

**DETECCIÓN AUTOMÁTICA DEL NIVEL DE ESTRATIFICACIÓN
SOCIOECONÓMICO URBANO USANDO REDES NEURONALES
CONVOLUCIONALES SOBRE IMÁGENES SATELITALES CON
INFORMACIÓN AUMENTADA**

DANIEL ALCIDES CARVAJAL PATIÑO

**UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICOMECHANICAS
ESCUELA DE INGENIERIA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA**

2018

**DETECCIÓN AUTOMÁTICA DEL NIVEL DE ESTRATIFICACIÓN
SOCIOECONÓMICO URBANO USANDO REDES NEURONALES
CONVOLUCIONALES SOBRE IMÁGENES SATELITALES CON
INFORMACIÓN AUMENTADA**

DANIEL ALCIDES CARVAJAL PATIÑO

**TRABAJO DE GRADO PARA OPTAR POR EL TITULO DE
INGENIERIA DE SISTEMAS**

Director

FABIO MARTINEZ CARRILLO

Ph.D. EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Codirector

Ph.D RAUL RAMOS POLLAN

**UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICOMECHANICAS
ESCUELA DE INGENIERIA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA**

2018

ESPACIO PARA NOTA

ESPACIO PARA CARTA AUTORIZACIÓN USO DE DATOS

CONTENIDO

	pág.
INTRODUCCION	13
1. OBJETIVOS	14
1.1. OBJETIVO GENERAL	14
1.2. OBJETIVOS ESPECIFICOS	14
2. MARCO TEÓRICO	15
2.1. ESTRATO SOCIAL	15
2.2. MACHINE LEARNING	16
2.2.1. OPENSTACK	17
2.2.1.1. KEYSTONE:	17
2.2.1.2. GLANCE	18
2.2.1.3. NOVA	18
2.2.1.4. NEUTRON	19
2.2.1.5. CINDER	19
2.2.1.6. HEAT	19
2.2.1.7. MAGNUM	19
2.2.2. ALTA DISPONIBILIDAD	19
2.2.2.1. ALTA DISPONIBILIDAD CON PACEMAKER Y COROSYNC	20
2.2.2.2. REPLICACION DE DATOS CON LSYNC	20
2.2.2.3. VIRTUALIZACION	21
2.2.2.4. CONTENEDORES	21
2.3. DEEP LEARNING	21
2.4. MACHINE LEARNING Y EL ESTRATO SOCIAL	21
3. METODOLOGIA	22
4. DESARROLLO DEL PROYECTO	24
4.1. FASE 1	24

4.2. FASE 2	25
4.3. FASE 3	29
4.3.1. Hosts	30
4.3.2. API REST	30
4.3.3. Interfaces	30
4.3.4. MariaDB Galera Cluster	32
4.3.5. RabbitMQ	34
4.3.6. Memcached	35
4.3.7. ETCD	35
4.3.8. Pacemaker, Corosyn, Heartbeat, la alta disponibilidad. y el almacenamiento compartido.	35
4.3.9. Servicio de Identidad	37
4.3.10. Servicio de Imagenes	37
4.3.11. Servicio de Computo	37
4.3.12. Servicio de Red	38
4.3.13. Servicio de Orquestacion	38
4.3.14. Servicio de almacenamiento de bloques	38
4.3.15. Dashboard de Horizon	38
4.4. FASE 4	42
4.4.1. Contenedores	42
4.4.2. Docker	43
4.4.3. Kubernetes	44
4.5. FASE 5	54
5. CONCLUSIONES	63
6. RECOMENDACIONES Y TRABAJO FUTURO	64
7. LIMITACIONES Y PROBLEMAS	65
8. REFERENCIAS	66
9. BIBLIOGRAFIA	68

LISTA DE FIGURAS

	pág.
1. Logo OpenStack	17
2. Requisitos OpenStack HA	25
3. Topología de Red GID-Conuss	26
4. Distribucion de software en Lactoria y Nautilus	27
5. Distribucion de software en Labroides y Sistemas	28
6. Modelo Conceptual	29
7. Modelo Lógico	30
8. Interfaces de Lactoria	32
9. Hosts	32
10. Configuración de Galera en Lactoria	33
11. Bases de datos.	34
12. Cola de mensajes con RabbitMQ	34
13. Modelo OpenStack HA	36
14. Virtual IP configurada	36
15. Inicio de sesion en la plataforma	39
16. Terminos y condiciones de uso GID-CONUSS página 1.	40
17. Vista general de Administrador	41
18. Vista general de proyecto	41
19. Docker	44
20. Kubernetes Cluster	45
21. Ilustracion de un nodo de kubernetes	47
22. Imagenes docker almacenadas localmente	48
23. Dockerfile	49
24. Tabla de todos los pods del cluster	50
25. Pantalla inicial del dashboard de kubernetes	51
26. Graficas de heapster en el dashboard de kubernetes	52
27. Total de pods después de desplegar servicios de monitoreo	52
28. Dashboard de Grafana	53

29. Dashboard de Wave Scope	53
30. Logo de Nagios	55
31. Modelo lógico nagios	55
32. Distribución de software nagios	56
33. Archivo de configuracion del servidor Labroides en el servidor nagios	57
34. Archivo de configuracion del agente nrpe en Labroides	58
35. Ubicación de los scripts en el servidor nagios	59
36. Dashboard de Nagios	60
37. Script de Telegram	61
38. Notificaciones en la aplicación Telegram	62

LISTA DE TABLAS

pág.

1. Red Conuss	26
2. Interfaces	31

RESUMEN

TITULO: DETECCIÓN AUTOMÁTICA DEL NIVEL DE ESTRATIFICACIÓN SOCIOECONÓMICO URBANO USANDO REDES NEURONALES CONVOLUCIONALES SOBRE IMÁGENES SATELITALES CON INFORMACIÓN AUMENTADA.

AUTORES: DANIEL ALCIDES CARVAJAL PATIÑO.

PALABRAS CLAVE: Contenedores, Cloud Computing, OpenStack, modulos y servicios.

DESCRIPCION:

La finalidad de este proyecto de grado, es continuar con la investigación que el grupo Conuss ha venido desarrollando durante varios semestres en la creación de una infraestructura nube para la comunidad estudiantil, que permita su uso para el desarrollo de otros proyectos e investigaciones.

Debido a la necesidad de crear diplomados y laboratorios virtuales de computación para fomentar el avance en la formación de ingenieros de calidad, se decide utilizar soluciones de código abierto que permitan el fácil acceso y administración de los servidores físicos y virtualización. Entre las muchas soluciones, se decide optar por OpenStack gracias a su amplia gama de módulos y su abundante comunidad por el cual es respaldado, a su vez, integrando docker como solución para la creación de contenedores.

Todo esto con el fin de que la comunidad estudiantil tenga acceso a recursos de computo que no están al alcance de sus manos, otorgándoles la capacidad de conocer, además de disfrutar, las nuevas tecnologías que hoy por hoy están mejorando y automatizando los procesos de las grandes industrias tecnológicas.

ABSTRACT

TITLE: AUTOMATIC DETECTION OF THE URBAN SOCIOECONOMIC STRATIFICATION LEVEL USING CONVOLUTIONAL NEURAL NETWORKS ON SATELLITE IMAGES WITH INCREASED INFORMATION.

AUTHORS: DANIEL ALCIDES CARVAJAL PATIÑO.

KEYWORDS: Container, Cloud Computing, OpenStack, modules and services.

DESCRIPTION:

The purpose of this thesis is to continue with the research that the Conuss group has incorporated during several semesters in the creation of an infrastructure for the student community that allows its use for the development of other projects and research.

Due to the need to create certified courses and virtual computer labs to promote the advancement in the training of quality engineers, it is decided to use open source solutions that allow easy access and administration of physical servers and virtualization. Among the many solutions, it is decided to opt for OpenStack thanks to its wide range of modules and its abundant community for which it is backed, in turn, integrating docker as a solution for the creation of containers.

All this in order that the student community has access to computing resources that are not available to them, that they can know as well as enjoy the new technologies that are improving today and automating the processes of the large technological industries.

INTRODUCCION

La medición del nivel económico de una zona urbana, actualmente, conlleva un trabajo extenso, como lo expresa el DANE, “en el caso de las revisiones generales urbanas, así como en la estratificación rural se apoya en censos de vivienda”¹. Es decir, se requiere la elaboración de una encuesta de gran tamaño, la cual consume mucho tiempo y personal. Posteriormente, si la encuesta no se realizó usando software de recolección de datos, es necesario realizar su tipeo. Según la metodología que usa el DANE²³, el cálculo final del estrato se realiza mediante modelos estadísticos y económicos especialmente calibrados para esta tarea.

En este contexto surgen varias interrogantes respecto a la capacidad de actualización de esta metodología: ¿Qué sucede cuando una ciudad tiene una alta tasa de desarrollo urbano?, ¿Cómo mantiene el gobierno actualizada la información de los estratos ante éstas circunstancias?, ¿Qué tan efectiva es la metodología actual ante estos casos de alto desarrollo urbano?

Por tanto, el objetivo de este trabajo seleccionar redes neuronales convolucionales y evaluar su capacidad para determinar automáticamente el estrato socioeconómico usando imágenes satelitales e información adicional (información catastral, presencia y consumo de servicios, etc.). No es la primera vez que se realiza una predicción del nivel social, han habido varias. Neal Jean en colaboración con otras personas y varias instituciones realizó un modelo ⁴⁵⁶, para predecir la pobreza en cinco países de África usando imágenes satelitales y datos extra para dicha tarea. En Colombia, más específicamente en

¹DANE. Estratificación - Preguntas frecuentes. [en línea]. (Recuperado en 05 oct 2017). Disponible en https://www.dane.gov.co/files/geoestadistica/Preguntas_frecuentes_estratificacion.pdf.

²DANE. Metodología de estratificación. [en línea]. (Recuperado en 05 oct 2017). <http://www.dane.gov.co/index.php/servicios-al-ciudadano/servicios-de-informacion/estratificacion-socioeconomica>.

³DANE. Procedimiento del cálculo. [en línea]. (Recuperado en 05 oct 2017). <http://www.dane.gov.co/files/geoestadistica/estratificacion/procedimientoDeCalculo.pdf>.

⁴NEAL jean. Combining satellite imagery and machine learning to predict poverty. [en Linea]. (Recuperado en 10 oct 2017) <http://sustain.stanford.edu/predicting-poverty/>

⁵NEAL jean. Combining satellite imagery and machine learning to predict poverty. [en linea]. (Recuperado en 10 oct 2017) <https://github.com/nealjean/predicting-poverty>

⁶NEAL Jean, MARSHALL Burke, † MICHAEL Xie, W. Matthew Davis, DAVID B. Lobell, STE-

Medellín, también se han realizado modelos⁷⁸o estudios para determinar los índices de pobreza en dicha ciudad, uno fue elaborado por Miguel Noreña, aunque este estudio no se centra en la utilización de técnicas de machine learning si se centra en la predicción de la pobreza. La diferencia del proyecto actual, con los proyectos aquí mencionados es lograr la predicción del “estrato social Colombiano” mediante el uso de técnicas de Deep Learning como lo son las CNN, por sus sigls en ingles. Convolutional Neural Network

FANO Ermon. Combining satellite imagery and machine learning to predict poverty. Science 353 (6301), p. 790-794. 2016

⁷EAFIT. Con imágenes satelitales miden los índices de pobreza en Medellín. [en linea]. (Recuperado en 10 oct 2017) <http://www.eafit.edu.co/investigacion/revistacientifica/edicion-167/Paginas/con-imagenes-satelitales-miden-los-indices-de-pobreza-en-medellin.aspx>

⁸NOREÑA Miguel. Detección y caracterización de zonas marginales en la ciudad de Medellín mediante el análisis exploratorio de datos espaciales [en linea]. (citado: 10 oct 2017) http://www.banrep.gov.co/sites/default/files/eventos/archivos/TesisMiguelNorena_0.pdf

1. OBJETIVOS

1.1. OBJETIVO GENERAL

1. Seleccionar y evaluar redes convolucionales para la determinación del nivel socio económico urbano mediante el uso de imágenes satelitales e información adicional.

1.2. OBJETIVOS ESPECIFICOS

1. Identificar fuentes de datos de imágenes satelitales e información adicional.
2. Diseñar y construir datasets integrando los datos obtenidos de las fuentes identificadas.
3. Seleccionar entre distintas arquitecturas de redes neuronales convolucionales existentes en la literatura y repositorios tecnológicos .
4. Entrenar las redes convolucionales probando configuraciones de datasets.
5. Evaluar el desempeño de las redes convolucionales con el uso de los distintos dataset.
6. Elegir la mejor configuración tanto de red convolucional como de conjunto de datos, teniendo en cuenta el desempeño obtenido.

2. MARCO TEÓRICO

2.1. ESTRATO SOCIAL

A través de los años el paradigma de la computación en la nube ha permitido que las empresas tengan un gran beneficio al ahorrarse múltiples costos en los departamentos de TI y organización en general, costos como el almacenamiento, procesamiento, redes y muchas otras, todo esto gracias a otras compañías que proveen estos servicios por un coste mensual que se establece según el tipo de servicio requerido y la cantidad de este mismo. Es decir las compañías proveedoras cobran por el número de gigabytes de almacenamiento que la organización vaya a requerir, así mismo funciona para el número de núcleos de procesador y gigabytes de ram entre otros. Actualmente existen diferentes tipos de nubes que proveen distintos tipos de servicio, estos pueden ser clasificados de la siguiente manera:

- IaaS - Infraestructura as a service:

Infraestructura como servicio es una forma de computación en la nube donde se ofrecen a los clientes recursos, físicos y virtuales, como máquinas virtuales, cortafuegos, sistemas de almacenamiento o平衡adores de carga, entre otros. Para poder ofrecer estos elementos se utilizan hipervisores como Xen, KVM, VMware ESX / ESXi o Hyper-V, entre otros. IaaS es la parte elemental de la computación en la nube, ya que es la que se encarga de proporcionar los recursos informáticos sobre los que se seguirán implementando el resto de conceptos.¹

- Paas - Infraestructure as a Service:

Plataforma como servicio es un concepto de computación en la nube mediante la cual los usuarios pueden desarrollar, ejecutar y administrar aplicaciones sin preocuparse por la infraestructura que haya por debajo. De esta manera, los desarrolladores solo tienen que preocuparse por la programación de las aplicaciones, nunca por la configuración ni el hardware que hay por debajo, ahorrando tiempo

¹IaaS.

y recursos. Por ejemplo, PaaS puede ejecutarse por encima del IaaS en máquinas físicas e incluso en contenedores.²

- SaaS - Software as a Service:

Software como servicio es un modelo de distribución de software por el que terceros desarrolladores ofrecen ciertas aplicaciones a través de Internet accesibles sólo a través de un cliente propio. Aunque no lo sepamos, este tipo de computación en la nube la utilizamos casi a diario, por ejemplo, al conectarnos a Twitter o Facebook desde sus respectivos clientes o al ver un vídeo o una película en streaming, por ejemplo, desde Netflix.³

2.2. MACHINE LEARNING

Para entrar en el amplio mundo de la computación en la nube, debemos conocer algunos conceptos básicos:

La computación en la nube es un paradigma informático que permite utilizar las aplicaciones comunes de un computador y ofrecerlas como servicio, es decir, bases de datos, firewall, recursos de cómputo y almacenamiento, entre otros. Los usuarios de estos servicios pueden hacer uso de ellos a través de internet en donde además podrán tener un control total de estos servicios que hayan adquirido.

Para ello se utiliza la tecnología de virtualización, partiendo de que un servidor cuenta con un hypervisor, es decir, una capa de abstracción del hardware para corresponder a un sistema operativo invitado (véase virtualbox, qemu-kvm, vmware) en máquinas virtuales. También existe una alternativa a las máquinas virtuales, los contenedores, en los cuales se utiliza un almacenamiento compartido entre el host y el contenedor, y se ejecuta un servicio aisladamente del resto del sistema operativo, proporcionando una fácil ejecución en cualquier entorno físico.

²IaaS.

³IaaS.

2.2.1. openstack es un proyecto que integra varios módulos para conformar una infraestructura de nube que controla grandes cantidades de recursos computacionales, de almacenamiento y de red en todo un centro de datos, esto es posible de administrar gracias a un panel que brinda control a los administradores y permite a los usuarios aprovisionar recursos a través de una interfaz web. al ser un sistema operativo en la nube, requiere de la comunicación de muchos servicios a través de la red entre distintos hosts. Estos servicios reciben el nombre de módulos, pues su implementación se puede realizar en un host independiente cada uno, y utilizar un protocolo de intercambio de datos, permitiendo escalabilidad.

1

Figura 1: Logo OpenStack



Fuente: [openstackIMG](#)

2.2.1.1. keystone: es uno de los componentes centrales de OpenStack, ya que comunica casi todos los servicios, ofrece el servicio de identidad integrando servicios de autenticación, catálogo y creación de políticas para administrar los usuarios "tenants." en los proyectos de OpenStack.

Podemos observar a Keystone como si estuvieramos en un parque de atracciones. Keys-

token otorga un token, un boleto para acceder a las distintas atracciones, cada vez que se quiere subir a alguna atracción, se debe mostrar el boleto para que sea validado. Una vez validado se puede ingresar en la atracción y disfrutar de ella. En OpenStack, Keystone autentica cada usuario o servicio que quiera hacer uso de un servicio de OpenStack.

KEYSTONE no solo posee autenticación por token, tambien posee autenticacion por nombre de usuario / contraseña, y el uso de backends tales como LDAP (lighweigth access directory) y PAM (Pluggable Authentication Modules)

2.2.1.2. glance GLANCE es el servicio de imagenes, funciona como un almacenamiento para los archivos de distintos formatos de imagenes de discos virtuales tales como iso, qcow2, raw, vdi, vmdk, entre otros. Con Glance se busca un facil acceso a las imagenes para lanzar instancias de una maquina virtual o un contenedor. Glance puede utilizar como almacenamiento otro tipo de back-ends como Swift, ya que por defecto viene configurado para utilizar un directorio de archivos en el sistema.

Hay que tener en cuenta que Nova utiliza un almacenamiento efímero al hacer uso de las imágenes de Glance, cuando una instancia se apaga, se pierden todos los cambios realizados hasta ese instante. Para ello se asignan volúmenes persistentes a las máquinas virtuales a través de Cinder.

2.2.1.3. nova NOVA es el modulo administrador de las maquinas virtuales y los recursos del host en openstack, y se compone de cinco elementos:

nova-api: es el componente que permite la comunicación con el usuario final y otros usuarios del sistema, permitiendo que lancen instancias y las manipulen via OpenStack API o EC2 API, tambien descarga las imágenes desde Glance para poder utilizarlas en las instancias.

nova-compute: es el componente encargado de inicializar y terminar instancias segun el API del hypervisor instalado, XenAPI para XenServer, Libvirt para KVM y VMWare

API para VMWare.

nova-scheduler: este elemento revisa cada host hypervisor para tener conocimiento de donde es mejor lanzar una instancia en base a los recursos del sistema.

nova-conductor: este elemento soporta la conexión de nova con la base de datos.

nova-novncproxy y **nova-consoleauth:** son elementos que nos permiten visualizar la consola de la máquina virtual desde el navegador mediante un proxy.

2.2.1.4. neutron ES el modulo de red como servicio, el cual nos permite la creación de redes internas y externas, creación de puertos y relacionarlos con interfaces de máquinas de nova, direccionamiento ipv4 e ipv6, creación de routers y asignación de ips flotantes, creación de switch virtuales y de mas configuraciones de red.

2.2.1.5. cinder MÓDULO que permite Block Storage, es decir, creación de volúmenes fijos de almacenamiento persistente que pueden ser montados en las máquinas virtuales, así como la clonación de datos para realizar backups.

2.2.1.6. heat ES el modulo de OpenStack encargado de utilizar plantillas para orquestar la creación de clusters, instancias, redes, y usuarios.

2.2.1.7. magnum ES el modulo de OpenStack que integra a Heat, Docker, y Kubernetes o Docker Swarm, para la inicialización de clusters en donde se desplieguen los contenedores.

2.2.2. alta disponibilidad LA redundancia es un mecanismo usado para mitigar fallos, ya que es un factor muy importante a tener en cuenta cuando se desea

implementar una infraestructura de nube. Los elementos físicos pueden fallar en cualquier instante, y la nube debe garantizar al usuario que sus datos no se pierdan en el camino. Para ello se trabaja en la alta disponibilidad de OpenStack, utilizando algunas herramientas software externas que permiten redundancia de datos en el sistema.

Hay dos modos de alta disponibilidad, activo / activo y activo/pasivo, la primera es la alta disponibilidad recomendada para sitios web, ya que permite el balanceo de carga entre las máquinas, disminuyendo así el volumen de conexiones entrantes, pero también porque es facil implementarlo en servicios stateless, mientras que el activo/pasivo es recomendado para aquellos servicios que son stateful, servicios los cuales responden en base a un estado.

En el modo activo / activo los nodos se mantienen ofreciendo los mismos servicios en el mismo instante de tiempo, mientras que en activo/pasivo, un nodo ofrece el servicio mientras el otro se queda en espera de que el primer nodo falle.

2.2.2.1. alta disponibilidad con pacemaker y corosync PACEMAKER es una herramienta de código abierto que permite manipular recursos para clusters en alta disponibilidad.

Pacemaker se ayuda de Corosync, un framework para alta disponibilidad que monitorea los servicios de varios nodos para actuar en base a una directiva planteada con Pacemaker.

Un ejemplo de esto es la asignación de una IP virtual a dos nodos, uno de los nodos recibe las solicitudes de conexión por tal IP mientras el otro se mantiene en un estado pasivo y no recibe las conexiones, ó, ambos pueden recibir las conexiones, segun como se especifique la manera de tratar el recurso por Pacemaker.

2.2.2.2. replica de datos con lsync LSYNC es un demonio que utiliza las funciones de rsync con ssh para sincronizar carpetas de un directorio. Es de vital

importancia para aquellas soluciones en las que los discos no están formateados para la réplica iSCSI en línea, garantizando alta disponibilidad en clusters pequeños. Para un cluster de tamaño considerable, se recomienda utilizar alternativas como Ceph RADOS.

2.2.2.3. virtualizacion LA virtualización es una tecnología que permite la simulación de recursos de hardware como disco, memoria ram y procesador para crear una maquina virtual que se ejecutará sobre un host utilizando un software hipervisor.

2.2.2.4. contenedores LOS contenedores son instancias de máquinas virtuales que permiten la ejecución de procesos de aplicaciones de software utilizando menos recursos computacionales y haciendo uso del sistema operativo del host. Haciendo una analogía con la programación orientada a objetos (POO) las máquinas virtuales son clases y los contenedores son objetos de esas clases.

2.3. DEEP LEARNING

2.4. MACHINE LEARNING Y EL ESTRATO SOCIAL

3. METODOLOGIA

El desarrollo de éste proyecto consta de las siguientes fases o actividades para su realización:

FASE 1: Inspección del hardware y software requeridos para llevar a cabo el proyecto.

Actividad 1: Revisar estado del arte del cloud computing, así como papers, y material educativo que permita orientar de manera efectiva la realización del proyecto.

Actividad 2: Determinar los requisitos mínimos y recomendados del hardware para la debida instalación del software.

FASE 2: Implementación base I.

Actividad 1: Preparación de las máquinas físicas, actualización del firmware de los servidores, conexión física de la red de acuerdo al mapa lógico, establecimiento de VLAN administrativa, de usuarios, y de pruebas.

Actividad 2: Determinar y organizar el software que contendrá cada nodo.

FASE 3: Implementación base II.

Actividad 1: Definición de los servicios a implementar de OpenStack.

Actividad 2: Implementación de cada servicio de OpenStack en los nodos disponibles segun la FASE 2.

Actividad 3: Configuración de la conexión entre los diferentes módulos de OpenStack para alta disponibilidad.

FASE 4: Implementación de Orquestación de contenedores.

Actividad 1: Determinar las conexiones requeridas para implementar Kubernetes en OpenStack.

Actividad 2: Implementación de Kubernetes junto a OpenStack y su respectiva configuración.

FASE 5: Implementación del sistema de notificaciones.

Actividad 1: Instalar y configurar Nagios.

Actividad 2: Implementar envío de notificaciones a Telegram desde Nagios.

TIEMPO EXTENDIDO: Tiempo acumulado por tolerancia a fallos.

FASE FINAL: Elaboración de libro en base al proyecto realizado para su posterior publicación con el objetivo de difundir el conocimiento.

NOTA: Las labores administrativas respectivas del Grupo GID-CONUSS no están listadas en las fases anteriormente descritas, se anexará una bitácora al final del proyecto con las actividades administrativas realizadas.

4. DESARROLLO DEL PROYECTO

Para iniciar el proyecto fue conveniente partir de cero, es decir, en base a la documentación del proyecto que se tenía previamente desarrollado por los ex compañeros, los cuales habían realizado un prototipo de instalación de OpenStack en máquinas virtuales, se determina que el hardware de los servidores físicos está bastante desactualizado en comparación a los últimos controladores disponibles para dicho sistema, también se encuentra la ventaja de tener las instancias de OpenStack en sistemas físicos en vez de virtualizados, debido a que los servicios virtualizados no tienen aceleración de hardware, esto implica que se pierda rendimiento en las instancias (Funcionarían a 32 bits en un servidor de 64 bits). Esto es considerado la **FASE 1** del proyecto.

4.1. FASE 1

Para tener una buena base sobre el tipo de proyecto que se va a desarrollar, fue necesario como primera medida la investigación de las nuevas tecnologías que hay en el mundo del cloud computing empezando desde las redes informáticas hasta llegar a los contenedores, de los cuales se hablará más adelante. Teniendo en cuenta que la computación en la nube es un tema demasiado extenso, desde el punto de vista de las tecnologías de la información, se logró llegar a tener las bases necesarias para continuar con el proyecto. Una infraestructura de nube se puede resumir en 3 niveles: Infraestructura como servicio (IaaS), Plataforma como servicio (PaaS) y Software como servicio (SaaS), descritos en el estado del arte de este libro.

Por buenas prácticas es recomendable utilizar la documentación del propio software para conocer los requisitos que se deben tener en cuenta a la hora de empezar un proyecto desde cero, en este caso se usó la documentación de OpenStack.

La siguiente figura muestra los requisitos mínimos otorgados por el proyecto de OpenStack para la construcción de una infraestructura de alta disponibilidad, sin embargo, como se observa en las tablas 1-4, se demuestra que los servidores del grupo aprueban

estas consideraciones.

Figura 2: Requisitos OpenStack HA

Node type	Processor Cores	Memory	Storage	NIC
controller node	4	12 GB	120 GB	2
compute node	8+	12+ GB	120+ GB	2

Fuente requisitos

Los nodos controlador y compute de OpenStack son el núcleo principal de una infraestructura por esta razón se le da mayor importancia a los recursos que éstas maquinas requieren en cuanto a procesamiento y memoria RAM.

4.2. FASE 2

Esta fase consiste en actualizar los controladores y el BIOS en tres servidores físicos (Sistemas,Lactoria y Labroides) quienes son miembros de la infraestructura tradicional del grupo Conuss. Posteriormente formatear y realizar la instalación limpia del sistema operativo Ubuntu Server 16.04.4 LTS (escogido por la facilidad y experiencia utilizando esta distribución de linux) en los servidores Lactoria, Labroides y Nautilus (Servidor otorgado por el Grupo Radiogis). El servidor sistemas quedó como punto de inflexión para no perder las conexiones de los servicios prestados, mientras se colocaba en producción OpenStack en alta disponibilidad.

La actualización de los drivers y las BIOS requirieron de una búsqueda exhaustiva en la red debido a que ya que no se encuentran con facilidad pues los servidores cuentan con varios años de uso y su tecnología se ha venido quedando atrás.

Una vez se tienen actualizadas las maquinas se procede a enlazarlos en una red LAN creada por el Switch 3Com y gestionada por el DSI (División de Sistemas de Informa-

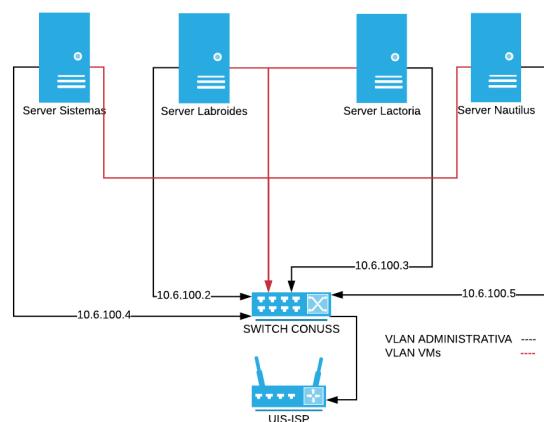
ción) de la Universidad Industrial de Santander, para este caso en específico por ser el grupo Conuss un grupo de investigación en cloud computing, servidores y servicios, se le fue asignado un total de tres VLANS de tal manera que puedan ser aprovechadas por los administradores para la configuración, gestión y administración de la red interna de Conuss. A continuación se da a conocer las tres VLANS y su respectivo uso.

Tabla 1: Red Conuss

Red Conuss			
	VLAN 1	VLAN 2	VLAN3
Direccion	10.6.100.0	10.6.101.0	10.6.102.0
Mascara	255.255.255.0	255.255.255.0	255.255.255.0
Gateway	10.6.100.1	10.6.101.1	10.6.102.1
Uso	Administración	Usuarios	Pruebas

La topología de red de Conuss se puede observar en la **Figura 3**.

Figura 3: Topología de Red GID-Conuss

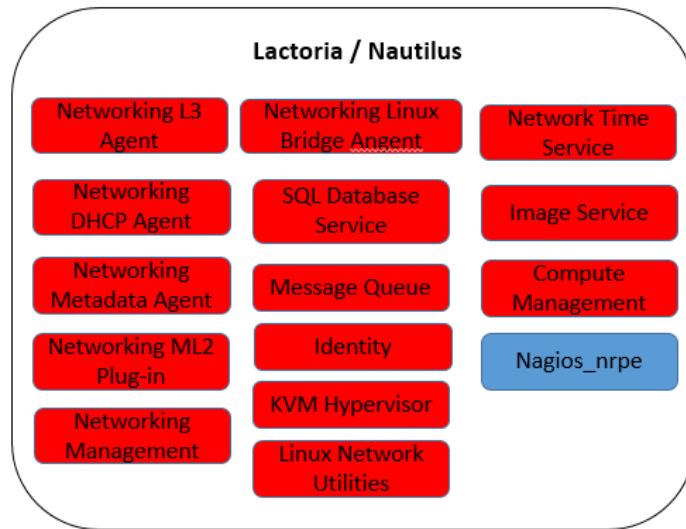


Fuente: [infraconuss](#)

La instalación de los módulos de OpenStack depende del entorno al que se quiera

adecuar la infraestructura, para el caso del GID-Conuss basta con la configuración base, la cual incluye los siguientes módulos presentes en las **FIGURAS 4 y 5**.

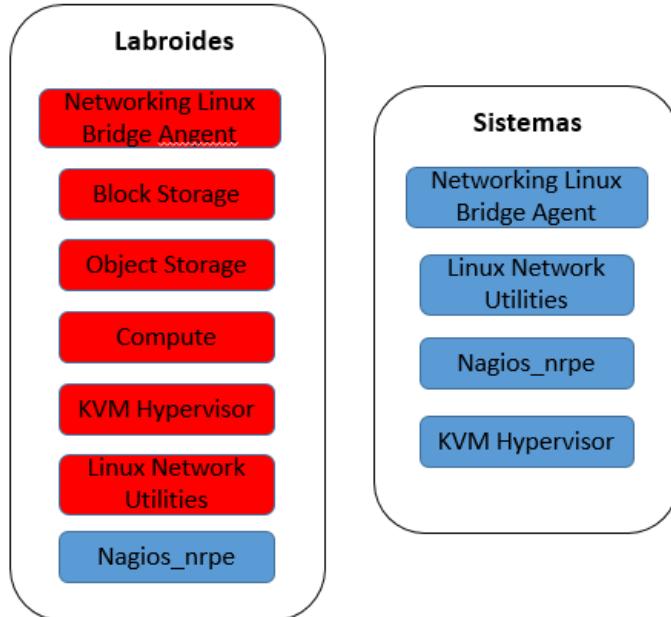
Figura 4: Distribucion de software en Lactoria y Nautilus



Fuente: propia

Debido a que el objetivo de dicha implementación es la alta disponibilidad, se requieren los mismos servicios en los nodos controladores (Lactoria y Nautilus) de esto se hablará más adelante.

Figura 5: Distribucion de software en Labroides y Sistemas



Fuente: propia

Los servicios que están en rojo hacen referencia a la infraestructura de OpenStack y los azules a otros servicios requeridos por el proyecto.

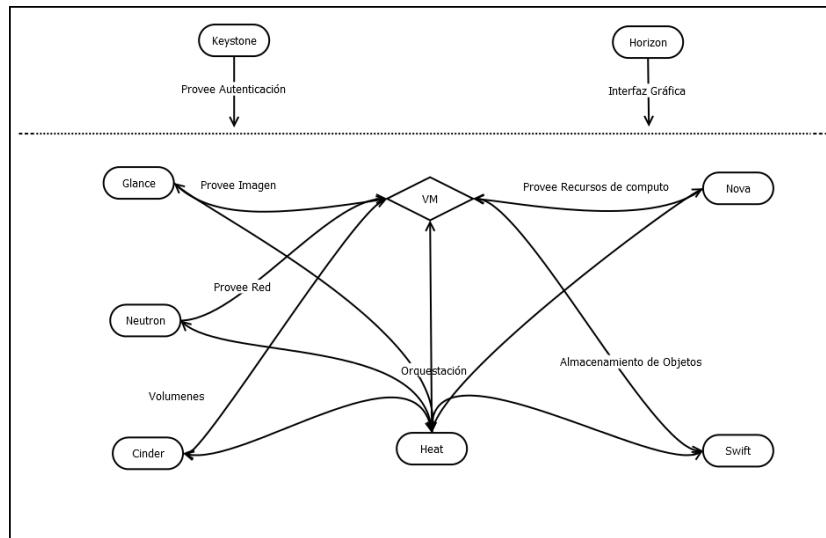
Labroides y Sistemas tienen la tarea de alojar las máquinas virtuales que harán parte del cluster de Kubernetes para los contenedores de Docker.

En los 4 servidores existe el servicio de nagios-nrpe que se encarga de informarle al servidor nagios sobre el estado de los recursos del servidor y otros servicios, esto se explicará en la fase 5.

4.3. FASE 3

En la **FASE 2** se determinaron los servicios a implementar en los servidores. Para la posterior instalación se elaboró un diagrama conceptual que permitiera ver de manera abstracta la implementación.

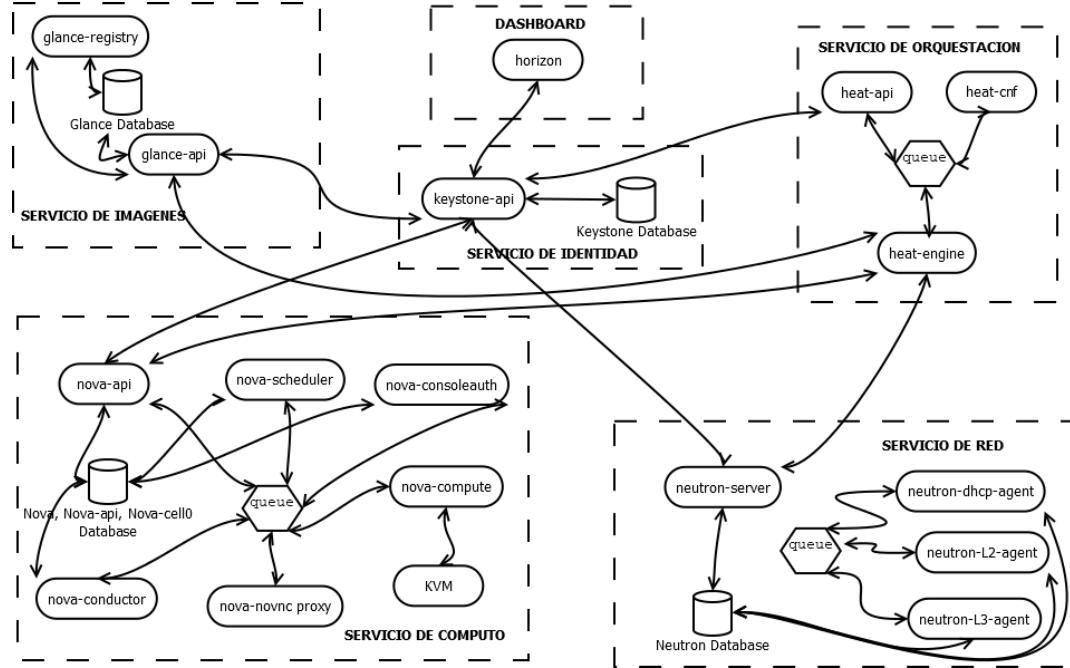
Figura 6: Modelo Conceptual



Fuente: propia

Para una vista mas detallada de lo visto en la **FIGURA 6** se realizó un modelo lógico (**FIGURA 7**) que abarca más a fondo la manera de operar de OpenStack entre sus módulos; cabe aclarar que no se incluyeron los servicios de Nagios para el monitoreo en estos modelos porque se consideran agentes externos a la implementación de OpenStack como tal.

Figura 7: Modelo Lógico



Fuente: propia

4.3.1. hosts La implementación comienza configurando cada nodo para que reconozca a los demás nodos en el archivo `/etc/hosts`. De ésta manera será más factible referenciarlos en las URLs que maneja OpenStack en su interior.

4.3.2. api rest OpenStack utiliza para su comunicación interna el protocolo HTTP, debido a que la totalidad de sus módulos poseen una API REST quien les da el carácter modular, es decir, sus estados y/o ejecuciones varían referentes a los encabezados HTTP recibidos tales como los métodos POST y GET.

4.3.3. interfaces En el archivo `/etc/network/interfaces` se configuran las interfaces para poder comunicarse. Cada nodo tiene 2 interfaces conectadas a excepción de

Sistemas, quien solo necesita un puente para darle red a sus hosts, ya que son máquinas virtuales prestadas en la antigua infraestructura.

Las interfaces de los nodos de OpenStack están definidas como en la **TABLA 7**.

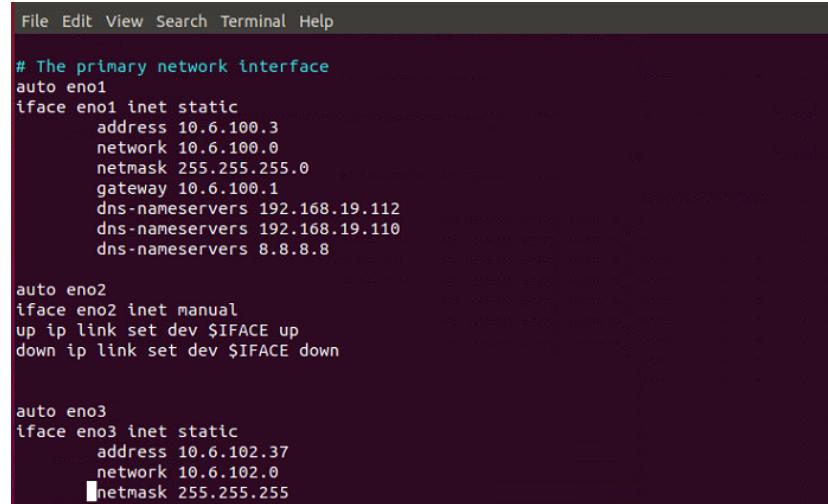
Tabla 2: Servidores e Interfaces

Interfaces de Red		
	Interfaz 1	Interfaz 2
Labroides	10.6.100.2	Proveedor de 10.6.101.0/24
Lactoria	10.6.100.3	Proveedor de 10.6.101.0/24
Nautilus	10.6.100.5	Proveedor de 10.6.101.0/24
Sistemas	10.6.100.4	N/A

La interfaz proveedora de define de la siguiente manera en Ubuntu Server 16.04 LTS:

```
auto INTERFACE_NAME
iface INTERFACE_NAME inet manual
    up ip link set dev IFACE up
    down ip link set dev IFACE down
```

Figura 8: Interfaces de Lactoria



```
File Edit View Search Terminal Help

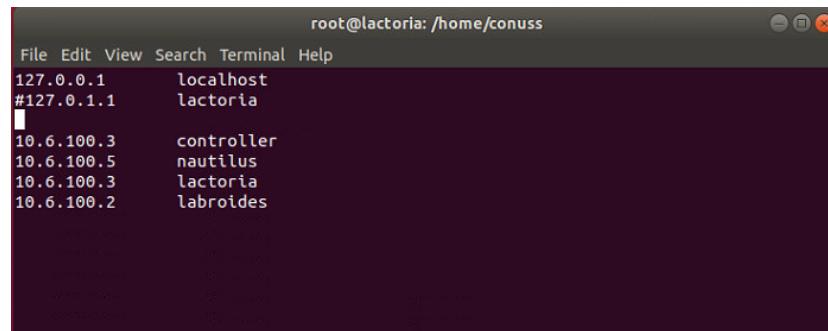
# The primary network interface
auto eno1
iface eno1 inet static
    address 10.6.100.3
    network 10.6.100.0
    netmask 255.255.255.0
    gateway 10.6.100.1
    dns-nameservers 192.168.19.112
    dns-nameservers 192.168.19.110
    dns-nameservers 8.8.8.8

auto eno2
iface eno2 inet manual
    up ip link set dev $IFACE up
    down ip link set dev $IFACE down

auto eno3
iface eno3 inet static
    address 10.6.102.37
    network 10.6.102.0
    netmask 255.255.255.0
```

Fuente: propia

Figura 9: Hosts



```
root@lactoria: /home/conuss
File Edit View Search Terminal Help
127.0.0.1      localhost
#127.0.1.1      lactoria
#
10.6.100.3      controller
10.6.100.5      nautilus
10.6.100.3      lactoria
10.6.100.2      labroides
```

Fuente: propia

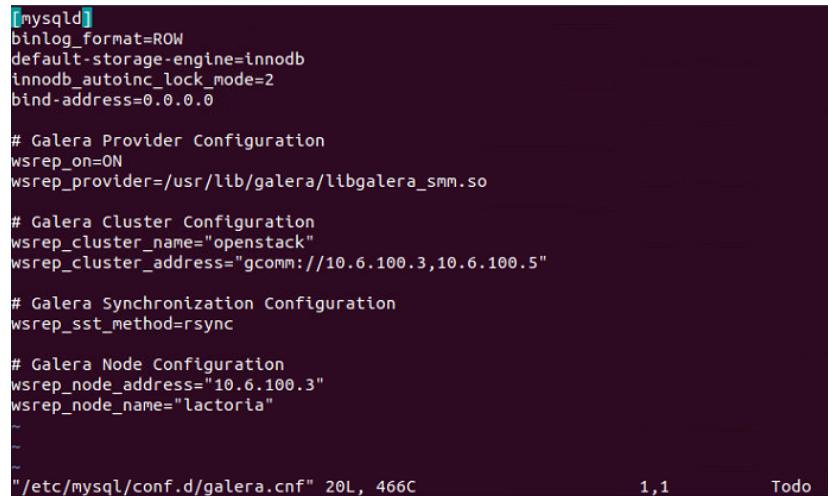
4.3.4. mariadb galera cluster Una vez configurada las interfaces de los servidores, se procede a configurar el servicio de Network Time Protocol (NTP) para sincronizar la hora entre los servidores, y posteriormente se registra el repositorio de Ubuntu Cloud OpenStack Queens, pues es la versión estable más reciente de OpenStack. También es necesario añadir el repositorio de MariaDB Galera Cluster, pues la

versión del repositorio de Ubuntu solo llega hasta la 10.0, versión que no tiene soporte para la replicación entre nodos (WSREP).

Es importante tener en cuenta que las bases de datos solo se instalan en los nodos controladores, por ende, solo Lactoria y Nautilus tendrán base de datos instaladas y sincronizadas con Galera.

Para sincronizarlas basta con editar el archivo de configuración /etc/mysql/conf.d/galera.cnf y habilitar la replicación entre nodos.

Figura 10: Configuración de Galera en Lactoria



```
[mysqld]
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0

# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so

# Galera Cluster Configuration
wsrep_cluster_name="openstack"
wsrep_cluster_address="gcomm://10.6.100.3,10.6.100.5"

# Galera Synchronization Configuration
wsrep_sst_method=rsync

# Galera Node Configuration
wsrep_node_address="10.6.100.3"
wsrep_node_name="lactoria"
~

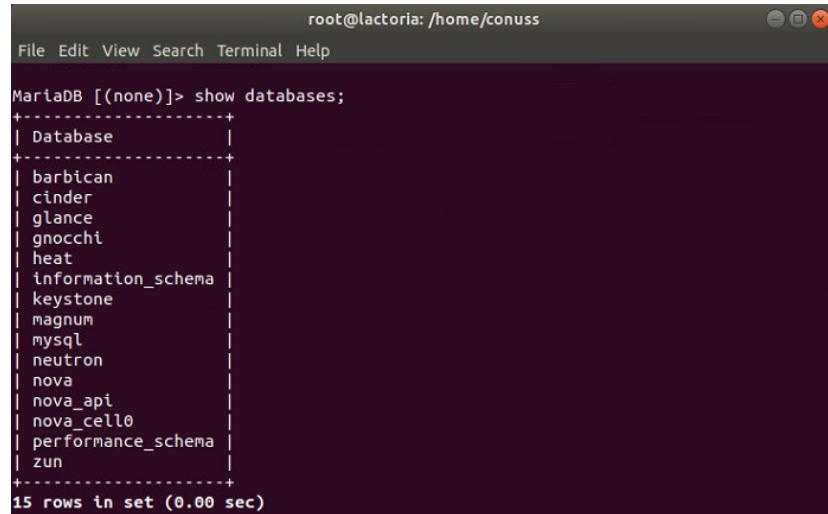
"/etc/mysql/conf.d/galera.cnf" 20L, 466C
```

1,1 Todo

Fuente: propia

A medida que se van instalando los módulos de OpenStack, se deben ir creando las bases de datos que contendrán la información de los mismos y otorgándoles privilegios para que puedan ser accedidos; cada servicio tiene sus credenciales usuario:contraseña para hacer uso de su respectiva base de datos.

Figura 11: Bases de datos.



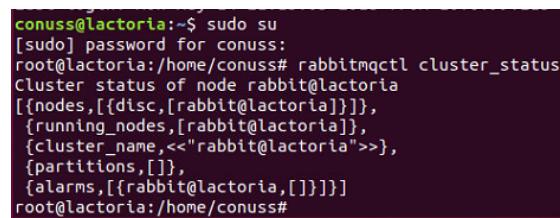
A screenshot of a terminal window titled "root@lactoria:/home/conuss". The window shows the output of the command "show databases;". The output lists 15 databases: barbican, cinder, glance, gnocchi, heat, information_schema, keystone, magnum, mysql, neutron, nova, nova_api, nova_cell0, performance_schema, and zun. At the bottom of the output, it says "15 rows in set (0.00 sec)".

```
MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| barbican |
| cinder   |
| glance   |
| gnocchi  |
| heat     |
| information_schema |
| keystone |
| magnum   |
| mysql    |
| neutron  |
| nova    |
| nova_api |
| nova_cell0 |
| performance_schema |
| zun      |
+-----+
15 rows in set (0.00 sec)
```

Fuente: propia

4.3.5. rabbitmq Los procesos de OpenStack necesitan un intermediario que los comunique, es aquí donde entra el concepto de Colas de Mensajes, los modulos de OpenStack envian sus peticiones a la cola de mensajes, y permanecen allí hasta que sean atendidas. La comunicación por medio de RabbitMQ se asemeja a el encaminamiento de un router, utilizando determinadas claves para poder dirigirse a un servicio en específico.

Figura 12: Cola de mensajes con RabbitMQ



A screenshot of a terminal window titled "conuss@lactoria:~\$ sudo su". The window shows the output of the command "rabbitmqctl cluster_status". The output displays the cluster status of node "rabbit@lactoria", including nodes, running nodes, cluster name, partitions, and alarms.

```
conuss@lactoria:~$ sudo su
[sudo] password for conuss:
root@lactoria:/home/conuss# rabbitmqctl cluster_status
Cluster status of node rabbit@lactoria
[{"nodes": [{"disc": [{"node": "rabbit@lactoria"}]}], "running_nodes": [{"node": "rabbit@lactoria"}], "cluster_name": "<rabbit@lactoria>", "partitions": [], "alarms": [{"node": "rabbit@lactoria"}]}
root@lactoria:/home/conuss#
```

Fuente: propia

RabbitMQ maneja cola de mensajes de alta disponibilidad, que no es más que clonar

una cola de mensajes ya especificada, y reenviada a los nodos competentes para que la procesen al mismo tiempo.

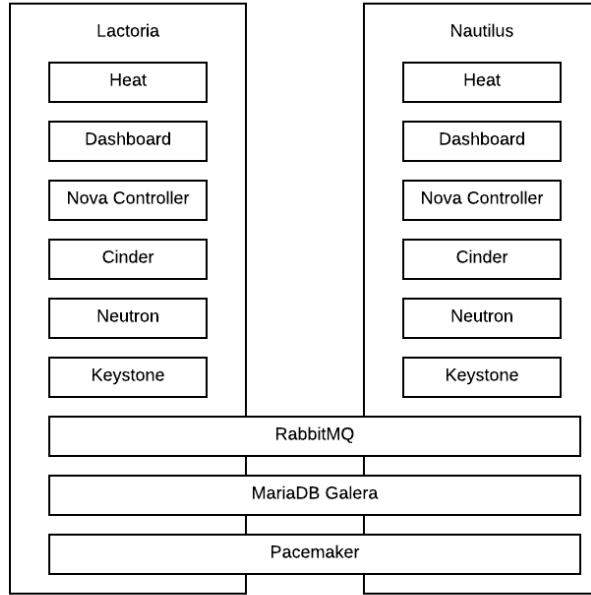
4.3.6. memcached Memcached es utilizado por el servicio de identidad para guardar las tokens que le asigna a los usuarios y servicios autenticados. El servicio utiliza la memoria ram de los nodos controladores para crear una especie de almacenamiento compartido en donde consultar objetos de manera breve.

4.3.7. etcd CoreOS nos ofrece una solución para mantener a los controladores al tanto de la vida de los servicios ya que dispone de un almacén de clave-valor confiable y distribuido, se utiliza en otros casos como el bloqueo distribuido de claves y la configuración de almacenamiento.

Finalmente, acabados los requisitos para un correcto funcionamiento de OpenStack, se implementan los módulos de OpenStack. El secreto de los módulos de OpenStack está en sus archivos de configuración,

4.3.8. pacemaker, corosyn, heartbeat, la alta disponibilidad. y el almacenamiento compartido. La alta disponibilidad de OpenStack vendrá configurada por un cluster de Pacemaker, permitiéndonos configurar que servicios queremos como activos/pasivos y activos/activos.

Figura 13: Modelo OpenStack HA



Fuente: propia

Para la manipulación de pacemaker se utiliza crmsh, una aplicación de Corosync, quien nos permite crear reglas para controlar en que nodo se ejecuta un servicio, o si se quiere que el servicio se encuentre disponible en ambos nodos al tiempo (clone), a éstas reglas se les denomina **constraints** y son la base de la alta disponibilidad.

Figura 14: Virtual IP configurada

```
root@lactoria:/home/conuss# crm status
Last updated: Mon May 14 23:00:53 2018          Last change: Mon May 14 11:46:24
2018 by root via cibadmin on nautilus
Stack: corosync
Current DC: lactoria (version 1.1.14-70404b0) - partition with quorum
2 nodes and 1 resource configured

Online: [ lactoria nautilus ]

Full list of resources:

  virtual_public_ip     (ocf::heartbeat:IPAddr2):      Started lactoria
```

Fuente: propia

En la **figura 14** se muestra configurada la IP virtual de acceso a los nodos en modo activo/pasivo, iniciada en Lactoria, esto quiere decir que si se accede a la IP virtual, se estaría accediendo al nodo de Lactoria, y en caso de que éste fallase, se accedería a Nautilus.

La IP virtual solo es uno de los tantos recursos que se pueden configurar con pacemaker. Para manipular los demás servicios se pueden utilizar desde scripts en bash llamados agentes OCF obtenidos desde el paquete openstack-resource-agents de Ubuntu, hasta el mismo LSB de systemd.

4.3.9. servicio de identidad El servicio de identidad a.k.a Keystone, es el servicio que gestionará la base de datos para la creación de usuarios, asignación de endpoints, creación de grupos de seguridad, manejo de roles para los usuarios, proyectos y dominios.

4.3.10. servicio de imágenes Las imágenes son uno de los componentes principales en el ciclo de vida de una instancia de OpenStack, por ello se requiere que el servicio sea altamente disponible, es decir, siempre se tengan a la mano las imágenes de determinado sistema operativo por si un nodo falla.

Debido a que no se cuenta con una partición de almacenamiento compartido, se incorporó el servicio lsyncd, que permite la sincronización de dos directorios a través de rsync y por SSH.

Este servicio se encarga de replicar las imágenes y mantenerlas al día entre los nodos.

4.3.11. servicio de computo El servicio de computo altamente disponible nos permite disponer de evacuación de las máquinas virtuales en caso de fallo del nodo en el que se encuentran virtualizadas, ejecutando una rutina en donde lanza la maquina virtual con la misma configuración pero en otro nodo de cómputo.

4.3.12. servicio de red El servicio de red a.k.a. Neutron, involucra su alta disponibilidad al crear dos agentes de dhcp y de l3 por nodo, en caso de que un nodo falle, se tomará como referencia el otro nodo al ser levantada la máquina virtual de nuevo.

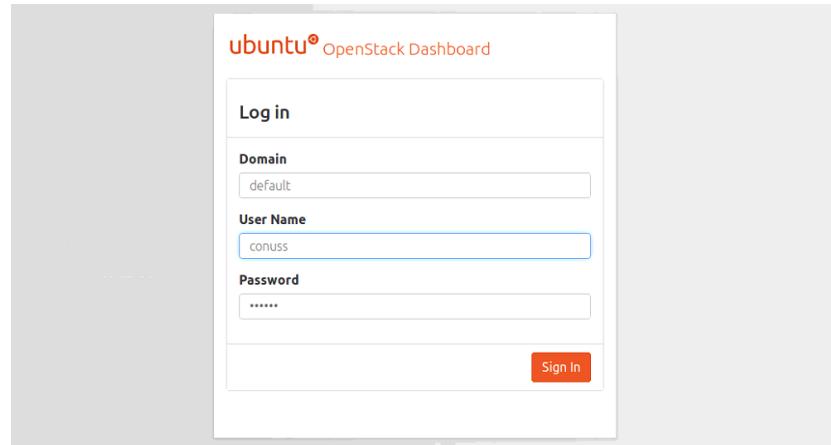
4.3.13. servicio de orquestacion Heat nos permite la creación de plantillas pre-configuradas para lanzar aplicaciones. Las plantillas de heat se conocen como stacks, y comienzan a llamar a los servicios necesitados según las instrucciones que posea, de ésta manera facilitamos la labor por parte de los administradores automatizando la provisión de los recursos.

4.3.14. servicio de almacenamiento de bloques El almacenamiento que provee Nova por defecto es efímero, es decir, en el momento en que la maquina sea destruida se perderá toda su información, por ende, es recomendable que en aquellas instancias en donde se quiera conservar la información, se trabajen volúmenes.

Con Cinder se configuran volúmenes lógicos en los servidores físicos para disponer de escalabilidad vertical en cuanto a almacenamiento se refiere, permitiendo así que sean asignados pequeños bloques a las máquinas virtuales también llamados volúmenes. Su labor principal es la de proveer un almacenamiento persistente.

4.3.15. dashboard de horizon Finalmente se puede observar la interfaz de acceso por parte de los usuarios a través de <http://10.6.100.10/horizon>.

Figura 15: Inicio de sesion en la plataforma



Fuente: propia

La interfaz requiere de las credenciales para el acceso otorgadas previamente por los administradores del grupo GID-CONUSS. Estas credenciales solo se otorgarán a los estudiantes y usuarios quienes acepten los términos y condiciones de uso, en los cuales se especifica claramente que GID-Conuss no se hace responsable del mal uso que se le pueda dar al servicio, y que llegado el caso, podrá realizar intervención alguna.

Figura 16: Terminos y condiciones de uso GID-CONUSS página 1.



cloudEISI
Documentación Administrativa
2016
Acuerdo de términos y condiciones de uso

Términos y Condiciones de Uso

Este acuerdo establece los términos y condiciones de uso aplicables para el señor **NOMBRE DE LA PERSONA RESPONSABLE** miembro de la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander y el responsable a cargo de la administración del recurso **NOMBRE DE PERSONA A CARGO DE LA ADMINISTRACION** en su uso de los servicios prestados (los "SERVICIOS") por el Grupo de Investigación y Desarrollo en Computación en la Nube, Seguridad, Servidores y Servicios ("GID-CONUSS") para la plataforma de infraestructura como servicio (IaaS) de computación en la nube cloudEISI.

El servicio prestado se encuentra asociado al dominio <http://cloudeisi.uis.edu.co> y consta de una instancia virtual identificada con:

Nombre de la máquina = GIGBA

Al usar hacer uso de los SERVICIOS, el USUARIO automáticamente acepta estos Términos y Condiciones de Uso y acuerda regirse por ellos.

1. **Usuarios elegibles.** Los servicios están abiertos para el uso de cualquier miembro de la comunidad académica de la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander.
2. **Cuenta de usuario, contraseña y seguridad.** Como parte de la creación de una cuenta de usuario se otorgará un nombre de cuenta y una contraseña. El USUARIO es completamente responsable por mantener la confidencialidad de su cuenta o contraseña. Si el USUARIO tiene conocimiento de que la privacidad de su contraseña ha quedado expuesta, debe notificar inmediatamente a GID-CONUSS e informar acerca de cualquier fallo de seguridad.
3. **Contenidos del usuario.** El USUARIO acepta usar los SERVICIOS sólo para fines legales y queda prohibida la publicación, distribución y alojamiento en los SERVICIOS de: (a) cualquier material ilegal, perjudicial, amenazante, abusivo, acosador, difamatorio u obsceno de cualquier tipo, incluyendo, entre otros, cualquier material que fomente conductas que constituirían una ofensa penal, que produjera responsabilidad civil o que de otro modo infrinja cualquier ley aplicable local, estatal, nacional o internacional. (b) del cual el USUARIO no tenga el derecho de poner a disposición conforme a ninguna ley o relación contractual; o (c) infrinja cualquier derecho de patente, mar **NOMBRE DE PERSONA A CARGO DE LA ADMINISTRACION** ca comercial, secreto de mercado, derecho de autor u otros derechos de propiedad de cualquier parte. El USUARIO acepta no transmitir, subir, publicar, enviar por email o poner a disposición

Fuente: propia

Si iniciamos sesión como administrador, tendremos un panel habilitado llamado `.^dministrador.en` el cual podremos habilitar o deshabilitar nodos de computo en la sección de Compute, así como administrar los volúmenes, redes, routers, ips flotantes, y las instancias creadas por todos los usuarios en el sistema.

Figura 17: Vista general de Administrador

Nombre del proyecto	VCPUs	Disco	RAM	Horas VCPU	Horas disco GB	Horas Memoria MB
admin	2	42GB	2MB	160,18	3184,93	151811,80

Fuente: propia

En el panel de proyecto es donde actúan y trabajan los usuarios, en él pueden configurar las redes como servicio que tienen a su disposición, como lanzar maquinas virtuales, stacks de orquestación, y almacenamiento de objetos en swift, siempre y cuando no sobrepase los límites establecidos por el administrador.

Figura 18: Vista general de proyecto

Servicio	Endpoint de servicio
CloudFormation	http://controller:8000/v1
Compute	http://controller:8774/v2.1

Fuente: propia

En la sección de Compute encontramos las maneras de configurar una máquina virtual, teniendo en cuenta sus volúmenes en caso de requerir almacenamiento persistente, su conexión de red, el par de claves para ser accedido por SSH, la conexión a consola de interface web, el grupo de seguridad que administrará las reglas de reenvío de tramas como ICMP, SSH, HTTP etc., por la máquina, y algunos scripts de configuración

Cloud-init.

En la sección de redes podemos crear Routers y redes privadas internas para establecer el routeo entre instancias de una manera lógica, contando también con un sistema que permite graficar en vivo la topología actual de la red.

En la sección de orquestación podemos incluir Stacks para automatizar el despliegue de máquinas virtuales.

4.4. FASE 4

Esta fase es un complemento para la infraestructura como servicio que se pretende tener en el grupo para que la comunidad universitaria pueda usarla y es la posibilidad de ofrecer contenedores como servicio. Inicialmente el proyecto estaba pensado para orquestar estos contenedores con Openstack con la ayuda de 2 de sus módulos, Mangnum es un modulo que permite administrar contenedores de docker que han sido orquestados con Kubernetes, Docker Swarm o Mesos, ya que estos 3 son compatibles con magnum, y Zun el mas reciente modulo de Openstack que salio con la versión de Pike y que ahora tiene soporte con las versiones de Queens y Rocky (latest). Zun es el servicio de contenedores de Openstack el cual crea sus propio contenedores sin una virtualización previa. Sin embargo durante el desarrollo del proyecto hubo varios problemas con las versiones de Openstack ya que es un software que esta en constante cambio. por esta razón se decidió manejar los contenedores directamente con kubeadm el orquestador de clusters de Kubernetes Para comprender bien el tema se empezara hablando sobre los contenedores y Docker, luego se explicará el proceso para crear un cluster de contenedores.

4.4.1. contenedores Los contenedores de aplicaciones son entornos ligeros de tiempo de ejecución que proporcionan a las aplicaciones los archivos, las variables y las bibliotecas que necesitan para ejecutarse, maximizando de esta forma su portabilidad. Si bien las máquinas virtuales (VM) tradicionales permiten la virtualización de

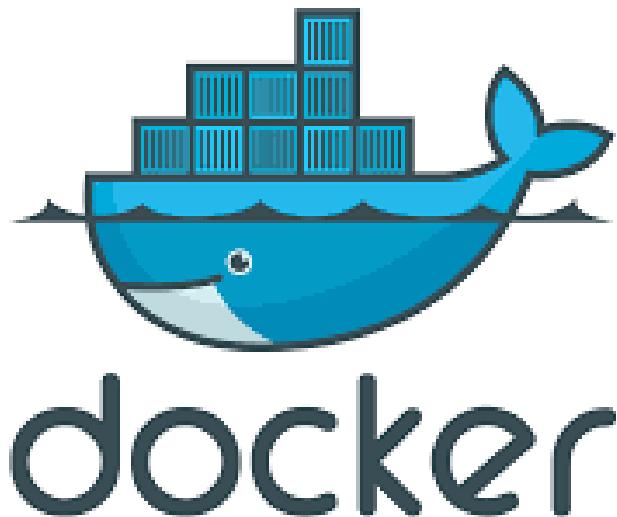
la infraestructura de computación, los contenedores habilitan la de las aplicaciones de software. A diferencia de las máquinas virtuales, los contenedores utilizan el sistema operativo (SO) de su host en lugar de proporcionar el suyo propio.¹. En pocas palabras un contenedor es una instancia de una máquina virtual que solo puede ejecutar un servicio.

4.4.2. docker Se encarga de gestionar contenedores, los contenedores Docker ejecutan un único servicio por lo cual hace que sea demasiado ligero, el peso de este sistema no tiene comparación con cualquier otro sistema de virtualización más convencional que estemos acostumbrados a usar. Por poner un ejemplo, una de las herramientas de virtualización más extendida es VirtualBox, y cualquier imagen de Ubuntu que queramos usar en otro equipo pesará entorno a 1Gb si contamos únicamente con la instalación limpia del sistema. En cambio, un Ubuntu con Apache y una aplicación web, pesa alrededor de 180Mb, lo que nos demuestra un significativo ahorro a la hora de almacenar diversos contenedores que podamos desplegar con posterioridad.².

¹**hpe-contenedore**.

²**openwebinars**.

Figura 19: Docker

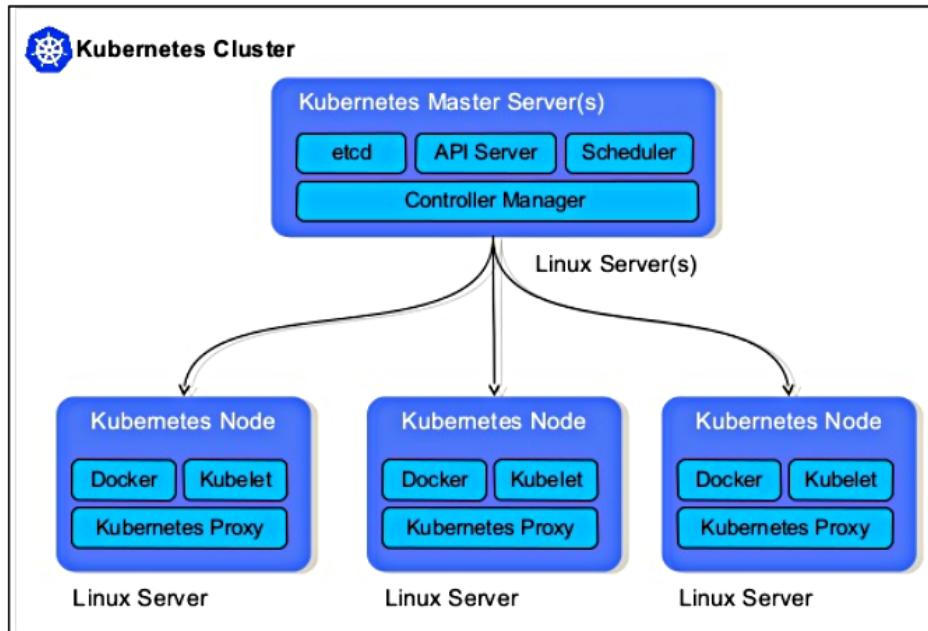


Fuente: [docker](#)

4.4.3. kubernetes Cuando se habla de kubernetes se habla de contenedores, kubernetes tiene 3 actores principales: kubeadm, kubectl, kubelet.

Kubeadm es el orquestador de clusters de contenedores docker. Kubelet es el agente que habla con el nodo master del cluster y Kubectl es el servicio para ejecutar comando en una consola de comandos. En un clusters normal de kubernetes existen 2 partes importantes: nodo master y nodo worker. Todas las solicitudes para un clutser como para su creacion se hacen el nodo master y es él quien le comunica y asigna tareas a sus trabajadores.

Figura 20: Kubernetes Cluster



Fuente: [kube-cluster](#)

Como se puede observar en la figura anterior el cluster esta conformado por 4 maquinas puedenser virtuales locales o que esten alojadas en la nube, fisicas o una combinacion de ellas, de estos 4 nodos uno toma el papel de master. Para este proyecto se usaran maquinas virtuales creadas por Openstack

Primero se debe escoger el numero de nodos a usar para el cluster y los recursos necesarios para cada uno. Teniendo eso presente y el sistema operativo con el cual se quieren los contenedores ya que los contenedores usan el mismo sistema operativo de su maquina donde está almacenado. Una vez se tengan las maquinas listas con los archivos de hosts configurados se procede a instalar kubeadm, kubelet, kubectl y docker.io.

Es importante aclarar que los nodos no deben llevar swap de lo contrario hay q usar el comando swapoff -a para deshabilitarla. Cuando todos los nodos tengan todos los paquetes instalados es hora es escoger quien de ellos va a ser el nodo master, dicho

deberá ejecutar este comando

```
# kubeadm init
```

con este comando creara un cluster y sera hará nodo master. Al finalizar la ejecución arrojará un token el cual se debe guardar ya que se usará para unir nodos workers al nodo master y escalar horizontalmente el cluster.

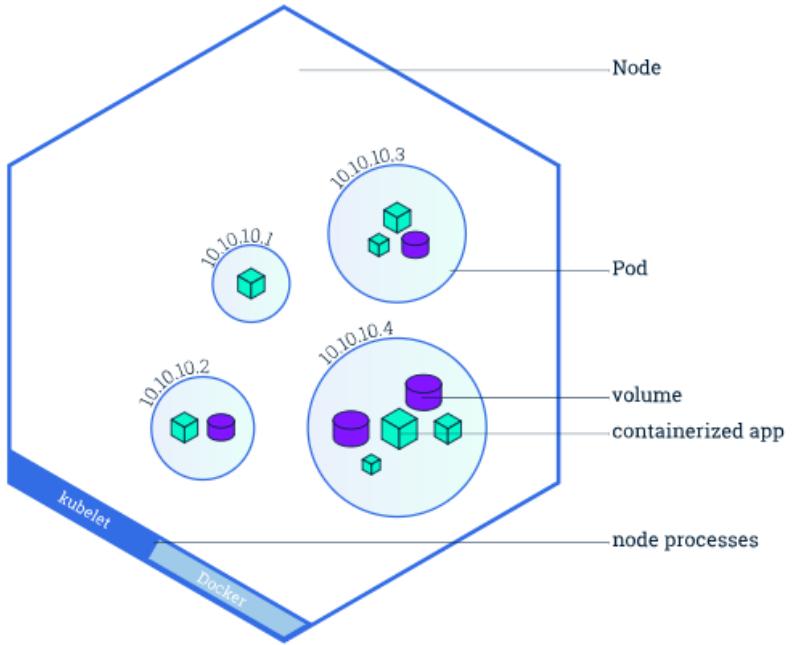
Ejemplo de token: kubeadm join 192.168.10.10:6443 –token yaiayi.kkgdmmybpap2ackk –discovery-token-ca-cert-hash sha256:055cbc786ebd5d6e9a749e0c239165e410e4d454d6a

Antes de empezar a unir nodos al cluster es necesario instalar un red para los pods que se vayan a ejecutar mas adelante, los drivers de red mas populares son franelas y weave-master. Una vez creada la red interna ahora los nodos podrán unirse al cluster.

Cuando se crea el cluster por primera vez se crean contenedores que contienen unos servicios encendidos que hacen que el cluster funcione de buena forma

Los contenedores de Kubernetes estan organizados dentro de pods, un pod es la unidad mas pequeña con la que kubernetes interactua y esta conformada por uno o varios contenedores, servicios y almacenamiento. Un nodo puede tener uno o muchos pods. La siguiente imagen describe de una mejor manera como se compone un nodo del cluster.

Figura 21: Ilustracion de un nodo de kubernetes



Fuente: pod

Por esta razon es necesario una red interna para los pods, para que pods de otros nodos puedan comunicarse. Como se puede observar en la figura cada pod tiene una direccion ip interna dentro del cluster la cual es usada para comunicaciones internas. Luego de tener instalada la red de pods se procede a unir los nodos workers al cluster, usando el token que nos arrojó el nodo master, para esto se entra a cada nodo y se ejecuta el token como un comando de Linux.

Inicialmente para este proyecto se uso un total de 4 maquinas para la creación del cluster cada una con los mismos recursos: 10GB de disco, 2GB de ram y 2 cores de cpu. Este cluster puede escalar tanto verticalmente como horizontalmente, el escalamiento vertical hace referencia a aumentar los recursos de los nodos ya existente, es decir aumentarle la capacidad de almacenamiento y procesamiento. Para entornos de produccion es mas recomendable usar el escalamiento horizontal que se refiere a añadir otro nodo al cluster

con características similares.

Una vez terminada la creación del cluster se procedió a inicializar pods para la creación de los contenedores. Como se ha venido diciendo un contenedor solo puede ejecutar un servicio por ejemplo cada contenedor solo va a tener un servidor web como apache o nginx. Por esta razón si una aplicación requiere más de un servicio se necesitan crear otros contenedores que corran los demás servicios, por lo general Kubernetes crea otros pods para estos otros servicios y los conecta por medio de la red de pods para el intercambio de datos.

Para crear los contenedores en Kubernetes se debe primero tener una imagen docker que contenga la aplicación que se quiera correr, para esto existen imágenes ya prediseñadas que se pueden listar utilizando el comando \$ docker images, en el nodo master; cabe resaltar que todas las instrucciones se deben ejecutar en el nodo master del cluster.

Figura 22: Imágenes docker almacenadas localmente

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
k8s.gcr.io/kube-scheduler-amd64	v1.10.2	0dcb3dea0db1	2 weeks ago	50.4 MB
k8s.gcr.io/kube-controller-manager-amd64	v1.10.2	f3fc0d775c4e	2 weeks ago	148 MB
k8s.gcr.io/kube-apiserver-amd64	v1.10.2	e774f647e259	2 weeks ago	225 MB
k8s.gcr.io/kube-proxy-amd64	v1.10.2	77019aa0531a	2 weeks ago	97.1 MB
weaveworks/weave-npc	2.3.0	21545eb3d6f9	5 weeks ago	47.2 MB
weaveworks/weave-kube	2.3.0	f15514acce73	5 weeks ago	96.8 MB
k8s.gcr.io/etcd-amd64	3.1.12	52920ad46f5b	2 months ago	193 MB
k8s.gcr.io/k8s-dns-dnsmasq-nanny-amd64	1.14.8	c2ce1fb51ed	4 months ago	41 MB
k8s.gcr.io/k8s-dns-sidecar-amd64	1.14.8	6f7f2dc7fab5	4 months ago	42.2 MB
k8s.gcr.io/k8s-dns-kube-dns-amd64	1.14.8	80cc5ea4b547	4 months ago	50.5 MB
k8s.gcr.io/pause-amd64	3.1	da86e6ba6cal	4 months ago	742 kB

Fuente: propia

Si la imagen no se encuentra en el repositorio local entonces se puede recurrir al repositorio de imágenes que tiene docker en la nube www.hub.docker.com donde la comunidad y las empresas desarrolladoras de software suben sus imágenes para ser compartidas, sin embargo para contenedores específicos es necesario crear la imagen a partir de un Dockerfile, un archivo de texto que contiene las instrucciones que se harían en una consola de comandos para construir una imagen con la ayuda del comando \$ docker build.

Figura 23: Dockerfile

```
FROM python:3.4

RUN mkdir /code
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
ADD . /code/

# ssh
ENV SSH_PASSWD "root:Docker!"
RUN apt-get update \
    && apt-get install -y --no-install-recommends dialog \
    && apt-get update \
    && apt-get install -y --no-install-recommends openssh-server \
    && echo "$SSH_PASSWD" | chpasswd

COPY sshd_config /etc/ssh/
COPY init.sh /usr/local/bin/

RUN chmod u+x /usr/local/bin/init.sh
EXPOSE 8000 2222
#CMD ["python", "/code/manage.py", "runserver", "0.0.0.0:8000"]
ENTRYPOINT ["init.sh"]
```

Fuente: [dockerfile](#)

La figura anterior muestra un Dockerfile que parte de una imagen con python3.4 a la cual se le crea un directorio code y se empieza a trabajar desde ahí. La instrucción ADD . /code/ es el comando que permite que todo lo que hay en la carpeta donde está el dockerfile sea añadido a la nueva imagen en la ruta /code

Para la construcción de la imagen debe usarse el comando \$ docker build en la la carpeta donde esté el dockerfile y se le pasará como argumento el nombre de la imagen y la versión, mas adelante se mostrará un ejemplo.

Como se dijo anteriormente kubectl es la herramienta que permite ejecutar comandos para el cluster de kubernetes, si se quiere conocer los pods que hay cuando se creó el cluster basta con usar el comando **kubectl get pods –all-namespaces** y nos mostrará

una lista de todos los pods que hay en el cluster.

Figura 24: Tabla de todos los pods del cluster

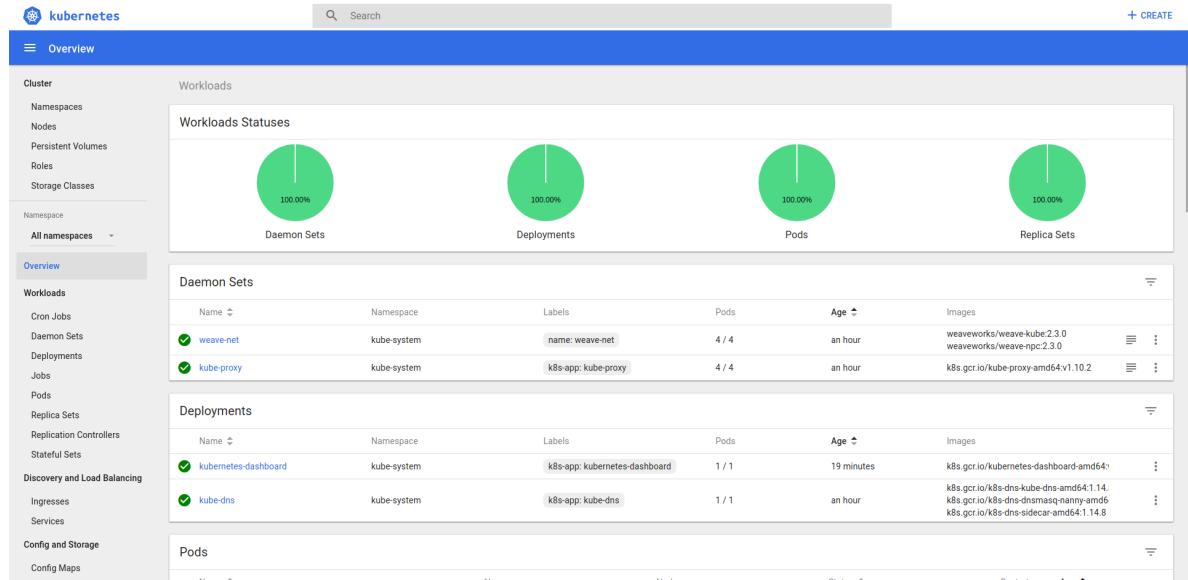
NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
kube-system	etcd-kube-master	1/1	Running	0	29m	10.6.101.71	kube-master
kube-system	kube-apiserver-kube-master	1/1	Running	0	29m	10.6.101.71	kube-master
kube-system	kube-controller-manager-kube-master	1/1	Running	0	29m	10.6.101.71	kube-master
kube-system	kube-dns-86f4d74b45-vnqqm	3/3	Running	0	30m	10.32.0.2	kube-master
kube-system	kube-proxy-dd9fp4	1/1	Running	0	25m	10.6.101.70	kube-nodo3
kube-system	kube-proxy-ffwcf	1/1	Running	0	25m	10.6.101.73	kube-nodo2
kube-system	kube-proxy-k2lfw	1/1	Running	0	30m	10.6.101.71	kube-master
kube-system	kube-proxy-p6mp	1/1	Running	0	25m	10.6.101.72	kube-node1
kube-system	kube-scheduler-kube-master	1/1	Running	0	29m	10.6.101.71	kube-master
kube-system	weave-net-78j76	2/2	Running	0	25m	10.6.101.70	kube-nodo3
kube-system	weave-net-8n8fv	2/2	Running	1	25m	10.6.101.72	kube-node1
kube-system	weave-net-ds79x	2/2	Running	0	25m	10.6.101.73	kube-nodo2
kube-system	weave-net-kpp4n	2/2	Running	0	26m	10.6.101.71	kube-master

Fuente: propia

Esto nos mostrara informacion acerca del estado de cada pod, el numero de replicas, tiempo de creacion, ip interna y el nodo donde se encuentra el pod. Pero toda esta informacion puede ser vista de una forma mas agradable para el usuario administrador haciendo uso del dashboard que provee kubernetes. El dashboard de kubernetes es una aplicación que se ejecuta en contenedores del mismo cluster.

Hay 2 formas de ejecutar un contenedor con kubectl, la primera es usando imagenes de docker que esten en el repositorio local para eso utilizamos el comando **kubectl run nombre_pod imagen/imagen**. La segunda es usando un archivo yaml que este en el el nodo master o que esté en internet para esto se usa el comando **kubectl apply -f ruta_archivo.yaml**

Figura 25: Pantalla inicial del dashboard de kubernetes



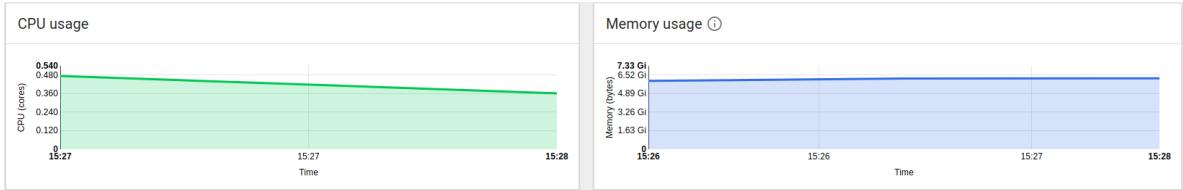
Fuente: propia

Para acceder a este dashboard se necesita la ip del master y el puerto por el cual el servicio esta corriendo.

Existen varios complementos para mejorar la administración y monitoreo de cluster, para este proyecto se usaron heapster, wave scope y grafana.

Heapster agrega una capa más al dashboard de kubernetes que mejora la síntesis de la información mediante gráficas de procesamiento y uso de ram en los pods, nodos, y cluster. Al igual que el dashboard estos otros complementos también son creados en contenedores en el mismo cluster.

Figura 26: Graficas de heapster en el dashboard de kubernetes



Fuente: propia

La imagen anterior muestra el estado actual de la CPU y la memoria total del cluster reuniendo los recursos de los 4 nodos, como se puede ver solamente desplegando estos pocos servicios para la orquestacion de los contenedores y visualización de estadísticas del cluster se requiere una buena cantidad de recursos

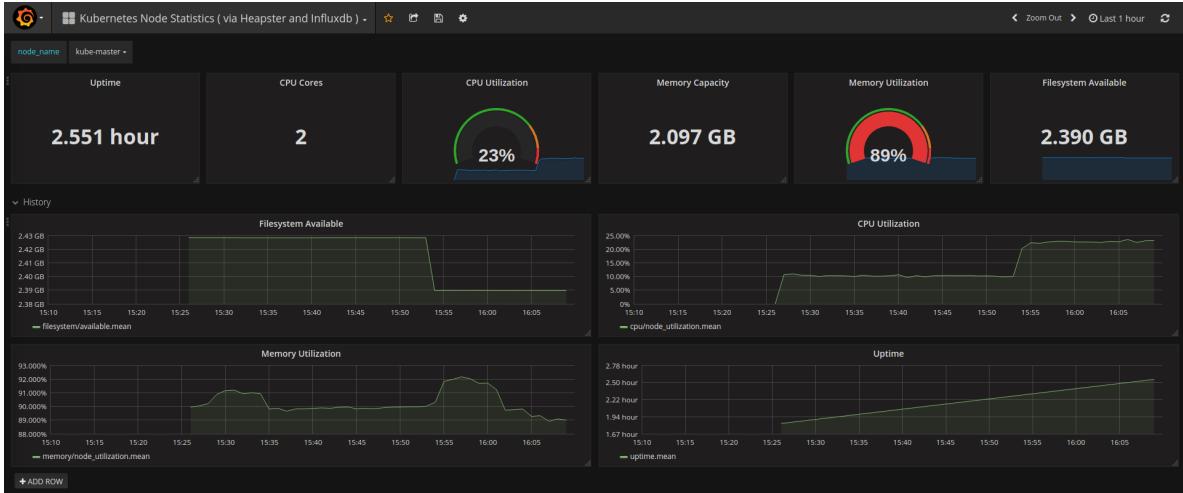
Figura 27: Total de pods después de desplegar servicios de monitoreo

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
kube-system	etcd-kube-master	1/1	Running	0	2h	10.6.101.71	kube-master
kube-system	heapster-69b5d4974d-28z17	1/1	Running	0	33m	10.44.0.1	kube-nodo2
kube-system	kube-apiserver-kube-master	1/1	Running	0	2h	10.6.101.71	kube-master
kube-system	kube-controller-manager-kube-master	1/1	Running	0	2h	10.6.101.71	kube-master
kube-system	kube-dns-86f4d74b45-vnqqm	3/3	Running	0	2h	10.32.0.2	kube-master
kube-system	kube-proxy-dd9p4	1/1	Running	0	2h	10.6.101.70	kube-nodo3
kube-system	kube-proxy-ffwcf	1/1	Running	0	2h	10.6.101.73	kube-nodo2
kube-system	kube-proxy-k2lfw	1/1	Running	0	2h	10.6.101.71	kube-master
kube-system	kube-proxy-p6mp	1/1	Running	0	2h	10.6.101.72	kube-node1
kube-system	kube-scheduler-kube-master	1/1	Running	0	2h	10.6.101.71	kube-master
kube-system	kubernetes-dashboard-5c469b58b8-m86k8	1/1	Running	0	1h	10.36.0.1	kube-nodo3
kube-system	monitoring-grafana-69df66f668-gm5nk	1/1	Running	0	31m	10.36.0.2	kube-nodo3
kube-system	monitoring-influxdb-78d4c6f5b6-wplnn	1/1	Running	0	32m	10.42.0.1	kube-node1
kube-system	weave-net-78j76	2/2	Running	0	2h	10.6.101.70	kube-nodo3
kube-system	weave-net-8n8fv	2/2	Running	1	2h	10.6.101.72	kube-node1
kube-system	weave-net-ds79x	2/2	Running	0	2h	10.6.101.73	kube-nodo2
kube-system	weave-net-kpp4n	2/2	Running	0	2h	10.6.101.71	kube-master
weave	weave-scope-agent-2k6ph	1/1	Running	0	3m	10.6.101.72	kube-node1
weave	weave-scope-agent-5q7z6	1/1	Running	0	3m	10.6.101.71	kube-master
weave	weave-scope-agent-lttm7	1/1	Running	0	3m	10.6.101.70	kube-nodo3
weave	weave-scope-agent-n448f	1/1	Running	0	3m	10.6.101.73	kube-nodo2
weave	weave-scope-app-db44d984-xzswn	1/1	Running	0	3m	10.44.0.2	kube-nodo2

Fuente: propia

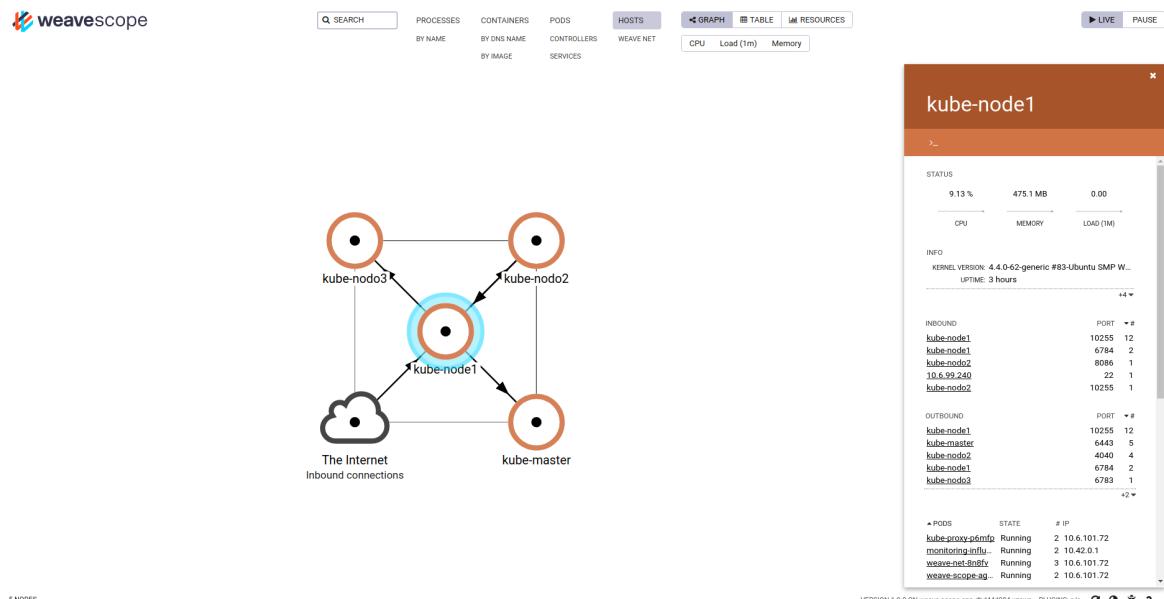
La imagen anterior muestra todos los pods que se necesitaron para tener un cluster con herramientas de monitoreo

Figura 28: Dashboard de Grafana



Fuente: propia

Figura 29: Dashboard de Wave Scope



Fuente: propia

La Figura 26 nos muestra una mejor vista del comportamiento del cluster, este dashboard tiene muchas opciones de configuración que le permiten al administrador tener una mejor experiencia y una personalización para las graficas.

La figura 27 es una gran herramienta que nos permite visualizar la topología del cluster tanto de la red como de los distintos procesos que se están ejecutando, ademas también puede mostrar de forma detalla información de los recursos, sin embargo el principal atractivo de está herramienta es la ejecución de una shell de Linux que permite ejecutar comando directamente en los nodos, pods o contenedores del cluster.

4.5. FASE 5

Tener sistemas de monitoreo para una infraestructura como la del grupo Conuss es de vital importancia ya que permite conocer el rendimiento de las maquinas así como de los servicios y la red. Los sistemas de monitoreo en tiempo real facilitan aun más la administración de la infraestructura la toma de decisiones en situaciones de incertidumbre y mejora la experiencia con el usuario. Sin embargo para que un monitoreo sea en tiempo real se necesitan de notificaciones que comuniquen a los administradores ante alguna eventualidad que haya ocurrido en el sistemas, ya que no siempre se puede estar mirando las pantallas de los computadores donde están los dashboards, por esta razón se implementó un sistema de notificaciones vía Telegram (aplicación de mensajería instantánea) que notifica en tiempo real cuando algún servicio haya cambiado su estado o esté en un estado crítico.

Para esta tarea se usó la aplicación Nagios, una herramienta de monitoreo para la infraestructura física del la nube CloudEisi que permite el chequeo de los servidores y vigila su comportamiento. Inicialmente se pretendia hacer esto con Grafana de la misma forma que se hizo con el cluster de kubernetes pero la falta de experiencia y de documentación no permitió poder configurar las notificaciones para este sistema. Sin embargo Nagios siempre fue una buena opción cuando se hizo la búsqueda de monitores de sistemas.

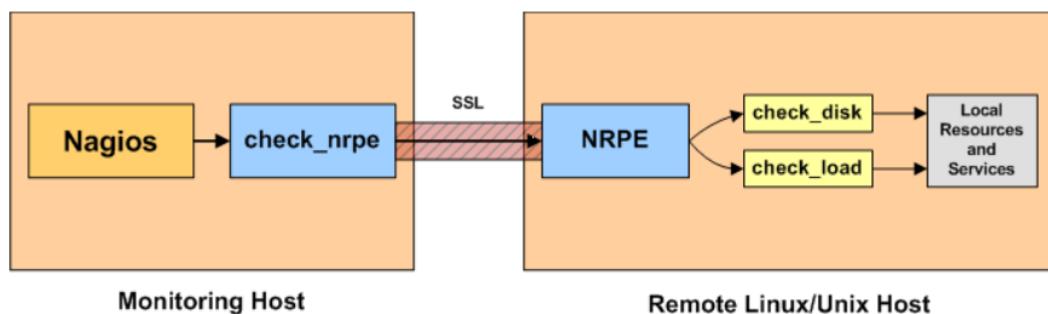
Figura 30: Logo de Nagios



Fuente: [nagios](#)

El servicio de nagios esta compuesto de 2 partes importantes para su uso en producción: Nagios-server y nagios-nrpe. Nagios-server es el actor principal quien recibe información de los servidores, la analiza y la expone con gráficos y estadísticas, nagios también cuenta con un dashboard que muestra información del sistema en graficos y porcentajes, el segundo actor es el agente nrpe que se instala en todos los servidores físicos y se comunica con el servidor nagios para llevar información de cada servicio y cada recurso que se haya configurado para que sea monitoreado. De forma mas sencilla se puede ver como un nodo master y nodos workers.

Figura 31: Modelo lógico nagios



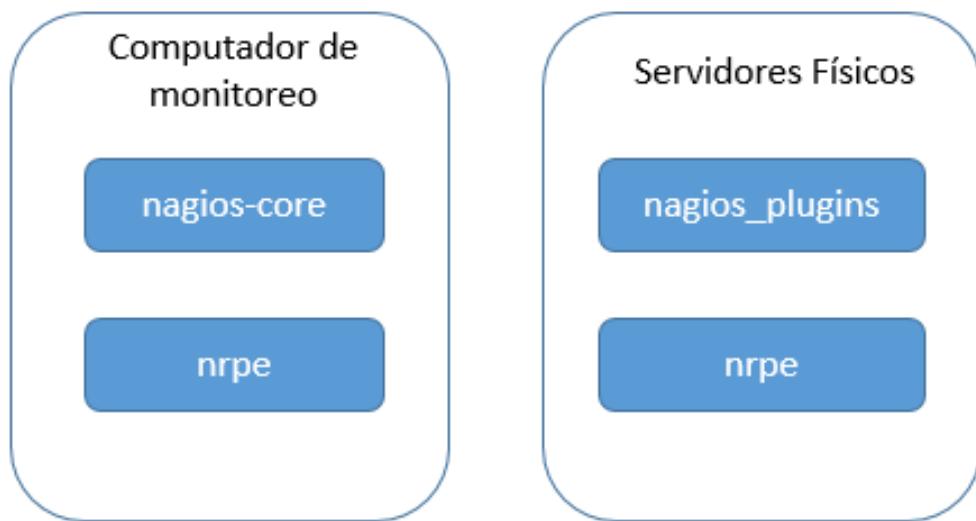
Fuente: [nagios-nrpe](#)

El modelo anterior muestra como el servidor nagios se comunica con los agentes nrpe de

los servidores quienes a su vez ejecutan unos scripts llamados check_disk y check_load que verifican el estados de estos 2 servicios específicos en este caso del uso del disco y de la carga de trabajo, sin embargo estos scrips están alojados en el servidor nagios.

Para la implementación de esta herramientas se utilizaron 3 paquetes principales los cuales están distribuidos de la siguiente manera

Figura 32: Distribución de software nagios



Fuente: propia

Al finalizar la instalación se procede a configurar los archivos de forma adecuada para que haya comunicación entre el servidor nagios y los agentes nrpe, esta configuración se realiza en todos los servidores físicos y en el computador de monitoreo.

Figura 33: Archivo de configuracion del servidor Labroides en el servidor nagios

```
define host {
    use                               linux-server,host-pnp
    host_name                         labroides
    alias                             labroides
    address                           10.6.100.2
    max_check_attempts                 5
    check_period                       24x7
    notification_interval              30
    notification_period                24x7
    process_perf_data                  0
}

define service {
    use                               generic-service,srv-pnp
    host_name                         labroides
    service_description                 CPU load
    normal_check_interval               1
    max_check_attempts                  3
    check_command                      check_nrpe!check_load!5.0,4.0,3.0!10.0,6.0,4.0
}

define service {
    use                               generic-service,srv-pnp
    host_name                         labroides
    service_description                 / free space
    normal_check_interval               1
    max_check_attempts                  3
    check_command                      check_nrpe!check_/
}
```

Fuente: propia

Este archivo de configuracion se encuentra localizado en la ruta `/usr/local/nagios/etc/objects/`

Figura 34: Archivo de configuracion del agente nrpe en Labroides

```
... to match the arguments format the plugins expect. Remember, these are
# examples only!

# The following examples use hardcoded command arguments...
# This is by far the most secure method of using NRPE

command[check_users]=/usr/local/nagios/libexec/check_users -w 5 -c 10
command[check_load]=/usr/local/nagios/libexec/check_load -r -w .50,.45,.40 -c .70,.65,.60
command[check_/_]=/usr/local/nagios/libexec/check_disk -w 20% -c 10% -p /dev/sda2
command[check_/_boot/efi]=/usr/local/nagios/libexec/check_disk -w 20% -c 10% -p /dev/sda4
command[check_zombie_procs]=/usr/local/nagios/libexec/check_procs -w 5 -c 10 -s Z
command[check_total_procs]=/usr/local/nagios/libexec/check_procs -w 600 -c 800
command[check_swap]=/usr/local/nagios/libexec/check_swap -w 20% -c 10%
command[check_ssh]=/usr/local/nagios/libexec/check_ssh -H 10.6.100.2
command[check_http]=/usr/local/nagios/libexec/check_http -u 5 -c 10 --ssl -H 10.6.100.2
command[check_ping]=/usr/local/nagios/libexec/check_ping -H 8.8.8.8 -w 50,50% -c 100,70%
command[check_memory]=/usr/local/nagios/libexec/check_memory2 -f -w 20 -c 10

# The following examples allow user-supplied arguments and can
# only be used if the NRPE daemon was compiled with support for
# command arguments *AND* the dont_blame_nrpe directive in this
# config file is set to '1'. This poses a potential security risk, so
# make sure you read the SECURITY file before doing this.
```

Fuente: propia

La imagen anterior es una parte del archivo de configuración del agente nrpe que se encuentra en todos los equipos en la ruta **/usr/local/nagios/etc/nrpe.conf** . En este archivo se especifican los servicios y recursos que se desean monitorear en el servidor ademas de especificar parametros para indicar el estado del servicio o recurso, los posibles estados pueden ser: "critical", "warning", "ok". Estos estados se verán de mejor manera en el dashboard.

Figura 35: Ubicación de los scripts en el servidor nagios

```
conuss@conuss-PC:~$ vim /usr/local/nagios/libexec/
attachment.php?link_id=1528  check_ftp          check_memory5      check_ping        check_tcp
check_apt                  check_hpjd         check_mem.py       check_ping2       check_time
check_bandwidth            check_http         check_mrtg         check_ping.pl    check_traffic
check_breeze                check_icmp          check_nagios       check_pop         check_udp
check_bw.sh                check_ide_smart     check_ifoperstatus check_procs      check_ups
check_by_ssh                check_ifstatus      check_ifstatus.pl check_real        check_uptime
check_clamd                check_imap          check_ntp          check_rpc         check_users
check_cluster              check_ircd          check_ntps         check_sensors    check_wave
check_dhcp                 check_jabber        check_nrpe         check_simap      negate
check_dig                  check_load          check_nt           check_smtp        urlize
check_disk                 check_log           check_ntp          check_snmp        utils.pm
check_disk_smb             check_mailq         check_ntp_peer     check_snmp_netint.pl utils.sh
check_dns                  check_memory2      check_ntp_time     check_spop        wget-log
check_dummy                check_memory3      check_ntpstat      check_ssh
check_file_age              check_memory4      check_oracle       check_sslsmtp
check_flexlm               check_overcr       check_overcr      check_swap

conuss@conuss-PC:~$ vim /usr/local/nagios/libexec/
```

Fuente: propia

Estos scripts contienen secuencias de comandos que son utilizados por los agentes nrpe en los servidores para verificar el estados de los servicios y proporcionar un balance de ellos mismo.

Figura 36: Dashboard de Nagios

Service Status Details For All Hosts						
Host	Service	Status	Last Check	Duration	Attempt	Status Information
labroides	/ free space	OK	05-15-2018 22:11:25	0d 9h 49m 43s	1/3	DISK OK - free space: /var/tmp 388559 MB (89.20% inode=99%):
	/boot/efi free space	OK	05-15-2018 22:11:34	0d 9h 49m 34s	1/3	DISK OK - free space: /boot/efi 236 MB (98.61% inode=-):
	CPU load	OK	05-15-2018 22:11:42	0d 9h 49m 26s	1/3	OK - load average per CPU: 0.15, 0.12, 0.11
	Current Users	OK	05-15-2018 22:11:51	0d 9h 49m 17s	1/3	USERS OK - 2 users currently logged in
	HTTP	CRITICAL	05-15-2018 22:11:18	1d 13h 58m 50s	3/3	connect to address 10.6.100.2 and port 443: Connection refused
	Memory	UNKNOWN	05-15-2018 22:12:00	0d 9h 50m 8s	3/3	NRPE: Unable to read output
	PING	WARNING	05-15-2018 22:11:10	0d 2h 10m 58s	3/3	PING WARNING - Packet loss = 0%, RTA = 59.72 ms
	SSH	OK	05-15-2018 22:11:19	0d 9h 49m 49s	1/3	SSH OK - OpenSSH_7.2p2 Ubuntu-4ubuntu2.4 (protocol 2.0)
	Swap	OK	05-15-2018 22:11:54	0d 9h 49m 14s	1/3	SWAP OK - 100% free (9477 MB out of 9535 MB)
	Total Processes	OK	05-15-2018 22:11:58	0d 9h 49m 10s	1/3	PROCS OK: 445 processes
lactoria	Zombie Processes	OK	05-15-2018 22:11:38	0d 9h 49m 30s	1/3	PROCS OK: 0 processes with STATE = Z
	/ free space	OK	05-15-2018 22:11:19	1d 3h 47m 23s	1/3	DISK OK - free space: /var/tmp 807351 MB (93.10% inode=100%):
	/boot/efi free space	OK	05-15-2018 22:11:29	1d 3h 47m 17s	1/3	DISK OK - free space: /boot/efi 507 MB (99.34% inode=-):
	CPU load	OK	05-15-2018 22:11:34	0d 3h 51m 34s	1/3	OK - load average per CPU: 0.29, 0.30, 0.32
	Current Users	OK	05-15-2018 22:11:44	1d 3h 46m 58s	1/3	USERS OK - 0 users currently logged in
	HTTP	CRITICAL	05-15-2018 22:11:18	1d 13h 58m 49s	3/3	connect to address 10.6.100.3 and port 443: Connection refused
	Memory	UNKNOWN	05-15-2018 22:11:54	1d 3h 46m 40s	3/3	NRPE: Unable to read output
	PING	UNKNOWN	05-15-2018 22:12:03	1d 3h 46m 34s	3/3	Usage:
	SSH	OK	05-15-2018 22:11:10	1d 3h 47m 3s	1/3	SSH OK - OpenSSH_7.2p2 Ubuntu-4ubuntu2.4 (protocol 2.0)
	Swap	UNKNOWN	05-15-2018 22:11:47	1d 3h 46m 34s	3/3	Usage:
	Total Processes	CRITICAL	05-15-2018 22:11:12	1d 13h 58m 33s	3/3	PROCS CRITICAL: 335 processes
	Zombie Processes	OK	05-15-2018 22:11:58	1d 3h 47m 30s	1/3	PROCS OK: 0 processes with STATE = Z

Fuente: propia

El dashboard de nagios nos muestra el estados de los servicios en colores de forma que el color verde representa un estado bueno (ok), el amarillo es peligro (warning), y el rojo es crítico (critical). El color naranja hace alusión a un servicio que no esta siendo procesado por lo tanto no se puede saber su estado.

Por último se implemento el sistema de notificaciones con Telegram, una implementación que permite Nagios y que es aprovechada en muchos sistemas de producción para tener alertas de fallos o cambios en el sistema.

Para este objetivo fue necesario utilizar un script hecho por la comunidad que se comunica con un bot creado en la aplicación Telegram al cual se le ha dado el nombre de Camilo y que se encuentra en un chat de grupo administrativo a donde llegan las notificaciones que el bot envía.

Figura 37: Script de Telegram

```
#!/usr/bin/env python2.7

import argparse
from twx.botapi import TelegramBot


def parse_args():
    parser = argparse.ArgumentParser(description='Notificaciones de Nagios con Telegram')
    parser.add_argument('-t', '--token', nargs='?', required=True)
    parser.add_argument('-o', '--object_type', nargs='?', required=True)
    parser.add_argument('--contact', nargs='?', required=True)
    parser.add_argument('--notificationtype', nargs='?')
    parser.add_argument('--hoststate', nargs='?')
    parser.add_argument('--hostname', nargs='?')
    parser.add_argument('--hostaddress', nargs='?')
    parser.add_argument('--servicestate', nargs='?')
    parser.add_argument('--servicedesc', nargs='?')
    parser.add_argument('--output', nargs='?')
    args = parser.parse_args()
    return args


def send_notification(token, user_id, message):
    bot = TelegramBot(token)
    bot.send_message(user_id, message).wait()

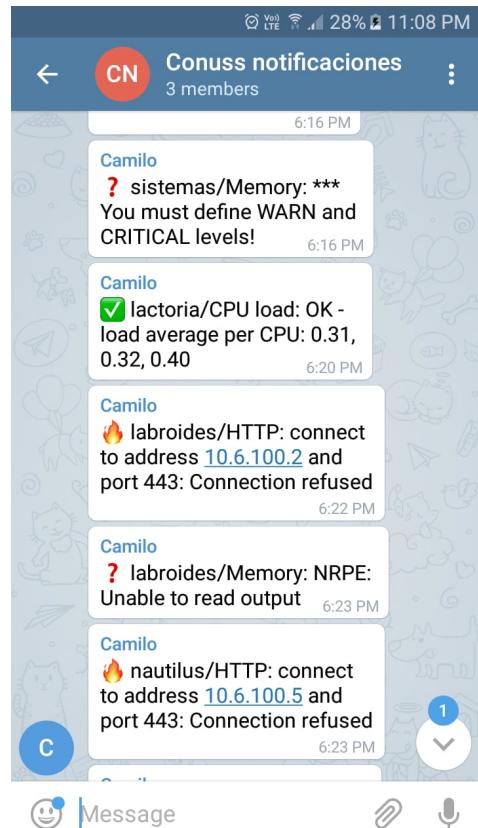

def host_notification(args):
    state = ''
    if args.hoststate == 'UP':
        state = u'\U00002705 '
    elif args.hoststate == 'DOWN':
        state = u'\U0001F525 '
    elif args.hoststate == 'UNREACHABLE':
        state = u'\U00002753 '


"/usr/local/bin/telegram_nagios.py" [Sólo lectura] 74L, 2081C
```

Fuente: propia

Sin embargo para que esto funcione se tuvo que modificar otros archivos de configuración del servidor nagios donde se indica el canal por el cual se desea enviar las notificaciones en este caso Telegram y el contacto al cual quiere ser enviado o sea Camilo.

Figura 38: Notificaciones en la aplicación Telegram



Fuente: propia

Una vez se termine de configurar el sistema empezaran a llegar al chat grupal mensajes de Camilo con alertas cuando un servicio cambia de estado.

5. CONCLUSIONES

Del trabajo realizado llegamos a concluir que:

1. La responsabilidad de una infraestructura cloud recae totalmente en el administrador de sistemas y su capacidad para brindar prontas soluciones a los errores presentados del día a día y a las peticiones de los usuarios finales en el préstamo de los servicios.
2. Ubuntu es una gran distribución y sencilla de utilizar para aquellos que desean empezar ha entrar en el mundo de Linux, no obstante, la implementación de OpenStack en una distribución Red Hat puede llegar a ser mucho más confiable debido a que principalmente el proyecto está desarrollado para esta distribución, mientras que en Ubuntu pueden existir inconsistencias en la paquetería.
3. Los contenedores son una abstracción de las maquinas virtuales, una tecnología maravillosa que permite tener aplicaciones en instancias muy ligeras que reducen de forma significativa el uso de los recursos de una maquina haciendo uso únicamente de los servicios necesarios. Ademas de esto los contenedores no necesitan de un sistemas operativo en especial ya que funcionan utilizando el sistema operativo de la maquina en la cual se está ejecutando por lo que esto le da una capacidad de adaptación.
4. Para un administrador de sistemas es esencial tener un registro de eventos que permita argumentar en respuesta a un usuario un fallo con el servicio y brindar estadísticas de confiabilidad para proponerse metas que generen calidad a futuro. Ademas de esto las notificaciones en tiempo real sobre una eventualidad del sistema permite una pronta reacción por parte de los administradores para mitigar dicha anormalidad.

6. RECOMENDACIONES Y TRABAJO FUTURO

- La tecnología de OpenStack está constantemente en desarrollo, por ende es posible que las configuraciones de alta disponibilidad no sean las más optimas en el momento de llevarse a producción si se utiliza una versión demasiado reciente como lo es Queens, sin embargo, el soporte que se le da a cada versión es lo suficientemente a largo plazo, como para ir aplicando pequeñas actualizaciones a lo largo de los proyectos de grado realizados en el grupo. Cabe destacar que la experiencia es lo que más cuenta, pues se debe trabajar desde lo más básico que es el entendimiento de la función de los procesos en un Sistema Operativo, atravesando la complejidad de las redes, y finalizando indefinidamente con el extenso mundo del Cloud Computing de hoy en día.
- Utilizar un cluster de storage es conveniente en la alta disponibilidad, para ello se requieren más servidores, en los cuales al instalar el sistema operativo otorgue un formato a los discos que permita el uso del protocolo iSCSI. La recomendación viene dada a utilizar el software Ceph RADOS, pues permite tener Shared File System, Object Storage y Block storage como servicio.
- Investigar acerca de los bugs comunes que impiden que la visualización del entorno de OpenStack a través de su Dashboard (Horizon) sea fluida en navegadores como Google Chrome, pues en navegadores Firefox va demasiado bien.
- Cambiar la distribución Ubuntu Server a una distribución como Red Hat o CentOS ya que la mayoría de los desarrollos de aplicaciones en la nube están destinadas para estas distribuciones que son mas enfatizadas para servidores y servicios.
- Investigar y aplicar herramientas de implementación y configuración como Ansible, Chef y Puppet que permitan el despliegue de aplicaciones y servicios de forma más sencilla.
- Continuar investigando sobre el uso de contenedores en el mundo del Cloud Computing ya que es una tecnología que está revolucionando la ejecución de aplicaciones.

7. LIMITACIONES Y PROBLEMAS

A pesar de las limitaciones listadas a continuación, el esfuerzo por hacer realidad éste proyecto quiere inspirar a la inversión en la nube CloudEISI para ofrecer un mejor servicio en el futuro.

-La no independencia de la Red interna de la UIS fue un problema muy agobiante, ya que tuvimos que acoplarnos a los constantes cambios de red que sucedieron con las modificación de la red en la Universidad, el cambio de las IP privadas clase C a clase A, afectó de una manera trágica los servicios prestados en la nube de aquel momento, pues sitios web como el aula virtual Meiweb, se vieron inutilizables por sus múltiples conexiones previamente establecidas para su alta disponibilidad, además de otras maquinas virtuales asignadas a estudiantes de proyecto que resultaron incomunicadas. También debemos tener en cuenta, que si por algún motivo falla la conexión del UIS-ISP, queda sin acceso la nube CloudEISI desde el exterior.

-Velocidad de transferencia interna. Los cables de red destinados a la conexión interna de la nube son cables Ethernet, es decir, su velocidad de transferencia tiende a rozar los 100 Mbps, aunque actualmente no sea un factor demasiado limitante por el pequeño tamaño de la nube, en un futuro será inevitable tener molestias debido a su naturaleza escalable.

-Cortes de luz. Actualmente la nube está lejos de ser considerado un datacenter, pues sus instalaciones físicas no están adaptadas para este fin, sin embargo se espera que haya un cambio en las infraestructura física del grupo que permita escalar de una mejor forma los servicios prestados.

-Las nuevas políticas de la red de la Universidad que ahora limita a 3 el número de direcciones MAC por puerto. A pesar de que el grupo actualmente cuenta con un numero pequeño de usuarios, esta limitación es muy grave para una infraestructura de nube que provee servicios de maquinas virtuales ya que se espera que con el tiempo, tanto el número de usuarios como el número de servicios aumente.

8. REFERENCIAS

A., Esaú. Docker, Qué es y sus principales características. [En línea] openwebinars.com, 2014 (Recuperado 14 mayo 2018) Disponibles en <https://openwebinars.net/blog/docker-que-es-sus-principales-caracteristicas/>.

AUTORÍA., Propia.

DOCKER. Docker. [En línea] Docker.org, (Recuperado 14 mayo 2018). Disponibles en <https://www.docker.com>.

ENTERPRISE, Hewlett Packard. ¿Qué son los contenedores? [En línea] hpe.com, (Recuperado 14 mayo 2018). Disponible en <https://www.hpe.com/es/es/what-is/containers.html>.

ESCRIBANO, Francisco. Descubriendo RabbitMQ: una solución para colas de mensajería. [En línea] Beeva.com, (Recuperado el 14 de mayo 2018). Disponible en <https://www.beeva.com/beevaview/tecnologia/descubriendo-rabbitmq-una-solucion-para-colas-de-mensajeria/>.

GALSTAD, Ethan. Modelo del funcionamiento de nagios. [En línea] assets.nagios.com, (Recuperado 15 mayo 2018). Disponibles en <https://assets.nagios.com/downloads/nagioscore/docs/nrpe/NRPE.pdf>.

INFLUXDBDATA. InfluxDB. [En línea] influxdata.com, (Recuperado 14 mayo 2018). Disponibles en <https://www.influxdata.com/time-series-platform/influxdb/>.

K., Rahul. Logo de Nagios. [En línea] tecadmin.net, (Recuperado 15 mayo 2018). Disponibles en <https://tecadmin.net/install-nagios-monitoring-server-on-ubuntu/>.

KUBERNETES. Esquema interno de un nodo de Kubernetes. [En línea] kubernetes.com.

MEMCACHED. Memcached. [En línea] memcached.org, (Recuperado el 14 de mayo del 2018) Disponible en <https://memcached.org/>.

MICROSOFT. Ejemplo de un Dockerfile. [En línea] docs.microsoft.com, (Recuperado 14 mayo 2018). <https://docs.microsoft.com/en-us/azure/app-service/containers/tutorialcontainer-docker-image>.

OPENSTACK. OpenStack: [En línea] Openstack.org, (Recuperado 7 abril 2018). Disponible en <https://www.openstack.org>.

OPENSTACK. Tabla de quisitos para Openstack. [En línea] openstack.org, (Recuperado 14 mayo 2018). Disponibles en <https://docs.openstack.org/ha-guide/environmenthardware.html>.

PHILIPS, Brandon. Etcd. [En línea] coreos.com, (Recuperado el 14 de mayo del 2018) Disponible en <https://coreos.com/etcd/>.

PROPIA., Autoría. Infraestructura Física GID-CONUSS.

SCHRODER, Carla. Kubernetes-cluster. [En línea] linux.com, (Recuperado 14 mayo 2018). Disponibles en <https://www.linux.com/news/learn/chapter/intro-to-kubernetes/2017/4/whatmakes-kubernetes-cluster>.

VELAZCO, Rubén. IaaS, PaaS, CaaS, SaaS – ¿Qué significan estos conceptos de Cloud Computing? [En línea] Redezone.net, 2016. (Recuperado 7 abril 2018). Disponibles en <https://www.redeszone.net/2016/07/17/iaas-paas-caas-saas-significan-estosconceptos-cloud-computing/>.

9. BIBLIOGRAFIA

- Alexellis (2018). openfaas/faas. [en línea] GitHub. Recuperado de: <https://github.com/openfaas/faas> [Último acceso 5 Mayo 2018].
- Anicas, M. (2015). How To Install Nagios 4 and Monitor Your Servers on Ubuntu 14.04 DigitalOcean. [en línea] Digitalocean.com. Recuperado de: <https://www.digitalocean.com/community/tutorials/how-to-install-nagios-4-and-monitor-your-servers-on-ubuntu-14-0-4> [Último acceso 13 Febrero. 2018].
- Brooks, J. (2017). Migrating Kubernetes on Fedora Atomic Host 27. [en línea] Projectatomic.io. Recuperado de: <https://www.projectatomic.io/blog/2017/11/migrating-kubernetes-on-fedora-atomic-host-27/> [Último acceso 1 Marzo. 2018].
- Butow, T. (2018). How to Create a Kubernetes Cluster on Ubuntu 16.04 with kudadm and Weave Net | Gremlin Community. [en línea] Gremlin.com. Recuperado de: <https://www.gremlin.com/community/tutorials/how-to-create-a-kubernetes-cluster-on-ubuntu-16-04-with-kudadm-and-weave-net/> [Último acceso 6 Abril. 2018].
- Cephalin, Ranieuve, Kriscrider and Robvanuden (2017). Use a custom Docker image for Web App for Containers - Azure. [en línea] Docs.microsoft.com. Recuperado de: <https://docs.microsoft.com/en-us/azure/app-service/containers/tutorial-custom-docker-image> [Último acceso 1 May 2018].
- Chekin, P. (2017). Multi-container pods and container communication in Kubernetes. [en línea] Mirantis | Pure Play Open Cloud. Recuperado de: <https://www.mirantis.com/blog/multi-container-pods-and-container-communication-in-kubernetes/> [Último acceso 8 May 2018].
- Cloud, G. (2018). Deploying a containerized web application | Kubernetes Engine Tutorials | Google Cloud. [en línea] Google Cloud. Recuperado de: <https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app> [Último acceso 1 Mayo 2018].

Cobos, A. and Nebrera, P. (2014). Despliegue de arquitectura cloud basada en OpenStack y su integración con Chef sobre CentOS. [en línea] Bibing.us.es. Recuperado de: <http://bibing.us.es/proyectos/abreproy/90140/fichero/Memoria.pdf> [Último acceso 10 Marzo. 2018].

CoreOS (2017). CoreOS. [en línea] Coreos.com. Recuperado de: <https://coreos.com/os/docs/latest/booting-on-openstack.html> [Último acceso 28 Marzo. 2018].

Dell (2015). Updating Dell PowerEdge servers via bootable media / ISO | Dell UK. [en línea] Dell.com. Recuperado de: <http://www.dell.com/support/article/co/es/cobsdt1/sln296511/updating-dell-powerededge-servers-via-bootable-media-iso?lang=en#2> [Último acceso 18 Febrero. 2018].

Dev, p. (2017). Introducción y ejemplo de cluster de contenedores con Docker Swarm. [en línea] Picodotdev.github.io. Recuperado de: <https://picodotdev.github.io/blog-bitix/2017/03/introduccion-y-ejemplo-de-cluster-de-contenedores-con-docker-swarm/> [Último acceso 1 Marzo. 2018].

Docs, D. (2018). Get Docker CE for Ubuntu. [en línea] Docker Documentation. Recuperado de: <https://docs.docker.com/install/linux/docker-ce/ubuntu/> [Último acceso 7 May 2018].

Docs, O. (2017). Gerrit Code Review. [en línea] Review.openstack.org. Recuperado de : <https://review.openstack.org/#/c/105660/10/specs/juno/dhcp-relay.rst> [Último acceso 25 Jan. 2018].

Docs, D. (2017). OpenStack. [en línea] Docker Documentation. Recuperado de: <https://docs.docker.com/machine/drivers/openstack/> [Último acceso 4 Enero. 2018].

Docs, O. (2017). OpenStack Docs: Developer Quick-Start. [en línea] Docs.openstack.org. Recuperado de: <https://docs.openstack.org/magnum/latest/contributor/quickstart.html> [Último acceso 27 Mar. 2018].

Docs, O. (2017). OpenStack Docs: Launch an instance. [en línea] Docs.openstack.org. Recuperado de: <https://docs.openstack.org/magnum/latest/install/launch-instance.html> [Último acceso 7 Febrero. 2018].

Docs.openstack.org. (2018). OpenStack Docs: Install OpenStack services. [en línea] Recuperado de: <https://docs.openstack.org/install-guide/openstack-services.htmlminimal-deployment> [Último acceso 28 Marzo. 2018].

Docs.openstack.org. (2018). OpenStack Docs: OpenStack Installation Guide. [en línea] Recuperado de: <https://docs.openstack.org/install-guide/> [Último acceso 28 Marzo. 2018].

Docs.openstack.org. (2018). OpenStack Docs: OpenStack Installation Guide Pike. [en línea] Recuperado de: <https://docs.openstack.org/pike/> [Último acceso 16 Mayo. 2018].

Ellison, Joseph. (2009). New and Improved check_mem.pl Nagios Plugin [en línea] Sysadminsjourney.com. Recuperado de: <http://sysadminsjourney.com/content/2009/06/04-new-and-improved-checkmempl-nagios-plugin/> [Último acceso 3 Marzo. 2018].

El Despistado. (2017). Nagios Core 4 + PNP4Nagios. Instalación y configuración desde fuentes en Debian 7 (wheezy). - El Despistado.. [en línea] Recuperado de: <http://www.eldespistado.com/nagios-core-4-pnp4nagios-instalacion-configuration-desde-fuentes-debian-7-wheezy/> [Último acceso 13 Mayo 2018].

Fedora (n.d.). Index of /pub/Mirrors/alt.fedoraproject.org/atomic/stable. [en línea] Ftp-stud.hs-esslingen.de. Recuperado de: <https://ftp-stud.hs-esslingen.de/pub/Mirrors/alt.fedoraproject.org/atomic/stable/> [Último acceso 19 Jan. 2018].

Galstad, E. (2017). NRPE DOCUMENTATION. [en línea] Assets.nagios.com. Recuperado de: <https://assets.nagios.com/downloads/nagioscore/docs/nrpe/NRPE.pdf> [Último acceso 16 Mayo 2018].

Geekk, G. (2017). Inter VLAN Routing by Layer 3 Switch - GeeksforGeeks. [en línea]

GeeksforGeeks. Recuperado de: <https://www.geeksforgeeks.org/inter-vlan-routing-layer-3-switch/> [Último acceso 9 Abril. 2018].

HPE-3com (2008). Command Reference Guide. [En línea] Recuperado de: http://h20628.www2.hp.com/km-ext/kmcstdirect/emr_na-c02579470-1.pdf [Último acceso 25 Abril. 2018].

Kalsin, V. (2017). How To Install Nagios 4 and Monitor Your Servers on Ubuntu 16.04 DigitalOcean. [en línea] Digitalocean.com. Recuperado de: <https://www.digitalocean.com/community/tutorials/how-to-install-nagios-4-and-monitor-your-servers-on-ubuntu-16-04> [Último acceso 7 Febrero. 2018].

Kubernetes.io. (2017). Installing kubeadm. [en línea] Recuperado de: <https://kubernetes.io/docs/tasks/tools/install-kubeadm/> [Último acceso 19 Febrero. 2018].

Kumar, P. and Kumari, M. (2017). Containers in OpenStack. [en línea] Packtpub.com. Recuperado de: https://www.packtpub.com/mapt/book/virtualization_and_cloud/9781788394383 [Último acceso 16 Enero. 2018].

Loges (2018). Steps to Install Kubernetes Dashboard - Assistanz. [en línea] Assistanz. Recuperado de: <https://www.assistanz.com/steps-to-install-kubernetes-dashboard/> [Último acceso 24 Abril. 2018].

Lopez, B. (2017). Notificaciones de Nagios vía Telegram - Bosco López. [en línea] Bosco López. Recuperado de: <https://www.boscolopez.com/notificaciones-de-nagios-via-telegram/> [Último acceso 10 Mayo 2018].

Lu, H., Teng, Q., Qiao, E. and Kumari, M. (2018). Magnum is not the OpenStack Container Service? How about Zun. [en línea] Openstack.org. Recuperado de: <https://www.openstack.org/assets/presentation-media/zunpresentationbarcelonasummit-3.pdf> [Último acceso 1 Mayo 2018].

Mas, O. (2018). Kubernetes: Heapster Influx Grafana - El Blog de Jorge de la Cruz. [en

[en línea] El Blog de Jorge de la Cruz. Recuperado de: <https://www.jorgedelacruz.es/2018/01/23/kubernetes-heapster-influx-grafana/> [Último acceso 5 Mayo 2018].

Moose (2017). 安裝Openstack ZUN模塊. [en línea] Pystack.org. Recuperado de: <http://pystack.org/2017/10/14/install-openstack-zun/> [Último acceso 6 Abril. 2018].

OpenStack. (2018). OpenStack - Kubernetes Integration, What Is Left?. [en línea] Recuperado de: <https://www.openstack.org/videos/boston-2017/openstack-kubernetes-integration-what-is-left> [Último acceso 16 Mayo 2018].

Pasandideh, M. (2018). 6. Weave UI - Developer Wiki - Confluence. [en línea] [Wiki.onap.org](https://wiki.onap.org/display/DW/6.+Weave+UI). Recuperado de: <https://wiki.onap.org/display/DW/6.+Weave+UI> [Último acceso 5 Mayo 2018].

People, F. (2017). Index of /groups/magnum. [en línea] Fedorapeople.org. Recuperado de: <https://fedorapeople.org/groups/magnum/> [Último acceso 10 Febrero. 2018].

Rh-atomix-bot, l. (2018). projectatomic/atomic-system-containers. [en línea] GitHub. Recuperado de: <https://github.com/projectatomic/atomic-system-containers> [Último acceso 3 Abril. 2018].

Rocy, l. (2017). projectatomic/atomic-system-containers. [en línea] GitHub. Recuperado de: <https://github.com/projectatomic/atomic-system-containers> [Último acceso 8 Febrero. 2018].

Saikatgithub (2017). Not able to access dashboard from external browser · Issue #2034 · kubernetes/dashboard. [en línea] GitHub. Recuperado de: <https://github.com/kubernetes/dashboard/issues/2034> [Último acceso 2 May 2018].

Sayalilunkad (2017). troubleshooting-diff. [en línea] Gist. Recuperado de: <https://gist.github.com/sayalilunkad/17913a0bc256b81d2e670fedd9db67df> [Último acceso 29 Abril. 2018].

Sjfbjs (2018). 使用kubeadm部署kubernetes集群. [en línea] blog.51cto.com. Recuperado de: <http://blog.51cto.com/11886896/2072527> [Último acceso 3 Mayo 2018].

Stangl, D. (2018). 5. Kubernetes UI - Developer Wiki - Confluence. [en línea] Wiki.onap.org. Recuperado de: <https://wiki.onap.org/display/DW/5.+Kubernetes+UI> [Último acceso 25 Abril. 2018].

Support.nagios.com. (2018). Nagios Core - Installing Nagios Core From Source. [en línea] Recuperado de: <https://support.nagios.com/kb/article/nagios-core-installing-nagios-core-from-source-96.html#Ubuntu> [Último acceso 16 Mayo 2018].

The Network Journal. (2012). Nagios: It appears as though you do not have permission to view information for any of the hosts you requested. [en línea] Recuperado de: <https://cyruslab.net/2012/10/19/nagios-it-appears-as-though-you-do-not-have-permission-to-view-information-for-any-of-the-hosts-you-requested/> [Último acceso 9 Marzo 2018].

The Network Journal. (2018). Nagios: It appears as though you do not have permission to view information for any of the hosts you requested. [en línea] Recuperado de: <https://cyruslab.net/2012/10/19/nagios-it-appears-as-though-you-do-not-have-permission-to-view-information-for-any-of-the-hosts-you-requested/> [Último acceso 9 Enero 2018].

Timme, F. (2013). Setting Up A High-Availability Load Balancer (With Failover and Session Support) With HAProxy/Heartbeat On Debian Etch. [en línea] Howtoforge.com. Recuperado de: <https://www.howtoforge.com/high-availability-load-balancer-haproxy-heartbeat-debian-etch> [Último acceso 2 Febrero. 2018].

World, S. (2018). Ubuntu 16.04 LTS : OpenStack Pike : Configure Cinder(Control Node) : Server World. [en línea] Server-world.info. Recuperado de: https://www.server-world.info/en/note?os=Ubuntu_16.04&p=openstack_pike2&f=1 [Último acceso 16 Mayo 2018].