Índice de figuras

v

Índice de figuras

de pasar satisfactoriamente la compilación y ejecución de los tests entonces se genera una nueva imagen docker con la última versión del codigo compilado y se versiona en Artifact Registry.

2.2.5. Visual Studio Code

Visual Studio Code [33] es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

2.2.6. Sistema operativo Ubuntu

Ubuntu [34] es una distribución Linux basada en Debian GNU/Linux y patrocinado por Canonical, que incluye principalmente software libre y de código abierto. Puede utilizarse en ordenadores y servidores, está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario.

14 Capítulo 2. Introducción específica

de pasar satisfactoriamente la compilación y ejecución de los tests entonces se genera una nueva imagen Docker con la última versión del codigo compilado y se versiona en Artifact Registry.

2.2.5. Visual Studio Code

Visual Studio Code [33] es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

2.2.6. Sistema operativo Ubuntu

Ubuntu [34] es una distribución Linux basada en Debian GNU/Linux y patrocinado por Canonical, que incluye principalmente software libre y de código abierto. Puede utilizarse en ordenadores y servidores, está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario.

3.4. Plataforma de desarrollo y ciclo de CI/CD

Durante el ciclo de desarrollo, se utilizó la infraestructura de CI formada por las herramientas descritas en el capítulo anterior. En la figura 3.15 se puede apreciar su arquitectura.

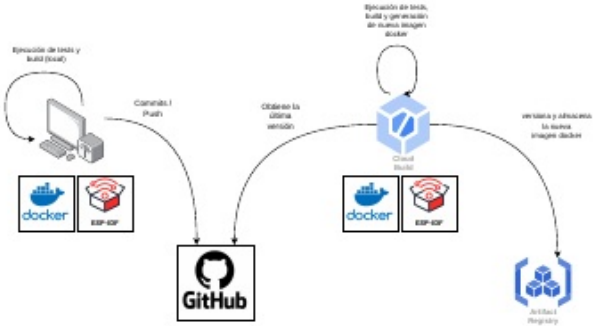

FIGURA 3.15. Plataforma de CI/CD utilizada.

Para cada prototipo desarrollado se creó una imagen Docker, extendiendo la de espressif/idf [52]. El conjunto de actividades del mismo fue el siguiente:

1. Codificar localmente en Ubuntu utilizando VSCode.
2. Construcción local en Ubuntu de imagen Docker, de acuerdo a la especificación de los siguientes pasos en el archivo `docker-compose.yml`:
   a) Compilación del código, enlazado de bibliotecas y empaquetado de la aplicación.
   b) Ejecución de los tests unitarios con *ceedling*.
   c) Despliegue (flash) de la aplicación en el ESP32.
3. Versionado del código en el repositorio GitHub por medio de los comandos `git commit` y `git push`.
4. Construcción en el ambiente de CI/CD por medio de Google Cloud Build, de acuerdo a la especificacion de los siguientes pasos definidos en el archivo `cloudbuild.yml`:
   a) Compilación del código, enlazado de bibliotecas y empaquetado de la aplicación.
   b) Ejecución de los tests unitarios con *ceedling*.
   c) Construcción de imagen docker.
   d) *Tagging* y versionado de imagen docker en Google Artifact Registry.

A continuación, se pueden apreciar capturas de pantallas de cada uno de los sistemas utilizados y los pasos ejecutados. En la imagen 3.16, se puede apreciar la salida por consola tras la ejecución de los tests unitarios y construcción de la imagen Docker de manera local.

3.4. Plataforma de desarrollo y ciclo de CI/CD 27

3.4. Plataforma de desarrollo y ciclo de CI/CD

Durante el ciclo de desarrollo, se utilizó la infraestructura de CI formada por las herramientas descritas en el capítulo anterior. En la figura 3.15 se puede apreciar su arquitectura.

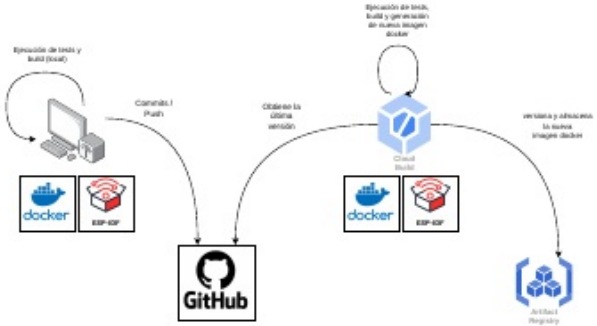

FIGURA 3.15. Plataforma de CI/CD utilizada.

Para cada prototipo desarrollado se creó una imagen Docker, extendiendo la de espressif/idf [52]. El conjunto de actividades del mismo fue el siguiente:

1. Codificar localmente en Ubuntu utilizando VSCode.
2. Construcción local en Ubuntu de imagen Docker, de acuerdo a la especificación de los siguientes pasos en el archivo `docker-compose.yml`:
   a) Compilación del código, enlazado de bibliotecas y empaquetado de la aplicación.
   b) Ejecución de los tests unitarios con *ceedling*.
   c) Despliegue (flash) de la aplicación en el ESP32.
3. Versionado del código en el repositorio GitHub por medio de los comandos `git commit` y `git push`.
4. Construcción en el ambiente de CI/CD por medio de Google Cloud Build, de acuerdo a la especificacion de los siguientes pasos definidos en el archivo `cloudbuild.yml`:
   a) Compilación del código, enlazado de bibliotecas y empaquetado de la aplicación.
   b) Ejecución de los tests unitarios con *ceedling*.
   c) Construcción de imagen Docker.
   d) *Tagging* y versionado de imagen Docker en Google Artifact Registry.

A continuación, se pueden apreciar capturas de pantallas de cada uno de los sistemas utilizados y los pasos ejecutados. En la imagen 3.16, se puede apreciar la salida por consola tras la ejecución de los tests unitarios y construcción de la imagen Docker de manera local.

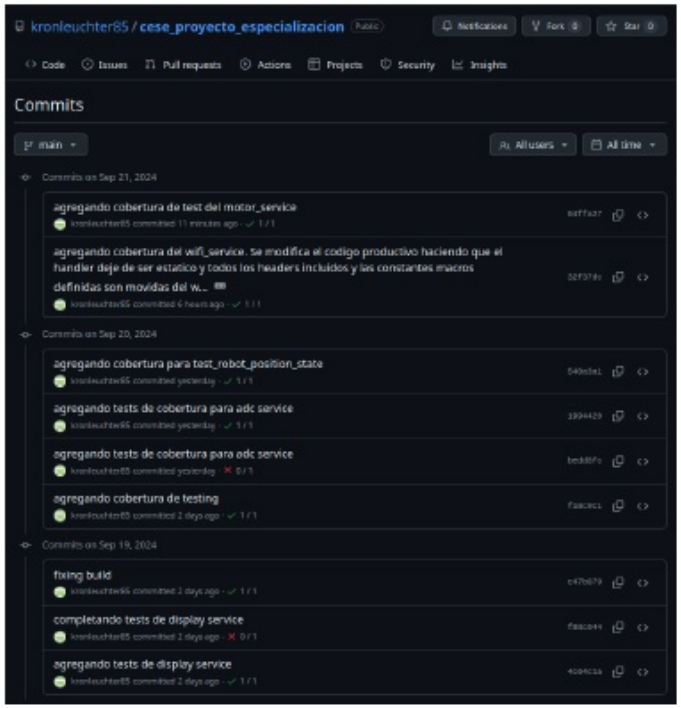

FIGURA 3.17. Listado de *commits* en Github.

En la figura 3.18, se pueden apreciar los diferentes builds disparados en Cloud Build referenciando los commits de GitHub.

3.4. Plataforma de desarrollo y ciclo de CI/CD 29

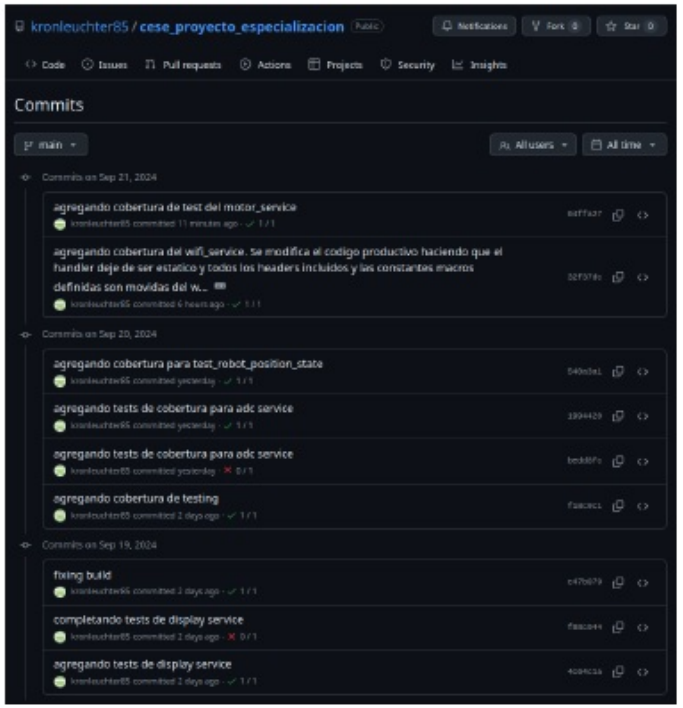

FIGURA 3.17. Listado de *commits* en GitHub.

En la figura 3.18, se pueden apreciar los diferentes builds disparados en Cloud Build referenciando los commits de GitHub.

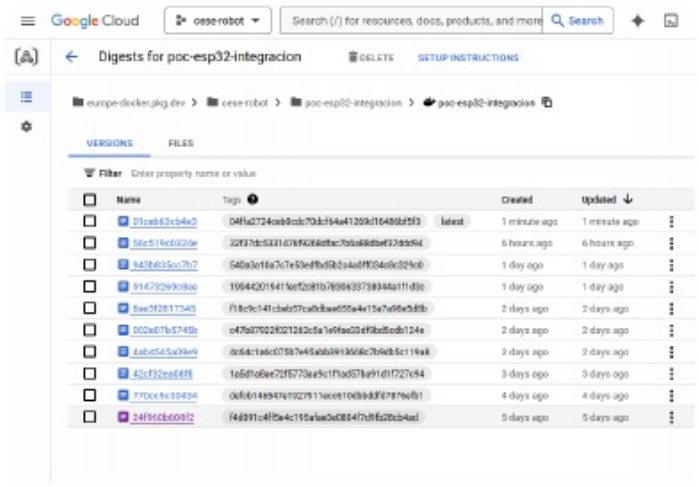

FIGURA 3.19. Listado de versiones de imágenes doker en Google ArtifactRegistry.

3.5. Reportes de ejecución y cobertura de testing unitario

A continuación, se presentan los reportes de testing generados por la herramienta *ceedling* con el complemento *gcov* donde se puede apreciar el nivel de cobertura logrado para cada servicio.

En la figura 3.20, se puede apreciar la salida por pantalla tras la ejecución local de ceedling con el plugin de cobertura, en donde se evidencia la cantidad de test cases.

3.5. Reportes de ejecución y cobertura de testing unitario 31

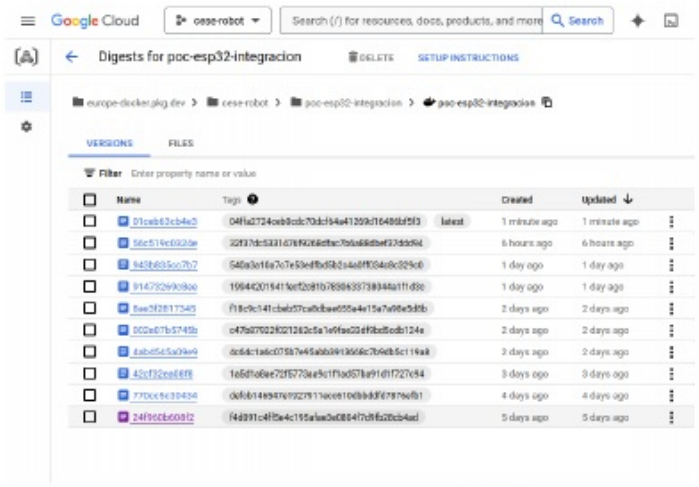

FIGURA 3.19. Listado de versiones de imágenes Docker en Google ArtifactRegistry.

3.5. Reportes de ejecución y cobertura de testing unitario

A continuación, se presentan los reportes de testing generados por la herramienta *ceedling* con el complemento *gcov* donde se puede apreciar el nivel de cobertura logrado para cada servicio.

En la figura 3.20, se puede apreciar la salida por pantalla tras la ejecución local de ceedling con el plugin de cobertura, en donde se evidencia la cantidad de test cases.