

UNIVERSIDAD DE LA REPÚBLICA

FACULTAD DE INGENIERÍA

INFORME DE PROYECTO DE GRADO

Implementación de Referencia de un Gestor de
Plataformas de Middleware para la Integración Inter-
organizacional

Autores:

Jimena Pintos

Gaston Salgado

Martín Inzaurrealde

Cliente:

Laura González

Tutor:

Guzmán Llambías

07 de Agosto del 2017

Durante las últimas décadas, las organizaciones han evolucionado, pasando de ser entidades estructuradas jerárquicamente, estáticas, monolíticas y centralizadas, a ser entidades distribuidas, dinámicas y federadas. Los procesos de negocio, en estos nuevos contextos inter-organizacionales, se llevan a cabo cada vez más integrando y componiendo servicios de negocio ofrecidos individualmente por cada organización.

Por otro lado, la integración de servicios en estos contextos se apoya cada vez más en una infraestructura especializada de hardware y software, comúnmente denominada Plataforma de Integración, la cual provee una capa de procesamiento intermedio a través de la cual las distintas organizaciones proveen y consumen servicios.

Tanto en la industria como en la academia, se han propuesto diferentes tipos de plataformas de integración para dar soporte a la colaboración entre organizaciones. Muchas de estas propuestas se enfocan en dominios específicos como gobierno electrónico o salud. Asimismo, el surgimiento de cloud computing ha dado lugar a soluciones que ofrecen plataformas de integración como servicio (Integration Platform as a Service, iPaaS). Las capacidades de integración en estas soluciones se implementan en general como mecanismos internos de productos de middleware. Esto ocasiona que el costo de desarrollar estas plataformas "desde cero" sea una barrera importante para disponibilizar estas soluciones.

Para abordar esta problemática, existen trabajos que proponen una arquitectura de referencia para Plataformas de Integración que den soporte a la Integración Inter-organizacional. Estas plataformas extienden las soluciones de integración tradicionales con funcionalidades avanzadas y para dar soporte a interacciones inter-organizacionales.

Este proyecto apunta a partir de un modelo teórico determinar la viabilidad de llevar a cabo la construcción de una plataforma de gestión de múltiples plataformas de integración. Para esto, se realiza un análisis del mercado de productos de middleware y de plataformas de integración ofrecidas como servicios, para luego diseñar una arquitectura tomando como base una arquitectura de referencia.

Luego se da paso al diseño de la plataforma y a la identificación de los requerimientos que esta plataforma debe cumplir. En base a ellos, se identifica un alcance mínimo que cubra las principales funcionalidades del sistema a construir, de modo de probar la viabilidad de la construcción de este tipo de software con las características planteadas, para finalmente, dar paso a la implementación del prototipo que cubre este alcance.

Palabras clave

Plataforma de integración, Openshift, Colaboración inter-organizacional, Integración de componentes.

Índice General

Capítulo 1: Introducción.....	6
1.1 Objetivos.....	6
1.2 Aportes.....	7
1.3 Organización del documento.....	7
Capítulo 2: Marco Conceptual.....	9
2.1 Cloud Computing.....	9
2.2 APIs.....	12
2.3 Servicios.....	12
2.4 API Managers.....	14
2.5 ESB.....	16
2.6 Containers Linux - Virtualización a nivel de Kernel.....	19
Capítulo 3: Análisis.....	27
3.1 Descripción del artículo.....	27
3.2 Extensión del artículo.....	29
3.3 Requerimientos implementados.....	34
3.4 Alcance de proyecto.....	36
3.5 Trabajo relacionado.....	36
Capítulo 4: Diseño de la solución.....	38
4.1 Descripción general de la solución propuesta.....	38
4.2 Vista de negocio.....	39
4.3 Arquitectura de la solución.....	39
4.4 Consideraciones de diseño.....	44
Capítulo 5: Implementación.....	46
5.1 Tecnologías utilizadas.....	46
5.2 Herramientas de desarrollo.....	50
5.3 Dificultades y limitaciones encontradas.....	50
5.4 Testing.....	51
Capítulo 6: Gestión del proyecto.....	58
6.1 Organización.....	58
6.2 Planificación del proyecto.....	58
6.3 Etapas del proyecto.....	60
Capítulo 7: Trabajo futuro.....	62
7.1 Mejoras.....	62
7.2 Pruebas a realizar.....	63
Capítulo 8: Conclusiones.....	65
Referencias.....	67
Glosario.....	70
Anexo I: Análisis de productos de API Management.....	72
1 Microsoft Azure API Management.....	72

2 WSO2 API Manager.....	73
Anexo II:Arquitectura básica API Manager.....	75
1 Introducción.....	75
2 Arquitectura básica.....	75
Anexo III:Análisis de productos ESB.....	77
1 Introducción.....	77
2 Mule ESB.....	77
3 Apache ServiceMix.....	78
4 JBoss Fuse.....	79
5 WSO2 ESB.....	80
Anexo IV:Análisis comparativo de plataformas.....	82
1 Introducción.....	82
2 ESB.....	82
3 API Management.....	84
Anexo V:Casos de uso críticos.....	87
1 Introducción.....	87
2 Detalle de los casos de uso.....	87
Anexo VI:Arquitectura completa de la solución.....	96
1 Introducción.....	96
2 Vista lógica.....	96
3 Vista de desarrollo.....	98
4 Vista de procesos.....	104
5 Vista física.....	113
6 Vista de escenarios.....	115
Anexo VII:Interfaz REST Conectores.....	121
1 Introducción.....	121
2 Especificación.....	121

Capítulo 1: Introducción

En el presente trabajo se describe el proyecto académico “*Implementación de Referencia de un Gestor de Plataformas de Middleware para la Integración Inter-organizacional*” en el cual se lleva a cabo el análisis e implementación a la solución conceptual planteada en el artículo “*A reference architecture for integration platforms supporting cross-organizational collaboration*” realizado por Laura González y Raúl Ruggia.[1]

Hoy en día se ha tornado una necesidad brindar soluciones de integración a las organizaciones debido a la proliferación de sistemas de software los cuales deben adaptarse con flexibilidad y rapidez a la evolución de los objetivos y requerimientos del negocio. Las organizaciones han estado evolucionando en las últimas décadas pasando de estructuras jerárquicamente organizadas, sistemas monolíticos y centralizados a sistemas altamente distribuidos. Para realizar sistemas dinámicos y autoadaptativos, el concepto de de arquitectura orientada a servicios (SOA) ha surgido como un mecanismo de abstracción adecuado. Estos sistemas a su vez deben lograr comunicarse entre sí y es ahí donde aparecen las plataformas de integración. Una plataforma de integración es una infraestructura especializadas que incluye una capa de procesamiento intermedio para facilitar la provisión e invocación de servicios de software interoperables. [2]

Se han propuesto diferentes tipos de plataformas de integración para dar soporte a la colaboración entre organizaciones. El surgimiento de Cloud Computing ha dado lugar a soluciones que ofrecen plataformas de integración como servicio (Integration Platform as a Service, iPaaS), los cuales son considerados la evolución de los middlewares del tipo Enterprise Service Bus(ESB) hacia una solución en la nube. Dichas plataformas han sido adoptadas por los usuarios debido a que permiten integrar una amplia variedad de sistemas utilizados por las organizaciones, incluyendo Software as a Service(SaaS), bases de datos, sistemas desarrollados internamente en la organización, entre otros. Estas soluciones han sido concebidas para abordar problemáticas específicas, por lo que su reutilización en diferentes dominios es limitada. [1]

En particular en [1] se propone y especifica una arquitectura de referencia para plataformas de integración que brinden una solución a la integración inter-organizacional. Dichas plataformas se basan en las soluciones de integración tradicionales con el agregado de funcionalidades que faciliten la interacción y colaboración entre las organizaciones.

La motivación principal del proyecto es proveer a las organizaciones un gestor de plataformas que facilite las necesidades de integración que plantean hoy en día las organizaciones siguiendo el modelo de referencia.

1.1 Objetivos

El objetivo general del proyecto es desarrollar un gestor de plataformas de middleware que permita la colaboración inter-organizacional, basado en el modelo conceptual planteado en [1].

Para lograr esto se plantean una serie de objetivos específicos:

- Analizar diferentes productos de tipo middleware y plataformas como servicio existentes en el mercado para luego seleccionar aquellos que brinden facilidades en su uso y mejor se ajusten a los requerimientos del proyecto.
- Diseñar una solución que permita la gestión de las plataformas, basándose y extendiendo los conceptos que se describen en el modelo presentado en [1].
- Implementar un prototipo que de soporte a dicha solución tomando como referencia el documento de la Arquitectura de Referencia para Plataformas de Integración presentado en [1]

1.2 Aportes

Los principales aportes del proyecto son:

- Análisis de productos ESBs y API Managers, que pueden servir como guía para posteriores proyectos que requieran realizar una comparativa entre distintos productos.
- Diseño e implementación de componentes para la plataforma que pueden ser reutilizados en otros proyectos. Por ejemplo en el caso de JBoss Fuse no contaba con una API para utilizar sus funcionalidades por lo que se implementó una API. Dicha API podría ser reutilizada en otros proyectos similares.
- Implementación de referencia de una Plataforma de integración inter-organizacional tomando como base la arquitectura de referencia.
- Diseño de una arquitectura extensible a nuevos componentes y funcionalidades que pueden ser implementadas a futuro, para continuar con el desarrollo del producto.

1.3 Organización del documento

El contenido del documento se organiza en capítulos de la siguiente manera.

En el capítulo 2 se desarrollan los conceptos y definiciones que son necesarios para comprender el resto del documento.

En el capítulo 3 se presenta un análisis de la problemática planteada, se describen los requerimientos del proyecto y se establece el alcance del mismo.

A partir de este análisis, en el capítulo 4 se elabora el diseño de la solución. En dicho capítulo se describen los principales casos de usos.

En el capítulo 5 se describe la implementación de los principales componentes de la plataforma, así como las tecnologías utilizadas para el desarrollo de los mismos. Además se mencionan algunas de las problemáticas que se presentaron durante esta etapa, finalizando con una descripción de las pruebas a las que fue sometida la implementación realizada de la plataforma.

En el capítulo 6 se describe cómo fue llevada a cabo la gestión del proyecto para su correcto desarrollo.

El capítulo 7 contiene el trabajo a futuro donde se mencionan posibles extensiones al proyecto.

Por último en el capítulo 8 se presentan las conclusiones del proyecto, resumiendo el trabajo realizado y describiendo sus contribuciones.

Capítulo 2: Marco Conceptual

En este capítulo se presentan los conceptos más relevantes manejados en el proyecto. En primer lugar se presenta una descripción general de cloud computing continuando con una breve introducción sobre algunos de los modelos de servicios como son IaaS, PaaS, iPaaS y SaaS. A continuación se brinda una definición de API y Servicio. Posteriormente, se presentan los conceptos de API Manager y Enterprise Service Bus (ESB) describiendo sus principales características así como algunos de los productos existentes en el mercado. Finalmente, se introducen los conceptos de Virtual Machine (VMs) y Container Linux describiendo los beneficios que brinda cada tecnología y realizando una comparación entre ambos, para luego dar una referencia a OpenShift, producto esencial en el desarrollo de la solución.

2.1 Cloud Computing

2.1.1 Definición

IBM define cloud computing como una solución integral que entrega tecnologías de la información como un servicio. Especifica que en la nube, los computadores están configurados para trabajar juntos y las diferentes aplicaciones utilizan el poder computacional colectivo, como si estuvieran corriendo sobre un mismo sistema. [3]

La asignación de recursos se realiza a demanda, lo que facilita y optimiza el uso de los recursos acumulados en el sistema de la nube, negando la necesidad de asignar hardware específico a una determinada tarea.

Las principales ventajas de la computación en la nube con respecto a la computación clásica son [3]:

- **Reducción de costos:** tanto operativos como de capital, ya que los recursos son adquiridos sólo cuando son necesarios, y se pagan únicamente cuando son utilizados.
- **Liberación de personal:** ya que en lugar de tener que realizar mantenimiento de hardware y software, se puede enfocar en la entrega de valor a través del software y hardware provisto.
- **Escalabilidad robusta:** el esquema de computación en la nube permite un escalamiento inmediato, tanto horizontal como vertical, en cualquier momento sin un compromiso a largo plazo.

Principalmente se distinguen tres esquemas de computación en la nube [3]:

- **Público:** las nubes públicas son suministradas y administradas por organizaciones que proveen el servicio de nube por internet, donde las organizaciones cliente pueden publicar sus aplicaciones (y/o contenidos), y son accesibles por todos los

usuarios. Son aquellas donde los clientes de los servicios son externos al proveedor del servicio de nube.

- **Privado:** las nubes privadas existen dentro de firewalls de las compañías a las que pertenecen, y son gestionadas por la propia organización.
- **Híbrido:** las nubes híbridas se refieren a la provisión, uso y administración de servicios coordinados basados en políticas a través de una mezcla de servicios en la nube externos (públicos) e internos (privados).

2.1.2 Modelos de servicios

El concepto de nube está basado en capas, cada una proporcionando un nivel distinto de funcionalidad. Las capas serían las siguientes: Infraestructura, Plataforma y Aplicación. Cada una de ellas determina un nivel de servicio que se puede ofrecer.

Los modelos de servicios establecen diferentes alternativas de comercialización de los recursos de cloud computing. Entre ellos se encuentran [4]:

2.1.2.1 Infrastructure as a Service (IaaS)

Es un modelo en el cual la infraestructura básica de cómputo (servidores, software y equipamiento de red) es gestionada por el proveedor como un servicio bajo demanda, en el cual se pueden crear entornos para desarrollar ejecutar o probar aplicaciones. El objetivo principal de este modelo es evitar la compra de recursos por parte de los suscriptores, ya que el proveedor ofrece estos recursos como objetivos virtuales accesibles a través de una interfaz de servicio.

Algunos de los principales proveedores de IaaS en el mercado son:

- Amazon Web Services [5]
- Microsoft Windows Azure [6]
- Google Compute Engine [7]
- IBM SmartCloud Enterprise [8]
- HP Enterprise Converged Infrastructure [9]

2.1.2.2 Platform as a Service (PaaS)

El servicio se entrega como bajo demanda, desplegándose el entorno (hardware y software) necesario para ello. De esta forma, se reducen los costos y la complejidad de la compra, el mantenimiento, el almacenamiento y el control del hardware y el software que componen la plataforma.

El suscriptor del servicio tiene control parcial sobre las aplicaciones y la configuración del entorno ya que la instalación de los entornos dependerá de la infraestructura que el proveedor del servicio haya desplegado.

Los servicios PaaS pueden consistir en funcionalidades preconfiguradas a las que los clientes puedan suscribirse, eligiendo las funciones que deseen incluir para resolver sus necesidades y descartando aquellas que no necesiten.

Los principales proveedores de PaaS en el mercado son:

- Microsoft Azure [10]
- CA Technologies [11]
- IBM Bluemix [12]
- MuleSoft CloudHub [13]
- RedHat Openshift [14]

2.1.2.3 Software as a Service (SaaS)

Consiste en un despliegue de software en el cual las aplicaciones y los recursos computacionales se han diseñado para ser ofrecidos como servicios de funcionamiento bajo demanda.

Los clientes no controlan la infraestructura subyacente como ser: recursos de red, servidores, sistema operativo, almacenamiento, etc.

El proveedor podrá brindar la posibilidad que los usuarios configuren algunos aspectos de las aplicaciones.

2.1.2.4 Integration Platform as a Service (iPaaS)

Es un servicio cloud que entra en la categoría de PaaS (Plataforma como servicio) que provee capacidades para implementar proyectos de integración que involucren cualquier combinación on-premise o cloud endpoints dentro de una organización o entre organizaciones. [15]

Permiten desarrollar, desplegar, ejecutar, gestionar y monitorear flujos de integración que conectan estos endpoints.

Los iPaaS nacen como una necesidad del mercado para brindar soluciones de integración del tipo middleware de forma de integrar diversos sistemas, alojados tanto en la nube como en servidores dentro de las organizaciones.

Las principales características de un iPaaS son [15]:

- Permiten la interconexión de sistemas alojados en la nube y en organizaciones.
- Permiten la orquestación de distintos servicios para exponerlos como servicios nuevos.
- Brindan mecanismos de seguridad para proteger tanto la integridad como la confidencialidad de los datos, tanto de los datos que procesa como los metadatos almacenados.
- Soporte para una gran cantidad de conectores/protocolos para la integración.
- Soporte para el mapeo de datos y transformación de mensajes.
- Brindan una interfaz de administración que permite realizar auditorías sobre los cambios realizados en las integraciones, y sobre ejecuciones de las mismas.

2.1.2.4.1 Proveedores de iPaaS en el mercado

Hoy en día existen diferentes implementaciones de iPaaS provistas por diferentes empresas tales como [15]:

- Dell Boomi [16]
- Informatica Cloud [17]
- IBM WebSphere Cast Iron [18]

2.2 APIs

En esta sección se brinda una definición del concepto API y cómo fue evolucionando dicho concepto.

2.2.1 Definición

Una API (siglas de Application Programming Interface, o Interfaz de Programación de Aplicaciones en español) es un conjunto de definiciones, operaciones, protocolos y herramientas que permiten construir software a partir de la interacción con otros elementos de software [19]. Dicho de otro modo, las APIs definen el modo en el que los diferentes componentes de software interactúan, permitiendo así a los programadores generar nuevos componentes de software que interactúen con los ya existentes.

2.2.2 Desarrollo del concepto API

El concepto de API está sufriendo una transformación en base a la cantidad de organizaciones que cada vez más están publicando sus APIs a desarrolladores externos. [20]

Hoy en día, las APIs se relacionan principalmente con aquellas interfaces de servicios ofrecidos a través de la Web, de manera pública (generalmente manejando la seguridad a través de los protocolos estándares de las diferentes implementaciones de las APIs) y que sirven de puerta de entrada para el uso de los diferentes servicios que una organización posee.

Es decir, que el concepto de API que hoy se está generalizando tiende a estar acotado a las APIs de tipo REST [21], donde los servicios son ofrecidos a través del servicios web RESTful, aunque en algunos casos, también se extienden a APIs SOAP [22], que son principalmente, servicios web que utilizan el protocolo SOAP. [23]

Microsoft [24] define una API como un conjunto de operaciones que están disponibles para ser utilizadas por un programador. También menciona que cada API tiene una referencia al servicio de back-end que implementa las operaciones definidas en la API.

En el White Paper Cloud-based API Management: Harnessing the Power of APIs [25] se hace referencia a esta distinción entre REST y SOAP, indicando que el protocolo SOAP con el tipo de datos XML es un estándar que se puede decir hoy en día que es legado, y que REST con JSON se ve como un nuevo estándar con más proyección de uso.

2.3 Servicios

En esta sección se define el concepto SOA necesario para continuar con el análisis del concepto de Servicio.

2.3.1 SOA

Es un patrón de arquitectura para el diseño de aplicaciones distribuidas para proveer soporte al desarrollo de software orientado a servicios. [26]

2.3.1.1 Principios de SOA

Para entender el concepto de SOA es necesario examinar los principios en los que se basa, que son los siguientes [27]:

- **Contratos de servicio estandarizados:** Los servicios describen sus capacidades en sus contratos de servicio. Los servicios en el mismo inventario se deben ajustar a los mismos estándares para el diseño de contratos.
- **Servicios con bajo acoplamiento:** Hace referencia al nivel de dependencia entre servicios, entre el proveedor y el consumidor. Cuanto menos acoplamiento se logra una mayor independencia para el diseño del servicio y su posterior evolución.
- **Abstracción:** Este principio pone énfasis en esconder los detalles innecesarios del servicio. Los contratos de los servicios sólo deben incluir la información esencial y la información de los servicios se limita a la que se publica en su contrato.
- **Reusabilidad:** La arquitectura SOA no busca la sustitución de las lógicas de negocio actuales sino que proporciona una forma de reaprovechar todos estos activos encapsulados en servicios para que a su vez puedan ser reutilizados por otros servicios.
- **Autonomía:** Los servicios son autónomos, esto es, tienen un alto control sobre su entorno de ejecución. La autonomía representa la independencia con que el servicio ejecuta su lógica.
- **Sin estado:** Los servicios minimizan el consumo de recursos difiriendo el manejo de su información de estado cuando es necesario.
- **Capacidad de descubrimiento:** Los servicios se complementan con metadatos comunicativos a través de los cuales pueden ser descubiertos e interpretados de forma efectiva. Esto apunta a incrementar el reuso de los servicios y reducir la posibilidad de desarrollar servicios cuya funcionalidad se solape o ya exista.
- **Composición:** Los servicios participan en composiciones de forma efectiva, sin importar el tamaño o complejidad de la composición. Este principio permite la agilidad ya que promueve la construcción de sistemas a partir de servicios ya existentes.

2.3.2 Definición

Según la w3c [28] un servicio es un recurso abstracto que representa una capacidad de realizar tareas que forman una funcionalidad coherente desde el punto de vista de las entidades de los proveedores y entidades solicitantes.

The Open Group define servicio como una representación lógica y repetible de una actividad de negocio que tiene un resultado de negocio. Es autocontenido, es una caja negra para otros servicios. [29]

2.3.3 Comparativa entre APIs y Servicios

Desde el punto de vista técnico los principios en los que se basa un servicio aplican al diseño de una API por lo que técnicamente una API es también un servicio.

El concepto de API ha tomado un gran valor debido al desarrollo que se ha producido en Internet donde existen un gran número de empresas que se dedican al desarrollo de APIs.

Se puede decir que la separación de ambos conceptos es relativamente nueva y aún existen controversias de si son lo mismo o difieren en algo. [30]

Según Claus T. Jansen de IBM [31] a gran diferencia radica en el objetivo que pretenden alcanzar. Las APIs tienen como objetivo aumentar el ritmo de innovación, mientras que el servicio pretende aumentar la agilidad y la eficacia de la prestación. Las APIs son la forma en que están re-empaquetadas esas capacidades (servicios), producidas y compartidas de una forma fácil de usar.

2.4 API Managers

En esta sección se describe el API Managers presentando una definición y describiendo sus principales funcionalidades. Finalmente se presenta una arquitectura básica de un API Manager. Para mayor información consultar en los Anexos I y IV.

2.4.1 Definición

Para Microsoft [24] se trata de un middleware que facilita a organizaciones la publicación de las APIs para su uso, agregando valor a las APIs gestionando seguridad, tráfico, y otras características de la comunicación entre el consumidor y la API publicada por la organización, a través de un ambiente seguro y escalable.

Según IBM [31], un API Manager es un middleware que permite a un desarrollador, gestionar sus APIs. Para un proveedor de APIs, la gestión se centra en la facilitación para la externalización de la API, administrando como parte de esta externalización, los mapeos de datos, la seguridad, el control de acceso a la API, el monitoreo y el control de versiones.

Por otro lado, la gran mayoría de estos proveedores de API Managers también destacan las características de la facilidad de uso que este tipo de middleware debe proveer tanto a los proveedores de APIs como a los consumidores de las mismas. [32]–[34]

2.4.2 Principales funcionalidades

La lista de funcionalidades básicas de la mayoría de las APIs Managers que se puede encontrar en las diferentes documentaciones de cada producto son las siguientes (ver referencias de Microsoft [24], IBM [31], CA Technologies [20], WSO2 [35], Amazon [32] y MuleSoft [33]):

1. **Documentación:** Un servicio de API Management debe proveer un mecanismo sencillo para acceder a la documentación de cada API y permitir a los desarrolladores que desean utilizar una API probarla antes de usarla. Por ejemplo, las APIs ofrecidas a través de Web Services SOAP tienen su documentación a través de su WSDL [36]. En cambio para las APIs REST existen otros tipos de lenguajes de especificación, como por ejemplo WADL [37], Swagger [38] o RAML [39].

2. **Analítica y Estadísticas:** Las API Managers deben ofrecer herramientas para ayudar a las organizaciones a entender cómo los clientes utilizan sus APIs. El almacenamiento de las estadísticas de uso de cada API, así como su presentación tanto en gráficas, como en reportes en forma de tablas está generalizado y la gran mayoría de los productos ya ofrecen algún tipo de analíticas en sus versiones out-of-the-box.
3. **Despliegue:** Debe ser flexible y soportar tanto nubes públicas como privadas, implementaciones on-premise, o combinaciones de cualquiera de éstas. En escenarios más complejos, existen modelos de despliegue que soportan algunos productos donde se produce una combinación de implementaciones. Muchas veces, estos productos desplegables en este tipo de escenarios son ofrecidos con otras formas de licenciamiento y costo, por ejemplo, en versiones Enterprise del mismo producto.
4. **Compromiso con el desarrollador:** Esta característica implica que los productos de API Management tendrían que ser capaces de ofrecer funcionalidades amigables para el desarrollador, como por ejemplo, la autogestión de sus claves y credenciales, fácil acceso a la documentación de las diferentes APIs, gestiones realizadas sobre el ciclo de vida de una API, entre otras.
5. **Entorno Sandbox:** Se le llama comúnmente Entorno Sandbox a la capacidad de la plataforma de ofrecer un ambiente de pruebas aislado del ambiente productivo de la API sobre la misma instalación del producto API Management. En otras palabras, el Entorno Sandbox es un entorno de pruebas que se ofrece para una API.
6. **Tráfico y caché:** Esta característica está asociada a la protección física del ambiente que aloja físicamente a la API gestionada. La administración del tráfico puede tener varias formas: esquemas de validación por ejemplo, no permitir más de cierta cantidad de invocaciones por unidad de tiempo. Por otro lado, la capacidad de mantener un caché de respuestas de la API también permite, en algunos casos, no permitir que las cargas de tráfico se propaguen hacia las APIs, protegiendo de esta manera las mismas ante escenarios de alta carga en el tráfico.
7. **Seguridad:** El servicio debe proveer, por lo menos, manejo de identidad y acceso para usuarios y desarrolladores, de manera de proteger los datos que se intercambian en cualquier invocación de una API. Por lo general, la seguridad en los API Managers se maneja a través de protocolos estándares de autenticación y autorización, como lo son el uso de certificados digitales en SSL/TLS [40], generación de claves de autorización HTTP, WS-Security [41] para casos de APIs SOAP, OAuth, o incluso OpenID [42].
8. **Monetización:** El concepto de monetización está ligado a las formas que puede tener una organización de generar ingresos a partir del uso que puedan tener sus APIs por parte de otras organizaciones [43]. Ofrecer esta capacidad dentro de un API Manager implica ofrecer funcionalidades que permitan al usuario proveedor de una API diagramar planes de uso de sus APIs. Esta funcionalidad de monetización también puede incluir por ejemplo, componentes que faciliten el pago y/o la facturación.

9. **Disponibilidad:** El sistema debe ser altamente disponible, escalable y redundante. Un ambiente de una API puede volverse demandante y el servicio debe ser capaz de manejar problemas o picos de tráfico temporales. Principalmente, esta característica se refiere a la capacidad del middleware de tener una disponibilidad suficiente para poder ofrecer todas las APIs que se gestionan sin inconvenientes.
10. **Sistemas legados:** También debe ofrecer soporte de integración a sistemas legados, es decir, permitir integrar APIs legadas en sistemas que no ofrecen facilidades para ser integrados a un middleware o para ser publicados directamente a los desarrolladores cliente.

2.4.3 Arquitectura básica

En base a los productos de API Managers evaluados a continuación se describe la arquitectura básica [35], [43]. Para mayor información consultar en el Anexo II.

En la Figura 2.1 se pueden observar los siguiente componentes que generalmente se encuentran en un API Manager.

- API Publisher: es una aplicación web cuya principal función es ofrecer al desarrollador un lugar donde publicar las APIs, gestionarlas y documentarlas.
- API Store: es una aplicación web utilizada como catálogo de APIs que permite al desarrollador suscribirse a las mismas y de esa forma poder utilizarlas.
- API Gateway: componente cuya principal función es la de habilitar la interconexión entre los servicios y los consumidores, a través de las APIs publicadas en él. Actúa como un proxy que intercepta las peticiones entrantes y aplica políticas de seguridad y disponibilidad.

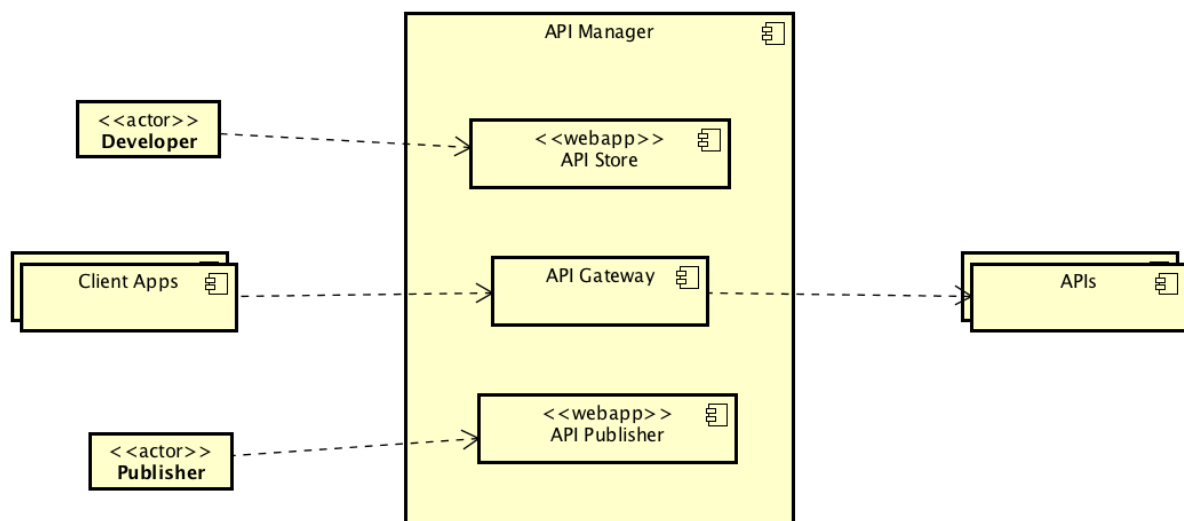


Figura 2.1: Arquitectura de un API Manager.

2.5 ESB

En esta sección se describe el Enterprise Service Bus presentando una definición del mismo y describiendo sus principales funcionalidades. Finalmente se presenta una arquitectura básica de un ESB. Para mayor información consultar en los Anexo III y IV.

2.5.1 Definición

Un ESB es una plataforma de integración basada en estándares que combina mensajería, web services, transformación de datos y ruteo inteligente para conectar y coordinar de forma confiable la interacción de un gran número de aplicaciones diversas a través de empresas extendidas con integridad transaccional. [44]

Una definición alternativa encontrada está dada de la siguiente manera: el ESB es una columna vertebral de mensajería basada en estándares abiertos y diseñada para posibilitar la implementación, despliegue y administración de soluciones SOA. Es un conjunto de capacidades de infraestructura implementadas vía tecnologías de middleware que posibilitan una SOA y alivianan problemas de disparidad entre aplicaciones que se ejecutan en plataformas heterogéneas y usan diversos formatos de datos. [45]

2.5.2 Principales funcionalidades

La lista que sigue contiene aquellas funcionalidades consideradas básicas que se encuentran presentes en la mayoría de los ESB:

1. **Conectividad/Adaptadores:** un ESB típicamente incluye varios adaptadores que le permiten interactuar con múltiples protocolos de comunicación o transporte como HTTP, FTP, POP3/SMTP, sistemas de archivos, etc. [31]
2. **Transformación de Mensajes:** permite modificar los mensajes que se envían a través del ESB. El mecanismo recibe un mensaje y devuelve otro transformado de acuerdo a una lógica de transformación como por ejemplo utilizando transformaciones XSLT para mensajes XML. Estas capacidades posibilitan, por ejemplo, que aplicaciones que utilizan distintos formatos o modelos de datos puedan comunicarse. [31]
3. **Enrutamiento:** es el encargado de determinar el destino de los mensajes, y se utiliza cuando existen varios destinos posibles, pero solamente se debe seleccionar uno. Este mecanismo posee además una lógica de ruteo, la cual es responsable de decidir a qué servicio se envía cada mensaje.

Algunos ejemplos de ruteo son:

- i. Ruteo basado en contenido: determina el destino de un mensaje en base a su contenido.
- ii. Ruteo basado en contexto: determina el destino de un mensaje en base a condiciones del ambiente de ejecución.

Permiten la utilización de enrutamiento basado en reglas por ejemplo a través de Drools. [31]

4. **Flujos de mediación:** especifican una secuencia de operaciones de mediación(ruteo, transformación, etc) a ejecutar en un mensaje. [31]
5. **Mensajería asincrónica (MOMs):** componente central de los ESB. Generalmente se usan en el ESB para ofrecer servicios como garantía de entrega, control y balanceo de carga, mapeo de tipos de interacción(sincrónico/asincrónico). [31]

6. **Monitoreo y administración:** incluyen funcionalidades de monitoreo que proveen valores para distintas métricas como por ejemplo, tiempos de respuesta, cantidad de mensajes procesados por servicios, entre otros. [46]
7. **Seguridad:** Autenticación, autorización, y funcionalidad de encriptación se proveen a través del ESB para asegurar los mensajes entrantes. Asimismo estas funcionalidades se pueden aplicar a mensajes salientes para satisfacer requerimientos de seguridad del proveedor del servicio a consumir. [46]
8. **Calidad de servicios:** Un ambiente de monitoreo y administración es fundamental para configurar el ESB para que sea confiable y tenga un alto desempeño, se deben tener en cuenta características como transparencia de localización, abierto a estándares, escalabilidad y alta disponibilidad. [46]

2.5.3 Arquitectura básica

Según Forrester un modelo de arquitectura de referencia para un ESB se muestra en la Figura 2.2. La arquitectura genérica se basa en las funcionalidades principales que debería proveer un ESB.

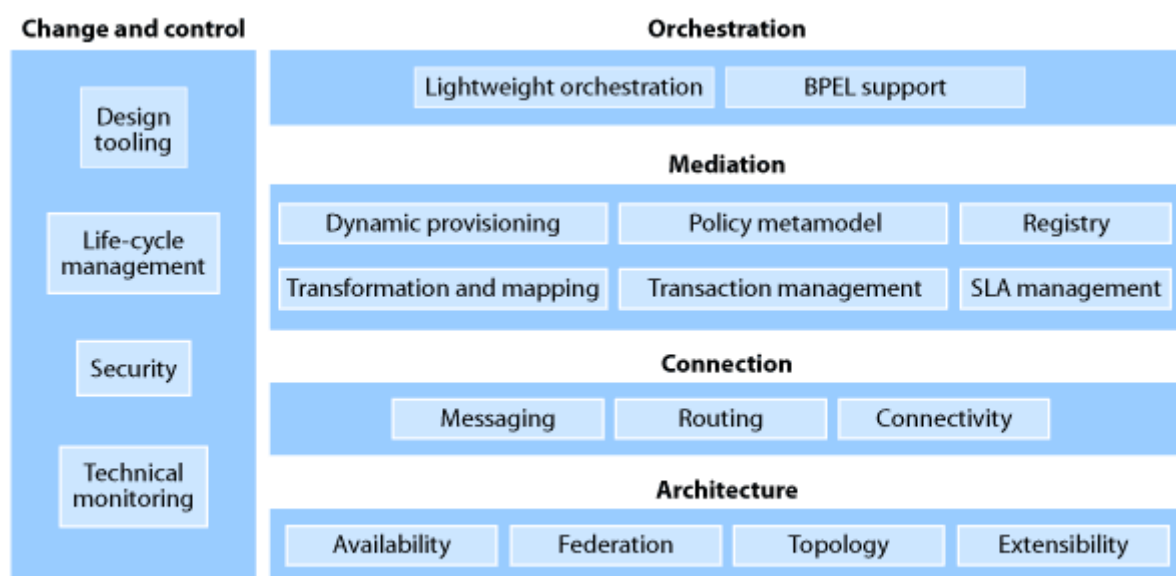


Figura 2.2 - Arquitectura de referencia para un ESB.[47]

En la Figura 2.2 se pueden observar los siguientes componentes principales de una arquitectura de referencia de un ESB:

- Orquestación: este componente permite la orquestación de servicios de forma ligera y robusta. Permite desarrollar procesos que pueden constituir servicios de negocio.
- Mediación: esta capa se encarga de todo lo relacionado con la transformación de protocolos, mapeo de servicios, implementación de SLAs, manejo de las transacciones.
- Conectividad: esta capa brinda una gran variedad de soporte a protocolos de comunicación, estándares y alternativas de conectividad.

- **Arquitectura:** en esta capa se encuentran componentes relacionados al rendimiento, la capacidad que deben tener los ESB de federarse con otros ESB, topologías soportadas y la escalabilidad.
- **Administración:** en esta capa se encuentran componentes de diseño, manejo del ciclo de vida, monitoreo y seguridad.

2.6 Containers Linux - Virtualización a nivel de Kernel

En esta sección se describen los containers Linux y las máquinas virtuales. Primeramente se brinda una definición y se presentan sus principales características, luego se muestra la arquitectura básica. Finalmente se presenta una comparativa entre ambas tecnologías.

2.6.1 Containers Linux

Los Containers Linux son una tecnología de virtualización a nivel del sistema operativo para Linux, que permiten empaquetar y aislar las aplicaciones con sus entornos de tiempo de ejecución completo, es decir, con todos los archivos necesarios para ejecutarse. Esto garantiza que el software se ejecutará siempre de la misma manera, independientemente del ambiente. [48]

2.6.1.1 Características

A continuación se presentan las principales características: [49]

- **Liviano:** se ejecutan en una sola máquina compartiendo el mismo kernel del sistema operativo. Comienzan instantáneamente y utilizan menos memoria RAM.
- **Abierto:** se basan en estándares abiertos, lo cual permite que los mismos puedan ser ejecutados en todas las principales distribuciones de Linux y sobre cualquier infraestructura.
- **Seguros por defecto:** aíslan a las aplicaciones y la infraestructura subyacente, mientras que proporciona una capa adicional de protección para la aplicación.

2.6.1.2 Arquitectura

En la Figura 2.3 se muestra una arquitectura de los Containers Linux, específicamente la de un Container Linux en Red Hat.

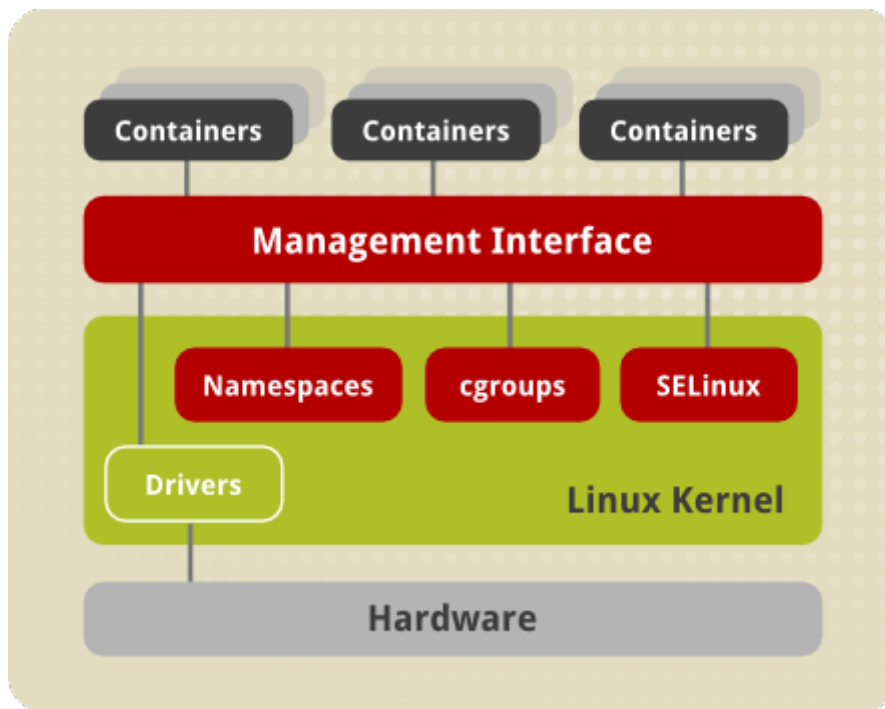


Figura 2.3 - Arquitectura de Containers Linux en Red Hat Enterprise Linux 7. [50]

Los principales componentes son [50]:

- **Namespaces:** garantiza el aislamiento de procesos.
- **cgroups:** se emplean para controlar los recursos del container.
- **SELinux:** se utiliza para asegurar la separación entre el host y el container, y también entre los containers.
- **Management Interface:** forma una capa superior que interactúa con los componentes del kernel mencionados anteriormente, y proporciona herramientas para la construcción y el manejo de los containers.

2.6.2 Virtualización a nivel de Kernel

Kernel Virtual Machine (KVM) es una solución de virtualización completa para Linux, en la que se pueden ejecutar múltiples máquinas virtuales que se ejecutan sin modificar imágenes de Linux o Windows. Cada máquina virtual tienen un hardware virtualizado privado: una tarjeta de red, disco, adaptador de gráfico, etc. [51]

Es un ambiente creado por el software de virtualización mediante la interceptación de accesos de ciertos componentes de hardware y ciertas características. La parte física de la computadora es llamada Host, mientras que la máquina virtual es a menudo llamada Guest. La mayoría del código de Guest se ejecuta sin modificaciones, directamente en el equipo host y el sistema operativo guest “piensa” que se está ejecutando en la máquina real.

2.6.2.1 Características

Las principales características son [52]:

- **Sistemas operativos compatibles:** se puede ejecutar el software escrito para un sistema operativo en otro (por ejemplo, el software de Windows en Linux) sin tener que reiniciar el sistema.
- **Consolidación de la infraestructura:** dado que el rendimiento completo de las computadoras de hoy en día, rara vez se necesita a tiempo completo, en lugar de ejecutar muchos de estos equipos físicos, se puede “empaquetar” muchas máquinas virtuales en unos pocos hosts potentes y balancear la carga entre ellos. Esto puede ahorrar una gran cantidad de costos de hardware: por ejemplo, mediante la consolidación de muchos servidores en unos pocos.
- **Pruebas y recuperación:** especialmente con el uso de snapshots, permite romper con un entorno informático ejecutándolo como una máquina virtual. Si algo sale mal, se puede cambiar fácilmente de nuevo a una snapshot anterior y evitar la necesidad de copias de seguridad y restauraciones frecuentes.

2.6.2.2 Arquitectura

En la Figura 2.4 se muestra una arquitectura de virtualización a nivel de Kernel, mostrando sus principales componentes, específicamente la de KVM en Red Hat. KVM está compuesto por un módulo kernel, que proporciona el core de la infraestructura de virtualización y un módulo específico de procesador.

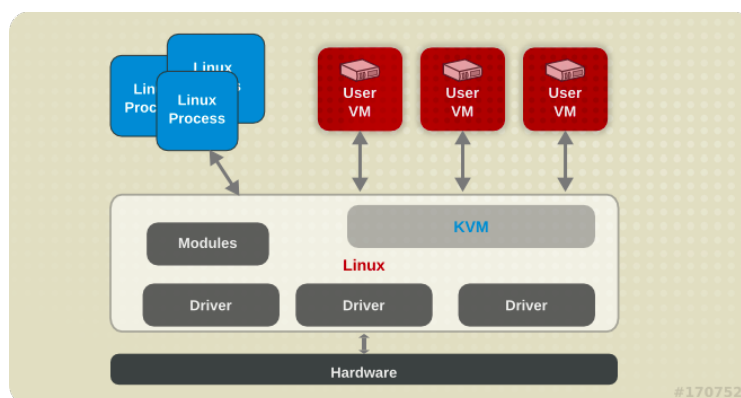


Figura 2.4 - Arquitectura de KVM en Red Hat Enterprise Linux 7. [53]

2.6.3 Comparación: Containers Linux vs Virtualización a nivel de Kernel

Las máquinas virtuales KVM requieren un kernel propio, mientras que los Containers Linux comparten el kernel del sistema operativo host. Por lo tanto, es posible desplegar un mayor número de containers que máquinas virtuales en el mismo hardware. [50]

Tanto los Containers Linux como la virtualización KVM tienen ciertas ventajas y desventajas que influyen en los casos de uso en los que estas tecnologías se aplican típicamente.

A continuación, se describen estas ventajas y desventajas para estas tecnologías [50]:

2.6.3.1 Virtualización a nivel de Kernel

- La virtualización KVM tiene como ventaja que permite bootear sistemas operativos completos de diferentes tipos, incluso los sistemas que no son Linux. La naturaleza cargada de recursos de las máquinas virtuales tiene como desventaja que el número

de máquinas virtuales que se pueden ejecutar en un host es menor que el número de containers que se pueden ejecutar en el mismo host.

- La ejecución de instancias con kernel separado generalmente proporciona separación y seguridad. La caída inesperada de uno de los kernels no inhabilita todo el sistema.
- La máquina virtual invitada está aislada de los cambios del host, lo que permite ejecutar diferentes versiones de la misma aplicación en el host y en la máquina virtual. La KVM también ofrece muchas características útiles, tales como la migración en vivo.

2.6.3.2 Containers Linux

- Los contenedores linux están diseñados para soportar el aislamiento de una o más aplicaciones.
- Los cambios en el sistema son visibles en cada container. Por ejemplo, si se actualiza una aplicación en la máquina host, este cambio se aplicará a todos los sandboxes que ejecuta instancias de esta aplicación.
- Dado que los containers son de peso liviano, se puede ejecutar un gran número de ellos simultáneamente en una máquina host.

En la tabla 2.1 se muestran algunas de las características de ambas tecnologías.

Característica	Containers Linux	Virtualización KVM
Incluye a la aplicación en el ambiente para la ejecución de la misma	Si	Si
Incluye los binarios y librerías necesarias en el ambiente para la ejecución de la aplicación	Si	Si
Incluye todo un sistema operativo invitado en el ambiente para la ejecución de la aplicación	No	Si
Espacio ocupado por el ambiente para la ejecución de la aplicación	Centenas de MBs	Decenas de GBs
Comparten el kernel	Si	No
Ejecutan como procesos aislados en modo usuario en el sistema operativo host	Si	No
Vinculado a infraestructura específica	No	Si
Peso	Liviano	Pesado
Nivel de virtualización	Sistema operativo	Hardware

Filesystem	Individual	Individual
Visualización de otros procesos	No	No
Uso de recursos computacionales	Puede ser acotado	Acotado

Tabla 2.1 - Características de Container Linux y Vms. [49], [50]

Los containers y las máquinas virtuales tienen similares beneficios de aislamiento y asignación de recursos, pero un enfoque arquitectónico diferente permite a los containers ser más portables y eficientes, porque utilizan características de aislamiento de recursos del *kernel* de Linux para permitir que los containers independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

2.6.3.3 Arquitecturas

En las Figuras 2.5 y 2.6 se expone la arquitectura de virtualización a nivel de Kernel y la de Containers Linux, respectivamente. Las mismas nos permiten visualizar las diferencias a nivel de arquitectura entre ambas tecnologías.

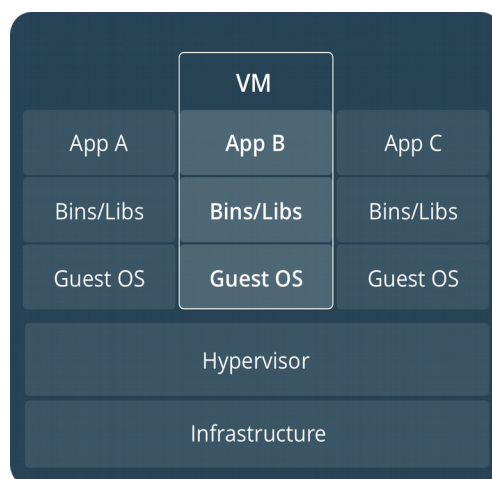


Figura 2.5 - Arquitectura de Virtualización a nivel de Kernel. [39]

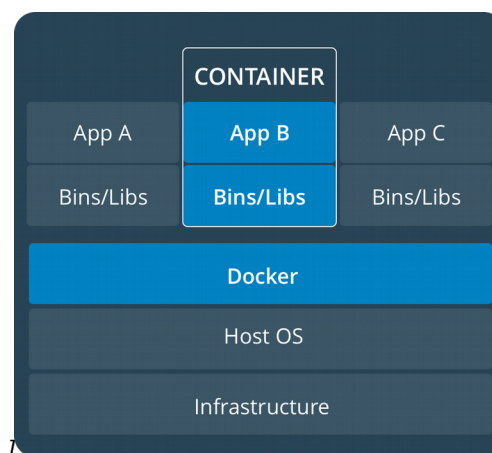


Figura 2.6 - Arquitectura de Containers Linux. [49]

2.6.4 Productos

2.6.4.1 Docker

Docker es una plataforma de código abierto que automatiza el despliegue de aplicaciones dentro de containers. Permite separar a las aplicaciones de su infraestructura y tratar su infraestructura como una aplicación administrada, mediante la combinación de características de contenerización del kernel con flujos de trabajo y herramientas que ayudan a administrar y desplegar las aplicaciones. [54]

Proporciona una forma de ejecutar cualquier aplicación de forma segura en un contenedor aislado. El aislamiento y la seguridad le permiten ejecutar muchos containers de forma simultánea en el host. La naturaleza liviana de los containers, que se ejecutan sin la carga adicional de un hipervisor, significa que se puede sacar más provecho del hardware. [54]

Esta plataforma proporciona un conjunto de herramientas y permite gestionar el ciclo de vida de sus containers [54]:

- Desarrollar aplicaciones (y componentes secundarios) utilizando containers.
- Distribuir y enviar los containers a los equipos para el desarrollo y testing.
- Desplegar las aplicaciones en ambientes de producción, ya sea en un data center local o en el cloud.

2.6.4.1.1 Arquitectura

Docker utiliza una arquitectura cliente-servidor, en la que el componente *Docker Client* se comunica con el componente *Docker Daemon* que se encarga de construir, ejecutar y distribuir los containers Docker. *Docker Client* y *Docker Daemon* pueden ejecutarse en el mismo sistema, o *Docker Client* puede conectarse al *Docker Daemon* de manera remota. Su comunicación es mediante una API REST, a través de sockets UNIX o una interfaz de red. [54]

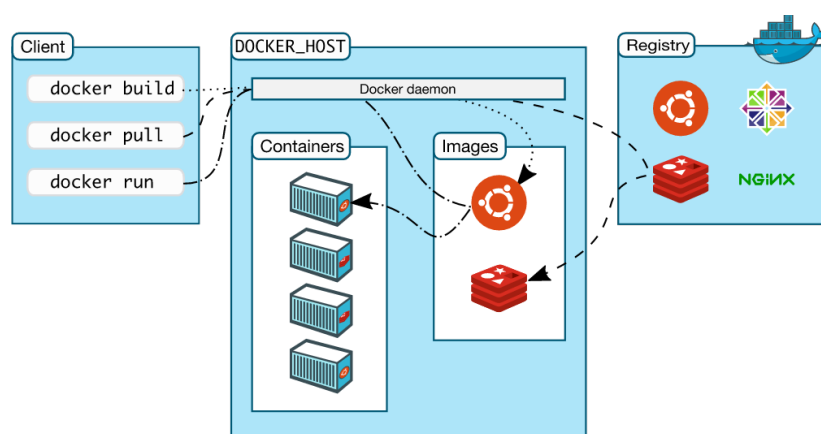


Figura 2.7 - Arquitectura de Docker. [54]

En la Figura 2.7 se muestra la arquitectura de Docker con sus principales componentes [54].

- **Docker Daemon:** expone la API de Docker y administra los objetos de Docker como imágenes, containers, redes y volúmenes. Puede comunicarse con otros *Docker Daemon* para administrar los servicios de Docker.

- **Docker Client:** es la principal forma que tienen los usuarios para interactuar con Docker. El comando *docker* utiliza la API de Docker.
- **Docker Registries:** son repositorios donde se almacenan imágenes de Docker.

2.6.4.2 OpenShift

OpenShift es una Plataforma como Servicio (PaaS) que le permite a los desarrolladores crear aplicaciones, hostearlas, y escalarlas en un ambiente cloud.

2.6.4.2.1 Características

A continuación se presentan las principales características [14]:

- **Desarrollo:** permite la creación de aplicaciones utilizando la consola web, la interfaz de línea de comandos (CLI), o un entorno de desarrollo integrado (IDE).
- **Despliegue:** orquesta las aplicaciones, determinando donde se ejecutarán las aplicaciones en los containers, y gestionando las aplicaciones para asegurar que las mismas estén disponibles para los usuarios finales.
- **Construcción:** el desarrollador puede subir el código construido y el mismo será ejecutado en OpenShift a través de su solución de control de versiones de software u OpenShift puede ser integrado con sistemas de construcción automática y sistemas de integración/entrega continua.
- **Administración:** permite monitorear, debuggear y realizar ajustes en tiempo de ejecución a las aplicaciones. Además, las mismas pueden ser escaladas de forma automática o manual, como por ejemplo tras el aumento en el tráfico web.

2.6.4.2.2 Principales componentes

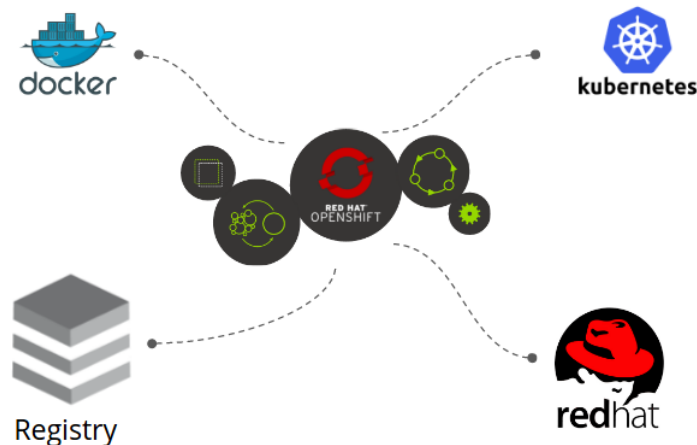


Figura 2.8 - Principales componentes del producto OpenShift.

En la Figura 2.8 se muestran los principales componentes del producto OpenShift.

- **Docker:** es una implementación de containers Linux. [55]
- **Kubernetes:** es un sistema de automatización del despliegue, escalabilidad y gestión de aplicaciones que ejecutan en containers. [56]

- **Red Hat Enterprise Linux:** es una distribución comercial del sistema operativo GNU/Linux desarrollada y mantenida por la compañía Red Hat Inc. [57]
- **Atomic Registry:** es un repositorio de imágenes Docker. Permite administrar sus imágenes sin *hubs* de terceros. [58]

Capítulo 3: Análisis

En este capítulo se realiza un análisis de la problemática planteada, describiendo los requerimientos que la solución debe cumplir.

El capítulo está dividido en tres grandes secciones donde en primer lugar se realiza un análisis de los conceptos presentados en el artículo. Primeramente se describe el modelo del cual se partió. Luego en base al estudio de la realidad y a la arquitectura de referencia planteada en el artículo se describen los requerimientos establecidos para el proyecto. Finalmente se presenta el alcance del proyecto.

El proyecto se basa en diseñar e implementar una solución para una Plataforma de Integración Inter-organizacional basándose en la arquitectura de referencia para Plataformas de Integración presentado en el artículo [1].

3.1 Descripción del artículo

El artículo propone una arquitectura de referencia para plataformas de integración que apoyan la colaboración entre organizaciones. Estas plataformas de integración son infraestructuras basadas en *middleware* que proporcionan una capa de procesamiento intermedia, con capacidades de integración, entre organizaciones. Esta arquitectura se presenta a través de cuatro vistas donde se describen los principales conceptos, casos de uso y capacidades de estas plataformas. [1]

3.1.1 Conceptos

El diagrama de la figura 3.1 muestra los conceptos y las relaciones entre ellos, que se presentan en el artículo.

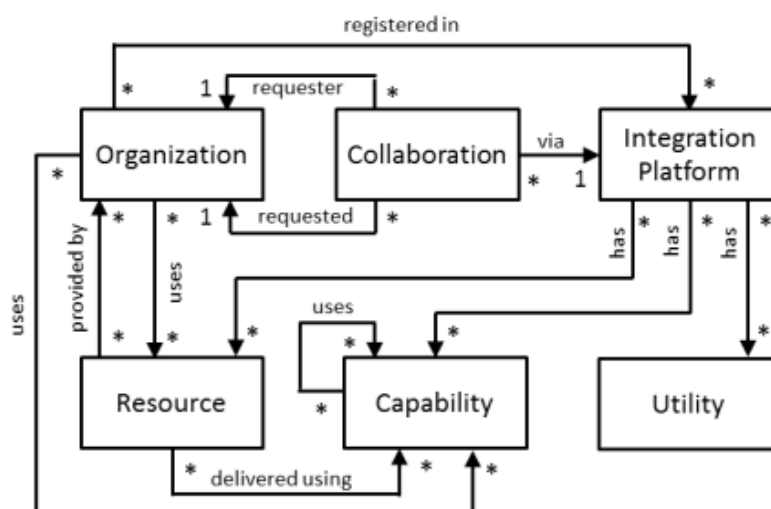


Figura 3.1 - Modelo conceptual del artículo [1].

Una Plataforma de Integración posee Capacidades que son mecanismos que tienen como objetivo facilitar los problemas de integración. A su vez, tiene Recursos que son

proporcionados por una Organización y se exponen a través de dicha plataforma utilizando las Capacidades de la misma. Las Organizaciones se registran en las Plataformas de Integración para utilizar sus Capacidades y Recursos, y realizar sus procesos de negocio. También colaboran entre sí a través de una Plataforma de Integración, estableciendo una relación entre ellas para que una de las Organizaciones utilice un recurso proporcionado por la otra. Además, las Plataformas de Integración proporcionan Utilidades que son funcionalidades dirigidas a los usuarios finales y brindadas a través de mecanismos adecuados.

3.1.2 Casos de uso

En la figura 3.2, se muestran los casos de uso de la arquitectura descrita en el artículo.

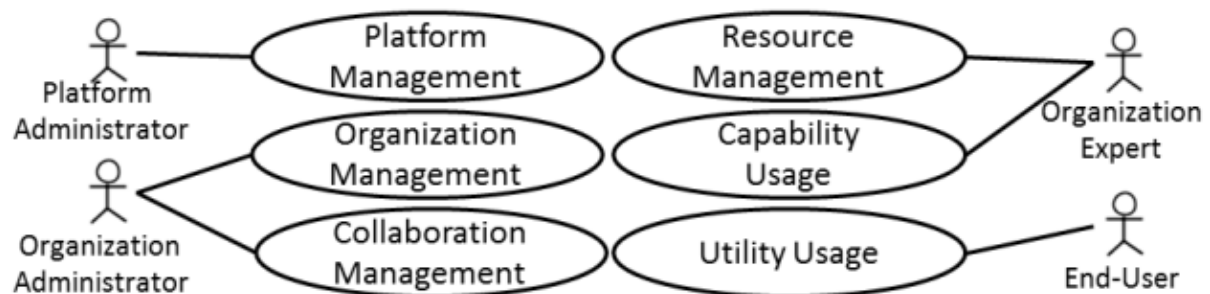


Figura 3.2 - Casos de uso de la arquitectura descrita en el artículo [1].

La administración de plataformas permite a los administradores de plataformas administrar una Plataforma de Integración.

La administración de la organización permite a los administradores de la organización administrar el registro de su organización en una Plataforma de Integración.

La gestión de colaboración permite a los administradores de la organización gestionar los aspectos relacionados con la colaboración con otras organizaciones registradas en la Plataforma de Integración.

La gestión de recursos permite a los expertos de la organización administrar los recursos de su organización e interactuar con los recursos de otras organizaciones.

El uso de una Capacidad permite a los expertos de la organización configurar una Capacidad disponible en una Plataforma de Integración.

El uso de una Utilidad permite a los usuarios finales usar una Utilidad disponible en una Plataforma de Integración.

3.1.3 Características del modelo

En la figura 3.3, se presentan las capacidades que la plataforma debe proporcionar para apoyar la colaboración entre organizaciones, y los componentes que las implementan.

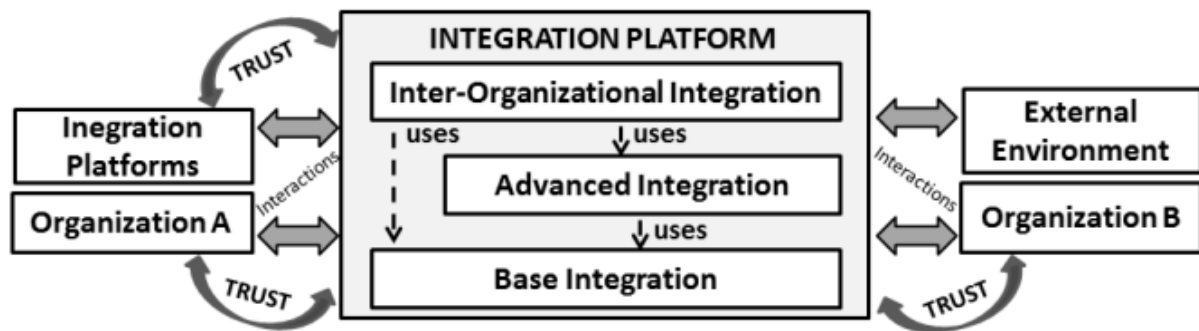


Figura 3.3 - Capacidades de la arquitectura descrita en el artículo [1].

Las organizaciones colaboran entre sí mediante el envío de mensajes a través de una Plataforma de Integración, e interactúan con una Plataforma de Integración para satisfacer las necesidades individuales de integración. Una Plataforma de Integración puede interactuar con el ambiente externo para satisfacer requisitos específicos de integración, e interactuar con otras Plataformas de Integración. Por lo tanto, una Plataforma de Integración actúa como una capa intermedia a través de la cual se procesan todas las interacciones, en particular las inter-organizacionales. Además, se debe tener en cuenta que podrían requerirse relaciones de confianza entre las partes que interactúan. [1]

3.2 Extensión del artículo

Se comienza el desarrollo del modelo conceptual de la solución a partir de la distinción de cuatro grupos de conceptos de manera de organizar la descripción del modelo completo de la extensión del artículo. Dichos grupos de conceptos son:

- Modelo de usuarios
- Modelo de organizaciones
- Modelo de capacidades
- Modelo de recursos

A continuación, se presentarán los grupos de conceptos mencionados anteriormente con sus correspondientes diagramas que son resultado del análisis desarrollado.

3.2.1 Modelo de usuarios

Dentro de esta sección del modelo propuesto, a los conceptos presentados originalmente se anexa este grupo de conceptos que agrega las relaciones entre los usuarios de la plataforma con los conceptos básicos presentados, en particular, con las Organizaciones.

En la figura 3.4, se muestra el modelo de usuarios de la solución propuesta con los conceptos que lo integran.

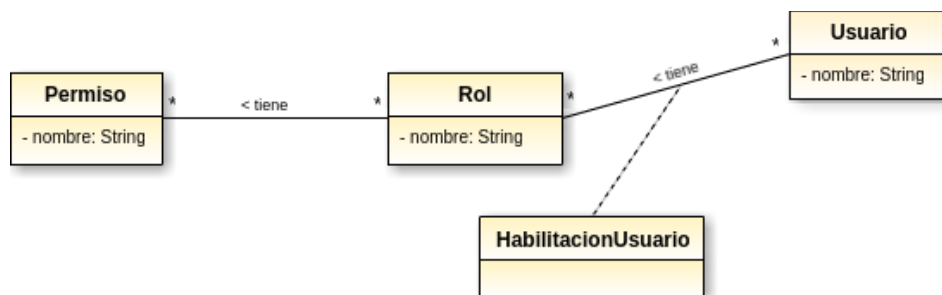


Figura 3.4 - Modelo usuarios de la solución propuesta.

Este modelo se basa principalmente en que los Usuarios que pueden utilizar la plataforma pueden tener varios Roles, que definen cuál será el conjunto de Permisos que tiene el Usuario dentro de una Organización. Esto permite que un Usuario pueda tener varios Roles dentro de una misma Organización, e inclusive, un Usuario pueda pertenecer a varias Organizaciones.

Estos usuarios utilizarán las interfaces de gestión para hacer uso de las funcionalidades que tienen definidos para cada Organización, en cada Rol.

Este modelo de usuarios es similar a otros vistos en diferentes productos existentes en el mercado, con la particularidad que este modelo se basa en Usuarios asociados a Organizaciones, y en diferentes Roles que pueden cumplir dichos Usuarios dentro de cada Organización a la que pertenecen.

3.2.2 Modelo de organizaciones

A continuación, en la figura 3.5 se presenta el modelo de organizaciones de la solución propuesta y los conceptos que lo componen.

Las Organizaciones crearán sus Plataformas a la hora de crear sus Recursos. Estas Plataformas serán creadas a través de una Especificación de Plataforma, que no es más que un conjunto de atributos que definen a esa Plataforma y a los cuales se les deberá asignar un valor al momento de crear la misma.

Por otro lado, las Organizaciones definirán cuáles serán los diferentes ambientes de ejecución que tendrán sus Recursos a la hora de ser publicados.

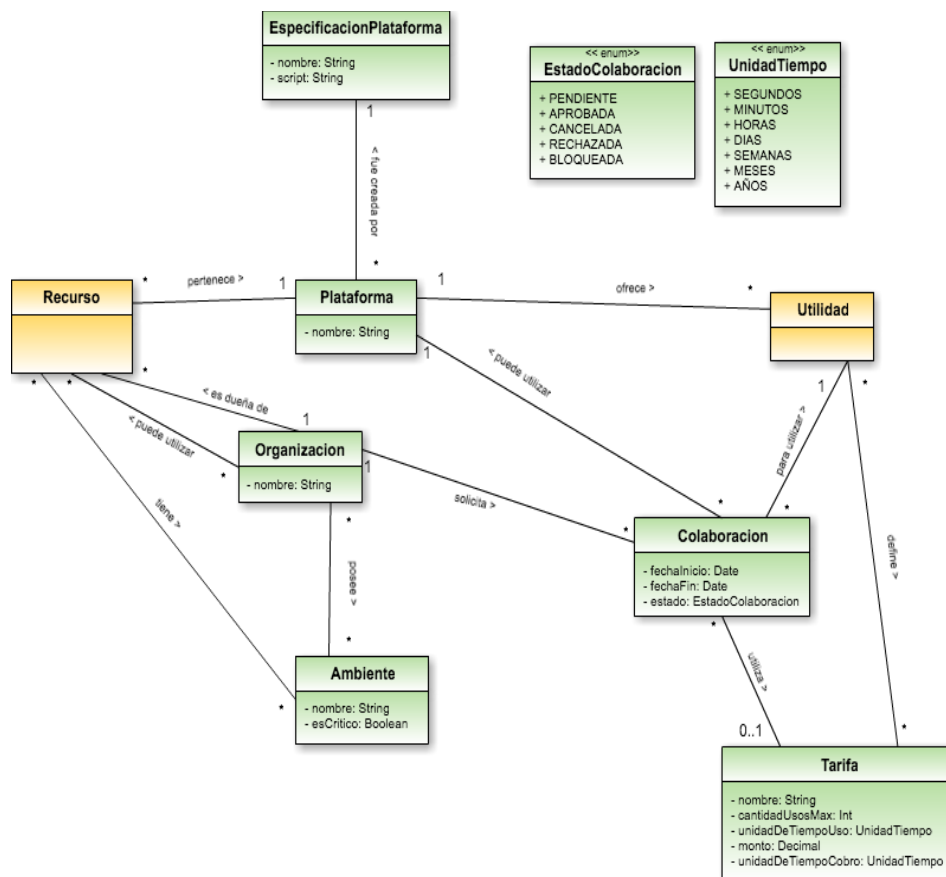


Figura 3.5 - Modelo de organizaciones de la solución propuesta.

Finalmente, las Organizaciones podrán solicitar Colaboraciones de otras Organizaciones para hacer uso de sus Plataformas a través de sus Recursos. Estas Colaboraciones deberán ser aprobadas por la Organización que recibe la solicitud, y cuándo esto sucede, la Organización solicitante tendrá acceso al uso de Recursos.

Los principales cambios respecto al modelo original son:

- La incorporación de manejo de ambientes dentro de las plataformas para cada Organización, que es una funcionalidad que se manejan generalmente en los diferentes productos, sobretodo de API Management. Existen algunos productos que manejan dos ambientes: un ambiente Sandbox y otro Productivo, y otros productos utilizan modelos más complejos, como el presentado en este diseño, donde cada Organización podría definir cuáles son los ambientes que posee. Esto permite que las organizaciones definan una configuración diferente para cada recurso que expongan por ambiente, y además, poder gestionar el ciclo de vida de cada recurso a través de los ambientes.
- Por otro lado también se incorpora el concepto de Especificación de Plataforma, que define las configuraciones necesarias para la creación de las diferentes Plataformas que las Organizaciones desean, desagregando las características de creación y configuración que puede tener una plataforma de la instanciación de la plataforma que solicita una Organización. Estas especificaciones definen los mecanismos de creación que tienen las plataformas, definiendo así un tipo de plataforma, e

independizando de cierta manera el modelo de la tecnología que se utilice para implementar este requerimiento.

- En el modelo original, una organización solicitaba colaboración con otra organización quedando asociada dicha colaboración únicamente entre organizaciones. En el modelo presentado más arriba, una organización realiza una solicitud de colaboración para un recurso en particular. De esta manera se es más restrictivo en lo que refiere al alcance de una colaboración en particular, pero a su vez, se es más flexible con quien comparte sus utilidades, ya que puede compartir una utilidad, pero no el conjunto entero de todas las utilidades que posee.
- Finalmente se incorpora el concepto de Tarifa, que define de qué manera se debe controlar el acceso a las diferentes utilidades. Cuando una organización define una Utilidad (es decir, cuando publica un Recurso), define el conjunto de Tarifas que podrá aplicarse al uso de dicha Utilidad. De esta manera, cuando se genera una solicitud para una utilidad, se debe definir qué tarifa se utilizará. Solamente podrán quedar sin asociación de Tarifas aquellas solicitudes de utilidades que no tienen asociadas tarifas.

3.2.3 Modelo de capacidades

En la figura 3.6, se muestra el modelo de capacidades de la solución propuesta y los conceptos que lo conforman.

Las Capacidades son aquellas funcionalidades ofrecidas por cada Plataforma y su existencia en ella es definida por las especificación de dicha Plataforma.

Las Capacidades además definen un conjunto de Características, que tienen un Tipo. Estas Características son los atributos variables que pueden definir las Capacidades.

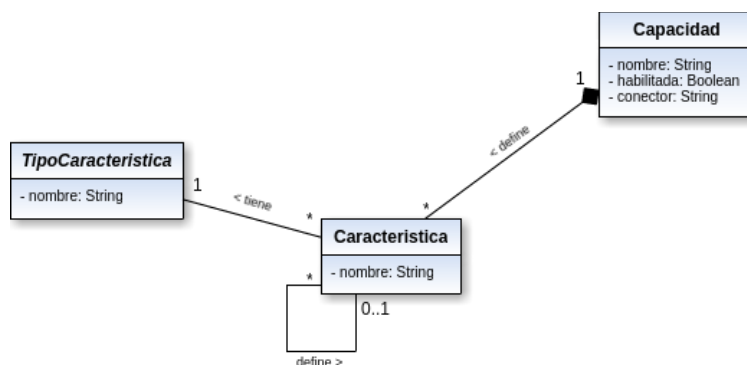


Figura 3.6 - Modelo de capacidades de la solución propuesta.

Ejemplos de Capacidades pueden ser Endpoints de diferentes tipos (SOAP, REST, JMS, ActiveMQ, otros MOMs, otros protocolos), Validaciones, Transformaciones, etc.

Esta generalización de la definición de las características de una capacidad es la diferencia principal que se puede encontrar con respecto al modelo original, y su finalidad es permitir la generalización de las características que se pueden definir, y luego configurar.

3.2.4 Modelo de recursos

A continuación, en la figura 3.7 se presenta el modelo de recursos de la solución propuesta y los conceptos que lo componen.

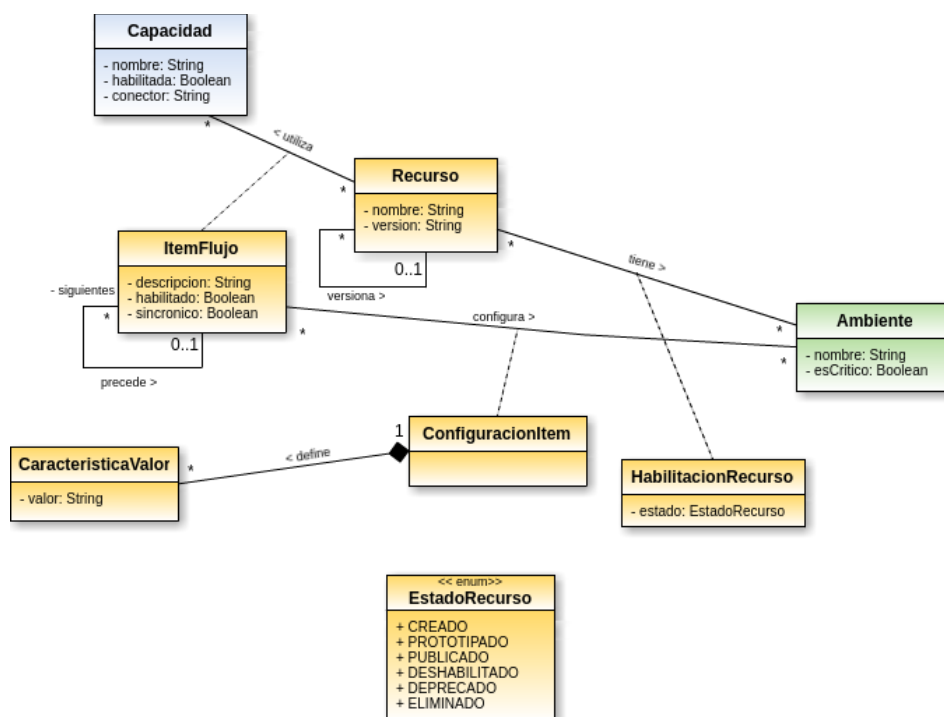


Figura 3.7 - Modelo de recursos de la solución propuesta.

Cuando las Organizaciones definen sus Recursos, van seleccionando Capacidades que se irán ejecutando en cierto orden, y a esto llamamos Ítem del Flujo de ejecución de un Recurso. Cada Ítem se podrá configurar para cada ambiente de manera indistinta, definiendo los Valores de las Características que cada Capacidad define para cada uno de dichos Ambientes. Estos Ítems se incorporan al modelo original ante la necesidad de representar a nivel de configuración de cada recurso, el flujo de ejecución que requiere cada recurso, utilizando las diferentes capacidades de la plataforma en la que se crea.

Por otro lado, los Recursos tendrán diferentes estados en los diferentes Ambientes que la Organización definió para ese Recurso, y es lo que llamamos el ciclo de vida del Recurso. Cuando un Recurso llega a ser Prototipado, se generará una Utilidad en la Plataforma donde se encuentra el Recurso. En este estado de Prototipación sólo se podrá acceder al recurso de manera privilegiada, es decir, únicamente podrá acceder la Organización que creó el Recurso. Recién en el momento que se Publica un Recurso en un ambiente, las Organizaciones que solicitaron colaboración con la Organización dueña del Recurso podrán utilizarlo. La organización podrá deshabilitar momentáneamente los Recursos, y luego volver a habilitar su publicación. Finalmente, las etapas finales del ciclo de vida determinan que el recurso puede ser Deprecado, marcando a las Organizaciones que lo utilizan que en cualquier momento se dejará de utilizar (pero que aún lo pueden utilizar), y finalmente eliminado.

Para poder lograr configurar este ciclo de vida para cada recurso (y cada ambiente), se agregan las entidades de Habilitación y Configuración, que permiten justamente habilitar las

Utilidades por ambiente y su Configuración, que depende de la especificación de Características que tenga cada Capacidad.

Finalmente, para el versionado de los Recursos, se generarán nuevos recursos, con números de versión gestionados por la Organización dueña, generando una relación jerárquica entre Recursos.

3.3 Requerimientos implementados

Los requerimientos fueron definidos en base a los objetivos del proyecto y a partir de las funcionalidades brindadas por productos analizados disponibles en el mercado.

Se enfatizó en proponer una solución que pueda simplificar y mejorar la operativa por parte de los usuarios a la hora de integrar sistemas y que está no dependiera de ninguna tecnología en particular.

Durante el desarrollo de los requerimientos se desarrollaron *mock ups* de las funcionalidades principales que fueron validadas por el cliente, como la creación de plataformas por parte de una organización y la creación de recursos en las plataformas creadas.

Los actores involucrados en el sistema son los siguientes:

- **Publicadores:** usuarios de las organizaciones encargados de gestionar la comunicación con otras organizaciones de forma simple y rápida. Serán los encargados de gestionar las APIs de la plataforma.
- **Usuarios finales:** son aquellos que podrán hacer uso de ciertas utilidades brindadas por la plataforma.
- **Administradores generales:** son los encargados de llevar a cabo la gestión del sistema.

Los requerimientos funcionales definidos son los siguientes:

Requerimiento	Descripción
Login/Logout	Un usuario no autenticado desea iniciar sesión.
ABM Usuarios	Gestión de usuarios del sistema. El sistema permitirá el registro de nuevos usuarios, así como la baja de los mismos y la modificación de sus datos.
ABM Roles	El sistema permitirá la creación, modificación y eliminación de roles del sistema.
ABM Organizaciones	El sistema permitirá el registro de nuevas organizaciones, así como la baja de las mismas y la modificación de sus datos.
ABM Recursos	Gestión de recursos. Los usuarios publicadores podrán dar de alta, modificar y eliminar recursos del sistema. Al crear un recurso se definen las capacidades que tendrá dicho recurso.
ABM Capacidades	Los administradores del sistema podrán dar de alta, modificar y

	eliminar capacidades.
Uso de utilidad	Un usuario final podrá invocar una URI del sistema para utilizar una utilidad desde una aplicación externa al sistema.
Activar/Desactivar recurso	Los publicadores podrán activar y desactivar un recurso que previamente fue dado de alta.
Solicitar recurso	Los publicadores podrán solicitar el uso de un recurso activo.
Compartir recurso	El sistema permitirá ante a un publicador aceptar una solicitud de un recurso que dio de alta.
Alta plataforma	Los publicadores podrán dar de alta plataformas a partir de un template previamente seleccionado.
Configurar capacidad	Los publicadores podrán configurar una determinada capacidad existente en el sistema.
Listar plataformas	El sistema permitirá listar las plataformas previametes ingresadas por un publicador en el sistema.
Listar especificación de plataformas	El sistema permitirá listar los templates de plataforma que están disponibles en el sistema.
Listar recursos	El sistema permitirá listar los recursos ingresados en la plataforma.
Listar recursos activos	El sistema permitirá listar los recursos que se encuentran activos en el sistema.
Listar capacidades	El sistema permitirá listar las capacidades disponibles en el sistema.
Listar organizaciones	El sistema permitirá listar las organizaciones dadas de alta en el sistema.
Listar roles	El sistema permitirá listar los recursos que se encuentran activos en el sistema.
Buscar organización	El sistema permitirá buscar una determinada organización.
Buscar recurso	El sistema permitirá buscar un determinado recurso.
Asignar rol	El sistema permitirá asignar un rol previamente dados de alta a un usuario existente en el sistema.
Versionar recurso	El sistema permitirá mantener varias versiones de un mismo recurso.
Activar/Desactivar capacidad	Los administradores del sistema podrán activar y desactivar una capacidad que previamente fue dada de alta.
Activar/Desactivar utilidad	Los administradores del sistema podrán activar y desactivar una utilidad.

Generar trazabilidad	El sistema registrará la trazabilidad de un usuario registrado.
Consultar trazabilidad	El sistema permitirá consultar la trazabilidad de los usuarios registrados en el sistema.

3.4 Alcance de proyecto

El alcance del proyecto se definió a partir del objetivo y en base a los requerimientos establecidos. Fue definido en conjunto con el tutor del proyecto.

El proyecto consiste en desarrollar una plataforma para brindar soporte a la integración inter-organizacional. Para diseñar una solución fue necesario investigar y estudiar plataformas de integración existentes en el mercado para obtener sus principales funcionalidades y tener un análisis sobre los mecanismos que cada uno de ellos emplean para resolver ciertos requerimientos.

Para determinar el alcance del proyecto se definieron tres escenarios posibles.

A continuación se detallan cada uno de los escenarios.

- **Escenario 1: Creación de plataformas** - El objetivo en este escenario es que una Organización pueda crear una plataforma de trabajo para gestionar sus recursos, centrándose en una plataforma API Manager, en este caso WSO2 API Manager, y en una plataforma ESB, para el cual se seleccionó JBoss Fuse.
- **Escenario 2: Creación de recurso** - En este escenario, una Organización crea un recurso de tipo API REST con autenticación básica y un recurso de tipo SOAP con un *Service Level Agreement* (SLA), seleccionando la plataforma JBoss Fuse previamente creada para realizar la gestión. Además, la misma Organización crea dos recursos seleccionando la plataforma WSO2 API Manager, uno de tipo API REST y otro tipo SOAP, ambos con autenticación básica.
- **Escenario 3: Uso de utilidades** - Finalmente, el último escenario implica el uso de las APIs desde una aplicación externa.

El alcance del proyecto quedó establecido por los tres escenarios mencionados anteriormente.

En una primera instancia se le dio un mayor enfoque al primer y segundo escenario ya que involucran las principales funcionalidades de la plataforma.

3.5 Trabajo relacionado

Al comenzar con este trabajo el avance que tenían las plataformas de iPaaS vistas en el mercado permitían realizar parcialmente algunas de las funcionalidades planteadas luego como requerimiento para este proyecto. Las plataformas analizadas fueron algunas de las expuestas por Gartner en su cuadrante mágico para Plataformas de integración empresariales como servicio [15], que fueron mencionadas en el Capítulo 2.

En particular, las plataformas vistas durante este análisis proponían una gestión de integración de sistemas a través de software propietario, y no permitían al usuario una gestión integral de sus plataformas de integración. Es decir, estos iPaaS permitían integrar diferentes componentes de software, pero en general de manera externa a la plataforma en

sí, y eran ofrecidos en conjunto con otros productos de cada proveedor, como por ejemplo, nubes, para poder realizar el despliegue de estos componentes de software que se quieren integrar.

La plataforma que se está buscando durante el desarrollo de este proyecto busca unificar la gestión de estas plataformas con la gestión de los recursos, a través de una única herramienta de software. A su vez, esta plataforma podría pensarse como una plataforma de integración de plataformas de integración, ya que la misma brinda las capacidades necesarias para que las diferentes plataformas de integración interactúen con el ambiente externo para satisfacer requisitos específicos de integración, e interactúen con otras Plataformas de Integración.

La principal diferencia entre las plataformas existentes en el mercado y la buscada es que las primeras tienen como objetivo proveer capacidades de integración empresarial, mientras que la otra tiene como objetivo la integración de servicios entre organizaciones. Si bien ambos tipos de plataformas tienen similares características, la diferencia en su objetivo primordial hace que la plataforma buscada necesite un conjunto de funcionalidades extra. Estas funcionalidades están relacionadas con proveer las capacidades, en sus diferentes modalidades, que proveen las plataformas de integración actuales del mercado, y además, situarse en una posición de intermediario entre todas las organizaciones que se desean integrar.

Al finalizar el período de desarrollo de este proyecto, alguna de las plataformas del mercado analizadas han evolucionado, como el caso de IBM Bluemix, otros actores han continuado su desarrollo de productos de iPaaS sobre software de tipo PaaS, como en el caso de Red Hat y sus desarrollos de JBoss Fuse sobre Openshift y su instalación en la nube, basado en OpenStack.

Capítulo 4: Diseño de la solución

El objetivo de este capítulo es presentar el diseño de la solución propuesta al problema planteado.

El capítulo se inicia dando una descripción general del sistema a construir, para luego ir profundizando en el diseño de su arquitectura, partiendo de una descripción general, pasando luego a detallar cada uno de los componentes que intervienen en la misma, y por último brindar una vista a nivel físico de la solución.

Finalmente, se enumeran ciertas consideraciones generales que se tuvieron en cuenta a la hora de definir la arquitectura presentada.

4.1 Descripción general de la solución propuesta

La descripción general de la solución se basará en el diagrama de componentes macro que se presenta en la Figura 4.1.

La arquitectura presenta tres componentes básicos:

- Las aplicaciones cliente que representa una aplicación desarrollada por terceros, y que tiene como fin utilizar los recursos publicados en la plataforma
- La plataforma de gestión que implementa el gestor de plataformas y recursos
- Y las plataformas de middleware gestionadas por el sistema generadas por las organizaciones

La solución de la plataforma de gestión que se propone consta de un Gestor de plataformas para el cual se utiliza una solución de plataforma como servicio (PaaS) para la gestión del despliegue de las plataformas pertenecientes a las diferentes organizaciones.

A este PaaS se le agrega una capa de negocio (Core) que permite la gestión centralizada tanto de las plataformas desplegadas en él, como los recursos que se incorporan a cada plataforma. Esta capa de software que se agrega consta de una interfaz web (UI) donde el usuario podrá generar las plataformas y los recursos dentro de cada plataforma, y una capa de servicios de negocio persistente.

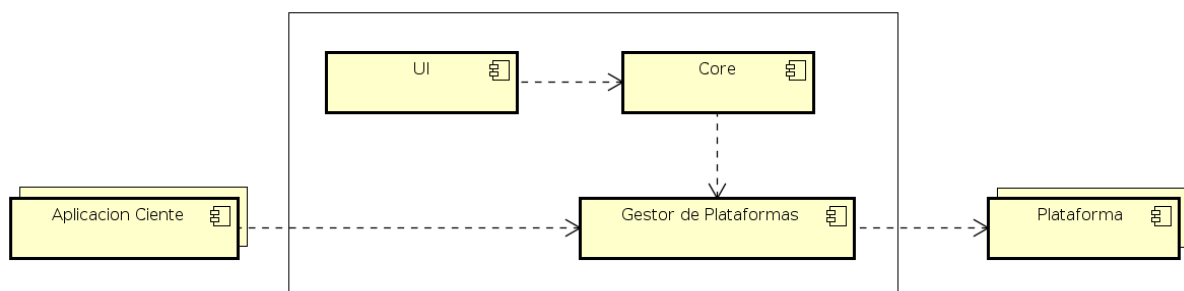


Figura 4.1 - Diagrama de componentes macro de la arquitectura.

Como parte de la solución deseada, también se pensó incorporar al diseño otros componentes que forman parte de algunas de las funcionalidades, pero no fueron implementados dentro de este proyecto por no formar parte del alcance. El detalle del diseño completo se describe en el Anexo VI.

4.2 Vista de negocio

Como se mencionó en la sección 4.1, la solución consta de tres partes: una capa de interfaz de usuario, una capa de servicios de negocio y un gestor de plataformas.

La capa de la interfaz de usuario contiene un componente de interfaz web donde los usuarios de las organizaciones gestionan sus plataformas, crean y comparten sus recursos. Como se muestra en la figura 4.2, aquí se inicia el flujo para el alta de recurso, proceso que se presentará en la sección 4.3.2.

Como muestra el diagrama de la figura 4.2, esta interfaz de gestión (UI) se comunica con la capa de servicios de negocio (Core), que almacena en una base de datos siguiendo el modelo de datos conceptual presentado en el capítulo 2, e implementa la gestión del ciclo de vida de las plataformas utilizando servicios del Gestor de plataformas y la gestión de los recursos dentro de estas plataformas interactuando con ellas (en el diagrama, se ejemplifica con una plataforma de tipo JBoss Fuse).

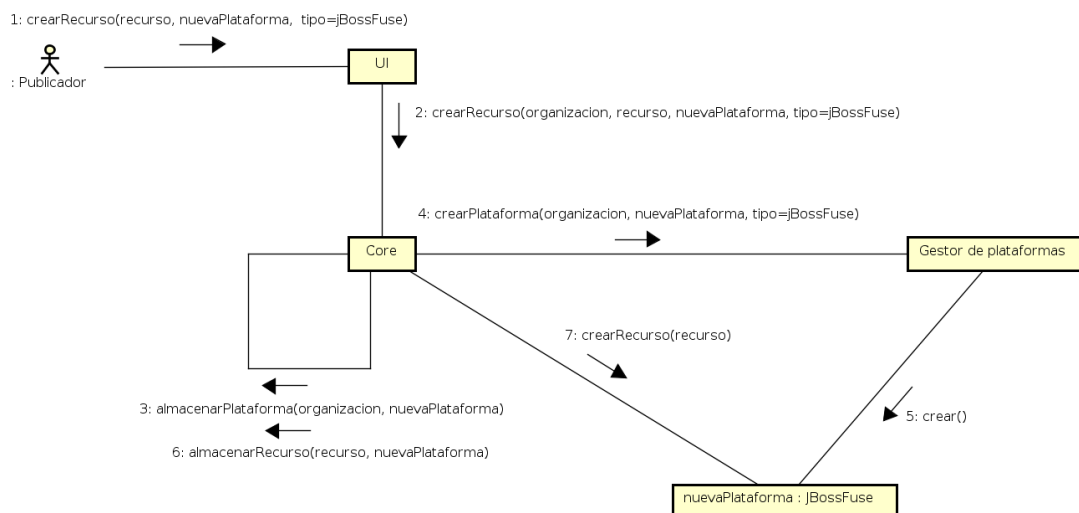


Figura 4.2 - Diagrama de comunicación entre componentes – alta de recurso con alta de plataforma Jboss Fuse.

Finalmente se encuentra el Gestor de plataformas, que brinda una solución para la gestión de las plataformas de las organizaciones:

- Despliegue
- Destrucción
- Chequeos del estado de salud

4.3 Arquitectura de la solución

A continuación se describen los atributos generales deseados en la arquitectura que llevaron a la especificación de la solución planteada en este capítulo.

La primer característica global de la arquitectura es la comunicación entre los componentes, que se realiza a través de APIs REST, lo que permite independizar tanto la implementación como el despliegue de cada uno de los mismos.

El uso de interfaces web amigables permite tener un buen nivel de usabilidad de cara al usuario final. También se tuvo en cuenta que todas las funcionalidades ofrecidas a través de estas interfaces web son también expuestas a través de APIs REST, de manera que el sistema permite la incorporación de clientes a medida o interacción con sistemas de gestión externos.

A nivel del modelo de datos que se desprende del modelo conceptual presentado en el capítulo anterior, el modelado de las capacidades y sus configuraciones fue realizado de manera genérica, permitiendo extensibilidad también a nivel de las diferentes configuraciones que se pueden realizar a la hora de crear nuevos recursos, e incorporar nuevos tipos de plataforma al sistema.

En resumen, como se desprende de la descripción anterior, los principales atributos de calidad que guiaron el diseño de la arquitectura son los siguientes: Interoperabilidad, Usabilidad, Extensibilidad, Modularidad y Mantenibilidad.

En las subsecciones siguientes se presentan diferentes vistas de la arquitectura de manera de poder comprender en mejor medida las decisiones tomadas, y los componentes de la misma.

4.3.1 Descripción lógica

En esta sección se brinda una vista lógica del diseño de la arquitectura, describiendo cada uno de los componentes lógicos que la componen.

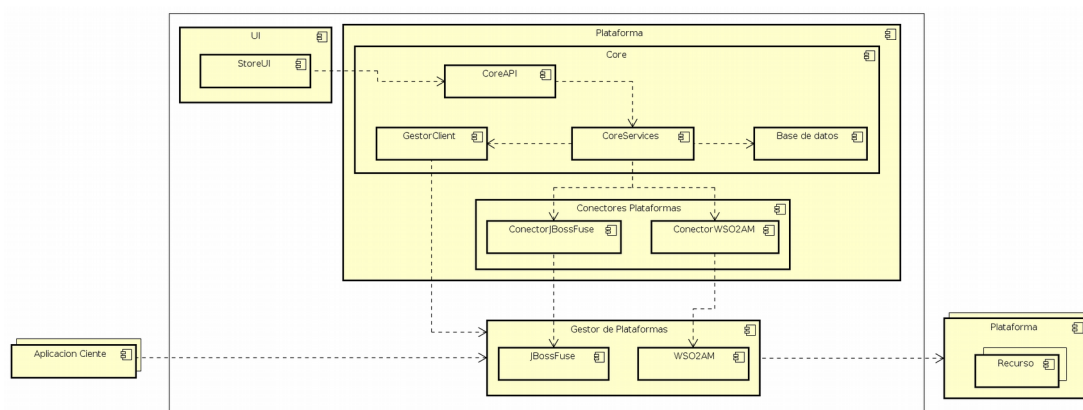


Figura 4.3 - Diagrama de componentes

Como se puede ver en la figura 4.3, los componentes mencionados anteriormente se descomponen en componentes más específicos:

- Interfaz web (UI): contiene un componente web:
 - Interfaz Store UI: Esta interfaz es el componente web que ofrece las funcionalidades necesarias para la creación, visualización y eliminación de plataformas, y para la creación y eliminación de recursos. Se comunica con el Core de negocios a través del uso de sus servicios ofrecidos como APIs REST.

- Core de negocios: este componente implementa los servicios de negocios del sistema, incluyendo la lógica del sistema, la persistencia de los datos, la comunicación con el Gestor de plataformas, y comunicación con cada una de las plataformas creadas. Como se puede ver en la figura 4.3, este Core de negocios contiene los siguientes componentes:
 - Capa de servicios (Core API): se trata de la capa donde se encuentran todos los servicios ofrecidos en forma de API REST, que expone todas las funcionalidades del sistema
 - Capa de negocios (Core Services): es la capa donde se implementa la lógica y la persistencia. Esta capa implementada con servicios empresariales EJB, brinda funcionalidades a la capa de servicios
 - Persistencia (Base de datos): es el componente de base de datos donde se almacena todo el modelo de datos
 - Conector con el Gestor de plataformas (Gestor Client): es un componente que implementa un cliente para la API REST ofrecida por el Gestor de plataformas
 - Conectores con plataformas: es una capa donde se implementan los diferentes conectores con cada uno de los tipos de plataforma soportados por el sistema. Cada conector implementa una interfaz REST unificada, especificada en el Anexo VII, y realiza toda la comunicación necesaria con las plataformas de las organizaciones para gestionar recursos.
- Gestor de plataformas: se trata del componente implementado a través de un producto PaaS que, como se dijo en la sección 4.2, implementa la gestión completa del despliegue de cada una de las plataformas.

4.3.2 Vista de procesos

Dentro de la vista de esta sección se presentan los procesos principales del sistema que fueron implementados en el marco de este proyecto.

Los procesos que se presentarán son la creación de recursos y la creación de plataformas, que son aquellos que fueron considerados como críticos dentro de todas las funcionalidades debido a que definen la funcionalidad de la mayor parte de los componentes de la plataforma. Esta vista de procesos se puede encontrar de forma ampliada en el Anexo VI. Ambos procesos están ligados ya que la creación de plataformas está incluido dentro de la creación de recursos, por la forma que fueron diseñadas las interacciones con el usuario y los requerimientos planteados para el caso de uso de alta de recurso.

El proceso de alta de recurso se inicia con el usuario ingresando al formulario para dar de alta el recurso el cual se puede ver en la figura 4.4. En el mismo, el usuario ingresa los datos básicos del recurso, como ser el nombre, la versión, el tipo de acceso, un identificador que se utilizará para el armado de la uri del recurso.

Luego el usuario selecciona las capacidades que formarán parte del flujo que tendrá el recurso, ingresando la configuración particular para cada capacidad del recurso. Algunas de estas configuraciones pueden ser por ejemplo, la URL de un servicio que se desea invocar

como parte del flujo del recurso, o la cantidad máxima de request permitidos, o bien el tipo de autenticación que se desea para el recurso que se está creando.

Aquí podrá dar de alta nuevos recursos.

Home **Recursos** Organización Plataformas

Creación de nuevo recurso

Nombre:

URI:

Versión:

Tipo de Acceso:

Selección de capacidades

GoJS evaluation
(c) 1999-2016 Northwoods Software
Not for distribution or production use
northwoods.com

Endpoint REST

Endpoint SOAP

Cota de mensajes

Endpoint REST

Throttler

Mapeo de parámetros

Selección de plataforma

Crear nueva plataforma

- ☐ WSO2 API Manager
- ☐ JBoss Fuse ESB
- ☐ Apache ServiceMix ESB
- ☐ Apigee API Manager
- ☐ Active MQ

Utilizar una plataforma existente

- ☐ WSO2 1
- ☐ WSO2 2
- ☐ Fuse 1

Figura 4.4 - Interfaz web para la creación de recursos

Finalmente, el usuario debe seleccionar la plataforma donde se alojará el recurso. Las opciones que se le ofrecen al usuario se clasifican en dos categorías: seleccionar una de las plataformas que la organización a la que pertenece el usuario ya posee creadas y que soportan la totalidad de las capacidades seleccionadas para el recurso, o crear una nueva plataforma de algún tipo de plataforma que soporte las capacidades incluidas en el recurso.

En este punto es donde se incluye el caso de uso de alta de plataforma. En la sección “Selección de plataforma” del formulario web se brinda una opción bajo el título “Crear nueva plataforma” donde se selecciona el tipo de plataforma que se desea crear y se completa con el nombre que se le desea dar a la misma para poder dar de alta una plataforma donde se despliega el recurso.

En el momento que el usuario finaliza la carga de datos en el formulario de la Figura 4.4 descrito, se lanza el proceso de creación, flujo que se puede ver en el Anexo VI donde se detallan los procesos de creación de recurso y creación de plataforma. Las características generales del mismo son las siguientes:

- Se realiza una validación a nivel de base de datos y se almacena en caso de ser exitosa esta validación.
- A través del conector con el Gestor de plataformas, se despliega una nueva plataforma del tipo seleccionado.
- Utilizando el conector con la plataforma del tipo seleccionado, se crea un nuevo recurso con las capacidades configuradas, dentro de la plataforma.

Luego de generar la plataforma y crear el recurso dentro de la misma, la interfaz web despliega al usuario, los datos de la plataforma y del recurso creado.

En caso de ocurrir un error en el proceso, la transacción de base de datos de retrotrae, al utilizarse transacciones de JPA, tanto en el proceso de alta de plataforma, como en el proceso de alta de recurso. En cuanto a la consistencia con el Gestor de plataformas, no fue implementado ningún tipo de validación contra el Gestor de plataformas para verificar o deshacer la creación de la plataforma, ni tampoco contra la plataforma creada con respecto a la creación del recurso.

4.3.3 Vista de despliegue

Dentro de esta vista se describe el esquema de despliegue sugerido para esta arquitectura.

Esta presentación se realiza a partir del diagrama de despliegue de la Figura 4.5, en donde se pueden ver los principales nodos de la aplicación.

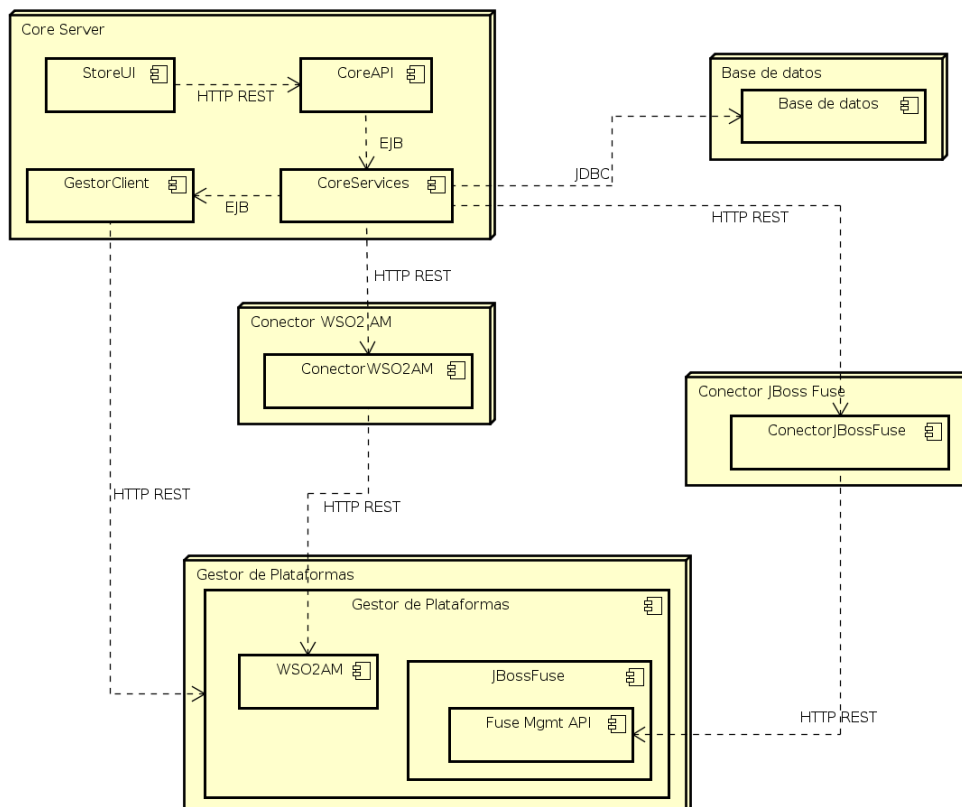


Figura 4.5 - Diagrama de despliegue de la arquitectura

En primer lugar se organizó el nodo al que se llamó Core Server, que nuclea los componentes principales de la interfaz web y de los servicios de negocio del sistema.

Eventualmente, dado que la comunicación entre el Store UI y el Core API se realiza vía API REST, se podría separar esta implementación en un nodo web y un nodo de servicios del core. El resto de las interacciones de los componentes de este nodo se realizan a través de llamadas a EJB.

De forma separada, formando parte del nodo lógico Core, se ubica el nodo de la base de datos.

Por otro lado, debido a las características deseadas en la arquitectura en cuanto a la independencia entre el despliegue de los servicios de negocio y los diferentes conectores con las plataformas, y teniendo en cuenta que la comunicación entre estos componentes se realiza a través de invocaciones a APIs REST, se ubicaron los dos conectores que se implementan en este proyecto de manera separada, como nodos diferentes: el conector con el JBoss Fuse y el conector con el WSO2 API Manager.

Finalmente se puede ver el nodo donde se encuentra el Gestor de plataformas, que ejecuta de manera separada al resto del sistema. Sobre este Gestor es que se despliegan las diferentes plataformas. Para el caso del WSO2 API Manager, se ejecuta el producto sin modificaciones debido a que éste ya ofrece una API REST para la gestión de recursos. Diferente es el caso del JBoss Fuse, para el cual fue necesario incorporar un componente desarrollado por el equipo dentro del propio runtime del Fuse: Fuse Mgmt API. Este componente es el que gestiona las APIs y Servicios dentro del Fuse a través de una API REST.

4.4 Consideraciones de diseño

En esta sección se describen brevemente algunas de las consideraciones generales que se tuvieron en cuenta a la hora de realizar el diseño de la solución.

4.4.1 Comunicación entre componentes

Como se mencionó anteriormente, la comunicación entre componentes se realiza a través de APIs REST, lo que permite independizar tanto su implementación como su despliegue. Esta característica también impacta en la mantenibilidad del sistema y la extensibilidad que se permite en cuanto a la evolución que pueda tener en el futuro.

El uso de APIs REST fue evaluado por encima de otras alternativas como por ejemplo, el uso de servicios empresariales como EJBs, que deja atada la implementación a las tecnologías utilizadas, o el uso de servicios web de tipo SOAP, que en la comparativa con los servicios REST, éstos últimos resultan más livianos y adecuados para el tipo de interacción que se tiene entre los componentes.

4.4.2 Conectores REST con las plataformas

Así como se explicó para la comunicación entre componentes internos del sistema, el diseño de los conectores se realizó utilizando interfaces REST para la comunicación entre el Core y cada conector. De esta manera se logra una independencia de despliegue y de tecnologías que permite implementar los conectores con la tecnología que se desee.

Por otro lado, este enfoque permite también lograr la extensibilidad necesaria para permitir la incorporación de nuevos tipos de plataforma, sin necesidad de afectar el funcionamiento del sistema.

4.4.3 Conectores con las plataformas como parte de la solución

Otra decisión que fue tomada para el diseño de los conectores con las plataformas fue la incorporación de los conectores a la solución, de esta manera se logra generar una interfaz única desde el punto de vista desde el Core del sistema, y permitiendo que cada conector implemente el protocolo de comunicación y lógica de administración necesaria para cada producto.

Como alternativa se manejó que el conector fuese parte de la plataforma en sí, pero esta opción fue descartada considerando que no todos los productos API Manager y ESB tienen la misma interfaz por API, incluso encontrando varios de ellos sin interfaz de administración por API.

4.4.3.1 Procedimiento para agregar un nuevo conector

La arquitectura desarrollada para la plataforma y las decisiones tomadas respecto al diseño de los conectores fueron guiadas, entre otras características, para darle extensibilidad a la solución a la hora de incorporar nuevos tipos de plataforma. Este atributo de la arquitectura puede verse reflejado en el procedimiento que se debe seguir al agregar el conector para el nuevo tipo de plataforma. El procedimiento es el siguiente:

1. Se desarrolla un servicio REST que implemente la interfaz genérica que se especifica en el Anexo VII. Este servicio hará de interlocutor entre el Core de negocios y las plataformas levantadas del nuevo tipo para la gestión de los recursos.
2. Este servicio REST desarrollado se deberá desplegar en un ambiente alcanzable por el Core de negocios.
3. Se configura luego la URL del conector en la especificación del nuevo tipo de plataforma en la base de datos del Core de negocios.

De esta manera, el nuevo conector queda preparado para ser utilizado para la gestión de los recursos en las plataformas del tipo creado.

4.4.4 Gestor de plataformas

Dentro de esta misma sección se hizo mención al Gestor de plataformas como un componente pensado para administrar el despliegue y el ciclo de vida de las plataformas, y que esta característica era propia de un software de tipo PaaS.

Como alternativa inicial se planteó el uso de diferentes tecnologías, como Vagrant o Docker para el despliegue de las mismas, necesitando un desarrollo dentro de este proyecto como parte del sistema que permita gestionar los despliegues con cualquiera de estas dos tecnologías.

Finalmente se optó por el uso de un producto de tipo PaaS que ya resuelve esta problemática, y agrega funcionalidades de gestión que pueden ser utilizadas a futuro durante el mantenimiento del sistema.

Una vez tomada esta decisión, otra característica que debía tener el producto PaaS que se seleccionara era la presencia de una administración vía API, de manera de poder automatizar la gestión de las plataformas dentro del PaaS desde el Core del sistema.

Capítulo 5: Implementación

En este capítulo se presenta detalladamente la implementación de los componentes más relevantes de la plataforma con las tecnologías utilizadas para dicha implementación. Se describen las principales decisiones tomadas a la hora de implementar diferentes funcionalidades, principalmente aquellas funcionalidades y características consideradas clave en la arquitectura propuesta.

5.1 Tecnologías utilizadas

En la Figura 5.1 se muestra el diagrama de componentes presentado en el capítulo de diseño, con las tecnologías utilizadas para la implementación de cada uno.

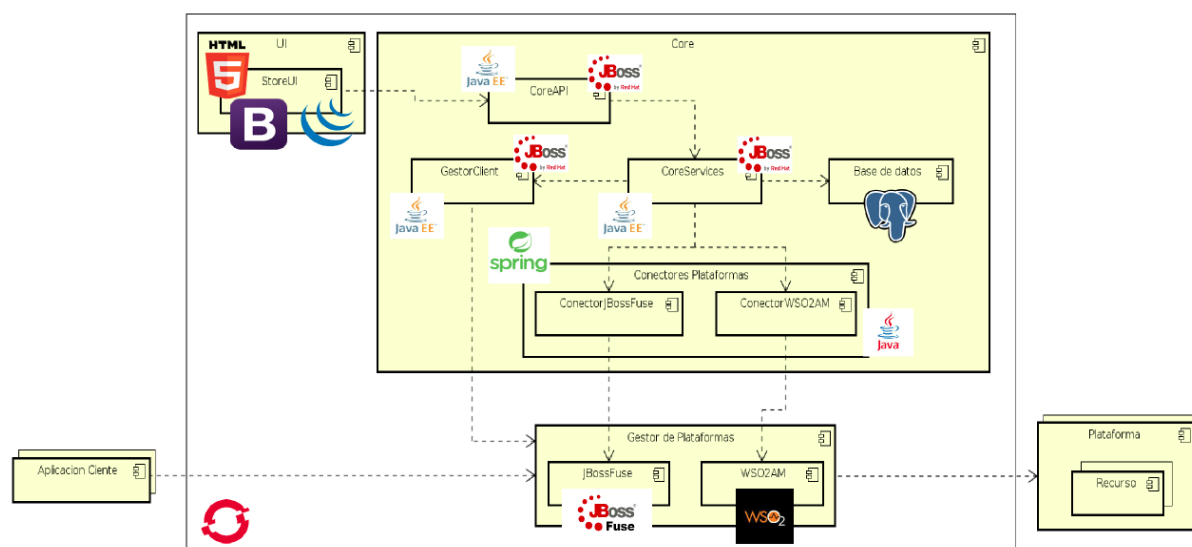


Figura 5.1 - Diagrama de componentes con tecnologías utilizadas.

5.1.1 Interfaz gráfica de usuario

La interfaz de usuario fue implementada únicamente utilizando HTML5 y JQuery, invocando servicio REST del Core a través de Ajax. Estas herramientas fueron seleccionadas de manera de reducir los tiempos en la etapa de implementación, debido a que se consideraba que la utilización de algún otro tipo de framework web, u otras herramientas, requeriría cierto tiempo de dedicación disponible para transitar sus curvas de aprendizaje.

La selección de los estilos y la parte visual de las páginas HTML confeccionadas para la demo se realizaron utilizando el framework Bootstrap como base, debido a la experiencia previa que el equipo contaba en el uso de esta herramienta.

Para generar la visualización de los flujos de capacidades que puede tener un recurso se seleccionó GoJS que es una herramienta para la edición de diagramas utilizando JQuery y HTML5. Si bien es una herramienta paga, se puede utilizar en forma trial pero teniendo una marca de agua dentro del canvas donde se utilice. La elección de esta herramienta para la

implementación de la interfaz de usuario de la demo fue debido a que existía experiencia dentro del equipo con el uso de esta herramienta, y no se analizaron otras alternativas para evitar los tiempos que lleva la introducción de nuevas herramientas.

5.1.1.1 Alternativas a GoJS para la visualización de flujos

La alternativa mejor valorada según las revisiones de funcionamiento realizadas para una solución completa es la herramienta desarrollada por IBM Node-RED. Se trata de una herramienta de código abierto desarrollada utilizando NodeJS e implementa visualmente la posibilidad de desarrollar flujos de actividad entre diferentes servicios, mayormente orientado a soluciones en la nube sobre internet de la cosas [59].

Esta herramienta se descartó para este proyecto debido a que si bien se contaba con cierto conocimiento teórico, no se contaba con la experiencia suficiente que se necesitaba para realizar un desarrollo dentro de los tiempos necesarios para culminar con el proyecto.

5.1.1.2 Alternativas para la implementación de la interfaz gráfica de usuario

Se evaluó comenzar la implementación de la interfaz web utilizando un framework web de Java, en particular, en base a la experiencia previa del equipo, se evaluó el uso de Wicket de Apache. La ventaja principal del uso de un framework web de este tipo es que la lógica de conversión de datos, y parte de la lógica de visualización puede ser resuelta ejecutando del lado del servidor, y no del cliente, como en la versión implementada con HTML y JQuery. Sin embargo, las herramientas utilizadas fueron HTML5 y JQuery, porque nos permitieron minimizar los tiempos de implementación como se comentó anteriormente, a diferencia de Wicket que nos implicaba un mayor esfuerzo y por lo tanto, no poder cumplir con los tiempos planificados.

5.1.2 Core

El componente Core fue implementado como una aplicación Java Empresarial en capas, utilizando servicios empresariales como EJB, desplegados en un servidor de aplicaciones JBoss EAP, y exponiendo servicios REST para el uso desde la interfaz web. Se seleccionó JBoss EAP para el despliegue de los artefactos del prototipo desarrollado debido a la gran experiencia que existía dentro del equipo con este tipo de tecnologías.

Para la integración con el Gestor de Plataformas, para el cual se utilizó la tecnología OpenShift Origin y cuya implementación se describe en la sección 5.3, se utilizó el cliente Java REST para la arquitectura de la versión 3 de OpenShift basada en Kubernetes. Este cliente es un trabajo en progreso para proporcionar, a través de una API programática, funcionalidad y características similares de la interfaz de línea de comandos y es utilizada por JBoss Tools para OpenShift. Dicho cliente fue utilizado para crear, editar, borrar y obtener información de la plataformas creadas.

5.1.2.1 Alternativas

La alternativa que se manejó en un principio para este componente central es el uso de un contenedor capaz de ejecutar Apache Camel, como por ejemplo, Apache Karaf, o JBoss Fuse, para desplegar los servicios, que serían implementados con rutas Apache Camel, que brinda una interfaz apropiada para la orquestación de servicios, tal y como es requerido en el Core de la solución.

Se optó por una aplicación Java Empresarial en capas en lugar de una aplicación basada en flujos con Apache Camel, porque nos permitió enfocarnos fácil y rápidamente en las funcionalidades del negocio de la plataforma. A su vez, la aplicación resulta más liviana que lo que sería en el otro caso, aunque la alternativa podría brindar una forma automatizada de despliegue en caso de implementar los servicios como bundles.

5.1.3 Gestor de Plataformas

La plataforma OpenShift Origin fue seleccionada como tecnología para implementar el Gestor de Plataformas, y sobre el cual se despliegan y desplegarán todos los componentes que forman parte de nuestro sistema. Los principales motivos por los cuales decidimos utilizar esta tecnología se enumeran a continuación:

- Orquestación de containers con Kubernetes
- Almacenamiento de containers
- Automatización
- Tecnología open source.
- Experiencia de integrantes del equipo trabajando con OpenShift.

OpenShift Origin fue seleccionado por el equipo para poder incorporar al sistema la capacidad de gestionar las implementaciones de las diferentes plataformas que soportaría el sistema. Openshift brinda capacidades de despliegue, de gestión de instancias, de chequeos de salud, de actualización y de destrucción de contenedores de manera sencilla, eficaz y automatizable.

Otro de los principales motivos por los cuales esta tecnología fue seleccionada es que provee un conjunto de APIs REST para la gestión de los servicios, instancias, accesos y demás conceptos que se utilizan en la implementación de la gestión de plataformas. Estas APIs fueron probadas a través de pruebas realizadas utilizando Postman como herramienta, para luego implementar un prototipo desarrollado en Java previo a la incorporación al Core de negocios.

5.1.4 Conectores

Los conectores con las diferentes plataformas soportadas fueron implementados utilizando Spring Boot para su despliegue, que brinda un marco de trabajo liviano en Java, en particular utilizando Spring para el desarrollo de los servicios. Esta tecnología fue seleccionada ya que el equipo contaba con experiencia en la misma y permite desplegar, para los efectos del proyecto, un contenedor web liviano sobre una JVM disponibilizando APIs RESTful sobre el framework Spring.

Las interfaces diseñadas para la incorporación de los diferentes conectores necesarios para operar sobre las plataformas gestionadas por el sistema fueron implementadas como APIs REST, ya que nos permite tener independencia de tecnología o lenguaje en la que se implementen los conectores, escalabilidad y flexibilidad. Además, esta implementación cumple con el diseño de estas interfaces, la cuales fueron definidas de modo que se pueda independizar completamente la implementación y despliegue del sistema, con la del conector.

En particular, para las plataformas utilizadas para esta etapa del desarrollo de la solución, se utilizó una interfaz REST implementada con Spring y desplegada a través de Spring Boot. Este Spring Boot genera un artefacto JAR (Java Archive) autocontenido que al ejecutarlo, despliega un contenedor web (en el caso de los ejemplos implementados, se optó por Jetty, pero nativamente también se puede incorporar Tomcat), y dentro de dicho contenedor web, el servicio implementado. Este servicio fue ejecutado luego en un contenedor Docker, que para las pruebas individuales pudo ser ejecutado localmente, pero para la operativa del sistema, fue desplegado sobre el mismo Openshift que gestiona las plataformas.

La interfaz seleccionada, especificada en el Anexo VII, permite a su vez que esta implementación seleccionada no sea la requerida para poder funcionar el proyecto completo, sino que permite que sea implementado en el lenguaje que se desee, utilizando las tecnologías más convenientes para el desarrollador de dicho conector.

Resumiendo la especificación presentada en el Anexo VII, esta interfaz ofrece cinco operaciones que permiten la creación de recursos, la modificación y eliminación de los recursos creados, así como también una operación para generación de tokens de seguridad para el uso de los recursos requerido por cierto tipo de plataformas, en el caso de las seleccionadas, WSO2 API Manager requería que se implementara esta operación, pero JBoss Fuse no, y por último una operación “echo” que permite consultar por el estado de salud del conector.

5.1.4.1 Alternativas

Como alternativa principal para la implementación de los conectores con las plataformas se manejó el uso de Bundles para Karaf (o en su defecto, Fuse), en el caso de realizar una implementación del Core también sobre este tipo de contenedores OSGI. Esta alternativa fue desechada debido a que la opción implementada permitía mayor independencia en la implementación y el despliegue de los diferentes conectores, que la opción de implementación con Bundles OSGI no brindaba.

5.1.5 Plataformas

A continuación se mencionan las tecnologías seleccionadas como ejemplos de Plataformas, un API Management y un ESB. Estas herramientas fueron seleccionadas tomando como referencia la experiencia con la que se contaba dentro del equipo y prototipadas para analizar algunas de sus funcionalidades.

5.1.5.1 API Management

La elección del API Manager de WSO2 como ejemplo a implementar en el caso de las plataformas de API Management fue tomada pensando en la API REST que éste producto ofrece nativamente, que permite la automatización de la gestión de los recursos desde un sistema externo, en este caso, el sistema que se implementa.

5.1.5.2 ESB

Como ejemplo de plataforma de tipo ESB para el prototipo del sistema se seleccionó JBoss Fuse, en base a la experiencia previa que el equipo tenía en esta plataforma, y teniendo en cuenta la capacidad de realizar algún desarrollo en Apache Camel como se explicó en la

sección 5.1.2.1 para la incorporación de las funcionalidades necesarias para el sistema dentro de una plataforma de este tipo.

En primera instancia se implementó un prototipo con Apache Camel instalado sobre JBoss Fuse, que permitía probar que las funcionalidades que se deben agregar a este producto para la incorporación al sistema de plataformas implementado en este proyecto era viable.

Apache Camel fue utilizado para generar el punto de entrada a la plataforma, es decir, se utilizaron las capacidades de esta tecnología para incorporar los elementos de software necesarios para que el sistema implementado se pudiese comunicar con JBoss Fuse en tiempo de ejecución.

5.2 Herramientas de desarrollo

En primer lugar, se seleccionó Git como herramienta a utilizar como repositorio del código en base a la disponibilidad que se contaba dentro de la infraestructura del proyecto, y en base también a la experiencia del equipo en el uso de la misma. En particular, se contaba con una instalación de Gitlab dentro de la infraestructura de la Facultad de Ingeniería.

El lenguaje de programación seleccionado para la implementación fue Java, debido a la experiencia que el equipo presentaba sobre múltiples tecnologías en dicho lenguaje que permiten el rápido desarrollo de ciertos componentes esenciales dentro de la arquitectura propuesta.

Para poder permitir la futura automatización de los procesos de desarrollo, testing y despliegue de los componentes, Apache Maven provee herramientas para la construcción de los artefactos con los que el equipo presentaba experiencia.

Para el despliegue de varios componentes livianos, de manera independiente, se optó por Docker como contenedor a utilizar. También se utiliza Docker dentro del proyecto para el despliegue de las diferentes plataformas de las organizaciones, aunque para estos despliegues, se gestiona la implementación de los contenedores a través de Openshift.

5.3 Dificultades y limitaciones encontradas

En esta sección se describen las principales dificultades y limitaciones enfrentadas durante la implementación del sistema y las pruebas a la que fue sometida la implementación del sistema.

- OpenShift es una tecnología relativamente nueva que se encuentra en pleno desarrollo y se compone de tecnologías (Docker y Kubernetes) con similares características. Si bien se ha ido estabilizando con el transcurso del proyecto, se han encontrado algunas dificultades operativas con el uso de la versión inicial, como por ejemplo el excesivo consumo de disco y la utilización de imágenes de Docker con errores de funcionamiento, entre otros.
- El cliente Java para la API REST de OpenShift no es oficial y su documentación es muy básica. Esto nos implicó realizar pruebas más específicas de investigación del cliente, en la cuales se detectó que no contaba con todos los métodos que necesitábamos para la integración. Por lo tanto, dichos métodos fueron implementados por nosotros en base a la documentación oficial de OpenShift Origin.

- Se encontraron pocos productos de API Management de código abierto como alternativas para seleccionar e incorporar a esta versión de la plataforma. Los productos de código abierto encontrados fueron: API Umbrella, Fusio, ApiAxle y WSO2 API Manager que fue el utilizado.
- Las APIs ofrecidas por WSO2 API Manager están tecnológicamente desactualizadas con respecto a los estándares REST, así como también poco funcionales e intuitivas. Por ejemplo, cuando se desea acceder a cualquier API a través de un cliente REST externo, se debe invocar primero a la API de *login* para garantizar que el usuario esté autenticado y el sistema almacena la *cookie* de sesión generada en un archivo, la cual se utiliza en las próximas invocaciones de la API. Hoy en día, estas APIs ya están obsoletas y no serán soportadas en futuras versiones.
- Ninguno de los ESB analizados ofrecían una herramienta de administración vía API, por lo que se optó seleccionar JBoss Fuse e incorporar en la distribución del docker una API desarrollada por el equipo.
- Para los recursos desplegados sobre la plataforma Jboss Fuse se optó por mantener un puerto diferente por cada recurso debido a que se nos presentaron dificultades para implementar la autenticación básica.

5.4 Testing

En esta sección se describe el ciclo de testing realizado.

El primer punto describe las pruebas realizadas detallando qué componentes son afectados, los resultados esperados y obtenidos. Mientras que el segundo punto detalla las pruebas a las que debería ser sometida la plataforma para asegurar la calidad del producto.

5.4.1 Metodología de testing

A continuación se describe los pasos realizados para realizar las pruebas sobre la implementación de cada uno de los componentes de la plataforma:

1. Para el caso de las APIs de WSO2, Openshift y la generada para el Jboss Fuse, se testearon las funcionalidades que se necesitaban a través de Postman.
2. A medida que se realizaba la implementación de los conectores con las plataformas WSO2, Jboss Fuse y Openshift, se ejecutaba un testing unitario de las funcionalidades implementadas de cada componente. En los tres casos, se optó por no utilizar un mock de sus APIs debido a que las funcionalidades están muy ligadas al comportamiento de dichos componentes.
3. Con la implementación completa de los conectores de WSO2 y Jboss Fuse, que exponen APIs REST, se realizó un testing a través de Postman de dichas APIs.
4. Luego se realizó un testing unitario de los componentes de software del Core de negocios que utilizaban dichos componentes.
5. Posteriormente, se realizó el testing unitario de los EJB de negocio que encapsulaban las funcionalidades implementadas para el prototipo, con los conectores integrados, y las funcionalidades de persistencia a través de JPA.

6. La siguiente fase de pruebas fue invocar con Postman la API REST del Core que utiliza los EJBs probados, probando la integración desde la API, pasando por los conectores y terminando en las diferentes plataformas.
7. Finalmente, se probó manualmente la funcionalidad completa desde la Interfaz Web.
8. Como última fase del testing, se realizó una automatización de las pruebas realizadas en el paso 6.

Como se mencionó, esta sucesión de pruebas fue realizada siempre utilizando los componentes integrados, desplegados en el servidor de aplicaciones seleccionado para esta etapa.

En esta ocasión no se realizaron otros tipos de test, como los tests de seguridad, los tests de performance y los tests de carga.

5.4.2 Herramientas utilizadas

En primer lugar, se listan las herramientas utilizadas para llevar a cabo las pruebas:

- **JUnit:** es un framework para la ejecución de test unitarios en java. [60]
- **Postman:** herramienta que permite probar servicios que exponen una API REST. [61]
- **SoapUI:** herramienta que permite probar de forma ágil web services SOAP y REST. Permite la realización de test unitarios, de performance y de carga. [62]
- **Selenium:** herramienta que permite realizar pruebas automatizadas en interfaces web [63]. Algunas de las utilidades de esta herramienta son:
 - Crear conjuntos y pruebas de automatización regresivas, robustas y basadas en el navegador.
 - Escalar y distribuir scripts a través de muchos ambientes.

El principal testing realizado a la solución fue el testing funcional. Para realizar dichas pruebas, se dividió la plataforma en los siguientes componentes:

- **Interfaz web:** En primer lugar se decidió que las pruebas de la interfaz web se iban a realizar manualmente debido a que eran pocas funcionalidades las que se iban a implementar en esta etapa. Pero luego, se implementó la automatización de alguna de las pruebas realizadas con la herramienta Selenium, ya que es una mejor estrategia, permitiendo validar y verificar que futuros cambios no afecten las funcionalidades actuales.
- **Core:** Una de las estrategias llevadas a cabo para testear el core fue realizar pruebas con la plataforma integrada, desde la API del Core hasta la plataforma Openshift, tanto para el manejo de la persistencia, como para la implementación de la lógica de control. En este caso, al tratarse de APIs, se utilizó Postman como herramienta para invocar los servicios.
- **Conectores con las plataformas:** Los conectores con las plataformas ofrecen interfaces API REST por lo que fueron testeadas de manera manual utilizando

Postman. Estas pruebas fueron realizadas con las plataformas correspondientes levantadas y respondiendo a través de sus APIs, de manera de evitar el desarrollo de mocks para sustituirlas, y de esta manera, poder corroborar que las funcionalidades logran los efectos deseados sobre las mismas.

- **Plataforma como servicio utilizada - OpenShift:** Los componentes relacionados con la integración de OpenShift son componentes EJB del core que implementan la conexión con OpenShift. Las pruebas respecto al funcionamiento que se deseaba de las APIs de Openshift se realizó utilizando Postman.
- **Testeo de las APIs de los productos ESB y API Manager seleccionados:** Como todas las APIs, éstas también fueron testeadas utilizando Postman. Tanto para la API provista por WSO2, como la API desarrollada para Jboss Fuse, se testearon con una instancia de cada producto desplegado, de manera de probar que la creación de recursos se realizaba de la manera deseada.
- **Testing automatizado de la web:** Finalmente se seleccionó una de las funcionalidades para implementar el test automatizado utilizando Selenium, quedando cubierto el testing desde la web hasta el impacto en la base de datos, en el gestor y en las plataformas.

5.4.3 Pruebas realizadas

Los escenarios implementados fueron el alta de recurso y el alta de plataforma, teniendo en cuenta según lo visto en la sección 4.3.2, que el proceso de alta de recurso a su vez incluye el proceso de alta de plataforma.

Por lo tanto, las funcionalidades sobre las que se realizaron las pruebas fueron las siguientes:

- Creación de plataforma, en conjunto con las funcionalidades anexas listado de plataformas y visualización de los datos de una plataforma.
- Creación de recurso, en conjunto con la creación de plataforma, la selección de plataformas existentes y el listado de plataformas

Si bien fue implementada para facilitar algunas tareas a la hora de realizar las pruebas, la eliminación de plataformas fue probada solamente en su flujo normal, no generando casos para los flujos alternativos.

5.4.4 Resultados de las pruebas

Para la presentación de los resultados de las pruebas se utiliza la tabla 5.1, donde se resume por cada funcionalidad en cada interfaz probada, los casos más relevantes.

Funcionalidad	Interfaz / componente	Caso(s) de prueba	Resultado esperado	Resultado obtenido
Creación de plataforma WSO2 API Manager	API de Openshift	Crear plataforma de nombre wso2 .	La plataforma wso2 se crea	El esperado
Creación de plataforma Jboss Fuse	API de Openshift	Crear plataforma de nombre fuse .	La plataforma fuse se crea	El esperado
Recuperación del estado de una plataforma	API de Openshift	- Plataforma fuse - Plataforma wso2	Se obtiene el estado de la instancia fuse o wso2 según el caso.	El esperado
Eliminación de una plataforma	API de Openshift	- Plataforma fuse - Plataforma wso2	La instancia de la plataforma se elimina	El esperado
Crear recurso REST	API Jboss Fuse	- Crear API REST (con y sin throttling, con y sin autenticación basic)	Se crea el recurso seleccionado con las capacidades opcionales seleccionadas	El esperado
Crear recurso SOAP	API Jboss Fuse	- Crear WS SOAP (con y sin throttling, con y sin autenticación basic)	Se crea el recurso seleccionado con las capacidades opcionales seleccionadas	El esperado
Crear recurso REST	API WSO2 API Manager	- Crear API REST (con y sin throttling, con y sin autenticación basic)	Se crea el recurso seleccionado con las capacidades opcionales seleccionadas	El esperado, excepto con el uso del throttling ya que las opciones que brinda WSO2 para esta capacidad son limitadas
Crear recurso SOAP	API WSO2 API Manager	- Crear WS SOAP (con y sin throttling, con y sin autenticación basic)	Se crea el recurso seleccionado con las capacidades opcionales seleccionadas	El esperado, excepto con el uso del throttling ya que las opciones que brinda WSO2 para esta capacidad son limitadas

Crear recurso REST	API Conector Jboss Fuse	- Crear API REST (con y sin throttling, con y sin autenticación basic) en el fuse identificado	Se crea el recurso seleccionado con las capacidades opcionales seleccionadas en la plataforma seleccionada	El esperado
Crear recurso SOAP	API Conector Jboss Fuse	- Crear WS SOAP (con y sin throttling, con y sin autenticación basic) en el fuse identificado	Se crea el recurso seleccionado con las capacidades opcionales seleccionadas en la plataforma seleccionada	El esperado
Crear recurso REST	API Conector WSO2 API Manager	- Crear API REST (con y sin autenticación basic) en el wso2 identificado	Se crea el recurso seleccionado con las capacidades opcionales seleccionadas en la plataforma seleccionada	El esperado
Crear recurso SOAP	API Conector WSO2 API Manager	- Crear WS SOAP (con y sin autenticación basic) en el wso2 identificado	Se crea el recurso seleccionado con las capacidades opcionales seleccionadas en la plataforma seleccionada	El esperado
Crear Plataforma	API Core	- Crear wso2 - Crear fuse	Se crea la plataforma seleccionada	El esperado
Crear Recurso	API Core	- Crear REST en wso2 - Crear SOAP en wso2 - Crear REST en fuse - Crear SOAP en fuse	Se crea el recurso en la plataforma seleccionada	El esperado
Obtener información de una plataforma	API Core	Obtener información de una plataforma creada	Se obtiene el estado de la plataforma desplegada e información sobre los datos de la plataforma	El esperado

Eliminar plataforma	API Core	- Eliminar una plataforma con recursos - Eliminar una plataforma sin recursos	Se elimina la plataforma con todos sus recursos	El esperado
Crear Plataforma	Interfaz Web	- Crear REST en wso2 - Crear SOAP en wso2 - Crear REST en fuse - Crear SOAP en fuse	Se crea el recurso en la plataforma seleccionada	El esperado
Ver datos de plataforma	Interfaz Web	Obtener información de una plataforma creada	Se obtiene el estado de la plataforma desplegada e información sobre los datos de la plataforma	El esperado
Eliminar plataforma	Interfaz Web	- Eliminar una plataforma con recursos - Eliminar una plataforma sin recursos	Se elimina la plataforma con todos sus recursos	El esperado

Crear recurso	Interfaz Web	- Crear REST en wso2 existente (con y sin autenticación basic) - Crear SOAP en wso2 existente (con y sin autenticación basic) - Crear REST en un nuevo wso2 (con y sin autenticación basic) - Crear SOAP en un nuevo wso2 (con y sin autenticación basic) - Crear REST en fuse existente (con y sin throttling y auth basic) - Crear SOAP en fuse existente (con y sin throttling y auth basic) - Crear REST en nuevo fuse (con y sin throttling y auth basic) - Crear SOAP en nuevo fuse (con y sin throttling y auth basic)	Se crea el recurso con las capacidades seleccionadas en la plataforma seleccionada (se crea la plataforma si no existe)	El esperado
---------------	--------------	--	---	-------------

Tabla 5.1 – Resumen de los resultados de las pruebas.

Capítulo 6: Gestión del proyecto

En este capítulo se presenta el proceso que el equipo llevó adelante para la realización de este proyecto.

6.1 Organización

Para la realización de este proyecto, se plantearon reuniones cada dos semanas con el tutor, de esta manera iterar el trabajo en períodos de trabajo cortos, y planteando objetivos realistas en cada período. Luego el equipo optaba por la división de tareas, lo más pequeñas posibles, para luego integrar todo el trabajo realizado previo a cada reunión con el tutor.

Por otro lado, existían ciertos conceptos y requerimientos que debían ser aclarados con el cliente del proyecto, por lo que se definieron algunas reuniones para aclarar y definir lo necesario para continuar con el mismo.

De esta manera, se podía hacer un seguimiento más minucioso del avance del proyecto, teniendo en cuenta también que existían ciertos requerimientos que podían ir cambiando durante el transcurso del proyecto, que habían conceptos que aún no estaban del todo claros para el equipo y que no se tenía un conocimiento real de la dimensión que podía tener la construcción de una solución para el problema completo.

Si bien este ritmo de trabajo fue el deseado durante todo el proyecto, existieron momentos donde no se pudo mantener, debido a diferentes factores externos que afectaron a todos los integrantes del proyecto: licencias y viajes, carga de trabajo por responsabilidades laborales, responsabilidades para con otras materias y estudios, problemas personales y enfermedades.

6.2 Planificación del proyecto

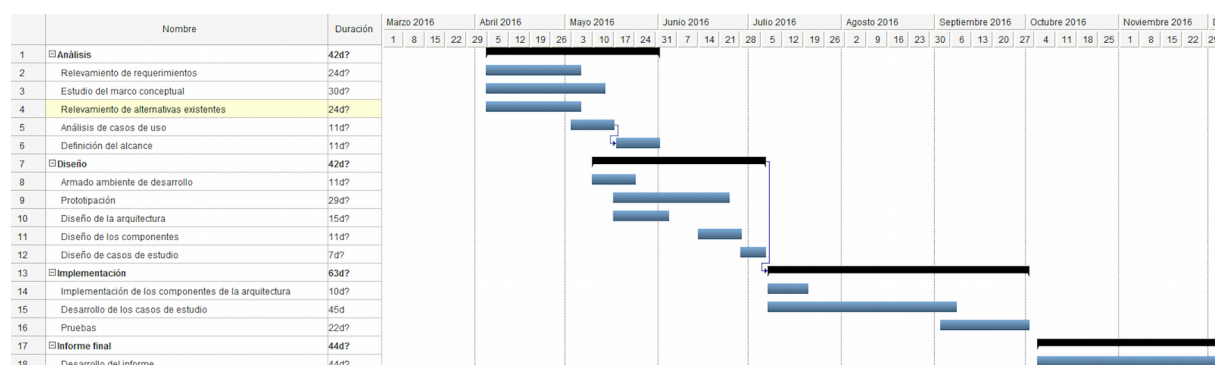


Figura 6.1 - Diagrama de Gantt con lo planificado originalmente.

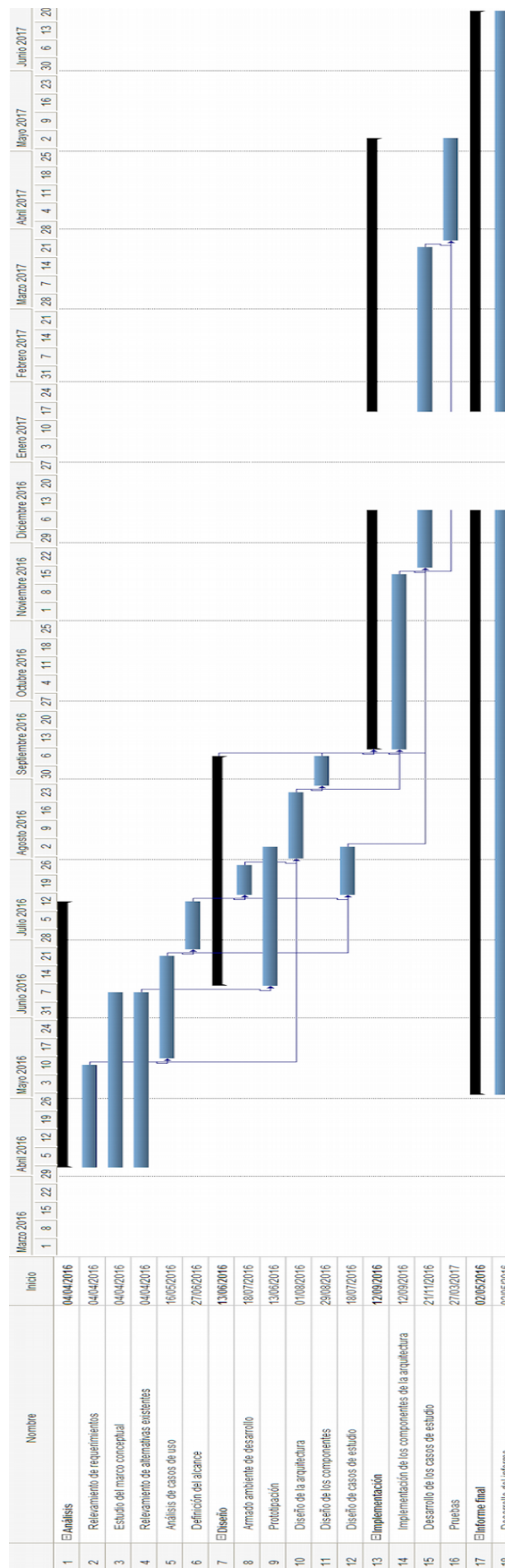


Figura 6.2 - Diagram de Gantt con los tiempos efectivos dedicados a cada actividad.

Durante la etapa de análisis del proyecto se elaboró una planificación general del proyecto, de manera de tener una estimación de los tiempos que éste implicaría, como muestra el diagrama de Gantt de la Figura 6.1.

Debido a lo explicado en la sección anterior sobre los factores de desvío respecto a la planificación inicial, se elaboró el diagrama de la Figura 6.2 que muestra el transcurso real del proyecto. En dicho diagrama se puede ver tanto las tareas que tuvieron un desvío, como el período en el cual el equipo se encontraba de licencia y no se ejecutaron tareas.

Por otro lado, la definición de las tareas fue realizada utilizando el correo electrónico como herramienta para mantener comunicado al grupo lo definido en cada reunión con el tutor, y/o con el cliente, en base a las tareas generadas en la herramienta online Trello.

6.3 Etapas del proyecto

El proyecto estuvo dividido en cuatro etapas. En esta sección se hará mención a cada una de ellas, describiendo brevemente sus principales características, principales tareas, sus resultados y cuáles fueron los problemas enfrentados, así como también de qué manera se solucionaron.

6.3.1 Investigación inicial y análisis del proyecto

Dadas las características del proyecto, la primera etapa implicó realizar una investigación inicial acerca del estado del arte sobre el tipo de software que se deseaba construir. Se consideraba que el proyecto trataba de una implementación de referencia de un Software de integración de plataformas, es decir un iPaaS, pero que tiene como principal característica la gestión de plataformas alojadas dentro del propio iPaaS.

Paralelamente a esta investigación, esta primer etapa también incluía un análisis inicial del proyecto, partiendo del artículo [1] de Laura González: *"A reference architecture for integration platforms supporting cross-organizational collaboration"*. En este análisis se pretendía generar un conocimiento previo de los conceptos involucrados en el proyecto, de los requerimientos que debían cumplirse para finalmente definir el alcance.

Las principales características de la investigación realizada eran: en primer lugar, el análisis y la búsqueda de los diferentes conceptos que serían necesarios para el entendimiento futuro del proyecto, y en segundo lugar, el análisis de los diferentes productos de software disponibles en el mercado.

Esta etapa originalmente fue planificada para ser realizada aproximadamente en los primeros 2 meses, pero dadas las características del proyecto, y de los análisis que se debían realizar, se extendió a poco más de 3 meses. Durante este tiempo, también se tuvo un período donde se atrasó la planificación debido a que se dio en el medio un período de exámenes y entregas de otros proyectos.

Dada la extensión en la duración de esta etapa, y como consecuencia, la falta de definición a tiempo del alcance del proyecto, lo que se optó fue por comenzar a trabajar paralelamente en el armado de prototipos, planificados para la etapa de diseño, de manera de poder acelerar en la toma de decisiones de las etapas siguientes del proyecto.

6.3.2 Diseño de la solución propuesta

Luego de tener claros los conceptos involucrados en el proyecto, y los productos disponibles en el mercado, se planteaba la tarea de profundizar en los conceptos y requerimientos del proyecto, para dar paso al diseño de la arquitectura de la solución propuesta.

Durante esta etapa, los principales hitos fueron la generación de prototipos de productos, y tener claro los componentes que se deben desarrollar para el alcance definido del proyecto.

Esta etapa estaba planificada para ser realizada en dos meses, y si bien se cumplió con la estimación de la duración, la etapa comenzó tardíamente, por lo que terminó después de lo planificado. De todas maneras, se lograron cumplir los principales hitos de la etapa, como ser la prototipación de algunas tecnologías candidatas a ser utilizadas, la definición de la arquitectura con todos sus componentes, y la elección de ciertas herramientas para el desarrollo.

Durante esta etapa no se registraron impedimentos importantes, aunque por contrapartida, algunas definiciones no habían podido ser terminadas.

6.3.3 Implementación y testing

Finalmente, la etapa de construcción del producto fue la etapa donde el equipo se encontró con más impedimentos para poder cumplir con lo planificado originalmente.

En primer lugar, la etapa comenzó más tarde de lo previsto, y ni bien comenzó, aún existían ciertos detalles que no habían terminado de ser aclarados, y al principio de esta etapa el equipo se encontró con cierta indisponibilidad para aclarar estos puntos. Si bien se podía continuar paralelamente con otras tareas, esto retrasó algunas decisiones.

También al ser una etapa muy larga, fue la que más desviación registró respecto a la estimación original, donde el equipo claramente subestimó esta fase.

Por otro lado, llegando a la mitad de la etapa, comenzaron a llegar las licencias y vacaciones, y algunos problemas de índole personal de algunos de los integrantes hicieron que el tiempo se estirara aún más.

Finalmente se decidió cerrar la etapa, cerrando con el testing del prototipo de implementación al que se había llegado.

Un punto donde el equipo se vio muy beneficiado para contrarrestar el atraso con respecto a la planificación, fue a la hora de seleccionar las tecnologías a utilizar, donde en varios casos, se optó por tecnologías donde el equipo ya presenta cierta experiencia.

6.3.4 Documentación

Transversalmente al resto de las etapas en las que se dividió el proyecto, se fue armando incrementalmente la documentación del mismo, generando la mayoría de los contenidos de este documento a lo largo de todo el proyecto, principalmente los técnicos.

La herramienta para edición colaborativa que se utilizó en este caso fue Google Documents, ya que permite una edición completamente en línea y de manera simultánea por todos los integrantes del equipo. De esta manera, a medida que se fueron cerrando algunos puntos se iba presentando de manera separada al tutor en formato PDF.

Capítulo 7: Trabajo futuro

En este capítulo se presenta un conjunto de funcionalidades, componentes y herramientas que consideramos que agregarían un importante valor al producto como para que el mismo pueda ser desplegado en un ambiente de producción a futuro.

7.1 Mejoras

Ambientes: despliegue de tres ambientes (*Testing*, *Staging* y *Production*).

Integración y entrega continua: instalación y configuración de un conjunto de herramientas para aplicar integración y entrega continua (GitLab, Jenkins, Sonarqube, y Nexus o JFrog, etc). Tiene como objetivo que los desarrolladores de la plataforma tengan un ambiente para desarrollar los conectores de las plataformas que se vayan agregando, así como también las modificaciones de las funcionalidades de la plataforma en sí.

API Management: incorporación de un API Management como componente propio de la plataforma para administrar las APIs de la misma y las de los recursos de las organizaciones (Escalabilidad, Tiempo de actividad, Portal de desarrolladores, Control de acceso y Seguridad, Analítica, y Monetización).

Gestión de identidad y acceso: integración de un componente de gestión de identidad y acceso (Single Sign-on), con características como: servidor de autenticación, *identity broker*, federación de usuarios, APIs REST e interfaz gráfica de usuarios de administración. Tiene como objetivos fundamentales agregar autenticación a las aplicaciones y asegurar los servicios, sin la necesidad de ocuparse de almacenar usuarios ni de autenticar usuarios (todo está disponible en componentes externos que se integran al mismo). Este componente deberá integrarse al API Management.

Nuevos conectores: implementación de otros conectores para diferentes productos de ESB, API Management u otro tipo de plataforma, para ampliar la oferta y combinación de productos en base a las diferentes capacidades que los mismos prevean.

Nuevas funcionalidades desde la web: añadir funcionalidades de la plataforma a través de la Web.

- Login y logout.
- Alta, baja y modificación de usuarios.
- Alta, baja y modificación de roles.
- Asignar roles a usuario.
- Alta, baja y modificación de organizaciones.
- Asignar usuarios a organizaciones.
- Solicitar, eliminar y modificar colaboración de un recurso por parte de una organización.

- Aceptar, cancelar y modificar solicitud de colaboración de un recurso por parte de una organización.
- Eliminar y modificar plataforma.
- Eliminar y modificar recurso.
- Alta, baja y modificación de especificación de plataforma.

Monitoreo: utilización de herramientas para centralizar tanto el monitoreo de las plataformas desplegadas en OpenShift como el monitoreo del mismo desde una herramienta desplegada externamente. Una herramienta de monitoreo para las plataformas desplegadas en OpenShift podría ser Hawkular que tiene integración nativa con el mismo. Por el otro lado, Zabbix y Nagios son herramientas que podrían ser utilizadas para realizar el monitoreo de OpenShift.

Gestión centralizada de logs: dado que se trata de un sistema distribuido, sería deseable utilizar un conjunto de herramientas que permitan realizar trazas y auditorías sobre todos los componentes desplegados en el sistema. Elasticsearch, Fluentd y Kibana (EFK) son tres herramientas que se combinan muy bien y permiten recopilar, indexar, buscar y visualizar logs de manera centralizada. Es importante destacar que OpenShift cuenta con integración nativa para estas herramientas.

Consistencia de datos entre Core y PaaS: a futuro sería conveniente agregar una funcionalidad que verifique la consistencia entre los datos existentes en el Core y los generados en el PaaS y sus plataformas, y que corrija sus diferencias. De esta manera se lograría asegurar que lo gestionado a través del sistema sea correcto, y que en el PaaS no existan plataformas y recursos que el Core no está gestionando.

7.2 Pruebas a realizar

Para probar las funcionalidades de nuestro sistema distribuido de manera efectiva y eficiente, analizamos los problemas asociados con las pruebas del mismo y presentamos algunas soluciones para asegurar la liberación de nuevas funcionalidades con alta confiabilidad.

Estrategia de Testing:

1. Pruebas unitarias para verificar las funcionalidades más pequeñas y específicas de la aplicación y el cumplimiento del comportamiento deseado. Estas pruebas se ejecutan de manera automática, y permiten observar una distinción importante basada en si la unidad a probar está o no aislada de sus colaboradores. Dado que el proyecto está íntegramente implementado utilizando Maven, éstas ejecutan cada vez que se realiza un *build* de la solución. De esta manera, teniendo un buen porcentaje de cubrimiento en las pruebas unitarias, y contando con alguna herramienta que permita la automatización del empaquetado y despliegue, como por ejemplo, Jenkins, las ejecución de las pruebas unitarias estaría a cargo de dicha herramienta, previo a la construcción de los artefactos a desplegar.
2. Pruebas de integración con el objetivo de verificar las rutas de comunicación y las interacciones entre los componentes para detectar defectos de las interfaces. Realizar pruebas automatizadas de los módulos que interactúan con componentes

externos, con el objetivo de verificar que el módulo pueda comunicarse correctamente. Con pruebas de este tipo, se apunta a cubrir los flujos básicos de éxito y error a través del módulo de integración.

3. Pruebas de componente: limitan el alcance del sistema a una porción más reducida, a la cual se le realizan pruebas manipulando el sistema a través de interfaces de código interno y utilizando stubs o mocks para aislar el código bajo prueba de otros componentes. En otras palabras, se realizan pruebas sobre un componente simulando los eventos y mensajes que le envían los demás componentes.
4. Pruebas de contrato, es decir, pruebas en el límite de un servicio externo que verifiquen que cumple con el contrato esperado por un servicio consumidor.
5. End-to-end con la meta de verificar que un sistema cumpla con los requisitos externos y logre sus objetivos, probando todo el sistema de extremo a extremo. Se considera que es necesario que la aplicación se encuentre desplegada en algún ambiente, y que las pruebas ejecutadas desde la UI Web pueden ser automatizadas a través de herramientas como Selenium, como se dijo anteriormente, y lanzadas a través de la herramienta Jenkins.

Etapas de prueba:

- Core - Conectores
- Core - OpenShift
- Core - Conectores - OpenShift
- Web - Core
- Web - Core - Conectores
- Web - Core - OpenShift
- Web - Core - Conectores - OpenShift

Cada una de estas estrategias tiene como objetivo proporcionar cobertura a cada capa y entre capas del servicio.

Capítulo 8: Conclusiones

Como conclusión general del proyecto se puede destacar que a partir del modelo de referencia de [1], se realizó el diseño del gestor de plataformas de middleware destinado a la colaboración inter-organizacional, que extiende dicho modelo, y luego se implementó el alcance definido dentro del marco de este proyecto utilizando los productos de software previamente analizados.

Las conclusiones alcanzadas para cada uno de los puntos específicos planteados como objetivos de este proyecto son las siguientes:

Los productos de middleware analizados y presentados en el capítulo 2, y en los anexos que allí se referencian, son productos ESB y API Managers.

En cuanto a los ESB incluidos dentro del análisis, si bien existen varios productos open source que cumplían la mayor parte de los requerimientos, se concluyó que JBoss Fuse era el que mejor se ajustaba al desarrollo del proyecto debido a la experiencia que tenía el equipo en el uso de la herramienta, y que es una herramienta de larga trayectoria en el mercado, teniendo ya cierto tiempo de madurez en su uso, por tratarse de un producto de RedHat también. La única limitante que se encontró con esta herramienta era que no contaba con una API capaz de realizar la gestión de los recursos que las organizaciones debían crear, por lo que se desarrolló una.

Por otra parte, en cuanto a los productos API Managers, se encontraron pocos productos open source que se ajustaban a las necesidades del proyecto, siendo WSO2 API Manager el seleccionado para su uso ya que contaba con una API capaz de realizar todas las tareas que se ofrecen en la UI para la gestión desde la aplicación que finalmente se desarrolló. A medida que fue implementándose la solución final, se encontró que a pesar que la herramienta cuenta con el nivel de madurez necesario, algunas funcionalidades, como el trabajo con el Throttling, o la incorporación de Authentication a nivel de API no tenía la capacidad necesaria para ser utilizada de la manera que se había diseñado. Por otro lado, la API provista no cumple con todos los requisitos de una API RESTful, lo que hizo más complejo el trabajo de integración con el sistema desarrollado. Otro detalle encontrado es que es una herramienta que es bastante monolítica en su diseño e implementación, lo que dificulta el trabajo de la herramienta sobre un Openshift. Estos puntos encontrados hacen que, en caso de encontrar otras herramientas open source capaces de realizar el trabajo deseado, sea conveniente a futuro considerar el cambio de producto API Manager.

Al momento del análisis de los PaaS disponibles en el mercado, se encontraron solamente algunos ejemplos de software open source que pudieran ser incluidos dentro de la solución que se buscaba, encontrando que Openshift de Red Hat era el producto que mejor se adaptaba a los requerimiento del proyecto y viabilizaba la solución diseñada, ya que permitía una instalación on-premise por parte del equipo sin necesidad de utilizar espacios en la nube con cuentas de prueba o pagas. Se concluyó que es una herramienta que ofrece grandes funcionalidades, que es capaz de automatizar de por si solo varias de las

funcionalidades que se necesitaban desde el proyecto, y que además, ofrece APIs bien documentadas y orientadas a un uso RESTful con todas las funcionalidades que brinda el producto. Como contrapartida, si bien es un producto que ya tiene un tiempo en el mercado, aún está en fase de maduración, lo que provocó que durante el desarrollo del proyecto, se encontraran varios impedimentos en cuanto a la operativa del producto. De todas maneras, el crecimiento del desarrollo de este producto es tal que a medida que se iba avanzando con la implementación de la solución, las nuevas versiones de Openshift iban corrigiendo varios de estos problemas encontrados.

A partir del análisis realizado del modelo del paper [1] se generó un modelo extendido y se diseñó una plataforma de gestión de componentes de middleware que tiene como objetivo la colaboración inter-organizacional. Este diseño se basa en un producto PaaS que permite la gestión de las diferentes plataformas de middleware, que a su vez, permiten la generación de recursos y compartirlos.

Por otro lado, se implementó un prototipo de la plataforma de integración inter-organizacional con todos los requerimientos planteados dentro del alcance de este proyecto, siguiendo el diseño realizado a partir del análisis del modelo de [1]. Este prototipo implementado permite probar la viabilidad de la construcción de un producto de software a partir del modelo del paper. Dentro de esta implementación, se incluyó la aplicación web que permite a las organizaciones la auto-gestión de sus servicios generados dentro de la plataforma, así como también el uso de las herramientas investigadas y mencionadas en el punto 1, como el PaaS Opensfhith, el ESB JBoss Fuse y el API Manager de WSO2.

Como conclusión general del testing realizado sobre este tipo de plataforma, se observó que la metodología de testing difiere levemente de un sistema tradicional, ya que las funcionalidades implementadas debieron ser testeadas en más de un componente. Esto significa que para testear la implementación de un caso de uso, primero se debió probar el funcionamiento del PaaS y de las plataformas WSO2 y Fuse, para luego pasar a testear el conector con cada uno de ellos, seguido de las pruebas realizadas al Core y finalmente probar todo de la UI.

Referencias

- [1] L. González and R. Ruggia, "A reference architecture for integration platforms supporting cross-organizational collaboration," 2015, pp. 1–4.
- [2] L. González and R. Ruggia, "Towards a Compliance-Aware Inter-organizational Service Integration Platform," in *On the Move to Meaningful Internet Systems: OTM 2014 Workshops*, 2014, pp. 8–17.
- [3] "Cloud computing fundamentals," 23-Jan-2012. [Online]. Available: <http://www.ibm.com/developerworks/ssa/cloud/library/cl-cloudintro/index.html>. [Accessed: 06-Jul-2017].
- [4] P. Mell, T. Grance, and others, "The NIST definition of cloud computing," 2011.
- [5] "AWS | Cloud Computing - Servicios de informática en la nube," *Amazon Web Services, Inc.* [Online]. Available: <http://aws.amazon.com/es/>. [Accessed: 07-Jul-2017].
- [6] "Microsoft Azure: plataforma y servicios de informática en la nube." [Online]. Available: <https://azure.microsoft.com/es-es/>. [Accessed: 07-Jul-2017].
- [7] "Compute Engine - IaaS," *Google Cloud Platform*. [Online]. Available: <https://cloud.google.com/compute/?hl=es>. [Accessed: 07-Jul-2017].
- [8] "IBM - Cloud Computing for Builders & Innovators," 19-Feb-2016. [Online]. Available: <https://www.ibm.com/cloud-computing/>.
- [9] "ConvergedSystem - Converged Infrastructure Solutions & Systems." [Online]. Available: <https://www.hpe.com/us/en/integrated-systems/converged.html>. [Accessed: 07-Jul-2017].
- [10] "Cloud Services - Deploy web apps & APIs | Microsoft Azure." [Online]. Available: <https://azure.microsoft.com/en-us/services/cloud-services/>. [Accessed: 08-Jul-2017].
- [11] "Cloud Solutions - CA Technologies." [Online]. Available: <https://www.ca.com/us/why-ca/hybrid-cloud-solutions.html>. [Accessed: 08-Jul-2017].
- [12] "Cloud Infrastructure, Storage, Security &, More - IBM Bluemix." [Online]. Available: <https://www.ibm.com/cloud-computing/bluemix/>. [Accessed: 08-Jul-2017].
- [13] "CloudHub | Hybrid Integration | iPaaS," *MuleSoft*, 07-Sep-2016. [Online]. Available: <https://www.mulesoft.com/platform/saas/cloudhub-ipaas-cloud-based-integration>. [Accessed: 08-Jul-2017].
- [14] "OpenShift: Container Application Platform by Red Hat, Built on Docker and Kubernetes." [Online]. Available: <https://www.openshift.com/>. [Accessed: 07-Jul-2017].
- [15] Gartner, "Magic Quadrant for Enterprise Integration Platform as a Service." 24-Mar-2016.
- [16] "Integrate Anything with Dell Boomi's industry leading Cloud iPaaS platform," *Dell Boomi*, 08-Sep-2016. .
- [17] "iPaaS: Cloud Integration Platform as a Service | Informatica US." [Online]. Available: <https://www.informatica.com/products/integration-platform-as-a-service.html#fbid=zcz1GQHZEu5>. [Accessed: 08-Jul-2017].
- [18] "IBM - WebSphere Cast Iron Cloud Integration," 01-Jan-2016. [Online]. Available: <https://www.ibm.com/software/products/es/cast-iron-cloud-integration>. [Accessed: 08-Jul-2017].
- [19] "API (Application Program Interface) Definition." [Online]. Available: <https://techterms.com/definition/api>. [Accessed: 07-Jul-2017].
- [20] "Choosing the Right API Management Solution for the Enterprise User - CA Technologies." [Online]. Available: <https://www.ca.com/us/collateral/white-papers/choosing-the-right-api-management-solution-for-the-enterprise-user.html>. [Accessed: 13-Jul-2017].

- [21]“REST - Semantic Web Standards.” [Online]. Available: <https://www.w3.org/2001/sw/wiki/REST>. [Accessed: 07-Jul-2017].
- [22]“SOAP Specifications.” [Online]. Available: <https://www.w3.org/TR/soap/>. [Accessed: 07-Jul-2017].
- [23]“SOAP vs. REST API Challenges | SoapUI Testing Dojo.” [Online]. Available: <https://www.soapui.org/testing-dojo/world-of-api-testing/soap-vs--rest-challenges.html>. [Accessed: 07-Jul-2017].
- [24]steved0x, “Azure API Management overview and key concepts.” [Online]. Available: <https://docs.microsoft.com/en-us/azure/api-management/api-management-key-concepts>. [Accessed: 07-Jul-2017].
- [25]CITO Research and Microsoft, “Cloud-based API Management: Harnessing the Power of APIs.” 2015.
- [26]M. Keen *et al.*, “Patterns: Implementing an SOA using an enterprise service bus,” *IBM Redb.*, vol. 336, 2004.
- [27]T. Erl, *SOA: principles of service design*. Upper Saddle River, NJ: Prentice Hall, 2008.
- [28]“Web Services Glossary.” [Online]. Available: <https://www.w3.org/TR/ws-gloss/>. [Accessed: 07-Jul-2017].
- [29]“Service-Oriented Architecture.” [Online]. Available: <https://www.opengroup.org/soa/source-book/soa/>. [Accessed: 28-Jul-2017].
- [30]“Microservices, SOA, and APIs: Friends or enemies?,” 21-Jan-2016. [Online]. Available: http://www.ibm.com/developerworks/websphere/library/techarticles/1601_clark-trs/1601_clark.html. [Accessed: 07-Jul-2017].
- [31]C. T. Jensen, “IBM - APIs for Dummies.” 2015.
- [32]“Amazon API Gateway Documentation,” *Amazon Web Services, Inc.* [Online]. Available: <https://aws.amazon.com/documentation/apigateway/>. [Accessed: 07-Jul-2017].
- [33]“API Management Platform | API Manager,” *MuleSoft*, 02-Sep-2016. [Online]. Available: <https://www.mulesoft.com/platform/api/manager>. [Accessed: 07-Jul-2017].
- [34]“Understand monetization,” *Apigee Product Documentation*, 21-Oct-2013. [Online]. Available: <http://docs.apigee.com/monetization/content/basics-monetization>. [Accessed: 07-Jul-2017].
- [35]“Introducing the API Manager - API Manager 1.10.0 - WSO2 Documentation.” [Online]. Available: <https://docs.wso2.com/display/AM1100/Introducing+the+API+Manager>. [Accessed: 07-Jul-2017].
- [36]“Web Service Definition Language (WSDL).” [Online]. Available: <https://www.w3.org/TR/wsdl>. [Accessed: 07-Jul-2017].
- [37]“Web Application Description Language.” [Online]. Available: <https://www.w3.org/Submission/wadl/>. [Accessed: 07-Jul-2017].
- [38]“Swagger – The World’s Most Popular Framework for APIs.” [Online]. Available: <https://swagger.io/>. [Accessed: 07-Jul-2017].
- [39]“Welcome,” *RAML*. [Online]. Available: <https://raml.org/>. [Accessed: 07-Jul-2017].
- [40]“What is TLS/SSL?: Logon and Authentication.” [Online]. Available: [https://technet.microsoft.com/en-us/library/cc784450\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc784450(v=ws.10).aspx). [Accessed: 07-Jul-2017].
- [41]“OASIS Web Services Security (WSS) TC | OASIS.” [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss. [Accessed: 07-Jul-2017].
- [42]“OpenID Foundation website.” .
- [43]steved0x, “How to add operations to an API in Azure API Management.” [Online]. Available: <https://docs.microsoft.com/en-us/azure/api-management/api-management-howto-add-operations>. [Accessed: 07-Jul-2017].
- [44]D. A. Chappell, *Enterprise Service Bus: Theory in Practice*. O’Reilly Media, Inc., 2004.
- [45]M. P. Papazoglou, *Web services: principles and technology*. Harlow: Pearson/Prentice Hall, 2008.
- [46]N. M. Josuttis, *SOA in practice*, 1st ed. Beijing ; Sebastopol: O’Reilly, 2007.

- [47] "Ken Vollmer's Blog." [Online]. Available: http://blogs.forrester.com/ken_vollmer/10-05-11-comparing_esb_and_integration_suite_functionality. [Accessed: 08-Jul-2017].
- [48] "Understanding Linux Containers." [Online]. Available: <https://www.redhat.com/en/containers>. [Accessed: 08-Jul-2017].
- [49] "What is a Container," *Docker*, 29-Jan-2017. [Online]. Available: <https://www.docker.com/what-container>. [Accessed: 07-Jul-2017].
- [50] "Chapter 1. Introduction to Linux Containers - Red Hat Customer Portal." [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/overview_of_containers_in_red_hat_systems/introduction_to_linux_containers. [Accessed: 07-Jul-2017].
- [51] "KVM." [Online]. Available: http://www.linux-kvm.org/page/Main_Page. [Accessed: 07-Jul-2017].
- [52] "Virtualization – Oracle VM VirtualBox." [Online]. Available: <https://www.virtualbox.org/wiki/Virtualization>. [Accessed: 07-Jul-2017].
- [53] "Virtualization Getting Started Guide." [Online]. Available: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html-single/Virtualization_Getting_Started_Guide/index.html. [Accessed: 08-Jul-2017].
- [54] "Docker overview," *Docker Documentation*, 07-Jul-2017. [Online]. Available: <https://docs.docker.com/engine/docker-overview/>. [Accessed: 07-Jul-2017].
- [55] "What is Docker," *Docker*, 14-May-2015. [Online]. Available: <https://www.docker.com/what-docker>. [Accessed: 07-Jul-2017].
- [56] "Kubernetes," *Kubernetes*. [Online]. Available: <https://kubernetes.io/>. [Accessed: 07-Jul-2017].
- [57] "The world's open source leader." [Online]. Available: <https://www.redhat.com/en>. [Accessed: 07-Jul-2017].
- [58] "Project Atomic." [Online]. Available: <http://www.projectatomic.io/>. [Accessed: 07-Jul-2017].
- [59] "Node-RED : Documentation." [Online]. Available: <https://nodered.org/docs/>. [Accessed: 07-Jul-2017].
- [60] "JUnit - About." [Online]. Available: <http://junit.org/junit4/>. [Accessed: 07-Jul-2017].
- [61] "Postman," *Postman*. [Online]. Available: <https://www.getpostman.com/>. [Accessed: 07-Jul-2017].
- [62] "SoapUI | Functional Testing for SOAP and REST APIs." [Online]. Available: <https://www.soapui.org/>. [Accessed: 07-Jul-2017].
- [63] "Selenium Documentation — Selenium Documentation." [Online]. Available: <http://www.seleniumhq.org/docs/>. [Accessed: 28-Jul-2017].
- [64] "OAuth Community Site." [Online]. Available: <https://oauth.net/>. [Accessed: 06-Aug-2017].
- [65] MuleSoft, "Mule ESB." [Online]. Available: <https://www.mulesoft.com/sites/default/files/mule-esb.pdf>. [Accessed: 06-Aug-2017].
- [66] "Welcome to Apache ServiceMix!" [Online]. Available: <http://servicemix.apache.org/>. [Accessed: 07-Jul-2017].
- [67] "JBoss Fuse." [Online]. Available: <https://www.redhat.com/es/technologies/jboss-middleware/fuse>. [Accessed: 07-Jul-2017].
- [68] "WSO2 Enterprise Service Bus - Only 100% Open Source ESB." [Online]. Available: <http://wso2.com/products/enterprise-service-bus/>. [Accessed: 07-Jul-2017].
- [69] P. B. Kruchten, "The 4+1 View Model of architecture," *IEEE Softw.*, vol. 12, no. 6, pp. 42–50, Nov. 1995.

Glosario

Middleware: es el soft-sistema que permite las interacciones a nivel de aplicación entre programas en un ambiente distribuido.

Soft-sistema: software posicionado entre una aplicación y un sistema de menor nivel.

Ambiente computacional distribuido: ambiente computacional cuyos programas o BDs están ubicados en dos o más computadoras.

On-premise: software instalado y ejecutado en servidores locales de la organización que utiliza el software.

SLA: acuerdo oficial entre un proveedor de servicios y el cliente, sobre aspectos del servicio como responsabilidades, calidad y disponibilidad.

Monetización: adopción de cualquier variedad de medidas que una compañía toma para incrementar la rentabilidad de su propiedad intelectual, software.

Endpoint: componente utilizado como punto de entrada de un servicio.

Gobernanza: es el establecimiento de cadenas de responsabilidad para preparar a las personas, medición de la efectividad de los calibradores, políticas para guiar la organización con el fin de que cumpla sus objetivos, mecanismos de control para garantizar el cumplimiento, y comunicación para mantener todas las partes informadas.

API: interfaces de servicios ofrecidos a través de la Web, de manera pública y que sirven de puerta de entrada para el uso de los diferentes servicios que una organización posee.

SOA: es esencialmente una colección de servicios que se comunican entre sí, donde la comunicación puede implicar pasaje sencillo de datos o coordinación de alguna actividad entre dos o más servicios.

REST: estilo de arquitectura de software para sistemas hipermedia distribuidos.

Hipermedia: conjunto de métodos o procedimientos para escribir, diseñar o componer contenidos que integren texto, imágenes, videos, audios, mapas y otros. Además, el resultado obtenido debe tener la posibilidad de interactuar con los usuarios.

SOAP: es una especificación de un protocolo para el intercambio de información estructurada en la implementación de web services.

WSDL: es el formato estándar para describir un web service.

WADL: es un formato basado en XML para describir servicios web basados en HTTP, comúnmente REST.

Swagger: es una representación simple pero completa de una API RESTful.

RAML: es un lenguaje de modelado de API REST que facilita la gestión de todo el ciclo de vida de la API desde el diseño hasta que se comparte.

SSL/TLS: protocolos criptográficos que proporcionan autenticación, privacidad e integridad de los datos transferidos entre aplicaciones que se comunican a través de la red.

OpenID: es un estándar de identificación digital descentralizado que permite utilizar una cuenta existente para iniciar sesión en varios sitios web (que soporten el protocolo), sin necesidad de crear nuevas contraseñas.

XML (eXtensible Markup Language): es un lenguaje de marcado para los documentos que contienen información estructurada.

JSON (JavaScript Object Notation): es un formato de intercambio de datos rápido.

XSLT (eXtensible Stylesheet Language Transformations): es una aplicación del lenguaje XML para especificar reglas que permiten transformar un documento XML en otro.

XPath: está basado en una representación arbórea de los documentos XML. Provee la posibilidad de navegar por el árbol y seleccionar nodos del mismo según algún criterio definido. Además, permite realizar operaciones prefabricadas sencillas sobre los datos del documento, tales como operaciones matemáticas o lógicas.

JAAS: paquete de Java que ofrece el servicio de autenticación y control de acceso de usuarios en las aplicaciones.

JB1: especificación de una Aplicación de Integración Empresarial (EAI) en Java, siguiendo los principios de la Arquitectura Orientada a Servicios (SOA).

Proxy: es un servidor intermediario que reenvía las solicitudes de contenido desde varios clientes a diferentes servidores a través de Internet.

Reverse Proxy: es un tipo de servidor Proxy que normalmente se sitúa detrás del firewall en una red en una red privada y reenvía las solicitudes del cliente al servidor backend apropiado.

Colaborar: es el mecanismo mediante el cual las organizaciones comparten recursos.

Recurso: es un ítem dentro de la plataforma de integración que una organización utiliza para satisfacer sus necesidades de negocio. Un recurso es expuesto a través de la plataforma de manera que otra organización pueda consumirlo.

Capacidad: es un mecanismo que tiene como objetivo facilitar los problemas de integración.

Utilidad: es una funcionalidad brindada por la plataforma para los usuarios finales.

Plataforma: es un componente de software de cierto tipo capaz de gestionar un recurso.

Especificación de plataforma: es una especificación que indica el tipo de una plataforma, y define todo lo necesario para desplegar una nueva plataforma de este tipo, además de indicar cuáles son aquellas capacidades que pueden ser ejecutadas en esta plataforma a la hora de generar un recurso.

Hipervisor: ejecuta en modo kernel y da soporte a las máquinas virtuales. Realiza las operaciones que requieren privilegios en nombre del sistema operativo invitado.

Anexo I: Análisis de productos de API Management

1 Microsoft Azure API Management

Microsoft [19] hace referencia a las diferentes funcionalidades no solamente que ofrece su producto, sino que describe algunas de las funcionalidades que todo API Manager debería incluir:

- La funcionalidad principal de un producto API Manager debe ser la de ofrecer las operaciones definidas en la API de una manera altamente configurable, permitiendo controlar el mapeo de URLs, los parámetros de invocación (tanto en el query string como en el path de las URIs), el manejo del contenido de las solicitudes y las respuestas (Azure API Management lo realiza a través del concepto de Políticas), y el uso de cachés en las respuestas de las operaciones. De esta manera, se ofrece una API con cierto formato en sus parámetros, enmascarando otra API, con diferente paramétrica, realizando mapeos entre los parámetros de entrada y los de salida, que generalmente son leídos a partir de las especificaciones de estas APIs (WSDLs, WADLs, RAMLs, Swagger, y otros estándares que se utilizan para documentar). En el caso de Azure API Management, los mapeos de entrada se configuran a la hora de crear las APIs, donde se especifica la URI base de la API (incluyendo el nombre y la versión), y definiendo luego operaciones (en base a los métodos HTTP) que mapean los parámetros de entrada con los de salida para cada API [34].
- Seguidamente en orden de importancia, aparecen las funcionalidades de:
 - Control de acceso (por usuario y por IP)
 - Monitoreo
 - Trazas de eventos
 - Estadísticas de uso.

Microsoft hace referencia a un White Paper que incluye un análisis sobre los requerimientos comunes de los API Managers: [27]

- Desacoplamiento: principalmente, de las APIs respecto a los servicios que las implementan, permitiendo por ejemplo, la reutilización de APIs legadas sin tener que reescribirlas.
- Facilidad de acceso: se refiere a la facilidad y automatización del acceso que se le puede ofrecer a los diferentes desarrolladores que utilizarán las APIs. Esto hace referencia a la facilidad de acceso por parte de los usuarios que desarrollan aplicaciones que desean consumir una API publicada a un sistema de gestión donde poder solicitar suscripciones a APIs, acceso a APIs públicas en los diferentes

ambientes donde es publicada, generalmente accesibles a través de claves, que también pueden ser administradas a través de la misma interfaz de gestión.

- Seguridad: incluye el aseguramiento de los back-ends de las APIs, no exponiéndolos directamente, y la protección de las APIs contra el mal uso y el sobreuso.
- Visibilidad: ofrecer métricas de qué APIs se están usando, cuánto se utilizan y quiénes la utilizan.

Con respecto a las SLAs, Microsoft las implementa a través de las Políticas mencionadas anteriormente, y son definidas como aquellas acciones que pueden configurarse para modificar el comportamiento de una API. Se definen como un conjunto ordenado de sentencias que son ejecutadas secuencialmente tanto en la solicitud como en la respuesta de una API.

2 WSO2 API Manager

Por otro lado, WSO2 API Manager [31] también hace una breve enumeración de algunas características generales que debería tener un API Management.

Ellas son:

- Administración de APIs: WSO2 engloba con este concepto todo el manejo que deben poder realizar los proveedores de APIs.
 - Creación de APIs: se menciona que la creación de APIs debe ser sencilla para el usuario final, y debe ofrecerse a través de interfaces amigables. Permite generar APIs manejadas a partir de APIs existentes, copiando su interfaz, o también permite crear nuevas APIs, permitiendo transformaciones. Para ejemplificar el segundo caso, WSO2 permite crear una API REST a partir de una API SOAP ya existente, permitiendo definir todas las transformaciones necesarias de los datos para convertir de un protocolo a otro.
 - Manejo del ciclo de vida: el manejo del ciclo de vida de una API implica controlar las etapas de prueba anteriores a su publicación, y luego a la deshabilitación temporal y posible deprecación de la misma, para luego darlas de baja. Para esto, maneja una serie de estados para cada API: CREADA, PROTOTIPADA, PUBLICADA, DEPRECADA, RETIRADA y BLOQUEADA.
 - Versionado: se realiza generando copias casi exactas de las APIs, permitiendo configurar nuevamente una API, asociada a la versión anterior.
 - Monetización y Gobernanza: el concepto de gobernanza abarca algunas característica ya mencionadas, como el manejo de ciclo de vida, o el manejo de la monetización, por ejemplo, a través de SLAs que controlan también el acceso y lo registran de modo de poder luego sacar estadísticas, o permiten también al proveedor de la API generar ingresos por consumo. Como resultado de esta gobernanza también se incluye lo referente a las estadísticas y monitoreo.
- Seguridad
 - Control de acceso: se basa principalmente en la autenticación y autorización que tiene de las APIs que se pueden publicar. Permite diferentes modalidades de

seguridad, como ser autenticación por tokens autogestionados por el propio desarrollador en la interfaz web provista para ello, a través de protocolos como OAuth [64], a través de certificados digitales, o también permite a través de su administrador, generar otros métodos de seguridad customizables. Por otro lado, al proveedor de la API le permite seleccionar qué tipo de acceso es necesario para cada API: si es una API pública (accesible por todo el mundo), privada (accesible por usuarios registrados) o restringida (accesible únicamente para aquellos usuarios registrados que el proveedor disponga).

- Protección de las APIs: como parte de los SLAs que implementa WSO2 para sus APIs y que permite configurar al proveedor, se hace énfasis en algunos casos en que estas prácticas permiten a la plataforma proteger las APIs de algunos tipos de ataque, por ejemplo, limitando la cantidad de llamadas, ya sea en general o por usuario.
- Escalabilidad: un middleware de este tipo debe ser escalable, para no convertirse en un cuello de botella de tráfico para todas las APIs que se gestionan en él, permitiendo escalar horizontalmente trabajando en cluster.

Otra funcionalidad que se menciona agregada a esta lista básica es la publicación de APIs para el manejo del API Manager. WSO2 incorpora tres APIs RESTful que permiten utilizar sus funcionalidades a través de las mismas y poder controlar desde otras plataformas externas el propio API Manager.

Las APIs publicadas son:

- una API para el proveedor de APIs
- una API para el desarrollador que consumirá las APIs publicadas y
- una API para el manejo de tokens.

Anexo II: Arquitectura básica API Manager

1 Introducción

En este documento se presenta la arquitectura básica de dos de los productos de API Manager existentes en el mercado como son Azure API Management de Microsoft y API Manager de WSO2, describiendo sus principales componentes y funcionalidades.

2 Arquitectura básica

En la Figura 1 se describe la arquitectura del producto de Microsoft, Azure API Management de la siguiente manera:

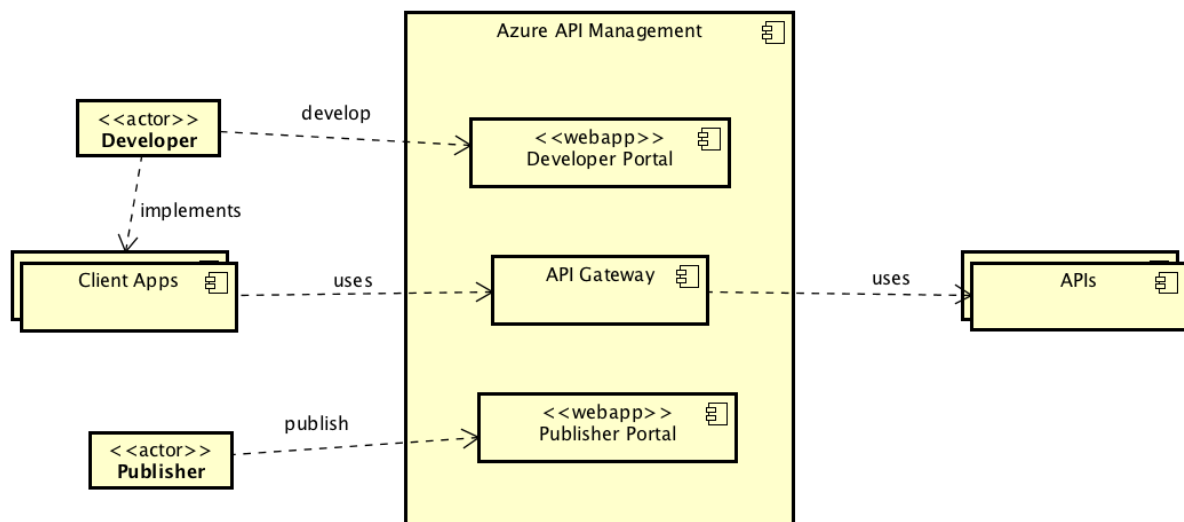


Figura 1 - Arquitectura de Azure API Management.

Un desarrollador de APIs utiliza el Developer Portal de Azure API Management para definir sus APIs. Luego un publicador de la organización publica sus APIs a través del Publisher Portal.

Por otro lado, otro desarrollador que desea utilizar la API publicada, utiliza el Developer Portal para suscribirse a la API (y eventualmente registrar su aplicación en el caso de las APIs que no son públicas), y luego invoca desde su aplicación (en el diagrama, Client Apps) la API a través del API Gateway.

Por otro lado, WSO2 presenta la arquitectura básica de su producto API Manager de la siguiente manera:

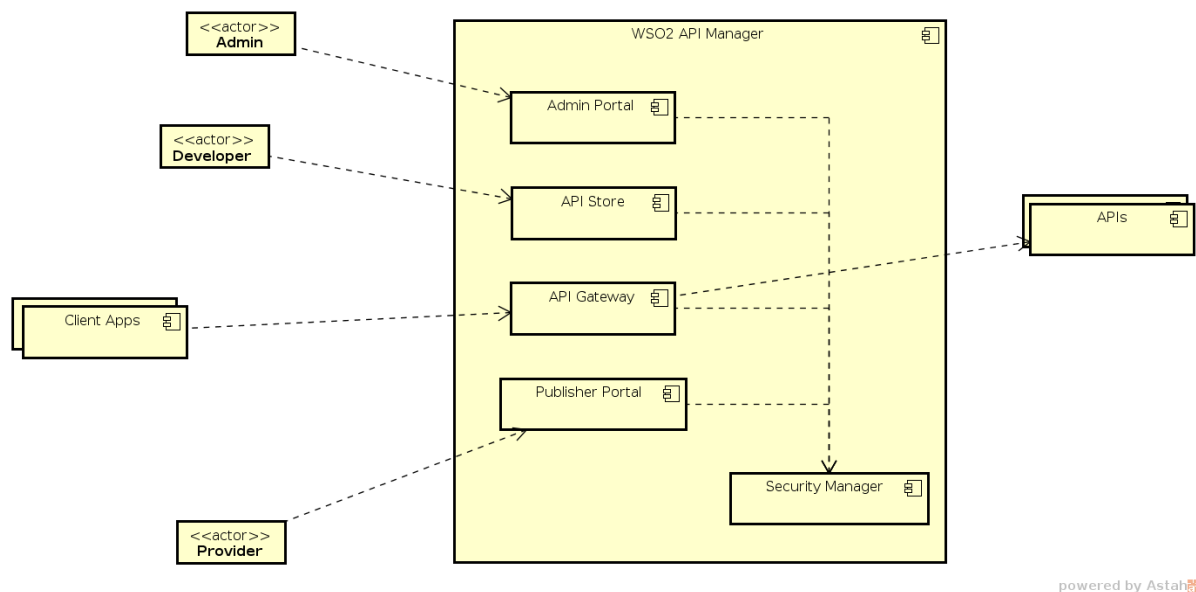


Figura 2 - Arquitectura de WSO2 API Manager.

En esencia, los componentes principales de este diagrama simplificado de la arquitectura de WSO2 API Manager presenta los mismos componentes que el presentado para Azure API Management. En el diagrama se separa la parte de seguridad en un componente aparte debido a que tanto el API Store (que actúa de manera similar al Developer Portal de Azure) el Publisher Portal y el Security Manager exponen también APIs RESTful (no representadas explícitamente en el diagrama anterior) para que otras plataformas puedan hacer uso del API Manager a través de las mismas, APIs que en el caso del producto de Microsoft no están presentes.

Anexo III: Análisis de productos ESB

1 Introducción

Relevamiento de los principales productos ESB existentes en el mercado.

2 Mule ESB

Mule ESB es uno de los primeros ESB open source. Es una plataforma de integración basado en Java que permite a los desarrolladores conectar sus aplicaciones de forma rápida y sencilla. Permite la integración entre sistemas existentes independientemente de las tecnologías utilizadas en las aplicaciones incluyendo múltiples protocolos de comunicación o transporte tales como HTTP, FTP, POP3/SMTP, sistemas de archivos, entre otros. Mule fue diseñado para integrarse fácilmente con servidores de aplicación o ejecutar como un servidor standalone. [46], [65]

Principales funcionalidades

A continuación se enumeran las características principales brindadas por Mule ESB [46].

- Soporta una gran cantidad de conectividad(alrededor de 60 tecnologías).
- Múltiples protocolos de transporte como por ejemplo JMS, HTTP, FTP, entre otros.
- Ruteo de mensajes basado en contenido a través de Xpath y XPath.
- Permite comunicación síncrona y asíncrona.
- Amplitud de conectividad (más de 60 tecnologías).
- Soporta transformación de datos XML, utilizando transformaciones XSLT.
- Incluye IDE gráfico para el desarrollo de aplicaciones.
- Opciones de seguridad, brindando funcionalidades de autenticación y autorización.
- Es el ESB Open Source más utilizado.

Arquitectura básica

En la Figura 1 se presenta la arquitectura de Mule ESB obtenida de la documentación oficial.

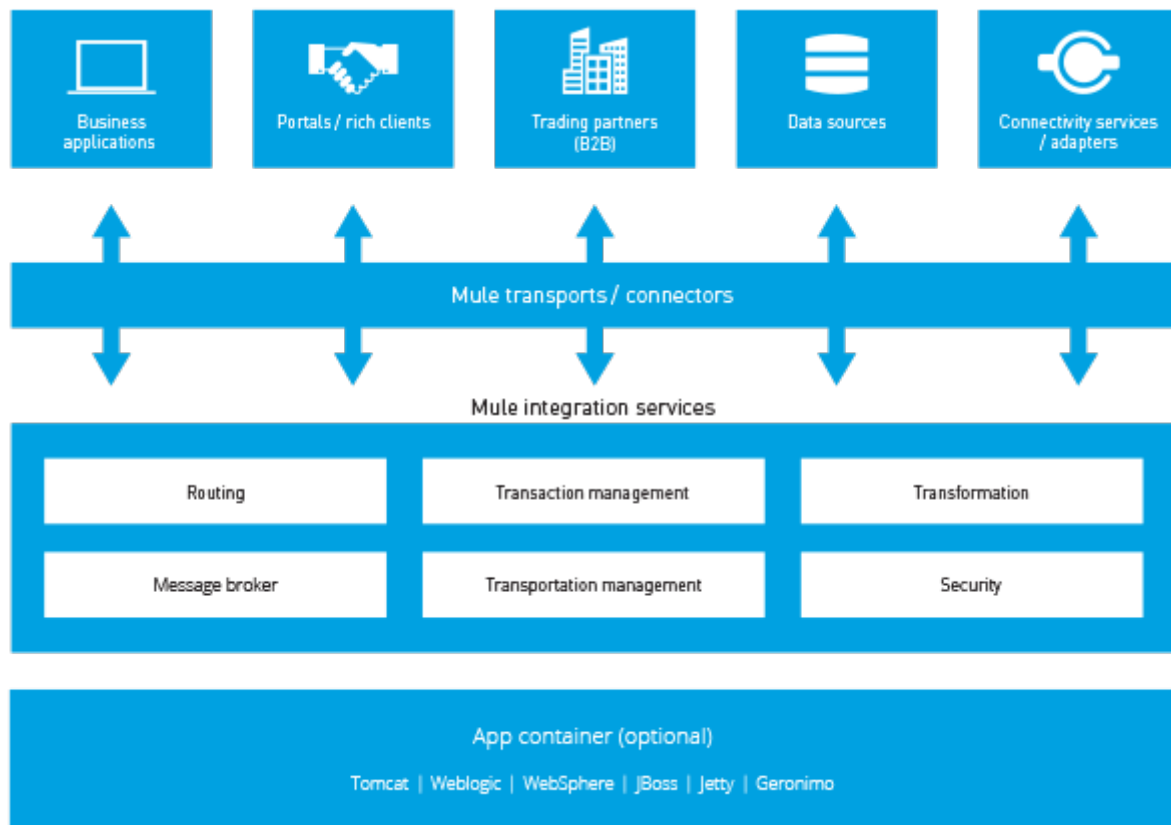


Figura 1. - Arquitectura de Mule ESB

3 Apache ServiceMix

Es un contenedor de integración flexible y de código abierto que unifica las características y funcionalidades de Apache ActiveMQ, Camel, CXF, ODE y Karaf en una plataforma de ejecución de gran alcance. Cuenta con el respaldo de The Apache Software Foundation lo cual es considerado como una buena referencia. Utiliza el estándar JBI el cual provee componentes como JMS, BPEL, Web services y Camel. [64]

Principales funcionalidades

A continuación se enumeran las características principales brindadas por ServiceMix.

- Soporta múltiples protocolos de transporte como por ejemplo JMS, HTTP, FTP, Sistema de ficheros, entre otros
- Soporta mensajería confiable con ActiveMQ
- Provee mensajería, ruteo y patrones de integración empresarial (EIP) con Apache Camel
- Soporta web services SOAP y RESTful a través de Apache CXF
- Servidor basado en OSGi con Apache Karaf
- Permite transformación de documentos XML a través de XSLT y XSLT Component.
- Dispone de ruteo basado en contenido a partir de expresiones XPath sobre mensajes XML normalizados

- Posee motor de orquestación de servicios BPEL
- Soporta la integración con servidores de aplicaciones como Geronimo, Jboss, Jetty, Tomcat, WebLogic, WebSphere
- Provee seguridad a través de JAAS, WS-Security
- Es compatible con Business Process Management(BPM) a través de Activiti
- Gestión de transacciones XA mediante Java Transaction API (JTA) a través de Apache Aries.
- Dispone de una consola web para tareas de administración

Arquitectura básica

ServiceMix es un contenedor JBI por lo tanto provee toda la infraestructura y herramientas requeridas por la especificación JBI. En la Figura 2 se muestra la arquitectura de Apache ServiceMix.

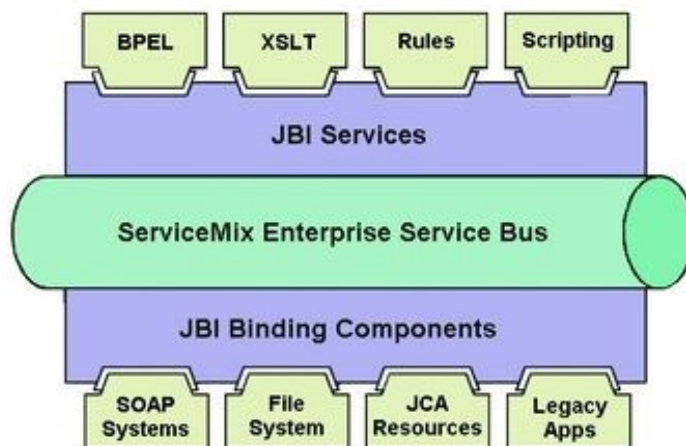


Figura 2 - Arquitectura de Apache ServiceMix

4 JBoss Fuse

Es un ESB Open source ofrecido por Red Hat. Es una solución basada en el proyecto Apache ServiceMix. Es un ESB liviano y flexible que permite la mediación, transformación y ruteo de información entre múltiples aplicaciones. En el año 2012 Red Hat Inc anunció la adquisición de FuseSource desarrollado por Progress Software Corporation. Debido a esto Fuse ESB fue rebautizado como Jboss Fuse. Es una plataforma que soporta JBI and OSGii. Está integrado en la plataforma Jboss Enterprise SOA Platform. Está basado en Apache Karaf un poderoso y liviano contenedor OSGi para el despliegue y manejo de bundles que permiten facilitar la gestión del ciclo de vida del software (tales como instalación, parada, rediseño, arranque o desinstalación de paquetes). [65]

Principales funcionalidades

A continuación se enumeran las características principales brindadas por Jboss Fuse.

- Soporta el estándar JBI

- Soporta múltiples protocolos de transporte, como por ejemplo JMS, TCP/IP, Email, entre otros
- Posee motor de orquestación de servicios BPEL
- Transformación de mensajes XML, a través de XSLT y Smooks. Se dispone de una gran variedad de tipos de datos soportados
- Posee diferentes tipos de ruteo de mensajes, como por ejemplo, ruteo basado en contenido.
- Se integra de forma nativa con jBPM y con JBoss AS
- Es posible cambiar fácilmente el comportamiento de las acciones predefinidas, como transformaciones y ruteos
- Incluye IDE gráfico para el desarrollo.

Arquitectura básica

En la Figura 3 se muestra la arquitectura básica de JBoss Fuse.

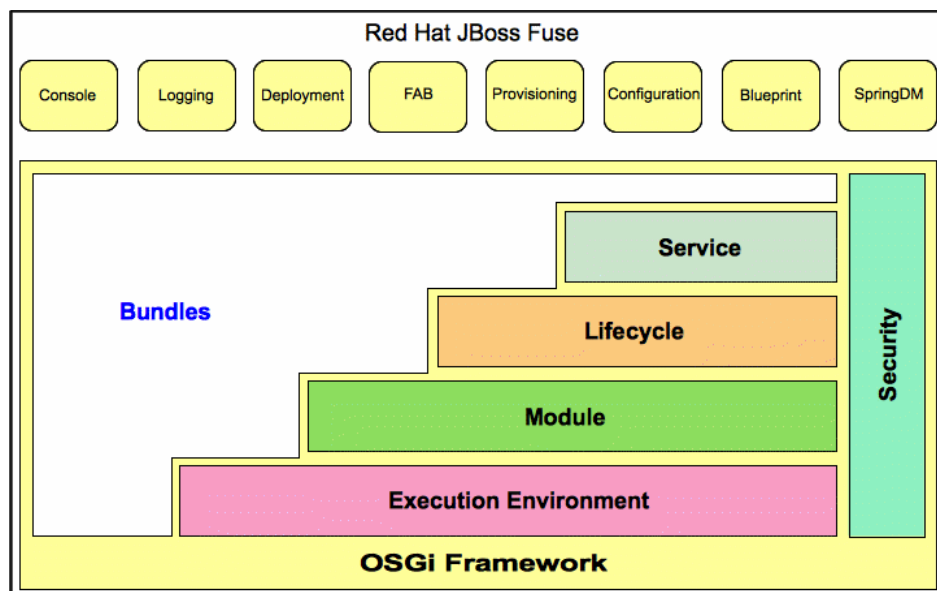


Figura 3 - Arquitectura de Jboss Fuse

5 WSO2 ESB

Es una plataforma rápida, liviana y es 100% open source. Está disponible bajo la licencia de Apache Software License. Está basado en el bus de servicios empresariales Apache Synapse de la Apache Software Foundation, añade una mejor gestión y desarrollo/soporte de configuración y capacidades SOA. Permite la utilización de una gran variedad de enterprise integration patterns(EIPs) incluyendo filtrado, transformación y ruteo. [67]

Principales funcionalidades

A continuación se enumeran las características principales [67], [68]:

- Soporta de XPath, XSLT, XQuery

- Soporte de estándares de Web Services como WS-Addressing, WS-Security, WS-Working, WS-Policy
- Provee interoperabilidad basado en los proyectos Apache Synapse y Apache Axis2
- Puede ser extendido utilizando extensiones de Java, POJO o Spring así como JavaScript, Ruby, Groovy o otro lenguaje de scripting
- Soporta una gran variedad de protocolos para la integración como JMS, FTP, SMTP, POP3, IMAP, AMQP, Financial Information eXchange (FIX)
- Está diseñado para facilitar la extensión de las capacidades que ofrece. Tiene incorporadas las siguientes capacidades:
 - Ruteo basado en contenido
 - Servicio de virtualización
 - Balanceo de carga
 - Transformación de mensaje
 - Monitoreo
 - Enterprise integration patterns
- Ofrece una consola de administración
- Cuenta con un IDE Gráfico WSO2 Developer Studio

Arquitectura básica

En la Figura 4 se muestra la arquitectura básica de WSO2 ESB.

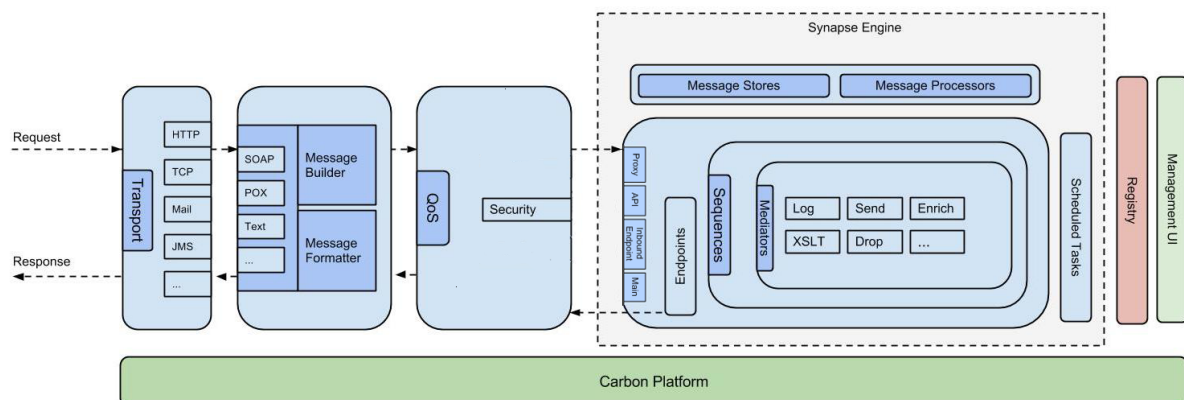


Figura 4 - Arquitectura de WSO2 ESB

Anexo IV: Análisis comparativo de plataformas

1 Introducción

Los recursos disponibles en el sistema serán desplegados en plataformas provistas por el sistema.

Los tipos de plataformas disponibles son:

- ESB
- API Manager
- Cola de mensaje

2 ESB

Con la necesidad de aumentar la flexibilidad de los entornos de IT dentro de las organizaciones, las aplicaciones deben ser integradas y fácilmente adaptables a los nuevos requerimientos. Un ESB es una excelente solución para la integración de aplicaciones ya que evita tener que escribir el propio código de integración cada vez que se necesite integrar una aplicación. Actualmente las tecnologías open source ofrecen una gran competencia, existen una gran variedad de ESB open source que se encuentran al mismo nivel que los productos comerciales.

En base a la información recabada en el marco conceptual acerca de los productos ESB existentes en el mercado realizamos una comparación para establecer qué características soportaba cada producto.

La Tabla 1 contiene algunos productos de ESB. Este relevamiento se enfocó en un conjunto de características presentadas a continuación.

	Jboss Fuse	MuleSoft	ServiceMix	WSO2 ESB
Conectividad/Adaptadores	múltiples conectores	gran cantidad de conectores/adaptadores	múltiples conectores.	múltiples conectores.
Transformación de mensajes	con Smooks y XSLT, Drools, Xpath	XSLT, XQuery, Smooks, AnyPoint DataMapper.	XQuery, XSLT,	XPath, XSLT, XQuery, Smooks
Sistemas de mensajería	JbossMQ, JbossMessaging, MQSeries, OracleAQ, TibcoEMS y	soporte de JMS, open source: ActiveMQ, OpenJMS, JbossMQ.	soporte de JMS vía Apache ActiveMQ	soporte de JMS.

	ActiveMQ	Comerciales: WebLogicMQ, WebSphereMQ, etc.		
Monitorización	consola de administración.	Consola de administración	JMX	consola de administración, JMX
Seguridad	Jboss Security,	plugins de seguridad: Spring security, JAAS, SSL/TLS, SAML, PGP, OAuth.	JASS, WS-Security.	WS-Security with Apache Rampart, LDAP, Kerberos, OpenID, SAML, XACML
Servidor de aplicaciones	Not Standalone, Jboss AS.	Standalone, Tomcat, Jboss, WebLogic, Jetty, Geronimo, WebSphere, Resin	Tomcat, Jboss, Geronimo, WebSphere, WebLogic	Standalone, Jboss, Tomcat, etc
IDE	JBoss Studio, plugins para Eclipse.	Anypoint Studio, plugins para Eclipse.	Plugins para Eclipse.	WSO2 Developer Studio, plugins para Eclipse.
Configuración	basada en XML	basada en XML	basada en XML	basada en XML

Tabla 1 - Comparativa entre productos ESB

Como conclusión en base a la información recabada podemos decir que MuleSoft es el ESB open source más utilizado. Es una plataforma de integración que es utilizada por los desarrolladores para conectar sus aplicaciones debido a su facilidad de configuración. Tiene una gran variedad de conectores, opciones de transformaciones, ruteo que pueden ser reutilizados en otras soluciones.

ServiceMix es un solución flexible de ESB open source que unifica las características y funcionalidades de productos de Apache en una poderosa plataforma que se puede considerar como solución a los problemas de integración. Es muy sencillo integrar nuevos flujos en tiempo de ejecución permitiendo agregar nuevos componentes.

Jboss Fuse combina varias tecnologías como Apache Camel, Apache CXF, Apache ActiveMQ, Apache Karaf y Fabric8 en una misma plataforma de integración, además está basado en Apache ServiceMix por lo que podemos decir que le aplica un valor extra a lo que ofrece ServiceMix.

A diferencia del resto de los productos analizados WSO2 es 100% open source, no dispone de una versión comercial. Es un ESB liviano que ofrece una gran flexibilidad y agilidad para satisfacer las necesidades de integración.

3 API Management

Al igual que con los ESB se realizó una comparativa de las diferentes productos de API Manager existentes en el mercado.

A partir del marco conceptual se establecieron un conjunto de funcionalidades que caracterizan a las API Management, siempre tomando en cuenta que los productos de API Management se han desarrollado como una opción de integración de sistemas con fuerte énfasis en la usabilidad y simplicidad.

En la Tabla 2 se presentan algunos productos existentes en el mercado con sus correspondientes capacidades.

	Azure	WSO2	CA	MuleSoft	IBM	Amazon
Documentación	Swagger, WADL, WSDL	Swagger, WSDL	Swagger, WADL, RAML, WSDL	RAML, WSDL	Swagger, WADL, RAML, WSDL	Swagger, RAML, WSDL
Analítica y estadística	Sí	Sí	Sí	Sí	Sí	Sí
Despliegue	Cloud	Cloud, On-Premise, Híbrido, Cluster	Cloud, On-Premise, Híbrido, Cluster	Cloud, On-Premise, Híbrido, Cluster	Cloud, On-Premise, Híbrido, Cluster	Cloud
Compromiso con el desarrollador	Sí	Sí	Sí	Sí	Sí	Sí
Entorno Sandbox	Sí. Lo maneja con tenants	Sí. Lo maneja dentro del ciclo de vida con permisos	Sí.	Sí. Lo maneja con un ambiente diferente dentro de la misma plataforma	Sí. Lo maneja con catálogos	Sí, poco claro como lo realiza.
Tráfico y cache	Sí	Sí	Sí	Sí	Sí	Sí
Seguridad	OAuth, certificados, encriptación	Pocas opciones	Muchas opciones	OAuth, credenciales, firmas, encriptación.	OAuth, LDAP, RBAC(role-based access control), security gateway	varias herramientas para control de acceso, AWS Identity, Access Management y Amazon

						Cognito
Monetización	Sí	Sí	Sí	Si	Si	Sí
Disponibilidad	Sí	Si	Sí	Sí	Sí	Sí

Tabla 2 - Comparativa entre productos de API Management

En base a los resultados de las comparaciones realizadas y presentadas en la Tabla 2 de los diferentes productos de API Management vistos, se puede ver como todos los productos cumplen con los requerimientos planteados, aunque en alguno de ellos, hay diferentes matices. Por ejemplo, para el despliegue, Amazon y Azure tienen cierta desventaja ya que solamente funcionan en sus propias clouds, en cambio el resto de los productos no solamente pueden desplegarse en clouds, sino que existen opciones para ejecutarlos on-premise, o en ambientes híbridos. Otro ítem que tiene ciertas diferencias es la parte de Seguridad, donde vemos que la mayoría manejan estándares de autenticación como OAuth o certificados, en cambio WSO2 no parecen ser claras las instrucciones para utilizar alguno de estos métodos, sí manejando por defecto el estándar de autenticación básica de HTTP a través de claves generadas en la plataforma. También existen diferencias en cuanto a las formas soportadas de documentación de APIs, aunque todas manejan estándares abiertos como Swagger y WSDL.

Como conclusión de este análisis podemos decir que la herramienta más completa de las que se presenta es el producto de CA Technologies junto con la herramienta de IBM, y en cambio la de Azure y Amazon pierden bastante terreno principalmente por lo dicho sobre el despliegue de sus productos.

Tanto los ESB como las API Management son tecnologías utilizadas para integrar sistemas pero que tiene enfoques diferentes. Para lograr comprender la diferencia entre ambas tecnologías realizamos una tabla comparativa que pretende esclarecer cuándo es conveniente usar cada tecnología .

En la Tabla 3 se presenta una tabla con las principales diferencias entre API Management y ESB. [5][8]

	API Management	ESB
Objetivo	Publicación de APIs seguras, gobernadas y monetizables	Implementación de una SOA.
Conectividad	Se basa en una conectividad liviana.	Incluyen una gran cantidad de conectores/adaptadores
Definición	Centrado en la configuración/scripting(no programadas)	Se basan en el flujo o codificación en algún lenguaje.
Gobernanza	Aplicación de políticas que permiten manejar y determinar el comportamiento de las APIs. (Policy Driven)	Se basa en la gestión de procesos de negocio

Exposición	Generalmente se exponen en un portal para desarrolladores	No incluyen esta funcionalidad
Políticas de uso de APIs	Incluyen planes de condiciones de uso de una determinada API.	Algunos traen políticas de uso incorporadas pero no incluyen especificación de planes de uso.
Tecnologías	Tiende a usar tecnologías como REST, HTTP basic auth, OAuth, JSON.	Tiende a usar tecnologías como SOAP, WS-Security, and SAML.
Público objetivo	Abierto a Internet	Inter o entre empresas

Tabla 3 - Comparativa entre las tecnologías de API Management y ESB.

En base al estudio realizado logramos concluir que los ESB se enfocan en implementar una SOA es decir llevar un determinado sistema a una SOA. Se basan en el concepto de servicios y están orientados a prestar servicios inter o entre empresas.

Por otro lado tenemos a las API Management que son una tecnología más nueva cuyo objetivo es la de publicar APIs de forma segura, gobernadas y monetizables. A diferencia de los ESB estas generalmente proporcionan APIs que se encuentran abiertas a internet para que diferentes usuarios puedan consumirlas.

Ambas tecnologías tiene como objetivo la integración de sistemas pero cada una de ellas tiene un enfoque diferente.

Anexo V: Casos de uso críticos

1 Introducción

A continuación se detallan los casos de uso que se consideran críticos, llamando críticos a aquellos que se ven involucrado directamente en alguno de los escenarios planteados.

2 Detalle de los casos de uso

Alta recurso

Actor principal: Publicador

Descripción: El caso de uso comienza cuando un un publicador logueado en el sistema quiere dar de alta un nuevo recurso en el sistema.

La creación de recursos se considera como el caso de uso más crítico ya que define gran parte de los componentes críticos de la arquitectura de la solución.

Pre-condiciones:

- El usuario debe estar logueado y debe tener los permisos adecuados para ejecutar la funcionalidad.

Flujo Principal:

1. Incluye CU login, Listar capacidades, Configurar capacidades, Listar especificación de plataformas, Alta plataforma.
2. El usuario indica que desea dar de alta un recurso
3. Ingresa nombre, uri con la que se identificará dentro de la organización, versión del recurso que va a crear e indica si el acceso será público o privado.
4. El usuario indica la entrada es decir qué tipo de conector será el recurso.

Los tipos de recurso son:

- REST
 - SOAP
 - Cola de mensaje
5. Luego se listaran las capacidades disponibles y el usuario deberá ir seleccionando las que desee. El orden en que se coloquen es importante ya que determinarán el orden del flujo de ejecución de las mismas.
 6. Luego de seleccionar las capacidades el usuario debe configurar las mismas ingresando ciertos parámetros dependiendo del tipo de capacidad seleccionada.
 7. El sistema deja habilitadas aquellas especificaciones de plataformas que podrá seleccionar el usuario en base a las capacidades seleccionadas.

8. El usuario debe seleccionar una especificación de plataforma para dar de alta la plataforma donde se desplegará el recurso.
9. El usuario da de alta la plataforma.
10. El sistema verifica que la configuración de la plataforma sea válida para las capacidades seleccionadas
11. El usuario confirma el alta.
12. Fin del CU.

Flujos Alternativos:

7A. Plataforma existente.

1. En caso de que la organización a la cual pertenece el usuario logueado disponga de plataformas ya dadas de alta el usuario podrá seleccionar una de estas.
2. Se vuelve al paso 9.

5A. Especificación de plataforma no válida.

1. El sistema devuelve un error indicando= que la especificación no es válida.
2. Se vuelve al paso 4

Post-condiciones:

- Se da de alta en el sistema un recurso con los atributos establecidos.
- En caso de dar de alta una plataforma se crea una instancia del tipo de plataforma seleccionada con la configuración establecida.

Modificar recurso

Actor principal: Publicador

Descripción: El caso de uso comienza cuando un un publicador logueado en el sistema quiere modificar un recurso que existe en el sistema.

Pre-condiciones:

- Debe existir en el sistema el recurso.

Flujo Principal:

1. Incluye CU login
2. El usuario modifica los datos del recurso que desea. Puede modificar la configuración de una capacidad que anteriormente había especificado o agregar una nueva nueva configuración de una capacidad soportada por la plataforma.
3. El sistema valida los datos.
4. El usuario confirma la operación.
5. Fin de CU.

Flujos Alternativos: No tiene

Post-condiciones:

- Si el usuario confirma la operación los datos del recurso son reemplazados por los nuevos.

Alta plataforma

Actor principal: Publicador

Descripción: El caso de uso comienza cuando un un publicador logueado en el sistema quiere dar de alta una plataforma.

Pre-condiciones:

- El usuario debe estar logueado.
- El usuario debe haber seleccionado un tipo de template para la plataforma.

Flujo Principal:

1. Incluye CU login
2. El usuario indica que desea dar de alta una plataforma en base a una especificación de plataforma seleccionada.
3. El usuario ingresa nombre para identificar la plataforma.
4. El sistema a partir de una configuración pre establecida para el tipo de plataforma crea una instancia de la misma.
5. El usuario confirma la operación.
6. Fin de CU.

Flujos Alternativos:

No tiene

Post-condiciones:

- Se crea una instancia de la plataforma con la configuración establecida.

Configurar capacidad

Actor principal: Publicador

Descripción: El caso de uso comienza cuando un un publicador logueado en el sistema quiere configurar una capacidad.

Pre-condiciones:

- El usuario debe estar logueado.
- Se debe seleccionar algún tipo de capacidad para configurar.

Flujo Principal:

1. Incluye CU login
2. El usuario ingresa ciertos atributos dependiendo el tipo de capacidad que va a configurar. Las capacidades se dividen en dos grandes conjuntos, las relacionadas a los tipos de endpoints y las relacionadas con las transformaciones, validaciones y ruteos.

Las capacidades que soporta el sistema son las siguientes:

- Endpoint REST
- Endpoint SOAP
- Cola de mensajes
- Mapeo de parámetros
- Descomponer mensaje
- Componer mensaje
- Validar
- Throttle
- Agregar información al mensaje
- Filtrar información del mensaje

Si el usuario selecciona la capacidad de Endpoint SOAP debe establecer los siguientes valores:

- WSDL (URL donde se encuentra publicado o el WSDL como texto)
- URL de los diferentes ambientes donde se encuentra publicado el servicio.

3. El sistema valida la configuración.
4. El usuario confirma la operación.
5. Fin de CU.

Flujos Alternativos:

2A. Capacidad Endpoint REST:

1. Se debe ingresar los siguientes datos:
 - Swagger (URL donde se encuentra publicado el JSON o el JSON completo como texto)
 - URL en los diferentes ambientes donde se encuentra corriendo el servicio

2B. Capacidad Cola de mensajes:

1. Se debe ingresar los siguientes datos:
 - Nombre que tendrá la cola de mensajes

2C. Capacidad Mapeo de parámetros:

1. Se debe ingresar los siguientes datos:
 - Para cada parámetro del servicio anterior (REST o SOAP) que se quiera modificar se debe proveer un nuevo nombre

2C. Capacidad Descomponer mensaje:

1. Se debe ingresar los siguientes datos:
 - No se requieren datos extras

2C. Capacidad Componer mensaje:

1. Se debe ingresar los siguientes datos:

- No se requieren datos extras

2C. Capacidad Validar:

1. Se debe ingresar los siguientes datos:

- Se deberá indicar la expresión regular que se validará
- Se deberá indicar el lenguaje en el que está la expresión regular (deberá ser un lenguaje soportado por el sistema)

2C. Capacidad Throttle:

1. Se debe ingresar los siguientes datos:

- Cantidad máxima de mensajes por unidad de tiempo
- Unidad de tiempo en la que se desea medir

2C. Capacidad Agregar información al mensaje:

1. Se debe ingresar los siguientes datos:

- Se podrá proveer una lista de parámetros que se agregarán al mensaje con su valor.
- También se podrá proveer una expresión que se agregará al mensaje.

2C. Capacidad Filtrar información del mensaje:

1. Se debe ingresar los siguientes datos:

- Se podrá proveer una lista de parámetros que se eliminarán del mensaje
- También se podrá proveer de una expresión que podrá ser sustituida por un valor que provea el usuario

Post-condiciones:

- La capacidad queda configurada.

Listar Capacidades

Actor principal: Publicador

Descripción: El caso de uso comienza cuando un un publicador logueado en el sistema desea visualizar las capacidades ingresadas en el sistema.

Pre-condiciones:

- El usuario debe estar logueado.

Flujo Principal:

1. Incluye CU Login.
2. El usuario indica que desea visualizar todas las capacidades soportadas por el sistema.

3. El sistema lista las capacidades soportadas.
4. Fin de CU.

Flujos Alternativos: No tiene

Post-condiciones: No tiene

Listar especificaciones de plataformas

Actor principal: Publicador

Descripción: El caso de uso comienza cuando un un publicador logueado en el sistema quiere seleccionar un templete de plataforma.

Pre-condiciones:

- El usuario debe estar logueado.

Flujo Principal:

1. Incluye el caso de uso Login
2. El usuario indica que desea visualizar las especificaciones de plataformas disponibles
3. El sistema lista los tipos de templates disponibles en el sistema.
4. Estos templates son:
 - Jboss Fuse ESB
 - ActiveMQ
 - WSO2 API Manager
5. Fin del CU.

Flujos Alternativos: No tiene

Post-condiciones: No tiene

Listar plataformas

Actor principal: Publicador

Descripción: El caso de uso comienza cuando un publicador logueado en el sistema quiere seleccionar un plataforma que fue dada de alta anteriormente por la organización.

Pre-condiciones:

- El usuario debe estar logueado y debe pertenecer a la organización a la cual pertenece la plataforma.

Flujo Principal:

1. Incluye el caso de uso Login
2. El usuario indica que desea visualizar sus plataformas disponibles.
3. El sistema lista las plataformas disponibles de la organización.
4. Fin del CU.

Flujos Alternativos: No tiene

Post-condiciones: No tiene

Activar recurso

Actor principal: Publicador

Descripción: El caso de uso comienza cuando un publicador desea activar un recurso que dio de alta anteriormente.

Pre-condiciones:

- El usuario debe estar logueado.

Flujo Principal:

1. Incluye CU Login.
2. El usuario selecciona un recurso que dio de alta para activarlo.
3. El sistema activa el recurso.
4. Fin de CU.

Flujos Alternativos: No tiene

Post-condiciones:

- El recurso queda activado para poder ser utilizado por otra organización.

Listar recursos activos

Actor principal: Publicador

Descripción: El caso de uso comienza cuando el publicador desea visualizar los recursos que se encuentran activos en el sistema.

Pre-condiciones:

- El usuario debe estar logueado.

Flujo Principal:

1. Incluye CU Login.
2. El usuario indica que desea visualizar los recursos activos de una organización.
3. El sistema lista los recurso que se encuentran activos.
4. Fin CU.

Flujos Alternativos: No tiene

Post-condiciones: No tiene

Solicitar recurso

Actor principal: Publicador

Descripción: El caso de uso comienza cuando un un publicador logueado en el sistema quiere solicitar un recurso a una organización.

Pre-condiciones:

- El usuario debe estar logueado y debe tener los permisos adecuados para ejecutar la funcionalidad.
- Debe existir en el sistema algún recurso activo.

Flujo Principal:

1. Incluye CU Login.
2. El usuario busca un recurso que se encuentra activo.
3. El sistema muestra los datos del recurso.
4. El usuario selecciona el tipo de tarifa que desea.

Las tarifas disponibles son:

- plan básico
 - plan empresarial
 - plan empresarial premium
5. El usuario solicita el recurso
 6. El sistema notifica a la organización que existe una solicitud de recurso pendiente de aprobación.
 7. Fin de CU

Flujos Alternativos: No tiene

Post-condiciones:

- El recurso queda pendiente de aprobación.

Compartir recurso

Actor principal: Publicador

Descripción: El caso de uso comienza cuando un publicador logueado en el sistema acepta la solicitud de un recurso.

Pre-condiciones:

- El usuario debe estar logueado.
- Debe existir una solicitud del recurso.

Flujo Principal:

1. Incluye CU Login.
2. El sistema lista las solicitudes de recursos pendientes de aprobación
3. El usuario selecciona la solicitud que desea aceptar
4. El usuario acepta la solicitud
5. El sistema notifica al publicador que el recurso fue compartido.
6. Fin de CU.

Flujos Alternativos: No tiene

Post-condiciones:

- El recurso ha sido compartido.
- El sistema informa al usuario que realizó la petición de que la solicitud fue aprobada

Uso de Utilidad

Actor principal: Usuario final

Descripción: El caso de uso comienza cuando un desarrollador de una organización invoca una URI del sistema para utilizar una utilidad de la misma desde una aplicación externa al sistema.

Pre-condiciones:

- La organización debe tener credenciales para el uso del sistema.
- La organización debe tener permiso de uso de la utilidad, ya sea porque es dueña del recurso asociado a la utilidad, o porque el dueño ha aprobado su solicitud de uso.
- La organización no debe haber consumido la cantidad de usos límite definido en la tarifa asociada a la utilidad, en caso de existir.

Flujo Principal:

1. Se invoca la URI de la utilidad con un mensaje y las credenciales de la organización.
2. El sistema verifica las credenciales de la organización
3. En caso de existir una tarifa asociada y el invocador no ser el dueño de la utilidad, el sistema verifica la contabilización de usos que tiene la organización
4. El sistema invoca el recurso dentro de la plataforma
5. La plataforma retorna una respuesta, en caso de ser necesaria
6. Fin de CU.

Flujos Alternativos:

2B - Las credenciales no son válidas

3. Se devuelve un código de error
4. Fin de CU

3C - La organización alcanzó el límite de usos

4. Se devuelve un código de error
5. Fin de CU

Post-condiciones:

- El sistema devuelve la respuesta a la invocación en caso de existir una
- El sistema contabiliza el uso en caso de no ser una utilidad propia al invocador

Anexo VI: Arquitectura completa de la solución

1 Introducción

En este documento se especifica la arquitectura de la solución planteada.

La arquitectura de la solución será presentada utilizando el modelo de vistas 4 + 1 de Kruchten [69].

2 Vista l3gica

Esta vista muestra una descripción lógica de la arquitectura, donde se incluye una descripción breve del diagrama de clases ya presentado en el capítulo de análisis y del funcionamiento de los casos de uso principales del sistema, presentados a través de diagramas de secuencia.

Diagrama de clases

A partir de lo presentado en la sección de análisis, se presenta el diagrama de clases completo en la figura 1. En esta sección se dará una breve descripción del mismo.

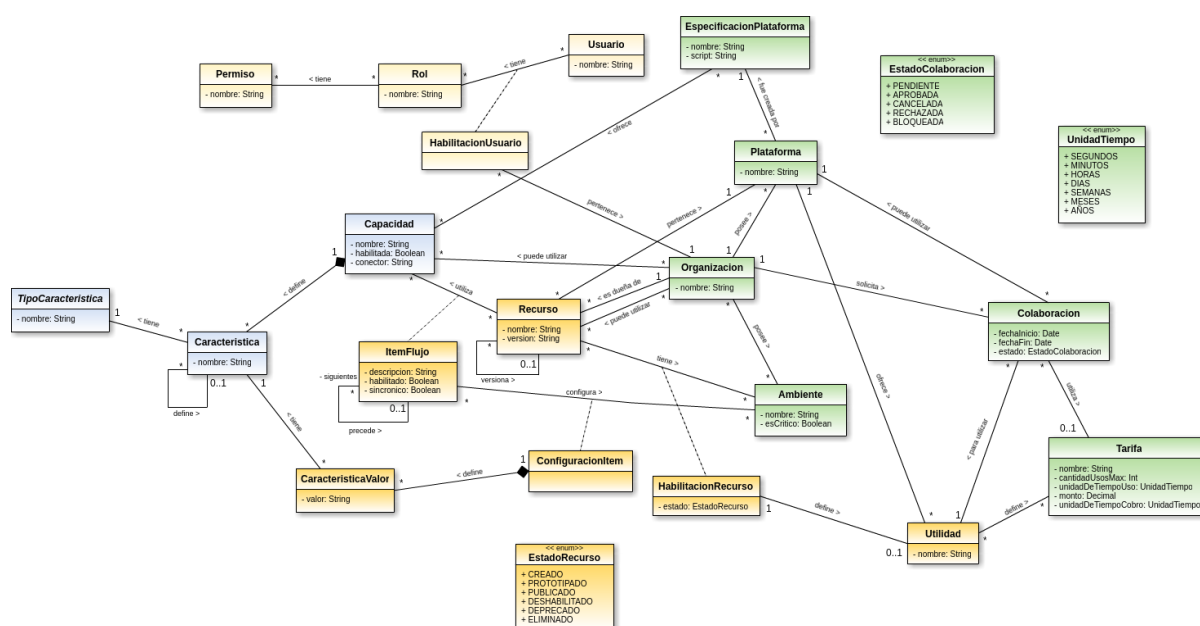


Figura 1 - Diagrama de clases del sistema.

En el diagrama de clases de la figura 1 se caracteriza por presentar organizaciones que generan plataformas. Cada plataforma tiene una especificación, que contiene la definición de un tipo de plataforma soportado por el sistema. Esta definición incluye tanto la forma en la que se despliega la plataforma, como las capacidades que ésta soporta. Cada capacidad a su vez, define una lista de características, cuyo uso se explicará más adelante.

Por otro lado, los usuarios de las organizaciones pueden generar recursos, que se definen a través de las capacidades que se le seleccionan, formando un flujo de capacidades ya que la ejecución de estas capacidades tienen un orden determinado por el usuario. Al momento de la selección de cada capacidad, el usuario deberá configurar cada una de las características para cada uno de los ambientes definidos por la organización. En el diagrama de la figura 1 esto se ve a través de las clases ItemFlujo, que oficia de instancia de una capacidad para ese recurso, y de ConfiguracionItem, que nuclea dichas configuraciones para cada ambiente a través de una colección de CaracteristicaValor, que no es más que el valor que el usuario le da a cada una de las características de la capacidad.

Otra sección de relevancia dentro del diagrama de clases es la que está compuesta por las entidades relacionadas con la colaboración entre organizaciones. Lo que se representa en este diagrama es la capacidad que se le da a una organización de solicitar una colaboración con otra organización para el uso de una utilidad, entendiendo por utilidad como un recurso publicado.

Diagramas de secuencia

A continuación se describen los principales casos de uso utilizando los diagramas de secuencia de las figuras 2 y 3.

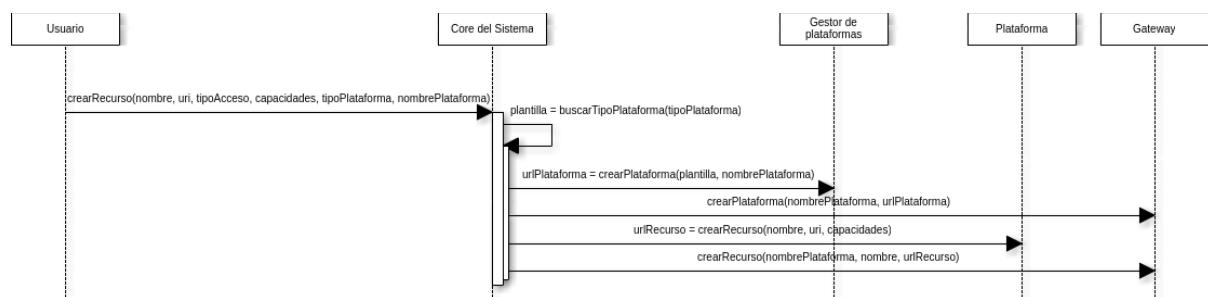


Figura 2 - Diagrama de secuencia: Alta de recurso.

Los casos de uso fueron descriptos en el Anexo IV, donde se puede encontrar el detalle de la especificación de cada uno. Los casos de uso considerados para esta vista por son el de Alta de Recurso, que contiene el del Alta de Plataforma y los de Colaboración, tanto el de Solicitar Recurso como el de Compartir Recurso, donde se aprueba la solicitud de colaboración. Fueron elegidos estos dos casos de uso por considerarse que son aquellos que más componentes afectan a nivel de arquitectura.

Como se puede ver en el diagrama de secuencia de la figura 2, el usuario al dar de alta un nuevo recurso, dispara la funcionalidad de creación de recurso en el core del sistema. Esta funcionalidad implica comunicación con el gestor de plataformas para dar de alta la plataforma, y con el gateway para permitir el acceso a la misma. Por otro lado, cuando se efectiviza la creación del recurso, se obtiene la URL de la plataforma que se generó para invocar a través del conector correspondiente, la creación del recurso. Finalmente se vuelve a invocar al gateway para que éste permita el acceso al recurso a la organización, para que este pueda ser utilizado.

El otro caso de uso considerado en esta sección es el de Solicitar Recurso, que en el diagrama de secuencia de la figura 3, se presenta en conjunto con el de Compartir Recurso.

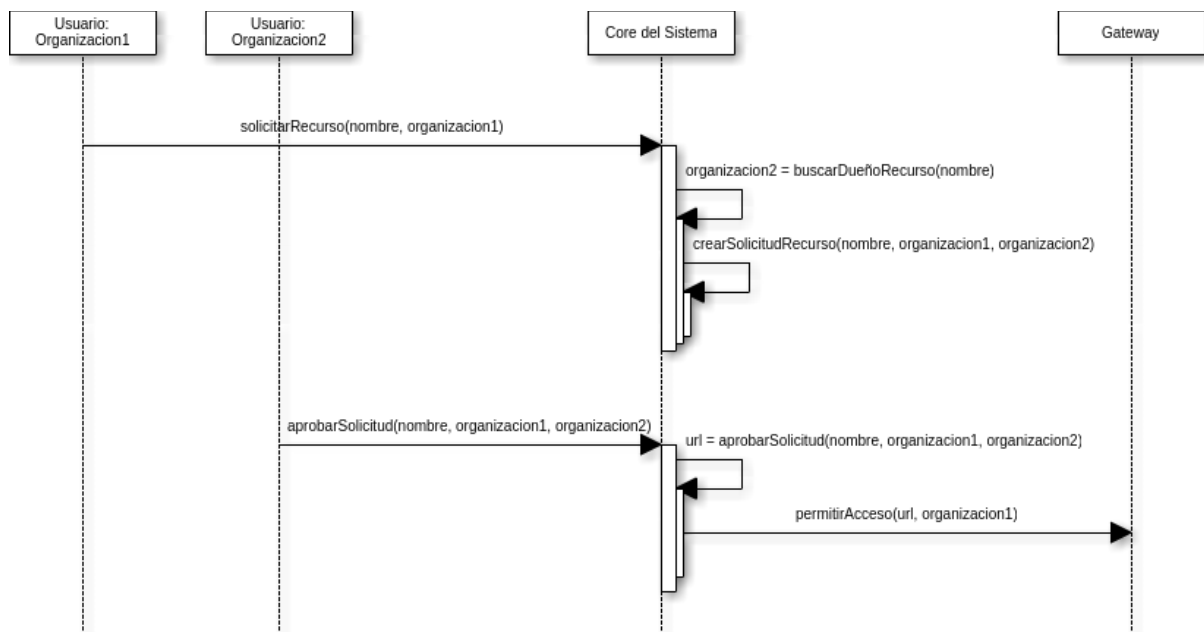


Figura 3 - Diagrama de secuencia: Colaboración.

En dicho diagrama se puede ver como el caso de uso Solicitar Recurso genera internamente en el Core del Sistema una solicitud de una organización externa a la organización dueña del recurso.

Luego en el caso de uso de Compartir Recurso, se aprueba dicha solicitud, teniendo que generar en el gateway un acceso permitido a la organización solicitante.

3 Vista de desarrollo

El objetivo principal de esta vista es mostrar los componentes principales de la arquitectura propuesta en la solución, basándose en el diagrama de componentes que representa dicha arquitectura de alto nivel.

La sección se inicia con la presentación del diagrama de componentes, en la figura 4, que representa la arquitectura que se propone como solución al problema planteado. En este diagrama no se toman en cuenta criterios físicos de componentización, sino que se muestran componentes lógicos a alto nivel.

Tal y como se puede notar en el diagrama de componentes de la figura 4, la arquitectura propuesta cuenta con dos partes bien diferenciadas:

- Una parte donde se realiza toda la gestión de la configuración necesaria para crear plataformas, recursos, utilidades, etc., a la que llamaremos Core de la plataforma.
- Otra segunda parte, que llamaremos Gateway, que es en donde se realiza toda la interacción entre las aplicaciones cliente y las utilidades publicadas en las plataformas.

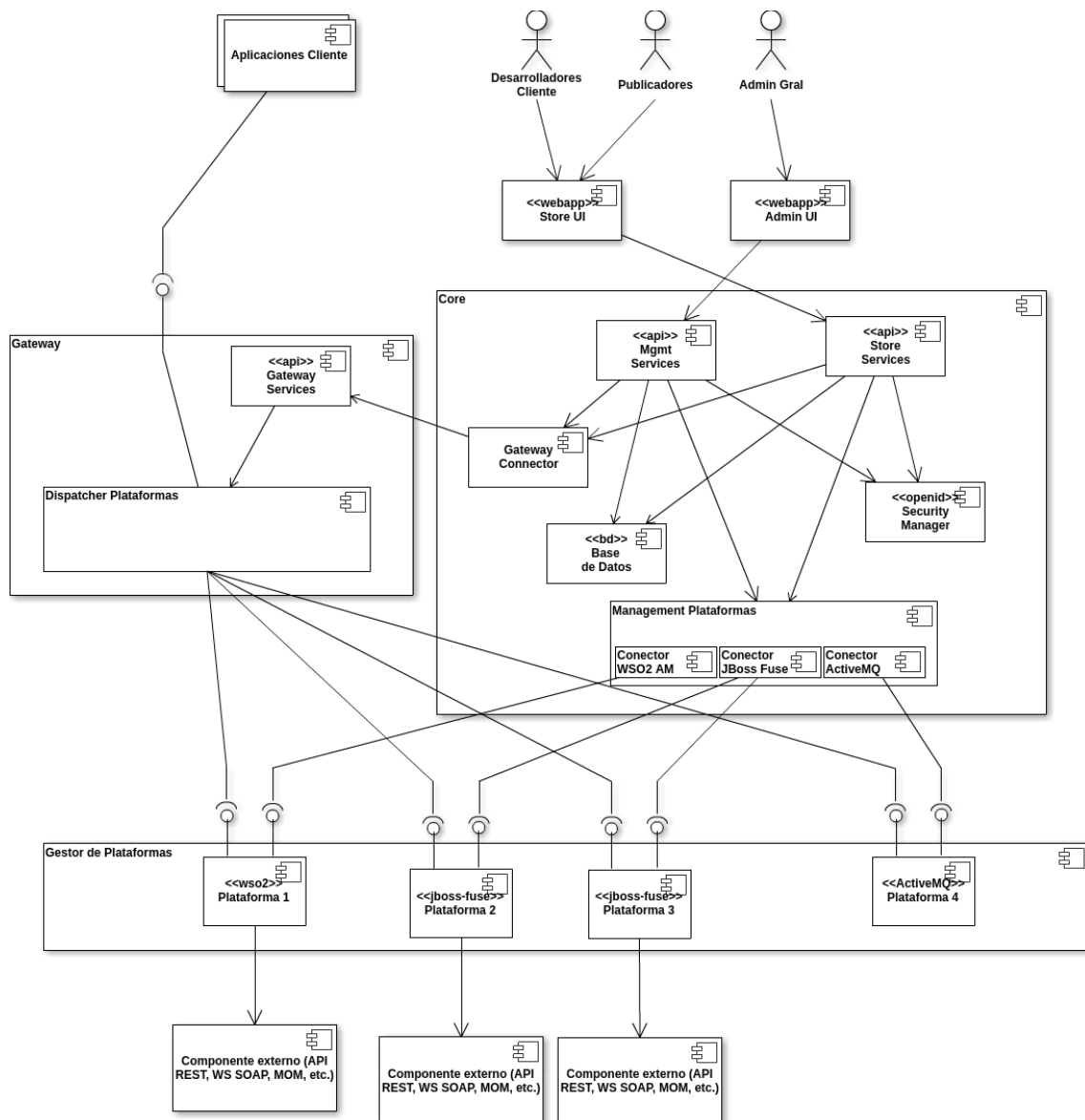


Figura 4 - Diagrama de componentes de la solución.

Para continuar con la presentación del diseño de la solución, como paso siguiente se realiza una descripción funcional de cada uno de los componentes planteados dentro del diagrama presentado en la figura 4.

Frontends

Los Frontends Web son las interfaces de usuario en las que se ofrecen las funcionalidades del sistema. Se distinguen dos tipos de Frontends:

- Un frontend administrativo llamado Admin UI, que ofrece todas las funcionalidades de gestión de la plataforma a un administrador general, propio de la plataforma.
- Un frontend de gestión para usuarios de organizaciones externas, llamado Store UI, donde se ofrecen las funcionalidades de la plataforma que permite a las organizaciones realizar la gestión de recursos, utilidades y plataformas creadas en el sistema. A su vez, también podrán realizar la gestión del acceso a utilidades de

organizaciones terceras (o propias) para poder conectar sus aplicaciones a las mismas.

Admin UI

El frontend Admin UI es una interfaz web que contiene todas las funcionalidades necesarias para llevar adelante la gestión completa de la plataforma. Los usuarios que harán uso de esta interfaz web serán usuarios propios de la plataforma, que tendrán conocimiento del funcionamiento de la misma, y tendrán a su cargo, tareas administrativas como ser:

- Gestión de la paramétrica general de la plataforma
- Consultas a la auditoría de la plataforma
- Gestión de usuarios de la plataforma (tanto administrativos como organizacionales)
 - Crear y actualizar usuarios
 - Habilitar y deshabilitar usuarios
 - Búsquedas y listados que permitan realizar las operaciones anteriores
- Gestión de roles de usuario
- Gestión de organizaciones registradas
 - Aprobar y revocar registros de organizaciones
 - Habilitar y deshabilitar organizaciones registradas
 - Modificación de datos de organizaciones registradas
 - Búsquedas y listados que permitan realizar las operaciones anteriores
- Gestión de especificaciones de plataformas
 - ABM de Especificaciones
 - Habilitar y deshabilitar tipos de plataformas
 - Búsquedas y listados que permitan realizar las operaciones anteriores
 - Gestión de capacidades de cada tipo de plataforma
 - ABM de Capacidades
 - Activar y desactivar capacidades dentro de cada tipo de plataforma
 - Búsquedas y listados que permitan realizar las operaciones anteriores

Store UI

Se trata de una interfaz web que permite a las organizaciones gestionar sus plataformas, los recursos de cada una y las utilidades que brindan, a su vez, solicitar colaboraciones con otras organizaciones para el uso de sus utilidades. Las tareas permitidas dentro de esta interfaz serán:

- Inicio y finalización de sesiones de usuario
- Gestión de la organización

- Registro de la organización
- Baja y modificación de datos de la misma
- Gestión de usuarios de la organización
 - ABM de usuarios
 - Cambiar de rol un usuario
 - Habilitar y deshabilitar usuarios
 - Búsquedas y listados que permitan realizar las operaciones anteriores
- Gestión de los ambientes de la organización
 - ABM de ambientes
 - Búsquedas y listados que permitan realizar las operaciones anteriores
- Gestión de plataformas de una organización
 - Activar y desactivar plataformas
 - Dar de baja plataformas con todos sus recursos
- Gestión de colaboraciones de la organización con otras organizaciones
- Gestión de recursos
 - ABM de los recursos de una organización, que incluye:
 - Gestión de la configuración de aquellas capacidades que utiliza un recurso
 - Selección de la plataforma para publicar el recurso, ya sea:
 - Selección de una plataforma previamente creada
 - O creación de una nueva plataforma
 - Compartir y dejar de compartir recursos con otras organizaciones
 - Solicitar recursos compartidos de otras organizaciones
 - Versionado de recursos
 - Gestión del ciclo de vida completo de cada recurso
 - Publicación final del recurso que finaliza con la creación de la utilidad en la plataforma
 - Búsquedas y listados que permitan realizar las operaciones anteriores
- Gestión de las utilidades
 - Activar y desactivar utilidades
 - Auditorías
 - Búsquedas y listados que permitan realizar las operaciones anteriores

Core

El Core de la plataforma se trata de un conjunto de componentes que ofrecen servicios a las interfaces gráficas descritas arriba. Estos servicios se ofrecerán en forma de APIs RESTful. La principal ventaja del uso de esta tecnología para la publicación de estos servicios es que opcionalmente se podrían publicar estas APIs de manera de que otras organizaciones puedan realizar tareas de mantenimiento de sus recursos a través de ellas, en lugar de hacerlo a través de la interfaz web Store UI, además de dar independencia tecnológica a los frontends con respecto a los servicios de este Core.

Los principales componentes identificados dentro del Core de la Plataforma son:

- **Store Services:** conjunto de servicios que brindan las funcionalidades necesarias para el uso de la Store UI tal y como fue especificada en el punto anterior.
- **Management Services:** conjunto de servicios que brindan las funcionalidades necesarias para el funcionamiento de la Admin UI especificada anteriormente.
- **Base de Datos:** motor de almacenamiento de la configuración realizada a través de los servicios anteriores.
- **Seguridad:** componente que gestiona los usuarios y los accesos de los mismos a las diferentes funcionalidades. Este componente es utilizado por todos los servicios (de Store y de Admin) de modo de verificar, tanto la validez de las sesiones de los usuarios, como la correctitud del acceso de cada usuario a cada funcionalidad.
- **Gateway Connector:** conector entre el Core y el Gateway
- **Manager de Plataformas:** se trata de un componente que permite dialogar para gestionar los recursos y utilidades dentro de cada plataforma

Store Services

Los servicios expuestos aquí se correlacionan con las funcionalidades descritas en el Store UI. Se trata de una API RESTful que proporciona las funcionalidades a dicha UI.

La seguridad es manejada a través del componente de Seguridad, el almacenamiento se realiza en la Base de Datos, y la comunicación con las plataformas se realiza a través del Manager de Plataformas.

A su vez, para la publicación de nuevas utilidades, la creación de nuevas plataformas y otras funcionalidades que modifiquen el acceso a las plataformas creadas y sus recursos, se comunicará con el Gateway a través de un conector específico.

Management Services

Los servicios expuestos aquí se correlacionan con las funcionalidades descritas en el Admin UI. Se trata de una API RESTful que proporciona las funcionalidades a dicha UI.

La seguridad es manejada a través del componente de Seguridad, el almacenamiento se realiza en la Base de Datos, y la comunicación con las plataformas se realiza a través del Manager de Plataformas.

A su vez, para la publicación de nuevas utilidades, la creación de nuevas plataformas y otras funcionalidades que modifiquen el acceso a las plataformas creadas y sus recursos, se comunicará con el Gateway a través de un conector específico.

Base de datos

La base de datos se encargará de almacenar únicamente información sobre la configuración de los elementos publicados en la plataforma y no sobre su uso.

Seguridad

Este componente se encargará de registrar los usuarios, iniciar y finalizar sus sesiones, gestionar los accesos a las funcionalidades, dependiendo de los roles que posean los usuarios.

Gateway Connector

Componente que realizará la comunicación entre el Core y el Gateway para la gestión de las diferentes funcionalidades que involucran modificaciones de configuración del Gateway.

Manager de Plataformas

Este manager contendrá una variedad de conectores cada uno de los cuales tendrá la capacidad de realizar las configuraciones necesarias para dar de alta, modificar y dar de baja recursos dentro de cada tipo de plataforma. Existirá un conector por tipo de plataforma, lo que permite la flexibilidad necesaria para incorporar a futuro nuevos tipos de plataforma, y realizando modificaciones en los conectores será que se pueden actualizar las mismas.

Gateway

El Gateway es el encargado de realizar la interfaz entre las aplicaciones cliente y los recursos publicados por las diferentes plataformas.

De esta manera se provee de una única interfaz, potencialmente con un único endpoint, para la comunicación de las aplicaciones con los recursos expuestos en las diferentes plataformas gestionadas a través del sistema, permitiendo así un grado más de facilidad de uso de todos los componentes por parte del programador de la aplicación cliente.

A su vez, guardará una traza de acceso a cada recurso, que podrá ser consultada en caso de fallas, auditorías y funcionamientos no esperados de los recursos.

A grandes rasgos se identifican dos componentes:

- Dispatcher: despachador de los pedidos de las aplicaciones cliente hacia las diferentes plataformas
- Gateway Services: API RESTful que ofrece las funcionalidades de configuración hacia el Core de la Plataforma

Dispatcher

Este es el componente que recibe los pedidos de las aplicaciones clientes y los envía a la plataforma correcta, según la URI que se está invocando, para lo cual se deberá mantener un catálogo de URIs que se define por cada utilidad.

Este catálogo deberá ser mantenido de manera externa, por ejemplo por el Core, y será administrado en el Dispatcher a través del Gateway Services.

Gateway Services

Se trata de un conjunto de servicios publicados en forma de APIs RESTful que ofrecen las funcionalidades necesarias para que el Core pueda realizar las configuraciones de manera de que el Dispatcher pueda funcionar correctamente con todas las plataformas creadas en el sistema, y todas las utilidades ofrecidas en cada plataforma.

Las funcionalidades principales de esta API son:

- Gestión de plataformas
 - Creación, habilitación y deshabilitación de plataformas
 - Búsqueda de plataformas
- Gestión de utilidades
 - Creación, habilitación y deshabilitación de utilidades dentro de una plataforma, que incluye la definición de la URL que se utilizará para el uso de una utilidad
 - Búsqueda de utilidades de una plataforma
- Gestión de los accesos a las utilidades
- Auditoría por plataforma y por utilidad, es decir la consulta a las trazas dejadas por el uso de las diferentes utilidades y plataformas.

Gestor de Plataformas

Llamamos gestor de plataformas al software de base que servirá para administrar físicamente las plataformas, crearlas, mantenerlas y actualizarlas, a nivel de infraestructura.

Este gestor manejará templates de plataformas, a partir de los cuales, podrá generar nuevos contenedores donde ejecutarán las plataformas, los cuales serán administrados por el propio gestor.

4 Vista de procesos

El objetivo de esta vista es presentar los procesos que están involucrados dentro de los escenarios planteados.

Como se presentó en el capítulo de análisis, existe una gran cantidad de requerimientos que se deben cubrir, dentro de los cuales, existen algunos que fueron considerados críticos por ser aquellos que definen el comportamiento principal de la plataforma, y a su vez, definen las características principales de cada componente.

Para definir estos requerimientos críticos se partió de la definición de ciertos escenarios también en el capítulo de análisis, y a partir de ellos es que se desprenden que los procesos que se consideran críticos dentro de la plataforma son los siguientes:

- Creación de Plataforma
- Alta de Recurso
- Solicitud de Recurso
- Utilización de Utilidad

La forma en que se presentará cada uno de estos procesos será a través de capturas de pantalla realizadas para mostrar la interacción del usuario con el sistema en cada uno de ellos, y de diagramas de comunicación de componentes que muestran el comportamiento de cada proceso, desde el punto de vista de la interacción entre los componentes.

Creación de Plataforma

En este proceso, el usuario selecciona un tipo de plataforma a crear y el sistema deberá dejar disponible dicha plataforma de modo que el usuario pueda agregar recursos a ella, y luego de ser publicados, los mismos puedan ser utilizados, tanto por la organización a la que el usuario pertenece, como por todas aquellas organizaciones que el usuario disponga.

La interacción con el usuario en este caso de uso se realiza como sigue. Como se muestra en la figura 5, para la creación de una plataforma el usuario únicamente deberá indicar cuál será el nombre de la misma.

En el momento de finalizar la creación, se dispara el proceso interno que permite la creación de dicha plataforma.

Figura 5 - Interfaz de usuario propuesta para la creación de plataformas.

A continuación se presentará en la figura 6 el flujo entre los diferentes componentes de la arquitectura que se seguirá para la creación de una nueva plataforma. El siguiente diagrama de comunicación representa dicho flujo:

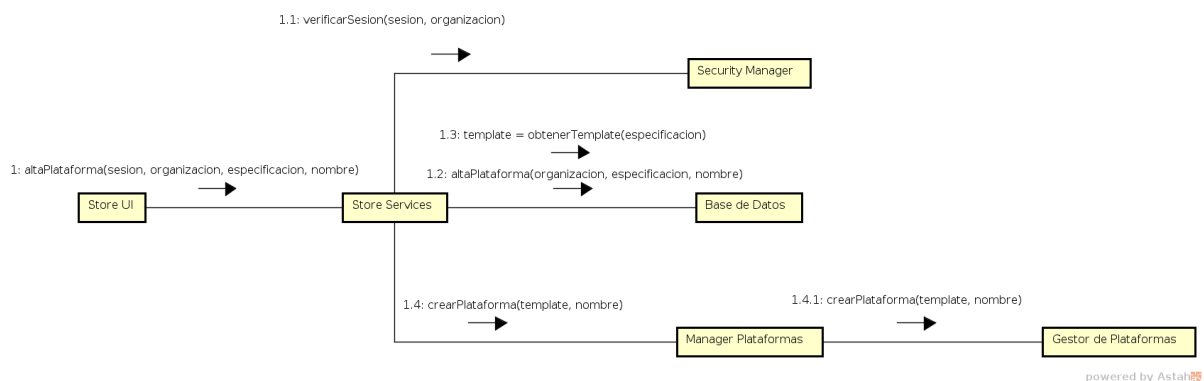


Figura 6 - Diagrama de comunicación entre componentes para la creación de plataformas.

El flujo de la figura 6 muestra como la operación de alta de plataforma en el Store Services gestiona primero la seguridad de la operación, delegando en el Security Manager la verificación de seguridad.

Luego almacena en base de datos la nueva plataforma, y obtiene de la misma el template asociado a la especificación que se debe utilizar a través del Manager para crearla físicamente.

Finalmente el último paso es la creación física de la plataforma. Esta acción se delega en el Manager de Plataformas, que no hará más que delegar nuevamente la acción en el Gestor de Plataformas. Este componente recibe el template que debe utilizar y el nombre de la plataforma, y a partir de estos datos, generará un nuevo contenedor donde dejará disponible además, la plataforma configurada y ejecutando.

Alta de Recurso

El proceso especificado en el escenario llamado alta de recurso involucra el uso de varias funcionalidades del sistema. En primer lugar, el usuario seleccionará las capacidades que tendrá el recurso y configurará cada ítem, dando paso luego a la selección de la plataforma donde se dará de alta el recurso, pudiendo ser ésta una nueva plataforma o una ya creada. A efectos de simplificación del escenario, y dado que ya existe un escenario donde se muestra el alta de una nueva plataforma, en este proceso se mostrará en detalle únicamente la selección de una plataforma previamente creada por la organización que da de alta el recurso.

Para mostrar la interacción con el usuario, se armaron capturas de pantallas de una interfaz de usuario ejemplo de modo de poder mostrar con detalle los diferentes pasos que sigue el usuario.

La figura 7 muestra la interfaz completa de alta de recurso. Se distinguen los tres espacios diferentes de acciones que se deben completar para crear el recurso.

En primer lugar, el usuario deberá ingresar los datos básicos del recurso, como se muestra en la figura 8.

Los datos que se deben ingresar en esta instancia son:

- Nombre: nombre descriptivo del recurso
- URI: uri que se utilizará para el recurso. El valor que se ingresa en este campo formará parte de la URI final con la que se identificará el recurso en el sistema
- Versión: versión del recurso.
- Tipo de acceso: público para que el recurso pueda ser utilizado sin restricciones de acceso en el momento que se publique, o privado para que el recurso pueda ser utilizado únicamente por la organización que lo crea y por todas las organizaciones a las cuales la organización dueña le apruebe las solicitudes de colaboración para este recurso.

Aquí podrá dar de alta nuevos recursos.

Home

Recursos

Organización

Plataformas

Creación de nuevo recurso

Nombre:

URI:

Versión:

Tipo de Acceso:

Público

Selección de capacidades

GuJS evaluation
(c) 1998-2016 Northwoods Software
Not for distribution or production use
nw.woods.com

Endpoint SOAP

Cóla de mensajes

Endpoint REST

Throttlér

Mapeo de producción

GuJS evaluation
(c) 1998-2016 Northwoods Software
Not for distribution or production use
nw.woods.com

Selección de plataforma

Crear nueva plataforma

Utilizar una plataforma existente

WSO2 API Manager

JBoss Fuse ESB

Apache Servicemix ESB

Apigee API Manager

Active MQ

WSO2 1

WSO2 2

Fuse 1

Cancelar

Finalizar

Figura 7 - Interfaz de usuario propuesta para la creación de recursos.

Creación de nuevo recurso

Nombre:

URI:

Versión:

Tipo de Acceso:

Público

Figura 8 - Sección de datos básicos del recurso en la interfaz de usuario propuesta.

En segundo lugar, el usuario seleccionará las capacidades que formarán parte del recurso. La figura 9 muestra a modo de ejemplo, una interfaz de usuario donde se permite realizar esta selección.

En dicha figura se puede ver cómo, a partir de una paleta de capacidades disponibles en el sistema, el usuario irá seleccionando aquellos que desea que estén presentes en su recurso, armando así un diagrama de flujo, que será el flujo por donde pasará cada solicitud al recurso que se está creando.

En esta paleta se ven 3 tipos diferentes de elementos, distinguidos por colores y formas:

- Los elementos azules indican qué tipo de entrada tiene el recurso (en los casos que se están presentando solamente se muestran tres tipos de endpoint: API REST, Servicio SOAP o Cola de Mensaje), pero pueden llegar a configurar otros tipos de endpoints.
- Los elementos rojos son las diferentes capacidades que se pueden configurar dentro de un recurso. Los elementos que se muestran son los que previamente fueron nombrados: el Throttler, el Mapeo de Parámetros, Endpoints a los que se puede invocar (SOAP, REST y Colas de Mensaje), y otros tipos de capacidades que se nombran más adelante en la sección de casos de usos.
- El elemento Fin que indica el fin del flujo de las solicitudes. Este elemento está asociado a los elementos de inicio que indican el tipo de entrada, ya que el tipo de entrada determina el tipo de salida que tendrá.

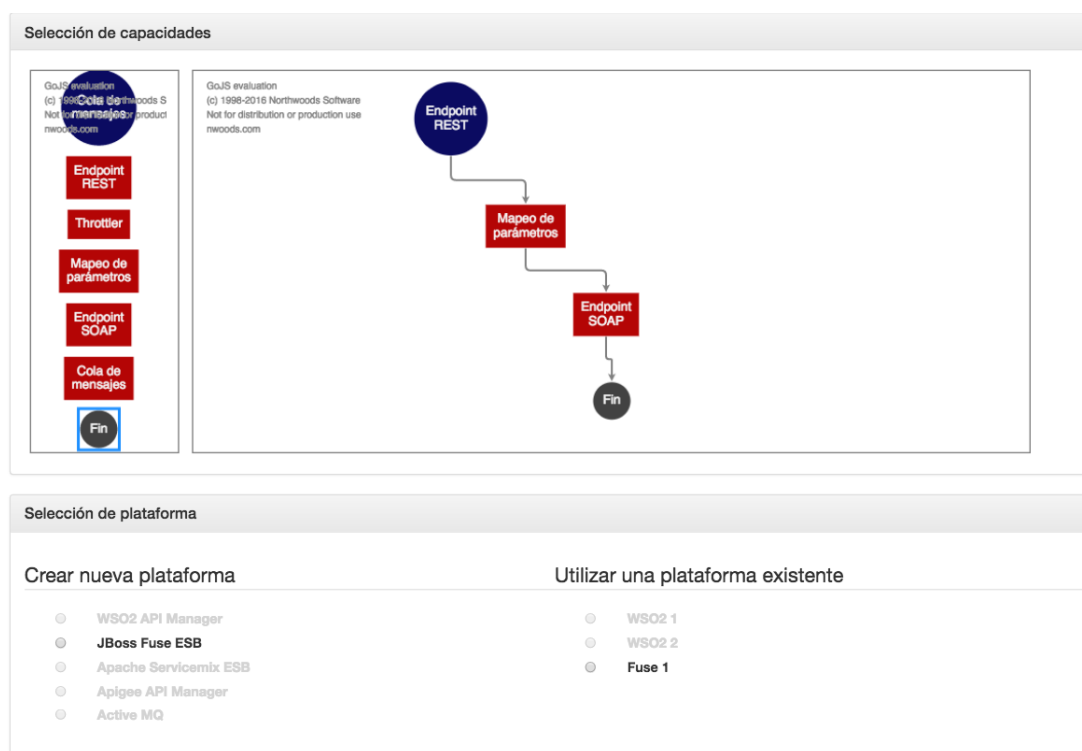


Figura 9 - Sección de selección de capacidades de la interfaz de usuario propuesta.

A medida que se van agregando capacidades al recurso, se muestra también cómo se van desactivando las plataformas (o especificaciones de plataforma en el caso de la creación de nuevas plataformas) que no cumplen con alguna de las capacidades seleccionadas.

A su vez, cada una de estas capacidades debe ser configurada para poder ser agregada al recurso. La imagen de la figura 10 muestra cómo configurar un endpoint SOAP, que tiene configurados ciertos atributos como ser:

- Método de obtención del WSDL: indica si el WSDL será cargado a la capacidad a insertando todo el texto del WSDL o debe ser obtenido a partir de una URL donde se encuentra disponible dicho WSDL.

- WSDL: en caso de que el WSDL se obtenga en modo texto, será el WSDL completo; en caso de que se deba obtener desde una URL, será la URL donde el sistema podrá obtener el WSDL para ser cargado en el sistema.
- URL del endpoint en ambiente de Testing: la URL completa donde se encuentra el endpoint con el servicio web SOAP disponible para ser consumido en ambiente de testing.
- URL del endpoint en ambiente de Producción: la URL completa donde se encuentra el endpoint con el servicio web SOAP disponible para ser consumido en ambiente de producción.

Figura 10 - Sección de configuración de capacidad de tipo endpoint SOAP de la interfaz de usuario propuesta.

Finalmente, como se muestra en la figura 7, se debe seleccionar la plataforma donde se alojará el recurso siendo las opciones las diferentes especificaciones de plataforma que permiten crear una nueva o las distintas plataformas previamente creadas por la organización. Cabe aclarar que tanto las especificaciones como las plataformas creadas disponibles para ser seleccionadas son únicamente aquellas que cuentan con las capacidades seleccionadas para dicho recurso.

El diagrama de comunicación de la figura 11 muestra el flujo de información dentro del sistema cuando se finaliza la interacción con el usuario al crear un recurso.

El flujo de la figura 11 muestra cómo Store Services gestiona también la seguridad de la operación a través del Security Manager, que controla que la sesión iniciada sea válida y a su vez pueda acceder al servicio.

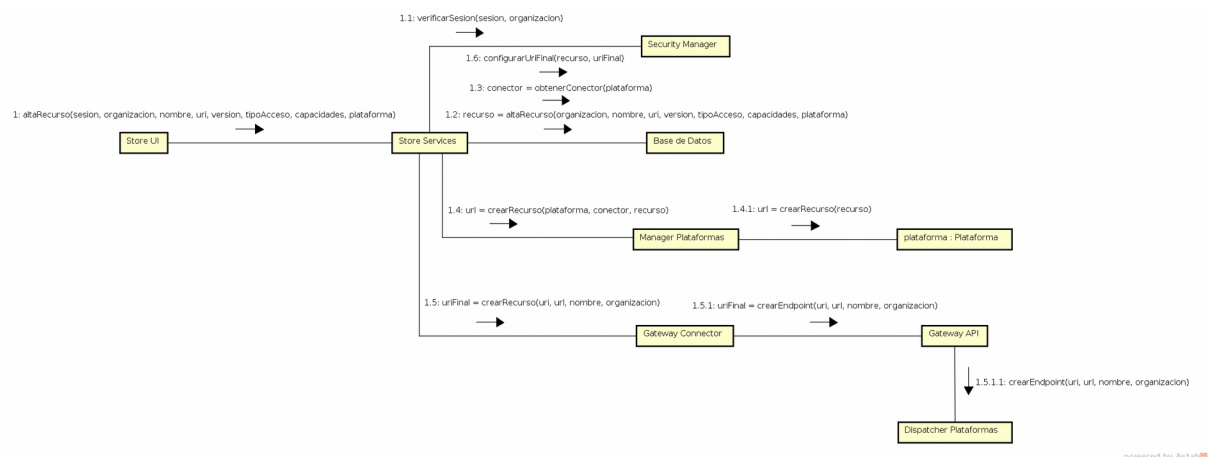


Figura 11 - Diagrama de comunicación de la creación de recursos.

Luego da de alta el recurso en la base de datos, y obtiene el nombre del conector que gestiona los recursos de la plataforma seleccionada.

Con el nombre del conector, le indica al Manager de Plataformas que debe crear el recurso. Este Manager de Plataformas, que en su interior contiene los diferentes conectores a las plataformas, selecciona el conector obtenido para la plataforma, y le solicita la creación del recurso. Este conector tendrá que estar implementado de modo que sea capaz de dialogar con la plataforma (cada conector dialoga con una plataforma) para que efectivamente se cree el recurso con las configuraciones realizadas por el usuario en la plataforma seleccionada. Esta creación generará una URL relativa a la plataforma que deberá ser devuelta.

Finalmente, el Store Services le solicita al Gateway a través del Gateway Connector que se conectará a la API publicada por el Gateway para generar un endpoint en el Dispatcher de Plataformas. Este Dispatcher deberá ser capaz de generar una URI que mapee con la URL interna que la plataforma asignó al recurso, y a su vez, deberá asignar el permiso de acceso únicamente a las claves de la organización que lo creó.

La URI final entonces del recurso será una URI del Gateway que será mapeada con la URL interna de la plataforma en el propio Dispatcher.

Solicitud de Recurso

A través del proceso de solicitud de recurso (y posterior aprobación de dicha solicitud), una organización pide a otra organización acceso a un recurso. Este acceso puede ser monetizado, dependiendo de las configuraciones de monetización que la organización dueña haya destinado al recurso.

La interacción con el usuario de la organización solicitante es lo que se describe a continuación.

Primeramente, el usuario de la organización deberá buscar el recurso al que desea solicitar acceso, como se puede apreciar en la figura 12.

En la interfaz mostrada en la figura 12 se puede ver al usuario ingresando a una interfaz donde puede filtrar los diferentes recursos creados en el sistema, tanto propios como ajenos. Para aquellos recursos que no son propios, se muestra si ya está compartido o no.

Los recursos creados como públicos no necesitan ser compartidos, en cambio aquellos que son privados sí. En caso de no estar compartido, se puede seleccionar la opción de Solicitar, y al hacerlo, se muestra una interfaz como la de la figura 13, donde se solicitará la colaboración a través de las tarifas definidas por el creador del recurso.

Dicha imagen muestra la interacción con el usuario a la hora de solicitar una colaboración para un recurso. En este caso, el recurso está configurado de manera que existen tres tipos de tarifa que se aplican, por cantidad de usos en cierto período de tiempo.

Aquí podrá buscar recursos y solicitar colaboración de otras organizaciones.

Home Recursos Organización Plataformas

Búsqueda de recursos

Nombre:

URI:

Organización:

Tipo de Acceso:

Filtrar

Resultados de su búsqueda

Id	Nombre	Uri	Versión	Organización	Tarifa	Tipo de acceso	Accesible
3	API Cobro	cobro	v3	Fantasia S.A.	No aplica	Publico	Propio
10	API Ventas	ventas	v2	Fantasia S.A.	No aplica	Privado	Propio
13	WS Usuarios	usuarios	v1	Fantasia S.A.	No aplica	Privado	Propio
24	WS Mensajería	mensajes	10.1	Tel Co.	Tarifa Básica	Privado	Compartido
28	API Consulta Cliente	consulta	2.3	Tel Co.	No compartido	Privado	Solicitar
61	WS Consulta Identidad	consulta-ci	v1	Instituto de Gobierno	No compartido	Privado	Solicitar

Figura 12 - Interfaz de usuario propuesta para la búsqueda de recursos.

Una vez que se envía la solicitud, la organización dueña del recurso podrá aprobar la solicitud para que la organización solicitante pueda comenzar a utilizar el recurso.

Cabe aclarar que no está pensado que el sistema sea el encargado de gestionar el pago y facturación de las tarifas que se muestran, sino que esto deberá ser gestionado por fuera del sistema por la organización dueña del recurso en conjunto con la organización solicitante para luego aprobar la solicitud.

Búsqueda de recursos

Nombre:

URI:

Organizació:

Tipo de Acceso:

Seleccionar tarifa

- ☐ Tarifa Básica: 500 usos por Día por un precio de USD 1500,00 por Mes
- ☐ Tarifa Plus: 1000 usos por Día por un precio de USD 2500,00 por Mes
- ☐ Tarifa Premium: Infinite usos por Día por un precio de USD 5000,00 por Mes

Resultados de su búsqueda

Id	Nombre	Uri	Versión	Organización	Tarifa	Tipo de acceso	Accesible
3	API Cobro	cobro	v3	Fantasia S.A.	No aplica	Publico	Propio
10	API Ventas	ventas	v2	Fantasia S.A.	No aplica	Privado	Propio
13	WS Usuarios	usuarios	v1	Fantasia S.A.	No aplica	Privado	Propio
24	WS Mensajería	mensajes	10.1	Tel Co.	Tarifa Básica	Privado	Compartido
28	API Consulta Cliente	consulta	2.3	Tel Co.	No compartido	Privado	Solicitar
61	WS Consulta Identidad	consulta-ci	v1	Instituto de Gobierno	No compartido	Privado	Solicitar

Figura 13 - Interfaz de usuario propuesta para la solicitud de colaboración de recursos.

Continuando con el desarrollo del proceso, se muestra la figura 14, donde se puede ver un diagrama de comunicación que indica cómo se comunican los componentes al dar de alta una solicitud.

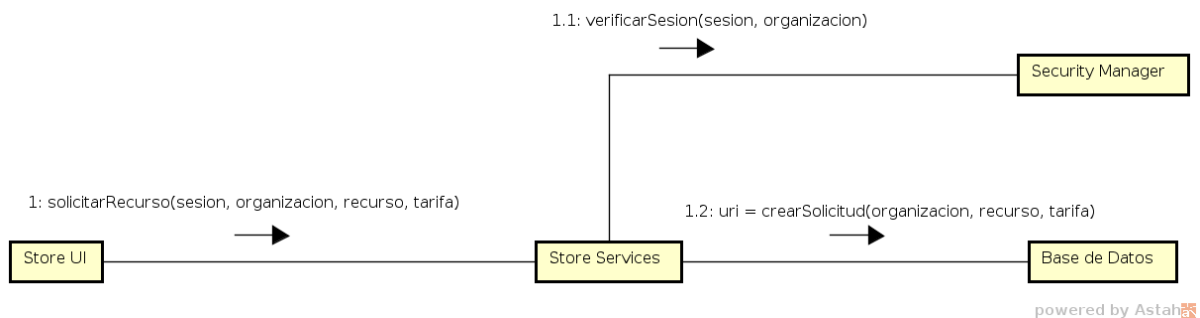


Figura 14 - Diagrama de comunicación de componentes para la solicitud de colaboración.

La interacción en este caso simplemente se resume al control de acceso que se verifica con el Security Manager, y luego dar de alta la solicitud en la base de datos.

De manera de entender el flujo completo, y no como parte de este escenario, se presenta también el flujo que muestra el proceso de aprobación de dicha solicitud, como se puede ver en el diagrama de comunicación de la figura 15.

En el diagrama se muestra que luego de verificar el acceso en el Security Manager y de registrar en la base de datos la aprobación de la solicitud, el Store Services solicita al Gateway a través del Gateway Connector darle acceso a la organización solicitante a la URI generada a la hora de la creación del recurso.

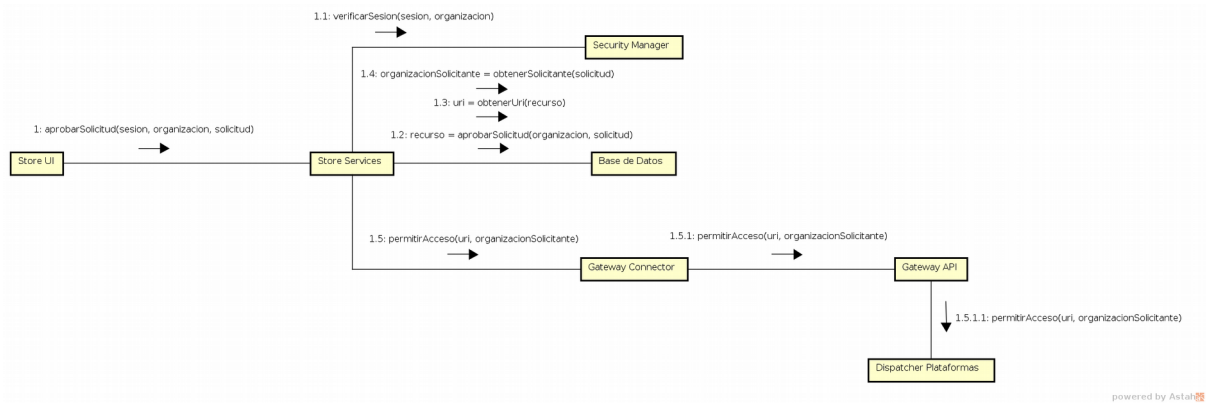


Figura 15 - Diagrama de comunicación de componentes para la respuesta a una solicitud.

Utilización de Utilidad

Finalmente, se presenta el proceso que sigue cuando se desea utilizar una Utilidad.

En este caso, la interacción del usuario depende del tipo de recurso que se esté accediendo. De todas maneras, en todos los casos, cada recurso tiene una URI única en el Gateway, que en definitiva, es lo que el usuario final accede enviando la solicitud correspondiente.

El flujo en este caso simplemente se limita a la verificación de la clave de acceso a la hora de recibir la solicitud en una URI determinada y luego reenviarla a la URL interna de la plataforma correspondiente, como se muestra en el diagrama de la figura 16.

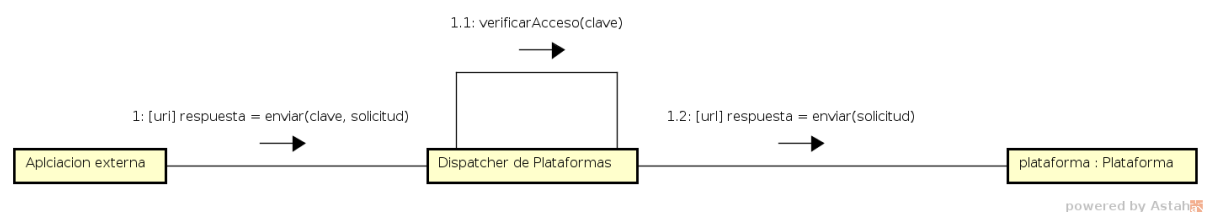


Figura 16 - Diagrama de comunicación de componentes en la utilización de una utilidad.

5 Vista física

En esta sección se presenta la vista física de la arquitectura. El objetivo es que a partir de los componentes planteados anteriormente, en conjunto con el análisis funcional del impacto de los procesos críticos en cada uno de los componentes y con las conclusiones funcionales de cada componente, se logre una distribución física de los mismos.

En el diagrama de despliegue de la figura 17 se presenta una alternativa para la distribución física de los diferentes componentes.

Dicho diagrama muestra una posible división en componentes físicos. Se optó por esta distribución ya que separa los componentes según el tipo de aplicación que debe implementarse en cada uno.

- **Web Server:** En primer lugar existen dos aplicaciones de interfaz de usuario que se agrupan en un Web Server (entre paréntesis, se da un ejemplo de un contenedor

web capaz de implementar dicho nodo, pero se debe aclarar que es a modo de ejemplo únicamente).

- **Application Server:** Luego se cuenta con un AppServer, es decir, un servidor de aplicaciones que sirve una aplicación de tipo empresarial que contiene los componentes principales del core del sistema. El ejemplo muestra también que este componente puede ser implementado sobre un JBoss Fuse.
- **Security Manager:** Como se vio en la vista de procesos, el Security Manager se encarga de la gestión de la seguridad, en cuanto a la identidad de usuarios y administración de sesiones. Por tal motivo, se puede implementar utilizando algún Identity Manager que soporte estándares en el manejo de usuarios y gestión de accesos, como por ejemplo, OpenId u OAuth2. Un ejemplo de este tipo de software puede ser el CAS Server de Apereo.
- **Base de Datos:** En este nodo se ejecutará algún motor de base de datos relacional, por ejemplo, PostgreSQL.
- **Gateway:** Realizando un análisis a partir de la vista de procesos, se puede decir que las funcionalidades de este Gateway son muy similares a las que proveen los softwares de Reverse Proxy, como nginx o httpd de Apache.
- **Management de Plataformas:** Este nodo resume todos aquellos conectores que se deban implementar para permitir gestionar las diferentes plataformas que se darán de alta en el sistema. Dada la naturaleza de cada conector, pueden ser desplegados de manera independiente, inclusive de manera independiente del core del sistema.
- **Gestor de Plataformas:** El último nodo de relevancia en la arquitectura física presentada es este Gestor de Plataformas, que se trata de un software capaz de realizar tareas de gestión de contenedores, dando de alta, baja, modificando características relativas a la plataforma desde un punto de vista físico. Un ejemplo claro de este tipo de software es el analizado en capítulos anteriores, como OpenShift de Red Hat.

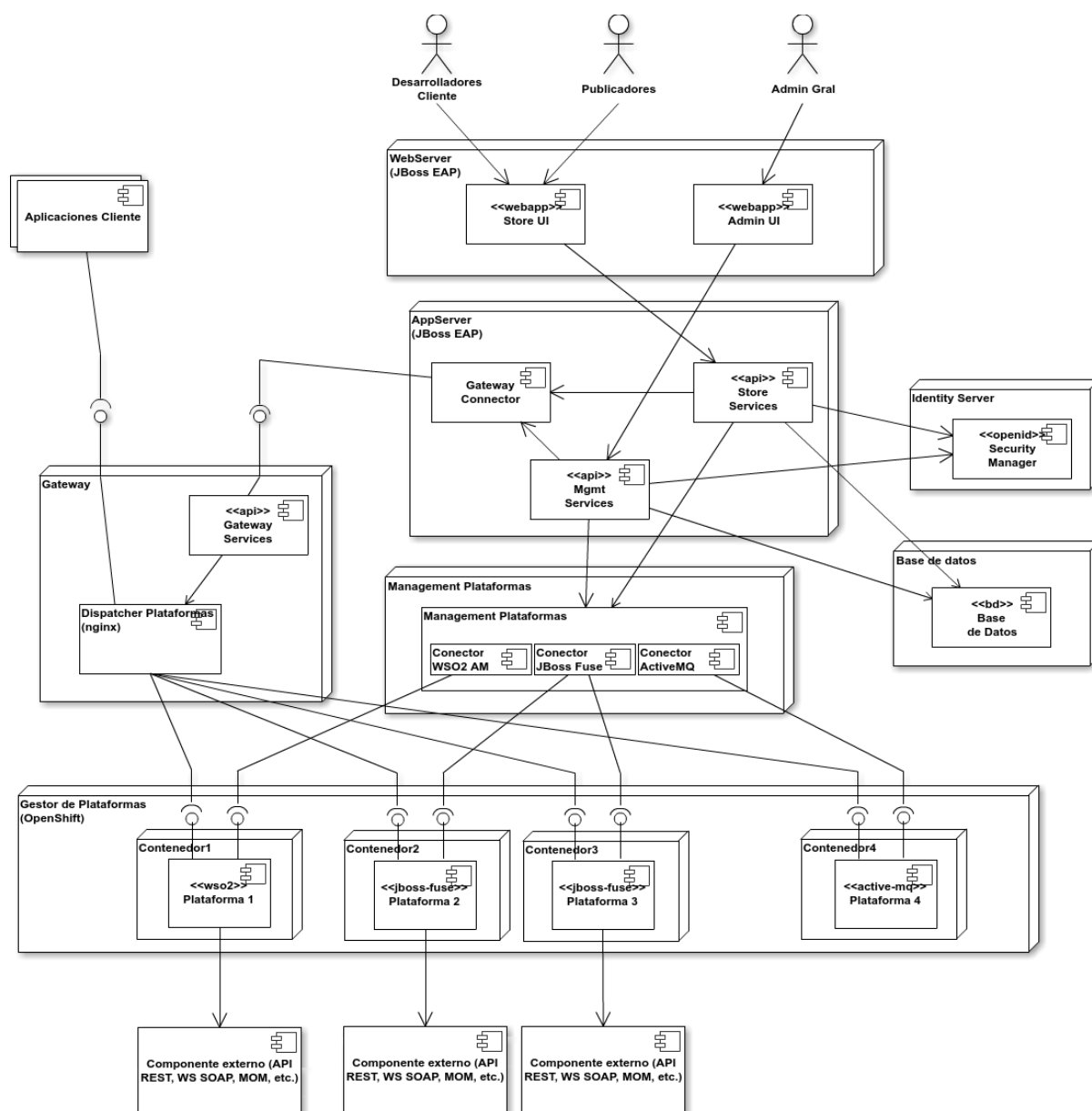


Figura 17 - Diagrama de despliegue propuesto para los componentes del sistema.

6 Vista de escenarios

Dentro de esta vista se realizará una introducción de los escenarios seleccionados para el diseño y de los casos de uso afectados. El detalle de todos los casos de uso identificados luego de plantear los requerimientos en la etapa de análisis se encuentra detallado en el Anexo IV.

Lo primero que se presentará en esta sección es la descripción general de los escenarios, para luego describir aquellos casos de uso involucrados en los escenarios.

Escenarios planteados

Cómo se planteó en la sección de análisis, se plantearon cuatro escenarios en los cuales se basó luego la definición del alcance. A continuación se brinda una descripción más detallada de cada uno.

Escenario 1: Creación de plataformas

Una Organización A registrada y logueada en nuestro sistema, define un nuevo recurso de tipo servicio REST, al cual le llama “Cobro”. Para definir y gestionar el mismo decide crear una plataforma de tipo API Manager y seleccionar WSO2 como producto de dicho tipo de plataforma.

Luego de definir y configurar cada una de las capacidades que la API utilizará, el sistema decidirá si es posible utilizar el API Manager que el usuario seleccionó para gestionar su API. En caso de ser posible, al momento de finalizar la creación de la API Cobro como recurso, el usuario deberá configurar la creación de esta nueva plataforma, indicando el nombre de la misma. Al confirmar la acción, el sistema deberá desplegar una nueva plataforma, utilizando la configuración de creación de la Especificación de Plataforma definida para el API Manager seleccionado.

Esta configuración de creación determinará la creación de un nuevo contenedor o una nueva máquina virtual, con el sistema operativo configurado, y el API Manager WSO2 instalado y ejecutando.

Luego, el sistema solicitará mediante una interfaz de creación de recurso al API Manager, la creación de la API REST “Cobro” creada por el usuario.

Por otro lado, una Organización B define de manera similar un servicio SOAP, para ser gestionado a través de un ESB (por ejemplo, JBoss Fuse), y al finalizar la definición del servicio SOAP, el sistema generará la nueva plataforma de manera similar, utilizando esta vez una especificación de plataforma de ESB seleccionada por el usuario.

Escenario 2: Creación de recurso

Una Organización A desea definir un servicio REST “Cobro” utilizando una plataforma de tipo API Management para su gestión.

Para esto, al momento de crear el recurso, el usuario de la organización selecciona las Capacidades para utilizar dentro del servicio, cada una de las cuales estará relacionada a una Especificación de Plataforma, que determina cuáles son todas las Capacidades que la Plataforma ofrecerá.

Algunos ejemplos de estas Capacidades pueden ser:

- Filtrado de contenido: permite transformar el contenido del mensaje entrante, ofuscando contenido sensible o modificando el mismo.
- Throttler: permite asegurar que un endpoint específico no sea sobrecargado, o que no se exceda un SLA con algún servicio externo. Esto incluye la configuración del máximo número de solicitudes por período, el período de tiempo, y otras parámetros más específicos.
- Validación: permite utilizar una expresión o un predicado para validar el contenido de un mensaje, es decir, garantizar que los mensajes sean válidos antes de intentar procesarlos. A su vez, la expresión de la validación podrá ser escrita en un conjunto de lenguajes, el cual será seleccionado por el desarrollador al momento de escribir la misma en el formulario de configuración web.

Luego de definidas las capacidades, el usuario de la Organización A selecciona una Plataforma de tipo API Management previamente creada por su Organización para realizar la gestión de la API, que cumpla con todas las capacidades previamente seleccionadas. Alternativamente, podría seleccionar la opción de generar una nueva plataforma de API Management, como se plantea en el Escenario 1.

Escenario 3: Solicitud de colaboración

Por otro lado una Organización B desea definir una composición otros servicios SOAP en una nueva API utilizando una plataforma ESB para su gestión, incluyendo el servicio REST publicado por la Organización A.

Para esto, previo a la definición del recurso para su servicio SOAP, deberá buscar la API que necesita de la Organización A y solicitar la colaboración de la misma para el uso de dicha API. Como parte de la solicitud, deberá indicar qué tipo de tarifas se aplicarán para su uso. Estas tarifas fueron previamente definidas por la Organización A y asociadas al recurso de la API REST en cuestión.

Esta acción generará una solicitud de colaboración pendiente de aprobación en una bandeja de entrada en la interfaz de usuario para todos los usuarios de la Organización A. Dicha solicitud podrá ser aprobada o revocada por cualquiera de estos usuarios.

En caso de ser aprobada, la Organización B podrá definir como parte de las capacidades en el momento de configurar su servicio SOAP, el recurso de la Organización A (como endpoint REST).

Escenario 4: Uso de utilidades

Una Organización C que ya tiene aprobada la colaboración de la Organización A para el uso de la API REST Cobro, desea utilizarla en una aplicación cliente que está desarrollando.

Utilizando las credenciales generadas por la propia Organización C para identificarse, invocará una operación de la API Cobro dirigida a una interfaz del sistema. En ese momento, el sistema validará que la Organización C pueda utilizar la API Cobro, y redirigirá la solicitud hacia el API Manager que gestiona la API.

Como parte de esta validación, se realizará el chequeo necesario para saber si se está cumpliendo con las condiciones definidas en la tarifa seleccionada por la Organización C al momento de solicitar la colaboración de la Organización A para este recurso.

Luego, el API Manager, ejecutará todas las Capacidades que fueron definidas por el usuario creador de la Organización A para dicha API en el momento de la creación, tal y como fueron configuradas por el usuario.

Finalmente, el API Manager invocará la URL del servicio que está detrás de la API Cobro definida, y aplicará más Capacidades a la respuesta, hasta obtener la respuesta definitiva que se encargará de despachar a través del propio sistema.

Casos de uso por escenario

A continuación se presentarán los casos de uso críticos, mostrando un diagrama de casos de uso por cada uno de los escenarios planteados. El objetivo de esta sección es contextualizar en cada escenario los casos de uso presentados.

Escenario 1: Creación de plataformas

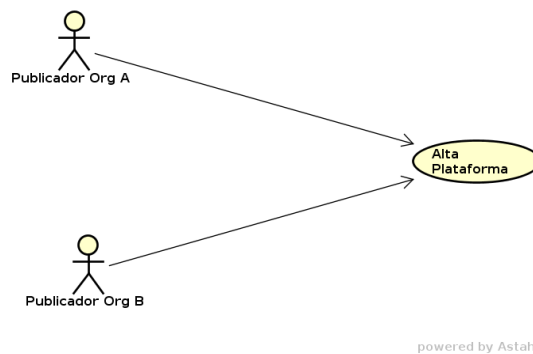


Figura 18 - Diagrama de casos de uso para el Alta de Plataforma.

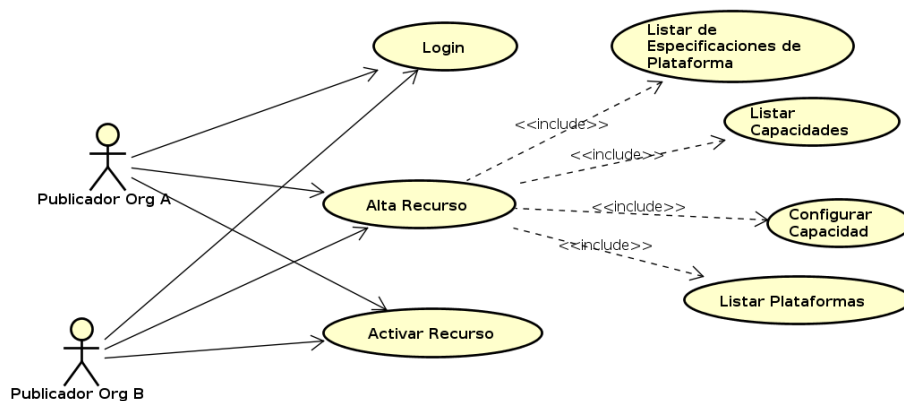
Para este escenario, el caso de uso crítico es el alta de plataforma, a través del cual el sistema crea toda la infraestructura virtual para luego desplegar la propia plataforma.

Posteriormente, configura todas las capacidades que desea utilizar dentro de la plataforma para finalmente dar de alta el recurso.

Escenario 2: Creación de recurso

Como indica el diagrama de casos de uso de la figura 19, los principales casos de uso que están involucrados en este escenario son:

- Login: necesario para poder ingresar con el publicador de la Organización A.
- Alta de Recurso: ambos publicadores deberán dar de alta sus recursos. Este caso de uso, incluye otros caso de uso adicionales como:
 - Listar Especificaciones de Plataforma, donde se selecciona el tipo de plataforma sobre el que desean desplegar sus recursos.
 - Listar Capacidades, donde se selecciona cuáles serán las capacidades que su recurso utilizará.
 - Configurar Capacidad, donde las capacidades seleccionadas son configuradas.
 - Listado de Plataformas, que permite al usuario seleccionar una plataforma previamente creada por su Organización.
- Activar Recurso: luego de darlo de alta, deberán Activar el Recurso principalmente el usuario de la Organización A de modo que el usuario de la Organización B pueda utilizarlo.

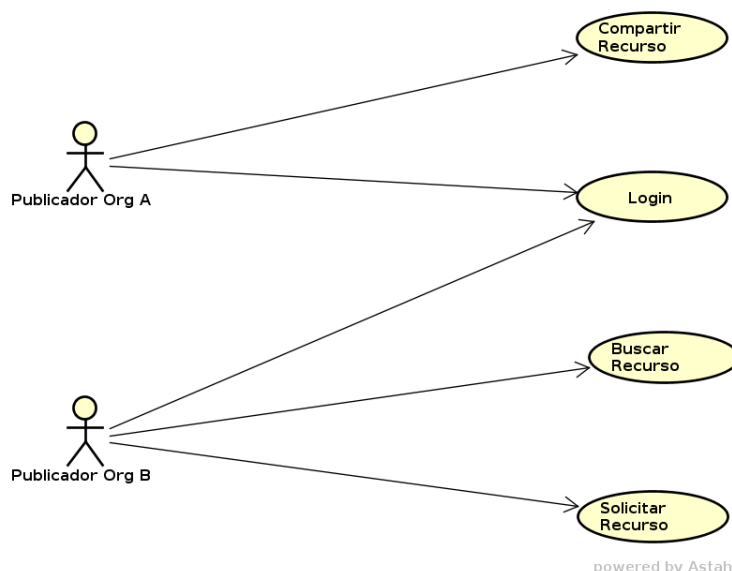


powered by Astah

Figura 19 - Diagrama de casos de uso para el Alta de Recurso.

Escenario 3: Solicitud de colaboración

Como muestra el diagrama de casos de uso de la figura 20, los casos de uso involucrados en este escenario son los siguientes:



powered by Astah

Figura 20 - Diagrama de casos de uso para la Solicitud de Colaboración.

- Login: necesario para poder ingresar con el publicador de la Organización B para realizar la búsqueda del recurso y solicitar la colaboración, como con el publicador de la Organización A para aprobar la solicitud.
- Búsqueda de recurso: el publicador de la Organización B deberá buscar el recurso de la Organización A para poder realizar la solicitud de colaboración.
- Solicitar Colaboración: el publicador de la Organización B deberá solicitar colaboración a la Organización A de manera de poder hacer uso de su recurso en el momento de dar de Alta su propio Recurso. Como parte de esta solicitud, la Organización B deberá seleccionar una de las tarifas configuradas por la Organización A para ese recurso.

- Aprobar Colaboración: el publicador de la Organización A deberá aprobar la colaboración con la Organización B para que éste pueda utilizar la API.

Escenario 4: Uso de utilidades

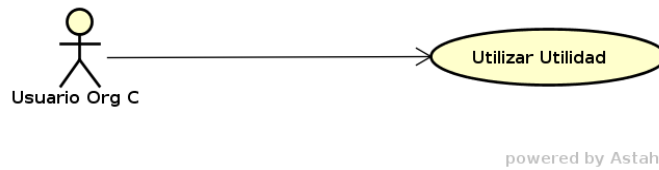


Figura 21 - Diagrama de casos de uso para el Uso de Utilidad.

En este escenario, el usuario hará uso del recurso publicado en forma de utilidad. Para este escenario el único caso de uso que está involucrado es el uso de utilidad, como se muestra la en la figura 21.

Anexo VII: Interfaz REST Conectores

1 Introducción

En este documento se presenta la especificación completa de la interfaz REST para ser implementada por los conectores con cada una de las plataformas que se pueden gestionar a través de la solución planteada.

2 Especificación

A continuación se detalla la especificación que debe seguir el servicio.

Operaciones

En la tabla 1 se presenta una lista de las operaciones REST que se espera que tenga la interfaz de los conectores que se implementen para cada plataforma.

En primer lugar existen tres operaciones sobre recursos: creación (PUT), modificación (POST) y eliminación (DELETE).

Por otro lado, existe una operación para generación de tokens de seguridad para invocar la API que se utiliza en cierto tipo de plataformas que ya proveen algún tipo de API para gestionar sus recursos.

Finalmente también se espera un servicio de echo, que permite saber el estado del conector, si está activo o no.

Path	Verbo	Campo	Valor
/recursos	PUT	Headers entrada	Content-Type: application/json
		Payload entrada	Objeto JSON de tipo RecursoDTO
		Headers salida	Content-Type: application/json
		Payload salida	Objeto JSON de tipo RespuestaOkDTO en caso de respuesta exitosa, o de tipo RespuestaErrorDTO en caso de respuesta errónea
		Códigos salida	• 200 - OK: en caso de éxito • 400 - Bad Request: en caso de ocurrir algún error de validación • 500 - Internal Server Error: en caso de ocurrir un error inesperado • 501 - Not implemented: en caso de no implementar esta operación

/recursos	POST	Headers entrada	Content-Type: application/json
		Payload entrada	Objeto JSON de tipo RecursoDTO
		Headers salida	Content-Type: application/json
		Payload salida	Objeto JSON de tipo RespuestaOkDTO en caso de respuesta exitosa, o de tipo RespuestaErrorDTO en caso de respuesta errónea
		Códigos salida	• 200 - OK: en caso de éxito • 400 - Bad Request: en caso de ocurrir algún error de validación • 500 - Internal Server Error: en caso de ocurrir un error inesperado • 501 - Not implemented: en caso de no implementar esta operación
/recursos	DELETE	Headers entrada	Content-Type: application/json
		Payload entrada	Objeto JSON de tipo RecursoDTO en parámetro en la url de nombre "recurso"
		Headers salida	Content-Type: application/json
		Payload salida	Objeto JSON de tipo RespuestaOkDTO en caso de respuesta exitosa, o de tipo RespuestaErrorDTO en caso de respuesta errónea
		Códigos salida	• 200 - OK: en caso de éxito • 400 - Bad Request: en caso de ocurrir algún error de validación • 500 - Internal Server Error: en caso de ocurrir un error inesperado • 501 - Not implemented: en caso de no implementar esta operación
/token	GET	Headers entrada	
		Payload entrada	Url de la plataforma a la que se le quiere generar el token en parámetro en la URL del servicio llamado "url"
		Headers salida	Content-Type: application/json
		Payload salida	Objeto JSON de tipo RespuestaOkDTO en caso de respuesta exitosa, o de tipo RespuestaErrorDTO en caso de respuesta

			errónea
		Códigos salida	• 200 - OK: en caso de éxito • 400 - Bad Request: en caso de ocurrir algún error de validación • 500 - Internal Server Error: en caso de ocurrir un error inesperado • 501 - Not implemented: en caso de no implementar esta operación
/echo	GET	Headers entrada	
		Payload entrada	
		Headers salida	Content-Type: application/json
		Payload salida	Objeto JSON de tipo RespuestaOkDTO en caso de respuesta exitosa con contenido "echo-ok", o de tipo RespuestaErrorDTO en caso de respuesta errónea
		Códigos salida	• 200 - OK: en caso de éxito • 400 - Bad Request: en caso de ocurrir algún error de validación • 500 - Internal Server Error: en caso de ocurrir un error inesperado • 501 - Not implemented: en caso de no implementar esta operación

Tabla 1 - Operaciones REST de la interfaz de los conectores con las plataformas.

Estas operaciones están disponibles también en el código fuente, fueron especificadas las cinco operaciones que se deben implementar en una interfaz Java en el proyecto "core-common": IRecursoService.

```
public interface IRecursoService {
    RespuestaDTO crearRecurso(RecursoDTO recurso);
    RespuestaDTO modificarRecurso(RecursoDTO recurso);
    RespuestaDTO eliminarRecurso(RecursoDTO recurso);
    RespuestaDTO echo();
    RespuestaDTO generarToken(PlataformaDTO plataforma);
}
```

Modelo de datos utilizado

En la tabla 4 se hace referencia a algunas entidades de tipo DTO para definir los datos que se transfieren en cada operación.

Los tres tipos de entidades que se hacen referencia son `RecursoDTO`, `RespuestaOkDTO` y `RespuestaErrorDTO`.

Estas entidades también se encuentran en el proyecto `core-common` del código fuente, en el package `model.dto`.

- `RecursoDTO`: esta clase describe todo lo que debe contener el recurso para poder ser dado de alta, que incluye:
 - `id`, `nombre`, `contexto`, `uriCompleta`, `version`, `tipoDeAcceso`, que son atributos generales sobre el recurso
 - `organizacionDueña`, que contiene toda la información sobre la organización dueña del recurso (y de la plataforma)
 - `plataforma`, que contiene toda la información sobre la plataforma sobre la que se está dando de alta el recurso, en particular contiene el campo `uriCompleta`, que contiene la URL donde está alojada la plataforma
 - `versionAnterior`, que contiene la especificación del recurso en su versión anterior
 - `capacidades`, que contiene la lista de capacidades que deben configurarse para el recurso
 - `puerto`, que, en el caso que corresponda, contiene el puerto donde debe ser publicado el recurso dentro de la plataforma
- `RespuestaOkDTO`: esta clase describe la respuesta a cualquiera de las operaciones mencionadas anteriormente:
 - `timestamp`, que es la fecha y hora en la que se ejecutó la operación
 - `status`, que contiene el código HTTP de estado de salida de la operación
 - `message`, que contiene el mensaje de salida de la operación
 - `path`, que contiene el path del recurso sobre el que se aplicó la operación
 - `contenido`, que contiene el payload de la respuesta
- `RespuestaErrorDTO`: esta clase contiene datos sobre los errores que pueden ocurrir al ejecutar una operación:
 - `timestamp`, que es la fecha y hora en la que se ejecutó la operación
 - `status`, que contiene el código HTTP de estado de salida de la operación
 - `message`, que contiene el mensaje de salida de la operación
 - `path`, que contiene el path del recurso sobre el que se aplicó la operación
 - `error`, que contiene el error ocurrido
 - `exception`, que contiene la excepción ocurrida