Universidad de las Ciencias Informáticas

Facultad 3

Módulo de notificaciones para OpenNebula

Trabajo de Diploma para optar por el Título de
Ingeniero en Ciencias Informáticas

Autores: Luis Ramiro Mendiburo Martínez

Tutores: Ing. Cecilia Esther Hernández Espinosa

La Habana, julio de 2018
"Año 60 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro por este medio que yo Luis Ramiro Mendiburo Martínez, con carnet de identidad 95010430466 soy el autor principal del trabajo de diploma titulado: "*Módulo de notificaciones para OpenNebula"* y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste firmo la presente a los ___ días del mes de _____ del año 2018

Autor:

_____

Luis Ramiro Mendiburo Martínez

Tutor:

_____

Ing. Cecilia Esther Hernández Espinosa

DEDICATORIA

A mi madre por ser mi maestra más exigente y mi amiga más leal.

A mi esposa por luchar codo a codo junto a mí.

A mi abuela *Mami* por haberme amado tanto.

AGRADECIMIENTOS

A mi mama por todos sus regaños y sermones que hicieron que todo esto fuera posible.

A mi hermana *Erika* por ser tan especial.

A mi abuelo *Pipo* por enseñarme que el conocimiento en la vida no solo se encuentra en libros.

A mi abuela *Mami* por amarme como nadie.

A mi tía y mis primas por brindarme tanto apoyo.

A la familia de mi esposa por sacrificarse tanto por mí, acogerme como a uno más, hacerme sentir tan bien a su lado.

A los *Causas* por ser mis hermanos en la vida.

A mis amigos de la UCI por sopórtame y ayudarme tanto.

A todos mis profesores por su dedicación y sacrificio.

A mi tutora por su tiempo y comprensión.

A mi esposa *Dayi* por ser mi principal triunfo en la vida, por sacrificarse tanto junto a mí, por aceptar mis fallos y aplaudir mis aciertos, y por hacerme tan feliz cada día.

A todas y cada una de las personas que hicieron este día realidad.

RESUMEN

La gestión de eventos y notificaciones de alertas sobre los recursos de computo permite a los administradores de Tecnologías de la Información estar al tanto de un cambio de estado que tenga importancia para la gestión de un elemento de configuración o un servicio de Tecnologías de la Información. Actualmente, las infraestructuras nubes deben lograr una alta disponibilidad, tolerancia a fallos, escalabilidad y un uso óptimo de los recursos, estos requerimientos son los que hacen de la gestión de alertas un elemento significativo. Soluciones propietarias como: Microsoft Azure, IBM Service Delivery Manager y VMware–vCloud, presentan de manera nativa las notificaciones sobre los cambios ocurridos en sus elementos de infraestructura. OpenNebula cuenta con alta aceptación en el mercado de la computación en la nube, dada su flexibilidad, modularidad, interoperabilidad, usabilidad y sencillez; sin embargo, brinda muy poco soporte respecto al monitoreo y control de sus recursos. El monitoreo y supervisión de OpenNebula puede mejorar con el desarrollo de una solución que permite enviar alertas sobre un comportamiento crítico de los elementos involucrados en la infraestructura. El presente trabajo tiene como objetivo el desarrollo de un módulo que permite enviar alertas sobre un comportamiento del estado de los recursos de computo que forman parte de la infraestructura de OpenNebula. El módulo obtenido permite el control de recursos como CPU, memoria RAM y almacenamiento.

Palabras claves:

Alertas, computación en la nube, notificaciones, OpenNebula.

ABSTRACT

The management of events and notifications of alerts about computing resources allows Information Technology administrators to be aware of a change in status that is important for the management of a configuration element or an Information Technology service. At present, cloud infrastructures must achieve high availability, fault tolerance, scalability and optimal use of resources, these requirements are what make alerts management a significant element. Proprietary solutions such as: Microsoft Azure, IBM Service Delivery Manager and VMware-vCloud, have natively the notifications about the changes that have occurred in their infrastructure elements. OpenNebula has high acceptance in the cloud computing market, given its flexibility, modularity, interoperability, usability and simplicity; however, it provides very little support in monitoring and controlling its resources. The monitoring and supervision of OpenNebula can be improved with the development of a solution that allows sending alerts about the critical behavior of the elements involved in the infrastructure. The objective of this work is the development of a module that allows sending alerts about the behavior of the state of the computing resources that are part of the OpenNebula infrastructure. The module obtained allows the control of resources such as CPU, RAM and Hard Disk.

Keywords:

Alerts, cloud computing, notifications, OpenNebula.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El incremento de las redes y los dispositivos interconectados es cada día mayor, y su uso en los procesos de las organizaciones y entidades se ha vuelto un factor necesario. Cada día se depende más del buen funcionamiento de los sistemas de comunicaciones, por lo que los usuarios y recursos que interactúan en una red requieren que el tiempo de respuesta a sus solicitudes sea cada vez menor. Es por ello que las organizaciones buscan vías más eficientes de organizar y gestionar las Tecnologías de la Información y las Comunicaciones (TIC) para la entrega de sus servicios. Uno de los retos que enfrentan, consiste en dimensionar óptimamente los recursos de computación, para satisfacer rápidamente las constantes y variadas demandas del negocio.

En este contexto, la computación en la nube ha surgido como otra manera de hacer llegar la tecnología a los usuarios de las Tecnologías de la Información y las Comunicaciones. Con la adopción de la nube se proponen soluciones escalables, flexibles y más eficientes en cuanto al uso de los recursos. En la actualidad Cuba, como parte de sus acciones para informatizar la sociedad, analiza las vías más factibles para lograr este objetivo. Instituciones educaciones, como la Universidad de las Ciencias Informáticas (UCI), investigan la factibilidad de las tendencias mundiales en el campo de la computación y las comunicaciones, y su adecuada implementación en el territorio nacional. Como parte de estas investigaciones el departamento docente de Técnicas de Programación y Sistemas Digitales perteneciente a la Facultad 1 de la UCI, lleva a cabo un proyecto de investigación y desarrollo (I+D) para la implementación de una nube privada en la comunidad universitaria.

Para la implementación de una nube privada en la UCI destaca OpenNebula por resaltar dentro de las propuestas de Software Libre y Código Abierto (SLCA) como la más integral, al destacar su penetración y la variedad de hipervisores soportados. Además, sobresalen sus mecanismos de virtualización, soporte para la migración en vivo de máquinas virtuales, sistemas de almacenamiento y tipos de red soportados. Gracias a su arquitectura abierta, sus interfaces y los componentes que lo conforman, OpenNebula proporciona la flexibilidad y extensibilidad que muchas empresas necesitan para la adopción de nubes [1].

OpenNebula no cuenta con una gestión eficiente de los elementos involucrados en su infraestructura, debido a que solo proporciona a los usuarios y administradores reportes de contabilidad. Estos reportes solo incluyen el consumo de red, recursos de CPU y memoria asignada; no muestran información sobre la subutilización o sobreutilización de los recursos de las máquinas virtuales (MV) y en los nodos. Un elemento fundamental cuando se brindan servicios de computación en la nube es garantizar una sólida supervisión y monitoreo de

los recursos involucrados; este elemento es significativo en aras de lograr una alta disponibilidad, tolerancia a fallos, escalabilidad y uso óptimo de los recursos [1].

Esta debilidad mostrada por el gestor se equilibra con la implementación en paralelo de una herramienta de monitoreo y supervisión, sin embargo, la introducción de otro sistema obliga a destinar valiosos recursos de cómputo que pudieran estar en función de brindar una mejor calidad de servicio. Para el proyecto de I+D del departamento docente de Técnicas de Programación y Servicios Digitales de la Facultad 1 resulta significativo el uso eficiente de los recursos que tienen destinados, por lo que se buscan vías que permitan conocer el estado de los recursos en tiempo real, recibir notificaciones y consumir la menor cantidad de recursos posible.

Ante la problemática descrita anteriormente se define como **problema de la investigación:** ¿Cómo contribuir a la generación de notificaciones del uso de los recursos de hardware que forman parte de la infraestructura de OpenNebula?

Se establece como **objeto de estudio:** Administración de notificaciones del uso de los recursos de hardware que forman parte de la infraestructura de una nube, y se declara como **campo de acción:** Administración de notificaciones del uso de los recursos de hardware que forman parte de la infraestructura de OpenNebula.

Lo que permite que se plantee como **objetivo general:** Desarrollar una herramienta que permita la generación de notificaciones del uso de los recursos de hardware que forman parte de la infraestructura de OpenNebula.

Lo que puede originar **posibles resultados** como:

Una herramienta que pueda monitorear el uso de los recursos de hardware que forman parte de la infraestructura de OpenNebula, a partir de la generación de notificaciones relacionadas con el uso de estos recursos.

Para dar solución al problema se desarrollaron los siguientes **objetivos específicos**:

1. Caracterizar las tendencias actuales asociadas a las principales soluciones nubes y su gestión de notificaciones.
2. Realizar el análisis y diseño de la propuesta de solución para la generación de notificaciones de recursos de hardware.
3. Implementar la herramienta y realizar pruebas que permitan validar la solución propuesta.

Con el propósito de dar solución al objetivo general antes mencionado se utilizaron los siguientes métodos:

Métodos Teóricos

- **Analítico-Sintético:** Permitió identificar los principales conceptos, características y funcionalidades de los principales gestores nubes y el manejo de notificaciones. Se analizaron y estudiaron documentos, trabajos de diploma, artículos y bibliografía de diferentes autores, para así extraer los elementos más significativos para la investigación.
- **Histórico-Lógico:** Se utilizó con el objetivo de realizar un estudio sobre los antecedentes, evolución y tendencias actuales de la gestión de notificaciones en diferentes gestores nubes. Se realizó una valoración para el futuro desarrollo de una herramienta para el manejo de notificaciones en el gestor nube OpenNebula.
- **Modelación:** Se empleó para modelar los diversos diagramas que se construyen como resultado de cada una de las fases del proceso de desarrollo de la herramienta.

Métodos Empíricos

- **Observación:** Permitió hacer un análisis y evaluación del comportamiento del manejo de notificaciones de algunos de los principales gestores nubes, con el objetivo de identificar elementos fundamentales y comunes entre ellos que puedan ser aprovechados para el desarrollo de la solución.

El presente trabajo está estructurado en introducción, tres capítulos de contenido, conclusiones, recomendaciones y referencias bibliográficas. El orden y contenido de los capítulos se describe a continuación:

Capítulo 1. "Fundamentación teórica**"**, se definen los principales conceptos a tener en cuenta, elementos que resultan la base que sustenta el desarrollo de los objetivos específicos. Se realiza un análisis de las principales soluciones nubes y su gestión de notificaciones, con el objetivo de determinar los elementos que engloban dicha gestión. Se definieron las tecnologías y herramientas seleccionadas para el desarrollo del proyecto.

Capítulo 2. "Análisis y diseño de la propuesta de solución**"**, se define el funcionamiento del negocio actualmente, y se plantea la propuesta de solución. Se identifican los requisitos funcionales y no funcionales de la solución. Se generan los artefactos correspondientes a las fases de análisis y diseño.

Capítulo 3. "Implementación y validación de la propuesta de solución**"**, en este capítulo se lleva a cabo la implementación de la solución propuesta y se realiza una valoración del desempeño y funcionamiento de la

herramienta en ambientes reales y virtuales. Además, se le realizan pruebas unitarias funcionales y de aceptación para una mejor verificación de la solución desarrollada. Y, por último, se muestra un análisis de los resultados arrojados por las comprobaciones.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

En este capítulo se abordan los principales conceptos asociados a la investigación. Se expone información relacionada a la arquitectura de OpenNebula y su sistema de monitoreo. Se enfatiza en el manejo de las notificaciones que brindan los gestores nubes, con el fin de encontrar las métricas comunes en ellos. Además, se seleccionan los umbrales correspondientes a las métricas seleccionadas. Y, por último, se seleccionan las herramientas y tecnologías que serán utilizadas para el desarrollo de la solución.

1.1. Conceptos asociados al dominio del problema

A continuación, se relacionan los principales conceptos que ayudan a entender el desarrollo de la investigación.

Computación en la nube: La computación en la nube es un modelo que provee el acceso bajo demanda a un conjunto compartido de recursos tales como: redes, almacenamiento, aplicaciones y servicios; a través de una red de datos. Este modelo promueve la disponibilidad de recursos de cómputo y se encuentra compuesto por cinco características esenciales: auto servicio bajo demanda, acceso de red con banda ancha, concentración de recursos, elasticidad rápida y servicio contabilizado; tres modelos de servicios: Infraestructura como Servicio (*IaaS, Infrastructure as a Service*), Plataforma como Servicio *(PaaS, Platform as a Service*) y Software como Servicio (*SaaS*, *Software as a Service*); y cuatro modelos de despliegue: Nube Pública, Nube Privada, Nube Comunitaria y Nube Híbrida [2].

Gestores nube: El software de gestión de la nube funciona sobre una infraestructura de servidor, almacenamiento y red altamente virtualizada, para ofrecer las funciones y los beneficios principales de la computación en la nube. El software de gestión de la nube, a veces incluido en una plataforma de gestión de la nube integrada, debe incluir compatibilidad con las siguientes capacidades:

- Aprovisionamiento y desaprovisionamiento automatizados de las máquinas virtuales en función de la demanda de servicios.
- Despliegue y optimización automatizados y basados en políticas de las cargas de trabajo de aplicaciones en la infraestructura de la computación en la nube virtualizada.
- Capacidad para crear, publicar y mantener un catálogo de servicios en la nube.

- Una interfaz de autoservicio a través de la cual los usuarios empresariales puedan consumir los servicios en la nube.
- Medición y facturación (o reembolso) de los servicios de varios inquilinos [3].

OpenNebula: Es una solución simple, pero cuenta con gran flexibilidad, haciéndola ideal para la administración integral de centros de datos virtualizados. OpenNebula se puede utilizar principalmente como una herramienta de virtualización, admite nubes híbridas (*Hybrid Clouds*) para combinar la infraestructura local con la infraestructura pública basada en la nube, lo que permite entornos de alojamiento altamente escalables. OpenNebula también admite nubes públicas al proporcionar interfaces de la nube para exponer su funcionalidad para máquinas virtuales, almacenamiento y administración de red [4].

Sistemas de Supervisión: Son los sistemas que se encargan de recopilar y analizar datos para determinar el rendimiento, mantenimiento y disponibilidad de la aplicación y los recursos de los que esta depende. Ayuda a comprender el funcionamiento detallado de los componentes de la aplicación y aumenta el tiempo de actividad al notificarle, de forma proactiva, cuestiones críticas para que pueda resolverlas antes de que se conviertan en problemas [5].

Métricas: Una métrica es un contenedor de información en un contexto de supervisión. Las métricas se asocian con una o más expresiones de valores de métricas que, cuando se evalúan, dan un valor a la métrica. Ayudan a supervisar el rendimiento del sistema, así como de sus componentes [5].

Notificaciones: Según la Real Academia de la Lengua Española es: Acción y efecto de notificar. Notificar: Dar noticia de algo o hacerlo saber con propósito cierto. En el alcance de la presente investigación se define como notificación a: Las alertas que emiten ciertos programas o servicios para advertir al usuario [6].

1.2. Características de OpenNebula y su sistema de monitoreo

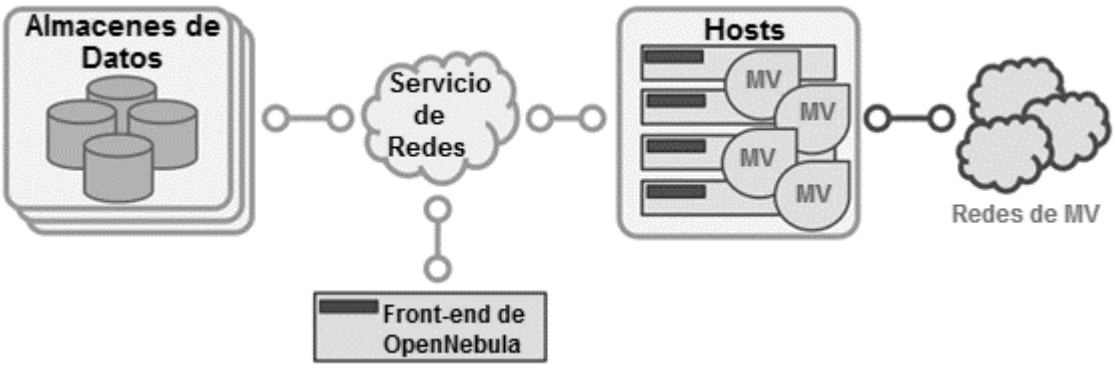
1.2.1. OpenNebula, arquitectura y componentes

Las nubes computarizadas son el siguiente paso en la evolución de la virtualización de los Centros de Datos (Data Center DC). OpenNebula permite la construcción y manejo de nubes computarizadas y Centros de Datos virtualizados, que combina la virtualización de las tecnologías existentes con características avanzadas, para lograr un suministro automático y elasticidad. OpenNebula sigue un aprovechamiento a fondo por parte de los administradores de sistema, desarrolladores y usuarios en general [7].

Visión General de la Arquitectura

La infraestructura física de OpenNebula adopta una arquitectura de clúster clásica con un Frontend, y un conjunto de nodos, donde se ejecutan las máquinas virtuales. Es necesaria al menos una red física para comunicar a todos los hosts con el Frontend.

Figura 1. Arquitectura de OpenNebula [7].



Los componentes básicos de un sistema de OpenNebula son:

Frontend: Ejecuta los servicios OpenNebula.

Hosts (se pueden ver también como anfitriones o nodo): Hipervisor habilitado, que proporcionan los recursos necesarios para las máquinas virtuales.

Almacenes de datos (Datastore): Contienen las imágenes base de las máquinas virtuales.

Redes: Utilizan dos tipos, redes físicas, que se utilizan para apoyar los servicios básicos tales como la interconexión de los servidores de almacenamiento y operaciones de control de OpenNebula, y redes virtuales, que son las VLAN para las máquinas virtuales [7].

1.2.2. Sistema de monitoreo de OpenNebula

El subsistema de monitoreo de OpenNebula recopila información relativa a los hosts y las máquinas virtuales, como el estado del host, los indicadores básicos de rendimiento, el estado de la máquina virtual y el consumo de capacidad. Esta información se recopila ejecutando un conjunto de sondas estáticas proporcionadas por OpenNebula. La salida de estas sondas se envía a OpenNebula usando un mecanismo de empuje.

Cada host periódicamente envía datos de monitoreo a través de UDP[1] al Frontend, que lo recoge y procesa en un módulo dedicado. Este sistema de monitoreo distribuido se asemeja a la arquitectura de sistemas de monitoreo dedicados, que utilizan un protocolo de comunicación ligero y un modelo de empuje.

OpenNebula inicia un demonio que se ejecuta en el Frontend que escucha conexiones UDP en el puerto 4124. En el primer ciclo de supervisión, OpenNebula se conecta al host utilizando SSH e inicia un demonio que ejecutará los scripts de sonda y enviará los datos recopilados al Frontend. De esta forma, el subsistema de monitoreo no necesita hacer nuevas conexiones SSH para recibir datos [9].

1.3. Principales gestores nubes, métricas que monitorean

A continuación, se analizan las principales soluciones nubes tanto propietarias como de software libre y código abierto, así como las principales métricas que son controladas por los sistemas de monitoreo y supervisión.

1.3.1. Análisis de suites propietarias

Microsoft Azure

Azure es un conjunto integral de servicios en la nube que permite crear, implementar y administrar aplicaciones a través de la red global de centros de datos. Promueve la productividad de los desarrolladores, pues admite un amplio abanico de sistemas operativos, lenguajes de programación, marcos de trabajos, bases de datos y dispositivos. Ofrece coherencia híbrida en el desarrollo de aplicaciones, administración y seguridad, administración de identidades y en toda la plataforma de datos. Desde el reconocimiento de imágenes hasta otros servicios, es posible aprovechar Azure y su inteligencia artificial para crear soluciones escalables. Posee gran seguridad, privacidad, transparencia y una amplia cobertura de cumplimiento normativo entre otras características [10].

[1] Protocolo que ofrece a las aplicaciones una forma de enviar y recibir un flujo ordenado y verificado de los paquetes de datos a través de red [8].

Principales métricas que monitorea:

Tabla 1. Principales métricas utilizadas por Microsoft Azure [13].

Métrica	**Unidad**
CpuPercentage	Porcentaje
MemoryPercentage	Porcentaje
Disco físico(_Total)% de tiempo de disco	Porcentaje
Disco físico(_Total)\Lecturas de disco/s	Cantidad por segundos
Disco físico(_Total)\Escrituras de disco/	Cantidad por segundos
Red\Paquetes Enviados	Cantidad
Red\Paquetes Recibidos	Cantidad
NetworkInterface\BytesTransmitted	Bytes
NetworkInterface\BytesReceived	Bytes

IBM Service Delivery Manager

IBM Service Delivery Manager es un dispositivo de software preempaquetado y autocontenido, que se implementa en un entorno de centro de datos virtual. Permite que el centro de datos acelere la creación de plataformas de servicio, para una amplia gama de tipos de carga de trabajo con un alto grado de integración, flexibilidad y optimización de recursos. Permite implementar una solución de software completa, para automatizar la gestión de servicios en un entorno de centro de datos virtual, lo que a su vez puede conformar una infraestructura más dinámica. De forma general es una solución simple que ofrece todos los componentes de software necesarios para la implementación de la nube. IBM Service Delivery Manager permite aprovechar las ventajas de este modelo de entrega en una parte definida de su centro de datos o para un proyecto interno específico [11].

Principales métricas que monitorea

- Utilización del CPU
- Utilización del disco

- Utilización de la memoria
- Tráfico de red de entrada
- Tráfico de red de salida [12]

VMware – vCloud

VMware vCloud Suite es una solución integrada que combina el hipervisor líder VMware vSphere con la plataforma de gestión de nubes híbridas multiproveedor VMware vRealize Suite. VMware permite a vCloud Suite crear y gestionar tanto nubes privadas basadas en vSphere como nubes híbridas multiproveedor. Es una de las plataformas de virtualización líderes en el mundo que funciona como base para diversas iniciativas del centro de datos. Consolida los servidores y los centros de datos. Mejora la disponibilidad y el rendimiento de las aplicaciones. Aborda los requisitos de escalabilidad vertical y horizontal de las aplicaciones. Detecta y corrige los problemas de forma proactiva, y garantiza el rendimiento y la disponibilidad de las aplicaciones en toda infraestructura y en entornos multicloud. Automatiza la distribución de una infraestructura preparada para entornos de producción en entornos multicloud mediante la automatización y el control basado en políticas, reduciendo el tiempo necesario para responder a las solicitudes de servicios de las tecnologías de la información. Acelera la distribución para las aplicaciones tradicionales y basadas en contenedores al permitir a los desarrolladores utilizar libremente las herramientas que les ayudan a ser más productivos, al tiempo que garantiza la transferencia fluida de aplicaciones del portátil del desarrollador al entorno de producción entre otras características [13].

Principales métricas que monitorea

Tabla 2. Principales métricas utilizadas por VMware [17].

Nombre de la Métrica	**Descripción**
Cpu.usage.average	Vista de host del promedio de uso activo de CPU de esta máquina virtual como porcentaje del total disponible. Incluye todos los núcleos de todos los sockets
Mem.usage.average	Memoria utilizada por esta máquina virtual como porcentaje del total de memoria configurada

Disk.provisioned.latest	Espacio de almacenamiento asignado al disco duro virtual en el centro de datos virtual de la organización en la que se encuentra
Disk.used.latest	Almacenamiento utilizado por todos los discos duros virtuales
Disk.read.average	Promedio de velocidad de lectura de todos los discos duros virtuales
Disk.write.average	Promedio de velocidad de escritura de todos los discos duros virtuales

1.3.2. Análisis de suites de software libre y código abierto

CloudStack

Apache CloudStack es un proyecto de alto nivel de la Apache Software Foundation (ASF). El proyecto desarrolla software de código abierto, para desplegar nubes de infraestructura como servicio(IaaS) públicas y privadas confiables y escalables. Es un proyecto basado en Java que proporciona un servidor de administración y agentes (si es necesario) para hosts de hipervisor para que pueda ejecutar una nube IaaS. Funciona con hosts que ejecutan XenServer / XCP, KVM, Hyper-V y / o VMware ESXi con vSphere. Proporciona una interfaz de usuario amigable basada en web para administrar la nube. Además, brinda una API nativa o puede proporcionar una API compatible con Amazon S3 / EC2 de manera opcional. Administra el almacenamiento de las instancias que se ejecutan en los hipervisores como almacenamiento primario, así como las plantillas, instantáneas e imágenes ISO como almacenamiento secundario. Orquesta servicios de red desde la capa de enlace de datos (L2) a algunos servicios de nivel de aplicación (L7), como DHCP, NAT, firewall, VPN, entre otros. Realiza la contabilidad de los recursos de red, computación y almacenamiento entre otras características [14].

Principales métricas que monitorea

Tabla 3. Principales métricas utilizadas por CloudStack [19].

Nombre de la métrica/Unidad	Descripción
Storage % Used/Porcentaje	Porcentaje de almacenamiento utilizado
CPU % Used/Porcentaje	Porcentaje de CPU utilizada
CPU Total/GHz	CPU Total
CPU Used/GHz	CPU Utilizada
Memory % Used/Porcentaje	Porcentaje de memoria utilizada
Memory Total/GB	Memoria Total
Lectura de red/kB	kB de red leídos por segundo
Escritura de red/kB	kB de red escritos por segundo

OpenStack

OpenStack es un conjunto de herramientas de software para construir y administrar plataformas de computación en la nube para nubes públicas y privadas. Permite a los usuarios implementar máquinas virtuales y otras instancias que manejan diferentes tareas para administrar un entorno de nube. Hace que el escalado horizontal sea sencillo, lo que permite que las tareas que se benefician de la ejecución simultánea pueden servir fácilmente a más o menos usuarios sobre la marcha con solo girar más instancias.

Otras de las ventajas es que OpenStack es un software de código abierto, lo que facilita que se pueda acceder al código fuente, hacer los cambios o modificaciones que necesite y compartir libremente estos cambios con la comunidad en general. También OpenStack tiene el beneficio de que miles de personas en todo el mundo trabajen en conjunto para desarrollar el producto más fuerte, más robusto y más seguro que puedan [15].

Principales métricas que monitorea

Tabla 4. Principales métricas utilizadas por OpenStack [21].

Nombre de la métrica/Unidad	**Descripción**
Uso medio del CPU/Porcentaje	Porcentaje medio de utilización del CPU
Porcentaje de disco utilizado/Porcentaje	Porcentaje de disco utilizado
Rendimiento de lectura/kB/s	Cantidad media de datos leídos (en kilobytes) en operaciones de lectura del disco que la instancia realiza en un segundo
Rendimiento de escritura/kB/s	Cantidad media de datos escritos (en kilobytes) en operaciones de lectura del disco que la instancia realiza en un segundo
Porcentaje de memoria utilizada/Porcentaje	Porcentaje de memoria utilizada
Tasa de bytes entrantes/Bytes por segundos	Número de bytes entrantes de datos por segundo en la interfaz de red
Tasa de bytes salientes	Número de bytes salientes de datos por segundo de la interfaz de red
Paquetes entrantes/Cantidad	Número de paquetes entrantes de datos en la interfaz de red
Paquetes salientes	Número de paquetes salientes de datos de la interfaz de red

1.3.3. Análisis de los gestores nubes y sus principales métricas

La **Tabla 5** propone un análisis de las principales métricas que manejan los gestores nubes anteriormente analizados, métricas que se desean tener en cuenta en la solución, por la importancia que tienen para la supervisión de la infraestructura de una nube. Es importante destacar que no se analizaron todas las métricas de cada solución, solo las más relevantes en cuanto a la importancia del recurso que controlan, el hecho de

que una métrica determinada no haya sido analizada no significa que dicha métrica no sea monitorizada por su gestor nube correspondiente.

Tabla 5. Comparación de las principales métricas utilizadas por los gestores nubes previamente analizados (Elaboración propia).

Métrica (Unidad)\Gestor Nube	Microsoft Azure	IBM Service Delivery Manager	VMware – vCloud	CloudStack	OpenStack
Utilización del CPU (Porcentaje)	**X**	**X**	**X**	**X**	**X**
Memoria usada (Porcentaje)	**X**	**X**	**X**	**X**	**X**
Disco usado (Porcentaje)	**X**	**X**	**X**	**X**	**X**
Velocidad de lectura en disco (kB/s)	**X**		**X**		**X**
Velocidad de escritura en disco (kB/s)	**X**		**X**		**X**
Tráfico de red de entrada (kB/s)	**X**	**X**		**X**	**X**
Tráfico de red de salida (kB/s)	**X**	**X**		**X**	**X**
Paquetes entrantes (Cantidad)	**X**				**X**
Paquetes salientes (Cantidad)	**X**				**X**

La gran mayoría de las métricas existentes aparece en los sistemas de casi todos los gestores nubes analizados. Evidenciando así el gran peso del control y supervisión de los recursos, por lo que resulta complejo brindar un buen servicio sin garantizar un correcto monitoreo.

Las métricas seleccionadas a tener en cuenta en la solución fueron:

- **Porciento de uso de CPU**
- **Porciento de uso de RAM**
- **Porciento de uso de Almacenamiento**

Las métricas seleccionadas se hacen comunes en todas las soluciones analizadas, evidenciado su importancia para el negocio. Además, estos componentes son algunos de los más importantes en el funcionamiento de una computadora, ya que el CPU es la unidad donde se ejecutan las instrucciones de los programas y se controla el funcionamiento de los distintos componentes del ordenador, la RAM es donde el computador guarda los datos que está utilizando en el momento presente, necesarios para que todos los procesos puedan ser ejecutados correctamente, y los dispositivos de almacenamiento son un conjunto de componentes utilizados para leer o grabar datos en algún soporte, en forma temporal o permanente.

Los elementos analizados (CPU, RAM y almacenamiento) son medidos en porciento, facilitando así el establecimiento de umbrales para poder clasificar el estado en el que se encuentran los recursos. De esta forma, el estado de una métrica estará dado por el intervalo de umbrales en que se encuentre el valor de la métrica en dicho momento.

1.4. Análisis de los umbrales de rendimiento

Los umbrales de rendimiento son un tipo de condición desencadenante que genera notificaciones cuando un recurso cae fuera de un valor especificado. Creando alertas desencadenadas por valores de umbral, es posible estar informado sobre problemas de rendimiento. Pueden establecerse límites para el rendimiento cuando se definen los umbrales. Cuando los datos de rendimiento recopilados caen fuera de estos límites, se notifica la infracción [16].

Varios son los criterios acerca de cuáles son umbrales que no se deben cruzar cuando se monitorean estas métricas. Pero todos coinciden que estos componentes (CPU, RAM y almacenamiento), necesitan un cuidadoso monitoreo y supervisión, pues se encargan de realizar las principales tareas de cualquier dispositivo. No tener un control adecuado de estos parámetros puede traer consigo fallas y errores en el correcto funcionamiento de un sistema, generando problemas como los cuellos de botellas, bloqueos parciales, pérdida de información, entre otras molestias.

Para lograr establecer intervalos que muestren el estado del rendimiento de los recursos, se analizaron varias bibliografías y las propuestas de soluciones que estas ofrecen. A continuación, se brinda un análisis de los documentos estudiados.

Porciento de uso de CPU

1. En el módulo de administración de VMM 2008, los monitores de PRO establecen un umbral de uso de CPU del 80% para host [17].
2. De forma predeterminada para Windows Server 2008 y Windows Server 2003 el umbral para el uso del CPU es de 95% [18].
3. El valor del umbral predeterminado para la rutina de supervisión de utilización de la CPU es de 95% [19].
4. El valor de este umbral se mide como un porcentaje de la capacidad de la CPU, el predeterminado es el 60% [20].
5. El porciento de uso normal de CPU que se establece para el uso del sistema es por encima de 60% [21].

Porciento de uso de RAM

6. El umbral de uso de memoria definido es del 90 por ciento para todos los hosts [17].
7. El nivel alto de uso de porciento de memoria está fijado a partir del 80% [22].

Porciento de uso de almacenamiento

8. El umbral a partir del cual el porciento de uso de disco empieza a entrar en la categoría de advertencia para la supervisión de los discos físicos y lógicos es del 90% [23].
9. El porciento máximo de espacio ocupado en el disco debe ser de 95% [20].
10. El porciento recomendable de uso del disco es del 80% [24].

Los criterios en cuanto a la definición de umbrales son amplios y muy variados. Por tal motivo para llegar a un acuerdo de criterios se conformaron cuatro intervalos, para ubicar la cantidad de umbrales antes analizados pertenecientes a cada intervalo. En la **Tabla 6** aparecen la cantidad de umbrales encontrados para ese intervalo, junto al listado con la numeración de cuales fueron.

Tabla 6. Cantidad de umbrales de rendimiento por intervalo (Elaboración propia).

Intervalos	**[0-50)**	**[50-75)**	**[75-90)**	**[90-100]**
Cantidad	0	2 [4;5]	5 [1;4;5;7;10]	10 [1;2;3;4;5;6;7;8;9;10]

.

Luego del análisis de la tabla anterior es posible concluir que a medida que el intervalo se acerca a 100% la cantidad de umbrales que coinciden es mayor. Mientras mayor sea la carga de trabajo de CPU, RAM y almacenamiento, más altos serán los porcientos de uso de los mismos y más umbrales de rendimiento cruzarán. En consecuencia, a la cantidad de umbrales por intervalos, se definió una etiqueta para reflejar el estado en que se encuentra operando dicho recurso. Las notificaciones lanzadas por la herramienta estarán definidas por el estado en que se encuentre un recurso de un host determinado. La **Tabla 7** muestra los estados definidos para cada intervalo.

Tabla 7. Relación Intervalo-Estado (Elaboración propia).

Intervalo	Estado
[0-50)	Óptimo
[50-75)	Aceptable
[75-90)	Crítico
[90-100]	Muy Crítico

1.5. Marco Tecnológico

A continuación, se explican las metodologías, técnicas y herramientas que se utilizaron para el desarrollo de la solución.

1.5.1. Metodología de desarrollo de software

Según Sommerville una metodología de desarrollo de software es un enfoque estructurado para el desarrollo de software que incluye modelos de sistemas, notaciones, reglas, sugerencias de diseño y guías de procesos. Las metodologías se pueden clasificar como ágiles y tradicionales. Las tradicionales son aquellas que fundamentalmente están centradas en el control del proceso, además son las más efectivas para proyectos de gran envergadura. Las metodologías ágiles representan una opción razonable a las metodologías de software tradicionales para ciertas clases de software y ciertos tipos de proyectos. Ha demostrado su utilidad al entregar sistemas exitosos con rapidez [25].

Extreme Programming (XP) propone la retroalimentación continua entre el cliente y el equipo de desarrollo, la comunicación fluida entre todos los participantes, y la simplicidad en las soluciones implementadas. Los

valores y principios de las metodologías ágiles expuestos en el Manifiesto Ágil son la inspiración de la misma. Esta metodología se define como adecuada para proyectos con requisitos imprecisos y cambiantes, y donde existe un alto riesgo técnico. Es una metodología encaminada para el desarrollo; consiste en una programación rápida o extrema, cuya particularidad radica en tener como parte del equipo, al cliente, pues es muy importante para llegar al éxito del proyecto [26].

Teniendo en cuenta que el equipo es pequeño, integrado en su totalidad por dos personas (el desarrollador y el cliente, quien participa constantemente con el equipo de desarrollo), los requerimientos están sujetos a cambios, el proyecto es pequeño y de corto plazo, se selecciona como metodología XP para guiar el proceso de desarrollo de la solución.

1.5.2. Herramienta CASE

Se decidió utilizar como herramienta CASE[2] *Visual Paradigm 8.0* pues esta ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo de software a través de la representación de todo tipo de diagramas. Presenta licencia gratuita y comercial. Es fácil de instalar, actualizar y compatible entre versiones. Esta herramienta fue diseñada para una gran variedad de usuarios interesados en la construcción de sistemas de software de forma confiable, mediante la utilización de un enfoque orientado a objetos. Entre sus características resaltan la disponibilidad en múltiples plataformas (*Windows, Linux*), diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad. También usa un lenguaje estándar común a todo el equipo de desarrollo, que facilita la comunicación, capacidades de ingeniería directa e inversa, modelo y código que permanece sincronizado en todo el ciclo de desarrollo. También soporta aplicaciones web, varios idiomas, compatibilidad entre ediciones, soporte de UML versión 2.1, generación de base de datos, transformación de diagramas Entidad-Relación en tablas de base de datos, generador de informes y editor de figuras, entre otras [28].

Esta herramienta permite aumentar la calidad del software, a través de la mejora de la productividad en el desarrollo y mantenimiento del software. Aumenta el conocimiento informático de una empresa ayudando así a la búsqueda de soluciones para los requisitos. También permite la reutilización del software, portabilidad y

[2] (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas o programas informáticos destinadas a aumentar la productividad en el desarrollo de software [27].

estandarización de la documentación, además del uso de las distintas metodologías propias de la ingeniería de software. En la UCI se ha generalizado su uso y es la más utilizada para realizar el modelado del proceso de desarrollo de software. Por estas razones se selecciona esta herramienta para el modelado de los artefactos a generar como parte de la solución.

1.5.3. Lenguaje de modelado

UML (*Unified Modeling Language*): Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML es un lenguaje expresivo, claro y uniforme, que no garantiza el éxito de los proyectos, pero sí mejora sustancialmente el desarrollo de los mismos, al permitir una nueva y fuerte integración entre las herramientas, los procesos y los dominios. Es independiente del proceso, aunque para utilizarlo óptimamente se debe usar en un proceso que sea dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental. Es importante resaltar que UML es un lenguaje para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar. UML en su versión 2.1 es el lenguaje de modelado que es empleado en la versión seleccionada de la herramienta CASE, estas características que posee dicho lenguaje posibilitó que UML haya sido seleccionado por el equipo de desarrollo [29].

1.5.4. Ambiente de desarrollo

A continuación, se describen los lenguajes y bibliotecas a utilizar en la implementación de la solución.

Lenguaje de programación

Java fue creado por Sun Microsystems Inc. es un lenguaje de desarrollo de propósito general, y es válido para realizar todo tipo de aplicaciones profesionales. Incluye una combinación de características que lo hacen único y está siendo adoptado por multitud de fabricantes como herramienta básica para el desarrollo de aplicaciones comerciales de gran repercusión.

Una de las características más importantes es que los programas “ejecutables”, creados por el compilador de Java, son independientes de la arquitectura. Se ejecutan indistintamente en una gran variedad de equipos con diferentes microprocesadores y sistemas operativos. Varias de las principales características de Java son:

- Es público. Puede conseguirse un JDK (Java Developer's Kit) o Kit de desarrollo de aplicaciones Java gratis.
- Permite desarrollar pequeños programas que se insertan en una página HTML y se ejecutan en el ordenador local.
- Se pueden desarrollar para intrarredes, aplicaciones cliente/servidor, aplicaciones distribuidas en redes locales y en Internet.
- Es fácil de aprender y está bien estructurado.
- Las aplicaciones son fiables. Puede controlarse su seguridad frente al acceso a recursos del sistema y es capaz de gestionar permisos y criptografía. Además, según Sun, la seguridad frente a virus a través de redes locales e Internet está garantizada.
- Es intrínsecamente orientado a objetos.
- Funciona perfectamente en red.
- Aprovecha características de la mayoría de los lenguajes modernos evitando sus inconvenientes. En particular los del C++.
- No tiene punteros manejables por el programador, aunque los maneja interna y transparentemente.
- El manejo de la memoria es sencillo, la gestiona el propio lenguaje y no el programador.
- Incorpora multihilo, para permitir la ejecución de tareas concurrentes dentro de un mismo programa.

Por estas ventajas ofrecidas se selecciona Java como lenguaje de programación [30].

Bash

Bash es una consola de comandos Unix escrita para el proyecto GNU. Su nombre es un acrónimo de Bourne-again shell —que es un juego de palabras (Bourne again / born again / Renacimiento) de la shell Bourne (sh). Bash es la consola por defecto en la mayoría de sistemas Linux, como openSUSE. Está diseñado para cumplir con el estándar IEEE POSIX P1003.2 / ISO 9945.2. Ofrece mejoras funcionales para programación y uso interactivo. Además, la mayoría de los scripts sh pueden ser ejecutados por Bash sin modificaciones.

Las mejoras ofrecidas por Bash incluyen:

- Edición de línea de comando

- Historial de comando de tamaño ilimitado
- Control de trabajo
- Funciones y alias de Shell
- Matrices indexadas de tamaño ilimitado
- Aritmética de enteros en cualquier base de 2 - 64

En este trabajo se selecciona Bash en las líneas de comando encargadas de capturar el valor de las métricas en la ejecución del monitoreo. Este lenguaje posibilita filtrar las salidas de la consola de Linux, con el fin de la correcta ejecución de las notificaciones y reportes [31].

Biblioteca Jsch

Jsch es una implementación pura de Java de SSH2. Jsch permite conectarse a un servidor sshd y usar el reenvío de puertos, la transferencia de archivos, entre otros, y puede integrar su funcionalidad en sus propios programas Java. Jsch está bajo la licencia de estilo BSD.

Jsch tiene las siguientes características:

- Se encuentra en java, pero depende de JavaTM Criptography Extension (JCE)
- Soporte de protocolo SSH2
- Intercambio de claves
- Cifrado
- MAC
- Conexión a través del proxy HTTP
- Conexión a través del proxy SOCKS5
- Reenvío de puertos
- Reenvío de flujo
- Cambio de contraseña para una clave privada
- Autenticación parcial
- Protocolo de transferencia de archivos SSH (versión 0, 1, 2, 3)

Se selecciona trabajar con esta biblioteca de java, ya que ofrece la comodidad de poder realizar conexiones SSH, necesaria para realizar las peticiones de los valores de las métricas en los monitoreos realizados [32].

Biblioteca Jdbc

Java DB(JDBC) es la distribución compatible de Oracle de la base de datos Apache Derby de código abierto. Su facilidad de uso, el cumplimiento de las normas, el conjunto completo de características y su reducido tamaño lo convierten en la base de datos ideal para los desarrolladores de Java. Java DB está escrito en el lenguaje de programación Java, proporcionando portabilidad. Puede integrarse en aplicaciones Java, sin requerir administraciones por parte del desarrollador o usuario [33].

Esta biblioteca fue escogida por el equipo de desarrollo con el objetivo de poder realizar tareas de forma fáciles con la base de datos. Esta biblioteca posibilita a los desarrolladores un trabajo más sencillo con la base de datos, implementando desde java los métodos encargados de realizar las consultas a la base de datos. Además, posibilita una abstracción del código SQL para el manejo de datos, reduciendo el tiempo de preparación del equipo de desarrollo y simplificando la aplicación.

Sistema Gestor de Base de Datos (SGBD)

PostgreSQL : Este es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD[3] y con su código fuente disponible libremente. Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. Funciona bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema. Se selecciona PostgreSQL como sistema gestor de base de datos [34].

Además, se utiliza la extensión Pgcrypto, que ofrece la posibilidad de encriptar datos en Postgresql, a partir de pg-9.1 se instala con la línea de comando: “create extension pgcrypto;”. Para ello se debe tener instalado el paquete postgresql-contrib de la versión del postgres que se esté trabajando [35].

[3] Es una licencia de software libre permisiva como la licencia de *OpenSSL* o la *MIT License*.

Herramienta para la administración de la base de datos

***PgAdmin III*:** Se usa para manejar bases de datos en *PostgreSQL 9.4* y superiores, funciona sobre casi todas las plataformas. Este software fue diseñado para responder a las necesidades de todos los usuarios, desde la escritura de simples consultas *SQL* a la elaboración de bases de datos complejas. La interfaz gráfica es compatible con todas las características de *PostgreSQL* y facilita su administración. La aplicación también incluye un editor de la sintaxis *SQL*, un editor de código del lado del servidor y un agente para la programación de tareas. Para usarlo en Linux solo hay que instalar el paquete pgadmin3. El equipo de trabajo selecciona PgAdmin III por aportar las potencialidades buscadas para este tipo de herramientas [36].

1.5.5. Entorno Integrado de Desarrollo (IDE)

NetBeans IDE permite desarrollar rápida y fácilmente aplicaciones de escritorio, móviles y web Java, así como aplicaciones HTML5 con HTML, JavaScript y CSS. También proporciona un gran conjunto de herramientas para desarrolladores PHP y C / C ++. Es gratuito, de código abierto y tiene una gran comunidad de usuarios y desarrolladores en todo el mundo.

Es el IDE[4] oficial para Java 8. Con sus editores, analizadores de código y conversores. Puede actualizar de manera rápida y sin problemas sus aplicaciones, para usar nuevas construcciones de lenguaje Java 8, operaciones funcionales y referencias de métodos.

El Editor de NetBeans introduce líneas, coincide con las palabras y los corchetes, y resalta el código fuente sintáctica y semánticamente. Le permite refactorizar fácilmente el código, con una gama de herramientas útiles y potentes, mientras que también proporciona plantillas de códigos, sugerencias de codificación y generadores de códigos. El editor admite varios idiomas, desde Java, C / C ++, XML y HTML, hasta PHP, Groovy, Javadoc, JavaScript y JSP. Como el editor es extensible, puede conectar el soporte para muchos otros idiomas.

[4] Siglas correspondientes al termino en inglés: Integrated Development Environment

NetBeans IDE proporciona diferentes vistas de sus datos, desde ventanas de proyectos múltiples a herramientas útiles para configurar sus aplicaciones y administrarlas de manera eficiente, lo que le permite profundizar en sus datos de forma rápida y sencilla entre otras ventajas que hacen del NetBeans una herramienta ideal para casi todo tipo de trabajos en Java. Es uno de los entornos de desarrollo más utilizados en el mundo por sus grandes facilidades, además, el equipo de desarrollo se encuentra familiarizado con dicha herramienta y posee gran integración con los lenguajes de programación que se utilizarán en el proyecto. Por ello se selecciona NetBeans como IDE [37].

1.6. Conclusiones parciales

Al finalizar el presente capítulo se arribó a las siguientes conclusiones parciales:

- El análisis y la fundamentación teórica de los principales conceptos asociados a la investigación, permitió lograr una mayor comprensión del alcance de la investigación y esclarecer su objeto de estudio.
- El análisis de otras herramientas existentes encargadas de realizar estas tareas, permitió la definición de las métricas y umbrales que serán manejados por la herramienta.
- La sistematización del marco teórico de la investigación científica, del estado actual de las plataformas para la gestión de nubes, y de la documentación de OpenNebula posibilitó la definición del ambiente de desarrollo para la implementación de la solución propuesta.

CAPÍTULO 2. ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN

El presente capítulo muestra el análisis y diseño de la herramienta. Para lograrlo se presenta la descripción de la propuesta de solución y los requisitos obtenidos durante el análisis. Además de un grupo de artefactos como historias de usuario, tarjetas clase-responsabilidad-colaborador, entre otros, que ayudan a comprender mejor el diseño de la herramienta.

2.1. Modelo de dominio

La **Figura 2** muestra el modelo de dominio que expresa como se comportaba el negocio en cuestión antes de desarrollar la propuesta de solución. Un modelo de dominio expresa como una empresa realiza su negocio y debe ser el mismo para cualquier otra empresa que realice el mismo negocio. En este caso el empleado encargado de la supervisión de la nube se encarga de controlar las herramientas de monitoreo y supervisión, la interfaz web Sunstone y la interfaz de líneas de comando (CLI) que a su vez controlan los cambios de estados ocurridos en OpenNebula y los nodos que esta utiliza.

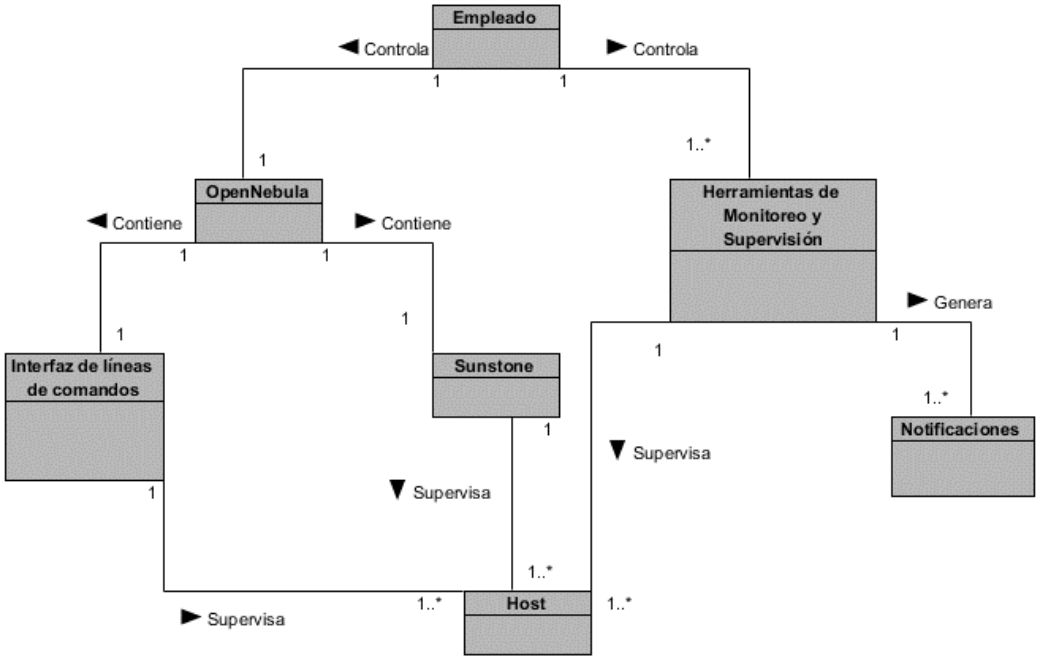


Figura 2. Diagrama de modelo de dominio (Elaboración propia).

2.1.1. Descripción de clases del modelo del dominio

OpenNebula: Gestor nube desplegado.

Hosts: Componentes básicos de los servicios de OpenNebula.

Empleado: Encargado de monitorear y controlar los cambios ocurridos en OpenNebula.

Herramientas de monitoreo y supervisión: Herramientas auxiliares que se acoplan a OpenNebula y se encargan del monitoreo y la supervisión de la misma.

Notificaciones: Mensajes lanzados por las herramientas de monitoreo y supervisión con el fin de dar a conocer al empleado de algún cambio importante en alguno de los recursos de los Nodos.

Sunstone: Es la interfaz web de OpenNebula muestra el estado de los recursos en tiempo real.

Interfaz de líneas de comando: muestra información de otros recursos.

2.2. Descripción de la propuesta de solución

Con este trabajo, se propone el desarrollo de una herramienta que permita al empleado encargado de monitorear OpenNebula monitorear grupos de host. En estos monitoreos se controlan los valores de un grupo de métricas como: porciento de uso de CPU, porciento de uso de RAM y porciento de uso de almacenamiento. Para ello es necesario agregar los datos de los hosts que se proponga controlar mediante un formulario donde se deben añadir los campos: IP, usuario y contraseña, estos datos serán almacenados por la herramienta en una base de datos que guarda el listado de los hosts que el programa monitorea. La herramienta permitirá además eliminar los hosts que no se deseen monitorear especificando sus IPs. Modificar los hosts es otra de las opciones que ofrecerá el sobre los hosts almacenados en la base de datos. De esta forma, si se desea modificar un host, primero se debe especificar qué campo (IP, usuario, contraseña o todo) y luego ingresar el nuevo valor. Mostrar registro sería otra de las funcionalidades, en la que se mostrará una tabla con la información de (Fecha/Hora, IP %UsoCPU, %UsoRAM y %UsoDisco), cuyos datos se encuentren almacenados en una base de datos y posibilite realizar un análisis sobre dichos datos con el objetivo de poder mejorar el negocio. Los tipos de monitoreo que se podrán realizar son: estándar o personalizado. En caso de seleccionarse la opción estándar un grupo de parámetros definidos por el programa controlará el monitoreo y solo se lanzarán notificaciones para aquellas métricas que se encuentren en estado “Crítico” o “Muy crítico”, en el caso que se seleccione la opción personalizada se debe poder especificar un grupo de parámetros y

características que permitan al usuario poder adaptar el monitoreo a las necesidades que tenga en ese momento, algunas de las opciones que ofrecerían son:

- Especificar con qué frecuencia se realicen los monitoreos, los cuales se harán de forma automática transcurrido el tiempo especificado.
- Seleccionar las alertas que desea recibir en consecuencia de las métricas y el estado seleccionado.

2.3. Levantamiento de requisitos

Técnicas de obtención de requisitos

Para el desarrollo de la herramienta la técnica de obtención de requisitos que se utilizó fue la **Observación de sistemas semejantes.** Para ello se realizó un estudio de soluciones semejantes, lo que permitió obtener una base de propuestas de requisitos que sirvió para el desarrollo de la solución en la presente investigación.

2.3.1. Requisitos funcionales

Un requisito funcional (RF) define una función del sistema de software o sus componentes. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Los requerimientos funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que se supone que un sistema debe cumplir. Estos son complementados por los requisitos no funcionales, que se enfocan en cambio, en el diseño o la implementación [38].

Tabla 8. Listado de requisitos funcionales del software (Elaboración propia).

Nº	**Requisitos funcionales**
RF1	Realizar monitoreo
RF2	Modificar frecuencia
RF3	Modificar alertas
RF4	Mostrar registro
RF5	Filtrar registro
RF6	Insertar host
RF7	Eliminar host

RF8	Modificar IP
FR9	Modificar usuario
RF10	Modificar contraseña
RF11	Modificar todo

2.3.2. Requisitos no funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el sistema debe tener. Estas propiedades o cualidades se refieren a las características que hacen al sistema estable, usable, rápido, confiable y escalable [38].

RNF - Usabilidad

- La herramienta será una aplicación de escritorio ubicada en el servidor.
- Se requiere un nivel medio o alto de conocimientos de informática.
- El usuario necesitará una preparación previa para operar con la herramienta.
- La herramienta posee un diseño e interfaces intuitivas, que permiten la comprensión del usuario.

RNF- Interfaz

- Debe tener una interfaz amigable.
- La interfaz debe tener un diseño sencillo y ser de fácil comprensión para el usuario.

RNF- Implementación

- La herramienta será desarrollada apoyándose en: NetBeans IDE en su versión 8.1 RC2 y PostgreSQL en su versión 9.4.
- Los lenguajes de programación usados fueron Bash para la obtención de los valores específicos de las métricas y Java 8 para la implementación de la herramienta.

RNF- Software y hardware

- Características de hardware:
    - Procesador Core i3 4 GHz.
    - Memoria RAM 4 GB.

- Disco Duro con 80 Gb libres para datos.

- Características de software en el servidor:
  - Sistema operativo:
    Linux
  - Dependencias:
    SSH

2.3.3. Historias de usuarios

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento estas pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Además, son escritas por el cliente, en su propio lenguaje, como descripciones cortas de lo que el sistema debe realizar. La diferencia más importante entre las historias y los tradicionales documentos de especificación funcional se encuentra en el nivel de detalle requerido. Las historias de usuarios deben tener el detalle mínimo como para que los programadores puedan realizar una estimación poco riesgosa del tiempo que llevará su desarrollo. Cuando llegue el momento de la implementación, los desarrolladores dialogarán directamente con el cliente para obtener todos los detalles necesarios. Las historias de usuarios deben poder ser programadas en un tiempo entre una y tres semanas. La **Tabla 9** presenta un ejemplo de las historias de usuarios registradas en el proceso de desarrollo. Las demás se encuentran en los anexos [26].

Tabla 9. Historia de usuario # 1 (Elaboración propia).

Historia de Usuario	
Código: HU1	**Nombre Historia de Usuario:** Realizar monitoreo
Referencia: RF1	
Usuario: empleado	
Programador: Luis Ramiro Mendiburo Martínez	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Tiempo Estimado:** 1 semana

Riesgo en Desarrollo: Alta	**Tiempo Real:** 1 semana

Descripción: el usuario puede realizar el monitoreo de dos formas posible, a través del botón de la interfaz principal *Monitoreo estándar* o mediante la opción *Personal monitoreo* la cual lo hará de forma sistemática una vez transcurrido el tiempo especificado por el usuario.

De ambas formas el monitoreo se ejecutará utilizando las credenciales almacenadas en la tabla “host” de la base de datos “BaseDatos”, para establecer una conexión que se realiza por el protocolo seguro SSH. Luego ejecuta los comandos que le permiten conocer el estado actual de la utilización de los recursos: CPU, RAM y HDD en ese host.

Toma los valores obtenidos de los recursos del nodo monitoreado y los compara contra el umbral establecido para cada recurso. En caso de encontrar algún tipo de alerta la herramienta mostrará una interfaz en forma de tabla con la hora y fecha que se realizó el monitoreo, el IP del host en cuestión, la métrica y el valor correspondiente a la misma y el estado en que se encuentre.

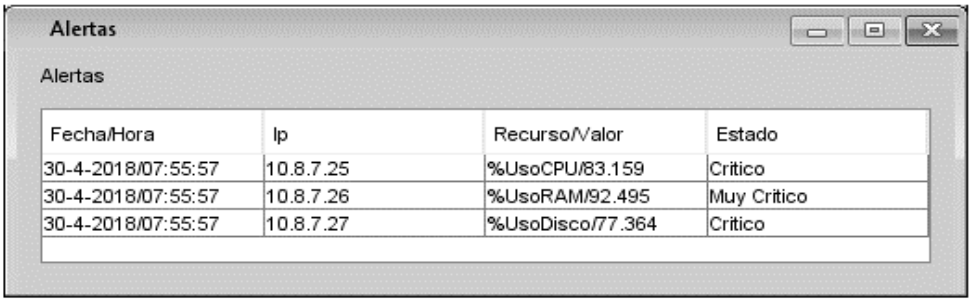
Cada vez que se realice un monitoreo los datos son almacenados en la tabla “registros” de la base de datos “BaseDatos”.

Observaciones: Se requiere para que funcione automático que el usuario haya seleccionado la opción *Personalizar monitoreo* de la pantalla principal y haya especificado el tiempo y las alertas que desee recibir.

Se requiere que el usuario haya insertado al menos un host.

Los host deben tener instalado sistema operativo GNU/Linux y permitir conexión SSH.

Interfaz:



2.4. Plan de iteraciones

Las historias de usuarios seleccionadas para cada plan de entrega son desarrolladas y probadas en un ciclo de iteración de acuerdo al orden preestablecido, se estima el tiempo de duración de cada una de ellas. Cada historia de usuario se traduce en tareas específicas de programación. La herramienta será desarrollada en tres iteraciones, la primera implementará las historias de usuarios de prioridad alta para el negocio, en una segunda las de prioridad media y en una tercera iteración las de prioridad baja [39].

Tabla 10. Plan de iteraciones (Elaboración propia).

Iteración	Historia de Usuario	Prioridad	Duración de la Historia de Usuario(días)	Duración total(semanas)
1	Realizar monitoreo	Alta	7	1 semana
2	Mostrar registro	Media	3	1 semana y 2 días
	Modificar frecuencia	Media	3	
	Modificar alertas	Media	3	
3	Filtrar registros	Baja	1	7 semana
	Añadir Host	Baja	1	
	Eliminar Host	Baja	1	
	Modificar IP	Baja	1	
	Modificar usuario	Baja	1	
	Modificar contraseña	Baja	1	
	Modificar todo	Baja	1	
Total				3 semanas y 2 días

2.5. Fase de diseño

En esta fase es donde se confeccionan las tarjetas Clase Responsabilidad Colaborador (CRC). Además, se define la arquitectura, el modelo de base de datos y otras características de la herramienta.

2.5.1. Tarjetas Clase-Responsabilidad-Colaborador(CRC)

Las tarjetas CRC son utilizadas para representar las responsabilidades de las clases y sus interacciones. Estas tarjetas permiten trabajar con una metodología basada en objetos, permitiendo que el equipo de desarrollo completo contribuya en la tarea del diseño. En cada tarjeta CRC el nombre de la clase se coloca a modo de título, las responsabilidades se colocan a la izquierda y las clases que se implican en cada responsabilidad a la derecha, en la misma línea que su requerimiento correspondiente.

Una clase es cualquier persona, evento, concepto, pantalla o reporte. Las responsabilidades de una clase son las cosas que conoce y las que realizan, sus atributos y métodos. Los colaboradores de una clase son las demás clases con las que trabaja en conjunto para llevar a cabo sus responsabilidades. La **Tabla 11** muestra un ejemplo de tarjeta CRC las demás se encuentran en los anexos [40].

Tabla 11. Tarjeta CRC: PC.java (Elaboración propia).

Tarjeta CRC	
Clase: PC.java	
Descripción: Almacena los datos de los host	
Atributos	
Nombre	**Descripción**
String ip	Guarda la dirección IP
String user	Guarda el nombre del usuario que identifica al host
String pass	Guarda la contraseña correspondiente al usuario del mismo host
Responsabilidades:	**Colaboraciones:**

Crear un objeto de tipo PC que contenga la información de un host y que posibilite al programa ejecutar una conexión SSH con dicho host y conocer el valor de las métricas.	Control.java

2.5.2. Modelo de datos

A través del modelado y diseño de la base de datos se hace posible definir todos los datos que se introducen almacenan y producen dentro de la aplicación. En este caso la base de datos de la aplicación "BaseDatos" contiene dos tablas. La primera tabla llamada "hosts" consta de 3 columnas: IP (dirección IP del host), usuario (nombre de usuario del host) y pass (contraseña del host), este último se almacena por cuestiones de seguridad de forma encriptada a través de la función (3 DES[5]). Su objetivo es poder almacenar los datos del host, para que estos puedan ser guardados, eliminados y modificados por el programa. Además, con estos datos es posible que la aplicación pueda realizar conexiones SSH, factor necesario para poder realizar el monitoreo. La segunda tabla llamada (registros) contiene 6 columnas: id_registros (valor que permite identificar a cada registro), fechahora (fecha y hora en la que se realizó el monitoreo), IP (dirección IP del host), porcusocpu (porciento de uso de CPU para ese host en el monitoreo), porcusoram (porciento de uso de RAM para el host en el monitoreo) y porcusodisco (porciento de uso de almacenamiento para el host en el monitoreo). Esta tabla permitirá almacenar los valores de los monitoreos ejecutados, permitiendo así realizar análisis sobre estos datos con el fin de obtener información para futuras investigaciones.

[5] 3 DES se basa en el algoritmo de encriptación estándar de datos (*Data Encryption Standard*) por lo que es muy fácil de utilizar. Además, tiene la ventaja de fiabilidad comprobada y una longitud de clave más larga que elimina muchos de los ataques [41].

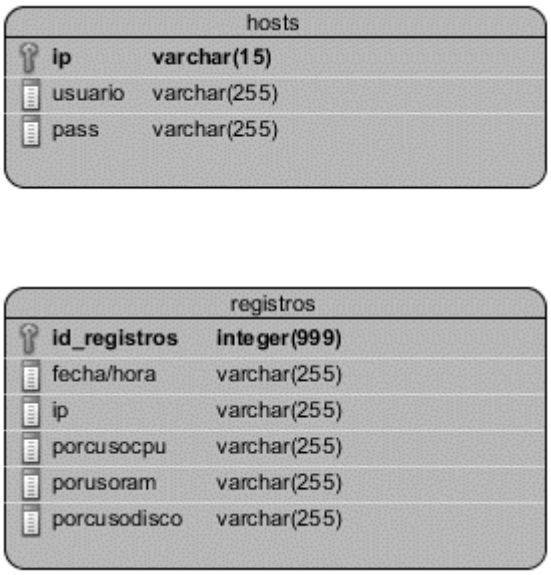


Figura 3. Modelo de la base de datos (Elaboración propia).

2.5.3. Arquitectura del sistema

Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software. Estos permiten expresar un esquema de organización estructural esencial para un sistema de software [42].

Para este trabajo se hace uso del estilo arquitectónico en capas, ya que mediante su uso se logra un sistema más organizado y se gana en orden lógico a la hora de programar.

Arquitectura en capas

La arquitectura en capas se define como una organización jerárquica donde cada capa proporciona servicios a la inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Con esto se logra abstraer las funcionalidades de una capa de manera tal que pueda ser totalmente remplazada sin afectar a las otras, solamente cambiar las referencias de las implicadas en el cambio [43].

- **Capa de interfaz:** presenta la información al usuario. Determina la interfaz que se muestra finalmente al cliente para su intercambio con la aplicación. Esta capa está presente en cada una de las interfaces del sistema *Principal.java, Adicionar.java, Eliminar.java, Modificar.java, ModificarIp.java, ModificarUser.java, ModificarPass.java, ModificarTodo.java,*

MostrarRegistro.java ya que estas clases son las encargadas de interactuar con el usuario directamente.

- **Capa de negocios:** recibe las órdenes del usuario y se encarga de mostrar las interfaces. Gestiona las peticiones del usuario y se encarga de darle respuestas. La clase *Control.java* desempeñaría la tarea de controlar las funcionalidades principales del programa ya que, mediante las opciones seleccionadas por el usuario esta clase ejecutaría la función correspondiente y la clase (PC.java) realiza la función de construir los hosts con los que se trabajan en la herramienta
- **Capa de acceso a datos:** Es la capa encargada de acceder a los datos almacenados en la base de datos y ejecutar consultas sobre los mismos. Para ello esta capa utiliza la biblioteca (jdbc) la cual permite realizar consultas a bases de datos desde aplicaciones en java.
- **Capa de datos:** Es la capa que contiene la base de datos con las tablas que almacenan la información de las credenciales de los hosts y los monitoreos realizados.
- **Capa de conexión ssh:** recibe las credenciales de los hosts, de la clase (Control.java) y mediante la biblioteca (jsch) ejecuta las peticiones de las métricas a través del comando seguro SSH.

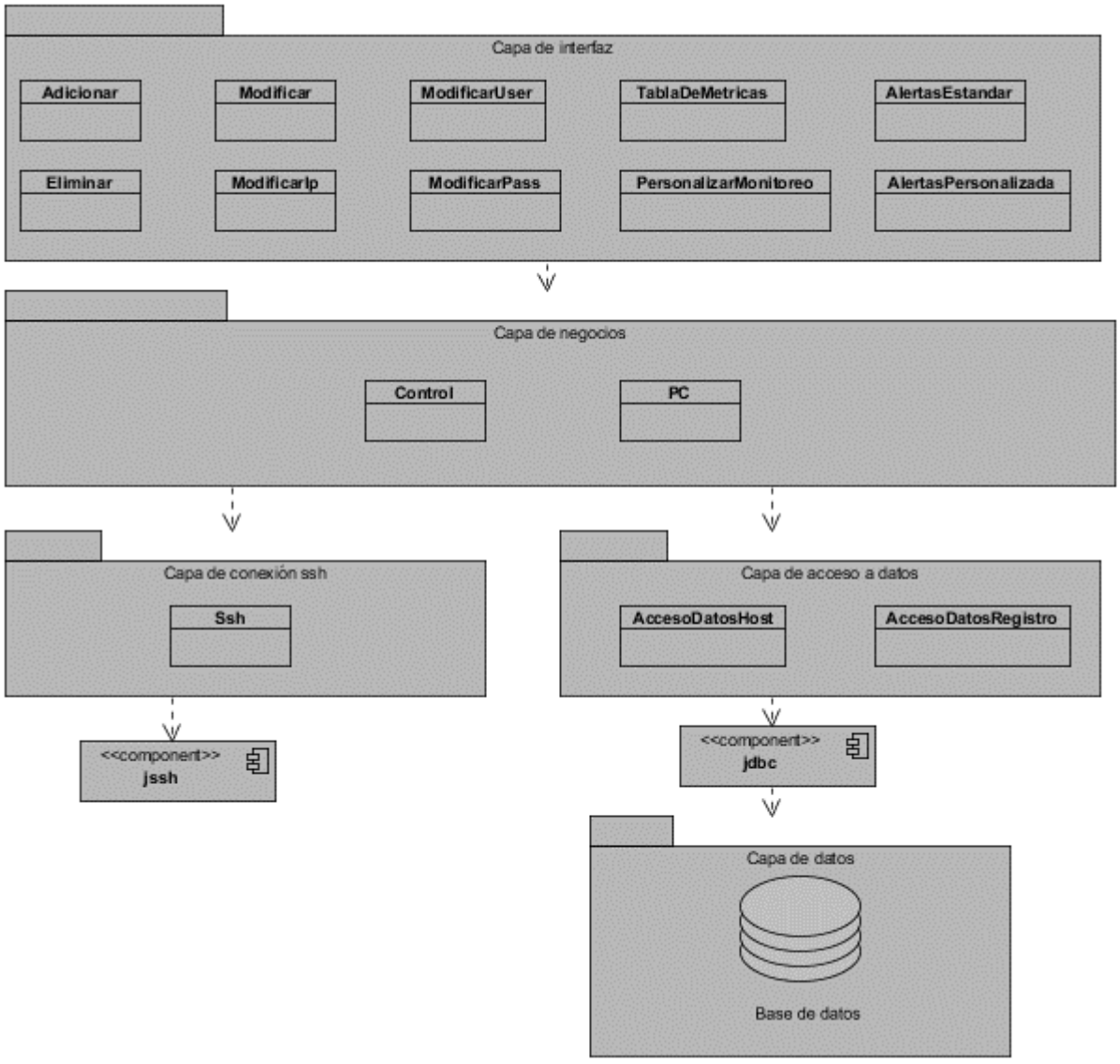


Figura 4. Representación de la arquitectura (Elaboración propia).

2.5.4. Patrones de diseño

Patrones Generales de Software para Asignación de Responsabilidades (GRASP)

Experto

El patrón Experto o Experto en Información es el que define a las clases u objetos encargados de realizar una labor especifica ya que están diseñados para ellos y cuentan con la información necesaria para desempeñar

esa labor. En este caso la clase *Ssh.java* evidencia el patrón, ya que su función es ejecutar comandos a través de una conexión SSH médiate un usuario, un IP, y una contraseña y posteriormente retornar la salida en la consola [44].

```
public class ssh {

    /** Constante que representa un enter ...3 lines */
    private static final String ENTER_KEY = "n";
    /** Sesión SSH establecida ...3 lines */
    private Session session;

    /** Establece una conexión SSH ...13 lines */
    public void connect(String username, String password, String host, int port)
        throws JSchException, IllegalAccessException {...15 lines }

    /** Ejecuta un comando SSH ...16 lines */
    public final String executeCommand(String command)
        throws IllegalAccessException, JSchException, IOException {...31 lines }

    /** Cierra la sesión SSH ...3 lines */
    public final void disconnect() {
        this.session.disconnect();
    }
}
```

Figura 5. Ejemplo del patrón experto en la clase Ssh.java (Elaboración propia).

Creador

El patrón creador tiene la responsabilidad de crear nuevos objetos ya que contiene la información de dicho objeto. La clase que se encarga de esta acción es la clase *PC.java*, ya que esta puede encargarse de la creación de objetos [44].

```
public class PC {

    private String ip;
    private String user;
    private String pass;

    public PC() {
        ip = "";
        user = "";
        pass = "";
    }

    public PC(String ip, String user, String pass) {
        this.ip = ip;
        this.user = user;
        this.pass = pass;
    }
```

Figura 6. Ejemplo del patrón creador en la clase PC.java (Elaboración propia).

Controlador

El patrón controlador tiene la responsabilidad de manejar los eventos y en consecuencia de estos ejecutar acciones. Actúa como una especie de intermediario entre la interfaz y las clases especializadas. La clase que cumple la función del patrón controlador y por ende ocupa la responsabilidad de controlar el flujo del programa, es la clase *Control.java*. Esta actúa en consecuencia de las acciones y eventos externos del programa ejecutando la acción necesaria en cada momento [44].

```
public ArrayList<String> obtenerValoresRegistros(String item1, String item2) {...41 lines

public String[][] monitorear() {...17 lines }

private static String ejecutarSSH(String user, String pass, String host, String comando)

public ArrayList<PC> llenar() {...29 lines }

public Boolean verificarLLave(String key) {
    boolean keyflag = false;
    for (int i = 0; i < ListaHost.size(); i++) {
        if (ListaHost.get(i).getIP().equals(key)) {
            keyflag = true;
        }
    }
    return keyflag;
}
```

Figura 7. Ejemplo del patrón controlador en la clase Control.java (Elaboración propia).

Bajo acoplamiento

El uso de este patrón se garantiza en la aplicación basándose en la propia arquitectura del sistema, ya que en la arquitectura en capas la dependencia entre las clases es mínima puesto que solamente las clases de una capa se pueden comunicar con las de la capa inmediatamente inferior [44].

```
private static String ejecutarSSH(String user, String pass, String host, String comando) {
    String result = "";
    try {
        Ssh sshConnector = new Ssh();
        if (!sshConnector.connect(user, pass, host, 22)) {
            result = "Sin Conexion";
        } else {
            result = sshConnector.executeCommand(comando);
            sshConnector.disconnect();
        }
```

Figura 8. Ejemplo del patrón bajo acoplamiento en la clase Control.java (Elaboración propia).

Alta cohesión

El patrón alta cohesión se observa cuando una clase tiene la responsabilidad de realizar una labor dentro del sistema, no desempeñada por el resto de los componentes del diseño. Este patrón se evidencia en la aplicación en conjunto con el patrón bajo acoplamiento, de forma tal que cada clase realice sus acciones y se evita que otra clase realice acciones correspondientes a la clase con la que está relacionada [44].

```
public void ConectarBaseDatos() {
    Connection c = null;
    try {
        Class.forName("org.postgresql.Driver");
        c = DriverManager
                .getConnection("jdbc:postgresql://localhost:5432/BaseDatos",
                        "postgres", "1234");
    } catch (Exception e) {
        e.printStackTrace();
        System.err.println(e.getClass().getName() + ": " + e.getMessage());
        System.exit(0);
    }
}
```

Figura 9. Ejemplo del patrón alta cohesión en la clase AccesoDatosHost.java (elaboración propia)

Patrones GoF aplicados

Creacionales: Método de Fábrica (del inglés Factory Method)

Provee un interfaz para crear objetos donde se delega esa acción a otras clases. Esto permite devolver una instancia de esa clase en consecuencia de los datos que reciba. Además, centraliza el sitio donde se crean los objetos permitiendo una mayor organización. En este caso la clase *Adicionar.java* utiliza el patrón Método de Fábrica ya que esta proporciona una interfaz donde el usuario puede ingresar los campos de cada PC y posteriormente dichos datos son almacenados en una base de datos y a través de otra clase se crea una instancia de la clase PC haciendo posible manejar mejor los datos.

```
    public Adicionar(java.awt.Frame parent, boolean modal) {...5 lines }

    public Adicionar(java.awt.Frame parent, boolean modal, Listado listado) {...4 lines }
    /** This method is called from within the constructor to initialize the form ...5 lines */
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {...15 lines }

    private void jPasswordField1ActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

    /**...3 lines */
    public static void main(String args[]) {...38 lines }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JPasswordField jPasswordField1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
```

Figura 10. Ejemplo del patrón Creacional: Método de Fábrica en la clase Control.java (Elaboración propia).

Estructurales: Fachada

Provee una interfaz unificada y simple que permite acceder a una interfaz o grupo de interfaces del sistema. La clase *Principal.java* es la encargada de desempeñar esta función al ofrecer una interfaz simple pero intuitiva desde la cual se puede acceder a todas las funcionalidades del programa.

```
public class Principal extends javax.swing.JFrame {
    DefaultListModel lista;
    Listado listado;

    static void Actualizarlista() {...3 lines }

    /** Creates new form Principal ...3 lines */
    public Principal() {...7 lines }

    public void ActualizarLista(){...5 lines }

    /** This method is called from within the constructor to initialize the form ...5 lines */
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {...5 lines }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {...5 lines }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {...5 lines }

    private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {...4 lines }

    private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {...4 lines }
```

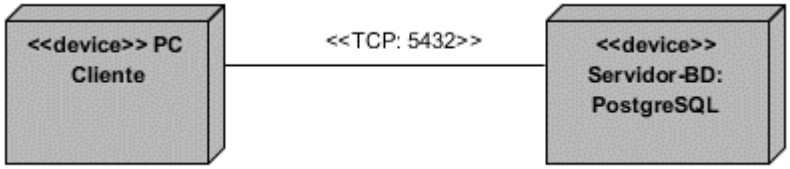
Figura 11. Ejemplo del patrón estructural: Fachada en la clase Principal.java (elaboración propia)

2.6. Modelo de despliegue

Un modelo de despliegue consiste en una representación estructural de la arquitectura del sistema desde el punto de vista de la distribución de los artefactos del software en los destinos de despliegue; definiendo a los artefactos como representaciones de elementos concretos en el mundo físico que son el resultado de un proceso de desarrollo [45].

En la **Figura 12**, se muestra el diagrama de despliegue propuesto para la herramienta. El mismo, muestra la disposición física de los nodos que componen el sistema y el reparto de los componentes en dichos nodos.

Figura 12. Modelo de despliegue (Elaboración propia).



Este diagrama se considera importante para lograr un despliegue exitoso de la herramienta. En él, se definen

las estaciones de trabajo (Dispositivo Cliente: PC de escritorio o PC portátil) que el usuario utilizará para ejecutar la herramienta. Este dispositivo debe tener como sistema operativo GNU/Linux y correctamente configuradas todas las dependencias SSH. La estación de trabajo se conecta mediante el protocolo TCP a través del puerto 5432 al servidor de Base de Datos PostgreSQL.

2.7. Conclusiones parciales

Al finalizar el presente capitulo se arribó a las siguientes conclusiones parciales:

- El análisis del Modelo de Dominio ayudó en la comprensión de cómo entender más cómo funciona el negocio sin la solución, y cómo podría ayudar a mejorar este proceso.
- Mediante la propuesta de solución se expuso una revisión detallada de cada una de las funcionalidades que ofrece el módulo.
- La recopilación de los requisitos tanto funcionales como no funcionales recogidos para el desarrollo de la solución permitieron definir las características principales del producto.
- El desarrollo y confección de los artefactos arrojados por la metodología escogida por el equipo de trabajo, tales como historias de usuarios, tarjetas CRC y el modelo de datos permitieron documentar el proceso de desarrollo.
- La selección de la arquitectura, así como los patrones utilizados en la evolución del software posibilitaron la obtención y validación de un diseño con mayor calidad.

CAPÍTULO 3. IMPLEMENTACIÓN Y VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

3.1. Introducción

En el presente capítulo se exponen las tareas de ingeniería que han sido generadas en la fase de desarrollo de la metodología de software utilizada. Se define los estándares de codificación que se estará utilizando en el desarrollo de la solución para garantizar una buena comprensión del código desarrollado. Además, se detallan las pruebas efectuadas al software cuyo objetivo es detectar y corregir el máximo de errores en la herramienta antes de ser entregada al cliente.

3.2. Fase de desarrollo

En esta fase se realiza la codificación o implementación de la herramienta, donde se da cumplimento al plan de iteraciones definidas en la fase de diseño. En esta fase se realiza la implementación de las HU que fueron seleccionadas en cada iteración, además se crean las tareas de ingeniería para organizar la implementación exitosa de las HU. Al finalizar esta fase es posible comenzar a realizarle las pruebas a la herramienta.

3.2.1. Tareas de ingeniería por iteraciones

Las tareas de ingeniería son las actividades que conforman las historias de usuarios y que favorecen su implementación. En la **Tabla 12** se muestran las tareas a desarrollar para la implementación de cada historia de usuario.

Tabla 12. Tareas de Ingeniería (Elaboración propia).

Iteración	Historia de Usuario	Tarea de Ingeniería
1	Realizar monitoreo	Obtener alertas del monitoreo
2	Mostrar registro	Mostrar registro de métricas
	Modificar frecuencia	Modificar frecuencia de los monitoreos personalizados
	Modificar alertas	Modificar alertas emitidas por los monitoreos personalizados

3	Filtrar registros	Filtrar la búsqueda de registros de monitoreos
	Añadir host	Añadir host a la base de datos
	Eliminar host	Eliminar host de la base de datos
	Modificar IP	Modificar el IP de la base de datos
	Modificar usuario	Modificar el usuario de la base de datos
	Modificar contraseña	Modificar la contraseña de la base de datos
	Modificar todo	Modificar todos los parámetros de la base de datos

3.2.2. Tareas detalladas

En la **Tabla 13** se muestra la tarea de ingeniería Obtener alertas del monitoreo. Las demás se encuentran en los anexos.

Tabla 13. Tarea de ingeniería: Realizar Monitoreo (Elaboración propia).

Tarea de ingeniería	
Número de tarea: 1	**Historia de usuario:** Realizar Monitoreo
Nombre de la tarea: Obtener alertas del monitoreo	
Tipo de tarea: Desarrollo	**Puntos estimados(días):** 7
Fecha de inicio: 1/3/2018	**Fecha de fin:** 7/3/2018
Programador responsable: Luis Ramiro Mendiburo Martínez	
Descripción: Implementar una funcionalidad que conocido el IP usuario y contraseña de un grupo de hosts permita, mediante la biblioteca jsch, realizar una conexión SSH y ejecutar una serie de comandos en consola para obtener los valores de las métricas. Luego compara estos valores con los umbrales de las	

métricas. En caso de encontrarse en los estados definidos para las alertas notifica al usuario mostrando la hora y fecha del monitoreo, así como el IP del host, recurso, valor de la métrica y el estado. En caso de ser un monitoreo personalizado, este proceso debe repetirse cada vez que transcurra el tiempo especificado, mostrando las alertas definidas por el usuario.

Los valores de los monitoreos: fecha, hora, IP, porciento de uso de CPU, porciento de uso de RAM y porciento de uso de almacenamiento deben ser guardados en una base de datos para su posterior uso.

3.2.3. Estándares de codificación

La metodología escogida para el desarrollo de la herramienta resalta que la comunicación de los programadores es a través del código. Es por ello que resulta fundamental que se sigan ciertos estándares de programación para lograr un entendimiento. De esta forma cualquier persona del equipo de desarrollo puede realizar modificaciones sobre el código de la aplicación. Además, se hace preciso que el código sea entendible para que posteriormente pueda reutilizarse el código.

Definición de clases

Para la definición de las clases se utiliza el estilo de codificación *"UpperCamelCase".* Este estilo establece que los nombres de las clases inician siempre con letra mayúscula. En caso de que el nombre de la clase posea más de una palabra, la primera letra de cada palabra se pondrá en mayúscula y el resto en minúscula.

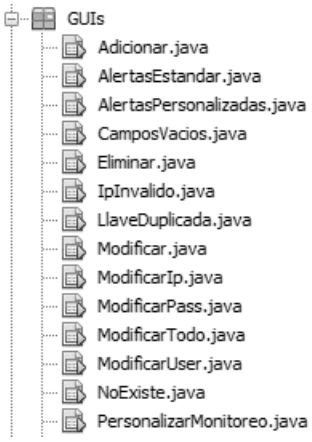


Figura 13. Ejemplo del estilo de codificación UpperCamelCase (Elaboración propia).

Métodos

Los nombres de los métodos se representan en minúscula, en caso de ser un nombre compuesto, la inicial de la primera palabra se simboliza en minúscula, y la de las otras palabras que lo componen en mayúscula. Se utiliza el estilo de codificación “lowerCamelCase”, el cual establece que los nombres inician con letra minúscula y cada nueva palabra debe iniciar con mayúscula.

```
public Control() throws IOException{...14 lines }

public void conectarBaseDatos() {...15 lines }

public void crearBaseDatos() {...17 lines }

  public void crearTablas() {...32 lines }

  public void ejecutarCryptoLab() {...19 lines }
```

Figura 14. Ejemplo de declaración de métodos en el código (Elaboración propia).

Variables

Los nombres de las variables son cortos, pero con significados lógicos, capaces de permitir identificar su función. Se utiliza el estilo de codificación “lowerCamelCase”, el cual establece que los nombres inician con letra minúscula y cada nueva palabra debe iniciar con mayúscula.

```
private String ip;
private String user;
private String pass;
```

Figura 15. Ejemplo de declaración de variables en el código (Elaboración propia).

Importaciones

Las importaciones en el código se encuentran en líneas separadas.

```
import com.jcraft.jsch.JSchException;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.HashSet;
import java.util.Set;
```

Figura 16. Ejemplo de uso de importaciones en el código (Elaboración propia).

3.3. Fase de prueba

Para el desarrollo de cualquier solución informática es necesario para la validación un gran cúmulo de pruebas. Estas pruebas normalmente se ejecutan en varias ocasiones y se ven afectadas por los cambios que se introducen conforme se va desarrollando la solución. La metodología XP divide las pruebas en dos grupos: pruebas de aceptación o pruebas funcionales diseñadas por el cliente final, destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida y pruebas unitarias, encargadas de verificar el código y diseñadas por los programadores, además, descomponen las funciones del programa en comportamientos comprobables discretos que se pueden probar como unidades individuales.

3.3.1. Prueba de caja blanca

Las pruebas de caja blanca se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. Se escogen distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa cerciorándose que se devuelvan los valores de salida adecuados [42].

Una técnica de las pruebas unitarias es la prueba del camino básico. Esta técnica permite obtener la complejidad lógica de un procedimiento o algoritmo y usar esta como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa [46].

La complejidad ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa la técnica camino básico, el valor calculado como complejidad ciclomática define el número de caminos independientes. Los caminos independientes no son

más que cualquier camino del programa que introduce un nuevo conjunto de sentencias de proceso o una nueva condición, del conjunto básico, o sea el conjunto de caminos independientes, de un programa. Además, proporcionan un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez [46].

Para las pruebas de caja blanca se seleccionó la técnica del camino básico, junto a la métrica complejidad ciclomática.

Para llevar a cabo esta técnica se deben realizar los siguientes pasos:

- Representar el programa en un grafo de flujo
- Calcular la complejidad ciclomática
- Determinar el conjunto básico de caminos independientes
- Derivar los casos de prueba que fuerzan la ejecución de cada camino

Para la realización de esta prueba se seleccionó el método verificarIP() de la clase Adicionar.java. Este método se encarga de validar un IP para ser añadido como campo de un host.

```
private boolean verificarIP(String ip) {
    if(ip.equals("localhost")){
return true;
    }
    boolean band=true;
    String [] arr=ip.split("\\.");
    if(arr.length!=4){
    band=false;
    }
    else {
    for (int s = 0; s < arr.length; s++) {
        if(isNumeric(arr[s])){
            if(Integer.parseInt(arr[s])>255){
            band=false;
            }}
        else return false;
        }
    }
    return band;
    }
```

Figura 17. Método verificarIP() de la clase Adicionar.java (Elaboración propia).

Representación del método en un grafo de flujo

Grafo de flujo para el método analizado:

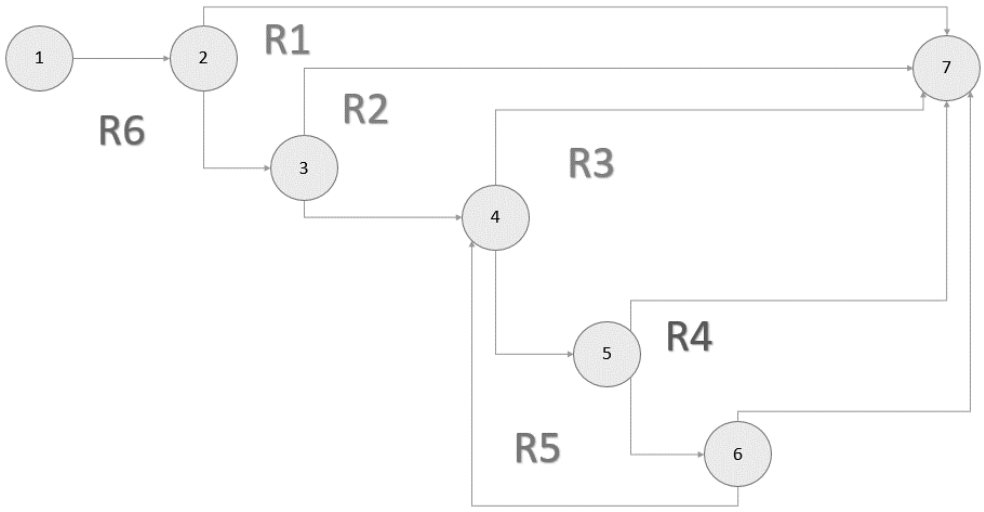


Figura 18. Grafo de flujo del método verificarIP() (Elaboración propia).

Complejidad ciclomática

Existen varias formas de calcular la complejidad ciclomática de un programa a partir de un grafo de flujo:

1. $(G) = A - N + 2$

A: número de aristas

N: número de nodos

$V(G) = (11 - 7) + 2 = 6$

2. $V(G) = P + 1$

P: número de nodos predicados[6]

$V(G) = 5 + 1 = 6$

3. $V(G) = R$

R: número de regiones del grafo

[6] Nodos de los cuales parten dos o más aristas.

$V(G) = 6$

Luego de haber calculado mediante las tres fórmulas se obtuvo que la complejidad ciclomática es seis, lo cual indica que existen seis caminos independientes. Seis, representa el mínimo número de casos de pruebas.

Conjunto básico de caminos independientes

Los caminos independientes resultantes son:

Camino 1: 1-2-7

Camino 2: 1-2-3-7

Camino 3: 1-2-3-4-7

Camino 4: 1-2-3-4-5-7

Camino 5: 1-2-3-4-5-6-7

Camino 6: 1-2-3-4-5-6-4-7

Casos de prueba

A continuación, se muestran algunos casos de pruebas para los caminos 1, 2, 4 y 6 por ser de los más importantes.

Tabla 14. Caso de prueba camino 1 (Elaboración propia).

Caso de prueba camino 1	
Condición de ejecución	El IP introducido es “localhost”
Entrada	IP =”localhost”
Resultados esperados	Se valida el IP y permite ingresarlo correctamente a la base de datos

Tabla 15. Caso de prueba camino 2 (Elaboración propia).

Caso de prueba camino 2	
Condición de ejecución	El IP introducido no es “localhost” y no tiene el formato (xxx.xxx.xxx.xxx)
Entrada	IP =”10.0.0.0.0.2”
Resultados esperados	Se invalida el IP y se informa al usuario del error: IP invalido.

Tabla 16. Caso de prueba camino 4 (Elaboración propia).

Caso de prueba camino 4	
Condición de ejecución	El IP introducido no es “localhost”, tiene el formato (xxx.xxx.xxx.xxx), algún valor que forma el IP no es numérico.
Entrada	IP =”10.0.o.5”
Resultados esperados	Se invalida el IP y se informa al usuario del error: IP invalido.

Tabla 17. Caso de prueba camino 6 (Elaboración propia).

Caso de prueba camino 6	
Condición de ejecución	El IP introducido no es “localhost”, tiene el formato (xxx.xxx.xxx.xxx), todos los valores son numéricos y están entre 0 y 255.
Entrada	IP=”10.0.0.1”

Resultados esperados	Se valida el IP y permite ingresarlo correctamente a la base de datos

Análisis de los resultados

El resultado de haber aplicado esta prueba fue satisfactorio. Esto evidencia la lógica y estabilidad del código. Para la verificación del código generado en el desarrollo de la herramienta se seleccionaron varios de los principales métodos aplicados. Los métodos seleccionados fueron: verificarIP(), analizado anteriormente y los métodos verificarLlave() y monitorear() que se encuentran en los anexos.

3.3.2. Prueba de caja negra

Según lo definido por Pressman, las pruebas de caja negra se llevan a cabo sobre la interfaz del software. Se trata de demostrar que las funciones del software son operativas, que las entradas se manejan de forma adecuada y que se produce el resultado esperado. [46]

Existen múltiples técnicas para la realización de estas pruebas. Una de ellas es la técnica de partición de equivalencia la cual permite comprobar los valores válidos e inválidos de las entradas existentes en la aplicación. En esta técnica los datos de entrada y los resultados de salida se agrupan en clases diferentes, en las que todos los miembros de dicha clase están relacionados Cada una de estas clases es una partición de equivalencia en la que el programa se comporta de la misma forma para cada miembro de la clase Se supone que la prueba de un valor representativo de cada clase sea equivalente a la prueba de cualquier otro valor [47].

Como resultado final de las pruebas funcionales, se obtuvo, en una primera iteración un total de ocho no conformidades, dos de ortografía, cuatro de funcionalidad y dos de validación. De estas, se resolvieron cuatro, y cuatro quedaron pendientes. En una segunda iteración, no se identifican nuevas no conformidades y de las cuatro pendientes, solo dos se mantuvieron para la próxima iteración, donde fueron resueltas y no se detectaron nuevas no conformidades. En una cuarta iteración no se identificaron nuevas inconformidades, obteniéndose, resultados satisfactorios. La siguiente gráfica, muestra los resultados antes descritos:

Figura 19. Resultados de las pruebas funcionales.

3.3.3. Prueba de aceptación

El uso de cualquier producto de software tiene que estar justificado por las ventajas que ofrece. Sin embargo, es muy difícil determinar antes de empezar a usarlo si sus ventajas realmente justifican su uso. Las pruebas de aceptación resultan ser los mejores instrumentos para esta determinación. En esta prueba se evalúa el grado de calidad del software con relación a todos los aspectos relevantes para que el uso del producto se justifique.

La Junta Internacional de Cualificaciones de Pruebas de Software (ISTQB por sus siglas en inglés) define la “Aceptación” como: Pruebas formales con respecto a las necesidades del usuario, requerimientos y procesos de negocio, realizadas para determinar si un sistema satisface los criterios de aceptación que permitan que el usuario, cliente u otra entidad autorizada pueda determinar si acepta o no el sistema [48].

A continuación, uno de los casos de pruebas de aceptación, técnica seleccionada para llevar a cabo las pruebas de aceptación de sistema. Los demás casos de prueba de aceptación se encuentran en los anexos.

Tabla 18. Caso de prueba de aceptación 4(Elaboración propia).

Caso de prueba de aceptación		
Código: CPA 4	**Historia de Usuario:** HU 6 Insertar host	
Funcionalidad que se prueba: Añadir un host a la base de datos		
Condiciones de ejecución: En la interfaz principal se selecciona la opción Añadir host.		
Acción:	**Datos de entrada:**	**Resultado esperado**
El cliente introduce los datos del nuevo host dejando campos vacíos y selecciona la opción Añadir.	Ip=”10.0.0.1” Usuario=nulo Contraseña=nulo	El programa muestra una ventana con el mensaje: Campos Vacíos informando que existen campos vacíos y solicitando que se introduzcan todos los datos.
Evaluación de la prueba: Satisfactoria		

Análisis de los resultados

Se diseñaron un total de 20 casos de prueba de aceptación para comprobar que la respuesta de la herramienta era la esperada por el cliente. En la primera iteración en la ejecución de las pruebas se detectaron un total de 4 no conformidades de validación. En la segunda iteración fueron corregidas todas las inconformidades detectadas en la primera iteración y no se encontraron nuevas. Luego de haber sido corregidas todas las no conformidades, el cliente avaló la solución con la entrega del Certificado de Aceptación del Producto. En los anexos se encuentra una imagen de la carta de aceptación.

3.4. Resultados obtenidos

Al finalizar el presente capitulo se arribó a las siguientes conclusiones parciales:

- Se pudo organizar el trabajo en una secuencia lógica de pasos a través del desarrollo de las tareas de ingeniería correspondientes a cada historia de usuario.
- El estándar de codificación utilizado proporcionó un buen entendimiento del código y una mejor organización del mismo.

- Las pruebas efectuadas facilitaron detectar, documentar y corregir las no conformidades existentes en la herramienta.
- Al concluir el período de pruebas se obtuvo una aplicación que cumple de forma correcta con la totalidad de las funcionalidades esperadas por el cliente.

CONCLUSIONES

De manera general, la presente investigación concluyó con el desarrollo del Módulo de notificaciones para OpenNebula; que sirve de apoyo al proceso de control y monitoreo de los recursos desplegados por el gestor nube OpenNebula. El módulo fue desarrollado a partir de las características identificadas en los gestores nubes estudiados. La herramienta obtenida permite notificar en forma de alertas cuando cualquiera de los recursos monitoreados de la infraestructura de OpenNebula sobrepase un umbral prefijado.

Otros aspectos significativos que se pueden destacar son:

1. El análisis y la fundamentación teórica de los principales conceptos asociados a la investigación, permitió lograr una mayor comprensión del alcance de la investigación y esclarecer su objeto de estudio.
2. La sistematización del marco teórico de la investigación científica y del estado actual de los gestores nubes seleccionados, posibilitó determinar las métricas y los umbrales utilizados en la implementación de la herramienta.
3. La integración de diversas áreas del conocimiento como son: los sistemas digitales, la ingeniería y gestión de software, base de datos, programación, entre otras, permitió obtener el análisis, diseño e implementación del Módulo de alertas para OpenNebula.
4. La solución fue validada a partir de la definición correcta de una estrategia de pruebas, que permitió comprobar el correcto funcionamiento de la herramienta, a partir de los requerimientos definidos por el cliente.

RECOMENDACIONES

Para el desarrollo de futuras investigaciones relacionadas con la presente, se propone:

1. La integración de la herramienta en forma de plugin en la interfaz web Sunstone.
2. Adicionar nuevas métricas a monitorear que permita tener un mejor control de la infraestructura de OpenNebula.
3. Adicionar nuevas funcionalidades que permitan obtener más información del desempeño de los hosts monitoreados a partir de los datos recogidos.
4. Adicionar nuevas funcionalidades que permitan realizar acciones de forma activa cuando ocurra alguna alerta.

BIBLIOGRAFÍA

[1] Lilia Rosa García Perellada, Orisel González Cartaya, y Alfredo Moreno Trujillo, «Sistema de Gestión de Reportes para OpenNebula», Trabajo de Diploma, Instituto Superior Politécnico José Antonio Echeverría, La Habana, 2014.

[2] P. M. Mell y T. Grance, «The NIST definition of cloud computing», National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-145, 2011.

[3] «Software de gestión de la nube | Akamai». [En línea]. Disponible en: https://www.akamai.com/es/es/resources/cloud-management-software.jsp. [Accedido: 25-ene-2018].

[4] «FAQ – OpenNebula». [En línea]. Disponible en: https://opennebula.org/about/faq/#toggle-id-1. [Accedido: 25-ene-2018].

[5] Rob Boucher y Sunny Deng, «Supervisión de aplicaciones y recursos de Azure», *Microsoft Azure*. [En línea]. Disponible en: https://docs.microsoft.com/es-es/azure/monitoring-and-diagnostics/monitoring-overview. [Accedido: 26-jun-2018].

[6] «Diccionario de la lengua española - Edición del Tricentenario». [En línea]. Disponible en: http://dle.rae.es/?w=diccionario. [Accedido: 11-jun-2018].

[7] «Open Cloud Architecture — OpenNebula 5.4.13 documentation». [En línea]. Disponible en: http://docs.opennebula.org/5.4/deployment/cloud_design/open_cloud_architecture.html. [Accedido: 11-jun-2018].

[8] «Qué es UDP | Establecer una conexión a través del protocolo UDP». [En línea]. Disponible en: https://www.eltima.com/es/what-is-udp/. [Accedido: 11-jun-2018].

[9] «Monitoring — OpenNebula 5.4.13 documentation». [En línea]. Disponible en: http://docs.opennebula.org/5.4/deployment/open_cloud_host_setup/monitoring.html#mon. [Accedido: 11-jun-2018].

[10] «¿Qué es Azure? Servicio en la nube de Microsoft | Microsoft Azure». [En línea]. Disponible en: https://azure.microsoft.com/es-es/overview/what-is-azure/. [Accedido: 05-abr-2018].

[11] «IBM Knowledge Center - Visión general de IBM Service Delivery Manager». [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SSBH2C_7.2.2/com.ibm.isdm_7.2.2.doc/c_productoverview.html. [Accedido: 16-feb-2018].

[12] «IBM Knowledge Center - Informes de supervisión». [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SSBH2C_7.2.2/com.ibm.isdm_7.2.2.doc/c_monitoring_reports.html. [Accedido: 06-abr-2018].

[13] «vCloud Suite | Cloud Suite», *VMWare*. [En línea]. Disponible en: https://www.vmware.com/es/products/vcloud-suite.html. [Accedido: 16-feb-2018].

[14] «Apache Cloudstack», *Apache Cloudstack*. [En línea]. Disponible en: https://cloudstack.apache.org/about.html. [Accedido: 16-feb-2018].

[15] «What is OpenStack?», *Opensource.com*. [En línea]. Disponible en: https://opensource.com/resources/what-is-openstack. [Accedido: 16-feb-2018].

[16] «IBM Knowledge Center - Umbrales de rendimiento». [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SS5R93_5.2.10/com.ibm.spectrum.sc.doc/fqz0_c_thresholds.html. [Accedido: 05-jun-2018].

[17] «Ajustar umbrales de rendimiento de PRO». [En línea]. Disponible en: https://technet.microsoft.com/es-es/library/ee423768.aspx. [Accedido: 05-jun-2018].

[18] «Supervisión de procesadores (Windows Server 2008 y Windows Server 2003)». [En línea]. Disponible en: https://technet.microsoft.com/es-es/library/dd279711.aspx. [Accedido: 05-jun-2018].

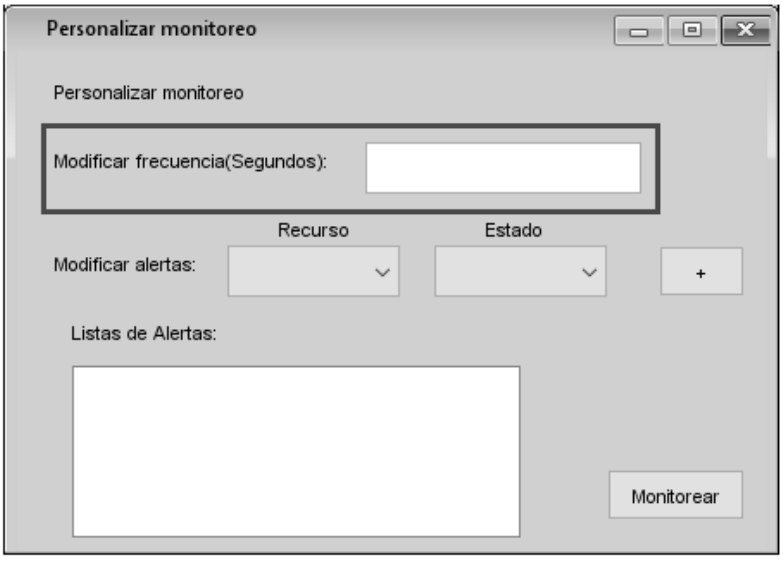
[19] «Porcentaje de uso total de la CPU - Microsoft.Windows.Server.2003.OperatingSystem.TotalCPUUtilization (UnitMonitor) (ESN)». [En línea]. Disponible en: https://systemcenter.wiki/?GetElement=Microsoft.Windows.Server.2003.OperatingSystem.TotalCPUUtilization&Type=UnitMonitor&ManagementPack=Microsoft.Windows.Server.2003&Version=6.0.7323.0&Language=ESN. [Accedido: 05-jun-2018].
[20] «Umbrales», 16-abr-2004. [En línea]. Disponible en: http://publib.boulder.ibm.com/tividd/td/ITM/SH19-4569-03/es_ES/HTML/dmumst26.htm. [Accedido: 05-jun-2018].
[21] «Funcionamiento del uso de los recursos del sistema del análisis bajo demanda de VirusScan Enterprise». [En línea]. Disponible en: https://kc.mcafee.com/corporate/index?page=content&id=KB55145&locale=es_ES&viewlocale=es_ES. [Accedido: 05-jun-2018].
[22] «Configurar la memoria disponible para las aplicaciones del servidor de informes». [En línea]. Disponible en: https://msdn.microsoft.com/es-es/library/ms159206(v=sql.120).aspx. [Accedido: 05-jun-2018].
[23] «Supervisión de los discos lógicos y físicos». [En línea]. Disponible en: https://technet.microsoft.com/es-es/library/dd262028.aspx. [Accedido: 05-jun-2018].
[24] «Monitorización del uso del disco». [En línea]. Disponible en: https://docs.plesk.com/es-ES/onyx/administrator-guide/76190/. [Accedido: 05-jun-2018].
[25] I. Sommerville, *Software engineering*, 9th ed. Boston: Pearson, 2011.
[26] Ing. José Joskowicz, «Reglas y Prácticas en eXtreme Programming». 10-feb-2008.
[27] «Herramientas CASE», *MindMeister*. [En línea]. Disponible en: https://www.mindmeister.com/742289673/herramientas-case. [Accedido: 11-jun-2018].
[28] «Visual Paradigm Product Overview». [En línea]. Disponible en: https://www.visual-paradigm.com/support/documents/vpuserguide/12/13/5963_visualparadi.html. [Accedido: 19-mar-2018].
[29] Alonso Martínez, Heydi, Soler Arcia, Daneidys, y Reyes Hernández, Kenia, «Desarrollo del Módulo de Reportes del Sistema Integrado de Gestión Bibliotecaria Koha para la Biblioteca Nacional de Cuba José Martí.», Trabajo de diploma, Universidad de las Ciencias Informáticas, La Habana, 2010.
[30] «Introducción al lenguaje Java». 2013-2012.
[31] «Bash - openSUSE». [En línea]. Disponible en: https://es.opensuse.org/Bash. [Accedido: 12-may-2018].
[32] «JSch - Java Secure Channel». [En línea]. Disponible en: http://www.jcraft.com/jsch/. [Accedido: 15-may-2018].
[33] «Java SE Technologies - Database». [En línea]. Disponible en: http://www.oracle.com/technetwork/java/javase/jdbc/index.html. [Accedido: 30-may-2018].
[34] «PostgreSQL-ES | Emc2Net». [En línea]. Disponible en: https://e-mc2.net/es/postgresql-es. [Accedido: 19-mar-2018].
[35] «PostgreSQL: Documentation: 9.4: pgcrypto». [En línea]. Disponible en: https://www.postgresql.org/docs/9.4/static/pgcrypto.html. [Accedido: 05-jun-2018].
[36] «pgAdmin III — pgAdmin III 1.22.2 documentation». [En línea]. Disponible en: https://www.pgadmin.org/docs/pgadmin3/1.22/. [Accedido: 19-mar-2018].
[37] «NetBeans IDE - Overview». [En línea]. Disponible en: https://netbeans.org/features/index.html. [Accedido: 13-mar-2018].
[38] I. Sommerville, *Ingeniería del software*. Pearson Educación, 2005.
[39] P. Letelier y M. C. Penadés, «Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)», *www.cyta.com.ar/ta0502/v5n2a1.htm*, 15-abr-2006. [En línea]. Disponible en: http://www.cyta.com.ar/ta0502/v5n2a1.htm. [Accedido: 19-mar-2018].

[40] S. Casas y H. Reinaga, «Identificación y modelado de aspectos tempranos dirigido por tarjetas de responsabilidades y colaboraciones», presentado en XIV Congreso Argentino de Ciencias de la Computación, 2008.
[41] «Triple Data Encryption Standard (Triple-DES)». [En línea]. Disponible en: https://www.vocal.com/cryptography/tdes/. [Accedido: 11-jun-2018].
[42] «Ingenieria de Software Un Enfoque Practico.6th.edicion-.Roger.pressman.1», *Scribd*. [En línea]. Disponible en: https://es.scribd.com/doc/27182020/Ingenieria-de-Software-Un-Enfoque-Practico-6th-edicion-Roger-pressman-1. [Accedido: 02-jun-2018].
[43] «Arquitectura basada en capas. – Juan Pelaez Web Site». .
[44] «Patrones de GRASP | adictosaltrabajo». .
[45] Sarmiento J, «UML: Diagrama de despliegue. Obtenido de Visión general de los diagrama de despliegue.», 2016. [En línea]. Disponible en: http://umldiagramadespliegue.blogspot.com/.
[46] R. S. Pressman, *Ingeniería del software: un enfoque práctico*. México: McGraw-Hill, 2010.
[47] «Particion de Equivalencia». [En línea]. Disponible en: https://www.fing.edu.uy/inco/cursos/ingsoft/pis/memoria/dvd01/experiencia2005/MUM/protest/guias/guias/particionE.htm. [Accedido: 08-jun-2018].
[48] Edgardo Rojas, «Plan de pruebas de software», 08:28:28 UTC.

ANEXOS

Anexo 1

Historia de usuario # 2 (Elaboración propia).

Historia de Usuario	
Código: HU2	**Nombre Historia de Usuario:** Modificar frecuencia
Referencia: RF2	
Usuario: empleado	
Programador: Luis Ramiro Mendiburo Matinés	**Iteración Asignada:** 2
Prioridad en Negocio: Media	**Tiempo Estimado:** 3 días
Riesgo en Desarrollo: Media	**Tiempo Real:** 3 días
Descripción: Desde una ventana generada por el botón *Personalizar monitoreo* ubicado en la ventana principal, se accede a una ventana donde se puede variar algunos de los parámetros de monitoreo. Mediante una casilla descrita por la etiqueta *Modificar Frecuencia* se puede ingresar el tiempo que describe la frecuencia con la que se ejecutara el monitoreo.	
Observaciones: El valor del tiempo debe ser entero y positivo.	
Interfaz: 	

Historia de usuario # 3 (Elaboración propia).

Historia de Usuario	
Código: HU3	**Nombre Historia de Usuario:** Modificar alertas
Referencia: RF3	
Usuario: empleado	
Programador: Luis Ramiro Mendiburo Matinés	**Iteración Asignada: 2**
Prioridad en Negocio: Media	**Tiempo Estimado:** 3 días
Riesgo en Desarrollo: Media	**Tiempo Real:** 3 días
Descripción: Desde una ventana generada por el botón *Personalizar monitoreo* ubicado en la ventana principal, se accede a una ventana donde se puede variar algunos de los parámetros de monitoreo. En esta ventana la opción *Modificar alertas* permite agregar las alertas que se deseen recibir, el primer elemento: *Recurso* permite agregar el recurso que se desee monitorear y el elemento *Estado* define el estado que definirá la alerta emitida para el recurso seleccionado, luego se presiona el botón + y se agrega la alerta a la lista de alertas. En caso de querer agregar más alertas se procede de igual manera.	
Observaciones: Se debe añadir al menos una alerta.	
Interfaz:	

Anexo 2

Ssh.java (Elaboración propia).

Tarjeta CRC	
Clase: Ssh.java	
Descripción: realiza la conexión SSH	
Atributos	
Nombre	**Descripción**
Responsabilidades:	**Colaboraciones:**
Realizar las conexiones SSH a través de un IP, un usuario, y una contraseña y retorna el valor devuelto por la consola o “Sin Conexión” en caso de ocurrir algún error a la hora de realizar la conexión SSH	Control.java

Tarjeta CRC: Control.java (Elaboración propia).

Tarjeta CRC	
Clase: Control.java	
Descripción: Contiene el listado de host y opera sobre él.	
Atributos	
Nombre	**Descripción**
ArrayList<PC> ListaHost	Contiene el listado de host a monitorear
Responsabilidades:	**Colaboraciones:**
Contiene todos los host a monitorear y realiza operaciones sobre ellos.	Ssh.java PC.java AccesoDatosHost.java AccesoDatosRegistros.java Adicionar.java Eliminar.java Modificar.java ModificarIp.java ModificarUser.java ModificarPass.java ModificarTodo.java MostrarRegistro.java Modificar.java AlertasEstandar.java PersonalizarMonitore.java

Tarjeta CRC: Adicionar.java (Elaboración propia).

Tarjeta CRC	
Clase: Adicionar.java	
Descripción: Interfaz para introducir los datos de los host a monitorear.	
Atributos	
Nombre	**Descripción**
Responsabilidades:	**Colaboraciones:**
Muestra una interfaz en forma de formulario que permite a los usuarios introducir los datos de los host a monitorear.	Principal.java Control.java. CamposVacios.java IpIncorrecto.java LlaveDuplicada.java

Anexo 3

Tarea de ingeniería: Mostrar registro de métricas (Elaboración propia).

Tarea de ingeniería	
Número de tarea: 2	**Historia de usuario:** Mostrar registro
Nombre de la tarea: Mostrar registro de métricas	
Tipo de tarea: Desarrollo	**Puntos estimados(días):** 3
Fecha de inicio: 8/3/2018	**Fecha de fin:** 10/3/2018
Programador responsable: Luis Ramiro Mendiburo Martínez	

Implementar una funcionalidad que permita mostrar una tabla con la hora y fecha del monitoreo, así como los IPs de los hosts y los valores de cada una de las métricas y en consecuencia con el estado de cada recurso la casilla correspondiente tome un color definido por dicho estado. Además, se debe permitir al usuario filtrar los resultados a mostrar en la tabla, a través de la fecha y hora y el IP de un host en específico. Estos datos deben ser cargados desde una base de datos que almacene los datos de todos los registros realizados.

Tarea de ingeniería: Modificar frecuencia de los monitoreos personalizados (Elaboración propia).

Tarea de ingeniería	
Número de tarea: 3	**Historia de usuario:** Modificar frecuencia
Nombre de la tarea: Modificar frecuencia de los monitoreos personalizados	
Tipo de tarea: Desarrollo	**Puntos estimados(días):** 3
Fecha de inicio: 11/3/2018	**Fecha de fin:** 13/3/2018
Programador responsable: Luis Ramiro Mendiburo Martínez	
Descripción: Implementar una funcionalidad que permita al usuario ingresar la frecuencia con que quiere que se realicen los monitoreos, de esta forma cada vez que se cumpla el tiempo especificado el programa ejecutara el monitoreo automáticamente. No se debe permitir que el usuario deje campos vacíos.	

Tarea de ingeniería: Modificar alertas emitidas por los monitoreos personalizados (Elaboración propia).

Tarea de ingeniería	
Número de tarea: 4	**Historia de usuario:** Modificar alertas
Nombre de la tarea: Modificar alertas emitidas por los monitoreos personalizados	
Tipo de tarea: Desarrollo	**Puntos estimados(días):** 3

Fecha de inicio: 14/3/2018	**Fecha de fin:** 16/3/2018
Programador responsable: Luis Ramiro Mendiburo Martínez	
Descripción: Implementar una funcionalidad que permita al usuario modificar el tipo de alertas que quiera recibir especificando el recurso y el estado que desea conocer. No se debe permitir que el usuario deje este campo vacío a la hora de realizar un monitoreo personalizado. No se debe permitir que el usuario deje campos vacíos.	

Anexo 4

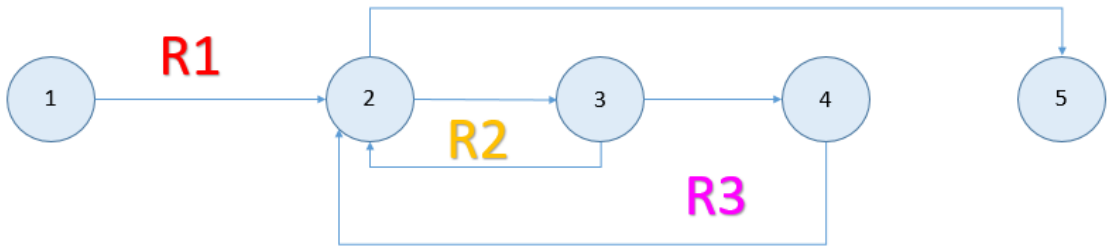


Diagrama de flujo del método verificarLlave().

Complejidad ciclomática

Existen varias formas de calcular la complejidad ciclomática de un programa a partir de un grafo de flujo:

1. $V(G) = A - N + 2$

A: número de aristas

N: número de nodos

$V(G) = (6 - 5) + 2 = 3$

2. $V(G) = P + 1$

P: número de nodos predicados

$V(G) = 2 + 1 = 3$

3. $V(G) = R$

R: número de regiones del grafo

$V(G) = 3$

Luego de haber calculado mediante las tres fórmulas, se obtuvo que la complejidad ciclomática es tres, lo cual indica que existen tres caminos independientes. Esto significa que tres representa el mínimo número de casos de pruebas.

Conjunto básico de caminos independientes

Los caminos independientes resultantes son:

Camino 1: 1-2-5

Camino 2: 1-2-3-2-5

Camino 3: 1-2-3-4-2-5

Casos de prueba

A continuación, se muestran los casos de pruebas.

Caso de prueba camino 1 (Elaboración propia).

Caso de prueba camino 1	
Condición de ejecución	La lista de hosts está vacía
Entrada	IP=”10.0.0.1”
Resultados esperados	Se retorna falso

Caso de prueba camino 2 (Elaboración propia).

Caso de prueba camino 2	
Condición de ejecución	El IP introducido ya se encuentra en la lista de hosts.
Entrada	IP=”10.0.0.1”

Resultados esperados	Se retorna falso

Caso de prueba camino 3 (Elaboración propia).

Caso de prueba camino 3	
Condición de ejecución	El IP introducido no se encuentra en la lista de hosts.
Entrada	IP =”10.0.0.2”
Resultados esperados	Se retorna verdadero

Anexo 5

Caso de prueba de aceptación 1 (Elaboración propia).

Caso de prueba de aceptación		
Código: CP1	**Historia de Usuario:** Modificar frecuencia	
Funcionalidad que se prueba: Modificar frecuencia de los monitoreos personalizados		
Condiciones de ejecución: En la interfaz principal se selecciona la opción Personalizar Monitoreo		
Acción:	**Datos de entrada:**	**Resultado esperado**
El cliente introduce los el valor del tiempo de la frecuencia de los monitoreos, selecciona las alertas y selecciona Monitorear.	Tiempo=-2	El programa reporto un error porque el tiempo no era mayor que 0 y dejo de funcionar
Evaluación de la prueba: Insatisfactoria		

Caso de prueba de aceptación 2 (Elaboración propia).

Caso de prueba de aceptación		
Código: CP2	**Historia de Usuario:** Modificar frecuencia	
Funcionalidad que se prueba: Modificar frecuencia de los monitoreos personalizados		
Condiciones de ejecución: En la interfaz principal se selecciona la opción Personalizar Monitoreo		
Acción:	**Datos de entrada:**	**Resultado esperado**
El cliente introduce las alertas y selecciona la opción Monitorear.	Tiempo=nulo	El programa muestra una ventana informando que existen campos vacíos y solicitando que se introduzcan todos los datos.
Evaluación de la prueba: Satisfactoria		

Anexo 6