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 entran 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

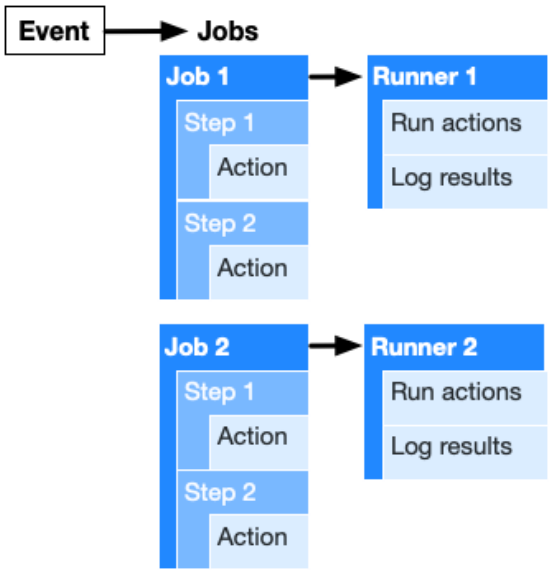

Figure 1: *Diagrama de componentes en GitHub Actions*

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

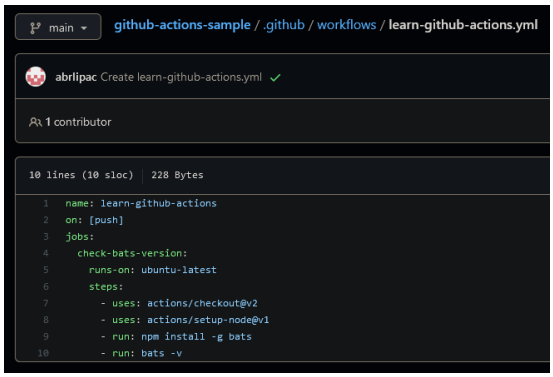

Figure 2: *Captura de la ruta del archivo creado*

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.

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. Comparación

ii.1 Jenkins y Github Actions

Diferencias

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*

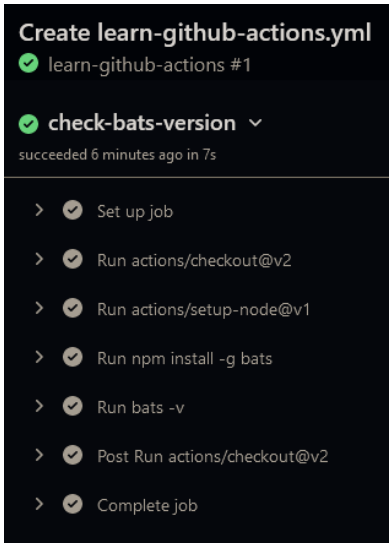

Figure 3: *Captura del flujo de trabajo en la pestaña Actions*

- 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, 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.

ii.2 GitHub Actions y Azure Pipelines

Similitudes

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

- GitHub Actions (en la nube) solo es para GitHub, mientras que Azure Pipelines se puede usar fácilmente con diferentes sistemas de control de versiones.
- Github Actions se puede usar para automatizar las revisiones de código, la administración de ramas y la clasificación de issues, mientras que Azure Pipelines se usa principalmente para automatizar los flujos de trabajo de CI/CD.
- Azure Pipeline soporta entornos (*environments*), grupos de implementación y grupos / colas de agentes, mientras que en GitHub Actions solo puede administrar los **self-hosted runners** mediante grupos (que solo están disponibles en cuentas empresariales).
- Azure Pipelines admite aprobaciones, *gates* y verificaciones, mientras que GitHub Actions no, lo que dificulta el uso de GitHub Actions para organizar los lanzamientos empresariales.
- Azure Pipelines tiene conexiones de servicio y *web hooks* como una forma de organizar las conexiones a servicios externos y remotos en los *pipelines*, mientras que GitHub Actions solo tiene *web Hooks*.
- Azure Pipelines tiene variables (*secrets* y en texto sin cifrar) y grupos de variables para ayudar a administrar configuraciones en los *pipelines*, mientras que GitHub Actions solo tiene *secrets* (vinculados al repositorio u organización).
- Azure Pipelines tiene Secure Files, GitHub Actions no.
- Azure Pipelines soporta Stages en el mismo archivo YAML, GitHub Actions no.
- Las extensiones de Azure Pipelines pueden agrupar varias tareas, mientras que en GitHub Actions es una acción dentro del repositorio de GitHub.
- GitHub Actions admite acciones basadas en contenedores, lo que las hace muy flexibles. Las extensiones de Azure Pipelines solo se pueden crear en Typecript o PowerShell.
- GitHub tiene un escaneo de código integrado y puede integrarse con *security gates* de terceros, mientras que Azure Pipelines solo puede integrarse con *security gates* de terceros.
- En Azure Pipelines, los scripts se pueden configurar para generar errores si se envía algún resultado a *stderr*. GitHub Actions no admite esta configuración.
- GitHub Actions requiere una configuración explícita y la estructura YAML no se puede omitir.

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

Figure 4: *Tabla comparativa de sintaxis de Azure Pipelines y GitHub Actions*

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

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] 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