Trabajo Práctico 8

Materia: Ingeniería de Software 3

Alumno: Vietto Santiago

Docente: Fernando Bono

Institución: UCC

Año: 2022

Ejercicio 1:

Integración continua (IC): esta permite a los desarrolladores de software evitar una larga y problemática fase de integración al final de un proyecto. En lugar de compilar todos los componentes al final, con la IC se van implementando todas las novedades directamente en el código base. Esto requiere disciplina y un proceso eficiente, ya que de lo contrario la IC obstaculizará más de lo que ayudará. El proceso se puede facilitar además con software específico. A veces de forma totalmente autónoma y otras veces en combinación con otras aplicaciones, las herramientas de integración continua (CI tools) ayudan en la creación de un repositorio, en la ejecución de las pruebas y en la compilación, así como en el control de versiones y, por supuesto, en la propia integración continua. En la actualidad, internet ofrece una gran variedad de herramientas para la integración continua, en donde todas tienen como objetivo ayudar al desarrollador en la implementación de esta metodología, y lo hacen de diferentes modos y con la ayuda de características distintas. Pero estas herramientas no solo se diferencian unas de otras en cuanto a sus características, sino que también existe una gran variedad en lo que respecta a precios y licencias. Mientras que muchas de ellas son de código abierto y se encuentran disponibles de forma gratuita, otros fabricantes ofrecen herramientas comerciales.

_ A continuación, analizamos las siguientes herramientas más utilizadas y examinamos sus características y funciones:

- Travis CI: esta herramienta de integración continua trabaja en estrecha relación con el popular software de control de versiones GitHub. Esta herramienta puede configurarse con un sencillo archivo YAML que se guarda en el directorio raíz del proyecto. GitHub informa a Travis CI de todos los cambios efectuados en el repositorio y mantiene el proyecto actualizado. Este esta programado en Ruby, es multiplataforma, como dijimos antes funciona con GitHub, se configura con un archivo YAML, es gratuito para proyectos de código abierto, el precio para proyectos comerciales es entre 69 y 489 dólares/mes, y es de código abierto (licencia MIT).

- Circle CI: esta herramienta funciona tanto con GitHub como con Bitbucket. En las fases de prueba, pueden emplearse tanto contenedores como máquinas virtuales. CircleCI confiere mucha importancia a la ejecución de procesos de desarrollo sin interferencias, por lo que arroja de forma automática builds compatibles con otros entornos. Su configuración es a través de un archivo YAML, soporta también el despliegue continuo, posee alojamiento propio o en la nube, se ejecuta en contenedores Docker, máquinas virtuales Linux y MacOS, es gratuito para un contenedor, pero de otro modo, vale entre 50 y 3 150 dólares al mes.

- Codefresh: esta herramienta es una plataforma de CI/CD cloud nativa desde su concepción específicamente pensada para Kubernetes. Ofrece la posibilidad de ejecutarse en ambientes on-premise o cloud, y SaaS. No es una plataforma tan popular como Gitlab y Jenkins pero sin duda que merece un lugar en la lista. Entre las características que tiene, se destaca una biblioteca extensa de plugins, un registro de imágenes de containers, repositorios de Charts de Helm, despliegue nativo en containers, VMs, Serverless y más, posee una sintaxis basada en YAML, posee un modelo de enfoque basado en GitOps, permite integraciones con servicios

externos. Posee una edición comunitaria gratis para hasta 5 desarrolladores y hasta un tiempo de ejecución de Argo, y además posee una versión paga de $49/mes por desarrollador permitiendo acceder a más beneficios.

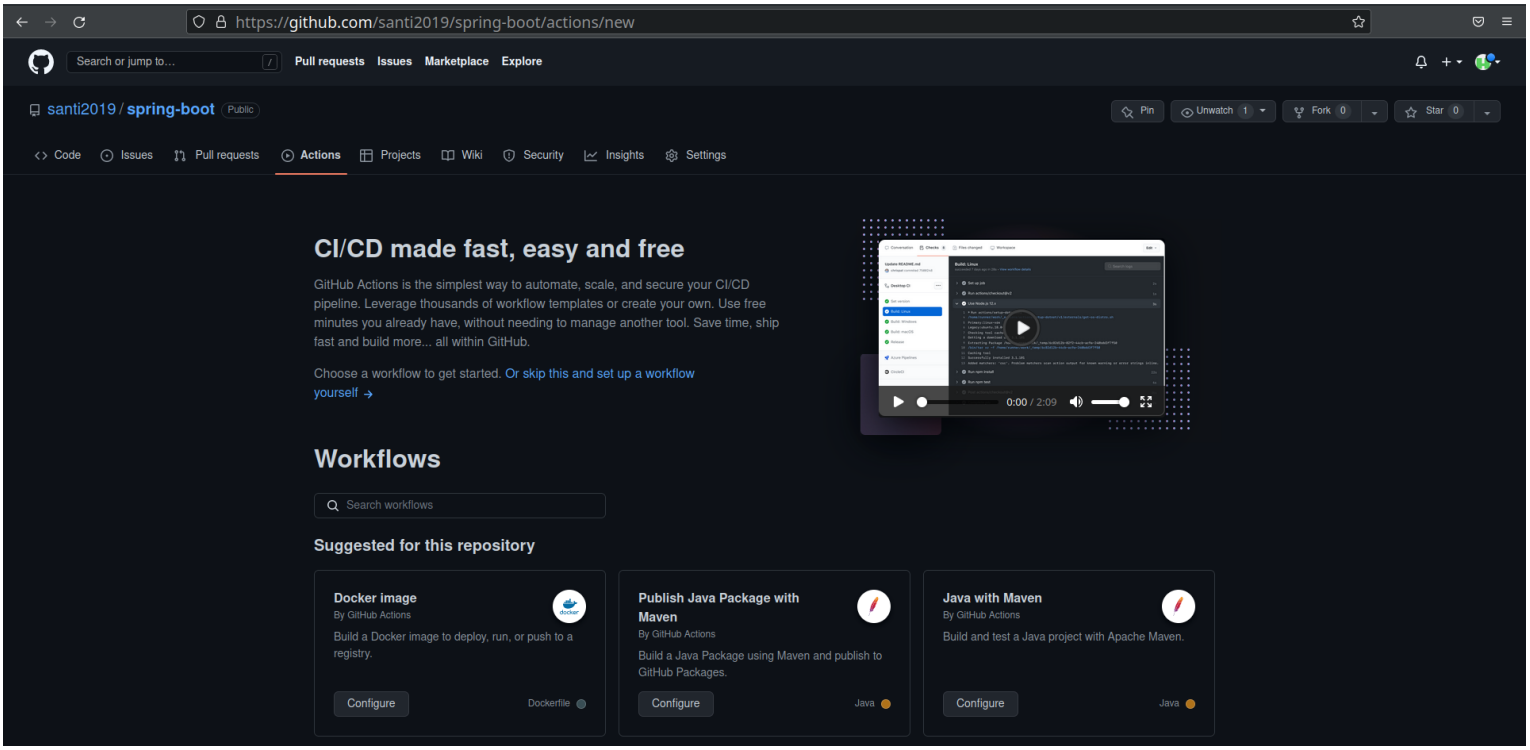
- GitHub Actions: se define como una plataforma utilizada para la integración continua (CI) y la entrega continua (CD), que permite la automatización de procesos de compilación, implementación y prueba. De la misma manera, esta herramienta destaca por contribuir en el desarrollo y creación del código de forma directa a través de la plataforma. GitHub Actions permite, además, el inicio de workflows como el desarrollo de issues, la creación de nuevas versiones y demás, al tiempo que se combinan y ajustan acciones para los servicios que se usan, crean, actualizan y mantienen la comunidad de la plataforma. Actions utiliza paquetes de códigos en los contenedores de Docker, los cuales se ejecutan en los servidores de GitHub y que, a su vez, son compatibles con cualquier lenguaje de programación. Esto hace que puedan funcionar con servidores locales y nubes públicas. Se definen mediante archivos YAML y con él se puede hacer el build, test, package, reléase o deploy de un proyecto. Esta funcionalidad es gratis para todos los repositorios de código abierto e incluye 2000 minutos al mes de compilación sin coste para los repositorios privados. Si, por el contrario, esto no es suficiente para nuestras necesidades, se puede elegir otro plan de forma sencilla.

- Gitlab: es una solución web de software libre de forja, control de versiones e integración continua que ayuda al desarrollo de software colaborativo basada en Git y que además también tiene algunas funcionalidades de gestión de proyectos integrada. GitLab CI forma parte de GitLab. Además de integración continua, GitLab ofrece despliegue y entrega continua. Al igual que con Travis CI, la configuración de GitLab CI se lleva a cabo con un archivo YAML, por lo demás, su utilización es sencilla, está programado en Ruby y Go, asiste también en la entrega y el despliegue continuo, es Open Core, posee alojamiento propio o en la nube, la prueba gratuita de GitLab está disponible durante 30 días, y además tiene cuatro planes de precios para la solución SaaS, es decir, Gratis, Bronze ($ 4 por usuario por mes), Silver ($ 19 por usuario por mes) y Gold ($ 99 por usuario por mes).

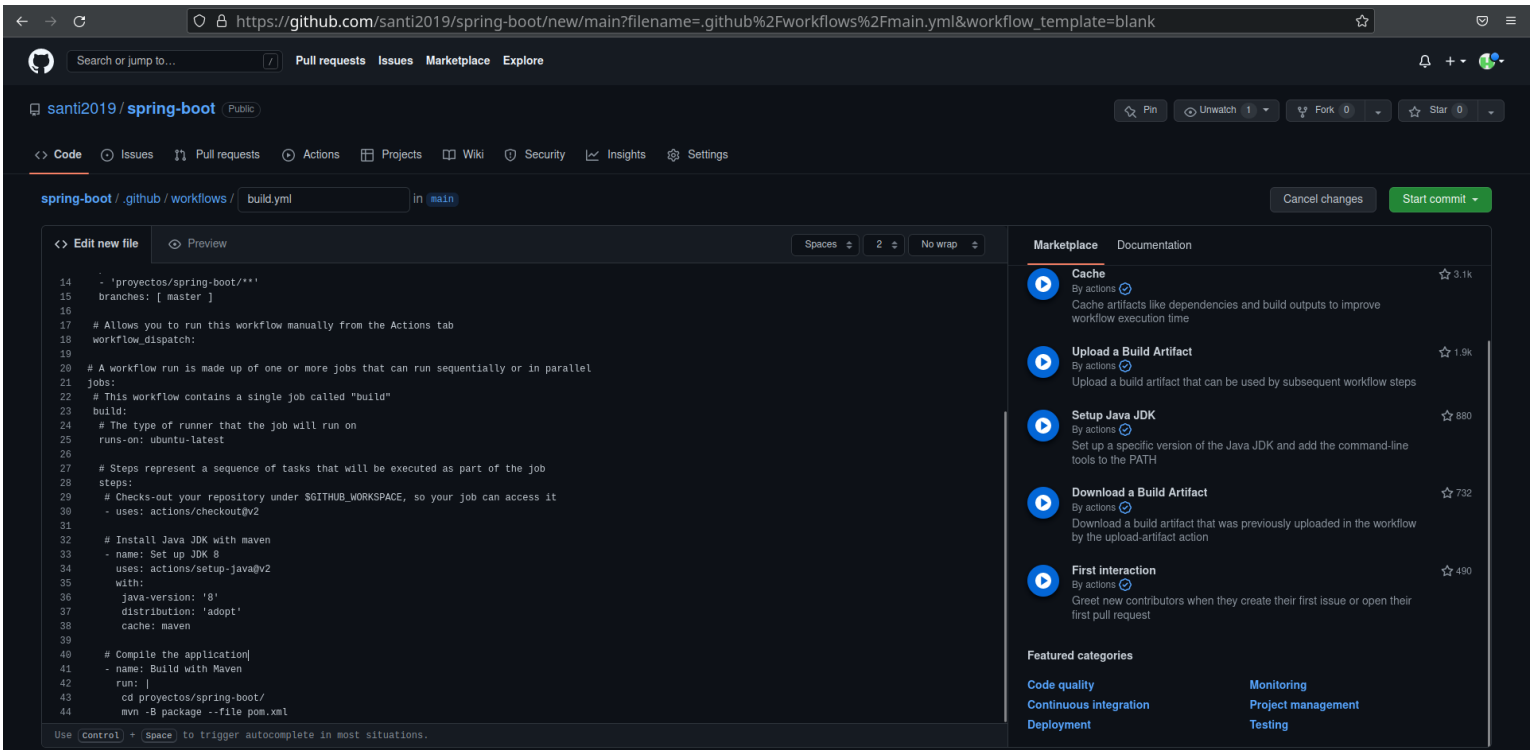
- Jenkins: explicado en el trabajo anterior.

Ejercicio 2:

1)_ En GitHub, en el repositorio donde se encuentra la aplicación spring-boot, accedemos a la opción Actions:



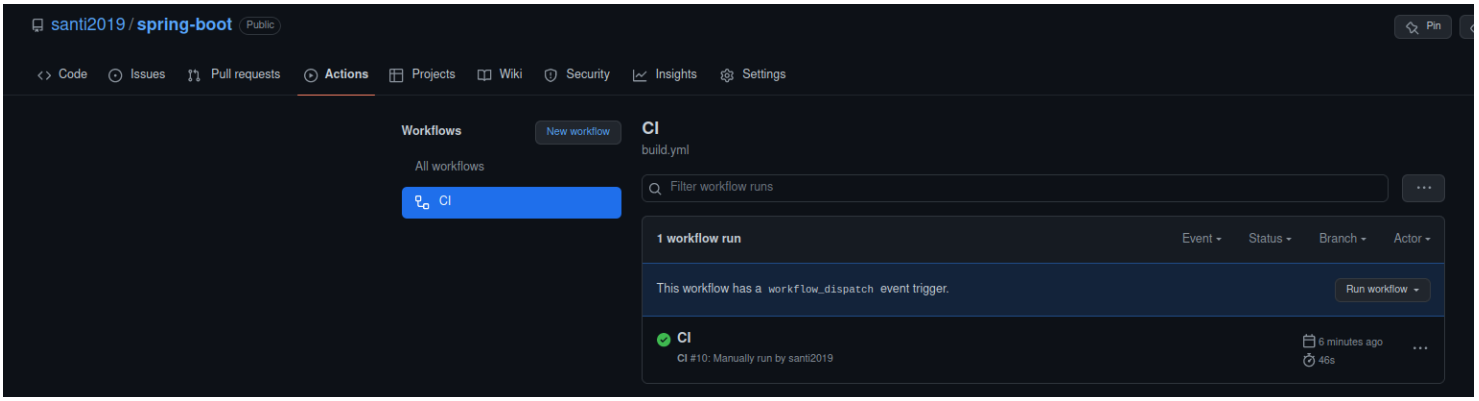
2)_ Creamos un nuevo workflow, llamado build.yml:

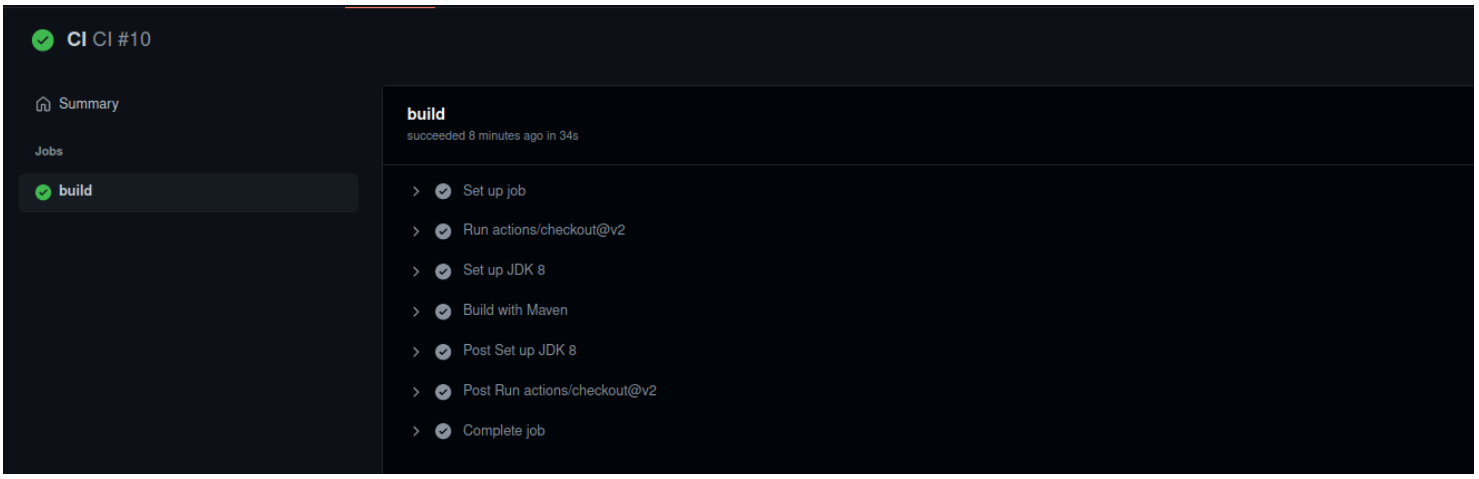


_ Colocamos el siguiente script:

```yaml
# This is a basic workflow to help you get started with Actions

name: CI

# Controls when the workflow will run
on:
 # Triggers the workflow on push or pull request events but only for the master branch
 push:
  paths:
  - 'spring-boot/**'
  branches: [ main ]
 pull_request:
  paths:
  - 'spring-boot/**'
  branches: [ main ]

 # Allows you to run this workflow manually from the Actions tab
 workflow_dispatch:

# A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:
 # This workflow contains a single job called "build"
 build:
  # The type of runner that the job will run on
  runs-on: ubuntu-latest

  # Steps represent a sequence of tasks that will be executed as part of the job
  steps:
   # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
   - uses: actions/checkout@v2

   # Install Java JDK with maven
   - name: Set up JDK 8
     uses: actions/setup-java@v2
     with:
      java-version: '8'
      distribution: 'adopt'
      cache: maven

   # Compile the application
   - name: Build with Maven
     run: |
      mvn -B package --file pom.xml
```

3)_ Le damos start commit, y luego procedemos a correr manualmente el workflow (pipeline) creado:

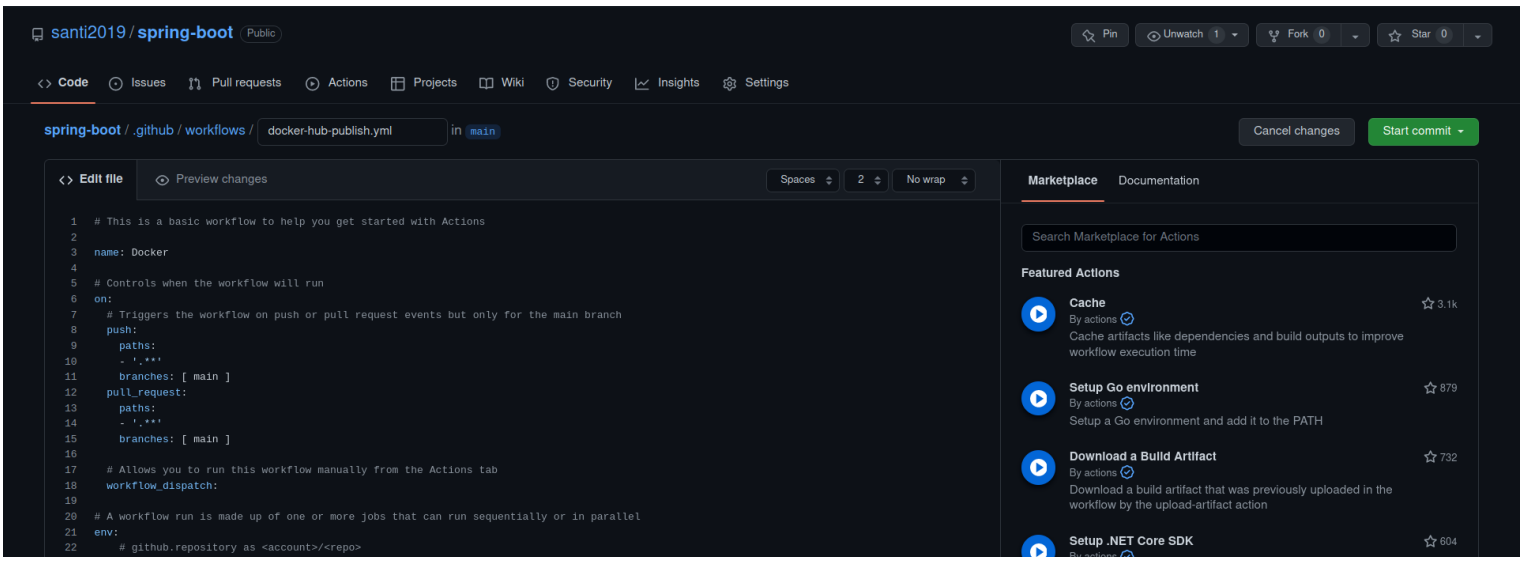




4)_ Explicando un poco lo que hace el pipeline, básicamente como primera medida definimos la ruta en GitHub donde esta el repo de spring-boot, luego en la parte de jobs se hace uso de actions/checkout@v2 que permite justamente acceder al repositorio de spring-boot, despues instala Java JDK 8 con Maven, y por último compilamos y corremos la aplicación con Maven.

Ejercicio 3:

1)_ Creamos un nuevo workflow, llamado docker-hub-publish.yml:



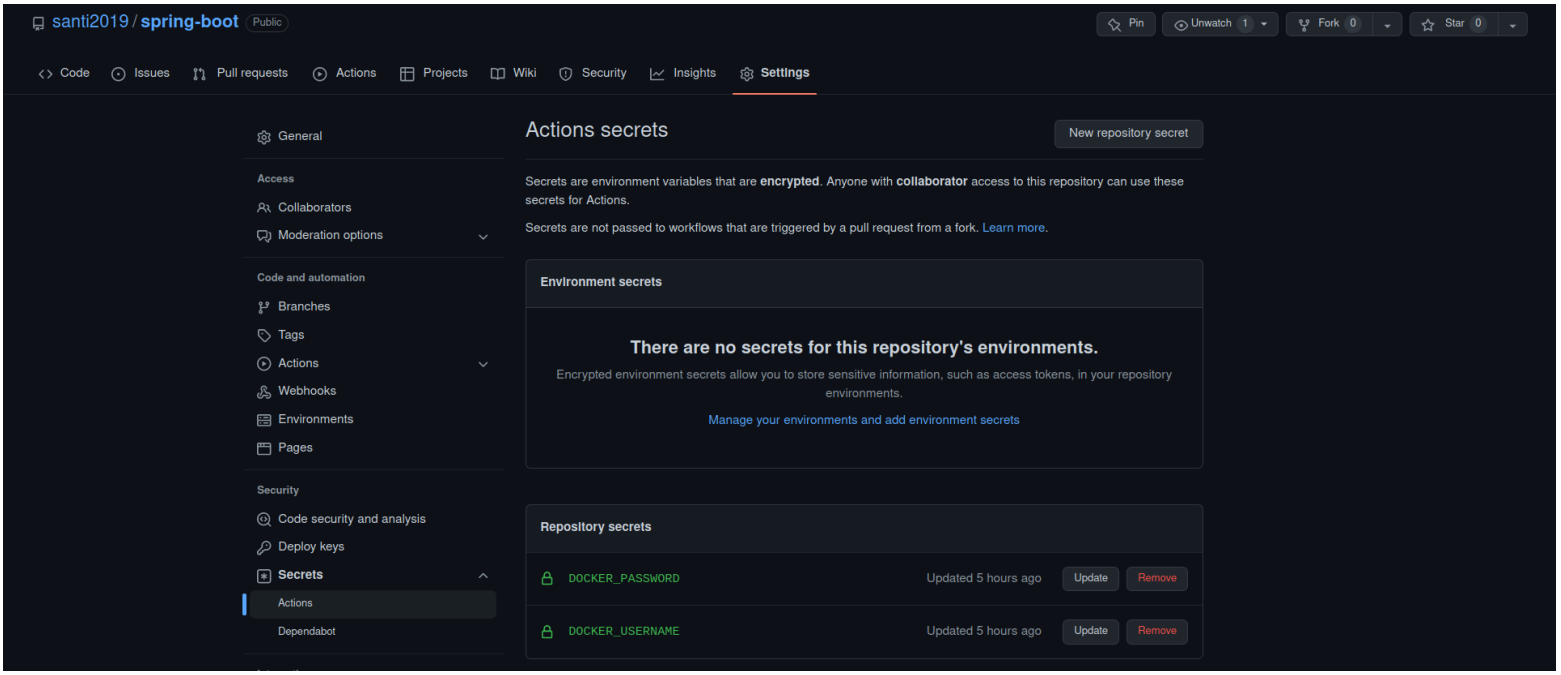
_ El script es el siguiente (me generó conflicto el hecho de que mi usuario de github es diferente que el de docker hub, pero pude solucionarlo):

96 lines (81 sloc) | 3.52 KB

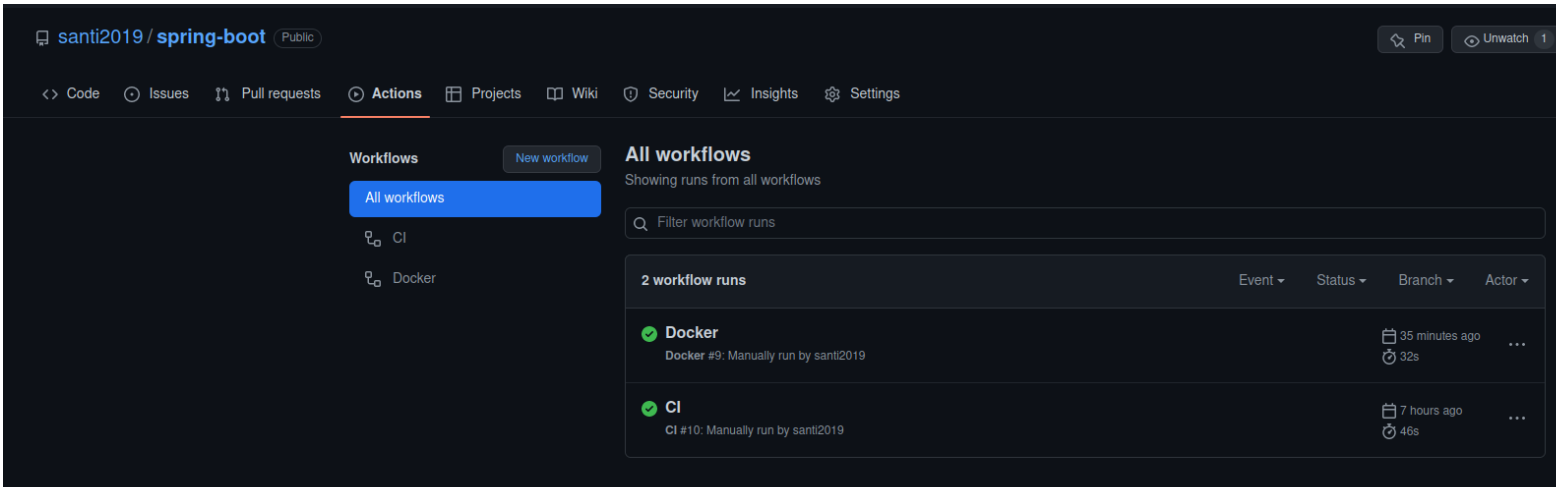
```yaml
# This is a basic workflow to help you get started with Actions

name: Docker

# Controls when the workflow will run
on:
  # Triggers the workflow on push or pull request events but only for the main branch
  push:
    paths:
    - '.**'
    branches: [ main ]
  pull_request:
    paths:
    - '.**'
    branches: [ main ]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

# A workflow run is made up of one or more jobs that can run sequentially or in parallel
env:
    # github.repository as <account>/<repo>
    IMAGE_NAME: spring-boot-github

jobs:
  # This workflow contains a single job called "build"
  build-docker:

      runs-on: ubuntu-latest
      permissions:
        contents: read
        packages: write
        # This is used to complete the identity challenge
        # with sigstore/fulcio when running outside of PRs.
        id-token: write

      steps:
        - name: Checkout repository
          uses: actions/checkout@v3

        # Install the cosign tool except on PR
        # https://github.com/sigstore/cosign-installer
```

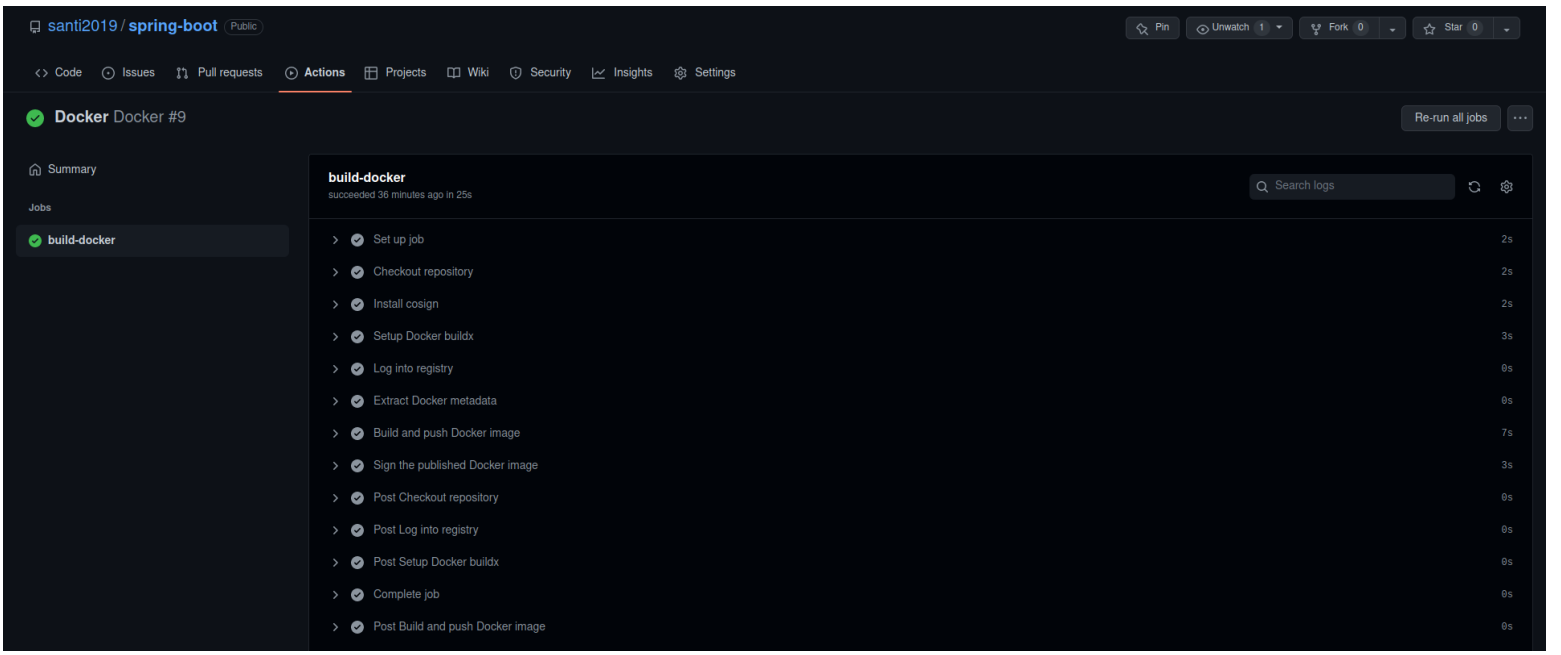
```yaml
      - name: Install cosign
        if: github.event_name != 'pull_request'
        uses: sigstore/cosign-installer@f3c664df7af409cb4873aa5068053ba9d61a57b6 #v2.6.0
        with:
          cosign-release: 'v1.11.0'


      # Workaround: https://github.com/docker/build-push-action/issues/461
      - name: Setup Docker buildx
        uses: docker/setup-buildx-action@79abd3f86f79a9d68a23c75a09a9a85889262adf

      # Login against a Docker registry except on PR
      # https://github.com/docker/login-action
      - name: Log into registry ${{ env.REGISTRY }}
        if: github.event_name != 'pull_request'
        uses: docker/login-action@28218f9b04b4f3f62068d7b6ce6ca5b26e35336c
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      # Extract metadata (tags, labels) for Docker
      # https://github.com/docker/metadata-action
      - name: Extract Docker metadata
        id: meta
        uses: docker/metadata-action@98669ae865ea3cffbcbaa878cf57c20bbf1c6c38
        with:
          images: ${{secrets.DOCKER_USERNAME}}/${{ env.IMAGE_NAME }}

      # Build and push Docker image with Buildx (don't push on PR)
      # https://github.com/docker/build-push-action
      - name: Build and push Docker image
        id: build-and-push
        uses: docker/build-push-action@ac9327eae2b366085ac7f6a2d02df8aa8ead720a
        with:
          context: .
          push: ${{ github.event_name != 'pull_request' }}
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
          cache-from: type=gha
          cache-to: type=gha,mode=max


      # Sign the resulting Docker image digest except on PRs.
      # This will only write to the public Rekor transparency log when the Docker
      # repository is public to avoid leaking data.  If you would like to publish
      # transparency data even for private images, pass --force to cosign below.
      # https://github.com/sigstore/cosign
      - name: Sign the published Docker image
        if: ${{ github.event_name != 'pull_request' }}
        env:
          COSIGN_EXPERIMENTAL: "true"
        # This step uses the identity token to provision an ephemeral certificate
        # against the sigstore community Fulcio instance.
        run: echo "${{ steps.meta.outputs.tags }}" | xargs -I {} cosign sign {}@${{ steps.build-and-push.outputs.digest }}
```

2)_ Generamos los secretos, en donde colocamos nuestro usuario y contraseña de docker hub:

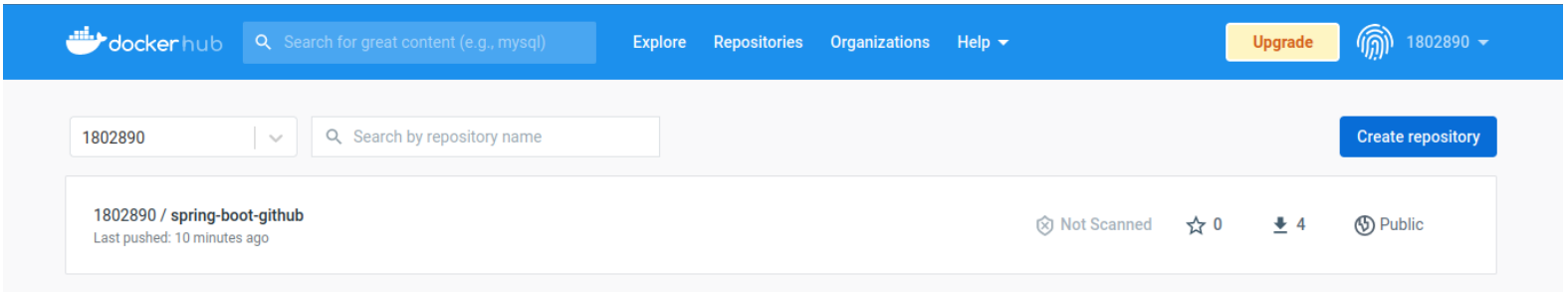


3)_ Procedemos a correr manualmente el workflow (pipeline) creado:





_ Corroboramos que se haya subido correctamente el repositorio a Docker Hub:



Ejercicio 4:

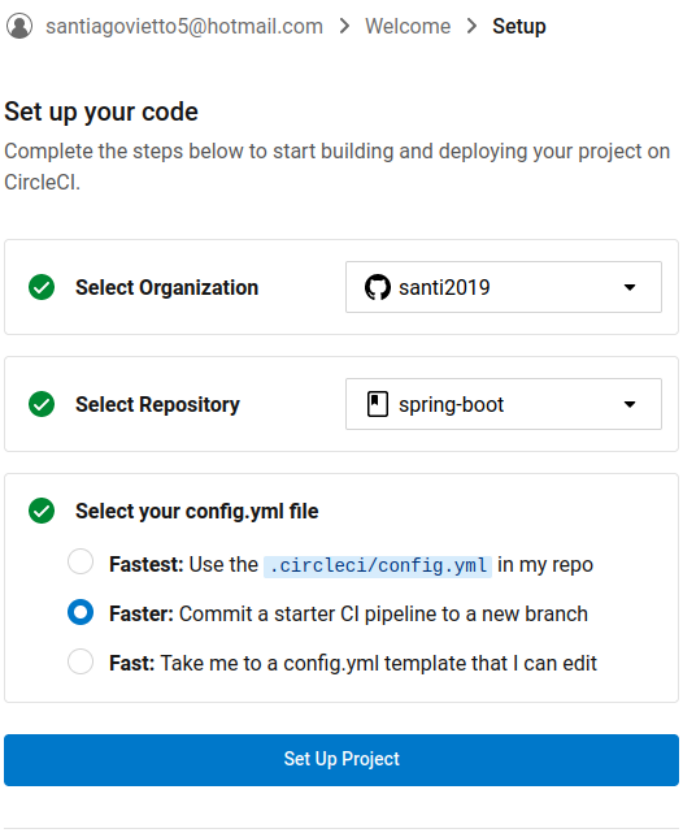
_ A modo opcional, similar al ejercicio 2, configuramos un build job para el mismo proyecto, pero utilizando Circle CI. A continuación creamos el script, en donde este mismo es un ejemplo básico que provee Circle CI para mostrar un “Hello World”:

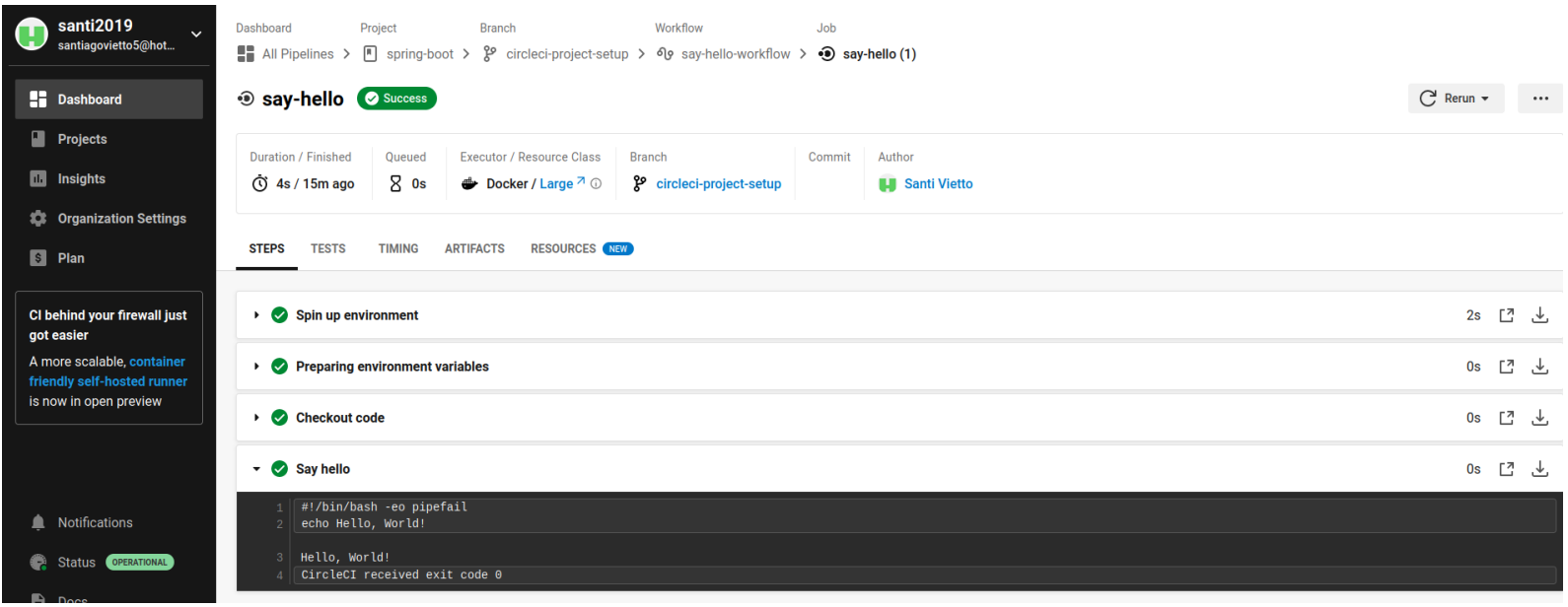
- https://circleci.com/docs/hello-world/

30 lines (30 sloc) | 1.04 KB — Raw | Blame

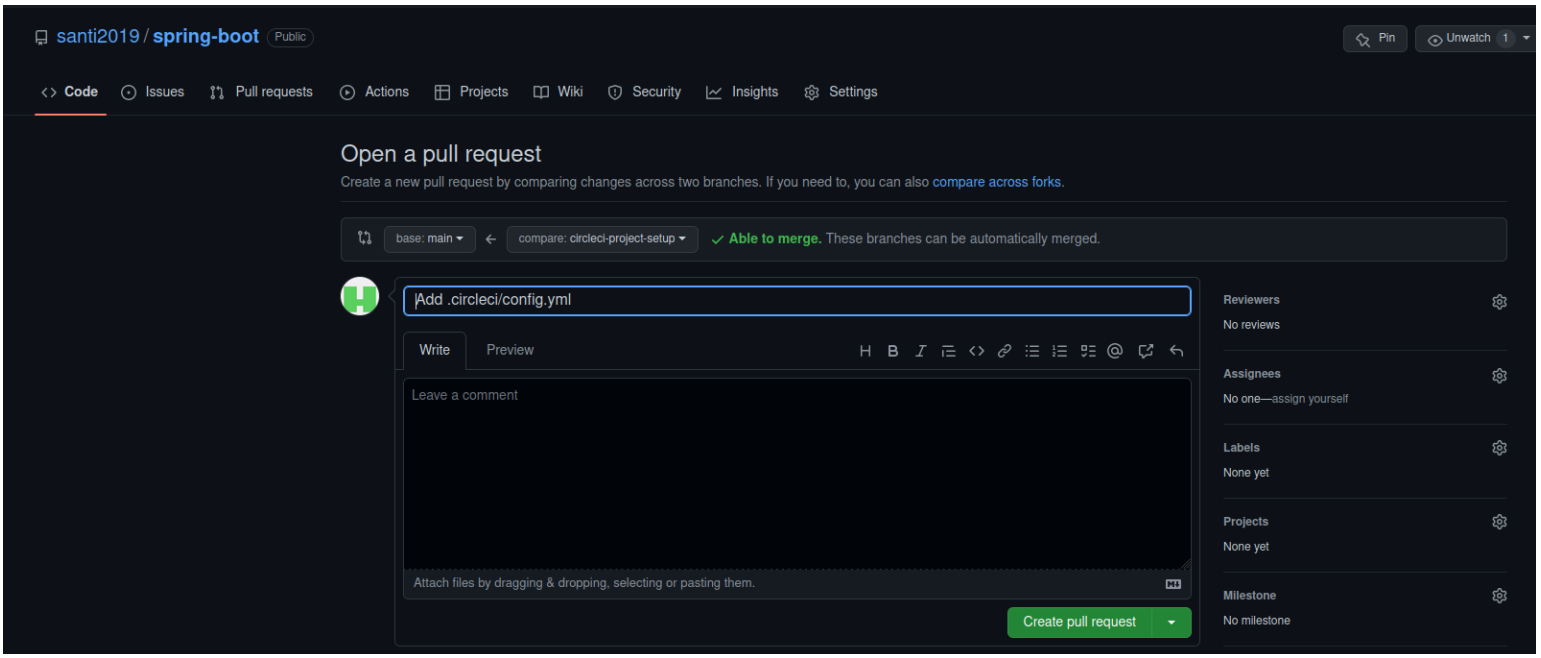
```
# Use the latest 2.1 version of CircleCI pipeline process engine.
# See: https://circleci.com/docs/2.0/configuration-reference
version: 2.1
orbs:
  maven: circleci/maven@1.3.0
# Define a job to be invoked later in a workflow.
# See: https://circleci.com/docs/2.0/configuration-reference/#jobs
jobs:
  say-hello:
    # Specify the execution environment. You can specify an image from Dockerhub or use one of our Convenience Images from CircleCI's Developer Hub.
    # See: https://circleci.com/docs/2.0/configuration-reference/#docker-machine-macos-windows-executor
    docker:
      - image: cimg/base:stable
    # Add steps to the job
    # See: https://circleci.com/docs/2.0/configuration-reference/#steps
    steps:
      - checkout
      - run:
          name: "Say hello"
          command: "echo Hello, World!"
# Invoke jobs via workflows
# See: https://circleci.com/docs/2.0/configuration-reference/#workflows
workflows:
  say-hello-workflow:
    jobs:
      - say-hello
  maven_test:
    jobs:
      - maven/test:
          command: '-B package --file pom.xml'
```

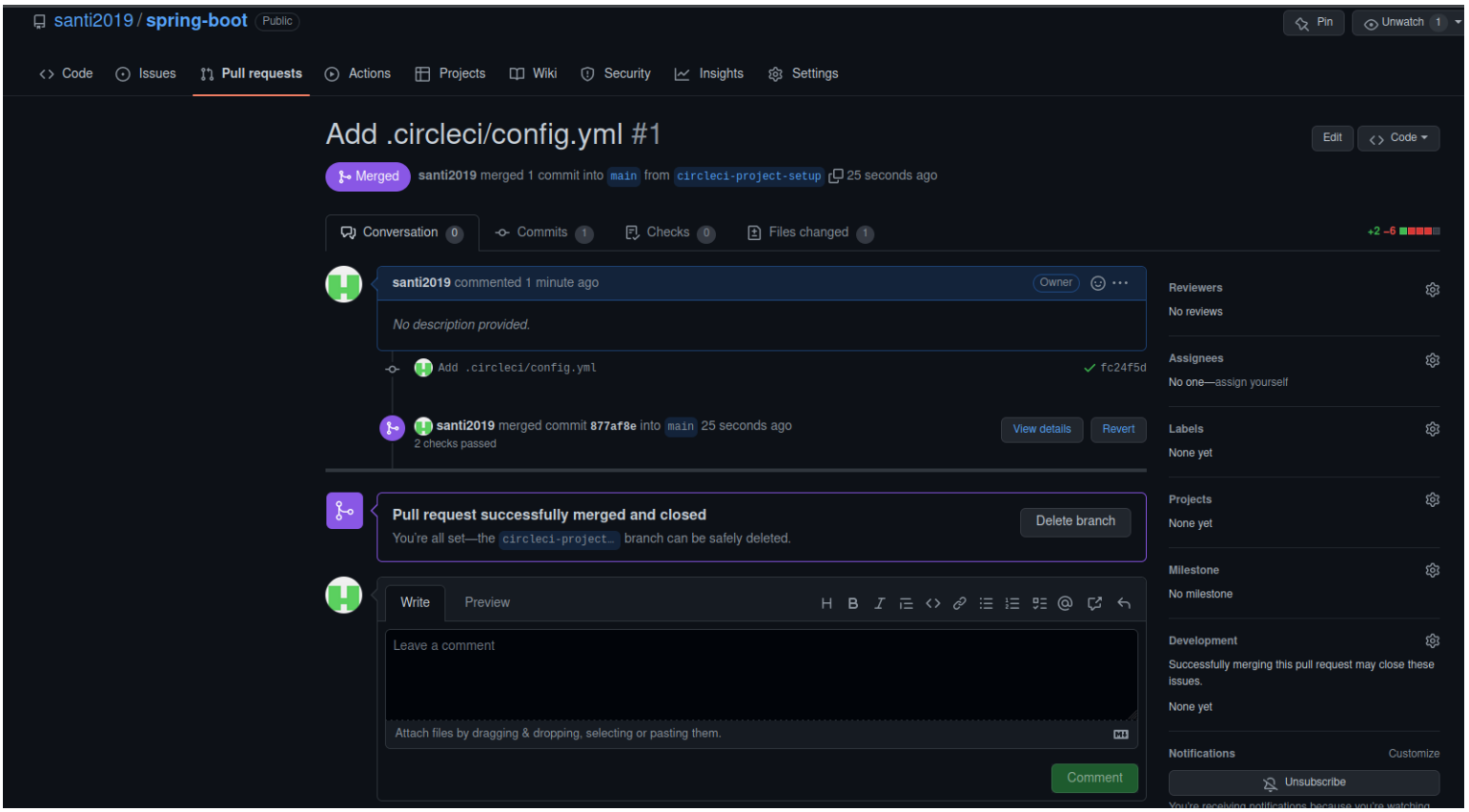
_ A continuación accedemos al sitio de Circle CI, nos registramos, nos vinculamos con GitHub, asignamos el script como podemos observar y lo corremos:

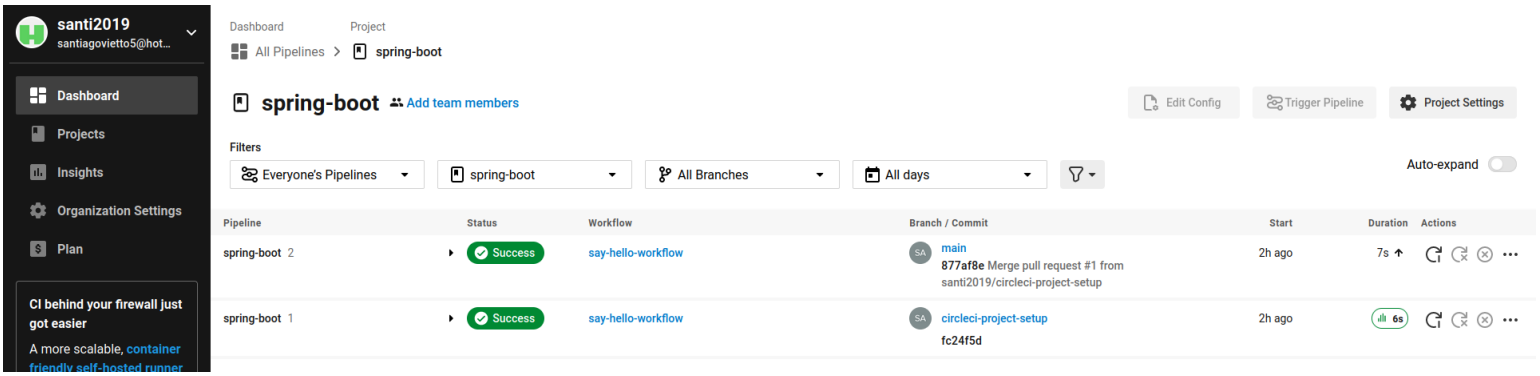




_ Luego, si regresamos a GitHub, nos sugiere realizar un merge pull request ya que Circle CI generó una branch llamada circleci-project-setup en donde esta el archivo config.yml, entonces para combinar esta branch con la branch main creamos un pull request, y como vemos al final, se mergeo todo a la branch main. Si regresamos al sitio de Circle CI, podemos ver dos pipelines del mismo workflow pero de diferentes branches.






santi2019
santiagovietto5@hot...
Dashboard
Projects
Insights
Organization Settings
Plan
CI behind your firewall just got easier
A more scalable, container friendly self-hosted runner
Dashboard
Project
All Pipelines
spring-boot
spring-boot
Add team members
Edit Config
Trigger Pipeline
Project Settings
Filters
Everyone's Pipelines
spring-boot
All Branches
All days
Auto-expand
Pipeline
Status
Workflow
Branch / Commit
Start
Duration
Actions
spring-boot 2
Success
say-hello-workflow
main
877af8e Merge pull request #1 from santi2019/circleci-project-setup
2h ago
7s
spring-boot 1
Success
say-hello-workflow
circleci-project-setup
fc24f5d
2h ago
6s