Estudio de soluciones para Redes de Contenedores y su desempeño en la Nube

Carlos Fernández Hernández

Abstract- La tecnología de contenedores ocupa un espacio importante en la industria de aplicaciones Web y esta siendo un foco de investigación importante, esto debido a que su uso se a vuelto necesario para entregar un buen servicio aplicaciones orientadas a ser desplegadas en la Nube. Para entender el funcionamiento de esta tecnología es importante familiarizarse con otros conceptos, como lo son la arquitectura de microservicios con la cual esta estrechamente relacionada por ser una forma de virtualizacion que permite implementar esta arquitectura. Por otro lado uno de los principales desafíos que este tipo de tecnología esta enfrentando es el relacionado a la comunicación entre host que implementan estos sistemas, siendo la configuración de la red uno de los principales cuellos de botella al momento de maximizar el desempeño. Dentro de las herramientas mas utilizadas para desarrollar aplicaciones en este contexto se encuentra Docker, el cual presenta soluciones para enfrentar el problema de la conectividad entre contenedores las cuales no son las únicas ni las mejores. En este proyecto se explora el impacto en el desempeño throuhgput(UDP y TCP) y latencia que tiene el uso de distintas soluciones(Flannel, Calico, Docker Swarm Overlay) en la comunicación de dos contenedores Docker en un ambiente de la Nube(Amazon Web Service). Se observa como Calico una implementación que no introduce latencia de encapsulado en los paquetes. Se compara de igual manera el desempeño en maquinas con distintas capacidades de enlace.

I. Introducción

Hoy en día la existencia de aplicaciones web de gran escala es común y la necesidad de que estas sean rápidamente escalables, tengan tiempos de mantenimiento mínimo y sean confiables se a tornado en una de las principales preocupaciones de la industria. La manera usual de enfrentar el desarrollo de aplicaciones era usando el enfoque de una arquitectura monolítica la cual presentaba la aplicación como un solo proyecto el que todas sus partes sean estrechamente interconectadas volviéndolas dependientes unas de otras para el correcto funcionamiento de la aplicación, naturalmente este enfoque presenta limitaciones para lograr entregar la mejor calidad de servicio según los aspectos mencionados anteriormente[1].

En el año 2009 Netflix se topo con las limitaciones que presentan las arquitecturas monolíticas al ver dificultado el mantenimiento a su plataforma de streaming producto de lo engorroso que se producía el buscar errores o la fuente de problemas. Por otro lado Netflix estaba en un proceso de ampliación de plataformas y crecimiento de usuarios a lo que e le vio imposible mantener el ritmo. Como solución a estos problemas Netflix implementa lo que se denomina arquitectura de microservicios. Este enfoque plantea modularizar las aplicaciones las cuales deben construirse mediante aplicaciones con propósito especifico independientes que se comunican entre si.

Para implementar la arquitectura de microservicios era necesario romper con el esquema que implementa una sola aplicación por servidor, naturalmente esto lleva a una sub utilización de recursos en una sola maquina y a una necesidad abrumadora de maquinas para implementar la arquitectura. El concepto de virtualizacion de servidores viene a enfrentar este problema, este proceso permite crear múltiples instancias dentro de un mismo servidor y cada instancia ejecuta su propio SO(Sistema Operativo) al utilizar maquinas virtuales, a estas instancias se les asigna sus propios recursos repartiendo distribuyendo así los recursos del servidor.

A pesar de esto las maquinas virtuales presentan sus propias problemas en términos de utilización de recursos, pues se necesita incluir en cada instancia todos los archivos necesarios para correr el sistema operativo, esto a pesar de que la aplicaciones a ejecutar no los necesitara. Para enfrentar este problema surge un método alternativo de virtualizacion llamado contenedores, este proceso empaqueta las aplicaciones en conjunto con los archivos necesarios para que estas puedan ejecutarse, produciendo archivos de menor tamaño y tiempos de despliegue mas pequeños en comparación a las maquinas virtuales.

La tecnología de contenedores posee una gran popularidad hoy en día para un fácil y rápido despliegue de aplicaciones en la nube. Una de las herramientas mas utilizadas para trabajar con estos elementos es Docker el cual presenta una serie de opciones para configurar, desplegar y manejar contenedores de manera eficiente. Para escalar el uso de estos elementos de tal manera que se permita la construccion de aplicaciones complejas utilizando contenedores, es necesario un elemento que maneje y automatice el despliegue de estos elementos a gran escala, para esto existen lo denominados orquestadores de contenedores que permiten coordinar las aplicaciones que se desplieguen, manejando ~~parámetros como son~~ los recursos del servidor, networking, tiempos de despliegue, respaldos etc. Dentro de los mas populares se encuentran Kubernetes y Docker Swarm.

Una de las cosas mas importantes que se debe tener en cuenta al momento de trabajar con contenedores es la selección adecuada de la solución de comunicación para establecer conexión entre contenedores que por naturaleza se encuentran aislados. Este factor toma mayor relevancia aun cuando se habla de contenedores que se encuentran en distintos maquinas, ya que factores de desempeño comienzan a tomar protagonismo en el desempeño de las aplicaciones y se transforma en un cuello de botella. Por esto ultimo se convierte en una necesidad el tener conocimiento de las distintas soluciones que existen, nativas o no a la herramienta utilizada(en este caso Docker).

Este proyecto tiene el objetivo de estudiar tres de las soluciones mas comunes que se encuentran hoy en dia para el trabajo con contenedores. Estas solucions son:

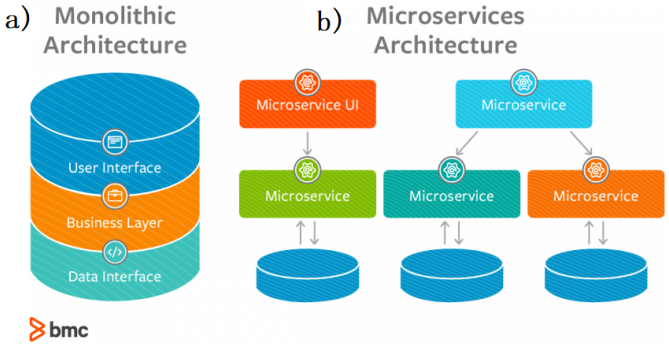

Fig. 1. a)Esquema arquitectura monolítica b) Esquema arquitectura de Microservicios [2].

- Docker Swarm Overlay
- Flannel
- Calico

Por otro lado se implementaran estas soluciones en contenedores de prueba utilizando instancias en el servicio de la Nube que ofrece Amazon Web Service.

II. Arquitectura de microservicios y Virtualizacion

A. Microservicios

La arquitectura de microservicios emerge como alternativa al enfoque que se tiene al momento de desarrollar aplicaciones, promoviendo que estas sean estructuras modulares cuyos elementos sean independientes y su interaccion construya la aplicacion final. Esta descripción se hace evidente al observar la figura 1-a ,que es el método usual de desarrollar aplicaciones mediante una estructura monolítica con elementos directamente conectados y dependientes entre si, y se compara con la arquitectura que se observa en la figura 1-b que presenta esta arquitectura de microservicios, en la cual se reconocen los elementos independientes que conectan con sus propias base de datos. Aplicaciones desarrolladas e implementadas de manera modular permiten realizar mantenimientos con un esfuerzo menor al de las arquitecturas monolíticas debido a que cada parte se encuentra aislada del resto lo que permite focalizar los esfuerzos en alguna parte determinada y monitorear cada una de estas de manera independiente. Donde se alcanzan mayores ventajas es al momento de escalar el servicio, esto es especialmente importante cuando se habla de aplicaciones con una cantidad considerable de usuarios(millones), en la que se a demostrado que el uso de este tipo de arquitectura puede bajar los costos en infraestructura hasta un 77.08% [1].

B. Virtualizacion

El concepto de virtualización es clave para entender la implementación de la arquitectura de microservicios. Este concepto nace como una solución a la subutilización que estaban sufriendo los servidores en su momento, los cuales dedicaban sus recursos a un solo propósito (mail,web, aplicaciones, etc) y un solo sistema operativo, esto era confiable ya que un servidor dedicado impedía que distintos programas interactuaran entre si pudiendo causar problemas en el servicio. La subutilización

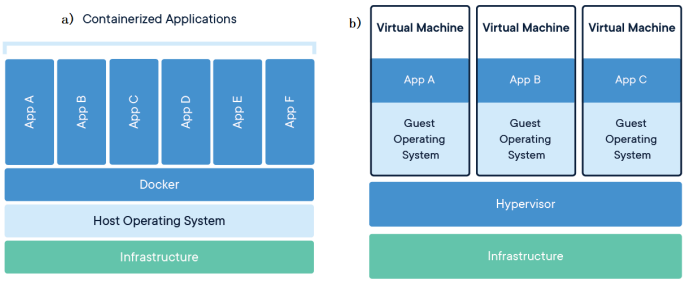

Fig. 2. a)Esquema Virtualizacion usando contenedores b) Esquema Virtualizacion usando VM.[4]

devenía al tener tiempos en la que la aplicación no era requerida en su máximo potencia. A su vez la demanda por servidores solo iba en aumento por lo que los costos se volvieron otro problema para los administradores de data center, que veían la falta de espacio y el alto consumo de energía como un factor preocupante.

Existen tres formas de acercarse a este concepto, Full virtualization, Para-Virtualization y OS-Level Virtualization. Full-Virtualziation se basa en un componente esencial conocido como hypervisor o VMM(Virtual Machine Manager), el cual se encarga de monitorizar los recursos físicos del servidor a la vez que los asigna a las distintas máquinas virtuales o servidores invitados que corren sistemas operativos existentes en el servidor(por lo general este componente es instalado directamente en el hardware) los cuales se encuentran corriendo en su propia máquina virtual. El esquema de virtualizacion utilizando maquinas virtuales(VM) se pude observar en la figura **??**-b en la que se reconocen las partes esenciales, teniendo un Hypervisor entre el hardware y las distintas maquinas virtuales las cuales cada una ejecutaran su propia aplicación estando cada una separadas de la otra.

Como alternativa a este método se tiene lo que se conoce como OS-Virtualization el cual es implementado mediante Contenedores, los cuales a diferencia de lo que ocurre con las Maquinas Virtuales comparten el kernel del sistema operativo en vez del hardware de la maquina. El esquema de este tipo de virtualizacion se puede observar en la figura 2-a

III. Docker y el uso de contenedores

A. Contenedores

~~Como se menciono antes~~ a nivel de OS-Virtualization se tiene los contenedores los cuales consisten en paquetes que incluyen solamente aplicaciones y las dependencias necesarias para que esta pueda ser ejecutadas generando archivos mucho más livianos que las máquinas virtuales ya que mientras los contenedores están en el orden de Mb, las VM pueden estar en el orden de Gb[3]. A diferencia de las maquinas virtuales que comparten el hardware cuyos recursos son coordinados por un hypervisor, los contenedores comparten el kernel y son manejados por un container engine(Docker por ejemplo) por lo tanto al hardware solo es accedido por una sola parte.

Para entender la implementación de los contenedores el concepto de Namespaces es indispensable ya que de esta

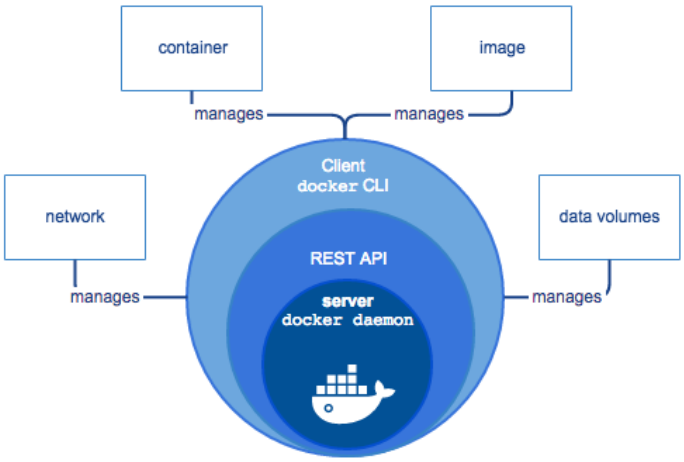

Fig. 3. Flujo de componentes en la aplicación Docker.[4]

forma se logra aislar los contenedores del resto de procesos/contenedores que puedan estar corriendo en el host, esto lo realiza mediante la asociación de procesos con recursos específicos lo que crea un ambiente aislado de otros procesos, pues una vez agrupado recurso a un namespace este queda inhabilitado para otros existentes. Si bien cada contenedor queda aislado de lo que existe fuera de su namespace, el host es capaz de ver cada proceso y administrarlos por lo cual por este medio es posible de establecer enlaces entre los contenedores y hacia afuera del host habilitando los mecanismos de comunicación y coordinación.

B. Docker Engine

Dentro de las herramientas mas utilizadas para trabajar con contenedores se encuentra Docker Engine. Este es una plataforma de desarrollo, creación y manejo de aplicaciones encapsuladas en contenedores. Las componentes de la cual se construye esta herramienta se puede observar en la figura 3 y se pasan a detallar a continuación:

- Server Docker Daemon: Este servidor es la parte mas interna de Docker y funciona como un proceso daemon corriendo en la maquina en la que se utiliza la herramienta, es el nucleo encargado de responder e implementar las funciones de Docker.
- Rest API: Corresponde a la interfaz entre los comandos Docker introducidos por el usuario en el client Docker CLI y el servidor Docker Daemon.
- Client Docker CLI: Corresponde al cliente de docker que presenta al usuario con una serie de comandos que le permiten manejar las diversas funciones de Docker, como pueden ser el manejo de imágenes de contenedores, la configuración de red de estos, ETC.

Estos elementos principales de Docker interactúan con los componentes que otorga el usuario conocidos como objetos Docker. Dentro de estos se tiene:

- Dockerfile: Definición de los pasos necesarios para crear una imagen especifica.

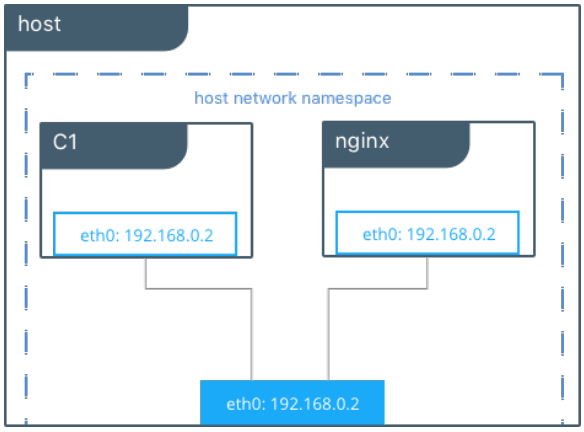

Fig. 4. Arquitectura de solución Host[4]

- Imágenes: Define las dependencias necesarias y el código de la aplicación a correr en el contenedor.
- Contenedores: Corresponde a la imagen corriendo y desplegando la aplicación definida en la imagen.

IV. Estudio Docker Networking

Para establecer la comunicación entre contenedores existen diversas soluciones que son nativas de la herramienta Docker y otras que son de terceras partes y se pueden incluir como drivers. A continuación se presenta un estudio de las que se implementaron en este proyecto.

A. Host

Esta solución es la mas simple de todas las que se tiene a disposición, en la figura 4 se observa la configuración que implementa en la cual solamente conecta los contenedores al interfaz de red de la maquina que los hospeda. Esta solución introduce una mínima latencia a la comunicación, pero desvirtúa totalmente el objetivo de la implementación de contenedores que es el tener distintas aplicaciones aisladas entre si en una sola maquina al no ocupar la separacion mediante namespaces. De esta forma esta configuración de red para contenedores es una forma de tener una idea de la mínima latencia que se puede obtener en la comunicación así como también del máximo desempeño posible en términos de througput TCP y UDP.

B. Docker Swarm Overlay

Esta solución es la que presenta Docker como nativa para solucionar el problema de la comunicación entre contenedores en distintos Host esta implementación se puede observar en la figura 5. Esta estructura se basa en la construcción de un canal de comunicación de la capa de enlace de los contenedores(Overlay Network) sobre la red que comunica los host(Underlay Network). Esto ultimo se logra utilizando el encapsulado VxLan el cual agrega un headear de este protocolo a los frames generados por los contenedores y lo envia mediante protocolo UDP utilizando las direcciones de los host para dirigir los paquetes mediante la red externa, esto es precisamente lo que se observa en la figura 5 teniendo el

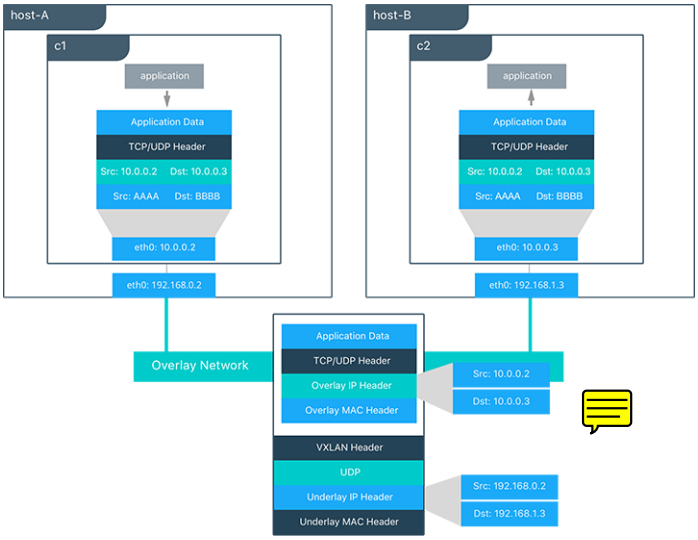

Fig. 5. Arquitectura de solución Docker Overlay Swarm[4]

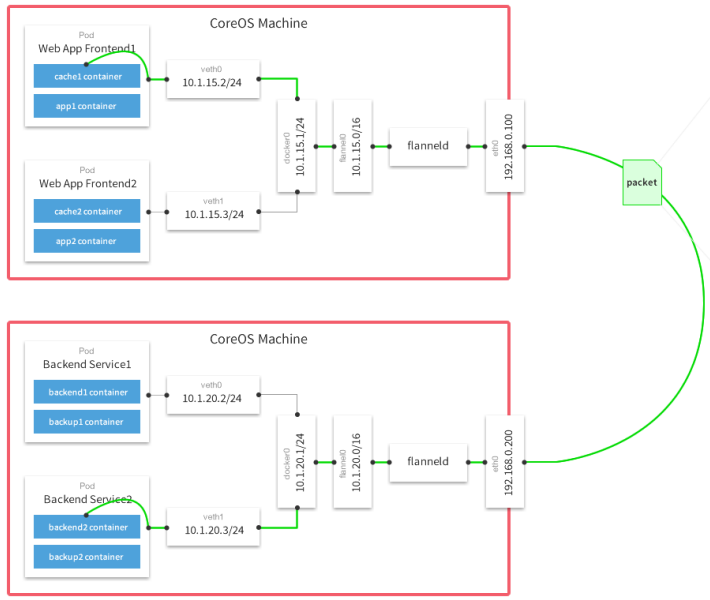

Fig. 6. Arquitectura de solución Flannel[7]

paquete generado por el contenedor c1 en la maquina host-A el cual al salir es empaquetado agregando la información de la maquina de fuente y de destino la IP de host-B. El empaquetado y desempaquetado de frames que utilizan VXLAN se encuentra soportado a nivel de kernel, pero aun asi existe overhead que se agrega a la comunicación producto de este proceso que degrada el desempeño de la red.

C. Flannel

Desarrollado por CoreOS, Flannel es una solución externa a Docker para establecer comunicacion entre contenedores y utiliza como base la llave distribuida ETCD para construir un cluster con los host que tendran los contenedres con los cuales se quiere establecer comunicacion, esta llave distribuida permite mantener informacion sobre la configuraicon de red asi como almacenar el mapeo de direcciones IP de cada host asociadas a los contenedores que existan en cada uno. Para los procesos de comunicacion propiamente tal establece un proceso Flannel el cual genera una subred a la cual debe asociarse Docker para poder incluir posteriormente a los contenedores, esto precisamente se aprecia en la figura 7 en la cual se observa la estructura de una red Flannel.

Por otro lado los paquetes son enviados en dos posibles encapsulados(sse deja a la libertad del usuario la configuración elegida), al igual que Docker Swarm Overlay utilizando el protocolo VXLAN o utilizando encapsulad UDP. Al igual que ocurre con la solucion nativa de Docker esta incluye un overhead de encapsulado siendo el encapsulado UDP el cual afecta mas el desempeño al no estar soportado a nivel de kernel como si ocurre con el protocolo VXLAN, además de esto constituye un proceso mas complejo de comenzar a utilizar al necesitar la configuración previa de ETCD.

D. Calico

Esta solución a diferencias de las presentadas hasta ahora(A excepción de Host) no ocupa ningun tipo de encapsulado ni red overlay para implementar la comunicación entre contenedores, en cambio plantea una estructura de solución semejante a como funciona la tercera capa. La implementación de calico considera una llave distribuida para almacenar configuración de la red, tablas ip y de ruteo que sean accesibles por los host que quieran comunicar sus contenedores. Para establecer la comunicación se configura un proceso o contenedor calico en cada maquina que sera el nodo que maneje la red, este aprovecha las cualidad de IP Forwarding que tiene el kernel de linux para implementar un vRouter el cual es el encargado de redireccionar los paquetes generados por los contenedores hacia la red, además de comunicar mediante el protocolo BGP la listas de ruteo actualizadas. Al igual que sucede con flannel, Calico asigna los contenedores a sub-redes asignándoles direcciones IP de un rango establecido y acordado por todos los host que participan en la comunicación, permitiendo que el mapeo de estas direcciones quede almacenado en la llave ETCD y sea accesible por cada nodo Calico.

~~Como se menciono antes~~ los nodos Calico establecerán comunicación con los routers de la estructura de red en la que se encuentran los host y comunicandole las rutas hacia los contenedores para permitir el reenvio de paquetes que estos generen. Sin embargo por lo general en los servicios de la nube que existen disponibles en los routers de estos Data Center se encuentra activada la opción Check Source/Destination lo que produce que aquello paquetes cuya direccion de destino o fuente no este registrada por los dispositivos sea descartada, por este motivo existe la opción de que calico utilice el encapsulado IP sobre IP para esconder la fuente y destino de los paquetes que envían los contenedores usando la de los host. Si bien esto ultimo puede introducir un overhead por el encapsualdo, este debería ser en teoría menor al que introduce VXLAN al agregar menos bits en el encapsulado.

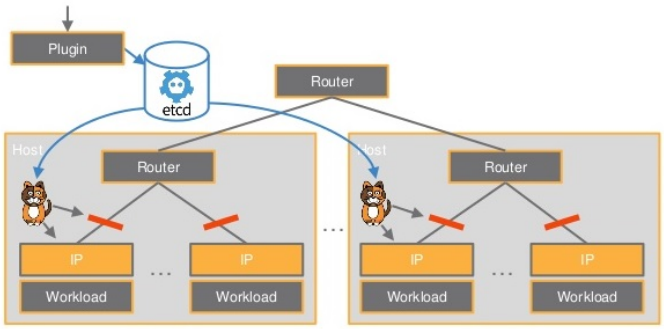

Fig. 7. Arquitectura de solución Calico[8]

E. Llave distribuida ETCD

~~Como se menciono antes~~ tanto Flannel como Calico utilizando para sus configuraciones el concepto de llave distribuida ETCD, el cual tiene como objetivo almacenar datos para que sean accesibles por un cluster de maquinas distribuidas, en este contexto estas maquinas serán los host que tienen contenedores que quieren comunicarse y los datos serán las configuraciones, mapeos de IP contenedor-Host, tablas de ruteo, ETC. En la figura 8 se observa la estructura de este tipo de llave distribuida a la cual todos los miembros del cluster tienen accesos pudiendo escribir o leer los datos allí almacenados. Para distribuir la información utiliza el protocolo raft, el cual es un algoritmo de consenso que busca que todos los nodos del cluster es ten de acuerdo con los valores almacenados, por lo cual cada uno almacena un registro el cual es sincronizado utilizando este algoritmo.

V. Implementación y Resultados

A. Configuración

Las pruebas realizadas implementando las diversas configuraciones de networking de contenedores que se detallaron anteriormente se realizaron utilizando los servicios gratuitos que ofrece Amazon Web Services(AWS). Esta plataforma permite desplegar instancias EC2 de servidores con maquinas de distintas caracteristicas, para este proyecto se trabajo especificamente con dos instancias distintas, estas son:

- t2.micro: 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, Network 1Gib/s
- r4.8xlarge: 32 vCPU, 2.3Ghz, Intel Broadwell E5-2686v4, 244 GiB memory, Network 10 GiB/s[5]

Como se observa en el detalle de las instancias t2.micro y r4.8xlarge la principal diferencia corresponde a la capacidad de enlace. Por otro lado las pruebas a realizar corresponde a las siguientes:

- Latencia: Entre contenedor host 1 a contenedor a host 2 medición de latencia utilizando comando ping
- Througput TCP: Se comprobara el desempeño de la comunicacion TCP utilizando Iperf.

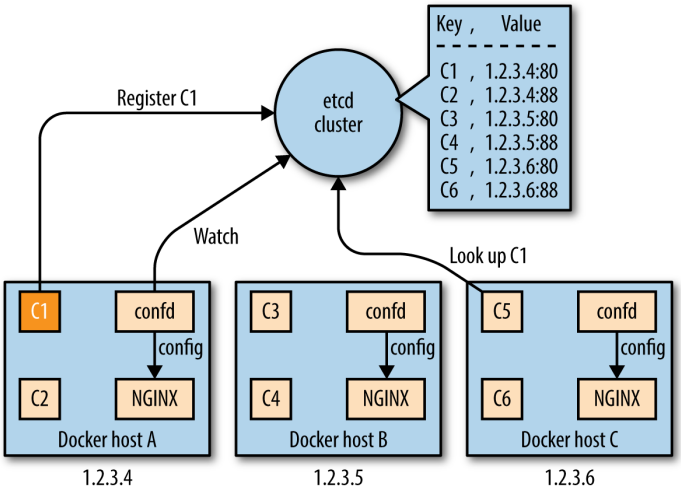

Fig. 8. Estructura llave distribuida ETCD [8]

- Througput UDP: Se comprobara el desempeño de la comunicacion UDP utilizando Iperf.

B. Latencia

~~Como se detallo en la sección anterior~~ las medidas de latencia se hicieron utilizando el comando ping, para esto se tomaron 100 muestras para cada configuración de red y se gráfico el valor máximo, mínimo y promedio de las latencias. El resultado de esta prueba se puede observar en la figura 9, en este gráfico los valores de latencia máxima son los que mas destacan del cual se pueden realizar algunos comentarios generales, como el hecho de que se marca una diferencia entre la latencia de la configuración host con el resto de las configuraciones, esto es lo esperable pues no introduce ningún timo de overhead entre el contenedor y la salida a la red externa.

De todas formas el valor máximo no indica ningún patrón(al tratarse de un solo valor) y no se puede sacar ninguna conclusión con el, de esta forma en la imagen 10 se elimina para acentuar la curva de promedios. En esta gráfica se consolida la configuración host como la que introduce mínima latencia a la comunicación seguida de cerca por la solución calico, esto es lo esperable ya que el retardo que introduce calico es debido a que los paquetes deben pasar por el nodo calico para ser reenviados a su destino, esto degrada el desempeño en menor medida que el encapsulado que debe realizar tanto Flannel como Docker Swarm Overlay. Por ultimo el que mayor retardo impone es la configuracion Flannel que utiliza el encapsulado UDP, esto debido a que a diferencia de VXLAN no se encuentra soportado a nivel de kernel.

C. Througput TCP instancia t2.micro

Para medir el desempeño TCP ~~como se menciono antes~~ se utilizó la herramienta iperf y como instancia AWS se utiliza t2.micro que posee una capacidad de enlace de 1GB/s. Los resultados se pueden observar en la figura 11 y en primera instancia se notan algunas discrepancias con lo que se

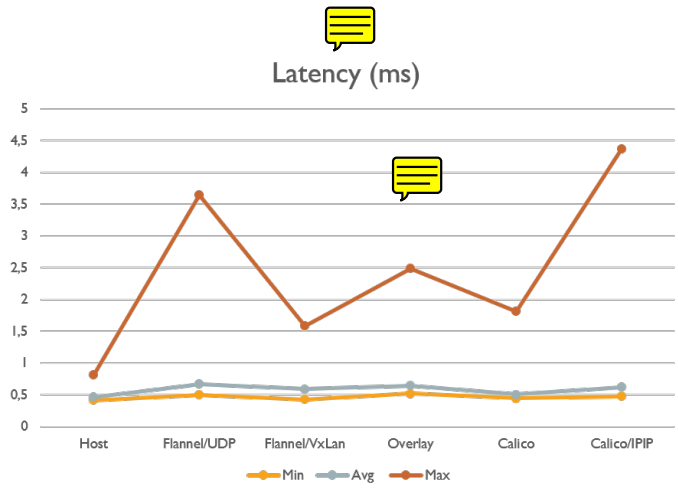

Fig. 9. Resultados pruebas latencia

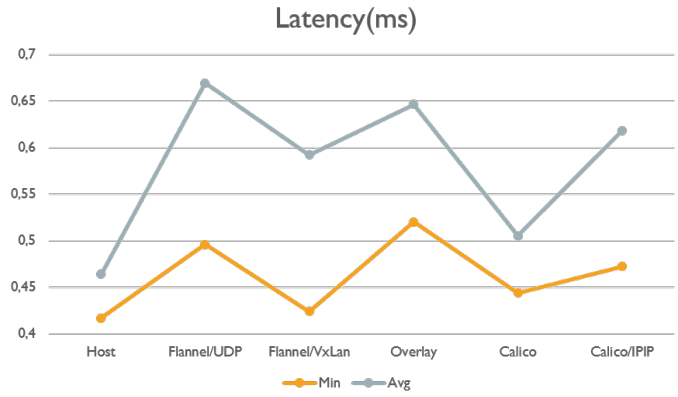

Fig. 10. Resultados pruebas latencia sin curva de valores máximos

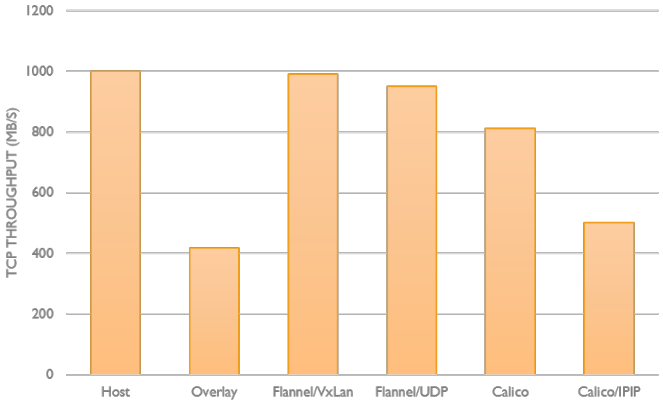

Fig. 11. Resultados pruebas throughput TCP

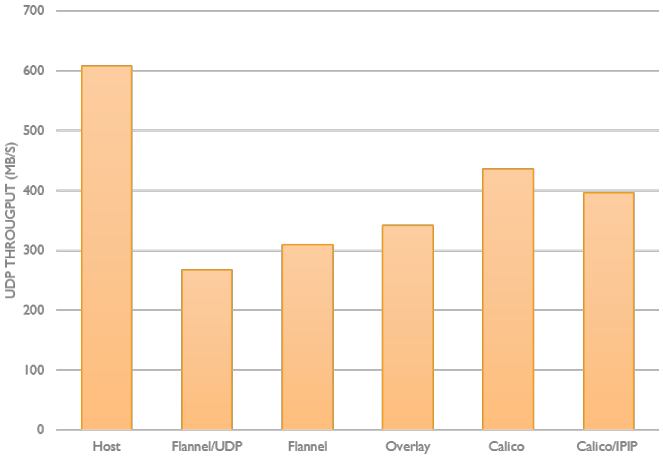

Fig. 12. Resultados pruebas throughput UDP

menciona en la teoría, esto en particular al desempeño que se muestra para la solución Flannel, tanto VXLAN como UDP, siendo la que destaca junto con la solución Host alcanzando aproximadamente la máxima capacidad del enlace. Esto ultimo sucede debido a que no se considero un Maximum Transmission Unit(MTU) distinto para Flannel en comparación a las otras soluciones implementadas, mientras estas ultimas tiene por configuración por defecto 1500 MTU, Flannel utiliza la que el host tiene en su interfaz de red que en este caso es de 8950. Esto evidentemente permite que la cantidad de bytes a la salida para un solo paquete sea mejor incrementando el throughput, de esta forma se entiende la diferencia con los resultados que se obtuvieron en [6] en los cuales no se detallan las condiciones de implementacion.

D. Througput UDP instancia t2.micro

Los resultados obtenidos en la medición del throughput UDP se pueden observar en la figura 12 y a diferencia de lo que sucede con el throughput UDP se observa el rendimiento real y bajo las mismas condiciones de cada solución, esto debido a que para esta prueba se limito el tamaño de paquetes a enviar en 1024bits, de igual forma se limito el ancho de banda máximo para asegurar perdidas de paquetes menores al 1%. El limitar el tamaño de paquetes permite obtener resultados como los que se observan en el gráfico de la figura 12 que representan un desempeño comparable al no estar distorsionados por la diferencia de MTU. Nuevamente se observa lo que es esperable según la teoría siendo la solución Calico la que presenta el mejor desempeño seguido de Calico IPIP, el pero desempeño se lo lleva Flannel UDP por las razones que ya se mencionaron. Estos resultados van en acuerdo con lo que se pudo observar en [6] por lo menos en terminos cualitativos.

E. Throughput TCP y UDP instancia r4.8xlarge

En este caso se utilizo una instancia que ofrece una capacidad de enlace de 10GB/s notando diferencias claves y no se considero aquellas configuraciones alternativas de Flannel(encapsulado UDP) y Calico(encapsulado IPIP). En el caso de througput TCP se realizo la prueba de igual manera que con la instancia t2.micro, pero en este caso se nota que el desempeño es mas tolerante a la diferencia de MTU, esto debido principalmente a que la capacidad de enlace compensa esta diferencia en el tamaño de los paquetes incrementando el througput de manera equitativa y viendo las diferencias en los valores cercanos a la capacidad máxima del enlace. El caso de througput UDP es mas claro y vuelve a repetir lo que sucede

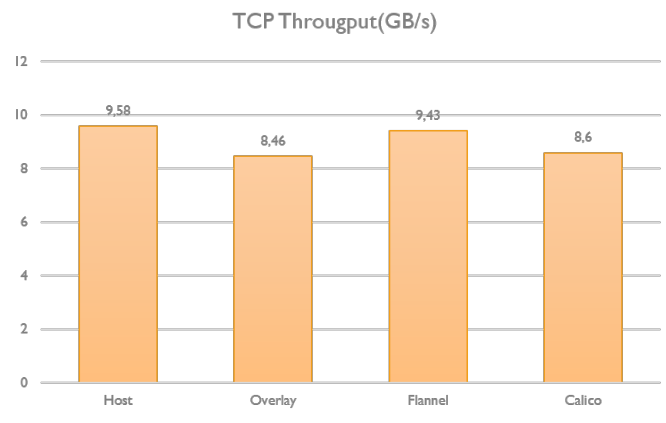

Fig. 13. Resultados pruebas throughput TCP instancia enlace de 10GB/s

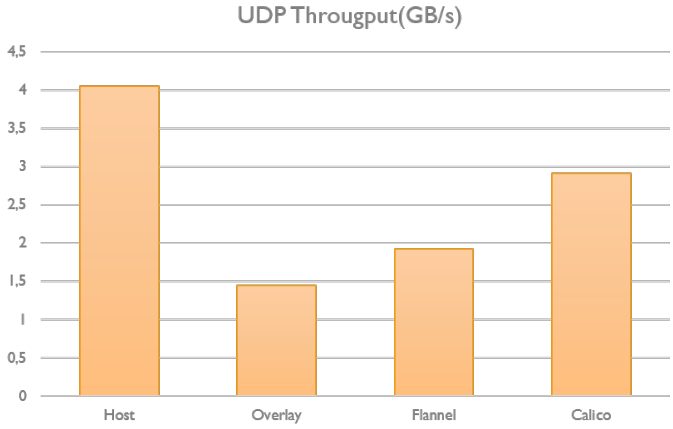

Fig. 14. Resultados pruebas throughput UDP enlace de 10GB/s

con la instancia de enlace de menor capacidad, en este caso se limito también el tamaño de los paquetes por lo cual los desempeños son comparables.

VI. Conclusión

En este proyecto se trabajó con una tecnología de contenedores que esta siendo fuertemente utilizada hoy en día en la industria para mejorar la calidad de los servicios y aplicaciones web, en especial aquellas que cuentan con una cantidad grande de usuarios como lo son la plataforma de streaming de Netflix, Amazon, ETC. Por estos motivos el estudiar sus características, fortalezas y limitaciones para poder seguir desarrollando estas herramientas de tal forma que cada vez sean mejores. En este caso se estudio uno de los puntos débiles actualmente de esta tecnología que corresponde a la configuración de red y su desempeño en la nube, contexto muy estudiado hoy en día. En particular se probo con tres soluciones al problema de la comunicación entre contenedores que se encuentran disponibles y son utilizadas con frecuencia.

De lo realizado en el proyecto se desprende que en términos de desempeño puro, tanto throughput y latencia la solución de Calico es la que mejor se comporta cuando se compara con las otras en mismas condiciones al introducir la menor cantidad de overhead en la comunicación, esto a pesar de se una de las soluciones externas a Docker. Por esto calico como solución sera la indicada bajo el contexto de lograr mínimo degrade en el desempeño. Aun asi se debe considerar que existen otros factores a considerar al momento de decidir que tipo de red se va a configurar para comunicar contenedores, como puede ser el problema de la seguridad o desempeño ante grandes cantidades de contenedores conectados(escalabilidad), de esta forma este estudio puede extenderse aun mas. Algunos parámetros como el MTU deben ser tomados en cuenta así como la capacidad del enlace al momento de elegir la configuración de red ya que como se observo anteriormente se puede obtener mejoras en el desempeño con un incremento de MTU o estas no serán necesarias si es que la capacidad del enlace es suficiente.

References

[1] M. Villamizar, *Infrastructure Cost Comparison of Running Web Applications in the Cloud using AWS Lambda and Monolithic and Microservice Architectures* .16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2016.
[2] BMC, https://www.bmc.com/blogs/microservices-architecture/.
[3] R. Morabito, *Hypervisors vs. Lightweight Virtualization: a Performance Comparison*. Jorvas, Finland:IEEE International Conference on Cloud Engineering, 2015.
[4] Docker,"https://https://success.docker.com/article/networking/," .
[5] R. Bankston,Performance o ainer Network Technologies in Cloud Environments, IEEE 2018
[6] H. Zeng, "Measurement and Evaluation for Docker Container Networking," *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery.*, 2017.
[7] https://coreos.com
[8] https://www.projectcalico.org/
[9] U. Abbasi, E.H.Bourhim, and H. Elbiaze, "A Performance Comparison of Container Networking Alternatives," IEEE Network, July/August 2019.