



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 muestra como el uso de la programación generativa y las tecnologías de la información mejoran el proceso aprendizaje y desarrollo 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 cloud de diversa naturaleza partiendo de una especificación de alto nivel. Normalmente, las etapas de desarrollo software están segmentadas en el desarrollo del código, la compilación de este código junto con las dependencias de terceros necesarias para producir un ejecutable, binario o artefacto para su posterior ejecución, la configuración necesaria para poder ejecutar el artefacto resultante así como la infraestructura virtual necesaria para desplegar y soportar la actividad de negocio de dicho software, siendo este accesible desde Internet.

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 automatización genérica de las etapas básicas de producción software pueden dar lugar a soluciones funcionales ahorrando al desarrollador numerosas horas de trabajo. Otro enfoque igual de valido es el de dotar al desarrollador inexperto de un punto de partida en su aprendizaje.

Este proyecto pone de manifiesto que independientemente del tipo de aplicación web o servicio cloud que el desarrollador quiera producir existen unas etapas básicas, comunes y transversal por las que la gran mayoría aplicaciones han de pasar comenzando por la implementación del código fuente hasta alcanzar la madurez necesaria para ser accesibles a través de Internet. En este proyecto dichas etapas se sintetizan de la siguiente manera:

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 así como se entrará en detalle de cada una de estas etapas.

Índice general

1. Introducción	9
1.1. Motivación	9
1.2. Objetivos	9
2. Descripción del Problema	11
2.1. El Problema	11
2.2. Generadores de aplicaciones	11
2.3. Despliegue de código	11
3. Descripción de la Solución	13
3.1. Planificación	14
3.2. Análisis	14
3.2.1. Requisitos de Información	14
3.2.2. Requisitos Funcionales	15
3.2.3. Requisitos No Funcionales	17
3.2.4. Casos de Uso	17
3.3. Desarrollo	19
3.3.1. Back-End	20
3.3.2. Front-end	25
3.3.3. Infraestructura Virtual	25
4. Implementación y Pruebas	27
5. Conclusiones	29
6. Anexo	33
6.1. Plantilla Corrección Tipo Test	33

Índice de figuras

3.1. Casos de uso - Gestión Aplicaciones I[?]	18
3.2. Casos de uso - Gestión Aplicaciones II[?]	18
3.3. Casos de uso - Gestión Credenciales[?]	19
3.4. Modelo-Vista-Controlador[]	19
3.5. Arquitectura MongoDB[]	22
3.6. Virtualización Docker[56]	24

Capítulo 1

Introducción

1.1. Motivación

La motivación de este proyecto es la de mejorar el proceso de desarrollo de aplicaciones web y servicios cloud, ayudando al desarrollador a comprender y adquirir conocimientos relativos a las tecnologías de la información. En el proceso de desarrollo de software informático no existe un proceso universal que indique las pautas a seguir para la correcta producción software. Comúnmente, este proceso ha de amoldarse a la naturaleza del software que se desea desarrollar, a la tecnología con la que ha de desarrollarse y como no a los recursos humanos o desarrolladores que van a construir dicho software. La motivación de este proyecto es la de dotar a los desarrolladores de un sistema que los abstraiga de estas etapas, premiando la rápida producción y disponibilidad del software deseado.

1.2. Objetivos

El objetivo de este proyecto es el de sintetizar el proceso de desarrollo y operación de software en un conjunto automatizado de etapas. 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. La generalización ocurre al manejar los distintos sistemas de información:

1. **Aplicaciones web estáticas**
2. **Aplicaciones web dinámicas**
3. **Interfaces web**

4. **Api/Rest**
5. **Servicios**
6. **Cualquier pieza software que pueda ser consumida mediante protocolo tcp/http**

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 y/o configuración necesaria para materializar dicha especificación en un producto software, construir un artefacto auto-contenido que permita la ejecución del software generado, ejecutar dicho artefacto sobre la infraestructura virtual necesaria para que la aplicación especificada sea accesible por Internet y que de esta forma el usuario de este proyecto sea capaz de consumir la aplicación generada.

El objetivo de este proyecto es el de dotar a los usuarios o desarrolladores de un sistema generativo y automatizado de aplicaciones que mejore el proceso de desarrollo software y facilite el aprendizaje de nuevas tecnologías. El principal objetivo del proyecto es acuñar el lema "From Spec to Cloud.^{es} decir, dada una especificación de aplicación de alto nivel, materializarla en un sistema de información disponible en la nube.

OBJ-1.

OBJ-2.

Capítulo 2

Descripción del Problema

2.1. El Problema

El auge de las tecnologías de la información a nivel empresarial, social, ético...blacbla

La cloud esta formada por un conjunto de tecnologías que habilitan al consumidor una serie de servicios de información para ser consumidos vía Internet. Cada vez son mas y mas las empresas y entidades confían en este tipo de servicios para realizar sus actividades económicas. La diversidad de soluciones demandadas ha creado un ecosistema en el que interactúan diversos sistemas y diversas tecnologías. Debido a esto la demanda de personal cualificado o desarrolladores también se ha incrementado.

Hoy en día la sociedad esta rodeada de

Diversidad de aplicaciones cloud. Nuevas tecnologías entorno variado y cambiante. Demanda de nuevas tecnologías. Capacitación para desarrolladores. Diversidad de aplicaciones, mismo tratamiento, simplificar proceso de aprendizaje y desarrollo software. FullStack.

2.2. Generadores de aplicaciones

2.3. Despliegue de código

Capítulo 3

Descripción de la Solución

Con el objetivo de generalizar y definir aplicación software se han decidido usar las partes:

Repositorio Readme Makefile Licencia src código fuente config configuración

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.

Programación generativa: generar, automatizar y despliegue accesible a Internet.

La aplicación software es el elemento de negocio de este sistema. Para materializar esto, el sistema crea un repositorio de código por cada aplicación creada para albergar la producción de código especificado por el desarrollador.

Generar: Aplicaciones completas, independientemente de la tecnología, utilidad o fin. Las aplicaciones generadas por el sistema son auto-contenidas y automatizadas, es decir mantienen una semántica común
d test run pull push

Lo que permite al desarrollador integrar fácilmente nuevos cambios.

Definir el término producción dentro del marco de este sistema, consiste en la siguiente serie de etapas:

3.1. Planificación

3.2. Análisis

Como se ha visto, el termino producción en el termino de este sistema considera

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 comportamiento 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 **local-host** 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 usuarios 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 soportará la autenticación de usuarios mediante nombre de usuario y contraseña. Típicamente estas credenciales corresponderán al sistema de gestión de repositorios de código.

RF-2.2. El sistema realizará todas las operaciones que requieran autenticación mediante TOKEN.

RF-2.3. 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.

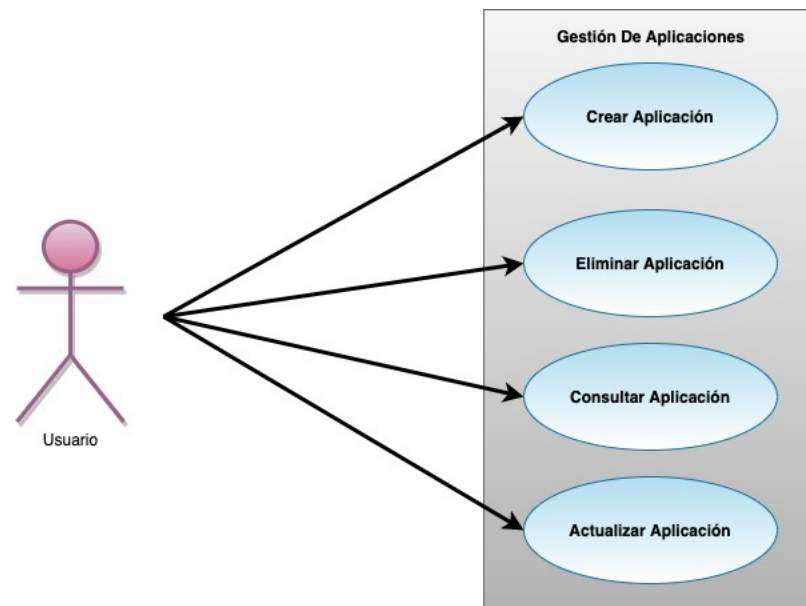


Figura 3.1: Casos de uso - Gestión Aplicaciones I[?]

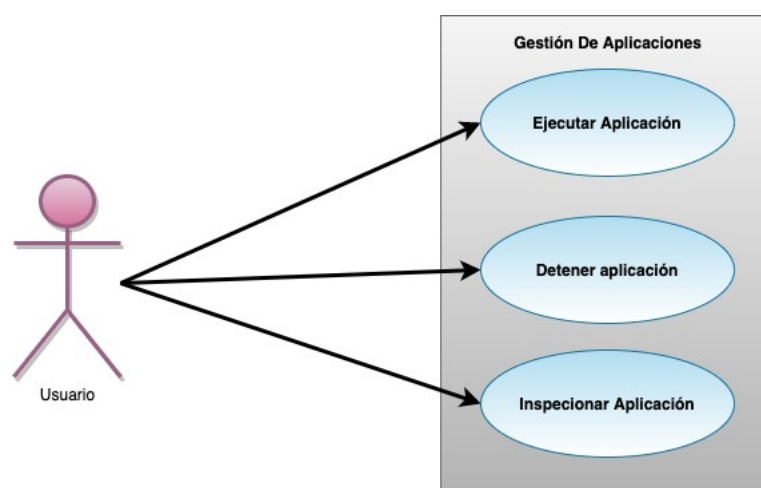


Figura 3.2: Casos de uso - Gestión Aplicaciones II[?]

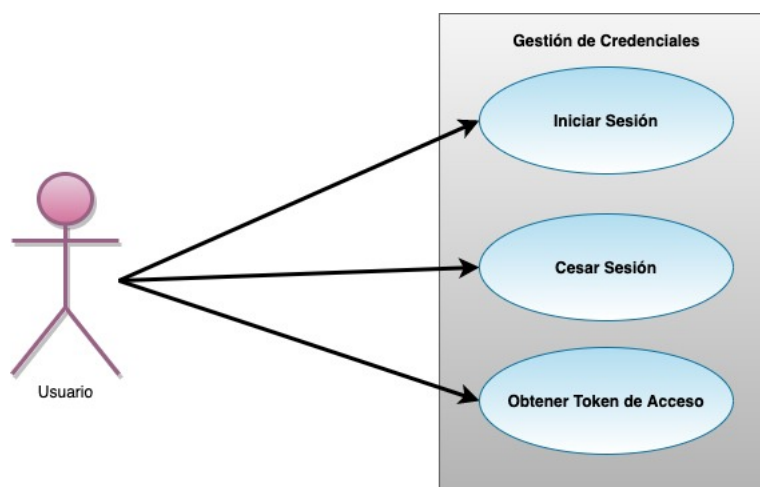


Figura 3.3: Casos de uso - Gestión Credenciales[?]

3.3. Desarrollo

En el proceso de desarrollo del sistema, se ha seguido el patrón de arquitectura software modelo-vista-controlador (MVC)[1] 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 esca-labilidad del software resultante.

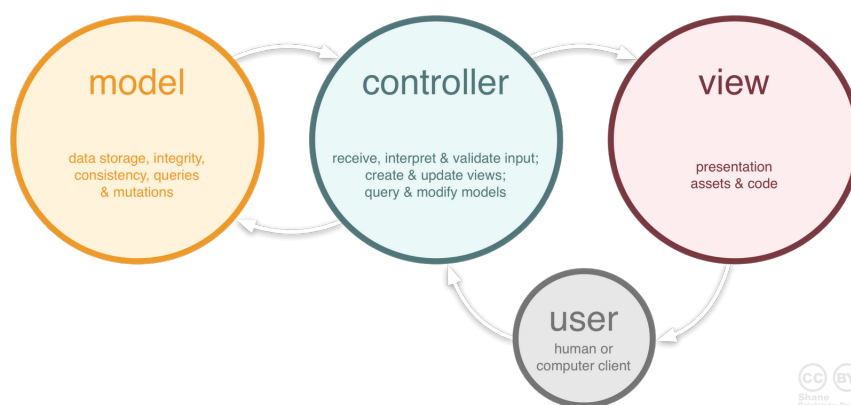


Figura 3.4: Modelo-Vista-Controlador[1]

Los componentes[1][2] 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.

3.3.1. 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[] 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[] 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.

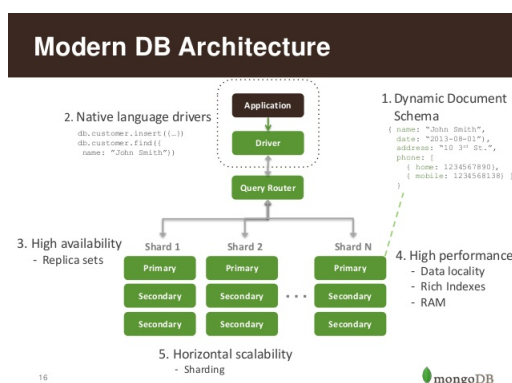


Figura 3.5: Arquitectura MongoDB[]

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[.].

GitHub[] 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[] 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[] 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 Generación 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[] 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[] 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.

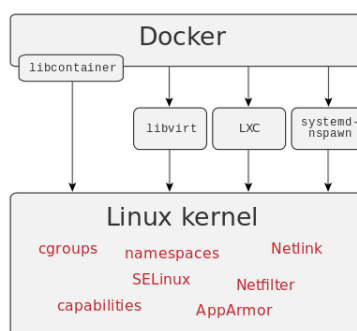


Figura 3.6: Virtualización Docker[56]

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 aislado 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 incluye un fichero 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 el una imagen docke como artefacto resultante. Este artefacto es utilizado para ccrear un contenedor

docker en un entorno virtual accesible a internet, habilitando el puerto necesario para garantizar que el tráfico es redireccionado al contenedor adecuado.

3.3.2. Front-end

El Front-End, capa de presentación o capa de cliente es la encargada de maquetar la estructura semántica del contenido, codificar el diseño en hojas de estilo y controlar la interacción con el usuario. En este proyecto la programación en el lado cliente juega un papel importante a la hora de mejorar la experiencia del usuario del sistema, manteniendo 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 los dinámicamente los datos que percibe el cliente.

Tecnología Cliente

JQuery[] 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[], 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

3.3.3. Infraestructura Virtual

Capítulo 4

Implementación y Pruebas

Capítulo 5

Conclusiones

Bibliografía

- [] MVC, *MVC: Patrón de arquitectura de software* http://librosweb.es/libro/jobeeet_1_4/capitulo_4/la_arquitectura_mvc.html
- [] MVC, *MVC: Diagrama* https://yalantis.com/media/content/ckeditor/2014/05/mvc_role_diagram.png
- [] MVC, *MVC: Componentes* <http://martinfowler.com/eaDev/uiArchs.html>
- [] MVC, *MVC Wikipedia* <https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador>
- [] Golang, *Golang* <https://golang.org>
- [] mongo, *MongoDB* <https://www.mongodb.com/>
- [] MongoDB, *Arquitectura MongoDB* <http://image.slidesharecdn.com/enterprisearchitectsview2014-oct-141029130130-conversion-gate-95/an-enterprise-architects-view-of-mongodb-16-638.jpg?cb=1414587854>
- [] GitHub, *How people build software* <https://github.com/>
- [] Conociendo GitHub, *readthedocs.io* <http://conociendogithub.readthedocs.io/en/latest/data/introduccion/#que-es-github>
- [] GitHub Actions, *Documentación de GitHub Actions* <https://help.github.com/es/actions>
- [] GitHub Packages, *Documentación de GitHub Packages* <https://github.com/features/packages>
- [] Docker, *Docker - Aislamiento de recursos* <https://www.docker.com/what-docker>

- [56] Docker-Wiki, *Docker - Wikipedia* https://es.wikipedia.org/wiki/Docker_%28software%29
- [] JQuery, *JQuery Oficial* <https://jquery.com/>
- [] Ajax, *Ajax: Libros Web* <https://librosweb.es/libro/ajax/>

Capítulo 6

Anexo

6.1. Plantilla Corrección Tipo Test