Propuesta para un acceso homogéneo a servicios PaaS en la Nube*

Miguel Barrientos, Jose Carrasco, Javier Cubo, y Ernesto Pimentel

Universidad de Málaga, Dept. Lenguajes y Ciencias de la Computación, España
{mbarrientos,josec,cubo,ernesto}@lcc.uma.es

Resumen. En el ámbito del *Cloud Computing* existen multitud de proveedores ofreciendo plataformas como un servicio, que proporcionan un conjunto de funcionalidades para apoyar el ciclo de vida completo de una aplicación, desde su desarrollo hasta el despliegue en la nube (incluso abordando la monitorización en ocasiones). Aunque la existencia de un número elevado de proveedores aumenta y enriquece la potencia de este tipo de servicios, el inconveniente surge cuando cada uno de ellos define servicios distintos, dando lugar a la problemática de la dependencia del vendedor o *vendor lock-in*. Esta variabilidad complica la selección y uso de los distintos proveedores, ya que cada uno de ellos especifica sus propios conceptos para el modelado de las aplicaciones y de los servicios requeridos durante el despliegue y ejecución de las mismas. En este trabajo, proponemos las bases para la descripción, tanto de las plataformas *cloud* como de las aplicaciones a desplegar, con el fin de generar una capa de homogeneización capaz de abstraer la interfaz de los servicios ofertados por las distintas plataformas, haciendo uso de una API unificada. Esto facilitará el manejo y selección de los servicios de los distintos proveedores. Para ilustrar la idea, se presenta un escenario de aplicación de chat usando servicios de plataformas *cloud*.

Palabras clave: Computación en la Nube, *Cloud Computing*, Plataformas *Cloud*, *Platform-as-a-Service*, Variabilidad,Homegeneidad,Unificación.

1. Introducción

La Computación en la Nube (*Cloud Computing*) [1] es un paradigma que ha experimentado un crecimiento en los últimos años, llegando a ser clave en Internet. Define un nuevo modelo de negocio en el que se exponen y comparten un conjunto ilimitado, desde el punto de vista del usuario, de recursos accesibles a través de la red y que pueden ser aprovisionados bajo demanda en breves espacios de tiempo [2]. Los proveedores *cloud* ofertan sus recursos a través de servicios clasificados en diferentes niveles de abstracción[3]: Infraestructura como Servicio, (*Infrastructure-as-a-Service*, IaaS), Plataforma como Servicio (*Platform-as-a-Service*, PaaS) y Software como Servicio (*Software-as-a-Service*, SaaS).

* Trabajo parcialmente financiado por los proyectos: TIN2012-35669, Ministerio Español de Economía y Competitividad y FEDER; P11-TIC-7659, Junta de Andalucía; FP7-610531 SeaClouds, Unión Europea; y Universidad de Málaga y Campus de Excelencia Internacional Andalucía Tech.

De acuerdo a estas características, son muchos los proveedores que orientan sus servicios en la nube siguiendo este modelo: Google, Amazon, HP, Microsoft, Heroku, OpenShift, etc. En este contexto, los usuarios se encuentran ante un mercado heterogéneo donde se oferta una gran cantidad de servicios de características diversas con inmensas posibilidades de elección. Así, el usuario puede seleccionar los servicios cuyas propiedades y restricciones mejor se adapten a sus necesidades. Sin embargo, como veremos a lo largo de este trabajo, la capacidad de elección de los usuarios se ve sesgada debido a los requisitos impuestos por los distintos proveedores y la complejidad que supone enfrentarse a la heterogeneidad de los mismos. Esto se debe a que cada proveedor ha desarrollado sus plataformas de manera aislada, sin seguir un estándar o metodología común, por lo que cada uno especifica y requiere diferentes mecanismos para el uso correcto de sus servicios.

Aunque la mayoría de los proveedores ofertan una relación de servicios con un perfil similar desde el punto de vista de la funcionalidad, bien es cierto que la falta de comunicación entre proveedores, se traduce en un problema que complica el uso homogéneo y sin distinción de los servicios de los distintos *clouds*. Este problema, recurrente en los distintos niveles de los servicios, implica que para alcanzar un mismo objetivo, usando recursos expuestos por proveedores distintos, es necesario conocer las características y requisitos impuestos por cada uno de ellos. Como hemos mencionado, esta problemática es recurrente en los distintos niveles de abstracción que presenta la nube. No obstante, dependiendo del nivel al que nos enfrentemos podemos encontrar alternativas distintas que permitan mitigar las cuestiones relacionadas con la heterogeneidad de los proveedores. Así, desde el punto de vista del despliegue y gestión de aplicaciones en la nube, las abstracciones más interesantes son en los niveles de IaaS y PaaS.

En el nivel de infraestructura (IaaS) podemos hacer referencia a iniciativas como el estándar de OASIS, TOSCA [4], Ubuntu Juju (`http://juju.ubuntu.com/`), Docker (`https://www.docker.com/`) o Interoperable Cloud [5], con especial relevancia de jclouds [6]. jclouds es una librería que, hasta el momento, ha unificado, bajo una API común, los servicios IaaS de más de 30 proveedores *cloud* [7]. De esta forma, los usuarios pueden trabajar directamente con esta herramienta, haciendo transparente el uso del proveedor utilizado. Por lo tanto, la unificación en el ámbito de IaaS está muy desarrollada, con trabajos recientes estudiando cuestiones como la portabilidad e interoperabilidad entre proveedores [8,9], y el despliegue heterogéneo o multi-despliegue de las aplicaciones *cloud* [10,11,12,13,14]. En este contexto, se sitúa Brooklyn [15], un proyecto de la Fundación Apache que provee un entorno de definición, despliegue y control de aplicaciones a nivel de IaaS basado en jclouds.

Nuestro objetivo se centra en el estudio de una solución a la unificación de servicios a nivel de PaaS, que continúe las líneas mencionadas respecto al nivel de IaaS. La principal idea es proveer mecanismos funcionales para usar de forma indiferente los servicios de las distintas plataformas que podemos encontrar en el mercado, haciendo frente a la problemática del *vendor lock-in* en el ámbito del PaaS. En este artículo, proponemos las bases de nuestro enfoque para alcanzar este objetivo, aproximando los mecanismos necesarios

tanto para la descripción de las aplicaciones *cloud* como para los artefactos y recursos precisos para el despliegue y ejecución de las mismas. Definimos también una primera aproximación de la abstracción necesaria para el uso unificado de los servicios expuestos por los proveedores *cloud*. Para ilustrar nuestro estudio, hemos integrado estas características dentro de Apache Brooklyn aprovechando su entorno de ejecución de aplicaciones y gestión de servicios, de forma que hemos habilitado el despliegue de aplicaciones PaaS usando los servicios que la plataforma Cloud Foundry [16] expone para su plataforma. Usamos Pivotal Web Services [17] (desarrollada por Pivotal [18]) a modo de implementación de la plataforma mencionada.

El resto de este artículo está organizado según la siguiente estructura. La Sección 2 presenta iniciativas y proyectos relacionados que intentan solventar parte de la problemática del *vendor lock-in*, principalmente enfocadas a nivel de PaaS. La Sección 3 describe un ejemplo de motivación para ilustrar la propuesta de este trabajo, así como los objetivos que se pretenden cubrir con nuestro enfoque. La Sección 4 presenta los detalles de la propuesta de homogeneización, incluyendo un análisis de las principales características que definen a los proveedores de PaaS, así como los objetivos abordados en la propuesta. Finalmente, presentamos las conclusiones en la Sección 5 junto al trabajo futuro planeado.

2. Estado del arte

Dentro del ámbito de nuestro trabajo, podemos encontrar varios proyectos, iniciativas y estándares con la intención de mitigar la problemática del *vendor lock-in* y la heterogeneidad de los proveedores. Se asume que es poco realista pretender que los proveedores acuerden una homogeneización mediante el cumplimiento de algún estándar a nivel de PaaS, al menos en la situación actual del mercado, donde la situación de ventaja de algunos proveedores hace difícil una solución que exija el compromiso de estos proveedores.

La estandarización es una de las líneas más activas de la investigación en la Computación en la Nube. Asociaciones como IEEE o OASIS están trabajando en sus propios estándares para apoyar la interoperabilidad y portabilidad entre plataformas mediante una unificación de las mismas. En este sentido, OASIS está desarrollando el estándar CAMP [19], el cual especifica las bases necesarias para realizar una unificación de las API de los proveedores a nivel de PaaS por medio de la abstracción de sus servicios en una interfaz común. Este estándar define un modelado para las aplicaciones y los artefactos que son necesarios para su ejecución, y los PaaS deben adoptar el modelo. Como se explicará a lo largo del artículo, nuestro trabajo aborda la problemática con una propuesta concreta de capa unificada y una implementación. Open Grid Forum Comumunity también está desarrollando su propio estándar, Open Cloud Computing Interface (`http://occi-wg.org/`), para describir una API común a los proveedores *cloud* basada en *drivers* concretos para definir la interacción con los servicios desde un punto de vista agnóstico. La versión actual sólo soporta servicios a nivel de IaaS, pero el trabajo no está finalizado y se espera la interacción con PaaS en próximas versiones. Estos estándares definen una metodología de homogeneización un tanto abierta, y propuesta para ser usada por cada plataforma, lo que dificulta en

ocasiones su adopción, además de que no proporcionan ninguna implementación oficial para utilizar los mecanismos que proponen.

El proyecto EU FP7 Cloud4SOA (`http://www.cloud4soa.eu/`) determina un *framework* de abstracción y unificación de servicios, que intenta solventar las cuestiones de interoperabilidad semánticas que existen en la actualidad a nivel de PaaS (y de IaaS) en los proveedores actuales. Cloud4SOA apoya a los desarrolladores en las fases de despliegue y control del ciclo de vida de las aplicaciones, haciendo transparente la selección del proveedor a utilizar y el uso de sus servicios. Además, proporciona un modelado para el perfil de las aplicaciones, lo que permite extraer la información necesaria para la automatización de su despliegue y gestión. El proyecto 4CaaSt (`http://4caast.morfeo-project.org/`) presenta objetivos genéricos para hacer frente a la heterogeneidad a niveles de IaaS y PaaS. Este proyecto define una plataforma propia para unificar la gestión de servicios PaaS de terceros enfatizando en elasticidad y escalabilidad de las aplicaciones. El proyecto europeo PaaSage (`http://www.paasage.eu/`) también está enfocado principalmente en proveer una plataforma estandarizada y abierta para la interacción y unificación de servicios de plataformas e infraestructuras. Al igual que en proyectos anteriores es necesario que los desarrolladores adopten un lenguaje de modelado para especificar las características de las aplicaciones para posibilitar su despliegue y gestión automática. En estos proyectos, se mantiene la abstracción de las plataformas como una parte interna de su ecosistema, por lo que parece complicado extraer dicha abstracción a modo de librería para facilitar su reutilización. En este punto, nuestro trabajo se enfoca en la construcción de una biblioteca independiente que pueda ser usada en la fase de despliegue de aplicaciones o en la generación de modelos de distribución de aplicaciones.

Hasta ahora hemos visto como el modelado de las aplicaciones conforma una parte de los proyectos mencionados. No obstante, podemos encontrar algunas alternativas independientes para el modelado de las aplicaciones y la especificación de sus características. Apache Stratos (`http://stratos.apache.org/`) es un *framework* que define un formato de empaquetado para aplicaciones *cloud* y sus dependencias. En la actualidad se aplica en el contexto de PaaS, pero se está estudiando su integración a nivel de IaaS dándole soporte mediante jclouds. BuildPack (`http://docs.cloudfoundry.org/buildpacks/`) es un ejemplo interesante, que casi representa un estándar de facto en el ecosistema de PaaS, con varios proveedores que soportan este formato como Heroku o Pivotal (Cloud Foundry). Cartridges (`http://openshift.github.io/documentation/oo_cartridge_developers_guide.html`), de Red Hat, es otro claro ejemplo de empaquetado de aplicaciones que también se está usando en el contexto del PaaS, siendo OpenSifht uno de los proveedores que le da soporte. Las soluciones adoptadas en estos trabajos plantean una interacción compleja con los servicios PaaS utilizados, teniendo que especificar la instalación y configuración de la lógica que conforman estos servicios, mientras que nuestra propuesta propone una solución más sencilla enfocada en la descripción de las aplicaciones y el contexto PaaS (conjunto de servicios) en el que serán desplegadas.

3. Motivación y objetivos

En esta sección, describimos un escenario inicial sencillo usado a lo largo del artículo para ilustrar nuestra propuesta de homogeneización en PaaS, y que pretende poner de relieve los objetivos perseguidos. El ejemplo usado es una Aplicación Chat (*Chat Application*), cuya estructura se muestra en la Figura 1. Dicha aplicación está compuesta por un módulo Web (desarrollado con JSP) al que se conectan los usuarios para intercambiar los mensajes, los cuales persisten en una base de datos (usando MySQL). Este caso de uso tiene un propósito ilustrativo y será ampliado a una aplicación más compleja en trabajos posteriores.

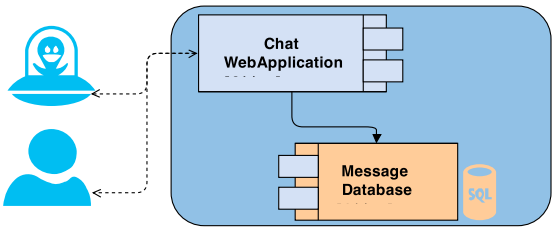

Figura 1: Estructura de la Aplicación de Chat.

Esta aplicación puede ser desplegada usando proveedores PaaS como Pivotal Web Services (Cloud Foundry), Heroku, OpenShift, Google, AWS, etc. Cada proveedor ofrece un rango de tecnologías y servicios (*add-ons*) diferentes para cada tipo de aplicación, cada uno con sus propias restricciones, sus capacidades y su ciclo de vida. Cada uno de los módulos que componen la aplicación también posee sus propiedades y requisitos. Ante la variabilidad de los posibles escenarios de despliegue que podrían tener lugar seleccionando alguno de los servicios disponibles en el mercado, parece interesante desarrollar una abstracción que unifique el uso de estos servicios para facilitar su uso. Además, esto permitiría que los usuarios no tuviesen que conocer los detalles de los mecanismos de conexión y uso de los servicios que se podrían utilizar de forma potencial.

En este trabajo pretendemos hacer frente a los siguientes objetivos o retos:

- **Análisis de los servicios PaaS.** Dada la proliferación de proveedores que ofrecen servicios a nivel de PaaS es necesario conocer y entender la manera en la que estos trabajan, de forma que podamos clasificar el funcionamiento y el uso de estos servicios. Por ejemplo, podríamos clasificar la forma de desplegar una aplicación en un proveedor de acuerdo a los artefactos soportados o los pasos de iniciación y configuración requeridos por los proveedores en los que se podría desplegar la Aplicación Chat.
- **Estudio del alcance de cada PaaS.** Cada proveedor ofrece un número limitado de posibilidades, como las tecnologías soportadas o un ecosistema de add-ons para añadir (o completar) la funcionalidad del sistema desplegado. Por ejemplo, la Aplicación Chat requerirá distintos servicios y add-ons en cada uno de los proveedores que le dan soporte a su despliegue.
- **Homogeneización de interfaces de los proveedores.** En la actualidad, los desarrolladores de aplicaciones necesitan familiarizarse tanto con la interfaz de los proveedores donde quieren desplegar sus aplicaciones, como con el

modelo de aplicación *cloud* que definen. Nuestra aproximación propone solventar esta heterogeneidad en el uso de los servicios, mediante la unificación de sus interfaces en una capa de abstracción a través de la definición de una API común. Esto permitiría aislar aún más las tareas de desarrollo de las operaciones de despliegue y gestión de aplicaciones. En este sentido, se podría utilizar esta homogeneización de los servicios para desplegar la aplicación propuesta, evitando así el bloqueo que implica usar de forma directa los servicios que proveen cada una de las plataformas.

- **Descripción de aplicaciones.** Una aplicación está compuesta por un conjunto de módulos relacionados entre sí, los cuales presentan su propia configuración además de sus requisitos funcionales y no funcionales. Dada la variabilidad de las aplicaciones consideramos que es necesario definir una metodología para la descripción de las mismas. De esta manera, se asegura que desde la abstracción que se propone de los servicios se pueda gestionar una aplicación de acuerdo a su propia estructura y requisitos. En efecto, un nivel de abstracción que homogeneice la diversidad de componentes y módulos, ayudarían en el despliegue y gestión de los mismos de una forma automatizada.

4. Propuesta

En esta sección presentamos, junto a un breve análisis de las características a destacar en el contexto de PaaS (presentando varios proveedores), nuestra propuesta de homogeneización de las plataformas de los distintos proveedores *cloud*. Además, se describen varios proveedores PaaS lo que permitirá entender la variabilidad a la que hacemos referencia, e ilustrará la necesidad de la homogeneización que persigue la actual propuesta.

4.1. Análisis de PaaS

Cloud Foundry. El modelo de despliegue utilizado por Cloud Foundry se centra entorno a aplicaciones Web (*applications*) y servicios (*services*), como se puede ver en la Figura 2. El módulo de aplicación actúa como núcleo principal, al que se le añaden (*binding*) una o más instancias de servicios (*Service Instance*), añadiendo diversas funcionalidades externas al módulo de aplicación. En el caso de Cloud Foundry, estos servicios pueden ser ofertados por terceras partes o creados por el usuario, aunque se comportan de manera idéntica. Además del código de la aplicación, Cloud Foundry dispone de un contexto común a todas las aplicaciones donde se almacenan las variables de entorno necesarias para conectar los distintos módulos.

OpenShift. OpenShift[20] establece como elemento principal de despliegue el concepto de *application*, definida por el código fuente alojado en un repositorio *Git* y un conjunto de variables de entorno. Para una aplicación, el usuario puede definir un número variable de *cartridges* o 'cartuchos' (ver Figura 3). Los cartuchos cumplen una función similar a los servicios en Cloud Foundry, englobando todas aquellas tareas y funcionalidades adicionales que se añaden a la aplicación principal de manera transparente. El proceso de despliegue de una aplicación Openshift se basa en alojar cada uno de los cartuchos definidos

previamente en distintos contenedores Linux seguros, llamados *gears*, que son gestionados por la misma plataforma. Es importante destacar que el conjunto de variables de entorno de una aplicación es común para todos los contenedores, lo que permite realizar las conexiones entre los servicios definidos en distintos cartuchos. Además, los recursos asignados a estos contenedores, así como los servicios ofrecidos por cada uno de los cartuchos añadidos, varían según el plan contratado por el usuario.

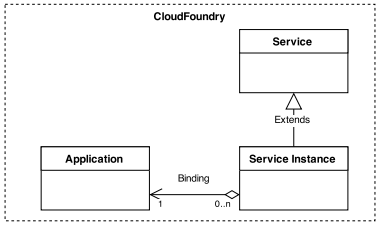

Figura 2: Modelo de aplicación *cloud* en Cloud Foundry.

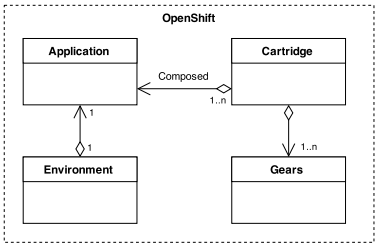

Figura 3: Modelo de aplicación *cloud* en OpenShift.

Heroku. Heroku plantea un modelo claramente centrado en el despliegue de aplicaciones Web [21]. El elemento de despliegue principal es denominado *app* y está determinado por el tipo de lenguaje o entorno en el que se basa la aplicación (Ruby, PHP, Node.js, Java, etc). Como podemos ver en la Figura 4, dentro de una aplicación el usuario puede añadir cualquier número de recursos (*resources*), los cuales se clasifican en dos tipos principales: *Dynos* y *Addons*. Los primeros representan el entorno de ejecución principal, en forma de contenedores Linux donde ejecutar tanto la aplicación web (*Web Dyno*) como cualquier otro tipo de tareas en segundo plano (*Worker Dyno*). Los llamados *addons* representan, al igual que en los proveedores previamente descritos, todos aquellos servicios y funcionalidades externos que pueden añadirse de manera independiente a la aplicación principal.

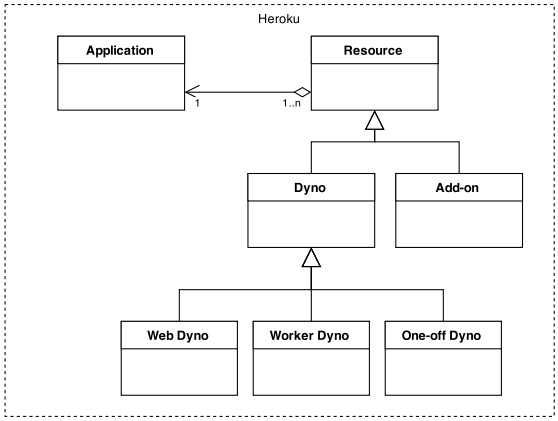

Figura 4: Modelo de aplicación *cloud* en Heroku.

4.2. Abstracción y unificación de servicios PaaS

Como mostramos en la Figura 5, nuestra propuesta pretende abstraer los servicios ofrecidos por los distintos proveedores mediante una biblioteca que actúe como capa de homogeneización y unifique la gestión de los servicios encapsulados.

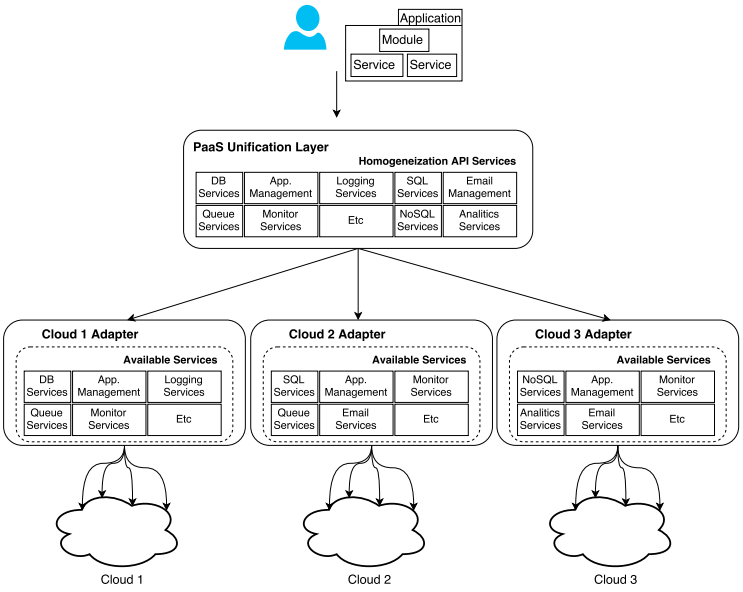

Figura 5: Propuesta de unificación de servicios PaaS.

Para cada proveedor hemos definido un elemento denominado *Adaptador Cloud* (*Cloud Adapter*), que abstraen los servicios PaaS ofrecidos por los distintos proveedores. Observando estos elementos, podemos ver que cada plataforma puede ofrecer servicios diferentes a los que se encuentran en los otros proveedores, de ahí la heterogeneidad en el PaaS.

De esta manera, estos adaptadores están alineados para redefinir y abstraer las interfaces de los servicios del proveedor al que representan, homogeneizando así los servicios con funcionalidades similares alojados en plataformas diferentes bajo una misma API unificada. Mediante estos adaptadores se evita el uso de las interfaces de los servicios fijadas por las plataformas, incurriendo así en el desbloqueo con los proveedores. Además, permiten usar servicios equivalentes de proveedores distintos a través de la API unificada que se ofrece en la Capa de Unificación PaaS (*PaaS Unification Layer*), evitando la necesidad de conocer los detalles de los proveedores finales utilizados. Esta Capa de Unificación permite seleccionar un conjunto de servicios que están descritos en la API de homegeneización y elegir los proveedores en los que se van a ejecutar. La interacción con dicho proveedor se llevará a cabo de forma transparente mediante el adaptador que le corresponda. De esta forma, se proporciona un desbloqueo de la heterogeneidad de las distintas plataformas.

En la Sección 1 describimos cómo el contexto de PaaS ofrece un entorno en el que desplegar y ejecutar tanto aplicaciones como los servicios necesarios durante su ejecución. Así, una aplicación, compuesta por un conjunto de elementos lógicos y relaciones, presenta una gran variabilidad en su construcción y especificación. Hasta ahora hemos presentado cómo la Capa de Unificación puede gestionar servicios de manera uniforme e independiente a los proveedores que los ofrecen. Sin embargo, la diversidad en la descripción de una aplicación añade un grado más de dificultad a la hora de seleccionar los servicios adecuados para el despliegue. En este sentido, proponemos a modo de nomenclatura estructural un modelo para describir las aplicaciones de manera sistemática que facilite su composición y permita a la Capa de Unificación reconocer los componentes que la conforman con el fin de aplicar los servicios necesarios para el despliegue y ejecución de la misma, como aparece en la Figura 6. En esta imagen podemos ver cómo una aplicación está compuesta por módulos interconectados entre sí, los cuales además pueden presentar conexiones con servicios también definidos dentro del modelado de la aplicación. Por supuesto, teniendo en cuenta que estamos tratando de homogeneizar la especificación de las aplicaciones para el uso de la Capa de Unificación, lo lógico es que los servicios antes mencionados apunten directamente a los que dicha Capa de Unificación proporcionan en su API común. En la Figura 7 podemos ver el modelado de la Aplicación de Chat (mencionada en la Sección 3) donde aparecen dos módulos, uno con la lógica Web y otro con las bases de datos, que usan servicios para la interacción con la plataforma (PaaS), JBossService y MySQLService respectivamente. En este ejemplo podemos ver que se están usando servicios de la plataforma Cloud Foundry Web Services a través de la Capa de Unificación de plataformas propuesta.

De este modo, la propuesta persigue abordar el problema de la homogeneización, tanto en el tratamiento de los servicios PaaS, como de la descripción de las aplicaciones que los usan. En la siguiente sección presentamos los requisitos que se deben contemplar a la hora de desarrollar esta propuesta.

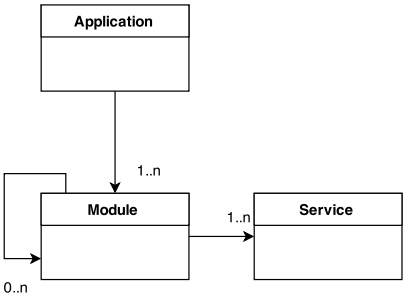

Figura 6: Modelado de una aplicación.

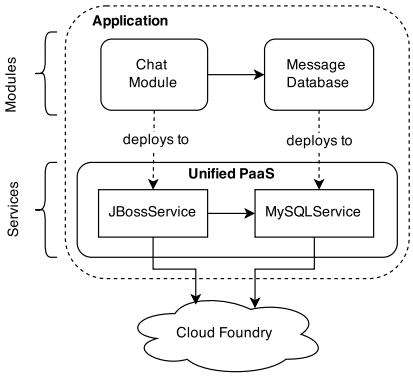

Figura 7: Modelado de la Aplicación Chat usando servicios Cloud Foundry.

4.3. Requisitos de la propuesta

En la construcción de una API común que abstraiga la heterogeneidad de los servicios PaaS de un conjunto de proveedores, debemos tener en cuenta los siguientes requisitos para que la API sea efectiva.

PaaS es una entidad autónoma. A la hora de desarrollar las abstracciones de cada plataforma a la que se le pretende dar soporte, adaptadores, hay que tener en cuenta que cada proveedor especifica sus propios servicios de manera independiente, como ya se ha mencionado a lo largo de las secciones anteriores. Además, es necesario conocer el ciclo de vida de cada uno de estos servicios, ya que también se han desarrollado de forma aislada de acuerdo a las necesidades de cada plataforma. Por ejemplo, los pasos necesarios para desplegar una aplicación en Cloud Foundry y Heroku son distintos, ya que cada proveedor especifica una interacción y comportamiento propio para sus recursos y servicios. Al principio de esta sección hemos presentado un análisis preliminar de la composición general y el funcionamiento de las tres plataformas sobre las que desarrollamos nuestros estudios preliminares: Cloud Foundry, Heroku y OpenShift.

Abstracción con limitaciones. Aunque se pretende alcanzar una abstracción total de los servicios que proveen las distintas plataformas, es complicado ofrecer una separación total de los requisitos impuestos por los propios proveedores. Por ejemplo, en el contexto de la Computación en la Nube el pago por uso (*pay per use*) es uno de los mecanismos más extendidos para realizar el cobro por la utilización de los servicios. Aún así, las tarifas e incluso la nomenclatura de las franjas de precio varían para cada proveedor. En este sentido, aunque nuestra propuesta pretende desbloquear a los clientes de la heterogeneidad del uso de los servicios PaaS, estos siempre deberán estar registrados e identificados en dichas plataformas para usar estos servicios.

Clasificación de servicios PaaS. Ya hemos mencionado que realizar una API común para formalizar los servicios PaaS es el objetivo primordial de este trabajo. Para esto, una vez conocido cómo funcionan cada uno de los proveedores con los que se pretende trabajar, es necesario identificar y clasificar todos los servicios que presenten funcionalidades similares a lo largo de todos ellos. De esta forma, se podrán agrupar de acuerdo a sus características con el fin de especificar los servicios que serán abstraídos y de qué manera. Por ejemplo, una clasificación plausible podría ser la de todos los servicios que permitan desplegar aplicaciones Web según el lenguaje, o la de los servicios de bases de datos, clasificados por el tipo de sistema gestor. Siguiendo el ejemplo propuesto en la Sección 3, se necesitaría un módulo de aplicación Web Java y un servicio de base de datos MySQL.

Redefinición de los servicios del adaptador de los proveedores. Identificados los servicios que van a ser incluidos en el adaptador que abstrae cada proveedor, es necesario formular una API que *(i)* abstraiga el uso y comportamiento de todos los servicios que se pretenden homogeneizar, y *(ii)* sea compatible con la interfaz definida por el proveedor de cada uno de los servicios que va a representar. Hay que tener en cuenta que la diversidad en la definición y uso podría impedir que algún servicio de un proveedor en concreto pueda ser

equiparado al resto con los que comparte la misma clasificación, por lo que no sería incluido en la capa de homogeneización para dicho proveedor. Por supuesto, este servicio podría ser representado con una API propia en el adaptador de la plataforma que los contiene. Por ejemplo, se podrían unificar los servicios de despliegue de aplicaciones Java de Cloud Foundry, Heroku y OpenShift bajo una misma interfaz común que le dé cobertura a todas las funciones expuestas para este contexto (Java) en los tres proveedores mencionados.

Construcción de la Capa de Unificación. Una vez realizados los adaptadores de los proveedores contaríamos con una representación similar para las plataformas de los mismos, por lo que la integración de estos elementos darían como resultado una API común que homogeneizaría los servicios de los distintos proveedores PaaS, componiendo así la Capa de Unificación propuesta en este trabajo, tal y como se describió en la Figura 5 de la subsección previa. Esta capa será la encargada de recibir una aplicación descrita de acuerdo al esquema propuesto en la Sección 4.2. A continuación se seleccionará el adaptador correspondiente al proveedor final para usar sus servicios de forma transparente con objeto de llevar a cabo el despliegue y puesta en marcha de la aplicación *cloud* sobre la plataforma indicada.

Aunque con esta propuesta asentamos las bases de nuestro trabajo de homogeneización y realizamos una análisis de los aspectos más relevantes de cara a llevar a cabo esta tarea, en la siguiente sección presentamos un prototipo que hemos desarrollado, a modo de aproximación basada en Apache Brooklyn, con el que hemos llevado a cabo una primera abstracción de los servicios de Cloud Foundry y del modelado de las aplicaciones.

4.4. Primera aproximación con Apache Brooklyn

Este primer prototipo se ha desarrollado, como prueba de conceptos, dentro del contexto de Apache Brooklyn, un proyecto de código abierto que está siendo impulsado por la Fundación Apache. Este proyecto ofrece un entorno de despliegue de aplicaciones *cloud* sobre un amplio conjunto de proveedores a nivel de infraestructura (IaaS) a través de la librería jclouds (otro proyecto de Apache), la cual ofrece una abstracción y homogeneización de la API de muchos proveedores, tal y como se describió en la Sección 1. Además, este proyecto ofrece un marco de trabajo con el que se pueden describir aplicaciones e indicar sobre qué proveedores se quiere trabajar, información que es analizada para llevar a cabo el proceso de despliegue de las aplicaciones. No obstante, el alcance de Brooklyn no termina aquí, ya que también ofrece un entorno para la monitorización de las aplicaciones previamente desplegadas y la posibilidad de gestionarlas de acuerdo al comportamiento que presenten en tiempo de ejecución. Por tanto, Brooklyn ofrece un escenario de partida estable y con las características suficientes como para desarrollar nuestra propuestas. Además, presenta esfuerzos para la alineación con el estándar CAMP de OASIS, el cual describe una metodología para la unificación de servicios *cloud* a nivel de PaaS bajo una API común y homogeneizada. Sin embargo, Brooklyn está focalizado en la gestión de servicios IaaS mediante jclouds y aún no está aplicando al desarrollo y abstracción de servicios dentro del ámbito de PaaS, lo que también justifica nuestro trabajo.

En esta aproximación hemos realizado una primera iteración para abstraer la plataforma de Cloud Foundry implementada por Pivotal Web Services (desarrollada por Pivotal). Con este objetivo hemos añadido varios elementos a Brooklyn, denominados entidad (*Entity*) . Hemos usado una entidad para representar módulos de aplicaciones Java que van a ser desplegadas en un JBoss encapsulado como servicio PaaS, *JBossEntity*, y otra para representar una base de datos MySQL alojada también en una plataforma, *MySQLService*. En la Figura 8 se representa cómo la Aplicación de Chat (Sección 3) se describe como una Aplicación de Brooklyn mediante estas entidades; para la lógica Web se usa JBossEntity, mientras que la base de datos viene representada por MySQLEntity. Además, tal y como se indicó en la descripción de la aplicación, el módulo Java mantiene una dependencia con la base de datos, asegurando así la persistencia de los mensajes.

En esta Figura 8 también está representada la primera versión de la capa de homogeneización de PaaS (*Unified PaaS*) que proporciona los servicios JBossService y MySQLService, ambos usados por las entidades JBossEntity y MySQLEntity, respectivamente, para desplegar y relacionar los módulos de la aplicación a los que representan. Además, como se indicó anteriormente, hemos desarrollado un adaptador de Cloud Foundry para conectar los servicios de la Capa de Unificación, JBossService y MySQLService, con los de la plataforma, JBossCloudFoundry y MySQLCloudFoundry. Aunque en esta imagen no lo hemos indicado de una forma explícita, este adaptador permite llevar a cabo el inicio de sesión en Pivotal con una cuenta registrada para hacer uso de los servicios de la plataforma.

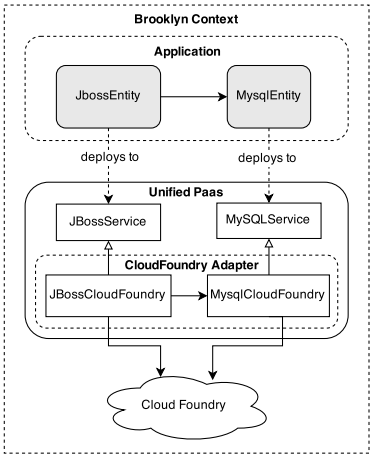

Figura 8: Integración de servicios PaaS en Brooklyn.

Siguiendo el modelo de CAMP, Brooklyn permite definir las aplicaciones mediante un archivo YAML en el que se especifican cada uno de los módulos que la componen, además de cómo están relacionados. En el Cuadro 1.1 presentamos la descripción de la Aplicación de Chat mediante esta metodología. Se declaran los módulos principales de la aplicación usando las entidades descritas previamente, cada uno define su propia configuración y los módulos de los que depende. Por

ejemplo, *DATABASE_ENDPOINT* describe la variable de entorno del módulo web de la aplicación mediante la cual se establecerá la conexión con la base de datos. También podemos encontrar otros parámetros que definen la ubicación de los artefactos que implementan la aplicación, como *datastore.creation.script* o *app_git_repo_url*, o los puertos de conexión soportados por el sistema una vez ha sido desplegado (*http_port*).

```
name: Chat Application
location: cloud-foundry
services:
- serviceType: MysqlEntity
  name: Message Database
  id: messageDatabase
  brooklyn.config:
    datastore.creation.script: create-database.sql
- serviceType: JbossEntity
  version: 7.0
  brooklyn.config:
    http_port: 8080
    app_git_repo_url: chat-application-repo.git
    environment:
      DATABASE_ENDPOINT: messageDatabase.attributeWhenReady("host.
          address")
```

Cuadro 1.1: Descripción de la Aplicación Chat mediante un YAML.

5. Conclusiones y trabajo futuro

En este trabajo, hemos propuesto una homogeneización de los servicios en el contexto PaaS de la Computación en la Nube. Definimos una abstracción de los proveedores mediante una Capa de Unificación basada en una API común que pretende ofrecer una alternativa plausible para hacer frente a la problemática del *vendor lock-in*. Asentamos los objetivos de nuestro trabajo e indicamos las bases y cuestiones más relevantes para la evolución de nuestra propuesta. A modo de aproximación, hemos desarrollado un primer adaptador para abstraer y encapsular los servicios de la plataforma Cloud Foundry, realizando una integración del mismo dentro de Apache Brooklyn. De esta forma, abordamos los principales objetivos planteados, que engloban: analizar los proveedores PaaS, identificar los servicios que van a ser abstraídos y construir un adaptador para la plataforma.

Respecto al trabajo futuro, consideramos interesante estudiar la plataforma de otros proveedores y analizar sus servicios para desarrollar nuevos adaptadores. Dichos adaptadores deben mantenerse alineados a una API común, de cara a la construcción de la Capa de Unificación propuesta. Una vez que estos elementos sean desarrollados, estudiaremos su integración dentro de un gestor de aplicaciones como Brooklyn para comprobar su versatilidad y usabilidad.

Por último, cabe mencionar que el objetivo primordial del trabajo propuesto es el proporcionar un adaptador genérico que ofrezca la Capa de Unificación como una librería independiente, englobando los diferentes servicios en el ámbito de PaaS.

Referencias

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53** (2010) 50–58
2. Mell, P.M., Grance, T.: Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States (2011)
3. Youseff, L., Butrico, M., Silva, D.D.: Towards a unified ontology of cloud computing. In: in Proc. of the Grid Computing Environments Workshop (GCE08. Volume 1., IEEE (2008) 1–10
4. OASIS: TOSCA 1.0 (Topology and Orchestration Specification for Cloud Applications), Version 1.0. `http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf` (2012)
5. DMTF: Interoperable Cloud. `http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0101_1.0.0.pdf` (2013)
6. Fundación Apache: Proyecto jclouds. `https://jclouds.apache.org/` (2014)
7. Fundación Apache: Proveedores *cloud* soportados por jclouds. `https://jclouds.apache.org/reference/providers/` (2014)
8. Leymann, F., Fehling, C., Mietzner, R., Nowak, A., Dustdar, S.: Moving applications to the cloud: an approach based on application model enrichment. International Journal of Cooperative Information Systems **20** (2011) 307–356
9. Petcu, D.: Portability and interoperability between clouds: challenges and case study. In: Towards a Service-Based Internet. Volume 6994. Springer (2011) 62–74
10. Nicolae, B., Cappello, F., Antoniu, G.: Optimizing multi-deployment on clouds by means of self-adaptive prefetching. In: Euro-Par 2011 Parallel Processing. Volume 6852. Springer (2011) 503–513
11. Yang, X., Zhang, H.: Cloud computing and soa convergence research. In: Computational Intelligence and Design (ISCID), 2012 Fifth International Symposium on. Volume 1., IEEE (2012) 330–335
12. Carrasco, J., Cubo, J., Pimentel, E.: Propuesta de metodología de despliegue de aplicaciones en nubes heterogéneas con tosca. In: Proceedings of 19th Spanish Conference on Software Engineering and Databases (JISBD), September 16-19, 2014, Cádiz (Spain). Volume 1. (2014) 321–334
13. Brogi, A., Ibrahim, A., Soldani, J., Carrasco, J., Cubo, J., Pimentel, E., D'Andria, F.: Seaclouds: a european project on seamless management of multi-cloud applications. ACM SIGSOFT Software Engineering Notes **39** (2014) 1–4
14. Carrasco, J., Cubo, J., Pimentel, E.: Towards a flexible deployment of multi-cloud applications based on tosca and camp. In: Communications in Computer and Information Science. Volume 508., Springer (2015) 278–286
15. CloudSoft: Brooklyn project. `https://brooklyn.incubator.apache.org/` (2012)
16. Cloud Foundry: Página de Cloud Foundry. `http://cloudfoundry.org/` (2015)
17. Pivotal: Página de Pivotal Web Services. `https://run.pivotal.io/` (2012)
18. Pivotal: Página de Pivotal Web Services. `https://pivotal.io/` (2012)
19. OASIS: CAMP 1.1 (Cloud Application Management for Platforms), V1.1. `http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html/` (2012)
20. OpenShift: Understanding OpenShift. `https://www.openshift.com/products/architecture` (2015)
21. Heroku: Reference documentation, Heroku Dev Center. `https://devcenter.heroku.com/categories/reference` (2015)