Universidad de las Ciencias Informáticas
FACULTAD 6

Título: Componente de captura de métricas para el Sistema de monitorización de servidores PostgreSQL: Naire v2.0

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor: Denys Retureta Mailero

Tutor(es): Ing. Yoan M. Pérez Piñero

Ing. Rosnel Venero Acosta

La Habana, Cuba
"Año 54 de la Revolución"
Junio 2012

En la tierra hace falta personas que trabajen más y critiquen menos, que construyan más y destruyan menos, que prometan menos y resuelvan más, que digan mejor ahora que mañana.

Ernesto "Che" Guevara

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 6 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste se firma la presente a los ____ días del mes de _____ del año _____.

_____

Denys Retureta Mailero

Autor

_____ _____

Ing. Yoan M. Pérez Piñero

Tutor

Ing. Rosnel Venero Acosta

Tutor

DATOS DE CONTACTO

Autor:
Denys Retureta Mailero
Universidad de las Ciencias Informáticas, Habana, Cuba
e-mail: dretureta@estudiantes.uci.cu

Tutor:
Ing. Rosnel Venero Acosta
Universidad de las Ciencias Informáticas, Habana, Cuba
e-mail: rvacosta@uci.cu

Tutor:
Ing. Yoan Manuel Pérez Piñero
Universidad de las Ciencias Informáticas, Habana, Cuba
e-mail: ymperez@uci.cu

AGRADECIMIENTOS:

A mis tutores por siempre mostrar gran confianza en mí y guiarme de la mejor manera posible.

A mis amigos Reisel, Ariel, Jose, Frank, Hector, Magdiel, Adrian, Marlon y de manera muy especial a Jorge y a Ismel.

A Maylen por ser como una hermana para mí en estos 5 años y por siempre mostrarme que puedo contar con ella.

A mi amiga Betty y de manera muy especial quisiera agradecerle a Yeneysi por haber sido mi hada madrina, persona sin la cual mi trayecto de estudiante a ingeniero hubiera sido un poco tempestuoso.

"A todos los que de una forma u otra han hecho posible el sueño de muchos años."

DEDICATORIA:

A mi madre por siempre haberme apoyado en todo, aún cuando ella pensara que estaba equivocado y por educarme como un hombre de bien.

A mi padre por siempre estar ahí para mí, por aconsejarme y orientarme en la vida.

A mis hermanas Yaneidis y Yoanna y en especial a mi hermano Dany por ser mi modelo a seguir desde pequeño.

A mi abuelita por quererme tanto y siempre apoyarme en todo.

A mi familia por todo el apoyo que me han dado y por hacerme saber que nunca voy a estar solo.

RESUMEN

La Universidad de las Ciencias Informáticas promueve el desarrollo de la ciencia y la tecnología mediante la realización de proyectos productivos. Como parte del desarrollo y soporte a las tecnologías de bases de datos surge el Centro de Tecnologías de Gestión de Datos, en el cual se desea monitorizar y controlar el desempeño de los servidores PostgreSQL por la importancia que tiene saber los detalles sobre el comportamiento de los mismos por parte de los administradores.

El presente trabajo de diploma tuvo como objetivo el desarrollo de un componente de captura de métricas versión 2.0 para el Sistema de monitorización de servidores PostgreSQL, llamado DNaire-Client v2.0, software que permite a los administradores de bases de datos llevar un control específico del funcionamiento y rendimiento de los servidores PostgreSQL. Para ello se realizó un estudio de las herramientas de monitorización en tiempo real existentes y las métricas definidas para monitorizar servidores de bases de datos.

PALABRAS CLAVES

métricas, monitorización, PostgreSQL, Sistema, servidores

INDICE

INTRODUCCIÓN

La evolución de la informática a través del tiempo ha hecho necesario que el hombre se desarrolle junto con ella para poder adaptarse a las exigencias del mundo actual. La informatización sin precedentes que se lleva a cabo en la actualidad, ha contribuido a la aparición de nuevas y complejas aplicaciones, así como tecnologías que ya se utilizan en todos los aspectos de la vida cotidiana. Al ser la industria del software una de las más crecientes hoy en día, las empresas a través de la competencia se han visto obligadas a crear soluciones más rápidas y efectivas para solucionar los disimiles problemas que puedan presentar sus clientes.

Enmarcando cada situación en su momento histórico, actualmente no se puede hablar de educación, ni de cultura, ni de desarrollo si no se aplican eficientemente y en todas las esferas de la sociedad las nuevas Tecnologías de la Información y Comunicaciones, llamadas TICs. La informatización actualmente forma parte de la lucha por elevar la calidad de vida del pueblo cubano y lograr una sociedad cada vez más justa, equitativa y solidaria. (1).

Como producto del proceso de informatización de la sociedad se ha revolucionado la manera en la que se gestiona la información en las empresas cubanas, por lo que el crecimiento de los volúmenes de información en las mismas, así como su difícil manejo ha hecho imprescindible hacer uso de Sistemas Gestores de Bases de Datos (SGBD), para poder almacenar y acceder de manera rápida a toda la información generada.

Existen diversos SGBD como MySQL, SQLServer, Oracle y PostgreSQL, al ser la gran mayoría de estos software de tipo privativo, se les descartó, quedando como opción obvia a utilizar PostgreSQL, el cual es desarrollado por una comunidad internacional que colabora constantemente con el objetivo de convertirlo en uno de los SGBD de código abierto más avanzados de la actualidad y que a su vez posee su código fuente abierto Esto contribuiría enormemente a lograr que Cuba como país subdesarrollado alcance un alto nivel de independencia tecnológica.

La proliferación, en Cuba, del uso de PostgreSQL como gestor de bases de datos de código abierto, que también forma parte del gran proceso de migración que se está llevando a cabo hacia el software libre, ha generado la necesidad de contar con soluciones que faciliten la monitorización y generación de reportes del uso de varios servidores de bases de datos.

Introducción

La Universidad de las Ciencias Informáticas (UCI), es una de las instituciones que en el país está dando gran apoyo a la migración hacia el software libre. El departamento PostgreSQL perteneciente al Centro de Tecnologías de Gestión de Datos (DATEC), concibió en el año 2010 la idea de desarrollar un Sistema de monitorización de servidores para dicho SBGD, al cual se nombró Naire, el mismo permite la monitorización y control de varios servidores de bases de datos PostgreSQL a través de reportes, mediante gráficas y valores cuantitativos.

Actualmente los procesos fluyen de la siguiente manera: existe un demonio que no es más que un proceso no interactivo del sistema operativo, instalado en cada uno de los servidores PostgreSQL que se quiere supervisar, el mismo se encarga de recoger información de las bases de datos haciendo consultas directas al gestor y las envía hacia una base de datos MongoDB, a la cual accede la aplicación web con el objetivo de visualizar la información obtenida y construir reportes.

El proceso de comunicación entre estos componentes no se realiza en tiempo real, es decir no se puede escribir en la bases de datos y leer al mismo tiempo la información proveniente de ella, por lo que el usuario no estaría recibiendo la información que requiere en el preciso instante que la solicitó. Además cuando existen grandes volúmenes de información almacenados en la base de datos MongoDB el tiempo de respuesta de la aplicación aumenta considerablemente y el usuario tendría que pasar por un proceso engorroso de espera para obtener los datos esperados.

En función de lo antes expuesto se ha identificado el siguiente **problema de la investigación:** ¿Cómo disminuir el tiempo de respuesta en el proceso de monitorización en Naire?

Se define como **objeto de estudio** los Sistemas de monitorización de servidores de bases de datos, enmarcado en el **campo de acción** Componentes de monitorización a servidores de bases de datos PostgreSQL.

Para resolver el problema se trazó como **objetivo general**: Desarrollar el componente de captura de métricas versión 2,0 para Naire: Sistema de monitorización de PostgreSQL.

A partir del análisis del objetivo general se derivan los siguientes **objetivos específicos:**

1. Analizar las herramientas existentes.

2. Diseñar mecanismo de comunicación en tiempo real con el componente de reportes.

3. Implementar el componente de captura de métricas.

Para el cumplimiento de los objetivos antes expuestos, se definieron las siguientes **tareas de investigación:**

1. Revisión bibliográfica de los protocolos de comunicación en tiempo real.
2. Revisión bibliográfica de las tecnologías a utilizar.
3. Identificación de nuevas métricas.
4. Definición del mecanismo de envío y recepción de los datos en tiempo real.
5. Definición de clases.
6. Implementación del mecanismo de envío y recepción de los datos en tiempo real.
7. Implementación de las métricas identificadas.
8. Definición de las pruebas.
9. Validación de la implementación en base a las pruebas realizadas.

Luego de la realización de las tareas antes mencionadas se espera obtener el siguiente **resultado:**

Componente de captura de métricas 2,0 que disminuya el tiempo de respuesta en el proceso de verificar el rendimiento y desempeño de los servidores PostgreSQL.

La realización de esta investigación ha estado guiada por los siguientes **Métodos Científicos:**

Métodos Teóricos:

Análisis Histórico-Lógico: Se utilizó con el objetivo de conocer las tendencias actuales de los sistemas de monitorización en tiempo real existentes y realizar un estudio de nuevas métricas que hayan sido definidas por diferentes empresas para la monitorización de servidores de bases de datos en tiempo real. También permitió recopilar información para describir las tecnologías a utilizar, aprovechando elementos teóricos esenciales para el desarrollo de la aplicación.

Analítico-Sintético: Se utilizó con el fin de extraer lo esencial de la bibliografía consultada, .esto permite comprender sus características generales tras sintetizar éstas como un todo, concretando los elementos más importantes relacionados con los sistemas de monitorización en tiempo real.

Inductivo-deductivo: Facilita el análisis de elementos generales a elementos más particulares, o sea, se dispone de un grupo de conocimientos generales sobre las tendencias actuales y características generales de los sistemas de monitorización en tiempo real y se pretende llegar a enfocar lo conocido en función de la situación problemática presentada.

El presente trabajo se ha estructurado de la siguiente manera:

Capítulo 1: Fundamentación teórica.

El presente capítulo comprende un estudio de los Sistemas de monitorización, los SGBD, específicamente PostgreSQL, las herramientas de monitorización en tiempo real existentes en el mundo, así como las tecnologías existentes para este propósito, las tecnologías o herramientas mediante las cuales es posible comunicarse en tiempo real con la web y además la identificación de nuevas métricas utilizadas para realizar una monitorización eficiente del desempeño de los servidores de bases de datos, que no hayan sido definidas en la versión 1,0. También se mencionan las herramientas y tecnologías a utilizar para el desarrollo de la nueva versión del componente de captura de métricas de la presente investigación.

Capítulo 2: Descripción de la solución.

En el capítulo se describe el flujo actual de los procesos involucrados en la versión 1,0 y todos los agregados en esta nueva versión, que permitirá conocer particularidades del entorno donde se desarrollará el componente de captura de métricas en tiempo real. Además se propone redefinir la arquitectura de comunicación entre el componente de captura de métricas y el de reportes. También se describen las principales características de este, se definen los requerimientos e historias de usuario y se realiza un diagrama de clases del diseño que se utiliza como complemento de la metodología definida, para lograr un mejor entendimiento del componente.

Capítulo 3: Implementación y prueba de la solución.

Introducción

En el presente capítulo se realiza la validación del componente propuesto en el capítulo anterior. Se selecciona la estrategia de prueba a realizar, además se selecciona el método de validación de las pruebas definidas por la metodología XP, que permite verificar el correcto funcionamiento del componente. Se muestran además los resultados de estas últimas en cada una de las iteraciones del proceso de desarrollo de software.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

INTRODUCCIÓN

El presente capítulo comprende un estudio de los Sistemas de monitorización, los SGBD, específicamente PostgreSQL, las herramientas de monitorización en tiempo real existentes en el mundo, así como las tecnologías existentes para este propósito, las tecnologías o herramientas mediante las cuales es posible comunicarse en tiempo real con la web y además la identificación de nuevas métricas utilizadas para realizar una monitorización eficiente del desempeño de los servidores de bases de datos, que no hayan sido definidas en la versión 1,0. También se mencionan las herramientas y tecnologías a utilizar para el desarrollo de la nueva versión del componente de captura de métricas de la presente investigación.

1.1. Conceptos asociados a la investigación.

1.1.1. Servidores de Gestión.

Los sistemas de monitorización generalmente se despliegan en una PC que se encarga de servir como servidor a otros ordenadores que se conectan a él. Existen diferentes tipos de servidores, entre ellos se destacan los servidores de aplicaciones designados a veces como un tipo de middleware (software que conecta dos aplicaciones), servidores de chat los cuales permiten intercambiar información a una gran cantidad de usuarios ofreciendo la posibilidad de llevar a cabo discusiones en tiempo real y los servidores de bases de datos que se encargan de almacenar y manejar grandes y complejos volúmenes de datos. Estos últimos surgen para solucionar los problemas de las empresas que manejan grandes volúmenes de información.

Los servidores de gestión de bases de datos deben ser capaces de proporcionar un conjunto de herramientas que permitan administrar, gestionar y monitorizar el propio SGBD y a su vez garantizar servicios para mejorar el mismo de forma fiable, rentable y con alto rendimiento.

1.1.2. Sistemas Gestores de Bases de Datos.

Se denomina Sistema Gestor de Base de Datos al software que permite la utilización y/o actualización de los datos almacenados en una (o varias) base(s) de datos por uno o varios usuarios desde diferentes puntos de vista. El objetivo fundamental de un SGBD consiste en suministrar al usuario las herramientas que le permitan manipular, en términos abstractos, los datos, o sea, de forma que no le

sea necesario conocer el modo de almacenamiento de los mismos en la computadora, ni el método de acceso empleado (2).

Henry Korth, uno de los autores del libro *Fundamentos de las bases de datos*, define los sistemas de bases de datos como una colección de datos interrelacionados y un conjunto de programas para acceder a dicha información. Plantea que el objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto práctica como eficiente. Además permite gestionar grandes cantidades de información, lo que implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información (3).

Los SGBD en si son un tipo de software dedicado específicamente a servir de interfaz entre los usuarios, las base de datos y las aplicaciones que los utilizan. Los SGBD también permiten definir los datos a distintos niveles de abstracción y manipular dichos datos, garantizando la seguridad e integridad de los mismos.

1.1.3. El Sistema Gestor de Bases de Datos PostgreSQL.

PostgreSQL es un sistema de gestión de base de datos relacional orientada a objetos y libre. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada que se ha ganado una sólida reputación de fiabilidad, integridad y corrección de los datos. Se ejecuta en los principales sistemas operativos, incluyendo Linux, UNIX, Mac OS X, Solaris y Windows. Tiene soporte completo para claves foráneas, uniones, vistas, disparadores y procedimientos almacenados (en varios idiomas). Cuenta con interfaces nativas de programación para C/C++, Java, Net, Perl, Python, Ruby y posee una documentación excepcional. Es altamente escalable, tanto en la enorme cantidad de datos que puede manejar, como en el número de usuarios concurrentes que puede acomodar (4).

PostgreSQL es uno de los sistemas de gestión de bases de datos de código abierto más avanzado del mundo y en sus últimas versiones posee muchas características que solo se podían ver en productos comerciales de alto calibre, lo que ha llevado a que sea el SGBD más usado entre los libres existentes.

1.1.4. Monitorización a servidores de bases de datos.

El término monitorización está relacionado con la acción de controlar o supervisar continuamente una o varias actividades, o sea es el seguimiento rutinario de la información prioritaria de un programa durante un tiempo establecido.

También se puede decir que en su concepción más amplia, la monitorización es una herramienta de gestión y de supervisión para controlar el avance de los proyectos, programas o planes en ejecución, que proporciona información sistemática, uniforme y fiable, permitiendo comparar los resultados con lo que se planificó. (5)

Haciendo referencia a la teoría de la planificación del desarrollo también se podría definir el seguimiento o monitorización como: *"…un ejercicio destinado a identificar de manera sistemática la calidad del desempeño de un sistema, subsistema o proceso a efecto de introducir los ajustes o cambios pertinentes y oportunos para el logro de sus resultados y efectos en el entorno..."* (6).

1.1.5. Sistemas en Tiempo Real

Un sistema de tiempo real (STR) es un sistema informático que interacciona repetidamente con su entorno físico y además responde a los estímulos que recibe del mismo dentro de un plazo de tiempo determinado. Algunos autores definen STR como:

"…cualquier sistema donde el tiempo en que se produce su salida es significante. Esto es debido a que generalmente la entrada corresponde a algún instante del mundo físico y la salida tiene relación con ese mismo instante. El retraso transcurrido entre la entrada y la salida debe ser lo suficientemente pequeño para considerarse una respuesta puntual…" (7)

La monitorización en tiempo real a servidores de bases de datos surgió a partir de la necesidad de verificar o controlar ciertos indicadores que permitieran evaluar el funcionamiento de los mismos para así poder conocer su rendimiento total. Como parte de esos parámetros se podría mencionar el tiempo de respuesta, el consumo de los recursos del servidor, o el comportamiento del mismo ante determinadas operaciones. El hecho de poder conocer los valores de estos indicadores, ayuda enormemente al mejoramiento del funcionamiento de estos servidores, así como poder dar una respuesta adecuada y en el menor tiempo posible ante cualquier situación que se pueda presentar en el funcionamiento de los mismos.

1.1.6. Protocolos de Comunicación en tiempo real.

Un Protocolo de Comunicación en tiempo real se define como: una serie de normas que usan los equipos informáticos para gestionar sus diálogos en los intercambios de información. Dos sistemas diferentes se pueden comunicar sin problemas en el caso que usen el mismo protocolo de

comunicaciones. Vinculado con la evolución de internet se ha visto la aparición de nuevos protocolos a los que se han ido adaptando los productos de cada fabricante, para asegurarse la compatibilidad con el resto de los productos existentes.

- **XMPP**

El Protocolo extensible de mensajería y comunicación de presencia, XMPP por si siglas en inglés, consiste en un conjunto de tecnologías abiertas para la mensajería instantánea, presencia, chat multi-partido, llamadas de voz y video, colaboración, middleware[1] ligero, sindicación de contenidos y generalización de enrutamiento de datos XML.

XMPP fue desarrollado originalmente por la comunidad Jabber siendo este de código abierto ofreciendo así un proceso abierto, seguro, libre de spam[2] y a la vez alternativa descentralizada a los servicios de mensajería instantánea de código cerrado en ese momento. (8).

XMPP permite la fácil implementación de un cliente en casi cualquier lenguaje de programación. Otra de las características relevantes de este protocolo es que la IETF[3] formalizó XMPP como uno de los protocolos básicos para la comunicación de los servicios de mensajería instantánea. Otra importante característica es su arquitectura descentralizada, mediante la cual le es posible a cualquier persona o empresa implementar y poner a disposición de los usuarios su propio servidor. Además el hecho de usar XML[4] como formato de intercambio de datos es posible lograr un alto nivel de interoperabilidad.

- **STOMP**

El Protocolo de mensajería orientado a Streaming de texto, STOMP por sus siglas en inglés, es un protocolo que proporciona una gran interoperabilidad de mensajería, fácil y generalizada entre los muchos lenguajes, plataformas y servidores para el paso de mensajes. STOMP es un protocolo muy

[1] Middleware es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones.
[2] Spam no es más que correo basura o mensaje basura, no deseados o de remitente no conocido.
[3] La Internet Engineering Task Force (IETF) es una gran comunidad internacional abierta, de diseñadores, operadores, vendedores, e investigadores interesados en la evolución de la arquitectura de Internet y el buen funcionamiento de la Internet.
[4] XML es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

sencillo y fácil de implementar, que fue creado en la escuela de diseño de HTTP[5], el servidor puede ser difícil de implementar, pero es muy fácil escribir un cliente para conseguir conectarse. (9)

- **IRC**

IRC (*Internet Relay Chat*) es un protocolo de comunicación en tiempo real basado en texto y a la vez también es un sistema de teleconferencia, que (a través del uso del modelo cliente-servidor) se adapta bien a que se ejecute en muchas máquinas de forma distribuida. Una configuración típica consiste en un único proceso (el servidor) que forma un punto central para los clientes (u otros servidores) para conectarse a, realizar el necesario mensaje de entrega/multiplicación y otras funciones. El protocolo IRC ha sido desarrollado en sistemas que utilizan el protocolo TCP/IP de la red, aunque no es necesario que este sea el único ámbito en el que opere. (10)

La investigación realizada permitió seleccionar el protocolo STOMP para la implementación del sistema de comunicación en tiempo real, ya que este protocolo trabaja con el formato de datos JSON el cual presenta una estructura muy similar a los documentos de las BD NoSQL MongoDB y además permite enviar los datos en tiempo real en el mismo formato que se mostrarán haciendo uso de las librerías JavaScript seleccionadas para el trabajo en el Componente de Reportes.

1.1.7. Patrones de Mensajería

Se puede definir un patrón de mensajería como un patrón de arquitectura orientado al trabajo de las aplicaciones de redes, mediante el cual se describe cómo dos partes diferentes de un sistema se conectan y se comunican entre sí a través del paso de mensajes.

- **Publicación-Subscripción**

Es un patrón de mensajería mediante el cual una persona o una aplicación puedan enviar una solicitud de información pública y a través de una notificación de eventos se transmite a todos los suscriptores autorizados. En general, la relación entre el editor y el suscriptor está mediada por un servicio que recibe las solicitudes de publicación y el envió de las notificaciones de los eventos para los suscriptores, permite además manejar las listas de las personas o las aplicaciones que están autorizadas a publicar o suscribirse. El punto focal para la publicación y la suscripción es un "nodo"

[5] Hypertext Transfer Protocol o HTTP (en español *protocolo de transferencia de hipertexto*) es el protocolo usado en cada transacción de la World Wide Web.

para que los editores puedan enviar los datos y del cual los suscriptores reciben notificaciones de eventos. Los nodos también pueden mantener un historial de eventos y prestación de servicios que complementan el modelo Publicación-Subscripción. (11)

- **Petición-Respuesta**

Es un patrón de intercambio de mensajes en los que un solicitante envía un mensaje de solicitud a un sistema de respuesta que recibe y procesa la solicitud, para en última instancia, devolver un mensaje de respuesta. Este es un patrón de mensajería simple, pero potente que permite que dos aplicaciones puedan tener una *'conversación'* de dos vías entre sí por un canal. Este patrón es especialmente común en las arquitecturas cliente-servidor (12)

El patrón de mensajería seleccionado es Publicación-Subscripción ya que mediante este patrón el usuario o aplicación es capaz de subscribirse a un canal específico de información, logrando así una diferenciación de la misma. Una vez subscrito, se les puede enviar información constantemente sin que las aplicaciones clientes la soliciten.

1.1.8. Tecnologías para la comunicación en tiempo real con la Web

Existen muchas tecnologías y técnicas para lograr comunicación en tiempo real con la web, muchas de ellas se han desarrollado a partir de AJAX[6], algunos ejemplos de esto son:

- **Comet**

No es una tecnología en sí, sino un conjunto de técnicas mediante las cuales se puede describir e implementar un modelo de diseño web mediante el cual una petición HTTP que se encuentre abierta puede permitir al servidor enviar datos al navegador, sin que este último los solicite.

Algunas personalidades como Alex Russell, fundador y presidente de la Fundación Dojo, se refiere a Comet como: las aplicaciones que pueden suministrar datos al cliente en cualquier momento, no sólo en respuesta a la entrada del usuario. Los datos se entregan en una sola conexión, previamente abierta. Este enfoque reduce la latencia para la entrega de datos de manera significativa.

[6] AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo Web para crear aplicaciones interactivas o RIA (Rich Internet Applications).

La arquitectura se basa en una vista de datos que está orientada a eventos en ambos lados de la conexión HTTP. El único cambio sustancial es que el punto final es el navegador.

Aunque Comet es similar al Ajax en que es asincrónico, las aplicaciones que implementan este estilo se pueden comunicar los cambios de estado con una latencia prácticamente inapreciable. Esto hace que sea conveniente para muchos tipos de aplicaciones de colaboración de vigilancia y multi-usuario que de otro modo sería difícil o imposible de manejar en un navegador sin necesidad de plugins[7]. (13)

Algunos ejemplos de sitios que implementan esta técnica:

- Integración de Gtalk en Gmail (www.google.com/talk/).
- Meebo (www.meebo.com)

- **BOSH**

Stream-Bidireccional sobre HTTP sincrónico, BOSH por sus siglas en inglés, es una tecnología para la comunicación bidireccional a través del Protocolo de transferencia de hipertexto (HTTP). BOSH es significativamente eficaz con más ancho de banda, además es más sensible que la mayoría de los otros protocolos de transporte bidireccional basada en HTTP y técnicas conocidas como AJAX. BOSH proporciona eficiencia y baja latencia, pero lo hace sin recurrir a la respuesta HTTP fragmentada como se hace en la técnica conocida como Comet. Hasta la fecha, BOSH ha sido utilizado principalmente como medio de transporte para el tráfico intercambiado entre Jabber/XMPP clientes y servidores (por ejemplo, para facilitar las conexiones de los clientes web y de clientes móviles en redes intermitentes). Sin embargo, BOSH no está ligada exclusivamente a XMPP y puede ser utilizado para otros tipos de tráfico. (14)

- **WebSockets**

Define un canal de comunicación bidireccional que opera a través de un solo socket en la web. WebSockets representa un gran avance, sobre todo para tiempo real y aplicaciones web controladas por eventos. (15). WebSockets ofrece una conexión bidireccional entre el servidor y el cliente. Esta conexión también es en tiempo real y está permanentemente abierta hasta el cierre de la misma de

[7] Un plugin es una Aplicación que se relaciona con otra para aportarle una nueva función

forma explícita. Esto significa que cuando el servidor quiere enviar algo a su cliente, el mensaje se inserta en su navegador de inmediato. (16)

- **Orbited**

Ofrece un socket de JavaScript/HTML en el navegador. Se trata de una pasarela y firewall web que permite integrar aplicaciones web con arbitrarios sistemas. Con esta herramienta se puede implementar cualquier protocolo de red en el navegador, sin necesidad de recurrir a plugins. Además se puede tener soporte para los protocolos, IRC, XMPP y STOMP. Además esta aplicación sólo utiliza los estándares web. (17)

Haciendo uso de Orbited se pueden conectar las aplicaciones clientes a cualquier servicio TCP/IP (servidores de IRC, aplicaciones personalizadas y sistemas de Mensajería Orientada a Middleware (MOM) que utilicen el protocolo STOMP como son RabbitMQ y ActiveMQ). (18)

La investigación realizada permitió seleccionar Orbited como tecnología web para la comunicación en tiempo real con el Componente de Reportes, Orbited es la implementación de Comet usando Python como lenguaje de programación, por lo que se pueden crear aplicaciones web de tiempo real. Esta herramienta está desarrollada usando como base a Twisted el cual es un framework para el desarrollo de aplicaciones orientadas al trabajo con las redes y basado en eventos que permite utilizar tanto XMPP, IRC como STOMP.

1.1.9. Mensajería Orientada a Middleware (MOM).

La Mensajería Orientada a Middleware (MOM) es una de las categorías de middleware que proporcionan conectividad de programa a programa, mediante el paso de mensajes. MOM generalmente soporta múltiples protocolos y comprende una infraestructura que brinda soporte fiable y escalabilidad de alto rendimiento para las redes distribuidas de las distintas aplicaciones. La mayoría de los software MOM se implementan con la capacidad de colas de almacenamiento y reenvío mensajes, es decir, Middleware de Encolado de Mensajes, MQM por sus siglas en inglés. En general, MOM proporciona una base de alto rendimiento para la interoperabilidad de aplicaciones en entornos diversos y complejos. (19)

Algunos de los productos MOM que brindan soporte para STOMP, XMPP e IRC son:

- Apache ActiveMQ.

- MorbidQ (Servidor STOMP desarrollado en Python usando Twisted como framework para el trabajo de redes).
- RabbitMQ (Servidor de encolado de mensajes, escrito en Erlang, que cuenta con soporte para STOMP).

La selección de MOM está dada porque haciendo uso de este tipo de software, se puede lograr trabajar con cualquier protocolo siempre que el servidor MOM lo soporte, además no es necesario implementar un servidor para manejar los mensajes, también brinda una gran interoperabilidad, lo que quiere decir que mediante su uso se pueden comunicar dos sistemas arbitrarios como si formaran parte de la misma aplicación.

1.2. Herramientas que monitorizan servidores de aplicaciones y bases de datos.

Para que las empresas puedan tener un control específico y regulado del funcionamiento de sus sistemas informáticos, una magnífica opción sería la monitorización en tiempo real a sus servidores, con el objetivo de coleccionar información y estadísticas que permitan analizar el desempeño de los mismos y de los servicios que brindan.

Con el objetivo de hacer de la monitorización una tarea sencilla se han creado con el tiempo una gran cantidad de herramientas, capaces de realizar la monitorización de muchas aplicaciones no importa lo complejas que estas sean. Algunas de estas herramientas se basan en la monitorización de aplicaciones o servidores específicos y en muchos casos tienen soporte para la monitorización de múltiples aplicaciones o servidores tales como, sistemas operativos, servidores de aplicaciones, servidores de bases de datos, servidores web y muchos otros tipos de herramientas.

Entre las diferentes herramientas que realizan la monitorización a servidores de bases de datos se encuentran:

Nagios: Es un sistema de monitorización de redes de código abierto ampliamente utilizado, que vigila los equipos (hardware) y servicios (software) que se especifiquen, alertando cuando el comportamiento de los mismos no sea el deseado. Entre sus características principales figuran la monitorización de servicios de red, de los recursos de sistemas hardware, independencia de sistemas operativos y la posibilidad de programar plugins específicos para nuevos sistemas (20)**.** Esta herramienta presenta

una interfaz con la cual es muy difícil trabajar y además como herramienta de monitorización no está destinada para los servidores de bases de datos sino más bien para servicios de red.

Application Manager: Provee capacidades de monitoreo para los servidores Linux fuera del sistema. Ayuda a asegurar que los servidores se encuentran funcionando y que se ejecuten a su máximo desempeño monitoreando la utilización del CPU, utilización de la memoria, procesos, utilización, entre otros (21). Este producto de la empresa ManageEngine es una solución de monitorización sin agentes, lo que no es muy conveniente para los administradores pues hay que establecer una conexión remoto a cada servidor y el monitoreo se realiza servidor a servidor.

ManageEngine que ofrece capacidades integrales de monitorización al sistema gestor PostgreSQL, para ello proporciona diferentes métricas, clasificadas de la siguiente manera:

- Métricas de supervisión.

  -Utilización del disco / memoria.

  -Estadísticas de conexión.

- Métricas de rendimiento.

  -Escaneo de índices.

  -Transacciones.

- Métricas de realización de una consulta.

  -Estadísticas de consultas.

  -Bloqueo de las estadísticas

Postgres Enterprise Manager: ha sido diseñado para administradores de bases de datos e instalaciones con bases de datos de grandes volúmenes. Su arquitectura distribuida gestiona todas las instancias de PostgreSQL, en las instalaciones o máquinas virtuales. Los agentes de vigilancia de recogida y envío de datos permiten tener un sistema de administración centralizada al cual se puede acceder desde múltiples consolas o Interfaces de Usuario. Los tableros de control de rendimiento proporcionan métricas personalizadas para la memoria, Entrada/Salida, almacenamiento, la actividad

de objetos, sistema operativo y sistema en espera, cada uno con sus propios gráficos y además permite generar alertas a las condiciones de excepción (22). Esta es una de las herramientas más recientes de EnterpriseDB una empresa norteamericana dedicada a crear soluciones de nivel empresarial para el trabajo con PostgreSQL, enfocada directamente al trabajo de los administradores de bases de datos.

Esta empresa define un conjunto de métricas de seguimiento para realizar la monitorización a servidores de bases de datos PostgreSQL, estas están agrupadas en varias categorías

- Métricas de disponibilidad
- Métricas de desempeño.

  -Transacciones.

  -VACUUM / ANALYZE.
- Métricas de rendimiento.

  -Commits / Rollbacks.

  -Bloqueos / Buffer Hits.
- Métricas de utilización.

  -Filas insertadas, actualizadas y eliminadas.

  -Escaneo de índices.

Hyperic HQ: es una herramienta desarrollada por Spring Source, compañía especialista en el desarrollo de aplicaciones empresariales. La misma se centra en la monitorización de aplicaciones y la gestión del rendimiento para infraestructuras virtuales y físicas, además proporciona estadísticas de rendimiento sobre las aplicaciones y servidores de bases de datos, web y servicios de red más comunes (23). Los elementos clave de la arquitectura son el servidor central, que proporciona una gestión centralizada y la persistencia de los datos y el agente o demonio, que permite el seguimiento y control de cada plataforma.

La definición de su arquitectura permite monitorizar aplicaciones mediante el uso de un modelo basado en agentes. Un agente Hyperic se ejecuta en cada equipo que se desea administrar. Al ejecutarlo por primera vez, el agente auto-descubre los recursos de software que residen en la máquina y periódicamente hace un re-análisis de los cambios de configuración. Los agentes Hyperic reúnen información sobre disponibilidad, utilización, rendimiento, además guardan logs o archivos de registros y realizan seguimiento de eventos, también permiten iniciar acciones para el control del software ejemplo: iniciar y detener los servidores web y de aplicaciones. Los agentes Hyperic envían sus datos de inventario y métricas para el Servidor Hyperic Central**.** El servidor central recibe los datos y los almacena en la base de datos *(HQ Database)* (24). La Figura 2 representa la arquitectura de dicha herramienta**.**

Figura 1: Arquitectura de Hyperic HQ.

Esta herramienta al ser desarrollada por Spring Source, división de VMware, empresa americana líder en virtualización, plantea en su contrato de licencia con Hyperic, que el software es de origen estadounidense y se prohíbe las operaciones de exportación del mismo a cualquier persona que sea ciudadano, nacional o residente del gobierno de Cuba (25).

Tomando como punto de partida lo anteriormente expuesto se puede llegar a la conclusión de que la obtención de las herramientas antes mencionadas no asegura, ni mucho menos contribuye a alcanzar la soberanía tecnológica para Cuba, como parte del proceso de informatización de la sociedad y aunque algunas de estas herramientas son de código abierto no cumplen con las necesidades que podría satisfacer la herramienta que se quiere desarrollar y las demás pues al ser aplicaciones desarrolladas en Estados Unidos, pues se suman a todos los demás servicios y tecnologías que no se pueden utilizar por el simple hecho de ser cubanos. Aunque se debe aclarar que tomando referencia de algunas como Hyperic HQ se puede lograr un mejor enfoque a la hora de lograr la concepción del producto deseado y en especial de su arquitectura basada en agentes, la cual sin lugar a dudas

proporciona una mayor organización a la hora de llevar a cabo el proceso de monitorización, brindando además la posibilidad de monitorizar en tiempo real más de un servidor de forma centralizada.

La monitorización a servidores de bases de datos puede también realizarse haciendo uso de técnicas como: revisión de logs y consultas al catálogo, el cual es una de las herramientas más poderosas que tiene PostgreSQL como SBGD, esto es un poco complejo ya que requiere conocimientos avanzados del mismo. Existen además otras herramientas capaces de realizar monitorización a PostgreSQL como son: PgBench, PgFouine y PgAdmin aunque no son las más adecuadas para monitorizar un conjunto de servidores de forma centralizada.

Existen otras técnicas de monitorización a servidores de PostgreSQL como los benchmarks que permiten medir el rendimiento del mismo, ya sea con datos ficticios o aleatorios, como con datos reales de una base de datos. Este tipo de evaluaciones suele consumir muchos recursos de memoria, en especial cuando se hacen pruebas de stress sobre los servidores para conocer los límites de prestación del mismo (26).

Los sistemas de monitorización que sin duda tienen una mayor importancia son aquellos que realizan sus funciones en tiempo real y a varios servidores a la vez de forma centralizada. Unos de los papeles fundamentales en estos sistemas lo constituye el agente o demonio encargado de la monitorización. Este tiene la responsabilidad de recopilar en tiempo real, los valores de los indicadores a través de resultados cuantitativos desde los diferentes servidores de bases de datos.

Hyperic al ser otra de las empresas que desarrolla aplicaciones orientadas a la monitorización, para esto define métricas para supervisar el comportamiento de servidores de bases de datos PostgreSQL (24), organizadas en cuatro grupos fundamentales:

- Métricas generales

  -Backends / Bloqueos.

  -Commits / Rollbacks.

  -Uso de espacio en disco / Uso de espacio de índices.

- Métricas de tablas

-Número de filas insertadas.

-Número de filas actualizadas.

-Número de filas eliminadas.

- Métricas de índices.

-Escaneo de índices.

-Lectura de índices.

- Métricas específicas para el control PostgreSQL.

-Analyze.

-Reset Statistics.

-Vacuum.

1.3. Metodología de Desarrollo.

Programación Extrema o XP por sus siglas en ingles hace hincapié en el trabajo en equipo. Los gerentes, clientes y desarrolladores son socios iguales en un equipo de colaboración. Esta metodología además implementa un ambiente simple pero eficaz permitiendo a los equipos ser altamente productivos. El equipo se auto-organiza en torno al problema, para así resolverlo lo más eficientemente posible. El primer proyecto XP se inició 06 de marzo 1996. Programación Extrema es uno de los varios procesos ágiles populares, el cual se centra en la satisfacción del cliente.

XP mejora un proyecto de software en cinco aspectos esenciales, la comunicación, la sencillez, la retroalimentación, el respeto y el coraje. Los programadores que utilizan XP están en constante comunicación con sus clientes y colegas programadores. Mantienen su diseño simple y limpio. Reciben retroalimentación probando su software desde el primer día. Entregan el sistema a los clientes tan pronto como sea posible y ponen en práctica los cambios cuando se sugiere. Cada pequeño éxito profundiza su respeto por las contribuciones únicas de cada miembro del equipo, todos y cada uno. Con esta base, los programadores XP son capaces de responder con valentía a las necesidades cambiantes de la tecnología. (27).

1.4. Métricas a aplicar en el proceso de monitorización a servidores de bases de datos.

Para poder realizar una monitorización eficiente y adecuada a servidores de bases de datos, es de suma importancia definir métricas, estas tienen como objetivo analizar el rendimiento y funcionamiento de los mismos, para poder hacer análisis comparativos con los resultados obtenidos, mostrando así una mejora o retroceso en los servidores y poder de esta manera optimizar estos servidores.

La palabra métrica puede hacer referencia a muchos significados, por lo que es muy común asociarla con medición y medida. Algunos autores asocian la palabra métrica a:

Una medida *"proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto*" (28).

Sin embargo, la *medición "es el proceso por el cual los números o símbolos son asignados a atributos o entidades en el mundo real tal como son descritos de acuerdo a reglas claramente definidas" (29).*

Mientras que las métricas se pueden definir como: "Una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado" (30).

Hacer uso de métricas proporciona un valor cuantitativo sobre el estado del proceso de monitorización, dando a la vez estimaciones de las bases de datos y el comportamiento de manera general.

La definición de métricas permitirá establecer patrones de comportamiento para los servidores de bases de datos que serán monitorizados, partiendo de que éstas no pueden interpretar por sí solas un concepto medible, pues necesitan de indicadores. Estos son de gran importancia y tienen gran utilidad porque sirven de base para cuantificar conceptos medibles para una necesidad de información a métodos cuantitativos de evaluación o predicción y a su vez ofrecen información para la toma de decisiones (31).

Las diferentes clasificaciones que se le pueden atribuir a las métricas van de la mano de las necesidades y prioridades que tenga cada empresa, otorgándole mayor importancia a las operaciones y aplicaciones críticas, para así poder asegurar el correcto funcionamiento de los servidores de bases de datos. Teniendo en cuenta lo anteriormente planteado se seleccionan y definen las siguientes métricas e indicadores para aplicar al componente a desarrollar en su versión 1.0, las cuales se clasifican en dos categorías fundamentales.

Métricas genérales.

- **Uso de espacio en disco.**

  Descripción: Esta métrica se refiere al porciento de utilización del disco.

  ✓ **Indicadores**

  - Espacio en uso por PostgreSQL.
  - Espacio libre en disco.
  - Espacio en uso por el sistema.
  - Capacidad total del disco.

- **Uso de espacio en memoria.**

  Descripción: Esta métrica se refiere al porciento de utilización de la memoria RAM.

  ✓ **Indicadores**

  - Espacio libre en memoria.
  - Espacio en uso por el sistema.
  - Capacidad total de la memoria.

- **Conexiones.**

  Descripción: Esta métrica se refiere a la cantidad de conexiones de red que se están haciendo al servidor PostgreSQL.

  ✓ **Indicadores**

  - Cantidad de conexiones al servidor PostgreSQL (Backends).

- **Procesos activos por el servidor PostgreSQL.**

Descripción: Esta métrica se refiere a la cantidad de procesos activos que hay en el servidor.

✓ **Indicadores**

- Cantidad de procesos activos por PostgreSQL.

Métricas de Bases de Datos.

- **Transacciones.**

Descripción: Esta métrica se refiere a la cantidad de transacciones confirmadas y fallidas en cada base de datos

✓ **Indicadores:**

- Cantidad de Commits[8] por bases de datos.
- Cantidad de Rollbacks[9] por bases de datos.

- **Estadísticas de tablas.**

Descripción: Esta métrica se refiere al uso de las tablas por bases de datos, respecto a la cantidad de filas insertadas, actualizadas, eliminadas, retornadas, vivas, muertas, leídas por escaneos secuenciales y escaneos secuenciales en las mismas

✓ **Indicadores:**

- Cantidad de filas insertadas por bases de datos.
- Cantidad de filas actualizadas por bases de datos.
- Cantidad de filas eliminadas por bases de datos.
- Cantidad de filas retornadas por bases de datos.

[8] Transacciones completadas con éxito en una base de datos.
[9] Transacciones fallidas en una base de datos.

- Cantidad de filas vivas por bases de datos
- Cantidad de filas muertas por bases de datos.
- Cantidad de filas leídas por escaneos secuenciales.
- Cantidad de escaneos secuenciales.

- **Uso de los índices**

  Descripción: Esta métrica se refiere al uso de los índices.

  ✓ **Indicadores:**

  - Total de Índices.
  - Cantidad de índices escaneados.

Para la versión 2.0 del componente a desarrollar la definición de las nuevas métricas a implementar quedan de la siguiente manera:

Métricas genérales.

- **Estadísticas de la memoria virtual.**

  Descripción: Esta métrica se refiere a las estadísticas de la memoria virtual.

  ✓ **Indicadores**

  - Espacio libre en memoria virtual.
  - Espacio en uso por el sistema virtual.
  - Capacidad total de la memoria virtual.

- **Estadísticas del CPU.**

  Descripción: Esta métrica se refiere a las estadísticas del CPU.

✓ **Indicadores**

- Descripción del CPU.
- Cantidad de CPUs.
- Arquitectura del CPU.
- Tiempo de usuario.
- Tiempo del sistema.
- Tiempo ocioso.
- Tiempo para reiniciar procesos.
- Tiempo que la CPU pasa esperando para que ocurra una operación de Entrada/Salida.
- Interrupciones por segundo.
- Cambios de contexto
- Utilización de la CPU.

- **Estadísticas de los dispositivos de bloques (disco duro).**

  Descripción: Esta métrica se refiere a las estadísticas de los dispositivos de bloques (disco duro)

  ✓ **Indicadores**

  - Modelo del disco duro.
  - Ruta del dispositivo.
  - Transferencias por segundo.
  - Escrituras y lecturas de bloques en KB.

- Escrituras y lecturas de bloques por segundo en KB.

- **Estadísticas de la memoria.**

  Descripción: Esta métrica se refiere a las estadísticas de la memoria.

  ✓ **Indicadores**

  - Memoria activa e inactiva.
  - Buffer y cache.

- **Estadísticas de la interfaz de red.**

  Descripción: Esta métrica se refiere a las estadísticas de la interfaz de red.

  ✓ **Indicadores**

  - Cantidad de bytes enviados y recibidos.
  - Cantidad de bytes enviados y recibidos por segundo.
  - Cantidad de paquetes recibidos y enviados.

1.5. Tecnologías y herramientas a utilizar.

El uso de tecnologías y herramientas es fundamental para el desarrollo de aplicaciones. Para el desarrollo del componente de captura de métricas se utilizarán las siguientes tecnologías definidas en el documento de Arquitectura de Sistema de monitorización para PostgreSQL.

Sistema Operativo: Ubuntu 11.04

Ubuntu 11.04, distribución GNU/Linux basada en Debian GNU/Linux, concentra su objetivo en la facilidad de uso, la libertad de uso, lanzamientos regulares y la facilidad en la instalación. También emplea excelentes herramientas de traducción y accesibilidad, proporciona un entorno robusto y funcional, así como entre sus características se encuentra que el escritorio predeterminado es GNOME líder como escritorio y como plataforma de desarrollo (32).

Sistemas de Control de Versiones: Git v1.7.9

Git es un sistema distribuido de control de versiones de código abierto. Está diseñado para manejar todo, desde pequeños a los proyectos más grandes con gran velocidad y eficiencia. (33).

Gestión de Proyecto: Redmine (GESPRO v1.0)

Redmine es una herramienta para la Gestión de Proyecto de código abierto, multiplataforma y liberado bajo los términos de Licencia Pública General (GPL), con una administración e interface web bastante amigable y fácil de usar (34).

Herramienta de Modelado: Visual Paradigm for UML 8.0

Es una herramienta profesional y multiplataforma, que permite construir diagramas, generar código desde los diagramas y documentación, también brinda al equipo de desarrollo de software la posibilidad de realizar el análisis y diseño de los sistemas con mayor eficacia. Además utiliza UML como lenguaje de modelado (35), con UML se puede:

1 Visualizar: permite expresar de una forma gráfica un sistema para que otras personas puedan entenderlo.

2 Especificar: esto quiere decir que se puede especificar las características del sistema.

3 Construir: el sistema se puede construir a partir de los diseños

Lenguaje de Programación: Python 2.7.3

Para la implementación se ha optado por utilizar Python, ya que es un lenguaje que presenta una serie de ventajas que lo hacen muy robusto (36), entre las que se puede contar:

1 Es un lenguaje expresivo y un programa Python suele ser más corto que su equivalente en lenguajes como C.

2 Es muy legible, su sintaxis es elegante y permite la escritura de programas cuya lectura resulta más fácil que si se usara otro lenguaje de programación.

3 Ofrece un entorno interactivo que facilita la realización de pruebas.

4 Puede usarse como lenguaje imperativo procedimental o como lenguaje orientado a objetos.

5 Posee un rico juego de estructuras de datos que se pueden manipular de modo sencillo.

Base de datos: MongoDB 2.0.4

MongoDB es una nueva generación de sistemas de gestión de base de datos que combina un modelo de almacenamiento de documentos con un potente motor de consultas de una manera simple. Es una base de datos del tipo NoSQL orientada a documentos (JSON), entre sus características fundamentales se encuentran (37):

1 Libre de esquemas: las claves de un documento no están predefinidas o fijas en ninguna forma, esto permite las migraciones masivas de datos y ofrece a los desarrolladores flexibilidad para el trabajo con los modelos de datos.

2 Fácil escalado: su modelo de datos orientado a documentos permite que los datos sean divididos a través de la red, ofrece balanceo de datos y de carga a través del clúster.

3 Almacenamiento de funciones JavaScript, colecciones de tamaño fijo, almacenamiento de ficheros y un buen rendimiento.

Entorno de desarrollo: Geany v0.21

Geany. Es un editor de Texto ligero basado en Scintilla[10] con características básicas de entorno de desarrollo integrado (IDE). Está disponible para distintos sistemas operativos, como GNU/Linux, Mac OS X, BSD, Solaris y Microsoft Windows. Es distribuido como software libre bajo la Licencia Pública General de GNU. Tiene soporte para muchos lenguajes de programación distintos, como C, C++, Java, JavaScript, PHP, HTML, CSS, Python, Perl, Ruby, Fortran, Pascal, Haskell entre otros (38). Algunas de las características más destacadas de Geany son:

1. Autocompletado
2. Soporte multi-documento
3. Soporte de proyectos

[10] Scintilla es un componente libre de edición de código fuente.

4. Coloreado de sintaxis

5. Emulador de terminal incrustado.

Conclusiones del capítulo

En el capítulo se han resumido diferentes aspectos teóricos relacionados con los Sistemas de monitorización, los sistemas gestores de bases de datos, la monitorización a servidores de aplicaciones y bases de datos. Se explicó que son los sistemas en tiempo real, así como los protocolos utilizados para este fin, también se estudiaron las tecnologías más relevantes de comunicación en tiempo real con la web. La investigación realizada permitió seleccionar STOMP como protocolo de comunicación, así como Publicación-Subscripción como patrón de mensajería. Se presentaron algunas de las herramientas que realizan monitorización en tiempo real a servidores, cómo lo hacen y las métricas que emplean, lo que permitió la identificación de nuevas métricas que se incluirán a la versión en desarrollo del componente de captura de métricas, teniendo en cuenta la importancia y necesidad que tiene el empleo de estas últimas para verificar el desempeño de los servidores a monitorizar. También se abordaron brevemente las tecnologías y herramientas a utilizar para desarrollar el componente.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN.

INTRODUCCION

En el presente capítulo se describe el flujo actual de los procesos involucrados en la versión 1,0 y todos los agregados en esta nueva versión, que permitirá conocer particularidades del entorno donde se desarrollará el componente de captura de métricas en tiempo real. Además se propone una arquitectura de comunicación entre el componente de captura de métricas y el de reportes. También se describen las principales características de éste, se definen los requerimientos e historias de usuario y se realiza un diagrama de clases del diseño que se utiliza como complemento de la metodología definida, para lograr un mejor entendimiento del componente.

2.1. Flujo actual del proceso.

Si algún usuario desea conocer el desempeño de los servidores PostgreSQL, debe acceder al Componente de Reporte para visualizar la información deseada, la cual se obtiene a través de consultas directas al catálogo, vistas estadísticas definidas por el SGBD o comandos de Linux y PostgreSQL. Actualmente esta información que se obtiene no llega en tiempo real al componente de reporte, sino que es guardada en una base de datos NoSQL al cual se accede para obtener la información de las métricas anteriormente guardadas, por otro lado al encontrarse el demonio de monitorización enviando los datos cada un segundo es lógico que la cantidad de datos almacenados crezca rápidamente, lo cual hace difícil para el Componente de Reporte cargar dicha información para así mostrarla.

2.2. Modelo de dominio.

El Modelo de Dominio o Conceptual puede utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema. Este modelo es utilizado por el analista como un medio para comprender el sector de negocios al cual el sistema va a servir. En la Figura 2 se presentan los conceptos identificados en el proceso de monitorización a servidores de bases de datos.

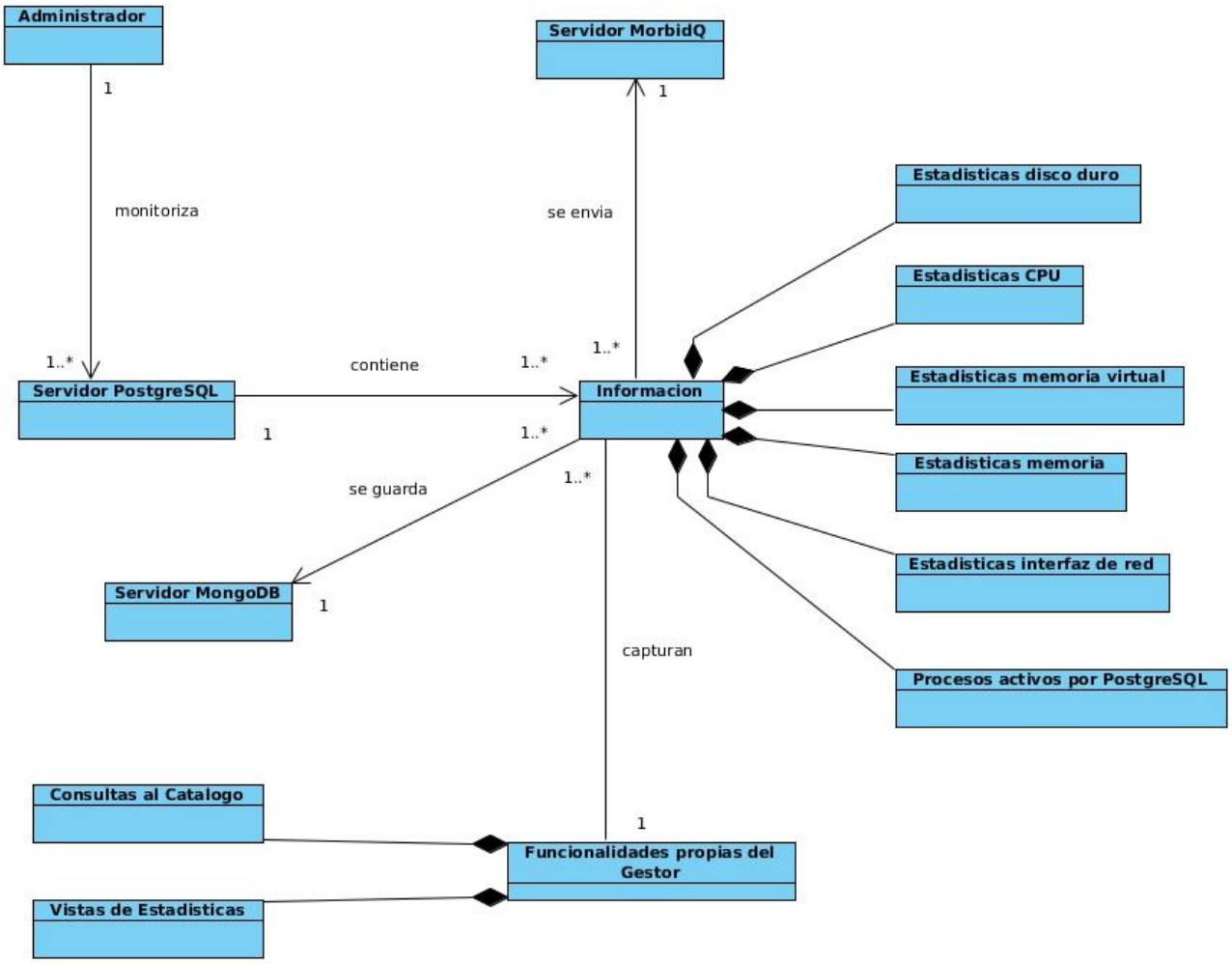


Figura 2: Modelo de dominio del componente de captura de métricas.

2.3. Arquitectura de comunicación en tiempo real para Naire.

Al definir una arquitectura de comunicación en tiempo real para Naire entre el componente de captura de métricas y el componente de reportes se tuvo en cuenta que, aunque son componentes de una misma aplicación, son además aplicaciones heterogéneas, o sea un demonio que estará en cada uno de los servidores a monitorizar y una aplicación web, la cual hará recepción de los datos enviados en tiempo real.

Atendiendo los puntos antes expuestos se puede lograr una integración de comunicación en tiempo real entre las dos aplicaciones eligiendo primeramente como se va a implementar la recepción de los datos por parte de la aplicación web.

El patrón de mensajería seleccionado es Publicación-Subscripción, esto está dado porque mediante este patrón aunque se envíen todos los datos a la vez hacia la aplicación web, solo se mostraría a través de la subscripción especifica de canales lo que el usuario desee ver. Otro punto es que si se enviara la información de todos los servidores hacia la aplicación web, aun utilizando Publicación-Subscripción se sobrecargaría el flujo de datos haciendo que la aplicación en cuestión perdiera rendimiento, por lo que se hace uso de la Mensajería Orientada a Middleware (MOM) que no es más que un tipo aplicación que hace posible la comunicación entre aplicaciones heterogéneas mediante el paso de mensajes.

A partir de la investigación realizada se decide utilizar STOMP como protocolo de comunicación, el hecho de utilizar este protocolo y no otro como XMPP que también soporta el patrón de mensajería seleccionado está dado porque de utilizar XMPP se tendría que utilizar XML como formato de trasmisión de datos, lo cual haría que de una forma u otra la aplicación perdiera en eficiencia, ya que toda la información tendría que convertirse de los diccionarios de Python que poseen un formato similar al JSON a XML para después volver a convertir a JSON para mostrar con las librerías de gráficas en la web. La Figura 3 representa la arquitectura propuesta para Naire en esta nueva versión.

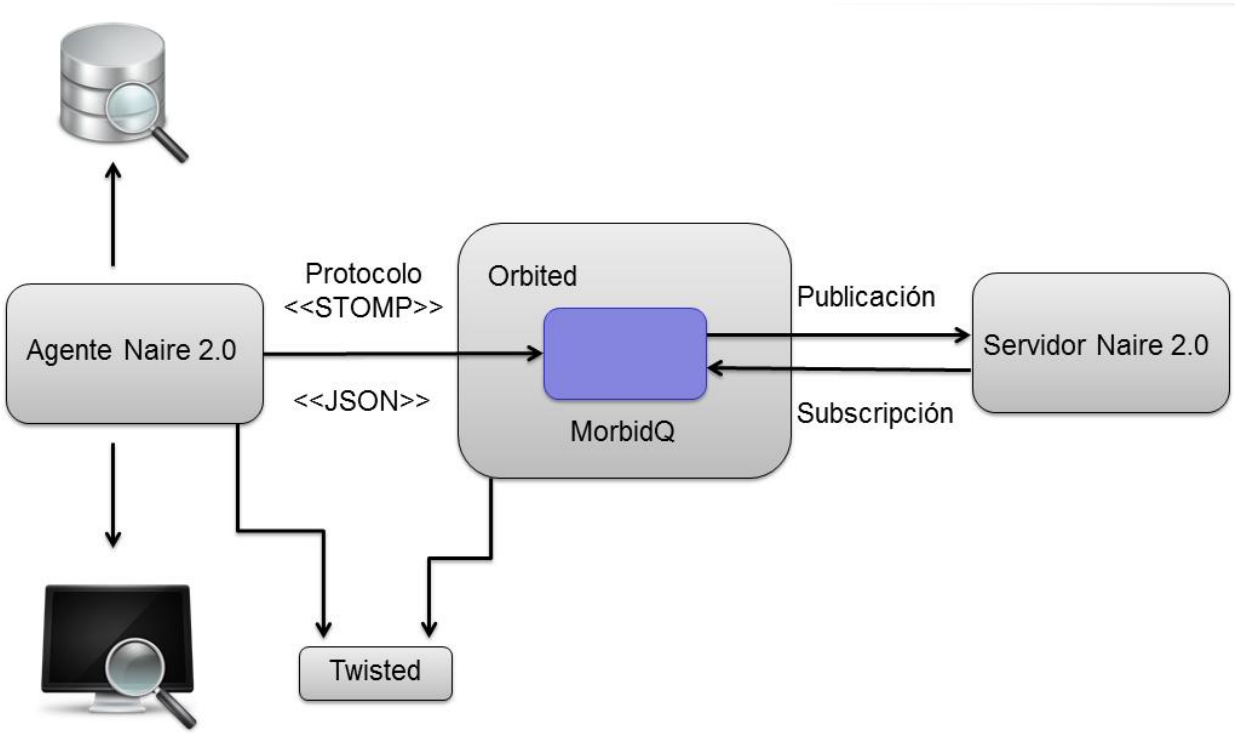


Figura 3: Arquitectura de comunicación en tiempo real para Naire.

2.4. Descripción del sistema propuesto.

El componente de captura de métricas propuesto a desarrollar en la versión 2.0 es el encargado de recolectar un grupo de métricas, clasificadas en generales y de bases de datos, que permitirá registrar el rendimiento en tiempo real de los servidores de PostgreSQL. El Administrador debe instalar el componente en cada uno de los servidores a monitorizar, una vez instalado, este último se comportará como un proceso del sistema operativo, que de acuerdo a su implementación estará enviando los datos en tiempo real hacia un servidor MorbidQ, al cual mediante canales y haciendo uso del patrón de mensajería Publicación-Subscripción desde el componente de reporte el administrador podrá subscribirse a los distintos canales obteniendo así la información de cada una de las métricas evaluadas en los servidores a monitorizar; para que la información pueda ser tratada en tiempo real, se debe definir un intervalo de tiempo por el cual debe someterse el componente, éste puede encontrarse desde 1 segundo hasta 5 segundos, a la vez que el componente envía los datos en tiempo real hacia el servidor MorbidQ, también es encargado de salvar estos datos en el servidor MongoDB, para esto se debe definir un intervalo de tiempo que puede estar entre 1 minuto y 60 minutos, con el objetivo de tener los datos almacenados por si se quiere ver en un futuro un histórico del estado de los servidores.

La estructura interna del agente o demonio está compuesto por diferentes paquetes, uno de ellos es el paquete encargado de capturar las métricas, donde se va a definir cada una de las métricas e indicadores que permiten analizar el comportamiento tanto del clúster PostgreSQL como del sistema operativo. El paquete encargado de enviar los datos en tiempo real es el responsable de recoger los resultados de cada una de las métricas y enviarlos en tiempo real haciendo uso del protocolo de comunicación STOMP. El paquete de los útiles es el que tiene la responsabilidad de contener las clases que se van a utilizar en las demás, así como la encargada de cargar la configuración para ejecutar el componente definida por el Administrador durante el proceso de instalación, también se encuentra la clase que sirve como base y además responsable de que el componente se ejecute como un demonio del sistema operativo. Se encuentra además la clase principal llamada Monitor la cual es responsable de hacer uso de la clase Reactor de Twisted para poder enviar los datos en tiempo real en el intervalo de tiempo definido por el administrador durante la instalación. El archivo dnaire.log es un registro al que se le van a agregar líneas a medida que se realizan las acciones por el componente de captura de métricas, las incidencias, errores de conexión, entradas de datos inválidos, cualquier problema con el sistema operativo, todo esto se va a registrar en este archivo. El archivo dnaire.cfg es

el encargado de guardar los parámetros de configuración necesarios para el correcto desempeño del componente.

En la BD NoSQL los datos son almacenados en documentos jerárquicos al estilo JSON y estos en colecciones de documentos. Un documento equivale al concepto de fila y una colección puede ser considerada como una tabla en bases de datos relacionales (sólo que las colecciones pueden almacenar documentos con diferentes formatos, en lugar de estar sometidos a un esquema fijo). Se pueden agrupar las colecciones en bases de datos, al igual que en una base de datos relacional se agrupan tablas.

2.5. Historias de usuario

La identificación de las historias de usuario (HU) es la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las HU es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas. A continuación se enumeran las HU y se muestran las dos más significativas.

- HU-1 Crear conexión al servidor MorbidQ.
- HU-2 Crear conexión al servidor MongoDB.
- HU-3 Crear colecciones JSON.
- HU-4 Enviar colecciones JSON en tiempo real a MorbidQ.
- HU-5 Salvar colecciones JSON en el servidor MongoDB.
- HU-6 Capturar estadísticas del servidor.

Tabla 1: Funcionalidades por historias de usuario.

Historias de Usuario(HU)	Funcionalidades
Crear conexión al servidor MorbidQ.	1. Crear conexión al servidor MorbidQ.
Crear conexión al servidor MongoDB.	1. Crear conexión al servidor MongoDB.

Crear colecciones JSON.	1. Crear colecciones JSON.
Enviar colecciones JSON en tiempo real a MorbidQ.	1. Enviar colecciones JSON en tiempo real a MorbidQ.
Salvar colecciones JSON en el servidor MongoDB.	1. Salvar colecciones JSON en el servidor MongoDB.
Capturar estadísticas del servidor.	1. Capturar modelo del disco duro. 2. Capturar ruta del dispositivo. 3. Capturar transferencias por segundo. 4. Capturar escrituras y lecturas de bloques en KB. 5. Capturar escrituras y lecturas de bloques por segundo en KB. 6. Capturar memoria activa versus inactiva. 7. Capturar buffer y cache. 8. Capturar uso de la memoria virtual. 9. Capturar modelo del CPU. 10. Capturar cantidad de CPU(s). 11. Capturar arquitectura del CPU. 12. Capturar tiempo de utilización de la CPU a nivel de usuario. 13. Capturar tiempo de utilización de la CPU a nivel de sistema. 14. Capturar tiempo en que la CPU se encuentra en estado ocioso. 15. Capturar tiempo que la CPU pasa sirviendo interrupciones de hardware. 16. Capturar tiempo que la CPU pasa sirviendo interrupciones de software. 17. Capturar tiempo para reiniciar procesos. 18. Capturar tiempo que la CPU pasa esperando para que ocurra una operación de Entrada/Salida. 19. Capturar interrupciones por segundo. 20. Capturar cambios de contexto. 21. Capturar utilización de la CPU. 22. Capturar bytes enviados y recibidos. 23. Capturar bytes enviados y recibidos por segundo. 24. Capturar cantidad de paquetes recibidos y enviados.

Tabla 2: HU “Enviar colecciones JSON en tiempo real a MorbidQ.”

Historia de Usuario	
Número: 4	**Nombre de la Historia de Usuario:** Enviar colecciones JSON en tiempo real a MorbidQ.
Cantidad de modificaciones a la Historia de Usuario: *Ninguna*	
Usuario: *Denys Retureta Mailero*	**Iteración asignada: 2**
Prioridad en negocio: Alta	**Puntos estimados: 3** semanas
Riesgo en desarrollo: Muy Alto	**Puntos reales:**.3 semanas
Descripción: Permite el envío de los datos en tiempo real al servidor MorbidQ, estos últimos serán enviado usando el formato JSON para el intercambio de datos.	
Observaciones:	
Prototipo de interfaces: No presenta prototipo de interfaz de usuario ya que la aplicación es un demonio del sistema operativo	

Tabla 3: HU “Salvar colecciones JSON en el servidor MongoDB.”

Historia de Usuario	
Número: 5	**Nombre de la Historia de Usuario:** Salvar colecciones JSON en el servidor MongoDB.
Cantidad de modificaciones a la Historia de Usuario: *Ninguna*	
Usuario: *Denys Retureta Mailero*	**Iteración asignada: 2**
Prioridad en negocio: Alta	**Puntos estimados: 2 semana**
Riesgo en desarrollo: Muy Alto	**Puntos reales:**.2 semana
Descripción: Permite la inserción de las colecciones JSON para cada servidor.	

Observaciones:
Prototipo de interfaces: No presenta prototipo de interfaz de usuario ya que la aplicación es un demonio del sistema operativo

2.6. Lista de reserva del producto

La lista de reserva del producto es una tabla que contiene los requisitos funcionales que debe cumplir la aplicación que se desea realizar, ordenados según la prioridad de implementación, clasificados en Muy Alta, Alta, Media y Baja, en esta última aparecen los requisitos de menor complejidad además de los requisitos no funcionales del sistema a desarrollar. Indica además la estimación de cada uno de los ellos y su implementación por semanas además del rol que hizo la estimación. A continuación se muestra la lista de reserva del producto con cada una de las funcionalidades clasificadas.

Tabla 4: Lista de reserva del producto

Ítem *	Descripción	Estimación	Estimado por
Prioridad: Muy Alta			
1	Crear conexión al servidor MorbidQ,	1 semana	*Programador*
2	Crear conexión al servidor MongoDB.	1 semana	*Programador*
Prioridad: Alta			
3	Crear colecciones JSON.	2 semanas	*Programador*
4	Enviar colecciones JSON en tiempo real a MorbidQ.	3 semanas	*Programador*
5	Insertar colecciones JSON en el servidor MongoDB.	2 semanas	*Programador*
Prioridad: Media			
6	Capturar estadísticas del servidor.	3 semanas	*Programador*
	Capturar estadísticas de los dispositivos de bloques (disco duro).		
6.1.	Capturar modelo del disco duro.		
6.2.	Capturar ruta del dispositivo.		
6.3.	Capturar transferencias por segundo.		
6.4.	Capturar escrituras y lecturas de bloques en KB.		
6.5.	Capturar escrituras y lecturas de bloques por segundo en KB.		
	Capturar estadísticas de la memoria.		
6.6.	Capturar memoria activa versus inactiva.		
6.7.	Capturar buffer y cache.		
	Capturar estadísticas de la memoria virtual		
6.8.	Capturar uso de la memoria virtual.		
	Capturar estadísticas del CPU.		
6.9.	Capturar modelo del CPU.		
6.10.	Capturar cantidad de CPU(s).		
6.11.	Capturar arquitectura del CPU.		
6.12.	Capturar tiempo de utilización de la CPU a nivel de usuario.		
6.13.	Capturar tiempo de utilización de la CPU a nivel de sistema.		

6.14.	Capturar tiempo en que la CPU se encuentra en estado ocioso.		
6.15.	Capturar tiempo que la CPU pasa sirviendo interrupciones de hardware.		
6.16.	Capturar tiempo que la CPU pasa sirviendo interrupciones de software.		
6.17.	Capturar tiempo para reiniciar procesos.		
6.18.	Capturar tiempo que la CPU pasa esperando para que ocurra una operación de Entrada/Salida.		
6.19.	Capturar interrupciones por segundo.		
6.20.	Capturar cambios de contexto.		
6.21.	Capturar utilización de la CPU.		
	Capturar estadísticas de la interfaz de red.		
6.22.	Capturar bytes enviados y recibidos.		
6.23.	Capturar bytes enviados y recibidos por segundo.		
6.24.	Capturar cantidad de paquetes recibidos y enviados.		
Prioridad: Baja **Requisitos no funcionales**			
1	**Hardware** • Para el cliente (componente de captura de métricas): o Microprocesador Pentium 4 o Superior. o 1 GB de RAM o superior. o 5 GB de espacio libre en disco. • Para el servidor: o Microprocesador Pentium 4 o Superior. o 1 GB de RAM o superior. o 500 GB de espacio libre en disco.		

2	**Software**
	• Para el cliente(componente de captura de métricas):
	  ○ Sistema gestor de bases de datos PostgreSQL.
	  ○ Se aconseja tener instalado como sistema operativo Ubuntu 11.04 o Debían 6.
	  ○ Se requiere tener instaladas las librerías python-pygresql, python-mongo, stompservice, python-twisted, python-demjson, python-stompy.
	• Para el servidor:
	  ○ Se requiere tener instaladas python-twisted, python-demjson, mongodb.		

2.7. Tareas de la ingeniería

Las tareas de la ingeniería se consideran como las entradas de trabajo para el equipo de programadores. Es la ficha que contiene el número identificador de la tarea, el identificador de la historia de usuario con la que está relacionada, el nombre de la tarea, la fecha de inicio, la fecha de fin, el equipo responsable y la descripción. A continuación se muestran dos ejemplos de algunas de las tareas de ingenierías asociadas a las historias de usuarios mencionadas anteriormente.

Tabla 5: Tarea de Ingeniería # 1: HU Crear conexión al servidor MorbidQ,

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 1
Nombre Tarea: Cargar configuración del servidor MorbidQ	

Tipo de Tarea : Desarrollo	**Puntos Estimados:** 0.5
Fecha Inicio: 5/12/11	**Fecha Fin: 7/12/11**
Programador Responsable: Denys Retureta Mailero	
Descripción: Se obtiene del fichero de configuración la dirección ip y el puerto del servidor MorbidQ.	

Tabla 6: Tarea de Ingeniería # 2: HU Crear conexión al servidor MorbidQ,

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** 1
Nombre Tarea: Crear conexión al servidor MorbidQ	
Tipo de Tarea : Desarrollo	**Puntos Estimados: 0.5**
Fecha Inicio: 7/12/11	**Fecha Fin: 9/12/11**
Programador Responsable: Denys Retureta Mailero	
Descripción: Se crea la conexión al servidor MorbidQ.	

2.8. Plan de iteraciones

Después de tener ya definidas las historias de usuario es necesario crear un plan de iteraciones para poder determinar las iteraciones en que serán implementadas cada una de ellas. También se determina en este plan la prioridad con la que serán implementadas las historias de usuario. La siguiente tabla muestra la cantidad de iteraciones establecidas para la implementación del componente.

Tabla 7: Plan de Iteraciones.

Iteración	Descripción de la iteración	Orden de la HU a implementar	Duración total
1	Esta iteración tiene como objetivo darle cumplimiento a las historias de usuarios 1 y 2 que serán las de vital importancia para el componente a desarrollar, pues son las que permitirán la conexión con el servidor MorbidQ encargado de la comunicación en tiempo real y el servidor MongoDB en el cual se van a salvar la información de las métricas.	1, 2	2 semanas
2	El objetivo de esta iteración es implementar las historias de usuarios 3, 4 y 5, que son de una alta prioridad para el componente a desarrollar, como organizar la información en colecciones JSON para un correcto envío de la misma en tiempo real.	3, 4, 5	7 semanas
3	Esta iteración está centrada en desarrollar una parte de las historias de usuario con prioridad media para el componente a desarrollar. Está conformada por las historias de usuario 6, referente a las nuevas métricas detectadas las cuales son de tipo General: Estadísticas de CPU, Estadísticas de interfaz de red, Estadísticas de disco duro, Estadísticas de memoria y Estadísticas de memoria virtual, organizadas en la Historia de Usuario Capturar estadísticas del servidor.	6	3 semanas

2.9. Modelo de Diseño

En XP la idea es que el diseño final sea la forma más simple de agrupar todos los requisitos. Un diseño complejo puede influir negativamente en el avance del desarrollo del producto de software debido a que los requisitos pueden cambiar constantemente. Hay que diseñar lo que las necesidades del problema requieren, no lo que se cree que deberá ser el diseño.

2.9.1. Diagrama de Clases

A continuación se presenta el diagrama de clases asociado al componente en desarrollo para así tener un mejor entendimiento del sistema.

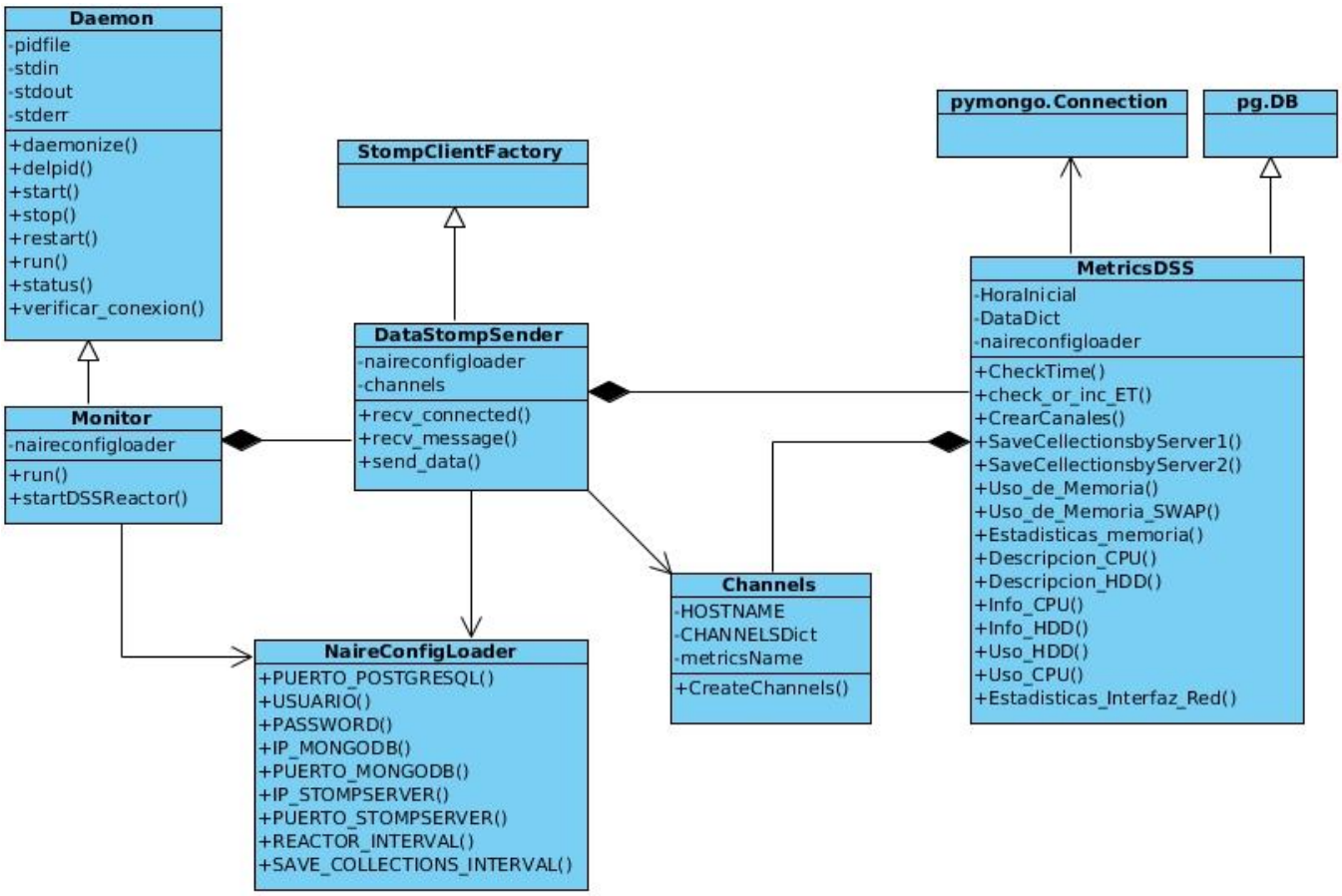


Figura 4: Diagrama de clases.

2.9.2. Tarjetas CRC

Las tarjetas CRC (Contenido, Responsabilidad y Colaboración), son la expresión más específica de las clases que se evidencian en la metodología XP, las mismas contienen las responsabilidades que cada clase debe cumplir ayudando a dar un mejor entendimiento del diseño del sistema, así como las colaboraciones que hace con otras clases. Estas tarjetas deben de estar bien enfocadas para de esta forma contribuir a la comunicación y el buen entendimiento del funcionamiento entre las clases conceptuales de este trabajo. No deben ser muy largas y contener la información necesaria para lograr los resultados esperados. A continuación se muestran tres ejemplos de las tarjetas CRC diseñadas en el sistema.

Tabla 8: Tarjeta CRC correspondiente a la clase MorbidQ.

Tarjeta CRC	
Clase: MorbidQ	
Responsabilidades	**Colaboraciones**
• Crear conexión al servidor MorbidQ.	

Tabla 9: Tarjeta CRC correspondiente a la clase MongoDB.

Tarjeta CRC	
Clase: MongoDB	
Responsabilidades	**Colaboraciones**
• Crear conexión al servidor MongoDB.	

Tabla 10: Tarjeta CRC correspondiente a la clase Colecciones.

Tarjeta CRC	
Clase: Colecciones	
Responsabilidades	**Colaboraciones**

• Crear colecciones JSON a partir de los datos de las métricas capturadas.	

2.10. Patrón de arquitectura

Los patrones de arquitectura ayudan a entender el funcionamiento del sistema, para de esta forma lograr su implementación lo más organizada posible, son los encargados de capturar los elementos esenciales de una arquitectura de software**.** Los patrones arquitectónicos brindan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre cómo pueden ser usados. (39)

En este trabajo se definió hacer uso del patrón arquitectura dirigida por eventos, el cual es un estilo de arquitectura que se caracteriza por la existencia de un número de los actores relativamente independientes que se comunican eventos entre ellos con el fin de lograr un objetivo coordinado. La arquitectura dirigida por eventos cuenta con la tecnología llamada Publicación-Subscripción que permite que la información sea distribuida en tiempo real. Este patrón de arquitectura se basa en el patrón de Publicación–Subscripción asincrónico. La Figura 5 muestra en qué consiste el patrón de arquitectura dirigida por eventos.

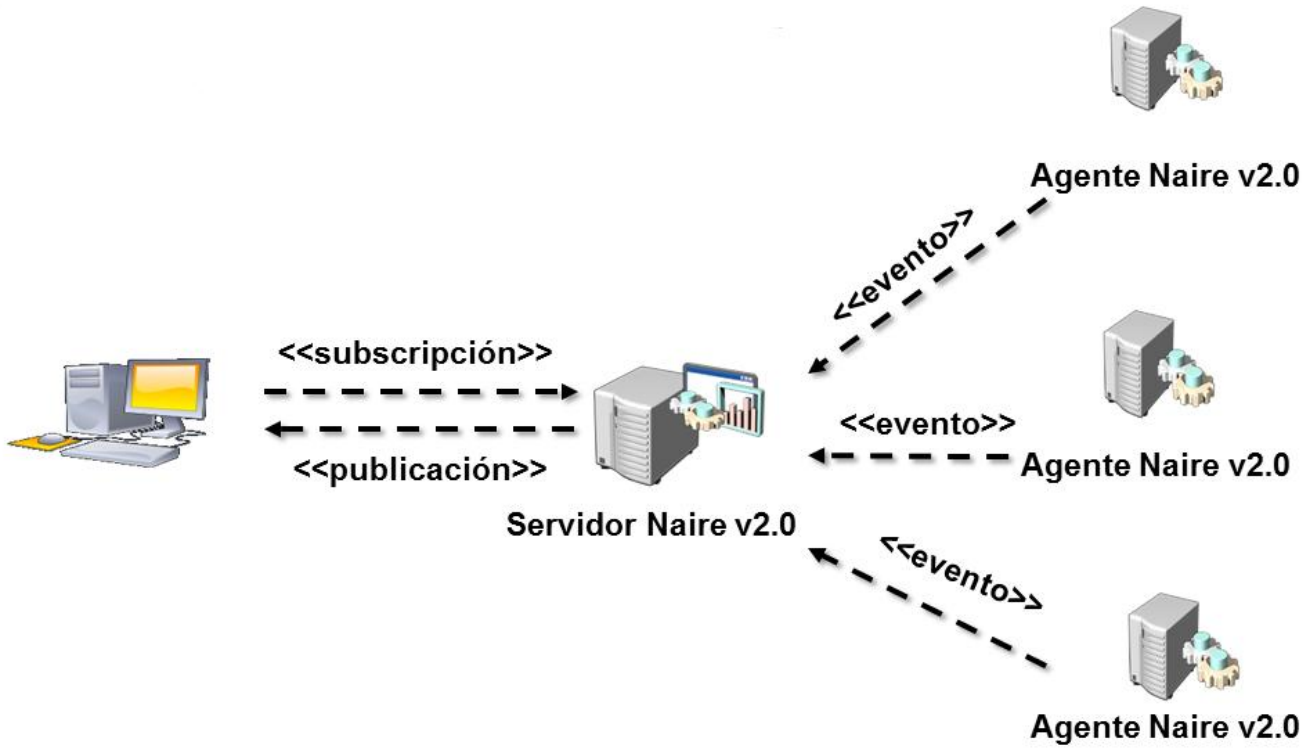


Figura 5: Representación del Patrón de arquitectura dirigida por eventos aplicado en el funcionamiento de Naire.

El uso de este patrón en la implementación del componente de captura de métricas está evidenciado en el uso de Twisted[11] para la implementación del mismo, además de que tanto los servidores de Orbited y MorbidQ utilizados para lograr el funcionamiento de la comunicación en tiempo real de los componentes involucrados están implementados usando este framework.

2.11. Patrones de diseño.

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y, para lograr una mayor calidad en el diseño. A continuación se definen los patrones empleados en este trabajo de diploma.

Creador.

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, el propósito fundamental de este patrón es encontrar un creador que conecte con el objeto producido en cualquier evento. Le asigna a una clase determinada la responsabilidad de crear una instancia de otro

[11] Framework de red para programación dirigida por eventos escrito en Python y licenciado bajo la licencia MIT

tipo de clase (40). MetricsDSS tiene datos de inicialización que se pasan a un objeto de Canales cuando sea creado (por tanto, MetricasDSS es un Experto con respecto a la creación de Canales). Ver Figura 6 para ver un ejemplo de la aplicación.

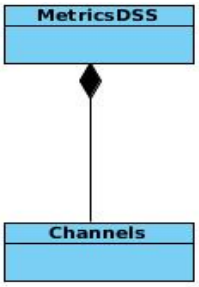


Figura 6: Ejemplo de la aplicación del patrón creador en el sistema.

Experto.

La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada (40). La clase DataStompSender es experta enviando la información de las métricas. En la figura 7 se muestra un ejemplo del uso del mismo.

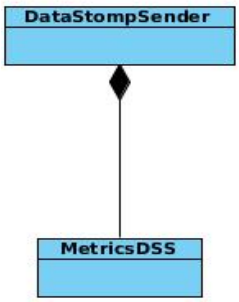


Figura 7: Ejemplo de la aplicación del patrón experto en el sistema.

Bajo acoplamiento.

El acoplamiento coincidido es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Acoplamiento bajo significa que una clase no depende de muchas clases. Acoplamiento alto significa que una clase recurre a muchas otras clases (40). A continuación en la Figura 8 se muestra cómo la clase MetricsDSS depende solamente de pg.DB y pymongo.Connection, por lo que no existe una dependencia entre las clases.

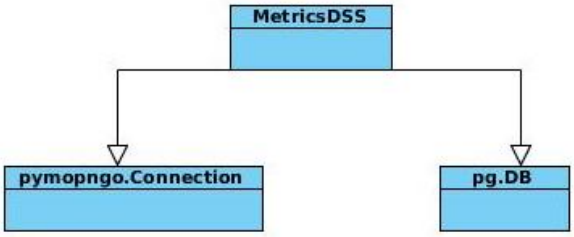


Figura 8: Ejemplo de la aplicación del patrón bajo acoplamiento en el sistema.

Alta cohesión.

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una baja cohesión hace muchas cosas no afines o realiza trabajo excesivo (40). A continuación en la Figura 9 se muestra un ejemplo del uso de este patrón.

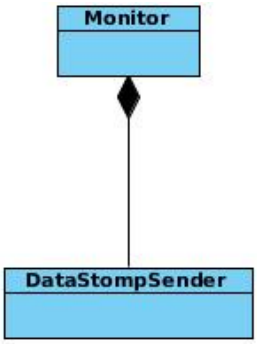


Figura 9: Ejemplo de la aplicación del patrón Alta cohesión en el sistema.

2.12. Estándares de codificación

El estilo de programación también llamado estándares de código o convención de código es un término que describe convenciones para escribir código fuente en ciertos lenguajes de programación.

Llevado a este caso específico el estándar de codificación utilizado para la implementación del componente de captura de métricas en su versión 2.0 es la Propuesta de Mejora para Python número 8 o PEP-8 (41) por sus siglas en inglés, el cual es el nombre clave de la Guía de estilo para código Python publicada en julio de 2001.

Conclusiones del capítulo.

En este capítulo se definieron las características esenciales del componente en desarrollo. Se identificaron los requerimientos del sistema y se realizó una descripción de las historias de usuario, como paso fundamental en el ciclo de desarrollo. Para lograr un mejor entendimiento del componente a desarrollar se definió un modelo de dominio. Como parte del diseño se elaboraron las tarjetas CRC. Se identificaron además los patrones de diseños presentes, el estándar de codificación a utilizar y el patrón arquitectónico utilizado en el desarrollo de la versión 2.0 del componente de captura de métricas.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA DE LA SOLUCIÓN

INTRODUCCIÓN

En el presente capítulo se realiza la validación del componente propuesto en el capítulo anterior. Se selecciona la estrategia de prueba a realizar, además se selecciona el método de validación de las pruebas definidas por la metodología XP, que permite verificar el correcto funcionamiento del componente. Se muestran además los resultados de estas últimas en cada una de las iteraciones del proceso de desarrollo de software.

3.1. Estrategias de Pruebas.

Pruebas Unitarias

Las pruebas unitarias son una de las piedras angulares de la Programación Extrema (XP). Sin embargo, el estilo de las pruebas unitarias de XP es un poco diferente. En primer lugar se debe crear o descargar un framework de casos de pruebas para ser capaz de crear suites de pruebas unitarias automatizadas. En segundo lugar se debe probar todas las clases en el sistema. Los métodos triviales getter y setter generalmente se omiten.

La pruebas unitarias proporcionan una red de seguridad para las pruebas de regresión y pruebas de validación, para que se pueda refactorizar e integrar con eficiencia. La creación de las pruebas unitarias antes de la codificación ayuda a solidificar aún más los requisitos y mejorar el enfoque de desarrollo. (42)

Las pruebas unitarias son la primera fase de las pruebas dinámicas y se realizan sobre cada módulo del software de manera independiente. El objetivo es comprobar que el módulo, entendido como una unidad funcional de un programa independiente, está correctamente codificado. En estas pruebas cada módulo será probado por separado y lo hará, generalmente, la persona que lo creó.

Pruebas de Aceptación

Las pruebas de aceptación son creadas a partir de las historias de usuarios. Durante una iteración las historias de usuarios seleccionadas durante la reunión de planificación del Plan de Iteraciones serán traducidas a pruebas de aceptación. El cliente especifica los escenarios para poner a prueba cuando una HU se ha aplicado correctamente. Una historia de usuario puede tener pruebas de aceptación de uno o muchos, lo que sea necesario para garantizar que la funcionalidad se ejecuta satisfactoriamente. (43)

Cuando el producto está listo para ser implantado en el entorno del cliente se deben aplicar estas pruebas, siendo el cliente quien las realice ayudado por personas del equipo de pruebas. Estas pruebas se caracterizan por la participación activa del usuario, que debe ejecutar los casos de prueba ayudado por miembros del equipo de pruebas, están enfocadas a probar las funcionalidades de la aplicación, o mejor dicho a demostrar que no se cumplen los requisitos y criterios de aceptación o el contrato. Si no se consigue demostrar esto el cliente deberá aceptar el producto.

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.

3.2. Método de prueba seleccionada.

También conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. El componente se ve como una “Caja Negra” cuyo comportamiento sólo puede ser determinado estudiando sus entradas y las salidas obtenidas a partir de ellas. No obstante, como el estudio de todas las posibles entradas y salidas de un programa sería impracticable se selecciona un conjunto de ellas sobre las que se realizan las pruebas. (44)

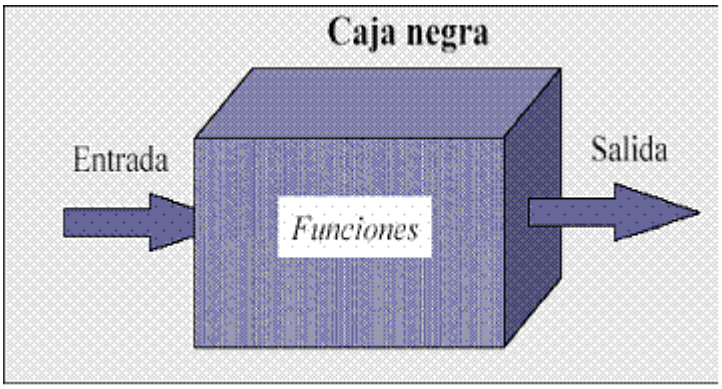


Figura 10 Representación del flujo de las pruebas de caja negra.

En este trabajo se decide utilizar el método de caja negra con la técnica de partición de equivalencia debido a que es el encargado de probar las funcionalidades del sistema. Este método pretende demostrar que las funciones del software son operativas, comprobando que la entrada de datos sea

correcta. Las pruebas de caja negra marcan el camino a seguir indicándole al equipo de desarrollo las funcionalidades más importantes. Además permiten que el cliente sepa cuando el sistema está funcionando y a la vez permiten que los programadores conozcan que es lo que les resta por hacer.

3.3. Casos de pruebas basados en HU.

Los casos de pruebas no son más que un conjunto de condiciones o variables bajo las cuáles el analista determinará si el requisito de una aplicación es parcial o completamente satisfactorio. A continuación se muestran dos ejemplos de casos de pruebas aplicados al sistema.

Tabla 11: Caso de prueba 1: HU 1: Crear conexión al servidor MorbidQ,

Escenario	Descripción	Variable 1	Variable 2	Respuesta del sistema	Flujo central
EC 1.1 Crear conexión al servidor MorbidQ con los datos correctos,	El componente crea exitosamente la conexión con el servidor MorbidQ,	V	V	El componente se conecta al servidor MorbidQ y se encuentra listo para enviar los datos en tiempo real,	Al iniciar el componente, este carga del fichero de configuración los parámetros del servidor MorbidQ,
EC 1.2 Crear conexión al servidor MorbidQ con los datos incorrectos,	El componente no crea la conexión con el servidor MorbidQ,	V	I	El componente muestra un mensaje de error en la consola,	Ir al fichero de configuración y cambiar los parámetros del servidor MorbidQ,
		I	V		
		I	I		

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
Variable 1	IP_MORBIDQ	Campo de texto	No	Dirección IP del servidor MorbidQ
Variable 2	PUERTO_MORBIDQ	Campo de texto	No	Puerto para la conexión con el servidor MorbidQ

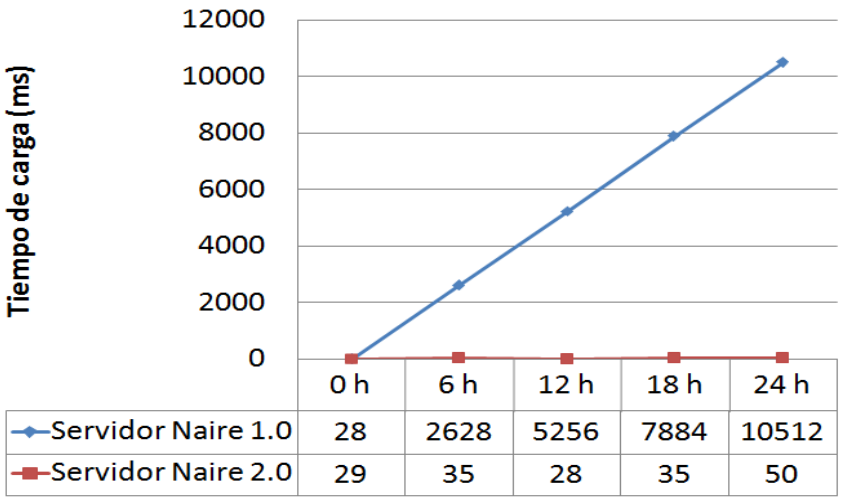
Tabla 12: Caso de prueba 2: HU 2 Crear conexión al servidor MongoDB.

Escenario	Descripción	Variable 1	Variable 2	Respuesta del sistema	Flujo central

EC 1.1 Crear conexión al servidor MongoDB con los datos correctos,	El componente crea exitosamente la conexión con el servidor MongoDB	V	V	El componente se conecta al servidor MongoDB y se encuentra listo para salvar los datos en el intervalo definido,	Al iniciar el componente, este carga del fichero de configuración los parámetros del servidor MongoDB,
EC 1.2 Crear conexión al servidor MongoDB con los datos incorrectos,	El componente no crea la conexión con el servidor MongoDB	V	I	El componente muestra un mensaje de error en la consola,	Ir al fichero de configuración y cambiar los parámetros del servidor MongoDB,
		I	V		
		I	I		

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
Variable 1	IP_MONGODB	Campo de texto	No	Dirección IP del servidor MongoDB
Variable 2	PUERTO_MONGODB	Campo de texto	No	Puerto para la conexión con el servidor MongoDB

Para validar el correcto funcionamiento del sistema de comunicación en tiempo real entre el componente de captura de métricas y el de reportes, en el intervalo de tiempo definido, se muestra la Figuras 11, la cual representa los valores de carga de la sección de reportes de cada uno de los Servidores Naire versiones 1.0 y 2.0, los cuales fueron obtenidos haciendo uso del software Apache JMeter. Además se muestran las figuras 12 y 13, las cuales son capturas de este último software, a las 12 horas de haber empezado la monitorización de ambos servidores como se muestra en las capturas el tiempo de carga el cual esta expresado en milisegundos(ms) es mucho mayor para la versión 1.0 y a medida que pase el tiempo y aumente la cantidad de información almacenada en MongoDB, aumentara el tiempo de carga de los reportes, mientras que para la versión 2.0 es capaz de mantener un tiempo de carga medio de 31 ms en el mismo intervalo.



	0 h	6 h	12 h	18 h	24 h
Servidor Naire 1.0	28	2628	5256	7884	10512
Servidor Naire 2.0	29	35	28	35	50

Figura 11: Tiempos de carga en milisegundos para 24 horas de monitorización continua.

Figura 12: Captura del Servidor Naire versión 1.0 a las 12 horas.

Figura 13: Captura del Servidor Naire versión 2.0 a las 12 horas.

3.4. Resultado de las pruebas.

Con el propósito de comprobar que todos los requisitos del sistema fueron desarrollados y cumplidos completamente, se aplicaron al componente pruebas funcionales, donde se creó un caso de prueba por cada Historia de Usuario. Se realizaron 2 iteraciones de pruebas por cada iteración del desarrollo. En la primera iteración del desarrollo se encontraron 2 no conformidades dándole respuesta en su totalidad. Para una segunda iteración del desarrollo se encontraron 3 no conformidades, las cuales fueron solucionadas completamente. En una tercera iteración del desarrollo se encontraron 3 no conformidades dándole respuesta a las tres. A continuación se presenta la Figura 14, 15 y 16 con los resultados en forma de gráfica para un mejor entendimiento.

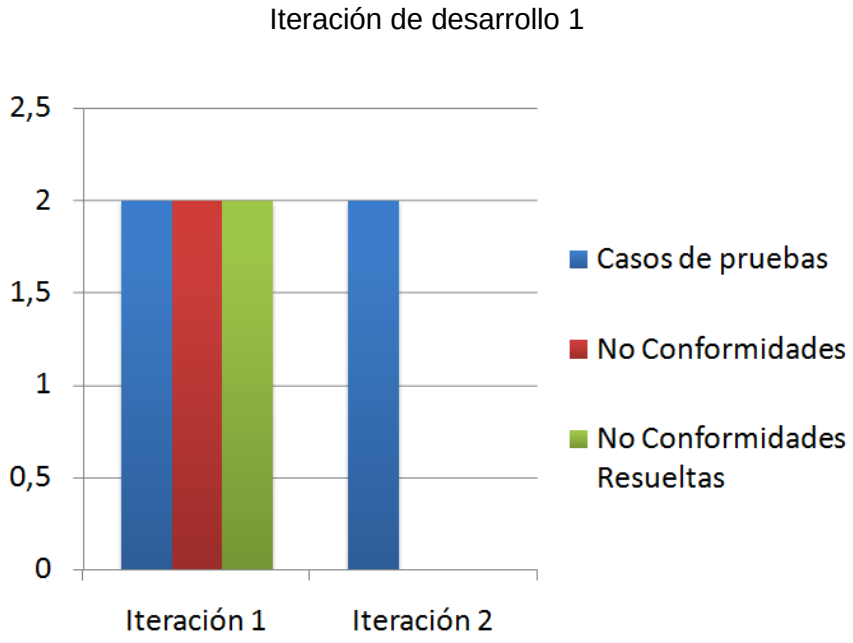


Figura 14: Resultados de las pruebas funcionales para la primera iteración del desarrollo.

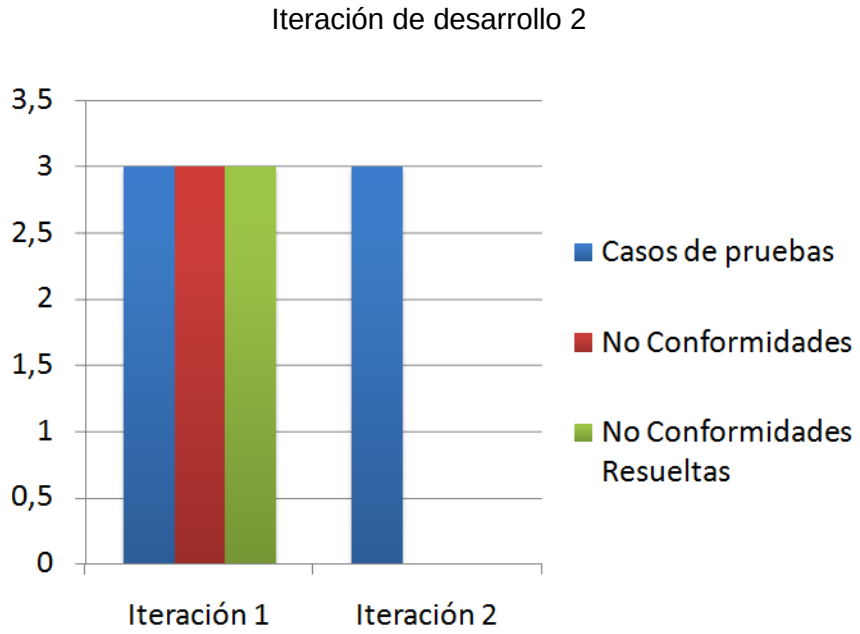


Figura 15: Resultados de las pruebas funcionales para la segunda iteración del desarrollo.

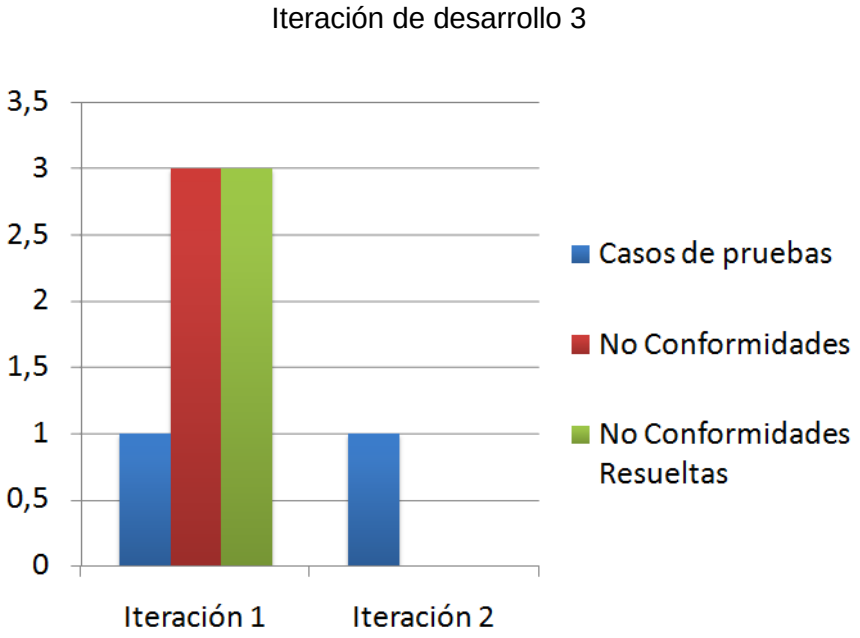


Figura 16: Resultados de las pruebas funcionales para la tercera iteración del desarrollo.

A continuación se muestra la Tabla 11 en la cual se presentan las no conformidades detectadas en la primera iteración para los casos de pruebas presentados en el epígrafe anterior: Crear conexión al servidor MorbidQ y Crear conexión al servidor MorbidQ.

Tabla 13: No conformidades para los casos de pruebas 1 y 2.

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Clasificación	Estado NC	Respuesta del Equipo Desarrollo
Aplicación	1	El componente no muestra una alerta al ocurrir un intento de conexión fallida al servidor MongoDB.	Conexión a MongoDB.	Prueba	S	PD: 9/05/12	
						RA: 10/05/12	Se agrega una función para verificar la disponibilidad del servidor MongoDB, en caso de no estarlo, esta muestra un mensaje en la consola al tratar de iniciar el componente.
Aplicación	2	El componente no muestra una alerta al ocurrir un intento de conexión fallida al servidor MorbidQ.	Conexión a MorbidQ.	Prueba	S	PD: 9/05/12	
						RA: 10/05/12	Se agrega una función para verificar la disponibilidad del servidor MorbidQ, en caso de no estarlo, esta muestra un mensaje en la consola al tratar de iniciar el componente.

Conclusiones del capítulo.

Con el fin de lograr que las funcionalidades del sistema se comporten según lo esperado se aplicaron las pruebas funcionales capaces de probar el componente completamente, se aplicaron un total de 6 casos de pruebas, uno por cada HU, los cuales arrojaron 8 no conformidades, 2 en la primera iteración, 3 en la segunda y 3 en la tercera, las cuales fueron solucionadas satisfactoriamente para así lograr un mejor funcionamiento del sistema. Además con el objetivo de validar el cumplimiento del objetivo del presente trabajo se realizaron pruebas haciendo uso del software Apache JMeter con el cual se comprobó que existe una disminución considerable del tiempo de respuesta en el proceso de monitorización de Naire.

CONCLUSIONES GENERALES

Con el desarrollo de la versión 2.0 del componente de captura de métricas, se cumplió con los objetivos trazados en la investigación, obteniendo como resultado un componente capaz de disminuir el tiempo de respuesta en el proceso de monitorización de Naire.

- El estudio de las herramientas de monitorización en tiempo real existentes en el mundo arrojó gran cantidad de información relevante para el desarrollo del componente tales como: protocolos de comunicación, tecnologías y patrones de mensajería.
- El estudio realizado permitió diseñar un mecanismo de comunicación en tiempo real entre el componente de captura de métricas y el componente de reportes capaz de disminuir el tiempo de respuesta en el proceso de monitorización de Naire.
- La realización de las pruebas a partir de la estrategia de pruebas seleccionada, además del uso del software Apache JMeter permitió verificar el correcto funcionamiento del componente, logrando así el cumplimiento del objetivo del presente trabajo de diploma.

RECOMENDACIONES

Con el objetivo de lograr mejoras en el funcionamiento del componente de captura de métricas se recomienda:.

- Continuar con el proceso de desarrollo del componente del tal forma que permita la monitorización de más de un clúster de datos en el mismo servidor.
- Migrar el uso de la librería python-pygresql a python-psycopg2 siendo esta última a la que se le está dando un mantenimiento activo por parte de los desarrolladores.

REFERNCIAS BIBLIOGRÁFICAS

1. **Gonzáles Cruz, Maité y Mouriz Coca, Yandira.** Cuba, las TICs y el Bloqueo. [En línea] [Citado el: 3 de Diciembre de 2011.] http://www.revistaciencias.com/publicaciones/EEZVpElyyplXUCmqSb.php.
2. **García, Rosa María Mato.** [aut. libro] Rosa María Mato García. *Diseño de bases de datos.* 1999.
3. **Korth, Henry y Silberschatz, Abraham.** *Fundamentos de las bases de datos. Cuarta edición.* 2002.
4. **Group, PostgreSQL Global Development.** PostgreSQL. [En línea] [Citado el: 19 de Octubre de 2011.] http://www.postgresql.org/.
5. **Trabajo, Organización Internacional del.** CINTENFOR. [En línea] [Citado el: 5 de Noviembre de 2011.] http://www.cinterfor.org.uy/public/spanish/region/ampro/cinterfor/temas/gender/em_ca_eq/m_eva.htm.
6. **Valle, Otto.** Monitoreo e Indicadores. *Texto de apoyo al proceso de construcción del Sistema Regional de Indicadores sobre Atención y Educación Inicial.* [En línea] 2010. http://www.oei.es/idie/mONITOREOEINDICADORES.pdf.
7. **A, Burns A. and Wellings.** *Real-time systems and programming languages.* New York : Addison Wesley, 1996.
8. **Foundation, XMPP Standards.** XMPP Standards Foundation. [En línea] [Citado el: 2011 de Octubre de 2011.] http://xmpp.org/about-xmpp/technology-overview/.
9. **STOMP.** STOMP. [En línea] [Citado el: 23 de Octubre de 2011.] http://stomp.codehaus.org/.
10. **help, Internet Relay Chat (IRC).** Internet Relay Chat (IRC). [En línea] [Citado el: 23 de Octubre de 2011.] http://www.irchelp.org/irchelp/rfc/chapter1.html.
11. **Foundatio, XMPP Standards.** XEP-0060: Publish-Subscribe. [En línea] [Citado el: 2 de Diciembre de 2011.] http://xmpp.org/extensions/xep-0060.html.
12. **Hohpe, Gregor.** *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* 2003.
13. **Russell, Alex.** Infrequently Noted. [En línea] [Citado el: 3 de Diciembre de 2011.] http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/.
14. **Foundation, XMPP Standards.** XMPP Technologies: BOSH. [En línea] [Citado el: 4 de Diciembre de 2011.] http://xmpp.org/about-xmpp/technology-overview/bosh/.
15. **Peter Lubbers, Brian Albers,and Frank Salim.** *Pro HTML5 Programming.* 2010.
16. **Sharp, Bruce Lawson and Remy.** *Introducing HTML5.* 2011.
17. **Orbited.** Networking for the Web. [En línea] [Citado el: 4 de Octubre de 2011.] http://labs.gameclosure.com/orbited2.
18. **Orbited**. Orbited documentation. [En línea] [Citado el: 5 de Diciembre de 2011.] http://orbited2.org/.
19. **Center, Middleware Resource.** Middleware Resource Center. [En línea] [Citado el: 5 de Diciembre de 2011.] http://www.middleware.org/general/mqm.html.
20. **Nagios.** Nagios Enterprises. [En línea] [Citado el: 7 de Diciembre de 2011.] http://www.nagios.com/.
21. **Corp, ZOHO.** Manage Engine. [En línea] [Citado el: 17 de Diciembre de 2011.] http://www.manageengine.com/products/applications_manager/postgresql-monitoring.html.
22. **EnterpriseDB.** EnterpriseDB. [En línea] [Citado el: 7 de Diciembre de 2011.] http://www.enterprisedb.com/products-services-training/products/postgres-enterprise-manager.
23. **SourceForge.** Hyperic-HQ. [En línea] [Citado el: 7 de Diciembre de 2011.] http://sourceforge.net/projects/hyperic-hq/.
24. **Hyperic, Spring Source.** Spring Source Hyperic. [En línea] [Citado el: 8 de Diciembre de 2011.] http://www.hyperic.com/products/applications-monitoring.
25. **Vmware.** Spring Source. [En línea] [Citado el: 9 de Diciembre de 2011.] http://www.springsource.com/license/hyperic-enterprise.

26. **SIU, Consorcio.** *Monitoreando y midiendo el rendimiento de un servidor Postgresql en sistemas Windows, Linux y Solaris, pág. 10. Parte 1.*
27. **Programming, Extreme.** Extreme Programming. [En línea] [Citado el: 10 de Diciembre de 2011.] http://www.extremeprogramming.org/.
28. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico.* s.l. : Concepción Femández Madrid. 5ta., 1998. ISBN: 84-481-3214-9.
29. **Fenton, Norman E. y Lawrence, Pfleeger Shari.** *Software Metrics. A rigorus and Practical Approach.* 1998. ISBN: 0534954251.
30. **IEEE.** IEEE. [En línea] [Citado el: 13 de Diciembre de 2011.]
31. **Olsina, Luis.** *Métricas e Indicadores.* 2003.
32. **Ubuntu, Comunidad de.** doc.ubuntu-es. [En línea] [Citado el: 8 de Enero de 2012.] http://doc.ubuntu-es.org/Sobre_Ubuntu.
33. **Junio Hamano, Linus Torvalds.** Control de versiones con Git. [En línea] [Citado el: 10 de Enero de 2012.] http://git-scm.com/.
34. **Lang, Jean Philippe.** Redmine. [En línea] [Citado el: 10 de Enero de 2012.] http://www.redmine.org/.
35. **Paradigm, Visual.** Visual Paradigm. [En línea] [Citado el: 12 de Diciembre de 2012.] http://www.visual-paradigm.com/.
36. **Varó, Marzal y Gracia Luengo, Isabel.** *Introducción a la Programación con Python.* 2003.
37. **Chodorow, Kristina y Dirolf, Michael.** *MongoDB: The Definitive Guide.* 2010. ISBN: 978-1-449-38156-1.
38. **Geany.** Geany. [En línea] [Citado el: 12 de Enero de 2012.] http://www.geany.org.
39. **Bass L., Clements P., Kazman R.** *Software Architecture in Practice: Second Edition.* s.l. : Addison-Wesley, 2005.
40. **Marcello Visconti; Hernán Astudillo.** Fundamentos de Ingeniería de Software. *Fundamentos de Ingeniería de Software.* [En línea] 2010. [Citado el: 14 de Enero de 2012.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.
41. **Python.** PEP 8 - Style Guide for Python Code. [En línea] [Citado el: 14 de Enero de 2012.] http://www.python.org/dev/peps/pep-0008/.
42. **Programming, Extreme.** Unit Test. [En línea] [Citado el: 15 de Enero de 2012.] http://www.extremeprogramming.org/rules/unittests.html.
43. **Programming, Extreme**. Acceptance Test. [En línea] [Citado el: 15 de Enero de 2012.] http://www.extremeprogramming.org/rules/functionaltests.html..
44. **Natalia Juristo, Ana M. Moreno, Sira Vegas.** *Tecnicas de evaluación de software.* 2006.

BIBLIOGRAFÍA

1. **Gonzáles Cruz, Maité y Mouriz Coca, Yandira.** Cuba, las TICs y el Bloqueo. [En línea] [Citado el: 3 de Diciembre de 2011.] http://www.revistaciencias.com/publicaciones/EEZVpElyyplXUCmqSb.php.
2. **García, Rosa María Mato.** [aut. libro] Rosa María Mato García. *Diseño de bases de datos.* 1999.
3. **Korth, Henry y Silberschatz, Abraham.** *Fundamentos de las bases de datos. Cuarta edición.* 2002.
4. **Group, PostgreSQL Global Development.** PostgreSQL. [En línea] [Citado el: 19 de Octubre de 2011.] http://www.postgresql.org/.
5. **Trabajo, Organización Internacional del.** CINTENFOR. [En línea] [Citado el: 5 de Noviembre de 2011.] http://www.cinterfor.org.uy/public/spanish/region/ampro/cinterfor/temas/gender/em_ca_eq/m_eva.htm.
6. **Valle, Otto.** Monitoreo e Indicadores. *Texto de apoyo al proceso de construcción del Sistema Regional de Indicadores sobre Atención y Educación Inicial.* [En línea] 2010. http://www.oei.es/idie/mONITOREOEINDICADORES.pdf.
7. **A, Burns A. and Wellings.** *Real-time systems and programming languages.* New York : Addison Wesley, 1996.
8. **Foundation, XMPP Standards.** XMPP Standards Foundation. [En línea] [Citado el: 2011 de Octubre de 2011.] http://xmpp.org/about-xmpp/technology-overview/.
9. **STOMP.** STOMP. [En línea] [Citado el: 23 de Octubre de 2011.] http://stomp.codehaus.org/.
10. **Internet Relay Chat (IRC) help.** Internet Relay Chat (IRC). [En línea] [Citado el: 23 de Octubre de 2011.] http://www.irchelp.org/irchelp/rfc/chapter1.html.
11. **Foundatio, XMPP Standards.** XEP-0060: Publish-Subscribe. [En línea] [Citado el: 2 de Diciembre de 2011.] http://xmpp.org/extensions/xep-0060.html.
12. **Hohpe, Gregor.** *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* 2003.
13. **Russell, Alex.** Infrequently Noted. [En línea] [Citado el: 3 de Diciembre de 2011.] http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/.
14. **Foundation, XMPP Standards.** XMPP Technologies: BOSH. [En línea] [Citado el: 4 de Diciembre de 2011.] http://xmpp.org/about-xmpp/technology-overview/bosh/.
15. **Peter Lubbers, Brian Albers,and Frank Salim.** *Pro HTML5 Programming.* 2010.
16. **Sharp, Bruce Lawson and Remy.** *Introducing HTML5.* 2011.
17. **Orbited.** Networking for the Web. [En línea] [Citado el: 4 de Octubre de 2011.] http://labs.gameclosure.com/orbited2.
18. **Orbited**. Orbited documentation. [En línea] [Citado el: 5 de Diciembre de 2011.] http://orbited2.org/.
19. **Center, Middleware Resource.** Middleware Resource Center. [En línea] [Citado el: 5 de Diciembre de 2011.] http://www.middleware.org/general/mqm.html.
20. **Nagios.** Nagios Enterprises. [En línea] [Citado el: 7 de Diciembre de 2011.] http://www.nagios.com/.
21. **Corp, ZOHO.** Manage Engine. [En línea] [Citado el: 17 de Diciembre de 2011.] http://www.manageengine.com/products/applications_manager/postgresql-monitoring.html.
22. **EnterpriseDB.** EnterpriseDB. [En línea] [Citado el: 7 de Diciembre de 2011.] http://www.enterprisedb.com/products-services-training/products/postgres-enterprise-manager.
23. **SourceForge.** Hyperic-HQ. [En línea] [Citado el: 7 de Diciembre de 2011.] http://sourceforge.net/projects/hyperic-hq/.
24. **Hyperic, Spring Source.** Spring Source Hyperic. [En línea] [Citado el: 8 de Diciembre de 2011.] http://www.hyperic.com/products/applications-monitoring.
25. **Vmware.** Spring Source. [En línea] [Citado el: 9 de Diciembre de 2011.] http://www.springsource.com/license/hyperic-enterprise.

26. **SIU, Consorcio.** *Monitoreando y midiendo el rendimiento de un servidor Postgresql en sistemas Windows, Linux y Solaris, pág. 10. Parte 1.*
27. **Programming, Extreme.** Extreme Programming. [En línea] [Citado el: 10 de Diciembre de 2011.] http://www.extremeprogramming.org/.
28. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico.* s.l. : Concepción Femández Madrid. 5ta., 1998. ISBN: 84-481-3214-9.
29. **Fenton, Norman E. y Lawrence, Pfleeger Shari.** *Software Metrics. A rigorus and Practical Approach.* 1998. ISBN: 0534954251.
30. **IEEE.** IEEE. [En línea] [Citado el: 13 de Diciembre de 2011.]
31. **Olsina, Luis.** *Métricas e Indicadores.* 2003.
32. **Ubuntu, Comunidad de.** doc.ubuntu-es. [En línea] [Citado el: 8 de Enero de 2012.] http://doc.ubuntu-es.org/Sobre_Ubuntu.
33. **Junio Hamano, Linus Torvalds.** Control de versiones con Git. [En línea] [Citado el: 10 de Enero de 2012.] http://git-scm.com/.
34. **Lang, Jean Philippe.** Redmine. [En línea] [Citado el: 10 de Enero de 2012.] http://www.redmine.org/.
35. **Paradigm, Visual.** Visual Paradigm. [En línea] [Citado el: 12 de Diciembre de 2012.] http://www.visual-paradigm.com/.
36. **Varó, Marzal y Gracia Luengo, Isabel.** *Introducción a la Programación con Python.* 2003.
37. **Chodorow, Kristina y Dirolf, Michael.** *MongoDB: The Definitive Guide.* 2010. ISBN: 978-1-449-38156-1.
38. **Geany.** Geany. [En línea] [Citado el: 12 de Enero de 2012.] http://www.geany.org.
39. **Bass L., Clements P., Kazman R.** *Software Architecture in Practice: Second Edition.* s.l. : Addison-Wesley, 2005.
40. **Marcello Visconti; Hernán Astudillo.** Fundamentos de Ingeniería de Software. *Fundamentos de Ingeniería de Software.* [En línea] 2010. [Citado el: 14 de Enero de 2012.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.
41. **Python.** PEP 8 - Style Guide for Python Code. [En línea] [Citado el: 14 de Enero de 2012.] http://www.python.org/dev/peps/pep-0008/.
42. **Programming, Extreme.** Unit Test. [En línea] [Citado el: 15 de Enero de 2012.] http://www.extremeprogramming.org/rules/unittests.html.
43. **Programming, Extreme**. Acceptance Test. [En línea] [Citado el: 15 de Enero de 2012.] http://www.extremeprogramming.org/rules/functionaltests.html..
44. **Natalia Juristo, Ana M. Moreno, Sira Vegas.** *Tecnicas de evaluación de software.* 2006..
45. **Patton, Ron.** *Software Testing, pagina 408.* s.l. : Sams Publishing, 2005.
46. **Mario G. Piattini, Jose A. Calvo-Manzano, Joaquin Cervera Bravo, and Luis Fernandez Sanz.** *Análisis y diseño de aplicaciones informáticas de gestión, una perspectiva de ingeniería del software, paginas 419-469.* s.l. : Alfaomega, 2004.
47. **Braude, Eric J.** *Ingenieria de software, una perspectiva orientada a objetos, paginas 337-398.* s.l. : Alfaomega Grupo editor, 2003
48. **Beck, Kent.** *Extreme Programming Explained.* s.l. : Addison-Wesley Professional, 1999.

49**. Open Source Systems Monitoring.** Spring Source Hyperic*.* [En línea]. [Citado el: 15 de Noviembre de 2012.] http://www.hyperic.com/products/open-source-systems-monitoring.

50. **Applications Monitoring.** Spring Source Hyperic*.* [En línea]. [Citado el: 16 de Noviembre de 2012.] http://www.hyperic.com/products/applications-monitoring.

GLOSARIO

AJAX: acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo Web para crear aplicaciones interactivas o RIA (Rich Internet Applications).

Base de datos: Conjunto de datos interrelacionados entre sí, almacenados con carácter más o menos permanente en la computadora. O sea, que una base de datos puede considerarse una colección de datos variables en el tiempo.

benchmarks: Técnica utilizada para medir el rendimiento de un sistema o componente del mismo o conjunto de procedimientos programas de computación para evaluar el rendimiento de un ordenador.

Componente de captura de métricas: Es el encargado de recolectar la información de cada servidor PostgreSQL supervisado, así como capturar las métricas a medir, que permitirán evaluar el desempeño de los servidores.

demonio: Es un tipo de proceso no interactivo, es decir, que se ejecuta en segundo plano permitiendo que los usuarios interactúen y trabajen de forma normal en la PC mientras este hace su trabajo.

IETF: la Internet Engineering Task Force (IETF) es una gran comunidad internacional abierta, de diseñadores, operadores, vendedores, e investigadores interesados en la evolución de la arquitectura de Internet y el buen funcionamiento de la Internet.

HTTP: Hypertext Transfer Protocol o HTTP (en español protocolo de transferencia de hipertexto) es el protocolo usado en cada transacción de la World Wide Web.

Indicadores: Son valores que sirven de base para cuantificar conceptos medibles para una necesidad de Información, métodos cuantitativos de evaluación o predicción y ofrecen información para la toma de decisiones.

JSON: (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos.

ManageEngine: Empresa productora de la Enterprise IT Managmet Software perteneciente a la corporación Zoho Corp.

Middleware: es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones.

Glosario de términos

Métricas: Una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.

Monitorización: Se asocia a la acción de controlar o supervisar cuidadosamente una actividad durante un tiempo acordado. Es el proceso continuo y sistemático mediante el cual verificamos la eficiencia y la eficacia de un proyecto mediante la identificación de sus logros y debilidades. Asimismo, es el responsable de preparar y aportar la información que hace posible sistematizar resultados y procesos.

MongoDB: Es una nueva generación de sistemas de gestión de base de datos que combina un modelo de almacenamiento de documentos con un potente motor de consultas de una manera simple.

NC: no conformidades.

Twisted: Framework de red para programación dirigida por eventos escrito en Python y licenciado bajo la licencia MIT

PgBench: Herramienta muy útil e informativa para el monitoreo.

PgFouine: Herramienta más popular de revisión de logs y está escrita en PHP.

pg_stat_database: Vista que define el gestor de bases de datos PostgreSQL utilizada para mostrar información relevante acerca de las bases de datos.

pg_stat_user_indexes: Vista que define el gestor de bases de datos PostgreSQL utilizada para mostrar información relevante acerca de los índices.

Plugin: un plugin es una Aplicación que se relaciona con otra para aportarle una nueva función.

Scintilla: es un componente libre para la edición de código fuente.

Servidor: Es un ordenador que forma parte de una red y que provee servicios a otros ordenadores o sea, sirve información a los que están conectados a él.

Sistema Gestor de Bases de Datos: Es el software que permite la utilización y/o la actualización de los datos almacenados en una (o varias) base(s) de datos por uno o varios usuarios desde diferentes puntos de vista.

Spam: no es más que correo basura o mensaje basura, no deseados o de remitente no conocido.

XML: es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium.

ANEXOS

Anexo 1: Formato canales para la aplicación del patrón de mensajería Publicación-Subscripción.


```
{

"generales”:

        {
        "Uso_de_Memoria_SWAP”: "/lp06-301-14l/Uso_de_Memoria_SWAP/",
        "Uso_CPU”: "/lp06-301-14l/Uso_CPU/",
        "Info_CPU”: "/lp06-301-14l/Info_CPU/",
        "Info_HDD”: "/lp06-301-14l/Info_HDD/",
        "Uso_de_Memoria”: "/lp06-301-14l/Uso_de_Memoria/",
        "Servidor”: "/lp06-301-14l/Servidor/",
        "Estadisticas_memoria" : "/lp06-301-14l/Estadisticas_memoria/",
        "Descripcion_CPU”: "/lp06-301-14l/Descripcion_CPU/",
        "Uso_HDD”: "/lp06-301-14l/Uso_HDD/",
        "Estadisticas_Interfaz_Red”: "/lp06-301-14l/Estadisticas_Interfaz_Red/",
        "Procesos_activos”: "/lp06-301-14l/Procesos_activos/",
        "Descripcion_HDD”: "/lp06-301-14l/Descripcion_HDD/"
        },

"base_datos”:
        {
        "Total_de_Indices”: "/lp06-301-14l/Total_de_Indices/",
        "Tuplas_Muertas”: "/lp06-301-14l/Tuplas_Muertas/",
        "Tuplas_Insertadas”: "/lp06-301-14l/Tuplas_Insertadas/",
        "Tuplas_leidas_x_Esc_Secuencial”: "/lp06-301-14l/Tuplas_leidas_x_Esc_Secuencial/",
        "Indices_Escaneados”: "/lp06-301-14l/Indices_Escaneados/",
        "Conexiones_X_BD" : "/lp06-301-14l/Conexiones_X_BD/",
        "Tuplas_Retornadas”: "/lp06-301-14l/Tuplas_Retornadas/",
        "Tuplas_Vivas”: "/lp06-301-14l/Tuplas_Vivas/",
        "Transacciones_Abortadas”: "/lp06-301-14l/Transacciones_Abortadas/",
        "Transacciones_Finalizadas”: "/lp06-301-14l/Transacciones_Finalizadas/",
        "ObtenerBDs" : "/lp06-301-14l/ObtenerBDs/",
        "Tuplas_Actualizadas”: "/lp06-301-14l/Tuplas_Actualizadas/",
        "Total_de_Bloqueos”: "/lp06-301-14l/Total_de_Bloqueos/",
        "Tuplas_Eliminadas”: "/lp06-301-14l/Tuplas_Eliminadas/",
        "Escaneo_Secuencial”: "/lp06-301-14l/Escaneo_Secuencial/"
        }

}
```

Anexo 2: Acta de liberación del producto.

Validez desconocida
Digitally signed by Marianela Gutiérrez Rodriguez
Date: 2012.06.13 15:17:18 CDT
Reason: Documento Oficial de la Facultad 6
Location: Cuba

Universidad de las Ciencias Informáticas

CENTRO DE TECNOLOGÍAS

"Nos distingue la excelencia"

Acta de Liberación de Productos Software

Fecha de liberación: 13/06/2012

Emitida a favor de: *Departamento de PostgreSQL, Centro de Tecnologías de Gestión de Datos.*
Nombre del Proyecto: *Sistema de monitorización de servidores PostgreSQL: Naire v2.0*
Nombre del Producto: *Componente de captura de métricas para el Sistema de monitorización de servidores PostgreSQL: Naire v2.0*

1. Datos del producto

Artefacto	Versión	Estado final	Cantidad Iteraciones	Tipos de pruebas realizadas
Aplicación	*2.0*	*0 NC*	*2*	*Pruebas funcionales, de ortografía y redacción.*

Dairys Febles Pérez
Nombre y Apellidos
Asesora de Calidad

Rosnel Venero Acosta
Nombre y Apellidos
Responsable Proyecto

Grupo de Mercadotecnia DATEC

datec@uci.cu