Servicio para el desarrollo de aplicaciones en el cloud basado en Euclid

Marc Riera Marcer

Resumen– Un proyecto en continuo desarrollo por la Agencia Espacial Europea (ESA), y diseñado para comprender el origen de la expansión acelerada del universo, es Euclid. Este proyecto, realiza un proceso de captación de imágenes y datos a través de un satélite, los cuales son almacenados, y posteriormente procesados en sus respectivos centros de datos. Antes de llegar a esta fase de ejecución, los científicos de Euclid, deben desarrollar las aplicaciones, que permitirán el análisis de estos datos, en un entorno llamado Eden. Debido a las dependencias críticas, que produce la construcción de este entorno, con el sistema operativo residente y algunas librerías o compiladores, Euclid proporciona al desarrollador de estas aplicaciones, la configuración necesaria para este entorno, construida sobre una máquina virtual. Esta solución tiene un gran consumo de recursos en el sistema local del usuario. En este documento, se realiza un análisis, sobre las posibilidades actuales para la integración e implementación del entorno de Euclid, con servicios de desarrollo de aplicaciones, integrados en un entorno de cloud. Se comparan dos posibles implementaciones sobre el servicio de desarrollo, construidos sobre una arquitectura centralizada y otra distribuida. Se observa como el servicio ejecutado en un entorno centralizado, proporciona una evolución del servicio más lineal, pero limitando el numero de usuarios a pequeña escala.

Palabras clave– namespaces, cgroups, containers, Docker, cloud computing, IDE, Eclipse Che, Openshift, Kubernetes, nfs, pnfs, Big data, OpenLdap, Keycloak, Postgres.

Abstract– A project in continuous development by the European Space Agency (ESA), and designed to understand the origin of the accelerated expansion of the universe, is Euclid. This project performs a process of capturing images and data through a satellite, which are stored, and then processed in their respective data centers. Before reaching this phase of execution, Euclid scientists must develop the applications, which will allow the analysis of this data, in an environment called Eden. Due to the critical dependencies, produced by the construction of this environment, with the resident operating system, and some libraries or compilers, Euclid provides the developer of these applications, the necessary configuration for this environment, built on a virtual machine. This solution has a large consumption of resources in the user's local system. In this document, an analysis is made about the current possibilities for the integration and implementation of the Euclid environment, with application development services, integrated in a cloud environment. Two possible implementations are compared on the development service, built on a centralized architecture and a distributed one. It is observed how the service executed in a centralized environment, provides a more linear service evolution, but limiting the number of users on a small scale.

Keywords– namespaces, cgroups, containers, Docker, cloud computing, IDE, Eclipse Che, Openshift, Kubernetes, nfs, pnfs, Big data, OpenLdap, Keycloak, Postgres.

1 Introducción

EUCLID es un proyecto actual en desarrollo, que se encarga de observar cientos de millones de galaxias en regiones del cielo divididas en octantes. En estas observaciones se registran imágenes y datos sobre las evidencias que va dejando la materia oscura, la energía oscura y la gravedad. Estos registros, se producen mediante

• E-mail de contacte: marc.rierama@e-campus.uab.cat
• Menció realitzada: Enginyeria de Computadors
• Treball tutoritzat per: Porfidio Hernández Budé (DACSO)
• Curs 2017/18

captación por satélite a través de un espectrofotómetro de infrarrojo cercano, y una cámara en el óptico. Este proceso produce a los investigadores del proyecto de Euclid, la obtención de datos y referencias, con las cuales se puede determinar una posible geometría del universo y una evolución del cosmos, en la formación de sus estructuras. Una parte de esta misión espacial, esta dedicada al desarrollo de las aplicaciones necesarias, para el análisis sobre los datos recibidos, y almacenados en sus correspondientes centros de datos.
El proceso sobre la configuración del sistema, para la instalación de un entrono de desarrollo para estas aplicaciones, despliega una problemática en la comunidad científica dedicada a este proyecto. En la mayor parte de los entornos de desarrollo de escritorio, existe alguna distribución compatible con el sistema operativo residente del usuario, y permiten la instalación de diferentes librerías, compiladores, y/o otros requisitos, que puedan ser necesarios para el correcto desarrollo de la aplicación. Como se observa en la tabla 1, en el caso de Euclid, por ejemplo, una de las dependencias críticas, se encuentra el la distribución, y versión del sistema operativo residente del usuario. Existen muchas otras dependencias sobre librerías, compiladores u otras herramientas necesarias. En la tabla 1, se muestran algunas de ellas.

Item	**Eden v1.2**	**Eden v2.0**
S.O	CentOs 7.2	CentOs 7.3
C++	4.8.5	4.8.5
Python	2.7.5	3.6.2
Healpix-C++	3.30	3.31
Gmock	1.7.0	in Google test 1.8
Fitsio	0.9.6	0.9.11

TABLA 1: ALGUNOS DE LOS REQUERIMIENTOS DEL ENTORNO DE EUCLID.

Si se desea una correcta configuración local del sistema, para el entorno de Euclid, primero se debe disponer de la distribución y versión, del sistema operativo requerido por Euclid instalada. La posterior configuración de cada uno de los requisitos necesarios, para el entorno de desarrollo antes de empezar a trabajar, requieren de un tiempo adicional, a las tareas que desarrollan los científicos que trabajan en el proyecto de Euclid, los cuales deberían emplear la mayor parte de su tiempo en hacer ciencia, no software.
Como se puede observar en la figura 1, el ciclo de vida de las aplicaciones del proyecto de Euclid, esta compuesto de las siguientes fases sobre la integración continua:

1. **A :** En esta parte principal del ciclo de desarrollo, se observa la primera fase de almacenamiento y control sobre las versiones de la aplicación, en repositorios Git. Estos repositorios están compuestos por dos ramas, en los que se encuentran la rama principal o maestra, y la rama de desarrollo. Aquí es donde el usuario realiza la primera actualización del código, sobre el desarrollo de sus aplicaciones.

2. **B :** Cada vez que se desarrolla una nueva versión, Estos repositorios se actualizan en Codeen, siguiendo el mismo patrón de ramas. Codeen es una plataforma que colabora con el proyecto de Euclid. Esta plataforma proporciona diferentes servicios basados principalmente en herramientas de integración continua y otras similares. Una vez en Codeen, cada actualización se compara con las versión anterior, y en caso de ser distinta realiza una nueva compilación de la aplicación, y posteriormente una ejecución de los tests de validación. Si los tests son pasados con éxito, se almacenan en repositorios YUM, como paquetes rpm.

3. **C :** Una vez compilados y almacenados en repositorios YUM, posteriormente son actualizadas en CernVM-FS. Este servidor del proyecto, es el encargado de permitir el acceso a los sistemas de ficheros compartidos, por todos los usuarios de Euclid. Aquí es donde se almacenarán las versiones debidamente testeadas

4. **D :** En esta última fase sobre el ciclo de vida de las aplicaciones de Euclid se pueden observar dos secciones. En la parte superior se puede observar SDC-X, el cual hace referencia al centro de datos, donde finalmente se lanzarán y se ejecutarán las aplicaciones, almacenadas en el sistema de ficheros compartido de CernVM-FS. En la parte inferior, se puede observar Lodeen, este módulo hace referencia a la maquina virtual que actualmente, es el recurso que ofrece Euclid, para acceder al entorno de desarrollo de la aplicación.

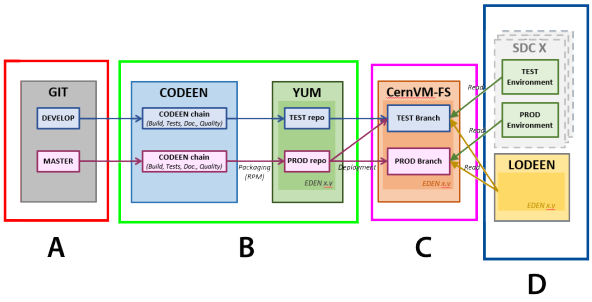

Fig. 1: Ciclo de vida de las aplicaciones del proyecto de Euclid.

En el caso de que el usuario, no pueda configurar el entorno especifico de desarrollo, en su host, ya sea por problemas de compatibilidad con el sistema operativo, u otros motivos, tiene la opción, como se observa en la figura 2, de hacer uso de Lodeen. Lodeen es el recurso actual que ofrece Euclid a sus desarrolladores. Se construye sobre una máquina virtual debidamente configurada para el proyecto, pero la virtualización sobre un hipervisor, tiene una gran desventaja, y es que provoca un gran consumo de recursos en el entorno local del usuario. Esta problemática, permite enfocar la propuesta de este proyecto, a buscar una herramienta que permita desarrollar las aplicaciones de Euclid, con una abstracción sobre la configuración y los recursos necesarios, por parte del usuario.

A continuación, se describen las dos problemáticas principales del entorno actual de Euclid, para las que se buscará una posible solución en este proyecto.

1. **Configuracion:** Se busca un entorno transparente a la configuración necesaria para Euclid, de acceso local,

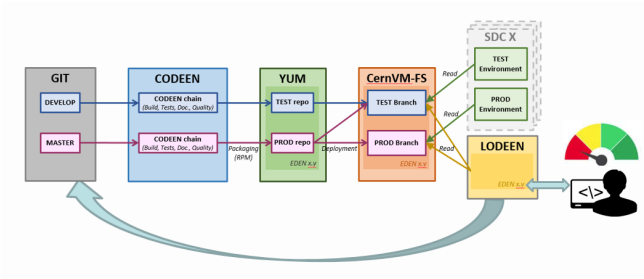

Fig. 2: Probemática sobre el consumo de recursos del entorno actual.

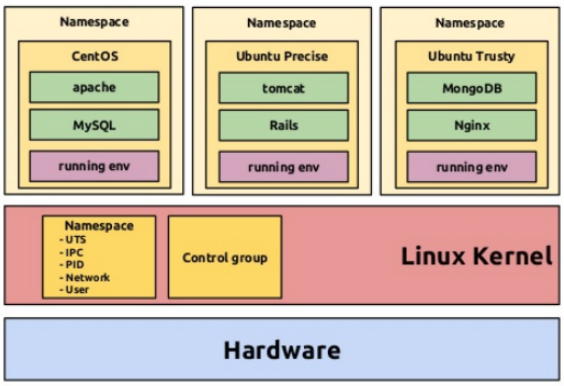

Fig. 3: Arquitectura de Lxc de Linux.

y que permita la integración de librerías o otras herramientas necesarias para el desarrollo de las aplicaciones. Es necesario, que sea independiente de la versión de sistema operativo utilizado por el usuario, y que permita el acceso a sistemas de ficheros remotos, con protocolos NFS y/o pNFS.

2. **Recursos:** Otro problema importante, en el actual desarrollo de las aplicaciones de este proyecto, se genera a consecuencia del gran consumo de recursos, que requiere la ejecución de la máquina virtual que actualmente proporciona Euclid, en un entorno local. Esto permite enfocar la integración del entorno de desarrollo de Euclid, con un IDE en el cloud, lo que permitiría una abstracción de los recursos necesarios por parte del sistema del usuario.

Con referencia a la configuración sobre el entorno de desarrollo, y en especial a los problemas que puede generar la versión de sistema operativo residente del usuario, esto proporciona un enfoque sobre una posible implementación de este entorno, mediante subsistemas aislados e independientes en el host anfitrión. Estos subsistemas, se crean con su propio espacio de usuario, sistema de ficheros, interfaces de red y versión del kernel compatible con los requisitos del proyecto. Este enfoque sobre el aislamiento de las aplicaciones ejecutándose en un mismo sistema, se puede implementar a través de la tecnología de contenedores. Los contenedores o subsistemas, en términos de consumo de recursos del sistema, son alrededor de tres veces más ligeros que una máquina virtual. Se crearon como entornos seguros, en los que un administrador podía compartir con diferentes usuarios, dentro o fuera de una organización, diferentes tipos de aplicaciones o servicios. El objetivo de estos subsistemas, estaba destinado a que los procesos, se crearan en un entorno modificado, en el que el acceso al sistema de archivos, a las redes y a los usuarios, estuviera virtualizado y no pudiera comprometer al sistema, de una manera genérica.
Más adelante, aparece LXC o contenedores Linux. Como se puede observar en la figura 3, está formado por grupos de control o cgroups, que son una función kernel que controla y limita el uso de recursos para un proceso o grupo de procesos. Además, los cgroups utilizan systemd, un sistema de inicialización que configura el espacio de usuario y gestiona sus procesos, para proporcionar un mayor control de estos de manera aislada. Otra parte importante de este tipo de contenedores, se observa en la parte superior de la figura 3, son los espacios de nombres de usuarios o namespaces, que permiten asignaciones por espacio de nombre de usuario y grupos, al sistema de ficheros del contenedor.

En el contexto de los contenedores, esto significaba que los usuarios y los grupos, podían tener privilegios para ciertas operaciones dentro del contenedor, sin tener esos privilegios fuera del contenedor. Esto era parecido a la primera idea sobre contenedores, pero con una seguridad adicional, que proporcionaba un mayor aislamiento de los procesos, en lugar del concepto de subsistema de un entorno modificado.
Una tecnología actual de contenedores es Docker [1] como se puede observar en la figura 4, Docker extiende el trabajo de LXC con herramientas mejoradas para los desarrolladores, como el motor de Docker o Docker engine, que facilita aún más el uso y la creación de contenedores, enfocados a las aplicaciones. Docker es una tecnología open source, y actualmente es el proyecto y el método más conocido para implementar y administrar contenedores. Una de las diferencias más significativas con los contenedores de Linux o LXC, es que Docker no ejecuta un sistema operativo dentro del contenedor, sino que está orientado a ejecutar lineas de comando dentro de un contenedor. Con relación a la potencia que nos ofrece la virtualización sobre contenedores y Docker, es posible usar este concepto, para la ejecución de algunos servicios en el cloud, ya que Docker tiene la capacidad de implementarse en varios servidores físicos, servidores de datos y plataformas en la nube. Esto es posible, por la independencia que proporciona el sistema entre los contenedores, y a su vez la cooperación entre ellos, para formar estructuras más complejas con intercambio de mensajes entre ellos. Se realiza, a través de una red interna que proporciona la tecnología de Docker. Con esta reflexión, se puede enfocar el proyecto de Euclid, hacia la posible integración de su entorno de desarrollo, en una tecnología de contenedores o Docker, y su ejecución en el cloud. Además esta idea podría permitir la integración de espacios de trabajo compartidos por los usuarios de Euclid, trabajando en una misma aplicación de forma paralela, y a tiempo real.

Con relación a los entornos de desarrollo que se ejecutan sobre contenedores Docker, y en un entorno de cloud, se encuentran distribuciones actuales como, Codeanywhere [2], Codenvy [3] o Eclipse Che [4]. Todas estas herramientas facilitan la codificación y ejecución de sus aplicaciones, permitiendo el acceso al entorno a través de un navegador web.
Con la idea de Docker, y la ejecución de servicios de desarrollo en el cloud mediante Docker, es posible realizar un

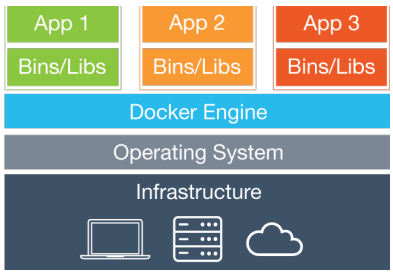

Fig. 4: Arquitectura de Docker.

nuevo enfoque, sobre la integración del proyecto de Euclid sobre esta tecnología. Se busca la posibilidad sobre la integración del entorno de desarrollo de Euclid ejecutado en un Docker, dentro del servicio de desarrollo, que se ejecutará en otro Docker. Aquí es donde se presentará la propuesta del proyecto, sobre el paradigma de "Docker en Docker".
Este proyecto, por una parte, se centra en la integración de un entorno configurado para el desarrollo de las aplicaciones de Euclid, dentro de un IDE en el cloud, y el cual de momento se ejecutará en un entorno centralizado. Posterior a esta idea, se realiza un estudio, sobre integrar este servicio, en una plataforma destinada a la orquestración o administración de contenedores en el cloud, a mayor escala. Esto se analiza con la finalidad de observar la escalabilidad del servicio en el caso que fuera necesario. Esto puede ser aplicable para esta, o cualquier otra aplicación de desarrollo en el campo de la ciencia, que requiera de cómputo de altas prestaciones o infraestructuras de Big data [5]. En la parte de obtención de resultados del documento se realiza un análisis sobre estas dos ideas, con la finalidad de obtener una estimación sobre los recursos necesarios, para la implementación de un servicio de estas características, así como permitir un mayor control sobre los contenedores, en un entorno de cloud.

2 Estado del arte

En este apartado se realiza un estudio, sobre las posibilidades actuales de servicios de desarrollo de aplicaciones en el cloud, que se ejecutan sobre imágenes de Docker.
En las siguientes tablas, se identifican las características más relevantes de las herramientas actuales ejecutadas bajo el paradigma de Docker, de manera que son posibles candidatas al enfoque para una posible solución al problema. Se pueden observar características como, temas de coste de licencias del software, o aspectos importantes en una herramienta de este tipo, como pueden ser la integración con repositorios Git, o soporte multi-usuario, este último, a efectos de proporcionar al usuario la capacidad de compartir espacios de trabajo en tiempo real.

Se pueden observar en la tabla 2 y 3, que las herramientas actuales son bastante completas. Se han seleccionado herramientas libres de licencias y de código abierto, ya que de esta manera permitirán la modificación del código fuente para su adaptación al entorno, en el caso de que fuera necesario. En el caso de este proyecto se utilizará Eclipse Che, ya que es un requisito por parte de la organización que supervisa el proyecto.

Con referencia a la ampliación y escalado del servicio, en la tabla 4, se observan diferentes herramientas que están

Característica	**CodeAnyWhere**	**Codenvy**
Precio	free	0-500
Auto completar	si	si
Multi-lenguaje	si	si
Git	si	no
Multi-plataforma	si	si
Depurador	si	si
Templates	no	si

TABLA 2: Servicios de desarrollo de aplicaciones en el cloud.

Característica	**CodeTasty**	**Eclipse Che**
Precio	free	free
Auto completar	si	si
Multi-lenguaje	si	si
Git	si	si
Multi-plataforma	no	si
Depurador	n/a	si
Templates	no	si

TABLA 3: Servicios de desarrollo de aplicaciones en el cloud.

disponibles en el mercado para la gestión de servicios basados en contenedores en el cloud. Una de las características más relevantes que se observan en la tabla 4, puede ser el aspecto relacionado con la escalabilidad de la plataforma y la administración de los contenedores en ejecución. Openshift [6] en una opción destinada a infraestructuras de mayor escala, ya que tiene un mayor consumo de recursos al implementar la plataforma de administración, y proporciona configuraciones enfocadas a balanceadores de carga, así como la posibilidad de generar replicas del servicio en otros nodos, si uno de estos se encontrará caído o en estado de parada por mantenimiento. También proporciona la capacidad para montar sistemas de ficheros distribuidos como GlusterFS [7]. Otra herramienta con menor complejidad, como se puede observar en la tabla 4 es Docker Swarm, no ofrece tantos recursos integrados destinados a la escalabilidad de la infraestructura y a la administración, pero es una buena solución para infraestructuras medias, ya que requiere de una menor complejidad por lo que a configuración se refiere, así como un menor consumo de recursos de sistema por parte de la infraestructura.

Característica	**DockerSwarm**	**Openshift**
Precio	free	free/pay
extensión de la API	moderada	amplia
Instalación	no compleja	compleja
Soporte	si	si
Escalabilidad	media	alta
Curva de aprendizaje	media	alta
Equilibrio de la carga	si	si
Monitorización	si	si

TABLA 4: Plataformas de orquestración de contenedores.

3 PROPUESTA

La propuesta de este proyecto, para una posible solución para Euclid, está dividida en dos implementaciones.
Se propone una primera implementación funcional de pruebas, sobre la integración del contenedor de Docker construido para la finalidad de Euclid, en el entorno de desarrollo en el cloud de Eclipse Che. Esta primera prueba se ejecutará en una infraestructura centralizada en un solo nodo. Eclipse Che, como se ha descrito anteriormente, se ejecuta mediante contenedores, tanto el servidor, como cada uno de los espacios de trabajo que ejecutan los usuarios.
Los contenedores en Eclipse Che, se ejecutan mediante la idea de Docker Compose [8], el cual usa un archivo YAML para configurar los servicios de su aplicación. Docker Compose, crea e inicia todos los servicios desde su configuración, para proporcionar un espacio multi-usuario, destinado a compartir espacios de trabajo entre los usuarios, en los que es posible ejecutar y comunicar diferentes aplicaciones sobre la administración de usuarios, como pueden ser bases de datos, o sistemas de autenticación de usuarios. Posteriormente se obtienen las estimaciones necesarias, sobre cual será la capacidad de los recursos de sistema necesarios, para cada usuario del proyecto en este primer entorno. A partir de los resultados sobre el comportamiento en un solo nodo, se presenta la idea sobre la ampliación del servicio, a dos nodos de cómputo. Esto se realiza a efectos de obtener un primer prototipo, para realizar un estudio sobre la viabilidad de portar el servicio a un entorno distribuido, y potencialmente escalable. Esta segunda implementación propone integrar todas las funcionalidades que proporciona el entorno de Euclid, sobre el servicio de desarrollo en el cloud, con una parte adicional dedicada al despliegue del servicio, en una plataforma de gestión de servicios destinada a infraestructuras de mayor escala.

3.1. Integración de Euclid sobre Eclipse Che

Para el desarrollo del proyecto de Euclid, se ha optado por la herramienta de código abierto Eclipse Che, que a diferencia de su versión de escritorio conocida por gran parte de la comunidad informática, esta versión de Eclipse, se presenta como un entorno de desarrollo en el cloud ejecutado como un servidor, dentro de un contenedor, y a través de un navegador web. Esto permite una abstracción de los recursos para el funcionamiento de la herramienta, así como de la configuración por parte del usuario del entorno local. Con referencia a la integración del entorno de Euclid sobre el servicio de Eclipse Che, Docker proporciona posibilidades adicionales, como la creación de contenedores a través de imágenes construidas por el usuario. Estas imágenes se construyen a partir de ficheros Docker o Dockerfiles [9]. Este método es el propuesto para la integración del entorno de Euclid dentro de Eclipse Che, construir una imagen de Docker con los requisitos sobre la versión del kernel, dependencias y librerías necesarias, entre otros parámetros. Una vez realizado este proceso se propone integrar esta imagen dentro del servicio de Eclipse Che, a través de la imagen personalizada de Docker. El objetivo de esta primera propuesta centralizada, es ejecutar el contenedor que contiene el entorno de Euclid, dentro del servicio de Eclipse Che, para proporcionar al usuario su entorno de desarrollo.

A efectos de proporcionar a todos los usuarios, su espacio de trabajo, y la capacidad de poder compartir sus proyectos, el servicio de Eclipse Che debe ejecutarse en modo multi-usuario, esto implicará la sincronización con otras dos aplicaciones, ejecutadas con la misma idea de contenedores. En esta modalidad, como se puede observar en la figura 5, juntamente al servidor de Eclipse Che, es necesario implementar un contenedor de Docker destinado al almacenamiento de los diferentes usuarios, y sus atributos en una base de datos de Postgres [10], y un contenedor del servicio de la herramienta de Keycloak [11], que proporciona al entorno la capacidad de administrar y validar la identidad de los usuarios pertenecientes al entorno de Eclipse Che.
Como se puede observar en la figura 5, el servidor de Che creará tantos contenedores como espacios de trabajo tengan los usuarios, y a través de la sincronización con un servidor de usuarios OpenLdap [12] y Keycloak, proporcionará al usuario un espacio de nombres mapeado a su identidad sobre OpenLdap.

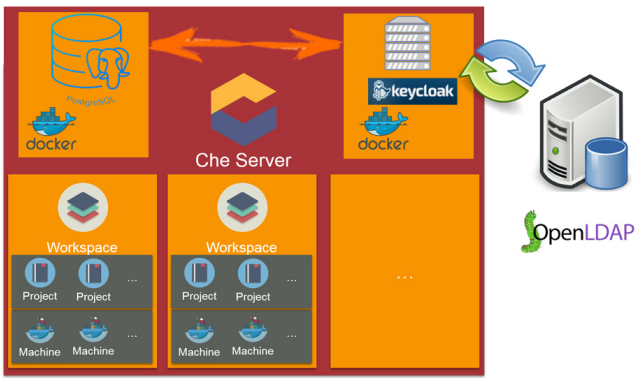

Fig. 5: Arquitectura de Eclipse Che en modo multi-usuario.

Como se muestra en la parte central de la figura 6, esta primera implementación se compone de un solo nodo (cheastro.pic.es), donde se ejecutará el esquema de la figura 5. Por último y no menos importante de este primer prototipo, como se muestra en la parte superior derecha de la figura 6, los espacios de trabajo que lo requieran, deberán disponer de la capacidad montar sistemas de ficheros externos a través de protocolos NFS y/o pNFS. Estos puntos de montaje, serán los encargados de permitir el acceso a los datos correspondientes para la ejecución de los tests de validación de la aplicación del usuario, los cuales se pueden observar resaltados en la parte derecha de la figura 6. Estos puntos de montaje deberán ser independientes por cada espacio de trabajo del usuario de Euclid.

3.2. Eclipse Che de Euclid sobre Openshift

En esta fase del proyecto, se define una propuesta sobre la viabilidad de portar el servicio de desarrollo de Eclipse Che, a un entorno distribuido. Esta portabilidad se analiza, a efectos de poder obtener una estimación, sobre los recursos necesarios para el usuario, al escalar el servicio, con esta o con otro tipo de aplicaciones con necesidad de ejecución en arquitecturas distribuidas, y con alta necesidad de almacenar y procesar datos. En los casos que se busca poner en producción un servicio a este tipo de escala, se debería hacer una estimación sobre todos los costes que supondrá la

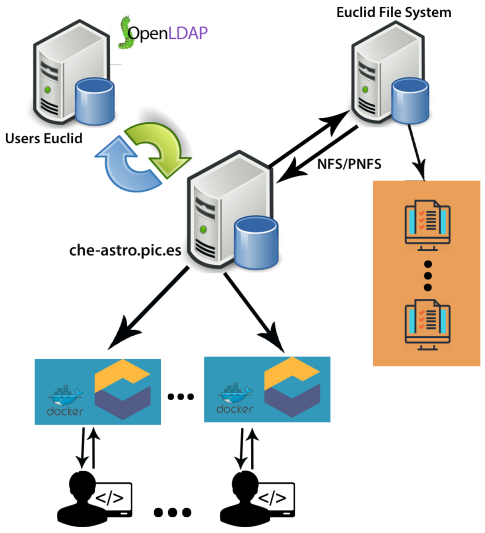

Fig. 6: Arquitectura centralizada sobre Docker.

instalación y el mantenimiento de la infraestructura necesaria, tanto en términos de personal como de tecnología. A menudo, dependiendo de la capacidad que tenga el despliegue del servicio, y con el fin de poder optimizar estos costes, podemos encontrar en el mercado actual diferentes soluciones de este tipo de servicios en el cloud. Algunas propuestas comerciales pueden ser AWS (Amazon Web Service) [13], o Azure de Microsoft [14]. Estas dos opciones comerciales como AWS o Azure, pueden ser muy adecuadas en según que casos. Estos proveedores de servicios en el cloud, ofrecen una arquitectura que tiene la capacidad de desplegar una plataforma de orquestación de contenedores, donde es posible ejecutar y administrar de igual manera, las instancias del servicio de Eclipse Che. Otra alternativa es implementar una de estas herramientas de gestión en un clúster propio y disponible. En este caso el administrador cargará con todos los procesos de instalación, configuración y mantenimiento. Para este estudio se propone desplegar el servicio sobre Openshift [6], debido a que por sus características sobre administración y robustez en entornos distribuidos, se ha decidido que es el más adecuado para la experimentación.
Openshift, se presenta como una plataforma como servicio (PaaS) desarrollada por RedHat, que actúa como una capa más de abstracción sobre la herramienta de administración de contenedores Kubernetes[15]. Openshift, proporciona una plataforma de gestión de estos servicios, y un entorno de desarrollo y test de las aplicaciones, que permite al desarrollador, crear y actualizar servicios que funcionen a través de internet.
Cabe decir que Openshift ofrece diferentes alternativas para el despliegue de esta plataforma en el cloud. Con referencia al proyecto de este documento enfocado a entornos distribuidos, RedHat nos ofrece 3 versiones distintas de Openshift [6] para desplegar la plataforma.

1. **Openshift.io :** En esta entrega de Openshift, RedHat ofrece una plataforma de pago. Esta plataforma cuenta con la capacidad gestionar nuestros recursos en el cloud, ofreciendo un servicio de reserva de recursos bajo demanda y con un servicio de soporte mas eficiente que la herramientas libres del mismo fabricante. Esta versión también ofrece aplicaciones ya instaladas y configuradas en la plataforma, como puede ser Eclipse Che entre otras.

2. **Openshift Cluster Platform :** Una versión de pago, y muy parecida a la descrita anteriormente, es OCP. Esta entrega, a diferencia de la anterior, el administrador del servicio de Opneshift, debe hacerse cargo de los costes sobre los recursos del sistema. Esta fase se puede realizar a través de otras empresas que ofrezcan estos servicios en el cloud, como pueden ser Azure o AWS. Otra alternativa es montar la plataforma en un sistema propio.

3. **Openshift Origin :** La única que ofrece RedHat totalmente abierta y sin ningún tipo de coste, es Origin. El inconveniente de esta versión de la herramienta, es que a parte que el administrador del servicio debe hacerse cargo de todos los gastos de recursos, mantenimiento y configuración, en comparación con las versiones de pago, el servicio de soporte es muy limitado, y en consecuencia mucho más complejo de instalar y configurar.

En este caso, se ha escogido la opción de Openshift Origin [16], ya que es la versión totalmente libre. El centro de datos *Port d'informació científica* ofrece a este proyecto los recursos, y la infraestructura necesaria, para el desarrollo y evaluación de este servicio.
Una diferencia principal en las construcciones de Kubernetes y OpenShift son el controlador de entrada frente a la ruta y el enrutador. Estas entradas, son contenedores dentro de OpenShift, donde la ruta se utiliza para definir las reglas que se aplicarán a las conexiones entrantes. La construcción correspondiente en Kuberentes para el mismo propósito se denomina Ingress. Al ser desarrollado sobre Kubernetes, OpenShift proporciona más características funcionales. Como se puede observar en la figura 7, esta plataforma a parte de gestionar y desarrollar servicios en el cloud, nos proporciona recursos para tener la posibilidad de poder escalar y desplegar Openshift, en un entorno distribuido. Esto se describe de manera que permite la posibilidad de montar sistemas de ficheros distribuidos como GlusterFS, así como balanceadores de carga y/o bases de datos distribuidas como MongoDB.

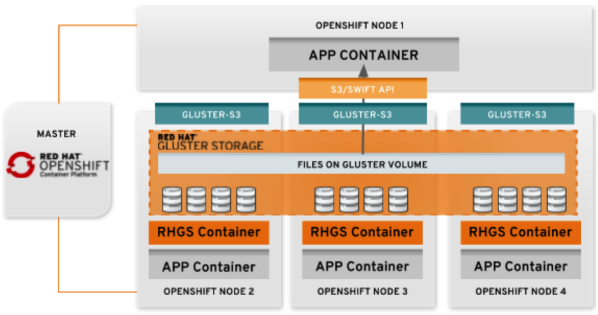

Fig. 7: Esquema de Openshift.

Después de haber descrito algunas propuestas disponibles para la actualización del servicio en un entorno distribuido, se presenta el diseño propuesto para esta implementación en dos nodos de cómputo. Como se puede observar en la figura 8, el diseño es parecido a la primera implementación sobre Docker. La diferencia entre este diseño y el anterior, es la plataforma de Openshift, la cual se encarga de gestionar los servicios en los nodos, y en este caso las entradas externas de datos, o los servidores de usuarios para sincronizarlos con Keycloak. Esta nueva infraestructura, es posible que suponga un incremento sobre el rendimiento del servicio. Este resultado no sorprende, debido a que los recursos de memoria y de cómputo son más del doble de la capacidad de su versión predecesora, de manera que sería lo esperado. La gran diferencia entre la primera versión sobre Docker en un entorno centralizado, y esta sobre Openshift, en uno distribuido, es la gestión del servicio de Eclipse Che en esta la plataforma que ofrece RedHat. En este caso, el administrador de Openshift tendrá la capacidad de gestionar de entre otros recursos disponibles, los siguientes destinados a la administración de Eclipse Che:

1. Creación y gestión de los contenedores que integran las aplicaciones que participan en el entorno multi-usuario (Keycloak y Postgres data base), así como los contenedores que ejecutan los espacios de trabajo de los usuarios.

2. Configuración de la red, y el enrutamiento de las aplicaciones para su correcto funcionamiento a través de un navegador web.

3. Asignación de nombres de dominios y subdominios de las aplicaciones, y resolución de nombres de DNS.

4. Montaje de volúmenes de datos necesarios de acceso compartido entre todos los nodos del sistema.

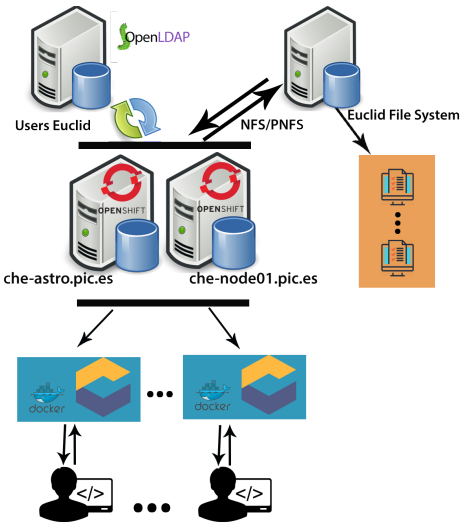

Fig. 8: Arquitectura del servicio implementado sobre Openshift.

Como se puede observar en la figura 9 El diseño del servicio ejecutado en un entorno distribuido con la plataforma proporcionada por Openshift, es probable que muestre una evolución sobre administración y la escalabilidad del proyecto de Euclid, si esto fuera necesario. El objetivo de este diseño distribuido, es obtener un sistema potencialmente escalable y robusto del servicio de Eclipse Che, con una mejor administración para entornos a gran escala. Este prototipo podría estar destinado tanto al entorno de Euclid, como a otras aplicaciones científicas orientadas al Big data.
Una característica importante sobre la implementación de un servicio, en este tipo de infraestructuras, es la alta disponibilidad que ofrece. Esto puede ser significativo, cunado un servidor de la infraestructura, y que ejecuta el servidor de Eclipse Che, o otro servicio complementario a él (Postgres y Keycloak), se encuentra fuera de servicio. Esta situación se puede producir debido a mantenimiento del servidor, o por un fallo de este. En estos casos Openshift, proporciona al administrador la capacidad de replicar el servicio a otro nodo activo, consiguiendo así, una pérdida prácticamente imperceptible por parte del usuario de la aplicación.

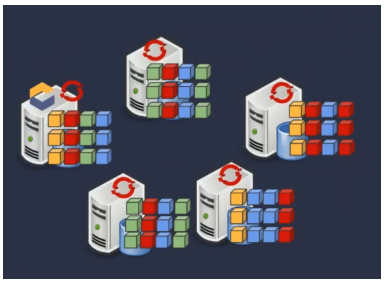

Fig. 9: Arquitectura final de la propuesta sobre Openshift.

4 Experimentación y resultados

A partir de la propuesta inicial del diseño, sobre el servicio de desarrollo en el cloud de Euclid, se despliega una primera versión centralizada de Eclipse Che sobre un entorno de Docker, en una modalidad multi-usuario. Dentro del servidor de Eclipse Che, que se ejecutará en un contenedor de Docker y con una imagen adaptada al proyecto de Euclid, construida a partir de un fichero de Docker o Dockerfile. La metodología de ensamblaje y construcción del Dockerfile como se puede observar en la figura 10 está compuesta de 3 fases.

1. Codificación

2. Construcción

3. Integración

Como se puede observar en la parte superior de la figura 10, en la fase de codificación, se realiza la escritura o receta del fichero, que posteriormente servirá para construir la imagen. Esta codificación, se realiza con unos estándares que recomienda Docker:

1. Debe empezar con la directiva FROM, que definirá la versión del kernel del sistema operativo con el que construirá el contenedor de Docker.

2. Definir parámetros como el autor y administrador de la imagen (MAINTAINER).

3. Definir todas las dependencias que deberá tener la imagen mediante el comando RUN.
4. Pueden establecer directorios del usuario, o copia de ficheros a la aplicación, en caso que sea necesario.
5. Pueden establecerse entrypoints o CMD. Los entrypoints o puntos de entrada, configuran el contenedor para que tenga la capacidad de ejecutarse como aplicación para el usuario, de manera que podrá ser utilizada para ejecutarse en diferentes espacios, por diferentes usuarios.

En la fase 2 de la figura 10, se construye este ensamblaje a través de una llamada al sistema, que ejecuta el servicio de Docker destinado a la construcción de imágenes.

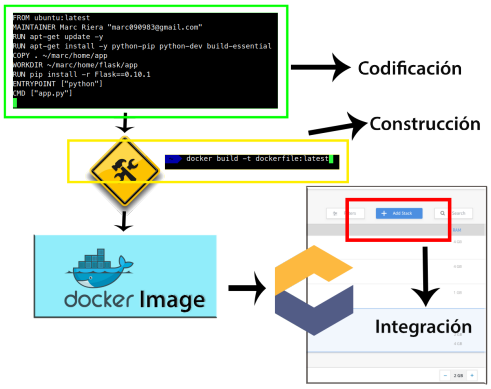

Fig. 10: Proceso de ensamblaje y construcción de la imagen de Docker.

Por último en la fase 3 de la figura 10, se observa, como una vez construido el contenedor que ejecutará el entorno de Euclid, solo falta integrarlo dentro de Eclipse Che. Este proceso se realiza a través de la interfaz web del servicio de desarrollo, enlazando el identificador de la imagen dentro del sistema, con el entorno de Eclipse.

4.1. Versión centralizada sobre Docker

En esta primera implementación del servicio, después de generar la imagen personalizada de Euclid, sobre un contenedor de Docker y a partir de la codificación de un Dockerfile, se realiza su integración en el entorno de Eclipse Che. A continuación se describen las tareas de aspecto más relevante de esta versión del proyecto.

1. Configuración de permisos en SElinux, y configuración del servicio de firewall del sistema, de esta manera permitirá ejecutar el servicio correctamente.
2. Configuración del servicio de Docker.
3. Creación del fichero de entorno de Eclipse Che, para pasar los parámetros necesarios a la ejecución del servicio de Eclipse Che.
4. Ejecución del servicio de Eclipse Che sobre el entorno multi-usuario.
5. Integración y sincronización del servidor externo OpenLdap, con la herramienta de Keycloak, esto permite habilitar el servicio para todos los usuarios del proyecto.
6. Almacenamiento de los usuarios sincronizados en Keycloak, en la base de datos Postgres, creada en Openshift.
7. Construcción de la imagen de Docker, de acuerdo a los requerimientos de Euclid. Esta tarea se realiza, a efectos de integrar Euclid en el entorno de Eclipse Che por defecto.
8. Montar sistemas de ficheros NFS y pNFS sobre el espacio de trabajo del usuario.
9. Ejecutar tests de validación, con entradas de datos accesibles desde sistemas de ficheros remotos, con protocolos NFS y/o pNFS.
10. Obtención de resultados.

4.2. Versión distribuida sobre Openshift

En esta parte del desarrollo del proyecto, se realiza la portabilidad del entorno de Euclid integrado en Eclipse Che, a un entorno distribuido y gestionado en una plataforma de administración de servicios en el cloud, sobre contenedores. La plataforma de Openshift, se ajusta al objetivo del proyecto, ya que utiliza como base del servicio, el paradigma de Docker. Como se ha descrito anteriormente, la versión escogida para este proyecto es Openshift Origin. Los requisitos mínimos según el fabricante para ejecutar esta plataforma son los siguientes.

Che-astro.pic.es	
S.O	Centos 7
Num. CPU	4
Memory	16 GB
Disk	60 GB

TABLA 5: RECURSOS MASTER.

Che-node01.pic.es	
S.O	Centos 7
Num. CPU	4
Memory	8 GB
Disk	60 GB

TABLA 6: RECURSOS NODE01.

En esta parte del desarrollo, primero se deben configurar los nodos en los que se realizará la instalación del servicio de Openshift. La fase de configuración de los nodos se realiza a través de un estándar sobre la automatización de la infraestructura y el software. Para esta acción Openshift en las versiones anteriores a la 3.0, usaba Puppet como herramienta para este fin. A partir de la versión 3.0, RedHat confía en Ansible [17]. Las dos herramientas, proporcionan la capacidad de configurar diferentes nodos a través de ficheros de configuración, que se propagan por los nodos asignados para la instalación y configuración de Openshift.

4.3. Evaluación del sistema centralizado

En este apartado se realizan las primeras pruebas sobre el entorno centralizado. Esta implementación integra la construcción de un Docker con los requisitos de Euclid, y su integración en un entorno de desarrollo en el cloud como Eclipse Che, que se ejecuta en otro contenedor de Docker.

4.3.1. Rendimiento del sistema

El primer caso de prueba se realiza sobre la respuesta al consumo de recursos del sistema. Este proceso es realizado por 2 usuarios activos del proyecto de Euclid. Estos se encuentran en la fase de ejecución de las aplicaciones de test que debe soportar el sistema, juntamente con la entradas de datos a través de los puntos de montaje NFS y pNFS del usuario.
Se puede observar en la figura 11, como el incremento es prácticamente de forma lineal en términos de consumo de recursos de CPU, es decir, a medida que los usuarios empiezan a ejecutar sus aplicaciones, se puede apreciar un consumo de una cuarta parte de los recursos totales del sistema por cada usuario. Esto en aspectos de recursos asignados a Docker, implica tener un consumo interno del total de los recursos de cómputo asignados a cada contenedor. También se puede observar en la figura 11, como el modo de ejecución en modo supervisor es mayor que el modo usuario, esto puede ser debido a que el acceso a los sistemas de ficheros externos, requieran de procesos con privilegios de sistema. Haciendo una estimación aproximada, y suponiendo que los otros recursos no se encontraran en estado de saturación, la capacidad de cómputo del sistema soportaría alrededor de 4 usuarios activos ejecutando este tipo de aplicaciones.
Llegado a ese punto el recurso de computo del sistema podría suponer un potencial cuello de botella.

En la figura 12, en relación al recurso de memoria principal, también se puede observar como el consumo de esta, en general aumenta entorno a los 2GB por usuario de una forma lineal. Esto implica que en términos de memoria asignada a Docker, esta estaría al limite de su capacidad. También se puede observar como el consumo sobre el recurso de memoria principal, tiene un aumento prácticamente proporcional al consumo que tenía el recurso de cómputo, es decir soportaría entorno los 4 usuarios simultáneos.

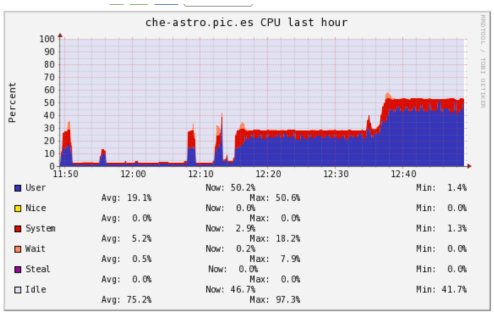

Fig. 11: Consummo de recursos de cómputo.

Otro aspecto relevante, como se puede observar en la figura 13, es el flujo sobre el paso de mensajes, o uso del recurso de red. Como puntos a destacar se pueden ver los picos de entradas y salidas de datos, esto podría ser a consecuencia de la entrada de datos a través de los sistemas de ficheros externos con protocolos NFS y pNFS.

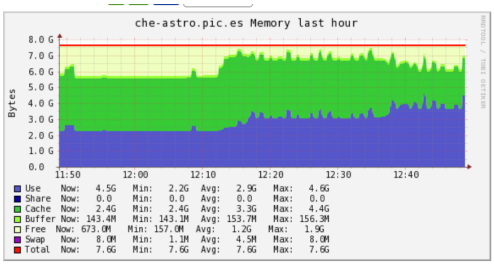

Fig. 12: Consumo de recursos de memoria.

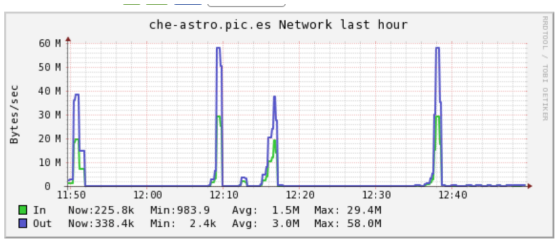

Fig. 13: Consumo de recursos de entrada y salida de datos.

4.3.2. Rendimiento sobre la disponibilidad del sistema

Este caso de prueba, se realiza para determinar la saturación del recurso de memoria principal, con usuarios conectados al servicio al mismo tiempo.
En este caso se realizan las pruebas del sistema para obtener una estimación sobre la reserva de recursos que realiza el sistema, y que es necesaria para soportar los usuarios llamados inactivos. Estos se identifican por encontrarse en una fase de desarrollo de su aplicación dentro del servicio, pero sin realizar ningún tipo de ejecución.

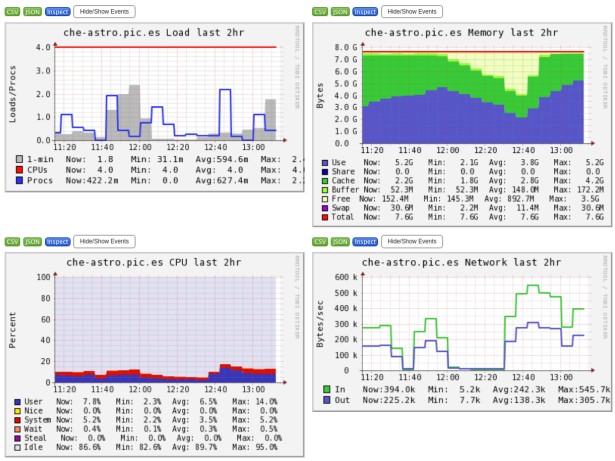

Fig. 14: Monitorizacion de recursos de sessión del usuario.

Como aspectos a destacar de la figura 14, se observa en la parte inferior izquierda de la como el recurso de cómputo tiene un consumo prácticamente mínimo. De igual manera en la misma sección se puede observar como aumentan los procesos ejecutados por el núcleo en comparación a la anterior, esto puede ser debido al gran número de contenedores creados por el sistema, y que se ejecutan en modo

privilegiado. También se puede observar en la parte inferior derecha, como aumenta el consumo del recurso de red, esto puede provocarse debido a la comunicación que se genera con la ejecución de múltiples contenedores sobre el entorno de Eclipse Che, y en su modo compartido multi-usuario. Por último en la parte superior derecha, se observa como el recurso de memoria, va recibiendo peticiones de reserva por parte de los usuarios, y se consume de forma lineal, a medida que los usuarios del servicio inician sesión y crean sus espacios de trabajo.

4.4. Evaluación del sistema distribuido

En la evaluación del sistema distribuido se analiza el consumo de recursos de la plataforma y sus servicios en los respectivos nodos, así como los recursos consumidos por los espacios de trabajo de los usuarios, creados en Eclipse Che. Se puede observar en la figura 15, como el nodo maestro (che-astro), es el encargado de realizar toda la reserva de memoria para gestionar los servicios ejecutados en Openshift, en este caso realiza una reserva total de memoria, destinada a buffers y memorias rápidas o cachés. También se observa como los 2 espacios de trabajo en ejecución, junto con toda la gestión del nodo maestro, respecto al servicio de Openshift, consumen un 75 % de los recursos de memoria principal de este nodo.

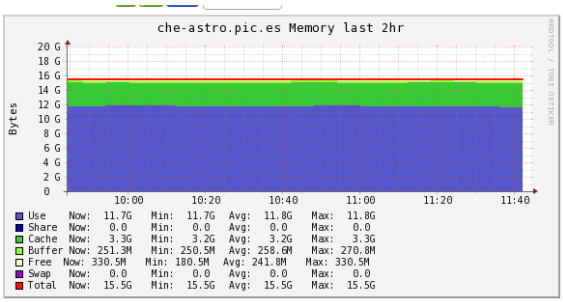

Fig. 15: Consumo de recursos de memoria.

En la figura 16, se observa como el recurso de red del nodo maestro, se encuentra permanentemente activo. Esto puede ser debido a que la comunicación de este nodo, con el nodo de cómputo, es necesaria en todo momento. Los servicios de Eclipse Che, en este caso se ejecutan en el nodo de computo, pero el nodo maestro, es el encargado de recibir las peticiones del servicio como punto final por parte del usuario, y posteriormente redireccionarlas al nodo que contiene el servicio. El nodo maestro también es el encargado de ejecutar la interfaz web, destinada a la administración de los contenedores en Openshift.

La otra parte sobre la ejecución de la plataforma de Openshift, se encuentra en el consumo de recursos por parte del nodo de cómputo (che-node01). Este nodo es el encargado de ejecutar los servicios del servidor de Eclipse Che, Keycloak y la base de datos de Postgres. En la figura 17, se puede observar como el recurso de memoria del nodo de cómputo queda limitado a la mitad de su capacidad, solamente con la puesta en funcionamiento de la plataforma de Openshift y el servicio de Eclipse Che, con sus respectivos servicios complementarios que permiten la ejecución de un entorno multi-usuario. Esto puede dar una ligera idea del

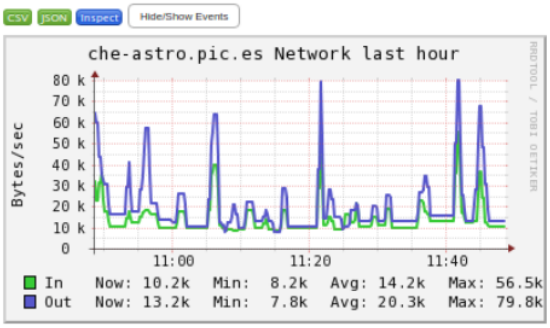

Fig. 16: Consumo de recursos de entrada y salida de datos.

alcance a nivel de servicio que puede abarcar la puesta en funcionamiento de esta plataforma de RedHat.

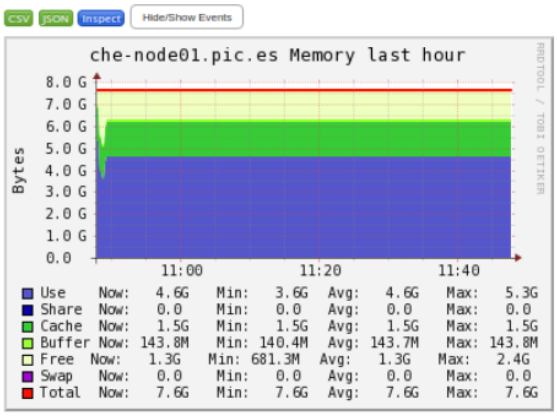

Fig. 17: Consumo de recursos de memoria.

Por último se puede observar en la figura 18, como la entrada y salida de datos en el nodo de cómputo es más constante que en el nodo máster, sobretodo en los datos de entrada. Esto puede ser debido a que los servicios que forman Eclipse Che en su entorno multi-usuario, están en permanente intercambio de información con el maestro, sobretodo en lo que refiere a datos de entrada, que son enviados desde el maestro, para responder a las peticiones del servicio por parte del usuario.

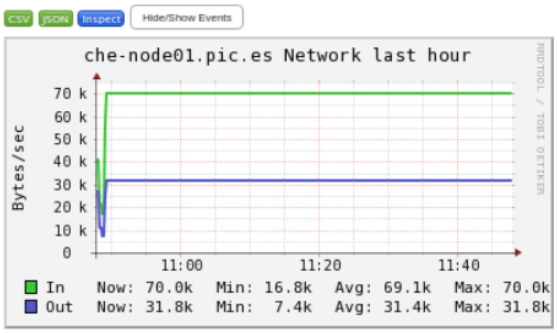

Fig. 18: Consumo de recursos de entrada y salida de datos.

4.5. Centralizado vs Distribuido

Si se realiza una pequeña comparativa sobre la evolución del servicio de Euclid, se puede observar como la versión centralizada sobre Docker (Fig. 19), al no necesitar de recursos por parte de una plataforma sobre la gestión y administración de contenedores en el cloud, permite una escalabilidad mucho más, lineal que en el entorno distribuido. La desventaja es que en este tipo de infraestructuras centralizadas sobre Docker, en el caso de un aumento considerable de usuarios, o de la necesidad de ejecución de aplicaciones de mayor consumo de recursos, podría quedar limitado.

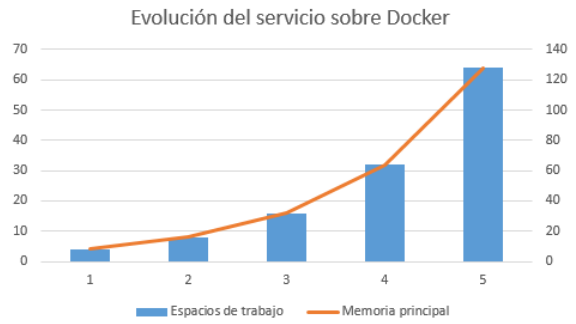

Fig. 19: Evolución del servicio en un entorno centralizado sobre Docker.

En la evolución sobre la implementación del servicio en un entorno distribuido gestionado por Openshift, se puede observar en la figura 20, como la instalación de la plataforma de RedHat, consume muchos mas recursos que en el caso de Docker. Esto se observa en que los 16 Gbytes de memoria principal del maestro, y 8 Gbytes del nodo de cómputo, permiten solamente la ejecución de 4 espacios de trabajo por parte de los usuarios, es decir, necesita más del doble de la capacidad de memoria principal, que la implementación sobre Docker, en un solo nodo, con 8 Gbytes. La ventaja de construir el servicio sobre Openshift, se encuentra en la potencia que ofrece, tanto en administración, como en escalabilidad, una vez son reservados los recursos destinados a la infraestructura.

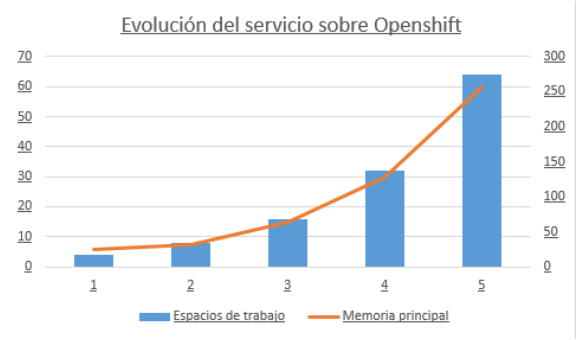

Fig. 20: Evolución del servicio en un entorno distribuido sobre Openshift.

5 Conclusiones

En primer lugar, remarcar la potencia de Docker en la necesidad de este proyecto. Estos contenedores, han permitido la construcción de un entorno para Euclid, con independencia del sistema operativo del ususario, para su posterior integración en un IDE en el cloud. Después de la puesta en práctica de los dos entornos en los que se ha desarrollado el proyecto, se puede decir que el proyecto de Euclid, debido a que la cantidad de usuarios que por el momento están dedicados a este tipo de desarrollo de aplicaciones, y que debe soportar el sistema, son alrededor de unos 50, no necesita de una gran infraestructura que lo soporte, de manera que un único nodo podría ser construido con 128 Gbytes de memoria principal y 32 unidades de cómputo, y cubriría el servicio a los actuales usuarios. Otra conclusión surge a raíz de la implementación del servicio de Eclipse Che en la infraestructura de Openshift. Se podría decir que esta plataforma esta más enfocada a aplicaciones con mayor necesidad sobre el consumo de recursos, como podrían ser aplicaciones HPC. Esto se puede deducir, debido a la cantidad de herramientas enfocadas a grandes infraestructuras que permite integrar, como pueden ser sistemas de ficheros distribuidos como GlusterFS, así como bases de datos distribuidas como MongoDB. De esta manera, al proporcionar la capacidad de montar sistemas de ficheros distribuidos, proporciona una visión general del sistema, lo que podría permitir un menor tiempo de acceso a los datos, y en consecuencia un menor tiempo de ejecución de las aplicaciones. También se puede observar la gran cantidad de parámetros que son posibles configurar en Openshift, en el momento de su instalación, así como la integración de contenedores destinados al enrutamiento de las aplicaciones y a la resolución de nombres de DNS. Esto refleja un poco la complejidad que requiere la instalación y la configuración de esta plataforma, y su integración con servicios de desarrollo en el cloud [18].

Agradecimientos

Trasladar mi más sincero agradecimiento, a todo el equipo de grandes profesionales que trabajan en el Pic (*Port d'informació científica*), por permitirme utilizar sus instalaciones y su infraestructura, para poder desarrollar este proyecto. También agradecer todo el soporte y la motivación, que me han brindado todo el equipo de administradores de servicios del Pic, a lo largo de todo el trabajo realizado, y en especial a mi tutor Francesc Torradeflot.

Referencias

[1] Harrison John Bhatti Babak Bashari Rad and Mohammad Ahmadi. *An introduction to docker and analysis of its performance.* Technical report, Asia Pacific University of Technology and Innovation Technology Park Malaysia, Kuala Lumpur, Malaysia, March 2017.

[2] CodeAnyWhere. Url: https://codeanywhere.com/. Technical report, CodeAnyWhere IDE, 2018.

[3] Codenvy. Url:https://codenvy.com. Technical report, Codenvy, 2018.

[4] Eclipse che. Url: https://www.eclipse.org/che/features. Technical report, Eclipse, 2018

[5] Nada Elgendy and Ahmed Elragal *Big Data Analytics: A Literature Review Paper.* Department of Business In-

formatics and Operations German University in Cairo (GUC), Cairo, Egypt

[6] RedHat. *Openshift container platform 3.5 installation and configuration.* Technical report, Redhat 2018, 2018-02-07.

[7] Cloud Storage for the Modern Data Center. *An Introduction to Gluster Architecture.* Versions 3.1.x Gluster.

[8] Shaun McCullough *Using Docker to Create Multi-Container Environments for Research and Sharing Lateral Movement Scenarios.* june, 2017 Sans Institute.

[9] Fawaz Paraiso, Stéphanie Challita, Yahya Al-Dhuraibi, Philippe Merle *Model-Driven Management of Docker Containers.* HAL Id: hal-01314827

[10] Jeff McCormick https://blog.openshift.com/deploying-postgresql-pod-openshift-v3/
november 6, 2014 RedHat Openshift

[11] Shoubhik Bose. https://medium.com/@sbose78/running-keycloak-on-openshift-3-8d195c0daaf6 Jun 16, Keycloak.

[12] V.Koutsonikola and A.Vakali. *Ldap: framework, practices, and trends.* Technical report, Aristotelian Univ. of Thessaloniki, Greece, 27 September 2004

[13] Ryan Cook and Scott Collier. *Deploying openshift container platform 3.5 on amazon web services.* Technical report, AWS Amazon web services and RedHat., 2018.

[14] Sherif Khattab Eman Hossny and Fatma Omara. *A case study for deploying applications on heterogeneous paas platforms.* Technical report, Comput. Sci. Dept., Cairo Univ., Cairo, Egypt, 26 May 2014

[15] David Bernstein. *Containers and cloud: From lxc to docker to kubernetes.* Technical report, Cloud Strategy Partners, Sept. 2014.

[16] Openshift Origin 3.9 Redhat
URL: https://docs.openshift.org/3.9/welcome/index.html

[17] Ansible 2.5 Ansible Documentation
URL: https://docs.ansible.com/ansible/latest/

[18] G. Fylaktopoulos, G. Goumas2, M. Skolarikis, A. Sotiropoulos and I.Maglogiannis *An overview of platforms for cloud based development* SpringerPlus, SpringerOpen journal 2016.