

Migración de una aplicación web de gestión de pacientes nefrológicos a la nube usando contenedores

F. Gómez Romero, I. Román Martínez^{1,2}, J. Calvillo Arbizu^{2,3}, L.M. Roa Romero^{2,3}

¹ Departamento de Ingeniería Telemática, Universidad de Sevilla, España, isabel@trajano.us.es

² CIBER- BBN

³ Grupo de Ingeniería Biomédica, Universidad de Sevilla, España, lroa@us.es

Resumen

En este trabajo se exponen los primeros pasos realizados para migrar la aplicación web e-Nefro, resultado de trabajos previos del Grupo de Ingeniería Biomédica de la Universidad de Sevilla, a una plataforma cloud en la modalidad IaaS (Infraestructura como servicio). Se utilizan contenedores como tecnología de virtualización y se presentan varios escenarios posibles, analizando ventajas e inconvenientes de cada uno de ellos.

1. Motivación

En proyectos previos del Grupo de Ingeniería Biomédica [1,2] se ha desarrollado un servicio Web, e-Nefro, para la gestión de Pacientes Nefrológicos. Dicho servicio se gestiona actualmente en una máquina dedicada, que incluye todos los componentes del servicio y que permite servir únicamente a un centro sanitario.

Este diseño ha resultado adecuado para trabajos anteriores y no ha dado ningún problema, sin embargo actualmente se plantean dos nuevos retos.

Primero; En uno de los proyectos que actualmente el grupo está desarrollando hay varios centros sanitarios implicados, este diseño fuerza a desplegar una máquina virtual para cada uno de los centros. De este modo los datos almacenados por cada uno de ellos se encuentran en bases de datos separadas (aunque de estructura idéntica). Esto no supone ningún inconveniente, dado que podemos hacer a posteriori la integración de datos para su análisis, pero utilizar una base de datos compartida podría optimizar el proceso de análisis que se debe realizar.

Segundo; Si pensamos en la posibilidad de escalar el servicio, más centros y usuarios, sería conveniente una solución que permitiera la elasticidad del servicio, optimizando el uso de recursos según las necesidades reales en cada momento. Al mismo tiempo sería necesario asegurar las prestaciones no funcionales del sistema a medida que éste crece (disponibilidad, fiabilidad, seguridad, eficiencia...).

En este punto se plantea la posibilidad de ofrecer el servicio utilizando tecnologías cloud y se comienzan a estudiar las posibilidades. En este trabajo se presentan los primeros pasos, la implementación de distintos prototipos que han permitido explorar las posibilidades que las tecnologías de contenedores, y las herramientas de orquestación de éstos,

ofrecen para realizar el despliegue en nube, según el modelo IaaS.

2. Estado de la técnica

2.1. Contenedores

Un contenedor es una tecnología de virtualización de un sistema operativo con un sistema de ficheros minimizado para ejecutar un único servicio o programa. Aunque los contenedores comparten el kernel de la máquina en la que se están ejecutando, los procesos que corren dentro de ellos actúan de forma aislada al resto del sistema, reservando en exclusividad recursos como CPU o memoria [3]. Así es más sencillo aumentar o reducir los recursos destinados a cada “microservicio”, de manera independiente a los demás.

Los contenedores van sobre el sistema operativo, compartiendo directamente el kernel, mientras que una máquina virtual necesita un hipervisor, que monta por encima un sistema operativo completo. De este modo los contenedores están diseñados para ser más ligeros y necesitar menor tiempo de despliegue que las máquinas virtuales, sólo necesita las librerías y binarios que vaya a utilizar el servicio que ejecuta. Al no depender de un hipervisor se consumen menos recursos de la máquina principal y se obtiene mayor rendimiento. Al compartir el kernel las aplicaciones que van sobre contenedores se interconectarán más fácilmente que las que corren en diferentes máquinas virtuales.

Sin embargo también hay algunos inconvenientes. Las tecnologías de contenedores obligan a que los servicios y aplicaciones se desarrollen para el mismo sistema operativo, el aislamiento y seguridad entre aplicaciones corriendo en distintas máquinas virtuales es mayor que si se utilizan contenedores y, por último, el contenedor está pensado para tener una “vida efímera”, por lo que la persistencia de información es más compleja de gestionar.

Para desplegar una aplicación compleja será necesario utilizar herramientas que permitan administrar aplicaciones multi-contenedor, en muchas ocasiones distribuidos en diferentes máquinas. Esta tarea se conoce como orquestación de contenedores.

2.2. Computación en la nube

La computación en la nube [4] cambia el tradicional modelo en el que la infraestructura de sistemas de información y comunicaciones es instalada y mantenida en el local del cliente, a uno en el que esta infraestructura se externaliza, y se gestiona y mantiene por el proveedor en sus propios “mega” centros de datos. Una de las principales ventajas es que los recursos dedicados al cliente para soportar sus requisitos TIC se pueden adaptar a las necesidades específicas en cada situación, dedicando siempre justo los recursos necesarios. Esta característica se conoce como elasticidad y conlleva también un ajuste de los costes para el cliente.

El servicio ofrecido por el proveedor puede seguir diversos modelos entre los que destacan:

- Software como Servicio (SaaS): el proveedor ofrece servicios directamente al usuario final, a nivel aplicación, siendo el acceso más habitual a estas aplicaciones vía web.
- Plataforma como Servicio (PaaS): el proveedor ofrece un entorno preparado para que los clientes puedan desarrollar sus aplicaciones y ofrecerlas al usuario final. La plataforma proporciona facilidades como replicación, balanceo de carga, fiabilidad o persistencia a las soluciones desarrolladas.
- Infraestructura como Servicio (IaaS): el proveedor ofrece básicamente máquinas virtuales y almacenamiento, liberando al cliente sólo del despliegue y mantenimiento de la infraestructura física, además de algunas facilidades básicas como cortafuegos o copias de seguridad.

3. Selección de soluciones

3.1. Docker

Tras el estudio de diferentes tecnologías de contenedores se decidió utilizar Docker [5] en el desarrollo de este trabajo. Entre otros motivos por su creciente aceptación, la bibliografía disponible y su portabilidad a distintas plataformas.

Docker incluye una aplicación cliente y un host. El host alberga los contenedores y ejecuta órdenes (lanzar un contenedor, detenerlo, monitorizarlo...). El cliente será el que mande las órdenes por medio de una API REST o usando sockets. Así el cliente y el host pueden ejecutarse en máquinas separadas. El ciclo de vida de un contenedor va ligado a la ejecución del proceso lanzado.

Una imagen de Docker es una plantilla donde se especifican los detalles de un contenedor. Utilizar imágenes construidas por otros, directamente o personalizándolas, resulta menos costoso que crear una nueva. El número de imágenes de contenedores ya disponibles en Docker Hub [6] es otro de los motivos para usar Docker.

3.2. Azure

Aunque la solución para computación en nube de Microsoft, Azure [7], ha sido la última en adelantar

posiciones en contenedores, no ha publicado hasta 2016 su servicio denominado Azure Container Service (ACS), ha sido la elegida. El motivo es que se dispone de licencias educativas y crédito para que los alumnos puedan utilizar los recursos que ofrece la plataforma.

3.3. Kubernetes

El trabajo ha sido desarrollado usando Kubernetes [8] como tecnología de orquestación, una solución de código abierto proporcionada por Google. Esta decisión se toma porque actualmente está en un nivel de madurez superior al de otras alternativas y ofrece algunas facilidades para el despliegue en Azure.

Las funcionalidades principales van desde el escalado y actualización de los componentes a la optimización de los recursos según los requisitos. Consigue que el estado de cada uno de los contenedores, y de la aplicación en su conjunto, sea continuamente el deseado y especificado. Por lo que el sistema resulta robusto y fiable.

4. Escenarios desarrollados

4.1. Migración de e-Nefro a contenedores

E-nefro está desarrollada con servlets de Java y actualmente se está utilizando la solución de Apache, Tomcat, como contenedor de servlets. Para persistir la información se usa PostgreSQL como gestor de base de datos.

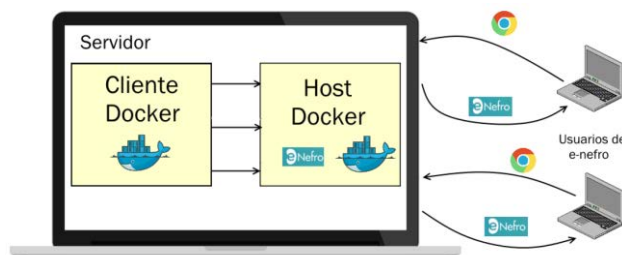


Figura 1. Arquitectura Docker en el primer escenario

En este primer escenario se ha automatizado el proceso de despliegue de E-nefro en un servidor local con contenedores Docker. Por simplicidad la misma máquina alberga el cliente y el host de Docker. Como muestra la figura 1.

Cada servicio que necesita la aplicación funciona en un contenedor diferente, lo que favorece la independencia entre ellos y la elasticidad de recursos. El sistema se comienza dividiendo en tres tipos de contenedores diferentes:

- El primero con Tomcat, que contiene el fichero enefro.war para el despliegue de la aplicación.
- El segundo con PostgreSQL, que contiene la base de datos que se inicializa con los ficheros sql de E-nefro.
- El tercero un contenedor de datos, que permite la persistencia de la información introducida aunque el contenedor de base de datos deba reiniciarse.

Se añade un cuarto con el fin de incluir un proxy inverso, intermediario del tráfico destinado a la aplicación. Éste incluye un servicio llamado Nginx [9], que se encargará de procesar todo el tráfico entrante para después redirigirlo a Tomcat. Nginx es un servidor web muy ligero, lo que lo convierte en el favorito para ser usado como proxy inverso. Este modelo aporta algunas ventajas entre las que destacan:

- Se puede distribuir el tráfico entre varios contenedores idénticos de Tomcat corriendo simultáneamente para evitar sobrecarga y proveer un servicio elástico que adapte sus recursos. En este caso se ha habilitado Nginx para que funcione como balanceador de carga usando el método por defecto (round-robin, que selecciona el contenedor destino de forma ordenada), pero Nginx permite definir otros algoritmos que optimicen el rendimiento del servicio.
- Se evita tener el contenedor de Tomcat expuesto al exterior, pudiendo establecer un punto único de autenticación y autorización para todas las peticiones entrantes.
- El proxy puede servir el contenido estático disminuyendo la carga para el resto de contenedores.
- Adicionalmente, puede funcionar como caché.

El Sistema final de contenedores se muestra en la figura 2, con el contenedor de Nginx recibiendo todo el tráfico y rediriéndolo a los contenedores de Tomcat.

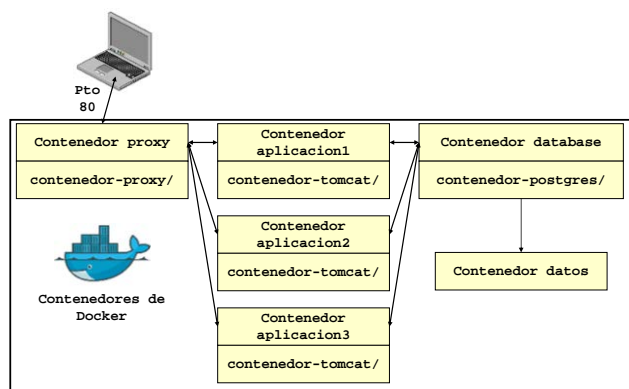


Figura 2.Detalle de la arquitectura de contenedores en la primera solución

Se ha utilizado la herramienta Docker Compose, que facilita la administración de una aplicación multi-contenedor. Ha sido necesario configurar el fichero docker-compose.yml incluyendo toda la información de cada uno de los contenedores; sus vínculos con los demás, volúmenes y el orden de ejecución. El resultado es que se reduce considerablemente el número de comandos a ejecutar para administrar la aplicación.

4.2. Despliegue de e-Nefro en la nube

El siguiente paso ha sido desplegar el escenario anterior en la plataforma Azure. Se lanza una máquina virtual Ubuntu en la que está instalado Docker, y los contenedores de la aplicación E-nefro se arrancan en el host siguiendo el procedimiento del apartado anterior, por medio de la herramienta en línea de comandos de Azure. El tipo de instancia que se ha creado por defecto es una A1 Standard,

que tiene un núcleo y 1.75 GB de memoria RAM. Azure proporciona herramientas de autoescalado del servidor, lo que permitirá que las especificaciones de CPU y memoria se adapten en tiempo real en función del tráfico.

Para estudiar las facilidades de la nube se han realizado varias pruebas modificando los recursos destinados a esta instancia, configurando el cortafuegos y creando una imagen, a partir de la máquina virtual con E-nefro funcionando.

En esta ocasión por tanto el host estará en la instancia de Azure mientras que el cliente Docker (para la gestión) estará instalado en una máquina local con conexión a internet, que necesitará tener instalada la herramienta de Azure (63).

La Figura 3 muestra como a través del cliente se ejecutan comandos, tanto usando la API de Azure para crear la máquina virtual o editar los endpoints, como con el host de Docker dentro de la instancia para construir las imágenes y lanzar contenedores. Para poner disponible la interfaz web para acceso al servicio se crea un endpoint en el puerto 80, por supuesto esta configuración le es totalmente transparente al usuario final.

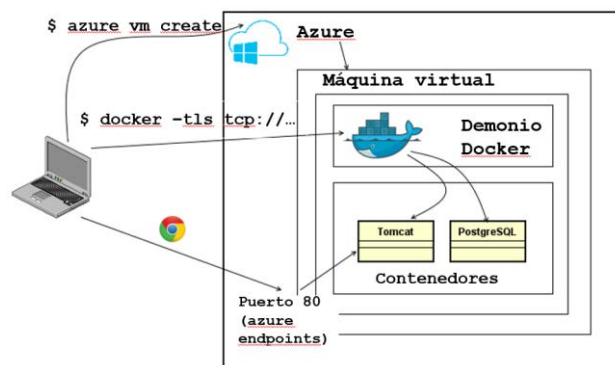


Figura 3.Arquitectura de la segunda solución

4.3. Despliegue de e-Nefro con Kubernetes

Los escenarios anteriores desplegaban E-nefro en un único servidor utilizando contenedores, primero de forma local y posteriormente en la nube, que permitía adecuar los recursos del servidor a las necesidades en cada momento (escalabilidad vertical).

En esta última solución se explota la escalabilidad horizontal, permitiendo añadir más instancias de servidores con el fin de mejorar el rendimiento, disponibilidad y fiabilidad. Para ello se ha hecho uso de la herramienta de orquestación de contenedores Kubernetes. Por medio de la replicación de pods se podrán reducir o aumentar en tiempo real el número de instancias de Tomcat o de PostgreSQL que se quieren en funcionamiento. En este caso se ha encontrado una limitación en la plataforma elegida: Kubernetes no ha implementado aún ninguna forma de lanzar volúmenes en Azure (aunque sí para las soluciones de Google y Amazon), de modo que lograr la persistencia se vuelve especialmente complejo.

Se monta primero el escenario, sin persistencia, que se presenta en la figura 4. Desde el master se controlan todas

las acciones. A través de la API el master mandará instrucciones a los kubelet de los dos nodos, y éstos crearán o destruirán los pods correspondientes. El proxy en este caso no es necesario ya que Kubernetes ofrece como funcionalidad el balanceo entre réplicas de pods.

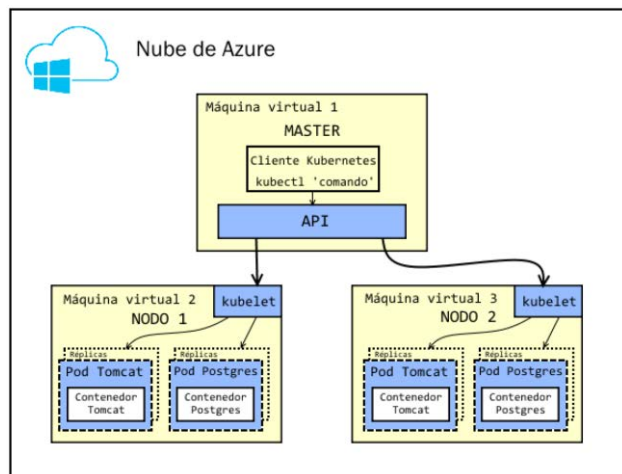


Figura 4. Arquitectura de la tercera solución

Para persistir la información en Azure se ha usado un volumen de tipo HostPath [10], que monta un directorio del nodo dentro del pod. Si se almacenan las tablas de la base de datos dentro de un directorio del nodo y este se monta cuando se inicializa un pod de PostgreSQL, se conservará la información. El inconveniente que acarrea esta implementación es que sólo se puede usar con un nodo.

5. Discusión y conclusiones

En este trabajo se han explorado las ventajas que pueden existir al desplegar e-Nefro en contenedores y en la nube y las dificultades que este proceso puede entrañar. Al separar cada componente en un contenedor se favorece la administración individual, permitiendo modificar los recursos destinados a cada uno según la demanda. La volatilidad en la vida de los contenedores hace que sean fácilmente reemplazables si aparece un error, evitando que se arrastre en el resto del sistema. El sistema resulta más ligero al incluirse tan solo las librerías y recursos que necesita cada servicio. Se pueden añadir nuevas funcionalidades de manera sencilla, como el proxy inverso, que permite el balanceo de carga.

Cuando se traslada el sistema de contenedores a una instancia en la nube se delega el mantenimiento de la infraestructura. Y nos beneficiamos de la elasticidad de la nube, pudiendo modificar los recursos de la máquina en tiempo real en función del tráfico. Creando una imagen cuando el sistema está en funcionamiento se puede clonar la aplicación en una nueva instancia.

Con una herramienta de orquestación como Kubernetes se aumenta el control sobre el sistema de contenedores. Este se encarga de monitorizar el estado de todos los servicios y mantener siempre la aplicación en funcionamiento. Cuando se despliega la aplicación en varias máquinas virtuales Kubernetes distribuye los contenedores entre

ellas de manera transparente y, si los servicios están replicados, se encarga de balancear.

La persistencia con contenedores ha resultado compleja al usar Azure como proveedor. Además, aunque Kubernetes ofrece scripts para el despliegue sencillo de un cluster en Azure, ha resultado complejo adaptarlo a necesidades específicas, como aumentar el número de nodos. Previsiblemente las prestaciones del servicio y su gestión mejorarían si se utilizara otra infraestructura de nube como la de Google o Amazon. Tras la finalización de este trabajo, ha sido publicado un servicio propietario de Azure para el despliegue de contenedores que puede facilitar esta labor.

Realizar el despliegue real de aplicaciones sanitarias en una infraestructura nube de proveedores externos requiere un análisis detallado de las implicaciones legislativas en cuanto a la seguridad y confidencialidad de la información sanitaria (como la LODP). El hecho de que la información esté almacenada en centros de datos del proveedor puede desaconsejar, o imposibilitar, este escenario totalmente externalizado. Con el fin de aprovechar las numerosas ventajas de estas tecnologías, se podrían utilizar nubes privadas o soluciones mixtas.

Agradecimientos

Este trabajo ha sido financiado por el Fondo de Investigaciones Sanitarias del Instituto de Salud Carlos III en el marco del proyecto DTS15/00195 (Evalua-Nefro).

Referencias

- [1] Roa Romero, Laura María, Calvillo Arbizu, Jorge, Milan Martin, Jose Antonio, Salgueira Lazo, Mercedes, Aresté Fosalba, Nuria, et. al.: Sistemas de e-Salud para pacientes crónicos. En: I + S. Informática y Salud. 2016. Vol. 115. Pag. 14-20
- [2] Calvillo Arbizu, Jorge, Roa Romero, Laura María. Aproximación metodológica al diseño de un sistema de teleasistencia para pacientes en pre-diálisis y diálisis peritoneal. En: Nefrología. 2014. Vol. 34. Núm. 2. 10.3265/Nefrologia.pre2014.Jan.12115
- [3] Linux containers isolation. [Online] <https://blog.engineyard.com/2015/linux-containers-isolation>. (último acceso Septiembre 2016)
- [4] Norma Y.3500 de la ITU-T: Cloud Computing overview and vocabulary. Disponible en <https://www.itu.int/rec/T-REC-Y.3500-201408-I>.
- [5] Documentación en línea de Docker. <https://docs.docker.com/> (último acceso Septiembre 2016)
- [6] Docker Hub. <https://hub.docker.com/> (último acceso Septiembre 2016)
- [7] Página principal de Azure: <https://azure.microsoft.com/es-es/> (último acceso Septiembre 2016)
- [8] Documentación en línea de Kubernetes. <http://kubernetes.io/docs/> (último acceso Septiembre 2016)
- [9] Documentación en línea de Nginx. <http://nginx.org/en/docs/> (último acceso Septiembre 2016)
- [10] Documentación en línea de Kubernetes HostPath. <http://kubernetes.io/docs/user-guide/volumes/#hostpath>. (último acceso Septiembre 2016)