

Herramientas de gestión de pruebas

Percy Taquila Carazas, Katerin Merino Quispe, Abraham Lipa Calabilla,
Edwart Balcon Coahila, Lisbeth Espinoza Caso

January 9, 2021

Abstract

The Test Management Tool is the tool that provides support for the test management and control of part of the testing process. Often it has multiple capabilities, such as managing test support products, test planning, results recording, process tracking, incident management, and test reporting. They have different approaches to testing and therefore have different sets of features. They are typically used to maintain and schedule manual tests, run or collect automated test execution data, manage multiple environments, and enter information on found defects.

Resumen

Las Herramienta de gestión de pruebas es la herramienta que proporciona soporte a la gestión de pruebas y control de parte del proceso de pruebas. A menudo tiene varias capacidades, tales como gestionar los productos de soporte de pruebas, planificación de pruebas, registro de resultados, seguimiento del proceso, gestión de incidencias y generación de informes de las pruebas. Tienen diferentes enfoques de prueba y, por lo tanto, tienen diferentes conjuntos de características. Generalmente se utilizan para mantener y planificar pruebas manuales, ejecutar o recopilar datos de ejecución de pruebas automatizadas, administrar múltiples entornos e ingresar información sobre defectos encontrados.

I. INTRODUCCION

Las herramientas de gestión de pruebas son aquellas que se utilizan para gestionar la información relativa a los «casos de prueba», normalmente los funcionales, para planificar actividades de testing, para gestionar los informes resultantes después de pasar dichos test, etc. Es fundamental para cualquier proyecto, salvo que sea muy pequeño, contar con alguna herramienta de gestión de pruebas. Hay herramientas que van por separado y otras que integran con herramientas complementarias, por ejemplo, con las de «bug tracking».

Probar no es solo buscar errores y regresiones. La gestión de pruebas también es vital. En cualquier empresa, es fundamental que pueda auditar la cobertura, la ejecución y los resultados de las pruebas. Aquí es donde en-

tran en juego las herramientas de gestión de pruebas.

II. DESARROLLO

i. GitHub Actions

Permite automatizar, personalizar y ejecutar flujos de trabajo de desarrollo de software directamente en tu repositorio. Así mismo, permite descubrir, crear y compartir acciones para realizar incluso **CI/CD**, y combinar acciones en un flujo de trabajo personalizado. [1]

GitHub Actions ayuda a automatizar tareas dentro del ciclo de vida del desarrollo de software. GitHub Actions está controlada por eventos, lo que significa que puede ejecutar una serie de comandos después de que haya ocurrido un evento específico. Por ejemplo, cada vez que alguien crea *pull request* para un repositorio,

puede ejecutar automáticamente un comando que ejecuta un script de prueba de software. [2]

i.1 Características

- Es posible encontrar un paso manual (*manual step*) codificado, probado y *open-source* como un **GitHub Action script** en **GitHub Marketplace**.
- Disponible para múltiples lenguajes y sistemas operativos.
- Permite *Continuous Deployment* a **Azure**, **AWS**, **Google** y otras plataformas en la nube.

i.2 Componentes

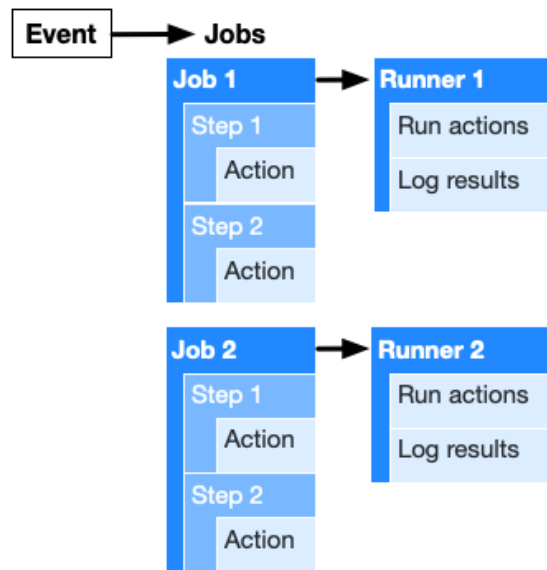


Figure 1: Diagrama de componentes en GitHub Actions

A continuación, se describirán los componentes de GitHub Actions. [2]

Workflows El flujo de trabajo (*workflow*) es un procedimiento automatizado que agrega a su repositorio. Los flujos de trabajo se componen de uno o más trabajos (*jobs*) y pueden ser programados o activados por un evento (*event*). El flujo de trabajo se puede usar para:

- Crear
- Probar
- Empaquetar
- Lanzar
- Implementar

Events Un evento (*event*) es una actividad específica que desencadena un flujo de trabajo. Por ejemplo, la actividad puede originarse en GitHub cuando alguien:

- Ejecuta un **push**
- Crea un problema (*issue*)
- Crea un **pull request**

Jobs Un trabajo (*job*) es un conjunto de pasos (*steps*) que se ejecutan en el mismo *runner*. De forma predeterminada, un flujo de trabajo con varios trabajos ejecutará esos trabajos en paralelo, pero también puede hacerlo de forma secuencial. Por ejemplo, un flujo de trabajo puede tener dos trabajos secuenciales que compilan y prueban código, donde el trabajo de prueba depende del estado del trabajo de compilación. Si el trabajo de compilación falla, el trabajo de prueba no se ejecutará.

Steps Un paso (*step*) es una tarea individual que puede ejecutar comandos en un trabajo. Un paso puede ser una acción o un comando de shell. Cada paso de un trabajo se ejecuta en el mismo *runner*, lo que permite que las acciones de ese trabajo compartan datos entre sí.

Actions Las acciones (*actions*) son comandos independientes que se combinan en pasos para crear un trabajo. Las acciones son el bloque de construcción portátil más pequeño de un flujo de trabajo. Es posible crear sus propias acciones o utilizar acciones creadas por la comunidad de GitHub. Para usar una acción en un flujo de trabajo, debe incluirla como un paso.

Runners Un runner es un servidor que tiene instalada la aplicación de ejecución de acciones de GitHub. Puede utilizar un runner alojado en GitHub, o puede alojar el

suyo propio. Un runner escucha los trabajos disponibles, ejecuta un trabajo a la vez e informa el progreso, los registros y los resultados a GitHub. Para los ejecutores alojados en GitHub, cada trabajo de un flujo de trabajo se ejecuta en un entorno virtual nuevo.

Los runners alojados en GitHub se basan en:

- Ubuntu Linux
- Microsoft Windows
- macOS

i.3 YAML

YAML es un lenguaje de serialización de datos diseñado para ser *leído y escrito por humanos*. Basa su funcionalidad en JSON, con la adición de líneas nuevas e indentación inspirada en Python. A diferencia de Python, YAML no permite tabulaciones literales. [3]

Las GitHub Actions utiliza la sintaxis YAML para definir los eventos, trabajos y pasos. Estos archivos YAML se almacenan en el repositorio, en el directorio `.github/workflows`. [2]

i.4 Ejemplo

Se puede crear un flujo de trabajo de ejemplo en un repositorio que activa automáticamente una serie de comandos cada vez que se realiza un **push**. En este flujo de trabajo, GitHub Actions verifica el código enviado, instala las dependencias del software y ejecuta **bats -v**.

Los pasos para crear este flujo de trabajo son los siguientes:

1. Crear un repositorio en GitHub.
2. Crear un nuevo archivo `learn-github-actions.yml` en el directorio `.github/workflows/` con el siguiente código:

```
name: learn-github-actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
```

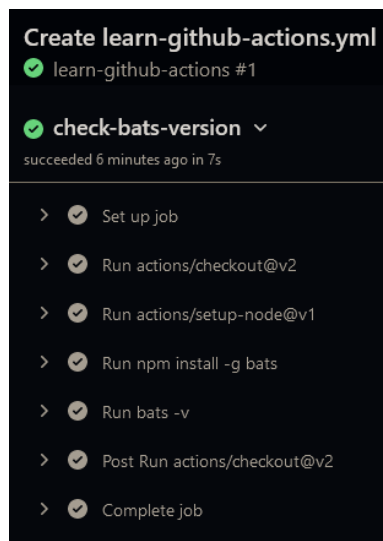


Figure 2: Captura del flujo de trabajo en la pestaña Actions

```
steps:
  - uses: actions/checkout@v2
  - uses: actions/setup-node@v1
  - run: npm install -g bats
  - run: bats -v
```

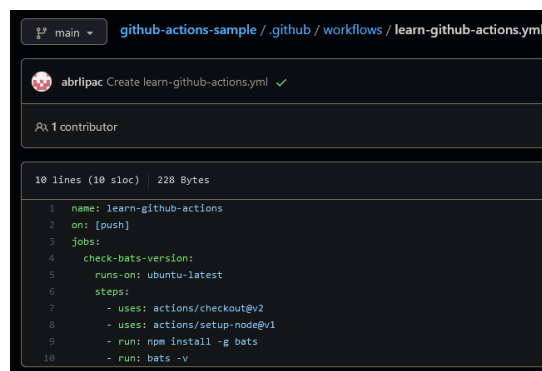


Figure 3: Captura de la ruta del archivo creado

3. Guardar los cambios (*commit*)

Cada vez que se haga *push* al repositorio, se ejecutará el flujo de trabajo creado.

Es posible ver la actividad de los flujos en la pestaña **Actions**. Al abrir un workflow se puede ver los pasos y los logs generados en cada uno.

Producto	Almacenamiento	Minutos (por mes)
GitHub Free	500 MB	2,000
GitHub Pro	1 GB	3,000
GitHub Free para organizaciones	500 MB	2,000
GitHub Team	2 GB	3,000
GitHub Enterprise Cloud	50 GB	50,000

Table 1: Tabla de los productos ofrecidos por GitHub, Almacenamiento y minutos mensuales

Sistema operativo	Tasa por minuto	Multiplicador de minutos
Linux	\$0.008	1
macOS	\$0.08	10
Windows	\$0.016	2

Table 2: Tabla de los sistemas operativos soportados, la tasa por minuto en dólares y el multiplicador

i.5 Precios

El uso de GitHub Actions es gratuito para los repositorios públicos. Para los repositorios privados, cada cuenta de GitHub recibe una cantidad determinada de minutos y almacenamiento gratuitos dependiendo del producto que se utilice con la cuenta.

Cada sistema operativo consume una cantidad diferente de minutos, siendo 1 minuto de trabajos ejecutados en Linux equivalente a 1 minuto de facturación mensual.

ii. Jenkins

Jenkins es un servidor de automatización de código abierto autónomo que se puede utilizar para automatizar todo tipo de tareas relacionadas con la creación, prueba y entrega o implementación de software.

Jenkins se puede instalar a través de paquetes de sistema nativos, Docker o incluso ejecutar de forma independiente en cualquier máquina con un entorno de ejecución de Java (JRE) instalado.

ii.1 Instalación

Para instalar Jenkins se requiere:

- Hardware:

- 256 MB de RAM, aunque se recomiendan más de 512 MB
- 10 GB de espacio en disco (para Jenkins y su imagen de Docker)

- Software:

- Java 8 u 11 (ya sea un JRE o un Java Development Kit (JDK))
- Docker

Para ejecutar Jenkins:

- Descargar Jenkins.
- Abrir un terminal en la carpeta de descargas.
- Ejecutar `java -jar jenkins.war -httpPort=8080`.
- Navegar a `http://localhost:8080`.
- Seguir las instrucciones para completar la instalación.

ii.2 Jenkins pipeline

Jenkins Pipeline (o simplemente Pipeline) es un conjunto de complementos que admite la implementación e integración de pipelines de entrega continua en Jenkins.

Un pipeline de entrega continua es una expresión automatizada de su proceso para obtener software desde el control de versiones hasta sus usuarios y clientes.

Jenkins Pipeline proporciona un conjunto extensible de herramientas para modelar, de lo simple a lo complejo, delivery pipelines como código. La definición de Jenkins Pipeline generalmente se escribe en un archivo de texto (llamado Jenkinsfile) que, a su vez, se registra en el repositorio de control de código fuente de un proyecto.

iii. Azure DevOps

iii.1 Definición

Donovan Brown, el Administrador de DevOps principal en el grupo de Azure lo describe: "DevOps es la unión de personas, procesos y productos para permitir la entrega continua de valor a los usuarios finales".[5]

DevOps permite que los roles que antes estaban aislados (desarrollo, operaciones de TI, ingeniería de la calidad y seguridad) se coordinen y colaboren para producir productos mejores y más confiables.[6]

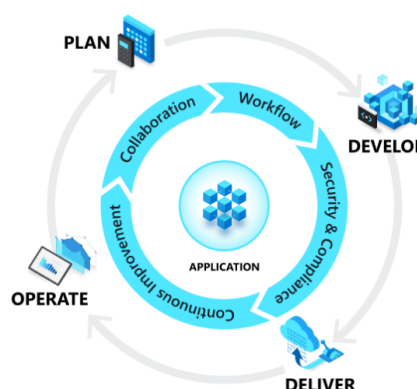
iii.2 Ventajas

Mejoran el rendimiento y crean productos de más calidad en menos tiempo, lo que aumenta la satisfacción de los clientes. Esta mejora de la colaboración y la productividad es fundamental también para alcanzar objetivos de negocio como estos:

- Reducción del tiempo de comercialización
- Adaptación al mercado y a la competencia
- Mantenimiento de la estabilidad y la confiabilidad del sistema
- Mejora del tiempo medio de recuperación

iii.3 DevOps y el ciclo de vida de las aplicaciones

DevOps influye en el ciclo de vida de las aplicaciones a lo largo de las fases de planeamiento, desarrollo, entrega y uso.



iii.4 Herramientas

- **GitHub** Aumenta la colaboración, flujos de trabajo del código
- **Azure Pipelines** Implementa CI/CD para compilar, probar e implementar soluciones de forma continuada
- **Azure Boards** Planea, controla el trabajo entre los equipos
- **Azure Monitor** Visibilidad sobre las aplicaciones, la infraestructura y la red
- **Visual Studio IDE** diseñado para crear aplicaciones eficaces y escalables para Azure
- **Azure Kubernetes Service** Distribuye aplicaciones en contenedores con más rapidez

iv. Comparación

iv.1 Jenkins y Github Actions

Diferencias

- Jenkins cuenta con pipelines monolíticos y secuenciales como un proceso en cascada, mientras que GitHub Actions cuenta con pipelines asíncronos y basados en componentes como un proceso ágil.
- Jenkins no está diseñado para la nube, mientras que GitHub Actions está en la nube (aunque también se puede instalar en un servidor local).
- Jenkins requiere una instalación, mientras que GitHub Actions, no.
- Jenkins tiene un diseño basado en cuentas y activadores que no se corresponde con un evento de GitHub, mientras que

GitHub Actions	Azure Pipeline
Son solo para GitHub	Se puede usar fácilmente para otros sistemas de control de código fuente.
se pueden usar para automatizar las revisiones de código, la administración de sucursales y la clasificación de problemas de la manera que desee	Se usa principalmente para automatizar los flujos de trabajo de CI / CD.
admite aprobaciones, puertas y comprobaciones	No tiene nada de eso, lo que dificulta mucho el uso de GitHub Actions para organizar las versiones empresariales.
Solo tienen enlaces web.	tiene conexiones de servicio y enlaces web como una forma de organizar las conexiones a servicios externos y remotos en las pipelines
Admiten acciones basadas en contenedores, lo que las hace muy flexibles.	Las extensiones de Azure Pipelines solo se pueden crear en TypeScript o PowerShell.
Requieren que separe las etapas en archivos de flujo de trabajo YAML separados.	Azure Pipelines admite las etapas definidas en el archivo YAML, que se pueden usar para crear flujos de trabajo de implementación

Figure 4: Diferencias entre Azure Pipeline y GitHub Actions

GitHub Actions, en los eventos de GitHub, entre los cuales están: *push*, *pull*, *issue*, *comment*, *clone*, *publish*, entre otros.

- GitHub Marketplace tiene más de 4,000 acciones lanzadas en menos de dos años. Jenkins ha lanzado más de 1,000 complementos diferentes en aproximadamente diez años.
- Jenkins tiene un costo de mantenimiento más alto que GitHub Action.
- GitHub Actions es más flexible que Jenkins al ser posible ejecutarlo localmente y enviarlo a GitLab u otro almacenamiento de versiones en la nube.
- Jenkins requiere de un servidor dedicado, mientras que GitHub Actions, no.

- Los flujos de trabajo incluyen uno o más trabajos.
- Los trabajos incluyen uno o más pasos o comandos individuales. Los trabajos y los pasos de Azure Pipelines son muy similares a los trabajos y los pasos de las GitHub Actions.
- Los pasos o tareas se pueden reutilizar y compartir con la comunidad.

Diferencias entre Azure Pipeline y GitHub Actions[7]

A pesar de usar, la sintaxis YAML, Azure Pipelines y GitHub Actions tienen *Keywords* diferentes con un significado estructural similar.

v. Azure Pipeline y GitHub Actions

v.1 Similitudes

- Los archivos de configuración del flujo de trabajo se escriben en YAML y se almacenan en el repositorio del código.

Azure Pipelines		GitHub Actions	
Concept	Keyword	Concept	Keyword
Pipeline		Workflow	
Trigger	<code>trigger / pr / schedules</code>	Event	<code>on</code> (with <code><event_name></code> argument)
Stage	<code>stage</code>	Not supported	
Job	<code>job</code> (optional)	Job	<code><job_id></code> (mandatory)
Dependency	<code>demands</code>	Dependency	<code>needs</code>
Execution condition	<code>condition</code>	Execution condition	<code>if</code>
Agent (from pool)	<code>pool</code>	Runner	<code>runs-on</code>
Script	<code>script / bash / powershell / push</code>	Script	<code>run</code> (with <code>shell</code> argument)
Task		Action	<code>uses</code>
Display name	<code>displayname</code>	Display name	<code>name</code> (can be <code>first</code> task/script argument)
Task-specific parameters	<code>inputs</code>	Action-specific parameters	<code>with</code>
Execution condition	<code>condition</code>	Execution condition	<code>if</code>

Figure 5: Tabla comparativa de sintaxis de Azure Pipelines y GitHub Actions

III. CONCLUSIONES

En conclusión, luego de un análisis, se puede notar que GitHub Actions facilita la creación, prueba e implementación de su código directamente desde GitHub, sin embargo, si se desea utilizar flujos de trabajo de GitHub Action solo para distribución o implementación continua se recomendaría usar Azure Pipelines y sus ya conocidas características para permitir implementaciones seguras y compatibles en los entornos de producción. Cada proyecto de software realizado y gestionado a través de estas herramientas permitirá una mejor toma de decisiones por parte de la gerencia o dirección de proyectos. Así mismo, es importante destacar que, para la implementación de la herramienta, se consideran aspectos relevantes en el área de pruebas de software, así como algunas métricas para software, todo esto con el propósito de enriquecer los criterios de evaluación y obtener un producto de calidad. La realización de este artículo permite entender que la gestión de las pruebas de software es una actividad que indiscutiblemente debe llevarse a cabo en todo proyecto de software.

IV. RECOMENDACIONES

- Se recomienda verificar todos los cambios antes y después de refactorizar la

aplicación. La herramienta elegida debe cumplir con los requisitos para realizar y entregar los resultados esperados.

- La capacidad de informar errores y fallas es importante para que los desarrolladores puedan reproducir el problema y solucionarlo. Es bueno tomar en cuenta las herramientas que permiten revisar los resúmenes de registro de cada prueba a lo largo de los plazos.
- La herramienta elegida siempre debe probar nuevos cambios de código y brindar retroalimentación. Una herramienta de prueba sólida debe tener la capacidad de admitir el marco de prueba, el control de revisión, la gestión de configuración de prueba, el seguimiento de problemas, la generación de informes y más.

REFERENCIAS

- [1] GitHub. (s. f.). Documentación de GitHub Actions - GitHub Docs. GitHub Actions. Recuperado 9 de enero de 2021, de <https://docs.github.com/es/free-pro-team@latest/actions>
- [2] GitHub. (s. f.-b). Introduction to GitHub Actions - GitHub Docs. Introduction to GitHub Actions. Recuperado 9 de enero de 2021, de <https://docs.github.com/es/free-pro-team@latest/actions/learn-github-actions/introduction-to-github-actions>
- [3] Leigh Brenecki, L. B. (s. f.). Learn yaml in Y Minutes. Aprende X en Y minutos Donde X=yaml. Recuperado 9 de enero de 2021, de <https://learnxinyminutes.com/docs/es-es/yaml-es/>
- [4] GitHub. (s. f.-a). Acerca de la facturación para GitHub Actions - GitHub Docs. Acerca de la facturación para GitHub Actions. Recuperado 9 de enero de 2021, de <https://docs.github.com/es/free-pro-team@latest/github/setting-up->

- and-managing-billing-and-payments-on-github/about-billing-for-github-actions
- [5] Cottman, B. H. (2020, 9 julio). Will Github Actions Kill off Jenkins? - The Startup. Medium. <https://medium.com/swlh/will-github-actions-kill-off-jenkins-f85e614bb8d3>
 - [6] Microsoft Azure. (n.d.). ¿Qué es DevOps? Explicación de DevOps | Microsoft Azure. Retrieved January 9, 2021, from <https://azure.microsoft.com/es-es/overview/what-is-devops/>
 - [7] Yermakhanov, M. (2020, 11 diciembre). Azure Pipelines vs. GitHub Actions: Key Differences. Medium. <https://medium.com/objectsharp/azure-pipelines-vs-github-actions-key-differences-45390ab132ee>
 - [8] J.G. (2013, October 23). Análisis y comparativa de herramientas de gestión de pruebas de uso gratuito. Javier Garzas. <https://www.javiergarzas.com/2013/10/herramientas-de-gestion-de-pruebas.html>
 - [9] Yermakhanov, M. (2020, 11 diciembre). Azure Pipelines vs. GitHub Actions: Key Differences. Medium. <https://medium.com/objectsharp/azure-pipelines-vs-github-actions-key-differences-45390ab132ee>
 - [10] Jenkins. (n.d.). Jenkins Documentation. Jenkins User Documentation. Retrieved January 9, 2021, from <https://www.jenkins.io/doc/>
 - [11] Jenkins. (n.d.-a). Getting started with the Guided Tour. Jenkins Documentation. Retrieved January 9, 2021, from <https://www.jenkins.io/doc/pipeline/tour/getting-started/>