Laboratory Practice Report

Version control (git) and continuous deployment (CI/CD)

9 de Abril de 2025

Departamento de Electrónica, Sistemas e Informática (DESI)

Cloud Architecture *(Arquitectura en la Nube)*

Mtro. Rodolfo Luthe Ríos



Michel Maris Mora

714291

Maestria en Sistemas Computacionales

*Keywords*

# Introduction

In this section, the objective of the practice, and the expected results should be explained concisely.

# Theoretical Framework

El uso de metodologías ágiles busca incrementar la eficiencia en el proceso de creación de proyectos, en este caso comienza a utilizar despliegue continuo para maximizar la entrega de resultados en un corto tiempo explicado a más profundidad durante una conferencia Arachchi y Perera [1] Integración continua y automatización de canalización de entrega continua para la gestión ágil de proyectos de software. La adaptación de prácticas ágiles permite flexibilidad, eficiencia y velocidad del ciclo de vida de desarrollo de software (SDLC), que atrae a las empresas de desarrollo de software. Según el manifiesto Agile, los doce principios han definido la integridad del proceso y las prácticas y la gestión de proyectos Agile, que se aplican en metodologías como Extreme Programming (XP), Scrum, Kanban, Crystal, Lean Software Development (LSD) y Feature-Driven Development (FDD). Las principales ventajas de las prácticas de IC son la reducción del riesgo y la ausencia de errores y fiabilidad del software, lo que elimina la barrera de la entrega frecuente. La aceleración del tiempo de comercialización, la mejora de la calidad del producto, la mayor satisfacción del cliente, la fiabilidad de la entrega, la mejora de la productividad y la eficiencia son ventajas clave que motivan a las empresas a invertir en CD. Dado que la mayoría de los desarrollos de software y móviles se despliegan en infraestructura como servicio (IaaS), el CICD también se ha convertido en parte esencial de la computación en nube.

El versionamiento mediante uso de git u otras tecnologías se ha vuelto una parte vital en implementaciones CICD, explicado por Charanjot, et al. [2] Uno de los factores clave para implementar la arquitectura de microservicios es la virtualización. La virtualización puede implementarse utilizando máquinas virtuales y la contenedorización. CI es un segmento de prácticas involucradas en los principios de programación de software. CI establece que todo el código de una aplicación debe mantenerse en un repositorio común para que cada vez que el desarrollador verifique el código en el repositorio, se active una secuencia de comandos que recoge el código más reciente del repositorio, lo integra con el código existente y ejecuta los casos de prueba diseñados de acuerdo con la aplicación. Existen múltiples herramientas de integración continua disponibles en el mercado, como Jenkins, Bamboo, Gitlab, Subversion, etc. Para que CI funcione tiene que estar integrado con un sistema de gestión de código fuente. Git es una herramienta común ampliamente utilizada por los desarrolladores para mantener las versiones del código. Cada desarrollador trabaja en su propia rama, que se extrae de una rama maestra. Cada vez que un desarrollador implementa una funcionalidad o corrige un error, presenta una solicitud de fusión de la rama actual a la rama maestra. Se activa un script para ejecutar los casos de prueba que ejecuta todas las pruebas unitarias escritas y, si el código supera todos los casos de prueba, la fusión se completa; de lo contrario, la fusión se rechaza. Cada desarrollador tiene un papel muy responsable a la hora de fusionar su código en la rama principal. Esto añade un sentido de responsabilidad al desarrollador, que debe asegurarse de que sus cambios no afecten a la compilación y no obstaculicen el trabajo de los demás desarrolladores. Para evitar los conflictos de fusión, se prefieren las ramas de vida corta con características pequeñas a las ramas de vida larga con características mayores.

# Architectural diagram

Diagram of the implemented architecture.

Should have reference: Fig. #. Architectural diagram

# Practice Development

### Version control

**Set up a local repository**

1. Install the git client
   1. [https://git-scm.com](https://git-scm.com/)
2. Configure git with your iteso account, substitute for you name and iteso email
   1. git config --global user.name "John Doe"
   2. git config --global user.email johndoe@example.com
3. Set up the repository
   1. Create a folder called git in My Documents
   2. From the console change to that directory (or in Windows right click on the folder and select “Git Bash here”)
      1. git init
4. Version control
   1. Create a file in the git directory: versions.txt with the content: Version 1
   2. Add the file to the git tracker, from the console in the folder: git add .
      1. The “.” is important to include all files in tracking
   3. Make a commit of the changes: git commit -m “first version”
   4. Edit the file and change the content to: Version 2
      1. Add and commit the changes:
         1. git add .
         2. git commit -m “second version”
   5. Do the same for Version 3

Texto

El contenido generado por IA puede ser incorrecto.

* 1. Check version history: git log

Texto

El contenido generado por IA puede ser incorrecto.

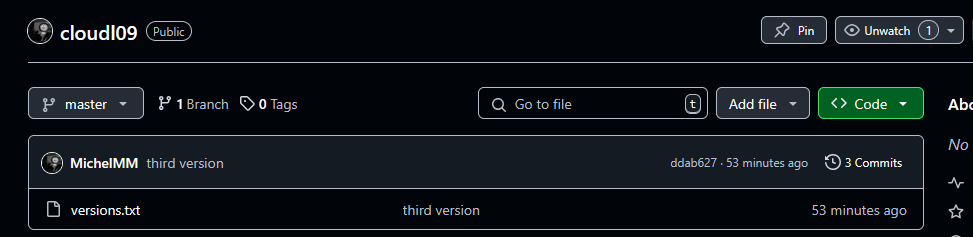
**Create a repository on GitHub**

1. Create an account with your iteso email on GitHub
2. Create a repository on GitHub
3. Check the repository URL
4. Add the GitHub repository to the local repository, from the git folder:
   1. git remote add Hub <URL>
   2. git push Hub master

Texto

El contenido generado por IA puede ser incorrecto.

1. Refresh the GitHub repository screen. What change do you see?



**Control versions of a document**

1. Create the report document, and add it for tracking
2. Make add and commit at the end of each section of the report -Framework, Diagram, etc- with a corresponding comment
3. Show log of changes to the document

### Continuous deployment environment

**Demo application**

1. Download Node.js sample applications used by Elastic Beanstalk <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/tutorials.html>

**Create Elastic Beanstalk environment**

1. In Elastic Beanstalk’s dashboard create a new application, for example: AN-cicdlab
2. Create environment
   1. Web server enviroment
   2. Plataform Node.js
   3. Upload your code: the zip that was downloaded from the demo application it simulates that it is your code
   4. Create
3. Test the URL of the environment, you should see the green web page of the demo application
4. Do not close the Elastic Beanstalk console page

**Create repository on GitHub**

1. Create a GitHub repository for practice, for example: AN-cicd-lab
2. Unzip *sample applications used by Elastic Beanstalk* zip file and upload the files to the GitHub repository

**Configure CodePipeline**

1. In CodePipeline’s dashboard create pipeline, for example: AN-cicd-lab
   1. Keep defaults
2. SourceProvider: GitHub
   1. Connect to GitHub account
   2. Paste full name of the repository created on GitHub
   3. Use master branch
   4. Keep GitHub webhooks
3. Skip Build stage
4. Deploy stage in AWS Elastic Beanstalk
   1. Select Application and Environment created previously in Elastic Beanstalk
5. Create pipeline
   1. Takes 5 to 10 minutes
6. Open Beanstalk’s dashboard in another window
   1. Test the URL of the environment, you should see the green web page of the demo application
7. Don't close the PipeLine console page

**Check automated deployment**

1. Open the application repository on GitHub
2. Edit the index.html file on GitHub and change the background color to Blue –like in

Laboratory 7: Automatically deploy and scale applications

* 1. Select Commit changes at the bottom of the page
  2. The change should be sent to our environment

1. Return to the CodePipeline console
   1. Refresh screen
   2. It should be seen that Deploy is doing a new deploy
   3. In 5 to 10 minutes the change should be reflected in the environment
      1. Deploy will be with a green check mark
2. Return to Elastic Beanstalk’s dashboard
   1. Test the URL of the environment, you should see the blue web page of the demo application
      1. If not, refresh the page or clear the cache for the last hour, check again

**Clean environment**

To avoid running out of credit, delete all environments

1. In CodePipeline’s dashboard
   1. Edit pipeline / Delete
   2. Confirm that it was deleted
2. In Elastic Beanstalk’s dashboard
   1. Actions / Terminate Environment
   2. Confirm that it was deleted
   3. In Environment / Delete application
   4. Confirm that it was deleted
3. You can delete the GitHub repository

# Problems and Solutions

List all the problems encountered during the development of the practice and how they were resolved.

# Experiments and Results

Do you find it convenient to use version control for documents that are not code?

Do you think you could use CI/CD for your school programs?

# Cost analysis

Explain the cost of the implemented solution, justifying the chosen solution based on costs. Should detail monthly and annual costs.

# Conclusions

Conclusions should be a reflective work presenting the knowledge gained from the experiments, results, and the theoretical framework presented. Conclusions should be brief and should be consistent with the introduction (1 or 2 paragraphs).

Invalid conclusions include: "I learned a lot!", "I really liked the practice", "everything worked correctly."

References

In IEEE style.