Universidad de las Ciencias Informáticas

Facultad 1

Sistema de reportes para una nube privada.

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

Autor: Adrián Pérez Prieto

Tutores: Ing. Cecilia Esther Hernández Espinosa

1er Tte. Ing. Dany Esquijarosa Bonilla

La Habana,16 de junio de 2017
"Año 59 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro por este medio que yo Adrián Pérez Prieto, con carné de identidad 92102737222 soy el autor principal del trabajo titulado “Sistema de reportes para una nube privada” 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 firmamos la presente a los 16 días del mes de junio del año 2017.

Autor:

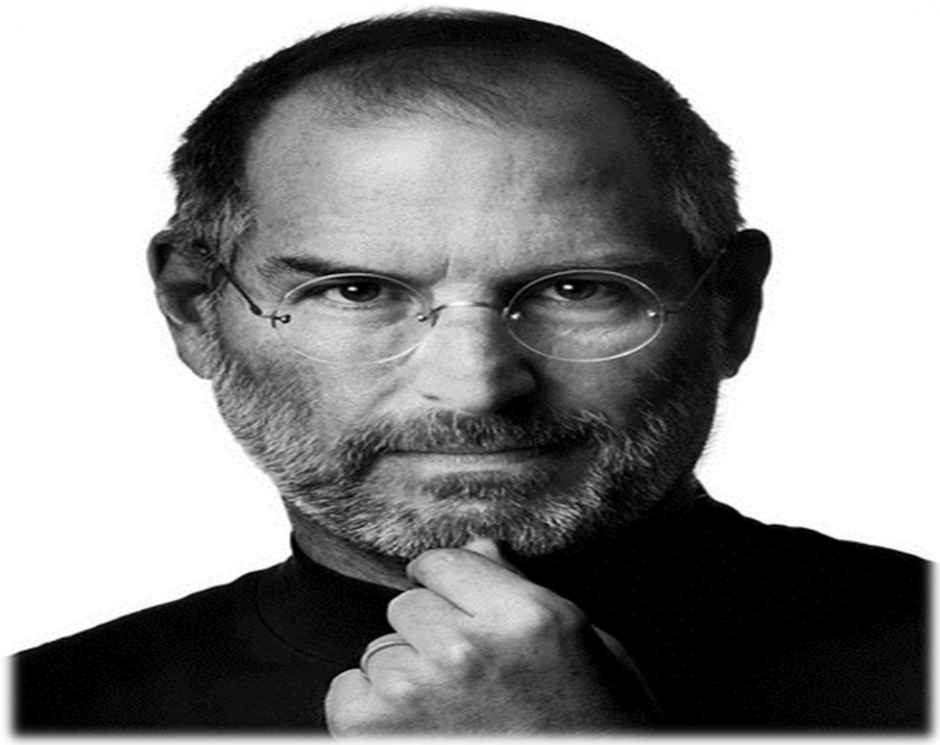
_____

Adrián Pérez Prieto

Tutores:

_____ _____

Ing. Cecilia Esther Hernández Espinosa 1er Tte. Ing. Dany Esquijarosa Bonilla



"Deja de malgastar tu vida viviendo la de otra persona. No sigas los dogmas, ya que solo sirven para vivir según el pensamiento de otros. No permitas que opiniones externas o el miedo al fracaso te impidan seguir a tu corazón. No tienes nada que perder ante la muerte. "

Steve Jobs

AGRADECIMIENTOS

Mi felicidad no estaría completa el día de hoy si me llego a olvidar de esas personas que estuvieron presentes en todo el camino transitado para ayudarme a obtener este resultado, por eso sería imperdonable dejar de agradecer a todos ellos.

Agradezco:

A mi mamá, papá, hermano y hermanas, mi abuela Zoila y todo el resto de la familia, por su apoyo, cariño, y confianza depositada en mí, por estar siempre pendientes y preocupados por mis estudios y mi vida personal, por ser las personas que cuando pensé irme de esta escuela y tuve que repetir un año, me dieron los mejores consejos que me guiaron el resto de la carrera.

A Rosmery, Rosa y Melchor por ser esa otra familia que te acoge en su casa y te trata como otro hijo más, por enseñarme cosas de la vida que me ayudaron mucho en estos últimos 3 años.

A mi tutora Cecilia, que a pesar del poco tiempo de conocernos depositó la responsabilidad del desarrollo de este trabajo tan importante para ella en mí, y por estar dispuesta siempre a ayudarme y colaborar en él.

A mis hermanos de batalla Jimmy, Robert y Horsford, por compartir los malos y buenos momentos vividos en esta universidad.

A Rafa, por ser la mente prodigiosa del piquete y estar dispuesto en todo momento a explicarme y ayudarme con el desarrollo de mi trabajo de diploma.

A todas las amistades y profesores que he conocido en esta hermosa y duradera estadía por la Habana que de una forma u otra han contribuido en mi formación como profesional y como mejor persona.

De forma general, a todos aquellos que aportaron su granito de arena e hicieron posible que hoy este culminado mis estudios y comenzando una nueva faena como Ingeniero Informático.

A todos ellos mis respetos y ***MUCHAS GRACIAS.***

RESUMEN

El reciente surgimiento de la Computación en la Nube impone nuevos retos en la gestión de los centros de procesamiento de datos convencionales. El objetivo de esta investigación fue desarrollar un sistema de reportes para una nube privada. Para ello se realizó una revisión bibliográfica con el fin de detectar la existencia de estándares o recomendaciones relacionados con el tema, propuestas por las principales organizaciones de estandarización. Además, fueron evaluados los sistemas de gestión de reportes de las soluciones de software libre *CloudStack, Cloud Foundry, EUCALYPTUS y OpenNebula* con el objetivo de determinar los tipos de reportes que generan, así como para determinar si permiten exportar dichos reportes. Se realizó una evaluación de las herramientas de monitoreo existentes, con vistas de determinar funcionalidades que debían estar presente en la propuesta de solución. Se define como metodología de desarrollo para guiar el trabajo en equipo *eXtreme Programming* siendo esta una metodología ágil que posibilita que el cliente este en constante intercambio con el equipo de desarrollo. Con toda la investigación antes expuesta se desarrolló un sistema que es capaz de cubrir gran parte de las necesidades de la entidad. Una vez culminado el proceso de implementación se le realizaron pruebas a la solución que validaron el correcto funcionamiento y el cumplimiento del objetivo planteado.

Palabras claves:

Centro de datos, nube, reporte, sistema.

TABLA DE CONTENIDOS

Índice de Ilustraciones

Índice de Tablas

INTRODUCCIÓN

En la actualidad el término computación en la nube tiene gran auge, propone soluciones escalables, flexibles y eficientes en cuanto al uso de los recursos para las instituciones. La computación en la nube es un modelo de negocio tecnológico donde un proveedor presta un servicio, en el cual las aplicaciones que administran la información de una organización y la infraestructura tecnológica reposan en más de un servidor virtual, de tal manera que el mantenimiento de dichas aplicaciones, la gestión y el acceso a los recursos compartidos se brinda como un servicio de computación de dos formas: bajo demanda o gratuito, o pago por uso ambas a través de internet, los cuales son procesados remotamente desde los servidores del proveedor (Víctor Zamora Yustres, 2014).

La ubicación física del hardware y de los dispositivos a los que se tiene acceso, normalmente no es conocida por el usuario final. Está diseñado a partir de nuevas tendencias, como la virtualización y las arquitecturas orientadas al servicio, y aprovechan la conectividad de las redes locales a fin de reducir el costo de los recursos de hardware para servicios informáticos, redes y almacenamiento (Víctor Zamora Yustres, 2014).

Un elemento fundamental en el despliegue de una nube privada es garantizar una sólida gestión de reportes en aras de lograr alta disponibilidad, tolerancia a fallos, escalabilidad, y un uso óptimo de los recursos. Los reportes brindan una visión efectiva del sistema y los servicios en la nube durante periodos de tiempo que pueden ser personalizados como diario, semanal, mensual, o trimestral. Ayudan también a realizar monitoreo basado en los datos de rendimiento que recogen, los cuales puede incluir análisis de patrones críticos, análisis de tendencias, planes de capacidad y auditorías de seguridad. Los reportes también proporcionan estadísticas sobre la disponibilidad y uso de las aplicaciones distribuidas, servidores y componentes específicos dentro de un servidor. (Amaris et al., 2010).

Un Sistema de Gestión de Reportes en una nube privada debe tener las siguientes funcionalidades: generación de reporte por usuarios y por grupos de usuarios, configuración de la frecuencia de los reportes, soporte para exportar la información a diferentes formatos, y permitir el acceso a usuarios y administradores a reportes de: contabilidad, eventos del sistema y uso de los recursos en los nodos (Garcia Perellada 2014).

En la actualidad existen proveedores que emplean diferentes soluciones para la gestión de la nube, entre estos proveedores destacan *Microsoft*, *VMWare* e *IBM*[1] con sus soluciones propietarias; y *CloudStack*, *OpenNebula* y EUCALYPTUS[2] con soluciones Software Libre Código Abierto (SLCA).

En investigaciones previas se pudo constatar que por lo general los gestores nubes en su interfaz de gestión solo brindan a los usuarios y administradores reportes de contabilidad. Estos reportes solo incluyen el consumo de red, recursos de CPU[3] y la memoria asignada, no muestran información sobre el uso de recursos como memoria o disco, tampoco cargo asociado a este consumo ni análisis sobre si se subutilizan o sobreutilizan los recursos de las MV[4] y en los nodos. Algunos permiten exportar los reportes y especificar la frecuencia con la que se generan. Los reportes son mostrados a través de la CLI[5] o en forma de gráfica a través de alguna interfaz de administración, pero no se envían alertas ni notificaciones del estado del sistema a través del correo electrónico.

Ante esta disyuntiva los administradores se ven obligados a utilizar alguna herramienta que complemente la gestión de los reportes. Esta alternativa trae como consecuencia la necesidad de familiarizarse con otro entorno de trabajo y una sobrecarga en el consumo de los recursos. Por otro lado, los reportes no se encuentran unificados lo que no contribuye a una gestión centralizada.

En tal sentido las empresas cubanas también se han sumado a brindar servicios en la nube y una de estas empresas es la XETID[6]. Esta empresa se encuentra trabajando en el desarrollo de una infraestructura que permita brindar servicios de computación en la nube a un grupo bien definido de usuarios. Para lograr este objetivo se impone una adecuada gestión de los recursos involucrados. Al igual que los restantes proveedores la XETID también enfrenta dificultades a la hora de lograr una gestión centralizada. Pudieran plantearse dos problemas fundamentales:

- El uso de la interfaz de gestión que trae el gestor nube no logra satisfacer las necesidades de información del centro de datos.

---

[1] Siglas en inglés para el término: *International Business Machines.*
[2] Siglas en inglés para el término: *Elastic Utility Computing Architecture for Linking your Programs to Useful Systems.*
[3] Siglas en inglés para el término: *Central Processor Unit*.
[4] Siglas para el término: *Máquina Virtual.*
[5] Siglas en inglés para el término: *Command Line Interface.*
[6] Siglas para el término: *Empresa de Tecnología e Información para la Defensa.*

- La utilización de una herramienta que complemente al gestor tampoco satisface las necesidades, pero obliga a destinar recursos de cómputo para esta labor y la curva de aprendizaje del personal encargado puede ser alta.

Ante estas dificultades la solución nube que implementa la XETID carece de una gestión que brinde información sobre el consumo de los recursos tanto de los nodos físicos como de los virtuales. También carece de los recursos de cómputos suficientes para destinarlos al despliegue de una herramienta complementaria y tampoco cuenta con el personal capacitado para atender este tipo de soluciones. Por otro lado, las herramientas de monitoreo y supervisión no son flexibles para ser integradas desde el código fuente al gestor nube.

Dada la **situación problemática** planteada anteriormente se extrajo como **problema de la investigación** la siguiente interrogante: ¿Cómo contribuir a la gestión de reportes para una nube privada?

Se establece como **objeto de estudio**, los reportes en nubes privadas centrando el **campo de acción** en los reportes de infraestructura de una nube privada.

Lo que permitirá darle cumplimiento al **objetivo general:** Desarrollar un sistema de reportes para la infraestructura en una nube privada.

El cumplimiento satisfactorio de lo antes expuesto contribuirá al desarrollo exitoso de la **idea a defender**: La implementación de un sistema de reportes para la infraestructura de una nube privada permitirá la supervisión de los recursos físicos y virtuales.

Para dar cumplimiento al objetivo general se definieron las siguientes **tareas de investigación**:

1. Caracterización de los sistemas de reportes para nubes privadas.
2. Definición de los requerimientos funcionales y no funcionales de la propuesta de solución.
3. Definición de la arquitectura de desarrollo.
4. Implementación de los requerimientos identificados de la propuesta de solución.
5. Validación de la propuesta de solución implementada.

Con el propósito de dar solución a las tareas antes mencionadas se utilizaron los siguientes métodos:

Métodos Teóricos

- **Analítico-Sintético:** Permitió identificar los principales conceptos, características y funcionalidades empleadas dentro de los sistemas de reportes para nubes privadas, analizando y estudiando documentos, trabajos de diploma, artículos y bibliografía de diferentes autores, para así extraer los elementos más significativos.
- **Histórico lógico:** Se utilizó con el objetivo de realizar un estudio sobre los antecedentes, evolución y tendencias actuales de los sistemas de reportes para nubes privadas, permitiendo determinar los aspectos más relevantes.

Métodos Empíricos.

- **Observación:** Permitió hacer un análisis y evaluación del comportamiento de los sistemas de reportes para nubes privadas, con el objetivo de identificar elementos que hagan de ellas una solución competente.

El presente trabajo está estructurado en introducción, tres capítulos de contenido, conclusiones, recomendaciones y referencias bibliográficas. El orden y contenido de los capítulos es el siguiente:

Capítulo 1. "Fundamentación teórica del sistema de reportes de una nube privada**"**, donde se definen los principales conceptos a tener en cuenta, resultando la base que sustenta el desarrollo de las tareas de investigación. Se realiza un análisis de los sistemas de reportes de diferentes gestores nubes, enfatizando en el módulo de infraestructura, con el objetivo de determinar elementos similares. Además, se determinan las herramientas y tecnologías que se emplearon en la construcción de la solución.

Capítulo 2. "Análisis y diseño del sistema de reportes de una nube privada", en el cual se describirán las características de la propuesta de solución. Se numera la lista de funcionalidades con que debe contar el sistema además de redactar las historias de usuario de cada una de ellas y las características del mismo. Se hace alusión a las fases de planificación y diseño propias de la metodología de desarrollo que se utilizará en la implementación del sistema, así como los elementos contenidos en cada una de ellas.

Capítulo 3. "Implementación y validación del sistema de reportes de una nube privada", en este capítulo se muestran las particularidades necesarias para describir el proceso de la implementación, para ello se definen los estándares de codificación que emplea el lenguaje de programación utilizado, además se hace alusión a la fase de prueba propuesta por la metodología de desarrollo y a los tipos de pruebas que implementa la metodología.

CAPÍTULO 1. **FUNDAMENTACIÓN TEÓRICA**

En el presente capitulo se definen los elementos teóricos relacionados con la computación en la nube, para una mejor comprensión de los aspectos más significativos a tratar en la investigación. También se aborda sobre las organizaciones líderes en la elaboración de estándares y recomendaciones en el paradigma de la computación en la nube, se realiza un estudio de herramientas homólogas con el fin de identificar funcionalidades que deben estar presentes en la propuesta de solución y se especifican las herramientas y tecnologías a utilizar en el desarrollo del trabajo.

1.1. Conceptos asociados al dominio del problema.

Virtualización: Es el proceso donde se abstraen recursos tecnológicos, con el fin de dividirlos en múltiples recursos lógicos para un aprovechamiento completo de sus funciones. Donde estos recursos lógicos divididos operan de forma aislada, minimizando la posibilidad de interferir entre ellos. Dígase recurso: a una plataforma de hardware, un dispositivo de almacenamiento, un SO[7] o algún recurso de la red. Pueden existir diferentes formas de virtualización, según lo que se pretenda virtualizar. Es posible virtualizar el hardware de un servidor, su software, virtualizar sesiones de usuario, aplicaciones y también se pueden crear MV en una computadora de escritorio (Yosmay Morales Suárez, 2016).

Hipervisor: Elemento de mayor relevancia que hace posible la virtualización, este software, también conocido como VMM[8], se encuentra entre el hardware y el huésped, separando el SO y las aplicaciones del hardware, realizando la función de intermediario. Es el responsable de llevar el control y asignar la cantidad de accesos que los sistemas operativos y aplicaciones tienen al procesador, memoria, disco duro, dispositivos de entrada-salida y conexiones de red (Tim Jones, 2009).

Máquina virtual: Una máquina virtual es un software que emula a un ordenador real y por lo tanto dispone de disco duro, memoria RAM[9], tarjeta gráfica, etc. y puede ejecutar programas como lo hace una computadora. En cierta medida, se puede ver como una partición del ordenador: la máquina real y la máquina virtual. Una característica esencial de las máquinas virtuales es que los procesos que ejecutan están limitados por los

[7] Siglas para el término: *Sistema Operativo.*
[8] Siglas en inglés para el término: *Virtual Machine Monitor.*
[9] Siglas en inglés para el término: *Random Access Memory.*

recursos y abstracciones proporcionados por ellas. Estos procesos no pueden escaparse de esta computadora virtual. El SO no puede establecer una diferencia entre una máquina virtual y una máquina física, ni tampoco lo pueden hacer las aplicaciones u otros ordenadores de una red (Alejandro Vega Velázquez, 2012).

Nube privada: Bajo el modelo de nube privada los recursos están disponibles para una sola organización. Se tiene una plataforma escalable en la que se puede decidir el nivel de exclusividad que se requiere, tanto en servidores como en conexiones. Hay que decidir el grado de aislamiento adecuado, pudiendo optarse por una red privada virtual a través de la cual se pueda acceder de forma segura a la infraestructura, garantizando que toda la información que se transfiera viaje de forma cifrada. De esta forma se establece una conexión con los servidores con la misma confiabilidad que en local, pero aprovechando las posibilidades de conexión de la nube. A su vez se puede recurrir al aislamiento de los recursos, de forma que toda la infraestructura de hardware (servidores, *firewall*, balanceadores de carga, sistemas de almacenamiento físico o lógico) esté aislada con dedicación exclusiva, lo que confiere un gran nivel de seguridad e integridad. Desde el punto de vista de la seguridad cabe destacar que todos los elementos de red (desde el *datacenter* remoto hasta las comunicaciones, la arquitectura de virtualización, etc.), están redundados para garantizar la máxima disponibilidad del servicio. A este respecto los proveedores deben hacer pública su política y al contratar los servicios se firma un SLA[10], en el que constan las indemnizaciones al cliente en caso de interrupción del servicio (Noemí Arbos et al, 2010).

Sistema de Reporte: tienen como objetivo principal mostrar una visión general de la situación de la empresa a través de una gran cantidad de datos manipulados de tal manera que satisfaga las necesidades del usuario y pueda realizar un análisis coherente, concreto y objetivo sobre los mismos. Consecuentemente, estos muestran la situación de las operaciones regulares de la empresa para que los directivos puedan controlar, organizar, planear y dirigir. Los reportes se pueden visualizar, exportar a otros formatos como PDF, HTML, XML, etc. y también se pueden imprimir en papel (Sistemas de Reportes - Inteligencia de Negocios, 2016).

[10] Siglas para el término: *Acuerdo de nivel de Servicio.*

1.2. Estandarización de los reportes.

Las actividades de estandarización, así como el respaldo de organizaciones internacionales son uno de los aspectos más importantes para la adopción de tecnologías innovadoras, pues permite reducir el riesgo de adopción, al garantizar interoperabilidad y persistencia en los mercados.

No hay ninguna organización de estándares universales para los sistemas de gestión de reportes hoy en día, por lo que los estándares de los sistemas de gestión de reportes se están formulando por diferentes grupos u organizaciones.

En la actualidad el NIST[11], la UIT[12] y el DMTF[13] son las organizaciones líderes en la elaboración de estándares y recomendaciones para el paradigma de computación en la nube. Sus trabajos se concentran fundamentalmente en lograr seguridad, portabilidad e interoperabilidad en entornos donde se apliquen estas tecnologías. A pesar de sus esfuerzos no existe un estándar que defina como debe ser un sistema de gestión de reportes para un CPD[14] a través del paradigma de computación en la nube.

Con respecto a su arquitectura de referencia la UIT plantea que la capa de "funciones comunes a las capas" constituye la fuente de información del proveedor, ya que debe proporcionar una vista consolidada de la distribución de los recursos y cuán eficiente están siendo explotados. En ella se lleva a cabo la gestión total del sistema: OAM&P[15], monitoreo, mecanismos de seguridad y la generación de alertas ante el mal funcionamiento de cualquier elemento de la arquitectura o servicio. No obstante, en los documentos brindados, no se especifican las tareas que debe cumplir la Gestión de Reporte (Focus Group and Cloud Computing Technical Report, 2012).

En la arquitectura de referencia del NIST, en la capa de Gestión de los Servicios se encuentra el bloque de Soporte al Negocio que posee los elementos encargados de ejecutar las operaciones que interactúan directamente con el usuario: gestión de usuario, de contrato, de inventario, contabilidad y facturación, reporte, auditoría y precios. La función de reporte y auditoría contempla las operaciones de monitoreo de los usuarios

[11] Siglas en inglés para el término: *National Institute of Standards and Technology.*
[12] Siglas para el término: *Unión Internacional de Telecomunicaciones.*
[13] Siglas en inglés para el término: *Distributed Management Task Force.*
[14] Siglas para el término: *Centro de Procesamiento de Datos.*
[15] Siglas en inglés para el término: *Operation, Administration, Maintenance and Provisioning.*

y la generación de reportes del sistema, pero en los documentos brindados no se especifican las formas o mecanismos para realizar estas tareas (Fang Lui et al. 2011).

Como parte del estándar de servicio de nube de la DMTF, el *Open Cloud Standards Incubator* plantea algunos requerimientos que los proveedores de servicios deben cumplir con respecto a los sistemas de reportes (Valerie Kane et al. 2010):

- Los proveedores de servicios deben brindar información de incidentes y alertas usando las interfaces definidas.
- La interfaz de gestión de eventos debe tener una gran variedad de fuentes de datos identificables (*logs* de aplicaciones, de los corta-fuegos, etc.) para que los consumidores de la nube puedan hacer inferencias acerca de sus sistemas.
- Los eventos y la información sobre estos deben permitir al cliente verificar los cargos hechos a su cuenta, el cumplimiento de las regulaciones y el monitoreo de características operacionales.
- Durante la operación de entidades de servicios en la nube, como por ejemplo VM, el proveedor debe monitorear las entidades del servicio para registrar eventos concernientes a la operación, facturación, etc. El proveedor de servicio en la nube debe facilitar esta información al consumidor en correspondencia con el SLA.
- La interfaz de eventos también debe ser lo suficientemente versátil como para hacer frente a la notificación, respuesta, contención y solución de incidentes.
- Los *logs* de operaciones deben estar disponibles de acuerdo con el SLA que define el tiempo de persistencia de los mismos.
- Se deben emplear mecanismos apropiados de conservación e integridad de *logs*, como la firma digital y dispositivos de almacenamiento de una sola escritura.

Por otro lado, el *Open Cloud Standards Incubator* destaca que, el consumidor debe ser capaz de solicitar y recibir reportes sobre el servicio contratado. Algunos tipos de reportes pueden ser: uso del servicio (por usuario o en general), consumo contra límite contratado y disponibilidad contra niveles contratados.

1.3. Roles definidos para la computación en la nube.

No existe una clasificación universal de los roles en la computación en la nube, por tanto, existen diferentes criterios que derivan en distintas clasificaciones. Partiendo de una visión muy general, se puede decir que como mínimo hay tres partes principales (Zamora Yustres, 2014):

- ✓ ***Cloud Consumer***: Se trata del usuario o empresa final que utiliza el servicio nube. El consumidor determina la precisión de los resultados y las mejoras necesarias para cumplir con los requisitos ya existentes o futuros.
- ✓ ***Cloud Provider****:* Se trata de la parte que proporciona el servicio de nube al consumidor, en base a los requisitos contratados. El *Cloud Provider* también puede llevar a cabo la subcontratación de servicios específicos para satisfacer las demandas que temporalmente excedan las capacidades del proveedor primario.
- ✓ ***Cloud Services Creator / Developer****:* El creador / desarrollador de servicios puede ser una entidad independiente, o puede residir en el dominio de alguna de los otros dos roles, consumidor o el proveedor.

1.4. Estudio de sistemas homólogos.

El presente epígrafe tiene como objetivo analizar si alguna de las herramientas existentes es capaz de satisfacer las exigencias de la presente investigación, además de contribuir a identificar funcionalidades que deben estar presente en un sistema de reportes para una nube privada.

1.4.1. Gestores nubes.

En este acápite se hace un análisis de los gestores nubes que son de interés para la investigación, por su capacidad a la hora de explotar los beneficios para efectuar la gestión de infraestructura en la nube, enfocado en una serie de aspectos que son de interés para el desarrollo de este trabajo:

- ✓ Los diferentes tipos de reportes que genera.
- ✓ Las operaciones que realizan o que no realizan dichos sistemas.
- ✓ En el formato que se pueden exportar los reportes generados.

CloudStack: El estudio de esta herramienta se debe principalmente a las referencias obtenidas sobre sus capacidades para efectuar la gestión de infraestructuras en la nube.

CloudStack despliega agentes de la nube por los Nodos de Computación (NC). Estos agentes son los encargados de registrar todos los eventos sobre los recursos en los nodos y las MV.

Los eventos en *CloudStack* son usados por los sistemas de monitoreo, de utilización de recursos y de facturación o por cualquier otro sistema que requiera el seguimiento de ellos. Los mismos registran cualquier acción realizada por un administrador o usuario y la almacenan en la base de datos del Servidor de Gestión. Los eventos se pueden clasificar de dos maneras: eventos estándares, en los cuales se registran sucesos que pueden o no representar procesos fallidos en el sistema y los eventos de ejecución, los cuales manifiestan el comienzo o finalización de tareas programadas (Rakesh Kumar et al. 2014).

De estos eventos registrados se generan los distintos reportes que son de gran utilidad para los diferentes roles de computación en la nube, de ahí parte el análisis que es llevado a cabo para la posterior toma de decisiones en las diferentes ramas, tanto en los consumidores y proveedores, así como los encargados del desarrollo y mantenimiento de dicho gestor.

En *CloudStack* las operaciones que generan eventos son: la creación, eliminación y ejecución de operaciones de mantenimiento de VM o mantenimientos en dispositivos de interconexión virtuales, la creación y eliminación de plantillas, de políticas para realizar el balanceo de cargas sobre la red y de volúmenes de almacenamiento, y finalizar o iniciar una sesión de usuario (Rakesh Kumar et al. 2014).

La notificación de los eventos se logra a través de la implementación del concepto de *bus* de abstracción de eventos en el Servidor de Gestión. Esto permite a los componentes y extensiones suscribir los eventos utilizando el protocolo AMQP[16]. Todos los eventos (alertas, eventos de acción, eventos de uso) y todas las categorías de eventos que cambian en el estado de los recursos, son publicados a través de la interfaz Web de *CloudStack* (Rakesh Kumar et al. 2014).

Una vez almacenada la información en la Base de Datos del Servidor de Gestión, una aplicación previamente instalada, denominada Servidor de Utilización, recopila toda la información estadística sobre la utilización de los recursos por los usuarios para generar reportes en forma de facturas de cobro a los mismos (Rakesh Kumar et al. 2014).

[16] Siglas en inglés para el término: *Advanced Message Queuing Protocol.*

El Servidor de Utilización se ejecuta por defecto una vez por día, aunque puede ser configurado para que tome las muestras de información varias veces al día. El Servidor de Utilización analiza los eventos y crea Registros de Uso a los cuales se puede acceder a través de la API[17] de *CloudStack*.

Los tipos de reportes que brinda el Servidor de Utilización son (Rakesh Kumar et al. 2014):

- ✓ VM funcionando y VM asignadas.
- ✓ Utilización de la red.
- ✓ Direcciones IP[18].
- ✓ Volumen de disco.
- ✓ Plantillas, isos e instantánea.
- ✓ Balanceador de carga o puerto de reenvío.
- ✓ Oferta de Red.
- ✓ Usuario VPN[19].

Una peculiaridad de *CloudStack* es que no muestra reportes referentes a la infraestructura (uso de los recursos en los nodos como son: almacenamiento, uso del CPU, RAM, entre otros), algo que es de gran importancia para este estudio.

Cloud Foundry: Lanzada el 24 de mayo de 2011 por *VMWare* y consiste en una PaaS bajo los estándares del *Open Source*. Soporta múltiples *Frameworks*, proveedores *Nube* y servicios de aplicaciones. Su utilidad reside en que permite acortar los tiempos necesarios para diseñar una aplicación, construir el código, y finalmente trasladarla a la nube, usando una solución PaaS abierta (Beka Kezherashvili, 2016).

Las herramientas que utiliza la plataforma son *Spring Source* (adquirida por *VMWare* en 2009) para desarrolladores *Java*, *Rails* y *Sinatra* para desarrolladores *Ruby*, y *Node.js* y otros *frameworks* JVM[20] incluyendo *Grails*.

Cloud Foundry presenta un buen grado de portabilidad. La plataforma no está vinculada a ningún entorno especial, soporta nubes privadas o públicas, incluyendo las desplegadas en *VMWare vSphere*, las

[17] Siglas en inglés para el término: *Application Programming Interface.*
[18] Siglas en inglés para el término: *Internet Protocol.*
[19] Siglas en inglés para el término: *Virtual Private Network.*
[20] Siglas en inglés para el término: *Java Virtual Machine.*

desarrolladas por *vCloud* de *VMWare*, nubes públicas no *VMWare* y, además, *Amazon Web Services* de *RightScale* (Beka Kezherashvili, 2016).

Existen tres formas de trabajar con *Cloud Foundry* (Rubén Aguilera Días Heredero, 2016):

- ✓ ***CloudFoundry.com***: es el *host* PaaS que ofrece *VMWare* para desplegar nuestras aplicaciones en la nube. Ahora mismo se encuentra en fase beta por lo que *VMWare* lo ofrece de forma gratuita a los desarrolladores que se registren, pero ya avisan en su página web que cuando finalice la fase beta se empezará a cobrar por el servicio, aunque todavía no se sabe cuál va a ser el precio final.
- ✓ ***CloudFoundry.org***: se trata del proyecto *Open Source* donde los desarrolladores pueden contribuir y colaborar con el proyecto.
- ✓ ***Micro Cloud Foundry***: *VMWare* ofrece una máquina virtual con una versión completa de *Cloud Foundry* para que los desarrolladores puedan hacer pruebas en local, asegurando que si funciona en local va a funcionar exactamente igual en la nube. Esta opción es la que presumiblemente quedará libre de cargo al finalizar la fase beta de *Cloud Foundry*.

EUCALYPTUS: Ofrece dos maneras para obtener reportes de la infraestructura. Se pueden obtener informes directamente desde el Controlador de la Nube (CLC), o se pueden obtener reportes de los datos exportados de la CLC e importados a un almacén de datos. Al instalarlo, automáticamente se obtienen reportes del sistema generados desde el CLC. Sin embargo, el lado negativo de usar el CLC para reportes es la latencia. Muchos entornos eligen centrar la función CLC en los procesos de nubes, más que en los procesos de presentación de reportes. Para estas necesidades, exportan los datos de la CLC al almacén de datos y realizan la elaboración de reportes desde ahí.

EUCALYPTUS permite generar reportes para supervisar el uso de recursos en el sistema. Cada tipo de reporte está destinado a un rango de tiempo específico.

En EUCALYPTUS se pueden generar los siguientes tipos de reportes de uso y contabilidad (Hewlett Packard Enterprise, 2016):

- ✓ Instancia: El reporte proporciona información acerca de la cantidad, la duración y la utilización de todas las instancias en ejecución. Se utiliza este reporte para comprender cuántas instancias ejecuta cada usuario, y si sus tipos de instancias son lo suficientemente grandes.

- ✓ S3: El reporte S3 proporciona información sobre el número de contenedores y objetos almacenados. Los contenedores vacíos no se reportan. Se utiliza este reporte para comprender las necesidades de almacenamiento de cada usuario y las necesidades de almacenamiento de la nube.
- ✓ Volumen: El reporte de volumen proporciona información sobre la cantidad, la duración y el tamaño de todos los volúmenes en uso. Se usa este reporte para comprender cuántos volúmenes están en marcha, y cuál es el tamaño de almacenamiento de cada volumen.
- ✓ Instantánea: El reporte de instantánea proporciona información sobre la cantidad de instantáneas de la nube. Mediante este reporte se puede comprender el número de instantáneas que hay, sus volúmenes, y cuál es el tamaño de cada instantánea.
- ✓ IP elástica: El reporte IP elástica proporciona información acerca del ciclo de vida de las IP elásticas en la nube, incluyendo los usuarios que están utilizando IP, qué IP está actualmente en uso, con qué frecuencia y que tiempo se asigna una IP.

Para generar reportes se pueden configurar las fechas de inicio y fin del intervalo de tiempo deseado. También se puede definir en qué formato se generará o exportará el reporte. Los formatos admitidos son CSV o HTML. En la bibliografía consultada no se encontró evidencia de que *EUCALYPTUS* genere reportes del uso de los recursos (CPU, memoria, almacenamiento, redes) en los nodos (clúster, nodos, VM, almacenes de datos), así como tampoco muestra una gestión de eventos que permita obtener los eventos del sistema para su clasificación y análisis.

OpenNebula: El sistema de monitoreo de *OpenNebula* gestiona información que incluye los parámetros de rendimiento y configuración típica para los nodos y MV, como son CPU, memoria RAM o el consumo de recursos de red. Estas métricas son recogidas por programas especializados, llamados sondas, que son incluidos por el sistema (OpenNebula Documentation, 2016).

Utilizando cualquiera de los dos modos de administración principales; la CLI o el *Sunstone* se pueden visualizar datos referentes al uso de los recursos en los nodos. Sin embargo, la información mostrada solo contiene los últimos valores recogidos por el sistema de monitoreo por lo que no permite ver el uso de los recursos en un intervalo de tiempo mayor. Además, mediante el *Sunstone* se pueden obtener plantillas de MV y nodos que brindan la información referente a los niveles de servicio.

El conjunto de herramientas de contabilidad de *OpenNebula* permite visualizar y reportar datos de uso de los recursos virtuales incluyendo el consumo de red de las MV y permite su integración con las plataformas de facturación.

OpenNebula incluye un buen sistema de gestión de usuarios y grupos de usuarios. Los usuarios en una nube de *OpenNebula* se clasifican en cuatro tipos: administradores, consumidores habituales, usuarios públicos y usuarios del servicio. En cuanto a reportes se pueden generar reportes de recursos asignados por usuarios independientes y por grupos de usuarios.
En cuanto a reportes de uso de los recursos en los nodos (aparte de la información de que se puede visualizar utilizando el *Sunstone* o el CLI) *OpenNebula* no brinda reportes. Tampoco ofrece reportes de los eventos ocurridos en el sistema (OpenNebula Documentation, 2016).

1.4.2. Herramientas de monitoreo y supervisión.

Para el análisis que se realizará de las herramientas se tendrá en cuenta la definición de una serie de parámetros. Tener estos aspectos definidos permitirá desechar las herramientas que no los cubran y trabajar más profundamente con las que sí.

Los parámetros a tener en cuenta son:

- ✓ El nivel de cubrimiento de cada herramienta en relación a los grupos de reportes.
- ✓ Que la herramienta permita el acceso multiusuario basado en roles que posibiliten brindar reportes a los usuarios o grupos de los eventos y recursos relacionados con ellos.
- ✓ Nivel de complejidad de configuración e interfaz visual.
- ✓ La integración con otras herramientas y *plugins*.

Nagios: El sistema *Nagios* consiste en dos componentes *Nagios Core* and *Nagios plugins*. Utiliza dos métodos para monitorear, activo y pasivo, ambos por medio de agentes desplegados por toda la infraestructura. La información que el sistema es capaz de recolectar está dada por los *plugins* instalados en él. *Nagios* ha sido implementado y probado en todo el mundo para diversos intereses, tiene una increíble cantidad de *plugins* y extensiones disponibles y permite la detección automática de nuevas máquinas de forma controlada (Nagios - The Industry Standard In IT Infrastructure Monitoring, 2016).

Nagios tiene una interfaz web donde se pueden ver los datos de los dispositivos monitoreados, así como el estado y los reportes para cada dispositivo. Entre los cuales están, reportes de uso, disponibilidad y rendimiento para nodos, grupos de nodos, o aplicaciones de servicios monitoreados, reportes de alertas, reportes de notificaciones, y reportes de eventos *logs*. Además cuenta con cientos de *plugins* que permiten

incrementar sus prestaciones como herramientas de interfaces *web* y de reportes las que permiten exportar reportes en diversos formatos, y generar reportes con métricas personalizadas (Nagios Core - Nagios, 2016).

Nagios es una aplicación muy fuerte y flexible para monitorear la disponibilidad y el uso de recursos en nodos y dispositivos de red. Sin embargo, el número y la complejidad de opciones de configuración pueden hacer a *Nagios* un poco complicado de implementar. Es importante resaltar que no soporta acceso multiusuario y que no se ajusta bien en las infraestructuras de computación en la Nube, debido al hecho de que ha sido diseñado para monitorear la infraestructura física en lugar de las infraestructuras virtuales altamente cambiantes (Nagios Plugins | The home of the official Nagios Plugins, 2016).

PandoraFMS: La herramienta *Pandora FMS* es un *software* capaz de monitorear infraestructuras virtuales en nubes privadas o semipúblicas que utilicen instancias de *Amazon EC2*, *Xen*, *VMWare* e *Hyper-V*; infraestructuras locales (redes, nodos, aplicaciones, bases de datos, escritorios, etc.); aplicaciones *web* (locales o basadas en la nube) y servicios de red remotos. *Pandora FMS* en cualquiera de sus versiones ofrece un amplio catálogo de monitorización de tecnologías profesionales. Este sistema permite que usted pueda adaptar nuestros *plugins* a sus necesidades. El acceso a los plugins *"enterprise"* de *Pandora FMS* está incluido en cualquiera de las licencias *Enterprise*, *MSP* y *NMS* (PanadoraFMS Enterprise, 2016).

Es fácil diseñar y/o ampliar un *plugin* en *Pandora FMS*, porque ha sido construido con una filosofía abierta, que permite que se puedan incorporar nuevas funcionalidades y/o intercambio de datos con terceros: APIs, Extensiones de consola, *Plugins* locales, *Plugins* de inventario, *Plugins* remotos, Comandos externos de Satélite: todo un *framework* para que *Pandora FMS* se adapte a su organización y no al revés (Guía Rápida General, 2016).

Existen muchos acrónimos para definir la Monitorización unificada, como *Application Performance Management* (APM), *Infrastructure Monitoring*, *Business Process Management* (BPM), *Service Level Monitoring*, *Event Monitoring*… ¿Pero por qué? La respuesta es que un operador necesita ver una información en concreto y el CIO[21] necesita ver otra información diferente. Para ello hay que ser capaz de obtener información de diferentes fuentes y ser capaz de agregarla, para construir los informes y vistas que se necesitan a diferentes niveles de la organización. Olvídese de tener diferentes herramientas para cada

[21] Siglas en inglés para el término: *Chief Information Officer.*

tecnología o departamento. Con *Pandora FMS* puede tenerlas todas en una sola herramienta y ahorrar mucho tiempo y dinero (Monitorización, 2016).

PandoraFMS permite monitorizar tu negocio de diferentes áreas, esto es un rasgo importante que no todas las herramientas de monitoreo y control posen ejemplo de estas áreas o tipos de monitorización son (Monitorización, 2016):

- ✓ **Monitorización de Infraestructuras:** *Routers*, *switches* y servidores. Desplegados en mapas de topología automatizada, o vistas definidas de usuario. Desde una vista única del monitor, vista de topología de la tecnología, vistas a nivel de nodo o de grupo. Todo ello le será notificado mediante notificaciones *push* o por SMS[22]. Personalice su vista de eventos para ver qué es lo que va mal y solucionarlo a través de la pasarela de acceso remoto de *Pandora FMS*.
- ✓ **Monitorización del Nivel de Servicio:** Cualquier caso real tiene cientos de métricas. Su jefe no quiere los detalles, sólo necesita que todo esté bajo control y que el servicio esté funcionando adecuadamente. Los mapas de servicio y vistas personalizadas sirven exactamente para ese propósito. Utilice datos de tiempo real para obtener cuadros de mando creados por usted mismo, añada información adicional mediante la utilización de los módulos de síntesis y desviaciones estimadas con nuestros algoritmos de predicción y tendrá delante únicamente lo que necesita ver, evitando información de más.
- ✓ **Monitorización de Aplicaciones:** Bases de datos, servidores de correo y aplicaciones de negocio están dando un servicio y usted necesita conocer su nivel de rendimiento, lo que implica un acceso de bajo nivel a las diferentes métricas, no sólo una comprobación de los procesos y puertos. Implica conocer exactamente lo que hacen peticiones, y los usuarios. Con *Pandora FMS* contará con la ayuda de una enorme cantidad de gráficos, informes y alertas.

PandoraFMS tiene una interfaz totalmente visual al usuario desde la cual se puede acceder a todo el sistema, brindando capacidades de visualización que permiten trabajar con los datos de forma más rápida y más inteligente. Es de destacar que la instalación y configuración de este sistema es sencilla.

[22] Siglas en inglés para el término: *Short Message Service.*

Hay un detalle que es de gran importancia mencionar y es que cuantos más sistemas quiera monitorizar la organización, más recursos (CPU, Memoria, velocidad del disco) tendrá que asignar al servidor de *Pandora FMS*.

Zabbix: Es un software de nivel empresarial diseñado para lograr la máxima disponibilidad y rendimiento de los componentes de la infraestructura de TI monitoreados. Las extensas capacidades de personalización de *Zabbix* permiten integrarlo en cualquier entorno y reunir datos de los sistemas.

Zabbix tiene una interfaz web desde la cual se puede acceder a todo el sistema, brindando capacidades de visualización que permiten trabajar con los datos de forma más rápida y más inteligente. Es de destacar que la instalación y configuración de este sistema es sencilla. Su *backend* está escrito en C y el *frontend* web está escrito en PHP[23].

Zabbix puede descubrir automáticamente los servidores, dispositivos de red y aplicaciones, permitiendo la recolección de estadísticas y datos de rendimiento precisos. El monitoreo de los indicadores de rendimiento como CPU, memoria, red, almacenamiento y los procesos de disco se pueden hacer fácilmente con el agente *Zabbix*, que está disponible para *Linux*, *UNIX* y *Windows* permitiendo generar reportes de uso y disponibilidad de recursos. Es un proceso nativo y no requiere de un entorno específico, como *Java* o *.NET.* Además recolecta datos de la mayoría de los dispositivos de la Infraestructura con los agentes SNMP[24], y se apoya en agentes IPMI[25] para recoger datos de hardware como temperatura, ventilación y estado físico de los discos, evitando tiempo de inactividad y pérdidas financieras (Zabbix The Enterprise-Class Open Source Network Monitoring Solution, 2016).

También se pueden crear usuarios con distintos privilegios, además se pueden formar grupos de usuarios. Estos grupos pueden estar basados en categorías como empresa a la que pertenecen, dispositivos a los que tienen acceso, ubicación física o cualquier otra definida por el administrador. Utilizando este tipo de acceso al sistema se puede brindar acceso a parte de la información recogida a algunos usuarios mientras que los administradores acceden a toda la información del sistema.

---

[23] Siglas en inglés para el término: *Hypertext Pre-Processor.*
[24] Siglas en inglés para el término: *Simple Network Management Protocol.*
[25] Siglas en inglés para el término: *Intelligent Platform Management Interface.*

El sistema incluye también la capacidad de monitorear los *logs* de texto y los logs de eventos de *Windows* como una función nativa de los agentes *Zabbix*. Los registros son analizados constantemente por el agente y cuando se encuentra un elemento de búsqueda definido, el servidor *Zabbix* es notificado y puede incluso tomar alguna acción o enviar una notificación a un usuario o grupo de forma automática (Richards Olups, 2010).

Zenoss: La herramienta *Zenoss* realiza una descripción detallada de cualquier dispositivo que gestiona y la relaciona con otros dispositivos, unidades de negocio u otras agrupaciones importantes que se definan. El sistema *Zenoss* implementa varias formas de monitoreo como scripts personalizados, utilizando protocolos SNMP, SSH[26], XML-RPC[27], telnet o utilizando su habilidad de auto descubrimiento. Permitiendo recoger toda la información de la infraestructura. Esta información está disponible a través de un navegador *web* estándar, desde el cual se accede a los reportes de todos los componentes del sistema (Zenoss Updates Open Source Management Offering - Computer Business Review, 2016).

El sistema *Zenoss* cuenta con acceso de usuarios por roles. Se puede asociar cualquier objeto del sistema con un usuario en particular o con un grupo de usuarios, permitiéndole al usuario o grupo monitorear y recibir reportes y alertas referentes a los objetos definidos. Las vistas de eventos personalizados también son definidas para cada usuario.

El Sistema realiza una gestión de rendimiento que brinda información importante sobre el uso de los recursos de TI (CPU, memoria, disco, red, entre otros) y hace un seguimiento de estos cambios en el tiempo permitiendo brindar reportes de disponibilidad, capacidad, rendimiento y la utilización de los recursos para cada dispositivo, aplicación o servicio del sistema así como gráficos de tendencias en el tiempo, además cuando los recursos experimentan situaciones definidas por los administradores entonces se generan alarmas o notificaciones a los mismos.

El sistema cuenta con cientos de paquetes disponibles en su página principal, entre los cuales hay algunos dedicados al trabajo con bases de datos, sistemas operativos, aplicaciones, servicios, gestores nubes como *CloudStack* y *OpenStack* y a tecnologías de virtualización como KVM[28], *Xen* y *VMWare*. Estos paquetes

[26] Siglas en inglés para el término: *Secure Shell.*
[27] Siglas en inglés para el término: *Remote Procedure Call.*
[28] Siglas en inglés para el término: *Kernel-based Virtual Machine.*

utilizan todas las funcionalidades de *Zenoss* e incorporan formas de recolección de datos, visualización y organización de información (Monitoring for Modern IT, 2016).

A medida que se reciben los eventos, Zenoss los ejecuta a través de un conjunto de normas que aumentan la información contenida en los mismos y se integra toda esta información en el modelo Zenoss para ser consultada posteriormente. Mediante configuración y clasificación de eventos Zenoss agrupa los eventos de diversas maneras permitiendo mostrar diferentes tipos de reportes como eventos comunes, alertas eventos personalizados y análisis de eventos, también se pueden mostrar los detalles de cada evento (Michael Badger, 2008).

ZenPacks provee una arquitectura orientada a *plugins* que permite a los miembros de la comunidad extender las funcionalidades de *Zenoss*. Se dice que en Zenoss cualquiera puede desarrollar un *Zenpack* y son libres de aplicarle la licencia que quieran. Como un incentivo para la compra de la versión empresarial, *Zenoss*, Inc. podría desarrollar *ZenPacks* exclusivos para esta versión. Zenoss, Inc. decidió liberar los *ZenPacks* incluidos en *Zenoss Enterprise* como código compartido, esto significa que los usuarios pueden modificar el código, pero no distribuir los cambios (Monitoring for Modern IT, 2016).

Ossim: Es una distribución de productos *open source* integrados para construir una infraestructura de monitorización de seguridad. Su objetivo es ofrecer un marco para centralizar, organizar y mejorar las capacidades de detección y visibilidad en la monitorización de eventos de seguridad de la organización, además de tratar la información generada, almacenarla y priorizarla utilizando técnicas de correlación.
Este sistema consta de las siguientes herramientas de monitorización (Julio Casal, 2003):

- ✓ Cuadro de Mandos para visibilidad a alto nivel.
- ✓ Monitores de Riesgo y Comportamiento para la monitorización a nivel medio.
- ✓ Consola Forense y Monitores de Red para el bajo nivel.

Estas herramientas se alimentarán de las nuevas capacidades desarrolladas en el posproceso de los SIM[29] y cuyo objeto es aumentar la fiabilidad y sensibilidad de la detección (Julio Casal, 2003):

- ✓ Correlación
- ✓ Priorización

[29] Siglas en inglés para el término: *Security Information Management*.

- ✓ Valoración de Riesgos

El posproceso a su vez es alimentado por los preprocesadores, estos son un número de detectores y monitores ya conocidos por la mayoría de administradores que se integraran en nuestra distribución (Julio Casal, 2003):

- ✓ IDS[30] (detectores de patrones)
- ✓ Detectores de anomalías
- ✓ Firewalls
- ✓ Monitores varios

OSSIM no quiere ser un producto, sino una solución, un sistema personalizado para las necesidades de cada organización formado por la conexión e integración de varios módulos especialistas.

Es una arquitectura de monitorización abierta pues integra diversos productos del mundo libre (*plugins*), intentando seguir siempre los estándares y las tendencias del mundo open source (de las cuales se cree que en soluciones de monitorización serán los estándares en todos los entornos). Facilita la integración de herramientas *open source* y herramientas propietarias lo permite tener una visión completa de la seguridad de la red desde una perspectiva de confidencialidad, integridad y disponibilidad (Angel Alonso Párrigas, 2016).

Es una solución integral pues es capaz de ofrecer las herramientas y funcionalidad para monitorización de todos los niveles desde el más bajo (firmas detalladas de un IDS, dirigido al técnico de seguridad), hasta el más alto (El Cuadro de Mandos dirigido a la Dirección Estratégica), pasando por: Consolas Forenses, niveles de Correlación, Inventariado de Activos y Amenazas, y Monitores de Riesgos (Angel Alonso Párrigas, 2016).

1.4.3. Comparación de las herramientas de monitoreo y supervisión analizadas.

La Tabla 1: Comparación entre las herramientas estudiadas(Fuente: elaboración propia.). muestra un resumen de la comparación realizada entre las herramientas estudiadas de mayor desarrollo teniendo en cuenta los parámetros definidos en el sub epígrafe 1.4.2.

[30] Siglas en inglés para el término: *Intrusion Detection System*.

Tabla 1: Comparación entre las herramientas estudiadas(Fuente: elaboración propia.).

Parámetros		Nagios	PandoraFMS	Zabbix	Zenoss	Ossim
Nivel de cubrimiento	Infraestructura	Uso, disponibilidad y rendimiento	Routers, switches y servidores	Uso y disponibilidad	Uso, disponibilidad rendimiento, capacidad y tendencias	Uso, disponibilidad y rendimiento
	Eventos del sistema	Análisis de eventos y alertas de notificaciones	Mapas de servicio y vistas personalizadas	Análisis de eventos, notificaciones y detalles de eventos	Análisis de eventos, alertas, notificaciones, detalles de eventos y eventos comunes	Vistas personalizadas, análisis de eventos, notificaciones, Alertas,
	Aplicaciones	Rendimiento, uso y disponibilidad	Rendimiento, uso y disponibilidad	Rendimiento	Rendimiento, uso y disponibilidad	Rendimiento, uso y disponibilidad
Acceso multiusuario basado en roles		No tiene	Si tiene, gracias a la monitorización unificada	Si tiene	Si tiene	En la bibliografía consultada no se encontró nada referente a este parámetro

Complejidad de configuración e interfaz visual	Complejo, interfaz amigable	Muy sencilla, interfaz amigable	Fácil, interfaz amigable	Fácil, interfaz amigable	Fácil, interfaz amigable
Integración con otras herramientas y plugins	Si	Si	Si permite integración aunque en la bibliografía consultada no se encontró si se le pueden instalar nuevos plugins	Si	Si

La comparación de las herramientas anteriores contribuyo a identificar funcionalidades que deben estar presentes en la propuesta de solución.

1.5. Herramientas y tecnologías.

En este epígrafe aparece una descripción de las herramientas y tecnologías que se emplearan en el desarrollo del sistema de reportes para una nube privada.

1.5.1. Metodología de desarrollo de software.

Se escoge **XP**[31] porque es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes,

[31] Siglas en inglés para el término: *Extreme Programming.*

simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent Beck, el padre de XP, describe la filosofía de XP sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente, otras publicaciones de experiencias se han encargado de dicha tarea.

Aunque ya existen libros asociados a cada una de las metodologías ágiles existentes y también abundante información en Internet, es XP la metodología que resalta por contar con la mayor cantidad de información disponible y es con diferencia la más popular. Por todo lo antes planteado se escoge la metodología XP (Patricio Letelier, 2016).

1.5.2. Lenguajes de programación.

Python es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de *Python* y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para el desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas (Guido van Rossum, 2003).

Para el desarrollo del módulo de gestión de reportes se propone utilizar *Python* como lenguaje de programación, ya que posee muchas ventajas de las cuales se mencionan las más importantes:

- ✓ Propósito general: Se pueden crear distintos tipos de programas.
- ✓ Multiplataforma: Hay versiones disponibles de *Python* en muchos sistemas informáticos distintos. Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible siempre y cuando exista un intérprete programado para este lenguaje.
- ✓ Interpretado: No se debe compilar el código antes de su ejecución. En realidad, sí que se realiza una compilación, pero esta se realiza de manera transparente para el programador.
- ✓ Interactivo: Dispone de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que puede ayudar a entender mejor el lenguaje y probar los resultados de la ejecución de porciones de código rápidamente.

- ✓ Orientado a objetos: La programación orientada a objetos está soportada en *Python* y ofrece en muchos casos una manera sencilla de crear programas con componentes reutilizables. Además, *Python* también permite la programación imperativa, programación funcional y programación orientada a aspectos.
- ✓ Funciones y librerías: Dispone de muchas funciones incorporadas en el propio lenguaje, para el tratamiento de strings11, números, archivos, etc. Además, existen muchas librerías que se pueden importar en los programas para tratar temas específicos.
- ✓ Sintaxis clara: Tiene una sintaxis muy visual, gracias a una notación indentada (con márgenes) de obligado cumplimiento. Esto ayuda a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar.
- ✓ Mixto: Se puede integrar de manera fácil con otros lenguajes de programación.

1.5.3. Herramienta CASE.

Las herramientas CASE son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida para el desarrollo de un software.

Visual Paradigm**,** es una herramienta de diseño que facilita el desarrollo de software. Ofrece un paquete completo útil para la captura de requisitos, la planificación del software, la planificación de pruebas, el modelado de clases y el modelado de datos (What is Visual Paradigm?, 2016).

Las principales características de la herramienta son:

- ✓ Soporta las últimas versiones del UML.
- ✓ Posee un poderoso generador de documentación y reportes en formato PDF, HTTP y JPG.
- ✓ Proporciona soporte para varios lenguajes en la generación de código e ingeniería inversa como: Java, C++, CORBA IDL, PHP, Ada y *Python*.
- ✓ Disponibilidad en múltiples plataformas *(Windows, Linux)* Capacidades de ingeniería directa e inversa.

Se selecciona *Visual Paradigm for UML* en su versión 8.1 como herramienta para el modelado UML, pues permite trabajar de forma colaborativa, hacer un trabajo organizado y ágil. Posibilita la realización de los

diagramas necesarios para el desarrollo y mejor entendimiento de la aplicación. Permite realizar ingeniería inversa a partir del código fuente. Al ser seleccionado el lenguaje de modelado UML, es conveniente tener en cuenta su vinculación con *Visual Paradigm*, resaltando que este último presenta abundante documentación y demostraciones interactivas.

1.5.4. Framework de desarrollo.

El concepto *framework* se emplea en muchos ámbitos del desarrollo de sistemas software, no solo en el ámbito de aplicaciones Web. Se pueden encontrar *frameworks* para el desarrollo de aplicaciones médicas, de visión por computador, para el desarrollo de juegos, y para cualquier ámbito.

En general, con el término *framework*, se refieren a una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un *framework* se puede considerar como una aplicación genérica incompleta y configurable a la que se puede añadir las últimas piezas para construir una aplicación concreta.

Los objetivos principales que persigue un *framework* son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Un *framework* Web, por tanto, puede ser definido como un conjunto de componentes (por ejemplo, clases en java y descriptores y archivos de configuración en XML) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas web.

Para el desarrollo de la aplicación se propone utilizar ***Django*** en su versión 1.10.2 que es un *framework* de desarrollo web totalmente implementado sobre *Python*, con el que se pueden crear y mantener aplicaciones de alta calidad. Incluye un servidor web ligero que se puede usar mientras se desarrolla. Al mismo tiempo, *Django* permite trabajar fuera de su ámbito según sea necesario. *Django* ofrece las siguientes facilidades (Django Tutorial, 2016):

- ✓ Sistema de plantillas para separar la presentación de un documento de sus datos.
- ✓ Construcción automática de interfaces de administración.
- ✓ Vistas genéricas que recogen ciertos estilos y patrones comunes en su desarrollo y los abstraen, de modo que se puede escribir rápidamente vistas comunes de datos sin tener que escribir mucho código.

- ✓ Sistema de caché robusto y con un nivel de granularidad ajustable, que permite guardar páginas dinámicas para que no tengan que ser recalculadas cada vez que se piden.
- ✓ Integración con bases de datos y aplicaciones existentes.
- ✓ Construcción de aplicaciones multilenguaje permitiendo especificar cadenas de traducción de más de cuarenta idiomas.

1.5.5. IDE de desarrollo.

Pycharm es un editor de código inteligente que proporciona soporte de primera clase para los lenguajes de programación: *Python*, *JavaScript*, *CoffeeScript*, *TypeScript*, HTML/CSS, *Cython*, lenguajes de plantilla, *AngularJS* y *Node*.js, y otros menos utilizados. *Pycharm* funciona en las plataformas Windows, Mac OS y Linux con una única clave de licencia, también ofrece un espacio de trabajo con colores personalizables y atajos de teclado. La decisión de seleccionar como IDE[32], *Pycharm* en su versión 3.4, está dada a que ofrece autocompletación inteligente de código, comprobación de errores sobre la marcha, soluciones rápidas y fácil navegación en el proyecto. *Pycharm* mantiene la calidad del código bajo control con chequeos, asistencia a pruebas, refactorizaciones inteligentes, y una serie de inspecciones, lo que ayuda a escribir un código limpio y fácil de mantener (Mundo Programa, 2016).

1.5.6. Gestor de Base de Datos.

PostgreSQL es un SGBD[33] que incorpora el modelo relacional para sus BD[34] y es compatible con el lenguaje de consulta SQL[35]. Es ampliamente considerado como una de las alternativas de sistemas de BD de código abierto.

PostgreSQL destaca por su amplísima lista de prestaciones que lo hacen capaz de competir con cualquier SGBD comercial (The PostgreSQL Global Development Group, 2016):

- ✓ Está desarrollado en C, con herramientas como Yacc y Lex.

---

[32] Siglas en inglés para el término: *Integrated Development Environment*.
[33] Siglas del término: *Sistema Gestor de Base de Datos.*
[34] Siglas del término: *Bases de Datos.*
[35] Siglas en inglés para el término: *Structured Query Language*.

- ✓ La API de acceso al SGBD se encuentra disponible en C, C++, *Java, Perl,* PHP*, Python* y TCL, entre otros.
- ✓ Cuenta con un rico conjunto de tipos de datos, permitiendo además su extensión mediante operadores definidos y programados por el usuario.
- ✓ Su administración se basa en usuarios y privilegios.
- ✓ Sus opciones de conectividad abarcan TCP/IP, sockets Unix y sockets NT, además de soportar completamente ODBC.
- ✓ Los mensajes de error pueden estar en español y hacer ordenaciones correctas con palabras acentuadas o con la letra 'ñ'.
- ✓ Es altamente confiable en cuanto a estabilidad se refiere.
- ✓ Puede extenderse con librerías externas para soportar encriptación, búsquedas por similitud fonética (*soundex*), etc.
- ✓ Control de concurrencia multi-versión, lo que mejora sensiblemente las operaciones de bloqueo y transacciones en sistemas multi-usuario.
- ✓ Soporte para vistas, claves foráneas, integridad referencial, disparadores, procedimientos almacenados, subconsultas y casi todos los tipos y operadores soportados en SQL92 y SQL99.
- ✓ Implementación de algunas extensiones de orientación a objetos. En *PostgreSQL* es posible definir un nuevo tipo de tabla a partir de otra previamente definida.

1.6. Conclusiones parciales.

Una vez concluido el presente capítulo se arriba a las siguientes conclusiones:

- ✓ Entre las organizaciones de estandarización NIST, UIT y DMTF esta última destaca por ser la que más información aporta en cuanto a cómo debe estar conformado un sistema de reporte.
- ✓ Los gestores nube, muestran información sobre el consumo de los recursos virtuales en tiempo real, pero no dan la posibilidad de conocer sobre el comportamiento de los recursos con cierta antigüedad.
- ✓ Las herramientas de monitoreo y supervisión brindan bien detallado y en diferentes formatos información sobre el uso de los recursos físicos y virtuales, pero no permiten trabajar adecuadamente

con la información que almacenan, pero sí contribuyeron a identificar funcionalidades que si deben estar presente en la solución.

- ✓ Fueron establecidas la metodología, herramientas y tecnologías que se utilizarían en la construcción de la solución.

CAPÍTULO 2. **ANÁLISIS Y DISEÑO**

En este capítulo se describirán las características de la propuesta de solución. Se describe la lista de funcionalidades con que debe contar el sistema y las características del mismo. Se hace alusión a las fases de planificación y diseño propias de la metodología de desarrollo (XP) que se utilizará en la implementación del sistema. Se realiza el plan de duración de las iteraciones a llevar a cabo y el plan de entregas donde se detalla en qué momento se hará entrega de las diferentes versiones del sistema, además se describe la arquitectura del software, el patrón arquitectónico MTV[36] propio de Django, los patrones de diseño utilizados y el modelo de BD para un mejor entendimiento de la propuesta de solución.

2.1. Flujo actual del proceso.

A través de la interacción con varios especialistas de la XETID y las especificaciones del cliente, fue posible obtener una visión de cómo funciona el monitoreo de los recursos y el envío de las alertas en el CD.

En este caso los empleados supervisan los *hosts* a través de herramientas personalizadas, pero que no se integran entre ellas lo que hace el proceso un poco engorroso. Los recursos generan alertas que son notificadas a los administradores por parte de los empleados en servicio en el momento que fueron generadas, por último, son los administradores quienes resuelven las alertas.

A continuación, en la Ilustración 1: Modelo de Dominio(Fuente: elaboración propia.). se muestra el modelo de dominio para ayudar a comprender el flujo actual del proceso.

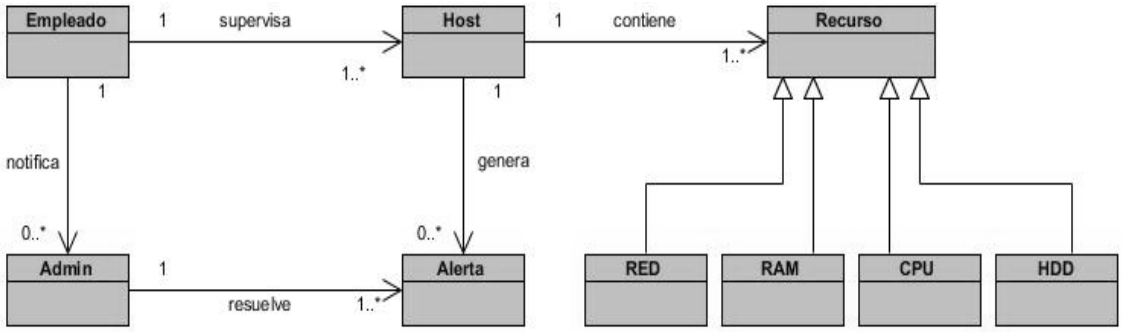


Ilustración 1: Modelo de Dominio(Fuente: elaboración propia.).

[36] Siglas en inglés para el término: *Models-Templates-Views.*

2.2. Propuesta de solución.

La solución propuesta está diseñada para permitir monitorear el estado del consumo de los recursos en tiempo real para cada uno de los elementos de la infraestructura (RED, RAM, CPU y Almacenamiento) tanto para los recursos físicos como para los que se encuentran virtualizados. Además, debe permitir visualizar los mismos por períodos o rangos de fechas especificados, utilizando gráficas de barras y de líneas que logren marcar un comportamiento promedio, el día de mayor consumo y saber cuándo se ha ido por encima y debajo de la media del consumo habitual. Para ello es necesario almacenar en una base de datos relacional todo lo antes descrito. Por otro lado, cuando un determinado recurso tenga un comportamiento aceptable casi llegando al límite para alcanzar la siguiente categoría de acuerdo a los umbrales definidos debe generan una alerta que notifique al personal de TI el posible colapso por falta de recurso. De las alertas se quiere almacenar el momento en que ocurre, la aplicación o servicio que la generó y cantidad de usuarios consumiendo el servicio. Por último, el sistema tiene que permitir exportar reportes a .pdf y rangos de fechas especificados como diario, semanal, mensual y en cualquier otro definido por el administrador donde muestre las especificaciones anteriores y la cantidad de alertas que se generaron.

2.3. Lista de funcionalidades.

Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer. Estos requerimientos dependen del tipo de software que se desarrolle, de los posibles usuarios del software y del enfoque general tomado por la organización al redactar requerimientos (Angel Gabriel Olivera Sosa, 2010).

A continuación, en la Tabla 2 se muestra la lista de funcionalidades con que debe contar el sistema.

Tabla 2: Lista de funcionalidades(Fuente: elaboración propia.).

Funcionalidades	
1. Crear umbral. 2. Eliminar umbral. 3. Modificar umbral. 4. Listar umbral. 5. Crear rol. 6. Eliminar rol. 7. Modificar rol. 8. Listar rol. 9. Eliminar usuario.	10. Modificar usuario. 11. Listar usuario. 12. Insertar *host.* 13. Eliminar *host.* 14. Crear reporte. 15. Exportar reporte. 16. Configurar alerta. 17. Visualizar recursos.

2.4. Características del sistema.

Son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requerimientos no funcionales a menudo se aplican al sistema en su totalidad. Normalmente apenas se aplican a características o servicios individuales del sistema. Los requerimientos no funcionales, como su nombre sugiere, son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento (Requerimientos Funcionales y No Funcionales (RF/RNF), 2017).

2.4.1. Usabilidad.

Se requiere un nivel medio o alto de conocimientos de informática. Además, el usuario necesitará una preparación previa para operar con el sistema. La aplicación posee un diseño e interfaces intuitivas, que permiten la comprensión del usuario.

2.4.2. Eficiencia.

El sistema debe responder en el mínimo de tiempo posible, ante las notificaciones por parte de las aplicaciones clientes. La eficiencia de la aplicación estará determinada en gran medida por la velocidad de las consultas a

la base de datos, la rapidez en el envío de las notificaciones, las características de la computadora donde se encuentre instalado el sistema y la velocidad de respuesta de los servidores a los que se conecta la aplicación.

2.4.3. Software y hardware.

- ✓ Características de hardware en el servidor:
    - Procesador Core i3 4 GHz.
    - Memoria RAM 4 GB. La cantidad de memoria RAM varía según la cantidad de usuarios Web, así como de la cantidad de tareas extras que ejecute el servidor.
    - Disco Duro con 80 Gb libres para datos.
- ✓ Características de software en el servidor:
    - Sistema operativo:
        - *Ubuntu Server* 9.04, *Linux* (kernel 2.6).
    - Deben ser instaladas las siguientes dependencias:
        - *PostgreSQL* 9.3.
        - Apache 2.
        - Servidor de correo (Zimbra 8.0.4)
- ✓ Características de hardware en el cliente:
    - Procesador Core Duo 1+ GHz.
    - Memoria RAM 1 Gb.
    - Disco Duro 10 Gb de espacio libres.
    - La resolución recomendada para la utilización del sistema y los visores de PDF es 1366x768 pixeles.
- ✓ Características de software en el cliente:
    - Sistema operativo:
        - Microsoft Windows 7 o superior, Ubuntu 14.04 o superior.
    - Deben ser instaladas las siguientes dependencias:
        - Navegador WEB (Mozilla Firefox, Chrome, Internet Explorer).
        - Lector de PDF (Adobe Reader).

2.4.4. Interfaz.

La solución propuesta poseerá una interfaz que cumpla con las pautas de diseño elaboradas por la Universidad de las Ciencias Informáticas, permitiendo que la misma cuente con una interfaz amigable e intuitiva para los usuarios que interactúen con ella.

2.5. Fase de Planificación.

Es la primera fase de la metodología XP, en esta se define el alcance real del sistema permitiendo que el equipo de desarrollo se familiarice con las herramientas, tecnologías y procesos. Durante esta fase el cliente establece la prioridad de cada historia de usuario, y en consecuencia los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Esta estimación se expresa utilizando como medida el punto. Un punto es considerado como una semana ideal de trabajo que constará de 5 días laborables de 8 horas cada uno para un total de 40 horas a la semana, donde los miembros del equipo de desarrollo trabajan el tiempo planeado sin interrupción. Se toman acuerdos sobre el contenido de la entrega y se determina un cronograma en conjunto con el cliente (Maite Rodríguez Corbea y Mayelin Ordoñez Pérez, 2007).

2.5.1. Historias de Usuario.

Las historias de usuarios (HU) son la técnica utilizada en XP para representar los requisitos del software con pequeños textos en los que el cliente detalla una actividad que realizará el sistema de forma sencilla y clara, mostrando solamente el perfil de la tarea a realizarse. Para hacer más comprensible las mismas, a continuación, se describe su leyenda (Maite Rodríguez Corbea y Mayelin Ordoñez Pérez, 2007):

- **Número:** Número de la historia de usuario incremental en el tiempo.
- **Nombre de la historia de usuario:** Nombre de la historia de usuario especificado por el programador.
- **Programadores responsables:** Personas involucradas en el desarrollo de las HU.
- **Prioridad en negocio (Baja, Media, Alta):**
    - Baja: Se le otorga a las HU que son de funcionalidades auxiliares y que son independientes del sistema.
    - Media: Se le otorga a las HU que son de funcionalidades a tener en cuenta, sin que estas tengan una afectación sobre el sistema que se esté desarrollando.
    - Alta: Se le otorga a las HU que son de funcionalidades fundamentales en el desarrollo del sistema.
- **Riesgo en desarrollo (Bajo, Medio, Alto):**

- Bajo: Cuando en la implementación de las HU puedan existir errores, pero éstos son tratados fácilmente y no afectan el desarrollo del sistema.
- Medio: Cuando en la implementación de las HU puedan existir errores y retrasen la entrega del producto.
- Alto: Cuando en la implementación de las HU pueda existir algún error y afecte la disponibilidad del sistema.

- **Puntos estimados:** Tiempo estimado que se demorará el desarrollo de la HU.
- **Iteración asignada:** Número de la iteración.
- **Descripción:** Breve descripción de la HU.
- **Observaciones:** Señalamiento o advertencia del sistema.

A continuación, se muestran 3 de las historias de usuario con las que cuenta el sistema las demás se encuentran en el Anexo # 1:

Tabla 3: HU # 12 Insertar host(Fuente: elaboración propia.).

Número:	12	**Nombre de la historia de usuario:**	Insertar *host*
Programadores Responsables:		Adrián Pérez Prieto	
Prioridad en negocio:	Alta	**Puntos estimados:**	2
Riesgo en desarrollo:	Alta	**Iteración asignada:**	1
Descripción:	El sistema debe mostrar un formulario en el cual el administrador debe propiciar la dirección IP y una Llave de autenticación para acceder al *host* y así extraer la información necesaria y verificar si es virtual o físico.		
Observaciones:	✓ El sistema después de que el administrador introduzca la información debe mostrar un mensaje que indique si la acción fue satisfactoria o no, en caso que		

	no se pueda acceder al *host* debe retornar a la vista inicial para probar otra vez con nuevas direcciones o llaves. ✓ En caso contario debe mostrar una vista con la información de todos los elementos de infraestructura contenidos en él.

Tabla 4: HU # 1 Crear Umbral(Fuente: elaboración propia.).

Número:	1	**Nombre de la historia de usuario:**	Crear Umbral
Programadores Responsables:		Adrián Pérez Prieto	
Prioridad en negocio:	Alta	**Puntos estimados:**	1
Riesgo en desarrollo:	Alto	**Iteración asignada:**	1
Descripción:	El sistema debe permitirle administrador en el momento que el desee(haciendo las pruebas pertinentes) definir los umbrales para cada elemento de la infraestructura.		
Observaciones:	✓ Si el administrador introduce la información de forma incorrecta, el sistema emite un mensaje notificando el error. ✓ Si el administrador introduce la información dejando campos obligatorios vacíos, el sistema emite un mensaje indicándole que los campos obligatorios deben de llenarse. ✓ En caso contario el sistema debe mostrar un mensaje indicando la confirmación que se crearon los umbrales de forma correcta.		

Tabla 5: HU # 2 Eliminar Umbral(Fuente: elaboración propia.).

Número:	2	**Nombre de la historia de usuario:**	Eliminar Umbral
Programadores Responsables:		Adrián Pérez Prieto	

Prioridad en negocio:	Media	**Puntos estimados:**	2/5
Riesgo en desarrollo:	Medio	**Iteración asignada:**	2
Descripción:	El sistema debe permitirle al administrador eliminar umbrales definidos con anterioridad.		
Observaciones:	✓ El sistema debe lanzar una alerta de confirmación al administrador verificando que la acción que desea hacer es segura. ✓ Luego debe mostrar la vista de los umbrales restantes.		

2.5.2. Estimación del esfuerzo de cada historia de usuario.

La medida utilizada para la estimación del esfuerzo asociado a la implementación de cada una de las historias de usuario es el punto (máximo esfuerzo), la cual queda establecida por el equipo de desarrollo. Esta medida habitualmente toma valores de 1 a 3 en dependencia de la complejidad de la HU, partiendo de los factores que el equipo decida para clasificarlas. Estos factores por lo general son: la complejidad, el esfuerzo y el riesgo que demande cada una de ellas. Cuando una HU tiene tiempo de duración inferior a una semana significa que está muy bien descrita y que puede ser combinada con otra si muchas dificultades y cuando tiene más de 3 indica que deben ser divididas en partes (Maite Rodríguez Corbea y Mayelin Ordoñez Pérez, 2007). Para la estimación del esfuerzo en este acápite se tomó el método del punto asignando a cada historia de usuario la cantidad de días necesarios para su implementación dentro de una semana (1 punto o 5 días).

Tabla 6: Estimación del esfuerzo por historia de usuario(Fuente: elaboración propia.).

Número	Historia de Usuario	Puntos Estimado
1	Crear umbral.	1 punto
2	Eliminar umbral.	2/5
3	Modificar umbral.	3/5

4	Listar umbral.	1/5
5	Crear rol.	3/5
6	Eliminar rol.	2/5
7	Modificar rol.	2/5
8	Listar rol.	1/5
9	Eliminar usuario.	3/5
10	Modificar usuario.	2/5
11	Listar usuario.	2/5
12	Insertar *host.*	2
13	Eliminar *host*.	2/5
14	Crear reporte.	1 punto
15	Exportar reporte.	2/5
16	Configurar alerta.	1 punto
17	Visualizar recursos.	1 punto

2.5.3. Plan de Iteraciones.

Luego de ser identificada, descrita y estimada cada una de las HU se procede a planificar la fase de implementación del proyecto, la cual se realizará en 3 iteraciones. Al finalizar cada una de las funcionalidades, estas serán mostradas al cliente con el objetivo de detectar cambios necesarios o errores existentes. Las iteraciones estarán conformadas según se explica a continuación:

- ✓ ***Iteración 1***

La primera iteración persigue como objetivo la implementación de las historias de usuario de prioridad alta. Al finalizar esta iteración se realizará la entrega de la primera versión del sistema. Las historias de usuario a implementar en esta iteración son:(1, 3, 5, 6, 12, 14, 16 y 17).

- ✓ **Iteración 2**

El objetivo de esta segunda iteración es la implementación de las historias de usuario de prioridad media. Al finalizar la misma se contará con una versión funcional del sistema, donde se podrán probar las funcionalidades más relevantes del sistema. Las historias de usuario que serán implementadas en esta iteración son: (2, 7, 9, 10, 13 y 15).

- ✓ ***Iteración 3***

En la tercera iteración se realizará el desarrollo de las historias de usuario de prioridad baja. Al concluir esta se contará con la versión 1.0 del producto final, adicionando a las funcionalidades resultantes de esta iteración el producto final de la realización de las anteriores iteraciones. Como resultado final de esta iteración se contará con el sistema listo para su despliegue. Las historias de usuario a implementar son: (4, 8 y 11).

2.5.4. Plan de Duración de las Iteraciones.

XP como parte de su ciclo de vida crea el plan de duración de iteraciones, este plan se encarga de mostrar las historias de usuarios en el orden en que se implementarán en cada una de las iteraciones, así como la duración estimada de cada una de estas. A continuación, en la Tabla 7 se muestra el plan de duración de las iteraciones perteneciente al sistema en desarrollo.

Tabla 7: Plan de duración de las iteraciones(Fuente: elaboración propia.).

Iteración	Orden de la historia de usuario a implementar	Duración total
1	1. Crear umbral. 2. Modificar umbral. 3. Crear rol. 4. Eliminar rol.	7 semanas y 3 días

	5. Insertar *host*. 6. Crear reporte. 7. Configurar alerta. 8. Visualizar recurso.	
2	1. Eliminar umbral. 2. Modificar rol. 3. Eliminar usuario. 4. Eliminar *host*. 5. Modificar usuario. 6. Exportar reporte.	2 semanas y 3 días
3	1. Listar umbral. 2. Listar rol. 3. Listar usuario.	4/5 semana

2.5.5. Plan de entregas.

En este plan se detalla la fecha de fin de cada iteración, permite la entrega de una versión desarrollada del producto hasta ese momento al cliente, en aras de lograr entregar el producto final en la fecha establecida. A continuación, se muestra el plan de entrega para cada iteración comenzando la implementación el día 6 de marzo de 2017. A continuación, en la Tabla 8 de muestra el plan de entregas.

Tabla 8: Plan de entregas(Fuente: elaboración propia.).

Iteración	Fecha de entrega
1	27/4/2017
2	15/5/2017
3	22/5/2017

2.6. Fase de Diseño.

Es la segunda fase de la metodología de desarrollo utilizada, en esta fase los programadores definen la arquitectura de software, el patrón arquitectónico a utilizar según el *framework* de desarrollo, así como los patrones de diseño que ayudan a emplear buenas prácticas de programación, además del diseño del modelo físico de la base de datos y el diagrama de despliegue.

2.6.1. Arquitectura de Software.

La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente, y los principios que orientan su diseño y evolución.

En la arquitectura cliente-servidor, el cliente envía un mensaje al servidor solicitando un determinado servicio (petición), y el servidor envía uno o varios mensajes con la respuesta. Mediante esta arquitectura el usuario (cliente) puede acceder a la información sin tener en cuenta su ubicación física y donde pueda estar alojada la misma.

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema. La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa (Noe González Mendoza, 2010).

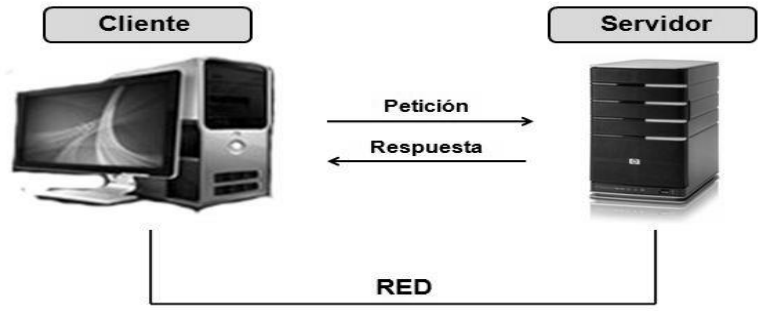


Ilustración 2. Arquitectura Cliente-Servidor (Fuente: elaboración propia.).

2.6.2. Patrón Arquitectónico.

Los patrones arquitectónicos son los que definen la estructura de un sistema, los cuales a su vez se componen de subsistemas con sus responsabilidades, también tienen una serie de directivas para organizar los componentes del mismo, con el objetivo de facilitar la tarea del diseño. Para el desarrollo del sistema el patrón

a seguir es Modelo-Vista-Plantilla (MTV) ya que uno de los *framework* a utilizar para el desarrollo de la solución es *Django*, el cual usa el Modelo–Vista–Plantilla como una modificación del patrón Modelo–Vista–Controlador, empleado en la implantación de la solución. En *Django* el modelo sigue siendo modelo, la vista pasa hacer la plantilla y el controlador la vista (Sergio Infante Montero, 2012).

- ✓ **Modelo:** la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos mostrando: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tienen, y las relaciones entre ellos.
- ✓ **Plantilla:** la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web u otro tipo de documento.
- ✓ **Vista:** la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: es como un puente entre los modelos y las plantillas.

Este patrón comienza su funcionamiento cuando el navegador envía una solicitud a la vista. La vista interactúa con la modelo para obtener los datos. La vista llama a la plantilla y esta envía la respuesta de la solicitud al navegador.

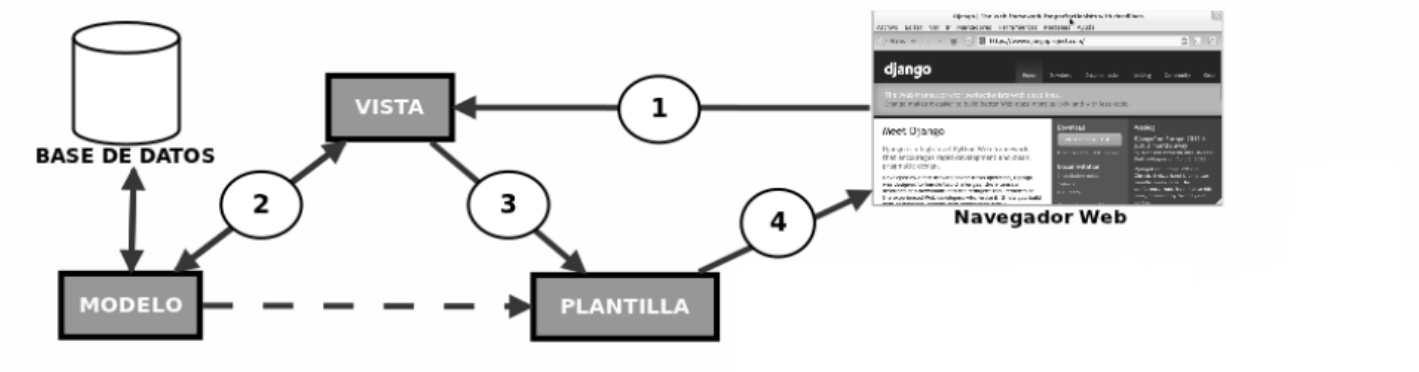


Ilustración 3: Funcionamiento del MTV de Django(Sergio Infante Montero, 2012).

2.6.3. Patrones de Diseño.

Un patrón de diseño es un conjunto de reglas que describen como afrontar tareas y solucionar problemas que surgen durante el desarrollo del software.

Los patrones GRASP[37] describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. A continuación, se mencionan y describen los patrones de diseño que se utilizaron durante el desarrollo del sistema (Maia Kord, 2011):

- ✓ **Experto en información:** Indica, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se logra un diseño con mayor cohesión y así la información se mantiene encapsulada disminuyendo el acoplamiento (Informáticos, 2005).

```
@login_required(login_url='/login/')
def user_list(request):
    users = User.objects.all()
    return render(request, 'admins/user/user_list.html', locals())
```

Ilustración 4: Código para obtener toda la información referente a los usuarios y mostrarla en la plantilla user_list.html(Fuente: elaboración propia.).

- ✓ **Controlador:** Asigna la responsabilidad de gestionar un mensaje de un evento del sistema a una clase controladora. Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado.

```
@login_required(login_url='/login/')
def user_delete(request, id):
    user = get_object_or_404(User, pk=id)
    user.delete()
    messages.success(request, 'Usuario eliminado satisfactoriamente.')
    return HttpResponseRedirect(reverse('user_list'))
```

Ilustración 5: Función para eliminar un usuario y mostrar el mensaje de satisfacción(Fuente: elaboración propia.).

- ✓ **Alta cohesión:** Consiste en asignar las responsabilidades teniendo en cuenta que permanezcan altamente cohesionados, es decir, que su utilización facilite la comprensión del diseño y el incremento

[37] Siglas en inglés para el término: *General Responsibility Assignment Software Patterns*.

de las capacidades de reutilización. Una alta cohesión permite que las clases con responsabilidades estrechamente relacionadas no realicen un trabajo enorme.

- ✓ **Bajo acoplamiento:** Plantea que debe existir una alta reutilización entre las funcionalidades de las clases con una mínima dependencia, contribuyendo así al mantenimiento de las mismas. Este patrón ya viene incluido con *Django* que permite un bajo acoplamiento entre las piezas, lo que evita las dependencias, por ejemplo, a la hora de realizar cambios en las configuraciones de las *URL*[38], la BD, plantillas HTML, etc., basta solo con realizarlo una sola vez.

Los patrones GoF[39] dan una solución implementable con su propio diagrama de clases que muestra la forma en que deben ser usados (Maia Kord, 2011).

- ✓ **Decorador:** Es un patrón estructural que extiende la funcionalidad de un objeto dinámicamente de tal modo que es transparente a sus clientes, utilizando una instancia de una subclase de la clase original que delega las operaciones al objeto original. Provee una alternativa muy flexible para agregar funcionalidad a una clase. Como ejemplo de este patrón se evidencia la utilización de los decoradores de Django (*@login_required, @permission_required*, etc.).

```
@login_required(login_url='/login/')
@permission_required(perm='change_user', login_url='/denegado')
```

Ilustración 6: funciones decoradoras de Django(Fuente: elaboración propia.).

2.6.4. Modelo Físico de la Base de Datos.

El modelo físico de la base de datos se obtiene del diagrama de clases persistentes, este modelo optimiza el rendimiento a la vez que asegura la integridad de los datos al evitar repeticiones innecesarias de datos, mostrando la relación que existe entre las entidades de la base de datos.

[38] Siglas en inglés para el término: *Uniform Resource Locator.*
[39] Siglas en inglés para el término: *Gang of Four.*

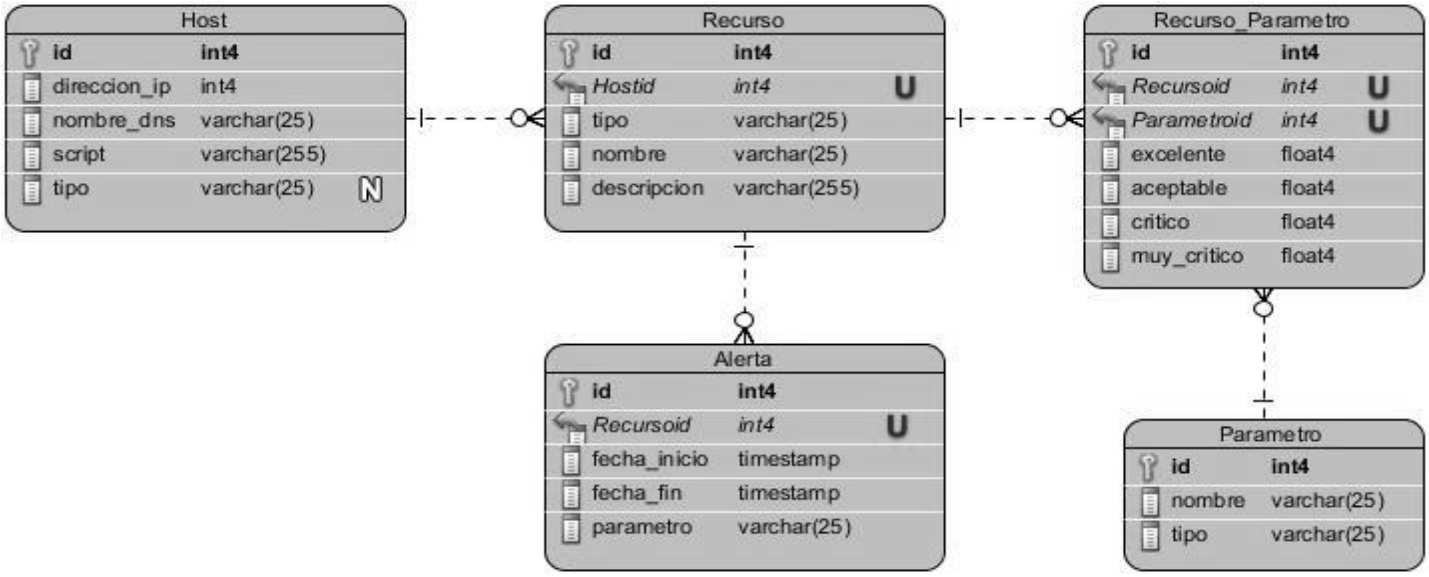


Ilustración 7: Modelo de datos del sistema(Fuente: elaboración propia.).

2.6.5. Diagrama de Despliegue.

Los diagramas de despliegue, son los complementos de los diagramas de componentes que unidos proveen la vista de implementación del sistema. Describen la topología del sistema la estructura de los elementos de hardware y el software que ejecuta cada uno de ellos. Los diagramas de despliegue representan a los nodos y sus relaciones. Los nodos son conectados por asociaciones de comunicación tales como enlaces de red, conexiones TCP/IP.

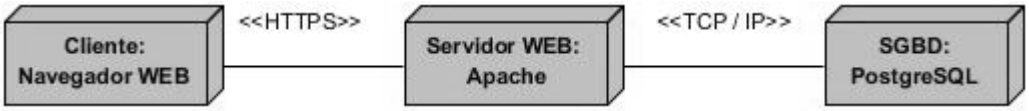


Ilustración 8: Diagrama de despliegue(Fuente: elaboración propia.).

2.7. Conclusiones parciales.

Una vez concluido el presente capitulo se arriba a las siguientes conclusiones:

- ✓ En un centro de datos con una infraestructura nube se necesita de la supervisión constante de los recursos, esto permitió identificar funcionalidades esenciales en la propuesta de solución.
- ✓ El uso de XP como metodología de desarrollo especificó la estructura, formato de los artefactos y documentación del proceso ingenieril.

- ✓ Se definió el uso de la arquitectura MTV y los patrones de diseño, utilizados para lograr una correcta estructuración del sistema.
- ✓ La distribución física del sistema quedó establecida con el modelo de despliegue.

CAPÍTULO 3. **IMPLEMENTACION Y PRUEBAS.**

En el siguiente capítulo se muestran las particularidades necesarias para describir el proceso de la implementación, para ello se definen los estándares de codificación que emplea el lenguaje de programación utilizado(*Python*), se hace alusión a la fase de prueba propuesta por la metodología de desarrollo la cual propone realizar pruebas unitarias y pruebas de aceptación para comprobar el cumplimiento de los requerimientos establecidos en las historias de usuarios.

3.1. Estándares de codificación.

XP enfatiza en la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible entre los miembros del equipo de desarrollo facilitando de esta manera los cambios que se puedan presentar.

De ahí se desprende la importancia de los estilos de programación, también conocidos como estándares o convenciones de código los cuales definen un grupo de convenciones para escribir código fuente en ciertos lenguajes de programación. A continuación, se define la relación de estándares de codificación a utilizar en la implementación del sistema (Guido Van Rossum y Barry Warsaw, 2007).

Tabla 9: Estándares de codificación de Python (Guido Van Rossum y Barry Warsaw, 2007).

Estándar	Descripción
Indentación	- Usa 4 espacios por cada nivel de indentación. - Para nuevos proyectos, se recomienda firmemente el uso de espacios en lugar de tabuladores. La mayoría de los editores tienen características que hacen esto bastante sencillo. - Las líneas de continuación deben alinearse verticalmente con el carácter que se ha utilizado (paréntesis, llaves, corchetes).
Tamaño máximo de línea	- Todas las líneas deben estar limitadas a un máximo de 79 caracteres. - Dentro de paréntesis, corchetes o llaves se puede utilizar la continuación implícita para cortar las líneas largas.

	- En cualquier circunstancia se puede utilizar el carácter "\" para cortar las líneas largas.
Líneas en blanco	- Separa las funciones no anidadas y las definiciones de clases con dos líneas en blanco. - Las definiciones de métodos dentro de una misma clase se separan con una línea en blanco. - Se puede utilizar líneas en blanco escasamente para separar secciones lógicas.
Codificaciones	- Utilizar la codificación UTF-8. - Se pueden incluir cadenas que no correspondan a esta codificación utilizando "\x", "\u" o "\U".
Importaciones	- Las importaciones deben estar en líneas separadas. - Siempre deben colocarse al comienzo del archivo. - Deben quedar agrupadas de la siguiente forma: 1. Importaciones de la librería estándar. 2. Importaciones terceras relacionadas. 3. Importaciones locales de la aplicación / librerías. - Cada grupo de importaciones debe estar separado por una línea en blanco.
Espacios en blanco en expresiones y sentencias	- Evitar utilizar espacios en blanco en las siguientes situaciones: ▪ Inmediatamente dentro de paréntesis, corchetes y llaves. ▪ Inmediatamente antes de una coma, un punto y coma o dos puntos. ▪ Inmediatamente antes del paréntesis que comienza la lista de argumentos en la llamada a una función. ▪ Inmediatamente antes de un corchete que empieza una indexación. ▪ Más de un espacio alrededor de un operador de asignación (u otro) para alinearlo con otro.

	- Deben rodearse con exactamente un espacio los siguientes operadores binarios: asignación (=), asignación aumentada (+=, -= etc.), comparación (==, <, >, !=, <>, <=, >=, *in, not in, is, is not), booleanos (and, or, not).* - No utilizar espacios alrededor del igual (=) cuando es utilizado para indicar un argumento de una función o un parámetro con un valor por defecto.
Comentarios	- Los comentarios deben ser oraciones completas. - Si un comentario es una frase u oración su primera palabra debe comenzar con mayúscula a menos que sea un identificador que comience con minúscula. - Nunca cambiar las minúsculas y mayúsculas en los identificadores de clases, objetos, funciones, etc. - Si un comentario es corto el punto final puede omitirse. Los comentarios de bloque generalmente consisten en uno o más párrafos construidos con frases completas, y cada frase debería terminar con un punto
Comentarios en bloque	- Deben estar indentados al mismo nivel que el código a comentar. - Cada línea de un comentario en bloque comienza con un numeral (#) y un espacio en blanco.
Comentarios en línea	- Se recomienda utilizarlos escasamente. - Se debe definir comenzando por un numeral (#) seguido de un espacio en blanco. - Deben ubicarse en la misma línea que se desea comentar.
Cadenas de documentación	- Deben quedar documentados todos los módulos, funciones, clases y métodos públicos. - Para definir una cadena de documentación debe quedar encerrada dentro de (“””). - Los (“””) que finalizan una cadena de documentación deben quedar en una línea a no ser que la cadena sea de una sola línea.

Convenciones de nombramiento	- Nunca se deben utilizar como simple caracteres para nombres de variables los caracteres ele minúscula “l”, o mayúscula “O”, ele mayúscula “L” ya que en algunas fuentes son indistinguibles de los números uno (1) y cero (0). - Los módulos deben tener un nombre corto y en minúscula. - Los nombres de clases al igual que los nombres de excepciones deben utilizar la convención “*CapWords*” (palabras que comienzan con mayúsculas). - Los nombres de las funciones deben estar escrito en minúscula separando las palabras con un guión bajo “_”. - Las constantes deben quedar escritas con letras mayúsculas separando las palabras por un guión bajo (_).

3.2. Validación de la propuesta de solución.

Uno de los pilares de la metodología de desarrollo utilizada es el proceso de pruebas. XP anima a probar constantemente tanto como sea posible. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones (Maite Rodríguez Corbea y Mayelin Ordoñez Pérez, 2007).

XP divide las pruebas en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final.

Las pruebas del sistema tienen como objetivo verificar la funcionalidad del sistema a través de sus interfaces externas comprobando que dicha funcionalidad sea la esperada en función de los requisitos del sistema. Generalmente las pruebas del sistema son desarrolladas por los programadores para verificar que su sistema se comporta de la manera esperada, por lo que podrían encajar dentro de la definición de pruebas unitarias que propone XP, aunque también pueden encajar dentro de la categoría de pruebas de aceptación. Las pruebas de aceptación son más importantes que las pruebas unitarias dado que significan la satisfacción del cliente con el producto desarrollado, el final de una iteración y el comienzo de la siguiente, por esto, el cliente

es la persona adecuada para diseñar las pruebas de aceptación (Maite Rodríguez Corbea y Mayelin Ordoñez Pérez, 2007).

3.2.1. Pruebas de Aceptación.

El objetivo de estas pruebas es verificar los requisitos, por este motivo, los propios requisitos del sistema son la principal fuente de información a la hora de construir las pruebas de aceptación, las cuales son creadas a partir de las historias de usuario, durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación.

El cliente o usuario especifica los aspectos a testear cuando una historia de usuario ha sido correctamente implementada. Una historia de usuario puede tener más de una prueba de aceptación, tantas como sean necesarias para garantizar su correcto funcionamiento y no se considera completa hasta que no supera sus pruebas de aceptación. Esto significa que debe desarrollarse un nuevo test de aceptación para cada iteración o se considerará que el equipo de desarrollo no realiza ningún progreso (Maite Rodríguez Corbea y Mayelin Ordoñez Pérez, 2007).

Una prueba de aceptación es como una caja negra, cada una de ellas representa una salida esperada del sistema. Es responsabilidad del cliente verificar la corrección de las pruebas de aceptación y tomar decisiones acerca de las mismas.

La garantía de calidad es una parte esencial en el proceso de XP. La realización de este tipo de pruebas y la publicación de los resultados debe ser los más rápido posibles, para que los desarrolladores puedan realizar con la mayor rapidez los cambios que sean necesarios.

A continuación, se muestran 3 de los casos de prueba diseñados para el sistema, los demás se encuentran en el Anexo # 2.

Tabla 10: CP # 1 Crear Umbral(Fuente: elaboración propia.).

Código:	SMS_CPA_1	**H.U:**	Crear umbral
Persona que realiza la prueba:	Adrián Pérez Prieto		

Descripción de la prueba:	Probar que se crea el umbral con la información correcta.
Condición de ejecución:	Estar autenticado en el sistema con el rol de administrador.
Entrada / Pasos de ejecución:	- Autenticarse en el sistema. - Acceder al menú Opciones/Administrar/Umbrales. - Se muestra una tabla con la información de todos los umbrales definidos hasta el momento.

Escenarios	Resultado esperado	Evaluación de la prueba
SC1: Seleccionar la opción añadir umbral.	Se muestra una vista donde se debe seleccionar el elemento de infraestructura al que se desee definir los umbrales.	Satisfactoria
SC2: Introducir la información correcta.	Luego de haber seleccionado el elemento de infraestructura se debe cargar un formulario en dependencia de este elemento, se debe introducir toda la información necesaria de forma correcta y al dar clic en el botón ACEPTAR mostrar un mensaje de confirmación que ha sido guardado en BD el nuevo umbral y volver a la vista inicial.	Satisfactoria
SC3: Introducir la información incorrecta o dejar campos obligatorios vacíos.	Al igual que en el **SC2** lo que introduciendo la información de forma incorrecta y dejando campos vacíos al dar clic en el botón ACEPTAR se debe mostrar un mensaje diciendo que existen errores en los datos y campos vacíos y debe mantenerse en la misma vista para volver a intentarlo.	Insatisfactoria

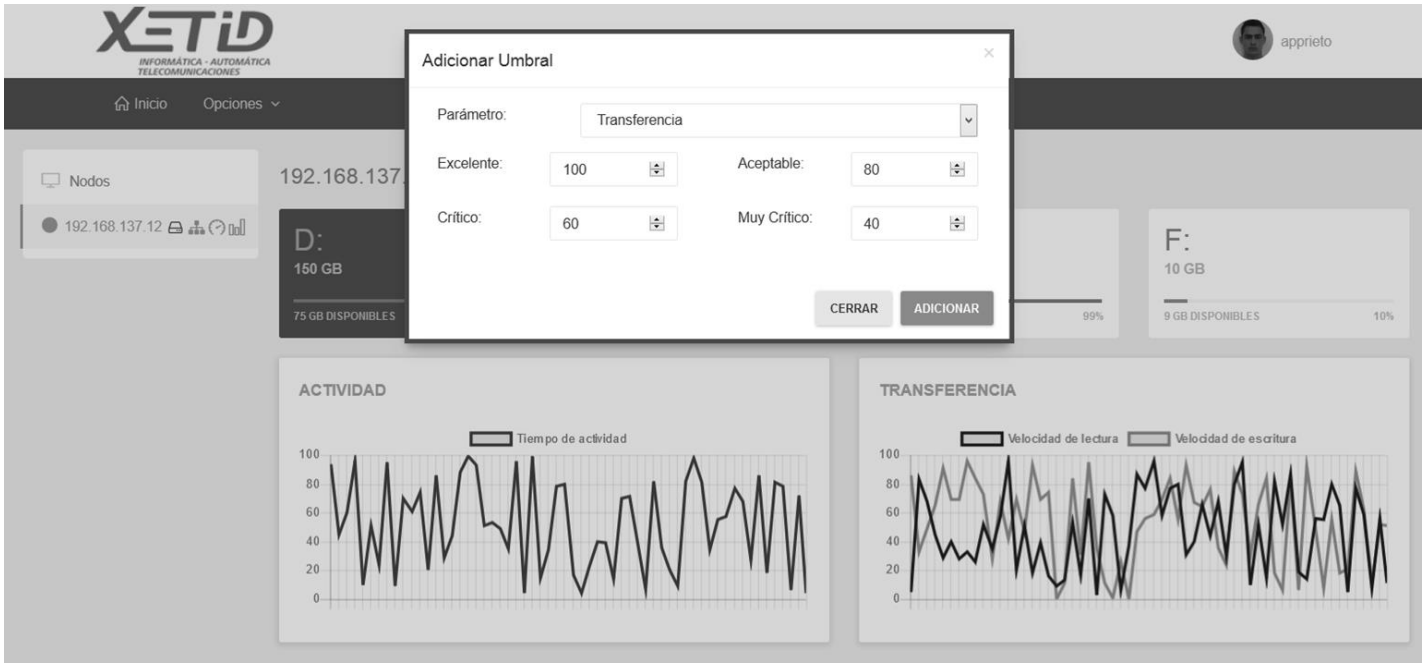


Ilustración 9: SC1 Seleccionar la opción añadir umbral(Fuente: elaboración propia.).

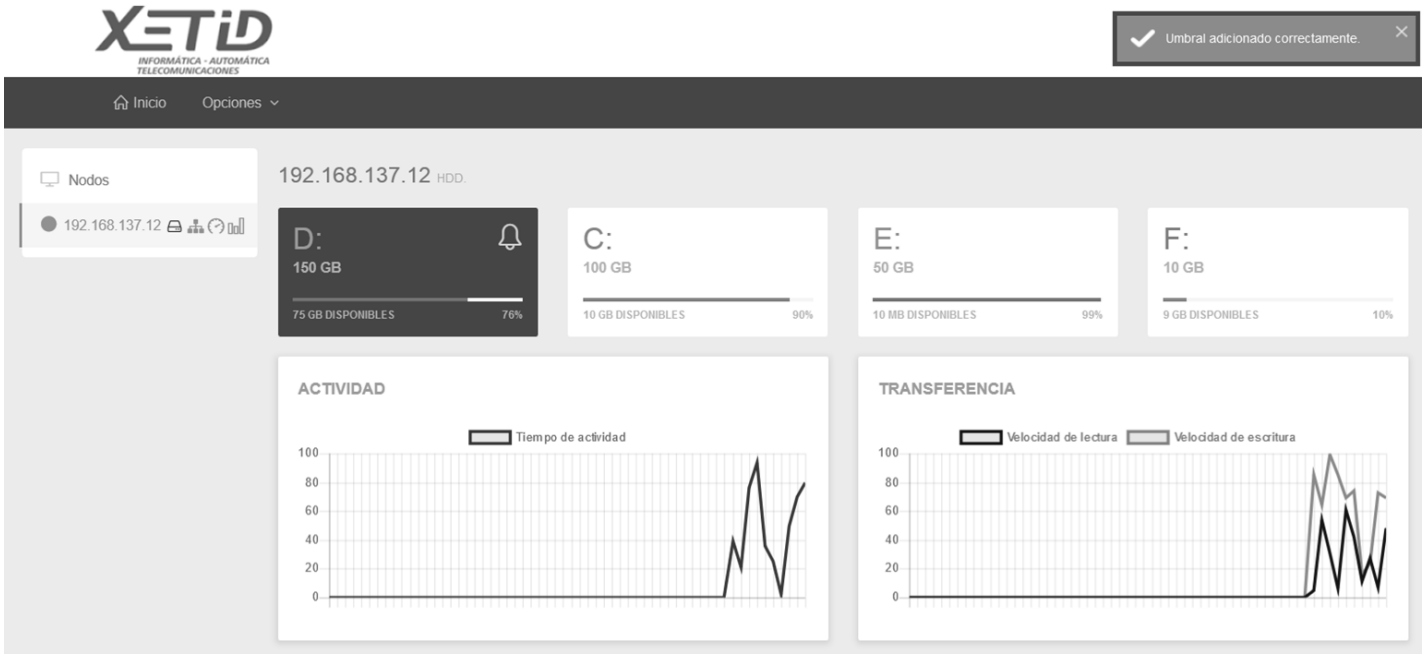


Ilustración 10: SC2 Introducir la información correcta(Fuente: elaboración propia.).

Tabla 11: CP # 2 Modificar Umbral(Fuente: elaboración propia.).

Código:	SMS_CPA_2	**H.U:**	Modificar umbral
Persona que realiza la prueba:	Adrián Pérez Prieto		
Descripción de la prueba:	Probar que se modifica el umbral con la información correcta.		
Condición de ejecución:	Estar autenticado como administrador en el sistema.		
Entrada / Pasos de ejecución:	- Autenticarse en el sistema. - Acceder al menú Opciones/Administrar/Umbrales. - Se muestra una tabla con la información de todos los umbrales definidos hasta el momento.		

Escenarios	**Resultado esperado**	**Evaluación de la prueba**
SC1: Seleccionar un umbral de la lista y acceder a la opción modificar.	Se deba mostrar una vista donde permita modificar los valores puestos anteriormente.	Satisfactoria
SC2: Introducir la información correcta.	Luego de haber seleccionado el elemento se debe introducir toda la información necesaria de forma correcta y al dar clic en el botón ACEPTAR mostrar un mensaje de confirmación que ha sido guardado en BD el umbral con los nuevos valores y volver a la vista inicial.	Satisfactoria

SC2: Introducir la información incorrecta.	Al igual que en el **SC2** lo que introduciendo la información de forma incorrecta y dejando campos vacíos al dar clic en el botón ACEPTAR se debe mostrar un mensaje diciendo que existen errores en los datos y campos vacíos y debe mantenerse en la misma vista para volver a intentarlo.	Satisfactoria

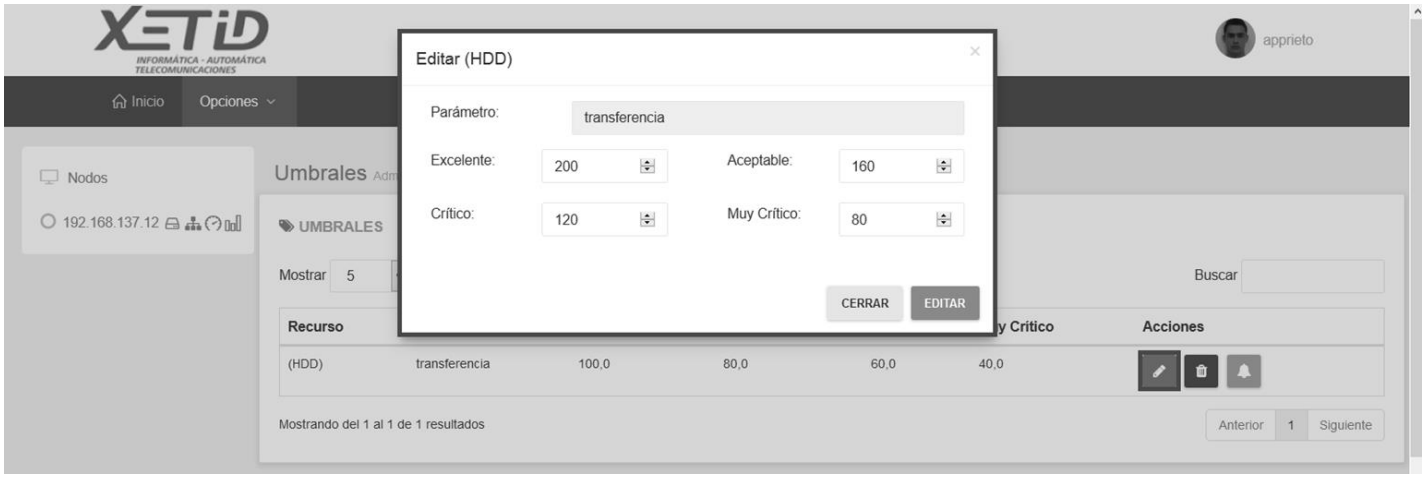


Ilustración 11: SC1 Seleccionar un umbral de la lista y acceder a la opción modificar(Fuente: elaboración propia.).

Tabla 12: CP # 3 Crear Rol(Fuente: elaboración propia.).

Código:	SMS_CPA_3	**H.U:**	Crear rol
Persona que realiza la prueba:	Adrián Pérez Prieto		
Descripción de la prueba:	Probar que se crea el rol con la información y los permisos asignados correctamente.		
Condición de ejecución:	Estar autenticado como administrador en el sistema.		
Entrada / Pasos de ejecución:	- Autenticarse en el sistema.		

	- Acceder al menú Opciones/Administrar/Roles. - Se muestra una tabla con la información de todos los roles definidos hasta el momento y la información de los permisos.

Escenarios	Resultado esperado	Evaluación de la prueba
SC1: Crear un nuevo rol y asignarle permisos.	Se debe mostrar una vista donde se muestre un formulario para añadir un nuevo rol y asignarle permisos sobre el sistema.	Insatisfactoria
SC2: Seleccionar los elementos correctos.	Al poner el rol nuevo y asignarle los permisos y dar clic en el botón ACEPTAR debe mostrar un mensaje de confirmación.	Satisfactoria

3.2.2. Pruebas Unitarias.

Las pruebas unitarias o *test* unitarios son lo más importantes para el practicante TDD[40]. Cada *test* unitario es un paso andado en el camino de la implementación del software. Todo *test* unitario debe ser: atómico, independiente, inocuo y rápido. Si no cumple estas premisas entonces no es un *test* unitario, aunque se ejecute con una herramienta tipo *xUnit*. Las pruebas unitarias son una forma de probar el buen funcionamiento de un módulo o una parte del sistema, con el fin de asegurar el correcto funcionamiento de todos los módulos por separado y evitar así errores futuros en el momento de la integración de todas sus partes.

Una de las actividades que conforma la fase de pruebas en XP es la refactorización, constante actividad de reestructuración del código. Dicha actividad tiene como objetivo remover la duplicidad, mejorar la legibilidad, simplificar y hacer más flexible la codificación, para facilitar los posteriores cambios.

Para garantizar la calidad del software en cuestión, el desarrollo de la aplicación se llevó a cabo utilizando una práctica llamada: Desarrollo guiado por pruebas o *TDD*, que presenta el siguiente ciclo de vida:

[40] Siglas en inglés para el término: *Test-Driven Development.*

- ✓ Confeccionar una prueba para que el código falle y así detectar vulnerabilidades.
- ✓ Escribir el código fuente para que pase dicha prueba.
- ✓ Llevar a cabo la optimización y refactorización del mismo, mientras que la prueba no falle.
- ✓ Repetir este proceso hasta que el proyecto esté completado.

Este flujo de trabajo permite un desarrollo incremental evitando tener que emplear tiempo en encontrar un error en particular, pues el desarrollador es capaz de ir directamente y cambiar la parte de la aplicación que esté presentando problemas. De esta forma es posible saber si la implementación de nuevas funcionalidades o cambios en las ya implementadas están afectando el comportamiento del sistema, solamente con ejecutar dichas pruebas.

Las pruebas unitarias se realizaron utilizando el IDE de programación *Pycharm*, que internamente hace uso del módulo *unittest.* Esto permitió la automatización del proceso y una mejor retroalimentación visual del estado del mismo.

A continuación, en la Ilustración 9 y 10 se muestra un ejemplo de las pruebas unitarias realizadas, el resultado de las de más relevancia pueden encontrarse en el Anexo # 3.

```
class PerfilTestCase(TestCase):
    def setUp(self):
        Perfil.objects.create(solapin="E100816", nombre="Adrian", apellidos="Perez Prieto",
                              categoria="estudiante",area="facultad 1",
                              foto="http://directorio.uci.cu/sites/all/modules/custom/"
                                   "directorio_de_personas/display_foto.php?id=")

    def test_perfil_categoria(self):
        perfil1 = Perfil.objects.get(nombre="Adrian")
        self.assertEquals(perfil1.categoria, "estudiante")

    def test_perfil_solapin(self):
        perfil2 = Perfil.objects.get(nombre="Adrian")
        self.assertEquals(perfil2.solapin, "E100816")
```

Ilustración 12: PU Para comprobar el modelo Perfil(Fuente: elaboración propia.).

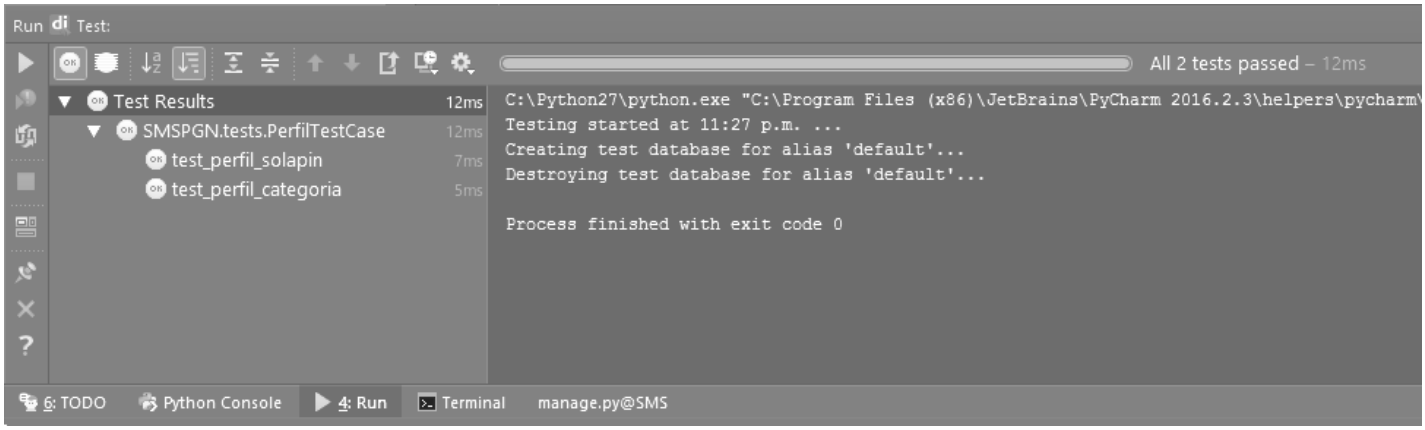


Ilustración 13: Resultado de la PU para comprobar el modelo Perfil(Fuente: elaboración propia.).

Fueron realizadas un total de 18 pruebas unitarias principalmente sobre los modelos definidos para verificar la correctitud de las consultas al SGBD y el tiempo de respuesta, que arrojaron resultados satisfactorios en todo momento.

3.2.3. No conformidades Detectadas.

Al realizar las pruebas, uno de los detalles a no pasar por alto son las no conformidades detectadas, las cuales se traducen en los errores encontrados y funcionalidades no deseadas por el cliente. Al final de cada iteración se le muestra al cliente una versión funcional del software de forma que pueda detectar aquellas no conformidades que serán corregidas al inicio de la siguiente iteración. La presente investigación está dividida en 3 iteraciones en las que fueron encontradas varias no conformidades, a continuación, en el Gráfico 1 se muestran los resultados de las no conformidades detectadas.

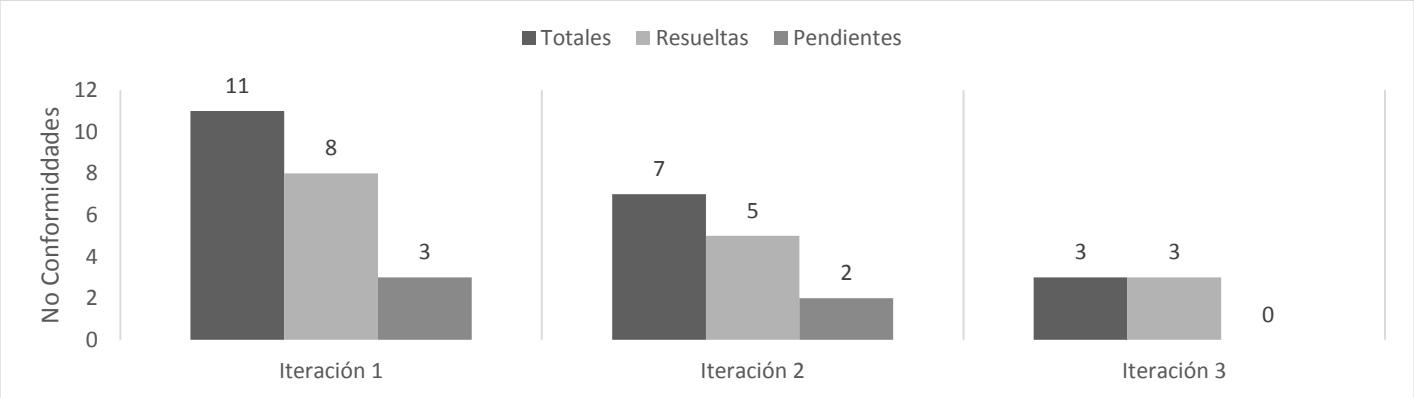


Gráfico 1: No conformidades detectadas(Fuente: elaboración propia.).

Como parte de la metodología seleccionada, las no conformidades encontradas en cada iteración son las primeras tareas a resolver de la iteración siguiente, siendo el cliente el encargado de ordenarlas por prioridad. Algunas de ellas al no ser críticas, son arrastradas a la siguiente iteración (pendientes). Llevando a cabo este proceso, se logran minimizar los niveles de aceptación de errores. De esta manera quedaron resueltas todas las no conformidades detectadas en la aplicación desarrollada.

3.3. Conclusiones Parciales.

- ✓ Los estándares de codificación definidos permitieron implementar códigos completamente legibles para el buen entendimiento de otros programadores facilitando así su recodificación.
- ✓ La realización de las pruebas unitarias y de aceptación permitieron comprobar el funcionamiento de las funcionalidades implementadas.
- ✓ La resolución de las no conformidades encontradas contribuyó a la entrega de un producto más completo y acorde con las exigencias del cliente.

CONCLUSIONES

Después de concluido el presente trabajo se arribó a las siguientes conclusiones:

- ✓ El desarrollo de la presente investigación permitió contribuir a la gestión de los elementos y/o recursos de infraestructura involucrados en un ecosistema nube.
- ✓ De acuerdo con los resultados obtenidos se puede afirmar que el trabajo cumple con las iniciativas de estandarización propuestas por el NIST, la UIT y la DMTF.
- ✓ La utilización de XP como metodología de desarrollo de software guió el proceso de desarrollo y estableció los artefactos que forman parte de la documentación del proceso ingenieril.
- ✓ La solución fue probada lo que permitió verificar el correcto funcionamiento de cada una de las historias de usuarios implementadas.

RECOMENDACIONES

Se recomienda para posteriores versiones que se incluyan elementos como:

- ✓ Un almacén de datos para la recuperación de todo tipo de información y que se logre archivar por largos periodos de tiempo.
- ✓ Además, se recomienda que se haga un estudio para integrarle elementos de inteligencia artificial que logren ante un incidente o desastre responder de forma autónoma.
- ✓ Integrar la solución al gestor nube *OpenNebula.*

BIBLIOGRAFÍA

1. **Sistemas de Reportes** - Inteligencia de Negocios. 2016. Accedido diciembre 4. https://sites.google.com/site/inteligenciadenegociossa/home/conceptos-basicos/1-2-4-sistemas-de-reportes.
2. **Alejandro Vega Velazques.** 2012. «Virtualizacion | VIRTUALIZACION.com». febrero 15. http://www.virtualizacion.com/tag/virtualizacion-4/.
3. **Amaris, Chris, Tyson Kopczynski, Rand Morimoto, y Alec Minty**. 2010. *Microsoft System Center Enterprise Suite Unleashed*. Sams.
4. **Angel Alonso Párrigas**. 2016. «Ossim, una plataforma clave para la seguridad en profundidad». En *Defensa en profundidad con OSSIM*. España. Accedido noviembre 25. http://www.angelalonso.es/doc-presentaciones/ossim-hakin9.pdf.
5. **Angel GabrielOlivera Sosa.** 2010. «Reporte de intalacion de apache». https://es.scribd.com/doc/37187866/Requerimientos-funcionales-y-no-funcionales.
6. **Beka Kezherashvili.** 2016. «Computación en la Nube». Almería: Universidad de Almería. Accedido noviembre 24. http://www.adminso.es/recursos/Proyectos/PFM/2011_12/PFM_cloud_beka.pdf.
7. **Casanova MG, y Rodríguez YM**. 2012. «Propuesta de Arquitectura de Gestión para una Nube privada que brinde Infraestructura como Servicio». La Habana: Instituto Superior Politécnico “José Antonio Echeverría” Facultad de Ingeniería Eléctrica.
8. **Fang Lui, Jin Yong, Robert Bohn, John Messina, Lee Badger, y Dawn Leaf**. 2011. «Recommendations of the National Institute of Standards and Technology». *Special Publication 5 00 - 292*, septiembre, 35.
9. **Focus Group, y Cloud Computing Technical Report.** 2012. «Functional requirements and reference architecture» Version 1.0 (febrero): 37.
10. **Garcia Perellada, Lilia Rosa**. 2014. «Propuesta de arquitectura para una Nube privada con soporte para Infraestructura como Servicio».
11. **Guia Rapida General.** 2016. Accedido noviembre 25. https://pandorafms.com/downloads/guias_rapidas.pdf.
12. **Guido van Rossum.** 2003. *Python Language Reference Manual*. http://knuth.luther.edu/~bmiller/CS151/Spring05/*Python*ref.pdf.
13. **Guido van Rossum, y Barry Warsaw**. 2007. «Guía de estilo del código *Python*». agosto 10. http://mundogeek.net/traducciones/guia-estilo-*Python*.htm.

14. **Hewlett Packard Enterprise (HPE) Support Help & Customer Service | HPETM**. 2016. Accedido diciembre 4. https://www.hpe.com/us/en/support.html.
15. **Informáticos**. 2005. «Patrones de Asignación de Responsabilidades (GRASP)». http://www.lsi.us.es/docencia/get.php?id=906.
16. **Julio Casal.** 2003. «Open Source Security Information Management». https://www.alienvault.com/docs/OSSIM-desc-es.
17. **Maia Kord**. 2011. «Patrones GRASP y GoF.» diciembre 3. https://sites.google.com/site/tuxnots/materias-de-la-facu/metodologia-de-sistemas/patronesgrasppatronesgofdiferenciaentregraspygof.
18. **Maite Rodriguez Corbea, y Mayelin Ordoñez Pérez**. 2007. «LA METODOLOGÍA XP APLICABLE AL DESARROLLO DEL SOFTWARE EDUCATIVO EN CUBA». La Habana: Universidad de las Ciencias Informáticas.
19. **Michael Badger**. 2008. *Zenoss Core: Network and System Monitoring*. 1.ª ed.
20. **Miranda ED.** 2013. «Experiencias iniciales en el despliegue de una Nube privada que brinda Infraestructura como Servicio». La Habana: Instituto Superior Politécnico “José Antonio Echeverría” Facultad de Ingeniería Eléctrica.
21. **Monitoring for Modern IT**. 2016. Zenoss Own IT. Accedido noviembre 25. https://www.zenoss.com/documentation/zsd/Zenoss_Service_Dynamics_Resour%20ce_Manager_Administration_26-032014-4.2-v10.pdf.
22. **Monitorización.** 2016. PandoraFMS. Accedido noviembre 25. https://pandorafms.com/ es/soluciones/monitorizacion-unificada/.
23. **Mundo Programas| Un mundo de Programas siempre full**. 2016. Accedido diciembre 7. http://www.mundoprogramas.net/jetbrains-pycharm-professional-v4-0-4-build-139-1001/.
24. **Nagios - The Industry Standard in IT Infrastructure Monitoring.** 2016. Nagios. Accedido noviembre 25. https://www.nagios.org/.
25. **Nagios Core - Nagios.** 2016. Nagios. The Industry Standard In IT Infrastructure Monitoring. https://www.nagios.com/products/nagios-core/.
26. **Nagios Plugins | The home of the official Nagios® Plugins.** 2016. Nagios Plugins. Accedido noviembre 25. https://nagios-plugins.org/.
27. **Noe Gonzalez Mendoza**. 2010. «Arquitectura cliente servidor». Educación, noviembre 16. https://es.slideshare.net/NoeGonzalezMendoza/arquitectura-cliente-servidor.

28. **Noemi Arbos, Luis Miguel Amoros, David González gnzález, y Antonio Oller Arcas**. 2010. «Servicios telemáticos sobre nubes privadas en plataformas virtualizadas y distribuidas». http://upcommons.upc.edu/handle/2117/10793.
29. **OpenNebula Documentation.** 2016. Accedido diciembre 4. http://docs.OpenNebula.org/5.2/.
30. **Patricio Letelier.** 2016. «Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)». España: Universidad Politécnica de Valencia (UPV). Accedido diciembre 7. http://www.cyta.com .ar/ta0502/b_v5n2a1.htm.
31. **PnadoraFMS Enterprise.** 2016. PandoraFMS. https://pandorafms.com/es/precios/.
32. **¿Qué es Django?** · Django Girls Tutorial». 2016. Accedido diciembre 7. https://tutorial.djangogirls.org /es/django/.
33. **Rakesh Kumar, Kanishk Jain, Hitesh Maharwal, Neha Jain, y Anjali Dadhich.** 2014. «Apache CloudStack: Open Source Infrastructure as a Service Cloud Computing Platform» 1 (Julio): 6.
34. **Requerimientos Funcionales y No Funcionales (RF/RNF).** 2017. Accedido febrero 23. http://ingenieriadesoftware.bligoo.com.mx/requerimientos-funcionales-y-no-funcionales-rf-rnf.
35. **Rihards Olups.** 2010. *Monitor your network hardware, server, and web performance effectively and efficiently*. https://www.packtpub.com/networking-and-servers/zabbix-18-network-monitoring.
36. **Ruben Aguilera Días Heredero**. 2016. «Primeros pasos con Cloud Foundry». *Adictos al Trabajo*. Accedido noviembre 24. https://www.adictosaltrabajo.com/tutoriales/cloud-foundry/.
37. **Sergio Infante Montero**. 2012. «CURSO DJANGO El framework para detallistas con deadlines.» En . http://www.maestrosdelweb.com/guias/#guias-django.
38. **The PostgreSQL Global Development Group**. 2016. Accedido diciembre 7. https://www.postgresql.org/files/documentation/pdf/9.2/postgresql-9.2-A4.pdf.
39. **Tim Jones**. 2009. «La anatomía de un hipervisor Linux». septiembre 31. http://www.ibm.com/ developerworks/ssa/library/l-hypervisor/.
40. **Valerie Kane, Paul Vancil, Vincent Kowalski, y Pail Lipton.** 2010. «Architecture for Managing Clouds, A White Paper from the Open Cloud Standards Incubator», junio. http://www.dmtf.org /sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf.
41. **Victor Zamora Yustres**. 2014. «Estudio del Cloud Computing y su interoperabilidad». España: Universidad Carlos III de Madrid. http://e-archivo.uc3m.es/handle/10016/22696.
42. **What is Visual Paradigm** 2016. Accedido diciembre 7. https://www.visual-paradigm.com/features/.

43. **Yosmay Morales Suárez.** 2016. «Plataforma de Virtualización para Nova Servidores». La Habana: Universidad de las Ciencias Informáticas.
44. **Zabbix The Enterprise-Class Open Source Network Monitoring Solution**. 2016. Accedido noviembre 25. http://www.zabbix.com/.
45. **Zamora Yustres, Víctor.** 2014. «Estudio del cloud computing y su interoperabilidad». http://e-archivo.uc3m.es/handle/10016/22696.
46. **Zenoss Updates Open Source Management Offering - Computer Business Review.** 2016. Accedido noviembre 25. http://web.archive.org/web/20080407193325/http://www.cbronline.com/article_news.asp? guid=5C9F141F-A7BE-4A97-B672-085A55FAE354.