

Definición de una infraestructura cloud de alta disponibilidad en un entorno distribuido para el
despliegue de una plataforma IoT

Jose David Rojas Aguilar

Trabajo de Grado para optar al título de Ingeniero de Sistemas

Director

Gabriel Rodrigo Pedraza Ferreira

Doctor en Ciencias de la computación

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de sistemas e informática

Bucaramanga

2019

Dedicatoria

Este trabajo viene dedicado para todas aquellas personas que apoyaron el desarrollo y ejecución de este trabajo de grado.

En especial reconozco la permanente presencia de Dios en mi camino de vida.

Agradecimientos

Agradezco a mi familia por el apoyo económico y moral que tuvieron para conmigo durante el desarrollo de mi carrera. También agradezco a mis amigos y compañeros por las vivencias de estos inolvidables años de universidad.

Un reconocimiento y agradecimiento importante lo realizo a mi director de trabajo de grado, por dedicar su tiempo, experiencia y conocimiento en la guía de mi proyecto.

Tabla de Contenido

Introducción	16
1. Objetivos	17
2. Estado del arte	17
3. Marco de referencia	23
3.1. Computación en la nube	23
3.1.1. Características esenciales	23
3.1.2. Modelos de servicio	24
3.1.3. Modelos de despliegue	25
3.2. Alta disponibilidad	26
3.2.1. Escalabilidad	28
3.3. Virtualización	30
3.4. Contenedores	32
3.5. Orquestación de contenedores	35
3.6. Microservicios	35
4. Marco técnico	36
4.1. Docker	36

4.2. Kubernetes	37
4.3. Objetos Kubernetes	38
4.4. Openshift	39
4.5. Ansible	40
4.6. GlusterFS	41
5. Desarrollo del proyecto	41
5.1. Ambientación tecnológica	42
5.1.1. Investigación fundamentos teóricos relacionados con el proyecto (Tecnologías y es- tándares)	42
5.1.2. Estudio	43
5.1.3. Sondeo	43
5.2. Definición de arquitectura de infraestructura cloud	46
5.2.1. Definición requisitos	46
5.2.2. Definición de métricas	49
5.2.3. Definición de pruebas	50
5.2.4. Definición arquitectura e infraestructura	52
5.3. Prototipado	57
5.3.1. Configuración básica cluster	57
5.3.2. Configuración infraestructura cluster	58
5.3.2.1. Servidor DNS	58

5.3.2.2. Servidor Glusterfs	59
5.3.2.3. Nodos	61
5.3.3. Despliegue aplicación	63
5.3.4. Automatización del proceso de despliegue	65
5.3.5. Implementación de pruebas	66
5.4. Validación de prototipo	67
5.4.1. Aplicación pruebas	67
5.4.2. Análisis resultados	69
5.5. Caso de uso	71
6. Conclusiones	73
7. Trabajo futuro	74
Referencias Bibliográficas	74
Apéndices	81

Lista de Figuras

Figura 1. Arquitectura sugerida por AWS para una aplicación en WordPress. Adaptado de (Aws, 2018)	20
Figura 2. Arquitecturas sugeridas por Azure para una estación de Blockchain. Adaptado de (Altimore, 2018)	21
Figura 3. Arquitecturas sugeridas por GCP para una aplicación basada en eventos. Adaptado de (Goo, 2018)	22
Figura 4. Responsabilidades según el modelo de servicio. Adaptado de (Gantenbein and Neira, 2017)	25
Figura 5. Modelo de un cluster activo-activo. Adaptado de (Villanueva, 2015)	27
Figura 6. Modelo de un cluster activo-pasivo. Adaptado de (Villanueva, 2015)	28
Figura 7. Escalabilidad vertical en contraste con escalabilidad horizontal. Adaptado de (Mauro, 2012)	29
Figura 8. Arquitectura tradicional en contraste con arquitectura con máquinas virtuales. Adaptado de (CONSULTING, 2012)	31
Figura 9. Máquinas virtuales vs contenedores. Adaptado de (Joy, 2015)	33
Figura 10. Solicitudes procesadas en 600 segundos. Adaptado de (Joy, 2015)	34
Figura 11. Arquitectura monolítica vs arquitectura de microservicios. Adaptado de (Kapagantula, 2019)	36

Figura 12. Arquitectura de un cluster Kubernetes. Adaptado de (Kub, 2016a)	38
Figura 13. Arquitectura de Openshift. Adaptado de (Ope, 2017)	40
Figura 14. Arquitectura de Glusterfs. Adaptado de (glu, 2019)	41
Figura 15. Arquitectura propuesta.	56
Figura 16. Infraestructura hardware del prototipo.	58
Figura 17. Plataforma web de administración de Openshift.	62
Figura 18. Proceso de despliegue de un artefacto dentro de la arquitectura propuesta en BPMN.	64
Figura 19. Mensaje enviado por Gitlab cuando ocurre un error durante el despliegue automático.	66
Figura 20. Plan de pruebas implementado en Jmeter.	66
Figura 21. Caso de uso.	72

Lista de Tablas

Tabla 1.	Tecnologías disponibles en el mercado	42
Tabla 2.	Sondeo de selección: Sistema Operativo	44
Tabla 3.	Sondeo de selección: Container Runtime	45
Tabla 4.	Sondeo de selección: Orquestador de contenedores	46
Tabla 5.	Métodos usados en las pruebas sobre el prototipo	51
Tabla 6.	Características de máquinas en cluster	57
Tabla 7.	Resolución de nombres en cluster	59
Tabla 8.	Artefactos de la aplicación IoT según su persistencia	60
Tabla 9.	Configuración de bricks de volúmenes	60
Tabla 10.	Cantidad de solicitudes fallidas durante un despliegue en un entorno de 1000 solicitudes por minuto	67
Tabla 11.	Tiempo promedio de respuesta para 1000 solicitudes por minuto	68
Tabla 12.	Número máximo de solicitudes por minuto sin solicitudes fallidas.	68
Tabla 13.	Porcentaje de solicitudes fallidas para 3500 solicitudes por minuto	69
Tabla 14.	Cantidad de solicitudes fallidas para 1000 solicitudes por minuto durante el fallo de un nodo	69

Lista de Apéndices

	pág.
Apéndice A. Dataset generado para pruebas	81
Apéndice B. Archivo Inventory	82
Apéndice C. Endpoints glusterfs	83
Apéndice D. Archivos de despliegue	84
Apéndice E. Manual de instalacion	85
Apéndice F. Archivos de configuración de despliegue continuo	86

Glosario

Backend Capa software que se encarga de gestionar los datos de una o varias aplicaciones.

Continuous deployment Práctica que consiste en realizar el proceso de despliegue de forma automática al ambiente de producción.

Dataset Conjunto de datos generados o extraídos de una fuente para su posterior procesamiento.

Development & Operations Es una práctica de ingeniería de software que tiene como objetivo automatizar y monitorear las etapas del ciclo de vida del software..

Infraestructura Conjunto de componentes informáticos que permiten mantener un conjunto de aplicaciones al servicio de usuarios.

Internet de las cosas Es un sistema de dispositivos de computación interrelacionados con la capacidad de transferir datos a través de una red, sin requerir intervención humana.

Sistema de alimentación ininterrumpida Dispositivo que almacena energía para suministrar a equipos en caso de alguna falla eléctrica

Resumen

Título: Definición de una infraestructura cloud de alta disponibilidad en un entorno distribuido para el despliegue de una plataforma IoT *

Autor: Jose David Rojas Aguilar **

Palabras Clave: Internet de las cosas, Escalabilidad, Smart Campus, Infraestructura, Cloud, Alta Disponibilidad.

Descripción: El internet de las cosas (IoT) ha tenido un gran impacto en los últimos años dentro de la sociedad en diferentes contextos. Una de sus aplicaciones más importantes es la mejora de la calidad de vida y el apoyo en el proceso de aprendizaje de los estudiantes dentro de las diferentes universidades. Un reto dentro del IoT es tener una infraestructura capaz de soportar una cantidad masiva de dispositivos enviando información de forma constante.

Este trabajo de investigación se busca diseñar una arquitectura software para el despliegue de una plataforma IoT en una infraestructura cloud de alta disponibilidad en un entorno distribuido. El proyecto inició con una fase de exploración con el fin de identificar las herramientas disponibles en el mercado. Luego se definieron unos criterios de selección a nivel técnico, unos requisitos de los artefactos a desplegar, un conjunto de métricas y un conjunto de pruebas para evaluar el desempeño de la arquitectura plateada. Después se definió una arquitectura la cual fue evaluada, en una primera oportunidad, por las pruebas diseñadas durante el proyecto y, finalmente, en una prueba de integración en un caso de uso de la plataforma IoT.

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de sistemas e informática. Director: Gabriel Rodrigo Pedraza Ferreira, Doctor en Ciencias de la Computación.

Abstract

Title: Definition of a cloud infrastructure of high availability in a distributed environment for the deployment of an iot platform *

Author: Jose David Rojas Aguilar **

Keywords: Internet of Things, Scalability, Smart Campus, Infrastructure, Cloud, High availability.

Description: The Internet of Things (IoT) has had a strong impact in recent years within society in different contexts. One of its most important applications is the improvement of the quality of life and the support in the learning process of the students in different universities. A challenge in the IoT is to have an infrastructure capable of supporting a massive amount of devices sending information constantly.

This research work looks for design software architecture for the deployment of an IoT platform in a high availability cloud infrastructure in a distributed environment. The project began with an exploration phase in order to identify the tools available in the market. Then technical selection criteria were defined, requirements of the artifacts to be deployed, a set of metrics and a set of tests to evaluate the performance of the proposed architecture. An architecture was defined, which was evaluated, at a first opportunity, by the tests designed during the project and, finally, in an integration test in a case of use of the IoT platform.

* Bachelor Thesis

** Faculty of Physical-Mechanical Engineering. School of Systems Engineering and Informatics. Advisor: Gabriel Rodrigo Pedraza Ferreira, PhD in Computer science

Introducción

La expansión de internet y los grandes avances la comunicación entre diferentes dispositivos ha permitido el surgimiento de un nuevo paradigma tecnológico: Internet de las cosas (IoT). IoT se define como la infraestructura mundial para la sociedad de la información, que permite poner diferentes servicios a dispositivos interconectados entre sí, gracias a las tecnologías de la información y comunicación, permitiendo la extracción y procesamiento masivo de datos.

El internet de las cosas ha tenido un impacto significativo en contextos de diferentes magnitudes. Grandes empresas como Amazon, Microsoft, Google, Cisco, Philips e Intel han invertido millones en investigación, generando diferentes servicios enfocados al hogar, empresas e incluso ciudades. Barcelona es una ciudad pionera en el uso de IoT en sus calles, por ejemplo, la línea 9 del metro de Barcelona se ha actualizado con ascensores inteligentes que utilizan los datos en tiempo real para adaptarse a las necesidades de los viajeros lo cual ha logrado disminuir las aglomeraciones y el consumo de energía para 30 millones de pasajeros al año(BAR, 2018).

Uno de los campos de aplicación del IoT se encuentra en las universidades con el fin de mejorar la calidad de vida y apoyar el proceso de formación en los estudiantes. La implementación de una solución IoT suele ser una tarea muy compleja que requiere de una infraestructura hardware y software capaz de soportar una cantidad masiva de dispositivos enviando información de forma continua por lo cual una característica fundamental dentro de una infraestructura IoT es la alta

disponibilidad.

En este trabajo de investigación se presenta el diseño de una infraestructura para el despliegue de una plataforma IoT en una infraestructura cloud de alta disponibilidad en un entorno distribuido con el fin de proveer un entorno que pueda soportar una gran cantidad de dispositivos conectados enviando datos de forma masiva.

1. Objetivos

Objetivo general

Definir e implementar mecanismos para el despliegue de una infraestructura cloud que provea alta disponibilidad y escalabilidad para el caso de una plataforma IoT en un entorno distribuido.

Objetivos específicos

Identificar las herramientas en el mercado para el despliegue de una infraestructura cloud.

Diseño de la arquitectura y pruebas para validar la arquitectura planteada.

Despliegue de la arquitectura planteada en un entorno de pruebas y aplicación de las pruebas planteadas.

2. Estado del arte

Es importante tener en cuenta los aportes que han hecho otros colegas en trabajos relacionados con el presente trabajo de grado, y por lo tanto, se hará referencia a algunos de estos proyectos.

Administración de prototipo infraestructura de computación en la nube del GID-CONUSS¹ con énfasis en la implementación de nuevos módulos de Openstack. Este trabajo tenía como objetivo administrar, monitorear y mantener el funcionamiento de la infraestructura de computación en la nube modelo de alta disponibilidad del GID-CONUSS, con énfasis en la exploración de nuevas tecnologías y otros módulos de Openstack en el GID-CONUSS (Martínez, 2017).

Estudio de una alternativa de integración continua que soporte el desarrollo de una infraestructura TI² de servicios de información del transporte público de pasajeros. Este trabajo tenía como objetivo diseñar y evaluar una alternativa de integración continua para el desarrollo de los componentes de una infraestructura TI que ofrece servicios de información al transporte público de pasajeros (Valbuena, 2017).

Implementación de una nueva infraestructura de computación en la nube con modelo de alta disponibilidad basada en Openstack y contenedores para el GID-CONUSS. Este trabajo tenía como objetivo investigar y complementar el trabajo hecho previamente (Martínez, 2017) en GID-CONUSS en la creación de una infraestructura de nube (Balaguera, 2018).

¹ Grupo de I+D Computación en la nube, Servidores, Seguridad y Servicios.

² Tecnología de la información

Despliegue y monitorización de un clúster Mesos. Este trabajo tenía como finalidad utilizar un sistema de gestión de recursos en un conjunto de nodos computacionales (Mesos) para el despliegue y evaluación del rendimiento de un cluster aplicaciones con diferentes complejidades algorítmica (López, 2016).

Despliegue de una aplicación usando Docker y Kubernetes. Este trabajo tenía como finalidad de optimizar el proceso de despliegue de una aplicación desarrollada por Sparta Consulting Ltd en un servidor con Minikube (Moilanen, 2018).

Diseño de una arquitectura cloud para una aplicación con varios usuarios. Este trabajo tenía como finalidad diseñar una arquitectura cloud que pudiera soportar una cantidad masiva de usuarios de una aplicación de pagos móvil (Schuchmann, 2018).

A su vez, existen arquitecturas cloud empresariales que pueden ser adaptadas según los requerimientos del proyecto a implementar.

AWS provee un conjunto de servicios y cada arquitecto arma su arquitectura según los requerimientos del proyecto y los servicios que proporciona AWS como podemos ver en la figura 1. Este modelo es también utilizado por Microsoft Azure (Figura 2) y GCP(Figura 3).

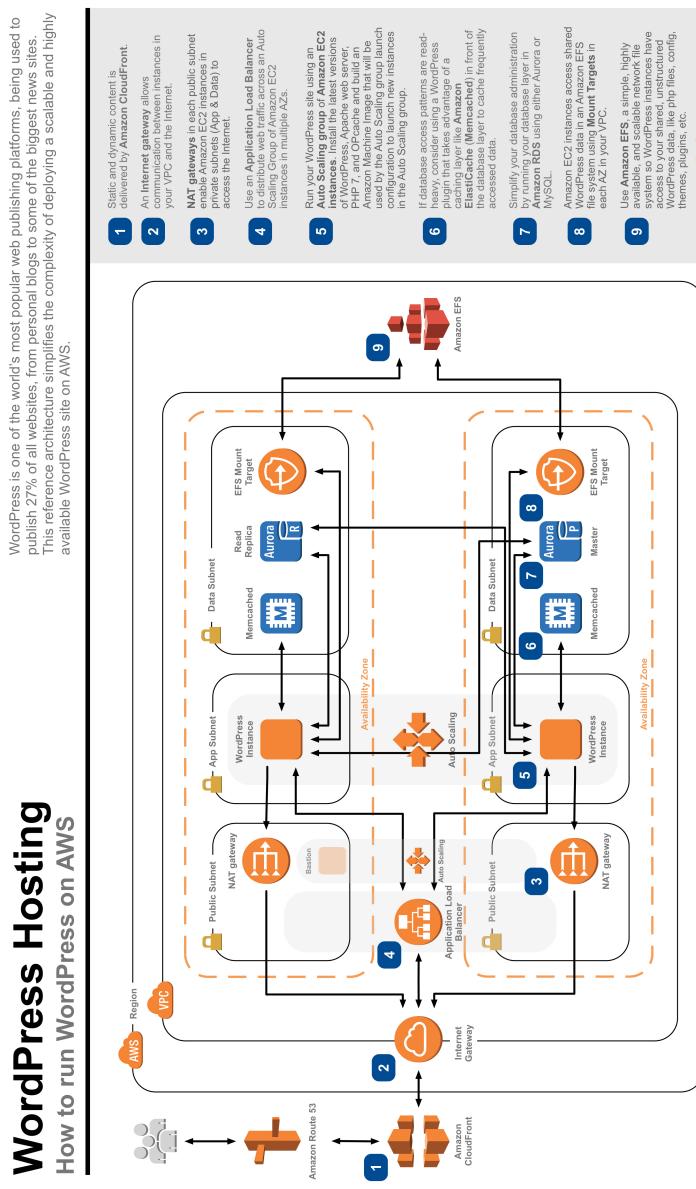


Figura 1. Arquitectura sugerida por AWS para una aplicación en WordPress. Adaptado de (AWS, 2018)

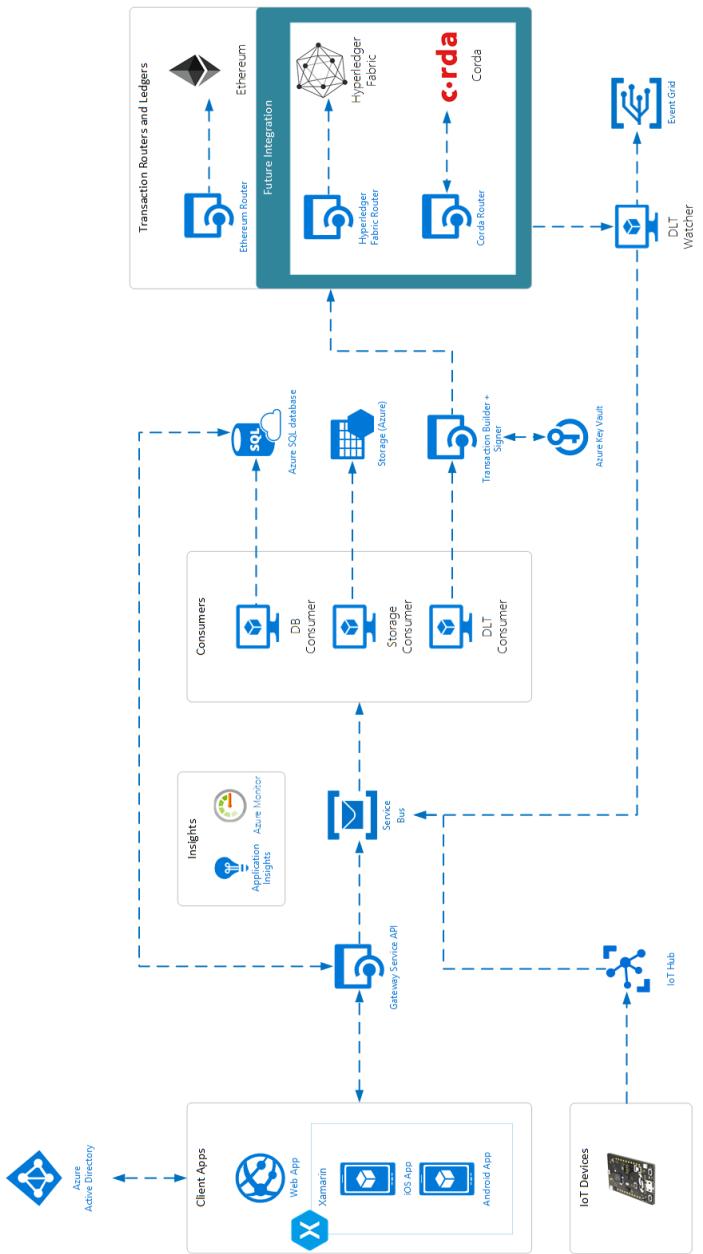


Figura 2. Arquitecturas sugeridas por Azure para una estación de Blockchain. Adaptado de (Altimore, 2018)

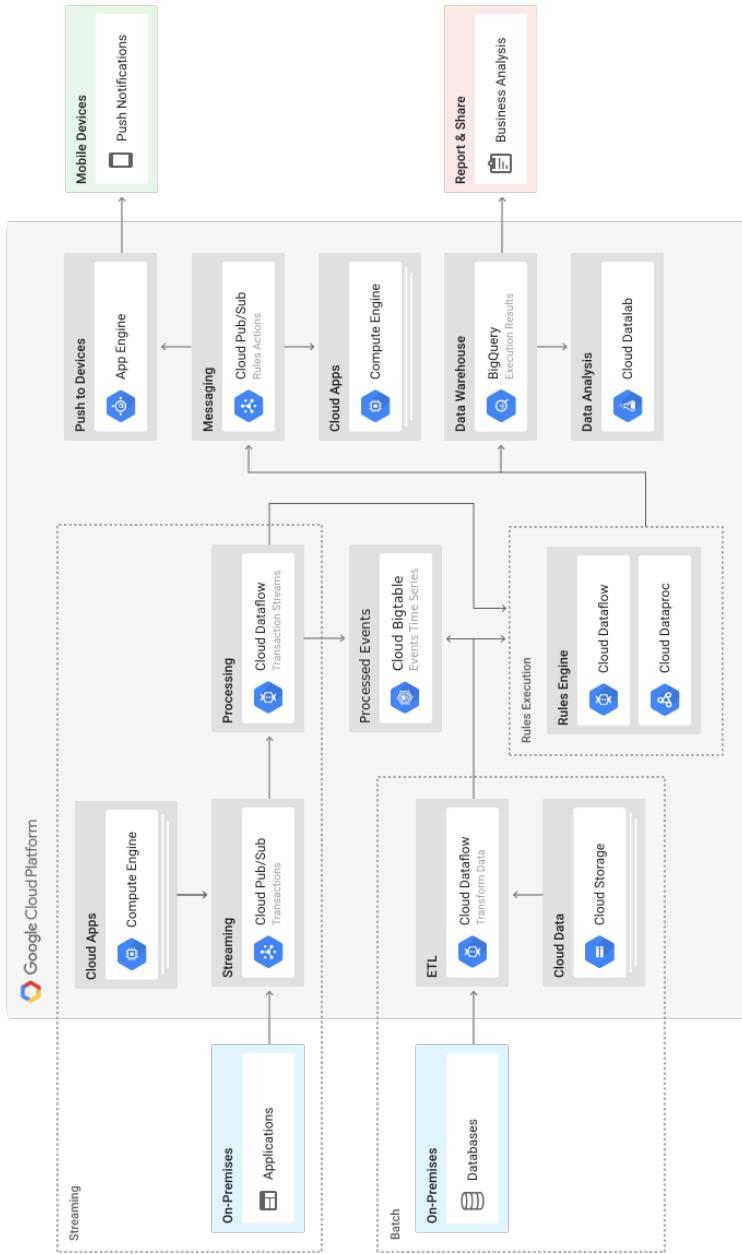


Figura 3. Arquitecturas sugeridas por GCP para una aplicación basada en eventos. Adaptado de (Goo, 2018)

3. Marco de referencia

En el presente Capítulo se presentan los conceptos básicos que serán abordados en Capítulos posteriores.

3.1. Computación en la nube

Es un modelo para permitir el acceso, de manera extensa, conveniente y bajo demanda, a un grupo compartido de recursos informáticos configurables (Por ejemplo redes, servidores, almacenamiento, aplicaciones y servicios) que pueden provisionarse y lanzarse rápidamente con un mínimo esfuerzo administrativo o la interacción del proveedor de servicios (Mell and Grance, 2011).

3.1.1. Características esenciales.

- **Autoservicio sobre demanda:** Los usuarios tienen acceso a recursos en la nube, por ejemplo capacidad de cómputo o almacenamiento, bajo demanda siempre que sean necesarios.
- **Amplio acceso a la red:** Los recursos están disponibles a través de la red y se accede por medio del mecanismo estándar que promueve el uso de la plataforma por un grupo variado de dispositivos cliente (Teléfonos móviles, tablets, laptops y estaciones de trabajo).
- **Agrupación de recursos:** Es una abstracción sobre la manera en la cual se separa la manera en la cual se encuentran los recursos físicamente distribuidos y la asignación de los mismo para los diferentes clientes. Los clientes suelen tener especificar la ubicación de los recursos a un nivel alto de abstracción (por ejemplo, país, estado o datacenter).

- **Elasticidad:** La capacidad de aumentar o disminuir los recursos asignados para poder escalar de la manera más óptima una aplicación. Esto puede ser manual o automático.
- **Servicio medido:** Los sistemas cloud poseen diferentes herramientas para poder medir el uso que se le da a los recursos asignados. Estas interfaces pueden ser gráficas o por línea de comando.

3.1.2. Modelos de servicio.

- **Software como servicio:** El cliente tiene acceso a una o varias aplicaciones que se encuentran ejecutando en la infraestructura cloud. El cliente se encuentra limitado al alcance que le provea la aplicación.
- **Plataforma como servicio:** El cliente puede desplegar aplicaciones en la infraestructura cloud siempre que se usen tecnologías, como lenguajes de programación, librerías, servicios y herramientas, soportadas por el proveedor del servicio.
- **infraestructura como servicio:** El cliente puede desplegar y correr software de forma arbitraria, esto incluye sistema operativo y aplicaciones, por lo cual no se ve limitado a ninguna configuración preliminar a nivel software.

En la figura 4 se puede ver las responsabilidades del cliente y proveedor según el modelo de servicio.

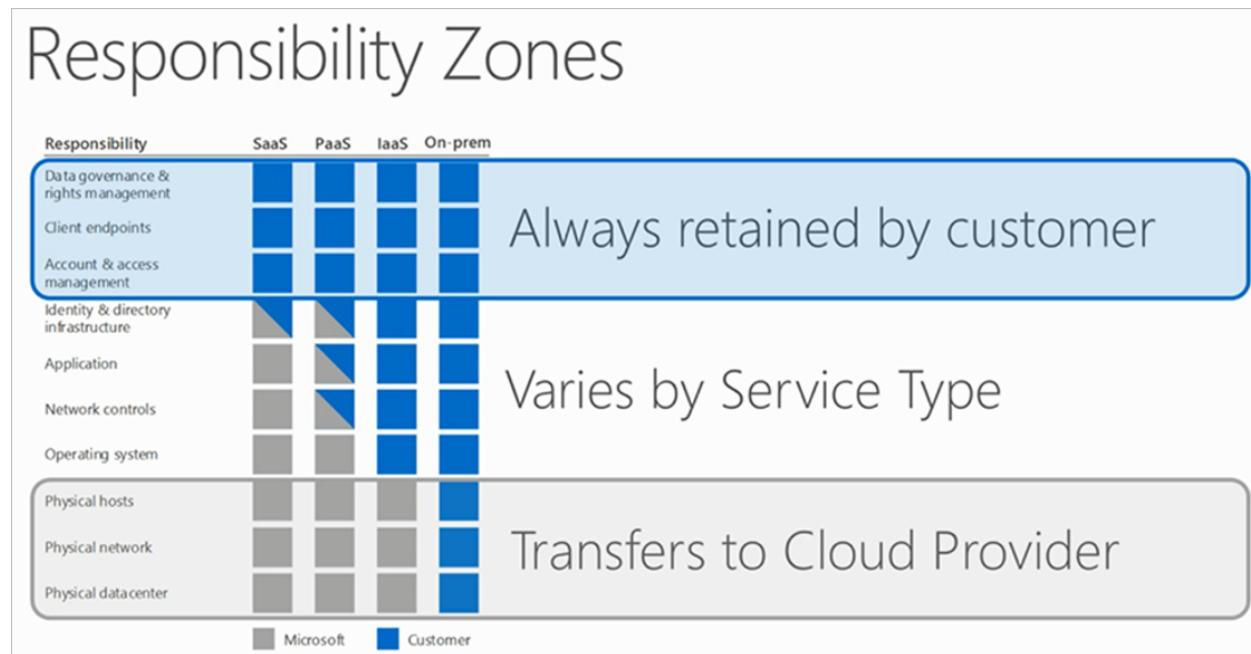


Figura 4. Responsabilidades según el modelo de servicio. Adaptado de (Gantenbein and Neira, 2017)

3.1.3. Modelos de despliegue.

- **Nube privada:** La infraestructura cloud es proveída para el uso exclusivo de una organización. Puede ser propiedad, administrada y operada por la organización, un tercero o una combinación de estos dos.
- **Nube comunitaria:** La infraestructura cloud es proveída por una organización para el uso exclusivo de una comunidad de consumidores que tienen unas necesidades comunes.
- **Nube pública:** La infraestructura cloud es proveída para el uso abierto de un público general. Está es administrada, operada y poseída por una empresa, académicos, una organización del gobierno o una combinación de los anteriores.

- **Nube híbrida:** La infraestructura cloud es una mezcla de dos o más tipos de infraestructura(Privada, comunitaria o pública).

3.2. Alta disponibilidad

En computación, disponibilidad es la capacidad de un módulo para ejecutar una función cuando se es requerido. La disponibilidad se expresa de la siguiente manera:

$$\text{Disponibilidad} = \frac{\text{Tiempodeservicio} - \text{Tiempodeinactividad}}{\text{Tiempodeservicio}} * 100 \quad (1)$$

Cuando hablamos de ?Alta disponibilidad?, hacemos referencia a que cumple el máximo estándar: la disponibilidad medida es de 99.999 % (Benz and Bohnert, 2013).

El objetivo de la alta disponibilidad es eliminar los puntos de fallo potencial en la infraestructura. Un punto de fallo potencial es un componente del stack tecnológico que puede producir una interrupción del sistema. Al igual, todo componente que sea necesario para el sistema y no tenga redundancia, también se considera un punto de falla (Heidi, 2016).

Existen 2 tipos de cluster de alta disponibilidad (Villanueva, 2015):

- **Activo-Activo:** Se suelen tener al menos 2 nodos, ambos se encuentran corriendo la misma clase de servicios de manera simultánea. El propósito principal de un cluster con alta disponibilidad de tipo activo-activo es lograr un buen balanceo de carga. El balanceo de cargas distribuye las solicitudes sobre todos los nodos con el fin de evitar que un solo nodo se sobrecargue. La configuración más sencilla es presentada en la figura 5

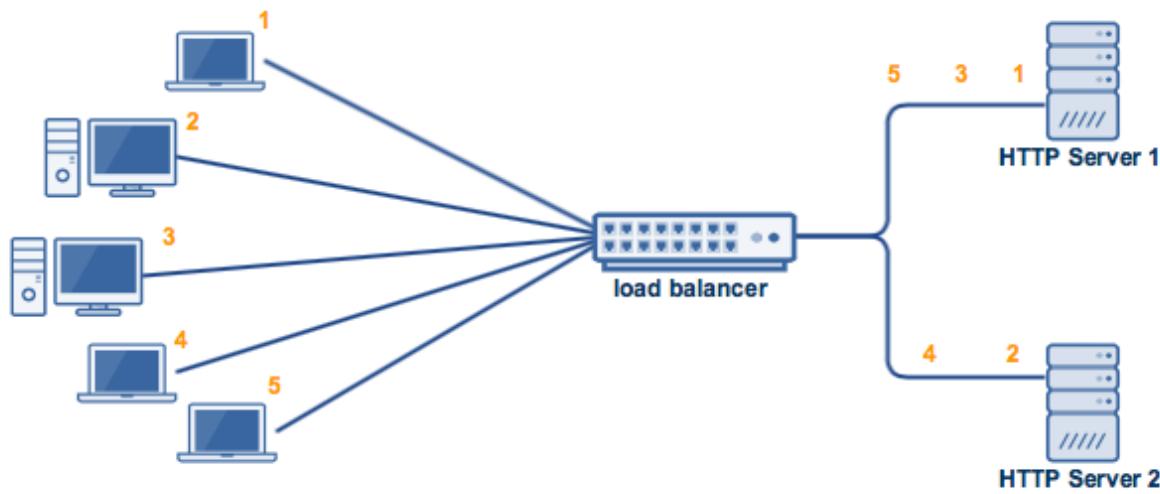


Figura 5. Modelo de un cluster activo-activo. Adaptado de (Villanueva, 2015)

Este modelo tiene un balanceador de carga y 2 servidores http. Los clientes no se conectan directamente a los servidores http, sino que las solicitudes pasan por medio de un balanceador de carga, el cual usa un algoritmo para determinar a qué servidor se direccionan las solicitudes.

Este modelo se recomienda para aplicaciones que van a tener una cantidad masiva de conexiones durante un tiempo prolongado.

- **Activo-Pasivo:** Al igual que la configuración activo-activo, la configuración activo-pasivo requiere al menos de 2 nodos. Se conoce como activo-pasivo por que no todos los nodos empiezan activos.

El nodo pasivo es un respaldo cuya función es recibir las solicitudes de los clientes tan pronto como el nodo activo se desconecte o quede inhabilitado para poder responder a las

solicitudes de los usuarios. En la figura 6 se presenta este modelo.

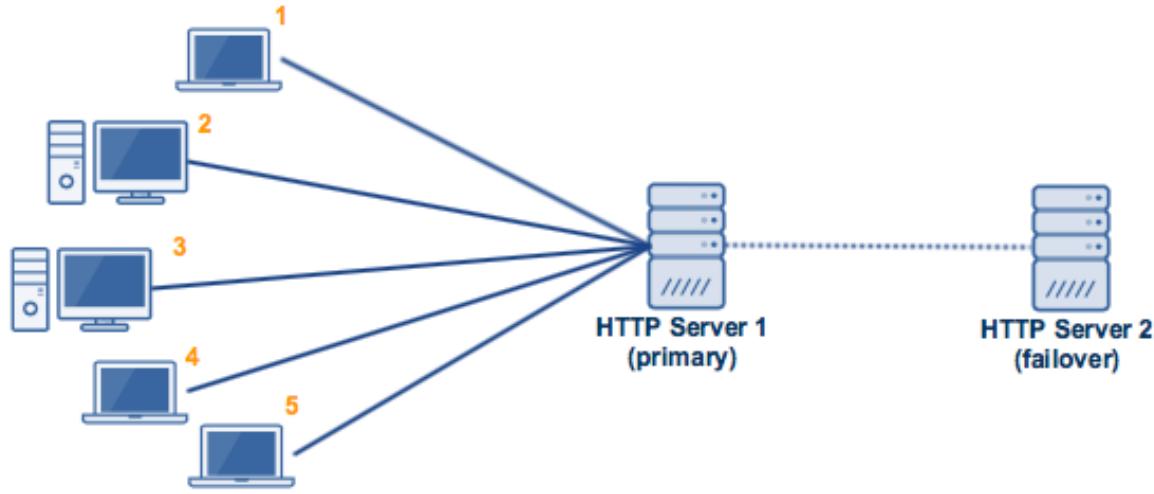


Figura 6. Modelo de un cluster activo-pasivo. Adaptado de (Villanueva, 2015)

Esta configuración se recomienda cuando no se va a tener que lidiar con una cantidad masiva de solicitudes durante un tiempo prolongado pero existen escenarios en los cuales la cantidad de peticiones puede aumentar de manera considerable.

3.2.1. Escalabilidad. Escalabilidad es la capacidad que tiene una solución para poder adaptarse al crecimiento en la demanda (Bansode, 2013).

Imaginemos que tenemos una aplicación corriendo en un servidor con ciertas características, de repente la cantidad de usuarios de nuestra aplicación aumenta por lo cual se hace necesario escalar nuestra infraestructura con el fin de poder seguir brindando el mejor servicio. Existen 2 maneras de escalar:

- **Escalabilidad vertical:** Era la forma de escalar convencional hace unos años y es básica-

mente aumentar las capacidades del equipo de computo o en su defecto, comprar uno nuevo con más capacidad. Es la forma más sencilla de escalar ya que ya que la arquitectura de la infraestructura no cambia.

- **Escalabilidad horizontal:** Consiste en distribuir la carga en un conjunto de equipos de cómputo, de tal manera que si necesitamos soportar una mayor carga, en lugar de cambiar todo el hardware por uno de mayor capacidad, lo que usualmente es muy costoso, simplemente añadimos un nuevo equipo al conjunto lo cual implica una redistribución de la carga y aumenta la cantidad que el conjunto puede llegar a soportar.

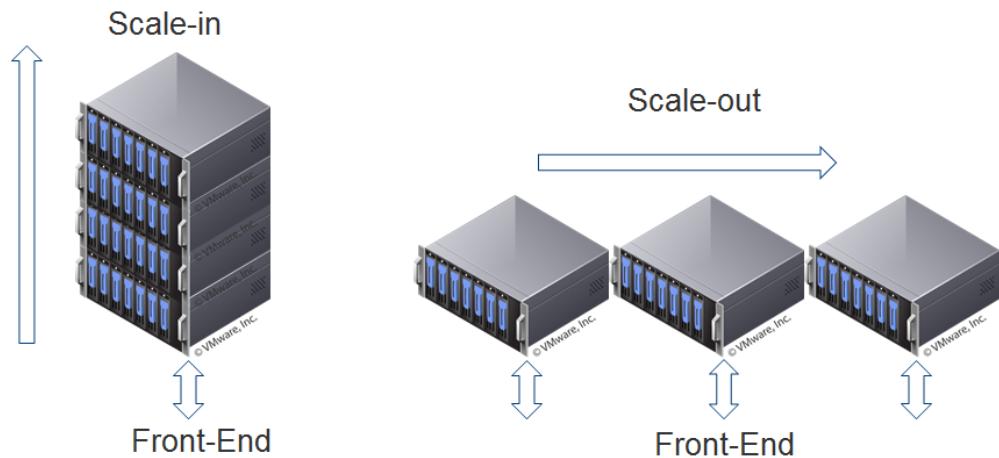


Figura 7. Escalabilidad vertical en contraste con escalabilidad horizontal. Adaptado de (Mauro, 2012)

La escalabilidad vertical tiene un problema: La ley de Moore, la cual dicta: Cada dos años, aunque en principio dijo que sería cada 18 meses, se duplica el número de transistores (Thompson

and Parthasarathy, 2006). Según la SIA³, aunque es físicamente posible que los fabricantes de microprocesadores lleguen a crear algunos chips más de lo estipulado por Moore, no sería práctico a nivel financiero, debido a los altos costos que implica (Yinug, 2015).

"Y, siendo optimistas, la fecha límite, de acuerdo con el presidente y CEO de la SIA John Neuffer sería, como mucho, 2030 (BBC, 2018). En otras palabras, va a llegar un punto en el cual, aunque tengamos dinero ilimitado para poder comprar el equipo de computo más poderoso en el mercado, va llegar un punto donde el equipo que pueda satisfacer nuestra demanda no exista, lo cual convierte la escalabilidad vertical en una solución inviable para aplicaciones que visionan un crecimiento masivo.

Por otra parte, la escalabilidad horizontal, a pesar de ser más compleja compleja en términos de arquitectura, puede escalar sin estar limitada por la ley de Moore además que, al tener la carga distribuida en varios nodos, podemos proveer un servicio de alta disponibilidad y los costos para escalar

3.3. Virtualización

La virtualización es una capa intermedia a nivel de sistema operativo que provee una abstracción de los recursos del sistema. Es tal el papel de la virtualización dentro del cloud computing que grandes compañías, como Amazon, Google y Microsoft, basan sus servicios cloud en la virtualización (Joy, 2015).

Las máquinas virtuales son impulsadas por los hipervisores. El hipervisor es un software

³ Asociación de la Industria de Semiconductores

que provee un entorno aislado para cada máquina virtual y es responsable de correr diferentes kernels a nivel del sistema operativo anfitrión.

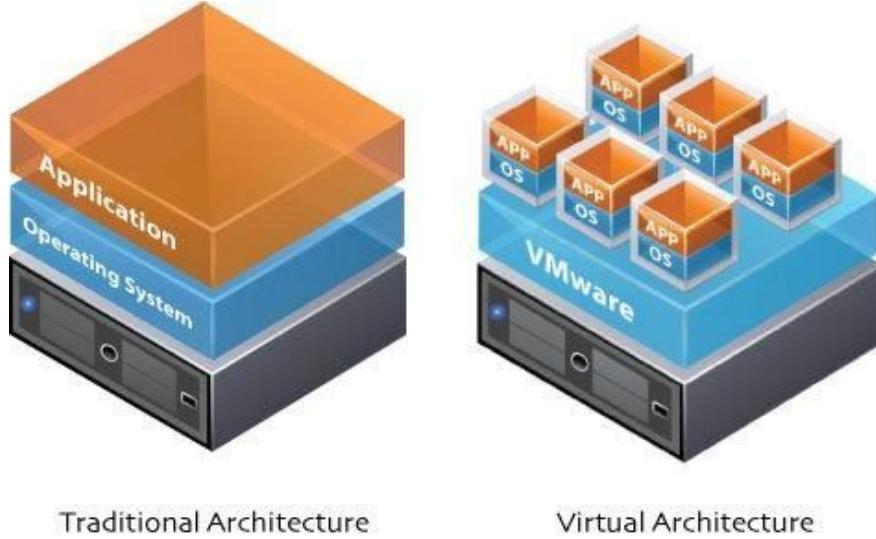


Figura 8. Arquitectura tradicional en contraste con arquitectura con máquinas virtuales. Adaptado de (CONSULTING, 2012)

Como podemos ver en la figura 8, en la arquitectura tradicional hay un acceso más directo al hardware pero posee las siguientes falencias:

- Solo se pueden ejecutar aplicaciones de un sistema operativo.
- Las aplicaciones no se corren en un ambiente aislado por lo que un fallo en el sistema afecta directamente a todas las aplicaciones.
- No hay portabilidad en las aplicaciones.

Las máquinas virtuales resuelven los problemas anteriormente mencionados usando el hypervisor, el cual nos permite crear máquinas virtuales con diferentes sistemas operativos corriendo

al tiempo por lo que podemos tener corriendo, en una misma máquina, aplicaciones para Linux y para Windows al tiempo, adicionalmente cada aplicación puede estar corriendo en ambientes aislados por lo que un fallo en una aplicación no va a afectar a las demás aplicaciones. Sin embargo, las máquinas virtuales tienen los siguientes problemas:

- Los recursos necesarios es significativamente mayor al enfoque tradicional debido a que se deben correr los servicios de cada sistema operativo adicional por cada máquina virtual.
- El rendimiento de las aplicaciones dentro de las máquinas virtuales se ve afectado.
- Los tiempos de encendido y apagado de las máquinas virtuales pueden llegar a ser del orden de minutos.

Para solucionar estos problemas aparecen los contenedores.

3.4. Contenedores

Las aplicaciones software suelen desplegarse como un conjunto de librerías y archivos de configuración en un entorno, por ejemplo, un servidor. Estas se despliegan en un sistema operativo con un conjunto de servicios corriendo, como puede ser un servidor de base de datos o un servidor http, sin embargo estos servicios pueden ser desplegados en cualquier ambiente que pueda proveer los mismos servicios, ya sea una máquina virtual o una máquina física.

Sin embargo, esta metodología tiene un problema relacionado con la actualización o parches ya que estos pueden, por problemas de compatibilidad, dejar una aplicación fuera de servicio. Otro escenario es el cual tenemos 2 aplicaciones en un mismo sistema operativo anfitrión las cuales comparten librerías, luego para solucionar un problema con la aplicación 1, surge la necesidad

de actualizar una de las librerías, en cuyo caso se corre el riesgo de afectar el funcionamiento de la aplicación 2. Para poder evitar cualquier inconveniente durante el despliegue, las compañías de software suelen hacer pruebas antes de realizar el despliegue en el sistema de producción, sin embargo, según la complejidad de la aplicación, estas pruebas pueden llegar a ser una tarea tediosa.

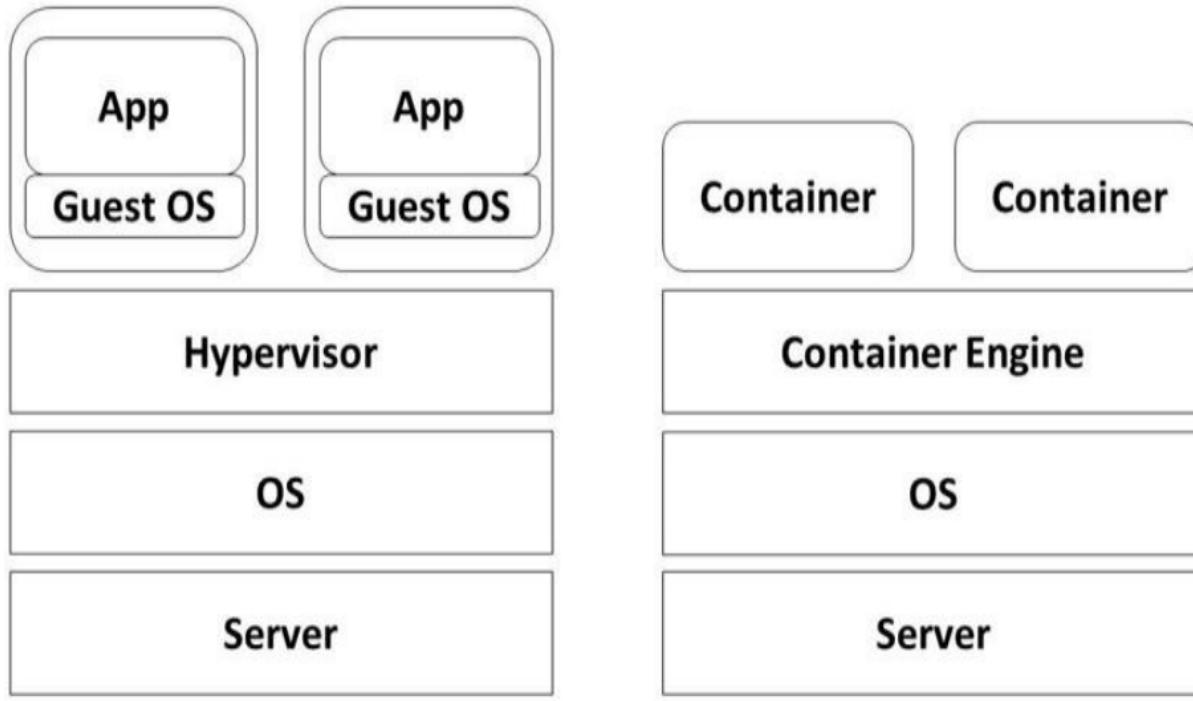


Figura 9. Máquinas virtuales vs contenedores. Adaptado de (Joy, 2015)

Como alternativa, aparecen los contenedores los cuales son un ambiente aislado dentro de un sistema operativo. Los contenedores toman ciertos beneficios de las máquinas virtuales, como la seguridad, el almacenamiento y el aislamiento de red, mientras que consumen muchos menos recursos que las máquinas virtuales (Joy, 2015). Adicionalmente, los contenedores nos proveen un rendimiento y escalabilidad mayor a las máquinas virtuales como se muestra en la figura 10

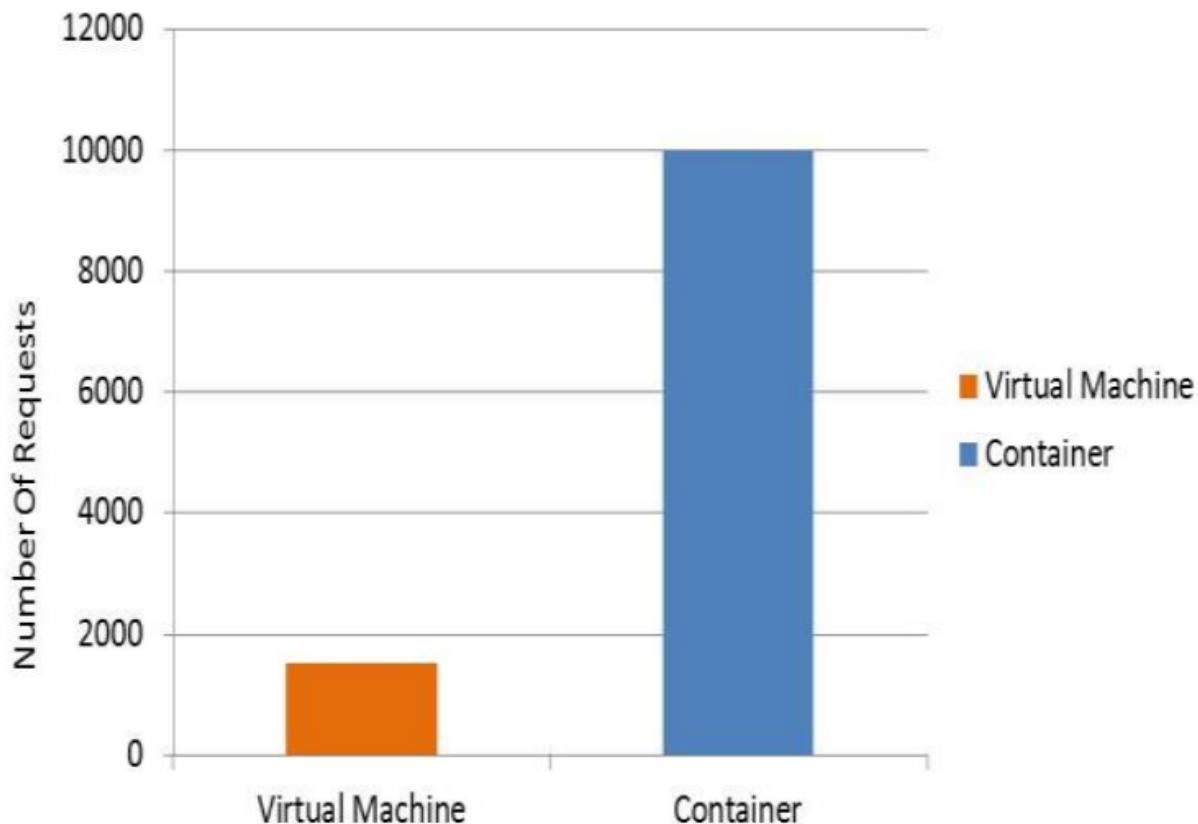


Figura 10. Solicitudes procesadas en 600 segundos. Adaptado de (Joy, 2015)

Los contenedores poseen las siguientes ventajas:

- Poco impacto sobre los recursos
- Ambiente aislado
- Despliegue rápido
- Portabilidad

3.5. Orquestación de contenedores

Supongamos que tenemos una aplicación desplegada usando contenedores y por alguna razón, ya sea un ataque por denegación de servicios o un simple error en el código de la aplicación, el contenedor que la contiene falla. En un sistema de disponibilidad alta debemos asegurar de alguna manera que la aplicación siempre va a estar disponible, por lo tanto debemos implementar una arquitectura que nos permita tolerar fallos, es aquí donde aparece el siguiente concepto: Orquestación de contenedores.

La orquestación de contenedores nos permite gestionar los ciclos de vida de los contenedores, especialmente en ambientes de gran tamaño y de naturaleza dinámica (Eldridge, 2018). Existen varias tecnologías que implementan una arquitectura para orquestar contenedores: Kubernetes, Docker Swarm o Fleet.

3.6. Microservicios

Los microservicios son una arquitectura y un enfoque sobre la escritura de software en el que las aplicaciones se dividen en componentes más pequeños e independientes entre sí. A diferencia de un enfoque tradicional y monolítico sobre las aplicaciones, en el que todo se crea en una única pieza, los microservicios están separados y funcionan conjuntamente para llevar a cabo las mismas tareas como podemos ver en la figura 11. Cada uno de estos componentes, o procesos, son los microservicios. Este enfoque sobre el desarrollo de software valora la granularidad por ser liviana y la capacidad de compartir un proceso similar en varias aplicaciones. (Wha, 2019)

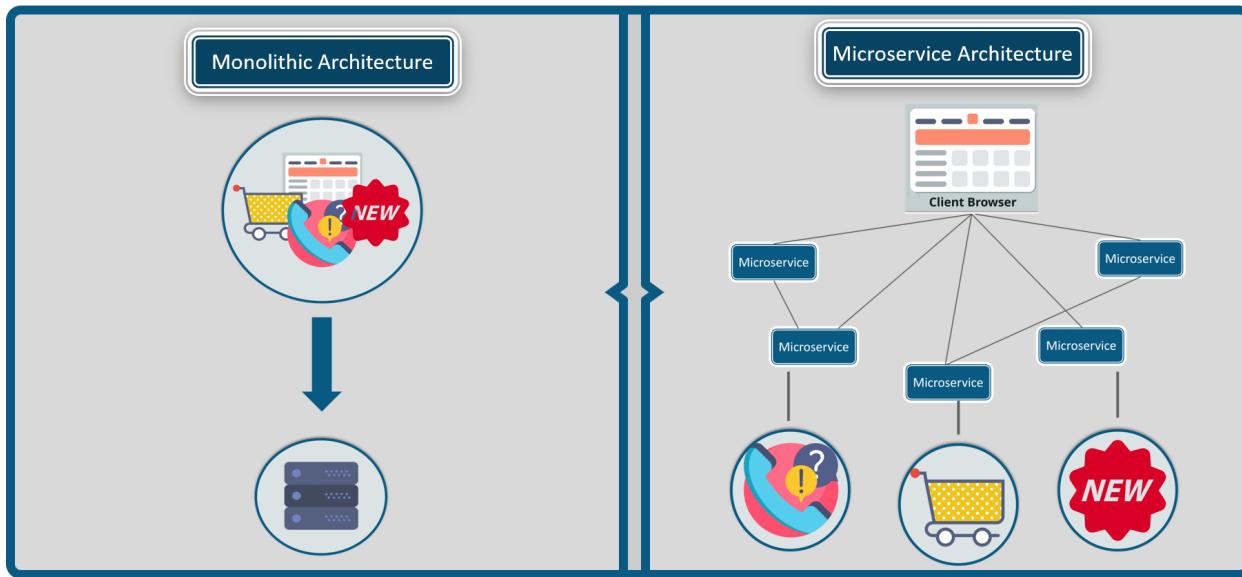


Figura 11. Arquitectura monolítica vs arquitectura de microservicios. Adaptado de (Kappagantula, 2019)

4. Marco técnico

En este capítulo se abordarán las tecnologías usadas durante el proyecto para implementar los conceptos abordados en el Capítulo 3.

4.1. Docker

Docker es una tecnología de creación y uso de contenedores Linux, para ello hace uso del kernel de Linux y funciones del mismo como Cgroups y namespaces para aislar procesos y que estos se puedan ejecutar de forma independiente.

Docker ofrece un modelo de implementación basado en imágenes lo cual permite compartir una aplicación o conjunto de servicios, junto con sus dependencias en varios entornos. Además, Docker automatiza la implementación de la aplicación (o conjuntos combinados de procesos que conforman una aplicación) dentro del entorno del contenedor (RHD, 2016).

Del ecosistema Docker, se destacan los siguientes componentes:

- **Contenedor:** Ambiente que permite ejecutar aplicaciones de manera aislada en un sistema anfitrión. Son creados a partir de imágenes. Haciendo una analogía con la programación orientada a objetos, son las instancias de las imágenes, es decir, a partir de una imagen pueden crearse n contenedores con el mismo estado.
- **Imagen:** Template que describe el estado inicial de un contenedor. Una imagen puede ser construida a partir de otra imagen disponible en un registro remoto. Haciendo una analogía con a la programación orientada a objetos, una imagen es una clase que puede ser instanciada n veces en forma de contenedor.
- **Registro:** Es un repositorio de imágenes. Puede ser público o privado, es decir que es posible trabajar con imágenes disponibles en internet. Uno de los registros públicos más populares es Docker Hub, el cual pertenece a Docker Inc.

4.2. Kubernetes

Es una plataforma open source que automatiza las operaciones de contenedores Linux. Elimina varios procesos en la implementación y escalabilidad de las aplicaciones en contenedores, lo cual nos permite construir y administrar un clúster de contenedores de forma eficiente y sencilla. Los clúster pueden ser construidos en nubes públicas, privadas, comunitarias o híbridas.

Un cluster Kubernetes es un conjunto de servidores que ejecutan diferentes servicios como podemos ver en la figura 12 (Kub, 2016b).

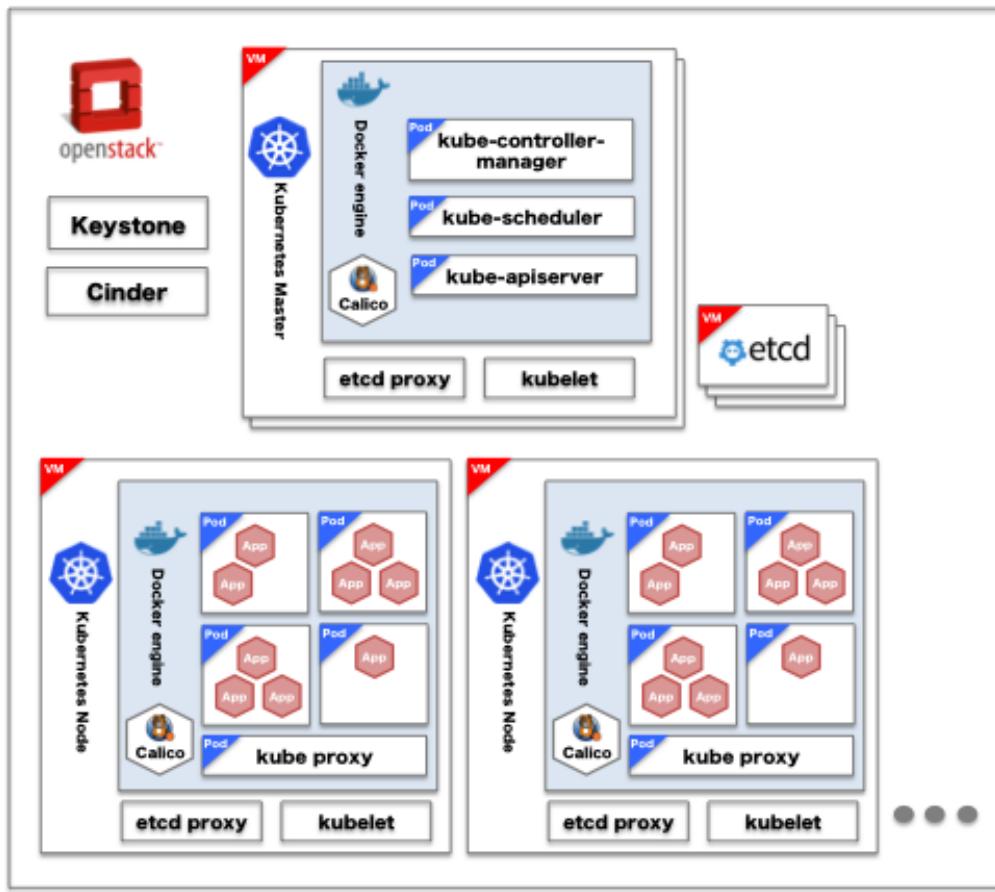


Figura 12. Arquitectura de un cluster Kubernetes. Adaptado de (Kub, 2016a)

4.3. Objetos Kubernetes

Los objetos Kubernetes son entidades que persisten dentro del sistema Kubernetes. Kubernetes usa estas entidades para representar el estado del cluster, para ello se definen en archivos en formato YAML o JSON. Específicamente, los objetos Kubernetes describen (Kub, 2019a):

- Qué aplicaciones se encuentran en ejecución.
- Los recursos disponibles para cada aplicación en ejecución.

- Las políticas del cluster.

El conjunto de objetos Kubernetes dentro del cluster define el estado deseado del cluster y Kubernetes se encargará de monitorear y gestionar los diferentes nodos con el fin de que el estado del cluster sea igual al estado deseado del mismo.

- **Pod:** Conjunto de contenedores. El pod es la unidad mínima en Kubernetes.
- **Service:** Interface entre cliente y la aplicación dentro de los pods. Es una pareja ip-puerto estática que se encarga de redirigir las solicitudes entre los diferentes pods disponibles.
- **Persistent Volumes:** Permite definir un sistema de almacenamiento dentro del cluster.
- **Persistent Volumes Claims:** Representa una solicitud para obtener recursos de almacenamiento del cluster.

4.4. Openshift

Openshift es un PaaS ofrecido por Red Hat con una integración nativa con Docker y Kubernetes, añadiendo funcionalidades de las cuales Kubernetes carece como (Cholewa, 2018):

- Integración con herramientas de integración continua como Jenkins.
- Manejo de diferentes proyectos en un mismo cluster.
- Plantillas que facilitan el despliegue de aplicaciones.

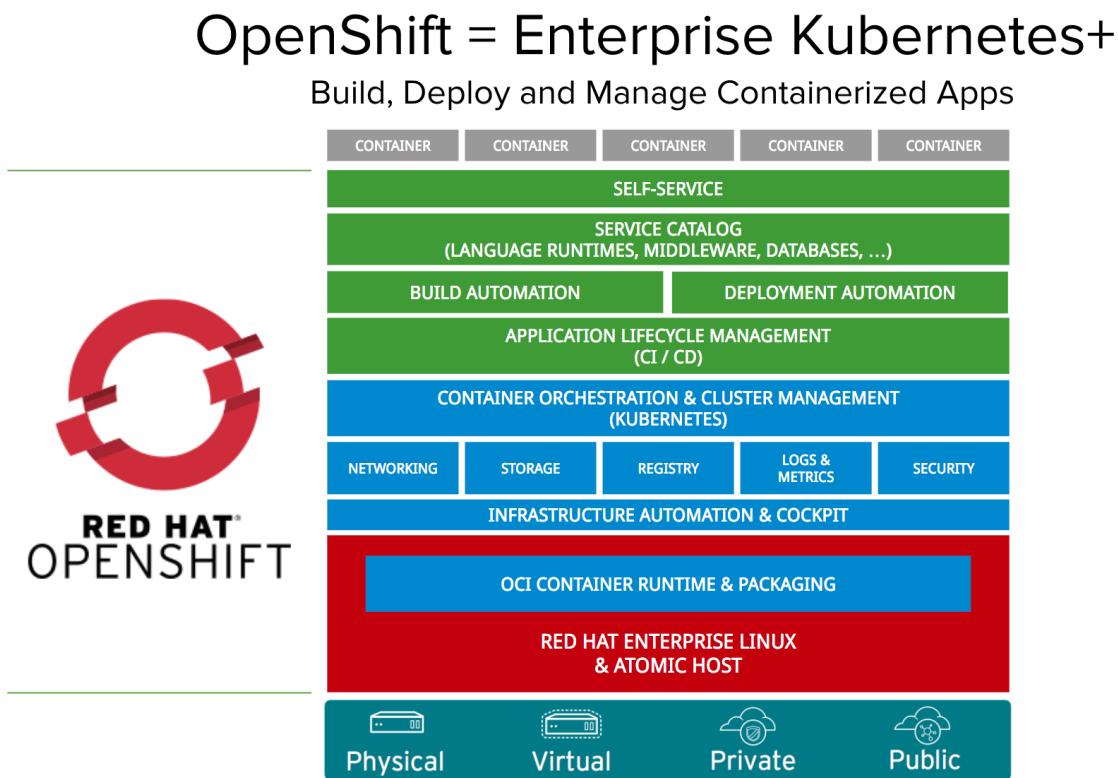


Figura 13. Arquitectura de Openshift. Adaptado de (Ope, 2017)

4.5. Ansible

Ansible es una herramienta para automatizar la administración de una infraestructura TI por medio de archivos de configuración llamados Ansible Playbooks en los cuales se describen los diversos pasos a automatizar(Ans, 2019).

En la arquitectura de Ansible, se tiene un archivo YAML, llamado Inventory, en el cual se describen los diferentes elementos de la infraestructura TI y variables de configuración. Posteriormente se ejecutan los playbooks usando el Inventory y Ansible indica si el proceso fue ejecutado con éxito o hubo algún problema durante alguno de los pasos del playbook.

4.6. GlusterFS

GlusterFS es un sistema de archivos en red escalable diseñado para tareas de alto uso de datos como almacenamiento cloud (glu, 2019).

- **Sistema de archivo en red** Es un sistema de archivos que distribuye los datos en diferentes nodos.
- **Brick** Es cualquier directorio o partición que es asignado a formar parte de un volumen.
- **Volumen** Es un conjunto lógico de Bricks. Las operaciones se basan en los diferentes tipos de volúmenes creados por el usuario.

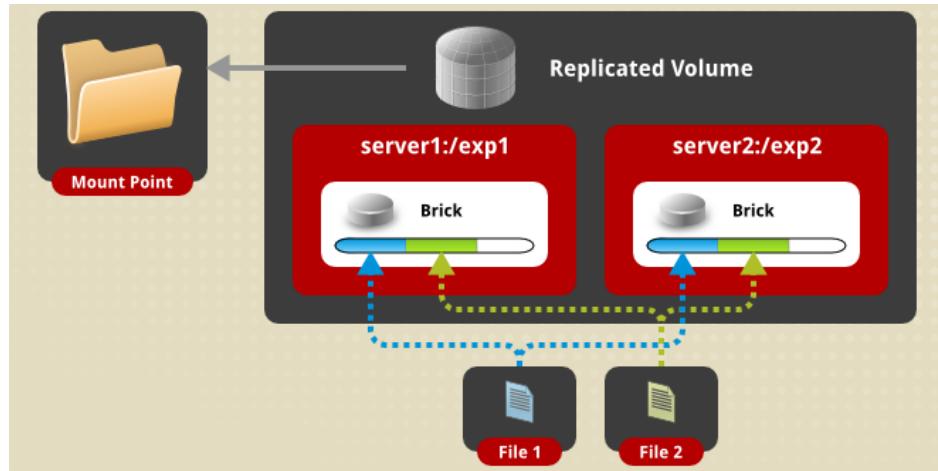


Figura 14. Arquitectura de Glusterfs. Adaptado de (glu, 2019)

5. Desarrollo del proyecto

En este Capítulo se presentará como fue desarrollado el proyecto.

5.1. Ambientación tecnológica

5.1.1. Investigación fundamentos teóricos relacionados con el proyecto (Tecnologías y estándares). Durante esta primera fase se investigó el estado del arte en cloud computing para poder identificar los conceptos comunes en las diferentes arquitecturas cloud enfocadas a IoT. De esta primera fase se encontró la gestión de contenedores como factor común por lo cual la investigación posterior siguió este enfoque. Posteriormente se identificaron las tecnologías más populares en IoT.

Tabla 1
Tecnologías disponibles en el mercado

Sistema Operativo	Container Runtime	Orquestador de contenedores
CentOS	Cri-O	Apache Mesos
CoreOS	Docker	Docker Swarm
Red Hat Enterprise Linux	Linux VServer	Kontena
Oracle Linux	LXD	Kubernetes
Ubuntu Server	Rkt	Nomad
Windows Server	Windows Containers	Openstack Magnum x

A medida que se desarrolló en proyecto, nuevas tecnologías fueron añadidas al stack pero ya de eso se hablará más adelante en el documento.

A medida que se desarrolló en proyecto, nuevas tecnologías fueron añadidas al stack pero ya de eso se hablará más adelante en el documento.

5.1.2. Estudio. Durante esta etapa se realizó una revisión general sobre cada una de las tecnologías, lo cual sirvió de base para la definir los criterios de selección para realizar un primer filtro sobre las tecnologías a usar en la infraestructura en la siguiente etapa.

- Licencia
- Stacks
- Soporte
- Uso libre en producción

5.1.3. Sondeo. Durante esta etapa se aplicaron los diferentes criterios entre cada una de las tecnologías y se consolidó está información en las tablas 2, 3 y 4.

Tabla 2
Sondeo de selección: Sistema Operativo

S. Operativo	Licencia	Stacks	Soporte	Gratis en producción
CentOS	GPL	1.85K	Comunidad	Si
CoreOS	Apache 2.0	164	Comunidad	Si
RHEL ^a	GPL ^b	-	Privado	No
Oracle Linux	GPL	-	Privado	Si
Ubuntu Server	GPL	9.1k	Híbrido ^c	Si
Windows Server	Múltiple ^d	3.4k	Privado	No

La cantidad de stacks fueron tomados de: <https://stackshare.io/>

^a Red Hat Enterprise Linux

^b Solo para efectos de desarrollo, en producción se cobra una tarifa.

^c Posibilidad de adquirir soporte pago pero posee una gran comunidad que lo respalda.

^d Microsoft provee diferentes licencias según los servicios solicitados.

Tabla 3
Sondeo de selección: Container Runtime

Container Runtime	Licencia	Stacks	Soporte	Gratis en producción
Cri-O	Apache 2.0	-	Comunidad	Si
Docker	Apache 2.0	16.6K	Comunidad/Privado	Si
Linux VServer	GPL	-	Comunidad	Si
LXD	Apache 2.0	39	Comunidad	Si
Rkt	Apache 2.0	21	Comunidad	Si
Windows Containers	SaaS	-	Privado	No

La cantidad de stacks fueron tomados de: <https://stackshare.io/>

Tabla 4
Sondeo de selección: Orquestador de contenedores

Orquestador	Licencia	Stacks	Soporte	Gratis en producción
Apache Mesos	Apache 2.0	177	Comunidad	Si
Docker Swarm	Apache 2.0 ^a	319	Comunidad/Privado	Si ^b
Kontena	Apache 2.0	7	Comunidad	Si
Kubernetes	Apache 2.0	4.25k	Comunidad	Si
Nomad	Mozilla 2.0	56	Comunidad/Privado	Si ^c
Docker Compose	Apache 2.0	3.19k ^d	Comunidad/Privado	Si

La cantidad de stacks fueron tomados de: <https://stackshare.io/>

^a Algunos componentes se encuentran bajo la licencia Apache 2.0 y otros bajo Docker Enterprise 2.0 y 2.1

^b Si se desea utilizar Docker Trusted Registry o Universal Control Pane, si es necesario adquirir una licencia.

^c HashiCorp ofrece planes pagos con funcionalidades adicionales

^d Se utilizó la cantidad de stacks de Openstack

5.2. Definición de arquitectura de infraestructura cloud

5.2.1. Definición requisitos. Se identificaron las características que debe tener una infraestructura, haciendo uso de las características esenciales de un modelo de cloud computing

(Novkovic, 2017).

- **Autoservicio sobre demanda:** Los recursos cloud del proveedor pueden ser asignados a los clientes sin la necesidad de interacción humana. Algunos proveedores proveen interfaces web a sus clientes con el fin de que estos puedan acceder a los recursos directamente.
- **Amplio acceso a la red:** Los recursos cloud están disponibles por medio de la red y pueden ser accedidos por las plataformas de los diferentes clientes, ya sea por Internet o una LAN⁴ en el caso de una nube privada.
- **Agrupación de recursos:** Hace referencia al uso compartido físicos por diferentes clientes. Para efectos del despliegue de la plataforma IoT, es necesario compartir los recursos entre los diferentes artefactos a desplegar dentro de la infraestructura.
- **Elasticidad:** Una de las grandes ventajas del cloud computing es la capacidad de proveer recursos dinámicamente en la medida en que las aplicaciones, por efectos de la demanda, lo vayan requiriendo. En otras palabras, una aplicación tendrá asignados pocos recursos cuando haya poca demanda pero a medida que esta aumenta, la infraestructura cloud aumenta los recursos asignados con el fin de que la aplicación pueda responder adecuadamente a la demanda. Esta propiedad es imprescindible en un entorno IoT.
- **Servicio medido:** Es fundamental poder medir los recursos usados por una aplicación con

⁴ Red de área local

el fin de identificar posibles puntos de cuello botella dentro de la plataforma IoT.

- **Políglota:** Teniendo en cuenta que la plataforma ha sido diseñada para ser extensible, es importante que la infraestructura permita el despliegue de artefactos independientemente del lenguaje en el cual ha sido desarrollado el nuevo artefacto a integrar.
- **Soporte de microservicios:** Durante la primera fase del proyecto se definió que el backend sería desarrollado bajo una arquitectura de microservicios, por lo cual es prioridad que la infraestructura permita el despliegue de una solución desarrollada bajo esta arquitectura.
- **Soporte para DevOps:** El uso de DevOps dentro de un proyecto software permite automatizar aquellas tareas mecánicas con el fin de que los diferentes integrantes del equipo de desarrollo puedan concentrarse en tareas de mayor valor para el proyecto. Adicionalmente, se elimina el factor del error humano durante la fase de despliegue de las nuevas versiones de la plataforma.
- **Seguridad:** Es un elemento importante en una arquitectura cloud ya que provee confianza a los usuarios con respecto a que la información que fluye por la plataforma es transmitida de forma segura.
- **Tolerancia a fallos:** Dentro de un entorno IoT es fundamental que la infraestructura sea tolerante a fallos para evitar la perdida de datos enviados por los diferentes dispositivos conectados a la plataforma en caso de un fallo.

5.2.2. Definición de métricas. La alta disponibilidad y elasticidad son las características cruciales en un entorno IoT en el cual tendremos una cantidad masiva de dispositivos conectados enviando datos constantemente. Para poder medir estas características se definieron las siguientes métricas:

- **Cantidad de solicitudes fallidas durante un despliegue en un entorno de 1000 solicitudes por minuto:** Durante el despliegue de una nueva versión de un sistema, este suele quedar fuera de servicio por un tiempo lo que se traduce, en un entorno IoT, en una cantidad determinada de mensajes perdidos. Por esta razón es importante determinar la cantidad aproximada de solicitudes que pueden fallar durante el despliegue. La característica que se evalúa en esta métrica es la alta disponibilidad.
- **Tiempo promedio de respuesta para 1000 solicitudes por minuto:** La plataforma IoT permite a otros sistemas conectarse con el fin de obtener los datos que fluyen por la plataforma. Por consiguiente, es importante evaluar el rendimiento de la plataforma en función de su escalabilidad.
- **Número máximo de solicitudes por minuto:** Teniendo en cuenta que la infraestructura debe soportar una gran cantidad de dispositivos conectados enviando solicitudes masivamente, es importante determinar el número máximo de solicitudes con el fin de aumentar la capacidad de la infraestructura cuando se esté llegando a rozar esta cantidad de solicitudes.
- **Porcentaje de solicitudes fallidas para 3500 solicitudes por minuto:** Permite tener un

punto de comparación equitativo entre los diferentes escenarios con la misma cantidad de solicitudes.

- **Cantidad de solicitudes fallidas para 1000 solicitudes por minuto durante el fallo de un nodo:** El fallo de un nodo es un escenario critico en un ambiente IoT por lo cual es importante medir la cantidad de solicitudes que pueden fallar durante este escenario.

5.2.3. Definición de pruebas. Para las pruebas, se plantearon los siguientes escenarios:

1. **Monolítico:** Es el escenario tradicional en el cual se tienen todos los componentes del sistema instalados en una misma máquina.
2. **Escenario distribuido con 1 instancia del backend:** Dentro de la infraestructura desplegará una instancia del backend.
3. **Escenario distribuido con 2 instancias del backend:** Dentro de la infraestructura despliegan dos instancia del backend.
4. **Escenario distribuido con 3 instancias del backend:** Dentro de la infraestructura despliegan tres instancia del backend.

Los siguientes métodos fueron seleccionados para aplicar las pruebas.

Tabla 5

Métodos usados en las pruebas sobre el prototipo

Método HTTP	Ruta	Descripción
POST	/users/user	Crea un nuevo usuario en la base de datos
POST	/users/authentication	Verifica las credenciales de un usuario
PUT	/users/user	Actualiza un usuario en la base de datos
DELETE	/users/user	Elimina un usuario en la base de datos

Cada una de las pruebas busca medir cada una de las diferentes métricas (sección 5.2.2) para los 4 escenarios planteados durante 10 minutos.

- 1. Cantidad de solicitudes fallidas durante un despliegue en un entorno de 1000 solicitudes por minuto** La prueba consiste en realizar un cambio de versión del backend mientras se están recibiendo solicitudes y medir la cantidad de solicitudes que fallan durante el proceso de despliegue. Para el ambiente monolítico, se tendrá un script que detenga una versión y arranque una nueva con el fin de evitar el error humano. Para el ambiente distribuido se hará uso del Continuous Deployment en la arquitectura propuesta.

2. **Tiempo promedio de respuesta para 1000 solicitudes por minuto** La prueba consiste en enviar 1000 solicitudes por un minuto a cada uno de los ambientes y medir el tiempo que tardan las solicitudes en ser procesadas y llegar al cliente.
3. **Número máximo de solicitudes por minuto sin solicitudes fallidas** La prueba consiste en enviar solicitudes de forma masiva sobre los diferentes ambientes para encontrar la máxima cantidad de solicitudes por minuto que pueden soportar sin presentar solicitudes fallidas.
4. **Porcentaje de solicitudes fallidas para 3500 solicitudes por minuto** La prueba consiste en enviar 1000 solicitudes por minuto y medir el porcentaje de solicitudes fallidas en cada uno de los ambientes.
5. **Cantidad de solicitudes fallidas para 1000 solicitudes por minuto durante el fallo de un nodo** La prueba consiste en apagar un nodo de forma abrupta mientras se están recibiendo 1000 solicitudes y medir la cantidad de solicitudes fallidas mientras el sistema vuelve a recuperarse del fallo. En el caso del ambiente monolítico, se reiniciará el servidor.

Para generar un dataset para las pruebas, se utilizó Mockaroo⁵ (Ver apéndice 1).

5.2.4. Definición arquitectura e infraestructura. La elección de los componentes de la infraestructura fue hecha teniendo en cuenta el impacto de las diferentes tecnologías en el mercado. Como podemos ver en la tabla 4, **Kubernetes** es la opción más popular, además de tener

⁵ <https://www.mockaroo.com/>

una licencia que nos permite hacer uso gratuito de la herramienta en producción junto a una de las comunidades más grandes entre las otras opciones estudiadas.

Profundizando sobre Kubernetes, apareció **Openshift Origin** (Ope, 2017), un proyecto de Red Hat que extiende las funcionalidades de Kubernetes. Se revisó la licencia y las ventajas con respecto a Kubernetes, como la seguridad. Finalmente se decidió usar Openshift Origin como plataforma para administrar Kubernetes dentro de la arquitectura.

Docker se seleccionó como container runtime por la popularidad, comunidad y uso libre en producción.

CentOS se seleccionó como sistema operativo fue por la comunidad, madurez y estabilidad que presenta con respecto a las alternativas.

La arquitectura se compone de los siguientes elementos:

- **Reverse proxy:** Es la puerta de entrada al backend. Su función es redirigir las solicitudes entrantes al componente de la capa service correspondiente basado en el endpoint solicitado, de esta forma podemos exponer todos los microservicios como si de un solo se tratase. Se seleccionó **Nginx** como proxy reverso por su alto rendimiento y bajo consumo de memoria.

Douglas et al. (2017)

- **Service layer:** En esta capa se encuentra una abstracción de los pods que nos provea un dominio fijo. Para la infraestructura se escogió usar los objetos **Kubernetes Services** los cuales nos proveen un nombre de dominio fijo para el acceso a la capa pod además de proveer balanceo de carga en el caso de los componentes que tienen más de una

instancia.

- **Deployment layer:** En esta capa se encuentran los archivos de definición de los elementos de la capa pod. Esta capa se encuentra conectada a un repositorio git por medio de una tecnología de CI/CD de tal forma que cuando los desarrolladores suban un cambio a la rama master del repositorio, se realice todo el proceso de compilado, publicación y actualización de los archivos de despliegue correspondientes. Para la infraestructura, se escogió usar los objetos **kubernetes deployment**.
- **Pod layer:** En esta capa se encuentran las instancias efímeras de la arquitectura. Para la arquitectura se escogió usar los objetos **Kubernetes pods** (Instancias de la capa Openshift deployment) de cada uno de los elementos desplegados. En el caso de los artefactos que poseen más de una instancia, como es el caso de admin-microservice o data-microservice, Openshift provee la funcionalidad de despliegue canario manteniendo $n-1^6$ instancias activas mientras que las nuevas instancias se encuentran listas para recibir solicitudes.
- **Persistence logic layer:** En esta capa se encuentra la conexión lógica con la persistence physical layer. Para la infraestructura se escogió usar los objetos **Kubernetes Volume**, los cuales conectan los pods con la persistence physical layer(Kub, 2019b). Las bases de datos y los brókers de mensajería son los principales artefactos que usan esta capa.

⁶ n: Número de instancias activas

- **Persistence physical Layer** En esta capa se encuentra el sistema de archivos para persistir datos en disco. Para el caso de la infraestructura se propone usar **glusterfs** por que nos permite distribuir de forma redundante los archivos en diferentes nodos, de tal manera que si alguno de los nodos falla, el sistema puede continuar haciendo uso de otro nodo para seguir persistiendo la información.
- **Code repository** Sistema de archivos en el cual se aloja el código fuente. Se seleccionó **git** como code repository por su popularidad y relevancia en el mercado. Por otra parte, **Gitlab** se escogió como proveedor por la gran cantidad de herramientas que ofrece para la gestión de proyectos software.
- **CI/CD** Entre las herramientas que ofrece **gitlab** se encuentra un sistema de CI/CD basado en pipelines. La gran ventaja que tiene con respecto a otras alternativas es su sencillez de integración en arquitecturas cloud.

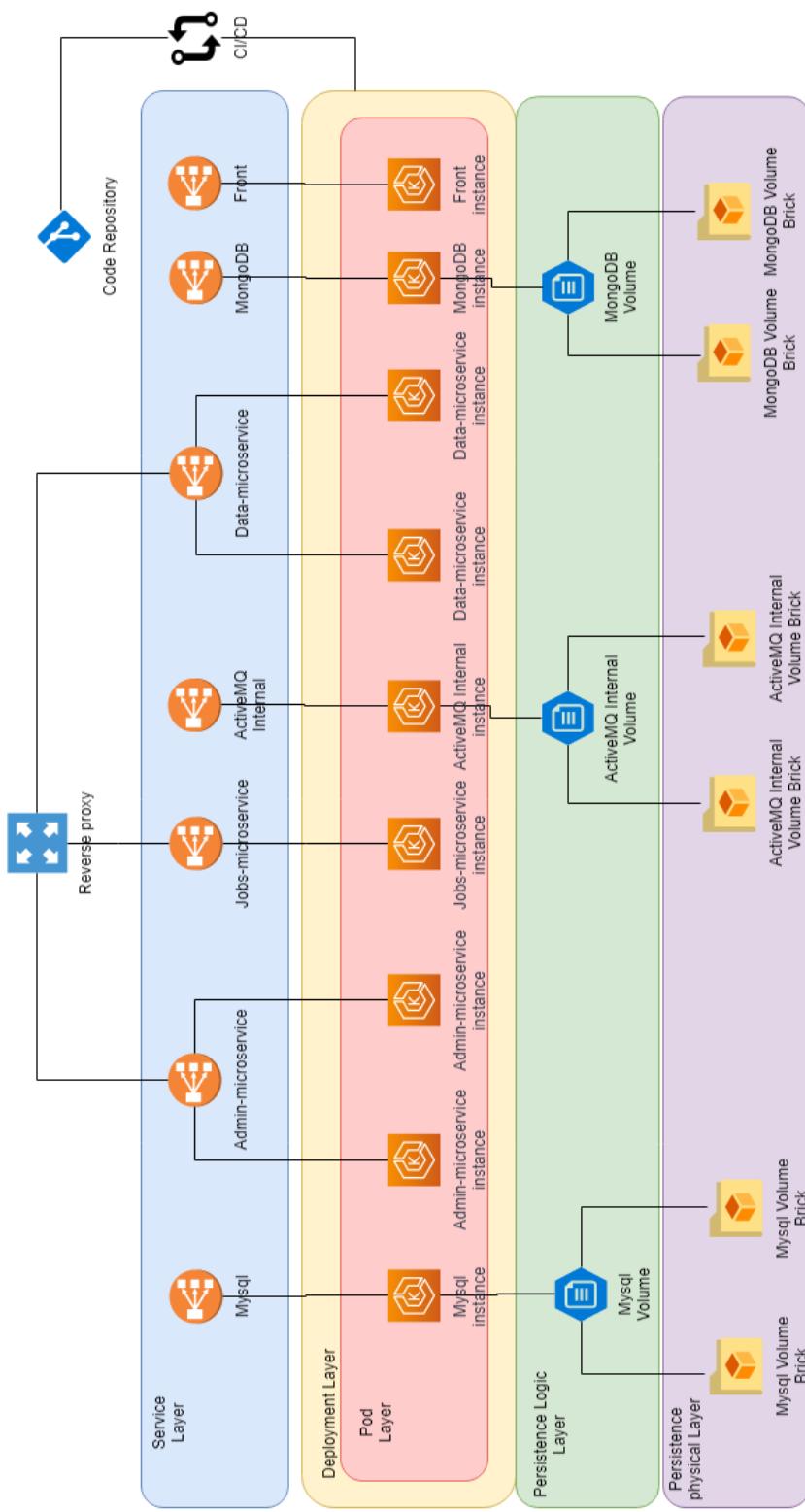


Figura 15. Arquitectura propuesta.

5.3. Prototipado

5.3.1. Configuración básica cluster. El cluster se conformó por 3 máquinas físicas, sin embargo para la configuración mínima de alta disponibilidad se necesitaron 5 servidores, por lo cual la máquina con más recursos se usó para la virtualización de 3 máquinas con KVM(Machado, 2016) como podemos ver en la tabla 6 y en la figura 16.

Tabla 6
Características de máquinas en cluster

Nombre de dominio	CPU	# Cores	RAM (GB)
dns.local.cluster	Intel core i7 4720H	1	1
glusterfs.local.cluster	Intel core i7 4720H	1	0.59
master1.local.cluster	Intel core i7 4720H	4	5.781
node1.local.cluster	Intel core i5 3230M	4	5.697
node2.local.cluster	Intel core i3 4005U	4	3.223

Para cada uno de las máquinas se configuró la ip de forma estática y una clave RSA, la cual será usada posteriormente para el acceso remoto a los mismos.

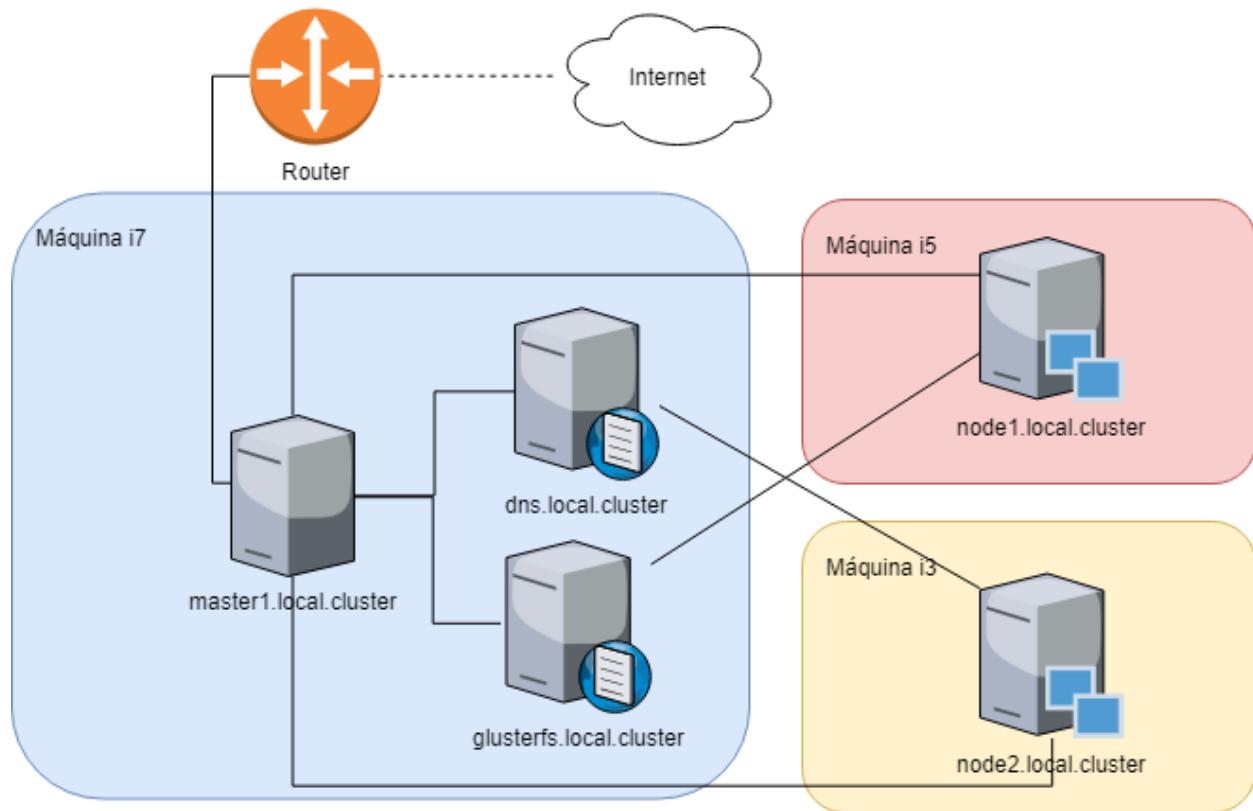


Figura 16. Infraestructura hardware del prototipo.

5.3.2. Configuración infraestructura cluster.

5.3.2.1. Servidor DNS. Se utilizó la máquina dns.local.cluster como servidor DNS

usando bind(Kumarpalani, 2014) con la siguiente configuración:

Tabla 7
Resolución de nombres en cluster

Nombre de dominio	IP
dns.local.cluster	192.168.0.133
glusterfs.local.cluster	192.168.0.134
master1.local.cluster	192.168.0.130
node1.local.cluster	192.168.0.131
node2.local.cluster	192.168.0.132

Adicionalmente se instaló glusterfs para posteriormente distribuir el almacenamiento entre los servidores glusterfs.local.cluster y dns.local.cluster.

5.3.2.2. Servidor Glusterfs. Junto con el equipo de desarrollo de la plataforma IoT se definieron los siguientes artefactos a desplegar dentro de la infraestructura.

- Proxy Reverso
- Servidor Mysql
- Servidor MongoDB
- Servidor ActiveMQ
- Backend

- Frontend

Posteriormente se clasificaron los artefactos según la necesidad de persistir información de la siguiente forma:

Tabla 8

Artefactos de la aplicación IoT según su persistencia

Con persistencia	Sin persistencia
Servidor Mysql	Proxy Reverso
Servidor MongoDB	Frontend
Servidor ActiveMQ	Backend

En nuestra arquitectura se escogió GlusterFS como el sistema de archivos distribuidos, por lo que para cada uno de los artefactos de la arquitectura que requieren persistencia, se configuró un volumen GlusterFS de la siguiente forma:

Tabla 9

Configuración de bricks GlusterFS

Volumen	Usuario Id	Grupo Id	Permisos
activemq	0	0	755
mongo	184	184	777
mysql	27	0	777

Finalmente se crea cada uno de los volúmenes con bricks tanto en el servidor dns.local.cluster como en glusterfs.local.cluster de tal forma que si alguno de los 2 servidores falla, esté el otro servidor para proveer los archivos y así asegurar en ese caso la disponibilidad del sistema.

5.3.2.3. Nodos. Openshift Origin provee 2 métodos de instalación: Basado en paquetes RPM y basado en un sistema de contenedores(ope, 2019).

Sin embargo la instalación por sistema de contenedores requiere de acceso al repositorio privado de contenedores de Red Hat, por lo que se decidió usar la instalación basada en paquetes RPM.

La instalación se realizó usando Ansible con el archivo Inventory del anexo 2. En este archivo definimos las diferentes máquinas y su rol dentro de la infraestructura.

Posteriormente se ejecutaron los siguientes ansible-playbook:

1. **prerequisites.yaml** Instala y actualiza los paquetes software necesarios para la instalación del cluster.
2. **deploy_cluster.yaml** Instala e inicia el cluster dentro de la infraestructura.

Luego se modificó el archivo /etc/origin/master/htpasswd, añadiendo el usuario 'admin' con contraseña 'admin'.

Finalmente se ingresa a la plataforma web de administración de Openshift como se ve en la imagen 17.

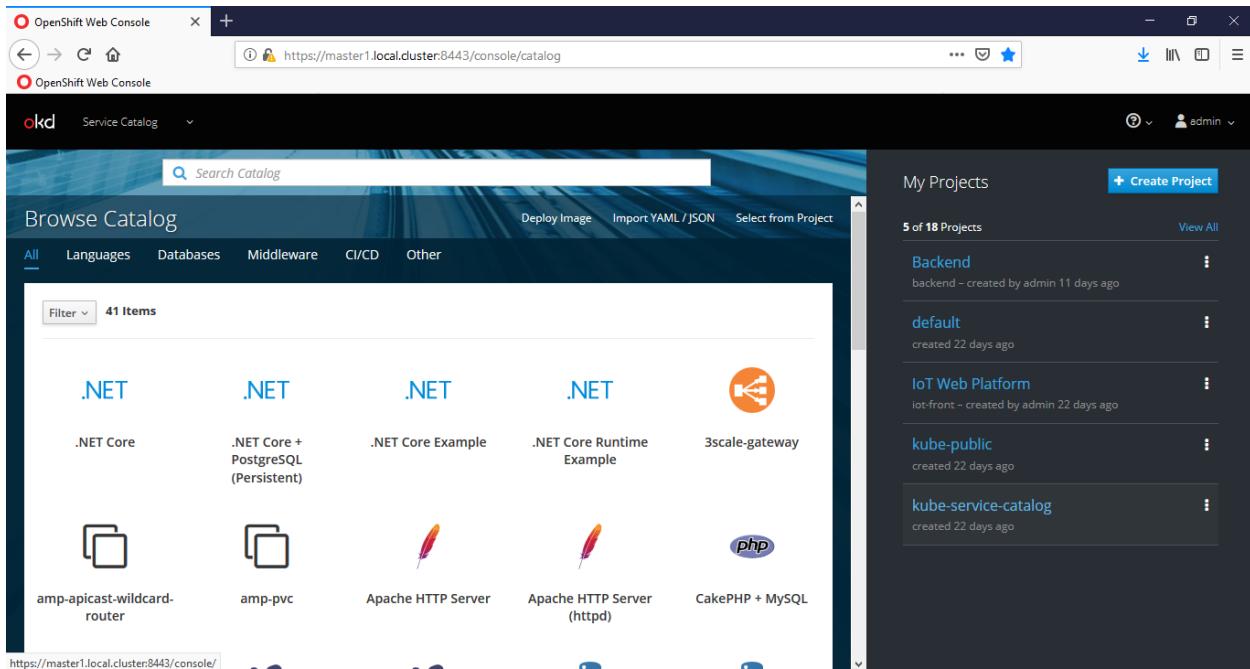


Figura 17. Plataforma web de administración de Openshift.

Seguidamente, se crearon 2 proyectos: iot-front y backend.

```
[root@master1 ~]# oc new-project iot-front
Now using project "iot-front" on server "https://master1.local.cluster:8443"
 ↵ .

[root@master1 ~]# oc new-project backend
Now using project "backend" on server "https://master1.local.cluster:8443".
```

Finalmente, se desplegó el archivo gluster-endpoints.yaml del anexo 3 en el proyecto backend para registrar los nodos en los cuales se encuentran configurados los bricks de GlusterFS.

```
[root@master1 ~]# oc project backend  
Now using project "backend" on server "https://master1.local.cluster:8443".  
[root@master1 ~]# oc create -f gluster-endpoints.yaml  
endpoints/glusterfs-cluster created
```

5.3.3. Despliegue aplicación. Durante el proyecto se definió el siguiente proceso para el despliegue de un artefacto dentro de la arquitectura propuesta con el fin de simplificar el proceso de despliegue y facilitar la integración de nuevos componentes a la arquitectura.

Los archivos de configuración⁷ para cada uno de los artefactos se encuentran en el anexo 4.

Adicionalmente se realizó un manual de instalación de la arquitectura disponible en el anexo

5.

⁷ Dockerfile, pv.yaml, pvc.yaml, deployment.yaml, service.yaml

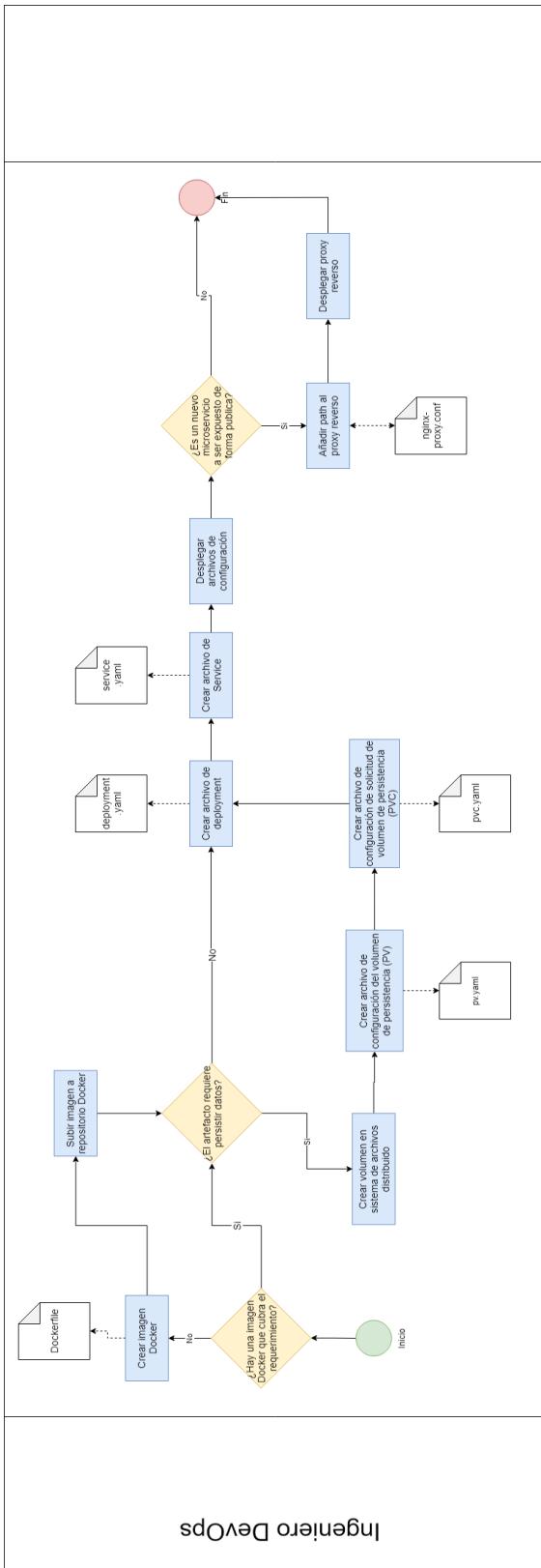


Figura 18. Proceso de despliegue de un artefacto dentro de la arquitectura propuesta en BPMN.

Ingeniero DevOps

5.3.4. Automatización del proceso de despliegue. Para realizar la automatización del proceso de despliegue se realizó usando GitLab CI/CD. En el proyecto se tienen 2 automatizaciones de despliegue, una para el proyecto del frontend y otro para el proyecto del backend. Estos se encuentran en el anexo 6.

Se definieron 3 etapas:

1. **Build:** En esta etapa se compila y ejecutan los casos de prueba del código a desplegar.
2. **Publish:** En esta etapa se crea de forma automática los contenedores con el código compilado y se publican en Docker Hub.
3. **Deploy:** En esta etapa se actualiza y despliega el archivo de deployment.yaml en el cluster.

De esta manera, cada vez que los desarrolladores realizan un cambio sobre la rama 'master', automáticamente se ejecutan las 3 etapas para el despliegue de la aplicación en el cluster. En caso de haber un error, Gitlab envía un correo a los miembros del proyecto indicando que hubo un problema durante el despliegue.

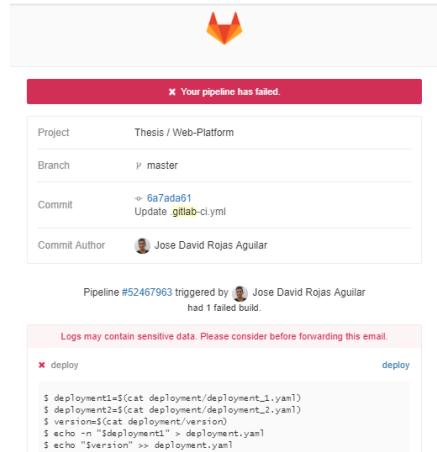


Figura 19. Mensaje enviado por Gitlab cuando ocurre un error durante el despliegue automático.

5.3.5. Implementación de pruebas. Para la implementación de las pruebas se realizó el siguiente plan de pruebas con Jmeter. El plan de pruebas consistió en enviar una solicitud para crear, consultar, modificar y eliminar un usuario.

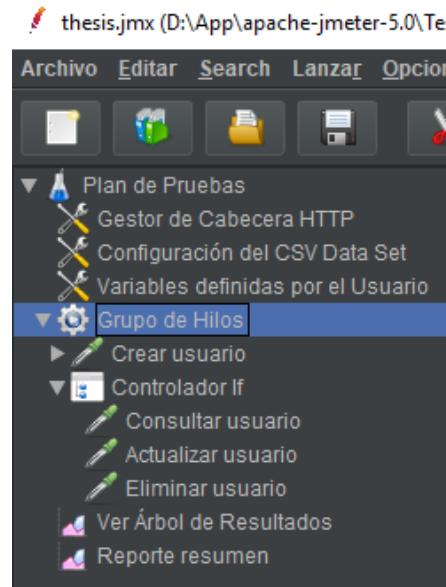


Figura 20. Plan de pruebas implementado en Jmeter.

La actividad 'Reporte resumen' fue la fuente de los datos presentados en los resultados de

la ejecución de las pruebas.

5.4. Validación de prototipo

5.4.1. Aplicación pruebas. Los resultados de las pruebas se consignaron en las tablas 10, 11, 12, 13 y 14.

Tabla 10

Cantidad de solicitudes fallidas durante un despliegue en un entorno de 1000 solicitudes por minuto

Escenario	Solicitudes
Monolítico	316
1 instancia	1051
2 instancias	326
3 instancias	320

Tabla 11

Tiempo promedio de respuesta para 1000 solicitudes por minuto

Escenario	Milisegundos
Monolítico	413
1 instancia	455
2 instancias	434
3 instancias	419

Tabla 12

Número máximo de solicitudes por minuto sin solicitudes fallidas.

Escenario	Solicitudes por minuto
Monolítico	2690
1 instancia	2616
2 instancias	3800
3 instancias	4320

Tabla 13

Porcentaje de solicitudes fallidas para 3500 solicitudes por minuto

Escenario	%
Monolítico	24.26
1 instancia	36.28
2 instancias	0
3 instancias	0

Tabla 14

Cantidad de solicitudes fallidas para 1000 solicitudes por minuto durante el fallo de un nodo

Escenario	Solicitudes fallidas
Monolítico	173
1 instancia	157
2 instancias	77
3 instancias	60

5.4.2. Análisis resultados.

- Durante la prueba de la tabla 10, la mayor cantidad de solicitudes fallidas se evidenció en el ambiente de 1 instancia ya que este no es compatible con el despliegue en release de

Kubernetes. Los ambientes de 2 y 3 instancias soportaban el tráfico entrante con la versión anterior mientras la nueva versión estaba lista.

- El ambiente monolítico tiene el tiempo de respuesta más óptimo en la prueba de la tabla 11 ya que este ambiente no debe realizar transferencia de datos por medio de la red.
- El número máximo de solicitudes es proporcional a la cantidad de instancias ya que el tráfico se distribuye en las mismas.
- Durante la prueba de la tabla 13 se evidenció que un ambiente monolítico tiene un mayor rendimiento que un ambiente 1 instancia ya que el ambiente monolítico debe transferir datos por medio de la red, lo cual impacta significativamente en su rendimiento.
- Como vemos en la tabla 14, la redundancia de recursos es fundamental para disminuir el impacto de un fallo en la respuesta de las solicitudes.
- A partir de 2 instancias, la carga se distribuye entre los diferentes servidores, lo cual lleva a una mejora significativa en el rendimiento de la arquitectura.
- La mejora de rendimiento entre 1 instancia y 2 instancias es mucho más significativa que la mejora entre 2 instancias y 3 instancias. Esto se debe a que para el primer caso, hay un cambio en la distribución de carga entre los servidores físicos, mientras que en el segundo caso, no hay un cambio de carga a nivel físico ya que siguen siendo 2 servidores físicos, el cambio se da a nivel software.

5.5. Caso de uso

En esta sección se realizó el despliegue de la plataforma IoT enfocada a Smart Campus, implementando un prototipo para demostrar que todos los componentes se integran correctamente. Para lograr esto, fue necesaria la participación de los autores de los proyectos de grado mencionados en los capítulos anteriores.

En el escenario planteado un usuario desea activar una red de bombillos ubicados en dos laboratorios de ingeniería eléctrica cuando el voltaje de un generador eléctrico pase un umbral de seguridad para alertar al personal presente en dichos espacios. Además, el usuario desea monitorear la temperatura generada en uno de los laboratorios con el objetivo de proteger de temperaturas extremas los dispositivos allí presentes.

Para simular el escenario anterior se hizo uso de dos dispositivos tipo gateway, tres actuadores (LEDs), un sensor de temperatura y un potenciómetro; a su vez, se crearon 4 procesos y 1 aplicación externa en la que se muestran los datos capturados en el caso de uso.

La arquitectura planteada para el caso de uso se encuentra en la figura 21.

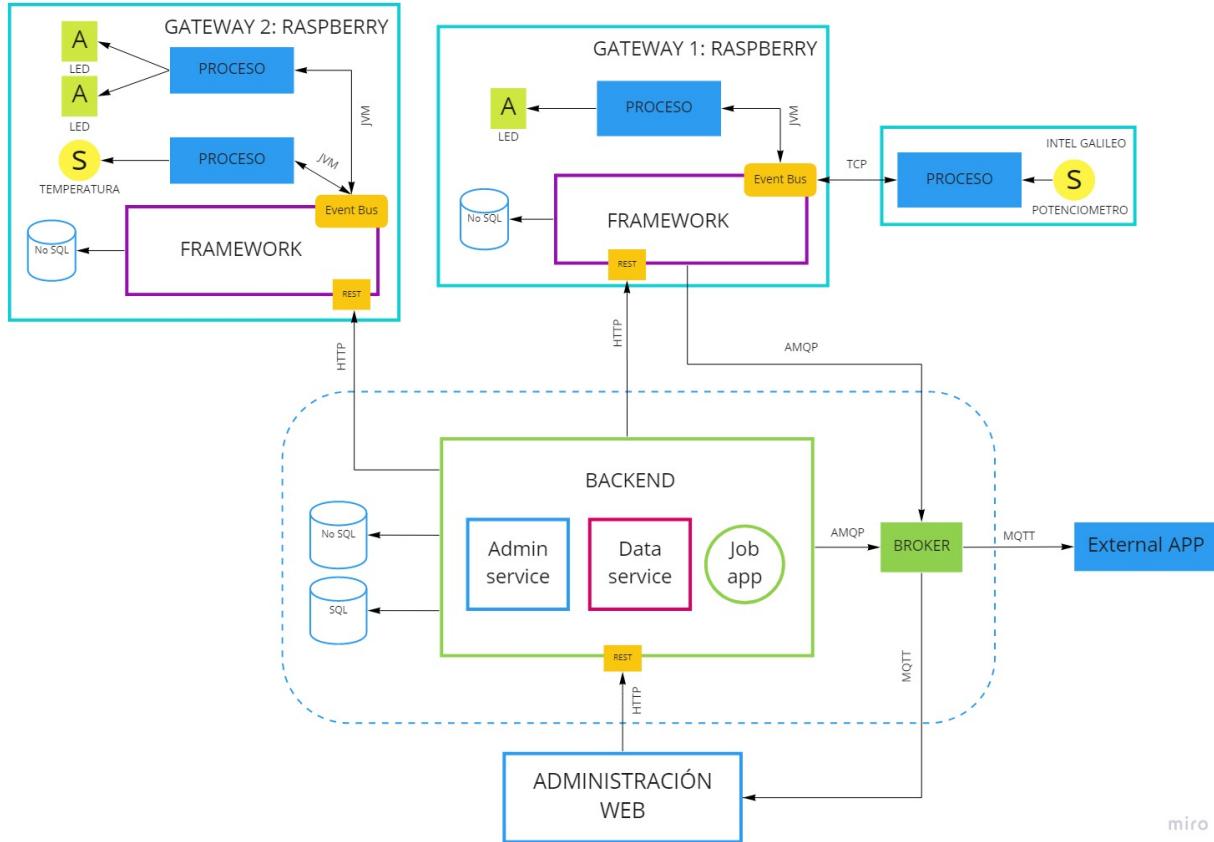


Figura 21. Caso de uso.

Para el caso de uso se desplegaron las últimas versiones del backend y frontend dentro del prototipo usado en la sección 5.3.

En la infraestructura se desplegaron 2 instancias del frontend y 3 instancias del backend.

Durante la prueba se monitoreó el uso de recursos en los servidores.

Durante una fase de la prueba se apagó uno de los nodos y el sistema continuó funcionando con normalidad gracias a la alta disponibilidad que provee la infraestructura desplegada a la plataforma IoT. Kubernetes detecta cuando falla el nodo y comienza a levantar instancias en el nodo restante con el fin de mantener consistente la cantidad de instancias activas dentro de la

infraestructura.

6. Conclusiones

A partir de los desarrollos presentados y los resultados obtenidos en el presente trabajo de grado, se llegó a las siguientes conclusiones:

1. La elección de herramientas software adecuadas es fundamental durante la primera fase de un proyecto software ya que son estas las bases técnicas sobre las cuales se construirá el proyecto. Durante la primera fase del proyecto Se logró identificar las diferentes herramientas en el mercado y escoger de estas las más adecuadas a partir de un conjunto de requerimientos extraídos en conjunto con el equipo de desarrollo de la plataforma IoT.
2. Una propiedad solicitada en un entorno de IoT es la alta disponibilidad ya que habrá una masiva cantidad de dispositivos conectados a la plataforma constantemente. Para soportar esta propiedad se diseñó una arquitectura⁸ en un ambiente distribuido que posteriormente se validó por medio de las pruebas realizadas sobre un prototipo.
3. Una característica importante dentro de una arquitectura software es su extensibilidad. Para simplificar el proceso de ingreso de nuevos componentes a la arquitectura original, se definió una metodología con un conjunto de convenciones para asegurar el despliegue de artefactos dentro de la infraestructura planteada.
4. El despliegue continuo es un componente fundamental en el desarrollo de proyecto software

⁸ Arquitectura explicada en la sección 5.2.4

modernos ya que automatiza tareas repetitivas del proceso de despliegue lo que permite a los diferentes miembros del equipo enfocarse en tareas de mayor valor para el proyecto. Durante el proyecto se integró con éxito un sistema de despliegue continuo dentro de la arquitectura lo cual permitió evidenciar la importancia de integrar una solución de despliegue continuo en un proyecto software.

7. Trabajo futuro

- Explorar los diferentes plugins que ofrece la comunidad de Openshift Origin.
- Utilizar servidores con las características recomendadas por la documentación de Openshift Origin.
- Hacer la instalación de los diferentes componentes directamente sobre las máquinas y no recurrir a la virtualización como fue necesario durante la implementación del prototipo.
- Con el fin de minimizar los riesgos durante un fallo eléctrico, sería de ayuda añadir una UPS a la infraestructura.
- Hacer uso de un sistema de almacenamiento en la nube (AWS, GCP, Azure o DigitalOcean).
- Instalar un registro de Docker privado para guardar las imágenes.
- Implementar End to End testing.

Referencias Bibliográficas

- (2016a). How we architected and run kubernetes on openstack at scale at yahoo! japan. Recuperado de <https://kubernetes.io/blog/2016/10/kubernetes-and-openstack-at-yahoo-japan/>.
- (2016b). Kubernetes components. *Kubernetes*. Recuperado de <https://kubernetes.io/docs/concepts/overview/components/>.
- (2016). What is docker? *Red Hat*. Recuperado de <https://www.redhat.com/en/topics/containers/what-is-docker>.
- (2017). Kubernetes and openshift: Community, standards and certifications. *Fernandes, J.* Recuperado de <https://blog.openshift.com/kubernetes-and-openshift-community-standards-and-certifications/>.
- (2018). Architecture: Complex event processing. *Google Cloud Platform*. Recuperado de <https://cloud.google.com/solutions/architecture/complex-event-processing>.
- (2018). ?el mundo se está quedando sin potencia computacional?: ¿ qué es la ley de moore y por qué le preocupa al ceo de microsoft? *BBC Mundo*. Recuperado de <http://www.bbc.com/mundo/noticias-42803619>.
- (2018). Hosting wordpress on aws. *Amazon Web Services*. Recuperado de <https://github.com/aws-samples/aws-refarch-wordpress>.

(2018). Smart city 3.0: pregunte a barcelona por la siguiente generación de ciudades inteligentes. *URBAN HUB.* Recuperado de <http://www.urban-hub.com/es/cities/la-ciudad-de-barcelona-gana-en-inteligencia/>.

(2019). How ansible works. *Ansible.* Recuperado de <https://www.ansible.com/overview/how-ansible-works>.

(2019). Installing glusterfs - a quick start guide. *Gluster Documentation.* Recuperado de <https://docs.gluster.org/en/latest/Quick-Start-Guide/Quickstart/>.

(2019). Planning your installation. Recuperado de <https://docs.okd.io/3.11/install/index.html>.

(2019a). Understanding kubernetes objects. *Kubernetes.* Recuperado de <https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>.

(2019). Understanding microservices. *Red Hat.* Recuperado de <https://www.redhat.com/en/topics/microservices>.

(2019b). Volumes. Recuperado de <https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes>.

Altimore, P. (2018). Azure blockchain workbench architecture. *Microsoft Azure.* Recuperado de <https://docs.microsoft.com/en-us/azure/blockchain/workbench/architecture>.

Balaguera, F. y López, A. (2018). *Implementación de una nueva infraestructura de computación*

en la nube con modelo de alta disponibilidad basada en Openstack y contenedores para el grupo de investigación GID-CONUSS. Universidad Industrial de Santander.

Bansode, R. (2013). What is the difference between horizontal and vertical scaling? *esds*.

Benz, K. and Bohnert, T. (2013). Dependability modeling framework: A test procedure for high availability in cloud operating systems. *Vehicular Technology Conference*. Recuperado de <https://ieeexplore.ieee.org/document/6692157>.

Cholewa, T. (2018). 10 most important differences between openshift and kubernetes. Recuperado de <https://cloudowski.com/articles/10-differences-between-openshift-and-kubernetes/>.

CONSULTING, J. V. (2012). ¿cómo elegir hipervisor para mi solución de virtualización? Recuperado de <https://www.jmgvirtualconsulting.com/servicios-virtualizacion/como-elegir-hipervisor-para-mi-solucion-de-virtualizacion/>.

Douglas, K., Sipiwe, C., and Sinyinda, M. (2017). Web server performance of apache and nginx: A systematic literature review. Recuperado de <https://pdfs.semanticscholar.org/3401/39cd741e234a39b0db7bbaa38aee7c5f944c.pdf>.

Eldridge, I. (2018). What is container orchestration? *New Relic*.

Gantenbein, H. and Neira, B. (2017). Securing paas deployments. *Microsoft Azure Documentation*. Recuperado de <https://docs.microsoft.com/en-us/azure/security/security-paas-deployments>.

Heidi, E. (2016). What is high availability. *DigitalOcean*. Recuperado de <https://www.digitalocean.com/community/tutorials/what-is-high-availability>.

Joy, A. (2015). Performance comparison between linux containers and virtual machines. *ICACEA*. Recuperado de <https://ieeexplore.ieee.org/document/7164727/>.

Kappagantula, S. (2019). Microservice architecture ? learn, build and deploy microservices. *Edu-reka*. Recuperado de <https://www.edureka.co/blog/microservice-architecture/>.

Kumarpalani, S. (2014). Setting up dns server on centos 7. Recuperado de <https://www.unixmen.com/setting-dns-server-centos-7/>.

López, S. (2016). Despliegue y monitorización de un clúster mesos. Master's thesis, Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València.

Machado, E. (2016). Instalación y configuración de kvm en centos 7. Recuperado de <https://eliasmachado20.wordpress.com/2016/04/04/instalacion-y-configuracion-de-kvm-en-centos-7/>.

Martínez, J. y Vásquez, C. (2017). *Administración de prototipo infraestructura de computación en la nube del grupo GID-CONUSS con énfasis en la implementación de nuevos módulos de Openstack*. Universidad Industrial de Santander.

Mauro, A. (2012). Consideration about storage architectures. Recuperado de <https://vinfrastructure.it/2012/07/consideration-about-storage-architectures/>.

Mell, P. and Grance, T. (2011). The nist definition of cloud computing. recommendations of the national institute of standards and technology. *National Institute of Standards and Technology*. Recuperado de <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>.

Moilanen, M. (2018). *Deploying an application using Docker and Kubernetes*. Oulu University of Applied Sciences.

Novkovic, G. (2017). Five characteristics of cloud computing. Recuperado de <https://www.controleng.com/articles/five-characteristics-of-cloud-computing/>.

Schuchmann, M. (2018). Designing a cloud architecture for an application with many users. Master's thesis, University of Jyväskylä.

Thompson, S. and Parthasarathy, S. (2006). Moore's law: the future of si microelectronics. *Materials Today*.

Valbuena, A. (2017). *Estudio de una alternativa de integración continua que soporte el desarrollo de una infraestructura ti de servicios de información del transporte publico de pasajeros*. Universidad Industrial de Santander.

Villanueva, J. (2015). Active-active vs active-passive high availability cluster. *Jscape*. Recuperado de <http://www.jspape.com/blog/active-active-vs-active-passive-high-availability-cluster>.

Yinug, F. (2015). What is moore's law and why is it so great? *Semiconductor Industry Association.* Recuperado de <https://www.semiconductors.org/what-is-moores-law-and-why-is-it-so-great/>.

Apéndices

Apéndice A. Dataset generado para pruebas

El dataset fue generado en usando mockaroo con la siguiente configuración:

The screenshot shows the mockaroo web interface for generating datasets. At the top, there's a green header bar with the mockaroo logo and a 'PRICING' button. Below the header, the main interface has three sections: 'Field Name', 'Type', and 'Options'. There are five fields defined: 'username' (Username, blank: 0%), 'password' (Password, blank: 0%), 'email' (Email Address, blank: 0%), 'name' (Full Name, blank: 0%), and 'is_admin' (Number, min: 0, max: 1, decimals: 0, blank: 0%). An 'Add another field' button is available. Below these settings, there are filters for '# Rows' (set to 1000), 'Format' (set to CSV), 'Line Ending' (set to Unix (LF)), and 'Include' (checkboxes for 'header' and 'BOM'). At the bottom, there are buttons for 'Download Data' (green), 'Preview', and 'More'.

Posteriormente, se exportaron 15 datasets en formato CSV que fueron concatenados debido a que mockaroo limita a 1000 la cantidad de filas en la versión gratuita.

El archivo se encuentra en el siguiente link: Plan de pruebas/MOCK_DATA.csv

Apéndice B. Archivo Inventory

```
1 [OSEv3:children]
2
3 masters
4
5 nodes
6
7 etcd
8
9 [OSEv3:vars]
10
11 ansible_ssh_user=root
12
13 openshift_deployment_type=origin
14
15 openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
16     'true', 'challenge': 'true', 'kind':
17     'HTPasswdPasswordIdentityProvider'}]
18
19 openshift_disable_check=memory_availability,disk_availability
20
21 [masters]
22
23 master1.local.cluster
24
25 [etcd]
26
27 master1.local.cluster
28
29 [nodes]
30
31 master1.local.cluster openshift_node_group_name='node-config-master-infra'
32
33 node1.local.cluster openshift_node_group_name='node-config-compute'
34
35 node2.local.cluster openshift_node_group_name='node-config-compute'
```

Apéndice C. Endpoints glusterfs

```
1 apiVersion: v1
2
3 kind: Endpoints
4
5 metadata:
6
7 name: glusterfs-cluster
8
9 subsets:
10
11 - addresses:
12
13 - ip: 192.168.0.133
14
15 ports:
16
17 - port: 1
18
19 - addresses:
20
21 - ip: 192.168.0.134
22
23 ports:
24
25 - port: 1
```

Apéndice D. Archivos de despliegue

En el repositorio thesis-files hay una carpeta llamada 'Despliegue de artefactos' la cual contiene los archivos utilizados para el despliegue de los diferentes artefactos de la infraestructura siguiendo las convenciones planteadas en la metodología.

Repositorio: <https://github.com/JoseDRojasA/thesis-files>.

Apéndice E. Manual de instalacion

Una vez concluido el proyecto se realizó un manual de instalación para facilitar el despliegue de la infraestructura propuesta.

El manual de instalación se encuentra disponible en la carpeta raíz del repositorio.

Repositorio: <https://github.com/JoseDRojasA/thesis-files>

Apéndice F. Archivos de configuración de despliegue continuo

Archivo de despliegue continuo para el front: thesis-files/master/Configuración de CD/front/.gitlab-ci.yml.

Archivo de despliegue continuo para el backend: thesis-files/master/Configuración de CD/backend/.gitlab-ci.yml.

Repositorio: <https://github.com/JoseDRojasA/thesis-files>