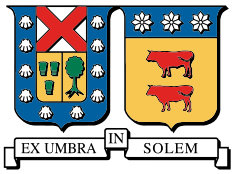


REDES DE COMPUTADORES II

Redes lógicas para la orquestación de contenedores mediante Kubernetes

Pablo Reyes Robles *201003038-5*
Matías Diaz *201121032-8*

20 de Enero de 2016

Índice

1. Resumen

Kubernetes es una plataforma que ofrece cluster de contenedores, entiendo que un contenedor es un espacio aislado similar a una máquina virtual pero con mayor eficiencia en el uso de recursos. El principal objetivo de estos clusters es alojar servicios y aplicaciones webs por lo que los contenedores deben tener acceso a la red individualmente y poder comunicarse entre ellos aún cuando el nodo en donde están contenidos no son los mismo. Ante este desafío Kubernetes utiliza una tecnología para cada objetivo, esta es docker para el acceso a internet de los contenedores y flannel para la comunicación entre contenedores internodales. Se pondrá explorará y pondra a prueba el funcionamiento individual de cada una de estas tecnologías, y se concluirá respecto a los resultados sus ventajas y sus desventjas.

2. Desarrollo

2.1. Introducción

En la actualidad se hace mucho uso de los servicios de "nube", principalmente en servicios de almacenamiento. Se ha transformado en elementos de uso diario pues, nos permite acceder a nuestros archivos desde cualquier lugar, gestionarlos como se nos ocurra, compartir información, etc. Sin embargo la nube no sólo ofrece servicios de almacenamiento, en la actualidad muchas empresas conocidas hacen uso de las herramientas de la computación en la nube o *Cloud Computing*, para llevar a cabo los objetivos planteados. Hay que ir un poco más allá y conocer las diferentes opciones que ofrece este tipo de tecnología y sentarse en la base del Cloud Computing para apreciar la diversidad de opciones, servicios, facultades de desarrollo y demás que nos ofrece esta herramienta. Esta base es la virtualización de recursos computacionales que se adecúan a las necesidades de cada organización que pretenda usar Cloud Computing para entregar sus servicios, siendo la **virtualización basada en contenedores** una opción eficiente que ha tomado mucha fuerza en los últimos años y está siendo muy utilizada por diferentes empresas. Es imperativo ser eficiente con los recursos que se trabajan y es por eso que este esquema resulta tan atractivo, pero ¿qué sucede con otros ambientes de virtualización? Más adelante se comparará esta estrategia de virtualización con la creación de máquinas virtuales, tan usadas en los ordenadores hoy en día. Para no ahondar mucho más en el tema e ir paso a paso se estudia lo que es Cloud Computing para entender el resto de los conceptos.

2.2. Cloud computing

Cloud computing, corresponde a un área del campo de las tecnologías de la información. Su impacto ha cambiado el punto de vista del cómputo, pues permite utilizar hardware o software como un servicio o recurso, lo que permite a diversas organizaciones hacer uso de los recursos virtuales que diversos proveedores entregan.
La NIST define a cloud computing como un modelo para acceder a recursos (redes, servidores, almacenamiento, aplicaciones y servicios) de manera conveniente y a demanda. IBM por otra parte define que cloud computing es la entrega de recursos bajo demanda (todo lo que pueda ser entregado por un datacenter) a través de internet y estos deben ser altamente escalables y flexibles bajo método de pago por uso. Se observa con estas definiciones que cloud computing presenta diversas clases de servicios:

- Cómputo.
- Almacenamiento.
- Redes.
- Servicios computacionales a proveedores bajo demanda.
- Otros.

Es posible identificar las distintas clasificaciones de servicios que ofrece cloud computing:

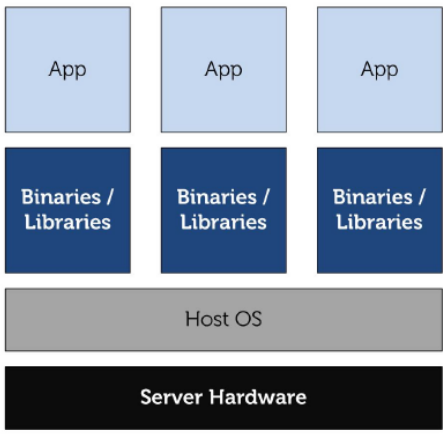
- IaaS (Ifrastucture as a Service): corresponde a la entrega de infraestructura al cliente, es decir servidores, almacenamiento, redes y cómputo, de manera que el cliente pueda comprar estos recursos en forma de subcontratación.

- PaaS (Platform as a Service): corresponde a la entrega de servicios y herramientas para el desarrollo de aplicaciones de manera que el cliente no deba comprar en software e infraestructura base.
- SaaS (Software as a Service): corresponde a la entrega de un software que se ejecuta en la infraestructura de la nube.

Cloud computing utiliza la virtualización de manera que otorgue flexibilidad de servicios a sus clientes. En la actualidad la mayoría de los servicios sobre nube como OpenStack y Amazon, utilizan virtualización basada en hipervisión (máquinas virtuales). Destaca entre el uso de esta tecnología la virtualización de servidores en datacenters, de manera que un administrador pueda crear una o más máquinas virtuales en un servidor, y lo vemos en plataformas de nube como las que se han mencionado. Sin embargo se pueden encontrar diversas tecnologías de virtualización entre las que se destaca los ***containers*** o **contenedores**, que es uno de los ejes de este trabajo. Google lleva la última década utilizando contenedores con lxc (*Linux containers*). Hoy en día diversas empresas hacen uso de containers para mejorar su proceso de *ContinuousDelivery*: Spotify, GILT, RackSpace, YELP, Ebay. Entonces tenemos dos entornos de virtualización recientemente mencionados: *Cointainer-based virtualization* e *Hypervision-based Virtualization.*

2.2.1. Cointainer-based virtualization

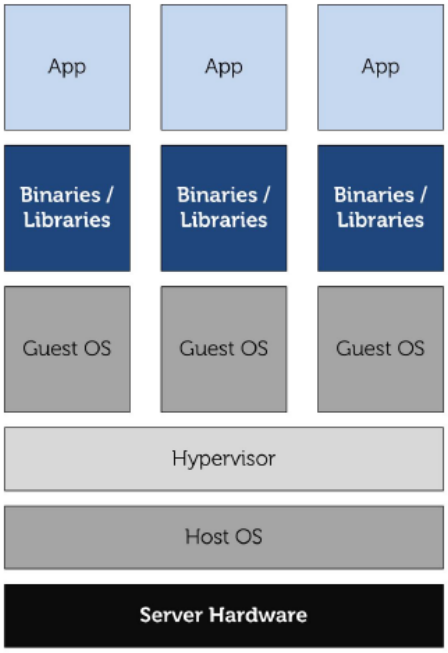
Es una virtualización liviana a nivel de software que usa el kernel del host para correr múltiples ambientes virtuales, los cuales se denominan contenedores. Las principales implementaciones de contenedores son Linux-VServer, OpenVZ, libcontainer y Linux Container (lxc). La ventaja de este entorno de virtualización es que para los n contenedores, no es necesario cargar un SO operativo a cada uno, pues hacen uso todos del kernel del SO del host. Para el SO operativo del host estos contenedores no son más que procesos que corren sobre el kernel. Estos ambientes aislados necesitan sólo de los binarios y librerías de otros SO que funcionen sobre el mismo kernel, para obtener características diferenciadoras. A continuación una figura que grafica lo previo.



Cointainer-based virtualization

2.2.2. Hypervision-based virtualization

En este caso se establece una o más máquinas virtuales sobre el SO del host, las cuales cuentan cada una con su propio SO y propio kernel. Se muestra a continuación una figura que presenta lo anterior.



Hypervision-based virtualization

Las diferencias apreciables entre las virtualizaciones presentadas nos señala que virtualización mediante containers, nos permite la implementación de ambientes aislados cargados sobre un host que comparte su kernel con cada uno de los ambientes, y además puede compartir archivos binarios y librerías con cada uno de ellos. Sucede además que al funcionar todos bajo el mismo kernel del host, hace el sistema más liviano y eficiente a la hora de realizar tareas.

Ahora que se sabe lo que son los contenedores o *containers*, se debe tener alguna herramienta que permita crear, correr, borrar aplicaciones de los entornos virtuales generados. La solución a este problema son los Dockers.

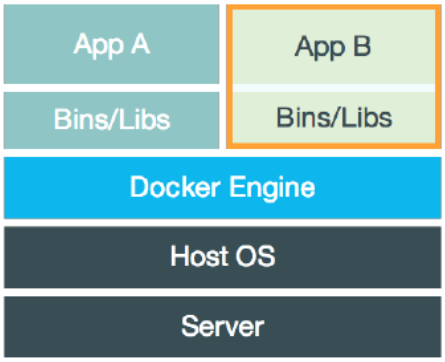
2.3. Dockers

Docker es un proyecto open-source que utiliza la tecnología de los containers para "construir, migrar y correr aplicaciones distribuidas". Simplifica el uso de containers pues provee una interfaz para crearlos y controlarlos, que es simple y segura. Permite también a los desarrolladores empaquetar y correr aplicaciones de manera sencilla e integrar con herramientas terceras que permiten administración y despliegue.

Dentro de Dockers podemos encontrar componentes como Docker Engine, Docker Images, Docker Containers, Docker Registries.

2.3.1. Docker Engine

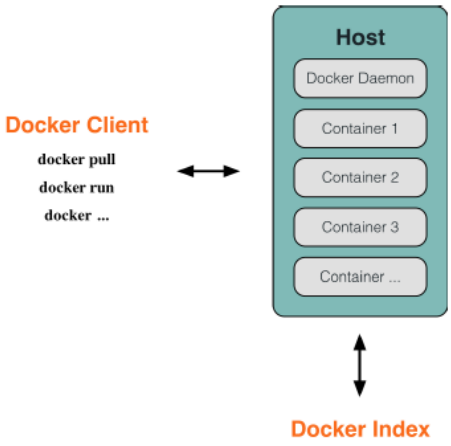
Corresponde a una herramienta liviana que permite el manejo y control de la virtualización mediante contenedores. La figura siguiente muestra la arquitectura de virtualización con containers usando Docker Engine.



Container-based virtualization, usando Docker

Esto dice que las aplicaciones corren sobre el servicio de Docker quien se encarga ejecutar y manejar aplicaciones en los contenedores. Docker utiliza una arquitectura de cliente-servidor y esto permitirá implementar una interfaz que permite interactuar con los contenedores, llamada Docker Client:

- Docker Client habla con Docker Daemon y este construye, maneja y corre los containers. Docker Client y Docker Daemon pueden correr en el mismo host o se puede conectar el cliente desde un host remoto. Observar la figura siguiente que ilustra el procedimiento bajo Docker Client.



Comunicación de Docker Client

2.3.2. Docker Images

Corresponde a una plantilla sólo de lectura, las que son usadas para crear Docker Containers. Si el usuario crea cambios en el contenedor, Docker añade una capa adicional con los cambios de la imagen. Por ejemplo se tiene una imagen de Debian y luego se agrega un paquete emacs y luego apache, las cuales se van apilando.
Docker utiliza un archivo Dockerfile para construir imágenes, donde cada uno de estos archivos corresponde a un script que ejecuta comandos y argumentos para realizar acciones en la imagen de base que permita crear una nueva imagen, de manera que puedan desplegarse aplicaciones desde el inicio hasta el final.

2.3.3. Docker Registries

Corresponden a repositorios, ya sean públicos o privados, donde los usuarios pueden compartir imágenes personalizadas o bases. DockerHub por ejemplo, es un registro público donde los usuarios pueden acceder al repositorio, donde tendrán la posibilidad de compartir y obtener imágenes cuya autenticidad e integridad pueden estar verificadas. Los registries son parte fundamental del sistema de Dockers pues impulsan el desarrollo de aplicaciones.

2.3.4. Docker Container

Es un ambiente virtual construido a partir de una imagen como ya se ha mencionado, tiene sistema operativo, archivos de usuario y metadatos. Cada imagen le indica al Dockers lo que tiene el contenedor, cuyo proceso se ejecuta cuando el contenedor es creado. Un ejemplo de creación de un contenedor usando Dockers se describe a continuación:

- Se ejecuta `docker run -i -t ubuntu /bin/bash`.
- Docker client le informa al Docker que debe correr un container.
- El comando será el init 0 del container en este caso en /bin/bash.
- Docker verifica la existencia de la imagen ubuntu y sino existe en el host, entonces Docker descarga de un registry ya sea privado o público. Si la imagen existe entonces crea el container.
- El container es creado en el filesystem y se añade una capa en modo read-write a la imagen.
- Crea la interfaz de red que permite que el Docker Container pueda hablar con el host a través del bridge (docker0).
- Asigna una ip del pool al container.
- Captura el output, input y errores.

Con todo lo que se sabe de Dockers es posible apreciar que falta superar un problema, que corresponde a la comunicación hacia Internet. ¿Quién se encarga de manejar la implementación de Dockers y encargarse de que todo funcione correctamente y bajo control? El "director de orquesta"de todo nuestro problema es Kubernetes.

2.4. Kubernetes

Kubernetes es un sistema open-source para el manejo de aplicaciones que corren en contenedores en diversos servidores, entregando mecanismos de despliegue, mantención y escalabilidad de estas aplicaciones. Es necesario entender algunos conceptos antes.

- Clusters: son recursos computacionales en el cual los contenedores son construidos. Kubernetes puede se utilizado en diversos ambientes como: Windows Azure, Debian, Ubuntu, RedHat y otros.
- Pods: es un grupo de Docker containers con volúmenes compartidos. Los pods son las unidades más pequeñas pueden ser creadas, programadas y manejadas por Kubernetes.
- Replication controllers: manejan el ciclo de vida de un pods. Estos se encargan de manejar el número de pods que se encuentran corriendo en un momento, creando y destruyendo los pods, como sea requerido.
- Services: proveen un único, estable nombre y dirección para un set de pods.
- Labels: son utilizados para organizar y seleccionar un grupo de objetos basado en pares del tipo key:value. Labels son fundamental para que los services y replications controllers, para obtener las listas de los servidores a donde el tráfico debe pasar, estas listas son seleccionadas en base a los labels que son dados.

Un cluster de Kubernetes contiene nodos agentes llamado kubelet y componentes de tipo master (maestro) como API, scheduler, etc.

2.4.1. Tipos de hosts

- Kubernetes Master: La unidad controladora de un cluster de Kubernetes es llamada master. El master es el punto donde se otorga a los servicios de cluster información de todo los nodos, para esto utiliza los siguientes componentes.

- Etcd: es uno de los fundamentales componentes, Kubernetes es dependiente de la disponibilidad de este servicio. El proyecto etcd desarrollado por CoreOS es un sistema liviano de distributed key-value store (almacenamiento distribuido de valor/llave) que puede ser distribuido en múltiples nodos. Kubernetes utiliza etcd para guardar la configuración y metadata que puede ser utilizada por cualquier nodo en el cluster.
- API server: es el punto de principal de administración del cluster, API server entrega acceso a Kubernetes API. Esto permite un usuario configurar a los miembros de Kubernetes y también es responsable que etcd y que las características de los servicios de los containers desplegados sean coherentes. API server permite que diferentes herramientas y librerías puedan comunicarse de manera más fácil. Un cliente llamado kubecfg permite comunicarse con el master desde otro nodo o un computador local.
- Controller Manager Server: es un servicio que es usado para manejar el proceso de replicación definido por las tareas de replicación. Los detalles de estas operaciones son escritos en el etcd, donde el controller manager observa por cambios. Cuando un cambio es detectado, el controller manager lee la nueva información e ejecuta el proceso de replicación hasta alcanzar el estado deseado.
- Scheduler Server es el servicio que asigna los pods a los nodos. Es usado para leer los requisitos del pod, analizar el ambiente del cluster y seleccionar los nodos aceptables. El scheduler es el responsable de monitorear la utilización recursos en cada host para asegurar que los pods no sobrepasen los recursos disponibles.

- Kubernetes node: Es el nodo donde se encuentran los servicios necesarios para que los containers puedan correr y sean manejados por el nodo maestro. Cada nodo utiliza Docker que se encarga del manejo de imágenes y la creación del container y sus componentes asociados.

  - Kubelet recopila las instrucciones o cambios de estado y los ejecuta en el host. Realiza el manejo de los pods y con esto sus containers, imágenes y volúmenes.
  - Kube-Proxy: este mecanismo observa Kubernetes master por alguna adición o eliminación de Service y Endpoints. Para cada Service este abre un puerto (que se elige al azar) en el nodo local. Cualquier conexión que se realice a ese puerto es proxied (redirrecionada a través de un proxy) a uno de los pods correspondientes. La selección del pod es basada a la afinidad SessionAffinity del Service. Por último, este instala reglas de firewall (iptables) las cuales captura el tráfico a la IP y puerto del servicio correspodiente.

2.5. Redes en kubernetes

En Kubernetes es importante que los nodos tenga acceso a la internet para poder ofrecer sus servicios y una red entre nodos para mantener la comunicación y la compartición de recursos y datos entre ellos, para una máquina virtual esta tarea es sencilla ya que ésta ve su propio kernel con sus propios puertos, pero para un cluster de contenedores que comparten un kernel la tarea se complica. Para ello existen dos elementos correspondiente a cada problema: para el acceso a red de los contenedores docker tiene su propia solución y para la comunicación entre nodos kubernetes utiliza a flannel.

2.5.1. Docker0

Para el caso del acceso a red de los contenedores se crea una subred virtual con un segmento privado administrado por una interfaz virtual que sirve de puente(bridge. Esta interfaz es creada al momento de instalar docker y es llamada docker0. Los contenedores utilizan esta interfaz como su gateway y el demonio de docker se encarga de suscribir a los contenedores a esta red. Esta interfaz hace nateo con ayuda del firewall (FirewallD). Luego cada contenedor tiene un interfaz virtual que le corresponde una ip address del segmento de la subred por loq ue cada contenedor tiene su propia dirección en IPv4. Esto se puede observar en el esquema de la figura 1

2.5.2. Flannel

Flannel es una red de sobre capa", permite manejar subredes grandes, en este caso, permite que las distintas subredes entre los nodos se comuniquen entre si, para lograr esto actúa como una VXLAN, una virtual extended

LAN, cuya diferencia con las VLAN es que crea un número identificador de red que es agregado al header de los paquetes UDP, esta información, como la dirección original de los nodos y sus estatus, entre otra información, es administrada por el servicio etcd que corre en el nodo central. Cada nodo se suscribe a etcd y sabe el estado de los demás a través de él. Una vez que las redes están identificadas los participantes de la red pueden navegar en esta red a través de su interfaz virtual, cada nodo utiliza una interfaz virtual llamada flannel.1 que administra el segmento de red de cada subnet y es utilizado como gateway para la comunicación a través de la infraestructura ofrecida por flannel. El redireccionamiento es a través del firewall, por lo que es importante que en cada nodo, y en especial en el nodo central, el firewall este funcionando. El esquema de la figura 6 muestra el funcionamiento de flannel.

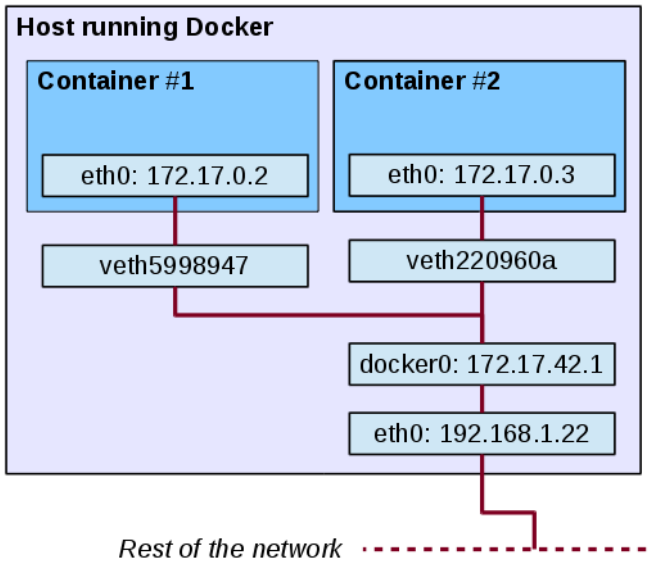


Figura 1: Esquema de redes de docker.

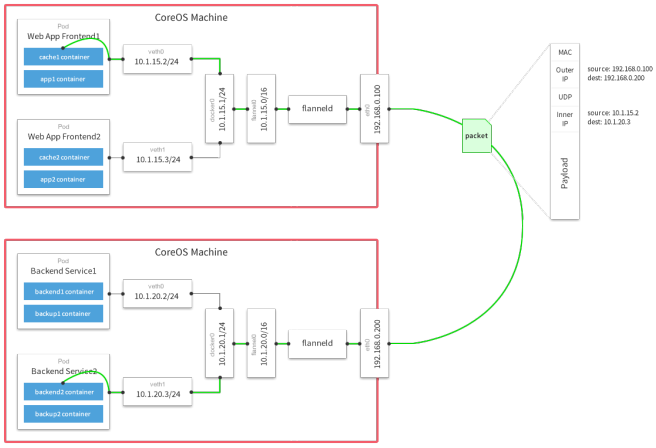


Figura 2: Esquema de redes con flannel.

2.6. Resultados experimentales

Para la fase de experimentación se utilizó la infraestructura ofrecida por el laboratorio de computadores del departamento de informática (LabComp) en donde un piloto del servicio Kubernet corre con 3 nodos. Este sistema piloto fue implementado por el alumno memorista Maximiliano Osorio de último año de informática. Kubernetes corre en 3 computadores: Playancha, Larraín y Concepción, los tres están conectados a una subred administrada por el DI, la 10.9.0.0. Playancha se utilizó como nodo central y en Larraín como en Concepción se creó un pod de un único contenedor.

El experimento concistió de 3 partes: verificar la salida a internet de cada contenedor, verificar la comunicación de cada nodo a través de las subredes y la captura de paquetes en las interfaces virtuales docker0 y flannel.1. El esquema del experimento se observa en la figura 6.

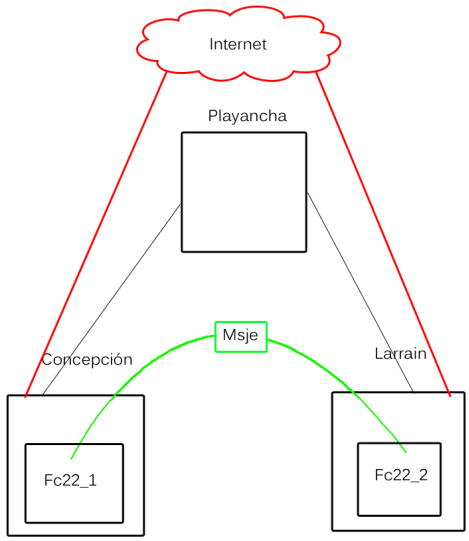


Figura 3: Esquema del experimento.

2.6.1. Comunicación con internet

En Larraín y Concepción se creó un contenedor de docker con la instalación más básica de fedora 22, se procedió a instalar en cada contenedor las herramientas necesarias: iproute, iputils y traceroute, para permitir hacer ping y traceroute. Una vez obtenido los permisos para utilizar ping se hizo ping al dns secundario de google: 8.8.4.4, luego se le hizo un traceroute a este para verificar que se ocupaba la interfaz docker0. Los resultados del traceroute se observan en la figura 4 lo marcado en rojo pertenece al ruteo que se hace desde el inicio del traceo hasta que sale de la infraestructura del departamento de informática, en ello se ve que ocupa a docker0 correspondiente a la ip del gateway de la subred 172.16.27.1.

2.6.2. Comunicación entre nodos

Para probar la red de flannel se utilizó los mismos contenedores creados en los nodos, se obtuvo la dirección ip de cada uno correspondiente a la subred mediante el comando ïp a". Posteriormente cada contenedor de los nodos hizo ping a la dirección de subred del otro contenedor. Una vez hecho esto se hizo un traceroute en cada contenedor a la dirección del contrario. La obtención de las direcciones ip se ilustra en la figura 5 y el traceroute del

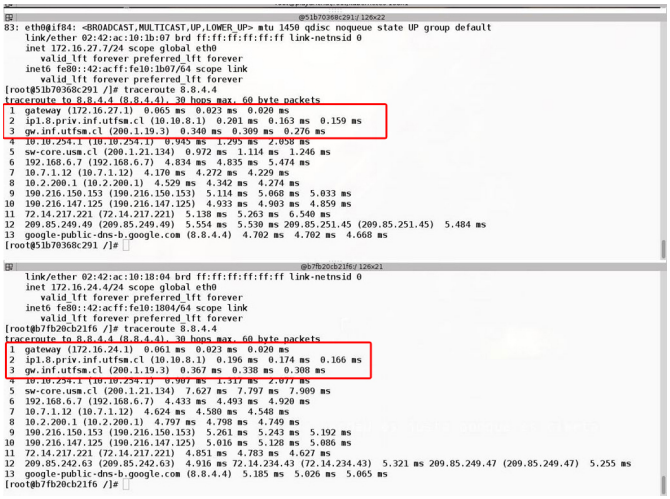


Figura 4: traceroute de los contenedores a google.cl.

contenedor 1 al contenedor 2 se ve en la figura **??**. En rojo se enfatiza que se están en dos contenedores distintos en 2 nodos distintos y se muestra la dirección de la subred de cada uno.

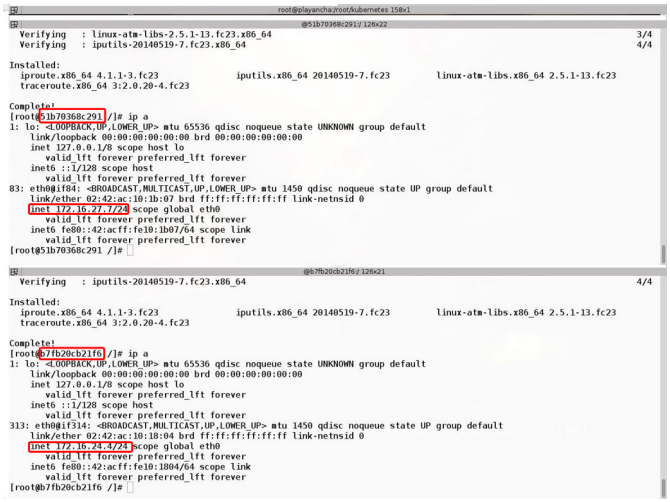


Figura 5: dirección ip de los contenedores.

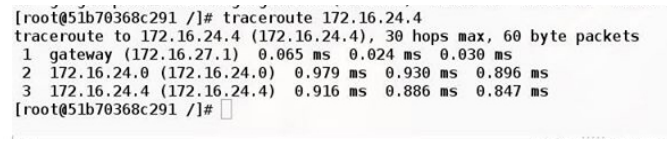


Figura 6: traceroute de un contenedor a otro.

2.6.3. Wireshark

Para conocer el tráfico de cada interfaz virtual se procedió a instalar wireshark en un contenedor diferente en Larraín y Concepción y se utilizó tshark, que es su versión para ejecución en terminal de linux. Primero tshark

se puso a escuchar en la interfaz de docker0 de uno de los contenedores, a continuación se hizo ping a google.cl. Esto se observa en la figura 7, en rojo el ping y su resultado en tshark.

Por último se inició una captura de tshark en la interfaz flannel.1 de uno de los contenedor mientras se le hacía ping y traceroute a través del otro observado en la figura 8, en rojo el ping con su resultado de captura de paquetes y en verde el traceroute con su respectivo resultado en tshark.

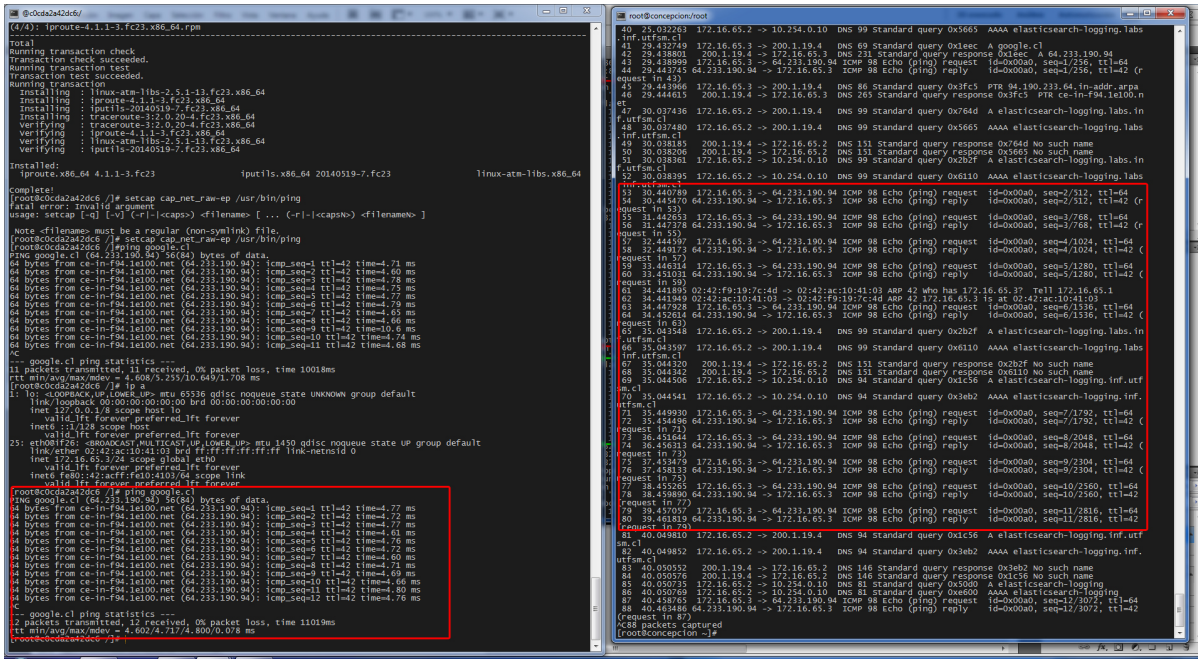


Figura 7: ping a google y captura de paquetes en docker0.

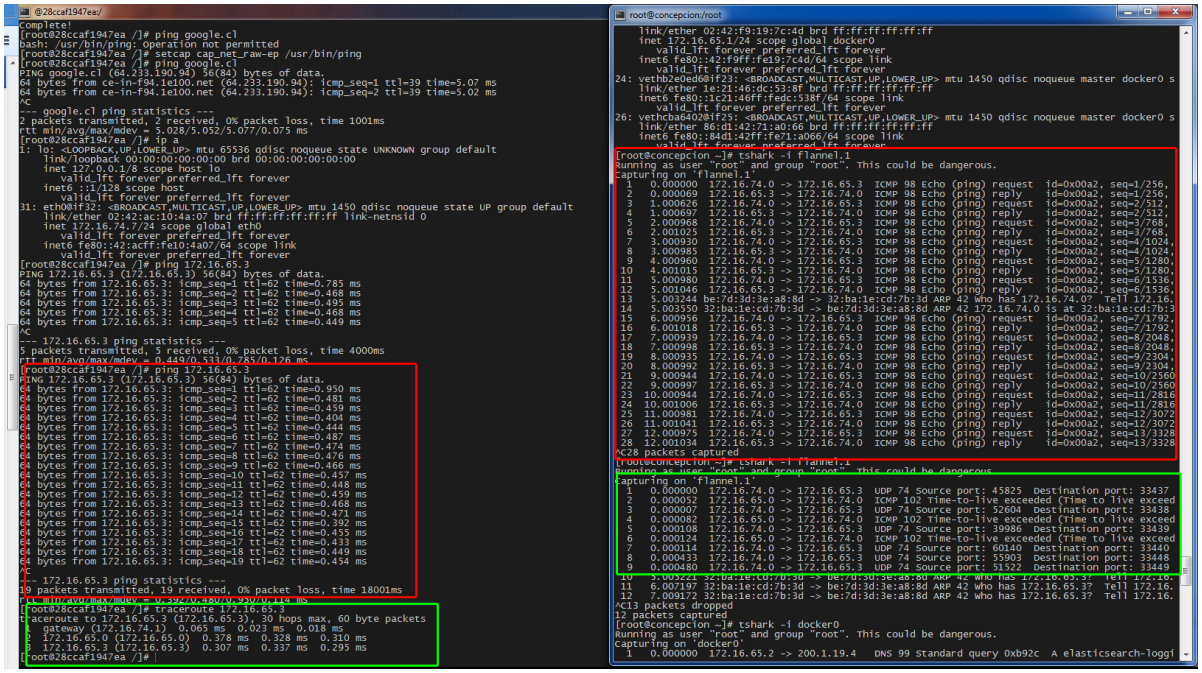


Figura 8: ping y traceroute entre conteendores y captura de paquetes en flannel.1 .

3. Conclusión

En estos tiempos las aplicaciones informáticas requieren plataformas que escalen y que puedan responder a una demanda masiva, es por ello que el uso de plataformas de cloud computing como Kubernetes que ahorran recursos redundantes es una buena opción y es parte de un paradigma nuevo en donde todo se distribuye y sólo se usan los recursos necesarios cuando son estrictamente requeridos. En este contexto es importante que los nodos distribuidos estén comunicados mediante una infraestructura ya presente como las redes de internet. La solución que da Kubernetes para la necesidad de conectividad de sus nodos y contenedores es efectiva y logra satisfacer los requerimientos.

Observando más de cerca esta infraestructura ayuda a aprovechar los recursos ya existentes ahorrando hardware para el proceso de comunicación al virtualizar servicios que deberían hacer dispositivos como un router o un switch, además de resolver el problema de la imposibilidad de una estructura física en una arquitectura lógica como lo son los cluster de contenedores. El sistemas es inteligente en el hecho que es tolerante al dinamismo de la propia red, de forma autónoma se va modificando a si mismo para el cambio en los contenedores, a la aparición o desaparición de estos y a cambios en los mismos nodos al ser una estructura en donde cada elemento entrante o saliente se anuncia frente a los demás, junto con ello automáticamente los demonios y servicios crean las condiciones y estructuras necesarias para que contenedor y nodo tengan red además de un monitoreo constante de que esto así sea.

Las desventajas de esta arquitectura es que Kubernetes depende de recursos ya existente por lo cual una infraestructura inadecuada tiende a saturarse y por otro lado hace más crítico servicios presente en los kernel de los hipervisores, por ejemplo, si el firewall llegase a tener un problema y se cayera arrastra con él a toda la red del nodo y a toda la red del cluster si se trata del nodo central. Cabe destacar que el servicio entero sólo es compatibles con sistemas basados en el kernel de Linux por lo cual su operación es restrigida, sin embargo el manejo de servicios de cloud computing en general estan orientadas para plataformas Linux.

Respecto al práctico con el ejercicio de laboratorio se pudo enfrentar a este desafío basados en un trabajo existente, la exploración de la red tuvo sus problemas que se fueron resolviendo a media que se aprendía más de la arquitectura de Docker y e Kubernetes con Flannel, para el funcionamiento de la red se requieren de varios servicios y se debió de aprender de ellos además del manejo propio de los nodos. Una vez superado estos desafíos los resultados fueron satisfactorios y se pudo corroborar el funcionamiento de las redes entre los contenedores.

4. Referencias

- Evaluación de kubernetes un sistema de orquestación de Docker containers para computación en la nube, Maximiliano Osorio, memorista Ing. Civil Informática, UTFSM.
- https://coreos.com/flannel/docs/latest/flannel-config.html
- https://docs.docker.com/engine/userguide/networking/dockernetworks/
- http://blog.shippable.com/docker-overlay-network-using-flannel