MODELO PARA LA ORQUESTACIÓN DE MICROSERVICIOS CON KUBERNETES APLICADO AL SERVICIO DE CONTROL DE VERSIONES GIT

LEONARDO DELGADO PEDRAZA 20172678030
OSCAR DAVID PINEDA PARRA 20172678029

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD TECNOLÓGICA
INGENIERÍA TELEMÁTICA
BOGOTA D.C.
2019

MODELO PARA LA ORQUESTACIÓN DE MICROSERVICIOS CON KUBERNETES APLICADO AL SERVICIO DE CONTROL DE VERSIONES GIT

LEONARDO DELGADO PEDRAZA 20172678030
OSCAR DAVID PINEDA PARRA 20172678029

TRABAJO DE GRADO MONOGRAFÍA PARA OPTAR POR EL TÍTULO DE INGENIERO EN TELEMÁTICA

TUTOR(A):
JAIRO HERNÁNDEZ GUTIÉRREZ
INGENIERO DE SISTEMAS

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD TECNOLÓGICA
INGENIERÍA TELEMÁTICA
BOGOTA D.C.
2019

Nota de aceptación: .

_____
_____
_____
_____
_____
_____

_____

Tutor

_____

Jurado

Bogotá D.C. 06 de agosto de 2019

AGRADECIMIENTOS

En primera instancia agradecemos a Dios por permitirnos desarrollar nuestro proyecto de forma adecuada, por darnos sabiduría, empeño y determinación para afrontar las adversidades y problemáticas que se presentaron en el camino.

En segundo lugar, y no menos importante, agradecer a nuestros padres por brindarnos el apoyo moral, anímico y económico para poder llevar este proyecto a feliz término.

A nuestros docentes quienes nos han formado académica, conceptual y personalmente para contribuir al buen aprendizaje de las temáticas planteadas desde el inicio de nuestras carreras, por fortalecer una a una las cualidades cognitivas y a enfocarnos en un perfil adecuado con el fin de ampliar la visión de los campos y aplicaciones que se tienen desde el área de sistemas.

Finalmente agradecer a nuestros compañeros los cuales han sido un apoyo incondicional en la distribución del conocimiento que ha servido para mejorar cada día más en la elaboración de este proyecto.

CONTENIDO

Pág.

INTRODUCCIÓN 12
RESUMEN 13
ABSTRACT 14
1. DEFINICIÓN, PLANEACIÓN Y ORGANIZACIÓN 15
1.1. INTRODUCCIÓN 15
1.2. TÍTULO 15
1.3. PLANTEAMIENTO DEL PROBLEMA 15
1.3.1. Descripción del problema 15
1.3.2. Formulación del problema 16
1.4. OBJETIVOS 17
1.4.1. Objetivo General 17
1.4.2. Objetivos Específicos 17
1.5. JUSTIFICACIÓN 17
1.6. ALCANCES Y LIMITACIONES 18
1.6.1. Alcances 18
1.6.2. Limitaciones 18
1.7. MARCO REFERENCIAL 19
1.7.1. Fuentes Primarias 19
1.7.1.1. Auto-scaling web applications in hybrid cloud based on docker ..19
1.7.1.2. Emergency communication system with docker containers, osm and rsync 21
1.7.1.3. Adaptive application scheduling under interference in kubernetes 23
1.7.2. Fuentes Secundarias 26
1.7.2.1. Docker swarm con docker machine, alta-disponibilidad 26
1.7.2.2. Networking in containers and container clusters 27
1.7.3. Proyectos Relacionados 29

1.7.3.1. Diseño e implementación del sistema de gestión de entornos para la oficina asesora de sistemas de la universidad distrital 29
1.7.3.2. Github 30
1.8. MARCO TEÓRICO 31
1.9. METODOLOGÍA DEL PROYECTO 33
1.9.1. Scrum 33
1.9.2. KanBan 34
1.9.3. SrumBan 36
1.10. MARCO CONCEPTUAL 36
1.11. FACTIBILIDAD DEL PROYECTO 37
1.11.1. Factibilidad técnica o tecnológica 37
1.11.2. Factibilidad Operativa 38
1.11.3. Factibilidad Legal 38
1.11.4. Factibilidad Económica 40
2. ANÁLISIS DE MODELOS DE ORQUESTACIÓN ACTUALES 43
2.1. INTRODUCCIÓN 43
2.2. DOCKER SWARM 43
2.3. APACHE MESOS 49
2.4. KUBERNETES 52
3. DEFINICIÓN DEL MODELO BASADO EN KUBERNETES 59
3.1. INTRODUCCIÓN 59
3.2. Explicación del modelo 61
4. MICROSERVICIO COMO CASO DE ESTUDIO 68
4.1. INTRODUCCIÓN 68
4.2. HERRAMIENTAS UTILIZADAS 68
4.2.1. Redis 68
4.2.2. PostgreSQL 69
4.2.3. Prometheus y Grafana 70
4.2.4. GitLab 71
4.2.4.1. Escalado y alta disponibilidad 72

4.2.5. Resiliencia 73
4.2.6. HPA – Horizontal Pod Autoscaler 74
5. PRUEBAS DE STRESS 76
5.1. INTRODUCCIÓN 76
5.2. Ejecución de las pruebas 76
5.3. Conclusiones de las pruebas 79
CONCLUSIONES 82
RECOMENDACIONES 83
BIBLIOGRAFÍA 84
ANEXO 1 - Instalación CoreOS 87
ANEXO 2 - Instalación de ETCD 89
ANEXO 3 - Configuración de la red - Masters 96
ANEXO 4 - Generación de certificados SSL 102
ANEXO 5 - Configuración de Kubernetes 108
ANEXO 6 - Habilitación de RBAC 122
ANEXO 7 - Caso de Estudio 126
ANEXO 8 - Pruebas de Stress 136

LISTA DE TABLAS

Nombre	Pág.
Tabla 1. Costos del hardware utilizado en el desarrollo del proyecto	40
Tabla 2. Costos del software	41
Tabla 3. Costos recursos humanos del proyecto	41
Tabla 4. Otros gastos del proyecto	42
Tabla 5. Total, de costos del proyecto	42
Tabla 6. Explicación de los elementos en el diagrama de componentes	61

LISTA DE FIGURAS

Nombre	Pág.
Figura 1. Arquitectura del sistema web	20
Figura 2. Arquitectura del sistema de emergencia	22
Figura 3. Esquema de implementación en arquitectura de nube	23
Figura 4: Comportamiento de un Pod	25
Figura 5. Balanceo de Carga con docker Swarm	26
Figura 6: Diagrama de un service IP en Kubernetes	28
Figura 7. Arquitectura de Docker	30
Figura 8. Flujo de un Sprint en Scrum	35
Figura 9. Tablero de listas en trello	35
Figura 10. Funcionamiento de los nodos de Docker Swarm	44
Figura 11. Funcionamiento de los servicios de Docker Swarm	46
Figura 12. Esquema de Seguridad en Docker Swarm	47
Figura 13. Arquitectura de Apache Mesos	50
Figura 14. Oferta de recursos Apache Mesos	51
Figura 15. Componentes de Kubernetes	54
Figura 16. Funcionamiento de un pod	56
Figura 17. Modelo general de componentes	60
Figura 18. Arquitectura del cluster físico	67
Figura 19. Servicio web de prometheus Status>Targets	70
Figura 20. Dashboard de grafana	71
Figura 21. Arquitectura de Gitlab para el caso de estudio	73
Figura 22. Configuración de elementos de Jmeter	75

Figura 23. Componentes de Jmeter 76

Figura 24. Configuración de Firefox 77

Figura 25. Estado inicial statefulset 78

Figura 26. Escalamiento automático 79

Figura 27. Porcentaje de respuesta del servicio 80

LISTA DE ANEXOS

Nombre	Pág.
Anexo 1 - Instalación de CoreOS	85
Anexo 2 - Cluster de etcd	87
Anexo 3 - Configuración de flannel	94
Anexo 4 - Generación de certificados	100
Anexo 5 - Configuración de kubernetes	106
Anexo 6 - Habilitación de RBAC	121
Anexo 7 - Caso de estudio	122
Anexo 8 - Pruebas de Stress	132

INTRODUCCIÓN

Con el rápido desarrollo de internet y de las aplicaciones web cuyos componentes son colaborativos, han surgido nuevas necesidades por parte de los clientes para el consumo de servicios. La mayoría de aplicaciones corporativas o de uso personal, están limitadas por los recursos de hardware que soportan la demanda requerida, partiendo de una arquitectura cliente-servidor. Ahora bien, en el crecimiento exponencial del uso de estos sistemas, predominan los recursos sobrevalorados o infravalorados, lo cual causa que la relación costo-beneficio no se calcule bajo demanda de usuarios, por lo tanto, las compañías rara vez pagan por el consumo real de sus aplicaciones web.

En este documento se muestra un modelo para la orquestación de microservicios con Kubernetes, usando como ejemplo una plataforma de control de versiones de código abierto con el fin de plantearlo en un contexto colaborativo, generando así un repositorio privado para la Universidad Distrital Francisco José de Caldas, en donde los estudiantes puedan compartir las versiones de software que se van desarrollando tanto en proyectos de clase como en proyectos de grado. No obstante, el proyecto se enfoca en la configuración, diseño y modelamiento de una arquitectura autoescalable para un aplicativo en específico.

Finalmente se muestran los resultados obtenidos a partir de un laboratorio planteado en donde se indiquen datos estadísticos de la respuesta de la aplicación ante una demanda incremental de usuarios, en distintos módulos de aplicación. Razón por la cual se demuestra que la arquitectura planteada es viable para la orquestación de servicios.

RESUMEN

El proyecto modelo para la orquestación de microservicios con kubernetes aplicado al servicio de control de versiones GIT, es una recopilación de configuraciones en la administración de servicios, aplicadas a un modelo general, que reúne varias características para el funcionamiento adecuado en la prestación de servicios.

Dentro de este documento, el lector puede tener una visión general del esquema que se plantea, pero también observa técnicamente los pasos de configuración que se siguieron para la aplicación del ejercicio, con el fin de brindar una apreciación transversal de todos los elementos del sistema.

La arquitectura planteada es basada en kubernetes, tecnología fundada por Google en el año 2014 y posteriormente donada a la Cloud Native Computing Foundation. En estos cinco años de vida, kubernetes ha logrado posicionarse fuertemente en el mercado debido a su versatilidad para la administración de microservicios y en cualquier tipo de infraestructura; tanto así que compañías reconocidas como Spotify, Pokemon Go, Huawei, IBM, eBay, Nokia, SAP y Bose son un ejemplo de casos de éxito por la aplicación de este tipo de arquitectura.

Para poder implementar kubernetes en este proyecto fue necesario aplicar un cambio de mentalidad, debido a que absolutamente todos sus componentes están orientados a trabajar de forma distribuida, volátil e inmutable, lo que abre cabida a nuevos paradigmas involucrando aspectos de red, seguridad y administración centralizada y apropiada.

En resumen, en este proyecto se realizaron configuraciones de 4 redes para su funcionamiento, se aplicaron tecnologías de microservicios como Docker y Kubernetes, se implementaron componentes de almacenamiento como etcd (con cifrado en reposo), postgres y Redis; configuración de cifrado a través criptografía asímétrica, alojamiento de Gitlab, aplicación escrita en Ruby, Javascript, CSS y html, Configuración de DNS y Configuración de un Servidor Web para acceder a los servicios dentro del cluster.

ABSTRACT

The model project for the orchestration of microservices with kubernetes applied to the git version control service, is a compilation of configurations in the administration of services, applied to a general model, which brings together several characteristics for the proper functioning in the provision of services.

Within this document, the reader will be able to have an overview of the scheme that is proposed, but also can technically see the configuration steps that were followed for the application of the exercise, in order to provide a cross-sectional assessment of all the elements of the system.

The proposed architecture is based on kubernetes, technology founded by Google in 2014 and subsequently donated to the Cloud Native Computing Foundation (CNCF). In these five years of life, kubernetes has managed to position itself strongly in the market due to its versatility for the administration of microservices and in any type of infrastructure; so much so that recognized companies such as Spotify, Pokemon Go, Huawei, IBM, eBay, Nokia, SAP and Bose are an example of success stories by the application of this type of architecture.

To be able to implement kubernetes in this project it was necessary to apply a change of mentality, because absolutely all its components are oriented to work in a distributed, volatile and immutable way, which opens room for new paradigms involving aspects of network, security and centralized administration and appropriate.

In summary, in this project 4 network configurations were made for its operation, microservice technologies such as Docker and Kubernetes were applied, storage components such as etcd (with idle encryption), postgres and Redis were implemented; Encryption configuration through asymmetric cryptography, Gitlab hosting, application written in Ruby, Javascript, CSS and HTML, DNS Configuration and Web Server Configuration to access services within the cluster.

1. DEFINICIÓN, PLANEACIÓN Y ORGANIZACIÓN

1.1. INTRODUCCIÓN

En el presente capítulo, se brinda una contextualización al lector, respecto a la razón de la creación de este proyecto, se definen las condiciones necesarias para poder tratar los temas que se piensan tratar para el desarrollo sistemático de la monografía en general. Finalmente, el lector podrá comprender cuál es el problema raíz expuesto en este documento, y el enfoque que se implantará para desarrollar una solución que aplique al contexto definido.

1.2. TÍTULO

MODELO PARA LA ORQUESTACIÓN DE MICROSERVICIOS CON KUBERNETES APLICADO AL SERVICIO DE CONTROL DE VERSIONES GIT

1.3. PLANTEAMIENTO DEL PROBLEMA

1.3.1. Descripción del problema

El software de control de versiones (GIT), se ha convertido en una herramienta fundamental en el ámbito laboral, con el propósito de llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos. En la actualidad encontramos que muy pocos estudiantes universitarios se encuentran familiarizados con la importancia y los beneficios que trae contar con un repositorio con control de versiones, esto permite que exista la necesidad de involucrar a los estudiantes con estas nuevas herramientas, que hacen parte de las buenas prácticas profesionales y con las cuales se van a enfrentar en el campo laboral.

Actualmente no existe un sistema para el almacenamiento, gestión, versionamiento y control de código que pertenezca completamente a la Universidad Distrital, con el cual se pueda culturizar a los estudiantes y docentes a llevar un histórico del desarrollo de sus proyectos de materias y de grado. Por otra parte, no se cuenta con una infraestructura que sea autoescalable y orientada a servicios, que pueda crecer de acuerdo a la demanda de usuarios para este caso de estudio específicamente.

Pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de una aplicación cuando ésta tiene un gran número de archivos de código fuente, Linus Torvalds desarrolló un software para el control de versiones. Pero ¿Qué es de un software sin una buena infraestructura que lo soporte?, una infraestructura

tecnológica de alta disponibilidad en la actualidad es de suma importancia, debido a la problemática que se presenta en los sitios web, ya que estos demandan de mucha concurrencia o visitas de los usuarios.

1.3.2. Formulación del problema

En este sentido, el problema principal es que no existe la cultura de programación colaborativa haciendo uso de repositorios y de plataformas para el control y versionamiento de código. Por otra parte, si esta forma de construcción de proyectos no ha sido implementada, es simplemente por desconocimiento; es por esto que la falla principal radica en que la Universidad Distrital no cuenta con un software que brinde confiabilidad, disponibilidad y autoescalabilidad que permita que los docentes crean fielmente en esta metodología de trabajo, no obstante, la limitación más grande que se tiene con la implementación de una plataforma independiente es la infraestructura física que esto conlleva, para que soporte la demanda de todos los usuarios de la universidad.

En contraste, no es muy asequible que la universidad disponga de los recursos económicos para el uso de repositorios privados, ya disponibles en el mercado, debido a que la mayoría de ellos cobran de acuerdo al número de repositorios creados. Es por esto, que una arquitectura para este tipo de solución requiere de una implementación en nube o en un Datacenter con la suficiente capacidad para soportar las operaciones de Lectura y escritura (IOPS), velocidad de procesamiento, capacidad de almacenamiento y ancho de banda necesario para la publicación de código de proyectos en un repositorio privado de la universidad. ¿Es posible construir una cultura de desarrollo colaborativo con una relación costo beneficio favorable para la institución y con una infraestructura lo suficientemente robusta que permita soportar alta disponibilidad en todos sus servicios, con el fin generar confianza en los docentes para alojar allí sus proyectos y los de sus estudiantes?

1.4. OBJETIVOS

1.4.1. Objetivo General

Desarrollar un modelo para el autoescalamiento de microservicios con Kubernetes aplicado a un servicio de versionamiento y control de código open source, que sea autoescalable conforme a la demanda de usuarios.

1.4.2. Objetivos Específicos

- Analizar y caracterizar los modelos de orquestación de microservicios actuales, explicando su funcionamiento
- Determinar el modelo de orquestamiento y arquitectura del sistema Kubernetes, teniendo en cuenta todos sus componentes
- Plantear la configuración requerida para que el sistema sea auto escalable horizontalmente, mejorando su rendimiento
- Utilizar la herramienta de control de versiones GIT, como caso de estudio, con el fin de evidenciar la resiliencia del orquestador de servicios.
- Realizar pruebas de Stress sobre un laboratorio a pequeña escala y mostrar los resultados de auto escalabilidad del sistema.

1.5. JUSTIFICACIÓN

La tecnología Kubernetes fue desarrollada por Google hace aproximadamente una década y se describe como un sistema de código abierto para la organización y gestión automáticas de aplicaciones en contenedores.

En la actualidad Kubernetes se define como un proyecto de código abierto cuyo manejo corresponde a la Cloud Native Computing Foundation (**CNCF**) y a la Fundación **Linux**.[1]

Si se analiza este proyecto desde una perspectiva académica, se podrá identificar que la implementación de arquitecturas tan robustas, las cuales son usadas por grandes de la tecnología como Linux, CNCF y Google, son de gran contribución para el desarrollo intelectual de la Universidad Distrital Francisco José de Caldas.

El hecho de implementar este tipo de herramientas tiene diferentes ventajas, entre ellas, la primera que resalta es el uso de herramientas de código abierto,

[1] Kubernetes. Nuevo Paradigma de innovación. ©2018 SUSE. Reservados todos los derechos. Disponible en internet: https://www.suse.com/es-es/solutions/kubernetes/

debido a que funciona de una manera acorde con la filosofía de la universidad, la cual impulsa el uso de software con estas características y que también nutre su buen nombre con acciones de este estilo.

En segundo lugar, el desarrollo intelectual y conceptual de este proyecto es bastante complejo, por lo cual forja bases para las futuras generaciones estudiantiles, lo que conlleva a que la institución se mantenga de cierta forma actualizada con las tecnologías más usadas del momento; independientemente de que Kubernetes se haya desarrollado hace más de una década no significa que lleve ese mismo tiempo implementándose.

Actualmente las aplicaciones de la universidad distrital no funcionan de esta forma. debido a que se sigue implementando el modelo de Cliente-Servidor, el cual está siempre limitado por los recursos de Hardware y esto no permite que se aproveche de la forma más eficiente posible.

Finalmente, desde un enfoque económico y financiero, la implementación de este proyecto admite la flexibilidad de herramientas existentes en el mercado para la configuración y diseño del modelo planteado. Esto significa que la universidad podría disminuir el costo en el montaje de la plataforma para el control de versiones de código de los estudiantes; Además este proyecto puede ser la puerta de acceso para la configuración de futuros servicios tecnológicos que se tengan contemplados o incluso mejorar el rendimiento de los existentes.

1.6. ALCANCES Y LIMITACIONES

1.6.1. Alcances

- El presente modelo orientado a servicios se hará basado en la arquitectura maestro-esclavo, diseñado para estar débilmente acoplado, pero a su vez ser extensible, para que pueda soportar un gran flujo de trabajo en función del uso de CPU, proporcionando un autoescalado.

- El modelo abarca únicamente un ejemplo de implementación de una herramienta GIT, con el propósito de llevar registro de los cambios, mostrar los resultados y beneficios de una arquitectura orientada a servicios.

1.6.2. Limitaciones

- Este documento se encuentra dirigido a personal profesional y técnico, principalmente a los administradores IT encargados de gestionar los sistemas informáticos, bases de datos y redes, responsables de mantener los sistemas

en funcionamiento, presentando los beneficios de una infraestructura autoescalable, resiliente y con un balanceo de carga.

- Este proyecto cumplirá con el planteamiento del modelo, teniendo en cuenta que su implementación y puesta en marcha queda a criterio de la universidad, debido a que genera costos y cambios estructurales en la organización y en su infraestructura.

1.7. MARCO REFERENCIAL

1.7.1. Fuentes Primarias

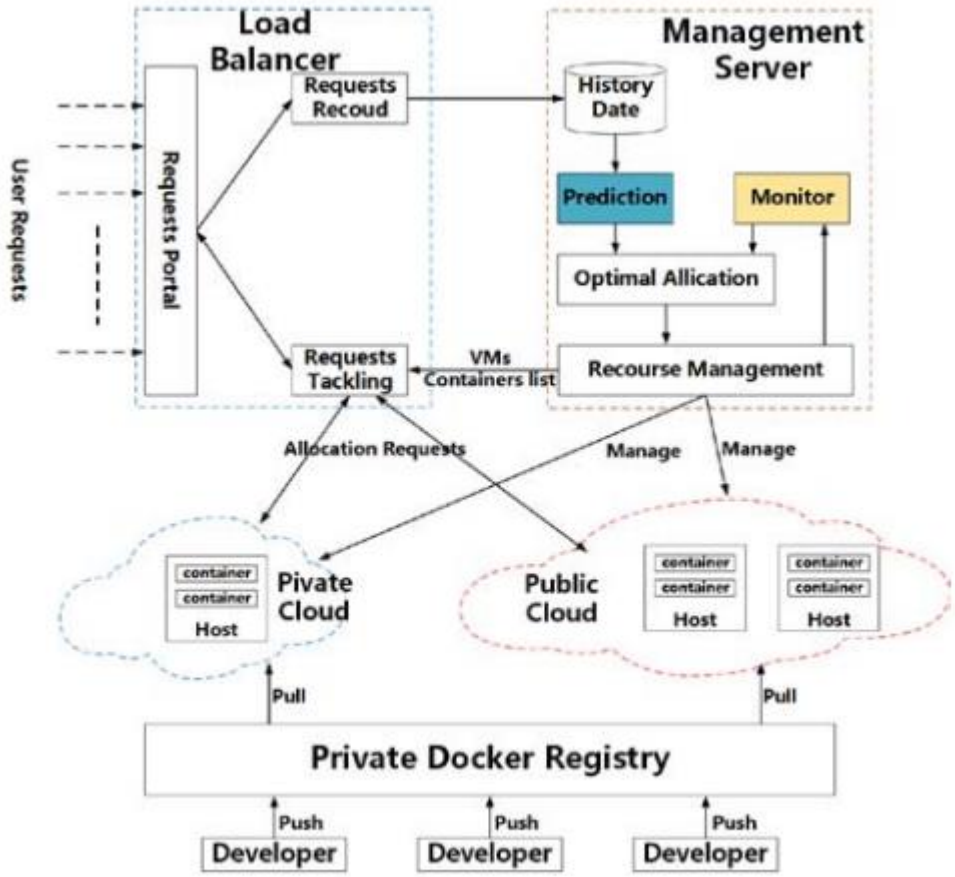
1.7.1.1. Auto-scaling web applications in hybrid cloud based on docker

Sino-German Joint Software Institute, School of Computer Science and Engineering. by Yunchun Li and Yumeng Xia (2016)

Este artículo trata de la auto escalabilidad de las aplicaciones ejecutadas en Docker mediante un el diseño de un algoritmo que se compone de tres componentes esenciales (Modelo Predictivo, Modelo Reactivo y Algoritmo de escalamiento)

En el primer tramo del documento, el autor plantea las ventajas de usar Docker en la automatización de despliegues, indica la posibilidad de realizarse en nubes públicas, privadas o híbridas, argumentando que este tipo de tecnología es compatible con cualquiera de ellas debido a que funciona de forma independiente de los recursos de la máquina virtual en donde se ejecutan los contenedores, esto se da debido a que, la imagen de Docker trae inmersos todos los componentes que va a usar para su respectivo funcionamiento.

Figura 1. Arquitectura del sistema web



Fuente: Auto-Scaling Web Applications in Hybrid Cloud Based on Docker[2]

Como se observa, la petición del usuario llega y es atendida por un balanceador de carga, lo que significa que dependiendo de las circunstancias redirigirá la petición a la nube pública o a la privada, las cuales alimentan sus contenedores de Docker con las imágenes alojadas repositorio privado, donde el desarrollador efectúa los cambios implementados.

El servidor Manager, principalmente cumple con las siguientes funciones:

1. Recolecta el número de peticiones para cada periodo de tiempo como histórico de datos.

[2] LI Yunchun, Xia Yumeng - Auto-Scaling Web Applications in Hybrid Cloud Based on Docker. Sino-German Joint Software Institute, School of Computer Science and Engineering, Beihang University, Beijing, China (2016)

2. Analiza el histórico de datos y predice el número de peticiones, para los próximos periodos de tiempo.
3. Calcula los recursos requeridos para la aplicación web y el número de contenedores

Finalmente define el funcionamiento del algoritmo, basándose en los modelos predictivos (Predicción de número de peticiones de los usuarios y determina la cantidad de contenedores y hosts requiere) y los modelos Reactivos (Permite analizar el alojamiento de los recursos que se van a disponer en determinado momento).

1.7.1.2. Emergency communication system with docker containers, osm and rsync

Computer science and Engineering National Institute of Technology. By Shiva Kumar Pentyala (2017)

El artículo en mención fundamenta su analítica de tecnologías de virtualización sobre un contexto de comunicación entre usuarios al momento de ocurrir un desastre natural como huracanes, terremotos, incendios, inundaciones, entre otros.

En su inicio el autor hace un resumen general del funcionamiento de la tecnología Docker, expone las principales ventajas que tiene en relación con la virtualización tradicional, la cual es implementada sobre Clusters de máquinas virtuales que corren sobre un software que soporte este tipo de tareas como Vmware, Hyper-V, KVM, entre otros.

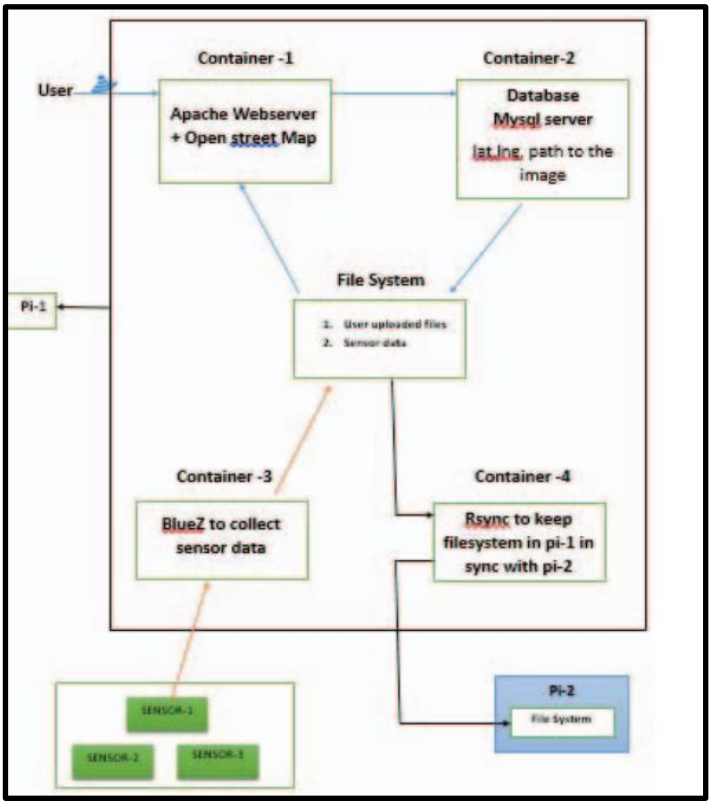
Luego del resumen introductorio elabora una explicación clara y concisa del modelo planteado. La justificación parte de una catástrofe natural, en donde un usuario con su teléfono celular puede enviar información de su ubicación a pequeños servidores (Raspberry PI) a través de bluetooth, con el fin de ser rescatados de situaciones de encierro o de dificultad.

Posteriormente, explica la solución completa que diseñó para este tipo de situaciones, indicando la integración de los contenedores de Docker con volúmenes de datos, los cuales almacenan dos tipos de información, la primera son todos los mensajes de texto que los usuarios finales envían, y la segunda se enfoca más en el lado de los servidores Raspberry PI y el OSM (Open Street Map).

En la Figura se observa el modelo planteado por el autor, en donde se explican detalladamente cada uno de los componentes de la aplicación. En primera instancia entra la petición del usuario en peligro, posteriormente el contenedor 1 se encarga de ofrecer la interfaz web y pintar un mapa mediante OSM, luego, el contendor 2 contiene la base de datos, que a su misma vez alimenta el filesystem ubicado en el centro del diagrama mediante Rsync.

En la parte de abajo, se observa el contenedor 3 el cual recolecta datos de los sensores, esta información también es enviada al filesystem y Finalmente el Rsync nuevamente se ejecuta para actualizar los 3 Raspberry Pi del modelo.

Figura 2. Arquitectura del sistema de emergencia



Fuente: EMERGENCY COMMUNICATION SYSTEM WITH DOCKER CONTAINERS[3]

[3] KUMAR, Shiva. EMERGENCY COMMUNICATION SYSTEM WITH DOCKER CONTAINERS, OSM AND RSYNC. Computer science and Engineering National Institute of Technology. Durgapur, India. 2017

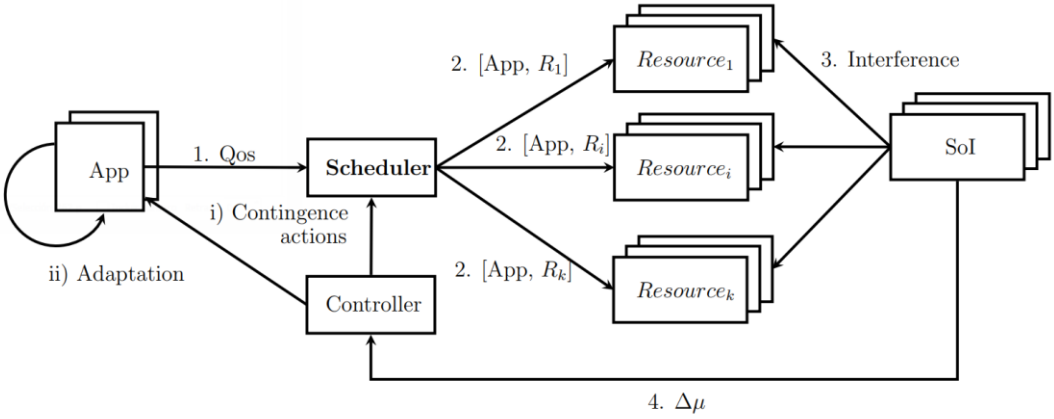
1.7.1.3. Adaptive application scheduling under interference in kubernetes

9th International Conference on Utility and Cloud Computing IEEE/ACM. By Víctor Medel, Omer Rana, José Ángel Bañares and Unai Arronategui (2016)

Dentro de este artículo se encuentra un modelo de recursos basado en la red gestionada por Kubernetes, principalmente para identificar mejor los problemas de rendimiento. Este modelo hace uso de datos obtenidos por micro-benchmarks, dentro de una implementación de Kubernetes, y puede ser utilizado como una base para un diseño escalable (y potencialmente tolerante a los fallos) para las aplicaciones alojadas dentro de Kubernetes.

El autor del artículo presenta un modelo de referencia basado en la gestión y el rendimiento para Kubernetes, identificando diferentes estados operacionales que pueden estar asociados a los "pods" y contenedores, teniendo en cuenta la competencia de estos para acceder a los recursos del sistema basado en el consumo de memoria, CPU, red, etc.

Figura 3. Esquema de implementación y adaptabilidad de una aplicación propuesta en una arquitectura de nube



Fuente: Adaptive Application Scheduling under Interference in Kubernetes[4].

La solicitud de recursos en ejecución se influye de manera diferente en función del tipo y la duración de la interferencia observada a lo largo de la vida de una aplicación. De esta manera, el desarrollador de la aplicación puede tener en

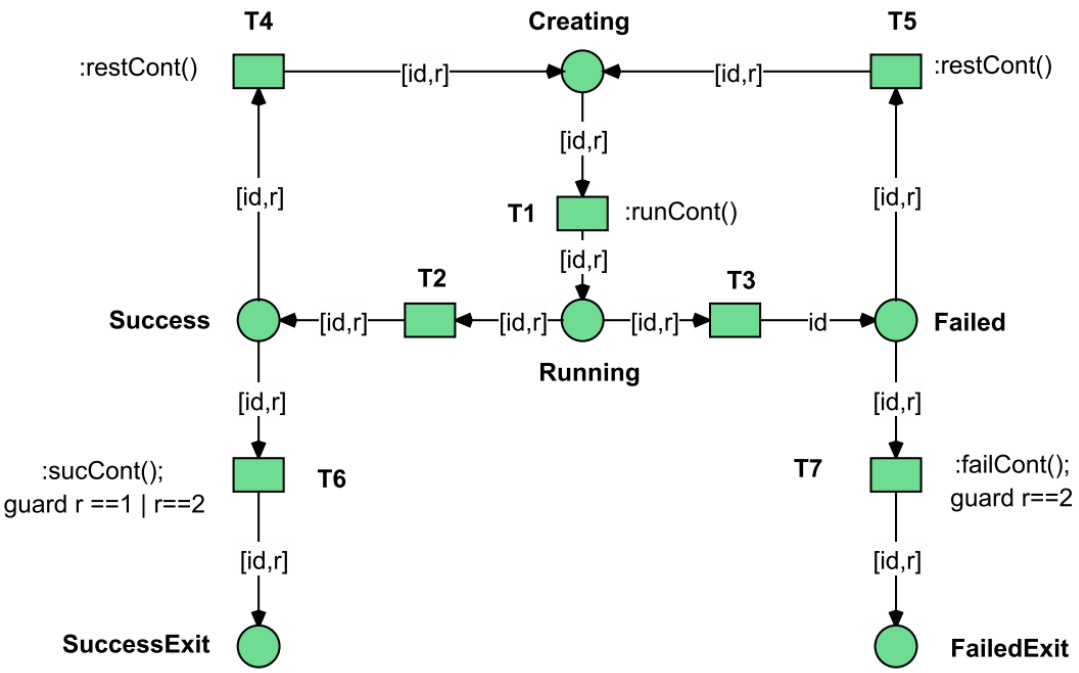
[4] Medel Víctor, Rana Omer, Bañares José Ángel and Arronategui Unai - Adaptive Application Scheduling under Interference in Kubernetes. 9th International Conference on Utility and Cloud Computing, IEEE/ACM (2016)

cuenta dicha interferencia como una fuente para mejorar el comportamiento de aplicación, generando diferentes modos de respuesta, aumentando la resiliencia a las fallas. Este enfoque está influenciado por el sistema Paragon[5], que utiliza un algoritmo de clasificación para determinar la influencia de varias fuentes de interferencia (SoI) en la ejecución de la aplicación. Con esta información, el programador intenta equilibrar las cargas de trabajo de la aplicación, dependiendo en las características de interferencia.

Una muestra del modelo, que representa el comportamiento de un Pod en Kubernetes a lo largo del ciclo de vida de sus contenedores Figura 5. Esta abstracción nos permite modelar sistemas jerárquicos. Sus distribuciones de probabilidad se obtienen midiendo un clúster real de Kubernetes. Por ejemplo, las transiciones T2 y T3 modelan el tiempo de ejecución de una aplicación y el tiempo hasta que falla el siguiente contenedor, respectivamente. Además, la configuración de implementación de la aplicación puede afectar este momento. Para modelar el comportamiento del contenedor, podemos modificar estas transiciones para representar la influencia causada por posibles fuentes de interferencia. Cada SoI se puede modelar con un lugar para el recurso (compartido para todos los contenedores) y un lugar que represente la sensibilidad a esa fuente para cada contenedor. La interferencia se modela como el tiempo de espera para tomar el recurso necesario. Para estimar estos valores, se deben usar algunas técnicas de micro-benchmarking.

[5] C. Delimitrou and C. Kozyrakis, “Paragon: Qos-aware scheduling for heterogeneous datacenters,” in ACM SIGPLAN Notices, vol. 48, no. 4. ACM, 2013, pp. 77–88.

Figura 4: Comportamiento de un Pod.



Fuente: Paragon: Qos-aware scheduling for heterogeneous datacenters

Las transiciones en negrita son transiciones cronometradas, relacionadas con el ciclo de vida de un pod. Sus distribuciones de probabilidad se obtienen midiendo un clúster real de Kubernetes. Por ejemplo, las transiciones T2 y T3 modelan el tiempo de ejecución de una aplicación y el tiempo hasta que falla el siguiente contenedor, respectivamente. Ambos dependen de la aplicación y pueden verse influenciados por la carga de trabajo de fondo. Podemos modificar estas transiciones para representar la influencia causada por posibles fuentes de interferencia. Cada SoI se puede modelar con un lugar para el recurso (compartido para todos los contenedores) y un lugar que represente la sensibilidad a esa fuente para cada contenedor. La interferencia se modela como el tiempo de espera para tomar el recurso necesario.

1.7.2. Fuentes Secundarias

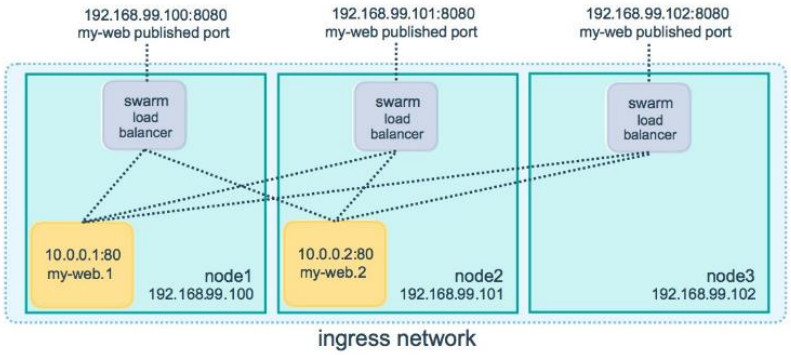
1.7.2.1. Docker swarm con docker machine, alta-disponibilidad[6]

Manuel Morejón. Ingeniero DevOps y consultor. Publicado el 30 de noviembre de 2017.

El documento expone algunas recomendaciones y ejemplos para brindar servicios de alta disponibilidad en ambientes productivos, basados en tecnología de contenedores y orquestación de servicios con docker swarm. El autor hace énfasis en exponer que en un ambiente productivo,el objetivo principal es brindar un servicio de forma continua y sin interrupciones, logrando así una capacidad de resiliencia automatizada, con el fin de recuperarse ante fallos tanto físicos como de componentes de software.

Posterior a ello, el documento muestra los pasos de configuración requeridos para brindar un cluster de Docker integrando alta disponibilidad. Muestra cómo generar un balanceo de carga inmerso en el orquestador Docker Swarm. El uso de la tecnología de Docker contiene una cantidad bastante elevada de información concerniente a ejemplos, configuraciones y especificaciones técnicas. En este documento se expone una guía práctica en donde se muestran las mejores prácticas para ambientes productivos, demostrando que ofrecer una alta disponibilidad basada en microservicios no es fácil de configurar, pero tampoco es imposible.

Figura 5. Balanceo de Carga con docker Swarm



Fuente: Docker Swarm con Docker Machine[7]

[6] M Manuel, Docker Swarm con Docker Machine, Alta-Disponibilidad, http://mmorejon.github.io/blog/ (2017)

[7] Ibid

El autor muestra la siguiente imagen en su documento, la cual muestra que el balanceador de carga está en la capacidad de redireccionar la petición del usuario al contenedor que tenga el servicio en ese momento. En este caso en específico el contenedor con dirección IP 10.0.0.3 se cayó y el Balanceador del nodo 3 (192.168.99.102) redirecciona las peticiones a los servicios de los nodos 1 y 2.

1.7.2.2. Networking in containers and container clusters

Proceedings of netdev, Google, Inc. Mountain View, CA USA. By Victor Marmol, Rohit Jnagal, and Tim Hockin (2015).[8]

Este artículo permite conocer el funcionamiento de la red de servicios que se crea dentro de kubernetes, se puede ver como los contenedores se están convirtiendo en una forma popular de implementar aplicaciones de forma rápida, económica y confiable.

Para entender el funcionamiento de la red del cluster, se debe observar primeramente las redes en contenedores, las cuales se centran principalmente en el uso de dos de los espacios de nombres del kernel de Linux: red y UTS (UNIX Timesharing System). Estos espacios de nombres permiten que un contenedor se presente al mundo y se en rute como si fuera un host. Docker se compone de varios subcomponentes que administran el tiempo de ejecución del contenedor, las capas del sistema de archivos y las imágenes de la aplicación. El aspecto de la red se maneja en la configuración en el tiempo de ejecución del contenedor.

Servicios en Kubernetes

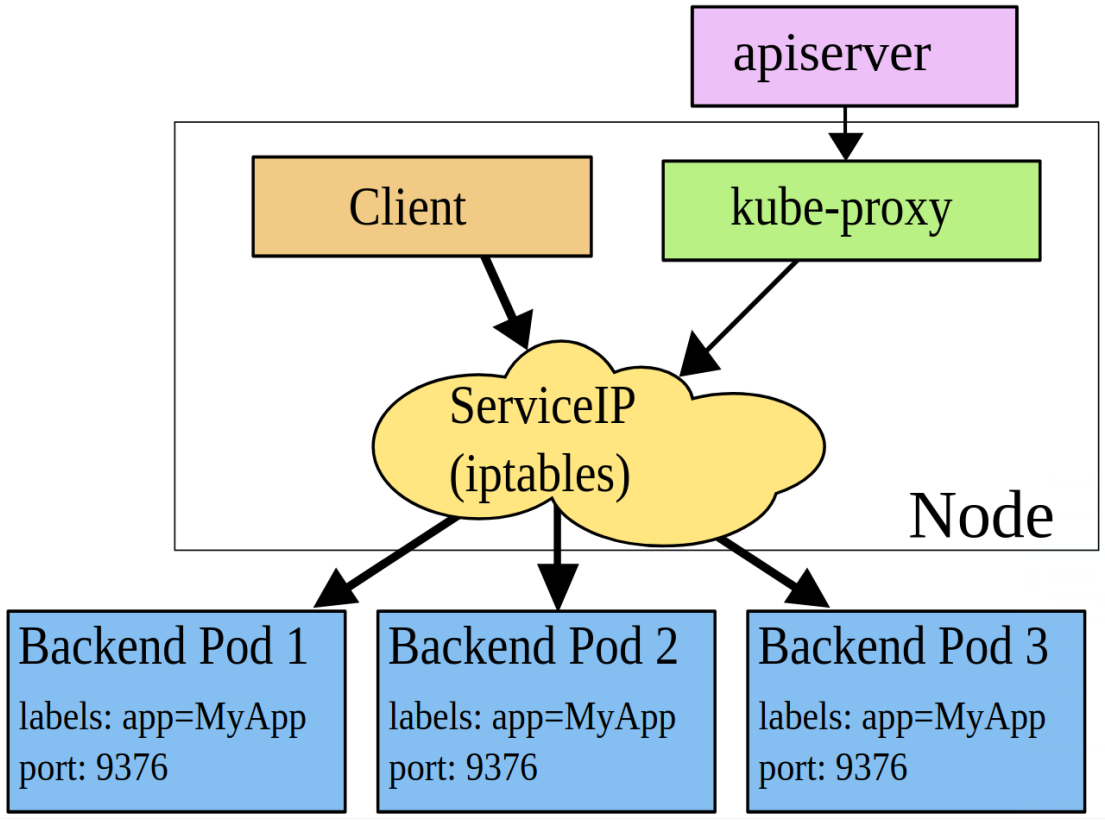
Los pods se consideran efímeros: pueden ir y venir. Una máquina en mantenimiento puede, por ejemplo, hacer que se reemplacen con diferentes pods que sirven con el mismo propósito. Esto hace que sea inconveniente e incorrecto abordar un pod por su dirección IP. Aquí es donde entran los servicios. Un servicio es una abstracción que permite el direccionamiento estable de un grupo de pods (a veces llamado microservicio)[9]. Funciona de manera muy similar a un grupo de pods colocados frente a un balanceador de carga. Un servicio tiene una IP que garantiza que sea estable. Las solicitudes a esa IP se equilibran de carga a las unidades activas que están detrás del servicio. A medida que van y vienen los pods, el servicio actualiza las rutas de cada uno de los pods disponibles, para

[8] Marmol Victor, Jnagal Rohit, and Hockin Tim Proceedings of netdev, Google, Inc. Mountain View, CA USA. By Victor Marmol, Rohit Jnagal, and Tim Hockin (2015).

[9] Kubernetes, “Services”, accessed February 6, 2014, https://github.com/GoogleCloudPlatform/kubernetes/blob/ master/docs/services.md

atender las solicitudes entrantes. Los servicios se implementan mediante una combinación de rutas de iptables y un proxy de servicio de espacio de usuario que se ejecuta en todos los nodos de Kubernetes.

Figura 6: Diagrama de un service IP en Kubernetes



Fuente: Service proxies[10]

Cuando un pod en un nodo realiza una solicitud a un servicio a través de la dirección IP de este último, las reglas de iptables redirige la solicitud al proxy de servicio en el nodo. El proxy de servicio mantiene una lista actualizada de todos los pods que pueden responder solicitudes para este servicio en particular. El proxy observa el estado del clúster compartido para ver los cambios de los miembros del servicio y puede representar el cambio rápidamente. Dada la lista de pods de miembros, el proxy de servicio realiza un balanceo de carga de round robin simple en el lado del cliente a través de los pods de miembros. Este proxy de servicio permite que las aplicaciones funcionen sin modificaciones en un clúster de Kubernetes. Los servicios en Kubernetes tienden a no estar expuestos al mundo

[10] Kubernetes.io, “service proxies”, accessed September 25, 2018 https://kubernetes.io/docs/concepts/services-networking/service/#proxy-mode-iptables

exterior ya que la mayoría de los microservicios simplemente hablan con otros microservicios en el clúster. Los servicios públicos también son extremadamente importantes, ya que algunos microservicios deben eventualmente proporcionar un servicio al mundo exterior. Desafortunadamente, los servicios públicos no se manejan completamente hoy en Kubernetes ya que no existe un proxy externo para actuar de manera similar al proxy del servicio en el nodo. La implementación actual de los servicios públicos tiene un equilibrador de carga que se dirige a cualquier nodo en un clúster de Kubernetes. Las solicitudes que llegan a este nodo son redireccionadas por el proxy del servicio a las solicitudes correctas de respuesta del pod para el servicio. Esta es un área activa de trabajo donde aún se está diseñando una solución más completa. Una alternativa a la abstracción del servicio de Kubernetes es usar el DNS para redirigir a los pods.

1.7.3. Proyectos Relacionados

1.7.3.1. Diseño e implementación del sistema de gestión de entornos para la oficina asesora de sistemas de la universidad distrital

Universidad Distrital Francisco José de Caldas, Facultad de Ingeniería. Por Manuel Fernando Muñoz Garcés. (2017)

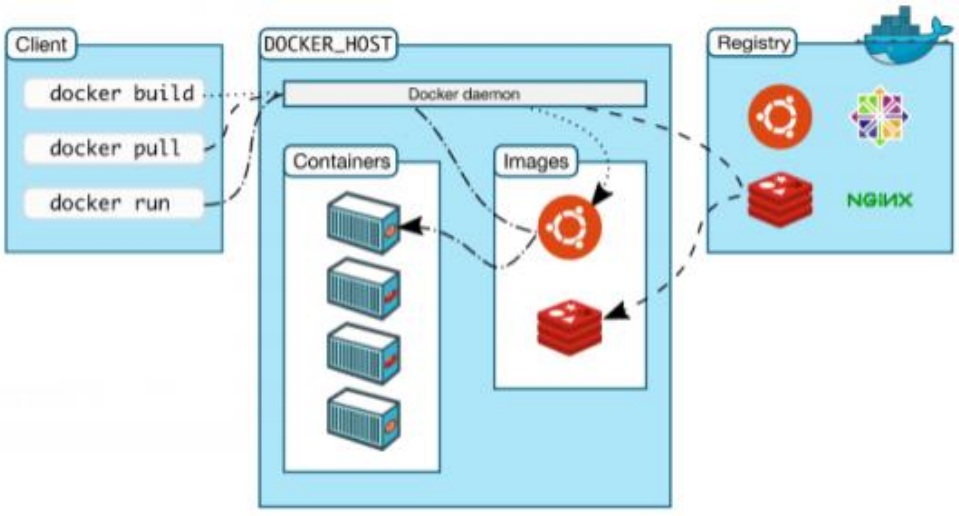
El objetivo de este proyecto es brindar las herramientas tecnológicas, para el mejoramiento y automatización de los procesos de despliegue de la OAS (Oficina de Asesorías de Sistemas), la cual es la encargada del desarrollo, mantenimiento y soporte del sistema de información de la Universidad Distrital.

El cuerpo del proyecto se focaliza en la explicación de la tecnología de Docker como CaaS (Container as a service), el cual permite replicar un microservicio cuantas veces sea necesario. Menciona también las diferencias entre un contenedor y una máquina virtual, haciendo énfasis en la utilización del núcleo del kernel compartido, lo cual permite que el “corazón del sistema operativo” administre los recursos tanto del servidor como de los contendores.

El proyecto se trabajó bajo una metodología Scrumban, puesto que no se trata de un proyecto de desarrollo de software sino de la implementación de una mejora en los procesos de administración de los entornos de TI para el sistema de información de la Universidad Distrital Francisco José de Caldas.

Por último, el autor muestra un gráfico del funcionamiento de la arquitectura de Docker en donde expone todos los componentes que hacen parte de este tipo de tecnología

Figura 7. Arquitectura de Docker



Fuente: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE GESTIÓN DE ENTORNOS PARA LA OFICINA ASESORA DE SISTEMAS DE LA UNIVERSIDAD DISTRITAL[11]

1.7.3.2. Github

GitHub es una plataforma de desarrollo inspirada en la forma en que trabajas. Desde el código abierto hasta el negocio, puede alojar y revisar el código, administrar proyectos y crear software junto con 28 millones de desarrolladores[12]

Este es el repositorio de control de versiones de código más reconocido a nivel mundial, tiene vínculo directo con empresas de renombre como SAP, IBM, Airbnb, Facebook, Spotify, PayPal, Node, Walmart, entre otras.

Su filosofía está basada en la implementación de código abierto con el fin de formar una comunidad de desarrolladores, que se vinculan a través de proyectos públicos o resolución de problemas comunes alojados en foros, manuales, documentación e implementación de diferentes servicios y arquitecturas.

---

[11] MUÑOZ Manuel, DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE GESTIÓN DE ENTORNOS PARA LA OFICINA ASESORA DE SISTEMAS DE LA UNIVERSIDAD DISTRITAL - Tesis de Grado - 2017

[12] GITHUB - Official Web Site - https://github.com/ - © 2018 GitHub, Inc

El equipo de trabajo detrás de este conocido proyecto es:

- Ariya Hidayat: Administrador de PhantomJS, una de las herramientas más populares, usadas por las empresas para la implementación de pruebas de integración.
- Russell Keith-Magee: Creador de BeeWare, proyecto que busca recopilar un conjunto de herramientas que faciliten la vida a los desarrolladores de Python. A diferencia de un IDE (entorno de desarrollo integrado) cada herramienta es independiente y se puede usar por separado.
- Kris Nova: Amante por el software de código abierto. Kris trabaja en herramientas open source en su trabajo diario, que incluye el mantenimiento de Kubernetes Operations (kops).
- Jess Frazelle: trabaja en Kubernetes tiempo completo. Anteriormente mantuvo Docker, una plataforma de contenedorización de software utilizada por miles de equipos.

Las ventajas de trabajar con Github, en primera instancia es el versionamiento de código, lo cual es de vital importancia en un proyecto de grandes magnitudes y de varios colaboradores.

Aprendizaje constante debido a que el trabajo en equipo ayuda a contribuir nuevas ideas para la resolución de problemas actuales, lo cual permite mantenerse informado y llevar de una forma ordenada un registro de incidencias respecto a un tema específico.

1.8. MARCO TEÓRICO

1.8.1. Virtualización

La virtualización es una tecnología que permite ofrecer servicios de TI, con recursos que están ligados al hardware. En términos más conocidos, la virtualización permite utilizar los recursos físicos de un servidor de una forma más eficiente con el fin de sectorizar los servicios allí alojados. Por ejemplo: se tienen tres servidores físicos, los cuales ofrecen servicios independientes. el primero es un servidor de logs, el segundo es un servidor DNS y el tercero ejecuta aplicaciones internas. Se utiliza cerca del 30% de la capacidad de cada servidor, es decir, solo una parte de su potencial de ejecución. Pero como las aplicaciones internas siguen siendo importantes para sus operaciones internas, tiene que conservarlas junto con el tercer servidor que las hospeda, Generalmente, es más fácil y confiable ejecutar procesos de forma independiente en cada uno de los servidores: un servidor, un sistema operativo y una tarea. No era sencillo asignar

múltiples tareas a un servidor. Pero la virtualización permite dividir el servidor de logs en dos servidores únicos que pueden administrar tareas independientes para que las aplicaciones internas y el servicio de DNS se puedan integrar en un solo sistema operativos. Se utiliza el mismo hardware, pero de manera más eficiente.[13]

Algunos tipos de virtualización son los siguientes:

- *Virtualización de los datos:*

La virtualización de datos permite a las empresas tratar a los datos como si fueran una cadena; de tal forma, que se obtiene mayor rendimiento en procesamiento que reúne los datos procedentes de varias fuentes.

- Virtualización de escritorios

La virtualización de escritorios provee una herramienta de administración para que un administrador cree entornos simulados de escritorio en muchas de máquinas físicas en un solo instante de tiempo.

- Virtualización del servidor

Funciona de la misma forma que los entornos virtuales, sin embargo, la diferencia es que los servidores procesan mayor cantidad de información por lo cual requieren de una gestión y automatización mucho más eficiente.

- Virtualización del sistema operativo

La virtualización del sistema operativo se ejecuta en el kernel, es decir, los encargados de gestionar las tareas centrales de los sistemas operativos. Es una forma práctica de lanzar los entornos Linux y Windows de manera simultánea.

- Virtualización de funciones de Red

La virtualización de las funciones de red sirve para replicar los componentes básicos de networking, como los servicios de DNS, el uso compartido de archivos y la configuración de direccionamiento para visualizarlas en los entornos.

1.8.2. Orquestación

[13] Zabaljáuregui M, Tesis de Maestría en Cómputo de Altas Prestaciones, Facultad de Informática Universidad Nacional de La Plata (2010) - Pág 25

En la actualidad, la orquestación de Servicios es de vital importancia en las aplicaciones web, debido a que permiten coordinar correctamente los componentes que integran las arquitecturas de software. La principal característica de este concepto es permitir la reutilización de servicios a una grande, mediana o pequeña escala.

La coreografía no depende de un orquestador central, debido a que cada servicio que interviene en el proceso tiene que conocer cuándo entrar en acción y cuándo estar inactivo. Desde el punto de vista de la ejecución de un proceso la orquestación se centra en el funcionamiento interno, mientras que la coreografía lo hace en la perspectiva externa, enfocándose en la interacción de este proceso.

Algunos de los estándares para la especificación detallada de la orquestación de los servicios son: Web Service Choreography Description Language (WS-CDL) y Web Service Choreography Interface (WSCI). Por otra parte, BPEL es una especificación de lenguaje basado en XML para la concepción de procesos de negocio a través de la interacción de servicios web y es utilizado para modelar el comportamiento tanto de procesos de negocio ejecutables como abstractos.[14]

1.9. METODOLOGÍA DEL PROYECTO

1.9.1. Scrum

En 1986 Takeuchi y Nonaka dieron a conocer una forma novedosa para la gestión de proyectos de forma ágil y flexible. Estas personas descubrieron que algunas empresas desarrollaban proyectos en un periodo de tiempo menor con respecto a otras. Partiendo de la filosofía de trabajo de compañías como HP, Honda, Canon y otras; se dieron cuenta que los proyectos partían de objetivos muy generales, es decir, se manejaban los procesos desde un inicio hasta un final, obviando las fases intermedias en el flujo normal del proyecto.

Años más tarde Jeff Sutherland y Ken schwaber postularon de una manera más formal y estructurada los pasos para el desarrollo de software. Esta metodología sirve para proyectos con las siguientes características:

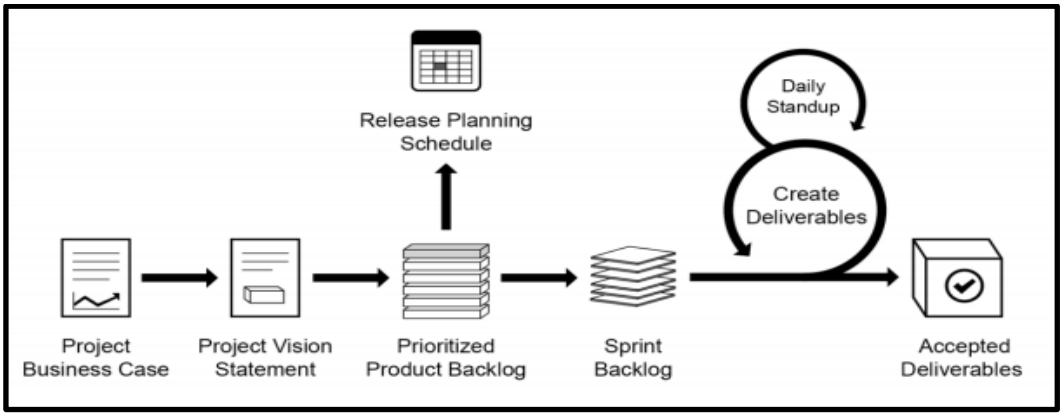
- Incertidumbre: Generalmente en los proyectos se define un objetivo que se quiere proporcionar sin establecer un plan que especifique el detalle del producto o servicio.

[14] CASTRO Claudia, Flóres Eva, Capacidad de Orquestación de Servicios Web en las Herramientas MULE ESB y Oracle Service Bus - Tesis de Grado - Lámpsakos julio-diciembre 2015 Medellín–Colombia - 2015

- Auto-organización: Esto indica que los equipos que desarrollan los proyectos están en la capacidad de autogestionarse, no obstante, se tienen que tener características de tipo autónomo, auto-superación y auto-enriquecimiento.
- Control modelado: Se va a tener un seguimiento constante para garantizar que se mantenga un orden y una priorización de las tareas. Trata de equilibrar la cantidad de trabajo entre los miembros del equipo con el fin de fortalecer la creatividad y espontaneidad de todos los integrantes
- Transmisión del conocimiento: La filosofía de Scrumb es “Todo el mundo aprende de todo el mundo”. Generalmente, dentro de las organizaciones, las personas no se enfocan en un solo proyecto, lo cual permite una retroalimentación constante entre las diferentes problemáticas, conocimientos y avances que se tengan en los diferentes proyectos.

Scrum al ser una metodología de desarrollo ágil tiene como pilar la creación de de ciclos cortos para la ejecución del proyecto, estas iteraciones son conocidas como Sprints. Estos microciclos se pueden expresar gráficamente de la siguiente forma

Figura 8. Flujo de un Sprint en Scrum



Fuente: A Guide to the SCRUM BODY OF KNOWLEDGE[15]

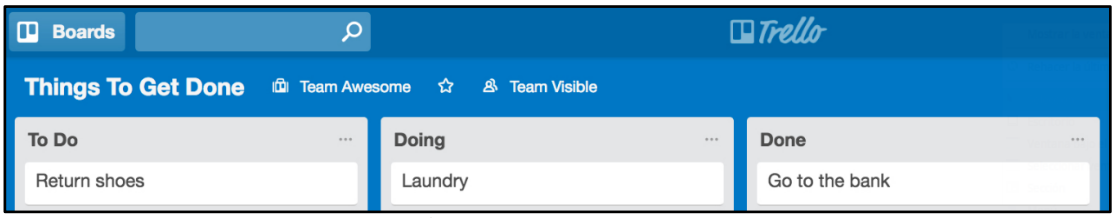
1.9.2. KanBan

Proveniente de Japón, Kanban es un símbolo visual que se utiliza para desencadenar una acción. A menudo se representa en un tablero Kanban para reflejar los procesos de su flujo de trabajo.

[15] SCRUMstudy™, a brand of VMEdu, Inc. A Guide to the SCRUM BODY OF KNOWLEDGE (SBOK™ Guide). (2016)

Kanban es un tablero de trabajo muy popular a la hora de implementar un desarrollo de software ágil. Requiere una comunicación en tiempo real sobre la capacidad y transparencia del trabajo total. Los elementos de trabajo se representan visualmente en un tablero kanban, lo que permite a los miembros del equipo ver el estado de cada uno en cualquier momento.

Figura 9. Tablero de listas en trello



Fuente: Official Web Site Trello[16]

¿Cómo funciona Kanban?

Los principios básicos de kanban permiten obtener el máximo rendimiento de su flujo de trabajo.

Visualice lo que hace (su flujo de trabajo): una visualización de todas sus tareas y elementos en una tabla contribuirá a que todos los miembros de su equipo se mantengan al corriente con su trabajo.

Limite la cantidad de Trabajo en Proceso (límites del TEP): establezca metas asequibles. Mantenga el equilibrio de su flujo de trabajo mediante la limitación de los trabajos en proceso para prevenir el exceso de compromiso en la cantidad de tareas que será incapaz de terminar.

Realice un seguimiento del tiempo: Realice un seguimiento de su tiempo de forma continua y evalúe su trabajo con precisión.

Lectura fácil de indicadores visuales: conozca lo que está ocurriendo de un solo vistazo. Utilice tarjetas de colores para distinguir los Tipos de trabajo, Prioridades, Etiquetas, Fechas límite y más.

Identifique los cuellos de botella y elimine lo que resulta descartable: aproveche al máximo los plazos y ciclos de ejecución, del Flujo Acumulativo y de los informes de tiempo. Estos criterios le permitirán evaluar su rendimiento, detectar los problemas y ajustar el flujo de trabajo en consecuencia.

[16] Tableros Básicos Trello, Official Web Site https://trello.com/guide/board-basics.html Atlassian

1.9.3. SrumBan

La metodología Scrumban nace de la combinación de principios de los métodos ágiles de gestión de proyectos más importantes en la actualidad: Scrum y Kanban. Aunque en principio pueden parecer iguales, las dos estrategias de gestión presentan diferencias en la manera de ejecutar el proyecto. Es por eso que el novedoso plan Scrumban se encarga de combinar aquellos elementos que resultan complementarios.

Scrumban es emplear lo mejor de Scrum y Kanban. Es decir, aquello que al combinarlo sirva para mejorar la productividad de un plan de empresa. En el terreno de la ejecución, el objetivo de Scrumban es llevar a cabo una secuencia de mejoras evolutivas a través de prácticas eficientes. El flujo de trabajo sigue siendo el mismo de Kanban (etapas relacionadas entre sí), aunque con la inclusión de algunos elementos de Scrum como las reuniones diarias de 15 minutos entre el grupo de trabajo y el gestor o los análisis retrospectivos para incorporar mejoras al proceso. Por ello, existen ciertos proyectos que se adecúan mejor al método mixto. En general, se trata de aquellos con un mayor nivel de complejidad. Algunos ejemplos son:

Proyectos de mantenimiento: aquellos en los que resulta indispensable la presentación de resultados de forma parcial para seguir avanzando.

Proyectos en los que los requisitos varían con frecuencia: aquellos en los que el cliente no tiene fijadas las condiciones y expectativas del proyecto y éstas se van introduciendo a lo largo de las distintas etapas.

Proyectos en los que surjan errores de ejecución: aquellos en los que se deba replantear el método usado y analizar retrospectivamente la evolución de las tareas.

1.10. MARCO CONCEPTUAL

- **Microservicio:** Hace referencia a un servicio específico dentro de un sistema.
- **Contenedor:** Permite empaquetar un microservicio, para que pueda funcionar en cualquier servidor virtual, independientemente de los prerrequisitos indispensables para su funcionamiento.
- **Pod:** Es el objeto más pequeño de kubernetes y su función es alojar uno o más pods para asignarles una dirección de red adecuada con el fin de comunicarlos con otros contenedores del sistema.

- **Nodo:** Es un elemento del sistema que simula la existencia de un servidor físico o virtual en donde se despliegan los objetos de kubernetes.
- **Cluster:** Se define como la unión de varios servidores, ya sea de forma virtual o física.
- **hpa:** Horizontal Pod Autoescaler, es un objeto de kubernetes que permite instanciar las métricas y parámetros adecuados para determinar cuándo un pod debe ser escalado o reducido según la demanda de peticiones.
- **resiliencia:** Es la tolerancia a fallo con la cual cuenta un cluster y cada uno de los elementos que lo componen
- **Escalabilidad:** Es la capacidad que tiene un sistema para crecer o decrecer según la demanda de usuarios en un momento dado.
- **Persistencia:** Es la capacidad que tiene un sistema para mantener un estado de algún elemento que haga parte de él.
- **StatefulSet:** Es un objeto de kubernetes que permite la instanciación de uno o varios pods dentro del sistema.
- **DaemonSet:** Es un objeto de kubernetes que permite ejecutar un demonio con un fin específico en cada uno de los nodos del cluster.
- **Ingress Controller:** Es un elemento de kubernetes que permite exponer los servicios internos del cluster hacia el exterior.
- **ReplicaSet:** Es un objeto de kubernetes que se encarga de monitorear el estado deseado de los pods dentro del cluster
- **ETCD:** Es un componente de almacenamiento llave:valor, que se encarga de almacenar la configuración base de kubernetes
- **Gitlab:** Es una aplicación de código abierto para el manejo y control de versiones de software
- **Redis:** Es un motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes pero que puede ser usada como una base de datos durable o persistente.
- **Postgres:** Es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto.

1.11. FACTIBILIDAD DEL PROYECTO

1.11.1. Factibilidad técnica o tecnológica

Para desarrollar satisfactoriamente el modelo para el control de versiones de software basado en orquestación de microservicios con kubernetes, contamos con algunas herramientas muy completas para nuestro fin.

Google ofrece de forma gratuita el orquestador Kubernetes oficial, es un sistema de código libre para la automatización del despliegue, ajuste de escala y manejo de aplicaciones en contenedores que fue originalmente diseñado por Google y donado a la Cloud Native Computing Foundation (parte de la Linux Foundation). Soporta diferentes ambientes para la ejecución contenedores, incluido Docker.

Para la solución de software control de versiones optamos por las soluciones disponibles de código abierto, es un modelo de desarrollo de software basado en la colaboración abierta. Se enfoca más en los beneficios prácticos (acceso al código fuente) que en cuestiones éticas o de libertad que tanto se destacan en el software libre. Para muchos el término “libre” hace referencia al hecho de adquirir un software de manera gratuita, pero más que eso, la libertad se refiere al poder modificar la fuente del programa sin restricciones de licencia, ya que muchas empresas de software encierran su código, ocultándolo, y restringiéndose los derechos a sí misma.

1.11.2. Factibilidad Operativa

En el desarrollo de este proyecto participan dos estudiantes de la Universidad Distrital Francisco José de Caldas, acompañados por el ingeniero director del proyecto; además cuentan con el apoyo de los docentes de la universidad, quienes aportan en la formación profesional de los estudiantes.

Gracias a las diferentes herramientas de software libre podemos desarrollar el modelo para el control de versiones de software basado en orquestación de microservicios con kubernetes.

1.11.3. Factibilidad Legal

Este modelo es un sistema abierto a la comunidad y se encuentra bajo la protección de cualquiera de los términos y condiciones de la licencia GPL versión tres.

La Licencia Pública General de GNU es una licencia libre, para el software y otros tipos de obras.

Las licencias para la mayoría del software y otros trabajos prácticos están diseñadas para quitarle a usted la libertad de compartir y modificar esos trabajos. Por el contrario, la Licencia Pública General de GNU pretende garantizar la libertad de compartir y modificar todas las versiones de un programa – para asegurarse de que sigue siendo software libre para todos sus usuarios. Nosotros, la Fundación para el Software Libre, usamos la Licencia Pública General GNU

para la mayoría de nuestro software; también se aplica a cualquier trabajo realizado de esta forma por sus autores. Usted puede aplicarla a sus propios programas.

Cuando hablamos de software libre, estamos refiriéndonos a libertad, no de precio. Nuestras Licencias Públicas Generales están diseñadas para asegurarnos de que usted tiene la libertad de distribuir copias de software libre (y cobrar por ello si lo desea), que reciba el código fuente o que pueda conseguirlo si lo quiere, de que se puede cambiar el software o utilizar fragmentos de él en nuevos programas libres, y que usted sabe que puede hacer estas cosas.

Para proteger sus derechos, necesitamos evitar que otros le nieguen estos derechos o pedirle que renuncie a los derechos. Por lo tanto, usted tiene ciertas responsabilidades si distribuye copias del software, o si lo modifica: responsabilidades que persiguen respetar la libertad de los demás.

Por ejemplo, si distribuye copias de un programa, ya sea gratuitamente o a cambio de una contraprestación, debe transmitir a los destinatarios los mismos derechos que usted recibió. Debe asegurarse de que ellos también reciben, o pueden conseguir el código fuente. Y debe mostrarles estas condiciones de forma que conozcan sus derechos.

Los desarrolladores que usen la GPL GNU protegen sus derechos de dos pasos: (1) imponen derechos al software, y (2) le ofrecemos esta licencia que le da permiso legal para copiar, distribuir y / o modificarlo.

1.11.4. Factibilidad Económica

Tabla 1. *Costos del hardware utilizado en el desarrollo del proyecto*

COSTOS DEL HARDWARE				
RECURSO	DESCRIPCIÓN	VALOR UNITARIO	CANTIDAD	IVA
Computador 1	Memoria RAM 6GB Disco duro 750GB Procesador Intel Core i5-3230M 2,6 GHz	$1,500,000.00	1	$1,785,000.00
Computador 2	Memoria RAM 16GB Disco duro 700GB Procesador Intel Core i7-3537U 2,0 GHz	$4,200,000.00	1	$4,998,000.00
Computador 3	Memoria RAM 12 GB Disco duro 500 GB SSD Procesador Core i7-7500U - 2.7 GHz	$2,622,780.00	1	$3,238,000.00
			TOTAL	$10,021,000.00

Fuente: Autores

Tabla 2. *Costos del software*

COSTOS DE SOFTWARE				
RECURSO	DESCRIPCIÓN	VALOR UNITARIO	CANTIDAD	SUBTOTAL
VirtualBox	Software de virtualización	$0.00	3	$0.00
CoreOS	S.O. de los nodos Kubernetes	$0.00	3	$0.00
Gitlab	Sistema de control de versiones e software	$0.00	1	$0.00
Kubernetes	Orquestador de contenedores Docker	$0.00	1	$0.00
HAproxy	Balanceador de carga	$0.00	1	$0.00
Redis	Base de datos de caché	$0.00	1	$0.00

PostgreSQL	Base de datos que usa GitLab	$0.00	1	$0.00
Docker	software de virtualización de contenedores	$0.00	3	$0.00
ETCD	Base de Datos Clave valor	$0.00	3	$0.00
Software de Monitoreo	Software para medir métricas de hardware	$0.00	1	$0.00
			TOTAL	$0.00

Fuente: Autores

Tabla 3. *Costos recursos humanos del proyecto*

RECURSOS HUMANOS				
NOMBRE	FUNCIÓN	VALOR HORA	CANTIDAD	SUBTOTAL
Leonardo Delgado	Estudiante de ingeniería telemática	$50,000.00	137	$6,850,000.00
Oscar David Pineda	Estudiante de ingeniería telemática	$50,000.00	137	$6,850,000.00
Jairo Hernández	Ingeniero de Sistemas	$120,000.00	25	$3,000,000.00
			TOTAL	$16,700,000.00

Fuente: Autores

Tabla 4. *Otros gastos del proyecto*

OTROS				
RECURSO	DESCRIPCIÓN	VALOR UNITARIO	CANTIDAD	SUBTOTAL
Papelería	Hoja de impresión	$100.00	$1,000.00	$100,000.00
Transporte	Costos de traslado	$2,300.00	$200.00	$460,000.00
			TOTAL	$560,000.00

Fuente: Autores

Tabla 5. *Total, de costos del proyecto*

COSTO TOTAL	
Costos Hardware	$10,021,000.00
Costos Software	$0.00
Recurso Humano	$16,700,000.00
Otros	$560,000.00
TOTAL	$27,281,000.00

Fuente: Autores

2. ANÁLISIS DE MODELOS DE ORQUESTACIÓN ACTUALES

2.1. INTRODUCCIÓN

En este apartado, se explica el funcionamiento de algunos de los orquestadores actuales, brindando una visión transversal de los componentes que intervienen y tomando como referencia la documentación oficial de cada uno de los fabricantes. Sin embargo, teniendo en cuenta que kubernetes es el núcleo de esta investigación, se ahondará más en esta arquitectura.

2.2. DOCKER SWARM

Docker swarm se puede definir como un administrador y/o orquestador de contenedores, ejecutados dentro de un clúster. Está compuesto por varios hosts de Docker que se ejecutan en modo enjambre. Cada uno de estos hosts cumplen una función específica dentro de la composición de los servicios. Por lo tanto, pueden tomar el rol de Manager o Worker, o incluso ambos. Como su nombre lo indica el Manager es el encargado de programar el despliegue y los recursos de los servicios (número de réplicas, recursos de red, almacenamiento disponible, puertos que el servicio expone, y más). en todos los nodos del clúster. Y los Worker se encargan de la ejecución y procesamiento de los servicios que el Manager le asigne[17].

NODO: Es una instancia que forma parte del "enjambre", comparte sus recursos de procesamiento y almacenamiento para el despliegue de contenedores, cuya configuración es asignada desde un Administrador.

SERVICIOS Y TAREAS: Un servicio es la definición de las tareas a ejecutar en cada uno de los nodos del clúster. Así mismo, es el encargado de interactuar con el usuario final al momento de hacer un llamado a cualquier recurso disponible en el enjambre, que esté debidamente expuesto al momento de configurarlo.

Por otra parte, la tarea es la encargada de contener todos los comandos requeridos para el funcionamiento y despliegue de los servicios dentro de un contenedor. Es la unidad de programación atómica del enjambre. Principalmente, los nodos Manager son quienes asignan tareas a los nodos worker.

BALANCEO DE CARGA: Docker Swarm, expone servicios con el fin de que el usuario final pueda acceder a cada microservicio conforme lo necesite, por lo cual, depende de un balanceador de carga, que cumpla la función de regulador para

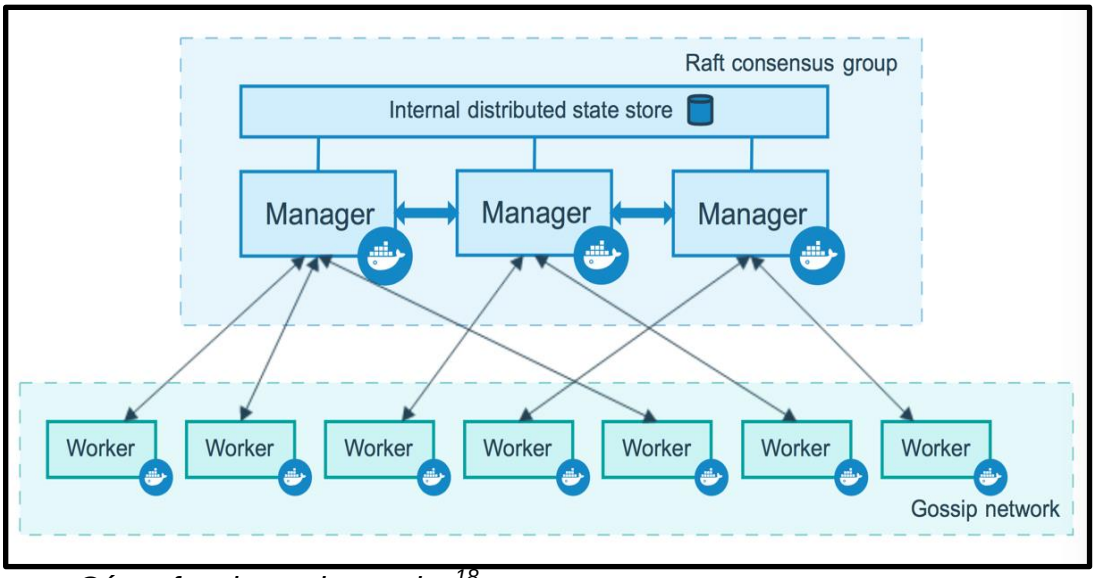
[17] GONZÁLEZ, Alberto - Docker Guía Práctica. Bogotá - Colombia. 2018.

que dirija correctamente las peticiones teniendo en cuenta los recursos, estado y saturación de todos los nodos del clúster.

¿Cómo funciona Docker Swarm?

Para iniciar todos los nodos del clúster deben correr una utilidad llamada Docker Engine, la cual permite acceder a todas las funcionalidades, librerías y comandos necesarios para el uso de Docker Swarm. Para explicar en mayor detalle la comunicación entre Nodos Manager y Worker, se debe entender el siguiente gráfico

Figura 10. Funcionamiento de los nodos de Docker Swarm



Fuente: Cómo funcionan los nodos[18]

Como se observa, en primer lugar, se debe tener un almacenamiento compartido entre los nodos del clúster, la cual guarda información referente al funcionamiento del clúster, también puede contener datos montados a través de volúmenes.

En segundo lugar, se tienen los nodos Manager, los cuales se encargan de varias tareas, entre las más relevantes se encuentra, programación de los contenedores (Quiere decir en cuál host se va a desplegar un servicio o asignar una tarea), configuración de los endpoints para el acceso a los servicios y monitoreo constante al estado del clúster.

[18] Documentación Oficial Docker - Cómo funcionan los nodos [en línea]. [09 de Abril de 2019]. Disponible en la web: https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/

En tercer lugar, se encuentran los nodos worker, los cuales también representan instancias de Docker, pero con el propósito de ejecutar contenedores. Estos nodos no hacen parte del estado distribuido Raft, puesto que no cuentan con la capacidad de tomar decisiones en la programación de tareas.

¿Cómo funcionan los Servicios?

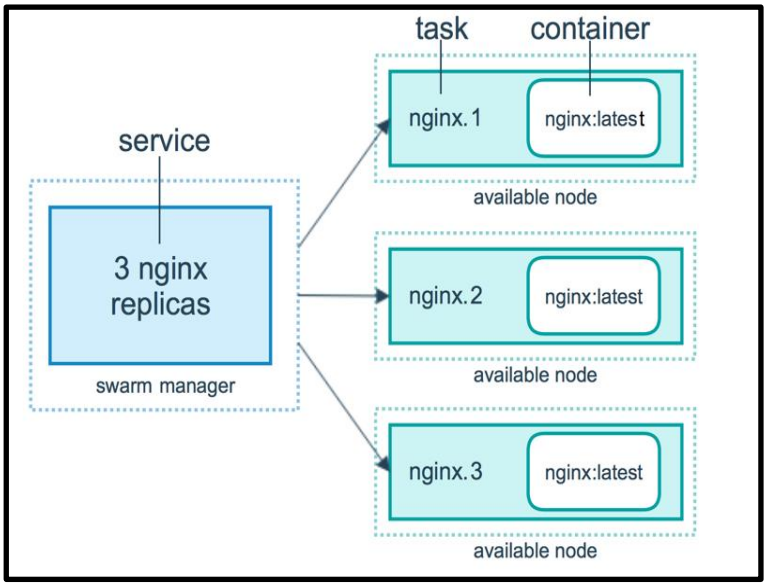
El servicio es el punto de contacto entre el cliente y el clúster de microservicios, es por esta razón que este tipo de orquestador, ofrece un mayor grado de resiliencia en comparación con una instalación standalone.

Para crear correctamente un servicio se deben tener en cuenta diferentes aspectos para lograr una buena configuración, entre ellos, se pueden resaltar:

- Imagen de Docker que se desea desplegar
- Los puertos expuestos del contenedor y adicionalmente los que se van a mapear en el host que lo contiene
- Enlace con otros microservicios
- Límites de memoria y CPU
- Volúmenes de almacenamiento de los contenedores
- Una política de actualización continua.
- Número predefinido de réplicas del servicio

Para explicarlo de mejor forma, se relaciona la siguiente figura y a continuación su explicación.

Figura 11. Funcionamiento de los servicios de Docker Swarm



Fuente: Cómo funcionan los servicios[19]

En la imagen se observa un clúster compuesto de 4 nodos, en donde uno de ellos es de tipo Manager y el resto de tipo worker, lo cual permite objetar que el nodo manager únicamente expone los servicios requeridos y él mismo los delega en el resto de nodos.

En este sentido, esta arquitectura tiene una ventaja y es que un servicio puede tener n réplicas, de acuerdo a la cantidad de recursos físicos que tenga el cluster en general.

Ahora bien, es necesario comprender, cómo se ejecutan estos contenedores antes de convertirse en un servicio visible para el usuario. Inicialmente, se define una tarea que funciona como una instrucción atómica, es decir se define una estructura lógica con diferentes instrucciones pero que finalmente ejecutan una sola acción, en este caso el despliegue de un contenedor. En esta secuencia, se define el número de réplicas que debe contener el servicio para aceptar peticiones de cualquier tipo, según como se defina inicialmente. Es importante aclarar que si una tarea o contenedor falla, el clúster debe volver a repetir el proceso cuantas veces sea necesario.

Seguridad

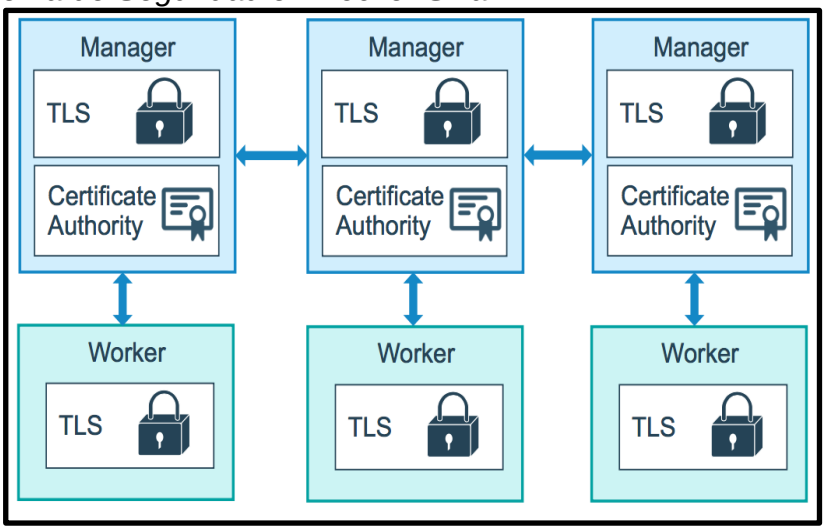
[19] Documentación Oficial Docker - Cómo funcionan los servicios [en línea]. [09 de Abril de 2019]. Disponible en la web: https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/

Docker Swarm admite el sistema de infraestructura de clave pública, el cual viene integrado con docker y se usa para el transporte, autenticación, autorización y cifrado de los datos.

Inicialmente cuando se inicia el clúster de Swarm, por defecto el reconoce ese mismo host como administrador y crea una CA, junto con un par de llaves que permiten la comunicación cifrada entre todos los nodos que se integren posteriormente.

Posteriormente Docker Swarm genera dos tokens, uno para unir un nodo de tipo worker y otro para unir un nodo de tipo manager. Los cuales son creados a partir de un trozo de caracteres extraídos del certificado emitido por la CA y un secret escogido aleatoriamente.

Figura 12. Esquema de Seguridad en Docker Swarm



Fuente: Administre la seguridad enjambre con infraestructura de clave pública (PKI)[20]

Una vez el nodo es registrado dentro del clúster con el token generado en el host manager, se genera un nuevo certificado que permite enviar de forma cifrada la información del nodo que se está conectando, más específicamente el ID y CN, el cual permite que el host sea único dentro del clúster.

Redes

[20] Documentación Oficial Docker - Administre la seguridad enjambre con infraestructura de clave pública (PKI) [en línea]. [09 de Abril de 2019]. Disponible en la web: https://docs.docker.com/engine/swarm/how-swarm-mode-works/pki/

Docker swarm usa overlay controller para gestionar internamente redes entre varios hosts del demonio de Docker. Esta red se superpone a la red interna de los contenedores, lo cual permite establecer comunicación entre los microservicios expuestos en el clúster.

La primera red se conoce como ingress y se encarga del control y manejo del tráfico de datos relacionados con Docker Swarm. Viene configurada por defecto y si no se especifica una red distinta al momento de iniciar el clúster, éste se conectará a ella de forma predeterminada.

La segunda red es conocida como docker_gwbridge (puente virtual que conecta las redes superpuestas), y está encargada de establecer comunicación entre los diferentes dominios de docker desplegados en cada uno de los hosts del clúster.

Docker swarm ofrece un stack de opciones para el manejo y configuración de red de todos los contenedores del clúster (crear, eliminar, modificar) y además admite el cifrado al momento de comunicar microservicios. Por defecto, utiliza el algoritmo AES en modo GCM (Galois/Counter Mode), los cuales viajan a través de túneles IPSec en el nivel de vxlan.

Almacenamiento

Uno de los grandes desafíos de este tipo de tecnología ha sido la implementación de almacenamiento persistente, teniendo en cuenta que tanto los servicios como los contenedores son volátiles; esto quiere decir que una vez finaliza el ciclo de vida de uno de ellos, la información que contiene también se destruirá. Por lo tanto, es necesario explicar de forma práctica la forma en que docker Swarm gestiona el almacenamiento.

Docker almacena los datos de forma local, debido a que la filosofía de los contenedores, es que deben ser inmutables, lo que significa que su configuración debe ser la misma durante todo su ciclo de vida. Por lo tanto, no es recomendable modificar un contenedor en ejecución para ambientes productivos.

Docker swarm también, ofrece 3 tipos de almacenamiento persistente, la primera y más popular se realiza mediante el uso de volúmenes, los cuales mapean una carpeta del sistema (/var/lib/docker/volumes/) en donde corre el contenedor, con una carpeta interna del docker.

El segundo tipo de montaje, permite almacenar información en cualquier parte del sistema no hay ningún tipo de restricción y es el más utilizado por los usuarios.

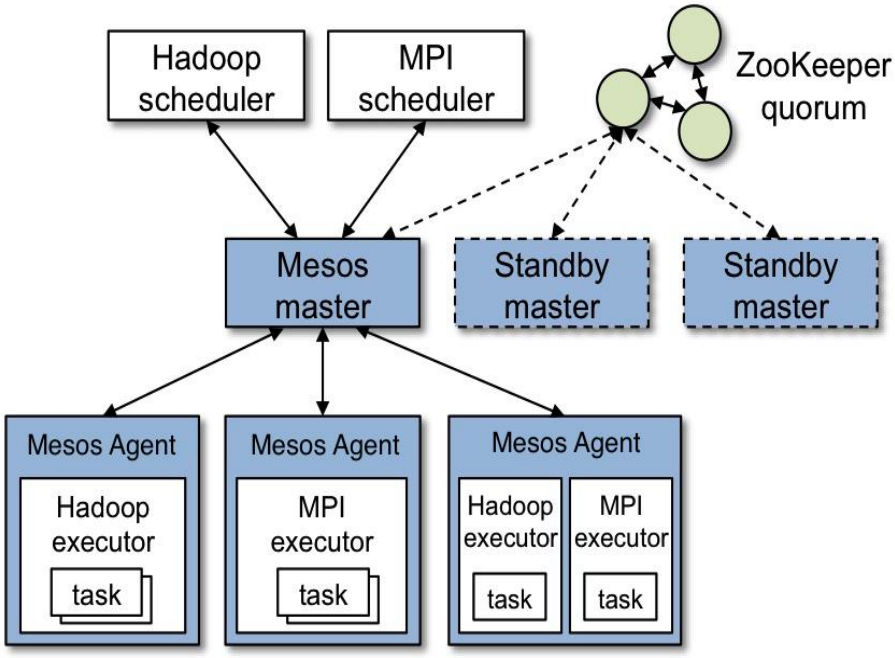
El tercer tipo, se utiliza para almacenamiento efímero, es decir, para archivos temporales que sean necesarios para un servicio completo, lo que implica que todos los contenedores asociados a un microservicio usen la misma información en ese instante de tiempo.

2.3. APACHE MESOS

Mesos es un orquestador de servicios, el cual funciona de forma similar al kernel de linux, corre en cada máquina del clúster y provee diferentes servicios con API's para la administración y programación de elementos entre todos los ambientes interactuantes dentro de un sistema.[21]

A nivel técnico Mesos ofrece la siguiente arquitectura para la orquestación de los servicios:

Figura 13. Arquitectura de Apache Mesos



Fuente: Mesos Architecture[22]

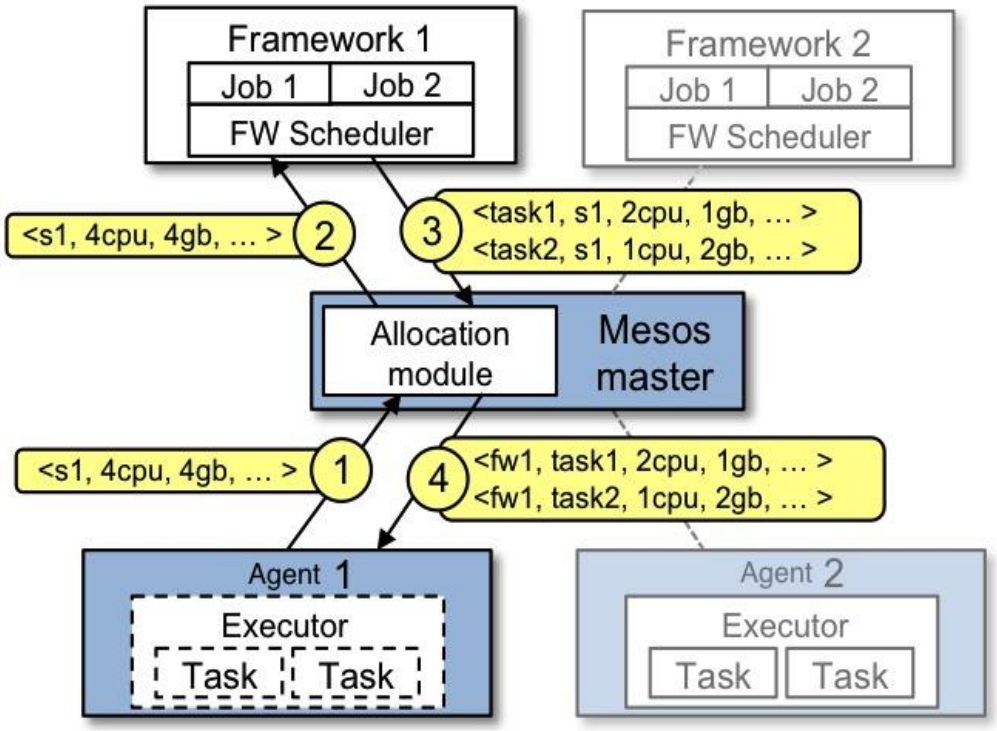
Lo primero que hay que resaltar a partir de la imagen, es la intervención de un nodo maestro el cual establece comunicación con los agentes de Mesos, desde

[21] Documentación Oficial Apache Mesos - Mesos Architecture [en línea]. [11 de Abril de 2019]. Disponible en la web: http://mesos.apache.org/documentation/latest/architecture/
[22] Ibid.

allí se administran los recursos necesarios para el funcionamiento adecuado del clúster; la forma en que funciona es a partir de recursos disponibles, se hacen ofertas a los nodos secundarios y dependiendo de la necesidad de los mismos, proceden a aceptarlos o denegarlos.

Por otra parte, el nodo primario, se compone de dos módulos indispensables para su funcionamiento, un scheduler que se define en el maestro con el fin de brindar los recursos computacionales necesarios para el funcionamiento de los microservicios y un executer que se instancia en los agentes de los nodos para ejecutar las tareas del framework, teniendo en cuenta que este agente conoce cuál de los recursos debe utilizar dentro del cluster. Finalmente, cuando todos los componentes de Mesos están en sincronía y todos los componentes tienen definidas sus tareas, entonces se inicia el proceso esperado.

Figura 14. Oferta de recursos Apache Mesos



Fuente: Example of resource offer[23]

Para entender el proceso, es necesario sintetizar la documentación expuesta por Apache, y se realizará de la siguiente manera:

[23] Documentación Oficial Apache Mesos - Example of resource offer [en línea]. [16 de mayo de 2019]. Disponible en la web: http://mesos.apache.org/documentation/latest/architecture/

1. El Agente 1 informa al maestro que tiene 4 CPU y 4 GB de memoria libre. El maestro luego invoca el módulo de la política de asignación, que le dice que al marco 1 se le debe ofrecer todos los recursos disponibles.
2. El maestro envía una oferta de recursos que describe lo que está disponible en el agente 1 al marco 1.
3. El programador del marco responde al maestro con información sobre dos tareas para ejecutar en el agente, utilizando <2 CPU, 1 GB de RAM> para la primera tarea, y <1 CPU, 2 GB de RAM> para la segunda tarea.
4. Finalmente, el maestro envía las tareas al agente, que asigna los recursos apropiados al ejecutor del marco, que a su vez inicia las dos tareas. Debido a que 1 CPU y 1 GB de RAM aún no están asignados, el módulo de asignación ahora puede ofrecerlos al marco 2 y el proceso se repite cada vez que haya recursos y tareas por ejecutar.

La arquitectura de Mesos permite escalar y permite que los marcos evolucionen de manera independiente, adicionalmente el modelo está diseñado para que el maestro brinde la potestad a los marcos de rechazar ofertas. Un marco rechazará las ofertas que no cumplan con sus restricciones y acepte las que sí lo hacen. Por ende, se sintetizan estas tareas represadas en algo llamado “programación de retardo”, en la cual los marcos esperan un tiempo limitado para adquirir nodos que almacenan los datos de entrada y permite que los procesos se ejecuten en un momento determinado sin afectar la integridad y el funcionamiento de los microservicios.

2.4. KUBERNETES

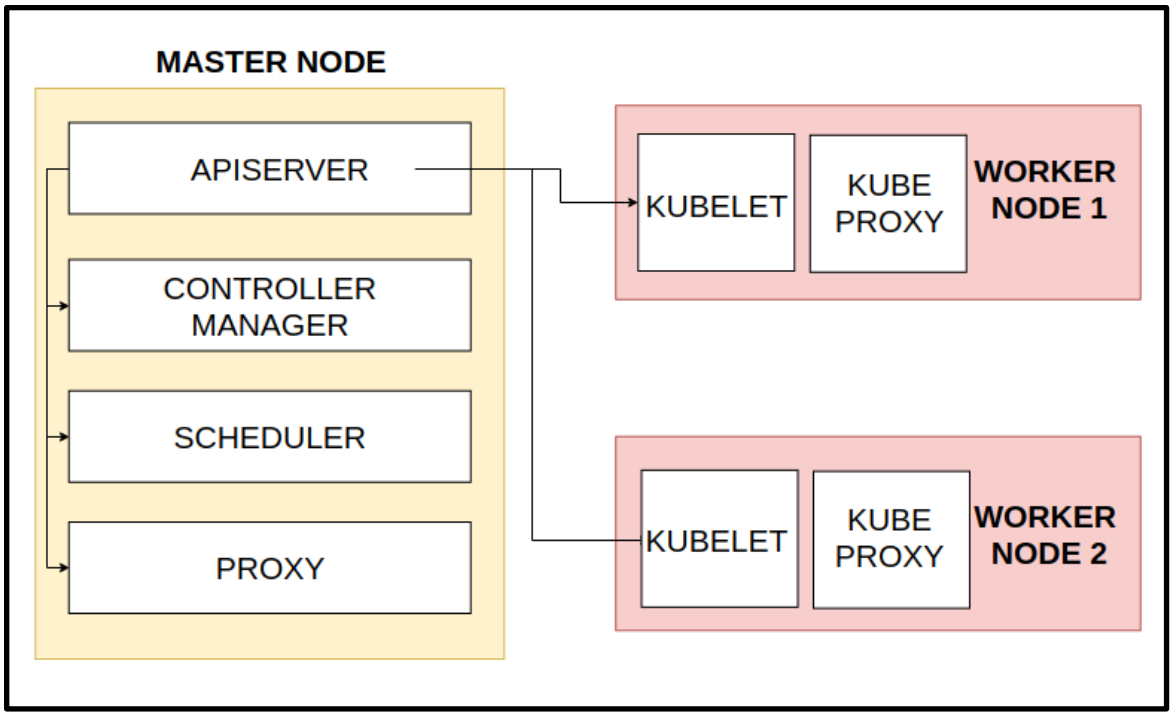
Kubernetes es conocido como un sistema de código abierto para automatizar la implementación, el escalado y la administración de aplicaciones en contenedores, Agrupa los contenedores que conforman una aplicación en unidades lógicas para una fácil administración y manejo. Kubernetes se basa en 15 años de experiencia en la ejecución de cargas de trabajo de producción en Google , combinadas con las mejores ideas y prácticas de la comunidad.[24]

Lo primero que se debe tener en cuenta para entender la arquitectura y el funcionamiento de kubernetes, es que toda la comunicación a nivel de cluster se da a partir de la API, la cual se comunica con los endpoints para cada servicio que se presta, además esta API sirve para describir el estado deseado de un clúster, como por ejemplo, qué aplicaciones u otras cargas de trabajo se desean ejecutar, qué imágenes de contenedor se deben usar, la cantidad de réplicas por microservicio, qué recursos de red y disco desea que estén disponibles, entre otros.

Para poder lograr los estados deseados, mencionados en el párrafo anterior es indispensable conocer todos los componentes de Kubernetes, tanto en sus nodos master como en los nodos worker, para ello se recurrirá a la siguiente imagen, la cual representa de forma visual los elementos que intervienen

[24] "Concepts - Kubernetes." 1 mayo 2019, https://kubernetes.io/docs/concepts/. Fecha de acceso 16 mayo 2019.

Figura 15. Componentes de Kubernetes



Fuente: Autores

Apiserver: El Apiserver de Kubernetes válida y configura datos para los objetos de API que incluyen pods, servicios, controladores de replicación y otros. El servidor API realiza las operaciones REST y proporciona la interfaz al estado compartido del clúster a través del cual interactúan todos los demás componentes.

Controller Manager: El Controller Manager de Kubernetes es un demonio que integra los bucles de control del núcleo enviados con Kubernetes. Por lo cual, un controlador es un bucle de control que vigila el estado compartido del clúster a través del Apiserver y realiza cambios intentando mover el estado actual hacia el estado deseado.

Scheduler: El scheduler de Kubernetes es una función de carga de trabajo rica en políticas, con reconocimiento de topología y que afecta significativamente la disponibilidad, el rendimiento y la capacidad. El scheduler debe tener en cuenta los requisitos de recursos individuales y colectivos, los requisitos de calidad de servicio, las restricciones de hardware y software, las especificaciones de afinidad y antiafinidad, la localidad de datos, la interferencia entre cargas de trabajo, los plazos, etc. Los requisitos específicos de la carga de trabajo se exponen a través de la API según sea necesario.

Proxy: El proxy de Kubernetes se ejecuta en cada nodo (Master y Worker). Esto refleja los servicios tal como se definen en la API de Kubernetes en cada nodo y puede realizar reenvío de secuencias TCP, UDP y SCTP o redireccionamiento de TCP, UDP y SCTP a través de un conjunto de backends. Las direcciones IP y los puertos del clúster de servicio se encuentran actualmente a través de variables de entorno compatibles con enlaces de Docker que especifican puertos abiertos por el proxy de servicio.

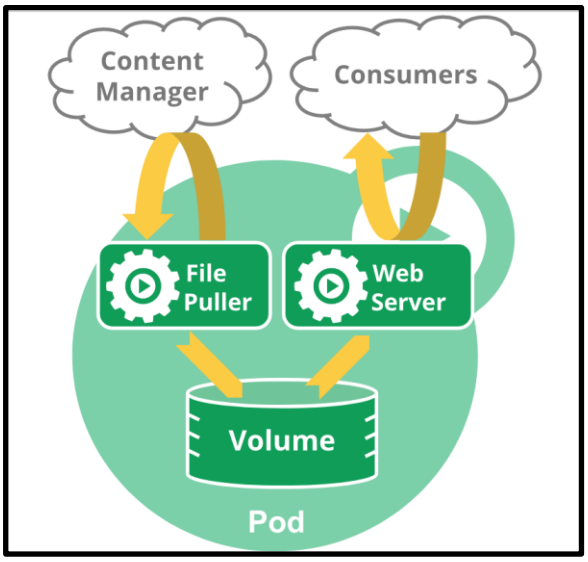
Kubelet: Es el "agente del nodo" primario que se ejecuta en cada nodo. El kubelet funciona en términos de un PodSpec. Un PodSpec es un objeto YAML o JSON que describe un pod. El kubelet toma un conjunto de PodSpecs que se proporcionan a través de varios mecanismos y garantiza que los contenedores descritos en esos PodSpecs estén funcionando y en buen estado. El kubelet no administra contenedores que no fueron creados por Kubernetes.

Adicionalmente, en kubernetes hay una serie de objetos en una capa más arriba, los cuales permiten hacer uso y despliegue de microservicios, a continuación, se realizará una breve descripción que permita entender de forma detallada la implementación de cada uno:

Pod: Un Pod es el componente básico de Kubernetes, la unidad más pequeña y simple en el modelo de objetos de Kubernetes que se crea o implementa. Un Pod representa los procesos que se ejecutan dentro de un Clúster.

Un Pod encapsula uno o más contenedores de una aplicación, los recursos de almacenamiento, una IP de red única y las opciones que rigen cómo deben ejecutarse los contenedores.

Figura 16. Funcionamiento de un pod



Fuente: Kubernetes[25]

Como se observa en la imagen el pod internamente requiere de tres componentes para poder comunicar los contenedores, en caso de existir más de dos de ellos.

- **Servicio:** Los servicios hacen referencia a la capa de conexión entre los microservicios y los ingress controller, es decir son los encargados de redireccionar las peticiones correspondientes a donde corresponda con ayuda del servidor dns interno.

  Este componente es de mucha importancia, puesto que los pods al ser volátiles, cambian de dirección IP cada vez que se inicia una nueva instancia del mismo, lo que hace que para la capa de servicios sea imposible rastrearlos.

  Además de esto, al apuntar a diferentes réplicas, el servicio accede a cualquiera de los pods disponibles para la ejecución de una petición en específico.

  Un servicio se puede ver como una abstracción de un conjunto de pods; por ejemplo, suponga que se tiene un microservicio de frontend con 1 réplica y un microservicio de backend con 3 réplicas. Para el frontend es transparente quien procesa su solicitud. Esto se da gracias a los servicios.

[25] "Pods - Kubernetes." 2 jul.. 2019, https://kubernetes.io/docs/concepts/workloads/pods/pod/. Fecha de acceso 12 jul.. 2019.

- **Volumen:** Uno de los grandes desafíos de los microservicios es el almacenamiento persistente, dado que la propiedad de inmutabilidad y volatilidad de los pods es innegable.

  Es a partir de esta necesidad que aparece el concepto de volúmenes, el cual cumple el rol de alojamiento de datos y de información para que cualquier pod dentro del cluster lo pueda consumir.

  Kubernetes permite satisfacer esta necesidad de diversas formas al integrarse con muchas soluciones de nube, sin embargo, las más populares son dos. En la primera intervienen otros conceptos como PV (Persistent Volume) y PVC (Persistent Volume Claim), los cuales, se definen como bloques de disco que son mapeados en cada pod dentro de su instanciación inicial, y apuntan a un espacio de almacenamiento dentro del cluster, Si se analiza, esta opción no es muy viable para sistemas que procesan gran cantidad de información.

  En segundo lugar, kubernetes admite el uso de NFS (Network File System), el cual se instancia de la misma manera dentro de los pods, pero con la diferencia de que todo el esquema de almacenamiento será almacenado en una parte externa al cluster, lo que permite mejor manejo, monitoreo y lo hace más escalable de acuerdo a la demanda de información.

- **Namespace:** Un namespace es un objeto que divide un solo clúster de Kubernetes en múltiples clústeres virtuales. lo que permite aislar elementos de forma práctica sin eliminar la funcionalidad de acceso entre ellos mismos.

  Por defecto kubernetes, crea tres Namespaces por defecto:

  - ***Default*:** Es donde se crean todos los objetos de kubernetes de forma predeterminada, es decir, si dentro de la creación de cada componente no se instancia su namespace, es aquí donde será desplegado.
  - ***kube-public*:** Es un espacio para recursos que todos los usuarios pueden leer públicamente. Este espacio generalmente está reservado para el uso del clúster.
  - ***kube-system*:** Es un espacio para alojar los objetos que corresponden al sistema (apiserver, controller, scheduler, proxy, entre otros)

- **ReplicaSet:** Este objeto es el encargado de revisar que los elementos dentro del cluster tengan las réplicas correspondientes, de acuerdo a su instanciación inicial.

  Un ReplicaSet se define con campos, que incluyen un selector que especifica cómo identificar los Pods que puede adquirir, varias réplicas que indican cuántos Pods se deben mantener y una plantilla de pods que especifica los datos de los nuevos Pods que debe crear para alcanzar el número deseado.

  Un ReplicaSet identifica nuevos Pods para adquirir utilizando su selector. Si hay un Pod que no tiene el parámetro OwnerReference no es un controlador y coincide con el selector de ReplicaSet, dicho ReplicaSet lo adquirirá de inmediato.[26]

- **Deployment:** Es un objeto que describe un estado deseado en una Implementación, y por medio de un ReplicaSet cambia el estado real al estado deseado a una velocidad controlada. Este componente puede definir implementaciones para crear nuevos conjuntos de réplicas o eliminar implementaciones existentes y adoptar todos sus recursos con las nuevas implementaciones.

  Es a partir de este tipo de objetos que se puede escalar el número de réplicas según se defina en la creación de los archivos de configuración. Es un objeto de jerarquía más alta que los pods o los replicaset.

- **StatefulSet:** StatefulSet es el objeto de la API de carga de trabajo utilizado para administrar aplicaciones con estado.

  Administra la implementación y la escala de un conjunto de Pods, y proporciona garantías sobre el orden y la singularidad de estos Pods.

  Como un deployment, un StatefulSet administra Pods que se basan en una especificación de contenedor idéntica. A diferencia de una deployment, un StatefulSet mantiene una identidad fija para cada uno de sus Pods. Estos pods se crean a partir de la misma especificación, pero no son

[26] "ReplicaSet - Kubernetes." 23 jul.. 2019, https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/. Fecha de acceso 23 jul.. 2019.

intercambiables: cada uno tiene un identificador persistente que mantiene a través de cualquier reprogramación.[27]

Un StatefulSet funciona bajo el mismo patrón que cualquier otro Controlador. simplemente se debe definir su estado deseado en un objeto StatefulSet, y el controlador StatefulSet realiza las actualizaciones necesarias para llegar allí desde el estado actual.

- **DaemonSet:** Un DaemonSet garantiza que todos los nodos (o algunos) ejecuten una copia de un Pod. A medida que se agregan nodos al clúster, se agregan Pods a ellos. A medida que se eliminan los nodos del clúster, estos Pods se recolectan como basura. Eliminar un DaemonSet limpiará los Pods que creó.[28]

  Algunos de los ejemplos para este tipo de objetos son:

- la ejecución de un demonio de almacenamiento del clúster, en cada nodo.
- La ejecución de un demonio de recopilación de registros en cada nodo, como logstash.
- La ejecución de un Demonio de monitoreo de nodo en cada nodo, como Prometheus exporter, zabbix, nagios, new relic o cualquier otro agente.

- **Job:** Es un objeto de kubernetes que crea uno o más Pods y asegura que un número específico de ellos termine con éxito. A medida que los pods se completan exitosamente, el Trabajo rastrea los completamientos exitosos. Cuando se alcanza un número específico de completamientos exitosos, la tarea o el Job se completa. Eliminar un trabajo limpiará los Pods que creó.

  Un ejemplo simple es crear un objeto de trabajo para ejecutar de manera confiable un Pod hasta su finalización. El Job iniciará un nuevo Pod si el primer Pod falla o se elimina (por ejemplo, debido a una falla de hardware del nodo o un reinicio del nodo). También se puede utilizar un Job para ejecutar varios Pods en paralelo.

---

[27] "Statefulset - Kubernetes." 23 jul.. 2019, https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/. Fecha de acceso 23 jul.. 2019.

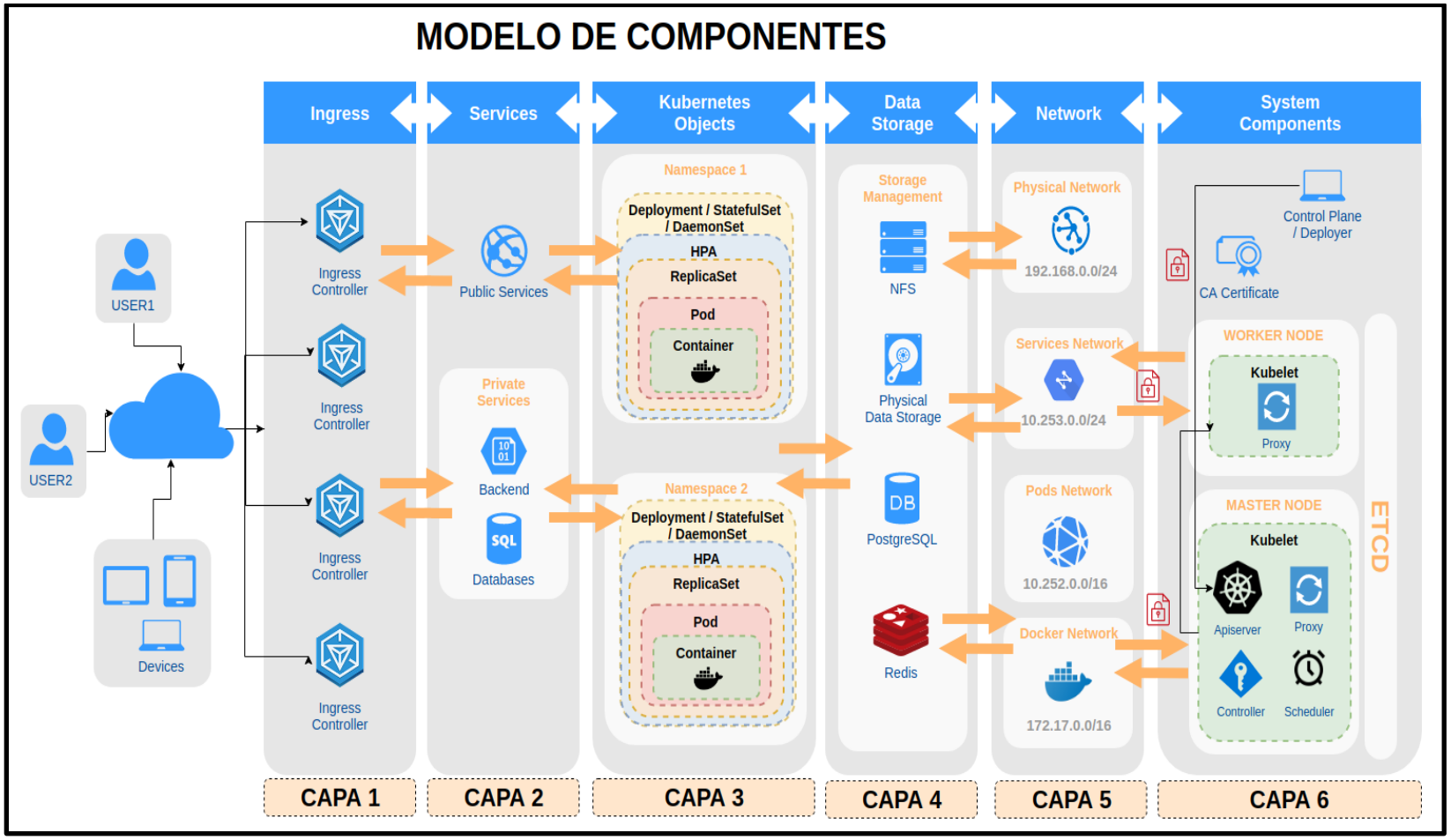
[28] "Daemonset - Kubernetes." 23 jul.. 2019, https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/. Fecha de acceso 23 jul.. 2019.

3. DEFINICIÓN DEL MODELO BASADO EN KUBERNETES

3.1. INTRODUCCIÓN

En el presente capítulo se muestra el diagrama planteado para la arquitectura del sistema, seguido de una explicación detallada de cada uno de sus elementos, para el fácil entendimiento de los elementos. Con el apoyo de la tabla 6, se pueden conocer cuáles son los objetos del sistema y su funcionalidad dentro y fuera del cluster.

Figura 17. Modelo general de componentes



Fuente: Autores

3.2. Explicación del modelo

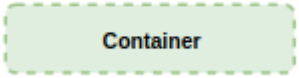
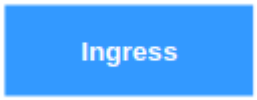
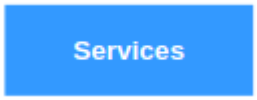
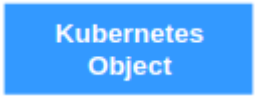
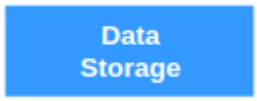
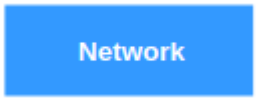
La figura 17 muestra el modelo general de componentes, en donde se sintetiza gráficamente la comunicación que se presenta entre todos los elementos del sistema telemático. En la siguiente tabla, se explican las figuras que se usaron para su construcción y se mencionan las funciones específicas dentro del modelo planteado.

Tabla 6. *Explicación de los elementos en el diagrama de componentes*

ELEMENTO	NOMBRE	DESCRIPCIÓN
	Usuario	Representa al usuario que consumirá cualquier servicio que se exponga
	Nube	Representa la conexión entre el usuario y el sistema. (En este caso es a nivel LAN)
	Servidor Web (ingress)	Expone hacia la nube los servicios que se le indiquen
	Servicios Públicos	Hace referencia a los servicios que son expuestos a través de un ingress
	Backend	Es el conjunto de servicios que interactúan con los datos privados del sistema
	Bases De Datos	Es el conjunto de servicios que almacenan los datos del sistema
	Network File System	Recurso de almacenamiento compartido en la red para que sea consumido por

		el pod o contenedor que lo requiera
	Almacenamiento Físico	Corresponde al almacenamiento físico de los datos del sistema
	Base De Datos Postgresql	Corresponde a la base de datos que aloja los datos de la aplicación, en este caso, Gitlab
	Base De Datos Redis	Corresponde a la base de datos que aloja los elementos llave-valor de la aplicación, en este caso, Gitlab
	Red Física	Hace referencia a la red física donde se conectan las máquinas virtuales que simulan el cluster
	Red De Servicios	Es la red destinada para exponer los servicios
	Red De Pods	Es la red destinada para el direccionamiento de pods
	Red De Contenedores	Es la red destinada para el direccionamiento de contenedores
	Apiserver	Corresponde al API de kubernetes para realizar acciones sobre el sistema
	Proxy	Kube-proxy, se ejecuta en cada nodo y se encarga de redireccionar el tráfico del sistema

	Controller	Es el componente encargado de hacer una revisión constante del estado de todos los componentes del sistema
	Scheduler	Es el componente responsable de programar el despliegue de un contenedor en cualquiera de los nodos del cluster
	Autoridad Certificadora	Se encarga de firmar los certificados generados para la comunicación por TLS
	Control Plane / Deployer	Elemento que se comunica con el API de kubernetes de forma cifrada para la manipulación de todos sus elementos
	Cifrado	Símbolo que representa que la información viaja cifrada
	Dispositivos	Hace referencia a todos los dispositivos que se puedan conectar a la aplicación web, en este caso, Gitlab
	Deployment / Statefulset / Daemonset	Elementos que contienen la configuración para los microservicios. Pertenecen al nivel más alto de jerarquía dentro de los objetos de kubernetes
	Replicaset	Es el elemento encargado de replicar la

		configuración establecida desde el nivel superior de jerarquía
	Pod	Es la unidad de ejecución básica de una aplicación en Kubernetes
	Contenedor	Es la unidad más pequeña de una aplicación en ejecución.
	Controlador	Es un conjunto de componentes mínimos, encapsulados a través del kernel para empaquetar un servicio a su mínima función
	Capa De Ingress	Hace referencia a la capa de exposición y direccionamiento de los microservicios
	Capa De Servicios	Es el conjunto de elementos visibles ante las peticiones de un ingress
	Capa De Objetos De Kubernetes	Es la capa que contiene los objetos de kubernetes necesarios para el funcionamiento de un servicio
	Capa De Almacenamiento	Es la capa responsable del almacenamiento persistente para los pods y demás componentes del sistema
	Capa De Red	Es la capa que contiene todas las redes necesarias para que el sistema funcione de

		forma adecuada
System Components	Capa De Componentes Del Sistema	Es la capa que contiene los elementos base para que el cluster de kubernetes funcione de forma adecuada

Fuente: Autores

Una vez definidos los objetos, es necesario explicar de forma detallada cómo se realizó la configuración para desplegar el servicio de gitlab, con todo su stack de componentes. De ahí que, a partir de la figura 17, es la forma más clara de entender el concepto que se desea plantear a lo largo de este documento.

El diagrama de componentes, contiene la transversalidad necesaria para describir los conceptos desde la primera hasta la última capa, por tal motivo se explicará brevemente a continuación, antes de iniciar con la configuración de los elementos que están integrados como anexos de este documento.

Las máquinas virtuales donde se ejecuta el cluster, cuentan con el sistema operativo CoreOS, debido a que este, redefine el sistema operativo como una distribución de Linux más pequeña y compacta, logrando un mejor rendimiento en la ejecución de procesos. Para ver la instalación de este sistema operativo revisar el ***anexo 1***.

La base de todo el sistema de kubernetes, es ETCD, el cual contiene los datos de tipo llave-valor que almacenan la configuración del cluster, para este proyecto se configuró en cluster con un nivel de tolerancia a fallo de (n/2)-1. Para más información de este componente ver el ***anexo 2***.

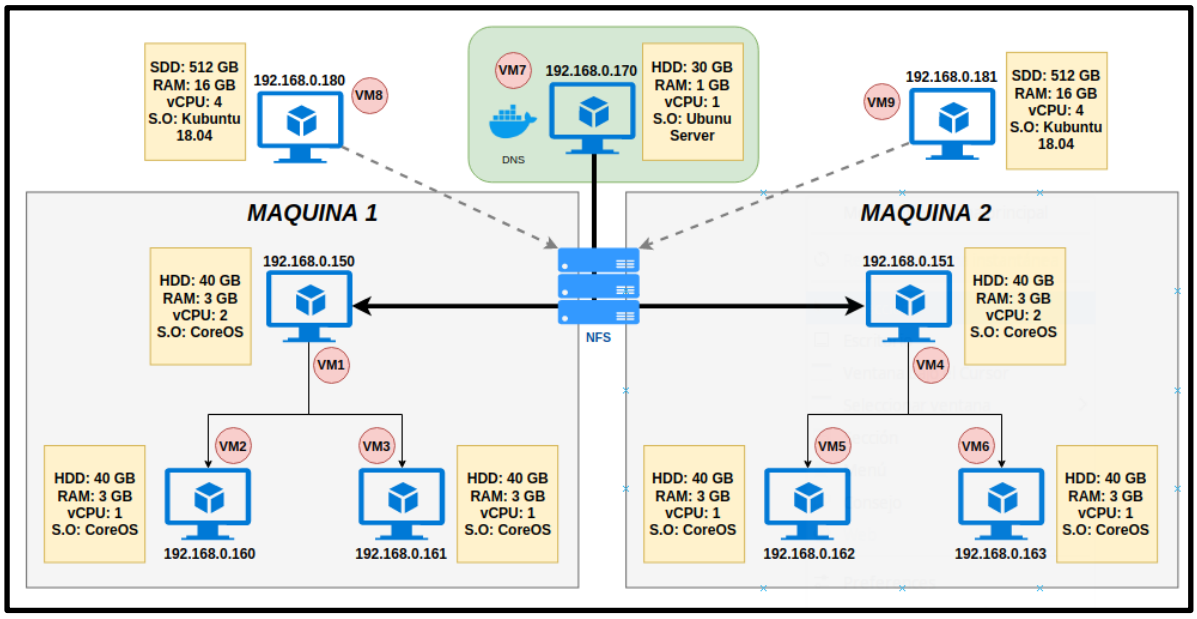
En segundo lugar, se realizó la configuración para la comunicación de las redes que intervienen en el sistema (Red física, Red de servicios, Red de pods y red de contenedores), para ello es necesario el uso de un CNI (Container Network Interface), el cual permite simular un router virtual para construir la tabla de enrutamiento entre las diferentes subcapas de red, en este caso, se usará flannel. Para comprender la configuración aplicada a este proyecto revisar el ***anexo 3***.

En este punto, todos los componentes base, están listos para admitir la configuración del cluster; el primer paso para lograr la configuración del cluster es definir cuántos nodos se despliegan y qué rol cumplen (worker o master), por

consiguiente, para este paso se deben tener en cuenta las siguientes consideraciones:

- Se instalaron 6 máquinas virtuales, con los recursos que se observan en la figura 18.
- Las máquinas virtuales est[an alojadas en dos computadores portátiles con los siguientes recursos:
    - **SSD**: 512 GB
    - **RAM**: 16 GB
    - **CPU**: intel Core i7 de 4 núcleos a 2.00 GHz
- Después se debe realizar la creación de los certificados SSL para la comunicación interna de los componentes, ver ***anexo 4***.
- Posterior a esto se procedió con la configuración del cluster con todos sus elementos base (Apiserver, Proxy, Controller y Scheduler), para este paso revisar el ***anexo 5***.
- Una vez montado el cluster, se deben asignar los permisos correspondientes a cada elemento, teniendo en cuenta que se usará la función de seguridad RBAC (Role-Based Access Control), Para más información ver ***anexo 6***.
- Continuando con el proceso, se debe realizar la configuración del servicio de gitlab que se relaciona con las capas 1 a la 4, y del cual se explicará de forma más detallada en el capítulo 4 de este documento.
- Finalmente, se deben especificar los ingress controller quienes cumplen la función de exposición de los servicios hacia la nube simulada.

Figura 18. Arquitectura del cluster físico



Fuente: Autores

En la figura 18, se muestran todos los componentes correspondientes al cluster para la simulación del modelo, las máquinas virtuales VM1 y VM2 cumpliran el rol de master, las máquinas VM3, VM4, VM5, VM6 son los nodos encargados de ejecutar las tareas, es decir, los nodos worker.

Adicionalmente en la parte superior se observan tres máquinas virtuales adicionales, la VM7 tiene como función ser el deployer o controlador para manejar los elementos del sistema por medio del api de kubernetes, este nodo cuenta con comunicación cifrada y su respectivo role en RBAC. Por otra parte, expone un servicio de DNS desplegado en un contenedor de docker para la comunicación de los nodos master (El dominio es *k8s-master.k8s.local*), este servicio de DNS será el servidor por defecto para todos los nodos, pods y contenedores del cluster.

Finalmente, las máquinas VM8 y VM9 comparten un servicio de NFS para todos los componentes del sistema, razón por la cual, el cluster contará con persistencia para el almacenamiento de los datos, puesto que los contenedores al ser efímeros e inmutables, requieren de esta funcionalidad.

4. MICROSERVICIO COMO CASO DE ESTUDIO

4.1. INTRODUCCIÓN

Dentro de este capítulo el lector puede ver definiciones básicas de los servicios implementados para el funcionamiento de la herramienta seleccionada por los autores como caso de estudio, para mostrar la resiliencia del orquestador de servicios. Además, se detalla información importante sobre la arquitectura y el funcionamiento del servicio dentro del orquestador.

4.2. HERRAMIENTAS UTILIZADAS

Git fue creado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente, es decir Git proporciona las herramientas para desarrollar un trabajo en equipo de manera inteligente y rápida[29].

La herramienta seleccionada como caso de estudio es GitLab, servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto.

Para la implementación y correcto funcionamiento del servicio web GitLab es necesario contar con los siguientes servicios, los cuales son configurados dentro del mismo cluster en donde trabaja Gitlab, de acuerdo con lo visto en el capítulo DEFINICIÓN DEL MODELO BASADO EN KUBERNETES.

4.2.1. Redis

Redis es un data structure store (almacén de estructura de datos) en memoria de código abierto (con licencia BSD), que se utiliza como base de datos, caché y agente de mensajes. Es compatible con las estructuras de datos tales como strings, hash, listas[30].

Para lograr su excelente rendimiento, Redis trabaja con un conjunto de datos en memoria. Dependiendo de su caso de uso, puede persistir volcando el conjunto de datos al disco de vez en cuando, o agregando cada comando a un registro. La

[29] "Qué es Git." https://codigofacilito.com/articulos/que-es-git. Fecha de acceso 23 jul.. 2019.
[30] "Introduction to Redis – Redis." https://redis.io/topics/introduction. Fecha de acceso 23 jul.. 2019.

persistencia se puede deshabilitar opcionalmente, si solo necesita un caché en memoria, en red y con muchas funciones[31].

En esta configuración, Redis no está altamente disponible y representa un único punto de falla. Sin embargo, en este entorno escalado, el objetivo es permitir que el entorno maneje más usuarios o aumente el rendimiento. Redis en sí es generalmente estable y puede manejar muchas solicitudes, por lo que es una compensación aceptable tener solo una instancia[32], para más detalles de la configuación de redis en este proyecto ver **anexo 7**.

4.2.2. PostgreSQL

PostgreSQL es un potente sistema de base de datos relacional de objetos de código abierto que utiliza y amplía el lenguaje SQL combinado con muchas características que almacenan y escalan de forma segura las cargas de trabajo de datos más complicadas. Los orígenes de PostgreSQL se remontan a 1986 como parte del proyecto POSTGRES en la Universidad de California en Berkeley y tiene más de 30 años de desarrollo activo en la plataforma central[33].

PostgreSQL se ha ganado una sólida reputación por su arquitectura probada, confiabilidad, integridad de datos, conjunto de características sólidas, extensibilidad y la dedicación de la comunidad de código abierto detrás del software para ofrecer constantemente soluciones innovadoras y de alto rendimiento.

En esta configuración, PostgreSQL no está altamente disponible y representa un único punto de falla. Sin embargo, cuenta con un volumen persistente, donde se almacenan todas las bases de datos para que, en caso de falla del servicio, sea posible recuperarlo desde un nuevo pod en este entorno escalado, el objetivo es permitir que el entorno maneje más usuarios o aumente el rendimiento. La configuración realizada para la base de datos se encuentra descrita en el **anexo 7**.

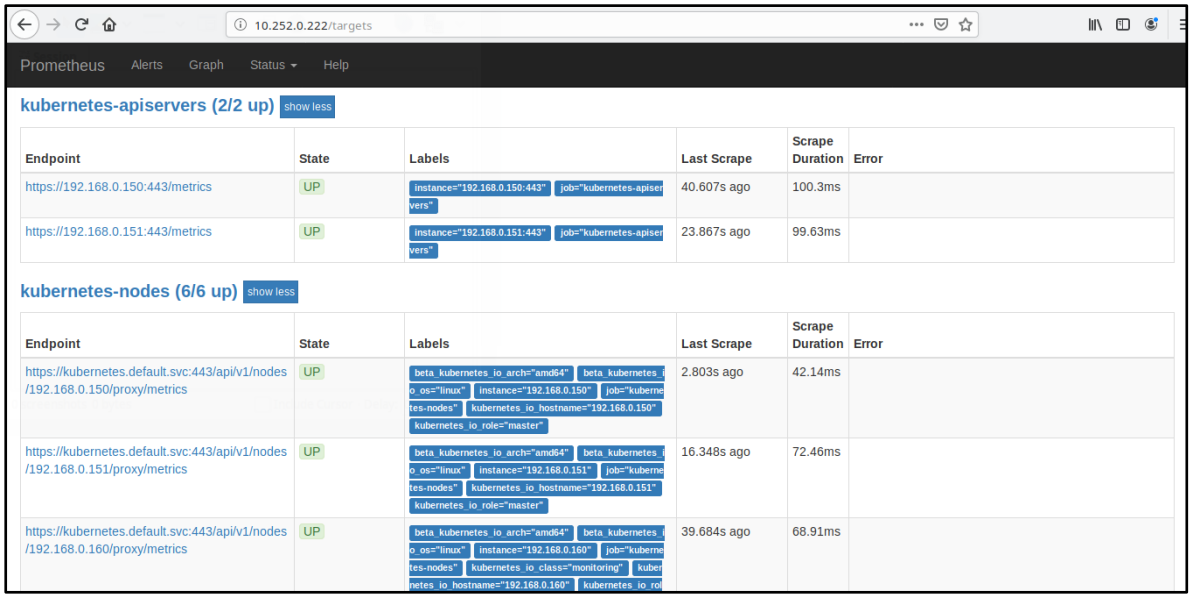
---

[31] "Introduction to Redis – Redis." https://redis.io/topics/introduction. Fecha de acceso 23 jul.. 2019.
[32] "Configuring non-Omnibus Redis for GitLab HA | GitLab." https://docs.gitlab.com/ee/administration/high_availability/redis_source.html. Fecha de acceso 23 jul.. 2019.
[33] "About - PostgreSQL." https://www.postgresql.org/about/. Fecha de acceso 23 jul.. 2019.

4.2.3. Prometheus y Grafana

Prometheus es un conjunto de herramientas de monitoreo y alerta de sistemas de código abierto construido originalmente en SoundCloud. Desde su inicio en 2012, muchas empresas y organizaciones han adoptado Prometheus, y el proyecto tiene una comunidad de desarrolladores y usuarios muy activa. Ahora es un proyecto de código abierto independiente y se mantiene independientemente de cualquier compañía. Para enfatizar esto, y para aclarar la estructura de gobierno del proyecto, Prometheus se unió a la Cloud Native Computing Foundation en 2016 como el segundo proyecto hospedado, después de Kubernetes[34].

Figura 19. Servicio web de prometheus Status>Targets



Fuente: Autores

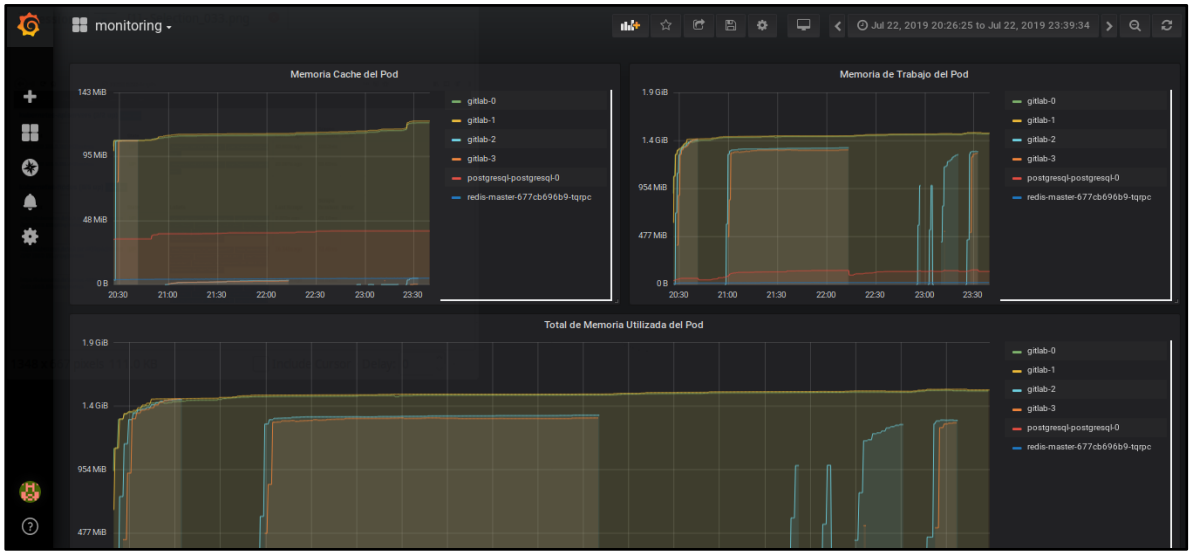
Grafana es la plataforma analítica para todas tus métricas. Grafana le permite consultar, visualizar, alertar y comprender sus métricas sin importar dónde estén almacenadas. Cree, explore y comparta paneles con su equipo y fomente una cultura basada en datos[35].

Grafana y prometheus no son requisito para el funcionamiento de GitLab si es altamente recomendado para un ambiente de producción, debido a que nos

[34] "Overview | Prometheus." https://prometheus.io/docs/introduction/overview/. Fecha de acceso 23 jul.. 2019.
[35] "Features - Grafana." https://grafana.com/grafana. Fecha de acceso 23 jul.. 2019.

permite monitorear de forma gráfica todos los datos capturados por medio del prometheus el cual los obtiene los datos del api de kubernetes.

Figura 20. Dashboard de grafana



Fuente. Autores

Se realizó la implementación de estos servicios, para poder tener plasmado gráficamente el comportamiento de la aplicación**.**

4.2.4. GitLab

Gitlab es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto[36].

Tomamos este servicio como caso de estudio debido a la importancia de un software de control de versiones, actualmente no existe un sistema para el almacenamiento, gestión, versionamiento y control de código que pertenezca completamente a la Universidad Distrital, por tal razón decidimos mostrar los beneficios de un ambiente clusterizado y resiliente con este servicio como caso de estudio.

[36] "GitLab - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/GitLab. Fecha de acceso 23 jul.. 2019.

4.2.4.1. Escalado y alta disponibilidad

GitLab admite varios tipos diferentes de clustering y alta disponibilidad. La solución que elija se basará en el nivel de escalabilidad y disponibilidad que necesite. Las soluciones más fáciles son escalables, pero no necesariamente altamente disponibles.

GitLab proporciona un servicio que generalmente es esencial para la mayoría de las organizaciones: permite a las personas colaborar en el código de manera continua. Por lo tanto, cualquier tiempo de inactividad debe ser corto y planificado. Afortunadamente, GitLab proporciona una configuración sólida incluso en un solo servidor sin medidas especiales. Debido a la naturaleza distribuida de Git, los desarrolladores aún pueden crear código localmente incluso cuando GitLab no está disponible. Sin embargo, algunas funciones de GitLab, como el seguimiento de problemas y la integración continua, no están disponibles cuando GitLab está inactivo.

Tenga en cuenta que todas las soluciones de alta disponibilidad vienen con una compensación entre costo/complejidad y tiempo de actividad. Cuanto más tiempo de actividad desee, más compleja será la solución. Y cuanto más compleja es la solución, más trabajo se requiere para configurarlo y mantenerlo. La alta disponibilidad no es gratuita y cada solución de alta disponibilidad debe equilibrar los costos con los beneficios.

Hay muchas opciones al elegir una arquitectura GitLab de alta disponibilidad. En este proyecto aplicamos la arquitectura de escalado básico.

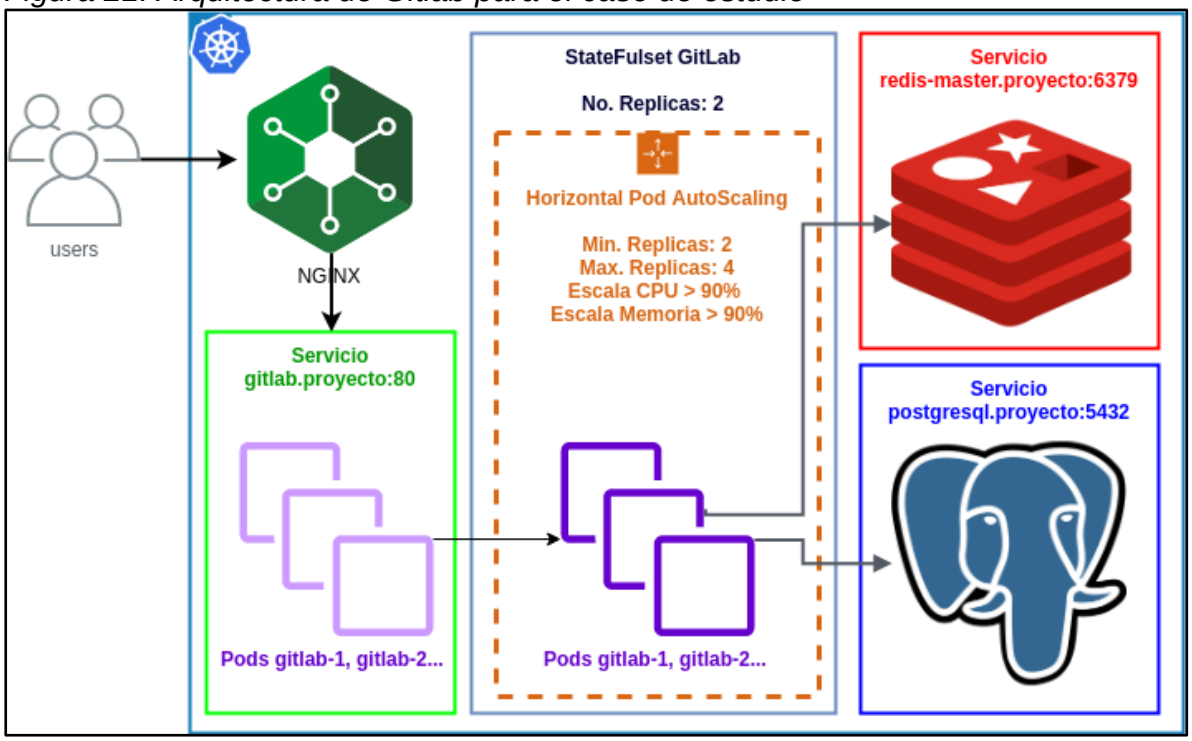
Definida en el sitio web oficial de GitLab como Escalado básico[37].

Esta es la forma más simple de escalar y funcionará para la mayoría de los casos. Los componentes backend, como PostgreSQL, Redis y almacenamiento, se descargan en sus propios nodos, mientras que los componentes restantes de GitLab se ejecutan en 2 o más nodos de aplicación.

En la (Figura 21) se puede observar gráficamente la arquitectura del servicio gitlab implementada dentro del proyecto, orquestado por kubernetes.

[37] "Scaling and High Availability - GitLab Docs." https://docs.gitlab.com/ee/administration/high_availability/. Fecha de acceso 23 jul.. 2019.

Figura 21. Arquitectura de Gitlab para el caso de estudio



Fuente: Autores

4.2.5. Resiliencia

La resilencia en este proyecto esta basada en dos características, la primera es la persistencia de la información, para cumplir con esta propiedad se configuró un servidor NFS, el cual aloja esta en un ambiente diferente al cluster, se recomienta realizar copias de seguridad periódicas de estos datos.

La configuración necesaria en los archivos YAML para configurar un volumen externo es la siguiente.

VolumeMounts ruta donde el pod guarda la información.
Volumes ruta de la carpeta almacenada en el servidor NFS. Ambos enlazados por el nombre **data.**

```
        volumeMounts:
        - mountPath: /bitnami/postgresql
          name: data
volumes:
      - name: data
        nfs:
          path: /mnt/k8s-nfs1/postgres
          server: 192.168.0.180
```

La segunda característica para lograr resiliencia, es la correcta configuración de los objetos de más alta jerarquía, en este caso “Deployment” y “Statefulset”, debido a que esta instanciación permite definir el estado esperado de cada uno de los pods. Ver **Anexo 7. Caso de estudio,** para comprender la configuración enviada a través de los archivos yaml (Ain't Markup Language).

Gracias a las características mencionadas, podemos cumplir con la definición de resiliencia la cual es, “capacidad de un sistema tecnológico de soportar y recuperarse ante desastres y perturbaciones”. Puesto que, por medio de la persistencia de los datos, los pods pueden ser efímeros, recuperándose a través de los objetos de más alta jerarquía (Statefulset y Deployment) sin afectar el estado del servicio, siempre y cuando se aplique la configuración requerida.

4.2.6. HPA – Horizontal Pod Autoscaler

HPA es un objeto de kubernetes que permite definir las directrices para el escalamiento automático de pods. Esta configuración es especificada mediante un archivo yaml y desde allí se pueden tipificar aspectos como número de réplicas mínimas y máximas. Adicionalmente, se configuran los umbrales que actúan como los criterios para que el cluster tome la decisión de aumentar o disminuir el número de pods según demanda.

La forma en como se definió el escalado automático para el caso de estudio se resume en las siguientes líneas

```
apiVersion: autoscaling/v2beta1 #Versión de kubernetes
kind: HorizontalPodAutoscaler   #Tipo de objeto
metadata:
 name: hpa-gitlab               #Nombre del objeto
 namespace: Proyecto            #Espacio de nombres
 labels:
   stack: proyecto
   app: gitlab
spec:
 scaleTargetRef:
   apiVersion: apps/v1beta1
   kind: StatefulSet             #Objeto de referencia para escalar
   name: gitlab                  #Nombre del statefulset
 minReplicas: 2                  #Número mínimo de pods
 maxReplicas: 4                  #Número máximo de pods
 metrics:
   - type: Resource
     resource:
       name: cpu
       targetAverageUtilization: 90    #Umbral de escalamiento CPU
```

```
  - type: Resource
    resource:
      name: memory
      targetAverageUtilization: 90    #Umbral de escalamiento RAM
```

5. PRUEBAS DE STRESS

5.1. INTRODUCCIÓN

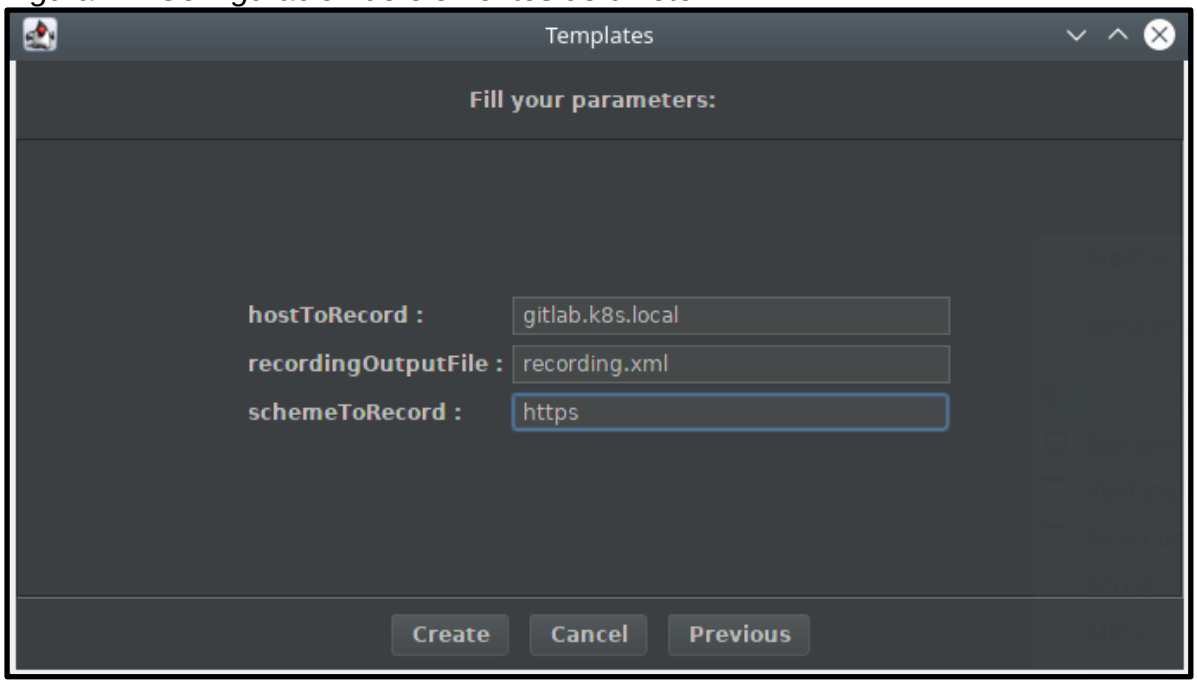
En el quinto y último capítulo de esta investigación se relacionan las pruebas que sustentan que el modelo planteado cumple con los objetivos propuestos inicialmente, teniendo en cuenta los conceptos de resiliencia y autoescalabilidad que se mencionaron desde el inicio del documento.

5.2. Ejecución de las pruebas

Para la ejecución de las pruebas de stress se usó el software apache jmeter, el cual permite realizar la grabación de peticiones, representadas en hilos de configuración para poder replicarlo la cantidad de veces que se requiera.
También se puede configurar el número de usuarios concurrentes y aplicarlo para un flujo en particular que haya sido previamente grabado y capturado por el software en mención.

Teniendo en cuenta este preámbulo, se mostrará la configuración de las pruebas y sus respectivos resultados.

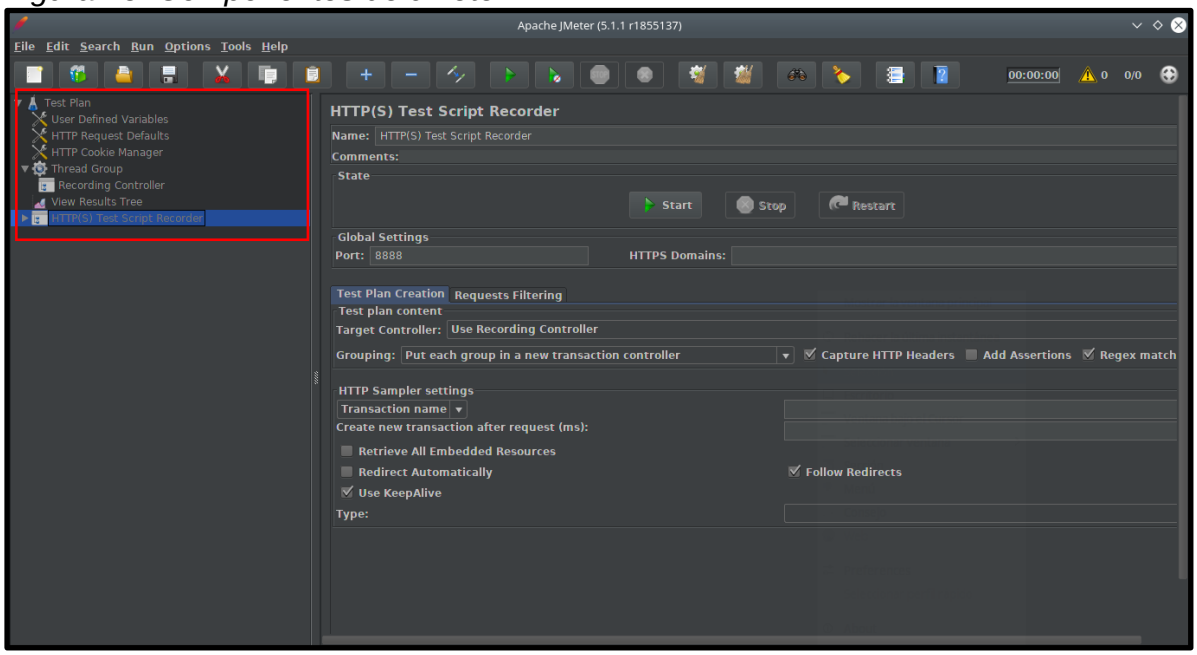
Figura 22. Configuración de elementos de Jmeter



Fuente: Autores

Como se observa en la figura anterior, se envían los parámetros de captura para el software de Jmeter, en este caso el dominio que responderá a las peticiones del usuario es http://gitlab.k8s.local y se almacena en el archivo recording.xml. Y en la figura 23 se muestran los componentes que se crean para la captura de hilos de ejecución.

Figura 23. Componentes de Jmeter



Fuente: Autores

Posteriormente, se hace el levantamiento del proxy de Jmeter y es configurado en Mozilla Firefox como se muestra en la figura 23.

Figura 24. Configuración de Firefox



Fuente Autores

Luego de esto, se procede a hacer la grabación del flujo de trabajo que se quiere plasmar en jmeter. En este caso, dentro de los procesos más relevantes se incluyeron acciones como:

- Login en la aplicación
- Acceso a los repositorios
- Creación de usuarios
- Asignación de permisos
- Acceso al monitoreo de la aplicación
- Visualización general del sistema
- Eliminación de usuarios
- Edición del perfil de usuario
- Entre otros

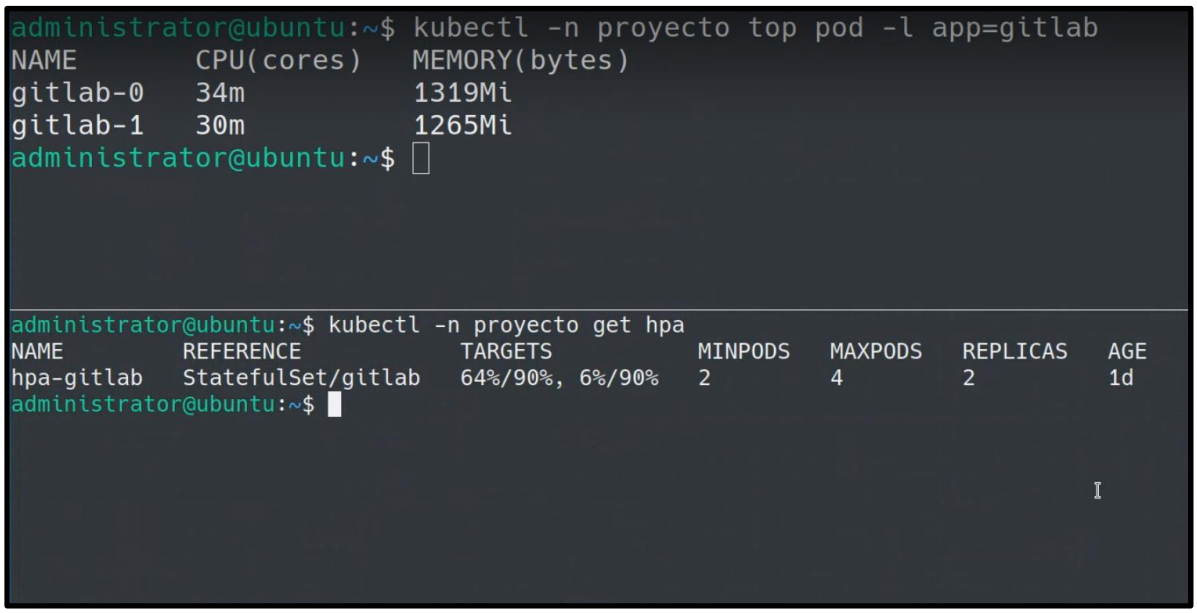
5.3. Conclusiones de las pruebas

El objetivo principal de este proyecto es demostrar que el sistema puede escalar de forma automática de acuerdo a la demanda de peticiones generada en un momento determinado.

En resumen, esta funcionalidad de kubernetes está configurada en los archivos yaml explicados en el ***anexo7***. Debido a que, mediante los objetos hpa, se puede definir el número mínimo de réplicas para un servicio determinado y también se puede especificar bajo qué condiciones el statefulset debe aumentar o disminuir el número de pods en ejecución, teniendo en cuenta la medición constante de cpu y memoria usada por el servicio en cada uno de los nodos del cluster.

En la figura 25 se muestra el estado inicial del statefulset en donde se especifican únicamente 2 réplicas para el tratamiento del servicio de gitlab.

Figura 25. Estado inicial statefulset

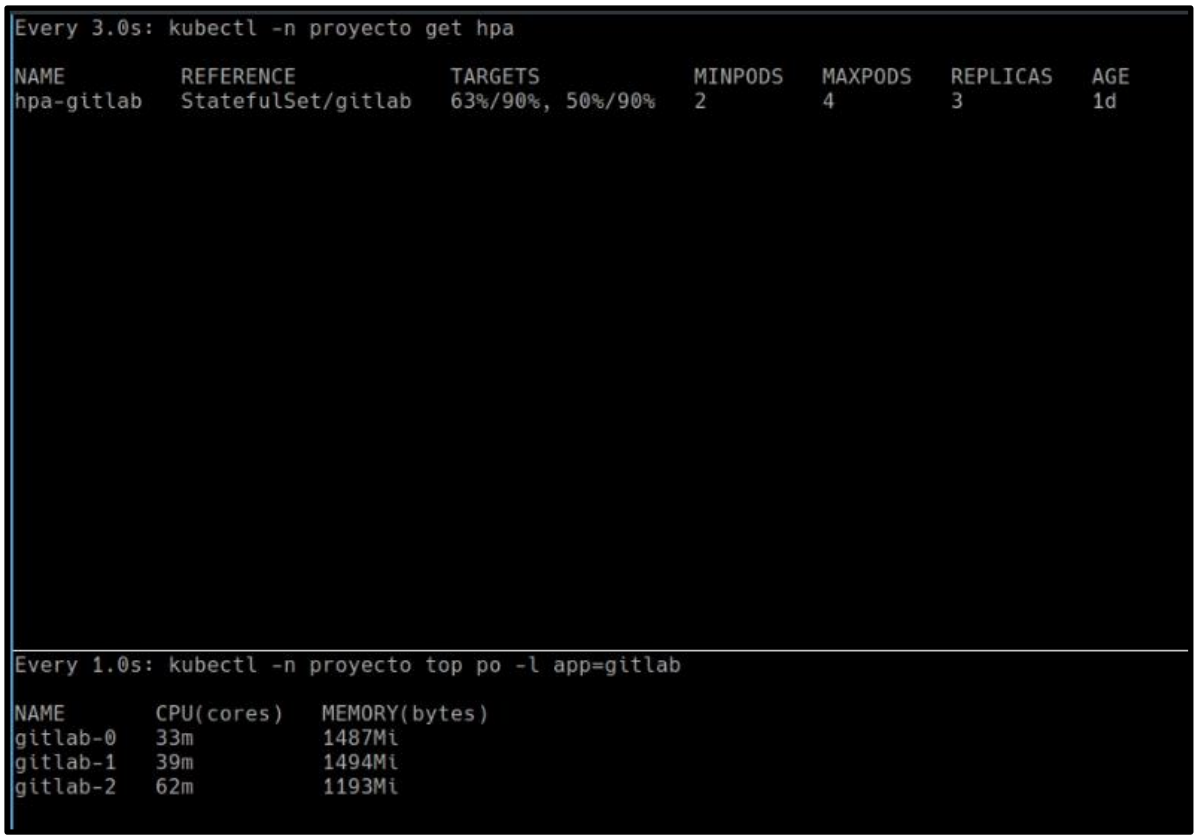


Fuente: Autores

En la ejecución de las pruebas se evidenció que el cluster empezaba a crecer en recursos de forma considerable a partir de 50 usuarios concurrentes repitiendo el flujo previamente establecido; aproximadamente ese es el valor para que el cluster autoescale los pods, teniendo en cuenta que la cpu y la memoria supera el tope establecido en su instanciación inicial. En la figura 26, se muestra la prueba para

50 usuarios concurrentes. y la forma en que se escala automáticamente el cluster para satisfacer la demanda de usuarios.

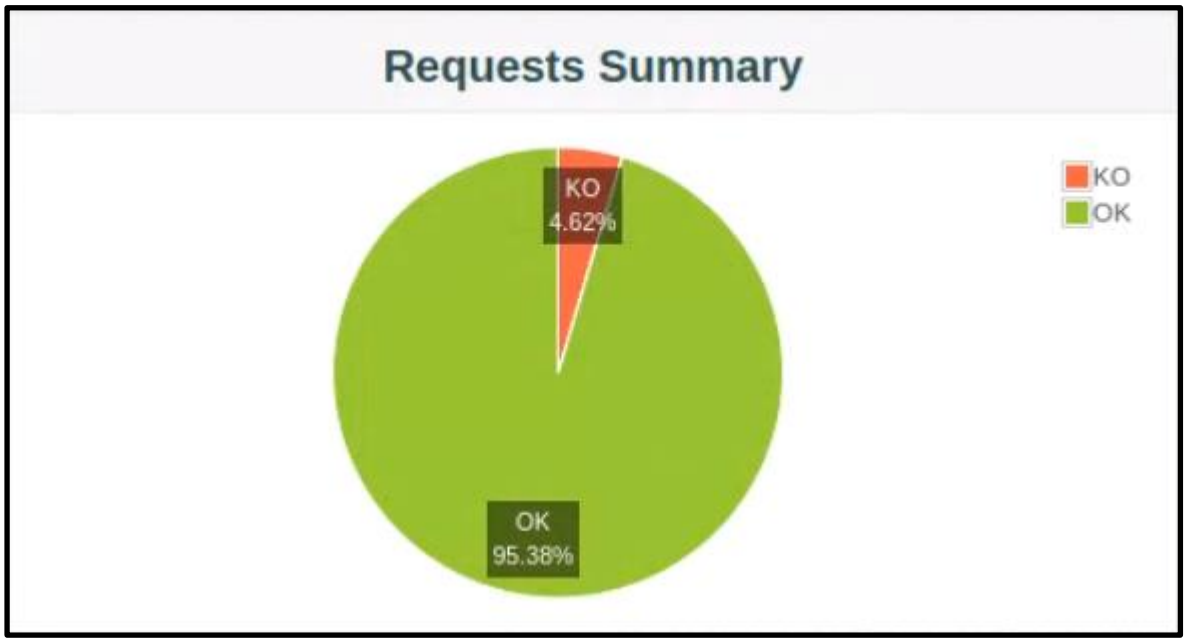
Figura 26. Escalamiento automático



Fuente: Autores

Adicionalmente en la figura 27, se muestra el resultado de esa prueba que indican el porcentaje de fallo de operaciones es satisfactorio en con un valor menor al 10% de fallo para usuarios concurrentes.

Figura 27. Porcentaje de respuesta del servicio



Fuente: *Autores*

El proceso se realizó para 1, 5, 10, 20, 50 y 100 usuarios concurrentes, y el comportamiento presentado fue aproximadamente el mismo. Para verificar todas las respuestas y el comportamiento del cluster revisar el ***anexo8***.

CONCLUSIONES

El proyecto permitió determinar una arquitectura basada en microservicios, que soportara una demanda considerable de usuarios para el uso de una aplicación, en este caso gitlab.

Kubernetes ofrece una amplia gama de componentes para la configuración y personalización en el orquestamiento de microservicios. Lo cual permite soportar diversos sistemas y aplicaciones.

El modelo propuesto en este proyecto es aplicable a cualquier arquitectura, desde infraestructura on premise tradicional, a cualquiera de los proveedores de nube actuales,

La configuración propuesta en esta investigación es adecuada para el cumplimiento de los objetivos planteados inicialmente, sin embargo, puede ser mejorada de diversas formas debido a la maleabilidad de los componentes y la facilidad de integración.

Actualmente hay diversos orquestadores para microservicios, como docker swarm y apache mesos, sin embargo, kubernetes posee mayor experticia en el área y cuenta con el respaldo de marcas a nivel internacional como spotify, huawei y Pokemon Go.

Después del análisis realizado en el documento, en cuanto a los orquestadores actuales, se concluyó que kubernetes ofrece la mayor fiabilidad para el cumplimiento de los objetivos.

La configuración planteada permite escalar los microservicios de forma horizontal, para aumentar la resiliencia cuando la demanda de usuarios así lo requiera.

Por medio de Jmeter se pudo medir el comportamiento del cluster, incrementando la concurrencia de los usuarios y de esta forma demostrar la autonomía de kubernetes para la disponibilidad del servicio.

RECOMENDACIONES

Se recomienda al lector de este documento tener conocimientos básicos en Linux, docker y una base referente al funcionamiento de los microservcios.

Una vez concluida esta tesis, se podría profundizar en:

- Este modelo puede aplicar para cualquier microservicio, teniendo en cuenta que la prioridad de un orquestador es mantener siempre disponible un servicio.
- La infraestructura física para el despliegue de kubernetes, se puede presentar en nubes privadas (On premise, openstack, vmware, KVM, etc), públicas (AWS, GCP, Azure, Rackspace, Digital Ocean, etc) o hibridas; tenga en cuenta, qué para las nubes públicas, este puede ser ya un servicio pre-configurado, por lo cual se recomienda utilizar el servicio del proveedor.
- Tenga en cuenta que este modelo se basa en la demostración de la disponibilidad de un servicio (Gitlab) con base en unos recursos predefinidos, no obstante, esta configuración puede ser extendida a un grado de complejidad más elevado y con una disponibilidad mucho mayor.

BIBLIOGRAFÍA

SUSE. Kubernetes. Nuevo Paradigma de innovación. {en línea}. {Consultado en 2018}. Disponible en: https://www.suse.com/es-es/solutions/kubernetes/

Sino-German Joint Software Institute, School of Computer Science and Engineering, Beihang University. Auto-Scaling Web Applications in Hybrid Cloud Based on Docker. {en línea}. {Consultado en 2018}. Disponible en: https://ieeexplore-ieee-org.bdigital.udistrital.edu.co/document/8070122

Computer science and Engineering National Institute of Technology. EMERGENCY COMMUNICATION SYSTEM WITH DOCKER CONTAINERS, OSM AND RSYNC. {en línea}. {Consultado en 2018}. Disponible en: https://ieeexplore-ieee-org.bdigital.udistrital.edu.co/document/8358533

IEEE/ACM. Adaptive Application Scheduling under Interference in Kubernetes. {en línea}. {Consultado en 2018}. Disponible en: https://ieeexplore-ieee-org.bdigital.udistrital.edu.co/document/7881670

ACM SIGPLAN. Paragon: Qos-aware scheduling for heterogeneous datacenters. {en línea}. {Consultado en 2018}. Disponible en: http://www.csl.cornell.edu/~delimitrou/papers/2013.asplos.paragon.pdf

Google, Inc. NETWORKING IN CONTAINERS AND CONTAINER CLUSTERS. {en línea}. {Consultado en 2018}. Disponible en: https://www.netdevconf.org/0.1/docs/Networking%20in%20Containers%20and%20Container%20Clusters.pdf

UNIVERSIDAD DISTRITAL. DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE GESTIÓN DE ENTORNOS PARA LA OFICINA ASESORA DE SISTEMAS DE LA UNIVERSIDAD DISTRITAL. {en línea}. {Consultado en 2018}. Disponible en: http://repository.udistrital.edu.co/bitstream/11349/6711/1/Proyecto%20de%20Grado.pdf

RED HAT. VIRTUALIZACIÓN. {en línea}. {Consultado en 2018}. Disponible en: https://www.redhat.com/es/topics/virtualization/what-is-virtualization#

SCRUMstudy. A Guide to the SCRUM BODY OF KNOWLEDGE (SBOK™ Guide). {en línea}. {Consultado en 2018}. Disponible en: https://www.scrumstudy.com/SBOK/SCRUMstudy-SBOK-Guide-2016.pdfSUSE.

Kubernetes. Nuevo Paradigma de innovación. {en línea}. {Consultado en 2018}. Disponible en: https://www.suse.com/es-es/solutions/kubernetes/

SCRUM.ORG. The Kanban Guide for Scrum Teams. {en línea}. {Consultado en 2018}. Disponible en: https://scrumorg-website-prod.s3.amazonaws.com/drupal/2018-04/2018%20Kanban%20Guide%20for%20Scrum%20Teams_0.pdf

CASTRO Claudia, Flóres Eva, Capacidad de Orquestación de Servicios Web en las Herramientas MULE ESB y Oracle Service Bus - Tesis de Grado - Lámpsakos julio-diciembre 2015 Medellín–Colombia - 2015

RIVAS, Andrés. Normas ICONTEC para trabajos escritos. {en línea}. {Consultado en 2019}. Disponible en: https://www.colconectada.com/normas-icontec/

GONZÁLEZ, Alberto. Docker Guía práctica. Bogotá Colombia 2018

Kubernetes Documentation. {en línea}. {Consultado en 2019}. Disponible en: https://kubernetes.io/docs/home/

Installing Addons. {en línea}. {Consultado en 2019}. Disponible en: https://kubernetes.io/docs/concepts/cluster-administration/addons/

DevOpskube Blog. List Of Best Docker Container Clustering And Orchestration Tools/Services. {en línea}. {Consultado en 2019}. Disponible en: https://devopscube.com/docker-container-clustering-tools/

CoreOS Documentation. {en línea}. {Consultado en 2019}. Disponible en: https://coreos.com/docs/

GitLab High Availability. {en línea}. {Consultado en 2019}. Disponible en: https://about.gitlab.com/solutions/high-availability/

Postgres Documentation. {en línea}. {Consultado en 2019}. Disponible en: https://www.postgresql.org/docs/

Nginx documentation. {en línea}. {Consultado en 2019}. Disponible en: https://nginx.org/en/docs/

Digital Ocean. How To Configure BIND as a Private Network DNS Server on Ubuntu 14.04. {en línea}. {Consultado en 2019}. Disponible en: https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-ubuntu-14-04

ANEXOS

ANEXO 1 - Instalación CoreOS

Esta guía muestra la forma de instalar CoreOS en máquinas virtuales alojadas en virtualbox.

Acerca de CoreOS

CoreOS Container Linux es el sistema operativo de contenedores líder, diseñado para administrarse y ejecutarse a gran escala, con una sobrecarga operativa mínima. Las aplicaciones con Container Linux se ejecutan en contenedores, proporcionando herramientas amigables para el desarrollador para implementar software.

Container Linux se ejecuta en casi cualquier plataforma, ya sea física, virtual o privada / pública.

Antes de comenzar

1. Se tienen que crear 6 máquinas virtuales en virtualbox. Se recomienda los siguientes recursos:
    - vCPU: 2
    - RAM: 2048 GB
    - HDD: 30 GB
2. Descargue la imagen ISO del sitio web oficial https://coreos.com/os/docs/latest/booting-with-iso.html
3. Proporcione conexión de red a máquinas virtuales, se recomienda usar la opción "Adaptador puente" en virtualbox.

Pasos para la instalación

1. Coloque la imagen ISO en el disco principal de las máquinas deseadas
2. Cuando las máquinas estén listas, ejecute el comando **`sudo openssl passwd -1 > cloud-config-file`**. Luego, escriba y vuelva a escribir la contraseña deseada

3. Edite el archivo de configuración de configuración con el comando **`sudo vim cloud-config-file`** `y` escriba las siguientes líneas

```
#cloud-config
    users:
      - name: username
        passwd: [password]
        groups:
            - sudo
            - docker
```

4. Aplicar la configuración e instalar el sistema operativo con el comando **`sudo coreos-install -d /dev/sda -C stable -c cloud-config-file`**
5. Finalmente, reinicie la máquina virtual.

Nota:

Cuando la máquina virtual está activada, después de reiniciar, verifique la conexión a Internet y la configuración de DNS

ANEXO 2 - Instalación de ETCD

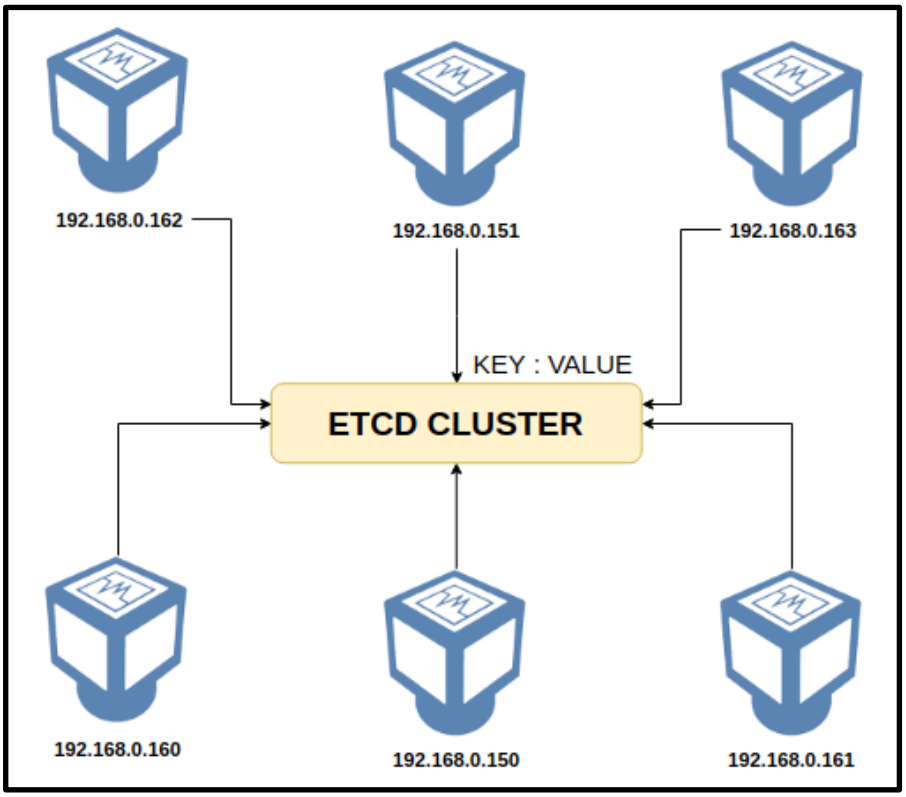
Esta guía muestra la forma de instalar el clúster ETCD.

Definición

Etcd es un almacén de valor de clave distribuida de código abierto que proporciona configuración compartida y descubrimiento de servicios para clústeres de Container Linux. etcd se ejecuta en cada máquina en un grupo y maneja con de forma adecuada la elección del líder durante las particiones de la red y la pérdida de alguna de las máquinas virtuales del cluster.

La siguiente imagen explica gráficamente la configuración de ETCD en el clúster:

Diagrama del cluster de ETCD



Fuente: Autores

Como se observa, se hace uso de las 6 máquinas virtuales configuradas en el ***Anexo 1***, cada una de ellas tiene una dirección IP fija asignada, con la configuración de navegación y DNS correcta.

Direccionamiento

- VM1 (Master 1): 192.168.0.150
- VM2 (Master 2): 192.168.0.151
- VM3 (Worker 1): 192.168.0.160
- VM4 (Worker 2): 192.168.0.161
- VM5 (Worker 3): 192.168.0.162
- VM6 (Worker 4): 192.168.0.163

Prerrequisitos

1. Aprovisionar 6 máquinas virtuales en virtualbox.

2. Instalar CoreOs en cada una de ellas. Ver ***Anexo 1***

3. Verificar la conexión a Internet y la configuración del DNS.

Instalación

Máquina virtual 1

1. Ejecutar: **`sudo mkdir -p /etc/systemd/system/etcd-member.service.d/ && sudo vim /etc/systemd/system/etcd-member.service.d/20-clct-etcd-member.conf`**

   Dentro del archivo ***20-clct-etcd-member.conf*** agregue las siguientes líneas

```
[Service]
Environment="ETCD_DATA_DIR=/var/lib/etcd"
Environment="ETCD_IMAGE_TAG=v3.2.15"
Environment="ETCD_OPTS=--name infra0 \
--initial-advertise-peer-urls http://192.168.0.150:2380 \
--listen-peer-urls http://192.168.0.150:2380 \
--listen-client-urls http://0.0.0.0:2379,http://0.0.0.0:4001 \
--advertise-client-urls http://192.168.0.150:2379 \
--initial-cluster-token etcd-cluster-1 \
```

```
--initial-cluster
infra0=http://192.168.0.150:2380,infra1=http://192.168.0.151:2380,i
nfra2=http://192.168.0.160:2380,infra3=http://192.168.0.161:2380,in
fra4=http://192.168.0.162:2380,infra5=http://192.168.0.163:2380 \
--initial-cluster-state new \
--advertise-client-urls http://192.168.0.150:2379"
```

2. Posteriormente, se debe activar el servicio de acuerdo a la configuración brindada, para ello se debe ejecutar:

- **`systemctl enable etcd-member`** -> Para habilitar el servicio cuando la máquina arranca.
- **`rm -rf /var/lib/etcd`** -> Eliminar datos existentes en este directorio.
- **`sudo systemctl daemon-reload`** -> Para aplicar cambios en los archivos de servicio.
- **`systemctl start etcd-member`** -> Para iniciar el demonio etcd

Máquina virtual 2

1. Ejecutar: **`sudo mkdir -p /etc/systemd/system/etcd-member.service.d/ && sudo vim /etc/systemd/system/etcd-member.service.d/20-clct-etcd-member.conf`**

   Dentro del archivo ***20-clct-etcd-member.conf*** agregue las siguientes líneas

```
[Service]
Environment="ETCD_DATA_DIR=/var/lib/etcd"
Environment="ETCD_IMAGE_TAG=v3.2.15"
Environment="ETCD_OPTS=--name infra1 \
--initial-advertise-peer-urls http://192.168.0.151:2380 \
--listen-peer-urls http://192.168.0.151:2380 \
--listen-client-urls http://0.0.0.0:2379,http://0.0.0.0:4001 \
--advertise-client-urls http://192.168.0.151:2379 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster
infra0=http://192.168.0.150:2380,infra1=http://192.168.0.151:2380,i
nfra2=http://192.168.0.160:2380,infra3=http://192.168.0.161:2380,in
fra4=http://192.168.0.162:2380,infra5=http://192.168.0.163:2380 \
--initial-cluster-state new \
--advertise-client-urls http://192.168.0.151:2379"
```

2. Procedemos a habilitar el servicio ejecutando:

- **`systemctl enable etcd-member`**

- **rm -rf /var/lib/etcd**
- **sudo systemctl daemon-reload**
- **systemctl start etcd-member**

Máquina virtual 3

1. Ejecutar: **sudo mkdir -p /etc/systemd/system/etcd-member.service.d/ && sudo vim /etc/systemd/system/etcd-member.service.d/20-clct-etcd-member.conf**

   Dentro del archivo ***20-clct-etcd-member.conf*** ponga las siguientes líneas

```
[Service]
Environment="ETCD_DATA_DIR=/var/lib/etcd"
Environment="ETCD_IMAGE_TAG=v3.2.15"
Environment="ETCD_OPTS=--name infra2 \
--initial-advertise-peer-urls http://192.168.0.160:2380 \
--listen-peer-urls http://192.168.0.160:2380 \
--listen-client-urls http://0.0.0.0:2379,http://0.0.0.0:4001 \
--advertise-client-urls http://192.168.0.160:2379 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster
infra0=http://192.168.0.150:2380,infra1=http://192.168.0.151:2380,i
nfra2=http://192.168.0.160:2380,infra3=http://192.168.0.161:2380,in
fra4=http://192
.168.0.162:2380,infra5=http://192.168.0.163:2380 \
--initial-cluster-state new \
--advertise-client-urls http://192.168.0.160:2379"
```

2. Al igual que en las máquinas anteriores ejecutar:

- **systemctl enable etcd-member**
- **rm -rf /var/lib/etcd**
- **sudo systemctl daemon-reload**
- **systemctl start etcd-member**

Máquina virtual 4

3. Ejecutar: **sudo mkdir -p /etc/systemd/system/etcd-member.service.d/ && sudo vim /etc/systemd/system/etcd-member.service.d/20-clct-etcd-member.conf**

   Dentro del archivo ***20-clct-etcd-member.conf*** ponga las siguientes líneas

```
[Service]
Environment="ETCD_DATA_DIR=/var/lib/etcd"
Environment="ETCD_IMAGE_TAG=v3.2.15"
Environment="ETCD_OPTS=--name infra3 \
--initial-advertise-peer-urls http://192.168.0.161:2380 \
--listen-peer-urls http://192.168.0.161:2380 \
--listen-client-urls http://0.0.0.0:2379,http://0.0.0.0:4001 \
--advertise-client-urls http://192.168.0.161:2379 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster
infra0=http://192.168.0.150:2380,infra1=http://192.168.0.151:2380,i
nfra2=http://192.168.0.160:2380,infra3=http://192.168.0.161:2380,in
fra4=http://192.168.0.162:2380,infra5=http://192.168.0.163:2380 \
--initial-cluster-state new \
--advertise-client-urls http://192.168.0.161:2379"
```

4. Para habilitar el servicio, ejecutar:

- **systemctl enable etcd-member**
- **rm -rf /var/lib/etcd**
- **sudo systemctl daemon-reload**
- **systemctl start etcd-member**

Máquina virtual 5

5. Ejecutar: **sudo mkdir -p /etc/systemd/system/etcd-member.service.d/ && sudo vim /etc/systemd/system/etcd-member.service.d/20-clct-etcd-member.conf**

   Dentro del archivo ***20-clct-etcd-member.conf*** ponga las siguientes líneas

```
[Service]
Environment="ETCD_DATA_DIR=/var/lib/etcd"
Environment="ETCD_IMAGE_TAG=v3.2.15"
Environment="ETCD_OPTS=--name infra4 \
--initial-advertise-peer-urls http://192.168.0.162:2380 \
```

```
--listen-peer-urls http://192.168.0.162:2380 \
--listen-client-urls http://0.0.0.0:2379,http://0.0.0.0:4001 \
--advertise-client-urls http://192.168.0.162:2379 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster
infra0=http://192.168.0.150:2380,infra1=http://192.168.0.151:2380,i
nfra2=http://192.168.0.160:2380,infra3=http://192.168.0.161:2380,in
fra4=http://192.168.0.162:2380,infra5=http://192.168.0.163:2380 \
--initial-cluster-state new \
--advertise-client-urls http://192.168.0.162:2379"
```

6. Para habilitar el servicio, ejecutar:

- **systemctl enable etcd-member**
- **rm -rf /var/lib/etcd**
- **sudo systemctl daemon-reload**
- **systemctl start etcd-member**

Máquina virtual 6

7. Ejecutar: **sudo mkdir -p /etc/systemd/system/etcd-member.service.d/ && sudo vim /etc/systemd/system/etcd-member.service.d/20-clct-etcd-member.conf**

   Dentro del archivo ***20-clct-etcd-member.conf*** ponga las siguientes líneas

```
[Service]
Environment="ETCD_DATA_DIR=/var/lib/etcd"
Environment="ETCD_IMAGE_TAG=v3.2.15"
Environment="ETCD_OPTS=--name infra5 \
--initial-advertise-peer-urls http://192.168.0.163:2380 \
--listen-peer-urls http://192.168.0.163:2380 \
--listen-client-urls http://0.0.0.0:2379,http://0.0.0.0:4001 \
--advertise-client-urls http://192.168.0.163:2379 \
--initial-cluster-token etcd-cluster-1 \
--initial-cluster
infra0=http://192.168.0.150:2380,infra1=http://192.168.0.151:2380,i
nfra2=http://192.168.0.160:2380,infra3=http://192.168.0.161:2380,in
fra4=http://192.168.0.162:2380,infra5=http://192.168.0.163:2380 \
--initial-cluster-state new \
--advertise-client-urls http://192.168.0.163:2379"
```

8. Finalmente, al igual que en las máquinas anteriores ejecutar:

- **systemctl enable etcd-member**

- **`rm -rf /var/lib/etcd`**
- **`sudo systemctl daemon-reload`**
- **`systemctl start etcd-member`**

Prueba del servicio

- Con el comando **`etcdctl member list`**, los nodos del clúster obtienen la siguiente salida.

Miembros del cluster de etcd

```
coreos@k8s-master-1 ~ $ etcdctl member list
6adc6246f910122: name=infra2 peerURLs=http://192.168.0.160:2380 clientURLs=http://192.168.0.160:2379 isLeader=false
271210ddaa455f62: name=infra1 peerURLs=http://192.168.0.151:2380 clientURLs=http://192.168.0.151:2379 isLeader=false
49c8de2bea7da45f: name=infra0 peerURLs=http://192.168.0.150:2380 clientURLs=http://192.168.0.150:2379 isLeader=false
6ed1823fed8155b7: name=infra3 peerURLs=http://192.168.0.161:2380 clientURLs=http://192.168.0.161:2379 isLeader=true
ef633c4e1c5f9daf: name=infra4 peerURLs=http://192.168.0.162:2380 clientURLs=http://192.168.0.162:2379 isLeader=false
efe4a7ae36d10042: name=infra5 peerURLs=http://192.168.0.163:2380 clientURLs=http://192.168.0.163:2379 isLeader=false
coreos@k8s-master-1 ~ $
```

Fuente: Autores

- Una vez los 6 nodos se encuentre activos, se puede crear una clave de prueba en VM1; para esto, se usa el comando **`etcdctl set key1 prueba`**
- Y luego, se puede consultar la misma clave en otros nodos del cluster (VM2, VM3) **`etcdctl get key1`**
- Finalmente, se puede apagar cualquier máquina virtual, incluso el master y se puede ver el comportamiento del clúster. con el comando **`etcdctl member list`** O **`etcdctl cluster-health`**

Salud del cluster de etcd

```
coreos@k8s-master-1 ~ $ etcdctl cluster-health
member 6adc6246f910122 is healthy: got healthy result from http://192.168.0.160:2379
member 271210ddaa455f62 is healthy: got healthy result from http://192.168.0.151:2379
member 49c8de2bea7da45f is healthy: got healthy result from http://192.168.0.150:2379
member 6ed1823fed8155b7 is healthy: got healthy result from http://192.168.0.161:2379
member ef633c4e1c5f9daf is healthy: got healthy result from http://192.168.0.162:2379
member efe4a7ae36d10042 is healthy: got healthy result from http://192.168.0.163:2379
cluster is healthy
coreos@k8s-master-1 ~ $
```

Fuente: Autores

ANEXO 3 - Configuración de la red - Masters

Los dos nodos master se debe hace la configuración del CNI requerido para el funcionamiento del cluster, en este caso se usará Flannel, para ello ejecutar los comandos especificados a continuación.

```
sudo mkdir -p /etc/flannel/
sudo vim /etc/flannel/options.env
```

En el archivo colocar las siguientes líneas para el **Master 1**

```
FLANNELD_IFACE=192.168.0.150
FLANNELD_ETCD_ENDPOINTS=http://192.168.0.150:2379,http://192.168.0.151:23
79,http://192.168.0.160:2379,http://192.168.0.161:2379,http://192.168.0.1
62:2379,http://192.168.0.163:2379
```

Y para el **Master 2** colocar las siguientes líneas

```
FLANNELD_IFACE=192.168.0.151
FLANNELD_ETCD_ENDPOINTS=http://192.168.0.150:2379,http://192.168.0.151:23
79,http://192.168.0.160:2379,http://192.168.0.161:2379,http://192.168.0.1
62:2379,http://192.168.0.163:2379
```

Para crear el servicio, se deben ejecutar los siguientes comandos en los **dos nodos Master**

```
sudo mkdir -p /etc/systemd/system/flanneld.service.d/
sudo vim /etc/systemd/system/flanneld.service.d/40-ExecStartPre-
symlink.conf
```

En el archivo colocar las siguientes líneas:

```
[Service]
ExecStartPre=/usr/bin/ln -sf /etc/flannel/options.env
/run/flannel/options.env
```

Configuración de Docker – Masters

Para que flannel administre la red de pod en el clúster, Docker debe configurarse para que permita la administración desde el CNI elegido. Todo lo que se debe hacer es cambiar la prioridad de ejecución de flannel antes que Docker. Para crear el servicio ejecutar los siguientes comandos:

```
sudo mkdir -p /etc/systemd/system/docker.service.d/
sudo vim /etc/systemd/system/docker.service.d/40-flannel.conf
```

En el archivo colocar las siguientes líneas:

```
[Unit]
Requires=flanneld.service
After=flanneld.service
[Service]
EnvironmentFile=/etc/kubernetes/cni/docker_opts_cni.env
```

Ahora se debe crear el archivo de opciones CNI de Docker, con los siguientes comandos:

```
sudo mkdir /etc/kubernetes/cni/ -p
sudo vim /etc/kubernetes/cni/docker_opts_cni.env
```

En el archivo coloque las siguientes líneas

```
DOCKER_OPT_BIP=""
DOCKER_OPT_IPMASQ=""
```

Ahora se debe realizar la configuración CNI de flannel, para ello se deben aplicar los siguientes pasos

```
sudo mkdir /etc/kubernetes/cni/net.d/ -p
sudo vim /etc/kubernetes/cni/net.d/10-flannel.conf
```

En el archivo de configuración coloque las siguientes líneas

```
{
    "name": "podnet",
    "type": "flannel",
    "delegate": {
        "isDefaultGateway": true
    }
}
```

Configurar la red de flannel en ETCD

Se debe registrar la red de pods en el cluster de etcs, para ello ejecutar

```
curl -X PUT -d
"value={\"Network\":\"10.253.0.0/16\",\"Backend\":{\"Type\":\"vxlan\"}}"
"http://192.168.1.224:2379/v2/keys/coreos.com/network/config"
```

Para que los cambios sean aplicados se debe recargar el demonio de flannel e iniciar los servicios, para ello aplicar los siguientes comandos:

```
sudo systemctl daemon-reload
sudo systemctl start flanneld
sudo systemctl enable flanneld
```

Configuración de la red – Workers

Para configurar la red en los nodos workers se debe configurar de la siguiente manera

```
sudo mkdir -p /etc/flannel/
sudo vim /etc/flannel/options.env
```

En el archivo colocar las siguientes líneas para el **Worker 1**

```
FLANNELD_IFACE=192.168.0.160
FLANNELD_ETCD_ENDPOINTS=http://192.168.0.150:2379,http://192.168.0.151:23
79,http://192.168.0.160:2379,http://192.168.0.161:2379,http://192.168.0.1
62:2379,http://192.168.0.163:2379
```

En el archivo colocar las siguientes líneas para el **Worker 2**

```
FLANNELD_IFACE=192.168.0.161
FLANNELD_ETCD_ENDPOINTS=http://192.168.0.150:2379,http://192.168.0.151:23
79,http://192.168.0.160:2379,http://192.168.0.161:2379,http://192.168.0.1
62:2379,http://192.168.0.163:2379
```

En el archivo colocar las siguientes líneas para el **Worker 3**

```
FLANNELD_IFACE=192.168.0.162
```

```
FLANNELD_ETCD_ENDPOINTS=http://192.168.0.150:2379,http://192.168.0.151:23
79,http://192.168.0.160:2379,http://192.168.0.161:2379,http://192.168.0.1
62:2379,http://192.168.0.163:2379
```

En el archivo colocar las siguientes líneas para el **Worker 4**

```
FLANNELD_IFACE=192.168.0.163
FLANNELD_ETCD_ENDPOINTS=http://192.168.0.150:2379,http://192.168.0.151:23
79,http://192.168.0.160:2379,http://192.168.0.161:2379,http://192.168.0.1
62:2379,http://192.168.0.163:2379
```

Para crear el servicio, se deben ejecutar los siguientes comandos en los **todos los nodos worker**

```
sudo mkdir -p /etc/systemd/system/flanneld.service.d/
sudo vim /etc/systemd/system/flanneld.service.d/40-ExecStartPre-
symlink.conf
```

En el archivo colocar las siguientes líneas:

```
[Service]
ExecStartPre=/usr/bin/ln -sf /etc/flannel/options.env
/run/flannel/options.env
```

Configuración de Docker – Workers

Al igual que en los nodos maestros, Docker debe configurarse para que permita la administración desde el CNI elegido. Para crear el servicio ejecutar los siguientes comandos:

```
sudo mkdir -p /etc/systemd/system/docker.service.d/
sudo vim /etc/systemd/system/docker.service.d/40-flannel.conf
```

En el archivo colocar las siguientes líneas:

```
[Unit]
Requires=flanneld.service
After=flanneld.service
[Service]
EnvironmentFile=/etc/kubernetes/cni/docker_opts_cni.env
```

Ahora se debe crear el archivo de opciones CNI de Docker, con los siguientes comandos:

```
sudo mkdir /etc/kubernetes/cni/ -p
sudo vim /etc/kubernetes/cni/docker_opts_cni.env
```

En el archivo coloque las siguientes líneas

```
DOCKER_OPT_BIP=""
DOCKER_OPT_IPMASQ=""
```

Ahora se debe realizar la configuración CNI de flannel, para ello se deben aplicar los siguientes pasos

```
sudo mkdir /etc/kubernetes/cni/net.d/ -p
sudo vim /etc/kubernetes/cni/net.d/10-flannel.conf
```

En el archivo de configuración coloque las siguientes líneas

```
{
    "name": "podnet",
    "type": "flannel",
    "delegate": {
        "isDefaultGateway": true
    }
}
```

Para que los cambios sean aplicados se debe recargar el demonio de flannel e iniciar los servicios, para ello aplicar los siguientes comandos:

```
sudo systemctl daemon-reload
sudo systemctl start flanneld
sudo systemctl enable flanneld
```

Como resultado en todos los nodos, tanto workers como masters, se debe iniciar un contenedor en rkt que cree las reglas de enrutamiento hacia las otras redes interactuantes.

Estado del contenedor de flanneld

```
k8s-worker-1 coreos # rkt list
UUID            APP             IMAGE NAME                                      STATE
09418bf3        flannel         quay.io/coreos/flannel:v0.11.0                  running
0d89439e        hyperkube       quay.io/coreos/hyperkube:v1.10.5_coreos.0       exited garbage
267f28d3        hyperkube       quay.io/coreos/hyperkube:v1.10.5_coreos.0       exited garbage
5693ac1f        hyperkube       quay.io/coreos/hyperkube:v1.10.5_coreos.0       exited garbage
85f3cd82        etcd            quay.io/coreos/etcd:v3.2.15                     running
cb9fa111        hyperkube       quay.io/coreos/hyperkube:v1.10.5_coreos.0       running
cf4b6856        hyperkube       quay.io/coreos/hyperkube:v1.10.5_coreos.0       exited garbage
d2a8a309        hyperkube       quay.io/coreos/hyperkube:v1.10.5_coreos.0       exited garbage
efc225e4        flannel         quay.io/coreos/flannel:v0.11.0                  exited
f3413200        hyperkube       quay.io/coreos/hyperkube:v1.10.5_coreos.0       exited garbage
```

Fuente: Autores

Interfaz virtual de red flannel

```
flannel.1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 10.253.92.0  netmask 255.255.255.255  broadcast 0.0.0.0
        inet6 fe80::a0e6:5ff:fe56:751d  prefixlen 64  scopeid 0x20<link>
        ether a2:e6:05:56:75:1d  txqueuelen 0  (Ethernet)
        RX packets 447177  bytes 996836931 (950.6 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 363706  bytes 27651028 (26.3 MiB)
        TX errors 0  dropped 17 overruns 0  carrier 0  collisions 0
```

Fuente: Autores

ANEXO 4 - Generación de certificados SSL

Un certificado SSL (Secure Sockets Layer) es un título digital que autentifica la identidad de un sitio web y cifra con tecnología SSL la información que se envía al servidor. El cifrado es el proceso de mezclar datos en un formato indescifrable que sólo puede volver al formato legible con la clave de descifrado adecuada.

Un certificado sirve como un identificador electrónico que establece las credenciales de una entidad en línea al hacer negocios en la Web. Cuando un usuario de Internet intenta enviar información de credenciales a un servidor web, el navegador del usuario accede al certificado digital del servidor y establece una conexión segura[38].

Un certificado SSL contiene la siguiente información:

- El nombre del titular del certificado
- El número de serie del certificado y la fecha de vencimiento
- Una copia de la clave pública del titular del certificado
- La firma digital de la autoridad que emite el certificado o CA

Para este proyecto es importante tener claro que la información dentro del cluster viaja cifrada, por lo cual se requiere la utilización de certificados SSL para establecer comunicación entre los nodos que intervienen. Por ello, se deben generar certificados para el Nodo Master, los nodos Workers y el administrador del clúster o Deployer.

CREACIÓN DE CA (AUTORIDAD CERTIFICADORA)

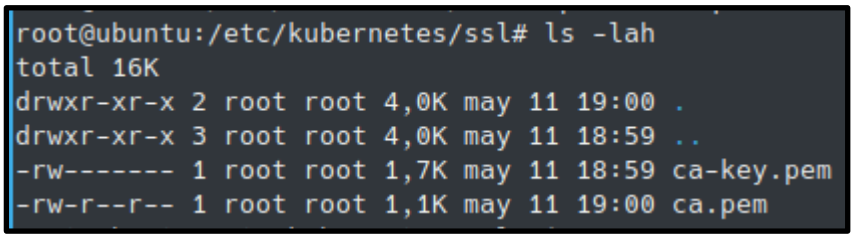
Se deben crear dos llaves, una pública y otra privada. Para ello se deben ejecutar los siguientes comandos:

```
openssl genrsa -out ca-key.pem 2048
openssl req -x509 -new -nodes -key ca-key.pem -days 365 -out ca.pem
-subj "/CN=kube-ca"
```

con esto, se obtiene la autoridad certificadora de la siguiente manera:

[38] "¿Qué es un Certificado SSL? | Certificados SSL - Ayuda de GoDaddy ES." https://es.godaddy.com/help/que-es-un-certificado-ssl-542. Fecha de acceso 11 may.. 2019.

LIaves privada y pública de la CA.



Fuente: Autores

CERTIFICADOS PARA EL APISERVER

Se debe realizar la creación de un archivo llamado openssl.cnf, el cual contiene la información del host autorizados para los certificados requeridos, en él, se deben añadir las siguientes líneas:

```
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = kubernetes
DNS.2 = kubernetes.default
DNS.3 = kubernetes.default.svc
DNS.4 = kubernetes.default.svc.cluster.local
DNS.5 = k8s-master.k8s.local
IP.1 = ${K8S_SERVICE_IP}
```

Después de esto se deben ejecutar los siguientes comandos para generar las llaves del apiserver

```
K8S_SERVICE_IP=10.252.0.1 envsubst < openssl.cnf > temp.cnf

openssl genrsa -out apiserver-key.pem 2048

openssl req -new -key apiserver-key.pem -out apiserver.csr -subj
"/CN=kube-apiserver" -config temp.cnf

openssl x509 -req -in apiserver.csr -CA ca.pem -CAkey ca-key.pem -
CAcreateserial -out apiserver.pem -days 365 -extensions v3_req -
extfile temp.cnf
```

CERTIFICADOS PARA LOS NODOS WORKER

Para el certificado del **Nodo Worker 1** ejecutar los siguientes comandos:

```
WORKER_IP=192.168.0.160 envsubst < worker-openssl.cnf >
worker_temp.cnf

openssl genrsa -out k8s-worker1-key.pem 2048

WORKER_IP=192.168.0.160 openssl req -new -key k8s-worker1-key.pem -
out k8s-worker1.csr -subj "/CN=k8s-worker1" -config worker_temp.cnf

WORKER_IP=192.168.0.160 openssl x509 -req -in k8s-worker1.csr -CA
ca.pem -CAkey ca-key.pem -CAcreateserial -out k8s-worker1.pem -days
365 -extensions v3_req -extfile worker_temp.cnf
```

Para el certificado del **Nodo Worker 2** ejecutar los siguientes comandos:

```
WORKER_IP=192.168.0.161 envsubst < worker-openssl.cnf >
worker_temp.cnf

openssl genrsa -out k8s-worker2-key.pem 2048

WORKER_IP=192.168.0.161 openssl req -new -key k8s-worker2-key.pem -
out k8s-worker2.csr -subj "/CN=k8s-worker2" -config worker_temp.cnf

WORKER_IP=192.168.0.161 openssl x509 -req -in k8s-worker2.csr -CA
ca.pem -CAkey ca-key.pem -CAcreateserial -out k8s-worker2.pem -days
365 -extensions v3_req -extfile worker_temp.cnf
```

Para el certificado del **Nodo Worker 3** ejecutar los siguientes comandos:

```
WORKER_IP=192.168.0.162 envsubst < worker-openssl.cnf >
worker_temp.cnf

openssl genrsa -out k8s-worker3-key.pem 2048

WORKER_IP=192.168.0.162 openssl req -new -key k8s-worker3-key.pem -
out k8s-worker3.csr -subj "/CN=k8s-worker3" -config worker_temp.cnf

WORKER_IP=192.168.0.162 openssl x509 -req -in k8s-worker3.csr -CA
ca.pem -CAkey ca-key.pem -CAcreateserial -out k8s-worker3.pem -days
365 -extensions v3_req -extfile worker_temp.cnf
```

Para el certificado del **Nodo Worker 4** ejecutar los siguientes comandos:

```
WORKER_IP=192.168.0.163 envsubst < worker-openssl.cnf > worker_temp.cnf

openssl genrsa -out k8s-worker4-key.pem 2048

WORKER_IP=192.168.0.163 openssl req -new -key k8s-worker4-key.pem -out k8s-worker4.csr -subj "/CN=k8s-worker4" -config worker_temp.cnf

WORKER_IP=192.168.0.163 openssl x509 -req -in k8s-worker4.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out k8s-worker4.pem -days 365 -extensions v3_req -extfile worker_temp.cnf
```

CERTIFICADOS PARA EL DEPLOYER

Estos certificados se usaron para poder hacer la administración del cluster de kubernetes desde una máquina externa (deployer). Para generar los certificados de este, se deben ejecutar los siguientes comandos:

```
openssl genrsa -out admin-key.pem 2048

hostnamectl set-hostname ubuntu

openssl req -new -key admin-key.pem -out admin.csr -subj "/CN=ubuntu"

openssl x509 -req -in admin.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out admin.pem -days 365
```

UBICACIÓN DE LOS ARCHIVOS

Nodos Master

Crear un directorio para alojar los archivos requeridos

```
sudo mkdir -p /etc/kubernetes/ssl
```

Copiar los archivos del master en la carpeta previamente creada:

- Archivo: `/etc/kubernetes/ssl/ca.pem`
- Archivo: `/etc/kubernetes/ssl/apiserver.pem`
- Archivo: `/etc/kubernetes/ssl/apiserver-key.pem`

Luego, asegúrese de haber establecido los permisos y propietarios adecuados para las llaves:

```
sudo chmod 600 /etc/kubernetes/ssl/*-key.pem
```

```
sudo chown root:root /etc/kubernetes/ssl/*-key.pem
```

Nodos Worker

Crear un directorio para alojar los archivos requeridos

```
sudo mkdir -p /etc/kubernetes/ssl
```

Copiar los archivos del master en la carpeta previamente creada:

- Archivo: `/etc/kubernetes/ssl/ca.pem`
- Archivo: `/etc/kubernetes/ssl/k8s-worker(1,2,3,4).pem`
- Archivo: `/etc/kubernetes/ssl/k8s-worker-key(1,2,3,4).pem`

Luego, asegúrese de haber establecido los permisos y propietarios adecuados para las llaves:

```
sudo chmod 600 /etc/kubernetes/ssl/*-key.pem
sudo chown root:root /etc/kubernetes/ssl/*-key.pem
```

Finalmente, se deben crear enlaces simbólicos con el fin de que no se afecte la estructura de configuración para kubernetes planteada en el ***anexo 5***

```
sudo ln -s /etc/kubernetes/ssl/k8s-worker(1,2,3,4).pem
/etc/kubernetes/ssl/worker.pem
sudo ln -s /etc/kubernetes/ssl/k8s-worker-key(1,2,3,4).pem
/etc/kubernetes/ssl/worker-key.pem
```

Nodo Deployer (kubectl)

- Reemplace ${MASTER_HOST} con la dirección del nodo maestro o el nombre utilizado en los pasos anteriores (*k8s-master.k8s.local*)
- Reemplace ${CA_CERT} con la ruta absoluta al ca.pem creado en los pasos anteriores
- Reemplace ${ADMIN_KEY} con la ruta absoluta al admin-key.pem creado en los pasos anteriores
- Reemplace ${ADMIN_CERT} con la ruta absoluta al admin.pem creado en los pasos anteriores

```
kubectl config set-cluster default-cluster --server=https://${MASTER_HOST} --certificate-authority=${PATH_CA_CERT}

kubectl config set-credentials default-admin --certificate-authority=${PATH_CA_CERT} --client-key=${PATH_ADMIN_KEY} --client-certificate=${PATH_ADMIN_CERT}

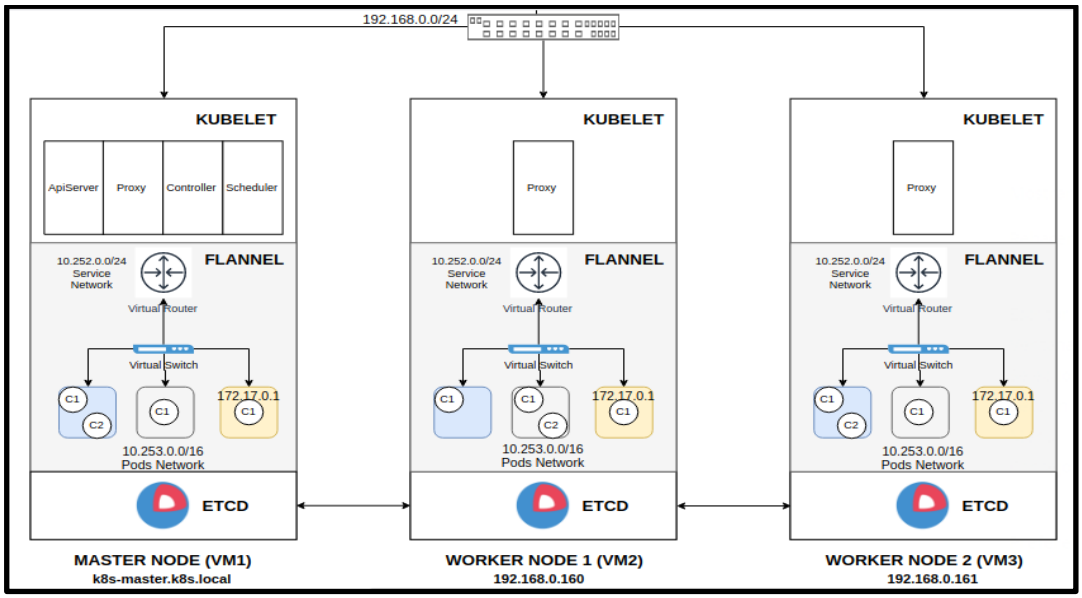
kubectl config set-context default-system --cluster=default-cluster --user=default-admin

kubectl config use-context default-system
```

ANEXO 5 - Configuración de Kubernetes

Para este punto, ya deben estar configurados los componentes esenciales para montar un cluster de kubernetes, para ello ver **anexo1**, **anexo2**, **anexo3** y **anexo4**.

Funcionamiento de componentes de kubernetes



Fuente: Autores

Configuración en nodos maestros

En primer lugar se debe configurar el servicio de **kubelet**, para ello se debe crear el archivo:

```
sudo vim /etc/systemd/system/kubelet.service
```

Y añadir el siguiente contenido:

```
[Service]
Environment=KUBELET_IMAGE_TAG=v1.10.5_coreos.0
Environment="RKT_RUN_ARGS=--uuid-file-save=/var/run/kubelet-
pod.uuid \
  --volume dns,kind=host,source=/etc/resolv.conf \
  --mount volume=dns,target=/etc/resolv.conf \
  --volume var-log,kind=host,source=/var/log \
  --mount volume=var-log,target=/var/log \
```

```
  --volume                                                    kube-
config,kind=host,source=/etc/kubernetes/kubeconfig.yaml \
  --mount                                                  volume=kube-
config,target=/etc/kubernetes/kubeconfig.yaml"
ExecStartPre=/usr/bin/mkdir -p /etc/kubernetes/manifests
ExecStartPre=/usr/bin/mkdir -p /var/log/containers
ExecStartPre=/bin/mkdir -p /var/lib/cni
ExecStartPre=-/usr/bin/rkt rm --uuid-file=/var/run/kubelet-pod.uuid
ExecStart=/usr/lib/coreos/kubelet-wrapper \
  --cni-conf-dir=/etc/kubernetes/cni/net.d \
  --network-plugin=cni \
  --container-runtime=docker \
  --allow-privileged=true \
  --pod-manifest-path=/etc/kubernetes/manifests \
  --hostname-override=192.168.0.150 \
  --cluster_dns=10.252.0.10 \
  --cluster_domain=cluster.local \
  --kubeconfig=/etc/kubernetes/kubeconfig.yaml
ExecStop=-/usr/bin/rkt stop --uuid-file=/var/run/kubelet-pod.uuid
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Posteriormente se debe hacer la creación del archivo que aloja la configuración de los certificados SSL previamente creados, para ello ejecutar:

```
sudo vim /etc/kubernetes/kubeconfig.yaml
```

y colocar el siguiente contenido:

```
apiVersion: v1
kind: Config
clusters:
- name: local
  cluster:
    certificate-authority: /etc/kubernetes/ssl/ca.pem
```

```
    server: http://127.0.0.1:8080
users:
- name: kubelet
  user:
    client-certificate: /etc/kubernetes/ssl/apiserver.pem
    client-key: /etc/kubernetes/ssl/apiserver-key.pem
contexts:
- context:
    cluster: local
    user: kubelet
  name: kubelet-context
current-context: kubelet-context
```

- **APISERVER**

Posteriormente, se deben crear los pods de la capa superior de kubernetes, iniciando con el Apiserver, para lo cual se debe ejecutar:

```
sudo mkdir -p /etc/kubernetes/manifests/
sudo vim /etc/kubernetes/manifests/kube-apiserver.yaml
```

y dentro del archivo agregar el contenido

```
apiVersion: v1
kind: Pod
metadata:
  name: kube-apiserver
  namespace: kube-system
spec:
  hostNetwork: true
  containers:
  - name: kube-apiserver
    image: quay.io/coreos/hyperkube:v1.10.5_coreos.0
    command:
    - /hyperkube
    - apiserver
    - '--bind-address=0.0.0.0'
    -                                                    '--etcd-
servers=http://192.168.0.150:2379,http://192.168.0.151:2379,http://
```

```
192.168.0.160:2379,http://192.168.0.161:2379,http://192.168.0.162:2
379,http://192.168.0.163:2379'
    - '--storage-backend=etcd2'
    - '--allow-privileged=true'
    - '--service-cluster-ip-range=10.252.0.0/24'
    - '--secure-port=443'
    - '--advertise-address=192.168.0.150'
    -                                                    '--admission-
control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorag
eClass,ResourceQuota'
    - '--tls-cert-file=/etc/kubernetes/ssl/apiserver.pem'
    -         '--tls-private-key-file=/etc/kubernetes/ssl/apiserver-
key.pem'
    - '--client-ca-file=/etc/kubernetes/ssl/ca.pem'
    -     '--service-account-key-file=/etc/kubernetes/ssl/apiserver-
key.pem'
    - '--runtime-config=extensions/v1beta1/networkpolicies=true'
    - '--anonymous-auth=false'
    - '--apiserver-count=2'
    - '--requestheader-client-ca-file=/etc/kubernetes/ssl/ca.pem'
    - '--requestheader-username-headers=X-Remote-User'
    - '--requestheader-group-headers=X-Remote-Group'
    - '--requestheader-extra-headers-prefix=X-Remote-Extra'
    - '--authorization-mode=RBAC'
    livenessProbe:
      httpGet:
        host: 127.0.0.1
        port: 8080
        path: /healthz
      initialDelaySeconds: 5
      timeoutSeconds: 2
    ports:
    - containerPort: 443
      hostPort: 443
      name: https
    - containerPort: 8080
      hostPort: 8080
      name: local
```

```
    volumeMounts:
    - mountPath: /etc/kubernetes/ssl
      name: ssl-certs-kubernetes
      readOnly: true
    - mountPath: /etc/ssl/certs
      name: ssl-certs-host
      readOnly: true
  volumes:
  - hostPath:
      path: /etc/kubernetes/ssl
    name: ssl-certs-kubernetes
  - hostPath:
      path: /usr/share/ca-certificates
    name: ssl-certs-host
  - hostPath:
      path: /var/log/
    name: kube-apiserver-log
```

- **PROXY**

Para la configuración del proxy, se debe ejecutar la siguiente sentencia

```
sudo vim /etc/kubernetes/manifests/kube-proxy.yaml
```

Y dentro del archivo añadir las siguientes líneas:

```
apiVersion: v1
kind: Pod
metadata:
  name: kube-proxy
  namespace: kube-system
spec:
  hostNetwork: true
  containers:
  - name: kube-proxy
    image: quay.io/coreos/hyperkube:v1.10.5_coreos.0
    command:
    - /hyperkube
    - proxy
```

```
    - --proxy-mode=iptables
    - --master=http://127.0.0.1:8080
    - --cluster-cidr=10.253.0.0/16
    securityContext:
      privileged: true
    volumeMounts:
    - mountPath: /etc/ssl/certs
      name: ssl-certs-host
      readOnly: true
  volumes:
  - hostPath:
      path: /usr/share/ca-certificates
    name: ssl-certs-host
```

- **CONTROLLER**

Para la configuración del controller manager, se debe crear el archivo con el comando:

```
sudo vim /etc/kubernetes/manifests/kube-controller-manager.yaml
```

Y dentro de él colocar la siguiente configuración

```
apiVersion: v1
kind: Pod
metadata:
 name: kube-controller-manager
 namespace: kube-system
spec:
 hostNetwork: true
 containers:
 - name: kube-controller-manager
   image: quay.io/coreos/hyperkube:v1.10.5_coreos.0
   command:
   - /hyperkube
   - controller-manager
   - --master=http://127.0.0.1:8080
   - --leader-elect=true
   -                                  --service-account-private-key-
file=/etc/kubernetes/ssl/apiserver-key.pem
```

```
    - --root-ca-file=/etc/kubernetes/ssl/ca.pem
    - --horizontal-pod-autoscaler-upscale-delay=50s

    resources:
      requests:
        cpu: 200m
    livenessProbe:
      httpGet:
        host: 127.0.0.1
        path: /healthz
        port: 10252
      initialDelaySeconds: 15
      timeoutSeconds: 15
    volumeMounts:
    - mountPath: /etc/kubernetes/ssl
      name: ssl-certs-kubernetes
      readOnly: true
    - mountPath: /etc/ssl/certs
      name: ssl-certs-host
      readOnly: true
  volumes:
  - hostPath:
      path: /etc/kubernetes/ssl
    name: ssl-certs-kubernetes
  - hostPath:
      path: /usr/share/ca-certificates
    name: ssl-certs-host
```

- **SCHEDULER**

Para la configuración del scheduler se debe crear un archivo con el comando:

```
sudo vim /etc/kubernetes/manifests/kube-scheduler.yaml
```

Y añadir la siguiente configuración dentro de él:

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: kube-scheduler
  namespace: kube-system
spec:
  hostNetwork: true
  containers:
  - name: kube-scheduler
    image: quay.io/coreos/hyperkube:v1.10.5_coreos.0
    command:
    - /hyperkube
    - scheduler
    - --master=http://127.0.0.1:8080
    - --leader-elect=true
    resources:
      requests:
        cpu: 100m
    livenessProbe:
      httpGet:
        host: 127.0.0.1
        path: /healthz
        port: 10251
      initialDelaySeconds: 15
      timeoutSeconds: 15
```

Finalmente se deben iniciar los servicios con los siguientes comandos:

```
sudo systemctl daemon-reload
sudo systemctl start kubelet
sudo systemctl enable kubelet
```

Posteriormente se deben ver los contenedores corriendo dentro de los nodos master, para ello ejecutar el comando `docker ps |egrep -v "(pause|exporter)"`

Contenedores en los nodos master

```
coreos@k8s-master-1 ~ $ docker ps |egrep -v "(pause|exporter)"
CONTAINER ID        IMAGE                         COMMAND                  CREATED
c667c33dc6e3        e6ea97fa6ba3                  "/hyperkube controll…"   2 hours ago
controller-manager-192.168.0.150_kube-system_62f20775f29517745e0de6d91e3438db_16
dc9d65e9408d        e6ea97fa6ba3                  "/hyperkube proxy --…"   5 hours ago
.0.150_kube-system_286a026492ce8aac10b0c75aa6912e64_11
77fd3bc26a4e        e6ea97fa6ba3                  "/hyperkube apiserve…"   5 hours ago
-192.168.0.150_kube-system_39b83eeee9512a72c3154adb5f5fc36b_9
a3408eb2877c        e6ea97fa6ba3                  "/hyperkube schedule…"   5 hours ago
-192.168.0.150_kube-system_60fa1820bb5b669cf22fc0543c0f64fc_16
coreos@k8s-master-1 ~ $
```

Fuente: Autores

Para probar que la configuración quede configurada correctamente, se debe ejecutar el comando `curl http://127.0.0.1:8080/version`

La salida debe arrojar lo siguiente:

```
{
  "major": "",
  "minor": "",
  "gitVersion": "v1.10.5+coreos.0",
  "gitCommit": "49020061462317aa0d54c2577f5bfb257e9cf58e",
  "gitTreeState": "clean",
  "buildDate": "2018-01-08T18:21:24Z",
  "goVersion": "go1.9.2",
  "compiler": "gc",
  "platform": "linux/amd64"
}
```

Configuración en nodos worker

Esta configuración aplica para los 4 nodos worker que intervienen en el cluster.

En primer lugar, se debe especificar la configuración del servicio de kubelet, para ello se debe crear el archivo:

```
sudo vim /etc/systemd/system/kubelet.service
```

Y dentro del archivo colocar las siguientes líneas

```
[Service]
Environment=KUBELET_IMAGE_TAG=v1.10.5_coreos.0
Environment="RKT_RUN_ARGS=--uuid-file-save=/var/run/kubelet-
pod.uuid \
--volume dns,kind=host,source=/etc/resolv.conf \
--mount volume=dns,target=/etc/resolv.conf \
```

```
--volume var-log,kind=host,source=/var/log \
--mount volume=var-log,target=/var/log"
ExecStartPre=/usr/bin/mkdir -p /etc/kubernetes/manifests
ExecStartPre=/usr/bin/mkdir -p /var/log/containers
ExecStartPre=/bin/mkdir -p /var/lib/cni
ExecStartPre=-/usr/bin/rkt rm --uuid-file=/var/run/kubelet-pod.uuid
ExecStart=/usr/lib/coreos/kubelet-wrapper \
--cni-conf-dir=/etc/kubernetes/cni/net.d \
--network-plugin=cni \
--container-runtime=docker \
--register-node=true \
--allow-privileged=true \
--pod-manifest-path=/etc/kubernetes/manifests \
--hostname-override=192.168.0.160 \
--cluster_dns=10.252.0.10 \
--cluster_domain=cluster.local \
--kubeconfig=/etc/kubernetes/worker-kubeconfig.yaml \
--tls-cert-file=/etc/kubernetes/ssl/worker.pem \
--tls-private-key-file=/etc/kubernetes/ssl/worker-key.pem
ExecStop=-/usr/bin/rkt stop --uuid-file=/var/run/kubelet-pod.uuid
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Posteriormente se debe hacer la creación de kube-config, para la validación de la autoridad certificadora

```
sudo vim /etc/kubernetes/worker-kubeconfig.yaml
```

Y dentro del archivo añadir las siguientes líneas:

```
apiVersion: v1
kind: Config
clusters:
- name: local
  cluster:
    certificate-authority: /etc/kubernetes/ssl/ca.pem
```

```
    server: https://k8s-master.k8s.local
users:
- name: kubelet
  user:
    client-certificate: /etc/kubernetes/ssl/worker.pem
    client-key: /etc/kubernetes/ssl/worker-key.pem
contexts:
- context:
    cluster: local
    user: kubelet
  name: kubelet-context
current-context: kubelet-context
```

- **PROXY**

El proxy es necesario para establecer la comunicación con los pods alojados en otros nodos del cluster, para ello se debe crear el archivo:

```
sudo mkdir -p /etc/kubernetes/manifests/
sudo vim /etc/kubernetes/manifests/kube-proxy.yaml
```

Y dentro del archivo colocar las siguientes líneas

```
apiVersion: v1
kind: Pod
metadata:
 name: kube-proxy
 namespace: kube-system
spec:
 hostNetwork: true
 containers:
 - name: kube-proxy
   image: quay.io/coreos/hyperkube:v1.10.5_coreos.0
   command:
   - /hyperkube
   - proxy
   - --proxy-mode=iptables
   - --master=https://k8s-master.k8s.local
   - --kubeconfig=/etc/kubernetes/worker-kubeconfig.yaml
   - --cluster-cidr=10.253.0.0/16
   securityContext:
```

```
      privileged: true
    volumeMounts:
    - mountPath: /etc/ssl/certs
      name: "ssl-certs"
    - mountPath: /etc/kubernetes/worker-kubeconfig.yaml
      name: "kubeconfig"
      readOnly: true
    - mountPath: /etc/kubernetes/ssl
      name: "etc-kube-ssl"
      readOnly: true
  volumes:
  - name: "ssl-certs"
    hostPath:
      path: "/usr/share/ca-certificates"
  - name: "kubeconfig"
    hostPath:
      path: "/etc/kubernetes/worker-kubeconfig.yaml"
  - name: "etc-kube-ssl"
    hostPath:
      path: "/etc/kubernetes/ssl"
```

Finalmente, se debe recargar el demonio del sistema para que los cambios sean aplicados, para ello ejecutar los siguientes comandos:

```
sudo systemctl daemon-reload
sudo systemctl start kubelet
sudo systemctl enable kubelet
```

Configuración del deployer

Coloque los pares de llaves TLS generados anteriormente en las siguientes ubicaciones

- Archivo: /etc/kubernetes/ssl/ca.pem
- Archivo: /etc/kubernetes/ssl/admin-key.pem
- Archivo: /etc/kubernetes/ssl/admin.pem

Se debe descargar el paquete de instalación de kubectl con el comando:

```
curl -O https://storage.googleapis.com/kubernetes-release/release/v1.9.1/bin/linux/amd64/kubectl
```

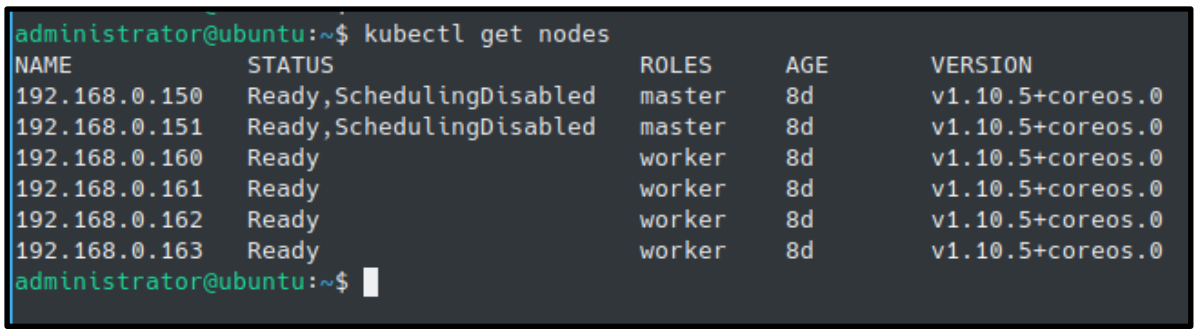
Posteriormente se deben dar permisos de ejecución y mover el ejecutable a la ruta correspondiente

```
chmod +x kubectl
```

```
sudo mv kubectl /usr/bin/kubectl
```

Para este punto ya se deben poder ver los nodos que intervienen en el cluster, para ello ejecutar el comando `kubectl get nodes` y esto arrojará la siguiente salida

Nodos del cluster



Fuente: Autores

NOTA:

La intención de kubernetes es que los nodos Master únicamente ejecuten los componentes necesarios para el funcionamiento del cluster, es decir, no se deben desplegar futuros servicios dentro de ellos, para ello se debe ejecutar lo siguiente:

```
kubectl patch node ${MASTER-NODE} -p '{"spec":{"unschedulable": true}}'
```

siento `${MASTER-NODE}` la ip del nodo, en el cual se quieren deshabilitar los despliegues

Además, se recomienda asignar una etiqueta a cada nodo dentro del cluster, para ello ejecutar:

- Nodos Maestros: `kubectl label node ${MASTER-NODE} kubernetes.io/role=master`
- Nodos Worker: `kubectl label node ${WORKER-NODE} kubernetes.io/role=worker`

Siendo `${MASTER-NODE}` la IP del nodo Master y `${WORKER-NODE}` la IP del nodo Worker

ANEXO 6 - Habilitación de RBAC

RBAC permite administrar los permisos dentro de un cluster de kubernetes, este componente ayuda a que cada elemento dentro del sistema tenga los permisos adecuados, partiendo siempre del principio del privilegio mínimo.

En primer lugar se debe comentar la línea `- '--authorization-mode=RBAC'` en el archivo `/etc/kubernetes/manifests/kube-apiserver.yaml` explicado en el ***anexo5*** , esto hará que el cluster deshabilite la autenticación por RBAC.

Luego de esto, se deben tomar todos los valores `Subject: CN=workers` y `Subject: CN=kube-admin` de cada uno de los certificados generados para ello ejecutar el comando

```
 openssl x509 -in /etc/kubernetes/ssl/k8s-worker.pem -text -noout
```

```
openssl x509 -in /etc/kubernetes/ssl/admin.pem -text -noout
```

Posteriormente, se deben brindar los permisos correspondientes a cada uno de los componentes de kubernetes.

- **Deployer**

Desde el deployer, se debe crear el archivo `deployer_permission.yaml` y añadir el siguiente contenido:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: deployer-permission
subjects:
- kind: User
  name: kube-admin
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

Posteriormente se deben aplicar los cambios con el comando

```
kubectl apply -f deployer_permission.yaml
```

- **Dashboard**

Desde el deployer, se debe crear el archivo `admin_permission.yaml` y añadir el siguiente contenido:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: deployer-permission
subjects:
- kind: User
  name: kube-admin
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

Posteriormente se deben aplicar los cambios con el comando

```
kubectl apply -f admin_permission.yaml
```

- **Worker 1**

Desde el deployer, se debe crear el archivo `worker1_permission.yaml` y añadir el siguiente contenido:

```
kind: ClusterRoleBinding
metadata:
  name: worker1-permission
subjects:
- kind: User
  name: k8s-worker-1
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: system:node
  apiGroup: rbac.authorization.k8s.io
```

Posteriormente se deben aplicar los cambios con el comando

```
kubectl apply -f worker1_permission.yaml
```

- **Worker 2**

Desde el deployer, se debe crear el archivo `worker2_permission.yaml` y añadir el siguiente contenido:

```
kind: ClusterRoleBinding
metadata:
  name: worker2-permission
subjects:
- kind: User
  name: k8s-worker-2
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: system:node
  apiGroup: rbac.authorization.k8s.io
```

Posteriormente se deben aplicar los cambios con el comando

```
kubectl apply -f worker2_permission.yaml
```

- **Worker 3**

Desde el deployer, se debe crear el archivo `worker3_permission.yaml` y añadir el siguiente contenido:

```
kind: ClusterRoleBinding
metadata:
  name: worker3-permission
subjects:
- kind: User
  name: k8s-worker-3
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: system:node
  apiGroup: rbac.authorization.k8s.io
```

Posteriormente se deben aplicar los cambios con el comando

```
kubectl apply -f worker3_permission.yaml
```

- **Worker 4**

Desde el deployer, se debe crear el archivo `worker4_permission.yaml` y añadir el siguiente contenido:

```
kind: ClusterRoleBinding
metadata:
  name: worker4-permission
subjects:
- kind: User
  name: k8s-worker-4
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: system:node
  apiGroup: rbac.authorization.k8s.io
```

Posteriormente se deben aplicar los cambios con el comando

```
kubectl apply -f worker4_permission.yaml
```

Finalmente se debe habilitar nuevamente el complemento de autenticación con RBAC, para lo cual se debe descomentar la línea que se comentó al inicio en el archivo `/etc/kubernetes/manifests/kube-apiserver.yaml`

ANEXO 7 - Caso de Estudio

Este anexo tiene como finalidad dar a conocer al lector la configuración realizada para la implementación del caso de estudio GitLab.
Las configuraciones a continuación son realizadas desde el deployer (ver **Anexo 5),** y requieren de un almacenamiento persistente del cluster, se recomienda la configuración de un NFS, para el almacenamiento de los datos, para conocer cómo configurar un NFS en Ubuntu Server 18.04 ver "**Install NFS Server and Client on Ubuntu 18.04 LTS**"[39] en el siguiente link:
https://vitux.com/install-nfs-server-and-client-on-ubuntu/

Para aplicar las configuraciones descritas en este anexo, el lector debe tener en cuenta los siguientes requisitos:

Cluster de kubernetes 10.5 o más

- NFS
- Deployer correctamente configurado con **kubectl** apuntando al cluster de kubernetes.

Para crear las configuraciones seguir los siguientes pasos:

Ejemplo de creación para el servicio de Redis

Paso 1. Crear namespace proyecto, para esto ejecutar en la consola del deployer el siguiente comando:

```
kubectl create ns proyecto
```

Paso 2. Crear archivo YAML con la configuración que desee crear, por ejemplo:

Consola del deployer, creando archivo e imprimiendo su información

```
administrator@ubuntu:~$ vim servicio-redis.yaml
administrator@ubuntu:~$ cat servicio-redis.yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  labels:
    app: redis
    role: master
    tier: backend
```

[39] "Install NFS Server and Client on Ubuntu 18.04 LTS - VITUX." 13 nov.. 2018, https://vitux.com/install-nfs-server-and-client-on-ubuntu/. Fecha de acceso 26 jul.. 2019.

Fuente: Autores

Paso 2. Crear el servicio por medio de kubectl:

```
kubectl create -f servicio-redis.yaml
```

Configuración Redis

Para realizar el despliegue del servicio Redis en el cluster, tener en cuenta que la información guardada por el servicio se utiliza como base de datos caché y agente de mensajes, leer **(MICROSERVICIO COMO CASO DE ESTUDIO)**.

Configuración del Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-master
spec:
  selector:
    matchLabels:
      app: redis
      role: master
      tier: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: master
        tier: backend
    spec:
      containers:
      - name: master
        image: k8s.gcr.io/redis
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
```

Configuración del servicio redis, para el cluster se crea un servicio con el nombre de `redis-master` que expone el puerto 6379 para que GitLab pueda consumirlo.

```
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  labels:
```

```
    app: redis
    role: master
    tier: backend
spec:
  ports:
  - port: 6379
    targetPort: 6379
  selector:
    app: redis
    role: master
    tier: backend
```

Configuración PostgreSQL

Para la creación del StatefulSet de PostgreSQL se debe crea un secret para la clave del usuario postgres, guardada en formato base64, configuración del secret:

```
apiVersion: v1
data:
 postgresql-password: T0ROWnM2UzVPcw==
kind: Secret
metadata:
 labels:
   app: postgresql
   chart: postgresql-5.3.12
   heritage: Tiller
   release: postgresql
 name: postgresql
 namespace: proyecto
type: Opaque
tier: backend
```

Configuración del servicio

```
apiVersion: v1
kind: Service
metadata:
 labels:
   app: postgresql
   chart: postgresql-5.3.12
   heritage: Tiller
   release: postgresql
 name: postgresql
 namespace: proyecto
spec:
 ports:
 - name: postgresql
   port: 5432
   protocol: TCP
   targetPort: postgresql
 selector:
   app: postgresql
```

```
    release: postgresql
    role: master
```

Configuración del StatefulSet, para esto se debe tener en cuenta que la base de datos cuenta con un almacenamiento persistente NFS, que es:

```
volumes:
      - name: data
        nfs:
          path: /mnt/k8s-nfs1/postgres
          server: 192.168.0.180
```

Se debe reemplazar **“/mnt/k8s-nfs1/postgres”** por la ruta de la carpeta compartida en el NFS, y reemplazar **“192.168.0.180”** por (Dirección ip, Host, Dominio) del NFS.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 generation: 1
 labels:
    app: postgresql
    chart: postgresql-5.3.12
    heritage: Tiller
    release: postgresql
 name: postgresql-postgresql
 namespace: proyecto
spec:
 selector:
    matchLabels:
      app: postgresql
      release: postgresql
      role: master
 serviceName: postgresql-headless
 template:
    metadata:
      labels:
        app: postgresql
        chart: postgresql-5.3.12
        heritage: Tiller
        release: postgresql
        role: master
      name: postgresql
    spec:
      containers:
      - env:
        - name: BITNAMI_DEBUG
          value: "false"
        - name: POSTGRESQL_PORT_NUMBER
          value: "5432"
        - name: PGDATA
          value: /bitnami/postgresql/data
        - name: POSTGRES_USER
          value: postgres
```

```
      - name: POSTGRES_PASSWORD
        valueFrom:
          secretKeyRef:
            key: postgresql-password
            name: postgresql
      image: docker.io/bitnami/postgresql:11.4.0-debian-9-r12
      imagePullPolicy: IfNotPresent
      livenessProbe:
        exec:
          command:
          - sh
          - -c
          - exec pg_isready -U "postgres" -h 127.0.0.1 -p 5432
        failureThreshold: 6
        initialDelaySeconds: 30
        periodSeconds: 10
        successThreshold: 1
        timeoutSeconds: 5
      name: postgresql
      ports:
      - containerPort: 5432
        name: postgresql
        protocol: TCP
      readinessProbe:
        exec:
          command:
          - sh
          - -c
          - |
            pg_isready -U "postgres" -h 127.0.0.1 -p 5432
            [ -f /opt/bitnami/postgresql/tmp/.initialized ]
        failureThreshold: 6
        initialDelaySeconds: 5
        periodSeconds: 10
        successThreshold: 1
        timeoutSeconds: 5
      resources:
        requests:
          cpu: 250m
          memory: 256Mi
      securityContext:
        runAsUser: 1001
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
      - mountPath: /bitnami/postgresql
        name: data
    dnsPolicy: ClusterFirst
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext:
      fsGroup: 1001
    terminationGracePeriodSeconds: 30
    volumes:
```

```
      - name: data
        nfs:
          path: /mnt/k8s-nfs1/postgres
          server: 192.168.0.180
 updateStrategy:
   type: RollingUpdate
   role: master
```

Configuración GitLab

Finalmente, la configuración de GitLab, esté consume de los servicios anteriormente configurados, el lector debe tener en cuenta que, si realiza cambios en los nombres de los servicios, es necesario que actualizar la configuración de Gitlab.

Configuración del servicio

```
apiVersion: v1
kind: Service
metadata:
 name: gitlab
 namespace: proyecto
 labels:
   name: gitlab
spec:
 selector:
   name: gitlab
 ports:
   - name: http
     port: 80
     targetPort: http
   - name: ssh
     port: 1022
     targetPort: ssh
   type: RollingUpdate
```

Configuración del StatefulSet, para esto se debe tener en cuenta que GitLab cuenta con un almacenamiento persistente NFS, que es:

```
volumes:
      - name: data
        nfs:
          path: /mnt/gitlab
          server: 192.168.0.180
```

Se debe reemplazar **“/mnt/gitlab”** por la ruta de la carpeta compartida en el NFS, y reemplazar **“192.168.0.180”** por (Dirección ip, Host, Dominio) del NFS.
El StatefulSet cuenta con 2 réplicas.

```
apiVersion: apps/v1beta1
```

```
kind: StatefulSet
metadata:
  name: gitlab
  namespace: proyecto
spec:
  serviceName: gitlab-ha
  replicas: 2
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        name: gitlab
        app: gitlab
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - gitlab
            topologyKey: "kubernetes.io/hostname"
      containers:
      - name: gitlab
        image: sameersbn/gitlab:8.16.3
        imagePullPolicy: Always
        env:
        - name: TZ
          value: Europe/Stockholm
        - name: GITLAB_TIMEZONE
          value: Stockholm
        - name: DEBUG
          value: "true"
        - name: GITLAB_SECRETS_DB_KEY_BASE
          value: P26qS5+Csz50Dkd0DLM2oN9owVBFg0PB
        - name: GITLAB_SECRETS_SECRET_KEY_BASE
          value: KVaMTKLAIElEp0s4L02c1O9JCP0Rfapb
        - name: GITLAB_SECRETS_OTP_KEY_BASE
          value: nXJJ358Qnci0yF9qpAsLrF2vImaoFR03
        - name: GITLAB_ROOT_PASSWORD
          value: GitlabRootPassword
        - name: GITLAB_ROOT_EMAIL
          value: root@example.com
        - name: GITLAB_HOST
          value: gitlab.k8s.local
        - name: GITLAB_PORT
          value: "80"
        - name: GITLAB_SSH_HOST
          value: "ssh-gitlab.k8s.local"
        - name: GITLAB_SSH_PORT
          value: "1022"
```

```
        - name: GITLAB_NOTIFY_ON_BROKEN_BUILDS
          value: "true"
        - name: GITLAB_NOTIFY_PUSHER
          value: "false"
        - name: GITLAB_BACKUP_SCHEDULE
          value: daily
        - name: GITLAB_BACKUP_TIME
          value: 01:00
        - name: DB_TYPE
          value: postgres
        - name: DB_HOST
          value: postgresql.proyecto
        - name: DB_PORT
          value: "5432"
        - name: DB_USER
          value: gitlab
        - name: DB_PASS
          value: +BP52QIxpT/flVCMpL3KXA==
        - name: DB_NAME
          value: gitlab_production
        - name: REDIS_HOST
          value: redis-master.proyecto
        - name: REDIS_PORT
          value: "6379"
        - name: SMTP_ENABLED
          value: "false"
        - name: SMTP_DOMAIN
          value: ""
        - name: SMTP_HOST
          value: ""
        - name: SMTP_PORT
          value: ""
        - name: SMTP_USER
          value: ""
        - name: SMTP_PASS
          value: ""
        - name: SMTP_STARTTLS
          value: "true"
        - name: SMTP_AUTHENTICATION
          value: login
        - name: IMAP_ENABLED
          value: "false"
        - name: IMAP_HOST
          value: imap.gmail.com
        - name: IMAP_PORT
          value: "993"
        - name: IMAP_USER
          value: mailer@example.com
        - name: IMAP_PASS
          value: password
        - name: IMAP_SSL
          value: "true"
        - name: IMAP_STARTTLS
          value: "false"
```

```
        ports:
        - name: http
          containerPort: 80
        - name: ssh
          containerPort: 22
        volumeMounts:
        - mountPath: /home/git/data
          name: data
        livenessProbe:
          httpGet:
            path: /users/sign_in
            port: 80
          initialDelaySeconds: 360
          timeoutSeconds: 15
        readinessProbe:
          httpGet:
            path: /users/sign_in
            port: 80
          initialDelaySeconds: 15
          timeoutSeconds: 1
        resources:
          requests:
            cpu: 500m
            memory: 2000Mi
      volumes:
      - name: data
        nfs:
          path: /mnt/gitlab
          server: 192.168.0.180
   type: RollingUpdate
```

Configuración del Horizontal Pod AutoScaler, con esta configuración el servicio escala a un número total de 4 replicas, cuando los recursos del StatefulSet sean mayores al 90% de la memoria Ram ó CPU.

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
 name: hpa-gitlab
 namespace: proyecto
 labels:
   stack: proyecto
   app: gitlab
spec:
 scaleTargetRef:
   apiVersion: apps/v1beta1
   kind: StatefulSet
   name: gitlab
 minReplicas: 2
 maxReplicas: 4
 metrics:
   - type: Resource
     resource:
```

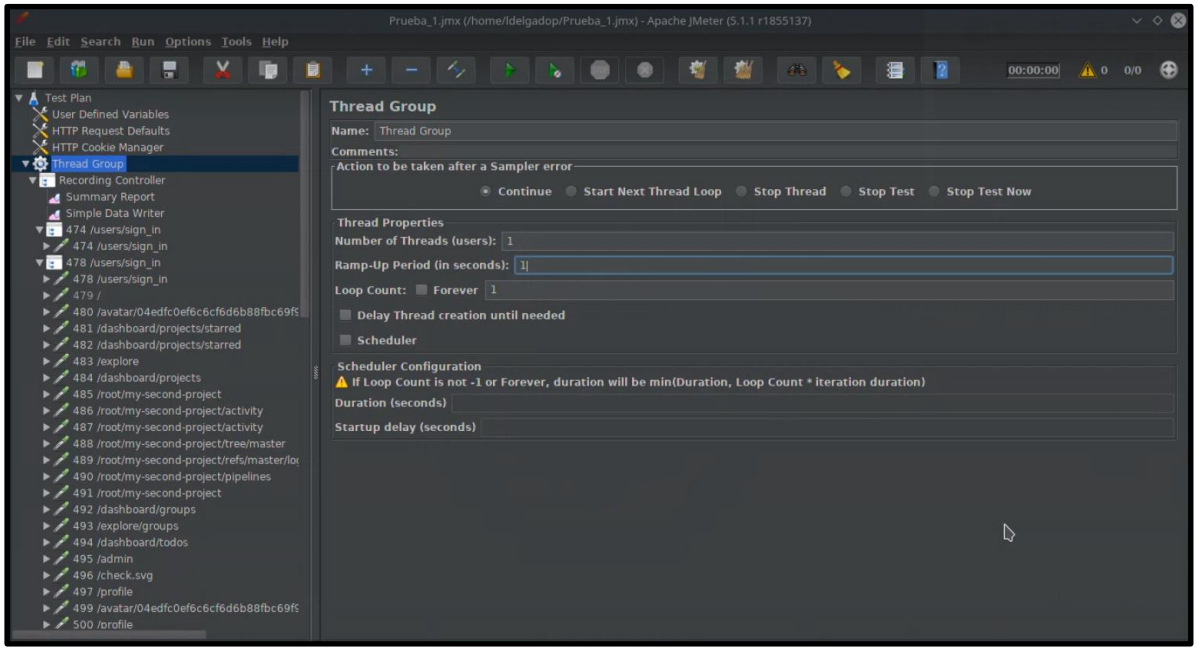
```
      name: cpu
      targetAverageUtilization: 90
  - type: Resource
    resource:
      name: memory
      targetAverageUtilization: 90
```

ANEXO 8 - Pruebas de Stress

En el siguiente documento se muestra la aplicabilidad de las pruebas realizadas en apache Jmeter para medir y corroborar la aceptación por parte del cluster ante diferentes números de usuarios concurrentes.

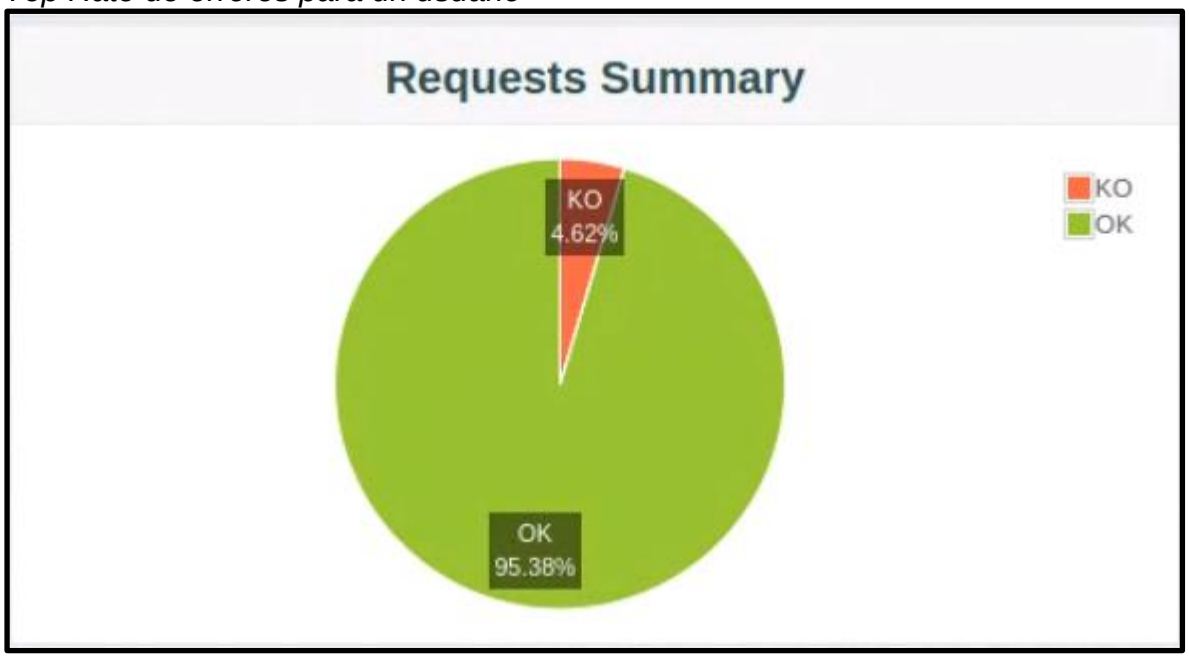
- **1 usuario concurrente**

Prueba para un solo usuario



Fuente: Autores

Top Rate de errores para un usuario



Fuente: Autores

Tabla de peticiones para un usuario

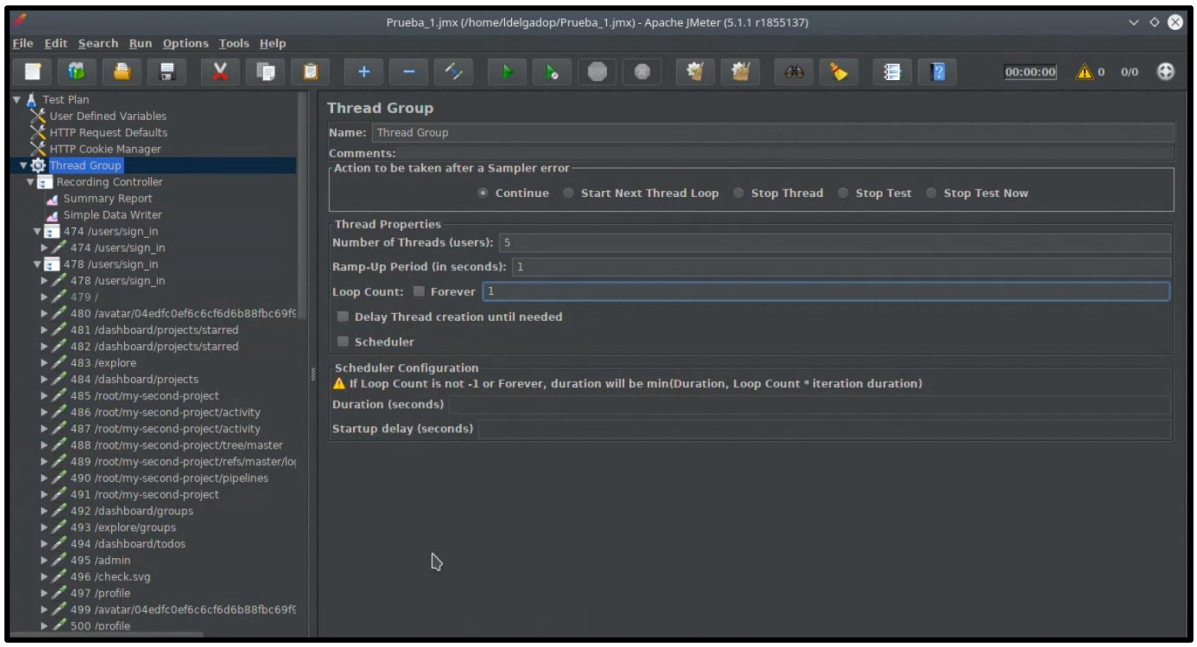
Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**130**	**6**	**4.62%**	**305.42**	**22**	**1672**	**576.00**	**778.65**	**1666.42**	**5.72**	**36.00**	**3.88**
474 /users/sign_in	2	0	0.00%	810.00	810	810	810.00	810.00	810.00	2.47	24.33	1.08
478 /users/sign_in	2	2	100.00%	8784.00	283	17285	17285.00	17285.00	17285.00	0.12	17.87	1.96
480 /avatar/04edfc0ef6c6cf6d6b88fbc69f9f9071	1	0	0.00%	145.00	145	145	145.00	145.00	145.00	6.90	21.92	2.40
481 /dashboard/projects/starred	1	0	0.00%	962.00	962	962	962.00	962.00	962.00	1.04	10.85	1.05
481 /dashboard/projects/starred-0	1	0	0.00%	386.00	386	386	386.00	386.00	386.00	2.59	1.29	1.33
481 /dashboard/projects/starred-1	1	0	0.00%	576.00	576	576	576.00	576.00	576.00	1.74	17.25	0.87
482 /dashboard/projects/starred	1	1	100.00%	145.00	145	145	145.00	145.00	145.00	6.90	2.92	4.46
483 /explore	1	0	0.00%	244.00	244	244	244.00	244.00	244.00	4.10	36.23	2.23
484 /dashboard/projects	1	0	0.00%	599.00	599	599	599.00	599.00	599.00	1.67	17.42	1.72
484 /dashboard/projects-0	1	0	0.00%	253.00	253	253	253.00	253.00	253.00	3.95	1.97	2.05
484 /dashboard/projects-1	1	0	0.00%	346.00	346	346	346.00	346.00	346.00	2.89	28.72	1.48
485 /root/my-second-project	1	0	0.00%	497.00	497	497	497.00	497.00	497.00	2.01	20.98	2.17
485 /root/my-second-project-0	1	0	0.00%	185.00	185	185	185.00	185.00	185.00	5.41	2.70	2.93
485 /root/my-second-project-1	1	0	0.00%	312.00	312	312	312.00	312.00	312.00	3.21	31.83	1.71
486 /root/my-second-project/activity	1	0	0.00%	480.00	480	480	480.00	480.00	480.00	2.08	21.75	2.29

Fuente: Autores

- **5 usuarios concurrentes**

Prueba para un cinco usuarios



Fuente: Autores

Top Rate de errores para cinco usuarios



Fuente: Autores

Tabla de peticiones para cinco usuario

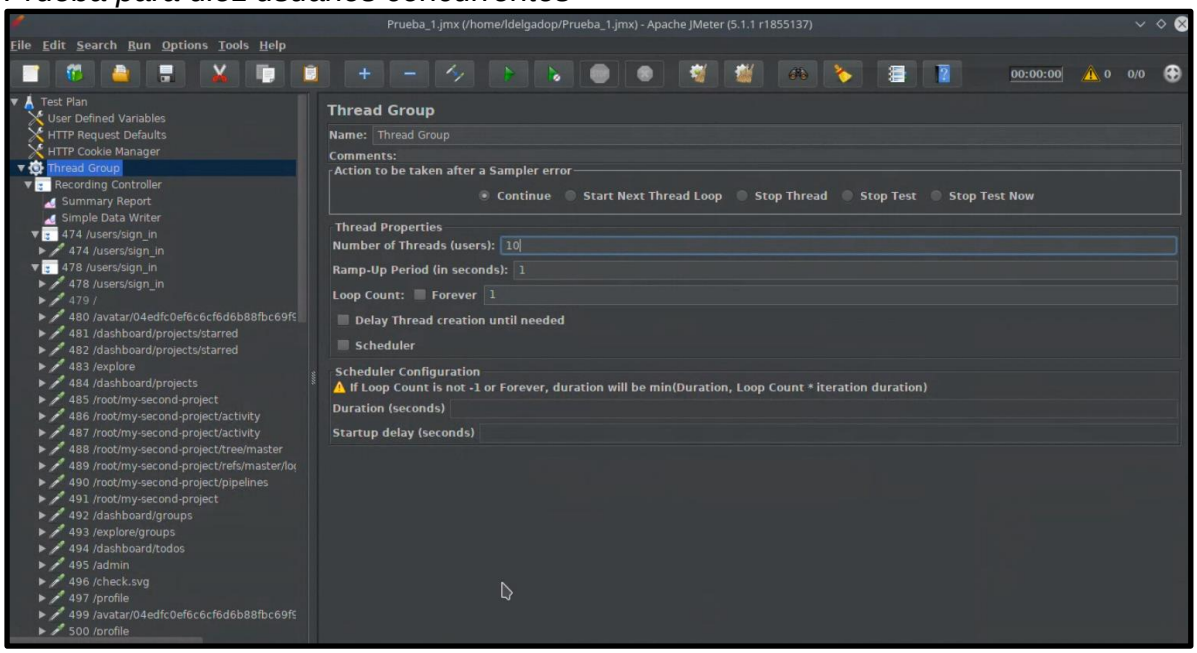
Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**650**	**30**	**4.62%**	**367.52**	**35**	**1176**	**670.00**	**763.35**	**1017.80**	**23.81**	**149.95**	**16.14**
474 /users/sign_in	10	0	0.00%	423.60	276	610	610.00	610.00	610.00	8.35	82.39	3.65
478 /users/sign_in	10	10	100.00%	9312.60	72	19511	19451.90	19511.00	19511.00	0.51	79.17	8.70
480 /avatar/04edfc0ef6c6cf6d6b88fbc69f9f9071	5	0	0.00%	134.80	121	149	149.00	149.00	149.00	4.78	15.19	1.67
481 /dashboard/projects/starred	5	0	0.00%	496.40	405	632	632.00	632.00	632.00	3.66	38.23	3.70
481 /dashboard/projects/starred-0	5	0	0.00%	233.20	123	310	310.00	310.00	310.00	4.23	2.11	2.16
481 /dashboard/projects/starred-1	5	0	0.00%	263.20	182	392	392.00	392.00	392.00	4.44	44.17	2.22
482 /dashboard/projects/starred	5	5	100.00%	212.60	127	298	298.00	298.00	298.00	5.67	2.40	3.66
483 /explore	5	0	0.00%	331.20	132	541	541.00	541.00	541.00	4.18	36.96	2.27
484 /dashboard/projects	5	0	0.00%	790.40	512	1030	1030.00	1030.00	1030.00	2.97	30.96	3.06
484 /dashboard/projects-0	5	0	0.00%	301.60	154	508	508.00	508.00	508.00	4.30	2.15	2.23
484 /dashboard/projects-1	5	0	0.00%	488.80	358	652	652.00	652.00	652.00	3.48	34.59	1.79
485 /root/my-second-project	5	0	0.00%	603.40	430	857	857.00	857.00	857.00	3.16	32.92	3.40
485 /root/my-second-project-0	5	0	0.00%	293.60	200	477	477.00	477.00	477.00	4.25	2.12	2.31
485 /root/my-second-project-1	5	0	0.00%	309.40	230	406	406.00	406.00	406.00	4.52	44.85	2.41
486 /root/my-second-project/activity	5	0	0.00%	603.60	411	707	707.00	707.00	707.00	3.77	39.29	4.15
486 /root/my-second-project/activity-0	5	0	0.00%	239.20	117	361	361.00	361.00	361.00	5.38	2.68	3.01

Fuente: Autores

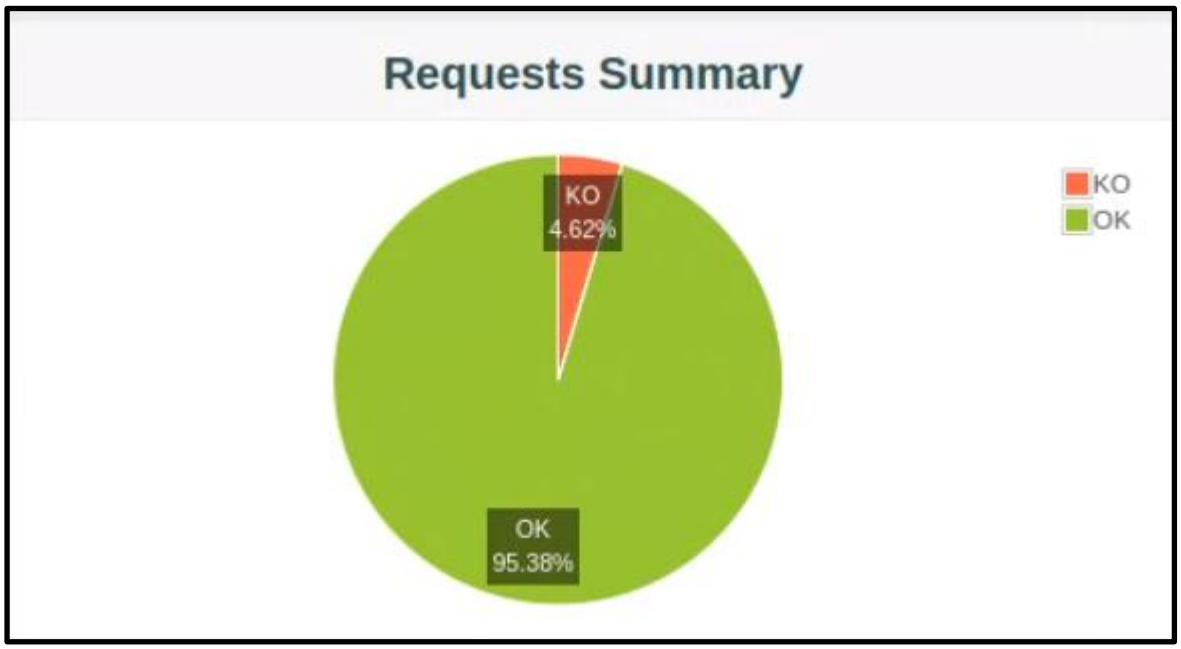
- **10 usuarios concurrentes**

Prueba para diez usuarios concurrentes



Fuente: Autores

Top Rate de errores para diez usuarios



Fuente: Autores

Tabla de peticiones para diez usuarios

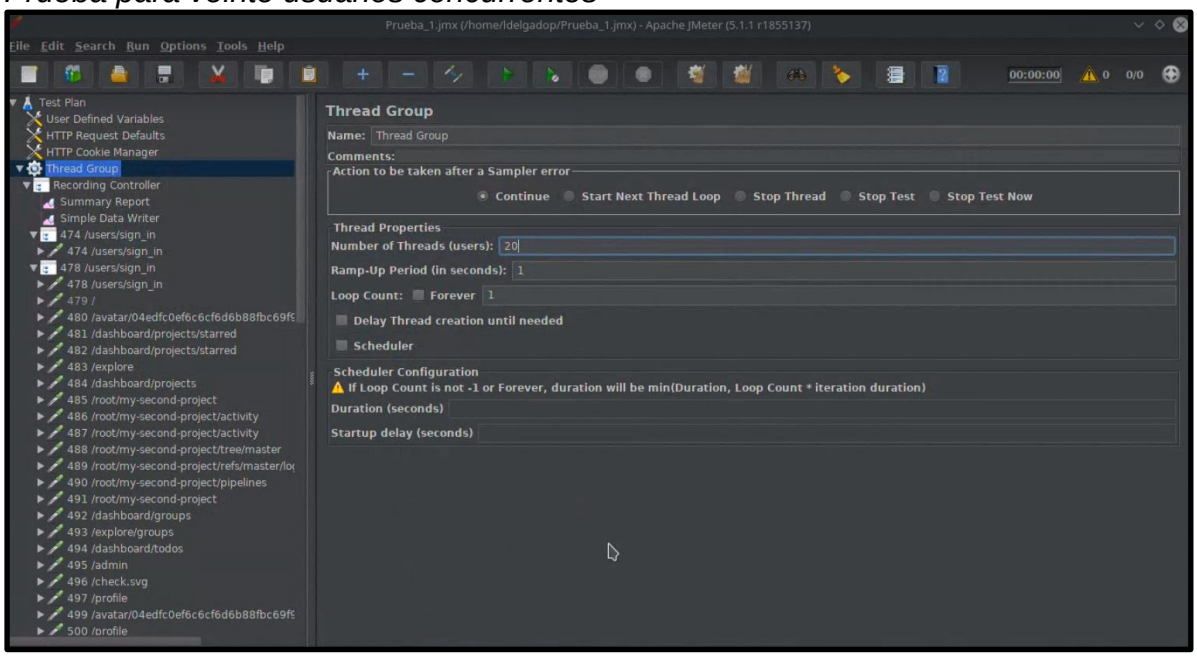
Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**1300**	**60**	**4.62%**	**677.04**	**36**	**2445**	**1313.50**	**1485.60**	**1912.87**	**26.34**	**165.88**	**17.85**
474 /users/sign_in	20	0	0.00%	446.80	254	828	807.00	828.00	828.00	13.11	129.38	5.72
478 /users/sign_in	20	20	100.00%	17096.55	56	36272	36030.60	36268.00	36272.00	0.54	82.77	9.10
480 /avatar/04edfc0ef6c6cf6d6b88fbc69f9f9071	10	0	0.00%	128.40	117	145	144.30	145.00	145.00	7.11	22.61	2.48
481 /dashboard/projects/starred	10	0	0.00%	1135.40	678	1687	1667.20	1687.00	1687.00	3.62	37.75	3.66
481 /dashboard/projects/starred-0	10	0	0.00%	473.30	177	853	840.90	853.00	853.00	5.19	2.59	2.65
481 /dashboard/projects/starred-1	10	0	0.00%	662.00	299	981	969.90	981.00	981.00	3.87	38.42	1.93
482 /dashboard/projects/starred	10	10	100.00%	443.60	173	840	818.90	840.00	840.00	4.23	1.79	2.73
483 /explore	10	0	0.00%	613.20	213	1746	1653.20	1746.00	1746.00	2.60	22.95	1.41
484 /dashboard/projects	10	0	0.00%	1145.40	630	1917	1866.70	1917.00	1917.00	2.13	22.26	2.20
484 /dashboard/projects-0	10	0	0.00%	459.90	200	913	898.30	913.00	913.00	2.53	1.26	1.31
484 /dashboard/projects-1	10	0	0.00%	685.30	297	1535	1463.50	1535.00	1535.00	2.30	22.82	1.18
485 /root/my-second-project	10	0	0.00%	1415.90	996	2040	2027.30	2040.00	2040.00	1.98	20.65	2.13
485 /root/my-second-project-0	10	0	0.00%	707.50	408	1467	1439.20	1467.00	1467.00	2.16	1.08	1.17
485 /root/my-second-project-1	10	0	0.00%	708.40	425	1008	995.60	1008.00	1008.00	2.38	23.66	1.27

Fuente: Autores

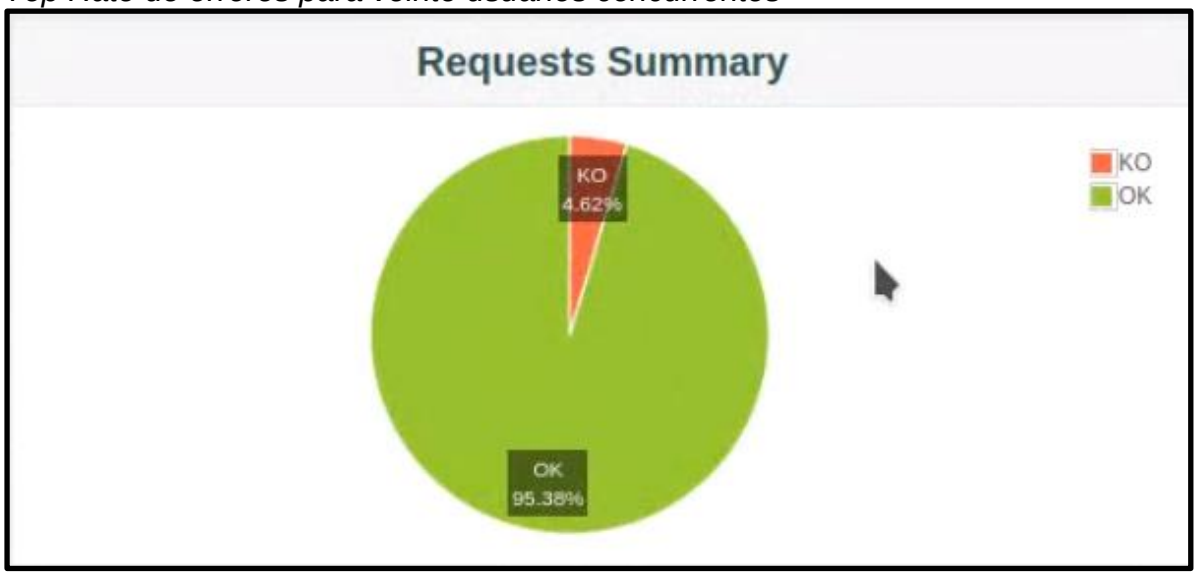
- **20 usuarios concurrentes**

Prueba para veinte usuarios concurrentes



Fuente: Autores

Top Rate de errores para veinte usuarios concurrentes



Fuente: Autores

Tabla de peticiones para veinte usuarios

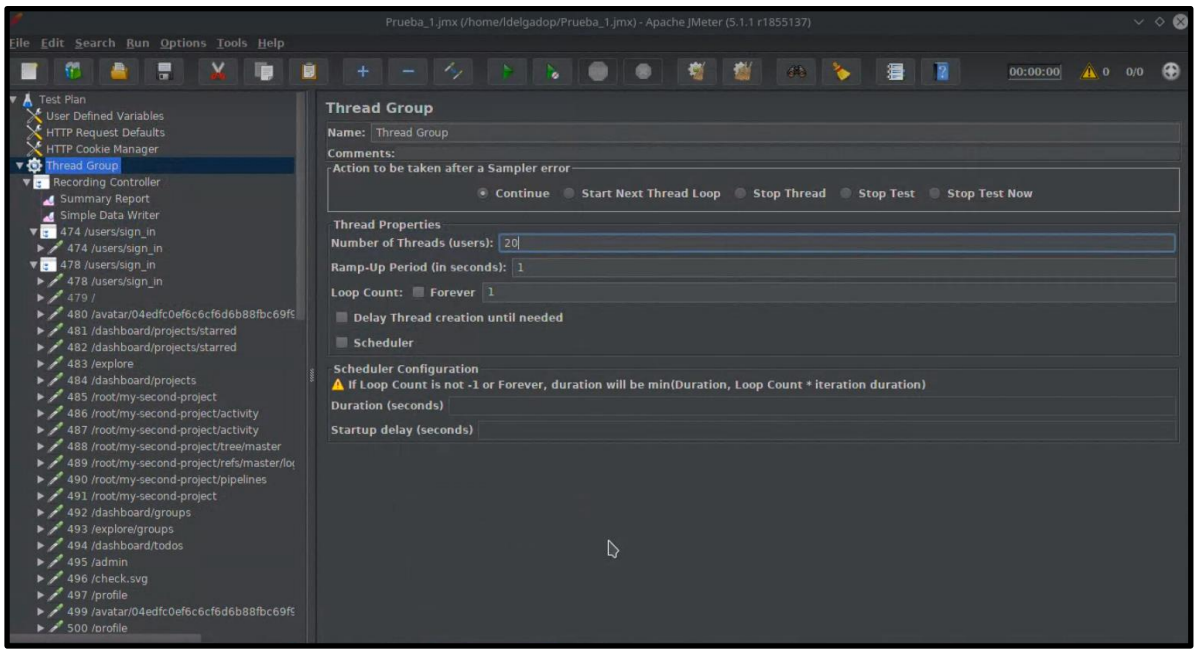
Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2600**	**120**	**4.62%**	**1477.65**	**21**	**24420**	**2782.60**	**3492.45**	**6589.41**	**21.55**	**135.71**	**14.61**
474 /users/sign_in	40	0	0.00%	788.95	356	1341	1165.00	1333.00	1341.00	12.25	120.81	5.35
478 /users/sign_in	40	40	100.00%	31972.62	83	71660	68210.50	71233.80	71660.00	0.54	83.88	9.22
480 /avatar/04edfc0ef6c6cf6d6b88fbc69f9f9071	20	0	0.00%	133.45	114	204	146.30	201.15	204.00	7.09	22.54	2.47
481 /dashboard/projects/starred	20	0	0.00%	1850.05	1280	2231	2150.80	2227.05	2231.00	4.21	43.88	4.25
481 /dashboard/projects/starred-0	20	0	0.00%	775.30	298	1404	1104.60	1389.75	1404.00	4.98	2.48	2.55
481 /dashboard/projects/starred-1	20	0	0.00%	1075.00	686	1415	1398.70	1414.35	1415.00	4.55	45.24	2.27
482 /dashboard/projects/starred	20	20	100.00%	1004.75	464	1622	1281.90	1605.15	1622.00	4.64	1.96	3.00
483 /explore	20	0	0.00%	1115.35	614	1843	1406.90	1821.25	1843.00	3.61	31.89	1.96
484 /dashboard/projects	20	0	0.00%	2435.65	1583	3615	3562.60	3614.65	3615.00	2.40	25.07	2.48
484 /dashboard/projects-0	20	0	0.00%	1092.00	515	2272	1688.10	2244.25	2272.00	3.34	1.66	1.73
484 /dashboard/projects-1	20	0	0.00%	1343.65	781	2329	2253.50	2326.15	2329.00	2.81	27.94	1.44
485 /root/my-second-project	20	0	0.00%	2256.95	898	4263	4157.10	4259.95	4263.00	2.57	26.80	2.77
485 /root/my-second-project-0	20	0	0.00%	1152.55	306	2262	2131.40	2256.10	2262.00	2.98	1.49	1.62
485 /root/my-second-project-1	20	0	0.00%	1104.35	232	2241	2114.90	2234.90	2241.00	2.89	28.73	1.54
486 /root/my-second-project/activity	20	0	0.00%	2633.30	751	4658	4454.90	4650.30	4658.00	2.18	22.76	2.40
486 /root/my-second-project/activity-0	20	0	0.00%	1227.05	187	2328	2207.40	2322.15	2328.00	2.78	1.39	1.55

Fuente: Usuarios

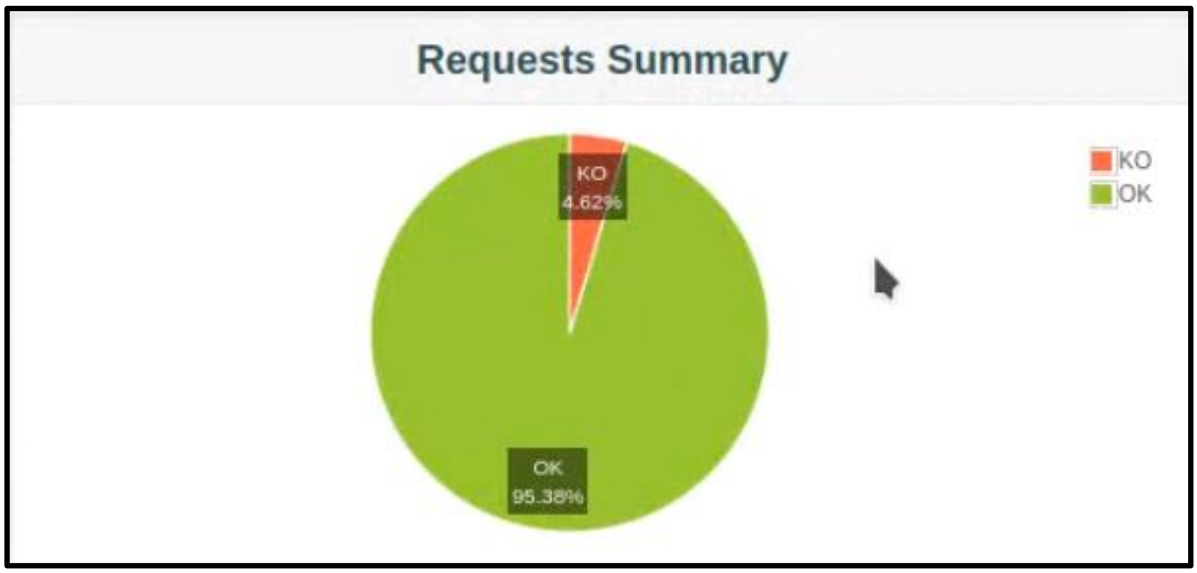
- **50 usuarios concurrentes**

Prueba para cincuenta usuarios concurrentes



Fuente: Autores

Top Rate de errores para cincuenta usuarios concurrentes



Fuente: Autores

Tabla de peticiones para cincuenta usuarios

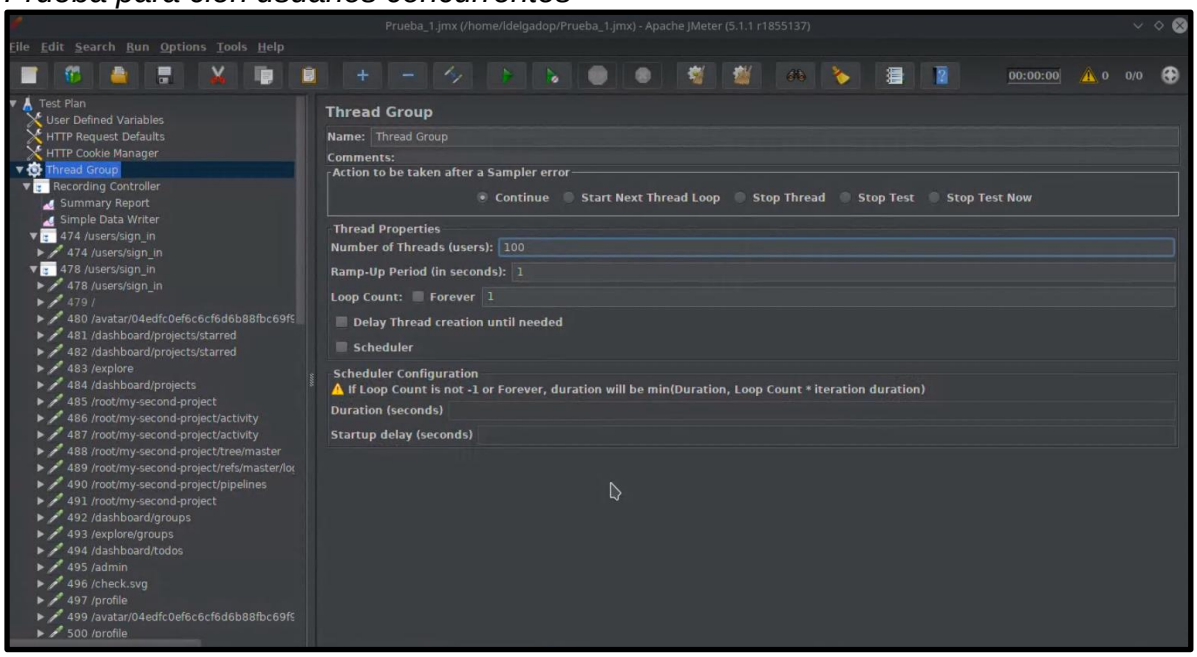
Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2600**	**120**	**4.62%**	**1477.65**	**21**	**24420**	**2782.60**	**3492.45**	**6589.41**	**21.55**	**135.71**	**14.61**
474 /users/sign_in	40	0	0.00%	788.95	356	1341	1165.00	1333.00	1341.00	12.25	120.81	5.35
478 /users/sign_in	40	40	100.00%	31972.62	83	71660	68210.50	71233.80	71660.00	0.54	83.88	9.22
480 /avatar/04edfc0ef6c6cf6d6b88fbc69f9f9071	20	0	0.00%	133.45	114	204	146.30	201.15	204.00	7.09	22.54	2.47
481 /dashboard/projects/starred	20	0	0.00%	1850.05	1280	2231	2150.80	2227.05	2231.00	4.21	43.88	4.25
481 /dashboard/projects/starred-0	20	0	0.00%	775.30	298	1404	1104.60	1389.75	1404.00	4.98	2.48	2.55
481 /dashboard/projects/starred-1	20	0	0.00%	1075.00	686	1415	1398.70	1414.35	1415.00	4.55	45.24	2.27
482 /dashboard/projects/starred	20	20	100.00%	1004.75	464	1622	1281.90	1605.15	1622.00	4.64	1.96	3.00
483 /explore	20	0	0.00%	1115.35	614	1843	1406.90	1821.25	1843.00	3.61	31.89	1.96
484 /dashboard/projects	20	0	0.00%	2435.65	1583	3615	3562.60	3614.65	3615.00	2.40	25.07	2.48
484 /dashboard/projects-0	20	0	0.00%	1092.00	515	2272	1688.10	2244.25	2272.00	3.34	1.66	1.73
484 /dashboard/projects-1	20	0	0.00%	1343.65	781	2329	2253.50	2326.15	2329.00	2.81	27.94	1.44
485 /root/my-second-project	20	0	0.00%	2256.95	898	4263	4157.10	4259.95	4263.00	2.57	26.80	2.77
485 /root/my-second-project-0	20	0	0.00%	1152.55	306	2262	2131.40	2256.10	2262.00	2.98	1.49	1.62
485 /root/my-second-project-1	20	0	0.00%	1104.35	232	2241	2114.90	2234.90	2241.00	2.89	28.73	1.54
486 /root/my-second-project/activity	20	0	0.00%	2633.30	751	4658	4454.90	4650.30	4658.00	2.18	22.76	2.40
486 /root/my-second-project/activity-0	20	0	0.00%	1227.05	187	2328	2207.40	2322.15	2328.00	2.78	1.39	1.55

Fuente: Usuarios

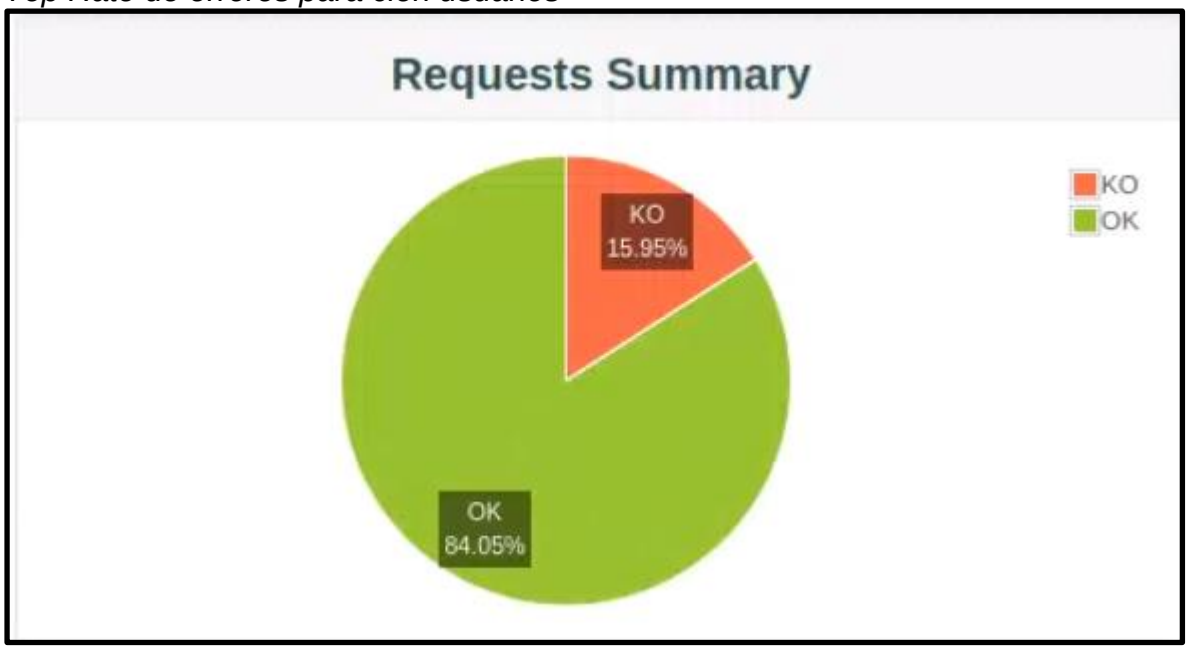
- **100 usuarios concurrentes**

Prueba para cien usuarios concurrentes



Fuente: Autores

Top Rate de errores para cien usuarios



Fuente: Autores

Tabla de peticiones para cien usuarios

Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**10964**	**1749**	**15.95%**	**5374.06**	**0**	**95246**	**13937.00**	**20482.25**	**58297.15**	**27.49**	**148.68**	**18.01**
474 /users/sign_in	200	0	0.00%	3764.02	647	9628	7115.10	8845.75	9626.67	17.69	174.45	7.72
478 /users/sign_in	200	200	100.00%	146721.56	669	368108	352453.10	359348.60	365990.33	0.53	64.17	8.24
480 /avatar/04edfc0ef6c6cf6d6b88fbc69f9f9071	100	0	0.00%	129.65	114	221	140.00	142.95	220.29	5.87	18.64	2.04
481 /dashboard/projects/starred	100	0	0.00%	9159.53	1736	25382	18522.20	19383.20	25375.97	2.64	27.51	2.67
481 /dashboard/projects/starred-0	100	0	0.00%	4356.79	657	13362	9202.60	11308.05	13351.19	3.86	1.93	1.98
481 /dashboard/projects/starred-1	100	0	0.00%	4802.63	723	15362	10329.90	11987.85	15360.11	3.02	29.95	1.51
482 /dashboard/projects/starred	100	100	100.00%	4857.92	19	15776	12308.10	12835.20	15760.94	2.25	0.95	1.45
483 /explore	100	17	17.00%	4404.23	10	15251	12589.00	14883.10	15250.44	2.23	16.52	1.22
484 /dashboard/projects	100	29	29.00%	10305.78	5	28173	18977.80	24246.65	28151.84	1.80	13.60	1.71
484 /dashboard/projects-0	85	0	0.00%	5954.09	231	15295	12942.60	14412.40	15295.00	1.89	0.94	0.98
484 /dashboard/projects-1	85	14	16.47%	6047.84	8	16327	14166.80	15460.60	16327.00	1.58	13.22	0.81
485 /root/my-second-project	100	28	28.00%	9202.53	4	29838	23453.10	24681.15	29822.91	1.38	10.50	1.33
485 /root/my-second-project-0	79	0	0.00%	5773.00	119	16350	12927.00	15986.00	16350.00	1.31	0.65	0.71
485 /root/my-second-project-1	79	7	8.86%	5741.97	9	18749	14377.00	15634.00	18749.00	1.14	10.31	0.61
486 /root/my-second-project/activity	100	19	19.00%	10019.62	5	30601	19053.90	23802.50	30577.11	1.19	10.15	1.21
486 /root/my-second-project/activity-0	84	0	0.00%	6593.85	194	17675	13006.50	14188.25	17675.00	1.03	0.52	0.58

Fuente: Autores