



Máster Universitario de Investigación en Ingeniería De Software Y Sistemas Informáticos

31105151 - Trabajo Fin de Máster

Título

Generative Cloud Manager: Code-Runner

Autor

César Hugo Bárzano Cruz

Tutor

Rubén Heradio

—
2019/2020

Máster Universitario de Investigación en Ingeniería De Software Y Sistemas Informáticos

**31105151 - Trabajo Fin de Máster - Generative Cloud
Manager: Code-Runner**

Trabajo Tipo A Propuesto por Alumno

Autor
César Hugo Bárzano Cruz

Tutor
Rubén Heradio



—
2019/2020

**DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO
CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE
MASTER**

Fecha: 30/Mayo/2020

Quién suscribe:

Autor(a): César Hugo Bárzano Cruz
D.N.I/N.I.E/Pasaporte.: 77138361H

Hace constar que es la autor(a) del trabajo:

Generative Cloud Manager: Code-Runner

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.

César Hugo Bárzano Cruz



**Impreso TFdM05_AutorPbl. Autorización de publicación
y difusión del TFM para fines académicos**

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

César Hugo Bárzano Cruz

Resumen

Este proyecto demuestra como el uso de conceptos ligados a la Programación Generativa y la integración de diversos servicios web pueden utilizarse de manera conjunta para sintetizar, generalizar y automatizar el proceso de desarrollo y operación de aplicaciones software. La lectura aquí propuesta, junto con el sistema software resultante abordan el proceso de desarrollo de aplicaciones software mediante un enfoque generativo, con esto se pretende proporcionar al desarrollador software de un sistema web orientado a la generación, construcción y despliegue de aplicaciones software, concretamente de aplicaciones web o servicios de datos en la nube.

En este proyecto se considera que las etapas que forman el ciclo de vida o proceso de desarrollo de aplicaciones software está altamente segmentado, por lo que presenta una solución generativa y automatizada al proceso de desarrollo software. Con este proyecto no se pretende desestimar a los procesos de desarrollo software tradicionales, lo que intenta es demostrar como la unificación de los conceptos comunes entre aplicaciones web de diversa naturaleza así como la síntesis y automatización genérica de las etapas básicas de producción software pueden dar lugar a soluciones funcionales ahorrando al desarrollador numerosas horas de trabajo. De igual forma, el sistema resultante puede ser aprovechado por el desarrollador inexperto como punto de partida en su aprendizaje de las denominadas tecnologías de la información.

Este proyecto pone de manifiesto que ante la gran diversidad de tecnologías y soluciones ofrecidas por la Cloud Computing existen definiciones, conceptos y etapas básicas que son comunes y transversales a todas ellas, independientemente de su naturaleza, dominio y propósito. Este proyecto sintetiza el proceso de desarrollo software en 3 etapas fundamentales:

1. **Code Generation.** Etapa inicial donde se sintetiza el proceso de desarrollo software, es decir, se produce el código necesario para la solución especificada por el desarrollador.
2. **Artifact Generation,** El objetivo de esta etapa es aprovechar el código fuente generado en la etapa Code Generation para producir un artefacto auto-contenido junto con la configuración necesaria para su ejecución en un entorno virtual.
3. **Deploy Generation.** En esta etapa se aprovecha el producto de la etapa Artifact Generation para disponibilizar la solución en Internet.

En los siguientes capítulos se profundiza en estas ideas, detallando cuales son los objetivos de alto nivel o casos de uso que se esperan del sistema. Se describirá el problema planteado y se analizará en detalle la solución alcanzada.

Palabras Clave: Cloud Computing, Programación Generativa, Desarrollo Software, Código Fuente, Automatización, Integración, Despliegue, Aplicaciones, Servicios, Contenedores.

Abstract

This project shows how using concepts of Generative Programming and the integration of web services can work together to synthesize, generalize and automate the process of software development and software operation. This project addresses the process of developing software applications through a generative approach, with the target of providing a web system oriented to software generation, artefact construction and applications deployment, specifically from web applications or cloud data services.

This project considers that the stages of software applications developments are highly segmented, so this project presents a generative and automated solution to the software development. This project is not intended to discredit traditional software development but rather to show how the unification of common concepts among web applications kinds as well as the generic synthesis and automation of the basic stages of software production can lead to functional solutions saving developers from large amounts of work. In the same line, the system can be used by the junior developers as a starting point in their learning path of the information technologies.

This project shows that given the great diversity of technologies and solutions offered by Cloud Computing, there are definitions, concepts and basic stages that are common and transversal to all of them, regardless of their nature, domain or purpose. This project synthesizes the software development process in 3 fundamental stages:

1. **Code Generation.** Initial stage where the software development process is synthesized. Source code necessary for the solution specified by the developer is produced.
2. **Artefact Generation** The goal of this stage is to use the source code generated in the Code Generation stage to produce a self-contained artifact along with the necessary setup for its execution in a virtual environment.
3. **Deploy Generation** Final stage where the product of the Artefact Generation stage is used to run the solution and make it available to developers over Internet.

The following chapters delve into these ideas, describing project objectives and use cases expected from the system. The problem will be officially presented and the solution reached will be analyzed in detail.

Keywords: Cloud Computing, Generative programming, Software Development, Source Code, Automation, Integration, Deployment, Applications, Services, Docker.

Agradecimientos

Al Amor de mi vida, donde quiera que esté.

Índice general

Índice de figuras	17
1. Introducción	21
1.1. Motivación	21
1.2. Objetivos	21
1.3. Problema	22
1.4. Solución	24
2. Trabajos Relacionados	27
2.1. Desarrollo Software	27
2.1.1. Ciclo de Vida del Desarrollo Software	27
2.1.2. Desarrollo Software y Buenas Prácticas	28
2.1.3. En este Proyecto	29
2.2. Programación Generativa	30
2.2.1. Principios	31
2.2.2. En este proyecto	32
2.3. Cloud Computing	33
2.3.1. Tipos de Nube	34
2.3.2. Tipos de Servicios en la Nube	34
2.3.3. Tecnologías en el Mercado	36
2.3.4. En este Proyecto	39
3. Planificación y Análisis	41
3.1. Planificación	41
3.2. Análisis Software	42
3.2.1. Requisitos de Información	43
3.2.2. Requisitos Funcionales	43
3.2.3. Requisitos No Funcionales	45
3.2.4. Casos de Uso	46
3.2.5. Diagramas de Secuencia	47

4. Implementación y Pruebas	53
4.1. Implementación y Desarrollo	53
4.1.1. Back-End	54
4.1.2. Front-End	58
4.1.3. Infraestructura Virtual	59
4.1.4. Seguridad	60
4.2. Caso de Uso	61
4.2.1. Autenticación y Autorización	61
4.2.2. Generando Aplicaciones	64
4.2.3. Consumiendo Aplicaciones	77
4.2.4. Actualizando Aplicaciones	83
5. Conclusiones	101
5.1. Mejoras y Trabajos Futuros	101
5.1.1. Proceso Generativo	101
5.1.2. Naturaleza y Tecnología	102
5.1.3. Testing	103
5.1.4. Material Didáctico	103
5.1.5. Interfaz de Sistema Cloud	103
6. Anexo	105
6.1. Ficheros de Ejemplo	105
Bibliografía	107

Índice de figuras

2.1. Tabla Comparativa: Tecnología VS. Funcionalidad	39
3.1. Planificación: Definición del Proyecto	41
3.2. Planificación: Análisis y Desarrollo Inicial	42
3.3. Planificación: Desarrollo Formal	42
3.4. Planificación: Documentación Formal	42
3.5. Casos de uso - Gestión Aplicaciones I [1]	46
3.6. Casos de uso - Gestión Aplicaciones II [1]	47
3.7. Casos de uso - Gestión Credenciales [1]	47
3.8. Diagramas de Secuencia - Inicio sesión [1]	48
3.9. Diagramas de Secuencia - Crear Aplicación [1]	49
3.10. Diagramas de Secuencia - Ejecutar, Inspeccionar y Detener Aplicación [1]	50
3.11. Diagramas de Secuencia - Eliminar Aplicación [1]	51
4.1. Modelo-Vista-Controlador [?]	53
4.2. Virtualización Docker [2]	57
4.3. Autenticación y Autorización: Index	62
4.4. Autenticación y Autorización: Autenticación	62
4.5. Autenticación y Autorización: Autorización	63
4.6. Generando Aplicaciones: Formulario de Creación	64
4.7. Generando Aplicaciones: Naturalezas	65
4.8. Generando Aplicaciones: Single-Page	66
4.9. Generando Aplicaciones: Single-Page	66
4.10. Generando Aplicaciones: Single-Page Repositorio	67
4.11. Generando Aplicaciones: Single-Page CI	68
4.12. Generando Aplicaciones: Single-Page Compilación	68
4.13. Generando Aplicaciones: Single-Page CI Completado	69
4.14. Generando Aplicaciones: Single-Page Release	70
4.15. Generando Aplicaciones: Single-Page Artefacto	70
4.16. Generando Aplicaciones: Single-Page Desplegada	71
4.17. Generando Aplicaciones: Single-Page Consultada	71

4.18 Generando Aplicaciones: Api-Rest Tecnología	72
4.19 Generando Aplicaciones: Api-Rest Modelo de Datos	73
4.20 Generando Aplicaciones: Data-Service Tecnología	74
4.21 Generando Aplicaciones: DevOps-Service Tecnología	74
4.22 Generando Aplicaciones: Workspace I	76
4.23 Generando Aplicaciones: Workspace II	76
4.24 Consumiendo Aplicaciones: Api-Rest Stop	77
4.25 Consumiendo Aplicaciones: Api-Rest Run	78
4.26 Consumiendo Aplicaciones: Api-Rest View	78
4.27 Consumiendo Aplicaciones: Api-Rest View Logs Index	79
4.28 Consumiendo Aplicaciones: Api-Rest View Logs UI	79
4.29 Consumiendo Aplicaciones: Api-Rest View Logs POST	80
4.30 Consumiendo Aplicaciones: Api-Rest View Logs DELETE	80
4.31 Consumiendo Aplicaciones: Data-Service View Logs	81
4.32 Consumiendo Aplicaciones: Data-Service View Logs I	81
4.33 Consumiendo Aplicaciones: Data-Service View Logs II	82
4.34 Consumiendo Aplicaciones: Data-Service View Logs III	82
4.35 Consumiendo Aplicaciones: Data-Service View Logs IV	83
4.36 Actualizando Aplicaciones: Single-Page en Ejecución	84
4.37 Actualizando Aplicaciones: Single-Page Consulta	84
4.38 Actualizando Aplicaciones: Single-Page Clonar I	85
4.39 Actualizando Aplicaciones: Single-Page Clonar II	85
4.40 Actualizando Aplicaciones: Single-Page CSS	86
4.41 Actualizando Aplicaciones: Single-Page CI Start	87
4.42 Actualizando Aplicaciones: Single-Page CI Finish	87
4.43 Actualizando Aplicaciones: Single-Page Release 0.0.1	88
4.44 Actualizando Aplicaciones: Single-Page Artefacto 0.0.1	88
4.45 Actualizando Aplicaciones: Single-Page Stop 0.0.0	89
4.46 Actualizando Aplicaciones: Single-Page Run 0.0.1	89
4.47 Actualizando Aplicaciones: Single-Page Consulta 0.0.1	89
4.48 Trabajo Local: Show Access Token	90
4.49 Trabajo Local: Access Token	90
4.50 Trabajo Local: Help	91
4.51 Trabajo Local: Single-Page Actualización CSS	91
4.52 Trabajo Local: Single-Page make push	92
4.53 Trabajo Local: Single-Page Release 0.0.2	92
4.54 Trabajo Local: Single-Page Artefacto 0.0.2	93
4.55 Trabajo Local: Single-Page Stop 0.0.1	93
4.56 Trabajo Local: Single-Page Run 0.0.2	94
4.57 Trabajo Local: Single-Page Consulta 0.0.2	94
4.58 Re-Generación: Single-Page 0.0.2	95

4.59 Re-Generación: Single-Page 0.0.2 Stop	95
4.60 Re-Generación: Single-Page Generating	95
4.61 Re-Generación: Single-Page Building	96
4.62 Re-Generación: Single-Page Ready	96
4.63 Re-Generación: Single-Page Release 0.0.3	97
4.64 Re-Generación: Single-Page Artefacto 0.0.3	97
4.65 Re-Generación: Single-Page 0.0.3 Running	98
4.66 Re-Generación: Single-Page 0.0.3 Consulta	98
4.67 Limpieza: Single-Page 0.0.3 Delete	98
4.68 Limpieza: Single-Page 0.0.3 Deleting	99
4.69 Limpieza: Workspace	99
5.1. Ejemplo App	104
5.2. Ejemplo Interfaz Sistema	104

Capítulo 1

Introducción

1.1. Motivación

Este proyecto surge con la motivación de demostrar como la programación generativa y la integración de diversos servicios web pueden utilizarse de manera conjunta para generalizar y automatizar el proceso de desarrollo y operaciones de aplicaciones web o servicios cloud.

Este proyecto aborda el proceso de desarrollo de aplicaciones web desde un punto de vista generativo. Con esta iniciativa se pretende acuñar el lema **From Spec to Cloud** es decir, dada una definición de aplicación específica de dominio, construir un sistema que generalice este tipo de soluciones y sea capaz de materializarlas en sistemas de información disponibles a través de Internet. De igual forma, con este proyecto se pretende dotar a los desarrolladores o clientes de un sistema que les ayude a comprender y adquirir conocimientos relativos a las tecnologías de la información y al ciclo de vida de las aplicaciones software.

A continuación se van a definir los objetivos propuestos a alcanzar siguiendo esta motivación.

1.2. Objetivos

El principal objetivo de este proyecto es el de sintetizar y automatizar el proceso de desarrollo y operación de software, concretamente el de aplicaciones web o servicios de datos en la nube mediante un enfoque generativo. Dicha síntesis consiste en generalizar la naturaleza de los distintos sistemas de información que pueden ser consumidos a través de Internet en una única entidad denominada comúnmente como aplicación.

Para alcanzar el principal objetivo, este proyecto ha de identificar,

modelar y automatizar las principales etapas del ciclo de vida de proyectos software, y construir un mecanismo genérico con el que producir aplicaciones. Las aplicaciones producidas por el proyecto han de ser auto-contenidas y auto-consumibles, es decir, han de contener todo lo necesario para ser ejecutadas y usadas fuera del marco de este proyecto.

Finalmente, este proyecto ha de definir e implementar un mecanismo genérico que permita consumir las aplicaciones producidas a través de Internet, cumpliendo así completamente con el principal objetivo.

De manera adicional, este proyecto tiene como objetivo presentar mejoras y automatizaciones relativas a los proceso de desarrollo software. El sistema resultante se presenta como una herramienta o ayuda para desarrolladores en la adquisición de conocimientos en las denominadas tecnologías de la información. Tanto el código fuente de este proyecto como las aplicaciones creadas por los usuarios que utilicen el sistema se consideran **Open Source**, es decir código abierto y disponible públicamente para quien lo necesite en el repositorio del proyecto <https://github.com/hugobarzano/GCM>.

- **OBJ-1.** Sintetizar y generar aplicaciones software de diversa naturaleza.
- **OBJ-2.** Sintetizar y generar aplicaciones software de diversa tecnología.
- **OBJ-3.** Generar aplicaciones auto-contenidas.
- **OBJ-4.** Producir y publicar el código fuente de las aplicaciones.
- **OBJ-5.** Producir y publicar paquetes o ejecutables de las aplicaciones.
- **OBJ-6.** Disponibilizar aplicaciones a través de Internet.
- **OBJ-7.** Analizar y Operar aplicaciones en ejecución.

A continuación se va a describir de manera literaria el problema planteado por este proyecto.

1.3. Problema

Como se verá en el capítulo dedicado a los trabajos relacionados, el conjunto de soluciones que ofrece la nube es muy variado. Para cada una de las soluciones cloud, se pueden encontrar diversas implementaciones

con diversas tecnologías, lo que incrementa de manera significativa el conjunto final de productos software disponibles para ser consumidos por Internet. Este es el primer problema que intenta abordar este proyecto, ¿Cómo manejar la diversidad tecnológica que produce trabajar en entornos de computación en la nube? ¿Qué solución es mas adecuada para un problema concreto de negocio? ¿Con qué tecnología implementar la solución elegida? Este proyecto aborda este problema dotando a los desarrolladores de una plataforma Open Source donde probar la tecnología que desean construir.

El conjunto de soluciones ofrecidas por los diversos proveedores de soluciones cloud aumenta día a día en función de las necesidades de negocio de los consumidores lo que obliga a los profesionales a estar continuamente renovando y certificando sus conocimientos. A raíz de esto han surgido muchas plataformas de aprendizaje o E-learning para que los profesionales adquieran conocimientos en las consideradas tecnologías de vanguardia, en otras palabras, tecnologías altamente demandadas por el mundo empresarial. Algunas de estas plataformas son:

- Linux Academy
- Coursera
- Udemy

Estas plataformas cuentan con diversos recursos virtuales para el aprendizaje (vídeos, tutoriales, laboratorios virtuales, ejercicios, test) pero ninguna de ellas materializa el tipo de software que el desarrollador desea aprender en una aplicación o servicio consumible a través de Internet siendo este uno de los objetivos a alcanzar por el proyecto aquí planteado.

Normalmente, las etapas que forman el ciclo de vida del desarrollo software están altamente segmentadas y debido a esto el tiempo y recursos necesarios para diseñar, implementar y desplegar un producto software es elevado. Este proyecto intenta mejorar esto dando un enfoque reducido, generativo y automatizado para la especificación, producción y despliegue de software en entornos de computación en la nube.

A continuación, se va a describir la solución alcanzada por este proyecto.

1.4. Solución

Este proyecto muestra como el uso de la programación generativa y las tecnologías de la información mejoran el proceso desarrollo y despliegue software en el ámbito de las soluciones orientadas a la web 2.0. Se pretende proporcionar al desarrollador software de un sistema web orientado a la generación, construcción y despliegue de aplicaciones web o servicios de datos de diversa naturaleza partiendo de una definición específica de dominio. Con esta solución se intenta que el desarrollador adquiera conceptos y conocimientos relativos a las tecnologías de la información y el ciclo de vida del desarrollo software.

Como solución se propone realizar las siguientes síntesis o generalidades. Por una parte, existe una gran diversidad de soluciones cloud:

1. **Aplicaciones web estáticas**
2. **Aplicaciones web dinámicas**
3. **Interfaces web**
4. **Api/Rest**
5. **Servicios de Datos**

Se propone sintetizar todas ellas como una única entidad **aplicación** y someter a esta entidad al mismo proceso o etapas independientemente de cual sea su naturaleza o dominio cumpliendo con el objetivo de dada una especificación de aplicación, producir el código fuente necesario para materializar dicha especificación en un producto software funcional.

La entidad de propósito general **aplicación** es definida por el siguiente conjunto de recursos:

1. **Repositorio de Código Fuente.** Lugar donde habita la aplicación.
2. **README.** Fichero descriptivo con información relativa a la aplicación.
3. **LICENCE.** Fichero descriptivo caracterizando la aplicación como Open Source.
4. **makefile.** Herramienta de construcción, permite realizar operaciones básicas a nivel de proyecto software:
 - 4.1. make setup: Instalación

- 4.2. make build: Compilación
 - 4.3. make test: Ejecución Pruebas Unitarias
 - 4.4. make run: Ejecución
 - 4.5. make pull: Descargar Artefacto
 - 4.6. make push: Crear Artefacto
5. **Código Fuente.** Representa el software en si, contiene la lógica de la aplicación.
 6. **Configuración.** Representa la configuración necesaria para ejecutar la aplicación.
 7. **Documentación.** Representa la documentación o especificación de la aplicación.
 8. **Dockerfile.** Define al artefacto necesario para ejecutar la aplicación. Dicho artefacto es una Imagen docker.

Por otra parte, para sintetizar el proceso de desarrollo y despliegue de software de manera automatizada y generativa, este proyecto define las siguientes etapas:

1. **Code Generation.** Etapa inicial donde se sintetiza el proceso de desarrollo software, es decir, se produce el código necesario para la solución especificada por el desarrollador.
2. **Artifact Generation.** El objetivo de esta etapa es aprovechar el código fuente generado en la etapa Code Generation para producir un artefacto auto-contenido junto con la configuración necesaria para su ejecución en un entorno virtual.
3. **Deploy Generation.** En esta etapa se aprovecha el producto de la etapa Artifact Generation para disponibilizar la solución en Internet.

Toda aplicación generada por el sistema es sometida a estas etapas, independientemente de su naturaleza, dominio o tecnología. Con ellas se consigue que dada una especificación de aplicación se genere y publique el código fuente que materializa la definición en producto software, transformando problemas específicos de dominio en soluciones generalizadas, la construcción de artefactos auto-contenidos que permita la distribución y ejecución del software generado, así como la disponibilización de las aplicaciones en Internet para que pueda ser consumida por los usuarios del sistema.

El resto del documento se organiza de la siguiente manera:

1. **Capítulo 2: Trabajos Relacionados.** Donde se tratan trabajos, publicaciones y tecnologías relacionadas con el campo de estudio aquí propuesto.
2. **Capítulo 3: Planificación y Análisis.** Donde se planifica y analiza el problema a resolver mediante ingeniería de requisitos.
3. **Capítulo 4: Implementación y Pruebas.** Donde se profundiza en los detalles técnicos y se muestra la consecución de los objetivos mediante un caso de uso funcional.
4. **Capítulo 5: Conclusiones.**

Capítulo 2

Trabajos Relacionados

En las siguientes secciones se presenta a modo de introducción conceptos relativos a Programación Generativa, Cloud Computing y buenas prácticas en el desarrollo software. Se pretende con esto dotar al lector de un contexto o conocimiento básico para abordar el proyecto. De manera adicional se comparan tecnologías, trabajos y publicaciones relacionadas para mostrar el estado actual del arte.

2.1. Desarrollo Software

2.1.1. Ciclo de Vida del Desarrollo Software

El ciclo de vida del desarrollo software (Software Development Life Cycle, SDLC) es un procedimiento sistemático para construir software. El objetivo de este procedimiento es el de garantizar la calidad y las exactitud del software a desarrollar, haciendo que cumpla con las expectativas del usuario.

Etapas SDLC

1. Planificación: Etapa inicial donde se define el software a implementar, determinando el alcance que se quiere lograr. Esta etapa permite discernir el presupuesto, el tiempo y los recursos necesarios para el proyecto.
2. Ingeniería de requisitos: Etapa en la que se pone de manifiesto cuales son los requisitos que el proyecto software ha de cumplir para ser aceptado por el cliente.
3. Diseño: Etapa que consiste en trasformas los requisitos de alto nivel solicitados por el cliente a una realidad técnica que los equipos

de expertos puedan llevar a cabo.

4. Desarrollo: Etapa en la que los desarrolladores materializan el diseño técnico en producto software.
5. Pruebas: Etapa en la que el equipo de calidad valida y verifica que el software desarrollado cumple los requisitos esperados por el cliente.
6. Productivización: Etapa en la que el equipo de operaciones dispone el software para que el cliente comience a utilizarlo.
7. Mantenimiento: Etapa en la que se corrigen fallos percibidos tras un uso real del software.
8. Disposición: Etapa en la que se contempla como será la retirada del software una vez que haya alcanzado su vida útil.

2.1.2. Desarrollo Software y Buenas Prácticas

En esta sección se profundiza en buenas prácticas relativas al desarrollo software.

Integración Continua

La integración continua consiste en un modelo informático orientado al desarrollo software que propone la integración de nuevas funcionalidades y la ejecución de pruebas de manera constante para así detectar posibles problemas o regresiones en el software que se está desarrollando así como mejorar su calidad de una manera automatizada. En el artículo **Continuous Integration** [3] Martin Fowler ejemplifica este modelo mediante la implementación de una funcionalidad mediante integración continua.

Entrega Continua

La entrega continua es un procedimiento informático que consiste en realizar entregas continuas de versiones del software que se está construyendo. Está íntimamente ligado al modelo de Integración Continua y permite a los desarrolladores liberar versiones del producto software para que sean validadas por los responsables de calidad o integradas en entornos productivos mientras continúan desarrollando nuevas funcionalidades. El objetivo de este procedimiento es el de disponer de un

artefacto listo para ser probado, ejecutado y distribuido. En este procedimiento, la automatización llega hasta el punto de generar dicho artefacto, su despliegue o distribución requiere de intervención humana. En el artículo **Continuous Delivery: Overcoming adoption challenges** [4], se proponen diversas estrategias para adoptar procedimientos de entrega continua en el mundo empresarial así como retos en materia de investigación propuestos a la comunidad científica.

Despliegue Continuo

El despliegue continuo consiste en un procedimiento automatizado similar a la entrega continua con la diferencia de que en este caso el artefacto implementado es distribuido o ejecutado sobre un entorno virtual sin interacción humana, en otras palabras, es desplegado. De esta forma los tiempos de entrega y operación de nuevas funcionalidades software se ven notablemente reducidos. En el artículo **Problems and Solutions of Continuous Deployment: A Systematic Review** [5] se realiza un repaso sobre numerosos artículos relativos al despliegue continuo poniendo de manifiesto problemas o retos encontrados al intentar adoptar este procedimiento en el mundo empresarial y que solución se adecua más a cada uno de los problemas presentados.

2.1.3. En este Proyecto

Este proyecto considera que las etapas que forman parte del ciclo de vida del desarrollo software están altamente segmentadas. Es cierto que a la hora de llevar a cabo proyectos informáticos a gran escala como los encontrados en el sector espacial o energético, los cuales involucran a diversas organizaciones, cada una de estas etapas es necesaria ya que el volumen y la dimensión de dichos proyectos así lo requieren pero en el ámbito del desarrollo software orientado a las aplicaciones modernas, estas etapas pueden ser sintetizadas y reducidas, mejorando notablemente los tiempos de desarrollo y abaratando costes. En las últimas 2 décadas, tras la publicación del **Agile Manifesto** [6] la industria del software ha dado un giro orientando sus procesos hacia el cambio y mejora continua en lugar de intentar evitar o contemplar errores a priori mediante un diseño extenso y una planificación estricta. En el artículo **A decade of agile methodologies: Towards explaining agile software development** [7] se recogen diversas publicaciones relacionadas con el desarrollo ágil así como las aportaciones realizadas al campo fruto de la investigación software.

En línea con las buenas prácticas a la hora de desarrollar software

presentadas en la sección anterior, así como las ventajas que aporta las metodologías ágiles, este proyecto sintetiza el proceso de desarrollo software mediante un enfoque generativo, materializando una especificación de usuario en código fuente funcional. Para implementar las buenas prácticas de Integración Continua y Entrega Continua este proyecto hace uso de **Github** [8] como sistema gestor de repositorios de código así como de las herramientas que ofrece. En el capítulo 4 titulado Implementación y Pruebas, sección 4.1.1 Repositorio de Código se profundiza en esta cuestión.

Por otra parte, para implementar la buena práctica de Despliegue Continuo se ha decidido utilizar como tecnología de virtualización **Docker** [9], lo que permite aplicar el mismo tratamiento a las diversas aplicaciones generadas por el sistema, independientemente de su naturaleza o dominio. En la sección 4.1.1 Infraestructura Virtual de las Aplicaciones se profundiza y justifica el uso de esta tecnología para cumplir con el objetivo de desplegar y disponibilizar las aplicaciones generadas a través de Internet. De manera adicional, en la sección 2.3.3 Tecnología en el Mercado se repasan las principales soluciones orientadas al problema aquí planteado que ofrece tanto la comunidad Open-Source como los proveedores de soluciones en la nube.

2.2. Programación Generativa

La programación generativa (Generative Programming) de manera directa trata de « Escribir programas que escriban programas. » para aportar beneficios al ciclo de vida del desarrollo software en aspectos tales como:

1. **Productividad:** Las técnicas de programación generativa pueden ahorrar numerosas horas de codificación.
2. **Calidad:** El código generado mantiene una calidad uniforme a lo largo de toda la línea base de código generativo.
3. **Consistencia:** El código generado mantiene uniformidad tanto en nombrado de variables como en la firma de funciones lo que facilita su posterior mejora mediante programación manual.
4. **Abstracción:** En algunos casos, los generadores de código utilizan como entrada un modelo abstracto del sistema que van a producir. Este modelo abstracto representa las reglas de negocio de alto nivel que el sistema producido ha de cumplir y permite que el gene-

rador sea portable ya que la misma entrada puede ser aprovechada para producir código en distintos lenguajes o tecnologías.

Tal y como se presenta en el artículo **Generative Programming** [10] la programación generativa consiste en diseñar e implementar módulos software que pueden ser combinados para generar sistemas especializados y optimizados en cumplir requisitos específicos de dominio. Los objetivos de la programación generativa son:

1. Disminuir la desviación conceptual entre el código software y los conceptos de dominio.
2. Reutilización y adaptabilidad de módulos software.
3. Simplificar la administración de muchas variantes de un componente.
4. Aumentar la eficiencia en capacidad y en tiempo de ejecución.

2.2.1. Principios

Para alcanzar sus objetivos, la programación generativa ha de cumplir los siguientes principios tratados en el artículo **Generative Programming and Active Libraries** [11]

1. **Separation of concerns (SoC)**. La separación de preocupaciones pone de manifiesto la importancia de lidiar con los problemas de una manera atómica. Es decir, crear pequeños fragmentos de código que cumplan con una funcionalidad muy específica e integrarlos para dar lugar al componente deseado.
2. **Parameterization of differences**. La parametrización de componentes da la posibilidad de representar familias de componentes.
3. **Analysis and modeling of dependencies and interactions**. Define el término conocimiento de configuración horizontal como la dependencia que existe en el valor que toma el parámetro de un modelo, condicionado el valor de otros parámetros del mismo modelo.
4. **Separating problem space from solution space**. Define el término conocimiento de configuración vertical, donde se pone de manifiesto que el espacio del problema es específico del dominio mientras que el espacio de la solución ha de ser generalizado.

5. **Eliminating overhead and performing domain-specific optimizations.** Donde se aconseja un proceso de refinamiento tras la generación estática de componentes para eliminar software no utilizado en el dominio específico.

2.2.2. En este proyecto

Resulta imperativo presentar como ejemplo la producción en cadena o la producción en escala que Henry Ford aplicó en 1908 en el sector del automóvil. Este proceso de producción industrial revolucionó el mercado ya que mejoraba los tiempos de entrega, abarataba precio de producción y garantizaba un nivel de calidad similar a lo largo de toda la línea de productos fabricados. La idea de la cadena de producción se basa en que dado un diseño, por ejemplo un coche, en cada etapa de la cadena de producción se agrega un determinado componente y que la suma de N etapas con sus correspondientes componentes dan lugar a un producto completo y funcional, en el caso de este ejemplo un Ford T.

En línea con este concepto de producción en cadena, el cual es aplicado hoy en día en numerosos sectores, el objetivo de este proyecto es construir una cadena de producción software que sea capaz de generar y desplegar aplicaciones, para ello este proyecto se apoya en la programación generativa para sintetizar el ciclo de vida del desarrollo software, abstrayendo al usuario de los detalles internos de programación y permitiendo generar aplicaciones mediante una especificación de alto nivel. En este sistema el objetivo de generar aplicaciones no se limita al código fuente, si no que involucra otros aspectos como creación y configuración de recursos virtuales como por ejemplo el repositorio de código donde contener el código fuente, la configuración necesaria para dotar a las aplicaciones generadas de un ciclo de integración continua para evolucionar el código generado si fuera necesario, configuración necesaria para compilar, distribuir y disponer de artefactos ejecutables, así como la configuración y recursos virtuales necesarios para el despliegue continuo de las aplicaciones.

En el libro **Code Generation in Action (In Action Series)** [12] se presentan diversos ejemplos prácticos que han sido utilizados como referente en la implementación de este sistema para cubrir aspectos tales como:

1. Generación de Interfaces de usuario.
2. Generación de documentación.
3. Generación de servicios web basados en capas.

4. Generación de lógica de negocio.

El sistema aquí propuesto entra en la categoría generativa de **model driven design**, donde la especificación de alto nivel proporcionada por el usuario define el modelo de aplicación o servicio web que se va a generar. Para el proceso generativo se ha construido un motor de plantillas capaz de producir aplicaciones de diversa naturaleza o dominio a partir del modelo de aplicación definido como entrada por el usuario. En el libro **Code generation with templates** [13] se presenta por una parte un enfoque matemático para justificar el uso y mejora de plantillas y por otro un enfoque práctico y aplicado de como trabajar con plantillas o como definir el meta-lenguaje necesario para las plantillas. Como trabajo previo relacionado se hace referencia al proyecto **Apigena** [14] donde se aborda la producción de APIs-REST mediante un enfoque generativo.

2.3. Cloud Computing

La real academia de la ingeniería, define la computación en la nube o Cloud Computing[15] como: “La Utilización de las instalaciones propias de un servidor web albergadas por un proveedor de Internet para almacenar, desplegar y ejecutar aplicaciones a petición de los usuarios demandantes de las mismas.” La computación en la nube es un paradigma en el que la información se almacena de manera permanente en servidores de internet y se ofrece, a través de aplicaciones, como servicio. El Cloud Computing es un nuevo modelo de prestación de servicios [16] de negocio y tecnología, pagando únicamente por el consumo efectuado. Este modelo de negocio ha conseguido un lugar importante en el mercado y en la computación por lo que se considera interesante hacer una introducción con el objetivo de que el lector tenga un mayor grado de comprensión, ya que la computación en la nube juega un papel fundamental en la resolución del problema planteado.

La nube o el cloud computing puede definirse como el conjunto de tecnologías que permiten acceder a servicios información a través de Internet, es decir, sin necesidad de instalar programas software. No hay que entender a la nube como una entidad física, sino como una red de servidores distribuidos en todo el mundo, que inter-conectados entre si funcionan como un único ecosistema. El paradigma que presenta la nube se basa en que en lugar de acceder a archivos y datos desde un equipo personal o local, se accede a ellos en línea desde cualquier dispositivo conectado a Internet.

2.3.1. Tipos de Nube

La nube puede clasificarse en distintos modelos en función del tipo de solución que quiera cubrir:

Nube Pública

Las nubes públicas son propiedad de los proveedores de servicio, es decir, el hardware, el software y todos los componentes necesarios para la infraestructura virtual subyacente son propiedad del proveedor de servicio cuya tarea es administrarlos. El consumidor accede y administra los recursos que va a consumir desde un explorador web. Algunos ejemplos de proveedores de nube pública son GoogleCloud [17] , Amazon Web Services [18] o Microsoft Azure [19]

Nube Privada

Las nubes privadas se caracterizan por que sus servicios e infraestructura subyacente se mantiene en una red privada, utilizados exclusivamente por una empresa u organización. Por ejemplo, una nube privada puede encontrarse físicamente en el centro de datos de una compañía o de un banco.

Nube Híbrida

Las nubes híbridas se caracterizan por combinar los modelos de nube pública y nube privada. La nubes híbridas inter-conectan ambos modelos con el objetivo de compartir datos y aplicaciones aportando mayor flexibilidad y opciones de desarrollo, optimizando el uso de la infraestructura subyacente y la seguridad.

2.3.2. Tipos de Servicios en la Nube

La computación en la nube permite la integración de servicios en red a nivel mundial y además contribuye al uso eficiente de la energía aunque la centralización de las aplicaciones y el almacenamiento de los datos origina una interdependencia con los proveedores de servicios. La disponibilidad de estos servicios necesita de conexión a Internet. En el artículo **Comparison Among Cloud Technologies and Cloud Performance** [20] se revisan las distintas tecnologías que configuran la nube comparando distintas herramientas para trabajar con este tipo de servicios.

Software como servicio

El Software como Servicio [21] (Software as a Service, SaaS) ofrece el consumo de una gran variedad de aplicaciones o servicios proporcionadas por los proveedores del servicio y que se ejecutan en la infraestructura de la nube. Las aplicaciones en la nube son accesibles por distintos dispositivos del cliente a través de una interfaz sencilla, como por ejemplo un navegador web. El cliente no gestiona la infraestructura del servicio, que incluye la red de comunicaciones, los servidores, los sistemas operativos y el almacenamiento, simplemente utiliza el servicio como es el caso de este proyecto.

Plataforma como servicio

La Plataforma como Servicio [22] (Platform as a Service, PaaS) es una categoría de servicios cloud que proporciona un entorno a los desarrolladores para crear aplicaciones y servicios que funcionen a través de Internet. Los servicios PaaS se alojan en la nube, y los usuarios pueden acceder a ellos a través de un navegador web. Este tipo de servicios consisten en funcionalidades pre-configuradas a las que los clientes puedan suscribirse, eligiendo las características que deseen integrar en función de sus necesidades. Algunas de las funcionalidades PaaS son:

1. Sistema operativo.
2. Entorno de scripting de servidor.
3. Sistema de gestión de base de datos.
4. Software de servidor.
5. Soporte técnico.
6. Almacenamiento.
7. Acceso a la red.
8. Herramientas de diseño y desarrollo.
9. Hosting.

Infraestructura como servicio

La Infraestructura como Servicio [23] (Infrastructure as a Service, IaaS) es un servicio cloud, que proporciona acceso a recursos informáticos situados en un entorno virtualizado, a través de una conexión pública. Los recursos informáticos ofrecidos son básicamente infraestructura

de procesamiento. Físicamente, el repertorio de recursos hardware disponibles procede de multitud de servidores y redes, generalmente distribuidos entre numerosos centros de datos, de cuyo mantenimiento se encarga el proveedor del servicio cloud. El cliente, por su parte, obtiene acceso a los componentes virtualizados para construir con ellos su propia plataforma informática.

Estas son algunas de las ventajas o características de una implementación basada en el modelo de Infraestructura como Servicio:

1. Escalabilidad.
2. Ausencia de la necesidad de invertir en hardware.
3. Independencia de la localización.
4. Seguridad física en los centros de datos.

2.3.3. Tecnologías en el Mercado

Actualmente existen soluciones cloud que ofrecen parte de la funcionalidad que este proyecto intenta cubrir, las más destacadas son:

Github

GitHub [8] es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git. El código es almacenado de forma pública o privada. GitHub aloja el código en repositorios y brinda herramientas muy útiles para trabajar de forma colaborativa. Github [24] ofrece varias herramientas:

1. Una wiki para el mantenimiento de las distintas versiones de las aplicaciones.
2. Un sistema de seguimiento de problemas que permiten a los miembros de un equipo de desarrollo detallar problemas software.
3. Una herramienta de revisión de código, donde se pueden añadir anotaciones en cualquier punto de un fichero y debatir sobre determinados cambios realizados en un commit específico.
4. Un visor de ramas donde se pueden comparar los progresos realizados en las distintas ramas de un repositorio.

Heroku

Heroku [25] es una plataforma en la nube basada en un sistema de contenedores gestionados, con servicios de datos integrados y un potente ecosistema para implementar y ejecutar aplicaciones. Heroku presenta una fácil integración con diversos servicios web como por ejemplo GitHub. Heroku se caracteriza por el fichero de configuración denominado Procfile depositado junto al repositorio de código definiendo comportamientos relativos a como ese código fuente ha de ser ejecutado o desplegado. Heroku proporciona una herramienta de línea de comandos que facilita la automatización de tareas y despliegues.

Kubernetes

Kubernetes [26] es una plataforma de código abierto que automatiza y orquesta las operaciones con contenedores Docker. Kubernetes permite agrupar conjuntos de contenedores para ejecutar aplicaciones o montar ecosistemas cloud de una manera rápida y sencilla a lo largo de las distintas tipos de cloud: públicas, privadas o híbridas. Su objetivo es el de mejorar el despliegue y operación de aplicaciones software que utilicen contenedores como motor de virtualización ligera.

GCP: Cloud Code

Google Cloud Platform [17] presenta Cloud Code [27] como un conjunto de herramientas y plugins para complementar entornos de desarrollo integrado (IDEs) tales como:

1. IntelliJ
2. Visual Studio

El objetivo de Cloud Code es el de mejorar el ciclo de vida de desarrollo software de aplicaciones cloud nativas orientadas a Kubernetes, la implementación del orquestador de contenedores en su propia plataforma cloud.

Travis CI

Travis-CI [28] es una plataforma de código abierto orientada a la integración, entrega y despliegue continuo, mejorando el ciclo de vida del desarrollo software al integrarse con servicios de terceros. Travis-CI permite definir las etapas del pipeline al que el software en cuestión ha de someterse mejorando notablemente el proceso de integración y testing.

Docker Hub

Docker Hub [29] es un repositorio de artefactos alojado en la nube. Pone a disposición de los usuarios un entorno virtual donde crear, probar, almacenar y distribuir imágenes Docker. El objetivo de Docker Hub es el de mejorar el proceso de entrega y despliegue continuo al disponibilizar su repositorio a través de Internet facilitando la automatización de estas buenas prácticas.

AWS:CodeDeploy

Amazon Web Services [18] presenta CodeDeploy [30] como servicio gestionado para automatizar tareas de despliegue software sobre diversos recursos virtuales de su propia nube como máquinas virtuales (EC2), contenedores docker (Fargate), funciones cloud (Lambda) o servidores locales. Este tipo de servicios optimizan el ciclo de vida del desarrollo software y permiten aplicar buenas prácticas como el despliegue continuo de software.

AWS: CodePipeline

Amazon Web Services [18] presenta CodePipeline [31] como servicio administrado para la implementación de entrega continua en proyectos software. Este servicio automatiza las actualizaciones de infraestructura y de aplicación. Su objetivo es el de mejorar el ciclo de vida del desarrollo software reduciendo las tareas manuales relativas al desarrollo y operaciones. Aunque inicialmente fue pensado para servicios de la propia nube de AWS sus últimas actualizaciones permiten su integración con servicios de terceros.

Comparativa

En la figura 2.1 se comparan las tecnologías del mercado prestando especial atención al tipo de funcionalidad que cubren en términos de los objetivos de este proyecto. La idea es mostrar como GCM:Coode-Runner integra las funcionalidades propuestas para servir una solución completa.

	FUNCIONALIDADES							
	Generación de Código	Gestión de Repositorios de Código	Integración Continua	Entrega Continua	Despliegue Continuo	Análisis de Logs	Operaciones	
T E C N O L O G I A S	<i>Github</i>	X	X	X	X			
	<i>Heroku</i>					X	X	X
	<i>Kubernetes</i>					X	X	X
	<i>GCP: Cloud Code</i>	X				X		
	<i>Travis CI</i>			X	X			
	<i>Docker Hub</i>		X	X	X		X	
	<i>AWS: CodeDeploy</i>					X	X	X
	<i>AWS: CodePipeline</i>			X	X			
	<i>GCM: Code-Runner</i>	X	X	X	X	X	X	X

Figura 2.1: Tabla Comparativa: Tecnología VS. Funcionalidad

2.3.4. En este Proyecto

Como se ha comentado en la introducción, las aplicaciones generadas por este sistema se caracterizan por ser aplicaciones del dominio de la web 2.0 es decir, aplicaciones accesibles a través de Internet por lo que la computación en la nube desarrolla un papel fundamental.

En este proyecto se ha de diferenciar por una parte la infraestructura virtual y los recursos de la nube utilizados para poner a disposición de los usuarios a través de Internet las aplicaciones generadas y por otra parte los recursos virtuales y de computación propios del sistema software necesarios para su correcto funcionamiento.

Como se puede observar en las herramientas y tecnologías que están actualmente presentes en el mercado, es notable la necesidad de integrar y automatizar tareas relacionadas con el ciclo de vida del desarrollo software y las buenas prácticas. Aunque las herramientas presentadas tienen un gran alcance, ninguna presenta una solución completa como la que se presenta en este proyecto. Este sistema software pone a disposición de los usuarios un marco de trabajo generativo para crear y desplegar aplicaciones en unos sencillos pasos con el único requisito de poseer un perfil en Github. Para ello se integran diversas funcionalidades de servicios cloud como Github, Heroku y máquinas virtuales proporcionadas por la nube pública de Google junto con Docker para construir un pipeline que permita generar y desplegar aplicaciones haciendo uso de las buenas prácticas anteriormente mencionadas. En la sección 4.1 Implementación y Desarrollo se justifica el uso de estas tecnologías.

Está actividad de negocio entra dentro del tipo de servicio en la nube clasificado como Software as a Service2.3.2 puesto que el usuario hace uso del sistema para consumir las aplicaciones que decide generar pe-

ro es abstraído del entorno virtual donde estas aplicaciones corren, es decir, el usuario trabaja a nivel de aplicación o servicio software. En la sección 4.1.1 Infraestructura Virtual de las Aplicaciones se profundiza en estos conceptos y en la tecnología utilizada para su implementación.

Por otra parte, para garantizar el correcto funcionamiento de este sistema software se requieren de los otros dos tipos de servicios en la nube. En la sección 4.1.3 Infraestructura Virtual se profundiza en los recursos de computación en la nube utilizados presentando los distintos entornos virtuales disponibles para que los usuarios puedan utilizar el sistema generativo.

Capítulo 3

Planificación y Análisis

Este capítulo se centra en analizar el problema y definir la solución del sistema a construir en términos de ingeniería software. Para ello, se incluye la planificación temporal de las tareas a realizar junto con el análisis software dirigido por requisitos y diagramas de casos de uso.

3.1. Planificación

La planificación temporal muestra mediante el uso de diagramas de Gant la distribución en el tiempo de las tareas necesarias para cumplir con los objetivos de este proyecto. Los diagramas de Gant han sido confeccionados con la herramienta online Tom's Planner [32]

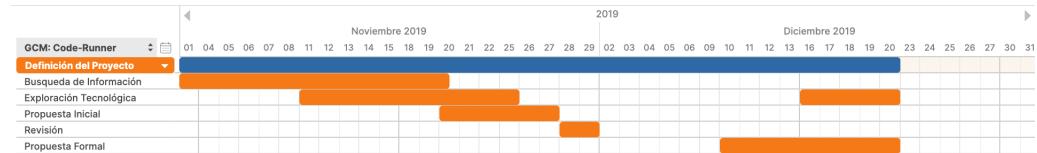


Figura 3.1: Planificación: Definición del Proyecto

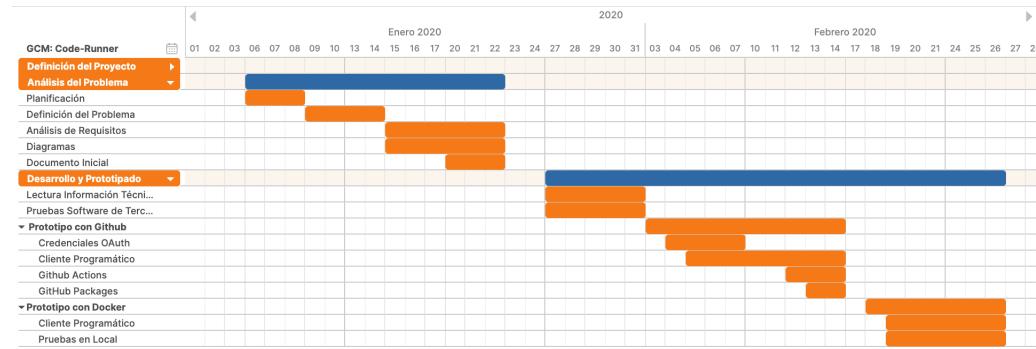


Figura 3.2: Planificación: Análisis y Desarrollo Inicial

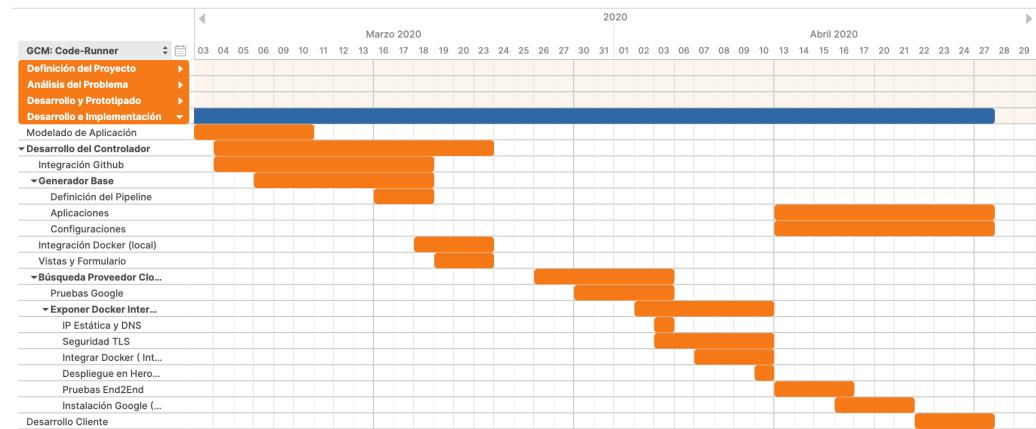


Figura 3.3: Planificación: Desarrollo Formal

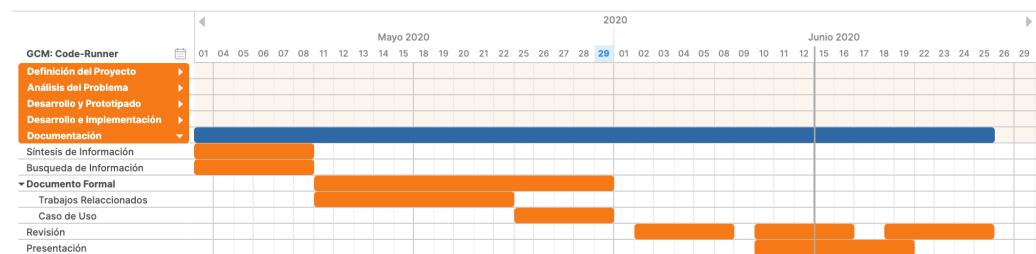


Figura 3.4: Planificación: Documentación Formal

3.2. Análisis Software

El análisis software pone de manifiesto la especificación del sistema software a construir. Dirigida por Ingeniería de requisitos y de casos de

uso, las siguientes secciones muestran las partes y funcionalidades que el sistema resultante ha de cumplir.

3.2.1. Requisitos de Información

Los requisitos de información se caracterizan por reunir la información relevante para la solución, es decir, que debe gestionar y almacenar el sistema software.

RI-1. Workspace: Representación del espacio de trabajo del usuario. Contiene el conjunto de las aplicaciones que el usuario ha creado usando el sistema. Concepto similar al del Escritorio o "Desktop" de un sistema operativo. Contenido: nombre de usuario o propietario del workspace, referencia al conjunto de aplicaciones generadas.

RI-2. App: Representación de cada una de las aplicaciones generadas por el sistema. Contenido en Anexo: Modelo de Datos6.1

RI-3. Información de Usuario: Información sensible del usuario o desarrollador que va a usar el sistema. Credenciales al sistema de gestión de repositorios para el código fuente, en este caso relativa al perfil de github. Contenido: nombre de usuario, correo electrónico, organización.

3.2.2. Requisitos Funcionales

Como se define en la ingeniería de requisitos, los requisitos funcionales establecen el comportamientos del sistema.

RF-1. Gestión de aplicaciones.

El sistema ha de gestionar el ciclo de vida de las aplicaciones generadas.

RF-1.1. El sistema permitirá crear aplicaciones.

- **RF-1.1.1.** El sistema generará aplicaciones de diversa naturaleza, tecnología o dominio. Añadir referencia a especificación de aplicaciones.
- **RF-1.1.2.** El sistema creará y guardará en repositorios de código las aplicaciones generadas.

- **RF-1.1.3.** El sistema construirá los artefactos necesarios para ejecutar las aplicaciones generadas.
- **RF-1.1.4.** El sistema ejecutará los artefactos construidos para disponibilizar vía Internet las aplicaciones creadas.

RF-1.2. El sistema permitirá eliminar aplicaciones.

- **RF-1.2.1.** El sistema detendrá los artefactos en ejecución de aplicaciones eliminadas.
- **RF-1.2.2.** El sistema eliminará los artefactos construidos de aplicaciones eliminadas.
- **RF-1.2.3.** El sistema eliminará los repositorios de código de aplicaciones eliminadas.

RF-1.3. El sistema permitirá ejecutar aplicaciones.

- **RF-1.3.1.** El sistema será capaz de hacer a las aplicaciones accesibles vía Internet.
- **RF-1.3.2.** El sistema proporcionará las herramientas y configuraciones necesarias para que las aplicaciones puedan ser ejecutadas en local y ser accesibles vía **localhost**.

RF-1.4. El sistema permitirá detener aplicaciones.

- **RF-1.4.1.** El sistema será capaz de detener las aplicaciones que sean accesibles vía Internet.
- **RF-1.4.2.** El sistema deberá liberar los recursos virtuales correspondientes a la aplicación detenida.
- **RF-1.4.3.** El sistema proporcionará las herramientas y configuraciones necesarias para que las aplicaciones ejecutadas en **localhost** sean detenidas de manera controlada.

RF-1.5. El sistema permitirá visualizar información de las aplicaciones.

- **RF-1.5.1.** El sistema permitirá consultar información de utilidad relativa a las aplicaciones, como por ejemplo, especificación, URL del repositorio donde se alberga el código fuente, URL o IP donde la aplicación puede ser consumida vía Internet.
- **RF-1.5.2.** El sistema permitirá inspeccionar aplicaciones. El sistema permitirá a los usuario consultar información de DEBUG de las aplicaciones accesibles vía Internet, es decir, permitirá consultar LOGs de aplicaciones en ejecución.
- **RF-1.5.3.** El sistema permitirá visualizar información del estado del ciclo de vida de las aplicaciones. El ciclo de vida de las aplicaciones del sistema se representará con el siguiente flujo de estados:

RF-2. Gestión de credenciales.

El sistema proporcionará los mecanismos necesarios para que los usuarios se autentiquen en el sistema.

RF-2.1. El sistema permitirá iniciar sesión a los usuarios mediante nombre de usuario y contraseña. Típicamente estas credenciales corresponderán al sistema de gestión de repositorios de código.

RF-2.3. El sistema permitirá cerrar sesión a los usuarios de manera controlada, es decir, mantendrá las aplicaciones en ejecución.

RF-2.3. El sistema realizará todas las operaciones que requieran autenticación mediante TOKEN.

RF-2.4. El sistema proporcionará a los usuarios el TOKEN necesario para trabajar localmente con las aplicaciones.

3.2.3. Requisitos No Funcionales

Los requisitos no funcionales, se refieren a todos los requisitos que no describen información a guardar, ni funciones a realizar por el sistema, sino características de funcionamiento.

RNF-1 Se necesitará acceso a Internet para utilizar las funcionalidades del sistema.

RNF-2 Se necesitará una cuenta en Github para utilizar las funcionalidades del sistema.

RNF-3 Se prestará especial atención en la gestión de credenciales de los usuarios.

RNF-4 Se prestará especial atención en la seguridad del las aplicaciones expuestas a Internet.

3.2.4. Casos de Uso

Los diagramas de casos de uso, son diagramas UML que representan gráficamente a todos los elementos que forman parte del modelo de casos de uso junto con la frontera del sistema. Delimitan el sistema a diseñar. Determinan el contexto del uso del sistema. Describen el punto de vista de los usuarios en el sistema.

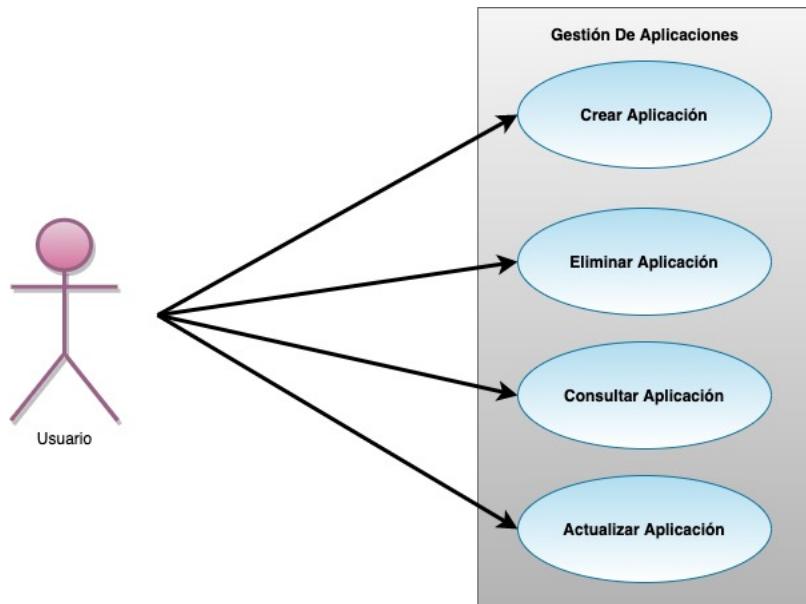


Figura 3.5: Casos de uso - Gestión Aplicaciones I [1]

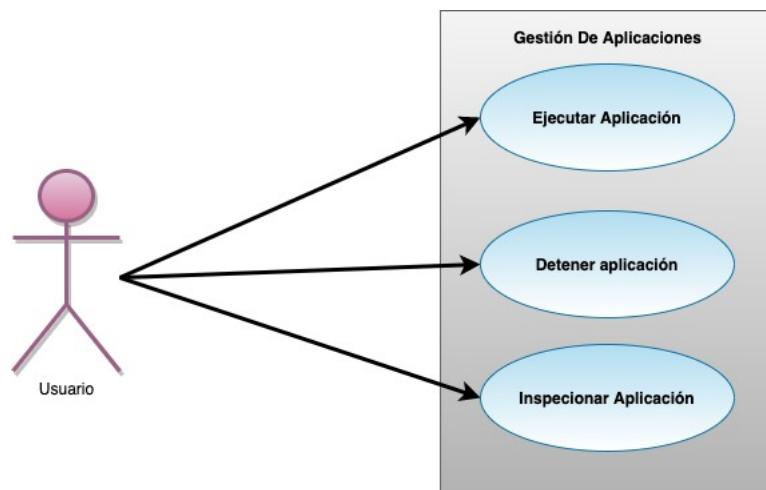


Figura 3.6: Casos de uso - Gestión Aplicaciones II [1]

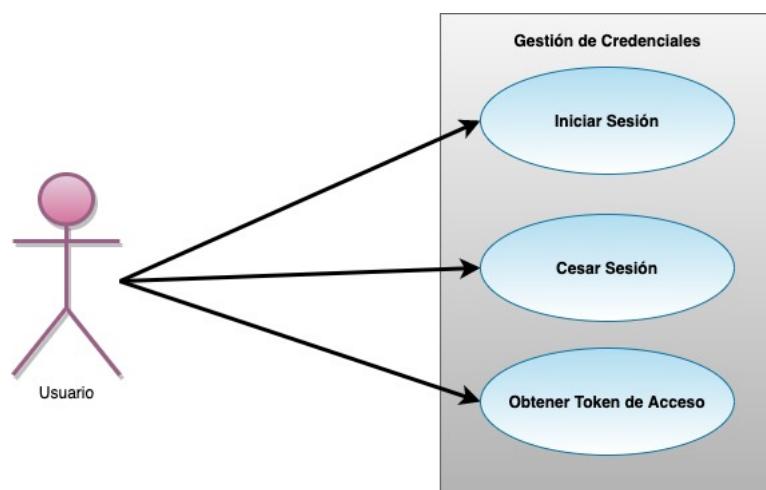


Figura 3.7: Casos de uso - Gestión Credenciales [1]

3.2.5. Diagramas de Secuencia

Los diagramas de secuencia especifican las interacciones temporales entre las piezas internas del sistema para cumplir con las funcionalidades requeridas. Normalmente la tecnología que implementa cada una de las piezas internas del sistema no es incluida en este tipo de diagramas pero en este caso se incluye para mejorar la compresión del sistema por parte del lector.

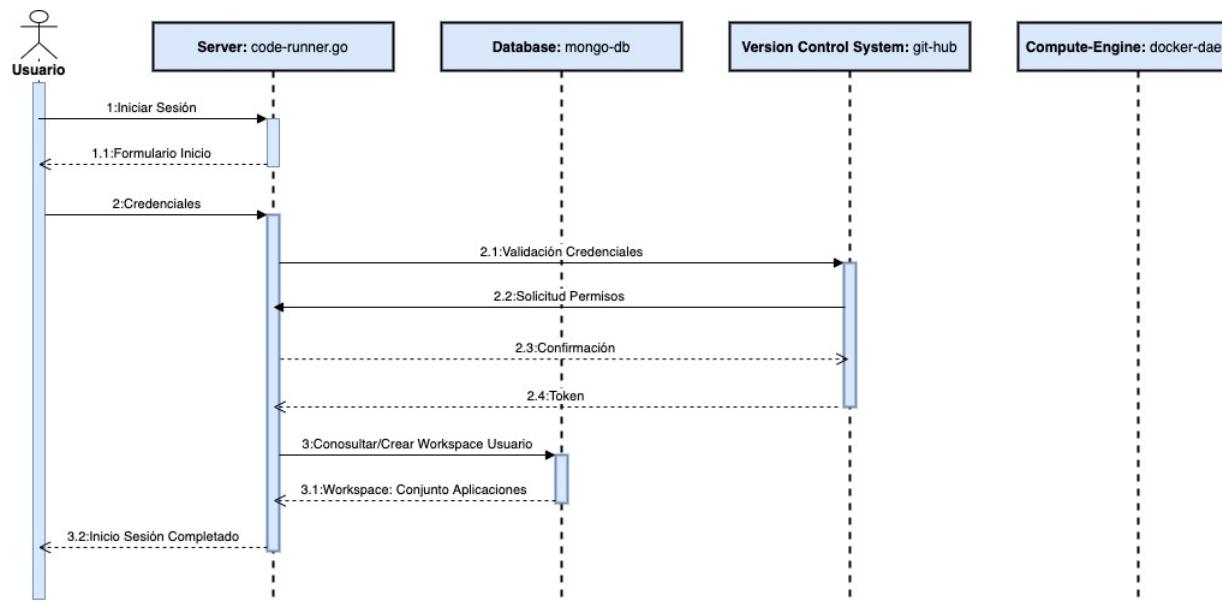


Figura 3.8: Diagramas de Secuencia - Inicio sesión [1]

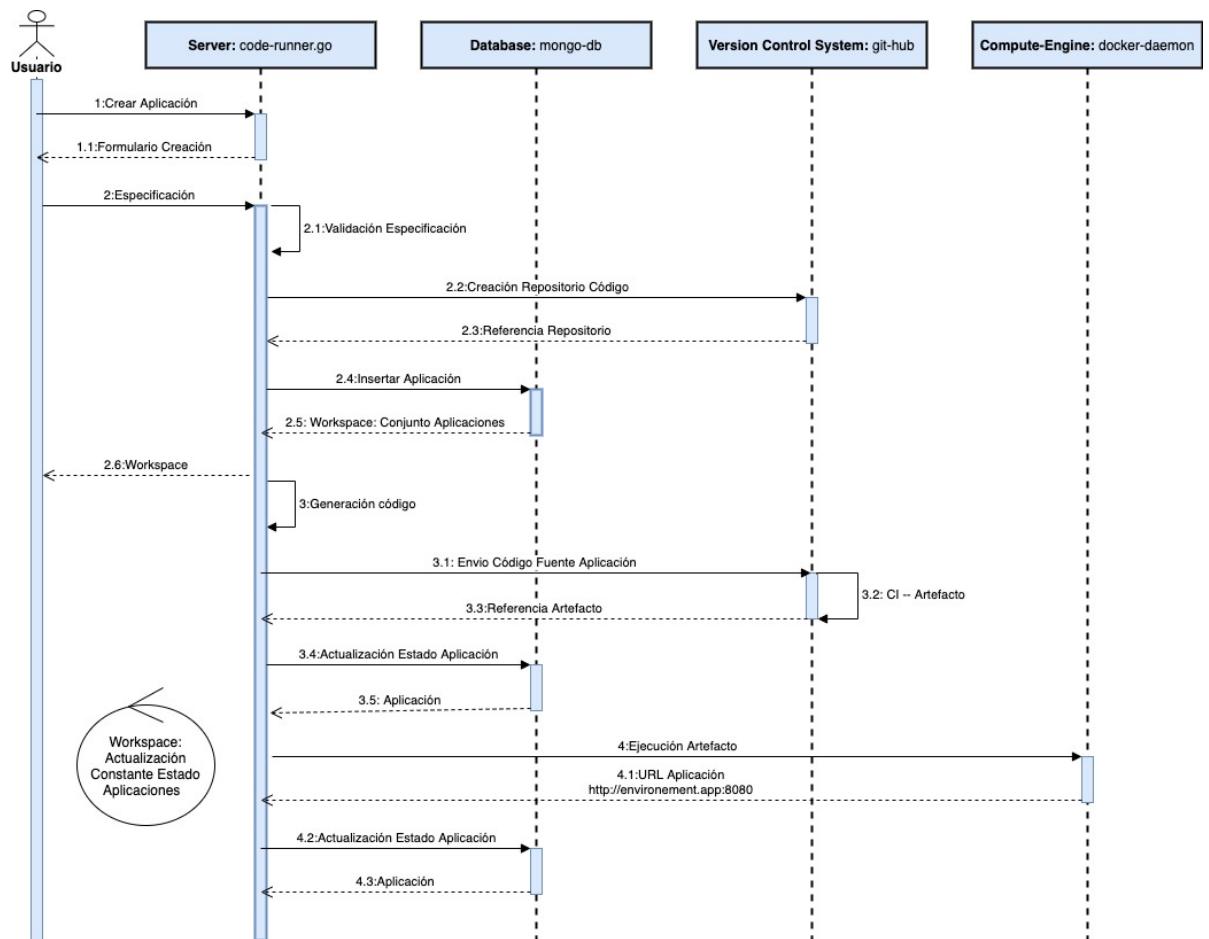


Figura 3.9: Diagramas de Secuencia - Crear Aplicación [1]

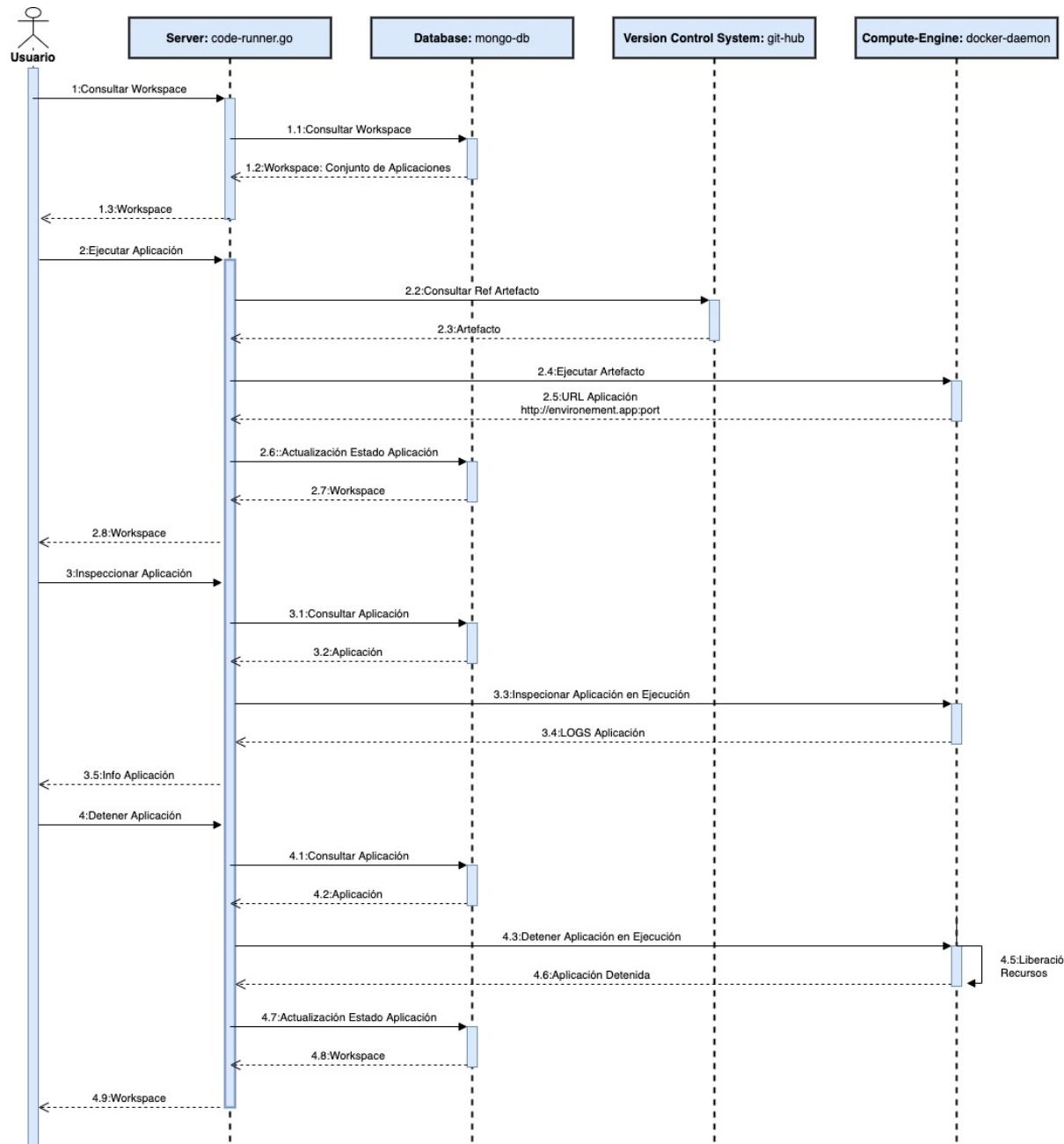


Figura 3.10: Diagramas de Secuencia - Ejecutar, Inspeccionar y Detener Aplicación [1]

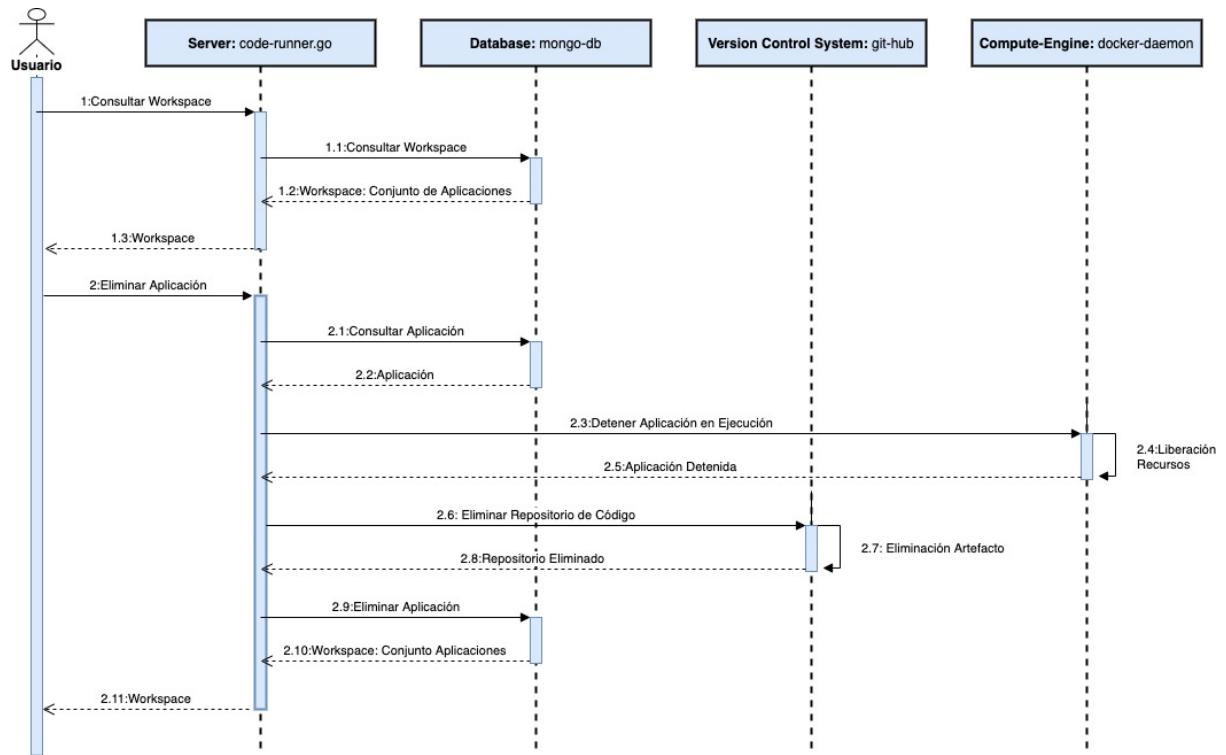


Figura 3.11: Diagramas de Secuencia - Eliminar Aplicación [1]

Capítulo 4

Implementación y Pruebas

4.1. Implementación y Desarrollo

En el proceso de desarrollo e implementación del sistema, se ha seguido el patrón de arquitectura software modelo-vista-controlador (MVC) [?] cuya finalidad es separar los datos y la lógica de una aplicación de la interfaz de usuario. Este patrón de arquitectura se basa en la re-utilización de código y separación de conceptos. El objetivo de esta metodología es el de facilitar el desarrollo, mantenimiento y escalabilidad del software resultante.

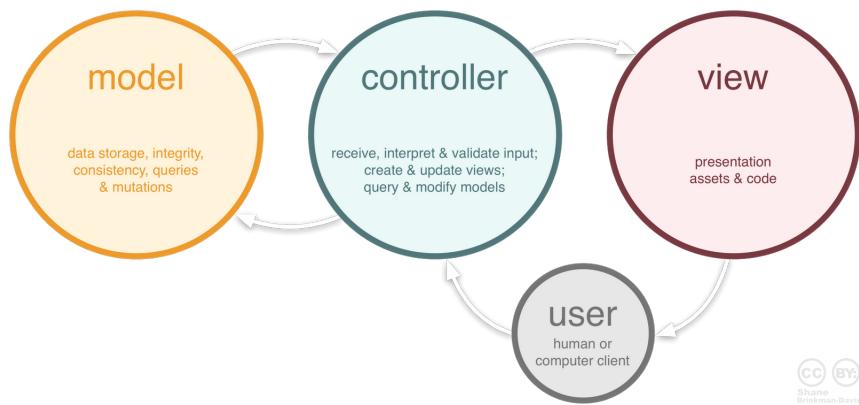


Figura 4.1: Modelo-Vista-Controlador [?]

Los componentes [33] del Modelo-Vista-Controlador se pueden definir en:

1. **Modelo:** Representa la información, gestionando los accesos a ella, la base de datos o capa de persistencia es un componente del modelo.

2. **Vista:** Representa el modelo en un formato adecuado para interactuar con él mediante la interfaz de usuario.
3. **Controlador:** Responde a eventos invocando peticiones al modelo cuando se hace alguna solicitud de la información (por ejemplo, editar un documento o un registro en base de datos). El controlador interactúa con su vista asociada para realizar cambios en la forma de representar el modelo.

Como parte del desarrollo del proyecto también se ha utilizado la metodología de ingeniería del software conocida como desarrollo basado en tests (Test-Driven Development, TDD). Dicha metodología consiste en escribir los test necesarios para cubrir las funcionalidades del sistema y a partir de ellos, escribir el código necesario para que dichas pruebas sean superadas, cumpliendo así la funcionalidad que se está poniendo a prueba. En las siguientes secciones se detallan las partes que forman el sistema así como las tecnologías utilizadas.

4.1.1. Back-End

El Back-End o servidor se refiere a la parte trasera del sistema, donde ocurre toda la lógica y funcionalidad pero que es transparente para el usuario final que consume el sistema. Para facilitar la comprensión de las partes que forman el Back-End del proyecto se van a desglosar en las siguientes secciones.

Servidor

El servidor es la entidad principal del Back-End, contiene la lógica del sistema y es el responsable de la comunicación entre el resto de componentes del sistema y de servir o renderizar las vistas al cliente. El servidor ha sido desarrollado utilizando Golang [34] como lenguaje de programación. Inicialmente se pensó en utilizar JavaScript bajo el framework Node.JS, ya que dicho marco de trabajo funciona de manera asíncrona, lo que facilita muchas de las operaciones en segundo plano que este sistema ha de realizar, además Node.JS facilita la integración del servidor con diversas tecnologías cliente basadas también en JavaScript. Una de las principales deficiencias observadas en el uso de JavaScript para el desarrollo del servidor de este proyecto es el manejo de plantillas para el generador de código por lo que este lenguaje fue descartado.

En la etapa de generación de código la templatización de los modelos de datos necesarios para producir aplicaciones es vital, no solo en

su implementación si no también en la eficiencia de estas operaciones y es aquí donde Golang destaca, ya que permite serializar estructuras de datos complejas y utilizar dicha representación para crear contenido. Otra ventaja que proporciona utilizar Golang en este proyecto es el uso de Go-Rutinas, es decir, la capacidad de ejecutar funcionalidad en segundo plano de una manera sencilla y controlada junto con la generalización que se alcanza al usar interfaces. Estos motivos hacen que Golang sea el lenguaje de programación adecuado para el desarrollo de este proyecto. Es interesante destacar que Golang ha sido diseñado para optimizar el uso concurrente de las aplicaciones creadas con él, lo que mejora considerablemente los tiempos de respuesta y la experiencia de usuario de cara a que el proyecto sea utilizado por numerosos clientes a través de Internet.

Persistencia de Datos

La parte imprescindible para la capa de acceso a datos es el sistema gestor de bases de datos, encargado de almacenar los modelos del sistema. Como propuesta inicial para el desarrollo del sistema, se pensó en utilizar un esquema relacional utilizando sistemas gestores de bases de datos relacionales como MySQL o PostgreSQL por la facilidad de diseño e implementación basada en entidad-relación.

La principal desventaja de modelar el problema mediante entidad-relación es la perdida en la flexibilidad de los modelos de datos ya que este esquema tradicional obliga a que todas las filas de una tabla representativa de un modelo tengan los mismos campos para mantener la entidad referencial y debido a la diversidad de aplicaciones que este sistema es capaz de generar se requiere que el modelo de datos sea flexible. Debido a esto, se ha decidido utilizar MongoDB [35] como sistema gestor de bases de datos no-relacionales también denominadas bases de datos documentales, con el objetivo de conseguir un esquema interno flexible y eficiente. Es decir, el modelo de aplicación que el usuario define para generar aplicaciones podría ser evolucionado y mejorado sin romper el esquema actual.

MongoDB es un sistema de base de datos NoSQL orientado a grandes cantidades de datos, guarda estructuras de datos en documentos similares a JSON. Se trata de una base de datos ágil que permite cambiar los esquemas de una aplicación cuando evoluciona. Los principales objetivos de estas bases de datos son la escalabilidad, rendimiento y gran disponibilidad.

En MongoDB los documentos se agrupan en colecciones. Las colecciones no imponen una estructura fija a los documentos que contienen,

ni siquiera al tipo de datos de cada campo. Esto es una ventaja cuando las entradas de la base de datos dependen de la naturaleza de las aplicaciones que el usuario final quiera generar.

Repositorio de Código

Las entidades generadas por este sistema (aplicaciones) están compuestas por un conjunto de ficheros (README, licencia, código fuente, configuración, scripts, makefile, dockerfile...etc), estos ficheros son almacenados en repositorios de código alojados en la nube de manera pública, para ello, este proyecto esta completamente integrado con Github [8] aprovechando de este gestor de repositorios las siguientes funcionalidades:

1. **Autenticación.** El usuario final que utilice este sistema para generar aplicaciones necesita una cuenta de Github, ya que la autenticación mediante usuario y contraseña corresponde con las credenciales de su perfil de Github. Esto permite al sistema crear repositorios y subir el código generado a dichos repositorios bajo el perfil del usuario registrado. El proceso de autenticación contra Github proporciona una Token de acceso el cual es aprovechado por una parte por el sistema generativo para realizar llamadas a la API de Github y automatizar ciertas operaciones en nombre del usuario final y por otra parte los usuarios finales del sistema pueden utilizar este Token para trabajar localmente con las aplicaciones generadas por el sistema.
2. **Integración Continua.** Github proporciona las denominadas GitHub Actions [36] que permiten automatizar, personalizar y ejecutar flujos de integración continua sobre los proyectos software alojados en sus repositorios, para ello, en el proceso de Code Generation este sistema produce un fichero en la ruta:
repositorio/.github/workflows/ci.yml 6.1 donde se describe el proceso o etapas de compilación, testing y empaquetado de la aplicación contenida en el repositorio, independientemente de la naturaleza de la aplicación, dando lugar a un artefacto auto-contenido listo para ser desplegado sobre un entorno virtual. Esto garantiza el uso de las buenas prácticas de integración y entrega continua relativas al desarrollo software en todas las aplicaciones generadas.
3. **Gestión de Artefactos** Github proporciona los denominados GitHub Packages [37] que permiten publicar y consumir paquetes o arte-

factos resultado de la compilación y empaquetado de las aplicaciones software albergadas en un repositorio de código. GitHub-Packages es capaz de gestionar paquetes de diversa naturaleza, siendo las imágenes docker las utilizadas en este proyecto.

Infraestructura Virtual de las Aplicaciones

La infraestructura virtual de las aplicaciones representa al entorno de computación en la nube que cuenta con los recursos necesarios para ejecutar o desplegar las aplicaciones generadas por el proyecto, permitiendo a los usuarios utilizar las aplicaciones generadas vía Internet. Con el objetivo de generalizar las operaciones sobre las aplicaciones se ha decidido utilizar Docker como tecnología de virtualización ligera.

Docker [9] es una herramienta para automatizar el despliegue de aplicaciones dentro de contenedores, proporcionando una capa de abstracción y automatización de virtualización a nivel de sistema operativo. Docker utiliza aislamiento de recursos del kernel de sistemas Unix para permitir que contenedores independientes se ejecuten dentro de una sola máquina, evitando la sobrecarga de iniciar y mantener máquinas virtuales completas.

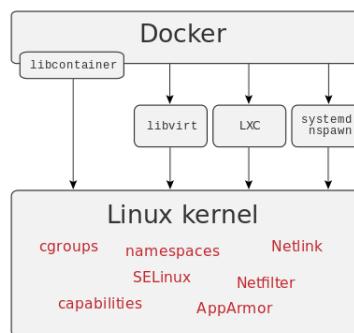


Figura 4.2: Virtualización Docker [2]

La principal diferencia con respecto a máquinas virtuales es el tamaño. Las máquinas virtuales incluyen la aplicación, el código, las librerías necesarias y un sistema operativo anfitrión completo mientras que los contenedores disponen de recursos similares pero de manera compartida con el núcleo del sistema operativo anfitrión y junto con otros contenedores los cuales se ejecutan como procesos aislados en el espacio de usuario del sistema operativo. No están vinculados con ninguna infraestructura específica.

Docker aporta a este proyecto la capacidad de abstraer las apli-

ciones de la infraestructura virtual subyacente. En la etapa de Code Generation se produce un fichero denominado Dockerfile con la definición necesaria para ejecutar la aplicación producida en un contenedor. Este fichero es aprovechado por el flujo de integración y entrega continua de la aplicación para producir una imagen docker como artefacto. Este artefacto es utilizado para crear un contenedor docker en un entorno virtual accesible a Internet (máquina virtual), habilitando el puerto necesario para garantizar que el tráfico es re-direcccionado al contenedor correspondiente y de esta manera permitir que el usuario del sistema generativo consuma a través de Internet la aplicación generada, siendo todo esto transparente para él.

4.1.2. Front-End

El Front-End, capa de presentación o capa de cliente es la encargada de controlar la interacción con el usuario. En este proyecto, la programación en el lado cliente se utiliza para mantener información actualizada del estado en el que se encuentran las aplicaciones producidas por el sistema. Para ello, se aprovecha el motor de plantillas proporcionado por el servidor Golang junto con JQuery y Ajax para realizar llamadas asíncronas al servidor y actualizar dinámicamente los datos que percibe el cliente. De manera adicional, el sistema proporciona una consola de monitorización que permite al usuario inspeccionar en tiempo real los logs de su aplicación así como el tráfico que dicha aplicación está soportando usando para ellos WebSockets.

Tecnología Cliente

JQuery [38] es una librería de JavaScript, que permite interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con el servidor mediante llamadas asíncronas realizadas con Ajax. Su objetivo principal es dar un comportamiento dinámico a la web. JQuery funciona en múltiples navegadores y es compatible con CSS3.

Ajax [39], abreviatura de JavaScript Asíncrono y XML es un conjunto de técnicas de desarrollo web en el cliente para crear aplicaciones asíncronas y dinámicas. Con Ajax, las aplicaciones pueden enviar y recibir datos desde el servidor en segundo plano sin que interfiera en la visualización o el comportamiento de la aplicación. Utilizar Ajax supone crear aplicaciones web en las que la capa de intercambio de datos ha sido separada de la capa de presentación, lo que permite un comportamiento dinámico sin la necesidad de recargar el navegador.

WebSocket es una tecnología basada en JavaScript que permite la comunicación bidireccional entre cliente y servidor mediante un único enlace TCP. Los WebSockets pueden entenderse como una evolución de Ajax, ya que permiten recopilar información del servidor sin tener que solicitarla expresamente, es decir, la información del servidor fluye hasta el cliente cuando esta es producida por lo que es ideal para implementar la consola de inspección en tiempo real. Para ello, desde la capa cliente se abre un socket TCP con el servidor, el cual emite por este canal toda la información que el docker de la aplicación produce al atender peticiones.

Por último destacar que para los estilos de la capa cliente se ha utilizado la librería CSS Bootstrap [40]

4.1.3. Infraestructura Virtual

Para que el sistema software sea consumido por los clientes, se han disponibilizado dos entornos independientes con las siguientes características:

Entorno de Desarrollo

El entorno <https://coderunner.herokuapp.com> ha estado disponible desde el comienzo de la etapa formal del proyecto. El objetivo de este entorno es facilitar el proceso de desarrollo, ya que como se ha visto en los diagramas de caso de uso y en la sección anterior dedicada al Back-End, el sistema está compuesto por 4 piezas principales y ha sido necesario disponer de ellas en Internet para trabajar en su integración y comportamiento de cara a cumplir con los requisitos propuestos. Este entorno se caracteriza por la siguiente distribución:

1. Servidor principal (code-runner.go): Desplegado en Heroku [25]
2. Capa de persistencia (MongoDB): Consumido desde el servicio de bases de datos gestionadas <https://mlab.com>
3. Gestor de repositorios: Github [8]
4. Demonio Docker: Máquina virtual expuesta a Internet con TLS sobre el proveedor de nube publica Google Cloud(trial) [17]

El objetivo de esta configuración ha sido el de conseguir el conjunto de recursos virtuales necesarios para el proyecto con el menor gasto posible (0€).

Entorno Formal

El entorno <https://apps.gcm-coderunner.com> ha sido creado en las últimas etapas del proyecto, teniendo claras las necesidades a cubrir y prestando especial atención a los posibles cuellos de botella, como puede ser la capacidad de las máquinas virtuales expuestas a Internet para instanciar contenedores. También se ha prestado especial atención en implementar medidas de seguridad como por ejemplo el uso de certificados (TLS) o no consumir servicios de terceros para la capa de persistencia y servir MongoDB de manera asilada. Este entorno se caracteriza por la siguiente distribución:

1. Servidor principal (code-runner.go): Desplegado en VM de Google-Cloud [17]
2. Capa de persistencia (MongoDB): Instalada en la misma VM donde corre el servidor principal y expuesta solo para ser consumida por procesos ejecutados en localhost.
3. Gestor de repositorios: Github github
4. Demonio Docker: Ejecutado en la misma VM que el servidor principal pero expuesto con TLS a localhost, limitando de esta forma los posibles ataques externos que reciba el sistema, pues solo se exponen a Internet los contenedores de los usuarios, no el servicio de contenedores en si.

El objetivo de esta configuración ha sido el de conseguir el conjunto de recursos virtuales necesarios para el proyecto prestando especial atención a la seguridad y también a la capacidad, ya que el dimensionamiento de las VMs de este entorno es notablemente superior al de el entorno de desarrollo, lo que permite la ejecución de números contenedores por diversos usuarios concurrentes. El mantenimiento de este entorno produce gasto directamente proporcional al dimensionamiento de las máquinas virtuales que se utilizan, así como el nombre de dominio usado.

4.1.4. Seguridad**Credenciales de Usuario**

Como se indicó anteriormente, las credenciales de usuario necesarias para utilizar el sistema se corresponden con las credenciales del perfil del usuario en Github. Esto es así debido a que la autenticación

(¿Quién es usted?) y la autorización (¿Qué le permite usted hacer al sistema sobre sus repositorios de código?) son delegadas a Github mediante las denominadas OAuth Apps. Una aplicación OAuth consiste en autenticación delegada a terceros, es decir, cuando el usuario comienza a utilizar este sistema, los servidores de Github validan sus credenciales y emiten un token de acceso con el que realizar operaciones vía API. Este token es almacenado en una sesión del servidor con un tiempo de expiración determinado.

Comunicación Cliente-Servidor

Este proyecto implementa Transport Layer Security (TLS) para garantizar que la información intercambiada entre el servidor y los clientes es segura y viaja encriptada exponiendo su servicio mediante protocolo HTTPS. Por otra parte, la comunicación del servidor con el demonio docker donde se ejecutan las aplicaciones generadas por el usuario también utiliza TLS.

4.2. Caso de Uso

Para mostrar las capacidades del sistema software **GCM:Code-Runner** se presenta el siguiente caso de uso desglosado en las distintas operaciones que el usuario final es capaz de realizar sobre el sistema en cualquiera de los dos entornos disponibles:

1. <https://apps.gcm-coderunner.com>
2. <https://coderunner.herokuapp.com>

En este caso de uso, se va a utilizar el primero de ellos por ser más seguro y disponer de un dimensionamiento mayor (mejores prestaciones).

4.2.1. Autenticación y Autorización

Como paso inicial, el usuario ha de autenticarse y autorizar al sistema a realizar operaciones sobre sus repositorios de código. Para ello, desde un explorador web ha de navegar hasta <https://apps.gcm-coderunner.com>. En la figura 4.3 se presenta la página de bienvenida.

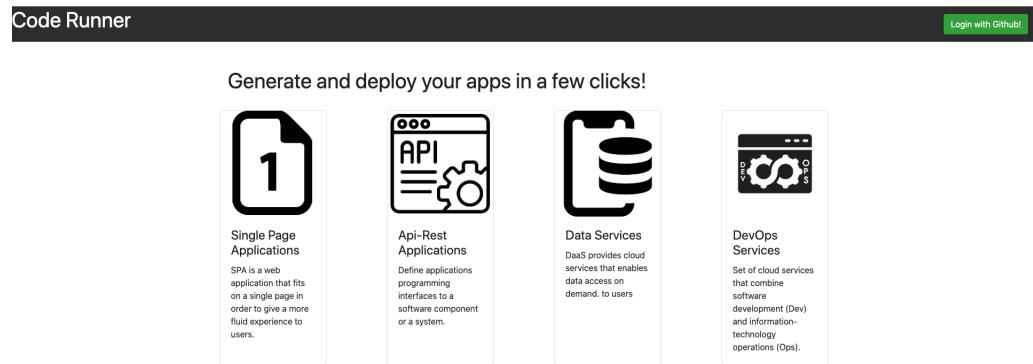


Figura 4.3: Autenticación y Autorización: Index

Para iniciar sesión, es necesario hacer click en el botón de **Login with Github**. La figura 4.4 muestra como el proceso de autenticación es delegado a este servicio.

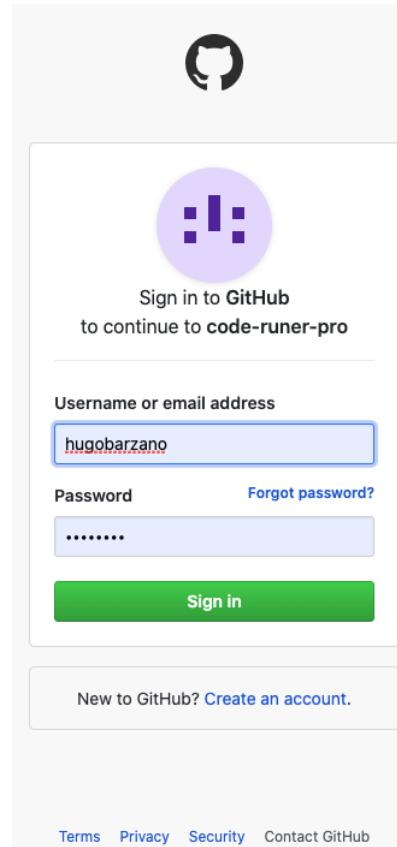


Figura 4.4: Autenticación y Autorización: Autenticación

Es necesario introducir usuario y contraseña correspondientes al perfil de Github. Haciendo click en el botón de **Sign In** el mecanismo de autenticación da paso a la autorización tal y como se muestra en la figura 4.5.

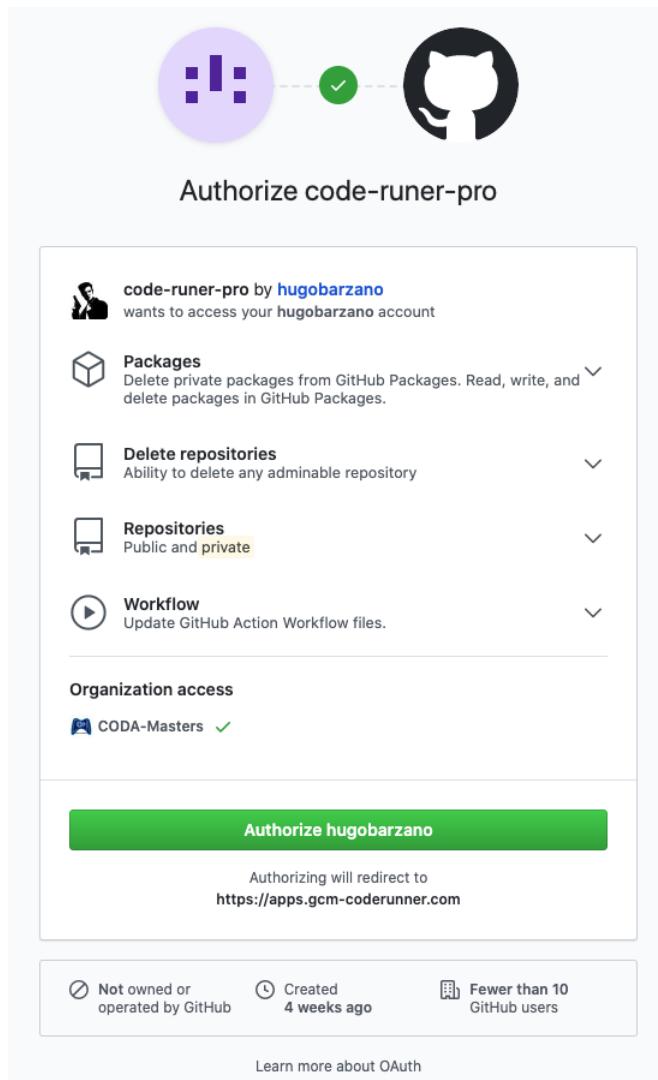
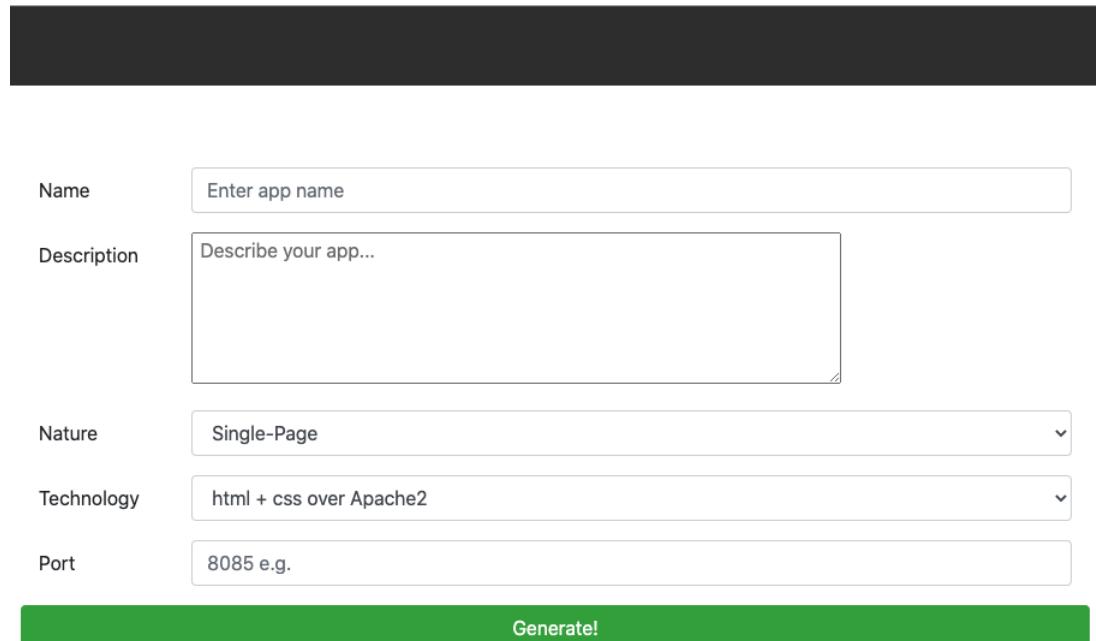


Figura 4.5: Autenticación y Autorización: Autorización

La autorización consiste en conceder permisos al sistema para realizar operaciones en nombre del usuario. Haciendo click en el botón de **Authorize** se conceden permisos para gestionar repositorios, gestionar paquetes, crear releases y utilizar flujos de integración continua. Finalmente accede a su **Workspace** personal.

4.2.2. Generando Aplicaciones

Para comenzar a generar aplicaciones, el usuario ha de hacer click en el botón de **Create App** situado a la izquierda de la barra de navegación superior. El usuario es dirigido al formulario que se muestra en la figura 4.6 donde especificar el tipo de aplicación que desea generar.



The screenshot shows a user interface for creating an application. At the top, there is a large black header bar. Below it, the main form area has the following fields:

- Name:** A text input field labeled "Enter app name".
- Description:** A text area labeled "Describe your app...".
- Nature:** A dropdown menu currently set to "Single-Page".
- Technology:** A dropdown menu currently set to "html + css over Apache2".
- Port:** A text input field containing "8085 e.g.".

At the bottom of the form is a green button labeled "Generate!".

Figura 4.6: Generando Aplicaciones: Formulario de Creación

De manera general, la especificación solicita el nombre de la aplicación, una descripción, la naturaleza de la aplicación, la tecnología y el puerto donde se desea desplegar. La figura 4.7 muestra las naturalezas soportadas en el formulario de creación. Estas naturalezas son:

1. **Single-Page:** Aplicaciones web estáticas que se caracterizan por servir contenido estático en una sola pagina con el objetivo de mejorar la experiencia de usuario.
2. **Api-Rest:** Aplicaciones web dinámicas caracterizadas por poseer lógica de negocio y definir interfaces programáticas sobre sistemas software. En algunos casos se incluyen Interfaces Web para ejecutar operaciones sobre estas interfaces.
3. **Data-Service:** Servicios cloud que ofrecen acceso a datos bajo demanda de los usuarios.

4. **DevOps-Service:** Servicios cloud orientados al desarrollo y operación de sistemas software.

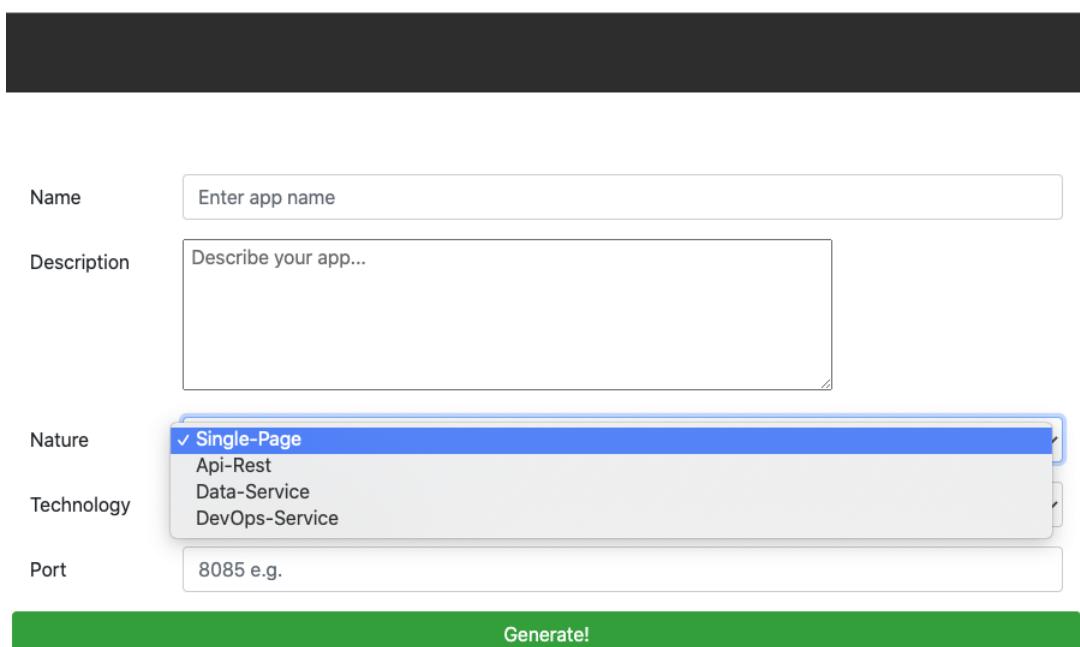
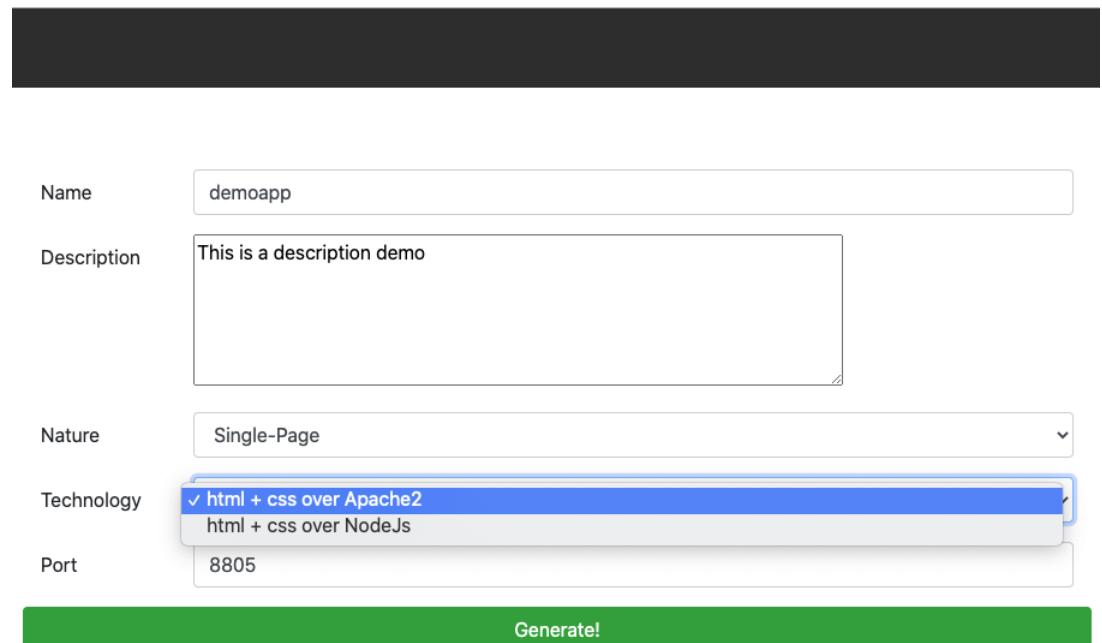


Figura 4.7: Generando Aplicaciones: Naturalezas

Single-Page

La figura 4.8 muestra las tecnologías soportadas para este tipo de aplicación.



The screenshot shows a configuration interface for generating a Single-Page application. The fields are as follows:

- Name:** demoapp
- Description:** This is a description demo
- Nature:** Single-Page
- Technology:** A dropdown menu showing two options: "html + css over Apache2" (selected) and "html + css over NodeJs".
- Port:** 8805

A green button at the bottom right is labeled "Generate!".

Figura 4.8: Generando Aplicaciones: Single-Page

Se establece el puerto de despliegue **8805** y se hace click sobre el botón **Generate**. El usuario es redirigido de nuevo al **Workspace**. La figura 4.9 muestra como la aplicación comienza a ser generada.

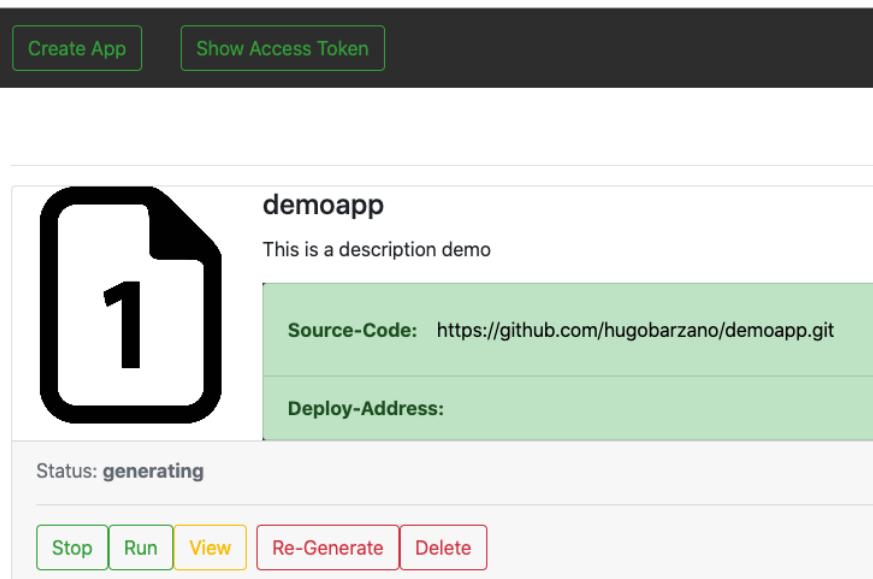


Figura 4.9: Generando Aplicaciones: Single-Page

Toda la producción de código fuente es albergada en un repositorio de código creado en la cuenta del usuario. Haciendo click en el enlace disponible en el apartado **Source-Code** el usuario puede consultar dicho código fuente. La figura 4.10 muestra el repositorio de código para la aplicación.

This is a description demo Edit

Manage topics

- 13 commits 1 branch 0 packages 1 release 0 contributors GPL-3.0

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

File	Description	Time
.github/workflows	Generating CI workflow action .github/workflows/ci.yml	1 minute ago
bin	Generating bin/_pushImage.sh	1 minute ago
html	Generating html/css/style.css	2 minutes ago
Dockerfile	Generating Dockerfile...	1 minute ago
LICENSE	Adding licence...	2 minutes ago
README.md	Starting app...	2 minutes ago
makefile	Generating makefile	1 minute ago

README.md

demoapp

Continuous Integration Workflow no status

Description

This is a description demo

This App has been generated

Timestamp 2020-05-27 16:02:50.769924297 +0000 UTC m=+0.000019236

gcm/0.0.0 source-code

Powered by CesarCorp

Figura 4.10: Generando Aplicaciones: Single-Page Repositorio

Cuando el proceso de generación finaliza, comienza el ciclo de integración continua, el cual se define como una **Github Action**. La figura 4.11 muestra dicho flujo de integración.

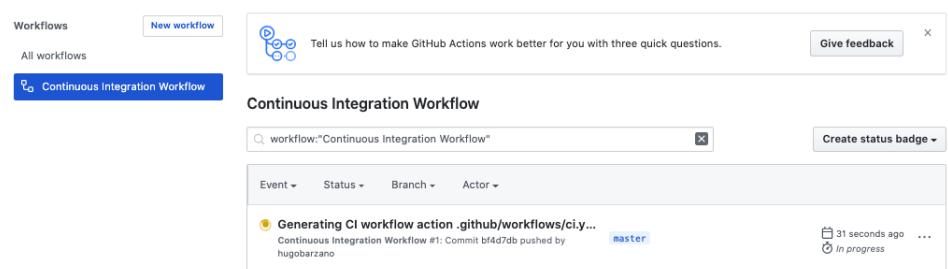


Figura 4.11: Generando Aplicaciones: Single-Page CI

Si el usuario vuelve al **Workspace** puede comprobar que efectivamente la aplicación ha sufrido un cambio de estado. La figura 4.12 muestra el estado **building**.

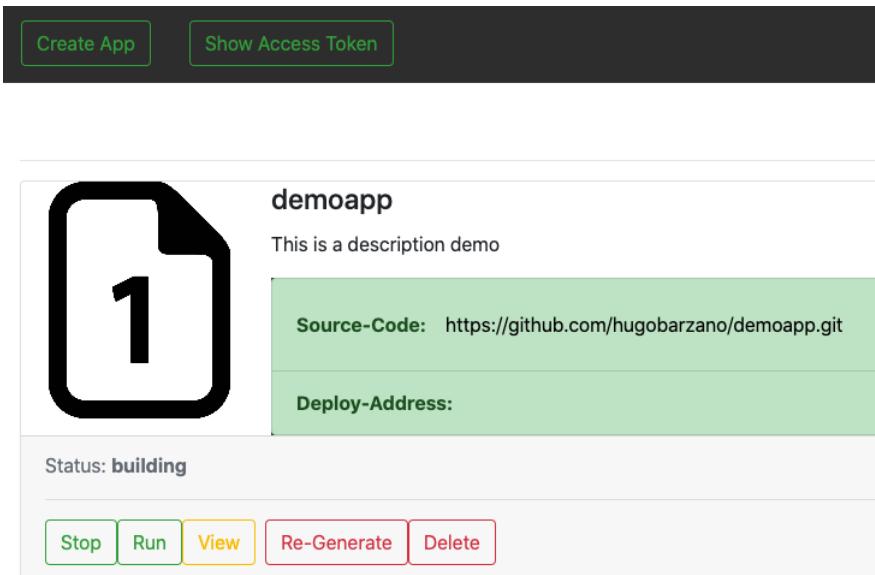


Figura 4.12: Generando Aplicaciones: Single-Page Compilación

Este flujo de integración continua se divide en una serie de fases en las que se ejecutan las pruebas unitarias, se compila la aplicación generada y finalmente se empaqueta el artefacto. La figura 4.13 muestra el ciclo de integración continua completado.

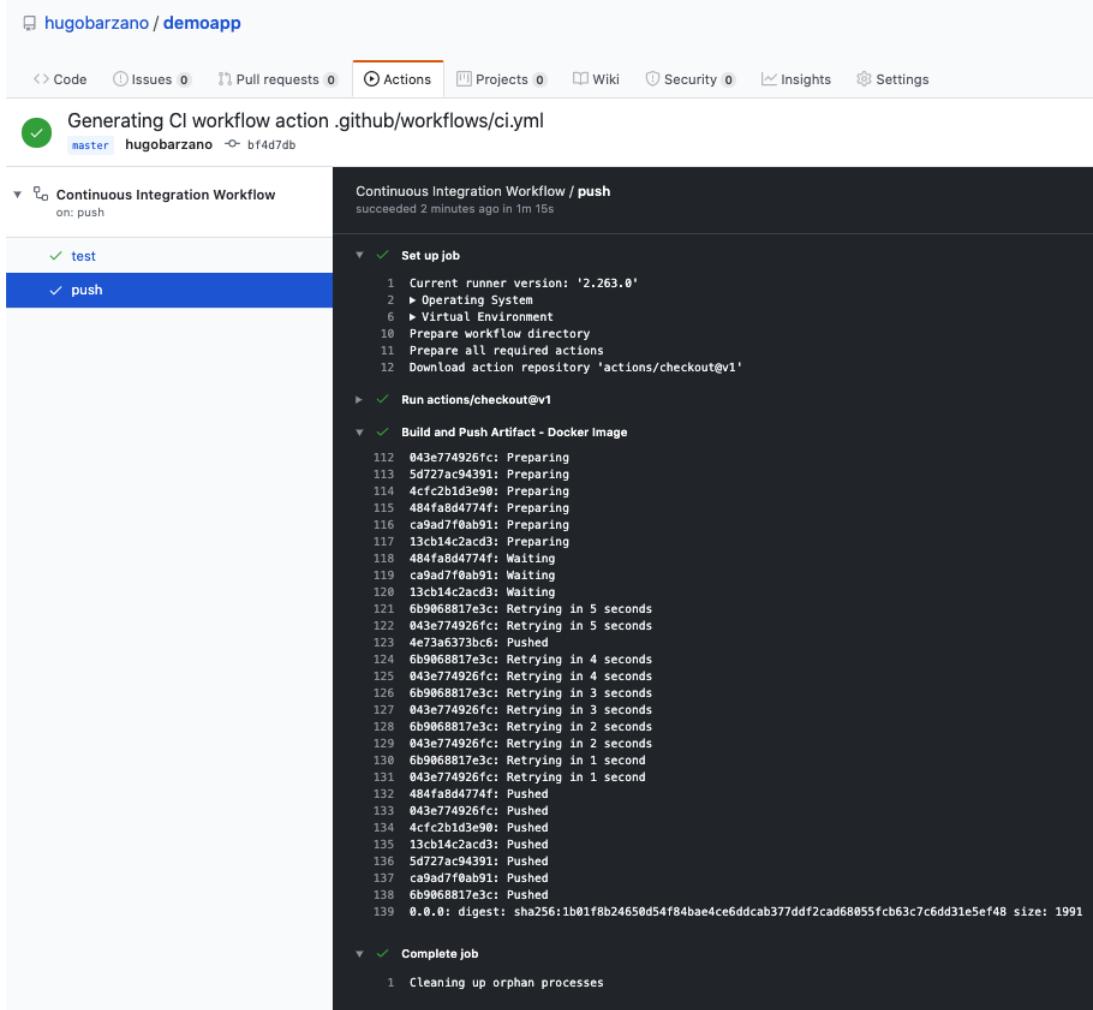


Figura 4.13: Generando Aplicaciones: Single-Page CI Completado

El resultado de este flujo es por una parte un Release que identifica de manera única al estado del código fuente tal y como se muestra en la figura 4.14 y por otra el artefacto que se va a desplegar. La figura 4.15 muestra el artefacto disponible dentro del repositorio de código.

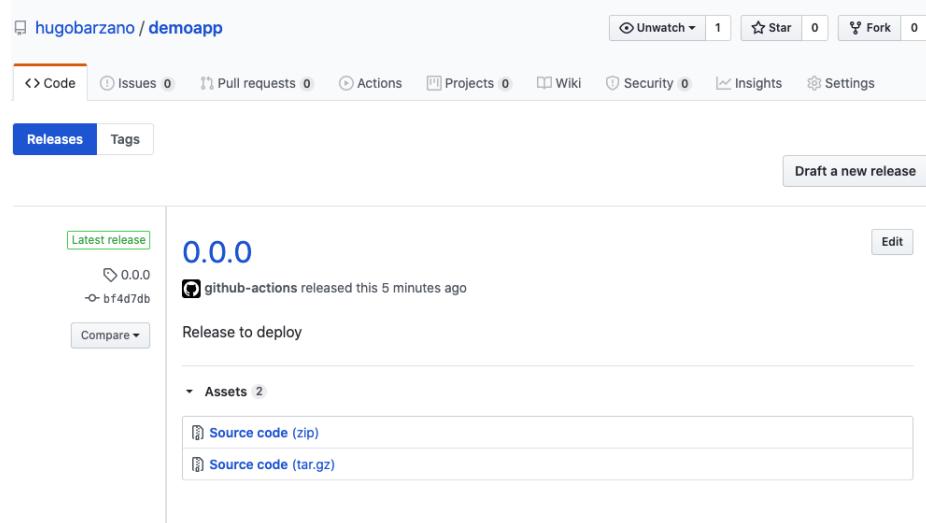


Figura 4.14: Generando Aplicaciones: Single-Page Release

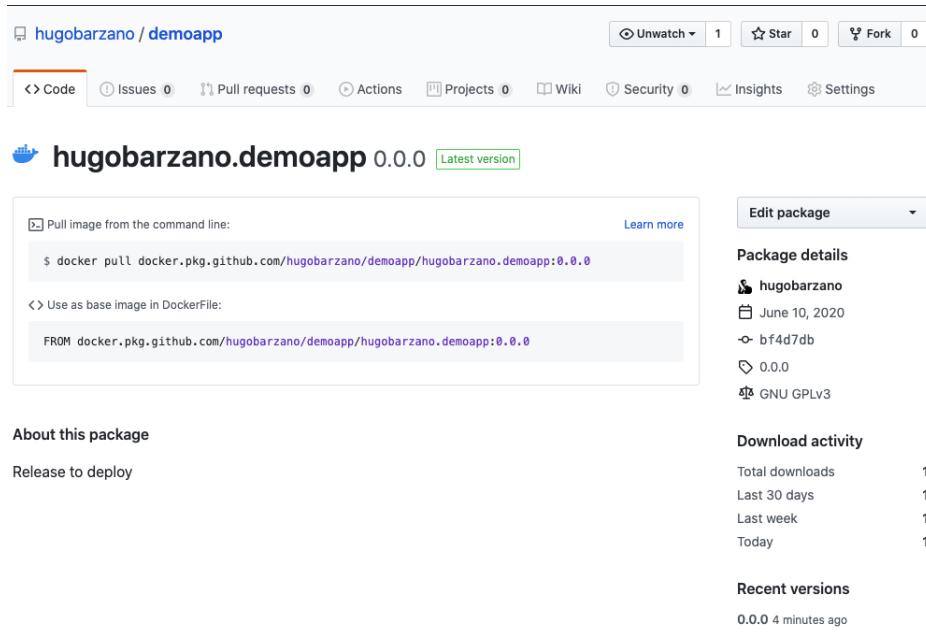


Figura 4.15: Generando Aplicaciones: Single-Page Artefacto

Si el usuario vuelve a poner el foco en su **Workspace** encuentra de nuevo un cambio de estado, la aplicación ha sido desplegada. La figura 4.16 indica el estado **running** de la aplicación.

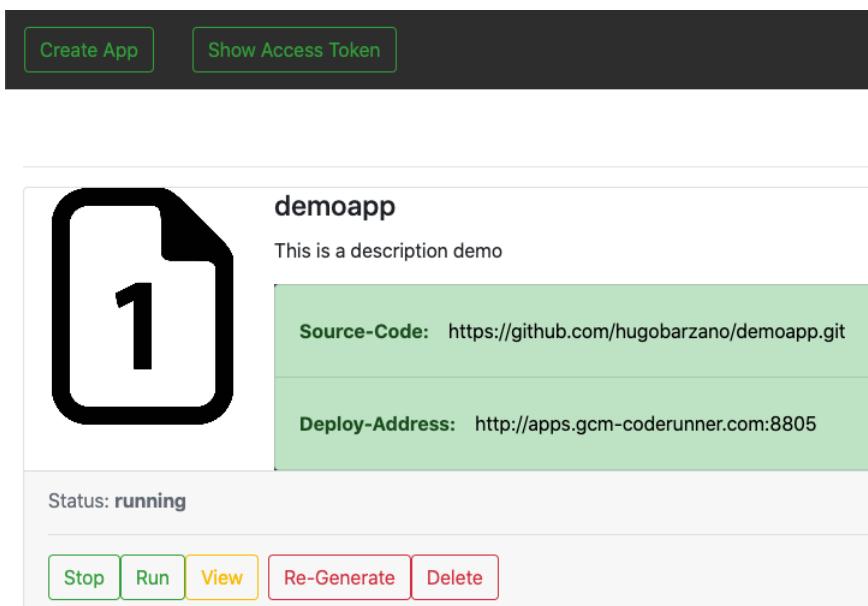


Figura 4.16: Generando Aplicaciones: Single-Page Desplegada

Haciendo uso del enlace disponible en la entrada **Deploy-Address** puede consumirla a través de Internet. La imagen 4.17 muestra una consulta a la aplicación.

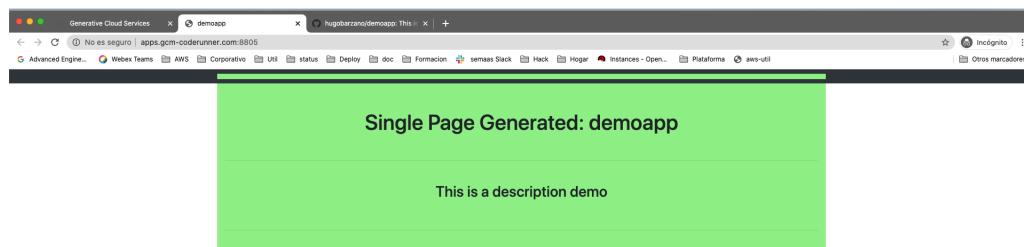


Figura 4.17: Generando Aplicaciones: Single-Page Consultada

Api-Rest

La figura 4.18 muestra las tecnologías para este tipo de aplicación.

The screenshot shows a user interface for generating an API-REST application. The form consists of several input fields:

- Name:** demoapi
- Description:** This is a description on demo
- Nature:** Api-Rest
- Technology:** A dropdown menu showing "go" (selected), "python", and "javascript".
- Model:** An empty text area.
- Port:** 8085 e.g.
- Generate!** A green button at the bottom.

Figura 4.18: Generando Aplicaciones: Api-Rest Tecnología

Una de las principales características de las aplicaciones base REST es que soportan lógica de negocio para realizar operaciones sobre un modelo de datos. Por ello, para esta naturaleza la especificación requiere de un modelo de datos en formato JSON. La figura 4.19 muestra el modelo por defecto que se sirve en el formulario de creación.

The screenshot shows a user interface for generating an API-REST application. The form consists of several input fields:

- Name: demoapi
- Description: This is a description on demo
- Nature: Api-Rest
- Technology: python
- Model: {"active":true, "name":"thing name", "value":33}
- Port: 8085 e.g.

At the bottom is a green button labeled "Generate!".

Figura 4.19: Generando Aplicaciones: Api-Rest Modelo de Datos

Data-Service

La figura 4.20 muestra las tecnologías para este tipo de aplicación.

The screenshot shows a user interface for generating a new application. The form fields are as follows:

- Name: demodb
- Description: This is a description on demo
- Nature: Data-Service
- Technology: A dropdown menu with three options: mongodb (selected), mysql, and redis.
- Port: (Field is empty)

A green "Generate!" button is located at the bottom right of the form.

Figura 4.20: Generando Aplicaciones: Data-Service Tecnología

Esta caracterización abarca principalmente servicios para la persistencia de datos o mecanismo de cache.

DevOps-Service

La figura 4.21 muestra las tecnologías para este tipo de aplicación, presentando un servidor de automatización de tareas como es el caso de Jenkins CI.

Name	demOps
Description	This is a description on demo
Nature	DevOps-Service
Technology	✓ jenkins
Port	8085 e.g.
Generate!	

Figura 4.21: Generando Aplicaciones: DevOps-Service Tecnología

4.2.3. Consumiendo Aplicaciones

El sistema dota al usuario de las operaciones básicas para consumir las aplicaciones generadas. Esta sección exemplifica cómo el usuario es capaz de consumir u operar una API-Rest y un servicio de datos. Las operaciones que se muestran a continuación son comunes a todas las aplicaciones producidas por el sistema.

Consumiendo Api-Rest

Para consumir la aplicación generada **demoapi** el usuario dispone de las siguientes operaciones. Haciendo click en el botón **Stop** detiene el actual despliegue de la aplicación, liberando los recursos virtuales consumidos por ella y cambiando su estado:

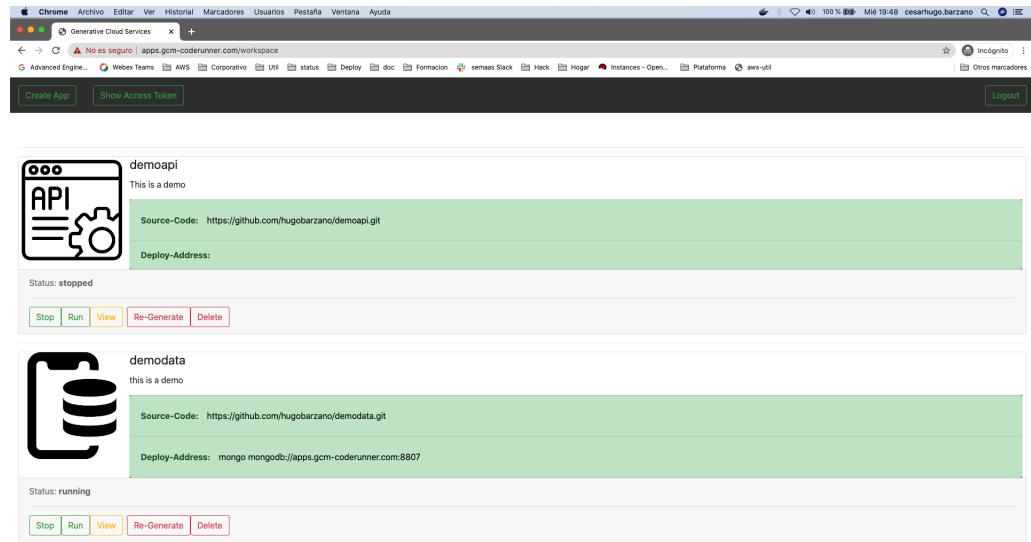


Figura 4.22: Consumiendo Aplicaciones: Api-Rest Stop

Haciendo click en el botón **Run** se produce un nuevo despliegue de la aplicación, utilizando para ello la última versión generada del artefacto (la imagen Docker)

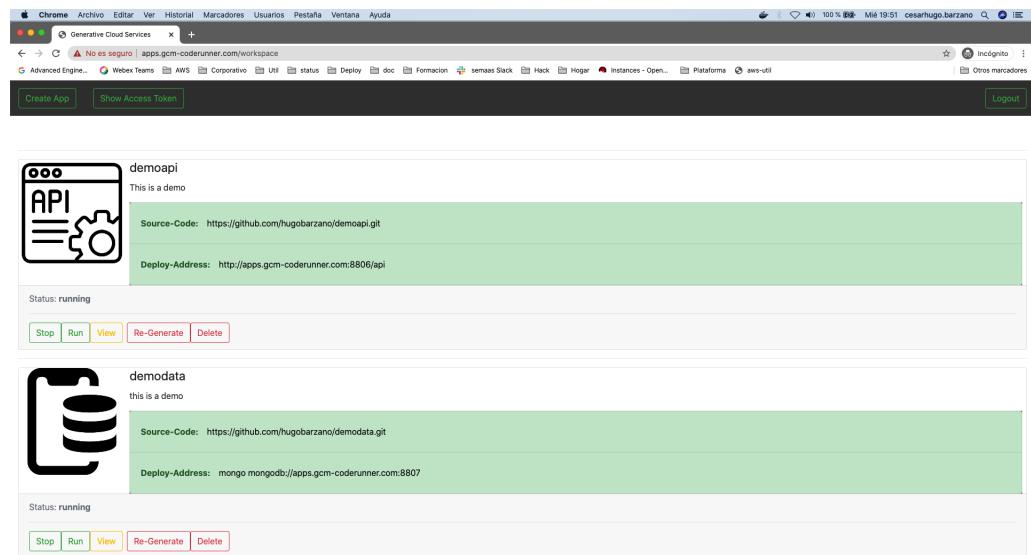


Figura 4.23: Consumiendo Aplicaciones: Api-Rest Run

Haciendo click en el botón **View**, el usuario es redirigido a una vista concreta de la aplicación.

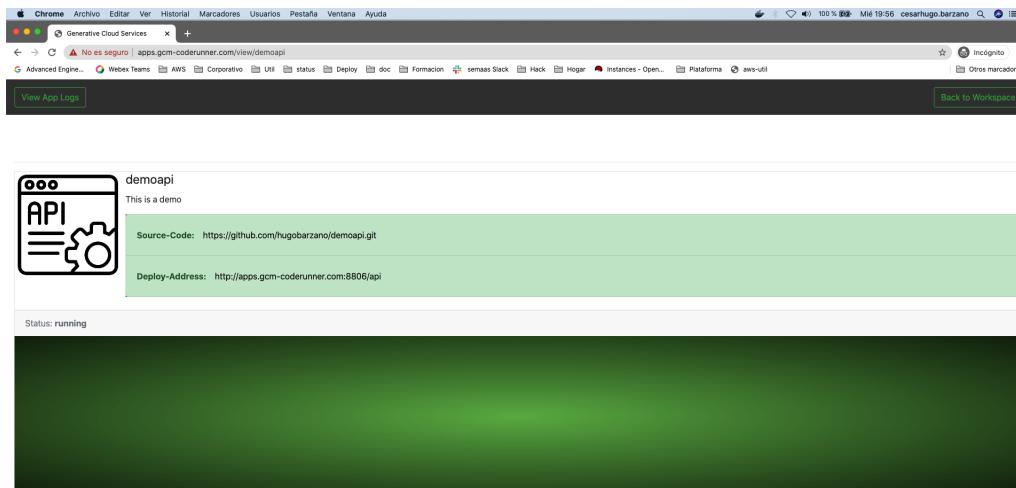


Figura 4.24: Consumiendo Aplicaciones: Api-Rest View

El sistema permite inspeccionar el tráfico atendido por la aplicación en tiempo real haciendo click en el botón **View App Logs**. Esta operación abre un socket TCP con el docker que ejecuta la aplicación y emite en streaming todos los logs producidos por la misma. Las siguientes evidencias muestran operaciones básicas realizadas utilizando la interfaz de usuario que acompaña a las aplicaciones de naturaleza Api-Rest:

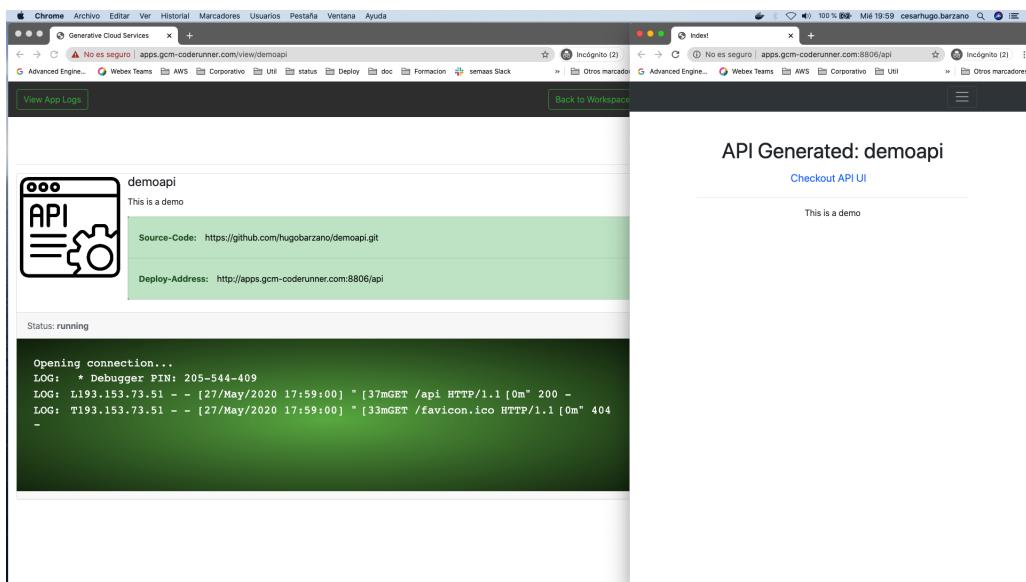


Figura 4.25: Consumiendo Aplicaciones: Api-Rest View Logs Index

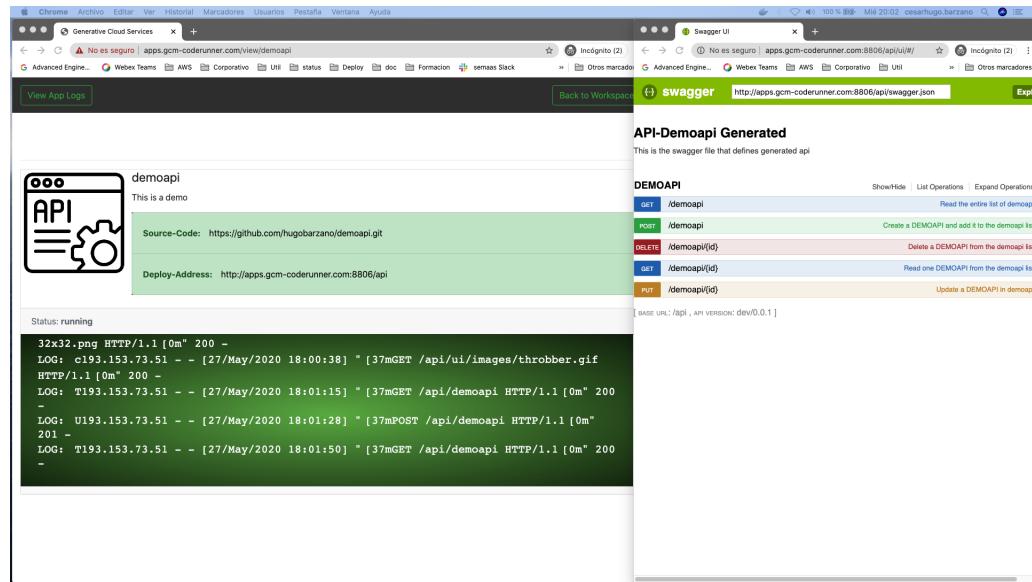


Figura 4.26: Consumiendo Aplicaciones: Api-Rest View Logs UI

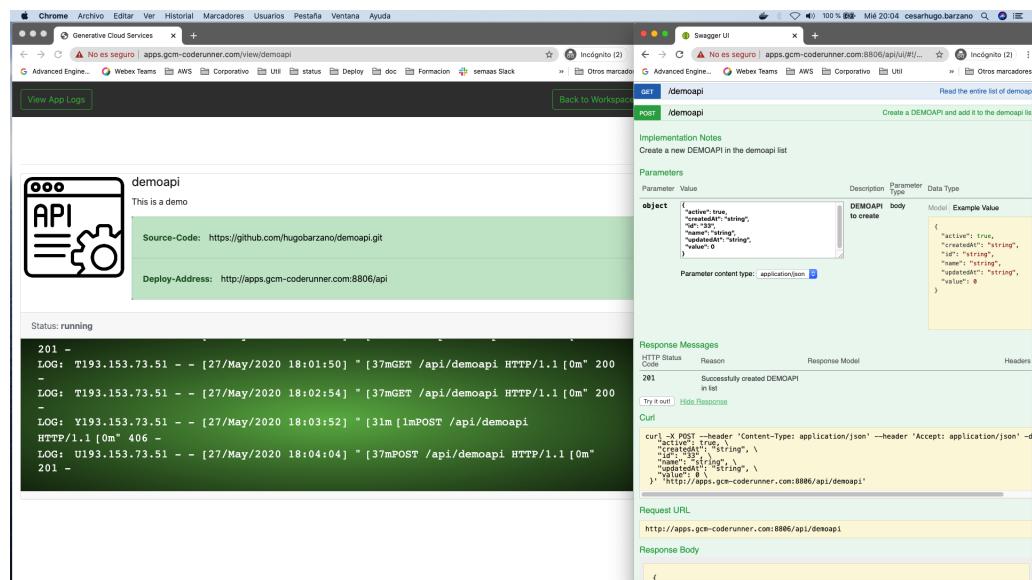


Figura 4.27: Consumiendo Aplicaciones: Api-Rest View Logs POST

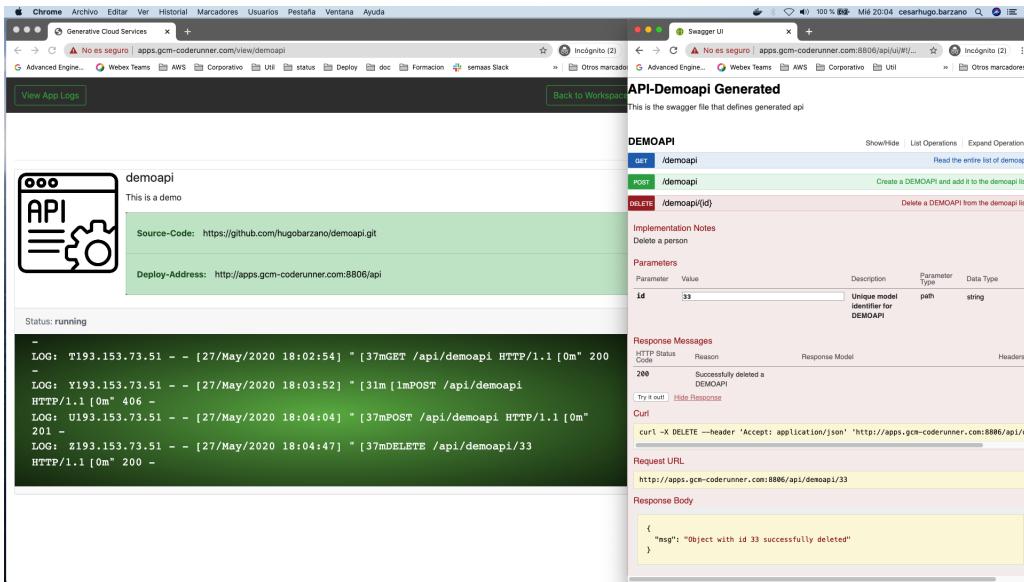


Figura 4.28: Consumiendo Aplicaciones: Api-Rest View Logs DELETE

Consumiendo Data-Service

Continuando con el caso de uso de como consumir las aplicaciones generadas se presenta un servicio de datos. En este ejemplo el usuario inspecciona una base de datos MongoDB:

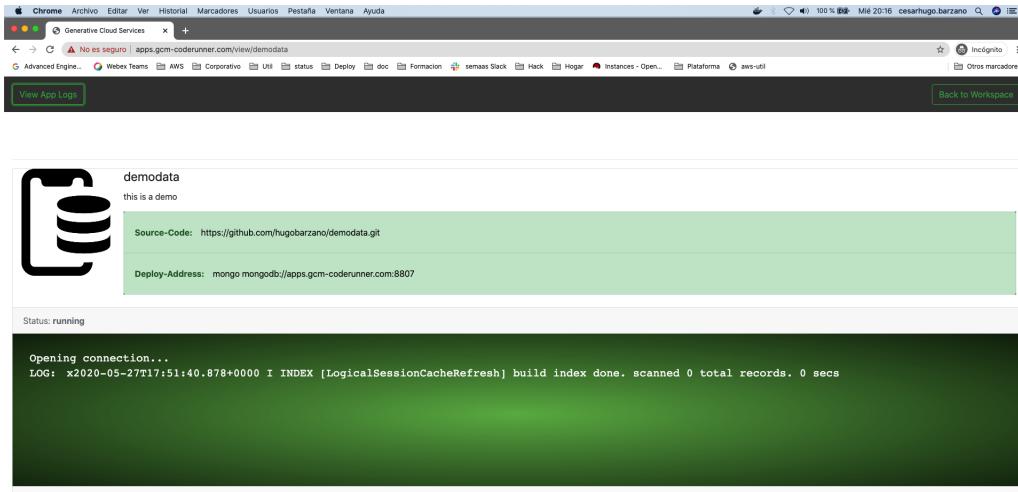
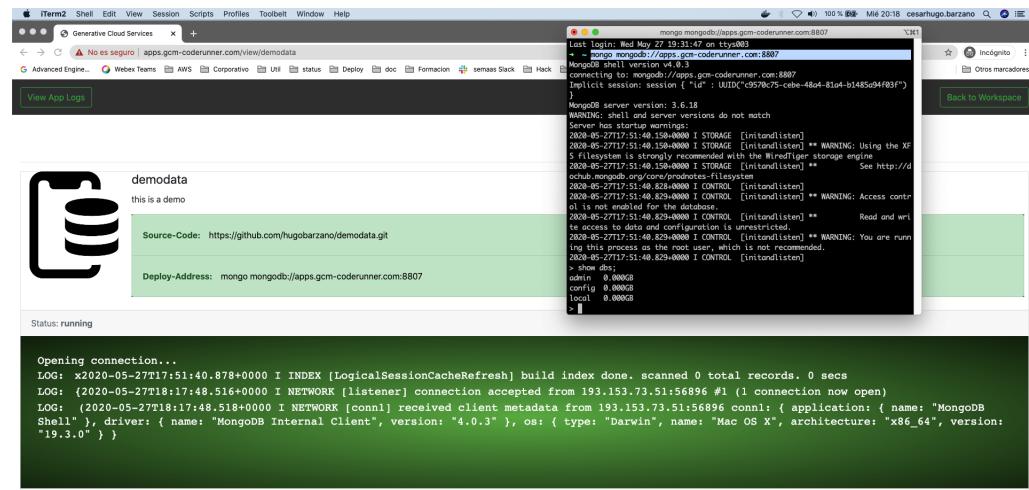


Figura 4.29: Consumiendo Aplicaciones: Data-Service View Logs

En este caso de uso, es necesario un cliente por linea de comandos para este tipo de base de datos haciendo uso de la terminal de un sistema operativo, los logs de la aplicación registran el numero de nuevas conexiones:



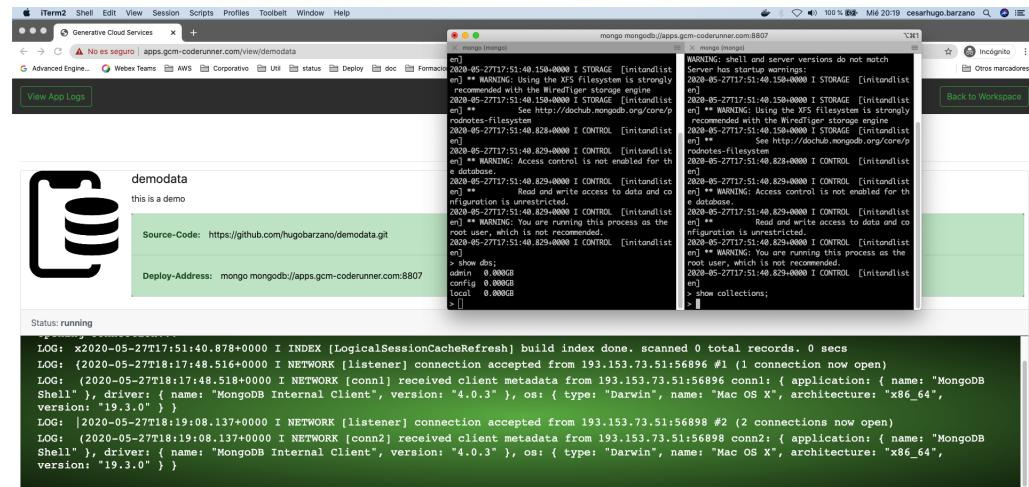
```
mongo mongoDB://apps.gcm-coderunner.com:8807
Last Login: Wed May 27 19:31:47 on ttys003
MongoDB shell version v4.0.3
connecting to: mongoDB://apps.gcm-coderunner.com:8807
Implicit session: session { "id" : UUID("c957bc75-cebe-48e4-81d4-b1485d4f83f") }
MongoDB server version: 3.6.18
WARNING: shell and server versions do not match
Server has startup warnings:
2020-05-27T17:51:48.156+0000 I STORAGE  [initandlisten]
2020-05-27T17:51:48.156+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2020-05-27T17:51:48.156+0000 I STORAGE  [initandlisten] ** See http://dochub.mongodb.org/cORE/products/filesystems
2020-05-27T17:51:48.823+0000 I CONTROL [initandlisten]
2020-05-27T17:51:48.823+0000 I CONTROL [initandlisten] ** WARNING: Access contr
ol is disabled for this process as the root user, which is not recommended.
2020-05-27T17:51:48.823+0000 I CONTROL [initandlisten] ** Read and wr
ite access to data and configuration is unrestricted.
2020-05-27T17:51:48.823+0000 I CONTROL [initandlisten] ** WARNING: You are runn
ing this process as the root user, which is not recommended.
> show dbs;
admin 0.000GB
config 0.000GB
local 0.000GB
> 
```

demodata
this is a demo
Source-Code: https://github.com/hugobarzano/demodata.git
Deploy-Address: mongo mongoDB://apps.gcm-coderunner.com:8807

Status: running

```
Opening connection...
LOG: x2020-05-27T17:51:40.878+0000 I INDEX [LogicalSessionCacheRefresh] build index done. scanned 0 total records. 0 secs
LOG: {2020-05-27T18:17:48.516+0000 I NETWORK [listener] connection accepted from 193.153.73.51:56896 #1 (1 connection now open)
LOG: {2020-05-27T18:17:48.518+0000 I NETWORK [conn1] received client metadata from 193.153.73.51:56896 conn1: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "4.0.3" }, os: { type: "Darwin", name: "Mac OS X", architecture: "x86_64", version: "19.3.0" } }
```

Figura 4.30: Consumiendo Aplicaciones: Data-Service View Logs I



```
mongo mongoDB://apps.gcm-coderunner.com:8807
mongo mongoDB://apps.gcm-coderunner.com:8807
Last Login: Wed May 27 19:31:47 on ttys003
MongoDB shell version v4.0.3
connecting to: mongo mongoDB://apps.gcm-coderunner.com:8807
Implicit session: session { "id" : UUID("c957bc75-cebe-48e4-81d4-b1485d4f83f") }
MongoDB server version: 3.6.18
WARNING: shell and server versions do not match
Server has startup warnings:
2020-05-27T17:51:48.156+0000 I STORAGE  [initandlisten]
2020-05-27T17:51:48.156+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2020-05-27T17:51:48.156+0000 I STORAGE  [initandlisten] ** See http://dochub.mongodb.org/cORE/products/filesystems
2020-05-27T17:51:48.823+0000 I CONTROL [initandlisten]
2020-05-27T17:51:48.823+0000 I CONTROL [initandlisten] ** WARNING: Access contr
ol is disabled for this process as the root user, which is not recommended.
2020-05-27T17:51:48.823+0000 I CONTROL [initandlisten] ** Read and wr
ite access to data and configuration is unrestricted.
2020-05-27T17:51:48.823+0000 I CONTROL [initandlisten] ** WARNING: You are runn
ing this process as the root user, which is not recommended.
> show dbs;
admin 0.000GB
config 0.000GB
local 0.000GB
> 
```

demodata
this is a demo
Source-Code: https://github.com/hugobarzano/demodata.git
Deploy-Address: mongo mongoDB://apps.gcm-coderunner.com:8807

Status: running

```
LOG: x2020-05-27T17:51:40.878+0000 I INDEX [LogicalSessionCacheRefresh] build index done. scanned 0 total records. 0 secs
LOG: {2020-05-27T18:17:48.516+0000 I NETWORK [listener] connection accepted from 193.153.73.51:56896 #1 (1 connection now open)
LOG: {2020-05-27T18:17:48.518+0000 I NETWORK [conn1] received client metadata from 193.153.73.51:56896 conn1: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "4.0.3" }, os: { type: "Darwin", name: "Mac OS X", architecture: "x86_64", version: "19.3.0" } }
LOG: |2020-05-27T18:19:08.137+0000 I NETWORK [listener] connection accepted from 193.153.73.51:56898 #2 (2 connections now open)
LOG: {2020-05-27T18:19:08.137+0000 I NETWORK [conn2] received client metadata from 193.153.73.51:56898 conn2: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "4.0.3" }, os: { type: "Darwin", name: "Mac OS X", architecture: "x86_64", version: "19.3.0" } }
```

Figura 4.31: Consumiendo Aplicaciones: Data-Service View Logs II

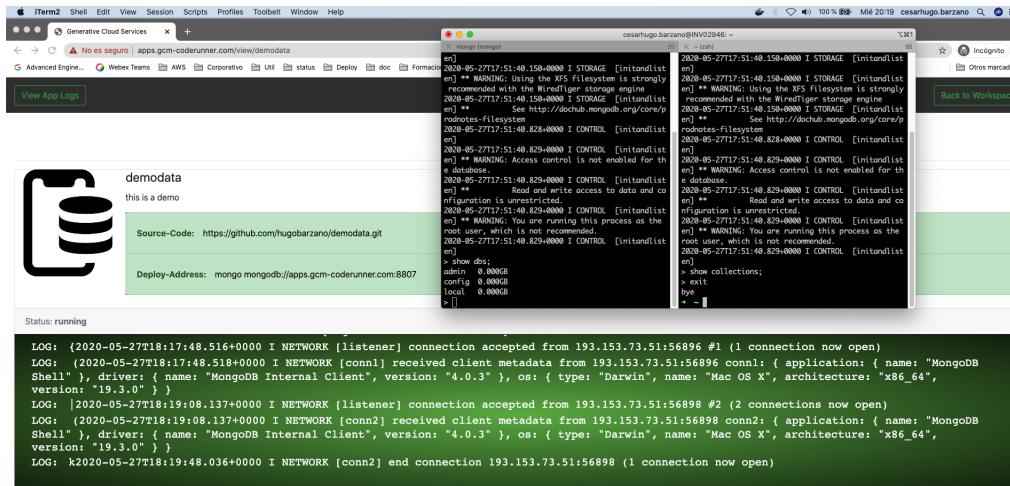


Figura 4.32: Consumiendo Aplicaciones: Data-Service View Logs III

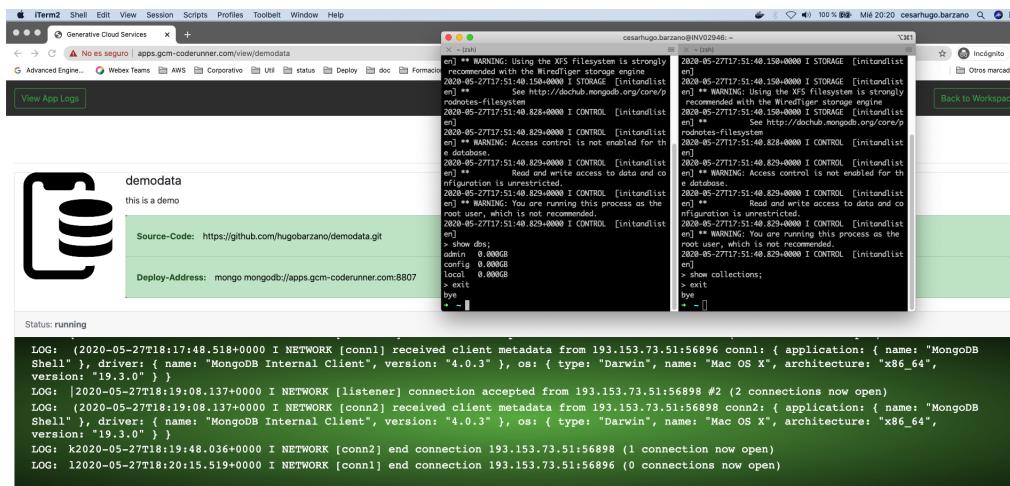


Figura 4.33: Consumiendo Aplicaciones: Data-Service View Logs IV

4.2.4. Actualizando Aplicaciones

El sistema proporciona al usuario los mecanismos necesarios para evolucionar las aplicaciones. De esta forma permite que el usuario maneje, pruebe, mejore y aprenda utilizando el código generado. En última

instancia el usuario es capaz de re-generar la aplicación devolviéndola a su estado inicial. El sistema presenta los siguientes mecanismos:

Flujo Continuo de Integración Software

El primer mecanismo integrado en todas las aplicaciones es el flujo de integración y entrega continuos que permiten al usuario integrar nuevas funcionalidades siguiendo buenas prácticas, para ello, dada una aplicación Single-Page en ejecución:

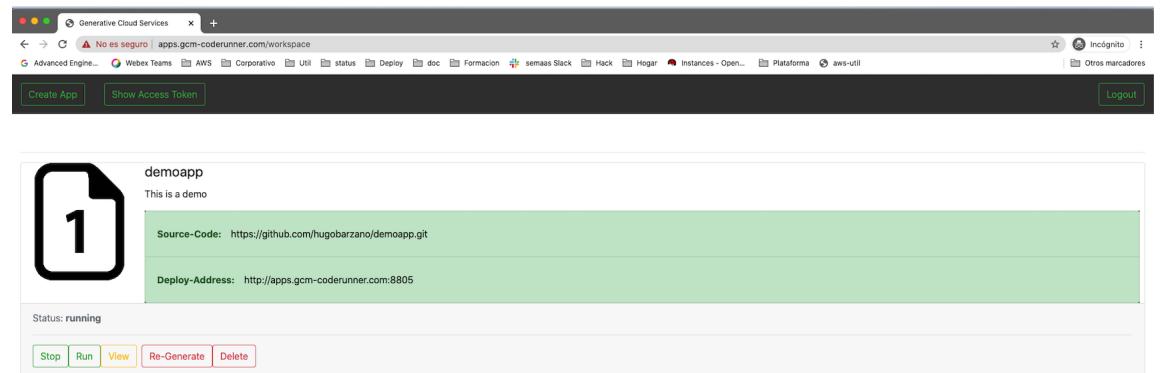


Figura 4.34: Actualizando Aplicaciones: Single-Page en Ejecución

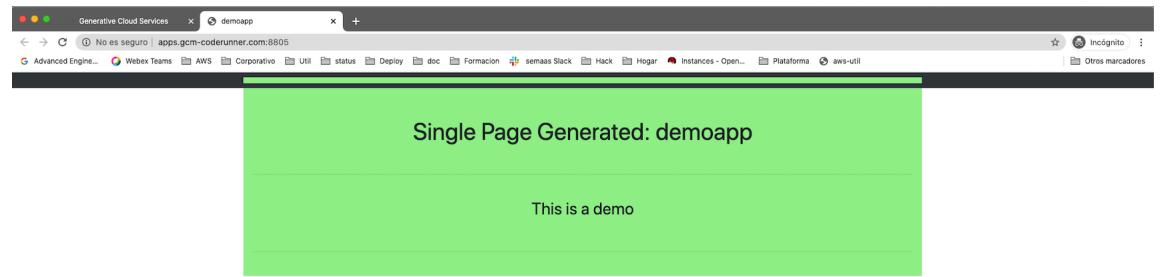


Figura 4.35: Actualizando Aplicaciones: Single-Page Consulta

El usuario ha de clonar la aplicación en su sistema local, haciendo uso de la URL del repositorio proporcionada:

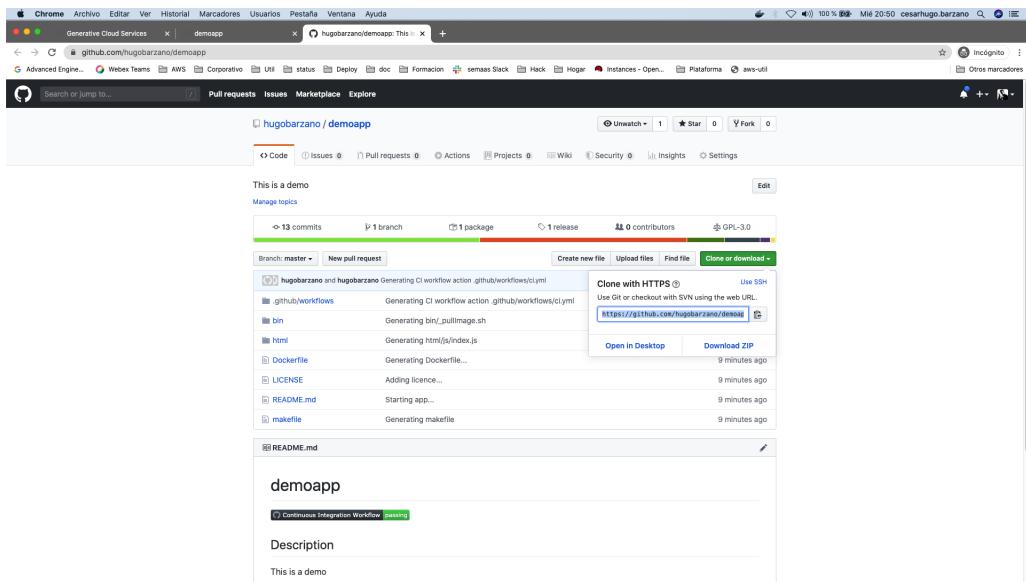


Figura 4.36: Actualizando Aplicaciones: Single-Page Clonar I

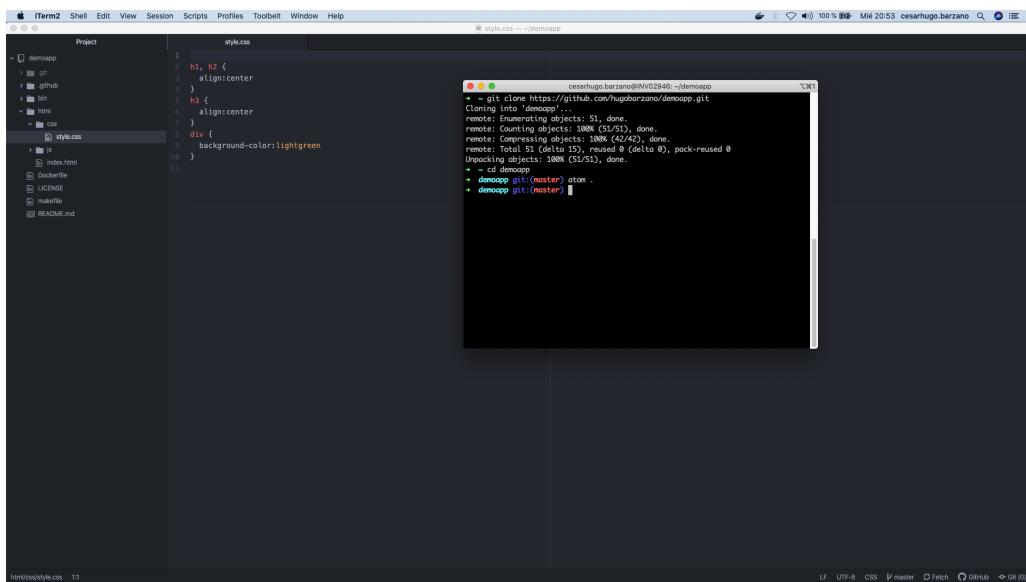


Figura 4.37: Actualizando Aplicaciones: Single-Page Clonar II

```

1 git clone https://github.com/hugobarzano/demoapp.git
2 cd demoapp
3 atom . #editor de texto

```

Ha de integrar nueva funcionalidad, en el caso de esta aplicación, modificar el color de la hoja de estilos CSS:

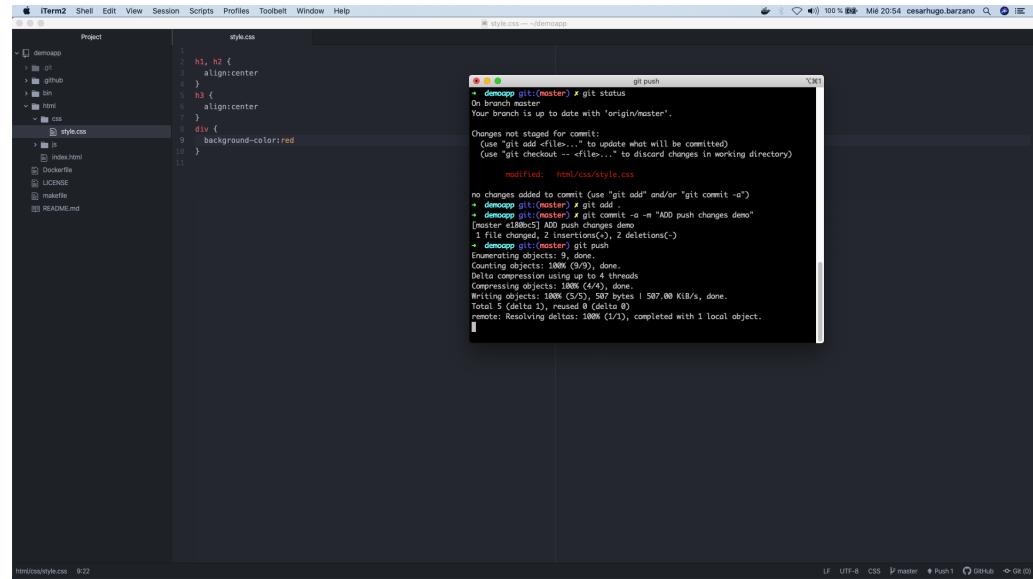


Figura 4.38: Actualizando Aplicaciones: Single-Page CSS

Subir los cambios a la rama Master del repositorio:

```

1 git add .
2 git commit -a -m "ADD push changes demo"
3 git push

```

El flujo de integración continua se dispara ante el evento **push** relacionado con subir nuevo código al repositorio y de nuevo somete a la aplicación a las mismas etapas que cuando fue generada:

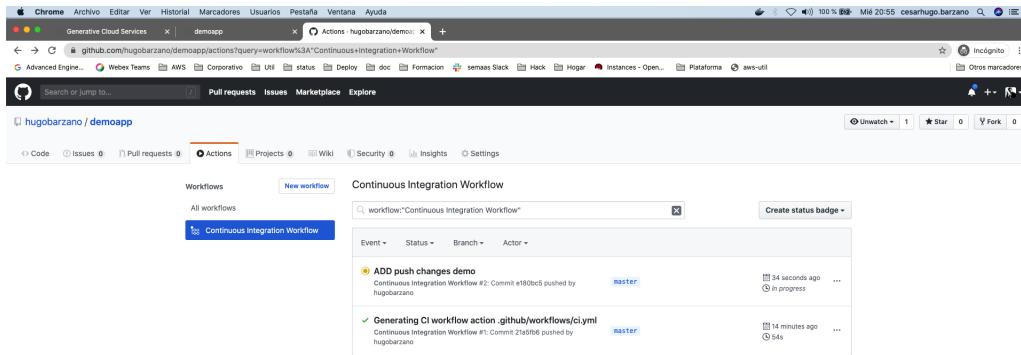


Figura 4.39: Actualizando Aplicaciones: Single-Page CI Start

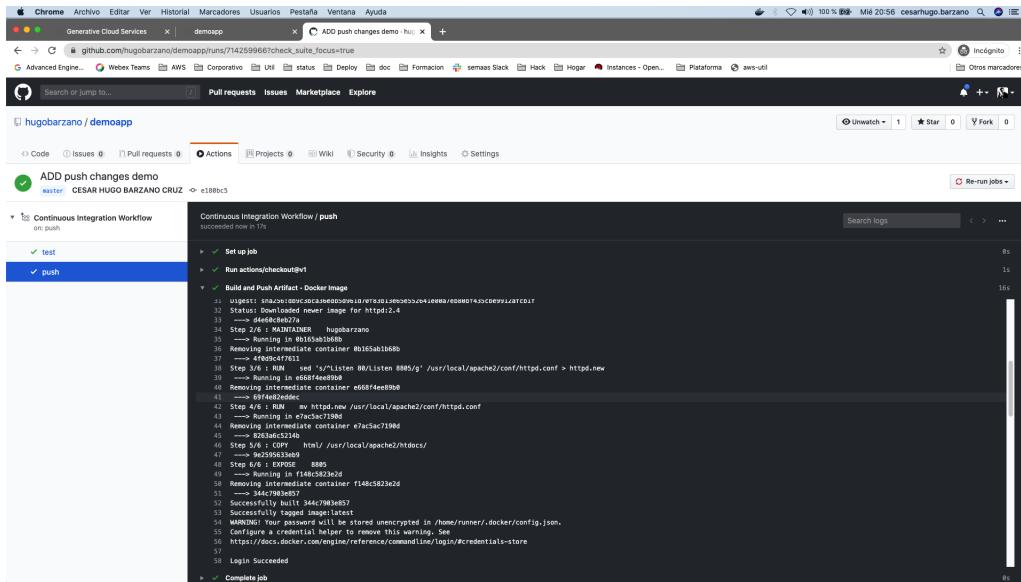


Figura 4.40: Actualizando Aplicaciones: Single-Page CI Finish

Una vez completado el ciclo de integración continua, el usuario percibe que una nueva Release y un nuevo Arterfacto están disponibles en el repositorio:

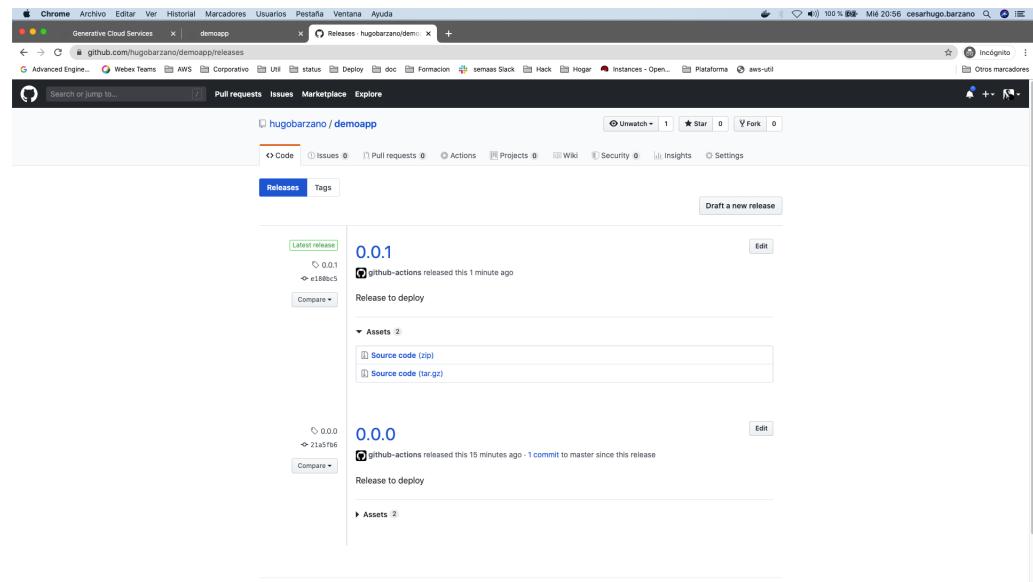


Figura 4.41: Actualizando Aplicaciones: Single-Page Release 0.0.1

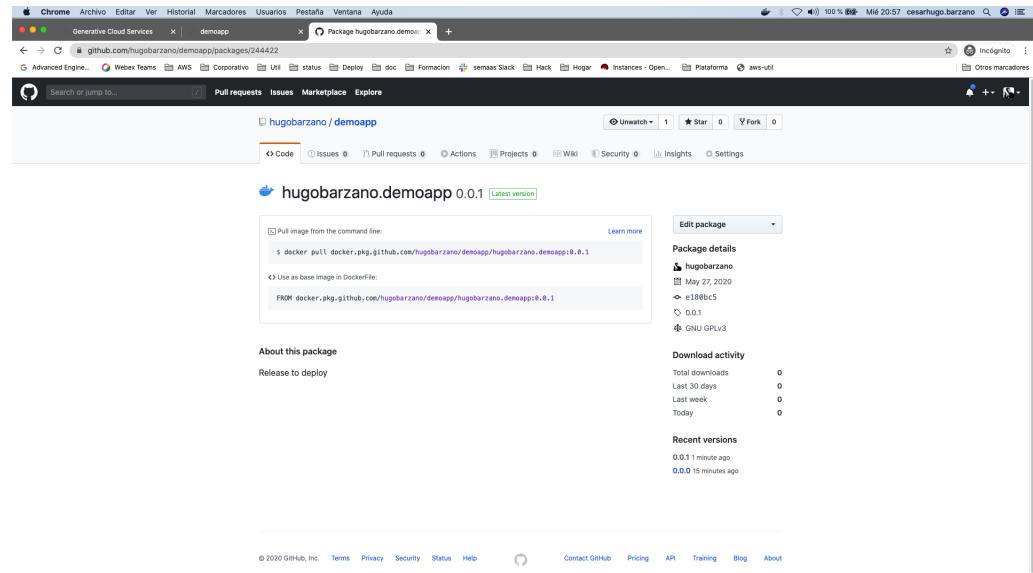


Figura 4.42: Actualizando Aplicaciones: Single-Page Artefacto 0.0.1

El usuario ha de poner el foco en su **Workspace**, hacer clic en el botón **Stop** y de nuevo en el botón **Run**, desplegando así la nueva versión de la aplicación.

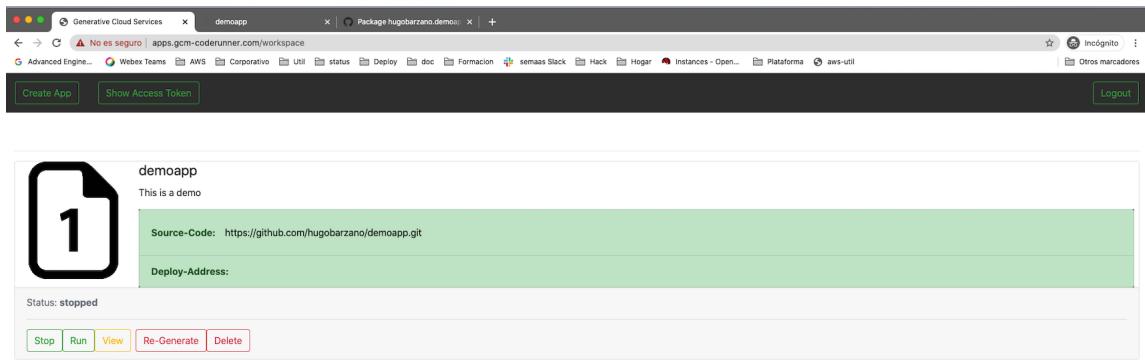


Figura 4.43: Actualizando Aplicaciones: Single-Page Stop 0.0.0

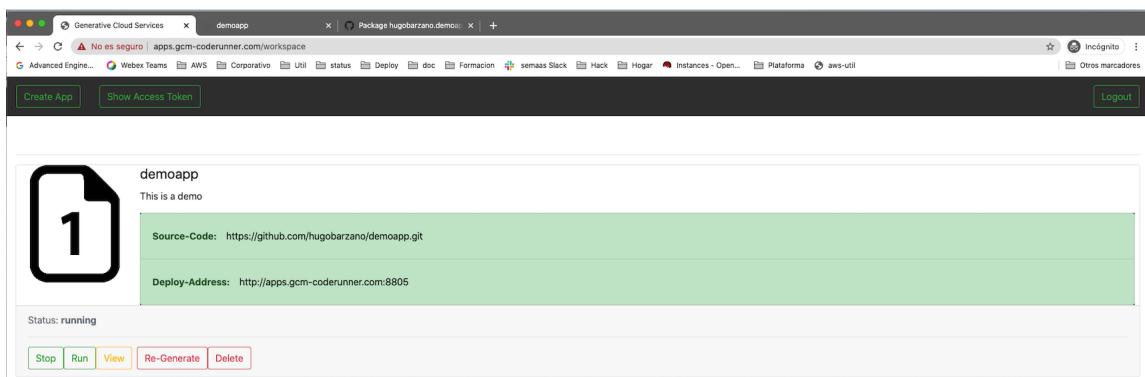


Figura 4.44: Actualizando Aplicaciones: Single-Page Run 0.0.1

Si ahora vuelve a consumir la aplicación en la dirección de despliegue proporcionada, percibe que el cambio ha sido aplicado y desplegado.

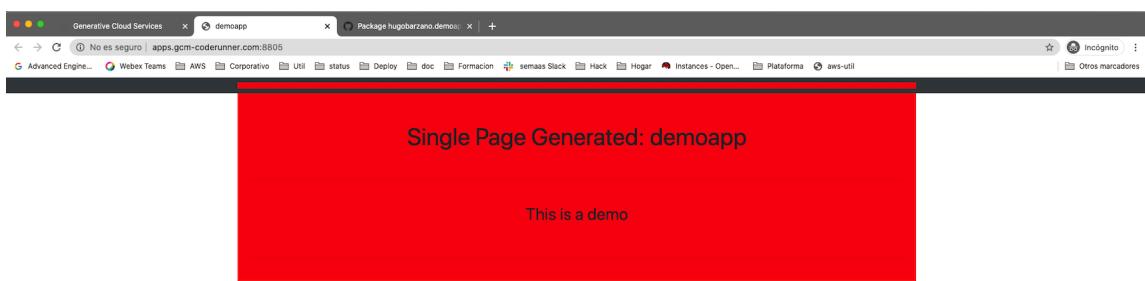


Figura 4.45: Actualizando Aplicaciones: Single-Page Consulta 0.0.1

Trabajo Local

El sistema produce las herramientas necesarias para trabajar en local con las aplicaciones. Para poder interactuar con el repositorio de código y realizar operaciones no públicas, como por ejemplo compilar y subir una nueva versión sin seguir el ciclo de integración software es necesario utilizar el token de acceso del usuario. Con su foco en el **Workspace**, el usuario ha de hacer click en el botón **Show Access Token**

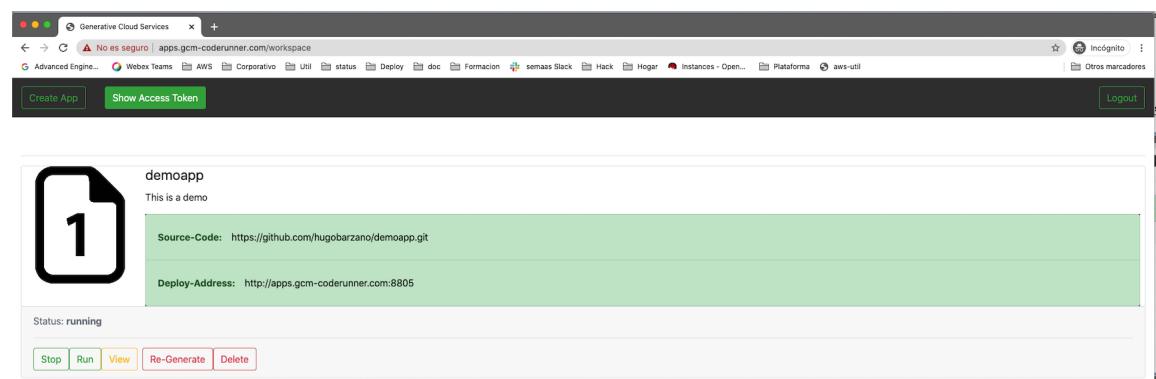


Figura 4.46: Trabajo Local: Show Access Token

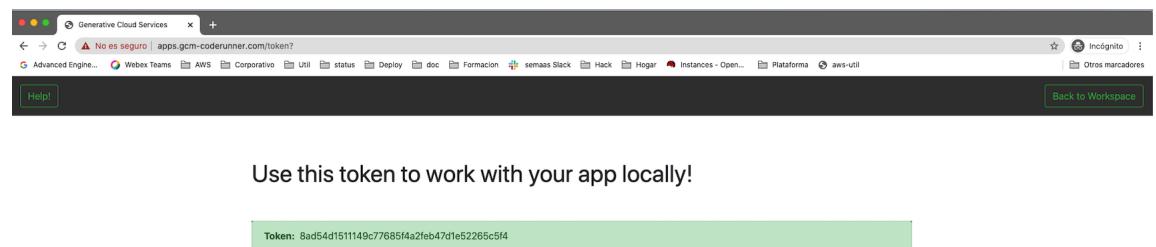


Figura 4.47: Trabajo Local: Access Token

La idea es que el sistema ilustre al usuario en como ha de trabajar con las aplicaciones generadas, por lo que haciendo click en el botón **Help!** el usuario dispone de una guía con las operaciones que puede realizar localmente sobre la aplicación:

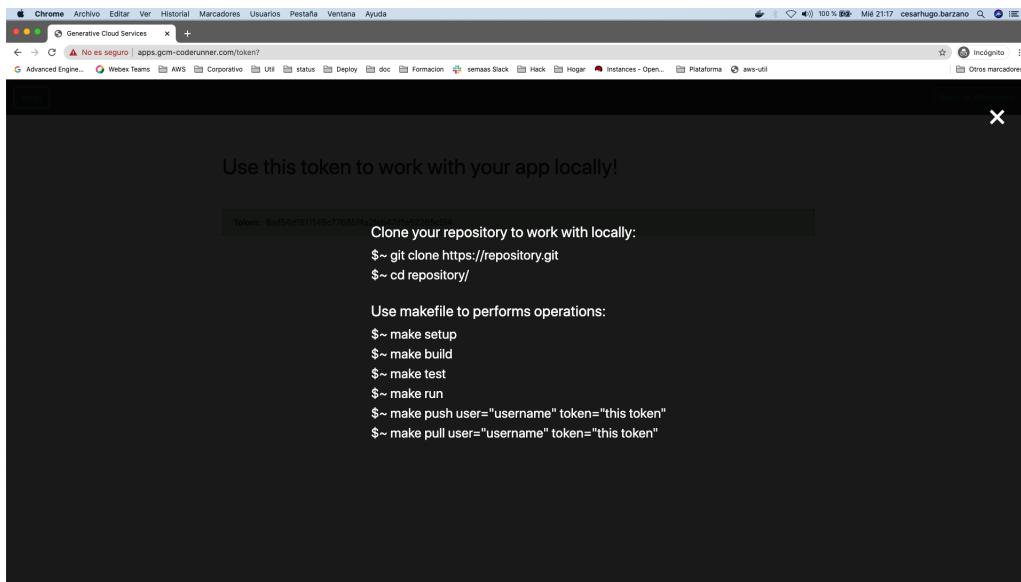


Figura 4.48: Trabajo Local: Help

Por ejemplo, si el usuario integra una nueva funcionalidad sobre la aplicación actual (0.0.1) como es el cambio de rojo a azul en la hoja de estilos, puede producir y desplegar una nueva versión sin ejecutar el ciclo de integración continua:

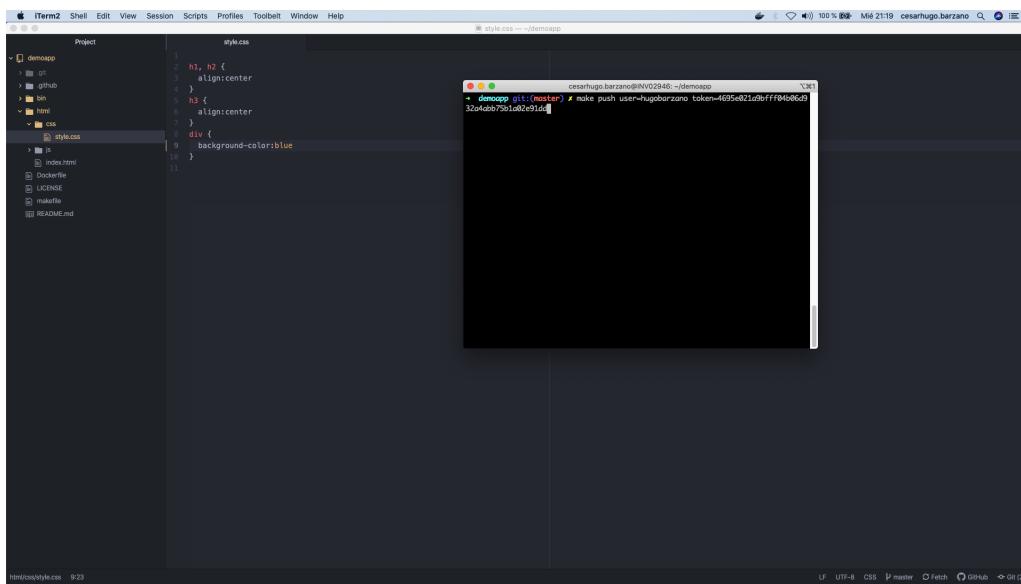


Figura 4.49: Trabajo Local: Single-Page Actualización CSS

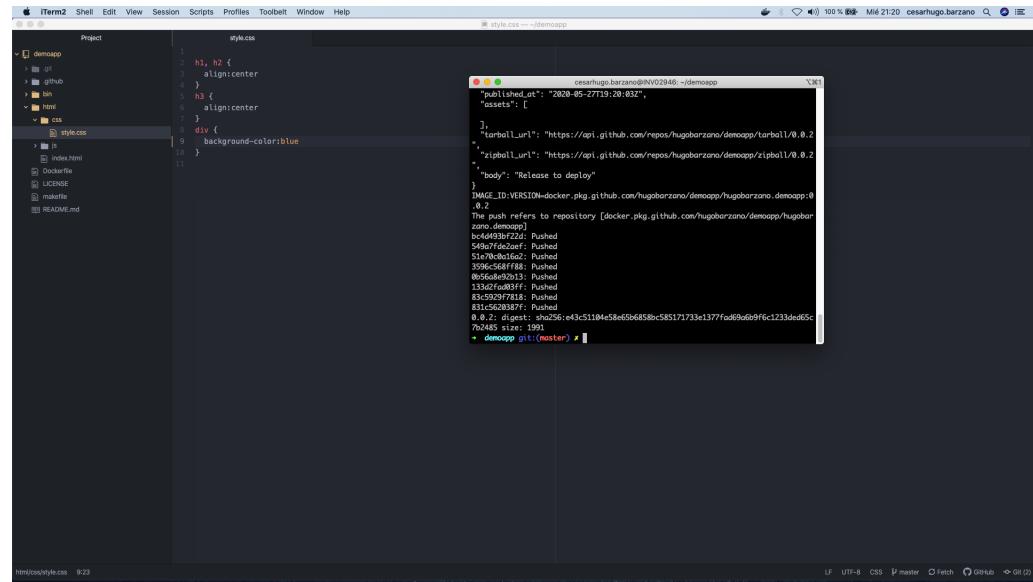


Figura 4.50: Trabajo Local: Single-Page make push

De esta forma, se crean una nueva Release y un nuevo artefacto :

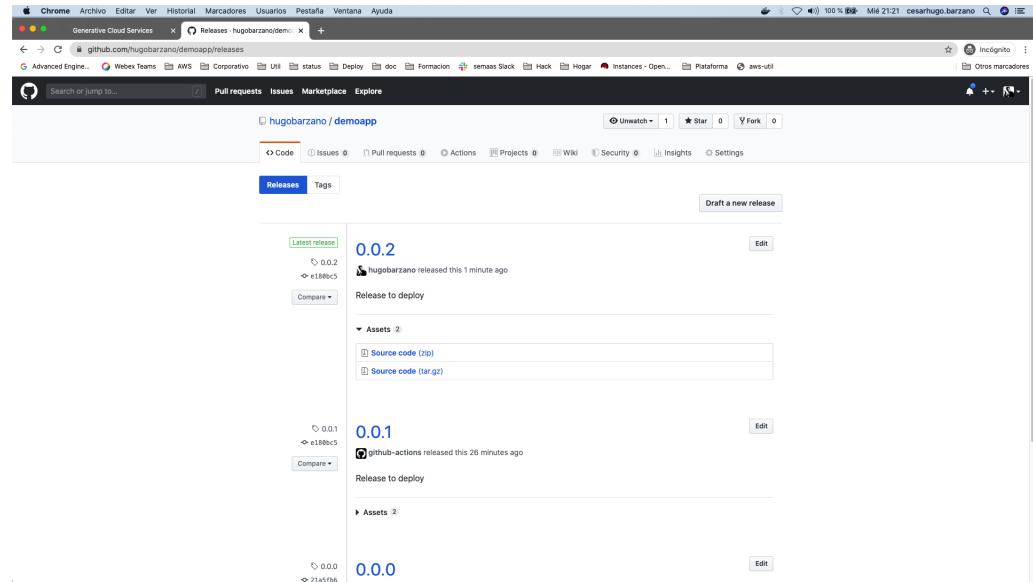


Figura 4.51: Trabajo Local: Single-Page Release 0.0.2

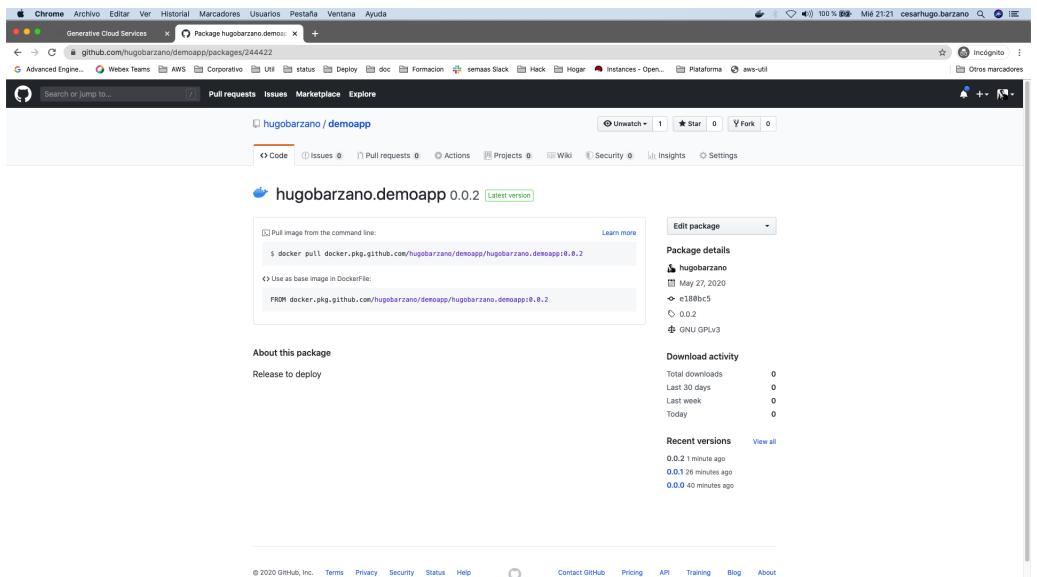


Figura 4.52: Trabajo Local: Single-Page Artefacto 0.0.2

El usuario ha de poner el foco en su **Workspace**, hacer clic en el botón **Stop** y de nuevo en el botón **Run**, desplegando así la nueva versión de la aplicación.

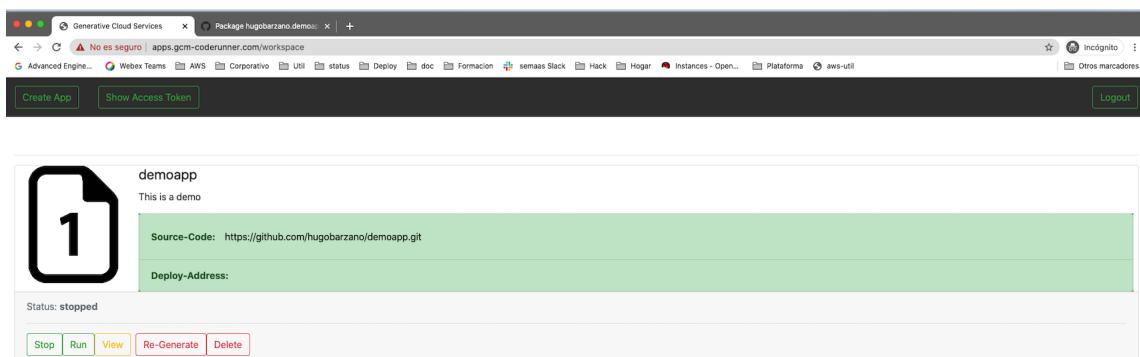


Figura 4.53: Trabajo Local: Single-Page Stop 0.0.1

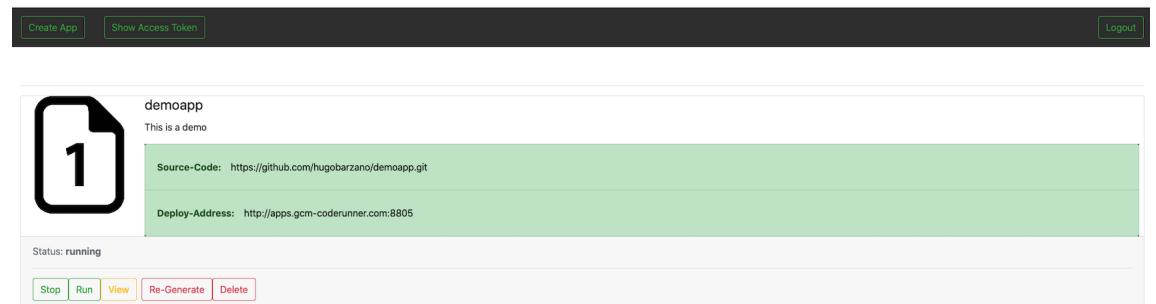


Figura 4.54: Trabajo Local: Single-Page Run 0.0.2

Si ahora vuelve a consumir la aplicación en la dirección de despliegue proporcionada, percibe que el cambio local ha sido aplicado y desplegado en forma de nueva versión

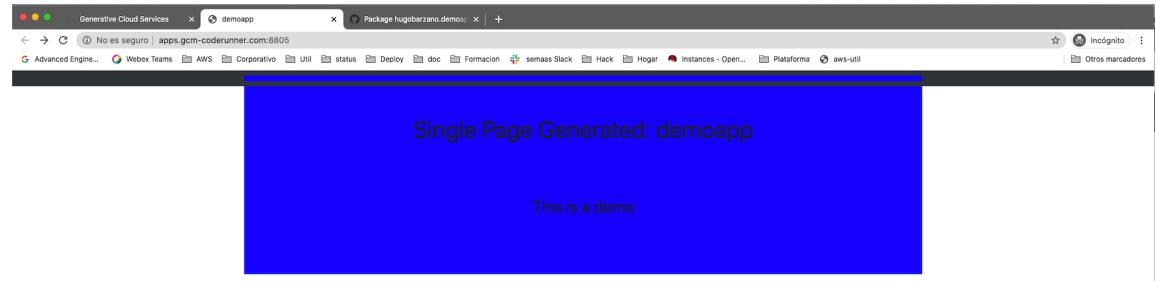


Figura 4.55: Trabajo Local: Single-Page Consulta 0.0.2

Re-Generación

Si por algún motivo, el usuario decide que quiere volver al punto de partida, el sistema proporciona la posibilidad de re-generar la aplicación en cuestión llevando el código fuente modificado a su estado base inicial. Para ello, con el foco en su **Workspace**, el usuario puede hacer click en el botón **Re-Generate**

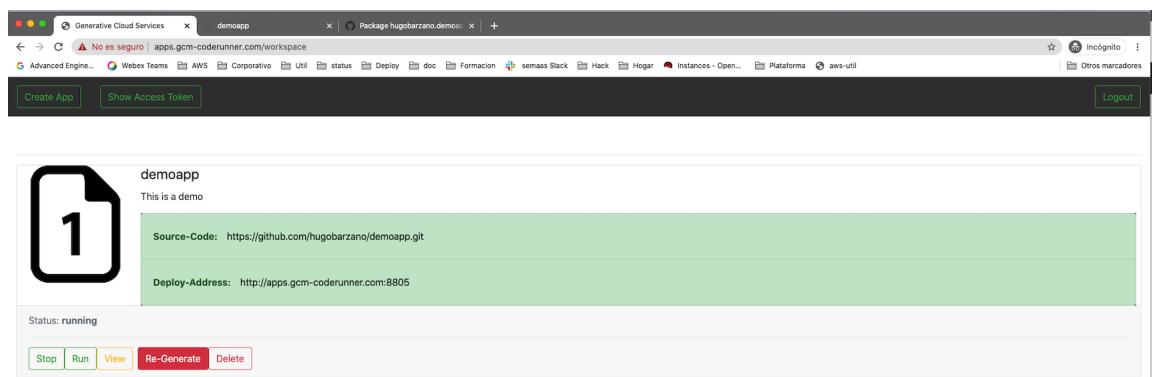


Figura 4.56: Re-Generación: Single-Page 0.0.2

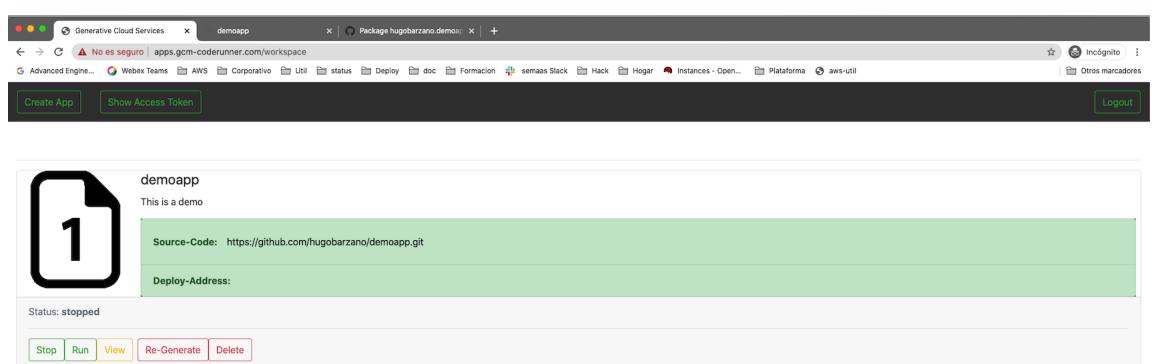


Figura 4.57: Re-Generación: Single-Page 0.0.2 Stop

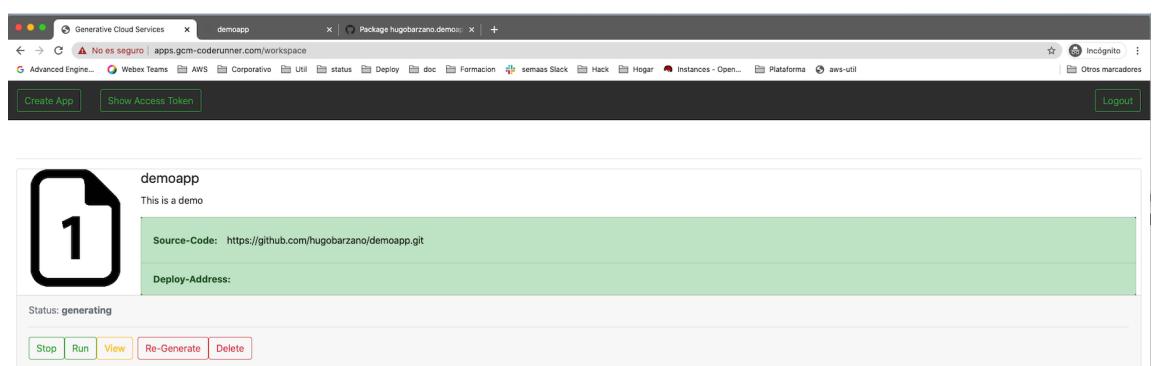


Figura 4.58: Re-Generación: Single-Page Generating

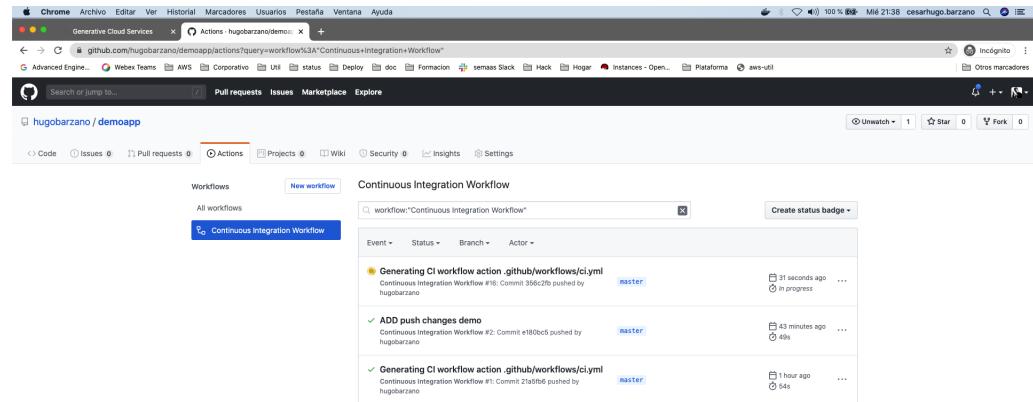


Figura 4.59: Re-Generación: Single-Page Building

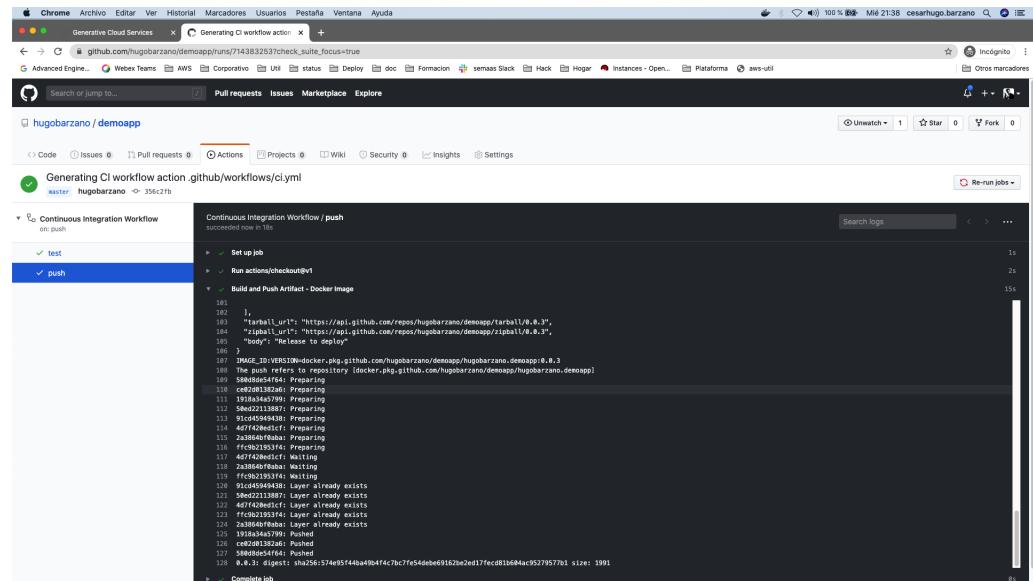


Figura 4.60: Re-Generación: Single-Page Ready

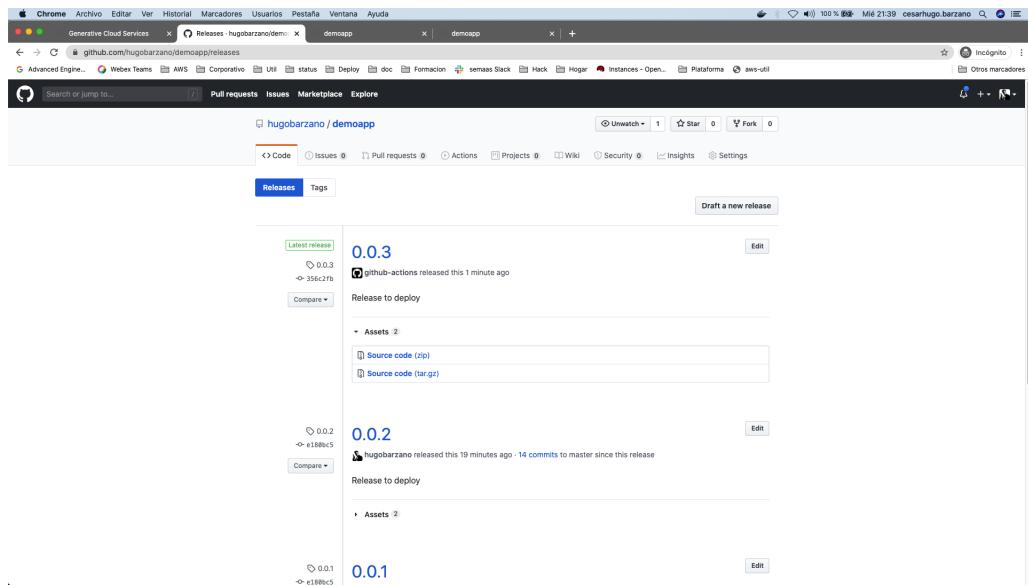


Figura 4.61: Re-Generación: Single-Page Release 0.0.3

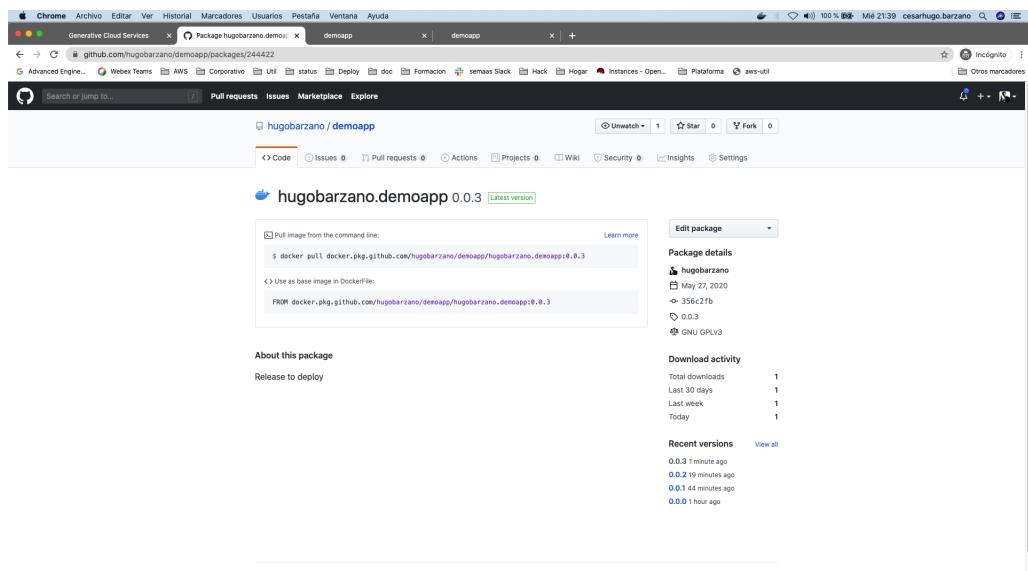


Figura 4.62: Re-Generación: Single-Page Artefacto 0.0.3

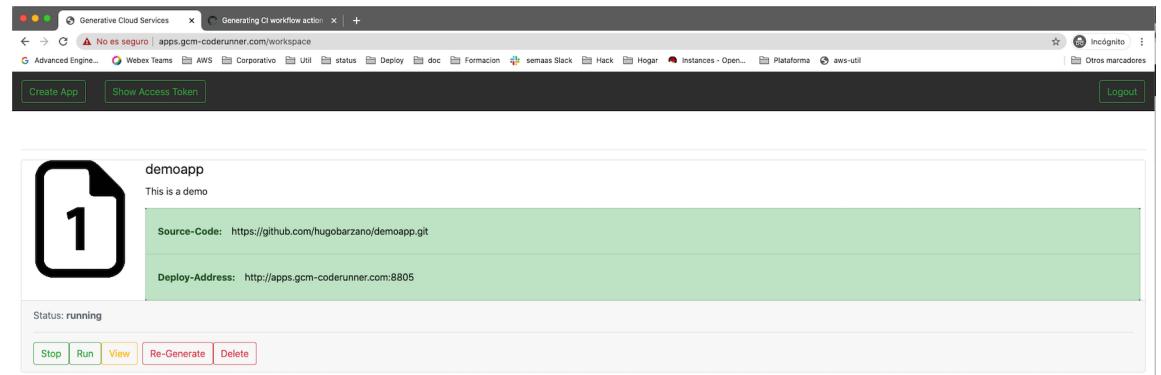


Figura 4.63: Re-Generación: Single-Page 0.0.3 Running

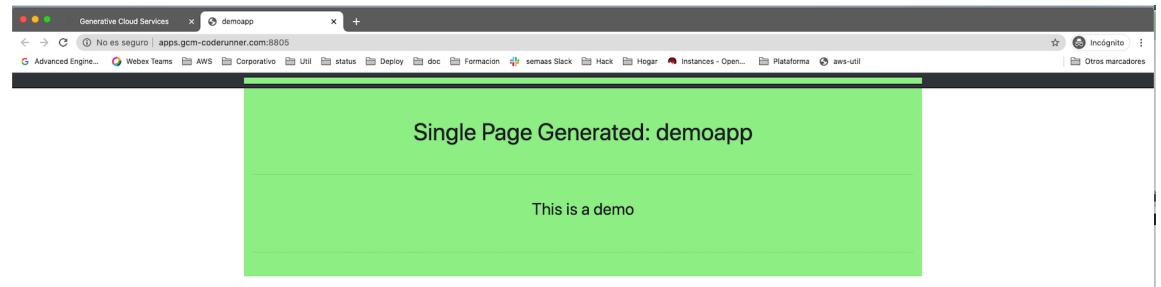


Figura 4.64: Re-Generación: Single-Page 0.0.3 Consulta

Limpieza

Si el usuario considera finalizado su trabajo puede eliminar todos los recursos virtuales creados por el sistema haciendo click en el botón **Delete**

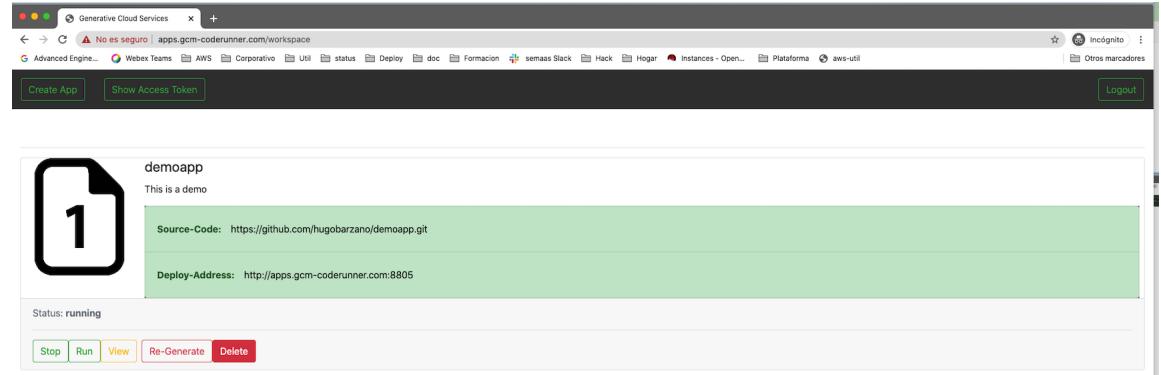


Figura 4.65: Limpieza: Single-Page 0.0.3 Delete

La aplicación entra en estado de no retorno **deleting** y se procede a la limpieza de los recursos virtuales utilizados.

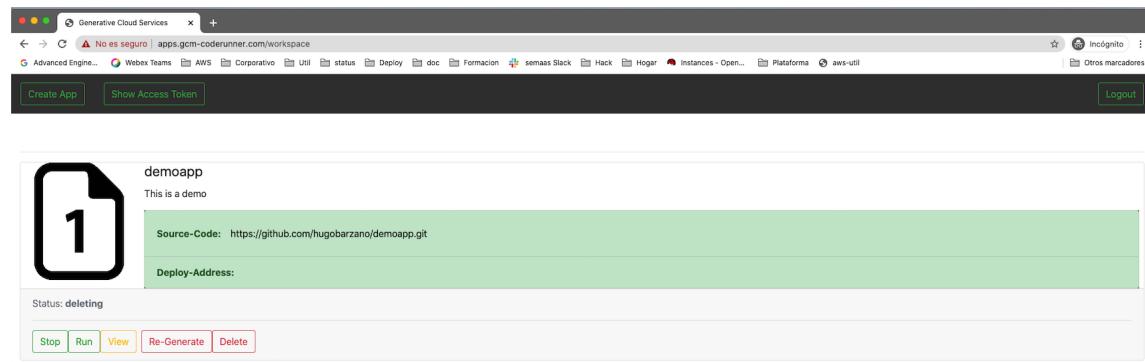


Figura 4.66: Limpieza: Single-Page 0.0.3 Deleting



Figura 4.67: Limpieza: Workspace

Capítulo 5

Conclusiones

Se ha demostrado cómo el uso de servicios cloud y la programación generativa son capaces de producir un sistema generativo que ofrece al usuario final un entorno virtual capaz de interpretar especificaciones de aplicación, generar su código fuente y crear los recursos necesarios para que sean consumidas por Internet. Este proyecto pone de manifiesto que la integración de buenas prácticas en los procesos relativos al ciclo de vida del desarrollo software mejora aspectos tales como la rápida iteración e integración de nuevas funcionalidades. Por otra parte, también se beneficia la entrega constante de nuevas versiones facilitando la evolución del software junto con mejoras en el tiempo de implantación fruto del despliegue continuo. El proporcionar un sistema cloud como éste, favorece el aprendizaje a los desarrolladores ya que disponen de un conjunto de aplicaciones de prueba en cuestión de segundos.

El proceso de desarrollo software mejora integrando buenas prácticas y automatizando las tareas recurrentes, la gran barrera a superar viene impuesta normalmente por los actuales procesos de las organizaciones, ya que actualizar su modelo de trabajo conlleva la inversión de una gran cantidad de recursos (económicos y humanos).

5.1. Mejoras y Trabajos Futuros

En esta sección, se proponen ciertas extensiones del proyecto que mejoran sus aspectos principales y dan paso a futuras ramas de estudio.

5.1.1. Proceso Generativo

Actualmente, el motor de plantillas encargado de generar las aplicaciones está completamente integrado en el código fuente de Code-Runner. Se propone como mejora la externalización de dichas plantillas

a un repositorio de código ajeno al proyecto. De esta manera, se podría trabajar en nuevas versiones de las aplicaciones generadas sin necesidad de liberar y desplegar una nueva versión del sistema software en si. Como punto de partida para la mejora se propone, por ejemplo, que Code-Runner sea capaz de cargar en memoria el contenido del repositorio de plantillas y utilizar dicho contenido para la producción de código. Adicionalmente, este mecanismo debería refrescar periódicamente la copia en memoria del repositorio. De esta forma, el sistema sería capaz de integrar nuevas versiones a generar sin perdida de servicio.

Especificación de Aplicación

Se propone como mejora adicional la actualización de la especificación de aplicación solicitada al usuario para cubrir escenarios mas complejos, como por ejemplo para la generación de aplicaciones Api-Rest podrían definirse diversos modelos de datos junto con las relaciones u operaciones que existan entre ellos. Los servicios de datos también puede mejorar si la especificación soportara configuración de credenciales(contraseñas).

Importar Aplicaciones

Se propone como mejora la capacidad de importar aplicaciones ya existentes en repositorios del usuario. La idea es la de inferir desde el repositorio de código lo necesario para categorizar la aplicación y poder generar los elementos software (configuración, scripts, código fuente, documentación) necesarios para que la aplicación forme parte del ecosistema de Code-Runner.

5.1.2. Naturaleza y Tecnología

Se ha demostrado que el uso de Docker como motor de virtualización ligera permite generalizar el despliegue y operación de aplicaciones de distinta naturaleza y de distinta tecnología. Se propone como mejora categorizar nuevas naturalezas de aplicación, así como integrar aplicaciones basadas en otras tecnologías. Para ello se presenta como referencia el conjunto de imágenes Docker disponibles públicamente en Docker Hub: [29] <https://hub.docker.com/search?q=&type=image>

Definir el Concepto Proyecto Generativo

La idea sería definir un nuevo modelo de datos dentro del proyecto con idea de representar a un conjunto reducido de aplicaciones de

diversa naturaleza y tecnología que funcionen en conjunto para formar una solución generativa. Por ejemplo, generar una Api-Rest y su servicio de persistencia como una sola entidad que englobe ambas aplicaciones. Para modelarlo se propone hacer uso de orquestadores de contenedores como Kubernetes [26] o Docker-Compose <https://docs.docker.com/compose>.

5.1.3. Testing

En el proceso de desarrollo software, el testing o validación es una etapa fundamental para garantizar el correcto funcionamiento de las piezas desarrolladas así como la integración de nuevas funcionalidades. Aunque en el ciclo de integración continua definido para las aplicaciones generadas por el sistema se contempla una etapa de ejecución de pruebas unitarias, no se está generando el código fuente necesario para materializarlas. Por ello, se propone como trabajo futuro la definición y generación de pruebas software para las aplicaciones generadas en función de su naturaleza. Las pruebas pueden ser, en primer lugar, de carácter unitario para ser ejecutadas durante el ciclo de integración continua y en segundo lugar, de carácter sistema o pruebas de caja negra para ser ejecutadas cada vez que se despliega una nueva versión.

5.1.4. Material Didáctico

Con el objetivo de mejorar el aprendizaje del usuario en el desarrollo, uso y consumo de aplicaciones web o servicios cloud, se propone como trabajo futuro la definición de material didáctico, documentación exhaustiva, ejemplos y retos que acompañen al código fuente generado. De esta forma, el sistema proporcionaría al usuario una línea de trabajo o actividades a realizar con el software que ha decidido generar, enriqueciendo y facilitando su proceso de aprendizaje.

5.1.5. Interfaz de Sistema Cloud

El estado actual del Code-Runner produce aplicaciones consumibles tal que <https://code-runner.com:8081/api/> donde probar la solución generada. Es decir:



Figura 5.1: Ejemplo App

La extensión que se propone sería la de incluir, en la etapa de generación de código, todo lo necesario para servir una interfaz interna o de sistema de manera común a todas las soluciones generadas, sirviendo cierta información sobre la aplicación:

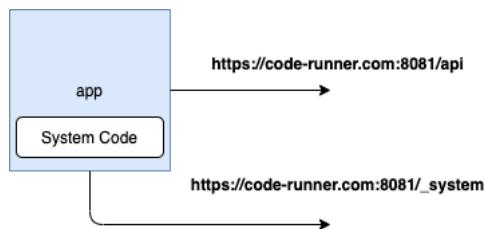


Figura 5.2: Ejemplo Interfaz Sistema

la interfaz `_system/` podría servir información como por ejemplo:

1. **`https://code-runner.com:8081/_system/version`** : Versión software
2. **`https://code-runner.com:8081/_system/config`** : Configuración de la Aplicación
3. **`https://code-runner.com:8081/_system/metrics`** : Métricas

La idea es aprovechar en la medida de lo posible esta interfaz para obtener información de las soluciones generadas y servir esta información desde la interfaz de usuario de Code-Runner junto con las obtenidas del contenedor donde se está ejecutando. Incluso las aplicaciones con esta interfaz podrían consumir recursos comunes al dominio cloud, mostrando un comportamiento de plataforma o de sistema cloud.

Capítulo 6

Anexo

6.1. Ficheros de Ejemplo

Modelo de Datos: Aplicación

```
1 {
2     "_id" : "example",
3     "repo" : "https://github.com/hugobarzano/example.git",
4     "spec" : {
5         "tech" : "python",
6         "modelJson" : "
7             {"active": true, "name": "thing name", "value": 33}\r\n
8             ",
9         "dockerId" : "aeccdfa4cf6f0bc0ede31977edc791ed87e0e
10            467b1dfd9222005e63feff8598",
11         "port" : "8083",
12         "nature" : "Api-Rest"
13     },
14     "des" : "this is a description",
15     "url" : "http://35.233.23.148:8083/api",
16     "owner" : "hugobarzano",
17     "status" : "running"
18 }
```

Flujo de Integración Continua

```
1 name: Continuous Integration Workflow
2 on:
```

```
3 push:
4   branches:
5     - master
6 jobs:
7   test:
8     runs-on: ubuntu-latest
9     steps:
10    - uses: actions/checkout@v1
11    - name: Run tests
12      run: |
13        make setup
14        make test
15 push:
16   needs: test
17   runs-on: ubuntu-latest
18   if: github.event_name == 'push'
19   steps:
20     - uses: actions/checkout@v1
21     - name: Build and Push Artifact - Docker Image
22       run: |
23         make push user=${{ github.actor }} token=${{ secrets.GITHUB_TOKEN }}
```

Bibliografía

- [1] C. H. Bárzano Cruz, Diagrama elaboración propia.
URL <https://www.draw.io>
- [2] S. Hykes, Docker 0.9: Introducing execution drivers and libcontainer, [Web].
URL <https://www.docker.com>
- [3] M. Fowler, M. Foemmel, Continuous integration, <http://www.martinfowler.com/articles/continuousintegration.html> (2005).
- [4] C. Lianping, Continuous delivery: Overcoming adoption challenges, Journal of Systems and Software 128 (2017).
doi:https://doi.org/10.1016/j.jss.2017.02.013.
URL <http://www.sciencedirect.com/science/article/pii/S0164121217300353>
- [5] A. Proulx, F. Raymond, B. Roy, F. Petrillo, Problems and solutions of continuous deployment: A systematic review, 2018.
URL <https://martinfowler.com/articles/continuousIntegration.html>
- [6] M. Fowler, J. Highsmith, The agile manifesto 9 (11 2000).
- [7] T. Dingsøyr, S. Nerur, V. Balijepally, N. B. Moe, A decade of agile methodologies: Towards explaining agile software development, Journal of Systems and Software 85 (6) (2012) 1213 – 1221, special Issue: Agile Development.
doi:https://doi.org/10.1016/j.jss.2012.02.033.
URL <http://www.sciencedirect.com/science/article/pii/S0164121212000532>
- [8] G. Github®, Github: Getting started.
URL <https://github.com>

- [9] D. C. Docker, Documentación - aislamiento de recursos.
URL <https://www.docker.com/what-docker>
- [10] K. Czarnecki, K. Østerbye, M. Völter, Generative programming., 2002, pp. 15–29.
- [11] K. Czarnecki, U. Eisenecker, R. Glück, D. Vandevenne, T. Veldhui-zen, Generative programming and active libraries, Vol. 1766, 2000. doi:[10.1007/3-540-39953-4_3](https://doi.org/10.1007/3-540-39953-4_3).
- [12] J. Herrington, Code Generation in Action, Manning Publications; Edición: 1-Simul and Rev, 2003.
- [13] J. Arnoldus, M. Brand, A. Serebrenik, Brunekreef, Code generation with templates, Rheologica Acta - RHEOL ACTA (01 2012). doi:[10.2991/978-94-91216-56-5](https://doi.org/10.2991/978-94-91216-56-5).
- [14] C. H. Bárzano Cruz, Apigena: Generación de apis como linea de productos software (2020).
URL <https://github.com/hugobarzano/apigena/blob/master/doc/doc.pdf>
- [15] R. Academia de Ingeniería, Computación en la nube.
URL <http://diccionario.raing.es/es/lema/computacion-en-la-nube>
- [16] M. Armbrust, F. Armando, G. Rean, D. J. Anthony, R. Katz, A. Kon-winski, A view of cloud computing, Communications of the ACM, 53 (4) (2010). doi:<https://doi.org/10.1145/1721654.1721672>.
- [17] Google, Google cloud platform: Proveedor nube pública.
URL <https://cloud.google.com>
- [18] Amazon, Amazon web services: Proveedor nube pública.
URL <https://aws.amazon.com>
- [19] Microsoft, Microsoft azure: Proveedor nube pública.
URL <https://azure.microsoft.com>
- [20] O. Ahmed, L. Haji, R. Zebari, S. Abass, M. Sadeeq, Comparison among cloud technologies and cloud performance, Journal of Applied Science and Technology Trends 1 (04 2020). doi:[10.38094/jastt1219](https://doi.org/10.38094/jastt1219).

- [21] M. Azure, SaaS software como servicio.
URL <https://azure.microsoft.com/es-es/overview/what-is-saas>
- [22] CLOUDFLARE, What is platform-as-a-service?
URL <https://www.cloudflare.com/learning/serverless/glossary/platform-as-a-service-paas>
- [23] M. A. Microsoft, IaaS infraestructura como servicio.
URL <https://azure.microsoft.com/es-es/overview/what-is-iaas>
- [24] G. Github®, Conociendo github.
URL <http://conociedogithub.readthedocs.io/en/latest/data/introduccion/#que-es-github>
- [25] S. Heroku 2020, Heroku: Platform as a service to build, run, and operate applications in the cloud.
URL <https://www.heroku.com/>
- [26] I. Copyright ©2020 Red Hat, Kubernetes: Implementación de contenedores.
URL <https://www.redhat.com/es/topics/containers/what-is-kubernetes>
- [27] G. C. P. Google, Cloud code: Escribir, depurar y ejecutar aplicaciones de kubernetes.
URL <https://cloud.google.com/code?hl=es-419>
- [28] G. ©TRAVIS CI, Test and deploy with confidence.
URL <https://travis-ci.org>
- [29] D. I. Docker, Docker hub: Create, manage, and deliver container.
URL <https://hub.docker.com/>
- [30] A. W. S. Amazon, Aws codedeploy: Automatización de implementaciones de código.
URL <https://aws.amazon.com/es/codedeploy>
- [31] A. W. S. Amazon, Aws codepipeline: Automatización de pipelines e integración continua.
URL <https://aws.amazon.com/es/codepipeline>
- [32] C. H. Bárzano Cruz, Diagrama gant planificación.
URL <https://www.tomsplanner.es>

- [33] M. Fowler, Gui architectures model view controller.
URL <https://martinfowler.com/eaaDev/uiArchs.html#ModelViewController>
- [34] A. Bonventre, Go 1.13 Release Notes,
<https://golang.org/doc/copyright.html>,
<https://golang.org/doc/go1.13>, 3 September 2019.
- [35] K. Chodorow, Mongo Database: The Definitive Guide, O'Reilly Media, 1449344682, 2013.
- [36] G. Github®, Documentación de github actions.
URL <https://help.github.com/es/actions>
- [37] G. Github®, Documentación de github packages.
URL <https://github.com/features/packages>
- [38] j. The jQuery Foundation, Documentación de jquery, write less, do more.
URL <https://jquery.com>
- [39] J. Eguiluz, Documentación de ajax: Introducción a ajax.
URL <https://librosweb.es/libro/ajax>
- [40] B. Bootstrap Team, Documentación de bootstrap.
URL <https://getbootstrap.com>