Integración Continua y Despliegue Continuo

Administración de Sistemas

Unai Lopez Novoa

unai.lopez@ehu.eus

eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Contenido

Introducción

- Flujo tradicional de puesta en producción
  - Partiendo de un repositorio de código.

Verificar el código (*Tests*)

Construir artefactos (p.e. imágenes)

Desplegar en un entorno de ejecución

Introducción

- Flujo tradicional de puesta en producción.

- Estas tareas suelen ser repetitivas.
- Realizarlas a mano supone:
  - Gasto de tiempo.
  - Riesgo de errores.

CI / CD

- Las técnicas de Integración Continua (CI) y Despliegue Continuo (CD) sirven para automatizar flujos de trabajo.

- El objetivo es reducir costes en:
  - Tiempo invertido a tareas repetitivas.
  - Tiempo solucionando errores.

DevOps

- Integración continua (CI):
  - Objetivo: Código listo para funcionar.
  - Basado en *tests* automatizados.
- Despliegue continuo (CD):
  - Objetivo: Poner una aplicación en marcha.
  - Basado en *workflows*/*pipelines* de despliegue.
- Las técnicas CI/CD son parte de la metodología DevOps:
  - Cultura que pretende acercar a los equipos de desarrollo (Dev) con los equipos de operaciones (Ops).

Fuente: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

DevOps

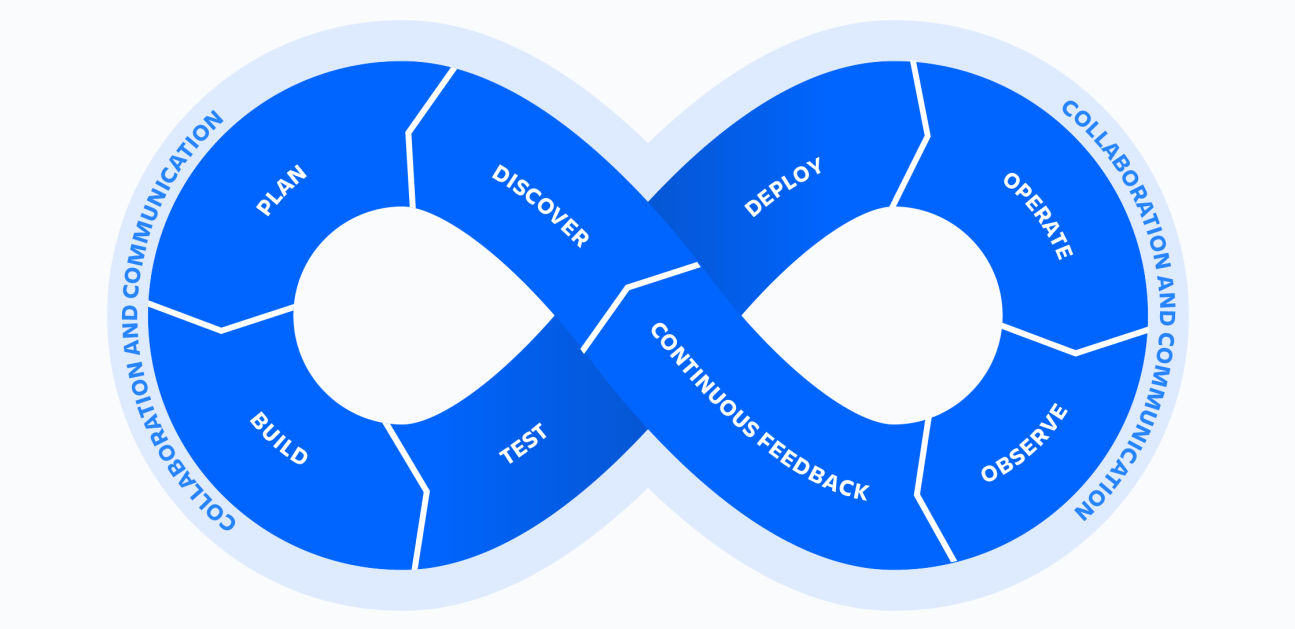
- Los equipos *Dev* y *Ops* tienen objetivos contrarios:

Desarrolladores:
- Nuevas funcionalidades
- Código en marcha cuanto antes

Operaciones:
- Maximizar eficiencia de los sistemas
- Minimizar errores

DevOps

- Ciclo de vida DevOps:



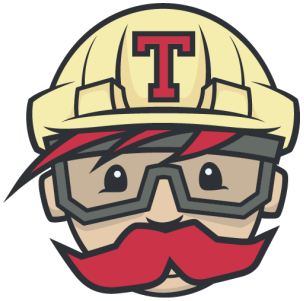
- Más información: https://www.atlassian.com/devops

Imagen: https://www.atlassian.com/devops

DevOps

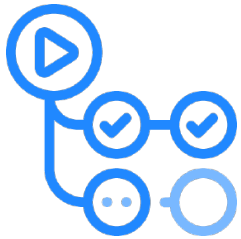
- Herramientas populares para CI/CD:

Jenkins



Travis

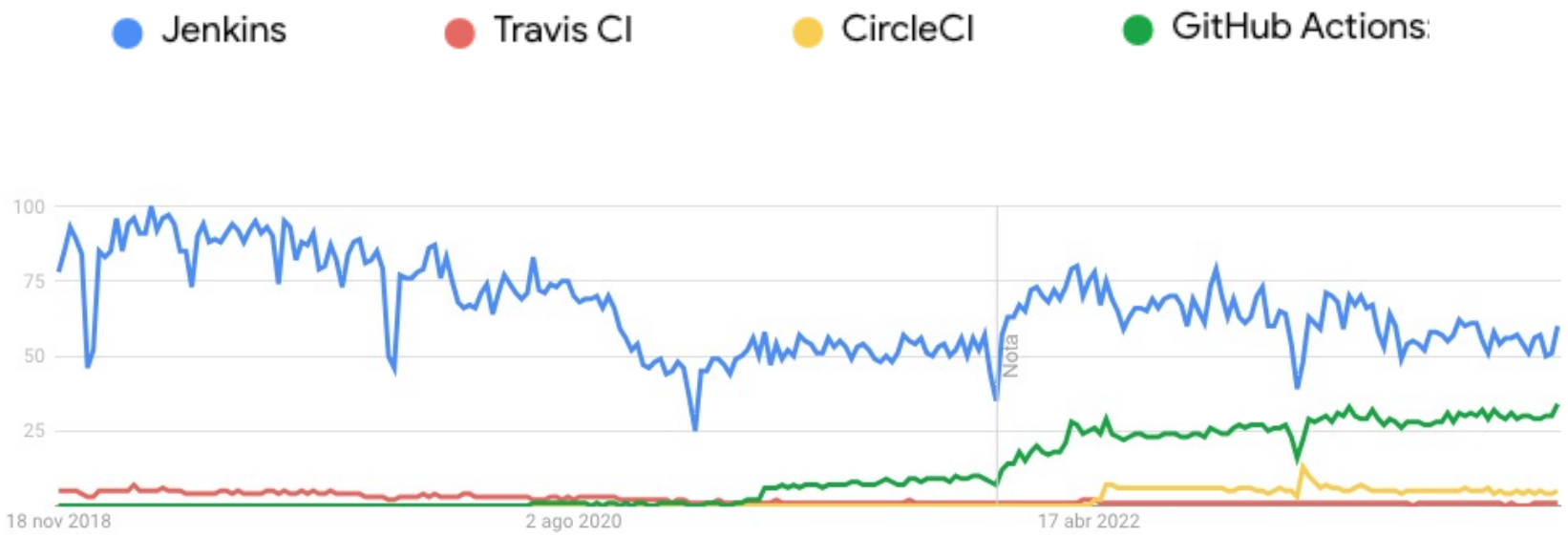
CircleCI



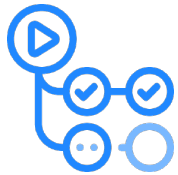
GitHub Actions

DevOps

- Herramientas populares para CI/CD:
  - Según Google Trends, fecha 13 de noviembre de 2023.

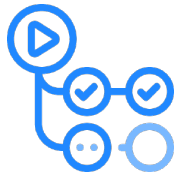


GitHub Actions



- Es una herramienta para automatizar flujos de trabajo (*workflows*).
- Está plenamente integrado con GitHub.
- Crear *workflows* es sencillo.
  - Comparado con otras herramientas.
- Es muy integrable con otras plataformas.

GitHub Actions



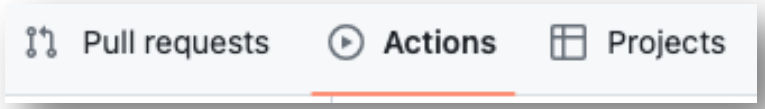
- Ejecuta acciones automáticamente en respuesta a eventos que suceden en un repositorio de código.
  - P.e. al hacer *Push*, tras un *Pull Request*, al abrir una incidencia, ...

- Al suceder un evento, se dispara un *workflow* asociado.
  - *Ejemplo*: cuando suceda *Push* en un repositorio, disparar un workflow que ejecute Tests.

GitHub Actions

- Los *workflow* se definen como ficheros YAML dentro del propio repositorio de código.
  - En la carpeta .github/workflows

- La actividad de los *workflow* se consulta en la pestaña Actions del repositorio.



GitHub Actions

- Un *workflow* está formado por tareas.
  - Ejemplo: Ejecutar una tarea “Hola mundo” tras un Push en el repositorio.

```
name: Primer workflow

on: push

jobs:
  hola_mundo:
    runs-on: ubuntu-latest

    steps:
      - name: Mostrar mensaje
        run: echo "Hola mundo"
```

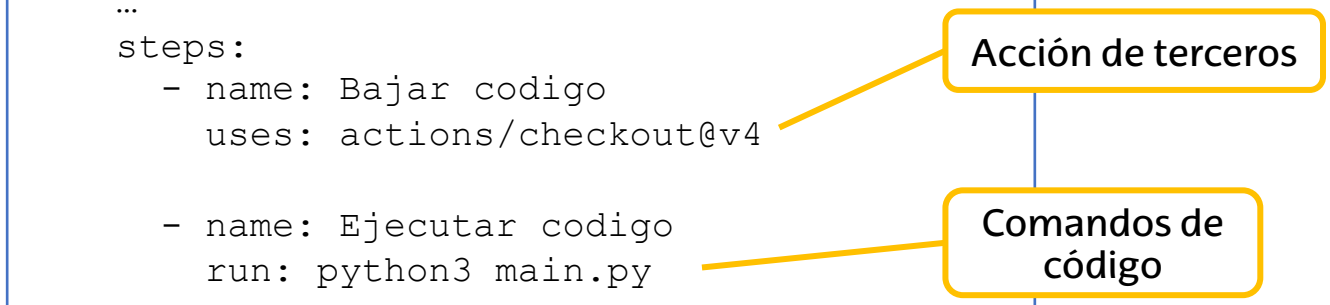
GitHub Actions

- Las tareas pueden contener acciones de terceros o comandos a ejecutar.
  - Ejemplo:

```
…
jobs:
  hola_mundo:
    …
    steps:
      - name: Bajar codigo
        uses: actions/checkout@v4

      - name: Ejecutar codigo
        run: python3 main.py

      - name: Ejecutar código en /tmp
        run: |
          cd /tmp
          python3 main.py
```



GitHub Actions

- Los *workflows* se ejecutan en servidores de GitHub llamados *runners*.
- Cada *job* de un *workflow* se ejecuta en una nueva máquina virtual dedicada y desechable.
- Es posible utilizar servidores propios como *runners* para ejecutar *workflows*.
  - Más información: https://docs.github.com/en/actions/hosting-your-own-runners/

Ejercicio 1

- Crear un repositorio de código basado en:
  - https://github.com/ulopeznovoa/AS-CI-CD-ej1

- Crear un *workflow* con los siguientes pasos:
  - Descargar el código.
  - Instalar pytest.
  - Utilizar pytest para ejecutar los *Tests* del código.

```
pip install pytest
```

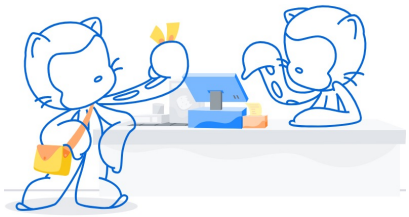
```
pytest test.py
```

- Introducir un fallo en el código y comprobar el resultado del *workflow*.

GitHub Actions

- Con Actions podemos definir el código que queremos que se ejecute en cada paso.
- *Pero...*
- En muchos casos, ese código es repetitivo y reusable en múltiples proyectos.

GitHub Marketplace



- Catálogo de aplicaciones y acciones integrables con GitHub Actions.
  - URL: https://github.com/marketplace
- Las acciones realizan tareas de uso común.
  - Nos ahorran múltiples líneas de código y configuración.
- Las aplicaciones proveen funcionalidades más avanzadas que las acciones.
  - Algunas son de pago.

GitHub Actions

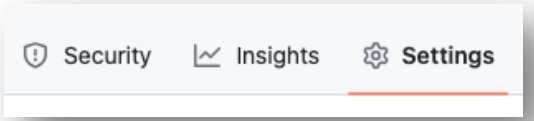
- Ejemplo:
  - Acción para construir una imagen Docker y subir a un repositorio.
    - URL: https://github.com/marketplace/actions/build-and-push-docker-images
  - Indicar parámetros con “with”, en este caso:
    - “push”: Subir imagen si/no.
    - “tags”: Nombre para la imagen construida.

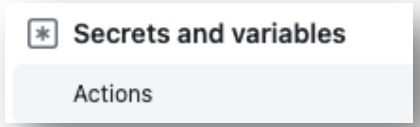
```
- name: Build and push
  uses: docker/build-push-action@v5
  with:
    push: true
    tags: ulopeznovoa/holamundo
```

GitHub Actions

- En algunas acciones, puede ser necesario usar información sensible.
  - P.e. credenciales de acceso.

- Recomendable utilizar secretos.
  - Variables que guardan información de manera cifrada.
  - Usables desde las acciones.
  - A nivel de repositorio, se configuran en la sección de Configuración.
    - Apartado de “Secretos y variables”





GitHub Actions

- Ejemplo:
  - Acción para autenticarse en Docker Hub.
    - URL: https://github.com/marketplace/actions/build-and-push-docker-images
  - Los parámetros “username” y “password” se leen como secretos.

```
- name: Login to Docker Hub
  uses: docker/login-action@v3
  with:
    username: ${{ secrets.DOCKERHUB_USERNAME }}
    password: ${{ secrets.DOCKERHUB_TOKEN }}
```

Ejercicio 2

- Crear un repositorio público con un fichero de código Python que contenga lo siguiente:
  - Una función que sume 2 números.
  - Dos variables con 2 números (a vuestra elección).
  - Un comando *print* para mostrar la suma de los 2 números usando la función.

- *Continua en la siguiente diapositiva*

Ejercicio 2

- Crear un workflow en el repositorio que utilice la siguiente acción para analizar la calidad del código.
  - *Nombre*: advanced-security/python-lint-code-scanning-action@v1
  - *Parámetros*:
    - linter: pylint
  - Es necesario añadir el siguiente parámetro al workflow:

```
runs-on: …
permissions:
  security-events: write
steps:
```

- Comprobar el resultado.
  - Repositorio, sección "*Seguridad*", apartado "*Análisis de código*".

Workflows

- Un workflow puede estar dividido en múltiples tareas
- Las dependencias entre tareas se indican con el parámetro *needs*.
  - Ejemplo: "build" se ejecutará sólo si "test" finaliza correctamente.

```
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      …

  build:
    runs-on: ubuntu-latest
    needs: test
    steps:
      …
```

Workflows

- Se puede indicar a qué eventos debe responder un workflow en el parámetro *on*.
  - Ejemplo: Disparar *workflow* si hay un evento Push en las ramas *feature-a* y *feature-b*.

```
on:
  push:
    branches:
      - feature-a
      - feature-b
```

- Listado completo de eventos en la documentación:
  - https://docs.github.com/en/rest/overview/github-event-types

Workflows

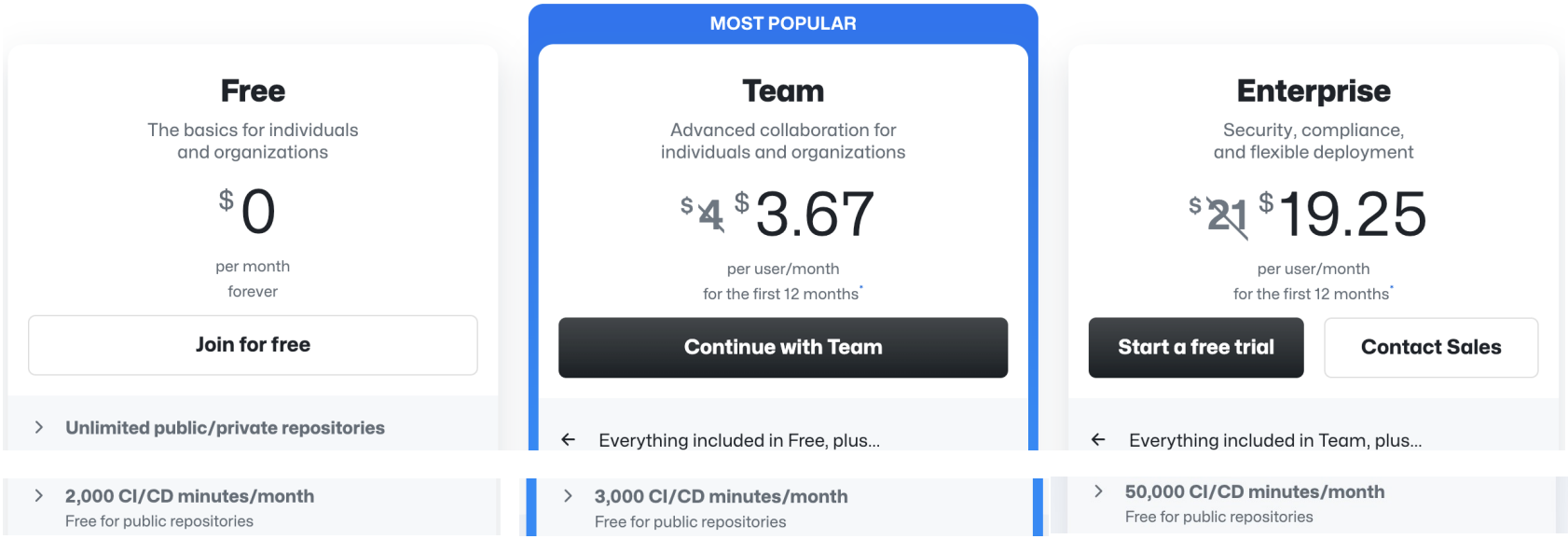
- Se pueden ejecutar *workflows* en Runners con diferentes sistemas operativos.
  - Listado completo en: https://docs.github.com/en/actions/using-github-hosted-runners
  - Ejemplo: ejecutar la misma tarea en Runners con Ubuntu, Windows y macOs.

```
jobs:
  test:
    runs-on: ${{matrix.os}}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macOs-latest]

    steps:
      - name: Bajar codigo
```

GitHub Actions

- GitHub funciona con un modelo *freemium*.
  - Precios a fecha 13 de noviembre de 2023.



GitHub Actions

- GitHub funciona con un modelo *freemium*.
  - Precios a fecha 13 de noviembre de 2023.
    - Fuente: https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions

Included storage and minutes

Plan	Storage	Minutes (per month)
GitHub Free	500 MB	2,000
GitHub Pro	1 GB	3,000
GitHub Free for organizations	500 MB	2,000
GitHub Team	2 GB	3,000
GitHub Enterprise Cloud	50 GB	50,000

Minute multipliers

Operating system	Minute multiplier
Linux	1
Windows	2
macOS	10

Ejercicio 3

- *Utilizar el repositorio de código creado en el ejercicio 1.*

- Crear 2 *workflows*:
  1) Se dispara al hacer *Push* en *main*, ejecuta los *Tests* como se describe en el ejercicio 1.
  2) Se dispara al hacer *Pull Request* en cualquier rama, ejecuta la acción Linter presentada en el ejercicio 2.

- Crear una nueva rama y añadir en el código:
  - Una función que incremente el saldo en 1000.
  - El Test correspondiente.

- Hacer una *Pull Request* para juntar la rama con *main*.
  - Verificar que cada *workflow* se ha ejecutado correctamente.

Bibliografía

- GitHub Actions Docs.
  - https://docs.github.com/en/actions
- Nana Janashia, “GitHub Actions Tutorial - Basic Concepts and CI/CD Pipeline with Docker”.
  - https://youtu.be/R8_veQiYBjI
- Miguel Ángel Durán, “GitHub Actions Tutorial”
  - https://youtu.be/sIhm4YOMK6Q

- Consultados en noviembre 2023