



APLICACIÓN WEB PROTOTIPO PARA LA EVALUACIÓN DE
COSTOS Y APROVISIONAMIENTO CON LIBCLOUD SOBRE
AMAZON WEB SERVICES

PRICECLOUD

TRABAJO DE GRADO

SEBASTIÁN AFANADOR FONTAL

sebastian.afanador@correounivalle.edu.co

Código 1629587

JOHN ALEXANDER SANABRIA ORDÓÑEZ PH.D

john.sanabria@correounivalle.edu.co

Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Universidad del Valle
Cali - Colombia

Noviembre 2023 – versión 1.7.1

La Cloud Pricing API¹ de *Infracost* es una API basada en *GraphQL* que incluye datos públicos de precios de *GCP AWS* y *Microsoft Azure*. *LibCloud* crea una capa de abstracción a partir de una *Application Programming Interface (API)* unificada usando controladores para cada tipo de servicio de los *CCSPs*.

RESUMEN

La aplicación web Priceloud integra la fase de evaluación de costos de los principales proveedores de servicios de *Cloud Computing (CC)* junto con su despliegue en una única interfaz para dar un criterio informado que ayude a minimizar los costos de funcionamiento de un proyecto de software con respecto a sus necesidades iniciales.

Usando una arquitectura de microservicios este proyecto recupera los costos de los principales *CCSP* mediante la *Cloud Pricing API* de *Infracost* y un proceso de actualización automático de la base de datos ejecutado semanalmente.

INTRODUCCIÓN

El **Cloud Computing** ha facilitado el despliegue de todo tipo de aplicaciones con un uso adaptable de recursos, esto es un punto importante del cloud, permitirle al cliente iniciar con recursos de base que luego pueden ser escalados a la medida de las necesidades que la aplicación requiera. En contraposición de una infraestructura *On-premise* las tecnologías cloud se adaptan en tiempo real a los requerimientos cambiantes de la demanda de estas aplicaciones y le permiten al cliente invertir recursos vitales en procesos mas relevantes eliminando los costos de mantenimiento e infraestructura tecnológica.

El aprovisionamiento de recursos en proveedores de servicios cloud es un proceso que cada vez toma mas relevancia en aplicaciones que han apostado por la tendencia a migrarse al **CC**. Sin embargo a la hora de elegir estos recursos se deben tener en cuenta aspectos como la arquitectura del proyecto, el tipo de recursos, su ubicación geográfica, las políticas donde residirán, la reputación del proveedor y por supuesto de su costo.

Uno de los aspectos clave para ayudar a garantizar la continuidad de un proyecto de software son precisamente sus costos fijos; el cloud los discretiza con el modelo *Pay as You Go* en términos de peticiones, duración, poder de cómputo, espacio de almacenamiento, usuarios, y demás variables que le agregan aún mas complejidad a la decisión de escoger el proveedor adecuado para los recursos requeridos por el proyecto.

Priceloud permite interactuar sobre una interfaz web donde el usuario puede tomar una decisión informada a partir del conocimiento de los costos de su configuración de recursos sobre varios **CCSP** y de manera complementaria es asistido en su aprovisionamiento sobre **AWS**.

*El **CC** es una tendencia de servicios y recursos que funcionan sobre Internet para facilitar el despliegue, uso y disponibilidad de aplicaciones de Software.*

*En una configuración *On-premise* el cliente se encarga de hospedar sus aplicaciones en uno o varios servidores en su propia infraestructura.*

*El modelo comercial *Pay as You Go* permite al cliente pagar solo por lo que ha usado después de haberlo usado.*

2

PLANTEAMIENTO DEL PROBLEMA

Según lo mencionado anteriormente esta propuesta de trabajo de grado busca responder las siguientes preguntas. ¿Cómo dar un criterio informado respecto a los costos de aprovisionamiento de una aplicación facilitando su despliegue?, ¿Cómo construir un servicio que brinde información actualizada de los costos de los proveedores?, ¿Cómo proponer un proyecto colaborativo que sea extensible a nuevos proveedores y servicios?, ¿Qué tecnologías y arquitectura usar para procurar su crecimiento?.

JUSTIFICACIÓN

3.1 JUSTIFICACIÓN ECONÓMICA

Actualmente no existe un proyecto *Open Souce* que permita aprovisionar recursos en [CCSPs](#) e integrar la validación de sus costos; gracias a que Priceloud es un proyecto de código abierto se puede implementar de manera gratuita por cualquier desarrollador para dar un punto de apoyo en este proceso que carece de un marco bien definido.

3.2 JUSTIFICACIÓN ACADÉMICA

Diseñar e implementar esta propuesta puso a prueba las destrezas técnicas y formación del aspirante a grado; Este proyecto de código libre permite tener un punto de partida para que mas interesados puedan contribuir en la solución del problema que parece tener carencia de participación.

El desarrollo de este proyecto requirió conocimientos de Desarrollo de Software, Bases de Datos Relacionales, Aplicaciones Web, Microservicios, Seguridad y Servicios en la Web, Redes de Computadores y Sistemas Operativos.

4

OBJETIVOS

Los controladores de [LibCloud](#) están categorizados en 3 grandes grupos Compute para máquinas virtuales y baremetal, Storage para contenedores de almacenamiento de Objetos como buckets de S3 y Container para plataforma de contenerización (ej: AppEngine)

4.1 OBJETIVO GENERAL

Diseñar, e implementar un prototipo de aplicación web para informar las tarifas de aprovisionamiento de recursos tipo *Compute*, *Storage* y *Container* en el [CC](#) y administrar su despliegue.

4.2 OBJETIVOS ESPECÍFICOS

1. Definir un modelo general de evaluación de costos para el análisis de los servicios en el [CC](#) usando información recuperable y relevante de la Web de sus proveedores.
2. Diseñar e implementar un prototipo basado en microservicios que recopile y almacene periódicamente las tarifas de los principales [CCSPs](#).
3. Diseñar e implementar un prototipo de aplicación web basada en microservicios para acceder a la información de costos de los proveedores y administrar los recursos de un usuario usando tecnologías agnósticas del proveedor.

RESULTADOS DEL TRABAJO

El proyecto cumple los objetivos específicos mencionados anteriormente como se describe a continuación.

1. El primer resultado con respecto al primer objetivo específico es la desarrollo e implementación de un modelo comparativo de costos que se ejecuta sobre la aplicación web. Este modelo es un punto de referencia que mide los servicios agrupados por categoría, costo y ubicación geográfica.
2. El segundo resultado con respecto al segundo objetivo específico es el desarrollo e implementación de dos microservicios encargados de administrar la información relevante de los precios de los [CCSP](#) y la lógica general de la aplicación. los microservicio se encuentran actualmente en línea, funcionan sin interrupciones 24/7 y su documentación está disponible en <https://api.dev.pricecloud.org/docs>.
3. El tercer resultado con respecto al tercer objetivo específico es un prototipo de aplicación web que está desplegada de manera pública en <https://pricecloud.org>, es una interfaz Web que puede ser desplegada por cualquier otro usuario sobre su propia infraestructura, su código fuente al igual que los microservicios mencionados están disponibles en el repositorio del proyecto <https://github.com/sebastianaf/pricecloud>.

Todos los servicios de la aplicación están disponibles y listos para desplegar por cualquier interesado con un script de docker-compose publicados en el repositorio oficial del proyecto. el script se encarga de desplegar cada parte en imágenes de [Docker](#) con Dockerfiles y sus instrucciones para el proceso de building.

La comunicación pública de todos los web services esta encriptada usando el protocolo [HTTPS](#) con certificados SSL/TLS de [Let's Encrypt](#), los datos sensibles almacenados usan el algoritmo de cifrado simétrico de dos vías [AES](#).

6

ALCANCES DE LA PROPUESTA

La aplicación cuenta con una base de 6 contenedores, *api-01(NestJS)*, *api-02(ApolloServer)*, *api-03(Python Flask)*, *ui(NextJS)*, *db-01(PostgreSQL)* y *db-02(PostgreSQL)*.

Priceloud recupera información de precios los principales CCSPs, estos son [AWS](#), [Microsoft Azure](#), [Google Cloud](#).

Priceloud tiene soporte básico de administración para los recursos de tipo *Compute* y *Storage*, a la fecha de publicación de este documento el controlador para operaciones de *Container* se encuentra en etapa experimental y su implementación no fue exitosa. Las dos operaciones listadas se aplicaron para el proveedor [AWS](#), el proyecto de *Python* esta construido con los principios de la programación modular, este proyecto tiene la posibilidad de extender la implementación de mas controladores en el futuro teniendo en cuenta la disponibilidad de de la librería¹ de [LibCloud](#).

La implementación de la interfaz web se construyó en el *framework* [NextJS](#) con la librería de [React](#) y [Material UI](#)², el objetivo principal es servir como una fuente de información mas que proveer una visualización atractiva, el desarrollo estuvo soportado por la plantilla *BloomUI*³ y no tuvo un componente complementario de [UI](#) y [UX](#).

La información recopilada es de uso libre y proporcionada por *Infracost* a través de su *Cloud Pricing API*⁴. La información de precios es actualizada semanalmente y se almacena en una base de datos [PostgreSQL](#).

El servidor principal de la aplicación esta construido en *NestJS* con *TypeScript* y *TypeORM*, el servidor esta desplegado en un contenedor de [Docker](#) y se encuentra disponible en detrás de un proxy inverso de [Nginx Proxy Manager](#) en <https://api.priceloud.org>.

La aplicación está desplegada sobre un servidor *on-premise* formato *miniITX* Levono ThinkCentre M70q Gen2 configurado con Ubuntu Server 22.04 LTS, el servidor cuenta con 12 núcleos de procesamiento, 16GB de memoria RAM y 256GB de almacenamiento en una unidad de estado sólido.

1 https://libcloud.readthedocs.io/en/stable/supported_providers.html

2 <https://mui.com/>

3 <https://bloomui.com/>

4 https://www.infracost.io/docs/cloud_pricing_api/overview/

MARCO REFERENCIAL

En este apartado se recopila información para describir el CC y los modelos de precios que usan los CCSPs para cobrar sus servicios.

7.1 ESTADO DEL ARTE

Actualmente existen recursos de software para la orquestación de recursos en la nube, estos son bien conocidos como [Cloud Resource Orchestration Frameworks \(CROFs\)](#), han surgido como sistemas para gestionar el ciclo de vida de los recursos de múltiples CCSPs y que cada vez exigen mecanismos de orquestación de recursos capaces de tratar con la heterogeneidad subyacente. [8].

7.1.1 Tecnologías de orquestación

LibCloud

Es una librería escrita en Python tiene licencia *Apache 2.0*. Es una de las mas completas para administrar recursos de CC teniendo unicamente carencias en el soporte de [DBaaS](#)¹. Esta librería oculta las diferencias entre las API de los CCSPs y permite administrar diferentes recursos de la nube a través de una API unificada y facil de usar.

Esta librería divide sus funciones en seis principales categorías². La primera permite administrar Servidores en la nube y *Block Storage*, este componente permite ejecutar secuencias de comandos para preparar al servidor recién creado. Otro tipo de recurso administrable es el almacenamiento de objetos en la nube *Object Storage* y la administración de [CDNs](#), Los servicios de balanceadores de carga *Load Balancer*, la API para administración de [DNSs](#) como servicio y los servicios de administración de Contenedores que permiten a los usuarios implementar y administrar contenedores usando software como [Docker](#) con los proveedores que ofrecen una API de [CaaS](#).

¹ <https://medium.com/@anthonyjpshaw>

² <https://libcloud.readthedocs.io/en/stable/index.html>

pkgCloud

Es una librería escrita en *Javascript* para usar sobre *NodeJS*, al igual *LibCloud* esta permite abstraer los diferentes mecanismos para acceder a las diferentes [APIs](#) de los [CCSPs](#) para centralizar su administración, de manera adicional es la única librería con soporte de [DBaaS](#), permite integración con *Rackspace* usando *MySQL* y [Microsoft Azure](#) con el servicio de *Azure Tables*, esta librería tiene una licencia tipo *MIT License* y además es un proyecto de colaboración disponible en *GitHub*³.

Cloudify

Es un [Framework](#) que usa la arquitectura *Multicloud* y el concepto de [Environment as a Service \(EaaS\)](#) usando *Blueprints*. Los recursos y cargas de trabajo pueden ser orquestados y migrados en tiempo real entre diferentes [CCSPs](#), su componente principal *Cloudify Manager* permite administrar los recursos via interfaz web *Cloudify Console*, este [Framework](#) esta licenciado bajo *Apache 2.0 License*, es uno de los [CROFs](#) mas completos gracias a su diseño lógico de [Plug-ins](#) ⁴.

Terraform

Es una herramienta de [Infrastructure as a Codes \(IaaS\)](#) que aprovisiona y administra infraestructuras usando un lenguaje de configuración de alto nivel, *Terraform* puede administrar múltiples proveedores de nube e incluso dependencias entre [CCSPs](#), *Terraform* aprovecha los [CCSPs](#) para proporcionar capacidades de escalado automático con disparadores de umbral en las métricas del sistema recopiladas por los servicios de monitoreo [8, p.14].

7.1.2 Costos de servicio

Amazon Web Services

Este [CCSP](#) tiene una página web en la [Figura 1](#) dedicada para el cálculo de los costos de sus servicios ⁵, la página permite la estimación de los recursos seleccionando uno o varios servicios y estableciendo su configuración para después exportarla en un [Portable Document Format \(PDF\)](#)

³ <https://github.com/pkgcloud/pkgcloud>

⁴ <https://github.com/orgs/cloudify-cosmo/repositories>

⁵ <https://calculator.aws>



Figura 1: Pagina de precios de [AWS](#)

Google Cloud

También conocido como [Google Cloud Platform \(GCP\)](#) este [CCSP](#) tiene un sitio web⁶ con una calculadora de estimación en la [Figura 2](#), también es posible acceder los precios usando una lista de precios dinámica para cada tipo de servicio mostrando su costo de operación por hora e incluso con algunas ofertas gratuitas para la capa mínima de servicios⁷.

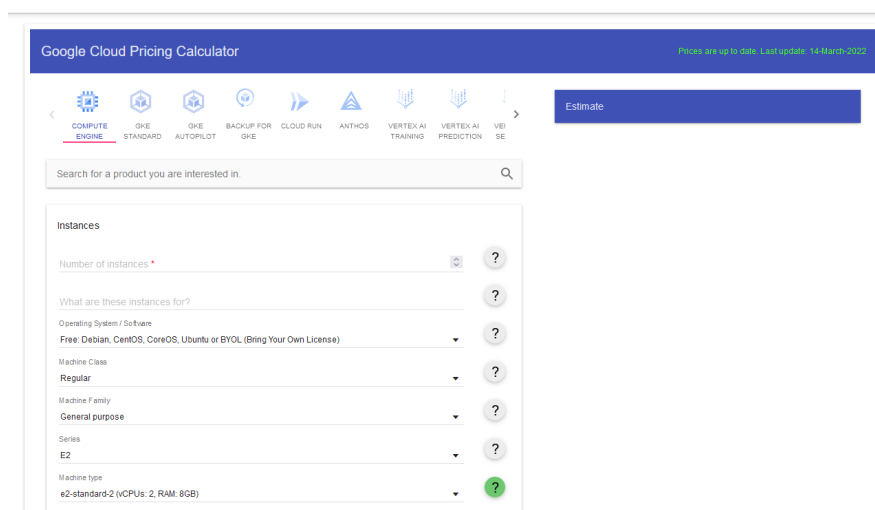


Figura 2: Pagina de precios de [GCP](#)

⁶ <https://cloud.google.com/pricing>

⁷ <https://cloud.google.com/pricing/list>

Microsoft Azure

Al igual que los demás [CCSP Microsoft Azure](#) tiene una calculadora y lista dinámica de precios ⁸ que puede ser consultada en cobro por horas. Véase [Figura 3](#).

SO/software:	Categoría:	Serie de VM:	Región:
SO Windows	Uso general	Todo	Oeste de EE. UU. 2
Moneda:	Precios mostrados por:	Mostrar los precios de la Ventaja híbrida de Azure	
Estados Unidos: dólar (\$) USD	Hora		

Instancia	Núcleos	RAM	Almacenamiento temporal	Pago por uso con AHB	1 años de reserva con AHB	3 años de reserva con AHB	Al contado con AHB	Agregar a estimación
B1s	1	1 GiB	4 GiB	\$0,0104/hora	\$0,0061/hora ~42 % de ahorro	\$0,0040/hora ~62 % de ahorro	--	+
B1ms	1	2 GiB	4 GiB	\$0,0207/hora	\$0,0123/hora ~41 % de ahorro	\$0,0079/hora ~62 % de ahorro	--	+
B2s	2	4 GiB	8 GiB	\$0,0416/hora	\$0,0244/hora ~42 % de ahorro	\$0,0157/hora ~62 % de ahorro	--	+

Figura 3: Lista de precios de [Microsoft Azure](#) para VMs con [OS Windows](#)

Oracle Cloud Infrastructure

Oracle tiene un estimador de costos y lista de precios en su sitio web.⁹ como se muestra en la [Figura 4](#).

Shape	GPUs	Architecture	GPU Interconnect	GPU Memory	CPU Cores	CPU Memory	Network	Price (GPU/hr)
VM.GPU2.1	1x NVIDIA P100	Pascal	N/A	16 GB	12	72 GB	8 Gbps	\$1.275
BM.GPU2.2	2x NVIDIA P100	Pascal	N/A	32 GB	28	192 GB	2x 25 Gbps	\$1.275
VM.GPU3.1	1x NVIDIA V100 Tensor Core	Volta	N/A	16 GB	6	90 GB	400 Mbps	\$2.95
VM.GPU3.2	2x NVIDIA V100 Tensor Core	Volta	NVIDIA NVLINK	32 GB	12	180 GB	800 Gbps	\$2.95
VM.GPU3.4	4x NVIDIA V100 Tensor Core	Volta	NVIDIA NVLINK	64 GB	24	360 GB	25 Gbps	\$2.95

Figura 4: Muestra de precios para instancias de [GPU](#) de [Oracle Cloud](#)

7.2 MARCO TEÓRICO

7.2.1 Cloud Computing

Según el [NIST](#) la computación en la nube está definida como un modelo que permite el acceso de red conveniente y bajo demanda a un grupo compartido de recursos informáticos configurables como

⁸ <https://azure.microsoft.com/es-es/pricing/>

⁹ <https://www.oracle.com/co/cloud/price-list.html>

redes, servidores, almacenamiento, aplicaciones y servicios que pueden aprovisionarse y liberarse rápidamente con un mínimo esfuerzo de gestión o interacción con el CCSP[4].

Modelos de despliegue

En el CC existen varios modelos de despliegue dependiendo de cual es el alcance de los recursos. La *Nube privada* donde los datos y procesos son administrados dentro de la organización quien también se encarga de dar mantenimiento a la infraestructura que reside en las mismas instalaciones o en un lugar remoto administrado por la organización. En la *Nube pública* los recursos son aprovisionados de manera dinámica a través de Internet, el CCSP le permite acceder a los todos los tipos de servicios bajo esquemas de pago por uso. En una *Nube híbrida* los recursos o servicios de la organización pueden ser complementarios entre ambas ubicaciones e incluso pueden incluir infraestructura de *Edge Computing* como dispositivos de IoT [2, p.2].

Tipos de servicios

Los servicios de CC ofrecen mas beneficios que la computación tradicional, ahorro de costos, escalabilidad, almacenamiento móvil, acceso desde cualquier momento o cualquier lugar, mejor seguridad, ahorro de energía[6]. En términos de taxonomía el CC está dividido en tres principales tipos de servicios, *Infrastructure as a Service (IaaS)* que puede verse como una evolución de los *Virtual Private Servers (VPSs)* la cual ofrece plataformas de virtualización donde los clientes deben configurar sus propio software, estos servicios son facturados con base en los recursos consumidos, el modelo de pago por alquiler permite usar hasta mínimo una hora en algunos casos donde la cantidad de instancias se duplican para satisfacer las necesidades los clientes. Entre los tipos de soluciones que se encuentran en esta categoría están *Amazon Elascitic Compute Cloud (EC2)*, *Rackspace Cloud*, *IBM Smart Bussiness Cloud solutions*, *Oracle Cloud Computing*, *Google Compute Engine*. [6, p.2]

Otro esquema de servicios ofrecidos por el CC son las *Platform as a Services (PaaS)*, como características principales tenemos que son proveídas a los clientes por una interfaz en un navegador para editar, depurar, desplegar y monitorear una aplicación.[3, p.1] A diferencia de las *IaaS* permiten una interacción de alto nivel con el cliente porque abstraen la configuración del entorno donde se ejecutan las aplicaciones, además de permitir la preconfiguración de un entorno de ejecución para un lenguaje de programación específico. Entre los CCSPs mas conocidos se encuentra también *EC2* con la posibilidad de preconfigurar los entornos, *Salesforce.com* y *Google App Engine*.

El siguiente nivel en términos de abstracción son las [Function as a Services \(FaaS\)](#) conocidas también como computación *Serverless* [5, p.1] que emerge como un paradigma conveniente para el despliegue de aplicaciones y servicios donde el desarrollador no se preocupa acerca de los aspectos operacionales, de despliegue, y mantenimiento y espera de este que sea tolerante a fallos y auto escalable especialmente el código que es escalable a cero donde no hay servidores corriendo cuando la función del usuario no está siendo usada. [FaaS](#) a diferencia de [Platform as a Services \(PaaS\)](#) evita que el cliente reciba cobros en los periodos de inactividad de la función.[1, p.5] Entre los servicios mas conocidos están *Azure Functions*, [AWS Lambda](#), *Google Cloud Functions* y *Oracle Cloud Functions*.

Por último y mas cerca del usuario se encuentran el [Software as a Service \(SaaS\)](#), en este modelo de servicios el cliente interactúa con una aplicación que da soporte a los procesos de su modelo de negocio o le ofrece un servicio, el cliente paga recurrentemente por las funcionalidades y la cantidad de usuarios que usan el servicio, el [SaaS](#) presenta ventajas frente al software tradicional, el cliente compra una suscripción y no una licencia perpetua, además el software tradicional tarda algunos años en publicar una nueva versión que el [SaaS](#) puede poner a disposición en cuanto esté completa [2, p.1]. Algunos ejemplos de [SaaS](#) son *Google Workspace*, *Microsoft 365*.

7.2.2 Modelos de precios

De manera general existen dos grupos de modelos de precios, los modelos de *precios dinámicos* donde el costo es flexible y depende de factores como el número de peticiones, el espacio usado, el tráfico generado, el poder de cómputo requerido o la cantidad de instancias necesarias. En los modelos de *precios fijos* se dice que el cliente tiene mayor seguridad sobre el costo del servicio porque tiene un cargo fijo periódicamente para pagar, sin embargo el beneficio nunca es real porque depende del uso que tenga el servicio contratado donde algunas veces puede llegar al sobre uso.[7, p.4].

Los [CCSPs](#) usan modelos de precios mixtos que incluyen el esquema de pago por suscripción y el pago por uso, *Pay-as-you-go* que usan las [IaaS](#) y las [PaaS](#).

Existen otros modelos como el *Modelo de pago por suscripción* donde los precios se establecen de acuerdo con el nivel de la suscripción, son comunmente usados por los proveedores de servicios de [Hosting](#). El *modelo de pago basado en costos* permite al [CCSP](#) calcular los precios dinámicos dependiendo de la demanda y las operaciones del los centros de datos del [CCSP](#). Entre los demás modelos de precios dinámicos están los *Modelos genéticos de precios*, *Modelos de precios basados en*

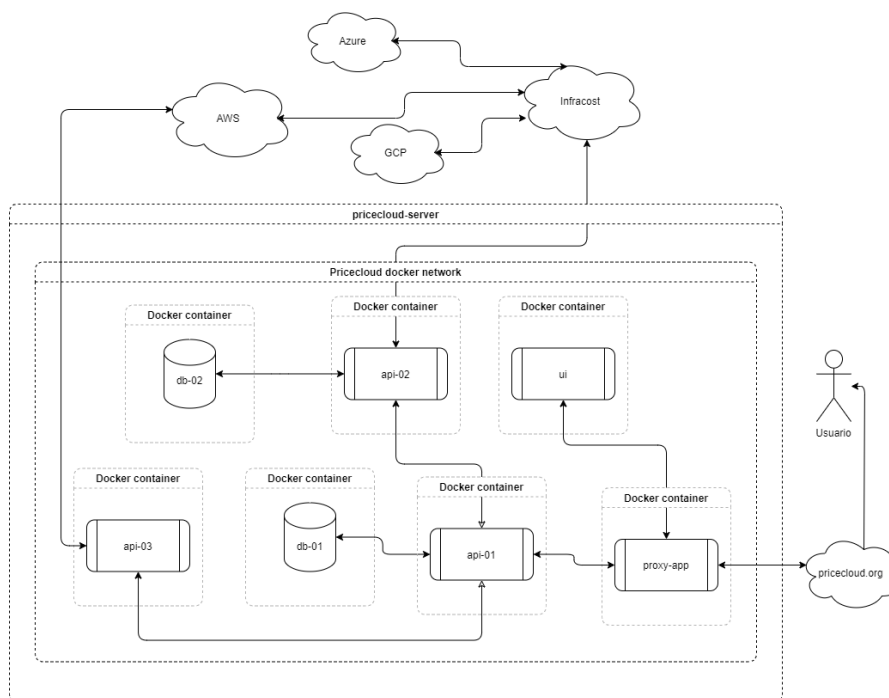
la competencia, Modelos de precios basados en el cliente, Modelos de precios con subasta dinámica y los Modelos de precios híbridos que establecen el precio con la conuinación de los mencionados anteriormente[7, p.7].

8

ARQUITECTURA

A cada uno de los servicios enunciados en este informe le corresponde una carpeta dentro del repositorio del proyecto. El código fuente es de acceso libre y se encuentra en <https://github.com/sebastianaf/pricecloud>.

8.1 ESQUEMA DE SERVICIOS



8.2 DESCRIPCIÓN DE LOS SERVICIOS

La aplicación está compuesta por seis microservicios con funciones específicas cada uno descritas a continuación.

8.2.1 *API REST* api-01

(NestJS) El primer microservicio del backend se encarga de autenticar los usuarios de la aplicación y centralizar todas las peticiones. Este servicio está expuesto en Internet en <https://api.pricecloud.org> y su documentación está en <https://api.dev.pricecloud.org> se comunica usando el protocolo **HTTPS** gracias a que se encuentra detrás del contenedor de **Nginx Proxy Manager**, también está comunicado con *api-03* para el aprovisionamiento de servicios en **AWS**, con *ui* autenticando cada petición con **JSON Web Tokens** cifrados con **Advanced Encryption Standard (AES)**; La responsabilidad de este servicio es responder todas las peticiones creadas por el frontend de la aplicación incluyendo las consultas de los precios de los **Cloud Computing**

[Providers \(CCSPs\)](#) almacenadas en la base de datos *db-02* accedida por *api-02*.

8.2.2 [API REST](#) api-02

(ApolloServer) El segundo microservicio de backend se encarga de recopilar la información de precios de los [CCSPs](#). Este servicio accede a la data de *db-02* donde hay mas de 3 millones de registros de precios de [CCSPs](#), esta parte de la aplicación también tiene la responsabilidad de actualizar la base de datos de precio cuando *api-01* se lo indique, esta actualización es una consulta a los servidores de *Infracost* accediendo a la base de datos libre de precios.

8.2.3 [API REST](#) api-03

(Python Flask) El tercer microservicio de backend se encarga de aprovisionar recursos en [AWS](#) usando [LibCloud](#) para la creación de instancias *EC2* y *S3*.

8.2.4 *Base de datos* db-01

(PostgreSQL) Este microservicio corresponde a la base de datos principal del proyecto, aquí residen los datos de todos los usuarios, sus roles y permisos el esquema de datos de este servidor esta definido en *api-01* por [TypeORM](#).

8.2.5 *Base de datos* db-02

(PostgreSQL) Por motivos de rendimiento se decidió separar la base de datos de precios de los [CCSPs](#) en un servidor aparte, esta base de datos es accedida por *api-02* y se actualiza semanalmente por *api-02*.

8.2.6 *Interfaz Web* ui

(NextJS) Este servicio se encarga de ejecutar el servidor web con la interfaz gráfica de usuario. Implementa un esquema de autenticación por *cookies* con tokens *JWT* encriptados con [AESs](#), este microservicio se comunica con *api-01* para solicitar todas las funcionalidades de la aplicaciones al resto de los contenedores. Esta parte de la aplicación es accesible en

RESUMEN DE ACTIVIDADES

A continuación se muestran los resultados de las actividades para cada actividad, se adjuntan los siguientes videos complementarios para ampliar el panorama del lector.

- Pricecloud - Descripción de infraestructura
https://youtu.be/j7gogj0c_uo
- Pricecloud - Uso de Aplicación
<https://youtu.be/zE3znyRGrjY>

9.1 1ER OBJETIVO ESPECÍFICO.

Las actividades de este objetivo específico tienen como propósito *definir un modelo general de evaluación de costos para el análisis de los servicios en el CC usando información recuperable y relevante de la Web de sus proveedores*. El resultado obtenido son las ecuaciones del modelo de evaluación de costos listas para implementación. A continuación el resumen de cada actividad.

9.1.1 Actividades 1 y 2.

Investigar acerca de los modelos de precios en los CCSPs y comparar los criterios de cobro entre cada CCSP:

Como resultado de esta actividad y teniendo en cuenta las *variables de costo* de cada tipo de servicio en la *sección 2.2.2* se formuló el siguiente criterio.

Dado un servicio bajo demanda s que tienen en común una pareja $ccsp$ donde $ccsp_i$ es un CCSP con $i \in [1, 2]$ se define que:

Compute o Container

Si s es del tipo *Compute* o *Container* la *variable de costo* principal es la demanda definida como el tiempo en que se usará el servicio. Si se conocen los *rangos de costo* para las variables cpu y ram entonces la *función de costo* del servicio es:

$$costo_i(s, ccsp_i, demanda, cpu, ram) = (costoCPU + costoRAM) * demanda$$

Donde:

- $ccsp_i$: Se define como uno de los [Cloud Computing Providers \(CCSPs\)](#) comparados.
- *demanda* : Es el tiempo en horas que se requiere el servicio.
- *cpu* : Es la cantidad de núcleos de procesamiento requeridos.
- *ram* : Es la cantidad de memoria RAM requerida en unidades de GB.
- $costoCPU(s, ccsp_i, cpu)$ Es el costo de la cantidad de *cpu* según el *rango de costo* del $ccsp_i$.
- $costoRAM(s, ccsp_i, ram)$ Es el costo de la cantidad de *ram* según el *rango de costo* del $ccsp_i$.

El puntaje comparativo *score* que tiene el $ccsp_1$ con respecto al $ccsp_2$ es:

$$score = \frac{costo_1(s, ccsp_1, demanda, cpu, ram)}{costo_2(s, ccsp_2, demanda, cpu, ram)}$$

Finalmente, si $score < 1$ entonces se dice que el $ccsp_1$ tiene preferencia sobre $ccsp_2$ para el servicio *s* según un requerimiento de *cpu*, *ram* y *demanda*.

Storage o SQL

Si *s* es del tipo *Storage* o *SQL* la *variable de costo* principal es la cuota de almacenamiento. Si se conocen los *rangos de costo* para la variable de disponibilidad del recurso llamada *disp* entonces la *función de costo* del servicio es:

$$costo_i(s, ccsp_i, cuota, disp) = costoDisponibilidad * cuota$$

Donde:

- $ccsp_i$: Se define como uno de los [Cloud Computing Providers \(CCSPs\)](#) comparados.
- *cuota* : Es el espacio en GB que se requiere.
- *disp* : Es el tipo de disponibilidad del recurso (Ej: Modo frio, alta velocidad).
- $costoDisponibilidad(s, ccsp_i, cuota, disp)$ Es el costo del tamaño de la cuota para una disponibilidad específica según el *rango de costo* del $ccsp_i$.

9.1.2 Actividad 3.

Identificar las variables principales que usan los CCSPs para el cobro de los servicios:

Como conclusión y resultado de esta actividad se definieron las variables principales que usan los CCSPs para el cobro de cada tipo de servicio. Cada variable depende del servicio en cuestión.

En el caso de el servicios tipo *Compute* y *Container* relacionados con máquinas virtuales y contenedores, la variable de costo principal es el *demanda temporal* del recurso en producto con la *cantidad de CPU's* y la *cantidad de memoria RAM*.

Si se habla de un recurso tipo *Storage* y *SQL* la variable de costo principal es la *cuota de almacenamiento* en producto con la *disponibilidad*, este último se relaciona con lo que será almacenado y cómo será recuperado. (Ej: Archivos, objetos o bases de datos. Modo frio o alta velocidad).

9.1.3 Actividad 4.

Efectuar pruebas de escritorio del modelo para 5 CCSPs:

Como resultado de esta actividad se implementó el comparador de costos desplegado en la aplicación web ubicado en la ruta <https://pricecloud.org/dashboard/compare>.

9.2 2DO OBJETIVO ESPECÍFICO.

Las actividades de este objetivo específico tienen como propósito *diseñar e implementar un prototipo basado en microservicios que recopile periodicamente y almacene las tarifas de los principales CCSPs*. El resultado obtenido es un Web service que desempeña las tareas mencionadas. A continuación el resumen de cada actividad.

9.2.1 Actividad 5.

Diseñar el modelo de datos para almacenar la información de costos:

Como resultado de esta actividad se diseñó el esquema de la base de datos usando el [ORM TypeORM](#) sobre [PostgreSQL](#), Los archivos relacionados con cada Entidad de la base de datos se encuentran en <https://github.com/sebastianaf/pricecloud/tree/master/api-01/src/> dentro de las carpetas *entities*.

9.2.2 Actividad 6.

Diseñar un microservicio que use técnicas de Crawling y Scraping para recuperar datos de precios de los CCSPs:

Esta actividad fue parcialmente modificada ya que la técnica de recopilación de los precios de los CCSPs cambio, estos fueron recuperados de la base de datos de *Infracost*, el código fuente de este servicio fue adaptado a partir del proyecto ubicado en <https://github.com/infracost/cloud-pricing-api>, en su momento era código público disponible bajo la licencia *Apache 2.0*. el código del servicio implementado para esta actividad se encuentra en <https://github.com/sebastianaf/pricecloud/tree/master/api-02/> y tiene ligeros cambios en la lógica para ejecutar las operaciones de actualización de la base de datos de precios y demás operaciones de gestión mediante sockets de *Socket.io*.

9.2.3 Actividad 7.

Implementar el modelo de evaluación de costos usando un microservicio:

Como resultado de esta actividad se hizo despliegue de la aplicación de manera pública en <https://pricecloud.org/dashboard/compare>. En esta y otras secciones de la aplicación los usuarios pueden operar entre diferentes servicios de los CCSPs y obtener un cociente comparativo entre ellos. El código fuente de este servicio se encuentra en <https://github.com/sebastianaf/pricecloud/tree/master/api-01/price>

9.2.4 Actividad 8.

Diseñar una configuración de microservicios en Docker Compose:

Como resultado de esta actividad se creó el archivo *docker-compose.yml* ubicado en <https://github.com/sebastianaf/pricecloud/blob/master/docker-compose.yml> donde están configurados todos los servicios de la aplicación.

9.2.5 Actividad 9.

Desplegar la configuración de microservicios en Docker Compose:

Como resultado de esta actividad se desplegaron dos servicios sobre el dominio <https://pricecloud.org> usando *Docker Compose* y *Nginx Proxy Manager* como *proxy inverso* para la encriptación en tránsito y la gestión automática de certificados *TLS/SSL*, toda la aplicación

se encuentra administrada por un entorno de CI/CD usando *Jenkins* y para el despliegue automático de cada cambio en el repositorio.

Los servicios se encuentran desplegados en las siguientes ubicaciones:

- <https://pricecloud.org> ->ui
- <https://dev.pricecloud.org> ->ui (Desarrollo)
- <https://api.pricecloud.org> ->api-01
- <https://api.dev.pricecloud.org> ->api-01 (Desarrollo)

Los demás servicios no están disponibles solo son accedidos a través de *api-01*.

9.3 3ER OBJETIVO ESPECÍFICO.

Las actividades de este objetivo específico tienen como propósito *Diseñar e implementar un prototipo de aplicación web basada en microservicios para acceder a la información de costos de los proveedores y administrar los recursos de un usuario usando tecnologías agnósticas del proveedor..* El resultado obtenido es un Prototipo de aplicación web para administrar recursos de tipo *Compute, Storage* y *Container* sobre *AWS* y obtener información de costos de los principales *CCSPs*. A continuación el resumen de cada actividad.

Actividad 10.

Diseñar el modelo de datos para almacenar la información de la aplicación. El resultado de esta actividad se menciona en la Actividad 5, el modelo de datos de toda la aplicación se encuentra ubicado de manera centralizada en *db-01* y descrito en <https://api.dev.pricecloud.org/docs> su código fuente se encuentra implementado por entidades de *TypeORM* en <https://github.com/sebastianaf/pricecloud/tree/master/api-01/src/> dentro de las carpetas *entities*.

Actividad 11.

Diseñar un microservicio para la administración básica de recursos sobre AWS usando LibCloud. El resultado de esta tarea es el microservicio *api-03* ubicado, esta parte de la aplicación no es accesible de manera pública, solo es accedida por el microservicio *api-01* y *ui* para aprovisionar recursos sobre *AWS* y obtener información de los mismos. El código fuente de este servicio se encuentra en <https://github.com/sebastianaf/pricecloud/tree/master/api-03/>

Actividad 12.

Diseñar a partir de un microservicio una aplicación web usando NodeJS con ReactJS para proveer una interfaz de usuario. El resultado de esta actividad es el microservicio *ui* ubicado que tiene la interfaz de usuario de la aplicación, este servicio es accesible desde <https://pricecloud.org> y <https://dev.pricecloud.org> para el entorno de desarrollo. El código fuente de este servicio se encuentra en <https://github.com/sebastianaf/pricecloud/tree/master/ui/>

Actividad 13.

Diseñar una configuración de microservicios en *Docker Compose*. El resultado de esta actividad es el archivo *docker-compose.yml* ubicado en <https://github.com/sebastianaf/pricecloud/blob/master/docker-compose.yml> donde están configurados todos los servicios de la aplicación, este archivo es el mismo que se usa para el despliegue de la aplicación en <https://pricecloud.org> y <https://dev.pricecloud.org>.

Actividad 14.

Desplegar la configuración de microservicios en *Docker Compose*. El resultado de esta actividad es el despliegue de la aplicación en <https://pricecloud.org> y <https://dev.pricecloud.org> para el entorno de desarrollo usando *Docker Compose* y *Nginx Proxy Manager* como *proxy inverso* de la misma manera como se menciona en la Actividad 9.

- **Logros y Objetivos Alcanzados:** La tesis logró desarrollar con éxito una aplicación web llamada Pricecloud que integra la evaluación de costos y el aprovisionamiento en servicios de Cloud Computing (CC), con especial enfoque en Amazon Web Services (AWS). Se cumplieron los objetivos específicos de diseñar e implementar un modelo comparativo de costos, desarrollar microservicios para la gestión de datos relevantes, y la creación de una interfaz web para facilitar el uso de la herramienta.
- **Contribuciones Técnicas y Prácticas:** Pricecloud aporta una solución innovadora al desafío de evaluar y comparar costos de servicios en la nube, una tarea compleja dada la variabilidad y la gran cantidad de opciones disponibles. La aplicación simplifica este proceso y permite una toma de decisiones más informada y eficiente para los usuarios que buscan optimizar sus recursos en la nube.
- **Desafíos y Aprendizajes:** A lo largo del desarrollo de la tesis, se enfrentaron desafíos técnicos, especialmente en la integración de diversas tecnologías y la orquestación de microservicios. Estos desafíos fueron oportunidades de aprendizaje y mejoraron significativamente la capacidad técnica y de resolución de problemas.
- **Implicaciones para Futuras Investigaciones y Desarrollos:** Esta tesis abre puertas a futuras investigaciones, especialmente en la expansión de Pricecloud para soportar más proveedores de CC y ofrecer funcionalidades adicionales. También se destaca la oportunidad de mejorar la interfaz de usuario y la experiencia del usuario, aspectos que pueden ampliar el alcance y la adopción de la herramienta.
- **Contribución a la Comunidad y Open Source:** La decisión de hacer de Pricecloud un proyecto de código abierto fomenta la colaboración comunitaria y el desarrollo continuo. Esto no solo aumenta el valor y la funcionalidad de la herramienta, sino que también contribuye al conocimiento compartido en el ámbito de la computación en la nube.
- **Reflexiones Finales:** Este proyecto demuestra cómo la combinación de conocimientos técnicos y un enfoque centrado en la solución de problemas reales puede llevar a la creación de herramientas valiosas. Pricecloud es un testimonio de la innovación

y la aplicación práctica de los conocimientos adquiridos en el campo de la ingeniería de sistemas y la computación.

GLOSARIO

Amazon Web Services Es una empresa líder mundial en servicios de [CC](#). [8](#), [26](#)

Cloud Computing Provider Empresa que presta servicios de Computación en la nube. [14](#), [17](#), [26](#)

Cloud Computing El CC es una tendencia de servicios y recursos que funcionan sobre Internet para facilitar el despliegue, uso y disponibilidad de aplicaciones de Software. [ii](#), [1](#), [26](#)

Docker Es una plataforma de contenerización para la implementación microservicios, abstrae recursos como redes, volúmenes de almacenamiento, y nodos de cómputo. [5](#), [6](#), [7](#), [24](#)

Docker Compose Es una tecnología para la automatización y orquestación de aplicaciones a partir de contenedores de [Docker](#). [19](#), [21](#)

Framework Conjunto de tecnologías que trabajan juntas para darle desarrollo a un proyecto que tiene objetivos específicos. [8](#)

Google Cloud Es una empresa de [CC](#) fundada por Google en el 2008. [6](#)

GraphQL Es un lenguaje de consulta para [APIs](#) que permite recuperar información de manera eficiente y flexible. [ii](#)

Hosting Es un servicio relacionado generalmente con el alojamiento de un sitio web en una empresa que presta el servicio. [12](#)

Hypertext Media Transfer Protocol Secure El protocolo que usa la web para enviar y recibir tráfico encriptado a través de una capa de sockets seguros. [26](#)

Infracost Es un proyecto de estimación de costos para Terraform y Kubernetes que ofrece una [API](#) pública para recuperar información de costos de los principales proveedores de servicios de [CC](#). [ii](#)

Let's Encrypt Es una entidad certificadora gratuita de lucro para expedición de certificados SSL/TLS. [5](#)

LibCloud Es una de las bibliotecas escrita en lenguaje Python más completa hasta el momento para interactuar con los servicios de los [CCSPs](#). [ii](#), [4](#), [6](#), [15](#), [20](#)

Material UI Es una librería de componentes que implementa el sistema de diseño de Material Design de *Google*. [6](#)

Microsoft Azure Es una plataforma de computación en la nube que ofrece servicios de [CC](#). [ii](#), [6](#), [8](#), [10](#)

NextJS Es un framework de React usado por grandes empresas para el desarrollo de aplicaciones web tipo [SSR](#). [6](#)

Nginx Proxy Manager Es un proxy inverso administrado por una interfaz Web sobre NodeJS y con las ventajas de Nginx. [6](#), [14](#), [19](#), [21](#)

Plug-in Son complementos a las funcionalidades de base que tiene una aplicación de software, pueden ser incorporados por el usuario a discreción. [8](#)

PostgreSQL Es un sistema de gestión de bases de datos relacionales de código abierto y gratuito conocido por su confiabilidad y robustez, este simplifica algunas labores de consultas y compatibilidad entre servidores de bases de datos. [6](#), [18](#)

React Es una biblioteca de Javascript para la construcción de aplicaciones web reactivas. [6](#)

Representational State Transfer API Es una interfaz de programación de aplicaciones que permite la comunicación a través de la web usando el protocolo HTTP. [26](#)

TypeORM Es un ORM agnóstico del proveedor para NodeJS que permite la conexión a bases de datos relacionales. [15](#), [18](#), [20](#)

User Interface Referente al diseño que tienen las interfaces de un proyecto de software o tecnología. [26](#)

AES Advanced Encryption Standard. [5](#), [14](#), [15](#)
API Application Programing Interface. [ii](#), [7](#), [8](#), [24](#)
API REST [Representational State Transfer API](#). [14](#), [15](#)
AWS [Amazon Web Services](#). [ii](#), [1](#), [6](#), [8](#), [9](#), [12](#), [14](#), [15](#), [20](#)

CaaS Container as a Service. [7](#)
CC [Cloud Computing](#). [ii](#), [1](#), [4](#), [7](#), [11](#), [16](#), [24](#)
CCSP [Cloud Computing Provider](#). [ii](#), [1](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [14](#),
[15](#), [16](#), [17](#), [18](#), [19](#), [20](#), [24](#)
CDN Content Delivery Network. [7](#)
CROF Cloud Resource Orchestration Framework. [7](#), [8](#)

DBaaS Data Base as a Service. [7](#), [8](#)
DNS Domain Name Server. [7](#)

EaaS Environment as a Service. [8](#)
EC2 Amazon Elascctic Compute Cloud. [11](#)

FaaS Function as a Service. [12](#)

GCP Google Cloud Platform. [ii](#), [9](#)
GPU Graphic Processor Unit. [10](#)

HTTPS [Hypertext Media Transfer Protocol Secure](#). [5](#), [14](#)

IaaS Infrastructure as a Code. [8](#)
IaaS Infrastructure as a Service. [11](#), [12](#)
IoT Internet of Things. [11](#)

NIST National Institute of Standards and Technology. [10](#)

ORM Object Relational Mapping. [18](#)
OS Operative System. [10](#)

PaaS Platform as a Service. [11](#), [12](#)
PDF Portable Document Format. [8](#)

SaaS Software as a Service. [12](#)
SSR Server Side Rendering. [25](#)

UI [User Interface](#). [6](#)
UX User Experience. [6](#)

VM Virtual Machine. [10](#)
VPS Virtual Private Server. [11](#)

BIBLIOGRAFÍA

- [1] Ioana Baldini et al. «Serverless Computing: Current Trends and Open Problems». En: *Research Advances in Cloud Computing*. Ed. por Sanjay Chaudhary, Gaurav Somani y Rajkumar Buyya. Singapore: Springer Singapore, 2017, págs. 1-20. ISBN: 978-981-10-5026-8. DOI: [10 . 1007 / 978 - 981 - 10 - 5026 - 8 _ 1](https://doi.org/10.1007/978-981-10-5026-8_1). URL: [https : //doi.org/10.1007/978-981-10-5026-8_1](https://doi.org/10.1007/978-981-10-5026-8_1).
- [2] Vidyanand Choudhary. «Software as a service: Implications for investment in software development». En: *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. IEEE. 2007, 209a-209a.
- [3] George Lawton. «Developing software online with platform-as-a-service technology». En: *Computer* 41.6 (2008), págs. 13-15.
- [4] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, Dawn Leaf et al. «NIST cloud computing reference architecture». En: *NIST special publication* 500.2011 (2011), págs. 1-28.
- [5] Theo Lynn, Pierangelo Rosati, Arnaud Lejeune y Vincent Emeakaroha. «A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms». En: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2017, págs. 162-169.
- [6] Ayob Sether. «Cloud computing benefits». En: *Available at SSRN* 2781593 (2016).
- [7] Aishwarya Soni y Muzammil Hasan. «Pricing schemes in cloud computing: a review». En: *International Journal of Advanced Computer Research* 7.29 (2017), pág. 60.
- [8] Orazio Tomarchio, Domenico Calcaterra y Giuseppe Di Modica. «Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks». En: *Journal of Cloud Computing* 9.1 (2020), págs. 1-24.