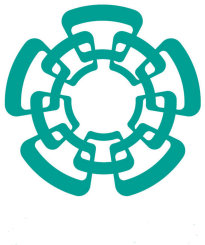


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Cinvestav Tamaulipas

Modelo de construcción de flujos de procesamiento agnósticos para gestionar ciclos de vida de productos de observación de la tierra

Tesis que presenta:

Dante Domizzi Sánchez Gallegos

Para obtener el grado de:

**Maestro en Ciencias
en Ingeniería y Tecnologías
Computacionales**

Dr. José Luis González Compeán, Co-Director
Dr. Victor Jesús Sosa Sosa, Co-Director

Cd. Victoria, Tamaulipas, México. Agosto, 2019

La tesis presentada por Dante Domizzi Sánchez Gallegos fue aprobada por:

_____

Dr. Iván Lopéz Arevalo

Dr. Hiram Galeana Zapién

Dr. José Luis González Compeán, Co-Director

Dr. Victor Jesús Sosa Sosa, Co-Director

Cd. Victoria, Tamaulipas, México., 6 de Agosto de 2019

A mi madre

Agradecimientos

- Agradezco a mis padres y hermanos por todo el apoyo que me han proporcionado.
- A mi madre por siempre apoyarme y creer en mi, alentándome a siempre seguir adelante, por siempre guiarme durante mi vida.
- A Diana, por todo el apoyo brindado durante esta etapa de mi vida, eres un gran motor de apoyo con el que pude salir adelante en momentos difíciles.
- A mis director de tesis, el Dr. José Luis González Compeán, por su dirección y apoyo durante la maestría.
- A mis revisores Dr. Hiram Galeana Zapién y Dr. Iván López Arévalo por sus valiosas enseñanzas durante la revisión de este trabajo.
- Al Dr. Raffaele Montella por su apoyo y dirección durante mi estancia en la Universidad de Napóles Parthenope.
- Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo financiero ofrecido durante los dos años de maestría.
- Al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional por brindarme una formación profesional durante mis estudios de maestría.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Publicaciones

Sánchez-Gallegos, D. D., Gonzalez-Compean, J. L., Sosa-Sosa, V. J., Marin-Castro, H. M., & Tuxpan-Vargas, J. (2018). *An interoperable cloud-based geoportal for discovery and management of earth observation products*. Computer Science & Information Technology, 1.

Sánchez-Gallegos, D. D., Gonzalez-Compean, J. L., Alvarado-Barrientos, S., Sosa-Sosa, V. J., Tuxpan-Vargas, J., & Carretero, J. (2018, July). *A containerized service for clustering and categorization of weather records in the cloud*. In 2018 8th International Conference on Computer Science and Information Technology (CSIT) (pp. 26-31). IEEE.

Sánchez-Gallegos, D. D., Gonzalez-Compean, J. L., Di Luccio, D., & Montella, R., (2019, April). *Internet of Things orchestration using DagOn* workflow engine*. In 4th edition of Globe-IoT 2019: Towards Global Interoperability among IoT Systems (Globe-IoT). IEEE.

Resumen

Modelo de construcción de flujos de procesamiento agnósticos para gestionar ciclos de vida de productos de observación de la tierra

por

Dante Domizzi Sánchez Gallegos
Unidad Cinvestav Tamaulipas
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2019
Dr. Victor Jesús Sosa Sosa, Co-Director
Dr. José Luis González Compeán, Co-Director

En el presente documento de tesis, se describe el diseño, desarrollo e implementación de un modelo de entrega continua basado en grafos acíclicos dirigidos (DAGs, por sus siglas en inglés) para manejar el ciclo de vida de los productos de observación de la tierra (EOPs, por sus siglas en inglés) mediante flujos de procesamiento agnósticos. En este modelo, los nodos representan etapas de un ciclo de vida, las aristas representan rutas de interconexión entre las etapas, y las tareas representan las aplicaciones que serán ejecutados en cada etapa del ciclo de vida. Por tanto, en este modelo, una notación DAG representa una solución para el manejo de un ciclo de vida. Para desplegar una solución en una infraestructura dada, un esquema de acoplamiento encapsula cada nodo del DAG en un contenedor virtual y los interconecta siguiendo las conexiones del DAG. En tiempo de operación, este proceso de acoplamiento permite entrega continua de datos entre las etapas y convierte a un flujo de trabajo tradicional en una solución agnóstica, la cual puede ser ejecutada y operada en diversas plataformas (sistemas operativos) que comúnmente se utilizan en escenarios reales. Los flujos de trabajo agnósticos se incorporan a un ecosistema controlado mediante arquitecturas de microservicios, lo cual permite la construcción automática de múltiples soluciones que incluso pueden reutilizar contenedores (etapas de otros flujos). Esquemas de patrones de paralelismo basados en tareas concurrentes sobre contenedores virtuales fueron agregados al modelo para mejorar la eficiencia de las soluciones creadas por las organizaciones. Una evaluación

experimental basada en estudios de caso de escenarios reales fue conducida para mostrar la factibilidad del modelo propuesto. La evaluación reveló la eficacia del modelo y evidenció la eficiencia de éste en comparación directa con soluciones similares disponibles en el estado del arte.

Abstract

Construction model of agnostic workflows to manage the life cycle of earth observation products

by

Dante Domizzi Sánchez Gallegos
Cinvestav Tamaulipas
Center for Research and Advanced Studies of the National Polytechnic Institute, 2019
Dr. Victor Jesús Sosa Sosa, Co-advisor
Dr. José Luis González Compeán, Co-advisor

In the present Thesis document, it is described the design, development, and implementation of a continuous delivery model based on directed acyclic graphs (DAGs) to manage the life cycle of earth observation products (EOPs) through agnostic workflows. In this model, nodes represent the stages of a life cycle, edges represent the interconnection paths between stages, and tasks represent the apps executed in each stage of the life cycle. Therefore, in this model, a DAG notation represents a solution to manage a life cycle. To deploy a solution in a given infrastructure, a coupling scheme encapsulates each node of the DAG in a virtual container and interconnects them following the connections from the DAG. During operation time, this coupling process produces continuous delivery of data between the stages and turns off a traditional workflow in an agnostic solution, which can be executed and managed on different platforms (operating systems) that commonly used in real-world scenarios. The agnostic workflows are incorporated into an ecosystem managed by microservices architectures, which allow the automatic construction of multiple solutions that even can reuse containers (the stages of another workflow). Schemes of parallel patterns based on concurrent tasks over virtual containers were included in the model to improve the efficiency of solutions created by organizations. The experimental evaluation based on study cases in real-world scenarios was conducted to show the feasibility of the proposed model. The evaluation reveals the effectiveness of this model and shows up the efficiency of this model in direct comparison with similar solutions available in state of the art.

Nomenclatura

API	Interfaz de programación de aplicaciones
DAG	Grafo acíclico dirigido
CD	Entrega continua
EOP	Producto de Observación de la Tierra
REST	Transferencia de Estado Representacional
VC	Contenedor Virtual
WF	Flujo de trabajo
WFE	Motor de flujo de trabajo

1

Introducción

En el presente capítulo se describen los antecedentes, motivación y planteamiento del problema abordado en esta tesis. Además, se presenta la hipótesis y objetivos planteados, así como la metodología de solución que se implementó para lograrlos.

1.1 Antecedentes y motivación

Las agencias espaciales, principalmente la NASA[1], ESA[2], Roscosmos[3] y la CNSA[4] son, generalmente, responsables de llevar a cabo estudios sobre la tierra y el espacio en entidades llamadas misiones. Las actuales agencias espaciales realizan básicamente dos tipos de misiones: las primeras enfocadas en el estudio del espacio (llamadas misiones espaciales) y las segundas enfocadas en el estudio de los fenómenos terrestres (llamadas misiones de observación de la

[1]Administración Nacional de la Aeronáutica y del espacio (NASA, por sus siglas en inglés
[2]Agencia Espacial Europea (ESA)
[3]Agencia Espacial Federal Rusa
[4]Administración Espacial Nacional China (CNSA, por sus siglas en inglés)

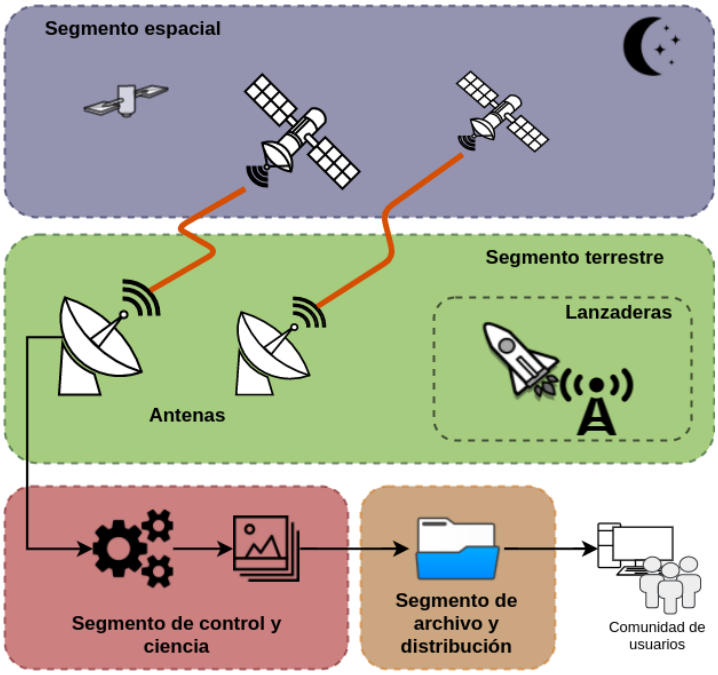


Figura 1.1: Segmentos en los que se dividen las misiones.

tierra).Tal como se muestra en la Figura 1.1, las misiones espaciales se dividen en segmentos tales como espacial/exploración, terrestre/lanzadera, control/ciencia y archivo/distribución [60, 70]. En el segmento espacial se considera la construcción, manejo y mantenimiento de dispositivos para la captura de información (satélites, sondas, exploradores, entre otros dispositivos). Por otra parte, en el segmento terrestre se consideran dispositivos para la captura de señales de los dispositivos espaciales. En este segmento se contempla la instalación, manejo y mantenimiento de antenas y dispositivos requeridos para la alineación con los dispositivos espaciales así como equipos para el lanzamiento de dispositivos espaciales. En el segmento de control/ciencia se procesan los datos capturados en el segmento terrestre y se generan productos de observación del espacio (SOP, por sus siglas en inglés) y/o productos de observación de la tierra (EOPs, por sus siglas en inglés). Finalmente, en el segmento de archivo y distribución se depositan los SOP/EOPs para que investigadores los utilicen como insumos.

Las misiones de observación de la tierra monitorizan a satélites (en el segmento espacial) para capturar información de la tierra (en el segmento terrestre) que produce repositorios de imágenes

satelitales, datos y metadatos. El segmento de archivo y distribución de la NASA maneja, mediante EOSDIS (por sus siglas en inglés *Earth Observation System Data and Information*), más de 9 petabytes (PB) de datos, agregando a su repositorio 6.4 de terabytes (TB) de datos y distribuyendo a un promedio de 11,000 usuarios un aproximado de 28 TB de datos cada día [6].

Los EOPs obtenidos en el segmento de control/ciencia son utilizados para realizar estudios sobre los fenómenos que ocurren en el planeta: cambio climático, deforestación, aumento del nivel de los mares, así como para la administración de los límites territoriales. Ejemplos y aplicaciones de cómo usar los EOPs son variados, encontrando desde estudios sobre el impacto del cambio climático en las granjas de mejillones y los océanos [51, 49], análisis de las tendencias climatológicas a través del tiempo en un país [7, 10, 26], hasta el análisis de imágenes satelitales para monitorear cultivos de arroz [33] o zonas urbanas [71].

Las agencias espaciales, hacen disponible sus EOPs para su consulta mediante servicios Web y geoportales, los cuales pueden incluir herramientas como un motor de búsqueda que permita a los usuarios finales encontrar los productos que se adecúen a sus intereses. Una vez que los usuarios finales han descargado los EOPs de su interés, pueden generar nuevos productos derivados, para obtener mayor información sobre algún aspecto a resaltar en el territorio de cobertura de la imagen. Los EOPs, también pueden ser combinados para crear mosaicos de imágenes, series de tiempo o mapas de cobertura. Algunos ejemplos y aplicaciones de productos derivados son los mapas temáticos utilizados en la petrología, la cuantificación de vegetación, caracterización del suelo, entro otras aplicaciones; los índices obtenidos de las imágenes *raster* usados para estimar la cantidad, calidad y desarrollo de la vegetación, para diferenciar nieve de otros elementos como nubes, para identificar áreas en llamas, y para estimar la radiación fotosintética activa que se consume en la cubierta vegetal.

En la Figura 1.2 se presentan algunas de las relaciones identificadas entre los EOPs y los diferentes componentes de su ciclo de vida. Como se puede observar en la Figura 1.2, los EOPs son consumidos por los usuarios finales (agencias, gobiernos y/o investigadores) a través de geoportales, los cuales manejan utilizan los metadatos recolectados por antenas, indexadores web o extraídos desde bases

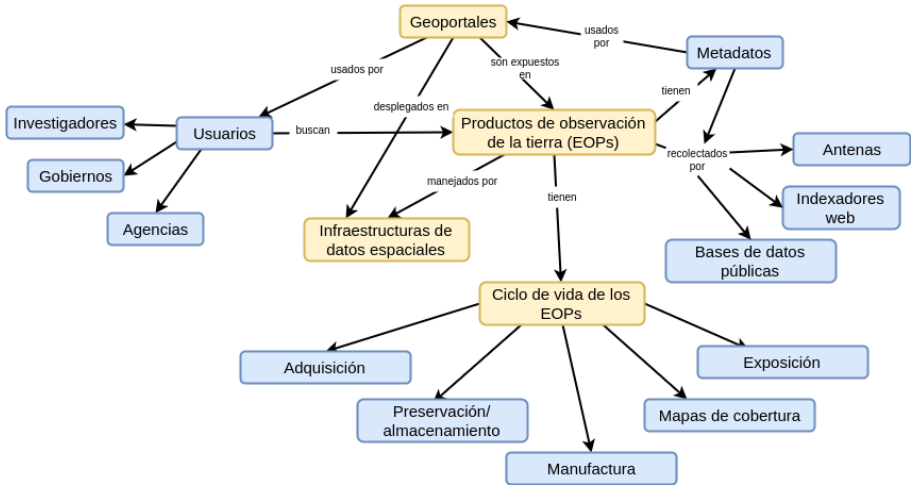


Figura 1.2: Relaciones de los EOPs con otros componentes de su ciclo de vida.

de datos públicas. Para lograr que los EOPs se encuentren disponibles para ser consumidos por los usuarios finales, estos deben de pasar por diferentes etapas para su creación, manejo y transformación.

Dichas etapas se ilustran en la Figura 1.3, donde se pueden observar tanto la adquisición, el transporte y la conservación de EOPs, así como la manufactura y exhibición de los mismos. Existe en la literatura un amplio catálogo de soluciones de software que se han propuesto para cada etapa. Por ejemplo, existen técnicas para procesar imágenes captadas por antenas en etapa de adquisición, modelos de procesamiento para crear productos derivados se encuentran disponibles para la etapa de manufactura, así como técnicas de descubrimiento y visualización de EOPs mediante geoportales han sido ampliamente estudiadas para la etapa de exhibición.

En este contexto existen dos soluciones principales en la literatura para manejar el ciclo de vida de los EOPs: las arquitecturas monolíticas y los flujos de trabajo . Cada una de esta soluciones presenta diferentes problemáticas a las que deben de hacer frente las organizaciones (ver Figura 1.4). Por ejemplo, en las soluciones monolíticas suelen ser difíciles de dar mantenimiento, además de que representan un único punto de falla, mientras que los flujos de trabajo suelen presentar problemas de eficiencia o manejo de las etapas del flujo, por mencionar algunos.

En la literatura, se suelen utilizar soluciones y técnicas de en de entrega continua [14, 16, 67], para

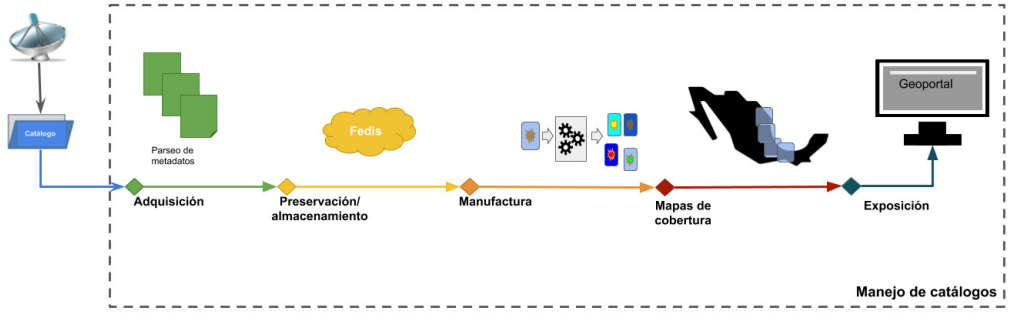


Figura 1.3: Ciclo de vida de los Productos de Observación de la tierra.

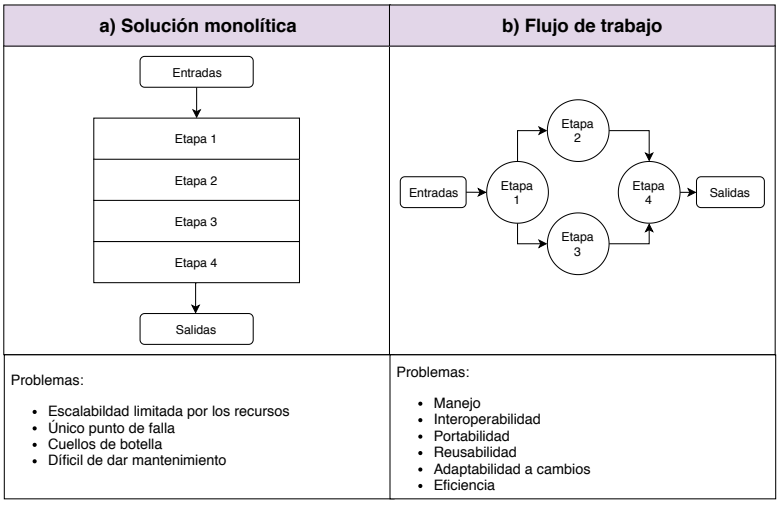


Figura 1.4: Comparación de soluciones monolíticas y flujos de trabajo utilizados para el manejo del ciclo de vida de los EOPs.

el manejar el ciclo de vida de los EOPs. Dichas técnicas, permiten integrar conjuntos de aplicaciones asociadas a diferentes etapas del ciclo de vida mediante esquemas de procesamiento automáticos procesan dichas tareas como flujos de trabajo.

Sin embargo, dichos flujos de trabajo, y las aplicaciones que maneja, suelen ser desarrollados para una infraestructura de hardware y/o software en específico. Lo anterior, complica que una organización pueda reutilizar un flujo de trabajo construido por otra organización para el manejo del ciclo de vida de sus EOPs. Lo anterior debido, a que las aplicaciones utilizadas en cada etapa del flujo de trabajo fueron desarrolladas para trabajar en una infraestructura en específico.

Es por lo anterior, que resulta deseable para las organizaciones el construir flujos de trabajo agnósticos. Se dice que una solución es agnóstica cuando está puede ser utilizada en diferentes tipos de sistemas o plataformas sin que se requiriera que el usuario final realice cambios mayores o ejecute procesos de solución de problemas de despliegue [18]. En los flujos de trabajo se busca que las tareas puedan ser transportadas entre infraestructuras (portabilidad) y que esta pueda seguir funcionando sin tener que volver a configurarla o modificando la aplicación contenida en la tarea. Lo anterior con el objetivo de que los tareas del flujo de trabajo puedan ser reutilizadas, y por lo tanto, poder replicar los resultados obtenidos con un flujo en diferentes infraestructuras.

1.2 Planteamiento del problema

Manejar el ciclo de vida de los EOPs como un flujo de trabajo agnóstico, en términos de la infraestructura y plataforma en la que se despliega, presenta los siguientes problemas: manejo, factibilidad, adaptabilidad, eficiencia, heterogeneidad, portabilidad y reutilización de los componentes que lo conforman. En la Figura 1.5, se clasificaron las problemáticas identificadas en el manejo del ciclo de vida de los EOPs en cuatro niveles: ciclo de vida, etapas, aplicaciones y ambiente de las aplicaciones.

En el nivel más alto se encuentra el ciclo de vida de los EOPs. En este nivel, la problemática radica

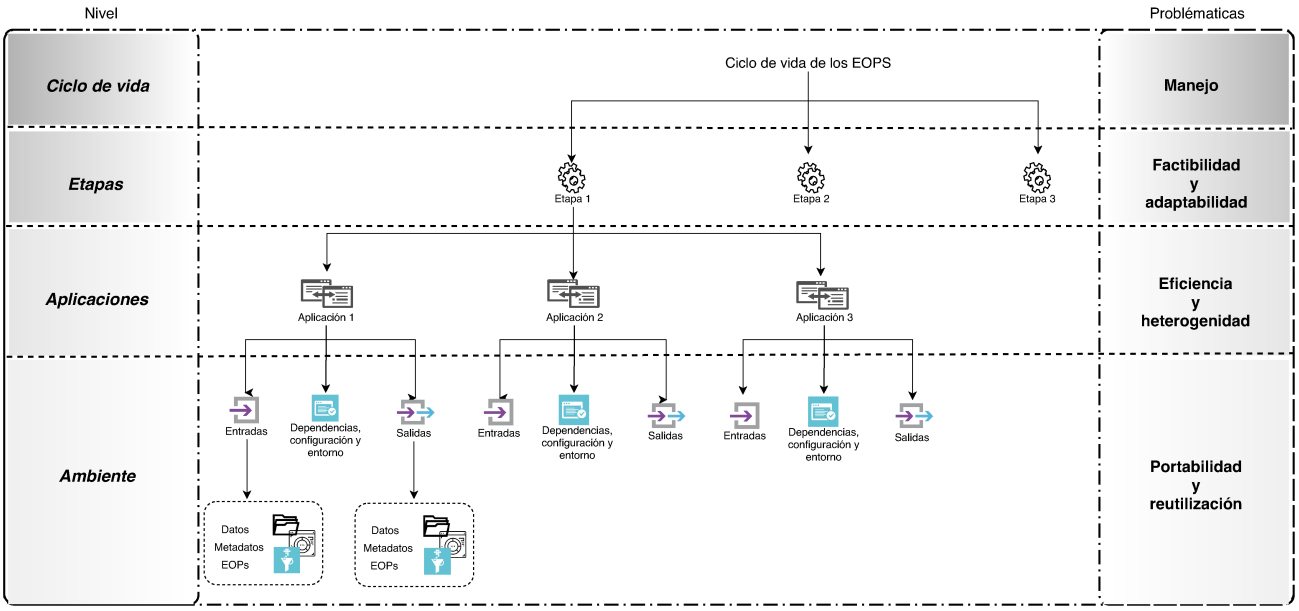


Figura 1.5: Problemáticas identificadas en cada nivel del modelado del ciclo de vida como un flujo de trabajo.

en que las organizaciones terminan manejando una sola solución de flujo de trabajo mediante el uso de las $etapas$ disponibles. En escenarios reales, sin embargo, las organizaciones podrían construir o requerir diferentes tipos de soluciones manejando diferentes tipos y número de etapas del ciclo de vida de los EOPs. Los flujos de trabajo, son una opción para resolver dichas problemáticas, ya que permiten modelar un ciclo de vida de los EOPs como un conjunto de nodos (etapas) interconectados. En la práctica, los motores de construcción de flujos de trabajo sólo consideran fases declarativas y de ejecución. En la fase declarativa, las entradas y salidas de cada etapa y el acoplamiento de la solución completa son descritos. En la etapa de ejecución, el esquema declarado es desplegado para su correspondiente ejecución. Sin embargo, esto significa que el transporte de los datos no está asegurado, los cuales se deben implementar como un servicio adicional para crear automáticamente un esquema de *entrega continua* hasta que culmine el ciclo de vida de un EOP. Adicionalmente, la reutilización de las etapas para hacer parte de otros flujos de trabajo con entrega continua no es un asunto trivial y que generalmente es realizado caso por caso creando soluciones *ad hoc*.

En el nivel de las etapas (segundo nivel), la principal problemática radica en que las organizaciones

deben hacer frente a cambios requeridos por las aplicaciones manejadas por una etapa. En cada etapa las organizaciones tienen un conjunto de aplicaciones que deben de manejar, las cuales varían a través del tiempo. Es por ello que se requieren esquemas donde resulte factible agregar, eliminar y/o reemplazar las aplicaciones de cada etapa, permitiendo a las etapas adaptarse a los cambios de su composición.

En el nivel de aplicaciones (tercer nivel), la principal problemática es la heterogeneidad de éstas, lo cual puede ocasionar problemas de eficiencia debido a los cuellos de botella generados por aplicaciones lentas que se ejecuten en una etapa dada. En la práctica, las agencias u organizaciones terminan manejando un conjunto heterogéneo de aplicaciones cuya integración no es fácil ni automática. Principalmente porque estas soluciones no se encuentran preparadas para ser parte de un sistema o integrarse a otras soluciones.

Finalmente, en la cuarta capa, se tiene el ambiente en el que se despliegan las aplicaciones, el cual incluye sus entradas y salidas, así como sus dependencias, configuraciones y entorno de ejecución. En esta capa se presentan problemas de portabilidad y reutilización de los componentes, ya que no se puede garantizar que la aplicación funcione de manera similar en diferentes infraestructuras, lo cual dificulta que las soluciones puedan ser replicadas por otras organizaciones.

Reproducir resultados de un experimento resulta crucial para los usuarios finales de los flujos de trabajo. Para ello, los usuarios deben de garantizar que las dependencias de la aplicación, así como sus entradas y entorno de ejecución sean los adecuados para obtener como salida los resultados esperados. Sin embargo, lo anterior no resulta trivial, dado que en ocasiones no se cuenta con la información necesaria para reutilizar la aplicación en otra infraestructura.

1.3 Hipótesis

A partir de las problemáticas identificadas en la Sección 1.2, se establece las siguientes premisas:

- Los contenedores virtuales garantizan la portabilidad y correcta ejecución de las aplicaciones.

- Los patrones de paralelismo permiten mejorar la eficiencia de las aplicaciones.
- La arquitectura de microservicios permite manejar la heterogeneidad de las aplicaciones (manejando aplicaciones escritas y/o desarrolladas con múltiples lenguajes de programación), reutilizar aplicaciones para servir a múltiples soluciones y garantiza la adaptabilidad a cambios de las soluciones de las soluciones creadas con este paradigma de construcción.
- Las soluciones agnósticas garantizan ser independientes de la infraestructura en la cual se despliegan y no requieren de la intervención humana para el correcto despliegue de sistemas y/o servicios.

En función de estas premisas se formulan las siguientes preguntas de investigación:

1. ¿Cómo se pueden construir soluciones agnósticas para gestionar los ciclos de vida de los EOPs?
2. ¿Cómo se puede crear un ecosistema de soluciones agnósticas de procesamiento de EOPs como servicios?
3. ¿Qué mecanismos o esquemas se requieren para aplicar en forma factible soluciones agnósticas en escenarios reales de procesamiento de EOPs?

A partir de estas preguntas se establece la siguiente hipótesis:

Los flujos de trabajo cuyos componentes sean encapsulados en contenedores virtuales, desplegados en la forma de patrones de paralelismo y gestionados mediante una arquitectura de microservicios permiten a las organizaciones crear un ecosistema de soluciones integrales y agnósticas con las cuales manejar ciclos de vida de EOPs del mundo real en forma flexible y eficiente.

1.4 Objetivos generales y particulares

General

Obtener un esquema de construcción para modelar ciclos de vida de productos de observación de la

tierra mediante flujos de trabajo agnósticos.

Particulares

1. Desarrollar una arquitectura de microservicios y contenedores virtuales que permita adaptabilidad a cambios, reutilización de componentes de software incluidos en flujos de trabajo, así como la portabilidad de las soluciones.

2. Diseñar un modelo de patrones de paralelismo basados en contenedores virtuales, para mejorar la *eficiencia* de tareas realizadas durante el ciclo de vida de los EOPs.

3. Validar el modelo desarrollado mediante la conducción de estudios de caso basados en escenarios reales de procesamiento de EOPs.

1.5 Metodología

En el presente tema de tesis se modeló el ciclo de vida de los EOPs en una arquitectura de microservicios basada en contenedores virtuales, en la que cada etapa del ciclo de vida sería abstraída como un microservicio. Los microservicios harían factible a las organizaciones el construir múltiples soluciones para diferentes grupos de usuarios, adaptando los flujos de trabajo a los cambios que requieran. Los problemas de heterogeneidad y eficiencia presentes en las etapas del ciclo de vida se hacen frente utilizando contenedores virtuales, los cuales permitirían encapsular las aplicaciones y replicarlas en patrones de paralelismo basados en tareas para mitigar el impacto de los cuellos de botella presentes en la etapa. Además, el uso de contenedores virtuales, resuelven el problema de portabilidad, dado que la aplicación es encapsulada con todas sus dependencias, por lo que puede ser transportada entre infraestructuras, permitiendo que esta sea reutilizada.

Para desarrollar e implementar la solución propuesta en este trabajo de tesis se desarrolló una metodología de trabajo que contempla cinco etapas: la revisión del estado del arte, el modelado de la propuesta descrita, la implementación de un prototipo a partir del modelo desarrollado para hacer

frente a las problemática identificadas en cada uno de los niveles del manejo del ciclo de vida como un flujo de trabajo, la evaluación experimental de la propuesta y la elaboración del documento de tesis, así como de las conclusiones finales.

1. **Revisión del estado del arte.**

   En esta etapa se presenta una recopilación de los trabajos relacionados con la solución propuesta, mediante la recolección y lectura de propuestas encontradas en la literatura. Los temas principales a tratar en esta sección son:

   - Ciclo de vida de los EOPs.
   - Flujos de trabajos.
   - Entrega continua.
   - Arquitecturas de microservicios.

2. **Modelado y diseño.**

   En esta etapa se presenta el diseño conceptual de los modelos que ayudarán a desarrollar la solución propuesta. Dentro de los diseños considerados se tienen los siguientes:

   a) Un esquema que permita modelar el ciclo de vida de los EOPs como flujos de trabajo basados en microservicios y contenedores virtuales, que permita la adaptabilidad a cambios de las soluciones, así como la reutilización y portabilidad de sus componentes.

   b) Un esquema de patrones de paralelismo de tarea para el procesamiento de datos obtenido en alguna de las etapas del ciclo de vida de los EOPs.

3. **Implementación y prototipado.**

   Esta etapa fue dividida en dos:

a) Desarrollo de los componentes de la solución y pruebas de funcionamiento.

Durante esta etapa se desarrollaron los componentes que conforman la solución propuesta, como lo son:

- **Cliente para la generación de flujos de trabajo**: el cliente se encarga de recibir un archivo escrito con la notación basada en el modelo propuesto para la generación de ciclos de vida como flujos de trabajo. Este modulo genera un archivo de configuración de Docker Compose, el cual contiene la descripción de todos los contenedores
- **Sistema de despliegue de los flujos de trabajo**: el sistema de despliegue toma el archivo de configuración creado por el cliente para realizar el despliegue de los contenedores virtuales y los patrones de cada etapa del ciclo de vida construido.
- **Arquitectura de microservicios**: La arquitectura de microservicios permite manejar los flujos de trabajo construidos para cada etapa del ciclo de vida, permitiendo agregar o eliminar componentes de forma flexible.

b) Despliegue del prototipo.

En esta etapa se desplegó el prototipo basado en el modelo propuesto utilizando diferentes configuraciones. Lo anterior con el objetivo de verificar que el prototipo cumple con lo objetivos marcados, y da solución a las problemáticas identificadas (ver Figura 3.8).

4. **Evaluación experimental.**

En esta etapa se analizarón los resultados obtenidos durante la experimentación, así como la realización de pruebas complementarias y su posterior análisis. Esta sección de divide en cuatro partes principales, las cuales se describen a continuación:

a) **Definición de experimentos**. Se definió el ambiente de experimentación, así como las baterías de pruebas a las cuales fue sometido el prototipo desarrollado. Se evaluó la eficiencia y factibilidad de las soluciones mediante las siguientes métricas:

- Tiempos de respuesta y servicio.
- Rendimiento o *throughput*.
- Aceleración de las soluciones.

b) **Experimentación controlada**. En esta etapa se evaluó el rendimiento de un flujo de trabajo para procesamiento de metadatos de imágenes satelitales. Para ello, se varió el volumen de entrada del flujo de trabajo, así como el número de contenedores virtuales en ejecución en paralelo configurado en las tareas en el flujo.

c) **Experimentación basada en estudios de caso**. El prototipo fue evaluado mediante dos estudios de caso, los cuales son:

1) **Estudio de caso I: desde una imagen satelital hasta la generación de mapas**. En este estudio de caso se describe el desarrollo de *GeoEris*, el cual es una plataforma construida utilizando el modelo propuesto. Esta plataforma fue construida para la Agencia Espacial Mexicana (AEM) para el manejo de EOPs capturados y producidos por una antena llamada AEM-ERIS.

2) **Estudio de caso II: servicio para el agrupamiento y clasificación de registros climatológicos**. En este estudio de caso de se realizó un flujo de trabajo para la adquisición, preprocesamiento, procesamiento y exhibición de los datos climatológicos recolectados por las estaciones automáticas de la CONAGUA. Se adquirieron los registros capturados durante 33 años (1985-2018) y se clasificaron utilizando un algoritmo de agrupamiento distribuido ($K - Means$).

d) **Experimentación de prototipos.** A partir de los prototipos desarrollados se realizaron pruebas para el procesamiento de EOPs.

e) **Análisis de resultados.** En esta etapa se llevó a cabo el análisis de los datos recolectados en la experimentación.

5. **Conclusiones.**

En esta etapa se redactaron las conclusiones obtenidas durante el desarrollo del modelo propuesto, así como de los resultados conseguidos durante la etapa de evaluación experimental conducida en la forma de estudios de caso.

1.6 Organización de la tesis

La presente tesis está conformada por ocho capítulos, los cuales se encuentran organizados de la siguiente manera. El Capítulo 1 presenta el contexto de este trabajo de tesis. En el Capítulo 2 se realiza una descripción del fundamento teórico requerido para el desarrollo de esta tesis. En el Capítulo 3 se presenta una revisión del estado del arte respecto a flujos de trabajo y el ciclo de vida de los EOPs. En el Capítulo 4 se describe el modelo propuesto así como sus componentes. En el Capítulo 5 se describe la metodología a seguir para evaluar el prototipo construido. Además, se discuten los resultados obtenidos. En el Capítulo 6 se describe el primer estudio de caso conducido para evaluar el modelo propuesto. El estudio de caso se realizó utilizando las imágenes satelitales captadas por la antena Eris de la Agencia Espacial Mexicana (AEM). En el Capítulo 7 se describe el estudio de caso II, el cual se realizó utilizando los datos de la red de estaciones climatológicas de la CONAGUA (Comisión Nacional del Agua). Finalmente, En el Capítulo 8 se muestran las conclusiones obtenidas durante el desarrollo del presente trabajo de tesis.

2

Marco teórico

En la presente sección se describen los conceptos clave involucrados en el desarrollo del presente trabajo de tesis.

2.1 Sistemas monolíticos

En una arquitectura monolítica todas las tareas son realizadas por un solo programa, el cual se encuentra dividido lógicamente en módulos, los cuales se encuentran altamente acoplados entre ellos, por lo que modificar una pieza del código puede afectar a todo el sistema. Los sistemas monolíticos, son útiles cuando se trabaja en un sistema pequeño a cargo de una sola persona o un equipo de pocas personas, sin embargo en sistemas grandes se presentan problemas como el tener un único punto de falla, cuellos de botella o dificultades en el mantenimiento [72].

En las arquitecturas monolíticas, las soluciones son aglutinadas en un solo servicio (principalmente web [12, 57]) dependiendo de las etapas que una organización le interese integrar. Por ejemplo, integrando soluciones desde la adquisición hasta la manufactura o desde la adquisición hasta el

almacenamiento. Estas soluciones, sin embargo, son rígidas ya que existe un alto acoplamiento entre las etapas que integran el servicio. Esto quiere decir que procesos para reemplazar, agregar o eliminar etapas resultan en cambios mayores a el servicio integrado. Por lo anterior, el adaptar estos servicios a las necesidades de otras organizaciones que requieren manejar diferentes etapas del ciclo de vida de EOPs no es trivial. Las soluciones monolíticas presentan problemas de escalabilidad, la cual es limitada a la infraestructura en la que el aplicación sea instalado. De la misma forma el mantenimiento de dichos sistemas se convierte en un desafío debido a que el código de la aplicación o servicio resultante termina creciendo en relación a las funciones, métodos o módulos agregados tanto a los aplicaciones como a los servicios.

2.2 Arquitectura en capas

En una arquitectura en capas los componentes se encuentran organizados en capas horizontales, donde cada capa tiene un rol específico en la aplicación. Generalmente, este tipo de arquitecturas se compone de tres capas: acceso, procesamiento/manejo y datos. En la primera capa, la de acceso, es el primer punto de interacción entre el usuario y el sistema. Se reciben y validan las peticiones recibidas mediante APIs, formularios y páginas web. Posteriormente, en la capa de procesamiento y manejo se atienden las peticiones autenticadas en la capa anterior. Además, en esta capa se gestiona la estrategia de almacenamiento que se usará en la capa de datos, la cual da persistencia a los datos [63].

2.3 Arquitectura de microservicios

Una arquitectura de microservicios consiste en pequeños servicios autónomos, donde cada uno de estos servicios es auto-contenido y tiene un alcance limitado. Los microservicios se comunican entre ellos, comúnmente mediante HTTP. Cada microservicio es autónomo, desplegado en diferentes

máquinas y exponiéndose mediante una API. Algunas de las ventajas que ofrecen los microservicios son [54]:

- **Heterogeneidad:** Dado que cada microservicio se encuentra aislado del resto, cada uno puede utilizar diferentes tecnologías para su ejecución. De esta manera, cada microservicio puede utilizar las herramientas que mejoren su desempeño sin afectar a los otros componentes del sistema.
- **Resistencia:** Si un componente del sistema falla, y éste no provoca una reacción en cadena, el resto del sistema puede seguir funcionando, dado que cada microservicio se encuentra aislado, por lo tanto el problema también se aísla.
- **Escalabilidad:** En una arquitectura monolítica, para escalar el sistema se tiene que manejar todo como una pieza, lo que complica el escalamiento de un módulo. Mientras que, en una arquitectura de microservicios, se puede lograr la escalabilidad de cada microservicio a diferentes niveles.
- **Fácil despliegue:** Actualizar un sistema monolítico es complicado debido que para actualizarlo se debe de volver a desplegar toda la aplicación para lanzar el cambio realizado. En una arquitectura de microservicios cada componente puede ser desplegado y actualizado independientemente del resto.

2.4 Flujos de trabajo

Un flujo de trabajo es la formalización de un proceso científico que permite el despliegue automático de tareas en una infraestructura de hardware. Comúnmente, los usuarios de los flujos de trabajo deben de seleccionar los datos y realizar algún proceso sobre estos para obtener los datos finales y si es necesario visualizarlos. Es por ello que los sistemas de flujos de trabajo deben de proveer herramientas para el manejo de datos, creación y despliegue de tareas, manejo de relaciones entre

datos y tareas, así como el monitoreo y recuperación de los fallos [76, 77, 78]. A través de un motor de flujos de trabajo, los usuarios deben de ser capaces de seleccionar las tareas y relaciones que conforman el flujo de trabajo. En un sistema de flujos de trabajo, participan diferentes tecnologías heterogéneas, como lo son el lenguaje o herramienta para formalizar el flujo de trabajo, el motor que se encarga de orquestar la ejecución del flujo de trabajo, el sistema donde el flujo es ejecutado y la infraestructura sobre la cual se ejecutan las tareas y se almacenan los datos [76].

Dependiendo la orientación del flujo de trabajo, éstos pueden ser clasificados en dos tipos: orientados a flujos de datos y orientados a flujos de tareas. En el primer caso, los flujos de trabajo son definidos con base en las dependencias entre tareas y el usuario tiene que definir explícitamente las dependencias entre tareas, mientras que en el segundo caso estos son definidos mediante las dependencias entre los datos necesarios por cada tarea, y el motor del flujo de trabajo se encarga de resolver las dependencias entre tareas [50].

2.4.1 Patrones de diseño de flujos de trabajo

Diferentes patrones de diseño pueden ser aplicados para la construcción de flujos de trabajo, siendo los más básicos los que se muestran en la Figura 2.1 [77], a partir de los cuales es posible crear patrones complejos.

- **Secuencial**. Una actividad en el proceso del flujo de trabajo es ejecutada inmediatamente después de que la actividad anterior ha sido completada.
- **Paralelo**. En algún punto del flujo de trabajo se pueden ejecutar diferentes hilos para ejecutar actividades en paralelo.
- **Sincronización**. Se utiliza cuando dos actividades paralelas convergen en una tercera actividad.
- **Elección exclusiva**. Se da cuando se debe de elegir cuál es la siguiente actividad en ser

ejecutada a partir de un conjunto de actividades candidatas. Para ello se implementa un conjunto de condicionales en el flujo de trabajo.

- **Mezcla simple**. Sucede cuando en algún punto del flujo de trabajo dos o más actividades, que no se ejecutan paralelamente, se unen sin sincronización.

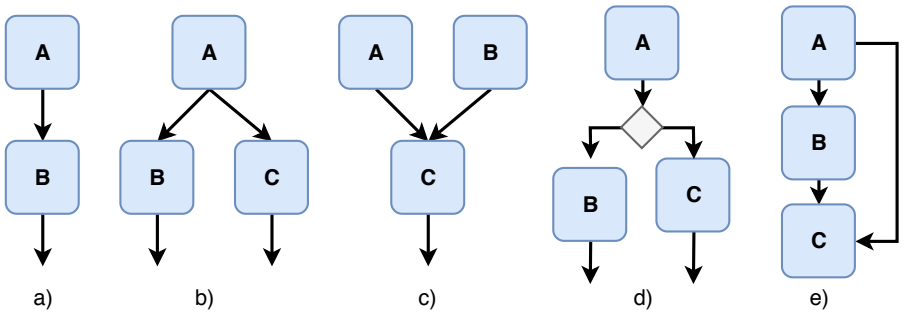


Figura 2.1: Patrones básicos utilizados en un flujo de trabajo. a) Secuencial. b) Paralelo. c) Sincronización. d) Elección exclusiva. e) Mezcla simple.

2.5 Cómputo en la nube

De acuerdo con el Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés), el cómputo en la nube es "un modelo que permite el acceso ubicuo, conveniente y bajo demanda a una piscina de recursos computacionales configurables que pueden ser rápidamente provistas y lanzados con el mínimo esfuerzo de manejo o interacción con el proveedor de servicio" [45]. Además, el NIST define que este modelo tiene cinco características esenciales: autoservicio bajo demanda, amplio acceso a la red, una piscina de recursos heterogéneos, rápida elasticidad y servicio medido. Los modelos de servicio a los que se tiene acceso en el cómputo en la nube son Software como Servicio (SaaS, por sus siglas en inglés), Plataforma como Servicio (PaaS, por sus siglas en inglés) e Infraestructura como Servicio (IaaS, por sus siglas en inglés), los cuales son desplegados en cuatro modelos de nube diferentes: pública, comunitaria, pública e híbrida [35].

2.5.1 Virtualización para la nube

La virtualización juega un papel importante en el cómputo en la nube, dado que permite proveer a los clientes de la nube de almacenamiento virtual, así como de otros recursos computacionales [81]. Básicamente, virtualizar es simular hardware y/o software sobre un equipo físico, creando imágenes que pueden ser utilizadas en diferentes equipos físicos al mismo tiempo, permitiendo la escalabilidad de los recursos de manera virtual, dado que permite particionar de forma lógica un equipo físico [43]. Existen dos tipos de virtualización, los cuales son:

- **Virtualización a nivel hardware.** Implica virtualizar el hardware en un equipo físico y crear máquinas virtuales (VMs, por sus siglas en inglés) que proporcionan la abstracción de una máquina física. Para llevar a cabo la virtualización, se debe de ejecutar un hipervisor, también conocido como monitor de máquina virtual, que emula hardware virtual como la CPU, memoria, periféricos de E/S y dispositivos de red para cada VM. En cada VM se ejecuta un sistema operativo independiente, en donde se ejecutan las aplicaciones [68].

- **Virtualización a nivel de sistema operativo.** También conocida como virtualización basada en contenedores, implica la virtualización solo del núcleo (*kernel*) del sistema operativo en lugar del hardware. Las VM, se denominan contenedores virtuales (VCs, por sus siglas en inglés), y cada VC encapsula un grupo de procesos aislados de otros VCs y procesos del sistema huésped. El núcleo del sistema operativo es responsable de organizar la compartición de los recursos de hardware (memoria, CPU, red, etc) [46, 68].

En la Figura 2.2 se pueden observar los componentes de una máquina virtual (virtualización a nivel de hardware) y un contenedor virtual (virtualización a nivel de sistema operativo). En la Figura 2.2 se puede observar que el número de componentes que se encapsulan en un contenedor virtual es menor en comparación con una máquina virtual, lo que hace que estos sean de menor tamaño, permitiendo que estas puedan ser transportadas y desplegadas en diferentes infraestructuras.

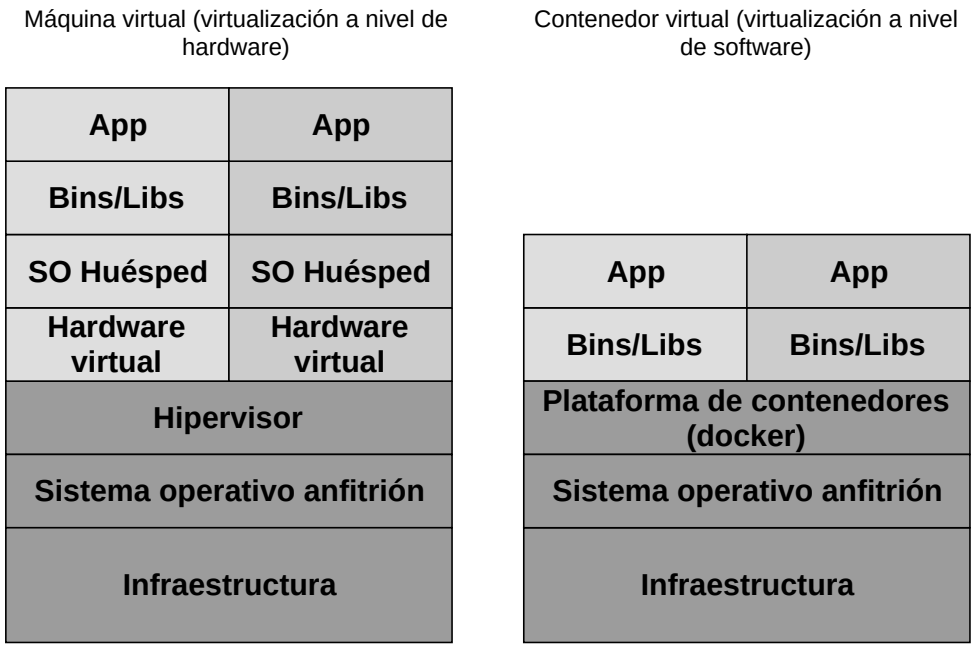


Figura 2.2: Comparación entre una máquina virtual y un contenedor virtual.

2.5.2 Implementaciones para la virtualización basada en contenedores

En la presente sección se describirán *Linux VServer*, *OpenVZ* y *Linux Containers* (LXC), las cuales son algunas de las implementaciones más importantes de virtualización basada en contenedores.

2.5.2.1. Linux-VServer

Linux-VServer es la implementación más antigua del sistema Linux basado en contenedores. En lugar de usar *namespace*[1] (espacios de nombres) para garantizar el aislamiento, Linux-VServer introdujo (a través de un parche) sus propias capacidades en el núcleo de Linux, tales como aislamiento de procesos, aislamiento de redes y aislamiento de CPU [17].

Linux-VServer utiliza la llamada tradicional del sistema *chroot*[2] para bloquear el sistema de

[1] Namespace es un conjunto de nombres en el cual todos los nombres son únicos.

[2] Chroot cambia el directorio raíz del disco aparente a otro directorio raíz. Al cambiar a otro directorio raíz no se puede acceder a los archivos y órdenes fuera de ese directorio.

archivos dentro de los contenedores. De esta forma, limita el alcance del sistema de archivos para los procesos. El aislamiento de los procesos se lleva a cabo a través de un espacio *Identificador de Procesos* (PID) global, que oculta todos los procesos fuera del alcance de un contenedor y prohíbe las comunicaciones no deseadas entre procesos de diferentes contenedores. Los beneficios de este enfoque es la escalabilidad para una gran cantidad de contenedores. Sin embargo, el inconveniente es que el sistema no puede implementar las técnicas de virtualización habituales, como la migración en vivo, puntos de control y reanudación, debido a la imposibilidad de volver a crear instancias de procesos con el mismo PID.

2.5.2.2. OpenVZ

OpenVZ ofrece algunas funcionalidades similares a LinuxVServer. Sin embargo, se basa en los espacios de nombres del kernel, asegurándose de que cada contenedor tenga su propio subconjunto aislado de un recurso. El sistema usa un espacio de nombres PID para garantizar el aislamiento del proceso entre diferentes contenedores. Es así como cada proceso de contenedor tiene sus propios ID de procesos únicos. Además, a diferencia de Linux-VServer, el espacio de nombres PID hace posible el uso de técnicas de virtualización habituales, como la migración en vivo, puntos de control y reanudación [73]. En OpenVZ cada contenedor tiene sus propios segmentos de memoria compartida, semáforos y mensajes, debido a la capacidad del espacio de nombres del núcleo IPC. Además, OpenVZ también usa el espacio de nombres de la red. De esta forma, cada contenedor tiene su propia pila de red. Esto incluye dispositivos de red, tablas de enrutamiento, reglas de firewall, etc.

2.5.2.3. LXC

De la misma forma que OpenVZ, LXC usa espacios de nombres de kernel para proporcionar aislamiento de recursos entre todos los contenedores. Durante el inicio del contenedor, de forma predeterminada, los PID, IPC y puntos de montaje se virtualizan y aíslan a través del espacio de nombres PID, espacio de nombres IPC y espacio de nombres del sistema de archivos, respectivamente.

Para comunicarse con el mundo exterior y permitir el aislamiento de la red, el sistema usa los espacios de nombres de la red. LXC ofrece dos configuraciones para configurar espacios de nombres de red: basados en rutas y basados en puentes. A diferencia de Linux-VServer y OpenVZ, la gestión de recursos sólo está permitida a través de *cgroups*[3]. En la perspectiva de red, *cgroups* define la configuración de espacios de nombres de red. El sistema usa múltiples controladores sobre el programador estándar de la CPU de Linux [41]. El control del proceso se realiza mediante *cgroups*, que tiene la función de limitar el uso de la CPU y aislar contenedores y procesos.

2.6 Docker

Docker[4] es un proyecto de código abierto que provee un sistema para el despliegue automático de aplicaciones dentro contenedores virtuales. Los contenedores Docker, son una extensión de los contenedores Linux (LXC, por sus siglas), en la cual se agrega un API a nivel de núcleo y aplicación, que en conjunción ejecutan procesos de manera aislada en: la CPU, memoria, E/S, etc [5].

Los contenedores de Docker se crean utilizando imágenes como base. Una **imagen de contenedor** es inmutable e incluye todas las dependencias e información necesaria para crear un contenedor. Incluye todas las dependencias (como sistema operativo, librerías y *frameworks*), así como las configuraciones de despliegue y ejecución utilizadas por el contenedor en tiempo de ejecución. Usualmente, las imágenes de contenedor se crean a partir de múltiples imágenes bases, que son capas apiladas una sobre otra, para formar el sistema de ficheros del contenedor [19]. Cada comando ejecutado dentro del contenedor crea una nueva capa en la imagen. Los comandos en la imagen pueden ser ejecutados manualmente desde la línea de comandos o automáticamente, utilizando un archivo **Dockerfile** [5].

Un archivo Dockerfile contiene las instrucciones para construir una imagen de contenedor. La primera línea del archivo establece la imagen base de la nueva imagen. Posteriormente se colocan

[3] Cgroups es una herramienta de Linux para controlar la asignación de recursos a los procesos.
[4] www.docker.com

las instrucciones requeridas para instalar programas, copiar archivos, definir variables de entorno, volúmenes, entro otros requerimientos para configurar el entorno de trabajo de la imagen [20].

Un *contenedor* es una instancia de una imagen de contenedor. Un contenedor ejecuta una aplicación, proceso o servicio. Está conformado por el contenido de una imagen de contenedor, un entorno de ejecución y un conjunto estándar de instrucciones [19].

2.6.1 Docker Compose

Docker Compose[5] es una herramienta para definir y ejecutar aplicaciones multicontenedor utilizando archivos YAML[6] (por su acrónimo recursivo en inglés YAML Ain't Markup Language o en castellano "YAML no es un lenguaje de marcado"). Con Docker Compose, una sola aplicación es definida utilizando múltiples imágenes de contenedor (con uno o más archivos YML). Cuando se han definido los servicios que conforman la aplicación, está puede ser desplegada con un solo comando (`docker-compose up`), la cual crea un contenedor para cada servicio definido en el archivo YML [19, 20]. En la Figura 2.3 se muestra una representación conceptual del flujo de trabajo que se sigue para desplegar servicios con Docker Compose a partir de un archivo YML.

2.6.2 Docker Swarm

Docker Swarm es una herramienta que permite desplegar de contenedores Docker en un clúster de computadores, mediante la consola interactiva de Docker. Docker Swarm tiene las siguientes características:

- Administración y orquestación de contenedores utilizando Docker.
- Diseño descentralizado.

[5] https://docs.docker.com/compose/
[6] YAML es un formato de serialización de datos legible por humanos.

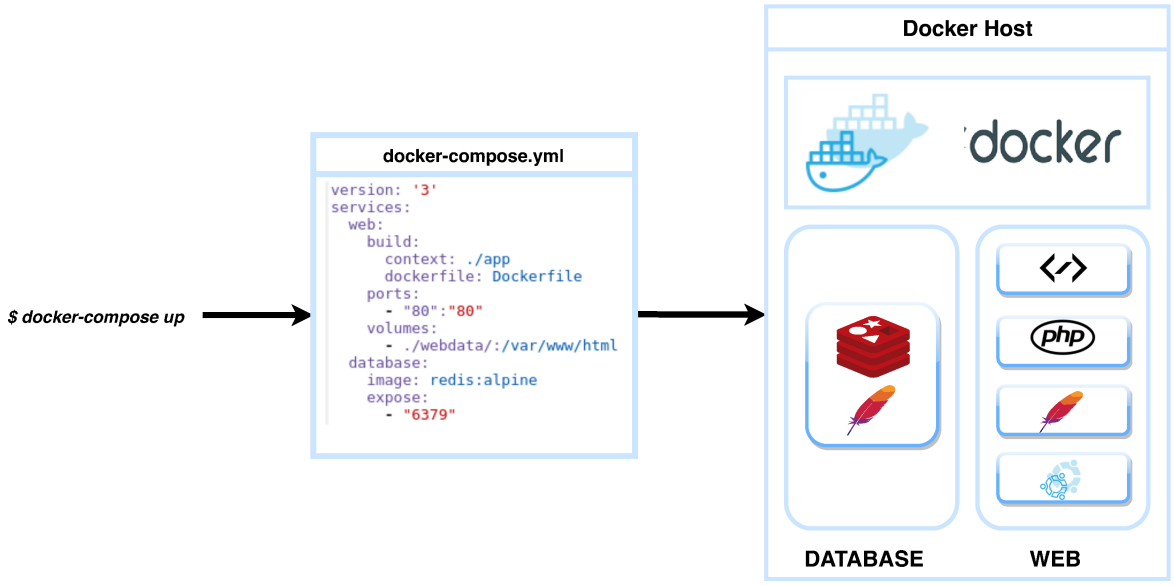


Figura 2.3: Despliegue de servicios con Docker Compose.

- Swarm permite escalar las aplicaciones, declarando el número de tareas que se desean ejecutar a partir de un contenedor.
- El manejador de Swarm constantemente monitores el estado del clúster para garantizar que estado de los contenedores lanzados sea el mismo que el deseado durante su lanzamiento.
- Descubrimiento de servicios.
- Balanceo de carga.

2.7 Resumen

En el presente capítulo se presentaron los conceptos relacionados con la presente tesis. Para ello se presentaron conceptos para manejar el ciclo de vida de los EOPs, tales como los sistemas monolíticos y los flujos de trabajo. Además, se describieron conceptos como la virtualización basada en contenedores virtuales, la cual es utilizada en el presente tema de tesis para encapsular aplicaciones y proveerlas con características de portabilidad e inmutabilidad. Finalmente, se presentó una descripción

de Docker, el cual es la herramienta utilizada en la presente tesis para el manejo de los contenedores virtuales.

3

Estado del arte

En la presente sección se presentan diferentes trabajos enfocados con cada una de las fases del ciclo de vida de los EOPs y flujos de trabajo. El principal propósito de este repaso del estado del arte es el de conocer cómo se han solucionado problemas para cada fase del ciclo de vida de los EOPs, y cómo estas pueden ser unidas como un flujo de trabajo, el cual, a partir de una fuente de datos, pueda desempeñar una tarea o más tareas por cada fase, exponiendo los resultados a través de una plataforma de visualización, como lo pueden ser un geoportal.

3.1 Ciclo de vida de los productos de observación de la tierra

En esta sección se describen trabajos relacionados con cada una de las etapas del ciclo de vida de los productos de observación de la tierra, con el objetivo de conocer soluciones existentes en la literatura para cada etapa.

3.1.1 Adquisición

En esta etapa los EOPs son adquiridos, por antenas ubicadas en el segmento terrestre. En el segmento de control se les agrega formato y se realizan correcciones para que puedan ser utilizadas por terceros (investigadores). Una vez los EOPs tienen al menos un formato de primer procesamiento (L1), son adquiridas en el segmento de archivo en donde se manejan como repositorios incluyendo los siguientes tipos de información:

- **Productos:** Son principalmente, aunque no están restringidos a estos, imágenes satelitales y los productos derivados que se pueden generar a partir de estas. Las imágenes, varían en resolución dependiendo el satélite con el que hayan sido capturadas, por lo que mientras que en algunas es posible observar países o continentes completos, en otras solo será visible una región en un país.
- **Datos:** Son todos aquellos datos espaciales y temporales obtenidos de la medición u observación de un fenómeno que ocurre en la tierra, como puede ser la temperatura de una zona, la cantidad de precipitación, el nivel de los mares, el crecimiento de las zonas urbanas, entre otros.
- **Metadatos:** Son aquellos datos que describen propiedades de los EOPs, como puede ser su ubicación, satélite o instrumento con el que fueron capturados, fecha y hora de adquisición, entre otros. Son utilizados para indexar los EOPs en motores de búsqueda y así facilitar su localización.

Existen diferentes repositorios públicos, a los que es posible acceder para descargar estos datos, sin embargo, estos datos no se encuentran centralizados en un solo repositorio, complicando la tarea de los investigadores de poder encontrar y obtener los datos que estos requieren. En este sentido, Steininger *et al.*[71], realizan la extracción de los datos directamente de bases de datos geoespaciales, concentrándose en un solo repositorio enfocado en la investigación de entornos urbanos, al cual los

investigadores pueden acceder para descargar los datos, además de poder cargar nuevos datos, que puedan ser de interés para el enfoque del repositorio. Otra solución propuesta en esta etapa es hacer uso de indexadores web que busquen en la web por recursos como EOPs y datos geoespaciales. Por ejemplo, Skluzacek *et al.*[69] implementaron un indexador web para la creación de un repositorio o dato de lagos, en donde se centralizan los datos geoespaciales encontrados a través de la web, con la finalidad de que cualquiera con permiso de acceso pueda consultarlos en cualquier momento que lo requiera.

3.1.2 Preservación

Los productos obtenidos son almacenados para su posterior uso en infraestructuras de datos espaciales (SDI, por sus siglas en inglés). Para esta etapa, tomando en cuenta que los EOPs son considerados como acervo por las misiones y las agencias, comúnmente se considera la aplicación de mecanismos que aseguren ciertos requerimientos no funcionales, tales como: confiabilidad (respecto a la tolerancia a fallos), y seguridad (englobando tanto integridad como control de autenticación). En este contexto, Molina y Bayarri [48] proponen un SDI para el manejo de los productos de una manera descentralizada, es decir mediante una arquitectura de nodos distribuidos, en donde una primer capa de nodos facilitadores proveen las herramientas para buscar y visualizar los productos almacenados en los nodos servidor. Sin embargo, esta solución requiere de infraestructura a las en ocasiones las instituciones no tienen acceso. Es por ello, que las soluciones en la nube han sido propuestas para el manejo de la preservación y transporte de los EOPs, proveyendo mecanismos para que el transporte se haga de una manera eficiente. Por ejemplo, González *et al.*[31] proponen un conjunto de componentes basados en la nube para la construcción de plataformas geoespaciales, llamado FedIDS. Con estos componentes, las organizaciones pueden colaborar entre sí para formar una federación, mediante la compartición de recursos, asegurando la integridad y confidencialidad de los productos durante su transporte y preservación/almacenamiento.

3.1.3 Manufactura de productos derivados

En esta etapa se encuentra el software necesario para generar productos derivados a partir de los EOPs. Además, se ubican tareas tales como:

- **Preprocesamiento de productos:** Este tipo de preprocesamiento se realiza sobre imágenes satelitales (tipo *raster*) de la superficie de la tierra para realizar correcciones que resulten en un nuevo producto, que pueda ser usado posteriormente. Ejemplos de este tipo de preprocesamiento, son las correcciones radiométricas y atmosféricas de las imágenes satelitales. Las primeras, tratan con los fallos en los sensores, los cuales generan píxeles incorrectos en la imagen. Por su parte, las correcciones atmosféricas tratan con el ruido causado por la interferencia de la atmósfera al captar la imagen.

- **Preprocesamiento de datos:** En el preprocesamiento de datos se realiza sobre los datos producidos en las diferentes etapas del ciclo de vida de los EOPs, y tiene como finalidad preparar los datos para que puedan ser utilizados por los usuarios finales/investigadores en algún estudio o como entrada para otras tareas del ciclo de vida. Dentro de este preprocesamiento, se consideran tareas tales como la eliminación de valores y variables que no serán de utilidad en el estudio, el rellenado de valores faltantes y la transformación de datos de un formato a otro.

- **Procesamiento de productos (incremental/decremental):** El procesamiento de productos se presenta de dos maneras: decremental e incremental. El procesamiento de productos decremental consiste en la generación de nuevos productos a partir de algunos componentes y características del producto original. Por ejemplo, mediante la combinación de algunas bandas espectrales[1] de las imágenes satelitales. Por otro lado, en el procesamiento de datos incremental se busca añadir nuevas cualidades a un conjunto de productos (por ejemplo,

[1] "Una banda espectral es una matriz de puntos definida por tres dimensiones, sus coordenadas y la intensidad relacionada con el resplandor." [55]

imágenes satelitales) para la generación un nuevo producto, por ejemplo, mapas temáticos y/o mapas de cobertura.

- **Procesamiento de datos (incremental):** El procesamiento de datos se realiza de manera decremental, debido a que a partir de un conjunto de datos, se busca obtener como resultado información que será de menor tamaño al conjunto de datos original. La información puede ser entregada como gráficas, tendencias, estadísticos de los datos (media, mediana, moda, desviación estándar, etc.), correlaciones, o un modelo que describa los datos.

Por ejemplo, Zhang *et al.*[82] proponen un mecanismo para la creación automática de productos a partir de imágenes satelitales capturadas con el satélite ZY-3. Dichos productos son utilizados como recursos de investigación, monitoreo ambiental y aplicaciones de mapeo. Otros de los productos que es posible generar a partir de las imágenes satelitales y sus subproductos, son los mapas de cobertura de la tierra, los cuales ayudan a los investigadores a encontrar patrones a escala global, por ejemplo, en cambio climático, topografía u otras características físicas de la tierra [24].

3.1.4 Exposición de productos a los usuarios finales

En esta etapa, los productos, datos y metadatos obtenidos en las anteriores fases del ciclo de vida de los EOPs son exhibidos utilizando geoportales [2]. En un geoportal, es posible exhibir al usuario final dos tipos de productos: los efímeros, que son producidos bajo demanda, por ejemplo, mosaicos y solapamientos; y los persistentes, que se encuentran almacenados en el SDI, como las imágenes satelitales y los productos derivados. Los geoportales, proporcionan a los usuarios finales herramientas (como consultas en lenguaje natural, trazado de polígonos, selectores de fecha, entre otros) para que puedan consultar los productos disponibles en la base de datos de la plataforma [4, 48, 75]. Por ejemplo, en el geoportal presentado por Molina *et al.*[48] se incluye un motor de búsqueda para que

[2] Un geoportal es un tipo de portal web utilizado para la consulta y acceso a información geográfica (información geoespacial) [9].

los usuarios puedan consultar los productos mediante palabras clave o expresiones, además, de que el geoportal presenta un serie de preguntas guía para ayudar a los usuarios con su búsquedas. Otra forma de realizar las búsquedas, es como la presentada por Aditya y Kraak [1], en el cual mediante RDFs se logra mejorar la indexación y búsqueda de contenidos en un geoportal.

Dependiendo el enfoque que se le quiera dar al geoportal, estos pueden ser multipropósito o estar enfocados en un tema de investigación de interés para un grupo científico. Por ejemplo, Granell *et al.*[33] proponen el desarrollo de un geoportal para el monitoreo de los campos de arroz en Europa, mediante la exposición de las imágenes satelitales y sus productos en el portal, que posteriormente pueden ser tomados por usuarios para realizar un estudio de un área de su interés. Otro trabajo, en donde se desarrolló un geoportal dirigido a un solo tema de investigación, es el desarrollado por Fry *et al.*[25], en donde mediante el geoportal se exponen datos geoespaciales, con el objetivo de que los usuarios para acceder a datos socioeconómicos de el país de Gales. Para ello se exponen datos, como encuestas gubernamentales, datos geo-referenciados, publicaciones, libros, tesis doctorales, datos del gobierno, entre otros.

3.2 Entrega Continua

Entrega Continua (CD, por sus siglas) es una técnica de ingeniería de software en la que los equipos de desarrollo siguen produciendo software en ciclos cortos para asegurar que este pueda ser liberado de forma fiable en cualquier momento [14]. El software desarrollado bajo esta técnica debe de pasar por un ciclo de procesamiento, el cual contempla las siguientes fases: fase de compilación, fase de construcción, fase de aceptación, fase de pruebas de desempeño, fase pruebas manuales y la puesta en producción (ver Figura 3.1) [14]. Neely y Stolt [53] describen algunas lecciones aprendidas de la adopción de CD en una organización. En su trabajo concluyen que la documentación de los procesos de desarrollo y la automatización de procesos son dos puntos claves para lograr CD.

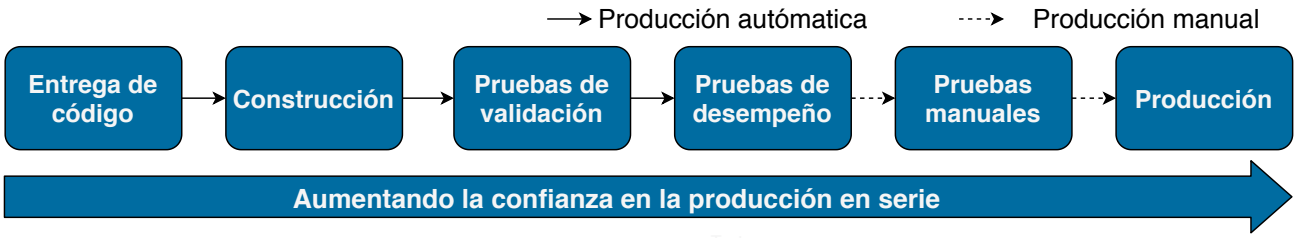


Figura 3.1: Ejemplo de una tubería de procesamiento de entrega continua. Fuente *Chen et al.[14]*.

Armenise [2] muestra como Jenkins[3] evolucionó de un orquestador de Integración Continua (CI, por sus siglas en inglés) a uno de CD. En CI el código de diferentes desarrollados es integrado en un solo repositorio, en el cual se aplican los cambios, y se hacen pruebas a cada cambio en el código de manera automática con el objetivo de detectar errores en la integración [23]. Jenkins nació como un servidor de integración continua, en el que se activan automáticamente flujos de trabajo para crear, probar e implementar componentes de software [52]. Sin embargo, con la introducción del concepto de CD, Jenkins evolucionó hasta convertirse en una plataforma de CD, el cual permite utilizar dos de los principales puntos clave de CD, como lo son: la colaboración entre todos los equipos involucrados en el ciclo de vida de producción de software, y en la automatización en el proceso de entrega [23]. Sin embargo, el uso de Jenkins para CD presenta diferentes problemáticas, como lo son, la gran cantidad de *plugins* que se tienen que instalar, además de que no esta preparado para trabajar con contenedores virtuales Docker de manera nativa, lo que limita a desplegar los procesos en máquinas físicas y virtuales.

3.3 Arquitecturas de microservicios

Las arquitecturas de microservicios son una técnica que permite el desarrollo de aplicaciones como pequeños servicios independientes. Los microservicios permiten la escalabilidad tanto horizontal como vertical de una aplicación. En la presente sección se describen algunas herramientas existentes para la orquestación y manejo de microservicios.

[3]https://jenkins.io/

3.3.1 Istio

Istio[4] fue lanzado por Google en el 2017 como una solución para la integración de microservicios, así como para el balanceo de carga entre microservicios desplegados en plataformas en la nube sin tocar el código de las aplicaciones [32].

Istio maneja una malla de servicios, la cual es utilizada para describir la red de microservicios mediante la interacción entre ellos. Se encarga de manejar requerimientos como descubrimiento de servicios, balanceo de carga, recuperación de fallas, monitorio, pruebas, control de acceso y autenticación punto a punto [36].

En la Figura 3.2 se muestra la arquitectura de Istio, la cual está dividida lógicamente en un plano de datos y un plano de control [36]. El plano de datos está compuesto por un conjunto de *proxies*, llamados *Envoy*, los cuales hacen de mediador y controlan toda la comunicación de red entre los microservicios y un concentrador de políticas y telemetría conocido como *Mixer*. El plano de control maneja y configura los *proxies* para enrutar el tráfico. Adicionalmente, el plano de control configura los *Mixers* para hacer cumplir las políticas y recolectar telemetría.

3.3.2 Spring Cloud

Spring Cloud encapsula el trabajo realizado por diferentes compañías como Pivotal, HashiCorp y Netflix en diferentes patrones. Mediante anotaciones en el código, los desarrolladores pueden hacer uso y desplegar las herramientas que encapsula Spring Cloud[11]. En la Figura 3.3 se presentan las tecnologías que encapsula Spring Cloud como patrones y anotaciones.

Las principales tecnologías que componen Spring Cloud son [11]:

- **Spring Boot**. Es el núcleo de las implementaciones de microservicios en Spring. Provee de anotaciones para la construcción de servicios REST, simplificando el mapeo de las acciones

[4] https://istio.io/

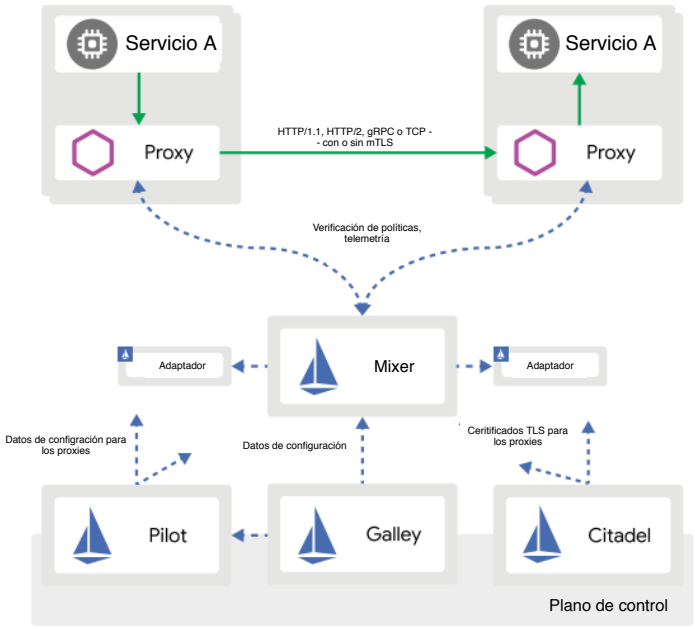


Figura 3.2: Arquitectura de Istio. Fuente *https://istio.io/docs/concepts/what-is-istio/*.

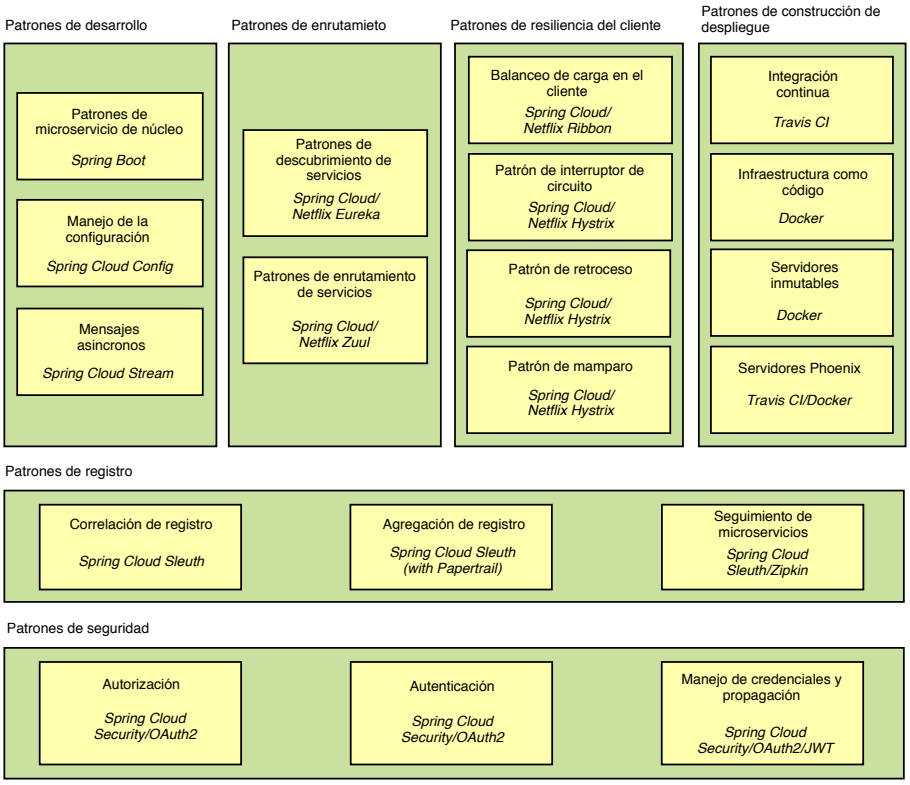


Figura 3.3: Tecnologías encapsuladas en Spring Cloud como patrones. Fuente: *Spring Microservices in Action [11]*.

HTTP (GET, PUT, POST y DELETE) a URLs, así como la serialización de objetos JSON a objetos Java, y viceversa.

- **Spring Cloud Config**. Maneja los datos de configuración a través de un servicio centralizado, separando los datos de configuración de los microservicios desplegados. Lo anterior asegura que los diferentes microservicios puedan compartir la misma configuración. Es compatible con diferentes proyectos de código abierto para el manejo de versiones (Git[5]) y el descubrimiento de servicios (Consul[6] y Eureka[7]).

- **Descubrimiento de servicios.** Permite abstraer la ubicación física (IP y nombre del servidor) de donde son desplegados los servicios. Además, maneja el registro de los servicios cuando éstos son levantados, y elimina el registro de los servicios cuando son apagados. Puede ser implementado utilizando Consul o Eureka.

- **Netflix Hystrix y Ribbon**. Hystrix[8] se encarga de gestionar las interacciones entre servicios, mientras que Ribbon[9] se encarga de la comunicación entre los procesos en la nube que se encargan del balanceo de carga.

- **Netflix Zuul**. Zuul[10] provee capacidades de enrutamiento para los microservicios. Zuul permite un enrutamiento dinámico, balanceo de carga, monitorización y seguridad de las peticiones.

3.3.3 Arquitecturas de microservicios para el procesamiento de datos científicos

Johanson *et al.*[39] presentan una arquitectura de microservicios para la visualización y procesamiento de datos de observación oceánica. Su arquitectura está dividida en tres capas

[5]https://git-scm.com/
[6]https:// www.consul.io/
[7](https://github.com/Netflix/eureka
[8]https://github.com/Netflix/Hystrix
[9]https://github.com/Netflix/ribbon
[10]https://github.com/Netflix/zuul

verticales: en la primera tienen los microservicios para el procesamiento de series de tiempo, mientras que en la segunda capa se encuentra el servicio para el análisis de datos espaciales, y finalmente en el tercer vertical cuentan con el servicio para el descubrimiento de patrones a través del tiempo. Sobre las tres capas, desplegaron un *gateway* (puerta de enlace) y un servicio de autenticación para el manejo del acceso a los servicios de capas inferiores. Cada uno de los microservicios, fueron encapsulados utilizando contenedores de Docker para garantizar que cada microservicio está aislado del resto.

Tavares de Sousa *et al.*[74] presentan una arquitectura de microservicios dividida en dominios, en donde cada dominio representa un conjunto de microservicios para diferentes tareas del procesamiento de datos científicos. En total, su arquitectura se divide en ocho verticales o dominios, como se observa en la Figura 3.4, los cuales son: adquisición, enriquecimiento, búsqueda, marcar, almacenar, preprocesar, procesar, analizar y publicar. Cada uno de estos dominios cuenta con microservicios que se encargan de realizar alguna tarea dentro de la arquitectura, habiendo tres microservicios en la mayoría de las capas, los cuales son: la interfaz de usuario, el API del servicio y un microservicio para la persistencia de los datos. En esta arquitectura los servicios se encuentran dentro de contenedores utilizando Docker, y desplegados utilizando técnicas de integración continua y entrega continua. En el trabajo no se presenta una evaluación de rendimiento de su arquitectura de microservicios, por lo que no se consideran esquemas para reducir los costos extras que agregan los contenedores virtuales y el uso de microservicios.

3.4 Flujos de trabajo para el procesamiento y manejo de datos

En la literatura se han propuesto diferentes motores y sistemas de flujos de trabajo que permiten a los usuarios desplegar aplicaciones que involucran hasta millones de tareas [50]. Comúnmente, los flujos de trabajo son modelados como redes de Petri o grafos acíclicos dirigidos (DAGs, por sus siglas

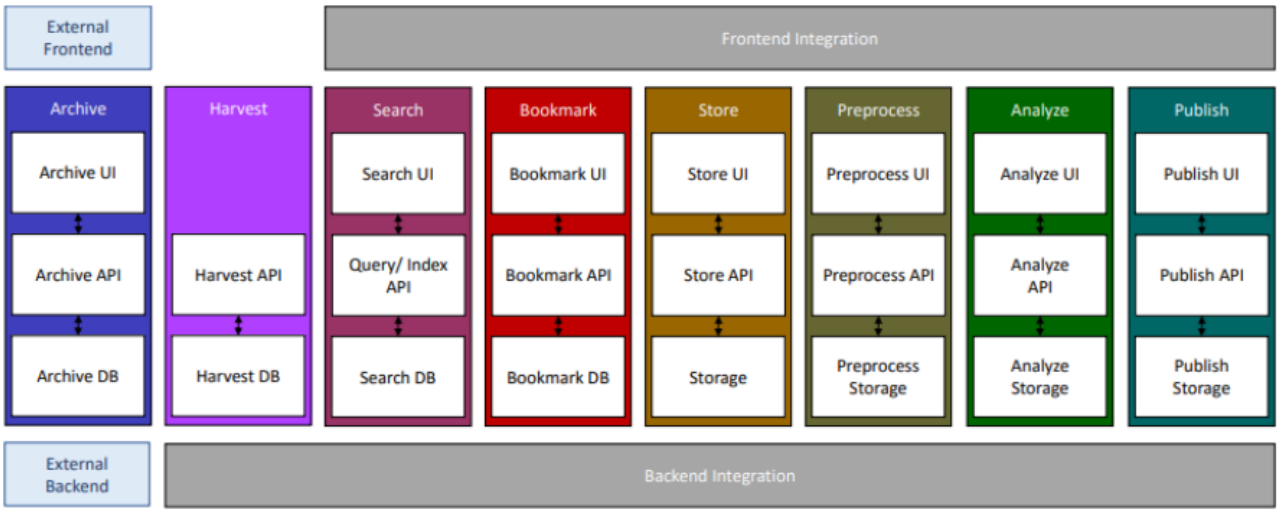


Figura 3.4: Arquitectura de microservicios presentada por Tavares de Sousa *et al.*[74].

en inglés), con la finalidad de facilitar su representación y procesamiento [3, 50, 83]. Para generar el modelo del flujo de trabajo, se han propuesto diferentes lenguajes de *scripting* como YAWL [45] o Swift [51], además, se han propuesto motores en donde la definición del flujo de trabajo se hace mediante lenguajes de programación de alto nivel como Python [37, 50, 83]. Otras herramientas, como *Galaxy Project* [42], tienen una interfaz gráfica en la que los usuarios pueden generar flujos de trabajo.

Los sistemas para la creación de flujos de trabajo despliegan las tareas en diferentes recursos computacionales como grupos de computadoras de alto rendimiento de cómputo (HPC, por sus siglas en inglés)[8, 50, 83], el *grid* [42, 51], la nube [49, 50, 83], contenedores [3, 29, 50] o en recursos locales como un servidor [50, 83]. Cada una de las tareas produce un conjunto de datos, que debe de ser accesible por otras tareas para atender dependencias, para ello se cada sistema cuenta con métodos para transportar los datos, que van desde compartir un sistema de archivos entre todas las tareas [29, 42, 50], enviar objetos serializados [3], bases de datos [37, 42], transferencia de archivos mediante protocolos como SCP, GridFTP o HTTP [3, 29] y memoria compartida [61, 29, 50].

Una de las aplicaciones de los flujos de trabajo es en el procesamiento de datos producidos en ambientes científicos o de investigación, en donde se pueden encontrar casos de uso como FACE-IT

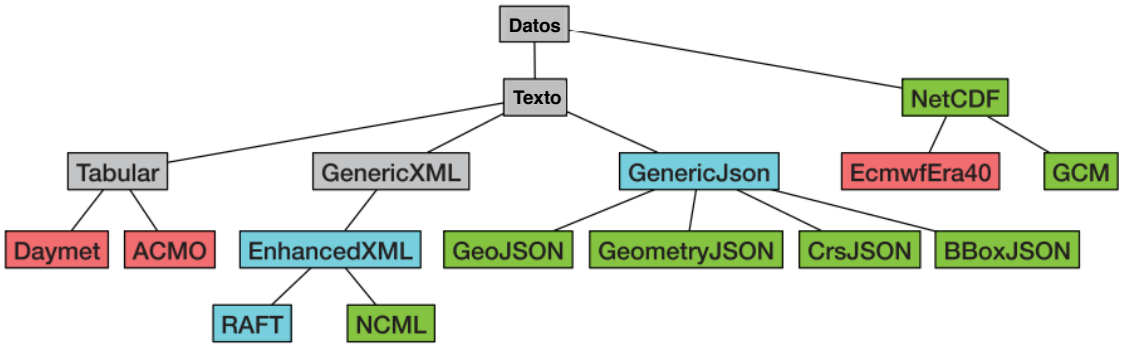


Figura 3.5: Tipos de datos manejados por FACE-IT. Fuente: *Montella et al.[51].*

(por sus siglas en inglés), presentado por Montella *et al.*[51], el cual permite a los investigadores desarrollar flujos de trabajo utilizando diferentes componentes de Galaxy [42], extendiendo los tipos de datos a aquellos que son de utilidad en investigaciones climáticas (Figura 2.2). En la Figura 3.6 se pueden observar los elementos que conforman FACE-IT, los cuales son tres: una entrada en forma de código y datos, flujos de trabajo creados mediante tuberías y simulaciones. FACE-IT ha sido utilizado para el monitoreo de granjas de mejillones en Nápoles, Italia [49], la cual es una de las aplicaciones que se le dieron a la plataforma dado que permite el manejo de diferentes fuentes de datos, la capacidad de crecer en términos del tamaño del dominio, y el desempeño computacional y efectividad necesarios para la toma de decisiones. En general, FACE-IT es utilizado para desarrollar tuberías complejas dedicadas a la adquisición, preprocesamiento, simulaciones y posprocesamiento de los datos, para conocer como la contaminación afecta los ecosistemas acuáticos.

Otro caso de uso es presentado por Cala *et al.*[8], los cuales presentan el diseño de un flujo de trabajo para el procesamiento de datos genómicos a larga escala. Los autores concluyen en que el uso de flujos de trabajo, combinados con la nube, proporcionan una mayor escalabilidad y flexibilidad, en términos del volumen de recursos computacionales en comparación de realizarlo en un clúster HPC. Otra de las ventajas presentadas, es la reproducibilidad de los bloques que componen el flujo de trabajo, permitiendo crear bloques trabajando en paralelo para mejorar la eficiencia en el procesamiento de los datos.

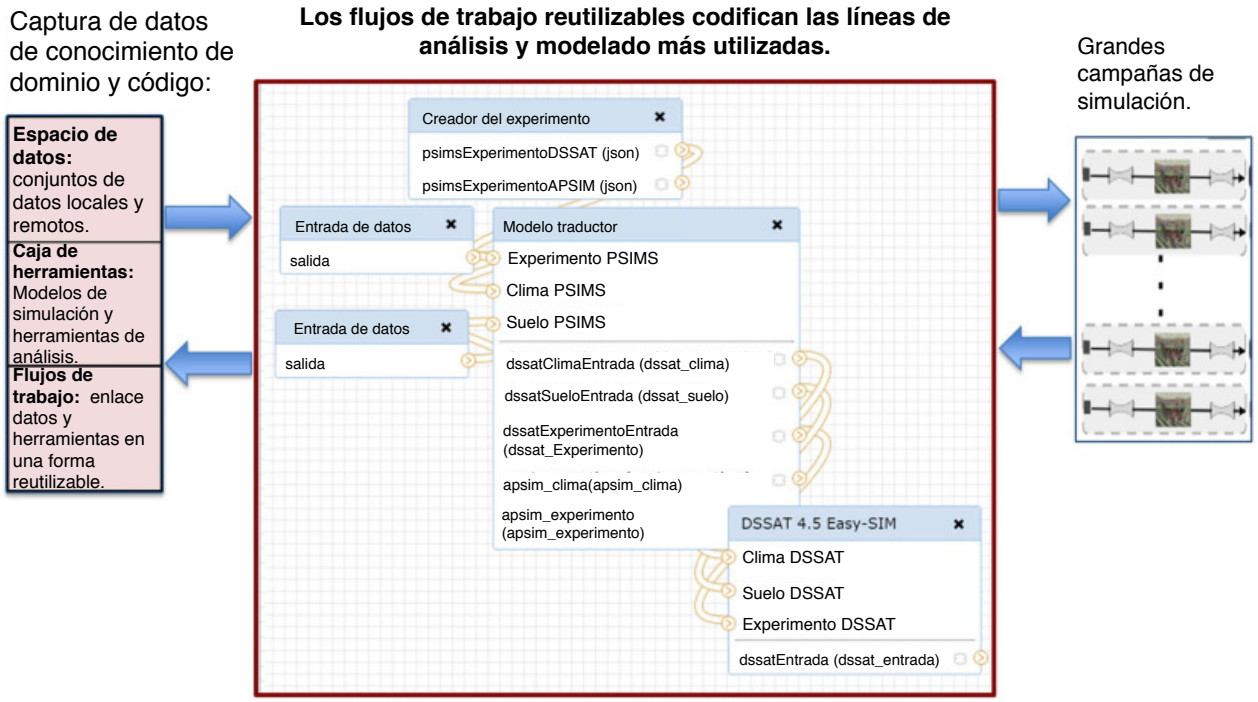


Figura 3.6: Elementos que conforman FACE-IT Fuente: *Montella et al.[51].*

En la literatura se han propuesto diferentes flujos de trabajo para el manejo y procesamiento de imágenes satelitales. González *et al.*[31] presentan FedIDS, el cual permite el transporte de datos geoespaciales (por ejemplo, imágenes satelitales y sus metadatos) a en flujos de trabajo colaborativos. FedIDS tiene dos componentes: Fed e IDS. El primero, permite a las organizaciones construir federaciones compartiendo recursos en la nube. Mientras que IDS permite a los usuarios compartir datos e imágenes satelitales en la federación evitando problemas de integridad y confidencialidad en los datos. Por otra parte, Liu *et al.*[40] proponen un flujo de trabajo para el procesamiento de datos geoespaciales utilizando unidades de procesamiento gráfico (GPU, por sus siglas en inglés) y esquemas de paralelismo. En su trabajo presentan como caso de estudio el procesamiento de imágenes MODIS (Moderate Resolution Imaging Spectroradiometer), probando el procesamiento en diferentes esquemas de paralelismo como sólo CPUs, sólo GPUs y una solución híbrida. A pesar de que logran reducir los tiempos de procesamiento, en su trabajo tienen problemas de cuello de botella en la entrada y salida de datos, debido a que se deben de realizar escrituras y lecturas constantes al

sistema de archivos.

Otros casos de estudio descritos en la literatura son en el procesamiento de datos otras ramas de las ciencias, como la química, la ciencia de los materiales, meteorología, bioinformática, cosmología y medicina [3, 37, 42, 50]. En general, los flujos de trabajo permiten el procesamiento a gran escala de datos de diferentes ramas de la ciencia, como se ha observado en los casos de uso descritos anteriormente.

3.4.1 Flujos de trabajo agnósticos

En el contexto de las tecnologías de la información, se dice que un componente de software o hardware tiene propiedades agnósticas cuando estos pueden operar en diferentes sistemas [18, 64]. En este sentido, un componente puede ser agnóstico de la plataforma (sistema operativo y arquitectura de hardware), agnóstico de los dispositivos (computadora personal, dispositivos móviles o servidores), agnóstico de la base de datos o agnóstico de los protocolos de comunicación, por mencionar algunos [64].

Ranabahu *et al.*[59] presentan SCALE una plataforma para la generación de flujos de trabajos científicos. SCALE es agnóstico de la plataforma en donde se ejecute, separando el código de los datos, transportando el código del flujo de trabajo entre diferentes infraestructuras para acercarlos a la ubicación de los datos. Sin embargo, SCALE no considera el uso de contenedores virtuales, los cuales permiten transportar las aplicaciones con todas sus dependencias, por lo que en SCALE cada plataforma donde se ejecute el flujo de trabajo, primeramente se deben de instalar las dependencias de la aplicación.

Reyes *et al.*[62] describe un modelo de construcción llamado Kulla, el cual está basado en entrega continua y aplicaciones agnósticas. Las aplicaciones se encuentran encapsuladas en contenedores virtuales en los cuales se instalan las dependencias de la aplicación, así como sus configuraciones y variables de entorno. El modelo de construcción está basado en DAGs, en donde los bloques de construcción son unidos mediante la entrada y salida de dato para formar patrones de procesamiento

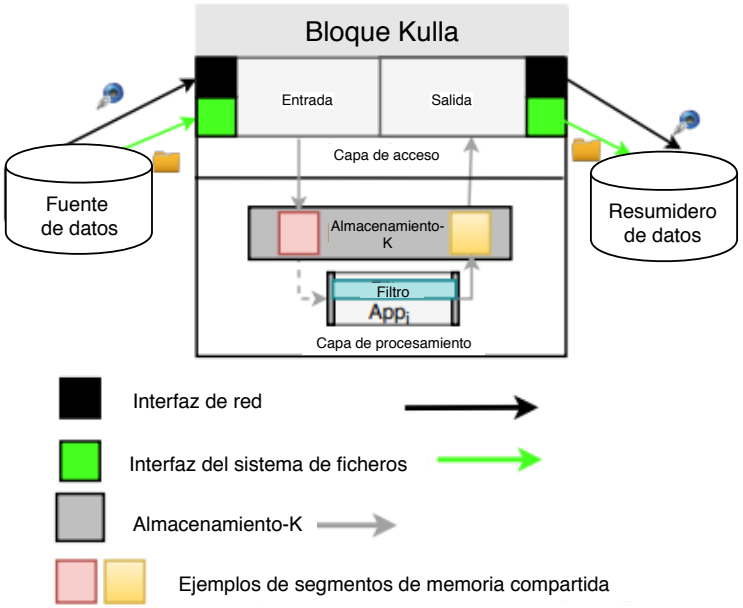


Figura 3.7: Representación conceptual de un bloque de construcción en Kulla. Fuente: *Reyes et al.* [62].

de datos. En la Figura 3.7 se presenta la representación conceptual de un bloque de construcción en Kulla, en donde la aplicación es encapsulada en un contenedor así como con otras estructuras de control para el manejo de datos, así como para la interoperabilidad con otros bloques.

3.5 Discusión

En esta sección se abordaron en general dos temas inherentes a la tesis a desarrollar, los cuales son el ciclo de vida de los EOPs y los flujos de trabajo. Primeramente se abordó el ciclo de vida de los EOPs, presentando trabajos desarrollados para cada una de las etapas que conforman este ciclo. A pesar de que se han propuesto soluciones para cada una de las fases del ciclo de vida de los EOPs, se han realizado esfuerzos para crear cadenas de estados con este ciclo que abarque desde la adquisición hasta la exhibición. Soluciones como la de Sevilla *et al.*[66] abarcan la adquisición de las imágenes satelitales desde las antenas, incluyendo el preprocesamiento de las imágenes y los metadatos, la extracción de los datos para almacenarlos en una base de datos que es consumida

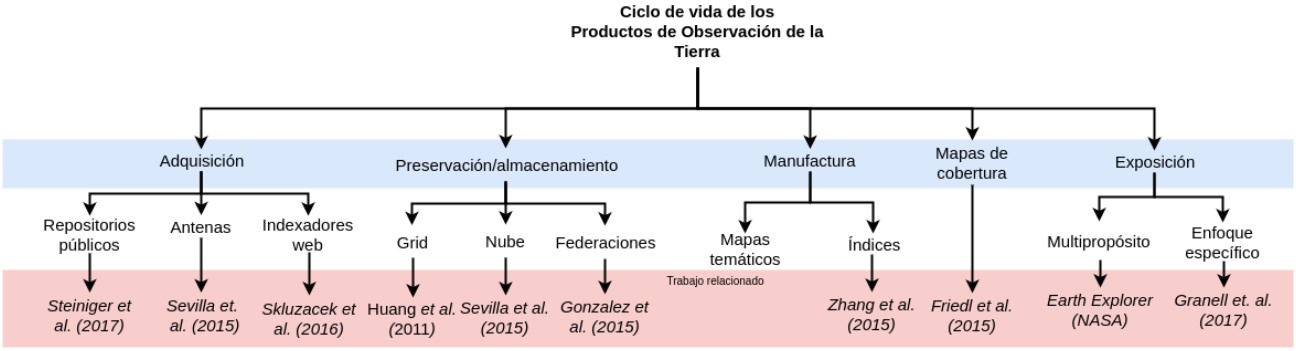


Figura 3.8: Taxonomía identificada a partir del ciclo de vida de los EOPs.

por un geoportal, con la finalidad de que los usuarios puedan consumir estos datos y las imágenes satelitales. Sin embargo, en estas soluciones no se contempla la generación automática de productos derivados a partir de las imágenes, así como esquemas para compartir los productos generados de diferentes etapas del ciclo de vida. Otro aspecto que no se contempla en la solución presentada en este trabajo, así como en el de Skluzacek *et al.*[69] es la de permitir a los usuarios, como investigadores, la selección de las etapas o tareas que requieren realizar, ya que muchas veces no requieren de todas las fases del ciclo de vida. Este último problema, podría ser resuelto mediante la implementación de una plataforma para la creación de flujos de trabajo, similar la presentada por Montella *et al.*[51].

En la Figura 3.8 se puede observar una taxonomía desarrollada a partir de los trabajos consultados. En la etapa de adquisición se ha identificado que los datos geoespaciales pueden ser adquiridos desde repositorios públicos, directamente desde antenas espaciales o utilizando indexadores web. Por su parte, la etapa de preservación/almacenamiento puede ser realizada utilizando entornos en la nube y federaciones. A partir de las imágenes satelitales, es posible manufacturar diferentes productos derivados, como lo son los mapas temáticos, los índices de vegetación, de calor, entre otros. Finalmente, en la exposición se identificado que está se realiza mediante geoportales, los cuales pueden ser de propósito general o enfocados a un tema de investigación como lo puede ser la deforestación o el crecimiento de la mancha urbana en una región. En las hojas de la taxonomía se encuentran trabajos relevantes para cada fase, y que se han descrito en este trabajo.

En lo que concierne a flujos de trabajo, se han identificado diferentes ventajas que tiene su implementación para los usuarios, siendo la principal que les permite concentrarse en el análisis de los resultados, olvidándose del manejo de los datos y la codificación de programas complejos para el procesamiento de los datos. En la Tabla 3.1 se muestra una tabla comparando cinco propuestas de flujos de datos descritas en la literatura, para ello se comparan diferentes características cualitativas de los trabajos, como lo son la infraestructura en donde fueron desplegadas, los datos utilizados en el artículo donde se describe la solución, la forma en la que se exhiben los resultados en el flujo de trabajo y el ámbito de estudio que se cubrieron con los estudios de caso presentados en los artículos. Como se puede observar, los flujos de trabajo se han aplicado en diferentes áreas de investigación científica, dado que los investigadores solo tienen que hacer uso de la plataforma sin preocuparse por la codificación de aplicaciones, así como de los recursos computacionales, requeridos para ejecutar dichas aplicaciones. En la Tabla 3.1, además se mencionan algunas restricciones a las que se presentan los flujos de trabajo descritos, como lo son el hecho de que las tareas no sean autocontenidas, la instalación de dependencias en máquinas remotas para la ejecución del flujo de trabajo, que las tareas requieran compartir el sistema de archivos, la transferencia de datos por medio de protocolos propietarios o que el usuario conozca un lenguaje de programación en específico, como Python.

En la Tabla 3.2 se muestra una comparación cualitativa de diferentes trabajos encontrados en la literatura para el diseño y despliegue de flujos de trabajo científicos. En esta comparación se consideran requerimientos funcionales tales como la eficiencia en el procesamiento y transferencia de datos, la portabilidad de los flujos de trabajo y las aplicaciones, así como la reusabilidad de estos mismos. Como se puede observar la solución propuesta en el presente tema de tesis considera diferentes aspectos no funcionales, los cuales permiten a las organizaciones construir soluciones utilizando patrones para mejorar la eficiencia de sus tareas, transportar tanto sus flujos de trabajo como sus aplicaciones entre plataformas y reutilizar sus aplicaciones o componentes de su flujo de trabajo en otros flujos.

Tabla 3.1: Comparación de las características de diferentes WFE para el procesamiento de datos.

Trabajo	**Representación del flujo de trabajo**	**Infraestructura**	**Caso de uso.**	**Diseño del flujo de trabajo**	**Restricciones**
Taverna-2013 (Wolstencroft *et al.*[80])	Diagramas y estructuras de datos.	Clúster, grid o en la nube.	Datos de ciencias de la vida.	Interfaz gráfica de usuario y archivo de comandos.	Las tareas no se encuentran aisladas entre ellas.
Goblus Galaxies (Madduri *et al.*[42])	Grafos acíclicos dirigidos y estructuras de datos.	Cluster, grid o HPC.	Datos genómicos, cardiovasculares, climáticos, cosmología, y ciencia de los materiales	Interfaz gráfica de usuario.	Requiere el despliegue de diferentes paquetes en cada nod.
FACE-IT 2015 (Montella *et al.*[51]) y 2018 (Montella *et al.*[49]).	Grafos acíclicos dirigidos y estructuras de datos.	Grid, la nube y clústers HPC.	Datos geoespaciales, de temperatura y marinos.	Interfaz gráfica de usuario.	Las tareas deben de compartir un sistema de archivos.
Parsl 2018 (Babuji *et al.*[3])	Grafos de tareas dependientes.	Tareas locales, en clústers HPC y la nube.	Bioinformática, química y ciencia de los materiales.	Código en Python con anotaciones especiales.	El manejo de datos se realiza mediante archivos.
Dagon* 2018 (Montella *et al.*[50])	Grafos acíclicos dirigidos.	En la nube, en contenedores y en clústers HPC.	Datos climáticos.	Código en Python y archivos JSON.	-

Tabla 3.2: Comparación cualitativa de los trabajos para diseño de flujos de trabajo de trabajo científicos.

		Eficiencia		Portabilidad		Reusabilidad	
Referencia	Enfoque	Patrones	Multihilo	FT	Apps	FT	Apps
Taverna *et al.*[80]	Motor de FT			✓		✓	
Jenkins [38]	Desarrollo de software		✓			✓	
Globus galaxies *et al.*[42]	Motor de FT		✓				
DagOnStar *et al.*[50]	Motor de FT		✓	✓		✓	
Parsl *et al.*[3]	Motor de FT	✓	✓	✓		✓	
Propuesta	Arquitectura de microservicios y contenedores	✓		✓	✓	✓	✓

FT: Flujos de trabajo

4

Diseño y desarrollo del modelo de construcción propuesto

En el presente capítulo se describe el modelo para la construcción y despliegue de flujos de trabajo basados en microservicios y contenedores virtuales para el manejo de ciclos de vida de los EOPs.

4.1 Descripción general de la propuesta

Se propone aplicar un modelo clásico de DAGs para modelar el ciclo de vida de los EOPs como flujos de trabajo. Esto significa que el modelo considera abstracciones tales como nodos, aristas y tareas. Los nodos representan etapas de un ciclo de vida, las aristas representan rutas de interconexión entre etapas de un ciclo y las tareas representan las aplicaciones que serán ejecutadas en cada etapa de un ciclo de vida. Un DAG entonces representa la unión de todos los aplicativos ($tareas$) de las etapas de procesamiento ($nodos$) de un ciclo de vida (*flujo de trabajo*) mediante un conjunto de

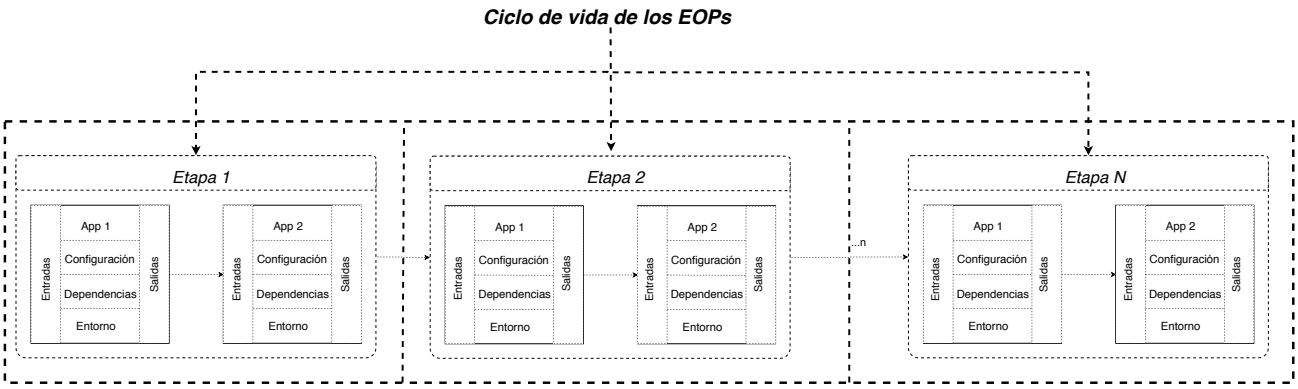


Figura 4.1: Representación conceptual de la solución propuesta.

interfaces de entrada y salida ($aristas$).

Construir un DAG para la construcción de un flujo de trabajo resulta relativamente común tomando en cuenta las opciones actualmente disponibles en el estado del arte (por ejemplo, Parsl[3], Jenkins[38] y DagOnStar[50], por mencionar algunas), las cuales permitirían desplegar un flujo de trabajo como el definido arriba.

Sin embargo, en escenarios del mundo real, las organizaciones tienen que hacer frente a diferentes problemáticas que se presentan cuando se desea modelar un ciclo de vida como un flujo de trabajo (ver Sección 1.2), por ejemplo, problemas de adaptabilidad, eficiencia y portabilidad.

En esta tesis se propone utilizar arquitecturas de microservicios para hacer frente al problema de manejo que se presenta en la capa de ciclo de vida de EOPs, así como la adaptabilidad en el nivel de etapa, donde las aplicaciones podrían ser modeladas como un conjunto de microservicios que eventualmente podrían ser agregados o eliminados de manera flexible, haciendo factible que las organizaciones puedan construir diferentes soluciones para cada ciclo de vida y para cada etapa (ver Figura 4.1).

Mientras, que para hacer frente a los problemas de heterogeneidad, en esta tesis se propone un modelo de construcción que encapsule las aplicaciones de las etapas en contenedores virtuales. Los contenedores virtuales han surgido como una alternativa a las máquinas virtuales para el transporte de aplicaciones y sus dependencias entre diferentes equipos, debido a que éstos son de menor tamaño

a las máquinas virtuales, dado que sólo se encapsula una versión reducida del sistema operativo, así como las aplicaciones y dependencias de la aplicación. Es por lo anterior que diferentes soluciones de la literatura han empezado a implementar las tareas desplegadas en un flujo de trabajo mediante el uso de contenedores virtuales [38, 50, 58].

Al encapsular las diferentes aplicaciones en contenedores virtuales, estás pueden ser desplegadas en la misma infraestructura, si es que ésta cuenta con una plataforma de manejo de contenedores (por ejemplo, Docker y contenedores Linux), sin que su funcionamiento afecte a otros procesos en el sistema. Las imágenes de los contenedores virtuales que se manejan en la arquitectura de microservicios propuesta en este trabajo de tesis incluyen un esquema de despliegue y acoplamiento basado en comunicaciones e intercambio de datos estandarizadas, los cuales ocultan los problemas de heterogeneidad a los usuarios finales.

Lo anterior, este método agrega un costo extra de despliegue y manejo de las tareas, debido a que se deben construir los contenedores virtuales que encapsulan las tareas [58], el cual afecta a la eficiencia de dichas soluciones.

La eficiencia en el procesamiento de una tarea impacta directamente a la etapa en el que es ejecutada, lo cual afecta también a la eficiencia del flujo de trabajo. Lo anterior en última instancia produce afectación a la experiencia de servicio del usuario final. Es por ello por lo que resulta crítico proveer a las distintas etapas de un flujo de trabajo con estructuras que reduzcan, tanto como sea posible, los tiempos de servicio de estas. Lo anterior con el fin de reducir el tiempo de respuesta de los flujos de trabajo y mejorar con esto la experiencia de servicio de los usuarios finales.

En el presente trabajo, los contenedores virtuales además de ser utilizados para hacer frente a los problemas de heterogeneidad también se utilizan para mejorar la eficiencia en el nivel de aplicación. Los contenedores virtuales permiten replicar una aplicación. Se propone que esta característica sea utilizada para organizar réplicas de una aplicación en la forma de patrones de paralelismo basados en tarea, lo cual reduce o mitiga el impacto de los cuellos de botella que aparezcan en una etapa dada.

Para dotar a una aplicación con propiedades de portabilidad, en la literatura se han propuesto

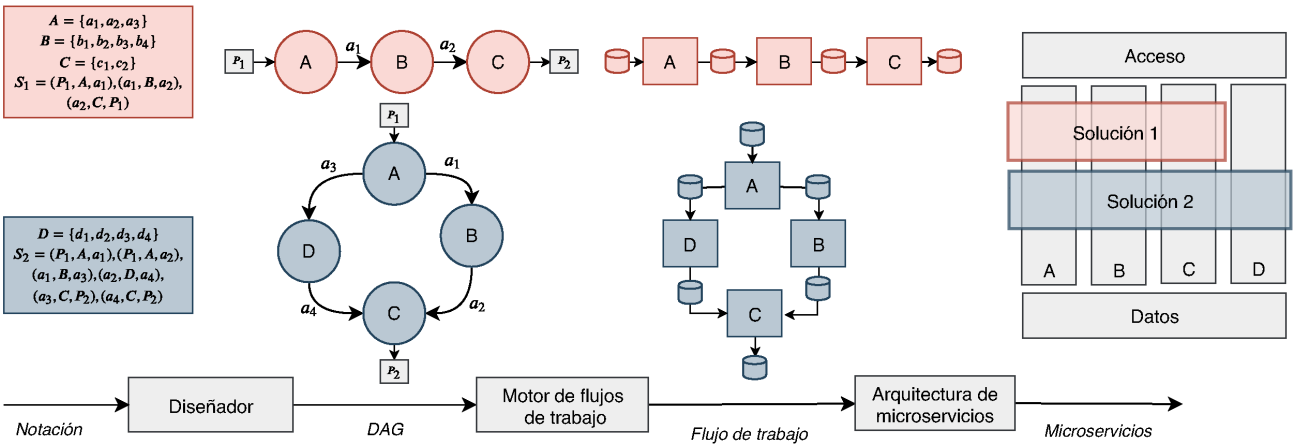


Figura 4.2: Etapas de despliegue de flujos de trabajo con el modelo propuesto.

el uso de dos esquemas. El primero es creando una descriptiva de metadatos relacionados las aplicaciones, en donde se incluya a detalle las configuraciones utilizadas para desplegarlas. Sin embargo, en esta solución no se garantiza que la aplicación funcionará exactamente igual infraestructuras diferentes a donde se desarrolló por primera vez [65]. La segunda solución, es la de encapsular el flujo de trabajo con todos sus aplicativos y dependencias en máquinas virtuales para transportarlas entre equipos. Sin embargo, las máquinas virtuales suelen ser de gran tamaño, lo que dificulta su transporte entre infraestructuras [58] y reduce la adaptabilidad de las soluciones desplegadas. En cambio, en el presente trabajo se propone la utilización de contenedores virtuales para garantizar la portabilidad de las aplicaciones y esquemas de acoplamiento para mejorar la reutilización de éstas.

El presente tema de tesis considera un modelo que incluye tres etapas de construcción. En la Figura 4.2 se presentan las etapas de construcción del modelo propuesto. En esta figura se puede observar que la primera etapa del modelo consiste de un esquema declarativo, en donde cada etapa del ciclo de vida de los EOPs sea modelada como un microservicio ($nodo$) que en su interior contenga aplicaciones encapsuladas en contenedores virtuales ($tareas$). Como resultado de esta etapa se obtiene un DAG que describe las rutas de transformación de un ciclo de vida de un EOP dado.

En la segunda etapa, un motor de construcción crea flujos de trabajo interpretando el DAG y ejecutando los contenedores virtuales a los que hace mención el DAG, los cuales se pueden utilizar como bloques de construcción que pueden desplegarse en forma agnóstica y funcionar en forma independiente.

En la tercera y última etapa se crea una arquitectura de microservicios para presentar el flujo de trabajo como una única solución. La arquitectura de microservicios permite dotar a dicha solución con estructuras de control de acceso y un intercambio de datos transparente para los contenedores virtuales usados en la solución creada dentro de la arquitectura de microservicios.

En este punto, diversas soluciones se pueden construir variando el número de etapas requeridas para modelar un ciclo de vida (ver Figura 4.2). Esto significa que esta arquitectura permite construir *artefactos de software* (se pueden entender como plantillas que incluyen dependencias, procesos de sincronización y control) para cada flujo de trabajo creado, lo cual garantiza un manejo del control de acceso e intercambio de datos transparente para el flujos de trabajo gestionado por el artefacto.

En las siguientes secciones se describen en detalle los componentes de las tres etapas del modelo propuesto.

4.2 Componentes del modelo de construcción

Los componentes principales del modelo de construcción son tres: las microaplicaciones que se ejecutan al interior de una etapa del ciclo de vida, los microservicios que son abstracciones de una de las etapas del ciclo de vida, y los EOPs de entrada y salida para cada una de las etapas y microaplicaciones.

En este sentido, una microaplicación o *Micro-App*, es la unidad básica de software del modelo de construcción propuesto. Básicamente, son cajas negras que contienen una aplicación utilizada en alguna de las etapas del ciclo de vida de los EOPs. Cada *Micro-App* se encuentra autocontenida en un contenedor virtual, como se puede observar en la Figura 4.3. Dentro del contenedor virtual se

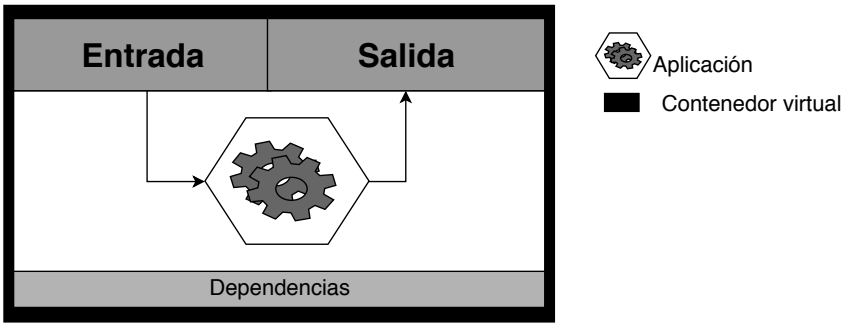


Figura 4.3: Diseño conceptual de una microaplicación (*Micro-App*). Cada *Micro-App* se encuentra autocontenida dentro de un contenedor virtual.

instalan todas las dependencias de software requeridas para que la *Micro-App* pueda ser transportada y ejecutada en otros equipos o infraestructura IT (por sus en inglés), como la nube o un servidor de un clúster, sin necesidad de requerir de una fase de configuración de las dependencias. Además, la *Micro-App* cuenta con dos interfaces: una de entrada de EOP llamada $EOP - E$ y otra para los EOPs de salida llamada $EOP - S$. Las entradas y salidas de las microaplicaciones pueden ser tanto por el sistema de archivos, así como por red (HTTP, sockets o CURL).

Las *Micro-Apps* son utilizadas como unidades de construcción para generar una solución para alguna de las etapas del ciclo de vida de los EOPs. En este sentido, los microservicios son utilizados como envoltorios de las *Micro-Apps* que conforman cada etapa, definiendo las entradas y salidas de la aplicación mediante un API REST, como se observa en la Figura 4.4. Cada microservicio es independiente del resto de microservicios, así como cada *Micro-App* lo es del resto de *Micro-Apps*, por lo que el contexto de cada microservicio está limitado a las *Micro-Apps* en su interior.

En este modelo los microservicios pueden ser encadenados entre sí, para crear flujos de datos a través de *flujos de trabajo* para diferentes tipos de soluciones. Estas cadenas pueden ser desplegadas en una infraestructura, por ejemplo, en la nube o un clúster HPC, como un servicio, lo cual permitiría que las agencias y misiones puedan compartir sus servicios con terceros para consumir sus funciones.

Los EOPs de entrada y salida se encuentran en piscinas de recursos de almacenamiento, como el sistema de archivos, bases de datos o un sistema de compartición de archivos en red. De esta piscina

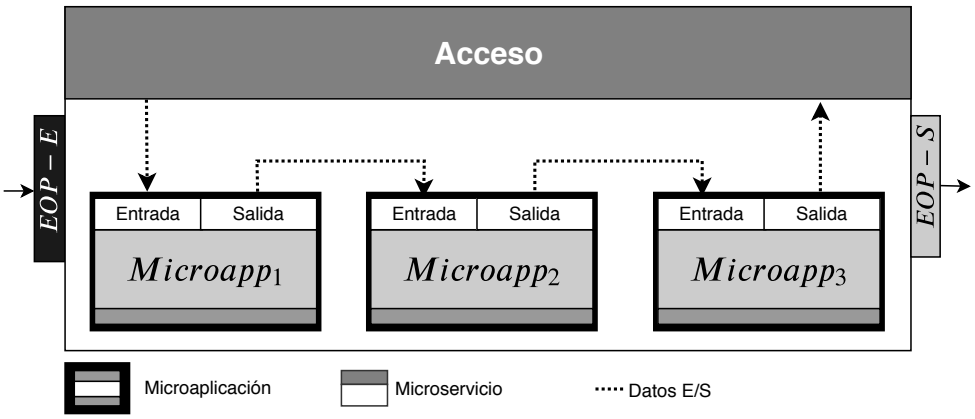


Figura 4.4: Diseño conceptual de un microservicio.

de almacenamiento, las microaplicaciones, y por ende los microservicios, pueden obtener los $EOP-E$ y entregar los $EOP-S$, así como sus respectivos metadatos. Dentro de la arquitectura, puede haber diferentes piscinas de almacenamiento, las cuales tienen que ser accesibles para el microservicio que lo requiera.

4.3 Modelado de flujos de trabajo de los EOPs como grafos dirigidos

Como fue descrito anteriormente, con el modelo propuesto las organizaciones pueden construir flujos de trabajo encadenando nodos, los cuales se encuentran abstraídos como microservicios que puede ser consumidos por el usuario final o por otros servicios.

En el modelo, las aplicaciones encapsuladas en una *Micro-App* se representan como abstracciones conocidas como *tareas* (t), mientras un $nodo$ representa una etapa del ciclo de vida de los EOPs, el cual ha sido encapsulado en un microservicio, y se representa como N. Esta abstracción es utilizada como un envoltorio, tanto de una tarea como de un conjunto de tareas ($t_1, t_2, \ldots, t_n \in N$).

$EOP-E$ representa una ruta para obtener un EOP requerido por una *tarea* como parámetro de entrada, mientras que $EOP-S$ es la ruta donde se almacenará la versión del EOP producida por

la *tarea*. Ambas abstracciones pertenecen a una abstracción llamada *piscina* (P), la cual representa un espacio de almacenamiento donde una tarea puede obtener $(EOP - E)$ y depositar $(EOP - S)$ EOPs.

Para representar una tarea ejecutada en paralelo se utiliza la abstracción *Tarea paralela* (Pt), donde una tarea t es clonada n veces y se denota como:

$$Pt = t_1 \parallel t_2 \parallel t_3 \cdots \parallel t_n. \tag{4.1}$$

Para encadenar dos nodos $(N_i \Rightarrow N_j)$ se utiliza una arista (a). En la arista se colocan los datos de salida de un nodo n_1, que son tomados como entrada por otro nodo n_2.

Una cadena (C) representa un servicio creado por un conjunto de nodos interconectados por un conjunto de aristas siguiendo una dirección dada. Su representación es dada por un conjunto de nodos conectados por aristas, lo cual es denotado como $< E, N, S >$, donde E representa la entrada de datos (*piscina* P o *Arista* a), N es el nodo que recibe los datos entrantes y S representa donde serán depositados los datos en una (*piscina* P o en una *Arista* a). Por ejemplo, $\{(P_i, N_1, a_1), (a_1, N_2, a_2), (a_2, N_3, a_2)\} \in C_1$, es la representación de la cadena mostrada en la Figura 4.5, compuesto por tres nodos y dos aristas, y puede ser leído como: *"El nodo 1 lee datos de la piscina 1 y escribe en la arista 1, desde donde el nodo 2 lee datos y los escribe en la arista 2, donde el nodo 3 lee los datos y escribe su salida en la piscina 2"*.

Un DAG es la representación de todas las cadenas (servicios) creados por cualquier usuario, misión, agencia u organización. Un DAG es representado como *Grafo* (G), e incluye un conjunto de cadenas de la forma $G = \{C_1, C_2, C_3, \ldots, C_n\}$.

Como se puede observar, este modelo permite a las organizaciones crear diferentes flujos de trabajo para el procesamiento de EOPs, dependiendo de las entidades incluidas en el ciclo de vida de los EOPs. Por ejemplo, cuando una organización solo necesita almacenar y exhibir las imágenes satelitales, la siguiente notación puede ser utilizada: $C_1 = (P_1, N_1, a_1), (a_1, N_2, P_2)$, donde N_1 y

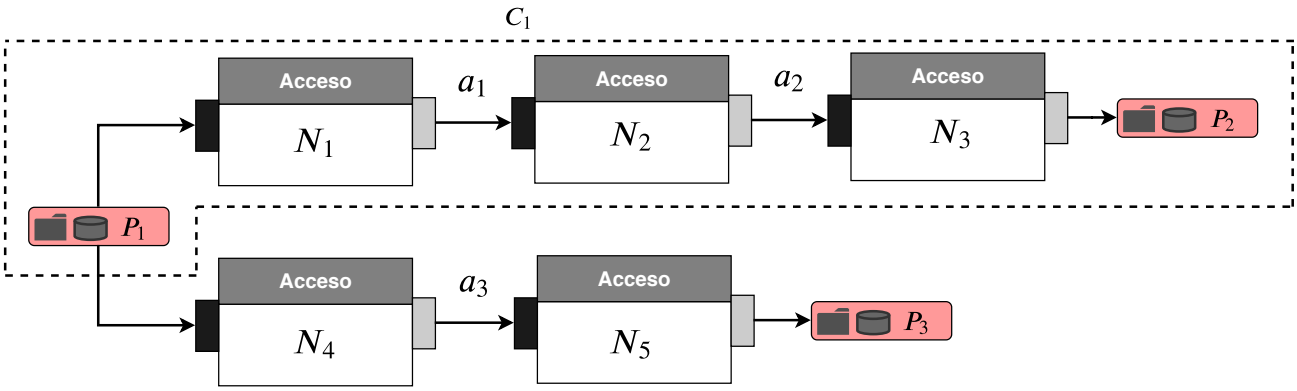


Figura 4.5: Ejemplo de un grafo creado con el modelo propuesto.

N_2 representan el almacenamiento y la exhibición (en la forma de un geoportal) respectivamente, mientras a_1 representa la arista donde N_1 coloca sus $EOP-S$ y N_2 lee sus $EOP-E$, así como P_1 y P_2 representan los recursos donde los $EOP-E$ son leídos por N_1 y donde los $EOP-S$ son escritos por N_2 respectivamente. Siguiendo este ejemplo, una organización puede agregar un nuevo nodo a esta cadena utilizando la siguiente notación:

$$C_2 = (P_3, N_3, P_1), C_1. \tag{4.2}$$

Donde $N_3 = Pt$, el cual es una etapa de manufactura, donde las tareas son realizadas en paralelo y el nodo fue añadido al inicio de la primera cadena (C_1). Como resultado de esta acción, la organización termina manejando $G = (C_1, C_2)$.

Tres características pueden ser observadas de esta notación:

1. La adaptabilidad del modelo para asimilar nuevas etapas en un ciclo de vida de los EOPs, solo reajustando las entradas y salidas de las aristas de un nodo N dado.

2. La reutilización de cadenas, nodos y aristas para crear nuevos servicios para los usuarios finales.

3. Los flujos de datos se están creando a medida que los microservicios se encarguen de entregar datos/EOPs/metadatos a las aplicaciones encapsuladas en *Micro-Apps*.

4.4 Construcción de patrones de paralelismo de tareas

Con el modelo propuesto es posible desarrollar patrones de paralelismo de tareas dentro de los nodos mediante el encadenamiento de *Micro-Apps* ejecutadas de una manera concurrente. Para mostrar la factibilidad de esta característica de nuestro modelo, en la presente tesis se ha desarrollado patrón Manejador/Trabajador (conocido en inglés como Manager/Worker, M/W).

En un patrón M/W, un $worker$ o trabajador representa una réplica de un *Micro-App* que procesa un EOP en el interior de un nodo. El $manager$ o manejador está a cargo de lanzar tantos trabajadores como sea definido en la configuración de un nodo y coordinar la entrega de los EOPs procesados por los trabajadores al siguiente nodo. Como resultado de este desarrollo, una tarea es procesada en paralelo por tantos trabajadores haya lanzado el manejador.

Por ejemplo, considerando la siguiente tarea:

$$\{t(M), t(w_1, w_2, \ldots, w_n)\} \in N_1. \tag{4.3}$$

Donde $t(M)$ es el manejador, $t(w_1, w_2, \ldots, w_n)$ son los trabajadores y N_1 es un nodo del grafo. $t(w_1, w_2, \ldots, w_n)$ puede ser reescrita de la siguiente manera:

$$P_t = t(w)_1 \parallel t(w)_2 \parallel t(W)_n \cdots \parallel t(w)_n. \tag{4.4}$$

El manejador recibe una lista de $EOP - E$ ($ListaEOP$), los cuales son distribuidos entre los trabajadores en P_t. Por ejemplo, dada una lista de $EOP-E$ con la forma $\{EOP_1, EOP_2, EOP_n\} \in EOP - E$, la notación utilizada para definir la distribución de los productos es:

$$t(M) \leftarrow ListaEOP. \tag{4.5}$$

$$t(M) \Rightarrow Pt. \tag{4.6}$$

Entonces, un nodo usando este patrón se denota como:

$$N_1 = \{t(M), P_t\}. \tag{4.7}$$

En la Figura 4.6 se muestra el despliegue de servicios creados utilizando la notación descrita. Como se puede observar, un nodo (microservicio) recibirá EOPs para ser procesados (por ejemplo, imágenes satelitales o archivos de metadatos), el manejador distribuirá los EOPs a los trabajadores, los cuales son la misma *Micro-App* clonada, que invocará una tarea para procesar los EOPs. Como resultado, T_p procesará los EOPs en paralelo. Esto mejorará el tiempo de respuesta de un nodo, cuando este tenga que procesar un conjunto de EOPs. Lo anterior se logra, dado que las aplicaciones son encapsuladas junto con todas sus dependencias, bibliotecas, variables de entorno y sistema operativo que garantice su correcto funcionamiento.

4.5 Despliegue de tareas agnósticas

El presente modelo considera dotar a las $tareas$ ejecutadas por un nodo con la característica de $agnosticidad$. Lo anterior permite a las organizaciones el reutilizar componentes de un flujo de trabajo previamente construido.

En el modelo las tareas son encapsuladas en contenedores virtuales, lo que permite que sean transportadas entre diferentes infraestructuras. En cada contenedor virtual se instala el código fuente de la aplicación así como todas sus dependencias, creando una imagen de contenedor que puede ser desplegada en diferentes infraestructuras. Por lo que diferentes aplicaciones heterogéneas pueden ser desplegadas en la misma infraestructura. Por ejemplo, en la Figura 4.7 se muestra como las etapas del ciclo de vida de los EOPs puede ser desarrollada utilizando una variedad de lenguajes de programación

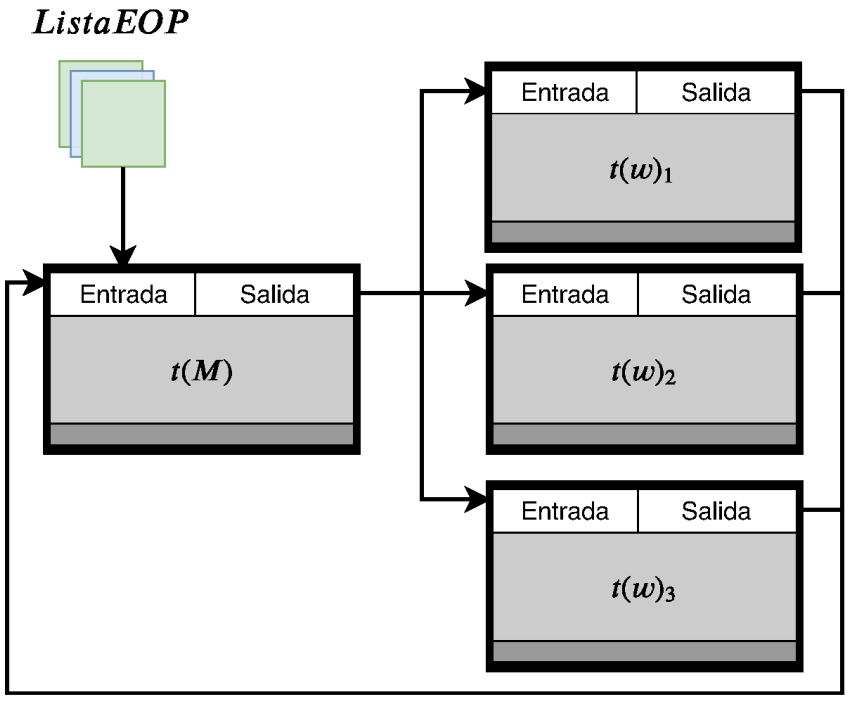


Figura 4.6: Patrón *M/W* desplegado con el modelo propuesto.

diferentes, así como diferentes tecnologías de almacenamiento de datos. Por ejemplo, el servicio de adquisición fue desarrollado utilizando Python y MongoDB para la persistencia de datos. Este servicio envía datos a los servicios de almacenamiento y manufactura, los cuales se desarrollaron utilizando Java y MatLab respectivamente. El servicio de exhibición desarrollado en PHP hace peticiones a los servicios de almacenamiento y *mapping* (JavaScript) para solicitarles EOPs en la forma de productos, metadatos y datos. Finalmente, el cliente y otras aplicaciones externas pueden realizar peticiones a los servicios en cualquier etapa del ciclo de vida de los EOPs mediante las APIs REST.

En este sentido, en la Figura 4.8 se muestra la arquitectura en capas de las tareas agnósticas manejadas por el modelo propuesto. En la capa superior (Entrada/Salida) se encuentran las interfaces de entrada y salida para llevar a cabo la comunicación entre las tareas desplegadas en un nodo. Posteriormente, en la capa intermedia se encuentran las aplicaciones que conforman las tareas, y que se encargan de procesar los datos de entrada para producir los datos de salida. Finalmente, en la capa inferior, se tienen los datos que utilizan las aplicaciones, los cuales pueden ser desde archivos

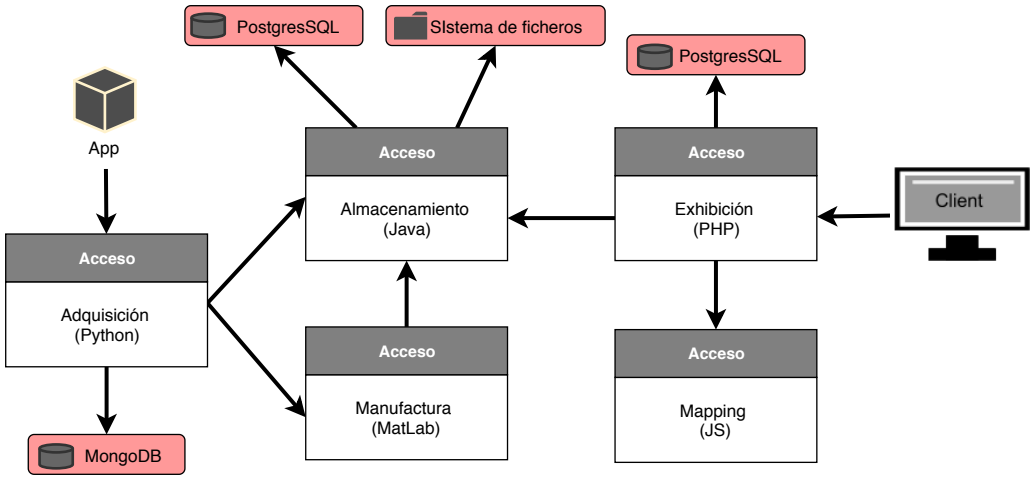


Figura 4.7: Despliegue de tareas heterogéneas utilizando el modelo propuesto.

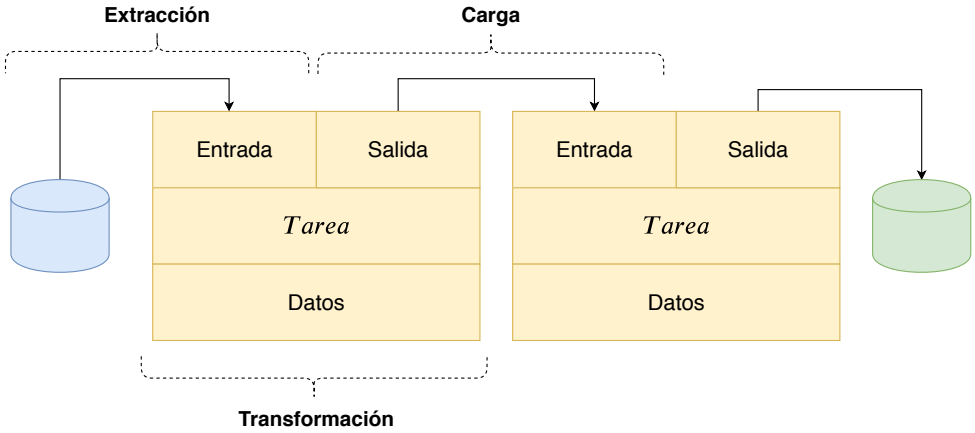


Figura 4.8: Arquitectura de las tareas manejadas en el modelo y del proceso ETL para el intercambio de datos entre tareas.

de configuración, bitácoras de ejecución o cualquier entrada o salida extra que requiera la aplicación.

En la Figura 4.8 también se ilustra el proceso de Extracción, Transformación y Carga (ETL, por sus siglas en inglés) [79] utilizado en el modelo para realizar el intercambio de datos entre tareas. Lo anterior significa que, para cada pieza de software encapsulada, las organizaciones deben declarar las rutas de *Extracción*, el aplicativo encargado de la *Transformación* y las rutas en las cuales se *Cargan* los resultados de la transformación realizada. Este proceso declarativo se realiza utilizando la notación previamente descrita, en la que los datos de entrada se modelan como $EOP - E$ y los de salida como $EOP - S$. Además, en el modelo las dependencias de datos entre tareas se declara

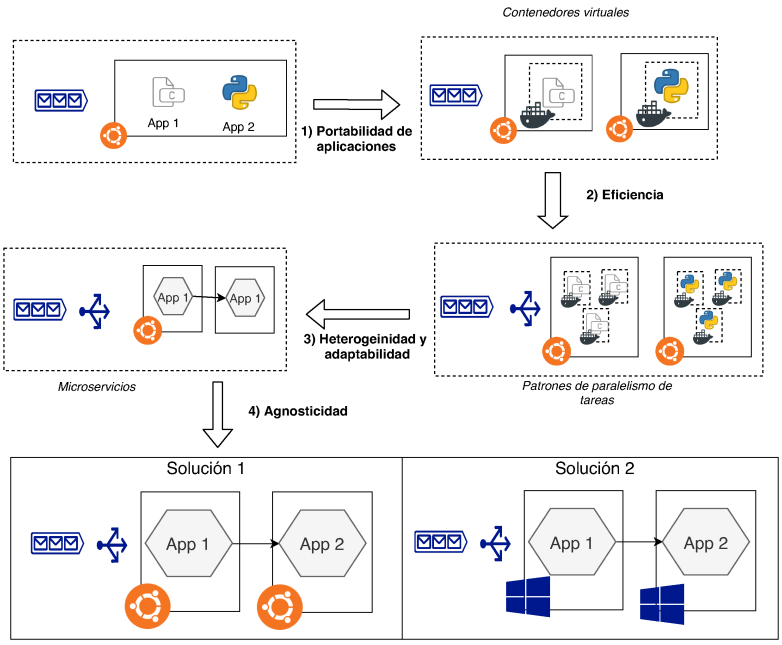


Figura 4.9: Flujo de construcción de soluciones agnósticas con el modelo propuesto.

como una $arista$.

En la Figura 4.9 se presenta el flujo de construcción de soluciones agnósticas utilizando el modelo propuesto. En la primera etapa las soluciones son encapsuladas en contenedores virtuales para resolver problemas de portabilidad, posteriormente se generan patrones de paralelismo de tareas con las aplicaciones encapsuladas para solventar los problemas de eficiencia de las tareas desplegadas en una etapa. En la tercer etapa, los flujos de trabajo se despliegan sobre una arquitectura de microservicios que permite ocultar los problemas de heterogeneidad, así como dotar a las soluciones de adaptabilidad a cambios. Hasta este punto se ha dotado a los flujos de trabajo construidos con este modelo con propiedades agnósticas, dado que se pueden desplegar en distintas plataformas sin que el usuario tenga que realizar procesos de modificación en el código de las aplicaciones.

4.6 Esquema de acoplamiento de soluciones

En un archivo DAG utilizando el modelo de construcción descrito se definen todos los nodos, tareas y aristas (conexiones entre nodos) utilizando la notación descrita. Dicho archivo, es enviado a un esquema de acoplamiento de soluciones, en donde cada $nodo$ es encapsulado en contenedores virtuales, y son interconectados a partir de las conexiones declaradas en el DAG. En la Figura 4.10 se muestra él proceso para la creación, construcción, despliegue y manejo de un flujo de trabajo a partir de una notación escrita con el modelo propuesto. Las etapas descritas en la Figura 4.10 se enlistan a continuación:

1. La notación escrita por el usuario es analizada y transformada en un archivo JSON por un cliente web, el cual es enviado al *Manejador* para que inicie con la construcción y despliegue del flujo de trabajo.

2. Se inicia la construcción de los archivos *Dockerfile* utilizados para la creación de las imágenes de contenedor.

3. Se obtienen las imágenes de contenedor que se utilizarán como base para la construcción de los contenedores.

4. Se construye un archivo YML en el cual se declaran todos los servicios (microaplicaciones y microservicios) que componen el flujo de trabajo. En el archivo se especifica la imagen de contenedor que utilizará cada servicio.

5. El archivo YML es enviado al *Manejador*.

6. Se realiza la construcción de las imágenes de contenedor utilizando el archivo YML y Docker Compose.

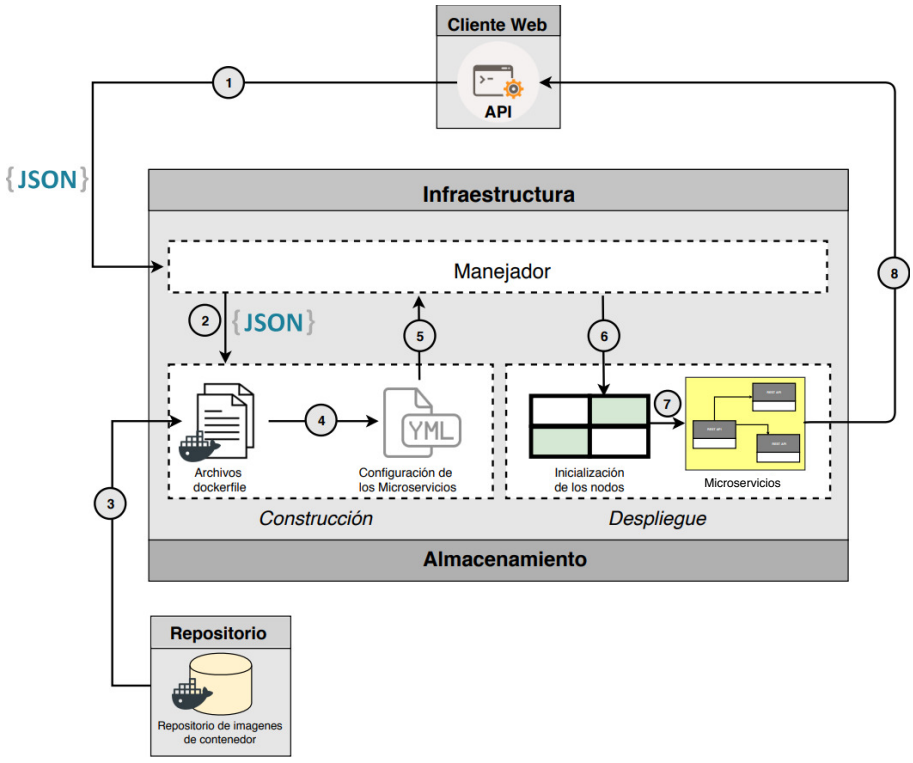


Figura 4.10: Ejemplo de la construcción y despliegue de flujos de trabajo.

7. Una vez desplegados los contenedores se obtiene una arquitectura de microservicios, en donde los contenedores marcados como $nodos$ se encargan de recibir carga de trabajo mediante un API REST.

8. El cliente consume las APIs de los microservicios generados para obtener los productos producidos en cada microservicio.

4.7 Servicios de EOPs a partir de grafos dirigidos

En el modelo propuesto en la presente tesis, los grafo la representación basada en grafos dirigidos no es solo utilizada para crear flujos de trabajo para cubrir las necesidades de los usuarios, además permiten el despliegue de servicios en una infraestructura dada, mediante la combinación de un conjunto de microservicios.

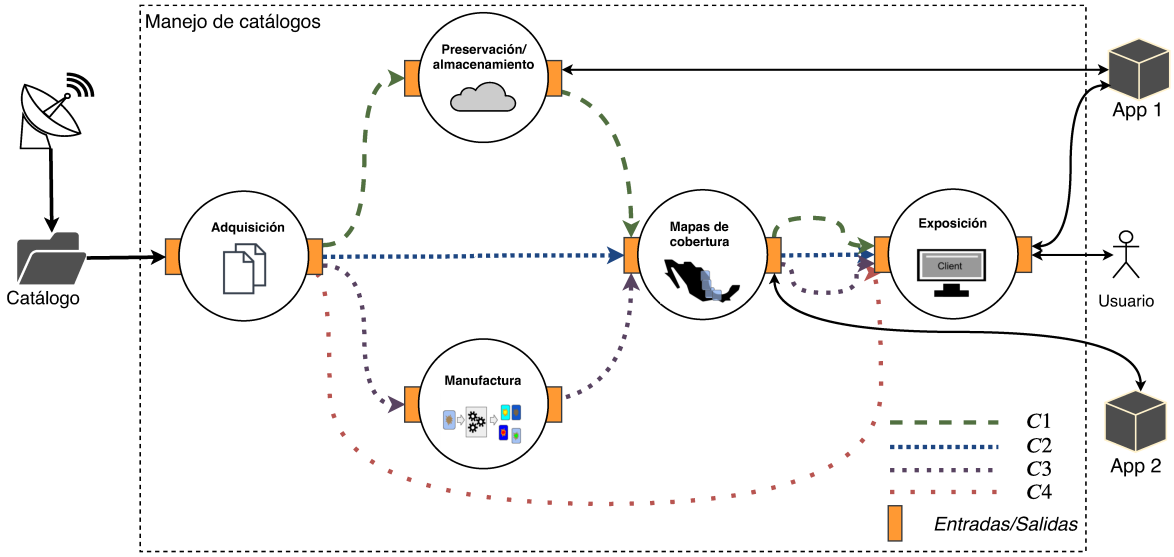


Figura 4.11: Representación conceptual de un servicio construido a partir de múltiples grafos.

En la Figura 4.11 se muestran algunas combinaciones de microservicios para crear múltiples soluciones (flujos de trabajo) para el procesamiento de EOPs. Como se puede observar, el usuario puede seleccionar el camino de microservicios dependiendo de los servicios en lo que este interesado. Las líneas del mismo color en la Figura 4.11 muestran ejemplos de flujos de trabajo, donde los caminos pueden ser seleccionados por usuarios finales (agencias, misiones y organizaciones). La notación de las combinaciones mostradas en la Figura 4.11 son:

$$C_1 = (P_1, N_{Adquisición}, a_1), (a_1, N_{Preservación}, a_2), (a_2, N_{Mapeo}, a_3), (a_3, N_{Exhibición}, P_2), \quad (4.8)$$

$$C_2 = (P_1, N_{Adquisición}, a_4), (a_4, N_{Mapeo}, a_5), (a_5, N_{Exhibición}, P_2), \quad (4.9)$$

$$C_3 = (P_1, N_{Adquisición}, a_6), (a_6, N_{Manufactura}, a_7), (a_7, N_{Mapeo}, a_8), (a_8, N_{Exhibición}, P_2), \quad (4.10)$$

$$C_4 = (P_1, N_{Adquisición}, a_9), (a_9, N_{Exhibición}, P_2). \tag{4.11}$$

Donde C_x es una cadena o solución construida, $N_{Adquisición}$ es un nodo de adquisición de EOPs, $N_{Preservación}$ es un nodo de preservación de datos, N_{Mapeo} es un nodo de generación de mapas, $N_{Exhibición}$ es la exhibición de los datos en un geoportal y $N_{Manufactura}$ es un nodo de generación de productos derivados. Además, a_x son las aristas que interconectan los nodos, así como P_1 y P_2 son la fuente y resumidero de datos respectivamente.

4.8 Resumen

En el presente capítulo se presento el diseño conceptual de un modelo para la construcción de flujos de trabajo basados en una arquitectura de microservicios para el manejo del ciclo de vida de los EOPs. Dicho modelo, contempla el uso de contenedores virtuales para hacer frente a los problemas de portabilidad de aplicaciones, mediante el encadenamiento de estos en la forma de patrones de paralelismo de tareas, hacer frente a problemas de eficiencia. En el método propuesto, se utilizan DAGs para modelar el ciclo de vida de los EOPs como flujos de trabajo. Esto significa que el modelo considera abstracciones tales como nodos, aristas y tareas. Los nodos representan etapas de un ciclo de vida, las aristas representan rutas de interconexión entre etapas de un ciclo y las tareas representan las aplicaciones que serán ejecutadas en cada etapa de un ciclo de vida.

5

Desarrollo de prototipo de experimentación

Con base en el modelo diseñado en Capítulo 4, se realizó la implementación de un prototipo de experimentación. En el presente capítulo se describe el diseño del prototipo, así como la infraestructura de experimentación, repositorios de datos utilizados, así como el diseño e implementación de un flujo de trabajo utilizado en una primera fase de evaluación controlada. Al final, del capítulo se presentan los resultados preliminares de la evaluación controlada, así como una comparación con dos motores de flujos de trabajo *Parsl*[3] y *DagOnStar*[50].

5.1 Infraestructura utilizada

Para realizar pruebas de los diferentes componentes se utilizaron los equipos mostrados en la Tabla 5.1. Para realizar las pruebas de desempeño del despliegue de patrones de paralelismo basados en contenedores se utilizó la máquina con la etiqueta compute6 (ver Tabla 5.1).

Tabla 5.1: Características de hardware de los equipos que componen la nube.

Máquina	Núcleos	RAM (GB)	Discos Duros	Frecuencia (Ghz)
compute6	12	64	1 (2.3 TB)	2.50
compute7	6	32	4 (3.7 TB)	3.07
compute8	16	64	4 (2.5 TB)	2.60

Tabla 5.2: Catálogos de imágenes satelitales de la AEM.

Imágenes satelitales			
	TERRA	AQUA	Landsat 5
Tamaño (GB)	393.14	172.15	448.246
Número de productos	1966	1011	1662
Fecha de adquisición	Julio 2008 - Mayo 2011	Diciembre 2010 - Septiembre 2013	Julio 2007 - Marzo 2016
Características de las imágenes			
	TERRA	AQUA	Landsat 5
Media (MB)	204.76	174.53	275.67
Mediana (MB)	185.35	181.09	276.06

5.1.1 Repositorios de datos utilizados

Para las pruebas de rendimiento de las soluciones construidas con el modelo propuesto se utilizaron dos repositorios de datos. El primero contiene imágenes satelitales capturadas por la antena AEM-ERIS. En la Tabla 5.2 se muestra un resumen de las características de las imágenes utilizadas. En total se cuentan con 1013.536 GB de productos, provenientes de tres sensores: Aqua, Terra y Landsat5.

El segundo repositorio de datos contiene datos adquiridos desde Estaciones Meteorológicas Automáticas (EMAS) de la CONAGUA, el cual se compone de los datos de temperatura máxima y mínima, así como de precipitación, de las estaciones desplegadas en territorio mexicano. Este conjunto de datos tiene un tamaño de 2 GB de datos no estructurados (texto plano).

5.2 Diseño del prototipo de experimentación

Con el propósito de evaluar el modelo de construcción descrito en el Capítulo 4 se desarrolló un prototipo funcional. El cual permite la construcción de flujos de trabajo basados en microservicios y contenedores virtuales para el modelado de ciclo de vida de los EOPs.

La construcción del prototipo permite evaluar el modelo de manera cuantitativa, evaluando el rendimiento de los flujos de trabajo construidos con el prototipo.

Los componentes del prototipo fueron creados utilizando contenedores virtuales en una nube privada gestionada por el orquestador de contenedores Docker Swarm. Este prototipo recibe como entrada una notación escrita por un usuario, a partir de la cual se construye el flujo de trabajo mediante el encademiento de microaplicaciones y microservicios.

Para realizar la transferencia de datos y productos ($I-EOPS$ y $O-EOPs$) entre microservicios remotos se utilizó SkyCDS[28], el cual es un sistema de distribución de contenidos (CDS, por sus siglas en inglés) basado en un modelo de publicación/subscripción de contenido en un ambiente en la nube. En la Figura 5.1 se muestra cómo es realizada la transferencia de datos y metadatos entre los nodos que conforman a la cadena de microservicios desplegada. La transferencia de metadatos entre los nodos se realiza mediante la API RES con la que cuenta cada nodo, mientras que la transferencia de datos se realiza utilizando SkyCDS. Para ello cada nodo publica catálogos en SkyCDS, en donde coloca sus $O-EOPs$, los cuales son enviados a la nube, en donde son resguardados utilizando técnicas de cifrado y replicación de datos. Para que otros nodos tengan acceso los productos generados por este nodo deben de suscribirse a el catálogo de dicho nodo, de esta manera los productos contendidos en el catálogo suscrito serán descargados desde la nube donde esté montado SkyCDS. Este proceso de publicación y suscripción solamente se debe de realizar una vez por cada catálogo, de esta manera los productos estarán sincronizados en cada uno de los nodos que se han publicado y suscrito catálogos.

Para realizar el despliegue de SkyCDS se utilizó una configuración de cinco nodos de

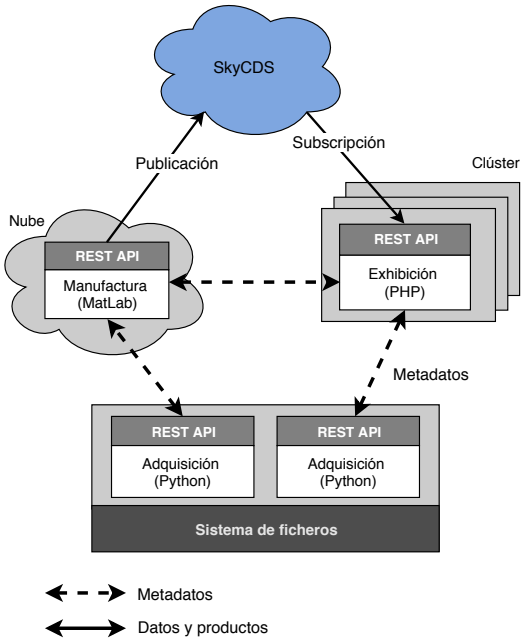


Figura 5.1: Transferencia de datos y productos utilizando SkyCDS.

Tabla 5.3: Características de hardware de los equipos utilizados para el despliegue de SkyCDS.

Máquina	Descripción	RAM (GB)	Procesador	Almacenamiento
Disys0	Manager	12	(1) Intel Xeon CPU E5645 2.40 GHz con 6 núcleos	1 disco duro de 1TB (HDFS) 1 disco duro de 500GB (SO)
Disys1	Nodo 1			
Disys2	Nodo 2			
Disys3	Nodo 3			
Disys4	Nodo 4			

almacenamiento y uno de metadatos. Todos los nodos se encuentran encapsulados en contenedores virtuales, los cuales tienen instalado Apache+PHP5 en su interior. En la Tabla 5.3 se muestran las características de los equipos utilizados para el despliegue de SkyCDS, en donde el equipo con la etiqueta Disys0 fue utilizado como nodo de metadatos y almacenamiento, mientras que el resto de nodos son utilizados como nodos almacenamiento. Cada uno de los microservicios desplegados cuentan con un clon del cliente de SkyCDS, el cual permite realizar las tareas de publicación/subscripción de catálogos, así como la carga y descarga de contenidos.

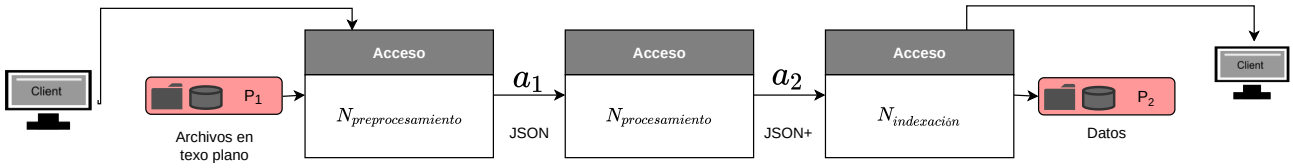


Figura 5.2: Flujo de trabajo generado para realizar la experimentación controlada.

5.3 Pruebas de validación del prototipo

Para evaluar el prototipo desarrollado se realizó una evaluación controlada, en la que se diseño un flujo de trabajo para el procesamiento de lote de metadatos de imágenes satelitales. Dicho flujo de trabajo está compuesto por tres nodos, tales como: preprocesamiento (extracción y transformación de los metadatos), procesamiento (enriquecimiento de información) e indexación en una base de datos.

El flujo de trabajo se implementó utilizando el modelo propuesto en la presente tesis, así como en *Parsl*[3] y *DagOnStar*[50], con el objetivo de realizar una comparación directa de rendimiento de ambas soluciones.

5.3.1 Flujo de trabajo desarrollado para la evaluación

Para llevar a cabo la experimentación controlada se desarrolló un flujo de trabajo, el cual realiza la indexación de imágenes satelitales en una base de datos. Para ello dicho flujo contempla etapas, tales como: $preprocesamiento$, $procesamiento$ e $indexación$. En la Figura 5.2 se presenta el diseño conceptual del flujo de trabajo, que es modelado como:

$$C_1 = (P_1, N_{preprocesamiento}, a_1), (a_1, N_{procesamiento}, a_2), (a_2, N_{indexamiento}, P_2). \quad (5.1)$$

En donde un nodo de $N_{preprocesamiento}$ lee los datos de P_1, el cual es el sistema de archivos, los preprocesa y los coloca en a_1, de donde $N_{procesamiento}$ los lee para procesarlos a_2, para que finalmente $N_{indexamiento}$ los indexe en P_2, el cual es una base de datos.

5.3.1.1. Nodos del flujo de trabajo

Cada una de las etapas del flujo de trabajo descrito fue modelado como un nodo del modelo propuesto. La configuración de cada uno de los nodos es:

- **Nodo de preprocesamiento ($N_{preprocesamiento}$) :** Nodo de preprocesamiento, en donde los metadatos son extraídos y leídos para transformarlos de un formato no estructurado (texto plano) a uno estructurado (JSON). Este proceso de transformación incluye tareas, tales como: el análisis de los archivos para la extracción de los metadatos y la unificación de formatos (por ejemplo, que se utilice un único formato para la representación de las coordenadas).

  En la Figura 5.3 se muestra el diseño del microservicio $N_{preprocesamiento}$, el cual implementa un patrón $Manager/Worker$, en donde el manejador es un distribuidor de carga, y los trabajadores son las tareas de extracción y transformación de metadatos, las cuales se encuentran encadenadas. Las salidas de las *Micro-Apps* de transformación se encuentran directamente encadenadas con las entradas del siguiente microservicio.

- **Nodo de procesamiento ($N_{procesamiento}$):** En este nodo las estructuras JSON son recibidas por el siguiente nodo y se les aplica un proceso de enriquecimiento de información, el cual incluye tareas, tales como el cálculo de valores faltantes para algunas imágenes (por ejemplo, el cálculo del punto central de la imagen a partir de las coordenadas de cada esquina) y la consulta en fuentes externas de información para obtener nuevos valores (como la ubicación en lenguaje natural de cada punto de coordenadas de las imágenes).

  En la Figura 5.3 se muestra el diseño del microservicio $N_{procesamiento}$, el cual recibe las salidas del microservicio $N_{preprocesamiento}$, y las distribuye entre N trabajadores de una manera balanceada. Cada trabajador realiza la tarea de enriquecimiento de los metadatos y entrega sus resultados a un consolidador, quien genera un sólo archivo JSON, que es enviado al microservicio de indexamiento para su preservación.

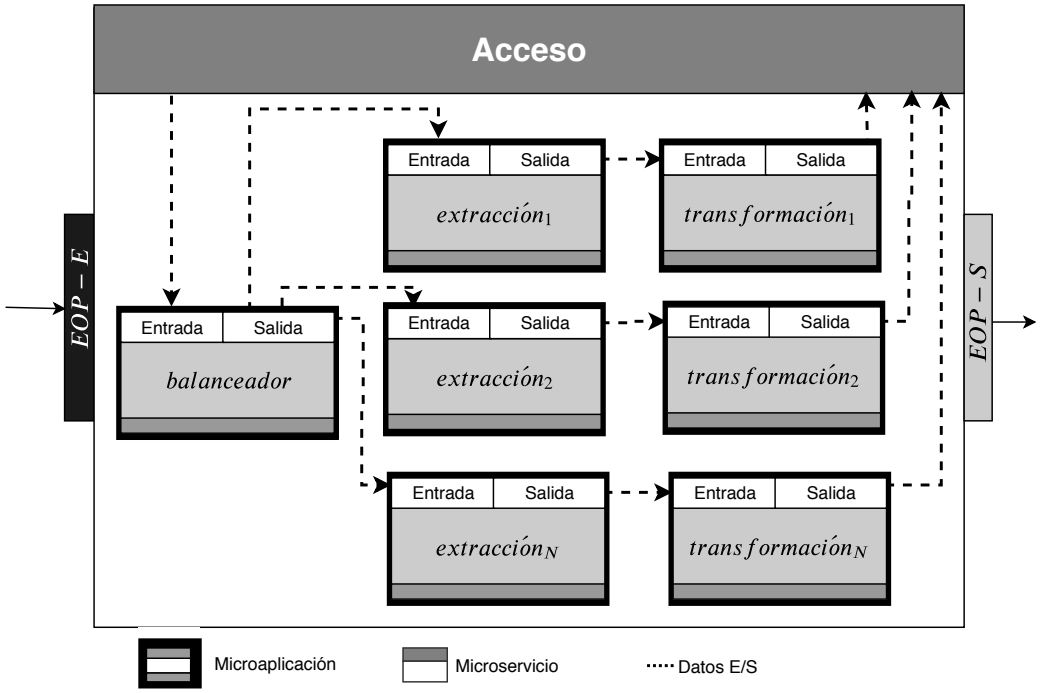


Figura 5.3: $Nodo$ para el preprocesamiento de metadatos.

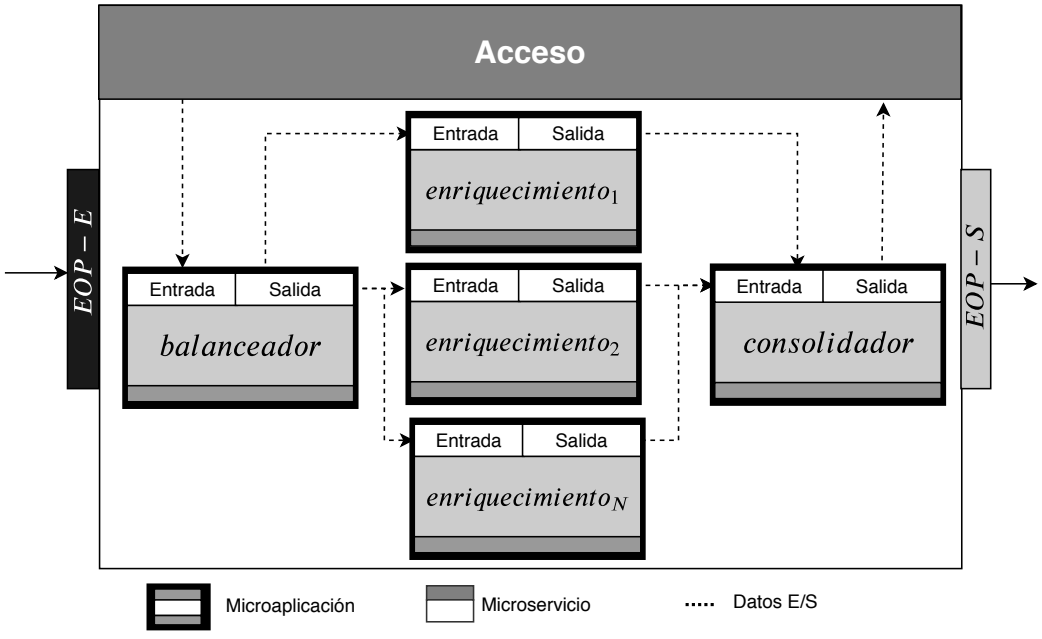


Figura 5.4: $Nodo$ para el procesamiento de metadatos.

- **Nodo de indexamiento ($N_{indexamiento}$):** En este nodo las estructuras JSON enriquecidas son indexadas en una base de datos SQL tradicional (postgresql). Este microservicio/nodo solo cuenta con una *Micro-App* en su interior, la cual se conecta directamente con la base de datos, para realizar la preservación de los metadatos extraídos y procesados en las etapas anteriores. El presente nodo implementa en su interior un balanceador de carga el cuál le permite recibir y atender peticiones concurrentes de manera simultanea.

5.3.2 Definición de pruebas de evaluación controlada

Para realizar la evaluación controlada del prototipo, se utilizó el flujo de trabajo descrito en la Sección 5.3.1. En este escenario se incluyen experimentos enfocados en el rendimiento del prototipo, así como una evaluación directa con dos motores de flujos de trabajo (*DagOnStar* y *Parsl*).

Para lo anterior se realizó un variación experimental, en la que en cada experimento lanzado, se varío en cada nodo el número de trabajadores desplegados en los patrones. Como conjunto de datos se utilizaron los archivos de metadatos de las imágenes satelitales descritas en la Sección 5.1.1.

La evaluación controlada se dividió en dos etapas de experimentación, en la que se evalúa el rendimiento del modelo mediante el ajuste de los parámetros de evaluación.

1. En la primera etapa se evaluó el prototipo desarrollado utilizando el flujo de trabajo descrito la Sección 5.3.1, con el objetivo de encontrar la configuración con el mejor rendimiento en términos del tiempo de servicio y *throughput*.

2. En la segunda etapa, se comparó la mejor configuración del prototipo con dos motores de flujos de trabajo del estado del arte: *Parsl*[3] y *DagOnStar*[50], con el objetivo de realizar una comparación directa de rendimiento de ambas soluciones.

5.3.3 Métricas de evaluación

Las métricas utilizadas para la evaluación de los experimentos realizados se describen a continuación:

- **Tiempo de servicio**: Representa el tiempo requerido por un servicio para completar una tarea dada.
- **Tiempo de respuesta**: Representa el tiempo observado por los usuarios finales desde que envían una solicitud a un servicio, hasta que este es completado.
- Tasa de procesamiento o $Throughput$ (T). Es la cantidad máxima de datos procesados por unidad de tiempo.
- Porcentaje de ganancia (PG(Tx_{s_1}, T_{s_2})). Representa el porcentaje de ganancia obtenida por una solución, en el tiempo de respuesta/servicio, con respecto al tiempo de respuesta/servicio de otra solución s_2 (Tx_{s_1} con respecto a Tx_{s_2}). Se obtiene de la siguiente manera:

$$PG(Tx_{s_1}, T_{s_2}) = 100 * \frac{Tx_{s_1} - T_{s_2}}{Tx_{s_1}}. \tag{5.2}$$

  Si es positiva, la segunda solución es mejor, en caso contrario la primera solución es la mejor.

5.4 Resultados preliminares

En esta sección se presenta un análisis de los resultados de la evaluación experimental del prototipo diseñado. Para evaluarlo se implementó el patrón descrito en la Sección 5.3.1 y el conjunto de datos que incluye todos los archivos de metadatos de las imágenes satelitales de la AEM.

La Figura 5.5 muestra en el eje horizontal el número de contenedores en paralelo lanzados en cada nodo del flujo de trabajo, mientras que en el eje vertical derecho muestra el tiempo de respuesta de las

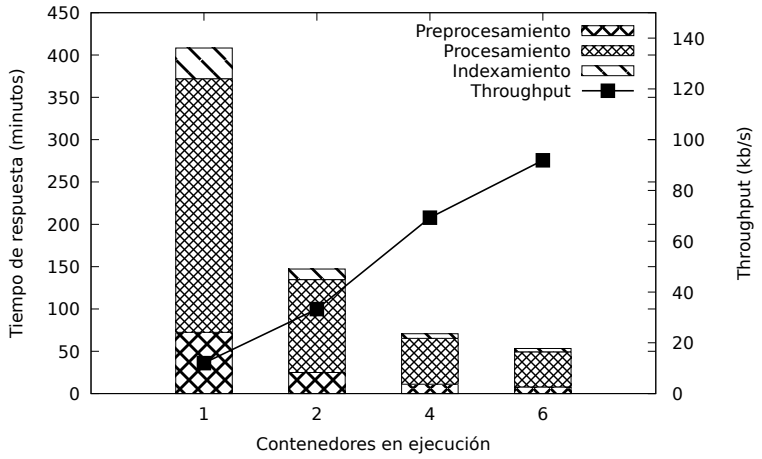


Figura 5.5: Tiempo de servicio obtenido por cada nodo del flujo de trabajo.

diferentes soluciones construidas. En el eje vertical izquierdo de la gráfica se observa el *Throughput* o rendimiento de las soluciones. En la gráfica los tiempos obtenidos por cada tarea del nodo se muestran de un color distinto. Como se puede observar, la tarea que consume el mayor tiempo de respuesta del nodo el la tarea de procesamiento de datos.

Como se puede observar en la Figura 5.5, cuando las tareas en el nodo despliegan patrones M/W utilizando 6 contenedores como trabajadores se obtiene una ganancia del $86\,\%$ comparado con la versión de un solo trabajador. Específicamente, los porcentajes de mejora son del $89.12\,\%$, $86.14\,\%$ y del $89.12\,\%$ para las tareas de preprocesamiento, procesamiento e indexamiento respectivamente.

Al analizar los resultados obtenidos, se puede resumir que utilizando el patrón M/W implementado en los dos primeros nodos del flujo de trabajo mejora notablemente el tiempo de respuesta en comparación de su versión secuencial. También se observó que el tiempo de respuesta de la tarea de indexación disminuye a pesar de que esté recibiendo peticiones concurrentes desde los trabajadores del nodo de procesamiento, lo que sugiere que el balanceador de carga implementado logra atender y procesar las peticiones sin impactar en el tiempo de respuesta e inclusive reduciéndolo.

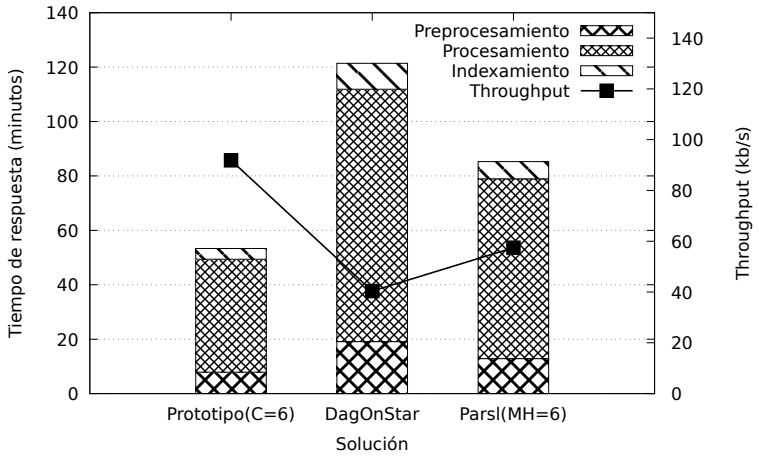


Figura 5.6: Comparación del tiempo de respuesta entre el prototipo desarrollado, *DagOnStar* y *Parsl*.

5.4.1 Discusión de los resultados de comparación con motores de despliegue de flujos de trabajo

En la presente sección se presenta un análisis de resultados de la comparación directa del prototipo desarrollado en la presente tesis contra dos soluciones encontradas en el estado del arte: *DagOnStar* y *Parsl*.

La comparación se realizó utilizando la mejor configuración del prototipo de la fase de evaluación anterior, la cual es utilizando 6 contenedores en el patrón M/W. En el caso específico de *Parsl*, el motor permite establecer el número máximo de hilos en ejecución en paralelo (MH) cuando se ejecutan flujos de trabajo utilizando paralelismo basado en tareas. En la presente experimentación, se optó por utilizar 6 hilos como máximo ($MH = 6$), con el objetivo de evaluar una solución con 6 hilos en paralelo con la mejor configuración (6 contenedores) obtenida del modelo propuesto.

En la Figura 5.6 se muestra en el eje vertical izquierdo el tiempo de respuesta para las tres soluciones implementadas (eje horizontal), mientras que en el eje vertical derecho se observa el rendimiento o *througput* de las soluciones. Como se puede observar, el patrón M/W utilizando 6 contenedores en paralelo produce una mejora en el tiempo de respuesta del $37\,\%$ en comparación con *Parsl*, y una mejora del $56\,\%$ en comparación con *DagOnStar*.

La ganancia en comparación con *DagOnStar*, se explica debido a que este motor de flujos de trabajo lanza un hilo por cada tarea de procesamiento que realiza, lo cual genera una sobrecarga en el equipo en el que se esta desplegando, mientras que el modelo propuesto lanza solamente n trabajadores, que tienen una carga de datos balanceados, lo cual permite que se puedan manejar de manera eficiente los recursos computacionales con los que se cuentan.

Por otro lado, Parsl resulta tener mejor rendimiento que DagOnStar debido a que en esta solución el procesamiento, a diferencia de DagOnStar, esta limitado a solo n hilos, por lo que una vez ejecutados n hilos, el hilo $n+1$ será ejecutado hasta que alguno de los n hilos termine de ejecutarse. Lo anterior, permite gestionar de mejor manera los recursos con los que se cuentan, sin embargo, por cada tarea a procesar, se tiene que levantar un hilo, lo que agrega una sobrecarga adicional al procesamiento de los datos. En contraste, en el prototipo propuesto en esta tesis, solo se lanzan n hilos, que corresponde con el número de trabajadores configurados, los cuales reciben un lote de tareas a procesar. Dichos lotes, son generados por el $Manager$ del patrón, balanceando la carga entre los trabajadores.

6

Evaluación del prototipo para caso de estudio: desde una imagen *raster* hasta la exhibición

En el presente capítulo se describe el desarrollo de *GeoEris*, el cual es una plataforma construida utilizando el modelo propuesto. Esta plataforma fue construida para la Agencia Espacial Mexicana (AEM) para el manejo de EOPs capturados y producidos por una antena llamada AEM-ERIS [21]. En total se desarrollaron tres soluciones, las cuales realizan la adquisición (*GeoAcq*), manufactura (*GeoProc*) y exhibición (*GeoEris*) de los EOPs en su ciclo de vida.

6.1 GeoEris: Diseño e implementación

En la Figura 6.1 se muestra una arquitectura de pila de tres servicios, de la plataforma desarrollada, la cual fue creada para el manejo del ciclo de vida de los EOPs. Se desarrolló una suite de tres servicios, llamados *GeoAcq*, *GeoProc* y *GeoEris*. Estos servicios fueron encapsulados en $Nodes$, los cuales son

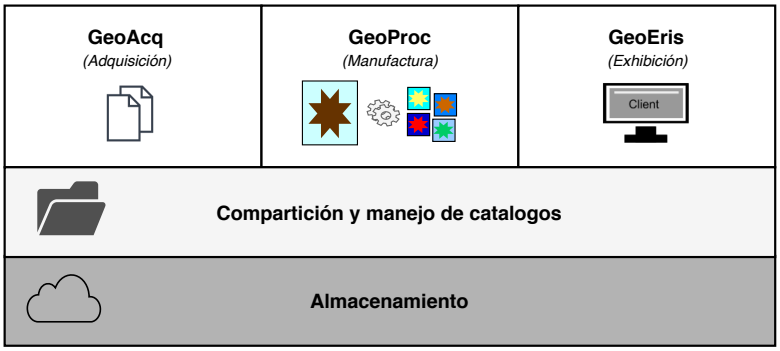


Figura 6.1: Arquitectura en pila desarrollada para el manejo de ciclo de vida de los EOPs.

agrupados, generando diferentes combinaciones para la creación de cadenas de valor.

Un nodo basado en un federación de almacenamiento en la nube $FedIDS$ [30] fue desarrollado para la compartición de EOPs y el transporte de catálogos de EOPs (ver módulo de almacenamiento en la Figura 6.1). Este servicio es consumido por *GeoAcq*, *GeoProc* y *GeoEris*.

El servicio *GeoAcq* está a cargo del manejo de la etapa de *Adquisición*, donde los metadatos e imágenes *raster* son obtenidos de los servidores de la antena en la forma de abstracciones conocidas como catálogos. El servicio $FedIDS$ está a cargo de manejar la etapa de *preservación* donde los EOPs y los repositorios en línea son almacenados en una manera confiable y segura. *GeoProc* está a cargo del manejo de la etapa de *Manufactura*, donde automáticamente los EOPs son corregidos a partir de una imagen *raster*, los cuales son utilizados como imágenes de segundo nivel. Este servicio considera el despliegue automático de aplicaciones para la construcción de mapas temáticos y productos derivados. La etapa de manufactura está conectada con la etapa de exhibición, donde los nuevos EOPs manufacturados (corregidos/productos derivados) son almacenados en repositorios de la plataforma en una manera automática y transparente. *GeoERIS* está a cargo de la etapa de *Exhibición*, la cual esta basada en geoportales para que los usuarios finales puedan descubrir EOPs (imágenes *raster* y productos derivados) y para que puedan crear mapas de cobertura de la tierra (mosaicos, solapamientos y mosaicos solapados).

6.1.1 Servicio GeoAcq

El servicio *GeoAcq* considera la adquisición y procesamiento de los metadatos, así como la indexación y preservación de los EOPs producidos en *catálogos Pub/Sub*, el *procesamiento de metadatos* y la *preservación de EOPs*.

6.1.1.1. Nodo para el manejo de catálogos

Este nodo incluye *Micro-Apps* para el manejo de catálogos asociados con las imágenes satelitales adquiridas por la antena AEM-ERIS. El $I - EOP$ de este nodo está enlazado con un servidor de la antena ERIS que almacena las imágenes *raster*.

Una *Micro-App* fue desarrollada para la *clasificación* de EOPs. Dicha *Micro-App* implementa un sistema de archivos reducido para el manejo de los EOPs producidos por la antena AEM-ERIS. En este sistema de archivos, son creados tantas carpetas como sensores de la antena. En cada carpeta, es creada una subcarpeta para cada tipo de archivos adquiridos por el servidor de la antena. La primera subcarpeta contiene todas las imágenes *raster*, incluyendo las bandas de la imagen. La segunda subcarpeta, es para los archivos de metadatos asociados con la imagen. La tercera es para una imagen de baja calidad creada como miniatura, y que es requerida en la etapa de exhibición. La última subcarpeta, contiene diferentes archivos que son creados dependiendo las necesidades de los usuarios finales.

La *Micro-App* de *Pub/Sub* crea mapas para indexar los EOPs previamente clasificados. Cada mapa es agregado a una lista de *catálogos*; como resultado cualquier aplicación puede acceder a todos los catálogos, carpetas o EOPs utilizando estos mapas. Los usuarios finales pueden utilizar las operaciones de publicación y localización para compartir un EOP con un socio, y ellos pueden tener acceso a otros catálogos realizando las operaciones de suscripción. Hasta este punto todos los EOPs son enviados al nodo de almacenamiento para que los usuarios finales puedan adquirirlos posteriormente.

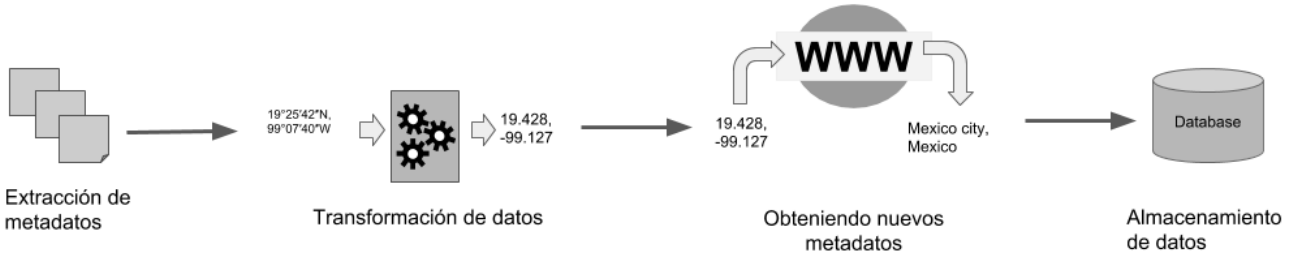


Figura 6.2: Proceso de construcción de metadatos.

6.1.1.2. Nodo para el procesamiento de metadatos

El nodo para el procesamiento de metadatos incluye tareas para la extracción, manejo y procesamiento de los archivos clasificados e indexados que se encuentran asociados a cada imagen en un catálogo.

Una *Micro-App* para la extracción y transformación de metadatos, convierte todos los archivos no estructurados y semiestructurados en archivos estructurados (JSON). Los datos de esta estructura son extraídos e inyectados en una base de datos, manejada por otra *Micro-App*. En esta *Micro-App*, los archivos de metadatos son analizados para extraer un conjunto de duplas de la forma $< key, value >$, los cuales son almacenados en memoria para completar el proceso de construcción de metadatos, mostrados en la Figura 6.2.

Una *Micro-App* para la transformación de datos homogeneiza los diferentes formatos de datos para valores con el mismo significado (por ejemplo, coordenadas, temperatura y fechas) para todos los datos extraídos de los archivos de metadatos. Por ejemplo, uno de los factores más relevantes en los archivos de metadatos es la representación de coordenadas, que pueden ser representadas en grados decimales o en formato de Grados/Minutos/Segundos (DMS, por sus siglas en inglés). El módulo de transformación convierte las coordenadas DMS en un formato decimal, que es almacenado en la base de datos.

Posteriormente, una *Micro-App* para el *enriquecimiento de los datos* recibe los metadatos homogeneizados para realizar un proceso de enriquecimiento de datos utilizando fuentes externas.

Por ejemplo, el nombre del lugar a donde hacen referencia las coordenadas son obtenidas en este proceso. Esta información ayuda a los usuarios finales a identificar un área descrita por una imagen. Esta *Micro-App* regresa un archivo JSON con la dirección en lenguaje natural de las coordenadas geográficas, para que los usuarios puedan hacer consultas utilizando el nombre la ciudad, estado y/o país [1].

6.1.1.3. Adquisición de metadatos como un patrón M/W

El flujo mostrado en la Figura 6.2 fue encapsulado en una tarea de *GeoAcq* como $t_{metadata}$. Para acelerar la adquisición de metadatos se utilizó el patrón M/W, descrito previamente para ejecutar $t_{metadata}$ como Pt (tarea en paralelo). Este patrón, además incluye un balanceador de carga para distribuir la lista de $EOPList$ entrante, entre todas las tareas $t_{metadata}$ replicadas. En este caso, $EOPList$ contiene todos los archivos de metadatos a ser procesados. El nodo regresa una lista a través de una arista o un recurso $Pool$, como una base de datos. El diseño conceptual de este nodo es similar al mostrado en la Figura 4.6, con la diferencia de que ahora todos los trabajadores regresan al manejador un conjunto de archivos JSON que son consolidados en un solo archivo JSON.

6.1.2 Servicio GeoProc

GeoProc es un servicio de manufactura de productos derivados, que transforma imágenes *raster* en nuevos EOPs (productos derivados con un valor agregado).

Este servicio incluye dos nodos: el primero para el preprocesamiento de las imágenes *raster* y el segundo para el procesamiento de las imágenes para crear mapas temáticos.

[1] Dependiendo de las coordenadas, en algunos casos los valores no pueden ser encontrados, resultando en valores nulos.

Tabla 6.1: Productos derivados creados por el nodo de procesamiento *GeoProc*.

Producto derivado	Aplicación
Índice de vegetación de diferencia normalizada (NDVI, por sus siglas en inglés)	Es utilizado para estimar la cantidad, calidad y desarrollo de vegetación [56].
Índice de agua de diferencia normalizada (NDWI, por sus siglas en inglés)	Es utilizado para conocer el estrés hídrico[44].
Índice de nieve de diferencia normalizada (NDSI, por sus siglas en inglés)	Es utilizado para diferenciar la nieve de otros elementos, como las nubes [34].
Índice de radio de quemaduras (NBR, por sus siglas en inglés)	Sirve para identificar las áreas quemadas [22].
Índice de resistencia atmosférica visible (VARI, por sus siglas en inglés)	Radiación fotosintética activa estimada absorbida en la cubierta vegetal [27].

6.1.2.1. Nodo de preprocesamiento

Este nodo fue creado para realizar la corrección automática de las imágenes satelitales. Este sistema incluye *Micro-Apss* para realizar las correcciones atmosféricas y radiométricas de las imágenes para todas las misiones Landsat 1-8.

6.1.2.2. Nodo de procesamiento

El nodo de procesamiento incluye *Micro-Apps* para realizar el cálculo de índices y generación de productos derivados de las imágenes Landsat. En la Tabla 6.1 se muestran los productos derivados que se pueden generar con las microaplicaciones encapsuladas en *GeoProc*. En la Figura 6.3 se muestra una cadena de valor donde una imagen es utilizada como $EOP - E$ del nodo de preprocesamiento, el cual produce una imagen corregida como un nuevo EOP el cual utilizado como $I - EOP$ por nodo de procesamiento para generar un mapa temático.

Figura 6.3: De una imagen *raster* a un producto derivado. a) Imagen *raster*. b) Imagen corregida. c) Producto derivado.

6.1.3 Servicio GeoEris

GeoEris es un servicio de exhibición de EOPs basado en un geoportal web[2]. En *GeoEris* los microservicios reciben solicitudes de otros microservicios para realizar tareas tales como: la inyección de los metadatos de nuevos productos en la base de datos, el manejo de los metadatos de los productos derivados, así como el manejo de la publicación y suscripción de catálogos de los usuarios.

En la Figura 6.4 se muestra la arquitectura de microservicios desarrollada para crear *GeoEris*. Los usuarios pueden acceder al geoportal utilizando una aplicación web conectada con el nodo *Front-end*, donde las solicitudes son recibidas por un *proxy* y las redirecciona a una pila de tres servicios. En esta pila, la *Micro-App* de acceso está a cargo de validar las credencias y los tokens de acceso de los usuarios y aplicaciones para poder acceder a los otros dos servicios en la pila. La *Micro-App* de manejo incluye todos los módulos y funciones para realizar tareas como la inyección de metadatos, así como la búsqueda de imágenes. Finalmente, la *Micro-App* de almacenamiento está compuesta por la base de datos del geoportal. En esta pila de servicios, el servicio *Back-end* entrega los resultados al servicio *Front-end*.

Siguiendo el modelo propuesto la arquitectura de *GeoEris* mostrada en la Figura 6.4 puede ser expresada como:

[2]Disponible en http://www.adaptivez.org.mx/AEM-Eris/.

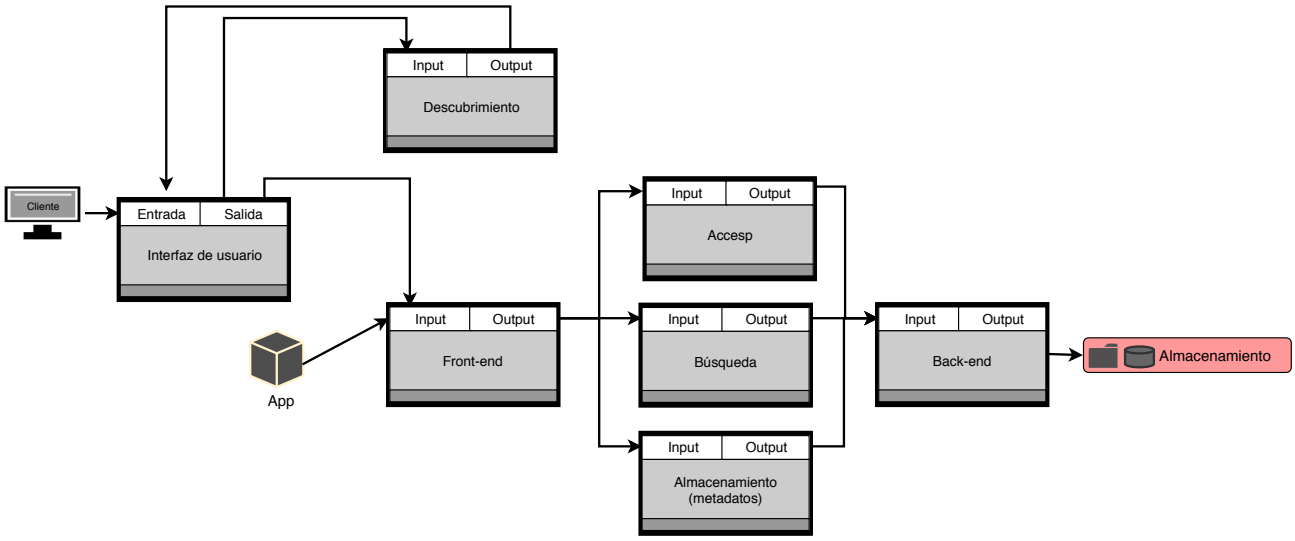


Figura 6.4: Representación conceptual de la arquitectura de microservicio de *GeoEris* para el descubrimiento y búsqueda de EOPs.

$$N_{geoeris} = (t_{gui} \Leftrightarrow t_{discovering}), (t_{gui} \Rightarrow t_{frontend}), \\ (t_{frontend} \Rightarrow t_{access,searching,storage} \Rightarrow t_{backend}) \tag{6.1}$$

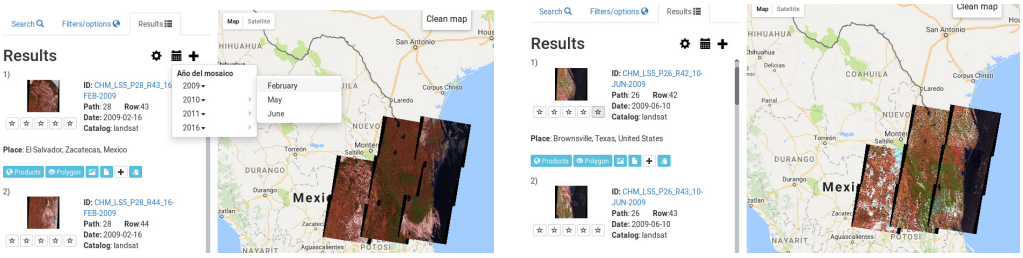
6.1.3.1. Nodos para la búsqueda y descubrimiento de EOPs

El geoportal incluye nodos para la búsqueda de EOPs basado en parámetros espaciales y temporales. Herramientas como selectores de fechas, dibujado de formas (polígonos, círculos y rectángulos), o la búsqueda de un lugar en específico escribiendo sus coordenadas o mediante su nombre en lenguaje natural, son incluidas.

Cuando el usuario utiliza *GeoEris* puede buscar EOPs individuales, como imágenes *raster*, metadatos y productos derivados indexados y creados por *GeoAcq* y *GeoProc*. Además, ellos pueden buscar por una combinación de productos, tales como, mosaicos de imágenes (basados en parámetros espaciales), solapamientos de escenas (basados en parametros temporales) y mosaicos solapados (basados en parámetros espaciales y temporales). Los usuarios finales, además pueden crear estos EOPs en línea, a través del geoportal (ver Tabla 6.2 para conocer la descripción de los EOPs

Tabla 6.2: EOPs generados bajo demanda.

Producto	Descripción
Mosaicos	Producto generado para un área de cobertura en la superficie de la tierra, seleccionada por el usuario. Las imágenes que están dentro de esa área son ordenadas secuencialmente de acuerdo con sus coordenadas.
Imágenes solapadas	Las imágenes satelitales son agrupadas por sus valores *Path/Row* y ordenadas de acuerdo a su fecha de adquisición. Una línea de tiempo para esta área de cobertura es creada para que los usuarios puedan observar los cambios a través del tiempo de dicha área.
Mosaicos solapados	Este producto es creado utilizando la combinación de los dos primeros productos. Los mosaicos del área de cobertura delimitada por el usuario son generados y además se ordenan respecto a su mes y año de adquisición. Como resultado, se genera un línea de tiempo de mosaicos, la cual es mostrada a los usuarios para que observen los cambios de las áreas de cobertura a través del tiempo (ver Figura 6.5).



(a) Imágenes satelitales de febrero de 2008. (b) Imágenes satelitales de junio de 2008.

Figura 6.5: Mosaicos solapados generados con las imágenes de la misma región en diferente mes.

considerados en este servicio.).

6.1.3.2. Descubrimiento de EOPs analizando la actividad y preferencias de los usuarios finales

GeoEris incluye *Micro-Apps* para la generación de recomendaciones basadas en la actividad de los usuarios, la cual se encuentra almacenada en archivo en el servicio de almacenamiento. Este módulo registra las acciones realizadas por los usuarios finales, considerando cada solicitud enviada a la *Micro-App* de manejo así como los clics de los usuarios en el geoportal.

El módulo de recomendaciones genera una lista de diferentes EOPs indexados en el geoportal, que pueden ser de interés para el usuario final con base a su actividad en el geoportal.

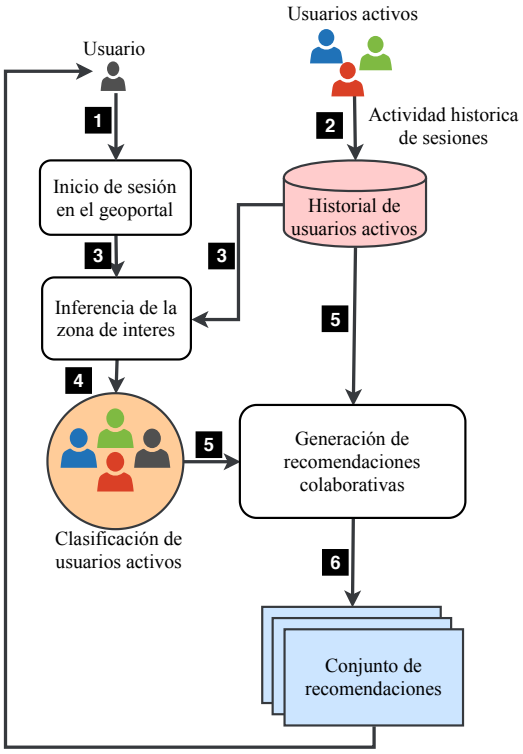


Figura 6.6: Flujo para crear recomendaciones colaborativas.

Para la generación de las recomendaciones se utilizó una técnica de filtrado colaborativo, siguiendo el enfoque basado en memoria (ver Figura 6.6). Una recomendación es producida a través de diferentes fases, que incluyen el almacenamiento de la actividad de los usuarios finales, la inferencia de la zona geográfica de interés del usuario, la clasificación de los usuarios finales utilizando técnicas de agrupamiento y la entrega de las recomendaciones a los usuarios finales a través del geoportal. Un perfil es creado para cada usuario final en este procedimiento, el cual incluye todas las acciones que ha realizado el usuario, como el número de clics, vistas y descargas para cada EOP, así como las configuraciones de las consultas que ha realizado en el geoportal. Este perfil es construido la primera vez que el usuario accede al geoportal, y es actualizado cada que utiliza el servicio.

Para inferir la zona de interés de los usuarios se utilizan las configuraciones espaciales utilizadas en sus consultas. Por ejemplo, cuando el usuario realiza búsquedas utilizando polígonos, se almacenan las coordenadas de cada punto del polígono, mientras que, si realiza búsquedas con círculos, se

almacenan las coordenadas del centro, así como el radio del círculo. En este contexto, los centroides de las áreas donde los usuarios finales realizan sus búsquedas representan una posible zona de interés para los usuarios finales.

Clasificación de usuarios activos

Los usuarios son clasificados para la generación de recomendaciones en el geoportal a partir de los intereses de otros usuarios similares al que se le están generando las recomendaciones. Los centroides calculados para cada usuario, son pasados por un algoritmo de agrupamiento conocido como DBSCAN, que fue incluido en el módulo de recomendaciones para crear usuarios que comparten preferencias similares utilizando la distancia Haversine.

Cuando los usuarios son creados, las recomendaciones para un usuario son construidas y enviadas al geoportal. Esta tarea es realizada por el Algoritmo 1, el cual recibe como entrada el grupo en el cual el usuario fue clasificado y el conjunto de EOPs con los que el usuario tiene alguna interacción (clic, vistas o descargas). Este algoritmo selecciona los EOPs más populares a partir de la actividad histórica de los usuarios en el grupo.

6.2 Evaluación experimental del estudio de caso

El prototipo en este estudio de caso fue evaluado en dos fases: en la primera se evaluaron los servicios desarrollados (*GeoAcq*, *GeoProc* y *GeoEris*) de manera aislada, mediante la creación de cadenas locales de *MicroApps*; mientras que en la segunda fase se evaluaron las cadenas de valor creadas mediante la combinación de estos servicios.

6.2.1 Análisis de los servicios de una manera aislada

En esta sección se describen los resultados obtenidos de los experimentos realizados para evaluar cada uno de los servicios desarrollados utilizando el modelo propuesto en la presente tesis.

Algorithm 1 Generación de recomendaciones colaborativas.

Require: $N = \{n_1, n_2, n_3, ..., n_n\}$: grupo de vecinos cercanos al usuario u;
$I = \{i_1, i_2, i_3, ..., i_m\}$: conjunto de EOP consultados por u.
Ensure: $R = \{r_1, r_2, r_3, ..., r_k\}$: conjunto de EOPs recomendados para u.

```
items = {}
for all n ∈ N do
    cliqueados = n.obtenerEOPcliqueado()
    descargos = n.obtenerEOPsDescargados()
    for all EOP ∈cliqueados or EOP ∈ descargados do
        if EOP ∉ I then
            incrementarContador(EOP.id)
        end if
    end for
end for
ordenar(items)
R = seleccionarPopulares(items)
return R
```

6.2.1.1. GeoAcq: extracción y enriquecimiento de los metadatos

En *GeoAcq* se midió el tiempo de respuesta de las cadenas de valor creada por las *Micro-Apps* incluidas en este servicio. Las cadenas incluyen las siguientes etapas:

1. **Adquisición**. Los EOPs son adquiridos desde el sistema de archivos.
2. **Extracción**. Se realiza la extracción de metadatos utilizando el patrón *M/W*.
3. **Enriquecimiento**. Se realiza el calculo de nuevos datos y valores faltantes, como el nombre en lenguaje natural de cada par de coordenadas de la imagen (esquinas y centro).
4. **Indexación y entrega**. Los metadatos son indexados en el nodo que contiene la base datos.

El tiempo de servicio promedio para la extracción de los metadatos fue de 16.16 segundos por imagen, mientras que el tiempo de servicio para el enriquecimiento de los metadatos fue de 14.6 segundos por imagen en promedio. En resumen, el tiempo producido por la cadena de valor fue de 32.32 segundos. Sin embargo, cuando el nodo de extracción de metadatos es ejecutado como un

Tabla 6.3: Tiempo de servicio de *GeoProc*.

Sensor	Bandas	Tiempo de servicio (segundos)						
		Pre	NDVI	NDWI	NDSI	NBR	VARI	Total
Landsat8	7	7.19	0.73	0.69	0.75	0.70	0.81	3.68
Landsat7	6	6.49	0.78	0.75	0.78	0.77	1.02	4.10
Landsat7	6	7.09	0.85	0.81	0.79	0.80	1.32	4.57
Landsat5	7	4.63	0.67	0.66	0.69	0.68	0.76	3.46
Landsat5	7	4.67	0.66	0.60	0.72	0.67	0.74	3.39

patrón de paralelismo M/W el tiempo de servicio es reducido a 0.58 segundos (en la Sección 6.2.2 se dan los resultados detallados del desempeño de *GeoACq* utilizando este patrón).

6.2.1.2. GeoProc: procesamiento de los EOPs

En esta sección se muestra el desempeño obtenido de las cadenas de valor para la corrección de EOPs, así como para la manufactura de productos derivados.

En la Tabla 6.3 se muestra el tiempo de servicio producido por los nodos de preprocesamiento y procesamiento de productos EOPs. El tiempo de servicio promedio para el procesamiento de imágenes Landsat5 fue de 4.63 segundos, y de 6.9 segundos para Landsat 7 y 8. Mientras que para el procesamiento (generación de productos derivados) el tiempo promedio para Landsat5 fue de 3.42 segundos y de 4.11 para Landsat 7 y 8. Esto significa que el nodo de preprocesamiento es el que consume la mayor parte del tiempo de servicio de *GeoProc*, mientras que el nodo de procesamiento consume menos de un segundo para la generación de cada índice.

A partir de los resultados obtenido se desarrolló un patrón de paralelismo *M/W* para el nodo más lento (el de preprocesamiento), reduciendo el tiempo de servicio a 2.6 segundos (ver Sección 6.2.2).

6.2.1.3. GeoEris: exhibición de los EOPs

En la presente sección se analizaron las cadenas de valor creadas en *GeoEris* para el descubrimiento y búsqueda de EOPs, así como para la creación de mapas de cobertura en línea por parte de los usuarios finales.

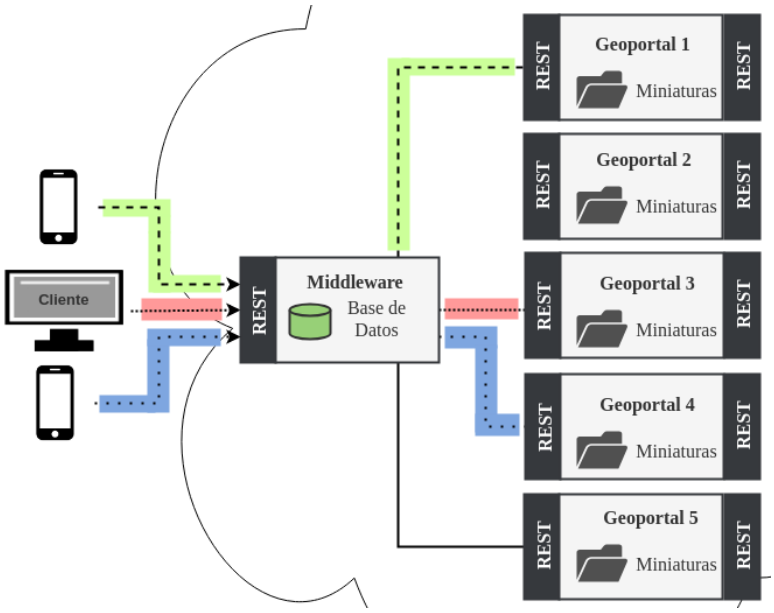


Figura 6.7: Clúster construido con *GeoEris*.

GeoEris fue desplegado en una nube privada utilizando OpenStack Mitaka (ver Figura 6.7). Este servicio fue desplegado como un patrón maestro/esclavo, donde los microservicios que contiene el geoportal fueron clonados cinco veces y desplegados en la nube para construir un clúster de cinco geoportales esclavos y un *middleware*, que es el maestro del clúster. El maestro esta a cargo de la distribución en una manera balanceada de las solicitudes enviadas por los usuarios. El maestro, además incluye una base de datos centralizada que es accedida por cada microservicio en los geoportales.

Un cliente "bot" fue desplegado para crear carga sintética, que representan solicitudes de los usuarios finales, al clúster de geoportales creado. EL cliente envía diferentes tipos de solicitudes para los diferentes tipos de EOPs manejados por el geoportal (imágenes, solapamientos y mosaicos).

En la Figura 6.8 se muestra en el eje vertical, el tiempo de respuesta requerido para el geoportal para mostrar los resultados de EOPs, tales como imágenes *raster*, solapamientos y mosaicos, cuando los usuarios finales seleccionan uno, dos o los tres satélites (eje horizontal) en el geoportal.

Básicamente la Figura 6.8 muestra dos efectos que afectan la experiencia de los usuarios finales al utilizar el servicio. La primera es que el número de resultados incrementa a medida que el usuario

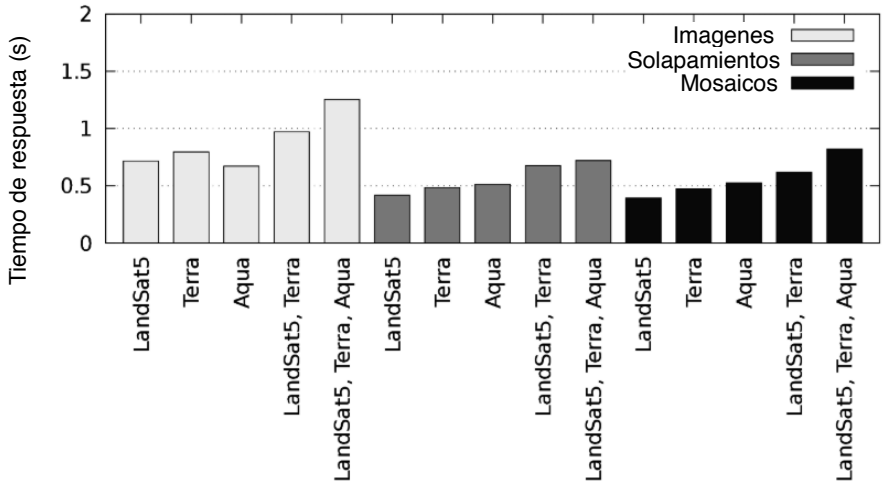


Figura 6.8: Tiempo de respuesta del geoportal para atender solicitudes de diferentes sensores.

selecciona más sensores (catálogos) en el geoportal. El segundo efecto detectado es que la forma seleccionada para realizar la búsqueda afecta en el número de resultados obtenidos por el geoportal. Por ejemplo, fue observado que las búsquedas basadas en polígonos producen tiempos de respuesta mayores a otros tipos de formas (por ejemplo, círculos y rectángulos), dado que el geoportal produce más resultados para este tipo de forma.

En la Figura 6.9 se muestra el número de resultados (en la forma de EOPs) producidos por solicitudes enviadas a *GeoEris*, así como el desempeño de estas solicitudes, en la forma de resultados por segundo. Estas solicitudes son de dos tipos: el primero es una lista de EOPs (I) para cada catálogo (Aqua, Terra y Landsat), y el segundo es la construcción de mapas de cobertura de EOPs en línea (O para solapamientos y M por mosaicos espaciotemporales).

Como se esperaba, cuando se envían solicitudes I, *GeoEris* produce más resultados y el desempeño es mejor que con solicitudes O y M. La construcción de productos M es la más costosa de los dos tipos de EOPs construidos en línea.

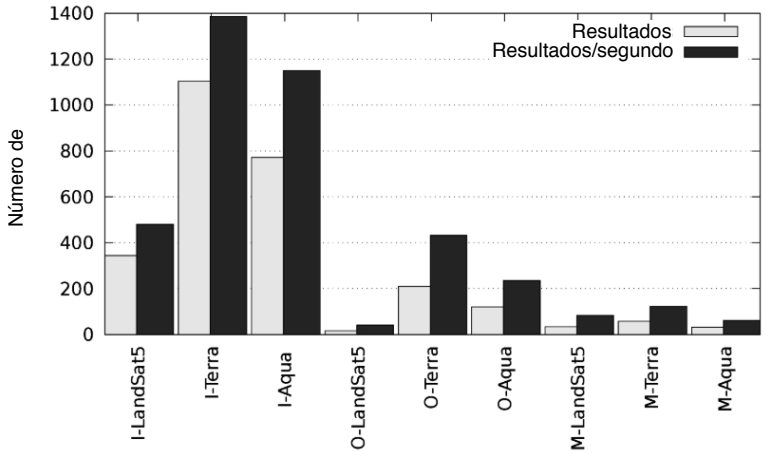


Figura 6.9: Número de resultados obtenidos por consulta para la construcción de mosaicos y solapamientos de EOP en línea.

6.2.2 Análisis de los patrones de paralelismo basados en contenedores

En la presente sección se realizó un análisis del desempeño de los nodos cuando son desplegados utilizando el patrón *M/W*. El propósito de estos experimentos es el de demostrar cómo estos patrones, que forman parte del modelo de construcción, pueden reducir el impacto de los nodos más lentos de *GeoAcq*, *GeoProc* y *GeoEris*.

En la Figura 6.10 se muestra, en el eje vertical, el desempeño de *GeoAcq* cuando son procesados los $4,639$ productos de los tres catálogos que se tienen (ver Tabla 5.2). Para realizar estas pruebas se configuró un patrón de paralelismo incluyendo diferente número de trabajadores (eje horizontal). El eje vertical izquierdo muestra el tiempo de respuesta de *GeoAcq* para la extracción los metadatos de los archivos, mientras que en el eje vertical derecho se muestra el *throughput* de la solución.

Como se puede observar, el patrón de paralelismo basado en contenedores reduce significativamente el tiempo de servicio de esta etapa del ciclo de vida de los EOPs, lo cual incremente el número de archivos procesados por segundo. Como se esperaba, mientras más trabajadores se desplieguen con el patrón, es mayor la mejora en el desempeño de la etapa. Por ejemplo, el patrón

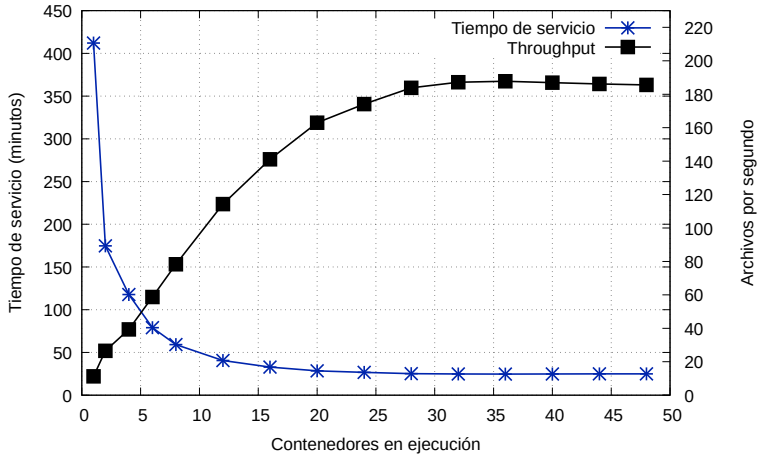


Figura 6.10: Tiempo de servicio y throughput de *GeoAcq* el nodo de preprocesamiento de metadatos utilizando diferente número de contenedores en paralelo.

M/W corriendo con 48 trabajadores en paralelo produce una mejora en el tiempo de respuesta del 93.33 % en comparación de ejecutarlo con un solo trabajador. La mejora depende de los recursos en donde sea desplegado el patrón, dado que se ha observado que el número de núcleos en la máquina física impone una restricción al número de trabajadores que se pueden desplegar.

Es importante mencionar que el código para el preprocesamiento utilizado en el patrón fue encapsulado en una *Micro-App* (contenedor virtual) y fue replicado en un Nodo sin la necesidad de modificar el código, lo cual significa que los patrones de paralelismo pueden ser desplegados en cualquier infraestructura que cuente con la plataforma Docker.

En la Figura 6.11 se muestra en el eje vertical el tiempo de servicio del nodo de preprocesamiento de *GeoProc* cuando un patrón *M/W* es aplicado para la corrección de 50 EOPs y el número de trabajadores es incrementado (eje horizontal). Este tiempo considera la corrección de cada banda de un EOP y la creación de una nueva versión del EOP (imagen corregida).

Como se puede observar, el patrón implementado en *GeoProc* tiene los mismos efectos al implementado en *GeoAcq*. El tiempo de servicio disminuye a medida que se incrementa el número de contenedores (trabajadores) corriendo en paralelo. Por ejemplo, para 16 contenedores en paralelo, se observó una mejora en el tiempo de servicio del $85.31\,\%$ en comparación de la versión de un solo

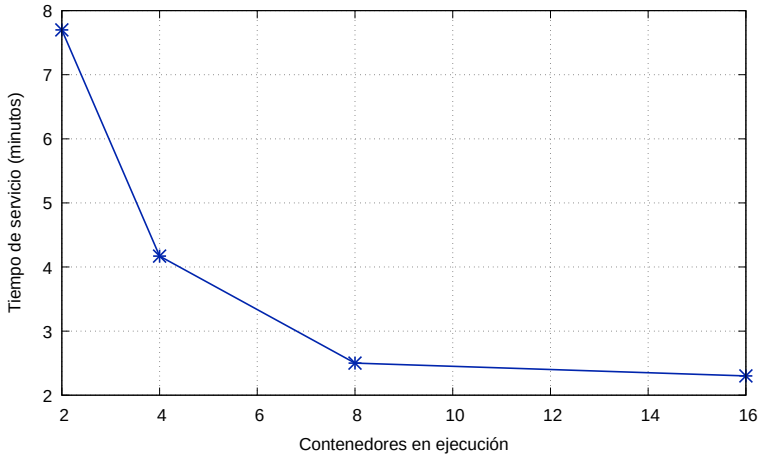


Figura 6.11: Tiempo de servicio de *GeoProc* cuando el nodo preprocesamiento realiza las correcciones radiométricas para Landsat5 utilizando diferente número de contenedores en paralelo.

contenedor.

En la Figura 6.12 se observa el tiempo de respuesta (eje vertical) producido por *GeoEris* utilizando dos configuraciones: con una y tres réplicas para servir diferente número de solicitudes concurrentes (eje horizontal). Como se puede observar, la configuración incluyendo tres réplicas de *GeoEris* reduce el tiempo de servicio en proporción con el incremento en el número de solicitudes concurrentes, en comparación con la solución tradicional, basada en un sólo servicio. La mejora en el tiempo de respuesta para 100 solicitudes concurrentes fue de 34.10 % en comparación con la versión de un solo servicio de *GeoEris*.

Las organizaciones pueden configurar el número de réplicas en los patrones, dependiendo en las cargas de trabajo y la infraestructura disponible para el despliegue de una solución dada y el modelo adapta las cadenas de valor a estos parámetros de manera automática y transparente.

6.2.3 Analizando soluciones basadas en cadenas de valor

Para conducir esta etapa de evaluación se combinaron *GeoAcq*, *GeoProc* y *GeoEris* para crear dos cadenas de valor. En la primera *GeoAcq* adquiere los EOPs, *GeoProc* procesa estos EOPs para crear EOPs corregidos y productos derivados, los cuales son exhibidos por *GeoEris*. En la segunda

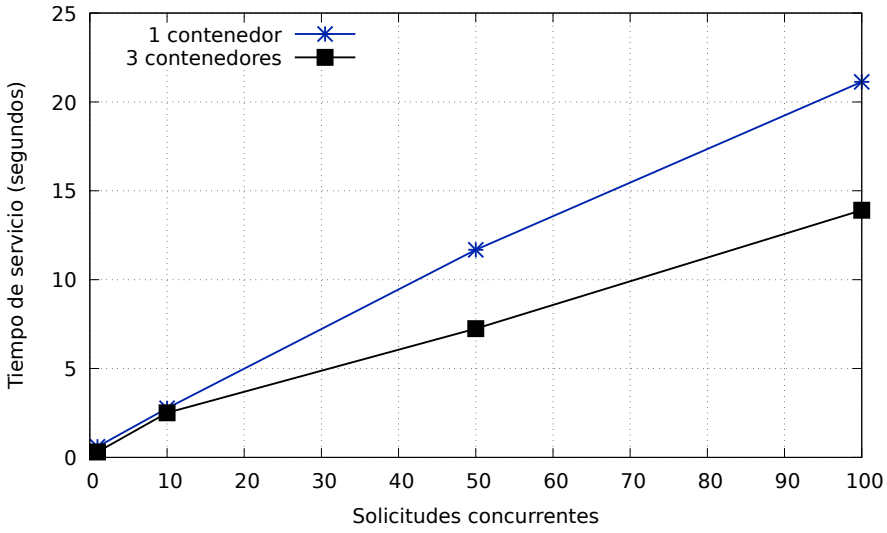


Figura 6.12: Tiempo de servicio del geoportal en *GeoEris* para atender diferente número de solicitudes concurrentes con uno y tres nodos.

cadena solamente se utiliza *GeoAcq* y *GeoEris* para adquirir y exhibir los EOPs respectivamente.

Ambas cadenas producen EOPs en cada etapa, los cuales son preservados e indexados dado que ambas cadenas utilizan *FedIDS* como servicio de almacenamiento y preservación.

En la Figura 6.13 se muestran las dos cadenas previamente descritas (*GeoAcq + GeoEris*) y (*GeoAcq + GeoProc + GeoEris*). La notación para construir las cadenas es:

$$\begin{aligned} C_1 =&(P_1, N_{GeoAcq}, e_1), (e_1, N_{FedIDS}, e_2), (e_2, N_{GeoProc}, e_4), \\ &(e_4, N_{FedIDS}, e_5), (e_5, N_{GeoEris}, Client). \end{aligned} \tag{6.2}$$

$$C_2 = (P_1, N_{GeoAcq}, e_1), (e_1, N_{FedIDS}, e_5), (e_5, N_{GeoEris}, Client). \tag{6.3}$$

Donde C_1 es $GeoAcq + GeoProc + GeoEris$ y C_2 es $GeoAcq + GeoEris$.

En la Figura 6.14 y Figura 6.15 muestran el tiempo de servicio (eje vertical) para C_1 y C_2 respectivamente, cuando el número de contenedores en paralelo es incrementado (eje horizontal). En este experimento, cada servicio (*GeoAcq + GeoProc + GeoEris*) es ejecutado con patrones que incluyen el mismo número de contenedores en paralelo (trabajadores). Por ejemplo, la columna dos

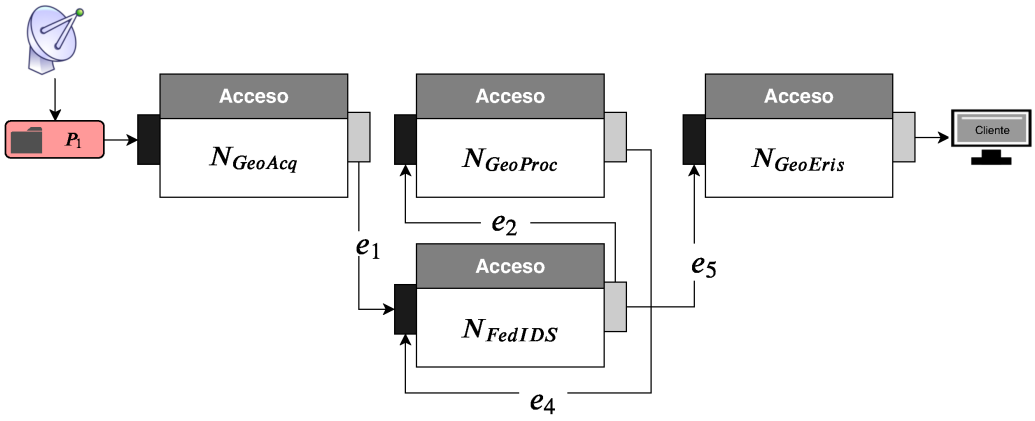


Figura 6.13: Cadenas de valor construidas mediante la combinación de servicios.

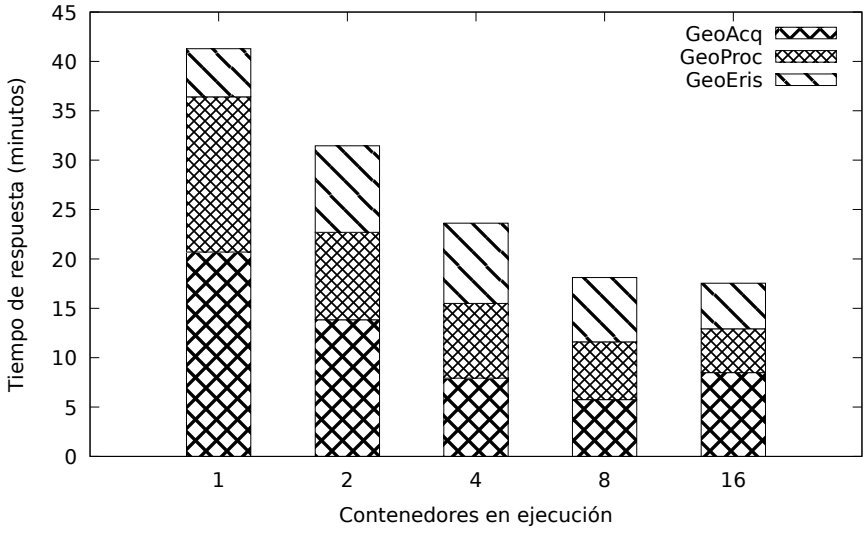


Figura 6.14: C_1 (*GeoAcq + GeoProc + GeoEris*).

en Figura 6.15 muestra que *GeoAcq* fue ejecutado utilizando un patrón de dos clones ejecutados en paralelo, lo cual también fue hecho para *GeoEris*.

Como se puede observar en ambas gráficas (Figura 6.14 y Figura 6.15), el tiempo de servicio es reducido para cada servicio (y para cada cadena de valor) cuando el número de contenedores ejecutado en paralelo es incrementado. Por ejemplo, para la cadena C_2, los patrones con 16 contenedores por servicio (columna 5 en la Figura 6.15) producen una mejora en el tiempo de servicio del 78.15 %, en comparación con el servicio que no utiliza patrones de paralelismo mientras que para la cadena C_1

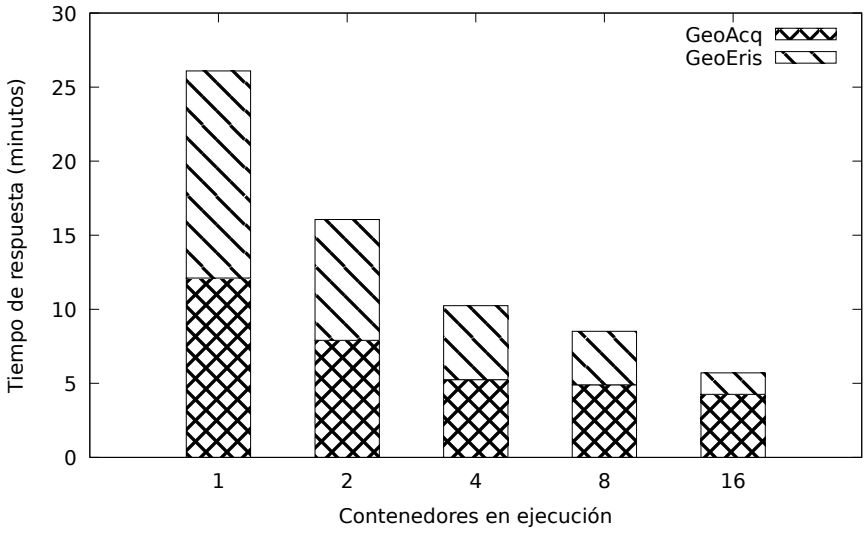


Figura 6.15: C_2 (*GeoAcq* + *GeoEris*).

la mejora fue del 57.5 %. Este efecto es mantenido mientras que los contenedores tengan suficientes recursos disponibles. Por ejemplo, en la Figura 6.14 hay tres servicios ejecutando 16 contenedores en paralelo cada uno, como resultado, existen 48 (16×3) procesos concurrentes, lo cual es más que los núcleos disponibles en el servidor donde la cadena de valor fue desplegada (24 núcleos; 12 físicos y 12 virtuales). Por otro lado, cuando se están procesando pequeñas cantidades de datos, el número de clones puede ser mayor al número de núcleos en el servidor, como en el caso de la cadena C_2, dado que se trabaja con los archivos de metadatos. Por otro lado, cuando se tienen que trabajar con grandes cantidades de datos es recomendable lanzar tantos contenedores como núcleos físicos tenga la máquina, como en el caso de C_1, donde $GeoProc$ trabaja con las imágenes en crudo (50 imágenes de 275MB).

6.3 Resumen

En el presente capítulo se presentó el desarrollo de *GeoAcq*, *GeoProc* y *GeoEris* utilizando el modelo propuesto para el manejo del ciclo de vida de las imágenes satelitales capturadas por la antena ERIS de la AEM. *GeoAcq* adquiere y extrae metadatos desde los archivos producidos por la

antena. *GeoProc* incluye microaplicaciones para el preprocesamiento de imágenes y su procesamiento para la creación de productos derivados de manera automática. *GeoEris* incluye un conjunto de microaplicaciones para el descubrimiento y exhibición de EOPs, así como para la generación de mosaicos y solapamientos bajo demanda, en un geoportal. Además, se incluye un módulo para la generación de recomendaciones a partir del historial de búsqueda los usuarios.

La evaluación de un estudio de caso basado en la implementación de este conjunto de servicios muestra el desempeño de cada servicio. Los patrones de paralelismo implementados mejoran significativamente el desempeño de las cadenas de valor creadas mediante la combinación de estos servicios, haciendo posible a las organizaciones crear soluciones para el manejo de EOPs, desde su adquisición hasta su exhibición, pasando por la generación de productos derivados.

7

Evaluación del prototipo para estudio de caso: servicio para el agrupamiento de registros climatológicos

En el presente capítulo se presenta el desarrollo de un servicio de agrupamiento y categorización de registros climatológicos, diseñado con el modelo propuesto en la presente tesis. Este servicio tiene microservicios y microaplicaciones que permiten a las organizaciones y usuarios finales manejar registros climatológicos desde su adquisición hasta su exhibición de resultados en un geoportal, incluyendo las fases de preprocesamiento y procesamiento. En este servicio, un *crawler* web se encarga de adquirir los registros, los cuales son enviados a un microservicio que realiza el agrupamiento distribuido de los datos adquiridos (temperatura y precipitación) mediante parámetros espacio-temporales. Los grupos encontrados son exhibidos en un geoportal donde el microservicio de estadísticas produce gráficas de regresión bajo demanda. Para evaluar este servicio se realizó un

Figura 7.1: Ubicación de las estaciones climatológicas de la CONAGUA [15].

estudio de caso basado en los registros capturados durante 33 años por la red de estaciones climatológicas de la CONAGUA (Comisión Nacional de Agua).

7.1 Repositorio de datos de la CONAGUA utilizado

Para realizar este estudio de caso se procesaron los conjuntos de datos disponibles para red de estaciones climatológicas de la CONAGUA en todo el territorio mexicano (ver Figura 7.1) [15]. Dicha red incluye 754 Estaciones Meteorológicas Automáticas (EMAS), que contienen registros de temperatura máxima y mínima, así como valores de precipitación diaria medida en milímetros. Para realizar el presente estudio de caso se seleccionaron las estaciones con datos desde 1985 hasta 2018.

7.2 Principios de diseño de la solución

El diseño del servicio fue desarrollado siguiendo las etapas mostradas en la Figura 7.2, donde los registros son adquiridos y preprocesados para obtener los datos útiles, que son utilizados por el módulo de procesamiento, en donde se le aplica a los datos un algoritmo de agrupamiento distribuido

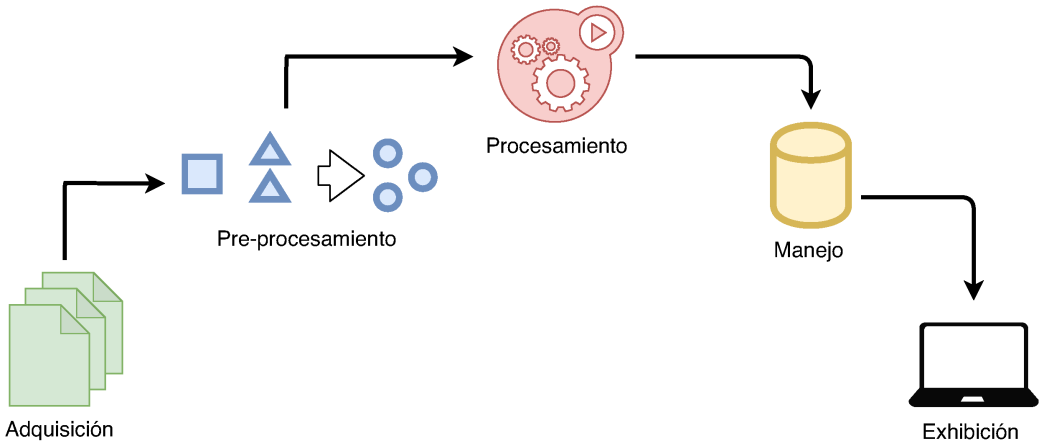


Figura 7.2: Flujo de trabajo creado para el manejo de los datos climatológicos.

(K-Means)[13]. Los resultados de dicho procesamiento son expuestos por una API REST que es consumida por un geoportal.

El diseño permite a los usuarios finales crear flujos de trabajo y analizar los resultados de agrupamiento obtenidos bajos demanda, de manera automática; como resultado, los usuarios pueden analizar regiones del país mediante parámetros espacio-temporales (coordenadas para un área de cobertura en un periodo de tiempo específico) elegidos a través del geoportal.

7.3 Detalles de implementación del servicio

El flujo de trabajo descrito en la Figura 7.2 fue dividido en dos etapas principales: la primera realizada fuera de línea (adquisición y preprocesamiento) y otro realizada bajo demanda (procesamiento), la cual es lanzada cuando el usuario envía una solicitud utilizando parámetros espacio-temporales, al geoportal.

7.3.1 Adquisición y procesamiento de registros climatológicos

En la primera etapa se construyó un patrón *M/W*, el cual considera una *Micro-App* llamada *Dispatcher*, que incluye un crawler web configurado para obtener todos los registros disponibles para

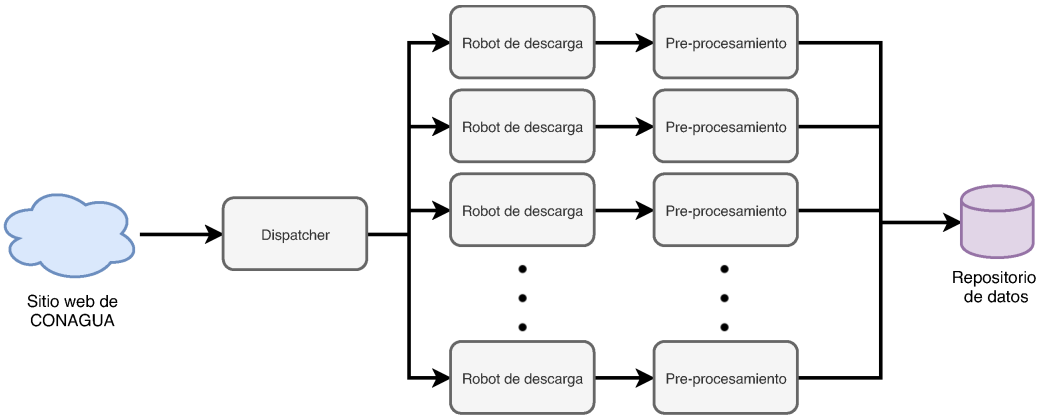


Figura 7.3: Patrón de adquisición y preprocesamiento.

cada estación. Esta *Micro-App* recibe las URLs de las estaciones como parámetros de entrada y distribuye las tareas de descargas (t) a n trabajadores. Cada trabajador incluye dos *Micro-Apps* encadenadas. La primera *Micro-App* realiza la descarga de los registros disponibles en una URL, mientras que la segunda se encarga del preprocesamiento de los datos descargados, incluyendo tareas como transformación de los datos, limpieza de los datos e integración de los datos. En la Figura 7.3 se describe el patrón que siguen los datos para ser preprocesados en esta etapa.

Para el estudio de caso, el crawler ubicado en el *Dispatcher* recolecta los metadatos asociados con las EMAS desde la página de la CONAGUA y extrae las URLs asociadas con cada estación climatológica. Cada URL está asociada con un archivo que contiene los registros de 33 años para una estación dada, en el cual se tienen datos como la temperatura y precipitación para cada día observado por la estación. Cuando el *Dispatcher* ha recolectado los metadatos de todas las estaciones lanza n trabajadores que se encargan de descargar los datos. El *Dispatcher* distribuye las URLs a cada trabajador de una manera balanceada, utilizando la técnica *Two-Choices* [47]. Cada trabajador recibe una lista del subconjunto de URLs que tiene que descargar desde la fuente de datos. Cada vez que un trabajador descarga un archivo, inmediatamente es enviado a la *Micro-App* de preprocesamiento, donde los datos son transformados en un formato único y coherente y los valores faltantes son rellenados utilizando una técnica de regresión lineal. Los archivos transformados son depositados en un repositorio de datos, que es utilizado por la etapa de procesamiento.

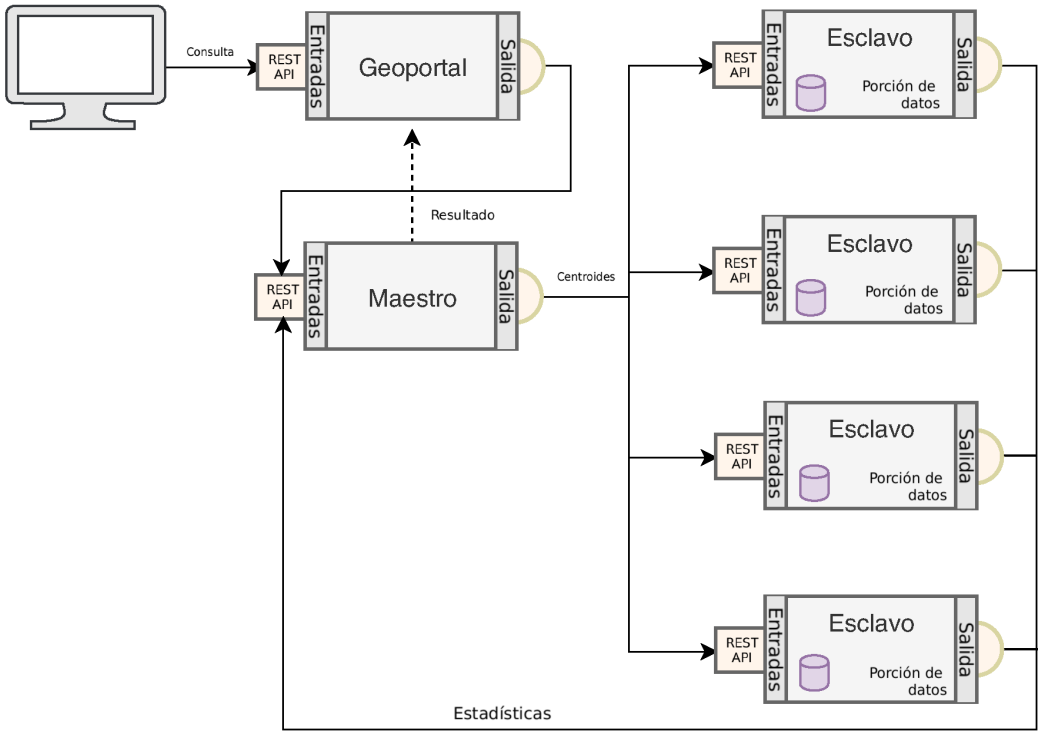


Figura 7.4: Servicio de agrupamiento distribuido.

7.3.2 Agrupamiento y categorización de registros climatológicos

El repositorio de datos producido por el patrón de preprocesamiento es utilizado como parámetro de entrada por las *Micro-Apps* de la etapa de procesamiento. En esta etapa un servicio de agrupamiento de datos distribuido fue desarrollado utilizando un patrón *M/W* (ver Figura 7.4). Este patrón es ejecutado bajo demanda mediante solicitudes que son enviadas por los usuarios mediante un geoportal.

El agrupamiento distribuido es una implementación del algoritmo K-Means, como es descrita por Cesario y Thalia [13]. En este patrón, la *Micro-App* conocida como *Maestro*, está a cargo de inicializar y recalcular los centroides, mientras que las *Micro-Apps* conocidas como *esclavos* están a carga de agrupar las estaciones climatológicas con otras estaciones similares utilizando los centroides proporcionados por el *maestro* y los registros de cada estación. Cada *esclavo* envía estadísticas a *maestro*, al finalizar cada iteración del algoritmo. Dentro de las estadísticas se incluyen: la distancia

intra-cluster, la distancia inter-cluster y el número de elementos de cada grupo generado. Estas estadísticas son utilizadas por el *maestro* para recalcular los centroides. Este proceso es finalizado, cuando el algoritmo alcanza un número máximo de iteraciones o cuando este converge[1].

Este servicio, además incluye un microservicio desarrollado en el lenguaje de programación R, que realiza el cálculo de la regresión de los grupos identificados para todos los registros de las todas las estaciones que se encuentren en el área de cobertura seleccionada por el usuario. Los resultados producidos por este servicio son consumidos por el geoportal, en donde se muestran gráficas de regresión (lineal y polinomial) para cada grupo generado.

7.4 Evaluación experimental del estudio de caso

En la presente sección se describe la metodología seguida para la evaluación basada el estudio de caso de la adquisición automática y agrupamiento de los registros de la red de estaciones climatológicas de la CONAGUA.

7.4.1 Resultados

En el primer experimento se midió el tiempo de servicio del esquema de preprocesamiento utilizando diferente *Micro-Apps* (contenedores) de preprocesameinto corriendo en paralelo. En la Figura 7.5 se muestra, en el eje horizontal, el número de contenedores corriendo en paralelo, mientras que en el eje vertical se muestra el tiempo de servicio de la solución cuando todos los archivos son descargados. La Figura 7.5 muestra que duplicar el número de contenedores produce una velocidad de 2x.

En la Figura 7.6 se muestra una comparación del tiempo de servicio del algoritmo de agrupamiento distribuido utilizando uno y 12 contenedores (eje vertical) para un diferente número de grupos de estaciones (eje horizontal). Como se puede observar, cuando se utiliza un solo contenedor, el tiempo

[1]El algoritmo converge cuando la suma del error cuadrático deja de cambiar entre iteraciones.

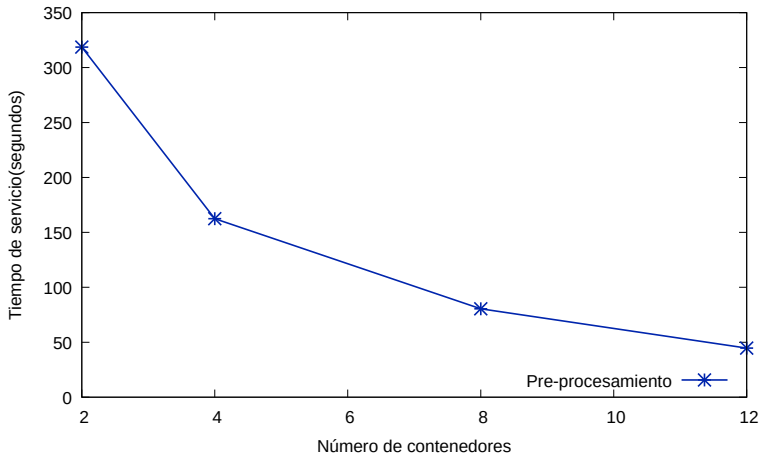


Figura 7.5: Tiempo de servicio del esquema de preprocesamiento utilizando diferente número de contenedores virtuales corriendo en paralelo.

de servicio tiende a incrementar significativamente cuando se incluyen un mayor número de solicitudes en el agrupamiento. Mientras la cantidad de contenedores corriendo en paralelo sea incrementada, el tiempo de servicio observado por los usuarios finales se reducirá, haciendo manejable el incremento en la carga (estaciones) en las solicitudes de los usuarios.

Figura 7.5 y Figura 7.6 muestran una mejora en el rendimiento cuando el número de contenedores en los patrones es incrementado, sin necesidad de incrementar el número de recursos computacionales.

Figura 7.7 muestra el porcentaje del tiempo de servicio de cada tarea que esta incluidas en el procesamiento: el agrupamiento distribuido y la generación de regresiones (lineales y polinomiales). Mientras que el número de esclavos corriendo en paralelo disminuye el porcentaje de tiempo del agrupamiento aumenta, mientras que el porcentaje de regresión se mantiene constante, dado que es realizado por un solo contenedor.

7.5 Resumen

En el presente capítulo se presentó un servicio para el agrupamiento y categorización de registros climatológicos utilizando el modelo propuesto. Este servicio considera un esquema de

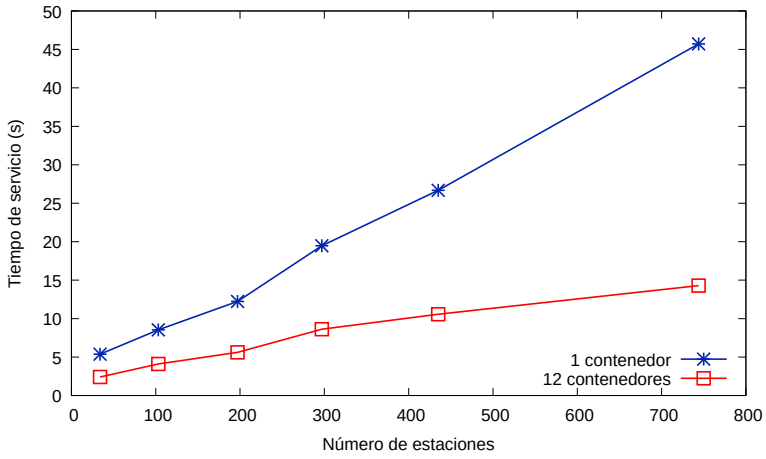


Figura 7.6: Porcentaje del tiempo de servicio para el agrupamiento y generación de gráficas utilizando diferente número de estaciones.

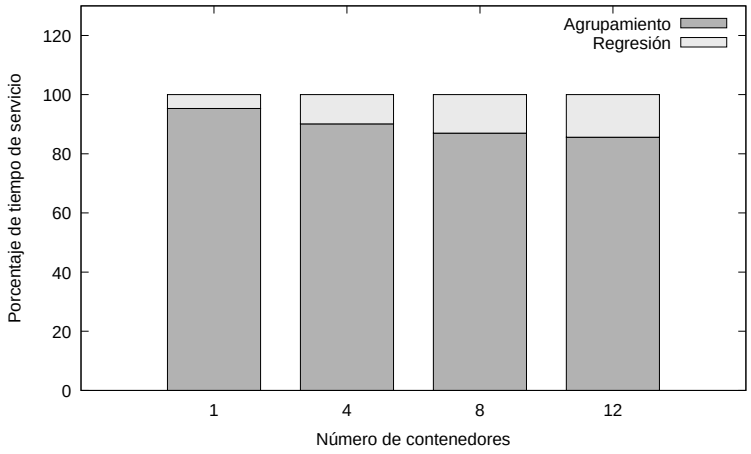


Figura 7.7: Porcentaje del tiempo de servicio para cada tarea realizada en el procesamiento, para diferente número de contenedores.

microserviciones y microaplicaciones para que las organizaciones puedan manejar las diferentes etapas de procesamiento de los datos climatológicos. El estudio de caso basado en registro de 33 años de la red de estaciones climatológicas de la CONAGUA fue realizado, configurando patrones para la descarga, preprocesamiento y procesamiento de los registros.

8

Conclusiones, limitaciones y trabajo futuro

En el presente capítulo se presentan las conclusiones del trabajo de tesis desarrollado, así como de la evaluación experimental. Además, se presentan las limitaciones de este trabajo, así como el trabajo futuro.

8.1 Conclusiones

El presente documento de Tesis se presentó el diseño, desarrollo e implementación de un modelo de entrega continua basado en DAGs para manejar el ciclo de vida de EOPs mediante flujos de trabajo agnósticos.

En este modelo los nodos son encapsulados en un conjunto de contenedores virtuales, los cuales son acoplados siguiendo la notación del DAG. Esta organización produce entrega continua de datos entre las etapas de los flujos de trabajo tradicionales. Como resultado, el modelo convierte un flujo de trabajo tradicional en una solución agnóstica capaz de ser ejecutada y operada en diversas plataformas (sistemas operativos) presentes en escenarios del mundo real. Este modelo representa el

objetivo general propuesto en el presente trabajo de Tesis.

El modelo despliega los flujos de trabajo agnósticos en un ecosistema controlado mediante arquitecturas de microservicios, lo cual permite la construcción automática de múltiples soluciones eficientes que incluso pueden reutilizar etapas existentes. La arquitectura de microservicios permite hacer frente a los problemas de adaptabilidad, heterogeneidad y portabilidad de componentes de un flujo de trabajo.

La experimentación conducida en la forma de estudios de caso donde se construyeron múltiples y diversas soluciones para el manejo de los ciclos de vida de datos satelitales y meteorológicos contemplando diferentes etapas de procesamiento, reveló la habilidad de manejar heterogeneidad de aplicaciones ya que múltiples soluciones pudieron ser desplegadas en diferentes tipos de infraestructura (mostrando la propiedad de portabilidad). De la misma forma la conducción de los estudios de caso reveló que se pueden reutilizar soluciones agregando nuevos etapas a los flujos desplegados (mostrando la propiedad de adaptabilidad). Esta manejo de soluciones basado en arquitecturas de microservicios representa la consecución del primer objetivo particular propuesto en el presente trabajo de Tesis.

El modelo también considera un esquema de procesamiento basado en patrones de paralelismo de tareas basados en contenedores virtuales. Este esquema permite que la implementación de flujos de trabajo agnósticos sea factible en escenarios reales donde comúnmente se presentan problemas de rendimiento y cuellos de botella.

La evaluación experimental reveló que los patrones de paralelismo basados en contenedores permiten mejorar el rendimiento de las soluciones sin que el usuario final deba modificar el código de las etapas de un flujo para paralelizar rutinas o aplicaciones. Lo anterior permite reducir el tiempo de espera observado por un usuario al utilizar soluciones creadas por el modelo (lo cual agrega la propiedad de eficiencia a los flujos de trabajo agnósticos). Este esquema de procesamiento representa la consecución del segundo y último objetivo particular propuesto en el presente trabajo de Tesis.

La evaluación también reveló que las soluciones desarrolladas con el modelo de construcción

propuesto en la presente Tesis produjeron mejoras en los tiempos de respuesta en comparación con dos motores de flujo de trabajo disponibles en la literatura: *DagOnStar* y *Parsl*.

En el caso de *Parsl*, el modelo propuesto obtuvo una mejora en los tiempos de respuesta (utilizando un patrón M/W con seis trabajadores) del $37\,\%$, mientras que obtuvo un $56\,\%$ de ganancia con respecto a *DagOnStar* (utilizando un patrón M/W con seis trabajadores). Esta mejora muestra que los patrones de paralelismo no solo logran mitigar el impacto sobre el rendimiento producido por los componentes que hacen parte del modelo (contenedores virtuales y estructuras de control de los microservicios), sino que, además, logran reducir los tiempos de respuesta de las aplicaciones, y, por ende, de las soluciones generadas, lo cual mejora la experiencia de servicio del usuario final.

Al hacer un análisis específico sobre los resultados de rendimiento se observó que la mejora de rendimiento producida por un patrón, basado en contenedores virtuales depende de la cantidad de datos procesados y del número de núcleos físicos del equipo donde se despliegue el patrón. Se identifican las siguientes reglas empíricas que se observaron constantemente en la evaluación experimental controlada y los estudios de caso:

1. Cuando se procesan archivos grandes, el patrón es tan eficiente como la cantidad de núcleos físicos que existan en la computadora donde el patrón sea desplegado.

2. Cuando el procesamiento de archivos grandes sobrepasa el uso de núcleos físicos se observa una disminución en el rendimiento del patrón debido a sobrecarga ocasionada por la gestión de los contenedores virtuales.

3. Los patrones M/W producen mejores resultados cuando se procesan archivos pequeños y cuando el conjunto de archivos es largo.

4. Cuando el procesamiento de archivos pequeños sobrepasa el uso de núcleos físicos se podrá incluso observar un incremento en el rendimiento del patrón.

5. Cuando el procesamiento de archivos pequeños sobrepasa el uso de núcleos físicos y virtuales, se podrá observar un decremento en el rendimiento del patrón.

8.2 Limitaciones

Las limitaciones del presente trabajo son las siguientes:

- En la versión actual del prototipo del modelo de construcción, la comunicación entre microservicios se lleva a cabo mediante un API REST (HTTP) y no se consideró en este trabajo el manejo de memoria compartida distribuida. Esto representa un área de oportunidad ya que, si dos microservicios se despliegan en un mismo equipo, la comunicación por medio de memoria mejoraría el rendimiento de las soluciones y actualmente esto no es posible en el estado actual del prototipo.
- No se implementaron patrones de paralelismo orientados a datos, por lo que el modelo actualmente solo considera patrones orientados a tareas, como el patrón M/W.
- Para garantizar la portabilidad de las soluciones, la infraestructura en donde se desea realizar el despliegue del flujo de trabajo debe de contar con la plataforma de contenedores virtuales Docker.

8.3 Trabajo futuro

Como trabajo futuro se espera integrar las siguientes características al prototipo desarrollado:

- Implementar esquemas de manejo de memoria distribuida compartida entre microservicios para la mejora de la eficiencia en las soluciones.
- Realizar la integración de múltiples patrones de paralelismo orientados a datos con el modelo de construcción propuesto.

- Explorar y aplicar de técnicas de manejo de operaciones (DevOps) en la construcción de flujos de trabajo para la mejora sistemática de las soluciones ya desplegadas.

A

Implementación de un prototipo funcional basado en el método propuesto

En el presente capítulo se describirán las técnicas utilizadas en la implementación del prototipo funcional basado en el modelo descrito en el Capítulo 4.

A.1 Requerimientos y componentes del prototipo funcional

Para demostrar la factibilidad del modelo propuesto se realizará una evaluación basada en prototipos. Por lo anterior, en la presente sección se describe la implementación de los diseños conceptuales descritos en la Sección 4. A continuación se describen los requerimientos y componentes del prototipo implementado con el modelo propuesto.

A.1.1 Requerimientos funcionales del prototipo

Para realizar la implementación del prototipo basado en el modelo propuesto se definieron los siguientes requerimientos funcionales:

- Contar con una aplicación web (cliente) que permite la carga e interpretación la notación de flujos de trabajo.
- Desarrollar un API que permita la construcción y encadenamiento de *Micro-Apps* para la construcción de nodos (microservicios).
- Dotar a los microservicios de un API REST que permita el intercambio de mensajes entre microservicios y aplicativos externos al flujo de trabajo.
- Desplegar un canal de intercambio de datos (EOPs) entre microservicios.

A.1.2 Componentes del prototipo

El prototipo está compuesto por tres componentes principales: i) el cliente que se encarga de recibir la notación de los flujos de trabajo utilizando la notación descrita en el capítulo anterior; ii) un motor de flujos de trabajo que recibe la notación y se encarga de generar las cadenas de valor contenidas en los microservicios (nodos); iii) y una arquitectura de microservicios que se encarga manejar los microservicios lanzados.

A.1.2.1. Cliente para la generación de flujos de trabajo

Para realizar la construcción de flujos de trabajos utilizando el modelo propuesto en el Capítulo 4 se diseñó un cliente web que permite realizar el diseño de los flujos mediante la escritura de una notación. El cliente fue desarrollado utilizando HTML5+JavaScript para el *frontend* y Python en el *backend*.

En el *frontend* se encuentra la interfaz gráfica de usuario de la aplicación y en ella el cliente puede cargar o escribir una notación siguiendo las reglas definidas en el modelo. Además, en el *frontend* el cliente puede seleccionar las microaplicaciones que conformarán sus nodos de un catálogo de servicios existentes o, en otro caso, generar nuevas microaplicaciones.

La notación generada en el *frontend* es enviada al *backend*, el cual analiza la analiza para verificar que haya sido escrita correctamente. Para realizar lo anterior, se desarrolló una aplicación en Python, utilizando la librería *Lark*[1], la cual permite transformar una notación en objetos de Python a partir de un archivo de configuración de reglas gramaticales que deben de seguir los componentes de la notación.

En la Figura A.1 se muestra el archivo de configuración definido para definir las reglas de escritura de los diferentes componentes del modelo propuesto. La regla básica de la notación es la de un *pair*, el cual es una tupla de la forma $clave : valor$, donde la clave es el nombre de la variable y el valor puede ser cualquiera de los siguientes:

- $microapp$: define el comando a ejecutar por una *Micro-App* mediante una cadena de caracteres escrita entre comillas dobles, por ejemplo, **microapp1:"python app1.py"**.

- $inputdata$: se utiliza para especificar la entrada de $I - EOPs$ en un nodo. Para indicar que se trata de una fuente de entrada de datos se debe de escribir el nombre de la variable seguido del símbolo de menor-que ($<$), de la siguiente forma: **nombre<datos**. La entrada de datos puede ser mediante el sistema de ficheros local o a recursos remotos. En el primer caso, solo se debe de escribir la ruta a los recursos como una cadena de caracteres escrita entre comillas dobles, por ejemplo, **nombre </ruta/a/los/datos"**. En el segundo caso, se debe de escribir la palabra **remote** y entre paréntesis indicar la dirección IP para acceder a los datos, así como el nombre de usuario para acceder a ellos y la ruta en donde se ubican los datos, por ejemplo, **nombre <remote("198.1.1.5","user","/ruta/a/los/datos")**.

[1] https://github.com/lark-parser/lark

- $outputdata$: define la salida de datos ($O-EOP$) de un nodo en un recurso de almacenamiento. Para indicar la salida de datos se debe de escribir el nombre de la variable seguido del símbolo de mayor-que ($>$), de la siguiente forma: **nombre>datos**. Al igual que con $inputdata$, se puede apuntar tanto a recursos locales como remotos, utilizando la misma notación descrita en el caso anterior.

- $parallel$: se utiliza para definir un patrón de paralelismo M/W. Para ello se debe de utilizar la palabra reservada $pattern$ y escribir entre paréntesis el nombre de la *Micro-App* a paralelizar, y el número de trabajadores, de la siguiente manera: **nombre: pattern(microapp1,3)**.

- $microapp_chain$: se utiliza para definir cadenas de microaplicaciones (nodos). Para ello, se debe de escribir los nombres de variables de las *Micro-App* en una lista encerrada en llaves ($\{\}$), por ejemplo, **chain:{microapp1, microapp2, microapp3}**. **microservices_chain**: define cadenas de valor mediante el encadenamiento de nodos ($microapp\_chain$) mediante un arreglo de tuplas. Por ejemplo, **graph:((input_data,c1,c2),(c2,output_data))**, define una cadena compuesta de dos nodos (microservicios), en donde el primer nodo toma los datos de entrada de un objeto $inputdata$, los procesa utilizando la cadena de microservicios $c1$ y los envía a la cadena $c2$ (segundo nodo), la cual escribe sus salidas en un objeto $outputdata$.

En la Figura A.2 se muestra un ejemplo de una notación escrita utilizando el modelo propuesto y las reglas definidas. La notación es enviada a un programa en Python, el cual se encarga de verificar que sea valida y de generar un archivo JSON, el cual es utilizado para mostrar de manera gráfica al usuario, el grafo generado a partir de dicha notación.

A.1.2.2. Despliegue de flujos de trabajo

A partir de la notación cargada por el usuario, el archivo JSON generado es enviado a un motor de flujos de despliegue de flujos de trabajo. El motor se encarga de la construcción de los contenedores de cada *Micro-App* definida en la notación. Para ello se pueden dar dos casos: el primero es cuando

```
?start: value
    ?value: node
          | parallel
          | item
          | inputdata
          | outputdata
          | microapp
          | microapp_chain
          | microservices_chain
          | SIGNED_NUMBER      -> number
          | "true"             -> true
          | "false"            -> false
          | "null"             -> null

    ?location: local
          | remote

    microapp:ESCAPED_STRING
    microapp_chain : "{" [name ("," name)*] "}"
    inputdata : name"<"location
    outputdata : name">"location

    local: ESCAPED_STRING
    remote: "remote("ESCAPED_STRING","ESCAPED_STRING","ESCAPED_STRING")"
    item : [pair]
    microservices_chain :  "("["("name","name","name")" ("," "("name","name")")*]")"
    parallel : "pattern(" [name","SIGNED_NUMBER] ")"
    workers : /[0-9]+/
    name : NAME
    pair   : name ":" value
    %import common.CNAME -> NAME
    %import common.SIGNED_NUMBER
    %import common.ESCAPED_STRING
    %import common.SIGNED_NUMBER
    %import common.WS
    %ignore WS
```

Figura A.1: Archivo de configuración desarrollado para definir las reglas de escritura de la notación basada en el modelo propuesto.

```
parsing:"command1"
transformation:"command2"
input_data < "/home/data"
output_data > "/home/data"
pt:pattern(parsing,3)
c1: {pt,transformation}
c2: {pt,transformation}
graph:((input_data,c1,c2),(c2,output_data))
```

Figura A.2: Ejemplo de la notación de entrada.

la imagen de contenedor ya exista en el repositorio de imágenes, entonces solamente se tiene que lanzar el contenedor a partir de dicha imagen; el segundo caso es cuando la imagen de contenedor no existe, por lo que tiene que ser construida utilizando como base una imagen existente.

Para desplegar los flujos de trabajo se desarrolló un módulo en Python que resuelve las dependencias entre tareas de un nodo a partir de la notación proveída por el usuario. El módulo crea una cadena de aplicaciones mediante las entradas y salidas de cada tarea. El proceso de construcción de los flujos de trabajo, a partir de la notación recibida, es el siguiente:

- Las aplicaciones son encapsuladas en contenedores virtuales junto con todas sus dependencias, para la generación de microaplicaciones (tareas).
- Se crea un volumen de entrada y salida de datos para cada tarea.
- Se genera un archivo de configuración para cada tarea, indicando la ruta del volumen de entrada y salida de datos.
- Se construyen las imágenes de contenedor de cada tarea y se despliegan los contenedores virtuales dentro de cada nodo.

Para el despliegue de los flujos de trabajo se implementó un API que incluye un conjunto de funciones que son utilizadas para la construcción y despliegue de flujos de trabajo utilizando el modelo propuesto. Las funciones que provee el API son:

- **agregarTarea(identificador, comando, parámetros)**: esta función permite agregar una nueva tarea al flujo de trabajo. Sus parámetros de entrada son un identificador de la tarea, el comando que ejecutará la tarea y los parámetros de entrada y salida de datos de la tarea.
- **crearDependencia(tarea1, tarea2)**: esta función permite crear una dependencia directa entre dos tareas, de este modo la ejecución de la segunda tarea siempre se hará una vez terminada la primer tarea. Sus parámetros son las referencias a las tareas.

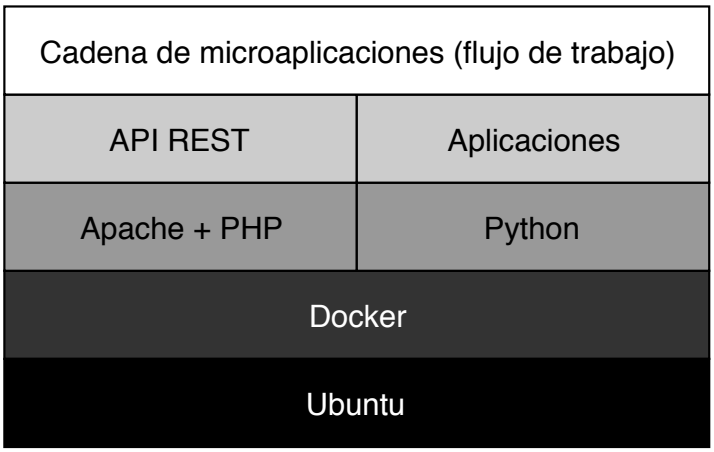


Figura A.3: Componentes del contenedor virtual en el que se despliegan los microservicios.

- **desplegarPatronMW(tarea, #trabajadores)**: esta función despliega un patrón M/W a partir de una tarea dada. Sus parámetros son la referencia a una tarea y el número de trabajadores (replicas de la tarea).

- **construirNodo(listaTareas)**: encapsula un conjunto de tareas pertenecientes a una etapa del ciclo de vida de los EOPs en un nodo (microservicio).

A.1.2.3. Arquitectura de microservicios

Una vez que se han construido las cadenas de *Micro-Apps* (nodos), los flujos trabajo son encapsulados utilizando microservicios, los cuales se encargan de lanzar y administrar los flujos en su interior, así como de exponer, mediante un API REST, todas las tareas de flujo, para que otros microservicios puedan consumirlos.

Cada microservicio es desplegado utilizando un contenedor que en su interior tiene instalado como sistema operativo base Ubuntu 16.04, Docker (para el manejo de las *Micro-Apps*), Apache+PHP (para el manejo de la API REST) y Python como punto de entrada del contenedor para el despliegue de los microservicios en el interior del microservicio (ver Figura A.3).

Los microservicios son desplegados utilizando Docker Compose[2]. Para ello, la notación JSON

[2]https://docs.docker.com/compose/

es traducida en un archivo YML[3], en donde se definen las *Micro-Apps* (nodos) de las etapas del ciclo de vida que el usuario haya seleccionado en la notación. En el archivo YML se definen las configuraciones de los microservicios, tales como volúmenes de datos, puertos, dependencias entre contenedores, variables de entorno y las imágenes de contenedor.

Una vez desplegados los microservicios estos pueden ser monitorizados desde el cliente que fueron desplegados, en el cual se informa el estado de cada uno los microservicios (activo, caído, en escucha, etc.). Para ello el cliente consume constantemente las APIs de los microservicios para conocer los cambios de estados.

Los microservicios pueden ser desplegados tanto en una máquina local como en ambientes distribuidos. En el segundo caso, el despliegue de los contenedores se realiza con el orquestador de contenedores Docker Swarm.

[3]Docker Compose recibe como entrada un archivo YML en el que se describen los servicios y dependencias entre contenedores.

Bibliografía

[1] Aditya, T. and Kraak, M.-J. (2007). Aim4gdi: Facilitating the synthesis of gdi resources through mapping and superimpositions of metadata summaries. *GeoInformatica*, 11(4):459–478.

[2] Armenise, V. (2015). Continuous delivery with jenkins: Jenkins solutions to implement continuous delivery. In *Proceedings of the Third International Workshop on Release Engineering*, pages 24–27. IEEE.

[3] Babuji, Y., Chard, K., Foster, I., Katz, D. S., Wilde, M., Woodard, A., and Wozniak, J. (2018). Parsl: Scalable parallel scripting in python. In *10th International Workshop on Science Gateways (IWSG 2018)*.

[4] Bernard, L., Kanellopoulos, I., Annoni, A., and Smits, P. (2005). The european geoportal—-one step towards the establishment of a european spatial data infrastructure. *Computers, environment and urban systems*, 29(1):15–31.

[5] Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

[6] Blumenfeld, J. (2018). Getting petabytes to people: How the eosdis facilitates earth observing data discovery and use | earthdata.

[7] Bravo Cabrera, J., Azpra Romero, E., Zarraluqui Such, V., Gay García, C., and Estrada Porrúa, F. (2012). Cluster analysis for validated climatology stations using precipitation in mexico. *Atmósfera*, 25(4):339–354.

[8] Cała, J., Marei, E., Xu, Y., Takeda, K., and Missier, P. (2016). Scalable and efficient whole-exome data processing using workflows on the cloud. *Future Generation Computer Systems*, 65:153–168.

[9] Calderón, L. J., Campoverde, J. Y., and Hoehne, A. V. (2014). El usuario como factor de éxito en el diseño de un geoportal. *GeoFocus. Revista Internacional de Ciencia y Tecnología de la Información Geográfica*, 0(14):181–210.

[10] Calmanti, S., Dell'Aquila, A., Maimone, F., and Pelino, V. (2015). Evaluation of climate patterns in a regional climate model over italy using long-term records from synop weather stations and cluster analysis. *Climate Research*, 62(3):173–188.

[11] Carnell, J. (2017). *Spring Microservices in Action*. Manning Publications Co.

[12] Castronova, A. M., Goodall, J. L., and Elag, M. M. (2013). Models as web services using the open geospatial consortium (ogc) web processing service (wps) standard. *Environmental Modelling & Software*, 41:72–83.

[13] Cesario, E. and Talia, D. (2012). Distributed data mining patterns and services: an architecture and experiments. *Concurrency and Computation: Practice and Experience*, 24(15):1751–1774.

[14] Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32(2):50–54.

[15] CONAGUA (2018). Estaciones meteorológicas automáticas (emas). [Online; accessed May 03, 2018].

[16] Cordeiro, L., Fischer, B., and Marques-Silva, J. (2010). Continuous verification of large embedded software using smt-based bounded model checking. In *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pages 160–169. IEEE.

[17] des Ligneris, B. (2005). Virtualization of linux based computers: the linux-vserver project. In *19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, pages 340–346. IEEE.

[18] Dictionary, C. (2019). Significado de agnostic - en inglés. https://dictionary.cambridge.org/es/diccionario/ingles/agnostic, Accedido el día 05 de Junio de 2019.

[19] docker

(2019). Docker terminology. https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/container-docker-introduction/docker-terminology.

[20] Docker Documentation (2019). Overview of docker compose. http://company.com/.

[21] ECOSUR (Accessed July 28, 2017). Eris: Estación recepción administrada por ecosur.

[22] Escuin, S., Navarro, R., and Fernández, P. (2008). Fire severity assessment by using nbr (normalized burn ratio) and ndvi (normalized difference vegetation index) derived from landsat tm/etm images. *Int. J. Remote Sens.*, 29(4):1053–1073.

[23] Fowler, M. and Foemmel, M. (2006). Continuous integration. *Thought-Works) http://www.thoughtworks. com/Continuous Integration. pdf*, 122:14.

[24] Friedl, M., McIver, D., Hodges, J., Zhang, X., Muchoney, D., Strahler, A., Woodcock, C., Gopal, S., Schneider, A., Cooper, A., Baccini, A., Gao, F., and Schaaf, C. (2002). Global land cover mapping from modis: algorithms and early results. *Remote Sensing of Environment*, 83(1):287 – 302. The Moderate Resolution Imaging Spectroradiometer (MODIS): a new generation of Land Surface Monitoring.

[25] Fry, R., Berry, R., Higgs, G., Orford, S., and Jones, S. (2012). The wiserd geoportal: A tool for the discovery, analysis and visualization of socio-economic (meta-) data for wales. *Transactions in GIS*, 16(2):105–124.

[26] Ghasemi, A. R. (2015). Changes and trends in maximum, minimum and mean temperature series in iran. *Atmospheric Science Letters*, 16(3):366–372.

[27] Gitelson, A. A., Stark, R., Grits, U., Rundquist, D., Kaufman, Y., and Derry, D. (2002). Vegetation and soil lines in visible spectral space: A concept and technique for remote estimation of vegetation fraction. *International Journal of Remote Sensing*, 23(13):2537–2562.

[28] Gonzalez, J., Perez, J. C., Sosa-Sosa, V. J., Sanchez, L. M., and Bergua, B. (2015). Skycds: A resilient content delivery service based on diversified cloud storage. *Simulation Modelling Practice and Theory*, 54:64 – 85.

[29] Gonzalez-Compean, J., Sosa-Sosa, V., Diaz-Perez, A., Carretero, J., and Yanez-Sierra, J. (2018a). Sacbe: A building block approach for constructing efficient and flexible end-to-end cloud storage. *Journal of Systems and Software*, 135:143–156.

[30] Gonzalez-Compean, J., Sosa-Sosa, V. J., Diaz-Perez, A., Carretero, J., and Marcelin-Jimenez, R. (2018b). Fedids: a federated cloud storage architecture and satellite image delivery service for building dependable geospatial platforms. *International Journal of Digital Earth*, 11(7):730–751.

[31] González, J. L., Sosa-Sosa, V., Díaz-Pérez, A., Carretero, J., and Marcelín Jiménez, R. (2017). Fedids: a federated cloud storage architecture and satellite image delivery service for building dependable geospatial platforms. *International Journal of Digital Earth*, pages 1–22.

[32] Google (2019). Istio. https://cloud.google.com/istio/?hl=es.

[33] Granell, C., Miralles, I., Rodríguez-Pupo, L. E., González-Pérez, A., Casteleyn, S., Busetto, L., Pepe, M., Boschetti, M., and Huerta, J. (2017). Conceptual architecture and service-oriented implementation of a regional geoportal for rice monitoring. *ISPRS International Journal of Geo-Information*, 6(7).

[34] Hall, D. K. and Riggs, G. A. (2011). *Normalized-Difference Snow Index (NDSI)*, pages 779–780. Springer Netherlands, Dordrecht.

[35] Hayes, B. (2008). Cloud computing. *Communications of the ACM*, 51(7):9–11.

[36] Istio (2019). What is istio? `https://istio.io/docs/concepts/what-is-istio/`. Accedido el 9 de Junio de 2019.

[37] Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G.-M., Hautier, G., et al. (2015). Fireworks: A dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17):5037–5059.

[38] Jenkins (2019). Jenkins. `https://jenkins.io/`.

[39] Johanson, A., Flögel, S., Dullo, C., and Hasselbring, W. (2016). Oceantea: exploring ocean-derived climate data using microservices. In *International Workshop on Climate Informatics*. NCAR.

[40] Liu, J., Feld, D., Xue, Y., Garcke, J., Soddemann, T., and Pan, P. (2016). An efficient geosciences workflow on multi-core processors and gpus: a case study for aerosol optical depth retrieval from modis satellite data. *International Journal of Digital Earth*, 9:748–765.

[41] Ltd, C. (2017). Lxc. https://linuxcontainers.org.

[42] Madduri, R., Chard, K., Chard, R., Lacinski, L., Rodriguez, A., Sulakhe, D., Kelly, D., Dave, U., and Foster, I. (2015). The globus galaxies platform: delivering science gateways as a service. *Concurrency and Computation: Practice and Experience*, 27(16):4344–4360.

[43] Malhotra, L., Agarwal, D., and Jaiswal, A. (2014). Virtualization in cloud computing. *J. Inform. Tech. Softw. Eng*, 4(2).

[44] McFEETERS, S. K. (1996). The use of the normalized difference water index (ndwi) in the delineation of open water features. *International Journal of Remote Sensing*, 17(7):1425–1432.

[45] Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing. *NIST*.

[46] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014.

[47] Mitzenmacher, M. (2001). The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104.

[48] Molina, M. and Bayarri, S. (2011). A multinational sdi-based system to facilitate disaster risk management in the andean community. *Computers & Geosciences*, 37(9):1501 – 1510.

[49] Montella, R., Brizius, A., Luccio, D. D., Porter, C., Elliot, J., Madduri, R., Kelly, D., Riccio, A., and Foster, I. (2018a). Using the face-it portal and workflow engine for operational food quality prediction and assessment: An application to mussel farms monitoring in the bay of napoli, italy. *Future Generation Computer Systems*.

[50] Montella, R., Di Luccio, D., and Kosta, S. (2018b). Dagon*: Executing directed acyclic graphs as parallel jobs on anything. In *Supercomputing 2018*.

[51] Montella, R., Kelly, D., Xiong, W., Brizius, A., Elliott, J., Madduri, R., Maheshwari, K., Porter, C., Vilter, P., Wilde, M., et al. (2015). Face-it: A science gateway for food security research. *Concurrency and Computation: Practice and Experience*, 27(16):4423–4436.

[52] Moutsatsos, I. K., Hossain, I., Agarinis, C., Harbinski, F., Abraham, Y., Dobler, L., Zhang, X., Wilson, C. J., Jenkins, J. L., Holway, N., et al. (2017). Jenkins-ci, an open-source continuous integration system, as a scientific data and image-processing platform. *SLAS DISCOVERY: Advancing Life Sciences R&D*, 22(3):238–249.

[53] Neely, S. and Stolt, S. (2013). Continuous delivery? easy! just change everything (well, maybe it is not that easy). In *2013 Agile Conference*, pages 121–128. IEEE.

[54] Newman, S. (2015). *Building Microservices*. O'Reilly Media, Inc., 1st edition.

[55] Ose, K., Corpetti, T., and Demagistri, L. (2016). 2 - multispectral satellite image processing. In Baghdadi, N. and Zribi, M., editors, *Optical Remote Sensing of Land Surface*, pages 57 – 124. Elsevier.

[56] Pettorelli, N., Vik, J. O., Mysterud, A., Gaillard, J.-M., Tucker, C. J., and Stenseth, N. C. (2005). Using the satellite-derived ndvi to assess ecological responses to environmental change. *Trends in Ecology & Evolution*, 20(9):503 – 510.

[57] Pickle, E. (2010). *GeoNode–A New Approach to Developing SDI*. na.

[58] Qasha, R., Cała, J., and Watson, P. (2016). A framework for scientific workflow reproducibility in the cloud. In *2016 IEEE 12th International Conference on e-Science (e-Science)*, pages 81–90. IEEE.

[59] Ranabahu, A., Anderson, P., and Sheth, A. (2011). The cloud agnostic e-science analysis platform. *IEEE Internet Computing*, 15(6):85–89.

[60] Resources, S. E. O. (2018). Suomi npp (national polar-orbiting partnership) mission. `https://eoportal.org/web/eoportal/satellite-missions/content/-/article/suomi-npp-part-1`, Accedido el día 29 de Julio de 2018.

[61] Reyes, H. (2017). *Método para la construcción de tuberías de procesamiento de datos para la nube*. Master thesis, Cinvestav Tamaulipas.

[62] Reyes-Anastacio, H. G., Gonzalez-Compean, J. L., Carretero, J., Garcia-Blas, J., and Sosa-Sosa, V. J. (2019). Kulla: A virtual container-based model for building distributed and parallel agnostic applications. "No publicado".

[63] Richards, M. (2015). *Software Architecture Patterns*. O'Reilly Media, Inc.

[64] Rouse, M. (2019). What is agnostic? - definition from whatis.com. `https://whatis.techtarget.com/definition/agnostic`, Accedido el día 05 de Junio de 2019.

[65] Santana-Perez, I., da Silva, R. F., Rynge, M., Deelman, E., Pérez-Hernández, M. S., and Corcho, O. (2017). Reproducibility of execution environments in computational science using semantics and clouds. *Future Generation Computer Systems*, 67:354–367.

[66] Sevilla, J., Julien, Y., Sória, G., Sobrino, J. A., and Plaza, A. J. (2015). New geo-portal for modis/seviri image products with geolocation-based retrieval functionality. *Journal of Applied Remote Sensing*, 9(1):096079.

[67] Shahin, M., Babar, M. A., and Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943.

[68] Sharma, P., Chaufournier, L., Shenoy, P., and Tay, Y. (2016). Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference*, page 1. ACM.

[69] Skluzacek, T. J., Chard, K., and Foster, I. (2016). Klimatic: A virtual data lake for harvesting and distribution of geospatial data. In *2016 1st Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS)*, pages 31–36.

[70] Spatiales, C. N. D. (2016). Ground segment. `https://calipso.cnes.fr/en/CALIPSO/GP_segment_sol.htm`, Accedido el día 29 de Julio de 2018.

[71] Steiniger, S., De la Fuente, H., Fuentes, C., Barton, J., and Muñoz, J. (2017). Building a geographic data repository for urban research with free software – learning from observatorio.cedeus.cl. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-4/W2:147–153.

[72] Stephens, R. (2015). *Beginning Software Engineering*. Wrox Press Ltd., Birmingham, UK, UK, 1st edition.

[73] SWsoft, I. (2016). Openvz. https://openvz.org.

[74] Tavares de Sousa, N., Hasselbring, W., Weber, T., and Kranzlmüller, D. (2018). Designing a generic research data infrastructure architecture with continuous software engineering. In *CSE 2018: 3rd Workshop on Continuous Software Engineering*. CEUR-WS. org.

[75] Teke, M., Tevrizoğlu, İ., Öztoprak, A. F., Demirkesen, C., Açikgöz, İ. S., Gürbüz, S. Z., Küpcü, R., and Avenoğlu, B. (2015). Geoportal: Tübitak uzay satellite data processing and sharing system. In *Recent Advances in Space Technologies (RAST), 2015 7th International Conference on*, pages 233–238. IEEE.

[76] Terstyanszky, G., Kukla, T., Kiss, T., Kacsuk, P., Balasko, A., and Farkas, Z. (2014). Enabling scientific workflow sharing through coarse-grained interoperability. *Future Generation Computer Systems*, 37:46 – 59. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications.

[77] van Der Aalst, W. M., Ter Hofstede, A. H., Kiepuszewski, B., and Barros, A. P. (2003). Workflow patterns. *Distributed and parallel databases*, 14(1):5–51.

[78] van der Aalst, W. M. P. (1998). The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8:21–66.

[79] Vassiliadis, P. (2009). A survey of extract–transform–load technology. *International Journal of Data Warehousing and Mining (IJDWM)*, 5(3):1–27.

[80] Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., et al. (2013). The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic acids research*, 41(W1):W557–W561.

[81] Xing, Y. and Zhan, Y. (2012). Virtualization and cloud computing. In *Future Wireless Networks and Information Systems*, pages 305–312. Springer.

[82] Zhang, Y., Wang, B., Zhang, Z., Duan, Y., Zhang, Y., Sun, M., and Ji, S. (2014). Fully automatic generation of geoinformation products with chinese zy-3 satellite imagery. *The Photogrammetric Record*, 29.

[83] Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T., and Wilde, M. (2007). Swift: Fast, reliable, loosely coupled parallel computation. In *2007 IEEE Congress on Services (Services 2007)*, pages 199–206.