Testing continuo en un flujo CI/CD

Testing continuo en un flujo CI/CD

ELDAR ACADEMY

Versión: 1.0

Fecha: 20 de noviembre de 2024

Objetivo

El objetivo de esta actividad o reto es realizar una *investigación completa* sobre la integración continua, la entrega continua y la importancia que tiene con DevOps, por otro lado investigar la configuración de los pipelines con pruebas automatizadas y sus diferentes herramientas. El alcance de esta investigación es que el colaborador pueda aprender y entender lo básico del testing continuo en un flujo CI/CD.

Introducción

Para esta investigación vamos a proceder a explicar un concepto clave que va a ser necesario entender para realizar el cuestionario principal.

¿Qué es un pipeline?

Un pipeline de CI/CD es una serie de pasos que optimizan el proceso de entrega de software. A través de un enfoque de ingeniería de DevOps, la CI/CD mejora el desarrollo de aplicaciones mediante la supervisión y la automatización.
Los pipelines automatizados pueden ayudar a prevenir los errores que resultan de los procesos manuales, permitir iteraciones rápidas del producto y proporcionar comentarios consistentes durante el proceso de desarrollo. Cada paso de un pipeline de CI/CD es un subconjunto de tareas agrupadas en las etapas del pipeline.

Para poder crear un pipeline vamos a necesitar diferentes herramientas como un repositorio de código como por ejemplo gitlab, un editor de código fuente (IDE) como por ejemplo Visual Studio Code, otra herramienta importante es el uso de Docker la cual esta proporciona un modelo de implementación basado en imágenes, permitiendo compartir fácilmente una aplicación o un conjunto de servicios, con todas las dependencias en varios entornos. Esta herramienta está diseñada a partir de los contenedores de Linux. Hay otras herramientas donde se pueden crear pipelines, estas mencionadas son algunas que específicamente usaré para la actividad.

Actividad

1. **Investiga qué es CI-CD y cómo el testing encaja dentro del ciclo.**

CI-CD (integración Continua- Entrega continua)

CI/CD es un conjunto de procesos automatizados para integrar, probar y desplegar software de manera continua, asegurando la entrega rápida y confiable de nuevas versiones. Su objetivo es mejorar la calidad y velocidad del desarrollo a lo largo del ciclo de vida del software mediante la automatización. Al implementar CI/CD en las fases de desarrollo, pruebas, producción y monitoreo del ciclo de vida del desarrollo de software, las organizaciones logran desarrollar código más seguro y eficiente. El testing juega un rol clave en este proceso, ya que valida que los cambios realizados en el código no rompan el sistema y cumplan con los requisitos del cliente. Sus principales beneficios incluyen:

- Detectar errores rápidamente, reduciendo costos de corrección.
- Aumentar la confianza en la calidad del código.
- Prevenir que versiones defectuosas lleguen a producción.

2. **Realiza un ejemplo básico donde configures un pipeline que incluya un paso de pruebas automatizadas.**

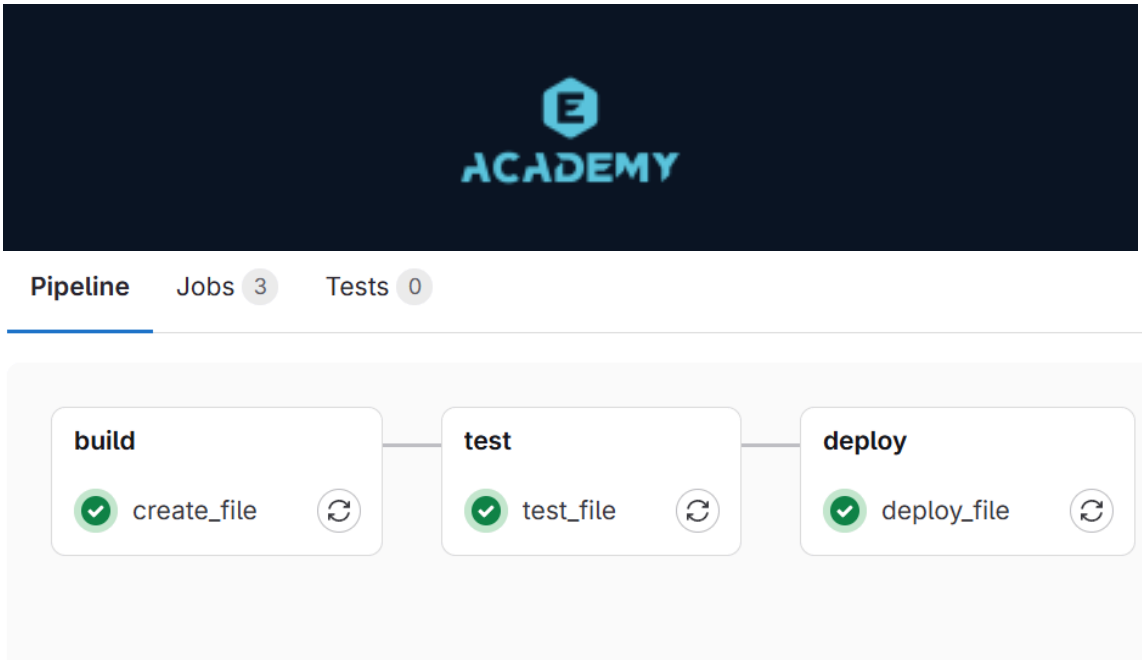
El CI/CD es un componente esencial de DevOps, diseñado para fomentar la colaboración entre los equipos de desarrollo y operaciones. Su objetivo principal es la automatización de los procesos de integración y despliegue del código, por lo que agilizan los procesos mediante los cuales una idea (ya sea función nueva, una solicitud de mejora o la corrección de un error) se transforma en una solución implementada en producción, generando valor para el usuario final.

Algunas ventajas que tienen el equipo de DevOps al implementar pruebas automatizadas:

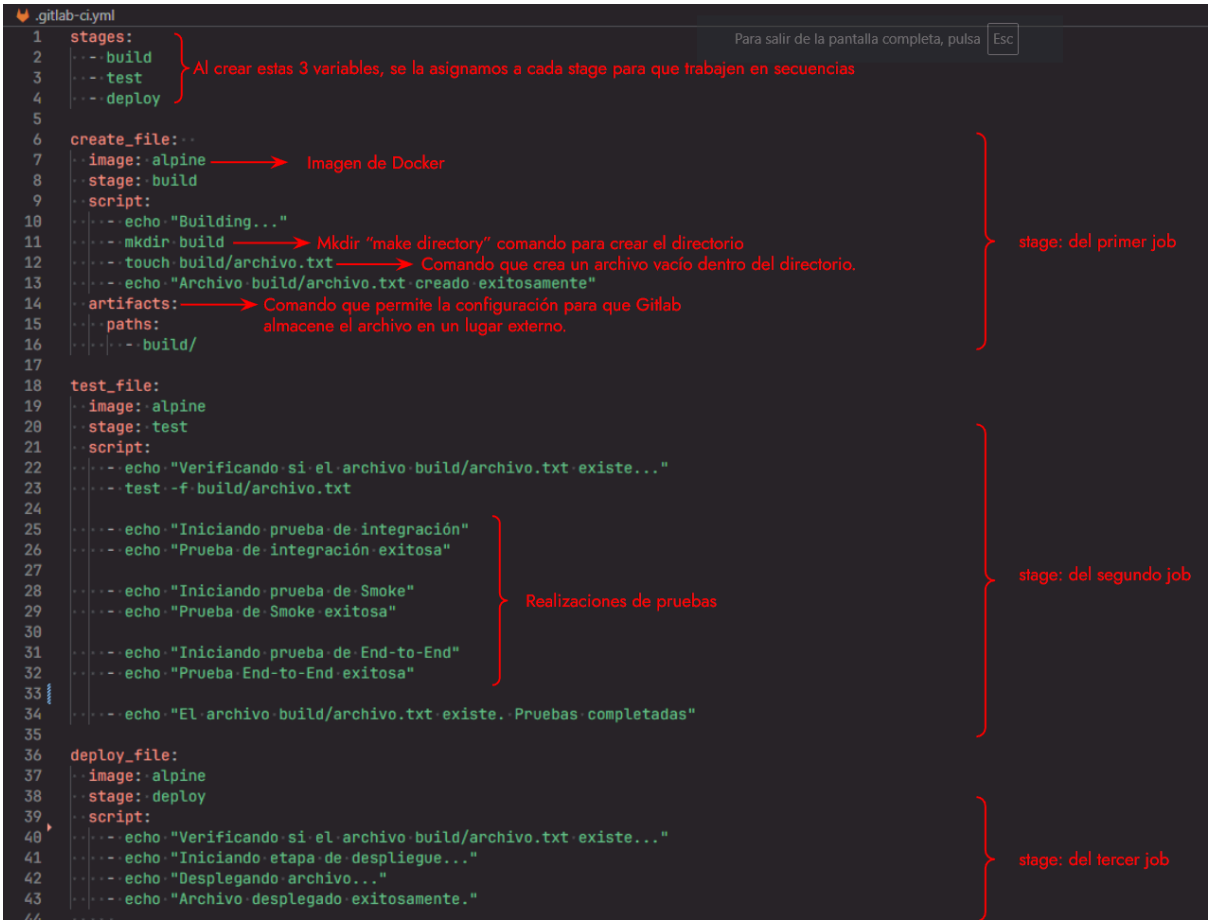
- Velocidad sin renunciar a la calidad: Acelera el desarrollo sin comprometer la estabilidad ni la funcionalidad.
- Colaboración en los equipos: los equipos trabajan de manera mas integra y eficiente.
- Fiabilidad: Menos errores en producción por un mayor cobertura de pruebas.
- Escalabilidad: Procesos que se adaptan fácilmente al crecimiento del proyecto.
- Seguridad: Cumplimiento normativo y control automatizado de riesgos.
- Satisfacción del cliente: Respuesta rápida a las necesidades del usuario y confianza del producto.

Creación del pipeline:

Para crear el pipeline lo primero es definir las herramientas y el entorno de trabajo a usar. En este caso, utilizare el repositorio de código **GitLab,** el cual incluye **Docker** como entorno construido por defecto, permitiéndonos especificar qué imagen queremos usar. Ademas, el **IDE** que utilizaré es **Visual Studio Code**.

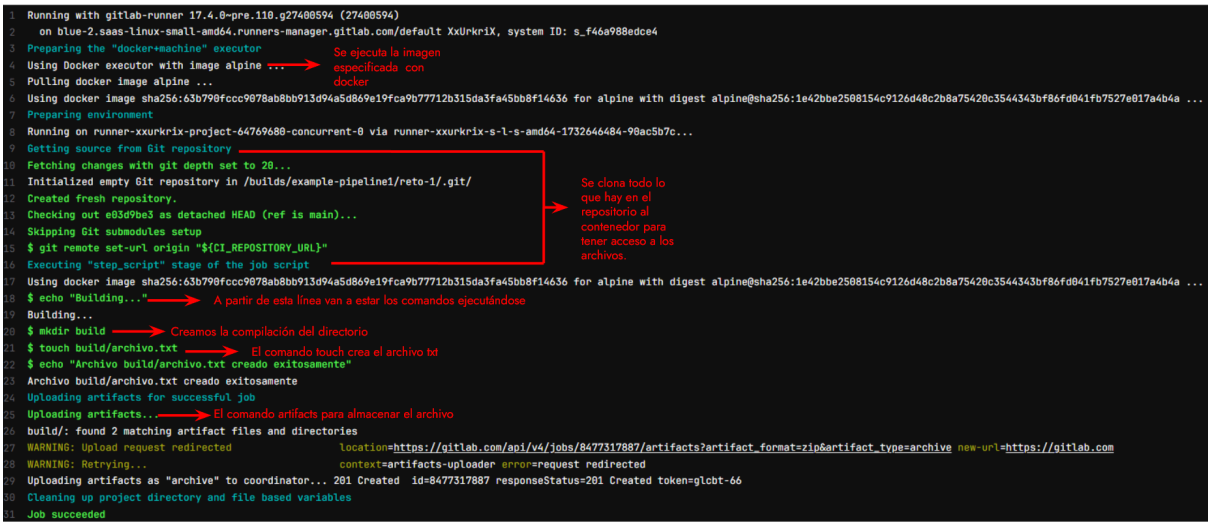


Script del pipeline con sus trabajos realizados:



Stage 1: Job - Build

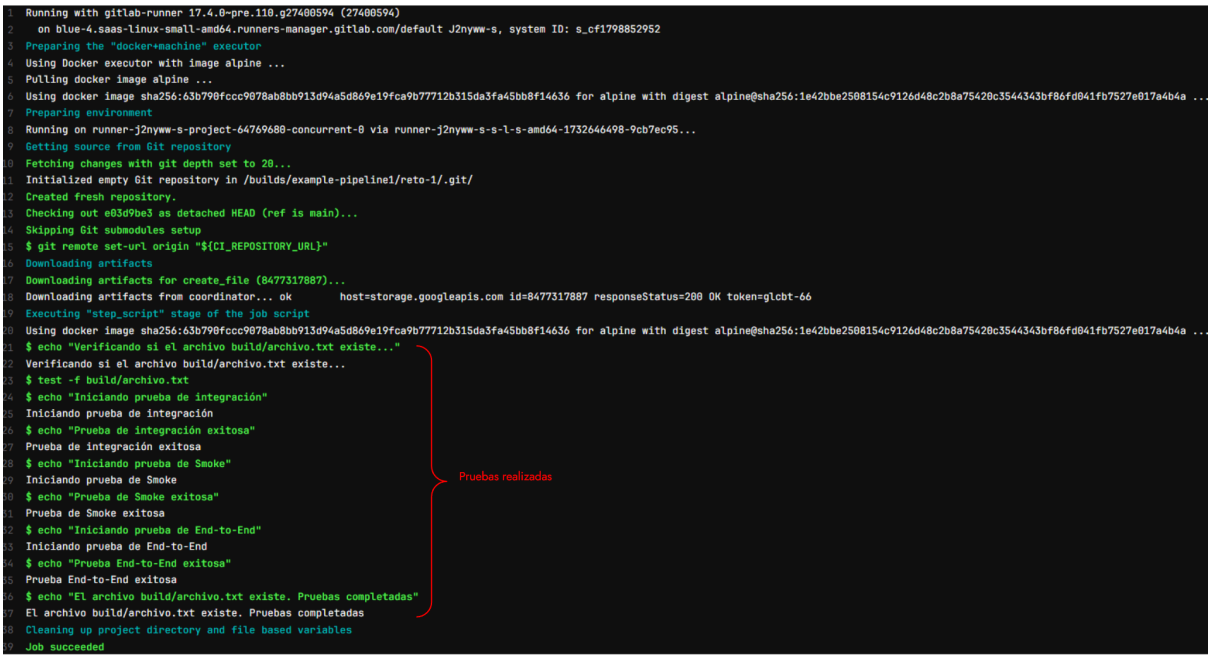
La tarea del job es crear un directorio de trabajo llamado build, luego genera un archivo llamado archivo.txt dentro del directorio. Este directorio se guarda como artefacto, lo que significa que estará disponible para las etapas posteriores.




```
Running with gitlab-runner 17.4.0~pre.110.g27400594 (27400594)
  on blue-2.saas-linux-small-amd64.runners-manager.gitlab.com/default XxUrkriX, system ID: s_f46a988edce4
Preparing the "docker+machine" executor
Using Docker executor with image alpine ...
Pulling docker image alpine ...
Using docker image sha256:63b790fccc9078ab8bb913d94a5d869e19fca9b77712b315da3fa45bb8f14636 for alpine with digest alpine@sha256:1e42bbe2508154c9126d48c2b8a75420c3544343bf86fd041fb7527e017a4b4a ...
Preparing environment
Running on runner-xxurkrix-project-64769680-concurrent-0 via runner-xxurkrix-s-l-s-amd64-1732646484-90ac5b7c...
Getting source from Git repository
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/example-pipeline1/reto-1/.git/
Created fresh repository.
Checking out e03d9be3 as detached HEAD (ref is main)...
Skipping Git submodules setup
$ git remote set-url origin "${CI_REPOSITORY_URL}"
Executing "step_script" stage of the job script
Using docker image sha256:63b790fccc9078ab8bb913d94a5d869e19fca9b77712b315da3fa45bb8f14636 for alpine with digest alpine@sha256:1e42bbe2508154c9126d48c2b8a75420c3544343bf86fd041fb7527e017a4b4a ...
$ echo "Building..."
Building...
$ mkdir build
$ touch build/archivo.txt
$ echo "Archivo build/archivo.txt creado exitosamente"
Archivo build/archivo.txt creado exitosamente
Uploading artifacts for successful job
Uploading artifacts...
build/: found 2 matching artifact files and directories
WARNING: Upload request redirected                  location=https://gitlab.com/api/v4/jobs/8477317887/artifacts?artifact_format=zip&artifact_type=archive new-url=https://gitlab.com
WARNING: Retrying...                                context=artifacts-uploader error=request redirected
Uploading artifacts as "archive" to coordinator... 201 Created  id=8477317887 responseStatus=201 Created token=glcbt-66
Cleaning up project directory and file based variables
Job succeeded
```

Stage 2: Job - Test

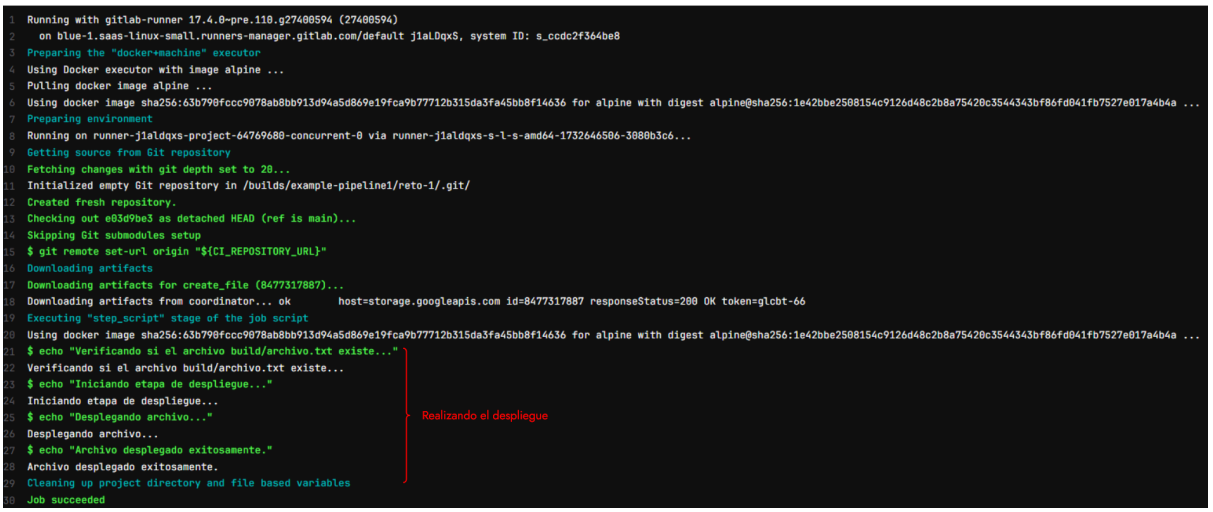
En esta etapa se ejecutan las pruebas automatizadas para garantizar que el sistema funcione según lo esperado, también se valida la existencia del archivo y se simulan las diferentes pruebas.




```
Running with gitlab-runner 17.4.0~pre.110.g27400594 (27400594)
  on blue-4.saas-linux-small-amd64.runners-manager.gitlab.com/default J2nyww-s, system ID: s_cf1798852952
Preparing the "docker+machine" executor
Using Docker executor with image alpine ...
Pulling docker image alpine ...
Using docker image sha256:63b790fccc9078ab8bb913d94a5d869e19fca9b77712b315da3fa45bb8f14636 for alpine with digest alpine@sha256:1e42bbe2508154c9126d48c2b8a75420c3544343bf86fd041fb7527e017a4b4a ...
Preparing environment
Running on runner-j2nyww-s-project-64769688-concurrent-0 via runner-j2nyww-s-s-l-s-amd64-1732646498-9cb7ec95...
Getting source from Git repository
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/example-pipeline1/reto-1/.git/
Created fresh repository.
Checking out e03d9be3 as detached HEAD (ref is main)...
Skipping Git submodules setup
$ git remote set-url origin "${CI_REPOSITORY_URL}"
Downloading artifacts
Downloading artifacts for create_file (8477317887)...
Downloading artifacts from coordinator... ok        host=storage.googleapis.com id=8477317887 responseStatus=200 OK token=glcbt-66
Executing "step_script" stage of the job script
Using docker image sha256:63b790fccc9078ab8bb913d94a5d869e19fca9b77712b315da3fa45bb8f14636 for alpine with digest alpine@sha256:1e42bbe2508154c9126d48c2b8a75420c3544343bf86fd041fb7527e017a4b4a ...
$ echo "Verificando si el archivo build/archivo.txt existe..."
Verificando si el archivo build/archivo.txt existe...
$ test -f build/archivo.txt
$ echo "Iniciando prueba de integración"
Iniciando prueba de integración
$ echo "Prueba de integración exitosa"
Prueba de integración exitosa
$ echo "Iniciando prueba de Smoke"
Iniciando prueba de Smoke
$ echo "Prueba de Smoke exitosa"
Prueba de Smoke exitosa
$ echo "Iniciando prueba de End-to-End"
Iniciando prueba de End-to-End
$ echo "Prueba End-to-End exitosa"
Prueba End-to-End exitosa
$ echo "El archivo build/archivo.txt existe. Pruebas completadas"
El archivo build/archivo.txt existe. Pruebas completadas
Cleaning up project directory and file based variables
Job succeeded
```

Stage 3 : Job - Deploy

En esta última etapa se simula el despliegue del sistema.



Conclusión

Al realizar esta actividad de investigación me permitió comprender y aplicar los fundamentos básicos de un pipeline en CI/CD destacando como este enfoque de automatización mejora significativamente la eficiencia y la confiabilidad de los proyectos de software. Al automatizar cada "jobs" por secuencia garantiza que cada cambio se pruebe e integre de manera exhaustiva, minimizando riesgos y errores. Esto no sólo acelera el ciclo de desarrollo, sino que también mantiene altos estándares de calidad del código garantizando los resultados positivos del proyecto.

Por otro lado puedo decir que se destaca la importancia de seleccionar y aplicar el uso de las herramientas correctas, ya que existen múltiples herramientas disponibles con características específicas y que pueden adaptarse mejor según las necesidades del proyecto.

Recomendaciones a tener en cuenta:

- Elegir herramientas adecuadas que se integren fácilmente con el flujo de trabajo y compatibilidad con el proyecto.
- Priorizar la documentación y que sea entendible para el que la va a leer.
- Fomentar una cultura de calidad en el equipo de trabajo involucrando a todos los miembros.
- Utilizar entornos aislados para asegurar que cada etapa del pipeline cuando se ejecute evite problemas.

Bibliografía

Unity:
https://unity.com/es/topics/what-is-ci-cd

Red hat:
https://www.redhat.com/es/topics/devops/what-is-ci-cd
https://www.redhat.com/en/topics/devops/what-cicd-pipeline
https://www.redhat.com/es/topics/containers/what-is-docker

atlassian:
https://www.atlassian.com/es/devops/devops-tools/test-automation

Mulesoft:
https://www.mulesoft.com/es/resources/api/continuous-integration-continuous-delivery

DataDog:
https://docs.datadoghq.com/es/continuous_testing/cicd_integrations/

Pipeline:
https://about.gitlab.com/es/topics/ci-cd/cicd-pipeline/

Videos:
https://www.youtube.com/watch?v=K1s6qbHatrw&t=199s
https://www.youtube.com/watch?v=z7nLsJvEyMY
https://www.youtube.com/watch?v=2V-pXM26NvE&t=1s