Universidad de las Ciencias Informáticas

Facultad 1


# “Nova Brouwer, servicio RESTful de administración centralizada de la distribución cubana GNU/Linux Nova”

**Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas**

**Autor:** Carlos Rubén García Pérez

**Tutor:** Ing. Nurisel Palma Pérez

Ing. Yosel Lázaro Vera González

**Habana, junio de 2018**

**“Cuando lo creas todo perdido,**
**no olvides que aún te queda**
**el futuro, tu cerebro, tu voluntad**
**y dos manos para cambiar tu destino”**

**Wernher von Braun**

## Dedicatoria

*Dedico el presente trabajo de diploma a mi familia, por su bendición y apoyo incondicional en cada decisión afrontada; y a los amigos, por esos ánimos que restauraron la voluntad para persistir en mis aspiraciones.*

**Agradecimientos**

A la Universidad de Ciencias Informáticas por la oportunidad de estudiar en tan prestigiosa institución y comprometerme en un entorno científico-profesional.

A los profesores docentes y especialistas del centro CESOL que brindaron sus conocimientos y dedicaron tiempo a mi formación profesional.

A los tutores del presente trabajo de diploma por su compromiso y guía durante toda la investigación.

A todos los colegas que se preocuparon por mi progreso en la etapa de estudiante y con los que compartí muy buenos momentos específicamente a mis paisanos de Pinar del Rio.

A los rockeros de Punto Ciego por los ratos divertidos que pasamos juntos y por esos sueños tan increíbles que compartimos, en especial al director general Ernesto por recordarme que la amistad no entiende de tiempos, ni de dificultades solo de más historias por vivir.

A mis compañeros de grupo por ayudarme a enfrentar cada reto y motivarme a superar las adversidades.

A la gente del BaKno (123), hogar de amistades, TITANES y algún que otro súper héroe simplemente por existir y por todos los recuerdos que tenemos en común, los tendré siempre presentes, GRACIAS por estos cinco años: Dibaniel, David, Yurien, Roda, Diwel, LLillo, VlaZ, Osvaldo, Ramón, AmaZ, Papi Raulí, Manu, Michel, Juanka y Mauro.

A mis amistades Xami, Ania, Elier, Dario y Pepe por los ánimos para persistir en mis aspiraciones y por sacar siempre la mejor versión de mí.

*A todos los miembros de mi familia y personas allegadas por su apoyo incondicional.*

*A las parejas de mis padres por siempre estar al tanto de mis preocupaciones y aceptarme como un hijo más.*

*A mis tíos y primos por su afecto y por hacer de cada momento familiar único, en especial a mi generación: Helmi, Roger, Daily, Koki y Azi.*

*A mi tío Carlos Javier y a mi brother Juanma por ser una constante motivación para continuar superándome profesionalmente.*

*A mis hermanitos por ser una razón más para esforzarme en ser mejor persona y un ejemplo a seguir para ellos, los quiero mucho.*

*A mis abuelos por estar dispuestos siempre a escucharme y por el amor cultivado en cada etapa de mi vida.*

*A mi padre por contar conmigo cada semestre vencido, por cada consejo dedicado, por cada abrazo de despedida que ayudaba a enfrentar la semana y sobre todo por nunca dejar de ser mi mejor amigo.*

*A mi madre por su constante educación, por no dejar nunca de insistir en mis estudios, por siempre mantenerme positivo, por cada llamada de preocupación, por su amor absoluto, y sobre todo por estar presente en cada paso que he dado en la vida y preparada para enfrentar juntos el próximo desafío.*

## Declaración Jurada de Autoría

Declaro por este medio que yo, Carlos Rubén García Pérez, con carné de identidad 93101900245, soy el autor principal del presente Trabajo de Diploma “Nova Brouwer, servicio *RESTful* de administración centralizada de la distribución cubana GNU/Linux Nova” y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste, firman la presente declaración jurada de autoría en La Habana a los ___ días del mes de _____ del año _____.

**Carlos Rubén García Pérez**

_____

Firma del autor

**Ing. Nurisel Palma Pérez**

_____

Firma de la tutora

**Ing. Yosel Lázaro Vera González**

_____

Firma del tutor

## Datos de Contacto

**Ing.** Nurisel Palma Pérez, graduada de Ingeniera en Ciencias Informáticas en el 2013 en la Universidad de las Ciencias Informáticas (UCI). Posee la categoría docente de Instructor. Forma parte del departamento Servicios Integrales en Migración, Asesoría y Sistemas (SIMAYS). Posee publicaciones y participación en eventos como el II Congreso Internacional de Ingeniería Informática y Sistemas de Información, la Conferencia Científica de la UCI en sus tres ediciones y el III Congreso Multidisciplinario de Ciencias Aplicadas en Latinoamérica. (npalma@uci.cu)

**Ing.** Yosel Lázaro Vera González, graduado de ingeniero en Ciencias Informáticas en el 2016 en la Universidad de las Ciencias Informáticas (UCI). Forma parte del departamento Servicios Integrales en Migración Asesoría y Sistemas (SIMAYS) donde se desempeña como jefe del proyecto Nova 360. Ha cursado diferentes cursos de postgrado y desempeñado el rol de especialista de migración en el Ministerio de las Comunicaciones (MIC), Banco Central de Cuba (BCC) y la Empresa Constructora de Obras de Arquitectura e Industriales número 3 (ECOAIND3). (ylvera@uci.cu)

**Resumen**

En la presente investigación se tiene como objetivo desarrollar un servicio *RESTful* para la administración centralizada de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova. Para su logro se documenta el estudio de soluciones informáticas y de sus correspondientes herramientas como base para fundamentar el desarrollo de un servicio informático que responda a las necesidades encontradas. El servicio basa su funcionamiento en una arquitectura cliente-servidor, especificando a *Python* como lenguaje de programación, *PyCharm* como entorno de desarrollo y *Django REST Framework* como *framework* de desarrollo. Se emplea la metodología de desarrollo AUP-UCI, y en el documento se especifican los resultados alcanzados en cada una de las iteraciones. Luego de realizada la implementación de las funcionalidades definidas, el producto de software final resuelve desplegar configuraciones predeterminadas y personalizadas de forma centralizada sobre las estaciones de trabajo que tengan instalada la distribución cubana GNU/Linux Nova, lo que facilita el trabajo llevado a cabo por los especialistas de migración.

**Palabras clave:** administración, centralizada, estación de trabajo, servicio web.

## Índice de Contenido

*Introducción* ..... *0*

*Capítulo 1: Proceso de administración centralizada de las estaciones de trabajo con el sistema operativo GNU/Linux* ..... *5*

**1.1 Conceptos fundamentales** ..... **5**

**1.2 Proceso de administración centralizada de estaciones de trabajo** ..... **5**

1.2.1 Proceso de administración centralizada de estaciones de trabajo en GNU/Linux Nova ..... 7

**1.3 Sistemas de administración de la configuración** ..... **7**

1.3.1 Desktop Central ..... 7
1.3.2 Sophos Central ..... 8
1.3.3 Landscape ..... 8
1.3.4 Chef Compliance ..... 9
1.3.5 CFEngine Enterprise ..... 9
1.3.6 Puppet Enterprise ..... 10
1.3.7 Saltstack Enterprise ..... 10
1.3.8 Ansible Tower ..... 10
1.3.9 Análisis de los sistemas de administración de la configuración ..... 11

**1.4 Herramientas de administración de la configuración** ..... **11**

1.4.1 Resumen de características ..... 14
1.4.2 Comparación entre las herramientas ..... 19
1.4.3 Aplicación de la metodología QSOS ..... 19

**1.5 Caracterización de Ansible** ..... **23**

1.5.1 Funcionamiento ..... 24
1.5.2 Playbooks ..... 26

**1.6 Servicio web RESTful** ..... **27**

**1.7 Lenguajes y herramientas de desarrollo** ..... **28**

**1.8 Metodología de desarrollo de software** ..... **30**

1.8.1 Metodología de desarrollo AUP-UCI ..... 31

**1.9 Consideraciones finales** ..... **31**

*Capítulo 2: Servicio RESTful para la administración centralizada de la distribución cubana GNU/Linux Nova* ..... *32*

**2.1 Propuesta de solución** ..... **32**

**2.2 Requisitos** ..... **33**

2.2.1 Fuentes de obtención de requisitos ..... 34
2.2.2 Técnicas para identificar requisitos ..... 34
2.2.3 Requisitos funcionales ..... 34
2.2.4 Requisitos no funcionales ..... 39
2.2.5 Validación de requisitos de software ..... 40

2.2.6 Historias de usuario (HU) ..... 40
**2.3 Análisis y diseño ..... 42**
2.3.1 Arquitectura del sistema ..... 42
2.3.2 Estilo arquitectónico para back-end ..... 43
2.3.3 Modelo de los datos ..... 44
2.3.4 Modelo de clases UML ..... 44
2.3.5 Patrones de diseño ..... 46
**2.4 Consideraciones finales ..... 47**
***Capítulo 3: Implementación y pruebas ..... 48***
**3.1 Implementación ..... 48**
3.1.1 Diagrama de componentes ..... 48
3.1.2 Diagrama de despliegue ..... 49
3.1.3 Estándares de codificación ..... 50
3.1.4 Estándares de nomenclatura ..... 51
**3.2 Pruebas de software ..... 51**
3.2.1 Pruebas unitarias ..... 52
3.2.2 Pruebas funcionales ..... 57
3.2.3 Pruebas de aceptación ..... 60
3.2.4 Evaluación del objetivo de la investigación ..... 61
**3.3 Consideraciones finales ..... 63**
***Conclusiones finales ..... 64***
***Recomendaciones ..... 65***
***Referencias Bibliográficas ..... 66***

**Introducción**

Durante las últimas décadas se han realizado notables progresos en el mercado del software a partir de tecnologías y procedimientos subordinados a automatizar los procesos que se desarrollan en las entidades. En este marco se destaca el desarrollo del software libre y de código abierto, que brinda a los usuarios las libertades de usar, estudiar, distribuir y mejorar cualquier distribución GNU/Linux. Con el fin de solventar necesidades específicas muchas organizaciones han optado por esta alternativa tecnológica que responde a menor costo de inversión, mayor seguridad y facilidad para automatizar procesos. El futuro tecnológico del país apuesta por el sistema operativo GNU/Linux como principal plataforma para llevar a cabo el programa de informatización de la sociedad cubana; desde el año 2004 se estableció el inicio del proceso migratorio hacia software libre, trazando así una ruta viable a la soberanía tecnológica [1].

Entre los motivos a favor de la migración que impulsan a Cuba hacia el cambio se destaca el tema de la independencia tecnológica expuesto a través de los lineamientos 223 y 226 del Partido Comunista de Cuba (PCC) que proponen "*elevar la soberanía tecnológica…"* así como *"ejecutar inversiones en la industria electrónica y de informática y comunicaciones…"* [2]. También se manifiestan otros factores económicos relativos al pago de licencias y las dificultades para adquirir el software propietario por vías alternas, con la limitante de no disponer del código fuente imposibilitando corregir errores de programación, vulnerabilidades, así como adaptarlo a exigencias del cliente. Por lo antes planteado la dirección del país impulsa el desarrollo de soluciones tecnológicas apoyando el trabajo de instituciones comprometidas con la evolución del software libre con el objetivo de reducir la dependencia económico-comercial y a largo plazo la necesidad social de adquirir software privativo ofreciendo mayor seguridad y servicios de forma gratuita [1].

La Universidad de Ciencias Informáticas (UCI) funge como institución coordinadora del desarrollo técnico de distribuciones y herramientas libres, disponiendo de los departamentos pertenecientes al Centro de Software Libre (CESOL) que tiene a su cargo el desarrollo de la distribución cubana GNU/Linux Nova y la misión de conducir procesos de migración a aplicaciones de código abierto, desde un modelo de integración de la formación, la investigación y el postgrado, contribuyendo así a la formación de estudiantes comprometidos con la Revolución que respondan a las necesidades del progreso científico-técnico y socio-económico del país.

El centro posee en su haber una variada gama de productos y soluciones entre las que se destacan las tres variantes de la distribución: Nova Escritorio, Nova Ligero y Nova Servidor, orientadas a sustituir el

software privativo en los Organismos de Administración Central del Estado, unidades presupuestadas, centros de educación y salud, bajo el paradigma de código abierto y los principios de seguridad, sostenibilidad, socio-adaptabilidad y soberanía tecnológica que no reemplazan las cuatro libertades definidas por el Proyecto GNU - *Free Software Foundation* sino que sumados a estas pretenden materializar las razones que llevan al país a ejecutar el necesario proceso de migración tecnológica.

Con la política migratoria puesta en marcha de forma gradual y escalonada, diferentes instituciones del país cuentan con la distribución cubana instalada en sus estaciones de trabajo. Como parte del proceso de migración la dirección de CESOL propone incrementar los servicios de consultoría, personalización, capacitación y soporte para las entidades que han migrado hacia GNU/Linux Nova. En este escenario los especialistas de migración para realizar configuraciones en las computadoras deben acceder al panel de control de estas y hacer las acciones pertinentes en cada una a la vez. Este inconveniente incurre en el trabajo repetitivo que conlleva a la pérdida de tiempo y esfuerzo, además de estar sujeto a errores humanos en tareas comunes, lo que trae consigo la insatisfacción y el rechazo de los especialistas de migración para llevar a cabo las configuraciones en las estaciones de trabajo migradas hacia GNU/Linux Nova.

En el sector privado del software existen sistemas dirigidos a la administración centralizada de las estaciones de trabajo a través de las políticas de dominio, que permiten administrar de forma remota funciones y características instaladas en los equipos interactuando con una intuitiva interfaz, tal es el caso de: *Desktop Central* orientado conjuntamente a dispositivos móviles y *Sophos Central* enfocado más a la seguridad. En el mercado libre se reconoce *Landscape*, como parte de los servicios de la empresa de software Canonical, con la virtud de gestionar las actualizaciones lanzadas a los equipos que trabajan con Ubuntu en la red [3]. También son reconocidas *Chef Compliance*, *CFEngine Enterprise*, *Puppet Enterprise, Saltstack Enterprise y Ansible Tower*, empleadas en la orquestación de la configuración para mejorar la automatización de los procesos en las entidades. Los anteriores sistemas ofrecen características únicas para la administración centralizada de la configuración de las máquinas, dispositivos y periféricos conectados a una red por lo que su estudio es acertado en la proximidad de modelar una solución para elevar la capacidad de administración de la distribución cubana GNU/Linux Nova.

La situación problemática antes descrita, permite plantear como **problema científico:** ¿Cómo garantizar la administración centralizada de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux

Nova con el propósito de incrementar la satisfacción de los especialistas de migración? El **objeto de estudio** lo constituye el proceso de administración centralizada de las estaciones de trabajo. Se establece como **campo de acción** el proceso de administración centralizada de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova. Se determina como **objetivo general**: desarrollar un servicio *RESTful* para la administración centralizada de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova con el propósito de incrementar la satisfacción de los especialistas de migración.

Para cumplir con el objetivo general de la investigación se definen los siguientes **objetivos específicos**:

- Analizar el proceso de administración centralizada de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova.
- Diseñar un servicio *RESTful* que permita la administración centralizada de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova.
- Implementar el servicio *RESTful* de administración centralizada de estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova.
- Evaluar el servicio *RESTful* de administración centralizada de estaciones de trabajo implementado.

Para el cumplimiento de los objetivos específicos se definen las siguientes **tareas de investigación:**

- Análisis de bibliografías relacionadas con el proceso de administración centralizada de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova para así comprender los principales elementos del proceso.
- Estudio de sistemas homólogos que permitan modelar un diseño que responda a exigencias definidas por la distribución cubana GNU/Linux Nova.
- Diseño de la propuesta de solución para la distribución cubana GNU/Linux Nova partiendo de los resultados previamente obtenidos.
- Implementación de la propuesta de solución para la distribución cubana GNU/Linux Nova.
- Aplicación de pruebas internas y de aceptación para verificar las funcionales del servicio *RESTful* desarrollado para la distribución cubana GNU/Linux Nova.
- Empleo de la técnica de Iadov para comprobar la satisfacción de los especialistas de migración en correspondencia con el servicio *RESTful* desarrollado para la distribución cubana GNU/Linux Nova.

La presente investigación propone como **hipótesis** que el desarrollo de un servicio *RESTful* para la administración centralizada de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux

Nova, permite incrementar la satisfacción de los especialistas de migración. Se identifica como **variable independiente** un servicio *RESTful* para la administración centralizada de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova.

La **variable dependiente** es la satisfacción de los especialistas de migración, ésta se operacionaliza en el Anexo 1.

En el desarrollo del trabajo de diploma se utilizaron los siguientes **métodos de la investigación científica:**

**Métodos Teóricos**

- **Histórico-Lógico:** aplicado para constatar teóricamente la evolución que han tenido los sistemas orientados a administrar estaciones de trabajo y los requisitos de configuración necesarios para configurar estaciones de trabajo de forma centralizada.
- **Analítico-Sintético:** utilizado para estudiar los elementos del problema planteado y analizar sus características para su posterior síntesis con el objetivo de elaborar la propuesta de solución de la investigación.
- **Modelación:** enfocado en representar el funcionamiento del proceso de administración centralizada de estaciones de trabajo mediante modelos para facilitar su estudio y discernimiento.

**Métodos Empíricos**

- **Entrevista:** utilizado para conocer y analizar el funcionamiento del proceso de configuración por parte de los especialistas de migración en la distribución cubana GNU/Linux Nova.
- **Encuesta:** aplicado a especialistas de migración para conocer su nivel de satisfacción sobre la propuesta implementada.

Con el objetivo de facilitar la comprensión del contenido de la investigación, se organizó la información en introducción, tres capítulos, conclusiones, bibliografía empleada y anexos; la estructura capitular se resume del siguiente modo:

**Capítulo 1: Proceso de administración centralizada de las estaciones de trabajo con el sistema operativo GNU/Linux.** En este capítulo se procede con una descripción del proceso de administración de las estaciones de trabajo. Se realiza el estudio de la documentación vinculada a los sistemas y herramientas que administran la configuración de forma centralizada, describiendo sus características y principales funcionalidades. Se seleccionan las tecnologías a emplear en la solución y el escenario adecuado correspondiente a la metodología.

**Capítulo 2: Servicio *RESTful* para la administración centralizada de la distribución cubana GNU/Linux Nova.** En este capítulo se realiza la propuesta de solución y se definen requisitos, historias de usuario, arquitectura y patrones de diseño del sistema para la administración centralizada de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova.

**Capítulo 3: Implementación y pruebas.** En este capítulo se evalúa el servicio *RESTful* desarrollado mediante la aplicación de pruebas de software para comprobar la calidad del sistema, además se emplea la técnica de Iadov para medir la satisfacción de los usuarios finales e identificar las diferencias del proceso automatizado con su antecedente manual.

## Capítulo 1: Proceso de administración centralizada de las estaciones de trabajo con el sistema operativo GNU/Linux

El presente capítulo se fundamenta en una descripción del proceso de administración centralizada de las estaciones de trabajo para sistemas GNU/Linux, y en el estudio analítico de sistemas similares enfocado a formular una posible estrategia de desarrollo. Se define empleando la metodología QSOS (*Qualification and Selection of Open Source Software*) la herramienta de administración de la configuración a utilizar; y las tecnologías de desarrollo considerando las tendencias modernas del desarrollo web. Además, se selecciona el escenario adecuado de la metodología AUP-UCI para el desarrollo del servicio *RESTful* a proponer como solución.

### 1.1 Conceptos fundamentales

**Administración centralizada:** proceso de realizar acciones sobre un grupo definido dependiente de un poder central [4].

**Estación de trabajo:** equipo informático con características avanzadas que se destina a una labor profesional, técnica o científica; por lo general está conectada a una red, vinculada a diferentes periféricos (impresoras, escáneres, etc.), a otras computadoras y a un servidor [4].

**Configuración:** serie de datos que establecen el valor de ciertas variables de un software o que indican cómo debe funcionar un aparato; estas informaciones suelen estar predefinidas, pero pueden ser modificadas [4].

### 1.2 Proceso de administración centralizada de estaciones de trabajo

Posterior al estudio de diferentes bibliografías y conceptos el autor define la administración centralizada de estaciones de trabajo como un proceso orientado a administrar el ciclo de vida de un equipo de cómputo ayudando a configurar, administrar y controlar de forma automatizada máquinas, laptops, dispositivos móviles e incluso puntos de acceso desde una ubicación central. Para garantizar un óptimo rendimiento es imprescindible contar con un computador de altas prestaciones destinado al trabajo técnico, este tendrá la misión de facilitar el acceso de los usuarios hacia los servidores y periféricos de la red. Las posibles acciones a desplegar varían en dependencia de las funcionalidades del sistema utilizado y de las características de las estaciones de trabajo. En este escenario la acción administrar se aplica a elementos de configuración como: sistema, recursos, software, hardware, red, usuarios, roles, actualizaciones,

antivirus, energía, tareas, interfaz, seguridad y copias de seguridad; ejemplificándose desde algo simple como cambiar el tapiz del escritorio hasta personalizar una copia de seguridad del servidor. A continuación, se describe cada uno de estos elementos de configuración.

**Sistema:** es un conjunto interconectado de componentes de hardware y software que comparten información electrónica para llevar a cabo una tarea solicitada.

**Recursos:** es cualquier componente físico o virtual de disponibilidad limitada en una PC que incluye medios para entrada, procesamiento, producción, comunicación y almacenamiento [5].

**Software:** es el soporte lógico de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas [6].

**Hardware:** son las partes físicas tangibles de un sistema informático; sus componentes eléctricos, electrónicos, electromecánicos y mecánicos.

**Red:** es un conjunto de equipos informáticos y software conectados entre sí por medio de dispositivos físicos que envían y reciben impulsos eléctricos, ondas electromagnéticas o cualquier otro medio para el transporte de datos, con la finalidad de compartir información, recursos y ofrecer servicios [7].

**Usuarios:** es un conjunto de permisos y de recursos (o dispositivos) a los cuales se tiene acceso, puede ser tanto una persona como una máquina, un programa, etc [8].

**Roles:** permiten definir grupos de usuarios para facilitar asignar tareas a estos.

**Actualizaciones:** módulo, paquete o parche que permite actualizar una aplicación o un sistema. Puede tratarse de una pequeña actualización para corregir algunos defectos, mejorar un programa o ponerlo al día, o puede ser una gran actualización que implica un cambio de versión.

**Antivirus:** programa informático que tiene el propósito de detectar y eliminar virus y otros programas perjudiciales antes o después de que ingresen al sistema.

**Energía:** garantiza el funcionamiento de los equipos de cómputo permitiendo configurar el ahorro de energía acorde a un plan determinado.

**Tareas:** software o aplicaciones del sistema que pueden estar o no en ejecución, también pueden ser gestionadas y programadas a conveniencia**.**

**Interfaz:** medio para señalar a la conexión que se da de manera física y a nivel de utilidad entre dispositivos o sistemas. Facilita la interacción del usuario con la computadora a través de la utilización de un conjunto de imágenes y objetos pictóricos (iconos, ventanas.) además de texto.

**Seguridad:** es el área relacionada con la informática y la telemática que se enfoca en la protección de la

infraestructura computacional y todo lo relacionado con esta y, especialmente, la información contenida en una computadora o circulante a través de las redes de computadoras [9].

**Copias de seguridad:** es una copia de los datos originales fuera de la infraestructura que se realiza con el fin de disponer de un medio para recuperarlos en caso de pérdida.

### 1.2.1 Proceso de administración centralizada de estaciones de trabajo en GNU/Linux Nova

Actualmente la administración centralizada de estaciones de trabajo en la distribución GNU/Linux Nova se realiza de forma manual y solo dispone del Gestor de Recursos de Hardware y Software (GRHS) para el control de inventario y de la aplicación *Gclient* para recoger inventarios de las PC clientes, ejecutar acciones y detectar incidencias para GRHS. En cuanto a las configuraciones se especifican *GSettings* y Gufw, el primero es una API (*Application Programming Interface)* que contiene la herramienta de línea de comandos *gsettings* para acceder o modificar la configuración compartida por muchos componentes de un escritorio; mientras que el segundo es una interfaz gráfica de software libre para el cortafuegos UFW (Uncomplicated FireWall) que facilita la configuración de las reglas de *Iptables.*

Para realizar el proceso de administración centralizada de estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova es necesario contar con un sistema de administración de la configuración destinado a ejecutar operaciones que respondan a las necesidades definidas en este proceso; por lo cual resulta inminente llevar a cabo el estudio de estos sistemas.

## 1.3 Sistemas de administración de la configuración

Los sistemas de administración de la configuración son software que ayudan a administrar y controlar múltiples equipos, facilitando las configuraciones realizadas, así como las instalaciones y actualizaciones de programas de forma paralela desde una *interfaz* web. Estos sistemas son necesariamente flexibles, diseñados para realizar acciones a una sola PC, a todas ellas o a un grupo específico, y a otros dispositivos conectados a la red como portátiles, servidores, móviles y demás. A continuación, se describen los sistemas de administración de la configuración identificados.

### 1.3.1 Desktop Central

*Desktop Central* ofrece amplias prestaciones para la administración y estandarización de los puestos de trabajo, servidores y dispositivos móviles desde una intuitiva consola web disponible en español. Permite automatizar las tareas de gestión desde su consola centralizada durante todas las fases de su ciclo de

vida, ahorrando costes, tiempo y optimizando los recursos. Entre las muchas funcionalidades que ofrece *Desktop Central* destacan el control remoto, la instalación de software, parches y paquetes de servicio, la estandarización del puesto de trabajo aplicando configuraciones comunes, las políticas de seguridad y control de dispositivos USB, las políticas de ahorro energético, la configuración de aspectos de Windows incluyendo opciones de Internet *Explorer*, *Outlook* y *Office* y la auditoría con un completo inventario de hardware y software, detección de aplicaciones prohibidas y control de utilización de las licencias adquiridas. Con su arquitectura flexible, permite administrar dispositivos en diferentes entornos de red como Directorio Activo o Grupos de Trabajo y *Novell eDirectory*, ahorrando considerablemente tanto en costes como en recursos. Los dispositivos administrados pueden estar en la LAN *(Local Area Network)* o distribuidos a través de una WAN (*Wide Area Network*), incluso en múltiples dominios o usuarios móviles. El módulo de Gestión de Dispositivos Móviles permite configurar los *smartphones* y tabletas *iOS, Android* y *Windows Phone* desde una consola central, permitiendo aplicar criterios diferenciados según sean los dispositivos de propiedad corporativa o personal [10].

### 1.3.2 Sophos Central

*Sophos Central* administra todo lo referido a *endpoint*, móvil, web, correo electrónico, servidor y seguridad inalámbrica. Usando una plataforma de administración de seguridad sincronizada, que brinda al cliente seguridad, políticas que siguen a los usuarios, fácil configuración, informes detallados y resumidos, y alertas de prioridad automática. Administra políticas de seguridad y múltiples dispositivos que se registrarán directamente para recibir nuevas configuraciones, todo desde una única interfaz web. Proporciona políticas predeterminadas y recomienda configuraciones para garantizar que obtenga la protección más efectiva desde el comienzo. Permite que sus productos compartan seguridad en tiempo real creando una protección más efectiva contra malware avanzado y ataques dirigidos aislando automáticamente los dispositivos infectados y propiciando una respuesta específica e inmediata. Gestiona toda la seguridad de manera simple y eficaz mediante su consola de administración integrada y basada en la nube [11].

### 1.3.3 Landscape

*Landscape* es una aplicación libre desarrollada por Canonical que ayuda a gestionar y monitorizar múltiples máquinas, facilitando las instalaciones y actualizaciones de programas en forma paralela. Ofrece dos versiones: *Hosted Edition*, a la que se accede a través de la web, y *Dedicated Server Edition*, que se

instala directamente en una PC. La principal virtud de *Landscape* es que permite administrar las actualizaciones que se tienen que lanzar a los equipos que trabajan con Ubuntu en una red. Permite monitorizar múltiples parámetros de los equipos e identifica automáticamente las actualizaciones de paquetes que incluyen correcciones de seguridad, por lo que se puede priorizar para mitigar rápidamente el riesgo. Los procesos de gestión consolidados en *Landscape* comienzan con la instalación de un agente liviano en los sistemas de destino. El agente realiza tareas administrativas y supervisa la recopilación de datos críticos en tiempo real sobre hardware, software, y almacena los datos en un repositorio centralizado de bases de datos en la consola de administración; los datos del rendimiento se controlan gracias a un módulo gráfico que representa variables como las tendencias de la temperatura, uso de disco y memoria, la carga del sistema o métricas personalizadas. De esta forma, los administradores solo necesitan acceder a una única interfaz para identificar detalles sobre todos los *endpoint* soportados. Los elementos de inventario de hardware, incluido el modelo de sistema, el tipo y la información de configuración, se pueden usar para agrupar lógicamente sistemas admitidos para la administración. Todo esto se controla a través de una interfaz web, sin embargo, existe la posibilidad de instalar *Landscape* en un servidor dedicado [12].

### 1.3.4 Chef Compliance

Es un producto de software de código abierto desarrollado por la compañía Chef orientado a la implementación y actualización rápida de aplicaciones críticas para el negocio. Esta plataforma se centra en automatizar la verificación de las políticas de seguridad del servidor para permitir la entrega rápida de aplicaciones ahorrando tiempo y evitando errores ya que no es necesario aplicar la configuración manual. Escanea toda la infraestructura en busca de riesgos de seguridad y problemas de cumplimiento, obtiene informes sobre riesgos y problemas clasificados por gravedad y niveles de impacto, y crea pruebas automatizadas en sus líneas de despliegue [13].

### 1.3.5 CFEngine Enterprise

Es una plataforma de automatización desarrollada por la empresa Cfengine AS, utiliza un enfoque basado en modelos para administrar su infraestructura y aplicaciones, a la vez que proporciona escalabilidad, seguridad, visibilidad y control para cualquier entidad que opte por esta solución. Hace que sea fácil rastrear servidores desplegados, simplemente instala el agente liviano en cada host y ejecuta el agente concentrador para configurar una relación de confianza entre el concentrador y los hosts; el agente

obtiene información sobre cada host (nombre de host, dirección, versión del sistema operativo, utilización del disco, paquetes instalados, etc.). Incluye administración de cumplimiento, informes, y herramientas para el manejo necesario de la complejidad. Permite cambiar la infraestructura completa de TI, grande o pequeña en poco tiempo, dada su capacidad de respuesta y agilidad que tolera implementar soluciones críticas de manera rápida y eficiente [14].

### 1.3.6 Puppet Enterprise

Es una plataforma de código abierto desarrollada por la corporación *Puppet Labs* que ayuda a las organizaciones a gestionar la configuración desde una interfaz. Obtiene todos los beneficios de código abierto y la garantía de tener una plataforma unificada y fácil de usar que combina un enfoque basado en modelos con una ejecución de tareas para administrar con eficacia la infraestructura híbrida en todo su ciclo de vida. Impone el estado deseado de sus configuraciones y soluciona automáticamente cualquier cambio inesperado. Proporciona control de versiones, revisión de códigos, pruebas automatizadas, integración continua e implementación automatizada. Inspecciona e informa sobre los paquetes que se ejecutan en la infraestructura identificando los que son para actualizaciones de mantenimiento, parches de seguridad y renovaciones de licencias [15].

### 1.3.7 Saltstack Enterprise

Es una API de código abierto desarrollada por *Saltstack*, escalable y fácil de usar para las organizaciones de TI o el desarrollo para la automatización y la orquestación. Permite la automatización impulsada por eventos de la administración de la infraestructura y la seguridad, así como un consumo eficiente de datos y eventos de *Salt*, incluye una base de datos integrada y rastrea el historial y el uso de toda la actividad del sistema. Incluye una interfaz gráfica de usuario, funciones de informes y auditoría, controles y permisos basados en el usuario, una base de datos e integraciones y soporte para tecnologías de terceros patentadas [16].

### 1.3.8 Ansible Tower

Es una API centralizada de automatización basada en la web con una interfaz simplificada e intuitiva que hace que *Ansible* sea aún más fácil de usar. Plataforma desarrollada por Red Hat que centraliza y controla *Ansible* desde una herramienta gráfica, aportando gestión y control de roles de usuarios, programación de trabajos y gestión gráfica de inventario. La supervisión de nodos en tiempo real, las visualizaciones de

actividades laborales recientes y una lista de nodos problemáticos permiten realizar rápidas evaluaciones de salud. El control de acceso basado en roles mantiene los entornos seguros y los equipos eficientes; toda la actividad de automatización se registra de forma segura y se almacena. [17], [18].

### 1.3.9 Análisis de los sistemas de administración de la configuración

El desarrollo y despliegue de estos sistemas está distribuido hacia diferentes mercados: software privativo y software libre; por consecuente no es posible implementar un sistema privado en una distribución GNU/Linux y hacer uso de Windows para utilizar un software propietario no es opción ya que no está contemplado en la estrategia de informatización del país. Canonical, la empresa que desarrolla Ubuntu ofrece a *Landscape* como una alternativa del mercado libre pero solo es posible acceder actualmente a una versión de prueba de 60 días. La entidad como parte de su modelo de suscripción y soporte taza un costo por unidad al año de US$100.00, por lo que este sistema no está disponible, sino que forma parte de los servicios de soporte que facilita Canonical. En el caso de *Chef Compliance*, *CFEngine Enterprise*, *Puppet Enterprise, Saltstack Enterprise y Ansible Tower* la adquisición de las licencias para una suscripción estándar por unidad al año se resuelve en un costo de US$72.00, US$100.00 US$120.00, US$150.00 y US$50.00 respectivamente; entonces implementar uno de estos sistemas como solución resulta inconcebible para la situación económica actual del país. Con el fin absoluto de mantener el principio de soberanía tecnológica y formular una solución viable para la situación a resolver se hace forzado el estudio y análisis de la arquitectura, funcionalidades y elementos visuales de estos sistemas. Todo enfocado a desarrollar una solución para automatizar el proceso de administración de las estaciones de trabajo en la distribución GNU/Linux Nova. En aras de implementar una solución para solventar las dificultades del proceso se encauza el estudio a las herramientas de administración de la configuración que emplean estos sistemas.

## 1.4 Herramientas de administración de la configuración

Las herramientas de administración de la configuración (*CMT*, *Configuration Management Tool*) permiten la automatización de la instalación, configuración y actualización de software en un sistema. Adicionalmente simplifican las tareas de gestión de grandes y complejos despliegues manteniendo los sistemas actualizados, reduciendo los costos de operación. De forma general estas herramientas presentan las siguientes funcionalidades:

- Configuración del nodo maestro/hijos

- Instalación de paquetes
- Configuración de usuario/grupo
- Desplegar un archivo estático
- Desplegar un archivo de plantilla
- Ejecutar un servicio

La variedad de estas herramientas disponibles en el mercado hace difícil la selección de una adecuada que se ajuste a las necesidades del cliente. Se realizó un estudio previo que limitó la investigación de otras herramientas como *Isconf, Juju, Cdist, Smartfrog, Quattor, LCFG* y *Rex* por escasa documentación, falta de soporte para Linux, desconocimiento de las organizaciones que las emplean, poco respaldo de la comunidad y falta de actualización (10 años desde la última) [19].

Con el propósito de seleccionar una *CMT* como herramienta de administración para el servicio *RESTful* prediseñado se procede al estudio de cinco de las *CMT* más reconocidas: *Chef, CFEngine, Puppet, Saltstack y Ansible*. Para la selección se definieron criterios que dan lugar a un marco de comparación que define las características imprescindibles para establecer un juicio [20].

**Chef**

Desde su lanzamiento en enero del año 2009, este programa ha ido evolucionando hasta convertirse en una herramienta imprescindible para multitud de empresas; contiene soluciones para sistemas pequeños y grandes, con características y precios para los respectivos rangos. Bajo la licencia *Apache* y escrito en *Ruby* y *Erlang* utiliza un lenguaje *Ruby*, específico de dominio (*domain-specific language - DSL*) para escribir la configuración del sistema. Los usuarios elaboran "recetas" que cómo *Chef* gestiona las aplicaciones y utilidades del servidor *Apache*, *HTTP Server*, *MySQL* o *Hadoop* describen y sus configuraciones. *Chef* puede ejecutarse en modo cliente/servidor, o en una configuración independiente llamada "chef-solo". Brinda la posibilidad, usando un solo servidor *Chef*, de acceder y administrar la importante cifra de más de 10 mil nodos, tanto físicos como virtuales o en la nube; en este sentido, los recursos se ofrecen gratis solo hasta 25 nodos [21].

**CFEngine**

Ya con más de dos décadas de creada desde su lanzamiento en 1993 por Mark Burgess, *CFEngine* se consolida en el mercado como una herramienta madura y ampliamente utilizable de administración de la configuración. Escrita en lenguaje C de tipo declarativo y bajo la Licencia Pública General de GNU (GPL) provee un funcionamiento basado en la arquitectura cliente-servidor y orientado a servicios. Esta CMT con

lenta curva de aprendizaje permite verificar contenido e integridad de archivos, desplegar aplicaciones y actualizar sistemas bajo la autonomía de hosts como principio básico de seguridad. Se distingue por su arquitectura distribuida enfatizada en agentes que se ejecutan en cada host administrado; el servidor central solo se emplea para coordinar la distribución de las configuraciones expresadas como "*promises*" que describen el estado deseado del sistema. *CFEngine* compara el sistema en tiempo real con el estado deseado y en caso de que el sistema esté equivocado de estado, realiza los cambios necesarios de modo automático, y actualiza de la misma forma a los clientes, que pueden mantenerse en funcionamiento con los datos almacenados en caso de que el servidor desapareciera. El costo de su versión *Enterprise* puede disminuir a la mitad tratándose de una institución educacional [19], [22].

**Puppet**

Es una herramienta de administración de la configuración de código abierto fundada por Luke Kanies en 2005. Escrita en *Ruby* y liberada bajo la Licencia Pública General de GNU (GPL) permite administrar la configuración de sistemas de forma declarativa e imperativa. El usuario describe los recursos del sistema y sus estados, esta información es almacenada en archivos denominados "manifiestos". La utilidad llamada *Facter* encuentra la información del sistema, y compila los manifiestos en un catálogo específico del sistema que contiene los recursos y la dependencia de estos. *Puppet* presenta una permite controlar la ejecución de diferentes servicios en una máquina y la instalación de paquetes en estas, administrar la presencia y disponibilidad de puntos de montaje, gestionar cualquier archivo dentro de un sistema, así como los usuarios y grupos pertenecientes al mismo. Ostenta una curva de aprendizaje lenta y un importe considerable que incluye los servicios de *Amazon Web Services* (AWS) usados; aunque es posible descargar una versión de evaluación gratuita para administrar 10 nodos en caso de conveniencia [23].

**Saltstack**

A pesar de que su lanzamiento no se consideró estable hasta noviembre del 2011, Thomas S Hatch desarrolló una plataforma de administración de sistemas y configuración reconocida por su escalabilidad y poder. Escrita en *Python* y bajo la licencia *Apache* optimiza la gestión de un gran número de servidores reduciendo el tiempo de instalación del sistema operativo, las aplicaciones y los errores que se producen en este proceso. Los usuarios se benefician de una automatización única orientada a eventos para orquestar y controlar cualquier infraestructura y entorno de software. Permite replicar configuraciones de forma periódica, ejecutarse en múltiples servidores fácilmente, configurar un servidor con un rol determinado y mantener un control de versiones de la configuración; además verifica los servidores para

comprobar que su estado de configuración es correcto y se adapta a las plantillas de los archivos de configuración que ya existen. La oferta completa *SaltStack Enterprise* proporciona a los clientes implementaciones más rápidas a costos de propiedad más bajos negociables con el proveedor; los clientes suscritos reciben atención exclusiva, priorizada y acceso a servicios de consultoría. La plataforma presenta una curva de aprendizaje significativa para los nuevos usuarios, incluso para los más experimentados profesionales [24], [25].

**Ansible**

Es una herramienta reciente que ha tenido un crecimiento formidable en los últimos 5 años, desarrollada por Michael DeHaan a inicios del 2012 es usada en importantes empresas por su flexibilidad y simpleza. Escrita en *Python* y liberada bajo la Licencia Pública General de GNU (GPL), es reconocida por la gestión de la configuración, el despliegue de aplicaciones y la orquestación de servicios de forma limpia y sencilla. A diferencia de otras soluciones similares su funcionamiento es mediante la ejecución remota de comandos; y se liberan nuevas versiones cada dos meses que se desarrollan valorando la simplicidad en el diseño del lenguaje y la configuración para mantener la facilidad de uso y su rápida curva de aprendizaje. *Ansible* no necesita agentes ni configuraciones complicadas pudiéndose ejecutar en modo *Ad*-Hoc o utilizar "*playbook*"; el primero permite efectuar acciones simples sobre los *hots* haciendo uso de los módulos de la herramienta; los *playbooks* se componen por varios *plays* escritos en YAML (*YAML Ain't Markup Language*) y son los que ayudan a manejar/configurar los diferentes hosts pudiendo realizar el trabajo a gusto del usuario haciendo uso de los inventarios, *tags* y roles [17], [26].

### 1.4.1 Resumen de características

La siguiente tabla resume las principales características de las CMT caracterizadas en el epígrafe anterior.

**Tabla 1.** Resumen de características de las CMT

(Fuente: basado en la referencia [20])

| **Criterios de Comparación** | **Herramientas de Administración de la configuración** | | | | |
|---|---|---|---|---|---|
| | Chef | CFEngine | Puppet | Saltstack | Ansible |
| Propiedades de las especificaciones | | | | | |
| Tipo de lenguaje | declarativo, imperativo | declarativo | declarativo, imperativo | declarativo | declarativo |

| Interfaz de usuario | CLI,Web | | | | |
|---|---|---|---|---|---|
| Mecanismos de modularización | Si | Si/Design Center | Si/Forge | Si | Si/Galaxy |
| Propiedades del despliegue | | | | | |
| Escalabilidad | pequeños-medios-grandes | medios-grandes | medios-grandes | pequeños-medios-grandes | pequeños-medios |
| Flexibilidad | | | | | |
| Servidores | AIX, RHEL/Centos, FreeBSD, Oracle Linux, Arch Linux, Debian, Fedora, OS X, Solaris, Ubuntu, Windows | CentOS, Debian, RHEL, Ubuntu | Red Hat Enterprise Linux, CentOS, Oracle Linux, Scientific Linux, SUSE Linux Enterprise Server, Ubuntu | Plataformas Unix con Python 2.6 o superior e inferior a 3.0 | Plataformas con Python 2.6 o 2.7 (No Windows) |
| Clientes | AIX, RHEL/Centos, FreeBSD, Oracle Linux, Arch Linux, Debian, Fedora, OS X, Solaris, Ubuntu, Windows | AIX, CentOS, Debian, UP-UX, RHEL, SLES, Solaris, Ubuntu, Windows | Red Hat Enterprise Linux, CentOS, Oracle Linux, Scientific Linux, SUSE Linux Enterprise Server, | Plataformas Unix o Windows | Plataformas con Python 2.4 o superior e inferiores a 3.0 |

| | | | | | |
|---|---|---|---|---|---|
| | | | Ubuntu, Debian, Windows | | |
| Dispositivos | host, dispositivos de red | host | host, dispositivos de red | | |
| Propiedades de gestión | | | | | |
| Integración con el entorno | Características de los host administrados durante la ejecución | | | | |
| Resolución de conflictos | NE | Conflictos de Modalidad | Conflictos de Modalidad | NE | NE |
| Soporte para SVC | si/Repositorio externo | | | | |
| Control de acceso | Autenticacion integrable con recursos externos y Autorización | | | | |
| Usabilidad | | | | | |
| Soporte para test de especificación | Manual, comprobación de sintaxis | Manual, comprobación de sintaxis | Puppet-lint, rspec-puppet, Beaker framework | Manual, componentes para test de integración y unitario | Manual, comprobación de sintaxis |
| Monitoreo de la infraestructura | Si | | | | |
| Soporte | | | | | |
| Documentación disponible | Documentación, referencias, tutoriales, presentaciones, videos | | | | |
| Documentación de instalación | web+pdf | web | Web | web+pdf | web+pdf(para Tower) |
| Formas de | repo, script, | repo, script, | repo, script, | repo, manual, | repo, manual, |

| | | | | | |
|---|---|---|---|---|---|
| instalación | manual | manual o Vagrant | manual | script | script wizard para Tower |
| Elementos para instalación | RVM, Ruby, Git, Hosted Chef | Instalador del servidor de políticas y el de los clientes. | Puppet Master, PuppetDB y PostgreSQL, Console, Agents | msgpack-python, YAML, Jinja2, MarkupSafe, apache-libcloud, Request, ZeroMQ o RAET | python-simplejson, Ansible Core, Ansible Tower, libselinux-python(si esta habilitado SELinux) |
| Recursos para evaluar el producto | Maquina virtual de prueba hasta 25 nodos | Hasta 25 nodos versión Empresarial con Vagrant | Maquina virtual de prueba hasta 10 nodos, cursos online | Vagrant para prueba de un master con 2 nodos | Tower con Vagrant y recursos para AWS |
| Frecuencia/política de liberación | NE | 2 por año | NE | NE | cada 2 meses |
| Funcionalidades adicionales de la versión comercial | GUI con dashboards para monitoreo de estado de los host, alertas, inventario y planificacion de tareas | | | | |
| Comunidad | | | | | |
| Contribuidores | 526 | 90 | 467 | 1944 | 3104 |
| Canales de comunicación | Redes sociales, IRC, RSS, Youtube, grupos de | Redes sociales, IRC, Newsletter, RSS, Youtube | Redes sociales, IRC, Newsletter, RSS, Youtube, | Redes sociales, IRC, Newsletter, RSS, Youtube, | Redes sociales, IRC, Newsletter, RSS, Youtube, grupos de |

| | | | | | |
|---|---|---|---|---|---|
| | Google, Reddit | | grupos de Google, Puppet Ask, Reddit | grupos de Google, Reddit | Google |
| Actualización | | | | | |
| Tiempo desde la ultima entrega | <6 meses | <6 meses | <6 meses | <6 meses | <6 meses |
| Año de creación | 2009 | 1993 | 2005 | 2011 | 2012 |
| Popularidad | | | | | |
| Clientes | Facebook, Mozilla, Airbnb, Expedia, IGN, Marshall University Socrata, University of Minesota, Disney | Intel, Linkedin, SAMSUNG,Chevron, Comcast, DIRECTV | SONY, redhat, Twitter, SUN Oracle, Motorola, WMWare, github, Intel, at&t , NESTLE | Linkedln, NASA JPL, Departamento de Defensa de los E.U, Universidad de Columbia Y Hardvard, Rakspace, Salesforce | Evernote, Universidad de Hardvard, Verizon, Twitter |
| Socios | Amazon Web Services, Microsoft Azure y Google Cloud Platform | IBM, Cisco, Amazon, Microsoft, HP, VMWare | VMWare, Redhat, RackSpace | NE | Redhat, VMWare, Cisco, HP |
| Tipo de licencia | Apache License v2.0 | GPL v3.0 | Apache License v2.0 | Apache License v2.0 | GPL v3.0 |
| Precio anual | $7200.00 | $5000.00/$1000 | $10500.00 | $15000.00 | $3000.00**/**$1000 |

| | | | | | |
|---|---|---|---|---|---|
| para 100 nodos con version Enterprise | | 0.00 | | | 0.00 |
| Lenguaje de desarrollo | Ruby, Erlang | C | Ruby | Python | Python |
| Requerimientos de hardware | | | | | |
| Servidor | 4 núcleos, 8GB RAM, 5GB HDD en opt, 5GB HDD en var | 12 núcleos, 2GB RAM, 100MB HDD por agente | 4 núcleos, 16GB RAM, 142GB HDD | 4GB RAM, 30GB HDD | 4GB RAM, 20GB HDD |
| Cliente | NE | 256MB RAM,100MB HDD | NE | | |

NE: no especificado oficialmente.

### 1.4.2 Comparación entre las herramientas

Para facilitar la selección de una herramienta de administración de la configuración se propone comparar las descritas anteriormente. Se concibe la metodología QSOS, disponible bajo los términos de la GNU *Free Documentation License* para calificar, seleccionar y comparar software de forma objetiva, trazable y argumentada; el proceso consiste en cuatro etapas: definición, evaluación, calificación y selección. Se establece un método de gestión de riesgos y calificación de software para cuantificar y medir las posibilidades reales de implantación del software ofreciendo posibilidad de comparación al establecer criterios ponderados, en base a los cuales calificar el software y hacer una selección final de la manera más objetiva y beneficiosa [27].

### 1.4.3 Aplicación de la metodología QSOS

Para la selección de la herramienta adecuada se precisa una comparación entre las descritas anteriormente haciendo uso de la metodología QSOS; se seleccionará la que en su totalidad posea mayor puntaje atendiendo a los siguientes criterios de comparación:

1. **Escalabilidad:** Nivel de escalabilidad del software en escenarios

La escalabilidad es la propiedad de aumentar la capacidad de adaptarse a cambios significativos sin comprometer el funcionamiento y calidad del negocio. Para escalar una herramienta CMT se utilizan técnicas similares a las aplicadas sobre aplicaciones web: usar una red de mayor potencia que admita más solicitudes, usar un servidor más robusto para el maestro y/o usar múltiples maestros para equilibrar la carga [28]. *Saltstack* es más escalable debido a su característica de varios maestros y la comunicación rápida de *zeroMQ*. *Chef* es lo suficientemente flexible como para ajustarse a cualquier dinámica de escala sin requerir cambiar el flujo de trabajo [19].

0. No es escalable
1. Escalable en rango medio-grande o pequeño-medio
2. Escalable en rango el pequeño-medio-grande

2. **Curva de aprendizaje:** Velocidad con que se puede dominar la herramienta

"*Ansible* tomó menos horas para configurar un proyecto (2 horas). *Saltstack* tiene una curva de aprendizaje más alta y tomó un poco más de tiempo (5 horas). *Puppet* presentó algunos parches ásperos y tomó (9 horas). *Chef* fue el más duro y tomó (12 horas). Obstaculizaron la configuración a *Puppet* y *Chef*, las incoherencias, documentación desactualizada y flujos confusos" [28]. *CFEngine* ha adquirido reputación por ser difícil de aprender, principalmente porque exige a los usuarios pensar de una manera diferente en cuestiones abstractas como predictibilidad y alineación comercial. En cambio, los usuarios perciben a *Puppet* como más fácil de entender en comparación a *CFEngine* porque permite a los administradores mantener los hábitos a los que están acostumbrados, incluso cuando tales hábitos son inconcebibles [22]. Por lo antes expuesto se concluye que *Chef*, *CFEngine* y *Puppet* están en total desventaja en cuanto a usabilidad se refiere.

0. Aprendizaje lento
1. Aprendizaje asequible
2. Aprendizaje rápido

3. **Seguridad:** Seguridad de los modelos y protocolos que utilizan

En un artículo publicado en *ScriptRock* el 17 de marzo del 2014 donde se compara *Ansible* con *Salt* se afirma que muchos usuarios consideran la configuración de *Ansible* sin maestro y solo puro SSH (*Secure Shell*) más seguro que los modelos de agente maestro que usan *Saltstack* con *zeroMQ*, *Puppet* con SSL (*Secure Sockets Layer*), Chef con HTTPS (*Hypertext Transfer Protocol Secure*) y *CFEngine* con Open-

SSH [20]. *Ansible* emplea pares de claves públicas y privadas a cada nodo, de forma que puedan autenticarse mutuamente y emplea el protocolo SSH para mantener la confidencialidad. *Puppet* hace uso de una infraestructura de clave pública, empleando certificados para proteger las peticiones que se realizan entre los componentes de la arquitectura. *Chef* utiliza el protocolo HTTPS para proteger los mensajes y emplea secretos pre-compartidos entre el agente y el servidor maestro. Saltstack usa la librería de seguridad *ZeroMQ* como una forma de conexión asincrónica para enviar mensajes subyacentes de alta velocidad entre el nodo principal y el nodo cliente por medio de llaves numéricas [29]. *CFEngine* emplea un protocolo específico para encriptación y autenticación basado en un tratamiento simplificado del protocolo *Open-SSH* que proporciona seguridad y ligereza en la transferencia de datos; este protocolo ha sido aprobado para los Estándares Federales de Procesamiento de la Información (FIPS) 140-2 del inglés *Federal Information Processing Standard* [22].

0. Protocolo poco seguro
1. Protocolo seguro
2. Modelo y protocolo seguros

4. **Potencialidad:** Capacidad de la herramienta para el desarrollo

*Saltstack* y *Ansible* son sistemas con una arquitectura basada en *Python* por lo que son más ligeros y fáciles de aprender mientras que *Chef* y *Puppet* son más complejos por su arquitectura basada en *Ruby* proporcionando un medio más experimentado, probado y con más funcionalidades en comparación a sus homólogos. En comparación *CFEngine* se muestra como una herramienta robusta con mayor madurez que al estar basada en C y descentralizada resuelve ser más rápida y eficiente que las ya mencionadas basadas en *Ruby* [19].

0. Herramienta simple
1. Herramienta compleja

5. **Contribuidores:** Cantidad de usuarios contribuyentes del proyecto

El elemento más importante en el escenario competitivo de una comunidad son sus contribuidores. Los resultados basados en métricas de *GitHub*, exponen a *Puppet* y *Chef* con puntaje alto en cuanto a seguidores, ofertas de trabajo y eventos. Pero para deleite *Ansible* y *SaltStack* tiene muchos más contribuidores e incluso estrellas de *GitHub* [28].

0. Menos de 500 contribuyentes
1. Entre 500 y 3000 contribuyentes

2. Más de 3000 contribuyentes

6. **Licencia:** La licencia bajo la cual se distribuye posee la cualidad *copyleft*

La licencia GPL v3.0 posee *copyleft* por lo que exige la publicación del código fuente y que todos los trabajos derivados del original conserven la misma licencia GPL. La licencia Apache v2.0 es permisiva pues no requiere que los trabajos derivados sean publicados bajo la misma licencia y tampoco exige la liberación del código fuente [30].

0. Posee la cualidad *copyleft*
1. No posee la cualidad *copyleft*

7. **Requerimientos de hardware:** Requisitos mínimos de memoria RAM y HDD

Los requerimientos de hardware definen las características indispensables que debe tener el hardware de un equipo de computo para poder soportar y/o ejecutar una aplicación de software. En caso de que el equipo no cumpla con los requisitos mínimos de hardware para una herramienta será imposible la ejecución de esta; y a menor consumo de recursos (RAM, HDD, CPU) mayor probabilidad de emplearse la herramienta en equipos de bajas prestaciones.

0. Emplea mucha memoria RAM y espacio en HDD
1. Emplea poca memoria RAM y mucho espacio en HDD, o mucha memoria RAM y poco espacio en HDD
2. Emplea poca memoria RAM y espacio en HDD

**Tabla 2.** Selección de la herramienta CMT

(Fuente: elaboración propia)

| | **Chef** | **CFEngine** | **Puppet** | **Saltstack** | **Ansible** |
|---|---|---|---|---|---|
| Escalabilidad | 2 | 1 | 1 | 2 | 1 |
| Curva de aprendizaje | 0 | 0 | 0 | 1 | 2 |
| Seguridad | 1 | 1 | 1 | 1 | 2 |
| Potencialidad | 1 | 1 | 1 | 0 | 0 |
| Contribuidores | 0 | 0 | 0 | 1 | 2 |
| Licencia | 1 | 0 | 1 | 1 | 0 |
| Requerimientos de hardware | 1 | 2 | 0 | 2 | 2 |
| Total | 6 | 5 | 4 | 8 | 9 |

Posterior a investigar y comparar las herramientas de administración de la configuración (CMT) aplicando la Metodología para la Clasificación y Selección de Software Libre y Código Abierto QSOS, se resuelve emplear *Ansible* como CMT para implementar la solución pues se adapta mejor a los requisitos descritos. Aunque cabe destacar que *CFEngine* a pesar de su complejo aprendizaje y dominio proporciona la solución más completa en términos de funcionalidad y módulos; la seguridad y usabilidad que brinda *Ansible* es primordial para eliminar riesgos en el desarrollo de la solución.

### 1.5 Caracterización de Ansible

Para el trabajo con *Ansible* se hará uso de su *Comand-Line Interface* (CLI) que permitirá dar instrucciones precisas en cuanto a configuraciones ejecutando comandos de *Shell* sobre SSH, como un medio de transporte y transferencia de módulos desde una máquina de control a las máquinas gestionadas a distancia. Estos módulos funcionan con *JSON* y sirven de base para evaluar instrucciones de automatización, por ejemplo, módulos de gestión de paquetes, de servicios, de edición de archivos y de línea de comandos; describen cómo obtener un recurso determinado del estado actual de un entorno a otro y pueden estar escritos en cualquier lenguaje dinámico [31]. A continuación, se muestra la arquitectura de *Ansible* para facilitar la comprensión de dicha herramienta, ver Imagen 2.

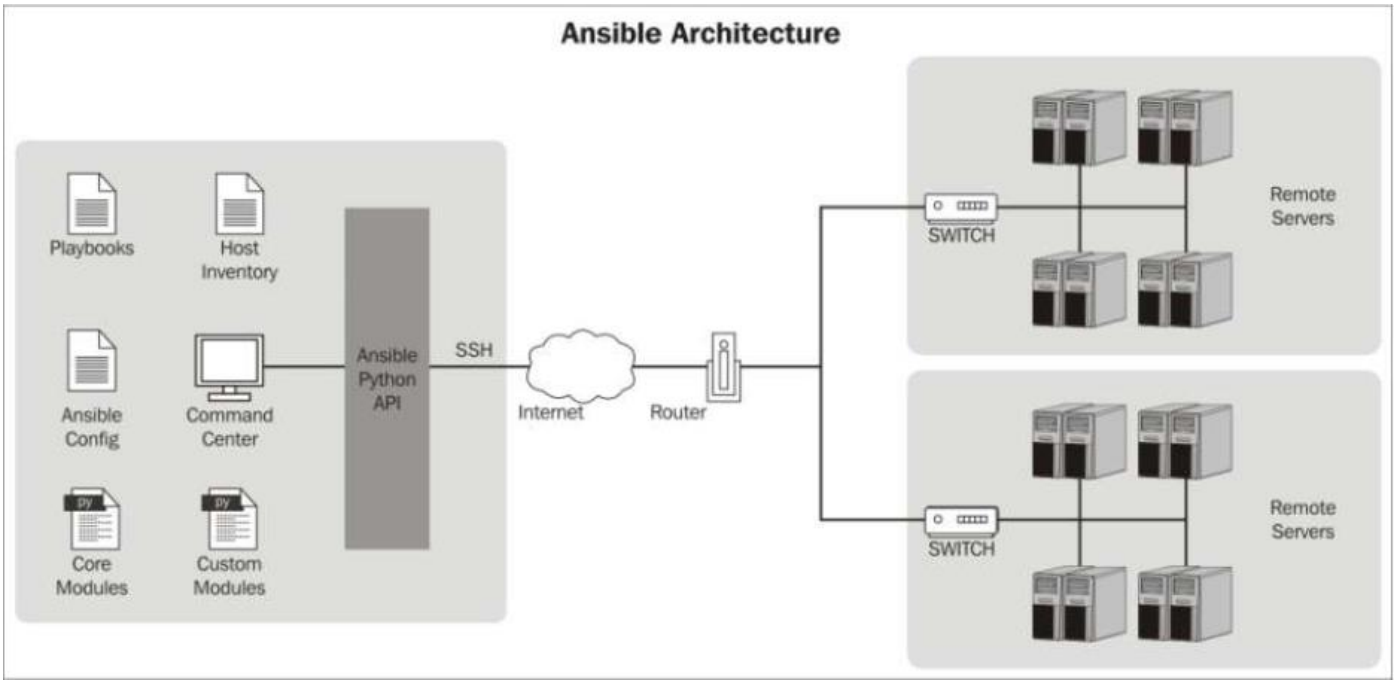


**Imagen 1.** Arquitectura de *Ansible*

(Fuente: referencia [29])

*Ansible* es un sistema de aprovisionamiento *agentless*, pues solo debe instalarse en la máquina que controlará al resto lo que permite ahorrar el proceso de instalación y gestión de los agentes en las máquinas remotas, facilitando enormemente el mantenimiento de la arquitectura. Se centra en la facilidad de uso, facilidad de desarrollo/cambio de contenido de automatización y seguridad. Su enfoque es distinto a las otras herramientas, pues esta empuja la configuración hacia fuera a través del protocolo SSH, además de proporcionar capacidad de usar un cliente estándar [32].

### 1.5.1 Funcionamiento

Toda la configuración de *Ansible* se almacena en el directorio */etc/ansible/*, este contiene por defecto dos ficheros: hosts y *ansible.cfg*, el primero se refiere a los inventarios mientras que el segundo contiene la configuración general. *Ansible* necesita hacer uso de un inventario que contenga las máquinas que desea configurar para poder administrarlas. El archivo de inventario predeterminado se denomina hosts y se coloca en */etc/ansible/host*, con formato de archivo INI y contiene un listado de las direcciones *IP* (Internet Protocol) o *hostname* de cada nodo. Los nombres de los grupos están encerrados entre corchete, y los parámetros se asignan a ese grupo, hasta el siguiente título de grupo; una máquina puede estar en muchos grupos a la vez y las máquinas desagrupadas se especifican antes que los grupos [26]. Un ejemplo de grupo puede ser:

*[laboratorio]*
*pc01*
*10.8.123.53*
*192.168.56.1*

Una vez definidos los hosts en el inventario de *Ansible*, antes de comenzar a ejecutar comandos es recomendable probar la conectividad de las PC para confirmar que podemos configurarlas. *Ansible* incluye un módulo simple llamado ping que permite probar la conectividad mediante la conexión SSH que envía y ejecuta en la máquina administrada. Esto significa que las PCs cliente no necesitan tener instalado *Ansible* y que la CMT puede usar los mismos canales que ya está utilizando para administrar las PCs [26]. La conexión SSH del servidor a los nodos se realiza desde el server *Ansible* mediante los pasos:

1) Editar o crear el archivo */etc/ansible/hosts* y agregar los *IPs* o *hostnames* de los servidores a administrar con *Ansible*:

10.8.123.53

2) Generar llave SSH:

*ssh-keygen*

3) Pasar la llave generada a cada nodo:

*ssh-copy-id -i ~/.ssh/id_rsa.pub crgarcia@10.8.123.53*

Debe pedir la contraseña de usuario de la estación de trabajo

4) Comprobar conexión ssh para un nodo con:

*ansible all –m ping -u crgarcia*

Se debe producir el siguiente código resultante:

*10.8.123.53 | success >> {*

*"changed": false,*

*"ping": "pong"*

*}*

4.1) Chequear conexión ssh para todos los nodos:

*ansible all -m ping*

Debería dar la siguiente salida:

*10.8.123.53 | success >> {*

*"changed": false,*

*"ping": "pong"*

*}*

*example.domain.com | success >> {*

*"changed": false,*

*"ping": "pong"*

*}*

*example2.domain.com | success >> {*

*"changed": false,*

*"ping": "pong"*

*}*

Requisito: para el éxito de la conexión SSH del servidor con los nodos, estos necesariamente tienen que tener instalado el paquete *openssh*.

El fichero *ansible.cfg* ubicado en /etc/ansible/ansible.cfg indica la configuración general de ansible, esta configuración contiene el fichero con el catálogo de clientes, el usuario remoto a utilizar y la clave pública que se va a utilizar para el acceso SSH [26]. Ejemplo del contenido de este fichero:

*[defaults]*
*hostfile = hosts*
*remote_user = debian*
*sudo = yes*

### 1.5.2 Playbooks

*Ansible* admite la automatización de configuraciones a través de lineas de código llamadas *playbooks* que permiten realizar tareas más complejas; estos se componen de un conjunto de acciones o *plays*, que se ejecutan de forma secuencial sobre una máquina destino. El objetivo de una jugada es reproducir en uno o varios hosts destino, un conjunto de tareas que lo dejarán en el estado deseado. Una tarea, no es nada más que una llamada a un módulo de *Ansible*. En el índice de módulos de *Ansible*, encontramos módulos para casi cualquier acción relacionada con la gestión de un servidor, como:

- copy - copia archivos a *hosts* remotos
- get_url - descarga archivos de HTTP, HTTPS o FTP a nodos
- apt - administra paquetes para Debian o Ubuntu
- shell – ejecuta comandos shell en hosts remotos
- dconf - modifica y lee de la base de datos *dconf*
- ufw - administra el *firewall* con UFW
- script - ejecuta un *script* de *bash*

Un *playbook* se codificará en formato YAML, un formato de datos serializados especialmente diseñado para ser entendible por el ser humano, lo que facilita tanto la definición de tareas, como su posterior lectura [31]. Los *playbook* constan de tres secciones:

- Sección objetivo: define los hosts en los que se ejecutará la jugada, y cómo se ejecutará, aquí se configura el nombre de usuario de SSH.
- Sección de variable: define las variables que estarán disponibles mientras se ejecuta un play.
- Sección de tarea: enumera todos los módulos en el orden deseado.

Para ejecutar un *playbook* se usa el comando: *ansible-playbook ejemplo-play.yml.* Un ejemplo completo de una playbook se define en el siguiente fragmento de código:

*- hosts: laboratorio*

*user: root*

*vars:*

*tasks:*

name: Install packages

apt: name= {{item}} state=installed

with_items:

- pidgin
- clementine

### 1.6 Servicio web RESTful

El consorcio *World Wide Web* (W3C) define los Servicios Web como sistemas de software diseñados para soportar una interacción interoperable máquina a máquina sobre una red. El estilo de arquitectura de software conocido como *Representational State Transfer* (REST) constituye una opción de estilo de uso de los Servicios Web. Los Servicios Web *RESTful* emulan al protocolo *Hypertext Transfer Protocol* (HTTP) o protocolos similares mediante la restricción de establecer la interfaz a un conjunto de operaciones estándar (*GET, POST, DELETE, PUT*); por lo que este estilo se centra más en interactuar con recursos, que con mensajes y operaciones. El término es utilizado en el sentido de describir cualquier interfaz que transmite datos específicos de un domino sobre HTTP sin una capa adicional; a continuación, se muestran las principales características y ventajas de los servicios web *RESTful* [33].

**Características:**

- Las operaciones se definen en los mensajes
- Una dirección única para cada instancia del proceso
- Cada objeto soporta las operaciones estándares definidas
- Componentes débilmente acoplados

**Ventajas:**

- Bajo consumo de recursos
- Las instancias del proceso son creadas explícitamente

- El cliente no necesita información de enrutamiento a partir de la URI inicial
- Los clientes pueden tener una interfaz "*listener*" (escuchadora) genérica para las notificaciones
- Generalmente fácil de construir y adoptar

### 1.7 Lenguajes y herramientas de desarrollo

Para el desarrollo del servicio *RESTful* se hace imprescindible contar con un conjunto de herramientas y tecnologías que complementen su progreso, así como garantizar la calidad y eficiencia del producto final. A continuación se exponen cada una de estas tecnologías y las características y funcionalidades de las mismas.

**Python v3.5:** *Python* es un lenguaje de programación multiplataforma, poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. Es un lenguaje interpretado, lo cual puede ahorrar mucho tiempo durante el desarrollo ya que no es necesario compilar ni enlazar. El intérprete de *Python* es de código abierto y la extensa biblioteca estándar estan a libre disposición en forma binaria y de código fuente para las principales plataformas desde el sitio web de *Python*, y puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de terceros, programas, herramientas y documentación adicional [34]. Es el lenguaje en que está basada la CMT seleccionada, *Ansible* y cuenta con módulos y paquetes para distintos elementos configurables en el repositorio para *Python*. La versión de *Python* escogida para el desarrollo es la v3.5 pues en esta se agregan nuevos módulos de bibliotecas, mejoras de rendimiento y seguridad [35].

**PyCharm v2017.3:** *PyCharm* es un IDE (Entorno de desarrollo integrado) desarrollado por la compañía *Jetbrains* utilizado específicamente para programar en el lenguaje *Python*. Cuenta con una versión gratuita llamada "*Community Edition*" liberada bajo la licencia Apache que presenta entre sus características: autocompletado, depuración de código gráfico, refactorización, inspección de código, integración de control de versiones, ejecución de *tests*, soporte para el desarrollo web con Django además de contar con un editor inteligente y una consola *python* integrada. La versión de *PyCharm* seleccionada es la v2017.3 pues incluye mejoras para reducir el tiempo de espera del proceso de indexación, también indexar el código *JavaScript* es más rápido; mayor velocidad y facilidad de configuración del intérprete remoto SSH e incluye un cliente *REST* construido para *PyCharm* 2017.3 que permite enviar una solicitud compleja para probar su funcionalidad [36].

**Django REST Framework v3.8.2:** Es una aplicación *Django* que contiene un conjunto de herramientas potentes y flexibles para crear API (*Application Programming Interface*) web. Permite construir proyectos de software bajo la arquitectura REST (*Representational State Transfer*), incluye gran cantidad de código para reutilizar (*Views*, *Resources*, etc.) y una interfaz administrativa desde la cual es posible realizar pruebas sobre las operaciones HTTP como lo son: *POST y GET, PUT y DELETE*. El uso del marco *REST* ofrece ventajas para sus desarrolladores, permite flujos simples de autorización al incluir los estándares abiertos OAuth1a y OAuth2, admite fuentes de datos ORM y no ORM, amplia documentación y apoyo de la comunidad. Se selecciona *Django REST Framework* v3.7.2 como *framework* para la capa *Back-end* por ser la más actual, por las correcciones realizadas y funcionalidades incorporadas [37].

**Swagger v2.1.2:** *Swagger* es una herramienta para describir, producir, consumir, visualizar y documentar *APIs RESTful*. Especifica la lista de recursos disponibles en la *API REST* y los métodos (*get, put, post, delete*) usados para llamar a estos recursos, también muestra la lista de parámetros de una operación, si los parámetros son necesarios u opcionales, e información sobre los valores aceptables para estos parámetros. Además, de incluir un esquema *JSON* adicional que describe la estructura del cuerpo de la solicitud que se envía a una operación en la *API REST*, y el esquema *JSON* que describe la estructura de los cuerpos de respuesta devueltos de una operación. Para documentar la *API RESTful* se empleará *Swagger* v2.0 por ser la última versión estable identificada [38].

**MySQL v5.7:** *MySQL* es considerado como el sistema de gestión de bases de datos (*DBMS, Database Management System*) de código abierto más popular del mundo. Desarrollado bajo la licencia GPL es un sistema relacional, multihilo y multiusuario; ofrece bajo costo en requerimientos, velocidad en las operaciones, robustez, gran cantidad de tipos de datos para las columnas, así como flexibilidad y seguridad en su sistema de contraseñas y gestión de usuarios. Para la gestión de la información contenida en la base de datos de la solución a implementar se empleará *MySQL* v5.7 por ser la última versión estable identificada [39].

**Git v2.1:** *Git* es un sistema de control de versiones distribuido (*DVCS, Distributed Version Control System*), de código abierto que permite revertir archivos o el proyecto completo a un estado anterior, comparar cambios a lo largo del tiempo, recuperar archivos perdidos o dañados y más. Ofrece seguridad en caso de fallo o pérdida de los repositorios principales pues todos los nodos manejan la información en su totalidad y por lo tanto pueden actuar de cliente o servidor en cualquier momento. Cada usuario genera sus propios directorios dentro del árbol para ser cargados posteriormente al repositorio central por lo que

no necesita de permisos especiales de lectura y escritura. Permite trabajar cuando y donde se estime conveniente pues no depende de una conexión de red permanente ya que solo es preciso para la sincronización del proyecto Para salvaguardar los cambios y versiones de la solución a implementar se utilizará *Git* v2.1 por ser la última versión estable identificada [40].

**Visual Paradigm v8.0:** *Visual Paradigm* es una herramienta de Ingeniería de Software Asistida por Computadora (*CASE*, *Computer Aided Software Engineering*), orientada al diseño de software y adaptada para proyectos de software ágil. Proporciona asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Agiliza la construcción de aplicaciones con calidad a un menor coste. Posibilita la generación de bases de datos, transformación de diagramas de Entidad-Relación en tablas de base de datos, así como ingeniería inversa de bases de datos [41], [42]. Sin omitir lo antes expuesto, se selecciona esta herramienta por su característica de ser libre, poseer excelente documentación y por tener dominio sobre la misma; se propone la versión 8.0 por ser la más actualizada y estable disponible.

**UML v2.0:** El lenguaje unificado de modelado (UML) es un un lenguaje de modelado visual común, empleado para diseñar, graficar, generar y documentar artefactos en sistemas de software complejos. Utilizado para el modelado de los elementos arquitectónicos del desarrollo de software, facilita comunicar la estructura de un sistema complejo, especificar el comportamiento deseado del sistema, comprender mejor lo que se está construyendo y descubrir oportunidades de simplificación y reutilización. Contiene un conjunto de notaciones específicas para la programación orientada a objetos y ofrece una amplia variedad de diagramas para visualizar el sistema desde varias perspectiva y representar las distintas etapas del desarrollo de software en un proyecto. Se utilizará la versión 2.0 por su facilidad de uso con respecto a la versión 2.5 y porque contiene todos lo diagramas necesarios para modelar el negocio [43].

### 1.8 Metodología de desarrollo de software

Una metodología de desarrollo de software es un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de software [44]. Estas son indispensables para crear o actualizar software de calidad que cumpla con los requisitos de los usuarios; son una parte fundamental

de la Ingeniería de software orientada al estudio y determinación del método más adecuado para dar incremento al producto de software [45].

### 1.8.1 Metodología de desarrollo AUP-UCI

Para desarrollar una solución acorde al estándar definido en la Universidad de las Ciencias Informáticas (UCI) y mantener la homogeneidad en los procesos de desarrollo de software de la institución se selecciona la metodología híbrida de desarrollo AUP-UCI. Basada en una variación de la metodología Proceso Unificado Ágil (AUP, del inglés *Agile Unified Process*) como medio para controlar el ciclo de vida de desarrollo del software. AUP-UCI ejecuta de forma cronológica tres fases; la fase de inicio donde se lleva a cabo las actividades relacionadas con la planeación del proyecto; la fase de ejecución donde se ejecutan las actividades requeridas para desarrollar el software; y la fase de cierre donde se analizan tanto los resultados del proyecto como su ejecución.

En el presente trabajo de diploma se desarrolla la fase de ejecución y se transita por las siguientes disciplinas de la metodología: Requisitos, Análisis y diseño, Implementación, Pruebas internas y de Aceptación. AUP-UCI propone cuatro posibles escenarios, de estos se ajusta a la propuesta de solución el escenario No. 4; aplicable a proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido, donde el cliente estará siempre con el equipo de desarrollo para convenir detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU (historia de usuario) no debe poseer demasiada información.

## 1.9 Consideraciones finales

En este capítulo fueron estudiados diferentes sistemas de administración de la configuración; posterior a su estudio se concluye que no constituyen una solución viable, y se enfoca la investigación al análisis y comparación de las CMT que emplean estos sistemas. La aplicación de la metodología QSOS permitió fundamentar la selección de *Ansible* como la herramienta de administración de la configuración a emplear en el desarrollo de la solución. Se seleccionó un grupo de tecnologías libres siguiendo así las normas del proceso de migración; y se definió el uso de la metodología híbrida AUP-UCI como medio para controlar el ciclo de vida de desarrollo del software. Con los resultados de la investigación y definidas las herramientas y metodología a usar, quedan creadas las bases teóricas para proceder al diseño y modelado de la propuesta de solución.

## Capítulo 2: Servicio *RESTful* para la administración centralizada de la distribución cubana GNU/Linux Nova

En el presente capítulo se expone la propuesta de solución y los requisitos del servicio *RESTful* que respaldan dicha propuesta. Se describen las historias de usuario detallando cada campo y describiendo la validación del mismo. Se diseñan los prototipos de interfaz de usuario, diagramas de componentes, de clases y entidad-relación, se describen los patrones de diseño y arquitectura a emplear. Todo el análisis y diseño de la solución propuesta parte de la ruta marcada por la metodología AUP-UCI con el único fin de desplegar un producto con la calidad demandada aplicando los principios que favorecen el desarrollo ágil del proceso.

### 2.1 Propuesta de solución

Partiendo de los resultados definidos hasta este punto de la investigación, para dar solución al problema planteado se propone el desarrollo del servicio *RESTful* Nova Brouwer que permitirá administrar la configuración de las estaciones de trabajo que utilicen la distribución cubana GNU/Linux Nova empleando *Ansible* como herramienta de administración de la configuración. A continuación, se modela el funcionamiento del proceso de administración centralizada de estaciones de trabajo para facilitar la comprensión del negocio, ver Imagen 2.

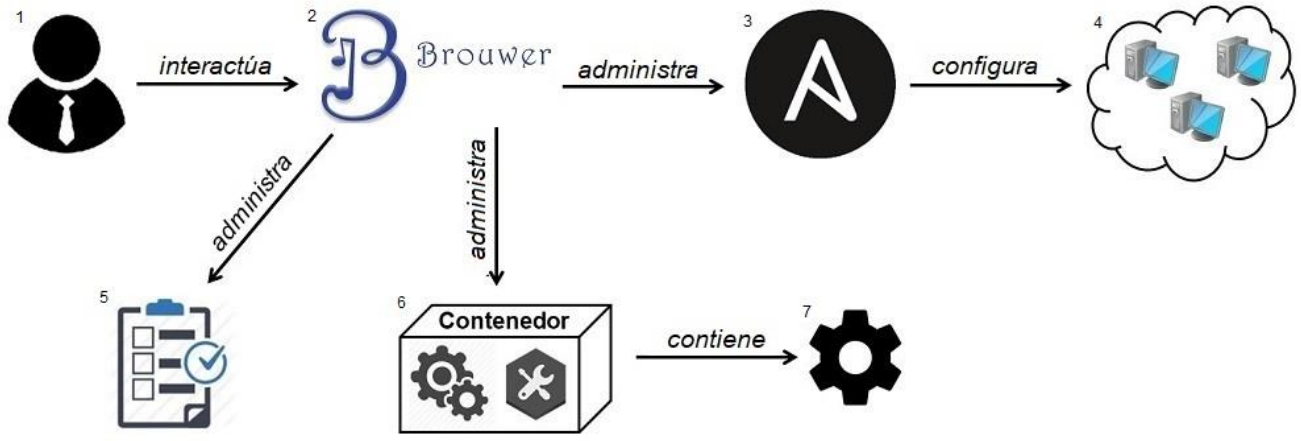


**Imagen 2.** Proceso de administración centralizada de estaciones de trabajo desde Nova Brouwer
(Fuente: elaboración propia)

**Conceptos**

1. **Usuario:** administrador, especialista de migración o persona con los permisos requeridos para poder usar la herramienta.
2. **Herramienta Nova Brouwer:** herramienta centralizada de administración de la configuración de estaciones de trabajo que recurre a la gestión de contenedores de configuraciones y su ejecución.
3. **Ansible:** herramienta de administración de la configuración.
4. **Grupo de estaciones de trabajo:** cantidad definida de máquinas conectadas a una red y destinadas a una labor profesional, técnica o científica.
5. **Inventario:** grupos lógicos que contienen los hosts de las estaciones de trabajo a administrar
6. **Contenedor de configuraciones:** destinado a almacenar configuraciones para su posterior ejecución con el objetivo de modificar las configuraciones de las estaciones de trabajo.
7. **Configuraciones:** serie de datos que establecen el valor de ciertas variables de un software que suelen estar predefinidas, pero pueden ser modificadas.

La solución propuesta bajo la arquitectura cliente-servidor modificará de forma remota un grupo de configuraciones propias de la distribución cubana mediante la administración de múltiples estaciones de trabajo, facilitando la instalación, desinstalación y actualización de paquetes de forma paralela como también habilitar e inhabilitar el cortafuegos, desplegar reglas de seguridad, modificar la apariencia y programar tareas. El servicio *RESTful* permitirá seleccionar los contenedores de configuraciones a desplegar y el inventario al cual establecer las configuraciones. El especialista de migración en lugar de configurar cada estación de trabajo individualmente y de forma manual, tan solo debe indicar desde la máquina servidor las modificaciones en la configuración a ejecutar en las estaciones de trabajo usando el protocolo SSH como medio de transferencia de los módulos.

Definido el funcionamiento del proceso de administración centralizada de estaciones de trabajo, ilustrado en la Imagen 2 se procede a definir los requisitos del servicio *RESTful* Nova Brouwer.

### 2.2 Requisitos

Los requisitos de un sistema describen los servicios que ha de ofrecer el sistema y las restricciones asociadas a su funcionamiento [46], estos aportan valor y utilidad al cliente y a los usuarios finales. Se clasifican en requisitos funcionales (RF) o requisitos no funcionales (RNF).

### 2.2.1 Fuentes de obtención de requisitos

Las fuentes para la obtención de requisitos utilizadas fueron las siguientes:

- Análisis del sistema de administración de la configuración *Ansible Tower* (Ver epígrafe 1.3)
- Especialistas de CESOL

### 2.2.2 Técnicas para identificar requisitos

Las técnicas de identificación de requisitos de software corresponden a mecanismos que se utilizan para recolectar información con el objetivo de identificar las necesidades del negocio; las empleadas fueron:

**Entrevista**

La entrevista es la forma de obtener información directamente con el cliente acerca del sistema de software a desarrollar. A partir de un grupo predefinido de preguntas hechas por el desarrollador, el cliente describe las necesidades del sistema como respuesta a estas. Esta técnica permitió obtener una comprensión general de los requerimientos del sistema y cómo podría dicho sistema interactuar con los *stakeholders* [47]. La entrevista (Ver Anexo 2) fue realizada al compañero jefe de departamento y desarrollador principal de GNU/Linux Nova Ing. Juan Manuel Fuentes Rodríguez.

**Desarrollo de prototipos**

El desarrollo de prototipos permite que el sistema, o algunas partes, se construyan rápidamente para aclarar ciertos aspectos en los que se aseguren que el desarrollador, el usuario y el cliente estén de acuerdo en lo que se necesita así como también la solución que se propone para dicha necesidad y de esta forma minimizar el riesgo y la incertidumbre en el desarrollo, este modelo se encarga del desarrollo de diseños para que estos sean analizados y prescindir de ellos a medida que se adhieran nuevas especificaciones, es ideal para medir el alcance del producto, pero no se asegura su uso real [48].

### 2.2.3 Requisitos funcionales

Para desarrollar la solución propuesta se implementarán un conjunto de funcionalidades que el servicio *RESTful* debe cumplir en correspondencia a los requisitos planteados por el cliente. La complejidad de los requisitos se obtuvo mediante el producto de trabajo “Evaluación de requisitos”, propuesto por la metodología en el expediente de proyecto definido en la universidad. A continuación, se listan y describen los requisitos funcionales para el servicio *RESTful* de administración centralizada de la distribución cubana GNU/Linux Nova.

**Tabla 3.** Requisitos funcionales

(Fuente: elaboración propia)

| ID | Nombre del requisito | Descripción del requisito | Prioridad | Complejidad |
|---|---|---|---|---|
| **RF1** | Autenticar usuario | El servicio debe permitir autenticar al usuario haciendo uso de contraseña | Alta | Media |
| **RF2** | Insertar usuario | El servicio debe permitir insertar un nuevo usuario a la base de datos | Alta | Baja |
| **RF3** | Eliminar usuario | El servicio debe permitir eliminar un usuario de la base de datos | Alta | Baja |
| **RF4** | Modificar usuario | El servicio debe permitir modificar datos de un usuario de la base de datos | Alta | Baja |
| **RF5** | Mostrar usuario | El servicio debe permitir mostrar los usuarios de la base de datos | Alta | Baja |
| **RF6** | Insertar host | El servicio debe permitir insertar un nuevo *host* a la base de datos | Alta | Baja |
| **RF7** | Eliminar host | El servicio debe permitir eliminar un *host* de la base de datos | Alta | Baja |
| **RF8** | Modificar host | El servicio debe permitir modificar datos de un *host* de la base de datos | Alta | Baja |
| **RF9** | Mostrar hosts | El servicio debe permitir mostrar los *hosts* de la base de datos | Alta | Baja |
| **RF10** | Insertar grupo | El servicio debe permitir insertar un nuevo grupo de *hosts* en la base de datos | Alta | Baja |
| **RF11** | Eliminar grupo | El servicio debe permitir eliminar un grupo de *hosts* de la base de datos | Alta | Baja |
| **RF12** | Modificar grupo | El servicio debe permitir modificar los datos un grupo de *hosts* de la base de datos | Alta | Baja |
| **RF13** | Mostrar grupos | El servicio debe permitir mostrar los grupos de hosts de la base de datos | Alta | Baja |
| **RF14** | Asignar hosts a grupo | El servicio debe permitir asignar *hosts* a un | Alta | Baja |

| | | | | |
|---|---|---|---|---|
| | | grupo | | |
| **RF15** | Eliminar hosts del grupo | El servicio debe permitir remover *hosts* de un grupo | Alta | Baja |
| **RF16** | Insertar configuración | El servicio debe permitir insertar una configuración a la base de datos | Alta | Media |
| **RF17** | Eliminar configuración | El servicio debe permitir eliminar una configuración de la base de datos | Alta | Media |
| **RF18** | Modificar configuración | El servicio debe permitir modificar los datos de una configuración de la base de datos | Alta | Media |
| **RF19** | Mostrar configuraciones | El servicio debe permitir mostrar las configuraciones de la base de datos | Alta | Media |
| **RF20** | Insertar contenedor de configuraciones | El servicio debe permitir insertar un contenedor de configuraciones a la base de datos | Alta | Media |
| **RF21** | Eliminar contenedor de configuraciones | El servicio debe permitir eliminar un contenedor de configuraciones de la base de datos | Alta | Media |
| **RF22** | Modificar contenedor de configuraciones | El servicio debe permitir modificar los datos de un contenedor de configuraciones de la base de datos | Alta | Media |
| **RF23** | Mostrar contenedor de configuraciones | El servicio debe permitir mostrar los contenedores de configuraciones de la base de datos | Alta | Media |
| **RF24** | Asignar contenedor de configuraciones a grupos | El servicio debe permitir asignar un contenedor de configuraciones a grupos de hosts | Alta | Media |
| **RF25** | Remover contenedor de configuraciones de grupos | El servicio debe permitir remover un contenedor de configuraciones de un grupo de hosts | Alta | Media |

| | | | | |
|---|---|---|---|---|
| **RF26** | Asignar configuraciones a un contenedor de configuraciones | El servicio debe permitir asignar configuraciones a un contenedor de configuraciones | Alta | Media |
| **RF27** | Remover configuraciones de un contenedor de configuraciones | El servicio debe permitir remover configuraciones de un contenedor de configuraciones | Alta | Media |
| **RF28** | Habilitar contenedor | El servicio debe permitir habilitar un contenedor de configuraciones | Alta | Baja |
| **RF29** | Deshabilitar contenedor | El servicio debe permitir deshabilitar un contenedor de configuraciones | Alta | Baja |
| **RF30** | Ejecutar contenedor de configuraciones | El servicio debe permitir ejecutar un contenedor de configuraciones | Alta | Media |
| **RF31** | Modificar configuraciones de instalación de paquetes de software | El servicio debe permitir utilizando el protocolo SSH la instalación de paquetes de software para grupos definidos | Alta | Alta |
| **RF32** | Modificar configuraciones de desinstalación de paquetes de software | El servicio debe permitir utilizando el protocolo SSH la desinstalación de paquetes de software para grupos definidos | Alta | Alta |
| **RF33** | Modificar configuraciones de actualización de paquetes de software | El servicio debe permitir utilizando el protocolo SSH la actualización de paquetes de software para grupos definidos | Alta | Alta |
| **RF34** | Modificar configuraciones de purgar paquetes de software | El servicio debe permitir utilizando el protocolo SSH purgar de paquetes de software para grupos definidos | Alta | Alta |
| **RF35** | Modificar configuración | El servicio debe permitir modificar las | Alta | Alta |

| | | | | |
|---|---|---|---|---|
| | del cortafuegos | configuraciones del cortafuegos para grupos definidos | | |
| **RF36** | Modificar configuración de proxy de la red | El servicio debe permitir modificar la configuración de proxy de la red para grupos definidos | Alta | Alta |
| **RF37** | Modificar configuración de apagado automático | El servicio debe permitir modificar configuración de apagado automático para grupos definidos | Baja | Media |
| **RF38** | Modificar configuración de reinicio automático | El servicio debe permitir modificar configuración de reinicio automático para grupos definidos | Baja | Media |
| **RF39** | Modificar configuración de volumen de salida | El servicio debe permitir modificar configuración de volumen de salida para grupos definidos | Media | Media |
| **RF40** | Modificar configuración de fondo de escritorio | El servicio debe permitir modificar configuración de fondo de escritorio para grupos definidos | Alta | Alta |
| **RF41** | Modificar configuración de aplicaciones de búsqueda | El servicio debe permitir modificar configuración de las aplicaciones de búsqueda | Media | Alta |
| **RF42** | Modificar configuración de zona horaria automática | El servicio debe permitir modificar configuración de zona horaria automática para grupos definidos | Media | Alta |
| **RF43** | Modificar configuración de fecha y hora automática | El servicio debe permitir modificar configuración de fecha y hora automática para grupos definidos | Media | Alta |
| **RF44** | Modificar configuración de zona horaria | El servicio debe permitir modificar configuración de zona horaria para grupos definidos | Media | Alta |
| **RF45** | Modificar configuración | El servicio debe permitir modificar | Media | Media |

| | | | | |
|---|---|---|---|---|
| | de fecha y hora | configuración de fecha y hora para grupos definidos | | |
| **RF46** | Modificar configuración de ejecutar script | El servicio debe permitir modificar configuración de ejecutar script para grupos definidos | Alta | Alta |

### 2.2.4 Requisitos no funcionales

Para aumentar el rendimiento y funcionalidad de la propuesta de solución se detallan propiedades y cualidades no funcionales que debe poseer el producto de software. Las categorías de los requisitos fueron definidas mediante el producto de trabajo “Evaluación de requisitos”, propuesto por la metodología en el expediente de proyecto definido en la universidad. A continuación, se listan y describen los requisitos no funcionales para el servicio *RESTful* de administración centralizada de la distribución cubana GNU/Linux Nova.

**Tabla 4.** Requisitos no funcionales

(Fuente: elaboración propia)

| ID | Categoría | Descripción |
|---|---|---|
| **RNF1** | Funcionalidad | El servicio debe facilitar al usuario el acceso a todas las operaciones |
| **RNF2** | Funcionalidad | El servicio debe ser capaz de validar los datos introducidos y mostrar mensajes de error en caso de que los datos sean inválidos |
| **RNF3** | Seguridad | Disponible solo para personal autorizado: el acceso al servicio debe ser de uso restringido evitando cambios no deseados por personal no autorizado |
| **RNF4** | Mantenibilidad | El servicio deberá permitir posteriores actualizaciones y se debe garantizar el mantenimiento de la misma |
| **RNF5** | Interoperabilidad | El servicio tiene que ser funcional para la plataforma GNU/Linux Nova y específicamente para la versión Nova Servidores v6.0 |
| **RNF6** | Interoperabilidad | El servicio debe emplear *Nginx* como servidor web |
| **RNF7** | Interoperabilidad | El servicio debe emplear *MySQL* como servidor de base de datos |
| **RNF8** | Interoperabilidad | El servicio debe utilizar *Ansible* v2.5 |
| **RNF9** | Interoperabilidad | El servicio debe utilizar *Python* v3.5 como lenguaje de programación |
| **RNF10** | Portabilidad | La PC destinada a desempeñarse como servidor debe tener como mínimo |

| | | |
|---|---|---|
| | | 4 GB de memoria RAM y disponible 20 GB de HDD |

### 2.2.5 Validación de requisitos de software

La validación de requisitos tiene como objetivo tratar de mostrar que estos realmente definen el sistema que el cliente desea. Coincide parcialmente con el análisis ya que este implica encontrar problemas con los requisitos pues un cambio en estos normalmente significa que el diseño y la implementación del sistema también deben cambiar lo que puede conducir a pérdida de tiempo y al trabajo repetitivo [47]. A continuación, se describen las técnicas de validación de requisitos empleadas.

**Revisiones de requerimientos:** Una revisión de requerimientos es un proceso manual, informal o formal que involucra tanto a clientes como desarrollador. Los requerimientos son analizados sistemáticamente por un equipo de revisores. Ellos verifican el documento de requerimientos en cuanto a anomalías y omisiones. Los conflictos, contradicciones, errores y omisiones en los requerimientos deben ser señalados por los revisores y registrarse formalmente [47].

**Construcción de prototipos:** La construcción de prototipos permite mostrar un modelo ejecutable del sistema a los usuarios finales y a los clientes. Estos pueden experimentar con este modelo para ver si cumple sus necesidades reales [47].

### 2.2.6 Historias de usuario (HU)

Las historias de usuario constituyen una práctica común utilizada para representar los requisitos del software e introducirlos en el proceso de desarrollo. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [49]. A continuación, se muestran dos de las historias de usuario de complejidad Alta definidas para la elaboración de la solución propuesta.

**Tabla 5.** HU_9 Requisito Mostrar host

(Fuente: elaboración propia)

| **Número:** HU_9 | **Nombre del requisito:** Mostrar *hosts* |
|---|---|
| **Programador:** Carlos Rubén García Pérez | **Iteración Asignada:** Primera iteración |
| **Prioridad:** Alta | **Tiempo Estimado:** 10 horas |

| | |
|---|---|
| **Riesgo en Desarrollo:** Medio | **Tiempo Real:** 10 horas |
| **Descripción:** El servicio permitirá mostrar los *hosts* almacenados en la base de datos mediante el método *GET* perteneciente al recurso *host*. | |
| **Observaciones:**<br>• El usuario debe estar autenticado.<br>• La *URL (Uniform Resource Locator)* para acceder a la funcionalidad está compuesta por la dirección *IP* donde está público el servicio, el puerto de acceso y el *enpoint api/hosts* para mostrar todos los hosts o *api/hosts/{host_id}* para mostrar un *host* en específico. | |

**Tabla 6.** HU_22 Requisito Modificar contenedor de configuraciones

| | |
|---|---|
| **Número:** HU_22 | **Nombre del requisito:** Modificar contenedor de configuraciones |
| **Programador:** Carlos Rubén García Pérez | **Iteración Asignada:** Primera iteración |
| **Prioridad:** Alta | **Tiempo Estimado:** 10 horas |
| **Riesgo en Desarrollo:** Medio | **Tiempo Real:** 10 horas |
| **Descripción:** El servicio permitirá modificar un contenedor de configuración almacenado en la base de datos mediante el método *PUT* perteneciente al recurso *container*. | |
| **Observaciones:**<br>• El usuario debe estar autenticado.<br>• El usuario debe ingresar el id del contenedor a modificar.<br>• La *URL* para acceder a la funcionalidad está compuesta por la dirección *IP* donde está público el servicio, el puerto de acceso y el *enpoint api/containers/{container_id}.* | |

### 2.3 Análisis y diseño

En esta disciplina, si se considera necesario, los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema (incluyendo su arquitectura). Además, en esta disciplina se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales [50]. En el presente epígrafe se definen elementos del análisis y el diseño considerados para implementar la solución propuesta.

#### 2.3.1 Arquitectura del sistema

Se define como arquitectura del servicio *RESTful* la presentación desacoplada (ver Imagen 3), que establece cómo debe realizarse el manejo de las acciones del usuario, la manipulación de la *interfaz* y los datos de la aplicación. Este estilo separa los componentes de la interfaz del flujo de datos para el manejo de la interacción de la representación de los datos con que trabaja el usuario. Se caracteriza por la descomposición funcional de las aplicaciones, componentes de servicio y su despliegue distribuido, que ofrece mejor escalabilidad, disponibilidad, rendimiento, manejabilidad y uso de recursos [51].

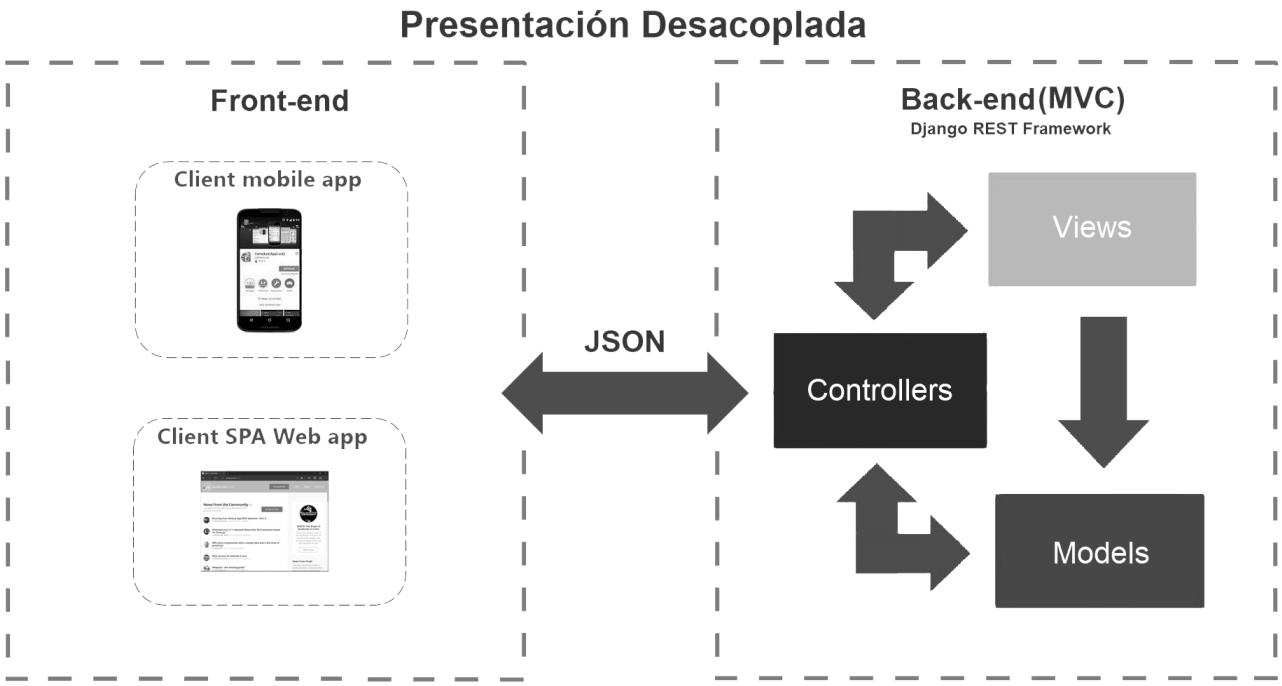


**Imagen 3.** Arquitectura del sistema

(Fuente: elaboración propia)

### 2.3.2 Estilo arquitectónico para back-end

Para el desarrollo de la solución se emplea el patrón arquitectónico MVC (*Model-View-Controller*), utilizado en el marco de trabajo de *Python Django Rest Framework*. Este patrón divide las partes que conforman una aplicación en tres componentes: el Modelo, las Vistas y los Controladores, permitiendo la implementación por separado de cada uno, garantizando la actualización y mantenimiento del software de forma sencilla y en un reducido espacio de tiempo. El Modelo constituye el objeto que maneja los datos del programa y sus transformaciones. La Vista es el objeto que maneja la presentación visual de los datos representados por el Modelo, esta interactúa preferentemente con el Controlador, y directamente con el Modelo a través de una referencia al mismo. El Controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo; cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. [52]. A continuación, se muestra el diagrama de clases del diseño correspondiente a las historias de usuario HU_10, HU_11, HU_12 y HU_13*,* ver Imagen 4*.*

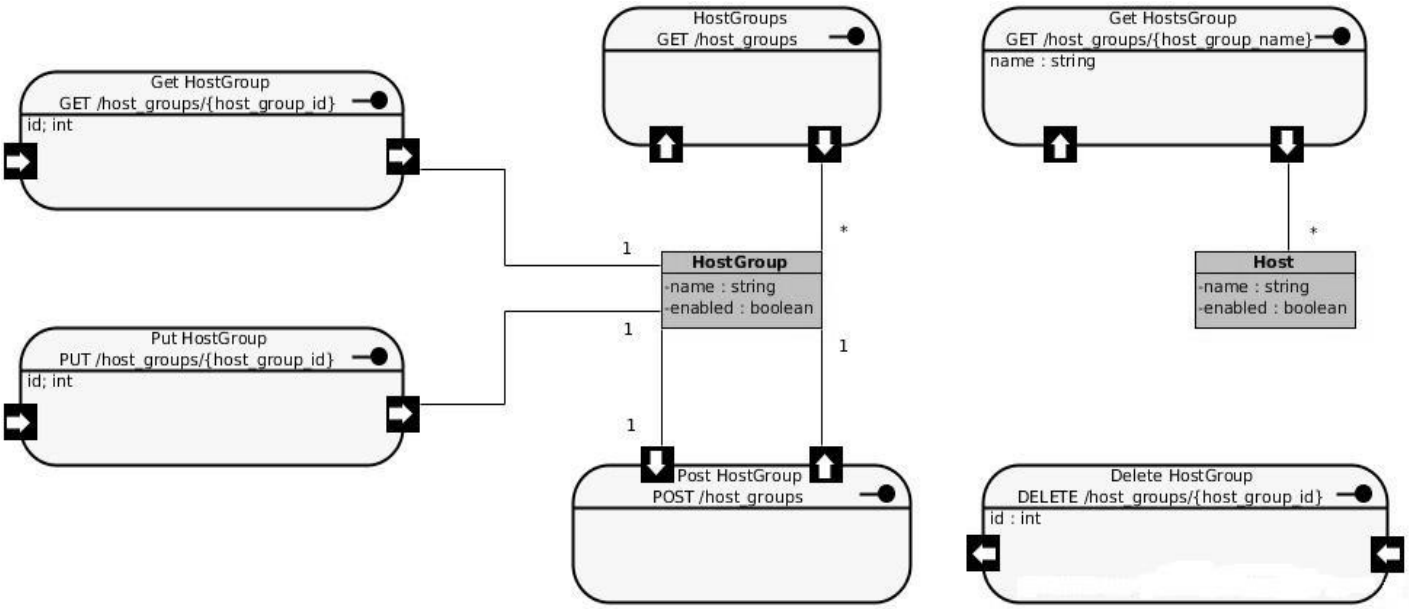


**Imagen 4.** Diagrama de clases del diseño correspondiente a las HU (12, 13, 14, 15) para *Back*-end
(Fuente: elaboración propia)

### 2.3.3 Modelo de los datos

Un modelo de datos es una colección de herramientas conceptuales para describir los datos, las relaciones, la semántica y las restricciones de consistencia. Para ilustrar el concepto de un modelo de datos, se describen dos modelos, el modelo entidad-relación y el modelo relacional, en este apartado se modelaron los datos haciendo uso del modelo entidad-relación [53]. A continuación, se muestra el modelo de datos para la propuesta de solución, ver Imagen 5.

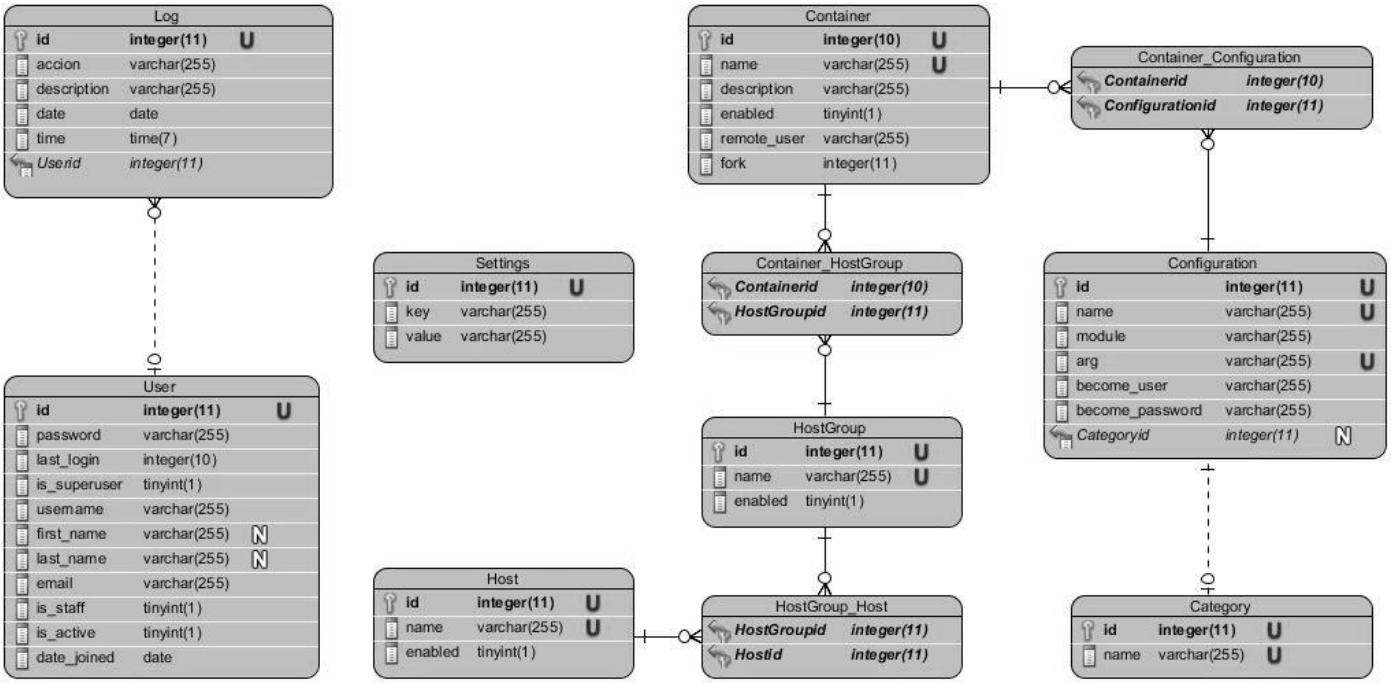


**Imagen 5.** Modelo Entidad-Relación

(Fuente: elaboración propia)

### 2.3.4 Modelo de clases UML

Los diagramas de clases permiten modelar y describir los tipos de objetos de un sistema, así como los distintos tipos de relaciones que pueden existir entre ellos; se consideran la técnica más potente para el modelado conceptual de un sistema software, la cual suele recoger los conceptos claves del modelo de objetos subyacente al método orientado a objetos [54]. A continuación, se muestra el diagrama de clases de las entidades para la propuesta de solución, ver Imagen 6.

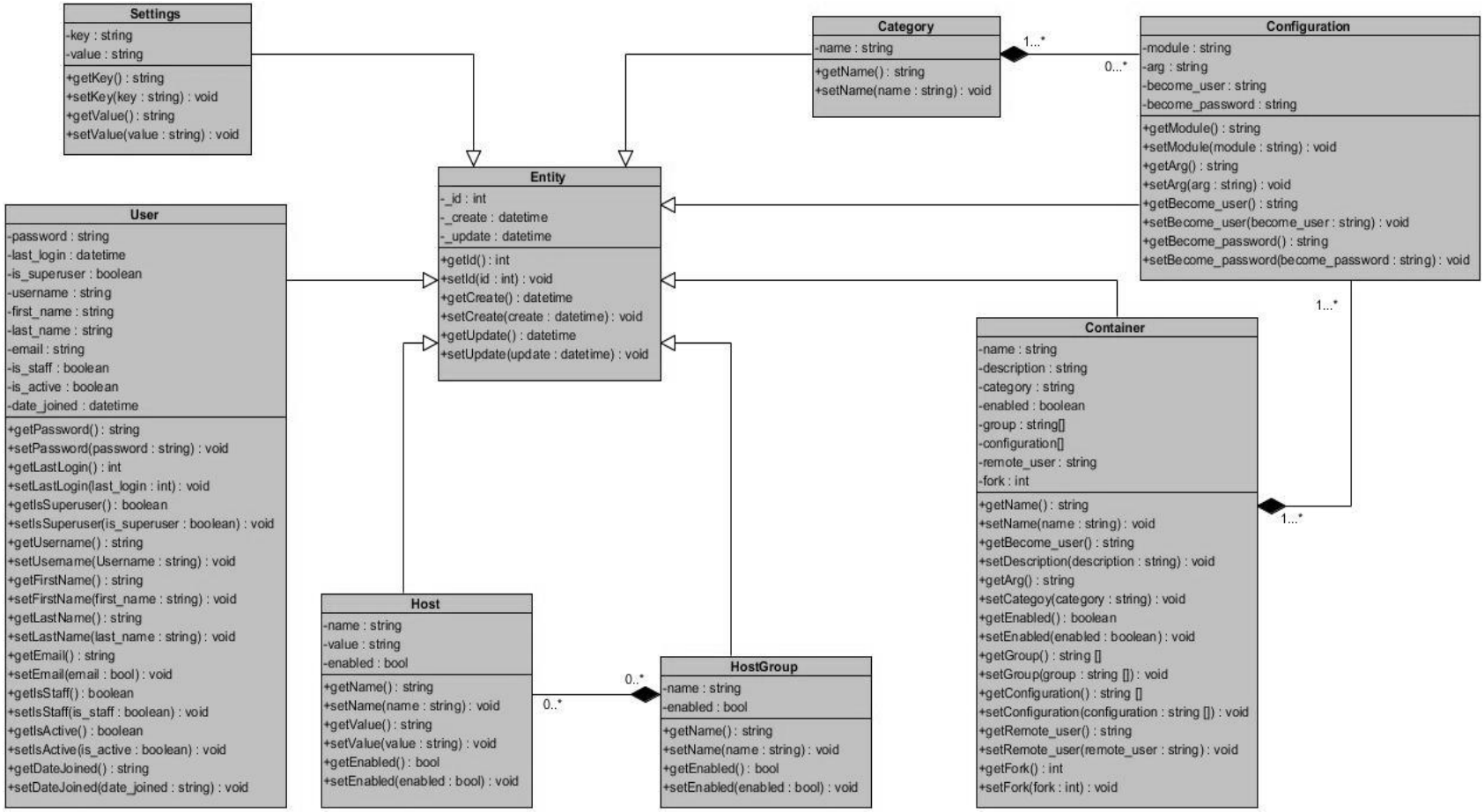


**Imagen 6.** Diagrama de clases

(Fuente: elaboración propia)

### 2.3.5 Patrones de diseño

Un patrón de diseño es una buena práctica documentada de la solución de un problema que ha sido aplicado satisfactoriamente en múltiples entornos. Es una solución recurrente a un problema común observado o descubierto durante el estudio o construcción de numerosas aplicaciones. Su principal objetivo es incrementar la calidad del *software* en términos de reusabilidad, mantenimiento y extensibilidad. En el diseño de la propuesta de solución se utilizaron los patrones GRASP (General *Responsibility Assignment Software Patterns*), que describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones [55].

**Experto:** Es el patrón que define el principio básico de asignación de responsabilidades; indica, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtiene un diseño con mayor cohesión y se mantiene la información encapsulada [55]. El uso de este patrón se evidencia en la plantilla *ansible_services* responsable de ejecutar los *playbook*.

**Creador:** Es el patrón que ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. La nueva instancia deberá ser creada por la clase que: contiene la información necesaria para realizar la creación del objeto, o usa directamente las instancias creadas del objeto, o almacena o maneja varias instancias de la clase [55]. El uso de este patrón se evidencia en la plantilla *serializers* encargada de crear entidades a partir de los modelos y garantizar su persistencia en la base de datos.

**Controlador:** Es el patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, lo que permite aumentar la reutilización de código y tener un mayor control [55]. El uso de este patrón se evidencia en las vistas que contienen las clases *ViewSet* y *Details* encargadas de ejecutar los métodos *get*, *post*, *put* y *delete* dedicados a escuchar o atender los elementos del sistema.

**Alta cohesión:** Este patrón plantea que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase [55]. El uso de este patrón se evidencia en las vistas que contienen las clases *ViewSet* y *Details* dedicadas a agrupar métodos (*get*, *post*, *put* y *delete*) altamente relacionados en clases individuales.

## 2.4 Consideraciones finales

En este capítulo se propuso el desarrollo del servicio *RESTful* Nova Brouwer como propuesta de solución y quedaron planteadas las ventajas que impulsan su desarrollo. Se especificaron las funcionalidades y restricciones de la herramienta mediante la obtención de 46 requisitos funcionales y 10 requisitos no funcionales. Se definió como arquitectura, la presentación desacoplada y el estilo arquitectónico MVC para el *back-end*, así como los patrones de diseño a emplear como buenas prácticas en el desarrollo de la herramienta. En este capítulo se evidenció el tránsito por las disciplinas, Requisitos y Análisis y diseño aplicando de esta forma lo establecido por la metodología.

## Capítulo 3: Implementación y pruebas

En el presente capítulo se desarrolla lo referente a la fase de implementación y pruebas, determinante en el proceso de desarrollo de software y en la validación de su funcionamiento. La propuesta de solución Nova Brouwer debe transitar por un conjunto de pruebas con el objetivo de descubrir y corregir errores, en los siguientes epígrafes se describen las pruebas de software aplicadas. Además, se genera el código fuente en la etapa de implementación y se muestran los resultados arrojados por cada una de las pruebas realizadas, permitiendo evaluar la propuesta de solución.

### 3.1 Implementación

A partir de los resultados obtenidos en la fase de análisis y diseño, se procede a materializar los mismos a código; la complejidad y la duración de esta etapa está íntimamente ligada a las tecnologías y lenguajes de programación utilizados.

#### 3.1.1 Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Los componentes representan todos los elementos de software que entran en la fabricación de aplicaciones informáticas; pueden ser simples archivos, paquetes o bibliotecas cargadas dinámicamente. Las relaciones de dependencia indican que un componente utiliza los servicios ofrecidos por otro componente [47]. A continuación se muestra el diagrama de componentes para la propuesta de solución, ver Imagen 7.

**Descripción de los componentes:**

***settings*:** fichero que contiene las variables de configuración del *framework.*

***urls*:** fichero que contiene la ruta *URL* del proyecto.

***wsgi*:** fichero que actúa como punto de entrada para servir el proyecto cuando sea desplegado al servidor.

***ansible_services*:** fichero que contiene las configuraciones para ejecutar *playbooks.*

***playbooks*:** fichero que construye los *playbooks.*

***callback*:** fichero que contiene la información a mostrar al ejecutar *playbooks.*

***models*:** grupo de ficheros que contiene todas las clases del modelo.

***serializers*:** fichero que contienen las clases con los datos serializados de las instancias del modelo.

***views*:** grupo de ficheros que contiene todas las *views.*

***urls:*** fichero que contiene todas las *URL.*

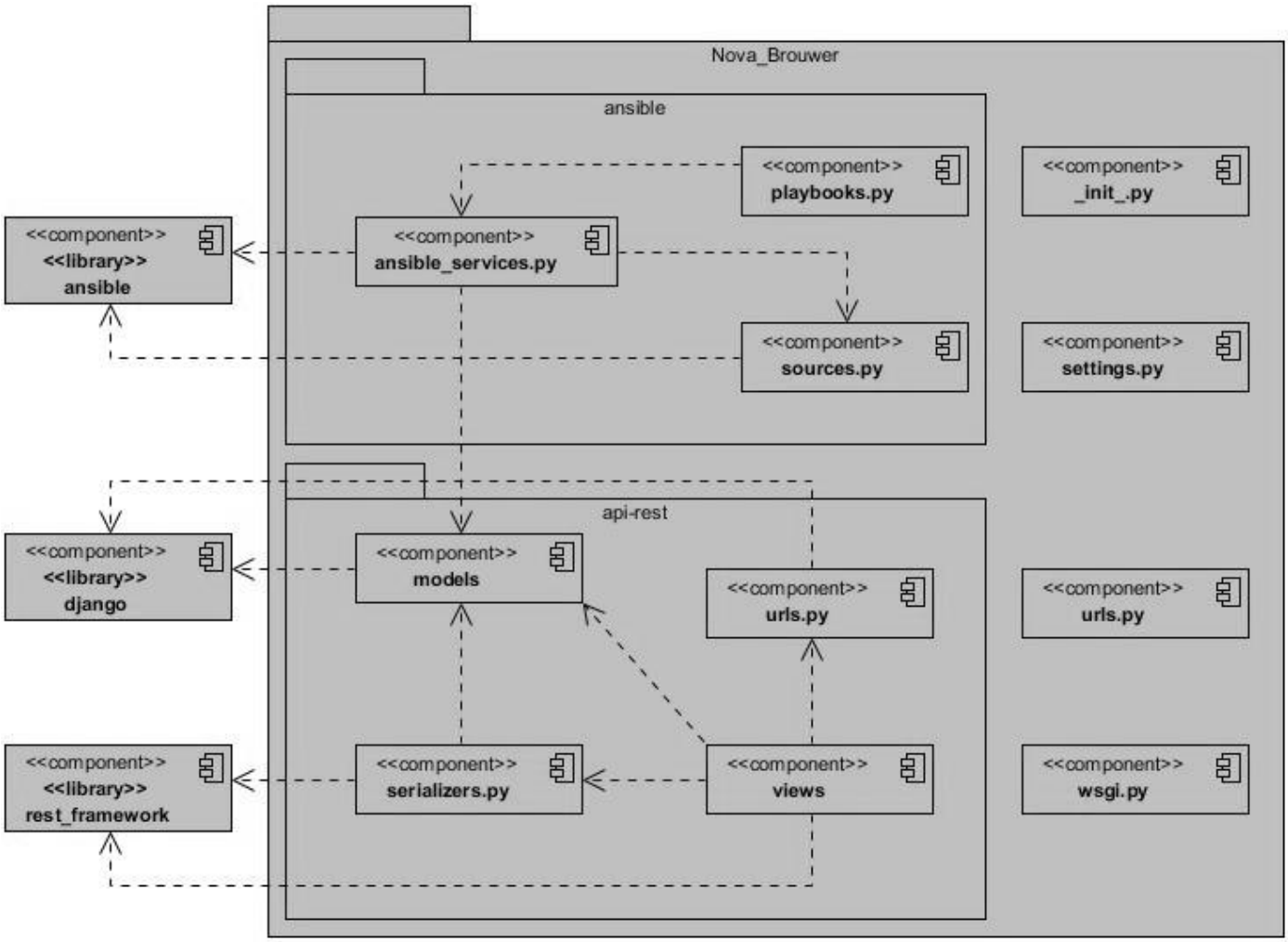


**Imagen 7.** Diagrama de componentes para el *Back-end*

(Fuente: elaboración propia)

### 3.1.2 Diagrama de despliegue

Los diagramas de despliegue permiten modelar la disposición física o topología de un sistema; mostrando las relaciones entre sus componentes y las conexiones físicas entre el hardware. Un diagrama de despliegue está compuesto por: nodos, dispositivos y conectores [47]. A continuación, se muestra el diagrama de despliegue para la propuesta de solución, ver Imagen 8.

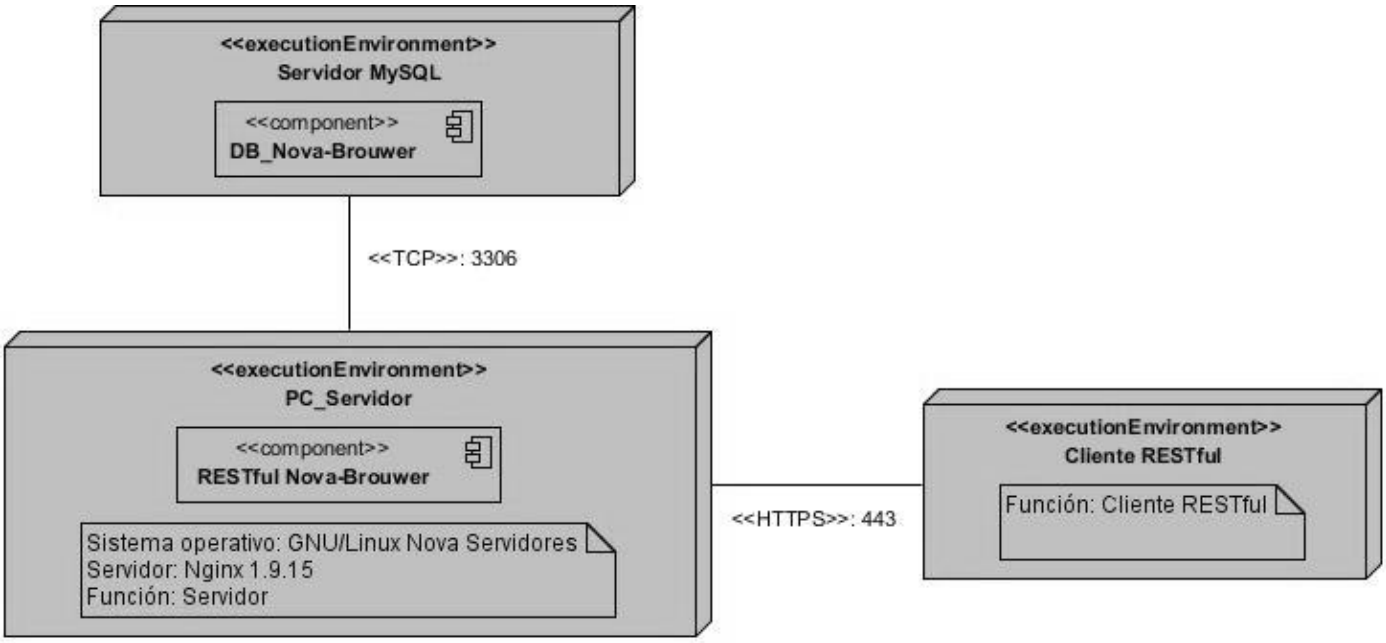


**Imagen 8.** Diagrama de despliegue

(Fuente: elaboración propia)

**Descripción de los nodos:**

**PC_Servidor:** funcionará como PC servidor bajo la distribución GNU/Linux Nova en su variante para servidores y empleará a *Nginx* en su versión 1.9.15 como servidor web; dispondrá del servicio RESTful de administración centralizada Nova Brouwer.

**Cliente RESTful:** referido a PC, laptop o dispositivo móvil que actuará como cliente.

**Servidor MySQL:** contendrá la información almacenada en la base de datos DB_Nova-Brouwer.

### 3.1.3 Estándares de codificación

Los estándares de codificación permiten definir un estilo de programación homogéneo en un proyecto con el objetivo de que todos los participantes lo puedan entender en menor tiempo y que el código en consecuencia sea mantenible [56]. La codificación de Nova Brouwer fue realizada en inglés y siguiendo los estándares de codificación para Python utilizados en la Universidad de las Ciencias Informáticas.

### 3.1.4 Estándares de nomenclatura

**Nomenclatura de las clases**

Se determina que los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación *CamelCase*, lo que posibilita que al leer el nombre de la clase se reconozca el propósito de la misma. Ejemplo: *UserViewSet.*

**Nomenclatura de las variables**

Se determina que los nombres de variables deberán estar escritos en letras minúsculas, en el caso de que sea una palabra compuesta será separada mediante guiones bajos según sea necesario para mejorar la legibilidad. Ejemplo: *user_id.*

**Nomenclatura de las funciones**

Se determina que los nombres de funciones deberán estar escritos en letras minúsculas, en el caso de que sea una palabra compuesta será separada mediante guiones bajos según sea necesario para mejorar la legibilidad. Ejemplo: *get.*

**Otras consideraciones**

- Se determina el uso de 4 espacios por cada nivel de indentación.
- Se determina el uso de espacios entre los operadores aritméticos, relacionales y lógicos.

## 3.2 Pruebas de software

Las pruebas de software son un conjunto de actividades que se realizan con el objetivo de detectar fallos y evaluar las características del software, regulan la ejecución de los proyectos y garantizan la calidad del producto desarrollado. Incluyen técnicas y métodos específicos del diseño de casos de prueba [57]. La importancia de esta fase será mayor o menor según las características del sistema desarrollado, llegando a ser vital en sistemas de tiempo real u otros en los que los errores sean irrecuperables. Las pruebas no tienen el objetivo de prevenir errores sino de detectarlos. Se efectúan sobre el trabajo realizado y se deben encarar con la intención de descubrir la mayor cantidad de errores posible [58]. Para el servicio RESTful Nova Brouwer, desarrollado bajo los procedimientos y técnicas definidos por la metodología hibrida AUP-UCI se establece como escenario de prueba la aplicación de pruebas internas y de aceptación. Se definen como pruebas internas, las pruebas unitarias mediante el método de caja negra

con el objetivo de demostrar que cada funcionalidad es plenamente operacional y pruebas funcionales mediante el método de caja blanca para garantizar el funcionamiento interno del producto [59].

### 3.2.1 Pruebas unitarias

Las pruebas unitarias se concentran en el esfuerzo de verificación de la unidad más pequeña del software, el componente o módulo de software. Tomando como guía de partida la descripción del diseño a nivel de componentes, se prueban importantes caminos de control para descubrir errores dentro de los límites del módulo. Centran su actividad en verificar la funcionalidad y la estructura (lógica interna) de cada elemento individualmente, una vez que ha sido codificado. Las pruebas unitarias se desarrollaron utilizando la técnica de camino básico del método de prueba caja blanca [57].

**Técnica de prueba de camino básico**

Esta técnica permite obtener una medida de la complejidad lógica de un diseño procedimental y que use esta medida como guía para definir un conjunto básico de rutas de ejecución. Los casos de pruebas derivados para ejercitar el conjunto básico deben garantizar que se ejecute cada instrucción por lo menos una vez durante la prueba. Al aplicar esta técnica se utilizó la métrica del software complejidad ciclomática, métrica propuesta por Thomas McCabe en 1976 que proporciona una medida cuantitativa de la complejidad lógica de un programa o procedimiento [57].

La técnica descrita anteriormente fue aplicada al código del software, a continuación, se describe el proceso mediante el código perteneciente al método *put* (actualizar) de la clase *ContainerDetails* correspondiente al requisito funcional 24 con nombre Modificar contenedor de configuraciones, ver Imagen 9.

```
def put(self, request, container_id):#----->1
    if not Container.objects.filter(pk=container_id).exists():#----->2
        return JsonResponse({"message": "Object Not Found"}, status=status.HTTP_404_NOT_FOUND)#----->4
    else:#----->3
        queryset = Container.objects.get(pk=container_id)#----->3
        queryset.name = request.data["name"]#----->3
        queryset.remote_user = request.data["remote_user"]#----->3
        queryset.fork = request.data["fork"]#----->3
        from project.backend.models.host_group import HostGroup#----->3
        l = []#----->3
        for host_group_id in request.data["host_group"]:#----->5
            if not HostGroup.objects.filter(pk=host_group_id).exists():#----->6
                l.append(host_group_id)#----->7
        if len(l) > 0:#----->8
            return JsonResponse(#----->9
                {"message": "HostGroup {} Not Found".format(str(l))},#----->9
                status=status.HTTP_404_NOT_FOUND)#----->9
        queryset.host_group.clear()#----->10
        for host_group_id in request.data["host_group"]:#----->11
            queryset.host_group.add(HostGroup.objects.get(pk=host_group_id))#----->12
        from project.backend.models.configuration import Configuration#----->13
        l2 = []#----->13
        for conf_id in request.data["configuration"]:#----->14
            if not Configuration.objects.filter(pk=conf_id).exists():#----->15
                l2.append(conf_id)#----->16
        if len(l2) > 0:#----->17
            return JsonResponse(#----->18
                {"message": "Configurations {} Not Found".format(str(l))},#----->18
                status=status.HTTP_404_NOT_FOUND)#----->18
        queryset.configuration.clear()#----->19
        for conf_id in request.data["configuration"]:#----->20
            queryset.configuration.add(Configuration.objects.get(pk=conf_id))#----->21
        queryset.save()#----->22
        result = ContainerSerializer(queryset, many=False)#----->22
        return JsonResponse(result.data, status=status.HTTP_200_OK)#----->23
```

**Imagen 9.** Código del método *put*

(Fuente: elaboración propia)

La Imagen 10 muestra el grafo de flujo de control referente al código del procedimiento a analizar:

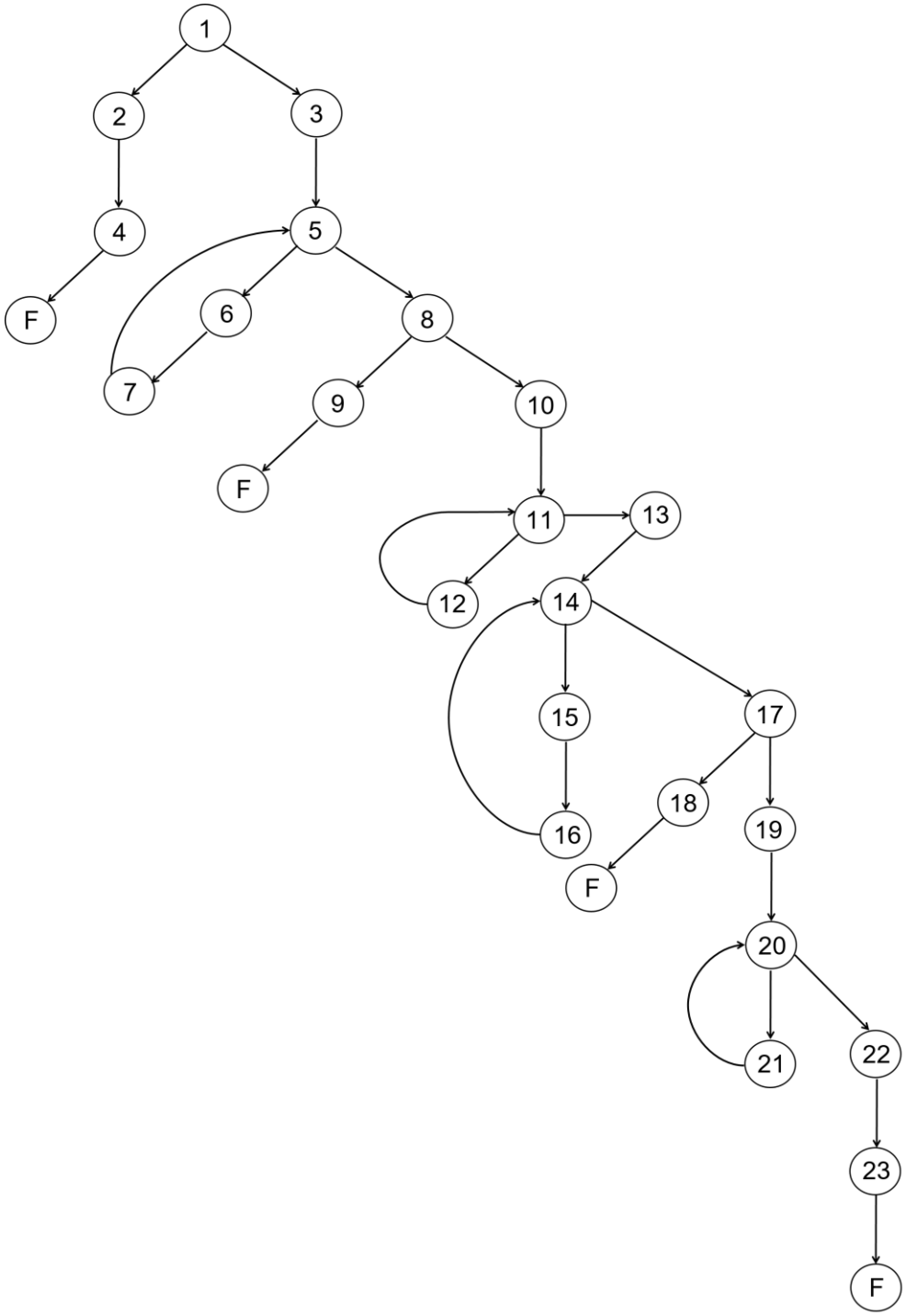


**Imagen 10.** Grafo de flujo de control

(Fuente: elaboración propia)

A continuación, se muestra el resultado del cálculo de la complejidad ciclomática (M), donde E es el número de aristas y N el número de nodos.

M = E - N + 2

M = 30 - 27 + 2

M = 5

Una vez calculada la complejidad ciclomática, se puede determinar el riesgo que supone utilizando los rangos definidos en la siguiente tabla:

**Tabla 7.** Rangos para la evaluación del riesgo

(Fuente: según Thomas McCabe)

| Complejidad Ciclomática | Evaluación del Riesgo |
|---|---|
| 1-10 | Programa simple, sin mucho riesgo |
| 11-20 | Más complejo, riesgo moderado |
| 21-50 | Complejo, programa de alto riesgo |
| 50 | Programa no testeable, muy alto riesgo |

A partir del análisis de muchos proyectos McCabe determinó que un valor 10 es un límite superior práctico para el tamaño de un módulo. Cuando la complejidad supera dicho valor se hace muy difícil probarlo, entenderlo y modificarlo; para el módulo evaluado los resultados arrojados reflejan un programa simple con un mínimo de riesgo.

**Caminos linealmente independientes**

- Camino básico 1: 1; 2; 4; F
- Camino básico 2: 1; 3; 5; 8; 9; F
- Camino básico 3: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 17; 19; 20; 21; 20; 22; 23; F
- Camino básico 4: 1; 3; 5; 8; 10; 11; 12; 11; 13; 14; 17; 19; 20; 21; 20; 22; 23; F
- Camino básico 5: 1; 3; 5; 8; 10; 11; 13; 14; 17; 19; 20; 21; 20; 22; 23; F
- Camino básico 6: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 17; 19; 20; 21; 20; 22; 23; F
- Camino básico 7: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 15; 16; 14; 17; 19; 20; 21; 20; 22; 23; F
- Camino básico 8: 1; 3; 5; 8; 10; 11; 12; 11; 13; 14; 15; 16; 14; 17; 19; 20; 21; 20; 22; 23; F
- Camino básico 9: 1; 3; 5; 8; 10; 11; 13; 14; 15; 16; 14; 17; 19; 20; 21; 20; 22; 23; F

- Camino básico 10: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 15; 16; 14; 17; 19; 20; 21; 20; 22; 23; F
- Camino básico 11: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 17; 18; F
- Camino básico 12: 1; 3; 5; 8; 10; 11; 12; 11; 13; 14; 17; 18; F
- Camino básico 13: 1; 3; 5; 8; 10; 11; 13; 14; 17; 18; F
- Camino básico 14: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 17; 18; F
- Camino básico 15: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 15; 16; 14; 17; 18; F
- Camino básico 16: 1; 3; 5; 8; 10; 11; 12; 11; 13; 14; 15; 16; 14; 17; 18; F
- Camino básico 17: 1; 3; 5; 8; 10; 11; 13; 14; 15; 16; 14; 17; 18; F
- Camino básico 18: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 15; 16; 14; 17; 18; F
- Camino básico 19: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 17; 19; 20; 22; 23; F
- Camino básico 20: 1; 3; 5; 8; 10; 11; 12; 11; 13; 14; 17; 19; 20; 22; 23; F
- Camino básico 21: 1; 3; 5; 8; 10; 11; 13; 14; 17; 19; 20; 22; 23; F
- Camino básico 22: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 17; 19; 20; 22; 23; F
- Camino básico 23: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 15; 16; 14; 17; 19; 20; 22; 23; F
- Camino básico 24: 1; 3; 5; 8; 10; 11; 12; 11; 13; 14; 15; 16; 14; 17; 19; 20; 22; 23; F
- Camino básico 25: 1; 3; 5; 8; 10; 11; 13; 14; 15; 16; 14; 17; 19; 20; 22; 23; F
- Camino básico 26: 1; 3; 5; 6; 7; 5; 8; 10; 11; 12; 11; 13; 14; 15; 16; 14; 17; 19; 20; 22; 23; F

**Caso de prueba para comprobar la ejecución del camino 10**

**Tabla 8.** Caso de prueba para el camino 10

(Fuente: elaboración propia)

| **Diseño para el caso de prueba del camino 10** | |
|---|---|
| Descripción | Los datos de entrada no pueden ser nulos |
| Condición de ejecución | Es válido el objeto contenedor y la llave primaria coincide con la que está almacenada en la base de datos |
| Entrada | container_id |
| Resultado esperado | Se modifica el contenedor y se guarda la información referente al mismo en la base de datos |

En este camino se prueba de forma satisfactoria la funcionalidad de modificar un contenedor de configuraciones.

### 3.2.2 Pruebas funcionales

Las pruebas funcionales se concentran en las acciones visibles para el usuario y en la salida del sistema que este puede reconocer. La validación se alcanza cuando el software funciona de tal manera que satisface las expectativas razonables del cliente [57]. Las pruebas funcionales de desarrollan utilizando la técnica de partición de equivalencia del método de prueba caja negra.

**Técnica de prueba de partición de equivalencia**

Esta técnica busca dividir el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de prueba. La partición equivalente se esfuerza por definir un caso de prueba que descubra ciertas clases de errores, reduciendo así el número total de casos de prueba que deben desarrollarse. El diseño de casos de prueba para partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada. Las entradas o salidas del sistema en grupos de acuerdo a un común comportamiento o por características comunes, por lo tanto, se espera que sean procesados de la misma manera; el resultado de la prueba para una entrada de una clase es representativo para todas las entradas de la misma clase [57].

La técnica descrita anteriormente fue aplicada a la interfaz de aplicación del software, se describe el proceso para el requisito funcional 24 con nombre Modificar contenedor de configuraciones, a continuación se muestran los resultados:

**Tabla 9.** Diseño de caso de prueba para el RF 24

(Fuente: elaboración propia)

| **Nombre del requisito** | Modificar contenedor de configuraciones | |
|---|---|---|
| **Descripción general** | **Escenarios de prueba** | **Flujo del escenario** |
| El servicio debe permitir modificar los datos de un contenedor de configuraciones | EP 1.1: Modificar contenedor correctamente. | 1. Se muestra la interfaz del *Swagger.*<br>2. Se autentica el usuario en el servicio.<br>3. Se selecciona el recurso ***containers***.<br>4. Se selecciona la operación ***put*** (modificar) del recurso.<br>5. Se introduce el id del contenedor a modificar, este debe existir en la base de datos.<br>6. Se introducen correctamente los datos a |

| | | |
|---|---|---|
| | | modificar.<br>7. El servicio devuelve el código de estado de repuesta HTTP 200 (OK). |
| | EP 1.2: Modificar contenedor no existente. | 1. Se muestra la interfaz del *Swagger.*<br>2. Se autentica el usuario en el servicio.<br>3. Se selecciona el recurso ***containers****.*<br>4. Se selecciona la operación ***put*** (modificar) del recurso.<br>5. Se introduce el id del contenedor a modificar, este no existe en la base de datos.<br>6. El servicio devuelve el código de estado de repuesta HTTP 404 (Not Found) y el mensaje de error "Object Not Found". |
| | EP 1.3: Modificar contenedor incorrectamente. | 1. Se muestra la interfaz del *Swagger.*<br>2. Se autentica el usuario en el servicio.<br>3. Se selecciona el recurso ***containers.***<br>4. Se selecciona la operación ***put*** (modificar) del recurso.<br>5. Se introduce el id del contenedor a modificar, este debe existir en la base de datos.<br>6. Se introducen incorrectamente los datos a modificar.<br>7. El servicio devuelve el código de estado de repuesta HTTP 400 (Bad Request) o 404 (Not Found) y un mensaje correspondiente al error. |

A continuación, se muestra la descripción de las variables

**Tabla 10.** Descripción de variables para el RF 24

(Fuente: elaboración propia)

| No | Nombre de la variable | Tipo de dato | Puede ser nulo | Descripción |
|---|---|---|---|---|
| 1 | name | string | No | Se especifica el nombre del contenedor |
| 2 | description | string | Si | Se especifica la descripción del contenedor |
| 3 | enabled | boolean | No | Se especifica el estado del contenedor (true para habilitado o false para inhabilitado) |
| 4 | remote_user | string | No | Se especifica el usuario remoto del contenedor |
| 5 | fork | integer | No | Se especifica la confluencia del contenedor |
| 6 | category | integer | No | Se especifica la categoría del contenedor |
| 7 | host_groups | integer[ ] | No | Se especifican los grupos de host asignados al contenedor |
| 8 | configurations | integer[ ] | No | Se especifican las configuraciones del contenedor |

**Tabla 11.** Juego de datos a probar para el RF 24

(Fuente: elaboración propia)

| ID | EP | Variable 1 | Variable 2 | Variable 3 | Variable 4 | Variable 5 | Variable 6 | Variable 7 | Variable 8 | Respuesta del sistema |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | E.P 1.1 | Cont.1 | Permite modificar…/vacío | true/false | nova | [0-100] | **id existente en la BD** | **ids existentes en la BD** | **ids existentes en la BD** | El servicio devuelve el código de estado de repuesta HTTP 200 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | (OK). |
| 2 | E. P 1.3 | **vacío** | Permite modificar…/vacío | **caracteres incorrectos/ vacío** | **caracteres incorrectos/ vacío** | **caracteres incorrectos/ vacío** | **id incorrecto/vacío** | **ids incorrecto/vacío** | **ids incorrectos/vacío** | El servicio devuelve el código de estado de repuesta HTTP 400 (Bad Request) o 404 (Not Found) y un mensaje correspondiente al error. |

Las pruebas funcionales se realizaron en tres iteraciones, en la primera se encontraron un total de 14 no conformidades, 9 referidas a funcionalidad, 3 a validación y 2 de tipo error http, a las cuales fueron aplicadas acciones correctivas; para la segunda iteración se detectaron 5 no conformidad, 4 de funcionalidad y 1 de error http, sobre las cuales fueron aplicadas las mismas acciones; para la tercera iteración no se detectaron no conformidades.

### 3.2.3 Pruebas de aceptación

Las pruebas de aceptación se realizan sobre el producto terminado e integrado; están concebidas para que sea un usuario final quien detecte los posibles errores y verifique el cumplimiento de las funcionalidades del software. Estas pruebas generalmente son funcionales y se basan en los requisitos definidos por el cliente. Se clasifican en dos tipos: pruebas Alfa y pruebas Beta. Las pruebas Alfa se realizan por un cliente en un entorno controlado por el equipo de desarrollo, y las pruebas Beta se realizan en las instalaciones propias de los clientes [58]. El cliente realizó pruebas de aceptación de tipo Alfa, en

las cuales no se detectaron no conformidades y en conformidad con la herramienta desarrollada emitió un acta de aceptación de productos de trabajo.

### 3.2.4 Evaluación del objetivo de la investigación

Con la puesta en práctica de la propuesta de solución se esperan obtener los siguientes beneficios:

- Punto único y centralizado en cuanto a políticas de seguridad.
- Fácil reutilización del trabajo para la migración de empresas y organismos.
- Reducción de incidencias por máquina y mejora en la seguridad de estas.
- Reducción de tareas repetitivas en cada una de las estaciones de trabajo.
- Reducción del tiempo de instalación de paquetes y configuraciones particulares del sistema.

La satisfacción del cliente dentro del proceso de desarrollo de software se establece como un punto central en la validación de cualquier investigación científica. Para medir la satisfacción del cliente existen diversas técnicas, dentro de las que se encuentra la técnica de Iadov. La técnica de Iadov constituye una vía indirecta para el estudio de la satisfacción, ya que los criterios que se utilizan se fundamentan en las relaciones que se establecen entre tres preguntas cerradas que se intercalan dentro de un cuestionario, ver Anexo 3; y cuya relación el sujeto desconoce. Estas tres preguntas se relacionan a través de lo que se denomina el "Cuadro lógico de Iadov". El número resultante de la interrelación de las tres preguntas indica la posición de cada sujeto en la siguiente escala (de 1 a 6) de satisfacción: [60]

1. Clara satisfacción (A).
2. Más satisfecho que insatisfecho (B).
3. No definida (C).
4. Más insatisfecho que satisfecho (D).
5. Clara insatisfacción (E).
6. Contradictoria (C).

**Tabla 12.** Cuadro lógico de Iadov

(Fuente: elaboración propia)

| Pregunta 4. Posterior a haber interactuado con el servicio *RESTful* Nova Brouwer evidencie su grado de aceptación en correspondencia con | Pregunta 2. ¿Considera usted necesaria la administración centralizada de las configuraciones en las estaciones de trabajo que utilicen GNU/Linux Nova? | | |
|---|---|---|---|
| | **Si** | **No** | **No sé** |

| la herramienta. | Pregunta 3. ¿Considera usted factible contar con un servicio web que permita automatizar el proceso de administración centralizada de las configuraciones en las estaciones de trabajo que utilicen GNU/Linux Nova? | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Si** | **No** | **No sé** | **Si** | **No** | **No sé** | **Si** | **No** | **No sé** |
| Me gusta | 1 | 6 | 2 | 2 | 6 | 2 | 2 | 6 | 2 |
| Me gusta más de lo que me disgusta | 2 | 6 | 2 | 2 | 6 | 2 | 2 | 3 | 3 |
| Me da igual | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Me disgusta más de lo que me gusta | 6 | 4 | 4 | 6 | 4 | 4 | 3 | 4 | 4 |
| No me gusta nada | 6 | 5 | 4 | 6 | 5 | 5 | 6 | 5 | 4 |
| No sé qué decir | 6 | 4 | 3 | 6 | 4 | 3 | 3 | 3 | 3 |

Según el valor de satisfacción de cada elemento de la muestra se establece una referencia grupal al calcular el índice de satisfacción de los encuestados. El Índice de Satisfacción Grupal (ISG) se calcula por la siguiente fórmula:

$$ISG = \frac{A(1)+B(0,5)+C(0)+D(-0,5)+E(-1)}{N}$$

Donde, A, B, C, D, E, representan el número de respuestas con índice individual 1; 2; 3 o 6; 4; 5 respectivamente y N representa el número total de respuestas que se obtuvieron. Para poder ponderar el ISG, se establece una escala numérica entre +1 y -1, como se aprecia en la siguiente tabla:

**Tabla 13.** Ponderación del ISG

(Fuente: elaboración propia)

| | |
|---|---|
| **+1** | Máximo de satisfacción |
| **0.5** | Más satisfecho que insatisfecho |
| **0** | No definido y contradictorio |
| **-0.5** | Más insatisfecho que satisfecho |
| **-1** | Máxima insatisfacción |

Los valores que se encuentran comprendidos entre - 1 y - 0,5 indican insatisfacción; los comprendidos entre - 0,49 y + 0,49 evidencian contradicción y los que caen entre 0,5 y 1 indican que existe satisfacción [59].

Los valores de la satisfacción individual, teniendo en cuenta la valoración de 5 especialistas de migración, se comportaron de la siguiente manera:

**Tabla 12.** Resultados de la satisfacción individual

(Fuente: elaboración propia)

| Categorías | N=5 | Escala |
|---|---|---|
| Clara satisfacción | 3 | A |
| Más satisfecho que insatisfecho | 2 | B |
| No definido | 0 | C |
| Más insatisfecho que satisfecho | 0 | D |
| Clara insatisfacción | 0 | E |
| Contradictorio | 0 | C |

El cálculo del ISG, teniendo en cuenta los valores de la Tabla 12, fue de 0.8. Este resultado, según la escala de clasificación de la Tabla 11 indica la satisfacción de los encuestados con respecto al servicio *RESTful* Nova Brouwer.

### 3.3 Consideraciones finales

En este capítulo se diseñaron los diagramas de componentes y despliegue para facilitar la comprensión de las relaciones entre los componentes de software y hardware. Se definieron estándares de codificación para garantizar que otras personas puedan entender el código y depurar partes del mismo. Se emplearon las pruebas unitarias y funcionales para comprobar el correcto flujo de trabajo de la solución, así como identificar y corregir las no conformidades detectadas en cada iteración, garantizando así el correcto funcionamiento del servicio *RESTful* Nova Brouwer. Las pruebas de aceptación y la aplicación de la técnica de Iadov, permitieron evaluar la satisfacción del cliente y usuarios finales respectivamente, constatando así el cumplimiento del objetivo de la investigación.

## Conclusiones

Mediante la investigación realizada y el desarrollo de la herramienta web Nova Brouwer, en el presente trabajo de diploma se cumplen los objetivos trazados, destacando los siguientes aspectos:

- El estudio y análisis de las herramientas informáticas para la administración centralizada de la configuración permitió definir *Ansible* como solución capaz de automatizar el proceso de administración centralizada de la configuración para estaciones de trabajo que utilicen GNU/Linux Nova.
- La etapa de análisis y diseño realizada sobre la solución propuesta permitió estructurar el servicio *RESTful* y precisar las tareas a desarrollar en el período de implementación.
- Se desarrolló siguiendo los estándares de codificación definidos un servicio *RESTful* para la administración centralizada de las estaciones de trabajo que utilicen GNU/Linux Nova.
- Las pruebas y técnicas aplicadas al producto de software final permitieron evaluar la investigación, el correcto funcionamiento del servicio y determinar la satisfacción del cliente y usuarios.

Por lo antes expuesto, se concluye que se desarrolló satisfactoriamente una solución informática que permite la administración centralizada de la configuración para las estaciones de trabajo que utilicen GNU/Linux Nova.

**Recomendaciones**

El servicio *RESTful* Nova Brouwer dispone de las funcionalidades necesarias para modificar las configuraciones en las estaciones de trabajo, por lo que se sugieren las siguientes recomendaciones:

- Desarrollo de un *script* que permita copiar llaves *ssh* de forma paralela en las estaciones de trabajo.
- Desarrollo de una solución para establecer configuraciones globales de *gsettings*.
- Desarrollo de una solución que impida el cambio de las configuraciones establecidas por el especialista de migración.
- Desarrollo del *frontend* para el servicio *RESTful* Nova Brouwer empleando marcos de trabajo modernos.
- Incorporación de otras configuraciones a modificar en las estaciones de trabajo, en respuesta a necesidades eventuales.

**Referencias Bibliográficas**

[1]. MONTES DE OCA MONTANO, Jose Luis. *La migración hacia software libre en Cuba: complejo conjunto de factores sociales y tecnológicos en el camino de la soberanía nacional*. Revista Universidad y Sociedad [seriada en línea], 2015, 7 (3). pp. 119-125. [Consultado en noviembre del 2017]. Disponible en: http://rus.ucf.edu.cu/index.php/rus/article/view/242.

[2]. República de Cuba. Partido Comunista de Cuba. Lineamientos de la política económica y social del Partido y la Revolución: La Habana: PCC. (2011). [Consultado en noviembre del 2017]. Disponible en: http://www.cubadebate.cu/wp-content/uploads/2011/05/folleto-lineamientos-vi-cong.pdf

[3]. Landscape. Sitio oficial de Landscape. [En línea]. [Consultado en noviembre del 2017]. Disponible en: https://landscape.canonical.com/.

[4]. PORTO PEREZ, Julián y GARDEY, Ana. Definición.de. [En línea]. [Consultado en noviembre del 2017]. Disponible en: https://definicion.de/.

[5]. MORLEY, Deborah. *Understanding Computers: Today and Tomorrow*, 13ra ed. Course Technology, Stamford, CT. (2010). [Consultado en noviembre del 2017]. Disponible en: http://trove.nla.gov.au/work/7135175.

[6]. JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. *El Proceso Unificado de Desarrollo de Software*. Pearson Addisson-Wesley. (2000). [Consultado en noviembre del 2017]. Disponible en: https://ingenieriasoftware2011.files.wordpress.com/2011/07/el-lenguaje-unificado-de-modelado-manual-de-referencia.pdf

[7]. CRAIG, Zacker. *Redes. Manual de Referencia*. McGraw Hill. (2002). [Consultado en noviembre del 2017]. Disponible en: https://disqus.com/home/discussion/channel-ninttesbiobous/redes_manual_de_referencia_craig_zacker_pdf/.

[8]. IBM Knowledge Center. Sitio oficial de IBM. [En línea]. [Consultado en noviembre del 2017]. Disponible en: https://www.ibm.com/.

[9]. "What is Computer security?", Matt Bishop, IEEE Security and Privacy Magazine 1(1):67 - 69, 2003. DOI: 10.1109/MSECP.2003.1176998. [Consultado en noviembre del 2017]. Disponible en: http://nob.cs.ucdavis.edu/bishop/papers/2003-spcolv1n1/whatis.pdf

[10]. ME Desktop Central (Ficha del Producto). [Consultado en noviembre del 2017]. Disponible en: http://www.ireo.com/es/fabricantes-y-productos/manageengine/desktop-central/

[11]. Sophos-Central (Ficha del Producto). [Consultado en noviembre del 2017]. Disponible en:

http://www.ireo.com/fabricantes-y-productos/sophos/sophos-central/
[12]. ENTERPRISE MANAGEMENT ASSOCIATESC (EMA) White Paper Prepared for Canonical. Enterprise Class Ubuntu Management with Canonical Landscape. (2012). [Consultado en noviembre del 2017]. Disponible en: https://insights.ubuntu.com/wp-content/uploads/226b/EMA-WP.pdf
[13]. Chef. Sitio oficial de Chef. [En línea]. [Consultado en noviembre del 2017]. Disponible en: https://www.chef.io/solutions/compliance/.
[14]. CFEngine. Sitio official de CFEngine. [En línea]. [Consultado en noviembre del 2017]. Disponible en: https://cfengine.com/product/.
[15]. Puppet. Sitio oficial de Puppet. [En línea]. [Consultado en noviembre del 2017]. Disponible en: https://puppet.com/products/puppet-enterprise
[16]. Saltstack. Sitio oficial de Saltstack. [En línea]. [Consultado en noviembre del 2017]. Disponible en: https://saltstack.com/saltstack-enterprise
[17]. Ansible. Sitio oficial de Ansible. [En línea]. [Consultado en noviembre del 2017]. Disponible en: https://www.ansible.com/tower
[18]. Ansible Tower User Guide (Manual de Usuario) ]. [Consultado en noviembre del 2017]. Disponible en: http://docs.ansible.com/ansible-tower/latest/pdf/AnsibleTowerUserGuide.pdf
[19]. TORBERNTSSON, Kim y RYDIN, Ylva. “A Study of Configuration Management Systems: Solutions for Deployment and Configuration of Software in a Cloud Environment”. (2014). [Consultado en noviembre del 2017].
[20]. CARBONELL MUELA, Enrique y GARCÌA PÈREZ, Ana María. Comparación de Herramientas de Gestión de la Configuración. (2016). [Consultado en noviembre del 2017]. Disponible en: http://www.informaticahabana.cu/sites/default/files/ponencias/GES64.pdf
[21]. NELSON-SMITH, Stephen. Test-Driven Infrastructure with Chef. (2011). [Consultado en noviembre del 2017]. Disponible en: http://docs.linuxtone.org/ebooks/autOps/
[22]. CFEngine AS. CFEngine and Other Configuration Management Software. (2010). [Consultado en noviembre del 2017]. Disponible en: https://docs.cfengine.com/docs/archive.bakSpecialTopic_Comparison.pdf
[23]. LOOPE, James. Managing Infrastructure with Puppet. (2011). [Consultado en noviembre del 2017]. Disponible en: http://freepdf-books.com/managing-infrastructure-with-puppet/
[24]. HALL, Joseph. Mastering SaltStack. (2015). [Consultado en noviembre del 2017]. Disponible en:

http://freepdf-books.com/mastering-saltstack-book/
[25]. Tutorials Point. Saltstack. (2017). [Consultado en noviembre del 2017]. Disponible en: https://www.tutorialspoint.com/saltstack/saltstack_tutorial.pdf
[26]. HALL, Daniel. Ansible Configuration Management. (2013). [Consultado en noviembre del 2017]. Disponible en: http://droppdf.com/v/hZu73
[27]. RAMOS, Galo y PÁEZ, Jaime. Análisis del método para calificación de software QSOS para la selección de software aplicable a procesos educativos. (2011). [Consultado en noviembre del 2017]. Disponible en: http://www.ingenieria.ute.edu.ec/enfoqueute/index.php/revista/article/view/12/12
[28]. INFRASTRUCTURE CONFIGURATION MANAGEMENT TOOLS. Revista Universidad y Sociedad [seriada en línea], 10-2014. pp. 37-40. [Consultado en noviembre del 2017].
[29]. SANCHEZ RUBIO, Manuel y LOPEZ-CIVERA, German y MARTINEZ HERRAIZ, José Javier. Automatic Generation Of Virtual Machines For Security Training. (2016). [Consultado en noviembre del 2017]. Disponible en: http://www.revistaieeela.pea.usp.br/issues/vol14issue6June2016/14TLA6_44SanchezRubio.pdf
[30]. PEREZ GOMEZ, Elena. Estudio sobre la validez jurídica de la licencia GPL v3 en el marco normativo español de los derechos de autor y otras licencias Opensource. (2009). [Consultado en noviembre del 2017]. Disponible en: http://www.lapastillaroja.net/MT-archives/images/EstudioGPLv3.pdf
[31]. MUÑOZ MARIN, Simón. Aplicación de herramientas *DevOps* en entornos de Desarrollo Web. (2015). [Consultado en noviembre del 2017]. Disponible en: https://riunet.upv.es/bitstream/handle/10251/49121/Memoria.pdf
[32]. JACOB, Adam. DevOps Tools. (2016). [Consultado en noviembre del 2017]. Disponible en: http://www.indiangnu.org/wp-content/uploads/2017/09/Devops-Comparison-v3.pdf
[33]. NAVARRO MARSET, Rafael. Modelado, Diseño e Implementación de Servicios Web. (2007). [Consultado en noviembre del 2017]. Disponible en: http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf
[34]. VAN ROSSUM, Guido. El Tutorial de Python. (2009). [Consultado en noviembre del 2017]. Disponible en: http://python.org.ar/pyar/Tutorial
[35]. Python 3.6.4rc1 documentation. Python Software Foundation. (2017). [Consultado en noviembre del 2017]. Disponible en: https://docs.python.org/3/
[36]. PyCharm Python IDE for Professional Developers. Sitio oficial de Jetbrains. [En línea]. [Consultado

en noviembre del 2017]. Disponible en: https://www.jetbrains.com/pycharm/
[37]. Django REST Framework. Sitio oficial de Django REST Framework. [En línea]. [Consultado en noviembre del 2017]. Disponible en: http://www.django-rest-framework.org/
[38]. Swagger. Sitio oficial de IBM. [En línea]. [Consultado en noviembre del 2017]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SSMKHH_10.0.0/com.ibm.etools.mft.doc/bi12018_.htm
[39]. GILFILLAN, Ian. La Bilia de MySQL. (2003). [Consultado en noviembre del 2017]. Disponible en: http://didepa.uaemex.mx/clases/Manuales/MySql/MySql-La%20biblia%20de%20mysql.pdf
[40]. CHACON, Scott y STRAUB, Ben. Pro Git. [Consultado en noviembre del 2017]. Disponible en: https://git-scm.com/book/es/v2
[41]. Visual Paradigm. Sitio oficial de Visual Paradigm. [En línea]. [Consultado en noviembre del 2017]. Disponible en: https://www.visual-paradigm.com/
[42]. RONDÓN, Y. et al. Diseño de la base de datos para sistemas de digitalización y gestión de medias. Revista de Informática Educativa y Medios Audiovisuales, 2011, 8(15), p. 17-25. [Consultado en noviembre del 2017]. Disponible en: https://www.researchgate.net/publication/301750902_Extension_de_la_herramienta_Visual_Paradigm_for_UML_para_la_evaluacion_y_correccion_de_Diagramas_de_Casos_de_Uso
[43]. HERNANDÉZ ORALLO, Enrique. El lenguaje Unificado de Modelado (UML). [En línea]. Consultado en noviembre del 2017]. Disponible en: http://www.acta.es/medios/articulos/informatica_y_computacion/026067.pdf
[44]. MENÉNDEZ-BARZANALLANA ASENSIO, Rafael. Universidad de Murcia. [En línea]. (2012). [Consultado en noviembre del 2017]. Disponible en: http://www.um.es/docencia/barzana/IAGP/IAGP2Metodologias-de-desarrollo.html
[45]. RIVAS, Carlos Ignacio [et al]. Metodologías actuales de desarrollo de software. (2015). [Consultado en noviembre del 2017]. Disponible en: http://www.ecorfan.org/bolivia/researchjournals/Tecnologia_e_innovacion/vol2num5/Tecnologia_e_Innovacion_Vol2_Num5_6.pdf
[46]. DECSAI (Departamento de Ciencias de la Computación e I.A). Especificación de requerimientos. Universidad de Granada. [Consultado en noviembre del 2017]. Disponible en: http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf

[47]. SOMMERVILLE, Ian. *Ingeniería de Software.* Madrid: Pearson Addison Wesley, 2005. ISBN:84-7829-074-5.

[48]. LAWRENCE PELEEGER, Shari. Ingeniería de Software Teoría y Práctica. (2002).

[49]. LETELIER TORRES, Patricio y SANCHEZ LOPEZ, Emilio. Metodologías ágiles para el desarrollo de software. (2003). [Consultado en noviembre del 2017]. Disponible en: http://issi.dsic.upv.es/archives/f-1069167248521/actas.pdf

[50]. SÁNCHEZ RODRÍGUEZ, Tamara. 2015. Metodología de desarrollo para la actividad productiva de la UCI. Universidad de las Ciencias Informáticas. La Habana. Cuba : s.n., 2015.

[51]. DE LA TORRE, Cesar. Guía de Arquitectura de N-Capas orientada al Dominio con .NET 4.0 Guía. 2010. [Consultado en noviembre del 2017]. Disponible en: http://interaktiv.cl/clases/csharp/Guia_Arquitectura_N-Capas_DDD_NET_4_%28Borrador_Marzo_2010%29.pdf

[52]. FERNÁNDEZ ROMERO, Yenisleidy; DÍAZ GONZÁLEZ, Yanette. Patrón Modelo-Vista-Controlador. Revista Telem@tica [seriada en línea], 2012, 11 (1). pp. 47-57. [Consultado en noviembre del 2017]. Disponible en: http://revistatelematica.cujae.edu.cu/index.php/tele/article/view/15.

[53]. SILBERSCHATZ, Avi; KORTH, Hank y SUDARSHAN, S. Fundamentos de bases de datos. (2002). [Consultado en noviembre del 2017]. Disponible en: https://unefazuliasistemas.files.wordpress.com/2011/04/fundamentos-de-bases-de-datos-silberschatz-korth-sudarshan.pdf

[54]. GARCÍA PEÑALVO, Francisco José y PARDO AGUILAR, Carlos. Diagramas de Clase en UML 1.1. [Consultado en noviembre del 2017]. Disponible en: https://repositorio.grial.eu/bitstream/grial/353/1/DClase.pdf

[55]. LARMAN, Craig. UML y Patrones. Segunda edición. (2003). [Consultado en noviembre del 2017]. Disponible en: http://www.fmonje.com/UTN/ADES%20%20208/UML%20y%20Patrones%20%202da%20Edicion.pdf

[56]. ARIAS CALLEJA, Manuel. Carmen. Estándares de codificación. (2013). [Consultado en noviembre del 2017]. Disponible en: http://www.cisiad.uned.es/carmen/estilo-codificacion.pdf

[57]. PRESSMAN, R.S. 2010. Software engineering: a practitioner's approach. 7th ed. New York : EUA, 2010. ISBN:978-0-07-337597.

[58]. SUARÈZ, Pablo; FONTELA, Carlos. Documentación y pruebas. Antes del paradigma de objetos.

(2003). [Consultado en noviembre del 2017]. Disponible en: http://materias.fi.uba.ar/7507/content/20101/lecturas/documentacion_pruebas.pdf

[59]. ElevenPaths Blog: QA: Pruebas para asegurar la calidad del producto de software (III). [En línea]. [Consultado en noviembre del 2017]. Disponible en: http://blog.elevenpaths.com/2014/12/qa-pruebas-para-asegurarla-calidad-del.html

[60]. MONTIELO TORRADO, Jordan Abdul; GONAZÁLEZ CASTRO, Yoandry; CARBONELL TAMAYO, Arianna. ANÁLISIS Y DISEÑO DE UN SISTEMA PARA LA GESTIÓN DE LA INFORMACIÓN DE LOS PROCEDIMIENTOS DEL SERVICIO DE HEMODINÁMICA. (2016). [Consultado en noviembre del 2017]. Disponible en: http://www.informaticahabana.cu/sites/default/files/ponencias/SLD09.pdf