TESIS EPG UNA - PUNO

Universidad Nacional del Altiplano

UNIVERSIDAD NACIONAL DEL ALTIPLANO

ESCUELA DE POSGRADO

PROGRAMA DE MAESTRÍA

MAESTRÍA EN INFORMÁTICA

TESIS

ARQUITECTURA PARA EL DESARROLLO E IMPLEMENTACIÓN DE SERVICIOS WEB

PRESENTADA POR:

FREDY ABEL HUANCA TORRES

PARA OPTAR EL GRADO ACADÉMICO DE:

MAGISTER SCIENTIAE EN INFORMÁTICA
MENCIÓN EN GERENCIA DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIONES

PUNO, PERÚ

2017

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

TESIS EPG UNA - PUNO

Universidad Nacional del Altiplano

UNIVERSIDAD NACIONAL DEL ALTIPLANO

ESCUELA DE POSGRADO

PROGRAMA DE MAESTRÍA

MAESTRÍA EN INFORMÁTICA

TESIS

ARQUITECTURA PARA EL DESARROLLO E IMPLEMENTACIÓN DE SERVICIOS WEB

PRESENTADA POR:

FREDY ABEL HUANCA TORRES

PARA OPTAR EL GRADO ACADÉMICO DE:

MAGISTER SCIENTIAE EN INFORMÁTICA
MENCIÓN EN GERENCIA DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIONES

APROBADA POR EL SIGUIENTE JURADO:

PRESIDENTE

Dra. MARIA MAURA SALAS PILCO

PRIMER MIEMBRO

D.Sc. ALEJANDRO APAZA TARQUI

SEGUNDO MIEMBRO

M.Sc. ROBERTO ELVIS ROQUE CLAROS

ASESOR DE TESIS

M.Sc. SAMUEL DONATO PEREZ QUISPE

Puno, 25 de agosto de 2017

ÁREA: Ingeniería de software.
TEMA: Arquitectura de desarrollo Web.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

TESIS EPG UNA - PUNO

Universidad Nacional del Altiplano

DEDICATORIA

Con mucho afecto para mis seres queridos:

Mis padres Pácido Huanca Mamani y Teresa Torres Sanchez,

mis hermanos Efraín Arturo Huanca Torres y

en memoria de Eudocia Huanca Torres

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

AGRADECIMIENTOS

- A Dios en primer lugar, por conducir mi vida.
- A mi familia por sus consejos y confianza puesta en mí.
- A mi asesor y a todas aquellas personas quienes contribuyeron con sus aportes relevantes en el desarrollo de la investigación.
- Al proyecto: Plataforma digital inteligente y Bigdata para el turismo rural comunitario en la región Puno, aprobado mediante Resolución de Dirección Ejecutiva Nº 144-2015-FONDECYT-DE.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

ÍNDICE GENERAL

Pág.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

METODOLOGÍA

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

ÍNDICE DE CUADROS

Pág.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

ÍNDICE DE FIGURAS

Pág.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

ÍNDICE DE ANEXOS

Pág.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

RESUMEN

En este trabajo se identificaron necesidades en el proceso de desarrollo de software e implementación de servicios web. En el desarrollo de aplicaciones web se están imponiendo estilos arquitectónicos, y las nuevas tendencias en tecnología como: Microservicios, Plataforma como servicio, entornos aislados de desarrollo, entregas continuas, despliegue, ejecución en cualquier ambiente y su implementación son hoy en día una necesidad, lo cual se convierte en un requerimiento de en los proyectos de desarrollo de software. Razón por la cual surge la siguiente interrogante: ¿Cómo diseñar una arquitectura para el desarrollo de aplicaciones web y su posterior implementación, que permita escalabilidad y reproductibilidad de forma simple y rápida, utilizando tecnologias emergentes?; El objetivo general fue construir un ambiente de desarrollo para aplicaciones web con estilo arquitectónico basado en Microservicios, para ello implementaremos prototipos de entornos aislados, bajo las tecnologías que proporcionan los contenedores, de esta manera las aplicaciones se ejecutarán de forma local. Así mismo se implementaron estos servicios en la nube, específicamente en una instancia de Amazon EC2. Esta propuesta brinda ventajas, desde la escalabilidad independiente de cada aplicación empaquetados en contenedores hasta el ahorro en consumo de recursos de los servidores.

Palabras claves: Amazon EC2, arquitectura microservicios, contenedores, despligue, Docker, virtualización.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

ABSTRACT

In this work, needs were identified in the process of software development and implementation of web services. In the development of web applications, architectural styles are being imposed, and new trends in technology such as: Microservices, Platform as a service, isolated development environments, continuous delivery, deployment, execution in any environment and their implementation are nowadays a necessity, which becomes a requirement of software development projects. Which is why the following question arises: How to design an architecture for the development of web applications and their subsequent implementation, which allows scalability and reproducibility in a simple and fast way, using emerging technologies? The general objective was to build a development environment for web applications with architectural style based on Microservices, for this we will implement prototypes of isolated environments, under the technologies provided by the containers, in this way the applications will be executed locally. We will also implement these services in the cloud, specifically in an Amazon EC2 instance. This proposal offers advantages, from the scalability independent of each application packaged in containers to the saving in consumption of server resources.

Keywords: Amazon EC2, containers, deploy, Docker, microservices architecture, virtualization.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

INTRODUCCIÓN

Los equipos de desarrollo de software requieren construir servicios web de forma tal que no solo satisfagan requerimiento funcionales, sino también requerimientos no funcionales, como por ejemplo atributos de calidad: escalabilidad, portabilidad, mantenimiento, lo que requiere replantear la forma clásica en que se desarrolla software, que va desde el ecosistema del ambiente de desarrollo hasta la puesta en producción, es por ello que se requiere la implementación de tecnologías emergentes, como la adopción de la arquitectura de microservicios y las tecnologías de contenedores, tanto para desarrollo como para producción y despliegue en la nube.

La investigación corresponde a la disciplina de Ingeniería de software y al área de conocimiento Diseño de software (arquitectónico).

La investigación ha sido estructurada en cuatro capítulos los cuales se organizan de la siguiente manera:

En el Capítulo I. Problemática de la investigación: abarca el planteamiento, formulación del problema, justificación y los objetivos, alcance limitaciones de investigación.

En el Capítulo II. Marco teórico: se presenta la revisión bibliográfica de los antecedentes de la investigación y marco referencial.

En el Capítulo III. Metodología: se define el tipo de investigación, metodologías teóricas, también se presenta los materiales tecnológicos empleados.

En el Capítulo IV. Resultados y discusión: se exponen los resultados del modelo de arquitectura para el desarrollo e implementación de servicios web.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

CAPÍTULO I

PROBLEMÁTICA DE INVESTIGACIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

Los entornos de desarrollo de aplicaciones web, están basados en la combinación de herramientas que son relativamente heterogéneos en la mayoría de equipos de desarrollo de software, requiriendo instalar dependencias y bibliotecas de forma aislada. Poder replicarlo, requiere invertir tiempo y esfuerzo a cada uno de sus miembros, y la manera en que son configurados no permiten la reproductibilidad de estos entornos. Cuantas veces nos hemos enfrentado al problema de que un código fuente se ejecuta correctamente en el computador del desarrollador, pero ese mismo código no funciona en un entorno de prueba y/o producción, se requiere contar una solución para usar una plataforma que permita crear un entorno adecuado y reproducible, sobre todo basado en una arquitectura Microservicios.

Cuando nos referimos al desarrollo de software, no solo debemos enfocarnos en el código fuente, más es toda una pila de software, que consiste en un grupo de frontend, backend, componentes de base de datos, librerías y dependencias; también tenemos que asegurar que todos estos componentes trabajen en otros entornos con plataformas muy diferentes, como por ejemplo en la nube.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Por otro lado, la mayoría de aplicaciones web que se encuentran en producción, residen en servidores que han sido configurados cumpliendo uno a uno los requerimientos de dependencias, lo cual también requiere tiempo y esfuerzo; agregando además que no permite escalabilidad para asegurar la persistencia del servicio; debido a que han sido desplegados de forma tradicional, siguiendo arquitecturas monolíticas; ejecutados como un único servicio, que si bien es cierto puede ser configurado en un clúster para asegurar la disponibilidad, requiere una inversión en recursos e infraestructura; Si requiriéramos desplegar un servicio web basado en tendencias actuales, se requeriría de infraestructura tecnológica que lo soporte.

Por lo descrito anteriormente se formuló el siguiente problema:

¿Cómo diseñar una arquitectura para el desarrollo de software e implementación de servicios web, que permita escalabilidad, reproductibilidad y portabilidad de forma simple y rápida, utilizando tecnología emergente?

1.2 JUSTIFICACIÓN

Los entornos de desarrollo, despliegue e implementación de servicios web basado en arquitectura microservicio, requiere contar con un modelo de referencia, ya que nos encontramos en una transición de desarrollar y ejecutar sistemas tradicionales (monolíticos) a sistemas con arquitectura de microservicios. Será relevante su implementación en la Plataforma digital inteligente y big data para el turismo rural comunitario de la región Puno y áreas de desarrollo de software de la región, ya que no poseen una arquitectura lo cual afecta en la reproductibilidad y escalabilidad de los servicios web, requiriendo tiempos prolongados para su implementación, así mismo permita representar la

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

estructura y todas las características de distintas aplicaciones basadas servicios web actuales, como la referida arquitectura.

La propuesta planteada en la investigación, es viable ya que se utilizará herramientas y sistemas operativos open source, para que sirva como referencia de creación de ambientes de desarrollo e implementación de aplicaciones web basados en Microservicios, tanto a nivel académico, así como a los departamentos - industrias de desarrollo de software web.

1.3 OBJETIVOS

1.3.1 Objetivo General

Proponer una arquitectura para el desarrollo e implementación de servicios web.

1.3.2 Objetivos Específicos

- Proponer el modelo arquitectonico para el desarrollo de Aplicaciones Web, basado en microservicios.
- Definir una Arquitectura para la implementación y despligue de aplicaciones web, mediante la tecnología de contenedores.
- Evaluar la arquitectura mediante la ejecución de pruebas unitarias, cobertura - calidad de código, y pruebas de integración dentro de contenedores de servicios web.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

1.4 ALCANCES Y LIMITACIONES

1.4.1 Alcances

La arquitectura esta propuesta para ambientes de desarrollo y despliegue de aplicaciones web, que involucran computadores de escritorio y e instancias EC2 de Amazon WS de la cloud computing, enfocado a los equipos de desarrollo y a profesionales de tecnologías de información que están encargados del despliegue de los servicios web.

1.4.2 Limitaciones

La presente investigación se limitó al ámbito tecnológico, en los ambientes desarrollo, implementación y despliegue, tanto a nivel local como en la nube. Ejecutados en arquitecturas de hardware computacional de 64 bits.

Para la prueba de las arquitecturas propuestas, se han utilizado en sistemas operativos basados en GNU/Linux y MacOS.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

CAPÍTULO II

MARCO TEÓRICO

2.1 ANTECEDENTES

Se presenta los antecedentes de estudios en trabajos de tesis y artículos científicos a nivel nacional e internacional.

Cito, Ferme y Gall (2016) plantean la reproducibilidad como repetibilidad de un proceso determinado para establecer un hecho o las condiciones bajo las cuales podemos observar el mismo hecho. La necesidad y capacidad de replicar y reproducir resultados científicos se ha convertido en un tema cada vez más importante para muchas disciplinas académicas. En Ciencia de la Computación y, más específicamente, en la ingeniería de software y web (SE/WE), las contribuciones del trabajo científico se basan en algoritmos desarrollados, herramientas y prototipos, evaluaciones cuantitativas y otros análisis computacionales. Concluye discutiendo sobre las ventajas, desafíos y limitaciones del uso de contenedores para permitir la reproducibilidad en la investigación SE / WE.

Duarte (2016) propone una arquitectura para crear servicios web con las características y cualidades necesarias mediante la combinación de componentes de varias tecnologías. Selecciona las tecnologías más recientes

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

para el desarrollo e implementación de un servicio web completo siguiendo la arquitectura propuesta. Aplicar la arquitectura propuesta mediante la tecnología seleccionada como parte de la solución en la transformación digital de un negocio; y concluye que: Actualmente existen una gran cantidad de aplicaciones de consola, escritorio, web y móviles que aún no permiten intercambiar información. Es importante que los sistemas actuales expongan un API para comunicarse entre si, evitando, por ejemplo, tener una conexión directa a un motor de base de datos, distribuir archivos planos. Utilizar un servicio web con las características del servicio web completo presentado, garantiza evitar riesgos de robo de información sensible y disminuir las vulnerabilidades del sistema. La arquitectura propuesta en la metodología no solamente se podrá implementar con las tecnologías seleccionadas, también se podrán utilizar otras como las de Oracle, por lo tanto, se invita a que prueben descubrir marcos de trabajo y componentes que faciliten su desarrollo e implementación. Se lograron reducir drásticamente los tiempos de desarrollo de implementación del servicio web completo, incluso reducir los tiempos de respuesta a las solicitudes. Utilizar el modelo o patrón RPC de RabbitMQ tiene como consecuencia añadir una complejidad innecesaria a la depuración. En vez de simplificar el software, RPC, puede resultar en código espagueti difícil de mantener. Utilizar un lenguaje de modelamiento es útil para realizar un análisis de cada uno de los elementos que conforman una organización, modelo que luego me permitirá llegar a cualquiera de sus capas, en este caso, la capa de aplicación y desde allí realizar una implementación casi que, automatizada de acuerdo a las necesidades o problemas detectados o a resolver, en una organización. En la arquitectura empresarial en las fases de arquitectura de negocio, aplicación e infraestructura

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

no hay que llevar el detalle a un nivel algorítmico, se tiende a confundir el proceso con el algoritmo. Con Archimate no se dejan vacíos en la descripción de componentes, sin necesidad de lanzar código o instrumentación. La documentación es síntoma de no saber escribir, por eso a nivel de código hay que utilizar descripciones en lenguaje natural para poder identificar muy rápido los componentes y que el mismo código de programación sea tan descriptivo como sea posible. Es importante cada vez que se adquiera un nuevo conocimiento apropiarse del discurso propio del mismo. El desarrollo de la arquitectura empresarial con Archimate puede llevarse a cabo comprometiéndose con cosas pequeñas, progresar y hacer entregas (metodologías de desarrollo rápido, ejemplo: Scrum), esto mejora la métrica comenzando a ver evidencias de entregables, así sean pequeños. Un diagrama solamente modela un área del conocimiento, un punto de vista de Archimate le permite a uno modelar diferentes áreas del conocimiento.

Urra (2016) presenta un scraper de datos distribuidos diseñado bajo un estilo arquitectónico distribuido llamado arquitectura de microservicio implementada con tecnologías Docker, diseñado bajo un clúster Docker Swarm que implementa y distribuye automáticamente scraper de datos y otros servicios involucrados entre los diferentes nodos del clúster, donde cada servicio trabaja independientemente para proporcionar colecciones de información altamente disponibles a un punto final para ser consumido por y la aplicación. Y concluye diciendo que: el resultado del estudio, desarrollo, implementación y despliegue de un sistema distribuido con arquitectura de microservicio utilizando Docker ha sido un gran desafío. No sólo por el potencial que el sistema puede ofrecer y por las ventajas que la arquitectura de microservicios y la oferta de Docker, por la

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

razón de que la arquitectura de microservicios y Docker van de la mano y ambos son las nuevas tendencias para la computación en nube y para la creación de aplicaciones distribuidas de alta disponibilidad.

Todea (2016) diseña e implementa un sistema para la entrega continua de aplicaciones web basado en contenedores Docker, el objetivo que se plantearon fue reducir el tiempo para desplegar las aplicaciones de la empresa BiiT Sourcing Solutions en los entornos de producción. Todo ello teniendo garantías plenas de que el despliegue de software cumple con unos mínimos estándares de calidad. Para ello se ha implantado un sistema de entrega continua, donde una vez que los desarrolladores han terminado una nueva versión del producto, se realicen todas las pruebas necesarias automáticamente que aseguran el correcto funcionamiento del software. Además, la solución está basada en virtualización mediante contenedores Docker para así asegurar su portabilidad y optimizar los recursos de la organización. Concluye que los problemas planteados fueron solucionados y que ello ha supuesto cambios en la estructura organizativa, política de testeo y arquitectura de los servidores. Además de las mejoras en los tiempos de despliegue, también se ha trabajado en la virtualización de toda la infraestructura necesaria para el desarrollo, basándose en contenedores Docker. Mejorando así la portabilidad de la solución, así como la gestión de recursos dentro del entorno de desarrollo.

Killalea (2016) plantea los dividendos ocultos de los microservicios (The Hidden Dividends of Microservices), tuvo como objetivo identificar algunos de los dividendos ocultos de Microservicios que los ejecutores deben hacer un esfuerzo consciente para cosechar. El más fundamental de los beneficios que impulsan los Microservicios es la clara separación de las preocupaciones, centrando la

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

atención de cada servicio en un aspecto bien definido de la aplicación general; llegando a la conclusión de que los Microservicios no son para todas las empresas, y el camino no es fácil. A veces la discusión sobre su adopción ha sido efusiva, centrándose en la autonomía, la agilidad, la resistencia y la productividad de los desarrolladores. Los beneficios no terminan ahí, sin embargo, y para lograr que en trayecto se tenga resultados, es importante cosechar los dividendos adicionales.

Safina, Mazzara y Montesi (2015) propone la posibilidad de expresar opciones a nivel de tipos de datos, una característica bien representada en estándares para Servicios Web, por ejemplo, WSDL (Web Services Description Language); llegando a la conclusión de que Jolie es un lenguaje de programación integral basado en el paradigma orientado al servicio, que surgió en el contexto de un esfuerzo de investigación extendido dirigido a formalizar la Informática Orientada a Servicios sobre los modelos ampliamente aceptados de concurrencia. Debido a su apoyo a la creación de prototipos rápidos de servicios sencillos y la coordinación de servicios complejos, Jolie se ha utilizado en el desarrollo de otros proyectos de investigación relacionados con la programación y el despliegue de servicios. Sin embargo, se ha identificado cómo el idioma aún carece de tipos de datos y operaciones capaces de enriquecer la sintaxis y añadir flexibilidad adicional al programa de escenarios SOA comunes.

Redhat (2016) propone como conseguir una arquitectura de microservicios exitosa: señalando que, La arquitectura de Microservicios es un nuevo estilo arquitectónico para crear servicios de bajo acoplamiento, al igual que autónomos. Las nuevas tendencias tecnológicas, como DevOps, la plataforma como servicio (PaaS), contenedores, así como los métodos de integración al

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

igual que distribución continua (CI/CD), permiten a las organizaciones crear y gestionar estos sistemas modulares a una escala que supera los enfoques anteriores, como la arquitectura orientada al servicio (SOA). Sin embargo, las organizaciones que refactorizan las aplicaciones monolíticas a Microservicios experimentan diversos grados de éxito. La clave para utilizar el estilo arquitectónico de Microservicios con eficacia es disponer de unos conocimientos amplios acerca de cómo y por qué las organizaciones deben usar el estilo señalado, para la creación de aplicaciones; La conclusión a la que llegan es que la arquitectura de Microservicios puede ofrecer variedad de ventajas a las organizaciones: desde escalabilidad independiente de diversos componentes de aplicaciones, hasta un desarrollo y un mantenimiento del software más rápidos y sencillos. Sin embargo, la arquitectura Microservicios no siempre beneficia a todos los equipos o proyectos, y puede ocurrir que la inversión sea considerable y sin un retorno inmediato. Concluyen también que la transición hacia los Microservicios debe ser un proceso gradual, y la refactorización de ciertas partes de las aplicaciones existentes es decir parcial (sin llegar a una transición total) también puede reportar beneficios. Para el éxito del desarrollo de aplicaciones basadas en Microservicios, es que las organizaciones deben crear primero una aplicación bien diseñada de acuerdo con los estándares de la plataforma existente y, a continuación, refactorizar la aplicación en un conjunto de Microservicios en función de las necesidades de las reglas de negocio. Con las personas, las herramientas y los procesos adecuados, los Microservicios pueden acelerar el desarrollo y la implementación, facilitar el mantenimiento, mejora la escalabilidad y ofrecer mayor libertad frente a la elección de variedad de tecnología a largo plazo.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Dragoni et al. (2017) manifiesta que los principales lenguajes para el desarrollo de aplicaciones de servidor, como Java, C/C++ y Python, proporcionan abstracciones para descomponer la complejidad de los programas en módulos. Sin embargo, estos lenguajes están diseñados para la creación de artefactos ejecutables únicos, también llamados monolitos, y sus abstracciones de modularización se basan en el intercambio de recursos de la misma máquina (memoria, bases de datos, archivos). Puesto que los módulos de un monolito dependen de dichos recursos compartidos, no son ejecutables independientemente. Concluyen que han preferido proporcionar una presentación evolutiva para ayudar al lector a comprender las principales motivaciones que conducen a las características distintivas de los Microservicios y se relacionan con paradigmas bien establecidos como OO y SOA.

Amaral et al. (2015) realizan la evaluación de desempeño de la arquitectura microservicios utilizando contenedores, manifestando que las máquinas virtuales son un componente básico ampliamente utilizado para la gestión y despliegue de la carga de trabajo. Son muy utilizados tanto en entornos de centros de datos tradicionales como en nubes (nubes privadas, públicas e híbridas). El término comúnmente utilizado máquina virtual (VM) se refiere a la virtualización del servidor, que se puede lograr a través de virtualización completa o paravirtualización. En los últimos meses, hubo un resurgimiento del interés en las tecnologías de contenedores, que proporciona un mecanismo más ligero: la virtualización del sistema operativo. Los contenedores son ligeros y rápidos - un solo servidor x86 puede tener razonablemente cientos de contenedores en ejecución (la memoria suele ser el recurso escaso); Además, los contenedores comienzan muy rápidamente - en 1 a 2 segundos en la mayor parte de los casos.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Muchas razones hay para este resurgimiento, pero desde el punto de vista técnico, dos de las razones más importantes son las mejoras en el apoyo de los nombres en el Kernel Linux, están disponibles en las distribuciones más usadas, y una aplicación específica de contenedores - Docker - ha creado con éxito un atractivo formato de embalaje, herramientas útiles y ecosistemas diversos; y concluyen indicando que los contenedores están ganando impulso porque ofrecen capacidades ligeras de virtualización de SO. Se utilizan comúnmente para alojar procesos único y aislados en el sistema. Por lo que notamos ofrece ventajas en su ejecución (ligereza y rendimiento), sin embargo, es importante señalar desde la perspectiva de gestión de la infraestructura muestran limitaciones. Por otro lado, la Virtualización de Servidores ha sido ampliamente adoptada en sectores e industrias, ya que proporciona mecanismos sencillos para administrar la infraestructura y agrupar procesos y aplicaciones. Sin embargo, introduce varias sanciones significativas en términos de tiempo de despliegue, consumo de memoria y gastos generales de procesamiento que varían con la naturaleza de las aplicaciones que alojan.

Levcovitz, Terra y Valente (2016) elaboran una técnica para extraer microservicios de sistemas empresariales monolíticos, manifestando que los sistemas monolíticos inevitablemente se hacen muy grandes a lo largo del tiempo, desviándose de su arquitectura pretendida y volviéndose difíciles, arriesgados y costosos de evolucionar. A pesar de estos problemas, los sistemas empresariales a menudo adoptan estilos arquitectónicos monolíticos. Por lo tanto, un gran desafío hoy en día en el desarrollo de software de la empresa es evolucionar el sistema monolítico en los tiempos apretados, presupuesto, manteniendo la calidad, disponibilidad y fiabilidad; Concluyen con la descripción

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

de una técnica para identificar Microservicios en sistemas monolíticos. Aplicando con éxito la técnica propuesta en un sistema bancario monolítico del mundo real, lo que demostró la factibilidad de nuestra técnica para identificar candidatos de sistemas monolíticos a Microservicios. Los procesos para cada subsistema ayudan considerablemente a evaluar las funciones, identificar y describir Microservicios. Sin embargo, hay subsistemas que no fueron clasificados como candidatos para migrar a Microservicios. Los escenarios que encontraron requerirían un considerable esfuerzo adicional para migrar el subsistema a un conjunto de Microservicios. Ejemplo: subsistemas que comparten la misma tabla de base de datos; Microservicio que representa una operación que siempre está en medio de otra operación; y transacciones comerciales que implican más de un subsistema de negocios, Transferencia de dinero de una cuenta de cheques a una cuenta de ahorro). Más importante aún, la migración a la arquitectura de Microservicio puede hacerse de forma incremental. En otras palabras, podemos beneficiarnos de la arquitectura de los Microservicios, por ejemplo, los servicios que se desarrollan y despliegan independientemente, y la independencia de la tecnología, sin migrar todo el sistema a Microservicios. Ambos tipos de arquitectura de sistemas, monolíticos y Microservicios, pueden coexistir en una solución de sistema. De hecho, un desafío en el uso de Microservicios es decidir cuándo tiene sentido usarlo, que es exactamente el objetivo final de la técnica propuesta en este artículo.

2.2 MARCO REFERENCIAL

2.2.1 Arquitectura de software

Según Clements (1996) la Arquitectura de Software es, una vista del sistema la cual incluye componentes principales del mismo, la conducta de

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

esos componentes según se percibe desde el resto del sistema y las maneras en que los componentes interactúan unos con otros, al igual que como coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el nivel más alto de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.

2.2.2 Servicios Web

El Servicio Web (Web Service, WS) es un término que es muy utilizado actualmente, aunque tiene ligeramente diferente significado. Más por el contrario, los conceptos y tecnologías implícitas siendo en gran medida, independientes de la forma que pueden ser interpretados (Alonso, Casati, Kuno y Micharaju, 2004).

La definición realizada por Booth, Haas, McCable, Newcomer, Ferris y Orchard (2004), tiene relevancia por que son similares al glosario del consorcio de World Wide Web (W3C), en el que define que un Web Service es un software diseñado para soportar interacciones interopables de maquina a maquina, sobre una red de comunicaciones.

Un Servicio Web tiene una interfaz (API), descrita en un formato procesable por computadores, que es generalmente Web Service Description Language (WSDL). Existe la posibilidad que otros sistemas pueden interactuar con el Servicio web utilizando mensajes SOAP, generalmente transmitidos mediante el uso del protocolo HTTP, utilizando en el cuerpo un standard en el lenguaje de Etiquetado Extensible (XML) en conjunción

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

con otros estándares y/o formatos de intercambio relacionados con la Web (por ejemplo Json).

Otra defínción, que consideramos más apropiada es definida por Papazoglou (2008), indica que los Servicios web integran un conjunto de protocolos para el intercambio de mensajes entre aplicaciones desarrolladas en diversos lenguajes de programación con la característica de ser ejecutadas en entorno heterogéneos.

La definición anterior se ajusta al aporte que realiza Docker y que abordamos en el presente trabajo, ya que el sin el uso de contenedores las aplicaciones web no son necesariamente ejecutables en cualquier plataforma, ya requeriría un esfuerzo y tiempo para migrar de un computador a otro.

Según Cabrera et al. (2010) los servicios web se han convertido en una tecnología de referencia para la implementación de software de todo tipo. Este éxito se ha traducido en la proliferación de WS, de manera que para una funcionalidad determinada pueden encontrarse una gran cantidad de WS.

2.2.2.1 Modelo de Web Service

La arquitectura de Web Services clásicamente se basada en tres roles de interacciones: proveedor, registro y consumidor de servicios respectivamente. Su uso e interacción involucra las operaciones de publicación, búsqueda y vinculación de servicios. En conjunto, estos roles y operaciones actúan sobre los artefactos y módulo de Software de los Web Services (Kreger, 2001)

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

En la Figura 1 (San José, 2016) se aprecia, el clásico escenario, donde el proveedor de servicios recibe un módulo accesible desde la red, definiendo una descripción del servicio para dicho WS y lo publica en un registro de servicios. El consumidor de servicios ejecuta una operación de búsqueda para recuperar la descripción del servicio, a nivel local o desde el registro de servicios web, por lo que utiliza la descripción (del WS) para enlazar con el proveedor de servicios para finalmente interactuar con la implementación de WS. Los roles de proveedor y solicitante de servicios son construcciones lógicas y un servicio puede presentar características de ambos roles. (San José, 2016).

En la figura 1 se aprecia la tecnología de Web Service tiene como base los estándares principales: Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP) y Universal Description Discovery an Integration (UDDI). Según (Booth, Haas, McCabe, Newcomer, Ferris y Orchard).

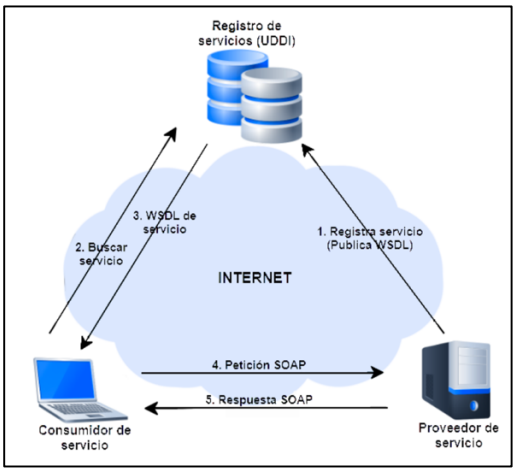

Figura 1. Arquitectura típica de un servicio web

Fuente: (San José, 2016)

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

El lenguaje de descripción de servicios web (WSDL) se basa en XML que permite describir las características de los WS, sin embargo actualmente se han vuleto muy popylar el uso del formato Json; SOAP es un protocolo estándar que define cómo dos objetos localizados remotamente logran comunicarse por medio de intercambio de datos mediante XML y que funciona sobre HTTP, teniendo también enlaces con otros protocolos de internet; y UDDI es un registro que proporciona mecanismos estándar para publicar documentos WSDL, que contiene descripciones de WS y realizan búsqueda sobre los mismos. (San José, 2016).

Para que una aplicación pueda tomar ventaja del uso de los WS, existen tres operaciones que se pueden desarrollar en una arquitectura de WS. (Kreger, 2001).

- **Registro de servicios.** Para que un consumidor de servicios pueda encontrar un servicio, la descripción del servicio que quiere utilizar tiene que estar publicada en algún lugar. Los WS se publican en el lenguaje WSDL. Para ello utiliza un registro UDDI que es el encargado de registrar el WS para su posterior utilización. Esto debe ocurrir para todos los WS de cualquier organización que desee que sus servicios sean accesibles globalmente.

  Una vez que están los WS registrados, los consumidores de servicios pueden buscarlos para ejecutarlos y obtener así sus objetivos.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

- **Proveedor de servicio.** En la operación de búsqueda, el consumidor de servicios recupera una descripción directa o consulta el registro de servicios para el tipo de servicio requerido.

  La operación de búsqueda puede involucrar dos fases al consumidor de servicios: en tiempo de diseño, con el objetivo de recuperar la descripción de la interfaz del servicio para el correcto desarrollo del programa, y en tiempo de ejecución, para obtener la descripción del servicio de enlace y la ubicación para dicha invocación.

  El proceso de búsqueda de servicios lo realiza el propio archivo UDDI, que recibe como entrada la funcionalidad del servicio y debe devolver todos aquellos servicios que cumplen dicha funcionalidad. El registro de servicios devuelve al cliente una lista con los archivos de descripción de los servicios, en WDSL, cuyas capacidades cubran las necesidades del cliente.

- **Consumidor de servicio.** Eventualmente un servicio necesita ser invocado. En la operación de enlace, el consumidor de servicio invoca o inicia una interacción con el servicio en tiempo de ejecución utilizando los detalles existentes en la descripción del servicio para localizar, ubicar e invocar dicho servicio.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

El consumidor de servicio dispone de los archivos WSDL de todos los servicios a los que el cliente puede acceder para alcanzar un objetivo. Una vez seleccionado el servicio más apropiado se realiza la petición al proveedor del servicio a través de un mensaje SOAP. Posteriormente se ejecuta el WS utilizando los parámetros indicados en el mensaje SOAP y se devuelve al cliente, en otro mensaje SOAP, el resultado de la ejecución.

2.2.2.2 Lenguaje de Descripción de Servicios Web (WSDL)

Este lenguaje de Descripción de Servicios Web (Web Service Description Languaje, WSDL) proporciona un modelo y un formato XML para describir WS. WSDL 2.0 permite separar la descripción de la funcionalidad abstracta ofrecida por el WS de los detalles más concretos relativos a “como” y “dónde” obtener dicha funcionalidad (Chinncini, Moreau, Ryman y Weerawarana, 2007). Por lo tanto, podemos decir que una descripción WSDL 2.0 está estructurada en dos niveles fundamentales: una concreta y una abstracta.

En el nivel abstracto, WSDL 2.0 describe un WS en términos de los mensajes que este envía y recibe. Los mensajes están descritos independientemente del lenguaje utilizado, empleándose para ello un sistema de tipos como es el XML Schema.

Una operación (operation) asocia un patrón de intercambio de mensajes a uno o más mensajes. Un mensaje de intercambio de patrón (message exchange pattern), identifica la secuencia y el orden de los mensajes enviados y/o recibidos, así como a quién se

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

envía y/o de quién se recibe. “Una interface” se encarga de agrupar las operaciones, sin concretar un formato específico.

En el nivel concreto, un enlace (binding) especifica los detalles relativos al formato, para uno o más interfaces. Un punto final (endpoint) se encarga de asociar una dirección de red con una vinculación. Un servicio (service), agrupa los endpoints que implementa una “interface” común.

La descripción de servicio de WSDL 2.0, nos indica cómo los clientes potenciales pueden interactuar con el servicio descrito, la interfaz de WSDL 2.0, describe las posibles interacciones con un WS.

2.2.2.3 Descripción Universal, Descubrimiento e Integración.

Descripción Universal, Descubrimiento e Integración (Universal Description Discovery and Integration, UDDI) es la definición de un conjunto de servicios de apoyo y descubrimiento de empresas, organizaciones y otros proveedores de WS que están disponibles; y las interfaces para poder acceder a estos WS. (Clement, Hately, Von y Rogers, 2004).

UDDI sirve para dos propósitos fundamentales: facilitar la descripción de los WS para ser útiles en el proceso de búsqueda y aportar las herramientas necesarias para hacer las descripciones de los WS lo suficientemente competas, para que tanto las personas como los programas puedan descubrir cómo interaccionar con los WS que acaban de descubrir.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

UDDI es una iniciativa industrial y abierta de Organization for the Advancement of Structured Information Standards (OASIS), que es un consorcio internacional que abarca una serie de organizaciones más importantes a nivel mundial y que se orienta al desarrollo y adopción de estándares para el negocio electrónico y WS.

Está basado en un conjunto de estándares común, entre los que se encuentran HTTP, XML, y XML Schema, proporciona una infraestructura estándar e interoperable que permite a las empresas y a las aplicaciones en contar y usar de forma rápida, fácil y dinámicas WS a través de Internet.

Los WS sólo tienen sentido si existen un mecanismo por el cual los usuarios son capaces de encontrar información sobre dichos servicios para poder ejecutarlos. UDDI se centra en la definición de un conjunto de servicios para dar soporte a la descripción y descubrimiento (Clement et al, 2004):

1. Negocios, organizaciones y otros proveedores de WS.
2. Los WS que éstos hagan disponibles.
3. Las interfaces técnicas que pueden ser utilizadas para acceder a dichos WS.

Los registros UDDI están basados en estándares bien conocidos como HTTP y XML, y están formados por tres partes fundamentales:

- Páginas blancas. Se encargan de registrar información acerca de la dirección, el contacto y otros identificadores conocidos.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

- Páginas amarillas. Indican una categorización industrial basada en taxonomías.
- Páginas verdes. Ofrecen información técnica sobre los servicios que aportan las empresas.

Esta infraestructura permite el acceso tanto a los WS que estén disponibles como a aquellos que se exponen en el interior de las organizaciones. La información que se muestra en los registros UDDI está formada por instancias de cuatro tipos de estructuras de datos, como podemos ver en la figura 2 (Clement et al, 2004). Que son: la entidad de negocio (businessEntity), la plantilla de vinculación (bindingTemplate), el servicio de negocio (bussinessService) y el tModel.

Se precisa cada una de las estructuras de datos mencionadas.

- Estructura de negocio (businessEntity). Posee información descriptiva de una organización o negocio. A Través de sus relaciones con entidades de servicio de negocio, también posee información acerca de los servicios que ofrece.
- Estructura de servicio de negocio (businessService). Representa un servicio lógico y contiene información descriptiva en términos de negocio. Cada servicio de negocio se relaciona con la entidad de negocio que proporciona un determinado servicio.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

- Plantilla de vinculación (bindingTemplate). En esta estructura se encuentra la descripción técnica de los WS, etas reciben una instancia de un WS ofrecida desde una dirección de red. Adoptando información acerca de un tipo de WS ofrecido, parámetros específicos de la aplicación y configuraciones, utilizando referencias a las entidades tModel.

- tModel. Proporcionan un sistema de referencia basado en abstracciones para indicar la conformidad de los WS en determinadas especificaciones.

2.2.2.4 Protocolo de Acceso simple a Objeto (SOAP)

El protocolo de Acceso Simple a Objeto (SOAP) se utiliza en una aplicación de WS estándar. El mecanismo de comunicación es proporcionado para intercambio de datos en formato XML a través de un protocolo de red, habitualmente http. Al igual que ocurre con WSDL, estos mensajes se basan en documentos XML utilizando elementos básicos como cabecera, cuerpo, etc. (Goncalves, 2010).

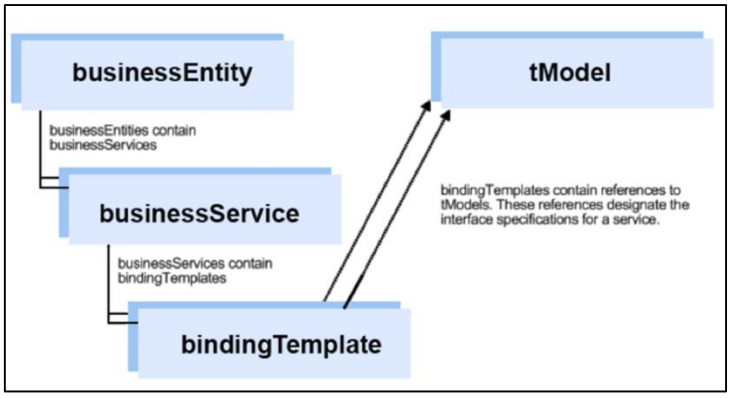

Figura 2. Estructura de datos UDDI

Fuente: (Clement et al*, 2004)*

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Es importante mencionar que SOAP es un protocolo ligero que permite el intercambio de información estructura en un entorno descentralizado y distribuido, utiliza la tecnología XML para definir un marco de intercambio de mensajes extensibles, que permite trabajar sobre múltiples pilas de protocolos. Ha sido desarrollado para ser independiente de cualquier lenguaje de programación y ha sido diseñado para ser simple y extensible. (Gudgin, Hadley, Mendelson, Moreau, Nielsen, Karmarkar y Lafon, 2007).

Un mensaje SOAP es una transmisión entre nodos SOAP, desde un emisor SOAP a un receptor SOAP en un único sentido. De este modo y teniendo en cuenta el objetivo de la simplicidad, SOAP aporta un paradigma de intercambio de mensajes sin estado y en una sola dirección, aunque cabe la posibilidad de que las aplicaciones, teniendo en cuenta el objetivo de la extensibilidad, puedan crear patrones de interacción más compleja.

Existen un gran número de especificaciones que determinan cómo han de realizarse algunas de las tareas más importantes relacionadas con SOAP: reglas para el procesador de mensajes, modelo de extensibilidad, reglas para la vinculación con SOAP, etc.

Podemos ver en la figura 3, la estructura de los mensajes SOAP. Todo mensaje SOAP, está compuesto por los siguientes elementos:

- Contenedor SOAP (SOAP Envelope). Proporciona un mecanismo para extender un mensaje SOAP de forma modular y descentralizada. En este elemento se incluye la información de

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

control, distinta a la que se incluye en el cuerpo del mensaje. Dentro de esta información de control, se incluyen directivas de transición e información contextual, relacionada con el procesado del mensaje, permitiendo a los nodos intermedios proporcionar servicios de valor añadido.

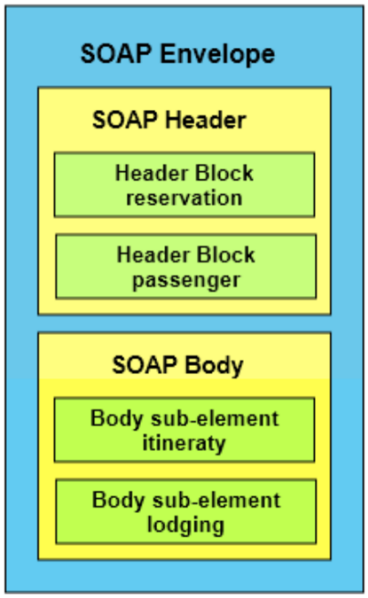

Figura 3. Estructura de un mensaje SOAP

Fuente: (San José, 2016)

- La cabecera está formada por “header blocks”, o bloque de cabecera, que representan una agrupación lógica de datos. Estos bloques intermedios, pueden estar destinados a nodos intermedios específicos en el camino entre el emisor y el receptor del mensaje.

- Cuerpo SOAP (SOAP Body). Proporciona un mecanismo para transmitir la información, relativa a la aplicación, al receptor SOAP final. Este elemento contiene, además del nombre, un especio de

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

nombres y una lista de atributos, sub-elementos que contiene la información útil del mensaje.

2.2.2.5 Arquitectura Orientada a Servicios (SOA)

La Arquitectura Orientada a Servicios (Service Oriented Architecture, SOA) es definido como un paradigma para organizar y utilizar capacidades de forma distribuidas, y que pueden estar bajo el control de diferentes dominios de propiedad. (MacKenzie, Laskey, McCabe, Brown, Metz y Hamilton, 2006)

SOA es una arquitectura de software que propone la construcción de aplicaciones mediante el ensamblado de bloque reutilizables, débilmente acoplado y altamente interoperables, cada uno de los cuales es representado como un servicio. Estos bloques pueden encontrarse distribuidos y pertenecer a diferentes propietarios. Es por ello que dicha arquitectura es utilizada para la Integración de Aplicaciones Empresariales (Enterprise Application Integration, EAI) y Comercio Electrónico entre Empresas (Business-to-Business, B2B) (Canto, Pereda y Segurola, 2006).

En general, las entidades (personas y organizaciones) crean capacidades para resolver o proporcionar una solución a los problemas a los que se enfrentan en el ejercicio de su actividad. Es natural pensar en que las necesidades de una persona pueden ser cumplidas por las capacidades ofrecidas por otra persona; o que en el mundo de la computación distribuida los requerimientos de un agente, pueden ser satisfechos por otro agente distinto.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Las arquitecturas SOA se caracterizan por tener las siguientes propiedades (Booth, Haas, McCabe, Newcomer, Ferris y Orchard, 2004):

- Vista lógica. Un servicio es una vista abstracta y lógica de aplicaciones, Base de Datos, procesos de negocio, etc., definidos en términos de qué hace y normalmente llevado a cabo en una operación de lógica de negocio.

- Orientado a mensajes. El servicio está definido, formalmente, como un intercambio de mensajes entre entidades que suministran servicios y entidades que solicitan servicios, sin mencionar las propiedades de las entidades en sí.

Esta característica cobra mayor importancia cuando se pretende interactuar con sistemas heredados (legacy Systems) ya que, al abstraerse de cualquier conocimiento relacionado con la estructura interna de una entidad software, cualquier componente software que permita ser accedido a través de un sistema basado en mensajes, podrá ser utilizado por medio de servicios.

- Orientado a la descripción. Un servicio se describe a través de metadatos procesables por el computador. La descripción da soporte a la naturaleza pública de SOA: sólo aquellos detalles que se exponen públicamente y que son importantes para describir el servicio, debe ser incluidos en la descripción.

- Granularidad. Los servicios tienen a utilizar un pequeño número de operaciones, con mensajes relativamente largos y complejos.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

- Orientados a la red. Los servicios tienen a ser orientados hacia el uso sobre una red de comunicaciones de la plataforma. XML es un formato que cumple esta restricción.

- En general, la aplicación de WS para el desarrollo SOA es conveniente cuando la aplicación a desarrollar cumple los siguientes requisitos (Booth et al, 2004):

  - Entornos que operan a través de Internet y que no pueden garantizar la fiabilidad y la velocidad.

  - Entornos donde no hay posibilidad de gestionar el despliegue de la aplicación, de forma que todos los proveedores y solicitantes son actualizados a la vez.

  - Entornos donde los componentes del sistema distribuido se ejecutan en diferentes plataformas y productos de proveedores.

  - Entornos donde una aplicación existente necesite ser expuesta para su uso a través de una red y que pueda ser empaquetada como un WS.

SO es una arquitectura que permite organizar mucho mejor los sistemas de Tecnologías de la Información (Information Techonologies Systems, IT Systems) de una empresa. Este tipo de organización aporta una serie de ventajas muy destacables, como las citadas a continuación (Alba, 2008):

- Escalabilidad.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

- Robustez.
- Homogeneidad.
- Facilidad en la adaptación de nuevos servicios.
- Facilidad en la reestructuración de sistemas.
- Aplicar lógica en el middleware para implementar procesos de negocio.
- Recoger información y procesarla para obtener resultados más útiles.
- Ahorro en tiempos de implantación.
- Ahorro en tiempos de mantenimiento y operación.

A pesar de tener un gran número de ventajas, SOA también tiene sus desventajas. Una de ellas es la velocidad de intercambio de información entre sistemas, más lenta que una conexión directa. Que SOA sea una arquitectura muy estudiada para aportar grandes beneficios, no implica que sea recomendable su uso para todos los escenarios. Recomendar la aplicación de SOA, su alcance, dónde y cómo aplicar, etc., suele ser un proceso lento, debido al gran impacto que tiene en los sistemas que se encuentran en producción (Alba, 2008).

2.2.3 Servicios Web REST (REST WS)

Representational State Transfer (REST) fue originalmente introducido como un estilo arquitectónico para la construcción de sistemas hypermedia

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

distribuidos de gran escala. Este estilo arquitectónico es una entidad más abstracta cuyos principios se han utilizado para explicar la excelente escalabilidad del protocolo HTTP 1.0 y también se ha limitado el diseño de su sucesor, HTTP 1.1. De ahí que el término REST se utilice muy a menudo en conjunción con HTTP (Pautasso, Zimmerman y Leymann, 2008)

REST está basado en cuatro principios:

- **Identificación de recursos a través de URI.** Un servicio web RESTful muestra un conjunto de recursos que identifican los objetivos de la interacción con sus clientes. Los recursos, son identificados por URLs, que proporcionan un espacio de direccionamiento global de recursos y descubrimiento de servicios. La sintaxis básica de las URLs, puede consultarse en (Bernerss-Lee, Fielding, Fielding y Masinter, 2005)

- **Interfaz uniforme**. Los recursos son manipulados a través de un conjunto fijo de cuatro operaciones (créate, read, update, delete): PUT, GET, POST y DELETE. PUT modifica un recurso existe, DELETE elimina el recurso, GET recupera el estado actual de un recurso en una representación y POST transfiere un nuevo estado a un recurso.

- **Mensajes auto-descriptivos**. Para los que no están conectados a su representación, y que se pueda acceder al contenido a través de una gran variedad de formatos (por ejemplo, HTML, XML, texto plano, PDF, etc.). Los metadatos de un determinado recurso, están disponibles y se utilizan, por ejemplo, para

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

controlar el almacenamiento en memoria caché, detectar errores de transmisión, negociar el formato de representación Y llevar acabo la autentificación o controlar el acceso.

- **Interacciones de estado a través de hipervínculos.** Cada interacción que se realiza con un recurso no tiene estado, Es decir, Los mensajes de solicitud son autónomos. Las interacciones que sí poseen estado, se basa en el concepto de transferencia de estado explícito. Existen varias técnicas para el intercambio de estado, como por ejemplo la reescritura de URIs, cookies y los campos de formularios ocultos. Un Estado puede estar en bebido en los mensajes de respuesta, para señalar como válidos estados futuros de la interacción.

Gracias a las aplicaciones de los principios de desarrollo REST WS, cada vez es más adoptada y elegida, estos servicios al momento de exponer sus datos internos y funcionalidades, Como recursos URI identificados. En contraste con la perspectiva de operación céntrica de los WS WSDL/SOAP, los REST WS permite ver las aplicaciones desde una perspectiva centrada en los recursos (Zhao y Doshi, 2009). Algunas de las ventajas de estos WS son:

- **Ligero.** Uso de HTTP como protocolo de invocación evitando los marcadores XML innecesarios o la encapsulación extra para las APIs de entrada/salida. Como resultado, este tipo de WS son más sencillos de desarrollar Y Utilizar que los WS WSDL/SOAP, especialmente en el contexto de la Web 2.0 además no dependen

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

tanto del proveedor de software y los mecanismos que implementa la capa superior de SOAP sobre HTTP.

- **Fácil accesibilidad.** Las URIs utilizadas para identificar cualquier WS RESTful puede ser compartidas y enviadas a cualquier cliente de servicio dedicado o a cualquier aplicación, Con el objeto de poder ser reutilizadas. Las URIs y la representación de recursos son auto-descriptivos y, por lo tanto, hacen que estos WS Sean fácilmente accesibles. Este tipo de WS han sido utilizados para desarrollar aplicaciones de la Web 2.0 y matchups.

- **Escalabilidad.** La estabilidad de los REST WS proviene de su capacidad para apoyar el almacenamiento en caché y por qué saneamiento de las URIs. Gracias a la operación GET, operación que está libre de efectos secundarios, pueden almacenar en la caché exactamente la misma información que está almacenada en los proxies, puertas de enlace y Redes de entrega de Contenido (Content Delivery Networks, CDNs). En comparación con el particionamiento a-doc que hay detrás de las funciones de las interfaces SOAP, el particionamiento basado en URIs es más genérico y flexible.

- **Declarativo**. Los REST WS toman un enfoque declarativo, que permite visualizar operaciones desde un ángulo de vista de los recursos. Al ser declarativo se centra en la descripción de los recursos, en lugar de describir cómo se realiza las funciones. Este

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

enfoque desconsiderado la mejor opción a la hora de construir sistemas SOA flexibles, escalables Y debidamente acoplados.

La mayoría de los recursos asociados a los WS RESTful, pueden ser directamente asignados aló recursos dominio, Ya sea un conjunto de recursos O a recursos individuales. Además de estos dos tipos de recursos, se identifica otro tercer tipo, que consumen algunos recursos o manipulan recursos relacionados que no se pueden asignar directamente los recursos dominio O a colecciones de recursos (Zhao y Doshi, 2009). A continuación, los describimos en mayor detalle:

- **Tipo 1: conjunto de servicios de recursos:** este tipo de servicios se asignó un conjunto de recursos del dominio Y se pueden utilizar para la captura de recursos a nivel de concepto o en el conjunto de recursos de instancia, es un tipo de servicios de apoyo a las operaciones GET, PUT, DELETE y POST.

- **Tipo 2: servicios de recursos individuales.** Los recursos del dominio pueden ser moderados con este tipo de servicios, que representan a los recuerdos individuales dentro de un conjunto de recursos. Este tipo de servicios, puede utilizarse para captar recursos del de instancia Y es compatible con las operaciones GET, PUT y DELETE. En este caso, POST no es aplicable, ya que se ha creado el recurso individual y posee un URI para identificarlo.

- **Tipo 3: Servicios de Transición (Transitional Services).** Aunque la mayoría de los WS RESTful se asignan a los recursos

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

del dominio O como conjuntos de recursos, algunos de los servicios son más de transición O de transición orientada. La funcionalidad de este tipo de servicios se define como servicios que consumen recursos, creo recursos y actualizan o transforman los estados de los recursos relacionados. Este tipo de WS pueden ser utilizados para capturar las funcionalidades orientados a la transición y sólo son compatibles con operación POST.

2.2.4 Computación en la nube (Cloud Computing)

2.2.4.1 Introducción

El término nube (Cloud) O computación en la nube (Cloud Computing) es un concepto tecnológico cada vez más utilizado. Las organizaciones ven a esta tecnología como la solución a muchos problemas de infraestructura al igual que económicos (Joyanes, 2012).

El modelo Cloud Computing permite, Convenientemente, el acceso ubicuo a un conjunto compartido recursos informáticos configuradores, Como servidores, redes, almacenamiento aplicaciones y servicios. Estos recursos pueden ser rápidamente a provisionados Y liberados, Con un esfuerzo mínimo de gestión o de interacción con el proveedor de servicios (Mell y Grance, 2011).

La nube es un conjunto de hardware y software de almacenamiento, servicios e interfaces que facilitar la entrada de información como un servicio. En el modleo Cloud se tiene un gran número de

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

participantes o actores. Los grupos de intereses dentro del Cloud Computing son:

- **Vendedores o proveedores.** Proporciona las aplicaciones Y facilitar las tecnologías, Infraestructura, plataformas e información correspondiente.
- **Socios de los proveedores.** Creo servicios para la nube ofreciendo servicios a los clientes.
- **Líderes de negocios.** Ellos evalúan los servicios de la nube para implantarlos en sus empresas y organizaciones.
- **Usuarios finales.** Requieren utilizan los servicios de la nube gratuitamente o pagando una tarifa apropiada.

Este modelo de Cloud Computing según el Instituto Nacional de Estándares y Tecnología (National Institute of Standards and Technology, NIST) se conforma esencialmente por cinco características, tres modelos de servicio y cuatro modelos de despliegue. Describiremos las características y modelos a continuación.

2.2.4.2 Características.

Según NIST el Cloud Computing posee las siguientes características (Joyanes, 2012):

- **Auto servicio bajo demanda**. Un usuario puede proveerse, unilateralmente, de tiempo de ejecución de un servidor y almacenamiento en red a medida que lo necesite sin requerir interacción humana con el proveedor de servicio.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

- **Acceso ubicuo a la red.** Se realiza mediante dispositivos estándar que promueven el uso por plataformas de clientes ligeros (teléfonos móviles, computadores portátiles, PDAs, Tablets, etc.).
- **Distribución de recursos independientes de la posición.** Los recursos de computación del proveedor son de pagos para servir a múltiples consumidores, Utilizando un modelo multi-distribuido y que tienen diferentes recursos virtuales y físicos, asignados y reasignados dinámicamente conforme a la demanda del consumidor. Se genera una sensación de independencia de la adopción de modo que el cliente, normalmente, No suele tener control ni conocimientos sobre la posición exacta de los recursos proporcionados.
- **Elasticidad rápida.** Las funcionalidades que pueden proporcionarse muchas veces son de modo inmediato, elástico y en ocasiones, se proveen automáticamente. Esta características de aprovisionamiento genera una sensación de contar con recursos ilimitadas y pueden adquirirse en cualquier cantidad o momento.
- **Servicio medido.** Los sistemas que gestiona Cloud Computing optimizan y controlan automáticamente el uso de recursos, incrementando la capacidad de medición en un nivel de abstracción apropiado de un tipo de servicio. El uso de estos recursos puede ser monitorizado, controlado he informado, generando transparencia para el proveedor y para el consumidor.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

A la hora de trabajar con Cloud Computing no es necesario disponer de un equipo potente, sólo es necesario un dispositivo con conexión a Internet. Esto se debe muchas veces a que el dispositivo del usuario no realizará ningún proceso complejo y los archivos pueden almacenarse en la nube. Los servidores donde se encuentran alojados los programas son los encargados de realizar las tareas de procesamiento (Avila, 2011).

Con el uso del Cloud Computing, el usuario no siente el requerimiento de conocer la infraestructura que detrás, ya que pasa hacer una abstracción donde las aplicaciones y servicios pueden crecer fácilmente, Funcionan rápido y con pocos errores.

Podemos mencionar también las siguientes características dentro del Cloud Computing:

- **Auto-reparable.** En caso de que surja algún fallo, la última copia de respaldo de una determinada aplicación, es convertida automáticamente en la copia primaria y a partir de esta, se vuelve a generar todo nuevamente.
- **Escalabilidad.** Todo el sistema y su respectiva arquitectura, Es eficiente y predecible. Por ejemplo, si un servidor maneja 1000 transacciones, Para manejar 2000 transacciones será necesario dos servidores. Se establece un nivel de servicios que crean nuevas instancias, en función a la demanda de operaciones existente, de tal forma que se reduzca el tiempo de espera y los cuellos de botella.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

- **Virtualización.** Las aplicaciones son independientes del componente hardware en el que se ejecuten, Incluso varias aplicaciones pueden ejecutarse en una misma máquina O una aplicación puedo utilizar varias máquinas a la vez. El usuario puede usar la plataforma que desee eso terminal (Mac, Windows, Linux, Unix, etc.), ya que, al utilizar las aplicaciones existentes en la nube, puede estar seguro de que sus archivos (trabajo) podrán conservar a sus características bajo cualquier otra plataforma.
- **Disponibilidad de la información.** No es necesario que un usuario almacene en un dispositivo los documentos, ya que la información radicará en Internet, Permitiendo su acceso desde cualquier dispositivo conectado a la red, con la autorización requerida.

2.2.4.3 Modelos de servicio.

Los modelos definidos de servicio se refieren a los servicios más específicos a los que pueda acceder en una plataforma Cloud Computing. Estas tecnologías ofrecen tres modelos de servicios (Joyanes, 2012):

- **Software como servicio (Software as a Service)**. Se le ofrece al usuario la capacidad de que las aplicaciones aprovisionadas se desenvuelvan en una infraestructura en la nube, haciendo que las aplicaciones sean accesibles a través de un navegador web, como puedes ser el correo web. El usuario adolece de cualquier control sobre la infraestructura o sobre las propias aplicaciones,

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

aunque puede haber excepciones para posibles personalizaciones o configuraciones hechas por el mismo usuario.

- **Plataforma como servicio (Platform as a service).** El usuario tiene la posibilidad de desplegar aplicaciones propias, adquiridas o desarrolladas por el mismo en la infraestructura de la nube de su proveedor. El proveedor lo ofrece la plataforma de desarrollo Y las herramientas de programación. El usuario preserva el control de la aplicación, aunque no de toda la infraestructura subyacente.

- **Infraestructura como servicio (Infraestructure as a Service)**. El proveedor ofrece recursos como capacidad de procesamiento, almacenamiento o comunicaciones, que el usuario puede utilizar para ejecutar cualquier software.

Estos modelos de servicio formal arquitectura de la nube, Como podemos ver en la figura 4. El Cloud Computing basa su arquitectura haciendo una separación entre el software, la plataforma y aplicaciones (Avila, 2011).

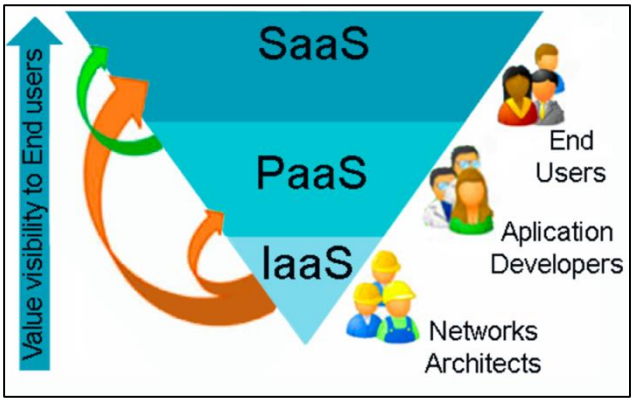

Figura 4. Arquitectura de capas en la nube
Fuente: (Avila, 2011)

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

2.2.4.4 Modelos de despliegue. Los modelos despliegues se refiere a la posición y administración de la infraestructura de la nube. (pública, comunitaria, híbrida). Podemos consultar este modelo en la figura 5.

- **Private Cloud (Nueve Privada).** La infraestructura está preparada para el uso exclusivo de los clientes de una organización.

- **Public Cloud (Nube Pública).** La infraestructura es operada por un proveedor que ofrece servicios al público en general.

  Puede ser propiedad, estar gestionado y operado por una empresa, Organización gubernamental, Entorno académico o una combinación de ellos.

- **Hybrid Cloud (Nube Híbrida).** Resultado de la combinación de dos o más nubes individuales. Estas nubes pueden ser privadas, compartidas o públicas. Permite el envío de datos o aplicaciones entre ellas.

- **Community Cloud (Nube Comunitaria).** Organizada para servir a una función a propósito común. Es preciso compartir objetos comunes.

  Puede ser propiedad, estar gestionada y operada por una o más organizaciones de dicha comunidad, por un tercero o una combinación de ellos.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

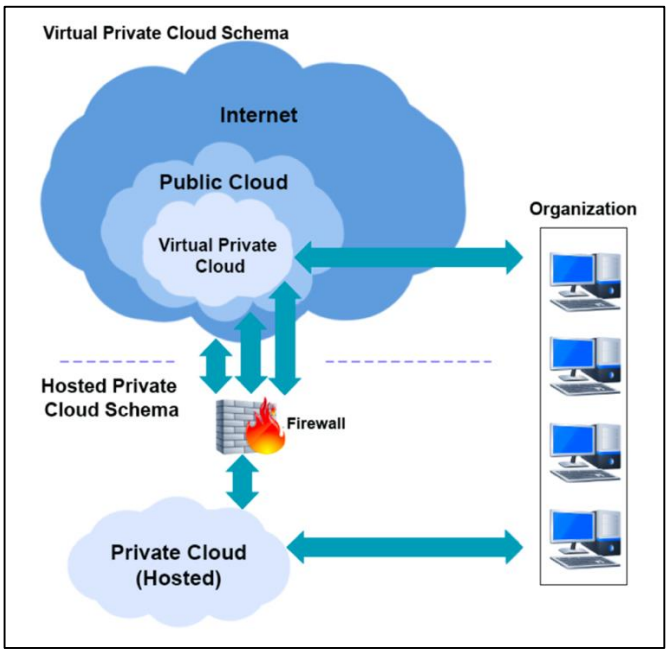

Figura 5. Modelos de despliegue: tipos de nubes computacionales

Fuente: (Joyanes, 2012)

2.2.5 Tecnologías

En esta sección describiremos las tecnologías que se utilizan en los entornos de computación en la nube (Zhang, Cheng y Boutaba, 2010).

- **Diseño arquitectónico de los Data Centers.** Un data center es un núcleo para la computación en la nube Y contiene dispositivos como servidores, conmutadores (Switches) o encaminadores (routers). Es fundamental una buena planificación de esta arquitectura de red, ya que influye gran medida del rendimiento de aplicaciones y en el entorno de computación distribuida. Además, se deben considerar cuidadosamente las funciones de escalabilidad y capacidad.

  En la actualidad, un modelo de capaces básicos del diseño de una

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

arquitectura de red. Las capas básicas de un data Center son: núcleo, agregación y acceso, Como podemos ver en la figura 6. La capa de acceso es donde se encuentran los racks de servidores, existiendo de 20 a 40 servidores por rack y están conectados con switch a través de un enlace de 1Gbps. Los switches de acceso generalmente se conectan a dos switches de agregación para permitir la redundancia, A través de enlaces de 10Gbps.

La capa de agregación general proporciona funciones importantes, Como: servicios de dominio, servicio de ubicación, equilibrio de carga del servidor, etc. esta capa central proporciona conectividad a múltiples switches de agregación, proporcionando también un tejido resistente del enrutado, sin ningún punto único de fallo.

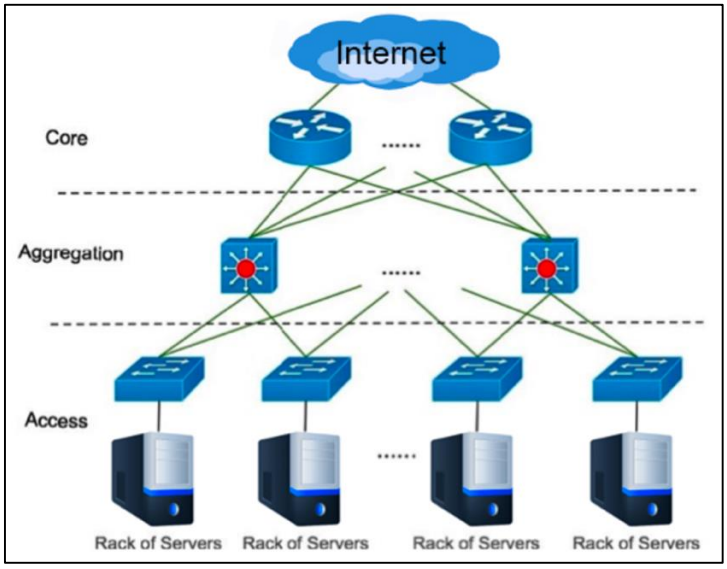

Figura 6. Modelo de capas de un centro de datos

Fuente: (Zhang, Cheng y Boutaba, 2010)

Los routers de cada capa núcleo son los encargados de gestionar tráfico hacia/desde el data center. Una práctica habitual consiste en aprovechar la

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

comodidad que proporcionan los switches ethernet y los routers, para construir la infraestructura de la red.

Básicamente, el diseño de una arquitectura de red de un data center debe cumplir los siguientes objetivos (Greenbeng et al, 2009): Alta capacidad uniforme, Migración libre de máquinas virtuales, capacidad de recuperación, escalabilidad Y retro-compatibilidad.

Otra tarea de rápida innovación en este campo es el diseño y despliegue de los centros de datos modulares (Modular Data Centers, MDC). En los MCD hasta 2000 servidores pueden estar interconectados a través de switches para conformar una estructura de red.

- **Sistema de archivos distribuidos sobre la Nube.** Un primer ejemplo de este tipo de sistemas de archivos es Google File System (GFS) (Ghemawat, Gobioff y Leung, 2003). GFS es un sistema de archivos distribuidos diseñados para proporcionar un acceso eficaz fiable a los datos, utilizando grandes grupos de servidores.

  Los archivos se dividen en partes de 64 MB y por lo general, se anexan o leen muy pocas veces una vez que es que han sido particionados. En comparación con los sistemas de archivos tradicionales, FGS está diseñado y optimizado para ejecutarse los Data Centers, para proporcionar grandes transferencias de datos, Con una baja latencia y sobrevivir a fallas que puedan producirse.

  Inspirado hoy en FGS, Hadoop Distributed File System (HDFS) (Perera y Gunarathne, 2013) es una versión de código abierto que almacena archivos de gran tamaño a través de múltiples máquinas. HDFS lograr la confiabilidad mediante la replicación de los datos entre varios

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

servidores y almacena los datos distintos nodos. Los datos se proporcionan a través del protocolo HTTP, lo que permite el acceso a todo el contenido desde un navegador web U otro tipo de cliente. Los nodos pueden comunicarse entre sí para reequilibrar la distribución de datos, Para mover copias de uno a otro y mantener la reputación de datos.

- **Framework de aplicaciones distribuidas sobre la nube.** Aplicaciones basadas en HTTP que normalmente se ajustan algún framework de aplicaciones web, como puede ser Java EE. En los Data Centers los Cluster de servidores también se suelen utilizar para trabajos de cálculo Y De uso intensivo de datos, cómo puede ser el análisis de la actividad financiera o el cine de animación.

  Algunos ejemplos son MapReduce (Dean y Ghemawat, 2008), framework introducido por Google para apoyar la computación de grandes conjuntos de datos en Clusters de computadores y Hadoop MapReduce (Perera y Gunarathne, 2013), versión en código abierto de MapReduce.

2.2.6 Plataformas y Servicios

En la actualidad, existe una gran cantidad de plataformas de servicios en la nube. Siendo que además de los servicios de plataforma, existen también una gran cantidad de servicios ofrecidos como software, Que permite utilizar la nube como otra máquina virtual para almacenar, manipular Y sincronizar datos. A continuación, se describe algunos de los servicios más conocidos (Avila, 2011):

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

- Google Apps. Es uno de los diversos servicios que ofrece Google, que proporcione herramientas eficaces para la gestión y personalización de utilidades para dominios o nombres internet

- Amazon Elastic Computer Cloud (Amazon EC2). Amazon EC2 es un WS que proporciona capacidad informática en la nube, con posibilidad de modificar tamaño. Se ha diseñado con el fin de que la computación basada en la web resulte más sencilla a los desarrolladores.

- Windows Azure. Plataforma que se ofrece como WS y que está alojada en los centros de procesamiento de datos de Microsoft. Ofrece distintos servicios para aplicaciones, desde aquellos que permite guardar aplicaciones en alguno de los centros de procesamiento de datos, hasta otros que ofrecen comunicación segura y asociación entre aplicaciones.

2.2.7 Virtualización

2.2.7.1 Casos de uso de la virtualización

La virtualización se utiliza todo en los sistemas de información de todo el mundo. Grandes empresas como Google, Facebook, Amazon o Microsoft lo utilizan en varias capas de su infraestructura, pero esta tecnología no se utiliza sólo en clústeres de servidores, millones de personas lo utilizan en sus computadoras personales también.

Las pruebas son una parte importante en el ciclo de vida de desarrollo de cada software. Especialmente para grandes proyectos

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

multiplataforma, las pruebas normalmente incluyen ejecutar el producto en varias configuraciones del sistema. Una configuración diferente puede significar otra versión del sistema operativo y bibliotecas usadas o incluso un sistema operativo completamente diferente. Los recursos del sistema disponibles, el hardware y los controladores instalados son aspectos adicionales que contribuyen al comportamiento de la aplicación, por lo que el intercambio o la limitación de ellos también podría ser el foco de las pruebas. Tener estas diferentes configuraciones en computadoras físicas resultaría impráctico tanto desde el punto de vista oportuno como desde el punto de vista económico.

Las tecnologías que contamos hoy para virtualización a menudo están estrechamente relacionadas con la emulación, actualmente más visibles cuando se emplean máquinas virtuales. Gracias a ellos, es posible ejecutar software escrito para arquitecturas totalmente diferentes, como ejecutar aplicaciones ARM en sistemas x86 o software Linux en Windows y viceversa.

Frecuentemente, los expertos en seguridad necesitan ejecutar una aplicación en un entorno con caja de arena, cuando el análisis estático no lleva a conclusiones significativas. Por otra parte, incluso los usuarios ordinarios pueden preocuparse por la forma en que una pieza no confiable de software podría afectar a su sistema principal. Los beneficios que ofrece la capacidad de ejecutar la aplicación dentro de un sistema aislado son lo que hace que las tecnologías de virtualización sean extremadamente valiosas en tales escenarios.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Muchas empresas de TI no tienen su propia infraestructura en la nube, sino que se basan en el uso de una solución externa. Por otra parte, una empresa que tiene una gran capacidad de cálculo puede decidir alquilar parte de ella. Hay muchos productos del negocio de Cloud Computing en el mercado.

La virtualización tiene lugar en las tres capas, incluso en IaaS, ya que a menudo no representa el hardware directamente, sino más bien un conjunto de recursos aislados del resto de la nube utilizando los medios de virtualización. El entorno del sistema operativo en el caso de PaaS o las instancias de aplicación en el caso de SaaS también se aíslan.

2.2.7.2 Virtualización basada en máquinas virituales

Monitores de máquinas virtuales

Los monitores de máquinas virtuales, a veces denominados hipervisores, son soluciones de software o hardware que permiten la creación y el funcionamiento de máquinas virtuales. (Popek y Goldberg, 2013) han declarado las siguientes características de los monitores de máquinas virtuales:

- La propiedad de eficiencia. Todas las instrucciones inocuas son ejecutadas directamente por el hardware, sin intervención alguna por parte del programa de control.
- La propiedad de control de recursos. Debe ser imposible que dicho programa arbitrario afecte a los recursos del sistema, es

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

decir, a la memoria, a su disposición; El asignador del programa de control debe invocarse en cualquier intento.

- La propiedad de equivalencia. Cualquier programa X que se ejecute con un programa de control residente, con dos posibles excepciones, realiza de una manera indistinguible del caso cuando el programa de control no existió y X tenía cualquier libertad de acceso a instrucciones privilegiadas que el programador había previsto.

Se distinguen dos tipos de hipervisores. El primer tipo se ejecuta directamente en el hardware, mientras que los hipervisores del segundo tipo actúan como aplicaciones dentro de otro sistema operativo. El sistema operativo con acceso directo al hardware se llama host, mientras que los sistemas en entornos virtualizados se llaman invitados.

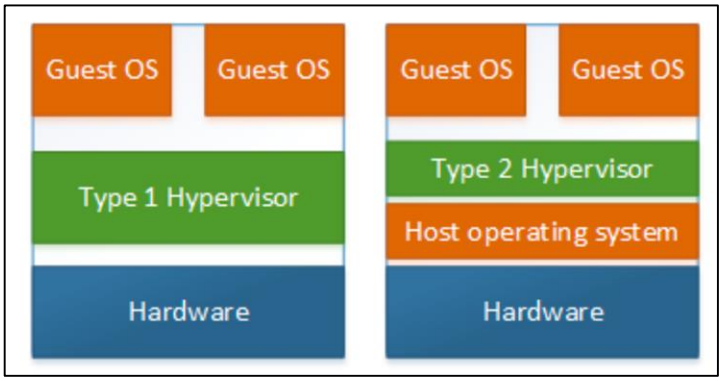

Figura 7. Hipervisores de tipo 1 y tipo 2

Fuente: (VirtualBox y Vmware)

Virtualización basada en software

Dado que los sistemas operativos invitados siempre se ejecutan en la parte superior de un hipervisor, no acceden directamente al

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

hardware. Así mismo, desde la seguridad como punto de vista, el no debería ser capaz de hacerlo, ya que de lo contrario serían capaces de corromper el anfitrión o tomar el control de él. En las subsecciones siguientes, se explican las causas detalladas de las dos penalizaciones de rendimiento más notables de las máquinas virtuales.

CPU

Las CPUs modernas utilizan un esquema de protección de recursos donde se crean varios anillos de privilegios y cada uno de ellos representa un nivel diferente de confianza. Una arquitectura x86 ofrece hasta cuatro anillos (numerados de 0 a 3) y normalmente un número de anillo inferior representa un código más confiable. A veces, los números de anillo se conocen como nivel de privilegio actual - CPL. La parte del sistema operativo, que necesita más privilegios, se ejecuta en el anillo 0, los controladores de dispositivo funcionan en el anillo 1 o más y para la mayoría de las otras aplicaciones, el número de anillo más alto es suficiente. Algunas instrucciones, por ejemplo, E/S directa requieren los privilegios más altos. Cuando una aplicación de usuario necesita tales instrucciones, llama a una función proporcionada por el sistema operativo, lo que desencadena la ejecución del sistema operativo, el código del Kernel, que se ejecuta en el anillo 0. Una vez que la función finaliza, la función devuelve el resultado apropiado y la ejecución del código de la aplicación continúa, con los privilegios retrocediendo. Este mecanismo se denomina llamada de sistema, la capa privilegiada

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

inferior (anillo), donde las aplicaciones ejecutan marcas el espacio de usuario, y la capa del Kernel se etiqueta como kernelspace.

Desde la perspectiva de la virtualización, esto proporciona un mecanismo de seguridad, que ayuda a satisfacer la propiedad de seguridad del monitor de máquinas virtuales. Al no ejecutar el código invitado en el anillo 0, el sistema operativo host es el único con control total de los recursos de la computadora. Sin embargo, cuando el operador invitado intenta ejecutar una llamada de un sistema, fallaría, puesto que no funciona en el anillo 0. Para superar este problema, se puede emplear la paravirtualización, una técnica de compilación en la que las instrucciones Imposibles de ejecutar en entornos virtualizados se sustituyen estáticamente. Además, se pueden instalar controladores adicionales, que permiten al host comunicarse con el hipervisor. Otra opción es reemplazar las instrucciones en tiempo de ejecución, escaneando el código de ejecución y remendándolo cuando sea necesario.

Memoria

Los procesadores modernos usan un concepto de memoria virtual, donde cada aplicación consigue la ilusión, que puede usar toda la memoria del sistema para sí mismo. Para que esta técnica funcione, el sistema operativo tiene que gestionar la asignación de la memoria virtual a la memoria física. Si la memoria total consumida por las aplicaciones es mayor que la memoria física, partes de la memoria virtual se mueven temporalmente a un almacenamiento secundario (disco).

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Dado que el sistema operativo invitado no sabe que el host ya está administrando la memoria virtual, esto introduce una gran penalización de rendimiento para las máquinas virtuales, ya que cada acceso de memoria del invitado tiene que ser asignado en primer lugar en el espacio de direcciones del invitado Y luego nuevamente en el espacio de direcciones del host. La solución es sombrear la tabla de páginas (la estructura que mantiene un registro de las asignaciones de memoria), donde cada vez que el invitado mapea una memoria, el hipervisor realiza una asignación directa a la memoria del host en la tabla de sombra. A continuación, cuando el invitado intenta acceder a la memoria, se utiliza la tabla de páginas de sombra en lugar de la administrada por el invitado.

Virtualización asistida por hardware

Históricamente, no había mucho apoyo para la virtualización en el hardware de la computadora. Sin embargo, en 2005 y 2006, se lanzaron dos nuevas tecnologías, AMD-V para procesadores AMD e Intel VT para procesadores Intel. Si bien ambas tecnologías son muy similares en su funcionalidad, la terminología difiere ligeramente. He decidido utilizar la terminología Intel-V en el resto del capítulo.

CPU

Se ha introducido una nueva estructura: el bloque de control de la máquina virtual (VMCB), que representa una máquina virtual dentro de la CPU. Cuando un VMCB se ejecuta (instrucción VMRUN), la CPU ejecuta los siguientes pasos:

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

- El estado de hosts se guarda en un área de memoria especificada por VCMB
- El estado de invitados se carga desde un área de memoria especificada por VCMB
- El código de invitado comienza a correr

La estructura VCMB también contiene el nivel CPL deseado, permitiendo que la máquina virtual funcione incluso en el anillo 0. Sin embargo, algunas operaciones, como las E/S directas del dispositivo, están prohibidas y hacen que la máquina virtual salga del estado invitado. Después de la salida, el monitor de la máquina virtual puede decidir cambiar varios campos en el VCMB, emulando efectivamente las E/S y ejecutando VMRUN nuevamente.

Memoria

Intel y AMD también han venido con un mecanismo para eliminar la necesidad de sombrear tablas de páginas. Las tablas de páginas extendidas (Intel) y la indexación rápida de virtualización (AMD) son ejemplos de traducción de direcciones de nivel secundario, donde las tablas de páginas anidadas son creadas y mantenidas por el hardware. En este modelo, el hardware realiza las traducciones del espacio de direcciones virtuales del invitado al espacio de direcciones físico del invitado y del espacio de direcciones físico del huésped al espacio de direcciones físico del host.

VirtualBox

VirtualBox es un proyecto open source, usa el hipervisor tipo 2. Fue desarrollado inicialmente por Innotek GmbH, una compañía

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

alemana, que más tarde fue comprada por Sun Microsystems. Hoy en día, VirtualBox se marca como producto de Oracle, ya que, en el 2010, Oracle adquirió Sun Microsystems. El nombre completo del producto es, por tanto, Oracle VM VirtualBox.

VirtualBox soporta todos los principales sistemas operativos como hosts o invitados y con la excepción de un OS X invitado en hardware que no sea de Apple (aunque esto es posible usando una imagen agrietada de la instalación de OS X). Tanto la virtualización basada en software como la asistida por hardware se implementan. Los clientes de 64 bits son compatibles siempre y cuando el host sea de 64 bits o la virtualización de hardware de 64 bits sea compatible con la CPU.

Otras características notables son carpetas compartidas, portapapeles compartido o clonación de máquinas virtuales. Un estado completo de cualquier máquina virtual en ejecución (instantánea) también se puede guardar en un archivo y restaurar más tarde. VirtualBox ofrece una interfaz GUI rica, una interfaz de línea de comandos (VBox-Manage), un Python Shell interactivo y una API web. Una versión portátil que no requiere instalación está disponible también bajo el nombre Portable-VirtualBox.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

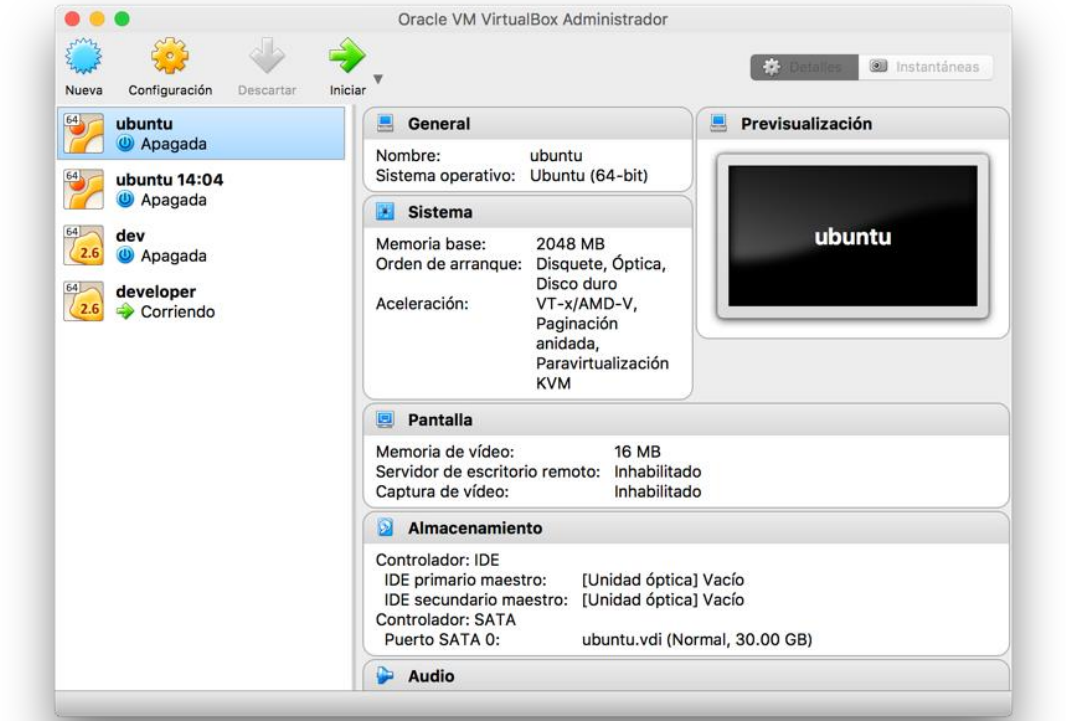

Figura 8. Interfaz gráfica de VirtualBox

Fuente: Captura de pantalla VirtualBox

Virtualización basada en contenedor

Proporcionar un entorno aislado dentro del sistema operativo de alojamiento se conoce comúnmente como virtualización de nivel de sistema operativo y tal entorno aislado puede definirse como contenedor: "Un contenedor es un entorno de ejecución autónomo que comparte el Kernel de la (Opcional) aislados de otros contenedores del sistema". Las tecnologías más utilizadas son los Contenedores o Zonas Solares, OpenVZ, cárceles FreeBSD y LXC.

Chroot y Jail

A medida que Linux se desarrolló, surgió la idea de aislar un proceso del sistema de archivos del host y se creó un comando chroot para este propósito. Ch-root que es corta para cambiar root es tanto una

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

utilidad como una llamada al sistema y permite especificar un nuevo directorio raíz que no sea /. El proceso y sus hijos, a continuación, no pueden acceder a los archivos por encima del nuevo directorio raíz, mientras que los programas de otros lugares todavía pueden ver dentro de la nueva raíz. Es crucial que ninguno de los procesos en el interior pueda obtener privilegios de root, ya que potencialmente les permite salir del directorio especificado. Este procedimiento se denomina a menudo jailbreak y se puede realizar fácilmente emitiendo chroot de nuevo mientras deja abierto un descriptor de archivo que apunta a un archivo fuera del directorio raíz recién seleccionado.

Hoy en día chroot se utiliza para proporcionar entornos aislados básicos para probar aplicaciones desconocidas e inestables o para descubrir dependencias no deseadas. Las herramientas de creación de paquetes como Pbuilder para Debian o Mock para Fedora también utilizan chroot para proporcionar aislamiento y permitir pruebas en diferentes distribuciones de Linux.

Espacios de nombres

Los espacios de nombres son una de las principales características del Kernel de Linux para soportar la virtualización ligera. "El propósito de cada espacio de nombres es envolver un recurso de sistema global particular en una abstracción que hace que aparezca a los procesos dentro del espacio de nombres que tienen su propia instancia aislada del recurso global". Para obtener una mejor

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

comprensión, me centraré en los tipos individuales. Actualmente hay seis espacios de nombres disponibles en Linux.

Mount namespace fue el primero implementado y su comportamiento es bastante intuitivo. Todos los montajes/desmontajes de los nombres globales son visibles en él. Los montajes/desmontajes que ocurren en el paso de nombres permanecen invisibles para todos los demás espacios de nombres, incluido el global. Sin embargo, también es posible establecer una relación maestro-esclavo para permitir la propagación de dispositivos montados.

UTS (Unix timestamp sharing) permite aislar gethostname(), getdomainname() identificadores, así como los miembros correspondientes de uname(). Cualquier cambio realizado llamando sethostname() o setdomain-name() sólo son visibles dentro del namespace de la persona que llama.

El espacio de nombres IPC (Interprocess communication) también proporciona aislamiento para System V IPC (memoria compartida, semáforos) y sistemas de archivos diferentes para colas de mensajes POSIX.

El espacio de nombres PID (ID de proceso) aísla la lista de identificadores de proceso. Permite que procesos de diferentes espacios de nombres tengan el mismo PID y anidamiento de espacios de nombres PID. El principio es que un proceso de un espacio de nombres particular puede ver (y enviar señales) sólo a

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

procesos en su propio espacio de nombres o en espacios de nombres anidados debajo de él. El primer proceso creado en el espacio de nombres PID tiene PID = 1, y cuando cualquier proceso en el espacio de nombres actual muere, todos sus procesos huérfanos se convierten en hijos del proceso con PID = 1. No es posible enviar SIGKILL a ningún proceso con PID = 1.

Los espacios de nombres de red permiten que cada espacio de nombres tenga su propia pila de red, incluyendo, pero no limitado a: direcciones IP, números de puertos, tablas de enrutamiento, reglas de firewall, dispositivos de red. Cuando se crea un nuevo espacio de nombres de red, sólo contiene el dispositivo de bucle (lo), pero los dispositivos de red se pueden mover a través de los nombres de red. La regla es que un dispositivo de red distinto de lo sólo puede pertenecer a un espacio de nombres de red. Además, los dispositivos físicos no se pueden mover desde el espacio de nombres predeterminado. Por lo tanto, si uno quiere dar capacidades de red a un espacio de nombres de red no predeterminado, el enfoque habitual es crear un dispositivo virtual ethernet (veth) en el espacio de nombres por defecto, un extremo con dispositivos físicos y mover el otro extremo al espacio de nombres, lo cual necesita redes. También es posible anidar los espacios de nombres de red.

Espacio de nombres de usuario. Los archivos /$proc_id/uid_map y /proc/$proc_id/gid_map contienen la asignación real de los rangos de ID de usuario (grupo) desde los espacios de nombres de usuario primario. El principal beneficio es que incluso un usuario no

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

privilegiado puede iniciar un proceso con privilegios de root, dentro de un espacio de nombres particular. Se admite la anidación de espacios de nombres de usuario.

La implementación de espacios de nombres agregó dos nuevas llamadas de sistema, setns() para unirse a un espacio de nombres existente y unshare(), que permite que los procesos de llamada continúen en un espacio de nombres recién creado.

La llamada al sistema clone() ha añadido 6 nuevos indicadores, uno para cada uno de los espacios de nombres, para permitir que el proceso secundario resultante de la llamada comience en un espacio de nombres recién creado.

Grupos de control

Con el uso de espacios de nombres, los procesos podrían aislarse unos de otros, pero eso aún está lejos de lo que ofrece la virtualización estándar. Como sugiere el término, los monitores de máquinas virtuales proporcionan también formas de administrar las máquinas virtuales, no solo crearlas y ejecutarlas. Aquí es donde los grupos de control, frecuentemente abreviados a cgroups, vienen en la imagen.

Grupos de control es una característica del kernel iniciada originalmente en 2006, que permite a los administradores restringir y/o limitar el uso de recursos del sistema para grupos de procesos. Los grupos pueden anidarse en una estructura parecida a un

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

bosque, donde las raíces de los árboles son los grupos por defecto de cada sistema.

Los subsistemas son los controladores que se refieren a tipos de recursos individuales del sistema. Las configuraciones para el grupo de cada subsistema se almacenan en el sistema de archivos (normalmente /cgroups/subsystem/), donde residen los valores predeterminados de los grupos. También se almacenan aquí los ajustes para los grupos secundarios, almacenados de forma recurrente en directorios adicionales. Los subsistemas más comúnmente utilizados son los siguientes:

- Blkio - Control del acceso a los dispositivos bloque IO. Se puede utilizar para limitar la velocidad de lectura / escritura o establecer la prioridad de lectura / escritura de un grupo, ya sea globalmente o por cada dispositivo. Las velocidades se pueden ajustar en IO operaciones o bytes por segundo.
- CPU - Asigna la potencia total de la CPU, ya sea estableciendo la prioridad de la CPU o el tiempo absoluto de un período determinado que los procesos pueden ejecutar.
- Cpusets: especifique las CPUs individuales que el grupo puede usar dispositivos: permite crear una lista blanca que contiene derechos de acceso a dispositivos individuales.
- Memoria - impone límites a la memoria utilizada por todos los procesos de los grupos. Es posible incluir memoria de intercambio.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

- Net_prio: puede reemplazar la opción SO_PRIORITY de los procesos, que se utiliza como nivel de prioridad de los paquetes en las colas de red de Linux.

Además, los cgroups proporcionan un subsistema de congelación, que puede utilizarse para congelar un grupo no root de procesos. Cgroups puede ser utilizado también para el monitoreo, donde cada subsistema relevante mantiene archivos de estadísticas, el seguimiento de cada grupo de consumo de recursos. Un mecanismo de notificación puede usarse para capturar varios eventos, como alcanzar el límite de recursos asignado.

Herramientas de virtualización basadas en contenedor

LXC (contenedores Linux) combina las características ofrecidas por el espacio de nombres y los grupos de control para crear un entorno completamente aislado, que se puede administrar fácilmente, un contenedor. Dado que los contenedores son la unidad con la que LXC trata, se proporcionan las operaciones básicas: crear, iniciar, detener, listar y retirar contenedores. Un contenedor vacío no será de mucha utilidad para nadie, por lo tanto, es posible crear contenedores basados en una plantilla, que configura el sistema de archivos y la configuración inicial del contenedor. LXC viene incluido con plantillas para varias distribuciones principales de Linux. Además, LXC puede congelar y descongelar contenedores, dando la posibilidad de suspender y luego reanudar su ejecución. También es posible crear un punto de control, una información sobre el estado de un contenedor. El contenedor puede ser restaurado

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

posteriormente a su estado anterior, capturado por el punto de control. También es posible la clonación de contenedores. Los contenedores también pueden interactuar de forma programática, ya sea utilizando ganchos de gestión del ciclo de vida de los contenedores o utilizando una API - liblxc, que tiene binging para varios idiomas, incluyendo Java, C, Python, Ruby y Go.

2.2.8 Docker

La idea detrás del proyecto Docker está bien expresada por el siguiente objetivo establecido por su equipo de desarrollo: "Construir el botón 'que permite que cualquier aplicación sea construida y desplegada en cualquier servidor, en cualquier lugar". En su núcleo, Docker es una plataforma de código abierto permitiendo que las aplicaciones se desplieguen dentro de contenedores de software. Esta puesta en marcha desde el Silicon Valley ha capturado rápidamente la atención de las empresas más importantes en el mundo de las TI como: Amazon, Google, Microsoft y Red Hat han añadido soporte para Docker a sus plataformas y continuamente contribuyen al proyecto.

Pero Docker es algo más que una biblioteca de virtualización, abstrae las diferencias entre las distribuciones de sistemas operativos y crea un entorno estandarizado para desarrollar aplicaciones. Un desarrollador de software puede crear una aplicación estandarizada que se convierte en portátil y puede ejecutarse en todas partes donde se instala el motor Docker. Esto ahorra mucho trabajo para el autor ya que ya no es necesario soportar muchas plataformas y distribuciones de sistemas operativos diferentes. Los administradores de sistemas necesitan dedicar menos

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

tiempo a la configuración de la aplicación, ya que viene con todas sus dependencias. El hecho de que cada aplicación se ejecuta en su propio contenedor resuelve muchos problemas comunes como desinstalar o reemplazar completamente o cuando dos aplicaciones requieren dos versiones diferentes de la misma dependencia.

Docker es una pieza relativamente nueva de la tecnología, se ejecuta tanto en sistemas operativos Linux, Mac y Windows. Docker se ejecuta nativamente en Linux, y aunque es posible usar software adicional para ejecutarlo en Windows u OSX.

Historia de Docker

El proyecto fue lanzado inicialmente como open source en dotCloud en marzo de 2013. Dos meses después, se lanzó el registro público de Docker. En la segunda mitad del año, Google, Yandex y Baidu (motores de búsqueda más utilizados en Rusia y China) han integrado Docker en sus servicios en la nube.

Docker entró en 2014 al completar un fondo de 15 millones de dólares, lo que le permitió invertir fuertemente tanto en el proyecto de código abierto como en el plan de apoyo empresarial, así como en la expansión de la plataforma comunitaria. En abril, LXC fue eliminado como el entorno de ejecución por defecto a favor del propio libcontainer de Docker. El mes que viene, Ubuntu 14.04 se convirtió en la primera distribución de Linux de calidad empresarial que se envía con Docker nativamente empaquetado, trayendo a millones de servidores Ubuntu a no más de tres comandos de usar contenedores Docker. La versión 1.0 fue finalmente lanzada en junio

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

en la primera conferencia centrada en Docker - DockerCon. En septiembre se anunció que se había recaudado otro importante fondo de 40 millones de dólares, valorando el proyecto en unos 400 millones de dólares. Un mes más tarde, Docker y Microsoft declararon su asociación con el objetivo de crear Docker Engine para Windows Server y un modelo de contenedor multi-Docker, incluyendo soporte para aplicaciones que constaban de contenedores Linux y Windows Docker. En diciembre tuvo lugar en Amsterdam la primera conferencia oficial de Docker en Europa, anunciando varios nuevos proyectos relacionados con Docker, así como Docker Hub Enterprise. Docker terminó el año 2014 con el lanzamiento de 1.4.

En febrero de 2015, se lanzó la versión 1.5, con soporte para IPv6, contenedores de sólo lectura y soporte de múltiples Dockerfiles por proyecto. Poco después, un trío de herramientas de orquestación fue anunciado: Docker Machine, Docker Swarm y Docker Compose. La versión estable actual (17) fue lanzada en abril del presente año.

Daemon Docker

Internamente Docker utiliza un modelo cliente-servidor, donde el servidor es un daemon que puede ejecutarse en un sistema completamente diferente al cliente. El daemon puede iniciarse utilizando el comando Docker al pasar el indicador -d, o iniciar un servicio con Docker de inicio de systemctl o inicio del Docker de servicio. Se requiere una cuenta privilegiada para ejecutar Docker en modo daemon, aunque se hace un esfuerzo significativo para eliminar este inconveniente, por lo que incluso los usuarios regulares podrían ejecutar contenedores.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Libcontainer

En Docker 0.9 se introdujo un concepto de controladores de ejecución y se admitieron dos controladores: el controlador LXC, que utilizaba el anterior liblxc requerido y el controlador nativo utilizando la biblioteca propia de Docker, libcontainer. Está escrito puramente en Go y maneja la administración de contenedores, usando las capacidades de kernel mencionadas anteriormente como espacios de nombres y grupos de control. Se espera que Libcontainer sea portado a otros idiomas y que soporte sistemas operativos, en la versión actual (17) a eliminando la necesidad de herramientas adicionales como Boot2docker cuando Docker se está ejecutando en Windows.

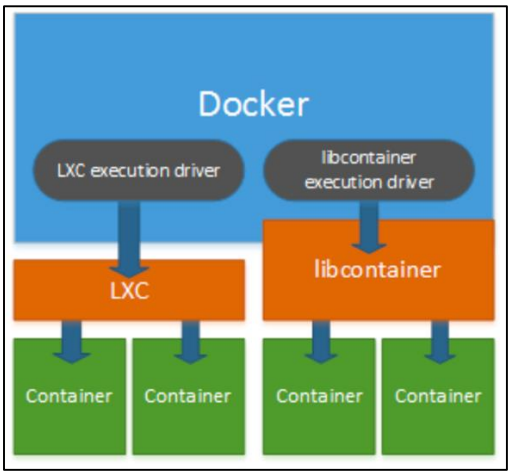

Figura 9. Ejecución de drivers en Docker

Fuente: (Sitio web Docker Inc, 2017)

Sistema de archivos capas en Docker

Al lanzar un contenedor, Docker utiliza un mecanismo llamado unión mount los sistemas de archivos no están montados en lugares diferentes, pero

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

encima uno del otro, por lo tanto, un contenido de directorio puede estar compuesto de archivos de directorios de diferentes sistemas de archivos.

En Docker, las aplicaciones suelen especificar una imagen principal. Por ejemplo, una aplicación web podría depender de un servidor web específico, que en contraste dependería sólo de un sistema operativo. Esto significa que cada imagen agrega una nueva capa de sólo lectura en el sistema de archivos encima de la capa de su padre. Una vez que se inicia la aplicación como contenedor, se coloca una capa adicional de escritura en la parte superior. Cuando se necesita cambiar un archivo de una capa de sólo lectura, este archivo se copia en la capa de escritura y se realiza el cambio allí. Es importante que los cambios en la capa superior persistan incluso después de que el contenedor salga y por lo tanto sigan en efecto durante su siguiente ejecución.

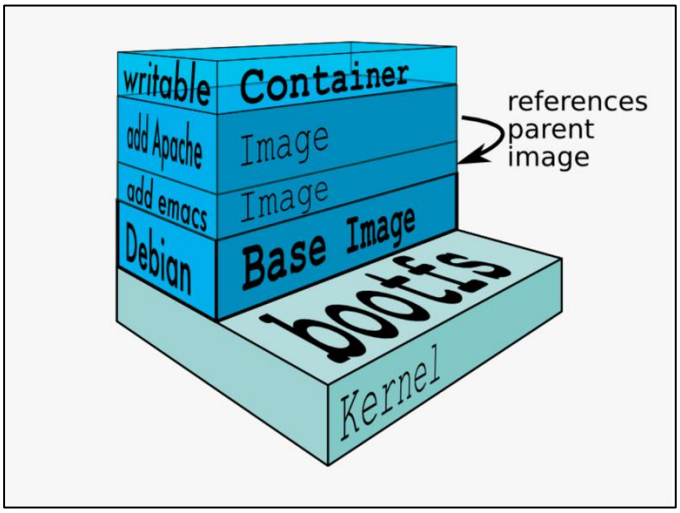

Figura 10. Sistema de archivos de Docker (Filesystem)

Fuente: (Sitio web de Docker Inc, 2017)

Los volúmenes de datos son una excepción al archivo de unión existente. Ellos dan paso al intercambio de datos entre contenedores, realizando

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

cambios directos al sistema de archivos y estos cambios no se incluyen al realizar actualizaciones a la imagen utilizada. Uno puede crear un volumen de datos vacío o montar un archivo o directorio del host. Un volumen de datos puede ser añadido a un contenedor específico o puede crearse un contenedor de volumen de datos dedicado, que puede no sólo ser compartido entre contenedores, sino que también ofrece funciones de avance tales como migraciones, copias de seguridad y restauraciones por parte de Docker.

Seguridad

Docker actualmente requiere privilegios de root, por lo tanto, si se compromete, el host también estará expuesto. Principalmente debido a esta amenaza de seguridad, una de las metas para Docker es la capacidad de los usuarios no root para ejecutar contenedores. Docker ha trabajado en ello, cambiando la arquitectura a dos demonios. El daemon actual se ejecuta en el espacio del usuario, mientras que las operaciones confidenciales serán reenviadas a un nuevo servicio en el espacio del kernel. En diciembre de 2014, Docker promovió una nueva característica llamada firma de imágenes. Hace mucho que se pedía que las imágenes contaran con una firma criptográfica, para que se verifiquen antes de su ejecución. Sin embargo, una inspección detallada de la implementación reveló que "el reporte de Docker de que se verifica una imagen descargada se basa únicamente en la presencia de un manifiesto firmado y Docker nunca verifica la suma de comprobación de la imagen del manifiesto. Un atacante podría proporcionar cualquier imagen junto con un manifiesto firmado. [23] Incluso se encontraron problemas adicionales como el tarsum

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

mal construido utilizado para la verificación de la imagen, el procesamiento del manifiesto después de la extracción de la imagen o el hecho de que si el manifiesto es incorrecto sólo se emite una advertencia y la imagen sigue correr. En respuesta a los descubrimientos mencionados, el equipo de Docker inició una auditoría de seguridad y prometió revisar las recomendaciones de Docker.

Rendimiento

Según Felte, Ferreira, Rajamony y Rubio (2014) en un informe de investigación de IBM, indican que Las pruebas que comparaban Docker con las máquinas virtuales cubrían el acceso a la memoria, la E/S en bloque, la creación de redes y el benchmarking de instancias Redis y MySQL. Teniendo en cuenta las diferencias entre las dos tecnologías de virtualización, los resultados confirmaron lo esperado: Tanto las VM como los contenedores son una tecnología madura que se ha beneficiado de una década de optimización incremental de hardware y software. En general, Docker iguala o supera el rendimiento de KVM en todos los casos probados. Los resultados demuestran que tanto KVM como Docker introducen una sobrecarga insignificante para el rendimiento de la CPU y la memoria (excepto en casos extremos). Las pruebas también han revelado que el rendimiento de Docker puede diferir significativamente dependiendo de si está usando la red del anfitrión o un Puente NAT. Del mismo modo, la sobrecarga es más lenta cuando los datos se almacenan en un volumen compartido en lugar de la unión del sistema de archivos.

Docker ha inspirado la creación de muchos proyectos que aprovechan su funcionalidad. Mientras que estos proyectos están en desarrollo, todavía

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

están surgiendo nuevos, ya que todo el mundo quiere llenar una brecha en el mercado lo antes posible. Aún queda por ver, cuáles surgirán como ganadores de este ambiente competitivo, mencionaremos los que parecen mostrar las premisas más grandes.

Kubernetes

Google, sin duda uno de los mayores operadores de centros de datos, también ha admitido que cada uno de sus servicios se ejecuta dentro de un contenedor Linux. Aunque todavía no han compartido su planificador de tareas interno principal, Omega; han lanzado otro gestor de contenedores Kubernetes, como un proyecto de código abierto. En la actualidad, los contenedores se ejecutan dentro de Docker.

Kubernetes introduce el concepto de Pods - grupos de contenedores que están relativamente y estrechamente acoplados y son tratados como la unidad más pequeña desplegable. Un caso de uso común sería un contenedor principal que ejecuta una aplicación web que utiliza varios otros contenedores con sus servicios auxiliares. Tiene sentido para estos contenedores para iniciar/detener al mismo tiempo y para ejecutar en el mismo host. Las vainas normalmente tienen etiquetas (clave, pares de valores), lo que permite consultar los nodos del clúster que ejecutan un grupo particular de cápsulas.

La siguiente abstracción es Servicio: grupos de puertos definidos por etiquetas, que exponen el mismo puerto y ejecutan la misma aplicación. Un pod puede dejar de ejecutarse en un nodo y ser replicado en otro, por lo que, con el uso de un servicio, otros pods no tiene que seguir la pista, que

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

pod está ejecutando de qué nodo, en su lugar, pueden utilizar una IP virtual de la Servicio.

Para admitir el escalado, Kubernetes introduce controladores de replicación. Un controlador de replicación utiliza una plantilla, según la cual se crean las cápsulas y permite establecer el número de réplicas de Pods a ejecutar. Esto se puede utilizar para aumentar o disminuir fácilmente el número de instancias en ejecución de una aplicación. Otro caso de uso sería la actualización de una aplicación, al utilizar un controlador de replicación para la versión antigua y otra para la nueva versión, se podría lograr un tiempo de actividad continuo.

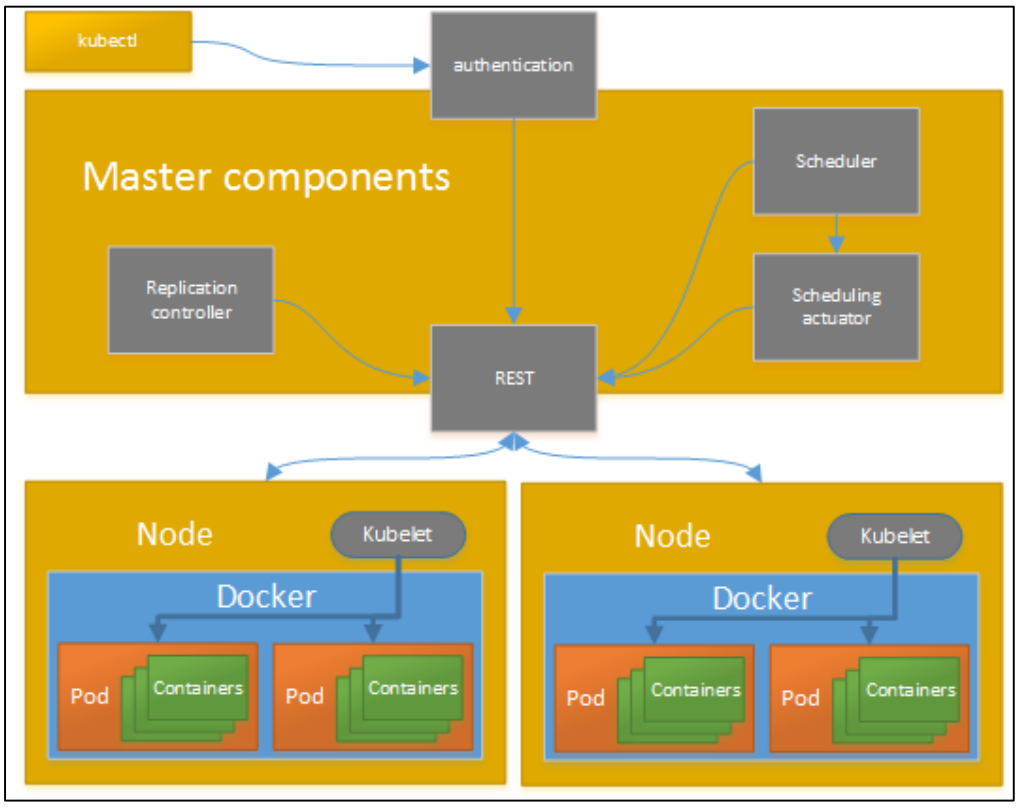

Figura 11. Arquitectura Kubernetes

Fuente: (Google Inc, 2017)

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Coreos

CoreOS es un sistema operativo de código abierto basado en el sistema operativo Chrome de Google. Con su primer lanzamiento en el 2013 comenzó a empujar su visión de un sistema operativo centrado en el contenedor. Esto significa que CoreOS no incluye un gestor de paquetes, pero las aplicaciones se proporcionan en lugar de los medios de los contenedores. Docker se utiliza actualmente como gestor de contenedores.

CoreOS intenta apuntar la infraestructura de la nube con dos utilidades clave, etc.d y flota. Etcd es un daemon de gestión de configuración de clúster compartido, que proporciona una API para la propagación de cambios de configuración en todo un clúster de instancias de etcd. Por otro lado, la flota es un daemon de control systemd de nivel de agrupación. Permite desplegar contenedores ya sea globalmente (en todas las máquinas del clúster) o en una sola máquina con soporte para conmutación por error.

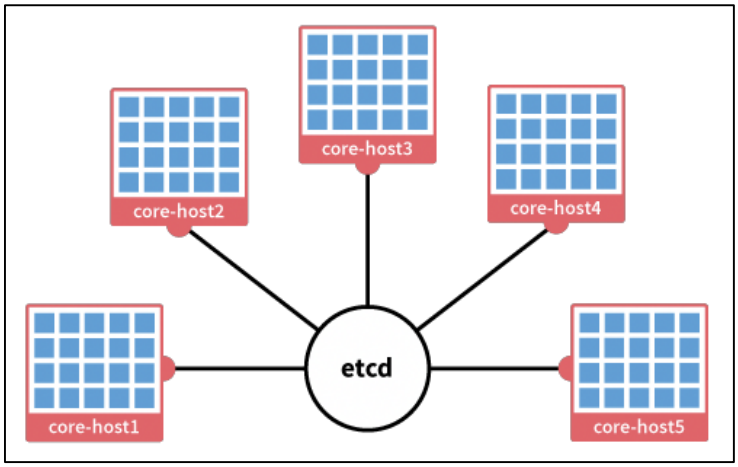

Figura 12. Cluster de hosts CoreOs con contenedores

Fuente: (CoreOS, 2017)

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Rkt

En diciembre de 2014, el equipo de CoreOS ha expresado su preocupación por la dirección que el proyecto Docker ha tomado y lanzó una entrada en el blog, lo que inmediatamente llamó la atención. Expresaron su desacuerdo con la amplitud del alcance de Docker:

"Cuando Docker nos fue presentado a principios de 2013, la idea de un contenedor estándar era llamativa e inmediatamente atractiva: un componente simple, una unidad componible, que podría utilizarse en una variedad de sistemas. El repositorio Docker incluyó un manifiesto de lo que debería ser un contenedor estándar". "Pensamos que Docker se convertiría en una unidad simple en la que todos podamos estar de acuerdo". "Se quitó el manifiesto de contenedor estándar. Deberíamos dejar de hablar de los contenedores Docker, y empezar a hablar de la plataforma Docker. No se está convirtiendo en el simple bloque componible que habíamos imaginado".

Esta crítica probablemente fue dirigida al trío de proyectos, Docker Machine, Docker Swarm y Docker Compose, que fueron lanzados en 2015. También afirman que el modelo arquitectónico de Docker, donde todo corre a través de un daemon central, es "fundamentalmente defectuoso" de La perspectiva de la seguridad. Dichos argumentos generaron discusiones si Docker intentaba o no hacer demasiado y convertirse en una gran plataforma monolítica, sin proporcionar una modularidad razonable, o sólo ha entrado en el área de negocios de otros proyectos / startups y este blog es un ejemplo de un Intento de defensa.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Además, el post contenía un anuncio de Rocket - un tiempo de ejecución de nuevos contenedores y un competidor directo a Docker. La visión de CoreOS es centrarla alrededor de una especificación para contenedores y el "ejecutor de Contenedor de Aplicación", que incluyeron y animaron a los desarrolladores a enviar comentarios para. Más tarde, fue renombrada a rkt - "rock it". Rkt está actualmente bajo un gran desarrollo, con muy poca documentación o ejemplos, pero ya permite ejecutar imágenes nativas de Docker.

El equipo también ha expresado que es posible que Docker aplique su especificación App Container, haciendo que los dos proyectos sean interoperables. Una declaración, que CentOS continuará enviando ya que Docker fue incluido también.

2.3 MARCO CONCEPTUAL

2.3.1 Contenedores

Los contenedores de software son un paquete de elementos que permiten ejecutar una aplicación determinada en cualquier sistema operativo.

2.3.2 Docker

Es un proyecto de código abierto capaz de automatizar el despliegue de aplicaciones dentro de contenedores de software, proporcionándonos así una capa adicional de abstracción y automatización en el nivel de virtualización de sistema operativo sobre Linux. Docker hace uso de las utilidades de aislamiento de recursos del núcleo de Linux como los cgroups y espacios de nombre del Kernel, para permitir que "contenedores" independientes se ejecuten como una instancia única de Linux, evitando

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

así la elevada sobrecarga en el arranque y mantenimiento de máquinas virtuales.

2.3.3 Imágenes

Una imagen es una especie de plantilla, una captura del estado de un contenedor. Ya comenté que un contenedor no es una máquina virtual, pero para que te hagas una idea, podríamos decir que una imagen de un contenedor es como un snapshot de una máquina virtual, pero mucho más ligero.

2.3.4 Microservicios

Es una aproximación para el desarrollo software que consiste en componer una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP). Cada servicio se encarga de implementar una funcionalidad completa del negocio (principio único de responsabilidad). Cada servicio es desplegado de forma independiente y puede estar programado en distintos lenguajes y usar diferentes tecnologías de almacenamiento de datos.

2.3.5 Plataforma

Se refiere al sistema operativo o a sistemas complejos que a su vez sirven para crear programas, como las plataformas de desarrollo.

2.3.6 RESTful

La Transferencia de Estado Representacional (en inglés Representational State Transfer) o REST es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo.

2.3.7 Virtualización

Es la creación a través de software de una versión virtual de algún recurso tecnológico, como puede ser una plataforma de hardware, un sistema operativo, un dispositivo de almacenamiento u otros recursos de red.

2.3.8 Web Service

Un servicio web (en inglés, web service o web services) es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de computadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares. Es una máquina que atiende las peticiones de los clientes web y les envía los recursos solicitados.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

CAPÍTULO III

METODOLOGÍA

3.1 TIPO DE INVESTIGACIÓN

Consiste en trabajos experimentales o teóricos que se emprenden principalmente para adquirir nuevos conocimientos, pero fundamentalmente dirigidos hacia un objeto práctico específico, determina la manera como el conocimiento pueden general aplicaciones tecnológicas para el aprovechamiento de una oportunidad. Las investigaciones aplicadas a menudo tratan de llevar a cabo experimentos y construir implementaciones de prueba de principio, y recolectar experiencias de ellos.

3.2 METODOLOGIA

3.2.1 Metodología XP

La metodología XP (Xtreme Programing) o metodología de programación extrema, es una metodología ágil para el desarrollo de software, que se centra en la programación de la solución de software, siendo una buena solución para los proyectos donde los requerimientos están cambiando constantemente.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

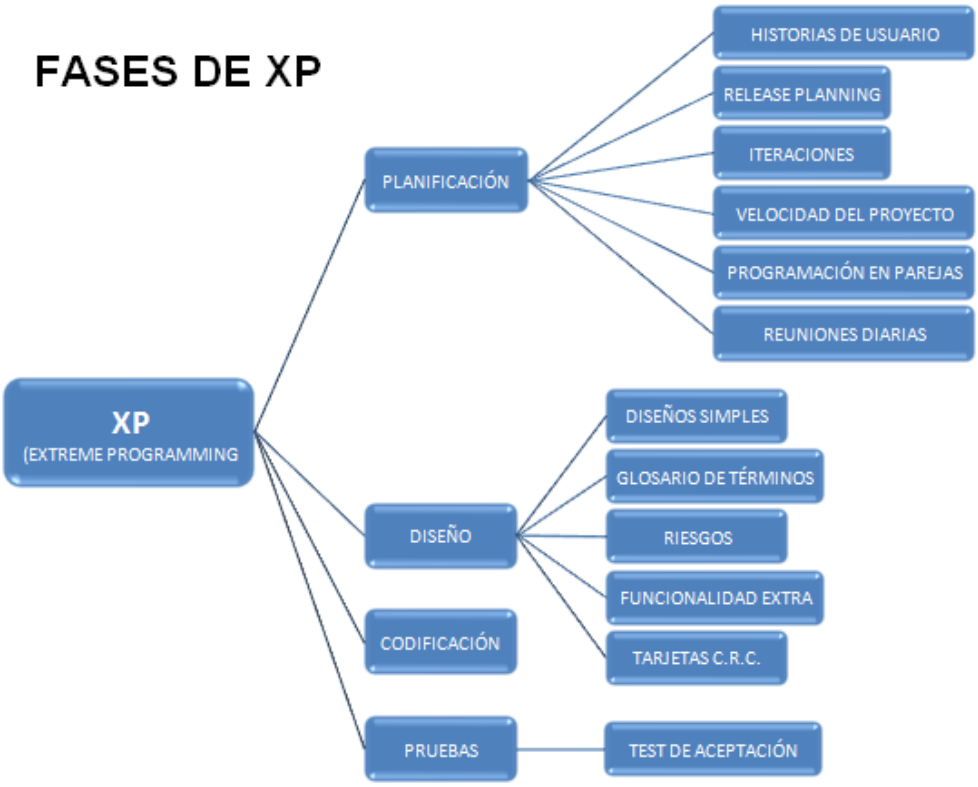

Figura 13. Etapas de la metodología XP

Fuente: Sachez (2017)

Consta de las siguientes Etapas: Planificación, Considerada como un dialogo continuo entre las partes involucradas del proyecto, comienza recopilando “historias de usuarios”, para luego ser evaluados en función al tiempo de desarrollo. Diseño: la metodología empleada hace especial énfasis en los diseños simples y claros, que requiera menos tiempo y esfuerzo de desarrollo. Implementación: basado en estándares de codificación y normas que ayudan a mantener el código legible y fácil de mantener y refactorizar. Pruebas: los módulos deben pasar las pruebas unitarias antes de ser liberados o publicados, además de ser definidas antes de realizar el código.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

3.2.2 Historia de usuario y Tarea de ingeniería (Task Card)

En el desarrollo del prototipo se utilizó las historias de usuario para especificar los requisitos del software, es un formato en el que el cliente describe brevemente las características que el sistema debe poseer, en este caso requisitos no funcionales.

Cuadro 1. Planitilla para historias de usuario

HISTORIA DE USUARIO	
Número:	**Usuario**:
Nombre de historia de usuario:	
Puntos estimados:	**Iteración asignada**:
Programador responsable:	
Descripción:	
Observaciones:	

Fuente: (Letelier & penades, 2006)

Unas historias de Usuario se descomponen en varias tareas de ingeniería, que describen que actividades se realizarán en cada historia de usuario, asimismo las tareas de ingeniería se vinculan mas al desarrollador, ya que permite tener un acercamiento con el código (Ferreira Escutia, 2013), en el cuadro 2, se muestra el formato.

Cuadro 2. Plantilla para tareas de ingeniería

TAREA DE INGENIERÍA	
Número de Tarea: Permite identificar a una tarea de ingeniería.	**Número de Historia**: Número asignado de la historia correspondiente.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Nombre de Tarea: Describe de manera general a una tarea de ingeniería.	
Tipo de tarea: Tipo al que corresponde la tarea de ingeniería.	**Puntos estimados**: número de días que se necesitará para el desarrollo de una tarea de ingeniería.
Fecha de inicio: Fecha inicial de la creación de la tarea de ingeniería	**Fecha fin**: fecha cuando la tarea de ingeniería es concluida.
Programador responsable:	
Descripción: Información detallada de la tarea de ingeniería.	

Fuente: (Ferreira, 2013)

3.2.3 Modelado de Arquitectura de aplicaciones

Es un diseño que describe los componentes principales del sistema y el modo en que interactúan entre sí para lograr los objetivos del diseño, están implicadas actividades: Descripción de requisitos, modelos arquitectónicos, componentes y sus interfaces e interacciones entre componentes.

Para el desarrollo de la arquitectura se utilizó Microservicios como estilo arquitectural, con el que se creó el modelo de arquitectura para el desarrollo y despliegue que proponemos, estas podrán ser usadas en aplicaciones nuevas o para la descomposición de aplicaciones monolíticas, ya que cuando el número de Microservicios incrementa necesitamos tener un modelo que permita su crecimiento organizado y en las capas que se requieran.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

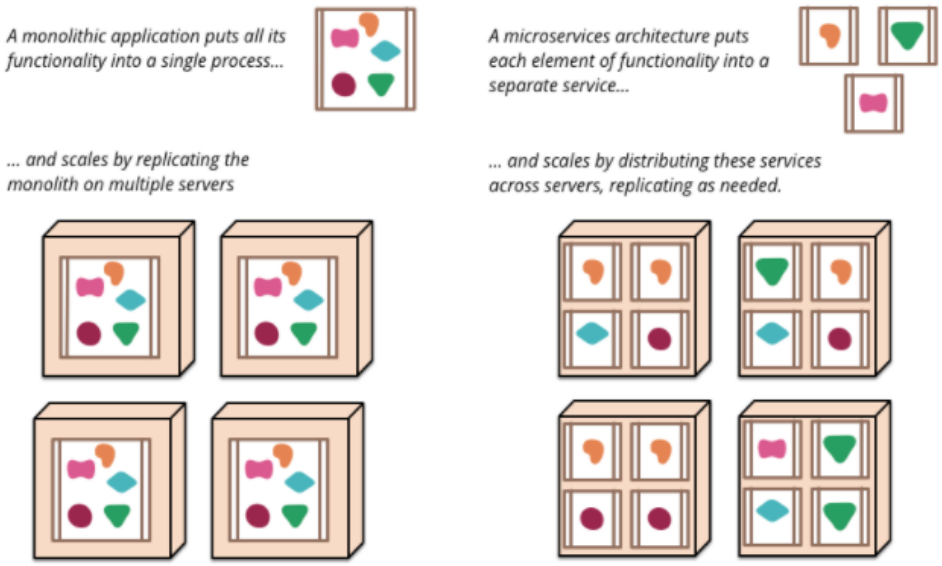

Figura 14. Sistemas Monolíticos y Microservicios

Fuente: (Fowler, 2014)

La investigación se ha organizado en tres grandes grupos:

3.2.4 Pruebas Unitarias

Es un método que consiste en probar aplicaciones para certificar su buen funcionamiento en el proceso de construcción del software, Unit Testing se concibe como la parte más pequeña de una aplicación susceptible de ser testeada, el testing de unidad es la metodología por la cual se testean de forma independiente estas unidades. (Gonzales, A., Álvarez, D., y Acosta, L., 2012), la ejecución de las pruebas unitarias permitirá asegurar el correcto proceso de desarrollo.

3.3 MATERIALES EMPLEADOS

3.3.1 GNU/Linux

GNU/Linux es un sistema operativo constituido por el núcleo (Linux) y las herramientas (GNU) que se requieren para su funcionamiento, es portable con versiones disponibles para su ejecución en diversos dispositivos de

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

sistemas de computo, como en servidores. A diferencia de otros sistemas operativos, su código fuente está disponible de forma gratuita, lo cual permite que los programadores lo configuren para ejecutar cualquier dispositivo y cumpla cualquier especificación. Es altamente modular, permitiendo la carga y descarga de múltiples módulos sobre demanda, haciéndolo un sistema operativo técnicamente robusto. Para el despliegue se utilizó este sistema operativo como una instancia EC2 de Amazon WS.

3.3.2 macOS

Es un sistema operativo creado por Apple para los computadores creados por la misma marca, para equipos de escritorios, portátiles y servidores, proporcionan gran estabilidad y productividad, a ser basado en BSD (Unix) es altamente compatible con sistemas operativos GNU/Linux, en la investigación se ha utilizado la versión 10.12.6 denominado “macOS Sierra” con el objetivo de preparar un entorno de desarrollo.

3.3.3 Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) proporciona capacidad de computación escalable en la nube de Amazon Web Services (AWS). El uso de Amazon EC2 elimina la necesidad de invertir inicialmente en hardware, de manera que puede desarrollar e implementar aplicaciones en menos tiempo. Puede usar Amazon EC2 para lanzar tantos servidores virtuales como necesite, configurar la seguridad y las redes, y administrar el almacenamiento. Amazon EC2 le permite escalar hacia arriba o hacia abajo para controlar cambios en los requisitos o picos de popularidad, con lo que se reduce la necesidad de prever el tráfico. Se utilizó para la implementación de los servicios web en la nube.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

3.3.4 Docker

Es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporciona una capa de abstracción y automatización de virtualización de las aplicaciones en distintos sistemas operativos, para este proceso utiliza aislamiento de recursos del Kernel Linux, como cgrups y namespaces para que los contenedores se ejecuten dentro de una sola instancia de Linux, de es manera se evita la sobrecarga de iniciar y mantener maquinas virtuales. Hemos utilizado esta tecnología para contenerizar las aplicaciones del modelo propuesto.

3.3.5 Nginx

Nginx es un servidor HTTP de alto rendimiento, libre y de código abierto, así como un servidor proxy IMAP/POP3. NGINX es conocido por su alto rendimiento, estabilidad, amplio conjunto de funciones, configuración simple y bajo consumo de recursos. NGINX es uno de los pocos servidores escritos para abordar el problema C10K. A diferencia de los servidores tradicionales, Nginx no depende de los hilos para manejar las solicitudes. En su lugar, utiliza una arquitectura mucho más escalable impulsada por eventos (asíncrona). Esta arquitectura usa cantidades pequeñas, pero más importantes, predecibles de memoria bajo carga. Incluso si no espera manejar miles de solicitudes simultáneas, aún puede beneficiarse del alto rendimiento y la pequeña huella de memoria de NGINX. NGINX escala en todas las direcciones: desde el VPS más pequeño hasta los grandes grupos de servidores. Se configuró Nginx como un proxy inverso al servidor web.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

3.3.6 Gunicorn

El "Unicornio verde" de Gunicorn es un servidor HTTP de la Interfaz de puerta de enlace del servidor web de Python (WSGI). El servidor de Gunicorn es ampliamente compatible con varios frameworks web, simplemente implementados, livianos en los recursos del servidor y bastante rápido. Se ha utilizado este servidor para interpretar código dinámico generado con el lenguaje Python.

3.3.7 Flask

SQLAlchemy es el conjunto de herramientas de Python SQL y Object Relational Mapper que ofrece a los desarrolladores de aplicaciones toda la potencia y flexibilidad de SQL. Proporciona un conjunto completo de conocidos patrones de persistencia a nivel empresarial, diseñados para un acceso eficiente y de alto rendimiento a la base de datos, adaptados a un lenguaje de dominio simple y basado en Python. Se utilizó para la persistencia de datos.

3.3.8 SQLAlchemy

SQLAlchemy es el conjunto de herramientas de Python SQL y Object Relational Mapper que ofrece a los desarrolladores de aplicaciones toda la potencia y flexibilidad de SQL. Proporciona un conjunto completo de conocidos patrones de persistencia a nivel empresarial, diseñados para un acceso eficiente y de alto rendimiento a la base de datos, adaptados a un lenguaje de dominio simple y basado en Python. Se utilizó para la persistencia de datos.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

3.3.9 PostgreSQL

PostgreSQL es una base de datos relacionales open-source. Es gratuito y libre, ofrece una gran cantidad de opciones avanzadas, siendo considerado el motor de base de datos más avanzado en la actualidad. Una característica importante de PostgreSQL es el control de concurrencias multiversión; o MVCC por sus siglas en inglés. Este método agrega una imagen del estado de la base de datos a cada transacción. Permite hacer transacciones eventualmente consistentes, ofreciéndonos grandes ventajas en el rendimiento, brinda mayor escalabilidad. Se ha configurado PostgreSQL para ser contenerizado y se vinculará con un servicio (usuarios).

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

En este capítulo presento los resultados y discusión concerniente a cada objetivo específico.

4.1 RESULTADOS CONFORME AL OBJETIVO ESPECÍFICO 1

Los requisitos de software y arquitectura, contiene una visión del problema desde la arquitectura Microservicios, primeramente, se expresó a través de la selección de tecnologías e infraestructura que soporte los servicios web, tanto para su desarrollo como para su despliegue.

4.1.1 Selección de tecnologías

Para definir las tecnologías adecuadas se ha tenido como referencia un lenguaje de programación ágil, en tal sentido las propuestas son en base a herramientas para el lenguaje de programación Python:

Cuadro 3. Selección de tecnologías web

Tecnología	**Nombre de la selección**
Servidor web proxy:	Nginx 1.12.1
Lenguaje de programación:	Python 3.5

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Framework de desarrollo:	Flask 1.11.4
Base de datos:	PosgreSQL 9.6.3
Servicio Cache:	Redis 4.0.1

4.1.2 Configuración y creación de los entornos de desarrollo

Para crear el entorno reproducible se han creado contenedores Docker 17.06.0-ce para cada un servicio.

Se realizó la configuración local con las versiones de Docker y herramientas mostradas en el anexo 1.

Modelo de arquitectura

Se programaron las aplicaciones web con el estilo arquitectónico de Microservicios, los mismos se hicieron de forma vertical considerando las siguientes partes:

- **Servicios Core**

  En esta primera capa, se define la persistencia, reglas y lógica del negocio.

- **Servicios compuestos**

  Tiene como objetivo orquestar un número de Servicios Core, para llevar a cabo una tarea o agregar información a un conjunto de Servicios Core.

- **Servicios Interfaz de programación de aplicaciones (API)**

  Se definieron aplicaciones web con el objetivo de exponer las funcionalidades permitidas al exterior, como a otros sistemas de información consumidores o a otros externos a la entidad.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

En relación a los dominios funcionales a los que pertenecen, se han dividido de manera horizontal.

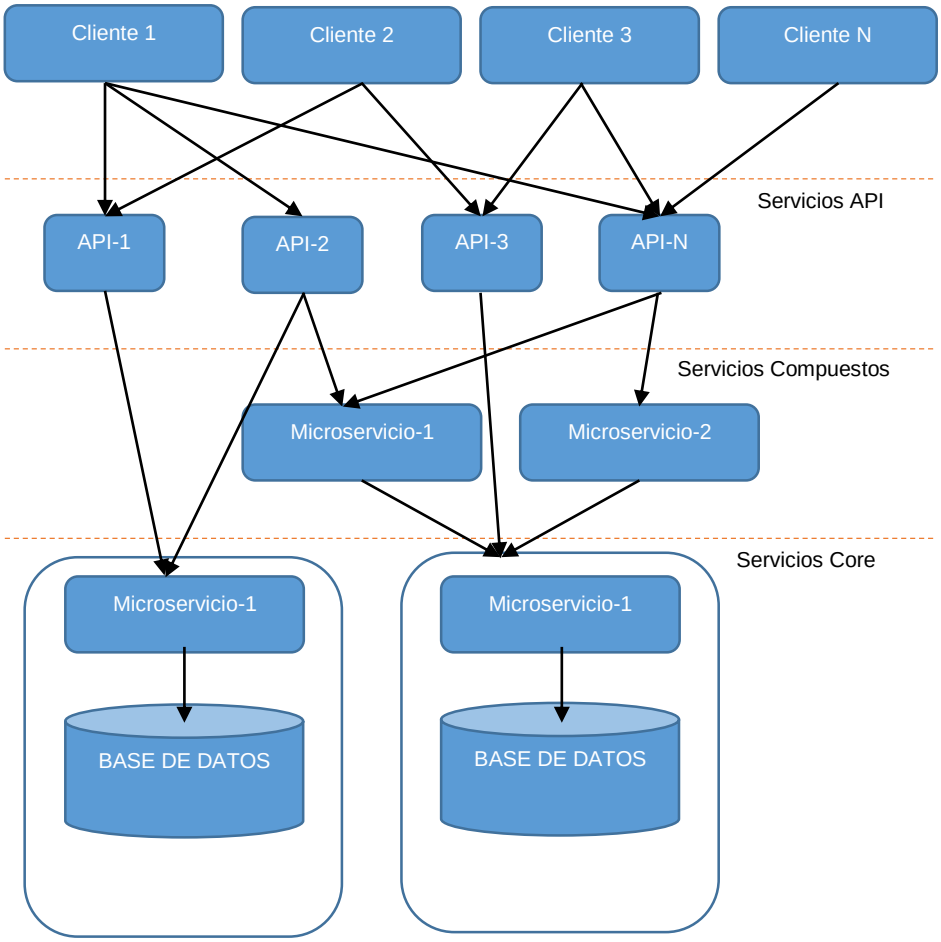

Figura 15. Modelo de arquitectura basado en microservicios

Modelo de arquitectura para el desarrollo de aplicaciones web basados en Microservicios.

En este modelo se considera arquitectura de organización de los diversos servicios para luego ser empaquetados/encapsulados para su reproductibilidad en el equipo de desarrollo.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

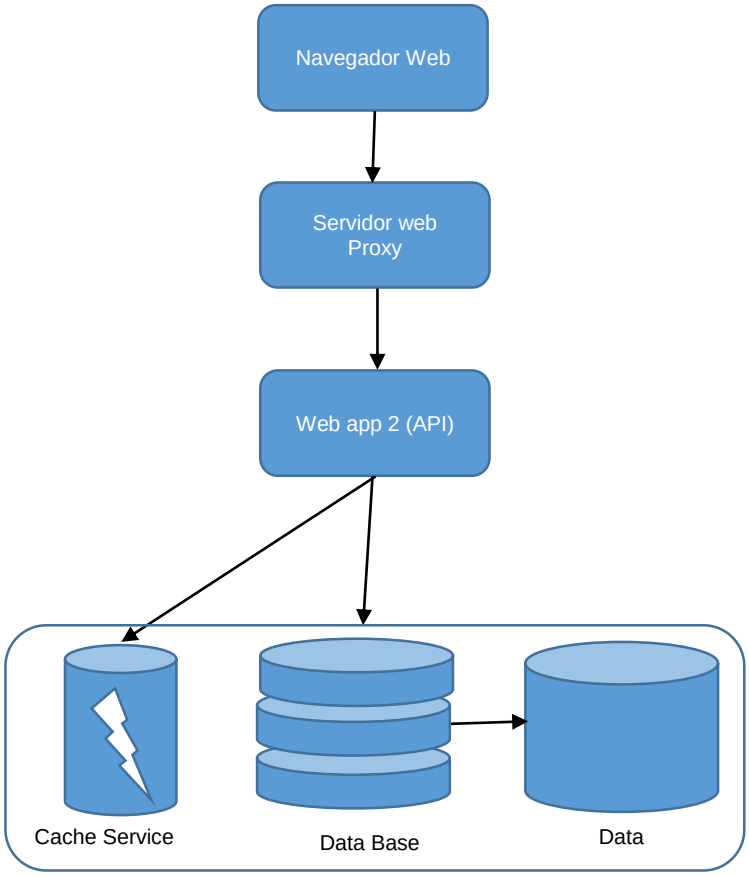

Figura 16. Arquitectura para el desarrollo de Aplicaciones Web

Discusión

Existen diferentes entornos de virtualización, desde la virtualización de aplicaciones, en Python se usa por default virtualenv, al igual que herramientas de virtualización de entornos de desarrollo como Vagrant, sin embargo las tecnologias emergentes contenerizan aplicaciones, como es el caso de Docker.

Docker permite además de tener un contenedor a medida que encapsula la arquitectura propuesta para el desarrollo de aplicaciones web, lo cual permite la reproductibilidad de entornos de desarrollo.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

4.2 RESULTADOS CONFORME AL OBJETIVO ESPECÍFICO 2

En este objetivo se alcanzó a diseñar la arquitectura para la implementación y despliegue en los siguientes modelos:

Modelo de arquitectura de implementación.

En este modelo se consideran las tecnologías que implementamos en cada capa. Para los servicios web se usa el servidor de aplicaciones Gunicorn, que interpreta scripts de Python, que es el lenguaje de programación elegida en esta implementación.

La implementación de los componentes del modelo fue:

Cuadro 4. Componentes

Componente	Full Stack Python
Servidor proxy reverso	Nginx
Enrutamiento dinámico y balanceqdor de carga.	Gunicorn y Docker Compose
Microservicio	Flask
Intercambio de Mensajes	RESTful

En el anexo 2, se visualiza la composición de cada servicio.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

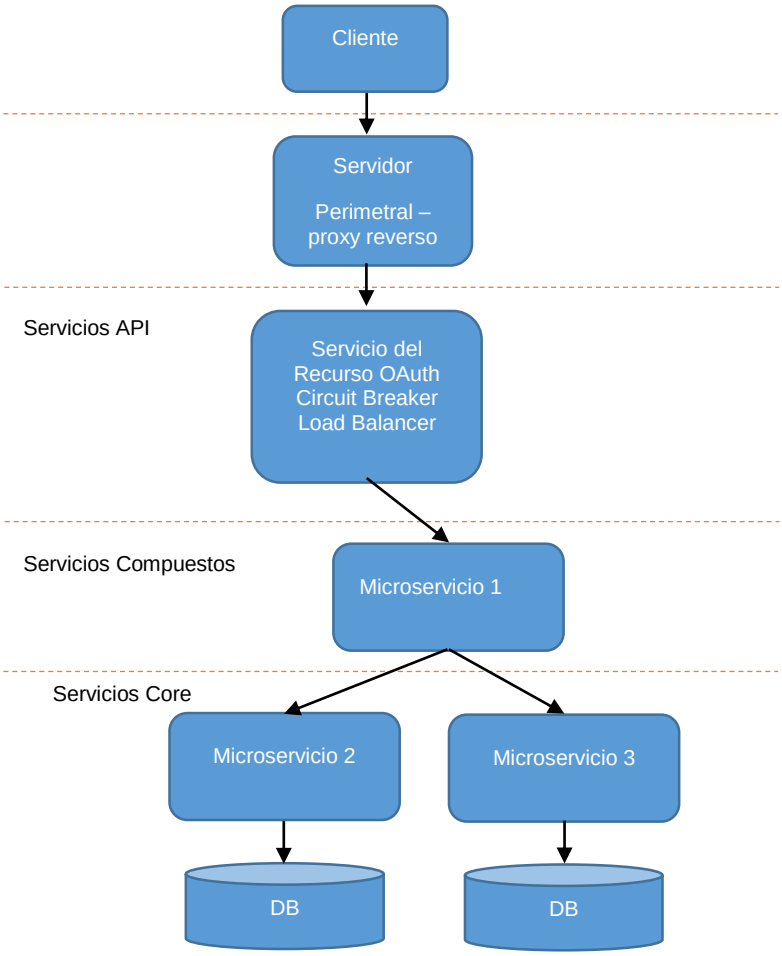

Figura 17. Arquitectura de implementación

Modelo de Despliegue

Para el despliegue en entorno local (Data Center Privado) y entornos de Cloud Computing (EC2 de Amazon Web Service) se tomó como modelo el estilo arquitectural basado en el modelo de arquitectura de Microservicios, los cuales requirieron ser aislados y desplegados en entornos que reduzcan el consumo elevado de recursos en los anfitriones, por lo que se usó la tecnología de Docker-machine, para entornos de desarrollo, despliegue y ejecución de aplicaciones encapsuladas, para ello modelamos la siguiente arquitectura:

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

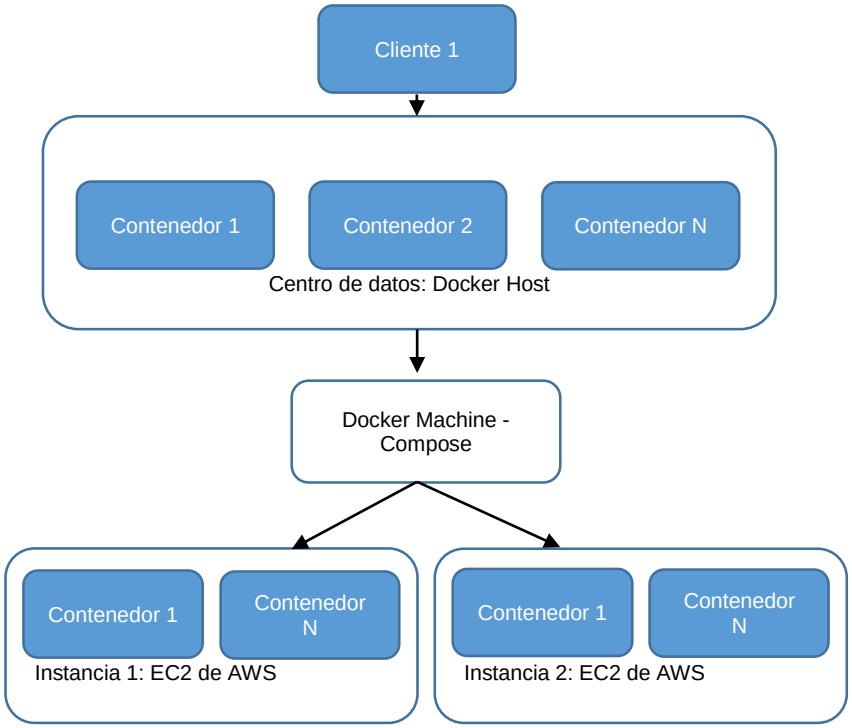

Figura 18. Arquitectura de despliegue

4.2.1 Discusión

De acuerdo a la arquitectura de despliegue planteado en la investigación, el despliegue con esta nueva propuesta, el tiempo se reduce hasta en 14 minutos con un ancho de banda de conexión a internet de 300kb/s, esto es realmente considerable ya que de la manera tradicional se requiere mucho tiempo que podría tomar hasta una jornada mínima de trabajo, por lo que su utilización es pertinente, no solo se trata de tiempo de subida, si no también de la configuración de todo el ecosistema, generalmente trae problemas con gestión de la configuración y las dependencias de bibliotecas utilizadas.

Se realizó el caso de estudio en un equipo de desarrolladores web, utilizando los dos estilos arquitectónicos, el monolítico y microservicios,

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

teniendo en cuenta que en ambos casos se utilizaron los mismos escenarios para luego realizar un análisis comparativo entre ambos estilos, tomando en consideración lo siguiente:

Características de la Aplicación Web

- Se implementó un servidor de autenticación utilizando el protocolo OAut2 y un módulo de gestión de roles y permisos de usuarios.
- Se tiene el home de la aplicación web, para autenticación se tiene que redirigir al registro de inicio de sesión, donde se le asigna los roles y permisos necesarios.
- La aplicación fue desarrollada utilizando las arquitecturas: Microservicios y monolítico, usando para el primero la tecnología de contenedores sobre servidores virtualizados y en la nube; para el segundo únicamente un servidor virtualizado.

Escenarios de ejecución de las aplicaciones en las dos arquitecturas.

Escenario 1. Arquitectura monolítica

Se utilizó un servidor virtualizado en el Centro de Datos privado, donde pusimos todos los componentes de la aplicación en el mismo host, es decir, Frontend, Backend y base de datos.

Escenario 2. Arquitectura Microservicios

Con la arquitectura propuesta e infraestructura de Docker, cada servicio web se contenerizó, es decir, se ha tenido los siguientes servicios: Servicio Nginx, Servicio PostgreSQL, Servicio API REST.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

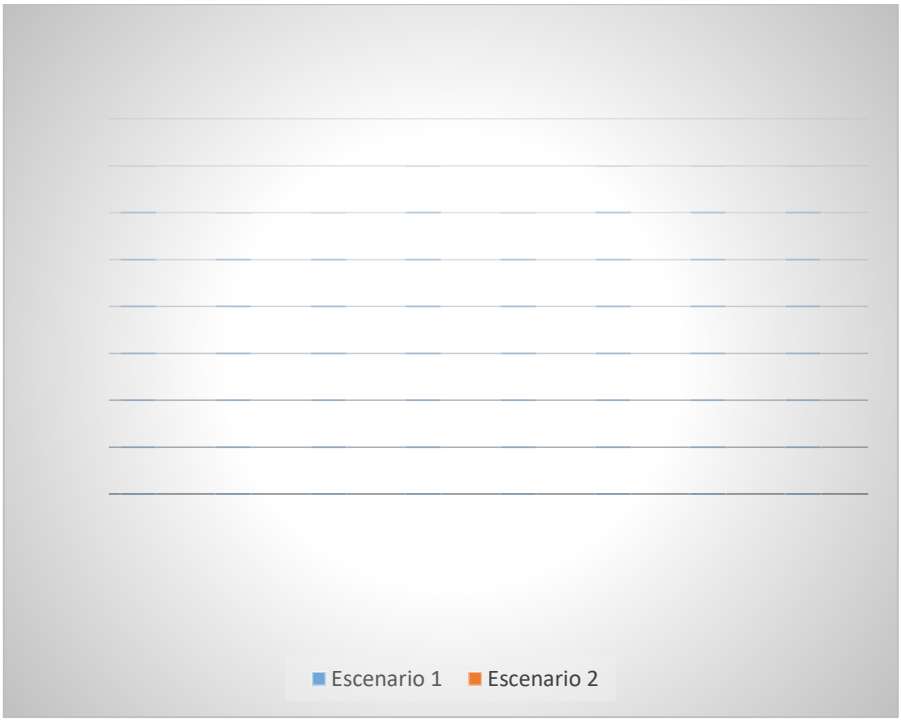

Figura 19. Tiempo de preparación de entornos de desarrollo de aplicaciones web

Donde:

Dev = Es cada uno de los desarrolladores que conforma un equipo.

Escenario 1 = Preparación de un entorno de desarrollo tradicional basado en arquitectura monolítica.

Escenario 2 = Preparación de un entorno de desarrollo basado en arquitectura microservicios e infraestructura Docker

El promedio empleado de la forma tradicional es de hrs. 7:37 por desarrollador frente a 0.14 de la misma forma por desarrollador.

Tiempo = de la forma tradicional.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

4.3 RESULTADOS CONFORME AL OBJETIVO ESPECÍFICO 3

Los resultados obtenidos en este objetivo específico, los organizamos en los siguientes componentes:

4.3.1 Pruebas Unitarias

Para la calidad del código, se diseñaron 15 pruebas unitarias (anexo 3), apreciamos en la figura 20 que las 15 pruebas se ejecutaron satisfactoriamente, notando que la ejecución de las pruebas unitarias se realizó dentro del contenedor.

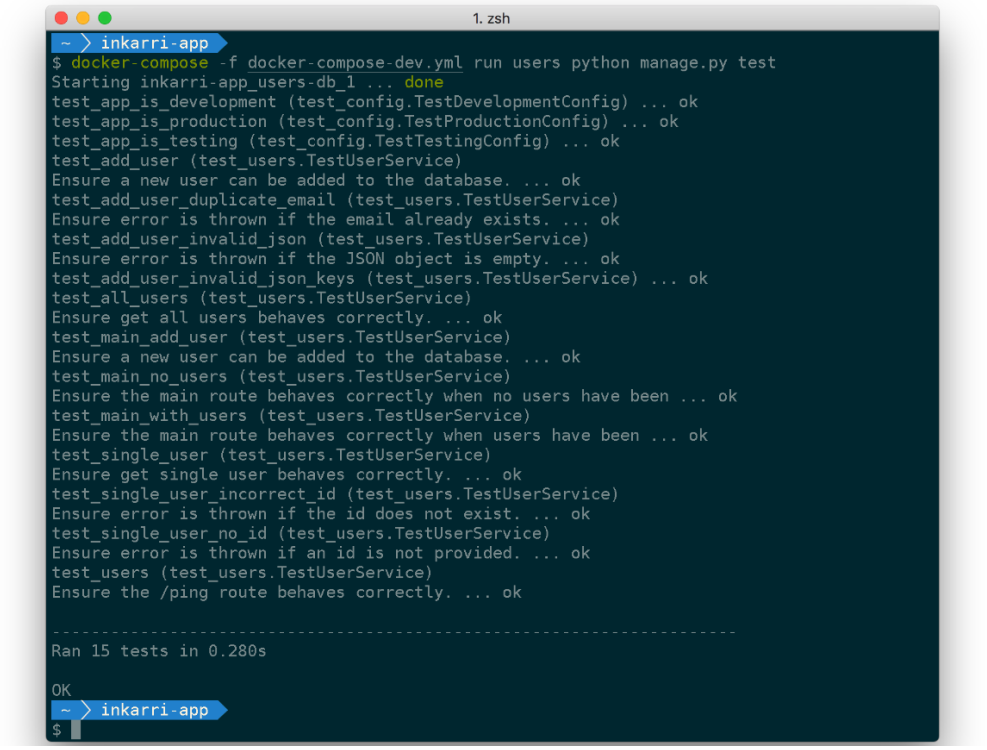

Figura 20. Ejecución de pruebas unitarias en un contenedor

Fuente: output del commando: $ docker-compose -f docker-compose-dev.yml run users python manage.py test

4.3.2 Cobertura de código

Ejecutadas la cobertura de codificación, se ha tenido como resultado total un 85% de codificación cubierta por las pruebas, tal como se muestra en el

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

cuadro 5, se han realizado la prueba a tres módulos: __init__.py con el 71% de cobertura, models.py con el 25% de cobertura y users.py 100%, es importante resaltar que el ultimo porcentaje aplicado al módulo users a alcanzado la totalidad de su cobertura, por que es la que contiene la lógica de programación para los usuarios del sistema, lo que es un resultado alcanzado.

Cuadro 5. Resumen de cobertura

Nombre del módulo	Stmts	Miss	Branch	BrPart	Cover
project/__init__.py	14	4	0	0	71%
project/api/models.py	12	9	0	0	25%
project/api/users.py	48	0	10	0	100%
Cobertura total	74	13	10	0	85%

Fuente: Output python manage.py coverage de una apliciación contenerizada con Docker.

Donde:

Stmts = número de casos a probar
Miss = número de casos no tratados
Brach = Cobertura de rama
BrPart = Cobertura parcial de rama
Cover = tasa de cobertura de código

Es importante destacar que esta cobertura se ha realizado en un entorno virtualizado, producto del empaquetamiento de la aplicación y sus dependencias en un contenedor Docker, la ejecución de esta prueba de cobertura se realizó ejecutando el siguiente comando:

```
$ docker-compose  run users python manage.py coverage
```


Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Lo cual demuestra que la arquitectura horizontal ha sido construida con éxito.

4.3.3 Calidad de código

Se realizó la verificación de errores de programación o de estilo de codificación de código fuente presentado mediante la utilización de la herramienta Flake8, por su compatibilidad de uso con los estilos propios de codificación de Python, como es la propuesta de mejora del lenguaje (Python Enhancement Propostal) en la guía de estilo para código Python (PEP8). Tras la ejecución del comando de verificación.

```
$ docker-compose  run users flake8 project
```

4.3.4 Pruebas de integración

Las pruebas de integración se han realizado considerando el repositorio del proyecto alojado en GitHub, tal como se muestra en el anexo 4, y su integración con Travis, para realizar una compilación y certificación externa se configuró con la herramienta Travis CI anexo 5, básicamente implica la construcción de los contenedores los resultados son mostrados con el resaltado del Job log de la herramienta mencionada, teniendo como parte del proceso la construcción de contenedores y ejecución de las pruebas unitarias, composición de contenedores y la cobertura de código, de cuyo resultado se tiene que todos los procesos ha sido ejecutados satisfactoriamente, como se muestra en el anexo 6, lo que implica que se encuentra listo para su ejecución, en este caso en instancias de EC2 de Amazon WS Anexo 7.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Cuadro 6. Tiempo empleado en las pruebas, construcción

Herramienta y commando ejecutado	Tiempo de ejecución
Docker `$ sudo service docker start`	0.02s
Github `$ git clone --depth=50 --branch=master https://github.com/abelthf/inkarri-app.git abelthf/inkarri-app`	0.61s
Curl `$ curl -L https://github.com/docker/compose/releases/download/${DOCKER_COMPOSE_VERSION}/docker-compose-'uname -s'-'uname -m' > docker-compose`	2.19s
Chmod `$ chmod +x docker-compose`	0.01s
Docker `$ docker-compose -f docker-compose-dev.yml up --build -d`	78.08s
Docker `$ docker-compose -f docker-compose-dev.yml run users python manage.py test`	2.97s
Docker `$ docker-compose -f docker-compose-dev.yml run users flake8 project`	1.66s
Otros procesos del sistema	0.23s
Total de ejecución expresado en unidad de tiempo	2 '5"

Fuente: Travis CI

4.3.5 Discusión

En el desarrollo de aplicaciones con una arquitectura de Microservicio y la infraestructura que lo soporte, no puede probar completamente todos los servicios hasta que se implemente en un servidor de producción o en etapas, este proceso requiere mucho tiempo para obtener retroalimentación. Sin embargo, al implementar la arquitectura en un entorno de contenedores (Docker) ayuda a acelerar este proceso al facilitar el enlace local de servicios pequeños e independientes, tal como se demuestra en el cuadro 6.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

CONCLUSIONES

- El desarrollo de aplicaciones web utilizando el modelo Arquitectónico para el desarrollo de Aplicaciones Web propuesto, mediante el estilo de Microservicios, resulta beneficioso ya que permite tener entregas continuas del producto, y que además proporciona reproductibilidad, flexibilidad y reusabilidad del módulo por otros componentes del sistema, tanto internos como externos; permitiendo la integración con metodologías ágiles de desarrollo de software.

- La utilización de la arquitectura propuesta para la Implementación y despliegue de aplicaciones web, mediante la tecnología de contenedores Docker, reduce ampliamente los tiempos empleados de la forma tradicional, generando también escalabilidad tanto en centro de datos privados y computación en la nube, notándose la eficiencia del despliegue.

- El desarrollo y despliegue de los Servicios Web RESTful en las aplicaciones desarrolladas en el caso de estudio presentaron reducción de tiempo en preparar ambientes de desarrollo y puestas de producción, ya que las tecnologías utilizadas son las más adecuadas para el

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

despliegue en centros de datos privados y servicios en Computación en la Nube, y servicios prestados como EC2 de Amazon Web Service, lo cual reduce la brecha en el uso do de estas nuevas tecnologías de información.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

RECOMENDACIONES

- Se recomienda el diseño adecuado de los centros de datos privados, ya que de ello depende la performance y desempeño de los contenedores, ya que un mal diseño puede generar cuellos de botella en la red de datos, con el objetivo de que el software distribuido con la arquitectura para servicios web propuesto sea desplegado como software como servicio, que soporte la portabilidad y despliegues en la nube, con el objetivo de minimizar las diferencias entre los entornos de desarrollo, producción y despliegue continuo.

- Se recomienda realizar la implementación en otros entornos de contenedores distintos a Docker, ya que esta se está convirtiendo en plataforma cada vez más aislada que podría generar un fuerte acoplamiento con esta tecnología, entre las mejores alternativas se tiene a CoreOS, como opción para una estandarización abierta de administración de contenedores.

- Se recomienda el uso de arquitecturas basadas en Web Service RESTful para el desarrollo de aplicaciones web modernas.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

BIBLIOGRAFÍA

Alba, J. (2008). *SOA: Arquitectura Orientada al Servicio.* Bit, ISSN 0210-3923, Nº. 167, 2008, pags. 52-53.

Alva, O. (2011). *Computación en la nube*. Obtenido de http://www2.izt.uam.mx/newpage/contactos/anterior/n80ne/nube.pdf

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). *Web Services*. Springer, 86-87

Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., & Steinder, M. (2015). *Performance Evaluation of Microservices Architectures using Containers*

Bernerss-Lee, T., Fielding, R., & Masinter, L. (2005). *Uniform Resource Identifier* (URI): Generic Syntax

Booth, D., Haas, H., McCabe, F., Newcomer, E., Ferris, C., & Orchard, D. (2004). *Web Services Architecture.* http://www.w3.org/TR/ws-arch.

Cabrera, O., Oriol, M., López, L., Franch, X., Marco, J., Fragoso, O., & Santaolaya, R. (2010). *WeSSQoS: Un Sistema SOA para la Selección de Servicios Web según su Calidad. Barcelona*, España

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Canto, M., Pereda, D., & Segurola, A. (2006). *Service Oriented Architecture* (SOA). Universidad de la República, Montevideo, Uruguay

Chinnici, R., Moreau, J., Ryman, A., & Weerawarana, S. (2007). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C Working Draft

Cito, J., Ferme, V., & Gall, G. (2016). *Using Docker Containers to Improve Reproducibility in Software and Web Engineering Research*. *Web Engineering: 16th International Conference*, 609-612

Clement, L., Hately, A., Riegen, C., & Rogers, T. (2004*). UDDI 3.0.2 Spec*. OASIS, uddi.org/pubs/uddi- v3.0.2-20041019.htm, accessed on 04/2017

Clements, P. (1996). *A Survey of Architecture Description Languages*. Proceedings of the International Workshop on Software Specification and Design, Alemania

Dean, J. & Ghemawat, S. (2008). *MapReduce: Simplified data processing on large clusters*. Communications of the ACM (51). 107-113

Dragoni, N., Giallorenzo, S., Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). *Microservices: yesterday, today, and tomorrow*. Present and Ulterior Software Engineering, Springer

Duarte, G. (2016). *Arquitectura Propuesta para un Servicio Web Completo: Metodología de Desarrollo e Implementación*. Universidad Distrital Francisco José de Caldas, Bogotá, Colombia

Fowler, M., & Lewis, J. (2014). *Microservice Architecture*. http://martinfowler.com

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Ghemawat, S., Gobioff, H. & Leung, S. (2003) *The Google File System. Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 29–43

Goncalves, A. (2010). *SOAP Web Services. In: Beginning Java™ EE 6 Platform with GlassFish™ 3*. Apress

Greenbeng, A., Hamilton, J., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltaz, D., Patel, P., & Sengpta, S. (2009). *Vl2: A Scalable and Flexible Data Center Network*. Communications ACM (54). 95-104

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H. F., Karmarkar, A., & Lafon, Y., (2007). *SOAP Version 1.2 Part 1: Messaging Framework* (Second Edition). http://www.w3.org/TR/soap12-part1/

Joyanes, L. (2012). *Computación en la nube. Estrategias de cloud computing para las empresas*. México DF: Alfaomega

Killalea, T. (2016). *The Hidden Dividends of Microservices*. *Communications of the ACM*, (59), 42-45

Kreger, H. (2001). *Web Services Conceptual Architecture*. IBM Software Group

Levcovitz, A., Terra, R., Valente, M. (2015). *Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems*. 3rd Brazilian Workshop on Software Visualization, Evolution and Maintenance, 97-104

MacKenzie, M., Laskey, K., McCabe, F., Brown, P., Metz R., & Hamilton, B. (2006). *Reference Model for Service Oriented Architecture 1.0.* Committe Specification 1, 2. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing*.

Papazoglou, M. (2008). *Web Services: Principles & Techonology*, Prentice-Hall

Pautasso, C., Zimmerman, O., & Leymann, F. (2008). *RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision*.

Perera, S., & Gunarathne, T. (2013). *Hadoop MapReduce Cookbook: Recipes for analyzing large and complex datasets with Hadoop MapReduce*. Packt Publishing

Popek, G., & Goldberg, R. (2013). *Formal requirements for virtualizable third generation architectures*, Communications of the ACM (17). 412–421.

Red Hat Inc. (2016). *Consiga una Arquitectura de Microservicios Exitosa*. España

Safina, L., Mazzara, M., & Montesi, F. (2015). *Data-driven Workflows for Microservices*. Innopolis University, Russia

San José, J. (2016). *Propuesta de arquitectura basada en servicios web y agentes para el desarrollo de aplicaciones de seguimiento y trazabilidad de productos*. Universidad de Castilla - La Mancha, España

Todea, V. (2016). *Diseño e implementación de un sistema de entrega continua para aplicaciones web sobre contenedores Docker*. Universidad Politécnica de Valencia, Valencia, España

Urra, I. (2016). *Distributed Microservice Architecture with Docker*. Universitat Oberta de Catalunya, España

Zhang, Q, Cheng, L., & Boutaba, R. (2010). *Cloud computing: state-of-the-art and research challenges*. The Brazilian Computer Society

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Zharo, H., & Doshi, P. (2009) *Towards Automated RESTful Web Service Composition*. Conference: IEEE International Conference on Web Service, ICWS, Los Ageles, USA

Ferreira, R. (2013). *XP Extreme Programming*. http://slideplayer.es/slide/84721/

Gonzales, A., Álvarez, D., & Acosta, L. (2012). *jPET 2.0: Un generador automático de casos de prueba sobre programas Java*. Universidad Complutense de Madrid, España

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

ANEXOS

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Anexo 1. Versiones de Docker

Se realizaron la configuración local con las herramientas Docker client, Server y Machine, con las versiones que se muestran a continuación.

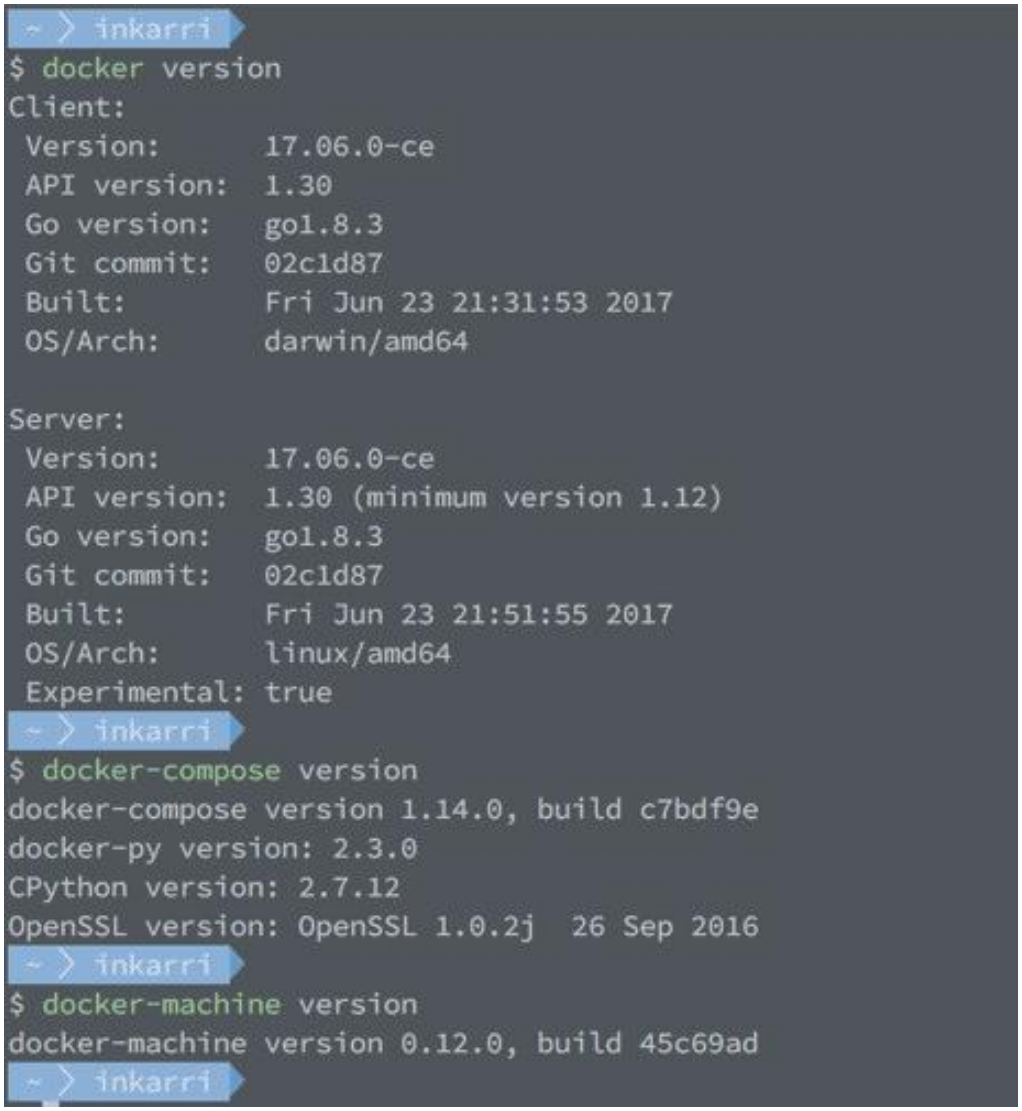

Figura 21. Versiones de Docker: client, server, compose y machine

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Anexo 2. Composición de contenedores

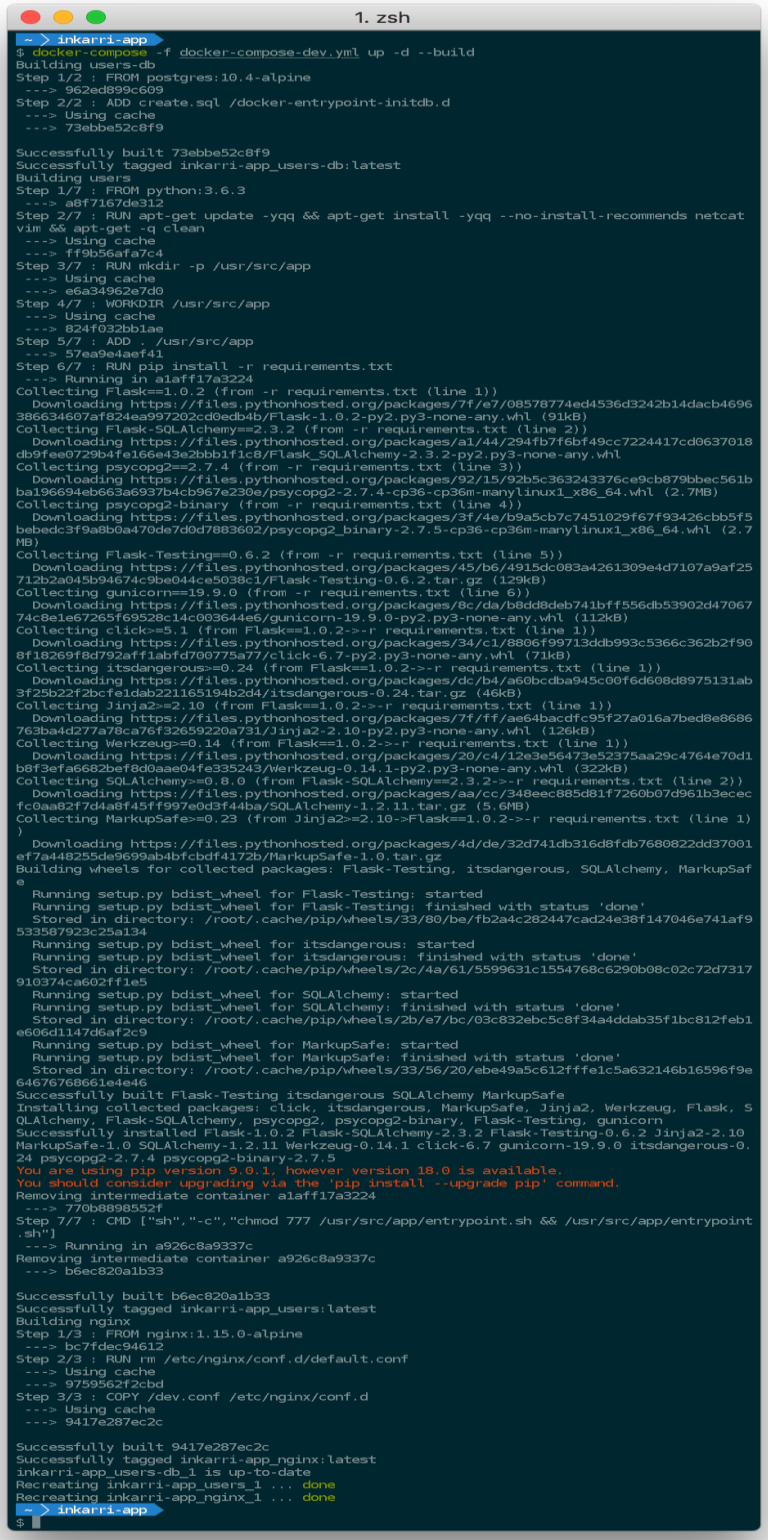

Figura 22. Composición contenedor de desarrollo

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

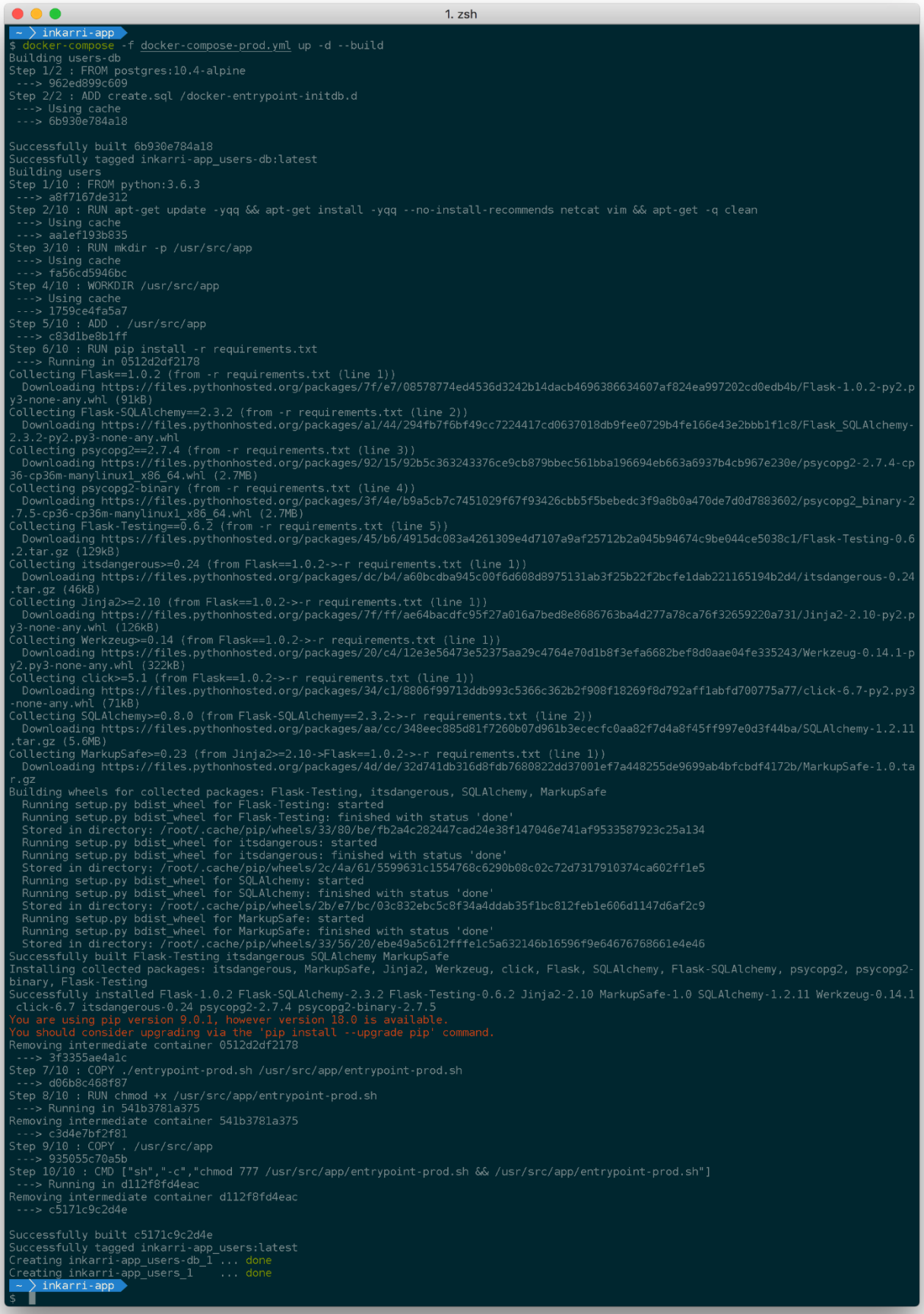

Figura 23. Composición contenedor producción

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Anexo 3. Pruenas unitarias

Test de configuración: Testing, Development, Producción.

```
class PruebaTestingConfig(TestCase):
    def create_app(self):
        app.config.from_object('project.config.TestingConfig')
        return app

    def test_app_is_testing(self):
        self.assertTrue(app.config['SECRET_KEY'] == 'my_secret_key')
        self.assertTrue(app.config['TESTING'])
                    self.assertTrue(
            app.config['SQLALCHEMY_DATABASE_URI'] ==
            os.environ.get('DATABASE_TEST_URL')
        )
        self.assertFalse(app.config['DEBUG_TB_ENABLED'])
```

Prueba de configuración para ambientes de desarrollo

```
class PruebaDevelopmentConfig(TestCase):
    def create_app(self):
        app.config.from_object('project.config.DevelopmentConfig')
        return app

    def test_app_is_development(self):
        self.assertTrue(app.config['SECRET_KEY'] == 'my_secret_key')
        self.assertFalse(current_app is None)
        self.assertTrue(
            app.config['SQLALCHEMY_DATABASE_URI'] ==
            os.environ.get('DATABASE_URL')
        )
        self.assertTrue(app.config['DEBUG_TB_ENABLED'])
```

Prueba de configuración para ambientes de producción

```
class PruebaProductionConfig(TestCase):
    def create_app(self):
        app.config.from_object('project.config.ProductionConfig')
        return app

    def test_app_is_production(self):
        self.assertTrue(app.config['SECRET_KEY'] == 'my_secret_key')
        self.assertFalse(app.config['TESTING'])
        self.assertFalse(app.config['DEBUG_TB_ENABLED'])
```


Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Anexo 4. Repositorio del proyecto

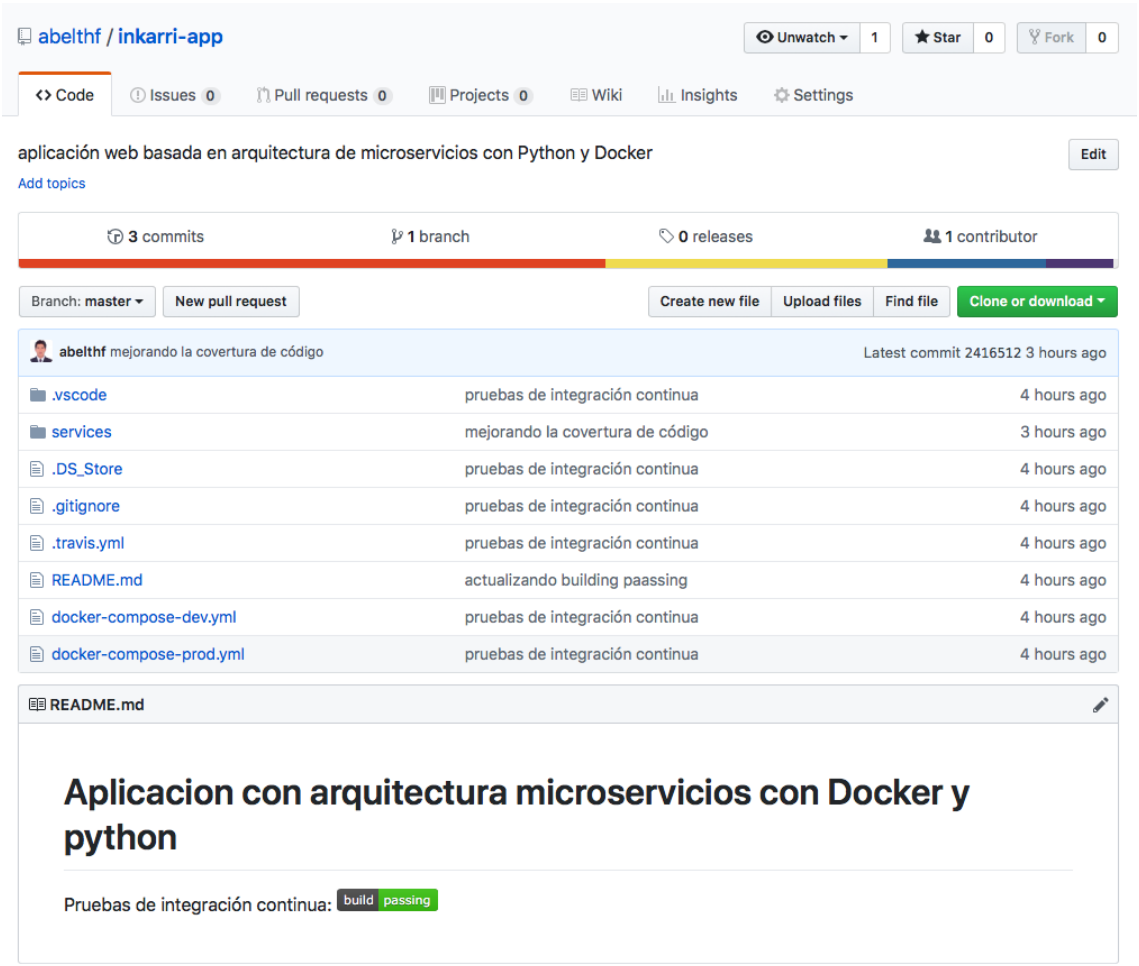

Figura 24. *Repositorio del proyecto en https://github.com/*

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Anexo 5. Integración continua

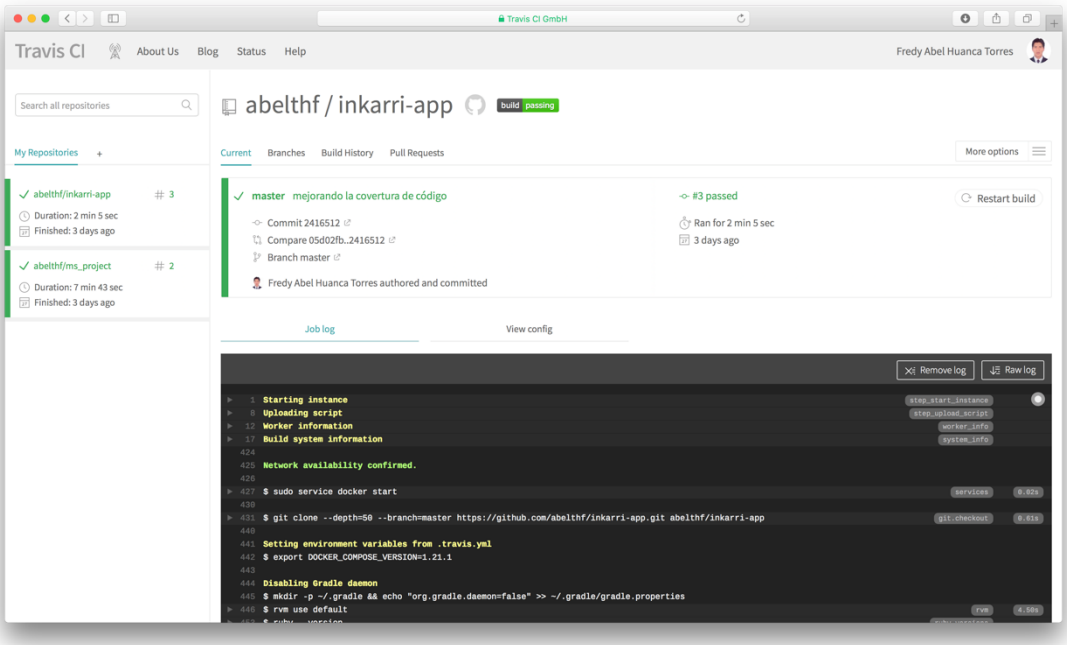

Figura 25. Validación del proyecto e integración contínua

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Anexo 6. Reporte de pruebas de integración

Figura 26. Resultado de pruebas de integración

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

Anexo 7. Proyecto desplegado en amazon web service EC2

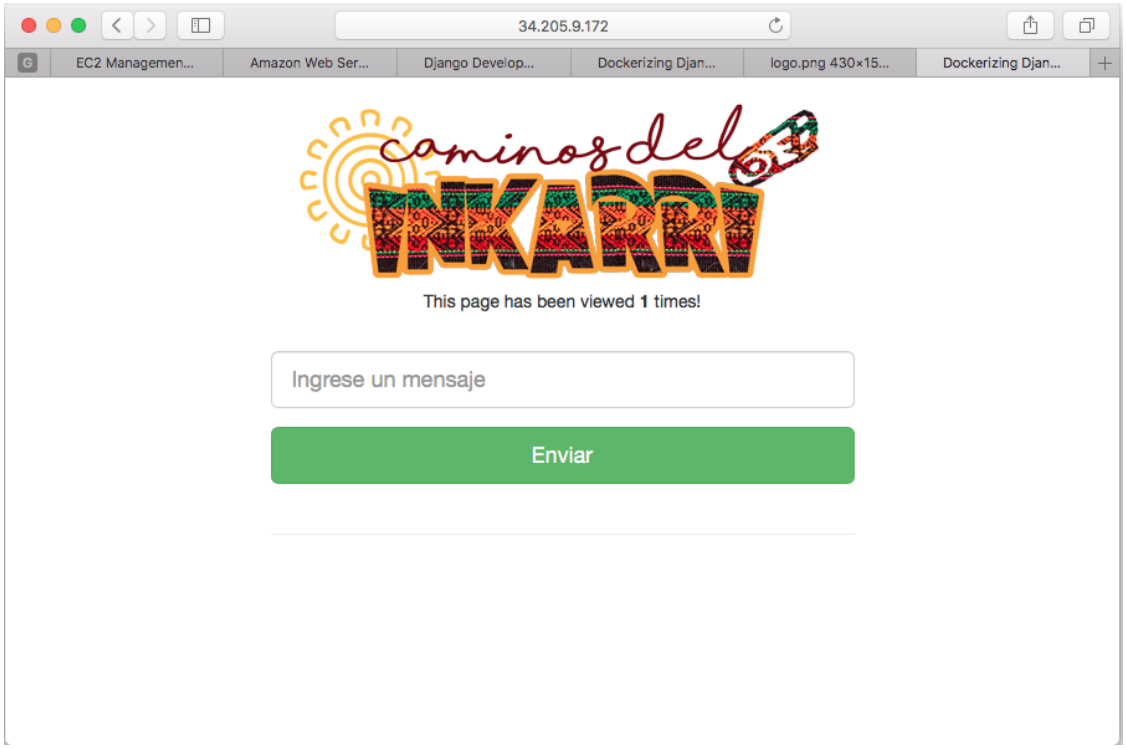

Figura 27. Resultado del teplate del proyecto

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Se utilizó el servidor Nginx con la configuración en el cuadro siguiente: Se creó el archivo project dentro de la carpeta site-enable:

```
server {

    listen 80;
    server_name example.org;
    charset utf-8;
    location /static {
        alias /usr/src/app/static;
    }
    location / {
        proxy_pass http://web:5000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
    }
}
```

Servicio OAuth

Se utilizó la aplicación djangorestframework para construir las API y django-oauth-toolkit como protocolo estándar de autorización, dentro de un proyecto django, con la configuración del archivo settings.py siguiente:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'django.contrib.admindocs',
    'rest_framework',
```


Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

```
    'corsheaders',
    'oauth2_provider',

    'oauth2_backend',
    'backend_utils',
    'django-oauth-toolkit',
]
```

Figura 28. Estructura de las API's del recurso OAuth

Composición de base de datos

La base de datos utilizada fue PostgreSQL, el mismo que ha sido configurado en un contenedor.

```
postgres:

              restart: always

              image: postgres:latest

              ports:

                - "5432:5432"

              volumes:

                - pgdata:/var/lib/postgresql/data/
```

Composición del servicio cache Redis

```
redis:

           restart: always

           image: redis:latest

           ports:

             - "6379:6379"

           volumes:

             - redisdata:/data
```

Configurado en el contenedor basado en la imagen oficial Redis, escuchando el puerto 6379, y utiliza el volumen Docker para los datos en /data.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis

Una vez configurado todos los contenedores, docker-machine despliega los contenedores locales para los desarrolladores, tal como se muestra:

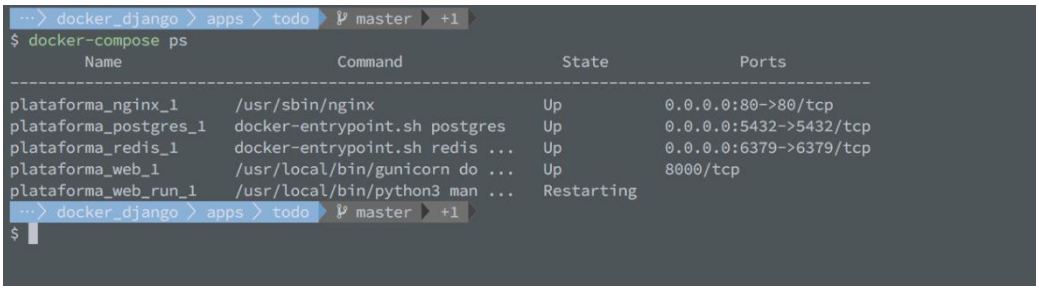

Figura 29. Composición de contendores para cada servicio

Fuente: output de los procesos de docker-compose

DESPLIEGUE

En esta fase del despliegue se hizo las pruebas con el servicio Elastic Compute Cloud (EC2) de Amazon Web Service (AWS) con el objetivo de aprovisionar contenedores dentro de cada instancia EC2 aprovisionada automáticamente por Docker, mediante la creación de usuario administrador en EC2

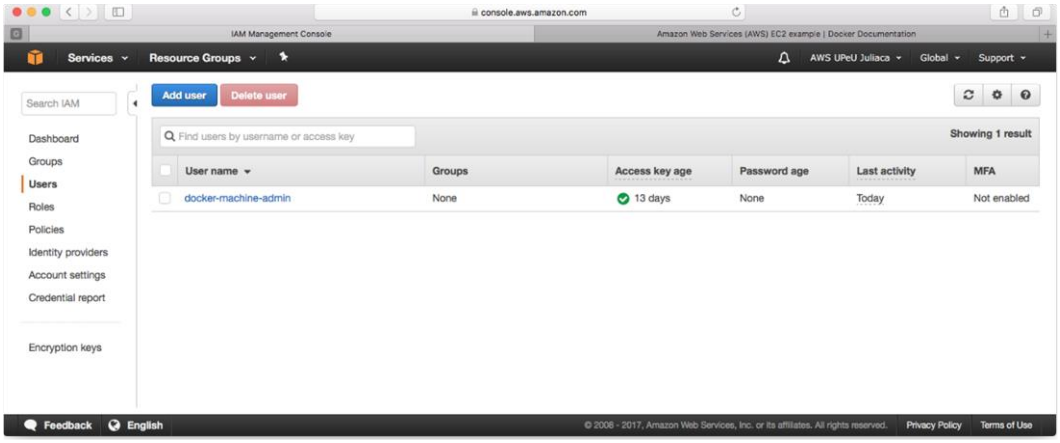

Figura 30. Creación de usuario administrador en EC2

Aprovisionamos una instancia EC2 de Linux en AWS, tal como se muestra en la figura siguiente.

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

```
$ docker-machine create --driver amazonec2 --engine-install-url=https://web.archive.org/web/20170623081500/https://get.docker.com production
Running pre-create checks...
Creating machine...
(production) Launching instance...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with ubuntu(systemd)...
Installing Docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env production
```

Figura 31. Aprovisionamiento de instancia EC2

El aprovisionamiento anterior creó una instancia EC2, tal cual podemos ver en la figura inferior, en el que podemos apreciar que se desplegó una instancia micro de Linux, al cual podemos acceder por el puerto 80 del IP asignado.

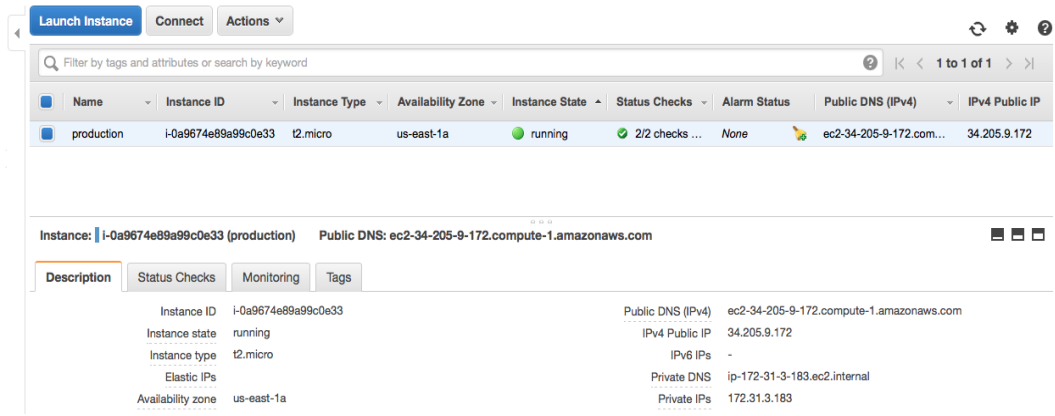

Figura 32. Consola de administración del panel de instancias EC2

Docker-machine ahora tiene desplegado dos máquinas, una local y otra en la nube, como se puede apreciar en la figura inferior, una tiene el driver VirtualBox y el otro Amazon EC2 respectivamente.

```
$ docker-machine ls
NAME         ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER        ERRORS
dev          *        virtualbox   Running   tcp://192.168.99.100:2376           v17.06.0-ce
production   -        amazonec2    Running   tcp://34.205.9.172:2376             v17.05.0-ce
```

Figura 33. Lista de máquinas virtuales desplegadas

Se Estableció las variables de entorno de la instancia aprovisionada por Docker en AWS, para su gestión, mediante el siguiente comando:

Repositorio Institucional UNA-PUNO

No olvide citar esta tesis

$ eval "$(docker-machine env production)"

La ejecución del comando anterior permitirá la gestión directa del cliente Docker con el servidor de la instancia AWS que contiene Docker como servidor. Construimos los requerimientos con el comando build, tal como se muestra en la figura siguiente.

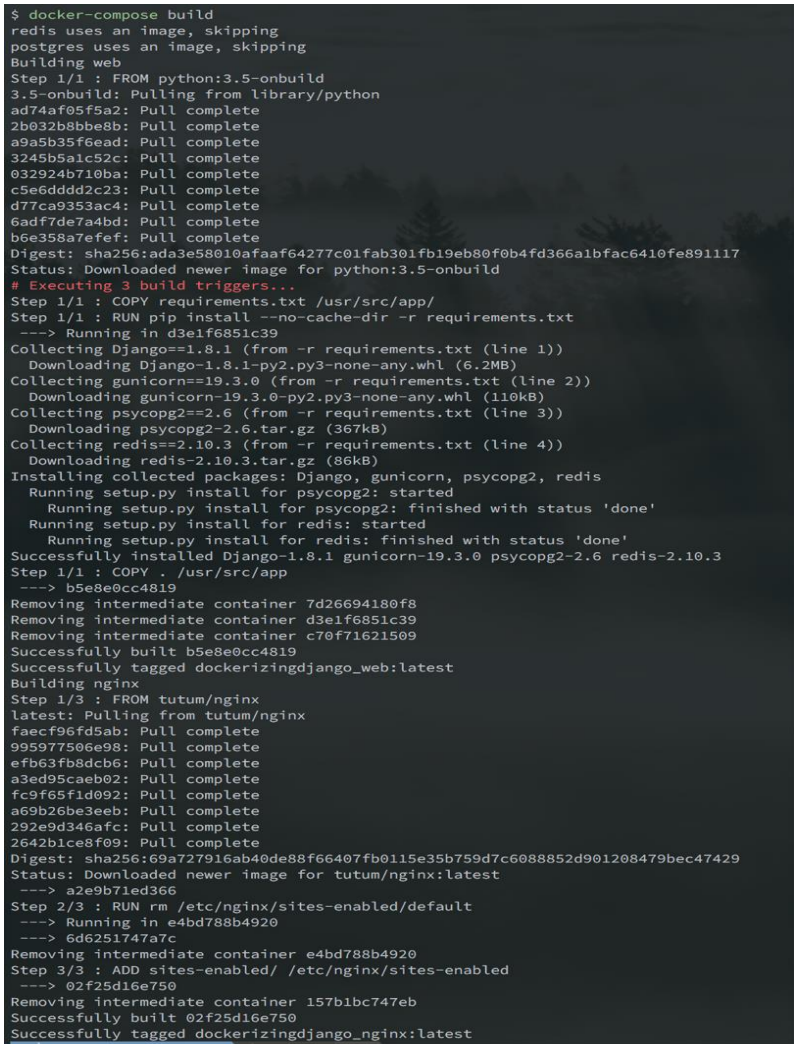

Figura 34. Construcción de componentes

Construcción de componentes

Hasta el momento se consiguió tener la instancia AWS en la nube con las imágenes necesarias de Docker, sin embargo, aún falta instalar los servicios, como Redis, PostgreSQL, Gunicon, Nginx.

Repositorio Institucional UNA-PUNO
No olvide citar esta tesis