UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso de Profesionales en Actividad

TÍTULO: Despliegue de aplicación On-Premise en Cloud Computing utilizando servicios de AWS

AUTOR: Perez Acosta Cristian Abel

DIRECTOR ACADÉMICO: Lic. Ismael Pablo Rodríguez

DIRECTOR PROFESIONAL: Ing. Fernando García

CARRERA: Licenciatura en Sistemas

Resumen

Muchas empresas brindan sus productos de manera on-premise, haciendo que los usuarios que quieran adquirirlos deban disponer y administrar de todos los recursos necesarios para esas instalaciones. Unos de los principales inconvenientes de este esquema es que no todos los usuarios pueden hacer frente a dichos costos. Ante este escenario, las empresas se plantean la necesidad de hacer llegar sus productos a un sector más amplios de usuarios y ven como una opción la migración a entornos cloud. En esta tesina se explica y desarrolla los pasos necesarios para migrar y poner a disposición una aplicación on-premise en un entorno Cloud. Para ello se procede a desplegar una infraestructura utilizando los servicios y facilidades que brinda Amazon Web Services.

Palabras Clave

Cloud Computing, AWS, on-premise, procesos de negocio, Deyel, BPMN, Optaris

Conclusiones

Las opciones para trabajar en la nube son muy amplias y los proveedores de cloud computing compiten entre ellos para ofrecer la mayor variedad de productos y servicios a los usuarios.

Se logró desarrollar un marco teórico y los lineamientos necesarios para poder migrar un software on-premise a un entorno Cloud a partir del conocimiento de los servicios que nos brinda AWS.

La implementación del modelo planteado permite el despliegue y la configuración de un entorno propicio en el Cloud para instalar y ejecutar aplicaciones.

Trabajos Realizados

En este trabajo se explican los conceptos básicos de Cloud Computing y se realiza un análisis de los principales proveedores indicando sus fortalezas y debilidades. Se describe el software Deyel y se explica por qué se opta por AWS como proveedor para realizar las instalaciones de Deyel en el Cloud. Se plantea un modelo de infraestructura que hace uso de los servicios que ofrece AWS. En base al modelo propuesto, se desarrolló la infraestructura profundizando en cada uno de los servicios y explicando cómo es su utilización.

Trabajos Futuros

Evaluar si existe algún servicio o herramienta que brinde la opción de poder realizar toda la implementación de manera automática.

Analizar los diferentes tipos de instancias bajo demanda que brinda AWS y determinar cuáles serían convenientes de acuerdo al tipo de aplicaciones.

Investigar sobre el proceso de devops de una aplicación dentro de AWS.

Estudiar y analizar la posibilidad de automatizar la incorporación de nuevos servicios sin la intervención manual.

Alcanzar mejoras en performance de seguridad, alta disponibilidad, recuperación ante desastres, y funciones de automatización con menores tiempos de respuesta de soporte y superior experiencia.

Fecha de la presentación: Junio 2020

Despliegue de aplicación On-Premise en Cloud Computing utilizando servicios de AWS

Autor:
Perez Acosta Cristian Abel

Director Académico:
Lic. Ismael Pablo Rodríguez

Director Profesional:
Ing. Fernando Garcia

26 de junio de 2020

Agradecimientos

Quiero agradecer principalmente a mi familia que desde un principio me apoyó, acompañó y brindó la motivación necesaria para lograr cumplir este objetivo. A esos amigos que me dio la facultad que aun pasando los años siguen apoyándome.

Agradezco a mis compañeros de profesión, que gracias a su apoyo y contribución pude realizar esta tesina. A mi director académico, el Lic. Ismael Pablo Rodríguez por haberme guiado en el desarrollo de este trabajo. A mi director profesional, el Ing. Fernando García por su apoyo, paciencia y acompañamiento. A la empresa Optaris por darme la libertad para llevar adelante este trabajo.

Por último, quiero agradecer a la Universidad Nacional de La Plata por haberme brindado una educación pública, gratuita y de calidad y a la Facultad de Informática, por haberme formado y darme las herramientas necesarias para poder desempeñarme como profesional.

"Si crees totalmente en ti mismo, no habrá nada que esté fuera de tus posibilidades"

Gracias.

Resumen

Muchas empresas actualmente brindan sus productos de manera on-premise, haciendo que los usuarios que quieran adquirirlos deban disponer y administrar de todos los recursos necesarios para esas instalaciones. Unos de los principales inconvenientes de este esquema es que no todos los usuarios pueden hacer frente a dichos costos. Ante este escenario, las empresas se plantean la necesidad de hacer llegar sus productos a un sector más amplios de usuarios y ven como una opción la migración a entornos cloud.

En esta tesina se explica y desarrolla los pasos necesarios para migrar y poner a disposición una aplicación on-premise en un entorno Cloud. Para ello se procede a desplegar una infraestructura utilizando los servicios y facilidades que brinda Amazon Web Services.

Palabras Claves

Cloud Computing, AWS, on-premise, procesos de negocio, Deyel, BPMN, Optaris

Índice general

Capítulo 1

Introducción

1.1. Motivación

Cada día más empresas comienzan a brindar sus productos y soluciones implementadas sobre plataformas Cloud. Los productos de muchas de ellas nacieron como aplicaciones para instalarse de forma on-premise para luego adaptarse y migrar a ambientes Cloud. Esto les permitió poner sus productos a disposición por medio de Internet y llegar a un grupo mayor de usuarios para que puedan utilizarlos.

Desde su definición, las instalaciones on-premise requieren que el cliente disponga de recursos de hardware y software necesarios para poder ejecutar las aplicaciones a utilizar: infraestructura de red, servidores, sistemas operativos, bases de datos, etc. No todos los clientes pueden disponer y administrar todos los elementos necesarios para una instalación on-premise, por lo que su implementación, costo y mantenimiento les son desfavorables.

Debido a los puntos negativos que presentan las instalaciones on-premise para los clientes que no pueden disponer de dicha opción y la necesidad de llegar a brindar el producto a un sector más amplios de usuarios, las empresas comienzan a optar por poner a disposición sus productos y soluciones mediante el entorno Cloud.

De esta forma un usuario puede:

- Disponer del producto de forma inmediata ahorrando en costos (ya que no requiere inversión en infraestructura de hardware y software) y con solo poseer una conexión a internet para poder trabajar con él.

- Disponer de servicios de protección, resguardo e integridad de datos, servicios de mantenimiento y actualización de software y escalabilidad

sin preocupación y administrado por la empresa que brinda el servicio

Por lo expuesto anteriormente, el presente trabajo propone explicar mediante un ejemplo real como una empresa puede adaptar un producto de solución on-premise, con el fin de ponerlo a disposición por medio del Cloud.

1.2. Objetivo

El objetivo del trabajo de investigación es explicar y desarrollar los pasos necesarios para migrar y poner a disposición una aplicación on-premise en un entorno Cloud. Para ello se procede a desplegar una infraestructura utilizando los servicios y facilidades que brinda Amazon Web Services.

1.3. Contribución

Con el propósito de cumplir el objetivo propuesto se procede a instruir al lector en el mundo del Cloud computing y enfocarlo en los pasos necesarios para poder migrar un producto que se ejecuta actualmente en ambientes on-premise al entorno cloud. A lo largo de la tesina se desarrollará un marco teórico con los lineamientos necesarios para comprender el contexto en el que nos encontramos, haciendo hincapié en los conceptos básicos para comprender el mundo del cloud computing y sus relacionados. Se tomará como caso de estudio a "Deyel"[1], producto perteneciente a la empresa Optaris, el cual será el utilizado para ser migrado de instalación on-premise a cloud. Para llevar a cabo dicho proceso se hará foco en:

- Investigación de proveedores de servicios Cloud actuales: analizar las opciones de proveedores disponibles teniendo en cuenta las características y servicios que nos brinda.
- Justificación de elección de Amazon Web Services como proveedor de servicios Cloud.
- Adaptaciones al producto para ser ejecutado en ambientes Cloud.
- Amazon Web Services: armado de infraestructura que contendrá las instalaciones de la aplicación.
    - Especificación de servicios utilizados para implementar la infraestructura, lo que lleva a explicar:

[1] Página Oficial de Deyel: https://deyel.com

- Integración de servicios.
- Esquema de seguridad y conectividad.
- Motor de base de datos.
- Monitorización y observación de las instalaciones.

Al culminar el trabajo de investigación se espera haber desarrollado un marco teórico y los lineamientos necesarios que fundamenten la migración de un software on-premise a un entorno Cloud a partir del conocimiento de los servicios que nos brinda AWS para construir y armar una infraestructura funcional, segura y operativa. Por otro lado, se espera alcanzar el despliegue y configuración de un entorno propicio en el Cloud, con el fin que la empresa Optaris pueda instalar y ejecutar su solución de software Deyel.

1.4. Organización

Esta tesina está dividida en 8 capítulos, siendo éste el que presenta la motivación, objetivo y contribución que genera el tema. En el capítulo 2 se describe una serie de conceptos relacionados al contexto del cloud computing, se desarrollan los proveedores cloud más importantes de la actualidad, se explica el producto Deyel y se procede a elegir el proveedor a utilizar. El capítulo 3 se desarrolla el estado del arte de AWS, en donde se explican los servicios más importantes y el esquema de seguridad que maneja AWS. El capítulo 4 explica la estrategia general adoptada en la tesina, se explica el modelo de infraestructura a utilizar, los servicios involucrados y el esquema de seguridad a utilizar. En el capítulo 5 explica cómo se realiza la implementación del modelo de infraestructura propuesto en el capítulo 4. En el capítulo 6 detallamos las modificaciones específicas que se realizan sobre el producto Deyel para ejecutar en un ambiente cloud y en el capítulo 7 explicamos la creación de servicios en el modelo propuesto. Finalmente, en el capítulo 8 se plantean posibles extensiones del trabajo y se desarrollan conclusiones.

Capítulo 2

Trabajo Relacionado

2.1. Cloud Computing

Esta sección presenta una descripción general de la computación en la nube, incluida su definición y una comparación con los conceptos relacionados.

2.1.1. Historia

Cloud Computing es un modelo surgido recientemente que se está volviendo popular entre casi todas las empresas. Implica el concepto de servicios bajo demanda, lo que significa utilizar los recursos de la nube a petición y podemos escalar los recursos según la demanda. La computación en la nube, sin duda, proporciona beneficios sin fin y es un modelo rentable[1].

El término nube se ha utilizado históricamente como una metáfora de Internet. Este uso se derivó originalmente de su descripción común en los diagramas de red como un esquema de una nube, utilizada para representar el transporte de datos a través de las redes troncales del operador (que poseía la nube) a una ubicación en otro lado de la nube. Este concepto se remonta a 1961, cuando el profesor John McCarthy sugirió que la tecnología de tiempo compartido de la computadora podría conducir a un futuro en el que la potencia informática e incluso aplicaciones específicas podrían venderse a través de un modelo comercial de tipo de utilidad.[1].

Esta idea se hizo muy popular a fines de la década de 1960, pero en la década de 1970 la idea se desvaneció cuando quedó claro que las tecnologías

[1]https://computinginthecloud.wordpress.com/2008/09/25/utility-cloud-computingflashback-to-1961-prof-john-mccarthy/ (último acceso 11/04/2020)

de la época no podían sostener un modelo informático tan futurista. Sin embargo, desde el cambio de milenio, el concepto se ha revitalizado. Fue durante este tiempo de revitalización que el término computación en la nube comenzó a surgir en los círculos tecnológicos.

La palabra "nube" fue utilizada por el CEO de Google, Eric Schmidt, para describir el modelo de negocio de la prestación de servicios a través de Internet en 2006. Para exponer varias ideas, el término nube se utilizó como término de marketing [2]. Desde entonces, el término Cloud Computing se ha utilizado principalmente como un término de marketing en una variedad de contextos para representar muchas ideas diferentes. Ciertamente, la falta de una definición estándar de computación en la nube ha generado no solo exageraciones en el mercado, sino también una buena cantidad de escepticismo y confusión. Por esta razón, recientemente se ha trabajado en la estandarización de la definición de computación en la nube. Como ejemplo, el trabajo en [3] comparó más de 20 definiciones diferentes de una variedad de fuentes para confirmar una definición estándar. En este documento, vamos a adoptar la definición proporcionada por el Instituto Nacional de Estándares y Tecnología (NIST):

> *"Cloud computing es un modelo para permitir el acceso de red ubicuo, conveniente y bajo demanda a un grupo compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que pueden aprovisionarse y liberarse rápidamente con un mínimo esfuerzo de administración Interacción del proveedor de servicios."[4]*

2.1.2. Tecnologías Relacionadas

Cloud Computing a menudo se compara con las siguientes tecnologías, cada una de las cuales comparte ciertos aspectos con la computación en la nube:

- **Grid Computing:** es un paradigma de computación distribuida que coordina los recursos en red para lograr un objetivo computacional común. Cloud Computing es similar a Grid Computing en que ambos emplean recursos distribuidos para lograr objetivos de nivel de aplicación. Sin embargo, el Cloud Computing da un paso más al aprovechar las tecnologías de virtualización en múltiples niveles (plataforma de hardware y aplicación) para realizar compartición de recursos y aprovisionamiento dinámico de recursos.

- **Virtualización:** La virtualización es una tecnología que abstrae los detalles del hardware físico y proporciona recursos virtualizados para aplicaciones de alto nivel. La virtualización forma la base de la computación en la nube, ya que proporciona la capacidad de agrupar recursos informáticos de grupos de servidores y asignar o reasignar dinámicamente recursos virtuales a aplicaciones bajo demanda.

- **Computación automática:** la computación autónoma tiene como objetivo construir sistemas informáticos capaces de autogestión, es decir, reaccionar a las observaciones internas y externas sin intervención humana. El objetivo de la informática autónoma es superar la complejidad de gestión de los sistemas informáticos actuales. Aunque la computación en nube presenta ciertas características autónomas como el aprovisionamiento automático de recursos, su objetivo es reducir el costo de los recursos en lugar de reducir la complejidad del sistema.

En resumen, la computación en la nube aprovecha la tecnología de virtualización para lograr el objetivo de proporcionar recursos informáticos como una utilidad. Comparte ciertos aspectos con la computación grid y la computación autónoma, pero difiere de ellos en otros aspectos.

2.1.3. Arquitectura del Cloud Computing

Generalmente la arquitectura del Cloud Computing se puede dividir en 4 capas: la capa de hardware/centro de datos, la capa de infraestructura, la capa de plataforma y la capa de aplicación; todas se presentan en la Figura 2.1[2].

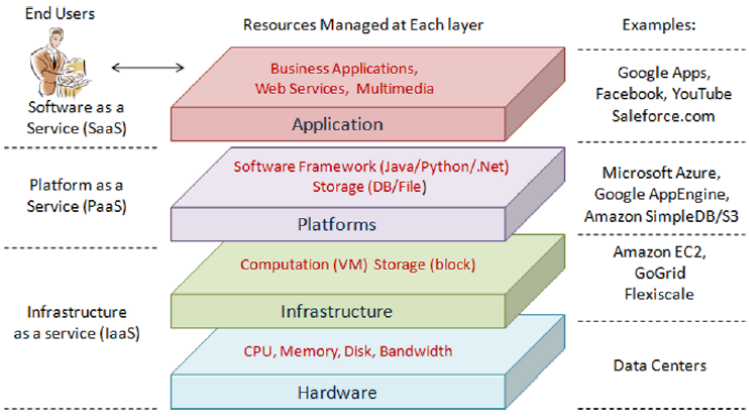


Figura 2.1: Arquitectura del Cloud Computing

Describimos cada una de ellas en detalle:

- **La capa de hardware:** Esta capa es responsable de administrar los recursos físicos del cloud, incluidos los servidores físicos, routers, switches y los sistemas de alimentación y refrigeración. En la práctica, la capa de hardware generalmente se implementa en lo que se denomina "data center" que por definición es una instalación para albergar un sistema de información de componentes asociados.

- **La capa de infraestructura:** También conocida como la capa de virtualización, la capa de infraestructura crea un conjunto de recursos de almacenamiento y computación al dividir los recursos físicos utilizando tecnologías de virtualización como Xen [5], KVM [6] y VMware [7].

- **La capa de plataforma:** Esta capa está constituida por los sistemas operativos y los frameworks de aplicaciones. El propósito de esta capa es minimizar la carga de implementar aplicaciones directamente sobre los contenedores de las máquinas virtuales.

- **La capa de aplicación:** En el nivel más alto de la jerarquía, la capa de aplicación consta de las aplicaciones Cloud. A diferencia de las aplicaciones tradicionales, las aplicaciones Cloud pueden aprovechar la función de escala automática para lograr un mejor rendimiento, disponibilidad y un menor costo operativo.

2.1.4. Modelo de negocio

Cloud Computing emplea un modelo de negocio basado en servicios. En otras palabras, los recursos de hardware y de plataforma se proporcionan como servicios a pedido. Sin embargo, en la práctica, los clouds ofrecen servicios que se pueden agrupar en tres categorías: software como servicio (SaaS), plataforma como servicio (PaaS) e infraestructura como servicio (IaaS). Procedemos a explicar cada una de ellas:

- **Infraestructura como servicio (IaaS):** se refiere a ofrecer servicios relacionados con el hardware. Estos incluyen servicios de almacenamiento o servidores virtuales. Un ejemplo de proveedor de IaaS es Amazon EC2 [8].
- **Plataforma como servicio (PaaS):** se refiere a proporcionar recursos de capa de plataforma, incluido el soporte del sistema operativo y los frameworks de desarrollo. Ejemplos de proveedores de PaaS incluyen Google App Engine [9], Microsoft Windows Azure [10] y Salesforce [11].
- **Software como servicio (SaaS):** se refiere a proporcionar aplicaciones a pedido a través de Internet. Salesforce.com[11] es considerado como el ejemplo más conocido de computación SaaS entre las aplicaciones empresariales. Otro ejemplos pueden ser Google Apps[2] y Rackspace [12].

[2]proporciona acceso en línea a través de un navegador web a las aplicaciones de oficina y comerciales

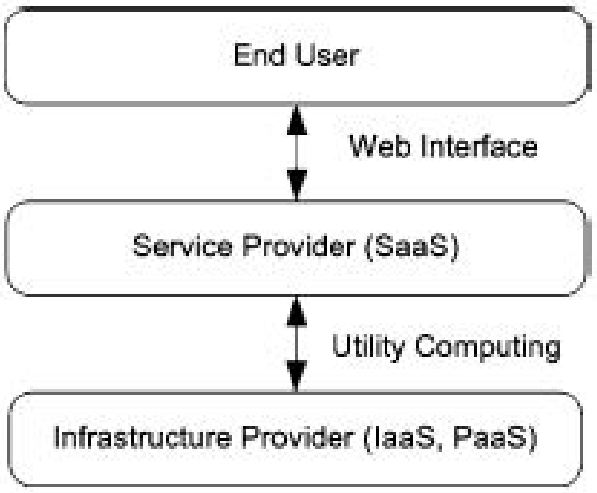


Figura 2.2: Modelo de Negocio del Cloud Computing

La Figura 2.2 representa el modelo de negocio de Cloud Computing. Según la arquitectura en capas, es completamente posible que un proveedor de PaaS ejecute su nube sobre la nube de un proveedor de IaaS. Sin embargo, en la práctica actual, los proveedores de IaaS y PaaS a menudo son parte de la misma organización (por ejemplo, Google y Salesforce).

2.1.5. Tipos de Cloud

Los cloud pueden clasificarse en:

- **Public Clouds:** en este tipo de Clouds los proveedores de servicios ofrecen sus recursos como servicios al público en general. Las nubes públicas ofrecen varios beneficios clave para los proveedores de servicios, incluida la ausencia de inversión de capital inicial en infraestructura y la transferencia de riesgos a los proveedores de infraestructura. Sin embargo, las nubes públicas carecen de un control detallado sobre los datos, la red y la configuración de seguridad, lo que dificulta su eficacia en muchos escenarios de negocios.

- **Private Clouds:** también conocidas como clouds internas, las clouds privadas están diseñadas para uso exclusivo de una sola organización.

Una cloud privada puede ser construida y administrada por la organización o por proveedores externos. Este tipo de nube ofrece el mayor grado de control sobre el rendimiento, la fiabilidad y la seguridad; sin embargo, a menudo son criticados por ser similares a las granjas de servidores propietarias tradicionales.

- **Hybrid Clouds:** una clouds híbrida es una combinación de modelos de cloud pública y privada que intenta abordar las limitaciones de cada enfoque. En una cloud híbrida, parte de la infraestructura de servicio se ejecuta en clouds privadas, mientras que la parte restante se ejecuta en clouds públicas. Las cloud híbridas ofrecen más flexibilidad que las cloud públicas y privadas. Específicamente, proporcionan un control y una seguridad más estrictos sobre los datos de las aplicaciones en comparación con las cloud públicas, al tiempo que facilitan la expansión y contracción del servicio a pedido. En el lado negativo, el diseño de una cloud híbrida requiere determinar cuidadosamente la mejor división entre los componentes de la cloud pública y privada.

2.1.6. Seguridad en entornos Cloud

La computación en la nube, sin duda, proporciona un muy buen servicio al usuario, pero aún muchas organizaciones no son compatibles con ella debido a los problemas de seguridad que puede presentar. Los principales problemas de seguridad son la seguridad de los datos y la protección de la privacidad. Estos problemas de seguridad dificultan que los gerentes o clientes respalden los servicios proporcionados por la computación en la nube [13]. Hay muchos riesgos asociados con la computación en la nube. Algunos de ellos son: la seguridad, leyes y regulación, riesgos de virtualización, falta de estándares y auditoría, costo viable no controlado, etc. El riesgo que es más preocupante es la Seguridad [14].

Los datos de los clientes pueden estar en la nube junto a miles de otros datos de otros clientes sin siquiera que cada uno de ellos sepa la ubicación de los mismos. Por lo tanto, los proveedores de servicios deben proporcionar un modelo que promueva la Confidencialidad, Integridad y Disponibilidad (CIA) [15]. La CIA se puede proporcionar mediante el cifrado de los datos, el control de acceso para evitar que usuarios no autorizados accedan a los datos y la copia de seguridad programada para garantizar la disponibilidad.

2.2. Cloud Computing vs On-Premise

Antes de hacer compromisos y optar por soluciones cloud, las organizaciones de todos los tamaños deben conocer los factores comerciales y los factores de costos involucrados en dichas acciones, para así poder determinar cuál de las opciones es posibles tomar. Los vendedores y cualquier empresa siempre buscan obtener la máxima cuota de mercado y que sea más atractiva.

2.2.1. Desventaja del modelo tradicional On-Premise

En la empresa se llama solución On-Premise a aquellos sistemas que son instalados en la propia empresa. Se trata de tener en «Casa» los servidores y el software que proporcionan un determinado servicio para la actividad desarrollada.

El tamaño y actividad de la empresa determinan el número de servidores y la dimensión de sus instalaciones. Son habituales los servidores de bases de datos SQL, los de almacenamiento de archivos, los de facturación, los de nóminas, los de ERP o CRM. Hasta hace no mucho, esta era la única alternativa que tenían las empresas. Algunas con departamento de IT y otras que usaban esos servidores tras la instalación por parte de algún proveedor. [16]

Problemas de las soluciones On-Premise en la empresa

A continuación, se listan algunos de los problemas más comunes de las soluciones on-premise:

- **Dependencia del proveedor:** Cuando una empresa compra hardware y software, quedan cautivos de un proveedor. Tras realizar esta inversión es muy difícil dar marcha atrás, se queda esclavizado por una infraestructura y un mantenimiento. Para cualquier tipo de incidencia es necesario contactar con el proveedor que en muchos casos enviará a un técnico y cobrará por el trabajo realizado.

  La dependencia incluye un coste oculto que inicialmente es difícil de percibir. Se trata del **coste de oportunidad**. Cuando una empresa instala servidores On-Premise espera que su inversión sea productiva al menos durante tres años. La informática avanza rápido y el hardware y software se quedan obsoletos. La cuestión primordial es que existe un coste de oportunidad intangible que hay que considerar seriamente. El coste de oportunidad consiste en que: durante ese tiempo

y con el rápido avance del Cloud Computing aparezcan aplicaciones o servicios más adecuados para la empresa. Aplicaciones que no van a poder ser aprovechadas porque ya se ha realizado la inversión en la infraestructura local.

- **Riesgo de la conexión a internet:** Mantener servidores On-Premise abiertos a conexiones externas por Internet es un gran riesgo. Una gran empresa puede permitirse tener un departamento IT encargado de labores de mantenimiento y de seguridad entre otras cosas. El mayor problema lo tienen las Pymes que no disponen de un departamento de este tipo. Actualmente el 85 % de los ataques se realizan sobre servidores de empresas. Saben que todos los días salen nuevos agujeros de seguridad en todos los sistemas. Conocen que estos sistemas On-Premise están desactualizados y carecen de medidas de seguridad adecuadas. Los propios routers que instalan las compañías de telecomunicaciones son un punto de debilidad.

- **Sin flexibilidad y sin escalabilidad:** A pesar de que las instalaciones On-Premise se realizan sobreestimando capacidades y esto repercute mucho en el coste inicial, terminan quedándose pequeñas y obsoletas. Los servidores locales no pueden aumentar ni reducir sus capacidades tal y como se hace en Cloud. Esto no solo ocurre con el hardware, también ocurre con el software que se adquiere por licencias o por puesto de trabajo. La tecnología está en constante evolución y cambio, los propios sistemas operativos cambian y han de actualizarse o sustituirse, la configuración inicial puede no servir transcurrido un tiempo.

- **Alto coste de instalación y mantenimiento:** La instalación inicial de cualquier sistema On-Premise tiene un alto coste. Han de adquirirse tanto el hardware como el software tan solo para ponerlo en marcha. En el coste de instalación han de incluirse la compra de servidores, la compra del software, elementos de red, su instalación y puesta en marcha. Los servidores son ruidosos y generan bastante calor, por ello es necesario buscar una ubicación bien refrigerada y que además no sea accesible fácilmente.

  La seguridad de un sistema On-Premise también depende de los elementos físicos involucrados, un robo o acceso indebido a estos elementos pueden provocar pérdidas irrecuperables para la empresa. El espacio ocupado es un factor a considerar pues tiene un coste depen-

diendo del valor inmobiliario de los metros cuadrados que tengan las oficinas.

Las instalaciones On-Premise son muy vulnerables a la estabilidad del suministro eléctrico. Los cortes en el suministro eléctrico o las subidas de tensión pueden provocar graves fallos en los sistemas, para evitarlos será necesario asumir también el coste de los elementos de protección eléctrica necesarios y su instalación.

A medio plazo los costes de mantenimiento superarán a los anteriormente descritos de instalación. Deberá disponerse de personal que lleve el mantenimiento del sistema, si se trata de personal interno será un coste de recursos y si se trata de un mantenimiento por parte de una empresa entonces supone un coste monetario.

- **Cuello de botella:** Es una consecuencia de la rigidez de los sistemas On-Premise. Si la empresa asume el riesgo de abrir algún servicio a Internet y hacerlo accesible desde el exterior emulando un servicio en la Nube casi nunca se tiene en cuenta este grave problema.

  Es muy sencillo y se basa en que la conexión a Internet no tiene una capacidad infinita. En una empresa la conexión está limitada por una capacidad que puede ser fácilmente superada cuando se suma el consumo interno más el externo. Este problema puede llevar a una saturación de la conexión que utiliza.

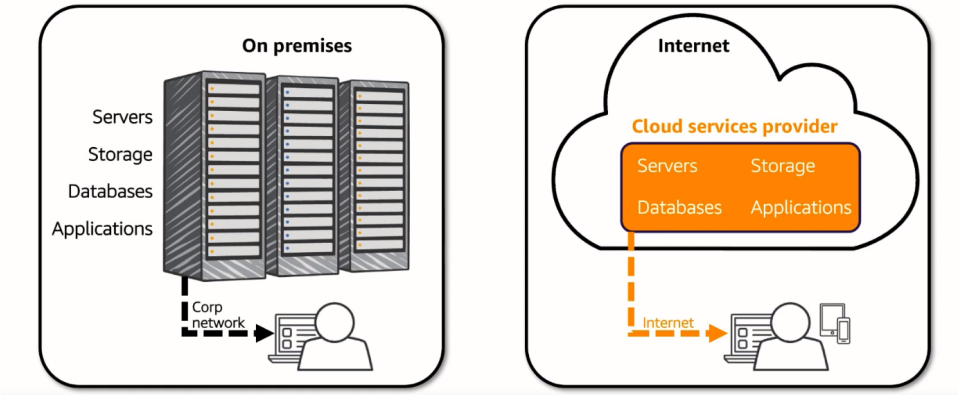


Figura 2.3: Diferenciación Modelo On-Premise - Modelo Cloud

2.2.2. Beneficios de usar un modelo Cloud

Debido a que los clientes generalmente no poseen la infraestructura que se utiliza en los entornos de computación en la nube, pueden renunciar al gasto de capital y consumir recursos como un servicio simplemente pagando por lo que usan. Muchas ofertas de computación en la nube han adoptado el modelo de computación y facturación de servicios públicos pagando por lo que usan, mientras que otras facturan por suscripción.

Al compartir la potencia informática entre múltiples usuarios, las tasas de utilización generalmente mejoran en gran medida, porque los servidores de computación en la nube no están inactivos por falta de uso. Este factor por sí solo puede reducir significativamente los costos de infraestructura y acelerar la velocidad del desarrollo de aplicaciones.

Debido a que los usuarios no están vinculados a un dispositivo específico (solo necesitan la capacidad de acceder a Internet) y a que Internet permite la independencia de la ubicación, el uso de la nube permite a los clientes acceder a sistemas independientemente de dónde estén ubicado o qué dispositivo eligen usar.

A continuación, se listan algunas de las ventajas principales de pasarse a un modelo Cloud:

- **Ahorro de costos:** Una de las motivaciones clave para que muchos se trasladen a la nube es el ahorro de costos asociados, tanto directos como indirectos. Dependiendo del negocio y de cómo se utilice la nube, un usuario puede ahorrar costos de varias maneras. Algunos ejemplos incluyen la adaptación de uso de procesamiento dependiendo de los picos de demanda de uso, la reducción de los recursos invertidos en la administración de hardware y software, etc.

- **Escalabilidad:** Uno de los beneficios clave de los servicios en la nube es la escalabilidad. La computación en nube ofrece la posibilidad de escalar rápidamente los requisitos de computación y almacenamiento para satisfacer las demandas del negocio. A través de un modelo de pago por uso, se puede controlar mejor los costos, aumentando o disminuyendo la escala para satisfacer la demanda estacional.

- **Competitive Edge:** En una empresa, su equipo de TI tiene una cantidad finita de recursos disponibles. Moverse a la nube permite que su negocio se mueva más rápidamente que sus competidores pudiendo asignar a su equipo de TI en proyectos que generen ingresos, en lugar de gestionar la infraestructura local.

- **Seguridad:** Los proveedores de cloud computing ofrecen a las empresas soluciones de seguridad mejoradas. Moverse a la nube pone los recursos de seguridad en manos del proveedor, liberando al equipo de la empresa para que se concentre en otras áreas.
- **Movilidad:** La computación en nube permite mantener a los empleados de una empresa al tanto, dondequiera que estén en el mundo. Al admitir el acceso móvil, la tecnología cloud ofrece a los empleados de la empresa la posibilidad de acceder de forma segura a los sistemas y datos a través de su dispositivo móvil. Apoya el trabajo a distancia y mantiene a los empleados conectados y productivos, incluso cuando están en movimiento.

2.2.3. Cuota de mercado del Cloud Computing

El mercado del cloud computing es enorme. Los nuevos datos del Synergy Research Group[17], a través de siete segmentos clave del mercado de servicios e infraestructuras en la nube, operadores y proveedores, han informado ingresos superiores a los 150.000 millones de dólares en el primer semestre de 2019. Un crecimiento del 24 % respecto al año anterior. Por muy grande que haya llegado a ser el mercado de la nube, hay un enorme margen de expansión. Especialmente si se tiene en cuenta que Gartner[18] proyectó un gasto en TI a nivel mundial de 3,79 billones de dólares en 2019.

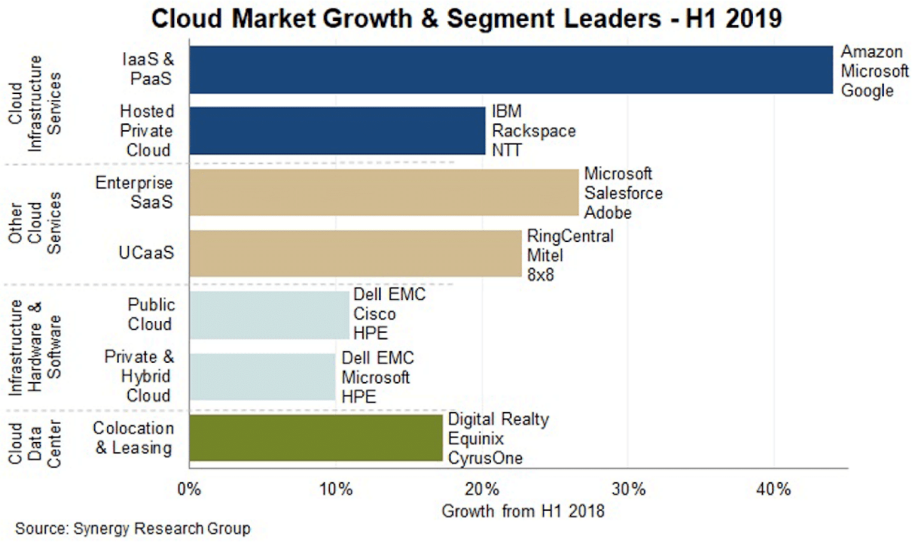


Figura 2.4: Crecimiento del mercado de la nube y líderes del segmento (Fuente: Synergy.com)

Como se muestra en la Figura 2.4, las soluciones de nube pública constituyen la mayoría.

2.3. Proveedores de Cloud Computing

La mayoría de la infraestructura de computación en la nube de hoy en día consiste en servicios altamente confiables y probados en el tiempo, construidos en servidores con diferentes niveles de tecnologías virtualizadas, que se entregan a través de grandes centros de datos que operan bajo acuerdos de nivel de servicio que aseguran 99.99 % o más de tiempo de actividad [19]. Las ofertas comerciales han evolucionado para cumplir con los requisitos de calidad de servicio de los clientes y, por lo general, ofrecen dichos acuerdos de nivel de servicio a sus clientes.

A continuación, se listan algunos lanzamientos de proveedores de Cloud Computing a lo largo de los años:

- Entre los años 2006-2007 Amazon lanzó Amazon Web Services (AWS)
- El software de nube de código abierto conocido como OpenStack fue lanzado conjuntamente por Rackspace Hosting y la NASA en julio de 2010

- IBM anunció el marco IBM SmartCloud para admitir Smarter Planet el 1 de marzo de 2011.
- Oracle anunció Oracle Cloud el 7 de junio de 2012.

Los principales proveedores de la industria están invirtiendo fuertemente en su hardware, software e infraestructura de redes globales para obtener una mayor cuota de mercado, lo que ha dado lugar a un rendimiento sin precedentes. Una competencia sana es siempre una victoria para los consumidores y los socios, ya que esto reduce los costos y les exige innovar constantemente para mantenerse a la vanguardia.

Normalmente, cuando pensamos en los proveedores de cloud computing nos referimos a los tres gigantes de la industria: Azure, Google Cloud y AWS. A continuación, se procede a detallar cada uno de ellos, las cuales entran en las consideradas "cloud públicas", a excepción de Azure que también ofrece "cloud hibrida".

2.3.1. Google Cloud Platform

La plataforma cloud de Google (GCP)[3] está compuesta esencialmente por un montón de servicios y soluciones diferentes que permiten utilizar la misma infraestructura de software y hardware que Google utiliza para sus propios productos, como YouTube y Gmail. Lanzaron su primer servicio, Google App Engine, en una vista previa pública en 2008.

Algunos de sus más de 50 productos incluyen:

- Google Compute Engine
- Google App Engine
- Google Container Engine
- Google Cloud Functions
- Google Cloud Datastore
- Google Storage
- Google Cloud DNS

[3]Página Oficial de Google Cloud: https://cloud.google.com

Uno de los principales productos que ofrece es Google Compute Engine, que junto a otros servicios asociados permite a los usuarios lanzar máquinas virtuales bajo demanda. Google Compute Engine se lanzó en una vista previa pública en junio de 2012[20], y luego se lanzó para su disponibilidad general en diciembre de 2013[21]. Las grandes marcas que ahora utilizan Google Compute Engine incluyen a Apple, HTC, Best Buy, Ubisoft, Philips, Domino's Pizza, Leadpages, Heathrow, PayPal Coca-Cola, Evernote, Sony Music y muchas otras más.

Fortalezas y Debilidades

GCP tiene una sólida oferta en contenedores, ya que Google desarrolló el estándar Kubernetes que ahora ofrecen AWS y Azure. Brinda una experiencia técnica profunda y posee herramientas líderes en la industria para "deep learning e inteligencia artificial", "machine learning" y "data analytics". También ofrece una escala de cómputo considerable y sabe trabajar con balanceadores de carga: Google sabe muy bien cómo trabajar con centros de datos y de disponer de un tiempo de respuesta rápido.

En el lado negativo, Google es un tercero distante en participación de mercado, tal vez porque no ofrece tantos servicios y características diferentes como AWS y Azure. Tampoco tiene tantos centros de datos globales como AWS o Azure, aunque se está expandiendo rápidamente.

Gartner dijo que sus "clientes generalmente eligen GCP como proveedor secundario en lugar de proveedor estratégico, aunque GCP es cada vez más elegido como una alternativa estratégica a AWS por clientes cuyas empresas compiten con Amazon, y que están más centrados en código abierto y por lo tanto están menos alineados con Microsoft Azure".

2.3.2. Amazon Web Services

Amazon Web Services (AWS) es una subsidiaria de Amazon.com[4] que se lanzó para proporcionar servicios de computación en nube a empresas e individuos en 2006. Al igual que la plataforma cloud de Google, tienen una multitud de servicios y soluciones diferentes.

Algunos de sus más de 200 productos incluyen:

- Amazon Elastic Compute Cloud (Amazon EC2)
- Amazon EC2 Container Service

[4]Página Oficial de Amazon: https://www.amazon.com/

- Amazon Lambda
- Amazon S3
- Amazon Route 53

Historia

Amazon ha desempeñado un papel vital en el desarrollo de la computación en la nube. Al modernizar sus centros de datos después del estallido de la burbuja de las .com en 2001, descubrió que la nueva arquitectura de nube que había implementado resultó tener mejoras de eficiencia interna muy significativas. Al proporcionar acceso a sus sistemas a usuarios de terceros sobre una base informática de servicios públicos, a través de Amazon Web Services, introducido en 2002, comenzó una especie de revolución. Amazon Web Services comenzó a implementar su modelo alquilando ciclos informáticos como un servicio fuera del dominio de un usuario determinado, en cualquier lugar del planeta en el que se pudiera ubicar ese dominio. Este enfoque modernizó un estilo de computación mediante el cual las capacidades relacionadas con TI podrían proporcionarse "como un servicio" a los usuarios. Al permitir a sus usuarios acceder a servicios habilitados por la tecnología "en la nube", sin necesidad de conocimiento, experiencia o control sobre cómo funcionaba la infraestructura tecnológica que respalda esos servicios, Amazon cambió radicalmente el enfoque de la informática. Este enfoque llamado Software as a Service (SaaS) transformó la computación en la nube en un paradigma por el cual los datos se almacenan permanentemente en servidores remotos accesibles a través de Internet y se almacenan temporalmente en caché en dispositivos cliente que pueden incluir computadoras de escritorio, tabletas, computadoras portátiles, dispositivos portátiles, teléfonos móviles, etc.

Fortalezas y Debilidades

La mayor fortaleza de Amazon es su dominio del mercado de la nube pública. AWS es un ejemplo perfecto de la verdadera computación en la nube que no solo proporciona excelentes servicios en la nube, sino que también brinda confidencialidad; integridad y disponibilidad de los datos del cliente [15]. AWS no solo proporciona recursos para desarrollar una aplicación, sino que también ayuda a implementar la aplicación a nivel mundial a un costo mínimo. Tradicionalmente, era difícil para una empresa generar rendimiento con usuarios distribuidos y se optaba trabajar en una sola región geográfica a la vez[22].

Los clientes eligen cada vez más AWS para alojar su infraestructura basada en la nube y obtener un mayor rendimiento, seguridad, confiabilidad y escala donde sea que vayan. Por noveno año consecutivo, se evaluó a AWS como líder en el Cuadrante Mágico para infraestructura en la nube[23] como servicio en todo el mundo, con el puntaje más alto en ambos ejes de medición (capacidad de ejecución y visión completa) entre los 6 principales proveedores en la industria, ilustrado en la Figura 2.5.

Figura 2.5: Cuadrante Mágico para infraestructura en la nube. Gartner 2019

Parte de la razón de su popularidad es, sin duda, el alcance masivo de sus operaciones. AWS tiene una enorme y creciente gama de servicios disponibles, así como la red más completa de centros de datos mundiales.

La gran debilidad de Amazon se relaciona con el costo. Si bien AWS baja

regularmente sus precios, a muchas empresas les resulta difícil entender la estructura de costos de la compañía y administrar esos costos de manera efectiva cuando ejecutan un gran volumen de cargas de trabajo en el servicio. Sin embargo, en general, estas desventajas son más que compensadas por las fortalezas de Amazon, y las organizaciones de todos los tamaños continúan usando AWS para una amplia variedad de cargas de trabajo.

2.3.3. Microsoft Azure

Es otro de los grandes proveedores de servicios en la nube. Para las organizaciones que buscan un proveedor de nube, Microsoft se destaca como uno de los principales proveedores en el mercado con una amplia gama de servicios disponibles, particularmente para TI híbrida (la TI híbrida es un modelo de computación empresarial en el que la organización proporciona algunos recursos a través de los sistemas de TI internos tradicionales, al tiempo que aprovecha una combinación de servicios de computación en la nube para otros recursos).

Según un informe de 2020 del Synergy Research Group[24], Azure es el segundo servicio más grande que está creciendo en el mercado y su participación es del casi el 18 % estando por detrás de Amazon que posee una participación del 33 %. Azure es particularmente popular entre aquellas empresas que usan software de Microsoft como Windows y Office.

Al igual que sus competidores, Azure ofrece variedad de servicios que los divide en categorías, entre los que podemos encontrar servicios de computo, redes, almacenamiento, contenedores, Iot, bases de datos, monitoreo, etc.

Historia

Microsoft Azure comenzó a mediados de la década de 2000 como una iniciativa interna llamada “Proyecto Red Dog”. En ese momento, Amazon ya había lanzado su servicio de computación en la nube, y Microsoft se apresuró a ponerse al día.

Durante la Conferencia de Desarrolladores Profesionales de Microsoft 2008, dos años después de que Amazon Web Services (AWS) se pusiera en marcha con su Servicio de almacenamiento simple, Ray Ozzie, el principal arquitecto de software de Microsoft, anunció que la compañía planeaba lanzar su propio servicio de computación en la nube llamado Windows Azure. El plan requería que Microsoft ofreciera cinco categorías clave de servicios en la nube: Windows Azure para computación, almacenamiento y redes; Microsoft SQL Services para bases de datos; Servicios de Microsoft .NET para

desarrolladores; Servicios en vivo para compartir archivos; y las ofertas de SaaS de Microsoft SharePoint Services y Microsoft Dynamics CRM Services.

Después del anuncio, Microsoft comenzó a implementar versiones preliminares de sus servicios en la nube y en febrero de 2010, la Plataforma Windows Azure se hizo comercialmente disponible. Las primeras revisiones del servicio fueron mixtas, y muchos analistas compararon Azure desfavorablemente con AWS. Sin embargo, Microsoft mejoró Azure dramáticamente con el tiempo. Reconociendo que su servicio de computación en la nube había ido mucho más allá de Windows, la compañía renombró Windows Azure como Microsoft Azure en abril de 2014.

En los años posteriores, Microsoft ha seguido ampliando sus capacidades en la nube, ha aumentado su soporte para software de código abierto, y hoy Azure es una opción razonable incluso para empresas que no ejecuten servidores de Windows.

Fortalezas y Debilidades

Microsoft llegó tarde al mercado de la nube, pero se dio un salto inicial al tomar esencialmente su software local: Windows Server, Office, SQL Server, Sharepoint, Dynamics Active Directory, .Net y otros, y reutilizarlo para la nube.

Una gran razón para el éxito de Azure: muchas empresas implementan Windows y otro software de Microsoft. Debido a que Azure está estrechamente integrado con estas otras aplicaciones, las empresas que usan mucho software de Microsoft a menudo encuentran que también tiene sentido que usen Azure. Esto genera lealtad para los clientes existentes de Microsoft. Además, si ya se es un cliente empresarial de Microsoft, existen importantes descuentos en los contratos de servicio.

En el lado negativo, Gartner[5] encuentra fallas en algunas de las imperfecciones de la plataforma. “Si bien Microsoft Azure es una plataforma preparada para la empresa, los clientes de Gartner informan que la experiencia del servicio se siente menos preparada para la empresa de lo que esperaban, dada la larga historia de Microsoft como proveedor empresarial”, dijo. “Los clientes citan problemas con el soporte técnico, la documentación, la capacitación y la amplitud del ecosistema de socios ISV”.

[5]Página Oficial: https://www.gartner.com/en

2.4. Deyel

2.4.1. ¿Qué es Deyel?

Deyel[6] es una Low Code application platform (Lcap/NoCap) que te permite construir tus aplicaciones en un entorno seguro. Su enfoque se encuentra orientado al desarrollo de soluciones de procesos de negocio como también el brindar soluciones ya armadas como se ilustran en la Figura 2.6:

Figura 2.6: Soluciones de Deyel

Deyel actualmente se encuentra funcionando en varios clientes ofreciendo sus servicios para construir aplicaciones, tales como: Banco Nación, Grupo Lauman, ARBA, Wework, Finagil, entre otros.

Deyel como plataforma de desarrollo brinda un gran abanico de características que permiten facilitar el desarrollo de aplicaciones a cualquier usuario. Entre estas características podemos encontrar:

- **Colaboración y Automatización:** La colaboración es la clave para aumentar la productividad y eficiencia en toda organización. Es por eso que Deyel incluye una mensajería social empresarial, realizada a través de chats que permiten que toda la comunicación entre emisor y receptor quede asociada al caso como parte propia de su ejecución y resolución.
- **Modelador de Procesos de Negocio:** Los procesos de negocio generan ventajas competitivas, y el poder diseñarlos libremente, automatizarlos al máximo, e integrarlos al resto del negocio como también a sus actores más relevantes, es la clave para la transformación digital y para el éxito del negocio en sí mismo.
- **Modelador de Formularios:** Todo formulario utilizado en una organización (facturas, comprobantes, gastos, solicitudes, consultas, en-

[6]Página Oficial de Tomcat: https://deyel.com - Junio 2020

cuestas, informes, etc.) puede ser modelado con tan sólo seleccionar, arrastrar, soltar y visualizar.

- **Define con quien interactúas:** Los participantes de un proceso son mucho más que un simple usuario; son empleados, colaboradores, clientes, proveedores, partners; son quienes dan vida al negocio, y de su desempeño depende en parte el éxito del mismo. En la actualidad estos participantes no necesariamente tienen que ser humanos. Muchas tareas ya son automatizadas a través de cosas inteligentes vía IoT y muchas otras son ejecutadas de forma transparente y eficiente por bots. Ante esto Deyel implementa Perfiles que representan una visión del participante –humano o cosa- en tiempo real dentro la empresa.

- **Administración de actividades:** Deyel ofrece el manejo de calendarios personales. Estos calendarios otorgan al usuario la simplicidad y tranquilidad de ver el panorama entero de su empresa permitiendo una fuerte integración de los procesos de negocio con su calendario personal.

2.4.2. Principales componentes

A continuación se procede a explicar en detalle los principales componentes de su arquitectura mencionados en la Figura 2.7.

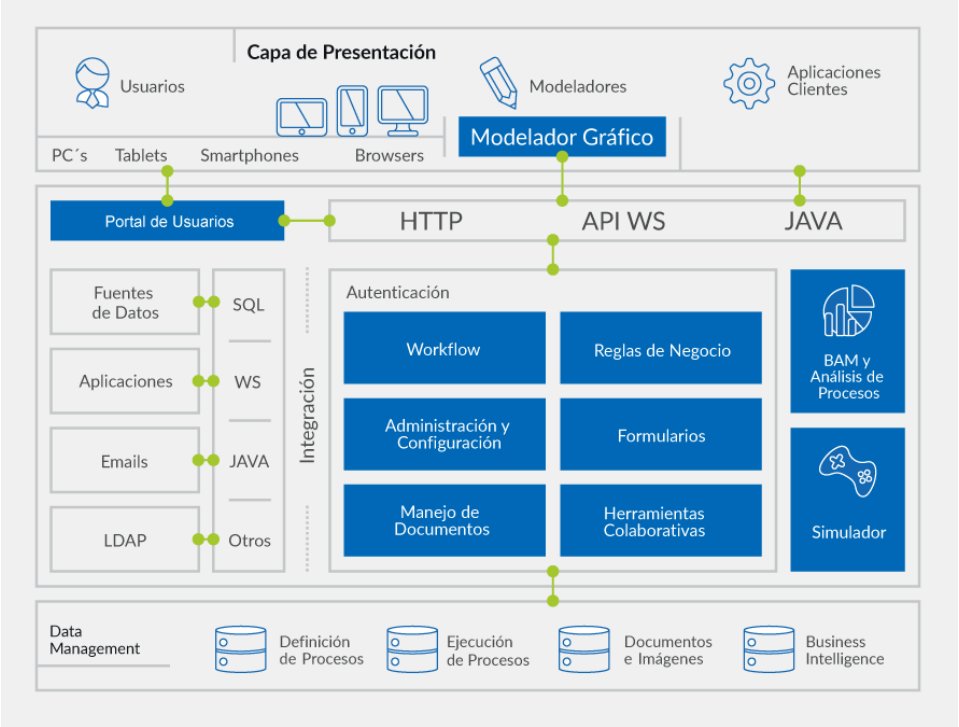


Figura 2.7: Arquitectura de Deyel

Modelador Gráfico

Con el modelador, los usuarios de negocio modelan gráficamente procesos utilizando símbolos de BPMN 2.0 (Business Process Model and Notation) en un ambiente intuitivo y fácil de usar. Pueden definir cada proceso con sus actividades, interconexiones, eventos, tiempos, alertas, etc. y también documentarlo. En este mismo espacio de trabajo se puede interactuar con aspectos de implementación de los procesos como las interfaces de usuario de las actividades, integración con otras aplicaciones, variables y reglas de negocio, etc.

Portal de Usuarios

Con el portal de Deyel, los usuarios pueden interactuar con su lista de actividades y ejecutar sus tareas, iniciar casos de sus procesos, consultarlos, acceder a las herramientas colaborativas, reportes de monitoreo, etc. También a través del portal, se tiene acceso a la funcionalidad de Administración

y Configuración de Deyel.

El portal está basado en estándares tales como HTML5 y CSS3. Se puede utilizar en dispositivos móviles o PCs de escritorio a través de navegadores web. Es configurable en su apariencia a las características propias de cada Organización.

Motor de Workflow

Deyel está sostenido alrededor de esta componente central. El motor de workflow interpreta y ejecuta la definición de los procesos de negocio que son modelados en el modelador gráfico. El modelador, almacena los procesos modelados gráficamente dentro del repositorio de procesos, para que el motor de workflow pueda interpretar estas definiciones y ejecutarlas, dejando el estado de su ejecución en el repositorio de ejecución de casos.

Administrador de Formularios

A través de esta herramienta de Deyel, se generan los formularios que serán la interfaz de usuario de las actividades humanas de los procesos modelados. Los formularios contienen las variables de proceso que son utilizadas por las reglas de negocio y demás componentes que gobiernen el comportamiento de los mismos.

Los formularios disponen de una interfaz rica en componentes (jQuery y Bootstrap), soportan el concepto de RWD (Responsive Web Design) y están basados en estándares tales como HTML5 y CSS3.

Reglas de Negocio

En Deyel a través de reglas de negocio se define el comportamiento de los procesos. Las reglas pueden ser usadas en validaciones, en lógica de negocio específica, control de los flujos de procesos, integración con otras aplicaciones, etc.

Deyel ofrece al usuario de negocios asistentes que le permiten definir de manera intuitiva el comportamiento de la regla, presentándole las estructuras de control, el acceso a parámetros y variables disponibles, reglas de negocio previamente definidas y componentes para ejecutar, los operadores lógicos, etc. Las reglas son almacenadas en un catálogo, en donde se permite administrar su ciclo de vida.

Herramientas Colaborativas

Con el sistema de mensajería de Deyel, los usuarios pueden comunicarse libremente. Los mensajes pueden ser utilizados entre usuarios en forma privada, o pueden ser públicos al estar asociados a casos o a definiciones de procesos. Asociarlos a casos brinda una herramienta de colaboración entre los participantes del caso, haciendo que estos mensajes sean parte del mismo. Por su parte, los mensajes relacionados a la definición de procesos, habilitan un canal de comunicación directo para la mejora de procesos, entre los participantes del caso y los dueños de procesos. También, a través de suscripciones, se puede optar por recibir los mensajes de casos o procesos de interés.

Integración

Deyel provee herramientas y servicios que permiten que cualquier aplicación interactúe en forma bi-direccional con los procesos definidos. Haciendo uso de adaptadores y componentes basados en Web Services, Java, JDBC y otros, permite integrar aplicaciones o fuentes de datos ya existentes.

También, dispone de API Rest y Web Services SOAP que permiten que cualquier aplicación pueda interactuar con los procesos a través de ellas.

BAM y Análisis de Procesos

Deyel, ofrece capacidades de análisis y navegación de estructuras multidimensionales que permiten analizar el funcionamiento de los procesos y sus tareas, tanto en forma histórica (Análisis de Procesos) como en tiempo real (BAM – Business Activity Monitoring).

A través de reportes predefinidos, se puede observar y comprender el funcionamiento de los procesos y detectar posibles mejoras.

Administración y Configuración

Deyel administra su configuración a través de un módulo web, en donde se pueden definir la seguridad y autenticación de los participantes de los procesos con sus distintos perfiles y roles, su relación con repositorios LDAP, los grupos de trabajo, sus delegaciones, calendarios, husos horarios, la apariencia del portal para usuarios y también aspectos técnicos de la solución.[25]

2.4.3. Requerimientos de instalaciones de Deyel

En su modalidad on-premise, Deyel requiere ciertos requisitos de hardware y software para su instalación y funcionamiento en los puestos de trabajo que lo vayan a utilizar. Como toda aplicación JAVA EE debe ser instalado en un servidor de aplicaciones (Tomcat [7], WebSphere [8],etc) y adicionalmente disponer de una base de datos para funcionar.

Las Figuras 2.8, 2.9 y 2.10 describen los requerimientos necesarios para disponer de una instalación de Deyel On-Premise:

HARDWARE	RAM	4 GB mínimo
	Resolución de pantalla	• Usuario final: 1024x768 mínima • Modelador: 1920x1080 mínima
SOFTWARE	Browser	• Chrome 57.0.4 mínima • Firefox 71.0 mínima • Edge 16.16299 mínima (Solo portal de usuario)

Figura 2.8: Requisitos para ejecutar en puestos de trabajos

HARDWARE	RAM	6 GB mínimo
	HD	5 GB tamaño mínimo libre.
	Procesador	64-bit / 2 cores / 2.4GHz mínimo
SOFTWARE	Web server	• WebSphere 8 / WebSphere 9 • Apache Tomcat 8 / Apache Tomcat 9
	Java	JRE 1.8

Figura 2.9: Requisitos para el servidor de aplicaciones

[7] Página Oficial: https://tomcat.apache.org
[8] Página Oficial: https://www.ibm.com/developerworks/ssa/websphere/newto/

HARDWARE	RAM	4 GB mínimo
	HD	10 GB tamaño mínimo libre
	Procesador	64-bit / 2 cores / 2.4GHz mínimo
SOFTWARE	Database	• Mysql 5.7 • DB2 9.5 o DB2 10.5 • Sql Server 2012 • Oracle 12c release 12.2

Figura 2.10: Requisitos para el servidor de base de datos

2.5. Elección del proveedor de Cloud Computing

Luego de dar una introducción teórica al Cloud Computing, describir lo que ofrecen los 3 principales proveedores cloud y explicar que es el software Deyel, explicaremos cual es el proveedor elegido y los motivos que llevaron a elegirlo para realizar las instalaciones cloud del producto Deyel.

2.5.1. Seguridad

La computación en la nube, sin duda, ofrece muchos servicios y tiene innumerables ventajas, pero todavía hay muchos problemas que aún existen y que deben resolverse para aumentar el mercado de una tecnología de clase mundial [13]. La preocupación en la computación en la nube es la SEGURIDAD en torno a los datos, el acceso y protección de la privacidad. AWS (Amazon Web Service) tiene un rendimiento sobresaliente en la computación en la nube debido a su excelente trabajo en el área de Seguridad de datos.

Para AWS, la confidencialidad, la integridad y la disponibilidad (CIA) [15, 26] de los datos del usuario es la tarea más importante.

Parte del trabajo de AWS es:

- Proporcionar seguridad de red.
- Servicio para Disaster Recovery (DR).[27].
- Seguridad a escala: inicio de sesión en AWS.
- Asegurar datos con cifrado.
- Enfoque de respaldo y recuperación.

2.5.2. Requerimientos de Optaris

El objetivo principal de la empresa Optaris es que todos sus clientes dispongan inmediatamente de Deyel sin preocuparse en realizar proyecciones de uso para realizar inversiones en recursos antes de usar el producto.

A finales del año 2017, se decidió invertir tiempo y dinero para evaluar las plataformas Cloud para luego planificar nuestro enfoque y comenzar a migrar a nuestros clientes a la nube.

Como se menciona en este trabajo, evaluamos a los tres proveedores principales de la nube, incluidos Microsoft Azure, GCP (Google Cloud Platform) y AWS (Amazon Web Services). Tras varias pruebas y toma de decisiones en diferentes aspectos (económicos, perfomance, servicios, etc) se tomó la decisión de aterrizar en AWS por su madurez de plataforma, características de respaldo / recuperación, características de automatización, costo / valor, cobertura global y el mejor rendimiento general basado en los requisitos de nuestra plataforma.

El desempeño fue un factor importante en el proceso de evaluación y las instancias de EC2 (Elastic Compute Cloud) en la nueva plataforma Nitro de AWS superaron al resto de las ofrecidas por los otros proveedores. La plataforma Nitro nos proporcionó un rendimiento casi absoluto en procesadores Intel y al mismo tiempo se obtuvo todos los beneficios de las características de EC2 [28]. Otro factor importante fue el respaldo que nos dio gente conocida del sector para elegir la plataforma fundamentando su elección a través de sus experiencias previas en ella.

También hay que considerar que nuestro producto no involucra código licenciado de Microsoft ni tampoco el uso de cloud híbridas, por lo que la apuesta por Azure tampoco fue una opción que nos brindara más beneficios. De la misma manera sucedió con Google, quien tiene su fuerte en inteligencia artificial.

De esta manera, con la elección de AWS, Optaris se beneficia de acceder a una gran cantidad de servicios que sirven para la implementación de las instalaciones y que están respaldadas por la seguridad y confianza del proveedor de servicios cloud más importante de la actualidad.

2.6. Conclusión

A lo largo del capítulo se explicaron los temas más importantes para comprender que es el cloud computing y cuáles son los proveedores más importantes de servicios cloud que existen en la actualidad. Para cada uno de ellos y se evaluaron sus fortalezas y debilidades. Posteriormente se explicó

en que consiste el software Deyel para luego finalizar el capítulo explicando por qué se elige a AWS como proveedor Cloud para realizar las instalaciones de Deyel.

Capítulo 3

Estado del Arte de AWS

En este capítulo se describen conceptos teóricos relacionados al contexto del tema estudiado. Dada la temática de esta tesina de grado, se presentan las definiciones claves de los servicios que nos ofrece AWS con el objetivo de lograr una mejor comprensión de la estrategia desarrollada en el capítulo 4.

3.1. Servicios de AWS

3.1.1. Introducción

Antes de comenzar a explicar los servicios que nos ofrece AWS, debemos saber que AWS dispone de zonas y regiones en el mundo en donde tiene sus centros de servidores para dar soporte a los servicios que ofrece.

Una región en AWS es una ubicación en el mundo en donde son alojados los recursos de informática de AWS. Cada región de AWS es un área geográfica independiente y en cada una de ellas existen varias ubicaciones aisladas conocidas como zonas de disponibilidad[29].

Los servicios que ofrece AWS pueden ser categorizados dependiendo de cómo funcionan acorde a las regiones y zonas disponibles. Existen servicios que se ejecutan con un alcance global en donde la configuración que le explicitemos es la misma para todas las regiones. También hay servicios que su funcionamiento puede ser configurado por regiones y/o incluso también por zonas.

A continuación, se explicarán los servicios más importantes que consideramos que nos ofrece AWS.

3.1.2. Route 53

Amazon Route 53 es un servicio web de DNS escalable y de alta disponibilidad que brinda Amazon[30]. Se configura y funciona como un servicio global siendo independiente de las regiones y zonas donde se utilice. Permite administrar dominios registrados.

3.1.3. VPC

Amazon Virtual Private Cloud (Amazon VPC) es un servicio de AWS que permite administrar una sección de la nube de AWS aislada de forma lógica, en la que puede lanzar recursos de AWS en una red virtual. Dentro de ella se puede controlar todos los aspectos del entorno de redes virtuales, incluida la selección de su propio rango de direcciones IP, la creación de subnets y la configuración de tablas de ruteo y gateways de red.[31]

Descripción General

Dentro de la VPC se administran diferentes recursos: instancias de Amazon EC2, RDS, ALB, etc. Todos ellos se intercomunican mediante tablas de ruteo que son administradas por los routers de AWS y determinan el tráfico en la red.

A continuación, daremos un detalle de los componentes que forman la VPC y permiten la comunicación entre los diferentes servicios que pueden ejecutarse en ella.

Subnet

Una subnet es un rango de direcciones IP en la VPC. Las subnets son utilizadas para la distribución de los recursos.

Route Tables

Las route tables contienen conjuntos de reglas, denominadas rutas, que se usan para determinar a dónde se dirige el tráfico de red. La VPC posee un router implícito el cual usa las tablas de ruteo para controlar hacia dónde se dirige el tráfico de red. Cada subnet de la VPC debe estar asociada a una tabla de ruteo donde la misma controla el direccionamiento de la subnet. La subnet solo puede asociarse a una tabla de ruteo a la vez; sin embargo, puede asociar varias subnetes a la misma tabla de ruteo[32].

Internet Gateway

Un Internet gateway es un componente de la VPC que admite el tráfico IPv4 e IPv6 y permite la comunicación entre las instancias EC2 de la VPC e Internet [33].

NAT Gateway

Una NAT Gateway permite a los componentes de una subnet privada conectarse a Internet (por ejemplo, para actualizaciones de software) o a otros servicios de AWS. La NAT Gateway reenvía el tráfico desde las instancias EC2 de la subnet privada a Internet o a otros servicios de AWS y, a continuación, envía la respuesta de nuevo a las instancias EC2[34]. AWS ofrece dos tipos de dispositivos NAT: una NAT gateway o una instancia NAT. Las NAT Gateway son administradas por AWS y no requieren de configuración adicional, mientras que las intancias NAT son instancias EC2 que utilizan una AMI particular para actuar como tales.

Elastic IP

Las elastic IP son direcciones IPv4 estáticas y públicas, diseñadas para la informática dinámica en la nube. Se puede asociar una dirección IP elástica a cualquier instancia o a cualquier interfaz de red de cualquier VPC. El uso de elastic IP está limitado, todas las cuentas de AWS están limitadas a cinco (5) direcciones IP elásticas por región, porque las direcciones públicas de Internet (IPv4) son un recurso público escaso.

Security Groups

Un security group funciona como un firewall virtual para controlar el tráfico entrante y saliente a un componente dentro de la VPC. Los security group actúan en el ámbito del componente y no en el de la subnet. Por lo tanto, cada componente de la subnet de la VPC puede poseer distintos conjuntos de security group asociados. Un componente en la VPC puede ser una intancia de EC2, una instancia de Aurora, un ALB, etc.

Para cada security group, es necesario añadir reglas que controlan el tráfico entrante a las instancias, así como un conjunto de reglas distinto que controla el tráfico saliente.

ACL de Red

Una lista de control de acceso (ACL) de red es una capa de seguridad opcional para la VPC que actúa como firewall para controlar el tráfico entrante y saliente de una o varias subnetes. Se puede configurar ACL de red con reglas similares a sus grupos de seguridad para añadir una capa de seguridad adicional a la VPC.

3.1.4. ELB

AWS Elastic Load Balancing (ELB) distribuye automáticamente el tráfico entrante de las aplicaciones entre varios destinos, como las instancias EC2. Monitorea el estado de los destinos registrados y dirige el tráfico solamente a los destinos que se encuentran en buen estado. Elastic Load Balancing admite tres tipos de balanceadores de carga: Application Load Balancers, Network Load Balancers y Classic Load Balancers. De estos 3 el que nos interesa es el ALB.

Un **Aplication Load Balancer (ALB)** es un balanceador de carga que funciona en la capa de aplicación, la séptima capa del modelo de interconexión de sistemas abiertos (OSI)[35]. El ALB distribuye el tráfico de aplicaciones entrantes en múltiples destinos, como instancias EC2, en múltiples zonas de disponibilidad[36]. Es el único punto de contacto que tienen a la infraestructura los clientes al acceder desde Internet.

El ALB posee **listeners**. Un listener es un proceso que chequea los request de las conexiones que realizan los usuarios desde internet hacia los dominios que tienen asociados las instalaciones de Deyel. Su funcionamiento es comprobar las solicitudes de conexión a un puerto y un protocolo (por ej. protocolo HTTPS y puerto 443) y a continuación, reenviar dichas solicitudes a uno o más de los destinos registrados y asociados a esos dominios. Cada una de las relaciones entre dominio DNS, protocolo y puerto se asocian a destinos mediante las **reglas** que se definen en los listener.

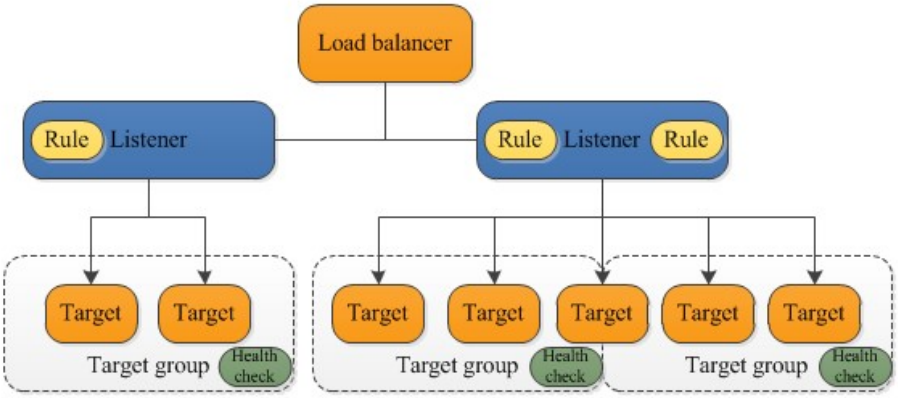


Figura 3.1: Esquema de balanceo de carga del ALB

Target Groups

Son utilizados en los listeners del ALB para direccionar a uno o más destinos registrados.

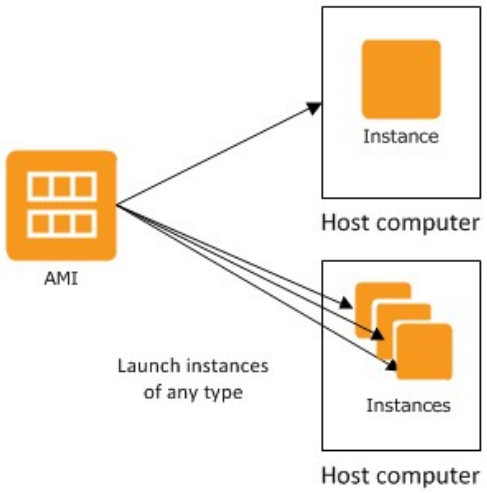
Cada target group posee la siguiente configuración:

- Descripción general (VPC, load balancer asociado, etc). Un target puede estar asociado a un solo load balancer. Hay un target group por servicio.
- Los destinos asociados (en nuestro caso es 1 instancia de EC2)
- Health Checks (configuración de comprobación de estado para determinar que se encuentra en buen estado, es decir, que responde el destino)
- Monitoreo
- Tags

3.1.5. EC2

Amazon Elastic Compute Cloud (Amazon EC2) es un servicio web que proporciona capacidad informática en la nube de AWS. Amazon EC2 trabaja con "imágenes de máquinas de Amazon", más comúnmente llamadas "AMI". Una AMI es una plantilla que contiene una configuración de software (por ejemplo, un sistema operativo, un servidor de aplicaciones y aplicaciones).

A partir de las AMI se inicializan las "instancias", que es una copia de la AMI que se ejecuta como un servidor virtual en la nube[37]. Las instancias pueden ser de diferentes tipos, donde la diferencia entre cada una de ellas radica en el hardware que poseen asociado.



EBS

Amazon Elastic Block Store (EBS) es un servicio de almacenamiento diseñado para su uso con Amazon Elastic Compute Cloud (EC2) tanto en cargas de trabajo intensivas de rendimiento como de transacciones, a cualquier escala[38].

3.1.6. ECS

Amazon Elastic Container Service (Amazon ECS) es un servicio de orquestación de contenedores que se integra de forma nativa con otros servicios como Amazon Route 53, Secrets Manager, AWS Identity and Access Management (IAM) y Amazon CloudWatch[39].

Amazon ECS es compatible con Docker[1] y organiza su ejecución en clústers de instancias EC2.

En Amazon ECS existen varios conceptos y componentes a conocer para comprender su funcionamiento:

[1]Página Oficial de Docker: https://www.docker.com

- **Clúster:**es una agrupación lógica de tareas o servicios que se ejecutan sobre instancias EC2.
- **Servicio:** un servicio representa a un número determinado de instancias de una definición de tarea simultáneamente.
- **Definición de Tarea:** es un archivo de texto, en formato JSON, que describe uno o más contenedores, hasta un máximo de diez, que forman su aplicación.
- **Tarea:** es la instanciación de una definición de tarea dentro de un clúster ECS. Cada tarea puede ser considerara como un contenedor Docker creado a partir de la imagen de la aplicación definida en la definición de tarea del servicio.
- **Agente de contenedores:** es un componente que se ejecuta dentro de cada EC2 que corre en el clúster ECS. Envía información sobre las tareas actuales y la utilización de recursos a Amazon ECS, como también arranca y detiene las tareas cada vez que el servicio Amazon ECS lo solicita.

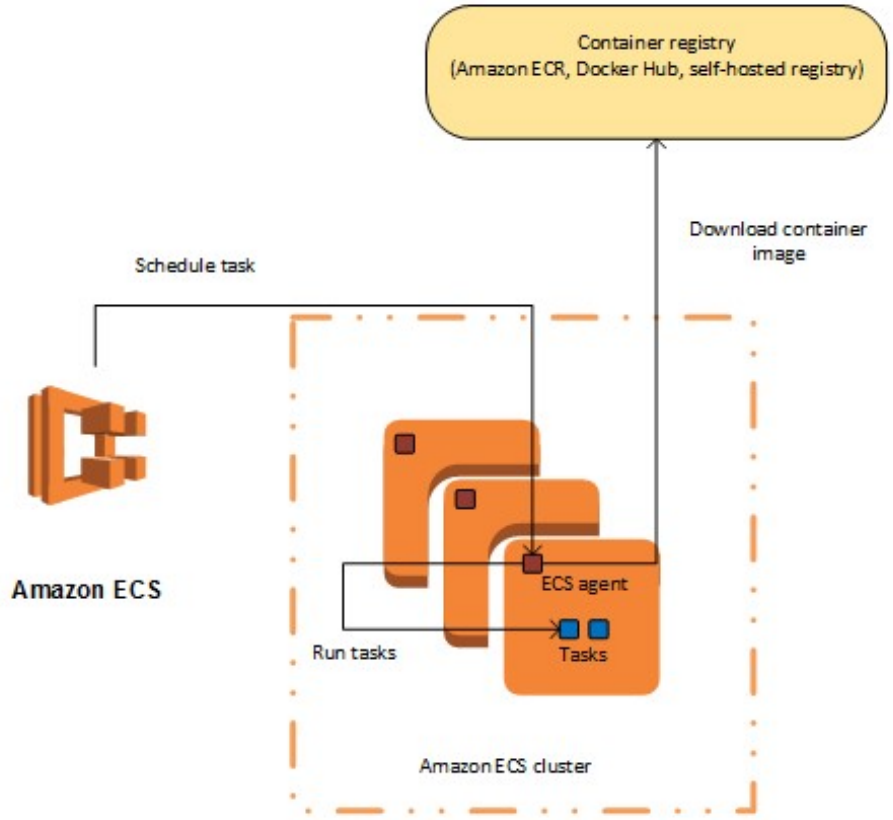


Figura 3.2: Agente de contenedores

Amazon ECR

Amazon Elastic Container Registry (ECR) es un registro de contenedores de Docker completamente administrado que facilita a los desarrolladores las tareas de almacenamiento, administración e implementación de imágenes de contenedores de Docker. Amazon ECR se integra con Amazon Elastic Container Service (ECS), lo que permite simplificar el desarrollo para el flujo de trabajo de producción[40].

3.1.7. RDS

Amazon Relational Database Service (RDS) es un servicio que facilita la configuración, el funcionamiento y el escalado de una base de datos relacional en la nube de AWS. Proporciona capacidad rentable y redimensionable para una base de datos relacional estándar de la industria y gestiona tareas comunes de administración de bases de datos[41].

Amazon RDS está disponible para varios tipos de instancias de base de datos (optimizadas para memoria, rendimiento u operaciones de E/S) y ofrece seis motores de bases de datos conocidos: Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database y SQL Server.

AWS Aurora

Amazon Aurora es una base de datos relacional compatible con MySQL y PostgreSQL[42]. Entre las características que posee se incluye:

- Es tolerante a errores
- Alta disponibilidad y replicación de datos.
- Disponibilidad de tener una conexión para realizar lectura/escritura y una conexión específica para sólo lectura.

Seguridad en Aurora

Amazon Aurora ofrece varios niveles de seguridad para la base de datos. Se incluyen el aislamiento de red por medio de Amazon VPC, el cifrado en reposo mediante claves que puede crear y controlar a través de AWS Key Management Service (KMS) y el cifrado de datos en tránsito con SSL. En una instancia de Amazon Aurora cifrada, los datos del almacenamiento subyacente están cifrados, al igual que las copias de seguridad, las instantáneas y las réplicas automatizadas del mismo clúster.

3.1.8. S3

Amazon Simple Storage Service (Amazon S3) es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos y seguridad[43]. El almacenamiento dentro del servicio S3 se organiza en buckets, similar a la estructura clásica de directorios.

3.1.9. API Gateway

Amazon API Gateway es un servicio que facilita a los desarrolladores la creación, la publicación, el mantenimiento, el monitoreo y la protección de API[44]. Las API actúan como la "puerta de entrada" para que las aplicaciones accedan a los datos, la lógica empresarial o la funcionalidad de sus servicios de backend. Con API Gateway, se pueden crear API RESTful y API WebSocket.

API Gateway gestiona todas las tareas implicadas en la aceptación y el procesamiento llamadas a API simultáneas, entre ellas, la administración del tráfico, compatibilidad con CORS, el control de autorizaciones y acceso, la limitación controlada, el monitoreo y la administración de versiones de API.

3.1.10. Lambda

AWS Lambda es un servicio que permite ejecutar código sin aprovisionar ni administrar servidores[45]. AWS Lambda puede ejecutar código automáticamente en respuesta a varios eventos, como solicitudes HTTP a través de Amazon API Gateway, modificaciones realizadas en objetos en buckets de Amazon S3, actualizaciones de tablas en Amazon DynamoDB, etc.

Al código que se ejecuta en AWS Lambda se lo denomina "función de Lambda". Después de crear una función de Lambda, esta siempre estará lista para ejecutarse en cuanto se active, de manera similar al funcionamiento de una fórmula en una hoja de cálculo. Cada función incluye código y cierta información de configuración asociada, incluidos el nombre de la función y los requisitos en materia de recursos. Las funciones de Lambda "no tienen estado" y no tienen ninguna afinidad con la infraestructura subyacente, por lo que Lambda puede lanzar rápidamente tantas copias de la función como resulten necesarias.

3.2. Esquema de Seguridad

3.2.1. Introducción

La infraestructura implementada en AWS aplica un enfoque integral para proteger y reforzar los datos de nuestros clientes incluyendo medidas físicas, operativas y de software.

AWS cuenta con las garantías de seguridad, control, disponibilidad, integridad de procesamiento, confidencialidad y privacidad detallados en el reporte "System and Organization Controls 3 (SOC 3)" de AWS [46].

AWS cuenta con la certificación de conformidad con las normas:

- ISO/IEC 27001:2013[2]
- 27017:2015 [3]
- 27018:2019 [4]
- ISO/IEC 9001:2015 [5]

Categoría	Casos de uso	Servicio de AWS
Identity & Access Management	Administre de manera segura el acceso a los servicios y los recursos	AWS Identity & Access Management (IAM)
Protección de infraestructuras	Protección frente a ataques DDoS	AWS Shield
	Filtre el tráfico web malintencionado	AWS Web Application Firewall (WAF)
	Administración central de reglas de Firewall	AWS Firewall Manager
Protección de datos	Aprovisionamiento, administración e implementación de certificados públicos y privados SSL/TLS	AWS Certificate Manager
	Alterne, administre y recupere datos confidenciales	AWS Secrets Manager
Conformidad	Portal gratuito autoservicio para el acceso bajo demanda a los informes de conformidad de AWS	AWS Artifact

Figura 3.3: Servicios de Seguridad, Identidad y Conformidad utilizados

Además de los servicios propuestos en la sección 3.1, también se utilizan servicios asociados a los aspectos de seguridad. Entre ellos tenemos a: AWS Shield, AWS Web Application Firewall (WAF), AWS Secrets Manager, AWS Cloudtrial y AWS Identity and Access Management (IAM); todos servicios certificados ISO/IEC.

3.2.2. Servicios de Seguridad

Shield

AWS Shield es un servicio de protección contra ataques de denegación de servicio distribuidos (DDoS). AWS Shield proporciona una mitigación en línea automática y una detección siempre activa que minimizan el tiempo de inactividad y la latencia de la aplicación[47]. Ofrece protección ante los

[2]https://d1.awsstatic.com/certifications/iso_27001_global_certification.pdf
[3]https://d1.awsstatic.com/certifications/iso_27017_certification.pdf
[4]https://d1.awsstatic.com/certifications/iso_27018_certification.pdf
[5]https://d1.awsstatic.com/certifications/iso_9001_certification.pdf

ataques DDoS más comunes, que normalmente ocurren en la capa de red y transporte presentes en el modelo OSI[35].

WAF

AWS Web Application Firewall (WAF) es un firewall que protege las aplicaciones web contra ataques web comunes que pueden afectar la disponibilidad, poner en riesgo la seguridad o consumir demasiados recursos. AWS WAF brinda control sobre cómo el tráfico llega a sus aplicaciones permitiéndole crear reglas de seguridad que bloquean los patrones de ataque comunes, como la inyección de SQL o las secuencias de comandos entre sitios, y las reglas que filtran patrones de tráfico específicos[48].

Secrets Manager

AWS Secrets Manager es un servicio de AWS que ayuda a proteger los datos confidenciales necesarios para acceder a aplicaciones, servicios y recursos de TI. El servicio permite alternar, administrar y recuperar fácilmente credenciales de bases de datos, claves de API y otros datos confidenciales durante su ciclo de vida. Los usuarios y las aplicaciones recuperan datos confidenciales con una llamada a las API de Secrets Manager, lo que elimina la necesidad de codificar información confidencial en texto sin formato. Secrets Manager ofrece la alternación de datos confidenciales con integración incorporada para Amazon RDS, Amazon Redshift y Amazon DocumentDB [49].

SNS

Amazon Simple Notification Service (Amazon SNS) es un servicio web que coordina y administra la entrega o el envío de mensajes a los puntos de enlace o clientes suscritos[50]. Permite filtrar y enviar mensajes a un gran número de suscriptores, incluidas las funciones sin servidor, las colas y los sistemas distribuidos.

CloudTrial

AWS CloudTrail es un servicio que registra la actividad realizada en nuestra cuenta de AWS y entrega los archivos de registros en su bucket de Amazon S3. Permite realizar auditorías de gobernanza, de conformidad, operativas y de riesgo en nuestra cuenta de AWS. Se puede registrar, monitorear de manera continua y retener la actividad de la cuenta relacionada

con acciones en toda nuestra infraestructura. Nos proporciona el historial de los eventos de actividad en nuestra cuenta, incluidas las acciones efectuadas a través de la consola de administración de AWS, los SDK de AWS, las herramientas de línea de comandos y otros servicios de AWS. El historial de eventos simplifica el análisis de seguridad, el seguimiento de cambios de recursos y la resolución de problemas. Además, lo podemos usar para detectar actividad inusual. Estas funciones ayudan a simplificar los análisis operativos y la solución de problemas[51]. AWS CloudTrail alivia los desafíos comunes que se experimentan en un entorno on premise y además ayuda a facilitarle la demostración del cumplimiento de las políticas o los estándares reglamentarios.

Certificate Manager

AWS Certificate Manager (ACM) es un servicio que permite aprovisionar, administrar e implementar con facilidad certificados de capa de conexión segura/seguridad de la capa de transporte (SSL/TLS) públicos y privados para su uso con servicios de AWS y recursos internos conectados. Los certificados de SSL/TLS se usan para proteger comunicaciones por red y para definir la identidad de sitios web mediante Internet y recursos en redes privadas. AWS Certificate Manager elimina el arduo proceso manual de compra, carga y renovación de los certificados de SSL/TLS[52].

IAM

AWS Identity and Access Management (IAM) permite administrar el acceso a los servicios y recursos de AWS de manera segura, puede crear y administrar usuarios y grupos de AWS, así como utilizar permisos para conceder o negar el acceso de estos a los recursos[53].

IAM: Autenticación con identidades

La autenticación es la manera de iniciar sesión en AWS mediante credenciales de identidad. Para controlar el acceso en AWS, se crean políticas y se asocian a identidades de IAM o recursos de AWS. Una política es un objeto de AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando una entidad principal (usuario raíz, usuario de IAM o rol de IAM) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega.

Un *usuario de IAM* es una entidad de la cuenta de AWS que dispone de permisos específicos para una sola persona o aplicación. Un usuario de IAM

puede tener credenciales a largo plazo, como un nombre de usuario y una contraseña o un conjunto de claves de acceso.

Un *grupo de IAM* es una identidad que especifica un conjunto de usuarios de IAM. Se los utiliza para especificar permisos para varios usuarios a la vez.

Un *rol de IAM* es una entidad de la cuenta de AWS que dispone de permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una determinada persona.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales permanentes a largo plazo y los roles proporcionan credenciales temporales.

3.3. Conclusión

En este capítulo se incluyen los conceptos necesarios para enmarcar lo desarrollado en esta tesina de grado. Se hace especial énfasis en las definiciones de los servicios que nos ofrece AWS para el armado de infraestructura como para la gestión de la seguridad. En el capítulo 4 se abordará en detalle la estrategia planteada para representar la infraestructura que contendrá las instalaciones de Deyel haciendo uso de estos servicios. Se profundizará en cada uno de los servicios y como sería su utilización. Posteriormente en el capítulo 5 explicaremos la implementación de los mismos.

Capítulo 4

Estrategia general

Como ya mencionamos anteriormente, el desafío que concierne nuestra tesina de grado está ligado a la migración de un sistema on-premise a instalaciones cloud explicitando los cambios y desarrollos necesarios para llevarlo a cabo.

Finalizado el análisis de los elementos que componen el problema, contemplando las características del mismo, como así también las limitaciones presentes, se comenzó a diseñar una solución que nos permitiera definir una infraestructura cloud utilizando los servicios de AWS.

Para esto, se efectuó un análisis de los diferentes servicios que brinda la plataforma de AWS y se confeccionó un modelo de infraestructura que sea seguro, confiable y extensivo para albergar más de una instalación de Deyel.

Los aspectos más importantes donde se concentraron los mayores retos, tantos técnicos como conceptuales, fueron:

- Realizar las adaptaciones necesarias para que Deyel sea ejecutado en un ambiente Dockerizado (adaptación del producto)
- Definir la infraestructura a partir de los servicios de AWS a utilizar e integrarlos entre ellos.
- Definir el esquema de seguridad y conectividad que se utilizará en la infraestructura.
- Definir el motor de base de datos, su seguridad y su uso.
- Definir e implementar un mecanismo para generar fácilmente instalaciones para usuarios de manera segura, en poco tiempo y que sea mantenible a lo largo del tiempo.

A continuación, se explicará el modelo de infraestructura y el esquema de seguridad propuesto.

4.1. Modelo de Infraestructura

En esta sección presentamos el modelo propuesto para dar solución a nuestra problemática de definir nuestra infraestructura. Posteriormente, en el capítulo 5 explicaremos su implementación.

4.1.1. Introducción

El modelo de infraestructura que definimos se encuentra constituido por el grupo de servicios de Amazon mencionados en las sección 3.1 y está organizado principalmente en lo que llamamos "Clúster Deyel". Definimos como "Clúster Deyel" al grupo de recursos AWS que definen la infraestructura para albergar a los ambientes de Deyel y sus soluciones. Cada "Clúster Deyel" está representado por una VPC (Virtual Private Cloud Network) y se encuentra definido en una región de AWS. Cada ambiente de Deyel es representado de ahora en más como un "Servicio" dentro de la infraestructura.

A continuación, se presenta un diagrama de la infraestructura a modelar e implementar:

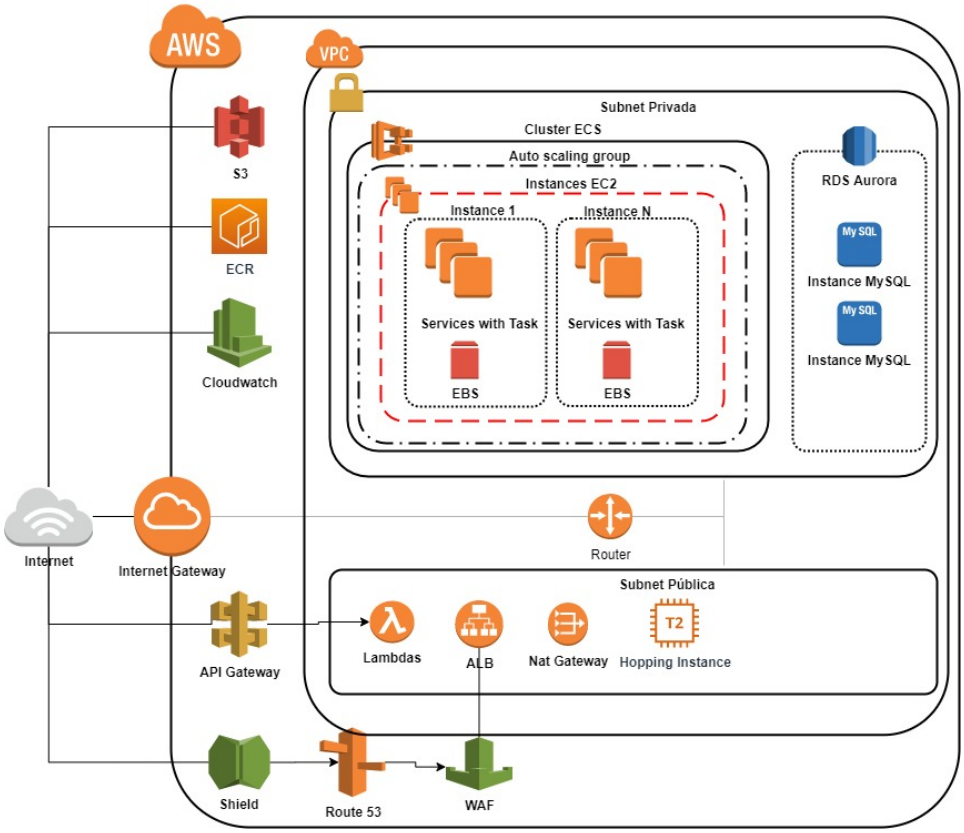


Figura 4.1: Modelo de Infraestructura de Clúster Deyel

Descripción del modelo propuesto

La infraestructura implementada utiliza el servicio ECS (Elastic Container Service) para el procesamiento y el servicio RDS Aurora como repositorio de datos definidos en una VPC accesible desde la web a través de Route 53 utilizando el protocolo HTTPS.

El ECS nos permite ejecutar aplicaciones en contenedores compatibles con Docker (aplicaciones dockerizadas) y organizar su ejecución en clúster de instancias EC2.

Una aplicación dockerizada sobre ECS se corresponde con un servicio que implementa cada ambiente de Deyel; a su vez el servicio para funcionar utiliza una definición de tarea la cual posee la configuración de uso para la imagen docker que dará origen a los contenedores que ejecutará el servicio. Cada imagen docker de Deyel se corresponde con una versión liberada. Un

servicio posee 1 tarea ejecutando en el clúster.

Dichas tareas son stateless, es decir, que pueden reiniciarse en cualquier momento sin causar pérdida de ningún tipo de información ni interrupción en el servicio.

A continuación, se explican en detalle los servicios AWS utilizados para definir el modelo descrito anteriormente.

4.1.2. Servicios a utilizar

Route 53

El servicio Route 53 es utilizado para administrar el dominio de deyel.com y sus respectivos subdominios. Cada servicio que se crea dentro de AWS posee un subdominio perteneciente a deyel.com y a través del Route 53 se determina a qué ALB específico se redirige la solicitud. La creación y eliminación de los subdominios se realiza mediante la ejecución de funciones lambda, dicha funcionalidad será explicada en capítulos posteriores.

VPC

Dentro de la VPC se administran los diferentes recursos que dan forma a nuestro modelo de infraestructura: como las instancias de Amazon EC2, RDS, ALB, etc. Todos ellos se intercomunican mediante tablas de ruteo que son administradas por los routers de AWS y determinan el tráfico en la red.

A continuación, daremos un detalle de los componentes que forman la VPC y permiten la comunicación entre los diferentes servicios utilizados.

Subnet

En nuestro modelo se definen subnets públicas para recursos que deben estar disponibles a Internet, y subnets privadas para recursos que no estarán disponibles a Internet. Los EC2 que ejecutan los servicios de Deyel y la base de datos se encuentran presentes en subnets privadas para prevenir su acceso desde Internet sin autorización.

El modelo propuesto dispone de 6 subnets. 4 subnets privadas, donde los servicios de Deyel que ejecutan dentro del ECS estarán ejecutando en 2 de ellas independientes de las 2 subnets privadas restantes en donde se ejecutará el motor de bases de datos Aurora. Las 2 subnets públicas serán utilizadas para la instancia EC2 a la que llamamos “Hopping Instance” y por los servicios de AWS Lambda, el balanceador de carga ALB y la NAT Gateway.

Route Tables

Dispondremos de 2 route tables que serán utilizadas por la VPC y las subnets que la van a componer.

Internet Gateway

Se utiliza un Internet Gateway para la VPC. Su uso es exclusivamente para acceder desde internet a la instancia EC2 "Hopping Station" (instancia EC2 que permite el acceso a las subnets privadas de la VPC).

NAT Gateway

De las 2 opciones explicadas en la sección 3.1 se utilizará la NAT Gateway: proporcionan una mayor disponibilidad, ancho de banda automático y no requiere de configuración avanzada como las instancias NAT, aunque requieran el uso de una Elastic IP para ser accedidas.

Dispondremos de una NAT Gateway en la VPC para que se conecten las subnets privadas correspondientes y las mismas puedan tener conexión a los servicios utilizados de AWS. Como no se puede asociar un security group a una NAT Gateway, los security groups de las instancias en las subnets privadas son los encargados de controlar el tráfico entrante y saliente de estas instancias.

Elastic IP

Se utiliza una elastic IP para asociar a la NAT Gateway que nos dará acceso a Internet controlado a las subnets privadas. Esto permitirá la integración de las subnets privadas con servicios de AWS externos a ellas.

Security Groups

Se definen tres security groups: uno para el acceso al balanceador de carga ALB, uno para el acceso a las intancias EC2 dentro del clúster ECS y otro para el acceso al motor de base de datos.

ACL de Red

Se dispondrá de una ACL de Red por defecto, pero no estará configurada dejando el trabajo de filtrar la seguridad a los security groups.

ALB

Se define un ALB el cual tiene un listener tanto para HTTP como otro para HTTPS. Las peticiones HTTP son dirigidas a HTTPS para que el listener de HTTPS las redirija al target group correspondiente. Dicha redirección se realiza mediante una regla definida en el ALB.

El alta y baja de las reglas del listener HTTPS es administrado mediante los lambda de creación y eliminación de servicios.

Target Groups

Por cada servicio utilizado en la infraestructura se dispondrá de una Target Group. La creación y eliminación de target groups se realiza mediante los lambda de creación y eliminación de servicios respectivamente.

EBS

El servicio EBS brinda los volúmenes de disco que utilizan las instancias EC2, donde por cada instancia EC2 existen 2 volúmenes asociados: un volumen que representa la AMI de la instancia (de 8GB de tamaño) y un volumen para manejo de datos que como mínimo tiene 22 GB de tamaño.

ECS

En nuestro modelo, cada versión de Deyel está representada por una imagen Docker almacenada en Amazon ECR. Cada usuario posee una instalación de Deyel representada por 1 servicio. Ese servicio posee su propia definición de tarea que hace uso de la correspondiente imagen Docker y ejecuta 1 tarea para dicho servicio. En el clúster ECS definido puede haber muchos servicios de usuarios y cada uno de esos servicios son administrados por el agente de contenedores de cada instancia EC2 que conforma el clúster.

La creación del servicio, la definición de tarea y la puesta en ejecución del mismo se realiza mediante funciones lambdas que serán explicadas en los próximos capítulos.

Aurora

En nuestra infraestructura optamos por utilizar un motor de base de datos MySQL. En vez de elegir un servidor MySQL normal, optamos por utilizar un clúster Aurora gracias a los beneficios que nos brinda, tales como replicación, backups, monitoreo, etc. Dentro del clúster Aurora disponemos

de 2 instancias para garantizar la alta disponibilidad de los servicios. Cada servicio que existe definido en Amazon ECS se asocia a un esquema de base de datos almacenado dentro de Aurora. Los esquemas de las bases de datos se crean por instalación de Deyel al igual que los servicios ECS, todo administrado por los lambdas de creación que especificaremos más en detalle en los próximos capítulos.

S3

En nuestra infraestructura vamos a disponer de un bucket en el cual internamente almacenaremos los siguientes elementos:

- Los archivos .jar que utilizaran los lambdas para la administración de los servicios.
- Los archivos .sql que representan las bases de datos de las versiones de Deyel a publicar como servicios.
- Las descarga de logs de cada servicio, los cuales son manuales y a demanda.

API Gateway

Se define una API que es utilizada como punto de entrada a la infraestructura para poder llevar a cabo la creación y eliminación de servicios. La seguridad en la misma está provista por el uso de API Key y autenticación contra el servicio de AWS IAM.

Lambda

En nuestra infraestructura el servicio AWS Lambda ejecuta funciones implementadas en el lenguaje Java que se utilizan para la creación y eliminación de servicios. Estas funciones lambdas definidas utilizan el acceso a los recursos de AWS (RDS, ECR, etc) mediante la utilización de la VPC y son invocadas desde el servicio de API Gateway. En los siguientes capítulos se profundizará el funcionamiento de las mismas.

4.2. Esquema de Seguridad

4.2.1. Introducción

Como se explicó en la sección 3.2, existen varios servicios de AWS certificados en ISO/IEC[54]. En nuestro modelo de infraestructura utilizamos

los siguientes:

- Amazon Elastic Container Service (ECS)
- Amazon Elastic Container Registry (ECR)
- Amazon Relational Database Service (RDS)
- Amazon Elastic Compute Cloud (EC2)
- Amazon Elastic Block Store (EBS)
- Amazon Route 53
- Amazon Virtual Private Cloud (VPC)
- Amazon Simple Storage Service (S3)
- AWS CloudTrail
- AWS Shield
- AWS Web application Firewall (WAF)
- AWS Identity and Access Management (IAM)
- AWS API Gateway
- AWS Lambda
- AWS Secrets Manager
- AWS Certificate Manager

De la lista anterior, los siguientes servicios se vinculan con los aspectos de seguridad: AWS Shield, AWS Web Application Firewall (WAF), AWS Secrets Manager, AWS Cloudtrial y AWS Identity and Access Management (IAM).

4.2.2. Servicios de Seguridad

Para definir la seguridad en la infraestructura, se utilizan los siguientes servicios:

Shield

Se utiliza AWS Shield en su versión estándar junto a Amazon Route 53 para obtener protección de disponibilidad integral contra todos los ataques DDoS más comunes a la infraestructura.

WAF

Se incorporan sobre el Load Balancer las reglas administradas para AWS WAF, las mismas son un conjunto de reglas pre-configuradas administradas por AWS. Las reglas administradas para WAF abordan problemas como los 10 principales riesgos de seguridad de OWASP (Open Web Application Security Project)[55].

Secrets Manager

Su uso se aplica en la administración de las contraseñas de los usuarios de los esquemas de base de datos de los servicios.

SNS

Es utilizado para almacenar las listas de emails a los que se les notificará un mensaje cuando las alarmas definidas en AWS Cloudwatch se activan.

CloudTrial

Mediante CloudTrial hacemos seguimientos de los eventos que ocurren en la infraestructura para detectar posibles problemas o cambios sobre nuestra cuenta de AWS que afecten a la seguridad, funcionalidad y/o definición de nuestra infraestructura. CloudTrail nos brinda información importante sobre cada acción, incluido quién ha efectuado la solicitud, qué servicios se han utilizado, qué acción se ha realizado, los parámetros de las acciones y los elementos de la respuesta enviada por el servicio de AWS

Certificate Manager

Se utiliza para administrar el dominio a utilizar para exponer las instalaciones a Internet, es decir, deyel.com.

IAM

Meidante IAM se controla el acceso a las API de servicios de AWS y a recursos concretos. Cada usuario definido en el sistema está asociado a un grupo que posee las políticas de seguridad asociadas a partir del rol funcional que cumple cada uno. Por cada grupo disponemos de un usuario asociado a dicho grupo

Los grupos definidos son:

- **Administrador**: grupo de usuarios que administran la infraestructura. Posee asociado la política de acceso administrador de toda la infraestructura.
- **Operador**: grupo de usuarios que consultan los monitores de seguimiento del funcionamiento de la infraestructura. Posee asociado políticas de lectura del servicio Cloudwatch y permisos de consulta a logs de Aurora.
- **Ejecutor API Gateway**: grupo de usuarios que pueden ejecutar externamente el API Gateway. Posee asociado las políticas de invocación para el API Gateway.

Adicionalmente se dispone de un usuario llamado "Deyel" que se utiliza dentro de los contenedores en donde se ejecuta Deyel. Este usuario posee los privilegios para que Deyel pueda escribir en los logs de Cloudwatch y tenga acceso de lectura al servicio Secret Manager para obtener la contraseña del usuario para conectar a la base de datos.

Otro de los componentes definidos en la infraestructura son los roles. Se crean roles para que los servicios que se ejecutan en la infraestructura puedan ejecutar sus operaciones y sólo las que les especificamos. Para ello se crean 2 roles:

- **LambdaRole**: rol utilizado por las funciones lambdas de creación y eliminación. Los permisos asignados son de acceso full a los servicios: Route 53, VPC, ECS, EC2 y Cloudwatch.
- **ECSRole**: rol utilizado por el servicio ECS en el clúster de instancias EC2. Las políticas de permisos asignados corresponden al control total del servicio ECS y de los contenedores en instancias EC2.

4.3. Conclusión

En este capítulo se expuso en detalle el modelo planteado para representar la infraestructura que alojará las instalaciones de Deyel. Se profundizó en cada uno de los servicios y como sería su utilización. Posteriormente en el capítulo 5 explicaremos la implementación de los mismos.

Capítulo 5

Implementación de la Infraestructura

En este capítulo se detallará la implementación del modelo descripto en el capítulo 4, obteniendo la infraestructura para alojar las instalaciones de Deyel.

La implementación de la infraestructura se efectuó de forma manual, donde cada servicio se configuró sin interacción de algún asistente o servicio de despliegue de infraestructura.

También se explicará el proceso de integración de los servicios de AWS dentro del clúster Deyel respetando el esquema de seguridad propuesto en el capítulo 4, sección 3.2.

5.1. Integración de servicios

Los servicios seleccionados y descriptos en el capítulo 4 se integran de diferentes formas en nuestra infraestructura. Algunos de ellos son servicios que poseen una configuración global, otros son servicios que se configuran por zonas de disponibilidad y otros deben ser configurados dependientes de la VPC del clúster de Deyel para poder funcionar. A partir de lo mencionado, se procedió a realizar la siguiente clasificación:

- **Servicios globales:** servicios que se configuran y utilizan indistintamente de la región en la que se defina la infraestructura. Por ejemplo: AWS Shield y AWS Route 53.
- **Servicios por zonas:** dependiendo de la zona en la que definamos la infraestructura, vamos a tener que configurar los servicios para ser

utilizados. Entre los servicios que utilizamos podemos mencionar: AWS WAF, AWS API Gateway, AWS ECR, AWS S3 y AWS Cloudwath.

- **Servicios por zonas con dependencia de VPC:** son los servicios que dependiendo de la zona que elegimos para definir la infraestructura deben ser configurados para utilizarse dentro de las subnets que se definan en la VPC. Ej.AWS Lamdas, AWS VPC.

A continuación, explicaremos cada uno de ellos.

5.1.1. VPC y servicios asociados

El siguiente diagrama muestra más en detalle los componentes claves del modelo descripto en el capítulo 4:

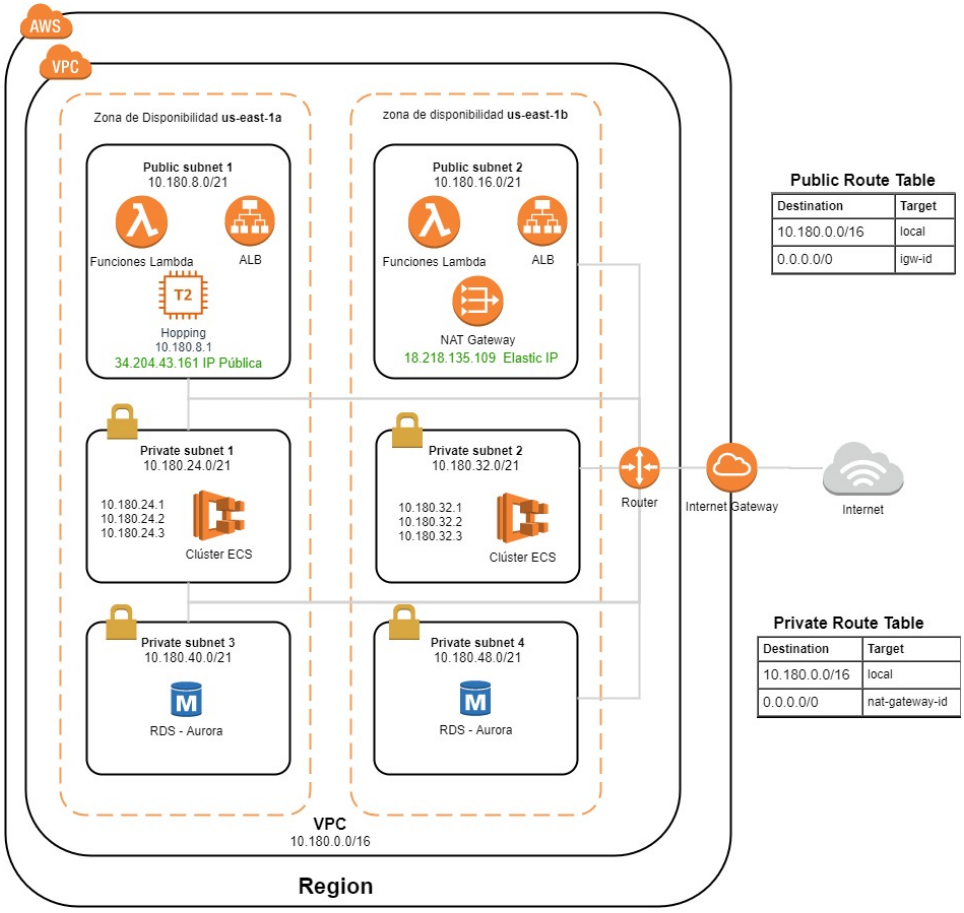


Figura 5.1: Configuración utilizada en la VPC del clúster Deyel

La configuración incluye lo siguiente:

- Una VPC con bloque de CIDR IPv4 de tamaño /16.

  Ejemplo: 10.180.0.0/16.

- Dos subnets públicas con bloque de CIDR IPv4 de tamaño /21 (ejemplo: 10.180.6.0/21 y 10.180.16.0/21). Esto proporciona 2048 direcciones IPv4 privadas por cada subnet. Una subnet pública es una subnet asociada a la tabla de ruteo con ruta al Internet Gateway.

- Cuatro subnets privadas con bloque de CIDR IPv4 de tamaño /21 (ejemplo: 10.180.24.0/21, 10.180.32.0/21, 10.180.40.0/21 y 10.180.48.0/21). Esto proporciona 2048 direcciones IPv4 privadas por cada subnet.

- Una tabla de ruteo asociada a las subnets públicas. Esta tabla de ruteo contiene una entrada que permite que las instancias EC2 de las subnets se comuniquen con otras instancias de la VPC a través de IPv4 y una entrada que permite que se comuniquen directamente con Internet a través de IPv4.

- Una tabla de ruteo asociada a las subnets privadas. La tabla de ruteo contiene una entrada que permite que las instancias EC2 de las subnets se comuniquen con otras instancias de la VPC a través de IPv4 y una entrada que permite que se comuniquen con Internet mediante la NAT Gateway a través de IPv4.

- Un Internet Gateway. Esto conecta la VPC a Internet y a otros servicios de AWS.

- Una NAT Gateway con su propia dirección IPv4 elástica. Las instancias EC2 de las subnets privadas pueden enviar solicitudes a Internet mediante la NAT Gateway a través de IPv4 (por ejemplo, para actualizaciones de software).

- Un balanceador de carga de aplicaciones (ALB) que se configura para ser utilizado en las subnets públicas para acceder a internet y recibir/enviar las peticiones que llegan desde los dominios de los servicios instalados.

- Instancia EC2 "Hopping" de tipo t2.micro[1]en la subnet pública con dirección IPv4 pública que le permiten estar accesible desde Internet para realizar tareas en la infraestructura.

- Clúster ECS formado por 2 instancias EC2 de tipo t2.micro [1] con direcciones IPv4 privadas en el rango de las subnets 1 y 2 (ejemplos: 10.180.24.0/21, 10.180.32.0/21). Esto les permite comunicarse entre sí y con otras instancias en la VPC. Las instancias EC2 de las subnets privadas no tienen direcciones IP públicas y no pueden ser accedidas desde Internet; sin embargo, pueden enviar solicitudes a Internet mediante la NAT Gateway.

- Un clúster Aurora con 2 instancias[2] distribuidas entre las subnets privadas 3 y 4 para otorgar alta disponibilidad.

[1]https://aws.amazon.com/es/ec2/instance-types/
[2]https://docs.aws.amazon.com/es_es/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html

- Las funciones Lambda utilizadas en la administración de las instalaciones de Deyel son asociadas a las subnets públicas para que tengan acceso a los servicios externos a la VPC mediante internet y puedan manipular los servicios internos, tales como los ECS y Aurora al momento de crear los servicios.

En este escenario, todo el tráfico de cada subnet vinculado a servicios externos de AWS pasa a través del Internet Gateway. Los servidores de base de datos de las subnets privada no pueden recibir tráfico de Internet directamente porque no tienen direcciones IP elásticas ni públicas asociadas. Sin embargo, los servidores de base de datos pueden enviar y recibir tráfico de Internet a través del NAT Gateway definido en la subnet pública, pero su acceso está controlado por la seguridad definida en lo security groups correspondientes.

5.2. Seguridad

Para garantizar la mayor seguridad, se siguieron las siguientes prácticas recomendadas por AWS:

- Utilizar varias implementaciones de zonas de disponibilidad para disfrutar de una alta disponibilidad.
- Utilizar grupos de seguridad y ACL de red (opcionalmente).
- Utilizar políticas de IAM para controlar el acceso.
- Utilizar Amazon CloudWatch para supervisar las instalaciones y servicios de la infraestructura.

5.2.1. Zonas de disponibilidad

Como se menciona en la sección 5.1.1 se utilizó para la implementación de la infraestructura una VPC en la región de Ohio (us-east-2) y se definieron las subredes involucradas en las zonas de disponibilidad us-east-1a y us-east-1b para garantizar alta disponibilidad de los servicios.

5.2.2. Security Groups y ACL de Red

AWS nos brinda dos características que utilizamos para aumentar la seguridad de la VPC: los security group y las ACL de red. Los security group controlan el tráfico de entrada y salida de las instancias, mientras que

las ACL de red controlan el tráfico de entrada y salida de las subnets. En la mayoría de los casos, los security group bastan para controlar la seguridad. No obstante, se puede usar también las ACL de red si deseamos agregar un nivel de seguridad adicional en la VPC. La VPC incluye un security group predeterminado. Una instancia de un componente que se lanza en la VPC se asocia automáticamente al security group predeterminado si no especifica ningún security group predeterminado durante el lanzamiento.

En nuestra infraestructura, se utilizarán solo security groups y se dispondrá de una ACL de Red por defecto, pero no estará configurada dejando pasar todo el tráfico entre las subnets. A continuación, definimos los 3 security groups utilizados.

LoadBalancerSG

Se utiliza para controlar el acceso a los balanceadores de carga que recibirán el tráfico de internet y lo dirigirán a las instancias EC2 del clúster ECS en donde se ejecutará Deyel.

La tabla siguiente describe las reglas del security group:

Reglas de Entrada				
Tipo	**Protocolo**	**Rango de Puertos**	**Fuente**	**Descripción**
HTTPS	TCP	443	0.0.0.0/0	Acceso HTTPS al ALB
Reglas de Salida				
Tipo	**Protocolo**	**Rango de Puertos**	**Fuente**	**Descripción**
All traffic	All	All	0.0.0.0/0	Se permite todo el tráfico

ECSSG

Se utiliza para controlar el acceso a las instancias EC2 que forman parte del ECS y a los servicios y tareas que se ejecutan en él.

La tabla siguiente describe las reglas del security group:

Reglas de Entrada				
Tipo	**Protocolo**	**Rango de Puertos**	**Fuente**	**Descripción**
All traffic	All	All	LoadBalancerSG	Se permite todo el tráfico del SG LoadBalancerSG
Reglas de Salida				
Tipo	**Protocolo**	**Rango de Puertos**	**Fuente**	**Descripción**
All traffic	All	All	0.0.0.0/0	Se permite todo el tráfico

DataBaseSG

Se utiliza para controlar el acceso a los servidores de base de datos en la subnet privada. Únicamente pueden acceder a los servidores las instancias que posean el security group ECSSG.

La tabla siguiente describe las reglas del security group:

Reglas de Entrada				
Tipo	**Protocolo**	**Rango de Puertos**	**Fuente**	**Descripción**
MYSQL/Aurora	TCP	3306	ECSSG	Se permite todo el tráfico del SG ECSSG
Reglas de Salida				
Tipo	**Protocolo**	**Rango de Puertos**	**Fuente**	**Descripción**
All traffic	All	All	0.0.0.0/0	Se permite todo el tráfico

Conclusión

Las instancias de componentes asignadas a un security group pueden estar en distintas subnets. Sin embargo, en nuestra infraestructura, cada security group corresponde al tipo de función que desempeña una instancia, y cada función requiere que una instancia de componente esté en una subnet determinada. Por lo tanto, en este escenario, todas las instancias asignadas a un security group estarán en la misma subnet.

5.2.3. Control de Acceso

Para garantizar el control de acceso a la infraestructura se crearon los usuarios correspondientes con las políticas de Iam acordes a su función dentro de la infraestructura como así también se definieron los roles a utilizar por los servicios de AWS. Esta definición de acceso se define con más detalle en la sección 4.2.2.

5.2.4. Monitoreo con Amazon Cloudwatch

Amazon CloudWatch es un servicio de monitorización y observación creado para ingenieros de DevOps, desarrolladores, ingenieros de fiabilidad de sitio (SRE) y administradores de TI. CloudWatch ofrece datos e información procesable para monitorizar aplicaciones, responder a cambios de rendimiento que afectan a todo el sistema, optimizar el uso de recursos y lograr una vista unificada del estado de las operaciones. CloudWatch recopila datos de monitorización y operaciones en formato de registros, métricas y eventos, lo cual ofrece una vista unificada de los recursos, las aplicaciones y los servicios de AWS que se ejecutan en servidores locales y de AWS.

Vamos a explicar algunos conceptos fundamentales en Cloudwatch que utilizaremos en nuestra infraestructura:

- **Métricas**: Las métricas son el concepto fundamental en CloudWatch. Una métrica representa un conjunto de puntos de datos ordenados por tiempo que se publican en CloudWatch. Piense en una métrica como una variable para monitorear, y los puntos de datos representan los valores de esa variable a lo largo del tiempo. Por ejemplo, el uso de CPU de una instancia EC2 particular es una métrica proporcionada por Amazon EC2.

- **Alarmas**: Una alarma observa una sola métrica durante un período de tiempo específico y realiza una o más acciones específicas, en función del valor de la métrica en relación con un umbral a lo largo del tiempo. La acción a tomar al ejecutarse la alarma puede ser una notificación enviada a una lista de emails de Amazon SNS.

- **Dashboard**: es una página de inicio personalizable en la consola de CloudWatch que puede usarse para monitorear los recursos en una sola vista. En él se pueden visualizar widgets para disponer de representaciones gráficas de lo que uno monitorea.

- **Widgets**: son elementos gráficos que pueden usarse en los dashboard para representar métricas.

Uso de Cloudwatch en nuestra infraestructura

En nuestra infraestructura utilizamos CloudWatch para detectar comportamientos anómalos, definir alarmas, comparar registros y métricas y resolver problemas.

Se define para cada clúster Deyel dashboards de servicios monitoreados y dashboards de la instalación para monitorear el estado de la instalación y los servicios. Sobre los servicios se definen alarmas con notificaciones por email.

Todas las alarmas se crean automáticamente, las alarmas de instalación se crean cuando se crea el clúster Deyel y las alarmas sobre los servicios se crean cuando se crea el servicio (y se borran cuando se elimina el servicio).

Las alarmas se definen sobre métricas de Cloudwatch, además de utilizar las métricas que brindan los servicios de AWS se agregan métricas de Optaris que trabajan sobre los logs de Deyel. También se define 1 grupo de logs de Deyel que se envían a Cloudwatch para ser utilizados en las alarmas: Errores de aplicación (ver más en detalle en la sección 6.3).

Las herramientas de monitoreo utilizadas son:

- Métrica de Amazon CloudWatch: Amazon Aurora envía métricas automáticamente a CloudWatch cada minuto para cada instancia y clúster de base de datos y ECS activos. También se envían métricas a Cloudwatch desde Deyel para verificar errores.

- Alarmas de Amazon CloudWatch: se definen alarmas sobre las diferentes métricas de ambientes, ECS o Amazon RDS durante un periodo de tiempo determinado. El envío de notificaciones por email a los equipos definidos en el servicio SNS.

- Amazon CloudWatch Logs: se monitorea los archivos de registros en CloudWatch Logs. También se envían métricas a Cloudwatch Logs desde Deyel para verificar el detalle de los errores.

A partir de estas herramientas se va a monitorear la infraestructura mediante el uso de los dashboard que ofrece AWS Cloudwatch. Utilizaremos 2 dashboard, uno para monitorear toda la infraestructura y otro para monitorear los ambientes instalados.

Dashboard de Instalación

Este dashboard se utiliza para monitorear los recursos de la infraestructura. En él vamos a disponer de los widgets:

- **Uso de CPU del clúster ECS**: se registra el uso total del clúster ECS definido.
- **Uso de memoria del clúster ECS**: se muestra el uso de memoria utilizado por las instalaciones dockerizadas.
- **Tráfico de Red**: se registra el tráfico de red de entrada y salida de las instancias EC2 que pertenecen al clúster ECS definido.
- **Uso de CPU de Aurora**: muestra el uso de CPU del servidor de base de datos. Posee asociada una alarma que nos notifica de uso excesivo si el uso del mismo es mayor al 75 % durante 5 minutos.
- **Memoria disponible de Aurora**: muestra la cantidad de memoria RAM disponible que posee el servidor. Al superar el límite de menos de 600 Mega-bytes de RAM disponible se activa una alarma definida.
- **Cantidad de conexiones usadas en Aurora**: según la instancia seleccionada existe una restricción en las conexiones máximas a las bases de datos. Como ejemplo se configuró una alarma que si se superan las 800 nos notifique.

Los widgets mencionados pueden visualizarse en la Figura 5.2 .

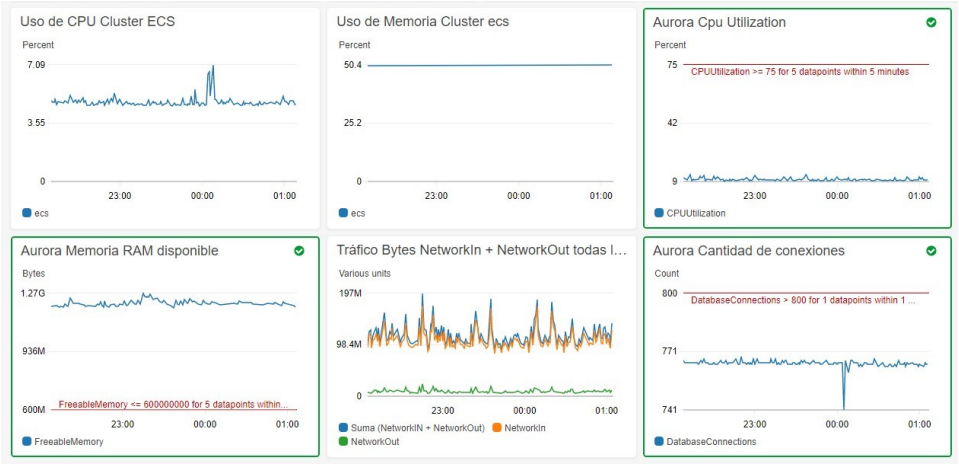


Figura 5.2: Dashboard de instalación.

Dashboard de ambientes instalados

En este dashboard se monitorean cada uno de los ambientes. Los widgets definidos en él son:

- **Uso de CPU por servicio**: Dado un cluster el uso de CPU por servicio es el porcentaje de CPU Units consumido por todas las tareas, respecto del Total de CPU Units disponibles en el Cluster (suma de CPU Units de todas las instancias registradas).[56]

$$\text{Cluster CPU utilization} = \frac{\text{(Total CPU units used by tasks in cluster)} \times 100}{\text{(Total CPU units registered by container instances in cluster)}}$$

Figura 5.3: Cluster Utilization

Como se muestra en la Figura 5.4 cada servicio registra su uso de CPU, en donde se establece como límite de uso crítico el consumo de CPU mayor al 75 % del Cluster.

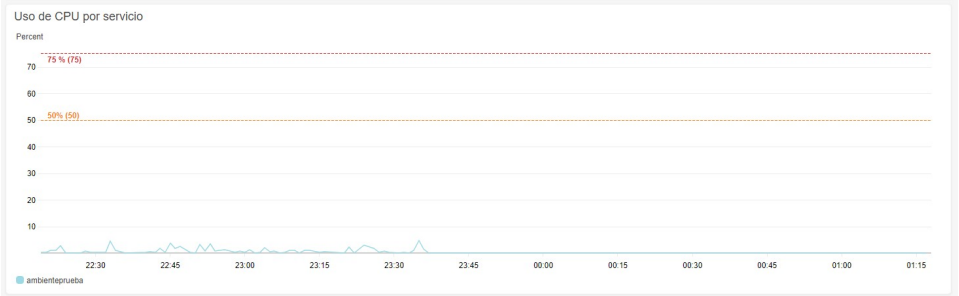


Figura 5.4: Uso de CPU por servicio.

Este límite está controlado por una alarma asociada al servicio. La alarma verifica las métricas de cada servicio para determinar si el mismo posee un uso alto de CPU y en caso de que esto ocurra 5 veces en 5 minutos alerta a los mails registrados en servicio SNS previamente mencionado en la sección 3.2.

- **Errores por servicio**: Cada servicio de Deyel registra los errores que ocurren en el mismo. Asociado a cada servicio al momento de crearse se le asocia una alarma que es alimentado por las métricas que generan

cada error. Esta alarma se activa y notifica si en un lapso de 5 minutos ocurren más de 30 errores. La Figura 5.5 muestra el widget definido en el dashboard.

Figura 5.5: Errores de aplicación por servicio.

- **Tráfico por servicio**: Cada servicio de Deyel posee una alarma que controla la cantidad de requerimientos que le llegan al servicio desde internet, la misma es creada junto al servicio y se activa cuando se supera un máximo de 250 requerimientos por minuto. En la Figura 5.6 podemos ver el widget definido en el dashboard. De la misma manera que las alarmas anteriores, esta envía una notificación a los emails registrados en servicio SNS previamente mencionado en la sección 3.2.

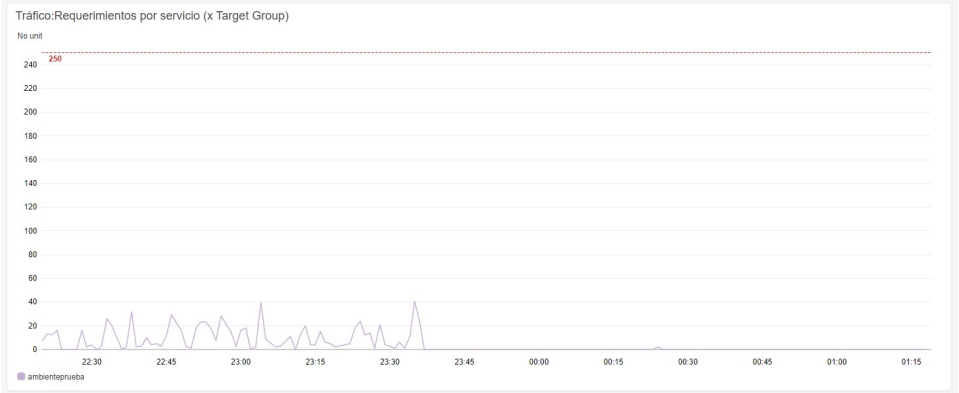


Figura 5.6: Errores de aplicación por servicio.

5.2.5. Backup a nivel infraestructura

En nuestra infraestructura el backup del servidor de base de datos se hace diariamente en forma automática por Aurora generando un snapshot que queda alojado en un bucket S3 administrado por Aurora y con un periodo de retención de 7 días. Adicionalmente y en caso de necesitarlo, se puede realizar un backup manual en un momento determinado seleccionando fecha, hora y minutos deseados. Los snapshots generados pueden restaurarse en una nueva instancia de Aurora y recuperar los datos.

Capítulo 6

Adaptación de Deyel

6.1. Introducción

El objetivo de definir una infraestructura sobre AWS es para poder soportar Deyel PaaS y las soluciones SaaS.

Deyel PaaS se puede definir como la plataforma de baja codificación de Deyel que permite crear y modelar aplicaciones para profesionales IT y usuarios de negocios sobre la infraestructura cloud de AWS. Además de ser una plataforma para modelar aplicaciones, Deyel, incluye soluciones, en donde una solución es una aplicación definida bajo licencia. Por ej: Autoatención de empleados, CRM y Contratos. Ahí es donde surge el concepto de soluciones Saas: una solución SaaS es una aplicación implementada bajo la infraestructura de AWS para disponer a los usuarios de negocio la utilización de una aplicación en un ambiente productivo y funcional.

En este capítulo se especificarán las adaptaciones que tuvieron que realizarse para que el producto Deyel pueda ser ejecutado y sea funcional en instalaciones Cloud en sus 2 plataformas. Como se describió en capítulos anteriores, Deyel es una aplicación desarrollada en Java.

Para facilitar su instalación en ambiente cloud y disponer de varias instalaciones ejecutando bajo la misma infraestructura definida en Amazon se optó por Dockerizar su instalación y usarse bajo Amazon Elastic Container Service [1].

[1] Página Oficial: https://aws.amazon.com/es/ecs/

6.2. Migrado a Docker

Para ejecutar Deyel dentro de un contenedor Docker se especificó un archivo dockerfile[2] que contiene la configuración necesaria para generar la imagen Docker para ejecutar Deyel.

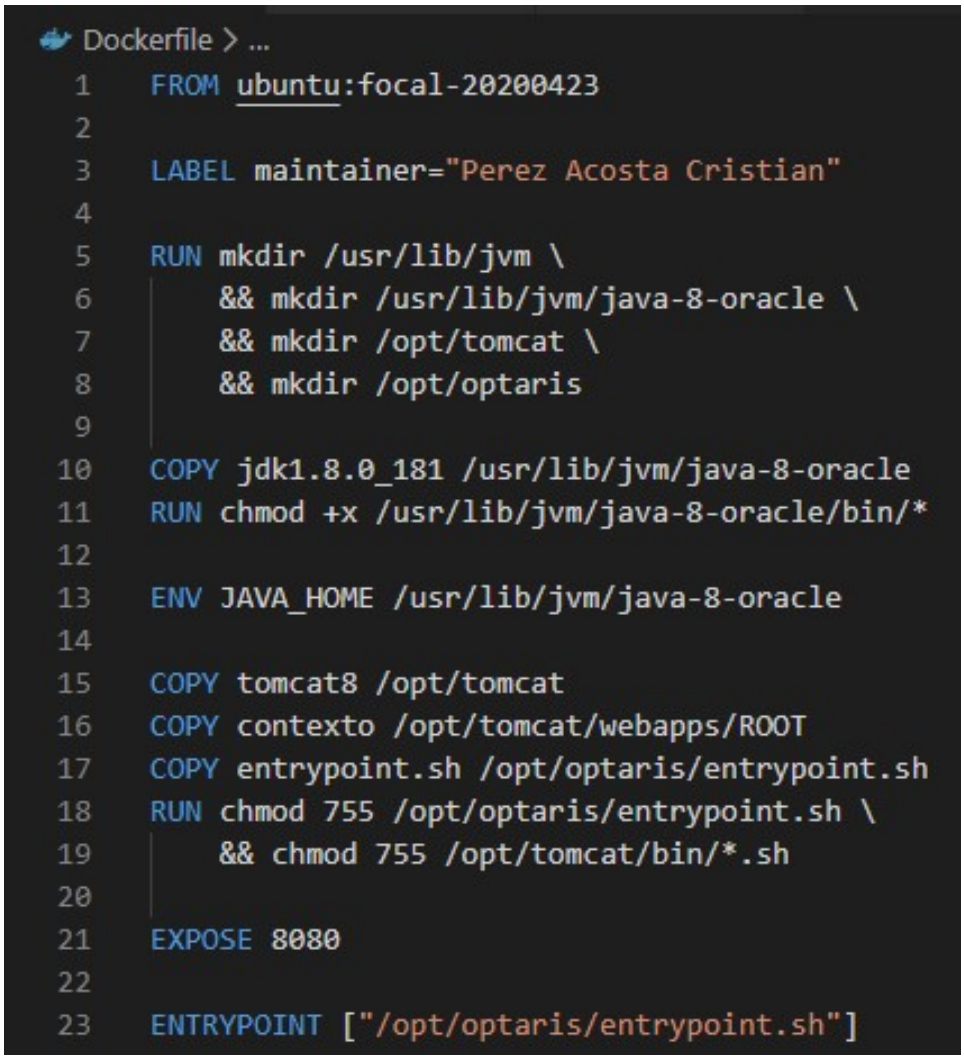


Figura 6.1: Archivo Dockerfile de creación de imagen de Deyel

[2]es un documento de texto que contiene todos los comandos que un usuario podría llamar en la línea de comandos para ensamblar una imagen.

Como muestra la imagen 6.1 se configuró el archivo Dockerfile de la siguiente manera:

1. Creamos la imagen a partir de la imagen de Ubuntu Focal 20.04[3].
2. Creamos 3 directorios que almacenarán:
   - El JDK de JAVA versión 1.8.
   - Apache Tomcat versión 8.
   - Script de inicialización de Tomcat (inicializa Tomcat al iniciar el contenedor).
3. Copia del JDK dentro del directorio especificado anteriormente, asignación de permisos para ejecución de los mismos y declaración del directorio como variable de entorno para usar en Tomcat.
4. Copia de carpeta de instalación de Tomcat a la carpeta especificada.
5. Copia de carpeta contexto a carpeta especificada (la carpeta contexto contiene el war descomprimido de Deyel).
6. Copia de script de inicialización a capeta específica y asignación de permiso.
7. Exposición del puerto 8080 de Tomcat para ser enlazado a un puerto externo para ser utilizado en el contenedor.
8. Especificación de punto de ejecución inicial de Docker al iniciar el contenedor: ejecutar el script entrypoint.sh.

6.3. Adaptaciones de logs

Deyel como cualquier producto de software registra logs internos. Estos logs son utilizados para determinar los problemas que puedan ocurrir en el sistema ante su uso y ayudan a los programadores del producto a resolverlos. En instalaciones on-premises estos logs son almacenados dentro del servidor de aplicaciones, pero en una infraestructura cloud su acceso y monitereo se hace más difícil al disponer de muchas instalaciones. Por lo tanto, es necesario que dichos logs críticos sean registrados en un solo lugar.

Para poder solucionar el inconveniente mencionado, se optó por que cada instalación de Deyel en nuestra infraestructura registre los logs en el servicio

[3]la más actual a la fecha

AWS Cloudwatch. Para eso fue necesario utilizar el SDK que brinda AWS para realizar las adaptaciones correspondientes dentro del producto [4].

Dichas adaptaciones involucraron:

- Creación de un grupo de logs por servicio. Se utiliza el nombre del servicio como nombre del grupo de logs.
- Dentro de cada grupo de logs por servicio se crean por cada día de ejecución un log stream que poseerá internamente los errores del servicio para ese día.
- Por cada error que suceda en el sistema, se carga un valor de métrica para la alarma definida en Cloudwatch que controla la cantidad de errores por servicio.

A partir de estos cambios nos beneficiamos de disponer en la consola de AWS los logs centralizados, visualizar fácilmente los errores y recibir alertas de notificaciones de la alarma "Cantidad de errores por servicios" definida en el lambda de creación de servicio.

[4]SDK Java: https://docs.aws.amazon.com/sdk-for-java/v2/developer-guide/setup-install.html

Capítulo 7

Creación de servicios

En este capítulo se detalla cómo se crea un servicio de Deyel mediante un llamado externo a la infraestructura. Se definirá como es el modelo propuesto para realizar dicha operación y se presentará en detalle la implementación del mismo. Se hará énfasis en cómo se utilizan los servicios de AWS Lambda y AWS APIGateway.

7.1. Introducción

Como mencionamos en capítulos anteriores, la creación y eliminación de instalaciones de Deyel se realiza mediante el uso de los servicios AWS Lambda y AWS APIGateway. La creación de los mismos se realiza ejecutando una función de un API Rest definida en el servicio de AWS API Gateway.

Para las operaciones de creación y de eliminación se definen 2 funciones lambdas que llevarán a cabo dicha funcionalidad e internamente ejecutarán código Java. Estas funciones harán uso del SDK que nos provee AWS para utilizar sus servicios.

Estas funciones lambdas serán invocadas a través de una API REST publicada sobre el API Gateway. Para poder trabajar de manera ordenada el API REST se encuentra configurado para trabajar contra entornos de producción y test de manera aislada. A cada uno de estos entornos en nuestra implementación los representamos mediante “stage”. En el API Gateway un “stage” es una referencia específica de una implementación, lo que equivale a una instantánea de la API. Los “stage” los utilizamos para administrar cada uno de estos entornos donde en cada uno de ellos definimos las “variables de entorno” que actúan como variables que almacenan valores que serán utilizados posteriormente en las invocaciones a los lambdas.

La invocación a los lambdas dentro del API Rest serán discriminados por alias dependiendo del entorno en donde se van a ejecutar. Conceptualmente un alias de Lambda es un puntero a una versión de lambda específica. Una versión de lambda es una publicación de una configuración del lambda identificada con un número y descripción. El uso de punteros nos permite referenciar a diferentes versiones de un mismo lambda sin necesidad de cambiar la configuración en la API, de esta manera podemos utilizar diferentes versiones de lambda en los diferentes entornos e intercambiar las versiones de manera simple directamente desde el servicio de AWS Lambda.

La Figura 7.1 nos muestra un ejemplo de cómo invocando a los diferentes stage definidos en la API Gateway podemos llamar al mismo lambda con su alias e impactar por su diferente implementación en diferentes bases de datos.

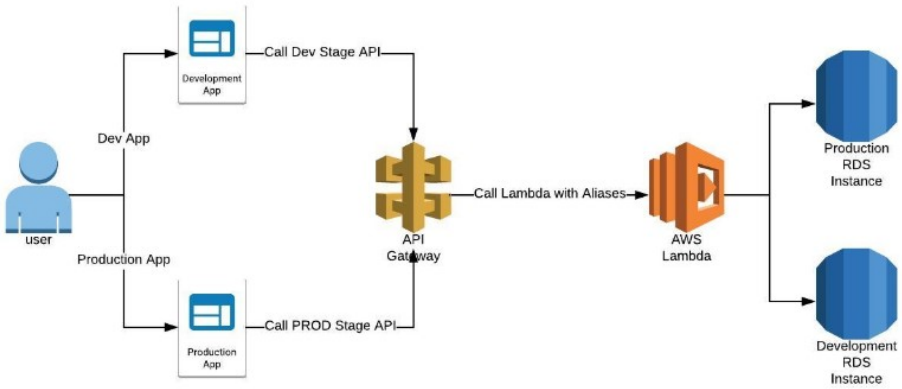


Figura 7.1: Uso de alias en Lambdas

7.2. Definición de API Gateway

El API Gateway a utilizar está representada por un API al que llamamos "apiDeyelAWS". Sobre el mismo disponemos del recurso "services". El recurso "services" tiene varios métodos que se utilizan para ejecutar diferentes operaciones, tal como se muestra en la Figura 7.2.

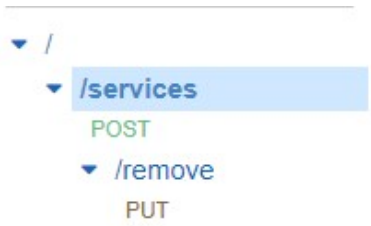


Figura 7.2: Recursos y métodos HTTP del API Rest

Estas operaciones se asocian a los métodos HTTP en donde para el recurso **services** definimos los siguientes métodos:

- POST: (usado para crear un recurso) se utiliza para crear servicios mediante la invocación de la función lambda *create_service*.
- Controlador Remove: (usado para borrar un recurso) se utiliza para eliminar servicios mediante la invocación de la función lambda *delete_service*.

Cada uno de los métodos HTTP pueden ser asociados a diferentes puntos de integración, pero en nuestra implementación solamente utilizamos la integración a funciones lambdas.

7.2.1. Seguridad

La seguridad asociada a la ejecución de cada uno de los métodos del API Rest está establecida por un "plan de uso". Los planes de uso son configuraciones que especifican la forma de uso que tiene un API. En nuestra implementación vamos a hacer uso de un plan el cual lo utilizamos para especificar:

- Que stage del API que está habilitado para su consumo.
- La cuota de cantidad de acceso al API, la cual la limitamos por día (es posible también por semana o mes).
- Las API Keys que puede utilizar el API Rest.

Los recursos disponibles a ser consumidos, además de tener la seguridad de acceso por parte del plan de uso (en donde únicamente mediante el uso del apy key establecido pueden conectarse) también disponen de seguridad mediante autorización de AWS IAM. Por lo tanto, al momento de interactuar contra el API Rest es necesario autenticarse mediante el uso de un usuario en AWS IAM y de disponer del Apy Key correspondiente para el API.

7.2.2. Especificación de métodos HTTP

Cada uno de los métodos HTTP dentro del API Rest necesitan ser configurados para su funcionamiento. El proceso de configuración podemos dividirlo en 4 etapas: method request, integration request, method Response e integration response. Cada una de ellas posee configuraciones particulares para el método y la integración tanto para request del método como para el response. En nuestra implementación los response utilizamos los valores default que nos otorga el API Gateway y para los request se detallará su configuración a continuación.

Method Request

En esta sección se configura el tipo de autorización, el uso de API Key, la configuración de parámetros en la URL, la utilización de headers HTTP y el content type a usar en el cuerpo del request. En nuestra implementación hacemos uso de autorización mediante IAM, uso de API Key y esperamos como body un JSON. La Figura 7.3 ilustra la configuración.

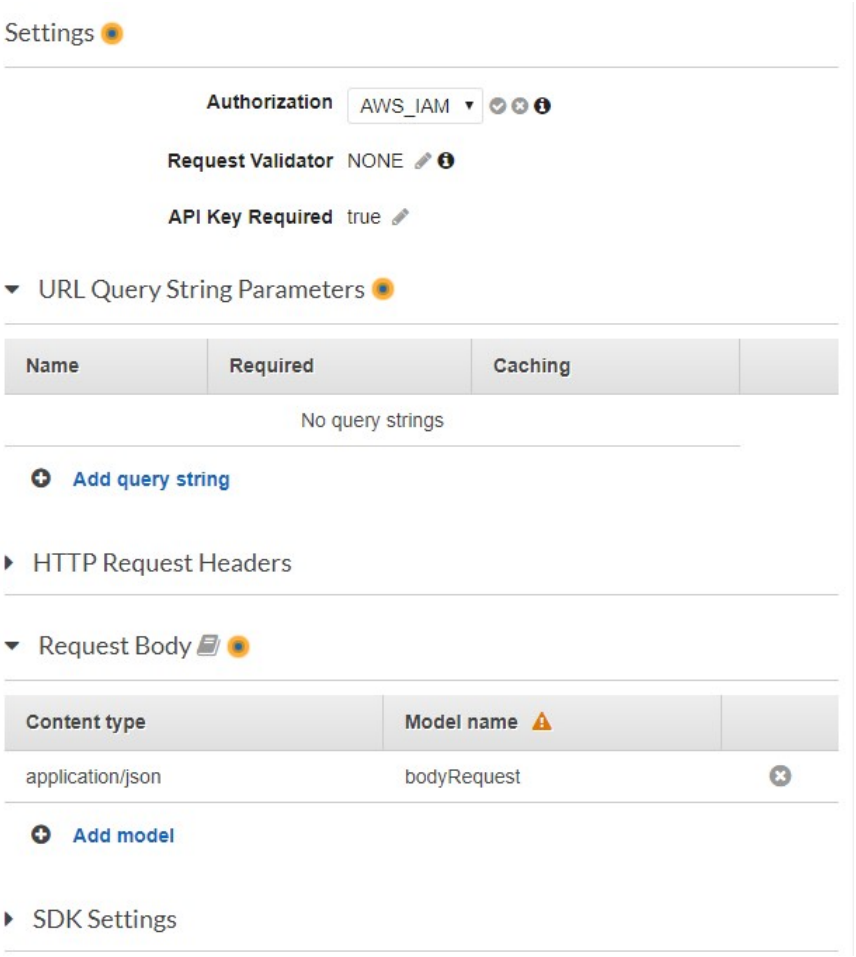


Figura 7.3: Configuración del Method Request

Integration Request

En esta sección se especifica qué parte del back-end será invoca por el método. Se especifica la forma de integración y dependiendo de la misma las opciones son variadas.

Para nuestro caso en donde utilizamos la integración con lambda especificamos algunas de todas las opciones que nos brinda el API Gateway:

- Región en donde se ejecuta el labmda
- Definición de la función lambda de ejecución

- Templates de mapeo para utilizar las variables del stage como valores de entrada en los lambda.

Definición de la función lambda

Al momento de crear un método para un recurso y asociarle su función lambda le especificamos una variable del stage que será utilizada como indicadora de alias para esa función. Dicha asociación requiere posteriormente que al lambda utilizado se le agreguen los permisos para su ejecución por parte del API Gateway [57]. La referencia a un Alias del Lambda se especifica con la convención $<functionLambda>:<alias>$, en donde el alias lo especificamos que se obtiene desde una variable del stage mediante la referencia $\$\{stageVariables.<nombreDeVariable>\}$. En la Figura 7.4 podemos apreciar dicha configuración en donde utilizamos para identificar el alias el valor de la variable *lambdaAlias*.

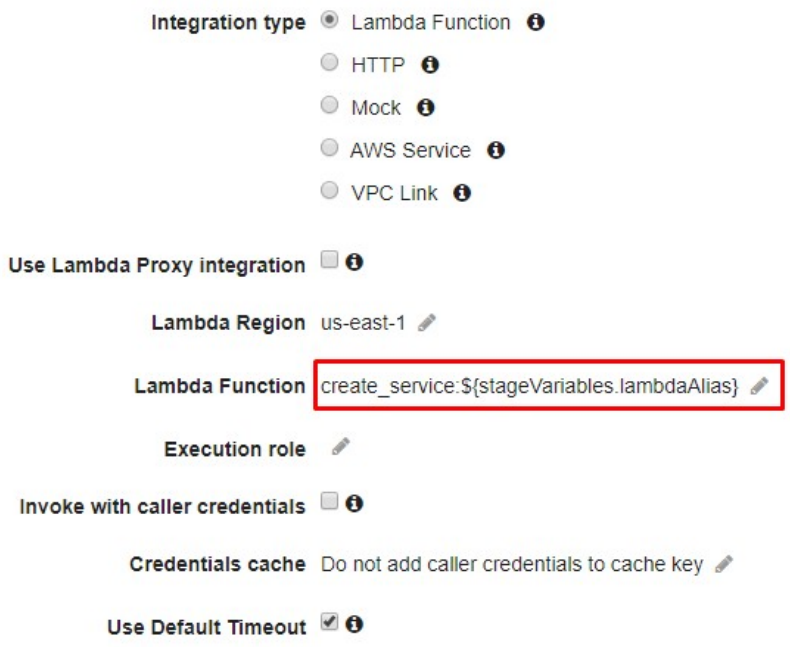


Figura 7.4: Configuración de integración con Lambda

Definición Template de valores de entrada

Una de las opciones más interesantes de esta sección es la utilización de template de valores de entrada. Esta opción que nos brinda el API Gateway

nos permite definir un template de tipo JSON en donde mapeamos las variables especificadas en nuestro stage con claves que posteriormente podemos obtener en las funciones lambdas para configurar nuestras ejecuciones. La Figura 7.5 ilustra dicha asociación y muestra las variables del stage que más adelante explicaremos en más detalle.

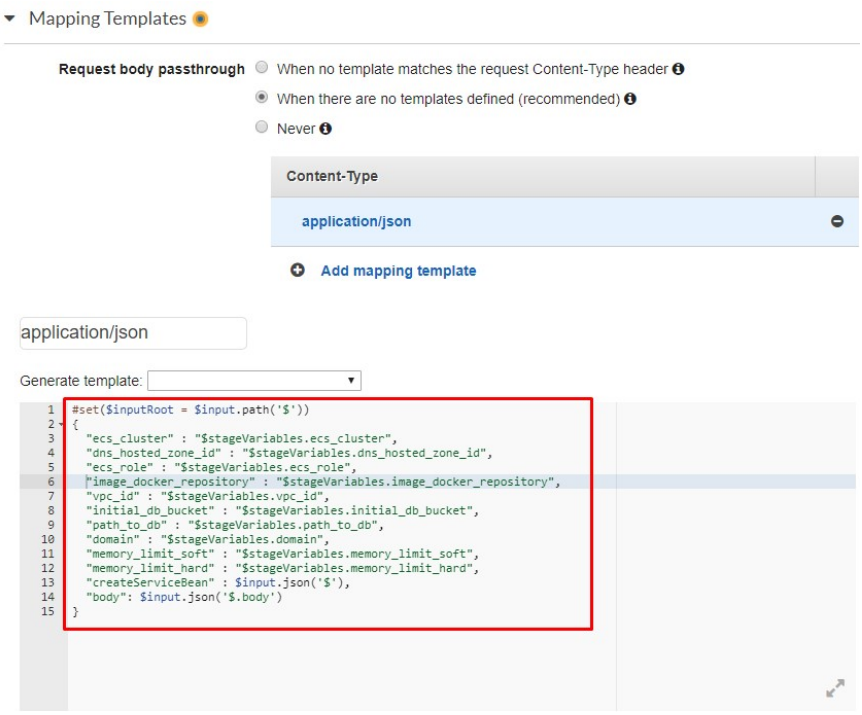


Figura 7.5: Configuración de template de valores de entrada en función Lambda

7.2.3. Especificación de stages

Como se especificó al comienzo del capítulo, en el API Gateway un "stage" es una referencia específica de una implementación. Los stage los utilizamos para trabajar en los diferentes entornos de ejecución en donde se permitirá la creación de servicios. En cada uno de estos entornos las variables a utilizar para especificar las configuraciones del mismo son distintas, es ahí en donde entran en juego los stage.

En nuestra infraestructura para cada uno de los entornos de ejecución necesitamos el uso de variables de entorno que nos brindan diferentes con-

figuraciones. Estas variables dependiendo del entorno pueden tener o no un valor idéntico.

A continuación, mencionamos las variables que definimos para realizar las configuraciones de los diferentes stage en donde se van a crear/eliminar servicios:

- *build_default*: sirve para indicar la versión a instalar de Deyel. También es utilizado como clave para obtener el archivo de base de datos y la imagen Docker para crear el servicio.
- *subdomain*: nombre a utilizar para el servicio y para el subdominio a registrar en el servicio Route 53 para acceder por DNS.
- *domain*: nombre del dominio a utilizar para registrar el servicio en el Route 53.
- *region_aws*: región en donde se va a crear el servicio. Ej. us-east-2 (Ohio).
- *vpc_id*: id de la VPC en la que se va a crear el servicio.
- *dns_hosted_zone_id*: hosted zone id del dominio en el que se registrará el servicio.
- *elb_load_balancer*: nombre del balanceador de carga ALB.
- *ecsCluster*: nombre del clúster ECS.
- *ecs_role*: nombre del rol Iam utilizado para cada servicio que se cree en el ECS.
- *image_docker_repository*: nombre del repositorio ECR en la que se consultará la imagen Docker a utilizar para crear el servicio.
- *path_to_db*: path dentro del bucket del S3 al que se accederá para obtener el archivo .sql de la base de datos a utilizar para crear el schema. Utiliza como raiz el valor de la propiedad *initial_db_bucket*.
- *initial_db_bucket*: nombre del Bucket del servicio S3
- *sql_driver*: clase que representa el driver de la base de datos a la que se conecta. En nuestro caso el valor es: com.mysql.jdbc.Driver para la base de datos MySQL Aurora.
- *db_master_user*: usuario Root de la base de datos.

- *db_master_pass*: contraseña del usuario ROOT de la base de datos.
- *db_host*: dirección IP del motor de base de datos
- *db_port*: puerto habilitado de la base de datos en la que se permite su acceso.
- *db_cluster_identifier*: valor del identificador de la instancia del motor de Aurora en la que se creará el schema de la base de datos.

7.3. Lambdas: creación/eliminación de servicios de servicio

Las operaciones de creación y eliminación de instalaciones de Deyel dentro de la infraestructura son realizadas por operaciones lambdas implementadas en lenguaje Java. Estos lambdas tienen como objetivo funcional utilizar el SDK que nos brinda AWS y representar a nivel de objetos cada uno de los servicios involucrados en las operaciones. De esta manera se procede a explicar cómo está constituida la estructura de objetos que darán origen a las funciones lambdas a implementar: Ver Anexo.

7.4. Descripción de funciones lambda

En esta sección se presentará el modelo propuesto para dar implementación a dicha funcionalidad. Se utilizarán diagramas de clases UML para presentar las distintas clases involucradas y se implementará dicha funcionalidad en el lenguaje Java.

La primera clase que surge es *LambdaService*. Como se puede apreciar en la Figura 7.6 es una clase abstracta que de la cual heredan funcionalidad el resto de las clases definidas.

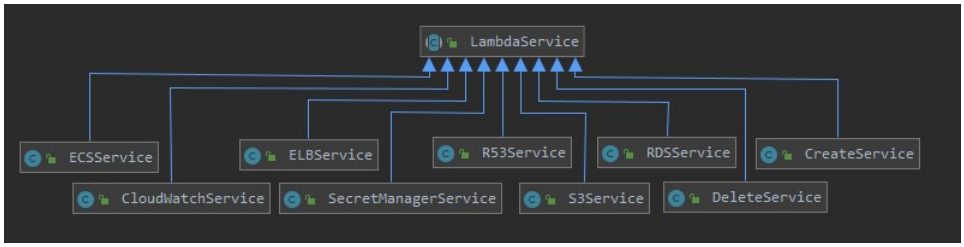


Figura 7.6: Herencia de Clases General

La clase *LambdaService* posee definidas variables que se corresponden con cada uno de los atributos definidos en la Sección 7.2.3. Cada uno de estas variables serán utilizadas por las subclases para funcionar correctamente y realizar sus operaciones. En la Figura 7.7 se puede ver la clase *LambdaService* con sus atributos y métodos comunes para todas las clases que heredan de ella.

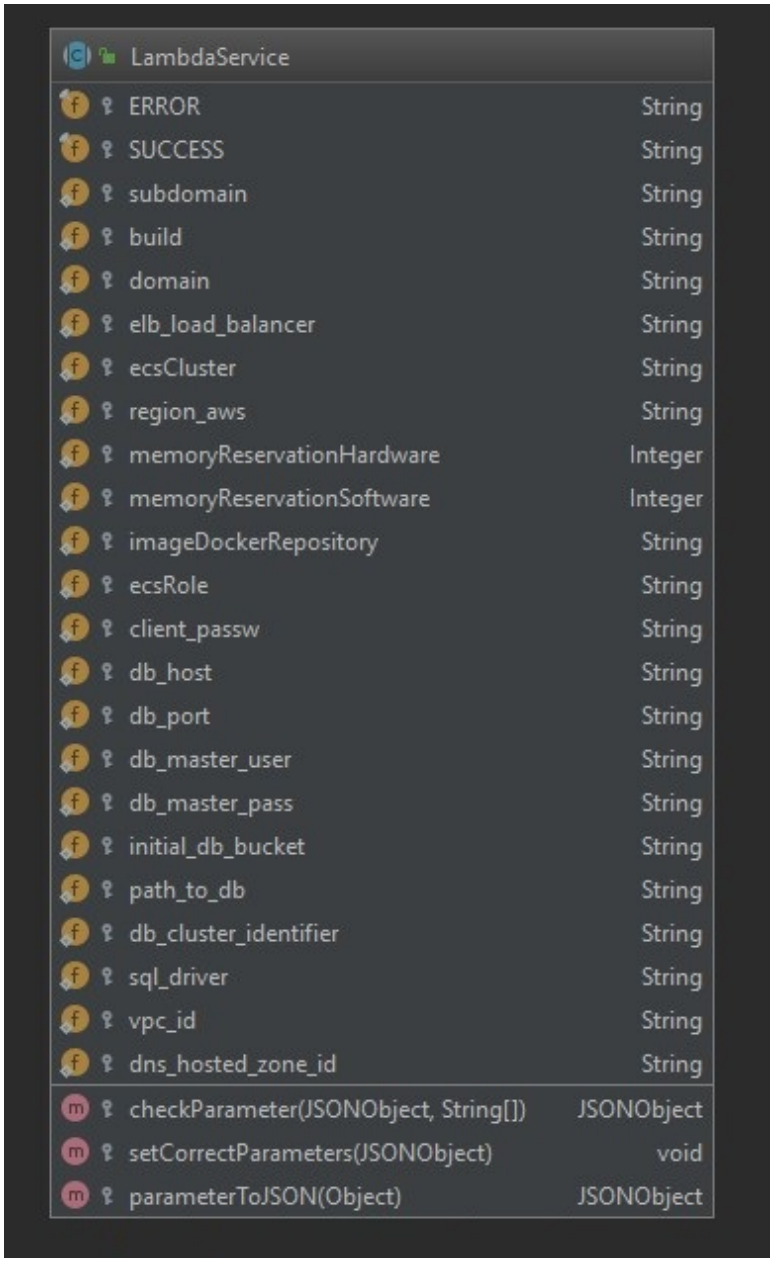


Figura 7.7: Clase *LambdaService*

Veamos con más detalles los métodos definidos para esta clase *LambdaService*:

- parameterToJSON(Object):JSONObject; este método se invoca para filtrar y corregir posible conflicto de caracteres de los datos provenientes como parámetros a los lambdas de creación y eliminación. Recibe como parámetro de entrada un objeto que almacena texto y retorna un objeto JSONObject con la estructura clave-valor para cada uno de los atributos enviados en el parámetro de entrada.

- checkParameter(JSONObject, String[]):JSONObject; este método se invoca para validar que los parámetros que vienen del llamado a la API estén con valores asignados y no sean vacíos ni nulos. Devuelve un objeto JSONObject con el resultado confirmando o no que todos los parámetros ingresados sean válidos.

- setCorrectParameters(JSONObject); este método se invoca para asignar a cada uno de los atributos definidos en la clase con su correspondiente valor proveniente del llamado de la API.

Cada uno de los servicios de AWS está representado por una subclase de la *LambdaService*, a excepción de las clases *CreateService* y *DeleteService* que representan la acción de crear/eliminar servicios y hacen uso de las otras subclases. Cada una de estas clases posee definido internamente una variable llamada *logger* que es utilizada para el registro de logs durante la ejecución de cada una de ellas.

La clase *ECSService* tiene como funcionalidad disponer de la lógica necesaria para trabajar contra el servicio ECS y ECR de AWS. Como se muestra en la Figura 7.8 la clase posee 3 variables definidas en donde se destacan las variables *ecs_client* y *ecr_client*, cada una de ellas representando los objetos que nos otorga el SDK de AWS para comunicarnos con sus respectivos servicios.

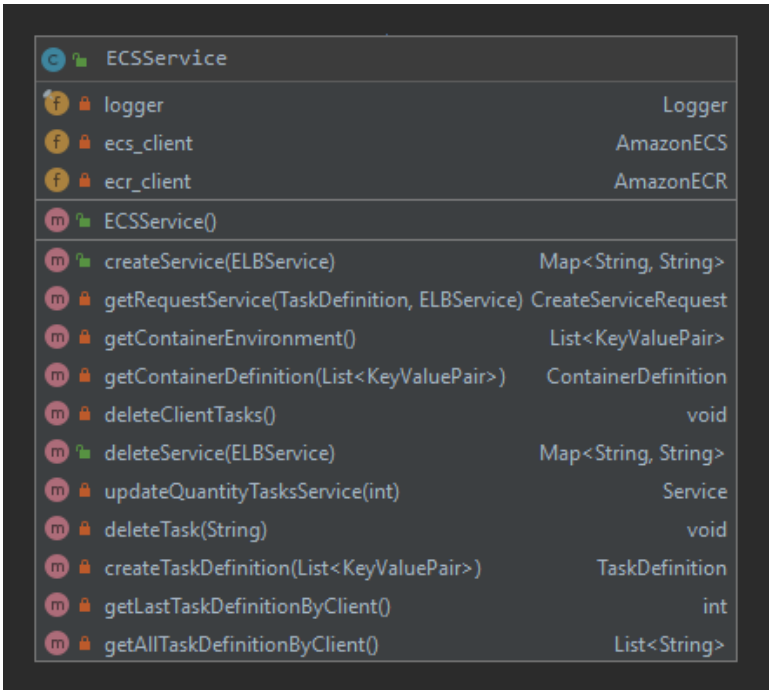


Figura 7.8: Clase *ECSService*

Veamos los métodos definidos más importantes de la clase *ECSService*:

- createService(ELBService):Map; este método es uno de los dos métodos públicos que posee la clase. Recibe como parámetro de entrada una instancia del objeto *ELBService*. Retorna un mapa con clave-valor del resultado de la ejecución, indicando "SUCCESS" o "ERROR" como clave y especificando como valor el error encontrado si el mismo existe. Se encarga de realizar:
  - Creación de la definición de tarea del servicio. Se especifica la configuración del contenedor docker.
  - Creación del target group con posterior creación del listener en el balanceador de carga asociado al target group creado.
  - Creación del servicio dentro del clúster ECS.
- deleteService(ELBService):Map; este método es el segundo método público que posee la clase. Recibe como parámetro de entrada una instancia del objeto *ELBService*. Retorna un mapa con clave-valor del

resultado de la ejecución, indicando “SUCCESS” o “ERROR” como clave y especificando como valor el error encontrado si el mismo existe. Se encarga de realizar:

- La detención de tareas del servicio en caso que el mismo se encuentre ejecutando.
- Eliminación del listener del servicio en el balanceador de carga con posterior eliminación del target group.
- Eliminación de la definición de tarea del servicio con posterior eliminación del servicio del clúster ECS.

La clase *ELBService* posee la lógica para trabajar contra el servicio ELB de AWS, permitiéndonos manipular la creación/eliminación de los target groups y listener que se utilizan para los servicios. Posee su variable correspondiente al cliente que nos otorga el SDK de AWS para comunicarnos con el servicio. El la Figura 7.9 se muestran los métodos que la componen, siendo los más importantes *cleanALB()* y *registerServiceALB()*, ambos utilizados por la clase *ECSService*.

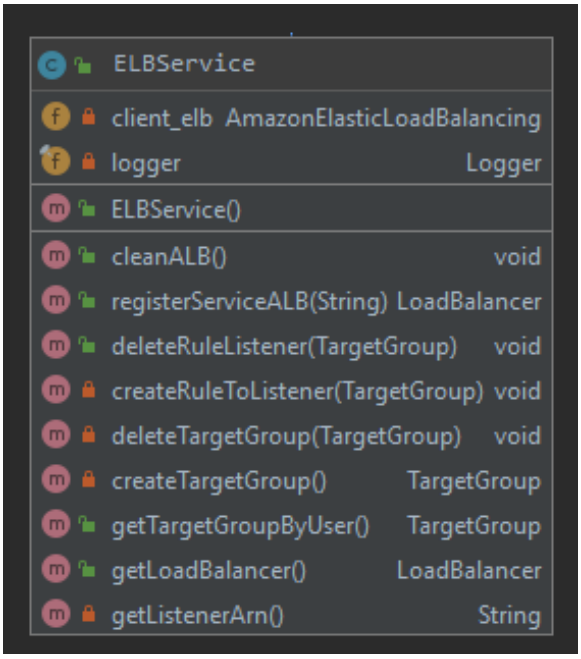


Figura 7.9: Clase *ELBService*

La clase *CloudWatchService* posee la lógica para trabajar con el servicio de CloudWatch de AWS. Posee dos variables asociadas a los clientes que otorga el SDK de AWS para comunicarnos con el servicio, una de ellas *awsLogsClient*, es utilizada para poder trabajar con los logs que se registran en Cloudwatch y la otra *cw_client* para trabajar con la manipulación de alarmas. El la Figura 7.10 se muestran los métodos que la componen.

Veamos los métodos definidos más importantes de la clase *CloudWatch-Service*:

- addAlarms(TargetGroup); este método se utiliza para asociar al servicio las siguientes alarmas de monitorio: "Cantidad de errores por servicios","Uso de CPU" y "Cantidad de Requerimientos por servicio".

- deleteAlarms(); este método elimina las alarmas asociadas al servicio. Borra las 3 alarmas que se crean en el método "addAlarms(TargetGroup)".

- deleteLog():Map; este método se encarga de limpiar los logs registrados por la instalación de Deyel dentro de Cloudwatch. Retorna un mapa con clave-valor del resultado de la ejecución, indicando "SUCCESS" o "ERROR" como clave y especificando como valor el error encontrado si el mismo ocurre.

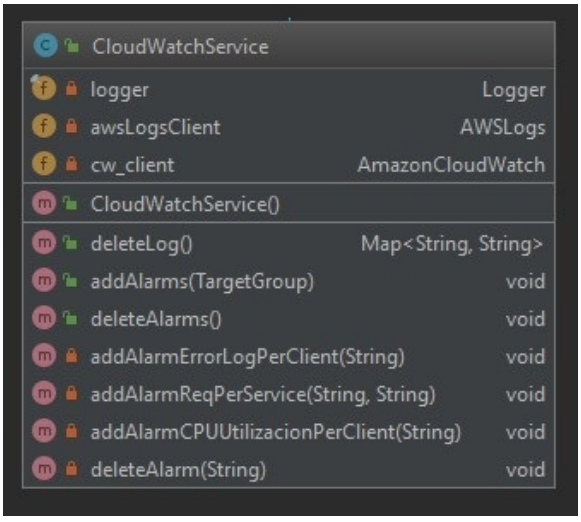


Figura 7.10: Clase *CloudWatchService*

La clase *RDSService* posee la lógica necesaria trabajar con la base de datos Aurora. También hay que mencionar la clase *S3Service* que proporciona el acceso al servicio S3 de AWS para poder obtener el archivo .sql que se utilizará en la creación del esquema de la base de datos que utilizará el servicio que contendrá a Deyel. En la Figura 7.11 se pueden visualizar como están compuestas ambas clases.

Ahora veamos los métodos más importantes de la clase *RDSService*:

- createDatabase():Map; este método es el encargado las operaciones de creación asociadas a la base de datos. Retorna un mapa con clave-valor del resultado de la ejecución, indicando "SUCCESS" o "ERROR" como clave y especificando como valor el error encontrado si el mismo ocurre. La funcionalidad que se desempeña en él es:
  - Crear el esquema de la base de datos con nombre igual al nombre del servicio.
  - Crear el usuario de la base de datos con nombre igual al nombre del servicio y aplicarle los permisos asociados al esquema creado previamente.
  - Creación del Secret Manager asociado al esquema creado previamente para disponer de manejo de claves segura.
  - Carga de datos de la base de datos mediante la ejecución de un restore utilizando como fuente un archivo sql almacenado en el servicio S3 de AWS.
- deleteDatabase():Map; este método es el encargado de las operaciones de eliminación asociadas a la base de datos. Retorna un mapa con clave-valor del resultado de la ejecución, indicando "SUCCESS" o "ERROR" como clave y especificando como valor el error encontrado si el mismo ocurre. La funcionalidad que se desempeña en él es la contraria a la explicada en el método *createDatabase()*: se elimina el esquema de la base de datos, el usuario y el secret manager asociado.

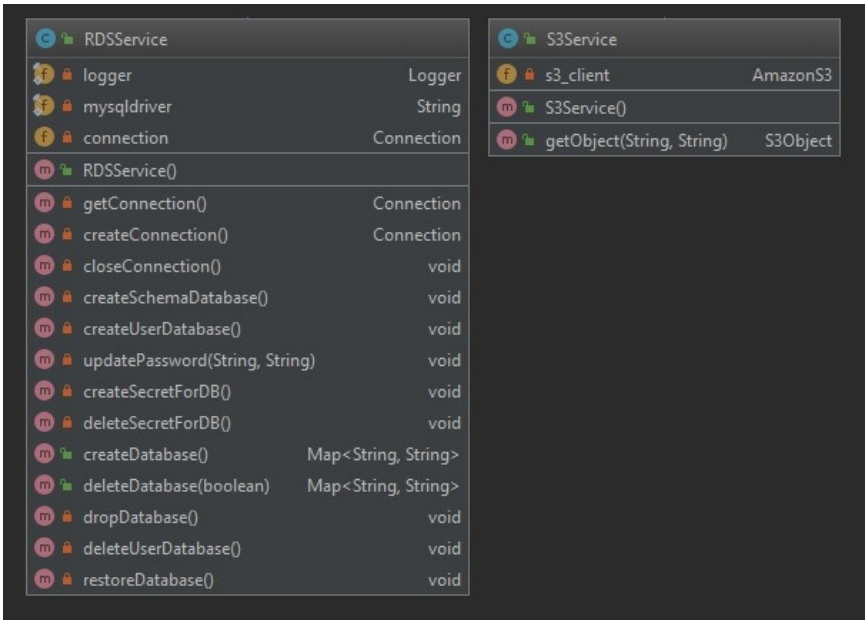


Figura 7.11: Clases *RDSService* y *S3Service*

Como se mencionó en la clase *RDSService*, también existe una clase llamada *SecretManagerService*. Esta clase se utiliza para manipular el servicio de SecretManager de AWS y mediante ella se puede crear, borrar y obtener un secreto para un servicio específico. Internamente en ella se define la clase *AWSSecret* que es utilizada para manipular en forma de objeto la respuesta JSON que proviene del servicio SecretManager de AWS. En la Figura 7.12 podemos apreciar las variables y métodos definidos en ambas clases.

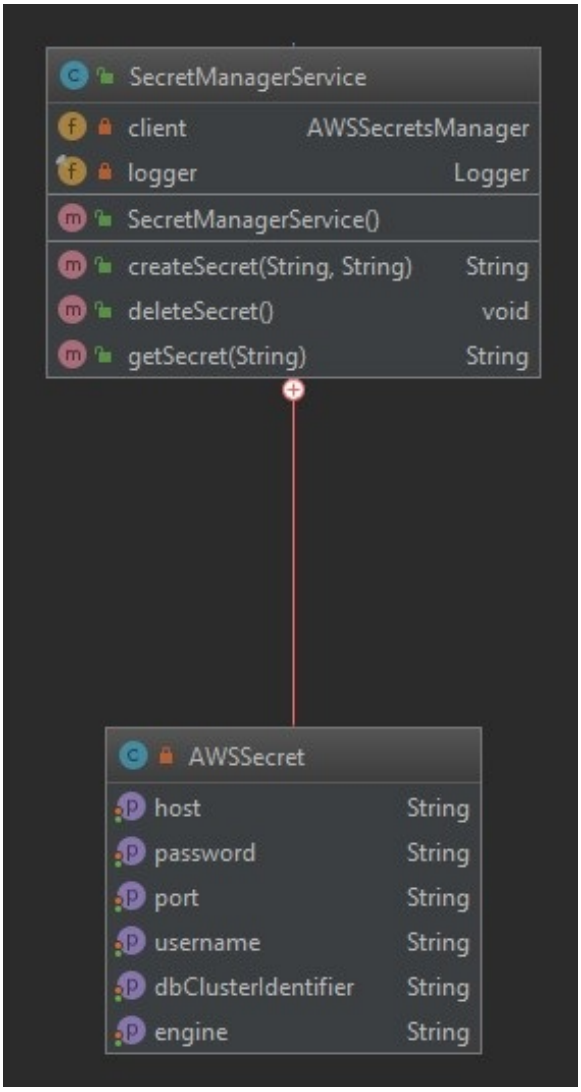


Figura 7.12: Clase *SecretManagerService* y subclase *AWSSecret*

La clase *R53Service* posee la lógica para trabajar con el servicio Route 53 de AWS, permitiendo consultar si existen dominios registrados mediante el método *existDomain()* y crear/eliminar registros de dominios mediante el método *manageDNS()*. Posee su variable correspondiente al cliente que nos otorga el SDK de AWS para comunicarnos con el servicio. El la Figura 7.13 se muestran los métodos que la componen, ambos utilizados por las clases *CreateService()* y *DeleteService()* que serán descriptas a continuación.

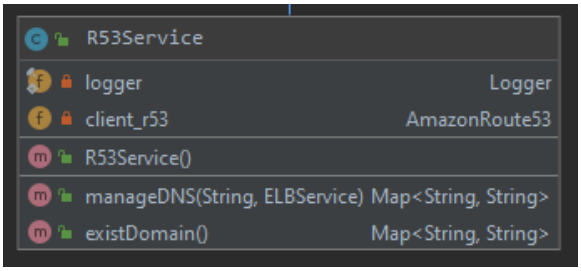


Figura 7.13: Clase *Route 53*

Las clases explicadas anteriormente nos ayudan a manipular cada uno de los servicios que necesitamos para crear/borrar un servicio completamente en AWS. A continuación, explicaremos en qué consisten las operaciones de creación y eliminación de servicios más en detalle haciendo uso de las clases anteriores.

Operación de Creación y eliminación de servicios

Como se mencionó anteriormente, además de las clases previamente descriptas existen las clases *CreateService* y *DeleteService*. Cada una de ellas dispone métodos implementados que nos permiten cumplir la funcionalidad de crear/borrar un servicio completo como se muestra en la Figura 7.14.

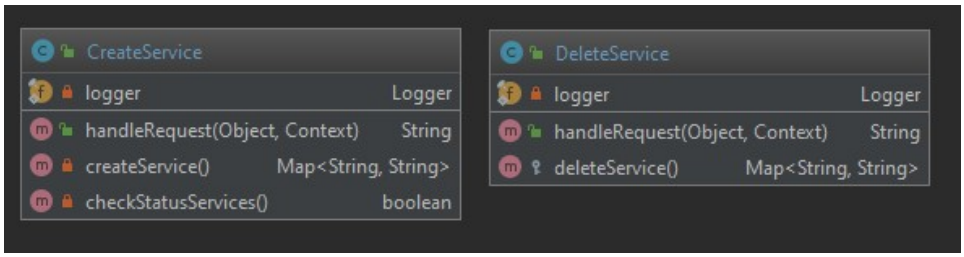


Figura 7.14: Clase *Create Service* y *DeleteService*

Veamos en detalle cada uno de ellos:

- createService():Map; este método es el que posee toda la lógica para la creación de un servicio. Hace uso de las clases que definimos anteriormente para trabajar con cada servicio. Retorna un mapa con clave-valor del resultado de la ejecución, indicando "SUCCESS" o "ERROR" como clave y especificando como valor el error encontrado si el mismo ocurre. La funcionalidad que incluye es:

- Verificación de existencia de servicio consultando al servicio de Route 53 para determinar si se puede crear o no el servicio.
- Creación de la base de datos invocando al método *createDatabase()* de la clase *RDSServices*.
- Creación del servicio dentro del clúster ECS y los pasos que involucra ejecutando el método *CreateService()* de la clase *ECSService*.
- Creación de las alarmas asociadas al servicion mediante la invocación del método *addAlarms()* de la clase *CloudWatchService*.
- Creación del dominio en el Route53 asociado al servicio creado. Se realiza invocando el método *manageDNS()* de la clase *R53Service*.

- deleteService():Map; este método realiza todas las operaciones a la inversa del método *createService()*. Retorna un mapa con clave-valor del resultado de la ejecución, indicando "SUCCESS" o "ERROR" como clave y especificando como valor el error encontrado si el mismo ocurre. La funcionalidad que incluye es el borrado del servicio completo, es decir, borra: el dominio del servicio en el Route53, elimina la base de datos, los logs y alarmas para el ambiente en CloudWatch y el servicio en ECS.

Para comprender mejor el comportamiento de las operaciones de creación y eliminación de servicios mostraremos la interacción de los llamados que forman parte de cada uno de ellos a través de diferentes diagramas de secuencia. Sin embargo, antes de comenzar con los diagramas de secuencia, es útil también comprender la dependencia de las clases dentro de la herencia representada en la Figura 7.6.

Como se ve en la Figura 7.15 podemos apreciar que desde la clase *CreateService* se hace uso de cada una de las clases que utilizamos para trabajar con servicios. Mediante las lineas punteadas podemos identificar que la clase *CreateService* hace uso o crea las otras clases y mediante las líneas azules determinamos que todas las clases heredan de la clase abstracta *LambdaServices*. El mismo comportamiento podemos observarlo para la clase *DeleteService* en la Figura 7.16.

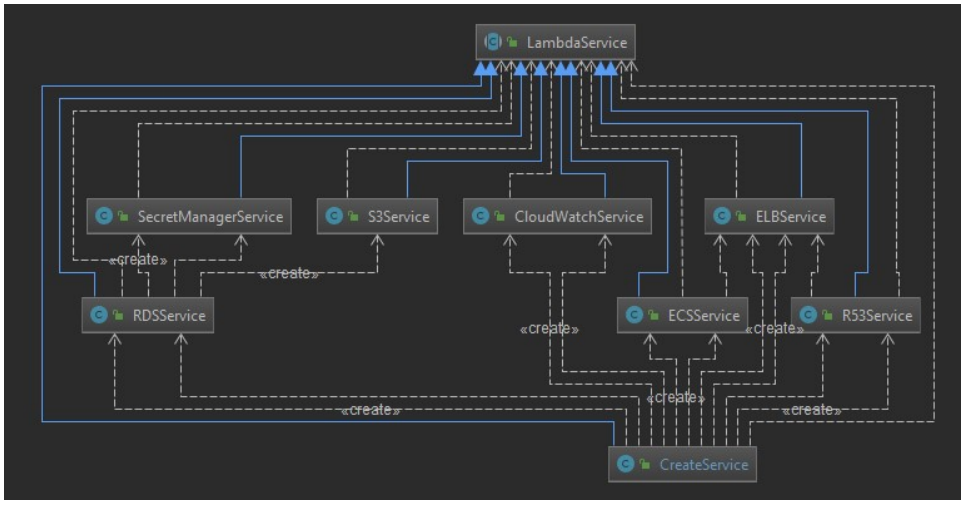


Figura 7.15: Dependencias entre clases asociadas a la creación de un servicio

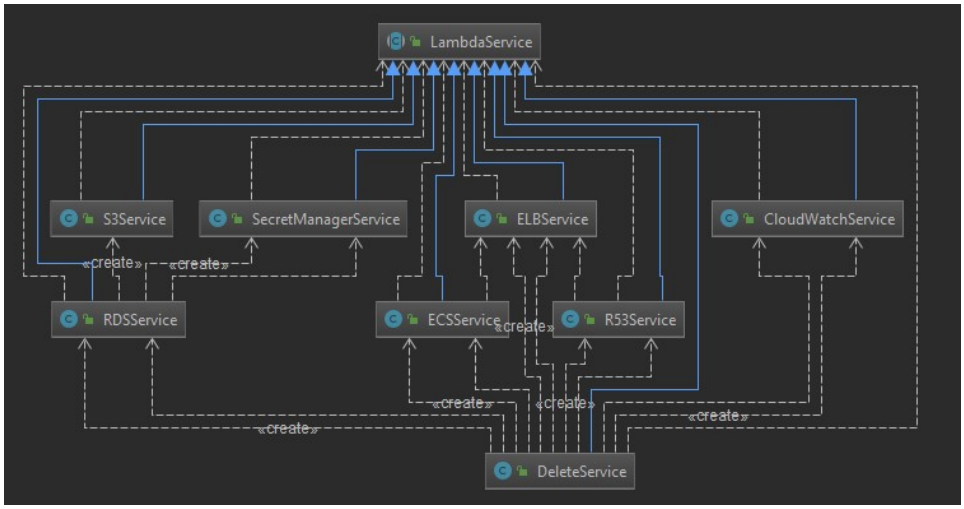


Figura 7.16: Dependencias entre clases asociadas a la eliminación de un servicio

Diagramas de secuencia

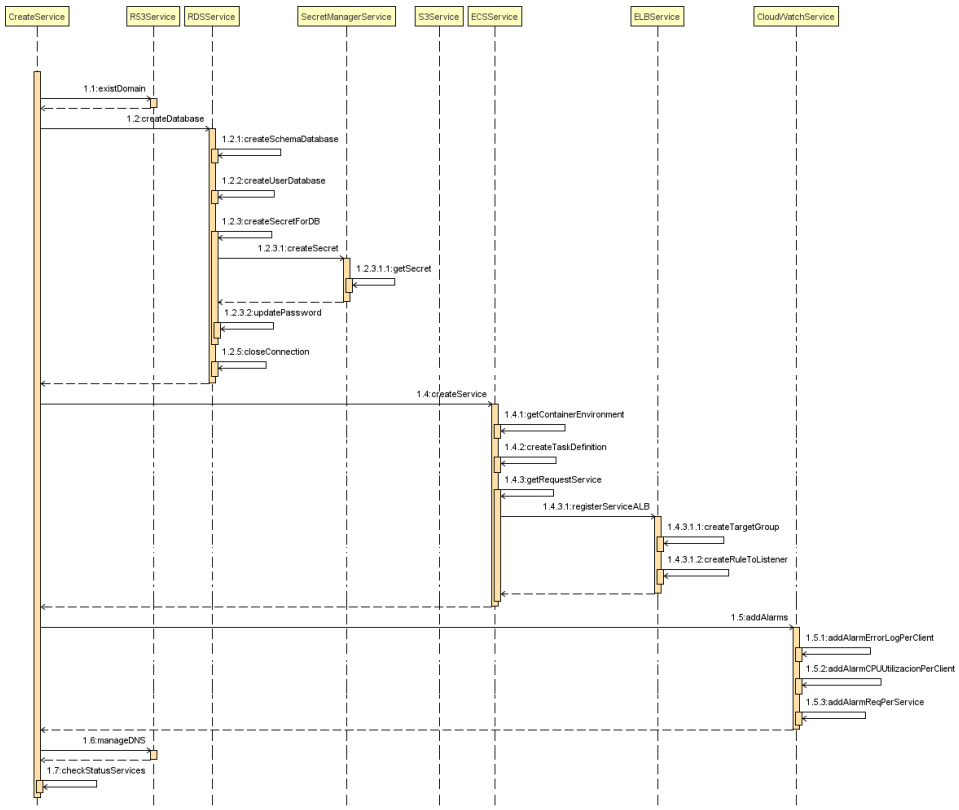


Figura 7.17: Diagrama de secuencia de creación de servicio

En la Figura 7.17 se puede apreciar el diagrama de secuencia a seguir cuando se crea un servicio. Se invoca a la clase *CreateService* lo que desencadena el evento de consulta al *R53Service* para ver si existe el dominio, en caso de no existir se ejecuta el evento de creación de base de datos llamando a la clase *RDSService*. Una vez creada la base, se ejecuta el evento de creación de servicio en la clase *ECSService* para posteriormente registrar las alarmas del servicio con la clase *CloudWatchService*. Finalmente se ejecuta el evento de verificación de creación exitosa en la clase *CreateService*.

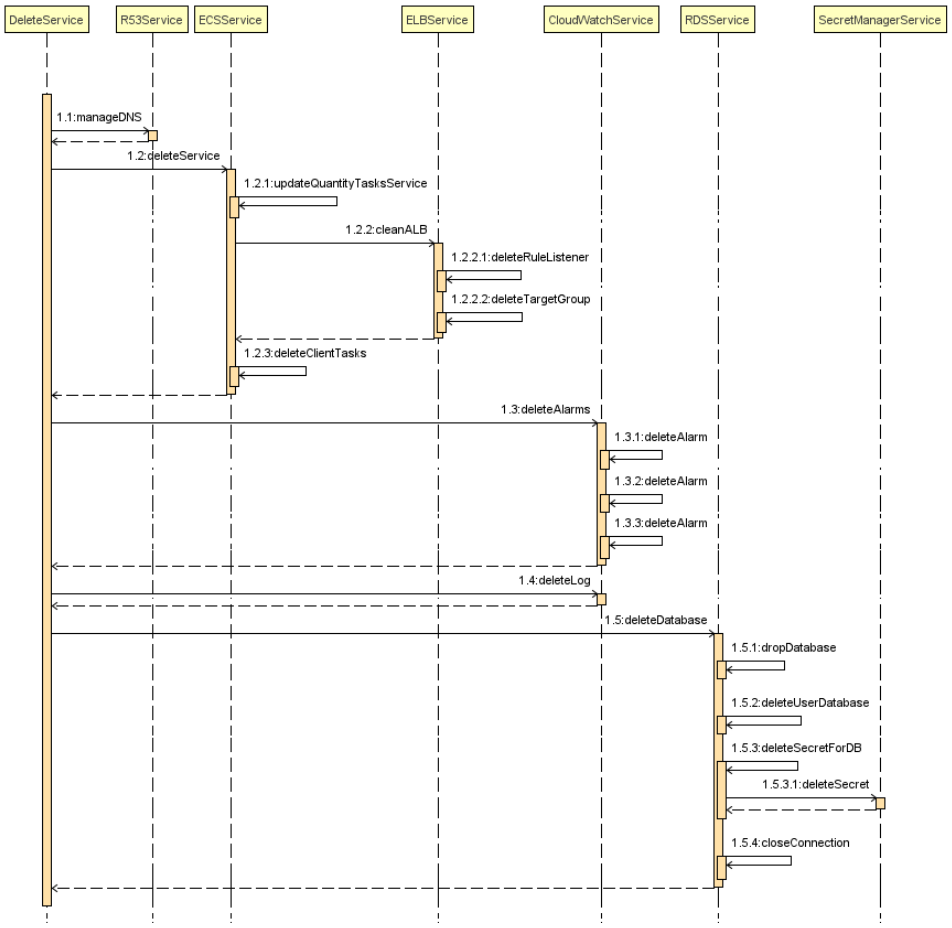


Figura 7.18: Diagrama de secuencia de eliminación de servicio

De la misma forma, en la Figura 7.18 se puede apreciar el diagrama de secuencia a seguir cuando se elimina un servicio. El mismo es muy similar al diagrama de creación de servicio, pero realizando las operaciones opuestas.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

Como conclusión final del trabajo realizado, luego de estudiar los conceptos, comprender y conocer todo lo que abarca el inmenso mundo del Cloud Computing se puede confirmar que las opciones para trabajar en la nube son muy amplias. Las principales empresas que ofrecen servicios cloud están compitiendo codo a codo entre ellas para ofrecer la mayor variedad de productos y servicios haciendo que esta competencia favorezca a los usuarios y mejore la oferta de soluciones entre los principales proveedores.

Podemos afirmar que mediante el desarrollo de esta tesina de grado se ha logrado desarrollar un marco teórico y los lineamientos necesarios para poder migrar un software on-premise a un entorno Cloud a partir del conocimiento de los servicios que nos brinda AWS. Sin embargo, en la solución planteada no se utilizan todos los servicios que puede llegar a ofrecer un proveedor de servicios cloud, sino que se hace uso de los servicios más básicos para lograr diseñar y desplegar la mínima infraestructura funcional, segura y operativa.

También, concluimos que hemos alcanzado la implementación del modelo planteado obteniendo el despliegue y la configuración de un entorno propicio en el Cloud sobre el cual la empresa Optaris pueda instalar y ejecutar su solución de software Deyel.

Con respecto al trabajo a futuro, en la sección 8.2 plantearemos distintas lineas de investigación que podríamos continuar en futuros trabajos de investigación.

Finalmente podemos concluir que el presente trabajo aporta lineamien-

tos para que futuros investigadores puedan contar con una guía de temas relacionados a:

- Conocer los conceptos básicos de cloud computing.
- Conocer los principales proveedores de cloud computing y cuáles son sus fortalezas y debilidades.
- Conocer un como es el proceso de migración de un producto al entorno cloud mediante un ejemplo real desarrollado sobre los servicios de AWS.

8.2. Trabajos futuros

Si bien la base necesaria para ejecutar un servicio en entorno cloud fue implementada, existen muchas funcionalidades que pueden agregar valor a la misma. A continuación, se especifican algunos posibles trabajos futuros que se desprenden de lo realizado en la tesina propuesta:

- Si bien la infraestructura a partir del modelo propuesto se implementó manualmente, sería recomendable considerar como una futura línea de investigación evaluar si existe algún servicio o herramienta que brinde la opción de poder realizar toda la implementación de manera automática para posteriormente poder replicar la infraestructura en varias zonas de AWS.
- En el modelo propuesto se utilizó instancias EC2 t2.micro bajo demanda para la implementación del clúster ECS. Un posible trabajo futuro podría considerar la investigación de los diferentes tipos de instancias bajo demanda que brinda AWS y determinar cuáles serían convenientes de acuerdo al tipo de aplicaciones. También es de interés evaluar si es posible la combinación de este tipo de instancias con instancias SPOT y como se efectuaría dicha combinación.
- Un punto que quedó sin desarrollar en nuestro modelo, es la funcionalidad de auto-escalado de instancias EC2. Otra posible línea de investigación podría ser estudiar y analizar la posibilidad de automatizar la incorporación de nuevos servicios sin la intervención manual.
- El presente trabajo sólo abarcó el despliegue de una aplicación On-Premise en Cloud Computing, sin embargo, es de interés considerar

si es posible investigar sobre el proceso de devops de una aplicación dentro de AWS.

- Por otro lado sería recomendable destinar tiempo de investigación para alcanzar mejoras de performance en seguridad, alta disponibilidad, recuperación ante desastres, y funciones de automatización con menores tiempos de respuesta de soporte y superior experiencia.

Índice de figuras

Bibliografía

[1] M. P. Saakshi Narula, Arushi Jain, "Cloud computing security: Amazon web service," 2015.

[2] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," 2010.

[3] L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," *Computer Communication Review*, vol. 39, pp. 50–55, 01 2009.

[4] NIST, "The nist definition of cloud computing." `https://csrc.nist.gov/publications/detail/sp/800-145/final`, 2011. Último acceso 26 Junio 2020.

[5] XenProyect, "Documentación de proyecto xen.org." `https://wiki.xenproject.org/wiki/XenSource`. Último acceso 26 Junio 2020.

[6] K. project, "Kernal virtual machine." `http://www.linux-kvm.org/page/Main_Page`. Último acceso 26 Junio 2020.

[7] VMWare, "Vmware esx server." `https://www.vmware.com/products/esxi-and-esx.html`. Último acceso 26 Junio 2020.

[8] AWS, "Amazon elastic computing cloud." `https://aws.amazon.com/ec2/`. Último acceso 26 Junio 2020.

[9] Google, "Google app engine." `https://cloud.google.com/appengine`. Último acceso 26 Junio 2020.

[10] Microsoft, "Windows azure." `https://azure.microsoft.com/`. Último acceso 26 Junio 2020.

[11] Salesforce, "Customer 360 platform." `https://www.salesforce.com/products/platform/overview/`. Último acceso 26 Junio 2020.

[12] Rackspace, "Dedicated server, managed hosting, web hosting by rackspace hosting." https://www.rackspace.com. Último acceso 26 Junio 2020.

[13] H. Z. Deyan Chen, "Data security and privacy protection issues in cloud computing," 2012.

[14] P. K. Mariana Carroll, Alta van der Merwe, "Secure cloud computing. benefits, risks and controls," 2011.

[15] B. P. John Harauz, Lori M. Kaufman, "Data security in the world of cloud computing," 2009.

[16] Dataprius, "Las instalaciones on premise. en qué consisten." https://blog.dataprius.com/index.php/2016/06/17/on-premise-servidores-problemas-solucion-cloud/. Último acceso 26 Junio 2020.

[17] Synergy, "Halfyearly review shows $150 billion spent on cloud services and infrastructure," 09 2019. Último acceso 26 Junio 2020.

[18] Gartner, "Gartner says global it spending to grow 1.1 percent in 2019." https://www.gartner.com/en/newsroom/press-releases/2019-04-17-gartner-says-global-it-spending-to-grow-1-1-percent-i, 04 2019. Último acceso 26 Junio 2020.

[19] AWS, "Contrato de nivel de servicio de amazon compute." https://d1.awsstatic.com/legal/AmazonComputeServiceLevelAgreement/Amazon%20Compute%20Service%20Level%20Agreement-ES.pdf, 03 2019. Último acceso 26 Junio 2020.

[20] Google, "Google compute engine launches." https://cloudplatform.googleblog.com/2012/06/google-compute-engine-launches.html. Último acceso 26 Junio 2020.

[21] Google, "Google compute engine available." https://cloudplatform.googleblog.com/2013/12/google-compute-engine-is-now-generally-available.html. Último acceso 26 Junio 2020.

[22] AWS, "Qué es amazon web service?." https://aws.amazon.com/es/what-is-aws/. Último acceso 26 Junio 2020.

[23] Gartner, "Magic quadrant for cloud infrastructure as a service, worldwide." https://pages.awscloud.com/Gartner-Magic-Quadrant-for-Infrastructure-as-a-Service-Worldwide.html. Último acceso 26 Junio 2020.

[24] Synergy, "Incremental growth in cloud spending hits a new high while amazon and microsoft maintain a clear lead," 02 2020. Último acceso 26 Junio 2020.

[25] Deyel, "Documentación de deyel." https://www.deyel.com/doc/Deyel67/. Último acceso 26 Junio 2020.

[26] AWS, "Amazon web services: Overview of security processes. white papers," 04 2020.

[27] A. N. Glen Robinson and C. Elleman, "Amazon web services- using aws for disaster recovery," 10 2014.

[28] AWS, "Aws nitro system." https://aws.amazon.com/es/ec2/nitro/, 03 2019. Último acceso 26 Junio 2020.

[29] AWS, "Documentation of regions and availability zones." https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html. Último acceso 26 Junio 2020.

[30] AWS, "Documentation of route 53." https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/Welcome.html. Último acceso 26 Junio 2020.

[31] AWS, "Documentation of vpc." https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html. Último acceso 26 Junio 2020.

[32] AWS, "Documentation of route tables." https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Route_Tables.html. Último acceso 26 Junio 2020.

[33] AWS, "Documentation of internet gateway." https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Internet_Gateway.html. Último acceso 26 Junio 2020.

[34] AWS, "Documentation of nat gateway." https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html. Último acceso 26 Junio 2020.

[35] Wikipedia, "Modelo osi." https://es.wikipedia.org/wiki/Modelo_OSI. Último acceso 26 Junio 2020.

[36] AWS, "Documentation of application load balancer." https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html. Último acceso 26 Junio 2020.

[37] AWS, "Documentation of elastic computing cloud." https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html. Último acceso 26 Junio 2020.

[38] AWS, "Amazon ebs." https://aws.amazon.com/es/ebs/?ebs-whats-new.sort-by=item.additionalFields.postDateTime&ebs-whats-new.sort-order=desc. Último acceso 26 Junio 2020.

[39] AWS, "Amazon elastic container service." https://aws.amazon.com/es/ecs/. Último acceso 26 Junio 2020.

[40] AWS, "Amazon elastic container registry." https://aws.amazon.com/es/ecr/. Último acceso 26 Junio 2020.

[41] AWS, "Amazon rds." https://aws.amazon.com/es/rds/. Último acceso 26 Junio 2020.

[42] AWS, "Amazon aurora." https://aws.amazon.com/es/rds/aurora/. Último acceso 26 Junio 2020.

[43] AWS, "Amazon s3." https://aws.amazon.com/es/s3/. Último acceso 26 Junio 2020.

[44] AWS, "Amazon api gateway." https://aws.amazon.com/es/api-gateway/. Último acceso 26 Junio 2020.

[45] AWS, "Amazon lambda." https://aws.amazon.com/es/lambda/. Último acceso 26 Junio 2020.

[46] AWS, "System and organization controls 3(soc 3) - report on the amazon web services system relevant to security,availability, and confidentiality." https://d1.awsstatic.com/whitepapers/compliance/AWS_SOC3.pdf, 2019. Último acceso 26 Junio 2020.

[47] AWS, "Amazon shield." https://aws.amazon.com/es/shield/. Último acceso 26 Junio 2020.

[48] AWS, "Amazon waf." https://aws.amazon.com/es/waf/. Último acceso 26 Junio 2020.

[49] AWS, "Aws secret manager." https://aws.amazon.com/es/secrets-manager/. Último acceso 26 Junio 2020.

[50] AWS, "Aws simple notification service." https://docs.aws.amazon.com/sns/latest/dg/welcome.html. Último acceso 26 Junio 2020.

[51] AWS, "Aws cloudtrail." https://aws.amazon.com/es/cloudtrail/. Último acceso 26 Junio 2020.

[52] AWS, "Aws certificate manager." https://aws.amazon.com/es/certificate-manager/. Último acceso 26 Junio 2020.

[53] AWS, "Aws identity and access management." https://aws.amazon.com/es/iam/. Último acceso 26 Junio 2020.

[54] AWS, "Certificación iso." https://aws.amazon.com/es/compliance/iso-certified/. Último acceso 26 Junio 2020.

[55] OWASP, "Who is the owasp foundation?." https://owasp.org. Último acceso 26 Junio 2020.

[56] AWS, "Aws cloudwatch." https://docs.aws.amazon.com/AmazonECS/latest/developerguide/cloudwatch-metrics.html. Último acceso 26 Junio 2020.

[57] AWS, "Using api gateway stage variables to manage lambda functions." https://aws.amazon.com/es/blogs/compute/using-api-gateway-stage-variables-to-manage-lambda-functions/. Último acceso 26 Junio 2020.

Anexos

Anexos A

Funciones Lambda

A continuación se especifican cada una de las clases que componen el proyecto Java para las funciones de creación/eliminación de servicios.

Clase CreateService

```
public class CreateService extends LambdaService implements
    RequestHandler<Object, String> {
2   private static final Logger logger = LogManager.getLogger(
    CreateService.class);

4   @Override
    public String handleRequest(Object input, @NotNull Context context
    ) {
6       JSONObject checker = new JSONObject();
        try {
8           JSONObject map = parameterToJSON(input);
            checker = checkParameter(map, new String[]{"subdomain"});
10          if (!checker.isNull(ERROR)) {
                return checker.toString();
12          }
            setCorrectParameters(map);
14          return new JSONObject(createService()).toString();
        } catch (Exception e) {
16          logger.error(e.getMessage());
            return checker.toString();
18      }
    }
20
    /**
22   * Create a Service
     * @return
24   */
    private Map<String, String> createService() {
26      Map<String, String> response;
```

```
        Map<String, String> map_resultService = new HashMap<>();

        R53Service r53_service = new R53Service();
        //Check exist domain register
        if (r53_service.existDomain().get(ERROR) != null) {
            map_resultService.put(ERROR, "There is the registered subdomain");
            return map_resultService;
        }

        //Create data base
        RDSService rds_service = new RDSService();
        map_resultService = rds_service.createDatabase();

        if (map_resultService.get(ERROR) != null) {
            logger.error("Error in create DB");
            rds_service.deleteDatabase();
            return map_resultService;
        }

        //Create Service
        ELBService elb_service = new ELBService();
        ECSService ecs_service = new ECSService();
        map_resultService = ecs_service.createService(elb_service);

        if (map_resultService.get(SUCCESS) != null) {
            //Register alarms
            CloudWatchService cw_service = new CloudWatchService();
            cw_service.addAlarms(elb_service.getTargetGroupByUser());
            //Create entry in Route 53
            map_resultService = r53_service.manageDNS(ACTION_CREATE, elb_service);
            if (map_resultService.get(SUCCESS) != null) {
                if (checkStatusServices()) {
                    logger.info("Create Services OK: " + "https://" + subdomain + "." + domain);
                } else {
                    logger.error("Could not start service");
                    r53_service.manageDNS(ACTION_DELETE, elb_service);
                    map_resultService.clear();
                    map_resultService.put(ERROR, "Could not start service");
                }
            } else {
                logger.error("Error in generate DNS");
            }
            response = map_resultService;
            if (map_resultService.get(ERROR) != null) {
                try {
                    ecs_service.deleteService(elb_service);
                    rds_service.deleteDatabase();
                } catch (Exception e2) {
                    logger.error("Delete services: " + e2.getMessage());
                }
                return response;
```

```
78              }
            } else {
80              try {
                    logger.error("Error in create services");
82                  rds_service.deleteDatabase();
                } catch (Exception e2) {
84                  logger.error("Delete services: " + e2.getMessage());
                }
86              response = map_resultService;
            }
88          return response;
        }
90  }
```

Clase DeleteService

```
    public class DeleteService extends LambdaService implements
        RequestHandler<Object, String> {
2    private static final Logger logger = LogManager.getLogger(
        DeleteService.class);
        @Override
4       public String handleRequest(Object input, Context context) {
            JSONObject map = parameterToJSON(input);
6           JSONObject checker = checkParameter(map, new String[]{"
        subdomain"});
            if (checker.isNull(SUCCESS)) {
8               return checker.toString();
            }
10          //Set correct user
            setCorrectParameters(map);
12          return new JSONObject(deleteService()).toString();
        }
14
        protected Map<String, String> deleteService() {
16          //First, delete the DNS
            R53Service r53_service = new R53Service();
18          ELBService elb_service = new ELBService();
            Map<String, String> response = r53_service.manageDNS(
        ACTION_DELETE, elb_service);
20          if (response.containsKey(SUCCESS)) {
                //delete the service, tasks and delete the rule in ALB and
         delete tg
22              ECSService ecs_service = new ECSService();
                response = ecs_service.deleteService(elb_service);
24              if (response.containsKey(SUCCESS)) {
                    //Desregister alarms
26                  CloudWatchService cw_service = new CloudWatchService()
        ;
                    cw_service.deleteAlarms();
28                  cw_service.deleteLog();
```

```
                //Delete the DB
                RDSService rds_service = new RDSService();
                response = rds_service.deleteDatabase();
            }
        }
        return response;
    }
}
```

Clase LambdaService

```
public abstract class LambdaService {
    protected final String ERROR = "error";
    protected final String SUCCESS = "ok";
    protected final String ACTION_CREATE = "CREATE";
    protected final String ACTION_DELETE = "DELETE";
    /**
     * Definition of environment variables
     */
    protected static String subdomain;
    protected static String build;
    protected static String domain;
    protected static String elb_load_balancer;
    protected static String ecsCluster;
    protected static String region_aws;
    protected static Integer memoryReservationHardware = 1200;
    protected static Integer memoryReservationSoftware = 900;
    protected static String imageDockerRepository;
    protected static String ecsRole;
    protected static String client_passw = "contraseniaDemo";
    protected static String db_host;
    protected static String db_port;
    protected static String db_master_user;
    protected static String db_master_pass;
    protected static String initial_db_bucket;
    protected static String path_to_db;
    protected static String db_cluster_identifier;
    protected static String sql_driver ;
    protected static String vpc_id;
    protected static String dns_hosted_zone_id;

    /**
     * Check list of parameters in nameParameters
     */
    protected JSONObject checkParameter(JSONObject parameters, String
    [] nameParameters) {
        JSONObject checker = new JSONObject();
        Map<String, String> errors = new HashMap<String, String>();
        for (int i = 0; i < nameParameters.length; i++) {
            if (parameters.get(nameParameters[i]).equals("")) {
```

```
                errors.put(nameParameters[i], nameParameters[i] + " is
 null or empty");
            }
        }
        if (errors.size() > 0) {
            checker.put(ERROR, new JSONObject(errors));
        } else {
            checker.put(SUCCESS, "");
        }
        return checker;
    }

    protected void setCorrectParameters(JSONObject map) {
        if (!map.isNull("build_default")) {
            build = ((String) map.get("build_default")).toUpperCase();
        }
        if ((!map.isNull("subdomain")) && (!map.get("subdomain").
equals(""))) {
            subdomain = (map.get("subdomain")).toString().toLowerCase
();
        }
        if (!map.isNull("ecs_cluster")) {
            ecsCluster = (map.get("ecs_cluster")).toString();
        }
        if (!map.isNull("sql_driver")) {
            sql_driver = (map.get("sql_driver")).toString();
        }else{
            sql_driver= "com.mysql.jdbc.Driver";
        }
        if (!map.isNull("db_host")) {
            db_host = (map.get("db_host")).toString();
        }
        if (!map.isNull("db_port")) {
            db_port = (map.get("db_port")).toString();
        }
        if (!map.isNull("db_cluster_identifier")) {
            db_cluster_identifier = (map.get("db_cluster_identifier"))
.toString();
        }
        if (!map.isNull("vpc_id")) {
            vpc_id = (map.get("vpc_id")).toString();
        }
        if (!map.isNull("domain")) {
            domain = (map.get("domain")).toString();
        }
        if (!map.isNull("region_aws")) {
            region_aws = (map.get("region_aws")).toString();
        }
        if (!map.isNull("elb_load_balancer")) {
            elb_load_balancer = (map.get("elb_load_balancer")).
toString();
        }
        if (!map.isNull("initial_db_bucket")) {
            initial_db_bucket = (map.get("initial_db_bucket")).
toString();
        }
```

```
            if (!map.isNull("path_to_db")) {
90              path_to_db = (map.get("path_to_db")).toString();
            }
92          if (!map.isNull("db_master_user")) {
                db_master_user = (map.get("db_master_user")).toString();
94          }
            if (!map.isNull("db_master_pass")) {
96              db_master_pass = (map.get("db_master_pass")).toString();
            }
98          if (!map.isNull("image_docker_repository")) {
                imageDockerRepository = (map.get("image_docker_repository"
    )).toString();
100         }
            if (!map.isNull("ecs_role")) {
102             ecsRole = (map.get("ecs_role")).toString();
            }
104         if (!map.isNull("dns_hosted_zone_id")) {
                dns_hosted_zone_id = (map.get("dns_hosted_zone_id")).
    toString();
106         }
        }
108

110     protected JSONObject parameterToJSON(Object input) {
            JSONObject map = new JSONObject();
112         try {
                //Check the parameters aren't null or empty
114             String xTemp = input.toString().replace("@@@", "*").
    replace("=", ":");
                xTemp = xTemp.substring(1);
116             //Represent parameters in the body of request
                String createService = xTemp.replace("createServiceBean:{"
    , "");
118             createService = createService.replace("body:{", "");
                createService = createService.replace("{", "");
120             createService = createService.replaceAll("}", "");
                String[] xAux = createService.split(",");
122             for (String xVar : xAux) {
                    map.put(xVar.substring(0, xVar.indexOf(":")).trim(),
    xVar.substring(xVar.indexOf(":") + 1).trim());
124             }
            } catch (Exception e) {
126             map.put(ERROR, e.getMessage());
            }
128         return map;
        }
130 }
```

Clase ECSService

```
public class ECSService extends LambdaService {
    private final Logger logger = LogManager.getLogger(ECSService.
    class);
    private AmazonECS ecs_client;
    private AmazonECR ecr_client;

    public ECSService() {
        ecs_client = AmazonECSClientBuilder.standard().withRegion(
    Regions.fromName(region_aws)).build();
        ecr_client = AmazonECRClientBuilder.standard().withRegion(
    Regions.fromName(region_aws)).build();
    }

    /**
     * Create the client's service.
     */
    public Map<String, String> createService(@NotNull ELBService
    pELBService) {
        Map<String, String> map_service = new HashMap<String, String
    >();
        try {
            List<KeyValuePair> containerEnvironment =
    getContainerEnvironment();
            //Create the task definition for user
            TaskDefinition xTaskDefinition = createTaskDefinition(
    containerEnvironment);
            //Create request for create service
            CreateServiceRequest request_service_create =
    getRequestService(xTaskDefinition,pELBService);
            //Execute request for create service
            CreateServiceResult result_service_create = ecs_client.
    createService(request_service_create);
            logger.info("Service created: " + result_service_create.
    toString());
            map_service.put(SUCCESS, "");
        } catch (Exception e) {
            logger.error("Rollback of client task: " + e.getMessage())
    ;
            deleteService(pELBService);
            map_service.put(ERROR, e.getMessage());
        }
        return map_service;
    }

    /**
     * Return the request for create a service from previous register
    new task in loadbalancer
     */
    private CreateServiceRequest getRequestService(@NotNull
    TaskDefinition xTaskDefinition, ELBService pELBService) {
        String client_container = xTaskDefinition.
    getContainerDefinitions().get(0).getName();
        //Register new task in loadbalancer
        LoadBalancer loadBalancer = pELBService.registerServiceALB(
    client_container);
```

```
42          int xRevisionLastTask = getLastTaskDefinitionByClient();
            if (xRevisionLastTask == 0) {
44              xRevisionLastTask = xTaskDefinition.getRevision();
            }
46          CreateServiceRequest request_service_create;

48          request_service_create = new CreateServiceRequest()
                    .withServiceName(subdomain.toLowerCase())
50                  .withTaskDefinition(subdomain.toLowerCase() + ":" +
    xRevisionLastTask)
                    .withCluster(ecsCluster)
52                  .withLoadBalancers(loadBalancer)
                    .withDesiredCount(1)
54                  .withRole(ecsRole);
            return request_service_create;
56      }

58      private List<KeyValuePair> getContainerEnvironment() {
            List<KeyValuePair> containerEnvironment = new LinkedList<>();
60          containerEnvironment.add(new KeyValuePair().withName("CLIENT")
    .withValue(subdomain));
            containerEnvironment.add(new KeyValuePair().withName("SQL_USER
    ").withValue(subdomain));
62          containerEnvironment.add(new KeyValuePair().withName("
    SQL_DRIVER").withValue(sql_driver));
            containerEnvironment.add(new KeyValuePair().withName("SQL_URL"
    ).withValue("jdbc:mysql://" + db_host + ":" + db_port + "/" +
    subdomain));
64          containerEnvironment.add(new KeyValuePair().withName("
    REGION_AWS").withValue(region_aws));
            return containerEnvironment;
66      }

68      /**
         * Get definition of docker container
70       */
        private ContainerDefinition getContainerDefinition(List<
    KeyValuePair> container_environment) {
72          //Get description of the repository
            DescribeRepositoriesRequest request_describe_repository = new
    DescribeRepositoriesRequest().withRepositoryNames(
    imageDockerRepository);
74          DescribeRepositoriesResult result_describe_repository =
    ecr_client.describeRepositories(request_describe_repository);
            Repository repository_deyel = result_describe_repository.
    getRepositories().get(0);
76          //Define the port mapping
            PortMapping port_mapping = new PortMapping()
78                  .withContainerPort(8080)
                    .withHostPort(0)
80                  .withProtocol(TransportProtocol.Tcp);
            //Set mount point for service
82          MountPoint mount_point = new MountPoint()
                    .withContainerPath("/opt/BACKUP")
84                  .withSourceVolume("backup");
            //Container definition with
```

```
86          return new ContainerDefinition()
                    .withEssential(true)
88                  .withImage(repository_deyel.getRepositoryUri() + ":" +
     build)
                    .withName(subdomain)
90                  .withMemory(memoryReservationHardware)
                    .withMemoryReservation(memoryReservationSoftware)
92                  .withMountPoints(mount_point)
                    .withEnvironment(container_environment)
94                  .withPortMappings(port_mapping)
                    .withWorkingDirectory("/opt/tomcat");
96      }

98      /**
         * Search for the client's task and iterate through the task
        definitions.
100      */
        private void deleteClientTasks() {
102         List<String> all_tasks_clients = getAllTaskDefinitionByClient
        ();
            if (all_tasks_clients != null) {
104             all_tasks_clients.forEach(this::deleteTask);
            } else {
106             logger.error("Not exist definition for a service");
            }
108     }

110     public Map<String, String> deleteService(ELBService pELBService) {
            HashMap<String, String> response = new HashMap<>();
112         try {
                //Desactive task service
114             Service service_client = updateQuantityTasksService(0);
                //Clean ALB
116             pELBService.cleanALB();
                //delete all cient task
118             deleteClientTasks();

120             DeleteServiceRequest request_delete_service = new
        DeleteServiceRequest()
                        .withService(service_client.getServiceName())
122                     .withCluster(ecsCluster);

124             DeleteServiceResult result_service_delete = ecs_client.
        deleteService(request_delete_service);
                logger.info("Service deleted: " + result_service_delete.
        getService());
126             response.put(SUCCESS, "");
            } catch (Exception e) {
128             response.put(ERROR, e.getMessage());
                logger.error("Deleting Service: " + e.getMessage());
130         }
            return response;
132     }

134
        /**
```

```
136      * updateQuantityTasksService
         */
138     private Service updateQuantityTasksService(int total_tasks) throws
         NullPointerException {
            UpdateServiceRequest request_update_service = new
        UpdateServiceRequest()
140                 .withService(subdomain.toLowerCase())
                    .withCluster(ecsCluster)
142                 .withDesiredCount(total_tasks);
            UpdateServiceResult result_update_service = ecs_client.
        updateService(request_update_service);
144         return result_update_service.getService();
        }
146
        /**
148      * Create the request to deregister the task.
         */
150     private void deleteTask(String task_id) throws
        NullPointerException {
            DeregisterTaskDefinitionRequest request_deregister_td = new
        DeregisterTaskDefinitionRequest().withTaskDefinition(task_id);
152         DeregisterTaskDefinitionResult result_deregister_td =
        ecs_client.deregisterTaskDefinition(request_deregister_td);
            logger.info("Deregister task definition: " +
        result_deregister_td.toString());
154     }

156     /**
         * Create the client's task.
158      */
        private TaskDefinition createTaskDefinition(List<KeyValuePair>
        containerEnvironment) throws NullPointerException {
160         //Get container definition
            ContainerDefinition container_definition =
        getContainerDefinition(containerEnvironment);
162         //Create volume to associate
            Volume volume = new Volume()
164                 .withName("backup")
                    .withHost(new HostVolumeProperties()
166                         .withSourcePath("/opt/BACKUP"));
            //Create request for register new task
168         RegisterTaskDefinitionRequest request_register_td = new
        RegisterTaskDefinitionRequest()
                    .withContainerDefinitions(container_definition)
170                 .withFamily(subdomain)
                    .withVolumes(volume);
172         //Register new task
            RegisterTaskDefinitionResult result_register_td = ecs_client.
        registerTaskDefinition(request_register_td);
174         logger.info("Task created: " + result_register_td.toString());
            return result_register_td.getTaskDefinition();
176     }

178     /**
         * Return the last client's task version
180      */
```

```
    private int getLastTaskDefinitionByClient() {
182         List<String> all_tasks_clients = getAllTaskDefinitionByClient
    ();
            if (all_tasks_clients.size() != 0) {
184             // Get the task definition's registered
                List<Integer> client_task_versions = all_tasks_clients.
    stream()
186                     .map(s -> Integer.parseInt(s.split(subdomain + ":"
    )[1]))
                        .collect(Collectors.toList());
188
                if (client_task_versions != null) {
190                 //Lambda function to get the last version of a client'
    s task definition
                    return client_task_versions.stream()
192                         .mapToInt(v -> v)
                            .max()
194                         .orElseThrow(NoSuchElementException::new);
                } else return 0;
196         } else return 0;
    }
198
    /**
200  * Search all client's task definitions
     */
202 private List<String> getAllTaskDefinitionByClient() {
        ListTaskDefinitionsRequest request = new
    ListTaskDefinitionsRequest();
204     request.setFamilyPrefix(subdomain);
        logger.info("Se busca definicin de tarea con prefijo " +
    subdomain);
206     return ecs_client.listTaskDefinitions(request).
    getTaskDefinitionArns()
                .stream()
208             .filter(s -> s.contains(subdomain))
                .collect(Collectors.toList());
210 }
}
```

Clase RDSService

```
public class RDSService extends LambdaService {
2
    private static final Logger logger = LogManager.getLogger(
    RDSService.class);
4   private static final String mysqldriver = "com.mysql.cj.jdbc.
    Driver";
    private Connection connection;
6
    /**
```

```
     * Get connection from RDS
     */
    private Connection getConnection() throws SQLException {
        if (connection == null) {
            return createConnection();
        }
        return connection;
    }

    /**
     * Create connection to RDS
     */
    private Connection createConnection() throws SQLException {
        Class.forName(mysqldriver);
        String string_connection = "jdbc:mysql://" + db_host + ":" +
db_port + "/?useUnicode=yes&characterEncoding=UTF-8&
useLegacyDatetimeCode=false&serverTimezone=UTC";
        connection = DriverManager.getConnection(string_connection,
db_master_user, db_master_pass);
        return connection;
    }

    /**
     * Close connection
     */
    private void closeConnection() {
        try {
            if (connection != null) {
                connection.close();
            }
        } catch (SQLException e) {
            logger.error("Error close connection");
            logger.error(e.getMessage());
        } finally {
            connection = null;
        }
    }


    /**
     * Create schema data base for getDatabase_name()
     */
    private void createSchemaDatabase() throws SQLException {
        logger.info("Start create schema data base");
        try (Statement statement = getConnection().createStatement())
{
            String sql_create_database = "CREATE DATABASE " +
subdomain + ";";
            statement.executeUpdate(sql_create_database);
            logger.info("The database has been successfully created");
        } catch (Exception e) {
            logger.error("An unexpected error occurred: " + e.
getMessage());
            throw e;
        }
    }
```

```
58
    /**
60   * Create user data base for getDatabase_user()
     */
62  private void createUserDatabase() throws SQLException {
        logger.info("Start create user data base");
64      try (Statement statement = getConnection().createStatement())
    {
            String create_user_database = "CREATE USER \"" + subdomain
     + "\"@'%' IDENTIFIED BY '" + client_passw + "' ;";
66          statement.executeUpdate(create_user_database);
            String privileges_user_database = "GRANT ALL PRIVILEGES ON
     " + subdomain + ".* TO " + subdomain + "@'%' IDENTIFIED BY '" +
    client_passw + "' ;";
68          statement.executeUpdate(privileges_user_database);
            String grant_option_user_database = "GRANT GRANT OPTION ON
     " + subdomain + ".* TO " + subdomain + "@'%' IDENTIFIED BY '" +
    client_passw + "' ;";
70          statement.executeUpdate(grant_option_user_database);
        } catch (Exception e) {
72          logger.error("Failed to create the user for the database.
    " + e.getMessage());
            throw e;
74      }
        logger.info("End Create User Database ");
76  }

78  private void updatePassword(String pUser, String password) throws
    Exception{
     try (Statement statement = getConnection().createStatement()){
80    String update_user_password = "ALTER USER \"" + pUser + "\"@'%'
    IDENTIFIED BY '" + password + "' ;";
      statement.executeUpdate(update_user_password);
82    logger.info("Password update");
     } catch (Exception e){
84    logger.error("Failed to update password. " + e.getMessage());
      throw e;
86   }finally {
            closeConnection();
88      }
    }
90
    private void createSecretForDB() throws Exception {
92      SecretManagerService xSecretManager = new SecretManagerService
    ();
        String passwordUpdate = xSecretManager.createSecret(subdomain
     , client_passw);
94      if(passwordUpdate != null){
            updatePassword(subdomain,passwordUpdate);
96      }
    }
98
    private void deleteSecretForDB(){
100  SecretManagerService xSecretManager = new SecretManagerService();
     xSecretManager.deleteSecret();
102 }
```

```
104     /**
         * Create data base
106      */
        public Map<String, String> createDatabase() {
108         HashMap<String, String> map_database = new HashMap<String,
        String>();
            try {
110             createSchemaDatabase();
                createUserDatabase();
112             createSecretForDB();
                restoreDatabase();
114             map_database.put(SUCCESS, "");
            } catch (Exception e) {
116             map_database.put(ERROR, e.getMessage());
            } finally {
118             closeConnection();
            }
120
            return map_database;
122     }

124     public Map<String, String> deleteDatabase() {
            Map<String, String> response = new HashMap<String, String>();
126         try {
                dropDatabase();
128             deleteUserDatabase();
                deleteSecretForDB();
130             response.put(SUCCESS, "");
            } catch (SQLException e) {
132             logger.error("An unexpected error occurred. " + e.
        getMessage());
                response.put(ERROR, e.getMessage());
134         } finally {
                closeConnection();
136         }
            return response;
138     }

140     private void dropDatabase() throws SQLException {
            try (Statement statement = getConnection().createStatement())
        {
142             String sql_delete_database = "DROP DATABASE IF EXISTS " +
        subdomain  + ";";
                statement.executeUpdate(sql_delete_database);
144         } catch (SQLException e) {
                logger.error("An unexpected error occurred. " + e.
        getMessage());
146             throw e;
            }
148     }

150     private void deleteUserDatabase() throws SQLException {
            try (Statement statement = getConnection().createStatement())
        {
```

```
152             String delete_user_database = "DROP USER IF EXISTS " +
    subdomain + "@'%';";
                logger.info("Sentence: " + delete_user_database);
154             statement.executeUpdate(delete_user_database);
                logger.info("[INFO] User Deleted");
156         } catch (SQLException e) {
                logger.error("An unexpected error occurred. " + e.
    getMessage());
158             throw e;
            }
160     }

162     /**
         * Restore DB for file SQL in S3
164      */
        private void restoreDatabase() throws Exception {
166         String currentSentence = "";
            BufferedReader reader;
168         String line;
            String delimiter = ";";
170         try {
                logger.info("Start obtain file from S3");
172             InputStream xInputStream = null;
                xInputStream = new S3Service().getObject(initial_db_bucket
    , path_to_db + build + ".sql").getObjectContent();
174             reader = new BufferedReader(new InputStreamReader(
    xInputStream));
                line = reader.readLine();
176         } catch (Exception e) {
                logger.error("Error connect to S3Service: " + e.getMessage
    ());
178             throw e;
            }
180         try (Statement statement = getConnection().createStatement())
    {
                statement.executeUpdate("USE " + subdomain);
182             while (line != null) {
                    if (line.toUpperCase().contains("DELIMITER")) {
184                     delimiter = line.split(" ")[1];
                    } else {
186                     currentSentence += "\n" + new String(line.getBytes
    (StandardCharsets.UTF_8));
                        if (line.endsWith(delimiter)) {
188                         if (!delimiter.equals(";")) {
                                currentSentence = currentSentence.replace(
    delimiter, ";");
190                         }
                            statement.executeUpdate(currentSentence);
192                         currentSentence = "";
                        }
194                 }
                    line = reader.readLine();
196             }
                logger.info("Finish restore DB");
198         } catch (Exception e) {
```

```
                logger.error("Error in restore database. Sentence: " +
    currentSentence);
200             logger.error(e.getMessage());
                throw e;
202         }

204     }
}
```

Clase SecretManagerService

```
public class SecretManagerService extends LambdaService {
 private AWSSecretsManager client;
 private final Logger logger = LogManager.getLogger(
    SecretManagerService.class);
4
 public SecretManagerService (){
6  client = AWSSecretsManagerClientBuilder.standard().withRegion(
    region_aws).build();
 }
8
 /**
10  * Create secret and return password
  */
12 public String createSecret(String pUser, String pPassword) throws
    Exception {
  CreateSecretRequest xCreateSecretRequest = new CreateSecretRequest()
    .withName(pUser)
14    .withSecretString("{" + "\"username\": \"" + pUser + "\"," + "\"
    engine\": \"mysql\","
      + "\"dbClusterIdentifier\": \""+db_cluster_identifier+"\","
16      + "\"host\": \""+db_host+"\","
      + "\"password\":\"" + pPassword + "\"," + "\"port\":\""+db_port+
    "\"" + "}")
18    .withDescription("Secret to the base of the service " + pUser);
  boolean returnValue = true;
20  try {
   CreateSecretResult xCreateSecretResult = client.createSecret(
    xCreateSecretRequest);
22   logger.info("Secret create with name: " + xCreateSecretResult.
    getName());
   returnValue = false;
24  } catch (InvalidRequestException e) {
   RestoreSecretRequest restoreSecretRequest = new
    RestoreSecretRequest();
26  restoreSecretRequest.setSecretId(pUser);
   client.restoreSecret(restoreSecretRequest);
28  } catch (ResourceExistsException e) {
   logger.info("El secret ya existe. " + e.getMessage());
30  } catch (Exception e) {
```

```
    logger.error("Error to create secret : " + e.getMessage());
32   returnValue = false;
     throw e;
34  }finally {
     if(returnValue){
36    return getSecret(pUser);
     }else return null;
38  }
}
40
public void deleteSecret() {
42  DeleteSecretRequest xDeleteSecretRequest = new DeleteSecretRequest()
      .withSecretId(subdomain);
    try {
44   DeleteSecretResult xDeleteSecretResult = client.deleteSecret(
      xDeleteSecretRequest);
     logger.info("Secret delete succesfully");
46  } catch (Exception e) {
     logger.error("Error to delete secret: " + e.getMessage());
48  }
}
50
public String getSecret(String secretName) {
52  // Create a Secrets Manager client
    AWSSecretsManager client = AWSSecretsManagerClientBuilder.standard()
      .withRegion(region_aws).build();
54  String secret, decodedBinarySecret;
    GetSecretValueRequest getSecretValueRequest = new
      GetSecretValueRequest().withSecretId(secretName);
56  GetSecretValueResult getSecretValueResult = client.getSecretValue(
      getSecretValueRequest);

58  if (getSecretValueResult.getSecretString() != null) {
     secret = getSecretValueResult.getSecretString();
60   Gson gson = new Gson();
     AWSSecret awsSecret = gson.fromJson(secret, AWSSecret.class);
62   return awsSecret.getPassword();
    } else {
64   decodedBinarySecret = new String(
       Base64.getDecoder().decode(getSecretValueResult.getSecretBinary()
     ).array());
66   return decodedBinarySecret;
    }
68 }

70 private class AWSSecret{
    private String username;
72     private String engine;
       private String dbClusterIdentifier;
74     private String host;
       private String password;
76     private String port;

78     public String getUsername() {
           return username;
80     }
```

```
82      public void setUsername(String username) {
            this.username = username;
84      }

86      public String getEngine() {
            return engine;
88      }

90      public void setEngine(String engine) {
            this.engine = engine;
92      }

94      public String getDbClusterIdentifier() {
            return dbClusterIdentifier;
96      }

98      public void setDbClusterIdentifier(String dbClusterIdentifier) {
            this.dbClusterIdentifier = dbClusterIdentifier;
100     }

102     public String getHost() {
            return host;
104     }

106     public void setHost(String host) {
            this.host = host;
108     }

110     public String getPassword() {
            return password;
112     }

114     public void setPassword(String password) {
            this.password = password;
116     }

118     public String getPort() {
            return port;
120     }

122     public void setPort(String port) {
            this.port = port;
124     }
    }
126 }
```

Clase ELBService

```
public class ELBService extends LambdaService {
```

```
2       private AmazonElasticLoadBalancing client_elb;
        private final Logger logger = LogManager.getLogger(ELBService.
        class);
4
        public ELBService(){
6           client_elb = AmazonElasticLoadBalancingClientBuilder.standard
        ().withRegion(Regions.fromName(region_aws)).build();
        }
8
        public void cleanALB() {
10          try {
                TargetGroup tg = getTargetGroupByUser();
12              deleteRuleListener(tg);
                deleteTargetGroup(tg);
14          } catch (Exception e) {
                logger.error("In cleanALB " + e.getMessage());
16          }
        }
18
        @Nullable
20      public LoadBalancer registerServiceALB(String
        client_container_name) {
            TargetGroup tg = this.createTargetGroup();
22          if (tg == null) {
                return null;
24          }
            try {
26              this.createRuleToListener(tg);
            } catch (NullPointerException e) {
28              deleteRuleListener(tg);
                deleteTargetGroup(tg);
30              return null;
            }
32          LoadBalancer load_balancer = new LoadBalancer()
                    .withTargetGroupArn(tg.getTargetGroupArn())
34                  .withContainerName(client_container_name)
                    .withContainerPort(8080);
36
            logger.info("Load balancer: " + load_balancer.toString());
38          return load_balancer;
        }
40
        /**
42       * Delete listener rule from target group
         */
44      public void deleteRuleListener(TargetGroup tg) throws
        NullPointerException {
            //Obtain all listeners
46          DescribeListenersRequest request_describe_listener = new
        DescribeListenersRequest()
                    .withLoadBalancerArn(getLoadBalancer().
        getLoadBalancerArn());
48          DescribeListenersResult result_describe_listener = client_elb.
        describeListeners(request_describe_listener);
            //Get the listener in 443 port
50          Listener listener = result_describe_listener.getListeners()
```

```
                .stream()
52              .filter(l -> l.getPort() == 443)
                .findAny()
54              .orElse(null);

56          if (listener == null) {
                logger.error("Listener null");
58              return;
            }
60          logger.info("Listener: " + listener.toString());

62          //Obtain the rule with the previous listener
            DescribeRulesRequest request_describe_rule = new
    DescribeRulesRequest()
64              .withListenerArn(listener.getListenerArn());
            DescribeRulesResult result_describe_rule = client_elb.
    describeRules(request_describe_rule);
66
            // search in rules conditions if the header exists
68          Rule rule_client = result_describe_rule
                    .getRules()
70                  .stream()
                    .filter(rule -> rule.getActions()
72                          .stream()
                            .filter(action -> action.getTargetGroupArn().
    equals(tg.getTargetGroupArn()))
74                          .findAny()
                            .orElse(null) != null)
76              .findAny()
                .orElse(null);
78          //Delete the rule listener
            DeleteRuleRequest request_delete_rule = new DeleteRuleRequest
    ()
80              .withRuleArn(rule_client.getRuleArn());
            DeleteRuleResult result_delete_rule = client_elb.deleteRule(
    request_delete_rule);
82          logger.info("Result delete rule: " + result_delete_rule.
    toString());
    }
84
    /**
86   * Create a Rule to the Load Balancer listener.
     */
88  private void createRuleToListener(@NotNull TargetGroup tg) throws
    NullPointerException {
        List<Integer> priorities = new LinkedList<>();
90      client_elb.describeRules(new DescribeRulesRequest().
    withListenerArn(this.getListenerArn())).getRules()
                .stream()
92              .forEach(rule -> {
                    try {
94                      priorities.add(Integer.parseInt(rule.
    getPriority()));
                    } catch (NumberFormatException e) {
96                      logger.error("Can't convert int " + rule.
    getPriority() + ". Error " + e.getMessage());
```

```
                    }
98                });
        Integer last_priority = (priorities
100                .stream()
                .mapToInt(v -> v)
102                .max().orElse(2) + 1);

104         String RULE_CONDITION_FIELD = "host-header";
        CreateRuleRequest request_create_rule = new CreateRuleRequest
    ()
106                 .withPriority(last_priority)
                .withListenerArn(this.getListenerArn())
108                 .withConditions(new RuleCondition()
                        .withField(RULE_CONDITION_FIELD)
110                         .withValues( subdomain + "." + domain))
                .withActions(new Action()
112                         .withTargetGroupArn(tg.getTargetGroupArn())
                        .withType(ActionTypeEnum.Forward));
114
        CreateRuleResult result_create_rule = client_elb.createRule(
    request_create_rule);
116         logger.info("Rule created: " + result_create_rule.toString());
        result_create_rule.getRules();
118     }

120     /**
         * Delete the target group.
122      */
    @NotNull
124     private void deleteTargetGroup(@NotNull TargetGroup tg) throws
    NullPointerException {
        DeleteTargetGroupRequest request_delete_tg = new
    DeleteTargetGroupRequest()
126                 .withTargetGroupArn(tg.getTargetGroupArn());
        DeleteTargetGroupResult result_delete_tg = client_elb.
    deleteTargetGroup(request_delete_tg);
128         logger.info("Target Group deleted: " + result_delete_tg.
    toString());
    }
130
    /**
132      * Create the target group
     */
134 private TargetGroup createTargetGroup() throws
    NullPointerException {
        CreateTargetGroupRequest request_create_tg = new
    CreateTargetGroupRequest()
136                 .withName(subdomain.toLowerCase())
                .withPort(80)
138                 .withVpcId(vpc_id)
                .withProtocol(ProtocolEnum.HTTP)
140                 .withTargetType(TargetTypeEnum.Instance)
                .withUnhealthyThresholdCount(3)
142                 .withHealthyThresholdCount(3)
                .withHealthCheckTimeoutSeconds(20)
144                 .withHealthCheckIntervalSeconds(30)
```

```
                .withHealthCheckPath("/")
146             .withHealthCheckProtocol(ProtocolEnum.HTTP)
                .withMatcher(new Matcher().withHttpCode("200"));
148
        CreateTargetGroupResult result_create_tg = client_elb.
    createTargetGroup(request_create_tg);
150     logger.info("Target group created: " + result_create_tg.
    toString());
        return result_create_tg.getTargetGroups().get(0);
152 }

154 public TargetGroup getTargetGroupByUser() throws
    NullPointerException {
        DescribeTargetGroupsRequest request_describe_tg = new
    DescribeTargetGroupsRequest()
156             .withNames(subdomain.toLowerCase());

158     DescribeTargetGroupsResult result_describe_tg = client_elb.
    describeTargetGroups(request_describe_tg);
        logger.info("Target group get: " + result_describe_tg.toString
    ());
160     return result_describe_tg.getTargetGroups().get(0);
    }
162
    /**
164  * Get load balancer from environment variable
     */
166 public com.amazonaws.services.elasticloadbalancingv2.model.
    LoadBalancer getLoadBalancer() {
        DescribeLoadBalancersResult result_describe_loader =
    client_elb.describeLoadBalancers(new DescribeLoadBalancersRequest
    ()
168             .withNames(elb_load_balancer));

170     if ((result_describe_loader == null) || (
    result_describe_loader.getLoadBalancers().isEmpty())) {
            logger.error("There isn't any load balancer");
172         return null;
        }
174
        com.amazonaws.services.elasticloadbalancingv2.model.
    LoadBalancer load_balancer = result_describe_loader.
    getLoadBalancers().get(0);
176     logger.info("Load balancer: " + load_balancer.toString());
        return load_balancer;
178 }

180 private String getListenerArn() throws NullPointerException {
        com.amazonaws.services.elasticloadbalancingv2.model.
    LoadBalancer load_balancer = this.getLoadBalancer();
182     DescribeListenersRequest request_describe_listener = new
    DescribeListenersRequest()
                .withLoadBalancerArn(load_balancer.getLoadBalancerArn
    ());
184     DescribeListenersResult result_describe_listener = client_elb.
    describeListeners(request_describe_listener);
```

```
        Listener listener_port_443 = result_describe_listener
                .getListeners()
                .stream()
                .filter(listener -> listener.getPort() == 443)
                .findAny()
                .orElse(null);
        logger.info("Listener: " + listener_port_443.toString());
        return listener_port_443.getListenerArn();
    }

}
```

Clase CloudWatchService

```
public class CloudWatchService extends LambdaService {
 private final Logger logger = LogManager.getLogger(CloudWatchService.
    class);
    private AWSLogs awsLogsClient;
    private AmazonCloudWatch cw_client;

    public CloudWatchService() {
        cw_client = AmazonCloudWatchClientBuilder.standard().
    withRegion(Regions.fromName(region_aws)).build();
        awsLogsClient = AWSLogsClientBuilder.standard().withRegion(
    Regions.fromName(region_aws)).build();
    }

    /**
     * Remove log for subdomain
     */
    public Map<String, String> deleteLog() {
        HashMap<String, String> response = new HashMap<String, String
    >();
        DeleteLogGroupRequest xDeleteLogGroupRequest = new
    DeleteLogGroupRequest().withLogGroupName( subdomain);
        try {
            DeleteLogGroupResult xDeleteLogGroupResult = awsLogsClient
    .deleteLogGroup(xDeleteLogGroupRequest);
            logger.info("Delete logGroup Correct!!. " +
    xDeleteLogGroupResult.toString());
            response.put(SUCCESS, "");
        } catch (Exception e) {
            response.put(ERROR, e.getMessage());
            logger.error("ERROR to delete logGroup. " + e.getMessage()
    );
        }
        return response;
    }

    /**
```

```
     * Add all alarms for a service
     */
    public void addAlarms(TargetGroup tg) {
        try {
            String tgName = tg.getTargetGroupArn().split(":")[5];
            addAlarmErrorLogPerClient("Cantidad de errores por
servicios");
            addAlarmCPUUtilizacionPerClient("Uso de CPU");
            addAlarmReqPerService("Cantidad de Requerimientos por
servicio", tgName);
            logger.info("Alarms created OK! ");
        } catch (Exception e) {
            logger.error("Alarm creation failed, roolback is performed
");
            deleteAlarms();
        }
    }
/**
     * Delete all alarms for a service
     */
    public void deleteAlarms() {
        deleteAlarm("Cantidad de errores por servicios");
        deleteAlarm("Uso de CPU");
        deleteAlarm("Cantidad de Requerimientos por servicio");
        logger.info("Alarms Delete OK! ");
    }

    private void addAlarmErrorLogPerClient(String pAlarmName) {
        Dimension dimension = new Dimension()
                .withName("AUDITORIA_ERRORES_TIEMPOS_AMBIENTES")
                .withValue( subdomain.toLowerCase());

        PutMetricAlarmRequest request = new PutMetricAlarmRequest()
                .withAlarmName(pAlarmName + " - " +  subdomain)
                .withAlarmDescription("Errores encontrados en log del
servicio que contabiliza los mismos")
                .withNamespace("OPTARIS/CLUSTER")
                .withMetricName("ERROR_LOG")
                .withStatistic(Statistic.Sum)
                .withPeriod(300)
                .withDimensions(dimension)
                .withThreshold(30.0)
                .withComparisonOperator(
                        ComparisonOperator.GreaterThanThreshold)
                .withEvaluationPeriods(1)
                .withTreatMissingData("notBreaching")
                .withActionsEnabled(true);

        PutMetricAlarmResult response = cw_client.putMetricAlarm(
request);
        logger.info("addMetricErrorLogPerClient Correct!! :  " +
response.toString());
    }

```

```
78      private void addAlarmReqPerService(String pAlarmName, String
        tgName){
         Dimension dimension = new Dimension()
80                  .withName("TargetGroup")
                    .withValue(tgName);
82
            PutMetricAlarmRequest request = new PutMetricAlarmRequest()
84                  .withAlarmName(pAlarmName + " - " +  subdomain)
                    .withAlarmDescription("Revisar la cantidad de
        requerimientos por servicio, ha superado el umbral definido.")
86                  .withNamespace("AWS/ApplicationELB")
                    .withMetricName("RequestCountPerTarget")
88                  .withStatistic(Statistic.Sum)
                    .withPeriod(300)
90                  .withDimensions(dimension)
                    .withThreshold(250.0)
92                  .withComparisonOperator(ComparisonOperator.
        GreaterThanThreshold)
                    .withEvaluationPeriods(1)
94                  .withTreatMissingData("notBreaching")
                    .withActionsEnabled(true);
96
            PutMetricAlarmResult response = cw_client.putMetricAlarm(
        request);
98          logger.info("addMetricSQLTimeLogPerClient Correct!! :  " +
        response.toString());
        }
100
        /**
102      * Si un ambiente supera el 75% de uso de cpu durante un minuto,
        se activa la alarma
         * @param pAlarmName
104      */
        private void addAlarmCPUUtilizacionPerClient(String pAlarmName) {
106         Dimension dimension = new Dimension()
                    .withName("ClusterName")
108                 .withValue(ecsCluster);
            Dimension dimension2 = new Dimension()
110                 .withName("ServiceName")
                    .withValue( subdomain.toLowerCase());
112
            PutMetricAlarmRequest request = new PutMetricAlarmRequest()
114                 .withAlarmName(pAlarmName + " - " +  subdomain)
                    .withAlarmDescription("Alarma que notifica cuando la
        CPU excede el 75% de uso")
116                 .withNamespace("AWS/ECS")
                    .withMetricName("CPUUtilization")
118                 .withStatistic(Statistic.Average)
                    .withPeriod(60)
120                 .withDimensions(dimension, dimension2)
                    .withThreshold(75.0)
122                 .withComparisonOperator(
                            ComparisonOperator.
        GreaterThanOrEqualToThreshold)
124                 .withEvaluationPeriods(1)
                    .withTreatMissingData("missing")
```

```
126                 .withActionsEnabled(true);

128         PutMetricAlarmResult response = cw_client.putMetricAlarm(
    request);
            logger.info("addUseCPU Correct!! :  " + response.toString());
130     }

132     private void deleteAlarm(String pAlarmName) {
            DeleteAlarmsRequest request = new DeleteAlarmsRequest()
134                 .withAlarmNames(pAlarmName + " - " +  subdomain);
            DeleteAlarmsResult response = cw_client.deleteAlarms(request);
136         logger.info("deleteAlarm " + pAlarmName + " - " +  subdomain +
     " Correct!! :  " + response.toString());
        }
138
}
```

Clase R53Service

```
public class R53Service extends LambdaService{
2  private static final Logger logger = LogManager.getLogger(R53Service.
       class);
       private AmazonRoute53 client_r53;
4
       public R53Service(){
6          client_r53 = AmazonRoute53ClientBuilder.standard().withRegion(
       Regions.fromName(region_aws)).build();
       }
8
       @NotNull
10     public Map<String, String> manageDNS( @NotNull String action,
       ELBService elb_service) {
           LoadBalancer load_balancer = elb_service.getLoadBalancer();
12         Map<String, String> map_dns = new HashMap<>();
           ChangeResourceRecordSetsRequest request_record_set = new
       ChangeResourceRecordSetsRequest()
14                 .withHostedZoneId(dns_hosted_zone_id)
                   .withChangeBatch(
16                         new ChangeBatch()
                                   .withComment("Web server for example.
       com")
18                                 .withChanges(new Change()
                                           .withAction(action)
20                                         .withResourceRecordSet(new
       ResourceRecordSet()
                                                   .withName(subdomain +
       "." + domain)
22                                                 .withType("A")
                                                   .withAliasTarget(
```

```
                                                    new
    AliasTarget().withHostedZoneId(load_balancer.
    getCanonicalHostedZoneId())
                                                                .
    withDNSName(load_balancer.getDNSName())
                                                                .
    withEvaluateTargetHealth(false)))));
        try {
            ChangeResourceRecordSetsResult result_record_set =
    client_r53.changeResourceRecordSets(request_record_set);
            map_dns.put(SUCCESS, ACTION_CREATE.equals(action) ? ("
    https://" + subdomain + "." + domain) : "");
            logger.info(ACTION_CREATE.equals(action) ? "The DNS (Route
     53) has been successfully created" : "The DNS (Route 53) has been
     successfully removed");
        } catch (Exception e) {
            if (ACTION_DELETE.equals(action)) {
                map_dns.put(SUCCESS, "");
            } else {
                map_dns.put(ERROR, e.getMessage());
            }
            logger.error(e.getMessage());
        }
        return map_dns;
    }

    /***
     * Verify that the domain exists
     */
    public Map<String, String> existDomain() {
        HashMap<String, String> map_database = new HashMap<>();
        String pDomain = subdomain + ".";
        try {
            ListResourceRecordSetsRequest
    xListResourceRecordSetsRequest = new ListResourceRecordSetsRequest
    ().withHostedZoneId(dns_hosted_zone_id).withStartRecordName(
    pDomain);
            ListResourceRecordSetsResult xListResourceRecordSetsResult
     = client_r53.listResourceRecordSets(
    xListResourceRecordSetsRequest);
            List<ResourceRecordSet> xResult =
    xListResourceRecordSetsResult.getResourceRecordSets();
            if (xResult != null && !xResult.isEmpty()) {
                //Check
                map_database.put(SUCCESS, "" + pDomain.
    equalsIgnoreCase(xResult.get(0).getName()));
            } else {
                map_database.put(SUCCESS, "false");
            }
        } catch (Exception e) {
            map_database.put(ERROR, e.getMessage());
        }
        return map_database;
    }
}
```

Clase S3Service

```
public class S3Service extends LambdaService {
    private AmazonS3 s3_client;

    public S3Service () {
        s3_client = AmazonS3ClientBuilder.standard()
                .withRegion(Regions.fromName(region_aws))
                .build();
    }

    /**
     * Get object from bucket
     */
    public S3Object getObject(String bucket, String pathToFile) throws
     SdkClientException {
        return s3_client.getObject(bucket, pathToFile);
    }
}
```