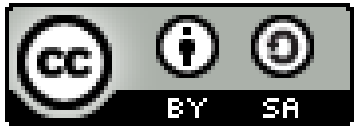
Herramientas para la Integración Continua

Asignatura: Integración Continua en el Desarrollo Ágil

Máster Universitario en Desarrollo Ágil de Software para la Web

José Ramón Hilera González



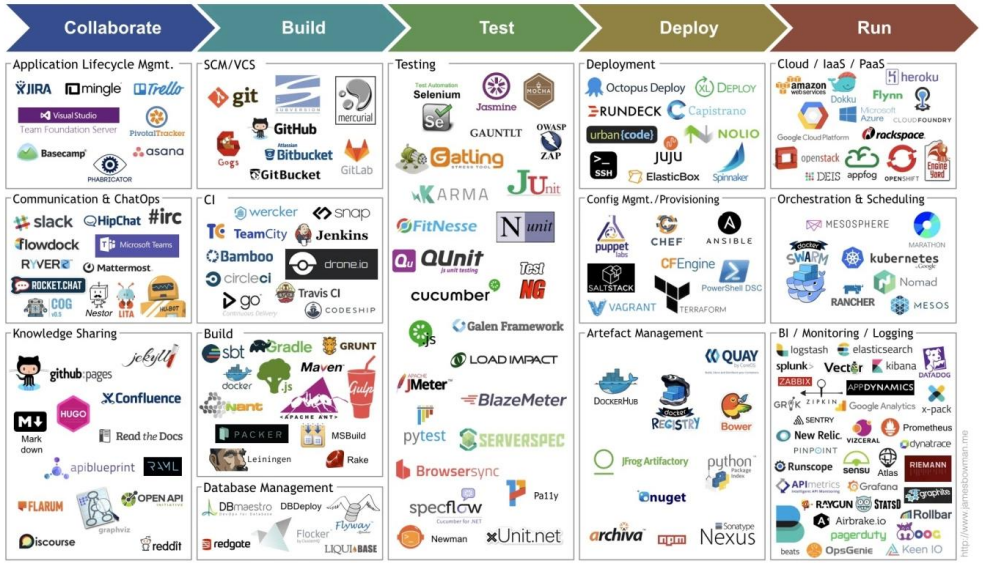
Contenido

1. Introducción
2. Categorías de herramientas para CI/CD
3. Comparación de herramientas
4. Conclusiones

1. Introducción

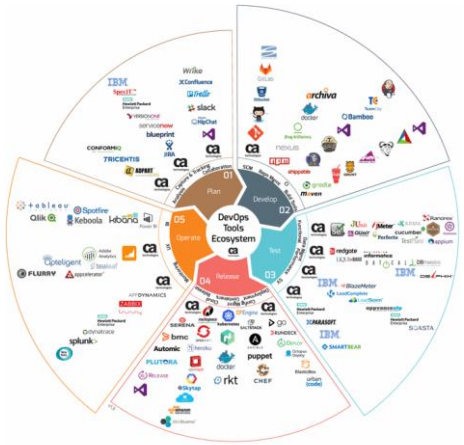
- Existe una gran cantidad de herramientas para utilizar en las diferentes etapas de un proyecto de desarrollo ágil con integración continua y entrega o despliegue continuo
- Se han publicado listas exhaustivas de herramientas, normalmente en el ámbito general de DevOps
  - Periodic Table of DevOps Tools
  - Herramientas de DevOps
  - Curated List of DevOps Tools
  - DevOps Tools Report
  - A Cambrian Explosion of DevOps Tools
  - Continuous delivery tool landscape
  - The Ultimate DevOps Tools Ecosystem

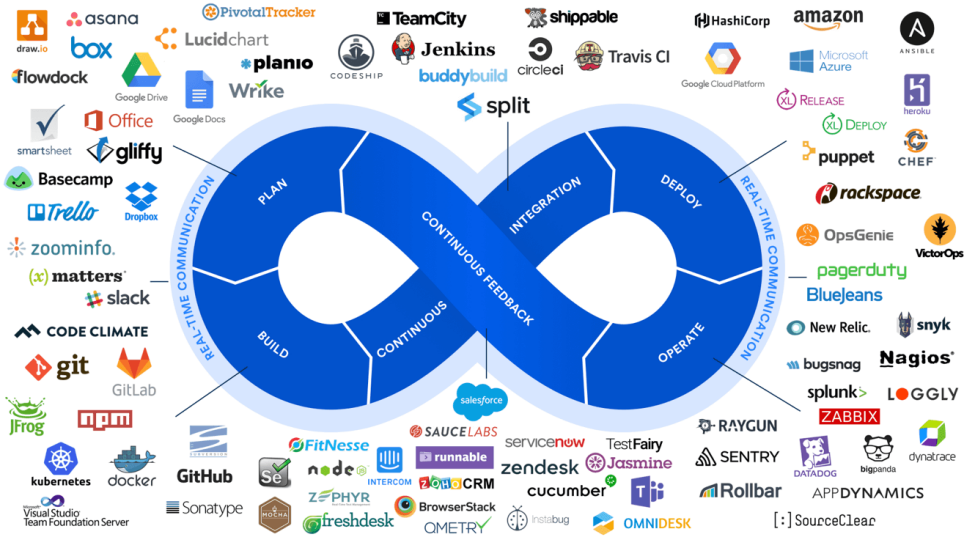
1. Introducción
Listas de herramientas



Fuente: James Bowman

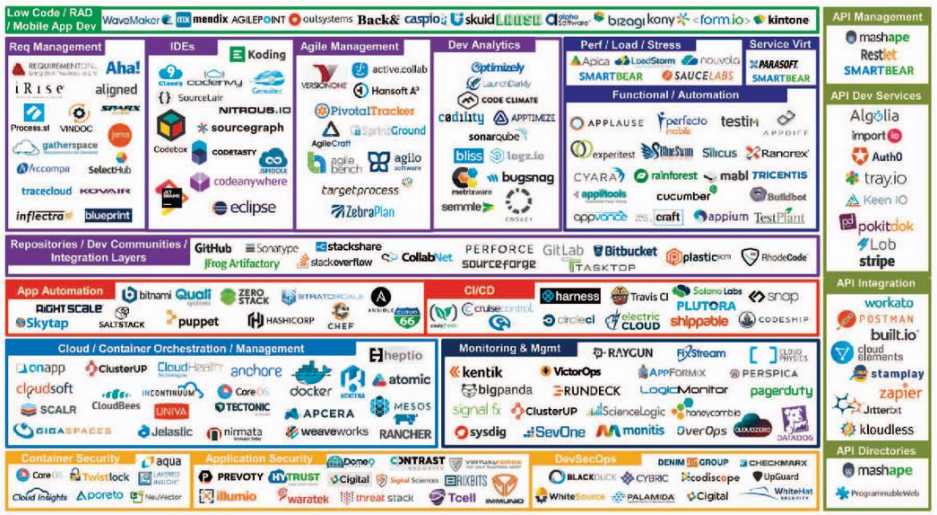
Fuente: Noga Cohen





Fuente: osolabs

Fuente: Mik Kersten



Fuente: digital.ai

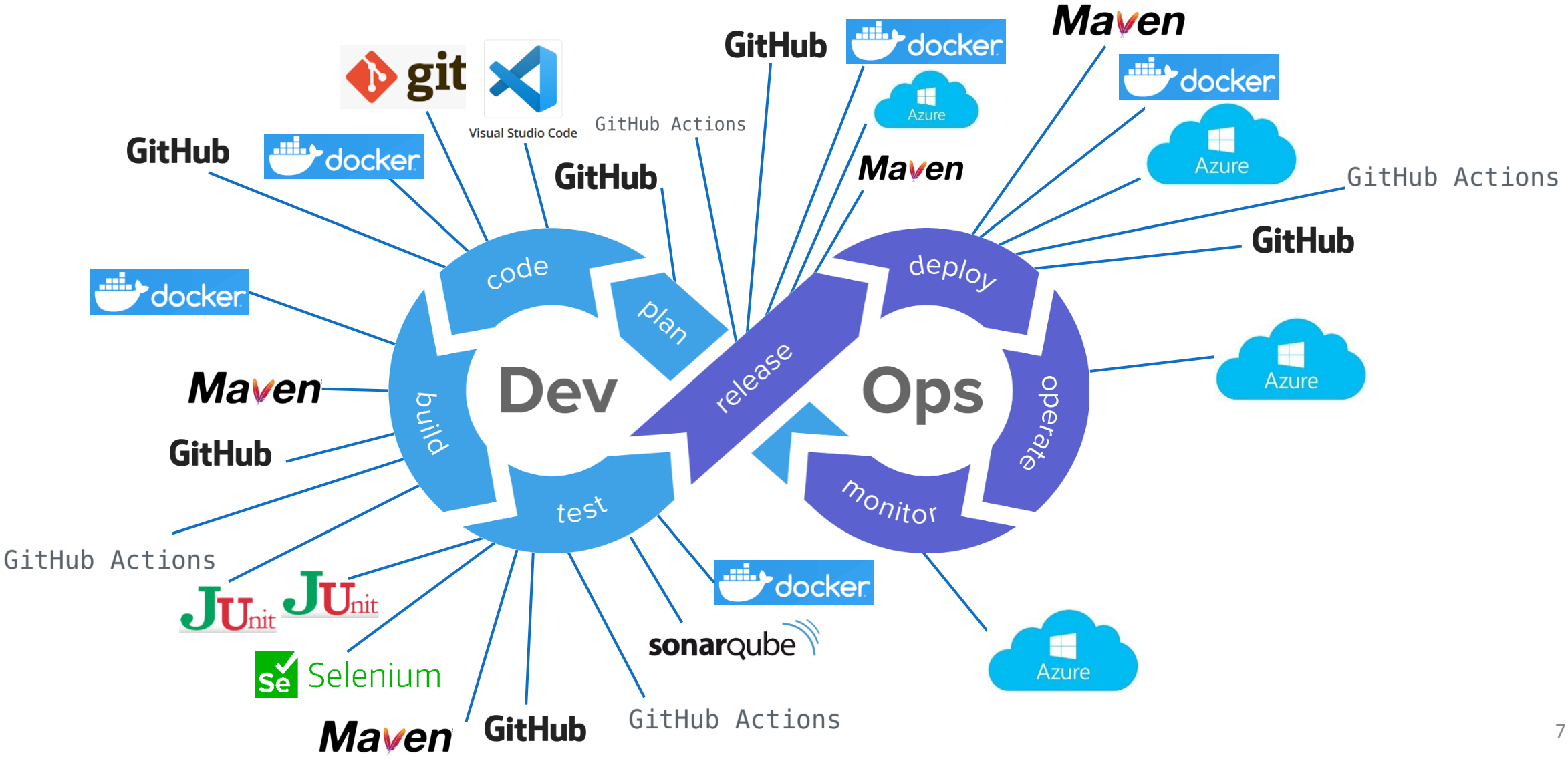
2. Categorías de herramientas para CI/CD

1. Editores de código (IDE)
2. Herramientas de construcción de software (software build)
3. Herramientas de gestión de versiones (versión control)
4. Gestores de repositorios compartidos
5. Gestores de proyectos ágiles
6. Servidores CI/CD
7. Herramientas de virtualización
8. Herramientas para pruebas unitarias
9. Herramientas para pruebas funcionales
10. Herramientas de análisis del código fuente
11. Herramientas para el despliegue
12. Otras

2. Categorías de herramientas para CI/CD Utilizadas en la asignatura (1/2)

Categoría	Editor (IDE)	Construcción	Gestión versiones	Repositorio compartido	Gestión ágil	Servidor CI/CD	Virtualización	Pruebas unitarias	Pruebas funcionales	Calidad código	Despliegue	Otras
Asignatura	VS Code	Maven	Git	GitHub	GitHub Projects	GitHub Actions	Docker	JUnit	Selenium	SonarQube	Azure	
Otras	IntelliJ Eclipse NetBeans …	Gradle …	Subversion …	GitLab Bitbucket …	GitLab Jira Trello …	GitLab CI Jenkins Circle CI …	LXC VirtualBox VMWare …	TestNG …	Cucumber Cypress …	Veracode …	AWS Google Cloud Heroku …	Repostorio artefactos (Nexus, …) Orquestación (Kubernetes, …) Monitorización (Datadog, …), Pruebas de rendimiento (JMeter, …)

2. Categorías de herramientas para CI/CD Utilizadas en la asignatura (2/2)



2.1 Editores de código (IDE: Integrated Development Enviroment)

- Cada desarrollador modifica el código fuente de la aplicación utilizando un editor de código (en su ordenador)
- No todos los desarrolladores de un mismo proyecto tienen que utilizar el mismo editor de código
- Un editor ampliamente utilizado es Visual Studio Code

- Otros editores: NetBeans, IntelliJ, Eclipse, Atom, Sublime, etc.
- Comparación

Eclipse vs IntelliJ IDEA vs

https://stackshare.io/stackups/eclipse-vs-intellij-idea-vs-visual-studio-code

Heads up! You are comparing apples to oranges. Here is a related, more direct comparison: Eclipse vs Qt Creator

UPDATED Nov 20, 2020

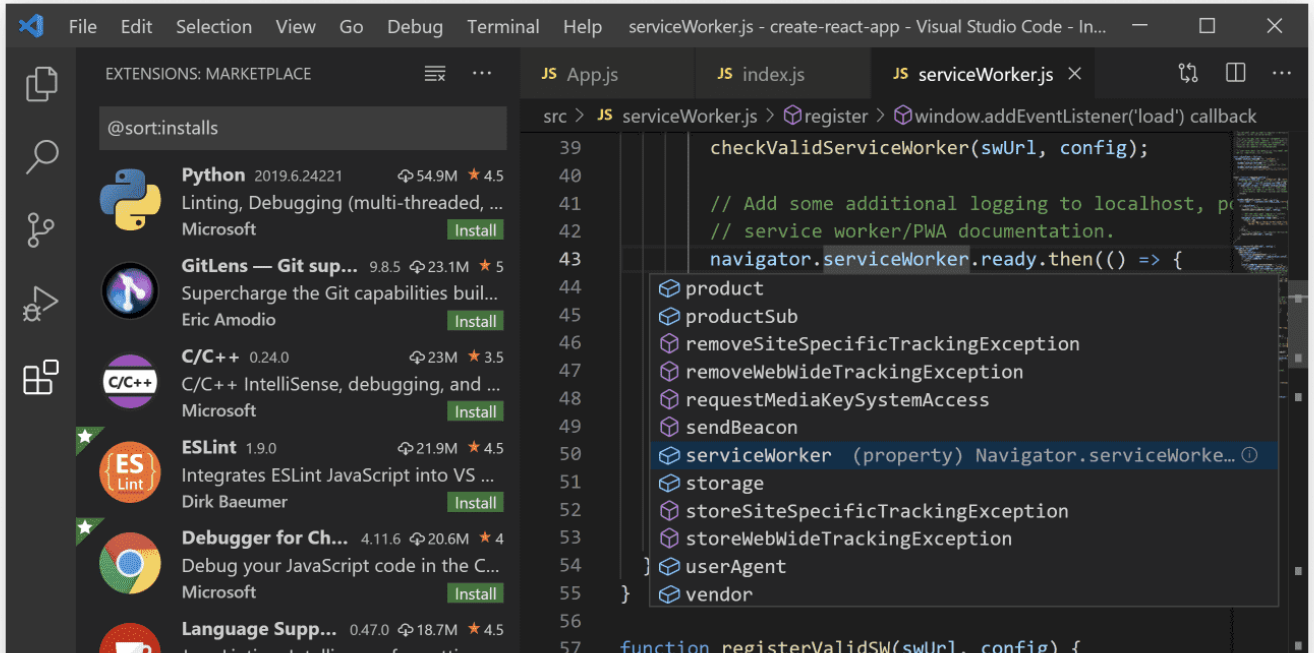
Eclipse vs IntelliJ IDEA vs Visual Studio Code

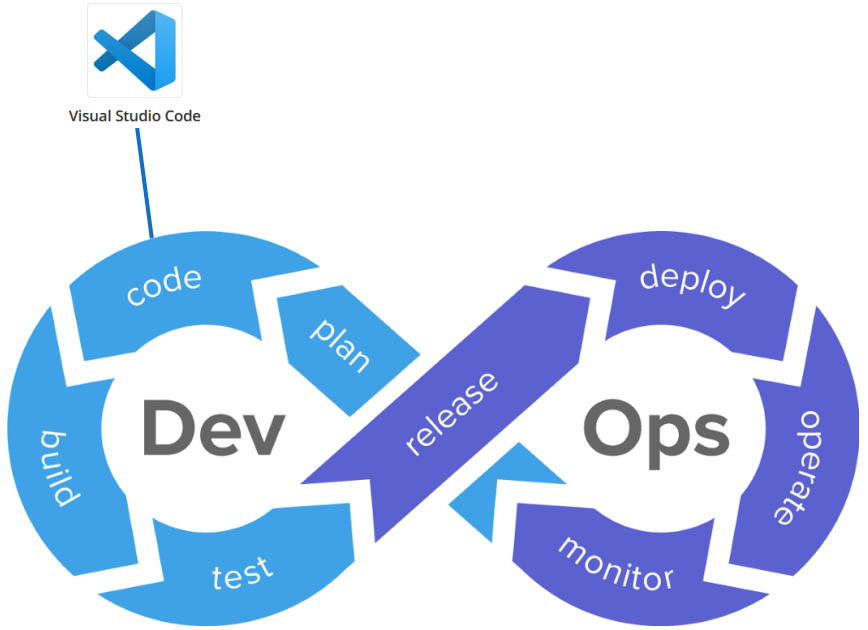
Need advice about which tool to choose?
Ask the StackShare community!

Get Advice

2.1 Editores de código
Cuándo usarlos (fases)

- **Codificación (code):** En esta etapa se crea el código fuente del software que se está desarrollando, con la ayuda de un editor o entorno de desarrollo integrado (IDE).





2.2 Herramientas de construcción de software (Software build)

- Son herramientas para automatizar la creación de software ejecutable a partir del código fuente
- Se utilizan desde un intérprete de comandos, y automatizan una variedad de tareas, como descargar dependencias, compilar y empaquetar código, ejecutar pruebas, desplegar ejecutables.
- Existen herramientas de construcción para uno o varios lenguajes de programación.
- Una herramienta ampliamente utilizada para Java es Maven

- Otras: Gradle, SBT, npm, Gulp, etc.
- Comparación

Eclipse vs IntelliJ IDEA vs | Gradle | Gradle vs Maven

https://gradle.org/maven-vs-gradle/

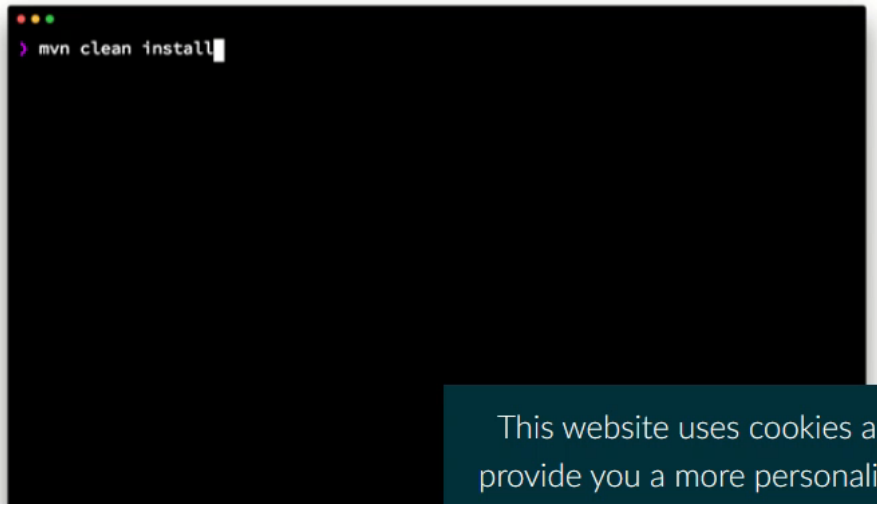
Have you checked out the Gradle build tool blog?

Gradle Build Tool

Docs About Training News Services Gradle Enterprise

Gradle vs Maven Comparison

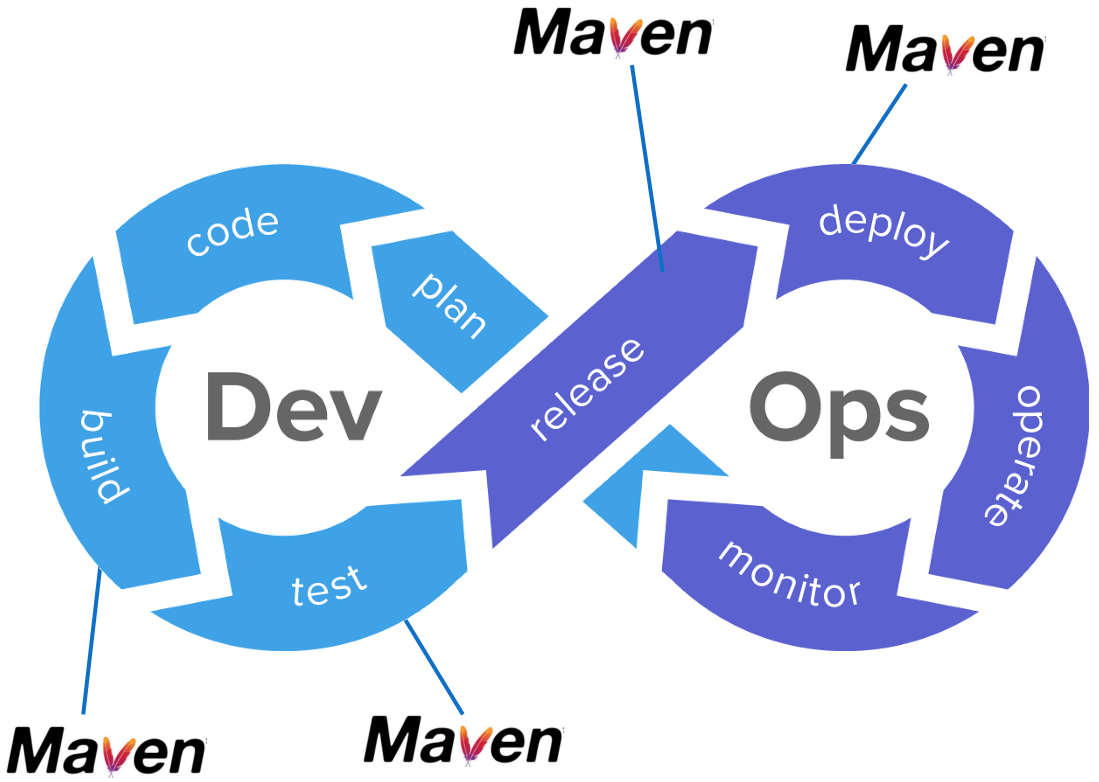
The following is a summary of the major differences between Gradle and Apache Maven: flexibility, performance, user experience, and dependency management. It is not meant to be exhaustive, but you can check the Gradle feature list and Gradle vs Maven performance comparison to learn more.



This website uses cookies and other technology to provide you a more personalized experience. Accept

2.2 Herramientas de construcción de software Cuándo usarlas (fases)

- **Construcción (build):** Por ejemplo, para compilar y empaquetar una aplicación Java con Maven, se puede hacer con el comando:
  - `mvn package`
- **Pruebas (test):** También se pueden ejecutar pruebas con Maven usando el comando:
  - `mvn test`
- **Liberación (release):** Si existe esta fase, se puede desplegar a un entorno de pre-producción, por ejemplo en Heroku, con el comando:
  - `mvn heroku:deploy -Dheroku.appName=ejemplo-pre`
- **Despliegue (deploy):** En este caso a producción:
  - `mvn heroku:deploy -Dheroku.appName=ejemplo-prod`

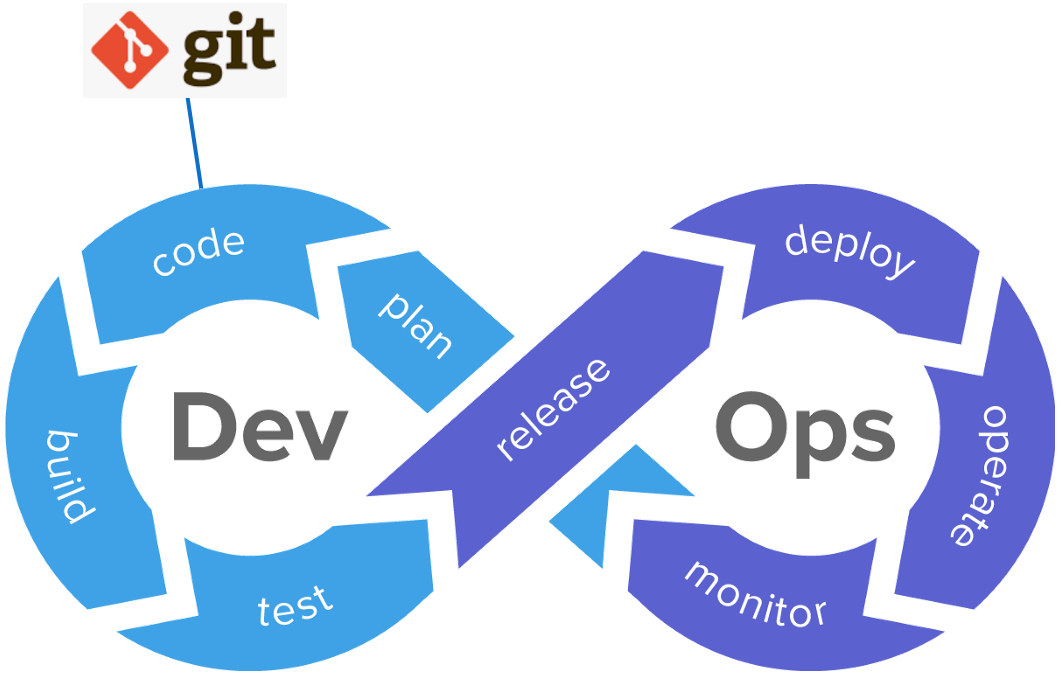


2.3 Herramientas de gestión de versiones (Version Control Systems - VCS)

- Los sistemas de control de versiones (VCS) son herramientas que permiten tomar una instantánea (snapshot o commit) de un conjunto de ficheros (repositorio) en un momento en el tiempo.
- Un commit tiene un identificador único con el que en cualquier momento se puede recuperar ese estado del repositorio (backtrack).
- Estas herramientas permiten crear ramas (branchs) a partir de la rama principal (master) del repositorio, para que cada desarrollador pueda trabajar independientemente, modificando el código fuente, y después mezclar (merge) sus cambios en la rama master, donde está el código fuente final del proyecto. Así no tiene que esperar a otros miembros del equipo.
- Una herramienta ampliamente utilizada es Git
- Otras: Subversion, Mercurial, etc.
- Comparación

2.3 Herramientas de gestión de versiones Cuándo usarlas (fases)

- **Codificación (code):** Por ejemplo, cuando un desarrollador termina una modificación en el código fuente, con el siguiente comando de git haría un commit del repositorio:
  - `git commit -m "commit inicial"`
- Realmente la herramienta se utiliza indirectamente en otras fases, como test, release y deploy, porque los servidores que se usan en estas fases suelen utilizar git para recibir los archivos del software desarrollado que hay que instalar en ellos.

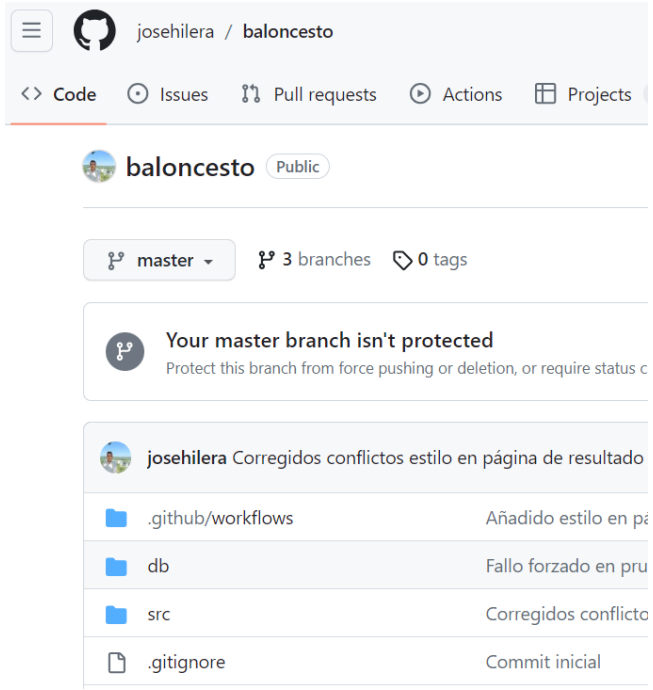


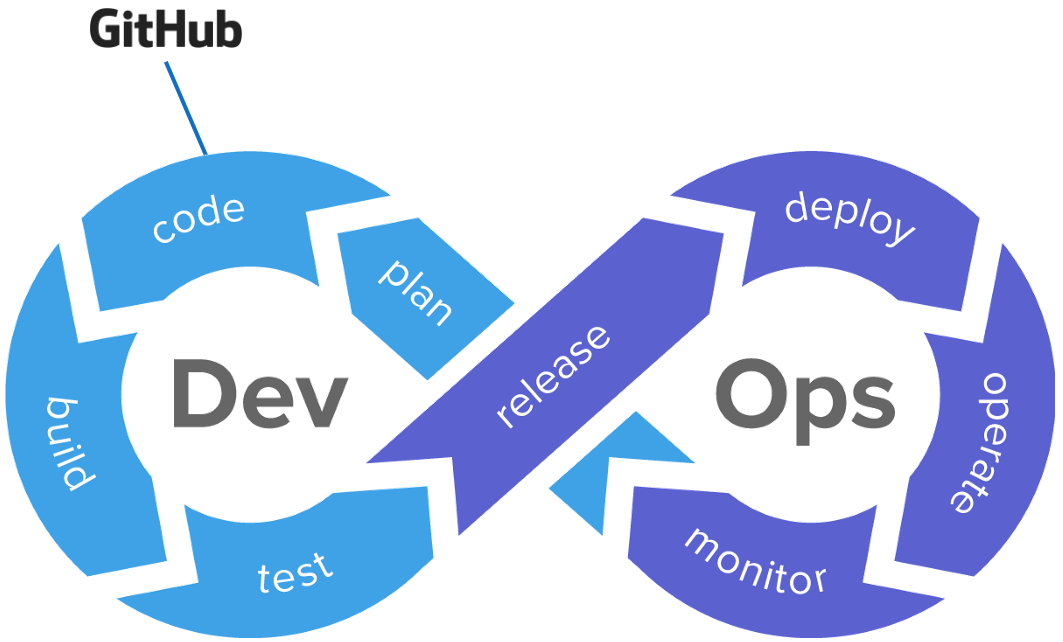
2.4 Gestores de repositorios compartidos

- Las herramientas de gestión de repositorios compartidos se basan en una tecnología de gestión de versiones, como Git, y permiten crear y gestionar repositorios en un servidor
- Se utilizan para compartir el código fuente de un proyecto con todos los desarrolladores del mismo
- Una herramienta ampliamente utilizada es GitHub 
- Otras: GitLab, Bitbucket, etc.
- Comparación

2.4 Gestores de repositorios compartidos Cuándo usarlas (fases)

- **Codificación (code):** Los desarrolladores deben “subir” al repositorio compartido los cambios que hacen en el código fuente.
  - `git push origin`



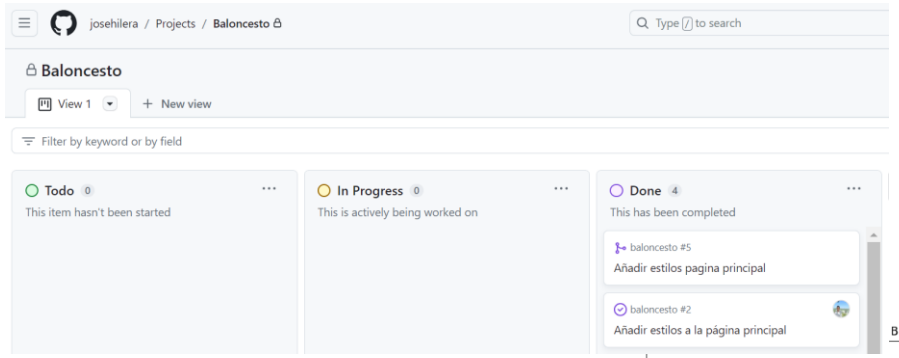


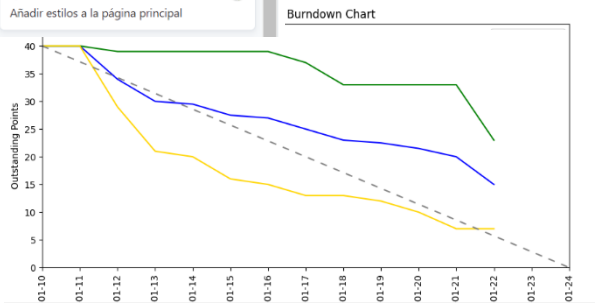
2.5 Gestores de proyectos ágiles

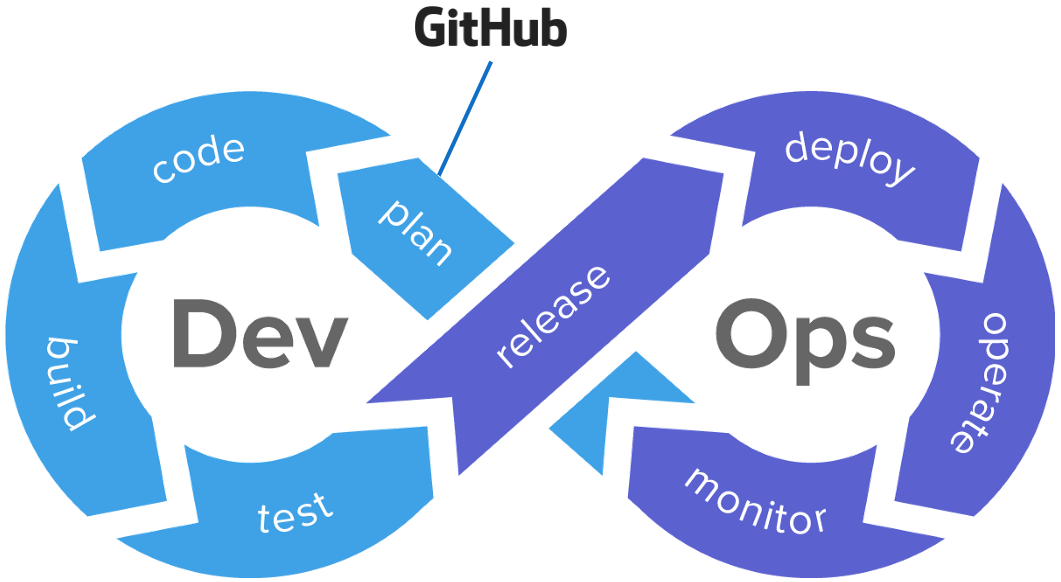
- Son herramientas que permiten planificar, organizar y hacer seguimiento de las tareas de un proyecto.
- Se suelen denominar ALM (Application Lifecycle Management tools)
- Cuando están enfocadas a proyectos ágiles (Agile-focused ALM) manejan historias de usuario, backlogs, tableros Kanban, diagramas burndown, etc.
- Una posible herramienta es GitHub Projects (es muy básica, pero se puede integrar con herramientas externas disponibles en repositorios de GitHub)
- Otras: GitLab, Jira, Trello, etc.
- Comparación

2.5 Gestores de proyectos ágiles
Cuándo usarlos (fases)

- **Planificación (plan):** Herramientas como GitHub Projects ofrecen la posibilidad de gestión de proyectos ágiles, mediante tableros Kanban, historias de usuario (issues), backlogs, sprints (milestones).
- En el caso de GitHub, con ayuda de herramientas externas se pueden generar diagramas burndown para un proyecto.







2.6 Servidores CI/CD

- Son servidores de integración continua (CI) y de entrega o despliegue continuo (CD)
- Permiten definir un flujo de trabajos automatizados (workflow o pipeline) para un proyecto
- Cuando detectan un cambio en el repositorio de código compartido, lanzan la ejecución de los trabajos definidos en el workflow del proyecto
- Tienen asociado un ejecutor de trabajos (runner)
- Una posible herramienta es GitHub Actions, que se puede combinar con el software GitHub Actions Runner para crear ejecutores de trabajos fuera de github.com
- Otras: GitLab CI, Jenkins, Travis, CircleCI, etc.
- Comparación

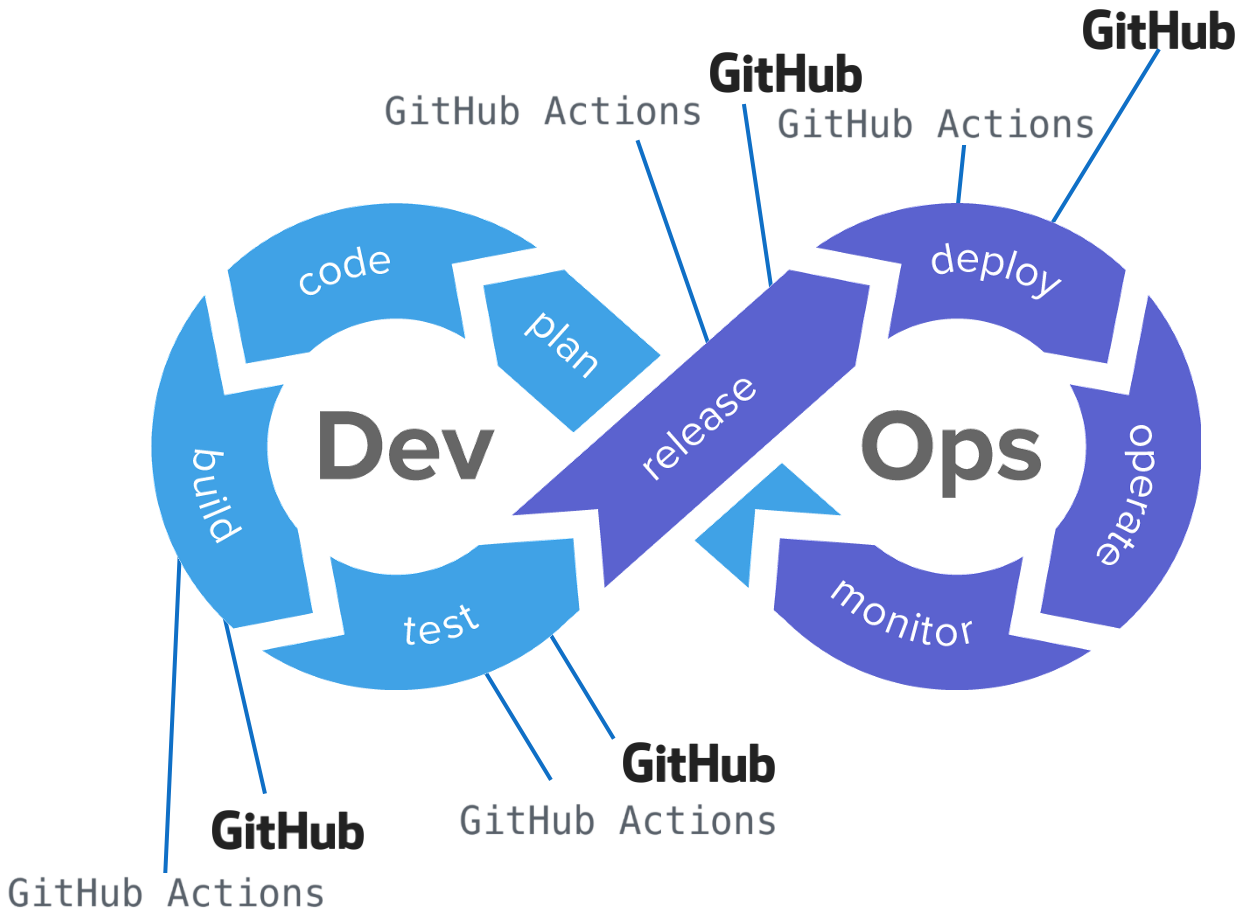
GitHub Actions

2.6 Servidores CI/CD
Cuándo usarlos (fases)

- **Construcción (build):** Por ejemplo, se puede definir un trabajo para compilar el código del repositorio y empaquetarlo. En GitHub Actions sería:

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: descargar repositorio
        uses: actions/checkout@v3
      - name: empaquetar
        run: mvn package -DskipTests=true
```

- **Pruebas (test):** Por ejemplo, se pueden definir pruebas funcionales
- **Liberación (release):** Si existe esta fase, se puede definir que el runner se encargue de desplegar a un entorno de pre-producción.
- **Despliegue (deploy):** Por ejemplo desplegar a Azure de forma automática



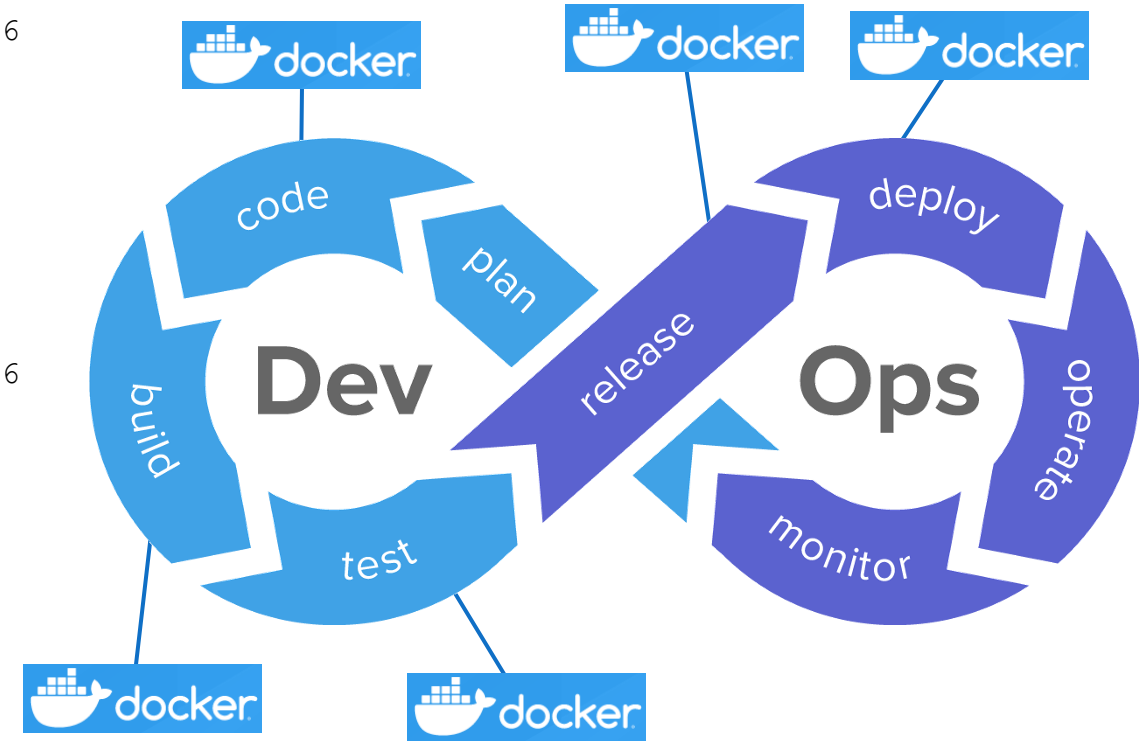
2.7 Herramientas de virtualización

- Son herramientas para simular una máquina con un sistema operativo y aplicaciones preinstaladas.
- Pueden ser contenedores o máquinas virtuales.
- En integración continua son útiles para simular entornos de desarrollo y pre-producción similares al entorno de producción real, y hacer pruebas en ellos.
- Una herramienta muy utilizada es Docker

- Otras: LXC, Virtual Box, VMWare, etc.
- Comparación

2.7 Herramientas de virtualización Cuándo usarlas (fases)

- **Codificación (code):** Simular en el ordenador de cada desarrollador un entorno para pruebas como máquina con sistema operativo Ubuntu con servidor web en puerto 8080 y base datos en puerto 3306. Ejemplo:
  - `docker run -it --name mi-entorno -p 8080:8080 -p 3306:3306 josehilera/ubuntu-ci`
- **Construcción (build):** Simular en el servidor de integración continua una máquina para compilar el código fuente
  - `docker run -it --name compilador josehilera/ubuntu-ci`
- **Pruebas (test):** Simular un entorno de desarrollo para pruebas. Ejemplo:
  - `docker run -it --name desarrollo -p 8080:8080 -p 3306:3306 josehilera/ubuntu-ci`
- **Liberación (release):** Simular entorno de pre-producción para pruebas.
  - `docker run -it --name pre-produccion -p 8080:8080 -p 3306:3306 josehilera/ubuntu-ci`
- **Despliegue (deploy):** Para desplegar a producción desde una máquina virtualizada, o cuando el entorno de producción está virtualizado, en cuyo caso, también se usaría en las fases Operación (opérate) y Monitorización (monitor).



2.8 Herramientas para pruebas unitarias

- Son herramientas para automatizar la comprobación de módulos aislados del código fuente.
- Normalmente son librerías, y el código de las pruebas programadas está integrado en el mismo repositorio del código fuente del proyecto, en un directorio específico.
- En el caso de Java, para cada clase de la aplicación (ej. clase “Cliente”) se suele programar una clase de prueba (ej. Clase “ClienteTest”) para comprobar el correcto funcionamiento de los métodos de esa clase.
- Una herramienta muy utilizada para Java es JUnit

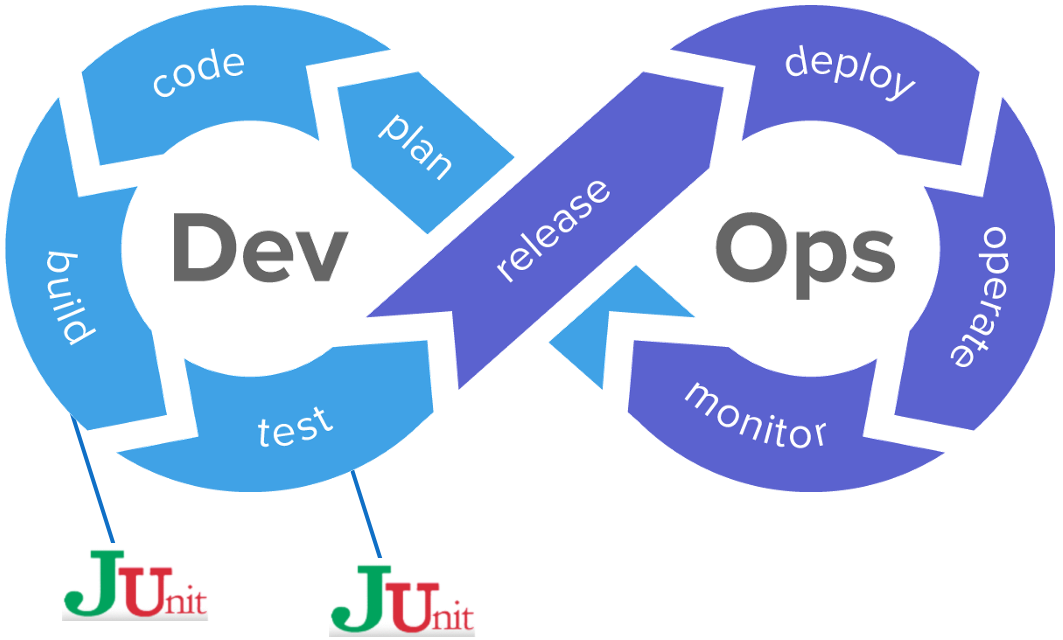
- Otras: TestNG (Java), PHPUnit (PHP), PyUnit (Python), etc.
- Comparación

2.8 Herramientas para pruebas unitarias Cuándo usarlas (fases)

- **Construcción (build):** Las pruebas unitarias se pueden ejecutar antes de compilar todo el código del proyecto, por lo que se pueden llevar a cabo en la fase de construcción o en la de pruebas. Si es en construcción, en GitHub Actions se definiría de la siguiente forma, donde el comando de Maven ejecutaría las pruebas unitarias programadas con JUnit que haya en la carpeta src/test/java del repositorio del proyecto (archivos con nombres terminados en “Test”):

```
- name: pruebas-unitarias
  run: mvn package
```

- **Pruebas (test):** En la fase de pruebas se suelen hacer pruebas funcionales, con herramientas como Selenium, que a su vez necesitan librerías de pruebas unitarias, como JUnit.



2.9 Herramientas para pruebas funcionales

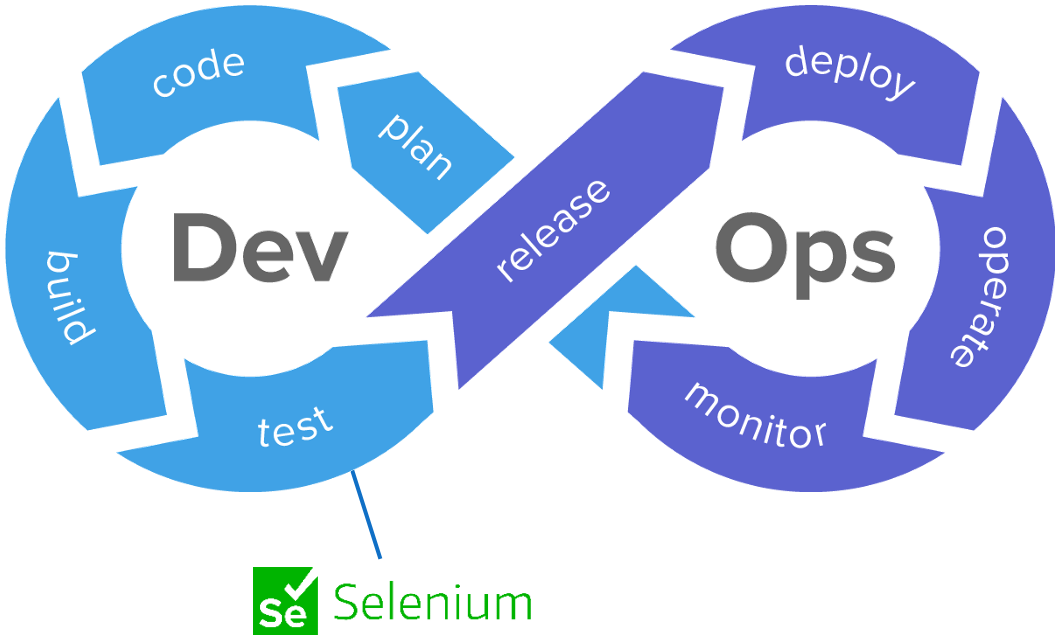
- Son herramientas para automatizar la comprobación del funcionamiento de toda la aplicación, por lo que también se denominan pruebas end-to-end (“de punta a punta”), con las siglas E2E.
- Se trata de emular la experiencia del usuario en la utilización de la aplicación.
- Normalmente se componen de librerías para simular por código el comportamiento de navegadores web.
- El código de las pruebas programadas está integrado en el mismo repositorio del código fuente del proyecto, en un directorio específico para pruebas.
- Una herramienta muy utilizada es Selenium

- Otras: Cucumber, Cypress, etc.
- Comparación

2.9 Herramientas para pruebas funcionales Cuándo usarlas (fases)

- **Pruebas (test):** Si se utiliza GitHub Actions, definiría de la siguiente forma, donde el comando de Maven ejecutaría las pruebas de integración programadas con Selenium y JUnit que haya en la carpeta src/test/java del repositorio del proyecto (archivos con nombres terminados en “IT”):

```
- name: pruebas-funcionales
  run:
    mvn failsafe:integration-test
        failsafe:verify
```



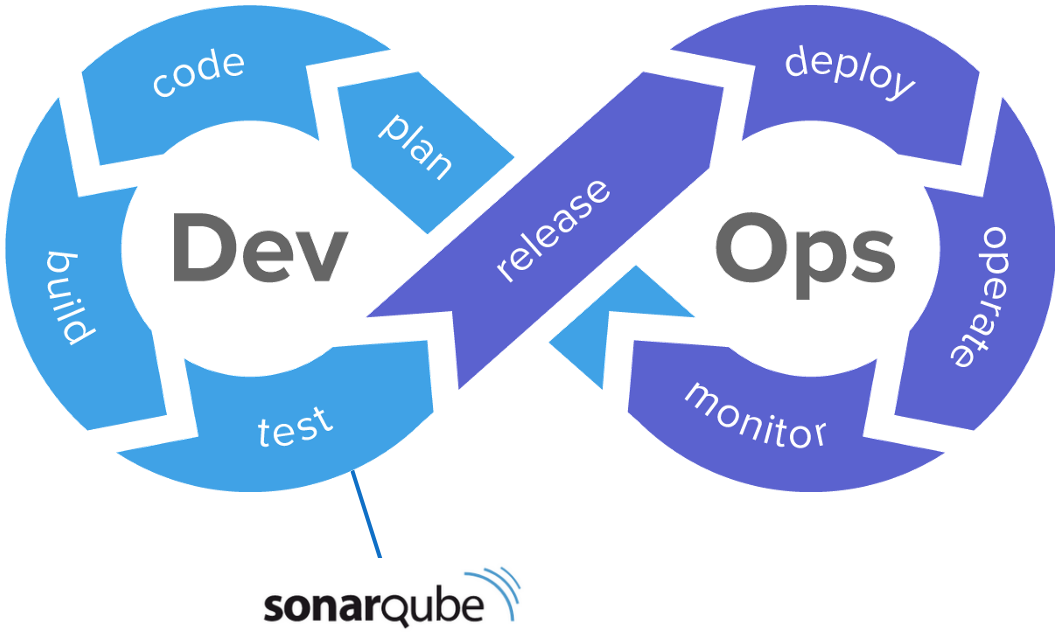
2.10 Herramientas de análisis del código fuente

- Son herramientas utilizadas normalmente en el ámbito del aseguramiento de la calidad (QA: Quality Assurance), para encontrar defectos o malas prácticas en el código fuente.
- También se denominan herramientas SAST (Static Application Security Testing), pues ayudan a detectar posibles problemas de seguridad.
- Una herramienta muy utilizada es SonarQube
- Otras: Veracode, Codacy, PMD, etc.
- Comparación

2.10 Herramientas de análisis del código fuente Cuándo usarlas (fases)

- **Pruebas (test):** Esta fase se podría dividir en dos: una para pruebas funcionales y otra para control de calidad (QA). Si se utiliza GitHub Actions, se definiría de la siguiente forma el análisis automático de todo el código fuente de la aplicación usando SonarQube instalado en el servidor localhost:9000:

```
- name: calidad-codigo:
  run:
    mvn sonar:sonar
        -Dsonar.host.url=http://localhost:9000
        -Dsonar.qualitygate.wait=true
```



2.11 Herramientas para el despliegue

- Son herramientas para desplegar una aplicación en un entorno de producción o pre-producción.
- Actualmente se suelen utilizar plataformas IaaS (Infrastructure as a Service) en la nube, que ofrecen herramientas para automatizar el despliegue de las aplicaciones mediante comandos o mediante plugins para herramientas de construcción.
- Una posible herramienta es Azure 

- Otras: AWS, Google Cloud, Heroku, etc.
- Comparación

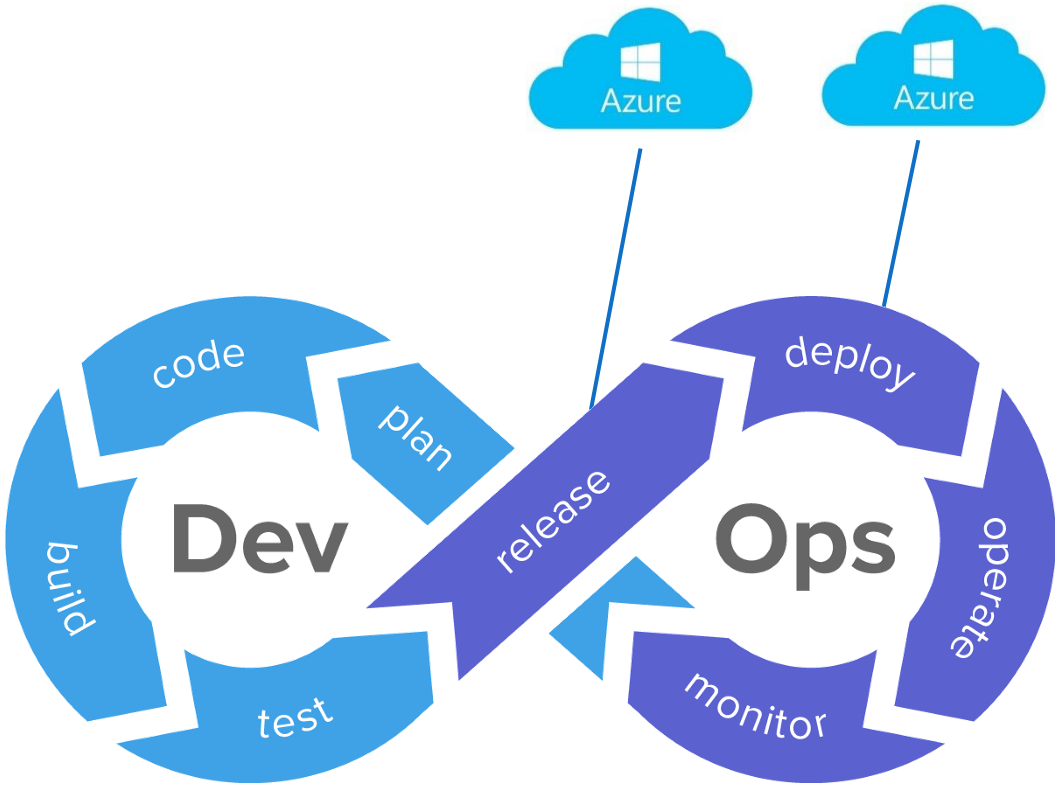
2.11 Herramientas para el despliegue Cuándo usarlas (fases)

- **Liberación (release):** Si existe, en esta fase se podría desplegar la aplicación en un entorno de pre-producción ubicado en una plataforma en la nube, como Azure. Si se usa GitHub Actions como servidor de integración, se podría definir como:

```
- name: Desplegar en Azure
  uses: Azure/webapps-deploy@v2
  with:
    app-name: baloncesto-hilera
    publish-profile: ${{...}}
    package: target/*.war
```

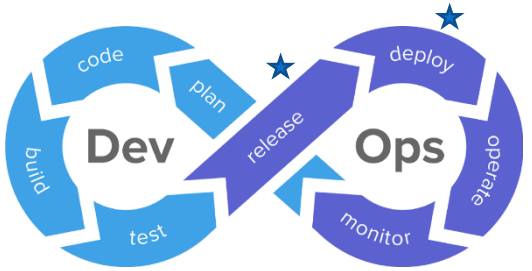
- **Despliegue (deploy):** En esta fase, si se aplica entrega continua en lugar de despliegue continuo, se puede añadir la condición de que una persona debe confirmar que se realice el despliegue:

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



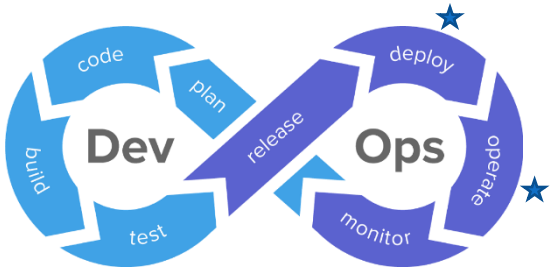
2.12 Otras herramientas
Repositorios de artefactos

- Son herramientas para almacenar artefactos o paquetes de código compilado (librerías, ejecutables) en servidores para que puedan ser descargados para su instalación por personas o, de forma automática, por herramientas de construcción (Maven, Gradle, etc.) al detectar la existencia de dependencias con dichos artefactos.
- Fases del ciclo de vida DevOps: Se suelen en las fases de Liberación (release) y Despliegue (deploy), cuando el resultado de un proyecto no es una aplicación, sino una librería para que ser utilizada por terceros.
- Ejemplos: Nexus, Artifactory, npm, etc.
- Comparación



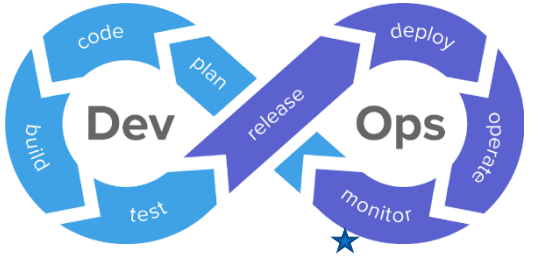
2.12 Otras herramientas
Orquestación

- Son herramientas que van más allá del despliegue de una aplicación en un servidor, permitiendo otras funciones avanzadas como la duplicación de la aplicación en varios servidores, el balanceo de carga, etc.
- Fases del ciclo de vida DevOps: Se utilizarían sobre todo en las fases de Despliegue (deploy) y Operación (operation).
- Ejemplos: Kubernetes, Nomad, Mesosphere, etc.
- Comparación



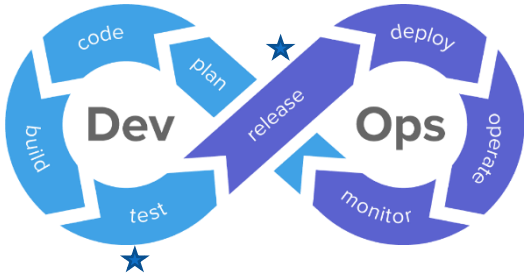
2.12 Otras herramientas Monitorización

- Son herramientas que extraen datos sobre el uso de una aplicación o de la infraestructura en la que está funcionado generando información que puede ayudar a mejorarla.
- Fases del ciclo de vida DevOps: Se utilizarían en la fase de Monitorización (monitor).
- Ejemplos: Datadog, Google Analytics, etc.
- Comparación



2.12 Otras herramientas
Pruebas de rendimiento

- Son herramientas para comprobar el comportamiento de una aplicación en situaciones extremas.
- Incluyen pruebas de carga (con muchas peticiones de usuarios concurrentes y gran número de transacciones), pruebas de estrés (forzar la carga hasta llegar a “romper” la aplicación), pruebas de estabilidad (con una carga continuada en el tiempo), pruebas de picos (con cambios drásticos en la carga).
- Fases del ciclo de vida DevOps: Se utilizarían en la fase de Pruebas (test), y en la de Liberación (release) para probar en un entorno de pre-producción.
- Ejemplos: JMeter, Load Runner, Gatling, etc.
- Comparación



3. Comparación de herramientas

- Existen muchas herramientas de cada categoría que son candidatas para ser usadas en un proyecto de integración continua
- Es necesario realizar una selección en base a una comparación de herramientas teniendo en cuenta criterios y casos de uso
- Los pasos para hacer la comparación pueden ser
  1. Definición de criterios de comparación
  2. Evaluación de los criterios de comparación para cada herramienta
  3. Comparación de las herramientas considerando los criterios
  4. Comparación de las herramientas considerando casos de uso

3.1 Definición de criterios de comparación

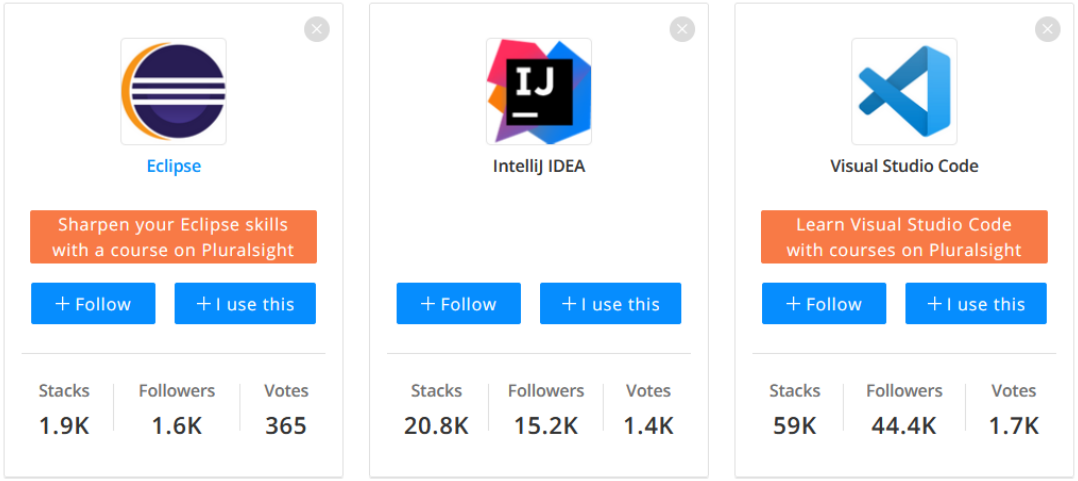
- Antes de comparar dos herramientas hay que definir los criterios que se usarán en la comparación
  - También llamados factores, propiedades, características, indicadores, etc.
- Cada criterio debe tener tiene un tipo de dato
  - Numérico, booleano (S/N), texto libre, fecha, escala, etc.
- Los criterios se pueden agrupar por categorías
  - Generales, Rendimiento, Funcionalidad, Usabilidad, etc.

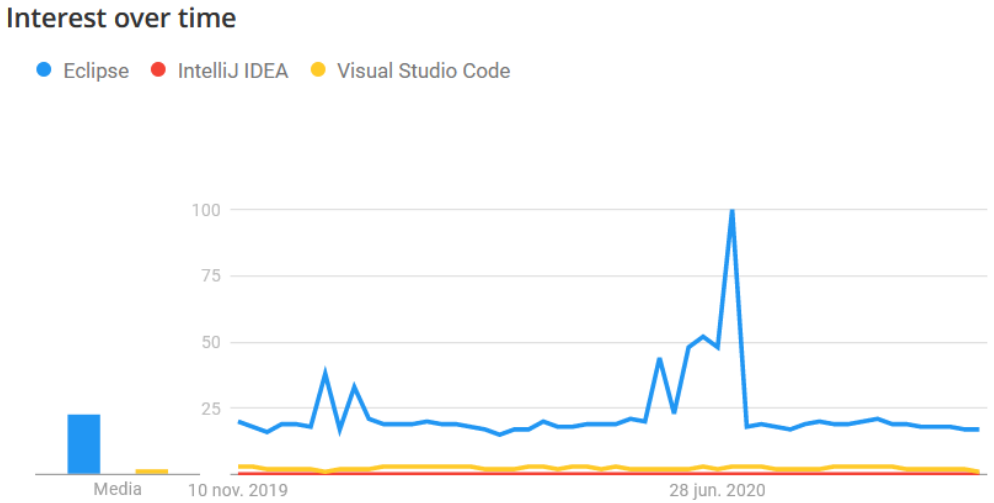
3.2 Evaluación de los criterios de comparación para cada herramienta

- Se trata de dar un valor a cada criterio para cada una de las herramientas
  - Respetando el tipo de dato del criterio
- La evaluación de los criterios puede hacerse
  - probando la herramienta
  - analizando fuentes de información primaria (fabricante)
  - analizando fuentes secundarias (artículos, encuestas, otras comparaciones, etc.)
- La evaluación de cada herramienta debe hacerse de forma independiente, sin compararlas

3.3 Comparación de las herramientas considerando los criterios

- Se trata de comparar los valores de los criterios para cada herramienta y comentar dichos valores
- En los criterios importantes, es conveniente resaltar de alguna forma (p.ej. colores), cuál de las dos herramientas es más relevante
- Suele presentarse en forma de tabla, aunque en algunos casos es conveniente utilizar otros tipos de presentaciones





Fuente: stackshare

3.4 Comparación de las herramientas considerando casos de uso

- Se trata de plantear diferentes situaciones o casos de uso y realizar la comparación, teniendo en cuenta los criterios relevantes en cada caso planteado
- En algunos casos es conveniente asignar pesos a los criterios
  - Por ejemplo cuando se trata de asignar una puntuación final a cada herramienta

4. Conclusiones

- Existen muchas herramientas que permiten automatizar la ejecución de los trabajos a realizar en los proyectos de desarrollo ágil con integración continua
- Las herramientas deben poder activarse mediante los comandos indicados en la definición del workflow (o pipeline) del proyecto
- Es necesario previamente realizar una selección de las herramientas más adecuadas en cada caso, para lo que hay que hacer una comparación rigurosa entre las herramientas disponibles en el mercado para cada categoría