Plataforma como servicio (PaaS) y software como servicio (SaaS)

(*platform as a service & software as a service*)

Remo Suppi Boldrito

PID_00247723

Tiempo mínimo previsto de lectura y comprensión: **9 horas**

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Los entornos conocidos como plataforma como servicio (**PaaS**) y software como servicio (**SaaS**) son sin duda los que más orientación a negocio/servicio empresarial propiamente dicho tienen, e intentan reducir el salto entre disponer de una plataforma (local o *cloud*) y ponerla en producción para el desarrollo, el despliegue o la provisión de servicios.

Cuando el usuario potencial llega al PaaS ya tiene todo el entorno preparado y configurado para comenzar a producir dentro de su ámbito de experiencia y no se preocupa por nada del resto; solo en desarrollar, probar, ejecutar y administrar sus aplicaciones. Esta orientación realiza una abstracción de toda la infraestructura software y hardware subyacente, donde el usuario no se ocupa en instalar/crear/mantener o administrar la infraestructura subyacente, la cual puede ser del propio prestador del servicio o puede ser a su vez un despliegue sobre una infraestructura de *cloud* público. En este caso el usuario dedica su tiempo y esfuerzo a su aplicación/desarrollo y sus datos.

En el caso del SaaS, la abstracción es mucho mayor ya que el usuario solo usa la aplicación/entorno por el cual tiene interés y no se preocupa ni por el software, ni por la infraestructura, ni por la gestión/administración de la aplicación que utiliza: solo de sus datos.

Dentro del paquete **PaaS** un usuario recibirá un conjunto de servicios/aplicaciones (*stack*) configurados, actualizados y optimizados para que pueda realizar sus desarrollos con garantías y dedicando su tiempo a lo que es su ámbito de interés. Dentro de este *stack* (y en forma general) se incluye: infraestructura física (cómputo, almacenamiento y red), sistema operativo, entornos software (*scripting*, gestor de BD, software de servidor, seguridad, almacenamiento seguro/fiable, herramientas de soporte/diseño-desarrollo/depuración/test, etc.), soporte y, en muchos proveedores, el *hosting* (capacidad para que las aplicaciones desarrolladas puedan ser desplegadas/alojadas sobre los mismos servicios donde fueron desarrolladas) [Ucc][Hcd][Mqp][Tre][WiP].

Todo esto representa un **beneficio notable** para los desarrolladores y las empresas (de aquí su visión de negocio), ya que no es necesario invertir ni en infraestructura física ni en obra civil (para alojar los servidores), ni en administradores de sistemas, ni en técnicos especializados en un tipo de arquitectura software. Si bien esto se puede tener también con un IaaS+ (por ejemplo, los *stacks* preconfigurados que ya disponen la mayor parte de proveedores de IaaS, que –muchos de ellos, como se ha podido ver en el módulo correspondiente–

ya no son solo proveedores de infraestructura, sino también de PaaS/SaaS), continúa siendo necesario personal de IT para configurar los *stacks*; sobre todo si estos son especializados a determinados paquetes.

La base del PaaS es «entrar y desarrollar» sin perder tiempo en instalar/configurar/mantener/ actualizar, que, como se sabe, implica gran cantidad de tiempo incluso para SysAdmin expertos, y obviamente bajo la política de pago por uso que permite no tener gran cantidad de activos inmovilizados antes o después de finalizado el desarrollo. Es decir, todo ello habilita a que usuarios no expertos en SysAdmin se puedan poner a producir en forma inmediata, ya que la mayoría de entornos dispone de una interfaz web donde, en poco tiempo, se puede tener el entorno de desarrollo listo para usar y sin grandes acciones/intervenciones y, además, facilitando la colaboración en línea entre grupos de trabajo para desarrollos con diferentes programadores/expertos. Un aspecto clave de estos entornos es la seguridad y el hecho de que, antes de contratar un servicio PaaS, se deberá verificar de qué tipo se dispone y si esto satisface los requerimientos del desarrollo, ya que tanto las aplicaciones como los datos compartirán infraestructura física y software con otros proyectos de otros clientes.

En muchos proveedores, con el fin de reducir costos, la arquitectura software generalmente es ***multitenancy***, que significa que se ejecuta una sola instancia software de la aplicación servidor y esta sirve a múltiples usuarios/clientes (*tenants*).

Esta arquitectura software compartida, aunque es más compleja tanto desde el punto de vista de la configuración como desde la seguridad y compartición de datos, permite un mejor aprovechamiento/eficiencia de la infraestructura hardware y reduce notablemente los requerimientos y, por lo tanto, los costos en relación con la otra arquitectura, donde cada cliente dispone de una instancia y solo es utilizada por él. En muchos casos, los proveedores permiten seleccionar si el cliente desea una arquitectura *multitenant* o no, y esto se ve reflejado en el costo de los recursos.

Es importante tener en cuenta que un entorno *multitenancy* es diferente a un entorno virtualizado, ya que en el primero los desarrolladores comparten la misma aplicación/servicio que se ejecuta sobre el mismo SO y mismo hardware con el mismo sistema de almacenamiento y donde la diferenciación entre los usuarios y sus datos es realizada en la infraestructura para que estos no puedan ver los datos de otros usuarios.

Obviamente, esto es una gran diferencia en relación con la virtualización, tanto de máquina como de contenedores, ya que cada cliente dispone de un entorno virtualizado de hardware/software subyacente (*sandbox*) a diferentes niveles.

Si bien la *multitenancy* permite el ahorro de costos (incluso en licencias; por ejemplo, no es necesario tener múltiples licencias de una BD, ya que solo se ejecutará una instancia que gestionará diferentes BD aisladas entre sí) debido a que se reducen los recursos necesarios para atender a otro cliente (memoria/CPU), presenta muchas dificultades en la escalabilidad y en mantener, para todos los usuarios, las mismas prestaciones, las cuales pueden verse seriamente afectadas cuando el número de clientes crece. Es por ello que se recomienda prestar atención a la SLA desde el punto de vista del cliente y desde el del proveedor tener un entorno de monitorización competente que permita detectar estas situaciones de sobrecarga y poder distribuirla hacia otros servidores menos cargados para no incurrir en violaciones a la SLA que luego puedan dar lugar a reclamaciones.

La evolución de las plataformas PaaS ha dado lugar a diferentes subcategorías como especializaciones o adecuaciones a un determinado grupo objetivo, por ejemplo, mPaaS (*mobile platform as a service*), la cual tiene por objetivo la provisión de un entorno interactivo de desarrollo (*interactive development environment*, IDE) para la creación de aplicaciones móviles o aPaaS (*application platform as a service*), donde se incluyen exclusivamente los servicios que ofrecen desarrollo y despliegue de aplicaciones, mientras que PaaS se refiere a todo el *middleware* como una oferta de servicios [Wma].

En un **SaaS**, el grado de abstracción es mucho mayor, ya que las aplicaciones son desplegadas en servidores del proveedor o en muchos casos en un *cloud* público y el usuario/cliente solo pagará (o tendrá un acuerdo de uso que en algunas plataformas es gratuito, pues el modelo de negocio viene dado por otra forma de pago, por ejemplo, publicidad) por su utilización.

> Por ejemplo, Adobe, Google, Facebook, Flickr, Office365 y Twitter son plataformas ejemplos de SaaS (o bien toda la plataforma o parte de ella); algunas disponen de una sección simplificada orientada a usuarios individuales y otra a usuarios corporativos que les permiten un grado más de configuración/adecuación/administración para acercarlas a las necesidades empresariales (detalles de costos, rediseño de la imagen corporativa, monitorización, integración con servidores corporativos; por ejemplo, para la autentificación, etc.).

El modelo operativo es muy simple ya que, por ejemplo, si se necesita un gestor de correo corporativo y toda la funcionalidad para los usuarios de la empresa, en lugar de comprar la infraestructura, software (si no *open-source*) e instalar/configurar/administrar/mantener el servicio, se contacta con un proveedor SaaS (por ejemplo, Gmail), se firma una SLA con las condiciones y calidad del servicio requeridas y solo se paga por su uso sin preocuparse de todo lo que ello implica. Ello implica no incurrir en costes adicionales (hardware, configuraciones, escalabilidad, actualizaciones, adecuaciones, por ejemplo, adaptación para diferentes dispositivos/navegadores, etc.). Además, como se mencionó anteriormente, dependiendo del proveedor, se podrán integrar la autentificación con los servicios corporativos para integrarlos a los servicios

SSO (*single-sign-on*) de la empresa o adecuación de la interfaz a los aspectos de imagen corporativa para integrarlos dentro del conjunto de aplicaciones/estilo de las mismas.

Además de las ventajas de «accedo & uso» desde cualquier dispositivo/lugar, el cliente no deberá preocuparse por aspectos críticos como, por ejemplo, la seguridad y el mantenimiento del servicio (*spam*, DDoS, virus, intrusiones, etc.), ya que todo ello estará en manos de los expertos del proveedor del servicio; en todo caso, si no cumple con la calidad del mismo, será objeto de una reclamación por incumplimiento de la SLA (siempre que haya sido tenido en cuenta).

Uno de los principales problemas del SaaS es que (generalmente) no se sabe dónde estarán los datos, lo cual para algunas empresas puede ser un punto negativo ya que muchas, en función del servicio y los datos, deberán cumplir con la legislación local, por ejemplo, la Ley de protección de datos de carácter personal (LOPD en España, pero en la UE todos los países tienen requerimientos similares derivados de Reglamento 2016/679 del Parlamento Europeo) que impone unas ciertas restricciones sobre dónde y cómo se almacenan los datos. Muchos proveedores incluyen en su carta de servicios condiciones adicionales –que luego reflejan en el precio– condiciones para cumplir con estos requerimientos.

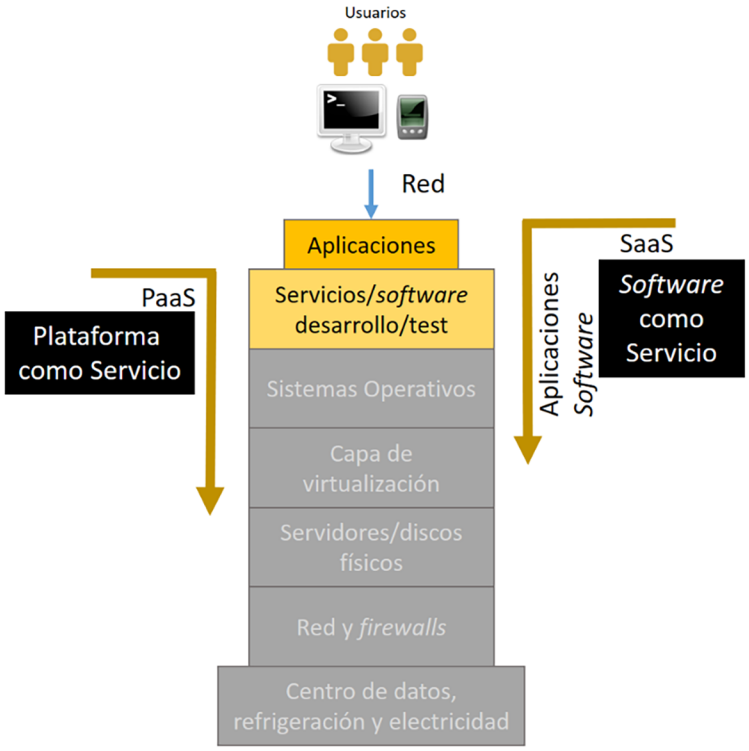
El otro problema que hay que considerar es que, dado que el servicio es remoto, la calidad de la conectividad y la alta disponibilidad del proveedor son dos de los parámetros que necesita explicitarse claramente en la SLA para estar cubierto en cuanto a la calidad del servicio, para que este no pueda afectar a la actividad de la empresa, y si lo hace, poder cubrir los costos de daños y perjuicio generados.

> Como ejemplo de SaaS, se podría considerar la plataforma G Suite de Google que, para el caso de correo empresarial, ofrece personalización en la dirección (@suempresa.com) con posibilidad de alias, 30 GB de almacenamiento en correos recibidos, compatibilidad con clientes como Microsoft Outlook, Apple Mail y Mozilla Thunderbird, entre otros, sin publicidad, soporte 24/7, ilimitadas listas de distribución (ventas@suempresa.com), personalización, correo para dispositivos móviles (*app*), incluido trabajo sin conexión, chat/videoconferencia integrado (*hangouts*), servicio de migración incluido, 99,9 % de tiempo de actividad garantizado, 0 % de tiempo de inactividad planificado y copias de seguridad automáticas. El potencial cliente deberá revisar todas las características y funcionalidades para ver si se adecuan a las necesidades de la empresa y escoger cuál de los planes satisface sus necesidades (costo frente a servicio) que van, para G Suite (que incluye no solo Gmail, sino una serie de productos en SaaS), desde 5 $ (*profesional basic*) a 10 $ (*bussiness*).

Una categoría adicional en SaaS es el OpenSaaS (acuñado por 2011 por Dries Buytaert de Acquia), que se refiere al modelo antes mencionado, pero basado en código *open-source*, que mantiene los aspectos específicos de este y tiene en cuenta además los criterios/actualizaciones/mejoras de un proveedor específico; hay autores que defienden su importancia dentro de ámbitos específicos, como por ejemplo el artículo *OpenSaaS and the future of government IT innovation*, de A. Hoppin, en el que defiende a WordPress como un ejemplo de éxito de OpenSaaS donde los clientes pueden construir sus sitios web instalando,

personalizando y manteniendo WordPress o acceder a WordPress.com y adoptar alguno de los planes de servicio (incluido uno gratuito) y solo preocuparse del contenido.

La figura siguiente muestra las capas de acceso y gestión tanto de PaaS como de SaaS.



Tanto en PaaS como es SaaS, el modelo de negocio es radicalmente diferente al modelo de licencia (generalmente basado en instalaciones/clientes y de tiempo ilimitado) adoptado por la mayoría de proveedores de software/servicios. En este caso se basan, en su gran mayoría, en una suscripción mensual/anual o en un plan de servicio que incorporan un conjunto de recursos/accesos/usuarios y dependiendo del tipo de plataforma por criterios de uso adicionales, por ejemplo GBytes almacenados, transacciones, eventos, instancias, BD, o bien algunos parámetros físicos (memoria, *cores*, tipo de disco).

Dado el costo relativamente bajo que supone, para un proveedor, un nuevo usuario (en un entorno *multenant*), muchos proveedores utilizan como reclamo el acceso gratuito a la plataforma (PaaS o SaaS) bajo modelos como *freemium* (palabra unión de *free* y *premium*) o basados en publicidad donde se ofrecen servicios básicos gratuitos y con costo por otros más avanzados o especiales.

Por ejemplo, Google App Engine ofrece un modelo *free* con límite diario en las instancias, transacciones NoSQL/red, almacenamiento, acceso a la API y un suplemento cuando se superan estos límites.

Tanto en PaaS como en SaaS los proveedores utilizan diferentes formas de interactuar con los servicios y recursos a través de protocolos de integración (basados, generalmente, en HTTP, REST y SOAP) e interfaces de programación (API) para permitir, o bien, en el caso del SaaS, acceder a los recursos internos del proveedor (BD, servicios de autentificación, sincronización, etc.), o en el caso del PaaS, para vincular las aplicaciones a recursos externos e importar/exportar datos, o también acceder a recursos propios desde las aplicaciones desplegadas.

Es importante también tener en cuenta que existen detractores del PaaS/SaaS; las causas más importantes son las cuestiones relacionadas con la seguridad (sobre todo en arquitecturas *multitenant*), la pérdida de datos o la corrupción de los mismos, que han hecho que los proveedores adopten diferentes soluciones para mitigar estos efectos (por ejemplo, encriptación, redundancia, *data escrow* –almacenado de una copia en un tercer proveedor–, etc.).

También es importante destacar, entre otras inconveniencias, que estos modelos incrementan la latencia de las aplicaciones; no son adecuados para aplicaciones que necesiten respuestas en el orden de los milisegundos; como todos los tipos de *cloud*, siempre existe la posibilidad de *data-lockin* o *vendor-lockin*; la integración con aplicaciones empresariales o datos puede significar costos elevados o riesgos adicionales, o que es difícil verificar (por más que se disponga de una SLA detallada) las garantías sobre la capacidad de los expertos, infraestructura, mantenimiento, tiempo de disponibilidad o profesionalidad del personal de IT que da soporte a los desarrollos del cliente.

1. PaaS: casos de uso

1.1. Heroku

Heroku (2007) es una PaaS que utiliza una estrategia ágil para construir y desplegar aplicaciones web y donde los desarrolladores de aplicaciones dedican su tiempo a su código de aplicación y no a gestionar servidores, despliegue, operaciones en curso o escalado de la infraestructura subyacente. Esta plataforma en la actualidad es propiedad de Salesforce (2011) y forma parte de Salesforce App Cloud.

El objetivo de Heroku es permitir que una aplicación pueda ser desarrollada y desplegada rápidamente para que los clientes puedan interactuar con ella; se favorece así la ventaja competitiva del usuario (desarrollador) y se aprovecha toda su experiencia en torno a las aplicaciones sin distraerse con otros requerimientos como servidores, hardware, librerías, servicios, etc. Es decir, facilita los procesos de despliegue, configuración, escalado, ajuste y administración de aplicaciones de forma que sean lo más simples y sencillos posibles, para que los desarrolladores puedan concentrarse en lo que es importante: crear aplicaciones que representan aportaciones de valor en relación con sus competidores en un tiempo mínimo.

Heroku, en sus inicios, fue desarrollado para Ruby, pero en la actualidad soporta además Node.js, Java, PHP, Python, Go, Scala, Clojure, que permite el desarrollo de *apps* y la ejecución dentro de *dynos* (contenedor inteligente que se ejecuta sobre un entorno de alta fiabilidad). No obstante Heroku dispone de otros productos (Postgresql, Redis, Apache Kafka, Connect, entre otros) y herramientas específicas para los desarrolladores como Add-ons, Buttons (para el aprovisionamiento, despliegue y configuración de herramientas de terceros), o Buildpacks como entornos de desarrollo en otros lenguajes y *frameworks*.

Heroku define como *stack* a una imagen del sistema operativo mantenida por Heroku basado en una distribución de Ubuntu LTS. Las aplicaciones de Heroku, a través de los *buildpacks*, se transformarán en código ejecutable construido específicamente para este *stack*; en la actualidad el *stack* predeterminado (*cedar-14*) está basado sobre Ubuntu 14.04.

Una pieza fundamental dentro de la arquitectura Heroku son los *dynos*, que, como se mencionó anteriormente, es un contenedor Linux sobre el que se puede ejecutar un comando específico disponible en el entorno por defecto (*cedar stack*) o en la aplicación. Heroku ejecuta *dynos* en tres configuraciones diferentes:

- *Web dynos* (para procesos que le estará permitido recibir tráfico http desde los *routers*).

- *Worker dynos* (puede ser cualquier tipo utilizado para trabajo en *background*, colas, etc.; se pueden ejecutar múltiples de ellos en una aplicación, por ejemplo, uno para las tareas urgentes y otro para las de largo término).

- *One-off dynos* (son *dynos* temporales que puede ser asociados o no al terminal y pueden utilizarse para tareas administrativas, migración o sesiones de consola.

Además, los *dynos* pueden ser de diferentes tipos (basados en sus prestaciones) y se denominan *free*, *hobby*, *standard* y *performance* dentro del entorno *common runtime* (para todos los usuarios de Heroku) y *private* para el entorno Heroku Enterprise.

Los *dynos* son gestionados por el *dyno manager* (distribuido por regiones y *tenants*) que los administra y controla su ciclo de vida para que el usuario no deba preocuparse de ello. Si la aplicación lo requiere, los *dynos* pueden escalar horizontalmente para manejar, por ejemplo, más peticiones HTTP y así soportar más volumen de tráfico o verticalmente para disponer de más recursos (por ejemplo, más RAM). Todos los *dynos* trabajan de forma aislada bajo criterios estrictos por motivos de seguridad que garantizan que, a todos los niveles (incluso más allá de los que provee el propio contenedor), no exista ningún tipo de intercambio de información entre ellos a pesar de que algunos (*free, hobby* y *standard*) compartan los recursos subyacentes.

Cada *dyno* recibe un sistema de archivos temporal (*ephemeral filesystem*), con una copia actualizada del código desarrollado que podrá utilizar durante el ciclo de vida, y una interfaz de red propia donde su configuración dependerá del tipo de *runtime* donde se ejecute. El *common runtime* provee un aislamiento de grado máximo a través de un *firewall* y solo el tráfico web entrante será habilitado si el *dyno* es web (*dynos worker* y *one-off* no podrá recibir peticiones).

Los procesos individuales dentro de un *dyno* podrán comunicarse utilizando TCP en una red 172.16.0.0/12 y no podrán compartir IP o subredes con otros *dynos*, pero sí podrán comunicarse externamente con cualquier servicio en internet [Hre].

Heroku cuenta con clientes destacados y ofrece una cuenta gratuita que permite crear *apps*, conectar con DB y servicios *add-on*, colaborar en las *apps* sobre la base de una plataforma propia para las *apps*, con gestión de estas y con la posibilidad de despliegue en pocos minutos. La cuenta gratuita dispone de un conjunto de horas mensuales de *dyno* (550 para cuentas no verificadas y 450 adicionales para las verificadas, con tarjeta de crédito), un *dyno* de cada tipo (*web, worker, one-off*) máximo por aplicación, 512 MB RAM por *dyno*, apagado del dyno después de 30' de inactividad (para no gastar los recursos) y despertado automático cuando se reciba una petición web, acceso a Heroku Dashboard y Heroku CLI para la gestión de aplicaciones, dominios configurables para cada aplicación (cuentas verificadas), y hasta un máximo de cinco *apps* gratuitas en cuentas no verificadas o cien en verificadas.

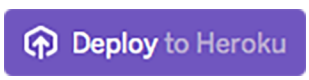
1.1.1. Despliegue de una aplicación con Heroku

Crear una aplicación y desplegar una aplicación es muy sencillo; en este caso se utilizará una *app* escrita en Ruby para demostrar cuáles son los pasos que se realizarán:

- buscar el código de la aplicación (página web en este caso),
- desplegar la aplicación,
- configurar los recursos externos (en este caso una base de datos Postgres),
- proveer el dominio y
- hacerla disponible.

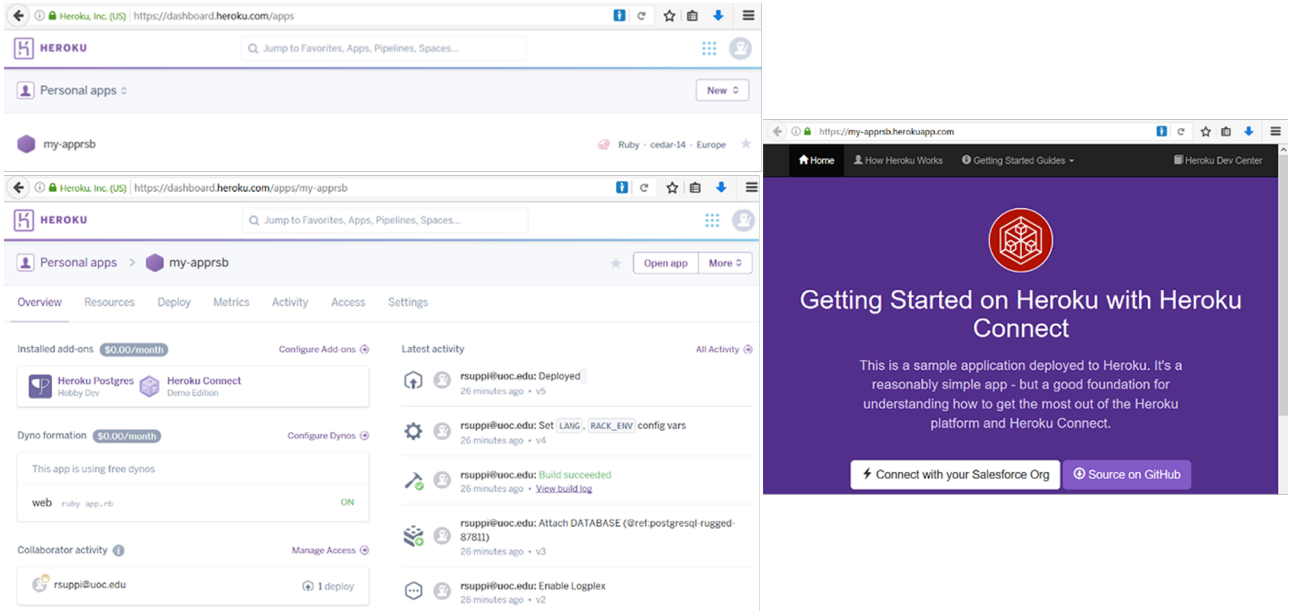
En este caso solo se desplegará la aplicación sin un entorno de desarrollo local, todo a través de Heroku Dashboard, y se realizarán modificaciones (también se conecta la BD de la aplicación a Salesforce Developer Edition, se sincroniza y modifica la aplicación para obtener datos de la BD utilizando Heroku Connect).

Para ello, se utilizará una aplicación de ejemplo que está en Github, por lo cual o bien desde este, o desde el tutorial, o desde el siguiente botón/enlace (se debe estar conectado a Heroku) haciendo un clic se abrirá la configuración de la *app* sobre Heroku sobre el *dashboard*:



Haciendo clic en *Deploy* se realizarán todos los pasos hasta el despliegue en Heroku, con lo cual ya se podrá ver la aplicación (*view*) o gestionarla (*manage app*). La URL será el *https://nombre_app.herokuapp.com* (en nuestro caso, la aplicación es *my-apprsb* y la URL será *https://my-apprsb.herokuapp.com/*).

Las imágenes a continuación muestran el *dashboard* con la aplicación, los detalles de esta y su resultado accediendo a un navegador. Como se ha mostrado, el procedimiento es muy simple y ya se dispone de una aplicación desplegada y que se ejecuta en un servicio sin tener que pensar en hardware, SO, entornos, *routers* o dominios.



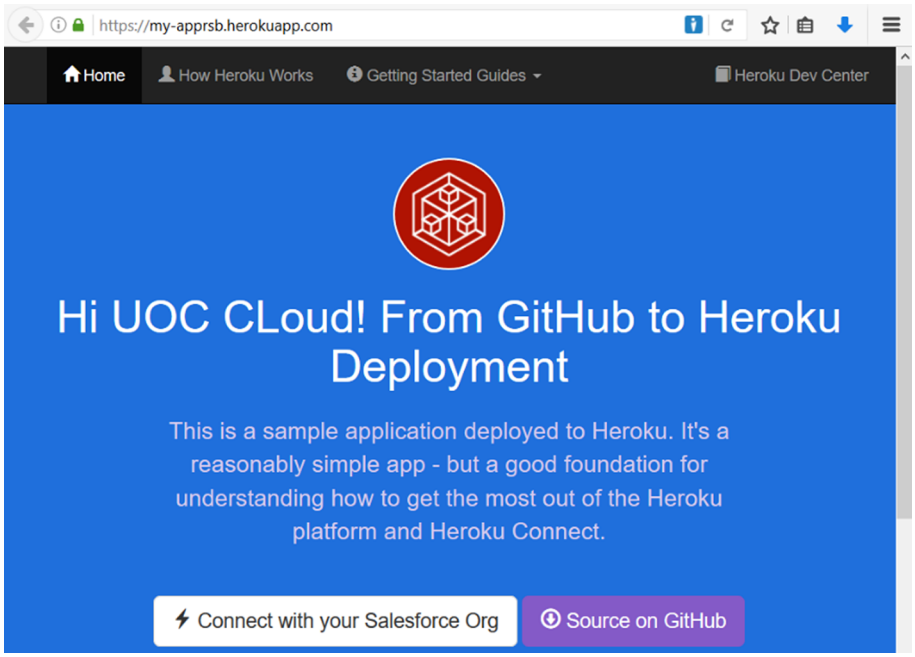
La modificación de la aplicación será sumamente fácil a través de GitHub; para ello se integrará la cuenta de Heroku con GitHub.

En primer lugar, será necesario disponer de una cuenta (gratuita) en GitHub en la cual se copiará el repositorio original (*https://github.com/heroku/no-local-dev-getting-started*) haciendo en este un clic en *Fork* (arriba a la derecha), lo cual creará una copia del repositorio en nuestra cuenta.

Es conveniente acceder a *settings* y cambiar el nombre del repositorio (*my-heroku-deployment* en nuestro caso) y sobre *Heroku* → *app a desplegar* → *Deploy* → *Connect to GitHub* y autorizar a la interconexión.

Luego con *Search* (buscar y seleccionar el repositorio que se desea vincular *my-heroku-deployment* en este caso) → *Connect* y en la sección *Automatic deploys* → *Enable Automatic Deploys*, con lo cual ya se tiene el código vinculado a Heroku y todas las modificaciones que se realicen sobre GitHub serán desplegadas automáticamente sobre la *app* y visibles externamente.

Para modificar la aplicación sobre GitHub → *my-heroku-deployment* → *Views directory* → *home.erb* → hay que editar con el ícono del lápiz a la derecha y modificar el texto → *Commit changes* (al final de la página). Volver a directorio raíz → *Public Directory* → *main.css* y modificar el parámetro *background* con otro color → *Commit changes*. Sobre Heroku → *App* → *Activity* se podrán ver los cambios actualizados y las diferencias entre las versiones en Github. La figura muestra la aplicación con cambios en el texto y en el color del fondo sobre la anterior.



Como se puede observar, la gran potencialidad de la plataforma y la simplificación total del desarrollo/despliegue y modificación de una aplicación permite, además, trabajar sobre la base de un ciclo continuo y colaborativo subiendo automáticamente los nuevos desarrollos a Heroku automáticamente.

1.1.2. Heroku with PHP y CLI sobre un entorno local

Para esta prueba, como máquina de desarrollo, se utilizará una MV Ubuntu 15.10; se ejecutará sobre KVM y sobre ella se instalará `php`, `composer` y `git` además de `heroku` CLI. Para ello, como *root* ejecutar (o con `sudo`):

```
apt-get install php5 php-mysql
apt-get install composer git
apt-get install software-properties-common
add-apt-repository "deb https://cli-assets.heroku.com/branches/stable/apt ./"
curl -L https://cli-assets.heroku.com/apt/release.key | sudo apt-key add -
apt-get update
apt-get install heroku
```

Para verificar que todo es correcto, ejecutar desde un terminal cada uno de los comandos (por ejemplo, preguntando por la versión) y comprobar que todo funciona adecuadamente: `php -v; composer -V; git –versión; heroku -v`.

A continuación, se debe autentificar sobre Heroku ejecutando `heroku login` e introduciendo la dirección de correo con el cual nos hemos registrado y el *passwd*. Para esta prueba se clonará un repositorio de una aplicación de prueba de Heroku y PHP terminará de configurar Git con el nombre del usuario y la dirección de correo:

```
git clone https://github.com/heroku/php-getting-started.git
cd php-getting-started
git config --global user.mail "dirección_de correo"
git config --global user.name "nombre_usuario"
```

En este directorio hay tres archivos importantes además del directorio de la aplicación (web), que son: *Procfile* (mecanismo para definir el *dynos* y los comandos a ejecutar sobre Heroku); *composer.json*, para manejar las dependencias en PHP (Heroku utiliza el comando `composer` para ello); y *app.json* (esquema para describir una aplicación web).

El paso siguiente es crear la aplicación en Heroku (el nombre es asignado en forma aleatoria, pero se puede pasar como parámetro al comando) que asociará un alias (*heroku*) a Git para gestionarla y ya se podrá «subir» el código de la aplicación que, como resultado, nos dará la URL a la cual se podrá acceder:

```
heroku create
git push heroku master
```

Se puede comprobar si se tiene, como mínimo, una instancia de la *app* ejecutándose con:

```
heroku ps:scale web=1
```

Y además de la URL, también se puede abrir la aplicación desde la línea de comandos (si se dispone de interfaz gráfica en la MV), ver los *logs* sobre la plataforma, o ver los *dynos* asociados a la aplicación:

```
heroku open
heroku logs --tail
heroku ps
```

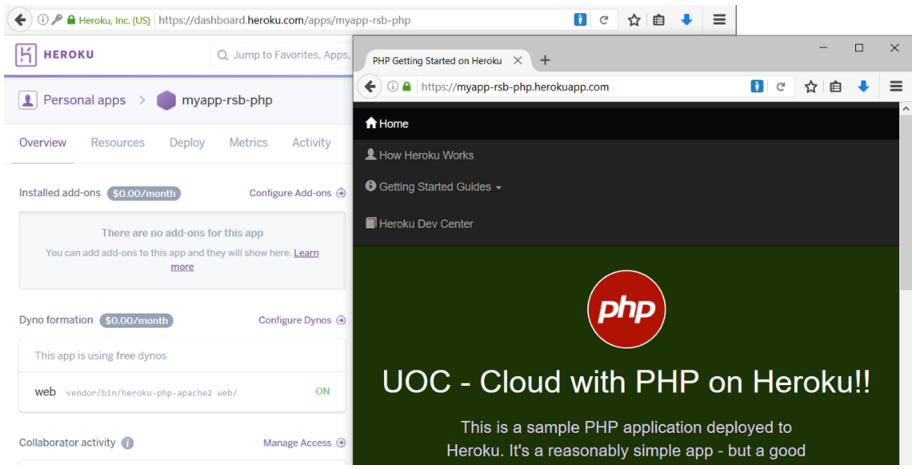
Si se desea realizar modificaciones sobre el repositorio local, primero se debe actualizar las dependencias localmente (las cuales se basan en el archivo *composer.json*):

```
composer update
```

A continuación, se puede modificar la página web y el estilo (por ejemplo, en el directorio *web/stylesheets* y *web/views*), añadir los cambios a Git, propagarlos y subirlos a Heroku y recargar la URL para verificar que están de acuerdo con lo que se deseaba:

```
git add .
git commit -m "Cambios menores"
git push heroku master
```

Como resultado podemos ver la aplicación en la imagen siguiente con los cambios actualizados (texto y color del fondo), donde también se muestran los detalles de la aplicación sobre Heroku:



De esta forma, y desde el dominio local, fácilmente se puede disponer de un entorno donde en pocos minutos realizar cambios y ponerlos en producción con un mínimo esfuerzo.

La plataforma PaaS Heroku dispone de una gran cantidad de *add-ons*,*buildpacks* y recursos que permiten construir cualquier tipo de aplicación y poner en producción en pocos segundos. Conocer toda esta funcionalidad requiere tiempo y dedicación, pero, como se ha mostrado, los pasos son sencillos y existe una extensa documentación y guías para obtener información sobre los diferentes aspectos de la plataforma [Hre].

Ved también

Consultad la documentación sobre tareas adicionales como agregar nuevas dependencias, aprovisionar nuevos *add-ons*, iniciar un *shell* interactivo, o aprovisionar una DB para la aplicación.

1.2. OpenShift (Origin)

OpenShift es una plataforma PaaS de RedHat basada en contenedores para el desarrollo y gestión de software donde los desarrolladores pueden usar Git para desplegar sus aplicaciones web en los diferentes lenguajes de la plataforma (aunque también acepta código binario y que se ejecute en RHEL, por lo cual no solo se limita a lenguajes y entornos que acepta la plataforma, sino a cualquier otro que sea soportado por RHEL (Red Hat Enterprise Linux)).

Esta plataforma incluye todos los servicios subyacentes habituales en una plataforma PaaS, así como los mecanismos de escalabilidad y alta disponibilidad necesarios para disponer de una *app* en producción. OpenShift utiliza Docker como tecnología para los contenedores y Kubernetes para la gestión de grupos de estos [Osd].

Existen diferentes plataformas de Openshift, las cuales se sustentan en una distribución *open-source* OpenShift Origin (disponible en GitHub bajo licencia Apache) y utilizado por OpenShift Online, OpenShift Dedicated y OpenShift Container Platform. Origin, además de Docker y Kubernetes, incluye un conjunto de herramientas DevOps para la gestión de todo el ciclo de vida de una aplicación; proporciona una plataforma basada en contenedores de aplicaciones de código abierto que es posible desplegar en un clúster local o en un IaaS para uso corporativo/privado.

- OpenShift Online es el servicio de desarrollo y alojamiento de aplicaciones de *cloud* público de Red Hat; soporta una gran variedad de lenguajes, *frameworks* y BD, pero los desarrolladores pueden ampliarlos a través de OpenShift Cartridge.

- OpenShift Dedicated es la oferta de clúster privado administrado de Red Hat y sobre RHEL.

- OpenShift Container Platform (anteriormente conocida como OpenShift Enterprise) es la plataforma privada local de Red Hat como producto con soporte sobre RHEL.

Entre los lenguajes, bases de datos y *frameworks* que soporta OpenShift por defecto, se pueden enumerar: Java, .Net, Node.js, Perl, PHP, Python, Ruby, Tomcat; MariaDB, MongoDB, MySQL, PostgreSQL, SQLite; CakePHP, Dancer, Django, Jenkins, Node.js, Ruby on Rails.

También proporciona una serie de servicios basados en Red Hat JBoss *Middleware* que proporciona servicios nativos del *cloud* para el desarrollo aplicaciones de forma más rápida, inteligente y flexible (por ejemplo, A-MQ, BPM, *Data Grid*, *Data Virtualization*, EAP, *Fuse Integration Services*, *Single Sign-On*, *Web Server*).

Entre otras **características** de OpenShift Online, también se pueden destacar:

- gestión de versiones (a través de Git y bajo SSH para las conexiones seguras),
- despliegue con un clic (utilizando el botón Git Push para reducir al máximo las necesidades de administración y gestión),
- soporte de diferentes estados de la aplicación (Dev, QA, Pre-Prod, y Prod),
- regiones de despliegue para mejorar las prestaciones y reducir la latencia,
- *workflows*,
- *snapshots*,
- resguardo y clonado simples,
- soporte a la integración continua,
- escalado automático,
- alta disponibilidad,
- colaboración,
- herramientas Devops, y
- un largo conjunto de prestaciones que se pueden consultar en la documentación [Gso].

Sobre OpenShift Online existen dos versiones (V2 y V3 o también llamada NextGen) donde existen cambios importantes de una en relación con la otra (por ejemplo, en la unidad de ejecución o en la gestión) que se pueden consultar en la documentación correspondiente.

Un inconveniente que existe sobre OpenShift Online V2 es que, desde agosto de 2016, ya no aceptan nuevos usuarios, y sobre OpenShift Online (V3-NextGen) Developer Preview, la autentificación es por GitHub, pero el acceso es limitado (por treinta días) y las peticiones son puestas en una lista de espera hasta que los recursos estén disponibles (se puede realizar la petición, pero nos informarán por correo electrónico del momento en que se podrá acceder).

Dados los inconvenientes mencionados previamente se realizará una pequeña prueba sobre OpenShift V2, pero luego se centrarán sobre OpenShift Origin dadas las restricciones para acceder a OpenShift Online V3.

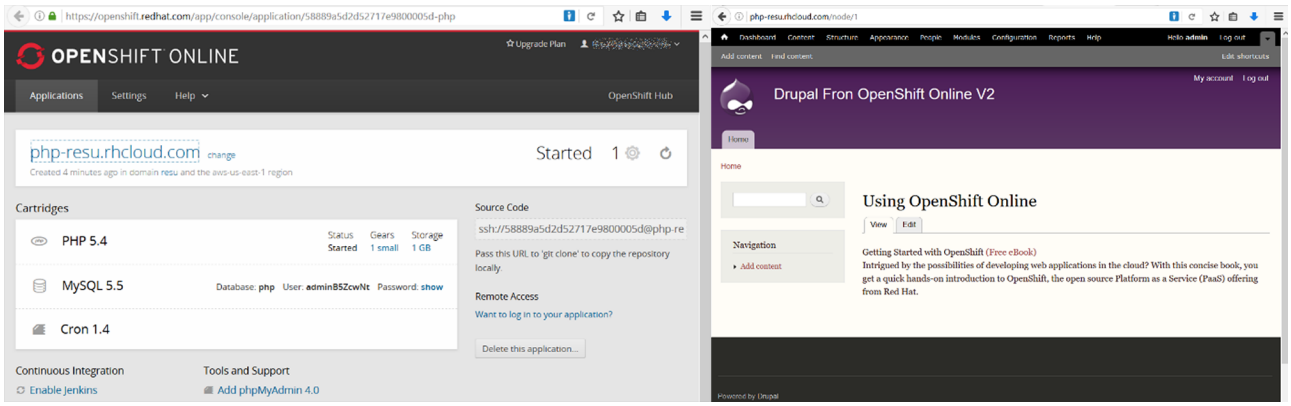
1.2.1. OpenShift Online (V2)

En este caso, y disponiendo de una cuenta de usuario obtenida antes de agosto de 2016, se puede acceder a la plataforma → *My Account* → *Open Shift WebConsole* y crear la primera aplicación.

OpenShift Online permite seleccionar *Cartridges*, que es un entorno de ejecución para una aplicación (por ejemplo, Jenkins Server, JBoss Application Server, Tomcat, Vert.x, WildFly, Ruby, PHP, Python, Node.js, Perl, entre otros) o también dispone de *QuickStart* como entornos preconfigurados (código y librerías), entre los cuales se pueden enumerar: Drupal, Ghost, WordPress, CapeDwarf, Ruby on Rails, Sinatra, CodeIgniter, HHVM, Laravel, Django, Ceylon, Go y MEAN, entre otros.

Después de seleccionar la aplicación, en nuestro caso como prueba, se creará un Drupal donde se podrá escoger la URL (*php-resu.rhcloud.com* en este caso, pero se puede cambiar posteriormente) y la región donde crear los *gears*. Un *gear* es un contenedor –servidor– con una serie de recursos que permite ejecutar las aplicaciones sobre el *cloud* de OpenShift y pueden ser de tres tamaños *small*, *medium*, y *large* cada uno con 1 GB de disco por defecto y 512, 1 o 2 GB de RAM respectivamente (en la cuenta gratuita el usuario dispone de 3 *gears*).

Cuando finalice el despliegue, se recibirá una pantalla de información (*passwd* y usuarios de la aplicación y la BD e instrucciones, instrucciones para copiar el código localmente y actualizarlo, etc.) y se podrá acceder a la consola y a la aplicación desplegada, como se muestra en las figuras siguientes.



Sobre esta se podría agregar phpMyadmin para gestionar la base de datos (que también se puede hacer desde la CLI) o habilitar Jenkinks para la integración continua (u otros *cartridges* que fueran necesarios).

Para trabajar con la aplicación localmente primero se deberá instalar el cliente sobre la máquina de desarrollo. Para ello ejecutar como *root* (o con `sudo`) la instalación y verificar que todo está correcto:

```
apt-get install ruby-full
ruby -e 'puts "     Hi PaaS User "'
      Hi PaaS User
sudo apt-get install rubygems      #dependiendo de la versión de Ubuntu no es necesario
apt-get install git-core
git --version
gem install rhc
```

Una vez instalado el cliente se debe configurar y solicitará el usuario/*passwd* a OpenShift, el *token* de autorización, la llave pública e información adicional:

```
rhc setup
```

Para crear una aplicación, por ejemplo, PHP 5.10, se deberá ejecutar:

```
rhc app create <app name> perl-5.10
```

Para clonar una aplicación sobre la máquina local (si se ha creado desde Webconsole, ya que si es desde la línea de comandos el clonado se hará automáticamente), hay que hacer cambios y subirlos a OpenShift (configurar previamente el *user.mail* y *user.name* de Git):

```
rhc apps      #mostrará info de la apps y la URL-git o también desde Webconsole
git clone <git_url> <directory to create>
git add .
git commit -m "A change to my application"
git push
```

Para acceder remotamente a los *gears* de una aplicación y realizar tareas sobre él, ver los *logs*, etc.:

```
rhc ssh <app_name>
```

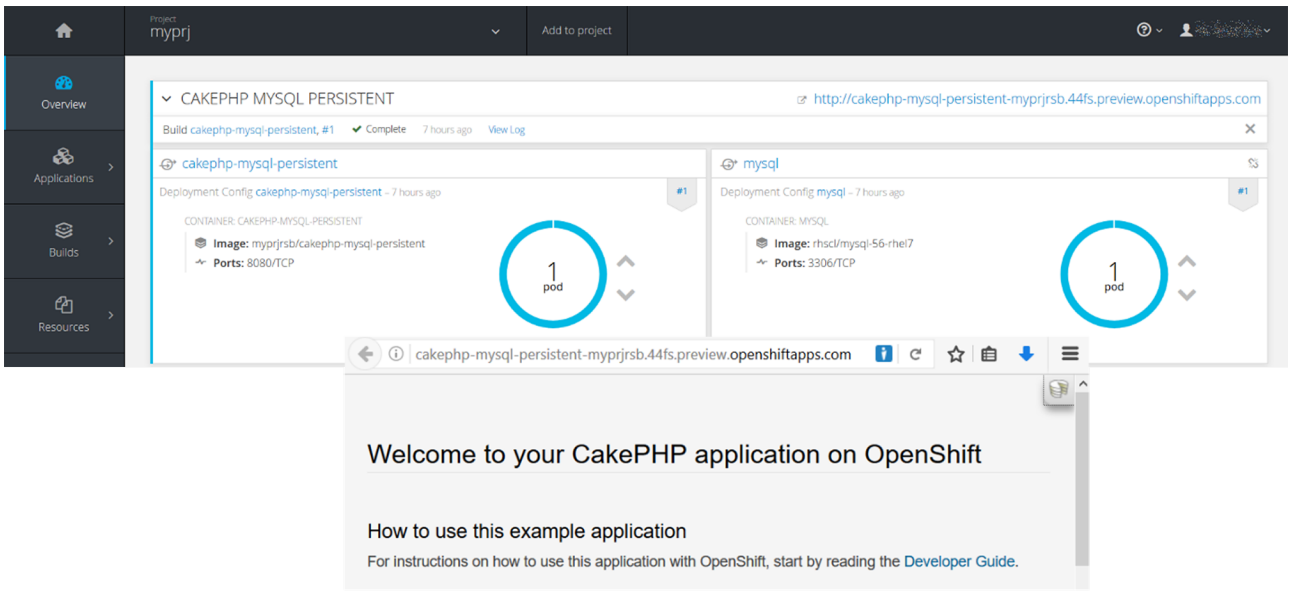
Una vez conectado se pueden utilizar los (algunos) comandos habituales de Linux y también la siguiente lista de particulares de un *gear*:

- `gear`: Controlar la aplicación (*start*, *stop*, *restart*, etc.)
- `tail_all`: Ver los archivos de *log*
- `export`: Listar variables de entorno
- `rm`: Borrar archivos/directorios
- `ls`: Listar archivos directorios

- `ps`: Listar procesos en ejecución
- `kill`: Parar procesos en ejecución
- `mysql`: MySQL shell
- `mongo`: MongoDB shell
- `psql`: PostgreSQL shell
- `quota`: Mostrar espacio de disco

En relación con la versión 3 (NextGen), luego de haber hecho la solicitud y recibir la correspondiente autorización para la cuenta gratuita (aprox. <1 semana), hay cambios significativos sobre los cuales ya se ha hablado previamente; se pueden consultar en la documentación del desarrollador. Más allá de la nueva estrategia/arquitectura basada en Docker-Kubernetes, modificaciones en la GUI o en los recursos asignados (de 3 *gears* a 4 *cores* + 2GiB RAM), de ya no disponer de *cartridges* sino de *images*, entre otros cambios, la forma de trabajar con la plataforma es similar a la V2 y donde se puede apreciar más agilidad y simplicidad a la hora de definir y desplegar un proyecto.

Como prueba, se ha desplegado un proyecto en PHP con CakePHP basado en el ejemplo del repositorio de OpenShift con un entorno persistente y una base de datos Mysql, como se puede apreciar en las figuras siguientes. En este ejemplo solo se ha realizado una prueba de concepto, pero tanto para la plataforma V2 como para esta (V3) es necesario analizar y experimentar con detalle para conocer sus prestaciones y funcionalidad.



1.2.2. OpenShift Origin

La arquitectura de OpenShift Origin v3 es un **sistema en capas** diseñado para extraer toda la potencialidad de contenedores Docker gestionados por Kubernetes en una infraestructura, de forma tal que sea simple y eficiente insertar aplicaciones por parte de un desarrollador, por ejemplo, que instale Ruby, inserte su código y cargue una BD MySQL.

En relación con su antecesor, OpenShift v2, aporta más flexibilidad a la configuración y cambia el concepto de aplicación a favor de uno más flexible como «servicios» que permite a dos contenedores web reutilizar una base de datos o exponerla directamente a la red.

Como se ha mencionado anteriormente, Docker proporciona la creación de contenedores ligeros basadas en Linux y Kubernetes, la administración del clúster y la orquestación de los contenedores en varios *hosts* y además la gestión de: código fuente, compilaciones e implementaciones para desarrolladores, despliegue de imágenes y su escalado, seguimiento de equipos y usuarios que intervienen en el desarrollo, monitorización, entre otros servicios *core*.

En la documentación se puede encontrar toda la información necesaria sobre su instalación y configuración en un clúster o en el *cloud* (tarea que exige dedicación y no es simple), pero en este apartado, como prueba de concepto, se utilizará una *all-in-one virtual machine* generada por los desarrolladores para probar y experimentar con Openshift Origin, la cual es una imagen con una instancia totalmente funcional de la plataforma que integra un registro Docker y generada a partir de los *scripts* disponibles en GitHub.

En primer lugar, se debe tener instalado Vagrant y Virtualbox 5.0 (si se trabaja con Ubuntu 15.10, la versión de Virtualbox es 5.1 y no es compatible con Vagrant, por lo cual se deberá bajar desde *virtualbox.org* la versión correspondiente e instalarla.

Para resolver las dos dependencias de Virtualbox 5.0 sobre Ubuntu 15.10, la primera (qt4) se resuelve con el propio repositorio ejecutando `apt-get install -f`, pero la segunda (libvpx2) se debe buscar el paquete e instalarlo (por ejemplo, desde LaunchPad). Luego, se deberá ejecutar sobre un directorio vacío ejecutar como root:

```
vagrant init openshift/origin-all-in-one
```

Esto creará un archivo (*Vagrantfile*) que se podrá modificar, por ejemplo, para cambiar el tamaño de la memoria RAM y adecuarlo a los recursos de la máquina (quitar los comentarios a la sección *config.vm.provider* y cambiar, por ejemplo, el tamaño de la RAM *vb.memory = "4048"*.

```
vagrant up --provider=virtualbox
```

Esto bajará la imagen y desplegará la máquina virtual; al final mostrará la URL para la conexión a WebConsola, que se podrá acceder con usuario/*passwd* **admin/password** respectivamente y se tendrá acceso a la consola de la plataforma PaaS OpenShift Origin V3. Para desinstalarlo, hay que cambiarse al directorio que contiene el Vagrantfile y ejecutar:

```
vagrant halt
vagrant destroy --force
vagrant box remove --force openshift/origin-all-in-one
```

Para interactuar desde la línea de comandos, es necesario bajarse el cliente `oc` desde GitHub (descomprimir el archivo y renombrar el directorio).

A partir de ello ya se podrá interactuar con la plataforma con los siguientes comandos:

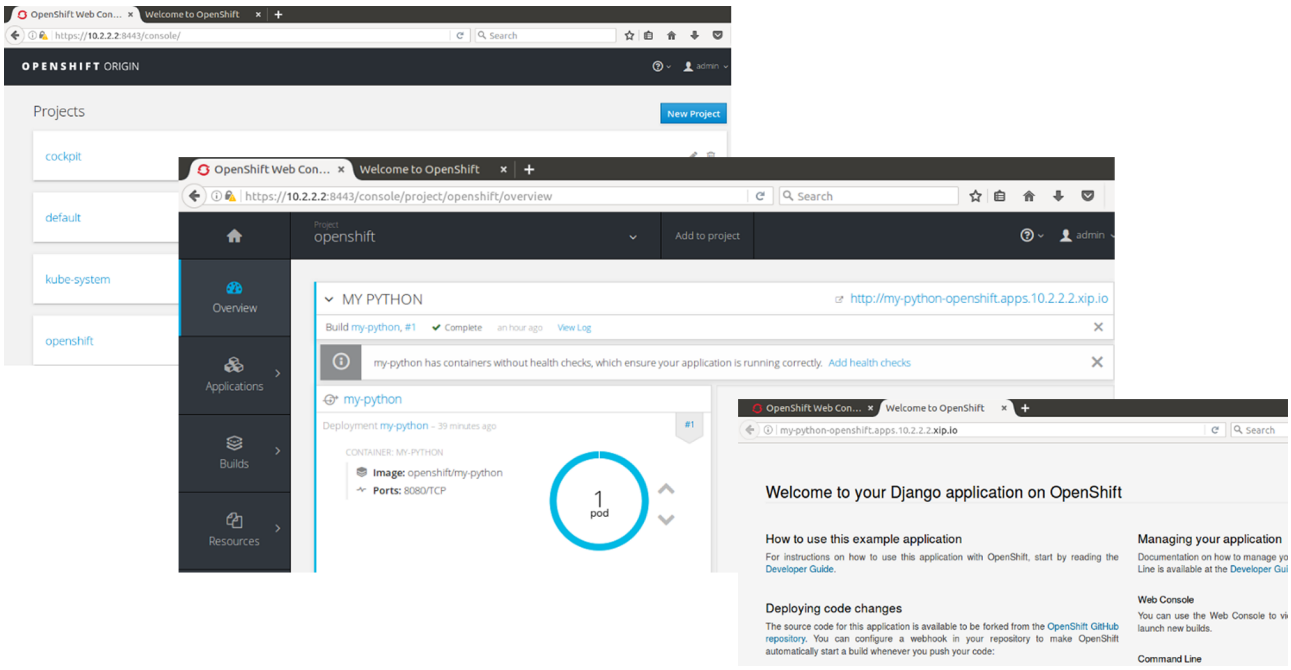
- `oc versión`
- `oc login`: Conexión a la plataforma, se debe indicar URL
- `oc projectopenshift`: Selección del proyecto *openshift*
- `oc status`: Estado del proyecto seleccionado
- `oc get routes`: Rutas disponibles
- `oc get pods`: Obtiene los *pods* en ejecución
- `oc get pods <nombre del pod> -o json`: Mostrará toda la información del *pod*.

Los *pods* es la mínima unidad desplegable sobre la plataforma y pueden estar formados por uno o más contenedores, aunque se garantiza que están en el mismo *host* (desde el punto de vista de la implementación, un *pod* no es más que un contenedor con uno o más contenedores dentro).

Cada *pod* tiene su propia IP, su espacio de *ports*; los contenedores del mismo *pod* pueden compartir espacio de almacenamiento y pueden ser marcados con *tags* para aplicarles operaciones de grupos. Por ejemplo, un *pod* puede tener un contenedor de un servidor Apache, otro de un analizador de *logs*, y otro de un servicio de archivos para gestionar los archivos que se cargan en el servidor.

Como primer ejemplo de aplicación, se ha creado desde WebConsole un nuevo proyecto, se ha seleccionado del catálogo un proyecto en Pyhton utilizando la python3.5 y como URL de Git para la carga de proyecto un ejemplo en

Django desde Github. Se ha desplegado el proyecto y el propio despliegue ha generado una URL y se ha visualizado. Las imágenes a continuación muestran este despliegue realizado en un tiempo muy breve.



Para complementar la experiencia se ha realizado el despliegue de una aplicación desde la CLI, en este caso la de un servidor Apache. Para ello se deberá ejecutar:

- `oc newproject myapache2`: Crea el nuevo proyecto
- `oc newapp fedora/apache`: Busca el contenedor Fedora/Apache y lo despliega

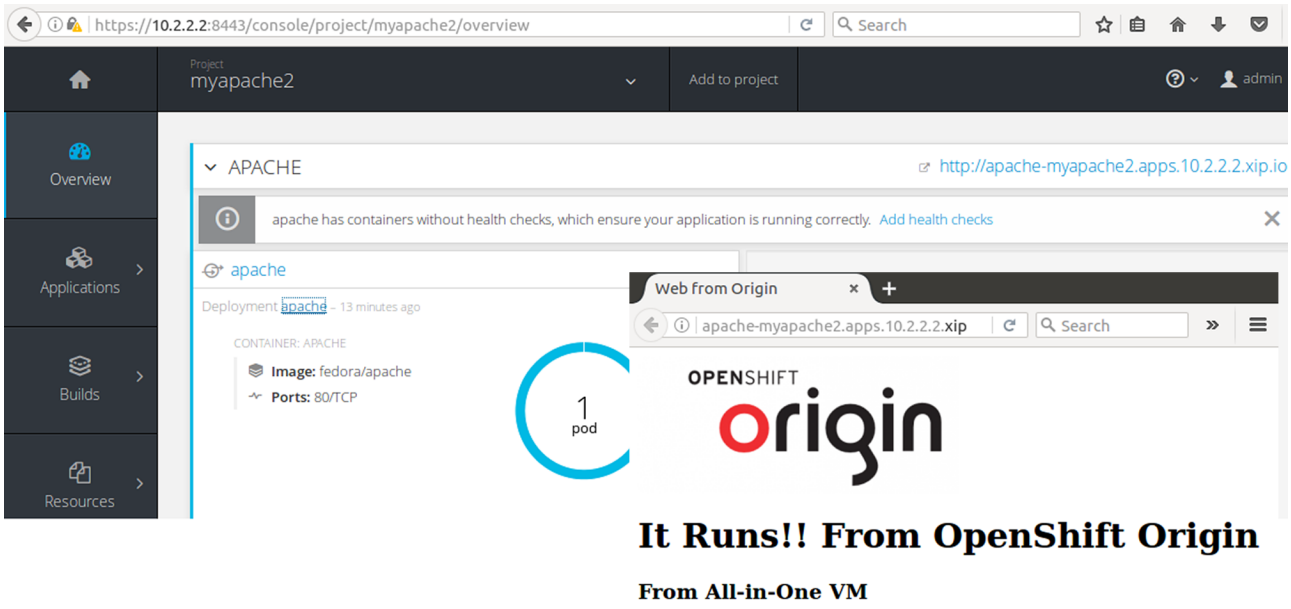
Esto debe descargar y desplegar la imagen, por lo cual tardará en función de la velocidad de la red y sobre WebConsole se podrá ver la evolución o también a través de la línea de comandos ejecutando:

```
oc get pods watch
```

Sobre la consola se podrá generar una ruta de acceso y conectarse a través del navegador al nuevo servicio desplegado. Si se desea hacer modificaciones sobre el servicio (por ejemplo, cambiar puntualmente la página web de inicio), se puede acceder interactivamente al contenedor primero averiguando el nombre del *pod* y luego con el parámetro *rsh* (*Ctrl* + *D* para salir):

```
oc get pods
oc rsh <nombre del pod>
```

Las imágenes siguientes muestran la pantalla de control del despliegue realizado y la página del servicio que se está ejecutando.



Algunos comandos útiles para gestionar la máquina con Vagrant son:

Formato `vagrant <comando><arg>`:

- `init`: Inicializa un nuevo entorno Vagrant (crea Vagrantfile)
- `up`: Despliega/inicia el entorno Vagrant (ejecutar después de un *halt*)
- `destroy`: Para y borra todos los archivos de una MV Vagrant
- `suspend`: Suspende la ejecución de la MV
- `halt`: Para la ejecución de la MV
- `provision`: Aprovisiona una MV
- `reload`: Reinicia una MV, carga nuevamente la configuración de Vagrantfile
- `resume`: Reinicia una MV suspendida
- `ssh`: Conexión a la MV vía SSH
- `status`: Estado de la MV
- `help`: Muestra la ayuda

Si se desea parar la MV, cabe utilizar `halt`, reiniciarla con `up --provision` y si se desea destruirla y comenzar de nuevo con `destroy` y volver a hacer el `up` (pero esta vez no la descargará).

Además, es necesario, sobre Origin, complementar el trabajo y experimentar con los diferentes aspectos de WebConsole y CLI o su instalación en un clúster [Oso].

Como en las anteriores recomendaciones, para esta plataforma solo se ha visto una pequeña parte de la funcionalidad disponible y de las prestaciones que puede dar, ya que es una plataforma PaaS extremadamente completa que se puede personalizar para cualquier tipo de desarrollo/aplicaciones web. Es importante dedicarle tiempo y experimentar con ella para conocer las diferentes opciones y propiedades que dispone; así, luego se tendrá el conocimiento suficiente para desplegarla en un clúster o en el *cloud*.

1.3. Apache Stratos (antes WS02)

En junio de 2013, la empresa WSO2 (2005) especializada en PaaS y *open-source* donó la plataforma **PaaS Stratos** a la fundación Apache con la intención de promover una comunidad de proveedores neutros dentro del marco del *open-source* que pudieran hacer frente al *cloud lock-in*. El proyecto se transformó en Apache Incubator y pasó a *Top-Level Project* (TLP) en mayo de 2014 con el soporte de SUSE, Cisco, Citrix, NASA (JPL), Sungard y Engine Yard. Los componentes principales (donados) fueron Stratos Controller, Elastic Load Balancer and Cloud Controller, Stratos Foundation Services, Pre-built Cartridges (MySQL, PHP and Tomcat). WSO2 continúa ofreciendo una plataforma PaaS privada adaptada a las necesidades del cliente basado en Apache Stratos.

Stratos es una plataforma extensible que permite ejecutar aplicaciones Apache Tomcat, PHP y MySQL; puede ampliarse a otros entornos para proporcionar a los desarrolladores un PaaS donde desarrollar, probar y ejecutar aplicaciones escalables. También permite a los proveedores de IT beneficiarse de la utilización eficiente de la infraestructura (altas tasas de utilización), administración automatizada de recursos y tener información instantánea de qué se está ejecutando y dónde con el fin de monitorizar la infraestructura y generar información para su posterior traslado de costes (facturación).

Esta infraestructura se basa en software *open-source* con el soporte de la comunidad y grandes desarrolladores, presenta una arquitectura abierta y extensible (mediante *cartridges*) con un diseño por capas y una API consolidada, comunicación unificada a través de un *bus* de mensajes o de eventos en tiempo real, con soporte para su fácil integración en un IaaS, escalado automático, capacidad para predecir cargas futuras, soporte para *multi-tenancy* con balanceo de

carga y aprovisionamiento automático, agrupaciones de recursos, seguridad multicapa y con políticas de bloqueo para garantizar la privacidad, medición y monitorización multinivel para obtener información de conjunto y a la vez detallada sobre lo que ocurre en la plataforma, y un diseño refinado para integrarse fácilmente con nubes privadas, públicas e híbridas y con balanceadores de carga de aprovisionamiento externos.

La versión actual (4.2.0) permite:

- soporte para aplicaciones con múltiples entornos (*runtimes*) con escalado automático,
- soporte para Docker, Kubernetes y CoreOS,
- agente gestor de *cartridges* (soporta PHP, MySQL, Tomcat, .NET, pero han sido probados Ruby, Node.js, WordPress, Drupal, Tomcat, HAProxy y NGINX) para la gestión y actualización de estos en *runtime*,
- Cloud Controller para conectarse (utilizando la API de Apache *jclouds*) con el IaaS, autoescalado multiobjetivo (basado en *Complex Event Processor* para las decisiones en tiempo real),
- soporte para AWS EC2 y OpenStack (o cualquiera que tenga soporte de Apache jclouds, Google Cloud, CloudStack etc.),
- despliegue en multi-*cloud* (cuando se satura las instancias de un IaaS puede abrir instancias en otro sin intervención del administrador),
- particionado a diferentes niveles (regiones, zonas de disponibilidad, *hosts*, subredes, etc.),
- comunicación levemente acoplada (utiliza *Advanced Message Queuing Protocol*: ActiveMQ),
- *multi-tenancy*,
- automatización de los *cartridges* con Puppet,
- soporte para balanceadores externos como los de HAProxy, NGINX,
- sincronización con repositorios externos (*git push*),
- consola web,
- REST API,
- CLI interactiva (CLI *Guide*),
- monitorización y medidas,
- almacenamiento persistente para los *cartridges* (*persistence volume mapping*),
- soporte para MQTT y AMQP.

Se puede observar la tecnología y arquitectura descrita en la documentación; es de la versión 4.1, extensible para la versión 4.2 [Asd].

1.3.1. Despliegue de Stratos localmente con Mock IaaS

Stratos puede ser desplegado en un IaaS (EC2, OpenStack o GCE) o Mock IaaS (basado en Apache Stratos Mock IaaS) que simula los requerimientos que necesita Stratos del IaaS, lo cual permite de forma fácil experimentar con Stratos sin tener que hacer grandes despliegues (con un alto costo) sobre un *cloud* público/privado).

Para la siguiente prueba se utilizará una MV KVM Ubuntu 16.04 Server, en la cual se han instalado los prerrequisitos indicados del repositorio Ubuntu: MySQL, Maven, *zip* pero no el JDK, ya que el actual (JDK 8) no es compatible con la versión 4.1.x de Stratos. Para ello, se ha ejecutado:

```
apt-get install maven zip mysql-server
```

Para instalar JDK 7 se ejecutará:

```
sudo add-apt-repository ppa:openjdk-r/ppa
sudo apt-get update
sudo apt-get install openjdk-7-jdk
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

Para instalar ActiveMQ ir a repositorio y descargar el archivo para Linux (*apache-activemq-5.14.3-bin.tar.gz*) –se recomienda utilizar esta versión, ya que con la integrada en el repositorio Ubuntu se han detectado algunos problemas– y ejecutar:

```
tar xzvf  apache-activemq-5.14.3-bin.tar.gz
```

Finalmente descargar y descomprimir Stratos de algunos de los repositorios, en este caso se ha utilizado http://apache.rediris.es/stratos/4.1.6/apache-stratos-4.1.6.zip:

```
unzip apache-stratos-4.1.6.zip
```

Luego dentro de directorio de *<ActiveMQ>/bin* se ejecutará:

```
./activemq start
```

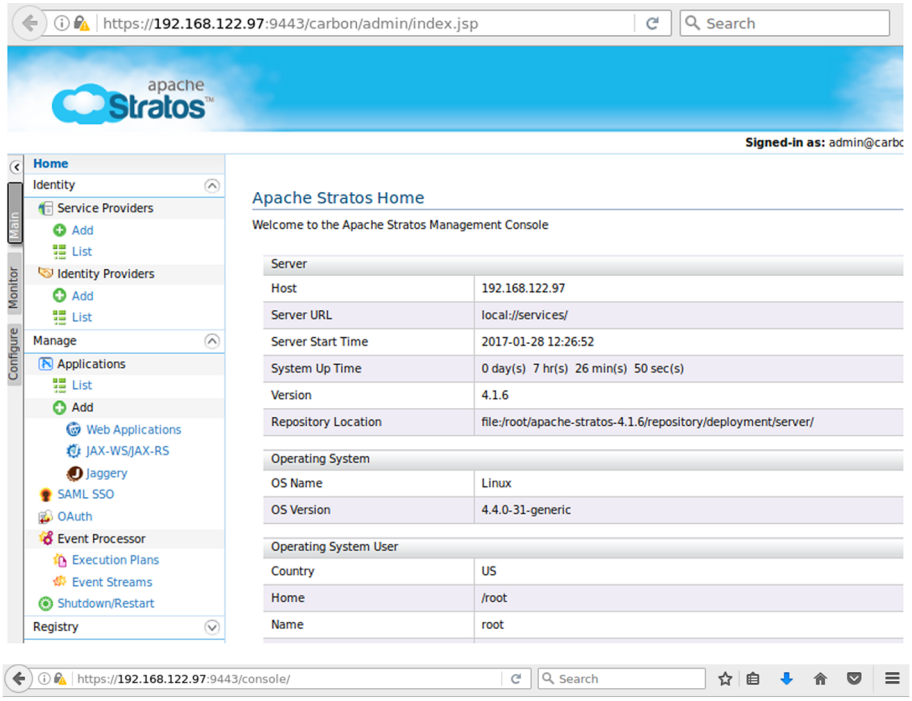
Si se desea parar el servidor de ActiveMQ se ejecutará el comando con *stop*. Y dentro de directorio de *<Stratos>/bin* se ejecutará (verificar que está bien configurada la variable *JAVA_HOME*):

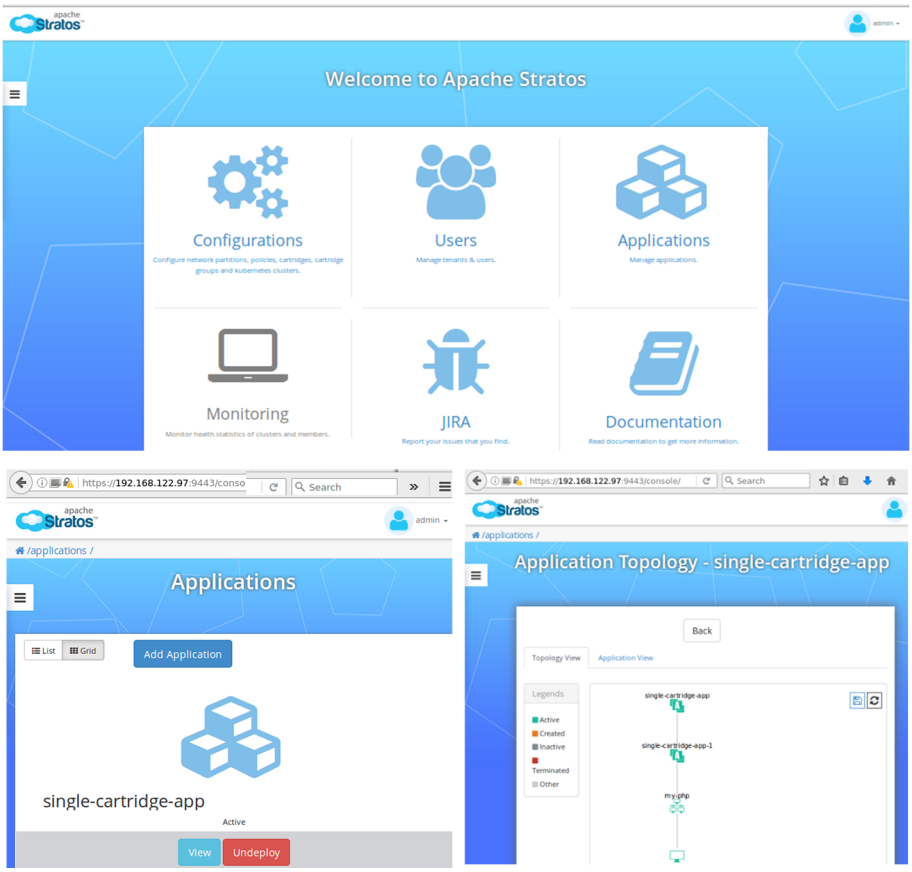
```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
./stratos.sh
```

Si se desea parar el servidor simplemente hacer *Crtl + C*. Finalmente, a través de un navegador se podrá acceder a la configuración del entorno (usuario/*passwd* **admin/admin** respectivamente) a través de la URL *https://ip_MV:9443* y a la consola de administración *https://ip_MV:9443/console*, como se muestra en la figura siguiente. En la documentación se describen los pasos para desplegar una aplicación: *network partition*, *deployment policy*, *auto-scaling policy*, *cartridge*, *cartridge group*, *application policy* y finalmente agregar *application*; no obstante, como prueba de concepto es aconsejable seguir los pasos indicados para instalar una aplicación utilizando la API y los *scripts* en el directorio *samples* del código fuente. Para ello, si solo se han descargado los binarios, instalar el código fuente:

```
git clone https://git-wip-us.apache.org/repos/asf/stratos.git mystratos
```

Luego, desde el directorio *mystratos/samples*, seguir las indicaciones de la documentación y se desplegará una aplicación con el *cartridge* por defecto de PHP; desde la interfaz gráfica se podrán ver los detalles que muestra la última figura.





También se puede experimentar haciendo el despliegue de una aplicación compleja, una anidada o escalar una aplicación. También se puede probar a desplegar una aplicación compleja, como WordPress, a través de *scripts*; para ello, cambiar al directorio:

```
cd mystratos/samples/applications/complex/wordpress-app/scripts/mock/
./deploy.sh
```

En la interfaz gráfica se tendrá toda la información sobre la aplicación desplegada y se podrá acceder a ella.

A diferencia de las otras plataformas analizadas, Stratos tiene una lógica, unas opciones y una metodología diferentes que permiten un amplio conjunto de prestaciones, pero que puede ser compleja de gestionar y configurar en un entorno distribuido. Ello requerirá que el administrador dedique tiempo y esfuerzo a analizar la documentación y experimentar con la plataforma Mock IaaS, en primer lugar, para luego pasar a desplegar esta sobre un IaaS.

1.4. Cloudify

Cloudify (2012) es un entorno *open-source* (disponible en GitHub) para la orquestación en el *cloud*; permite modelar las aplicaciones y servicios y automatizar su ciclo de vida incluyendo despliegue (en cualquier *cloud* o centro de datos, DC), monitorizar todos los aspectos de la aplicación desplegada, y detectar fallos/eventos y gestionarlos (manual o automáticamente).

Para describir una aplicación se utiliza un archivo (con sintaxis YAML) denominado *Blueprint*, que describe la aplicación (infraestructura, *middleware*, código, *scripts*, configuración, métricas y *logs*) a alto nivel y donde se puede definir todo el ciclo de vida. Cloudify desplegará las instancias, configurará la red, el almacenamiento y la seguridad para proveer los recursos requeridos para la aplicación y ejecutará los *scripts* (remotamente por SSH o localmente) o invocará herramientas de gestión para configurar los servidores y desplegar el *middleware* especificado y el código [Cdo].

Esta plataforma utiliza *workflows* simples para modificar o cambiar la estructura de la aplicación y, además, permite recolectar métricas y *log* a diferentes niveles para proveer información y agregación de datos y visualización para que sea más simple ejecutar los *workflows* y tomar decisiones a través de eventos (como escalado, reinicio, reejecución, etc.) sobre una interfaz gráfica (Cloudify Manager). Es una plataforma extensible a través de *plugins* (escritos en Python) que pueden ejecutar *scripts*, herramientas de configuración (CM *tools*), unificadores de *logs* y métricas, o cualquier otra herramienta necesaria.

La plataforma utiliza un lenguaje de especificación (DSL) llamado Cloudify's DSL (*Domain Specific Language*) basado en TOSCA (*Topology and Orchestration Specification for Cloud Applications*) promovido por Oasis.

La arquitectura a través de Python permite participar fácilmente en su desarrollo y utiliza diferentes paquetes *open-source* como Nginx, Elasticsearch, Logstash, RabbitMQ, Riemann, InfluxDB, Grafana, Flask, Gunicorn, Celery, Fabric, Diamond y Jinja2, entre otros.

Uno de los objetivos principales de Cloudify es reducir la complejidad del despliegue y mantenimiento de la gran cantidad de *stacks*, herramientas e infraestructuras que, combinadas, dan a los desarrolladores la potencia de generar aplicaciones web escalables en un tiempo reducido, de gran calidad y estabilidad que, si se ha de hacer manualmente, implica una gran cantidad de tiempo/recursos y conocimientos.

A través de los *blueprints* y utilizando un DSL (basado en TOSCA) independiente de la tecnología permite definir cualquier actividad de instalación, configuración y despliegue de cualquier paquete/entorno a través de modelando de la topología de la aplicación.

Cloudify permite integrarse con diferentes *clouds*/hipervisores (como OpenStack, VMware o con *plugins* para SoftLayer, Apache CloudStack, AWS), o con *plugins* para Docker con Chef, Puppet o incluso SSH con Fabric o Kubernetes.

La interacción con el usuario se realiza a través de una CLI o a través de un GUI llamada Cloudify Manager (CM). Este es un entorno dedicado que permite a los usuarios lo siguiente:

- gestionar los diferentes *plugins* (por ejemplo, Docker, Scripts, Chef y Puppet) para administrar los *hosts* de aplicaciones,
- mantener y gestionar los *blueprints*,
- crear diversos despliegues para cada *blueprint* e instalarlos,
- ejecutar diferentes *workflows* (escalado, reinicio...) para las aplicaciones instaladas (incluso simultáneamente para diferentes aplicaciones),
- monitorizar diversas métricas, registros, ver la topología de una aplicación y diferentes tareas de administración (por ejemplo, entorno seguro para administrar aplicaciones a través de autenticación y mecanismos de autorización, historial de métricas y eventos, administrar los agentes que se ejecutan en las máquinas *host* de una aplicación entre otras) y
- si bien se puede utilizar la CLI para desplegar recursos directamente, se recomienda utilizar el CM para administrar aplicaciones de nivel de producción.

1.4.1. Instalación local de pruebas con Vagrant y VirtualBox

En este apartado se utilizará una MV para interactuar con la CLI y el CM que cargará un *blueprint* que desplegará una aplicación web basada en *nodejs* la cual conectará con MongoDB para implementar una bodega virtual.

Para ello es necesario VirtualBox 4.3+ (en este caso se ha utilizado 5.0 sobre Ubuntu 16.04) y Vagrant 1.5+ (se ha utilizado 1.8.1), un mínimo de 2GB libres de RAM. Si se desea utilizar este dentro de una MV, es necesario que la MV esté configurada para soportar virtualización anidada.

En primer lugar, se debe descargar el Vagrantfile y ejecutar la MV (la primera vez se deberá descargar, por lo cual se tardará un poco más) que contendrá preinstalados la CLI y CM.

```
vagrant up
```

Una vez iniciada la MV se podrá conectar desde el *host* al CM a través de la URL *http://10.10.1.10/*

A continuación, se podrá conectar por SSH a la MV para ejecutar los comandos a través de la CLI y descargarse el *blueprint* que se utilizará de ejemplo:

```
vagrant ssh
```

El *blueprint* describe toda la orquestación de la aplicación incluidos los nodos que la componen, *workflows*, y relaciones/conexiones.

```
cd blueprints
git clone https://github.com/cloudify-cosmo/cloudify-nodecellar-example
cd cloudify-nodecellar-example/
git checkout tags/3.3
```

En el directorio *cloudify-nodecellar-example* existe un archivo *simple-blueprint.yaml* (junto con otros para desplegar la aplicación sobre otros recursos) que será utilizado para cargarlo (se debe tener en cuenta que este ejemplo utiliza como DNS 8.8.8.8, por lo cual se debe disponer acceso a él).

```
cfy blueprints upload -b nodecellar -p simple-blueprint.yaml
```

Donde `-b` especifica un nombre único para este *blueprint* y `-p` es el nombre del archivo y dentro del CM, en la pestaña *Blueprints*, ya se podrá visualizar y haciendo un clic se podrá observar su topología (ver figura siguiente): una VM, con *nodejs* y una DB MongoDB y una aplicación *nodejs* llamada *nodecellar*.

El *blueprint* define también unos parámetros de entrada que en este caso estarán en *blueprints/inputs/nodecellar-singlehost.yaml* y los valores para este ejemplo serán:

```
agent_private_key_path: /home/vagrant/.ssh/id_rsa
agent_user: vagrant
host_ip: 10.10.1.10
```

Para crear un despliegue:

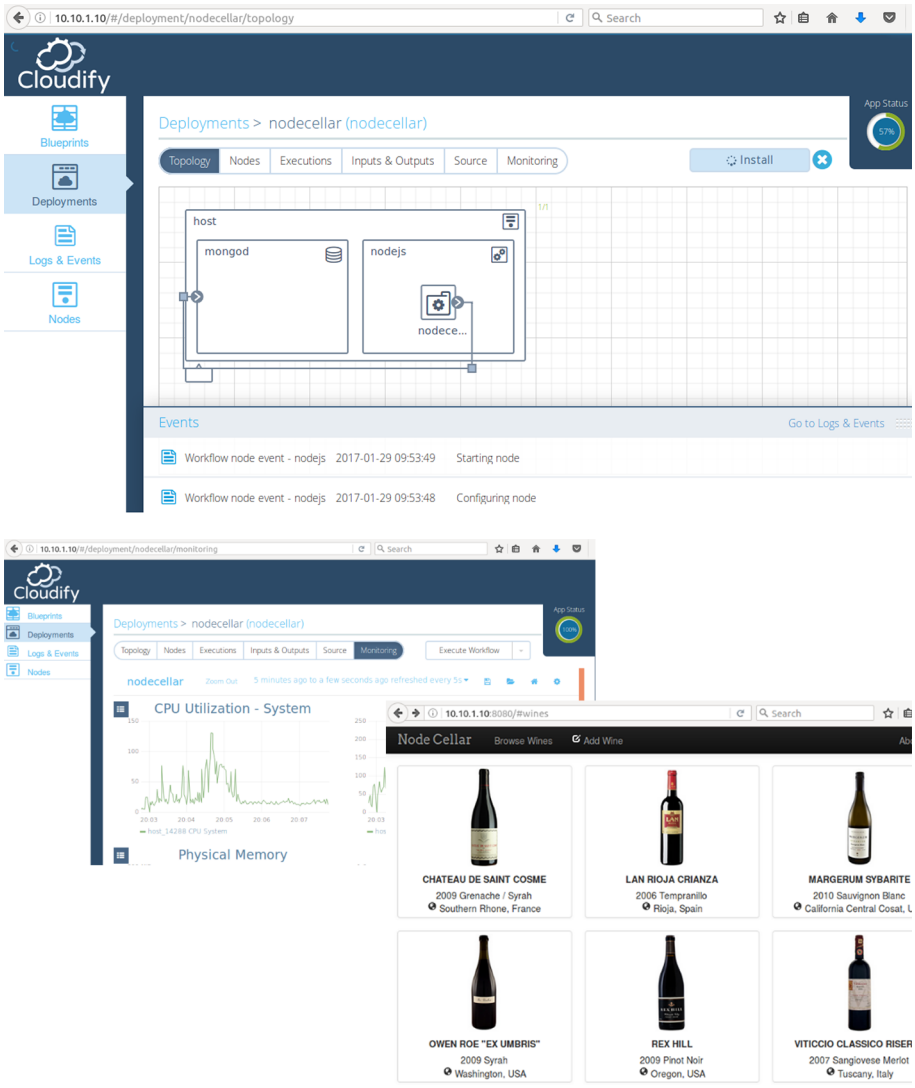
```
cfy deployments create -b nodecellar -d nodecellar --inputs \
```

```
    ../inputs/nodecellar-singlehost.yaml
```

El cual ya se podrá observar después de unos instantes en la pestaña *Deployments* del CM. Este despliegue no está materializado aún y es por ello que los nodos saldrán con 0/1 a costado de cada nodo. A continuación, se debe ejecutar el *workflow* para instalar el despliegue.

```
cfy executions start -w install -d nodecellar
```

En la parte superior derecha del CM → *Deployments* se verá un indicador de avance y los mensajes (también en la consola) de las tareas que va realizando Cloudify (para mayor detalle, se puede consultar la pestaña de *Log & Events* o dentro de *Deployments* → *Monitoring* para ver los recursos que se están utilizando). Esta acción tardará unos minutos (dependerá de la velocidad de la red) ya que debe bajarse e instalar/configurar los entornos. Cuando finalice la ejecución de la aplicación se podrá consultar a la URL *http://10.10.1.10:8080* desde el *host* donde se está ejecutando la MV. Las figuras siguientes muestran el entorno con el modelado de la aplicación, el despliegue y las métricas de uso y el resultado de la aplicación desplegada.



Para desinstalar el despliegue es necesario ejecutar otro *workflow*:

```
cfy executions start -w uninstall -d nodecellar
```

Para eliminar el despliegue del CM y apagar el CM:

```
cfy deployments delete -d nodecellar
cfy teardown -f
```

En un entorno real esto terminaría la MV del CM, pero en este caso, como también está instalada la CLI, solo apagará CM. Si se desea eliminar la MV, cabe recordar que se perderá todo lo que se haya realizado en ella), salir de la conexión interactiva (*Crtl* + *D*) y ejecutar (si se desea volver a poner en marcha solo indicar `up` como al principio):

```
vagrant destroy -f
```

Si se desea borrar completamente la MV desde el *host*, ejecutar:

```
vagrant box remove cloudify-box
```

1.4.2. Instalación local con CLI

La distribuciones actuales de Cloudify CLI son para Centos 6+(RHEL 6+), Debian 7+/Ubuntu 14+ y arquitecturas x64 only. Para instalarlo simplemente bajarse el paquete deb/rpm desde el sitio del desarrollador (previo registro), y como *root* ejecutar (reemplazar <pkg.rpm> o <pkg.deb> con el paquete descargado:

```
rpm -i <pkg.rpm >      para Centos/RH
dpkg -i <pkg.deb>      para Debian/Ubuntu
cfy -h
cfy --version
```

Después de la instalación se desplegará un pequeño sitio web de ejemplo a través de un *blueprint* para mostrar su funcionamiento que se deberá descargar y extractar:

```
unzip simple-python-web-server-blueprint-master.zip
cd simple-python-web-server-blueprint-master
cfy local init --blueprint-path blueprint.yaml
cfy local execute --workflow install
```

Donde el primer comando `cfy` es la inicialización del entorno y el segundo el despliegue del *workflow* donde se podrá ver el *host* y puerto (*localhost:8000* en este caso) donde atenderá el servicio (valores por defecto, pero se puede modificar con el parámetro - *-inputs* durante la inicialización). Es este caso, se ha desplegado sobre los recursos locales, pero a través de los *plugins* y con el *blueprint* correspondiente se podría desplegar sobre otro *cloud*.

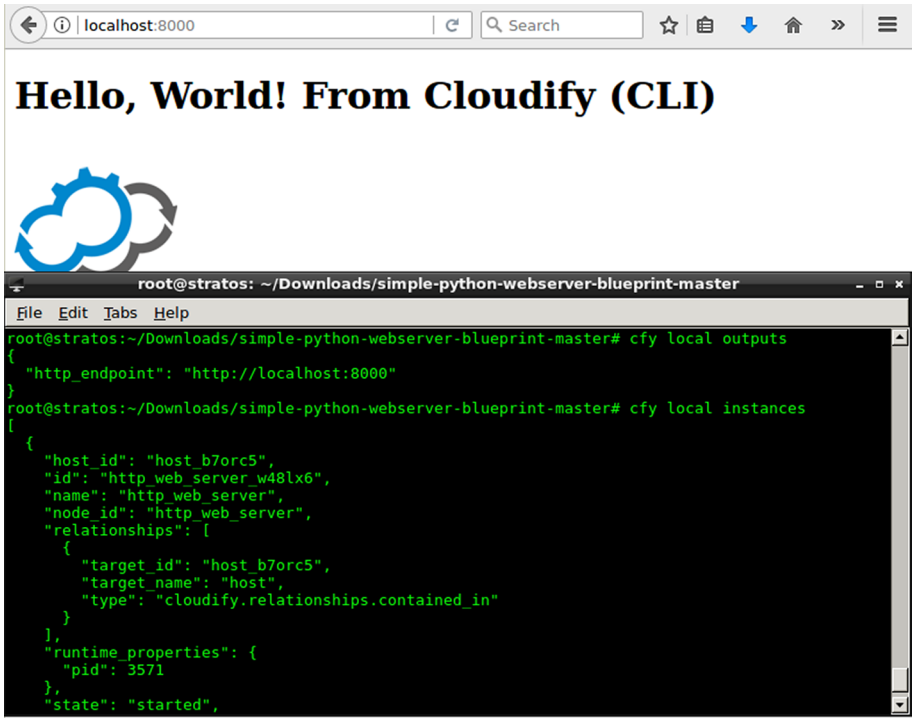
Los comandos siguientes mostrarán la configuración (donde el servicio atiende) de sitio desplegado y la información de la instancia:

```
cfy local outputs
cfy local instances
```

Para desinstalar la aplicación:

```
cfy local execute -w uninstall
```

La imagen siguiente muestra el sitio desplegado y la terminal donde se muestran los comandos antes mencionados.



Es conveniente consultar la documentación para conocer la estructura de los *blueprints* y la sintaxis/argumentos del comando `cfy`.

1.4.3. Instalación del Cloudify Manager (CM)

La instalación del CM requiere unos prerrequisitos de como mínimo 2 vCPU, 4GB RAM y 5GB de disco y Centos 7/RHEL 7 (en este caso se instalará sobre una MV KVM Centos 7). En esta prueba se instalará el CLI y el CM sobre la misma MV, que será también donde se realicen los despliegues, por lo cual es necesario disponer de acceso por SSH y por clave pública/privada al usuario *root* y permitir que este se pueda conectar como tal a la máquina, en resumen:

```
yum install openssh-server
ssh-keygen
ssh-copy-id <Ip MV>
vi /etc/ssh/sshd_config            modificar PermitRootLogin yes
```

Verificar que se puede acceder como *root* y sin *passwd* a la MV. Siguiendo los pasos del punto anterior, instalar la CLI de Cloudify, crear un directorio, cambiarse a él e inicializarlo con:

```
cfy init
```

Este comando generará un directorio *.cloudify*. Para desplegar CM se utilizará el mismo mecanismo de Cloudify basado en *blueprints*, por lo cual, dentro del directorio creado hay que copiar, desde */opt/cfy/cloudify-manager-blueprints/*, los archivos y directorios (no se utilizarán todos, pero es más práctico así):

```
cp -R /opt/cfy/cloudify-manager-blueprints/* .
```

Modificar en el archivo *simple-manager-blueprint-inputs.yaml* los siguientes parámetros: *public_ip*, *private_ip* (en ambos casos poner la IP de la MV), *ssh_user: 'root', ssh_key_filename: '/root/.ssh/id_rsa.pub', agents_user: 'root'*, y si no se dispone de mucha memoria, cambiar *minimum_required_total_physical_memory_in_mb: '3492'*. Luego iniciar el despliegue:

```
cfy bootstrap --install-plugins -p simple-manager-blueprint.yaml \
 -i simple-manager-blueprint-inputs.yaml
```

Con esto se iniciará la descarga de todos los paquetes necesarios, se realizará la configuración y después de un cierto tiempo, si no se han producido errores, se verá algo como:

```
Bootstrap complete
Manager is up at <ip MV>
```

En caso contrario, sobre la terminal se verán los errores causados, los cuales se deberán subsanar, y se tendrá que ejecutar nuevamente el comando `cfy bootstrap`. Para ver el estado de los servicios y de CM, ejecutar:

```
cfy status
```

Y ya se podrá conectar al CM poniendo como URL *<ip_MV>*. Para probar su funcionamiento, se utilizará el mismo ejemplo que utilizado anteriormente (bodega). Para ello, descargarse el ejemplo de Github en (formato zip), descomprimirlo y cambiarse al directorio. La carga del *blueprint* se puede hacer desde la interfaz gráfica o desde la CLI como:

```
cfy init
cfy blueprints upload -b nodecellar -p simple-blueprint.yaml
```

Para inicializar el despliegue es necesario modificar el fichero de entradas a partir del *template* en el directorio del ejemplo *inputs/singlehosts.yaml.template* renombrándolo *singlehost.yaml* con el siguiente contenido:

```
host_ip: 'ip_MV'
agent_user: 'root'
agent_private_key_path: '/root/.ssh/id_rsa'
```

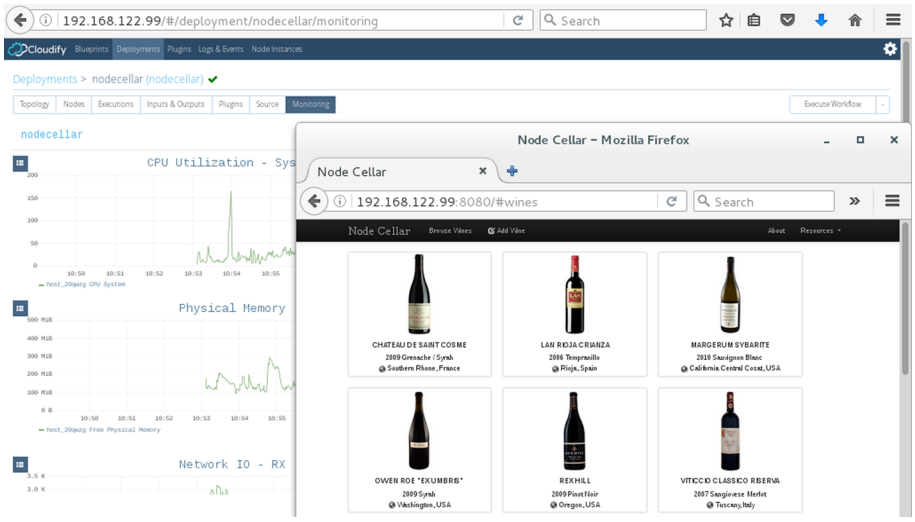
Después se puede hacer la inicialización como (también se podría hacer desde la UI):

```
cfy use -t <ip_MV>
cfy deployments create -b nodecellar -d nodecellar \
   --inputs inputs/singlehost.yaml
```

Con ello, ya se verá la inicialización del despliegue y se podrá ejecutar este, ya sea desde la UI o desde la CLI con:

```
cfy executions start -w install -d nodecellar
```

Sobre la UI se podrán ver las diferentes acciones y el detalle en la pestaña de *Logs &Events* y finalmente, cuando finalice el despliegue, conectarse a la URL <*ip_MV*>:8080 como se muestra en la figura siguiente.



Como se puede apreciar, es una plataforma muy extensa y completa que permite múltiples opciones, como PaaS, que el *Sysadmin* deberá analizar y experimentar para adecuarla a su entorno y necesidades [Cdo][Cco].

1.5. Dokku

Dokku es una plataforma PaaS extensible, *open-source*, basada en Docker; sus autores la consideran como la PaaS más pequeña que se conoce o como «mini-Heroku». Sus requerimientos son muy básicos (Ubuntu 16.04+ x64/Debian 8.2+ x64/ CentOS 7+ x64 –experimental– y como mínimo 1GB de RAM libre) y puede ser instalada sobre una MV (el procedimiento es similar).

Dokku tiene como base Docker, Nginx y Git como paquetes principales para gestionar las aplicaciones y está formado por un conjunto de *scripts* y comandos que, juntos, construyen una especie de *pipeline* donde la entrada es el código del usuario y la salida la aplicación ejecutándose.

El objetivo de la plataforma es proveer un entorno simple y modificable/extensible para que el desarrollador pueda, rápidamente, pasar código de su portátil al *cloud* y preocuparse solo por el código y no por el ciclo de vida de la aplicación.

Entre los comandos utilizados destacan `sshcommand` (simplifica la ejecución de un único comando sobre SSH y gestiona ACL), `plugn` (permite extender una aplicación con *plugins*) y `herokuish` (emula Heroku en la construcción/ejecución de tareas sobre contenedores) [Ddo].

Existe un proyecto llamado Dokku Alternative, que nace como una bifurcación de Dokku, para potenciar algunas opciones/UI y con otros desarrolladores, pero hace dos años que no está mantenido y en Github recomiendan migrar a Dokku original.

1.5.1. Instalación sobre una MV y despliegue de una aplicación

Su instalación es sumamente sencilla mediante la ejecución de un *shell script bash* (para esta prueba se utilizará una MV KVM con Ubuntu 16.04 server + LXDE para facilitar el trabajo mediante múltiples terminales). Para ello ejecutar como *root*:

```
wget https://raw.githubusercontent.com/dokku/dokku/v0.8.0/bootstrap.sh;
DOKKU_TAG=v0.8.0 bash bootstrap.sh
```

La instalación tardará unos instantes, dependiendo de la velocidad de la red, y cuando finalice se deberá abrir un navegador para introducir unos parámetros y la llave pública de SSH (si no se dispone de ella generarla con `sshkeygen`) en la *<IP_MV>*. Es importante realizar esta acción cuanto antes, ya que si la máquina dispone de conexión externa (por ejemplo, si es una MV en el *cloud*) cualquiera que introduzca su llave podrá desplegar *apps* sobre Dokku sobre ella. Para copiar la llave pública hacer:

```
cat $HOME/.ssh/id_rsa.pub
```

y copiarla (no utilizar el comando `more`), y si no se dispone de interfaz gráfica/copia-pegar existe un procedimiento por CLI para hacerlo. Sobre esta página se ha configurado la IP del *host* donde se está ejecutando Dokku y sin dominios virtuales, sino a través de un puerto. Consultar la documentación sobre la instalación en diferentes proveedores de IaaS. Luego de finalizada la

instalación se podrá ejecutar `dokku` para ver todos los comandos que dispone y verificar que todo está funcionado (por ejemplo, con `apps` o `ls` como argumentos para ver las *apps* o contenedores).

Para desplegar una aplicación se realizará vía el comando `git push` y, en este caso, se utilizará una aplicación de ejemplo Heroku Ruby on Rails.

1) En primer lugar, **descargar la aplicación** en la máquina local:

```
git clone https://github.com/heroku/ruby-rails-sample.git
```

2) Crear la apps: sobre el *host* Dokku (se debe tener acceso vía *ssh*; en nuestro caso, como es local no se debe hacer nada):

```
dokku apps:create ruby-rails-sample
```

3) Crear los servicios: Dokku no provee BD integradas, sino que deben instalarse como *plugins* (en este ejemplo será PostgreSQL), y posteriormente crear la BD y enlazarla con la aplicación; para ello ejecutar como *root*:

```
dokku plugin:install https://github.com/dokku/dokku-postgres.git
dokku postgres:create rails-database
dokku postgres:link rails-database ruby-rails-sample
```

4) Desplegar la aplicación: del entorno local (*ruby-rails-sample*) al servidor Dokku. Para ello, desde el directorio de la *app*:

```
git remote add dokku dokku@<ip_MV_servidor_Dokku>:ruby-rails-sample
git push dokku master
```

Cuando el despliegue termine dará como resultado la URL: puerto donde se podrá visualizar la aplicación. Luego se puede modificar la aplicación y para subir los cambios es simplemente (desde el directorio de la aplicación):

```
git add .
git commit -m "V2.0"
git push dokku master
```

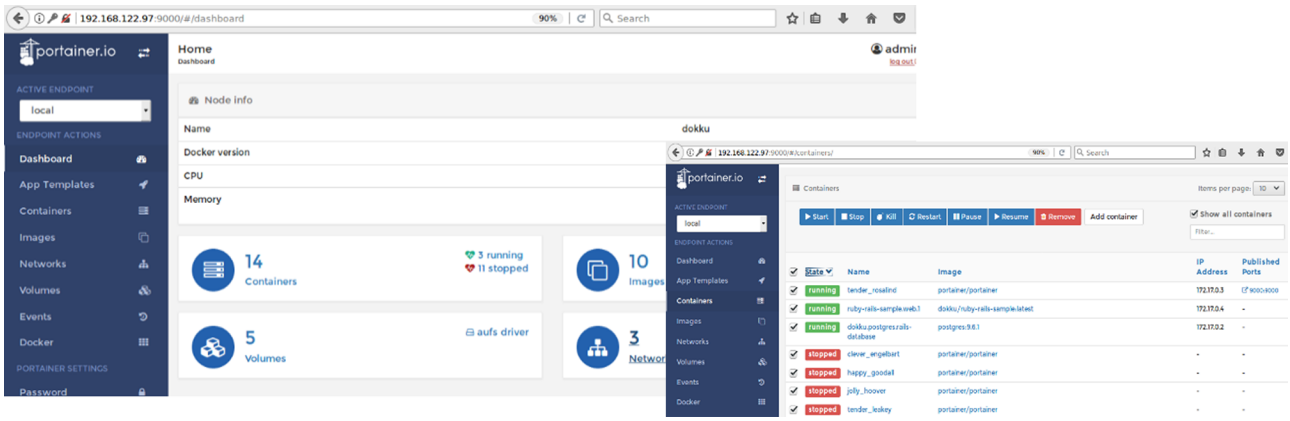
La imagen muestra la consola donde se ha realizado el despliegue y la visualización de la aplicación ejecutándose sobre el servidor (modificada en este caso).

Existen diversas herramientas/adaptaciones para proveer de una interfaz gráfica a Dokku como Doku-Dash, Dokku-Dashboard; o también se pueden utilizar herramientas administración *open-source* de Docker como Portainer (disponible en Github). En este caso, se utilizará esta última para visualizar y gestionar los *containers* sobre la plataforma. Su despliegue es sumamente simple, ya que está integrado en un contenedor, por lo cual sobre la MV donde se tiene instalado Dokku ejecutar (dado que se vinculará a Docker a través de su *socket* se debe indicar cuál es en la ejecución del contenedor):

```
docker run -d -p 9000:9000 -v \
   /var/run/docker.sock:/var/run/docker.sock portainer/portainer
```

Por medio de un navegador en la URL *http://<IP_MV>:9000* se podrá acceder a la aplicación, que la primera vez solicitará un *passwd*, y se deberá configurar el tipo de acceso seleccionando *Local Engine*. Portainer dispone de diferentes opciones de configuración y funcionalidad que se pueden consultar en su documentación. Existe una integración entre Dockerui (versión anterior de Portainer) que, con algunas adaptaciones, puede funcionar también para Dokku+Portainer.

Las imágenes siguientes muestran la ejecución de Portainer con las aplicaciones desplegadas en Dokku.



Existen una serie de tareas adicionales sobre Dokku a las que es necesario dedicarle atención, como: gestión de la *app* (redesplegar, eliminar, clonar, etc.), *logs*, escalado, despliegue de módulos privados/subdominios, gestión de usuarios, despliegue de *buildpacks*, configuración de DNS/Nginx/SSH, y otras tareas de interés que el *SysAdmin* deberá consultar en la documentación [Ddo].

Las ventajas de la plataforma han permitido que sea un reemplazo atractivo a Heroku y existen gran cantidad de documentación sobre la instalación de Dokku sobre VPS (por ejemplo, Morizyum, Exygy, Codehero) con un bajo costo o proveedores de IaaS que la han integrado dentro de su catálogo y con un simple clic se dispone la plataforma en funcionamiento sobre la infraestructura (por ejemplo, DigitalOcean, Packagecloud).

1.6. Tsuru

Tsuru es una PaaS extensible y *open-source* que facilita el desarrollo e implantación de las aplicaciones sin pensar en servidores/servicios; el desarrollador puede utilizar el lenguaje que prefiera y se permiten aplicaciones con *plugins* (como DB SQL, NoSQL, o *memcached*, *Redis*, entre otras), con una administración mediante CLI y con Git como única herramienta para el despliegue aplicaciones.

Entre las principales ventajas de la plataforma, se pueden enunciar:

- despliegue rápido, fácil y continuo, sin herramientas especiales (solo `git`) y sin complicaciones por las dependencias,
- posibilidad de crear versiones (*testing*, *staging* y *production*),
- gestión de la escalabilidad simplificada (solo añadir unidades y la plataforma hará el resto), y

- confiabilidad (dispone de un conjunto de herramientas que permite asegurarse de que las aplicaciones siempre están disponibles) [Tdo].

Existe una **serie de conceptos** que aplica la plataforma, entre los cuales es conveniente destacar:

- **Docker**: es el contenedor utilizado por la plataforma, de forma que cuando se despliega una *app* con `git push` o `tsuru app-deploy`, Tsuru creará una imagen Docker y la distribuirá a través del *cluster*.

- ***Clusters***: es un conjunto de nodos sobre las cuales Tsuru distribuirá las aplicaciones.

- ***Nodes***: es un nodo físico o una MV con Docker instalado. El nodo puede ser «gestionado» (creado y gestionado por Tsuru utilizando integración con el IaaS), y «no-gestionado» (el nodo es creado manualmente y solo registrado con Tsuru, pero este no puede gestionarlo).

- **Aplicaciones**: incluye el código fuente de una aplicación (Python, Ruby, Go, PHP, JavaScript, Java, etc.), las dependencias del SO (en un archivo *requirements.apt*), las del leguaje (en un archivo *requirements.txt*, *Gemfile*, etc.) y las instrucciones de cómo ejecutar el programa (en un archivo llamado *Procfile*).

- ***Units***: una *unit* es un contendor necesario para ejecutar una aplicación.

- ***Platforms***: es un *stack* con las dependencias instaladas para un lenguaje o entorno que será necesario para un tipo de aplicaciones y que puede ser un *containertemplate* (*Docker image*). Por ejemplo, Tsuru dispone de una *container image* para aplicaciones Python con *virtualenv* instalado y otros requerimientos necesarios para el desarrollo de aplicaciones en este lenguaje. Cada aplicación se ejecuta por encima de una plataforma.

- ***Services***: es una API bien definida con la que Tsuru se comunica para proveer funcionalidad extra como MySQL, Redis, MongoDB, etc. Tsuru dispone de servicios propios, pero es fácil crear y añadir nuevos servicios a Tsuru.

La **arquitectura** de Tsuru está formada por:

- **Daemon de Tsuru** (*tsurud*): servidor de API RESTful escrito en Go y responsable de gestionar los *workflows* de despliegue y del ciclo de vida de las aplicaciones.

- **Cliente CLI** (`tsuru` y `tsuru-admin`) y Tsuru *dashboard*.

- **Base de datos**: servidor MongoDB.

- ***Queue/Cache***: servidor Redis.

- ***Gandalf***: API REST para manejar repositorios y usuarios Git y acceso a ellos a través de SSH.

- **Registro**: se utiliza el *Docker registry* para almacenar y distribuir *Docker images*.

- ***Router***: responsable de redireccionar el tráfico a las *units* de la aplicación (contenedores Docker).

En la documentación se detallan los pasos para una instalación en producción sobre Ubuntu Server 14.04 LTS, pero para pruebas es recomendable utilizar Tsuru Now, que es un *script* que instalará *tsuru API*, *tsuru Client*, *tsuru Admin* y todas las dependencias sobre una única máquina junto con un nodo Docker para ejecutar las aplicaciones desplegadas. Otra opción es utilizar Tsuru-bootstrap, que es una implementación de *tsuruNow* sobre una MV Vagrant y que será la que se utilizará en este caso.

Antes de iniciar la instalación Tsuru, es necesario dedicar tiempo a la documentación para conocer los conceptos y aspectos que son necesarios para poder luego desplegar una aplicación (*platforms*, *containers*, *pools*, *teams*, *scheduler*, *register*, *git repositories*, *SSH keys*, *users and permissions*, *roles*) [Tdo].

1.6.1. Instalación y despliegue de *apps* utilizando Tsuru BootStrap

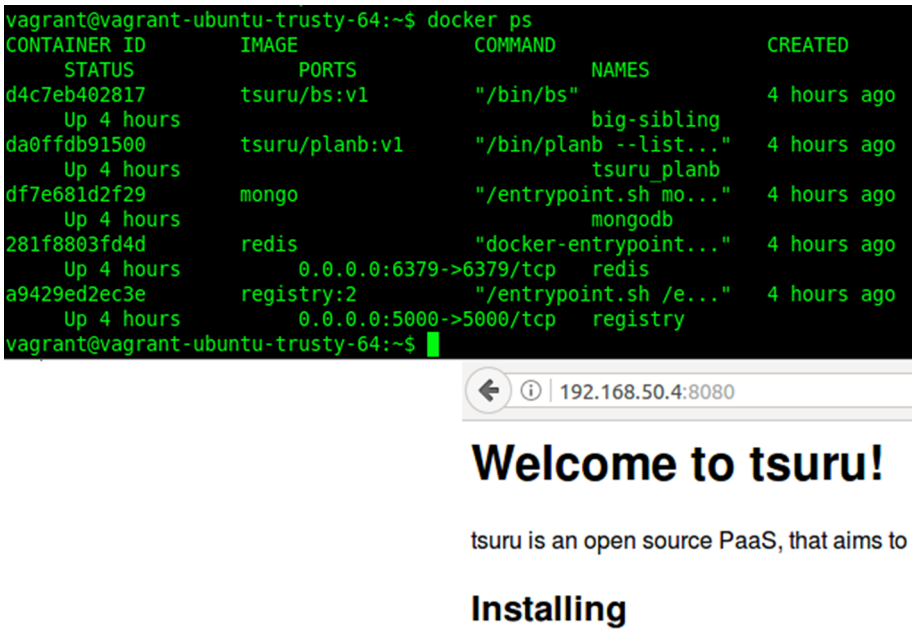
Para esta prueba se ha utilizado una MV KVM (3 GB RAM, 30GB disco, un adaptador de red en NAT y con la virtualización anidada activada: *host-passthrough*) sobre la que se ha instalado Ubuntu 14.04 desde Minimal ISO y los paquetes *openssh-server*, *firefox* y *Lubuntu* (es útil disponer de un entorno gráfico para disponer de varias terminales simultáneas y poder copiar y pegar). Se ha utilizado esta versión, ya que con posteriores se han obtenidos problemas (errores de instalación y puesta en marcha de la MV).

Sobre esta máquina (que la llamaremos *TsuruHost* ya que es la MV que ejecutará la segunda MV Vagrant-Virtualbox que realmente ejecutará Tsuru y que llamaremos *TsuruNode*) también se ha instalado Vagrant y VirtualBox desde la distribución, pues serán necesarias para arrancar Tsuru-Bootstrap (este proyecto consta de una serie de *scripts* que inicializa una máquina Vagrant e instala Tsuru Now sobre ella) y luego se ha clonado el repositorio y creado la MV. Para ello, como *root* (tardará un cierto tiempo en función de la velocidad de la red):

```
apt-get install vagrant git
apt-get install virtualbox
git clone https://github.com/tsuru/tsuru-bootstrap.git tsuru
cd tsuru
```

```
vagrant up
vagrant shh
```

Con esta última orden se conectará por `ssh` sobre la MV Vagrant (*TsuruNode*) y se podrán ver los servicios desplegados a través de comando `docker ps` (se pueden ver las imágenes con `docker images`), como se muestra en la figura.



También se puede observar que el *daemon tsurud* se está ejecutando y que se puede acceder al servicio inicial desde el navegador (en la máquina *TsuruHost*) a la URL http://192.164.50.4:8080, que mostrará la página de bienvenida a la Tsuru.

Para la configuración se han realizado los siguientes pasos (no son exactamente iguales a los de la documentación, pero se ha procedido así para evitar ciertos problemas encontrados), desde dentro de la MV Vagrant:

1) Definir el *default target*: `tsuru target-add default http://192.168.50.4:8080 -s`

2) Crear usuario *root* (pedirá el *passwd* dos veces): `tsurud root-user-create <email-root-user>`

3) Autenticarse como *root*: `tsuru login <email-root-user>`

4) Crear usuario normal (pedirá el *passwd* dos veces): `tsuru user-create <email-user>`

5) Asignar rol a usuario (normal):`tsuru roll-assign AllowAll <email-user>`

6) Crear el *pool* y *team* y asignarlo:

```
tsuru pool-add mypool
tsuru team-create myteam
tsuru-admin pool-teams-add mypool myteam
```

7) Crear el nodo:

```
tsuru-admin docker-node-add pool=mypool -register address=http://localhost:2375
```

8) Cargar una plataforma desde el repositorio Tsuru (PHP en este caso):

```
tsuru-admin platform-add php \
-d https://raw.github.com/tsuru/basebuilder/master/php/Dockerfile
```

Con esto se tendría configurado Tsuru y se podría consultar información sobre los *target*, *team*, *pool*, *app*, *roles*, *user*, *platform*, *node*, etc. (para verificar que todo está bien) con el comando `tsuru` los siguientes argumentos (sin argumentos mostrará todos los posibles): *target-list*, *team-list*, *app-list*, *pool-list*, *role-list*, *user-list*, *docker-node-list*...

A partir de esta configuración ya se podrá desplegar una aplicación que, como prueba, se hará desde el *TsuruHost* para simular un entorno real de trabajo. Para ello es necesario (se puede desconectar de MV Vagrant con Crlt-D o utilizar otra terminal) instalar sobre esta el cliente Tsuru (en este caso se utilizará tsuru-1.1.1-linux_amd64.tar.gz). Como aplicación se ha descargado un *site web* desarrollado en PHP (por K.Ahmed y en el repositorio Students3k) y se ha modificado el aspecto (también se puede utilizar simple-php-website).

A continuación, se ha creado la aplicación y se ha desplegado el *site* desde *TsuruHost* (es decir remotamente, aunque en esta prueba todo está en la misma máquina, pero en diferentes MV):

1) Definir el *default target*: `tsuru target-add default http://192.168.50.4:8080 -s`

2) Autenticarse como usuario (no como *root*): `tsuru login <email-user>`

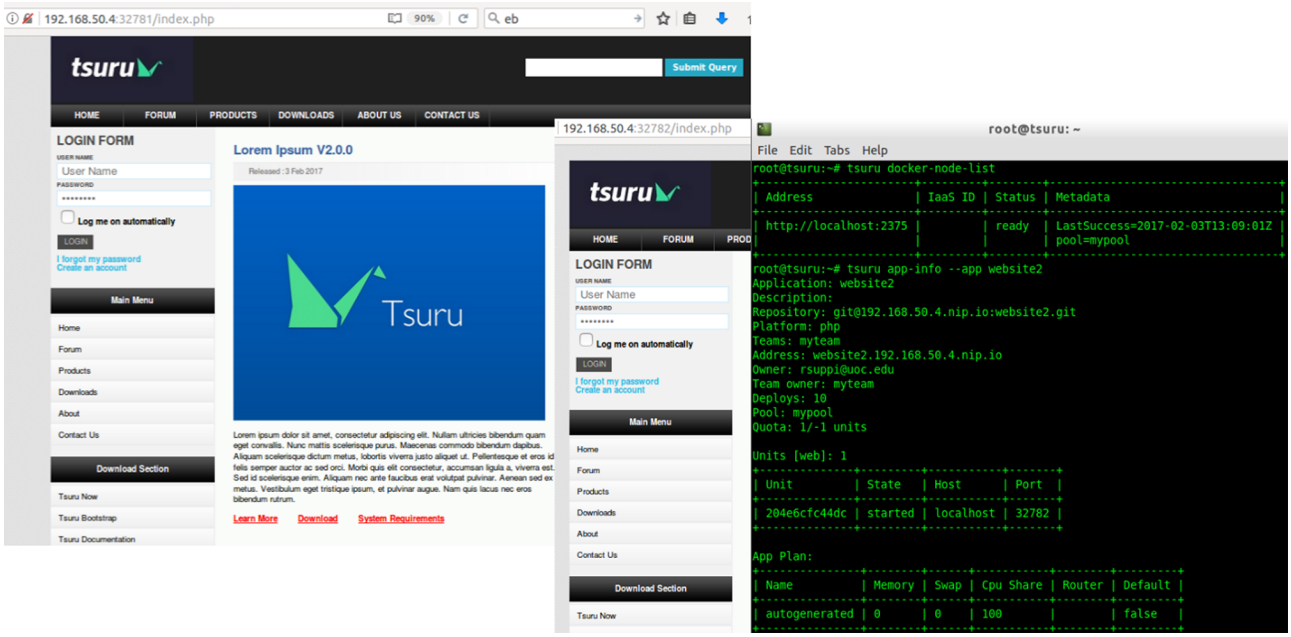
3) Crear la aplicación: `tsuru app-create website2 php -t myteam`

4) Desde el directorio de la aplicación:
`tsuru app-deploy . --app website2`

5) Finalizado el despliegue se podrá ver la información de la aplicación con

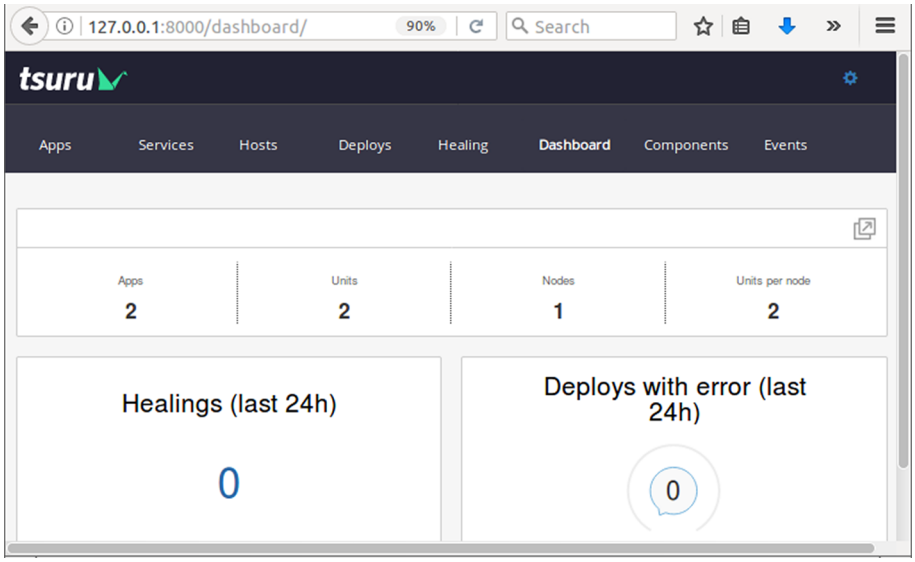
`tsuru app-info --app website2` donde se mostrará el puerto donde se puede acceder a ella. Luego de esto se podrán modificar y modificar/realizar el *deploy* tantas veces como se desee, gestionar la aplicación (*app-stop/app-start*, ver información (*app-log*, *app-deploy-list*), etc.

Las figuras siguientes muestran la ejecución desde la URL indicada y la información de la aplicación y su ejecución.



Tsuru cuenta también con un simple *dashboard* (escrito en Django) que se puede descargar, compilar y conectar al servidor Tsuru mediante una variable de entorno (en nuestro caso, *export TSURU_HOST*) y se pondrán en marcha en http://localhost:8000 (ver figura siguiente).

Como se puede ver, es una plataforma muy interesante, si bien ponerla a punto sobre una MV y dentro de otra MV implica un trabajo experimental adicional que se debe tener en cuenta. Es importante, dadas las prestaciones y funcionalidades que incluye, que el SysAdmin consulte la documentación sobre los aspectos de instalación y administración de la plataforma, ya que aquí solo se han descrito algunos de ellos y muy brevemente.



1.7. Flynn

Flynn es una plataforma PaaS *open-source* para ejecutar aplicaciones en producción; para algunos expertos, forma parte de la siguiente generación de entornos de desarrollo y la comparan con Deis y son consideradas como las dos mayores exponentes de la tecnología Micro-PaaS/Microservices (plataformas basadas en Docker y consideradas la última tendencia en PaaS de la cual forma parte además Tsuru, Rack, o Nanobox entre otras).

Flynn (como empresa) dispone de servicios Managed Flynn donde instalan y configuran/mantienen instalaciones de Flynn sobre el *cluster/cloud* del desarrollador para evitar que este tenga que disponer de los recursos humanos/físicos de IT y en un servicio 24/7 (para 25 nodos y soporte 24/7 el precio es de 4.299$/mes o 9.999$/mes para 50 nodos) [Fvi][Fdo].

Entre las principales **características** de Flynn, se pueden enumerar:

- Crea aplicaciones en ejecución directamente desde el código, por lo cual el desarrollador no necesita conocer nada de la administración ni de la configuración o crear *stack* personalizados para cada aplicación.

- Gestiona las rutas y los balanceos de carga del tráfico de cada aplicación/instancias desplegadas.

- Incluye bases de datos integradas como Postgres, MySQL y MongoDB, con alta disponibilidad automática y segura.

- Interconecta todos los microservicios y dispone de un descubrimiento de servicios integrado.

- Escala la aplicación con el tamaño adecuado para evitar costes innecesarios dentro del *cloud*.

La arquitectura Flynn está diseñada para ser simple y eficiente, y los componentes no son diferentes de las aplicaciones que se despliegan a través de ella.

Su objetivo es proporcionar una plataforma sólida para desplegar aplicaciones, con alta disponibilidad, fácil de ejecutar y administrar, y con una configuración mínima.

La plataforma está casi en su totalidad escrita en Go y utiliza JSON sobre HTTP para la comunicación, tanto interna como externa, por su facilidad de depurar e intercambiar información a través de este método y, como la mayoría de los desarrolladores lo conoce, es muy fácil interactuar con la API. Si bien la plataforma no tiene dependencias estrictas de SO, se toma como referencia Ubuntu 14.04 LTS amd64 como sistema operativo base.

El ***daemon flynn-host*** es el componente de nivel más bajo y es el único servicio que no funciona dentro de un contenedor; proporciona la API para iniciar y administrar contenedores, que es donde se ejecuta todo lo demás. Las API que ofrece *flynn-host* no son específicas de los contenedores de Linux, y se denomina a una unidad de trabajo en ejecución un *job*.

Después de que el *daemon flynn-host* se inicia sobre los *hosts*, la herramienta *bootstrap* crea un *cluster* Flynn encima de ellos, y los configura/pone en marcha a través de un manifiesto JSON. A continuación, se arrancan los servicios indicados y ya se tiene disponible el sistema en funcionamiento para atender a los desarrolladores.

Otros componentes, no menos importantes, son *flannel* (configuración de una red VXLAN), PostgreSQL, *Controller*, *Router*, *Buildpacks*, y *Log Agregator*, de los cuales se pueden encontrar sus detalles en la documentación [Fdo].

1.7.1. Instalación

Flynn puede ser instalado mediante una interfaz a través de un navegador sobre determinados *clouds* (AWS, Digital Ocean, Azure) o sobre servidores propios por SSH utilizando un Cloud Installer o también en un entorno local utilizando una máquina Vagrant como prueba y para experimentar con el entorno (no para producción), que es lo que se utilizará en este caso.

Su instalación y configuración es sumamente sencilla y se puede poner en marcha y desplegar aplicaciones muy rápidamente. Para ello, se han seguido las indicaciones de la documentación utilizando un MV KVM similar a la uti-

lizada en otras plataformas (4 GB RAM, 30GB disco, un adaptador de red en NAT, la virtualización anidada activada *–host-passthrough–*, Ubuntu 14.04 mas *git*, *virtualbox*, *openssh-server*, *firefox* y *Lubuntu*).

En este caso se ha tenido que descargar la última versión de Vagrant (desinstalando la del repositorio), ya que Flynn necesita la versión 1.6+. Sobre esta máquina virtual se ha ejecutado:

```
git clone https://github.com/flynn/flynn
cd flynn/demo
make init
apt-get install xdg-utils
make dashboard
```

Con esto se abrirá el navegador; deberemos aceptar el certificado y ya tenemos acceso a la plataforma instalada y en ejecución. Si se abre directamente la URL a través de un navegador (*http://dashboard.demo.localflynn.com*), nos solicitará un *token* que se puede obtener con:

```
flynn -a dashboard env get LOGIN_TOKEN
```

y que se deberá copiar y pegar en la página web.

Estos comandos se ejecutan a través de comando `make`, pero también se puede hacer a través de los comandos Vagrant. Otros comandos útiles para gestionar la plataforma son (consultar el archivo *Makefile* por otros comandos útiles):

- Para la MV: `make down`
- Inicia la MV y Flynn: `make up`
- Conexión a través de SSH a la MV: `make ssh`
- Borra la MV: `make destroy`
- Para y reinicia la MV: `make down/make up`
- Borra y despliega la MV: `make reset`

Para el **despliegue básico** de una aplicación y mostrar la funcionalidad de la plataforma (para más detalles consultar la guía indicada por el desarrollador), se utilizará el *cluster* previamente desplegado y la CLI/Dashboard; se asume que no se ha modificado el dominio (*demo.localflynn.com*) por defecto de la MV Vagrant (si es así, cambiarlo por el que se ha dado en la variable CLUSTER_DOMAIN del archivo *Makefile* durante el proceso de inicialización).

La aplicación que se utilizará como ejemplo en muy simple (escrita en Go) y despliega un servidor mínimo de HTTP que interactúa con la DB Postgres. Para ello se clonará un repositorio de Github desde los ejemplos de despliegue a *apps* de Flynn y se creará la aplicación:

```
git clone https://github.com/flynn-examples/go-flynn-example go-flynn
cd go-flynn
flynn create myexample
```

Al ejecutar el comando `git remote -v` se podrá observar los repositorios *git* que ha agregado el comando anterior (y modificarlos si es necesario) y con el comando `flynn route` la URL de la nueva aplicación:

```
flynn https://git.demo.localflynn.com/myexample.git (push)
flynn https://git.demo.localflynn.com/myexample.git (fetch)
origin https://github.com/flynn/nodejs-flynn-example.git (fetch)
origin https://github.com/flynn/nodejs-flynn-example.git (push)
```

Route	Service	ID	Sticky	Leader	Path
http:myexample.demo.localflynn.com	myexample-web	http/54d74a23....	false	false	/

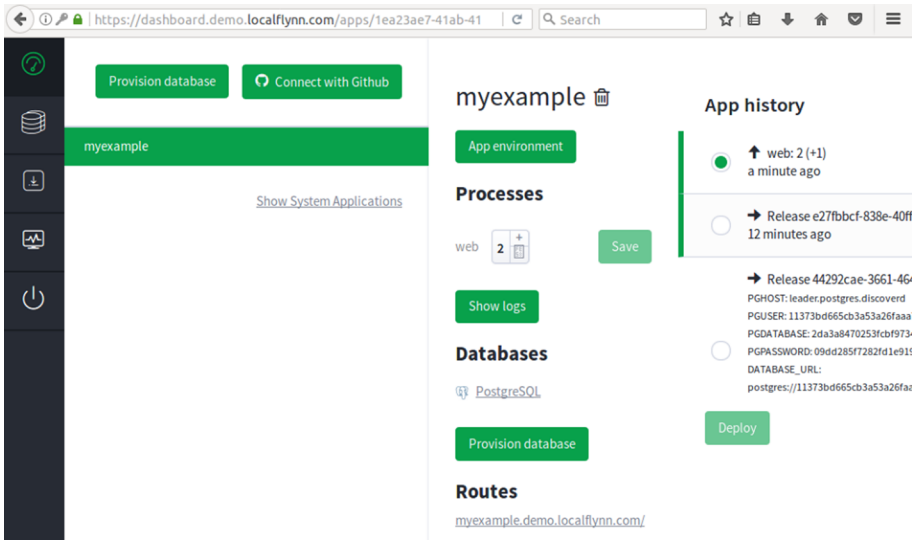
Dado que la aplicación necesita una BD, se añadirá Postgres a la aplicación:

```
flynn resource add postgres
```

Si se desea conocer la información inicializada para la BD se deberá ejecutar `flynn env`. El paso siguiente es desplegar la aplicación y acceder a ella a través de la URL indicada en *route* (a través de `curl` o a través de un navegador).

```
git push flynn master
curl -k http://myexample.demo.localflynn.com
Hello from Flynn on port 8080 from container 56a14f60…
Hits = 1
```

La figura siguiente muestra la interfaz gráfica y la información de la aplicación desplegada.



La siguiente prueba es el **escalado** de la aplicación; en este caso, se ha realizado desde la interfaz gráfica simplemente subiendo el indicador web de 1 a 2, pero se puede hacer desde la CLI.

Las aplicaciones declaran su tipo en el archivo *Procfile* (en el directorio *root* de la aplicación) y en este caso es *web: go-flynn-example*; Flynn, para este tipo de aplicaciones, le asigna por defecto un proceso/contenedor que se puede consultar con `flynn ps`. Si se desea escalar desde la CLI, se debe ejecutar `flynn scale web=2`, que iniciará todo el proceso de crear el nuevo contenedor y configurarlo para balancear la carga; cuando termine, se podrá observar que `flynn ps` devuelve los dos contenedores en marcha.

Si se ejecutan peticiones repetidas de HTTP, se verá que las respuestas son balanceadas a través de los dos diferentes contenedores, como muestra la figura siguiente (las primeras tres peticiones sobre el contenedor ...e30 y las siguientes sobre el contenedor ...fc6:

```
root@tsuru:~/flynn/demo/go-flynn-example# flynn route
ROUTE                            SERVICE        ID
 STICKY  LEADER  PATH
http:myexample.demo.localflynn.com  myexample-web  http/54d74a23-7db5-448c-a808-dd52331c2f50
 false   false   /
root@tsuru:~/flynn/demo/go-flynn-example# flynn ps
ID                                   TYPE  STATE  CREATED         RELEASE
flynn-8e731f23-fd25-4f6f-ab81-d65cc1022e30  web   up     15 minutes ago  e27fbbcf-838e-40ff-
766-e4dfd70d1a82
flynn-d390dbf9-9a24-4a2b-b10c-094321be4fc6  web   up     5 minutes ago   e27fbbcf-838e-40ff-
766-e4dfd70d1a82
root@tsuru:~/flynn/demo/go-flynn-example# curl http://myexample.demo.localflynn.com
Hello from Flynn on port 8080 from container 8e731f23-fd25-4f6f-ab81-d65cc1022e30
Hits = 14
root@tsuru:~/flynn/demo/go-flynn-example# curl http://myexample.demo.localflynn.com
Hello from Flynn on port 8080 from container 8e731f23-fd25-4f6f-ab81-d65cc1022e30
Hits = 15
root@tsuru:~/flynn/demo/go-flynn-example# curl http://myexample.demo.localflynn.com
Hello from Flynn on port 8080 from container 8e731f23-fd25-4f6f-ab81-d65cc1022e30
Hits = 16
root@tsuru:~/flynn/demo/go-flynn-example# curl http://myexample.demo.localflynn.com
Hello from Flynn on port 8080 from container 8e731f23-fd25-4f6f-ab81-d65cc1022e30
Hits = 17
root@tsuru:~/flynn/demo/go-flynn-example# curl http://myexample.demo.localflynn.com
Hello from Flynn on port 8080 from container d390dbf9-9a24-4a2b-b10c-094321be4fc6
Hits = 18
root@tsuru:~/flynn/demo/go-flynn-example# curl http://myexample.demo.localflynn.com
Hello from Flynn on port 8080 from container d390dbf9-9a24-4a2b-b10c-094321be4fc6
Hits = 19
```

Los ***logs*** (flujos de *stdout/stderr*) de todos los procesos son capturados y se pueden analizar con

```
flynn log
2017-02-03T17:52:59.366773Z app[web.flynn-8e73…]: hitcounter listening on port 8080
2017-02-03T18:03:06.489089Z app[web.flynn-d390…]: hitcounter listening on port 8080
2017-02-04T09:35:42.139301Z app[web.flynn-56a1…]: hitcounter listening on port 8080
2017-02-04T09:35:49.322287Z app[web.flynn-da52…]: hitcounter listening on port 8080
```

Las **modificaciones** sobre la aplicación y nuevo despliegue se pueden hacer fácilmente, por ejemplo, agregando la línea *fmt.Println* ("*He cambiado!*") al inicio de la función *main()* del archivo *main.go*, realizando el nuevo despliegue y verificando los cambios/*logs*:

```
git add main.go
git commit -m "Ver. 1.2"
git push flynn master
flynn ps
flynn log -n 6
```

Como se ha mostrado anteriormente, sobre la creación de un proceso web se crea una ruta por defecto en un subdominio del dominio por defecto (*default route domain*, por ejemplo, *myexample.demo.localflynn.com* en este caso). Si se desea utilizar un dominio diferente se puede **agregar otra ruta** (que será un CNAME a la inicial *example.demo.localflynn.com*).

```
flynn route add http nteum.demo.localflynn.com
flynn route
curl http://nteum.demo.localflynn.com
```

Donde se pueden agregar también dominios externos y modificar las cabeceras HTTP para tener comportamientos diferentes dentro de la aplicación.

Crear diferentes procesos y gestionarlos (escalarlos) es sumamente fácil; por ejemplo, sea la *app* dada en la guía antes mencionada, que es un comando simple que imprime la hora cada segundo. Para ello, crear un directorio/archivo *clock/main.go* y como contenido:

```
package main
import (
    "fmt"
    "time"
)
func main() {
    for t := range time.NewTicker(time.Second).C {
        fmt.Println(t)
    }
}
```

En el mismo directorio, crear un archivo Procfile que tenga una línea con el siguiente contenido *clock*: *clock* y crear, desplegar la aplicación y escalarla a 1 proceso:

```
git add clock/ Procfile
git commit -m "Add clock service"
git push flynn master
flynn scale clock=1
```

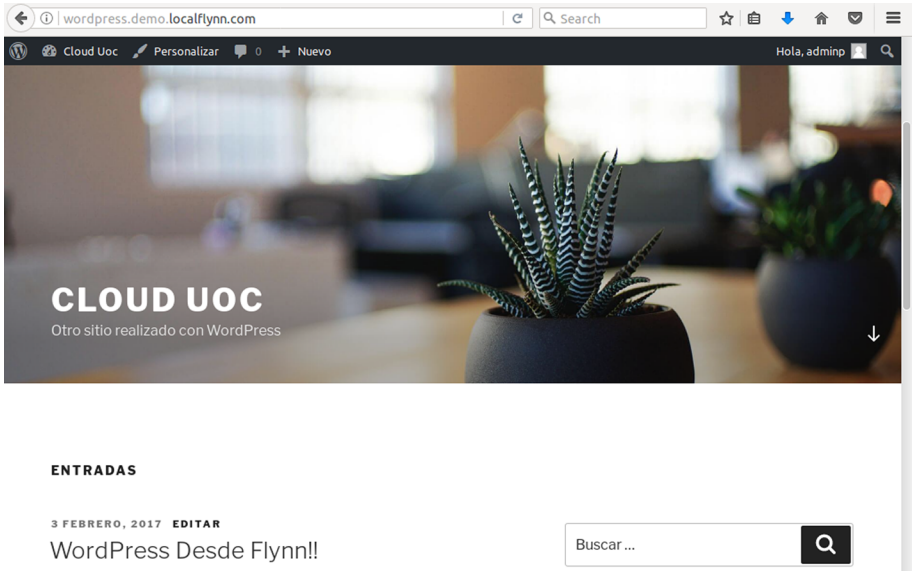
Y con el comando `flynn ps` se podrá ver el comportamiento de la aplicación, con `flynn log -t clock` los *logs* generados y con `flynn run bash` se podrá ejecutar un *shell* interactivo dentro de un contenedor. Es interesante consultar la ayuda de la CLI (`flynn`) en la documentación de la plataforma, así como las tareas avanzadas para crear y gestionar las aplicaciones, bases de datos, lenguajes, y Docker; y también las indicaciones que da el desarrollador para los sistemas en producción y diferentes aspectos relacionados con la plataforma [Fdo].

Dada la facilidad y funcionalidad con la plataforma, se desplegará una aplicación compleja para valorar las prestaciones y el grado de dificultad. Para ello, se ha escogido desplegar WordPress y se realizará con el repositorio ya creado en Github por Zomnium. Los pasos serán clonar el repositorio, crear la aplicación, crear la BD, desplegar la aplicación y conectarnos a ella:

```
git clone https://github.com/zomnium/flynn-wp.git wordpress
cd wordpress
flynn create wordpress
flynn resource add mysql
```

```
git push flynn master
```

Con ello ya se podrá conectar a la URL dada por *flyn route* y configurar la aplicación. La figura siguiente muestra la aplicación desplegada en unos pocos minutos.



Así como en otras plataformas PaaS su configuración y despliegue implica tiempo y conocimientos, Flynn (y en una primera toma de contacto) es una plataforma muy simple y fácil de poner en marcha, sin mayores inconvenientes, estable y con una gran funcionalidad apta para diferentes tipos de entornos IT/desarrolladores. Es evidente que en esta prueba solo se han tocado algunos aspectos básicos, y se deberán profundizar conceptos y aspectos sobre la instalación en un clúster o un *cloud* a través de las guías que provee la documentación.

1.8. AppScale

AppScale es una plataforma *open-source* orientada al desarrollo de aplicaciones *cloud* que automáticamente despliega y escala aplicaciones Google App Engine (sin modificar) sobre *clouds* públicos y privados y sobre *clusters* locales.

Es decir, AppScale implementa Google Cloud Platform, App Engine, en código abierto y además de poder ejecutarlos en GCP-AE permite ejecutarlos en cualquier otro lugar, es gratuito y fácil de usar y permite experimentar sobre *apps* basadas en AE sin el costo que podría tener en GCP-AE (después de haber pasado el período de prueba).

AppScale admite *apps* desarrolladas para la API de App Engine y soporta como lenguajes Python, Go, PHP y Java. Entre los *cloud* públicos que soporta se encuentran Google Compute Engine, Microsoft Azure, Amazon EC2, Alibaba Cloud, IBM SoftLayer, Rackspace, DigitalOcean, OpenStack, CloudStack, Eucalyptus, y también KVM, Xen, VirtualBox, y VMWare [Ado][Ard].

Los desarrolladores se pueden beneficiar de AppScale, ya que cuentan con una plataforma que permite desarrollar aplicaciones en forma muy rápida sobre las infraestructuras antes mencionadas; desacoplan la capa de la aplicación de la infraestructura subyacente, lo que permite a los administradores gestionar el entorno de desarrollo, el almacenamiento, la utilización de recursos, respaldos y migraciones.

AppScale fue inicialmente un proyecto de investigación liderado por C. Krintz en la Universidad de California (CSD) y, en 2012, se creó la empresa del mismo nombre para comercializar la infraestructura/servicios PaaS [Ado].

AppScale se puede desplegar en relación con las necesidades de los desarrolladores desde un entorno de Dev/Test de nodo único, hasta una implementación totalmente redundante y multinodo para aplicaciones de gran escala (las diferentes configuraciones se indican a través de un archivo AppScalefile).

El entorno (simple) Dev/Test facilita la prueba de *apps* antes de su expansión a producción y es la forma más sencilla y flexible de implementar una aplicación. Para ello se puede utilizar Vagrant y Docker como entornos flexibles y ligeros/replicables con mínimo esfuerzo. Una vez que las aplicaciones han sido depuradas y están preparadas para ir a producción, se puede utilizar AppScale multinodo con redundancia en el sistema de almacenamiento y escalabilidad con configuraciones adecuadas para soportar alta disponibilidad (por ejemplo, 6 máquinas: 3 nodos de base de datos –3x replicación de datos–, 2 nodos AppEngine –mejor tiempo de respuesta y tolerancia a fallos– y el nodo de inicio de sesión) [Are].

Existe una referencia en StackOverflow que recoge diferentes artículos/libros sobre AppScale (además de la documentación oficial), entre los cuales se puede destacar The AppScale Cloud Platform, donde describe la arquitectura y prestaciones de la plataforma o el libro/artículo de su creador AppScale: Open-Source PaaS [Are][Ama][Ado].

1.8.1. Instalación sobre Docker

Una de las instalaciones más simples es sobre Docker, para lo cual es necesario 4GB RAM. En este caso se ha utilizado una MV KVM similar a las utilizadas en otras PaaS (4 GB RAM, 30GB disco, un adaptador de red en NAT, la virtuali-

zación anidada activada *–host-passthrough–*, Ubuntu 14.04 más *git*, *virtualbox*, *vagrant*, *docker.io*, *openssh-server*, *firefox* y *Lubuntu*). Para ello, primero se deberá descargar la última versión del contenedor e iniciar su ejecución.

```
docker pull appscale/appscale:latest
docker run -i -t appscale/appscale:latest /bin/bash
```

Para desplegar una aplicación simplemente hay que cambiar de usuario/directorio, inicializar el entorno y desplegar la aplicación:

```
sudo -i
cd /root
bash appscale/scripts/fast-start.sh --no-demo-app
appscale deploy guestbook.tar.gz
```

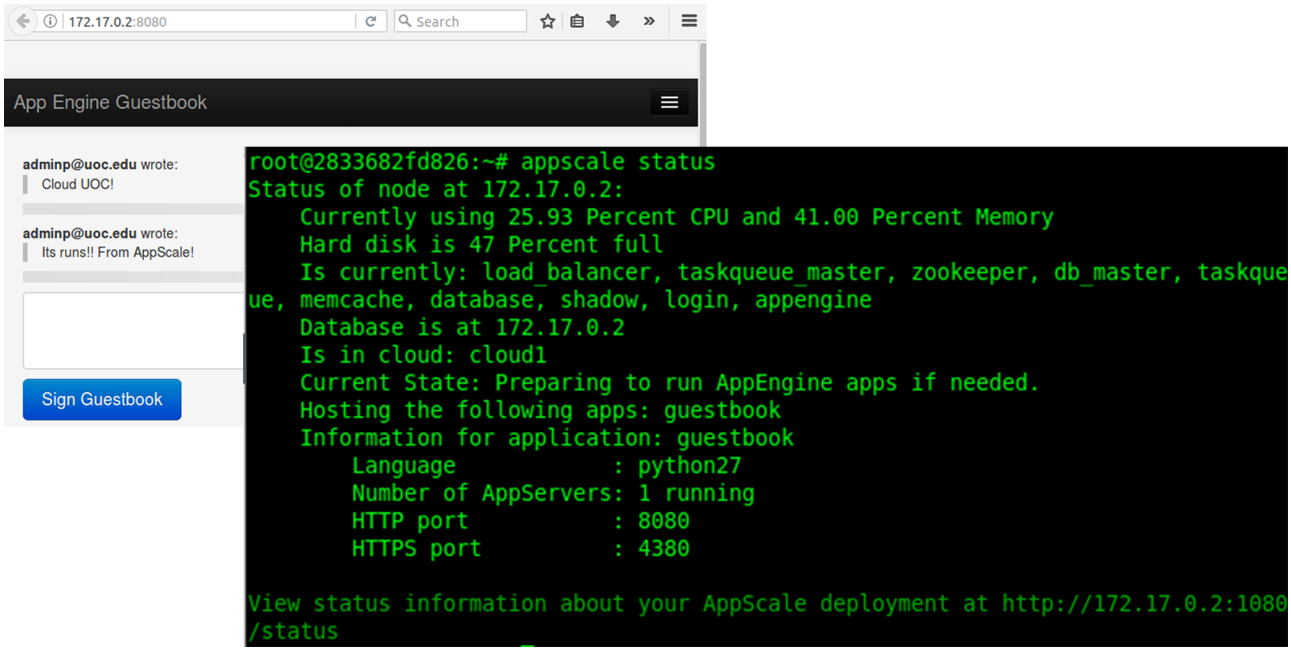
que mostrará la URL donde se ha desplegado la aplicación, pero que se puede reubicar con:

```
appscale relocate guestbook 80 443
```

Otros **comandos útiles** para gestionar AppScale son:

- Borrar una aplicación: `appscale remove <app>`
- Iniciar el entorno (desde el directorio donde se encuentra AppScalefile): `appscale up`
- Parar la plataforma: `appscale down`
- Estado: `appscale status`
- Conexión con el nodo máster: `appscale ssh`
- Generar *logs* sobre lo que ocurre en la plataforma: `appscale logs <directorio>`
- Actualizar la plataforma: `appscale upgrade`
- Borrar el estado y las aplicaciones: `appscale clean`
- Desplegar una aplicación: `appscale deploy <directorio de la aplicación>`
- Reubicación de una aplicación: `appscale relocate app-id http-port https-port`

Las imágenes siguientes muestran la ejecución de la aplicación de ejemplo (*guestbook*) sobre el contenedor y el estado de AppScale.



1.8.2. Instalación sobre Vagrant

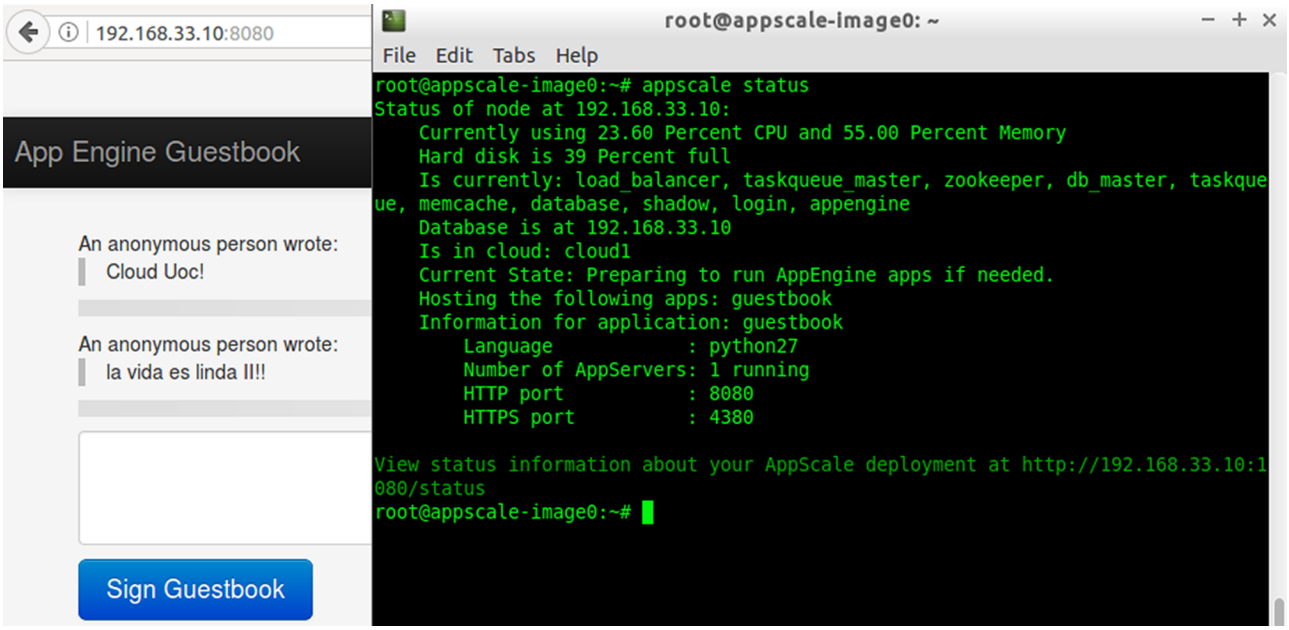
Para esta configuración es necesario disponer de Vagrant y Virtualbox; se utilizará la misma máquina que en el punto anterior, y primero se deberá descargar el *Vagrantfile*, luego aprovisionar la máquina y conectarse a ella como *root*.

```
curl -o Vagrantfile https://s3.amazonaws.com/appscale_CDN/files/Vagrantfile_template
vagrant up
vagrant ssh
sudo -i
```

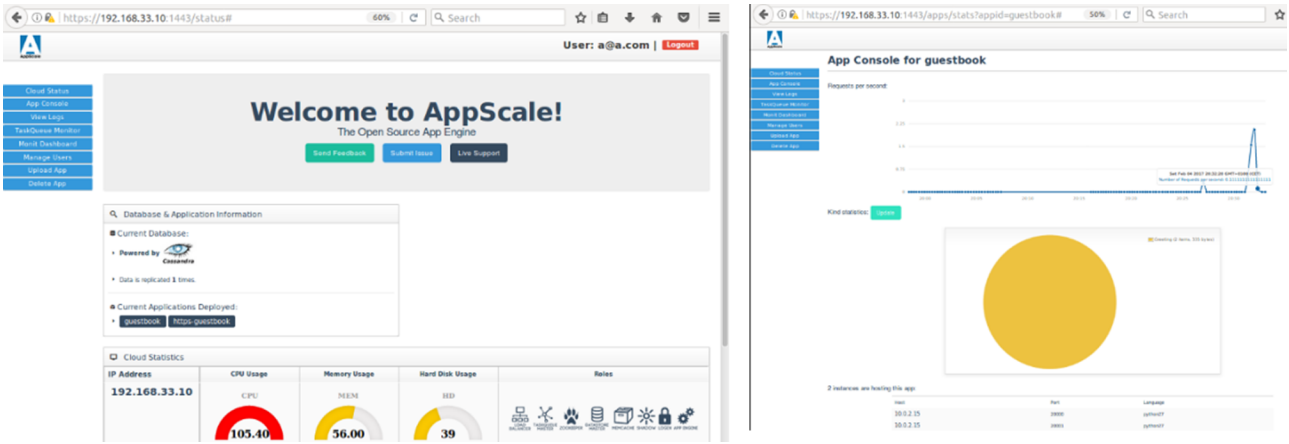
Para desplegar la aplicación es igual que el paso anterior: primero, inicializar el entorno Dev/Test y luego desplegarla (en esta primera prueba se desplegará la misma aplicación *guestbook* de ejemplo como en el caso anterior).

```
bash appscale/scripts/fast-start.sh --no-demo-app
appscale deploy guestbook.tar.gz
```

Después de desplegada se podrá conectar a la URL indicada (o reubicar los puertos para tenerlos donde se desee), como muestran las imágenes a continuación (similares a las anteriores, pero Vagrant crea una red en otro segmento).



Como se puede observar, al final de la ejecución de *appscale status* se indica el acceso al Dashboard de AppScale; conectándose a esta URL y accediendo con el **usuario/passwd** a@a.com/aaaaaa respectivamente (se pueden cambiar en el archivo AppScalefile indicando *admin_user: nuevo_us@whatever.com admin_pass: nuevo_password*), se tendrá acceso al entorno de control de AppScale, como se muestra en la figura siguiente, junto con la del estado de la aplicación desplegada (para más configuraciones consultar la página de FAQs).



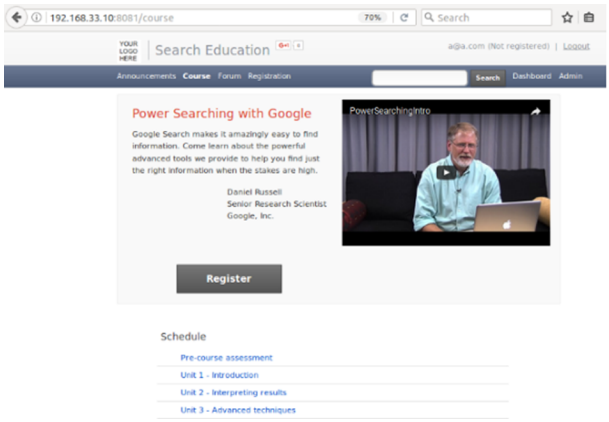
En la página antes indicada (FAQs) también se encontrará el usuario y *passwd* para acceder al TaskQueueMonitor (*appscale/appscale*; también se puede configurar en el archivo AppScalefile), que mostrará el estado de las tareas a través de la aplicación Celery Flower, que permitirá gestionarlas y obtener información.

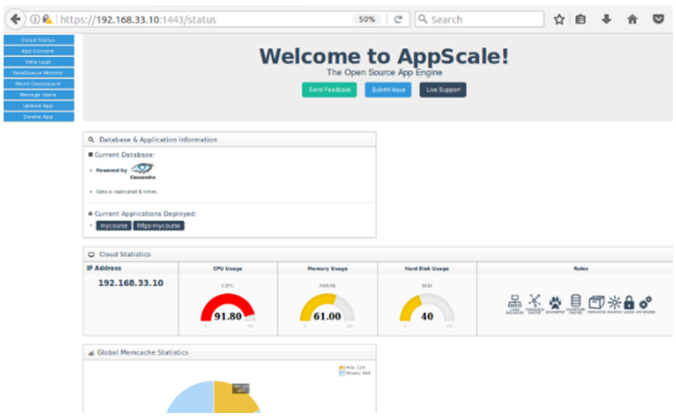
El despliegue de una aplicación es muy simple; por ejemplo, se mostrará el despliegue de CourseBuilder, que está dentro del repositorio de Appscale. Para ello, se clonará el sitio y luego se desplegará la aplicación.

```
git clone https://github.com/AppScale/coursebuilder.git course
```

```
appscale deploy ./course
```

Después de unos instantes se podrá acceder a la URL del curso y verificar su estado dentro de *Dashboard*, como muestran las figuras siguientes.





Como se ha visto, es una plataforma con una gran potencialidad que permite experimentar con GCP-AE, pero sin costo y en la infraestructura que se desee y con muy poco esfuerzo. Solo se debe tener en cuenta que en el despliegue/ejecución de aplicaciones grandes (como por ejemplo *coursebuilder*) en entornos locales (y sobre todo en Vagrant), consume mucha memoria; por ello, es necesario contar con máquinas que dispongan de ella y evitar muchas virtualizaciones (por ejemplo, en nuestro caso, la MV Vagrant se está ejecutando sobre una MV KVM como entorno de pruebas), ya que, sino, el entorno será totalmente funcional pero muy lento [Ado] [Ama] [Ard] [Are].

Nota

Es importante consultar la documentación indicada anteriormente para analizar y probar todas las funcionalidades que permite esta PaaS.

1.9. Azure Paas

Como ejemplo de plataforma como servicio (PaaS) públicas, Azure PaaS es un entorno de desarrollo en el *cloud* bajo el mismo *Dashboard* que para IaaS, con recursos que le permiten desplegar desde aplicaciones simples hasta sofisticados entornos empresariales en el *cloud*.

Una de las ventajas de Azure es que tanto la plataforma de desarrollo como la infraestructura se encuentran bajo el mismo proveedor, por lo cual se pueden utilizar planes de costo adecuados para el escalado de las aplicaciones, la simplificación de la integración y la compartición de la información y una serie de ventajas más que pueden encontrarse en falta cuando el proveedor del PaaS es diferentes que el IaaS.

Azure provee un *middleware* completo (más allá de lo que podría parecer dada su orientación a entornos Windows), herramientas de desarrollo, servicios de inteligencia empresarial (BI), sistemas de administración de bases de datos,

entre otros, que pueden utilizarse como plataforma independientemente del entorno de desarrollo del programador, si bien existe una plena integración con Visual Studio a través de un SDK específico.

La PaaS de Azure está diseñada para soportar el ciclo de vida completo de las aplicaciones web: construcción, pruebas, implementación, administración y actualización, y además, y sobre todo para sistemas Windows, provee una plataforma que evita el costo y la complejidad de comprar y administrar licencias de software, la infraestructura de aplicaciones y el *middleware* o las herramientas de desarrollo y otros recursos, así como el esfuerzo de administrar las aplicaciones y los servicios que son desplegados [Awa].

El propio proveedor sugiere como escenarios habituales para su PaaS:

a) Entorno de desarrollo: permite a los desarrolladores programar y personalizar aplicaciones basadas en el *cloud* utilizando una serie de componentes software incorporados, teniendo en unos pocos clics la escalabilidad, alta disponibilidad y capacidad de *multitenancy* de sus aplicaciones, y reduciendo el *time-to-market* y el esfuerzo necesario para mantener y administrar las aplicaciones.

b) Análisis o inteligencia de negocios (BI): las herramientas proporcionadas como un servicio sobre la PaaS permiten a las organizaciones analizar y buscar en sus datos masivos, encontrar patrones y la información importante que se encuentra en ellos y predecir resultados para mejorar la toma de decisiones en el diseño de productos, retornos de inversión u otras decisiones empresariales.

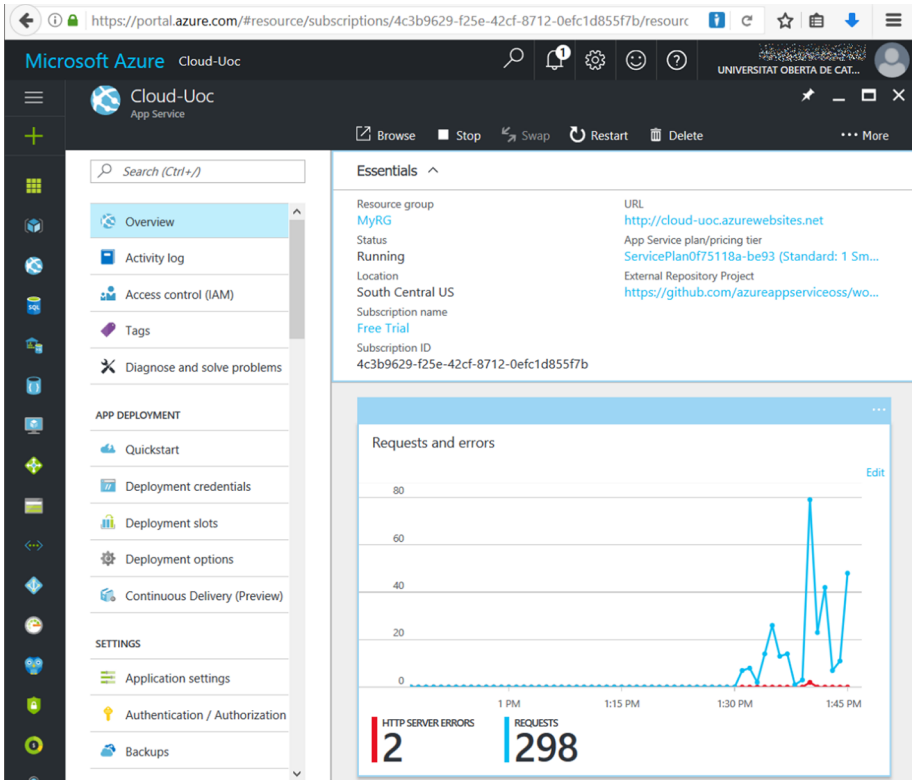
c) Servicios agregados: constituyen toda una serie de servicios de gran costo en IT, aunque sin ellos las aplicaciones pueden ver reducidas sus prestaciones/seguridad/escalabilidad (por ejemplo, análisis de flujo de trabajo, servicio de directorio, seguridad multinivel, *scheduling*, etc.).

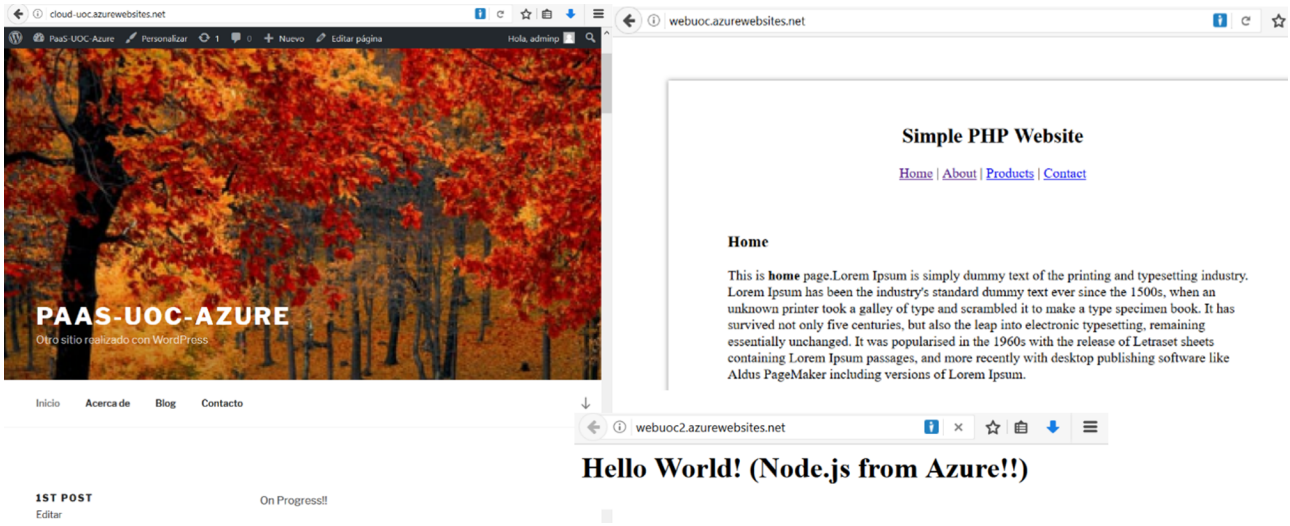
1.9.1. Desarrollo de aplicaciones en Azure

El desarrollador puede contar con algunos de los SDK (y ejemplos) que provee Azure para facilitar el trabajo con su PaaS; entre ellos: .NET, Python, Node.js, Java, Ruby, API REST, Azure CLI (para Windows, Linux/MacOS), PowerShell o entornos de desarrollo como *Visual Studio Tools for Azure*, *Docker Tools*, *PowerShell Tools for Visual Studio*, *Storage Explorer*, o *Python Tools*. Además, existe una gran cantidad de documentación y ejemplos de aplicaciones o cómo desplegarlas en un tiempo mínimo [Awa].

Como prueba de concepto, se ha desplegado e instalado una aplicación sitio WordPress desde la propia PaaS, y dos sitios de ejemplos basados en PHP y en Node.js a través de su sincronización con GitHub. Para ello, se han creado los ejemplos en Github y a continuación se ha creado el entorno en *Web + Mobile* > *Web App* y se configurado el repositorio externo previa autorización

en Github para acceder a él. Las imágenes siguientes muestran el *Dashboard* del despliegue de WordPress, y el resultado de los dos sitios creados posteriormente (una página web con PHP y otra con Node.js).





Como se puede ver, estas son solo unas simples aplicaciones de ejemplo para mostrar las facilidades que da la plataforma de despliegue rápido de aplicaciones web, pero sus posibilidades y prestaciones son muchas, incluido un apartado aún mayor para el despliegue de aplicaciones móviles; se debe profundizar en la documentación y en el tipo de entorno/servicios que es necesario para dar soporte a las aplicaciones que sea necesario desplegar [Awa].

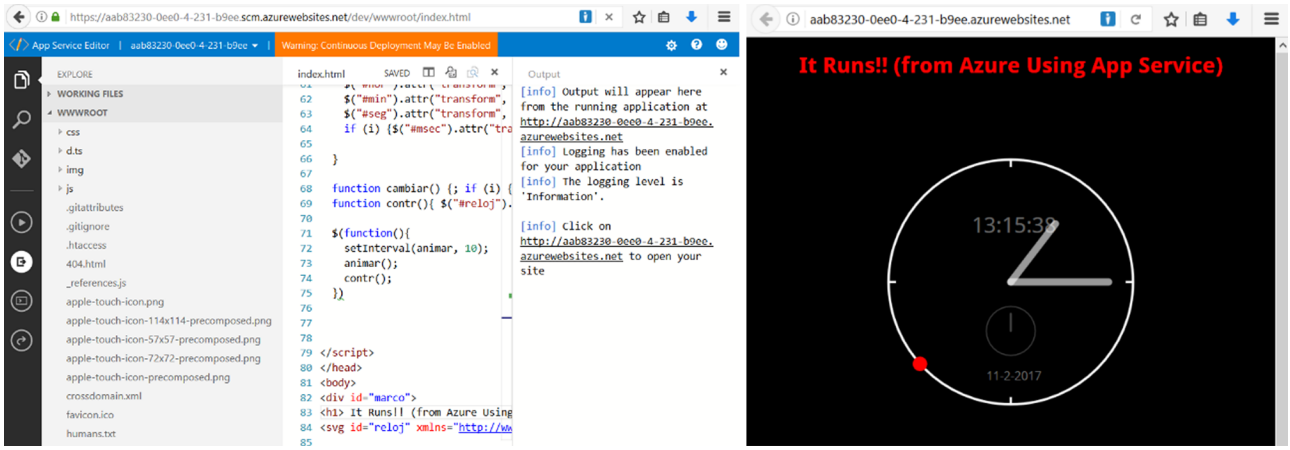
1.9.2. Azure App Service (Free)

Existe una gran cantidad de servicios PaaS que pueden ser utilizados desde Azure de forma gratuita (incluso después de que se haya terminado la subscripción gratuita), aunque estas aplicaciones estarán disponibles por un tiempo limitado (generalmente una hora, aunque pueden extenderse por más tiempo).

Entre los servicios PaaS que se pueden utilizar se encuentran:

- *App Service* (construir y crear hasta 10 *apps* web/móvil)
- *Notification Hubs* (hasta 1M de notificaciones *push*/mes)
- *Mobile Engagement*
- *Search*
- *Machine Learning*
- *Data Factory/Catalog*
- *IoT Hub*
- *Virtual Network*
- *Active Directory*
- *Visual Studio Team Services*
- *Application Insights*
- *Scheduler*
- *Automation*
- *Log Analytics*

Las imágenes a continuación muestran un *web site* desplegado por App Service (con una subscripción gratuita de una hora ampliada a 24) desde el sitio de desarrollo (similar al portal de Azure, pero centrado en la aplicación) y el funcionamiento sobre el PaaS.



1.10. Mendix

Dentro de las PaaS públicas, Mendix ha irrumpido en el escenario de las plataformas con gran fuerza, tal y como registra el informe de Gartner de 2016 *Magic Quadrant for Enterprise Application Platform as a Service, Worldwide* (se encuentra en el grupo de RedHat, IBM, Sap, Oracle, OutSystems en el cuadrante de «visionarios» y a la misma altura que Azure en el cuadrante de «líderes»).

> Esta plataforma permite a los usuarios crear, integrar e implementar aplicaciones web y móviles que se pueden construir desde el principio o como una capa encima de los sistemas existentes y que cuenta con una gran cantidad de usuarios de renombre.

Desde el punto de vista empresarial (y después de grandes cambios), la empresa del mismo nombre ha incluido una categoría de socios empresariales y tecnológicos entre los que se encuentra Pivotal. Esta empresa (*joint venture* de EMC, VMware y General Electric) es uno de los principales contribuidores de Cloud Foundry, una plataforma PaaS *open-source*, desarrollada por VMware, cedida a Pivotal y desde 2015 gestionada por laCloud Foundry Foundation. Se entiende que esta anexión, como socio, forma parte de la estrategia de Mendix en torno a Cloud Foundry y a sus intenciones de desarrollar el estándar de código abierto, así como integrar el entorno PaaS de Cloud Foundry con su ***app Platform-as-a-Service*** (aPaaS).

Una de las principales aportaciones de Mendix es la utilización de un enfoque de desarrollo de software visual y basado en modelos, conocido como *Model-driven software Development* (MDSD), altamente innovadores y que han sido reconocidos como tales por expertos en este ámbito (*Forrester Wave: Enterprise Public Cloud Platforms for Rapid Developers* y Gartner).

Esta *rapid* ***apps PaaS*** (aPaaS) centra su principal activo en el diseño y desarrollo de software bajo modelos MDSD visuales, que son altamente atractivos para los desarrolladores, ya que permiten aplicar abstracciones de gran nivel y el principio de «sin código» para transformar rápidamente ideas en aplicaciones. Además, aportan rapidez y agilidad en el diseño y reducen el *time-to-market* con una participación más amplia entre todos los miembros (incluso los no expertos en código) del equipo de desarrollo y, por lo tanto, «sociabilizan» el desarrollo con una mayor implicación de diferentes grupos y mayor productividad.

En el ámbito de desarrollo (sobre todo para dispositivos móviles), Mendix ofrece un gran conjunto de recursos (conectores, *widgets*, módulos y servicios) para acelerar la programación que permiten, mediante su inclusión, la conexión

con otras aplicaciones (BD, IoT, ML), el acceso a las funciones de entrada/salida, autentificación y mensajes. Existe un gran conjunto de guías para utilizar estos recursos y crear aplicaciones.

En resumen, la aPaaS Medix ofrece una plataforma de estas características:

- transforma rápidamente ideas en aplicaciones,
- está diseñada tanto para la empresa como para grupos de IT,
- permite aplicar metodologías ágiles de desarrollo para gestionar el ciclo de desarrollo y progreso,
- se basa en el desarrollo visual multiplicando hasta 6x la productividad,
- facilita el desarrollo de las aplicaciones (solo unos cuantos clics) y despliegue,
- es posible la operación eficiente a través de un punto central de gestión de las aplicaciones [Mdd].

1.10.1. Desarrollo de una aplicación web/móvil de pruebas

Mendix provee una cuenta de prueba (gratuita) que permite construir cualquier aplicación multidispositivo sin limitaciones, de hasta diez usuarios por aplicación, con el soporte de la comunidad y limitado a un entorno.

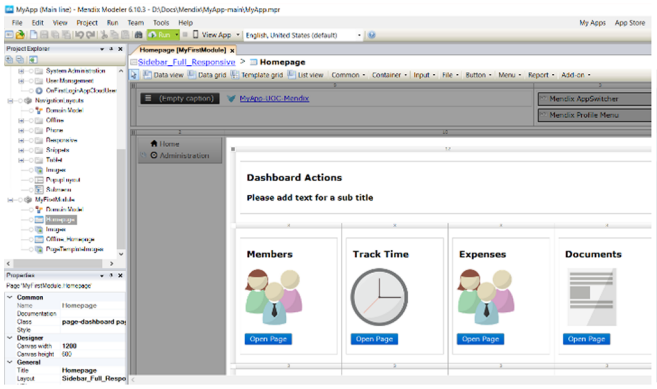
En la documentación proveen diferentes guías para comenzar; en este caso, se han seguido la de *Build a Simple HRM App*, y la de *Deploy Your First Hybrid Mobile App* junto con el Mendix Modeler disponible en la App Store de la plataforma. En la aplicación Mendix Modeler se puede crear/modificar/depurar/probar la aplicación; por ello se llama «modelador».

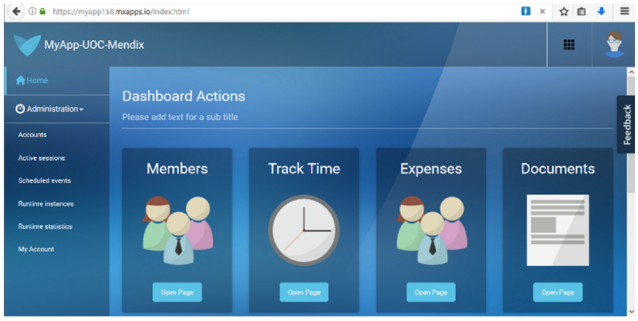
Un proyecto consiste en muchos documentos que se agrupan en módulos y carpetas con configuraciones, módulos (que serán los que darán la funcionalidad), modelos de dominio, páginas y microflujos (expresan la lógica de la aplicación). Todos ellos serán agrupados bajo un nombre de proyecto y el desarrollador podrá trabajar con diferentes proyectos simultáneamente, si es necesario manteniendo la independencia uno de otro, pero también el Modeler puede trabajar con dos instancias del proyecto al mismo tiempo.

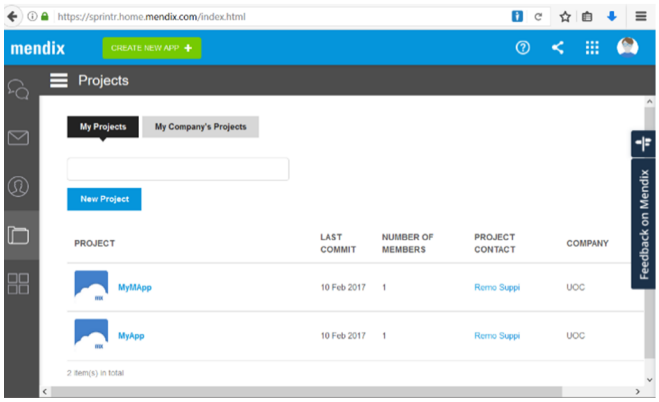
El Modeler permite ejecutar tanto la aplicación en local con los recursos de la máquina, o subirla a la aPaaS y ejecutarla desde allí; también para las aplicaciones móviles, pueden ser probadas en el entorno local como si fueran la simulación de un dispositivo o generando un código QR para que, a través de una *app* de Mendix previamente instalada sobre el dispositivo, se pueda hacer la descarga desde los servidores de Mendix y ejecutarla en el dispositivo físico.

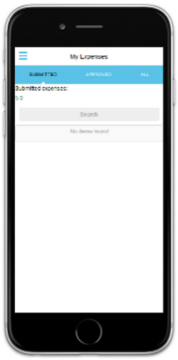
Las imágenes a continuación muestran una vista parcial del Modeler con una aplicación de recursos humanos de prueba que se ejecutan sobre la aPaaS (desarrollada y desplegada con el programa anterior), el entorno de gestión de la aPaaS y una aplicación móvil de ejemplo ejecutada sobre el simulador sobre la aPaaS.

Como conclusión es una plataforma interesante, complementada con un entorno de desarrollo que permite crear/modificar/depurar/desplegar rápidamente aplicaciones, y si bien el Modeler solo existe para entornos Windows, es posible configurar un *runtime* para ejecutar las aplicaciones sobre un servidor Mendix instalado sobre una máquina Linux (consultar los requerimientos) ampliando el potencial de la plataforma, ya que no necesariamente se debe trabajar durante el desarrollo con la aPaaS, sino que se pueden configurar entornos locales y luego llevarla al *cloud*.









1.11. Google App Engine

Google App Engine es una plataforma para crear aplicaciones web escalables y *backends* para *apps* móviles que proporciona servicios integrados y una API, así como BD NoSQL, *memcache* y una API de autenticación de usuario.

App Engine escalará la aplicación automáticamente en respuesta a la cantidad de tráfico que recibe y el usuario pagará solo por los recursos que utiliza (especificados en una tabla de precios donde todo lo que se puede consumir está especificado). Desde el punto de vista del usuario, no hay servidores que deba desplegar/monitorizar o mantener, y todo es transparente [Gae].

Para hacer una prueba de App Engine, se trabajará como lo haría un desarrollador en un entorno local, para luego desplegar la aplicación en el *cloud*. Primero, debemos tener un proyecto activo en la GCP e instalar el Google Cloud SDK.

Para ello, siguiendo la guía de instalación:

```
export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)"
echo "deb https://packages.cloud.google.com/apt $CLOUD_SDK_REPO main" | \
   sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
sudo apt-get update && sudo apt-get install google-cloud-sdk \
   google-cloud-sdk-app-engine-python
gcloud init
```

Además, es necesario instalar `php` (en este caso se ha utilizado un Ubuntu 16.04, por lo cual es `php7.0`) y también `php7.0-cgi` y `php7.0-bcmath` y `apache2-utils` si se desea utilizar el comando `ab` para hacer pruebas de rendimiento sobre el sitio desplegado.

Como prueba de aplicación, se utilizará una de ejemplo provista por Google en Github:

```
git clone -b phase0-helloworld \
  https://github.com/GoogleCloudPlatform/appengine-php-guestbook.git \
  helloworld
cd helloworld
```

Se puede probar la aplicación en local utilizando un *development server* (dev_appserver.py) incluido en el SDK, por lo cual desde el directorio de la aplicación ejecutar:

```
dev_appserver.py app.yaml --php_executable_path /usr/bin/php-cgi
```

Se podrá ver el resultado poniendo en un navegador *http://localhost:8080/* y se podrán hacer cambios y recargar la página para actualizarlos, ya que el *development server* mira si hay cambios y los recarga si es necesario. Puede pasar que el *dev_appserver* no esté en el PATH y se deberá anteponer al comando (algo como */usr/lib/google-cloud-sdk/bin/dev_appserver.py*).

Una vez terminada la aplicación se debe desplegar sobre App Engine, por lo cual se deberá ejecutar desde el directorio raíz de la aplicación:

```
gcloud app deploy
```

Este comando mostrará la URL donde se ha desplegado (dominio por defecto appspot.com), pero también se puede visualizar ejecutando:

```
gcloud app browse
```

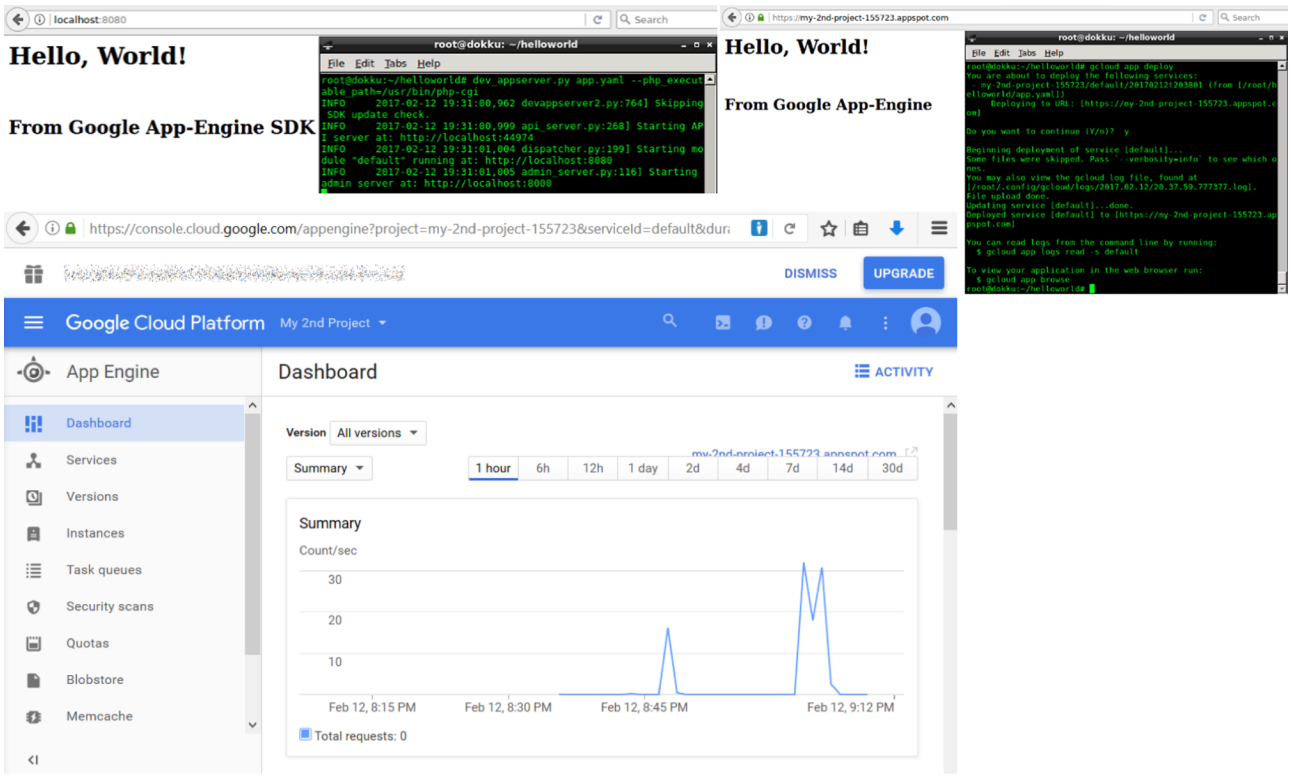
El archivo para la configuración de App Engine es *app.yaml* (en sintaxis YAML) que tiene un contenido como:

```
runtime: php55
api_version: 1
handlers:
- url: /.*
  script: helloworld.php
```

Donde indica que se debe ejecutar el *runtime php55* y la *API_version 1*, que la URL que satisfaga la expresión /.* (todas las URL), y que será manejada por el *scripthelloworld.php*.

Para eliminar la *app* se deberá eliminar el proyecto, pero se debe ir con cuidado ya que su eliminación también lo hará con cualquier otro despliegue que se haya realizado dentro de él, incluidas sus URL. Consultar la documentación para mantener el ID del proyecto y las URL.

Las figuras a continuación muestran el despliegue local, el despliegue en App Engine y el *dashboard*. Para generar peticiones y analizar el tiempo de respuesta y las prestaciones, se ha utilizado el comando `ab` (del paquete *apache2-utils*) con una sintaxis como `ab -n 1000 -c 100 <URL>`



1.12. Cloud IDE as a Service

Con el auge de las plataformas PaaS y especialmente adaptadas a desarrolladores como IDE (*Integrated Development Environment*), ha surgido un conjunto de plataformas que pueden ser utilizadas tanto como IaaS (como MV simplemente y sin posibilidad de configurar otras partes de la infraestructura) como PaaS, ya que disponen de un conjunto de *stacks* y pueden proveer los servicios típicos de una plataforma de estas características, o bien como una SaaS donde el servicio es programación y desarrollo/despliegue.

Existen gran cantidad de ellas (ya han sido mencionadas en los capítulos de introducción e IaaS), pero entre las renombradas por la comunidad están: Cloud9, Codebox, Codiad, Codeanywhere, Coding Ground, Collide, Codenvy, Eclipse Che, Koding, Ideone, Pyhton Fiddle, Orion.

Algunas son muy específicas, como por ejemplo Python Fiddle que, como su nombre indica, está orientada solamente a Phyton; otras se acercan más a una SaaS, ya que es un compilador como servicio (por ejemplo, IdeOne) o IDE como servicio (por ejemplo, Codiad), y no todas tienen cuentas gratuitas, o si la tienen necesitan una tarjeta de crédito para acceder a pesar de que funcionen con suscripción gratuita (por ejemplo, Cloud9, Koding). Muchas de ellas son *open-source* y disponen del repositorio (pueden ser la misma versión del PaaS o no) en Github; asimismo, la plataforma se puede instalar en servidores locales o IaaS (por ejemplo, Codenvy, Koding, Codebox, Codiad, Collide, Orion, Eclipse Che, esta utilizada por Codenvy), además de la plataforma PaaS (por ejemplo, Codenvy, Codeanywhere). Como Codenvy, Codeanywhe-

re como plataformas fueron tratadas en el capítulo de IaaS, en este apartado nos dedicaremos a Eclipse Che como ejemplo de PaaS sobre servidores locales, aunque también se puede poner sobre un IaaS.

1.12.1. Eclipse Che

Eclipse Che es una plataforma que proporciona espacios de trabajo (WS: *workspaces*) que incluyen *runtimes* e IDE donde el servidor del WS dispone de una API RESTful y todo se basa en un navegador. Además, incluye soporte para *plugins* (idiomas, *frameworks* y herramientas) y un SDK para crearlos y ensamblarlos.

Esta plataforma puede ser instalada localmente, a través de una cuenta SaaS (alojada en Codenvy y gratuita) o en una instancia de una IaaS, o también integrarse con otras PaaS como, por ejemplo, OpenShift.

Che define el WS como los archivos de código del proyecto y todas sus dependencias necesarias para editar, construir, ejecutar y depurarlos donde el IDE y el *runtime* de desarrollo son tratados como una dependencia del WS. Los WS están aislados entre sí y cada uno es responsable de gestionar el ciclo de vida de los componentes que están contenidos en él. El *runtime* predeterminado dentro de los WS son contenedores Docker; los usuarios pueden definir el contenido de su *runtime* mediante Dockerfiles y sin utilizar la sintaxis de Docker.

Un espacio de trabajo puede tener múltiples proyectos vinculados a repositorios de control de versiones remotos como *git*, *subversion* o *mercurial*, que se montan en el WS y están disponibles en el almacenamiento a largo plazo. Cada proyecto tiene un «tipo» (por ejemplo, *maven*) que, cuando se selecciona, activará una serie de complementos que modifican el comportamiento del WS. Además, cada WS tiene su propio navegador privado alojado en él (empaquetado como JavaScript y CSS) y su propio servidor SSH permite que los clientes remotos y los IDE se monten en el WS.

Dado que los espacios de trabajo son servidores que tienen sus propios *runtimes*, son colaborativos y compartibles y permite que varios usuarios puedan acceder al mismo espacio de trabajo al mismo tiempo a través de una URL propia. Tanto el servidor Che como cada espacio de trabajo tienen sus propias API RESTful incorporadas y todo los que hace el usuario en el *dashboard* o en el IDE se realiza a través de esta.

Che es un servidor de WS que se ejecuta en la parte superior de un servidor de aplicaciones (Tomcat) que carga, como aplicación web, el IDE y proporciona componentes de interfaz de usuario (asistentes, paneles, editores, menús, barras de herramientas y cuadros de diálogo) y el IDE se comunica con Che sobre RESTful APIs que administran e interactúan con un WS máster. En la

documentación puede encontrarse un mayor detalle de la arquitectura y su configuración/funcionalidades y prestaciones, y también guías de uso de diferentes funcionalidades, *plugins* y *stacks* (por ejemplo, múltiples WS, java/node.js/PHP/Wordpress/Android/..., etc.) [Ecd].

Para su instalación se ha utilizado una MV KVM (4 GB RAM, 30GB disco, un adaptador de red en NAT, la virtualización anidada activada *–host-passthrough–*, Ubuntu 14.04 + *openssh-server*, *git*, *Firefox*, *Lubuntu*); en ella, se ha instalado Docker 1.12.3 desde el repositorio de paquetes del desarrollador (la versión del repositorio de Ubuntu no es la adecuada ya que necesita un 11.1+).

El repositorio de Che se encuentra en Github, pero la instalación es sumamente sencilla a través de Docker:

```
docker run -it eclipse/che start
```

O si se desea un directorio local compartido para salvar datos del usuario, será:

```
docker run -it --rm -v /var/run/docker.sock:/var/run/docker.sock \
   -v <path>:/data eclipse/che start
```

Luego de instalado, se podrá acceder a él a través de *http://< ip>:8080 y a la API http://< ip>:8080/swagger*

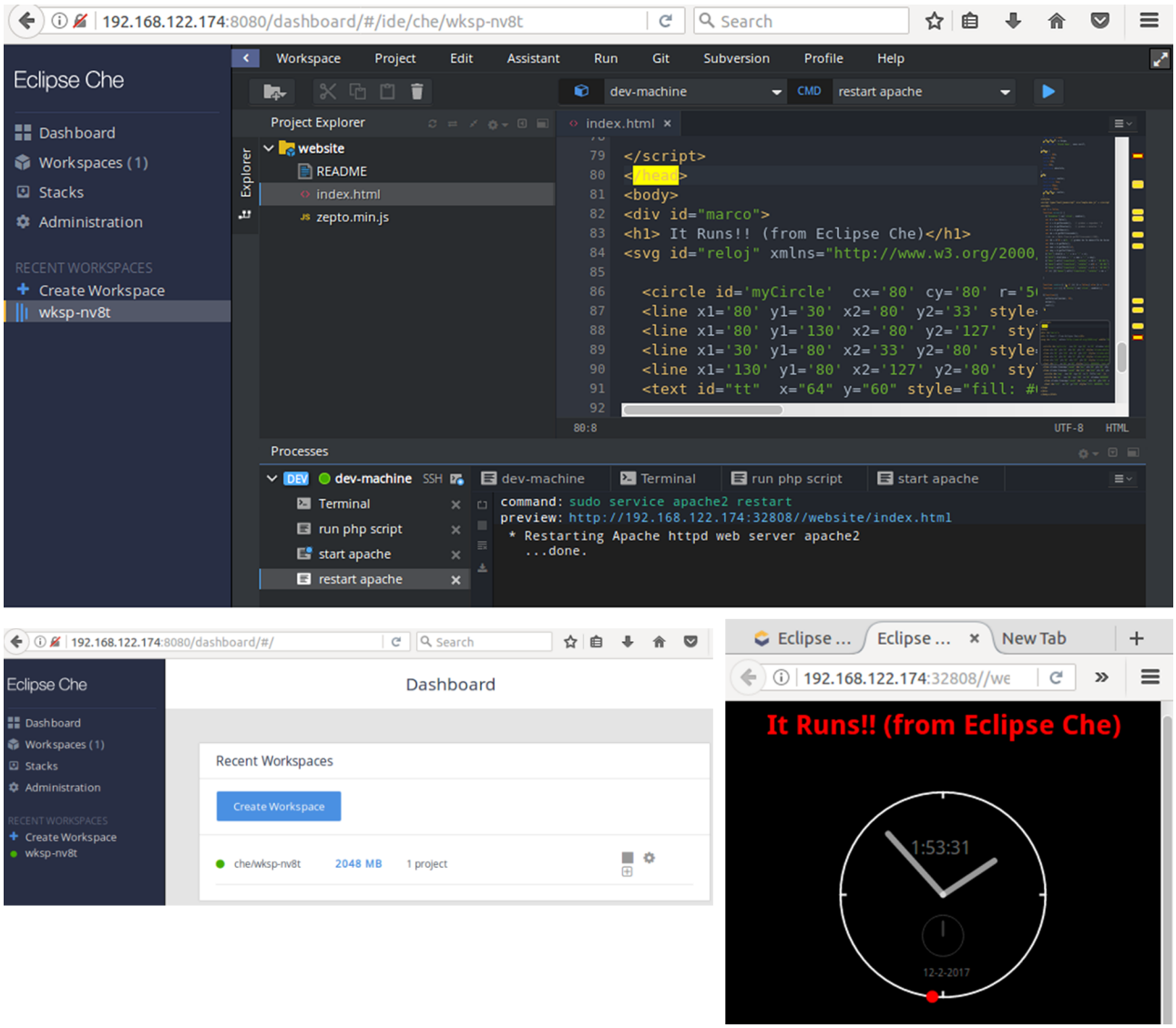
Comandos útiles:

- Pararlo: `docker run <DOCKER_OPTIONS> eclipse/che stop`
- Reiniciarlo: `docker run <DOCKER_OPTIONS> eclipse/che restart`
- Ayuda: `docker run eclipse/che`
- Para acceder desde otras máquinas, agregar la IP: `docker run <DOCKER_OPTIONS> \ -e CHE_HOST=<your-ip> eclipse/che start`

Se debe tener en cuenta que Che necesita 300 MB disco y 256MB RAM para sus servicios y las necesidades de los WS se agregarán a estas. Las imágenes de Docker consumen ~300MB de disco y dependerá del *stack*, pero aproximadamente unos 500MB RAM y los WS unos 250MB adicionales (dependiendo de la complexidad).

Como prueba de concepto se ha creado un WS de tipo PHP, se ha cargado el contenido de una página web y una librería JavaScript desde la máquina local y se ha desplegado a través de las opciones del IDE.

Las figuras siguientes muestran el entorno IDE y el *dashboard* del proyecto, así como la página ejecutándose en el servidor dentro del *runtime*. Como entorno más complejo, es interesante experimentar con un WS con múltiples *runtimes* que se comuniquen e intercambien datos en Java y MySQL, como se describe en la guía de ejemplos.



Tanto la opción Che en local como la plataforma PaaS/SaaS en Codenvy presentan una gran cantidad de funcionalidad, no tratada en este simple ejemplo, que debe ser analizada con detalle, así como la gran cantidad de opciones, *stacks* y posibilidades de ampliación que dispone. Para mayor detalle consultar la documentación y guías disponibles [Ecd].

1.12.2. Cloud9 SDK

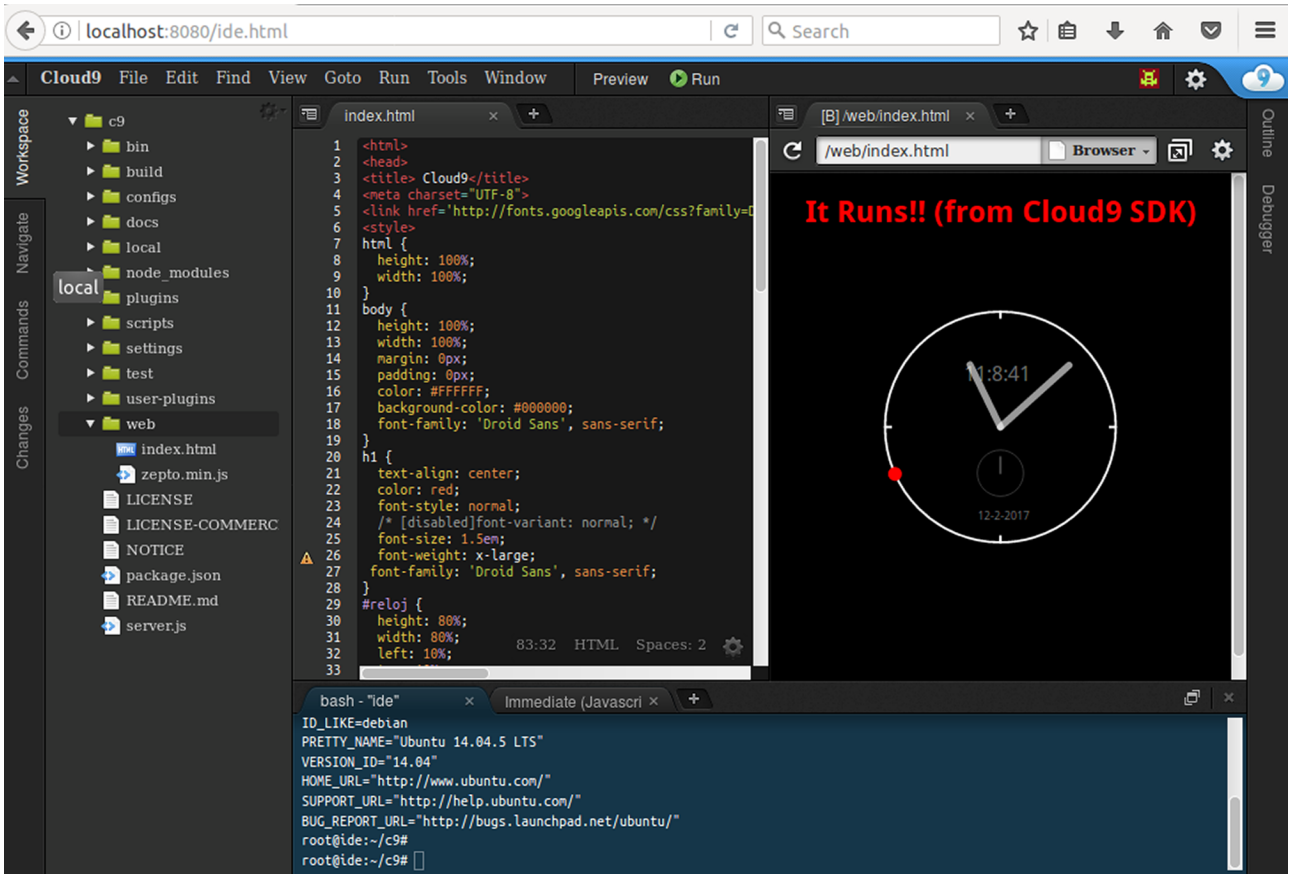
C9 es una plataforma *cloud* IDE con una filosofía basada también en *workspaces*, pero con una funcionalidad/prestaciones significativamente diferentes de las otras, aunque también utiliza Docker como *runtime* y está ubicada en GCE (en 2016, la compañía ha sido comprada por Amazon para formar parte de AWS).

La plataforma es totalmente extensible mediante *plugins* y puede instalarse localmente para su desarrollo como SDK desde su repositorio en GitHub. El acceso a la PaaS se puede hacer con una cuenta gratuita (si bien es necesario introducir una tarjeta de crédito) que anuncian que es perpetua y permite WS públicos, uno privado, con soporte de la comunidad y cuenta con un extenso conjunto de características y usuarios de renombre, así como una versión en Salesforce adaptada para el entorno y características específicas de este (Apex, Visualforce, Lightning) [C9d].

Su instalación en local es muy simple; para esta prueba se ha utilizado una máquina similar a la utilizada en Eclipse Che con algún software adicional, ya que la plataforma está escrita en JavaScript y utiliza Node.js como entorno de ejecución. Los pasos han sido:

```
apt-get install build-essential
apt-get install nodejs
git clone git://github.com/c9/core.git c9
cd c9
scripts/install-sdk.sh
nodejs server.js -p 8080 -a
```

Y se ha accedido a la URL *http://localhost:8080/ide.html*. Es importante destacar que es un SDK que permite desplegar una versión de Cloud9 para desarrollar los *plugins* localmente y crear un IDE personalizado basado en Cloud9, pero inicialmente no dispone de toda la infraestructura como la PaaS en línea. La imagen siguiente muestra su ejecución y el desarrollo de un sitio web y su previsualización.



1.13. Otras plataformas

Existen otras plataformas *open-source* no menos interesantes que también tienen un lugar en el escenario de las PaaS y que, por cuestión de recursos necesarios para su despliegue/complejidad, no se han tratado como ejemplos, si bien deben ser tenidas en cuenta; entre otras, se pueden mencionar:

- **Deis**: requiere una mención especial ya es una plataforma *open-souce* de renombre en el escenario de las PaaS; permite de forma fácil desplegar y manejar aplicaciones sobre un conjunto de servidores y está construida sobre la base de Docker y CoreOS, con el fin de proveer una plataforma «liviana» con un *workflow* inspirado en Heroku. La versión actual de la PaaS es Workflow Deis (v2), que presenta cambios significativos en relación con la versión anterior Deis Paas (v1); no obstante, ambas instalaciones aún en pruebas locales necesitan gran cantidad de recursos (RAM sobre todo). Esta plataforma se basa en la metodología Twelve-Factor App y se sustenta sobre Kubernetes, que es un gestor de clúster de código abierto desarrollado por Google y donado a la Cloud Native Compute Foundation. Kubernetes gestiona toda la actividad en su clúster, incluye convergencia al estado deseado, descubrimiento de servicio, monitorización del estado y resolución de DNS. Workflow se integra plenamente en este y permite construir contenedores directamente desde el código fuente de la aplicación, agregar registros, administrar las configuraciones de implementación y gestionar

las versiones de aplicaciones, entre otras características. Esta integración se realiza a través del paquete Helm [Dwd].

- **Cloud Foundry**: desarrollada inicialmente por VMware, cedida a Pivotal Software y desde 2015 bajo la gestión de Cloud Foundry Foundation como *open-source* y escrita mayoritariamente en Ruby, Java y Go. CF, al igual que otras PaaS, soporta todo el ciclo de vida completo de las aplicaciones como desarrollo/pruebas/publicación y permite implementar estrategias de entrega continua. Las tres partes esenciales de CF son: BOSH (*Bosh Outer Shell*; crea e implementa MV sobre la infraestructura física y despliega/ejecuta CF en la parte superior), *Cloud Controller* (ejecuta las aplicaciones y otros procesos en las MV, y equilibra la demanda y administra los ciclos de vida de las aplicaciones), *router* (encamina el tráfico entrante hacia las MV y trabaja conjuntamente con el balanceador de carga especificado).
  CF utiliza dos tipos de MV, las que constituyen la infraestructura de la plataforma y las máquinas virtuales *host* que alojan aplicaciones para el mundo exterior. Dentro de CF, el sistema «Diego» distribuye la carga de la aplicación alojada sobre todas las máquinas virtuales *host*, y lo mantiene en funcionamiento y equilibrado a través de sobrecargas de demanda, interrupciones u otros cambios. Para organizar el acceso de los usuarios y controlar el uso de los recursos, se definen *Orgs* y *Espacios* con roles predeterminados (administrador, desarrollador o auditor) que son gestionados mediante la autenticación y autorización de usuario (UAA) [Cfd].

Igualmente existen muchas otras opciones como PaaS propietarias y con un objetivo/escenario de *app* específico, como pueden ser AWS Elastic Beanstalk y AWS Lambda, CloudBees, EngineYard, Force.com, HPE Helion Stackato, IBMSmartCloud, LongJump, Morpheus, plataform-sh, Zoho Creator entre otras.

2. SaaS (*software as a service*)

Las tendencias actuales de «democratizar» el acceso a las tecnologías y el gran empuje de las aplicaciones (sobre todo en dispositivos móviles) han sido el factor de mayor importancia en el crecimiento de las SaaS.

Es importante decir que el avance de la tecnología (servicios/servidores, navegadores, programación, API, etc.) ha permitido lograr acercar un entorno tecnológico de procesamiento de la información desde cualquier dispositivo a usuarios no tecnológicos; esto ha ampliado las posibilidades de negocio, los ámbitos de nuevos mercados y la fidelización de clientes que, en otro momento, hubieran descartado, por ejemplo, ejecutar un aplicativo de retoque de imágenes o edición de texto en forma remota. Es decir, el Software como servicio (SaaS) hoy en día es una realidad y utilizado por una gran cantidad de usuarios (y muchos de ellos sin saber qué es un SaaS).

En la actualidad hay una estrecha colaboración entre una *app* en un dispositivo móvil y el *backend* del servicio en el proveedor (redes sociales, correo, agenda, almacenamiento, etc.), así como una gran cantidad de servicios empresariales que ya se ofrecen en formato SaaS y no en infraestructuras propias de la empresa (correo, desarrollo, organización, almacenamiento, colaboración, etc.). Esto permite que todo tipo de empresas y profesionales puedan tener acceso a servicios sin grandes inversiones en infraestructura o personal IT y así disponer de aplicaciones seguras, ubicuas, profesionales y adecuadas para dar soporte a las tareas de las empresas/profesionales actuales.

Los modelos de monetización de las aplicaciones SaaS son diversos, pero casi con total seguridad se pueden encontrar aquellos servicios que cubran las necesidades por el precio adecuado teniendo en cuenta que, con un SaaS y como se planteó en el primer capítulo, no es necesario ninguna instalación en la máquina/dispositivo del cliente (generalmente un navegador o una *app* a medida pero que actúa solo de marco contenedor); tampoco es necesario hacer copias de seguridad, ni grandes configuraciones, ni estar preocupado por el servicio/seguridad (obviamente teniendo la SLA adecuada) y las licencias, entre otras ventajas.

No obstante, se deberá tener en cuenta que los datos y el acceso a ellos no dependen de la voluntad del usuario, sino de la disponibilidad del servicio y la conectividad a la red. Esto trae aparejado que el cliente debe analizar con

detenimiento y buscar la SLA adecuada para no poner en riesgo sus datos y la accesibilidad a ellos o, en caso contrario, tener los mecanismos adecuados para poder ser indemnizado de forma justa.

Hoy en día, existen aplicaciones que no pueden ser concebidas bajo otro modelo que un SaaS (por ejemplo, las redes sociales), pero cada vez más ante las opciones propias con capital inmovilizado (previsto para el peor caso), con soporte local (que si debe ser 24/7 es sumamente oneroso), con licencias y personal experto salga más a cuenta desde distintos puntos de vista (económico, calidad de servicio, operatividad, soporte, etc.) la opción del SaaS escalable, la cual siempre deberá tener, obviamente, el respaldo de una SLA adecuada a las necesidades del cliente.

Las reticencias de algunas empresas (ya sea por la privacidad de sus datos o la prestación del servicio) han ido quedando superadas por un mayor conocimiento del modelo SaaS y los avances tecnológicos que han implementado los proveedores de servicio (por ejemplo, encriptación de los datos) y son muy pocas las empresas que, una vez adoptado un SaaS, vuelven al modelo tradicional. Otro aspecto interesante, desde el punto de vista empresarial, es el caso de las licencias de software que, en el modelo propio, implica grandes inversiones en licencias (en muchos casos perpetuas) que en la mayoría de las veces se utilizan solo por períodos cortos de tiempo, mientras que en el SaaS, y dependiendo del modelo de pago, generalmente no se deben hacer grandes inversiones en estas y solo se realizará por el período de uso; ello genera reducción de costos y mejora de la inversión.

Las principales ventajas que se pueden apreciar del SaaS por los clientes empresariales son:

- la reducción de personal especializado y de soporte 24/7,
- modelos de pago/cuota mensual/anual/sin pago por licencia,
- acceso ubicuo y gestión dinámica de servicios/usuarios/roles/... y
- no disponer de salas específicas de servidores, obra civil, refrigeración/electricidad y mantenimiento.

Entre las preocupaciones está el temor al *data-lock-in* o *provider-lock-in*, la inseguridad de si se dispone de la SLA adecuada que pueda cubrir adecuadamente los costos e indemnizaciones cuando sea necesario recurrir a ella o la accesibilidad.

Es evidente que algunas de estas desventajas pueden mitigarse, pero otras están más allá de las decisiones/acciones que pueda asumir el cliente, por lo cual será necesario analizar con cuidado todas las situaciones y posibilidades de forma objetiva para hacer un balance y tomar las decisiones adecuadas en la

migración hacia el SaaS. Es interesante el informe realizado por Deloitte a expertos donde se manifiesta que (en 2014) el 58 % de las empresas encuestadas ya utilizan un SaaS y el 67 % recomienda su implantación [Del].

3. SaaS: casos de uso

Este apartado se centrará en aplicaciones que permitan desplegar un servicio SaaS y no en la utilización de SaaS públicos o privados que están a disposición de los usuarios. No obstante, más allá de las recomendaciones de probar las plataformas habituales de SaaS para usuarios individuales, algunas ya sugeridas en el primer capítulo y que se muestran en la tabla siguiente (solo alguno de ellos con cuentas gratuitas individuales), se recomienda consultar los servicios empresariales de los proveedores en diferentes categorías (por ejemplo, en ofimática, Office365 para empresas, Gsuite para empresas) para analizar los diferentes modelos de negocio, rentabilidad, inversión y otros aspectos de interés para el cliente empresarial ya que, en muchos casos, distan (en forma y prestaciones) de los mismos servicios ofrecidos gratuitamente o con modelos de negocio no basados en un valor monetario.

Categoría	Sitio (solo algunos ejemplos; orden alfabético)
Almacenamiento	Dropbox, MediaFire, Mega
Colaboración	Asana, Evernote, Todoist, Trello
Documentos	GoogleDocs, Pixlr (edición fotos), Prezzi (creación presentaciones), SiteBuilder (creación web), Slideshare (presentaciones/documentos), StackEdit (edición en la web)
Media	Flickr (fotos), NetFlix (vídeos, películas), Spotify (música), Wikipedia (enciclopedia), Youtube
Servicios	bit.ly (URL), Flood (test), Ionic (apps), Gmaps, Mailchimp (mail+), Panda (antivirus)
Social	BuzzStream, Facebook, Instagram, Linkedin, LuckyOrange, Twitter, Zoho Connect

3.1. Drupal

Drupal es un sistema de gestión de contenido (CMS) diseñado para ser una solución adecuada de administración de contenidos para usuarios no técnicos que necesitan simplicidad y flexibilidad a través de un enfoque modular para la construcción de sitios.

Drupal está basado en módulos que permitirán que el usuario que creará los contenidos pueda gestionarlos, pero también definir muchos aspectos del CMS de acuerdo a sus necesidades y con la máxima eficiencia. Es por ello que en el mismo CMS podrán cohabitar, por ejemplo, un sitio de noticias, una

tienda en línea, una red social, un blog y un wiki; en este sentido, es necesaria solamente la combinación de los módulos adecuados y dejar el resto a la creatividad del usuario final.

Los módulos, al estar interconectados sobre la misma base, pueden intercambiar información, y es por ello que el usuario puede definir una página de noticias donde tenga las cinco últimas entradas de un blog incrustado en el mismo sitio con unos pocos clics (a través de una vista) y sin instalar complementos o integrar ni desarrollar código. Obviamente esto no es gratuito y está sustentado por centenares de módulos y un núcleo sofisticado que ha costado, como proyecto *open-source*, gran esfuerzo y dedicación de la comunidad (en 2017, 105.793 usuarios que contribuyen activamente, una media de dos mil *commits* por semana y quinientos comentarios) y con grandes usuarios corporativos/institucionales.

En su arquitectura, Drupal trata la mayoría de los tipos de contenido en una estructura llamada *nodo* en la que las páginas estáticas, las publicaciones en el blog y las noticias se almacenan de la misma manera y la estructura de navegación del sitio se diseña por separado editando menús, vistas (listas de contenido) y bloques (enlaces a diferentes secciones del sitio).

Los nodos tienen la información estructurada que pertenece a una entrada de blog (como título, contenido, autor, fecha) o una noticia (título, contenido, fecha de inicio, fecha de entrega), mientras que el sistema de menús, la taxonomía (etiquetado de contenido) y las vistas crean una arquitectura de información.

Finalmente, el tema definido junto con los módulos de visualización, como los paneles, controla cómo todo esto se visualiza en la pantalla. Esto facilita enormemente el diseño y la configuración, ya que estas capas se mantienen separadas, y se puede proporcionar una navegación y una presentación completamente diferentes de su contenido a diversos usuarios según sus necesidades específicas y funciones [Dru].

3.1.1. Instalación sobre Ubuntu

Para esta prueba se utilizará una MV KVM (Ubuntu 16.04, 3G RAM, 15 GB disco, conexión *bridged*, *openssh-server*, *firefox* y *Lubuntu*), pero si no se desea invertir tiempo en su instalación o configuración, se puede utilizar alguna de las Virtual Appliances provistas por el propio desarrollador, Turnkey, o Bitnami.

Para Drupal es necesario disponer de Apache o Nginx, MySQL y PHP (lo que se conoce como AMP *stack*; hay varias VA que permiten disponer de ellos, o también se puede utilizar Compose o Drush para su instalación); en esta prueba se instalarán individualmente.

```
apt-get update
```

```
apt-get install apache2
apache2ctl configtest
```

Se verá que ServerName no está definido, por lo cual hay que editar */etc/apache2/sites-enabled/000-default.conf* y agregar esta variable con el dominio de nuestra máquina. En esta prueba no se tiene un DNS, de modo que se ha agregado la ip + dominio-FQDN en */etc/hosts*.

```
systemctl restart apache2
apache2ctl configtest
```

Que deberá ejecutarse sin errores y podremos acceder a la página de pruebas poniendo en un navegador (dentro de la misma máquina) el dominio (si se desea probar desde un navegador externo, por ejemplo, desde el *host*, utilizar la IP o modificar el */etc/hosts* de este para incluirlo). Luego se debe instalar Mysql y PHP para ello:

```
apt-get mysql php php-mysql php-mcrypt libapache2-mod-php \
   php-simplexml php-gd
```

Indicar (y recordar) el *passwd* de Mysql y después se puede agregar mayor seguridad ejecutando el comando `mysql_secure_installation` (eliminará los usuarios anónimos, BD de ejemplo y cambiará las protecciones a las tablas).

Si se desea que *index.php* tenga prioridad frente a *index.html* se puede cambiar el archivo */etc/apache2/mods-enabled/dir.conf* poniendo uno delante de otro y reiniciando el servidor.

Para probar que *php* está integrado y funciona, se tiene que crear un archivo */var/www/html/index.php* con el siguiente contenido
`<?php phpinfo(); ?>` y al acceder a la URL (si hemos cambiado el orden para que se lea primero *index.php* o, sino, indicando al final el nombre este archivo), se podrá ver la información de PHP.

Luego se deberá habilitar el módulo *rewrite* para permitir URL limpias y permitir la lectura de *.htaccess* del directorio de Drupal (*/var/www/html/drupal* en este caso) y adecuar los archivos de instalación:

```
a2enmod rewrite
vi /etc/apache2/apache2.conf
<Directory /var/www/html/drupal>
AllowOverride All
    </Directory>
systemctl restart apache2
cd /var/www/html/drupal/sites/
chown www-data default
cd default
cp default.setting.php setting.php
chown www-data settings.php
```

Luego desde la URL *dominio/drupal* (o el directorio que se le haya dado) se podrá hacer la instalación, que indicará si falta algún complemento; lo cual se deberá solucionar antes de recargar y continuar.

Después pedirá el nombre de una nueva base de datos y el usuario y *passwd* para acceder a `mysql` (el que se dio durante la instalación para el usuario *root*).

Finalmente, hay que dar la información del usuario administrador, el nombre del sitio y ya estará en funcionamiento. En *Manage* → *Reports* → *Status reports* se podrá acabar de configurar el sitio, por ejemplo con los módulos de *Trusted Host Settings*, para evitar ataques de *HTTP HOST Headers*.

Este procedimiento es la instalación habitual, pero se desea que Drupal funcione como CMS SaaS y albergue muchos sitios bajo dominios diferentes basados en el mismo núcleo.

En primer lugar, para el entorno (aunque podrían ser los mismos parámetros para producción, pero sería conveniente disponer de un DNS propio) se adecuará para tener dos dominios diferentes en la misma IP; para ello modificamos */etc/hosts* y agregamos:

```
<ip-servidor>    saas1.nteum.org  saas1
<ip_servidor>    saas2.nteum.org  saas2
```

Luego, en */etc/apache2/sites-enabled/000-default.conf* (o donde se tenga configurado el sitio), agregar el siguiente contenido (en cursiva), salvar y reiniciar Apache:

```
<VirtualHost saas1.nteum.org:80>
    DocumentRoot /var/www.html/drupal
    ServerName saas1.nteum.org
</VirtualHost>
<VirtualHost saas2.nteum.org:80>
    DocumentRoot /var/www.html/drupal
    ServerName saas2.nteum.org
</VirtualHost>
systemctl restart apache2
```

Ahora se deben crear las bases de datos para los dos sitios; la forma más simple es hacerlo en la línea de comandos, desde un terminal (a cada comando pedirá el *passwd* del usuario *root*):

```
mysql -u root -p -e "CREATE DATABASE saas1 CHARACTER SET utf8 COLLATE utf8_general_ci";
```

Conectarse a la base de datos y cambiar los privilegios (cambiar en el comando el nombre, usuario y *passwd*):

```
mysql -u root -p
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER, CREATE TEMPORARY TABLES ON
 saas1.* TO 'root'@'localhost' IDENTIFIED BY 'mi-password';
exit
```

Repetir el procedimiento para crear y configurar un BD saas2. Ahora es necesario crear la estructura de archivos y directorios dentro de */var/www/html/drupal/sites*, que tendrán el nombre de sitio por comodidad (puede ser cualquier nombre):

```
cd /var/www/html/drupal/sites
mkdir saas1.nteum.org; mkdir saa2.nteum.org
cp default/default.settings.php saas1.nteum.org/settings.php
cp default/default.settings.php saas2.nteum.org/settings.php
chown -R www-data saas1.nteum.org
chown -R www-data saas2.nteum.org
```

Es decir, deben quedar dos directorios como el anterior *default* y con un archivo *settings.php* dentro con los permisos adecuados (tanto el directorio como el archivo). En cada uno de ellos se debe modificar la información de la base de datos para adecuarla a la configuración anterior; después de editarlo, cambiar *$databases = array ()*; por:

```
$databases = array (
  'default' =>
  array (
    'default' =>
    array (
      'database' => 'saas1',
      'username' => 'root',
      'password' => 'password',
      'host' => 'localhost',
      'port' => '',
      'driver' => 'mysql',
      'prefix' => '',
    ),
  ),
);
```

Repetir los pasos para el otro sitio y cambiar el nombre de la base de datos (en un entorno real, se debería haber puesto usuarios diferentes para la BD, y esto se debería reflejar aquí).

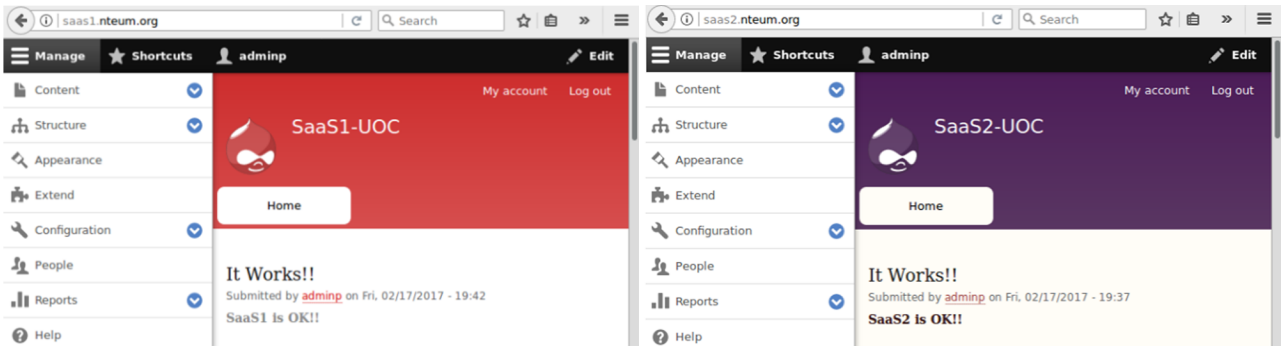
Finalmente, se debe crear el archivo */var/www/html/drupal/sites/sites.php* para agregarle el mapeo de *sitio* > *directorio*:

```
cd /var/www/html/drupal/sites
cp example.sites.php sites.php
vi sites.php
    $sites['saas1.nteum.org'] = 'saa1.nteum.org';
    $sites['saas2.nteum.org'] = 'saa2.nteum.org';
```

Ahora ya se podrá acceder a la URL con el argumento *install.php* y comenzar a instalar cada sitio (es lo que haría el usuario final: usuario de la plataforma CMS SaaS), por ejemplo,*http://saa1.nteum.org/install.php*.

Como se podrá observar ya cargará la BS de datos adecuada y solo se deberá introducir el *passwd*; luego, el procedimiento será como el anterior para cada sitio.

Las dos imágenes a continuación muestran el resultado del despliegue y una pequeña configuración para mostrar su independencia (si bien están utilizando la misma plataforma/componentes).



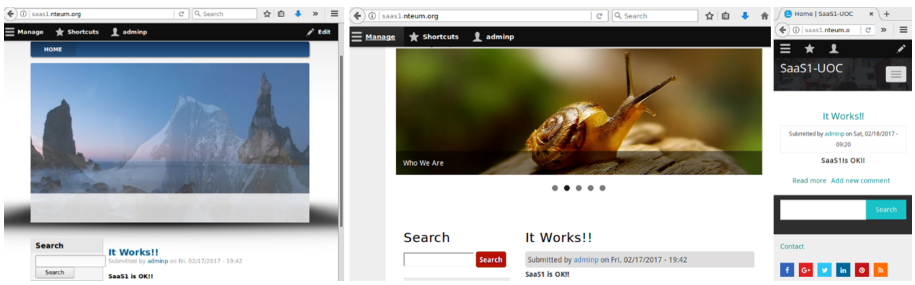
Una muestra de las posibilidades de configuración de Drupal se hará con la apariencia (*Themes*) donde se puede observar visualmente su potencialidad. Hay una gran cantidad de ellos construidos por la comunidad y de uso libre con diferentes objetivos, ámbitos, diseños, funcionalidad, etc. que pueden ser descargados e instalados (solo se debe tener en cuenta que en la búsqueda cabe indicar el *CoreCompatibility* a la versión 8.x, ya que los *themes* son dependientes de la versión de Drupal instalada).

Drupal dispone de varios sitios donde instalarlos; el más general es *directorio_instalación/themes*, pero también pueden ser en *sites/all/themes* o, si se desea, puede ser específico para un sitio en *sites/directorio_site/themes*. Solo se debe descargar y descomprimir el archivo, por ejemplo, el tema *business*:

```
tar xzvf business-8.x-1.7.tar.gz
mv business /var/www/html/drupal/themes
```

Luego, hay que ir a la interfaz de administrador de *Drupal* → *Appereance* → *Install and set as default* y ya se tendrá el tema cargado y el cliente lo podrá ajustar a sus necesidades. Es importante tener en cuenta que en *People* → *Roles/Permissions* se podrá habilitar si se deja que el cliente cambie o modifique el tema/módulos o cualquier otra cosa del sitio en función del rol que se le asigne y los permisos dados a este rol.

Las imágenes a continuación muestran ejemplos de la configuración de los temas *Business*, *Danland* y *Zymphonies* (que presenta un diseño *responsive* apto para dispositivos móviles).



Otros ejemplos SaaS en la creación y gestión de contenidos podrían ser *blogs* (por ejemplo, a través de WordPress), wikis (por ejemplo, con MoinMoin –*farms*– o MediaWiki –*families*), Social Networks (por ejemplo, OpenSource-SocialNetwork, Elgg, Mahara, Loved by Less, HumHub), entre muchas otras opciones.

3.2. ownCloud

Otra de las aplicaciones interesantes para mostrar la potencialidad del SaaS es ownCloud, que permite tener acceso universal a los archivos del cliente a través de una interfaz web (WebDAV), ver y sincronizar fácilmente los contactos, calendarios y marcadores en todos los dispositivos (cuenta con clientes para Linux, W, MacOS o Android/IOS) y la edición básica directamente desde la web. Además, ownCloud Server es extensible a través de una API simple, funciona con cualquier tipo de almacenamiento y es equivalente a productos como Dropbox, GoogleDrive o similares.

La empresa ownCloud Inc. es la responsable del desarrollo, y el modelo de negocio (similar a otras empresas con software *open-source*) es a través de servicios y soporte a clientes empresariales, que mantienen el software en versión *open-source* y con el aporte de la comunidad [Own].

Para su instalación es recomendable seguir los consejos dados por el desarrollador para pequeños grupos/departamentos (hasta ciento cincuenta usuarios), pymes (hasta mil usuarios) o grandes empresas (entre 5k y 100k usuarios) sobre la arquitectura adecuada para la autentificación, BD, *proxies* y otros componentes.

En esta prueba se realizará una instalación simple (apta para quinientos usuarios) y sin servicio de autentificación (Ldap) con Ubuntu16.04, Apache2.4, MySQL5.7 y PHP7 (un clon de la máquina virtual utilizada para Drupal).

Para su instalación sobre una máquina con AMP, a la cual hemos agregado un dominio propio en */etc/hosts* (*owncloud.nteum.org*) y adecuado Apache (como se hizo en Drupal), se han realizado los siguientes pasos:

```
apt-get install php-zip php-curl php-mbstring
tar xjvf owncloud-9.1.4.tar.bz2
mv owncloud /var/www/html/owncloud
mkdir /var/www/html/owncloud/data; chown www-data /var/www/html/owncloud/data
systemctl apache2 restart
```

Con esto ya se podrá acceder a la URL *http://owncloud.nteum.org/owncloudy* se podrá indicar si falta instalar algún módulo PHP, el cual se deberá instalar, reiniciar Apache y recargar la página.

A continuación, se podrá finalizar la instalación (introduciendo los datos como administrador, BD, directorio data, etc.) y ya se podrá acceder a la interfaz de administración para trabajar con archivos y directorios.

Además de la instalación que se ha realizado aquí a través del archivo, ownCloud soporta una instalación vía web, a través de los repositorios para cada distribución o una *Virtual Appliance* ya configurada y adecuada para ejecutar en una MV (diferentes formatos OVA, Qcow2, VHDX, VMDK, VMX).

Desde el menú de *Admin* (superior derecha) se podrán crear los usuarios/grupos necesarios y en el menú izquierdo instalar más aplicaciones, como por ejemplo módulos (algunos de ellos ya estarán instalados por defecto): antivirus, *tags* colaborativos, encriptación, comentarios a los archivos, recuperación de archivos borrados, capacidad para conectarse a sitios externos, conexión a almacenamiento externo (S3, Drive, Dropbox, etc.), federación de sitios ownCloud, notificaciones, PDF Viewer, galería de imágenes, visualizador de vídeo, compartición de archivos, editor de texto, calendario, contactos, tareas y copias de resguardo, entre las más utilizadas.

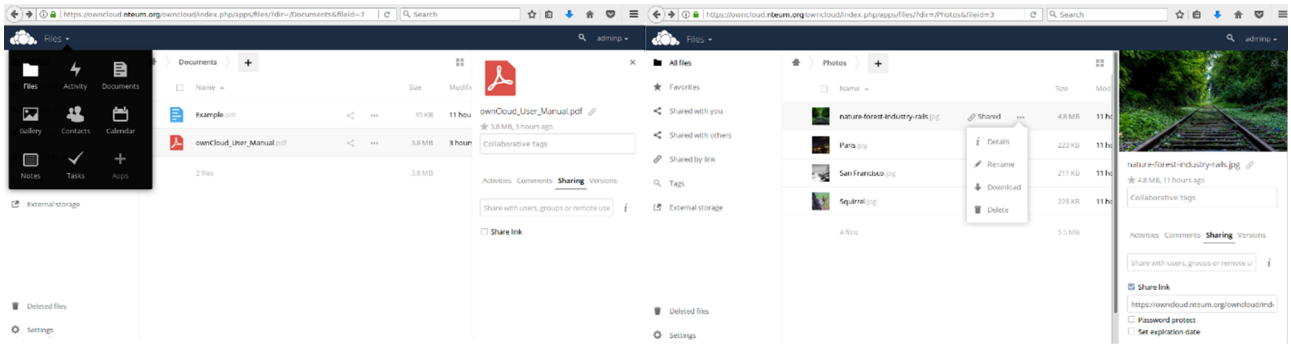
Un detalle que puede parecer que no se puede cambiar es el tamaño máximo de un archivo que se puede subir al servidor (desde la interfaz gráfica no es posible), pero dado que es un parámetro de php se deberá modificar */etc/php/7.0/apache2/php.ini* y modificar/crear el parámetro *upload_max_filezise = 10M* (en este ejemplo se ha puesto a 10Mbytes).

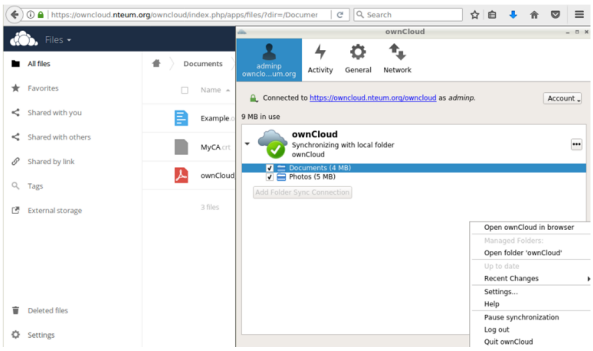
ownCloud dispone de una *appstore* donde se pueden encontrar más módulos e interactuar con los desarrolladores y consultar diferentes cuestiones en relación con la plataforma. Es conveniente, dada la cantidad de opciones y posibilidades de la plataforma, consultar detenidamente el manual de administrador de ownCloud y especialmente el apartado de configuración del servidor (por ejemplo, importación de certificados SSL, uso del comando `occ`, configuración del antivirus, *mail server*, *memory caching*, repositorios externos, federación, *reverse proxy*, entre otras).

Para complementar la confidencialidad del servicio (junto con la encriptación de los archivos sobre el servidor), es conveniente que se habilite **https** sobre Apache (simplemente habilitando el módulo ssl y configurar el sitio; se puede partir del ejemplo por defecto de ssl que incluye Apache). El problema será disponer de un certificado firmado por una CA que puede comprarse, aunque para ello se deberá tener el dominio registrado, o instalarse una CA y generar los certificados para el dominio que se disponga (*owncloud.nteum.org* en este caso). Para ello (como se especificó en la asignatura de Administración avanzada de GNU/Linux), se puede instalar el paquete XCA y generar una CA raíz y su certificado, un certificado de servidor (hay que poner en *Common Name* el dominio del servidor, por ejemplo, *owncloud.nteum.org*), la petición y firmarlo con la CA, exportar el certificado raíz de la CA y el certificado del servidor (como *pem + key*); este es el que se deberá poner en la configuración de Apache

(en el archivo que configure el *VirtualHost* para el puerto 443) como parámetro de la sentencia *SSLCertificateFile* (cabe recordar que hay que reiniciar Apache posteriormente). El certificado raíz se deberá instalar en los navegadores de los clientes (*Preferences* → *Advanced* → *View Certificates* → *Authorities* → *Import*), por lo cual es conveniente compartirlo por el propio ownCloud (*MyCa* en este caso) e indicar a los clientes cómo instalarlo.

Las figuras siguientes muestran ownCloud sobre https (con el certificado firmado, como se puede observar en el candado verde a la izquierda de la URL) con algunas de las posibilidades de la interfaz con algunos módulos instalados, así como el cliente para Linux con el menú de configuración/sincronización.





Para un sistema en producción es conveniente vincularlo a un sistema de almacenamiento externo; se puede hacer desde la CLI, pero también desde la interfaz gráfica a través de la *app External Storage Support*. Entre las posibilidades que soporta ownCloud se encuentran: Amazon S3, Dropbox, FTP/FTPS, Google Drive, Local, OpenStack Object Storage, ownCloud, SFTP, SMB/CIFS, WebDAV.

Como se ha podido apreciar, es una plataforma SaaS que permite una gran funcionalidad y características destacadas incluso superiores a muchas comerciales. Es importante profundizar en ella para lograr una correcta y segura configuración, y también existe un conjunto de proveedores que la ofrecen como servicio SaaS, por lo cual es posible contratarla o acceder a alguno de los planes gratuitos de los que disponen.

3.2.1. OwnCloud y LibreOffice

LibreOffice es un paquete *open-source* de ofimática que permite la creación y modificación de diferentes documentos en formato abierto a través de diferentes aplicaciones como procesador de textos, hojas de cálculo y presentaciones. Collabora, es una empresa que ha desarrollado una versión *open-source on-line* de LibreOffice y que a través de CODE (*Collabora Online Development Edition*) despliega un servidor, basado en Docker, que presenta un *web-service* a través del protocolo WOPI (*Web Application Open Platform Interface*). Este servicio permite disponer de interaccion y edición de documentos *online* y que puede ser integrado con las aplicaciones de sincronización y compartición de ficheros como ownCloud, NextCloud o Pydio entre otras. [Cod]

En la página de CODE se encuentran las instrucciones para descargar una MV Virtualbox y poner en marcha el entorno integrado de CODE + ownCloud. Se debe tener en cuenta que la MV es OpenSuse y se recomienda utilizar la interfaz de red de Vbox como Bridged, asignándole IP (*/etc/sysconfig/network/ifcfg-enp0s3*) o el dispositivo correspondiente, el gateway en */etc/sysconfig/network/routes*) para luego acceder a ella a través del host o desde una máquina externa (*http://<IP>/owncloud*). Además, es necesario gestionar los *trusted_domains* de Owncloud y se deberá agregar la IP del servidor a */srv/www/htdocs/owncloud/config/config.php* donde dice *trusted_domains* ... 0 => 'IP' reemplazando IP por la <direccion IP> antes configurada y recargar la página http://<IP>/owncloud). En las instrucciones para desplegar la máquina virtual se encuentra los usuarios/passwds para ownCloud que son **admin/admin** y para la MV que son *root/skew25.kiwis*.

No obstante, es interesante desplegar CODE desde el contenedor Docker e integrarlo con el OwnCloud instalado a través de un *plugin de Collabora OnLine*. Las instrucciones están disponibles en la página antes indicada, pero para esta prueba se ha utilizado una única máquina Ubuntu 16.04 LTS donde se ejecuta el contendor con CODE y se tiene instalado OwnCloud. En primer lugar, se debe descargar poner en marcha el contenedor. Existen unas pequeñas diferencias sobre la ejecución del contenedor ya que tanto OwnCloud como CODE van por SSL y para realizar la misma experiencia que podrá hacer un usuario en primera instancia, se ha partido de la hipótesis que no se disponen de certificados firmados por una CA sino que son certificados son autofirmados (y se crearán en esta prueba):

```
docker pull collabora/code
docker run -t -d -p 127.0.0.1:9980:9980 -e "domain=cloud.nteum.org" \
        -e "username=admin" -e "password=admin" -net host \
        --cap-add MKNOD collabora/code
```

En nuestro */etc/hosts* se tienen dos entradas donde la primera (sysubu) hace referencia al *reverse-proxy* (bajo SSL) necesario para acceder al contenedor desde cualquier sitio ya que CODE publica el servicio sobre el *localhost:9980* (y

también se evita tener que integrar el certificado de CODE con la instalación externa), y la segunda (*cloud*) correspondiente a servicio de ownCLoud (también bajo SSL pero en este caso con el certificado autofirmado, no como en el caso anterior que se utilizó *owncloud.nteum.org* con el certificado frimado por la propia CA):

```
192.168.122.1   sysubu.nteum.org  sysubu
192.168.122.1   cloud.nteum.org  cloud
```

Para generar los certificados se ha utilizado:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout /etc/ssl/private/cloud.key -out /etc/ssl/certs/cloud.crt
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout /etc/ssl/private/sysubu.key -out /etc/ssl/certs/sysubu.crt
```

Los archivos de configuración para Apache2: */etc/apache2/sites-available/owncloud.conf*

```
<VirtualHost _default_:443>
                DocumentRoot /var/www/html
                ErrorLog ${APACHE_LOG_DIR}/error.log
                CustomLog ${APACHE_LOG_DIR}/access.log combined
                SSLEngine on
                SSLCertificateFile        /etc/ssl/certs/cloud.crt
                SSLCertificateKeyFile  /etc/ssl/private/cloud.key
                <FilesMatch "\.(cgi|shtml|phtml|php)$">
                                SSLOptions +StdEnvVars
                </FilesMatch>
                <Directory /usr/lib/cgi-bin>
                                SSLOptions +StdEnvVars
                </Directory>
        </VirtualHost>
```

Y para CODE con el proxy server */etc/apache2/sites-available/colla.conf* (es copia modificada de la de referencia donde se han cambiado los dominios y las rutas a los certificados):

```
<VirtualHost *:443>
ServerName sysubu.nteum.org:443
               SSLEngine on
               SSLCertificateFile /etc/ssl/certs/sysubu-pub.crt
               SSLCertificateKeyFile /etc/ssl/private/sysubu.key
               SSLProtocol             all -SSLv2 -SSLv3
SSLCipherSuite ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-
SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-
```

```
SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-
SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-
RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-
DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-
SHA:DES-CBC3-SHA:!DSS
SSLHonorCipherOrder     on
               # Encoded slashes need to be allowed
               AllowEncodedSlashes NoDecode
               # Container uses a unique non-signed certificate
               SSLProxyEngine On
               SSLProxyVerify None
               SSLProxyCheckPeerCN Off
               SSLProxyCheckPeerName Off
               # keep the host
               ProxyPreserveHost On
               # static html, js, images, etc. served from loolwsd
               # loleaflet is the client part of LibreOffice Online
               ProxyPass           /loleaflet https://127.0.0.1:9980/loleaflet retry=0
               ProxyPassReverse    /loleaflet https://127.0.0.1:9980/loleaflet
               # WOPI discovery URL
            ProxyPass          /hosting/discovery https://127.0.0.1:9980/hosting/discovery retry=0
              ProxyPassReverse    /hosting/discovery https://127.0.0.1:9980/hosting/discovery
# Main websocket
               ProxyPassMatch "/lool/(.*)/ws$" wss://127.0.0.1:9980/lool/$1/ws nocanon
# Admin Console websocket
               ProxyPass   /lool/adminws wss://127.0.0.1:9980/lool/adminws
# Download as, Fullscreen presentation and Image upload operations
               ProxyPass           /lool https://127.0.0.1:9980/lool
               ProxyPassReverse    /lool https://127.0.0.1:9980/lool
</VirtualHost>
```

A continuación, se debe habilitar los sites, cargar los módulos y reiniciar Apache2:

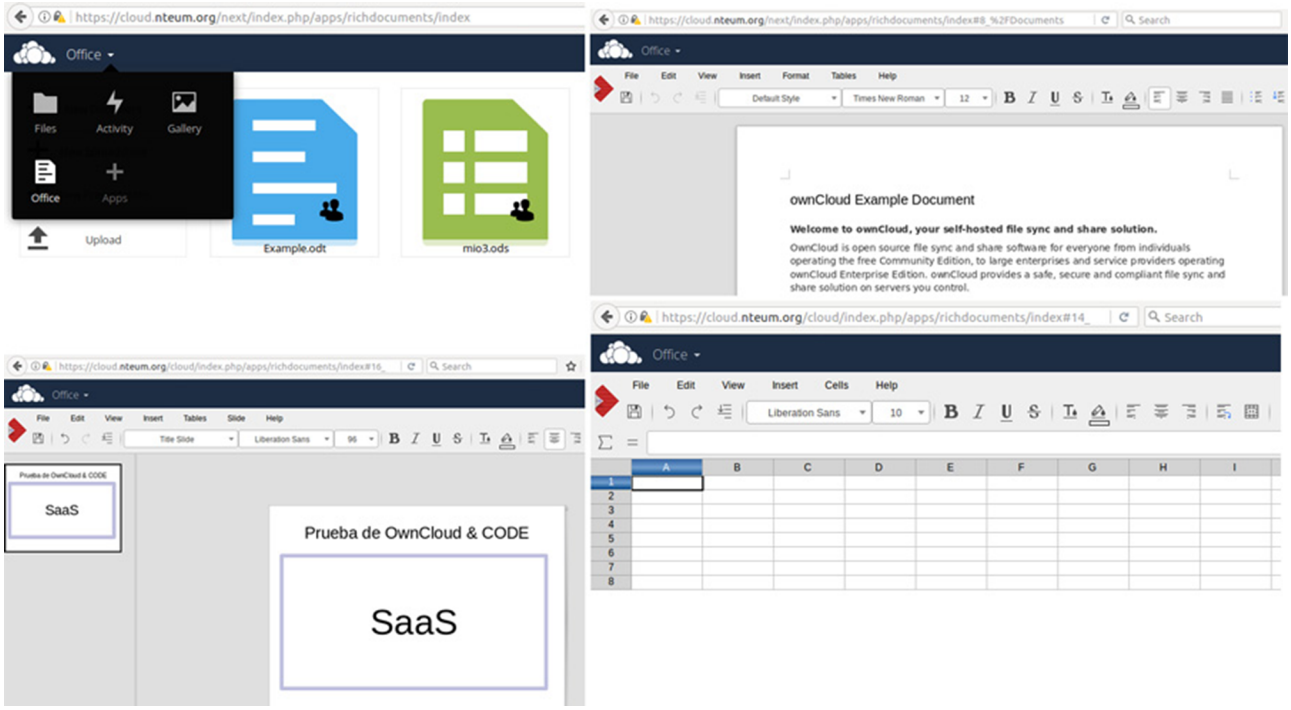
```
a2ensite colla
a2ensite owncloud
a2enmod proxy proxy_wstunnel proxy_http ssl
systemctl restart apache2
```

Antes de acceder a ownCloud se debe copiar el certificado del site proxy de CODE en ownCloud haciendo:

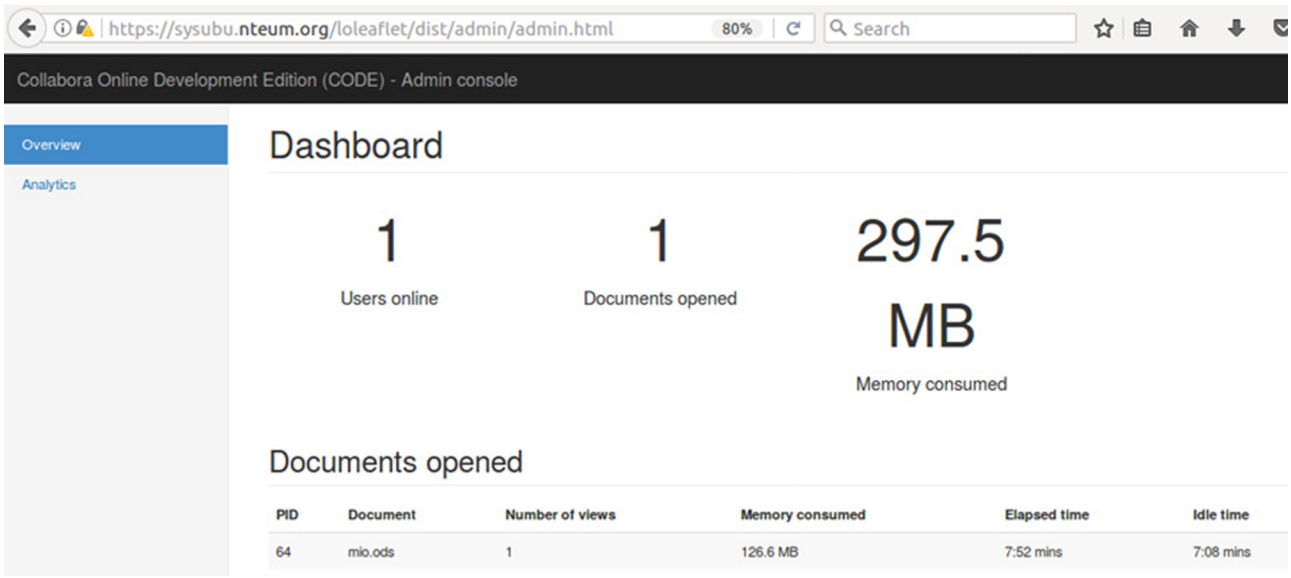
```
cat /etc/ssl/certs/cloud.crt >> \
/var/www/html/<Directorio-Owncloud>/resources/config/ca-bundle.crt
```

Luego se puede acceder a *OwnCLoud* → *Apps* → *Not Enabled* → *Collabora Online plugin* → clic en *Enable*. Desde el menú de *Admin* (arriba a la derecha), seleccionar *Admin* → Collabora Online → Collabora Online Server → introducir *https://sysubu.nteum.org* → *Apply*.

Luego desde *Files* → *Documents* → + o directamente desde Files → Office se podrán crear/abrir los documentos del procesador de textos/hoja de cálculo y presentaciones como se muestra en las figuras a continuación.



Además, se puede consultar el estado del servidor CODE accediendo a la URL *https://sysubu.nteum.org/loleaflet/dist/admin/admin.html* (con el usuario y *passwd* indicados cuando se puso en marcha el contenedor Docker) como muestra la figura siguiente:



En caso de disponer una CA se pueden sustituir los certificados autofirmados por los firmados, cambiarlos en los archivos de configuración si se les ha cambiado el nombre, reiniciar el servicio, copiar el certificado del servidor CODE a ownCloud y comprobar su funcionalidad.

Existen diferentes referencias como instalar CODE sobre Ubuntu 16.04 sin Docker para evitar las limitaciones que según algunos usuarios han detectado sobre este contenedor y que según algunas experiencias es más simple que la instalación del repositorio original.

3.3. OpenBravo

Otro ejemplo de SaaS orientado a pymes es Openbravo, un ERP (*enterprise resource planning*) basado en tecnología web de la empresa del mismo nombre (2001) bajo una licencia Openbravo Public License (la cual se basa en MozillaPL).

La plataforma actual es la evolución de diferentes productos; desde el año 2015 se denominan Suite de Comercio y Suite de Negocio, con el objetivo de satisfacer la industria/empresa minorista, y actualmente se puede desplegar en la forma tradicional, en el *cloud* o acceder como SaaS a través de terceros; en [Obg] pueden encontrarse las orientaciones e información relacionada, además de una guía de cómo comenzar en esta plataforma.

Esta plataforma, originalmente basada en Compiere ERP (*open-source*), presenta una arquitectura flexible basada en una abstracción que permite a los desarrolladores diseñar bajo paradigmas de modelos en lugar de código y una separación estricta entre la lógica de presentación y la lógica de negocios, lo cual facilita la integración con otros programas a través de una interfaz bien

definida. Esto le permite integrarse con otras aplicaciones *open-source* como Magento (tienda en línea), Pentaho Business Intelligence, ProcessMaker BPM, Liferay Portal y SugarCRM, entre otros.

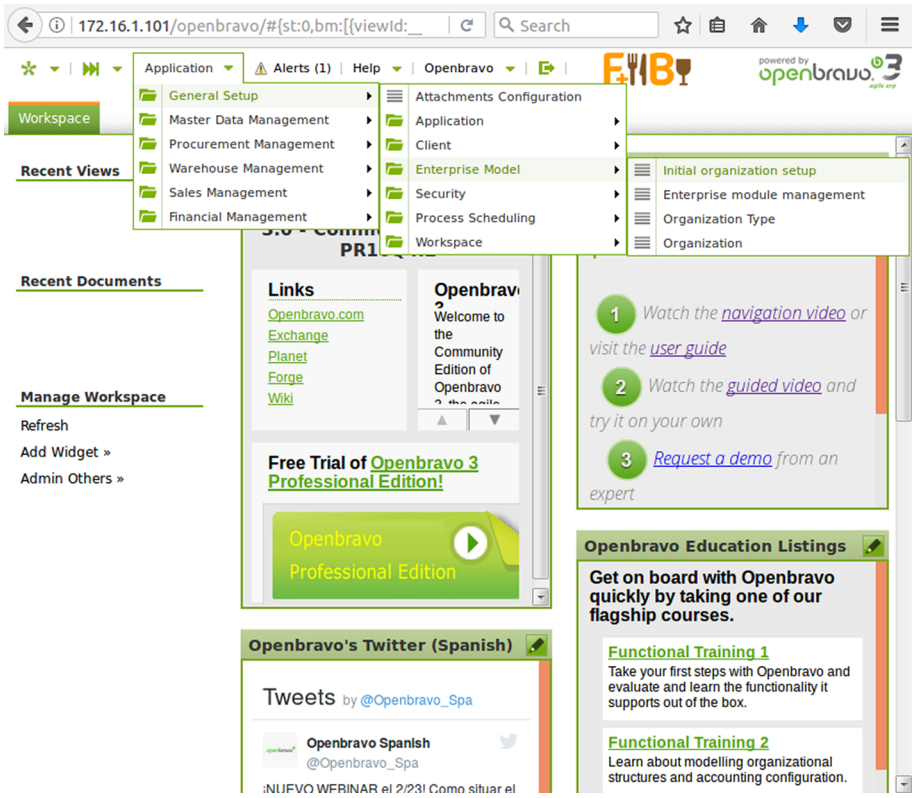
Su utilización permite a las empresas automatizar y registrar los procesos de negocio más comunes como, por ejemplo, ventas, compras, fabricación, proyectos, finanzas y MRP, y además, dadas las facilidades de extensión de la plataforma, que se puedan agregar módulos, lo cual genera un ecosistema disponible a través de OpenBravo Forge.

El modelo de negocio de la compañía se basa en el software *open-source* con el aporte de la comunidad y ofrece soporte, servicios, formación y módulos específicos mediante diferentes modelos de pago; se recomienda para usuarios (recién iniciados en OpenBravo) seguir la guía del usuario para conocer los términos, los conceptos, el flujo de trabajo y las áreas de aplicación de la plataforma [Obd][Obr].

Su instalación puede realizarse de la forma tradicional, lo cual solo es recomendable únicamente para expertos dada la cantidad de software y configuraciones que se deben realizar (BD, JDK, Tomcat, Ant, Apache, Mercurial y el propio OpenBravo), y desde una *Virtual Appliance* (disponible como Amazon EC2, ISO o VMware), además del OpenBravo On Demand (SaaS) ya mencionado. En esta prueba se utilizará la VA mediante un ISO y se instalará sobre VirtualBox como prueba de concepto.

Esta prueba se ha realizado sobre Ubuntu 16.04, donde se ha instalado Virtualbox y creado una interfaz *HostOnly* (para conectarse luego a la VA desde el *host* con la configuración habitual *Preferences* → *Network* → *HostOnlyNetwork* → Ip:172.16.1.100/24 y sin DHCP) y una MV (3GB RAM, 16GB de disco, una interfaz NAT y otra *HostOnly*). Desde SourceForge se ha descargado la ISO (*openbravo-3.0PR16Q4.2-amd64.iso* en este caso) y se ha instalado la VA, en la MV antes creada, en la forma habitual como si fuera un SO.

Una vez instalada y arrancada la MV se ha accedido con usuario/*passwd* **openbravo/openbravo** respectivamente, y se ha configurado eth1 en */etc/network/interfaces* en forma *static* con IP 172.16.1.101/24 y reiniciado los servicios de red. Desde el *host*, a través de un navegador, se ha conectado a la URL *http://172.16.1.101* y se ha accedido con el *usuario/passwd* **Openbravo/openbravo** respectivamente (tener en cuenta la O mayúscula del usuario); con ello se accede a la interfaz de la plataforma, como se puede ver en la imagen siguiente.



Para su evaluación, la VA viene configurada con una empresa de ejemplo, roles, regiones de la empresa y es necesario seleccionar el perfil y comenzar a configurar los módulos y flujos de trabajo de acuerdo a como se indica en la guía de usuario antes mencionada. Como se puede observar, es una plataforma compleja que exige un amplio conocimiento del ERP y que desarrolla todos los aspectos esenciales en la gestión de una empresa, por lo cual es necesario tiempo y dedicación para conocer su funcionalidad y potencialidad [Obd][Obr].

3.4. AppServer

Appserver.io (AS) es una infraestructura *open-source* de la nueva generación de PHP que consiste en un servidor de aplicaciones (*application server*) extremadamente rápido escrito en PHP y que incluye todos los servicios adicionales necesarios para proveer una infraestructura de este tipo.

Este permite utilizar todos los servicios o solo servicios seleccionados específicamente para la aplicación determinada sin ajustes adicionales, ni herramientas añadidas o servicios agregados. Es equivalente a otros servidores de aplicaciones como WildFy, pero en lugar de Java está escrito en PHP (incluye *Servlet Engine*, *Message Queue*, *Timer Service*, *Persistence Container*); es altamente eficiente y una alternativa a desarrolladores en PHP que no necesitan trabajar con otros lenguajes; también es adecuado para usuarios de diferentes roles (desarrolladores, vendedores, agencias, compañías...).

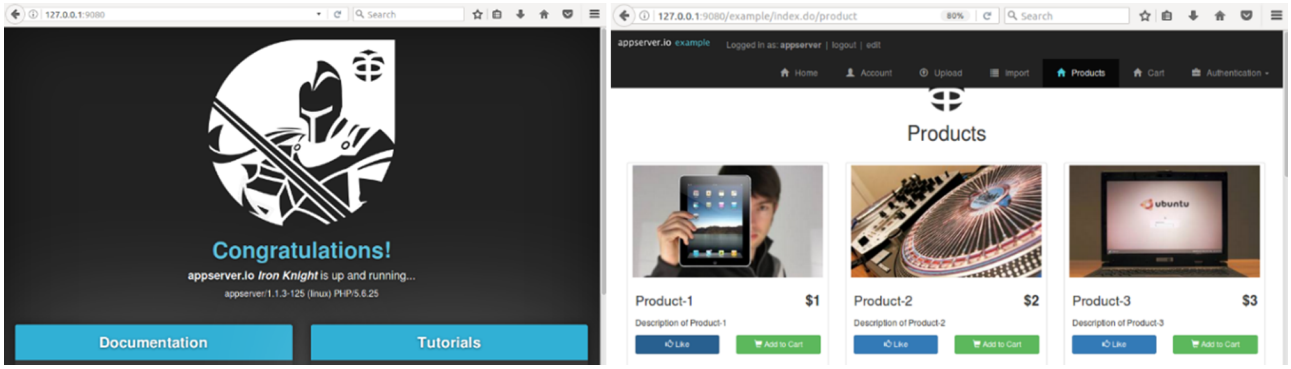
Appserver.io (*aka Iron Horse*) es el primer *application server* totalmente escrito en PHP que supera las prestaciones de un *stack* LAMP que presenta sus límites en entornos empresariales, sobre todo en la escalabilidad y mantenimiento. Para ello, AS utiliza como primera capa un *webserver* basado en HTTP/1.1, el *Persistence-Container* y *Servlet-Engine* como segunda y *Message-Queue* y *Timer-Service* como última. Estos servicios, completamente *stateful*, incluyen funcionalidades como AOP, DI, *Design by Contract* y *Annotations*, que aceleran el desarrollo de aplicaciones, la capacidad de mantenimiento y la reutilización. Además, soporta un entorno *multithreading* que permite construir aplicaciones eficientes y escalables de ámbito empresarial y sobre PHP [Ase].

Su instalación se ha realizado sobre una MV KVM con Ubuntu 14.04 (clon de las MV utilizadas anteriormente) para evitar problemas de dependencias de librerías. En primer lugar, se deben descargar los paquetes e instalarlos:

```
dpkg -i appserver-runtime_1.1.6-97-deb7_amd64.deb
apt-get install -f
dpkg -i appserver-dist_1.1.3-125-deb7_amd64.deb
apt-get install -f
```

Después de instalados, la aplicación estará en */opt/appserver*, donde en el directorio *webapps* se instalarán todas las aplicaciones que gestionará, y se puede poner en marcha/parar el servicio como cualquier otro a través del comando `service` para los tres servicios que conforman el AS: *appserver*, *appserver-php5-fpm* y *appserver-watcher*. Si se introduce como URL en un navegador *http://127.0.0.1:8090* se verá la pantalla base de AS.

Para mostrar su funcionalidad, incluye una aplicación de ejemplo a la que se accede por la URL *http://127.0.0.1:8090/example* y a la que se accede con el usuario/*passwd* **appserver/appserver.i0**, respectivamente, como se muestra en las figuras siguientes.



Como plataforma habilita su utilización como SaaS para que los clientes desplieguen sus aplicaciones (ver cómo crear la primera WebApp), pero como es PHP compatible se pueden instalar aplicaciones SaaS que utilicen un *stack* PHP.

Como prueba de ello se instalará Magento, que es una plataforma *open-source* para comercio electrónico en PHP (desarrollada inicialmente por Varien Inc., ahora Magento Inc., con el soporte de la comunidad); está considerada como una de las plataformas de renombre en su ámbito, con una cuota de mercado de casi el 30 % (informe del año 2015 de AheadWorks).

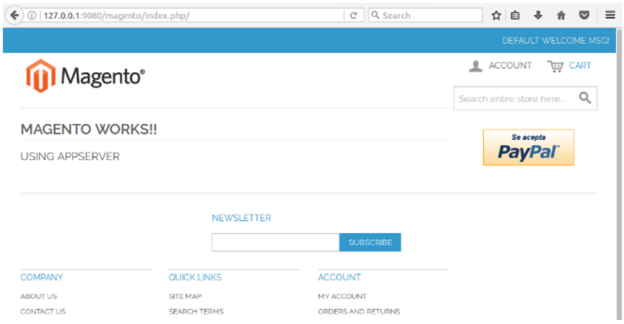
Magento utiliza una base de datos MySQL/MariaDB, PHP, el *framework* Zend; está diseñada con POO y se basa en una arquitectura modelo-vista-controlador y un modelo entidad-atributo-valor para el almacenamiento de los datos.

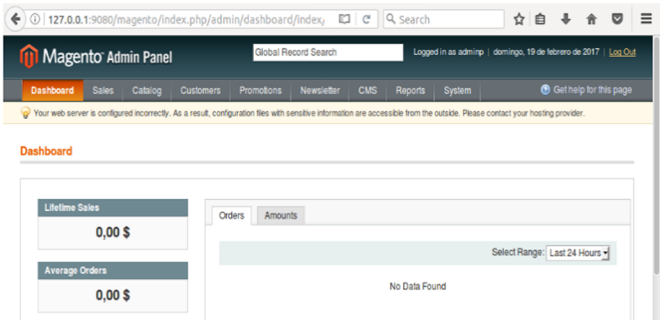
Para su instalación primero se debe instalar MySQL y configurar la base de datos, y a continuación descargar Magento (versión 1.9, ya que la 2.0 da algunas incompatibilidades) y configurar las protecciones:

```
apt-get install mysql-server
mysql -uroot -p
   create database magento;
   grant all on magento.* to "magento"@"localhost" identified by "magento";
   flush privileges;
cd /opt/appserver/webapps
git clone https://github.com/OpenMage/magento-mirror.git magento
chown -R www-data:www-data magento
chmod -R 755 magento
```

Finalmente se puede acceder por la URL *http://127.0.0.1:9080/magento* y finalizar la instalación (consultar la documentación para utilizar SSL). Una vez instalada, se puede acceder al *Frontend* a través de la URL *http://127.0.0.1:9080/magento/index.php* o al *Backend* agregándole a la URL */admin*.

Las dos imágenes siguientes muestran las pantallas de cada uno de los roles de Magento (*frontend* y *backend* de la tienda en línea).





Es necesario consultar la documentación para hacer los ajustes de seguridad requeridos, ya que AS no analiza mecanismos como *.htaccess* de las aplicaciones (se realiza a través de directivas en el archivo *appserver.xml*), permitir mecanismos de *rewrite*, configuración del *VirtualHost*, *logfiles* y otros ajustes pertinentes.

Como se puede observar, ofrecer aplicaciones empresariales (como Neos, Drupal, Wordpress, Joomla entre otras) en modalidad SaaS a través de appserver.io es muy sencillo y permite rápidamente disponer de un entorno *multitenancy* bajo la misma infraestructura, escalable y sin pérdida de rendimiento.

3.5. Sistemas operativos virtuales en línea

Una de las tendencias que han cobrado fuerza en los últimos años, con el auge del *cloud* y el SaaS, son los sistemas operativos en línea o virtualizados que algunos autores denominan *WebOS* por su interfaz de usuario sobre un navegador; sin embargo, no es conveniente, ya que este nombre identifica un SO para dispositivos de consumo como televisores/relojes y desarrollado inicialmente por Palm y hoy por LG.

> El objetivo de estos SO es que el usuario pueda realizar gran parte (todas) de las tareas que se realizarían sobre un ordenador local y con un sistema operativo instalado, pero sobre un recurso remoto (*cloud*) y solamente disponiendo de un navegador, conexión a internet y sin la necesidad de que el dispositivo local sea muy potente (puede ser una tableta, móvil o un ordenador de bajas prestaciones, pero que disponga de un navegador).

El navegador solo hará de contenedor para visualizar los resultados de la actividad realizada por el cliente, y la potencia y capacidad de procesamiento las aportará el servidor SaaS remoto.

Dadas las posibilidades de escalado y modelos de negocio como pago por uso que ofrece el SaaS, la potencialidad de estos sistemas es muy alta ya que, además, el cliente/usuario final aprovecha las ventajas de SaaS y no debe preocuparse por la instalación, mantenimiento, obsolescencia del SO remoto/aplicaciones ni de la infraestructura subyacente. No obstante, es importante tener en cuenta la necesidad de una conexión permanente, un ancho de banda adecuado a las necesidades del cliente, que los datos están en el servidor remoto, que es susceptible de tener *data/vendor lock-in* y el hecho de prestar especial atención a la SLA en cuanto a aspectos de privacidad y seguridad de la plataforma. Es necesario remarcar que **todos** los datos estarán sobre el servidor remoto (no como en otros casos SaaS, con solo el correo o los documentos).

Existen muchas iniciativas en este sentido y cada una con su estrategia y filosofía, por lo cual se mostrarán un conjunto de ellas que son las más referenciadas y/o utilizadas por la comunidad, centrándonos especialmente en proyectos *open-source*.

3.5.1. Chromium OS

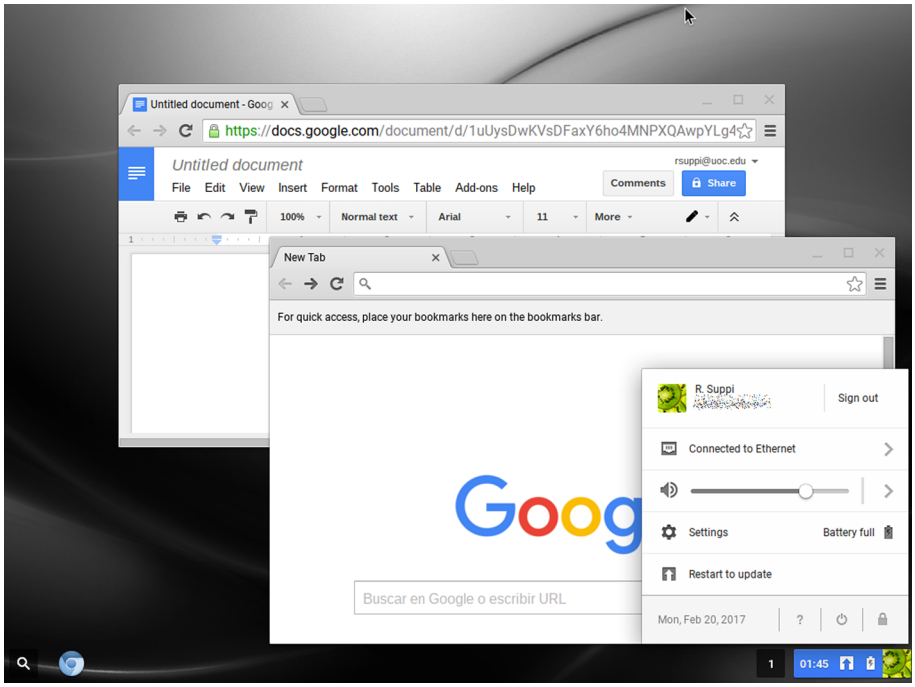
El objetivo del proyecto Chromium OS *open-source* es construir un sistema operativo que pueda ofrecer una alternativa rápida, sencilla y más segura para las personas cuya actividad depende de dispositivos móviles/ultraligeros y que disponen de una conectividad adecuada para realizar su actividad, ya sea privada o empresarial.

El SO se puede descargar y compilar a partir del código fuente y funciona sobre microprocesadores con tecnología x86, x64 o ARM; su funcionalidad se reparte en una parte mínima sobre el dispositivo local y el resto se encuentra sobre el *cloud*; de este modo, se reduce de forma considerable la potencia de procesamiento requerida localmente.

Un proyecto derivado de Chromium OS es ChromeOS, desarrollado por Google (desde 2009); es una versión propietaria instalada sobre el hardware específico de fabricantes asociados (Samsung, Acer, HP, Asus, Dell...) y orientada a dispositivos móviles/ultraligeros, también llamados ChromeBooks, y que cuenta con un AppMarket específico con aplicaciones compatibles para Chrome OS y el navegador Chrome (las últimas versiones son compatibles también con aplicaciones Android).

La forma simple de probar este sistema operativo es descargar una imagen para la arquitectura adecuada desde Arnoldthebat (se necesita algún paso más si se desea probarla en Virtualbox, ya que las imágenes están en formato IMG para construir directamente un USB y se deben transformar en ISO) o también desde empresas que generan versiones basadas en este (por ejemplo, NeverWare, que también ofrece la imagen para VirtualBox).

Para esta prueba se ha utilizado la VA de NeverWare, que se ofrece en formato OVA, por lo cual solo se debe descargar e importar la *appliance*. Después de arrancar la MV se podrá conectar con una cuenta de Google, crear una nueva o entrar como invitado.



3.5.2. OS.js

OS.js es un SO *open-source* web escrito en JavaScript, dentro del navegador, en el que el usuario tiene todo un entorno como si de un SO real se tratara (gestor de ventanas, API de aplicaciones, GUI, configuraciones, aplicaciones, etc.). Aún está en fase alpha (v83, febrero de 2017), pero es posible probarlo o descargarlo e instalarlo.

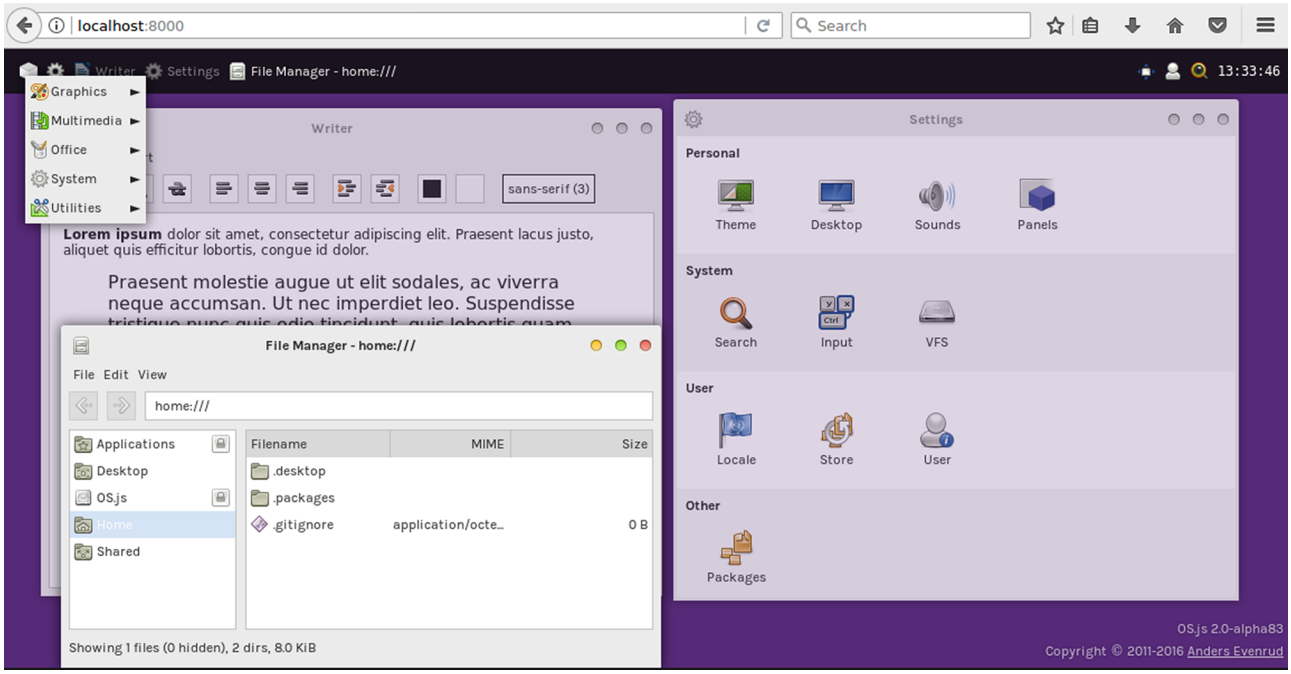
Como está desarrollado en JavaScript, funciona sobre cualquier navegador actual y puede ser instalado sobre todas las plataformas. El entorno está inspirado en los sistemas Linux, pero diseñado para que sea fácil y rápido acceder a las aplicaciones/configuraciones, e incluye un sistema de archivos virtual donde el usuario puede subir y bajar archivos o modificarlos/moverlos a través del *cloud* como, por ejemplo, Google Drive, Dropbox o OneDrive.

Entre las aplicaciones habituales que soporta están: gestor de archivos, reproductor de música y multimedia, editor de imágenes, calculadora, editor de texto, visor de PDF, chat XMPP, Google Mail, Google Contacts y Tetris, entre otras. Dispone además de una API en Javascript muy flexible que permite modificar las funcionalidades actuales o crear otras nuevas, así como también nuevas aplicaciones [Osj].

Para su instalación se ha utilizado una MV KVM con Ubuntu 16.04 y con similares configuraciones a las utilizadas anteriormente. Para su instalación es necesario tener `Node.js`, `npm` y `git` instalados; luego se debe bajar el repositorio, instalar, desplegar y poner en marcha el servidor:

```
apt-get install node.js npm git
git clone https://github.com/os-js/OS.js.git
cd OS.js
npm install --production
node osjs build
node osjs run --target=dist
```

Luego ya se podrá conectar a través de un navegador a la URL *http://localhost:8000*. Para poner en producción se recomienda instalarlo por seguridad detrás de un *reverse-proxy* (Apache o Nginx), como indica la documentación, donde también se puede consultar la integración con discos *cloud*, la conexión https u otros parámetros de configuración. La imagen a continuación muestra el SO y su entorno, y se ejecuta sobre un navegador.



3.5.3. OneEye

EyeOS es una plataforma en *cloud* privada con una interfaz basada en tecnología web que proporciona un escritorio completo para el acceso a archivos, herramientas de gestión de la información personal, herramientas colaborativas y aplicaciones.

Esta plataforma nació como proyecto *open-source* y revolucionó el escenario de los SO virtuales gracias a su funcionalidad/aplicaciones ya la combinación de HTML5, PHP, AJAX y JavaScript para crear un entorno de SO virtual sobre un navegador. La rápida evolución y el interés que despertó hizo que los desarrolladores cambiaran la licencia comercial (a partir de la versión 2.5) y crearan

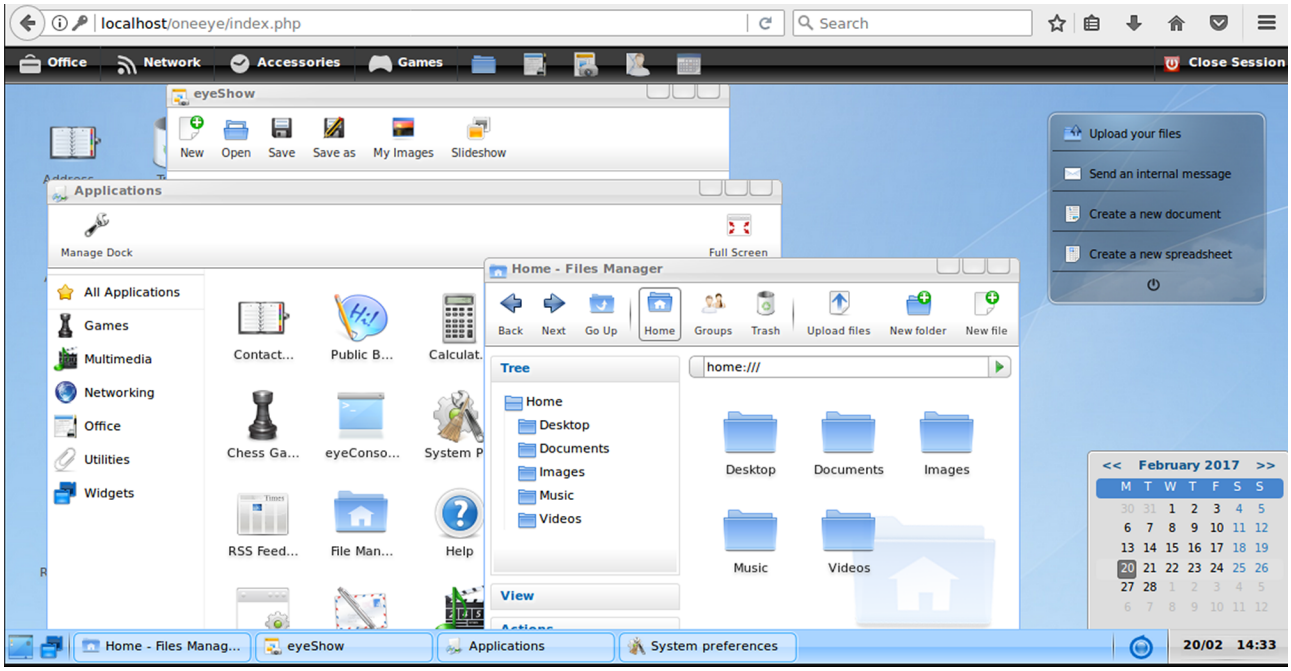
una empresa *startup* (bajo patrocino de Telefónica), pero que en diciembre de 2016 se disolvió por las pérdidas adquiridas (aunque existen diferentes repositorios donde descargarse la última versión *open-source*).

> Un proyecto derivado de EyeOS (a partir de la última versión *open-source*) es OneEye; si bien el proyecto muestra poca actividad en el último tiempo, se ha publicado una nueva versión *preview 0.96* hace poco tiempo.

Su instalación es sumamente fácil, ya que solo requiere un entorno Apache+PHP (con los módulos *mbstring*, *imap*, *sqlite3*) y simplemente se ha de realizar un *unzip* del archivo descargado en un directorio del *DocumentRoot* de Apache, cambiar los privilegios para que Apache (*www-data*) pueda escribir sobre este directorio y llamar a la URL *http://localhost/oneeye* (o el nombre que se le haya dado al directorio).

Si existen errores, el programa de instalación los mostrará para que se solucionen y recargando la página continuará su instalación, a la cual se podrá acceder luego desde la misma URL.

A continuación, se puede ver una imagen de OneEye; como se puede apreciar, dispone de gran funcionalidad y número de aplicaciones (si bien todavía existen algunos pequeños errores no resueltos, aunque se pueden solucionar fácilmente).



Dentro de los entornos propietarios en este ámbito existe un conjunto de soluciones, bajo diferentes tipos de licencia, orientado especialmente hacia la empresa con diferentes filosofías y funcionalidades, y modo de prueba y acce-

so (muchas de ellas tienen una cuenta básica para poder probar el entorno, u otra una versión limitada en aplicaciones o en el tiempo), por ejemplo (en orden alfabético): CloudTop, Oracle Secure Global Desktop, SilveOS (*Silverlight Windows operating system*), UniverseOS, WebTop, ZeroPC, entre otros.

4. Conclusiones

Como se ha podido observar, existe un número elevado de aplicaciones, plataformas, soluciones y entornos dentro del ámbito de PaaS/SaaS; además, es un ámbito con gran dinamismo y oportunidades del cual solo se ha mostrado una pequeña parte. Para muchos autores, la convergencia entre el SaaS y el PaaS es la consecuencia de la evolución de las aplicaciones web hacia entornos donde el usuario, con unos pocos clics, dispone de todo lo necesario para realizar su tarea y que estos tengan diferentes niveles de complejidad en función de sus conocimientos.

Al final, el SaaS emerge de un PaaS y no deja de ser lo mismo que se realizaba tiempo atrás con aplicaciones cliente/servidor y antes con aplicaciones instaladas en el propio ordenador del usuario. Es evidente que, con la tecnología liderada por HTML/JavaScript/PhP/Node.js/Pyhton/Ruby/Go y decenas de otros entonos/lenguajes, se ha mejorado el procesamiento de la información y el acceso a recursos tecnológicos que antes solo estaba disponible para grupos de desarrolladores/usuarios, que contaban con un gran equipo de IT y con grandes inversiones de infraestructura.

Hoy en día, solo se debe saber del entorno tecnológico que se desarrollará la aplicación en un PaaS o de la interfaz de usuario en un SaaS, y no preocuparse por nada más. Hace diez años atrás, por ejemplo, solo aquellas empresas que realizaban la inversión adecuada podían tener acceso a un recurso institucional/corporativo, mientras que hoy, en la mayoría de los casos, no es necesario disponer de infraestructura, ni licencias, ni de personal de IT experimentado para tener servicios equivalentes e integrados que cuestan solo una pequeña parte del costo proporcional y que gestionan enormes volúmenes de información con alta disponibilidad/privacidad/escalabilidad.

No obstante, hay una serie de factores que es necesario considerar para tener la imagen completa de esta tecnología, que son la dependencia de la conectividad a anchos de banda aceptables (teniendo en cuenta además que, para SaaS, se necesitará ancho de banda también en comunicaciones inalámbricas), la disponibilidad de los servicios, la privacidad y seguridad de los datos (que ya no están en los servidores corporativos) y el *data/vendor lock-in*.

Actividades

1. Desplegar dos aplicaciones de interés en PaaS público e instalar dos plataformas PaaS privado; repetir la experiencia de despliegue de la primera parte o similares. Analizar y valorar las posibilidades tanto del PaaS público como privado y extraer conclusiones sobre las prestaciones y elegibilidad para diferentes entornos corporativos/personales.

2. Considerando las diferentes plataformas SaaS, desplegar tres de ellas de diferentes tipos valorando sus posibilidades tanto desde el punto de vista de responsable de IT como desde el punto de vista del usuario final.

3. Analizar para una pyme tecnológica de un determinado rubro cuál sería la combinación perfecta de tecnología PaaS y SaaS para satisfacer las necesidades de negocio, teniendo en cuenta la inversión, las ventajas/desventajas y los costos derivados. Hacer el análisis desde el punto de vista de director de IT, escogiendo el sector de servicio/tecnológico que se desee.

Glosario

add-on Complemento o servicios añadidos generalmente en forma dinámica.

aPaaS *Application platform as a service.*

API Interfaz de programación.

Application Server Software que provee servicios web a los clientes (no confundir con servidor web).

apps Aplicaciones software (término normalmente utilizado para definir aplicaciones móviles).

backend Software/servicio que dan soporte a la ejecución de una aplicación y que se encuentran en servidores remotos.

BD Bases de datos.

buildpacks Aplicaciones encapsuladas para proveer un entorno determinado de desarrollo.

cartridges Entorno preconfigurado de aplicaciones y servicios de OpenShift 2.

CLI (*command-line interface*) Línea de comandos en un terminal.

cloudlock-in Equivalente al *vendor-lock-in*, pero referido a los proveedores de *cloud*.

CMS (*content management system*) Sistema de gestión de contenidos.

commits Acción de registrar un cambio sobre el código de una aplicación (Git).

dashboard Entorno de control de una plataforma/infraestructura/aplicación.

data escrow Acción de guardar una copia de la aplicación de *software-as-a-service* crítico en un tercero (proveedor independiente).

data-lock-in Cierre de acceso a los datos por diferentes motivos (judiciales, económicos, desastres...).

DDoS (*distributed denial of service*) Tipo de ataque sobre un servidor que impide que este pueda prestar su servicio.

DevOps Unión de software *DEVelopment e Information Technology OPerationS* que se utiliza para referirse a una serie de prácticas utilizadas en el desarrollo de software basadas en la colaboración y la comunicación entre los desarrolladores y los responsables de IT.

DNS (*domain name system*) Servicio jerárquico que traduce IP en nombre de dominio y viceversa.

Dynos Contenedor inteligente de Heroku que se ejecuta sobre un entorno de alta fiabilidad.

ephemeral filesystem Sistema de archivos temporal.

ERP (*Enterprise Resource Planning*) Software que permite ala automatización de todos los procesos de una empresa.

fork Bifurcación de un proyecto que continua con su propio desarrollo.

frameworks Entorno de trabajo o desarrollo específico.

freemium (palabra unión de *free* y *premium*) Modelo de monetización de aplicaciones o servicios.

gears Entorno de ejecución en OpenShift V2.

hangouts Vídeo conferencia propiedad de Google.

hosting Servicio que permite tener las aplicaciones ejecutándose sobre servidores de los cuales no se tiene la propiedad.

host-passthrough Parámetro para indicar que la máquina dispone de virtualización anidada activada.

HTTP Protocolo de la WWW.

IDE *Interactive development environment*

IP Dirección de la máquina especificada en notación de puntos (por ejemplo, 192.168.1.1).

logs Registro de la actividad de los servicios/aplicaciones.

LOPD Ley de protección de datos de carácter personal.

middleware Capa de software intermedia entre las aplicaciones y el SO/BD.

mPaaS *Mobile platform as a service*

multitenancy Cuando una aplicación en un servidor atiende a múltiples usuarios/clientes (*tenants*).

MV/VM Máquinas virtuales.

on-line Se aplica cuando un servicio o acción se realiza en forma interactiva.

OpenSaaS Aplicaciones SaaS basado en el código *open-source*.

PaaS Plataforma como servicio.

passwd Palabra clave de autentificación para el acceso a un servicio/plataforma.

plugins Complemento que aporta una funcionalidad extra.

ports Puerto, identificador por donde un servicio escuchará las peticiones de los clientes.

proxy Servidor que actúa como intermediario separando las peticiones de los clientes de los servidores finales.

REST (*representational state transfer*) o **RESTful Web Services** Forma de proveer interoperabilidad entre aplicaciones/servicios sobre internet.

RHEL Sistema operativo Red Hat Enterprise Linux.

root Usuario privilegiado en SO *Nix.

runtime Núcleo básico de ejecución de una aplicación. También se aplica al tiempo durante la ejecución de una aplicación.

SaaS Software como servicio.

sandbox Entorno de desarrollo aislado.

scripting Método de ejecución secuencial generalmente utilizado para la configuración de aplicaciones en modo interpretado por un intérprete de comandos y procesador de un lenguaje.

shell Intérprete de comandos en SO *NIX.

SLA (*service level agreement*) Contrato entre el proveedor de un servicio y el usuario final sobre la calidad y nivel de soporte.

snapshots Resguardo del estado/configuración de un SO/software o servicio que luego es posible recuperar y recomenzar el proceso desde este punto.

SOAP (*simple object access protocol*) Especificación para el intercambio de información estructurada en la implementación de servicio web.

spam Correo no deseado.

SSH (*secure shell*) Protocolo de red criptográfico para la comunicación interactiva entre ordenadores sobre una red insegura (internet). Nombre del comando/servicio que implementa el protocolo.

SSO (*single-sign-on*) Método de autentificación donde el usuario se valida solo una vez para todas las aplicaciones.

stack Conjunto de servicios/aplicaciones configurados, actualizados y optimizados listo para usar.

tags Identificador/marca utilizada para identificar paquetes/servicios/peticiones/...

time-to-market Tiempo entre el diseño de una apps y su salida al mercado.

vendor-lock-in Situación de bloqueo donde el costo de migración a otro proveedor es elevado y se permanece con el actual, aunque los costos sean más altos que los de la competencia.

webDAV (*web distributed authoring and versioning*) Extensión del protocolo HTTP para permitir que un cliente pueda escribir información sobre un servidor web.

webOS Sistemas operativos enlínea o virtualizados, aunque el término está mal utilizado ya que se confunde con el software del mismo nombre.

workflows Secuencia de pasos a realizar con un orden establecido.

YAML Lenguaje texto con tratamiento secuencial de datos utilizado normalmente para archivos de configuración.

Bibliografía

Todos los enlaces han sido visitados en febrero de 2017.

[Ado] AppScale Wiki Documentation. 2017. <https://github.com/AppScale/appscale/wiki>

[Ama] AppScale. Managing Apps, Machines, and Logs. Deploy AppScale. 2017. <https://github.com/AppScale/appscale/wiki/Managing-Apps%2C-Machines%2C-and-Logs> <https://www.appscale.com/get-started/deploy-appscale/>

[Ard] AppScale ReadTheDocs Documentation. <http://appscale.readthedocs.io/en/latest/index.html>

[Are] AppScale Resources & Community. 2017. <https://www.appscale.com/community/>

[Asd] Apache Stratos Documentation. 2017. <https://cwiki.apache.org/confluence/display/STRATOS/Home>

[Ase] AppServer.io Documentation. 2017. <http://appserver.io/products/community-edition.html>

[Awa] Azure Web Application Documentation. 2017. <https://docs.microsoft.com/en-us/azure/app-service-web/>

[C9d] Cloud9 Documentation. 2017. <https://docs.c9.io/docs>

[Cco] Cloudify Cosmo Repository. 2017. <https://github.com/cloudify-cosmo/>

[Cdo] Cloudify Documentation 3.4.0. 2017. <http://docs.getcloudify.org/3.4.0/intro/what-is-cloudify/>

[Cfd] Cloud Foundry Documentation. 2017. <http://docs.cloudfoundry.org/concepts/overview.html>

[Ddo] Dokku Documentation. 2017. <http://dokku.viewdocs.io/dokku/getting-started/installation/>

[Del] El futuro de los servicios Cloud. Informe Deloitte. 2014. <https://www2.deloitte.com/content/dam/Deloitte/es/Documents/tecnologia/Deloitte_ES_Tecnologia_Cloud-Software-como-servicio.pdf> <http://www.cionet.com/Data/files/groups/infografia_cloud-dxd.pdf>

[Dru] Drupal 8 Documentation. 2017. <https://www.drupal.org/docs/8/>

[Dwd] Deis Workflow Documentation. 2017. <https://deis.com/docs/workflow/>

[Ecd] Eclipse Che Documentation. 2017. <https://www.eclipse.org/che/docs/index.html> <https://www.eclipse.org/che/docs/tutorials/multi-machine/index.html>

[Fdo] Flynn. Documentation. 2017. <https://flynn.io/docs/basics>

[Fvi] Flynn. Vision. 2017. <https://flynn.io/about>

[Gae] Google App-Engine Documentation. 2017. <https://cloud.google.com/appengine/docs/about-the-standard-environment>

[Gso] Getting started with OpenShift. S. Pousty; K. Miller. 2014. O'Reilly. ISBN: 978-1-491-90143-4. Free ebook. <https://www.openshift.com/promotions/ebook.html>

[Hcd] Hybrid Cloud for Dummies. Judith Hurwitz; Marcia Kaufman; Fern Halper; Daniel Kirsch. 2012. ISBN: 978-1-118-12719-3.

[Hre] Heroku Reference. 2017. <https://devcenter.heroku.com/categories/reference>

[Mdd] mendix Developer Documentation. 2017. <https://docs.mendix.com/>

[Mqp] Magic Quadrant for Enterprise Application Platform as a Service, Worldwide. 2016. Documento solo bajo suscripción. <https://www.gartner.com/doc/3263917/magic-quadrant-enterprise-application-platform> (imágenes parciales)

[Opb] OpenBravo Wiki. 2017. <http://wiki.openbravo.com/wiki/Main_Page>

[Obg] Getting started with OpenBravo. 2017. <http://wiki.openbravo.com/wiki/Getting_started_with_Openbravo>

[Osj] OS.js JavaScript Cloud/Web Desktop Platform. 2017. <https://www.os-js.org/manual/>

[Opr] OpenBravo resources. 2017. <http://www.openbravo.com/es/resources/>

[Osd] Openshift Documentation. 2017. <https://docs.openshift.com/>

[Oso] OpenShift Origin. 2017. <https://docs.openshift.org/latest/getting_started/index.html>

[Own] OwnCloud Documentation. 2017. <https://doc.owncloud.org/>

[Ppd] Public PAAS for Dummies. L. Miller. 2016. Wiley & Sons. ISBN: 978-1-119-18583-3.

[Tdo] Tsuru Documentation. 2017. <https://docs.tsuru.io/stable/index.html>

[Tre] Top 8 Reasons Why Enterprises Are Passing On PaaS. 2014. Forbes. <http://www.forbes.com/sites/mikekavis/2014/09/15/top-8-reasons-why-enterprises-are-passing-on-paas/#7565134630b1>

[Ucc] Understanding the Cloud Computing Stack: SaaS, PaaS, IaaS. Executive Summary. Rackspace. 2017. <https://support.rackspace.com/white-paper/understanding-the-cloud-computing-stack-saas-paas-iaas/>

[WiP] What is PaaS? What is SaaS? Interoute. 2017. <http://www.interoute.com/what-paas> <http://www.interoute.com/what-saas>

[Wma] Why Mobile App Developers Turn to PaaS. K. Padir. 2014. CMS Wire. <http://www.cmswire.com/cms/mobile-enterprise/why-mobile-app-developers-turn-to-paas-024906.php>