Cloud computing con herramientas open-source para Internet de las cosas

Ariel M Campoverde M., Dixys L Hernández R., Bertha E Mazón O.

Unidad Académica de Ingeniería Civil. Universidad Técnica de Machala, Km. 5 1/2 Vía Machala Pasaje. Machala, Ecuador, 070102.

Autores de correspondencia: acampoverde_est@utmachala.edu.ec, dhernandez@utmachala.edu.ec, bmazon@utmachala.edu.ec

Fecha de recepción: 28 de septiembre 2015 - Fecha de aceptación: 12 de octubre 2015

RESUMEN

Uno de los componentes de un sistema de telemetría basado en Internet de las Cosas (IoT) es Cloud Computing; este está formado por un servidor o conjunto de servidores interconectados que involucran hardware y software para ofrecer servicios de monitoreo y control. En este trabajo se presenta una estructura de Cloud Computing para IoT de tipo SaaS (Software as a Service), que aplica herramientas open source seleccionadas en base a un estudio técnico, cuyos resultados fueron validados mediante la implementación de un dashboard en tiempo real. El objetivo de este trabajo consiste en definir una estructura a nivel de Cloud que permita implementar cualquier aplicación en el ámbito de IoT. Para ello se analizaron las herramientas IoT actuales aprobadas por la comunidad científica y se seleccionaron las óptimas para entornos con recursos restringidos, típicos en las redes de sensores inalámbricos (WSN).

Palabras clave: Internet de las Cosas, IoT, Cloud Computing, herramientas open-source, SaaS.

ABSTRACT

One of the components of a telemetry system based on Internet of Things (IoT) is Cloud Computing, which consists of one or multiple interconnected servers that involve hardware and software to provide monitoring and control services. This paper shows a SaaS (Software as a Service) Cloud Computing structure for IoT by applying chosen open source tools based on a technical study whose results were validated by implementing a real time dashboard. The objective of this work is to define a Cloud-level structure that allows deploying any application involved in IoT. For this purpose, current open source tools being adopted by the scientific community were analyzed and then those ones which better fit for restricted resources environments were selected, typical of wireless sensor networks (WSN).

Keywords: Internet of Things, IoT, Cloud Computing, open source tools, SaaS.

1. INTRODUCCIÓN

El creciente ámbito de Internet de las Cosas (IoT) ha tomado un gran impulso en estos últimos años. El término se refiere a la interconexión de cosas (sensores, actuadores, televisores, etc.) por medio de una red que funciona incluso sin la intervención de las personas.

La arquitectura general de IoT tomada como referencia en (Chen *et al.,* 2011), ha sido ampliada en la Figura 1, que especifica la arquitectura de una implementación IoT mostrando los distintos dominios tomados en cuenta. Ésta investigación se centra en el dominio de aplicación, que involucra una Cloud Computing. Se ha definido (Hernández *et al.,* 2015) que una Cloud Computing eficiente para IoT está compuesta básicamente por: software para servidor web, framework para aplicación web *(en tiempo real)*, servidor de base de datos *(NoSQL)* y protocolo de comunicación eficiente.

Este documento propone los componentes adecuados que constituyen la base de un sistema para IoT a nivel de Cloud Computing; a partir de una comparación de las herramientas actualmente disponibles y aprobadas por la comunidad científica en el ámbito de IoT. En primer lugar se caracterizan las herramientas más destacadas, luego se selecciona una herramienta para cada componente de la Cloud Computing y por último se ejemplifica con un sistema IoT implementado con las herramientas elegidas.

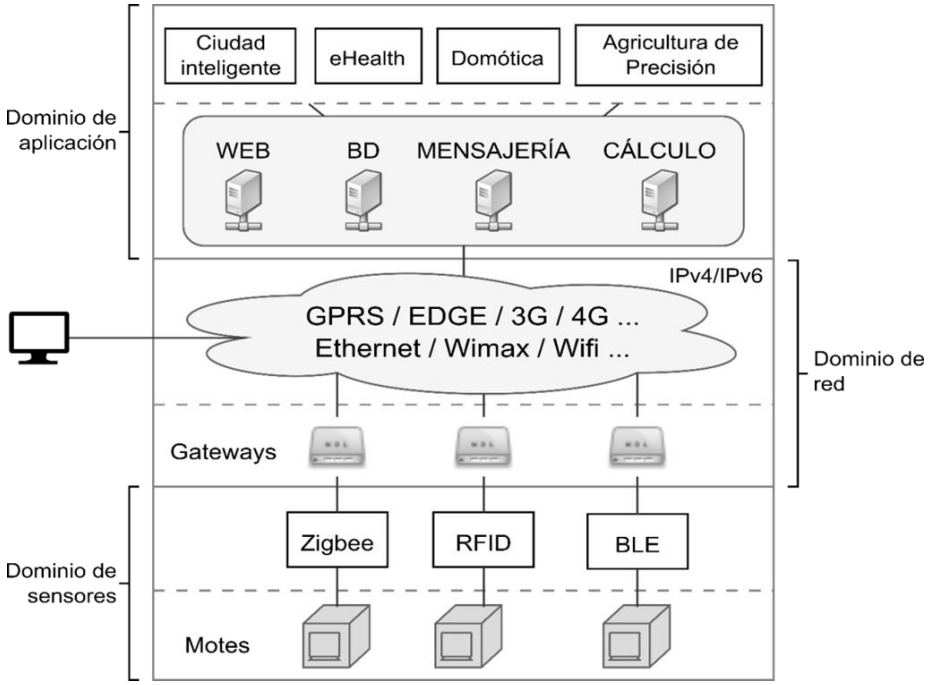

Figura. 1. Dominios de la arquitectura IoT.

2. CLOUD COMPUTING Y HERRAMIENTAS OPEN-SOURCE PARA IOT

Se analizaron herramientas open source para cada componente de la Cloud Computing de IoT, seleccionando las de mejor prestación en cada caso.

2.1. Cloud Computing

En Cloud Computing dos aspectos son de vital importancia, el modelo de despliegue y el modelo del servicio a implementar (Sosinsky, 2011). Entre los tres principales modelos de despliegue se encuentran; la nube pública, la cual se caracteriza por su libre acceso desde cualquier parte del mundo con posibles restricciones; la nube privada, la cual comúnmente se implementa dentro de las instalaciones de una empresa (on-premise) y es de su uso exclusivo; y la nube híbrida, la cual es una combinación de una nube pública y privada al mismo tiempo. Debido a que IoT está obviamente ligado a internet, una nube pública es la opción viable para su implementación, si lo que se desea es una alternativa on-premise, se pierde el concepto IoT y la solución viable en ese caso sería M2M (Machine-to-Machine).

Con respecto a los modelos de servicio, las posibilidades son: SaaS (software como servicio), PaaS (plataforma como servicio) e IaaS (infraestructura como servicio). SaaS ofrece de forma predefinida una infraestructura, plataforma y software de aplicación listo para su uso, en donde todo aspecto de la nube es abstracto para el usuario; algunos ejemplos de SaaS son Google Docs, Gmail y Dropbox. PaaS ofrece una infraestructura y plataforma predefinida a partir de las cuales el usuario puede implementar su aplicación mediante herramientas especificadas por el proveedor del servicio (bases de datos, lenguajes de programación, entre otros), en este modelo únicamente la infraestructura

de la nube es completamente abstracta al usuario. Microsoft Azure y Google Cloud Platform son los servicios PaaS más relevantes del mercado en la actualidad. IaaS provee un entorno de virtualización de recursos físicos para que el usuario sea el encargado de definir una infraestructura que se ajuste a sus necesidades, los servicios IaaS más populares son Amazon AWS y Rackspace.

En cloud computing la virtualización (Gouda *et al.*, 2014) es algo muy importante, consiste en simular un solo espacio físico como si fueran varios, aprovecha así de forma óptima los recursos del equipo, brinda mayor facilidad de mantenimiento a la infraestructura y abarata costos de operación. Es común utilizar un software especializado para la virtualización, mejor conocido como Hypervisor o Virtual Machine Monitor (VMM), el cual en el mejor de los casos se instala directamente en el equipo sin necesidad de un Sistema Operativo anfitrión.

2.2. Software de Servidor Web

Se ha tomado en consideración a los tres servidores web de mayor relevancia en la actualidad, como son Apache, Nginx y Lighttpd.

Apache es el servidor web más utilizado a nivel mundial, es estable y robusto; su manejo de peticiones se da por medio de hilos y procesos, donde cada petición es separada en un hilo de procesamiento diferente utilizando sockets síncronos. En cuanto a su rendimiento, Apache ofrece un servicio bastante eficiente, pero para aplicaciones que requieren de un tráfico considerablemente alto quizás no sea la mejor opción, esto debido a su naturaleza síncrona (Ellingwood, 2015).

Nginx es un servidor web y proxy inverso (mecanismo desplegado en una red para proteger a los servidores HTTP de una intranet, realiza funciones de seguridad que protegen a los servidores internos de ataques de usuarios en internet). Gestiona las peticiones por medio de una arquitectura orientada a eventos donde las peticiones son aceptadas mediante sockets asíncronos y son procesadas en un único hilo de ejecución, esto a fin de reducir memoria y uso de CPU. La comunidad de Nginx no es tan amplia en comparación a la de Apache debido a su corto tiempo de vida en el mercado. Los tiempos de respuesta son menores con Nginx, Apache se torna cada vez más lento cuando las peticiones aumentan y además tiende a usar más ancho de banda para servir las mismas peticiones (Nedelcu, 2013).

Lighttpd es un servidor HTTP sencillo pero potente (Lighttpd, 2015), cuya arquitectura es orientada a eventos. Al igual que Nginx, funciona en un solo hilo de procesamiento. Está diseñado y optimizado para ambientes de alto rendimiento con un consumo de memoria bastante reducido respecto a otros servidores web. Su comunidad y documentación no es tan grande, es menor que la de Nginx, pero esto no ha sido impedimento para que sea utilizado por grandes compañías tales como YouTube (para su servidor de descargas), Wikipedia (para su servidor de subida de imágenes), además de otros sitios con tráfico considerable.

2.3. Framework para aplicación web

El término framework se refiere a un conjunto de herramientas prediseñadas para un lenguaje de programación específico, en este caso para construir aplicaciones web. Los frameworks en tiempo real ofrecen mejores alternativas que los de arquitectura bloqueante en cuanto al manejo de grandes cantidades de peticiones concurrentes, ya que en lugar de dejar un proceso bloqueado mientras espera su finalización, la aplicación puede comenzar el proceso y darle una llamada de retorno (callback) que notificará a la aplicación cuando aquel proceso se complete; deja el bucle de entrada-salida disponible para servir a otros clientes. Por tales motivos, debido a que IoT supone una gran cantidad de dispositivos conectados al mismo tiempo, se ha optado por analizar solo a los frameworks en tiempo real.

Node.js es un framework que utiliza un modelo no bloqueante, orientado a eventos, que lo hace ligero y eficiente (About Node.js, 2015). Una de sus cualidades es su capacidad nativa de trabajar con websockets. Node.js es la plataforma web más popular para aplicaciones en tiempo real y por ende es una excelente alternativa para IoT, además cuenta con una gran comunidad que desarrolla cada vez más librerías que se incorporan con facilidad en cualquier proyecto. Actualmente ya se disponen incluso de librerías IoT para open hardware como: Arduino, Raspberry, Beaglebone, entre otros. Node

funciona con JavaScript y ha evolucionado hasta ser utilizado en dispositivos embebidos, pudiendo llegar a convertirse en un lenguaje de programación para aplicaciones de servidor, cliente (navegador web) y dispositivos embebidos.

Tornado es un framework potente y escalable escrito en Python. Es lo suficientemente robusto como para manejar un tráfico web intensivo (Dory *et al.,* 2012), es fácil de configurar y puede ser utilizado para una gran variedad de aplicaciones. Tornado es capaz de manejar decenas de miles de conexiones concurrentes y está diseñado específicamente para ser un framework de alto rendimiento. También posee varias herramientas para seguridad y autenticación de usuarios. Para reducir al mínimo el costo de conexiones concurrentes utiliza un bucle de eventos de un solo subproceso. Esto significa que todo el código de aplicación debe ser asíncrono y sin bloqueo, porque sólo una operación puede estar activa a la vez.

Play es un framework web no bloqueante orientado a eventos, funciona con Java o Scala, dos lenguajes de programación caracterizados por su alto rendimiento. Al igual que Node y Tornado, está construido para soportar un gran número de conexiones concurrentes sin que su rendimiento se vea afectado. Play tiene un gran soporte para las bases de datos no relacionales y para websockets. Su gestor de paquetes (Maven) es el segundo más grande después de npm de Node. A diferencia de Node y Tornado, Play es un framework *full-stack*, es decir, viene con todas las herramientas para la creación de aplicaciones web ya incorporadas para que los desarrolladores se concentren en la lógica de negocio y no tanto en la infraestructura de su aplicación (Petrella, 2013).

2.4. Base de datos

Para almacenar y procesar grandes cantidades de información, las alternativas de bases de datos relacionales podrían no satisfacer todas las necesidades requeridas (McCreary *et al.,* 2014). Es ahí donde las bases de datos NoSQL entran en acción debido a la flexibilidad que proporcionan al momento de almacenar y gestionar los datos, siendo esto muy importante en el ámbito de IoT.

MongoDB es una base de datos de propósito general. La facilidad que proporciona al momento de realizar consultas a la misma es una de sus cualidades, ya que posee un conjunto de funciones con algoritmos lo suficientemente aptos para devolver los datos solicitados en consultas no tan complejas. A pesar de ser un almacén de documentos también permite realizar consultas complejas, distinguiéndose de otras bases de datos y convirtiéndose en una opción muy potente. Proporciona además, una lista cada vez mayor de características importantes (Chodorow *et al.,* 2013) como indexación secundaria, única, compuesta, geoespacial y de texto completo.

El teorema CAP (Consistencia-Disponibilidad-Tolerancia de particionado) (Gilberth *et al.,* 2012) define que una base de datos sólo puede cumplir como máximo dos de sus apartados al mismo tiempo. Consistencia se refiere a que todos los clientes de la base de datos son capaces de ver los mismos datos, incluso con actualizaciones concurrentes. La disponibilidad asegura que todos los clientes sean capaces de acceder a versiones posiblemente distintas de los datos, pero siempre los tendrán a su disposición. Tolerancia de particionado se refiere a la facilidad de la base de datos para desplegarse en distintos servidores. En tal sentido, MongoDB asegura consistencia de sus datos y tolerancia de particionado (MongoDB, 2010), importante para IoT, especialmente la consistencia ya que sería un problema si un cliente visualizara en su dashboard (Interfaz de usuario con gráficas representativas de datos históricos y/o en tiempo real.) que un actuador esté apagado, mientras que otro que recibió una versión de ese dato anterior, visualiza que el mismo actuador aún está encendido. MongoDB ha sido utilizada por grandes y reconocidas empresas a nivel mundial e incluso se ha comprobado su eficiencia en entornos IoT en (MongoDB IoT, 2015).

CouchDB es una base de datos cuya arquitectura interna es bastante tolerante a fallos, es decir, cuando se produce algún error, éste se aísla en un entorno controlado y se le da tratamiento. Las consultas se basan en vistas *map-reduce* escritas en código JavaScript. Dentro del teorema CAP se garantiza disponibilidad y tolerancia de particionado (Anderson *et al*., 2015) y presupone un problema al priorizar la disponibilidad ya que distintos clientes podrían visualizar información diferente de un mismo dato, unos podrían obtener una versión actual, mientras que otros obtendrían una versión anterior, lo cual no sería conveniente.

Couchbase es una base de datos distribuida, originalmente basada en CouchDB y Membase, aprovecha un sistema integrado de capa de almacenamiento en caché de RAM, soporta operaciones muy rápidas, tales como crear, almacenar, actualizar y obtener datos. Dentro del teorema CAP, Couchbase se comporta diferente según su implementación (Biyikoglu, 2014), por ejemplo, si consta de un único clúster, entonces garantizaría consistencia de los datos y tolerancia de particionado, mientras que en un multi-clúster se garantizaría disponibilidad de los datos y tolerancia de particionado.

2.5. Protocolo de comunicación

El popular protocolo HTTP, no es la opción más viable para un entorno IoT ya que requiere de muchos recursos y ancho de banda. Se debe tener en cuenta que los dispositivos que usen este protocolo, tendrán limitaciones en la duración de sus baterías, memoria, ancho de banda, entre otros. Por tanto la selección adecuada del protocolo de comunicación incidirá tanto en el rendimiento individual de cada dispositivo, y en todo el sistema IoT.

El protocolo MQTT está diseñado específicamente para redes de dispositivos Máquina-A-Máquina (M2M) y aplicaciones móviles con restricciones de memoria, batería, red, entre otras. Sus mensajes se transmiten en formato binario y trabaja con el modelo publicador/suscriptor (HiveMQ, 2015), el cual funciona como una vía de comunicación de *uno-a-varios* clientes, lo cual es útil en IoT para el monitoreo de dispositivos remotos. Esto no impide establecer una conexión *uno-a-uno* (útil en caso de control de dispositivos en IoT) o de *varios-a-uno*. MQTT provee tres niveles de calidad del servicio (QoS) para la entrega de mensajes; nivel *0* (Máximo una vez entregado), nivel *1* (Al menos una vez entregado) y nivel *2* (Exactamente una vez entregado). Los clientes MQTT se comunican mediante tópicos, los cuales funcionan como un canal de comunicación privado. Existe incluso una variante de MQTT destinada a redes de sensores con recursos muy restringidos, la cual es definida en (Stanford-Clark & Truong, 2015). Existen varios brokers que implementan el protocolo MQTT, siendo el más famoso de ellos el bróker open source Mosquitto.

El Protocolo de Mensajería de Aplicaciones Web (WAMP, 2015) provee dos modos de comunicación: llamadas de procedimientos remotos (RPC) y publicador/subscriptor (PUB/SUB). Aunque su medio de transmisión es por defecto WebSockets, es posible utilizar medios diferentes (WAMP Protocol, 2015), donde los datos son transmitidos en texto plano o en binario. Las llamadas de procedimientos remotos son útiles para comunicaciones de uno a uno. Por otro lado, el modo publicador/suscriptor es utilizado en comunicaciones de uno a varios o de varios a uno. La comunicación entre clientes se establece mediante tópicos representados mediante URIs. WAMP es capaz de comunicarse de forma nativa con un navegador web sin necesidad de intermediarios. Por ende, resulta sencillo servir datos en tiempo real directamente en la interfaz de un usuario conectado desde algún navegador.

El Protocolo de Aplicación Restringida (CoAP) es utilizado también en dispositivos y redes cuyos recursos son limitados. Funciona bajo UDP con el modelo REST, y además utiliza el modo publicador/suscritor. Difiere de HTTP ya que CoAP no conlleva un alto consumo de recursos y descarta las cabeceras adicionales no necesarias en el entorno IoT. A pesar de sus similitudes con HTTP, CoAP no puede comunicarse nativamente con un navegador web, por lo cual se debe utilizar un gateway como intermediario que traduzca CoAP a HTTP. CoAP ofrece un manejo de calidad del servicio mediante el mecanismo de retransmisión *Stop-and-Wait* (Técnica utilizada para proveer transferencia confiable de paquetes bajo un sistema de transferencia poco confiable.), sin embargo, no posee un mecanismo de seguridad integrado, aunque puede ser provista por el protocolo de Seguridad de Capa de Transporte de Datagramas (DTLS). Uno de los mayores inconvenientes de DTLS para CoAP, es que no soporta el multicast (Karagiannis *et al.,* 2015), lo cual impone una limitación, ya que ésta es una de las características más importantes de CoAP.

3. RESULTADOS Y DISCUSIÓN

De acuerdo a las características de cada una de las herramientas descritas, se procedió a hacer una selección discriminatoria dentro de cada uno de los componentes de IoT.

En el caso del software para servidor web, a pesar que Apache es el más popular hoy en día, se puede apreciar que no es el más indicado para su implementación en IoT debido a su arquitectura de naturaleza bloqueante. Por su parte, Nginx y Lighttpd representan opciones muy viables, sin embargo, Nginx actúa también como un proxy inverso, lo cual añade una capa extra de seguridad entre el cliente y el servidor web, necesarios en entornos IoT como en cualquier aplicación en general.

En cuanto a los frameworks para el desarrollo de la aplicación web; las tres opciones han demostrado ser candidatos muy eficientes para IoT, pero Node.js es la elección más conveniente debido a su amplia gama de paquetes disponibles que pueden ser implementados para IoT. Además ya existen varias aplicaciones funcionales en Node.js como The Thing System (The Thing System, 2015), Node-RED (Node-RED, 2015), e incluso un microcontrolador llamado Tessel (Tessel, 2015) que funciona en su totalidad con Node.

De las distintas bases de datos analizadas, CouchDB presentó inconvenientes debido a su diseño, el cual dificulta el filtrado de datos por más de un campo, y además, garantiza disponibilidad de los datos antes que consistencia, un comportamiento no deseado en IoT. Couchbase en cambio, puede ser una opción muy buena si se implementa como un único clúster, caso contrario garantizaría disponibilidad antes que consistencia de los datos. MongoDB es la elección más adecuada, puesto que asegura consistencia de los datos sin importar el modo de implementación de la base de datos, siendo esta una gran ventaja sobre las demás en cuanto a IoT se refiere.

Si el almacenamiento de la cloud está diseñado para cantidades inmensas de información (lo cual es el caso de BigData), resulta conveniente utilizar soluciones de procesamiento y almacenamiento distribuido como Hadoop (Karanth, 2014), a fin de asegurar la escalabilidad de la infraestructura. Hadoop posee un sistema de archivos (HDFS) en el cual una vez que ha sido almacenada su información, ésta es inmutable; es decir, no puede ser modificada. Esto no representa inconvenientes en un sistema de telemetría como IoT ya que la información capturada desde los sensores o actuadores es fija. Cabe destacar que Hadoop y MongoDB son bastante compatibles y pueden ser utilizados de forma conjunta (Sitto & Presser, 2015).

De los protocolos de comunicación analizados, WAMP posee métodos de comunicación bastante adaptables a IoT, como RPC y Pub/Sub, pero su único inconveniente es no adaptarse bien a entornos con capacidades restringidas, específicamente en el caso del consumo de batería de los dispositivos remotos.

Por otra parte CoAP, está diseñado para este tipo de implementaciones con restricciones, pero puede perder estas características al añadir una capa de seguridad DTLS al protocolo. Si bien DTLS resuelve el problema de seguridad, aumenta también el consumo de recursos de los dispositivos del dominio de sensores.

MQTT es la mejor alternativa por ser un protocolo concebido para ambientes con recursos restringidos, diseñado exclusivamente para sistemas de telemetría con un sistema de seguridad integrado y además cuenta con MQTT-SN, una versión optimizada del protocolo para para redes de sensores, que minimiza al máximo el overhead.

Como resultado de este análisis, se resume las herramientas seleccionadas para el ámbito IoT a nivel de Cloud Computing, estas son: Nginx, Node.js, MongoDB y MQTT. En la tabla 1 se resumen los criterios tomados en cuenta para esta selección.

A partir de esta selección se propone una arquitectura para Cloud Computing, cuyo esquema simplificado se muestra en la Figura 2.

Como otro resultado de este trabajo y en aras de validar la selección de las herramientas antes mencionadas, se ha construido una aplicación web en tiempo real que utiliza dichas herramientas seleccionadas, consiguiendo excelentes resultados. Esta aplicación tiene un enfoque general para IoT, es capaz de persistir la información de telemetría en base de datos, generar reacciones ante eventos predefinidos, ejecutar tareas programadas, presentar información en un dashboard en tiempo real,

entre otras características; a partir de las cuales es posible desarrollar futuras aplicaciones de enfoque específico a distintos campos de la ciencia como la Agricultura de Precisión, Ciudades Inteligentes, Domótica, etc. En la Figura 3 se puede apreciar el panel de monitoreo y control para IoT con diferentes variables analógicas y digitales en tiempo real.

La arquitectura de la cloud computing propuesta en conjunto con la aplicación desarrollada se ha implementado bajo el modelo de servicio SaaS con el fin de ofrecer un servicio en línea a los distintos usuarios que así lo requieran, contando cada uno de ellos con su propio entorno de gestión y operación, lo cual obedece a una arquitectura multitenancy.

La propuesta aquí realizada podría ser implementada como PaaS siempre y cuando se ofrezca únicamente la plataforma y no la aplicación desarrollada, de modo que cada usuario sea capaz de crear su propia aplicación IoT utilizando las herramientas open source aquí propuestas.

Una implementación de tipo IaaS queda descartada debido a que la propuesta ya es en sí una infraestructura diseñada y lista para su despliegue.

El modelo de despliegue está pensado para ser una nube pública off-premise con el fin de ofrecer un solo servicio para varios clientes, los cuales se deben encargar únicamente de gestionar las cuentas de usuario a las que tengan acceso.

La Cloud propuesta puede ser desplegada en un servicio IaaS como Amazon AWS o Rackspace, mas no en un servicio PaaS, ya que no todas las tecnologías aquí seleccionadas son soportadas por los proveedores más famosos de este tipo de servicio.

Tabla 1. Análisis para la selección de las herramientas adecuadas en IoT.

CATEGORÍA	HERRAMIENTA	VENTAJAS	DESVENTAJAS
Servidor Web	Apache	− Amplia comunidad y documentación.	− Arquitectura bloqueante.
	Nginx	− Arquitectura orientada a eventos (no bloqueante). − Proxy inverso.	− Comunidad menor a la de Apache.
	Lighttpd	− Arquitectura orientada a eventos (no bloqueante).	− Comunidad menor a la de Nginx
Framework de desarrollo en tiempo real	Node.js (JavaScript)	− Varias implementaciones IoT existentes. − Amplia comunidad. − Posee el mayor repositorio de paquetes. − Framework de alto rendimiento.	− Mayor curva de aprendizaje.
	Tornado (Python)	− Framework de alto rendimiento.	− Comunidad no tan amplia. − Mayor curva de aprendizaje.
	Play Framework (Scala y Java)	− Framework full-stack. − Framework de alto rendimiento.	− Comunidad no tan amplia
Base de Datos	MongoDB	− Asegura consistencia de datos. − Amplia comunidad. − Conserva conceptos de RDBMS. − Estructura de datos BSON.	− No posee mecanismos de notificación en Tiempo Real.
	CouchDB	− Notificaciones en tiempo real ante algún cambio. − Sistema sencillo de replicación.	− No asegura consistencia de datos. − Mayor curva de aprendizaje.
	Couchbase	− Trabaja intensamente con datos almacenados en caché.	− No asegura consistencia de datos en entorno clúster.
Protocolos de Comunicación	MQTT (Publish/Sucribe)	− Entrega mensajes pendientes cuando los clientes se conectan. − Diseñado para sistemas de telemetría. − Estructura de tópicos robusta. − Posee seguridad integrada. − Capacidad de comunicarse	− No posee mecanismo para reconocer el formato de los mensajes.

CATEGORÍA	HERRAMIENTA	VENTAJAS	DESVENTAJAS
		directamente con un navegador web.	
	WAMP (Publish/Sucribe y RPC)	– Modos de comunicación RPC y PUB/SUB. – Capacidad de comunicarse directamente con un navegador web.	– No posee mecanismo para reconocer el formato de los mensajes. – No optimiza el consumo de recursos de los dispositivos.
	CoAP (Publish/Sucribe y REST)	– Posee mecanismo para reconocer formato de los mensajes.	– No posee sistema de seguridad integrado. – No puede comunicarse directamente con un navegador web.

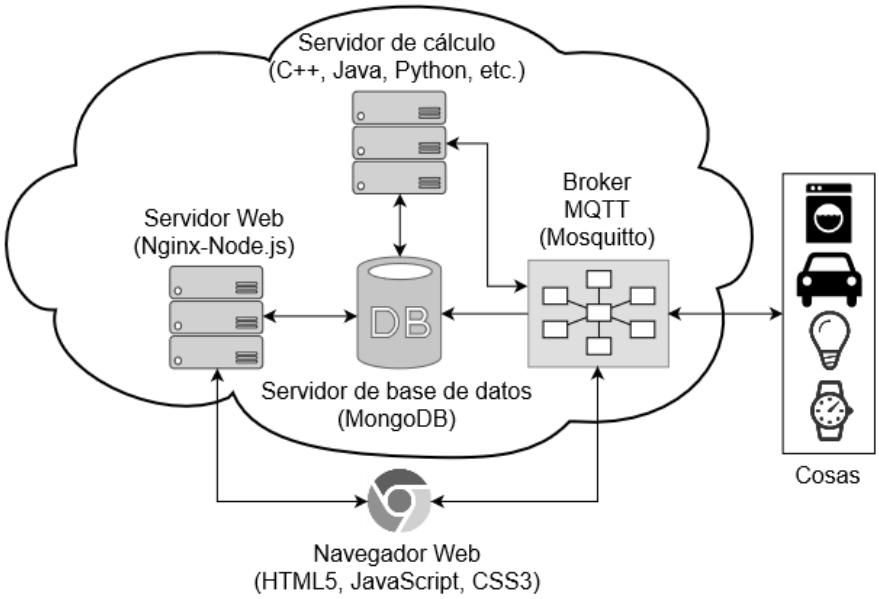

Figura. 2. Estructura propuesta de Cloud Computing simplificada.

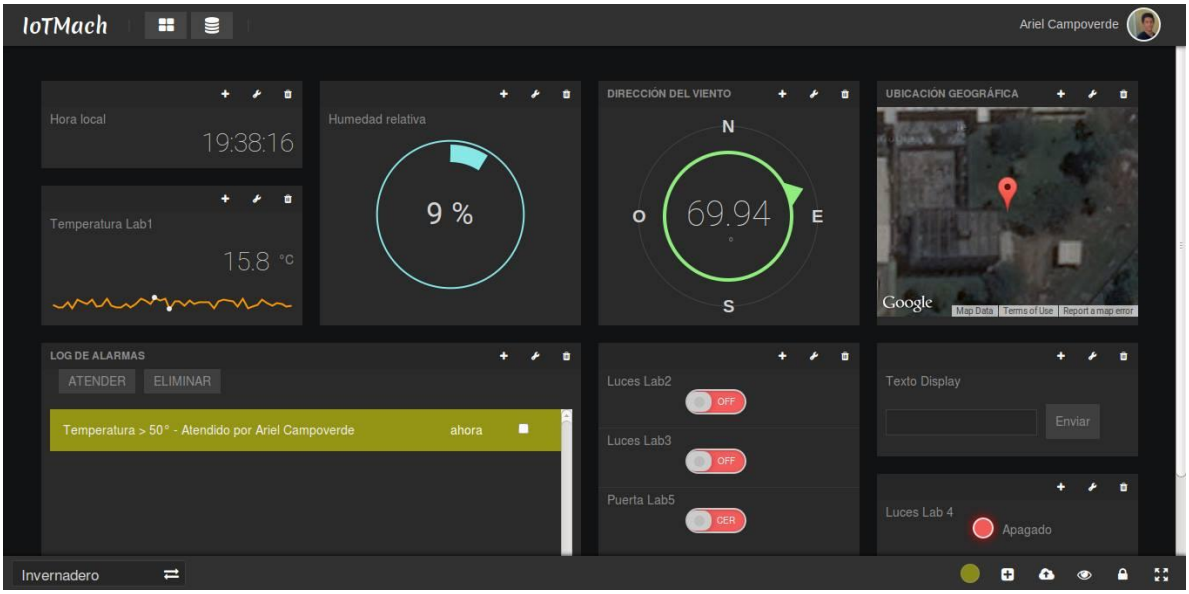

Figura. 3. Dashboard de propósito general funcionando en tiempo real.

4. CONCLUSIONES

La estructura de la Cloud Computing propuesta para IoT se realizó mediante un análisis de las herramientas actuales más destacadas disponibles e implementadas por la comunidad, que fueron explicadas a lo largo de este artículo. Se obtuvo como resultado que Node.js, MongoDB, MQTT y Nginx son un conjunto de herramientas open source cuya implementaión es viable para IoT, además se propone una cloud pública bajo el modelo de servicio SaaS. Esta arquitectura fue implementada en los servidores centrales de la Universidad Técnica de Machala sobre el sistema operativo CentOS 7, como parte de un Proyecto de Agricultura de Precisión que actualmente ejecuta dicha institución; la implementación ha demostrado alta eficiencia en un entorno IoT real (sensores y actuadores instalados en los campos de cultivo) y simulado (simulación software de sensores que envían información de forma constante el bróker MQTT), lo cual demuestra la validez de la estructura propuesta para su uso en múltiples aplicaciones en el ámbito del Internet de las Cosas. Esta implementación fue realizada exclusivamente a nivel de software, por tanto se recomienda para trabajos futuros definir una arquitectura física de servidores y su interacción; implementando además en cada uno de ellos un entorno clúster que permita soportar eficientemente la propuesta planteada.

REFERENCIAS

About Node.js, 2015. Disponible en https://nodejs.org/en/about/.

Anderson, C., J. Lehnardt, N. Slater, 2015. *CAP en CouchDB*. Disponible en http://guide.couchdb.org/draft/consistency.html.

Biyikoglu, C., 2014. *CAP en Couchbase*. Disponible en http://blog.couchbase.com/cap-theorem-and-couchbase-server-time-xdcr.

Chen, H., X. Jia, H. Li, 2011. A brief introduction to IoT gateway: *Proceedings of ICCTA*.

Chodorow, K., 2013. *MongoDB The Definitive Guide Second Edition*. San Francisco: O'REILLY, 3-4 pp.

Dory, M., A. Parrish, B. Brendam, 2012. *Introduction to Tornado*. California: O'REILLY, 2 pp.

Ellingwood, J., 2015. *Apache vs Nginx: Practical Considerations*. Disponible en https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations

Gilberth, S, N. Lynch, 2012. Perspectives on the CAP Theorem. *Computer: IEEE Computer Society*.

Gouda, K., A. Patro, D. Dwivedi, N. Bhat, 2014. Virtualization Approaches in Cloud Computing. *International Journal of Computer Trends and Technology (IJCTT).*

Hernández, D., B. Mazón, A. Campoverde, 2015. Cloud Computing para el Internet de las Cosas. Caso de estudio orientado a la agricultura de precisión: *I Congreso Internacional de Ciencia y tecnología UTMACH 2015*. ISBN 978-9942-21-149-1.

HiveMQ, 2015. *Paradigma de mensajería PUB/SUB*. Disponible en http://www.hivemq.com/mqtt-essentials-part2-publish-subscribe/.

Karagiannis, V., P. Chatzimisios, F. Vazques, J. Zarate, 2015. A Survey on Application Layer Protocols for the Internet of Things. *Transaction on IoT and Cloud Computing*.

Karanth, S., 2014. *Mastering Hadoop*. Birmingham: Packt Publishing, 11 pp.

Lighttpd, 2015. *Sitio oficial*. Disponible en http://www.lighttpd.net/

McCreary, D., A. Kelly, 2014. *Making sense of NoSQL*, Shelter Island: Manning, 19-20 pp.

MongoDB, 2010. *On Distributed Consistency*. Disponible en http://blog.mongodb.org/post/475279604/on-distributed-consistency-part-1.

MongoDB IoT, 2015. *MongoDB en entornos IoT*. Disponible en http://www.mongodb.com/use-cases/internet-of-things.

Nedelcu, C., 2013. *NginX HTTP Server Second Edition*. Birmingham: Packt Publishing, 213-217 pp.

Node-RED, 2015. *Sitio official de Node-RED*. Disponible en http://nodered.org/.

Petrella, A., 2013. *Learning Play! Framework 2*. Birmingham: Packt Publishing, 250 pp.

Sitto, K., M. Presser, 2015. *Field Guide to Hadoop*. California: O'REILLY, 31-33 pp.

Sosinsky, B., 2011. *Cloud Computing Bible*. Indiana: Wiley Publishing Inc., 3 pp.

Stanford-Clark, A., H. Truong, 2015. *MQTT-SN Specification*. Disponible en http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf.

Tessel, 2015. *Sitio oficial de Tessel*. Disponible en https://tessel.io/.

The Thing System, 2015. *Sitio oficial The Thing System*. Disponible en http://thethingsystem.com/

WAMP, 2015. *Transportes alternativos para WAMP*, 2015. Disponible en https://tools.ietf.org/html/draft-oberstet-hybi-tavendo-wamp-02#section-5.3.