UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Departamento de Sistemas Informáticos y Computación

Universitat Politècnica de València

Gestión Elástica de Clusters de Contenedores

Trabajo Fin de Máster

Máster Universitario en Computación Paralela y Distribuida

Autor: Yuriy Yatsyk

Tutor: Dr. Germán Moltó Martínez

Curso 2015/2016

Resumen

La virtualización basada en contenedores se ha extendido a gran velocidad gracias a *Docker*. Dos años después, tras liberar su código, ya es una plataforma soportada en *Amazon EC2*, *Google Cloud* y *Microsoft Azure*. Se trata de un sistema revolucionario que se encuentra en continua expansión combinando la facilidad de uso con la eficiencia. Ente sus últimos avances, permite usar una infraestructura física, virtual o mixta como un único ordenador lógico para desplegar servicios dentro de los contenedores. A continuación, se presentan y desarrollan los procesos de análisis, diseño e implementación un *cluster* de contenedores elástico, donde los nodos se aprovisionan y se terminan dinámicamente en función de las necesidades de cómputo. Esta implementación nos permite llevar un consumo responsable de la infraestructura al controlar el número de recursos que desplegamos. En el caso de *Cloud Computing* se ahorra en el coste de los nodos que se encuentran en marcha y en una granja de ordenadores se ahorra la energía eléctrica utilizada para el funcionamiento y la refrigeración.

Palabras clave: Cloud Computing, Docker, Contenedores, Elasticidad, Ahorro Energético

Tabla de contenidos

Tabla de ilustraciones

1. Introducción

En Informática, la virtualización consiste en crear a través de software una versión virtual de un recurso tecnológico, como puede ser un dispositivo de almacenamiento, hardware, un sistema operativo u otros recursos de red. Aporta beneficios significativos en el aprovechamiento hardware, en el tiempo de aprovisionamiento y en el tiempo de recuperación. Este término se utilizó por primera vez en 1960 y, desde entonces, han surgido distintos tipos de virtualización. Sin duda alguna, a día de hoy, la virtualización a nivel de sistema operativo, también conocida como virtualización basada en contenedores es la más popular.

Docker (1) es un proyecto de código abierto que automatiza el despliegue de servicios dentro de contenedores de software. Tuvo su lanzamiento en el año 2013 y actualmente lidera la virtualización basada en contenedores. Gracias a su marketing popularizó la virtualización a nivel de sistema operativo. A pesar de que esta técnica ya tenía mecanismos de virtualización desde el 1982 con la aparición de la operación *chroot* (2) en los sistemas *Unix* (3), nunca tuvo tanto reconocimiento como ahora. Dentro del ecosistema de *Docker* aparecen herramientas como *Docker Machine* (4) que posibilita despliegues de máquinas virtuales en las que es posible ejecutar contenedores y *Docker Swarm* (5) que permite desplegar un clúster sobre maquinas físicas o virtuales.

Este trabajo describe como dotar de elasticidad los *cluster* creados con *Docker Swarm* mediante la creación de un *plugin* para la herramienta CLUES (6) desarrollada por el Grupo de Grid y Computación de Altas Prestaciones (GRyCAP) del Instituto de Instrumentación para Imagen Molecular de la Universidad Politécnica de Valencia.

1.1. Objetivos y motivación

Docker Swarm soporta descubrimiento de nodos pero no existen mecanismos de elasticidad, por lo que se trata de un trabajo de investigación e innovación.

El trabajo describe como crear un *cluster* de contenedores elástico que permite terminar los nodos que no están siendo utilizados y desplegarlos automáticamente cuando exista una carga significativa de trabajo. Con esta técnica se aprovechará al máximo la infraestructura que componen todos los nodos y se ahorrará energía en el caso de desplegar un *cluster* de *Docker Swarm* en una infraestructura física. En el caso de desplegarlo en un proveedor de *Cloud* público el ahorro será económico al reducir el número de recursos (nodos de cómputo) utilizados

Uno de los principales objetivos es conocer todas las tecnologías mencionadas en la introducción. CLUES es un gestor de elasticidad para *clusters* que permite gestionar el número de nodos del mismo en función de la carga de trabajo del cluster. Trabaja con listas de trabajos y nodos, mientras que el concepto de trabajo no existe en *Docker*. Por lo que hay una parte de investigación e implementación para generar un concepto de trabajo dentro de *Docker Swarm.*

Se pretende poder aplicar los conocimientos adquiridos en Máster Universitario en Computación Paralela y Distribuida y destacar en todas las fases que componen este proyecto.

Una vez terminado el trabajo, adquiriremos suficientes conocimientos para construir un clúster escalable de contenedores.

1.2. Metodología de trabajo

La metodología que utilizamos para el desarrollo de software es el modelo espiral, ya que nos permite reducir riesgos e introducir mejoras durante el proyecto.

«El desarrollo en espiral es un modelo de ciclo de vida del software definido por primera vez por Barry Boehm en 1986, utilizado generalmente en la Ingeniería de software. Las actividades de este modelo se conforman en una espiral, en la que cada bucle o iteración representa un conjunto de actividades. Las actividades no están fijadas a ninguna prioridad, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior. » (7)

Las actividades a considerar en cada ciclo son las siguientes:

- Análisis:
  En el primer paso se estudian los requisitos que conlleva cada objetivo, se identifican los riesgos y sus posibles soluciones.

- Diseño:
  En esta etapa se realiza un diseño con los datos del paso anterior para describir las acciones a realizar.

- Implementación:
  En el tercer paso se realiza la programación y evaluaciones unitarias, tras haber elegido el modelo de desarrollo necesario.

- Pruebas:
  En el último paso es donde se revisa el proyecto, se realiza un test completo y la evaluación frente los objetivos propuestos. También se decide si se debe continuar con un ciclo más, en el caso afirmativo, se crearán planes para la siguiente fase.

Cada iteración crea una versión del software, que se vuelve más completa hasta que el software quede funcional.

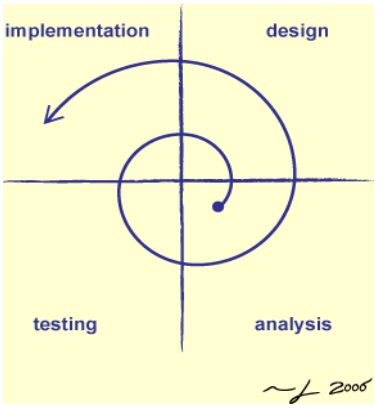

Ilustración 1. Modelo de desarrollo en espiral (8)

1.3. Estructura del documento

El proyecto contiene seis puntos principales que se detallan a continuación:

- Introducción: En este punto encontraremos una breve introducción al proyecto. Se presenta el problema, su importancia y su solución.

- Descripción de las tecnologías utilizadas: Tras la introducción se procede a explicar las tecnologías que vamos a usar y el funcionamiento de estas.

- Planificación: Con las tecnologías presentadas se procederá a analizar las tareas que se van a desarrollar. Se describirán los requisitos necesarios para obtener el cluster elástico. A partir de los requisitos se diseña la arquitectura y las conexiones de los componentes.

- Implementación: Esta parte muestra el entorno hardware y software utilizado, explicando con detalles su instalación y configuración. Se incluirán y comentarán las partes de código fuente más relevantes.

- Pruebas: Para verificar el correcto funcionamiento del producto final se muestran las pruebas realizadas.

- Conclusiones: Como finalización se exponen las conclusiones obtenidas argumentando si se han cumplido los objetivos planeados y se describen los trabajos futuros.

2. Tecnologías utilizadas

2.1. Docker

Docker es una plataforma abierta para el desarrollo, la transferencia y la ejecución de aplicaciones. Con *Docker* es posible separar las aplicaciones de la infraestructura y usar esta como administrador de aplicaciones. Con su uso se acorta el ciclo entre las etapas de desarrollo y producción.

En su esencia, *Docker* proporciona una forma de ejecutar casi cualquier aplicación de forma segura en un contenedor aislado. El aislamiento y la seguridad permiten ejecutar muchos contenedores de forma simultánea en el ordenador. La naturaleza ligera de contenedores, que se ejecutan sin la carga adicional de un virtualizador, permite que se pueda sacar más provecho del hardware.

Docker es una aplicación cliente-servidor con los siguientes componentes principales:

- Un proceso servidor encargado de administrar los contenedores.
- Una API REST que especifica las interfaces que se pueden usar para enviar comandos al servidor.
- Una interfaz de línea de comandos (CLI) como cliente.

Tal como muestra la imagen siguiente, la interfaz del cliente utiliza la API REST para interactuar con el servidor. El servidor se encarga de gestionar objetos como imágenes, contenedores, redes, volúmenes de datos, etc.

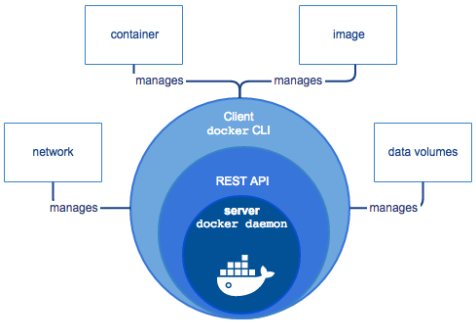

Ilustración 2. Arquitectura Docker (1)

Los tres recursos más importantes dentro de *Docker* son:

- *Docker images*: Las imágenes son plantillas de solo lectura de reducido tamaño. Contienen sistemas operativos Linux con aplicaciones instaladas.

- *Docker registries*: Es el almacén de imágenes. Existen muchas imágenes, tanto públicas como privadas, que pueden descargarse y modificarse libremente.

- *Docker containers*: Los contenedores son similares a los directorios, contienen lo necesario para hacer funcionar una aplicación sin la necesidad de acceder a elementos externos. Los contenedores se crean a partir de las imágenes y tienen distintos estados. Cada contenedor es una plataforma de aplicaciones aislada y segura, si bien el aislamiento que se puede alcanzar con contenedores no es equiparable al aislamiento que ofrecen las máquinas virtuales.

Un motivo importante del éxito de *Docker* es su ligereza frente a las máquinas virtuales. En el caso de las máquinas virtuales se dedican recursos para el virtualizador (*Hypervisor* en este caso), el sistema operativo virtualizado y en caso de replicación se duplican las librerías. Con *Docker* los contenedores se ejecutan directamente el *kernel* de *Linux*, por lo que no hay que mantener ningún virtualizador, esto permite disponer de más recursos que en las máquinas virtuales.

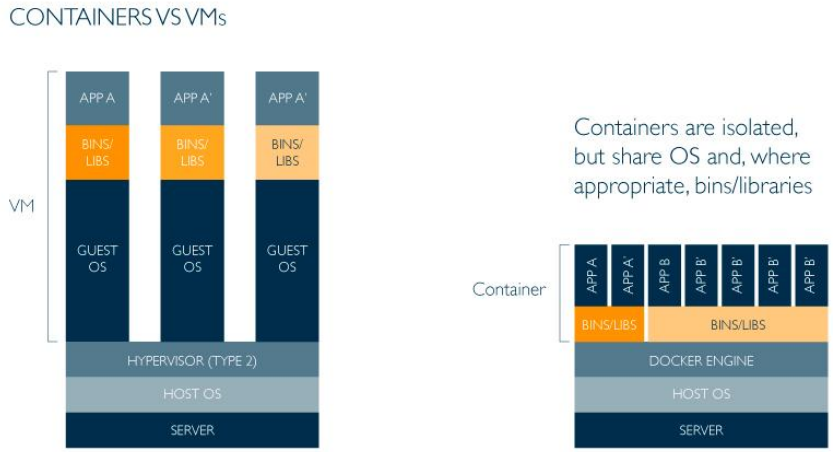

Ilustración 3. Contenedores y Máquinas virtuales (9)

Las imágenes de contenedores Docker se estructuran en base a capas que pueden ser reutilizadas. Por tanto, se pueden reutilizar elementos tales como librerías. A continuación podemos ver la estructura del contenedor.

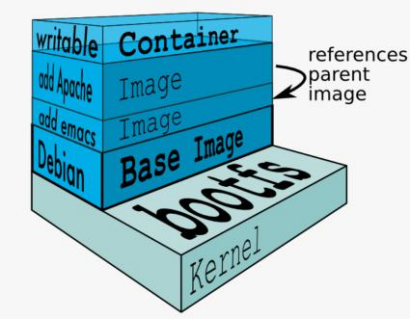

Ilustración 4. Estructura de un contenedor (10)

Poniendo un caso de ejemplo, suponiendo que una imagen de 1GB usada por 100MVs consumiría 100 GB. Si hay 900 MB inmutables, 100 contenedores *Docker* consumirían 0,9 + 0,1 * 100 = 10,9 GB.

Otra ventaja de *Docker* es que el inicio de las instancias es equivale al inicio de un proceso, lo que lo hace fácil de automatizar e implantar en entornos de integración continua.

El flujo de la información consiste en obtener una imagen almacenada en los servidores de *Docker* y después generar contenedores a partir de esta.

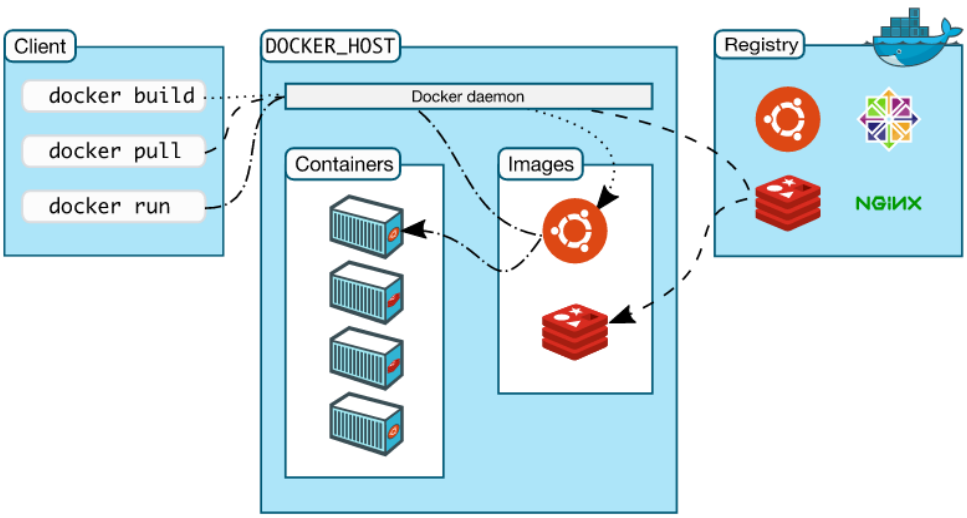

Ilustración 5. Flujo de datos dentro de Docker (1)

2.2. Docker Swarm

Docker Swarm permite crear un clúster nativo para *Docker*. Convierte un conjunto de nodos en un nodo único virtual para aprovechar todos los recursos de todas las máquinas. El reparto de contenedores entre los nodos esta implementado por *Docker*. Para la comunicación con el *cluster* se utiliza la API estándar de *Docker*, por lo que cualquier aplicación o herramienta que se comunicaba con un proceso *Docker*, puede comunicarse con *Docker Swarm* para escalar a varios nodos.

Algunas de las herramientas compatibles con la API de *Docker Swarm*:

- Dokku
- Docker Compose
- Docker Machine
- Jenkins

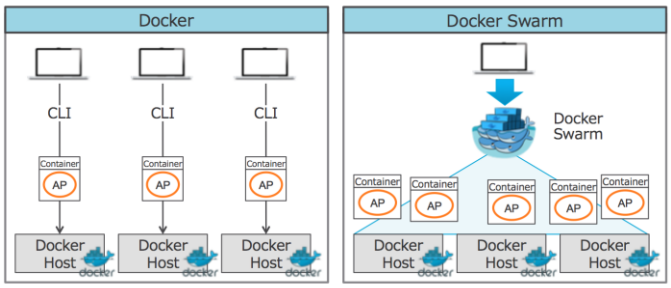

Ilustración 6. Docker y Docker Swarm (11)

Las reglas del clúster funcionan bajo los principios *plug and play* ya que se permite modificar el número de nodos dentro de *Docker Swarm* durante su ejecución gracias a sus servicios de descubrimiento.

Para crear el clúster se debe instalar *Docker* en todos los hosts, abrir un puerto *TCP* en cada nodo para la comunicación con el gestor de *Swarm*. Tras este paso, hay que configurar el contenedor de la imagen *Swarm* con el rol de agente o gestor. Por último se crean y administran los certificados *TLS* para proteger el clúster.

2.3. Docker Machine

Docker Machine es una herramienta que permite instalar *Docker* en una máquina virtual, también puede ser utilizado para gestionar las máquinas virtuales. Normalmente se utiliza en sistemas distintos de *Linux* para poder ejecutar contenedores, pero también puede servir para crear un clúster *Swarm*. Es posible instalar máquinas virtuales compatibles con *Docker* en *Windows*, *Mac*, en un centro de datos o en un proveedor cloud como *Amazon Web Services* (AWS), *Microsoft Azure* o *Digital Ocean*. Usando el comando *docker-machine* es posible iniciar, inspeccionar, detener, reiniciar y conectar un cliente a un host virtual.

Tal como se ha mencionado en la sección 2.1 el cliente Docker se comunica con el servidor a través de la API REST, en el caso de Docker machine la API sirve de proxy para comunicar el cliente con la API ligada al Docker Server ubicado en la máquina virtual. De esta manera es posible aprovisionar varios hosts Docker remotamente. Docker machine tiene la opción de configurar el CLI automáticamente con los parámetros del host remoto.

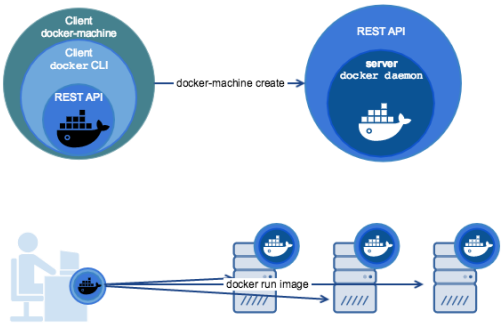

Ilustración 7. Conexión con un Docker server remoto (4)

2.4. CLUES

CLUES está desarrollado por el Grupo de Grid y Computación de Altas Prestaciones (GRyCAP) del Instituto de Instrumentación para Imagen Molecular de la Universidad Politécnica de Valencia. El sistema esta implementado completamente en *python*, lo cual permite ejecutar CLUES en cualquier plataforma, siempre que tenga el intérprete de *python* instalado. Se distribuye bajo la licencia de *Open Source GNU General Public License* - version 3.0. Esta licencia permite realizar modificaciones siempre que las futuras distribuciones mantengan el tipo de licencia.

"CLUES es un **sistema de gestión de energía para *clusters* de altas prestaciones (HPC) e infraestructuras cloud**, cuya función principal es la de **apagar los nodos internos del *cluster* cuando no están siendo utilizados** y, de forma recíproca, encenderlos de nuevo cuando son necesarios. El sistema CLUES **se integra con el *middleware* de gestión del *cluster***, como puede ser un gestor de colas o un sistema de gestión de infraestructura cloud, mediante el uso de una serie de conectores" (6)

Se puede utilizar en cualquier *cluster* con un gestor de colas o un gestor de infraestructura *Cloud*. El sistema se encarga de monitorizar los recursos locales mediante *plugins* de integración. Al detectar un nodo que no ha sido utilizado durante un tiempo, éste se considera candidato a ser apagado. Si ningún sistema integrado reclama su utilización, se modificará su estado. Uno de sus objetivos es afectar lo menos posible a las interacciones del usuario, proporcionando una apariencia de un *cluster* completamente encendido.

Su arquitectura es completamente modular, con el objetivo de que sea fácilmente adaptable a cualquier sistema de gestión de *clusters* y permitir la integración con cualquier infraestructura.

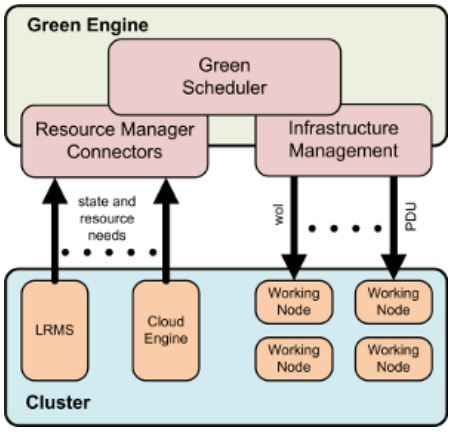

Ilustración 8. Arquitectura de CLUES (6)

La integración con la infraestructura se basa en *plugins* que se encargan de ejecutar comandos personalizados para cada tipo de cluster. También existe la posibilidad de gestionar infraestructuras heterogéneas.

Los conectores tienen dos tareas principales:

- Capturar los trabajos solicitados
- Monitorización de la utilización de nodos

2.4.1. Captura de los trabajos y monitorización de la utilización de nodos

Muchos de los sistemas de colas de *cluster* incorporan mecanismos para realizar acciones en el momento de lanzar un trabajo. Es el modo más adecuado de interceptar trabajos ya que permite una integración siguiendo la arquitectura del *cluster*. En el caso de que no exista dicho mecanismo, es posible escribir una aplicación o script que sustituya la llamada real, que se encargue de hacer la integración con CLUES y que posteriormente haga la llamada efectiva.

Prácticamente cualquier *cluster* incorpora comandos para obtener el estado de los nodos de trabajo. Por tanto, se recomienda utilizarlos para comunicar a CLUES el estado de la utilización de la infraestructura.

2.4.2. Políticas de ahorro de energía adaptables

Es posible establecer políticas de ahorro de energía personalizadas, gracias a que CLUES nos permite ajustar variadas características, las más importantes se comentan a continuación.

- Encendido por exceso, se hace una previsión de la demanda de futuros trabajos y enciende nodos para que las futuras solicitudes esperen lo menos posible.

- Tiempo para determinar si el nodo está siendo utilizado. Se establece un tiempo máximo para la inactividad del nodo. De esta manera es posible evitar los apagados por si existen tiempos de descanso del personal.

- Establecer nodos que no deben apagarse, de esta manera es posible tener un máximo número de nodos apagados sin perder prestaciones en el servicio.

2.5. Python

Python es un lenguaje de programación interpretado y multiparadigma. Soporta orientación a objetos, programación imperativa, programación funcional. Usa tipado dinámico. Es administrado por *Python Software Foundation*, posee una licencia de código abierto, compatible con la licencia GNU.

Fue creado en el 1991 por Guido van Rossum en el centro para las Matemáticas y la Informática en los Países Bajos. Su filosofía hace hincapié en la simplicidad y legibilidad fácil del código fuente.

3. Planificación

3.1. Requisitos

Tras estudiar las tecnologías utilizadas y analizar los requisitos de cada una se han concluido los siguientes requisitos para crear un *cluster Docker* elástico:

- Es necesaria la creación del concepto de trabajo dentro de *Docker Swarm*. *Docker* ejecuta contenedores directamente con el comando *docker run*. Por tanto, es necesario almacenarlos como trabajos para que CLUES pueda cuantificar el número de máquinas necesarias. Desde la documentación de CLUES, tal como se ha visto en el punto 2.4, se recomienda crear un comando para realizar una notificación a CLUES antes de enviar el comando original. Pero esto supone un problema, ya que sería necesario planificar un método de distribución para dicho comando y obligar a los usuarios a instalarlo y usarlo. Para evitar estos problemas, se ha decidido que el componente ira intercalado entre el cliente *Docker* y el nodo principal de *Docker Swarm*. De este modo la integración será transparente para el usuario.

- CLUES necesita un *plugin* para la obtención de un listado de trabajos y un listado de nodos.

- Para que CLUES pueda apagar y encender los nodos es necesario crear un *plugin* de aprovisionamiento.

3.2. Casos de uso

A partir de los requisitos se ha creado el diagrama de los casos de uso, para facilitar la comprensión de las acciones realizadas por el usuario.

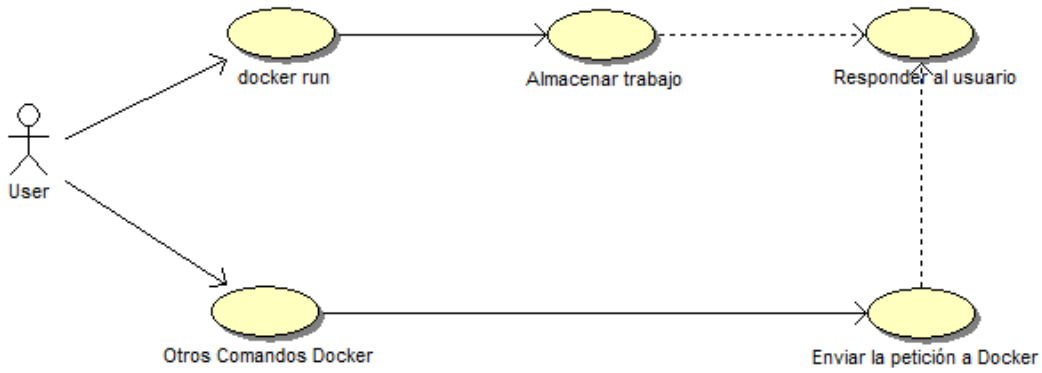

Ilustración 9. Casos de uso

A continuación explicamos con más detalle cada uno de los casos de uso.

Nombre:	*Docker run*
Descripción:	Permite al usuario almacenar como trabajo el lanzamiento de un contenedor.
Actores:	Usuario
Precondición:	No aplicable
Post-condición:	La lista de los trabajos es incrementada en una unidad.
Flujo Principal:	▪ El usuario envía el comando ▪ Se almacena el trabajo ▪ Se contesta al usuario
Flujo alternativo:	No procede.

Nombre:	Otros comandos *Docker*
Descripción:	Permite al usuario consultar y gestionar los contenedores dentro del cluster de *Docker Swarm*.
Actores:	Usuario
Precondición:	No aplicable
Post-condición:	El usuario obtiene la información relevante a sus acciones sobre el servidor.
Flujo Principal:	▪ El usuario envía el comando ▪ Se realiza una petición a *Docker Swarm* ▪ La respuesta del cluster se redirige al usuario
Flujo alternativo:	No procede.

3.3. Diseño de la arquitectura

Para satisfacer los requisitos establecidos, se ha diseñado la arquitectura del *cluster Docker Swarm* elástico.

En la parte del cliente el *Docker Client* se conectará con un procedimiento que hará el papel de servidor Proxy/HTTP. Dicho procedimiento se encargará de transferir todos los comandos al nodo principal del *cluster* excepto el comando *run*. Los lanzamientos de ejecución se almacenarán como trabajos.

Los trabajos y los nodos serán proporcionados al *plugin* de CLUES para que este haga los cálculos necesarios y apague o encienda los nodos necesarios haciendo uso de *Docker Machine*.

En la siguiente figura se puede apreciar un diagrama de la arquitectura con las conexiones existentes.

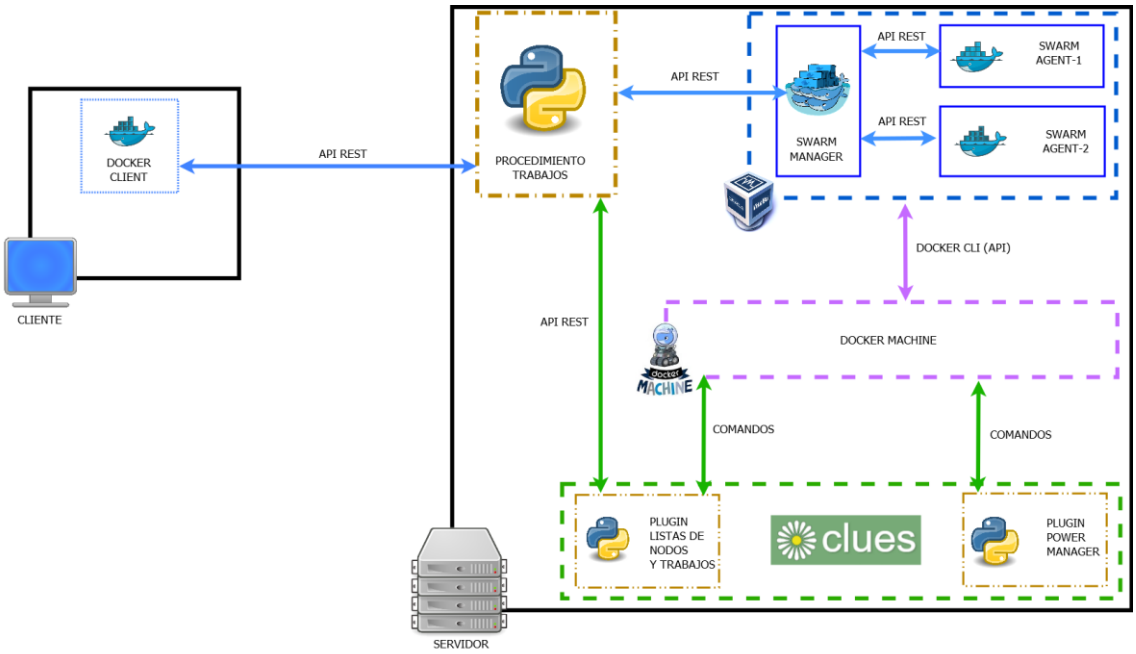

Ilustración 10. Arquitectura del cluster elástico.

3.4. Diagrama de secuencias del procedimiento

Los componentes interactúan entre sí, para proporcionar los servicios ofrecidos por la aplicación. Para facilitar la implementación del procedimiento se han creado los siguientes diagramas de secuencia. Representan las comunicaciones realizadas por el procedimiento componentes del sistema durante un escenario concreto.

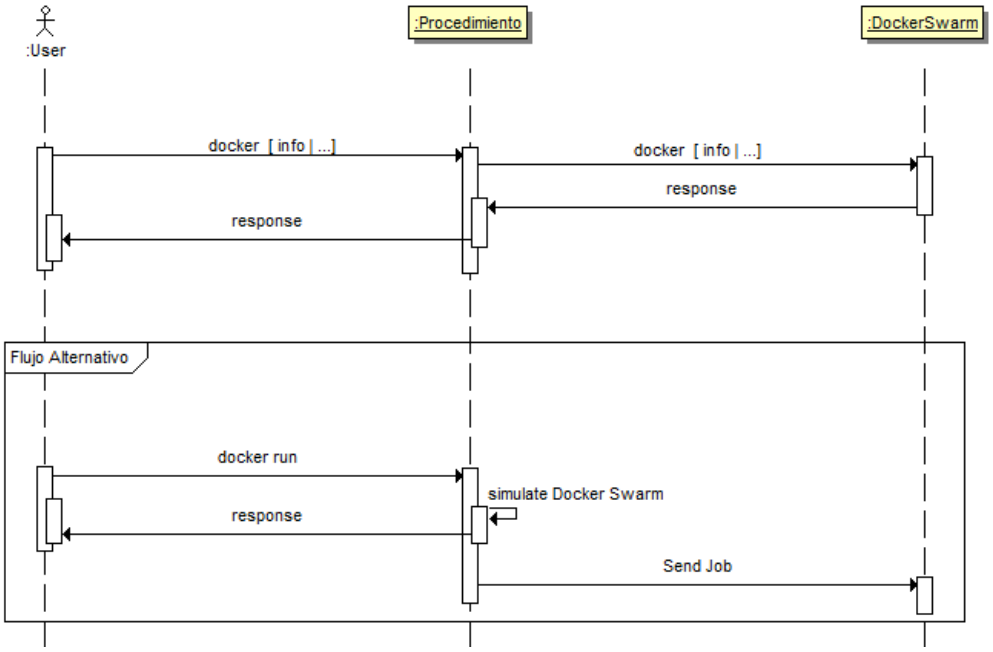

Ilustración 11. Comunicación entre el usuario y Docker Swarm

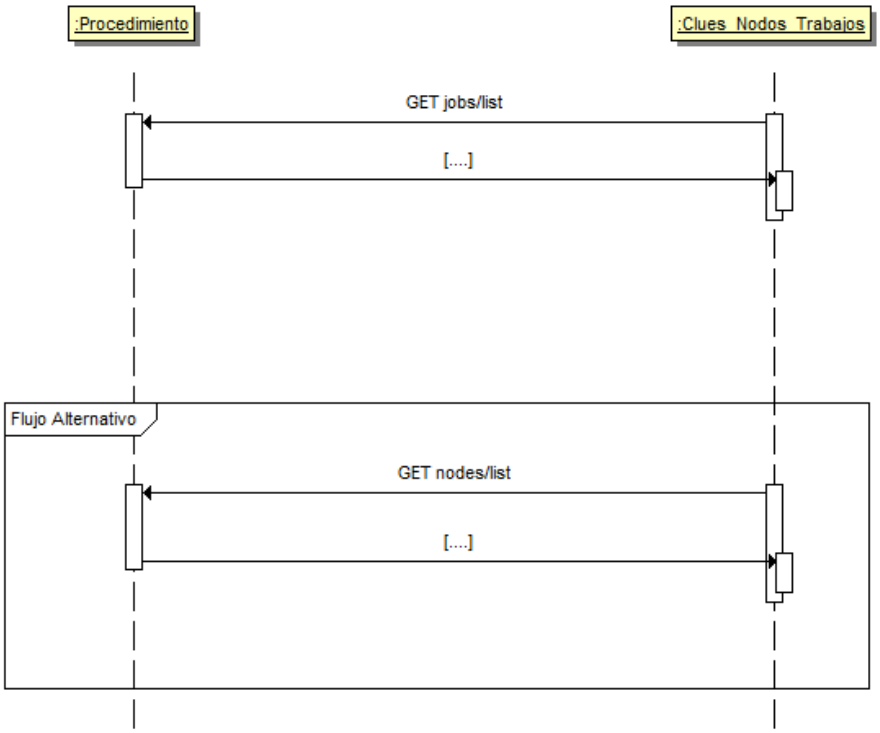

Ilustración 12. Comunicación entre CLUES y el procedimiento

3.5. Planificación del trabajo

Tras las reuniones y el diseño de la infraestructura se ha planificado el trabajo a realizar. A continuación se muestra las tareas de planificación estructuradas con el diagrama de *Gantt*.

	Nombre	Duracion	Inicio	Terminado
1	**TFM**	14,5 days	**29/07/16 9:00**	**18/08/16 15:00**
2	Reuniones y planificación	2 days	29/07/16 9:00	1/08/16 19:00
3	**Analisis de las tecnologías**	4 days	**1/08/16 9:00**	**4/08/16 19:00**
4	Docker	1 day	1/08/16 9:00	1/08/16 19:00
5	Docker Swarm	1 day	2/08/16 9:00	2/08/16 19:00
6	Docker Machine	1 day	3/08/16 9:00	3/08/16 19:00
7	CLUES	1 day	4/08/16 9:00	4/08/16 19:00
8	Diseño de la arquitectura	1 day	5/08/16 9:00	5/08/16 19:00
9	**Implementación**	6 days	**8/08/16 9:00**	**15/08/16 19:00**
10	**Procedimiento para Obtener Listado de Trabajos y Listado de Nodos**	3 days	**8/08/16 9:00**	**10/08/16 19:00**
11	Discriminar peticiones de trabajo y almacenarlas	1 day	8/08/16 9:00	8/08/16 19:00
12	Comunicar el cliente con el Docker Swarm	1 day	9/08/16 9:00	9/08/16 19:00
13	Comunicar el Cliente con el Docker Swarm	1 day	10/08/16 9:00	10/08/16 19:00
14	Plugin para Obtener Listado de Trabajos y Listado de Nodos	1 day	11/08/16 9:00	11/08/16 19:00
15	Plugin para Aprovisionar Nodos con docker-machine	2 days	12/08/16 9:00	15/08/16 19:00
16	Pruebas de Integración	1,25 days	16/08/16 9:00	17/08/16 11:00
17	Redacción de Memoria	12,5 days	2/08/16 9:00	18/08/16 15:00

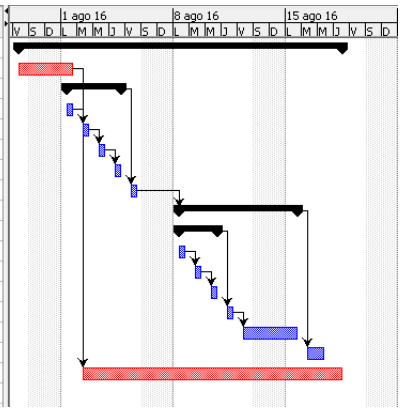

Ilustración 13. Gráfico de dependencias entre las tareas

Para la realización del diagrama de *Gantt* se ha utilizado software de administración de proyectos de código abierto denominado *ProjectLibre*.

4. Implementación

4.1. Entorno de desarrollo

4.1.1. Características del entorno

El trabajo fue desarrollado en un equipo informático con las siguientes características:

Procesador:	Intel(R) Core(TM) i7-3630QM 2.40GHz
Memoria RAM:	8 GB
S.O.:	Ubuntu 16.04.1 LTS

Software adicional para la documentación del trabajo:

MS Office:	Documentación.
ProjectLibre:	Diagramas de Gantt
DIA:	Diagrama de infraestructura
BOUML:	Diagramas UML de comunicación y casos de uso

4.1.2. Instalación de Virtual Box

Se ha visto en el apartado 2.3 que *Docker Machine* permite crear y gestionar máquinas virtuales. Por tanto, necesitamos un software de virtualización que se encargue de gestionar las máquinas virtuales, de modo que se ha optado por usar *VirtualBox*. Gracias a los repositorios *Linux* se instala con el siguiente comando.

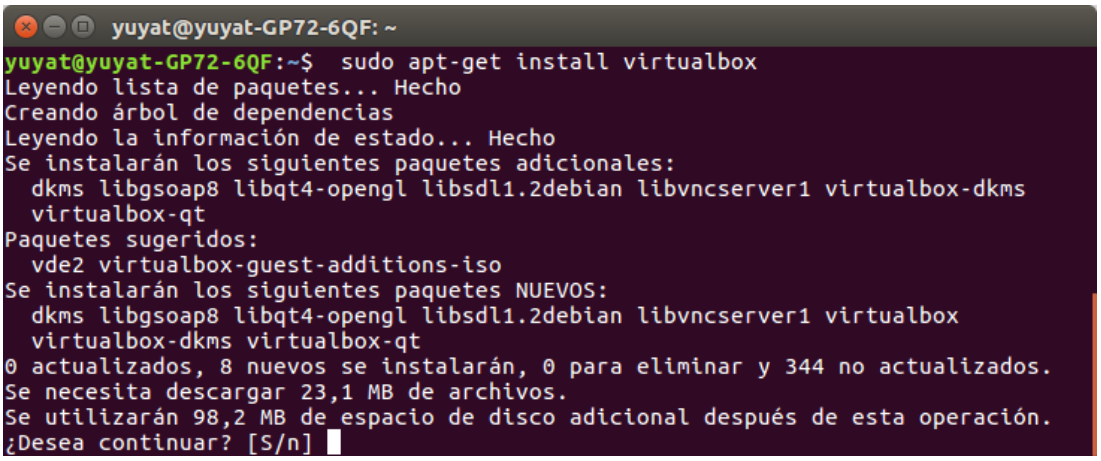

Ilustración 14. Instalación de VirtualBox

4.1.3.Instalación de Docker

Para instala *Docker*, primero se instalan los paquetes de dependencias.

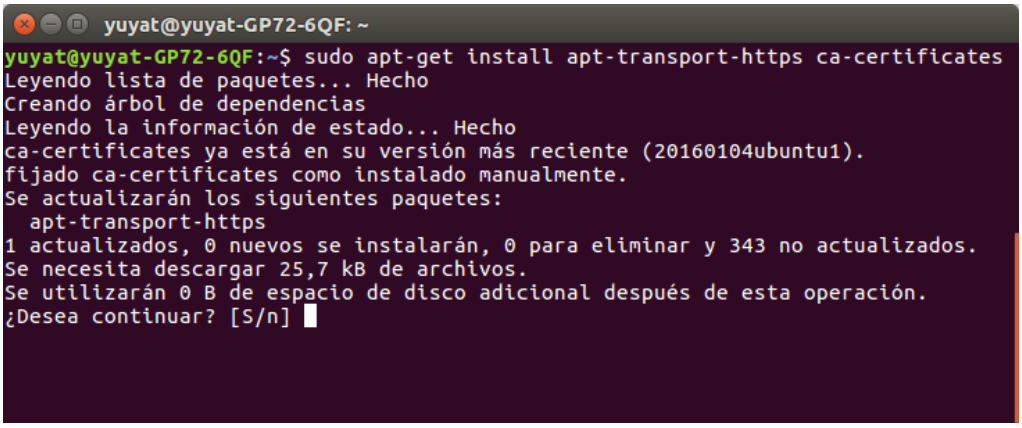

Ilustración 15. Instalando dependencias

Se añade la clave del repositorio con el siguiente comando.

Ilustración 16. Añadiendo certificado del repositorio

Se edita la el documento docker.list dentro de la ruta:
`vim /etc/apt/sources.list.d/docker.list`
Se añade el enlace del repositorio.

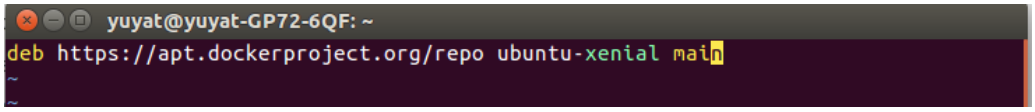

Ilustración 17. Añadiendo enlace del repositorio

Se actualiza la información de los paquetes. Se limpia el antiguo repositorio en el caso de que este instalado e instalamos los paquetes recomendados.

```
yuyat@yuyat-GP72-6QF:~$ sudo apt-get update
yuyat@yuyat-GP72-6QF:~$ sudo apt-get purge lxc-docker
yuyat@yuyat-GP72-6QF:~$ sudo apt-get install linux-image-extra-$(uname -r)
```

Ilustración 18. Limpieza e instalación de dependencias de Docker

En este punto ya está todo preparado para instalar *Docker*.

```
yuyat@yuyat-GP72-6QF: ~
yuyat@yuyat-GP72-6QF:~$ sudo apt-get install docker-engine
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  aufs-tools cgroupfs-mount
Se instalarán los siguientes paquetes NUEVOS:
  aufs-tools cgroupfs-mount docker-engine
0 actualizados, 3 nuevos se instalarán, 0 para eliminar y 341 no actualizados.
Se necesita descargar 19,6 MB de archivos.
Se utilizarán 102 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] S
```

Ilustración 19. Instalación de Docker

4.1.4. Instalación de Docker-Machine

En los sistemas OS X y *Linux* la instalación de *Docker Machine* es muy sencilla. Se trata de descargar el comando y darle permisos de ejecución.

```
root@yuyat-GP72-6QF: /home/yuyat
root@yuyat-GP72-6QF:/home/yuyat# curl -L https://github.com/docker/machine/releases/download/v0.7.0/docker-machine-`uname -s`-`uname -m` > /usr/local/bin/docker-machine && chmod +x /usr/local/bin/docker-machine
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   600    0   600    0     0    620      0 --:--:-- --:--:-- --:--:--   619
 28 34.5M   28 9995k    0     0  2143k      0  0:00:16  0:00:04  0:00:12 3863k
```

Ilustración 20. Instalación Docker Machine

4.1.5. Instalación de Docker Swarm

El primer paso es preparar los hosts virtuales, para el desarrollo usaremos 3 máquinas virtuales, un nodo principal y dos agentes. Creamos el host principal usando el comando *docker-machine*, especificando que nuestro virtualizador es *virtualbox*:

Ilustración 21. Instalación de VirtualBox

Nos conectamos al nodo principal y creamos el rol de *manager*, el resultado es un *token* que servirá para el descubrimiento de nodos.

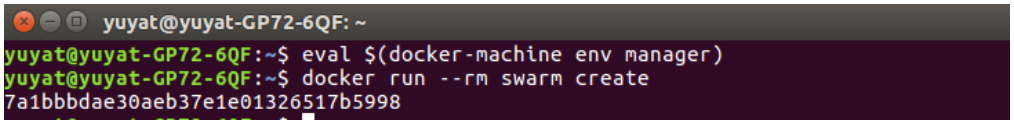

Ilustración 22. Generación del token de Docker Swarm

A partir del *token* obtenido se crea la instancia del nodo principal, se le proporcionan parámetros para indicar el puerto en el que se va a ejecutar, volumen con la ubicación de los certificados y los nombres que se ponen por defecto a los certificados digitales. Con este comando ya tenemos creado el nodo principal que se encuentra a la espera de las conexiones de los nodos agentes.

Ilustración 23. Aprovisionamiento de un nodo principal

Si listamos los contenedores dentro de *Docker* que se encuentra en el host virtual de manager, veremos el contenedor de *Swarm* en ejecución.

```
yuyat@yuyat-GP72-6QF:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                              NAMES
79889ed172a0        swarm               "/swarm manage -H 0.0"   22 seconds ago      Up 21 seconds       2375/tcp, 0.0.0.0:3376->3376/tcp   loving_boyd
```

Ilustración 24. Prueba de Swarm

Generamos a los otros dos nodos y los unimos al *cluster* como agentes indicando el *token* generado anteriormente.

```
docker-machine create -d virtualbox --swarm --swarm-discovery
token://7a1bbbdae30aeb37e1e01326517b5998 e swarm-agent-01

docker-machine create -d virtualbox --swarm --swarm-discovery
token://7a1bbbdae30aeb37e1e01326517b5998 swarm-agent-01
```

Generamos a los otros dos nodos y los unimos al *cluster* como agentes indicando el *token* generado anteriormente. Para verificar que las imágenes han sido creadas, podemos acceder al listado de las máquinas virtuales de *VirtualBox*.

Ilustración 25. Maquinas VirtualBox

A continuación se realiza la verificación del funcionamiento de *Docker Swarm*. Por lo que se configura el cliente de *Docker* para conectarse con el nodo principal del cluster, obteniendo la dirección IP de *docker-machine* e indicamos el puerto que se ha usado para crear el contenedor.

```
yuyat@yuyat-GP72-6QF:~$ export DOCKER_HOST=$(docker-machine ip manager):3376
```

Ilustración 26. Conexión a Swarm Manager

Tras solicitar la información al cluster, se puede observar que tiene dos nodos conectados, con dos contenedores en ejecución. Estos contenedores son los encargados de comunicarse con el nodo principal. Se concluye que el proceso de instalación y descubrimiento ha sido exitoso.

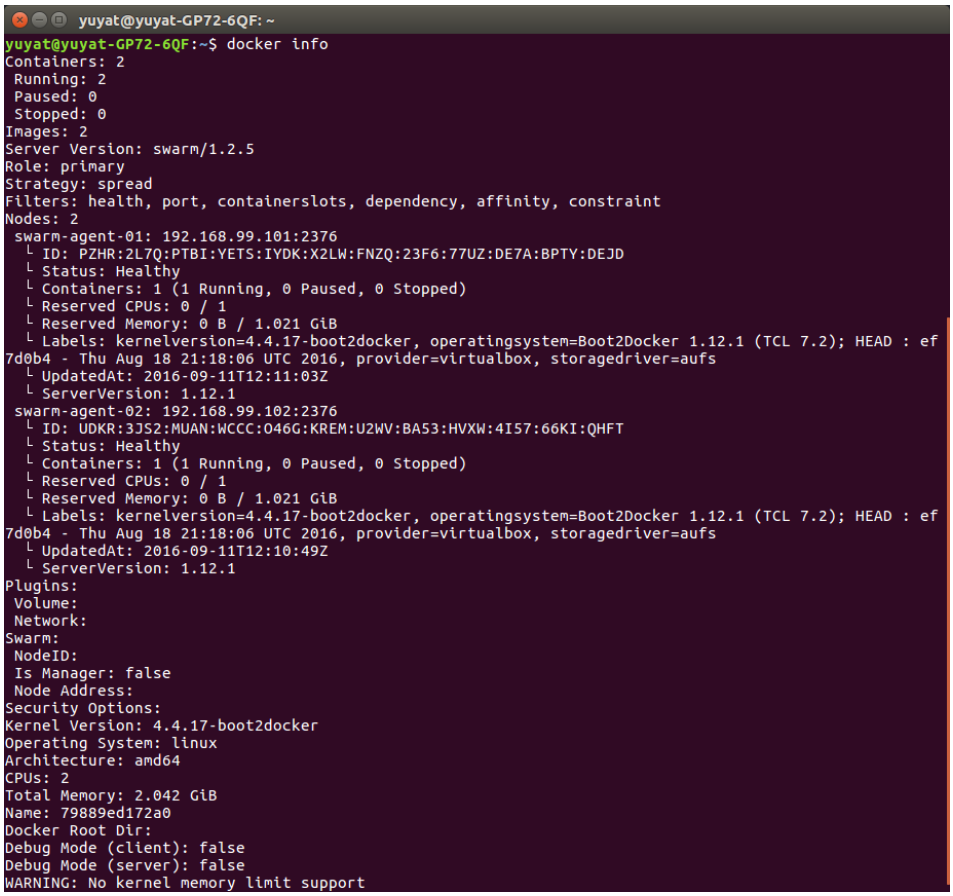

Ilustración 27. Datos del cluster

También hay que realizar una verificación funcional del *cluster*. Esta consiste en lanzar la ejecución de un contenedor y ver que efectivamente se ejecuta dentro de un nodo agente. En la siguiente ilustración se lanza la imagen de pruebas de *Docker*, en la pantalla aparece el resultado de la ejecución y se observa que el contenedor fue ubicado dentro del nodo llamado (swarm-agent-01). Por lo que podemos concluir el correcto funcionamiento del *cluster*.

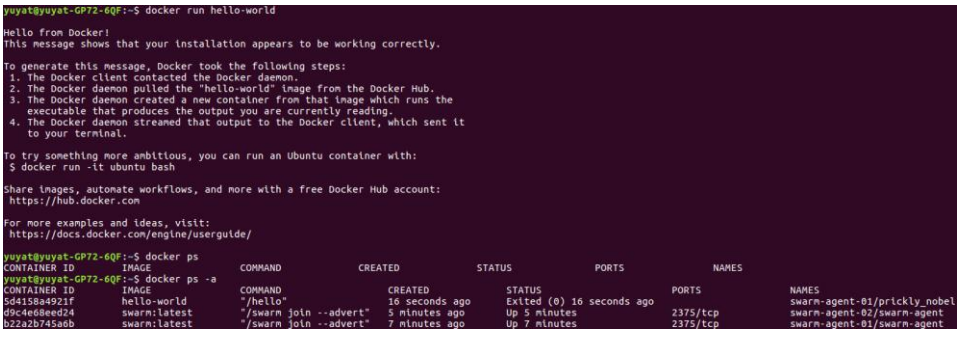

Ilustración 28. Verificación funcional

4.1.6. Instalación de CLUES

Para la instalación de CLUES es necesario tener instalado el intérprete de *Python* y la herramienta *easy_install*, se instalan con el siguiente comando. También se incluye el paquete *Git* para usarlo en los siguientes comandos.

Ilustración 29. Instalación de Python y python-stuptools

Según las instrucciones hace falta otro componente llamado *cpyutils*, también desarrollado por GRyCAP. Para la instalación clonamos el repositorio de *GitHub*, lo movemos al directorio */opt* y ejecutamos la instalación.

Ilustración 30. Clonación del repositorio de cpyutils

El comando para la instalación se tiene que ejecutar siempre que se añaden nuevos nodos.

```
yuyat@yuyat-GP72-6QF:/opt/cpyutils$ sudo python setup.py install --record installed-
files.txt
```

Ilustración 31. Ejecución del script de instalación

Para el correcto funcionamiento instalamos dos módulos que se indican en la documentación.

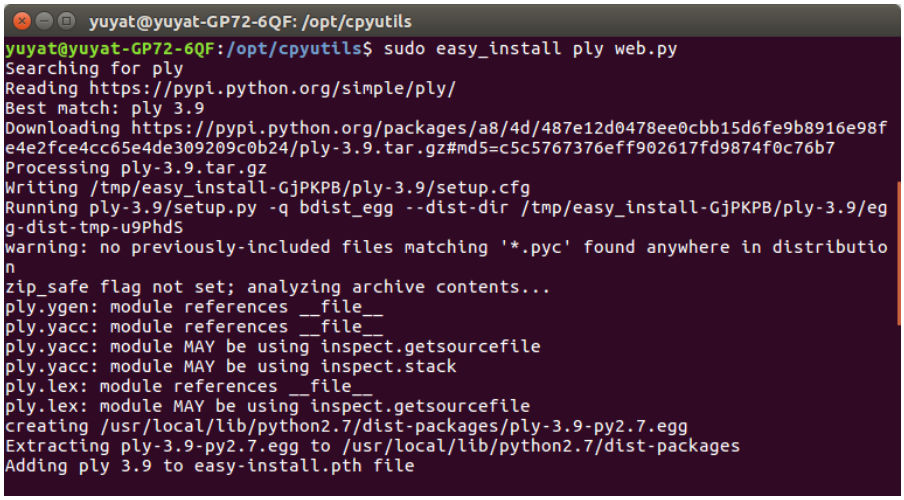

Ilustración 32. Instalación de dependencias de CLUES

En este punto ya está todo preparado para instalar CLUES. Se realizan las mismas operaciones que con el *cpyutils*, lo único que cambia es la dirección del repositorio.

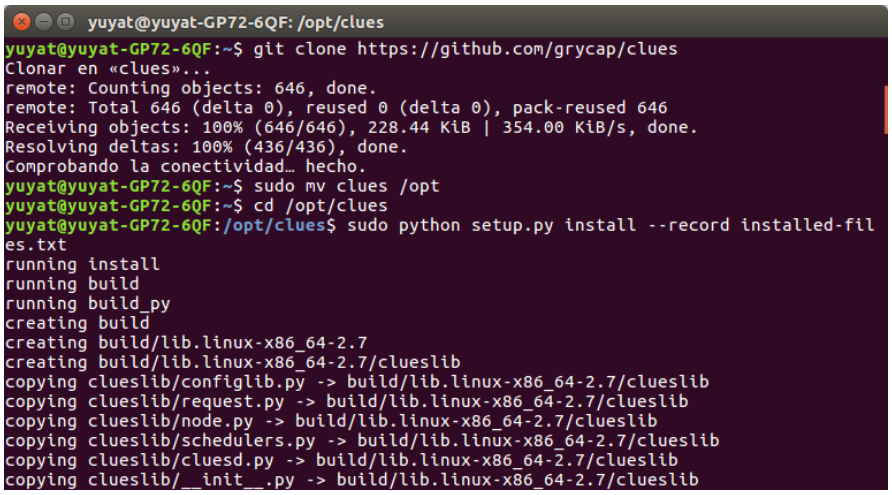

Ilustración 33. Instalación de CLUES

Para el funcionamiento de CLUES es necesario establecer una configuración. A continuación se muestra el archivo de configuración actual. Hay que tener en cuenta que los datos mostrados están pensados para realizar pruebas y no mantener un entorno de producción

```
[general]

CONFIG_DIR=conf.d
LRMS_CLASS=cluesplugins.docker
POWERMANAGER_CLASS=cluesplugins.dmachine
MAX_WAIT_POWERON=300
```

```
LOG_FILE=/var/log/clues2/clues2.log
LOG_LEVEL=debug

[monitoring]
PERIOD_MONITORING_JOBS=5
COOLDOWN_SERVED_REQUESTS=300

[scheduling]
SCHEDULER_CLASSES=clueslib.schedulers.CLUES_Scheduler_PowOn_Requests,
clueslib.schedulers.CLUES_Scheduler_Reconsider_Jobs,
clueslib.schedulers.CLUES_Scheduler_PowOff_IDLE,
clueslib.schedulers.CLUES_Scheduler_PowOn_Free
IDLE_TIME=10
RECONSIDER_JOB_TIME=150
EXTRA_SLOTS_FREE=1
EXTRA_NODES_PERIOD=60
```

- CONFIG_DIR: Indica el directorio donde se almacenan las configuraciones de los *plugins*.
- LRMS_CLASS: Activa el *plugin* para monitorizar los trabajos.
- POWERMANAGER_CLASS: Activa el *plugin* para el encendido y apagado de los nodos.
- MAX_WAIT_POWERON: Cota superior para indicar el tiempo que tarda un nodo en encenderse.
- PERIOD_MONITORING_JOBS: Periodo de monitorización de los trabajos.
- COOLDOWN_SERVED_REQUESTS: Es el tiempo durante el cual se pueden reservar los recursos solicitados a partir de indicios. (Ejemplo: se ha conectado un nodo)
- SCHEDULER_CLASSES: Características de encendido para el despliegue.
- IDLE_TIME: Es el tiempo durante el cual un nodo tiene que estar inactivo para ser considerado apagado.
- RECONSIDER_JOB_TIME: Tiempo durante el cual el trabajo debe estar en la cola.
- EXTRA_SLOTS_FREE: Establece el número de ranuras disponibles para la plataforma. Se asocia a que cada nodo tiene un número de ranuras que son ocupadas por trabajos.
- EXTRA_NODES_PERIOD: Frecuencia del planificador.

La configuración de CLUES se encuentra en la ruta: */etc/clues2/clues2.cfg*

4.2. Desarrollo del procedimiento para obtener trabajos y nodos

Como se ha comentado anteriormente, *Docker* no tiene un concepto de trabajo, de esta parte se encargará el procedimiento que describimos en este punto. Ya que toda la comunicación se realiza por API REST, el procedimiento tiene la estructura de un servidor web. Se han sobrescrito los manejadores de mensajes GET, POST, DELETE, HEAD para corresponder con la API de *Docker* que envía la consola.

Se han creado otros métodos para responder a las funcionalidades requeridas:

- **add_JOB**: Almacena la petición *run* recibida en una estructura de datos.
- **num_JOBS**: Devuelve el número total de trabajos almacenados.
- **list_JOBS**: Devuelve la lista con los trabajos almacenados y los trabajos en ejecución
- **send_JOB**: Envía el primer trabajo de la cola al *cluster*.
- **list_CONTAINERS**: Método auxiliar que sirve para obtener la lista de los contenedores en ejecución y determinar la carga que tiene cada nodo.
- **list_NODES**: Devuelve la lista de nodos que se encuentran en uso dentro de *Docker Swarm*.

Los nodos almacenados contienen la siguiente estructura de datos:

Nombre	Descripción
name	Nombre del nodo
Containers	El número total de contenedores en marcha.
CPU	Porcentaje de la CPU utilizada
RAM	Porcentaje de la Memoria utilizada

Los trabajos contienen un significativo número de propiedades, todas vienen generados por la consola, por lo que se almacenan sin alteraciones dentro del objeto de trabajo.

Nombre	Descripción
Status	Estado del trabajo (*Pending*) si es del usuario y (*Attended*) si está en ejecución
Data	Parámetros del trabajo.

Aparte de los trabajos pendientes, se añaden los trabajos (contenedores) que ya están en ejecución para la mejor gestión de CLUES.

Cuando el usuario envía un trabajo, este se almacena y lanza un hilo que se encargará de enviarlo al *cluster*. Con el primer trabajo se activa un semáforo que permite la ejecución periódica de los envíos, siempre que existan trabajos. Cuando no quede ningún trabajo pendiente, el semáforo se desactivará y la ejecución periódica se detendrá hasta que se envíe otro trabajo nuevo. Esto permitirá a CLUES desplegar nuevos nodos en el caso de que sea necesario.

```
def send_JOB(self):
    global JOBS_P
    global cert
    global key
    global IPDOCKER
    print "send_JOB"
    global timer
    if len(JOBS_P) > 0:
        JOB = JOBS_P.pop()
        r = requests.post("https://"+IPDOCKER+CRETEPATH, json=json.loads(JOB["Data"]), cert=(cert,key), verify=False)
        t = threading.Timer(30.0, self.send_JOB)
        t.daemon = True
        t.start()
    else:
        timer = False
    return
```

Ilustración 34. Método de envío de trabajos

Como ejemplo se muestra uno de los métodos para no ser redundantes ya que todos tienen una estructura similar. En este caso se trata del método *do_GET*. Si la petición va dirigida a alguna de las rutas que se ven en la estructura *if*, se invocará el método adecuado. Si no se cumple ninguna condición entonces la petición se retransmite al cluster.

```
def do_GET(self):
    print  self.path
    global cert
    global key
    global IPDOCKER
    global REQUEST
    REQUEST += 1
    if "jobs/num" in self.path:
        self.num_JOBS()
    elif "jobs/list" in self.path:
        self.list_JOBS()
    elif "nodes/list" in self.path:
        self.list_NODES()
    else:
        print "do_GET "+str(REQUEST)
        r = requests.get("https://"+IPDOCKER+self.path, cert=(cert,key), verify=False, headers=self.headers)
        self.send_response(r.status_code)
        for header in r.headers:
            self.send_header(header,r.headers[header])
        self.end_headers()
        self.wfile.write(r.content)
    return
```

Ilustración 35. Metodo do_Get

Para poner en marcha el procedimiento, hay que indicarle el puerto del nodo principal de *Docker Swarm* y los paramentos de conexión proporcionados por *Docker Machine*.

```
python docker-wrapper.py 3376 `docker-machine env manager`
```

A su vez, el cliente tiene que establecer la siguiente configuración:

```
export DOCKER_HOST=tcp://127.0.0.1:8880
```

En el caso de que previamente existiera una configuración de distinta hay que quitar las siguientes variables:

unset DOCKER_TLS_VERIFY

unset DOCKER_CERT_PATH

4.3. Desarrollo del plugin para obtener listas de trabajos y nodos

Teniendo solucionado el requisito de proporcionar listas de nodos y trabajos, se ha procedido a implementar el *plugin* para CLUES encargado para recibir las listas y procesarlas para tomar decisiones con los nodos.

Los *plugins* de CLUES se basan en dos métodos principales, *get_nodeinfolist* para obtener la lista de trabajos y mapearla a una estructura conocida y *get_jobinfolist* con el mismo propósito para la lista de trabajos. Los nodos se almacenan en objetos *NodeInfo*, y los datos se mapean a dicha estructura.

En el caso de los nodos, la lista que se recibe es incompleta, porque se envían solo los nodos que están ejecutando algo en este momento que no sea el contenedor de *swarm*. Para tener una visión completa de todos los elementos que intervienen en la arquitectura se consulta la lista de máquinas virtuales controladas por *Docker Machine*. Dentro de la lista, se inspeccionan los elementos que formen parte del *cluster*. Esto es posible gracias a que *Docker Machine* almacena el *token* de descubrimiento entre las propiedades de las máquinas. A la hora registrar los nodos se complementan sus datos con los parámetros de la lista recibida del procedimiento explicado en el apartado anterior.

Para determinar si un nodo está en estado IDLE, USED, OFF o UNKNOWN se tienen en cuenta las siguientes comprobaciones.

- IDL: El estado de la máquina virtual es *Running* pero no tiene contenedores en ejecución.
- USED: El estado de la máquina virtual es *Running* y tiene contenedores en ejecución.
- OFF: El estado de la máquina virtual no es *Running*
- UNKNOWN: Como control de excepciones se declara este estado por si ocurre alguna anomalía.

Para la parte de los trabajos el procedimiento es similar, solo que la fuente de información es exclusivamente la lista recibida del procedimiento. Los trabajos pendientes y los trabajos en ejecución comparten una estructura de datos, por lo que resulta fácil de mapearlos a los objetos *JobInfo* utilizados por CLUES.

La configuración de este *plugin* se encuentra en la ruta: */etc/clues2/conf.d/plugin-docker.cfg*

4.4. Desarrollo del plugin para el aprovisionamiento de nodos

A la hora de apagar o de encender un nodo, delegamos en *Docker Machine*. Se indican los comandos utilizado para instanciar la clase *PowerManager_cmdline*.

La configuración de este *plugin* se encuentra en la ruta: */etc/clues2/conf.d/plugin-dmachine.cfg*

```
try:
    config_dmachine
except:
    config_dmachine = cpyutils.config.Configuration(
        "DMACHINE",
        {
            "DMACHINE_CMDLINE_POWON": "/usr/local/bin/docker-machine start %%h",
            "DMACHINE_CMDLINE_POWOFF": "/usr/local/bin/docker-machine stop %%h "
        }

class powermanager(PowerManager_cmdline):
    def __init__(self):
        PowerManager_cmdline.__init__(self, config_dmachine.DMACHINE_CMDLINE_POWON, config_dmachine.DMACHINE_CMDLINE_POWOFF)
```

Ilustración 36. Código principal del plugin de aprovisionamiento.

5. Pruebas

Tal como se indica en la sección 1.2, la metodología utilizada es el modelo en espiral. Por tanto, con cada versión de cada componente desarrollado se hacían pruebas hasta cumplir todos los requisitos funcionales. A continuación se describen las pruebas realizadas en la última versión.

Se ha monitorizado el funcionamiento del procedimiento encargado de comunicar el cliente con el servidor *Docker Swarm*.

```
yuyat@yuyat-SATELLITE-L850-1UX:~$ export DOCKER_HOST=tcp://127.0.0.1:8880
yuyat@yuyat-SATELLITE-L850-1UX:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
10eeb9294753        swarm:latest        "/swarm join --advert"   15 minutes ago      Up 15 minutes       2375/tcp            swarm-agent-01/swarm-agent
yuyat@yuyat-SATELLITE-L850-1UX:~$ docker pull redis
Using default tag: latest
swarm-agent-01: Pulling redis:latest... : downloaded
yuyat@yuyat-SATELLITE-L850-1UX:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               latest              0d1cbfaa41da        2 weeks ago         185 MB
swarm               latest              942fdSfd3S7e        4 weeks ago         19.47 MB
yuyat@yuyat-SATELLITE-L850-1UX:~$ docker info
Containers: 1
 Running: 1
 Paused: 0
 Stopped: 0
Images: 2
Server Version: swarm/1.2.5
Role: primary
Strategy: spread
Filters: health, port, containerslots, dependency, affinity, constraint
Nodes: 1
 swarm-agent-01: 192.168.99.103:2376
  └ ID: IJIS:XI63:XNV4:3H62:DOBH:VMJD:IHK3:NYMG:UGXU:TZOL:NOXY:QBOT
  └ Status: Healthy
  └ Containers: 1 (1 Running, 0 Paused, 0 Stopped)
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 1.021 GiB
  └ Labels: kernelversion=4.4.17-boot2docker, operatingsystem=Boot2Docker 1.12.1 (TCL 7.2); HEAD : ef7d0b4 - Thu Aug 18 21:18:06 UTC 2016, provider=virtualbox, storagedriver=aufs
  └ UpdatedAt: 2016-09-17T23:48:18Z
  └ ServerVersion: 1.12.1
Plugins:
 Volume:
 Network:
Swarm:
 NodeID:
 Is Manager: false
 Node Address:
Security Options:
Kernel Version: 4.4.17-boot2docker
Operating System: linux
Architecture: amd64
CPUs: 1
Total Memory: 1.021 GiB
Name: 19c26c612053
Docker Root Dir:
Debug Mode (client): false
Debug Mode (server): false
WARNING: No kernel memory limit support
yuyat@yuyat-SATELLITE-L850-1UX:~$
```

Ilustración 37. Test del procedimiento

El resultado es satisfactorio, prácticamente no se nota la presencia del procedimiento. La única diferencia está en los procesos *run*. Por un lado es debido a que no se utilizan con el propósito general de *Docker* y por el otro, porque el comando devuelve paquetes con el tipo de datos *application/x-tar* para realizar *streaming* (12) y el procedimiento no está diseñado para estos. Por estos motivos no es posible acceder al contenedor a través del procedimiento, solo se debe usar para la gestión.

Para validar los *plugins* y la configuración de CLUES fue necesario simular cargas de trabajo y monitorizar los estados de las máquinas virtuales junto el log de CLUES. Partiendo de dos nodos agentes y un nodo principal, se ha dejado el *cluster* sin trabajos. Podemos afirmar que CLUES detecta los nodos correctamente.

```
First monitorization of LRMS:
List of nodes:
[NODE "swarm-agent-02"] state: idle (since 1474160173), 1/1 (slots) - ID: 001 @1474160173
[NODE "swarm-agent-01"] state: idle (since 1474160173), 1/1 (slots) - ID: 002 @1474160173
```

Ilustración 38. Test CLUES registro de nodos

Al cabo de un rato, CLUES detecta que hay dos nodos sin trabajo. Se decide apagar solo uno de ellos. Esto es debido a la configuración, concretamente al parámetro EXTRA_SLOTS_FREE = 1 que se explicó en la sección 4.1.6.

Ilustración 39. Test CLUES, apagado de un nodo

Se puede usar *Docker Machine* o la interfaz gráfica de VirtualBox para verificar que el estado del nodo se ha modificado. Como la máquina virtual se ha apagado correctamente, queda probada la primera función del *plugin* de aprovisionamiento de nodos.

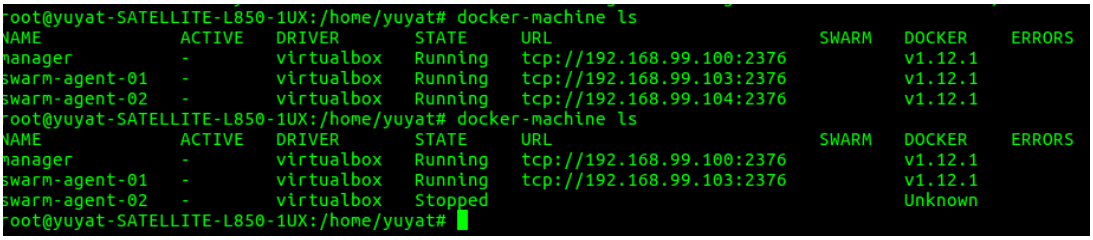

Ilustración 40. Test CLUES, primer cambio de estado del nodo

Para probar que los nodos se encienden correctamente, se han generado varios trabajos en la cola. Uno tras otro fueron enviados para la ejecución dentro del nodo disponible.

Ilustración 41. Test CLUES, trabajos en el cluster

CLUES detectó el incremento de la carga consultado las listas de nodos y trabajos. Por lo que se puede afirmar que registran los trabajos correctamente.

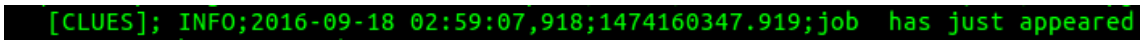

Ilustración 42. Test CLUES, detección de trabajos

Como consecuencia de la carga de trabajo, se decide poner en marcha el segundo nodo agente.

Ilustración 43. Test CLUES, encendido de un nodo

Se vuelve a verificar el estado dentro de *Docker Machine*. Queda cubierta la segunda función del *plugin* de aprovisionamiento de nodos. El nodo encendido se integra con normalidad dentro del *cluster*, gracias a que sigue manteniendo el *token* de descubrimiento. Al conectarse vuelve a ser un nodo funcional.

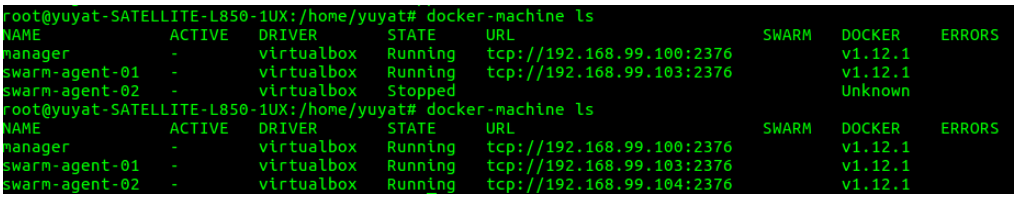

Ilustración 44. Test CLUES, segundo cambio de estado del nodo

6. Conclusiones y trabajos futuros

Se partía de unos conocimientos de usuario en *Docker*. Tras profundizar de forma teórica en las plataformas de *Docker* y CLUES, hemos conocido sus orígenes, sus componentes y el comportamiento de estos en el instante de ejecución. También hemos profundizado en otras herramientas que se encuentran dentro del ecosistema *Docker*. Todo esto ha permitido extender el uso de CLUES a *clusters* de contenedores.

En este trabajo se combinan las ventajas de Docker con las ventajas de CLUES. Docker hace un uso eficiente de los recursos de la infraestructura y proporciona despliegues de servicios en poco tiempo. CLUES permite una gestión eficiente del uso de los nodos y gracias a su diseño permite una integración inmediata con el *cluster*. Dicho de otro modo, no es necesaria su presencia desde el momento en el que se pone en marcha el *cluster*.

El producto final ha pasado las pruebas realizadas y cumple con los requisitos propuestos inicialmente. Hemos obtenido un *cluster* de contenedores que adapta sus componentes a la carga de trabajo.

Se plantean dos posibles mejoras. La primera consiste en adaptar el procedimiento desarrollado para obtener trabajos y nodos para que soporte el *streaming.* Haciendo la conexión con el *cluster* totalmente transparente para el usuario.

La otra mejora sería permitir el uso de nodos físicos o nodos dentro de un proveedor *Cloud*. El diseño permite usar nodos distintos a las máquinas virtuales sin realizar grandes cambios. Esto es debido a que CLUES es modular. Se pueden utilizar los *plugins* que ya se encuentran disponibles dentro de CLUES. El cambio principal supondría utilizar otro *plugin* de aprovisionamiento con una lista de nodos. Los cambios menos significativos consisten en modificar los archivos de configuración, cambiando los comandos de *docker-machine* encargados de aprovisionar nodos y proporcionar su configuración, por otros que tengan la misma finalidad dentro de la plataforma utilizada.

7. Bibliografía

1. **Docker.** Docker Docs. [En línea] https://docs.docker.com/engine/understanding-docker/.

2. **Wikipedia.** chroot. [En línea] https://en.wikipedia.org/wiki/Chroot.

3. —. Unix. [En línea] https://en.wikipedia.org/wiki/Unix.

4. **Docker.** Docker- Machine. [En línea] https://docs.docker.com/machine/overview/.

5. **Swarm.** Swarm overview. [En línea] https://docs.docker.com/swarm/overview/.

6. **GRyCAP.** CLUES. Cluster Eenergy Saving (For HPC and Cloud Computing). [En línea] http://www.grycap.upv.es/clues/es/index.php.

7. **Wikipedia.** Desarrollo en espiral. [En línea] [Citado el: 20 de Junio de 2016.] https://es.wikipedia.org/wiki/Desarrollo_en_espiral.

8. **Python Software Foundation.** cgl.ucsf.edu. [En línea] [Citado el: 20 de Junio de 2016.] http://www.cgl.ucsf.edu/Outreach/bmi219/slides/swc/lec/dev01.html.

9. **Poelwijk, Sander.** leaseweb.com. *Drag & drop infrastructure.* [En línea] http://blog.leaseweb.com/2014/07/24/drag-drop-infrastructure/.

10. **Tan, James.** jam.sg. [En línea] http://jam.sg/blog/mongodb-docker-part-2/.

11. **penflip.com.** akira.ohio · AppCatalyst Hands-on Lab en. [En línea] https://www.penflip.com/akira.ohio/appcatalyst-hands-on-lab-en/blob/master/About.txt.

12. **Wikipedia.** Streaming. [En línea] https://es.wikipedia.org/wiki/Streaming.

8. Materiales consultados

1. **Docker.** Installation on Ubuntu. [En línea] https://docs.docker.com/engine/installation/linux/ubuntulinux/.

2. **PtyhonTM.** BaseHTTPServer. [En línea] https://wiki.python.org/moin/BaseHttpServer.

3. **PyMOTW.** BaseHTTPServer – base classes for implementing web servers. [En línea] https://pymotw.com/2/BaseHTTPServer/.

4. **Mouat, Adrian.** *Using Docker: Developing and Deploying Software with Containers*. s.l. : O'Reilly Media, 2015.

5. **Wikipedia.** Virtualization. [En línea] https://en.wikipedia.org/wiki/Virtualization.

6. **Docker.** Docker Remote AP. [En línea] https://docs.docker.com/engine/reference/api/docker_remote_api/.

7. **Docker**. Docker Swarm API. [En línea] https://docs.docker.com/swarm/swarm-api/.