Aplicaciones bajo Arquitectura de Microservicios con una Descripción Semántica

Laura cristina Becerra González
Estudiante de Maestría en Ciencias de la Información y las Comunicaciones
Universidad Distrital Fransisco José de Caldas
Bogota D.C. , Colombia
Email: lauracriss@gmail.com

***Resumen*—Actualmente, los desarrollos web se encuentran divididos en dos factores; El primero, involucra una restricción debido a la interoperabilidad entre aplicaciones/servicios desde el punto de vista semántico, se pretende aportar a dicha interoperabilidad. La segunda, es una necesidad de independencia y escalabilidad bajo la arquitectura orientada a microservicios con el fin de hacer más estable, ágil y eficiente una aplicación web. Una herramienta software funcional en una empresa, debe soportar clientes de distintos tipos esto incluye navegadores de escritorio y aplicaciones móviles. Igualmente, dicha aplicación en algunas ocasiones podría integrarse con otras aplicaciones por medio de un servicio web o un intermediario de mensajes. El procedimiento de ejecución del servicio web, se encuentra dado por medio de una base de datos y el intercambio de mensajes con los otros sistemas cuya respuesta se muestra en HTML / JSON / XML.**

***Keywords*—IEEEtran, journal, LaTeX, paper, template.**

I. Introducción

Las aplicaciones generadas a través del entorno web, ofrecen diferentes intereses, por ejemplo aplicaciones de noticias, redes sociales, comercio electrónico entre otros. La construcción de dichas aplicaciones, ha llevado a diferentes investigaciones uno de ellos son los servicios web semánticos y su arquitectura e implementación. Al descomponer una aplicación en un conjunto de servicios que colaboran entre sí, por ejemplo el servicio de gestión de clientes, gestión de proveedores, facturación se convierte en una aplicación basada en micro-servicios cuya comunicación se da por medio de protocolos como HTTP/REST Cada servicio se ejecuta de modo independiente y en algunos casos cada uno con una base de datos independiente. En este artículo se pretende brindar una orientación para implementar dicha arquitectura identificando cómo criterios básicos los siguientes: •

- Debe existir un equipo de desarrolladores que trabajen en ello.
- La aplicación debe ser fácil de entender y modificar
- Se debe ejecutar varias copias de la aplicación en diferentes equipos con el fin de satisfacer los requisitos de escalabilidad y disponibilidad

II. Antecedentes y Marco Teórico

En el mundo de la programación, existen diversos enfoques que aportan a mejorar la integración de la información.

Figura 1: Proceso para implementar un micro servicio WEB, fuente: Autor

La arquitectura de micro servicios web y los estándares de servicios semánticos que se definen sobre ella cómo OWL-S, WSMO proporcionan un enfoque de desarrollo e implantación de servicios[1], para términos de esta investigación, en primera instancia se va a identificar terminología concerniente a web semántica, seguido de la estructuración de micro servicios.

II-A. Contexto de aplicaciones basadas en web semántica

Según Herman [2], el objetivo de la Web Semántica es crear un entorno general para el intercambio de información; de igual manera que prevé la interconexión de las administraciones de información personal y el intercambio global de datos comerciales, científicos y culturales. La construcción de información que es comprensible por los ordenadores en la Web se está convirtiendo rápidamente en prioridad clave para organizaciones, individuos y comunidades [3].

Se dice que: "Una ontología es una descripción formal de los conceptos y las relaciones entre conceptos" [4]. Las ontologías incluyen definiciones utilizables por máquinas, sobre conceptos básicos del dominio y de las relaciones existentes entre los mismos codifican conocimiento en un dominio y también conocimiento que se expande a través de varios dominios. De este modo, hacen que este conocimiento sea reutilizable. "El término ontología tiene su origen en la filosofía antigua que es percibida como originaria de la metafísica, y cuyo sentido lleva a la búsqueda de la esencia del ser (Platón, Aristóteles, Porfirio y sus seguidores)". [5] Conforme defiende Bentes Pinto (2006) [6] a partir del final del siglo XX, el concepto de ontología reaparece en el campo de las Ciencias Cognitivas con un nuevo significado relacionado con las informaciones divulgadas en la nube. La ontología tiene diferentes significados o aplicaciones; uno de ellos es la representación del conocimiento, redes semánticas y gestión de informaciones en el contexto de la web semántica [7]. Por otro lado en el campo de las Ciencias Cognitivas, la ontología constituye un modelo estructurado por nociones de dominio,

clases, subclases, propiedades, axiomas, individuos que poseen relaciones entre sí.

[7], Un patrón conceptual abarca un dominio de interés (sus conceptos y relaciones). En el campo de la informática, los patrones conceptuales deben transformarse de tal manera que puedan almacenarse en la memoria de los ordenadores y permitan aplicar algoritmos. [8] El propósito de las ontologías (originarias del campo de la Inteligencia Artificial) se apoya en proporcionar los modelos conceptuales almacenables y computables. En informática, una ontología designa la descripción de un dominio mediante un vocabulario común de representación. [9] Dentro de las herramientas informáticas, las ontologías generalmente se usan para especificar y comunicar el conocimiento en un dominio determinado. En la disciplina de los modelos de Información se le considera como: ün artefacto del software (o lenguaje formal) diseñado para un conjunto específico de usos y ambientes computacionales"[10] Los componentes de las ontologías según [4], [11] son:

Clases o conceptos: son las ideas básicas que se intentan formalizar. Las clases son la base de la descripción del conocimiento en las ontologías ya que describen los conceptos del dominio. Una clase es un conjunto de objetos (físicos, tareas, funciones, métodos, planes, estrategias, procesos de razonamiento, etc.). Desde el punto de vista de la lógica, los objetos de una clase se pueden describir especificando las propiedades que éstos deben satisfacer para pertenecer a esa clase. Una clase puede dividirse en subclases, las cuales representarán conceptos más específicos que la clase a la que pertenecen. Una clase cuyos componentes son clases, se denomina superclase o metaclase.

Instancias o individuos: se utilizan para representar objetos determinados de un concepto. Son miembros de una clase, que no pueden ser divididos sin perder su estructura y sus características funcionales.

Relaciones: representan la interacción y el enlace entre los conceptos del dominio. Suelen formar la taxonomía del dominio. Algunas de las relaciones que más se utilizan son: "subclase-de", "parte-de", çonectada-a".

Propiedades: describen las características o los atributos de conceptos o clases. Pueden adoptar diferentes tipos de valores. Las especificaciones, los rangos y las restricciones sobre estos valores se denominan características o facetas. Para una clase dada, las propiedades y las restricciones sobre ellos son heredadas por las subclases y las instancias de la clase Los axiomas, junto a la herencia de conceptos, permiten inferir el conocimiento oculto detrás de una taxonomía de conceptos.

Axiomas: especifican las definiciones de los términos en la ontología y las restricciones de sus interpretaciones. Son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. Los axiomas son necesarios para definir la semántica o el significado de los términos; permiten, junto con la herencia de conceptos, inferir conocimiento que no esté indicado explícitamente en la taxonomía definida por la ontología.

1) Lenguajes ontológicos: Las ontologías pueden ser definidas con base en estándares para lograr la definición de los meta-dato, entre otros RDF y OWL son los más difundidos, estos permiten transformar la información en una estructura global donde se puede compartir y reutilizar datos, documentos y otros recursos.

RDF según el W3C, el Resource Framework Description (RDF) es un lenguaje para representar recursos Web , apoyándose en ideas de Inteligencia Artificial, grafos conceptuales y de representación de conocimiento basado en lógica, entre otras y, cuyo objetivo radica en especificar datos en XML de manera estandarizada a fin de posibilitar interoperabilidad. [12] Según Dikstra RDF [9] es un lenguaje ontológico que permite construir jerarquías de conceptos y propiedades siendo suficiente para aplicaciones que sólo necesitan establecer un vocabulario básico. [13] Prefiere definir que si bien RDF es un modelo de representación de metadatos puede utilizarse como lenguaje general para la representación del conocimiento.

OWL es un mecanismo para "desarrollar temas o vocabularios específicos en los que asociar esos recursos. Lo que hace OWL es proporcionar un lenguaje para definir ontologías estructuradas que pueden ser utilizadas a través de diferentes sistemas. Las ontologías, que se encargan de definir los términos utilizados para describir y representar un área de conocimiento, son utilizadas por los usuarios, las bases de datos y las aplicaciones que necesitan compartir información específica, es decir, en un campo determinado como puede ser el de las finanzas, medicina, deporte, etc.

Para definir las reglas se ha usado SWRL (Semantic Web Rule Language), que se define como un lenguaje de expresión de reglas basado en OWL, permite escribir reglas expresadas como conceptos OWL proporcionando capacidades de razonamiento. Una regla SWRL contiene un antecedente para describir el cuerpo de la regla y un consecuente que se refiere a la cabeza de la misma, cada uno compuesto por un conjunto (puede ser vacío) de átomos. Tanto el cuerpo como la cabeza de la regla son conjunciones positivas de los átomos.

SWRL puede entender como significado que si todos los elementos en el antecedente son Verdaderos, entones el consecuente debe ser también cierto [14]. Las reglas SWRL están escritas en términos de clases, propiedades, instancias y valores de OWL. El desarrollo del proyecto implica el uso de dos métodos: El método analítico, en donde el proceso de análisis y extracción de información, permite armar el modelo ontológico y mediante el uso del lenguaje OWL y SWRL es posible definir las reglas necesarias para dar operatividad al modelo y como consecuente final aplicarlo a un indicadores ó variables de calidad. Se pretende extraer las partes que constituyen dicho proceso y poder plantear un modelo de parametrización inicial de la base de conocimiento, donde se incluyan los modelos cuantitativos con los que se realizan las mediciones en esta área.

El segundo método es la modelación, en tanto se crearán abstracciones que permitan explicar la realidad en la medida que un usuario responda ante actividades propuestas. Al modelar un sistema con el que interactúa el usuario, se posibilita un intermediario para la obtención implícita de datos para poder llevar a cabo el procesamiento debido y no operar de forma directa entre el usuario y la medición del proceso.

Finalmente, los beneficios de la incorporación de servicios de Web Semántica en aplicaciones web están bien documentados. Algunas de las barreras para la aplicación en el mundo real son el soporte a complejidades y herramientas relacionadas

con el desarrollo de Servicios Web Semántica. A continuación se argumenta desde el punto de vista de los servicios.

II-B. Arquitectura basada en Servicios

El objetivo de un servicio web, es construir un sistema extensible y distribuido bajo el estilo arquitectónico REST (Representacitional State Transfer), el cual consiste en un conjunto de restricciones que pueden ser aplicadas sobre cualquier otro sistema distribuido en diferentes medios. Para este caso limita la arquitectura a un sistema cliente/servidor con facilidades de código bajo demanda y con una interfaz equitativa a sus recursos. El protocolo HTTP proporciona una forma de acceder a los recursos mediante una interfaz común y que está basada en 4 métodos: GET, PUT, DELETE y POST. Esto implica que la manipulación de un recurso debe ser por medio de alguno de estos métodos.

II-C. Arquitectura basada en Microservicios

Un micro servicio, es definido como un estilo arquitectural que permite desarrollar una aplicación basada en un conjunto de pequeños servicios livianos [16]. Por lo general una aplicación, está compuesta para ser ejecutada como una sola tanto en su estructura de archivos, cómo en la base de datos y se compone de tres capas: capa presentación (interfaz en html), capa de aplicación del servidor (recibe las solicitudes, ejecuta la lógica de negocio) y capa de acceso a la base de datos. la ejecución de una aplicación con un único ejecutable lógico se denomina monolítica. Al descomponer una aplicación se pasa a hablar de microservicios, un ejemplo es visto en la figura 2, la construcción de una plataforma de comercio electrónico que recibe pedidos de los clientes, verifica el inventario y el crédito disponible y los envía. La aplicación consta de varios componentes y servicios para la comprobación de crédito en mantenimiento de inventario y el envío de pedidos.

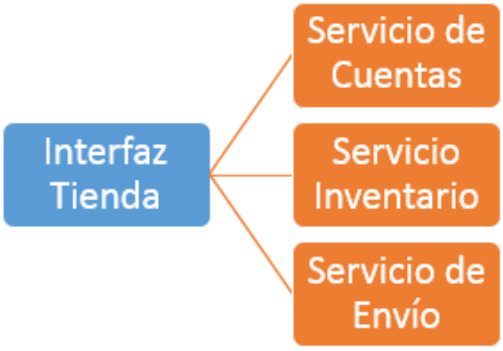

Figura 2: Ejemplo de Micro servicios, fuente: Autor

Al construir los servicios de manera que su ejecución sea independiente, es decir pueden estar en diferentes lenguajes de programación y diferentes bases de datos, se constituye una solución practica para evitar colapsar el sistema cuando tiene más demanda de la esperada. Un ejemplo claro es dado en la figura 3 donde, los microservicios son pequeños servicios independientes que trabajan coordinadamente juntos.

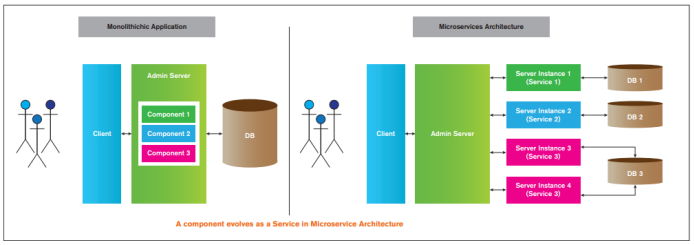

Figura 3: Comparación entre Arquitectura Monolítica y microservicios fuente: [16]

1) Descomponer una Aplicación en Servicios: Existen diferentes maneras de descomponer una aplicación en servicios, por ejemplo, iniciar sesión ó simplemente consultar los usuarios registrados, el programador desde su punto de vista objetivo puede lograr una escalabilidad infinita si se definen los límites de responsabilidad de cada servicio [17]. Una de las maneras para descomponer los servicios y convertirlos en micro servicios es:

1. Descomposición a partir de los casos de uso.
2. Descomposición a partir de la búsqueda de verbos y subcomponentes de ellos, por ejemplo gestión de clientes.

El modelo propuesto, se denomina modelo de escalabilidad y brinda un enfoque por medio del cubo de escala, el cual se describe en la figura 4 y contiene tres dimensiones (x,y,z). Cada dimensión se argumenta más adelante, básicamente en la dimensión del eje x se separan los servicios de la aplicación independiente de los clientes, el eje y busca ejecutarlos en diferentes servidores cómo copias, validando que un grupo de clientes tengan acceso a todos los servicios y en el eje z, se encarga de brindar un balanceo de carga de recursos a todos los clientes.

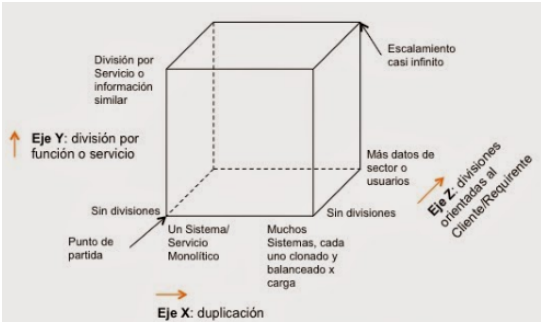

Figura 4: Modelo de Escalabilidad, fuente:[18]

En la **escala del eje x** se ejecutan múltiples copias de una aplicación al mismo tiempo, de manera que existe una capa encargada de balancear la carga de ejecución entre una y otra. De modo que si existen 10 copias, para ejecutar una equivale a 1/10, es decir que tiene derecho a consumir 0.1 de los recursos destinados a todas las copias. Una desventaja, es evidenciada en la memoria cache ya que requiere bastante espacio para ser eficaz ya que cada copia accede potencialmente a los datos.

En la **escala del eje y**, se ejecutan múltiples copias idénticas de la aplicación, de manera que divide la aplicación

en diferentes servicios y donde cada uno es responsable de una o más funciones relacionadas.

En la **escala del eje z**, cada servidor ejecuta una copia idéntica del código, es muy similar al eje x la diferencia se encuentra en que existen múltiples servidores y cada uno es responsable sólo de un subconjunto de datos, lo que indica una mejor utilización de caché y reducción del uso de memoria. De igual manera, mejora la escalabilidad de transacción dado que las solicitudes se distribuyen a través de múltiples servidores y en caso de fallar algo en la aplicación solo se suspenden los servicios afectados (es decir un solo servidor).

2) Mecanismos de comunicación en una arquitectura de Microservicios: Existen dos mecanismos de comunicación en un micro servicio, el primero entre los clientes y la aplicación y el segundo entre servicios de la misma aplicación cómo lo indica la figura 5.

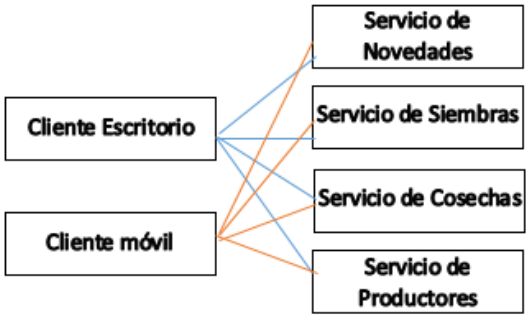

Figura 5: Llamada a Servicios Directamente, fuente: Autor

III. Metodología

Cómo se explico anteriormente, la idea es desacoplar la comunicación entre servicios, una herramienta que soporta alta disponibilidad a través de servicios redundantes es ZooKeeper [20] Este framework permite manejar un sistema de archivos donde los clientes pueden leer y escribir desde y hacía los nodos y también ayuda en el balanceo de carga. Para empezar con un ejercicio, se debe configurar 3 maquinas virtuales con el fin de que cada una asume el rol de nodo, como lo indica la figura 6, cada nodo cuenta con una estructura la cuál involucra, los datos almacenados por el usuario, la información de control de cambios y los hijos que tienen la misma descripción [20], ello se refleja en atributos cómo histórico de tiempos de creación y modificación, tamaño del nodo, identificador de cambios y de orden de ejecución, entre otros

III-A. Lista de Tareas de la Investigación

Durante el avance de la presente investigación, se contemplaron, tareas que paso a paso se fueron ejecutando, las herramientas utilizadas, se describen más adelante, a continuación se describen las siguientes tareas con su respectiva ejecución

Un procedimiento inicial es instalar y configurar el zk(zookeeper), cuya función, es la atención a los servicios redundantes y la distribución de la carga al momento de ejecutarlos. ZooKeeper proporciona sincronización distribuida,

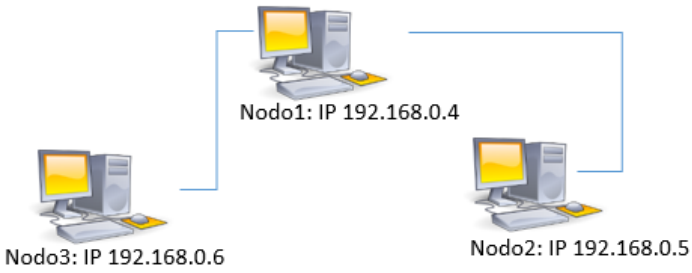

Figura 6: Nodos servicios a consumir en la aplicación, fuente: Autor

Tareas	Estado	Completado
1. Instalar y Configurar Zeekeeper	Completa	100 %
2. Estructuración y Descomposición acorde a los casos de uso	Completa	100 %
3. Requerimientos Software(Servidor WEB base de datos, extensión REST)	Completa	100 %
4. Creación de la base de datos	Completa	100 %
5. Script de conexión e interfaz	Incompleta	100 %
6. Agregar Solicitud cliente Avance Rest	completa	100 %
7. Script en PHP para obtener información del usuario	Completa	100 %
8. Script en PHP para obtener la condición de usuario de actualización	Completa	100 %
9. Ejecución final del servicio y microservicios	Completa	100 %

Tabla I: Tareas para la ejecución de la investigación

y la prestación de servicios en grupo, dichos servicios se utilizan de una forma u otra por las aplicaciones distribuidas. Debido a la dificultad para implementar una aplicación grande tanto en información cómo en consumo de recursos, una solución inicial es reducir en microservicios, lo que los hace frágiles con respecto a cambios o actualizaciones y de difícil manejo lo que conduce a la complejidad de gestión cuando se despliegan las aplicaciones.

Los procedimientos de configuración involucran el número de nodos, los puertos por cada uno, los tiempos de ejecución, el directorio de archivos temporales y de puntos de configuración, entre otros. Existe una consola de acceso a los nodos clientes y al servidor (balanceador de carga), cuando el servidor está activo detecta el ingreso de un cliente y éste esta disponible para la creación de nuevos nodos. Zookeeper permite la creación de nodos hijos, más no la eliminación en cascada, es decir no es posible eliminar un nodo que contenga nodos hijos.

III-B. Configuración de los Directorios

Un nodo puede ejecutar uno o varios micro servicios según la configuración establecida (de acuerdo a las capacidades hardware ó las dependencias necesarias). Cada micro servicio debe tener una página principal de acceso cómo lo indica la figura 7, hay de recordar que desde el punto de vista de los microservicios existe independencia entre sí, por lo tanto cada uno tiene una página de acceso único, pero todos enrutados con una dirección URL común seguido del puerto, por ejemplo localhost:81 y un estado para validar su uso, esta arquitectura es vista en la figura

Un ejemplo de aplicación puede empezar de tipo monolíti-

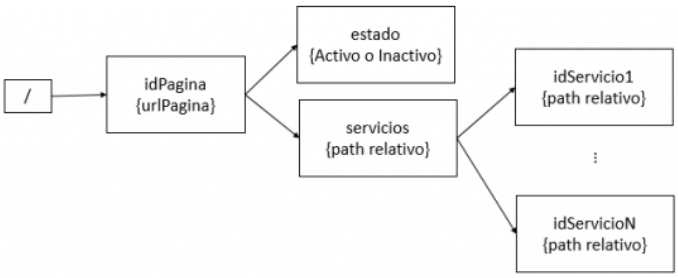

Figura 7: Configuración directorio zookeeper, fuente: [20]

ca para un caso aplicado a una finca, en el área de producción agrícola y que ofrece una cantidad limitada de productos y unas siembras reducidas. Dicha aplicación esta compuesta por diferentes componentes, una interfaz de usuario y varios servicios para gestionar la producción, actividades de mantenimiento a las siembras, productos. Cada servicio comparte el modelo de entidades tales como siembras, novedades, cosechas; La estructura para este ejemplo se visualiza en la figura 8, cómo anteriormente se explico cada nodo tiene una ip fija por lo que es necesario tener en red las maquinas virtuales y asignar una ip fija, con el fin de identificar la ubicación del servidor (balanceador de carga) y los clientes, para términos del ejemplo en mención se utilizó Vmware station y en el anexo *"Manual Paso a Paso para la Implementación de micro servicios"* se encuentran las indicaciones para manipular las maquinas en la misma red así cómo el acceso con diferentes servidores.

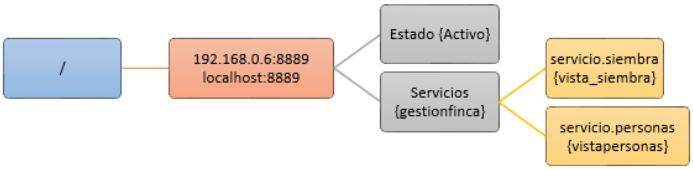

Figura 8: Estructura de servicios en un nodo, fuente: Autor

En esta situación, se habla de una implementación en entorno web, donde los archivos están alojados en un servidor WEB Apache, necesitan un navegador y el acceso a una base de datos, como lo indica la figura 9

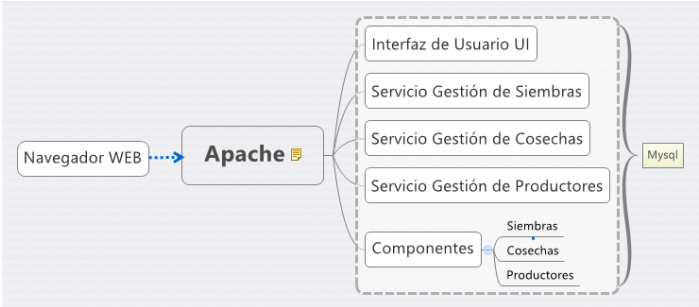

Figura 9: Descripción Arquitectura Monolítica, fuente: Autor

Si bien, una entidad más grande por ejemplo una asociación, desea gestionar un número indeterminado de fincas cada una con diferentes cultivos y cosechas en el año, resulta complejo realizar un despliegue de toda la aplicación para adaptar actualizaciones sin contar los recursos y el tiempo involucrados en ello. Una solución es construir los servicios de manera que su ejecución sea independiente, es decir pueden estar en diferentes lenguajes de programación y diferentes bases de datos.

1) Aplicación de Lineamientos de Descomposición: Se va a descomponer la aplicación de ejemplo de la finca en múltiples servicios pequeños con reducidas funciones. Es posible aplicar lineamientos de descomponer la aplicación monolítica según el **eje y** del cubo de escalabilidad visto previamente, es decir que cada servicio implementa un conjunto de funciones relacionadas, y a su vez, estos servicios se comunican entre sí a través de protocolos HTTP/REST, cómo lo indica la figura 10. Donde la aplicación se descompone en varios servicios de interfaces de usuarios y servicios back-end por ejemplo la interfaz siembra implementa la consulta y visualización de las siembras en una finca y el servicio Gestión novedades implementa la vista para realizar una actividad en una siembra ó para consultar las cosechas en el año. Finalmente se ha convertido cada uno de los principales componentes en un servicio independiente.

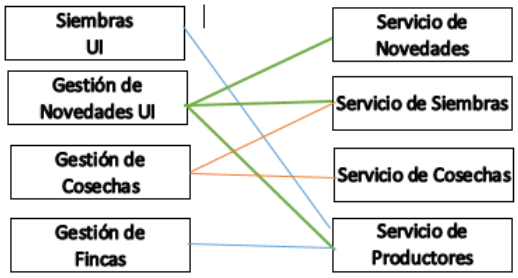

Figura 10: Modelo de Micro servicio Aplicado, fuente: Autor

III-C. Aplicación de los mecanismos de comunicación

La comunicación entre si de **Microservicios** se encuentra dada por una comunicación ínter-proceso por ejemplo HTTP, donde los servicios corren pro diferentes procesos y aveces en diferentes máquinas. La comunicación entre **clientes y Microservicios** se refiere a a solicitudes entre navegadores web y aplicaciones móviles, cada una dependiendo el servicio a necesitar hace una petición HTTP REST ful a través de un balanceador de carga cómo lo muestra la figura 11, el cual, corre en una ubicación determinada , consulta el registro e servicios y envía la solicitud a una instancia disponible del servicio.

ZooKeeper es una aplicación Java con scripts en C y es adaptable a varios lenguajes de programación, por ejemplo, hay una extensión de PHP creada y mantenida por Andrei Zmievski desde 2009, se puede descargar desde PECL u obtener directamente de GitHub PHP-ZooKeeper. El api en este caso contiene el código preestablecido para crear tanto nodos cómo hijos, de igual manera la administración

4. Creación de la base de datos

Se utilizó la herramienta xammp cuyos servicios ofrece un servidor apache, un servidor de base de datos (mysql) y

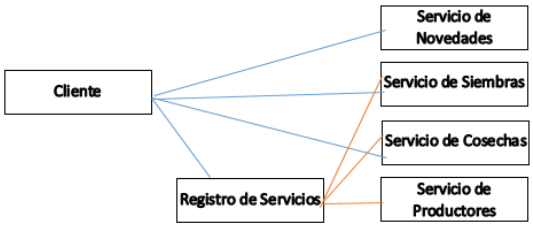

Figura 11: Llamada a Servicios Directamente, fuente: Autor

un interprete del lenguaje PHP, Por medio de phpmyadmin se crea la base de datos con una tabla usuarios cómo lo indica la figura 12, para poder conectar el servicio,

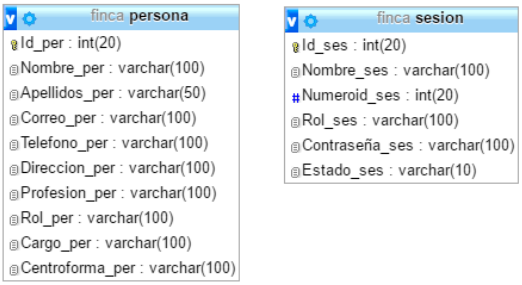

Figura 12: Llamada a Servicios Directamente, fuente: Autor

III-D. 5. Script de conexión e interfaz

Se debe crear un Script de conexión a la base de datos, ello debe contener la cadena de conexión como lo indica la figura 13 Organizar las carpetas en estructura jerárquica por cada servicio en este caso se identifica cada microservicio con una carpeta independiente : servicio.persona y servicio.sesion, es importante, la ubicación del proyecto debe estar en la carpeta httdocs del servidor apache.

```
//Código para conectar a la base de datos
$conexion=mysql_connect("localhost","root","");// Cadena de conexión usuario y contraseña
 mysql_query("SET NAMES 'utf8'");
 mysql_select_db("finca");//nombre de la base de datos
 return $conexion;
```

Figura 13: Llamada a Servicios Directamente, fuente: Autor

III-E. Ejecución del Servicio PHP

Para dicha ejecución, se necesita los siguientes Scripts: de conexión e interfaz, Solicitud cliente Avance Rest, Script en PHP para obtener información del usuario, en este caso los datos son envíados por la URL por el método POST, cómo lo indica la figura

III-F. Ejecución del Balanceo de Carga para cada Servicio PHP

El paso final es aplicar a cada servicio la asignación de nodos, la estructura un valor y una actualización con respecto a las maquinas ó host disponibles para acceder al servicio, en la configuración se crea la clase como lo indica la figura 15, en la

```
        include("persona.php");
        require_once("../claseconexion.php");
$objetoclase=new conectar;
$ejecutar=$objetoclase->conect();$pId_per=$_POST['idper'];
        $pNombre_per=$_POST['nombre'];
        $pApellidos_per=$_POST['apellido'];
        $pCorreo_per=$_POST['correo'];
        $pTelefono_per=$_POST['telefo'];
        $pDireccion_per=$_POST['direccion'];
        $pProfesion_per=$_POST['profesion'];
        $pRol_per=$_POST['rol'];
        $pCargo_per=$_POST['cargo'];
        $enviar=$_POST['enviar'];
```

Figura 14: Aplicación del método POST en el envío de datos para un servicio, fuente: Autor

estructura del código y bajo los principios de la programación orientada a objetos esta función declarada puede ser solicitada ó llamada más adelante por otro host. El balanceador de carga va registrando los accesos y administrándolos con el fin de brindar mayor rendimiento.

```
public function makePath(servicio.persona, $value = '1') {
    $parts = explode('/', servicio.persona);
    $parts = array_filter($parts);
    $subpath = '';
    while (count($parts) > 1) {
        $subpath .= '/' . array_shift($parts);
        if (!$this->zookeeper->exists($subpath)) {
            $this->makeNode($subpath, $value);
        }
    }
}
```

Figura 15: Estructura de un nodo nuevo asignado a un host, fuente: Autor

IV. Conclusión

- Es más fácil de entender para un desarrollador
- El sitio web se inicia más rápido, lo que hace que los desarrolladores sean más productivos, y acelera las implementaciones
- Es más fácil de escalar el desarrollo. Es posible organizar los recursos para el desarrollo de la aplicación en varios equipos. Cada equipo es responsable de un solo servicio. Cada equipo puede desarrollar, desplegar y escalar su servicio, independientemente de todos los otros equipos.
- Optimiza el aislamiento de fallos. es decir, si hay una pérdida de memoria en un solo servicio, entonces solamente este se verá afectado. Los otros servicios continuarán para manejar las peticiones. En comparación, una falla de un componente en una arquitectura monolítica puede derribar todo el sistema.
- Cada servicio puede ser desarrollado y desplegado de forma independiente

V. Lecciones Aprendidas

- Es más fácil de entender para un desarrollador

- El navegador se inicia más rápido, lo que hace que los desarrolladores sean más productivos, y acelera las implementaciones
- El uso de esta arquitectura, elimina cualquier compromiso a largo plazo a una tecnología
- Los desarrolladores deben hacer frente a la complejidad adicional de crear un sistema distribuido. o Herramientas para desarrolladores / IDE se orientan en la construcción de aplicaciones monolíticas y no proporcionan apoyo explícito para el desarrollo de aplicaciones distribuidas.
- La implementación de casos de uso que abarcan múltiples servicios sin el uso de transacciones distribuidas es difícil
- La implementación de casos de uso que abarcan múltiples servicios requiere una cuidadosa coordinación entre los equipos
- El aumento del consumo de memoria. La arquitectura microservicios sustituye a instancias de aplicación monolítica con N NxM instancias de servicios

REFERENCIAS

[1] Automatic Discovery of Semantic Manufacturing Grid Services,Date of Conference: 29-31 Oct. 2007, Page(s): 430 - 433, Conference Location: Shan Xi, 7, Publisher: IEEE.

[2] Herman I. 2010, Semantic Web Activity Statement, 2010.

[3] Wilman Vega, Henry Umaña, 2014, Diseño De Servicios Web Semánticos Utilizando El Desarrollo De Software Dirigido Por Modelos [Semantic Web Services Design Using Model-Driven Software Development]. Universidad de Manizales, Facultad de Ciencias e Ingeniería, Revista Ventana Informática.

[4] T. R. Gruber, A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition, ISSN 1042-8143, vol. 5, Nº 2, pp. 199-220, 1993.

[5] Flores Vitelli, 2011, Aplicación De Methontology Para La Construcción De Una Ontología En El Domino De La Microbiología, Centro de Ingeniería de software y Sistemas de la UCV, I.

[6] Virgínia Bentes Pinto, Departamento de Ciências da Informação. Universidade Federal do Ceará. Brasil. , Henvy de Holanda Campos, Departamento de Medicina Clínica. Universidade Federal do Ceará. Brasil. , Jefferson Leite Oliveira FerreiraEx-Bolsista do PIB

[7] Cruz, Luis Alejandro Riveros,2011, Aprendizaje automático y semi-automático de ontologías: una revisión, Universidad Nacional.

[8] Larburu I., Pikatza J., sobrado F., Garcia J. y López de Ipiña,2010, Hacia la Implementación de una Herramienta de soporte al proceso de desarrollo de Software, Departamento de Lenguajes y Sistemas Informáticos

[9] Norka Natalia Aguirre Helguero, 2011, Universidad de Argentina, Un Agente Basado En Un Razonador De Ontologías

[10] GUARINO, N., 2007, Formal Ontology and Information Systems, in Formal Ontology, Information Systems, Proceedings of FOIS'98, Trento, Italy, 6-8 June 1998.

[11] MCGUINNESS, D.L, 2002, Ontologies Come of Age - Fensel, Hendler, Lieberman and Wahlster (eds), Spinning the Semantic Web: Bringing the World Wide Web to its Full Potential, MIT Press, 2002. [consulta: 20 de marzo de 2007]. Disponible en Web: ¡http://www.ksl.stanford.edu/people/dlm/papers/ontologies-come-of-age-mitpress-(with-citation).htm¿

[12] Jesús M. Almendros Jiménez,2010, Using OWL and SWRL for the Semantic Analysis of XML Resource, Dpto. de Lenguajes y Computación, Universidad de Almería, Spain jalmen@ual.es

[13] Tartir, S.; B., Arpinar; Sheth, A., 2007, Ontological Evaluation and Validation, [consulta: 20 de marzo de 2007]

[14] Oconnor, Martin, Rabi Shankar And Other, 2008, Developing a Web-Based Application using OWL, Stanford Medical Informatics, Stanford University, Stanford, CA 94305-5479. 2008

[15] Oconnor, Martin, Rabi Shankar And Other, 2008, Developing a Web-Based Application using OWL, Stanford Medical Informatics, Stanford University, Stanford, CA 94305-5479. 2008

[16] Keshab Katuwal, 2011, Microservices: A Flexible Architecture for the Digital Age, Architecture A White Paper Series

[17] Martin Abbott y Michael Fisher, 2009, "The Art of Scalability".

[18] Martin Flowler,2014, "Microservices Resource Guide", http://martinfowler.com/microservices/

[19] Chris Richardson,2014, A pattern language for microservices, http://microservices.io/patterns/index.html

[20] Universidad de los Andes, Laboratorio Zoo-keeper, Arquitectura y Diseño de Software, https://sistemasacademico.uniandes.edu.co/ isis2503/dokuwiki/doku.php?id=laboratori