UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

# Un *framework* de adaptación de micro-servicios y contenedores para procesos DevOps

**Máster en Ingeniería y Tecnología de Sistemas Software**

**Departamento de Sistemas Informáticos y Computación**

**Autor**: Luisa Fernanda Gómez Muñoz

**Tutores**: Dra. Silvia Mara Abrahão Gonzales

Dr. César Emilio Insfrán Pelozo

**Curso**: 2019-2020

| | |
|---|---|
| Lugar: | **Valencia, España** |
| Fecha: | **Septiembre 2020** |
| Autor: | **Luisa Fernanda Gómez Muñoz** |
| Tutor: | **Dra. Silvia Mara Abrahão Gonzales** |
| Título: | **Un *framework* de adaptación de micro-servicios y contenedores para procesos DevOps** |
| Universidad: | **Universitat Politècnica de València** |
| Departamento: | **Departamento de Sistemas Informáticos y Computación** |
| Grado: | **Máster en Ingeniería y Tecnología de Sistemas Software** |
| Curso: | **2019-2020** |

# RESUMEN

Los procesos y la forma de desarrollar, desplegar y mantener software y su infraestructura están en constante cambio, lo cual conduce inevitablemente a desafíos técnicos y funcionales. Las empresas requieren soluciones y procesos configurables y flexibles que les permitan desplegar servicios en la nube y adaptarlos durante el ciclo de vida del servicio para asegurar las necesidades y expectativas de sus clientes. Frente a estas necesidades, actividades como la integración y el despliegue continuos y la capacidad del sistema para su propia evolución y reconfiguración en entornos variables, son claves en la mejora y automatización de los procesos de desarrollo e implantación de software.

Con el fin de realizar una contribución en este campo se plantea el presente proyecto, cuyo objetivo principal es proponer un *Framework* para la adaptación de sistemas software basados en micro-servicios y contenedores soportado en un proceso DevOps, formulado a partir de una identificación y clasificación taxonómica de estudios primarios orientados a la investigación e implantación de mecanismos de adaptación mediante una revisión sistemática de la literatura.

El resultado final consiste en un ***Framework* de clasificación taxonómica** que proporciona un esquema para la comprensión de los principales criterios de adaptación identificados en la revisión sistemática y un ***Framework* de adaptación** construido a partir del anterior, el cual propone una guía para seleccionar las tareas, técnicas y herramientas a incorporar en el diseño de un sistema, para dotarlo de capacidades adaptativas específicas. La adaptabilidad es incorporada mediante el uso de un flujo de retroalimentación MAPE-K, por lo que las tareas se identifican en función a la fase que apoyan al igual que las demás técnicas y herramientas a seleccionar en conformidad con el contexto de aplicación principal de las estrategias adaptativas, que puede ser de carácter interno (en caso de que los mecanismos de adaptación se ejecuten directamente sobre el sistema) o externo (si se ejecutan sobre la capa de infraestructura). Adicionalmente, para asegurar la evolución continua del sistema adaptativo resultante, se incluye una propuesta de las actividades DevOps a implementar. Por último, se propone un procedimiento de seis pasos para la adopción del *Framework* de adaptación. Para ejemplificar la implementación del *Framework*, se presenta como escenario de prueba un sistema de monitorización de micro-servicios desplegado en contenedores en la nube sobre el que se incorporan capacidades de adaptación.

# ABSTRACT

*The processes and way to develop, deploy, and maintain software systems are always changing, and inevitably it leads the engineers to technical and functional challenges. To ensure the needs and expectations of their customers, companies require flexible and configurable solutions and processes to support the development and deployment of cloud services and their adaptation throughout the service life cycle. In this scenario, activities for continuous integration and continuous deployment are paramount to provide software systems with adaptation capacities, allowing its evolution on changing environments by improving the development and implementation processes. In this project, we aim to propose a Framework for the adaptation of software systems based on microservices and deployed in containers, supported in a DevOps process strategy. We start with a taxonomic classification of primary studies endeavoured to the research and implementation of adaptation mechanisms employing a systematic literature review. Thus, this project presents a taxonomic classification Framework that provides a body of knowledge about the main adaptation criteria and mechanisms identified in the systematic review. We also proposes an adaptation Framework, based on the previous classification, with a proposal of a procedure built to choose the tasks, techniques and tools needed to incorporate adaptation mechanisms in the design of a software system to include adaptation qualities. The Framework integrates the adaptability capabilities through the use of a MAPE-K feedback loop. A set of tasks are identified accordingly with the supported MAPE-K phase, in conjunction with other tools and techniques bound to the context of the adaptation strategy. This context can be internal (when the adaptation mechanisms belong to the system layer) or external (when the adaptation mechanisms belong on the infrastructure layer). The Framework also suggest a set of DevOps activities to ensure the continuous evolution of the system. Finally, this proposal also provides a six-step procedure to put the adaptation Framework into practice. We present a test scenario for the Framework's implantation by adding adaptation capabilities to a microservices monitoring system deployed in cloud containers.*

# RESUM

Els processos i la manera de desenvolupar, desplegar i mantindre programari i la seua infraestructura estan en constant canvi, la qual cosa condueix inevitablement a desafiaments tècnics i funcionals. Les empreses requereixen solucions i processos configurables i flexibles que els permeten desplegar serveis en el núvol i adaptar-los durant el cicle de vida del servei per a assegurar les necessitats i expectatives dels seus clients. Enfront d'aquestes necessitats, activitats com la integració i el desplegament continus i la capacitat del sistema per a la seua pròpia evolució i reconfiguració en entorns variables, són claus en la millora i automatització dels processos de desenvolupament i implantació de programari.

Amb la finalitat de realitzar una contribució en aquest camp es planteja el present projecte, l'objectiu principal del qual és proposar un *Framework per a l'adaptació de sistemes programari basat en micro-serveis i contenidors suportat en un procés *DevOps, formulat a partir d'una identificació i classificació taxonòmica d'estudis primaris orientats a la investigació i implantació de mecanismes d'adaptació mitjançant una revisió sistemàtica de la literatura.

El resultat final consisteix en un ***Framework* de classificació taxonòmica** que proporciona un esquema per a la comprensió dels principals criteris d'adaptació identificats en la revisió sistemàtica i un ***Framework* d'adaptació** construït a partir de l'anterior, el qual proposa una guia per a seleccionar les tasques, tècniques i eines a incorporar en el disseny d'un sistema,per a dotar-lo de capacitats adaptatives específiques. L'adaptabilitat és incorporada mitjançant l'ús d'un flux de retroalimentació *MAPE-K, per la qual cosa les tasques s'identifiquen en funció a la fase que secunden igual que les altres tècniques i eines a seleccionar de conformitat amb el context d'aplicació principal de les estratègies adaptatives, que pot ser de caràcter intern (en cas que els mecanismes d'adaptació s'executen directament sobre el sistema) o extern (si s'executen sobre la capa d'infraestructura). Addicionalment, per a assegurar l'evolució contínua del sistema adaptatiu resultant, s'inclou una proposta de les activitats *DevOps a implementar. Finalment, s'entrega un procediment de sis passos per a la lectura i implementació per al **Framework* d'adaptació. Per a exemplificar la implementació del emph*Framework, es presenta com a escenari de prova un sistema de monitoratge de micro-serveis desplegat en contenidors en el núvol sobre el qual s'incorporen capacitats d'adaptació.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

Los procesos de desarrollo de software han evolucionado a medida que la capacidad de los componentes de procesamiento y demás elementos electrónicos revolucionan el ámbito de la computación. El enfoque de arquitectura monolítica como estrategia para el despliegue de sistemas eficientes, controlables y precisos ha sido remplazado por arquitecturas descentralizadas que, pesar de consumir comparativamente más recursos, facilita la escalabilidad, resiliencia y adaptabilidad de los sistemas software [39].

La combinación de estos factores da lugar a lo que ahora se encapsula bajo el término "micro-servicios", usado comúnmente para referirse tanto a la arquitectura como al paradigma de programación, que en términos generales consiste en la creación de aplicaciones modulares e independientes, con tiempos de desarrollo, depuración y despliegue notablemente más cortos (respecto a un sistema monolítico). Las arquitecturas orientadas a servicios usan estos conceptos en el diseño de sistemas complejos, mediante el uso de protocolos ligeros de comunicación e interacción [41]. Los avances en el campo de las telecomunicaciones y la capacidad de transferencia de información a través de Internet han dado lugar a la creación de esquemas novedosos para el despliegue de estas arquitecturas de forma completamente descentralizada mediante computación en la nube (*Cloud Computing* - Cloud). A pesar de que los nuevos escenarios tecnológicos han desencadenado una serie de retos y limitaciones en la generación de sistemas de software, de igual forma han potenciado sus procesos y metodologías de desarrollo. Los esquemas metodológicos para desarrollo ágil han evolucionado al punto de formular estrategias y tecnologías que buscan automatizar las tareas de despliegue de soluciones e incorporarlos como parte misma del desarrollo. Esta estrategia que fusiona los procesos de desarrollo y operaciones (*Development and Operations* - DevOps) ha tomado fuerza en los últimos años y ha impulsado las tecnologías de despliegue en contenedores [24].

El potencial que brindan estas tecnologías es ahora de especial interés académico en la creación de sistemas de software adaptativos. El esfuerzo en este sentido, sin embargo, se ha

centrado en solucionar problemas específicos de asignación dinámica de recursos sobre plataformas de computación en el *cloud* y en contraposición no existe una línea clara sobre las tecnologías y estrategias existentes para incorporar cualidades adaptativas a sistemas de software orientadas a microservicios, desplegadas en contenedores. Este proyecto busca presentar una propuesta en dicho sentido, consolidada en el planteamiento de un *Framework* de adaptación que encapsula las herramientas y tecnologías presentes en la literatura.

A continuación, se presenta la motivación del proyecto (sección 1.1), el planteamiento del problema abordado (sección 1.2), los objetivos propuestos (sección 1.3), el contexto en el que se aborda la investigación (sección 1.4) y la metodología aplicada (sección 1.5). Por último, en la sección 1.6 se presenta la organización del documento.

## 1.1. Motivación

Las aplicaciones informáticas han evolucionado considerablemente durante las últimas décadas. Esta evolución de los sistemas de software conduce inevitablemente a desafíos técnicos y funcionales. La capacidad del sistema de adaptarse y reconfigurarse se convierte en una cualidad deseable para afrontar dichos desafíos.

En términos generales, la adaptabilidad de un sistema es considerada una habilidad fundamental para garantizar su supervivencia. Este mecanismo permite afrontar los cambios continuos que se presentan en el entorno generando procesos internos de cambio constante. En los sistemas de software la adaptación es utilizada para la modificación o ampliación de una implementación sobre determinados comportamientos con el fin de mejorar las interacciones estáticas y dinámicas con el entorno [34]. La computación orientada a servicios dio origen al paradigma de micro-servicios como uno de los patrones arquitecturales mayormente usados en los últimos años para asegurar la escalabilidad, flexibilidad y evolución de las aplicaciones de software. Este paradigma, posibilita el desacoplamiento en sistemas de alta complejidad, mediante la definición de componentes independientes que implementan funcionalidades específicas, facilitando las actividades de desarrollo y pruebas [67], con la capacidad de evolucionar en servicios individuales y de comunicarse entre ellos mediante protocolos ligeros [41].

La infraestructura necesaria para soportar los continuos cambios que implican la adaptación de un sistema es un aspecto crucial. Pahl et al. en [49] muestran los resultados de un estudio realizado para determinar el estado del arte de tecnologías orientadas a la contenerización en entornos *cloud*. Allí se evidencia el auge en la adopción de contenedores y técnicas de orquestación en el despliegue y gestión de componentes de software y sus relaciones. La versatilidad y desacoplamiento de los componentes en una arquitectura de micro-servicios conduce naturalmente a la contenerización debido a la flexibilidad que proporcionan en la fase de des-

pliegue [20]. Las herramientas como *Docker*, construidas alrededor de motores de contenedores posibilitan el empaquetamiento de aplicaciones construidas bajo una arquitectura orientada a micro-servicios. La construcción de sistemas en continuo crecimiento en entornos distribuidos requiere de un ambiente de desarrollo y operación que facilite su evolución, por lo tanto, DevOps se presenta como un marco oportuno en esta materia, proveniente de la ingeniería de software continuo, ya que presenta un grupo de actividades para mantener activo el flujo de desarrollo y entrega de software bajo los principios de automatización de procesos relacionados con la construcción de software, mediciones de desempeño de las aplicaciones para asegurar su calidad y una cultura enfocada en el cliente y el intercambio de conocimiento entre diversos grupos que participan en el desarrollo y entrega de las aplicaciones [24].

Se propone este trabajo con la hipótesis de que la investigación de las herramientas y los dominios de aplicación de sistemas adaptativos que incorporan en su arquitectura el paradigma de micro-servicios o el uso de contenedores, nos permita identificar el nivel de madurez en la actualidad de estas tecnologías, así como caracterizar las tendencias y limitaciones tecnológicas en la adopción de propiedades adaptativas a este tipo de sistemas. Esto con el ánimo de proponer una guía para el diseño de sistemas con propósito adaptativo cuya pila tecnológica incorpore herramientas y técnicas propias de contenedores y micro-servicios en el marco de un proceso DevOps.

## 1.2. Planteamiento del problema

El auge de la computación en el *cloud* y la modularización de los componentes mediante el uso de microservicios y contenedores ha impulsado la adopción de mecanismos adaptativos en sistemas de software. Sin embargo, los propósitos de adaptación han sido orientados principalmente a la mejora de atributos de calidad de los sistemas, como la escalabilidad, elasticidad y eficiencia [39].

En particular, a pesar de que la computación en la nube ofrece un alto grado de dinamismo en el aprovisionamiento de recursos, se observa una falta generalizada de soporte a la gestión de adaptación dinámica de servicios *Cloud* centrado en el funcionamiento o comportamiento del sistema, ya que el soporte que existe actualmente se centra en la elasticidad mediante la escalabilidad horizontal. Por otra parte, en los últimos años, se han realizado algunos trabajos que utilizan una estrategia de integración y/o despliegue continuo de servicios *Cloud* basados en micro-servicios para reducir el esfuerzo de integración y el impacto energético en los recursos.

Sin embargo, se observa que, a la fecha de elaboración de este trabajo, la información relacionada a la adaptación del comportamiento del software y en particular la relacionada a

entornos *Cloud* que incorporen en su arquitectura microservicios y despliegue a través de contenedores está desperdigada, se echa en falta un estudio que obtenga y analice toda la evidencia existente sobre los estudios que se han realizado. Por este motivo, una revisión sistemática podría ser usada como base para la realización de una clasificación taxonómica a partir de la cual se podría proponer un *Framework* enfocado en la incorporación de estrategias adaptativas sobre procesos DevOps, para el desarrollo de sistemas de software orientados a microservicios y despliegue e implementación mediante contenedores.

## 1.3. Objetivos

El objetivo general de este proyecto consiste en definir un *Framework* enfocado en la incorporación de estrategias adaptativas en el marco de un proceso DevOps, para sistemas de software basados en micro-servicios y desplegados en contenedores.

Este objetivo general implica los siguientes objetivos específicos:

1. Identificar las propuestas y aproximaciones presentes en la literatura sobre la incorporación de mecanismos de adaptabilidad en procesos DevOps, para sistemas de software orientados a micro-servicios y desplegados en contenedores.

2. Clasificar taxonómicamente los estudios primarios presentes en la literatura, orientados a la investigación de mecanismos de adaptación en sistemas de software basados en microservicios y desplegados en contenedores a través de una revisión sistemática de la literatura.

3. Definir un *Framework* enfocado en la implementación de mecanismos de adaptación para sistemas de software orientados a microservicios y desplegados en contenedores.

4. Proponer e ilustrar mediante un ejemplo una estrategia para incorporar el *Framework* propuesto en el marco de un proceso DevOps.

## 1.4. Contexto de la investigación

Este proyecto de fin de máster se ha desarrollado en el contexto del grupo de investigación de Ingeniería del software y Sistemas de Información (ISSI) del Instituto Universitario Mixto Tecnológico de Informática (ITI) de la Universitat Politècnica de València (UPV).

En particular, este trabajo ha contribuido a los resultados del Proyecto Adapt@Cloud: "Adaptación dinámica de servicios en la *cloud* centrado en el usuario" de la convocatoria de ayudas a proyectos de I+D+i, orientada a los retos de la sociedad del año 2018 financiado por el Ministerio de Economía y Competitividad (España), Participantes: Universitat Politècnica

de València (España), Universidade Nova de Lisboa (Portugal), Université catholique de Louvain (Bélgica), software Engineering Institute, Carnegie-Mellon University (USA). IP: Silvia Abrahão y Emilio Insfrán. De enero de 2018 a diciembre de 2020.

## 1.5. Método de investigación

Las actividades que guían el estudio presentado en este trabajo están definidas siguiendo una extensión del modelo para la transferencia de tecnología basado en las necesidades de la industria, propuesto por Gorscheck et al. en [27] y detallado en 1.5.1. Para conseguir el propósito de las actividades enfocadas al entendimiento del contexto se realiza una revisión sistemática de la literatura, método para la recopilación y análisis de estudios primarios expuesto en 1.5.2.

### 1.5.1. Modelo para la transferencia de tecnología

Consiste en un método iterativo de ocho fases que busca soluciones realistas a un problema partiendo de una solución candidata. En el presente trabajo se emplean las primeras cuatro actividades del modelo presentado en la Figura 3.1 descritas a continuación:

1. Identificación del problema: En esta etapa, se identifican y priorizan áreas potenciales de mejora.

2. Formulación del problema: De acuerdo con las necesidades priorizadas identificadas en el paso anterior, se fórmula una ruta de investigación. El problema es expresado claramente, así como el contexto y responsabilidades del trabajo de investigación. El resultado de estas dos primeras fases se presenta en las secciones de motivación (1.1) y planteamiento del problema (1.2).

3. Revisión del estado del arte: En esta fase se lleva a cabo una revisión crítica de las aproximaciones abordadas para intentar dar solución al problema planteado. Se utiliza una revisión sistemática de la literatura como método para el desarrollo de esta actividad, el cual consiste en la revisión de estudios primarios para identificar evidencia relevante disponible respecto a un área temática concreta [37]. El estudio sistemático de la literatura se ejecuta usando las fases de planeación, ejecución y documentación de resultados. Estas etapas se detallan en 1.5.2.

4. Solución candidata: En esta fase se trazan una o más soluciones potenciales al problema planteado. El producto derivado de esta etapa se presenta en el Capítulo 4.

Se plantea como trabajo futuro la ejecución de las demás fases del método de transferencia: entrenamiento (5), orientada a la generación e incremento del conocimiento, validación

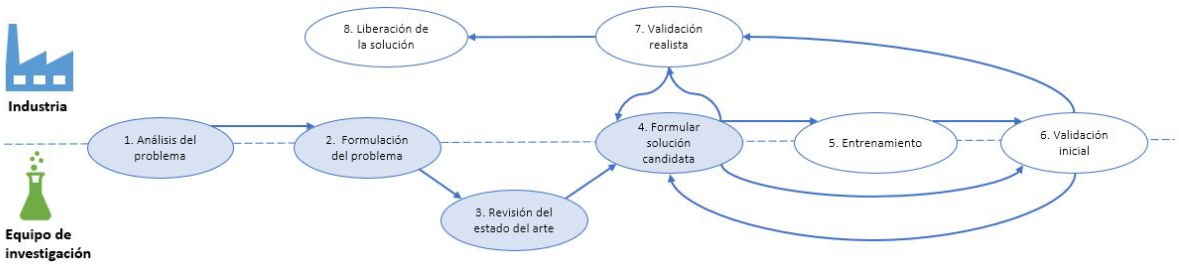


Figura 1.1: Método de investigación. Elaborado según Gorschek et al. [27]

inicial (6), consistente en la ejecución de una prueba controlada para evaluar la solución candidata, validación realista (7), en la cual se incorporan casos de estudio en entornos reales y lanzamiento de la solución (8), que corresponde con la valoración de los resultados y preparación de herramientas para ser usados en un contexto industrial.

### 1.5.2. Revisión sistemática de la literatura

Una Revisión Sistemática de la Literatura se fundamenta en una metodología rigurosa para identificar, analizar e interpretar de manera repetible y sin sesgo, todas las evidencias relativas a un tema de investigación [37]. Dentro de las principales características de una revisión sistemática se encuentran la definición de un protocolo de revisión que especifica la pregunta de investigación que se aborda y los métodos que se utilizarán para realizar la revisión, la integración de una estrategia de búsqueda estricta que tiene como objetivo detectar tanta literatura relevante como sea posible, la definición de criterios explícitos de inclusión y exclusión para evaluar cada posible estudio, así como la información que debe obtenerse de cada uno e incluye criterios de calidad mediante los cuales se evalúa la pertinencia de cada estudio primario.

De acuerdo con lo planteado por Kitchenham et al. en [37], una revisión sistemática de la literatura involucra varias actividades embebidas en tres fases principales: planeación, en esta fase se identifica la necesidad de la revisión, las preguntas de investigación son establecidas, el protocolo de búsqueda es definido y validado; ejecución de la revisión sistemática con base en el protocolo planteado para la extracción de los datos y por último la interpretación y difusión de los resultados obtenidos. Para la ejecución de la revisión sistemática en este estudio, se realiza una adaptación del procedimiento descrito anteriormente en conformidad con la extensión realizada por Rodriguez et al. presentada en [53]. Las etapas y actividades seguidas se describen a continuación:

1. **Planificación de la revisión**: Se plantea el objetivo, preguntas de investigación y el protocolo de revisión a utilizar a través de la ejecución de las siguientes actividades:
   - *Establecer el objetivo de investigación*: Definición del objetivo del proceso a partir de la

evaluación de la necesidad de ejecutar una revisión sistemática.

- *Establecer las preguntas de investigación*: En conjunto con el objetivo, las preguntas componen el alcance de la revisión sistemática y conducen el proceso de búsqueda, análisis y clasificación de los estudios primarios, al igual que la extracción de datos.
- *Definir el protocolo de búsqueda*: En un protocolo de revisión o búsqueda se describen los métodos que se utilizarán para realizar la revisión sistemática. Los componentes de un protocolo incluyen los elementos de la revisión necesarios para reducir la posibilidad de sesgo investigador:
  - **Antecedentes** que justifican la revisión.
  - Las **preguntas de investigación** establecidas.
  - La **estrategia de búsqueda**, que incluye los términos para ejecutar la búsqueda y los recursos en los cuales se realizará (bibliotecas digitales, revistas, congresos, entre otros).
  - Los **criterios de selección** de estudios para determinar cuáles estudios serán incluidos y cuales excluidos del proceso.
  - Los **procedimientos de selección** de estudios y resolución de conflictos frente a posibles discordancias entre el equipo investigador.
  - Los **criterios y actividades para la evaluación de calidad** de los estudios primarios seleccionados.
  - La **estrategia de extracción de datos**, es decir, como se obtendrá la información de cada estudio.
  - La **estrategia de síntesis** de datos para realizar un análisis y síntesis de la evidencia extraída de los estudios primarios.

2. **Ejecución**: Se realiza la búsqueda y selección de estudios primarios para extracción y clasificación de los datos con base en el protocolo de búsqueda planteado:

   - *Ejecutar la búsqueda*: Consiste en la identificación de las investigaciones relevantes mediante el desarrollo de la estrategia de búsqueda, puede involucrar una exploración preliminar con el ánimo de evaluar el volumen estimado de estudios y mejorar la definición de la cadena de búsqueda. El resultado de esta actividad es la extracción de las publicaciones desde los diferentes recursos. Para la construcción de la estrategia de búsqueda, un enfoque general es el de descomponer la pregunta en facetas individuales, es decir, población, intervención, comparación, resultados y contexto. A continuación, se elabora una lista de sinónimos, abreviaturas y palabras alternativas para cada faceta. Luego, cadenas de búsqueda sofisticadas se pueden construir

utilizando operadores booleanos AND y OR.

- *Selección de estudios primarios*: Una vez obtenidas las publicaciones relevantes para la investigación se aplican los criterios de inclusión y exclusión planteados en el protocolo de búsqueda con base en la pregunta de investigación para identificar los estudios primarios. Los criterios de selección pueden ser previamente evaluados mediante su aplicación manual sobre un estudio con alta probabilidad de ser incluido en la investigación, llamado estudio de referencia, para así determinar su precisión.
- *Evaluación de calidad de los estudios*: Sobre los estudios seleccionados se aplican criterios de evaluación de la calidad como medio de ponderar la importancia de los estudios individuales cuando se sintetizan los resultados.
- *Extracción de datos*: En esta etapa se registra la información obtenida de los estudios primarios a través de la estrategia de extracción de datos, en formularios diseñados para tal fin.
- *Síntesis de datos*: Consiste en la recopilación y resumen de los resultados mediante la estrategia de síntesis planteada en el protocolo. Varios de los datos relacionados con población, contexto, tamaño de la muestra y calidad de los estudios podrán ser tabulados usando una técnica cuantitativa, mientras que otros datos requerirán de una técnica descriptiva de síntesis.

3. **Resultados**: En esta etapa se lleva a cabo la diseminación de resultados y conclusiones:

   - *Resultados del informe*: El objetivo de esta actividad es presentar los resultados y conclusiones obtenidos tras la ejecución de la revisión sistemática a través de un análisis bibliométrico de los principales datos y atributos extraídos de los estudios primarios.
   - *Análisis Comparativo*: Dado que uno de los objetivos de la revisión sistemática propuesta en este trabajo consiste en el planteamiento de un *framework* de clasificación taxonómica de los atributos de adaptabilidad identificados en los estudios, en esta actividad, se presenta un esquema de clasificación de estos criterios y mecanismos.
   - *Refinamiento de la taxonomía*: Con base en el esquema de clasificación, se plantea, revisa y refina la estructura de un *framework* de clasificación taxonómica de los criterios y mecanismos de adaptación.

## 1.6. Estructura del trabajo

El contenido de este documento está distribuido siguiendo la estructura descrita a continuación:

- Capítulo 2: Fundamentos y trabajo relacionado:
  En este capítulo se introducen los conceptos clave para la comprensión de los temas abordados en este trabajo y se discuten los trabajos relacionados.

- Capítulo 3: Revisión sistemática de la literatura y taxonomía:
  En este capítulo se presenta el diseño, ejecución y resultados de la revisión sistemática de la literatura respecto a sistemas de software adaptativos orientados a micro-servicios y/o desplegados en contenedores. Como parte de los resultados, se describe un análisis bibliométrico y taxonomía de los estudios primarios identificados.

- Capítulo 4: *Framework* de adaptación:
  En este capítulo se presenta el *framework* propuesto para la adaptación de sistemas basados en micro-servicios y contenedores en *cloud* en conjunto con una estrategia para su incorporación en el marco de un proceso DevOps y una ejemplificación de su aplicación.

- Capítulo 5: Conclusiones:
  En este capítulo se describen las contribuciones más relevantes del presente trabajo y las posibles líneas de trabajo o investigación emergentes.

# Capítulo 2

# Fundamentos y Trabajo Relacionado

El rápido proceso que guía el camino hacia la transformación digital exige agilidad y flexibilidad para adaptarse a las necesidades y cambios emergentes. Las empresas de desarrollo de software han adoptado metodologías, patrones de diseño y herramientas que apoyen dicho propósito. Sin embargo, los procesos y la forma de desarrollar, desplegar y mantener software y su infraestructura están en constante cambio. La integración y el despliegue continuos son claves en la mejora y automatización de los procesos de desarrollo e implantación de software. Grandes empresas tales como Facebook, Netflix y Google han sido pioneras en tecnologías y prácticas de integración y despliegue continuo de aplicaciones. Sin embargo, esta experiencia aún no ha sido trasladada a pequeñas y medianas empresas para ayudar en sus procesos de integración y despliegue ya que demandan procesos más ágiles y herramientas personalizables que les permitan tratar la evolución en el tiempo adecuado y con unos costes razonables.

En este capítulo, se discuten algunas aproximaciones orientadas al soporte de la adaptación y evolución de sistemas software basados en servicios.

En la sección 2.1 se introducen los conceptos de micro-servicios y contenedores, así como su uso en sistemas *cloud*, la sección 2.2 se definen los principales conceptos y reglas asociados a DevOps. Por último, en la sección 2.3, se presenta el estado actual de la investigación relacionada con adaptabilidad, micro-servicios y contenedores en *cloud*.

## 2.1. Micro-servicios y contenedores en entornos *cloud*

La adopción de patrones de arquitectura que incrementen los beneficios de la computación *cloud* se hizo necesaria conforme esta última se posiciona como uno de los principales entornos para el desarrollo de software, debido a su versatilidad y flexibilidad en el despliegue y mantenimiento de las aplicaciones [15] [17]. La arquitectura basada en micro-servicios apareció como un paradigma de programación mediante pequeños componentes. Sus bases se encuentran en los conceptos de SOA (Service Oriented Architecture) y OO (Orientación a ob-

jetos). Provee cohesión en funciones de negocio mediante la división del sistema en pequeños servicios, facilitando el diseño de sistemas complejos [20]. Los sistemas con este tipo de arquitectura son más ágiles, resilientes, escalables a la vez que simplifica su implementación, mantenimiento y despliegue respecto a sistemas con arquitecturas monolíticas [17].

Los micro-servicios cooperan para proveer complejidad y agregar funcionalidades al sistema, esto resalta la importancia de que cada uno de ellos sea independiente y su función dentro del sistema se encuentre claramente definida. Sin estos dos aspectos no sería posible lograr la mantenibilidad del sistema mientras éste evoluciona mediante la adición de nuevas características. Independencia, en este contexto, implica la posibilidad de establecer para cada servicio su propio entorno de ejecución (arquitectura, plataforma, etc.), ciclo de desarrollo y que su despliegue y operación no dependa de los demás micro-servicios que componen el sistema [66].

Al ser la independencia, modularidad y cohesión las principales características de los micro-servicios, es importante que la plataforma e infraestructura que alberga el software sea flexible, mientras enfrenta los principales retos asociados a este tipo de arquitectura: la seguridad y el desempeño, heredados de SOA, debido al intercambio de información a través de la red [20]. En este escenario, los contenedores se presentan como un aliado, ya que proporcionan mayor escalabilidad al sistema, así como la reducción de tiempos y esfuerzos en la implementación y operación de las aplicaciones [60].

Los contenedores surgen como una alternativa ligera a las máquinas virtualizadas en el contexto de computación *cloud*, pero orientados al despliegue de aplicaciones mediante el modelo PaaS (*Platform as a Service*), haciendo uso del concepto de "*containerization*", es decir, la implementación de una aplicación o sus componentes en contenedores [48] [36]. De acuerdo con lo expuesto en [21], un contenedor es *"Un sistema operativo liviano que se ejecuta dentro del sistema host, ejecuta instrucciones nativas de la CPU central, eliminando la necesidad de emulación a nivel de instrucción o compilación just in time. Los contenedores proporcionan ahorros en el consumo de recursos sin la sobrecarga de la virtualización, al tiempo que proporcionan aislamiento"*. Los contenedores proporcionan a los sistemas dos características fundamentales para trabajar con componentes: portabilidad y modularización [38]. Lo anterior, sumado a su capacidad para proveer tiempos rápidos de arranque del sistema, capacidad de interconexión y consumo bajo de recursos en tiempo de ejecución ha logrado que los contenedores potencien el uso de arquitecturas de micro-servicios [36]. Una aplicación con una arquitectura basada en micro-servicios y contenedores desplegada en el *cloud* tendrá varias ventajas: su complejidad se verá reducida, será más fácil su implementación, operación y evolución, tanto general como de sus partes y la capacidad de recuperación del sistema será mejor.

## 2.2. DevOps

DevOps es una metodología que promueve la comunicación y colaboración entre los equipos de desarrollo y operaciones de una organización con el fin de que se genere valor de manera ágil y continua. Dado que DevOps mejora la forma en la que el negocio entrega valor a los *stakeholders*, constituye un proceso de negocio, más que una simple capacidad en el ámbito de TI [56]. El origen del término DevOps (Abreviación de *Development* & *Operations*) se remonta a 2010, año en el que fue adoptado por el belga Patrick Dubois en la organización de los *DevOps Days* como una estrategia para buscar solución a los problemas que se generaban en el despliegue de desarrollos cada vez más ágiles.

Para adoptar DevOps en la organización, IBM fue pionera en presentar una arquitectura de referencia, figura 2.1, que sugiere seguir cuatro caminos (caminos de adopción). En cada uno de los caminos se han de implantar una o varias prácticas:

1. Dirigir: Este camino adopta como práctica la planeación continua del negocio. Del planteamiento claro de objetivos del negocio y su evolución en conformidad con la retroalimentación del cliente dependerá en buena medida la correcta implantación de la metodología DevOps.

2. Desarrollo / Pruebas: Este camino abarca dos practicas:

   - Desarrollo colaborativo entre los diferentes roles que componen el o los equipos de desarrollo, ejecutando actividades de integración continua.
   - *Testing* continuo y temprano de las características mediante la adopción de capacidades como las pruebas automatizadas y la virtualización de servicios. Esto se traduce en la reducción de costos y tiempos de ciclos de prueba.

3. Despliegue: En este camino se busca la automatización de los procesos de liberación y despliegue continuo de funcionalidades en entornos de producción mediante el manejo de un pipeline para entrega primero a QA (*Quality Assurance* - Aseguramiento de Calidad) y luego al cliente [55].

4. Operación: En este camino se deben implementar dos prácticas que permitirán conocer el nivel de satisfacción del cliente respecto a los productos y tomar decisiones ágiles para dar respuesta a sus necesidades:

   - Monitoreo continuo para llevar control sobre el estado del producto en cualquiera de las fases, no necesariamente en producción.

- Retroalimentación continua del cliente y optimización para identificar los "puntos de dolor" y apreciaciones del cliente con el fin de ajustar los objetivos de negocio si es necesario.

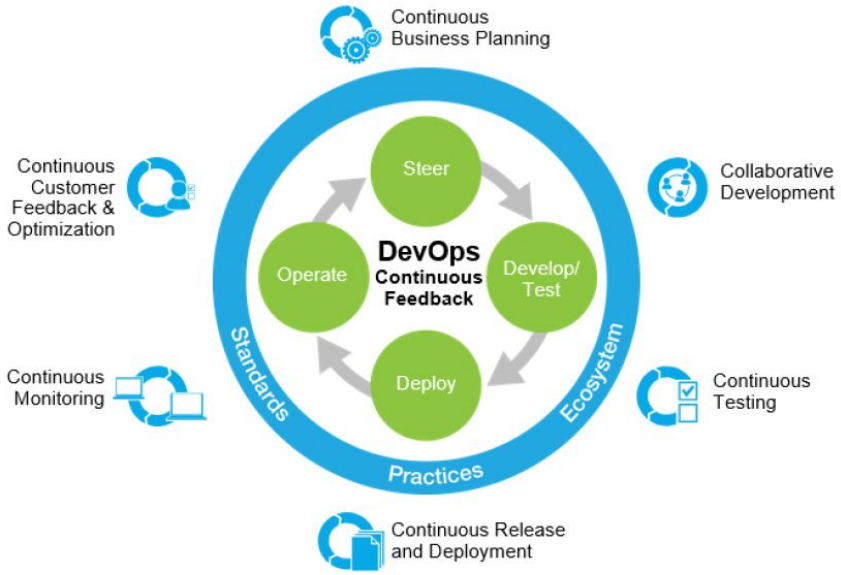


Figura 2.1: Arquitectura DevOps, según Sharma y Coyne [56]

El retorno de inversión en la adopción de DevOps se refleja en tres contextos:

1. Experiencia del cliente: Mejorar la experiencia de los clientes requiere que tengamos retroalimentación continua que nos permita entender y dar respuesta a sus necesidades, esto usualmente se logra a través de aplicaciones de compromiso (*engagement systems*) a las cuales el cliente accede constantemente.

2. Capacidad de innovación: Aumentar la innovación no siempre significa tener productos nuevos, sino reducir tiempo y esfuerzo en reprocesos de actividades con un alto valor para los interesados.

3. Entrega de valor ágil: Reducir el tiempo que tarda la compañía en entregar valor, significa automatizar aquellos procesos que se consideran críticos de acuerdo con lo que se haya definido como valor dentro de la organización.

Uno de los ejes fundamentales en la implementación de DevOps es la automatización, por esta razón en el mercado se encuentran diversas herramientas que apoyan las diferentes fases del proceso de ingeniería continua como se puede apreciar en la figura 2.2. Adicional a una infraestructura automatizada y métricas de control es importante adoptar valores en la organización que soporten el trabajo ágil y en equipo: respeto por la opinión, experiencia y decisiones tomadas por cada uno de los miembros del equipo; confianza entre las personas,

fomentando que se comparta información de manera transparente; tolerancia a fallos para lograr resolverlos en tiempo mínimo y estar preparados para afrontarlos cuando se presenten sin culpar a nadie respecto a lo sucedido.

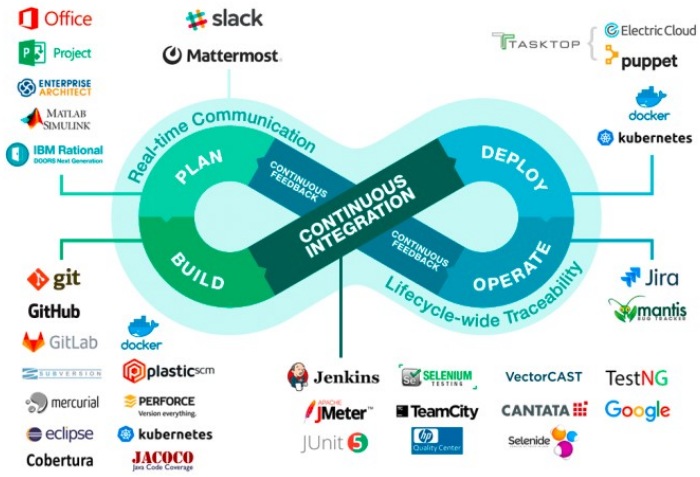


Figura 2.2: Herramientas DevOps. Tomado de `https://devops.com.vn`

El auge de la computación *cloud* durante los últimos años y sus estrictos requerimientos de calidad ha impulsado la necesidad de evolucionar los mecanismos y metodologías de desarrollo y entrega de las aplicaciones [15] [60], a su vez se ha convertido en un habilitador en la adopción de DevOps mediante la solución de problemas tecnológicos a los cuales se enfrenta el proceso tales como la reducción de cuellos de botella en el aprovisionamiento de los ambientes requeridos para el desarrollo y operación de las aplicaciones, la disponibilidad de herramientas para la automatización del despliegue y pruebas de las soluciones [56].

Respecto a la adopción de DevOps en entornos *cloud*, en [15] Cito et al. presentan los resultados de una encuesta realizada a 2.000 usuarios de GitHub, con 294 respuestas, mediante la cual se pretendía identificar cómo percibían las personas participantes el impacto de la computación *cloud* en el proceso de desarrollo y operación y qué herramientas/datos utilizaban con mayor frecuencia para su trabajo. Encontraron que, para los desarrolladores, la escalabilidad y elasticidad son los principales conceptos de impacto *cloud*, puesto que facilita el mantenimiento y evolución de la aplicación, sin embargo, también crea relaciones de dependencia para tener en cuenta en la fase de diseño. Respecto a los datos y herramientas, aquellos orientados al monitoreo del desempeño son los más adoptados. El 60 % de las empresas implementan su proceso de desarrollo con base en los conceptos de DevOps y consideran que esto ha mejorado la comunicación entre los equipos de desarrollo y operación, a pesar de ello, emplean un ingeniero dedicado al mantenimiento del código de la infraestructura. Taibi et al. en [61] mapea las técnicas específicas de micro-servicios existentes para comprender cómo entregar continuamente valor en un pipeline DevOps y caracterizan los diferentes estilos arquitecturales y

patrones para el diseño de micro-servicios con el fin de identificar las técnicas y herramientas DevOps adoptadas. Encontrando que no se propusieron técnicas para liberación o replanteamiento de la arquitectura de sistemas y que las fases del ciclo de desarrollo de sistemas basados en micro-servicios mayormente exploradas en términos de DevOps corresponden a los pasos de pruebas, despliegue y monitoreo, cuyas técnicas están principalmente orientadas a la automatización del proceso.

La relación entre el proceso DevOps y las capacidades de adaptación del sistema se ha explorado en varios trabajos: Barna et al. en [10] se enfoca en el escalado autónomo de una aplicación desplegada en contenedores en el *cloud* con múltiples niveles de aplicación y capas de infraestructura, mediante un mecanismo que actúa como integrador entre los equipos de desarrollo y operaciones. El equipo de desarrollo, en la fase de diseño, se encarga de perfilar la aplicación en términos de los recursos de infraestructura que requiere para operar, mientras que el sistema autónomo propuesto se asegura de regular el rendimiento de la aplicación en concordancia con el modelo plateado por el equipo de desarrollo y actualiza continuamente las condiciones del entorno mediante una herramienta DevOps. En [13], Brueno et al. utilizan los principios de DevOps para crear una infraestructura en la que, mediante el uso de monitoreo programable y abstracción de datos en línea, se obtenga información relevante para la optimización del ecosistema de *CloudWave*, un paradigma para el desarrollo y gestión de aplicaciones en la nube. En [18], De Sanctis et al., realizan un mapeo entre el ciclo de vida de un diseño para la adaptación de aplicaciones en el ciclo de vida clásico de DevOps y proponen las actividades a realizar en cada fase en el marco de una aplicación consciente de los niveles de calidad del servicio (QoS). Por otra parte, en [11], se propone un *framework* con enfoque en DevOps y arquitectura MAPE-K para la adaptación en tiempo de ejecución de sistemas de software web, enfocado en la prevención de vulnerabilidades respecto a las políticas de seguridad y privilegios de los usuarios.

En las propuestas revisadas, la adopción del enfoque y herramientas DevOps facilita la implementación en tiempo de ejecución de acciones diseñadas y construidas por el equipo de desarrollo. Sin embargo, con excepción de la aproximación presentada en [18], no se analiza su incorporación en diferentes fases del ciclo de vida de las aplicaciones adaptativas.

## 2.3. Adaptabilidad en sistemas *cloud*

La capacidad de adaptación de un sistema se traduce en la ejecución de una serie de procesos para extender o modificar sus dimensiones con el objetivo de habilitar o mejorar funcionalidades, interacciones y aspectos de comunicación entre los componentes del sistema o con su entorno. Concretamente en la capa de software de los sistemas se distinguen tres di-

mensiones: comportamiento, requerimientos y estructura. [34] [58]. Durante los últimos años los estudios en contextos cloud o micro-servicios se han enfocado en la adaptación orientada a características de calidad o requerimientos no funcionales de los sistemas, como es el caso de [26] en el que Fokaefs et al. presentan los resultados de un estudio empírico para la evaluación de tres métodos de adaptación sobre una aplicación *web* publicada en *cloud* y su impacto sobre la efectividad en el manejo de recursos, particularmente de CPU de las máquinas en las que se aloja la aplicación. Pahl et al. en [49] realizan una clasificación taxonómica de los estudios orientados a la implementación de tecnologías basadas en contenedores en entornos *cloud* que provee una imagen del creciente interés académico e industrial por el uso de contenedores, así como las propuestas existentes; sin embargo, en el análisis del componente arquitectural se centran solo en mecanismos de adaptación en términos de recursos, tales como la elasticidad, encontrando que existen un balance entre estos, los objetivos de nivel de servicio y los parámetros de calidad de la infraestructura.

Se encuentran varios estudios que orientan su investigación a la evaluación de la adaptación en términos funcionales, es decir, comportamiento y requerimientos, al igual que el presente trabajo. Es el caso de Jamshidi et al., quienes presentan en [33] los resultados de una clasificación taxonómica de las investigaciones con enfoque en la evolución del software centrado en la arquitectura, o Macías-Escrivá Frank et al. [42], quienes abordan los principales conceptos, herramientas, métodos y aplicaciones que habilitan la capacidad de autoadaptabilidad desde la perspectiva computacional. Sin embargo, al igual que en estos dos casos, en la mayoría de las investigaciones no se introducen en la búsqueda los conceptos de micro-servicios o contenedores. La excepción ocurre en la investigación de Bogner et al. [12], la cual reúne las diferentes tácticas de modificabilidad arquitectónica definidas para la evolución de sistemas y las distribuye en tres categorías: Aumento de la cohesión, reducción del acoplamiento y aplazamiento del tiempo de enlace. Esta lista es posteriormente comparada con ocho principios de los micro-servicios con el fin de identificar en qué medida se encuentran los dos conceptos relacionados y los patrones de diseño que se pueden utilizar para realizar las tácticas de modificabilidad en el contexto de micro-servicios, encontrando que la mayoría de las relaciones está orientada a la reducción del acoplamiento mediante patrones tales como *API Composition*, *API Gateway* y *Backend for Front End*.

De acuerdo con lo expuesto anteriormente, se puede evidenciar como la computación en la nube ha promovido el desarrollo de sistemas adaptativos en aras de mejorar la calidad del servicio. Sin embargo, las aproximaciones de adaptación de software orientadas a la evolución del sistema para dar cumplimiento a las necesidades funcionales del cliente aún restan por ser exploradas para entregar herramientas a los ingenieros que les permita construir aplicaciones conscientes de su entorno y con la capacidad de adoptar fácilmente nuevos comportamientos.

# Capítulo 3

# Revisión Sistemática de la Literatura y Taxonomía

El principal objetivo del trabajo presentado en este capítulo consiste en la revisión de los estudios primarios en el área de sistemas de software adaptativos cuya arquitectura se encuentre basada en microservicios y/o contenedores, con el fin de responder las preguntas de investigación planteadas y posteriormente clasificar la evidencia obtenida para así contar con una perspectiva amplia del desarrollo del tema en la investigación existente. Como resultado de la clasificación se construirá una taxonomía que proporcione una terminología común para los conceptos relacionados con la adaptación de este tipo de sistemas.

Para lograrlo, se define un procedimiento basado en una revisión sistemática, mecanismo que permite obtener una visión amplia del tema objeto de investigación y es utilizado principalmente para establecer si existe evidencia de investigación sobre un tema y la cantidad de evidencia hallada, así como la carencia de evidencia con el ánimo de dar dirección a la investigación. El uso de una metodología claramente definida, como el caso de las revisiones sistemáticas, reduce la probabilidad de sesgo en los resultados obtenidos y permite el uso combinado de datos mediante técnicas de meta-análisis [37].

El estudio se lleva a cabo en tres fases tomando como guía el planteamiento realizado por Kitchenham et al. en [37]: *planeación*, *ejecución* y *documentación (resultados)*. El procedimiento adoptado se puede observar en la figura 3.1.

Las actividades ejecutadas y resultados obtenidos en cada una de las fases se describen a continuación. En la sección 3.1 se presentan las actividades relacionadas con la fase de planeación, las actividades que conciernen a la fase de ejecución se describen en la sección 3.2 y, por último, en las secciones 3.3 y 3.4 se detallan los resultados obtenidos y la taxonomía propuesta, respectivamente.

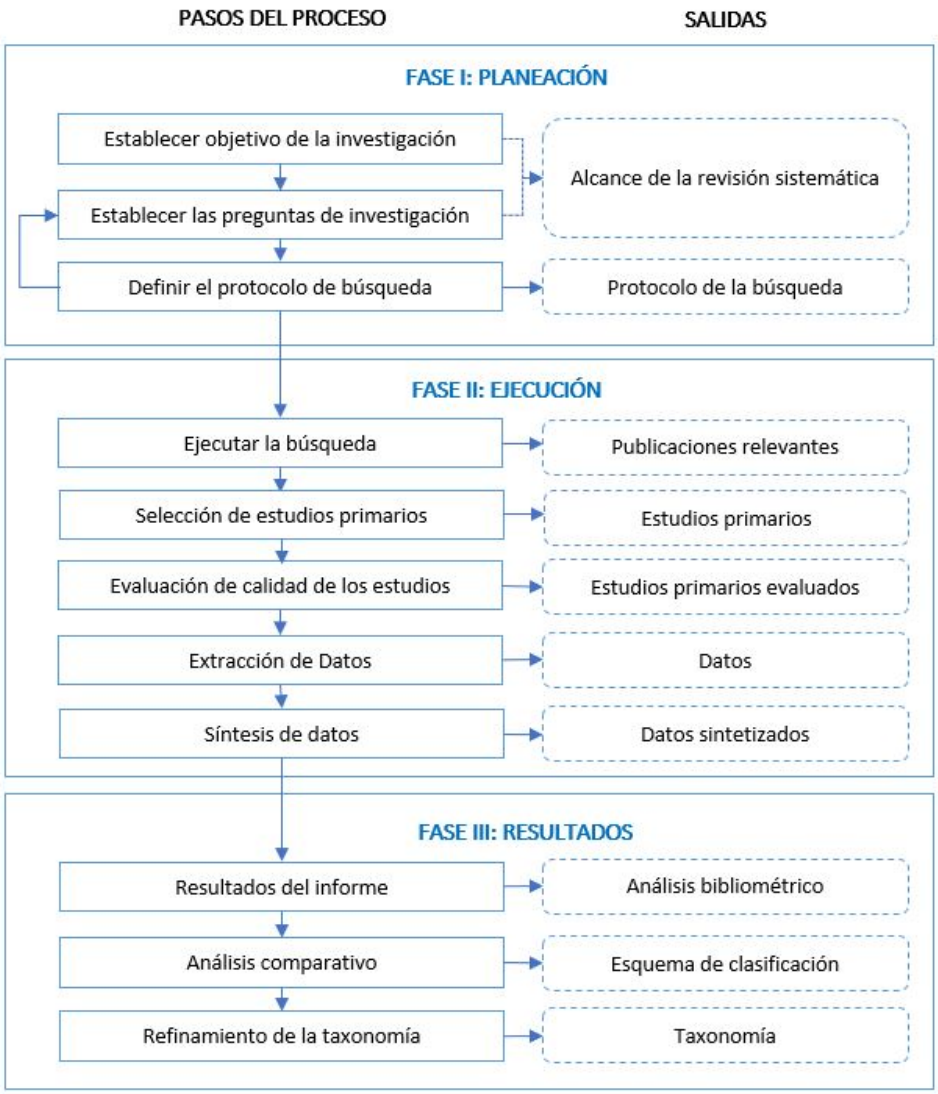


Figura 3.1: Fases del método de investigación. Elaborado según Kitchenham et al. [37]

## 3.1. Planificación

Esta fase comprende la identificación de la necesidad de realizar la revisión sistemática, que surge como parte del proceso de entendimiento del estado actual del tema a desarrollar en el presente trabajo y proporciona un marco de referencia que guía la posterior ejecución de este. Como resultado de la fase de planificación son planteados: el objetivo de investigación definido en la subsección 3.1.1, la formulación de las preguntas de investigación presentadas en la subsección 3.1.2, por último, se realiza el desarrollo del protocolo de búsqueda, detallado en la subsección 3.1.3.

### 3.1.1. Objetivo de investigación

El estudio tiene por objetivo realizar una revisión sistemática de la literatura respecto a las **estrategias de adaptación en sistemas software cuya arquitectura esté basada en micro-**

**servicios o hagan uso de contenedores** con el fin de identificar el estado actual de estas tecnologías en el proceso de desarrollo de software y clasificar los principales conceptos relacionados con la adaptación en una taxonomía.

### 3.1.2. Preguntas de investigación

El planteamiento de las preguntas de investigación se realizó con base en la guía proporcionada por Kitchenham et al. en [37] y siguiendo los conceptos introducidos por Kell el al. en [34]. El resultado constituye un conjunto de preguntas formuladas bajo la orientación del objetivo anteriormente expresado y una estructura básica para la posterior definición de la taxonomía de los conceptos de adaptación.
A continuación, se presentan las preguntas de investigación que guiarán el estudio y la correspondiente motivación que les da sustento:

- **PI1:** ¿Qué tipos de propuestas fueron planteadas para soportar la adaptación de sistemas software basados en micro-servicios o contenedores?
  **Motivación:** Caracterizar las diferentes aproximaciones propuestas con el propósito de lograr la adaptación en sistemas de software.

- **PI2:** ¿Qué tipos de adaptación se presentan en sistemas software basados en micro-servicios o contenedores?
  **Motivación:** Identificar las clases de adaptación que se han propuesto, diseñado o implementado, así como las etapas en las que se adoptan.

- **PI3:** ¿Qué factores motivan la necesidad de adaptación en sistemas software basados en micro-servicios o contenedores?
  **Motivación:** Identificar y analizar las causas o necesidades subyacentes al uso de mecanismos de adaptación en sistemas basados en micro-servicios o que hagan uso de contenedores.

- **PI4:** ¿Qué técnicas y herramientas se han empleado en la adaptación de sistemas software utilizando micro-servicios o contenedores?
  **Motivación:** Se busca con esta pregunta conocer el listado de herramientas que han sido utilizadas en las diversas fases del ciclo de vida de los sistemas basados en micro-servicios, principalmente aquellas orientadas al uso de contenedores, para así, posteriormente identificar el nivel de automatización empleado.

- **PI5:** ¿Qué tipo de evaluación se ha realizado para validar las aproximaciones, técnicas y/o herramientas propuestas?

**Motivación:** El objetivo de esta pregunta es identificar el tipo de estudio que se ha llevado a cabo para evaluar o validar las distintas aproximaciones, técnicas o herramientas propuestas.

### 3.1.3. Protocolo de búsqueda

Este paso de la fase de planeación proporciona un marco de trabajo con el ánimo de dar respuesta a las preguntas de investigación planteadas, que consiste en una estrategia de búsqueda a aplicar para la extracción de los estudios desde diferentes librerías, los criterios de selección y evaluación de calidad, así como las actividades a ejecutar para la selección y validación de los estudios primarios y las estrategias de extracción y síntesis de datos.

#### 3.1.3.1. Estrategia de búsqueda

La búsqueda de los estudios se ejecuta en los cuatro motores de búsqueda más utilizados en las revisiones sistemáticas realizadas en el ámbito de la Ingeniería del Software [37]:

- *IEEE Xplore* (`https://ieeexplore.ieee.org/`)
- *ACM Digital Library* (`https://dl.acm.org/`)
- *Science Direct* (`https://www.sciencedirect.com/`)
- *Springer Link* (`https://link.springer.com/`)

Los siguientes pasos fueron realizados con el fin de construir la cadena de búsqueda:

1. Identificación de los principales términos dentro de las preguntas de investigación.
2. Identificación de posibles sinónimos de estos conceptos clave.
3. Uso del operador OR para incorporar a la búsqueda los sinónimos establecidos en el punto 2.
4. Uso del operador AND para la vinculación de los términos clave.
5. Se realizaron varias pruebas piloto para refinar la cadena de búsqueda y adaptarla a cada una de las librerías digitales.

En la tabla 3.1 se exponen los conceptos bajo los que se define la cadena de búsqueda utilizada:

*(ADAPT*) AND ((MICROSERVICE* OR MICRO-SERVICE* OR "MICRO SERVICE") OR (CONTAINER* OR KUBERNET* OR DOCKER))*.

Donde el asterisco '*' significa cualquier conjunto de caracteres, lo cual hace posible incorporar variaciones de los conceptos a la búsqueda.

Tabla 3.1: Conceptos de Búsqueda

| CONCEPTOS | PALABRAS CLAVE | DESCRIPCIÓN | CONECTOR |
|---|---|---|---|
| ADAPTACIÓN | ADAPT* | Estudios que discutan adaptabilidad, adaptación, autoadaptación. En las bibliotecas que no admiten *wildcars*, se usarán los términos ADAPTIVE OR ADAPTATION | AND |
| MICRO-SERVICIOS | MICROSERVICE* OR MICRO-SERVICE* OR MICRO SERVICE | Estudios que discutan *microservice*, *microservices*, *microservice-based*, *microservice-oriented* o *microservice architecture* | OR |
| CONTENEDORES | CONTAINER* OR KUBERNET* OR DOCKER | Estudios que incorporen los términos *container* y *containers*. Se encuentra en algunos documentos el término que denota la herramienta utilizada, por lo que se incluyen las dos más comunes | |

De acuerdo con [33] el término *microservice* como aproximación arquitectural fue acuñado a mediados de 2011, por otro lado, la "contenerización" de aplicaciones, aunque nace con LXC, un método de virtualización a nivel de sistema operativo diseñado en 2008 para ejecutar múltiples sistemas aislados en un host utilizando un solo kernel de Linux, solo toma fuerza con la aparición de *Docker* en 2013 y herramientas de orquestación que facilitaron el uso de contenedores para el empaquetamiento y despliegue de aplicaciones, tales como *kubernetes* en 2014, *Docker Swarm* en 2014 y *Amazon Elastic Container Service* en 2015. En consecuencia, se ha decido tomar como referencia para la búsqueda de los estudios primarios el periodo comprendido entre **2012 y 2019**.

#### 3.1.3.2. Estrategia de selección

Los criterios de selección de estudios se utilizan para determinar cuáles de los estudios obtenidos en la búsqueda inicial se incluyen o excluyen según su grado de relevancia en la investigación.

Para la selección de estudios primarios, se tuvieron en cuenta artículos científicos que presentan aproximaciones o herramientas utilizadas para la adopción o implementación de mecanismos de adaptación en sistemas basados en micro-servicios o contenedores. Así como estudios que comparan o evalúan las aproximaciones o herramientas propuestas dentro del mismo marco. En este orden de ideas, para ser seleccionado un artículo como estudio primario debe cumplir con alguno de los siguientes aspectos:

1. **Criterios de Inclusión**:
   - CI1: Artículos que exponen claramente el uso de micro-servicios o contenedores en

el diseño y/o construcción de sistemas de software para la implementación de estrategias o mecanismos de adaptación.

- CI2: Artículos que proponen o presentan arquitecturas basadas en micro-servicios y/o contenedores con capacidades de adaptación.

2. **Criterios de Exclusión**: Se excluyen los estudios que cumplan con alguno o varios de los siguientes criterios:

   - CE1: Definan la adaptación para sistemas cloud en capa de infraestructura o plataforma, ya que no es de interés para la investigación.
   - CE2: Hagan referencia a adaptación, escalabilidad y ubicación de recursos en capa de infraestructura.
   - CE3: No están relacionados con el modelo actual de contenedores. Por ejemplo, artículos que hacen referencia a contenedores Java o contenedores de control.
   - CE4: Se encuentren orientados a la adaptación en proyectos de virtualización o desacoplamiento de redes de hardware tales como VFN (*Network functions virtualization*) o SD-WAN (*software-defined networking in a wide area network*).
   - CE5: No propongan explícitamente un método, herramienta o técnica para facilitar la implementación de mecanismos de adaptación.
   - CE6: No se desarrollen en el dominio de software o computación en la nube.
   - CE7: Se encuentren duplicados.
   - CE8: No se encuentren escritos en inglés.
   - CE9: Corresponden con tutoriales, *keynotes*, resúmenes, introducciones a *workshops* o capítulos de libros.
   - CE10: Artículos con menos de 6 páginas (*shortpapers*).
   - CE11: No fueron presentados en conferencias o revistas con revisión por pares. Corresponden con vistas previas o documentos que solo tienen disponible el *abstract*.
   - CE12: Presenten resultados de revisiones sistemáticas de la literatura relacionados con el objeto de investigación, es decir, estudios secundarios o estudios que comparen o evalúen las aproximaciones propuestas o herramientas para la adopción o implementación de mecanismos de adaptación en sistemas basados en micro-servicios o contenedores, pues este tipo de artículos se discuten en trabajos relacionados.

Tras la aplicación de los criterios de exclusión e inclusión, se realizan dos pasos: un escaneo rápido de todos los estudios extraídos (lectura del título, resumen y palabras clave - y si fuera necesario la introducción y conclusiones) y una lectura completa de estudios seleccionados.

#### 3.1.3.3. Estrategia de evaluación

Con el fin de evaluar la rigurosidad con la que se han realizado los estudios seleccionados, se ejecuta una valoración con base en las características detalladas en la lista de verificación expuesta en la tabla 3.2. Los criterios se encuentran agrupados en dos tipos: *General (G)*, aquellos que definen la calidad del estudio independientemente de su propósito y *específico (S)* que nos permite identificar la pertinencia del artículo respecto al objetivo de la revisión sistemática planteada y por esto, les es asignado un peso mayor (75 %) que a los criterios generales (25 %). Cada uno de los elementos de evaluación AI(*Assessment Item*) se calificó con el valor 1 si el estudio cumplía dicha característica, 0,5 si la cumplía parcialmente y 0 en caso contrario. El ponderado total para los artículos se calculó con base en la fórmula propuesta en [32] y adaptada a los criterios establecidos para este trabajo.:

$$\textit{Calificación Estudio} = \frac{\sum_{i=1}^{3} GAI_i}{3} + \frac{\sum_{i=1}^{4} SAI_i}{4} X3 \tag{3.1}$$

Tabla 3.2: Elementos de evaluación de calidad

| TIPO | AI | DESCRIPCIÓN |
|---|---|---|
| Específico | SAI1 | ¿La investigación se centra en estrategias de adaptación de software en términos funcionales o de comportamiento del sistema? |
| | SAI2 | ¿El uso de micro-servicios o contenedores apoya o facilita la adopción / implementación de los mecanismos de adaptación propuestos? |
| | SAI3 | ¿La motivación, técnica y tipos de adaptación se encuentran claramente especificados? |
| | SAI4 | ¿Los resultados se encuentran claramente validados en un contexto no trivial? |
| General | GAI1 | ¿Se define claramente el tipo de propuesta de la investigación? |
| | GAI2 | ¿El dominio de aplicación se encuentra especificado? |
| | GAI3 | ¿Las conclusiones y contribuciones del estudio son coherentes con los resultados presentados? |

Los estudios cuya calificación se encuentra entre los valores 3 y 4 son considerados de buena calidad, aquellos cuya calificación está en el rango de 2 a 3 son aceptables. Por último, los estudios con una calificación inferior a 2 son excluidos.

#### 3.1.3.4. Estrategia de extracción de Datos

El proceso de extracción y síntesis de datos permite obtener desde los estudios primarios, la información relevante para dar respuesta a las preguntas de investigación planteadas. La estrategia de extracción de datos, en este sentido, proporciona un marco para la clasificación de los atributos de adaptabilidad identificados en los estudios.

El primer paso entonces consiste en la recopilación de los datos básicos de cada una de las publicaciones: fecha de publicación, tipo (revista, conferencia, otros) y fuente de la publicación,

autores y tipo de propuesta (solución, opinión, revisión, otros). Luego, se han adaptado las cuatro dimensiones propuestas por Buckley et al. [14] *(when, where, what, and how)* para presentar una imagen clara de la evolución de las soluciones de adaptación que utilizan microservicios o contenedores en la nube. Fue agregada una quinta dimensión *why*, puesto que es de vital relevancia para este trabajo identificar los motivadores de la adaptación en los estudios primarios. En particular, nuestro análisis especifica qué, cómo, cuándo, dónde y por qué estas soluciones de adaptación se aplican a un entorno de computación en la nube. Las dos secciones siguientes detallan la justificación a la que se adhirió al decidir si las dimensiones se clasificaron como características del mecanismo o como factores influyentes. Por último, recopilamos los datos del estudio necesarios para abordar nuestras preguntas de investigación mediante un formulario de extracción de datos, ver Apéndice B.

La Tabla 3.3 resume los criterios de extracción de datos utilizados, que se explican con más detalle a continuación, organizados bajo las cinco dimensiones ya mencionadas: Propiedades temporales (*when*- cuándo se realiza el cambio), objeto de cambio (*where*- dónde se realiza un cambio), propiedades del sistema (*what*- qué se está cambiando), soporte del cambio (*how*- cómo se logra el cambio) y motivadores del cambio (*why*- por qué se está realizando el cambio).

Tabla 3.3: Criterios de Extracción

| DIMENSIÓN | CRITERIO | POSIBLES OPCIONES |
|---|---|---|
| Cuándo | Tiempo de adaptación | Estático, Tiempo de carga, Dinámico |
| | Frecuencia de adaptación | Periódica, Arbitraria, Continua |
| Dónde | Dominio de aplicación | Cloud Computing, Sistemas Distribuidos, SOA, Sistema Ubicuos |
| | Contexto de uso | Académico, Industrial |
| | Fase de adaptación | Análisis, Diseño, Implementación, Despliegue, Operación, Monitorización |
| Qué | Tipo de propuesta | Método, Herramienta, Metodología, Framework, Técnica, Enfoque, Modelo |
| | Condiciones de adaptación | Configuración, Protección, Optimización, Sanación |
| Cómo | Actividad de adaptación | Reactiva, Proactiva |
| | Mecanismo de adaptación | Abierto, Cerrado |
| | Nivel de autonomía | Completamente automático, Parcialmente Automático, Human- in the loop |
| | Lógica de adaptación | Probabilística, Basada en reglas, Basada en casos, Basada en lógica, Basada en ontología, Basada en evidencia, Basada en lógica difusa |
| | Herramienta de soporte | Nombre de la herramienta utilizada |
| | Protocolo o Estilo Arquitectural | MQTT, REST/HTTP, SOAP, Otros |
| | Modelo de proceso | DevOps, Desarrollo Ágil, DDM, Otro |
| | Método de evaluación | Experimento, CoS, Encuesta |
| Por qué | Propósito de adaptación | Correctivo, Perfectivo, Adaptativo, Preventivo |
| | Necesidades de adaptación | Nombre de la necesidad |

1. **Dimensión Cuándo - *When***: Los criterios en esta primera dimensión describen propiedades tales como cuándo se realizan los cambios adaptativos (tiempo de adaptación) y la frecuencia con la que ocurren (frecuencia de adaptación), descritas a continuación:

*a*) **Tiempo de adaptación**: Tiempo en el que se realiza el cambio que propicia la adaptación del sistema [14]. Puede ser:

- **Estático**: Los cambios son realizados en el código fuente, por lo tanto, es necesario recompilar la aplicación para ejecutar los cambios.
- **Tiempo de carga**: Se realizan los cambios cuando los componentes del software son cargados al sistema ejecutable.
- **Dinámico**: Los cambios ocurren en tiempo de ejecución.

*b*) **Frecuencia de adaptación**: Nivel de frecuencia con el que se ejecutan los procesos de adaptación en el sistema [14]. Puede ser:

- **Continua**: Sistemas con procesos menos formales de cambio que permiten incorporan cambios en su funcionamiento de manera continua.
- **Periódica**: Se incorporan cambios al sistema de manera periódica.
- **Arbitraria**: Se incorporan cambios al sistema según solicitud/cronograma de ejecución, usualmente mediante una ventana programada durante la cual el sistema no está disponible.

2. **Dimensión Dónde - *Where***: Esta dimensión consolida las ubicaciones en el sistema sobre las que se realizan los cambios y los mecanismos de soporte requeridos para tal fin. En esta dimensión se encuentran los siguientes criterios de extracción:

*a*) **Dominio de aplicación**: Este criterio identifica los tipos de sistemas que son sujeto de adaptación, es decir, las plataformas o estructuras tecnológicas y paradigmas sobre los que se ejecuta el sistema o aplicación con propósitos de adaptación. Por ejemplo: *Cloud computing* (SaaS, PaaS, IaaS), sistemas distribuidos, sistemas ubicuos, SOA, IoT, etc.

*b*) **Contexto de uso**: Contexto en el cual se realiza o implementa la propuesta, que pueden ser:

- **Industrial**: Si el caso de estudio se realiza en una empresa.
- **Académico**: Si es estudio se realiza en un contexto académico o estudios que utilizan ejemplos para su exposición o evaluación.

*c*) **Fase de adaptación**: Etapa del ciclo de vida del sistema en la que se realiza la adaptación [31]:

- **Análisis**: Etapa en la que se identifican los requerimientos para construcción del software.

- **Diseño**: En esta etapa se define la estructura del sistema mediante la construcción de una representación detallada de su arquitectura, modelos de datos, presentación y estructura de código.
- **Implementación**: Codificación del modelo construido en la fase de diseño. También se incorporan elementos para permitir la interacción entre diversos componentes o aplicaciones.
- **Despliegue**: Configuración e instalación de software en ambiente de producción.
- **Operación**: Fase de uso del sistema por parte de los usuarios. Esta fase incluye todas las operaciones relacionadas con el funcionamiento que deben realizarse de forma continuada durante toda la vida del software. Por ejemplo, ajuste de los recursos de acuerdo con la demanda o las características de crecimiento de las aplicaciones, la modificación dinámica de la infraestructura por causas de seguridad, rendimiento, entre otras, o la optimización de recursos y procedimientos que requieren cambios en el contexto de ejecución.
- **Monitorización**: Etapa en la que se realiza seguimiento al desempeño del sistema.

3. **Dimensión Qué - *What***: Esta dimensión se refiere al sistema de software que está experimentando la adaptación y sus atributos. Se incluyen en ella los siguientes criterios:

   *a*) **Tipo de propuesta:** El propósito de este criterio es identificar el tipo de contribución que realiza el estudio, puede consistir en alguna de las siguientes categorías:

   - **Método**: Un método proporciona el enfoque para resolver un problema o implementar una práctica [6]. Se clasifican en esta opción los estudios que presente una ampliación de un método existente o definición de un nuevo método de adaptación.
   - **Herramienta**: Diseño, implementación, reingeniería o análisis de una aplicación de software adaptativa o que implementa técnicas o modelos orientados a la adaptación.
   - **Técnica**: Una técnica comprende un paso especifico, usualmente en el marco de un procedimiento, con un resultado inmediato que puede ser medible y observable [6]. Se clasifican en esta opción los estudios que realicen un planteamiento, diseño o incorporación de una técnica de adaptación en sistemas software.
   - **Framework**: En esta categoría se incluyen todos los estudios que presenten un marco de referencia arquitectural o de diseño.

- **Enfoque/Aproximación**:Un enfoque es la propuesta o aproximación inicial de un modelo [57] que se origina a partir de teorías, conceptos e ideas para abordar un problema [6]. Los estudios primarios que presenten un enfoque o aproximación para abordar necesidades de adaptación en sistemas de software serán considerados en esta categoría.
- **Arquitectura**: Una arquitectura de software es la composición de los elementos fundamentales para la creación de un sistema software. En esta opción se clasifican los estudios que presenten una nueva arquitectura, diseño o estilo arquitectónico o patrones de diseño.
- **Modelo**: Representación esquemática de una situación, herramienta o estructura en un estado determinado [57].

*b*) **Condiciones de adaptación**: Condiciones bajo las que el sistema incorpora funciones de adaptación en tiempo de ejecución, es decir, cuyo tiempo de adaptación es dinámico [35]:

- ***Self-configuration*** **(Auto-configuración)**: Configuración autónoma y automática con base en políticas de alto nivel.
- ***Self-optimization*** **(Auto-optimización)**: Identificación de oportunidades de mejora en la eficiencia de su desempeño o costos.
- ***Self-healing*** **(Auto-curación)**: Detección y reparación de fallos a nivel de software y hardware
- ***Self-protection*** **(Auto-protección)**: Prevención y respuesta ante fallos y ataques.

4. **Dimensión Cómo -** ***How***: En esta dimensión se reúnen los criterios que determinan los mecanismos que soportan la adaptación en el sistema y el nivel de formalismo de la estrategia de adaptación. A continuación se describen los criterios que componen la dimensión:

*a*) **Actividad de adaptación**: Propiedad de los sistemas con tiempo de adaptación dinámico que determina si el sistema de software puede ser **Reactivo** (los cambios se impulsan externamente) o **Proactivo** (el sistema impulsa los cambios de forma autónoma). Usualmente, cuando la actividad es proactiva, el sistema hace uso de monitores que le permiten conocer el estado del sistema y el entorno para tomar decisiones de adaptación. Si es reactiva, el sistema recibe los datos del entorno a desde una entidad externa, a través de una interfaz [14].

*b*) **Mecanismo de adaptación**: Estrategias de adaptación propuestas en los estudios y que permiten identificar el tipo de adaptación implementada [47]. Puede ser:

- **Abierta**: Es posible incorporar nuevas funcionalidades o comportamientos al sistema en tiempo de ejecución.
- **Cerrada**: Tiene un número limitado de comportamientos de adaptación y no es posible agregar uno nuevo en tiempo de ejecución. Corresponden con sistemas *Self-contained* (Parcialmente automáticos).

*c*) **Nivel de Autonomía**: Grado de automatización de la adaptación de la propuesta. Para llevar a cabo sus actividades de adaptación, un sistema puede ser completamente auto adaptativo o requerir la ayuda de humanos. En este trabajo, cada propuesta de adaptación se clasifica de acuerdo con tres niveles de autonomía propuestos en [14]:

- **Completamente automático**: El proceso de adaptación es totalmente automático, no requiere ninguna intervención humana.
- **Adaptación autónoma o Parcialmente Automático**: El sistema requiere para determinadas actividades un nivel de participación humana.
- ***Human- in the loop* (Humano en medio)**: Existen tareas ejecutadas por humanos, usualmente actividades de verificación.

*d*) **Lógica de adaptación**: El sistema puede ser diseñado con base en diferentes tipos de lógicas para lograr los propósitos de adaptación. En este trabajo, cada propuesta de adaptación se clasifica de acuerdo con las siguientes lógicas de adaptación:

- ***Probabilistic-based***: Adaptación basada en un modelo probabilístico.
- ***Rule-based***: Adaptación basada en un conjunto de reglas. Por ejemplo, las aproximaciones que utilizan reglas de transformación de modelos para automatizar la adaptación se clasifican en esta categoría.
- ***Case-based***: Adaptación basada en razonamiento según casos.
- ***Logic-based***: Adaptación basada en lógica de predicados.
- ***Ontology-based***: Adaptación basada en una ontología.
- ***Evidence-based***: Adaptación basada en datos experimentales.
- ***Fuzzy-based***: Adaptación basada en lógica difusa.

*e*) **Herramientas de soporte**: El propósito de este criterio consiste en conocer las herramientas que soportan la adaptación dentro de la propuesta y la fase que apoyan. Además, si las herramientas se encuentran disponibles o fueron construidas para abordar el proyecto en particular.

*f*) **Protocolo**: Este criterio pretende identificar los protocolos utilizados para comunicación dentro de las propuestas. Esta información es de especial interés ya que nos

permitirá conocer cómo los servicios en una arquitectura de microservicios, que se utiliza para soportar la adaptación del sistema, se comunican entre sí. Estos pueden ser, entre otros:

- **REST/HTTP**: REST es un estilo arquitectural que define las reglas para construcción de servicios Web mediante el uso de los métodos definidos por el protocolo HTTP.
- **MQTT**: Protocolo de red de publicación-suscripción que transporta mensajes entre dispositivos.
- **SOAP**: Especificación de protocolo de mensajería para intercambiar información estructurada en la implementación de servicios Web.

*g*) **Modelo de Proceso**: Con este criterio, se pretende identificar cuál es el modelo de desarrollo utilizado. Por ejemplo: DevOps, Desarrollo Ágil o Desarrollo Orientado por Modelos.

*h*) **Métodos de evaluación**: Tipo de estrategia o método de evaluación que se ha utilizado para evaluar cada aproximación de adaptación propuesta (método, modelo, herramienta, etc.). Estos métodos proporcionan evidencia sobre la utilidad de la propuesta, y pueden ser de tres tipos:

- **Experimento**: Investigación formal y rigurosa basada en una hipótesis comprobable con variables de control establecidas.
- **Encuesta**: Un método empírico que se utiliza para recopilar información de o sobre las personas para describir, comparar o explicar su conocimiento, sus actitudes o su comportamiento.
- **Caso de estudio**: Una estrategia para investigaciones exploratorias detalladas que intentan comprender y explicar un fenómeno o construir / probar teorías, utilizando una combinación de análisis cualitativo y cuantitativo. Se considera un caso de estudio si se plantea más de una pregunta de investigación o se evalúa empíricamente un concepto teórico con un objetivo claro [64].

Por otra parte, si la propuesta carece de validación ésta se clasifica como "No validada". Entre las propuestas no validadas, se analiza si al menos proporcionan una prueba de concepto que ayuden a entender cómo se utiliza la propuesta mediante un ejemplo de aplicación.

5. **Dimensión Por qué - *Why***: Esta dimensión abarca los criterios que permiten identificar la razón por la que se incorporan elementos de adaptación al sistema de software. Estos criterios son:

*a*) **Propósito de adaptación**: este criterio corresponde con la motivación de la adaptación de adaptación en el sistema [31]. Usualmente, un sistema genera cambios con los siguientes propósitos:

- **Correctivo**: Modificación reactiva tras el despliegue para corregir problemas identificados.
- **Perfectivo**: Mejora sobre el sistema para detectar y corregir posibles fallas.
- **Adaptativo**: Cambios tras el despliegue con el fin de asegurar o mejorar la calidad del sistema durante cambios en el entorno.
- **Preventivo**: Cambios sobre el sistema, orientados a detectar y corregir incidentes.

*b*) **Necesidades de adaptación :** El propósito de este criterio es identificar cuáles son los requerimientos que motivan la adaptación. Cada propuesta de adaptación ha sido clasificada de acuerdo con una o más de las siguientes necesidades:

- ***SLA (Service Level Agreement)* - Acuerdos de nivel de servicio**: Cumplimiento de los acuerdos de nivel de servicio pactados con el usuario final o cliente.
- ***QoE (Quality of Experience)* - Calidad de Experiencia**: Mejora en la experiencia de usuario.
- ***QoS (Quality of Service)*- Calidad del Servicio**: Mantener o mejorar la calidad de servicio.
- ***FT (Fault Tolerance)* - Tolerancia a fallos**: Detección y recuperación ante fallos en tiempo de ejecución para evitar caídas.
- ***FA (Fault Avoidance)* - Evasión de fallos**: Disminuir el número de incidentes en el sistema evitando la incorporación de errores al mismo en las fases de diseño y codificación.
- ***FD (Fault Detection)*- Detección y corrección de fallos**: Eliminación de fallos previos al despliegue mediante actividades de verificación y validación, principalmente del código.
- ***NR(New requeriments)***: Incorporación de nuevos requerimientos funcionales o no funcionales.
- **Optimización de recursos**: Optimizar el uso de recursos físicos en conformidad con las necesidades del sistema.
- **Otro**: Necesidades de adaptación distintas a las que se ha mencionado anteriormente.

#### 3.1.3.5. Estrategia de síntesis de datos

El objetivo de esta actividad es recolectar y resumir los datos extraídos de los estudios primarios incluidos en la revisión sistemática. Consiste en la tabulación de las características de los estudios y sus resultados (síntesis cualitativa) y el uso de métodos estadísticos en aquellos casos que sea apropiado (síntesis cuantitativa o meta-análisis). En este trabajo, se emplean dos métodos cualitativos para sintetizar los datos extraídos de los estudios primarios: síntesis narrativa y análisis temático. La síntesis narrativa es un marco general de descripciones seleccionadas y ordenamiento de evidencia primaria basado en texto. La principal ventaja de este método es que ofrece una gran flexibilidad y es capaz de hacer frente a diversos tipos de evidencia [29]. Se utiliza la herramienta *Mendeley* para almacenar todos los documentos y anotar las descripciones que soportan la clasificación de los estudios en conformidad con los criterios descritos en la sección 3.3. El análisis temático permite clasificar los estudios primarios y resumir los hallazgos identificando a través de una codificación temas principales o recurrentes dentro de los estudios. La combinación de estos dos métodos nos permite dar respuesta a las preguntas de investigación. Para lograrlo, se llevan a cabo los siguientes pasos:

1. Lectura de los estudios e identificación de los segmentos de texto relevantes: Durante la lectura de los estudios primarios, con base en las preguntas de investigación planteadas en 3.1.2 y la estrategia de extracción de datos descrita en la sección 3.1.5, se identifican fragmentos de texto y conceptos que aportan datos relevantes para la investigación a través del mecanismo conocido como (*keywording*).

2. Generación de códigos: Los datos recopilados son señalados con códigos, en conformidad con los pasos sugeridos por Cruzes et al. en [16], para facilitar su posterior clasificación, usando el enfoque integrado, que emplea tanto el desarrollo inductivo de los códigos como un marco de organización deductivo para los tipos de códigos. De acuerdo con [16], "Los códigos son etiquetas para asignar unidades de significado a la información descriptiva o inferencial compiladas", por esto, los códigos son traducidos en temas coherentes y distintivos que nos proporcionan la clasificación final, orientada a la construcción de la taxonomía presentada en la sección 3.4.

3. Búsqueda de temas: Para cada elemento, se combinan diferentes códigos iniciales generados a partir de segundo paso hacia posibles temas.

4. Analizar los códigos para reducir solapamientos y definir temas: Algunos temas son definidos como resultado de las preguntas de investigación, mientras que otros aparecen como resultado de leer los estudios primarios.

5. Revisión y refinamiento de temas: Este paso nos permite definir temas claros y concisos.

Este método de síntesis es eficaz en lo que respecta a proporcionar un mapa y una visión general rápida de un tema de investigación particular.

## 3.2. Ejecución

En esta etapa se describen las actividades del proceso de ejecución de la revisión sistemática. Inicia con la aplicación de la estrategia de búsqueda para obtener las publicaciones relevantes. Sobre los resultados se aplican los criterios de inclusión y exclusión presentados en 3.1.3.2 para seleccionar los estudios primarios, el detalle de esta actividad se presenta en la subsección 3.2.1. Sobre los estudios primarios se aplica una evaluación que permite determinar el nivel de calidad de los artículos con relación al cumplimiento del objetivo de la investigación, los criterios aplicados y resultados de dicha evaluación son presentados en la subsección 3.2.2. En la subsección 3.2.3 se resumen los resultados de la aplicación del proceso de extracción de datos de los estudios seleccionados apoyado en la estrategia detallada en 3.1.3.4 y el mecanismo de síntesis de datos propuesto en 3.1.3.5.

La ejecución de la revisión sistemática se llevó a cabo durante el mes de septiembre de 2019. El paso inicial consistió en la aplicación de las cadenas de búsqueda en las librerías de acuerdo con lo detallado en la tabla 3.4. Posteriormente, fueron eliminados los estudios que no se encontraban escritos en inglés, no fueron publicados en conferencias o revistas, así como artículos cortos o duplicados, aplicando los criterios de exclusión CE6 a CE10; algunos de ellos correspondían con filtros proporcionados por las librerías, en otros casos, fue necesario realizar la exclusión de los documentos manualmente. En total, fueron excluidos 55.696 artículos. IEEE Xplore y Science Direct permiten también filtrar el dominio de aplicación, mientras que en Springer Link fue posible excluir los artículos cuyos títulos incluían términos relacionados con el uso de contenedores en puertos y barcos, aplicando así el criterio CE5.

Adicionalmente, se aplicaron en los filtros detallados en la tabla 3.4 y se personalizó la búsqueda para cada librería digital, para finalmente obtener 5476 artículos.

### 3.2.1. Selección de Estudios Primarios

Sobre los estudios resultantes se realizó una revisión de los títulos para excluir aquellos que claramente no pertenecieran al dominio de software o computación en la nube, como resultado de esta depuración permanecieron 2.264 artículos. De estos, 1.992 estudios fueron descartados debido a que, aunque correspondían a estudios en el marco de la Ingeniería de Software, el contexto de investigación difería del determinado para esta revisión.

En el paso final de la selección, para los 272 artículos restantes fue consultado el *abstract* con el fin de validar el objetivo del estudio, tras esta revisión 184 de los 272 estudios fueron

Tabla 3.4: Detalle búsqueda por librería

| Base de datos | Filtros aplicados | Artículos |
|---|---|---|
| IEEE Xplore | Solo artículos presentados en conferencias, acceso temprano y revistas.<br><br>Se omitieron los artículos cuyas palabras clave eran: ”tanks (containers)” (76), ”nonlinear control systems”(37), ”level control”(34) ”sea ports”(27), ”process control”(24), “optimisation”(21), ”scheduling”(20), ”three-term control”(19), “cranes”(16), ”fuzzy control”(16), “control engineering computing”(16), “Internet”(24).<br><br>Se mantuvieron aquellos que tuvieran los términos: “cloud computing (87)”, “adaptive control (52)”, “resource allocation (42)”, ”virtualisation (30)”, “Internet of Things (27)”, “learning (artificial intelligence) (23)”, “service-oriented architecture (23)”, “virtual machines (23)”, “middleware (20)”, “software architecture (19)”, “parallel processing (18)”, “mobile computing (17)”.<br><br>Se excluyeron manualmente los artículos relacionados, con la optimización o del uso de contenedores en grúas, puertos y barcos, tanques de agua, PI y PID Contollers. | 219 |
| ACM DL | Solo artículos presentados en conferencias y revistas.<br>Se eliminaron los artículos que solo tenían disponible el abstract. | 30 |
| Springer Link | Solo artículos presentados en conferencias y revistas. Se aplicó el filtro No preview-only. Fueron omitidos los artículos que contenían en su título las palabras ”Maritime”, ”Martimim”, ”ship”, ”shipping”, ”transport”, ”water”, ”tank”, çranes”, ”Stowage”, ”supply chain” | 5200 |
| Science Direct | Solo artículos de investigación y revisión.<br>La búsqueda se realizó solo sobre los datos en el título, abstract y palabras clave.<br><br>Wildcars o más de 8 conectores booleanos no son soportados por lo que se utilizó la cadena: (ADAPTATION) AND ((MICROSERVICE OR MICRO-SERVICE OR ”MICRO SERVICE.ºR MICROSERVICES) OR (CONTAINER OR KUBERNET OR DOCKER)).<br><br>Fueron excluidos los artículos publicados en las siguientes revistas:<br>1. European Journal of Operational Research<br>2. Transportation Research Part E: Logistics and Transportation Review<br>3. Journal of Transport Geography<br>4. Ecological Engineering<br>5. Science of The Total Environment<br><br>Se excluyeron manualmente los artículos que se relacionaban con optimización del uso de contenedores en almacenamiento, transporte, grúas y barcos | 27 |

excluidos, 5 estudios fueron confirmados y para los 83 restantes fue necesario consultar la introducción, conclusiones y en algunos casos hacer una lectura del estudio completo para descartar o confirmar su pertinencia con respecto a la revisión. Al finalizar el proceso de lectura, 73 artículos en total fueron excluidos. Solo se descartaron estudios una vez era claro que no se encontraban en el alcance de la presente investigación.

Tras este proceso, detallado en la figura 3.2, fueron seleccionados los 13 estudios primarios listados en la tabla 3.5. Estos estudios también se pueden encontrar en el Apéndice A.

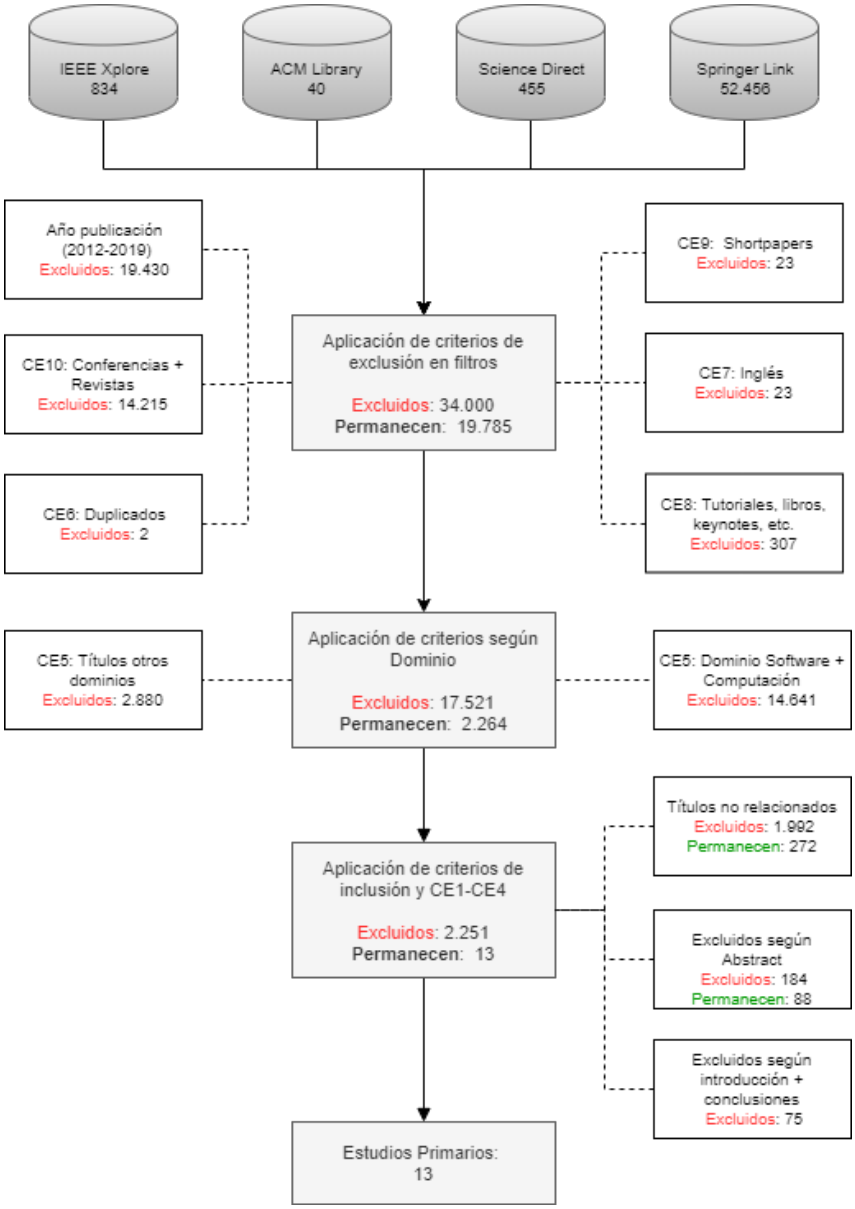


Figura 3.2: Proceso de selección de estudios primarios

### 3.2.2. Evaluación de calidad

Posteriormente se realizó la evaluación de los artículos aplicando los criterios de calidad establecidos en la sección 3.1.3.3. Como resultado de este proceso, se obtuvieron las calificacio-

Tabla 3.5: Estudios primarios

| ID | TITULO | AÑO | REPOSITORIO | AUTORES |
|---|---|---|---|---|
| S01 | A Microservices Architecture for Reactive and Proactive FaultTolerance in IoT Systems | 2018 | IEEE Xplore | Alexander Power, Gerald Kotonya |
| S02 | Adaptive sensing using internet-of-things with constrained communications | 2017 | ACM Digital Library | Mahmudur Rahman, Hua-Jun Hong, Amatur Rahman, Pei-Hsuan Tsai, Afia Afrin, Md Yusuf Sarwar U., Nalini V., Cheng-Hsin Hsu |
| S03 | ConMon: An automated containerbased network performancemonitoring system | 2017 | IEEE Xplore | Farnaz Moradi, Christofer Flinta, Andreas Johnsson, Catalin Meirosu |
| S04 | D3L-Based Service Runtime Self-Adaptation Using Replanning | 2018 | IEEE Xplore | Xianghui Wang; Zhiyong Feng; Keman Huang |
| S05 | Gru: An Approach to Introduce Decentralized Autonomic Behavior in Microservices Architectures | 2016 | IEEE Xplore | Luca Florio, Elisabetta Di Nitto |
| S06 | Run-Time Reliability Estimation of Microservice Architectures | 2018 | IEEE Xplore | Roberto Pietrantuono, Stefano Russo, Antonio Guerriero |
| S07 | Synchronous Reconfiguration of Distributed Embedded Applications During Operation | 2019 | IEEE Xplore | Kilian Telschig, Alexander Knapp |
| S08 | A Common Service Middleware for Intelligent Complex Software System | 2019 | IEEE Xplore | Dejun Ning; Yu Wang; Jiacheng Guo; Xuanming Zhang |
| S09 | Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity | 2017 | IEEE Xplore | Sara Hassan; Nour Ali; Rami Bahsoon |
| S10 | Self-managing cloud-native applications: Design, implementation, and experience | 2017 | Science Direct | Giovanni Toffetti, Sandro Brunner, Martin Blöchlinger, Josef Spillner, Thomas Michael B. |
| S11 | Improving microservice-based applications with runtime placement adaptation | 2019 | Springer Link | Adalberto R. Sampaio Jr., Julia Rubin, Ivan Beschastnikh, Nelson S. Rosa |
| S12 | Towards Integrating Microservices with Adaptable Enterprise Architecture | 2016 | IEEE Xplore | Justus Bogner ; Alfred Zimmermann |
| S13 | Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap | 2016 | IEEE Xplore | Sara Hassan; Rami Bahsoon |

nes detalladas en la tabla 3.6. De acuerdo con los umbrales definidos, el resultado del proceso deriva en la exclusión de los estudios[S12] *"Towards Integrating Microservices with Adaptable Enterprise Architecture"* y S13 *"Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap"*, debido principalmente a que ninguno de los dos presenta una evaluación del enfoque propuesto.

Tabla 3.6: Calificación de estudios según criterios de calidad

| AI/ESTUDIO | S01 | S02 | S03 | S04 | S05 | S06 | S07 | S08 | S09 | S10 | S11 | S12 | S13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **SAI1** | 1.0 | 1.0 | 0.5 | 1.0 | 0.5 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.5 | 0.5 |
| **SAI2** | 1.0 | 0.5 | 1.0 | 0.5 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 |
| **SAI3** | 1.0 | 1.0 | 0.5 | 1.0 | 1.0 | 1.0 | 1.0 | 0.5 | 0.5 | 1.0 | 1.0 | 0.5 | 0.5 |
| **SAI4** | 0.5 | 1.0 | 1.0 | 1.0 | 0.5 | 1.0 | 1.0 | 0.5 | 0.5 | 1.0 | 1.0 | 0.0 | 0.0 |
| **GAI1** | 1.0 | 0.5 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.5 | 1.0 | 1.0 | 1.0 | 0.5 | 0.5 |
| **GAI2** | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 0.0 |
| **GAI3** | 0.5 | 0.5 | 1.0 | 1.0 | 0.5 | 0.5 | 1.0 | 0.0 | 0.5 | 0.5 | 1.0 | 0.0 | 0.0 |
| **Calificación** | 3.46 | 3.29 | 3.25 | 3.63 | 3.08 | 2.33 | 4.00 | 2.75 | 3.08 | 3.83 | 4.00 | 1.67 | 0.92 |

### 3.2.3. Extracción y síntesis de datos

La extracción de datos se realizó entre octubre de 2019 y febrero de 2020. El proceso se llevó a cabo según la estrategia de extracción y síntesis de datos descrita en las secciones 3.1.3.4 y 3.1.3.5, respectivamente. A continuación, se presentan los resultados obtenidos en cada uno de los pasos ejecutados:

1. Lectura de los estudios e identificación de los segmentos de texto relevantes: En cada uno de los estudios primarios fueron identificados, señalados y resaltados los fragmentos de texto que dan respuesta, de manera parcial o completa, a las preguntas de investigación y aquellos conceptos que tienen relevancia en el proceso de adaptación.

2. Generación de códigos: Se decidió utilizar la siguiente nomenclatura para asignación de los códigos a los criterios de clasificación:

   **ECn.m** (3.2)

   Siendo:

   EC : Las siglas en inglés para Criterio de Extracción (*Extraction Criteria*).

   n : El identificador numérico de la clase o tema que agrupa los criterios.

   m : El identificador numérico del criterio.

   De esta manera, el EC1.1 corresponderá con el criterio de extracción número 1 relacionado al tema 1.

3. Búsqueda de temas: Para dar continuidad a la línea con la que los criterios de extracción fueron trazados, los temas se definen en conformidad con las preguntas de investigación planteadas en la sección 3.1.2. Con esto en mente, los criterios previamente definidos y los que se identificaron durante la lectura de los estudios se agrupan en los siguientes temas:

   1 **Aproximaciones**: Abarca los criterios que podrían dar respuesta a la pregunta de investigación PI1, relacionada con la identificación de los tipos de aproximación propuestos.

   2 **Tipo de adaptación**: Consolida los criterios que caracterizan el tipo de adaptación implementado o propuesto en el estudio y que son abordados en la pregunta de investigación PI2.

   3 **Motivadores de adaptación**: Con la pregunta de investigación PI3, se pretende identificar las necesidades en el sistema que impulsaron la adopción de mecanismos de

adaptación. Este tema agrupa los criterios orientados a dar respuesta a dicha pregunta.

4 **Técnicas de adaptación**: En este tema se recopilan los aspectos técnicos utilizados en los estudios para conseguir la adaptación del sistema.

5 **Herramientas**: Consolida las herramientas tecnológicas utilizadas en cada uno de los estudios para soportar la adaptación.

6 **Modelo de proceso de desarrollo**: Este tema recopila los diferentes modelos de proceso de desarrollo adoptados en los estudios. En conjunto con los temas [4] y [5], es abordado en la pregunta PI4.

7 **Nivel de madurez**: Con el ánimo de dar respuesta a la pregunta de investigación PI5, se han definido criterios que nos permitan identificar los tipos de evaluación empleados en el desarrollo de los proyectos. Este tema agrupa dichos criterios.

4. Analizar los códigos para reducir solapamientos, definición, revisión y refinamiento de temas: Una vez establecidos los temas listados en el paso anterior, se generó el esquema de clasificación presentado en la subsección 3.4.1. Este último ítem consolida los dos pasos finales (4 y 5) propuestos en la sección 3.1.3.5.

## 3.3. Análisis Bibliométrico

En esta sección se presentan los resultados obtenidos tras la revisión sistemática con el ánimo de dar respuesta a las preguntas de investigación planteadas.

Como se menciona en la sección 3.1.3, la búsqueda fue ejecutada entre 2012 y 2019, sin embargo, los estudios seleccionados como primarios se encuentran entre 2016 y 2019 como se muestra en la figura 3.3. Los artículos encontrados correspondientes a años anteriores que incorporaban el concepto de contenedores lo hacían en su mayoría para resolver problemas de escalabilidad (CE2) y aquellos que proponían arquitecturas con base en micro-servicios principalmente se publican desde 2017. Estos resultados muestran que estos temas se han empezado a trabajar recientemente, de ahí que no exista todavía un cuerpo de conocimiento amplio, pero se encuentra en auge.

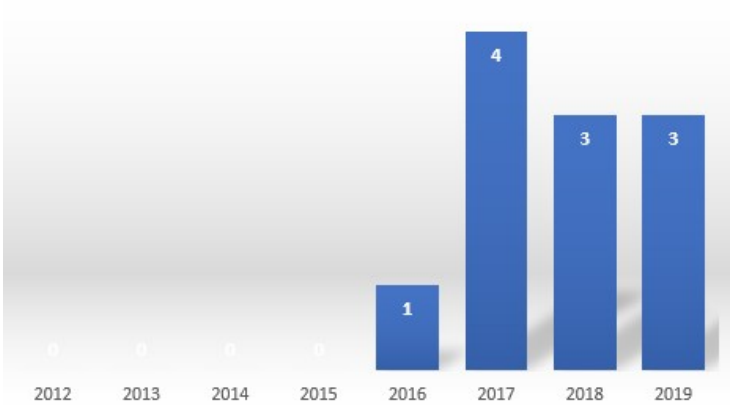


Figura 3.3: Estudios primarios por año

El objetivo de la revisión realizada se orienta principalmente a propuestas que involucren mecanismos de adaptación mediante la incorporación de contenedores, micro-servicios o los dos. Dentro de los estudios primarios obtenidos, el 55 %, es decir, 6 de los estudios integraban arquitecturas con base en micro-servicios, el 27 % hacen uso de tecnologías basadas en contenedores y tan solo el 18 % utilizan los dos conceptos para la implementación de estrategias de adaptación, tal como se observa en la figura 3.4.

Los estudios primarios fueron obtenidos en las librerías que se muestran en la figura 3.5 y se encuentran publicados en las revistas o conferencias listadas en la tabla 3.7.

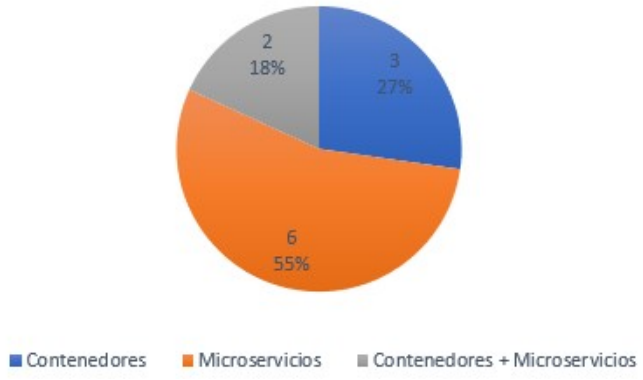


Figura 3.4: Estudios primarios por tema

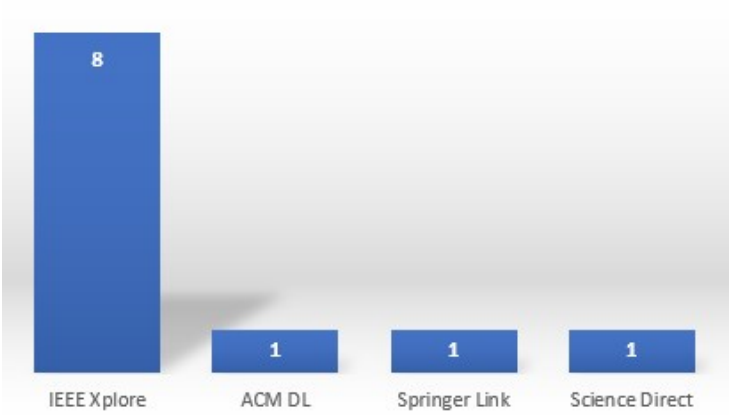


Figura 3.5: Estudios primarios según librería

Tabla 3.7: Medios de publicación y tipo de propuesta

| ESTUDIO | PUBLICADO EN | TIPO | PROPUESTA |
|---|---|---|---|
| [S01]A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems | 2018 IEEE 19th International Symposium on .^ World of Wireless, Mobile and Multimedia Networks"(WoWMoM) | Conferencia | Solución |
| [S02]Adaptive sensing using internet-of-things with constrained communications | ARM '17: Proceedings of the 16th Workshop on Adaptive and Reflective Middleware | Conferencia | Solución |
| [S03]ConMon: An automated container based network performance monitoring system | 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM) | Conferencia | Solución |
| [S04]D3L-Based Service Runtime Self-Adaptation Using Replanning | IEEE Access | Revista | Solución |
| [S05]Gru: An Approach to Introduce Decentralized Autonomic Behavior in MicroservicesArchitectures | 2016 IEEE International Conference on Autonomic Computing (ICAC) | Conferencia | Solución |
| [S06]Run-Time Reliability Estimation of Microservice Architectures | 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE) | Conferencia | Solución |
| [S07]Synchronous Reconfiguration of Distributed Embedded Applications During Operation | 2019 IEEE International Conference on Software Architecture (ICSA) | Conferencia | Solución |
| [S08]A Common Service Middleware for Intelligent Complex Software System | 2019 Prognostics and System Health Management Conference (PHM-Paris) | Conferencia | Solución |
| [S09]Microservice Ambients: An Architectural Meta-Modelling Approach for MicroserviceGranularity | 2017 IEEE International Conference on Software Architecture (ICSA) | Conferencia | Solución |
| [S10]Self-managing cloud-native applications: Design, implementation, and experience | Future Generation Computer Systems, Volume 72, July 2017, Pages 165-179 | Revista | Solución |
| [S11]Improving microservice-based applications with runtime placement adaptation | Journal of Internet Services and Applications | Revista | Solución |

## 3.4. Taxonomía

Uno de los propósitos de la revisión sistemática presentada en este capítulo, consiste en la construcción de una taxonomía de criterios de adaptabilidad a partir de los estudios primarios seleccionados. Esta sección presenta una vista del cuerpo de conocimiento proporcionado por los estudios primarios en relación con el uso de micro-servicios y contenedores para la adopción de mecanismos de adaptación en sistemas de software. Se encuentra estructurada con base en el esquema de clasificación presentado en 3.4.1 en conformidad con su incidencia en la respuesta a las preguntas de investigación planteadas en 3.1.2.

### 3.4.1. Esquema de clasificación

El esquema de clasificación aquí presentado es el resultado del proceso descrito en la sección 3.2.3. Luego de haber realizado la extracción de datos de los estudios primarios y la selección de los fragmentos de texto relevantes para la investigación, con base en los ya definidos conceptos base presentados en la sección 3.1.3.4, se realiza la codificación de los criterios relevantes para facilitar su posterior clasificación. Los temas resultantes para la clasificación, ordenados de acuerdo con la pregunta de investigación a la que dan respuesta, son listados en la tabla 3.8:

Tabla 3.8: Esquema de clasificación

| **P.I** | **CLASIFICACIÓN** | | **CRITERIO** | |
|---|---|---|---|---|
| | **CÓDIGO** | **DESCRIPCIÓN** | **CÓDIGO** | **DESCRIPCIÓN** |
| PI1 | EC1 | Aproximaciones | EC1.1 | Tipo de propuesta |
| | | | EC1.2 | Dominio de aplicación |
| | | | EC1.3 | Contexto de uso |
| PI2 | EC2 | Tipo de adaptación | EC2.1 | Fase de adaptación |
| | | | EC2.2 | Nivel de autonomía |
| | | | EC2.3 | Tiempo de adaptación |
| | | | EC2.4 | Mecanismo de adaptación |
| | | | EC2.5 | Actividad de adaptación |
| | | | EC2.6 | Condiciones de adaptación |
| PI3 | EC3 | Motivadores de adaptación | EC3.1 | Propósito de adaptación |
| | | | EC3.2 | Necesidades de adaptación |

**Tabla 3.8 continúa de la página anterior**

| P.I | CLASIFICACIÓN | | CRITERIO | |
|---|---|---|---|---|
| | CÓDIGO | DESCRIPCIÓN | CÓDIGO | DESCRIPCIÓN |
| PI4 | EC4 | Técnicas de adaptación | EC4.1 | Frecuencia de adaptación |
| | | | EC4.2 | Lógica de adaptación |
| | EC5 | Herramientas | EC5.1 | Herramientas |
| | | | EC5.2 | Protocolo comunicación |
| | EC6 | Modelo de Proceso de Desarrollo | EC6,1 | Modelo de proceso |
| PI5 | EC7 | Nivel de madurez | EC7.1 | Métodos de evaluación |

### 3.4.2. *Framework* de clasificación taxonómica

En la presente subsección se describen los conceptos obtenidos para cada uno de los criterios, con base en la contribución de cada uno al desarrollo de los estudios primarios y organizados según la pregunta de investigación a la que da respuesta y bajo la estructura definida en 3.4.1. El resultado de este ejercicio, presentado en la tabla 3.20, constituye un *framework* que permite mapear de manera ágil los criterios y elementos que componen la taxonomía propuesta.

#### 3.4.2.1. PI 1: Tipos de Aproximación

Esta pregunta se plantea con el ánimo de caracterizar las propuestas realizadas en materia de adaptación. Tras la extracción de datos con base en el criterio EC1.1, se evidencia que el 45 % de los estudios, 5 en total, plantean nuevos diseños o extensiones a diseños previos de arquitecturas o enfoques (aproximaciones) arquitectónicos, dos de ellos orientados a micro-servicios, uno a contenedores y dos más que involucran los dos temas. Entre los estudios restantes se encuentran dos propuestas de marcos de trabajo ambas enfocadas al uso de micro-servicios, dos nuevas herramientas de software que se implementan usando contenedores para facilitar su adaptación, un método de pruebas para aplicaciones basadas en micro-servicios y un método para análisis y diseño de aplicaciones con arquitectura de micro-servicios cuyo despliegue se realiza en contenedores. Esto se puede apreciar en la figura 3.6 y un poco más detallado en la tabla 3.9.

De acuerdo con los resultados obtenidos tras la evaluación del aspecto EC1.2 presentados en la tabla 3.10, las aproximaciones planteadas tanto en el ámbito de contenedores como micro-servicios apoyan principalmente sistemas en entornos *Cloud*, sistemas distribuidos y de internet de las cosas (*IoT*), dos de las propuestas abordan dominios de aplicación menos comu-

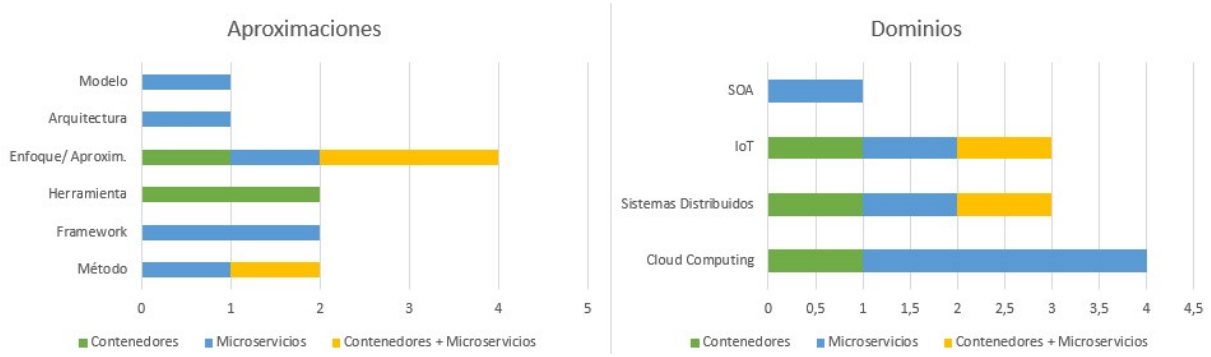


Figura 3.6: Aproximaciones/Dominios de Aplicación

Tabla 3.9: PI1: EC1.1: Tipo de propuesta

| OPCIONES | ESTUDIOS | # ESTUDIOS | %ESTUDIOS |
|---|---|---|---|
| Método | [S06][S08] | 2 | 18 % |
| Framework | [S01][S04] | 2 | 18 % |
| Herramienta | [S02][S03] | 2 | 18 % |
| Aproximación | [S05][S07][S08][S09] | 4 | 36 % |
| Arquitectura | [S10] | 1 | 9 % |
| Modelo | [S11] | 1 | 9 % |

nes, tales como *Smart Cities* [S02] o *intelligent complex software systems* (i-CSS) [S08]. Si bien en [S04] no se menciona explícitamente el término SOA (*Service-Oriented Architecture*), los autores enfocan su propuesta a aplicaciones con arquitectura basada en servicios. Lo anterior encuentra sentido en uno de los principios que guían la arquitectura basada en micro-servicios: la componetización de los sistemas mediante el uso de servicios [40]. Todos los dominios de aplicación identificados facilitan la incorporación del dominio previamente mencionado debido a su capacidad de establecer mecanismos ágiles de comunicación entre los diferentes componentes del sistema.

10 de los estudios, según lo estipulado para el criterio EC1.3, se desarrollan en el contexto académico, sin embargo, el artículo llamado "*A Common Service Middleware for Intelligent*

Tabla 3.10: PI2: Relación Propuesta Vs. Dominio

| | EC1.1 Propuesta | | | | | |
|---|---|---|---|---|---|---|
| **EC1.2 Dominio** | Método | Framework | Herramienta | Aproximación | Arquitectura | Modelo |
| Cloud Computing | [S06] | | [S03] | | [S10] | [S11] |
| Sistemas Distribuidos | | | | [S05][S07][S09] | | |
| IoT | | [S01] | [S02] | | | |
| SOA | | [S04] | | | | |
| i-CSS | [S08] | | | [S08] | | |

*Complex Software System*" [S08] realiza la verificación de su propuesta en cooperación con una compañía de tecnología que se enfoca en el desarrollo de sistemas i-CSS, usando como referencia para el diseño y desarrollo de la prueba de concepto proyectos de la empresa [45].

#### 3.4.2.2. PI 2: Tipos de Adaptación

Para la identificación de las clases de adaptación propuestas o implementadas se plantearon seis criterios de clasificación, expuestos en la tabla 3.8. El 81 % de los estudios proponen mecanismos de adaptación en tiempo de ejecución (EC2.3), es decir, el sistema se adapta de forma dinámica; el mismo número de estudios ejecuta los cambios con base en información entregada por monitores, es decir, su actividad es proactiva (EC2.5). Sin embargo, estas adaptaciones dinámicas se han contemplado previamente como parte de la lógica de adaptación implementada. Se echa en falta aproximaciones que tengan en cuenta nuevas circunstancias que surjan (en tiempo de ejecución), y que pongan de manifiesto la necesidad de poder aplicar nuevas opciones de adaptación o reconfiguración de los servicios que sean adicionales a la lógica de adaptación implementada. Particularmente el marco de trabajo detallado en "*A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems*" [S01] provee dos tipos de adaptación: Reactivo, cuando el sistema inicia la recuperación una vez se ha detectado el error y proactivo, cuando se inicia la estrategia de recuperación en una fase previa a la detección del error [51].

También se puede observar que 8 de los 9 estudios señalados en el párrafo anterior, presentan propuestas de procesos de adaptación completamente autónomos, cinco de estos estudios admiten en su propuesta la incorporación de funcionalidades en tiempo de ejecución, por lo que se consideran de estrategia abierta de adaptación (EC2.4).Tanto las propuestas con estrategias de adaptación abierta, como cerrada tienen condiciones de adaptación diversas (EC2.6), pueden ser observadas en la tabla 3.11.

Tabla 3.11: PI2: EC2.6: Condiciones de adaptación

| OPCIONES | ESTUDIOS | # ESTUDIOS | %ESTUDIOS |
|---|---|---|---|
| Auto-configuración | [S02][S03][S07][S09] | 4 | 36 % |
| Auto-optimización | [S05][S09][S11] | 3 | 27 % |
| Auto-curación | [S01][S04][S10] | 3 | 27 % |
| Auto-protección | [S01][S06] | 2 | 18 % |

Dentro del conjunto de propuestas con actividad proactiva, solo el estudio titulado *"Gru: An Approach to Introduce Decentralized Autonomic Behavior in MicroservicesArchitectures"* [S05] no es completamente autónomo, pues el flujo MAPE para adaptación que los agentes de software implementan, se ejecuta de manera periódica, según el intervalo de tiempo indicado por el

Tabla 3.12: PI2: EC2.2: Nivel de autonomía

| OPCIONES | ESTUDIOS | # ESTUDIOS | % ESTUDIOS |
|---|---|---|---|
| Autónomo | [S01][S02][S03][S04][S07][S09][S10][S11] | 8 | 72 % |
| Parcialmente autónomo | [S05][S06][S08] | 3 | 27 % |
| Humano en medio | | | |

usuario en la configuración de los agentes [25].

Por otra parte, el estudio "*Run-time Reliability Estimation of Microservice Architectures*" [S06] también se considera parcialmente autónomo debido a que el método se ejecuta tras una solicitud de evaluación de confiabilidad que puede proceder de un proceso programado o en el despliegue de un microservicio (EC2.2). Al no ser posible agregar nuevos comportamientos en tiempo de ejecución, se dice que su adaptación es cerrada (EC2.4). El método incluye actividades en tiempo de desarrollo con el objetivo de mapear el espacio demandado a una estructura matemática para obtener los casos de prueba y actividades en tiempo de ejecución para realizar la estimación de la confiabilidad en relación con la probabilidad de uso y la probabilidad de fallo de los micro-servicios a probar. Este conjunto provee al sistema la capacidad de prevención de fallos (*Self-protection* y de identificación y adopción de mejoras en su desempeño (*Self-optimization*) [50]. En la tabla 3.12 se puede observar el nivel de autonomía identificado en cada uno de los estudios primarios.

El listado de los estudios cuyo tiempo de adaptación corresponde al momento de ejecución o carga se puede apreciar en la tabla 3.13 de acuerdo con la actividad y mecanismo de adaptación empleados. La figura 3.7 proporciona una vista en árbol de los aspectos de adaptación relacionados con el tiempo que fueron identificados en los estudios primarios.

Tabla 3.13: PI2: EC2.3: Tiempo, condición, actividad y mecanismo de adaptación

| | | **EC2.3 Tiempo de Adaptación** | | | | |
|---|---|---|---|---|---|---|
| | | Carga | Ejecución | | | |
| | | **EC2.6 Condiciones** | | | | |
| **EC2.5 Actividad** | **EC2.4 Mecanismo** | Auto-Protección | Auto-Configuración | Auto-Optimización | Auto-Curación | Auto-Protección |
| Reactiva | Abierto | | | | | |
| | Cerrado | [S06] | | | [S01] | |
| Proactiva | Abierto | | [S03][S07] | [S11] | [S04][S10] | |
| | Cerrado | | [S02][S09] | [S05][S09] | | [S01] |

Si bien, el 81 % de los estudios la adaptación se realiza en la fase de operación del sistema (EC2.1), con excepción de [S06] en el cual la adaptación está presente en la capacidad del sistema de elegir el escenario de prueba a ejecutar y [S08] cuyo método de adaptación se aplica en la fase de diseño; algunos de los estudios realizan actividades de adaptación en fases alternas. Por

ejemplo, en [S03] la adaptación se realiza también en fase de monitorización, el sistema adapta el monitoreo para que sea ejecutado continuamente de manera precisa tanto en términos de ubicación como precisión métrica y así conseguir evaluaciones de desempeño oportunas [44]. En [S02] se despliegan funcionalidades en contenedores en conformidad con un disparador de un sensor, es decir, lleva a cabo la adaptación en fase de despliegue, pero los intervalos de envío de datos dependen de las características del entorno [52], durante la operación. En la tabla 3.14 se exponen las fases de adaptación utilizadas o impactadas en los estudios primarios.

Tabla 3.14: PI2: EC2.1: Fase de adaptación

| OPCIONES | ESTUDIOS | # ESTUDIOS | % ESTUDIOS |
|---|---|---|---|
| Operación | [S01][S02][S03][S04][S05][S07][S09][S10][S11] | 9 | 81 % |
| Despliegue | [S02][S03][S04][S07] | 4 | 36 % |
| Monitorización | [S01][S03][S06] | 3 | 27 % |
| Diseño | [S08] | 1 | 9 % |

El estudio [S08] llamado *"A Common Service Middleware for Intelligent Complex Software System"* provee a los arquitectos de software una solución sistemática para cumplir con los requisitos de adaptación en la fase de construcción de sistemas de software inteligentes de alta complejidad. Consta de un método de desarrollo basado en micro-servicios y contenedores, así como un método de análisis y diseño orientado a datos para lograr la evolución del sistema usando un bucle percepción-análisis-decisión-ejecución. También proporciona un servicio *middleware* basado en una arquitectura fractal *sea-cloud* y tecnología de inteligencia de datos compatible con los dos métodos mencionados [45]. El proceso de incorporación de nuevos requerimientos o cambios en el sistema se ejecuta en las fases de análisis y diseño (EC2.1), donde son evaluadas las capacidades del negocio para definir las micro aplicaciones y micro-servicios a incorporar o modificar. Dado que este ejercicio requiere la participación humana en la evaluación de los requerimientos para la adaptación, decimos que el sistema es parcialmente autónomo (EC2.2).

#### 3.4.2.3. PI 3: Motivos para la Adaptación

Esta pregunta tiene por objetivo identificar las necesidades de adaptación que motivan el desarrollo de las propuestas (EC3.1)(EC3.2). Se encontró que la mayoría de los estudios se centran en el cumplimiento de los niveles de calidad de servicio [S02][S05[S06][S09][S10] y [S11], es decir, la adaptación frente a los requerimientos no funcionales planteados para el sistema con el ánimo de mantenerlo operativo, incluso si los recursos físicos son limitados. Adicionalmente en [S10] el sistema tiene la capacidad de recuperarse tras un fallo y en [S06] de prevenirlo mediante la detección de los errores en tiempo de ejecución al ejecutar pruebas en la fase previa al despliegue del contenedor [50]. En [S01] la tolerancia ante fallos implica la validación constan-

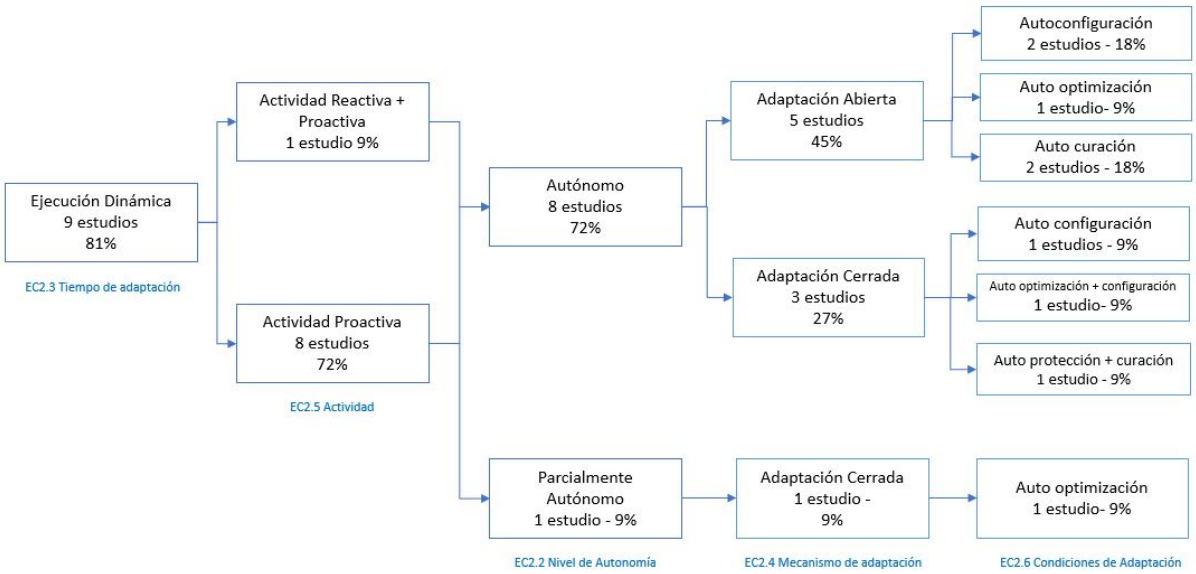


Figura 3.7: Tipos de adaptación en tiempo de ejecución

te de posibles errores en las siguientes categorías: replicación, tiempo, inversión, codificación, racionabilidad, controles estructurales y diagnósticos, mientras que en [S04] se reparan fallas en tiempo de ejecución para resolver conflictos semánticos con una estrategia de *re-planning*. [S07] y [S08] orientan sus estrategias de adaptación a la capacidad de incorporar nuevos requerimientos durante las fases de operación y diseño respectivamente, mientras que en [S03] la adaptación es motivada por la necesidad de optimizar los recursos en la capa de infraestructura asignados a los contenedores y en [S02] la optimización de los recursos de red para la lectura de datos de los monitores. Estas necesidades que motivan la adaptación son ilustradas en la tabla 3.15.

Tabla 3.15: PI3: EC3.2: Necesidades de adaptación

| OPCIONES | ESTUDIOS | # ESTUDIOS | % ESTUDIOS |
|---|---|---|---|
| Calidad del Servicio (QoS) | [S02][S05][S06][S09][S10][S11] | 6 | 54 % |
| Tolerancia a fallos (FT) | [S01][S04][S10] | 3 | 27 % |
| Nuevos Requerimientos (NR) | [S07][S08] | 2 | 18 % |
| Optimización de Recursos | [S03][S02] | 2 | 18 % |
| Detección de Fallos (FD) | [S06] | 1 | 9 % |

Cuando las necesidades están orientadas a la inclusión de nuevos requerimientos, la optimización de recursos de hardware y la calidad del servicio (*QoS*) el sistema tiene un propósito adaptativo [S02][S03][S05][S07][S08][S09][S11], es decir, mantener en operación el sistema mientras se realizan cambios en el sistema o el entorno. Por otra parte, cuando la necesidad va orientada a hacer el sistema resistente o resiliente ante fallos, el propósito se torna preventivo en [S06], perfectivo en [S04][S10] y en ocasiones tanto correctivo como preventivo [S01]. Lo

expuesto anteriormente, se puede observar en la tabla 3.16.

Tabla 3.16: Relación Propósitos Vs. Necesidades

| P/N | F.M. | QoS | N.R | O.R |
|---|---|---|---|---|
| Correctivo | 1 | | | |
| Perfectivo | 2 | 1 | | |
| Preventivo | 2 | 1 | | |
| Adaptativo | | 4 | 2 | 2 |

Donde:

- F.M.: Mitigación de fallos
- QoS: Calidad del servicio
- N.R.: Nuevos requerimientos
- O.R.: Optimización de recursos

#### 3.4.2.4. PI 4: Técnicas y Herramientas

Se pretende con esta pregunta, identificar los tipos de técnicas y herramientas utilizados para soportar la adaptación. Para dar respuesta a la misma, se determinaron tres criterios principales:

1. **Técnicas de adaptación**: La frecuencia de adaptación (EC4.1) en el 45 % de los estudios se realiza de manera continua [S01][S03][S04][S07][S09], en el 36 % se realiza de manera periódica [S05][S06][S10][S11], en [S05] un *loop* MAPE se activa según el intervalo de tiempo establecido por el usuario en la configuración de los agentes, en [S06] el tiempo de ejecución del proceso de adaptación depende de una la solicitud de evaluación de confiabilidad que se hace a los micro-servicios en la fase de liberación. En el caso de [S02] y [S08], la adaptación se realiza de manera arbitraria, en [S02] se incorporan cambios al sistema (despliegue de un nuevo contenedor) en conformidad con los parámetros recibidos de los sensores externos.

   Si bien la frecuencia de adaptación nos permite entender cuándo o en qué momento se realiza dicho proceso, la lógica de adaptación utilizada por el sistema dicta el cómo. Los estudios [S03[S05][S07][S09][S10] y [S11] emplean modelos heurísticos o conjuntos de políticas/reglas para determinar las acciones que debe tomar el sistema, por ejemplo, en [S10] el nodo encargado de *health management* compara la instancia en ejecución contra

el estado deseado del sistema y con base en el resultado puede terminar, reiniciar, eliminar componentes existentes , instanciar nuevos componentes o reconfigurar la conexión entre los componentes según dicten las reglas de adaptación, previamente definidas. Por otra parte, los estudios [S01] y [S06] incorporan métodos probabilísticos, [S02] y [S03] que adoptan mecanismos basados en lógica de predicados, mientras que [S08] usa un modelo maestro de datos de dominio unificado para que el sistema acumule experiencia y así modifique su estructura o comportamiento, por lo que decimos que es un método basado en una ontología.

En [S05] particularmente, se combina el uso de reglas y un modelo probabilístico, el planificador se encuentra basado en los conceptos de política y estrategia, las políticas son reglas que implican la ejecución de n actividades y las estrategias son algoritmos que le permiten al planificador escoger la mejor política a ejecutar con base en un peso otorgado a cada una. Finalmente se realiza una selección probabilística de un umbral y el reordenamiento de las políticas en conformidad con el peso determinado para cada una, con el fin de evitar que todos los agentes compartan el mismo comportamiento ejecutando siempre la política con el mayor peso.

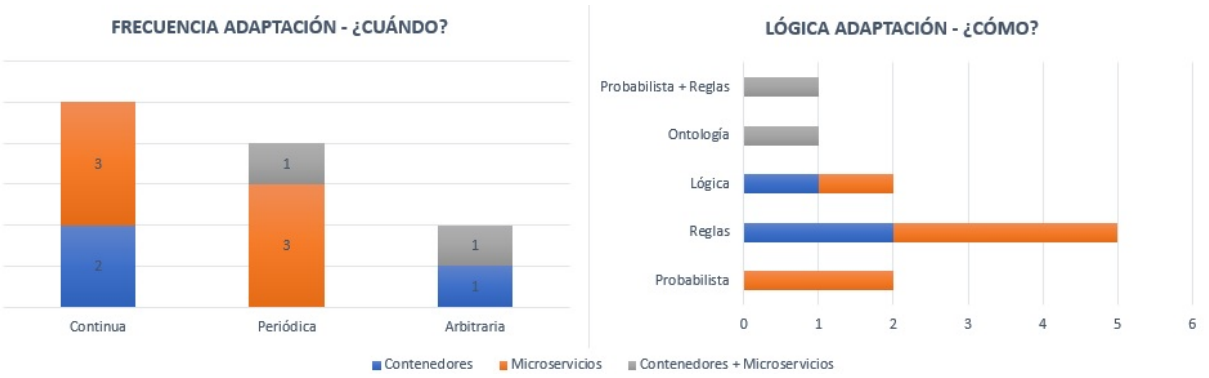


Figura 3.8: Técnicas de Adaptación

2. **Herramientas**: Dentro del conjunto de herramientas tecnológicas identificadas tras la extracción de datos, cuatro fueron construidas por los autores de los estudios primarios, dos de ellas como resultado final del estudio presentado y las dos restantes como herramientas de soporte a la adaptación en las fases de monitoreo y despliegue. Las demás herramientas utilizadas corresponden con tecnologías comerciales que se utilizaron en los casos de estudio o pruebas de concepto para evaluación de las propuestas.

   En la tabla 3.17 se encuentra el listado de herramientas utilizadas en las implementaciones expuestas por cada uno de los estudios y su descripción (EC5.1). Un alto porcentaje de las herramientas utilizadas se enfoca en el monitoreo del desempeño de los elementos del sistema (37 %), también se emplean herramientas en el despliegue automático de

las aplicaciones o componentes de software (21 %), *testing* (5 %) y comunicación entre las aplicaciones o servicios que la componen (11 %), también se utilizan herramientas orientadas a la gestión de las actividades de adaptación (5 %) y para almacenamiento de datos (21 %), como se muestra en la figura 3.9. A partir de los usos expuestos anteriormente se establecen seis subcriterios de clasificación de las herramientas, señalados a continuación:

- EC5.1.1 - Monitorización: En este subcriterio se incluyen las herramientas que fueron propuestas en los estudios o utilizadas en los casos de estudio para capturar información sobre el estado del sistema y su entorno. En esta categoría se encuentran: EnviroSCALE [S02], ConMon [S03], Zipkin [S11], Logstach [S10], Heapster [S11], Zuul [S06] y MetroFunnel [S06].
- EC5.1.2 - Despliegue: Es esta categoría contempla las herramientas empleadas por los estudios para el despliegue de servicios y componentes en contenedores o la orquestación de los contenedores, siendo Docker las más utilizada [S02][S03][S05][S08], seguida por Kubernetes [S02][S10][S11], Fleet [S10] y Dynamite [S10].
- EC5.1.3 - Pruebas: Esta categoría incorpora las herramientas para la ejecución automática de pruebas sobre los componentes del sistema. Dentro de los estudios solo se identificó una con este propósito: Travis, usada en [S11].
- EC5.1.4 - Almacenamiento: Corresponde a las herramientas que proveen capacidades de almacenamiento de datos al sistema propuesto o empleado para evaluación. En esta categoría se encuentran: Etcd [S10], Influxdb [S11], Elasticsearch [S10] y Eureka [S06].
- EC5.1.5 - Integración: En este subcriterio se listan las herramientas que facilitan la integración o comunicación entre componentes de una aplicación o de la aplicación con el entorno. Dentro de los estudios primarios fueron identificadas tres herramientas con ese objetivo: Eureka [S06], Open vSwitch [S03] y Axis2 [S04].
- EC5.1.6 - Gestión: En [S04] se utiliza la herramienta BPM Activiti para la generación y coordinación de flujos de trabajo que facilitaran la adaptación, por lo que se decide incorporarla en esta categoría.

A continuación, se realiza una descripción de las herramientas que fueron construidas exclusivamente para el estudio:

- EnviroSCALE(*Environmental Sensing and Community Alert Network* en [S02]: Es una plataforma de hardware que incorpora sensores y plataformas externas con una pila de software modular para recolectar y cargar datos a la nube. Corresponde con una

extensión de la plataforma de monitoreo basada en IoT, SCALE, creada en el contexto del *Smart America Challenge* y utilizada para aplicaciones de seguridad públicas. La plataforma está construida utilizando una Raspberry Pi para tareas de control y monitoreo, y un Arduino Nano para tareas de sondeo adicionales. Cuenta con sensores de gases, temperatura y humedad para determinar la calidad del aire. Usa un *broker* MQTT para cargar y publicar información, contenedores Docker para despliegue dinámico y *Kubernets* para la gestión de múltiples dispositivos IoT [52].

- ConMon en [S03]: Solución de software distribuida para observar métricas de desempeño de la red en entornos de ejecución de contenedores. La arquitectura tiene tres componentes principales, un controlador de monitoreo (MC), contenedores de monitoreo pasivo (PM) y contenedores de monitoreo activo (AM), los dos últimos pueden ejecutar diferentes funciones de monitoreo (MF). Los AM funcionan inyectando paquetes en la red a través de una sonda, posteriormente recibidos por una sonda diferente. Los PM reciben copias de los paquetes de la aplicación monitorizada en su interface, las MF de los PM capturan, filtran y evalúan los tiempos de dichos paquetes; también pueden identificar cambios en la capa de comunicación de la aplicación, en este último caso, dicha información es utilizada por MC para (re)configurar dinámicamente MFs activas o pasivas en reacción a los cambios. MC escucha continuamente los eventos generados por un *Container Management System* (por ejemplo, Docker), una vez identifica la creación de instancias de un contenedor de aplicaciones, MC solicita la instanciación de los contenedores de monitoreo necesario en conformidad con un flujo de trabajo preestablecido [44].
- MetroFunnel en [S06]: Herramienta personalizada para monitoreo de aplicaciones basadas en micro-servicios [50]. En este estudio, contrario a [S02] y [S03], la herramienta no es el propósito final de la propuesta planteada.
- Dynamite en [S10]: Motor de código abierto, construida en *Python*, para el escalado horizontal automático de aplicaciones basadas en contenedores y la instanciación de forma recursiva de composiciones de servicio [63]. La herramienta no constituye el objetivo final del estudio, pero es indispensable para ejecutar su estrategia de auto escalado.

De los estudios primarios fueron también extraídos los protocolos de mensajería utilizados para las implementaciones (EC5.2), encontrando que 8 de los estudios implementan el estilo arquitectural REST. Uno de los estudios utiliza un MQTT Broker para la publicación de datos y dos de los estudios no hacen mención del protocolo de comunicación empleado. En [S04], particularmente, se desarrollan dos adaptadores utilizados para in-

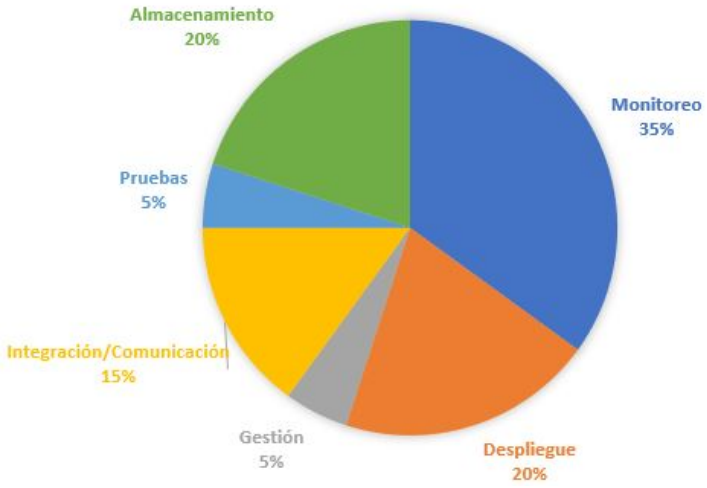


Figura 3.9: Herramientas según su uso

vocar servicios SOAP y RESTful, con el fin de soportar la comunicación con los dos tipos de servicio [65]. En la tabla 3.18 se muestra el número de estudios por protocolo de comunicación empleado, en donde se observa la clara tendencia por el uso de REST para la comunicación entre servicios debido a su flexibilidad y simplicidad.

3. **Modelos de proceso de desarrollo**: Tres de los estudios seleccionados mencionan a DevOps o la ingeniería de software continuo como el modelo de proceso seguido en el diseño o implementación de la propuesta (EC6.1): (i) [S03] para el desarrollo de la herramienta ConMon, como se mencionó en el ítem anterior, es un sistema basado en contenedores para automatizar el monitoreo continuo del desempeño de la red en un ambiente cloud [44], (ii) en [S11] se implementa un monitor que es capaz de identificar eventos DevOps recibidos de la herramienta *Travis* para integración continua cuando hay un nuevo despliegue. (iii) En [S08] en la construcción de CCOS, un servicio *middleware* para encapsular y compartir recursos de software a nivel de contenedores, el componente panel de control es responsable de la instalación de paquetes, DevOps y gestión de imágenes *Docker* [45]. En los demás estudios no se menciona la adopción de modelos de proceso de desarrollo para la implementación o construcción de las propuestas. Por ejemplo, en [S02] se propone e implementa una plataforma adaptativa de monitoreo y recopilación de datos ambientales en ciudades, sin embargo, no se aclara en el estudio el modelo o proceso de desarrollo empleado.

#### 3.4.2.5. PI 5: Nivel de madurez

La última pregunta de investigación planteada se orienta a la identificación de las evaluaciones realizadas sobre las propuestas y la rigurosidad con la que fueron ejecutadas (EC7.1). El 27 % de los estudios realizan experimentos para la validación de sus propuestas, como se

Tabla 3.17: Herramientas

| NOMBRE | ESTUDIO | DESCRIPCIÓN |
|---|---|---|
| Kubernetes | [S02][S10][S11] | Plataforma para despliegue, gestión y escalado de contenedores [4]. |
| Docker | [S02][S03][S05][S08] | Herramienta para automatizar el despliegue de aplicaciones en contenedores [19]. |
| EnviroSCALE | [S02] | Plataforma para monitoreo del ambiente y carga de datos recopilados en la nube [52]. |
| ConMon | [S03] | Solución de software distribuida para observar métricas de desempeño de la red en entornos de ejecución de contenedores [44]. |
| Open VSwitch | [S03] | Software utilizado como un switch virtual en entornos de servidores virtualizados [3]. |
| Axis2 | [S04] | Motor nuclear para servicios web [7]. |
| Activiti | [S04] | Motor de workflows en notación BPM 2.0 [2]. |
| Eureka | [S06] | Servicio REST de AWS que se comporta como un servidor para registrar y localizar micro-servicios existentes, informar su ubicación, estado y demás datos relevantes. [23]. |
| Zuul | [S06] | Aplicación de servicio perimetral para aplicar enrutamiento dinámico, monitoreo y seguridad a las solicitudes hechas hacía el backend de Netflix, principalmente [43]. |
| MetroFunnel | [S06] | Herramienta desarrollada a la medida por los autores para el monitoreo de aplicaciones basadas en microservicios [50]. |
| Etcd | [S10] | Almacén de key-values en sistemas distribuidos, usado como almacén de respaldo para kubernetes [22]. |
| Dynamite | [S10] | Aplicación Phyton de código abierto que orquesta la instanciación de componentes [63]. |
| Fleet | [S10] | Administrador de arranque de clúster distribuido [63] |
| Logstash | [S10] | Agregador de logs, encargado de organizar, ordenar y almacenar los eventos para posterior análisis [54] |
| Elasticsearch | [S10] | Herramienta para almacenar los eventos recopilados en el monitoreo [54] |
| Heapster | [S11] | Herramienta deprecada, previamente utilizada para monitoreo y análisis de desempeño de los contenedores [1]. |
| Zipkin | [S11] | Sistema de rastreo distribuido, recopila datos de tiempo necesarios para solucionar problemas de latencia en arquitecturas de servicio [68]. |
| Travis | [S11] | Automatización de testing [5]. |
| Influxdb | [S11] | *Data storage* para mantener el historial de ejecuciones de una aplicación |

muestra en la tabla 3.19. Estos estudios corresponden a: (i) [S06] en el cual a través de escenarios simulados se evalúa la eficiencia y precisión de la propuesta, (ii) [S10], sus experimentos tienen como objetivo demostrar que la arquitectura propuesta y la implementación de la misma abordan correctamente los requisitos identificados para las aplicaciones nativas de la nube: elasticidad y resiliencia, mediante la exploración a partir de dos preguntas de investigación y (iii) [S11], en el que se utilizó un modelo de red sin escala para representar micro-aplicaciones simuladas para validar la hipótesis subyacente de que los modelos de ley de potencia, como el modelo *Barabási-Albert*, son apropiados para modelar micro-aplicaciones.

Los estudios que utilizaron casos de estudio como mecanismo de evaluación conforman

Tabla 3.18: PI4: EC5.2: Protocolos

| OPCIONES | ESTUDIOS | # ESTUDIOS | % ESTUDIOS |
|---|---|---|---|
| REST/HTTP | [S01][S03][S04][S05][S06][S08][S10][S11] | 8 | 72 % |
| SOAP | [S04] | 1 | 9 % |
| MQTT | [S02] | 1 | 9 % |
| No Especificado | [S07][S09] | 2 | 18 % |

Tabla 3.19: PI5: EC7.1: Método de evaluación

| OPCIONES | ESTUDIOS | # ESTUDIOS | % ESTUDIOS |
|---|---|---|---|
| Experimento | [S06][S10][S11] | 3 | 27 % |
| Caso de Estudio | [S03][S04][S08][S09] | 3 | 36 % |
| Encuesta | | 0 | 0 % |

el 36 % de la muestra y corresponden a [S03], [S04], [S08] y [S09]. Particularmente en [S08], se realizaron observaciones sobre sistemas en tres proyectos diferentes en un entorno industrial utilizando métricas cuantitativas y cualitativas, sin embargo, se presenta de manera muy breve el método de evaluación en el artículo.

Las propuestas presentadas en los estudios [S01], [S02], [S05] y [S07] no siguen un proceso formal de validación, pero incluyen una prueba de concepto que describe en alto nivel su funcionamiento.

Ninguno de los estudios primarios utiliza la encuesta como mecanismo de validación, la mayoría de ellos se han centrado en evaluar las propuestas mediante métricas cuantitativas. Dentro de las principales métricas evaluadas se encuentran:

- La *Eficiencia* o *Efectividad* del sistema, particularmente en cuanto al uso de recursos [S06][S07] [S11][S02], capacidad de respuesta [S04][S05], margen de error[S02] y expresividad en el modelado [S09].
- El *Tiempo de adaptación*, determinado por el tiempo que le toma al sistema realizar las actividades propias de planeación y configuración para la adaptación de este [S04][S11], tales como despliegue de las aplicaciones [S02] o configuración de monitoreo [S03].
- La *Escalabilidad*, de acuerdo con las variaciones de carga [S10].
- La *Resiliencia* del sistema ante fallos [S10].
- La capacidad de *Concurrencia*, de acuerdo con el número de instancias activas del servicio [S05] o número de *host* almacenados [S11].
- *Flexibilidad* del modelado [S09].
- El *Esfuerzo* o carga de trabajo en [S08].

Con el ánimo de proporcionar un marco de referencia que permita identificar de manera ágil y sencilla los aspectos básicos de la adaptabilidad en sistemas de software se plantea la estructura taxonómica detallada en la tabla 3.20, el cual es el resultado de la extracción y clasificación de datos relacionados con el proceso de adaptación del sistema de los estudios

primarios. La taxonomía contempla tres niveles: (i) clase, es la agrupación principal, definida con base en el esquema de clasificación presentado en la sección 3.4.1, (ii) criterio de adaptación, determinado en conformidad con la exploración realizada sobre los principales aspectos de adaptación y detallada en la sección 3.1.3.4, algunos criterios pueden tener subcriterios de adaptación relacionados, con los cuales se pretende agregar mayor detalle para la comprensión de la taxonomía y (iv) opciones identificadas en los estudios primarios para cada uno de los atributos de adaptación.

La taxonomía planteada también apoya el posicionamiento de los trabajos existentes y trabajos futuros en cuanto a los distintos atributos de adaptabilidad y expresividad que poseen.

Tabla 3.20: Framework de clasificación taxonómica

| CLASE | CRITERIO | | OPCIONES |
|---|---|---|---|
| EC1 Aproximaciones | EC1.1 Tipo de propuesta | | Método, Framework, Herramienta, Modelo, Arquitectura, Enfoque /Aproximación |
| | EC1.2 Dominio de aplicación | | Cloud Computing, Sistemas Distribuidos, SOA, IoT, Ciudades Inteligentes, i-CSS (Intelligent Complex Software Systems) |
| | EC1.3 Contexto de uso | | Académico, Industrial |
| EC2 Tipo de Adaptación | EC2.1 Fase de adaptación | | Diseño, Integración, Despliegue, Operación, Monitorización |
| | EC2.2 Nivel de autonomía | | Completamente autónomo, Parcialmente autónomo, Humano en medio |
| | EC2.3 Tiempo de adaptación | | Diseño (Estático), Carga, Ejecución (Dinámico) |
| | EC2.4 Mecanismo de adaptación | | Abierto, Cerrado |
| | EC2.5 Actividad de adaptación | | Reactiva, Proactiva |
| | EC2.6 Condiciones de adaptación | | Auto-configuración, Auto-optimización, Auto-curación, Auto-protección |
| EC3 Motivadores | EC3.1 Propósito de adaptación | | Correctivo, Perfectivo, Adaptativo, Preventivo |
| | EC3.2 Necesidades de adaptación | | QoS (Calidad del Servicio), FT (Tolerancia a fallos), FD (Detección de fallos), NR (Nuevos Requerimientos), Optimización de recursos, SLA (Acuerdos de nivel de servicio). |
| EC4 Técnicas de adaptación | EC4.1 Frecuencia de adaptación | | Continúa, Periódica, Arbitraria |
| | EC4.2 Lógica de adaptación | | Probabilística, Basada en reglas, Basada en casos, Basada en lógica, Basada en ontologías, Basada en evidencia, Basada en lógica difusa |
| EC5 Herramientas | EC5.1 Herramientas de Soporte | EC5.1.1 Monitorización | EnviroSCALE, ConMon, Zipkin, Zuul, Logstash, Heapster, MetroFunnel |
| | | EC5.1.2 Despliegue | Docker, Kubernetes, Fleet, Dynamite |
| | | EC5.1.3 Pruebas | Travis |
| | | EC5.1.4 Almacenamiento | Etcd, Influxdb, Elasticsearch, Eureka |
| | | EC5.1.5 Integración | Eureka, Open vSwitch, Axis2 |
| | | EC5.1.6 Gestión | Activiti |
| | EC5.2 Protocolo / Estilo Arquitectura | | REST, MQTT, SOAP |
| EC6 Modelo de proceso | EC6.1 Modelo de proceso | | DevOps |
| EC7 Evaluación | EC7.1 Método de evaluación | | Experimento, Encuesta, Caso de estudio (CoS) |

## 3.5. Discusión

En relación con el objetivo de investigación:

- Si bien, durante algunos años se ha venido explorando la arquitectura de micro-servicios y el uso de contenedores en la implementación de sistemas de software, es reciente el interés por incorporar estrategias de adaptación a este tipo de proyectos. De acuerdo con los resultados de nuestra búsqueda, solo hasta 2016 se presentó el primer estudio que reúne los tres temas. Aunque ha aumentado el número de propuestas en esta dirección durante los últimos tres años, como se muestra en la figura 3.3, aún resta la exploración de nuevos caminos en la consecución de la autoadaptación de sistemas en entornos distribuidos y continuar el trabajo en pro de la maduración de las propuestas actuales.
- Fue posible construir un marco de referencia para la clasificación de los principales conceptos relacionados con adaptación, parte gracias a que, en general, los sistemas con enfoque a micro-servicios o que utilizan contenedores en el despliegue, utilizan definiciones base de adaptación para conseguir su evolución o resiliencia. Dados los resultados de la revisión sistemática, podemos concluir que las técnicas, tipos y motivadores relacionados con la adaptación son transversales e independientes de la arquitectura de la plataforma. No obstante, elementos como las aproximaciones y herramientas evolucionan en conformidad con las nuevas tecnologías.

En relación con las preguntas de investigación:

- En relación con los tipos de aproximación: El número de estudios primarios que coinciden con los criterios de selección planteados es bastante pequeño en relación con la cantidad de publicaciones encontradas, ver sección 3.3, lo cual nos lleva a la conclusión de que es un tópico en reciente exploración por parte de la academia y la industria, y explica por qué la mayoría de estudios seleccionados presentan propuestas orientadas al diseño de sistemas, tales como enfoques arquitecturales o métodos de diseño y desarrollo, según lo descrito en la subsección 3.4.2.5.
- En relación con el dominio de aplicación: Dentro de los principales beneficios de los micro-servicios se encuentran la simplificación del desarrollo distribuido y la facilidad de mantenimiento de las aplicaciones, debido a su tamaño y ágil despliegue [17]. Es por esta razón, que la construcción y publicación de estos debe ser soportada por plataformas de hardware que compartan dichas características. Lo anterior hace a los entornos Cloud un ambiente ideal para los sistemas con arquitecturas basadas en micro-servicios, debido a su escalabilidad, elasticidad y facilidad de configuración y mantenimiento de su

infraestructura [15] y esclarece el motivo por el cual es el principal dominio de uso de los estudios primarios seleccionados, como se muestra en la subsección 3.4.2.5.

- En relación con los tipos de adaptación: Los esfuerzos se concentran en la fase de recopilación y análisis de datos para comprender el comportamiento del sistema y así, proveer posteriormente mecanismos proactivos para la ejecución de cambios, ver figura 3.7. Lo anterior aporta a la autonomía del sistema en tiempo de ejecución, independientemente de la condición de adaptación empleada.

- En relación con las herramientas utilizadas: La adaptación requiere un flujo de datos constante sobre del estado del sistema para que de esta manera se puedan identificar las amenazas o inconsistencias a corregir, según sea el propósito del mecanismo. Esto conlleva a que uno de los principales elementos en las propuestas corresponda con herramientas de monitoreo que permitan recopilar y alertar al sistema sobre su desempeño y el del entorno.

# Capítulo 4

# *Framework* de Adaptación

En este capítulo se describe la propuesta de un marco tecnológico para la adaptación de servicios existentes, así como los artefactos y reglas de adaptación que lo soportan con base en los resultados obtenidos en la revisión sistemática de la literatura. El capítulo inicia con la descripción de los temas y conceptos relevantes para la construcción del *framework* en las secciones 4.1 y 4.2, continua con el planteamiento del *framework* en la sección 4.3, luego, en la sección 4.4 se presenta la estrategia para incorporación de DevOps a la propuesta inicial del *framework*. Por último, en la sección 4.5 se describen los pasos para la adopción del *framework* tomando como ejemplo la herramienta M3 System propuesta por Noor et al. en [46].

## 4.1. Adaptabilidad

Con adaptación nos referimos a acciones que conducen al cambio de un sistema software, desde la reconfiguración de la arquitectura y el reemplazo de componentes hasta cambios de parámetros.

La adaptación de un sistema software se divide en dos categorías dependiendo de quién, el sistema y/o el usuario final, esté involucrado en el proceso de adaptación: la adaptabilidad (*adaptability*) se refiere a la capacidad del usuario final para adaptar el sistema, mientras que la adaptabilidad (*adaptivity*) se refiere a la capacidad del sistema para realizar la adaptación. La adaptación de iniciativa mixta ocurre cuando tanto el usuario final como el sistema colaboran en el proceso de adaptación.

Actualmente los sistemas software deben considerar el cambio como parte del proceso de desarrollo para hacer que el software esté siempre disponible. Además, el software debe poder ejecutarse en condiciones que difieran de aquellas para las que fue diseñado inicialmente o con información incompleta del entorno de ejecución. Esto ha llevado al desarrollo de distintas aproximaciones alternativas para el diseño de software auto adaptativo [59], siendo una de las más prominentes las aproximaciones de adaptación basadas en arquitectura.

De acuerdo con la investigación presentada en [42], la capacidad de auto adaptación de un sistema está dada por la habilidad de modificar su comportamiento en tiempo de ejecución para afrontar los cambios en el entorno, en el mismo sistema o mejorar su desempeño con base en los resultados obtenidos de un proceso de retroalimentación. Desarrollar la habilidad de adaptabilidad del software se presenta en dos fases: la "modificabilidad" en el diseño y la adaptación en tiempo de ejecución. La primera consiste en la incorporación de elementos en la estructura del sistema durante la fase de diseño o compilación que le permita fácilmente abordar cambios en los requerimientos, ya sean de tipo funcional o no funcional. El segundo tipo está dado por la capacidad del sistema de cambiar respecto a su desempeño y las variables del entorno durante su operación [28].

A continuación, se describe en más detalle las características relacionadas a la modificabilidad en el diseño y la adaptación en tiempo de ejecución, en el contexto de los micro-servicios.

### 4.1.1. Modificabilidad en el Diseño

Como se mencionaba en 2.3, en el trabajo realizado por Bogner et al. [12] se realiza un análisis comparativo y de correlación entre las tácticas de modificabilidad arquitectónica para la evolución del software, los principios y patrones de diseño de los sistemas orientados a microservicios. Los principios o características asociados a los micro-servicios con los cuales se llevó a cabo el estudio fueron extraídos de [40] y se listan a continuación:

1. Componentes a través de servicios
2. Organización en torno a las capacidades empresariales.
3. Productos, no proyectos
4. *Smart endpoints*, *dumb pipes*
5. Gobernanza descentralizada
6. Gestión de datos descentralizada
7. Automatización de infraestructura
8. Diseño para el fracaso
9. Diseño evolutivo

Las tácticas de arquitectura corresponden a decisiones de diseño que impactan las propiedades de modificabilidad con el propósito final de reducir el tiempo y esfuerzo necesarios para introducir cambios futuros sobre el sistema. Los principios son características comunes

aplicables a las arquitecturas basadas en microservicios, mientras que los patrones sirven como modelo para adoptar soluciones que permitan abordar los problemas expuestos en la fase de diseño. De acuerdo con Bogner et al. [12], cinco de los principios de las arquitecturas orientadas a microservicios son satisfechos con algún tipo de táctica de modificabilidad: Componetización [P1], Organización entorno a las capacidades del negocio [P2], *Smart Endpoints, Dumb Pipes* [P3], Descentralización [P4] y Diseño evolutivo [P5], como se muestra en 4.1; siendo las tácticas las listadas a continuación:

T1 : Usar encapsulación

T2 : Mantener la *interface* existente

T3 : Restringir dependencias y vías de comunicación

T4 : Abstraer funcionalidades comunes

T5 : Incrementar la coherencia semántica

T6 : Anticipar cambios esperados

T7 : Dividir en módulos

T8 : Publicar-Suscribir

Los patrones de diseño que satisfacen las tácticas relacionadas a los principios son:

D1 : Servicio por Máquina Virtual o [D2]: Servicio por contenedor

D3 : CQRS (Segmentar responsabilidad de consulta de comandos) y [D4] : Base de datos por servicio

D5 : *Service Host*

D6 : Descomponer por dominio de negocio y/o por [D7] : Subdominio de negocio

D8 : Fuente de eventos, [D9] : Eventos de Aplicación y [D10] : Mensajería

### 4.1.2. Adaptación en tiempo de ejecución

Existen diversas aproximaciones para facilitar el diseño de sistemas con la capacidad de adaptación, principalmente son clasificadas como *Top-Down*, usualmente con un enfoque de guía centralizado, que incorpora mecanismos de monitoreo y análisis del comportamiento del sistema en relación a su objetivo de adaptación; o *Bottom-Up*, utilizadas mayormente por sistemas distribuidos que cooperan con un enfoque descentralizado y cuyas propiedades de

Tabla 4.1: Correlación de principios, tácticas y patrones de Diseño. Elaborado según Bogner et al. [12]

| CATEGORÍA | TÁCTICA | PRINCIPIOS | | | | | PATRONES | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | P1 | P2 | P3 | P4 | P5 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | D10 |
| Incrementar Cohesión | T1 | | | | | | | | | | | | | | | |
| | T2 | | | | | | | | | | | | | | | |
| | T3 | | | | | | | | | | | | | | | |
| Reducir Acoplamiento | T4 | | | | | | | | | | | | | | | |
| | T5 | | | | | | | | | | | | | | | |
| | T6 | | | | | | | | | | | | | | | |
| | T7 | | | | | | | | | | | | | | | |
| Diferir Enlace | T8 | | | | | | | | | | | | | | | |

adaptación emergen de las interacciones entre los componentes [42]. Entre las aproximaciones más conocidas aplicadas a la operación de sistemas adaptativos se encuentran:

- **Mecanismos externos de control**: Una entidad central, fuera del sistema, se encarga de tomar las decisiones en términos de adaptabilidad. Puede ser reutilizado y es independiente de la función primaria del sistema objetivo. Su grado de autonomía depende del nivel de participación de un usuario en la fase de operación. Este mecanismo es observado en [S5][S6].

- **Ingeniería Basada en Componentes**: El sistema, con el ánimo de habilitar nuevas funcionalidades o inhabilitar funcionalidades en ejecución, se diseña en torno a componentes que pueden ser almacenados en un repositorio y posteriormente ejecutados o inactivos. Este mecanismo es observado en [S3][S10].

- **Ingeniería Dirigida por Modelos**: El sistema es gobernado por modelos y el cambio o implementación de un modelo en particular es gobernado por una estrategia.

- **Sistemas Multiagente**: Los agentes son entidades autónomas e independientes dentro del sistema que pueden ser reutilizadas y tomar decisiones adaptativas. Este mecanismo es presentado en [S7].

- **Sistemas de Retroalimentación**: Uno de los principales componentes de la computación autonómica es el flujo de retroalimentación, que le permite al sistema responder de manera autónoma a sus necesidades o las del entorno a través de un proceso con cuatro fases: *monitoreo* para recolectar datos de comportamiento, usualmente mediante el uso de sondas, *análisis* para detección de variaciones en el comportamiento deseado o identificación de nuevos requerimientos, *planeación* y *ejecución* de estrategias de adaptación [8].

Este mecanismo se presenta en [S8][S9][S11].

## 4.2. Bucle MAPE-K

Como se menciona en la sección anterior, entre las aproximaciones con mayor influencia en el modelado de aplicaciones adaptativas en tiempo de ejecución se encuentran los bucles de control de retroalimentación. Particularmente, el bucle MAPE-K propuesto por IBM [30], es uno de los más ampliamente utilizados [8]. Por esta razón y debido a que corresponde con el enfoque empleado por varios de los estudios primarios seleccionados en la revisión presentada en el capítulo 3, se decide incorporar la estructura del bucle MAPE-K en el diseño del *framework* que constituye el propósito del presente trabajo. En esta sección se presenta una breve descripción de los flujos de control de retroalimentación MAPE-K y sus principales componentes.

MAPE-K puede considerarse un modelo de referencia para bucles de control en sistemas auto adaptativos. De acuerdo con este modelo, el sistema monitoriza continuamente el entorno de ejecución y su estado interno, analiza la información, planifica cómo adaptar el sistema, y ejecuta un plan de adaptación (o plan de cambios) basado en el conocimiento del sistema y el entorno. Por lo tanto, MAPE-K consiste en un flujo de cuatro fases: M(Monitoreo), A(Análisis), P(Planeación) y E(Ejecución), sobre una base de conocimiento K. Un **gestor autónomo** se encarga de la implementación parcial o general de los principales cuatro mecanismos MAPE e incluso, en ocasiones, gestiona otros gestores autónomos usando el mismo flujo, como se observa en la figura 4.1. Un gestor que se encarga exclusivamente de una tarea adaptativa es llamado **gestor autónomo de punto de contacto**, pues gestiona los recursos del sistema a través de un bloque de código que implementa las funcionalidades de un **sensor** o **efector**, llamado *touchpoint*, punto de contacto en español.

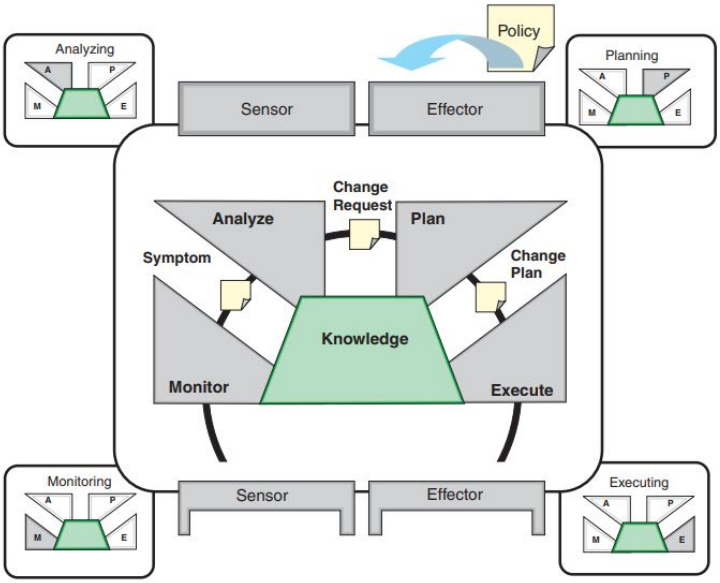


Figura 4.1: Gestor autónomo. Tomado de [30]

**Componentes del blucle MAPE-K**:

- **Sensor**: Es una interfaz encargada de exponer el estado de los recursos del sistema y sus transacciones.
- **Efector**: Es una interfaz que habilita la implementación de cambios en los recursos del sistema.
- **Monitor**: La función de monitor recopila los detalles de comportamiento, topología, configuración, propiedades, estado, capacidad, entre otros, de los recursos del sistema a través de puntos de contacto y los correlaciona con síntomas que se pueden analizar posteriormente. Los gestores autónomos deben recopilar, organizar y dar sentido a los datos obtenidos desde la interfaz del sensor de punto de contacto de un recurso.
- **Analizar**: Esta fase proporciona los mecanismos para observar y analizar situaciones y determinar si es necesario realizar algún cambio en el sistema. En muchos casos, la tarea de análisis modela el comportamiento complejo, por lo que puede emplear técnicas de predicción que permiten al gestor autónomo aprender sobre el entorno y predecir el comportamiento futuro, tales como modelos de colas y predicción de series de tiempo.
- **Planear**: La actividad de planeación se encarga de seleccionar o crear el procedimiento para aplicación de los cambios sobre el recurso administrado. Estos procedimientos pueden ser ir desde la ejecución de bloques de código hasta la implementación de un flujo de trabajo de alta complejidad. El conjunto de cambios a ejecutar se conoce como plan de cambios.

- **Ejecutar**: Se encarga de programar y ejecutar los cambios definidos en el plan de cambios mediante la interfaz del efector de punto de contacto de un recurso. Parte de la ejecución del plan de cambio podría implicar la actualización de los conocimientos que utiliza el gestor autónomo.
- **Base de conocimiento**: Fuente o repositorio de almacenamiento del conocimiento común de los gestores autónomos para la ejecución de las cuatro actividades principales, descritas anteriormente. En [30] se describen tres tipos diferentes de conocimiento: sobre la topología de la solución, sobre políticas o reglas de negocio y datos relevantes para la determinación de problemas, tales como métricas, síntomas y árboles de decisión.

## 4.3. Descripción del *Framework*

Las estrategias de adaptación se definen según el contexto y las necesidades del usuario en dicha materia. Según lo planteado por Liu, Peini et al. en [41], existen tres tipos de contexto en un sistema con arquitectura orientada a microservicios, según los cambios a realizar:

- Contexto externo: Comprende el hardware y sistema operativo, el sistema es consciente de dicho entorno, pero no lo controla.
- Contexto sistema: Comprende una instancia en ejecución del sistema o servicio.
- Contexto de aplicación: Incluye la topología de red y capa de aplicación.

De acuerdo con la taxonomía presentada en la sección 3.4, las principales necesidades de adaptación definidas en los estudios (EC3.2) corresponden con la calidad o desempeño del servicio (QoS), la posibilidad de incorporar nuevos requerimientos al sistema, la tolerancia ante fallos y la optimización de recursos. Así mismo, este listado de motivadores nos proporciona una hoja de ruta para definir los propósitos de adaptación a abordar y contextos a impactar.

Habiendo definido motivadores, propósitos y contextos para la adaptación, con enfoque en un mecanismo basado en un flujo de control de retroalimentación de acuerdo con lo expuesto en la sección 4.2, se clasifican las tareas y técnicas en concordancia con las fases del bucle MAPE-K. De esta manera, se construye la primera aproximación del marco de referencia para adaptación dinámica (en tiempo de ejecución) de sistemas cuya arquitectura se encuentre basada en micro-servicios y contenedores, presentado en la figura 4.2.

#### 4.3.0.1. Necesidad de adaptación

El *framework* propuesto, parte del motivo o necesidad de adaptación que se desea cubrir en el sistema. Las necesidades pueden ser, o relacionarse, con alguna de las principales cuatro identificadas en la taxonomía:

- **Mitigación de fallos**: Reúne las necesidades relacionadas con la prevención, detección y corrección de fallos [51]:
  - Tolerancia a fallos: Capacidad del sistema de detectar y recuperarse ante posibles eventos en tiempo de ejecución, con el fin de evitar fallas en el sistema.
  - Detección de fallos: Detección de errores en una fase de validación del sistema para evitar futuros fallos.
- **Requerimientos No Funcionales - QoS**: Reúne todos los sistemas cuya principal motivación corresponde con el cumplimiento, incorporación al sistema o mejora de requerimientos no funcionales, mayormente, aquellos orientados a la calidad del servicio.

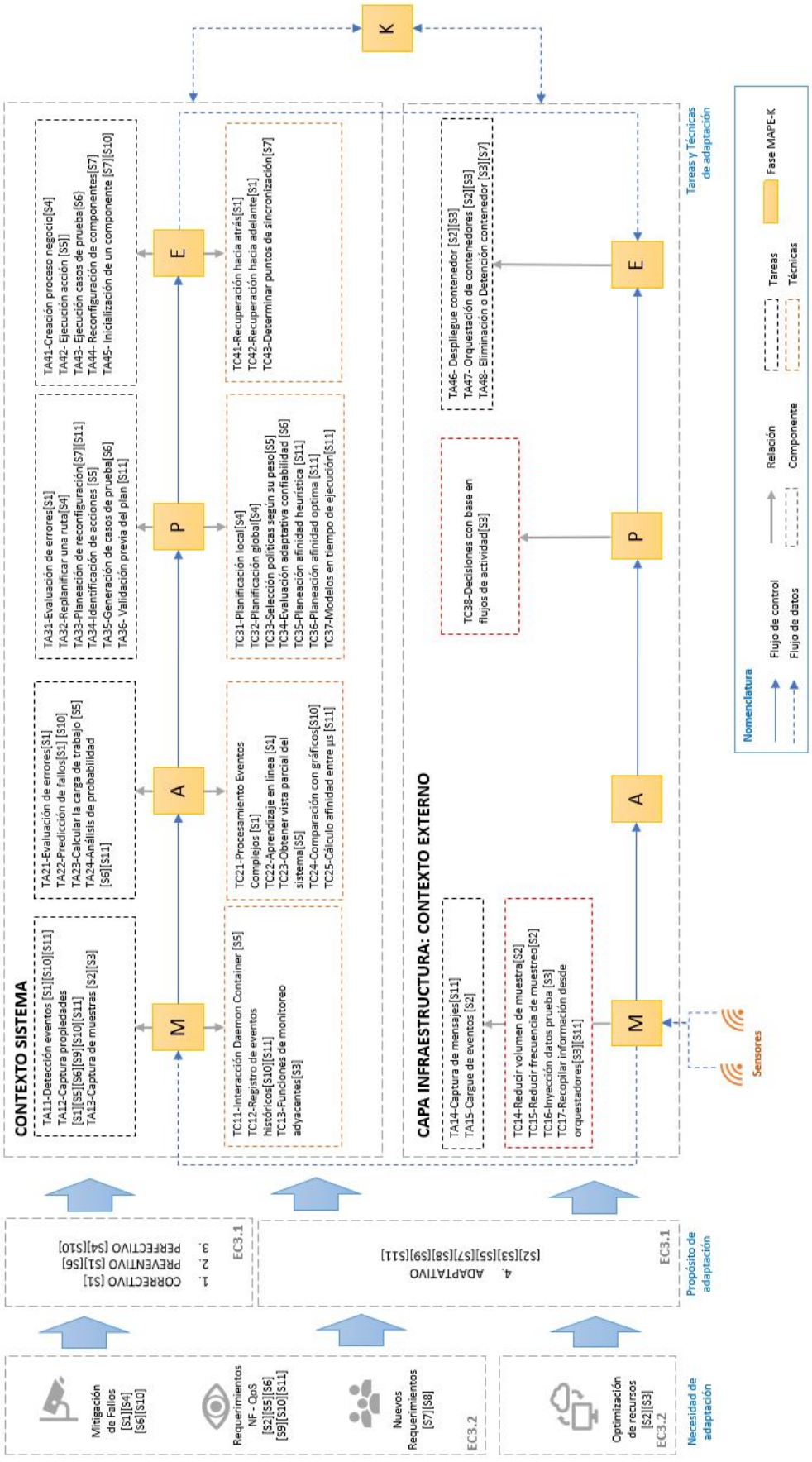


Figura 4.2: *Framework:* Tareas

- **Nuevos Requerimientos**: Capacidad del sistema de incorporar nuevos requerimientos de negocio o mejorar los existentes de forma evolutiva.
- **Optimización de recursos**: Facultad del sistema relacionada con la mejora o mantenimiento del desempeño de los recursos de hardware y software que lo componen.

#### 4.3.0.2. Propósito de adaptación

Como se expone en la sección 3.4, las necesidades y propósitos de adaptación se encuentran conectados. En concordancia con esto, en el diseño del *framework*, se propone la definición del propósito de adaptación en correspondencia con la necesidad previamente identificada, utilizando como referencia los resultados de la revisión sistemática en esta materia, ilustrados en la tabla 3.16.

#### 4.3.0.3. Contexto del sistema

De acuerdo con lo mencionado en la introducción de esta sección, es importante definir el contexto sobre el cual implementar las acciones de adaptación, Si bien, es posible impactar varios contextos independientemente del propósito, dentro del *framework* se establece la guía hacía dos contextos en particular:

- Contexto externo: Las necesidades de adaptación relacionadas con la optimización de recursos posiblemente requieran un mayor impacto sobre los componentes de la capa de infraestructura, en el entorno que nos compete, corresponden a contenedores, máquinas virtuales, sensores y demás elementos de hardware o software que no son parte de la aplicación misma.
- Contexto sistema: Comprende todos los elementos activados durante la ejecución de una o varias instancias de la aplicación. En este caso, el propósito de adaptación es indiferente, pero las necesidades las limitaremos a aquellas que no conllevan la optimización de recursos físicos o virtuales externos al sistema.

Dentro del contexto, el proceso de adaptación se lleva a cabo mediante un flujo de retroalimentación MAPE-K. Para cada una de las fases del flujo: monitoreo, análisis, planeación, ejecución, se establecen los elementos (tareas, herramientas y técnicas) identificados en los estudios seleccionados dentro de la revisión, que apoyan su ejecución; así como las relaciones entre ellos.

### 4.3.1. Tareas para la adaptación

Una vez se ha identificado la necesidad y propósito de adaptación, el *framework* proporciona una guía sobre las tareas y técnicas que pueden ser incorporadas en cada fase del flujo de control de retroalimentación MAPE, con el ánimo de facilitar la definición de la estrategia de adaptación a adoptar. Las tareas, en este escenario, corresponden a un conjunto de pasos necesarios para llevar a cabo el objetivo de la fase del flujo al cual pertenecen. A continuación, se describen las tareas o actividades que fueron identificadas en cada uno de los estudios primarios resultantes de la revisión en relación con el contexto y fase del flujo impactado, el listado completo puede apreciarse en la tabla 4.2. Cada tarea se encuentra señalada con las siglas T.A. (Tarea de adaptación), el número de fase (1: monitoreo, 2: Análisis, 3: Planeación: 4: Ejecución) y consecutivo de la tarea.

1. **TA11-Detección de eventos**: Un evento usualmente es un ligero cambio en el estado del sistema o sus componentes que puede alertar sobre un posible error. En [S01] se utiliza una interfaz API de comunicación entre el sistema y los dispositivos IoT para que estos últimos puedan publicar, a través de un método POST los errores detectados o recibidos para ser posteriormente analizados. Estas interfaces REST son provistas por un micro-servicio de tipo *Edge* [51]. En [S10] los eventos que reflejan el comportamiento de los componentes son detectados, unificados y almacenados. En el estudio [S11] los eventos son capturados y almacenados a través de un adaptador de monitoreo.

2. **TA12-Captura de propiedades**: Consiste en la identificación de datos que proporcionen al sistema información relevante sobre su estado. [S01] utiliza para la lectura de las propiedades el mismo mecanismo empleado en la detección de eventos. [S05] y [S11] colectan a través de monitores información del desempeño de los contenedores y micro-servicios, como, por ejemplo, tiempos de respuesta o capacidad de procesamiento empleada. En el estudio [S06] se captura el número de invocaciones de un método en un micro-servicio y la respuesta obtenida (exitosa/falla), a través de un analizador de registros. En [S09] se monitorizan y capturan parámetros en tiempo de ejecución relacionados con la calidad del servicio y la forma en la que los micro-servicios adaptan aspectos de granularidad, tales como tasa de cambio o tasa de fallo. Sobre cada parámetro monitorizado se almacena el tipo, nombre y secuencia de valores registrados.

3. **TA13-Captura de muestras**: Es una tarea que permite al sistema, usualmente a través de sensores, recopilar datos para posteriormente ser analizados. En [S02] cada sensor realiza una lectura periódica de la calidad del aire. Las lecturas, corresponden con números de valor real obtenidos del sensor en un momento y lugar determinados. Estos datos, lla-

Tabla 4.2: Tareas para la adaptación

| FASE | CONTEXTO | TAREA |
|---|---|---|
| Monitoreo | Sistema | TA11-Detección de eventos |
| | | TA12-Captura de propiedades |
| | | TA13-Captura de muestras |
| | Externo | TA14-Captura de mensajes |
| | | TA15-Carga de eventos |
| Análisis | Sistema | TA21-Evaluación de errores |
| | | TA22-Predicción de fallos |
| | | TA23-Calcular carga de trabajo |
| | | TA24-Análisis de probabilidad |
| Planeación | Sistema | TA31-Evaluación de errores |
| | | TA32-Replanificar ruta |
| | | TA33-Planeación de reconfiguración |
| | | TA34-Identificación de acciones |
| | | TA35-Generación de casos de prueba |
| | | TA36-Validación previa del plan |
| Ejecución | Sistema | TA41-Creación de proceso de negocio |
| | | TA42-Ejecución de acción |
| | | TA43-Ejecución de casos de prueba |
| | | TA44-Reconfiguración de componentes |
| | | TA45-Inicialización de un componente |
| | Externo | TA46-Despliegue de contenedor |
| | | TA47-Orquestación de contenedores |
| | | TA48-Eliminación o detención de un contenedor |

mados muestras, tienen una marca de tiempo y una etiqueta geográfica con los valores de latitud y longitud obtenidos del GPS [52]. En [S03] un contenedor recibe una copia de los paquetes enviados por la aplicación que está siendo monitorizada y se encarga de filtrarlos, copiarlos y guardar sus atributos de tiempo.

4. **TA14-Captura de mensajes**: Otra forma de observar el comportamiento de las micro-aplicaciones, consiste en la lectura de los mensajes intercambiados entre los micro-servicios, incluido el origen y el destino del mensaje, el tamaño de la carga útil y tiempo de respuesta del receptor. Esta tarea es ejecutada en [S11] a través de un adaptador de monitoreo [54].

5. **TA15-Carga de eventos**: Tras la detección de los eventos y datos del entorno por parte del sensor, dicha información requiere ser cargada al sistema para su posterior análisis. En [S02], por ejemplo, una rutina de carga es invocada periódicamente con el fin de cargar datos empaquetados por el componente de monitorización desde una cola en memoria.

6. **TA21-Evaluación de errores**: La evaluación de errores permite la inferencia de errores con base en datos anteriores. Toma como referencia una lista de errores y propiedades que se utilizan para detectar un nuevo error, así como acciones que se han ejecutado al intentar recuperarse del error [51]. En [S01] esta actividad se ejecuta mediante el método probabilístico CEP, descrito en la sección 4.3.2.

7. **TA22-Predicción de fallos**: Para conseguir la predicción de fallos, en [S01], el sistema recibe evaluaciones de errores y propiedades cuyos resultados son utilizados para entrenar el algoritmo de aprendizaje. Esto le permite no solo identificar errores y los estados del sistema que los llevaron a ellos, sino también como el sistema intenta recuperarse. Con este conocimiento, se generan patrones de fallas y usando los datos del sistema posteriores, es posible inferir probabilísticamente la ocurrencia de errores en el futuro [51]. En [S10] esta actividad se ejecuta mediante la comparación de un estado deseado del sistema y el estado real.

8. **TA23-Calcular la carga de trabajo**: En [S05] se utiliza este método de cálculo de la carga de trabajo de los micro-servicios con base en la información recopilada por el monitor respecto al tiempo de respuesta de estos y los recursos disponibles en los contenedores de aplicación. Los resultados obtenidos son mezclados con los datos de operación de otros micro-servicios con el fin de obtener una vista parcial del sistema.

9. **TA24-Análisis de probabilidad**: Una de las aproximaciones para conocer el estado del sistema corresponde al análisis de probabilidad de uso, ocurrencia de un evento o incluso fallo de uno de sus componentes. En [S06] con el fin de estimar la confiabilidad del

sistema, se determina la probabilidad de fallo en una demanda (invocación de un método de un micro-servicio) y de uso de los micro-servicios que son objeto de prueba.

10. **TA31-Evaluación de errores**: Esta actividad se ejecuta en [S01] tanto en la fase de análisis, como en la fase de planeación. En esta última, permite elegir las acciones a ejecutar para la recuperación del sistema con base en la supuesta falla que potencialmente causó el error y en conformidad con su clasificación. Debido a que las acciones para selección se definen en la etapa de diseño o carga, el mecanismo de adaptación podría ser principalmente cerrado.

11. **TA32-Replanificar una ruta**: La auto adaptación en tiempo de ejecución de servicios mediante replanificación tiene como actividades principales la planificación, la ejecución y la auto adaptación. Cuando ocurre una interrupción en un servicio basado en flujos de trabajo en tiempo de ejecución, la operación de auto adaptación invoca la operación de planificación para planificar una nueva ruta y luego invoca la operación en ejecución para ejecutar la ruta definida. De esta manera se asegura el cumplimiento de los objetivos del servicio [65]. Esta estrategia es utilizada en [S04] con el fin de reparar fallas en los servicios.

12. **TA33-Planeación de reconfiguración**: La reconfiguración de un sistema o componente es una de las vías utilizadas para lograr la adaptabilidad. Una de las tareas del planificador, consiste en la definición de los pasos y actividades necesarias para lograr la reconfiguración, posiblemente en tiempo de ejecución. En [S07] se propone reconfigurar aplicaciones distribuidas durante la operación basadas en una arquitectura de contenedor en tiempo real, en este caso, cada agente cuenta con una lista de instrucciones para su reconfiguración, de la cual solo se ejecutan los pasos aplicables a cada componente [62].

13. **TA34-Identificación de acciones**: Una de las tareas más comunes a realizar por parte del planificador consiste en la identificación y selección de acciones a ejecutar para lograr la adaptación del sistema. Usualmente, el número de acciones a ejecutar es limitado y se encuentran previamente definidas, por lo que decimos que el mecanismo de adaptación es cerrado. En [S05], el componente de planificación se encarga de decidir el modo de acción con base en la información parcial del sistema entregada por el analizador y un mecanismo que emplea políticas y estrategias.

14. **TA35-Generación de casos de prueba**: En los sistemas, cuyo propósito de adaptación es preventivo, como es el caso de [S06], una de las actividades principales del planificador se convierte en la definición de casos de prueba en tiempo de ejecución para evaluar la confiabilidad del componente que está siendo probado, de esta manera se pueden identificar

posibles fallos.

15. **TA36-Validación previa del plan**: Debido al entorno cambiante en el que se ejecutan los micro-servicios, realizar cambios o reconfiguraciones sobre los mismos en tiempo de ejecución podría resultar en una falla del sistema. Con esta tarea se pretende mitigar dicho riesgo, mediante la validación de los cambios en el modelo antes de la aplicación en la micro-aplicación. Esta validación previa de los movimientos, abordada en el estudio [S11] garantiza la seguridad e integridad de la aplicación.

16. **TA41-Creación de proceso de negocio**: Si se cuenta con un sistema cuyos servicios se ejecutan con base en un proceso, una de las actividades a ejecutar con el fin de recuperar el sistema en un punto de interrupción puede ser la generación y ejecución de un nuevo flujo de actividades. Así lo plantea [S04], para lo cual utiliza un motor de procesos que se encarga de convertir una ruta encontrada por el planificador en un nuevo proceso usando notación BPM. Este proceso, a su vez, se ejecuta empleando unidades auto adaptativas que podrían reemplazarse por un nuevo flujo de actividades en caso de presentarse un fallo [65].

17. **TA42-Ejecución de acciones**: El componente de ejecución se encarga de correr, en tiempo de ejecución, acciones que promuevan la adaptación del sistema. En el caso de [S05], acciones empaquetadas en una política, orientadas a la migración o replicación de micro-servicios, se ejecutan sobre el contenedor que aloja el micro-servicio.

18. **TA43-Ejecución casos de prueba**: En el estudio presentado en [S06], cuyo objetivo es la estimación de la confiabilidad de un micro-servicio, el sistema, tras la selección adaptativa de los mejores escenarios de prueba, se encarga de ejecutarlos para recopilar los resultados en términos de éxito o fallo, de manera que estos datos sirvan como insumo para la estimación.

19. **TA44-Reconfiguración de componentes**: La reconfiguración de componentes de software dentro del sistema como mecanismo para conseguir su adaptación, se puede ejecutar de varias maneras, según lo expuesto en [S07]:

    - Agregando un nuevo componente de software al sistema en ejecución
    - Cambiando a un nuevo nodo de ejecución un componente actual
    - Actualizando un componente de la aplicación
    - Eliminando un componente del sistema
    - Ejecutando en paralelo o secuencialmente las actividades descritas anteriormente.

En [S11] el planificador se encarga de la reconfiguración de la aplicación mediante la reubicación de los micro-servicios que la componen.

20. **TA45-Inicialización de un componente** Una tarea de adaptación puede consistir en la instanciación de un módulo de software o servicio que no se encontraba en ejecución. En [S07] para inicializar un nuevo agente de software dentro de un contenedor, la imagen debe haber sido previamente instalada. El sistema utiliza un contenedor de construcción, al cual se le asigna un porcentaje estático de los recursos del sistema en el nodo (red y CPU) para que se ejecute en paralelo con la aplicación integrada sin interferir con ella [62]. En [S10] un nuevo componente es instanciado incluso si falla el contenedor donde se ha estado ejecutando la lógica de escalado automático, en dicho caso, es posible iniciar un nuevo componente para que se haga cargo de la base de conocimientos y la lógica de escalado desde el último punto de ejecución [63].

21. **TA46-Despliegue de contenedor**: Los contenedores, como se menciona en la sección 2, facilitan la virtualización de aplicaciones en una forma ligera a través de la simplificación del proceso de despliegue y lanzamiento de servicios o aplicaciones, esto los convierte en un aliado para la ejecución de estrategias de adaptación. En [S02] los datos recopilados para análisis son empaquetados como bibliotecas y configuraciones en tiempo de ejecución en imágenes *Docker* debido a su eficiencia en el uso de recursos y rápido tiempo de respuesta. En el estudio [S03] las funciones de monitoreo son desplegadas en contenedores interconectados a través de *switches* virtuales para capturar datos a ser procesados e identificar posibles cambios en las comunicaciones de la aplicación [44].

22. **TA47-Orquestación de contenedores**: El creciente interés por la virtualización de aplicaciones y su despliegue a través de contenedores, refuerza la necesidad de la gestión y sincronización de estos contenedores; a dicho proceso se le ha llamado orquestación. Entre sus beneficios se encuentran la facilidad en el dimensionamiento y manejo de *clusters* de contenedores, rapidez y automatización del despliegue proporcionando flexibilidad en la migración y reconfiguración [49]. En el estudio [S02], se utiliza *Kubernetes* para la orquestación, despliegue y migración de las imágenes *Docker* que contienen los datos de muestra para análisis. La herramienta adopta el modelo cliente-servidor, donde varios dispositivos IoT están conectados a un servidor y los clientes informan periódicamente los niveles de recursos de los dispositivos IoT al servidor. Posteriormente, el servidor de *Kubernetes* orquesta en los clientes el lanzamiento de imágenes *Docker* en los dispositivos IoT en las ubicaciones correctas [52]. En [S03] cuando se recibe un evento para instanciar o remover un contenedor de aplicación, el sistema de gestión se encarga de iniciar o eliminar los contenedores de monitoreo requeridos y almacenar o actualizar en una base de

datos el mapeo entre los contenedores de aplicación y los de monitoreo [44].

23. **TA48-Eliminación/Detención de contenedor**: En ocasiones, la adaptación del sistema requiere que se eliminen funcionalidades o información desplegada previamente en un contenedor. El orquestador se encarga de ejecutar esta acción, un ejemplo se ilustró en la descripción de la tarea anterior para el estudio [S03]. En el estudio [S07] para llevar a cabo esta operación, se ejecuta un contenedor de construcción adicional en cada nodo durante toda la vida útil del sistema. El contenedor de construcción es un *job* en segundo plano que se ejecuta en un *cgroup* dedicado.

Las tareas pueden ser seleccionadas en función del propósito, condición, actividad y/o mecanismo de adaptación que se desee emplear, como se especifica en la tabla 4.3. Normalmente, las tareas relacionadas a las fases de monitoreo y análisis están determinadas por el tipo de actividad (Criterio EC2.5), mientras que las tareas de planeación y ejecución se establecen con base en las condiciones (EC2.6) y mecanismos de adaptación (EC2.4). Esta relación representa una sugerencia con base en los datos extraídos de los estudios primarios que fueron parte de la revisión sistemática.

Tabla 4.3: Criterios de selección de tareas

| TAREA | PROPÓSITO | | | | ACTIVIDAD | | CONDICIÓN | | | | MECANISMO | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Correctivo | Preventivo | Perfectivo | Adaptativo | Reactiva | Proactiva | Configuración | Optimización | Curación | Protección | Abierto | Cerrado |
| TA11 | ■ | ■ | ■ | ■ | ■ | ■ | | | | | | |
| TA12 | ■ | ■ | | ■ | ■ | ■ | | | | | | |
| TA13 | | | | ■ | | ■ | | | | | | |
| TA14 | | | | ■ | | ■ | | | | | | |
| TA15 | ■ | ■ | ■ | ■ | | ■ | | | | | | |
| TA21 | ■ | ■ | | | ■ | ■ | | | | | | |
| TA22 | ■ | ■ | | | | ■ | | | | | | |
| TA23 | | | ■ | ■ | | ■ | | | | | | |
| TA24 | | ■ | ■ | | | ■ | | | | | | |
| TA31 | ■ | ■ | | | ■ | ■ | | | ■ | ■ | | ■ |
| TA32 | | | ■ | ■ | | | ■ | ■ | ■ | | ■ | |
| TA33 | | | | ■ | | | ■ | ■ | | | ■ | |
| TA34 | | | | ■ | | | ■ | ■ | | | | ■ |
| TA35 | | ■ | | | | | | | | ■ | | ■ |
| TA36 | ■ | ■ | ■ | ■ | | | ■ | ■ | ■ | ■ | ■ | |
| TA41 | | | ■ | ■ | | | ■ | ■ | ■ | | ■ | |
| TA42 | ■ | ■ | ■ | ■ | | | ■ | ■ | ■ | ■ | | ■ |
| TA43 | | ■ | | | | | | | | ■ | | ■ |
| TA44 | | | | ■ | | | ■ | ■ | | | ■ | |
| TA45 | ■ | | | ■ | | | ■ | ■ | ■ | | ■ | |
| TA46 | ■ | ■ | ■ | ■ | | | ■ | ■ | ■ | ■ | ■ | ■ |
| TA47 | ■ | ■ | ■ | ■ | | | ■ | ■ | ■ | ■ | ■ | ■ |
| TA48 | ■ | ■ | ■ | ■ | | | ■ | ■ | ■ | ■ | | ■ |

### 4.3.2. Técnicas para la adaptación

En esta subsección se describen los mecanismos o estrategias empleados por los diferentes estudios primarios para lograr la adaptación de los sistemas. En la tabla 4.4 es posible observar el listado completo de técnicas según la fase del flujo MAPE a la que apoya, el contexto en el que se aplica y la lógica de adaptación que sigue. En total se identificaron 20 técnicas diferentes para el manejo u optimización del proceso de adaptación. Cada técnica se encuentra señalada con las siglas T.C., el número de fase (1: monitoreo, 2: Análisis, 3: Planeación: 4: Ejecución) y el consecutivo correspondiente.

Tabla 4.4: Técnicas para la adaptación

| FASE | CONTEXTO | TÉCNICA | LÓGICA |
|---|---|---|---|
| Monitoreo | Sistema | TC11-Interacción Daemon Container | Reglas |
| | | TC12-Registro de eventos históricos | Reglas |
| | | TC13-Funciones de monitoreo adyacentes | Reglas |
| | Externo | TC14-Reducir volumen de muestra | Lógica |
| | | TC15-Reducir frecuencia de muestreo | Lógica |
| | | TC16-Inyección datos prueba | Reglas |
| | | TC17-Recopilar información desde orquestadores | Reglas |
| Análisis | Sistema | TC21-Procesamiento eventos complejos | Probabilista |
| | | TC22-Aprendizaje en línea | Probabilista |
| | | TC23-Obtener vista parcial del sistema | Reglas |
| | | TC24-Comparación con gráficos | Reglas |
| | | TC25-Cálculo afinidad entre micro-servicios | Reglas |
| Planeación | Sistema | TC31-Planificación local | Lógica |
| | | TC32-Planificación global | Lógica |
| | | TC33-Selección políticas según su peso | Probabilista |
| | | TC34-Evaluación adaptativa de confiabilidad | Probabilista |
| | | TC35-Planeación de afinidad heurística | Reglas |
| | | TC36-Planeación de afinidad óptima | Reglas |
| | | TC37-Modelos en tiempo de ejecución | Reglas |
| | Externo | TC38-Decisiones con base en flujos de actividades | Reglas |
| Ejecución | Sistema | TC41-Recuperación hacia atrás | Probabilista |
| | | TC42-Recuperación hacia adelante | Probabilista |
| | | TC43-Determinar puntos de sincronización | Reglas |

1. **TC11-Interacción *Daemon Container***: Una manera de obtener información respecto al estado de contenedores *Docker* y las aplicaciones alojadas en ellos, corresponde con la incorporación de un *Daemon* para comunicar dichos datos al monitor de la aplicación. Esta estrategia es utilizada en [S05].

2. **TC12-Registro de eventos históricos**: En ocasiones, los registros de eventos por si mismos no entregan información relevante del comportamiento o desempeño del sistema, una técnica para aprovechar mejor estos datos consiste en almacenamiento de los eventos en conjunto con datos secundarios que faciliten su posterior análisis. Este mecanismo es empleado en [S10] y [S11].

3. **TC13-Funciones de monitoreo adyacentes**: Contrario a la captura de datos directamente del contenedor que aloja la aplicación (TC11), en [S3] se plantea esta estrategia para implantar funciones de monitoreo en contenedores adyacentes a los de aplicaciones que permitan recopilar eventos y propiedades relevantes de los contenedores de aplicación.

4. **TC14-Reducir volumen de muestra**: Estrategia de adaptación que permite realizar análisis sencillos de IoT y enviar información procesada de menor volumen a través de las redes de comunicación con el fin de optimizar los recursos de red. Para lograr esta estrategia, en [S02] se empleó un *middleware* basado en contenedores, donde la analítica se puede implementar de manera *plug-and-play* según el tipo de procesamiento de información necesario.

5. **TC15-Reducir frecuencia de muestreo**: Consiste en una estrategia, empleada en [S02] para reducir la carga en la comunicación mediante la observación de patrones de detección y posterior ajuste de los parámetros en los sensores. Particularmente, en el estudio es utilizada para reducir el volumen de datos que se envían a través de una red móvil, para lo cual, se adapta el intervalo de muestreo de los sensores con base en la capacidad de carga de la plataforma en un intervalo de tiempo dado, la relevancia del sensor o la varianza de las muestras [52].

6. **TC16-Inyección de datos de prueba**: Consiste en el envío de datos de prueba entre los componentes de un sistema, usualmente empaquetados, para evaluar las características de comunicación entre ellos. En [S03] un sensor se ejecuta dentro de un contenedor e inyecta paquetes en la red que se reciben en otro sensor. Los paquetes enviados se pueden recopilar en el sensor del receptor o reflejarse en el sensor del remitente. De esta manera, se estudia la interacción entre los paquetes de sondeo y el tráfico cruzado para comprender el comportamiento de la red [44].

7. **TC17-Recopilar información desde orquestadores**: Un enjambre de contenedores es gestionado a través de un orquestador, como *Docker Swarm* o *Kubernetes*. A través de estas herramientas se puede recopilar información del estado completo del sistema. Esta estrategia es utilizada en [S3] y [S11].

8. **TC21-Procesamiento de eventos complejos (CEP)**: Este mecanismo es utilizado en [S01] para analizar y combinar cadenas de eventos primitivos en pro de detectar nuevas situaciones complejas de alto nivel que sean de interés para el propósito final del proceso, en este caso, la identificación de los focos de falla. Lleva el mismo nombre del paradigma que lo sustenta *Complex Event Processing* [51].

9. **TC22-Aprendizaje en línea**: OL por sus siglas en inglés (*Online Learning*) es una técnica de *Marchine Learning* empleada en [S01] para predecir posibles fallos con base en los datos recopilados sobre el sistema y la efectividad de las estrategias de recuperación utilizadas con base en patrones de falla. En OL el algoritmo de predicción se actualiza y adapta con los datos que provee de manera continua el monitor para generar una secuencia de hipótesis.

10. **TC23-Obtener vista parcial del sistema**: Contrario a la identificación de los parámetros de comportamiento de un único micro-servicio o la totalidad de los micro-servicios del sistema, en [S05] se propone una actividad que permita capturar una visión parcial del estado del sistema mediante el análisis de los datos de desempeño de un micro-servicio y sus pares. esto puede conllevar a la toma de mejores decisiones en términos de recursos y de acuerdo con [25] la sobrecarga en el proceso de comunicación es mínima.

11. **TC24-Comparación de gráficos**: Parte de la idea de que una aplicación orientada a microservicios se puede representar con un tipo de gráfico de microservicios que se invocan entre sí y un gráfico de instancia que representa las múltiples instancias de microservicios que se están ejecutando para proporcionar garantías de rendimiento y resistencia. La estrategia consiste en la comparación del gráfico de la instancia del micro-servicio en ejecución contra el gráfico que representa el estado deseable del sistema a través de un componente de software dedicado a esta actividad para identificar posibles inconsistencias [63].

12. **TC25-Cálculo afinidad entre micro-servicios**: Es un mecanismo para detectar comportamientos inesperados en los componentes del sistema mediante la evaluación de la afinidad entre dos micro-servicios con base en el número de mensajes y la cantidad de data que se comparten entre ellos. De esta manera, si se evidencia que dos micro-servicios alojados en diferentes *hosts* mantienen una comunicación fluida con un número elevado de mensajes y/o volumen alto de datos por mensaje (afinidad alta) el planificador puede plantear una reubicación de los micro-servicios para hacer más efectivo el tránsito de mensajes. Esta estrategia es implementada en [S11].

13. **TC31-Planificación local**: En [S04] se propone este mecanismo que pretende la imple-

mentación de un planificador para buscar una ruta de proceso óptima para recuperación ante un fallo dentro de una base de conocimiento de servicio SKB(*Service Knowledge Base*), la cual consiste en un sistema DDL(*Dynamic Description Logic*) que agrupa todos los servicios denotados por una misma ontología [65]. Dentro de cada SKB se encuentra un planificador local.

14. **TC32-Planificación global**: Consiste en un mecanismo similar a la planificación local de rutas, pero en esta ocasión a través de un sistema distribuido de SKBs, D3L(*Distributed Dynamic Description Logic*), llamado DSKB (*Distributed Service Knowledge Base*), en el cual se crean diferentes reglas puente con el ánimo de resolver conflictos semánticos entre los diferentes conceptos. Usualmente, la replanificación local es ejecutada en primera instancia, si es exitosa, la ruta original seleccionada no se ve afectada, de lo contrario, se ejecuta la replanificación global, terminando previamente todos los servicios en ejecución en la ruta para reparar la ruta por completo.

15. **TC33-Selección de políticas según su peso**: Siendo las políticas reglas que incorporan la ejecución de una o más acciones, estas pueden ser seleccionadas en conformidad con su peso en relación con un umbral previamente definido. En el caso de [S05], el planificador establece para cada micro-servicio una escala ponderada, el peso de la política se calcula con base en dicho dato, el valor de la métrica del micro-servicio y un umbral definido por el usuario en la configuración previa del agente. Las políticas por ejecutar son seleccionadas según el resultado de un algoritmo que define un umbral de manera probabilística para un arreglo de políticas determinado aleatoriamente [65].

16. **TC34-Evaluación adaptativa de confiabilidad**: El algoritmo de evaluación adaptativa de confiabilidad para micro-servicios MART(*Microservice Adaptive Reliability Testing*) se utiliza para la generación de casos de prueba sobre una estructura de red donde cada nodo es un espacio de prueba y los enlaces entre los nodos representan una dependencia entre las estimaciones del evaluador respecto a la probabilidad de falla de los espacios de prueba de un mismo método. MART tiene por objetivo seleccionar el marco de prueba con mayor probabilidad de tener demandas(invocaciones) fallidas. Combina dos técnicas de muestreo: una basada en el peso y un muestreador aleatorio simple para seleccionar el siguiente marco de prueba. El mecanismo basado en el peso sigue los vínculos entre los nodos de la red para identificar posibles grupos de demandas fallidas a través de una búsqueda en profundidad, se equilibra con el muestreador aleatorio simple, el cual realiza una exploración amplia del espacio del marco de prueba [50].

17. **TC35-Planeación de afinidad heurística**: En [S11] el planificador heurístico organiza la

ubicación de los microservicios que componen una micro-aplicación en un clúster. El planificador calcula cómo reorganizar los microservicios para que aquellos con alta afinidad se ubiquen en un mismo *host*, teniendo en cuenta los recursos disponibles en el *host*. El cálculo de la lista movimientos necesarios para reconfigurar las micro-Apps está inspirado en el algoritmo de aproximación First-Fit [54].

18. **TC36-Planeación de afinidad óptima**: En este escenario, también presentado en el estudio [S11], el planificador se encarga de la optimización de la ubicación de micro-Apps con base en una lista de afinidades entre micro-servicios en un clúster, tratando de minimizar el número de *hosts* usados para el despliegue de los micro-servicios y maximizar la afinidad en cada clúster [54].

19. **TC37-Modelos en tiempo de ejecución**: El uso de modelos en tiempo de ejecución (*model@runtime*) simplifica la inspección y adaptación de sistemas complejos y heterogéneos ya que evita la aplicación de cambios directamente sobre el sistema [54]. El modelo es una vista de alto nivel que solo expone la estructura y el comportamiento relevantes del sistema de acuerdo con la intención de uso del modelo y puede ser expuesto en ambiente de producción. En [S11] el modelo proporciona una conexión causal la aplicación subyacente, ejecutada por un protocolo de meta-objetos (MOP) que mapea los elementos del modelo en la aplicación, de esta manera, es posible aplicar los cambios al modelo y que se reflejen en la aplicación en tiempo de ejecución, y viceversa. A través de esta técnica también se puede planificar acciones a ejecutar sobre las aplicaciones.

20. **TC38-Decisiones con base en flujos de actividad**: Las acciones a ejecutar son determinadas por un flujo de proceso previamente definido. En [S03], la decisión respecto a instanciación o eliminación de contenedores de monitoreo y el tipo de funcionalidades de monitoreo a incluir en dichos contenedores se realiza a partir de un flujo de actividades cuyo iniciador corresponde con la identificación de un cambio en los contenedores de aplicación.

21. **TC41-Recuperación hacia atrás**: El mecanismo de *Backward recovery* consiste en la restauración de un estado previo del sistema tras una caída. En [S01] se combina con un método de bloqueo que impide registrar datos errados en la base de datos usando el sistema CEP.

22. **TC42-Recuperación hacía adelante**: En [S01], el sistema, a través de una interfaz API expuesta por un micro-servicio de tipo *Edge* publica las acciones a ejecutar por parte de los componentes del sistema para llevarlo hacia un nuevo estado, libre de errores cuando se presenta o se predice una posible falla en un estado actual.

23. **TC43-Determinar puntos de sincronización**: Una de las técnicas utilizadas en [S07] para evitar fallos en el sistema durante el proceso de reconfiguración de los agentes de software, consiste en la identificación de puntos de sincronización entre los nodos previo a la ejecución de cambios. Esto se logra en el estudio mencionado en tres pasos que pueden ser ejecutados de manera secuencial o no [62]:

    - **Notificar y esperar**: Se crean y envían mensajes de notificación a los nodos que contengan identificación del punto de sincronización y un número de ciclo de continuación factible en el cual sea posible comenzar el siguiente bloque de pasos para la reconfiguración. Tras el envío de la notificación, el agente espera por un ciclo.
    - **Esperar por sincronización**: El agente espera una notificación asociada con un determinado de punto de sincronización. Al recibirlo, el agente espera por un ciclo.
    - **Esperar por un ciclo**: El agente espera cierto ciclo absoluto y factible para avanzar.

La relación entre tareas se puede observar en las figuras 4.3 y 4.4. Las técnicas pueden seleccionarse a partir de la tarea que se va a ejecutar en cada una de las fases del flujo de retroalimentación MAPE-K. Sin embargo, no todas las tareas tienen necesariamente relacionada una estrategia de ejecución. En algunos escenarios particulares una técnica perteneciente a una fase puede soportar la ejecución de una o más tareas de diferentes fases, este es el caso de la técnica TC21-Procesamiento de eventos complejos, estrategia de la fase de análisis, a través de la cual pueden lograrse las actividades de evaluación de errores tanto en la misma fase de análisis como en la de planeación (TA21 - TA31).

| | | TÉCNICAS | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | TC11 | TC12 | TC13 | TC14 | TC15 | TC16 | TC17 | TC21 | TC22 | TC23 | TC24 | TC25 |
| TAREAS | TA11 | ■ | ■ | ■ | | | | | | | | | |
| | TA12 | ■ | ■ | ■ | | | | | | | | | |
| | TA13 | | | ■ | ■ | ■ | ■ | ■ | | | | | |
| | TA14 | | | | | | | ■ | | | | | |
| | TA15 | | | | ■ | ■ | | | | | | | |
| | TA21 | | | | | | | | ■ | | | | |
| | TA22 | | | | | | | | | ■ | | ■ | |
| | TA23 | | | | | | | | | | ■ | | |
| | TA24 | | | | | | | | | | | | ■ |
| | TA31 | | | | | | | | ■ | ■ | | | |

Figura 4.3: Tareas Vs. Técnicas Monitoreo - Análisis

| | | TÉCNICAS | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | TC21 | TC22 | TC31 | TC32 | TC33 | TC34 | TC35 | TC36 | TC37 | TC41 | TC42 | TC43 |
| TAREAS | TA31 | ■ | ■ | | | | | | | | | | |
| | TA32 | | | ■ | ■ | | | | | | | | |
| | TA33 | | | | | | | ■ | ■ | | | | |
| | TA34 | | | | | ■ | | | | | | | |
| | TA35 | | | | | | ■ | | | | | | |
| | TA36 | | | | | | | | | ■ | | | |
| | TA44 | | | | | | | | | | ■ | ■ | ■ |
| | TA45 | | | | | | | | | | | ■ | ■ |

Figura 4.4: Tareas Vs. Técnicas Planificación - Ejecución

### 4.3.3. Base de conocimiento

La base de conocimiento del flujo MAPE-K se encarga de almacenar datos relevantes respecto al comportamiento del sistema, parámetros de configuración, resultados de ejecución de determinadas actividades y en algunos casos decisiones relacionadas con la adaptación. Para esta fase no son definidas tareas o técnicas, pero a continuación se describe la forma en que la gestión del conocimiento es abordada en diferentes estudios:

En [S01] la BD corresponde con un microservicio *back-end* que almacena datos de servicios autorizados, es decir, cuyos datos han sido previamente filtrados para evitar cargar en la BD datos errados que conlleven a predicciones erróneas.

En [S04], la base de conocimiento de servicios distribuidos (DSKB) se encarga de organizar los servicios en múltiples bases de conocimiento de servicios. Los servicios de la misma ontología de dominio se expresan como una base de conocimientos de servicios (SKB). Cada SKB registra la descripción estática, incluida la semántica y detalles de invocación a nivel sintáctico, así como características dinámicas sobre hechos conocidos (individuos y sus relaciones) e instancias de servicio disponibles. Para eliminar los conflictos semánticos entre diferentes SKB, se crean varias reglas puente, que incluyen principalmente reglas de subclase y equivalencia entre conceptos y predicados en diferentes ontologías de dominio [65].

En [S05] se proporciona un repositorio común a todos los micro-servicios en el cual se almacena la información de configuración de los agentes, así como las descripciones de los micro-servicios.

En [S08] se define una base de datos para el almacenamiento de la información relacionada con los objetos virtuales y una para los metadatos relacionados con la gestión, ubicación, estructura y comportamiento del sistema.

[S10] y [S11] utilizan bases de datos para almacenar la información relacionada con los eventos capturados en la fase de monitoreo, esto le permite al analizador realizar posteriores

validaciones sobre los datos para ejecutar tareas de predicción fallos o conocer en detalle el comportamiento del sistema.

### 4.3.4. Herramientas para la adaptación

En la figura 4.5 se muestra la incorporación en el *framework* de las herramientas usadas en los estudios primarios y descritas en la sección 3.4, ordenadas en conformidad con la fase del flujo de retroalimentación que apoyan. Cada herramienta es descrita con la nomenclatura: H(Herramienta), el número de la fase del flujo en las que son usadas dentro del estudio, 1:monitoreo, 2:análisis, 3:planeación, 4:ejecución, 5:Base de conocimiento y el consecutivo de cada herramienta. Algunas de las herramientas apoyan dos o más fases del proceso de adaptación, en esos casos, estarán señaladas con los números que competen a las dos fases y el consecutivo al final, por ejemplo, *Elasticsearch* es una base para almacenamiento de información sobre logs que apoya principalmente la fase de análisis, pues proporciona al analizador los datos requeridos para evaluar el comportamiento del sistema. La herramienta se identifica entonces con los dígitos 2 y 5 correspondientes a las etapas análisis y base de conocimiento, respectivamente y el consecutivo: H251.

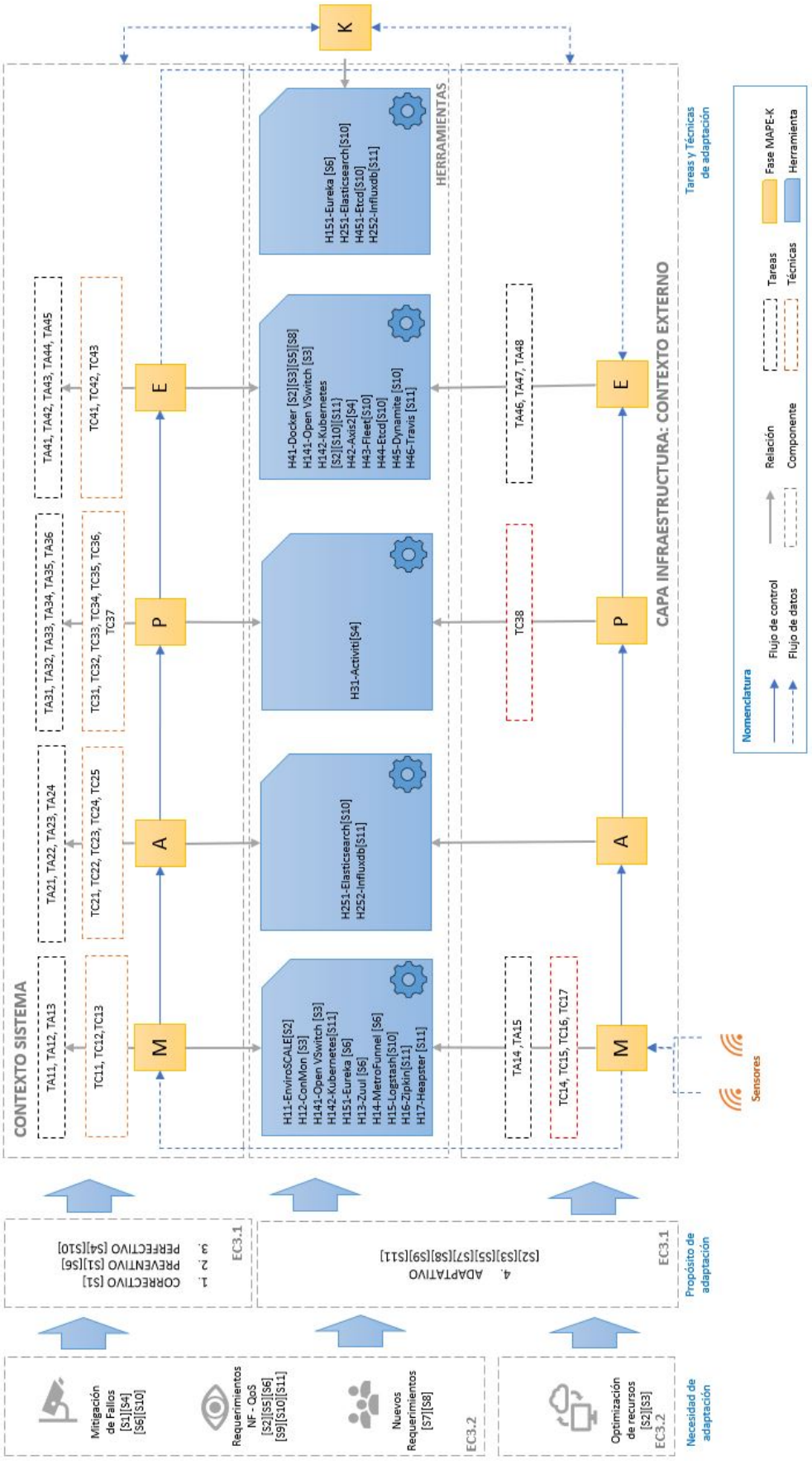


Figura 4.5: *Framework:* Herramientas

## 4.4. Proceso DevOps

DevOps es concebido como un grupo de prácticas en el marco de la ingeniería de software continua que integran los procesos de desarrollo y operación y tienen por objetivo facilitar y agilizar el proceso de transición a un ambiente productivo de la aplicación sin deteriorar la calidad del software en términos de mecanismos de entrega y código [9].

De acuerdo con lo expuesto en [24], las principales actividades en este marco relativas al proceso de desarrollo son:

- Integración continua: Proceso automático que incorpora las principales actividades relacionadas con la compilación, verificación y empaquetamiento del código.
- Entrega continua: Implementación del código en un entorno, no necesariamente a usuarios finales.
- Despliegue continuo: Esta práctica asegura que el software esté continuamente listo para despliegue a clientes finales e incorpora la entrega continua.
- Verificación continua: Actividades de verificación del código y funcionamiento de la aplicación durante todo el proceso de desarrollo.
- Prueba continua: Proceso automático de selección y/o ejecución de casos de prueba sobre la aplicación para la detección de *bugs*.
- Cumplimiento continuo: Esta práctica busca satisfacer los estándares de cumplimiento normativo de manera continua.
- Seguridad continua: Identificación de vulnerabilidades de seguridad durante el ciclo de desarrollo del producto.
- Evolución continua: Adecuaciones realizadas en fase de diseño sobre la arquitectura de la aplicación para asegurar su evolución y escalabilidad en el transcurso del tiempo.

A continuación, se describen las prácticas DevOps asociadas al proceso de operación:

- Uso continuo: Reconoce que la adopción inicial y el uso continuo del software se basan en diferentes parámetros, y que la retención de clientes puede ser una estrategia más eficaz que intentar atraer nuevos clientes.
- Confianza continua: Capacidad de trabajar colaborativamente con el cliente sin necesidad de explotar sus vulnerabilidades expuestas.

- Monitoreo continuo en tiempo de ejecución: Actividad de monitorización en tiempo real de la aplicación para la detección y prevención de fallos, así como el cumplimiento de los acuerdos de nivel de servicio pactados.

Una vez identificadas las actividades DevOps, de acuerdo con su definición, se realizó un mapeo frente a los estudios primarios recopilados en el capítulo 3, llegando así a la relación expuesta en la figura 4.6 entre las prácticas continuas y los estudios que las incorporan.

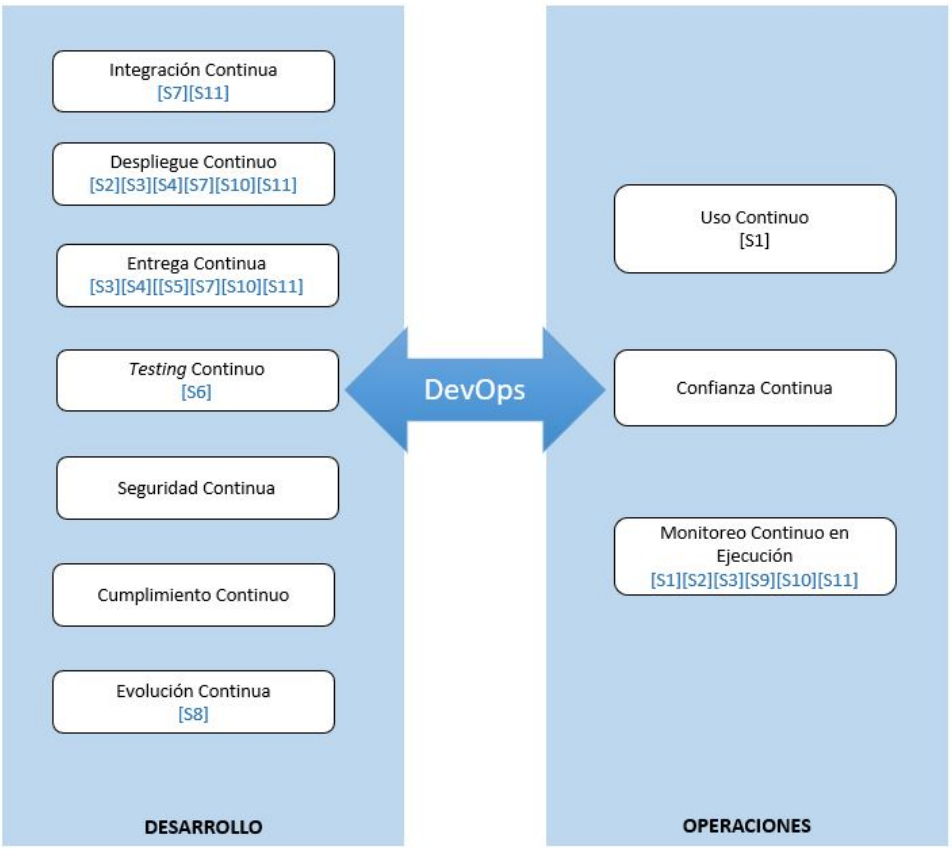


Figura 4.6: Prácticas DevOps. Elaborado según Fitzgerald et al. [24]

Posteriormente se validaron las prácticas frente a la fase de adaptación (criterio de clasificación taxonómica EC2.1) del sistema que soporta, es decir, la etapa en el ciclo de vida de la aplicación en la cual se realiza la adaptación. El resultado obtenido se encuentra representado en la figura 4.7. La actividad de monitoreo continuo en tiempo de ejecución corresponde con la única práctica del proceso de operación que se incorpora en los estudios [S02][S09][S10] y [S11] que comparten la operación como fase de adaptación y [S01][S03], los cuales implementan estrategias de adaptación en fase de monitoreo. La práctica de entrega continua se ve reflejada en la fase de operación de los estudios [S03][S04][S05][S07][S10] y [S11], mientras que la práctica de *testing* continuo tan solo se ejecuta en la fase de operación de [S06]. El despliegue continuo es ejecutado en las propuestas [S02][S03][S04][S07][S10][S11] y la integración continua en [S07] y [S11]. El estudio [S08] que se enfoca en la incorporación de capacidades de adaptación del

sistema en fase de diseño, asegura con esto la evolución continua del sistema. El detalle de las fases de adaptación de cada uno de los estudios se puede apreciar en la sección 3.4.

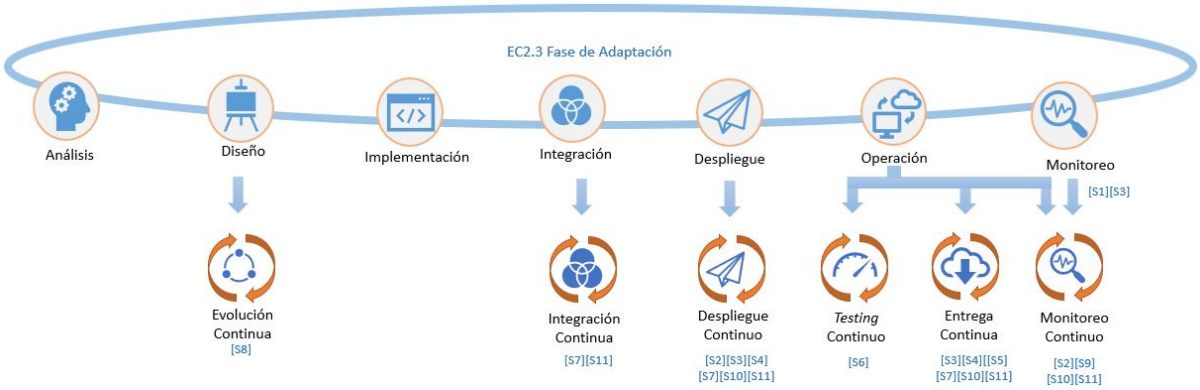


Figura 4.7: Actividades Continuas que soportan las fases de adaptación

A partir del resultado anterior, fueron identificadas las fases del flujo de retroalimentación MAPE-K que se impactan positivamente durante la ejecución actividades de ingeniería continua. La incorporación de las prácticas DevOps al *framework* de adaptación se realiza en conformidad y es expuesto en la figura 4.8.

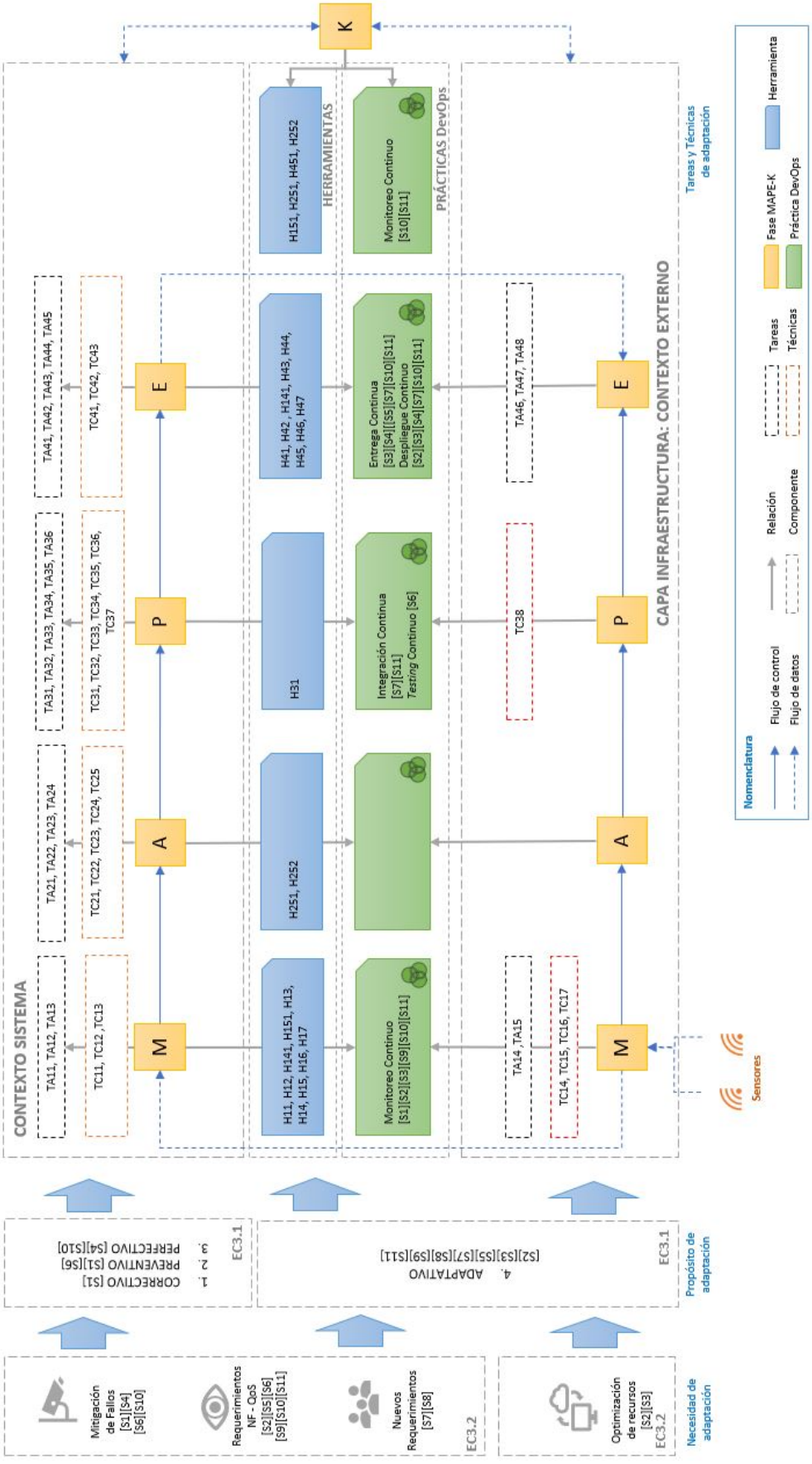


Figura 4.8: *Framework:* DevOps

## 4.5. Aplicación del *Framework* de adaptación

En esta sección se describen los pasos que guían la lectura e implementación del *framework* cuya definición final se puede observar en la figura 4.10. El procedimiento planteado para tal fin consta de seis pasos ilustrados en la figura 4.9. En cada paso se establecen las posibles opciones a considerar para la incorporación de aspectos de adaptabilidad en el diseño de un sistema con arquitectura basada en micro-servicios y desplegado en contenedores, en conformidad con el objetivo de adaptación propuesto para el sistema. Es por esta razón que las actividades iniciales del procedimiento se orientan a la identificación de los motivadores de adaptación, para finalmente definir las tareas, técnicas y herramientas que podrían apoyar la consecución de dicho objetivo, así como el contexto de aplicación (fase de adaptación y actividades DevOps). A continuación, se describen cada uno de los pasos y opciones:


Figura 4.9: Procedimiento para adopción del *framework*

1. **Definir necesidad de adaptación**: El primer paso consiste en la identificación de la necesidad de adaptación, de acuerdo con nuestra taxonomía podrían ser:
   - Mitigación de fallos en tiempo de ejecución.
   - Cumplimiento de requerimientos no funcionales orientados a mantener la calidad del servicio.
   - La incorporación de nuevos requerimientos o mejora de los actuales
   - La optimización de los recursos de *hardware* que alojan la aplicación o los recursos de red que facilitan la comunicación de la aplicación con su contexto externo.
2. **Definir el propósito de adaptación**: De acuerdo con la necesidad se determina el propósito de adaptación de la aplicación, el cual puede ser:
   - Adaptativo en el caso de que la necesidad sea la integración de nuevos requerimientos u optimización de recursos.
   - Correctivo, perfectivo o preventivo si la necesidad de adaptación está relacionada con la mitigación de fallos o calidad del servicio.

3. **Identificar la fase de adaptación**: Se debe establecer claramente la fase del ciclo de vida de la aproximación en la cual se desea incorporar la estrategia de adaptación para cubrir la necesidad inicial.

4. **Seleccionar Actividad Continua**: Una vez identificada la fase de adaptación, siguiendo el mapeo establecido en la sección 2.2 se podrá determinar la o las actividades continuas en el marco DevOps a adoptar.

5. **Seleccionar tarea y técnicas por fase**: El propósito de adaptación nos facilitará la identificación del contexto (sistema o externo) sobre el cual trabajar. A partir de allí, se podrán elegir las tareas y técnicas que mejor se acoplen en cada una de las fases del flujo MAPE para resolver la necesidad de adaptación. La selección de tareas se puede realizar con base en la formulación presentada en 4.3.

6. **Seleccionar herramientas**: Una vez establecidas las tareas a ejecutar, la selección de las herramientas a incorporar consiste en el paso a seguir dentro del diseño del sistema. Las herramientas deben ser seleccionadas considerando las tareas, técnicas y actividad continua a sustentar.

### 4.5.1. Escenario de prueba

Con el ánimo de ejemplificar la implementación del *framework* propuesto, se presenta como escenario de prueba un sistema diseñado para la monitorización de micro-servicios desplegado en contenedores en el *cloud* tomado de [46], cuyo diseño se detalla en la subsección 4.5.1.1, posteriormente es definido un objetivo de adaptación sobre dicha plataforma y mediante la adopción del *framework* de adaptación a través del proceso definido se establecen los artefactos y actividades continuas a incorporar en el diseño del sistema propuesto.

#### 4.5.1.1. Descripción de *M3 System*

El sistema M3 propuesto por Noor et al. en [46], consta de dos componentes principales, a saber, el administrador de monitoreo y el agente de monitoreo. Los agentes de monitoreo son componentes de software que se encuentran ubicados dentro de cada contenedor que rastrea el rendimiento de los microservicios subyacentes, recopilan las estadísticas a nivel del sistema para cada servicio y envían los datos al administrador cuando este los solicita a través de solicitudes HTTP. Los agentes de monitoreo están empaquetados en un archivo *jar* y configurados para ejecutarse durante el proceso de inicio del contenedor. Todos los agentes de monitoreo amplían un agente común, llamado *SmartAgent*, que consta de dos componentes (*SystemAgent* y *ProcessAgent*). *SmartAgent* es un servicio que ejecuta tres operaciones de comunicación con el

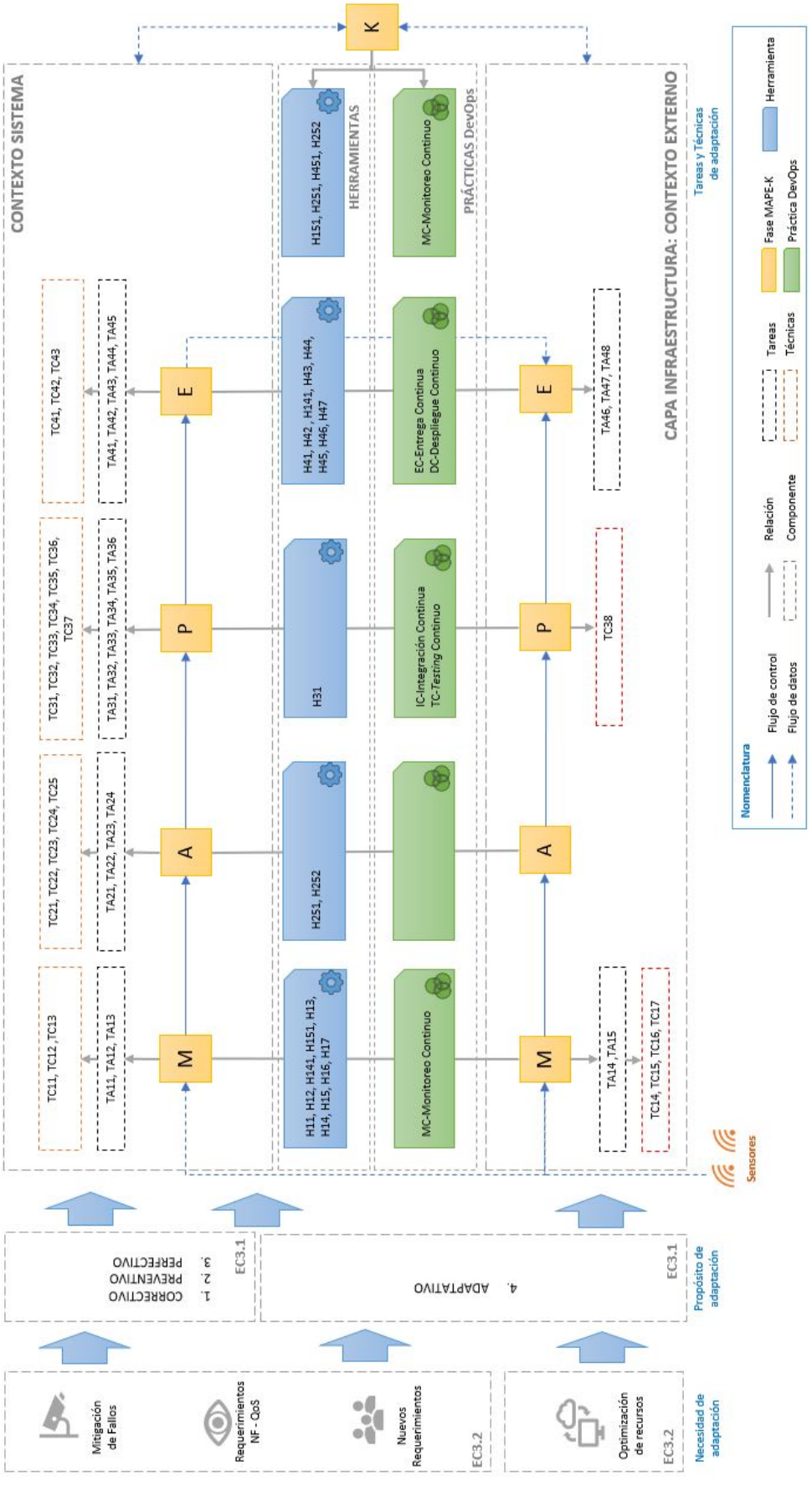


Figura 4.10: *Framework:* Vista General

administrador de monitoreo. SystemAgent monitorea el sistema como un todo, por ejemplo, un contenedor o una máquina virtual, mientras que *ProcessAgent* monitorea el proceso específico que se ejecuta en ese sistema.

El administrador desplegado recopila la información de diferentes agentes de monitoreo y almacena estos datos en la base de datos MySQL asociada. Proporciona una API para acceder a los datos guardados por la base de datos y otros servicios o aplicaciones. La transferencia de información entre el *SmartAgent* de monitoreo y el administrador de monitoreo ocurre usando una secuencia de pasos como se muestra en la figura 4.11: Primero, el *SmartAgent* envía una solicitud de suscripción al administrador, luego, el ejecutor del administrador está habilitado para recibir los datos enviados por el *SmartAgent* y obtiene las métricas. Por último, *SmartAgent* consulta periódicamente al administrador por su configuración. La configuración dinámica permite la gestión de agentes en tiempo real.

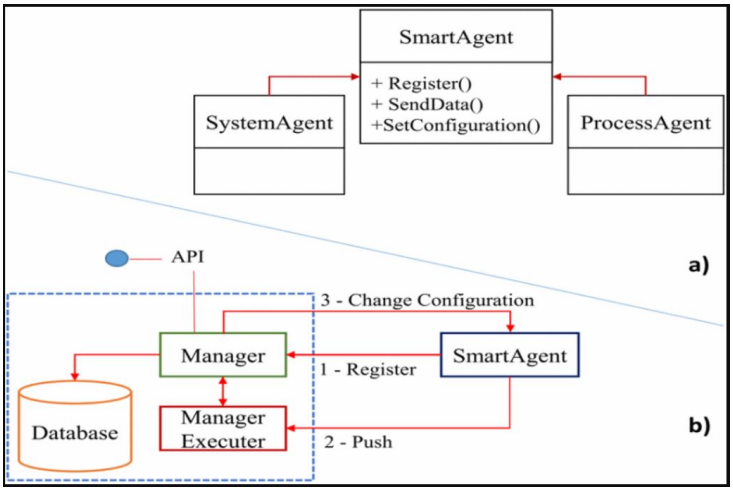


Figura 4.11: Modelo de agentes de seguimiento (a); modelo de comunicación entre administrador y agentes (b). Tomado de [46]

El proceso completo de monitoreo de microservicios se muestra en la Figura 4.12. El administrador del sistema inicia el agente de monitoreo y lo registra en el administrador de monitoreo. El agente de monitoreo continuamente recopila datos de los micro-servicios o contenedores para luego enviarla al administrador, quien recibe y almacena los datos en una base de datos compartida.

#### 4.5.1.2. Adaptación en *M3 System*

*M3 System* es una propuesta para el monitoreo en tiempo de ejecución de micro-servicios desplegados en la nube que se encarga de almacenar los datos recopilados en una base centralizada, para extender su funcionalidad, se plantea un objetivo de adaptación dinámica:

*La capacidad de identificar posibles fallos sobre los agentes de monitoreo y recuperar su operativi-*

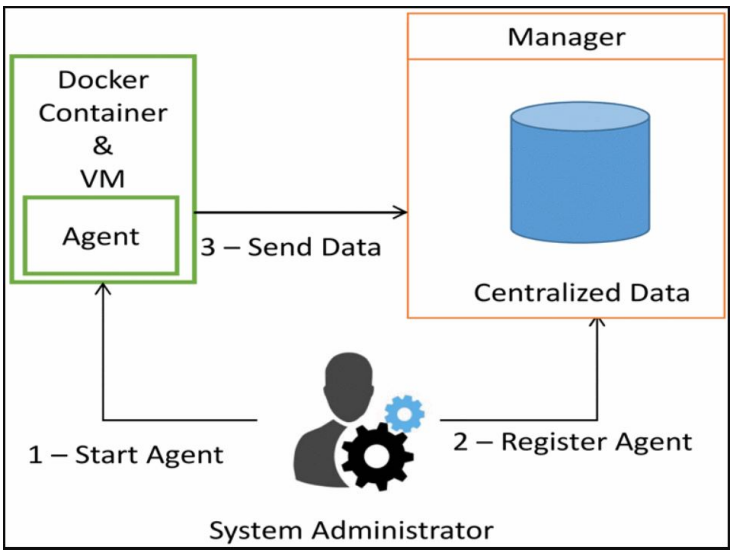


Figura 4.12: Modelo de adquisición de datos. Tomado de [46]

*dad en caso de interrupción del servicio.*

Siguiendo los pasos definidos para la adopción del *framework* previamente definidos, tenemos que:

1. La **necesidad de adaptación** en el caso de ejemplo es la **mitigación de fallos** en tiempo de ejecución.

2. Habiendo definido la necesidad, el **propósito de adaptación** puede ser, de acuerdo con lo expuesto en el *framework*: correctivo, preventivo o perfectivo. Con base en las definiciones realizadas en la sección 3.1.3.4, el propósito, en este caso, puede ser tanto **preventivo** como **correctivo**, ya que se pretende detectar posibles incidentes, pero también, apoyar la recuperación del sistema ante un eventual fallo.

3. Según el objetivo establecido, la adaptación debe ser dinámica, por lo que la fase en la cual el sistema realizará la adaptación será en **operación**.

4. En conformidad con la fase de adaptación identificada y de acuerdo con la formulación expuesta en 4.7 se sugiere la incorporación de actividades de **entrega, *testing* y monitoreo continuo**.

5. La necesidad y propósito de adaptación identificados nos guían hacia la integración de tareas y técnicas en el **contexto de sistema**. Con base en la referencia de tareas planteada en 4.3 y el propósito de adaptación definido. Las tareas que podrían ser adoptadas son:

    - Fase de monitoreo: Estas tareas nos permiten tener una actividad reactiva (identificación de fallas en un componente en tiempo de ejecución) y proactiva (recopilación de datos que permiten prever un posible fallo) simultáneamente. Las dos tareas pueden

ser ejecutadas en paralelo dado que recolectan datos diferentes de las aplicaciones. En nuestro ejemplo, las dos actividades serían llevadas a cabo por el *Smart Agent*:

- **TA11-Detección de eventos**
- **TA12-Captura de propiedades**

- Fase de Análisis: Una vez recopilados los eventos y propiedades en la fase de monitoreo, para cumplir nuestros propósitos de adaptación se establecen dos tareas, en este caso, la primera cubre los dos tipos de actividad esperados (reactiva y proactiva) y la segunda solo el propósito preventivo (proactivo):
  - **TA21-Evaluación de errores**
  - **TA22-Predicción de fallos**
- Fase de planeación:
  - **TA31-Evaluación de errores**: Esta tarea al igual que en la fase de análisis se encargará de la evaluación de los errores identificados para encontrar el mejor camino hacia la recuperación del sistema.
  - **TA35-Generación de casos de prueba**: Con esta tarea se pretende diseñar pro activamente escenarios de prueba a ejecutar sobre los *Smart Agents* para verificar su funcionamiento.
  - **TA36-Validación previa del plan**: Para evitar nuevas oportunidades de fallo se ejecuta una revisión de los escenarios de prueba a ejecutar.
- Fase de ejecución: Para el ejemplo abordado el mecanismo de adaptación será cerrado, es decir, se ejecutarán cambios al sistema sin incorporar nuevas funcionalidades. Para esto se adoptan las dos tareas siguientes:
  - **TA42-Ejecución de acción**: La acción a ejecutar consiste en la aplicación del escenario de prueba definido y verificado en la fase de planeación en las tareas TA35 y TA36, respectivamente.
  - **TA45-Iniciación de componente**: Para asegurar la continuidad de la operación, cuando se identifique un componente error en la fase de análisis, este será instanciado nuevamente en conformidad con la evaluación realizada por el planificador.

Las técnicas por implementar, como se describe en la sección 4.3, pueden ser seleccionadas de acuerdo con las tareas previamente adoptadas en conformidad con la relación establecida en 4.3 para las tareas de monitoreo y análisis, y 4.4 para las tareas de planificación y ejecución. Para lograr la adaptación en nuestro sistema ejemplo se utilizan las siguientes técnicas: Usando la estrategia **TC11-Interacción *Daemon Container*** capturaremos los

eventos (cambios de estado, cantidad de datos recopilados sobre la aplicación, número de invocaciones realizadas por el administrador, número de invocaciones exitosas) y propiedades (estado, capacidad utilizada, tiempo de respuesta) sobre el *Smart Agent*. La técnica **TC12-Registro de eventos históricos** nos presenta una guía para el almacenamiento de los datos recopilados sobre el estado de los agentes de monitoreo, de manera que puedan ser tenidos en cuenta en las actividades de evaluación de errores y predicción de fallos en las etapas posteriores. Para la ejecución de las tareas TA21, TA22 y TA31 se emplean las técnicas **TC21-Procesamiento de eventos complejos** y **TC22-Aprendizaje en línea**. El administrador implementa dos servicios de análisis, uno que utiliza el paradigma de CEP (*Complex Event Processing* para el reconocimiento de errores con base en los eventos reportados por el monitor y otro que implementa una estrategia de *machine learning* para generar conocimiento a partir del histórico de eventos sobre los componentes previamente registrados. La salida de estos dos procesos permitirá al planificador seleccionar la mejor acción de recuperación posible. Para la definición de los escenarios de prueba a ejecutar, con el objetivo de comprobar la hipótesis sobre el futuro estado de los agentes de monitoreo, producto de la actividad de predicción de fallos, se realiza una adaptación de la estrategia **TC34-Evaluación adaptativa de confiabilidad**, mediante la cual, se evalúa el histórico de eventos relacionados con la demanda del administrador de monitoreo hacía el agente de monitoreo. Previo a la ejecución de las acciones definidas por el planificador, se realiza una validación de estas mediante la implementación de un modelo en tiempo de ejecución, según lo sugerido en **TC37-Modelos en tiempo de ejecución**. Una de las acciones a ejecutar para recuperar la operación de un *Smart Agent* sin respuesta consiste en la iniciación de un nuevo componente de monitoreo en el contenedor que aloja el agente en problemas siguiendo la estrategia de **TC42-recuperación hacía adelante**. La definición de otras posibles acciones a efectuar no es del alcance de este *Framework*.

6. La correcta selección de las herramientas dependerá del conocimiento técnico de quien emplea el *Framework*, sin embargo, con base en las tareas y actividades del proceso DevOps adoptadas (entrega, *testing* y monitoreo continuo) en los pasos 4 y 5 es posible identificar herramientas que apoyen la implementación de las tareas seleccionadas. Para nuestro ejemplo, nos centraremos en las herramientas que apoyan las tareas y técnicas ubicadas en las fases de monitoreo, planeación y ejecución, dada su relación con las actividades DevOps ya mencionadas:

    - Para la detección y posterior registro de los eventos sobre los agentes de monitoreo se propone el uso de *H15-Logstash*, por ser la única herramienta identificada para dicho propósito en la revisión sistémica. Si bien esta herramienta es altamente

aceptada gracias a su flexibilidad y variedad de conectores, existen otras opciones similares en el mercado como *Filebeat* o *Logagent*.

- Se planeta el uso de *H45-Dynamite* para la orquestación e instanciación de los agentes de monitoreo desde el administrador. En varios de los estudios primarios se utilizan herramientas para el despliegue y orquestación de contenedores, como se puede apreciar en la tabla 3.17, sin embargo, seleccionamos *Dynamite* puntualmente por su capacidad de orquestación para el escalado independiente de cada micro-servicio o componente a través de un archivo de configuración [63].
- La base de conocimiento corresponde en parte con el registro del histórico de eventos recuperado, para cuyo almacenamiento se sugiere el uso de *H251-Elasticsearch* o *Influxdb* puesto que ambas están diseñadas para mantener el registro de ejecución de una aplicación.

De acuerdo con lo establecido anteriormente en la Figura 4.14 se expone una vista del *framework* de adaptación en la que se pueden identificar los artefactos seleccionados y el paso del proceso de adopción (Pasos 1 al 2 y 4 al 6). La ejecución del paso 3 se puede identificar claramente en la Figura 4.13.

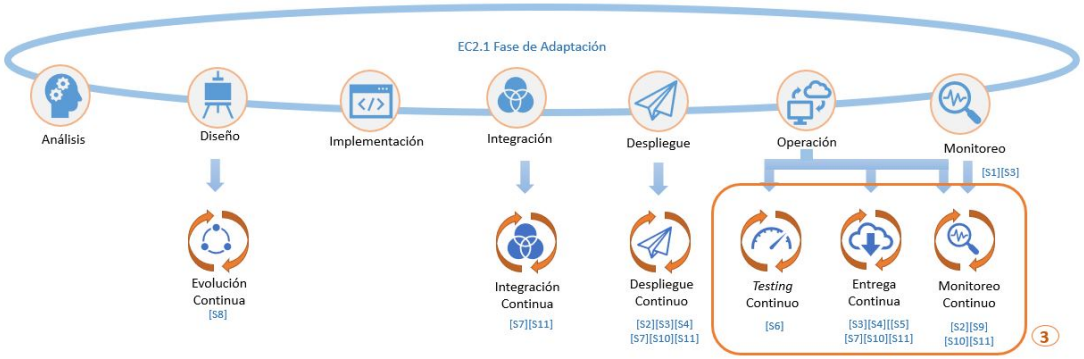


Figura 4.13: Adopción actividades continuas. Paso 3

La incorporación de las capacidades de adaptación sobre el sistema de referencia se puede evidenciar en la Figura 4.15. Los componentes de administración y agentes de monitoreo de la propuesta original se mantienen. Son integrados al administrador: (1) Un monitor que se consolida la información relacionada con el comportamiento y características de los agentes, también se encarga del registro de históricos en la base de conocimiento. (2) Un conjunto de servicios de análisis que tras recibir los datos del monitor se encargan de predecir fallos y evaluación de los errores presentados. El resultado del análisis en compartido con el planificador, este servicio es responsable de la selección de las acciones de remediación, generación de los escenarios de prueba y validación del plan previo a la ejecución. Por último, el ejecutor recibe del planificador las acciones validadas y se encarga de ejecutarlas dentro de cada uno de los

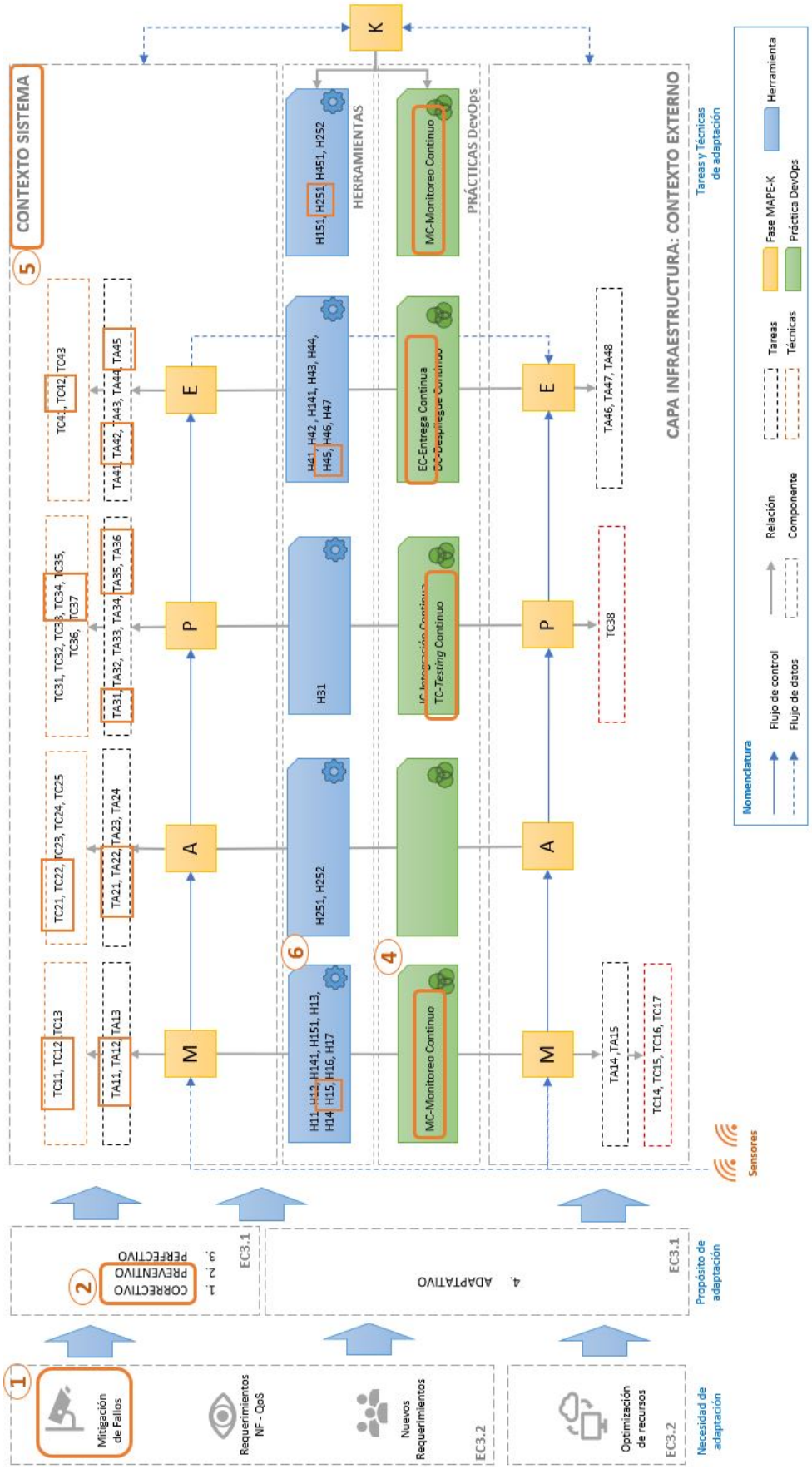


Figura 4.14: *Framework:* Adopción M3 System. Pasos 1-2, 4-6

contenedores, las acciones pueden ser desde la implementación de un caso de prueba hasta la instanciación de un nuevo agente.

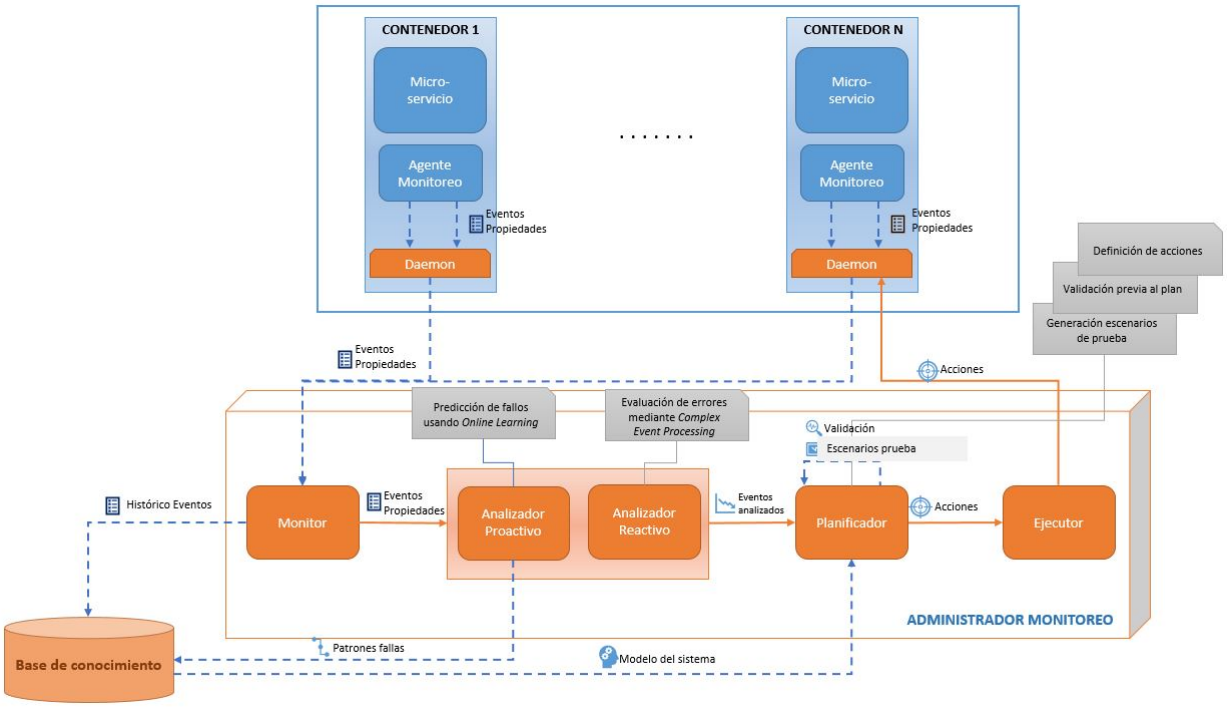


Figura 4.15: Diagrama sistema M3 con capacidades de adaptación

# Capítulo 5

# Conclusiones

## 5.1. Conclusiones

Mediante los componentes y relaciones presentados en el capítulo 4 y respaldados por el análisis sistemático de la literatura presentado a lo largo del documento, se definieron los artefactos considerados necesarios para incorporar una estrategia de adaptación sobre el diseño de sistemas de software regidos bajo principios de micro-servicios y desplegados en contenedores, cumpliendo el objetivo general del proyecto.

A pesar de no haber podido corroborar las capacidades del *framework* resultante mediante su aplicación a un caso de estudio más complejo, a partir de un ejemplo fundamentado en un sistema propuesto por un tercero se demostró el potencial de los mecanismos y herramientas propuestas en la introducción de capacidades de adaptación sobre un sistema dado.

El framework propuesto también pone de manifiesto varias carencias en el estado del arte en la adaptación de sistemas que siguen el enfoque de microservicios y contenedores en procesos DevOps, y la necesidad de realizar futuras investigaciones en varias áreas, para contestar las siguientes preguntas de investigación, entre otras:

- (Dev) ¿Qué soluciones arquitecturales ofrecen el más alto grado de flexibilidad para facilitar la evolución continua de sistemas cloud?
- (Dev) ¿Como se puede gestionar la “seguridad continua” y el “cumplimiento continuo” de microservicios desplegados en contenedores?
- (Ops) ¿Qué mecanismos son efectivos para medir el “uso continuo” y la “confianza continua” de microservicios desplegados en contenedores?
- ¿Qué tarea de adaptación (ver tabla 4.2) es más efectiva en qué contexto y entorno de adaptación?

- ¿Qué técnica de adaptación (ver tabla 4.4) es más efectiva en qué contexto y entorno de adaptación?
- ¿Cuáles son las barreras clave para el desarrollo y operaciones de microservicios desplegados en contenedores y cómo estas barreras se pueden remover?

Respecto a los objetivos específicos, se puede concluir lo siguiente:

1. Mediante una revisión estructurada de la literatura se identificó el contexto de aplicación y las principales tecnologías orientadas a la adaptabilidad en procesos DevOps, de acuerdo con los parámetros de este proyecto. Su revisión motivó el planteamiento de la propuesta principal: un *framework* de adaptación para sistemas en el *cloud* basados en una arquitectura de micro-servicios y contenedores.

2. Como resultado del proceso de revisión sistemática de la literatura sobre estudios primarios enfocados a la adaptabilidad en sistemas software basados en micro-servicios y/o desplegados en contenedores, se propuso un *framework* para la clasificación taxonómica de los principales atributos de adaptabilidad identificados. Si bien, la información que se encuentra en la literatura relacionada a la adaptación en las diferentes fases del ciclo de vida del sistema es extensa (como se puede apreciar en los resultados del capítulo 3), a la fecha en que se ejecutaron las búsquedas en librerías digitales, los estudios específicos sobre adaptación en entornos de micro-servicios y contenedores es reducida. La taxonomía definida presenta una propuesta inicial en materia de clasificación de los principales aspectos de adaptación para este ámbito.

3. Se formuló un *framework* de adaptación, fundamentado en los hallazgos de la revisión sistemática de la literatura y un procedimiento de adopción que permite identificar las acciones, propósito y estrategias de adaptación que se pueden incorporar al diseño de sistemas de software adaptativos basados en arquitecturas de micro-servicios y contenedores. Este *framework* sirve como hoja de ruta para arquitectos y diseñadores de este tipo de sistemas. Incorpora una estrategia para la selección de las actividades del proceso DevOps aplicables en conformidad con la fase y propósito de adaptación del sistema. La adopción del *framework* y la selección de las actividades DevOps se realiza siguiendo un procedimiento de seis pasos.

4. Se presenta como ejemplo de implementación del *framework*, la integración de capacidades y cualidades adaptativas al diseño del sistema "M3 System", diseñado por Noor et al. [46] bajo los paradigmas de micro-servicios y contenedores en la nube. El resultado obtenido es satisfactorio desde la perspectiva de transparencia y facilidad en el entendi-

miento del *framework*, obteniendo un ejemplo válido de implementación que servirá de guía ante trabajos e implementaciones futuras.

## 5.2. Trabajo futuro

Para lograr el *framework* de adaptación propuesto, creemos que los conceptos presentados en la descripción holística del *framework* deben operacionalizarse y estudiarse en casos de adaptación específicos. Como se mencionó en la sección 1.5, el presente trabajo aborda las primeras cuatro actividades del método de investigación para transferencia de conocimiento a la industria. Un ejercicio por realizar a futuro consistirá en la validación del *framework* de adaptación mediante un caso de estudio en un entorno realista y de esta forma completar el ciclo de evaluación y liberación de la propuesta candidata, conforme a lo establecido en método de investigación utilizado. El número de estudios relacionados con los temas de este proyecto demuestra un creciente interés por parte de la comunidad académica, por lo cual se hace deseable escalar el *framework* de adaptación mediante la incorporación de elementos emergentes en el ámbito de la adaptabilidad en entornos *cloud* en sistemas con enfoque en micro-servicios.

# Bibliografía


[1] " Cloud native Computing Foundation. Heapster 1.5.4 for Kubernetes — Helm Hub — Monocular" [Online]. Available: https://hub.helm.sh/charts/stable/heapster. [Accessed Mar, 2020].

[2] "Open Source Business Automation — Activiti" [Online]. Available: https://www.activiti.org/. [Accessed Apr, 2020].

[3] "Open vSwitch" [Online]. Available: https://www.openvswitch.org/. [Accessed Apr, 2020].

[4] "Orquestación de contenedores para producción - Kubernetes" [Online]. Available: https://kubernetes.io/es/. [Accessed Jul, 2020].

[5] "Travis CI - Test and Deploy Your Code with Confidence" [Online]. Available: https://travis-ci.org/. [Accessed Jun, 2020].

[6] ANDIAPPAN, V., AND WAN, Y. K. Distinguishing approach, methodology, method, procedure and technique in process systems engineering. *Clean Technologies and Environmental Policy 22*, 3 (4 2020), 547–555.

[7] APACHE. "Apache Axis2 – Apache Axis2/Java - Next Generation Web Services" [Online]. Available: http://axis.apache.org/axis2/java/core/. [Accessed Mar, 2020].

[8] ARCAINI, P., RICCOBENE, E., AND SCANDURRA, P. Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation. In *Proceedings - 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015* (8 2015), Institute of Electrical and Electronics Engineers Inc., pp. 13–23.

[9] BALALAIE, A., HEYDARNOORI, A., AND JAMSHIDI, P. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture, 5 2016.

[10] BARNA, C., KHAZAEI, H., FOKAEFS, M., AND LITOIU, M. Delivering Elastic Containerized Cloud Applications to Enable DevOps. In *Proceedings - 2017 IEEE/ACM 12th Inter-*

*national Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2017* (7 2017), Institute of Electrical and Electronics Engineers Inc., pp. 65–75.

[11] BEIGI, N., LITOIU, M., EMAMI-TABA, M., BEIGI-MOHAMMADI, N., TAHVILDARI, L., FOKAEFS, M., MERLO, E., AND ONUT, I. V. A DevOps Framework for Quality-Driven Self-Protection in Web Software Systems A DevOps Framework for Quality-Driven Self-Protection in Web Software Systems A DevOps Framework for Quality-Driven Self-Protection in Web Software. *Systems* (2018).

[12] BOGNER, J., WAGNER, S., AND ZIMMERMANN, A. Using architectural modifiability tactics to examine evolution qualities of Service- and Microservice-Based Systems: An approach based on principles and patterns. In *Software-Intensive Cyber-Physical Systems* (6 2019), vol. 34, Springer, pp. 141–149.

[13] BRUNEO, D., FRITZ, T., KEIDAR-BARNER, S., LEITNER, P., LONGO, F., MARQUEZAN, C., METZGER, A., POHL, K., PULIAFITO, A., RAZ, D., ROTH, A., SALANT, E., SEGALL, I., VILLARI, M., WOLFSTHAL, Y., AND WOODS, C. CloudWave: Where adaptive cloud management meets DevOps. In *Proceedings - International Symposium on Computers and Communications* (9 2014), vol. Workshops, Institute of Electrical and Electronics Engineers Inc.

[14] BUCKLEY, J., MENS, T., ZENGER, M., RASHID, A., AND KNIESEL, G. Towards a taxonomy of software change. In *Journal of Software Maintenance and Evolution* (9 2005), vol. 17, pp. 309–332.

[15] CITO, J., LEITNER, P., FRITZ, T., AND GALL, H. C. The making of cloud applications: An empirical study on software development for the cloud. In *2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings* (New York, New York, USA, 8 2015), Association for Computing Machinery, Inc, pp. 393–403.

[16] CRUZES, D. S., AND DYBÅ, T. Recommended steps for thematic synthesis in software engineering. In *International Symposium on Empirical Software Engineering and Measurement* (2011), pp. 275–284.

[17] DE LAURETIS, L. From monolithic architecture to microservices architecture. In *Proceedings - 2019 IEEE 30th International Symposium on Software Reliability Engineering Workshops, ISSREW 2019* (10 2019), Institute of Electrical and Electronics Engineers Inc., pp. 93–96.

[18] DE SANCTIS, M., BUCCHIARONE, A., AND TRUBIANI, C. A DevOps Perspective for QoS-Aware Adaptive Applications. In *Lecture Notes in Computer Science (including subseries*

*Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (5 2020), vol. 12055 LNCS, Springer, pp. 95–111.

[19] DOCKER. "Empowering App Development for Developers — Docker" [Online]. Available: https://www.docker.com/. [Accessed Dec, 2019].

[20] DRAGONI, N., GIALLORENZO, S., LAFUENTE, A. L., MAZZARA, M., MONTESI, F., MUSTAFIN, R., AND SAFINA, L. Microservices: Yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*. Springer International Publishing, 11 2017, pp. 195–216.

[21] DUA, R., RAJA, A. R., AND KAKADIA, D. Virtualization vs containerization to support PaaS. In *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014* (9 2014), Institute of Electrical and Electronics Engineers Inc., pp. 610–614.

[22] ETCD. "etcd documentation, The Linux Foundation" [Online]. Available: https://etcd.io/. [Accessed Dec, 2019].

[23] FERNANDEZ, D. "Arquitecturas basadas en microservicios: Spring Cloud Netflix Eureka - BI Geek Blog" [Online]. Available: https://blog.bi-geek.com/arquitecturas-spring-cloud-netflix-eureka/. [Accessed Marz, 2020].

[24] FITZGERALD, B., AND STOL, K. J. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software 123* (1 2017), 176–189.

[25] FLORIO, L., AND DI NITTO, E. Gru: An approach to introduce decentralized autonomic behavior in microservices architectures. In *Proceedings - 2016 IEEE International Conference on Autonomic Computing, ICAC 2016* (9 2016), Institute of Electrical and Electronics Engineers Inc., pp. 357–362.

[26] FOKAEFS, M., ROUF, Y., BARNA, C., AND LITOIU, M. Evaluating Adaptation Methods for Cloud Applications: An Empirical Study. In *IEEE International Conference on Cloud Computing, CLOUD* (9 2017), vol. 2017-June, IEEE Computer Society, pp. 632–639.

[27] GORSCHEK, T., GARRE, P., LARSSON, S., AND WOHLIN, C. A model for technology transfer in practice. *IEEE Software 23*, 6 (11 2006), 88–95.

[28] GRAY, J., KLEFSTAD, R., AND MERNIK, M. Adaptive and evolvable software systems: techniques, tools, and applications. Institute of Electrical and Electronics Engineers (IEEE), p. 1 pp.

[29] HUANG, X., ZHANG, H., ZHOU, X., BABAR, M. A., AND YANG, S. Synthesizing qualitative research in software engineering: A critical review. In *Proceedings - International Conference on Software Engineering* (5 2018), IEEE Computer Society, pp. 1207–1218.

[30] IBM CORP. An architectural blueprint for autonomic computing. Tech. rep., USA, 2005.

[31] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS., IEEE COMPUTER SOCIETY. SOFTWARE ENGINEERING STANDARDS SUBCOMMITTEE., AND IEEE STANDARDS BOARD. *IEEE Standard for developing software life cycle processes*. Institute of Electrical and Electronics Engineers, 1992.

[32] JAMSHIDI, P., AHMAD, A., AND PAHL, C. Cloud Migration Research: A Systematic Review. *IEEE Transactions on Cloud Computing 1*, 2 (7 2013), 142–157.

[33] JAMSHIDI, P., GHAFARI, M., AHMAD, A., AND PAHL, C. A framework for classifying and comparing architecture-centric software evolution research. In *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR* (2013).

[34] KELL, S. A survey of practical software adaptation techniques, 2008.

[35] KEPHART, J. O., AND CHESS, D. M. The vision of autonomic computing. *Computer 36*, 1 (1 2003).

[36] KHAZAEI, H., BARNA, C., BEIGI-MOHAMMADI, N., AND LITOIU, M. Efficiency analysis of provisioning microservices. In *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom* (7 2016), vol. 0, IEEE Computer Society, pp. 261–268.

[37] KITCHENHAM, B., CHARTERS, S., BUDGEN, D., BRERETON, P., TURNER, M., AND LINKMAN, S. Guidelines for performing Systematic Literature Reviews in Software Engineering. Tech. rep., Keele University, Durham, 2007.

[38] KOSKINEN, M., MIKKONEN, T., AND ABRAHAMSSON, P. Containers in Software Development: A Systematic Mapping Study. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (11 2019), vol. 11915 LNCS, Springer, pp. 176–191.

[39] LEHRIG, S., EIKERLING, H., AND BECKER, S. Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics. In *QoSA 2015 - Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, Part of CompArch 2015* (5 2015), Association for Computing Machinery, Inc, pp. 83–92.

[40] LEWIS, J., AND FOWLER, M. Microservices Guide, 8 2019.

[41] LIU, P., MAO, X., ZHANG, S., AND HOU, F. Towards Reference Architecture for a Multi-layer Controlled Self-adaptive Microservice System NSFC project View project Self-adaptive microservice syetem View project Towards Reference Architecture for a Multi-layer Controlled Self-adaptive Microservice System.

[42] MACÍAS-ESCRIVÁ, F. D., HABER, R., DEL TORO, R., AND HERNANDEZ, V. Self-adaptive systems: A survey of current approaches, research challenges and applications, 12 2013.

[43] MASTRANGELO, C. "Netflix/zuul Wiki" [Online]. Available: https://github.com/Netflix/zuul/wiki. [Accessed Mar, 2020].

[44] MORADI, F., FLINTA, C., JOHNSSON, A., AND MEIROSU, C. ConMon: An automated container based network performance monitoring system. In *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management* (7 2017), Institute of Electrical and Electronics Engineers Inc., pp. 54–62.

[45] NING, D., WANG, Y., GUO, J., AND ZHANG, X. A Common Service Middleware for Intelligent Complex Software System. Institute of Electrical and Electronics Engineers (IEEE), pp. 264–272.

[46] NOOR, A., JHA, D. N., MITRA, K., JAYARAMAN, P. P., SOUZA, A., RANJAN, R., AND DUSTDAR, S. A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments. In *IEEE International Conference on Cloud Computing, CLOUD* (7 2019), vol. 2019-July, IEEE Computer Society, pp. 156–163.

[47] OREIZY, P., GORLICK, M. M., TAYLOR, R. N., HEIMBIGNER, D., JOHNSON, G., MEDVIDOVIC, N., QUILICI, A., ROSENBLUM, D. S., AND WOLF, A. L. An Architecture-Based Approach to Self-Adaptive Software. Tech. rep., 1999.

[48] PAHL, C. Containerization and the PaaS Cloud. *IEEE Cloud Computing 2*, 3 (5 2015), 24–31.

[49] PAHL, C., BROGI, A., SOLDANI, J., AND JAMSHIDI, P. Cloud Container Technologies: a State-of-the-Art Review. *IEEE Transactions on Cloud Computing* (5 2017).

[50] PIETRANTUONO, R., RUSSO, S., AND GUERRIERO, A. Run-Time Reliability Estimation of Microservice Architectures. In *Proceedings - International Symposium on Software Reliability Engineering, ISSRE* (11 2018), vol. 2018-October, IEEE Computer Society, pp. 25–35.

[51] POWER, A., AND KOTONYA, G. A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems. In *19th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2018* (8 2018), Institute of Electrical and Electronics Engineers Inc.

[52] RAHMAN, M., RAHMAN, A., AFRIN, A., HONG, H. J., TSAI, P. H., UDDIN, M. Y. S., VENKATASUBRAMANIAN, N., AND HSU, C. H. Adaptive sensing using internet-of-things with constrained communications. In *Proceedings of the 16th Workshop on Adaptive and Reflective Middleware, ARM 2017* (12 2017), Association for Computing Machinery, Inc.

[53] RODRÍGUEZ, P., HAGHIGHATKHAH, A., LWAKATARE, L. E., TEPPOLA, S., SUOMALAINEN, T., ESKELI, J., KARVONEN, T., KUVAJA, P., VERNER, J. M., AND OIVO, M. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software 123* (1 2017), 263–291.

[54] SAMPAIO, A. R., RUBIN, J., BESCHASTNIKH, I., AND ROSA, N. S. Improving microservice-based applications with runtime placement adaptation. *Journal of Internet Services and Applications 10*, 1 (12 2019).

[55] SHAHIN, M., ALI BABAR, M., AND ZHU, L. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices, 2017.

[56] SHARMA, S., AND COYNE, B. *DevOps for Dummies*, 2nd ed. John Wiley & Sons, Inc., Hoboken, 2015.

[57] SHAW, M. Writing good software engineering research papers. Institute of Electrical and Electronics Engineers (IEEE), pp. 726–736.

[58] SHEVTSOV, S., BEREKMERI, M., WEYNS, D., AND MAGGIO, M. Control-theoretical software adaptation: A systematic literature review, 8 2018.

[59] SHEVTSOV, S., BEREKMERI, M., WEYNS, D., AND MAGGIO, M. Control-theoretical software adaptation: A systematic literature review, 8 2018.

[60] SINGH, V., AND PEDDOJU, S. K. Container-based microservice architecture for cloud applications. In *Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2017* (12 2017), vol. 2017-January, Institute of Electrical and Electronics Engineers Inc., pp. 847–852.

[61] TAIBI, D., LENARDUZZI, V., AND PAHL, C. Continuous architecting with microservices and DevOps: A systematic mapping study. In *Communications in Computer and Information Science* (3 2019), vol. 1073, Springer Verlag, pp. 126–151.

[62] TELSCHIG, K., AND KNAPP, A. Synchronous reconfiguration of distributed embedded applications during operation. In *Proceedings - 2019 IEEE International Conference on Software Architecture, ICSA 2019* (4 2019), Institute of Electrical and Electronics Engineers Inc., pp. 121–130.

[63] TOFFETTI, G., BRUNNER, S., BLÖCHLINGER, M., SPILLNER, J., AND BOHNERT, T. M. Self-managing cloud-native applications: Design, implementation, and experience. *Future Generation Computer Systems 72* (7 2017), 165–179.

[64] UNTERKALMSTEINER, M., GORSCHEK, T., ISLAM, A. K., CHENG, C. K., PERMADI, R. B., AND FELDT, R. Evaluation and measurement of software process improvement-A systematic literature review, 2012.

[65] WANG, X., FENG, Z., AND HUANG, K. D3L-Based Service Runtime Self-Adaptation Using Replanning. *IEEE Access 6* (2 2018), 14974–14995.

[66] YALE, Y., SILVEIRA, H., AND SUNDARAM, M. A microservice based reference architecture model in the context of enterprise architecture. In *Proceedings of 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference, IMCEC 2016* (2 2017), Institute of Electrical and Electronics Engineers Inc., pp. 1856–1860.

[67] ZHANG, S., ZHANG, M., NI, L., AND LIU, P. A multi-level self-adaptation approach for microservice systems. In *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analytics, ICCCBDA 2019* (4 2019), Institute of Electrical and Electronics Engineers Inc., pp. 498–502.

[68] ZIPKIN. "OpenZipkin - A distributed tracing system" [Online]. Available: https://zipkin.io/. [Accessed Apr, 2020].

# Apéndice A

# Estudios primarios seleccionados

A continuación se encuentra listado de los estudios primarios resultantes del proceso de selección.

S01 Alexander Power, Gerald Kotonya. A Microservices Architecture for Reactive and Proactive FaultTolerance in IoT Systems, 2018

S02 Mahmudur Rahman, Hua-Jun Hong, Amatur Rahman, Pei-Hsuan Tsai, Afia Afrin, Md Yusuf Sarwar U., Nalini V., Cheng-Hsin Hsu. Adaptive sensing using internet-of-things with constrained communications, 2017

S03 Farnaz Moradi, Christofer Flinta, Andreas Johnsson, Catalin Meirosu. ConMon: An automated containerbased network performancemonitoring system, 2017

S04 Xianghui Wang, Zhiyong Feng, Keman Huang. D3L-Based Service Runtime Self-Adaptation Using Replanning, 2018

S05 Luca Florio, Elisabetta Di Nitto. Gru: An Approach to Introduce Decentralized Autonomic Behavior in Microservices Architectures, 2016

S06 Roberto Pietrantuono, Stefano Russo, Antonio Guerriero. Run-Time Reliability Estimation of Microservice Architectures, 2018

S07 Kilian Telschig, Alexander Knapp. Synchronous Reconfiguration of Distributed Embedded Applications During Operation, 2019

S08 Dejun Ning, Yu Wang, Jiacheng Guo, Xuanming Zhang. A Common Service Middleware for Intelligent Complex Software System, 2019

S09 Sara Hassan, Nour Ali, Rami Bahsoon. Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity, 2017

S10 Giovanni Toffetti, Sandro Brunner, Martin Blöchlinger, Josef Spillner, Thomas Michael B. Self-managing cloud-native applications: Design, implementation and experience, 2017

S11 Adalberto R. Sampaio Jr., Julia Rubin, Ivan Beschastnikh, Nelson S. Improving microservice-based applications with runtime placement adaptation, 2019

# Apéndice B

# Extracción de datos

En las figuras contenidas en el presente anexo, se muestran las tablas de extracción de datos que fueron utilizadas para la clasificación de los estudios primarios. El formato se estableció en conformidad con la estrategia de extracción de datos y se agrupó posteriormente según los temas definidos para la clasificación taxonómica.

| ID | Artículo | Año | Publisher | Contenedores | Micro-servicios | EC1 Aproximaciones | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | EC.1.1 Tipo de propuesta | | | | | | | | EC1.2 Dominio de aplicación | | | | | | EC1.3 Contexto de uso | |
| | | | | | | Método | Framework | Herramienta | Enfoque/ Aproxim. | Arquitectura | Modelo | Técnica | Otro | Cloud Computing | Sistemas Distribuidos | Sistemas Ubicuos | SOA | IoT | Otro | Académico | Industrial |
| S01 | A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems | 2018 | IEEE | | 1 | | 1 | | | | | | | | | | | 1 | | 1 | |
| S02 | Adaptive sensing using internet-of-things with constrained communications | 2017 | ACM | 1 | | | | 1 | | | | | | | | | | 1 | 1 | 1 | |
| S03 | ConMon: An automated container based network performance monitoring system | 2017 | IEEE | 1 | | | | 1 | | | | | | 1 | | | | | | 1 | |
| S04 | D3L-Based Service Runtime Self-Adaptation Using Replanning | 2018 | IEEE | | 1 | | 1 | | | | | | | | | | 1 | | | 1 | |
| S05 | Gru: An Approach to Introduce Decentralized Autonomic Behavior in MicroservicesArchitectures | 2016 | IEEE | 1 | 1 | | | | 1 | | | | | | 1 | | | | | 1 | |
| S06 | Run-Time Reliability Estimation of Microservice Architectures | 2018 | IEEE | | 1 | 1 | | | | | | | | 1 | | | | | | 1 | |
| S07 | Synchronous Reconfiguration of Distributed Embedded Applications During Operation | 2019 | IEEE | 1 | | | | | 1 | | | | | | 1 | | | | | 1 | |
| S08 | A Common Service Middleware for Intelligent Complex Software System | 2019 | IEEE | 1 | 1 | 1 | | | 1 | | | | | | | | | | 1 | | 1 |
| S09 | Microservice Ambients: An Architectural Meta-Modelling Approach for MicroserviceGranularity | 2017 | IEEE | | 1 | | | | 1 | | | | | | 1 | | | | | 1 | |
| S10 | Self-managing cloud-native applications: Design, implementation, and experience | 2017 | ScienceDirect | | 1 | | | | | 1 | | | | 1 | | | | | | 1 | |
| S11 | Improving microservice-based applications with runtime placement adaptation | 2019 | Springer Link | | 1 | | | | | | 1 | | | 1 | | | | | | 1 | |

Figura B.1: Formato de extracción de datos: EC1

| ID | Artículo | Contenedores | Micro-servicios | EC2 Tipo de Adaptación | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | EC2.1 Fase de adaptación | | | | | | EC2.2 Nivel de Autonomía | | | EC2.3 Tiempo de adaptación | | | EC2.4 Mecanismo de adaptación | | EC2.5 Actividad | | EC2.6 Condiciones de Adaptación | | | |
| | | | | Diseño | Implementación | Integración | Despliegue | Operación | Monitorización | AT | PA | HL | Diseño (Estática) | Tiempo de carga | Ejecución (Dinámica) | Abierta | Cerrada | Reactiva | Proactiva | Self-configuration | Self-optimization | Self-healing | Self-protection |
| S01 | A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems | | 1 | | | | | 1 | 1 | 1 | | | | | 1 | | 1 | 1 | 1 | | | 1 | 1 |
| S02 | Adaptive sensing using internet-of-things with constrained communications | 1 | | | | | 1 | 1 | | 1 | | | | | 1 | | 1 | | 1 | 1 | | | |
| S03 | ConMon: An automated container based network performance monitoring system | 1 | | | | | 1 | 1 | 1 | 1 | | | | | 1 | 1 | | | 1 | 1 | | | |
| S04 | D3L-Based Service Runtime Self-Adaptation Using Replanning | | 1 | | | | 1 | 1 | | 1 | | | | | 1 | 1 | | | 1 | | | 1 | |
| S05 | Gru: An Approach to Introduce Decentralized Autonomic Behavior in MicroservicesArchitectures | 1 | 1 | | | | | 1 | | | 1 | | | | 1 | | 1 | | 1 | | 1 | | |
| S06 | Run-Time Reliability Estimation of Microservice Architectures | | 1 | | | | | | 1 | | 1 | | | 1 | | | 1 | 1 | | | | | 1 |
| S07 | Synchronous Reconfiguration of Distributed Embedded Applications During Operation | 1 | | | | | 1 | 1 | | 1 | | | | | 1 | 1 | | | 1 | 1 | | | |
| S08 | A Common Service Middleware for Intelligent Complex Software System | 1 | 1 | 1 | 1 | | | | | | 1 | | 1 | | | | | | | | | | |
| S09 | Microservice Ambients: An Architectural Meta-Modelling Approach for MicroserviceGranularity | | 1 | | | | | 1 | | 1 | | | | | 1 | | 1 | | 1 | 1 | 1 | | |
| S10 | Self-managing cloud-native applications: Design, implementation, and experience | | 1 | | | | | 1 | | 1 | | | | | 1 | 1 | | | 1 | | | 1 | |
| S11 | Improving microservice-based applications with runtime placement adaptation | | 1 | | | | | 1 | | 1 | | | | | 1 | 1 | | | 1 | | 1 | | |

Figura B.2: Formato de extracción de datos: EC2

| ID | Artículo | Contenedores | Micro-servicios | EC3 Motivadores | | | | | | | | | | | EC4 Técnicas | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | EC3.1 Proposito de Adaptación | | | | EC3.2 Necesidades de Adaptación | | | | | | | EC4.1 Frecuencia de adaptación | | | | EC4.2 Lógica de adaptación | | | | | |
| | | | | Correctivo | Perfectivo | Adaptativo | Preventivo | SLA | QoS | QoE | FT | FD | NR | OR | Otro | Continua | Periodica | Arbitraria | Probabilistic-based | Rule-based | Case-based | Logic-based | Ontology-based | Evidence-based | Fuzzy-based |
| S01 | A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems | | 1 | 1 | | | 1 | | | | 1 | | | | | 1 | | | 1 | | | | | | |
| S02 | Adaptive sensing using internet-of-things with constrained communications | 1 | | | | 1 | | | 1 | | | | | 1 | | | | 1 | | | | 1 | | | |
| S03 | ConMon: An automated container based network performance monitoring system | 1 | | | | 1 | | | | | | | | 1 | | 1 | | | | 1 | | | | | |
| S04 | D3L-Based Service Runtime Self-Adaptation Using Replanning | | 1 | | 1 | | | | | | 1 | | | | | 1 | | | | | | 1 | | | |
| S05 | Gru: An Approach to Introduce Decentralized Autonomic Behavior in MicroservicesArchitectures | 1 | 1 | | | 1 | | | 1 | | | | | | 1 | | 1 | | 1 | 1 | | | | | |
| S06 | Run-Time Reliability Estimation of Microservice Architectures | | 1 | | | | 1 | | 1 | | | 1 | | | | | 1 | | 1 | | | | | | |
| S07 | Synchronous Reconfiguration of Distributed Embedded Applications During Operation | 1 | | | | 1 | | | | | | | 1 | | | 1 | | | | 1 | | | | | |
| S08 | A Common Service Middleware for Intelligent Complex Software System | 1 | 1 | | | 1 | | | | | | | 1 | | | | | 1 | | | | | 1 | | |
| S09 | Microservice Ambients: An Architectural Meta-Modelling Approach for MicroserviceGranularity | | 1 | | | 1 | | | 1 | | | | | | | 1 | | | | 1 | | | | | |
| S10 | Self-managing cloud-native applications: Design, implementation, and experience | | 1 | | 1 | | | | 1 | | 1 | | | | | | 1 | | | 1 | | | | | |
| S11 | Improving microservice-based applications with runtime placement adaptation | | 1 | | | 1 | | | 1 | | | | | | | | 1 | | | 1 | | | | | |

Figura B.3: Formato de extracción de datos: EC3 y EC4

| ID | Artículo | Contenedores | Micro-servicios | EC5 Herramientas | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | EC5.1 Herramientas de soporte | | | | | | | | | | | | | | | | | | | | EC5.2 Protocolo / Estilo Arquitectura | | | | |
| | | | | EnviroScale | ConMon | Docker | Open vSwitch | Activiti | Axis2 | Fleet | Zipkin | Logstash | Influxdb | Elasticsearch | Heapster | Travis | Kubernetes | Etcd | Dynamite | Eureka | Zuul | MetroFunnel | RESTFul | CoAp | MQTT | SOAP | XMPP |
| S01 | A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems | | 1 | | | | | | | | | | | | | | | | | | | | 1 | | | | |
| S02 | Adaptive sensing using internet-of-things with constrained communications | 1 | | 1 | | 1 | | | | | | | | | | | 1 | | | | | | | | 1 | | |
| S03 | ConMon: An automated container based network performance monitoring system | 1 | | | 1 | 1 | 1 | | | | | | | | | | | | | | | | 1 | | | | |
| S04 | D3L-Based Service Runtime Self-Adaptation Using Replanning | | 1 | | | | | 1 | 1 | | | | | | | | | | | | | | 1 | | | 1 | |
| S05 | Gru: An Approach to Introduce Decentralized Autonomic Behavior in MicroservicesArchitectures | 1 | 1 | | | 1 | | | | | | | | | | | | | | | | | 1 | | | | |
| S06 | Run-Time Reliability Estimation of Microservice Architectures | | 1 | | | | | | | | | | | | | | | | | 1 | 1 | 1 | 1 | | | | |
| S07 | Synchronous Reconfiguration of Distributed Embedded Applications During Operation | 1 | | | | | | | | | | | | | | | | | | | | | NE | | | | |
| S08 | A Common Service Middleware for Intelligent Complex Software System | 1 | 1 | | | 1 | | | | | | | | | | | | | | | | | 1 | | | | |
| S09 | Microservice Ambients: An Architectural Meta-Modelling Approach for MicroserviceGranularity | | 1 | | | | | | | | | | | | | | | | | | | | NE | | | | |
| S10 | Self-managing cloud-native applications: Design, implementation, and experience | | 1 | | | | | | | 1 | | 1 | | 1 | | | 1 | 1 | 1 | | | | 1 | | | | |
| S11 | Improving microservice-based applications with runtime placement adaptation | | 1 | | | | | | | | 1 | | 1 | | 1 | 1 | 1 | | | | | | 1 | | | | |

Figura B.4: Formato de extracción de datos: EC5

| ID | Artículo | Contenedores | Micro-servicios | EC6 Modelo de proceso | | | | EC7 Evaluación | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | EC6.1 Modelo de proceso | | | | EC7.1 Método de Evaluación | | | |
| | | | | DevOps | Desarrollo Agil | MDD | Otro | No validada (PoC) | Experimento | Encuesta | CoS |
| S01 | A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems | | 1 | | | | | 1 | | | |
| S02 | Adaptive sensing using internet-of-things with constrained communications | 1 | | | | | 1 | 1 | | | |
| S03 | ConMon: An automated container based network performance monitoring system | 1 | | 1 | | | | | | | 1 |
| S04 | D3L-Based Service Runtime Self-Adaptation Using Replanning | | 1 | | | | 1 | | | | 1 |
| S05 | Gru: An Approach to Introduce Decentralized Autonomic Behavior in MicroservicesArchitectures | 1 | 1 | | | | NE | 1 | | | |
| S06 | Run-Time Reliability Estimation of Microservice Architectures | | 1 | | | | NE | | 1 | | |
| S07 | Synchronous Reconfiguration of Distributed Embedded Applications During Operation | 1 | | | | | NE | 1 | | | |
| S08 | A Common Service Middleware for Intelligent Complex Software System | 1 | 1 | 1 | | | | | | | 1 |
| S09 | Microservice Ambients: An Architectural Meta-Modelling Approach for MicroserviceGranularity | | 1 | | | | NE | | | | 1 |
| S10 | Self-managing cloud-native applications: Design, implementation, and experience | | 1 | | | | NE | | 1 | | |
| S11 | Improving microservice-based applications with runtime placement adaptation | | 1 | 1 | | | | | 1 | | |

Figura B.5: Formato de extracción de datos: EC6 y EC7