INGENIERÍA DE SERVIDORES (2016-2017)
DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS
UNIVERSIDAD DE GRANADA

Twitter: Apache Mesos, Finagle y Linkerd

Resumen

La gestión de carga de trabajo sobre un clúster de servidores es una tarea complicada si se quiere realizar a mano. Por ello, Twitter usa Apache Mesos, Finagle y Linkerd para repartir los procesos entre las distintas máquinas de las que dispone y establecer una buena comunicación entre ellas, evitando así fallos de sobrecarga e interrupciones debidas a la pobre interacción entre los componentes que actúan en presencia de errores.
Apache Mesos se encarga de abstraer los recursos de un cluster virtual o físico y distribuir entre los nodos la carga del sistema.
Finagle se ocupa de recibir y enviar las peticiones entre los servidores y hacer que, a pesar de los distintos lenguajes de las aplicaciones, éstos puedan entenderse.
Linkerd surge para solucionar los problemas de grandes compañías con las comunicaciones entre distintos servicios. Posee potentes funcionalidades para reducir los costes de sincronización y diferencias en las latencias.

Introducción.

Twitter es una red social de "microblogging" creada por Jack Dorsey[1] en 2006 en la cual los usuarios pueden publicar mensajes de texto plano de corta longitud (un máximo de 140 caracteres), imágenes o vídeos que se muestran a lo largo de la página principal de cada cuenta de usuario.

Los usuarios pueden seguir a otras cuentas para estar al tanto de los tweets que publiquen y que éstos se añadan a su "timeline", que es la página principal de Twitter, en la cual aparecen los mensajes de todos los usuarios a los que sigues.

Desde que en Twitter se lanzase en julio de 2006, la red ha ido ganando popularidad en todo el mundo, estimándose que hoy día tiene más de 300 millones de usuarios activos, generando 65 millones de tweets al día y resolviendo más de 800000 peticiones de búsqueda diarias. [2]

En el primer trimestre de 2010 [3], Twitter dejó de utilizar servidores de terceros para empezar a desarrollar su propia infraestructura de red internamente, comenzando a iterar sobre distintos diseños de red, hardware y proveedores.

Al principio, solo disponía de las capas *front-end* y *back-end*, desarrolladas en Ruby de una sola pieza. En 2010 existían varias decenas de millones de usuarios pero tan solo 100 máquinas actuando como *host*. Cuando sucedía algún evento importante la gente publicaba muchos tweets, provocando que la red se saturase; por ejemplo, el mundial de Sudáfrica de 2010 produjo caídas del sistema y "*Fail whales*" (referente al error 404, página no encontrada), debido a la gran cantidad de tweets que se intentaban enviar a la vez cuando un equipo anotaba un gol, ganaba un partido,etc.

Decidieron por tanto ampliar los servidores y "especializarlos" en tareas, pero ello implicaba tener que gestionar las cargas de trabajo previendo qué servicio iba a ser más utilizado en cada momento. Fue entonces, el momento en el que empezaron a usar Mesos (desarrollado más tarde por Apache y pasándose a llamar Apache Mesos) y a desarrollar Finagle para automatizar el *back-end*, añadiéndose un poco más tarde a ellos Linkerd.[4]

1. Apache Mesos.

Mesos es un software de código abierto dedicado a la administración de clústers de servidores desarrollado por estudiantes del doctorado de la Universidad de California, Berkeley. El proyecto fue presentado en 2009 en HotCloud '09, un evento de conferencias de temas de actualidad sobre *Cloud Computing*. [5]

En 2010, Twitter comenzó a desarrollar su propio *framework* de Mesos, Apache Aurora, del que hablaremos más adelante. Por otra parte, el 27 de julio de 2016, la corporación "Apache Software Foundation" anunció su primera versión de Mesos, Apache Mesos.

Apache Mesos está realizado con una ideología parecida a la de Linux en el aspecto del procesamiento de tareas, con una gran diferencia: Mesos aporta abstracción sobre un conjunto de máquinas en un clúster. [4]

El clúster que administra Mesos está dividido en pequeños agentes. Mesos consiste de varios agentes maestros que administran "demonios" esclavos, corriendo cada uno de ellos en una máquina del clúster, y diferentes frameworks que mandan trabajos para ejecutar en dichos esclavos. Cada framework ejecutándose en Mesos consiste de dos componentes: un planificador que se comunica con el maestro para que se le ofrezcan recursos, y un proceso ejecutor que se inicia en los nodos del agente para ejecutar las tareas requeridas por el framework. El agente maestro oferta trabajos recibidos del framework, junto a los recursos necesarios para realizar su ejecución, a cada uno de los esclavos por medio de mensajes. [7]

En la imagen 1.1 podemos ver un ejemplo de como un framework se planifica para ejecutar una tarea. En el punto 1, el esclavo número uno le dice al maestro que dispone de 4 CPUs y 4 GBs de memoria RAM para ejecutar cualquier tarea. Entonces, el maestro activa el módulo de asignación, mediante el que se le comunica que el framework número uno es la mejor opción para ofrecerle todos los recursos. En el punto 2, el maestro manda la oferta de recursos al framework número uno, describiendo estos recursos. En el punto 3, el planificador del framework responde al maestro con la información sobre las tareas que quiere ejecutar en el esclavo (una con 2 CPUs y 1 GB de RAM, y otra con 1 CPU y 2 GBs de RAM). Por último, en el punto 4, el maestro manda las tareas al esclavo, el cual asigna los recur-

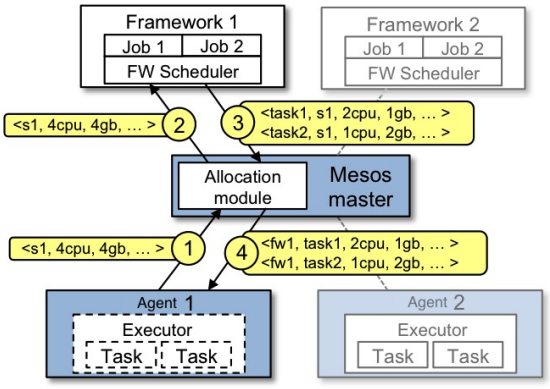

Figura 1.1: Ejemplo de oferta de recursos en Apache Mesos.

sos apropiados para el ejecutor del framework, que se encargara de ejecutar las dos tareas. Hay que notar que durante la ejecución de los procesos quedan libres en el esclavo 1 CPU y 1 GB de RAM, por lo que el módulo de asignación activado por el maestro podría ofrecer esos recursos al framework número 2; es más, en cuanto las tareas finalicen en el esclavo, el proceso de oferta se repetirá y así sucesivamente. [6]

1.1. Apache Aurora.

Como decíamos anteriormente, Twitter trabaja con un framework propio de Mesos llamado Apache Aurora. El desarrollo de Aurora comenzó en 2010, cuando Bill Farner, un ingeniero de Twitter, inició un proyecto para simplificar las operaciones de los muchos servicios que componían la arquitectura de Twitter. Con el paso del tiempo, un creciente equipo de ingenieros de Twitter comenzaron a contribuir directamente en el proyecto Aurora.

A finales de 2013, Twitter compartió su trabajo, liberando su código en la comunidad *open source* "Apache Incubator".

Apache Aurora aprovecha el administrador de clúster de Apache Mesos, que proporciona información sobre el estado actual de todos los servidores. Aurora utiliza ese conocimiento para tomar decisiones de planificación sobre los nodos del clúster; por ejemplo, cuando una máquina experimenta un fallo, Aurora "reprograma" automáticamente los servicios ejecutados anteriormente en una máquina exenta de fallos para mantener todas las tareas en funcionamiento. [8]

2. Finagle.

Renderizar hasta la página web más simple de Twitter requiere la colaboración de distintos servicios web, "hablando" cada uno en un protocolo diferente. Por ejemplo, para renderizar la página principal la aplicación envía solicitudes al *Social Graph Services*, el software *Memcached*, bases de datos y muchos otros servicios de red, donde cada uno de ellos usa un protocolo distinto. Adicionalmente, la mayoría de estos servicios están interactuando a la vez con otros servicios, pues actúan como servidor y cliente al mismo tiempo.

En estos sistemas, un frecuente caso de interrupciones es la pobre interacción entre los componentes que actúan en presencia de errores. Los errores más comunes son bloqueos del *host* y diferentes latencias entre servicio y cliente. Estos errores pueden provocar fallos en cascada en el sistema, causando largas colas para la ejecución de los servicios, reducción de rendimiento en el cliente por conexiones TCP en muy corto tiempo, agotamiento de la memoria o, en el peor de los casos, una "*Fail Whale*" (referente al error 404, página no encontrada).[9]

Por tanto, Twitter decide desarrollar Finagle para solucionar estos problemas. Finagle es un sistema de llamada de procedimiento remoto asíncrono para servidores de alta concurrencia, indiferente a protocolos de distintas plataformas [11]. Hace fácil escribir clientes y servidores robustos en Java, Scala u otros lenguajes aceptados por JVM (debido a que este software está implementado para máquinas virtuales de Java).

Finagle proporciona una implementación robusta sobre:

- *Connection Pool* (agrupamiento de conexiones) para mantener conexiones HTTP abiertas y combatir así la saturación de las conexiones TCP.
- Detectores de fallos para identificar las máquinas lentas o bloqueadas.
- Estrategias de conmutación por error para dirigir el tráfico fuera de las máquinas con problemas.
- Estrategias de balance de carga.

- Técnicas *Back-pressure* para defender los servidores contra clientes abusivos y envío masivo de peticiones (como ataques DDoS).

Adicionalmente, Finagle facilita la tarea de desarrollar un servicio tal que:

- publique estadísticas estándar, *logs*, y reportes de excepción;
- soporte el rastreo distribuido a través de protocolos;
- use ZooKeeper para la administración de clústers.

[9]

Hoy día, Finagle está desplegado en producción en Twitter en diversos sistemas de servicio de back-end y front-end, incluyendo el rastreador de URLs y el Proxy HTTP. Básicamente, Finagle actúa en la recepción y el envío de mensajes entre los distintos servicios, sirviendo de lenguaje común entre ellos y capturando errores si los hubiese.

A continuación, vemos una imagen que representa la estructura interna de Finagle. [10]

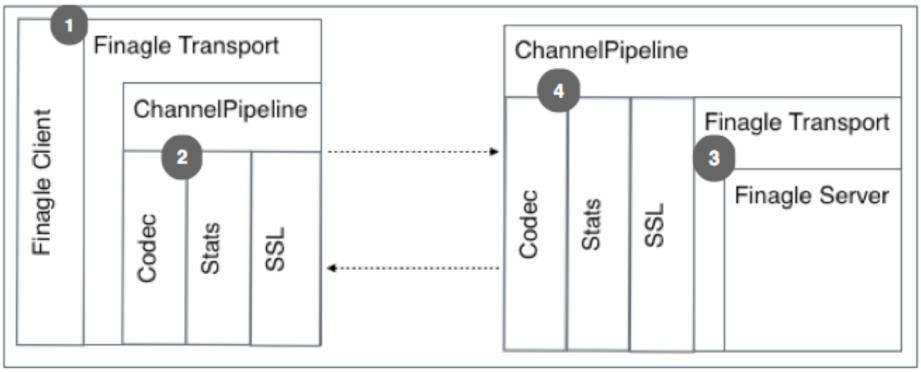

Figura 2.1: Representación gráfica de la estructura interna de Finagle.

Los elementos que aparecen marcados con números son los siguientes:

1) *Finagle Client*, el cual se encuentra sobre la capa del *Finagle Transport*, que se encarga de abstraer las aplicaciones lejos del usuario.
2) *Channel Pipeline* actual de la aplicación que contendrá todas la implementaciones del *Channel Handler*, el cual maneja los canales de comunicación y realiza el verdadero trabajo.
3) Servidor de Finagle, el cual es creado para cada conexión y proporciona el transporte para la lectura y escritura entre los clústers.
4) Manejadores de los canales de comunicación, que implementan la lógica de codificación/-decodificación de protocolos, estadísticas a nivel de conexión, manejo de TLS, etc.

3. Linkerd.

Linkerd es un “comunicador de servicios” escalable de código abierto para aplicaciones nativas en la nube [14], diseñado para hacerlas seguras y sanas al incluir servicios para gestión del balance de carga, manejo de errores, instrumentación y acceso a todas las comunicaciones entre servicios.

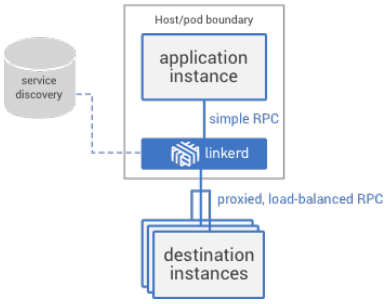

Figura 3.1: Configuración típica de Linkerd [13].

Linkerd actúa como un proxy HTTP, gRPC, thrift, etc., y puede ser introducido en aplicaciones ya existentes con un mínimo de configuración, a pesar del lenguaje en el que estén escritas. Además, éste se encuentra desarrollado por encima de Finagle. [15]

El software Linkerd fue desarrollado para solucionar los problemas que se encontraban al operar sobre grandes sistemas de producción en compañías como Twitter, Yahoo, Google y Microsoft. Al contrario de lo que se suele pensar, la fuente de la conducta mas compleja, sorprendente y emergente no son los servicios en sí mismos, si no la comunicación entre ellos. Linkerd aborda estos problemas no sólo controlando la mecánica de esta comunicación sino incluyendo una capa de abstracción sobre de ella. [14]

Proporcionando una capa uniforme y consistente de instrumentación, y control a través de los servicios, Linkerd libera a los propietarios de los servicios para elegir cual es el lenguaje más apropiado para sus servicios; y al desacoplar la mecánica de comunicación del código de la aplicación, Linkerd permite visibilidad y control sobre estos mecanismos sin cambiar la aplicación misma.

Linkerd puede ser visto como un enlazador dinámico para microservicios. En un SO, el enlazador dinámico se encarga de enlazar las bibliotecas de código a un determinado programa cuando se ejecuta; Linkerd lo hace de forma análoga para microservicios: coge el nombre de un servicio y de la llamada a hacer sobre ese servicio (HTTP, gRPC...) y realiza el trabajo requerido para hacer la llamada exitosa. [12]

Hoy día, compañías a lo largo del mundo usan Linkerd en producción para “impulsar el corazón” de su infraestructura de software. Linkerd tiene en cuenta la complejidad de las partes propensas a errores de la comunicación entre servicios, haciendo el código de la aplicación escalable, eficiente y flexible.

A continuación procederemos a la instalación de Apache Mesos, así como su la ejecución de procesos en él.

4. Instalación de Apache Mesos.

La documentación de Apache [16] recomienda usar tres máquinas como *master* (maestras). Para esta presentación de Apache Mesos vamos a realizar un sistema con solo una máquina maestra y dos máquinas esclavas (agentes). La instalación para más agentes o más maestros es similar a la aquí expuesta[1].

4.1. Descarga e instalación de paquetes.

Partimos de tres máquinas virtuales con Ubuntu 14.04 (*Trusty Tahr*) instalado con una imagen limpia. Primero tendremos que añadir los repositorios de `mesosphere` para poder descargar el software de Apache Mesos.

```
echo "deb http://repos.mesosphere.io/ubuntu trusty main" |
        sudo tee /etc/apt/sources.list.d/mesosphere.list
```

[18]

[1] El método de descarga, instalación y configuración de Mesos ha sido obtenido mediante la información disponible en la web [17].

En las máquinas maestras es necesario instalar Java, ya que Mesos se construye sobre una máquina de Java, y el paquete `mesosphere`. En las máquinas que actúan como agentes solo es necesario el paquete `mesos`, más ligero y con el software necesario para hacer las funciones de agente.

```
sudo apt-get install mesosphere
sudo apt-get install mesos
```

4.2. Configuración de ZooKeeper.

ZooKeeper es un administrador de procesamiento distribuido desarrollado por "Apache Software Foundation" y usado por Apache Mesos por su simplicidad y seguridad [19]. Viene incluido en los paquetes instalados anteriormente, aunque es necesario configurarlo.

Lo primero que tenemos que hacer es indicar la dirección IP de las máquinas maestras. Puesto que todas las máquinas enviarán mensajes a las máquinas maestras, tenemos que administrarlo en todas. En el archivo */etc/mesos/zk* escribiremos la línea:

`zk://[direccion_ip]:2181/mesos`

O en caso de varias maestras:

`zk://[direccion_ip_1]:2181,[direccion_ip_2]:2181,[...]/mesos`

Donde el valor `2181` es el puerto por el que nos comunicamos con ZooKeeper, el cual se aconseja no cambiar, pues ZooKeeper está programado para escuchar de ese puerto por defecto, y en caso de modificarlo tendríamos que modificar el código también.

Además, las máquinas maestras deben tener especificado un identificador único para cada componente. En nuestro, al disponer de solo una máquina, el único identificador será 1. Modificaremos por tanto el archivo */etc/zookeeper/conf/myid* y escribiremos únicamente un 1. Para varias máquinas maestras las numeraremos en orden ascendente sin repetir identificador, hasta un máximo de 255.

Por último, hay que enlazar cada identificador con su IP correspondiente. Para ello debemos modificar el archivo */etc/zookeeper/conf/zoo.cfg* y definir cada uno de los servidores maestros, en nuestro caso solo uno:
`server.1=[direccion_ip]:2888:3888`

4.3. Configuración de máquinas maestras.

Existe una variable esencial en la configuración de las máquinas maestras, el *quorum* [20]. Cada uno de los maestros tiene un voto cuando está activo, y para que una acción tenga validez, tiene que haber al menos un número de votos igual al valor de *quorum*. Es posible modificar el número de maestros que tienen que estar presentes en una decisión para aceptarla en */etc/mesos-master/quorum*. Por defecto viene a 1, es decir, solo es necesario que una máquina maestra esté presente. En nuestro caso, al sólo disponer de un *master*, el único valor posible es 1. Se recomienda que se establezca este número como la mitad de las máquinas totales. En caso de modificar el valor, hacerlo en todos las máquinas maestras por igual para no alterar la sincronización de éstas.

Para cada maestro necesitamos indicar la IP propia en dos archivos de la jerarquía de Mesos, `/etc/mesos-master/hostname` y `/etc/mesos-master/ip`. Basta con escribir la dirección IP en ambos archivos, sin ningún carácter más en la línea.

Para administrar el inicio del sistema y el reparto de tareas necesitaremos el framework Marathon [21] en cada servidor maestro[2]. Una vez más, necesitamos escribir la IP de la máquina en un archivo de configuración; esta vez de Marathon:

```
sudo mkdir -p /etc/marathon/conf
sudo cp /etc/mesos-master/hostname /etc/marathon/conf
```

Necesitamos indicar a Marathon los módulos de ZooKeeper que tiene a su disposición para administrar trabajos. Es la misma cadena que la expresada para identificar la dirección IP de las máquinas maestras en la sección 4.2, luego copiamos el archivo modificado anteriormente mediante el siguiente comando:

```
sudo cp /etc/mesos/zk /etc/marathon/conf/master
```

Esto permite que podamos conectar con Mesos, pero también queremos que tenga su propio módulo de ZooKeeper, luego copiamos:

```
sudo cp /etc/marathon/conf/master /etc/marathon/conf/zk
```

Notar que tenemos que cambiar la terminación de `/mesos` por `/marathon`; para ello, usaremos un editor de texto cualquiera, por ejemplo, `nano`.

Para continuar, deshabilitaremos los procesos relacionados con los agentes, ya que los paquetes de Mesosphere contienen los servicios de maestro y esclavo, y solo queremos utilizar los primeros; no deshabilitarlos conllevaría que una misma máquina funcionase como agente y maestro a la vez. Detenemos los procesos y sobrescribimos los archivos de inicio de los servicios de agentes para asegurarnos que no se volverán a ejecutar.

```
sudo stop mesos-slave
echo manual | sudo tee /etc/init/mesos-slave.override
```

[18]
Para poner en funcionamiento nuestra arquitectura de control maestra hay que reiniciar varios servicios para actualizar los cambios realizados sobre los archivos de configuración. Usaremos los comandos:

```
sudo restart zookeeper
sudo start mesos-master
sudo start marathon
```

Este es un buen momento para comprobar qué resultados estamos obteniendo. Accediendo en cualquiera de las máquinas maestras al buscador y escribiendo `http://[ip\_del\_maestro]:5050` se nos mostrará una interfaz de nuestro clúster funcionando. Por ahora no aparece ninguna tarea, tenemos 0 recursos activos y 0 agentes en el clúster. No obstante, Apache Mesos ya está funcionando. Podremos visualizar la interfaz de Marathon en el puerto 8080 de forma similar.

[2]Notar que Twitter tiene una versión propia de inicio llamada Aurora, pero su instalación es mucho más compleja.

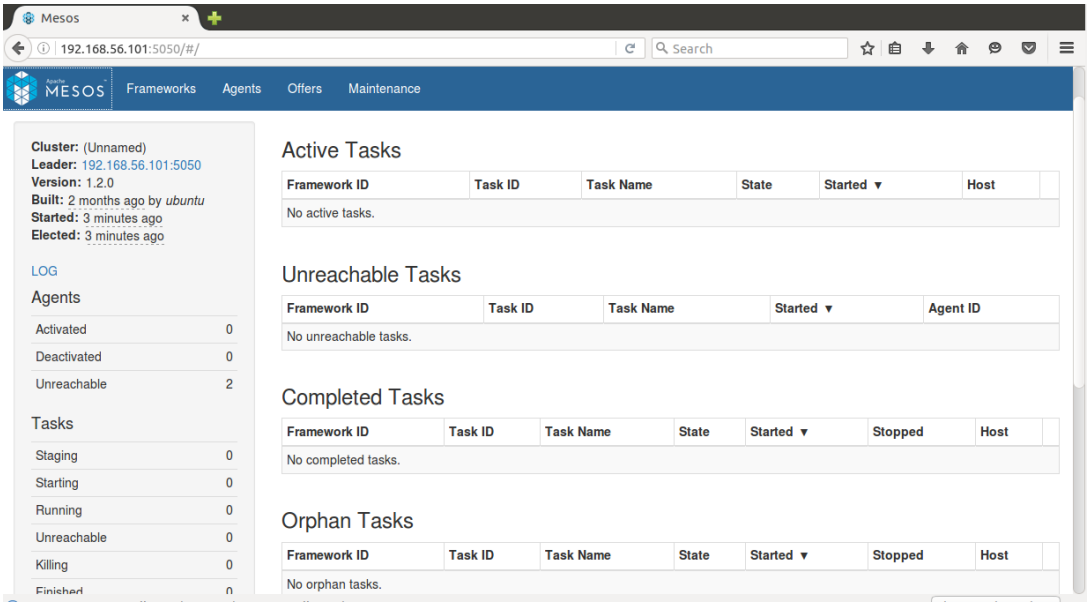

Figura 4.1: Comprobación del correcto funcionamiento de Apache Mesos[3].

4.4. Configuración de máquinas agente.

Ya que los agentes no necesitan instancias de ZooKeeper, desactivaremos los procesos de ZooKeeper de forma similar a la anterior.

```
sudo stop zookeeper \\
echo manual | sudo tee /etc/init/zookeeper.override
```

Igualmente desactivaremos los procesos de maestros.

```
echo manual | sudo tee /etc/init/mesos-master.override \\
sudo stop mesos-master
```

Ahora, al igual que realizamos en los maestros, indicamos a Mesos la dirección IP de cada una de nuestras máquinas esclavas. Habrá que indicarlas en dos archivos diferentes:

```
echo [direccion\_ip] | sudo tee /etc/mesos-slave/ip \\
sudo cp /etc/mesos-slave/ip /etc/mesos-slave/hostname
```

Ya podemos iniciar el agente y ver cómo se actualiza la información del cluster de Mesos.

```
sudo start mesos-slave
```

[3]En el apartado de Frameworks solo tenemos instalado Marathon y Chronos. Pueden encontrarse más en `http://mesos.apache.org/documentation/latest/frameworks/`.

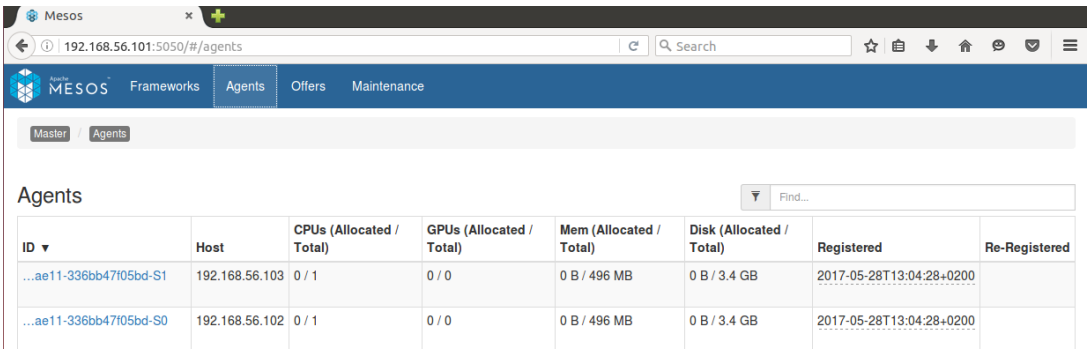

Figura 4.2: Visualización de Mesos ya configurado.

4.5. Lanzar aplicaciones con `curl`.

Desde otra máquina podemos enviar paquetes a Marathon mediante el comando `curl` [22], con la información del comando que queremos ejecutar; en nuestro caso, enviaremos los paquetes desde la máquina anfitriona.

Por ejemplo, creamos un archivo `ejemplo.json` como este:

```
{
     "id": "hello2",
     "cmd": "echo hello; sleep 10",
     "mem": 16,
     "cpus": 0.1,
     "instances": 1,
     "disk": 0.0,
     "ports": [0]
}
```

A continuación, usamos el comando `curl` como se muestra en la imagen. Estamos enviando el archivo `ejemplo.json` creado a la dirección `[direccion_ip_maestra]:8080`, es decir, al framework de Marathon activo en la máquina maestra. Automáticamente vemos aparecer una tarea con identificador `hello2` e IP `192.168.56.102` ejecutándose en el host; es decir, nuestro agente 1 estará ejecutándose en el host.

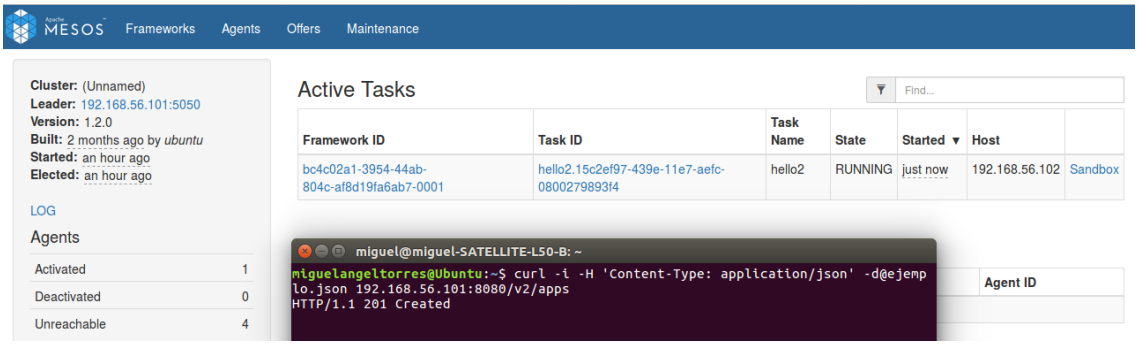

Figura 4.3: Ejemplo de ejecución de un programa en Apache Mesos.

5. Funcionamiento de Mesos, Finagle y Linkerd en Twitter. Conclusión final.

Imaginemos que disponemos de una cuenta de usuario de Twitter e iniciamos sesión en la página web. Lo primero que se nos mostrará será el *timeline*, en el cual se visualizarán los tweets

ordenados mediante el algoritmo de preferencia de Twitter, los *Trending Topics*, sugerencias de a quién seguir...

Para disponer de todas estas características, internamente, en los servidores de Twitter, habrá ocurrido algo similar a lo siguiente: uno de los maestros del clúster de servidores que administra Apache Mesos recibe una petición para mostrar el *timeline* de determinado usuario, y éste asigna la tarea a uno de los nodos esclavos de los que dispone, escogiendo la máquina de la que espere mejores resultados. Una vez asignada la tarea, Finagle interactuará con las bases de datos para obtener los tweets a mostrar, consultará los *Trending Topics*, etc. mientras que Linkerd establecerá y supervisará la comunicación entre los distintos servicios.

Por tanto, podemos concluir con que los 3 softwares presentados son herramientas muy útiles para controlar y administrar clústers de servidores de alta concurrencia, obteniendo un gran ahorro de tiempo sobre ellos al distribuir los servicios de manera eficiente.

Referencias

[1] JACK (@JACK), https://twitter.com/jack.

[2] TWITTER CRECE A PASO DE TORTUGA: LLEGA A 319 MILLONES DE USUARIOS AL MES, https://www.cnet.com/es/noticias/twitter-319-millones-usuarios-febrero-2017-q4-2016/.

[3] THE INFRASTRUCTURE BEHIND TWITTER: SCALE, https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale.html.

[4] FINAGLE, LINKERD, AND APACHE MESOS: MAGICAL OPERABILITY SPRINKLES FOR MICROSERVICES, https://www.linux.com/videos/finagle-linkerd-and-apache-mesos-magical-operability-sprinkles-microservices.

[5] MESOS: A PLATFORM FOR FINE-GRAINED RESOURCE SHARING IN THE DATA CENTER , https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center.

[6] MESOS: A PLATFORM FOR FINE-GRAINED RESOURCE SHARING IN THE DATA CENTER, http://people.csail.mit.edu/matei/papers/2011/nsdi_mesos.pdf.

[7] MESOS ARCHITECTURE, http://mesos.apache.org/documentation/latest/architecture/.

[8] ALL ABOUT APACHE AURORA, https://blog.twitter.com/engineering/en_us/a/2015/all-about-apache-aurora.html.

[9] FINAGLE: A PROTOCOL-AGNOSTIC RPC SYSTEM, https://blog.twitter.com/engineering/en_us/a/2011/finagle-a-protocol-agnostic-rpc-system.html.

[10] NETTY AT TWITTER WITH FINAGLET, https://blog.twitter.com/engineering/en_us/a/2014/netty-at-twitter-with-finagle.html.

[11] FINAGLE, https://twitter.github.io/finagle/guide/.

[12] FREQUENTLY ASKED QUESTIONS, https://linkerd.io/overview/faq/.

[13] HOW TO USE IT, https://linkerd.io/overview/how-to-use-it/.

[14] WHAT IS LINKERD, https://linkerd.io/overview/what-is-linkerd/.

[15] LINKERD/README.MD, https://github.com/linkerd/linkerd/blob/master/README.md.

[16] GETTING STARTED, http://mesos.apache.org/documentation/latest/getting-started/.

[17] HOW TO CONFIGURE A PRODUCTION-READY MESOSPHERE CLUSTER ON UBUNTU 14.04 , https://www.digitalocean.com/community/tutorials/how-to-configure-a-production-ready-mesosphere-cluster-on-ubuntu-14-04

[18] , TEE(1), http://man7.org/linux/man-pages/man1/tee.1.html.

[19] APACHE ZOOKEEPER, https://zookeeper.apache.org/.

[20] ZOOKEEPER GETTING STARTED GUIDE, https://zookeeper.apache.org/doc/r3.1.2/zookeeperStarted.html.

[21] OVERVIEW, https://mesosphere.github.io/marathon/.

[22] CURL(1), https://linux.die.net/man/1/curl.