



Máster Universitario de Investigación en Ingeniería De Software Y Sistemas Informáticos

31105151 - Trabajo Fin de Máster

Autor

César Hugo Bárzano Cruz

Tutor

Rubén Heradio

Generative Cloud Manager: code-runner

—

2019/2020

Máster Universitario de Investigación en Ingeniería De Software Y Sistemas Informáticos

**31105151 - Trabajo Fin de Máster Generative Cloud Manager:
code-runner**

Consultar el tipo de trabajo: Trabajo Propuesto por Alumno

Autor

César Hugo Bárzano Cruz

Tutor

Rubén Heradio



—
2019/2020

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 desprestigiar a los procesos de desarrollo software tradicionales pero si mostrar 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. **Artefact 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 Artefact 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.

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 provide 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.

Agradecimientos

– INTEGRAR ALGO BONITO :) –

Índice general

Índice de figuras	9
Bibliografía	11
1. Introducción	15
1.1. Motivación	15
1.2. Objetivos	15
1.3. Problema	16
1.4. Solución	18
2. Trabajos Relacionados	21
2.1. Cloud Computing	21
2.1.1. Tipos de Nube	22
2.1.2. Tipos de Servicios en la Nube	23
2.2. Programación Generativa	26
2.3. Desarrollo Software	27
2.3.1. Ciclo de Vida del Desarrollo Software (SDLC)	27
2.3.2. Desarrollo Software y Buenas Prácticas	27
2.3.3. En este Proyecto	29
3. Planificación y Análisis	31
3.1. Planificación	31
3.2. Análisis Software	31
3.2.1. Requisitos de Información	31
3.2.2. Requisitos Funcionales	32
3.2.3. Requisitos No Funcionales	34
3.2.4. Casos de Uso	35
3.2.5. Diagramas de Secuencia	36
4. Implementación y Pruebas	41
4.1. Implementación	41
4.2. Desarrollo	41
4.2.1. Back-End	42

4.2.2. Front-end	47
4.2.3. Infraestructura Virtual	47
4.2.4. Seguridad	47
4.3. Caso de Uso	47
5. Conclusiones	49
5.1. Mejoras	49
5.2. Trabajos Futuros	49
6. Anexo	51

Índice de figuras

2.1. Beneficios Cloud Computing[1]	22
2.2. Arquitectura Cloud Computing[2]	24
3.1. Casos de uso - Gestión Aplicaciones I[3]	35
3.2. Casos de uso - Gestión Aplicaciones II[3]	35
3.3. Casos de uso - Gestión Credenciales[3]	36
3.4. Diagramas de Secuencia - Inicio sesión [3]	36
3.5. Diagramas de Secuencia - Crear Aplicación[3]	37
3.6. Diagramas de Secuencia - Ejecutar, Inspeccionar y Detener Aplicación[3]	38
3.7. Diagramas de Secuencia - Eliminar Aplicación[3]	39
4.1. Modelo-Vista-Controlador[4]	41
4.2. Arquitectura MongoDB[5]	44
4.3. Virtualización Docker[6]	46

Bibliografía

- [1] M. José Villamizar Cano, Imagen de cloud computing hoy.
URL <https://es.slideshare.net/mariojosevillamizarcano/cloud-computing-hoy-todo-como-servicio>
- [2] D. I. de tecnología de la Universidad de Deusto., Imagen arquitectura cloud computing.
URL <http://image.slidesharecdn.com/tocandolanubev2-141007052808-conversion-gate01/95/cloud-computing-una-perspectiva-tecnologica-9-638.jpg?cb=1412659848>
- [3] C. H. B. Cruz, Diagrama elaboración propia - herramienta online draw.io.
URL <https://www.draw.io>
- [4] M. Diagrama, Modelo – vista – controlador (mvc).
URL https://yalantis.com/media/content/ckeditor/2014/05/mvc_role_diagram.png
- [5] www.mongodb.com, An enterprise architect's view of mongodb.
URL <http://image.slidesharecdn.com/enterprisearchitectsview2014-oct-141029130130-conversion-gate01/95/an-enterprise-architects-view-of-mongodb-16-638.jpg?cb=1414587854>
- [6] S. HYKES, Docker 0.9: introducing execution drivers and libcontainer.
URL [https://www.docker.com/blog/docker-0-9-introducing-execution-drivers-and-lib](https://www.docker.com/blog/docker-0-9-introducing-execution-drivers-and-libcontainer)
- [7] R. A. de Ingeniería, Computación en la nube.
URL <http://diccionario.raing.es/es/lema/computacion-en-la-nube>
- [8] M. Goossens, F. Mittelbach, A. Samarin, The L^AT_EX Companion, Addison-Wesley, Reading, Massachusetts, 1993.

- [9] R. G. A. D. J. R. K. A. K. Michael Armbrust, Armando Fox, A view of cloud computing, *Communications of the ACM*, 53 (4) (2010) 50–58. doi:<https://dl.acm.org/doi/10.1145/1721654.1721672>.
- [10] Google, Google cloud platform: Crea, moderniza y escala tu negocio.
URL <https://cloud.google.com>
- [11] Amazon, Amazon web services: Plataforma cloud pública.
URL <https://aws.amazon.com>
- [12] M. Azure, Microsoft azure: Convierta las ideas en soluciones con más de 100 servicios para crear, implementar y administrar aplicaciones.
URL <https://azure.microsoft.com>
- [13] M. Azure, ¿qué es saas? software como servicio.
URL <https://azure.microsoft.com/es-es/overview/what-is-saas>
- [14] CLOUDFLARE, What is platform-as-a-service?
URL <https://www.cloudflare.com/learning/serverless/glossary/platform-as-a-service-paas>
- [15] M. Azure.
- [16] K. Czarnecki, K. Østerbye, M. Völter, *Generative programming.*, 2002, pp. 15–29.
- [17] K. Czarnecki, U. Eisenecker, R. Glück, D. Vandevoorde, T. Veldhuizen, *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).
- [18] M. Fowler, *Continuous integration*, 2006.
URL <https://martinfowler.com/articles/continuousIntegration.html>
- [19] L. Chen, 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>
- [20] F. Potencier, *El tutorial jobeet. capítulo 4 – mvc: Patrón de arquitectura de software.*
URL http://librosweb.es/libro/jobeeet_1_4/capitulo_4/la_arquitectura_mvc.html

- [21] M. Fowler, Gui architectures – model view controller.
URL <https://martinfowler.com/eaDev/uiArchs.html#ModelViewController>
- [22] A. Bonventre, Go 1.13 Release Notes,
<https://golang.org/doc/copyright.html>,
<https://golang.org/doc/go1.13>, 3 September 2019.
- [23] K. Chodorow, MongoDB: The Definitive Guide, O'Reilly Media, 1449344682, 2013.
- [24] <https://github.com/>, Getting started with github.
URL <https://help.github.com/en/github/getting-started-with-github>
- [25] <https://github.com/>, Conociendo github.
URL <http://conociendogithub.readthedocs.io/en/latest/data/introduccion/#que-es-github>
- [26] <https://github.com/>, Documentación de github actions.
URL <https://help.github.com/es/actions>
- [27] <https://github.com/>, Documentación de github packages.
URL <https://github.com/features/packages>
- [28] D. Community, Docker - aislamiento de recursos.
URL <https://www.docker.com/what-docker>
- [29] T. jQuery Foundation, jquery, write less, do more.
URL <https://jquery.com>
- [30] J. Eguiluz, Introducción a ajax.
URL <https://librosweb.es/libro/ajax>

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 al 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.

- **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.

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, ¿Como 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 consumir.

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 esta plataformas son:

- Linux Academy
- Coursera
- Uдеми

Estas plataformas cuentan con diversos recursos virtuales para el aprendizaje (videos, 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 en los proyectos informáticos 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, ante la 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 (make build, make run, make test, make push, make pull)

- 4.1. build
 - 4.2. test
 - 4.3. run
 - 4.4. pull
 - 4.5. push
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. **Dockerfile.** Define al artefacto necesario para ejecutar la aplicación. Dicho artefacto es una Imagen docker.

Por otra parte, 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. **Artefact 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 Artefact 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 específico 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.

En los siguientes capítulos se mostraran trabajos, publicaciones y tecnologías relacionadas con el campo de estudio aquí propuesto. Con

el objetivo de facilitar la comprensión del sistema software a construir, se planifica y analiza el problema a resolver mediante ingeniería de requisitos. Tras el análisis se detalla la implementación realizada y la consecución de los objetivos propuesto mediante un caso de uso funcional.

Capítulo 2

Trabajos Relacionados

En las siguientes secciones se presenta a modo de introducción conceptos relativos a Programación Generativa, Cloud Computing y conceptos relacionados con el proceso de desarrollo software y el ciclo de vida de productos software, conceptos como la Integración Continua y el Despliegue Continuo. Se pretende con esto dotar al lector de un contexto o conocimiento básico para abordar el proyecto. De manera adicional se abordan tecnologías, trabajos y publicaciones relacionadas para mostrar el estado actual del arte.

2.1. Cloud Computing

La real academia de la ingeniería, define la computación en la nube o Cloud Computing[7] como: “La Utilización de las instalaciones propias de un servidor web albergadas por un proveedor de Internet[8] 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[9] 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, el usuario

puede acceder a estos servicios mediante Internet. 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. La siguiente imagen muestra los beneficios de la computación en la nube:

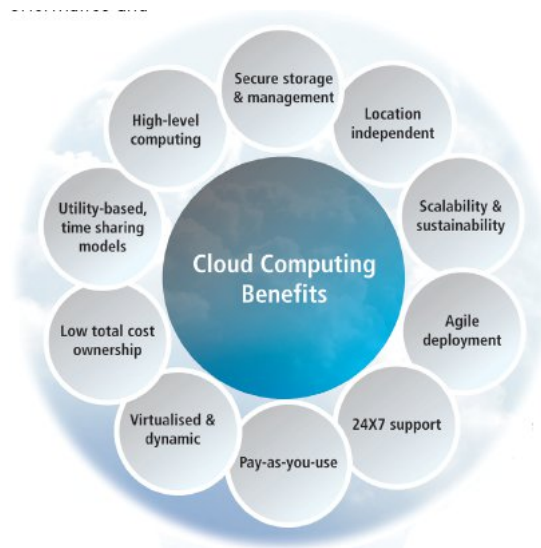


Figura 2.1: Beneficios Cloud Computing[1]

2.1.1. Tipos de Nube

Las nube puede clasificarse en distintos modelos en función del tipo de solución que quiera ofertarse:

Nube Pública

Las nubes públicas son propiedad de los proveedores de servicio, es decir, hardware, 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[10] , Amazon Web Services[11] o Microsoft Azure[12]

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.

Nube Híbrida

Las nubes híbridas se caracterizan por combinar los modelos de nube pública y nube privada. Las 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.

2.1.2. Tipos de Servicios en la Nube

La computación en la nube permite la integración de servicios en Red dando servicio a nivel mundial y además contribuye al uso eficiente de la energía pero 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. Los servicios que ofertados por la nube pueden clasificarse en las siguientes subsecciones:

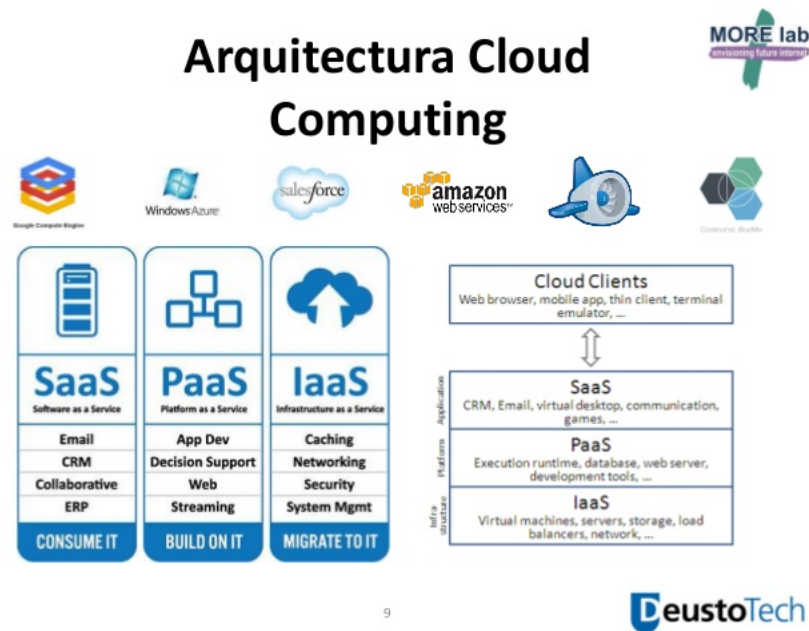


Figura 2.2: Arquitectura Cloud Computing[2]

Software como servicio (SaaS)

El Software como Servicio[13] (SaaS, Software as a Service) 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 (PaaS)

La Plataforma como Servicio[14] (PaaS, Platform as a Service) es una categoría de servicios cloud que proporciona un entorno a los desarrolladores 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. Los servicios PaaS consisten en funcionalidades pre-configuradas a las que los clientes puedan suscribirse, eligiendo las funciones que deseen incluir 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 (IaaS)

La Infraestructura como Servicio[15] (IaaS, Infrastructure as a Service) 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. Sin necesidad de invertir en hardware.
3. Independencia de la localización.
4. Seguridad física en los centros de datos.

2.2. Programación Generativa

Tal y como se presenta en el artículo Generative Programming[16] 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.

Para alcanzar estos objetivos, la programación generativa ha de cumplir los siguientes principios tratados en el artículo Generative Programming and Active Libraries [17]

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.

2.3. Desarrollo Software

2.3.1. Ciclo de Vida del Desarrollo Software (SDLC)

El ciclo de vida del desarrollo software 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 transformar 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 disponibiliza 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.

2.3.2. Desarrollo Software y Buenas Prácticas

En esta sección se profundiza en buenas practicas 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**[18] , 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 , 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**[19], 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**[?] 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 mas a cada uno de los problemas presentados.

2.3.3. En este Proyecto

Este proyecto ha sido desarrollando siguiendo con mayor o menor rigos las etapas del ciclo de vida del desarrollo software. El sistema software sintetiza el proceso de desarrollo y la operación software en una serie de etapas automatizadas o pipeline, agilizando las producción de aplicaciones modernas.

En línea con las buenas practicas a la hora de desarrollar software presentadas en la sección anterior, este proyecto las implementa todas ellas.

Como se verá en la siguiente sección relativa a la programación generativa, como parte del proceso generativo implementado para crea aplicaciones, se incluye la creación de un repositorio de código fuente donde albergar el código generado. El sistema de gestión de repositorios de código fuente elegido par ello ha sido github debido a su caracter En el artículo **Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management**[18]

Capítulo 3

Planificación y Análisis

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.

– TODO – INCLUIR DIAGRAMAS GANT

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 subsecciones 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, 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:

```
1 {
2   "_id" : "appName",
3   "repo" : "https://github.com/user/appName.git",
4   "spec" : {
5     "dockerId" : "appNameDockerId",
6     "port" : "4343",
7     "nature" : "staticApp"
8   },
9   "des" : "",
10  "url" : "http://deploy.domain:4343",
11  "owner" : "user>s",
12  "status" : "running"
13 }
```

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 autenticuen 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 de 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.

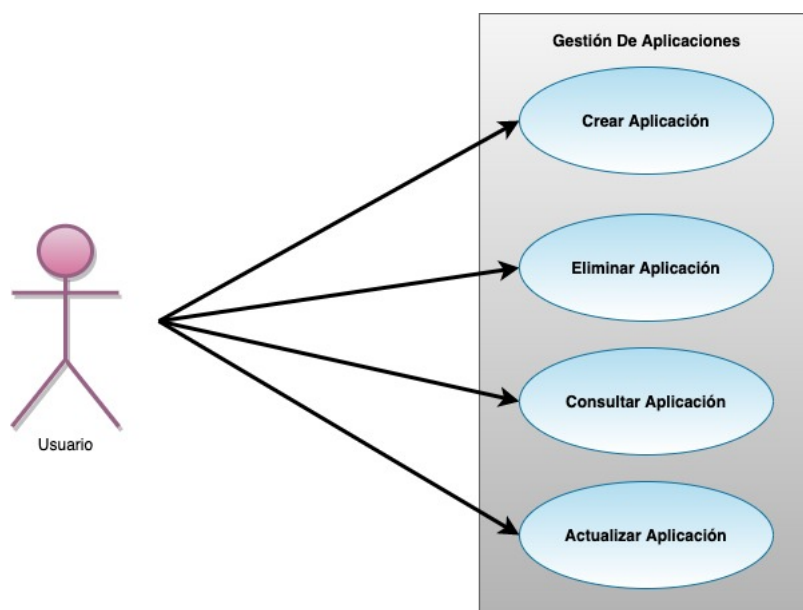


Figura 3.1: Casos de uso - Gestión Aplicaciones I[3]

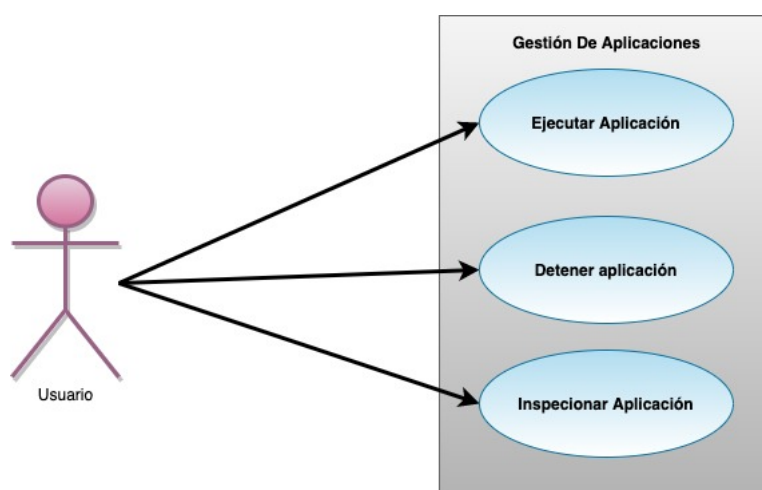


Figura 3.2: Casos de uso - Gestión Aplicaciones II[3]

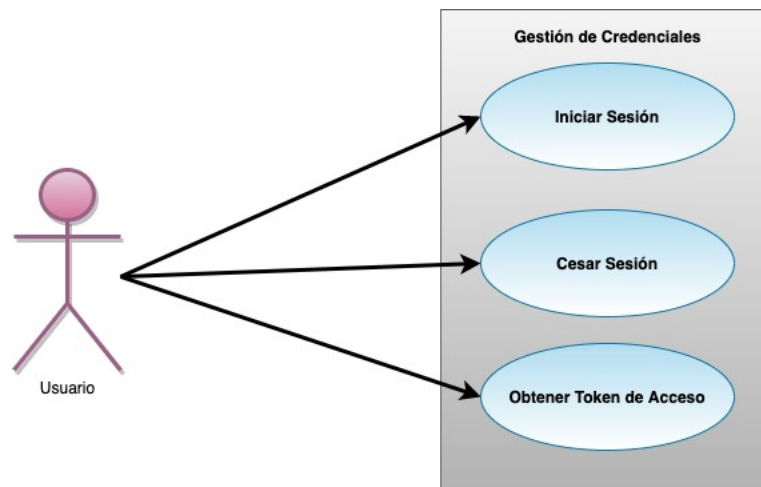


Figura 3.3: Casos de uso - Gestión Credenciales[3]

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.

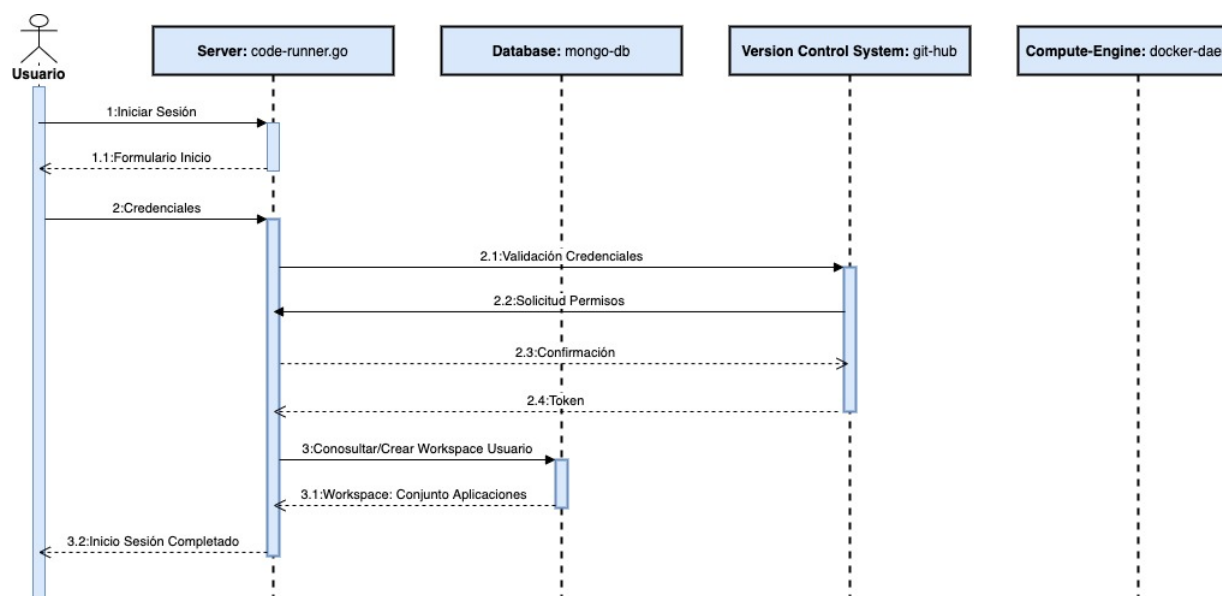


Figura 3.4: Diagramas de Secuencia - Inicio sesión [3]

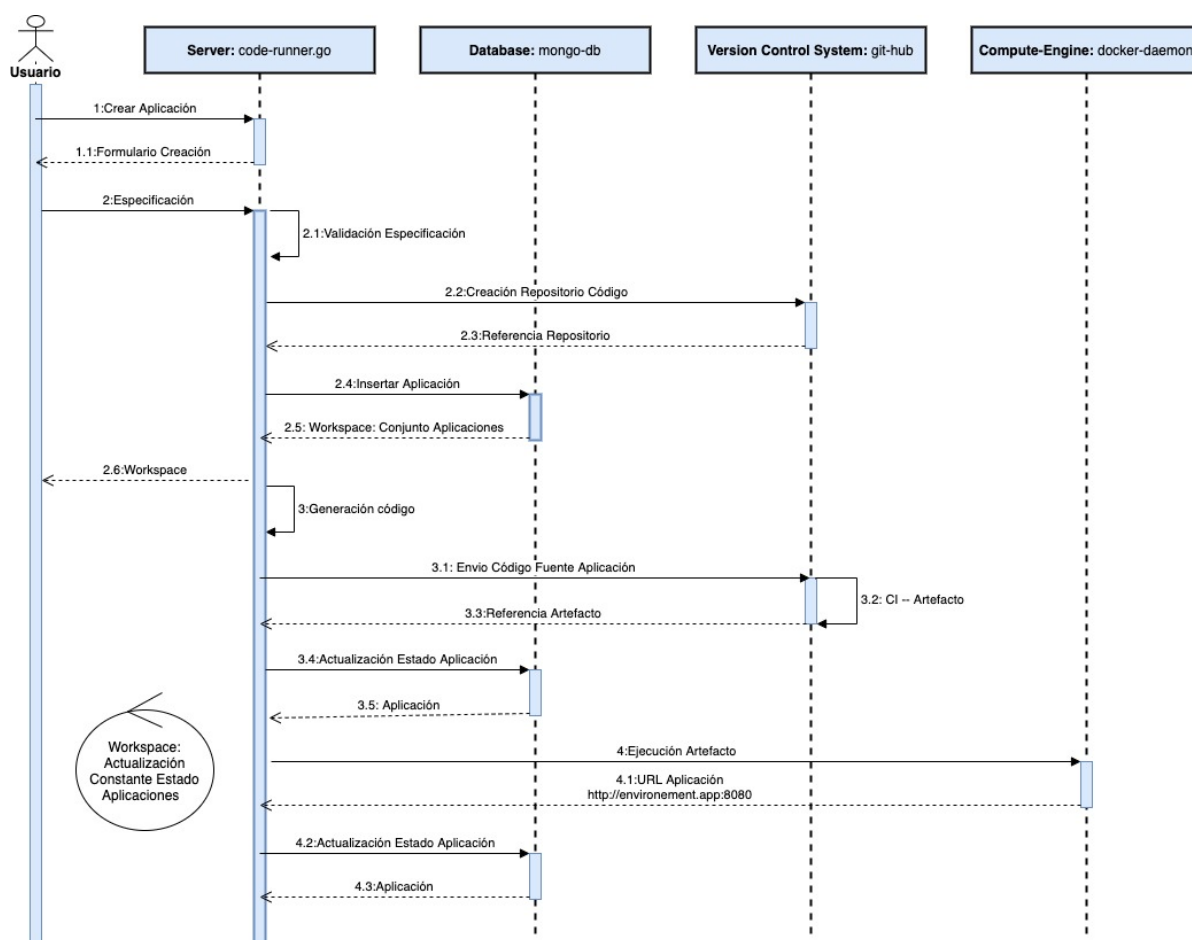


Figura 3.5: Diagramas de Secuencia - Crear Aplicación[3]

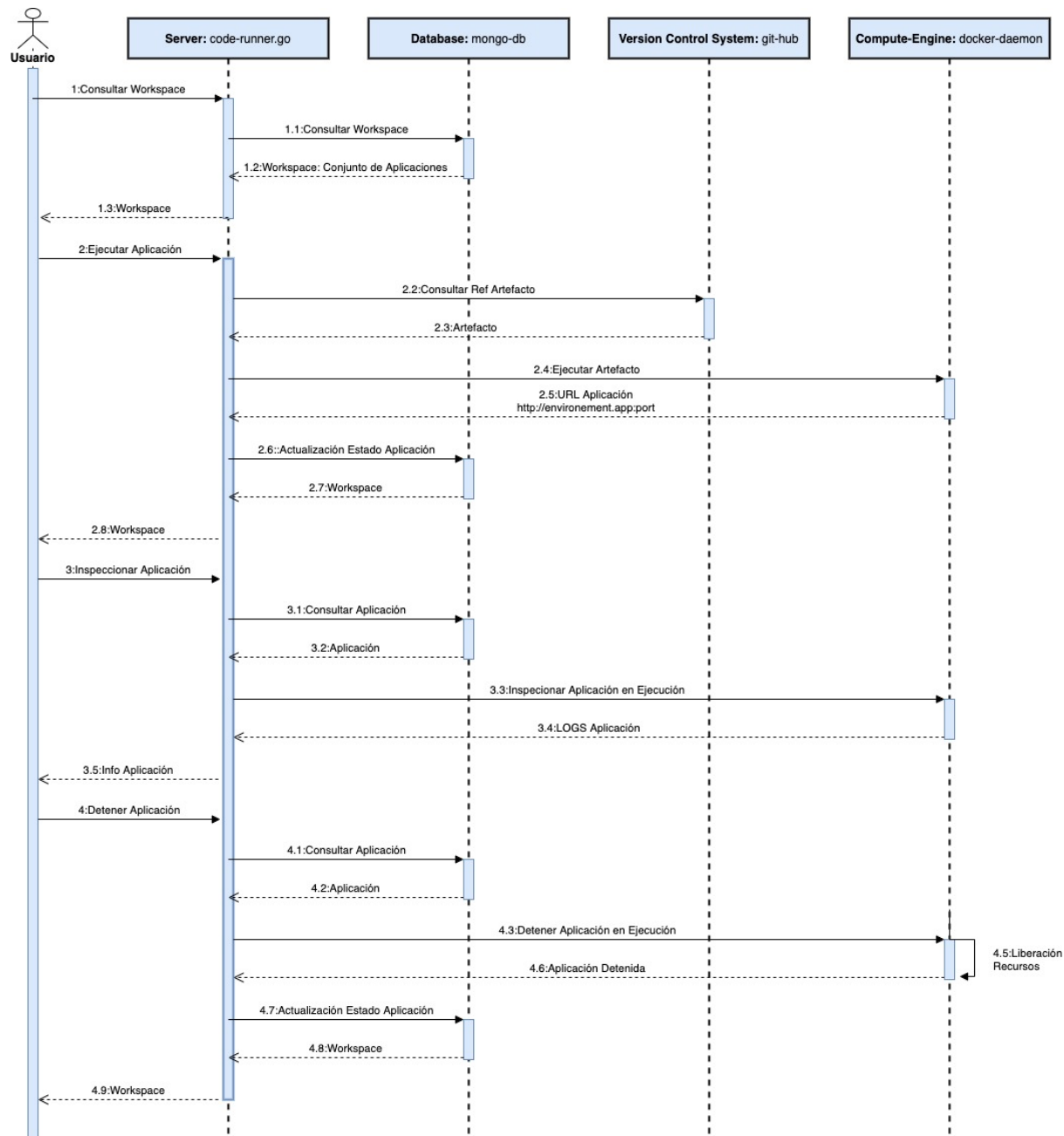


Figura 3.6: Diagramas de Secuencia - Ejecutar, Inspeccionar y Detener Aplicación[3]

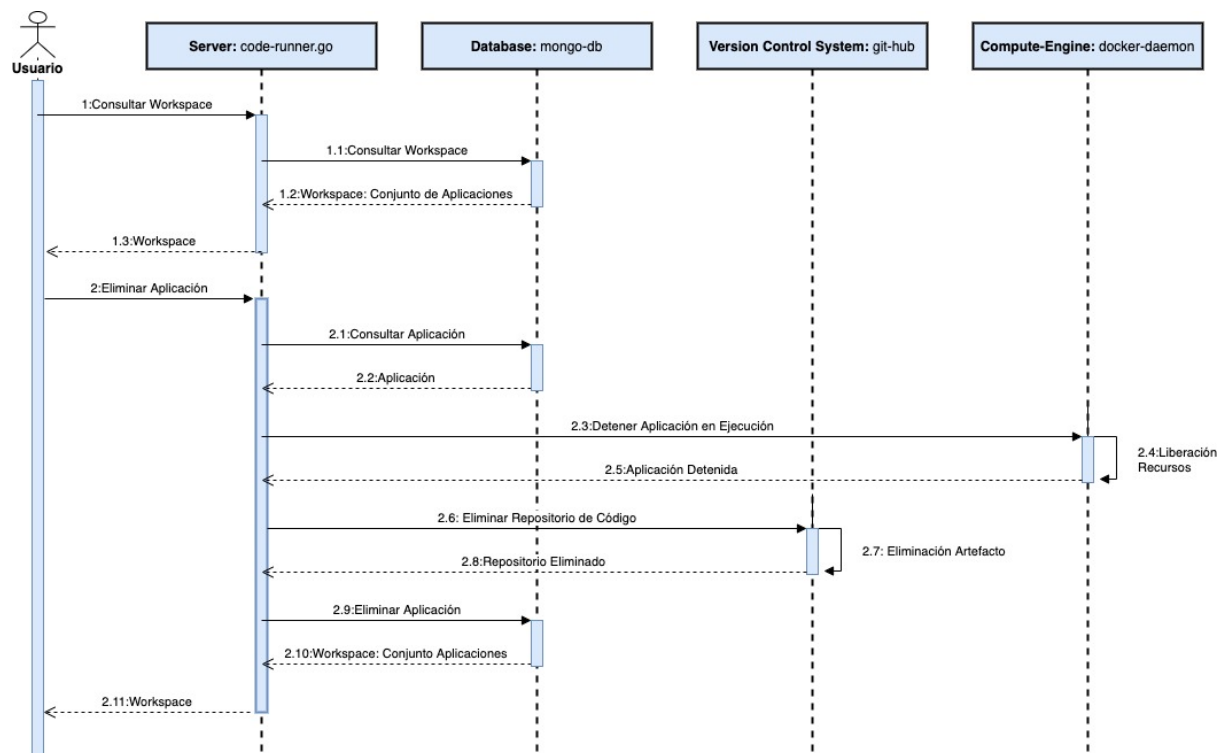


Figura 3.7: Diagramas de Secuencia - Eliminar Aplicación[3]

Capítulo 4

Implementación y Pruebas

4.1. Implementación

4.2. Desarrollo

En el proceso de desarrollo del sistema, se ha seguido el patrón de arquitectura software modelo-vista-controlador (MVC)[20] 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.

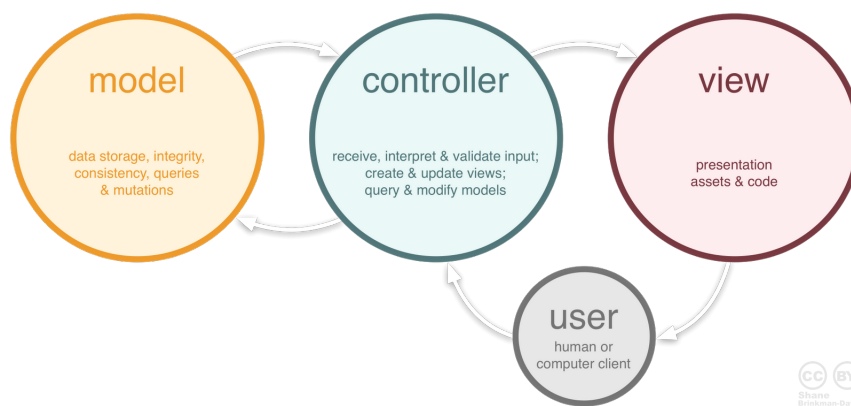


Figura 4.1: Modelo-Vista-Controlador[4]

Los componentes[21] 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 una 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 test o 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 esta poniendo a prueba. En las siguientes subsecciones se detallarán las partes que forman el sistema así como las tecnologías utilizadas para su desarrollo.

4.2.1. 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 subsecciones.

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 partes del back-end y de servir o renderizar las vistas al cliente. El servidor ha sido desarrollado utilizando Golang[22] como lenguaje de programación. Inicialmente se pensó en utilizar Javascript bajo el framework NodeJS, 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 NodeJS 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 templates o plantillas. 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-Routinas, es decir, la capacidad de ejecutar funcionalidad en segundo plano de una manera sencilla y controlada junto con la generalización de la implementación mediante interfaces 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.

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 entidad-relación.

La principal desventaja de modelar el problema mediante entidad-relación es la pérdida 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 manejar se requiere que el modelo de datos sea flexible. Debido a esto, se ha decidido utilizar MongoDB[23] como sistema gestor de bases de datos no-relaciones también denominadas bases de datos documentales, con el objetivo de conseguir un esquema interno flexible y eficiente.

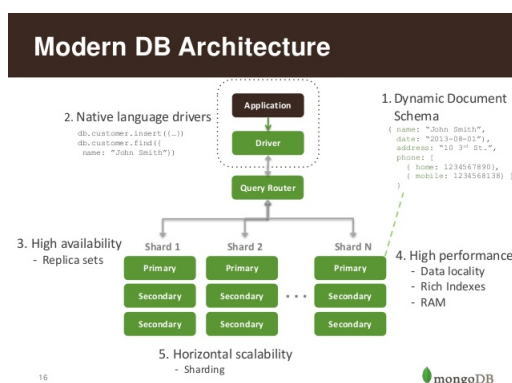


Figura 4.2: Arquitectura MongoDB[5]

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 esta 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 producidos por el sistema son almacenados en repositorios de código alojados en la nube de manera pública, para ello, este proyecto esta completamente integrado con Github[24].

GitHub[24] 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[25] 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.

Este proyecto aprovecha principalmente las siguientes funcionalidades ofertadas por Github:

1. **Autenticación.** El usuario final que utilice este sistema para generar aplicaciones necesita una cuenta de github, ya que la autenticación mediante user/pass corresponde con el user/pass de su cuenta de github. Esto permite al sistema crear repositorios y pushear código a dichos repositorios bajo el perfil del usuario registrado. El proceso de autenticación contra Github proporciona una accessToken el cual es aprovechado por los usuarios del sistema para trabajar localmente con las aplicaciones generadas por el sistema.
2. **Integración Continua.** Github proporciona las denominadas GitHub Actions[26] que permiten automatizar, customizar 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 el path <repositorio>/.github/workflows/ci.yml donde se generaliza el proceso 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. Los artefactos generados son Imágenes Docker las cuales se verán en mas detalle en la siguiente sub-sección.
3. **Gestión de Artefactos** Github proporciona los denominados GitHub Packages[27] que permiten publicar y consumir paquetes o artefactos 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 la gestión de aplicaciones se ha decidido utilizar Docker como tecnología de virtualización ligera.

Docker[28] 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 Linux, para permitir que contenedores independientes se ejecuten dentro de una sola máquina, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

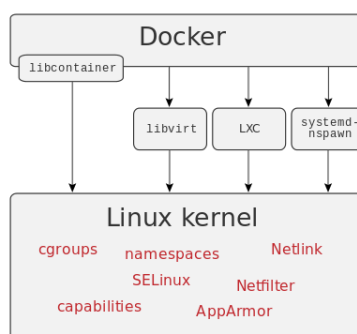


Figura 4.3: Virtualización Docker[6]

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 contienen lo mismo pero comparten el núcleo con otros contenedores corriendo como un proceso asilado en el espacio de usuario del sistema operativo anfitrión. No están vinculados con ninguna infraestructura específica.

Docker aporta a este proyecto la capacidad de abstraer las aplicaciones 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 continua definido en (incluir referencia a ci.yml) para producir una imagen docke como artefacto. Este artefacto es utilizado para crear un contenedor docker en un entorno virtual accesible a internet, habilitando el puerto necesario para

garantizar que el tráfico es redireccionado al contenedor adecuado.

4.2.2. 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 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 GO junto con JQuery y Ajax para realizar llamadas asíncronas al servidor y actualizar dinámicamente los datos que percibe el cliente.

Tecnología Cliente

JQuery[29] 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[30], 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.

Socket -> Logs aplicaciones en tiempo real.

4.2.3. Infraestructura Virtual

Heroku (plataforma as a service) para ejecutar code-runner.go + Maquina expuesta a internet en google cloud con un demonio docker expuesto por el puerto 5555 comunicada con el servidor mediante TLS.

4.2.4. Seguridad

mutual tls

4.3. Caso de Uso

Capítulo 5

Conclusiones

5.1. Mejoras

5.2. Trabajos Futuros

Capítulo 6

Anexo