

# DEVOPS: OPTIMIZANDO EL CICLO DE VIDA DEL DESARROLLO DE SW

Rubén Gavilán

Software Architecture Lead  
Izertis

**izertis**  
Passion for Technology





# Quiénes Somos

Somos una **Consultora Tecnológica** que ayuda a las organizaciones en su proceso de **Metamorfosis Digital**.

Nuestra diferenciación se basa en nuestro **Offering** de servicios y tecnologías de vanguardia, y en nuestra **Cultura** moderna, ágil, innovadora y flexible.



**+121**

M€ Ingresos Totales



**+1.800**

Profesionales



**28 años**

de historia

Cotizados en el  
**BME Growth**



# Qué Hacemos

## DX STRATEGY & TECH INNOVATION

Acompañamiento estratégico y táctico para el diseño y despliegue de soluciones y productos digitales innovadores basados en inteligencia artificial y otras tecnologías de vanguardia.

Consultoría Estratégica | IA | Data | Blockchain | PLN | Phygital

## SOFTWARE ENGINEERING

Desarrolla soluciones de software a medida para poner la tecnología al servicio de tu organización y de tus clientes.

IADev | Agile Development | DevSecOps | Mobile | Digital Platforms | ECM | BMP | Quality Assurance

## CLOUD & INFRASTRUCTURES

Construye una infraestructura tecnológica que te permita disponer de tus servicios internos y externos en cualquier lugar y dispositivo.

Modernización IT | Cloud Híbridas | Servicios Gestionados | Administración delegada Infraestructura | Productividad | Comunicaciones

## CUSTOMER XPERIENCE

Estrategias digitales donde la creatividad y la personalización a través del dato nos ayudan a crear experiencias únicas y rentables.

Strategy | Brand & Creativity | User Experience | Sales Growth | Data Intelligence | CRM Marketing Platform

## ENTERPRISE IT SECURITY

Manejo de "información sensible" y "protección del dato" en origen, tránsito y destino son garantía para la continuidad de tu negocio. Evaluamos tu nivel de resiliencia ante incidentes y facilitamos el cumplimiento normativo de tu organización.

Arquitecturas seguras | IA segura | Auditoría | Pentesting | Red Team | Vigilancia digital | Apoyo ante incidentes | GRC | Concienciación | Gobernanza de la ciberseguridad

## ENTERPRISE & IT GOVERNANCE

Logra los objetivos de tu organización definiendo e implantando modelos de gestión y gobierno eficientes, realizando una asignación óptima de tus recursos y gestionando tus proyectos y servicios mediante estructuras y procesos ágiles.

Gobierno TI | Procesos | Eficiencia operativa | Gestión del cambio | Gestión de proyectos, programas y portafolios | PMO | Gestión de servicios | SMO | Soluciones PPM/ESM/ITS

## BUSINESS SOLUTIONS

Implementa herramientas, tecnologías y estrategias que ayuden a tu organización a alcanzar los objetivos comerciales y mejorar su eficiencia operativa.

Salesforce | Microsoft Business CRM y ERP | SAP BI | Infor

Nuestra Inteligencia Artificial es líquida: fluye, se expande y se adapta a todos nuestros servicios, impulsando la eficiencia, el rendimiento y la innovación.

# **Descubriendo la Cultura y Prácticas de DevOps**

**¿QUÉ ES DEVOPS PARA TI?**

# Flujos de trabajo DevOps



# DevSecOps Stack

Gestión, colaboración y comunicación



Control de versiones



Automatización CI/CD



Análisis automático



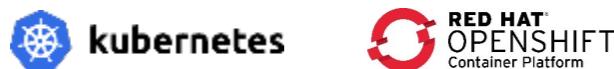
Gestión de artefactos



Cloud/Infraestructura de despliegue



Orquestación de contenedores



Monitorización



Ciberseguridad

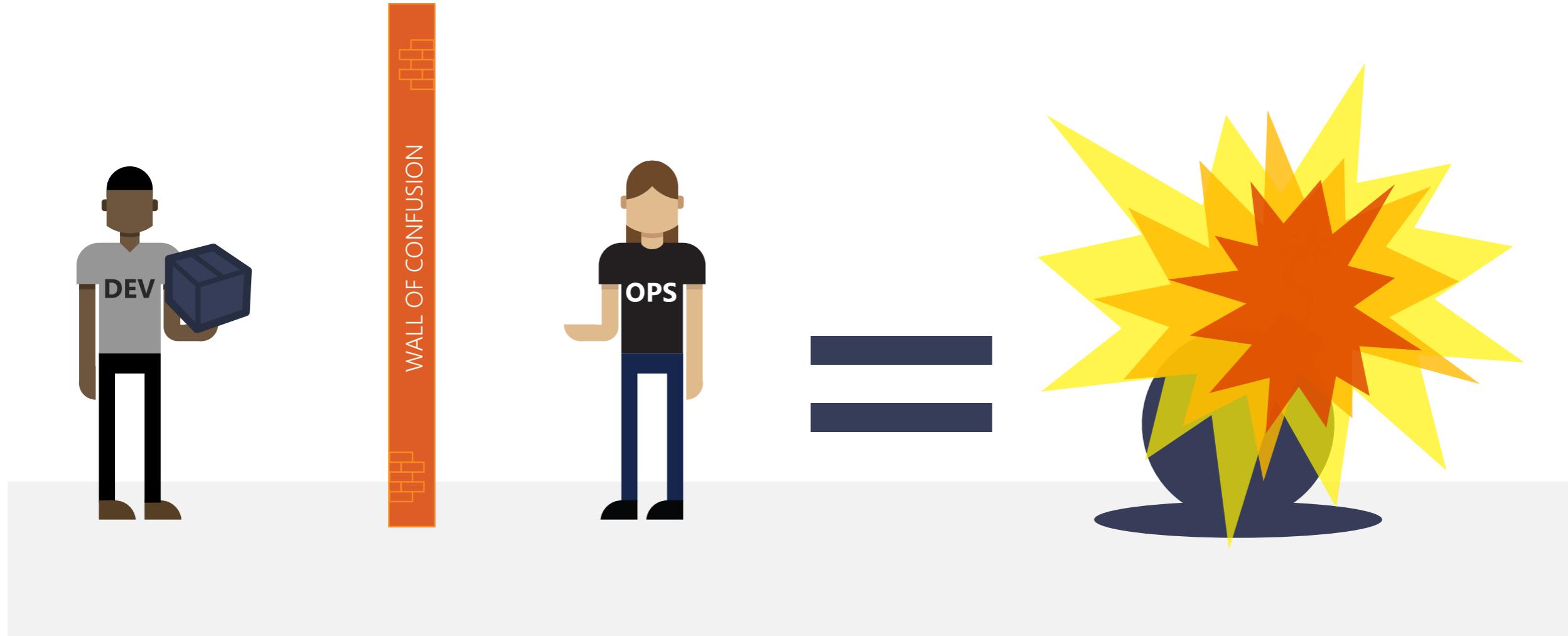


Operación



**¿ES SUFICIENTE?**

# Muro de confusión





## ¿Qué se requiere?



Cambio de cultura



Colaboración entre áreas

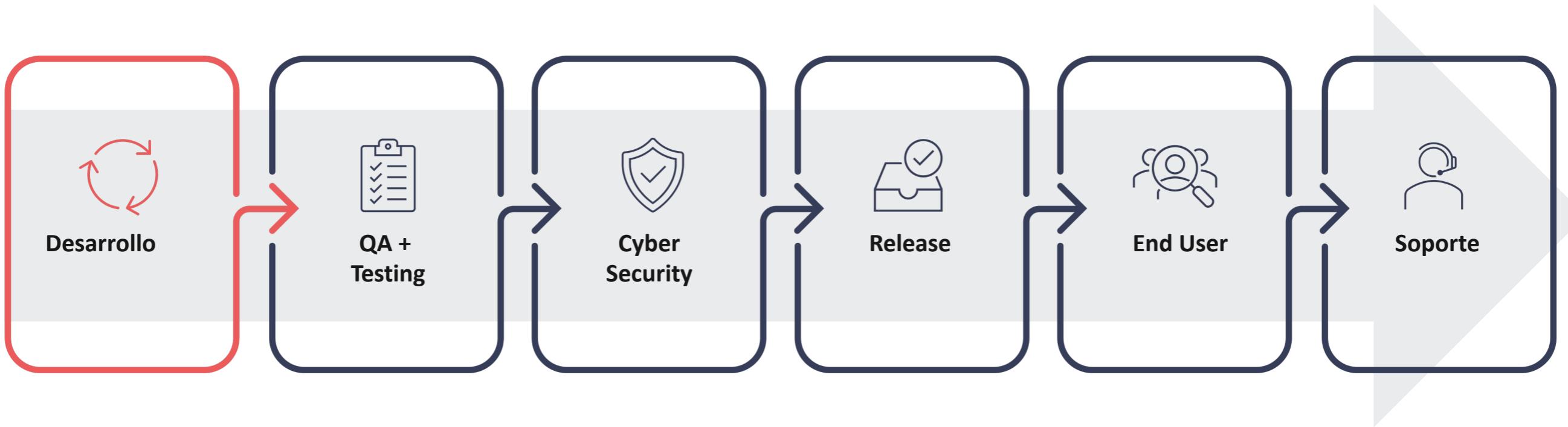


Carácter innovador



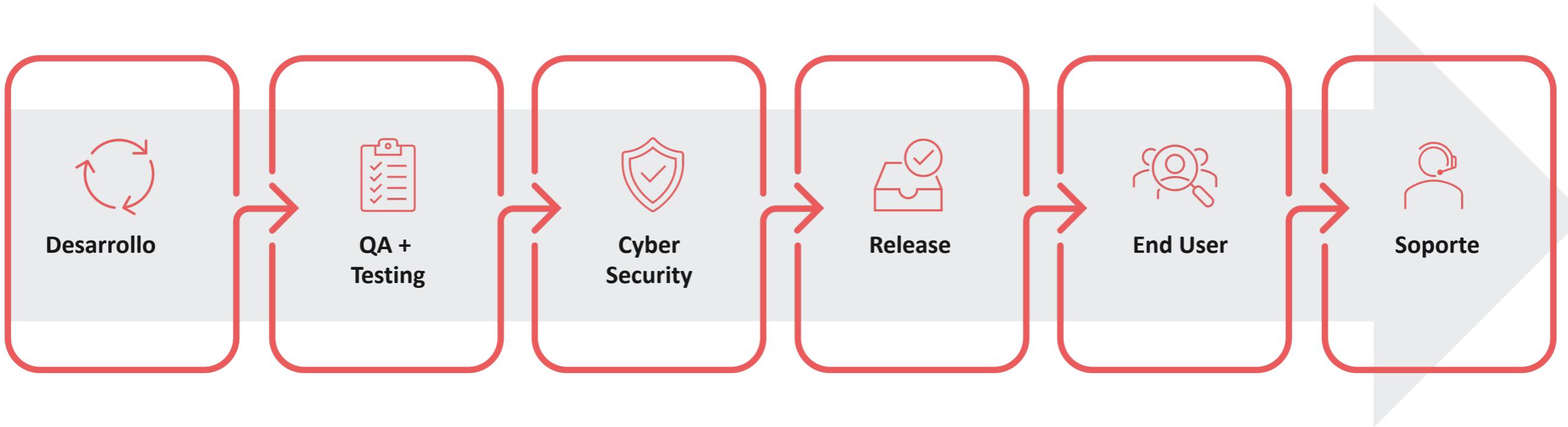
Soporte desde la alta  
dirección

# Visión tradicional



● Visibilidad del desarrollador

# Modelo DevSecOps – E2E



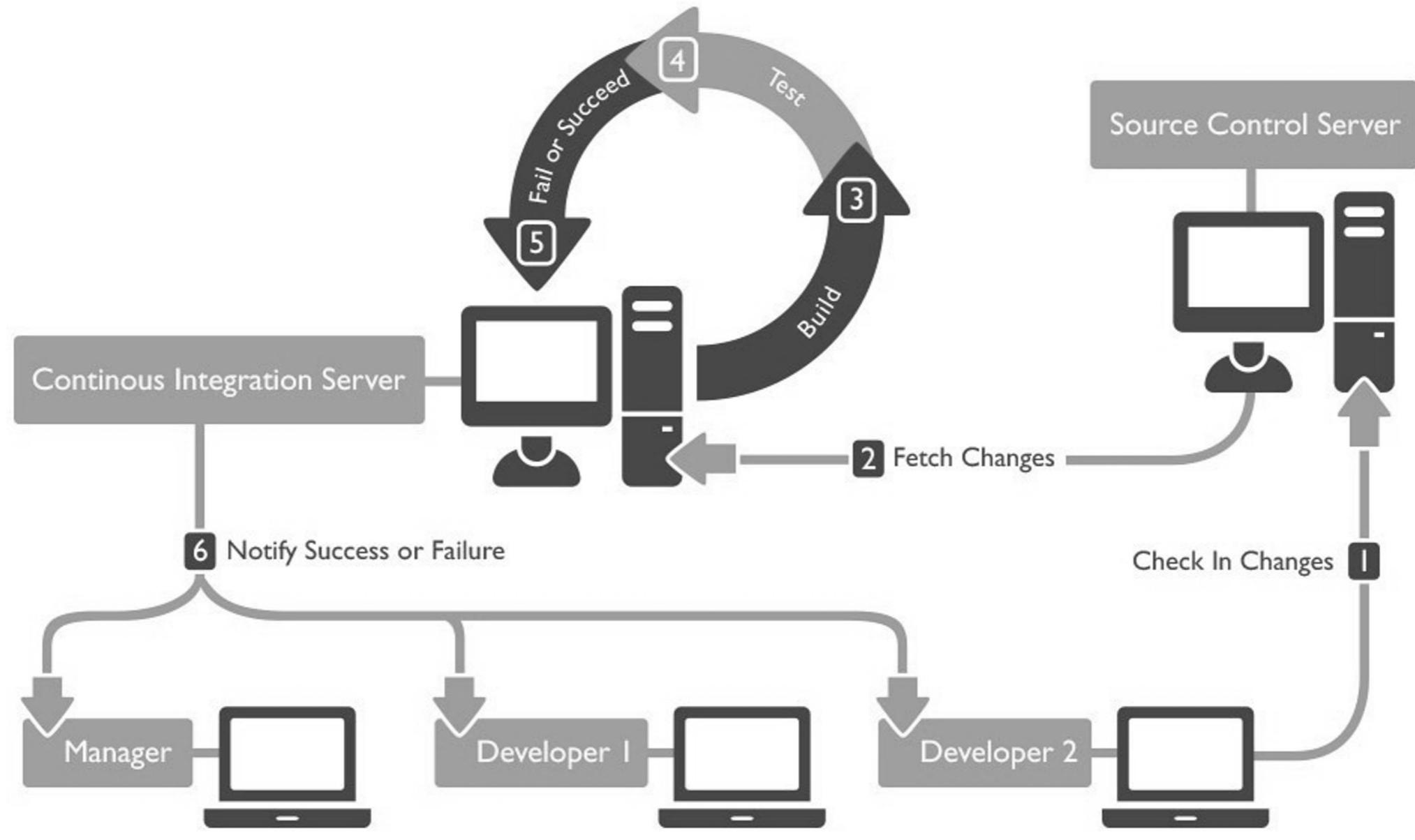
- Visibilidad del equipo del proyecto



# Beneficios



# CONTINUOUS INTEGRATION



# Beneficios CI



**Mejora de calidad  
del código**



**Detección de errores  
más rápida**



**Reducción de tareas  
repetitivas y manuales**



**Creación de versiones  
de prueba**



**Visibilidad completa  
del proyecto**



**Mayor confianza y  
seguridad del equipo**



## ¿Qué elementos se necesitan?

- Infraestructura de ejecución de pipelines
- Control de versiones
- Repositorio de artefactos (Maven, Nuget, NPM, ...)
- SDKs
- Herramientas de análisis y reporting (SCA, DC, SAST, ...)
- Canales de notificación
- *Documentación*

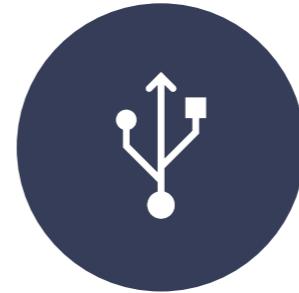
## CI empieza en uno mismo



Seguimiento de  
buenas prácticas



Documentación  
de los proyectos



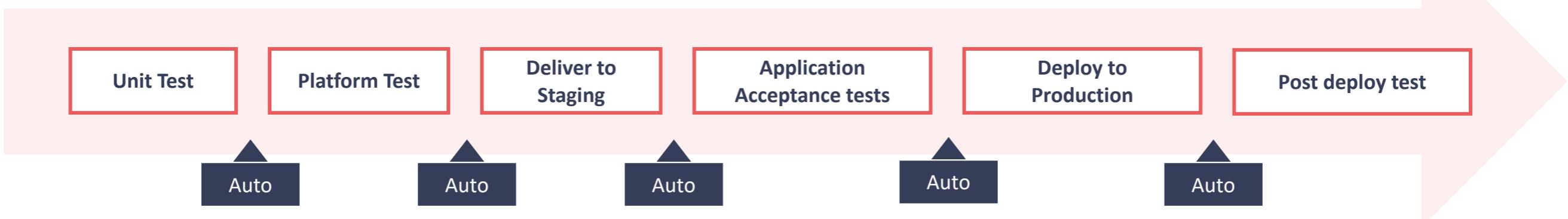
Plugins para IDE  
(sonarlint)

# CONTINUOUS DELIVERY

## Continuous Delivery



## Continuous Deployment



# Beneficios CD



Aplicaciones listas  
para despegar



Detección temprana  
de defectos



Eliminar el error  
humano



Rollbacks rápidos



Integrar QA en el  
proceso de desarrollo

# Estrategias de Gestión de Ramas en Git

# Tipos de ramas



**Ramas principales (main):**  
mantiene  
el histórico de  
publicaciones oficial



**Feature branches:**  
desarrollo de nuevas  
funcionalidades



**Release branches:**  
preparación de versiones  
estables para despliegue



**Hotfix branches:**  
Corrección rápida de  
errores críticos en  
producción

# Beneficios de una estrategia de gestión de ramas

## Colaboración eficiente

Permite a múltiples desarrolladores trabajar en paralelo sin interferir en el progreso de los demás, lo que mejora la eficiencia del equipo.

## Mayor control y trazabilidad

Facilita el seguimiento de cambios, quién los realizó y cuándo, lo que proporciona una mejor comprensión del progreso del proyecto y una gestión más efectiva del código.

## Reducción de conflictos y errores

Al separar el trabajo en ramas independientes, se minimizan los conflictos al integrar cambios, lo que ayuda a reducir errores y tiempos de resolución.

## Despliegue rápido y seguro

Permite mantener ramas de release estables y realizar correcciones rápidas en ramas de revisión, lo que agiliza la entrega del software y reduce el riesgo de fallos en producción.

## Flexibilidad y adaptabilidad

Las estrategias de gestión de ramas ofrecen flexibilidad para adaptarse a las necesidades específicas del proyecto y del equipo, permitiendo un flujo de trabajo ágil y eficiente.

# Estrategias de gestión de ramas

- Feature Branch workflow
- GitFlow
- Environment Branches



## GitFlow



Es uno de los workflows más populares para trabajo en equipo



Está orientado a desarrollos incrementales



Contempla el desarrollo de características, resolución de errores y generación de nuevas versiones



Se distribuye, de manera oficial, un conjunto de herramientas para gestionar el flujo

# Ramas principales: main y develop

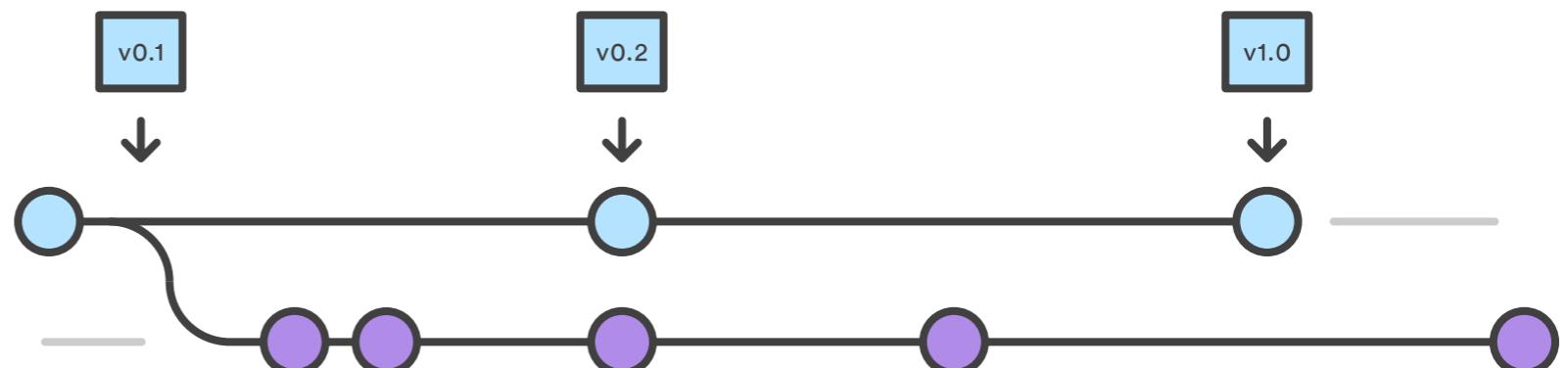
## Main

- Registro de versiones lanzadas en producción
- Solamente se añade código como parte del flujo



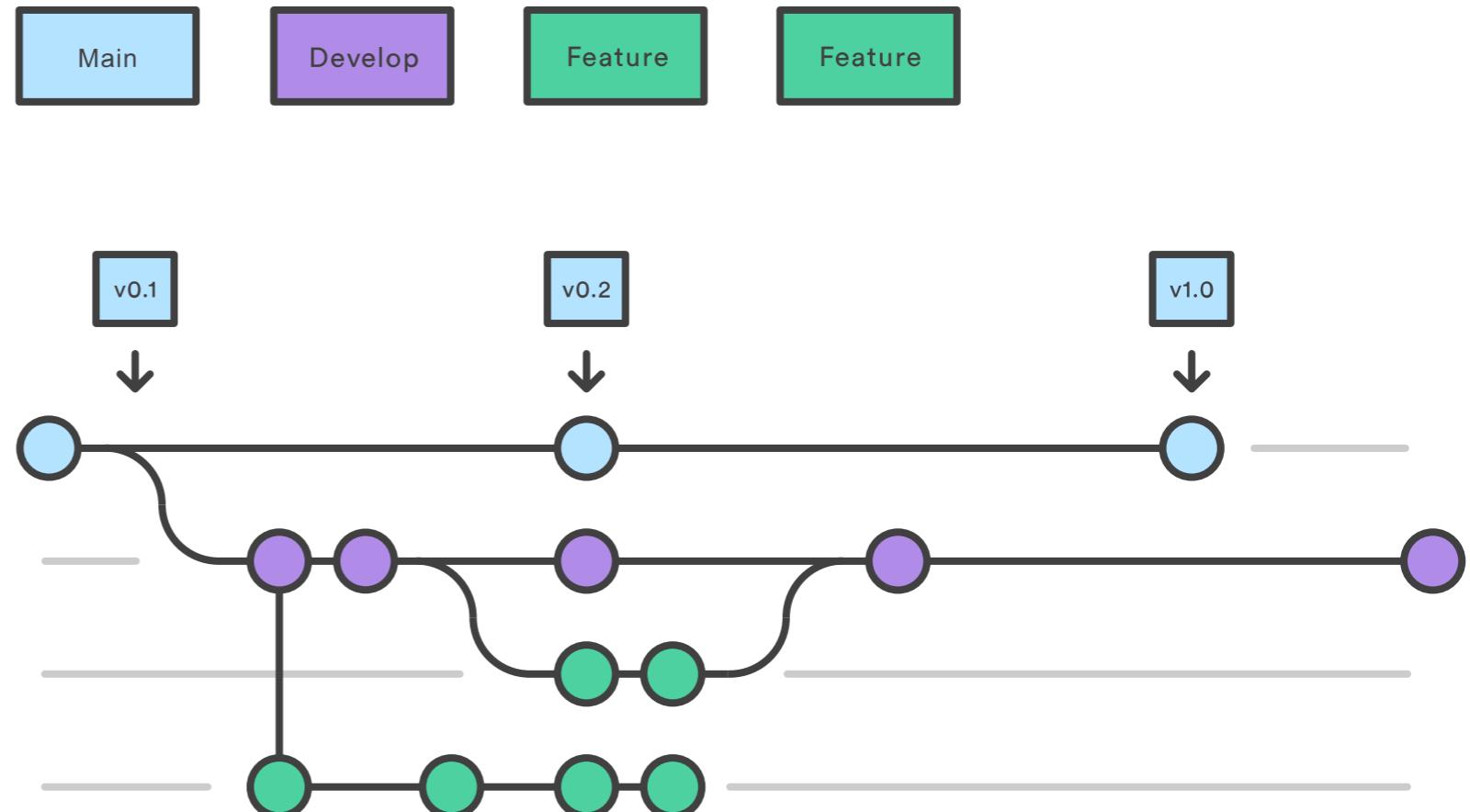
## Develop

- Rama de integración de características para la siguiente versión



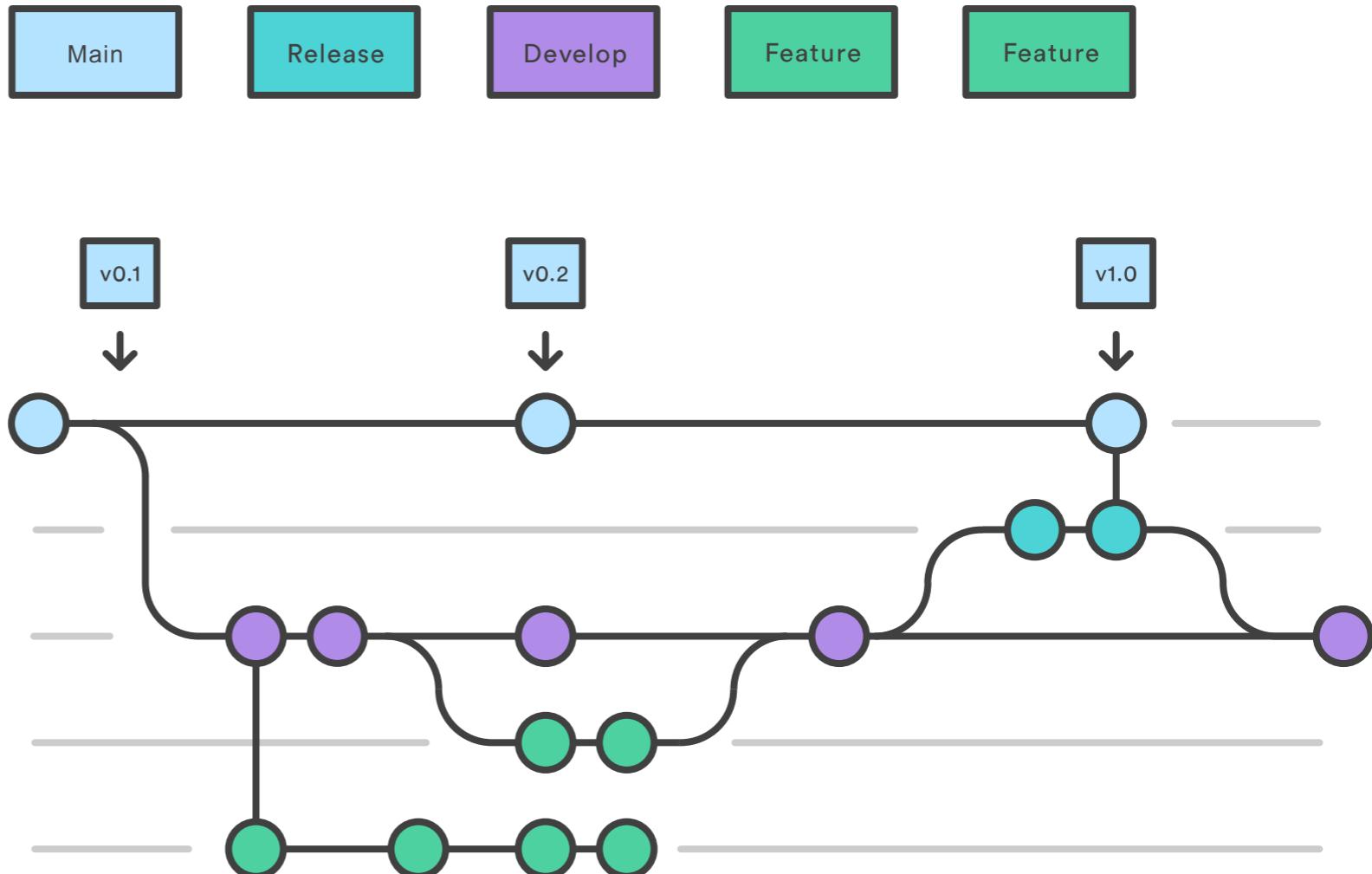
# Feature branches

- Features: cambios que se realizan en la aplicación y se incluirán en una futura versión
- Nomenclatura: feature/xxxx
- Siempre se crean a partir de develop
- Varios desarrolladores pueden trabajar sobre la misma feature
- Una vez finalizada, el resultado se integra en develop



# Release branches

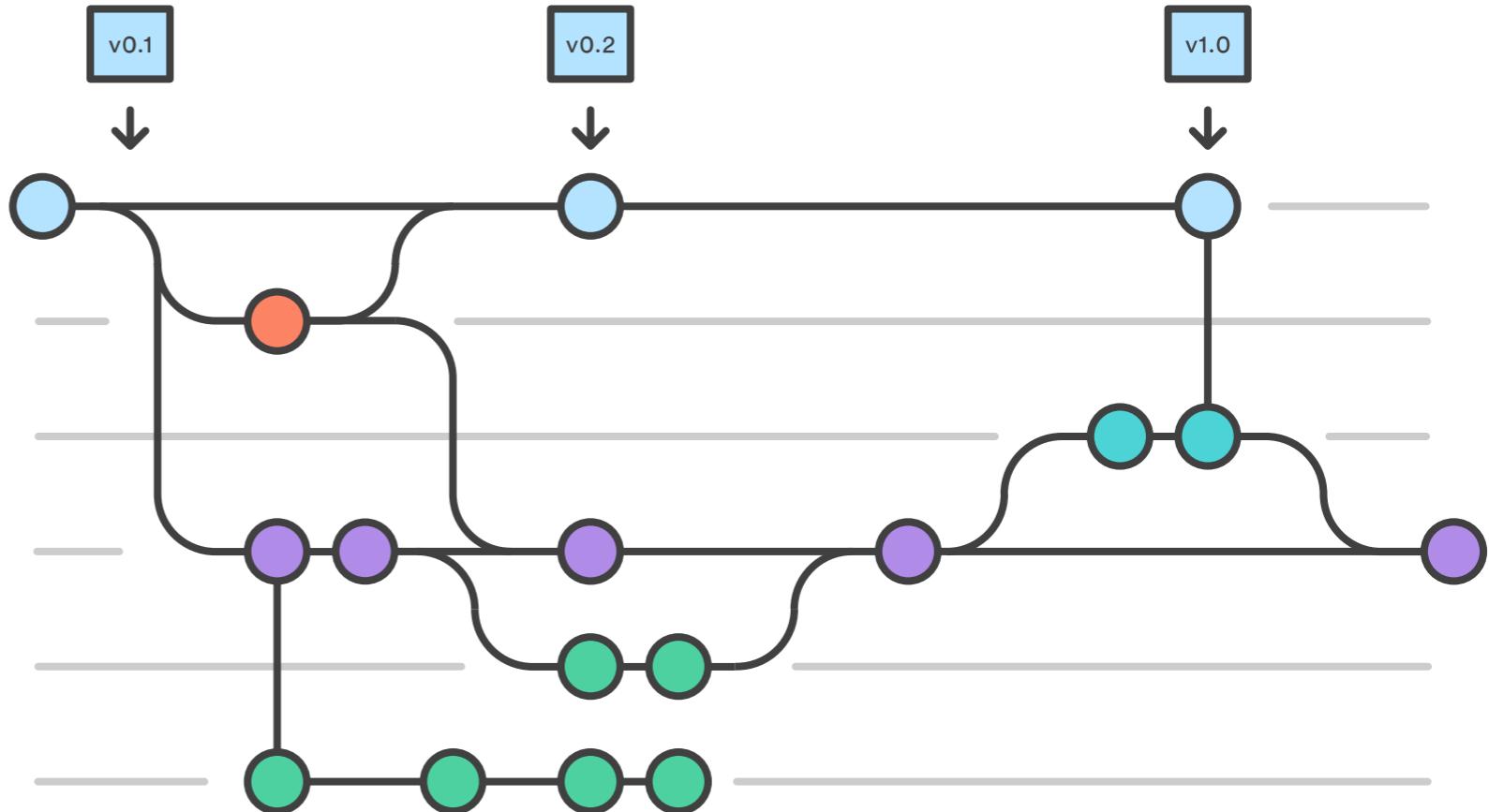
- El proceso de release implica la generación de una nueva versión de la aplicación
- Nomenclatura: release/xxxx
- Se crea una rama intermedia a partir de develop
- Durante la vida de la rama, se realizan los cambios necesarios
- Una vez finalizado, el resultado se reintegra de nuevo en la rama develop y en main
- Al finalizar una release, también se genera una etiqueta.



# Hotfix branches



- Un hotfix es la resolución de un bug crítico
- Nomenclatura: hotfix/xxxx
- Se crea una rama intermedia a partir de main, es decir, la última versión publicada.
- Durante la vida de la rama, se realizan los cambios necesarios
- Una vez finalizado, el resultado se reintegra de nuevo en la rama develop y en master
- Al finalizar un hotfix, también se genera una etiqueta.





## Comandos GitFlow

- Configurar repo para usar GitFlow
  - `git flow init`
- Feature branches
  - `git flow feature start <feature-name>`
  - `git flow feature finish <feature-name>`
- Release branches
  - `git flow release start <version-number>`
  - `git flow release finish <version-number>`
- Hotfix branches
  - `git flow hotfix start <hotfix-name>`
  - `git flow hotfix finish <hotfix-name>`

# CASO PRÁCTICO

# Integración eficiente de cambios en Git

# Buenas prácticas para la integración de cambios

## Realizar los desarrollos en ramas separadas

Manteniendo la rama principal alineada con las versiones finales y fomentando la colaboración eficiente.

## Utilizar Pull Request

El uso de PRs es permite gestionar las integraciones de código de una forma eficiente, facilitando la revisión del código, así como la configuración de condiciones para poder llevar a cabo la fusión.

## Proteger las ramas principales

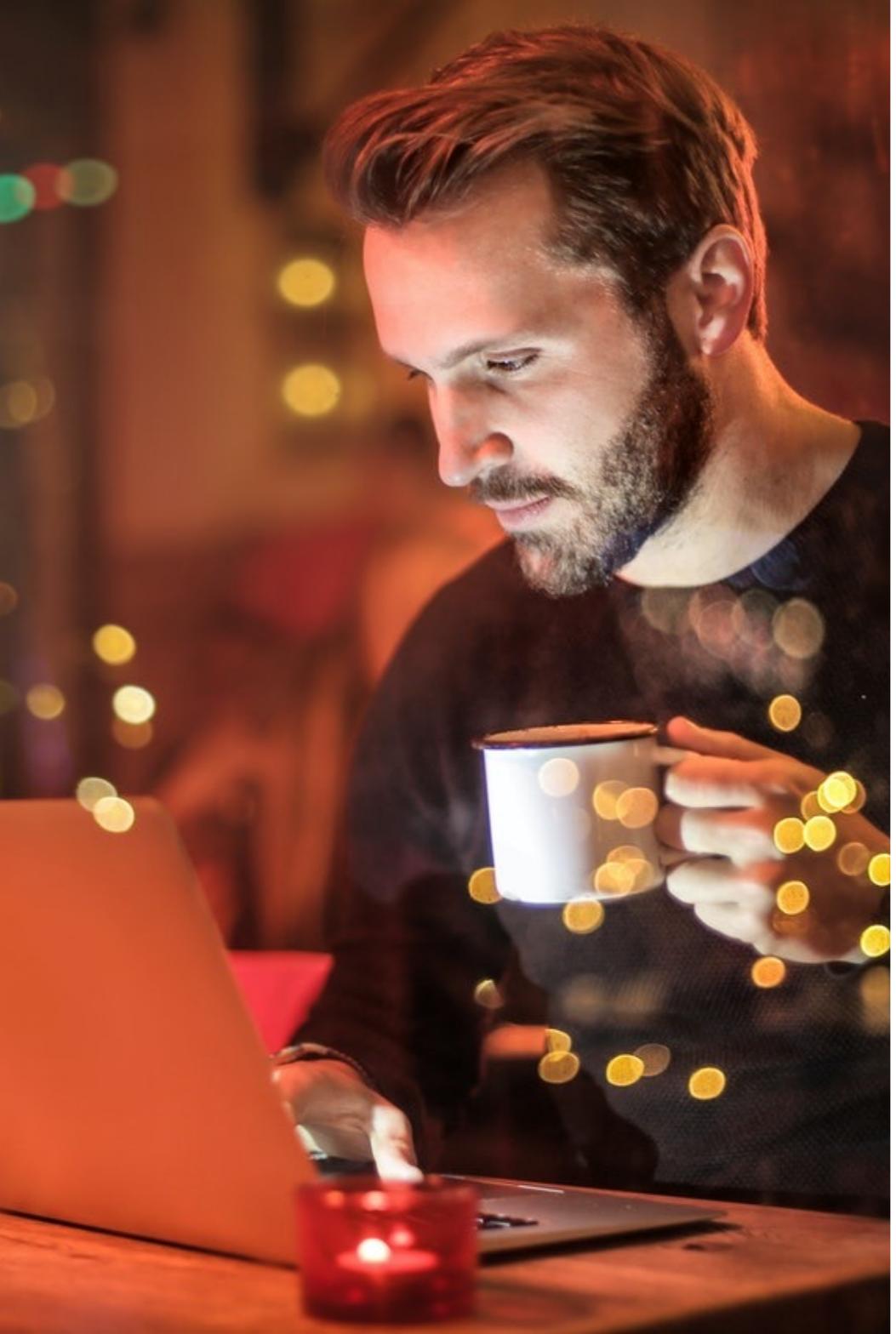
Permitiendo solamente cambios que vengan desde las Pull Request para garantizar su integridad.

## Fomentar la revisión de código

Tener el punto de vista de otros "colegas" mejora la calidad del código, así como la alineación de este a los estándares y buenas prácticas.

## Automatización de pruebas (CI)

Ejecutando automáticamente en cada PR pruebas que ayuden a identificar y corregir problemas de forma proactiva antes de que se fusionen los cambios.



## Protección de ramas

- Garantizar la estabilidad y la integridad del código en el repositorio.
- Evitar cambios no autorizados que puedan afectar la calidad del software.
- Limitar quién puede realizar cambios directamente en una rama determinada.
- Facilitar un flujo de trabajo colaborativo y seguro para el equipo de desarrollo.
- Obligar a que los cambios sean revisados antes de ser fusionados en una rama protegida.



## Pull Requests

- El propósito principal es solicitar la integración de los cambios en la rama principal
- Se crean a partir de una rama (feature o hotfix habitualmente)
- Proporcionan una visión clara de los cambios que se han llevado a cabo con respecto a la rama destino
- Facilitan el proceso de revisión de los cambios
- Junto con la protección de las ramas y la ejecución de Pipelines salvaguardan la consistencia del repositorio
- Ayudan a mantener el histórico de cambios ordenado



# Pull Requests en GitHub

## Code owners

- Permiten especificar los propietarios de cada parte del código
- Los propietarios serán los responsables de revisar y aprobar los cambios
- Permite asignar automáticamente revisores
- Se define mediante el fichero *CODEOWNERS*

## Pull Request template

- Son descripciones predefinidas para ayudar a la creación de una PR
- Establecen un formato y una estructura consistentes para la descripción del cambio
- Proporcionan pautas claras para la creación y revisión de la PR
- Se define mediante el fichero *PULL\_REQUEST\_TEMPLATE*

# CASO PRÁCTICO

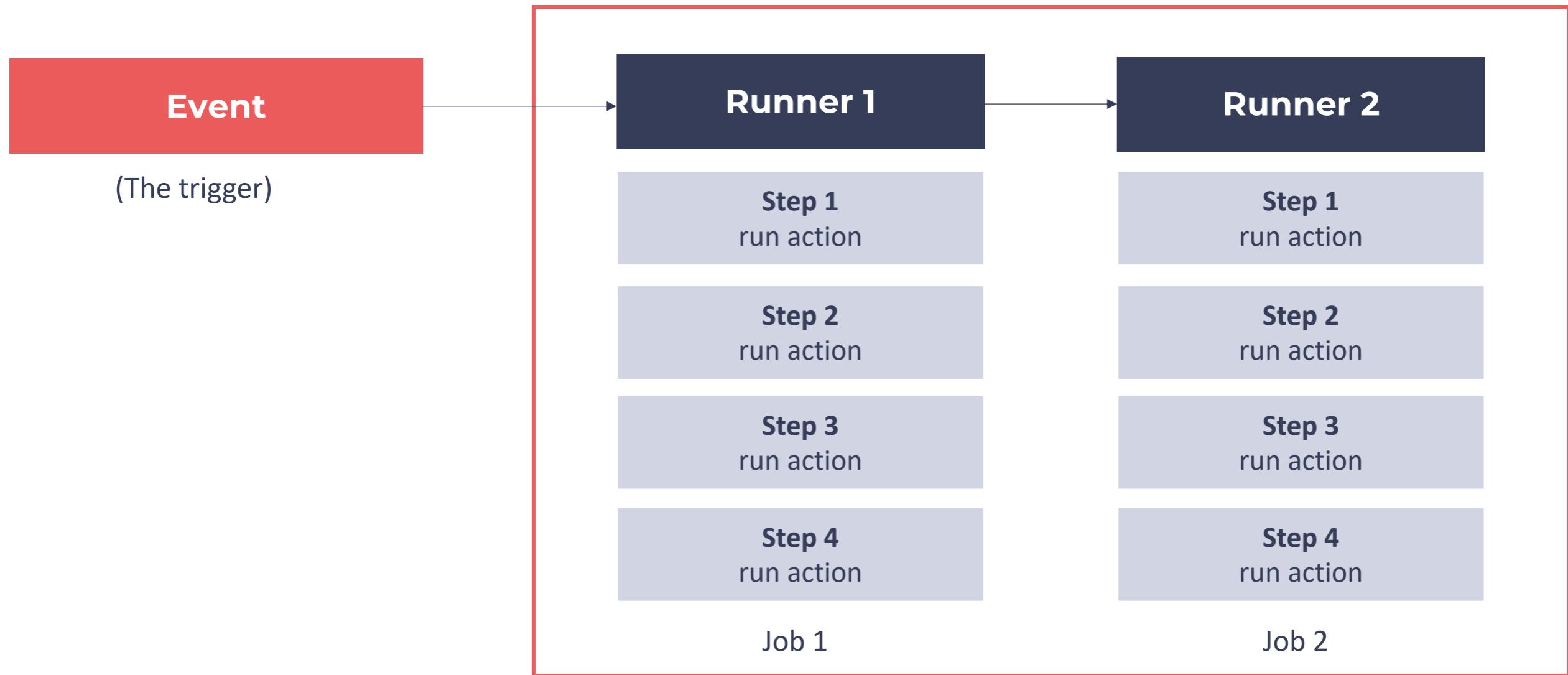
# Automatización de procesos con Github Actions



# Github Actions

- Automatización de tareas en el ciclo de vida del desarrollo de SW
- Se disparan ante ciertos eventos (cada vez que se hace un push, se actualiza una PR, ...)
- Componentes:
  - **Actions:** unidad mínima de trabajo. Pueden ser tareas predefinidas o bien acciones personalizadas
  - **Jobs:** unidad de trabajo individual. Compuestos una secuencia de actions que se ejecutan en el mismo runner. Configuración independiente
  - **Workflows:** se ejecutan cuando se produce un evento, están compuestos por uno o más Jobs que se puede ejecutar de forma secuencial o paralela
  - **Eventos:** desencadenantes del workflow. Pueden ser eventos de Github o eventos externos
  - **Runners:** Son las instancias donde se ejecutan las actions. Pueden ser SaaS o bien self-hosted

# Github Actions



*Github Workflow*

```
4   name: Node.js CI
5
6   on:
7     push:
8       branches: [ "main" ]
9     pull_request:
10    branches: [ "main" ]
11
12  jobs:
13    build:
14
15    runs-on: ubuntu-latest
16
17    strategy:
18      matrix:
19        node-version: [14.x, 16.x, 18.x]
20          # See supported Node.js release schedule at https://nodejs.org/en/about/releases/
21
22    steps:
23      - uses: actions/checkout@v3
24      - name: Use Node.js ${{ matrix.node-version }}
25        uses: actions/setup-node@v3
26        with:
27          node-version: ${{ matrix.node-version }}
28          cache: 'npm'
29      - run: npm ci
30      - run: npm run build --if-present
31      - run: npm test
```



Types

Apps

Actions

Categories

API management

Chat

Code quality

Code review

Continuous integration

Dependency management

Deployment

Deployment Protection Rules

IDEs

Learning

Localization

Mobile

Monitoring

Project management

Publishing

Recently added

 sort:popularity-desc

Sort: Most installed/starred

## Actions

An entirely new way to automate your development workflow.

22254 results for "sort:popularity-desc" filtered by Actions

### Actions



#### TruffleHog OSS

By trufflesecurity

Creator verified by GitHub

Scan Github Actions with TruffleHog

13.6k stars



#### yq - portable yaml processor

By mikefarah

create, read, update, delete, merge, validate and do more with yaml

10.5k stars



#### Gosec Security Checker

By securego

Runs the gosec security checker

7.4k stars



#### Metrics embed

By lowlighter

An infographics generator with 40+ plugins and 300+ options to display stats about your GitHub account

12.2k stars



#### Super-Linter

By super-linter

Creator verified by GitHub

It is a simple combination of various linters, written in bash, to help validate your source code

9.1k stars



#### OpenCommit — improve commits with AI 🤖

By di-sukharev

Replaces lame commit messages with meaningful AI-generated messages when you push to remote

5.2k stars

# CASO PRÁCTICO

# Práctica 1: Configuración de proyecto en Github

# Práctica 1: Configuración de proyecto en Github

El objetivo de este ejercicio es crear un nuevo repositorio en tu espacio personal de Github con una configuración que permita el trabajo colaborativo

1. Crea un nuevo repositorio en tu suscripción de Github
2. En el primer commit añade el código existente en <https://github.com/izertis/devops-workshop/archive/refs/tags/v1.zip> a la rama main
3. Configura Code owners (fichero CODEOWNERS) y Pull Request template
4. Configura la protección de ramas sobre main
  1. Requerir una PR
  2. Requerir aprobación de al menos un code owner
  3. Requerir que pasen todos los checks
  4. No permitir force push
5. Crea una feature branch (feature/practice1) a partir de main y hacer alguna modificación
6. Crea una PR para mergear los cambios a main



# **Práctica 2: Elaboración de pipeline usando Github Actions**

# Práctica 2: Pipeline usando Github Actions

El objetivo de este ejercicio es crear una pipeline de Github Actions para realizar CI de un repositorio

1. Utiliza el mismo repositorio que se ha creado para el ejercicio 1
2. Crea una feature (*feature/practice2*) a partir de la rama *main*
3. Crea un workflow de Github Actions que haga lo siguiente:
  1. Debe correr sobre Ubuntu
  2. Se ejecutará tanto en Pull Request a *main* como en push a esta rama
  3. Realizará las pruebas usando Node.js 20
  4. Ejecutará los comandos `npm ci` y `npm test`
  5. Usará caché para los `node_modules`
4. Añade otro job para ejecutar hacer el despliegue
  1. Ejecutará los comandos `npm ci` y `npm run deploy`
  2. Debe ejecutarse sobre Windows Server
  3. Se ejecutará tras la finalización del primer job y preferiblemente solo en push a la rama *main*

**Conoce más sobre nosotros:**



**izertis**  
Passion for Technology

# izertis

Passion for Technology

