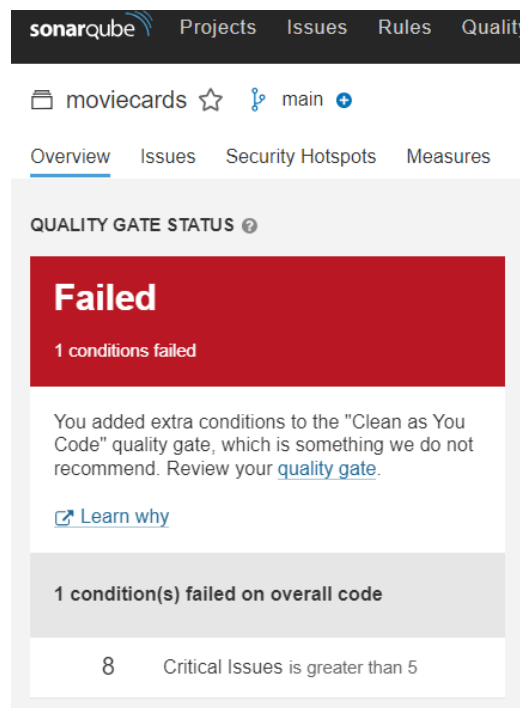
```
qa
failed 1 minute ago in 2m 11s

[INFO] ----- Check Quality Gate status
[INFO] Waiting for the analysis report to be processed (max 300s)
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 01:17 min
[INFO] Finished at: 2024-06-23T08:05:00Z
[INFO] -----
[ERROR] Failed to execute goal org.sonarsource.scanner.maven:sonar-maven-plugin:3.10.0.2594:sonar (default-cli) on project
moviecards: QUALITY GATE STATUS: FAILED - View details on http://localhost:9000/dashboard?id=com.lauracercas%3Amoviecards -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException
Error: Process completed with exit code 1.
```

Y que el detalle del fallo está en:

<http://localhost:9000/dashboard?id=com.lauracercas%3Amoviecards>

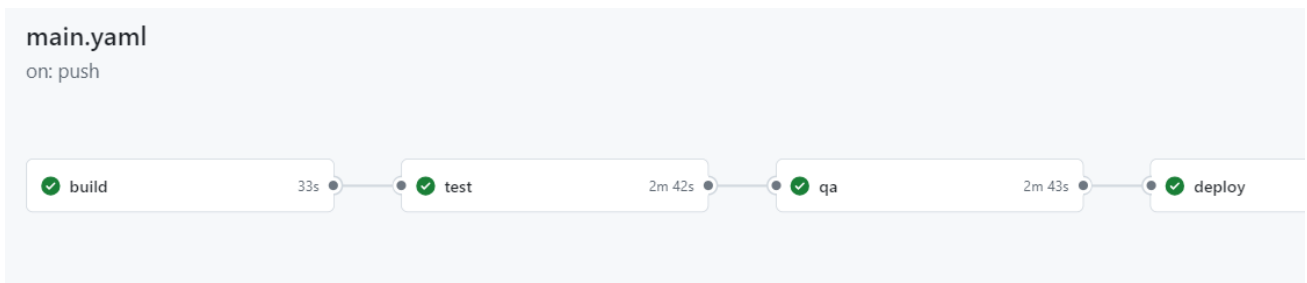
Si accedemos con usuario y contraseña, vemos que en efecto ha habido 8 errores de calidad críticos, que es mayor que el límite de 5 que habíamos definido.



Si en el servidor sonarqube aumentamos el número de fallos de calidad importantes permitidos a 8 en la condición de la Quality Gate que habíamos creado llamada “control calidad”:

control calidad			Rename	Copy
Security Hotspots Reviewed	is less than	100%		
Security Rating	is worse than	A (No vulnerabilities)		
Conditions on Overall Code				
Metric	Operator	Value		
Critical Issues	is greater than	8		

Y refrescamos de nuevo el trabajo “qa” en el workflow, deja de aparecer el aviso de no superación del trabajo porque el nivel de exigencia ahora es menor, ya que permitimos hasta 8 fallos de calidad críticos.



## 3.4 Trabajo deploy

Desplegar una aplicación Java en un hosting en la nube puede hacerse de dos formas:

- Creando en la nube una infraestructura que incluya un servidor Web como Apache Tomcat, y mediante comandos enviar a dicho servidor el archivo .jar que contiene la aplicación.
- Creando en la nube una infraestructura que permita ejecutar contenedores, después dockerizar la aplicación mediante la creación de una imagen que contenga la aplicación ya instalada en un servidor Web, a continuación, publicar dicha imagen en un registro de imágenes (como por ejemplo Docker Hub), y finalmente mediante comandos ejecutar un contenedor en la nube a partir de dicha imagen.

En esta práctica se va a desplegar la aplicación en Azure, donde previamente se habrá creado una infraestructura virtual. Azure permite crear una cuenta gratuita de estudiante y desplegar aplicaciones, aunque con algunas limitaciones respecto a las cuentas de pago.

En el apartado 1.2 creamos una cuenta gratuita en Azure que hay que utilizar en este apartado.

### 3.4.1 Despliegue continuo

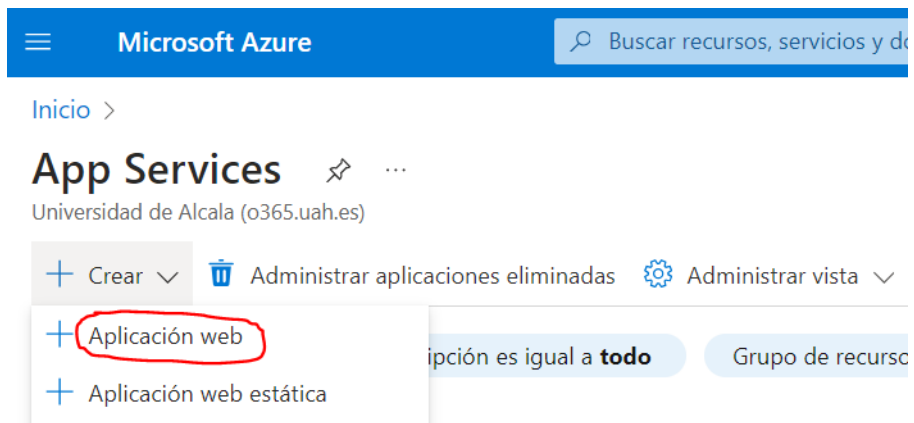
Empezaremos realizando un despliegue en modalidad de “despliegue continuo”, que supone que el despliegue se realizará de forma automática, sin intervención de ninguna persona.

En el siguiente apartado se harán algunos cambios para que el despliegue se realice en modalidad de “entrega continua”, que es lo más habitual, según la cual es necesario que una o varias personas del equipo autoricen el despliegue a producción.

Hay que acceder al portal de Azure: <https://portal.azure.com>, e identificarse con nuestro usuario de la Universidad de Alcalá. Siempre que previamente nos hayamos registrado siguiendo los pasos indicados en el apartado 1.2 de esta práctica.



Seleccionamos la opción “App Services” y después Crear > Aplicación web.



Creamos una aplicación con un nombre que esté disponible, por ejemplo “moviecards” junto a nuestro apellido.



Inicio > App Services >

# Crear aplicación web ...

Datos básicos Base de datos Implementación Redes Supervisión Etiquetas Revisar y crear

App Service Web Apps le permite generar, implementar y escalar rápidamente aplicaciones empresariales web, móviles y de API que se ejecutan en cualquier plataforma. Satisfaga los estrictos requisitos de rendimiento, escalabilidad, seguridad y cumplimiento sin renunciar a una plataforma totalmente administrada para el mantenimiento de la infraestructura. [Más información](#)

## Detalles del proyecto

Seleccione una suscripción para administrar los recursos implementados y los costos. Use los grupos de recursos como carpetas para organizar y administrar todos los recursos.

Suscripción \* ⓘ

Grupo de recursos \* ⓘ  [Crear nuevo](#)

## Detalles de instancia

Nombre \*  .azurewebsites.net

Pruebe un nombre de host predeterminado único. [Más información sobre esta actualización](#)

Publicar \*  Código  Contenedor  Aplicación web estática

Pila del entorno en tiempo de ejecución \*

Pila de servidor web Java \*

Sistema operativo \*  Linux  Windows

Región \*

Elegimos los siguientes parámetros:

- Publicar = Código
- Pruebe un nombre de host predeterminado único = No
- Pila del entorno en tiempo de ejecución = Java 11
- Pila de servidor web Java = Java SE (Embedded Web Server)
- Sistema operativo = Linux
- Region = East US
- Plan de precios = Gratis F1

## Planes de precios

El plan de tarifa de App Service determina la ubicación, las características, los costos y los recursos del proceso asociados a la aplicación. [Más información](#)

Plan de Linux (East US) \* ⓘ

ASP-appmoviecardshileragroup-8216 (F1) ▼

[Crear nuevo](#)

Plan de precios

**Gratis F1** (Infraestructura compartida)

[Revisar y crear](#)

[< Anterior](#)

[Siguiente: Base de datos >](#)

Al seleccionar el botón “Revisar y crear”, aparece una pantalla en la que hay confirmar seleccionando el botón “Crear”. Importante revisar que SKU sea “Gratis”, para que no nos descuente de nuestro crédito disponible.

Al cabo de unos segundos la infraestructura se ha creado y podemos administrar la aplicación, que inicialmente está vacía, hasta que hagamos el despliegue del código desde GitHub Actions.

The screenshot shows the Azure portal interface for a web application named 'moviecards-hilera'. The top navigation bar includes the Microsoft Azure logo and a search bar. Below the navigation bar, there is a search field and a set of action buttons: Examinar, Detener, Intercambiar, Reiniciar, Eliminar, Actualizar, Descargar perfil de publicación, Restablecer perfil de publicación, and Compartir con. The main content area is divided into two columns. The left column contains a sidebar with navigation options: Introducción (selected), Registro de actividad, Control de acceso (IAM), Etiquetas, and Diagnosticar y solucionar problemas. The right column displays the 'Essentials' section with the following details:

Grupo de recursos (mover)	: <a href="#">moviecards-hilera_group</a>	Dominio predeterminado	: <a href="#">moviecards-hilera.azurewebsites.net</a>
Estado	: En ejecución	Plan del servicio de aplicaci...	: ASP-appmoviecardshileragroup-8216 (F1: 1)
Ubicación (mover)	: East US	Sistema operativo	: Linux
Suscripción (mover)	: <a href="#">Azure for Students</a>	Comprobación de estado	: No se pueden capturar los datos de comproba
Id. de suscripción	: 9cd661bb-adf9-4e11-810c-e62caec06180		

A continuación, hay que acceder a la sección de configuración de la aplicación. En esa sección hay que activar la opción “Credenciales de publicación de autenticación básica de SCM”, y seleccionar el botón Guardar.

Microsoft Azure

Inicio > moviecards-hilera

moviecards-hilera | Configuración

preliminar)

Actualizar Guardar Descartar Salir de los comentarios

Haga clic aquí para actualizar a una SKU superior y habilitar características adicionales

**Configuración de plataforma**

Credenciales de publica...  Activado  Desactivado

Credenciales de publica...  Activado  Desactivado

Estado FTP Solo FTPS

Versión HTTP 1.1

Después hay que abrir el centro de implementación de la aplicación en Implementación > Centro de implementación, y en la pantalla que aparece rellenar lo siguientes apartados:

- Origen = GitHub
- Organización = nuestro usuario de GitHub
- Repositorio = moviecards
- Rama = master
- Opción de flujo de trabajo = Agregar un flujo de trabajo

moviecards-hilera | Centro de implementación

Guardar Descartar Examinar Administrar perfil de publicación Sincronizar Salir de los comentarios

Origen \* GitHub

Compilación con Acciones de GitHub. [Cambie el proveedor.](#)

**GitHub**

App Service colocará un flujo de trabajo de Acciones de GitHub en el repositorio elegido para compilar e implementar la aplicación siempre que haya una confirmación en la rama elegida. Si no encuentra ninguna organización o repositorio, es posible que tenga que habilitar permisos adicionales en GitHub. Debe tener acceso de escritura al repositorio de GitHub elegido para implementar con Acciones de GitHub. [Más información](#)

Sesión iniciada como josehilera [Cambiar cuenta](#)

Organización \* josehilera

Repositorio \* moviecards

Rama \* master

Opción de flujo de trab... \*  Agregar un flujo de trabajo: permite agregar un nuevo archivo de flujo de trabajo 'master\_moviecards-hilera.yml'

En la sección Configuración de autenticación hay que elegir Tipo de autenticación = Autenticación básica.

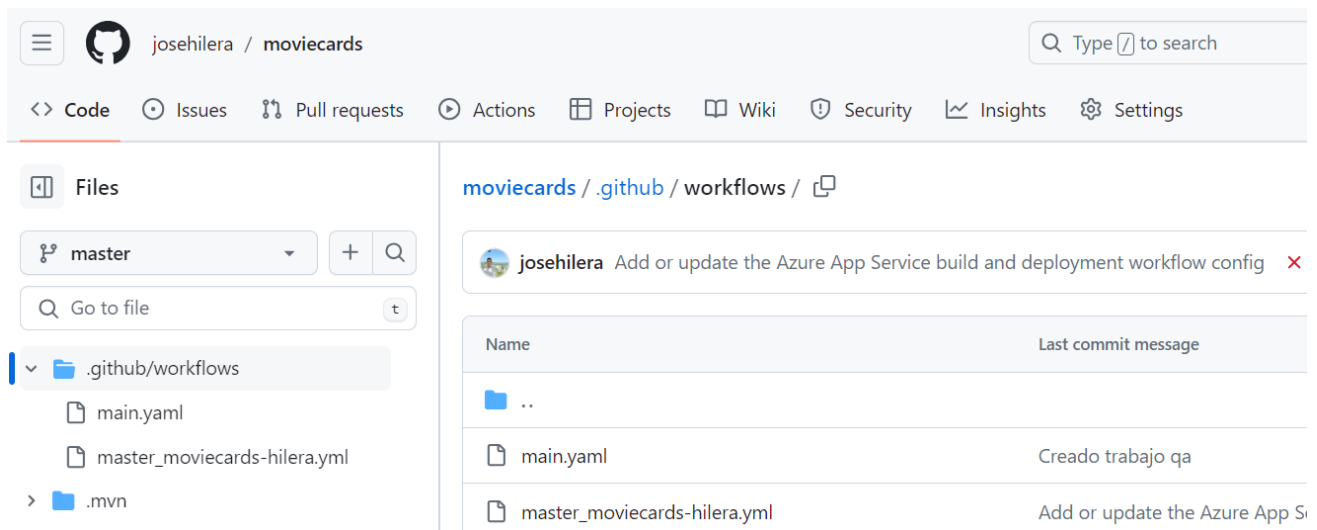
### Configuración de autenticación

Seleccione cómo desea que el flujo de trabajo de acción de GitHub se autentique en Azure. Si elige una identidad asignada por el usuario, la identidad seleccionada se federará con GitHub como cliente autorizado y se le concederán permisos de escritura en la aplicación. [Más información](#)

Tipo de autenticación \*  Identidad asignada por el usuario  
 Autenticación básica

Y seleccionamos el botón guardar.

Ahora hay que acceder al repositorio en GitHub, y veremos que Azure ha creado un nuevo archivo en la carpeta `.github/workflows`.



Name	Last commit message
..	
main.yaml	Creado trabajo qa
master_moviecards-hilera.yaml	Add or update the Azure App S

Para que ese archivo pase a nuestro repositorio local, debemos ejecutar:

```
C:\moviecards>git pull origin master
```

En local abrimos ese nuevo archivo `.yaml` y copiamos el contenido de la sección `deploy` en nuestro archivo `main.yaml`, cambiando donde pone `needs: build`, por `needs: qa`, para que el trabajo `deploy` se ejecute después del trabajo `qa`. También hay que cambiar `name: java-app` por `name: moviecards-java`, para que coincida con el nombre que se le dio al artefacto que se guardó en el trabajo `build`.

Agregamos la condición `if:github.ref=='refs/heads/master'` para que sólo se realice el despliegue cuando haya cambios en la rama `master` del repositorio. Esto se verá con más detalle en el apartado 4 de la práctica.

### Archivo C:\moviecards\.github\workflows\main.yaml

```
deploy:
  runs-on: ubuntu-latest
  needs: qa
  if: github.ref=='refs/heads/master'
  environment:
    name: 'Production'
    url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}

  steps:
    - name: Download artifact from build job
      uses: actions/download-artifact@v3
      with:
        name: moviecards-java

    - name: Deploy to Azure Web App
      id: deploy-to-webapp
      uses: azure/webapps-deploy@v3
      with:
        app-name: 'moviecards-hilera'
        slot-name: 'Production'
        package: '*.jar'
        publish-profile: ${{
secrets.AZUREAPPSERVICE_PUBLISHPROFILE_D6ED3BD417664637ADFE2D648524B336 }}
```





Para desplegar en Azure se utiliza la acción [azure/webapps-deploy@v3](#), creada por Microsoft para facilitar el despliegue desde GitHub Actions<sup>8</sup>:

Cuando se ha creado la aplicación en Azure, se concedió permiso para que Azure pudiera acceder a GitHub y modificar el repositorio. Y lo que ha hecho ha sido crear el archivo .yaml, pero también ha creado un secreto que se pasa como parámetro en la acción webapps-deploy, con las credenciales necesarias para permitir el despliegue en Azure desde GitHub. El uso de un secreto es para ocultar nuestras credenciales, ya que el acceso a nuestro repositorio es público.

Los secretos de los repositorios en GitHub se encuentran en Settings > Security > Secrets and Variables > Actions.

#### Repository secrets

New repository secret

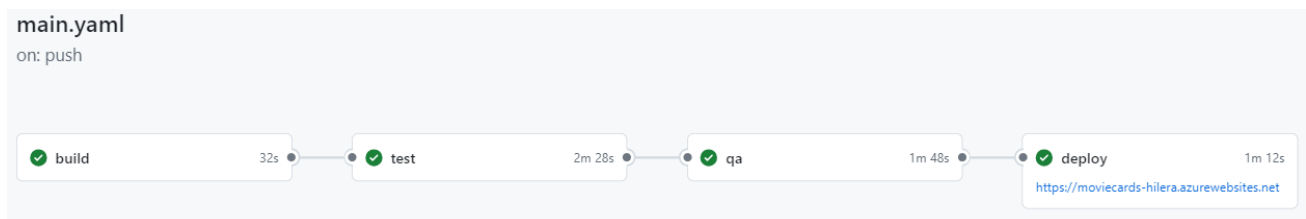
Name 	Last updated
 AZUREAPPSERVICE_PUBLISHPROFILE_D6ED3BD417664637ADFE2D648524B336	5 hours ago  

<sup>8</sup> Ver: <https://learn.microsoft.com/en-us/azure/app-service/deploy-github-actions>

En el archivo main.yaml se puede hacer referencia a los secretos con la expresión `${{secrets....}}`.

Ahora hay que borrar el archivo .yaml creado por Azure, guardar los cambios en el repositorio local y hacer push al remoto, y se comprueba que el trabajo termina correctamente:

```
D:\Moviecards> git add .
D:\Moviecards> git status
D:\Moviecards> git commit -m "Creado trabajo despliegue automático en Azure"
D:\Moviecards> git push
```



Al finalizar el trabajo de despliegue se indica que la aplicación ha quedado desplegada en <https://moviecards-hilera.azurewebsites.net>.

**deploy**  
succeeded 1 hour ago in 28s

Search logs

- > ✓ Set up job
- > ✓ Download artifact from build job
- ✓ **Deploy to Azure Web App**
  - 1 ▶ Run `azure/webapps-deploy@v2`
  - 7 Package deployment using ZIP Deploy initiated.
  - 8 Deploy logs can be viewed at <https://moviecards-hilera.scm.azurewebsites.net/api/deployments/e7ac3ef0-1ca9-41ea-9167-5a6a852e7430/log>
  - 9 Successfully deployed web package to App Service.
  - 10 App Service Application URL: <https://moviecards-hilera.azurewebsites.net>
- > ✓ Complete job

Si accedemos a esa URL, podemos comprobar que, en efecto, la aplicación está desplegada y es pública.

# Gestión de películas

## Inscripción Actor en Pelicula

[Inscripción Actor en Pelicula](#)

### Listado actores

[Listado de actores](#)

### Nuevo Actor

[Crear nuevo actor](#)

### Listado películas

[Listado de películas](#)

### Nueva película

[Crear nueva película](#)

### 3.4.2 Entrega continua

Cuando el despliegue de una aplicación a producción se hace sin intervención de una persona se suele decir que se realiza “despliegue continuo”; mientras que si hay alguna persona que da la orden de despliegue se dice que se realiza “entrega continua”.

Si queremos aplicar la entrega continua, entonces debemos incluir en el archivo main.yaml los comandos necesarios para controlar que una o varias personas autoricen el despliegue.

Esto se puede hacer añadiendo un nuevo paso en el trabajo deploy para la Aprobación manual, usando una acción llamada [trstringer/manual-approval@v1](https://github.com/trstringer/manual-approval@v1).

#### Archivo C:\moviecards\.github\workflows\main.yaml

```
deploy:
  runs-on: ubuntu-latest
```

```

needs: qa
if: github.ref=='refs/heads/master'
environment:
  name: 'Production'
  url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}

steps:
  - name: Aprobación manual
    uses: trstringer/manual-approval@v1
    with:
      secret: ${{ secrets.TOKEN }}
      approvers: josehilera

  - name: Download artifact from build job
    uses: actions/download-artifact@v3
    with:
      name: moviecards-java

  - name: Deploy to Azure Web App
    id: deploy-to-webapp
    uses: azure/webapps-deploy@v2
    with:
      app-name: 'moviecards-hilera'
      slot-name: 'Production'
      package: '*.jar'
      publish-profile: ${{
secrets.AZUREAPPSERVICE_PUBLISHPROFILE_D6ED3BD417664637ADFE2D648524B336 }}

```

La acción manual-approval necesita unas credenciales para acceder al repositorio en GitHub y el nombre de usuario de GitHub de la persona o personas (separados por coma) que deben aprobar el despliegue. Ponemos en este caso nuestro nombre de usuario.

Para establecer las credenciales de acceso a GitHub volvemos a utilizar un secreto al que llamaremos TOKEN, para que no vea su valor en el archivo main.yaml. Primero debemos conseguir su valor, y ello se hace desde la sección “settings” de nuestra cuenta de GitHub, es decir, no desde la página del repositorio, sino desde la página de la cuenta de usuario. No confundir con la sección settings del repositorio, en este caso es la sección de nuestra cuenta como usuario de GitHub.

Accedemos a Settings > Developer Settings > Personal access tokens > Tokens (classic)

The screenshot shows the GitHub Settings interface. At the top, there's a search bar with the text "Type to search". Below it, the navigation sidebar is visible with options: "GitHub Apps", "OAuth Apps", "Personal access tokens" (which is expanded to show "Fine-grained tokens" with a "Beta" badge and "Tokens (classic)"), and "Developer Settings". The main content area is titled "Personal access tokens (classic)" and has two buttons: "Generate new token" and "Revoke all". Below the title, there's a description: "Tokens you have generated that can be used to access the [GitHub API](#)." and a note: "Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#)."

Y seleccionamos el botón “Generate new token (classic)”. En la caja Note le damos el nombre TOKEN.

## New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

### Note

TOKEN

What's this token for?

### Expiration \*

30 days

The token will expire on Wed, Jul 24 2024

### Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input checked="" type="checkbox"/> workflow	Update GitHub Action workflows

Podemos poner un periodo de expiración o elegir que no expire nunca. Lo importante es que en la sección “Select scopes” marquemos sólo la opción “workflow”. Con ello estamos diciendo que quien utilice ese token, lo único que va a poder hacer en nuestra cuenta de GitHub es ejecutar workflows.








Seleccionamos el botón “Generate token”, y copiamos en el portapapeles el valor que aparece.

A continuación, volvemos a nuestro repositorio moviecards y creamos un secreto desde la sección Settings > Security > Secrets and Variables > Actions > New repository secret.

Llamamos a ese secreto TOKEN y pegamos el valor que tenemos guardado en el portapapeles. Ahora ya tenemos dos secretos en el repositorio.

### Repository secrets

New repository secret

Name 	Last updated		
 AZUREAPPSERVICE_PUBLISHPROFILE_D6ED3BD417664637ADFE2D648524B336	yesterday		
 TOKEN	now		

Guardamos los cambios del archivo main.yaml en el repositorio local y se hace push al remoto:

```
D:\Moviecards> git add .
D:\Moviecards> git status
D:\Moviecards> git commit -m "Despliegue con aprobación manual"
D:\Moviecards> git push
```

Lo que ocurre al ejecutarse el trabajo deploy, es que queda pendiente el despliegue hasta la aprobación manual.




Si accedemos a la sección Issues del repositorio moviecards, podemos comprobar que se ha creado un issue avisando que se necesita aprobación manual.

Cuando el usuario que tiene que dar la aprobación acceda al issue deberá escribir un comentario en la sección Write, con alguna de estas palabras: "approved", "approve", "lgtm", "yes".

# Manual approval required for workflow run 9644677033 #1


**Open** josehilera opened this issue 8 minutes ago · 0 comments


 **josehilera** commented 8 minutes ago Owner


Workflow is pending manual review.  
URL: <https://api.github.com/josehilera/moviecards/actions/runs/9644677033>










Required approvers: [josehilera]

Respond "approved", "approve", "lgtm", "yes" to continue workflow or "denied", "deny", "no" to cancel.



 **josehilera** self-assigned this 8 minutes ago


 **Add a comment**

Write Preview H B I         


yes


Al seleccionar el botón Comment, aparece un mensaje de que todos los usuarios que tenían que dar su aprobación lo han hecho, que se cierra el issue y se reanuda el trabajo de despliegue.

**Closed** Manual approval required for workflow run 9644677033 #1  
josehilera opened this issue 9 minutes ago · 2 comments

 **josehilera** commented now

All approvers have approved, continuing workflow and closing this issue.



 **josehilera** closed this as completed now

Podemos comprobar que el trabajo deploy finaliza y queda desplegada la aplicación en Azure.

### main.yaml

on: push



## 4. Realizar una segunda versión de la aplicación

### 4.1 Planificación ágil: crear historias de usuario (issues)

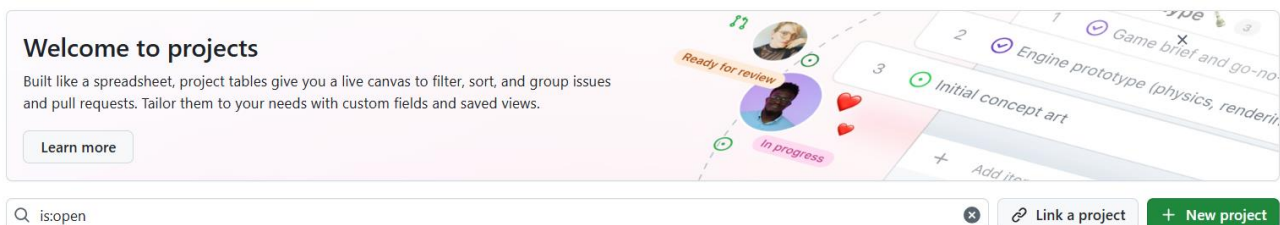
Se puede gestionar un proyecto ágil basado en Scrum con GitHub, teniendo en cuenta que la terminología no coincide exactamente con la que se maneja en Scrum. En esta tabla se muestra la equivalencia:

Agile (Scrum)	GitHub
User story	Issue
Sprint	Milestone
Board	Board
Product backlog	Issue list

En este apartado se va a crear un proyecto con un sprint (milestone) llamado “Aplicación con colores” para la segunda versión de la aplicación, a la que se añadirán colores a la página principal que no había en la primera versión.

#### 4.1.1 Crear un proyecto para el repositorio

Desde la sección Projects del repositorio hay que seleccionar el botón New Project.



Elegimos como template “Board”, para gestionarlo con un tablero Kanban, y llamamos “moviecards” al proyecto.

← Create project

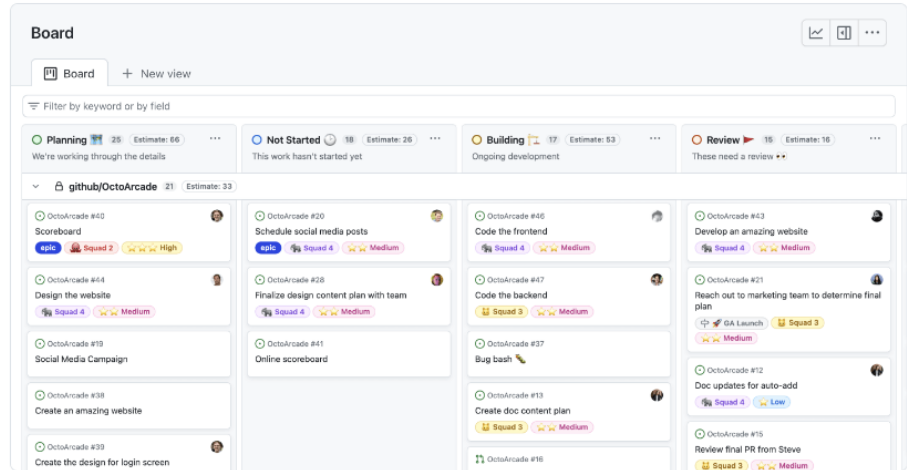
×

### New board

Start with a board to spread your issues and pull requests across customizable columns. Easily switch to a table or roadmap layout at any time.

### Project name

moviecards

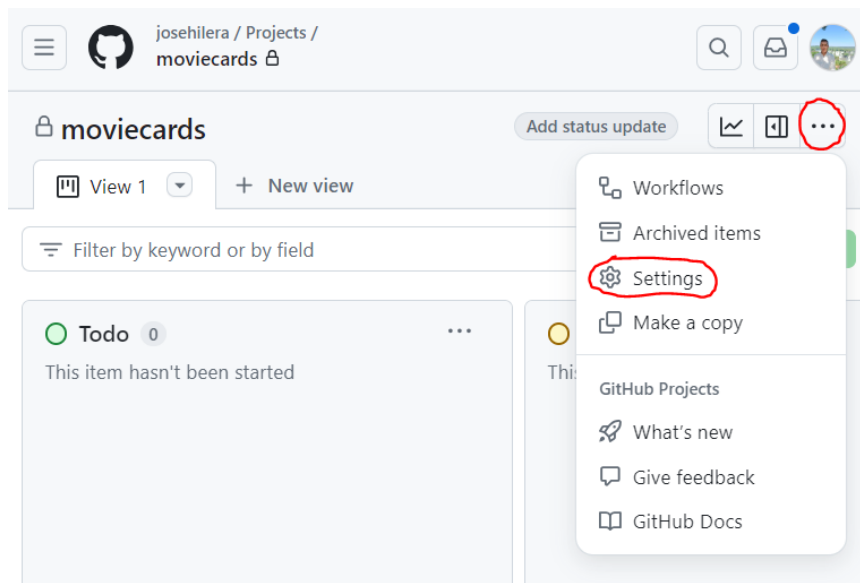


Create project

Al seleccionar el botón “Create project” se crea el proyecto y aparece su tablero Kanban vacío.

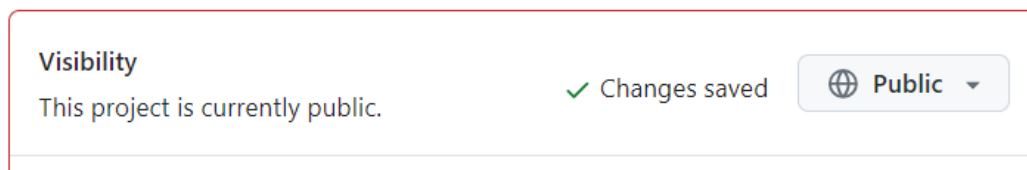


Hay que hacer público el proyecto, desde la sección de propiedades del proyecto o “settings”.



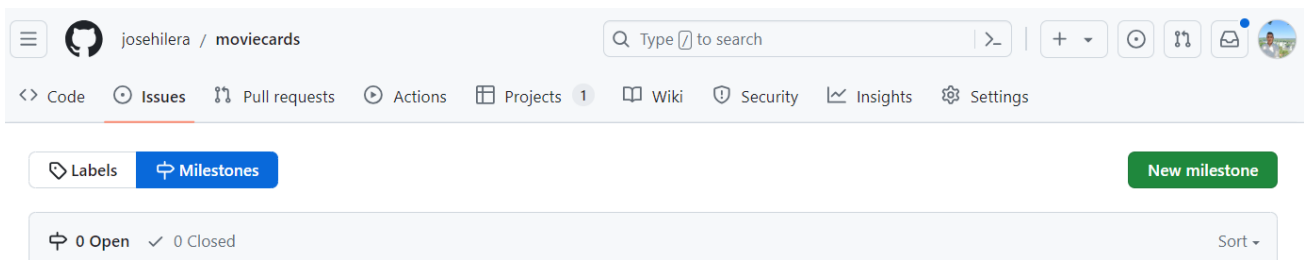
Donde hay que elegir la opción Public en la sección Visibility.

## Danger zone



### 4.1.2 Crear un sprint (milestone)

Para crear un primer sprint o milestone, debemos hacerlo desde el repositorio, accediendo a la sección Issues > Milestones.



Al seleccionar el botón New milestone hay que introducir lo siguiente:

- Title = "Aplicación con colores"
- Description = "Se añadirán colores a la página principal de la aplicación index.html"
- Due date = opcionalmente se puede poner la fecha de finalización prevista

## New milestone

Create a new milestone to help organize your issues and pull requests. Learn more about

### Title

Aplicación con colores

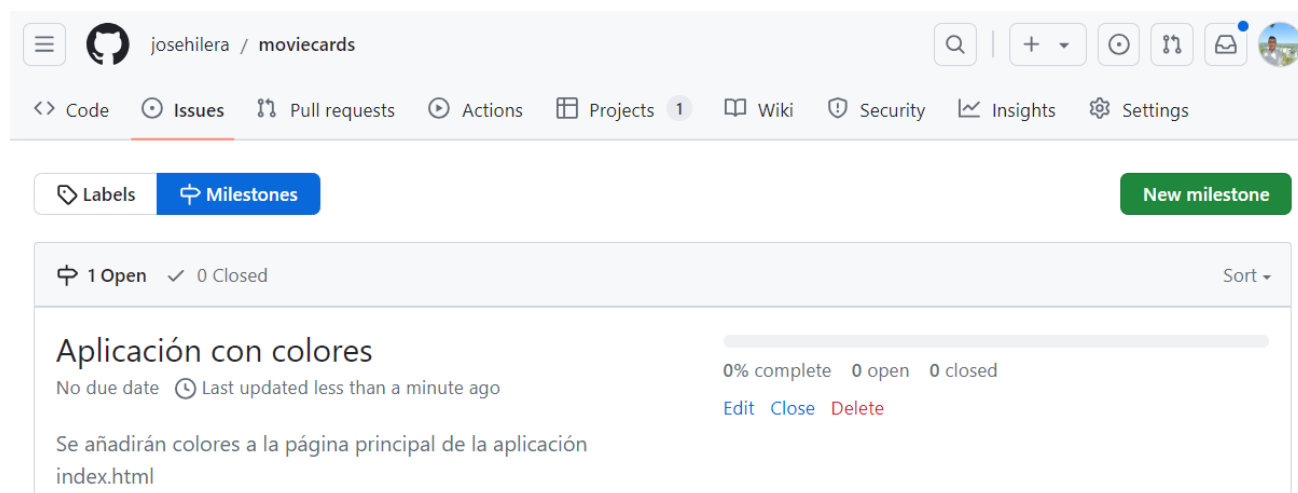
### Due date (optional)

dd/mm/aaaa

### Description

Se añadirán colores a la página principal de la aplicación index.html

Al seleccionar el botón Create milestone, puede comprobarse que ahora aparece en la lista de milestones del repositorio.



The screenshot shows the GitHub repository interface for 'josehilera / moviecards'. The 'Issues' tab is selected, and a 'New milestone' button is visible in the top right. Below the navigation bar, there are tabs for 'Labels' and 'Milestones', with 'Milestones' being the active tab. A 'New milestone' button is also present in the top right of the milestone list. The milestone list shows one open milestone titled 'Aplicación con colores' with a progress bar at 0% complete, 0 open, and 0 closed. The description of the milestone is 'Se añadirán colores a la página principal de la aplicación index.html'.

### 4.1.3 Crear historias de usuario (issues)

Se van a crear 2 historias de usuario (issues).

Los issues son los siguientes:

- **Añadir color de fondo a la página principal:** Se trata de poner color de fondo gris en la página index.html.
- **Añadir color al título de la página principal:** Se trata de poner color de texto rojo y fondo beige en el título de la página index.html.

Ahora hay que crear los dos issues que forman el milestone, para ello se selecciona el botón New issue de la sección Issues del repositorio. Aparece un formulario de creación del issue, y escribimos los siguientes valores:

- Title = Añadir color de fondo a la página principal
- Write = Se trata de poner color de fondo gris en la página index.html.

- Assignees = assign yourself
- Projects = moviecards
- Milestone = Aplicación con colores

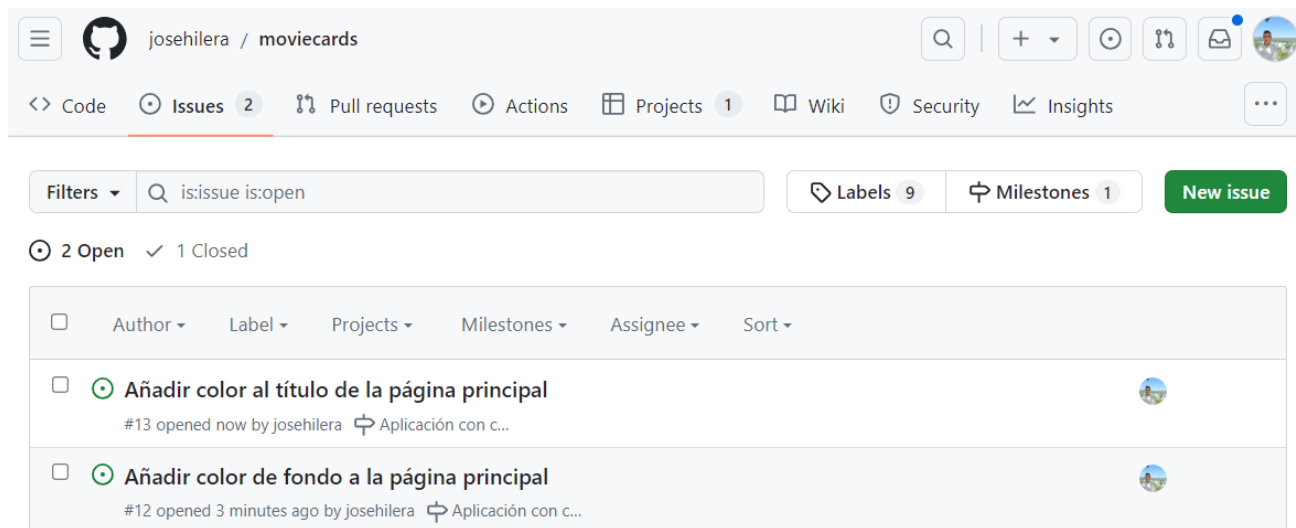
The screenshot shows a GitHub issue page for the repository 'josehilera / moviecards'. The issue title is 'Añadir color de fondo a la página principal'. The description field contains the text: 'Se trata de poner color de fondo gris en la página index.html.' The assignee is 'josehilera'. The project is 'moviecards' and the milestone is 'Aplicación con colores'.

Se procede de la misma forma para crear el otro issue con el título “**Añadir color al título de la página principal**”.

- Title = Añadir color al título de la página principal
- Write = Se trata de poner color de texto rojo y fondo beige en el título de la página index.html.
- Assignees = assign yourself
- Projects = moviecards
- Milestone = Aplicación con colores

The screenshot shows a GitHub issue page for the repository 'josehilera / moviecards'. The issue title is 'Añadir color al título de la página principal'. The description field contains the text: 'Se trata de poner color de texto rojo y fondo beige en el título de la página index.html.' The assignee is 'josehilera'. The project is 'moviecards' and the milestone is 'Aplicación con colores'.

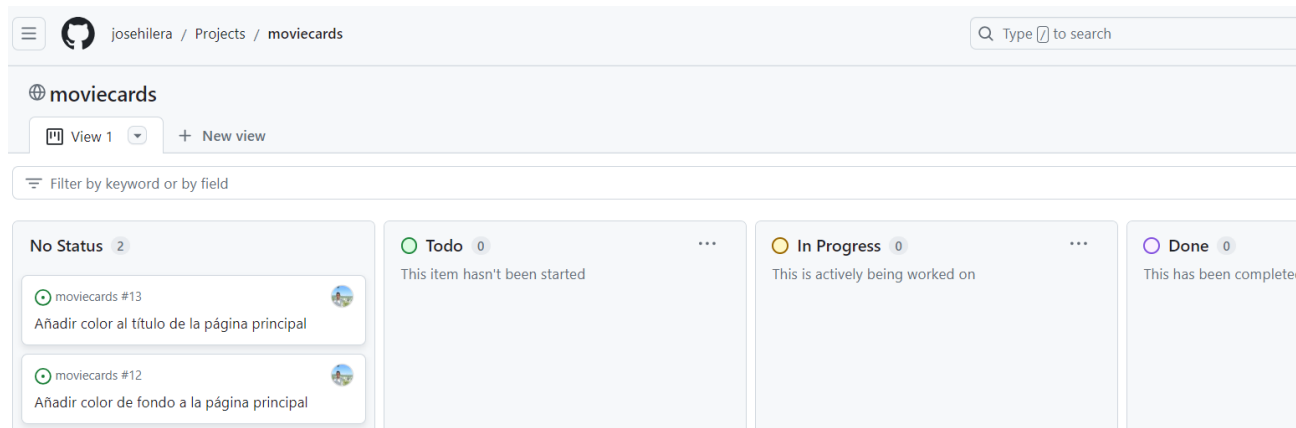
En la sección Issues está la lista de issues del repositorio. Podemos filtrar los issues de un proyecto o de un milestone. Si aplicamos Scrum, la lista del proyecto representaría el product backlog o pila de producto, siendo los issues las historias de usuario. En este caso como todas las historias son del mismo sprint, representaría realmente la pila del sprint.



The screenshot shows the GitHub interface for the repository 'josehilera / moviecards'. The 'Issues' tab is active, showing 2 open issues. The search filter is 'is:issue is:open'. The issues listed are:

- #13: Añadir color al título de la página principal (opened now by josehilera)
- #12: Añadir color de fondo a la página principal (opened 3 minutes ago by josehilera)

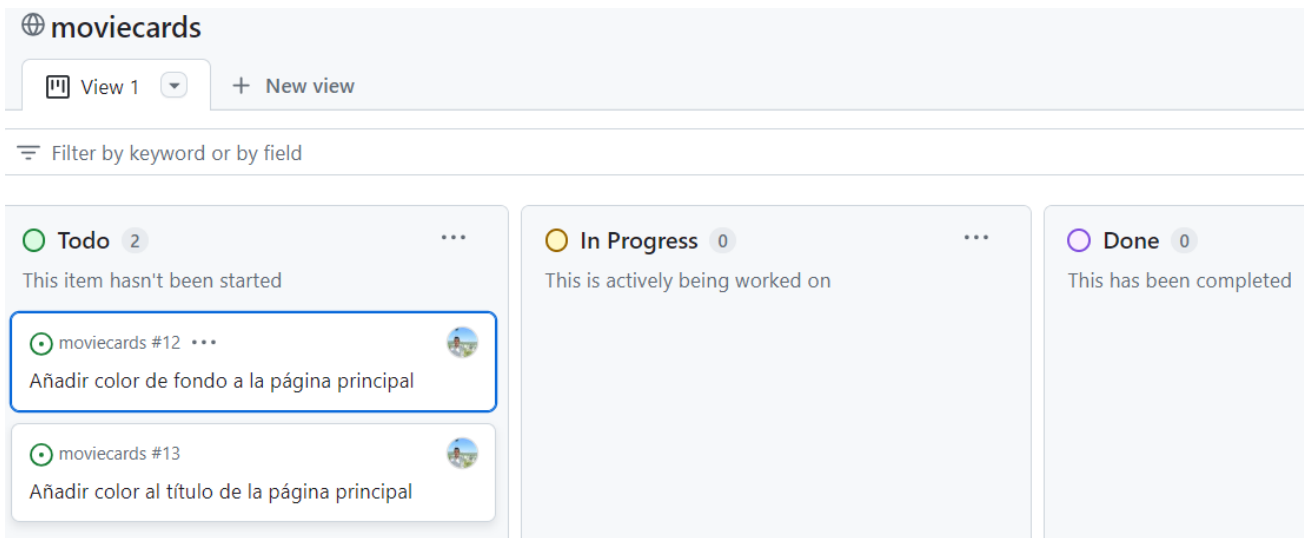
En la sección Projects > moviecards se puede ver el tablero Kaban del proyecto.



The screenshot shows the Kanban board for the 'moviecards' project. The board has four columns: 'No Status' (2 items), 'Todo' (0 items), 'In Progress' (0 items), and 'Done' (0 items). The 'No Status' column contains two issues:

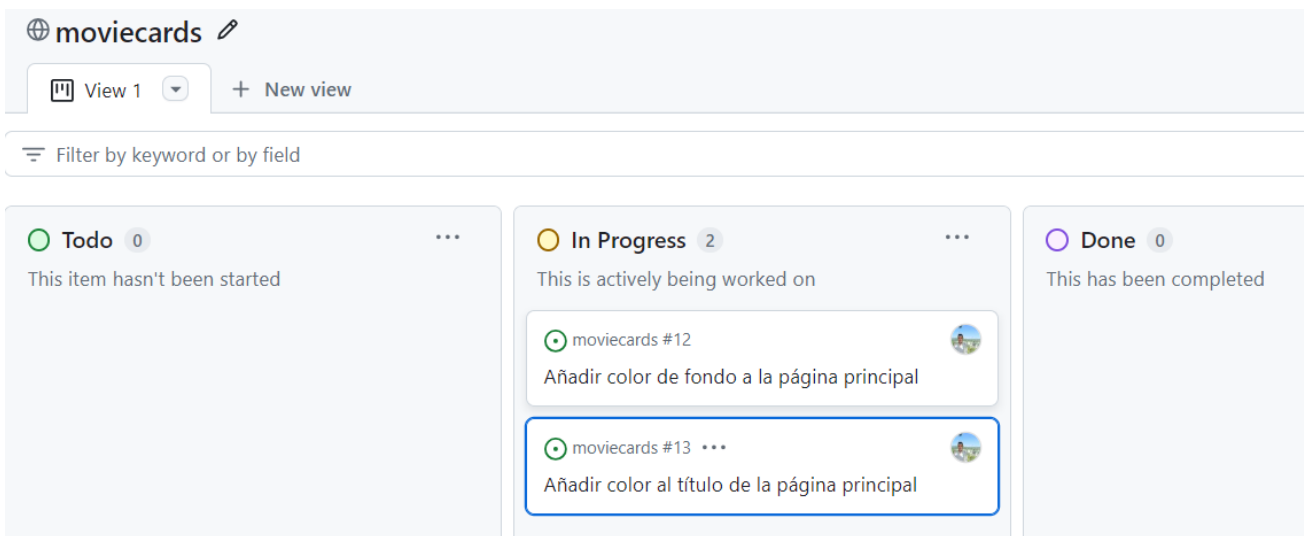
- moviecards #13: Añadir color al título de la página principal
- moviecards #12: Añadir color de fondo a la página principal

Inicialmente las dos historias de usuario o issues están en la columna No Status del tablero. Como se van a implementar los dos issues en el sprint, se pueden arrastrar y soltar en la columna Todo, que puede considerarse la pila del sprint.



Según vaya a comenzar en desarrollo de cada issue, hay que mover la ficha correspondiente a la columna In Progress.

Para simular un conflicto en el desarrollo porque se hagan cambios en el código fuente del proyecto incompatibles entre sí, vamos a suponer que los dos issues comienzan al mismo tiempo en este momento, y los movemos a la columna In Progress.



Vamos a simular ahora cómo trabajarían dos desarrolladores diferentes con el mismo repositorio compartido. Al haber sólo un desarrollador, lo que se hará a continuación puede parecer que no tiene mucho sentido. Si se prefiere, el alumno puede crear otro usuario de GitHub con otro email e invitarle a ser miembro del proyecto desde la sección del proyecto: Settings > Manage Access > Invite collaborators, asignándole el rol Write. Y asignar uno de los issues al nuevo miembro, desde la ficha del issue, sección Assignees.

Si suponemos que solo hay un desarrollador, será quien se encargue de los dos issues, pero los va a desarrollar simultáneamente.

## 4.2 Crear una rama con git para cada issue (feature)

Cada historia de usuario o issue implica añadir una nueva característica o feature al código de la aplicación que se está desarrollando. Existen varias políticas de gestión de ramas. En este proyecto seguiremos la más sencilla, que consiste en crear una rama partiendo de la rama máster para cada nueva feature, y cuando esté terminada, integrar (merge) los cambios de esa rama en la rama máster.

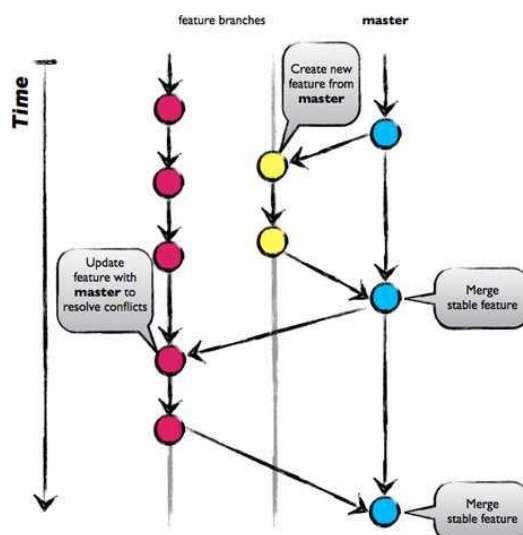


Imagen: [nicoespeon](https://nicoespeon.com)

Hay que trabajar en las dos features de los dos issues, que llamaremos:

- añadir-color-fondo
- añadir-color-titulo

Desde nuestro ordenador local como desarrollador encargado de la primera feature, crearemos una rama para la misma con el comando siguiente:

```
C:\moviecards> git branch añadir-color-fondo
```

Pero como también vamos a desarrollar la otra issue creamos la otra rama:

```
C:\moviecards> git branch añadir-color-titulo
```

Tenemos por tanto tres ramas. Con el comando `git branch` puede comprobarse, indicando con `*` en la que nos encontramos trabajando:

```
C:\moviecards> git branch
  añadir-color-fondo
  añadir-color-titulo
* master
```

Si usamos Visual Studio Code como editor, podemos instalar la extensión Git Graph para ver las ramas: <https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph>. Aunque en las últimas versiones de Visual Studio Code no sería necesario, porque pueden verse desde la sección "Control de código fuente", a la que se llega desde el botón de la izquierda con icono con ramas o con las teclas `Ctrl+Mayús+G`.

### 4.2.1 Programar issue “Añadir color de fondo a la página principal”

Vamos a pasarnos a la primera rama creada para que el desarrollador trabaje en la nueva característica (feature) de añadir color de fondo a la página principal. Para lo cual hay que cambiarse a dicha rama con el comando:

```
C:\moviecards> git checkout añadir-color-fondo  
Switched to branch 'añadir-color-fondo'
```

Para saber en qué rama estamos podemos ejecutar `git branch`, se muestra con `*` la rama actual.

Si trabajamos con Visual Studio Code, en la parte inferior izquierda siempre se indica la rama en la que trabajamos, en este caso estamos en `añadir-color-fondo`:



Estando en esa rama, vamos a modificar el archivo `C:\moviecards\src\main\resources\templates\index.html`.

Donde pone:

```
<body>
```

Añadir un estilo con color de fondo gris:

```
<body style="background-color: lightgray">
```

Con esto terminan los cambios en la rama `añadir-color-fondo` y se deben consolidar en el repositorio local antes de subirlos al repositorio remoto.

```
C:\moviecards> git add .  
C:\moviecards> git status  
C:\moviecards> git commit -m "Añadido color de fondo en página principal"  
C:\moviecards> git push origin añadir-color-fondo
```

En el servidor de integración continua de GitHub se crea de forma automática la rama `añadir-color-fondo` con el código fuente modificado, y se lanza la ejecución para esa rama de los trabajos del workflow `menos deploy`, ya que en el archivo `main.yaml` se indicaba que el trabajo `deploy` sólo se tenía que ejecutar cuando hubiera cambios en la rama `máster`.

Se puede consultar el resultado en la sección `Actions`.

## ✓ Añadido color de fondo en página principal #36

Re-run all jobs



### Summary

Jobs

✓ build

✓ test

✓ qa

🔄 deploy

Re-run triggered 10 minutes ago

josehilera f88ba4d [añadir-color-fondo](#)

Status

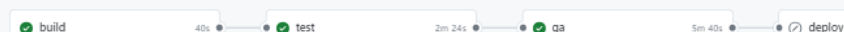
Success

Total duration

6m 8s

### main.yaml

on: push



Una vez pasado el workflow con éxito, se puede proceder a integrar los cambios en la rama máster.

Para ello hay que crear una solicitud de integración (pull request) desde la sección Pull request.

Se trata de hacer una integración desde la rama añadir-color-fondo a la rama master, por lo que antes de crear el nuevo pull request, hay que seleccionar la rama añadir-color-fondo en la sección compare, y dejar el valor master en la sección base.

## Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).



base: master



compare: añadir-color-fondo

✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. [Learn about pull requests](#)


Create pull request

Al seleccionar el botón Create pull request, se abre una pantalla de creación del pull request, donde se puede dejar el nombre por defecto, se puede seleccionar el proyecto y milestone al que pertenece, la persona del proyecto que lo tiene asignado o dejarlo en blanco y si tiene revisores asignados, que se deja en blanco.

## Open a pull request







Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).

base: master ← compare: añadir-color-fondo ✓ **Able to merge.** These branches can be automatically merged.

 **Add a title**

Añadido color de fondo en página principal

**Add a description**

Write Preview H B I  < >      @ ...

Add your description here...

**Reviewers**  
No reviews


**Assignees**  
No one—[assign yourself](#)



**Labels**  
None yet

**Projects**  
moviecards

**Milestone**  
Aplicación con colores

Una vez creado, se puede ver en la lista de pull requests del repositorio.

 1 Open ✓ 4 Closed

<input type="checkbox"/>	Author ▾	Label ▾	Projects ▾	Milestones ▾	Reviewers
<input type="checkbox"/>	 <b>Añadido color de fondo en página principal</b> ✓				
	#14 opened 22 minutes ago by josehilera  Aplicación con c...				

El miembro del equipo que esté designado para aceptar los cambios (por ejemplo, el product owner o quien haya decidido el equipo) puede seleccionar el pull request de la lista y acceder a la sección Files changed para ver los cambios en los archivos del repositorio. En este caso, se comprueba que el archivo index.html tiene un cambio.

# Añadido color de fondo en página principal #14

Edit <> Code

Open josehilera wants to merge 1 commit into master from añadir-color-fondo

Conversation 0 Commits 1 Checks 3 Files changed 1 +1 -1

Changes from all commits File filter Conversations Jump to 0 / 1 files viewed Review changes

```
@@ -10,7 +10,7 @@
10 <title>FichasPelículasApp | Aplicación de gestión de
fichas de películas</title>
11 <link th:href="@{/css/bootstrap.min.css}"
rel="stylesheet">
12 </head>
13 - <body>
14
15 <div class="container">
16 <div th:if="{message != null}" class='alert alert-
success' th:text="{message}" role='alert'></div>
17
18 </div>
19 </body>
20 </html>
```

El responsable puede editar los archivos si decide que alguno de los cambios no es aceptable, o bien aceptarlos y proceder a fusionarlos en la rama master, desde la sección Conversation del pull request, seleccionando el botón Merge pull request.

**Require approval from specific reviewers before merging**  
Rulesets ensure specific people approve pull requests before they're merged. Add rule

**All checks have passed** Hide all checks  
3 successful and 1 skipped checks

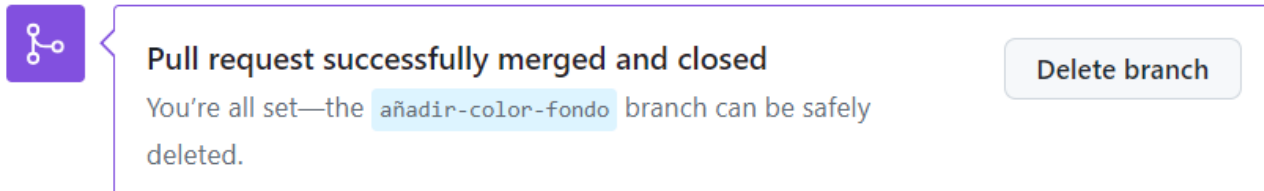
- ✓ CI / build (push) Successful in 40s Details
- ✓ CI / test (push) Successful in 2m Details
- ✓ CI / qa (push) Successful in 5m Details
- ⊘ CI / deploy (push) Skipped Details

**This branch has no conflicts with the base branch**  
Merging can be performed automatically.

**Merge pull request**

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Seleccionando Merge pull request y después Confirm merge, los cambios se integran en la rama master. **No seleccionar el botón “Delete branch”, para que el profesor pueda revisar el contenido de esa rama.**

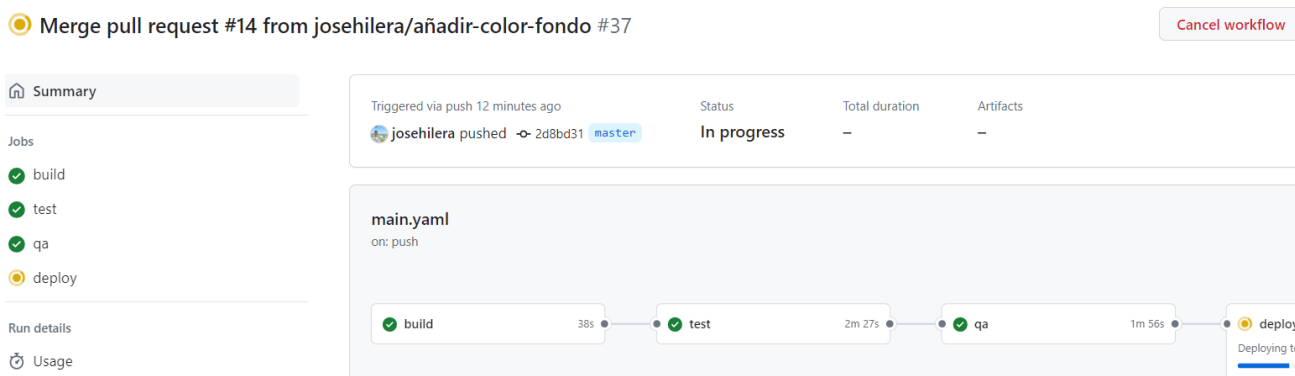


**Pull request successfully merged and closed**

You're all set—the `añadir-color-fondo` branch can be safely deleted.

[Delete branch](#)

Esto origina que se dispare la ejecución de todos los trabajos del workflow definido en main.yaml, pero ahora para la rama máster, por lo que se ejecutan todos los trabajos, incluido el trabajo deploy. Pero como no tiene sentido desplegar todavía a producción, porque no ha terminado el Sprint, no deberíamos aprobar el despliegue, y en su lugar podemos seleccionar el botón “Cancel workflow”.



Merge pull request #14 from josehilera/añadir-color-fondo #37 [Cancel workflow](#)

Summary

Jobs

- build
- test
- qa
- deploy

Run details

Usage

Triggered via push 12 minutes ago

Status: **In progress**

Total duration: --

Artifacts: --

Triggered by: josehilera pushed 2d8bd31 master

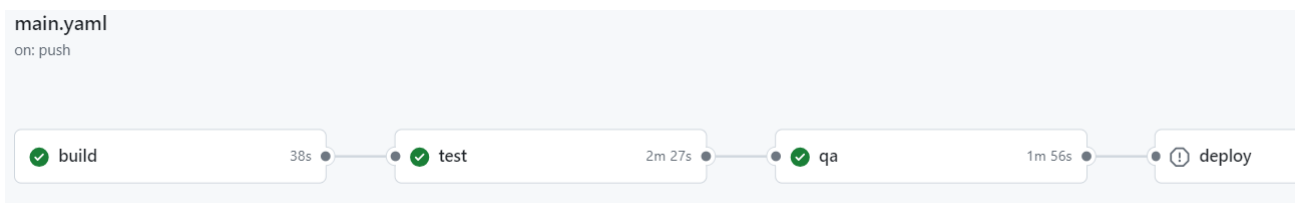
main.yaml

on: push

Workflow steps:

- build (38s) [Completed]
- test (2m 27s) [Completed]
- qa (1m 56s) [Completed]
- deploy (1m 56s) [Cancelled]

Y queda cancelado el trabajo deploy.



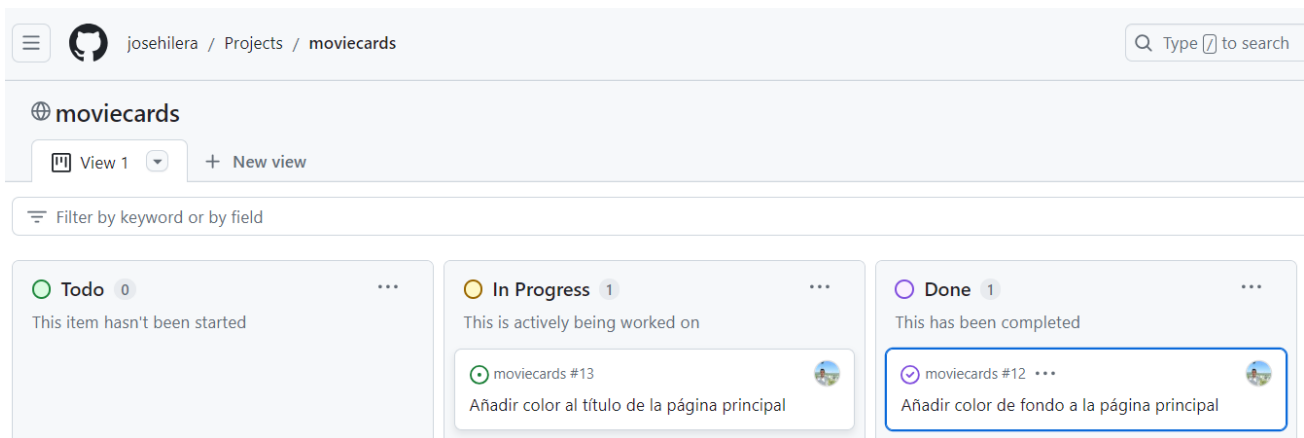
main.yaml

on: push

Workflow steps:

- build (38s) [Completed]
- test (2m 27s) [Completed]
- qa (1m 56s) [Completed]
- deploy (1m 56s) [Cancelled]

Como ha finalizado el issue, en el tablero Kanban hay que arrastrarlo desde la columna In Progress a la columna Done.



En la sección issues > milestones > Aplicación con colores vemos un resumen de lo que ha ocurrido hasta ahora en el sprint (milestone), y se indica que ya hemos completado parte del mismo.

Si nos interesa visualizar un diagrama Burndown, podríamos utilizar alguna herramienta externa que se pudiera conectar a GitHub<sup>9</sup>.

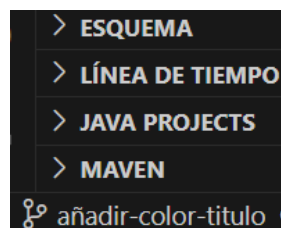
#### 4.2.2 Programar issue “Añadir color al título de la página principal” y resolver conflicto

La otra historia de usuario (issue) para el sprint (milestone) “Aplicación con colores” era la titulada “Añadir color al título de la página principal”, para la cual ya se había creado una rama llamada añadir-color-titulo.

Para trabajar en esa rama, hay que ejecutar el comando:

```
C:\moviecards> git checkout añadir-color-titulo  
Switched to branch 'añadir-color-titulo'
```

Si se trabaja con Visual Studio Code, puede verificarse en la parte inferior izquierda, que estamos en esa rama:



También puede comprobarse que en esa rama el archivo index.html es el original, no está cambiada la etiqueta <body> porque ese cambio se hizo la otra rama (añadir-color-fondo).

<sup>9</sup> Puede encontrarse un ejemplo de herramienta externa en <https://observablehq.com/@tmcw/github-burndown>.

En este caso el desarrollador encargado de poner color al título de la página principal decide hacer dos cambios en el archivo `C:\moviecards\src\main\resources\templates\index.html`.

Donde pone:

```
<body>
. . .
<h1 class="text-center">Gestión de películas</h1>
```

Añadir color de texto rojo en `<h1>` y, por error, color de fondo beige en `<body>`:

```
<body style="background-color: beige">
. . .
<h1 class="text-center" style="color: darkred">Gestión de películas</h1>
```

Es un error porque el color de fondo beige sólo había que ponerlo al título `<h1>`, esto se detectará después al tratar de integrar los cambios en la rama `main`.

Con esto terminan los cambios en la rama `añadir-color-titulo` y se deben consolidar en el repositorio local antes de subirlos al repositorio remoto.

```
C:\moviecards> git add .
C:\moviecards> git status
C:\moviecards> git commit -m "Añadido color en título página principal"
C:\moviecards> git push origin añadir-color-titulo
```

Se ha creado de forma automática la rama `añadir-color-titulo` en el repositorio compartido en GitHub y se ha lanzado para esa rama la ejecución de los trabajos `build`, `test` y `qa` del workflow del proyecto.

🟢 Añadido color en título página principal #38 Re-run all jobs

Summary

Jobs

- ✓ build
- ✓ test
- ✓ qa
- 🕒 deploy

Run details

Triggered via push 18 minutes ago	Status	Total duration	Artifacts
josehilera pushed <code>c40ad1e</code> <code>añadir-color-titulo</code>	Success	5m 23s	1

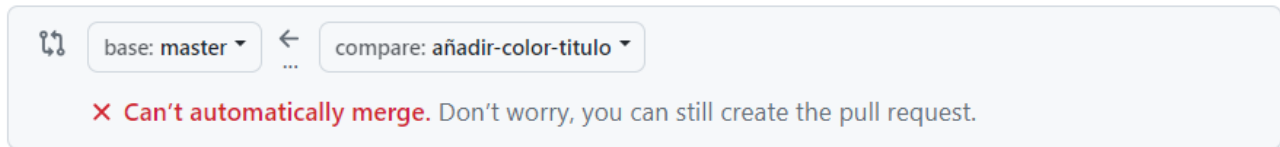
main.yaml  
on: push

✓ build 31s → ✓ test 2m 40s → ✓ qa 1m 40s → 🕒 deploy 0s

Ahora es el momento de solicitar su integración en la rama `main` del proyecto, desde Pull requests > botón `New pull request`, como se hizo en el apartado anterior, pero ahora seleccionando la rama `añadir-color-titulo` en la sección `compare`.

## Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).

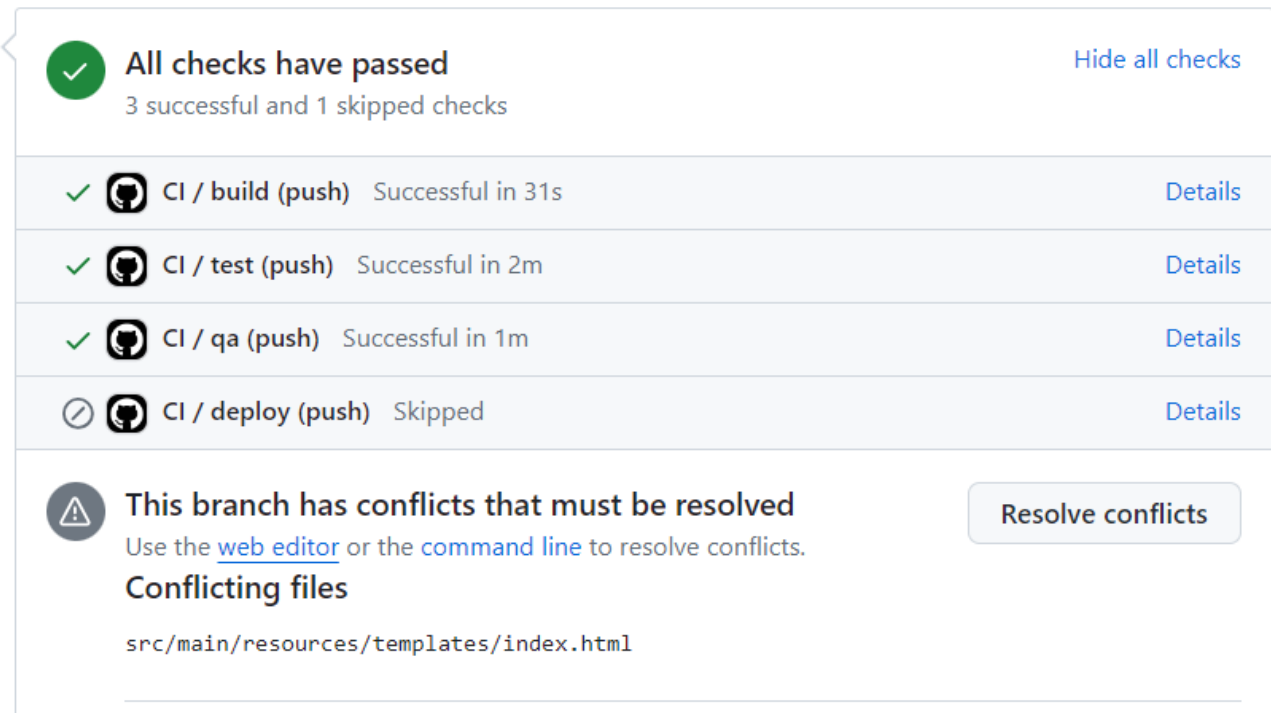


base: master ← compare: añadir-color-titulo

✗ **Can't automatically merge.** Don't worry, you can still create the pull request.

Comprobamos que se avisa de que no es posible hacer la integración de forma automática, pero que podemos crear el pull request, por lo que seleccionamos el botón Create pull request, indicando previamente que está asociado al proyecto moviecards, y milestone “Aplicación con colores”.

En la pantalla del nuevo pull request se explica el conflicto.



✓ **All checks have passed** [Hide all checks](#)  
3 successful and 1 skipped checks

✓	CI / build (push)	Successful in 31s	<a href="#">Details</a>
✓	CI / test (push)	Successful in 2m	<a href="#">Details</a>
✓	CI / qa (push)	Successful in 1m	<a href="#">Details</a>
⌛	CI / deploy (push)	Skipped	<a href="#">Details</a>

⚠ **This branch has conflicts that must be resolved** [Resolve conflicts](#)  
Use the [web editor](#) or the [command line](#) to resolve conflicts.

**Conflicting files**


src/main/resources/templates/index.html

Lo que ha ocurrido es que el desarrollador encargado de este segundo issue ha modificado el archivo `index.html` cambiado en `<body>` el estilo del color de fondo `style="background-color: beige"`. Pero el desarrollador del issue del apartado anterior también lo cambió con un valor diferente `style="background-color: lightgray"`. Este primer cambio ya está integrado en la rama master, pues se hizo en el apartado anterior.

Se puede ver el detalle del conflicto pulsando el botón [Resolve conflicts](#).

# Añadido color en título página principal #16

Resolving conflicts between `añadir-color-titulo` and `master` and committing changes → `añadir-color-titulo`

1 conflicting file	src/main/resources/templates/index.html	1 conflict	Prev
 index.html src/main/resources/templates/i ndex.html	13 <<<<<< añadir-color-titulo 14 <body style="background-color: beige"> 15  ===== 16 <body style="background-color: lightgray"> 17 >>>>>> master 18		

Los conflictos se pueden resolver directamente en GitHub, modificando el archivo que nos muestra con el conflicto, pero en general no es lo más conveniente. Es recomendable que los desarrolladores tengan en su ordenador local el último estado del repositorio compartido antes de enviar sus cambios. Para ello antes de hacer un push, hay que actualizar nuestro repositorio local usando el comando pull:

```
C:\moviecards> git pull origin master
```

```
C:\moviecards>git pull origin master
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 927 bytes | 25.00 KiB/s, done.
From https://github.com/josehilera/moviecards
* branch          master      -> FETCH_HEAD
  32f8945..2d8bd31 master    -> origin/master
Auto-merging src/main/resources/templates/index.html
CONFLICT (content): Merge conflict in src/main/resources/templates/index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Nos avisa de que ha habido problemas. Si abrimos el archivo problemático, podemos ver los conflictos y borrar manualmente la sección que decidamos que hay que cambiar.

En el caso del archivo `index.html`, si lo abrimos con Visual Studio Code, nos ofrece aceptar el cambio.

```

6 <!DOCTYPE html>
7 <html xmlns:th="http://www.thymeleaf.org" lang="es">
8 <head>
9   <meta charset="UTF-8">
10  <title>FichasPelículasApp | Aplicación de gestión de fichas de películas</title>
11  <link th:href="@{/css/bootstrap.min.css}" rel="stylesheet">
12 </head>
13 <<<<<< HEAD (Cambio actual)
14 <body style="background-color: beige">
15 =====
16 <body style="background-color: lightgray">
17 >>>>>> 2d8bd3151f121df31be282886ca521124d8a9e30 (Cambio entrante)

```

Realmente el desarrollador del segundo issue ha cometido un error, ya que el color de fondo beige era sólo para el título <h1> de la página. Se puede resolver dejando el estilo de fondo gris para <body> que puso el desarrollador del primer issue, y poniendo el estilo de fondo beige en el título <h1>.

Lo que haremos será seleccionar “Aceptar cambio entrante”, quedando el valor gris.

```

6 <!DOCTYPE html>
7 <html xmlns:th="http://www.thymeleaf.org" lang="es">
8 <head>
9   <meta charset="UTF-8">
10  <title>FichasPelículasApp | Aplicación de gestión de fichas de películas</title>
11  <link th:href="@{/css/bootstrap.min.css}" rel="stylesheet">
12 </head>
13 <body style="background-color: lightgray">
14

```

Y hay que añadir manualmente en <h1> un color de fondo beige.

```
<h1 class="text-center" style="color: darkred; background-color: beige">Gestión de películas</h1>
```

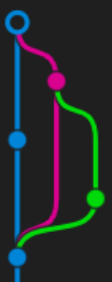




Estos cambios se deben enviar al repositorio compartido.

```


C:\moviecards> git add .
C:\moviecards> git status
C:\moviecards> git commit -m "Corregido conflicto colores"
C:\moviecards> git push origin añadir-color-titulo

```


Si hemos instalado en Visual Studio Code la extensión Git Graph, podemos ver gráficamente cómo han evolucionado las ramas del proyecto. La palabra “origin” se refiere al repositorio remoto.









Graph	Description
	<ul style="list-style-type: none"> <li> <b>añadir-color-titulo</b> <i>origin</i> <b>Corregido conflicto colores</b></li> <li> <b>origin/master</b> Merge pull request #14 from josehilera/añadir-color-fondo Añadido color en título página principal</li> <li> <b>añadir-color-fondo</b> <i>origin</i> Añadido color de fondo en página principal</li> <li> <b>master</b> Despliegue con aprobación manual</li> </ul>


Si volvemos a GitHub y refrescamos la pantalla del pull request, cuando se termina de ejecutar el workflow correspondiente, ya aparece el botón “Merge pull request” habilitado.


 **Require approval from specific reviewers before merging**  
[Rulesets](#) ensure specific people approve pull requests before they're merged.

---


 **All checks have passed**  
 3 successful and 1 skipped checks

-   **CI / build (push)** Successful in 35s
-   **CI / test (push)** Successful in 2m
-   **CI / qa (push)** Successful in 2m
-   **CI / deploy (push)** Skipped

 **This branch has no conflicts with the base branch**  
 Merging can be performed automatically.

**Merge pull request**  You can also [open this in GitHub Desktop](#) or view [com](#)

Si seleccionamos el botón Merge pull request > Confirm merge, se producirá la integración en la rama máster y se lanzará de nuevo el workflow para la rama máster. **No seleccionar el botón “Delete branch”, para que el profesor pueda revisar el contenido de esa rama.**


 **Pull request successfully merged and closed** **Delete branch**

You're all set—the `añadir-color-titulo` branch can be safely deleted.

En este caso se ejecutan todos los trabajos del workflow, incluido el de despliegue, que tiene la condición de aprobación manual, por lo que vamos a la sección Issues y en el nuevo issue que se ha creado de forma automática para la aprobación, escribimos “yes” en la caja Write y seleccionamos el botón Comment.

## Manual approval required for workflow run 9675159267 #1


Open josehilera opened this issue 1 minute ago · 0 comments


 josehilera commented 1 minute ago Owner ⋮


Workflow is pending manual review.  
URL: <https://api.github.com/josehilera/moviecards/actions/runs/9675159267>

Required approvers: [josehilera]

Respond "approved", "approve", "lgtm", "yes" to continue workflow or "denied", "deny", "no" to cancel.



 josehilera self-assigned this 1 minute ago

 Add a comment

Write Preview H B I ☰ <> 🔗 ☰ ☰ ☰ 📎 @ 🔄 ↩ 🗑

yes

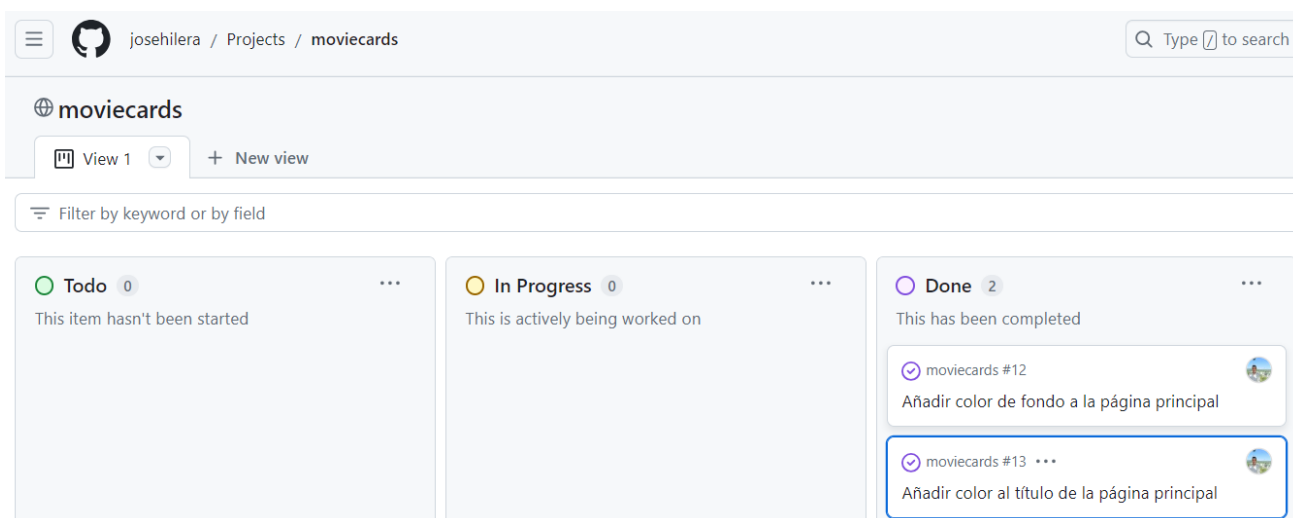
Cuando termina el despliegue, aparece en verde indicando que ha finalizado con éxito.



La aplicación modificada se ha desplegado en Azure. Y podemos comprobar que los colores de la página principal son correctos: fondo gris y título rojo con fondo beige.



Con esto hemos terminado el issue “Añadir color al título de la página principal”, por lo que debemos indicarlo en el tablero del proyecto, en Projects > moviecards, arrastrando la ficha del issue de la columna In Progress a la columna Done. Entramos al detalle del issue y vemos que aparece como Cerrado (“Closed”).



Si accedemos a la lista de sprints (milestones) en Issues > Milestones, vemos que el sprint “Aplicación con colores” está completado al 100%, pero está abierto todavía.

Navigation bar: [Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) **1** [Wiki](#) [Security](#) [Insights](#) [Settings](#)

Labels **Milestones**

📌 1 Open ✓ 0 Closed

### Aplicación con colores

No due date ⌚ Last updated 1 minute ago

Se añadirán colores a la página principal de la aplicación index.html

100% complete 0 open 4 closed

[Edit](#) [Close](#) [Delete](#)

Por lo que podemos entrar al milestone y cerrarlo seleccionando el enlace “Close”, y pasa a la sección de milestones cerrados.

Labels **Milestones**

📌 0 Open ✓ 1 Closed

### Aplicación con colores

Closed now ⌚ Last updated less than a minute ago

Se añadirán colores a la página principal de la aplicación index.html

100% complete 0 open 4 closed

[Edit](#) [Reopen](#) [Delete](#)