PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR
FACULTAD DE INGENIERÍA

Trabajo de Titulación como requisito previo para la obtención del título de
Magíster en Tecnologías de Información mención Gestión y Administración de TI

Estudio y diseño de una solución para el despliegue de aplicaciones con opciones de software libre basado en contenedores (Kubernetes)

Autor: Ing. Gustavo Malla
Director: Mgtr. Charles Escobar

Quito, 07 de julio 2022

ÍNDICE DE CONTENIDOS

Contenido

INTRODUCCIÓN

En los últimos años el uso de las nuevas tecnologías ha evolucionado camino a los microservicios esto debido a la necesidad de satisfacer la demanda de los usuarios y las empresas. Del mismo modo que las organizaciones empresariales evolucionan en cómo hacen las cosas, las aplicaciones ligadas a los procesos de dichas empresas deben seguir el mismo camino para lograr su único objetivo, satisfacer dichas necesidades y servir al usuario final de la forma más eficiente.

Las grandes empresas líderes en tecnología han desarrollado sus infraestructuras en arquitecturas basadas en microservicios. Esto le da una gran ventaja frente a la arquitectura monolítica, por ejemplo, el despliegue aislado de cada componente de la aplicación le da una mayor flexibilidad al momento que la aplicación enfrente nuevos niveles de escalabilidad, así como también facilitar las actualizaciones al no tener que parar todo el proyecto, únicamente el módulo a actualizar.

Los microservicios están estrechamente ligados al concepto de contenedores. Un contenedor es una unidad estandarizada donde se empaqueta las aplicaciones y que incluye todo lo necesario para que el software pueda ejecutarse. Sin necesidad de conocer cómo funciona internamente la aplicación puede ser versionado, reutilizado y replicado fácilmente tanto en ambientes de pruebas como en producción. El archivo de configuración de Docker bastará para adaptar el entorno de ejecución dónde va a ser escalado. A partir de ese fichero se puede generar una imagen que puede ser desplegada en un servidor en cuestión de segundos.

Gracias a esta tecnología se abre la oportunidad para el despliegue y desarrollo de servicios y aplicaciones con alta disponibilidad y máxima escalabilidad, ya que supone una manera de virtualización mucho más rápida y eficiente.

No basta con el despliegue de contenedores para asegurar una alta disponibilidad de las aplicaciones que corren sobre cada uno de ellos, ya que pudieran tener problemas como un alto tráfico de entrada que obligaría a escalar cada uno de estos servicios sin perder la continuidad del mismo. Para esto hace falta la orquestación de contenedores. Este servicio es conocido como Kubernetes y es usado cuando la cantidad de aplicaciones basadas en contenedores supera un límite imposible de administrar, en este caso su uso y dominio es fundamental.

Este proyecto surge de la necesidad de buscar un diseño en el cual las aplicaciones con arquitectura tanto monolíticas como de microservicios puedan contener sus servicios en una arquitectura escalable, de alta disponibilidad y con balanceo de carga, acompañada de la orquestación de contenedores por medio de kubernetes.

La motivación de este trabajo, por tanto, es desarrollar un diseño para el despliegue de aplicaciones basado en la arquitectura de los microservicios. Permitiendo de tal manera, la comunicación entre los distintos servicios que componen el diseño y así poder llevar a cabo su despliegue mediante kubernetes. De esta manera las aplicaciones desplegadas sobre este diseño serán escalables, mantendrán alta disponibilidad y serán redundante a fallos, además de otras ventajas que implica el uso de una plataforma como esta.

CAPÍTULO I: PLANTEAMIENTO DEL PROBLEMA

1.1. Formulación del problema

El desarrollo de sistemas complejos antes de la aparición de las tecnologías de microservicios se realizaba en base a una arquitectura monolítica, la cual su mantenimiento y escalado es muy complicada. Actualmente las empresas están migrando hacia las arquitecturas de los microservicios ya que ofrece grandes ventajas como la escalabilidad, la estandarización, el mantenimiento y agilidad.

Una arquitectura monolítica se construye como un sistema grande con una única base de código y se implementa como una sola unidad, generalmente detrás de un balanceador de carga. Por lo general este tipo de arquitecturas consta de cuatro componentes básicos, una interfaz de usuario, lógica empresarial, una interfaz de datos y una base de datos. Por lo cual, el momento que necesite un nuevo cambio, modificación o despliegue, se debería lanzar todo el sistema de nuevo.

De manera resumida se puede decir que la arquitectura monolítica es aquella en la cual el software está estructurado de tal manera que la mayoría de sus funciones quedan unidos y sujetos en un mismo programa. Toda la información que necesita este sistema para su trabajo queda alojada de forma estable en un único servidor.

Actualmente este tipo de soluciones trae consigo varios inconvenientes, ya que al ser sistemas monolíticos la implementación con tecnologías actuales como Devops se vuelve muy compleja y hasta imposible de realizar. Es por ello que existe la arquitectura de microservicios, una opción bastante efectiva en el desarrollo de software.

Se podría decir que los microservicios son una evolución del SOA (Service Oriented Architecture), cuya función principal se basa en el desarrollo de servicios independientes, y cada uno de ellos asociados a una misma aplicación.

A diferencia de SOA, los microservicios permiten que cada uno de sus componentes, sean actualizados, sustituidos, eliminados o dotados de mayores recursos de acuerdo a las necesidades de su entorno.

Implementar una infraestructura basada en microservicios no es complicado, ya que se tiene muchas tecnologías disponibles para su implementación. A nivel de código se tiene Spring Boot, Django, Lumen framework, a nivel de infraestructura, Docker, Kubernetes, Open Shift, Rancher, Ha Proxy, Apache, entre otros. Todas estas tecnologías ayudan o minimizan a solventar los problemas relacionados con su implementación.

Esta tecnología de microservicios se basa en un componente especial, los contenedores. Desde aproximadamente 20 años atrás se ha utilizado aplicaciones contenerizadas, de tal manera que se ha podido aprovechar los beneficios que esta tecnología dispone, por ejemplo, el aislamiento de recursos, es decir, que se puede definir límites para cada uno de sus recursos como la memoria, cpu, red, etc.

Con esta tecnología se puede tener la aplicación corriendo bajo una infraestructura, escalable, con alta disponibilidad, y con balanceo de carga. Además, con un despliegue que no necesita código ni configuraciones complejas se puede tener la aplicación corriendo sin necesidad de dependencias. Dada las capacidades de aislamiento que presenta la tecnología de microservicios se puede tener más de un contenedor corriendo en las instancias.

Pero surge un problema, de cierta manera, es posible que se pueda administrar de forma manual unas cuantas máquinas virtuales o servidores físicos, pero no hay manera de administrar de forma correcta un entorno de aplicaciones contenerizadas en ambientes de producción sin automatización, para resolver este problema se creó la orquestación de contenedores.

Como se comentó en el apartado anterior esta tecnología despliega varias instancias de contenedores dentro del clúster. En la actualidad la orquestación de contenedores se encarga de realizar auto escalamiento, comprobaciones de salud de cada pod, mantener el estado deseado y se encarga de buscar el mejor nodo worker para el despliegue de los pods. Es decir, si se tiene una aplicación que requiere 10vcpu y 10Gb de memoria, el orquestador se encargará de buscar el mejor worker dentro del pool de nodos.

La arquitectura de esta tecnología se basa en un clúster master-worker. Donde los nodos masters guardan el estado actual del clúster y los nodos workers es el lugar donde se alojan las aplicaciones.

La motivación de este proyecto es poder estudiar todas las posibles soluciones y poder plantear un diseño de arquitectura que soporte el despliegue de contenedores y su correcta orquestación por medio de kubernetes. De esta manera se tendrá un diseño que permitirá ejecutar gran parte de las funcionalidades y conceptos de una arquitectura monolítica, permitiendo una actualización controlada de la misma, hacia los microservicios.

1.2. Objetivos de la Investigación

Objetivo General

- Estudiar y diseñar una solución para el despliegue de aplicaciones con software libre basado en contenedores

Objetivos Específicos

- Definir los requerimientos que sean adaptables para el diseño en una arquitectura de microservicios.
- Diseñar un modelo con capacidades de conmutación por error y monitorización entre sí.
- Construir el diseño que soporte la arquitectura propuesta.
- Evaluar el diseño de la solución propuesta para el despliegue de la arquitectura de microservicios.

1.3. Justificación de la Investigación

Actualmente las empresas han venido desarrollando sus sistemas bajo un diseño de arquitectura monolítica que, a pesar de su bajo costo en el desarrollo, el poco margen de fallos, la sencilles para su despliegue y ejecución, su actualización se vuelve una tarea complicada debido a que este tipo de arquitecturas están desarrolladas sobre una base rígida entre sus componentes. Además, la escalabilidad no es un tema fácil ya que al no estar desarrollada por módulos (como la arquitectura de microservicios) el despliegue independiente de cada uno de sus componentes se vuelve un desafío. La administración en este tipo de arquitecturas demanda de recursos, dinero y tiempo ya que si se habla de varias aplicaciones es necesario contar con un sistema de monitoreo centralizado.

Por esta razón se propone un diseño de arquitectura que permita:

- Una correcta administración.
- Que sea redundante ante los fallos y disponga de balanceo de carga.
- Mayores tiempos de actividad de las aplicaciones.
- Incrementar las capacidades de cómputo de acuerdo a las necesidades.
- Menos tiempo en el monitoreo por parte del personal de TI.
- Tener una administración centralizada e intuitiva.

CAPÍTULO II: MARCO TEÓRICO

2.0. Herramientas y Tecnología

Una vez que se ha planteado los objetivos principales de este proyecto es necesario seleccionar y detallar las distintas tecnologías a usar para la implementación del diseño.

2.1. Docker

Es una plataforma de código abierto para crear, implementar, administrar y automatizar despliegues de aplicaciones virtualizadas en un sistema operativo común. A diferencia de las máquinas virtuales que proporcionan una capa abstracta de recursos físicos sobre el cual se encapsula un sistema operativo completo con código ejecutable. Docker utiliza el aislamiento de recursos en el kernel para ejecutar varios contenedores sobre un mismo sistema operativo.

Al ser unidades estandarizadas de software permite que estos contenedores puedan ser ejecutados en cualquier entorno, ya sea sistema operativo, servidor, proveedor de nube, etc. Mientras un contenedor puede necesitar varios megas en memoria para su despliegue, una máquina virtual necesitara varias gigas de memoria.

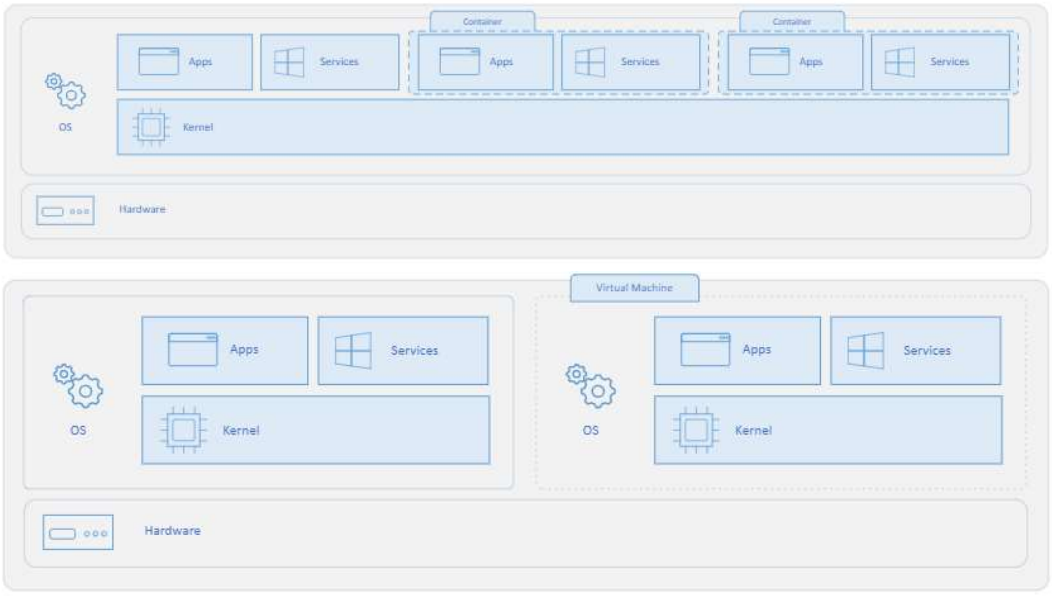

Figura 1.- Gerend, J. (2020, 22 septiembre). Contenedores frente a máquinas virtuales. learn.microsoft.com. https://learn.microsoft.com/es-es/virtualization/windowscontainers/about/containers-vs-vm

Esto no solo refleja mejoras en el espacio y recursos que consumen estos contenedores, también brindan la capacidad de diseñar sistemas adaptables a las arquitecturas basadas en microservicios, y cada uno de estos ser replicados en base al diseño que se está desarrollando. A diferencia de las máquinas virtuales se disminuye el consumo excesivo de los recursos y el tiempo que le toma a una aplicación realizar un despliegue.

Una de las virtudes que tienen los contenedores como método de virtualización es que son: Escalables, es decir, se pueden aumentar y distribuir automáticamente dentro de cualquier entorno. Portables, es decir, su ejecución se la puede realizar en cualquier entorno. Intercambiables, permite el despliegue de actualizaciones en caliente y ligeros ya que comparten el kernel del equipo host.

La manera en cómo Docker realiza el despliegue de sus contenedores es utilizando imágenes Docker. Una imagen es un file que contiene un sistema de ficheros para la creación de un container, es decir, contiene el código, las librerías, las variables, los archivos de configuración y los runtimes que necesita para su despliegue.

Estás imágenes son almacenadas en los registros públicos de Google, Amazon y otros privados como Harbor. Además, Docker permite subir imágenes a su propio repositorio llamado Docker Hub, en el cual se puede subir y descargar imágenes personalizadas que sirven como plantillas, de esta manera se evita la implementación de cero de una imagen.

Estas imágenes son almacenadas localmente en el host y son administradas por Docker cuando se necesite desplegar un contenedor que necesite de estas imágenes. Su funcionamiento se aprecia mejor en la siguiente imagen. Figura 2.

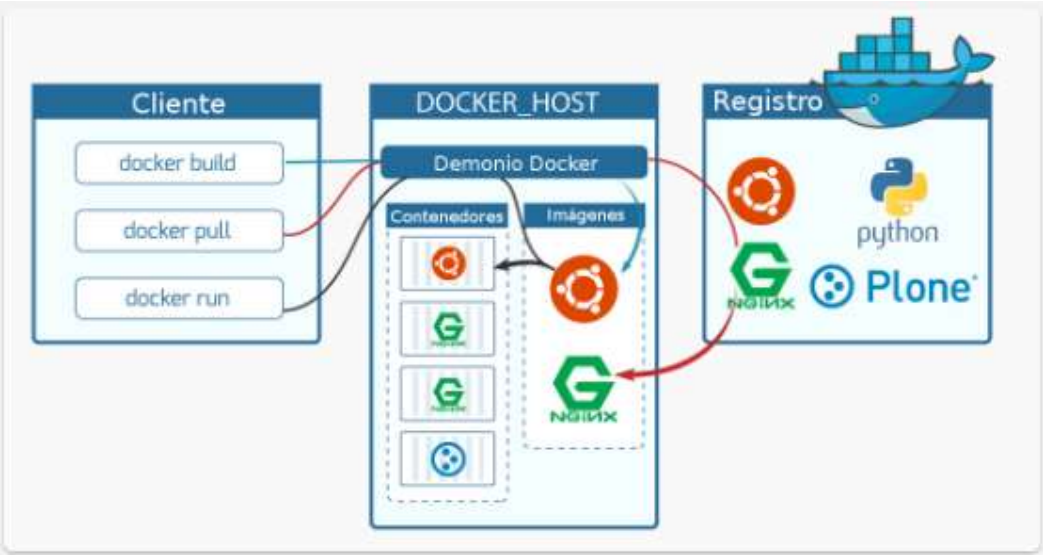

Figura 2.- Caballero, L. (2020, 19 noviembre). Funcionamiento básico de docker. lcaballero.wordpress.com. https://lcaballero.wordpress.com/2020/11/19/comando-basicos-de-docker-en-debian-10/

Docker además proporciona un sin número de comandos para su administración, siendo alguno de estos los mostrados en la figura 3 los comandos básicos para trabajar con Docker.

Comando	Acción
docker run <imagen>	Crea un contenedor a partir de una imagen.
docker run -d -p "puerto1:puerto2"	Crea un contenedor accessible desde el puerto 1 al puerto 2
docker stop/start <nombre o id>	Detiene o inicia un contenedor
docker ps -a	Indica la lista de contenedores desplegados, inclusive los parados
docker rm <nombre o id>	Elimina un contenedor
docker exec -it >nombre o id> bash	Abre la terminal del contenedor
docker load -i <nombre de la imagen>	Carga una imagen
docker images	Lista todas las imágenes descargadas

Figura 3.- Comandos básicos Docker

Los contenedores son volátiles, es decir, una vez eliminados toda la información contenido en ellos se pierde. Por ello, existen los volúmenes y Docker permite su utilización para el almacenamiento de su información y persistencia en el disco duro, esto quiere decir, que a pesar que el contenedor sea eliminado la información permanecerá almacenada en el disco hasta que otro contenedor sea creado y pueda volver a leer dicha información. Esta configuración es básica para el desarrollo de aplicaciones y puestas en producción.

A continuación, se indica dos maneras de montar los volúmenes en Docker.

docker volume create <nombre del volumen>
--**mount source**=<nombre del volumen>, **target**=/ruta/destino/de/la/información

-v /ruta/destino/de/la/información:/ruta/interna/del/contenedor

De esta manera se crea un volumen y se lo asigna a una ruta especifica en el disco duro.

2.2. Kubernetes

Plataforma open source lanzado por Google en el año 2014 que propone una solución para el despliegue, escalado y gestión de aplicaciones por medio de la orquestación de contenedores. Esto debido a la carencia en la administración de aplicaciones que contienen gran cantidad de containers y servicios.

De tal manera que permite administrar los contenedores y llevar tareas como donde debe colocarse un contenedor, monitorización de los contenedores, reinicio en caso de presentar un problema, escalamiento de forma automática y una lista de más acciones posibles.

Los dos recursos principales constan en la Figura 4.

- Los nodos Master coordinan el clúster y las actividades del mismo. Entre las tareas principales del nodo master están: mantener el estado deseado de las aplicaciones, organizar cada aplicación que será lanzada dentro del clúster, escalar las aplicaciones de acuerdo a los recursos que dispone cada nodo, etc. También se encarga de recopilar información de los nodos workers y de los pods.

- Los nodos Worker ejecutan las aplicaciones. Estos contienen un agente llamado Kubelet el cual se encarga de la gestión del nodo y mantiene la comunicación con los nodos masters. Son los nodos que reciben mayor cantidad de carga ya que las aplicaciones son depositadas en ellos dependiendo de los recursos disponibles.

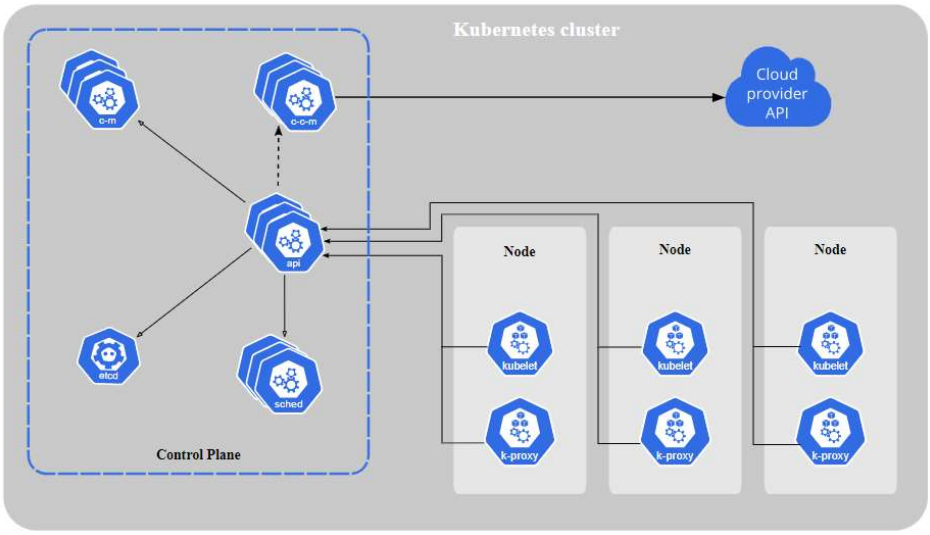

Figura 4.- Kubernetes Components. (2022, 23 julio). kubernetes.io. https://kubernetes.io/docs/concepts/overview/components/

Kubernetes presenta diferentes objetos básicos llamados controladores, los cuales le permite realizar varios despliegues en los diferentes nodos del clúster. Estos elementos son:

- **Deployment**. – Un deployment es un objeto que puede representar una aplicación del clúster. En base a una configuración se pide a kubernetes que actualice o cree una instancia de una aplicación. Tras ejecutar el deployment, el nodo Master decide en que nodo worker disponible de clúster se puede lanzar la aplicación. Una función del controlador de deployment es monitorizar continuamente las instancias de la aplicación, es decir, si por algún motivo el nodo donde se encuentra levantada la instancia se cae, el controlador del deployment de kubernetes la sustituye en otro nodo worker que se encuentre disponible del clúster. Esta funcionalidad de alta disponibilidad en los pods sugiere una manera diferente en la administración de las aplicaciones.

  Para crear un deployment se debe especificar en su spec la imagen del container, su nombre y el número de replicas que se quiere correr. Figura 5.

```yaml
apiVersion: apps/v1 # Usa apps/v1beta2 para versiones anteriores a 1.9.0
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # indica al controlador que ejecute 2 pods
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Figura 5.- Kubernetes Concepts. (2021, 16 agosto). kubernetes.io. https://kubernetes.io/es/docs/concepts/workloads/controllers/deployment/

- **Pods**. – "Un Pod es una unidad atómica de kubernetes y se trata de una abstracción que representa un grupo de uno o más contenedores y algunos recursos compartidos de esos contenedores (por ejemplo, red, volúmenes, ingress, puertos etc.)" (López, 2019, p. 14).

  Todos los contenedores embebidos dentro del pod comparten una IP, puertos, y su despliegue es en un mismo nodo. Por su condición de efímeros pierden la información al eliminarse. Figura 6.

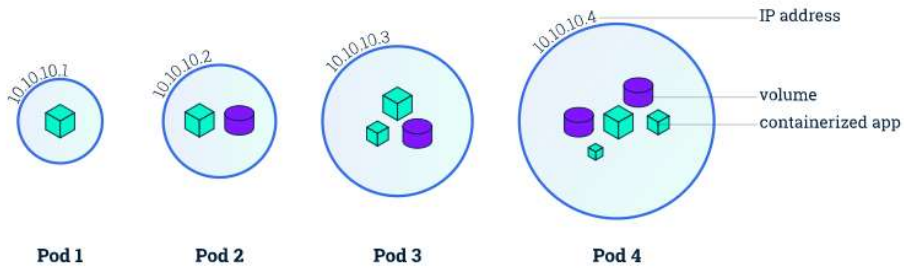

Figura 6.- Kubernetes Components. (2022, 2 octubre). kubernetes.io. https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/

- **ReplicaSets. –** Un ReplicaSet es un objeto de Kubernetes que se utiliza para mantener un conjunto estable de pods replicados que se ejecutan dentro de un clúster en un momento dado. De tal manera que se garantiza la disponibilidad de un número determinado de pods similares. Un ReplicaSet es definido por campos, tales como: un selector que se utiliza para identificar de que pods es responsable este ReplicaSet, plantilla define la plantilla de pods que usara el ReplicaSet para crear nuevos pods y cumplir con la cantidad requerida de replicas. Se puede decir que el propósito de un ReplicaSet en términos de uso es garantizar que los pods estén disponibles, así como su creación y eliminación para alcanzar el número esperado. Figura 7.

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modifica las réplicas según tu caso de uso
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

Figura 7.- Kubernetes Concepts. (2020, 10 noviembre). kubernetes.io. https://kubernetes.io/es/docs/concepts/workloads/controllers/replicaset/

- **Services. –** Dentro del clúster kubernetes cada pod tiene una dirección IP interna. Sin embargo, los pods son efímeros y sus direcciones IP cambian cada que son eliminados. Por tal motivo usar las direcciones IP internas no tiene sentido. Un servicio nos da la ventaja de mantener una IP estable que dura toda la vida del servicio incluso cuando las direcciones IP de los pods cambien. Además, otro servicio que proporciona es el balanceo de carga. Es decir, se hace una llamada a

la dirección IP del servicio expuesto y las solicitudes son balanceadas entre los pods miembros de ese servicio. Los pods son agrupados mediante etiquetas y selectores.

Existen distintos tipos de servicios tales como:

- **ClusterIP:** Expone el servicio en una IP interna del clúster. De esta manera el servicio es alcanzable solo dentro del clúster. Figura 8.

```
apiVersion: v1
kind: Service
metadata:
  name: "service-devops"
spec:
  selector:
    app: devops
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Figura 8.- Estructura del ClusterIP.

- **NodePort:** Expone el servicio en cada IP del nodo mediante el puerto interno del pod y el puerto que se quiere exponer en el rango 30000 y 32767. Figura 9.

```
apiVersion: v1
kind: Service
metadata:
  name: my-nodeport-service
spec:
  selector:
    app: my-app
  type: NodePort
  ports:
  - name: http
    port: 80
    targetPort: 80
    nodePort: 30036
    protocol: TCP
```

Figura 9.- Estructura del NodePort.

- **LoadBalancer:** Expone el servicio mediante un balanceador del proveedor de la nube. Figura 10.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  selector:
    app: example
  ports:
    - port: 8765
      targetPort: 9376
  type: LoadBalancer
```

Figura 10.- Estructura del LoadBalancer

- **ExternalName:** Mapea el servicio al campo externalName. Figura 11.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ExternalName
  externalName: my.database.example.com
```

Figura 11.- Estructura del ExternalName.

- **Volúmenes. –** "Los volúmenes son un almacenamiento externo que hace referencia a la estructura interna del pod y que persiste en el tiempo. Básicamente, un volumen es un directorio, posiblemente con algunos datos, al que pueden acceder los contenedores de un pod. La forma en que ese directorio llega a ser, el medio que lo respalda y el contenido del mismo están determinados por el tipo de volumen particular utilizado". Recuperado el 12 de septiembre del 2022, de https://kubernetes.io/docs/concepts/storage/volumes/

- **ConfigMaps. –** Se utiliza para almacenar datos no confidenciales en forma de pares clave-valor para que los pods los puedan consumir como variables de entorno posteriormente en despliegues parametrizados. Son utilizados para almacenar datos no sensibles ya que se guardan sin ningún tipo de codificación como ocurre en los secrets. Figura 12.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"
  #
  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

Figura 12.- Kubernetes Concepts. (2021, 18 abril). kubernetes.io. https://kubernetes.io/es/docs/concepts/configuration/configmap/

- **Secrets. –** Permite guardar información confidencial como passwords, claves ssh, y otros elementos que requieran codificación. A pesar que no cifra los datos contenidos en estos objetos, si codifica en base64. Figura 13.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysqlpassword
type: Opaque
data:
  password: cGFzc3dvcmQ=
```

Figura 13.- Ejemplo de Secret.

Estos elementos, así como la información de los propios clústeres son almacenados en la base de datos etcd propia de kubernetes. Además, pueden ser creados sobre la marcha mediante la implementación de YAML. Estos archivos nos permiten automatizar el despliegue de las aplicaciones. También de manera más interactiva se los puede crear por medio de Rancher, de esta manera es mucho más simple su implementación y su configuración.

2.3. Rancher

Es un software para equipos que trabaja con contenedores. Facilita el aprovisionamiento y la administración de los clústeres de Kubernetes permitiendo administrar su seguridad, crear usuarios, implementar métodos de autenticación externos como LDAP, asignar permisos, etc. "Aborda los desafíos operativos y de seguridad de administrar múltiples clústeres de Kubernetes, al tiempo que brinda a los equipos de DevOps herramientas integradas para ejecutar cargas de trabajo en contenedores". Rancher. (s.f) Why Rancher. https://www.rancher.com/why-rancher

Además, permite agilizar la implementación de clústeres en nubes privadas, nubes públicas o vSphere y protegerlos mediante políticas de seguridad globales.

Utiliza Helm o un catálogo propio para implementar y administrar aplicaciones en cualquiera de estos entornos o en todos, lo que garantiza la coherencia de múltiples clústeres con una sola implementación.

Se decidió utilizar Rancher por varios motivos tales como: la facilidad en la instalación, el apoyo, el soporte que proporciona y el conocimiento previo de haber instalado esta solución en varios proyectos personales.

Además, consta dentro de los líderes recibiendo las mejores calificaciones por:

- Tiempo de ejecución y orquestación.
- Características de Seguridad.
- Gestión de Imágenes.
- Visión.

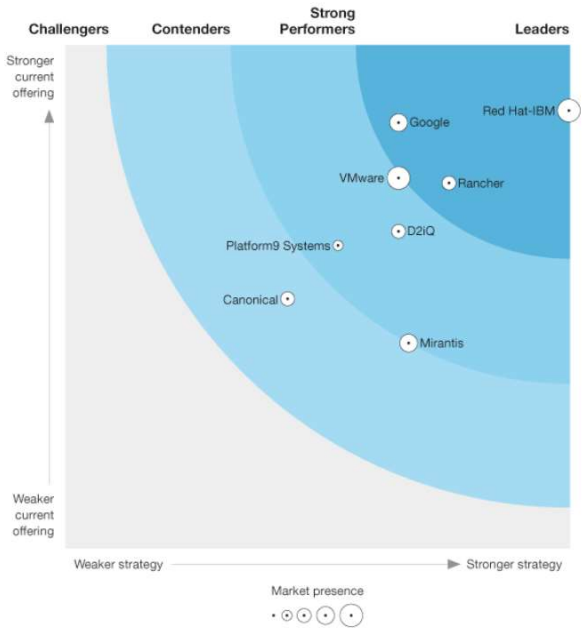

Figura 14.- Rancher Concepts. (2022). Rancher. https://www.rancher.com/why-rancher

La instalación y configuración más detallada de Rancher se describe en el capítulo 4.

2.4. HaProxy

“Haproxy es un proxy inverso gratuito, muy rápido y fiable que ofrece alta disponibilidad , equilibrio de carga y proxy para aplicaciones basadas en TCP y HTTP. Es particularmente adecuado para sitios web de muy alto tráfico y alimenta una parte significativa de los más visitados del mundo. A lo largo de los años, se ha convertido en el equilibrador de carga open source estándar de facto, ahora se envía con la mayoría de las distribuciones principales de Linux y, a menudo, se implementa de forma predeterminada en plataformas en la nube. Presenta algunas características que serán usadas en este proyecto”. HAProxy El balanceador de carga TCP/HTTP confiable y de alto rendimiento. (2022, 1 diciembre). HAProxy Community Edition. Recuperado 27 de enero de 2023, de http://www.haproxy.org/#desc

- **Lista de Control de Acceso (ACL)**

  Las ACL dentro del funcionamiento de un equilibrador de carga son usadas para probar diferentes reglas, condiciones o realizar alguna acción, por ejemplo, para el caso del diseño propuesto se usa para seleccionar un nodo worker. El uso de las ACL admite el reenvío del tráfico en función de una variedad de factores, por ejemplo, patrones que coinciden con una regla especifica o la cantidad de conexiones a un backend.

- **Back-end**

  Un backend es una granja o conjunto de servidores que reciben y procesan las solicitudes de entrada de los clientes. Un backend consta de uno o varios servidores. De tal manera que agregar más nodos en la granja implica aumentar la capacidad de carga potencial ya que esta será distribuida en todos los nodos del conjunto. Figura 15.

```
backend http_back
fullconn 10000
balance roundrobin
cookie JSESSIONID prefix nocache

#server worker_1 10.100.19.88:80 cookie worker_1 check
#server worker_2 10.100.19.91:80 cookie worker_2 check
server worker_1 192.168.0.14:80 cookie worker_1 check
```

Figura 15.- Definición del Backend.

- **Algoritmos de Equilibrio de Carga.**

  El algoritmo de equilibrio de carga que se utiliza determina qué servidor, en un backend, será seleccionado para equilibrar la carga. Haproxy ofrece varias opciones para algoritmos. Además del algoritmo de equilibrio de carga, a los servidores se les puede asignar un parámetro de *peso* para manipular la frecuencia con la que se selecciona el servidor, en comparación con otros servidores.

  - **Roundrobin.** – Este algoritmo selecciona los servidores backend por turnos. Es el algoritmo predeterminado.

  - **Leastconn**. – Con este algoritmo el servidor con el menor número de conexiones recibe la conexión. Recomendado cuando se esperan sesiones muy largas.

  - **Source**. – Algoritmo que selecciona un servidor a usar en función de un hash de la dirección IP de origen desde la que los usuarios realizan solicitudes. Este método asegura que los mismos usuarios se conectarán siempre a los mismos servidores.

- **Health Check.**

  Haproxy utiliza comprobaciones de estado para determinar si un servidor backend está disponible para procesar solicitudes. Esto evita tener que eliminar manualmente un servidor del backend si deja de estar disponible. La comprobación de estado predeterminada es intentar establecer una conexión TCP con el servidor.

  Si un servidor falla en una verificación de estado y, por lo tanto, no puede atender solicitudes, se deshabilita automáticamente en el backend y el tráfico no se reenviará hasta que vuelva a estar en buen estado. Si todos los servidores de un backend fallan, el servicio dejará de estar disponible hasta que al menos uno de esos servidores backend vuelva a estar en buen estado.

2.5. Apache HTTP Server

Un servidor web es un sistema de servidor en Internet que es responsable de manejar las solicitudes HTTP de clientes HTTP (por ejemplo, navegadores). HTTP significa Hyper-Text-Transfer-Protocol y especifica cómo deben verse las solicitudes al servidor y las respuestas del servidor. La comunicación entre un cliente y un servidor siempre comienza con una solicitud del cliente.

La primera línea de una solicitud HTTP contiene información sobre qué función quiere usar el cliente (por ejemplo, GET, HEAD, POST), un identificador uniforme de recursos (URI) diciéndole al servidor en qué recurso el cliente quiere usar esta función y finalmente información sobre la versión de HTTP. El resto de las líneas contiene los encabezados, que se utilizan para agregar información sobre la solicitud o modificarla.

El navegador del cliente y el servidor Apache utilizan TCP para establecer una conexión y transferir datos entre ellos. Los datos se envían en pequeños fragmentos llamados paquetes. TCP usa IP para enviar paquetes individuales al destino correcto. IP es un protocolo de mejor esfuerzo, lo que significa que intenta enviar los paquetes lo mejor que puede, pero sin ninguna garantía. Los paquetes enviados pueden o no recibirse en el destino final y lo más probable es que no lleguen en el mismo orden en que fueron enviados. Cuando la confiabilidad es importante, el protocolo IP debe combinarse con un protocolo de red confiable como TCP.

TCP asigna a cada paquete un número de secuencia específico para poder garantizar la entrega en orden. Cuando se recibe un paquete, el receptor enviará un acuse de recibo, ACK, de regreso al remitente, que contiene el siguiente número de secuencia que espera recibir. El remitente entonces estará seguro de que el paquete ha sido entregado y en qué número de secuencia se encuentra el receptor. Para hacer posible que el remitente no tenga que esperar un acuse de recibo para cada paquete individual antes de transmitir el siguiente paquete, TCP define una cantidad de bytes (llamada ventana de transmisión [20]) que puede enviar antes de esperar un acuse de recibo del receptor. Figura 16.

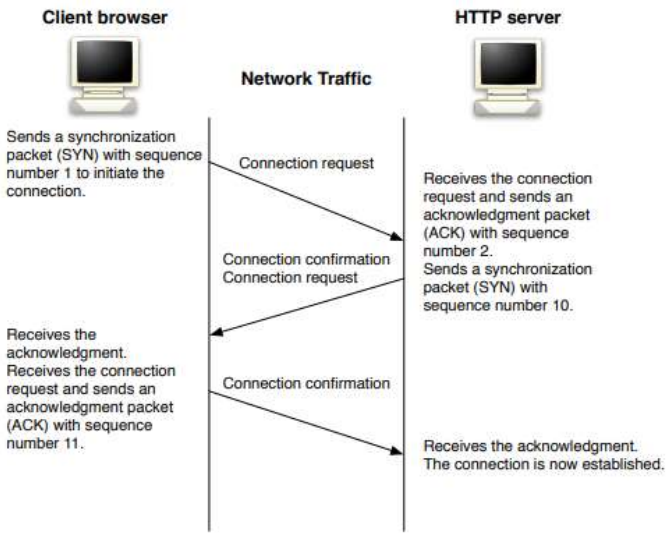

Figura 16.- Lindegren, E. (s. f.). Preparing the Apache HTTP Server for Feedback Control Application [MASTER THESIS]. Lund University.

CAPÍTULO III: MARCO METODOLÓGICO

3.0. Tipo de Investigación.

"El diseño es un proceso de búsqueda y de descubrimiento de nueva información sobre las alternativas que están disponibles y acerca de las consecuencias que se seguirán si se escogen esas alternativas. Pero el diseño es también un proceso de descubrimiento de metas a alcanzar y de restricciones a satisfacer. Las metas y restricciones no son más que elementos fijos del diseño [en mayor medida] que lo pueda ser cualquier otra cosa." (Simón H, 2007, p.159).

Por tal motivo este trabajo es considerado una investigación proyectiva, ya que, para su desarrollo, es necesario plantear una solución de diseño como respuesta a un problema y esto a partir de un proceso previo de indagación.

A través de este proceso se cumplió con el primero objetivo específico, que consiste en definir los requerimientos que sean adaptables para el diseño en una arquitectura de microservicios.

Además, se utilizó historias de usuario para poder definir cuál es el requerimiento específico para el usuario final y cuál es el objetivo esperado con dichos requerimientos, esto se profundizará en el capítulo 4.

3.1. Diseño de Investigación.

Para el desarrollo de esta arquitectura realizaremos algunas actividades.

1. **Definición de los requerimientos del diseño:** Se realizará la investigación, la priorización y la documentación de los requerimientos que de manera particular influyen sobre todos los componentes de la arquitectura.

2. **Diseño de la arquitectura:** Se define la estructura de la que se compone la arquitectura en base a los requerimientos previamente detallados y el ambiente en el cual se va a realizar la implementación.

3. **Documentación y socialización del diseño:** Es necesaria la capacitación, socialización y documentación del diseño a todos los interesados.

4. **Evaluación de la arquitectura**: Dado que el diseño de la arquitectura tiene un papel fundamental en su implementación, y para reducir los posibles riesgos en la misma, es aconsejable poner a evaluación dicho diseño cuando este ya ha sido documentado. La ventaja de evaluar el diseño antes de su implementación es que puede realizarse de manera temprana y las correcciones de los defectos identificados supondrían un costo menor al costo que tendrían corregirlos después de que el diseño ha sido implementado.

5. **Implementación de la arquitectura:** Cuando el diseño de la arquitectura haya sido establecido, se comienza con la implementación de la misma. Durante esta etapa lo mejor es evitar cambios con respecto del diseño original definido por el arquitecto.

3.2. Técnicas e instrumentos de recolección de datos.

Las técnicas de recolección de datos "comprenden procedimientos y actividades que le permiten al investigador obtener la información necesaria para dar respuesta a su pregunta de investigación" (Hurtado de Barrera, 2010, p.771). Con referencia a lo anterior, para el desarrollo de este proyecto se utilizó la revisión documental y la entrevista no estructurada.

Revisión Documental.

"La revisión documental constituye el punto de entrada a la investigación, incluso en ocasiones es el origen del tema o problema de investigación. Los documentos fuente pueden ser de naturaleza diversa: personales, institucionales o grupales, formales o informales" (Quintana, 1996, p.34).

Esta técnica nos permite revisar documentación previa referente al diseño de los sistemas basados en microservicios con el fin de aprender más sobre la funcionalidad básica que deben seguir estos sistemas. También se revisó los manuales y otras implementaciones ya realizadas lo cual fue de gran ayuda ya que se tenía conocimiento previo respecto al planteamiento del problema.

CAPÍTULO IV: ESTUDIO Y DISEÑO DE UNA SOLUCIÓN PARA EL DESPLIEGUE DE APLICACIONES CON SOFTWARE LIBRE BASADO EN CONTENEDORES

Este capítulo detalla todas las actividades que se llevarán a cabo para realizar el diseño de una solución para el despliegue de aplicaciones con opciones de software libre basado en contenedores.

4.0. Levantamiento de requerimientos.

4.0.1. Definición de los requerimientos del diseño

El presente trabajo fue desarrollado en base a las necesidades puntuales que hoy en día demandan las aplicaciones para mantener su alta disponibilidad, ser tolerante a fallos y tener la capacidad de escalar sin necesidad de requerir más recursos.

El trabajo no esta desarrollado para una empresa en específico, pero es adaptable a la necesidad de cualquier entorno siempre y cuando sea bajo arquitectura de microservicios.

En base a lo detallado anteriormente y bajo la metodología SCRUM se ha definido las historias de usuario que cumplan con el acrónimo INVEST, siendo independientes, negociables, valiosas, estimables, pequeñas, comprobables y que cubra la necesidad del desarrollo de este diseño.

- "*Como gerente de producto se requiere un diseño de infraestructura que permita la administración, el despliegue y la alta disponibilidad de todas las aplicaciones contenerizadas con el fin de alinear los objetivos del área con la planificación estratégica de la organización"*

Con la información obtenida mediate la revisión documental, se definieron varios requerimientos funcionales y no funcionales que debe cumplir el diseño. Esto fue ajustado en base a las historias de usuario.

1. **Requerimientos Funcionales.**

- *El diseño debe permitir el registro de nuevos nodos workers de tal manera que no altere su funcionamiento original.*

2. **Requerimientos No Funcionales.**

- *El diseño debe ser capaz de balancear la carga de manera equitativa a todos los nodos workers sin sobrecargar su funcionamiento.*

- *El diseño debe ser capaz de mantener una alta disponibilidad permitiendo cumplir con los SLA definidos por el área.*

- *El diseño debe tener un software para la administración y orquestación centralizada de aplicaciones contenerizadas de tal manera que su administración sea de manera sencilla.*

- *El diseño debe ser capaz de soportar múltiples peticiones por segundo de tal manera que no se vea afectado en su rendimiento.*

4.2. Diseñar un modelo con capacidades de conmutación por error y monitorización entre sí.

4.2.1. Diseño de la arquitectura.

Una vez establecidos los requerimientos que el sistema deberá cumplir, se tomaron en cuenta los componentes que formaran parte de la arquitectura para el despliegue de aplicaciones.

- **Servidor Apache HTTP Server.**

  El diseño contará en primera instancia con un servidor web apache. Este se encargará de ser la puerta principal de entrada para el consumo de las aplicaciones dentro del diseño, ya que aquí se configurará el dominio sobre el cual se va a trabajar. Además, se configurará un proxy reverso el cual ayudará a direccionar todas las peticiones de los contextos a los balanceadores correspondientes.

- **Balanceador de Carga Ha Proxy.**

  Otro componente que forma parte del diseño para el despliegue de aplicaciones es el balanceador de carga. Para esta arquitectura se utilizará dos balanceadores de carga. De esta manera se cumple con tres requerimientos no funcionales, mantener la alta disponibilidad, balancear la carga y soportar múltiples peticiones por segundo.

- **Nodo Despliegue.**

  El nodo de despliegue servirá para poder lanzar de manera centralizada la configuración. Aquí se detallarán patrones como que equipo será el que tome el rol de nodo master, que equipos serán los que tomen el rol de nodo workers y desde donde se enviara las llaves ssh de seguridad para cada equipo.

- **Nodos Master.**

  El nodo master se encarga de mantener el estado deseado en el clúster.

- **Nodos Workers.**

  Los nodos workers son los nodos en los cuales se lanzarán las aplicaciones.
  En la figura 17 se muestra la estructura general del diseño.

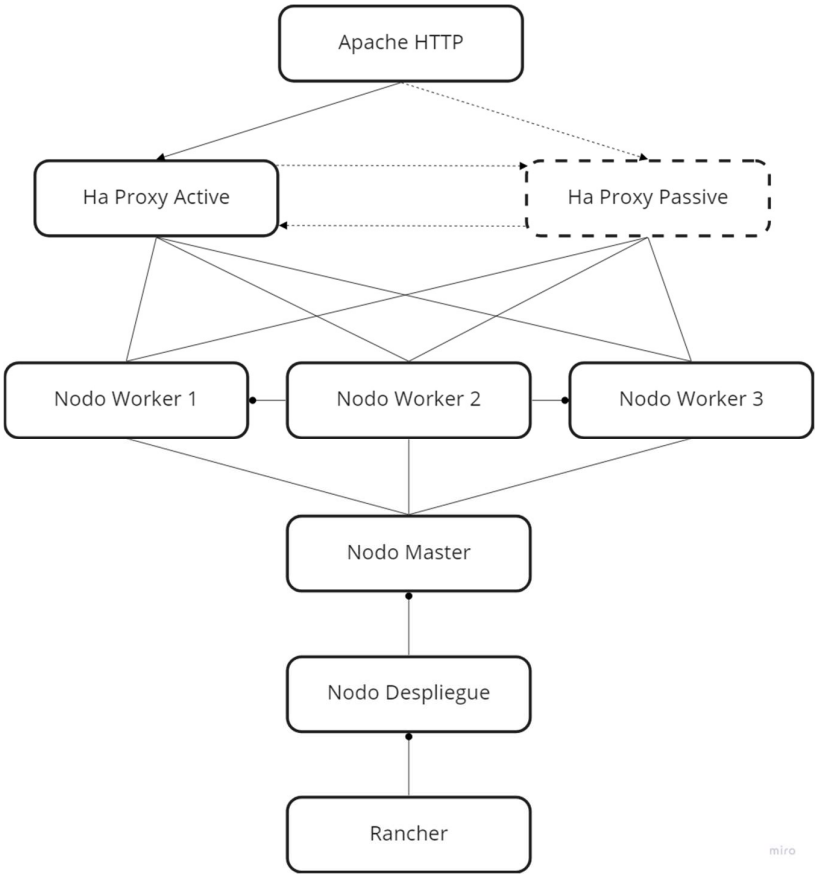

Figura 17.- Estructura General del Diseño.

4.2.2. Documentación y socialización del diseño.

En esta etapa se describe cada uno de los componentes que forman parte de la arquitectura del diseño.

- **Servidor Apache HTTP Server.**

Un servidor web es un servicio de red que sirve contenidos a un cliente a través de la web. Normalmente se trata de páginas web, pero también se puede servir cualquier otro documento. Los servidores web también se conocen como servidores http, ya que utilizan la dirección del protocolo de transporte de hipertexto http. El Apache http Server es un servidor web de código abierto desarrollado por la Apache Software Foundation.

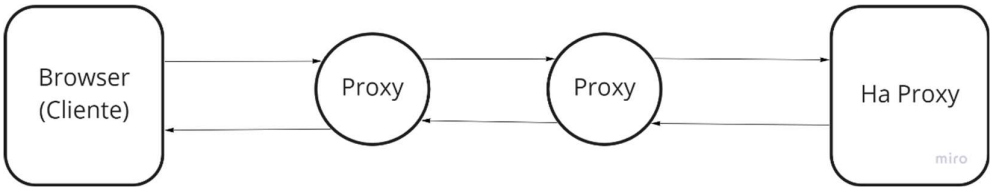

Figura 18.- Componentes Servidor Apache HTTP

El navegador envía la petición al servidor para obtener una respuesta.
El proxy es un componente del servidor web http que puede alterar la petición de acuerdo al requerimiento antes de pasar al ha proxy.
Los proxies manejan el almacenamiento en caché, el registro y el balanceo de carga.

- **Balanceador de Carga HA Proxy.**

"HaProxy es un proxy inverso gratuito, muy rápido y fiable que ofrece alta disponibilidad , equilibrio de carga y proxy para aplicaciones basadas en TCP y HTTP. Es particularmente adecuado para sitios web de muy alto tráfico". HAProxy El balanceador de carga TCP/HTTP confiable y de alto rendimiento. (2022, 1 diciembre). HAProxy Community Edition. Recuperado 27 de enero de 2023, de http://www.haproxy.org/#desc

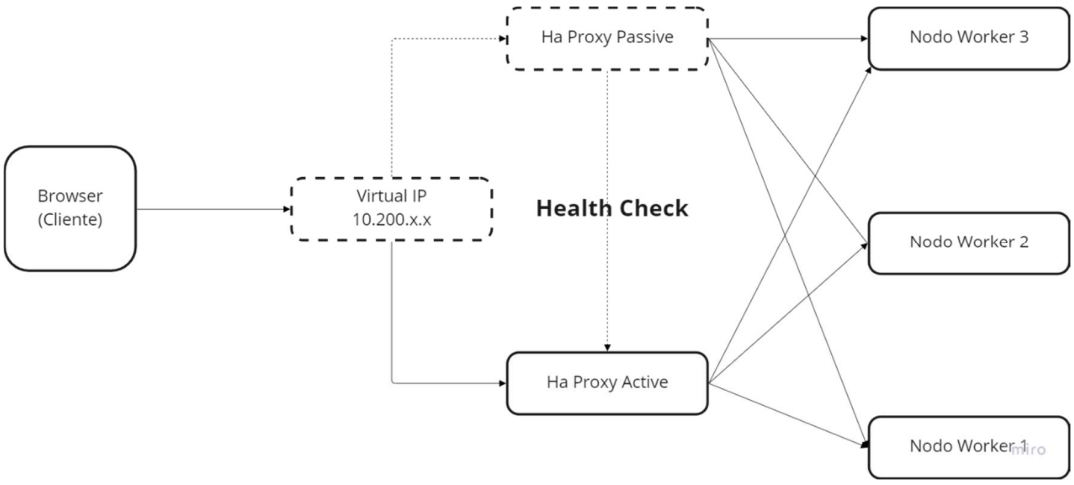

Figura 19.- Componentes HA Proxy.

Dentro de la implementación de HA Proxy se configura dos componentes que servirán para poder realizar el estado de comprobación del Clúster.

Corosync. - Es un sistema de comunicación grupal con características adicionales para implementar alta disponibilidad dentro de las aplicaciones.

Pacemaker. - Pacemaker es un administrador de recursos de clúster avanzado, escalable y de alta disponibilidad.

- **Nodos Master.**

  El nodo Master es el punto de entrada de todas las tareas administrativas.

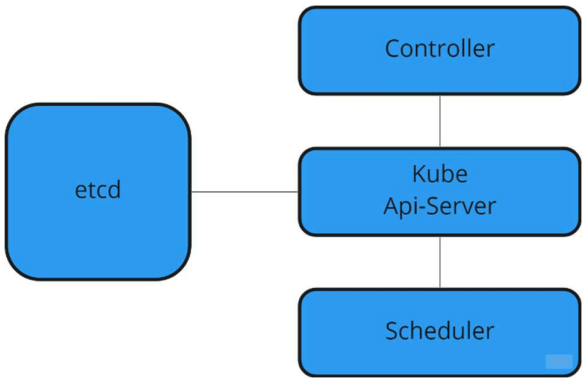

Figura 20.- Componentes Nodo Master

Servidor Api. - Realiza todas las tareas administrativas. Luego este las procesa y ejecuta, el estado actual se guarda en el etcd como un almacén de clave-valor.

Scheduler. - Es el encargado de programar los diferentes trabajos en los nodos workers, este considera los requisitos de QoS, la ubicación de los datos y otros parámetros similares. Luego programa el trabajo definido en términos de pods y servicios.

Controller. - El administrador del controlador tiene como misión regular el estado del clúster de Kubernetes. Compara el estado actual del objeto con el estado deseado. En caso de que los estados nos coincidan se toman medidas correctivas. Por lo tanto, el administrador del controlador se asegura de que su estado actual sea el mismo que el estado deseado.

Etcd. - Almacena el estado del clúster. Es un almacén de clave valor que también se utiliza para almacenar los detalles de la configuración, como las subredes y los mapas de configuración.

- **Nodos Workers.**

  Los nodos workers ejecutan las aplicaciones y están controlados por los nodos maestros. Los pods corren sobre estos nodos que tienen las herramientas necesarias para ejecutarlos y conectarlos.

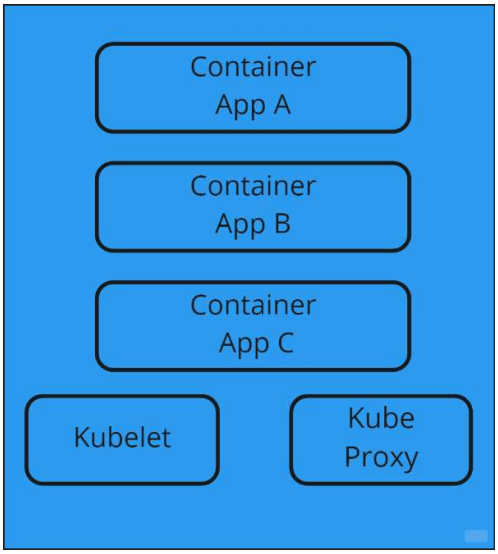

Figura 21.- Componentes Nodos Workers

Kubelet. - Es un agente que se ejecuta en cada nodo worker y se comunica con el nodo master. También comprueba que los contenedores donde se ejecutan los pods estén siempre sanos.

Kube-Proxy. - Es un proxy de red que se ejecuta en cada nodo worker. Escucha al servidor API para crear o eliminar puntos de servicio.

4.3. Construir el diseño que soporte la arquitectura propuesta.

En esta etapa se realiza la construcción del prototipo del sistema para el diseño propuesto. De tal manera que de forma práctica se pueda conocer si el diseño es capaz de cumplir con los requerimientos. Todos los componentes y las tecnologías usadas para el diseño están montadas sobre Linux, Centos 7.

El diseño está conformado por un servidor apache que servirá como proxy reverso para direccionar todas las solicitudes que ingresen al clúster. Él Ha Proxy se encargará de balancear la carga junto con los servicios de Pacemaker y Cronosync para mantener la alta disponibilidad del sistema por medio de una VIP. El clúster de kubernetes con sus nodos master y workers serán donde se alojarán todas las aplicaciones. Por último, un administrador del clúster que permita realizar los despliegues, administrar usuarios, permisos y ver el estado actual del clúster.

Configuración del servidor Apache.

La configuración del apache irá sobre un servidor con sistema operativo Centos 7. Para esto instalar el servicio con el siguiente comando ***yum install httpd -y***. Una vez ejecutado el comando se resolverán algunas dependencias como muestra la figura 22 y figura 23.

```
Dependencias resueltas

================================================================================
 Package           Arquitectura Versión                     Repositorio   Tamaño
================================================================================
Instalando:
 httpd             x86_64       2.4.6-97.el7.centos.5       updates       2.7 M
Instalando para las dependencias:
 apr               x86_64       1.4.8-7.el7                 base          104 k
 apr-util          x86_64       1.5.2-6.el7                 base           92 k
 httpd-tools       x86_64       2.4.6-97.el7.centos.5       updates        94 k
 mailcap           noarch       2.1.41-2.el7                base           31 k

Resumen de la transacción
================================================================================
Instalar  1 Paquete (+4 Paquetes dependientes)

Tamaño total de la descarga: 3.0 M
Tamaño instalado: 10 M
Downloading packages:
(1/5): apr-1.4.8-7.el7.x86_64.rpm                          | 104 kB   00:00
(2/5): apr-util-1.5.2-6.el7.x86_64.rpm                     |  92 kB   00:00
(3/5): httpd-tools-2.4.6-97.el7.centos.5.x86_64.rpm        |  94 kB   00:00
(4/5): mailcap-2.1.41-2.el7.noarch.rpm
(5/5): httpd-2.4.6-97.el7.centos.5.x86_64.rpm
```

Figura 22.- Dependencias a instalar.

```
Total
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Instalando    : apr-1.4.8-7.el7.x86_64
  Instalando    : apr-util-1.5.2-6.el7.x86_64
  Instalando    : httpd-tools-2.4.6-97.el7.centos.5.x86_64
  Instalando    : mailcap-2.1.41-2.el7.noarch
  Instalando    : httpd-2.4.6-97.el7.centos.5.x86_64
  Comprobando   : apr-1.4.8-7.el7.x86_64
  Comprobando   : mailcap-2.1.41-2.el7.noarch
  Comprobando   : httpd-tools-2.4.6-97.el7.centos.5.x86_64
  Comprobando   : apr-util-1.5.2-6.el7.x86_64
  Comprobando   : httpd-2.4.6-97.el7.centos.5.x86_64

Instalado:
  httpd.x86_64 0:2.4.6-97.el7.centos.5
```

Figura 23.- Confirmación de las dependencias a instalar.

Una vez descargadas las dependencias se procede a iniciar el servicio. Para ello se ejecuta el siguiente comando ***sudo systemctl start httpd***, como muestra la figura 24.

```
[root@localhost ~]# systemctl status httpd.service
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
   Active: active (running) since vie 2022-07-01 00:21:19 -05; 1s ago
     Docs: man:httpd(8)
           man:apachectl(8)
  Process: 21122 ExecStop=/bin/kill -WINCH ${MAINPID} (code=exited, status=0/SUCCESS)
 Main PID: 21164 (httpd)
   Status: "Processing requests..."
   CGroup: /system.slice/httpd.service
           ├─21164 /usr/sbin/httpd -DFOREGROUND
           ├─21165 /usr/sbin/httpd -DFOREGROUND
           ├─21166 /usr/sbin/httpd -DFOREGROUND
           ├─21167 /usr/sbin/httpd -DFOREGROUND
           ├─21168 /usr/sbin/httpd -DFOREGROUND
           └─21169 /usr/sbin/httpd -DFOREGROUND
```

Figura 24.- Se Inicia el servicio httpd.

Cuando finaliza la instalación del apache se genera el directorio /etc/httpd que se muestra en la figura 25, donde se encuentran los archivos que necesita el servicio para trabajar. A continuación, se destacan las configuraciones que se debe tener en cuenta.

/etc/httpd/

Nombre	Tamaño	Modificado	Permisos	Propiet...
..		6/7/2022 10:33:31	rwxr-xr-x	root
conf		30/6/2022 23:59:32	rwxr-xr-x	root
conf.d		1/7/2022 0:37:30	rwxr-xr-x	root
conf.modules.d		1/7/2022 0:24:30	rwxr-xr-x	root
logs		30/6/2022 23:59:32	rwxrwxrwx	root
modules		30/6/2022 23:59:32	rwxrwxrwx	root
run		30/6/2022 23:59:32	rwxrwxrwx	root

Figura 25.- Directorio configuración apache.

En el directorio /etc/httpd/conf se encuentra el archivo “httpd.conf”, en el que se encuentra la configuración del servidor apache, figura 26. Dentro del mismo se encuentra las siguientes líneas:

/etc/httpd/conf/

Nombre	Tamaño	Modificado	Permisos	Propiet...
..		30/6/2022 23:59:32	rwxr-xr-x	root
httpd.conf	12 KB	1/7/2022 0:17:36	rw-r--r--	root
magic	13 KB	24/3/2022 9:58:28	rw-r--r--	root

Figura 26.- Archivo de configuración.

Se indica hacia donde debe apuntar si se adiciona configuraciones a través de archivos .conf; los mismos deben estar en el directorio /etc/httpd/conf.d. Figura 27.

Load config files in the "/etc/httpd/conf.d" directory, if any.
IncludeOptional conf.d/*.conf

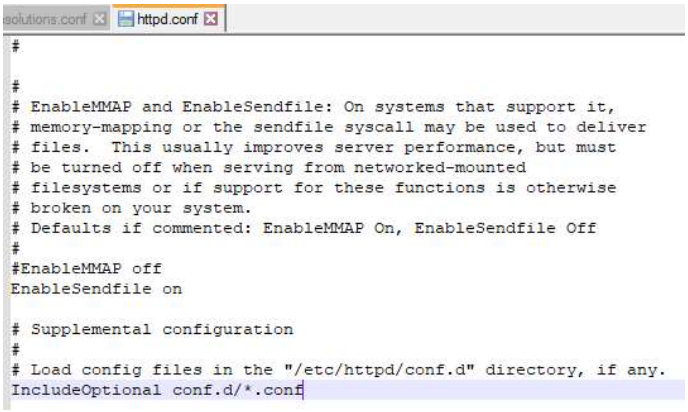

Figura 27.- Archivo de configuración.

En la figura 28 se ha configurado un archivo con el nombre gamasolutions.conf, el cual va a tener la configuración del dominio sobre el cual se va a desplegar el diseño. Adicional tendrá la configuración para el redireccionamiento de los contextos a usar dentro del diseño.

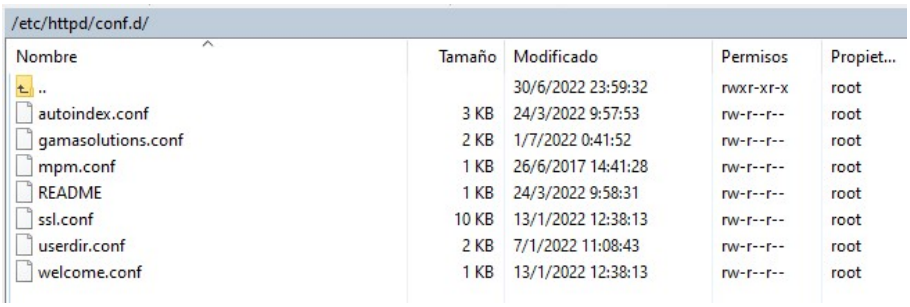

Figura 28.- Configuración archivo .conf

En la figura 29 se indica las configuraciones para habilitar el dominio sobre el cual se va a trabajar. Además, se establece la ruta donde va alojado los certificados de seguridad SSL. Estos certificados fueron auto firmados ya que para cuestiones de prueba es suficiente. Para ambientes de producción es necesario un certificado firmado por una entidad de confianza.


```
<VirtualHost *:443>

    ServerAlias gamasolutions.com
        Header edit Location ^http://gamasolutions.com/ https://gamasolutions.com/
    RewriteEngine on

    ProxyRequests Off

    <Proxy *>
            Require all granted
        </Proxy>

# DISABLE ALL CACHING WHILE DEVELOPING
  <FilesMatch "\.(html|htm|js|css|json)$">
#    FileETag None
    <IfModule mod_headers.c>
      Header unset ETag
      Header set Cache-Control "max-age=0, no-cache, no-store, must-revalidate"
      Header set Pragma "no-cache"
      Header set Note "CACHING IS DISABLED ON LOCALHOST"
      Header set Expires "Wed, 11 Jan 1984 05:00:00 GMT"
    </IfModule>
  </FilesMatch>


############################## CONTEXTOS ################################################



#########################################################################################################

    AllowEncodedSlashes On
    AllowEncodedSlashes NoDecode



        RedirectMatch ^/$ https://gamasolutions.com/

    SSLEngine on
#   SSLProxyEngine On
#   SSLProxyCheckPeerCN on
#   SSLProxyCheckPeerExpire on
    SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
    SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key
    #SSLCertificateChainFile /etc/httpd/ssl/chainCA.crt
    #Protocols h2 h2c http/1.1
    RequestHeader set X-Forwarded-Proto "https"
    RequestHeader set X-Forwarded-Port "443"
    ProxyPreserveHost On
    #SSLProxyCheckPeerCN off
    #SSLProxyCheckPeerExpire off
    #SSLProxyCheckPeerName off
    #SSLProxyVerify none
```

Figura 29.- Configuración del dominio.

En la figura 30 se puede ver que al apuntar al dominio configurado se tiene una respuesta del servidor web.

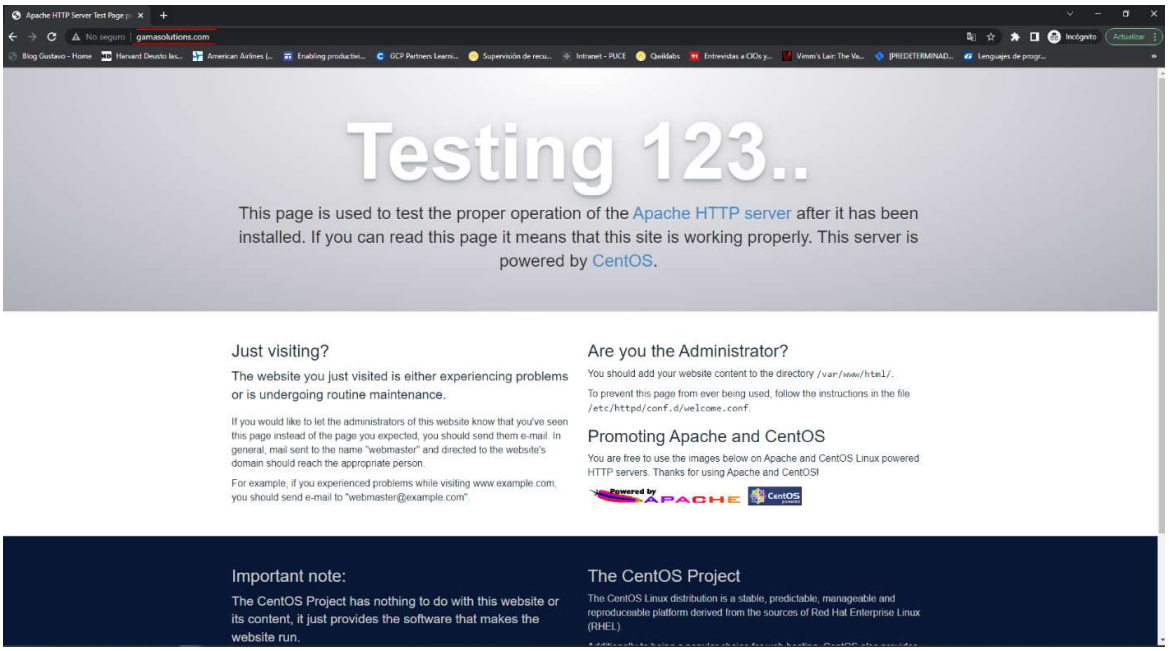

Figura 30.- Respuesta del servidor apache con el dominio configurado.

Configuración del Ha Proxy

Instalar gcc, pcre-static y pcre-devel con el comando.
yum -y install gcc pcre-static pcre-devel

```
[root@localhost ~]# yum -y install gcc pcre-static pcre-devel
Complementos cargados:fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirror.cedia.org.ec
 * extras: mirror.cedia.org.ec
 * updates: mirror.cedia.org.ec
Resolviendo dependencias
--> Ejecutando prueba de transacción
---> Paquete gcc.x86_64 0:4.8.5-44.el7 debe ser instalado
--> Procesando dependencias: cpp = 4.8.5-44.el7 para el paquete: gcc-4.8.5-44.el7.x86_64
--> Procesando dependencias: glibc-devel >= 2.2.90-12 para el paquete: gcc-4.8.5-44.el7.x86_64
--> Procesando dependencias: libmpfr.so.4()(64bit) para el paquete: gcc-4.8.5-44.el7.x86_64
--> Procesando dependencias: libmpc.so.3()(64bit) para el paquete: gcc-4.8.5-44.el7.x86_64
---> Paquete pcre-devel.x86_64 0:8.32-17.el7 debe ser instalado
---> Paquete pcre-static.x86_64 0:8.32-17.el7 debe ser instalado
--> Ejecutando prueba de transacción
---> Paquete cpp.x86_64 0:4.8.5-44.el7 debe ser instalado
---> Paquete glibc-devel.x86_64 0:2.17-326.el7_9 debe ser instalado
--> Procesando dependencias: glibc-headers = 2.17-326.el7_9 para el paquete: glibc-devel-2.17-326.el7_9.x86_64
--> Procesando dependencias: glibc-headers para el paquete: glibc-devel-2.17-326.el7_9.x86_64
---> Paquete libmpc.x86_64 0:1.0.1-3.el7 debe ser instalado
---> Paquete mpfr.x86_64 0:3.1.1-4.el7 debe ser instalado
--> Ejecutando prueba de transacción
---> Paquete glibc-headers.x86_64 0:2.17-326.el7_9 debe ser instalado
--> Procesando dependencias: kernel-headers >= 2.2.1 para el paquete: glibc-headers-2.17-326.el7_9.x86_64
--> Procesando dependencias: kernel-headers para el paquete: glibc-headers-2.17-326.el7_9.x86_64
--> Ejecutando prueba de transacción
---> Paquete kernel-headers.x86_64 0:3.10.0-1160.71.1.el7 debe ser instalado
--> Resolución de dependencias finalizada
```

Figura 31.- Paquetes haproxy

```
Dependencias resueltas

================================================================================
 Package                 Arquitectura            Versión
================================================================================
Instalando:
 gcc                     x86_64                  4.8.5-44.el7
 pcre-devel              x86_64                  8.32-17.el7
 pcre-static             x86_64                  8.32-17.el7
Instalando para las dependencias:
 cpp                     x86_64                  4.8.5-44.el7
 glibc-devel             x86_64                  2.17-326.el7_9
 glibc-headers           x86_64                  2.17-326.el7_9
 kernel-headers          x86_64                  3.10.0-1160.71.1.el7
 libmpc                  x86_64                  1.0.1-3.el7
 mpfr                    x86_64                  3.1.1-4.el7

Resumen de la transacción
================================================================================
Instalar  3 Paquetes (+6 Paquetes dependientes)

Tamaño total de la descarga: 34 M
Tamaño instalado: 63 M
Downloading packages:
(1/9): glibc-headers-2.17-326.el7_9.x86_64.rpm
(2/9): libmpc-1.0.1-3.el7.x86_64.rpm
(3/9): glibc-devel-2.17-326.el7_9.x86_64.rpm
(4/9): mpfr-3.1.1-4.el7.x86_64.rpm
(5/9): pcre-static-8.32-17.el7.x86_64.rpm
(6/9): pcre-devel-8.32-17.el7.x86_64.rpm
(7/9): gcc-4.8.5-44.el7.x86_64.rpm
(8/9): cpp-4.8.5-44.el7.x86_64.rpm
(9/9): kernel-headers-3.10.0-1160.71.1.el7.x86_64.rpm
--------------------------------------------------------------------------------
```

Figura 32.- Dependencias

Copiar el instalador del HA Proxy en cualquier ruta, figura 33.

Figura 33.- Copia del instalador

Desempaquetar el archivo con el siguiente comando:
tar xzvf haproxy-2.6.1.tar.gz

```
[root@localhost ~]# tar xzvf haproxy-2.6.1.tar.gz
haproxy-2.6.1/
haproxy-2.6.1/.cirrus.yml
haproxy-2.6.1/.gitattributes
haproxy-2.6.1/.github/
haproxy-2.6.1/.github/ISSUE_TEMPLATE/
haproxy-2.6.1/.github/ISSUE_TEMPLATE/Bug.yml
haproxy-2.6.1/.github/ISSUE_TEMPLATE/Code-Report.yml
haproxy-2.6.1/.github/ISSUE_TEMPLATE/Feature.yml
haproxy-2.6.1/.github/ISSUE_TEMPLATE/config.yml
haproxy-2.6.1/.github/errorfile
haproxy-2.6.1/.github/h2spec.config
haproxy-2.6.1/.github/matrix.py
haproxy-2.6.1/.github/vtest.json
haproxy-2.6.1/.github/workflows/
haproxy-2.6.1/.github/workflows/codespell.yml
haproxy-2.6.1/.github/workflows/compliance.yml
haproxy-2.6.1/.github/workflows/contrib.yml
haproxy-2.6.1/.github/workflows/coverity.yml
haproxy-2.6.1/.github/workflows/musl.yml
haproxy-2.6.1/.github/workflows/openssl-nodeprecated.yml
haproxy-2.6.1/.github/workflows/vtest.yml
haproxy-2.6.1/.github/workflows/windows.yml
haproxy-2.6.1/.gitignore
haproxy-2.6.1/.mailmap
haproxy-2.6.1/.travis.yml
```

Figura 34.- Se desempaqueta el instalador

Ingresar a la carpeta haproxy-2.6.1 y ejecutar el siguiente comando.
make TARGET=linux-glibc

```
[root@localhost haproxy-2.6.1]# make TARGET=linux-glibc
  CC      src/slz.o
  CC      src/ev_poll.o
  CC      src/ev_epoll.o
  CC      src/cpuset.o
  CC      src/namespace.o
  CC      src/mux_h2.o
  CC      src/mux_fcgi.o
  CC      src/mux_h1.o
  CC      src/tcpcheck.o
  CC      src/stream.o
  CC      src/stats.o
  CC      src/http_ana.o
  CC      src/server.o
  CC      src/stick_table.o
  CC      src/sample.o
  CC      src/flt_spoe.o
  CC      src/tools.o
  CC      src/log.o
  CC      src/cfgparse.o
  CC      src/peers.o
  CC      src/backend.o
  CC      src/resolvers.o
  CC      src/cli.o
  CC      src/connection.o
  CC      src/proxy.o
  CC      src/http_htx.o
  CC      src/cfgparse-listen.o
  CC      src/pattern.o
  CC      src/check.o
  CC      src/haproxy.o
  CC      src/cache.o
  CC      src/stconn.o
```

Figura 35.- Se compila el programa para Linux

Finalmente instalar HA Proxy con el siguiente comando:
make install

```
[root@localhost haproxy-2.6.1]# make install
«haproxy» -> «/usr/local/sbin/haproxy»
«doc/haproxy.1» -> «/usr/local/share/man/man1/haproxy.1»
install: creando el directorio «/usr/local/doc»
install: creando el directorio «/usr/local/doc/haproxy»
«doc/configuration.txt» -> «/usr/local/doc/haproxy/configuration.txt»
«doc/management.txt» -> «/usr/local/doc/haproxy/management.txt»
«doc/seamless_reload.txt» -> «/usr/local/doc/haproxy/seamless_reload.txt»
«doc/architecture.txt» -> «/usr/local/doc/haproxy/architecture.txt»
«doc/peers-v2.0.txt» -> «/usr/local/doc/haproxy/peers-v2.0.txt»
«doc/regression-testing.txt» -> «/usr/local/doc/haproxy/regression-testing.txt»
«doc/cookie-options.txt» -> «/usr/local/doc/haproxy/cookie-options.txt»
«doc/lua.txt» -> «/usr/local/doc/haproxy/lua.txt»
«doc/WURFL-device-detection.txt» -> «/usr/local/doc/haproxy/WURFL-device-detection.txt»
«doc/proxy-protocol.txt» -> «/usr/local/doc/haproxy/proxy-protocol.txt»
«doc/linux-syn-cookies.txt» -> «/usr/local/doc/haproxy/linux-syn-cookies.txt»
«doc/SOCKS4.protocol.txt» -> «/usr/local/doc/haproxy/SOCKS4.protocol.txt»
«doc/network-namespaces.txt» -> «/usr/local/doc/haproxy/network-namespaces.txt»
«doc/DeviceAtlas-device-detection.txt» -> «/usr/local/doc/haproxy/DeviceAtlas-device-detection.txt»
«doc/51Degrees-device-detection.txt» -> «/usr/local/doc/haproxy/51Degrees-device-detection.txt»
«doc/netscaler-client-ip-insertion-protocol.txt» -> «/usr/local/doc/haproxy/netscaler-client-ip-insertion-protocol.txt»
«doc/peers.txt» -> «/usr/local/doc/haproxy/peers.txt»
«doc/close-options.txt» -> «/usr/local/doc/haproxy/close-options.txt»
«doc/SPOE.txt» -> «/usr/local/doc/haproxy/SPOE.txt»
«doc/intro.txt» -> «/usr/local/doc/haproxy/intro.txt»
[root@localhost haproxy-2.6.1]#
[root@localhost haproxy-2.6.1]# mkdir -p /etc/haproxy
[root@localhost haproxy-2.6.1]# mkdir -p /var/lib/haproxy
[root@localhost haproxy-2.6.1]# touch /var/lib/haproxy/stats
[root@localhost haproxy-2.6.1]#
[root@localhost haproxy-2.6.1]# ln -s /usr/local/sbin/haproxy /usr/sbin/haproxy
[root@localhost haproxy-2.6.1]# ls -l
total 13940
```

Figura 36.- Instalación Haproxy.

Crear el directorio /etc/haproxy con el comando:
mkdir -p /etc/haproxy

Crear el directorio /var/lib/haproxy
mkdir -p /var/lib/haproxy

Crear el archivo stats en el directorio /var/lib/haproxy/
touch /var/lib/haproxy/stats

Crear un enlace simbólico al directorio
ln -s /usr/local/sbin/haproxy /usr/sbin/haproxy

```
[root@localhost haproxy-2.6.1]# mkdir -p /etc/haproxy
[root@localhost haproxy-2.6.1]# mkdir -p /var/lib/haproxy
[root@localhost haproxy-2.6.1]# touch /var/lib/haproxy/stats
[root@localhost haproxy-2.6.1]#
[root@localhost haproxy-2.6.1]# ln -s /usr/local/sbin/haproxy /usr/sbin/haproxy
```

Figura 37.- Creación de carpetas.

Colocarse sobre el directorio desempaquetado y copiar la carpeta examples al directorio /etc/init.d/haproxy.

cp examples/haproxy.init /etc/init.d/haproxy

Colocar permisos 755 al directorio /etc/init.d/haproxy
chmod 755 /etc/init.d/haproxy

Reiniciar el demonio con el siguiente comando
systemctl daemon-reload

Crear el usuario haproxy
useradd -r haproxy

Permitir el servicio http en el firewalld-cmd
firewall-cmd --permanent --zone=public --add-service=http

Reiniciar el firewall
firewall-cmd –reload

Crear el archivo /etc/haproxy/haproxy.cfg
vi /etc/haproxy/haproxy.cfg

```
[root@localhost haproxy-2.6.1]# cp examples/haproxy.init /etc/init.d/haproxy
[root@localhost haproxy-2.6.1]# chmod 755 /etc/init.d/haproxy
[root@localhost haproxy-2.6.1]# systemctl daemon-reload
[root@localhost haproxy-2.6.1]#
[root@localhost haproxy-2.6.1]# useradd -r haproxy
[root@localhost haproxy-2.6.1]# firewall-cmd --permanent --zone=public --add-service=http
success
[root@localhost haproxy-2.6.1]# firewall-cmd --reload
success
[root@localhost haproxy-2.6.1]# vi /etc/haproxy/haproxy.cfg
```

Figura 38.- Se crea el usuario y se da los permisos necesarios.

Configurar el archivo de la siguiente manera. Figura 39.

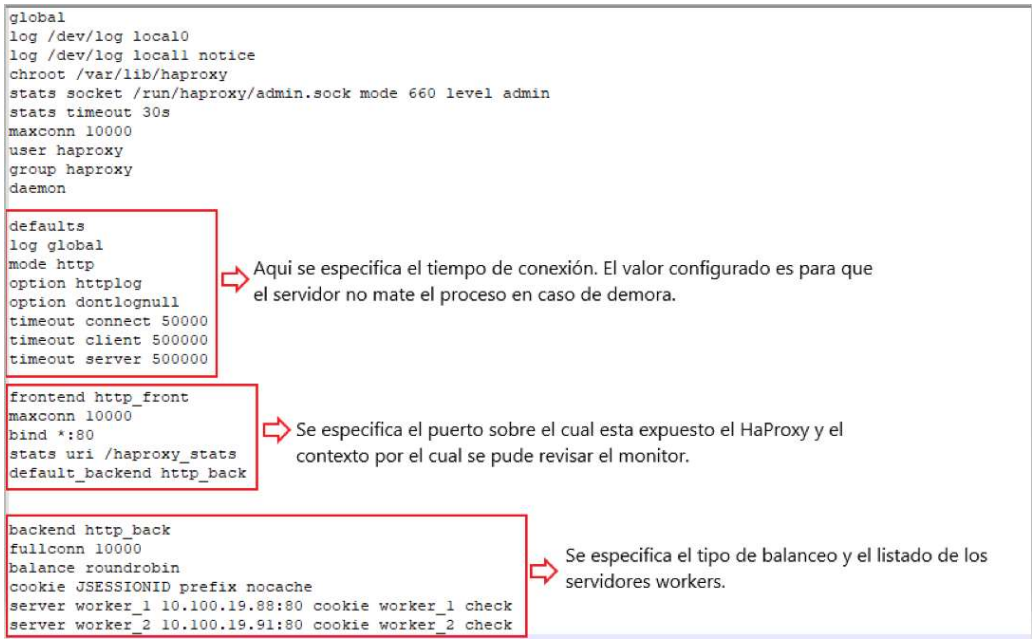

Figura 39.- Configuraciones Haproxy.

En el archivo /etc/init.d/haproxy incluir la siguiente línea al inicio del start del servicio.
mkdir -p /run/haproxy

```
start() {
  mkdir -p /run/haproxy
  quiet_check
  if [ $? -ne 0 ]; then
    echo "Errors found in configuration file, check it with '$BASENAME check'."
    return 1
  fi

  echo -n "Starting $BASENAME: "
  daemon $BIN -D -f $CFG -p $PIDFILE
  RETVAL=$?
  echo
  [ $RETVAL -eq 0 ] && touch $LOCKFILE
  return $RETVAL
}

stop() {
  echo -n "Shutting down $BASENAME: "
  killproc $BASENAME -USR1
  RETVAL=$?
  echo
  [ $RETVAL -eq 0 ] && rm -f $LOCKFILE
  [ $RETVAL -eq 0 ] && rm -f $PIDFILE
  return $RETVAL
}
```

Figura 40.- Configuración carpeta de inicio.

Reiniciar el Servicio.
systemctl daemon-reload

Reiniciar el Haproxy.
systemctl restart haproxy

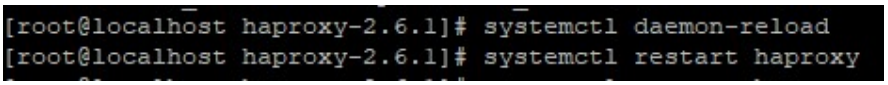

Figura 41.- Se reinicia los servicios.

Ingresar por el navegador con la ip y el contexto señalado en la figura 42 y deberá aparecer la información de los nodos workers corriendo sobre el Haproxy.

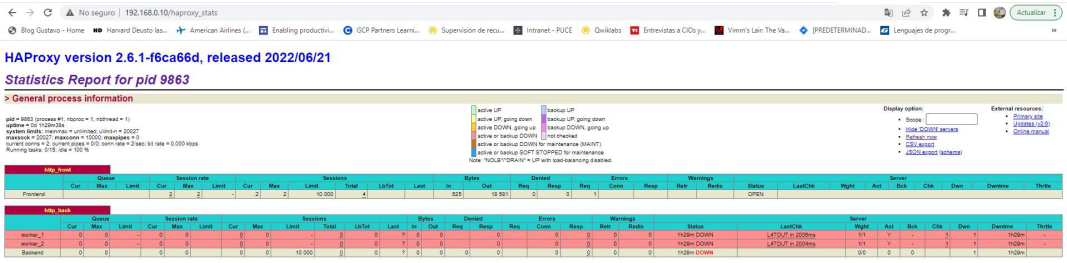

Figura 42.- Comprobación

Configuración Pacemaker.

Instalar los paquetes Pacemaker pcs, esto se lo realiza en los dos servidores. Figura 43.

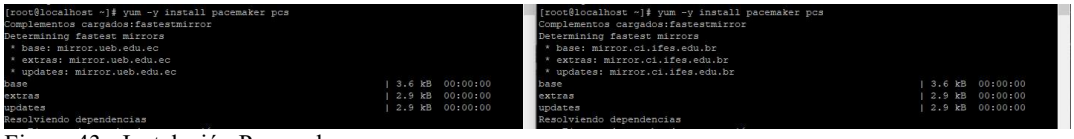

Figura 43.- Instalación Pacemaker

Iniciar el servicio. Figura 44.

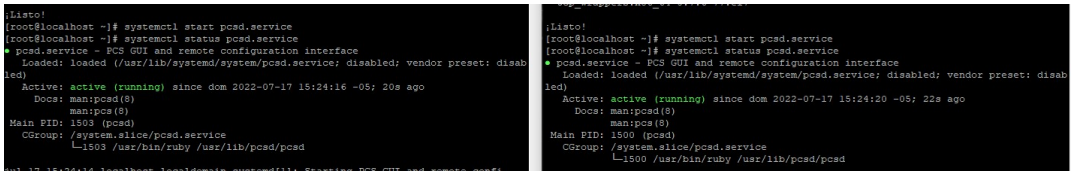

Figura 44.- Se inicia el servicio.

Cambiar la contraseña del usuario hacluster. Figura 45.

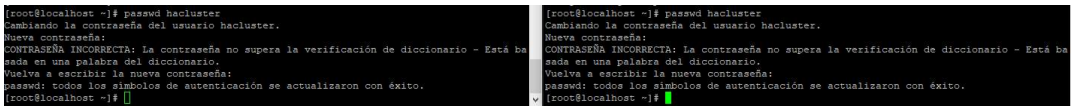

Figura 45.- Cambio de contraseña

Habilitar el servicio high-availability en el firewall. Figura 46.

Figura 46.- Se habilita el servicio.

Habilitar el servicio corosync.service. Figura 47.

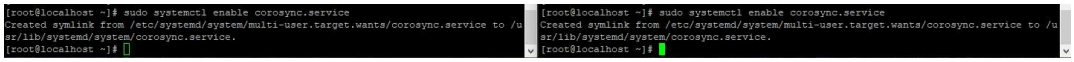

Figura 47.- Se habilita el servicio

Habilitar el servicio de Pacemaker. Figura 48.

Figura 48.- Se habilita el servicio.

Autorizar los dos nodos. Figura 49.

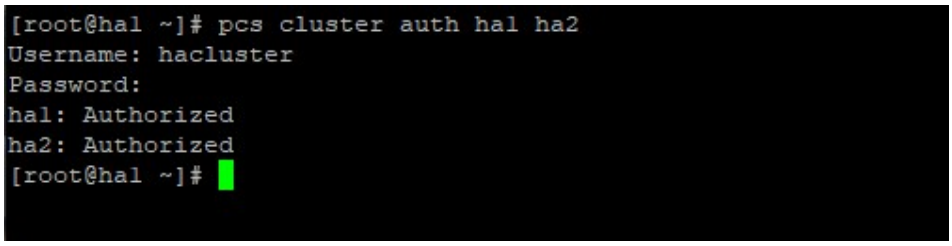

Figura 49.- Se autoriza los nodos.

Armado del clúster. Figura 50.

```
[root@ha1 ~]# pcs cluster setup --name hacluster ha1 ha2
Destroying cluster on nodes: ha1, ha2...
ha1: Stopping Cluster (pacemaker)...
ha2: Stopping Cluster (pacemaker)...
ha1: Successfully destroyed cluster
ha2: Successfully destroyed cluster

Sending 'pacemaker_remote authkey' to 'ha1', 'ha2'
ha1: successful distribution of the file 'pacemaker_remote authkey'
ha2: successful distribution of the file 'pacemaker_remote authkey'
Sending cluster config files to the nodes...
ha1: Succeeded
ha2: Succeeded

Synchronizing pcsd certificates on nodes ha1, ha2...
ha1: Success
ha2: Success
Restarting pcsd on the nodes in order to reload the certificates...
ha1: Success
ha2: Success
[root@ha1 ~]#
```

Figura 50.- Armado del clúster

Levantar el clúster. Figura 51.

```
[root@ha1 ~]# pcs cluster start --all
ha1: Starting Cluster (corosync)...
ha2: Starting Cluster (corosync)...
ha1: Starting Cluster (pacemaker)...
ha2: Starting Cluster (pacemaker)...
[root@ha1 ~]#
```

Figura 51.- Inicio del clúster.

Validar el estado del clúster. Figura 52.

```
[root@ha1 ~]# pcs status
Cluster name: hacluster

WARNINGS:
No stonith devices and stonith-enabled is not false

Stack: corosync
Current DC: ha1 (version 1.1.23-1.el7_9.1-9acf116022) - partition with quorum
Last updated: Wed Jul 20 18:04:23 2022
Last change: Wed Jul 20 18:03:22 2022 by hacluster via crmd on ha1

2 nodes configured
0 resource instances configured

Online: [ ha1 ha2 ]

No resources


Daemon Status:
  corosync: active/disabled
  pacemaker: active/disabled
  pcsd: active/enabled
```

Figura 52.- Estado del clúster.

Crear el recurso de la VIP con su respectiva mascara y el tiempo de intervalo que va a monitorear el servicio. Figura 53.

```
[root@ha1 ~]# pcs resource create Cluster_VIP ocf:heartbeat:IPaddr2 ip=192.168.0.12 cidr_netmask=24 op monitor interval=10s
[root@ha1 ~]# pcs status
Cluster name: hacluster
Stack: corosync
Current DC: ha1 (version 1.1.23-1.el7_9.1-9acf116022) - partition with quorum
Last updated: Wed Jul 20 18:20:33 2022
Last change: Wed Jul 20 18:20:27 2022 by root via cibadmin on ha1

2 nodes configured
1 resource instance configured

Online: [ ha1 ha2 ]

Full list of resources:

 Cluster_VIP    (ocf::heartbeat:IPaddr2):       Started ha1

Daemon Status:
  corosync: active/disabled
  pacemaker: active/disabled
  pcsd: active/enabled
```

Figura 53.- Configuración de la VIP

Crear el servicio el cual va a monitorear el Pacemaker. En este caso el servicio es el haproxy. Figura 54.

```
[root@ha1 ~]# pcs resource create HAproxy systemd:haproxy op monitor interval=10s
[root@ha1 ~]#
[root@ha1 ~]#
[root@ha1 ~]#
[root@ha1 ~]#
[root@ha1 ~]# pcs status
Cluster name: hacluster
Stack: corosync
Current DC: ha1 (version 1.1.23-1.el7_9.1-9acf116022) - partition with quorum
Last updated: Wed Jul 20 18:24:41 2022
Last change: Wed Jul 20 18:24:25 2022 by root via cibadmin on ha1

2 nodes configured
2 resource instances configured

Online: [ ha1 ha2 ]

Full list of resources:

 Cluster_VIP    (ocf::heartbeat:IPaddr2):       Started ha1
 HAproxy        (systemd:haproxy):      Started ha2

Daemon Status:
  corosync: active/disabled
  pacemaker: active/disabled
  pcsd: active/enabled
```

Figura 54.- Creación del servicio que monitoria Pacemaker.

En la figura anterior se puede ver que el servicio de la VIP se ejecuta en el server HA1, mientras que el servicio del haproxy se ejecuta en el server HA2. Esto está mal, ya que los dos servicios deben ejecutarse en un mismo equipo. Para esto se ejecuta el siguiente comando. Figura 55.

```
[root@ha1 ~]# pcs constraint colocation add HAproxy Cluster_VIP INFINITY
[root@ha1 ~]# pcs status
Cluster name: hacluster
Stack: corosync
Current DC: ha1 (version 1.1.23-1.el7_9.1-9acf116022) - partition with quorum
Last updated: Wed Jul 20 18:32:07 2022
Last change: Wed Jul 20 18:31:59 2022 by root via cibadmin on ha1

2 nodes configured
2 resource instances configured

Online: [ ha1 ha2 ]

Full list of resources:

 Cluster_VIP    (ocf::heartbeat:IPaddr2):       Started ha1
 HAproxy        (systemd:haproxy):      Started ha1

Daemon Status:
  corosync: active/disabled
  pacemaker: active/disabled
  pcsd: active/enabled
```

Figura 55.- Activar los dos servicios en el mismo equipo.

La VIP fue creada en el servidor HA1. Figura 56.

```
[root@ha1 ~]# ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:d9:6b:42 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.10/24 brd 192.168.0.255 scope global noprefixroute dynamic enp0s3
       valid_lft 2873sec preferred_lft 2873sec
    inet 192.168.0.12/24 brd 192.168.0.255 scope global secondary enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fed9:6b42/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figura 56.- Ubicación VIP.

Si existe algún evento sobre el servidor HA1 el servicio de Pacemaker enviará la VIP al otro clúster, en este caso el HA2. Figura 57.

```
C:\Users\Gus>ping ha1 -t

Haciendo ping a ha1 [192.168.0.10] con 32 bytes de datos:
Respuesta desde 192.168.0.10: bytes=32 tiempo<1m TTL=64
Respuesta desde 192.168.0.10: bytes=32 tiempo<1m TTL=64
Respuesta desde 192.168.0.10: bytes=32 tiempo<1m TTL=64
Respuesta desde 192.168.0.10: bytes=32 tiempo=1ms TTL=64
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
```

Figura 57.- Shutdown HA1.

La VIP ha saltado al equipo HA2 que forma parte del clúster y que se encuentra activo. Figura 58.

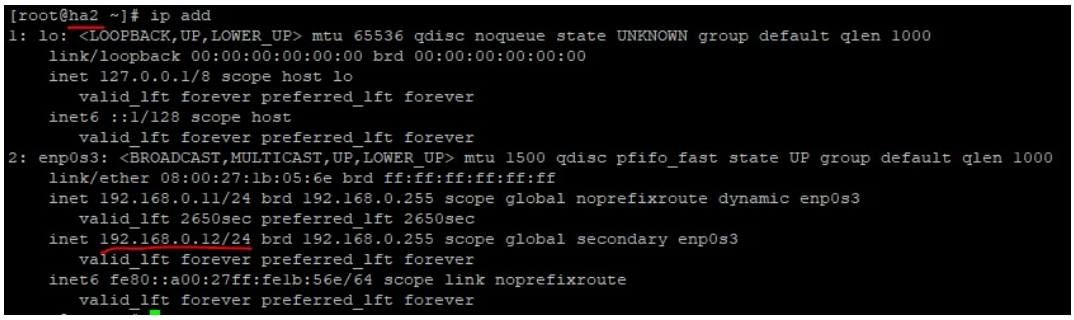

Figura 58.- Salto de la VIP al clúster activo.

El servicio del haproxy atado a la VIP. Figura 59.

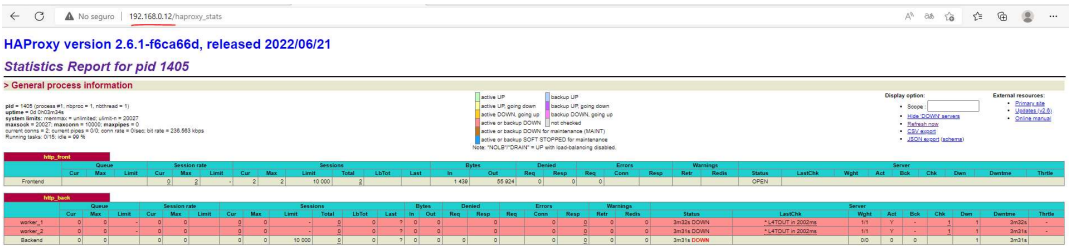

Figura 59. Servicio del haproxy atado a la VIP.

Instalación Clúster RKE.

El clúster tiene un nodo despliegue, un nodo master y un nodo worker. La instalación inicial para los tres equipos en conjunto son los siguientes.

La instalación como fue descrita en el diseño está montada sobre Centos 7.

Actualizar todos los componentes de Centos 7.

- ***yum update -y***

Instalar Docker para este caso se utilizará la versión 19.03.9. Esta versión es la recomendada por rancher para su despliegue.

- ***sudo yum install -y yum-utils***

- ***sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo***

- **sudo yum install docker-ce-19.03.9 docker-ce-cli-19.03.9 containerd.io docker-compose-plugin**

Habilitar el tráfico IPV4.

cat <<EOF>> /etc/sysctl.conf
net.bridge.bridge-nf-call-iptables=1
EOF

Abrir los siguientes puertos en el firewall.

- ***firewall-cmd --permanent --zone=public --add-port=80/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=443/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=22/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=2376/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=6443/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=2379/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=2380/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=9099/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=10250/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=10252/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=10254/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=10256/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=30000-32767/tcp***
- ***firewall-cmd --permanent --zone=public --add-port=30000-32767/udp***
- ***firewall-cmd --permanent --zone=public --add-port=8472/udp***
- ***firewall-cmd --permanent --zone=public --add-port=4789/udp***
- ***firewall-cmd --permanent --zone=public --add-port=9796/udp***

Crear el usuario k8s y asignar al grupo k8s. Este usuario permitirá la conexión entre los tres equipos que forman el clúster.

- useradd k8s && echo "Password01" | passwd --stdin k8s
- echo "k8s ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/k8s
- chmod 0440 /etc/sudoers.d/k8s
- usermod -aG docker k8s

La siguiente configuración se tiene que realizar únicamente en el nodo de despliegue.

Crear la llave ssh en el nodo de despliegue.

- ***ssh-keygen -t rsa***

Copiar la llave al resto de los nodos, worker y master.

- ***ssh-copy-id -i .ssh/id_rsa.pub k8s@192.168.0.13***
- ***ssh-copy-id -i .ssh/id_rsa.pub k8s@192.168.0.14***

Instalar RKE en el nodo de despliegue.

- ***sudo mv rke_linux-amd64 /usr/local/bin/rke***
- ***sudo chmod +x /usr/local/bin/rke***
- ***rke --version***

Instalar Helm en el nodo de despliegue.

- ***tar -zxvf helm-v3.1.2-linux-amd64.tar.gz***
- ***sudo mv linux-amd64/helm /usr/local/bin/helm***
- ***helm help***

Instalar Kubectl en el nodo de despliegue.

- ***curl -LO*** https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
- ***chmod +x kubectl***
- ***mkdir -p ~/.local/bin***
- ***mv ./kubectl ~/.local/bin/kubectl***

Crear el archivo con la configuración para el despliegue del clúster.

- ***nano cluster.yml***

Estructura del archivo. Aquí se especifica el rol de cada equipo.

- ***cluster_name: k8s***
  nodes:
  - address: 192.168.0.13
  user: k8s
  role:
  - controlplane
  - etcd
  - address: 192.168.0.14
  user: k8s
  role:
  - worker

El siguiente comando levanta todo el clúster con la configuración especificada en el archivo *cluster.yml.*

- ***rke up --config ./cluster.yml***

Aquí se muestra la secuencia de comandos que se corrió sobre el nodo de despliegue. Figura 60.

```
  6  sudo yum update -y
  7  sudo yum install docker-ce-19.03.9 docker-ce-cli-19.03.9 containerd.io docker-compose-plugin
  8  sudo yum install -y yum-utils
  9  sudo yum-config-manager     --add-repo     https://download.docker.com/linux/centos/docker-ce.repo
 10  sudo yum install docker-ce-19.03.9 docker-ce-cli-19.03.9 containerd.io docker-compose-plugin
 11  cat <<EOF>> /etc/sysctl.conf
 12  sysctl -p
 13  firewall-cmd --permanent --zone=public --add-port=80/tcp
 14  firewall-cmd --permanent --zone=public --add-port=443/tcp
 15  firewall-cmd --permanent --zone=public --add-port=22/tcp
 16  firewall-cmd --permanent --zone=public --add-port=2376/tcp
 17  firewall-cmd --permanent --zone=public --add-port=6443/tcp
 18  firewall-cmd --permanent --zone=public --add-port=2379/tcp
 19  firewall-cmd --permanent --zone=public --add-port=2380/tcp
 20  firewall-cmd --permanent --zone=public --add-port=9099/tcp
 21  firewall-cmd --permanent --zone=public --add-port=10250/tcp
 22  firewall-cmd --permanent --zone=public --add-port=10252/tcp
 23  firewall-cmd --permanent --zone=public --add-port=10254/tcp
 24  firewall-cmd --permanent --zone=public --add-port=10256/tcp
 25  firewall-cmd --permanent --zone=public --add-port=30000-32767/tcp
 26  firewall-cmd --permanent --zone=public --add-port=30000-32767/udp
 27  firewall-cmd --permanent --zone=public --add-port=8472/udp
 28  firewall-cmd --permanent --zone=public --add-port=4789/udp
 29  firewall-cmd --permanent --zone=public --add-port=9796/udp
 30  systemctl restart firewalld
 31  useradd k8s && echo "Password01" | passwd --stdin k8s
 32  echo "k8s ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/k8s
 33  chmod 0440 /etc/sudoers.d/k8s
 34  usermod -aG docker k8s
 35  sudo systemctl status docker
 36  sudo systemctl start docker
 37  sudo systemctl status docker
 38  sudo systemctl enable docker
 39  sudo systemctl status docker
 40  docker ps
 41  shutdown -h now
 42  ssh-keygen -t rsa
 43  ssh-copy-id -i .ssh/id_rsa.pub k8s@192.168.0.13
 44  ssh-copy-id -i .ssh/id_rsa.pub k8s@192.168.0.14
 45  ssh 'k8s@192.168.0.14'
 46  ssh 'k8s@192.168.0.13'
```

Figura 60.- Preparar el nodo de despliegue.

Levantar un docker para la instalación de Rancher. Esto permitirá administrar el clúster formado en el paso anterior. Se corre sobre el nodo de despliegue. Figura 61.

***docker run -d --restart=unless-stopped --privileged --name rancher ***
***-p 80:80 -p 443:443 ***
***-e NO_PROXY="localhost,127.0.0.1,0.0.0.0,192.0.0.0/8" ***
***-v /var/docker/rancher:/var/lib/rancher ***
***-v /var/docker/certs/cert.pem:/etc/rancher/ssl/cert.pem ***
***-v /var/docker/certs/key.pem:/etc/rancher/ssl/key.pem ***
***-v /var/docker/certs/ca.pem:/etc/rancher/ssl/cacerts.pem ***
rancher/rancher:v2.5.9-rc3

```
[root@localhost ~]#
[root@localhost ~]# docker run -d --restart=unless-stopped --privileged --name rancher \
>            -p 80:80 -p 443:443 \
>            -e NO_PROXY="localhost,127.0.0.1,0.0.0.0,192.0.0.0/8" \
>            -v /var/docker/rancher:/var/lib/rancher \
>            -v /var/docker/certs/cert.pem:/etc/rancher/ssl/cert.pem \
>            -v /var/docker/certs/key.pem:/etc/rancher/ssl/key.pem \
>            -v /var/docker/certs/ca.pem:/etc/rancher/ssl/cacerts.pem \
>            rancher/rancher:v2.5.9-rc3
```

Figura 61.- Se corre el comando.

Validar que el servicio este levantado con el siguiente comando. Figura 62.

- ***Docker ps.***

```
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE                      COMMAND             CREATED             STATUS              PORTS                                      NAMES
5bc80bb78f04        rancher/rancher:v2.5.9-rc3   "entrypoint.sh"     2 minutes ago       Up 2 minutes        0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp   rancher
[root@localhost ~]#
[root@localhost ~]#
```

Figura 62. Validar el servicio.

Validar por medio de la web con la ip del nodo de despliegue. Figura 63.

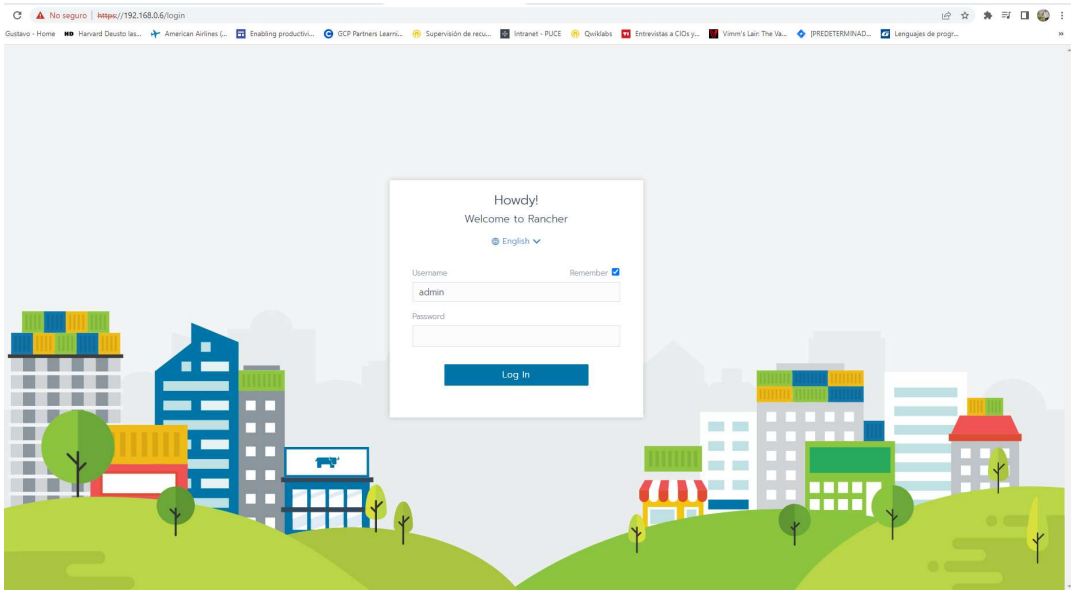

Figura 63.- Servicio de Rancher desplegado.

4.4. Evaluar el diseño de la solución propuesta para el despliegue.

Para validar el funcionamiento del diseño es necesario realizar una serie de pruebas. Estas pruebas se dividen en 3 partes:

- **Funcionalidad.**
- **Carga**.
- **Resistencia**.

Prueba de Funcionalidad.

La prueba de funcionalidad se la realizará ingresando a Rancher y desplegando una aplicación. Para esta prueba se va a desplegar keycloak, por cuestiones de prueba en la funcionalidad del diseño no se realizará la configuración de la herramienta como tal, simplemente que el diseño cumpla con la función de la prueba base para poder validar su implementación.

En este caso se lanzará la herramienta desde la consola del Rancher. Figura 64.

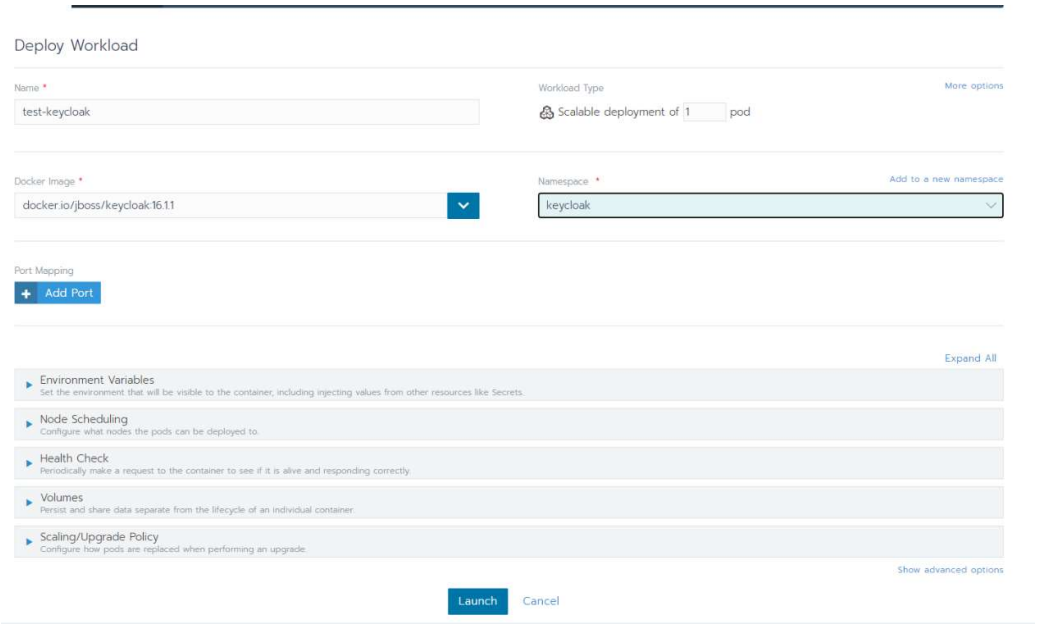

Figura 64.- Deployment Keycloak

Como se puede observar en la imagen el despliegue fue exitoso, asignando el pod en uno de sus nodos. Figura 65.

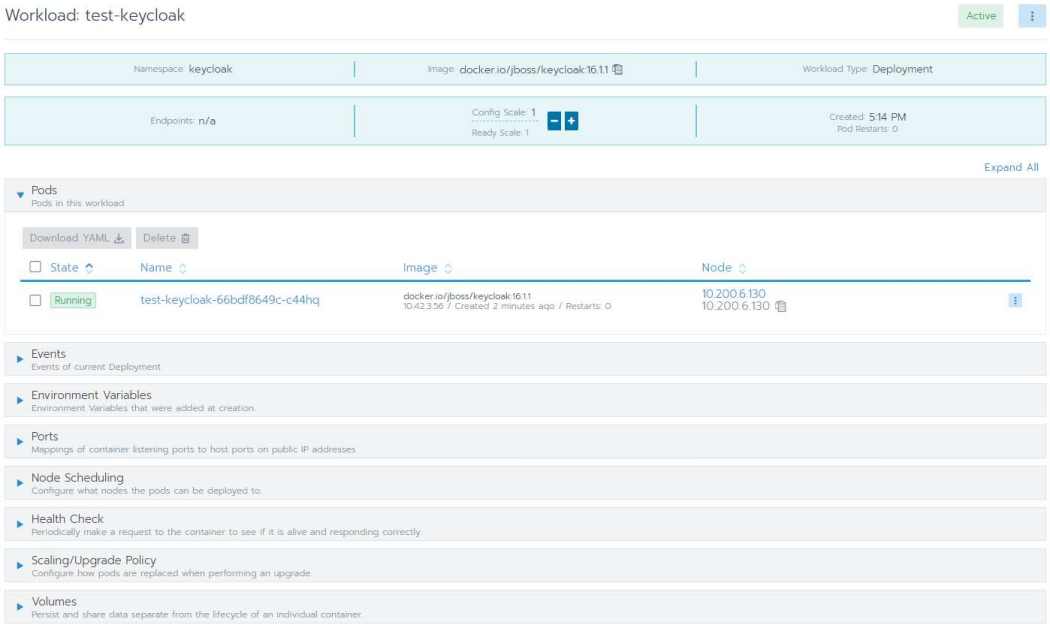

Figura 65.- Asignación del pod en uno de los nodos.

Prueba de Carga.

Consiste en desplegar la misma aplicación y validar que se coloque en un nodo worker distinto y que el balanceo de carga sea igual en ambos nodos.

Como se ve en la figura 66 los pods se están asignando en diferentes nodos.

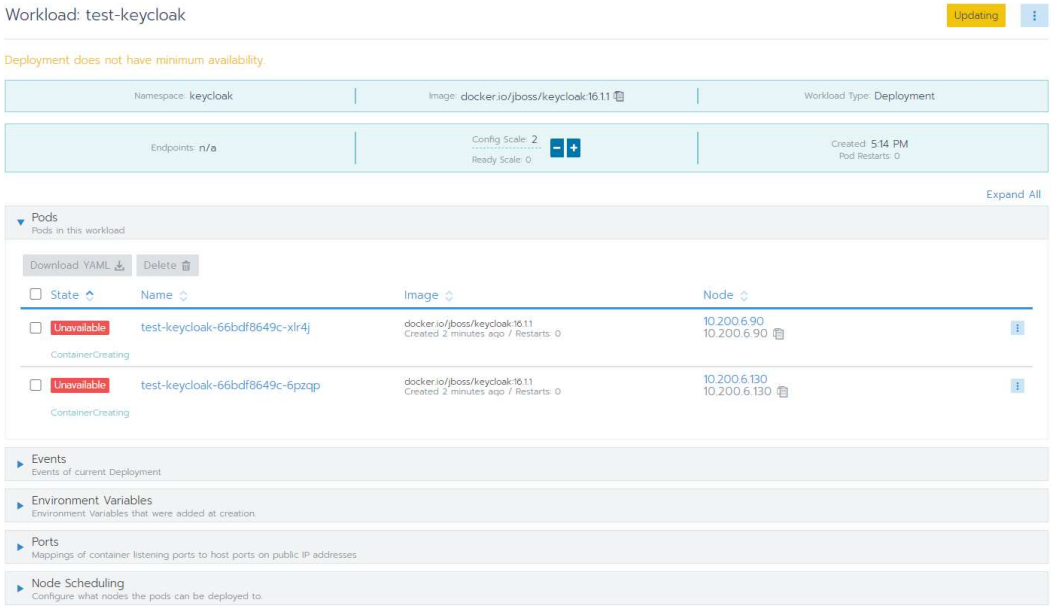

Figura 66.- Asignación de los pods.

En consecuencia, se puede ver los pods levantados y operativos cada uno en diferente nodo.

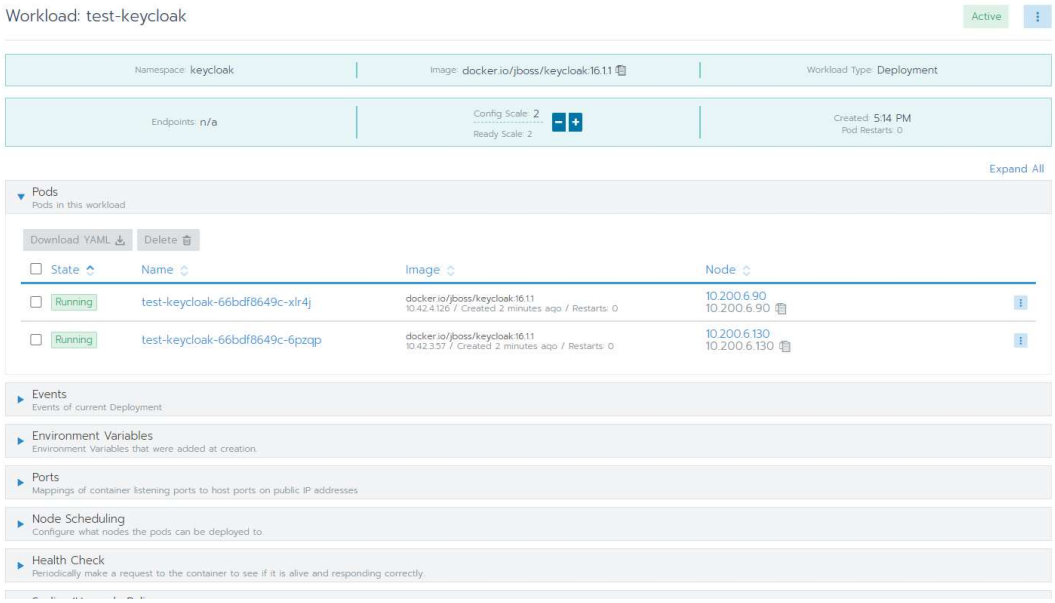

Figura 67.- Asignación de nodos.

Prueba de Resistencia.

Esta prueba valida el funcionamiento de la VIP montada sobre los Haproxy. De esta manera si un equipo balanceador de carga llegase a caer, la VIP automáticamente se levantará en el equipo que estará con estado standby.

Como se puede observar en la figura 68 la VIP y el servicio del Haproxy se encuentra en el servidor ha1.

```
[root@hal ~]# pcs status
Cluster name: hacluster
Stack: corosync
Current DC: hal (version 1.1.23-1.el7_9.1-9acf116022) - partition with quorum
Last updated: Wed Sep 28 19:13:33 2022
Last change: Wed Sep 28 19:05:36 2022 by root via cibadmin on hal

2 nodes configured
2 resource instances configured

Online: [ hal ha2 ]

Full list of resources:

 Cluster_VIP    (ocf::heartbeat:IPaddr2):       Started hal
 HAproxy        (systemd:haproxy):      Started hal

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
[root@hal ~]#
[root@hal ~]#
[root@hal ~]#
[root@hal ~]# ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:d9:6b:42 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.10/24 brd 192.168.0.255 scope global noprefixroute dynamic enp0s3
       valid_lft 3035sec preferred_lft 3035sec
    inet 192.168.0.12/24 brd 192.168.0.255 scope global secondary enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fed9:6b42/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
[root@hal ~]#
```

Figura 68.- Servicios Iniciados.

Previo a esta comprobación se procede a revisar que el servicio del Ha Proxy se muestre sobre la VIP mencionada anteriormente. Figura 69.

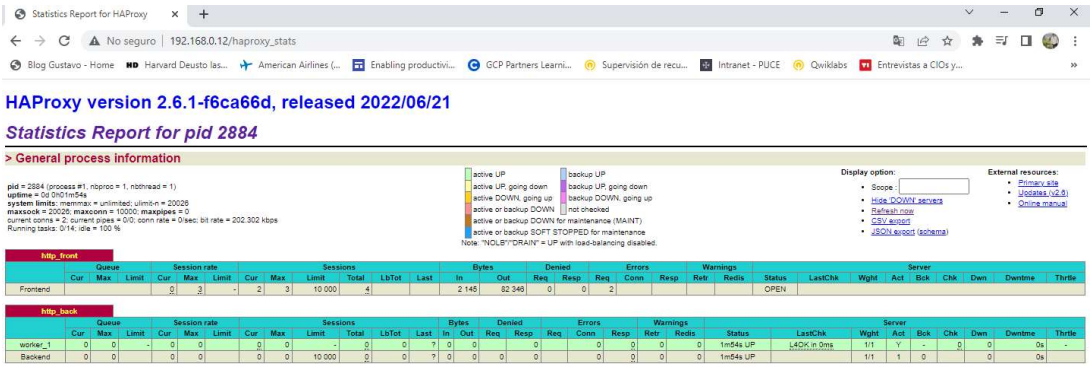

Figura 69.- Servicios Ha Proxy levantado sobre la VIP.

Se valida el equipo que hace de Standby. En este caso el equipo ha2 tiene los servicios levantados, pero hacen referencia al equipo ha1 donde se encuentra la VIP. Figura 70. Esto quiere decir que se encuentra activo y a la escucha para levantar la VIP en caso de un fallo en el ha1.

```
[root@ha2 ~]# pcs status
Cluster name: hacluster
Stack: corosync
Current DC: ha1 (version 1.1.23-1.el7_9.1-9acf116022) - partition with quorum
Last updated: Wed Sep 28 19:19:34 2022
Last change: Wed Sep 28 19:05:36 2022 by root via cibadmin on ha1

2 nodes configured
2 resource instances configured

Online: [ ha1 ha2 ]

Full list of resources:

 Cluster_VIP    (ocf::heartbeat:IPaddr2):       Started ha1
 HAproxy        (systemd:haproxy):      Started ha1

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
[root@ha2 ~]# ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:1b:05:6e brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.11/24 brd 192.168.0.255 scope global noprefixroute dynamic enp0s3
       valid_lft 2739sec preferred_lft 2739sec
    inet6 fe80::a00:27ff:fe1b:56e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
[root@ha2 ~]#
```

Figura 70. Servicios en el equipo Standby

Para este ejemplo se procede a simular que en el equipo ha1 con la VIP activa sufre una caída. Para esto se baja la interface de red y se comprueba que la VIP en efecto salta al equipo ha2. Figura 71.

Se procede a simular una caída del equipo principal y se comprueba que no existe dentro de sus interfaces la VIP 192.168.0.12.

```
[root@ha1 ~]#
[root@ha1 ~]#
[root@ha1 ~]# ip add
1: lo: <LOOPBACK> mtu 65536 qdisc noqueue state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1
000
    link/ether 08:00:27:d9:6b:42 brd ff:ff:ff:ff:ff:ff
[root@ha1 ~]#
[root@ha1 ~]#
[root@ha1 ~]#
[root@ha1 ~]#
[root@ha1 ~]#
[root@ha1 ~]#
[root@ha1 ~]# _
```

Figura 71.- Shutdown equipo principal.

Se comprueba que en el equipo ha2 se encuentre la VIP. En este caso saltó manteniendo la alta disponibilidad. Figura 72.

```
[root@ha2 ~]# ip ad
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:1b:05:6e brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.11/24 brd 192.168.0.255 scope global noprefixroute dynamic enp0s3
       valid_lft 2958sec preferred_lft 2958sec
    inet 192.168.0.12/24 brd 192.168.0.255 scope global secondary enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe1b:56e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
[root@ha2 ~]#
```

Figura 72.- Salto de la VIP al equipo Standby

Se procede a validar que el servicio de Ha proxy se encuentre operativo. Con esto se valida que el diseño mantiene la alta disponibilidad. Figura 73.

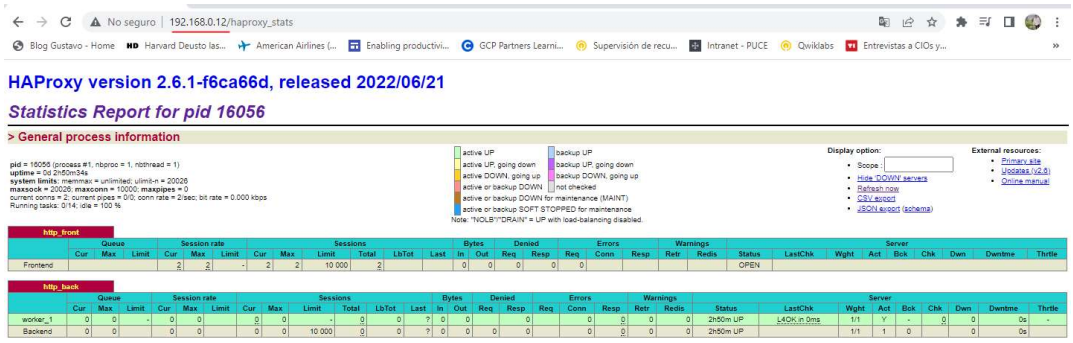

Figura 73.- Comprobación de la alta disponibilidad

En la siguiente tabla se puede observar los parámetros de evaluación que se validaron para el diseño propuesto. Figura 74.

Evaluación del diseño de la solución propuesta para el despliegue			
Pruebas	**Parámetros Evaluados**	**Si**	**No**
Funcionalidad	Se descargan las imágenes de los repositorios Se asigna la imagen a un namespace Se asigna la imagen a un nodo worker	x x x	
Escalado y Balanceo	El pod se asigna en un nodo distinto y mantiene su servicio en el nodo actual El HaProxy balancea de manera equitativa la carga Se puede visualizar el administrador del HaProxy	x x x	
Resistencia y Alta Disponibilidad	Ante un fallo la VIP salta al server en estado Standby El servicio de HaProxy atado a la VIP mantiene su operatividad Se puede visualizar el administrador del HaProxy despues de una caida	x x x	

Figura 74.- Comprobación de la alta disponibilidad

Conclusiones y Recomendaciones

Para el desarrollo de este diseño es importante realizar un estudio de cada componente a ser utilizado. En el mercado existen varias opciones de software libre que permiten armar una arquitectura como la diseñada en este trabajo y las cuales deben ser tomadas en cuenta, estudiadas y analizadas a la hora de diseñar una solución de arquitectura de microservicios.

Analizar los requerimientos por parte del usuario permitió entender a profundidad las necesidades y el comportamiento que el diseño debería tener para cumplir con los objetivos planteados.

El despliegue individual de cada componente que forma este diseño hizo la tarea un poco más sencilla.

La evaluación de este diseño a través de las pruebas de funcionalidad, carga, y resistencia ayudaron a validar que el diseño es capaz de soportar el despliegue de aplicaciones contenerizadas.

Las herramientas como Kubernetes, Rancher y Docker permitieron realizar el despliegue de las aplicaciones de manera más rápida y sencilla.

Referencias

López, A. (s. f.). *Despliegue de aplicaciones escalables con kubernetes* [Trabajo fin de grado]. Universidad de Almeria.

Fernández, I. (s. f.). *Aplicar kubernetes sobre un entorno cloud comunitario y descentralizado* [Grado de ingeniería informática]. Universitat Oberta de Catalunya.

Mora, N. (s. f.). Arquitectura basada en microservicios para sistemas de e-commerce de la empresa lcc opentech, c.a. [Trabajo Instrumental de Grado]. Universidad Católica Andrés Bello.

Lindegren, E. (s. f.-b). Preparing the Apache HTTP Server for Feedback Control Application [Master Thesis]. Lund University.

Hurtado de Barrera, J. (2010). Metodología de la investigación: guía para una comprensión holística de la ciencia (4º ed.). Caracas, Venezuela: Quirón Ediciones.

Quintana, A. y Montgomery, W. (Eds.) (2006). Psicología: Tópicos de actualidad. Lima: UNMSM

Kubernetes. (2022). Recuperado 21 de octubre de 2022, de https://kubernetes.io/docs/home/

Rancher. (s. f.). Recuperado 21 de octubre de 2022, de https://www.rancher.com/
HaProxy community edition. (s. f.). Recuperado 21 de octubre de 2022, de http://www.haproxy.org/

An Introduction to HAProxy and Load Balancing Concepts. (s. f.). Recuperado 21 de octubre de 2022, de https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts