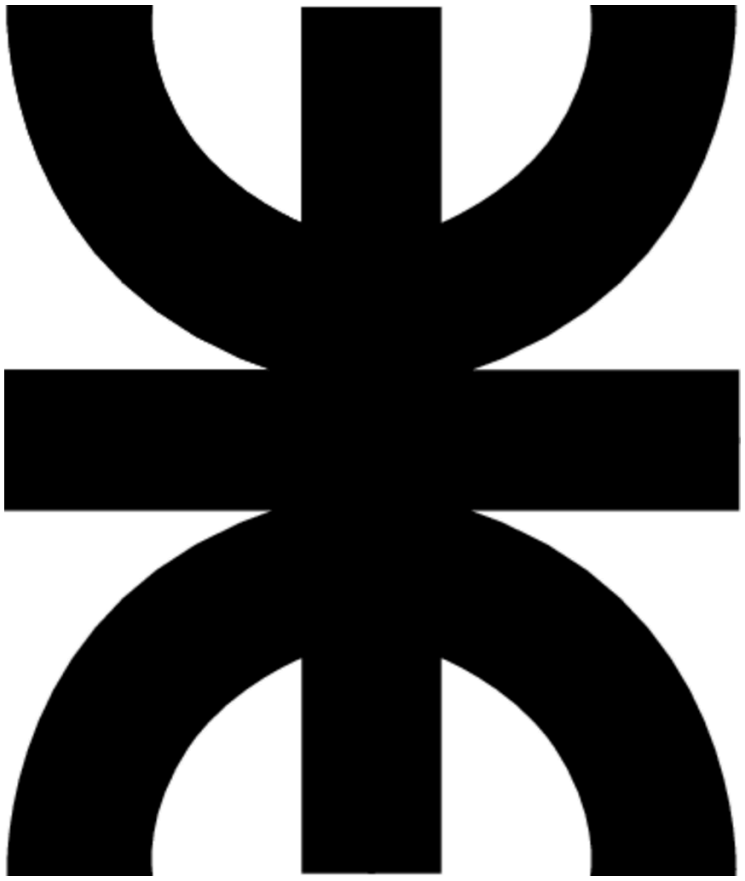


UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL DE RESISTENCIA
INGENIERÍA EN SISTEMAS DE INFORMACIÓN
INGENIERÍA Y CALIDAD DE SOFTWARE - AÑO 2025

Título: Taller de Integración y Entrega Continua

Alumno: Mariano Gastón Insaurralde

Profesores: Verónica Andrea Bollati, Mariano Minoli y Marcela Arias

Fecha: 12 de junio

Introducción

Este taller tiene como objetivo aplicar los conceptos de integración y entrega continua mediante la implementación de un entorno que incluya un repositorio central para el control de versiones, un servidor de integración continua, un entorno de entrega y un mecanismo de feedback. Se incluirán también pruebas automatizadas que abarquen los siguientes aspectos: pruebas unitarias a los componentes de la aplicación, control de calidad del código para seguir buenas prácticas del desarrollo de software y pruebas de estrés para probar el rendimiento de la aplicación en el entorno de entrega.

Con el desarrollo de este taller espero fijar los conceptos teóricos vistos en clase, aplicándolos al desarrollo de una aplicación simple, pero que de igual forma me permita aplicarlos efectivamente. Espero conocer las distintas opciones del mercado y poder definir un slack que se adapte a mis conocimientos previos en programación. También, espero conocer los desafíos asociados al desarrollo de software desde este rol y ser capaz de superarlos.

Primero que nada, a modo de introducción teórica a este taller, quiero mencionar por qué considero que es importante la integración y entrega continua:

- Con la integración continua cada miembro del equipo integra sus cambios en un repositorio central compartido al menos una vez al día. Cada una de estas integraciones son verificadas con una build automatizada, en la cual se incluyen pruebas automatizadas, para detectar errores lo más rápido posible. Con esto se asegura software funcional con cada nuevo cambio.
- Con el despliegue continuo el software siempre se encuentra en un estado en el cual se pueda publicar la última build, y llevándolo un paso más allá, también nos permite automatizar el proceso de publicación a producción cada cambio exitoso.

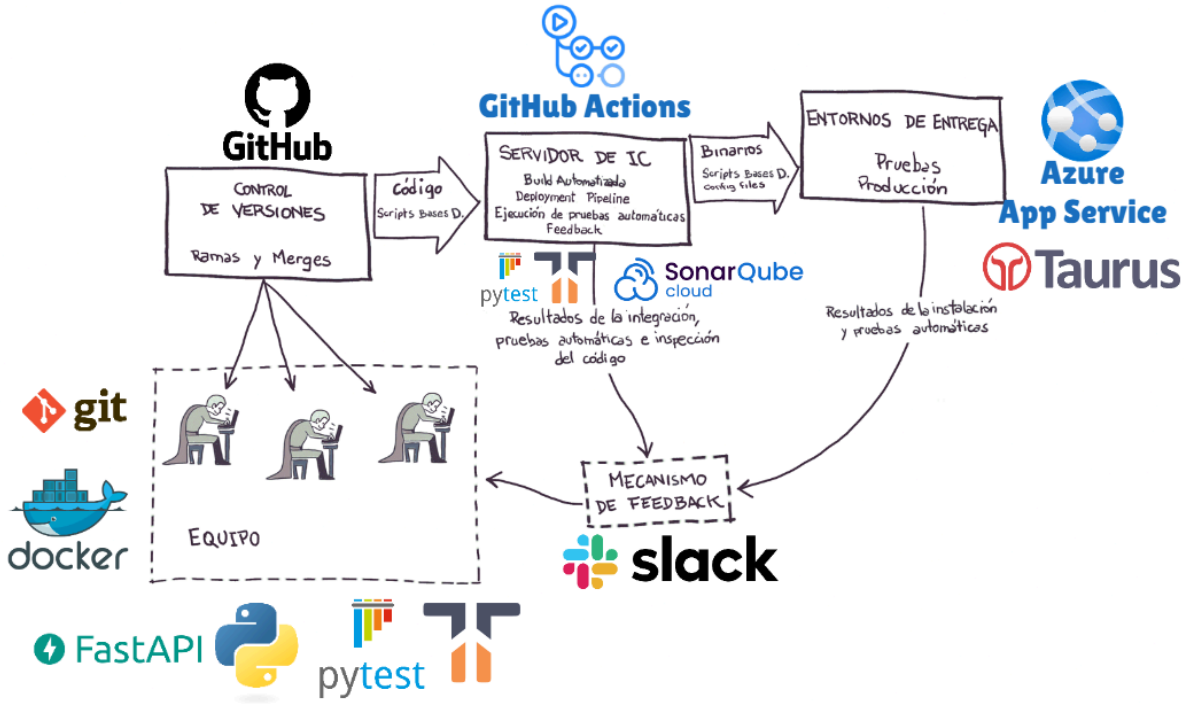
Juntas, estas prácticas reducen el riesgo y el tiempo desperdiciado en el proceso de entrega de software, y mejoran la calidad y la capacidad de respuesta a las necesidades del negocio y los usuarios.

Por último, pero no menos importante, voy a describir brevemente la aplicación a desarrollar; la cual no es más que un contador de caracteres y un contador de palabras, dados un texto y una cantidad de palabras a contar como entrada.

- Contador de Caracteres: Cuenta la cantidad de caracteres en un texto sin contar caracteres especiales, es decir, solo cuenta letras y números.
- Contador de Palabras: Cuenta la cantidad de palabras de una longitud específica de caracteres en un texto.

Entorno de Desarrollo

Diagrama de Componentes



Entorno Local de Desarrollo

En el entorno local se trabajará con Python, el desarrollo de la aplicación se hará con un servidor de FastAPI, se desarrollarán tests unitarios de los componentes de la aplicación con Pytest y Coverage, de manera que las mismas queden documentadas. Para garantizar la consistencia del entorno de ejecución, se utilizará Docker. Y para el manejo de versiones se utilizará Git.

Particularmente, nunca había trabajado con Pytest y Coverage, por lo que fueron herramientas que tuve que aprender, lo cual no resultó para nada complicado.

Para la ejecución y generación de reportes de las pruebas locales (Pytest + Coverage):

coverage run -m pytest

coverage xml

Para trabajar con Docker:

docker compose build

docker compose up -d

Control de Versiones

Como repositorio de código para el control de versiones y ramas, se utilizará GitHub, dado que era una herramienta que ya conocía.

Servidor de Integración Continua

Como servidor de integración continua se utilizará GitHub Actions por su facilidad de integración con GitHub, su simplicidad para definir flujos de trabajo que automaticen todos los pasos, desde la creación de la build, las pruebas e inspección, hasta el despliegue de la aplicación.

Esta herramienta no la había utilizado antes, pero no resultó tan complicada de aprender, ya que se utilizan archivos .YAML para su codificación, lo cual facilita su comprensión.

Pruebas Automatizadas e Inspección de Código

Dentro del flujo de trabajo de GitHub Actions también se incluyen pruebas automatizadas para verificar la build, tanto para la rama main, como para la rama dev también, asegurando que no se produzcan errores en el código. Para esto, utilice las mismas herramientas que se utilizan en el entorno local de desarrollo, Pytest y Coverage.

Un paso extra que decidí añadir al flujo fue el de inspección del código. Este es un trabajo que no había realizado antes, y considero que es un buen añadido al proyecto, porque el software SonarQube garantiza el aseguramiento de la calidad del código basándose en buenas prácticas, además proporciona información sobre la cobertura de los tests unitarios desarrollados con respecto al total del código. Es una excelente herramienta para visualizar distintas métricas del proyecto, como la complejidad, la cobertura de código o la duplicación de código.

Entorno de Entrega

El entorno de entrega será desarrollado utilizando Azure App Services. Esta decisión se basa en que ya tenía algo de experiencia utilizando Azure, y considero que es una herramienta muy demandada en el mercado, y seguir capacitándome en ella va a ayudarme a definir mi slack profesional. Además, el hecho de tener una cuenta con suscripción de estudiante facilita el uso de la plataforma. La desventaja que le encuentro es que por momentos resulta muy lenta a la hora de subir nuevos cambios y que se actualice la página del despliegue.

Una gran ventaja de utilizar Azure, es que a pesar de fallar los test, se mantiene activa la última versión de la página desplegada, porque se utiliza la estrategia de despliegue empaquetado (One Deploy). Esto significa que la nueva versión solo se activa una vez que se ha cargado completamente y está lista. Si algo falla durante el proceso de carga o activación, la versión anterior permanece en su lugar y sigue sirviendo el tráfico.

Esta configuración es correcta porque protege la producción de código defectuoso y mantiene la disponibilidad del servicio.

Entorno de Pruebas

Como extra, tome la decisión de implementar pruebas de estrés para probar el rendimiento de la aplicación en un entorno de entrega. Si bien no era un requisito obligatorio, me pareció muy interesante dedicarle tiempo a aprender esta nueva herramienta y llevarla a la práctica. Nunca había realizado pruebas de estrés, por lo que esto seguramente resultará todo un desafío para mi. Como herramienta decidí utilizar Taurus porque la misma fue mostrada en una exposición sobre

Benchmarking reciente por unos compañeros en la materia Administración de Recursos, me pareció ideal para aprenderla por su modo de trabajar con el lenguaje .YAML, además de que permite utilizar distintas herramientas de pruebas en un solo lugar, de esta manera teniendo a disposición distintos resultados de pruebas hechas por distintas aplicaciones, otra razón por la cual elegí este framework es porque es de código abierto.

Como herramienta de pruebas solo tuve tiempo de implementar JMeter, además de por el hecho de que fue mostrado en la exposición de mis compañeros, considero que fue ideal para utilizar porque es la herramienta más conocida, existe amplia documentación y es muy fácil de utilizar para APIs.

Como agregado tuve la idea de seleccionar otras dos herramientas, como lo son Locust y K6, para de esta manera aprovechar el uso de Taurus, pero no tuve el tiempo de estudiar lo suficiente estas herramientas.

La forma de trabajar es la siguiente:

- Primero se definen los archivos con las configuraciones de las pruebas.
- Luego, se carga el archivo con los datas (texto y cantidad).
- Luego, se ejecuta la prueba mediante Taurus usando su imagen de Docker.

cd stress-tests

docker run -it --rm -v "${PWD}:/bzt-configs" blazemeter/taurus jmeter-test/jmeter.yml

- Devuelve como salida un link a BlazeMeter para visualizar las métricas resultantes, aunque esta es una característica premium.
- Lo ideal es estresar al máximo al servidor, por ejemplo, ejecutando el comando desde varias terminales.

Problemas Encontrados y Soluciones Aplicadas

Tuve dificultades iniciales para desplegar la aplicación en Azure debido a que creé la Web App directamente desde el portal de Azure, lo que provocó confusión con las credenciales necesarias para la autenticación desde GitHub Actions. Al intentar iniciar sesión, devolvía un error de “No subscriptions found”, indicando que ese

cliente no tenía permisos sobre ninguna suscripción. Luego, con el paso de un día, se crearon solas las Federated Credentials que Azure necesitaba, resultó ser un problema interno de Azure que hizo que estas credenciales tarden mucho tiempo en crearse.

Conclusión

El desarrollo de este taller me permitió consolidar de manera práctica los conceptos de integración y entrega continua aplicados al ciclo de vida del software. A través de la implementación de herramientas como GitHub Actions, SonarQube, Docker, Azure App Services y Taurus, logré experimentar de primera mano los beneficios de automatizar las etapas de construcción, prueba, inspección y despliegue de una aplicación.

Cada una de las decisiones tomadas en la selección de tecnologías fue guiada tanto por su valor pedagógico como por su relevancia en el entorno profesional. Me enfrenté a varios desafíos, especialmente en la configuración del entorno de entrega y en la ejecución de pruebas de estrés, los cuales, aunque complejos, me brindaron un aprendizaje significativo y enriquecedor.

También pude explorar herramientas y prácticas que no conocía previamente, como GitHub Actions, SonarQube, mecanismo de feedback con Slack y Taurus, ampliando así mi stack de conocimientos y mejorando mi capacidad para enfrentar proyectos similares en el futuro. Más allá del objetivo académico, este taller me ayudó a entender mejor la importancia de contar con pipelines automatizados y entornos controlados, especialmente cuando se busca garantizar calidad, disponibilidad y velocidad en el desarrollo de software.

En resumen, considero que el taller cumplió su propósito: no solo reforcé los conocimientos teóricos, sino que adquirí nuevas habilidades técnicas valiosas y comprendí con mayor profundidad el rol del desarrollador en entornos donde la automatización y la calidad continua son pilares fundamentales.

Referencias Bibliográficas

Fowler, M. (2006). *Continuous integration*. ThoughtWorks. https://martinfowler.com/articles/continuousIntegration.html

Microsoft. (s.f.). *Package Deployment (One Deploy) in Azure*. Microsoft Learn. https://learn.microsoft.com/en-us/azure/azure-functions/functions-deployment-technologies?tabs=linux

Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous integration: Improving software quality and reducing risk*. Addison-Wesley.

Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley.

Majo. (n.d.). *Apunte Integrador Ingeniería y Calidad* [inédito, con memes incluidos, con errores, pero con buena intención].