



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una arquitectura para la
implantación de Algoritmos como servicio
(AaaS) en entornos de Industria 4.0

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Miguel Ángel Mateo Casalí

Tutor: Andrés Boza García

Director Experimental: Francisco Fraile Gil

Curso: 2020/2021

Agradecimientos

Quiero expresar mi más sincero agradecimiento al Centro de Investigación en Ingeniería Informática por brindarme la oportunidad de evolucionar como investigador dentro de la Universidad Politécnica de Valencia. Así mismo dar las gracias tanto a Andrés Boza cómo Francisco Fraile por guiarme y aportarme los conocimientos necesarios para el desarrollo de este trabajo de final de Máster.

Quiero dar las gracias a Matilde Ruiz por las horas de lectura de este trabajo y las sugerencias de corrección. Por otro lado, a mi prometida, Laura Moya por apoyarme, ayudarme y escucharme hasta largas horas de la noche durante el desarrollo de este trabajo de final de Máster

Finalmente quiero dar las gracias, también, a mis padres y a mi hermana, por confiar en mí.

Resumen

Este trabajo de Final de Master nace de la necesidad de definir una arquitectura a seguir a la hora de implementar un sistema de algoritmos como servicio dentro de la industria 4.0.

El objetivo de este trabajo es proporcionar una arquitectura para ofrecer los algoritmos como servicios utilizando el paradigma de *cloud computing*. Los algoritmos van a ir ganando importancia dentro de la industria 4.0, especialmente debido a su papel dentro del análisis de datos. Un modelo de servicio AaaS se basa en la definición de un algoritmo sobre un proveedor Cloud, y para el usuario final, el beneficio principal es que no tiene que preocuparse de la gestión o el despliegue de la infraestructura necesaria para instanciar o ejecutar el algoritmo. Actualmente, dentro del paradigma del *cloud computing*, no existe este servicio definido como tal y por tanto no cuenta con una arquitectura de referencia. Por ello, en este proyecto se hará un estudio que permita proponer una **arquitectura de referencia** enfocada a un servicio basado en algoritmos.

Para poder desarrollar este servicio dentro de la industria 4.0 se realizará un análisis de las diferentes arquitecturas de referencia que encontramos en la industria y a partir de ese estudio, desarrollar un plan de implementación partiendo de dicha arquitectura de referencia. Deberemos, siempre, tener en cuenta las diferentes perspectivas desde el punto de vista de negocio, aplicación, información y tecnología.

Finalmente, para facilitar el desarrollo del servicio, implementaremos ficheros de integración continua con la idea de automatizar algunas de las tareas entre el desarrollador y el despliegue del servicio.

Palabras clave: Algoritmos como servicios, AaaS, arquitectura de referencia, *Cloud computing*, Funciones como servicios, *FaaS*, industria 4.0, *industrial internet reference architecture (IIRA)*, *Serverless*

Abstract

This Master's thesis arises from the need to define an architecture to be followed when implementing a system of algorithms as a service within Industry 4.0.

The aim of this master's thesis is to provide an architecture to offer algorithms as services using the cloud computing paradigm. Algorithms will be gaining importance within Industry 4.0, especially due to their role within data analytics. An AaaS service model is based on the definition of an algorithm on a Cloud provider, and for the end user, the main benefit is that they do not have to worry about the management or deployment of the infrastructure needed to instantiate or run the algorithm. Currently, within the cloud computing paradigm, this service is not defined as such and therefore does not have a reference architecture. Because of this, in this project a study will be carried out to propose a reference architecture focused on a service based on algorithms.

To develop this service within Industry 4.0, a study of the different reference architectures found in the industry will be carried out and, based on this analysis, an implementation plan will be developed considering this reference architecture. We must always consider the different perspectives from the point of view of business, application, information, and technology.

Finally, to facilitate the development of the service, we will implement continuous integration files with the idea of automating some of the tasks between the developer and the deployment of the service.

Keywords: Algorithms as a service, AaaS, Cloud *computing*, Functions as a service, FaaS, industry 4.0, industrial internet reference architecture (IIRA), Reference Architecture, Serverless



Resum

Aquest treball de Final de Màster neix de la necessitat de definir una arquitectura a seguir a l'hora d'implementar un sistema d'algoritmes com a servei dins de la indústria 4.0.

L'objectiu d'aquest treball de fi de màster és proporcionar una arquitectura per oferir els algoritmes com a serveis utilitzant el paradigma de *cloud computing*. Els algoritmes aniran guanyant importància dins de la indústria 4.0, especialment pel seu paper dins de l'anàlisi de dades. Un model de servei AaaS es basa en la definició d'un algoritme sobre un proveïdor Cloud, i per a l'usuari final, el benefici principal és que no cal preocupar-se de la gestió o el desplaçament de la infraestructura necessària per instanciar o executar l'algoritme. Actualment, dins del paradigma del *cloud computing* no hi existeix aquest servei definit, i per tant no té una arquitectura de referència. Conseqüentment, en aquest projecte es farà un estudi que permeta proposar una arquitectura de referència enfocada a un servei basat en algoritmes.

Per poder desenvolupar aquest servei dins de la indústria 4.0 es realitzarà un estudi de les diferents arquitectures de referència que trobem en la indústria i a partir d'aquest anàlisi, desenvolupant un pla d'implementació tenint en compte dicha arquitectura de referència. Deberemos siempre tener en cuenta las diferentes perspectivas desde el punto de vista de negocio, aplicación de información y tecnología.

Finalment, per facilitar el desenvolupament del servei, implementarem fitxers d'integració contínua amb la idea d'automatitzar algunes de les tarifes entre el desenvolupador i el desplaçament del servei.

Paraules clau: Algoritmes com a servei, AaaS, Arquitectura de referència, *Cloud computing*, Funcions com a servei, *FaaS*, indústria 4.0, industrial internet reference architecture (IIRA), Serverless

Tabla de contenidos

| | |
|---|----|
| 1. Introducción..... | 1 |
| 1.1 Motivación | 2 |
| 1.2 Objetivos | 3 |
| 1.2.1 Objetivo general | 3 |
| 1.2.2 Objetivos específicos..... | 3 |
| 1.3 Estructura de la memoria..... | 3 |
| 2. Estado del arte | 4 |
| 2.1 Cloud Computing o Servicios en la nube | 4 |
| 2.2 Contenedores y Orquestadores..... | 6 |
| 2.3 Serverless computing | 7 |
| 2.4 Estructura de un FaaS..... | 8 |
| 2.5 Plataformas FaaS y su arquitectura..... | 9 |
| 2.5.1 AWS Lambda | 10 |
| 2.5.2 Azure Functions..... | 10 |
| 2.5.3 Google Cloud Functions | 11 |
| 2.5.4 IBM Cloud Functions | 12 |
| 2.5.5 Kubeless..... | 13 |
| 2.5.6 Knative | 13 |
| 2.5.7 Apache OpenWhisk..... | 14 |
| 2.5.8 Fission | 14 |
| 2.5.9 OpenFaaS | 15 |
| 2.6 Análisis de la arquitectura serverless y el concepto FaaS | 15 |
| 2.7 Industria 4.0 y cloud computing | 16 |
| 2.8 Iniciativas relevantes y arquitecturas de referencia dentro de la industria 4.0.. | 18 |
| 2.8.1 Industrial Internet Reference Architecture (IIRA)..... | 18 |
| 2.8.2 OpenFog | 21 |
| 2.8.3 Modelo de Arquitectura de Referencia para la Industria 4.0 (RAMI 4.0). | 24 |
| 2.8.4 ZDMP | 26 |
| 2.8.5 AI4EU..... | 27 |
| 2.8.6 Comparativa de las arquitecturas de referencia..... | 27 |
| 2.9 Arquitectura de referencia | 28 |
| 2.9.1 Business <i>Viewpoint</i> | 29 |

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio
(AaaS) en entornos de Industria 4.0

| | | |
|-------|---|----|
| 2.9.2 | Usage <i>Viewpoint</i> | 30 |
| 2.9.3 | Functional viewpoint | 31 |
| 2.9.4 | Implementation viewpoint | 32 |
| 2.10 | Análisis de la arquitectura para AaaS | 33 |
| 3. | Solución Propuesta | 35 |
| 3.1 | Plan de trabajo | 35 |
| 3.2 | Plan detallado..... | 36 |
| 4. | Diseño de la Solución..... | 38 |
| 4.1 | Etapa 1: Captura de Requerimientos..... | 38 |
| 4.1.1 | Fase 1: Definición de requerimientos Funcionales..... | 38 |
| 4.1.2 | Fase 2: Definición de requerimientos No Funcionales | 41 |
| 4.1.3 | Fase 3: Casos de Uso | 42 |
| 4.2 | Etapa 2: Vista de Negocio y de Operación | 51 |
| 4.2.1 | Fase 1: Vista de negocio | 51 |
| 4.2.2 | Fase 2: Vista de operaciones | 59 |
| 4.3 | Etapa 3: Vista funcional y de implementación..... | 67 |
| 4.3.1 | Fase 1: Vista funcional | 68 |
| 4.3.2 | Fase 2: Vista de implementación | 72 |
| 5. | Implementación..... | 79 |
| 5.1 | Desarrollo de fichero de integración continua..... | 79 |
| 5.2 | Desarrollo de un algoritmo..... | 82 |
| 5.3 | Preparación del entorno | 84 |
| 5.4 | Preparar nuestro algoritmo como servicio | 87 |
| 6. | Conclusiones..... | 89 |
| 7. | Referencias..... | 91 |

Índice de tablas

| | |
|---|----|
| Tabla 1. Casos de uso | 45 |
| Tabla 2. Caso de uso – Guías | 46 |
| Tabla 3. Caso de uso – Crear algoritmo..... | 46 |
| Tabla 4. Caso de uso – Template | 47 |
| Tabla 5. Caso de uso – Build | 47 |
| Tabla 6. Caso de uso – Repositorio del algoritmo..... | 48 |
| Tabla 7. Caso de uso – Configuración del entorno | 48 |
| Tabla 8. Caso de uso – Selección del algoritmo | 49 |
| Tabla 9. Caso de uso – Despliegue algoritmo..... | 49 |
| Tabla 10. Caso de uso – Petición | 50 |
| Tabla 11. Caso de uso – Cliente..... | 50 |
| Tabla 12. Tipos Stakeholders | 54 |
| Tabla 13. Stakeholder: Responsable de la toma de decisiones | 55 |
| Tabla 14. Stakeholder: Responsable de producción | 55 |
| Tabla 15. Stakeholder: Operarios de planta..... | 56 |
| Tabla 16. Stakeholder: Operarios de Mantenimiento | 56 |
| Tabla 17. Stakeholder: Programadores | 56 |
| Tabla 18. Stakeholder: Supervisor..... | 57 |
| Tabla 19. Stakeholder: Ingenieros de Sistemas de producción..... | 57 |
| Tabla 20. Stakeholder: Project Manager..... | 57 |
| Tabla 21. Stakeholder: Desarrolladores de algoritmos..... | 58 |
| Tabla 22. Stakeholder: Técnicos..... | 58 |
| Tabla 23. Taras: Producir un artículo | 60 |
| Tabla 24. Taras: Predecir calidad | 60 |
| Tabla 25. Taras: Comprobar manualmente..... | 61 |
| Tabla 26. Taras: Decidir modificación parámetros | 61 |
| Tabla 27. Taras: Proponer nuevos parámetros..... | 61 |
| Tabla 28. Taras: Decidir método para resolver | 61 |
| Tabla 29. Taras: Modificar automáticamente | 62 |
| Tabla 30. Taras: Modificar manualmente..... | 62 |
| Tabla 31. Taras: Admitir artículo | 62 |
| Tabla 32. Taras: Rechazar artículo | 62 |



Desarrollo de una arquitectura para la implantación de Algoritmos como servicio
(AaaS) en entornos de Industria 4.0

| | |
|--|----|
| Tabla 33. Taras: Corregir articulo | 62 |
| Tabla 34. Detectar datos..... | 62 |
| Tabla 35. Actividad: Producir | 64 |
| Tabla 36. Actividad: Producir objeto de baja calidad..... | 64 |
| Tabla 37. Actividad: Corregir automáticamente..... | 64 |
| Tabla 38. Actividad: Corregir manualmente | 64 |
| Tabla 39. Actividad: Predecir calidad | 64 |
| Tabla 40. Tareas Stakeholder: Ingeniero de Sistema..... | 65 |
| Tabla 41. Tareas Stakeholder: Ingeniero de Sistema..... | 65 |
| Tabla 42. Tareas Stakeholder: Ingeniero de Sistema..... | 65 |
| Tabla 43. Tareas Stakeholder: Ingeniero de Sistema..... | 65 |
| Tabla 44. Tareas Stakeholder: Ingeniero de Sistema..... | 66 |
| Tabla 45. Tareas Stakeholder: Ingeniero de Sistema..... | 66 |
| Tabla 46. Tareas Stakeholder: Ingeniero de Sistema..... | 66 |
| Tabla 47. Tareas Stakeholder: Project Manager | 66 |
| Tabla 48. Tareas Stakeholder: Desarrollador de algoritmos..... | 66 |
| Tabla 49. Tabla de Actividades | 67 |
| Tabla 50. Componente: Obtener datos | 69 |
| Tabla 51. Componente: Lanzar un algoritmo | 70 |
| Tabla 52. Componente: Suscribirse a un algoritmo..... | 70 |
| Tabla 53. Componente: Manejador de datos | 70 |
| Tabla 54. Componente: Unidad de computo..... | 70 |
| Tabla 55. Componente: Cliente | 71 |
| Tabla 56. Estructura API..... | 81 |



Índice de imágenes

| | |
|--|----|
| Figura 1. Representación de servicios | 5 |
| Figura 2. Arquitectura AWS Lambda..... | 10 |
| Figura 3. Arquitectura Azure Functions..... | 11 |
| Figura 4. Logo Google Cloud Functions..... | 11 |
| Figura 5. Arquitectura IBM Cloud Functions..... | 12 |
| Figura 6. Logo Kubeless | 13 |
| Figura 7. Logo Knative..... | 13 |
| Figura 8. Logo Apache OpenWhisk | 14 |
| Figura 9. Logo Fission | 14 |
| Figura 10. Logo OpenFaaS..... | 15 |
| Figura 11. IIRA Architecture Framework (Lin, Miller, et al., 2017) | 19 |
| Figura 12. Fog/Edge computing (ErpNews, 2018) | 21 |
| Figura 13. Pilares OpenFog..... | 22 |
| Figura 14. Arquitectura RAMI 4.0..... | 25 |
| Figura 15. Logo ZDMP..... | 26 |
| Figura 16. Logo AI4EU | 27 |
| Figura 17. IIRA vs RAMI 4.0 | 28 |
| Figura 18. IIRA: Arquitectura..... | 28 |
| Figura 19. IIRA: Business viewpoint..... | 29 |
| Figura 20. IIRA: Usage viewpoint..... | 30 |
| Figura 21. IIRA: Functional Viewpoint | 31 |
| Figura 22. IIRA: Implementation viewpoint..... | 32 |
| Figura 23. IIRA: Viewpoints..... | 33 |
| Figura 24. Requisitos no funcionales de la ISO 9126:2001 | 41 |
| Figura 25. Diagrama de casos de uso | 43 |
| Figura 26. BPM..... | 44 |
| Figura 27. ISA-95 Model | 51 |
| Figura 28. Isa-95 y Stakeholders | 52 |
| Figura 29. IIRA: Business viewpoint..... | 53 |
| Figura 30. Modelo de distribución de los Stakeholders | 54 |
| Figura 31. IIRA: Usage Viewpoint | 59 |
| Figura 32. Flujo de tareas | 63 |

| | |
|--|----|
| Figura 33. Diagrama funcional | 69 |
| Figura 34. Diagrama de secuencia | 71 |
| Figura 35. Logo Docker | 72 |
| Figura 36. Docker y Máquina Virtual | 73 |
| Figura 37. Logo Kubernetes..... | 73 |
| Figura 38. Nodos Kubernetes | 74 |
| Figura 39. Logo Gitlab | 75 |
| Figura 40. Logo SonarQube..... | 75 |
| Figura 41. Logo Kaniko..... | 76 |
| Figura 42. Arquitectura Kaniko..... | 76 |
| Figura 43. Logo docker-compose..... | 77 |
| Figura 44. Logo Rancher | 77 |
| Figura 45. Implementación de gitlab-ci | 78 |
| Figura 46. Implementación de docker-compose..... | 78 |
| Figura 47. Implementación de Rancher | 78 |
| Figura 48. Estructura Template AaaS | 80 |
| Figura 49. Jobs CI/CD | 83 |
| Figura 50. Container Registry | 84 |
| Figura 51. Pagina bienvenida Rancher | 86 |
| Figura 52. Dashboard Rancher | 87 |
| Figura 53. Estructura ficheros Helm..... | 88 |
| Figura 54. Añadir catálogo Rancher..... | 88 |
| Figura 55. AaaS Desplegado | 88 |

1. Introducción

Debido a la globalización económica y al aumento de la demanda por parte de la sociedad, se ha generado la necesidad de optimizar y mejorar todos los procesos de fabricación y producción dentro de la industria. Gracias a esto, actualmente nos encontramos ante una creciente evolución de las nuevas tecnologías aplicadas a la industria, las cuales buscan mejorar el rendimiento en la producción. Para ello, se investigan nuevas formas de obtener información de toda la cadena de producción y cómo emplearla con este fin. Surge así el concepto de optimización y digitalización de los procesos industriales mediante el uso de tecnologías como el Internet de las cosas (IoT), la Inteligencia Artificial (IA) y el Big Data o *Cloud Computing*. A toda esta revolución tecnológica se le denomina Industria 4.0 (Popkova et al., 2018).(Popkova et al., 2018).

El concepto de Industria 4.0 está presente desde hace algunos años y ha permitido aumentar la eficiencia de la cadena de suministro, optimizar el consumo de energía y reducir errores durante los procesos de fabricación (Mehrpooya et al., 2019). No obstante, aunque se ha demostrado que esta revolución industrial es beneficiosa, apenas el 48% de las empresas de fabricación están preparadas para afrontar cambios tecnológicos (Baur & Wee, 2015). El rápido crecimiento tecnológico que afecta a la industria supone una dificultad añadida para las pequeñas y medianas empresas, que deben actualizarse teniendo en cuenta estos nuevos avances tecnológicos y la inversión que requieren.

Uno de los avances tecnológicos más importantes y en el que se encuadra este trabajo de Final de Máster es el *cloud computing*. Esta tecnología nos proporciona la posibilidad de compartir diferentes recursos de forma remota a través de la red e internet, de modo que se puede facilitar el despliegue de nuevos servicios y el mantenimiento de estos. Gracias a este avance se han aumentado los dispositivos IoT que obtienen datos de diferentes puntos de la cadena de producción y que ayudan en el proceso de toma de decisiones con la intención de mejorar y optimizar la producción.

Para optimizar la producción es fundamental no solo recoger sino también analizar toda la información que se obtiene en cada momento de los diferentes dispositivos IoT. El fin es que con toda esta información se puedan reconfigurar las líneas de producción dependiendo de los datos obtenidos en cada momento, siempre de la forma más automática posible. Aquí es donde nace la necesidad de aplicar algoritmos que permitan y faciliten esta toma de decisiones a partir de los datos obtenidos en vivo aplicando inteligencia artificial a la cadena de producción.



Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

Un algoritmo se define como un conjunto de instrucciones estructuradas que siguen un orden lógico y cuyos pasos permiten resolver un problema, que en este caso estarían ligados a mejorar la producción. Aprovechando los avances tecnológicos propiciados por la industria 4.0, el *cloud computing* ofrece la posibilidad de proporcionar un algoritmo de forma remota a través de la red para esta optimización de la producción, siendo este el punto donde nace el concepto de Algoritmos como Servicios o en inglés *Algorithm as a Service (AaaS)*, y el objeto de este TFM, definir una arquitectura de referencia para la implementación de este tipo de servicios.

Para poder desarrollar una arquitectura adecuada para la introducción de un AaaS en la industria debemos conocer la estructura y necesidades de esta. Es por ello que, en este trabajo de Final de Máster analizaremos diferentes arquitecturas dentro de los servicios de *cloud computing* más conocidos, además de aquellas arquitecturas aplicadas a la organización dentro de la industria. De esta forma se busca desarrollar una arquitectura de referencia para poder implementar un servicio de algoritmos en base a *cloud computing*, teniendo en cuenta toda la estructura lógica, física y humana que podemos encontrar dentro de la industria.

1.1 Motivación

La motivación para desarrollar este proyecto nace de mi trabajo en el Centro de Investigación de Gestión e Ingeniería de la Producción (CIGIP) de la Universidad Politécnica de Valencia (UPV). Entre los proyectos nacionales y europeos en los que trabajan cabe destacar dos: *Industrial Data Services for Quality Control in Smart Manufacturing (i4Q)* y *Zero Defects Manufacturing Platform (ZDMP)*.

I4Q es un proyecto que trata de desarrollar un conjunto de soluciones para servicios de datos industriales fiables (RIDS) basada en el internet de las cosas (IoT), donde se desarrollarán un conjunto de herramientas que permitirán gestionar una gran cantidad de datos industriales (CORDIS, 2021).

ZDMP es un proyecto que trata de ofrecer una plataforma de código abierto con una serie de aplicaciones que buscan alcanzar el objetivo de cero defectos. Este término hace referencia a la necesidad que aparece en la producción cuando un producto debe de cumplir todos los requisitos de calidad exigidos (CORDIS, 2019).

Por otro lado, el desarrollo de mi trabajo de final de grado con el título de “Implementación de una aplicación para la guía en la implantación de sistemas de fabricación” (Mateo-Casali et al., 2019) ya me acercó a los conceptos de la industria 4.0

y las posibilidades que tiene la ingeniería informática dentro de los procesos de fabricación.

Como estudiante del Máster Universitario en Ingeniería Informática (MUIINF), actual investigador del CIGIP y dando continuidad a mi trabajo de final de grado, he deseado poder introducirme más en los conceptos tecnológicos de la industria, aplicando los conocimientos obtenidos durante mis estudios del Master y experiencia laboral.

1.2 Objetivos

1.2.1 Objetivo general

El objetivo principal de este trabajo de final de Máster es desarrollar una arquitectura de referencia para una correcta implantación de los algoritmos como servicio (AaaS) dentro de la industria 4.0.

1.2.2 Objetivos específicos

A partir del objetivo general planteado para este proyecto, surgen una serie de objetivos secundarios o específicos:

1. Conocer en profundidad las arquitecturas de referencia dentro de la industria 4.0 y las arquitecturas de servicios que nos ofrece *cloud computing*.
2. Desarrollar una planificación de trabajo para la implantación del servicio de algoritmos teniendo en cuenta las arquitecturas aprendidas en el proceso de investigación.
3. Simular la implementación de nuestro servicio de algoritmos utilizando la arquitectura de referencia propuesta.

1.3 Estructura de la memoria

La estructura de este trabajo de fin de máster se desarrolla en tres partes diferenciadas. Una primera parte de investigación, donde analizaremos y conoceremos todas las arquitecturas de referencia. Una segunda parte donde desarrollaremos un plan de implementación a partir de los conocimientos adquiridos de la investigación previa. Finalmente, se ha incluido una tercera parte donde, a modo de prueba, veremos el proceso de desarrollo e implementación del algoritmo como servicio utilizando la arquitectura definida en este proyecto.

2. Estado del arte

En esta sección haremos un recorrido por todos los aspectos necesarios para comprender por qué los algoritmos como servicios dentro de la Industria 4.0 pueden resultar un recurso muy interesante. Para ello, comenzaremos analizando los servicios conocidos como *cloud computing*, cada vez más presentes, profundizando en aquellos cuyas características resultan interesantes para el presente desarrollo. Seguidamente, veremos algunas arquitecturas de referencia dentro de la industria que nos permitan entender determinados modelos ya utilizados que pueden servir de base para plantear nuestra propia arquitectura para un AaaS. Finalmente, a lo largo de este apartado iremos contextualizando algunas de las herramientas que posteriormente utilizaremos.

Comenzaremos este apartado analizando brevemente las infraestructuras en la nube, los diferentes tipos que existen y su creciente relevancia. Una vez visto esto, nos detendremos en el concepto *serverless* asociado a los servicios *FaaS*, cuyas características, estructura, diferentes plataformas privadas que nos proporcionan este servicio, así como opciones *open-source* pueden servirnos como base para comprender el funcionamiento de nuestro sistema AaaS. Paralelamente, realizaremos un análisis de las diferentes arquitecturas más importantes del sector industrial para servicios que podemos encontrar y, finalmente, plantearemos una pequeña conclusión de la que partiremos para el desarrollo de nuestra arquitectura.

2.1 Cloud Computing o Servicios en la nube

Los avances tecnológicos en las últimas décadas nos han llevado a una nueva era dentro del ámbito industrial, también conocido como cuarta revolución industrial. Esta revolución busca establecer estándares para la fabricación en sí y todos los procesos asociados al mismo, con el fin de aumentar la productividad y la eficiencia. Durante todo este progreso, el desarrollo de software ha evolucionado junto con sus paradigmas o implementaciones, desde el uso de microservicios, pasando por dividir por capas o desarrollando arquitecturas basadas en *mainframes*. Estos cambios fomentan la aparición de tecnologías subyacentes que generan cada vez más opciones o soluciones óptimas y eficientes, teniendo en cuenta aspectos tan importantes como el rendimiento, el uso de recursos o el coste.

Una de las soluciones que más impacto ha generado son las infraestructuras en la nube conocidas como *cloud computing*. Estas infraestructuras ofrecen la posibilidad de desplegar aplicaciones y servicios sobre recursos que los proveedores proporcionan.

Grandes empresas como Amazon y Google han usado este término para describir aquellos servicios que permiten al usuario (ya sean empresas o particulares) almacenar datos de cualquier tipo, incluyendo aplicaciones, servicios, bases de datos, etc., sin necesidad de tener toda una infraestructura. No obstante, no fue hasta 2011 cuando el Instituto Nacional de Normas y Tecnología (NIST) acuñó este término describiéndolo del siguiente modo:

“La computación en nube es un modelo que permite un acceso a la red ubicuo, cómodo y a la carta a un conjunto compartido de recursos informáticos conjugables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente provisionados y liberados con un mínimo esfuerzo de gestión o de interacción del proveedor de servicios.” (Michael et al., 2011).

Algunas de las características esenciales que encontramos en los servicios en la nube son: autoservicio bajo demanda, amplio acceso a la red, agrupación de recursos, rápida elasticidad, servicio medido, etc.

Estas características permiten diferenciar quién gestiona la infraestructura, si es una nube pública, privada, híbrida o comunitaria. Dependiendo de las características de la infraestructura y su gestión, podemos diferenciar tres modelos de servicio distintos: *Software as a Service (SaaS)*, *Platform as a Service (PaaS)*, e *Infrastructure as a Service (IaaS)*. En los últimos años, además, han aparecido nuevos paradigmas de servicio, como pueden ser: *Container as a Service (CaaS)*, *Backend as a Service (BaaS)*, *Functions as a Service (FaaS)*, etc (figura 1).

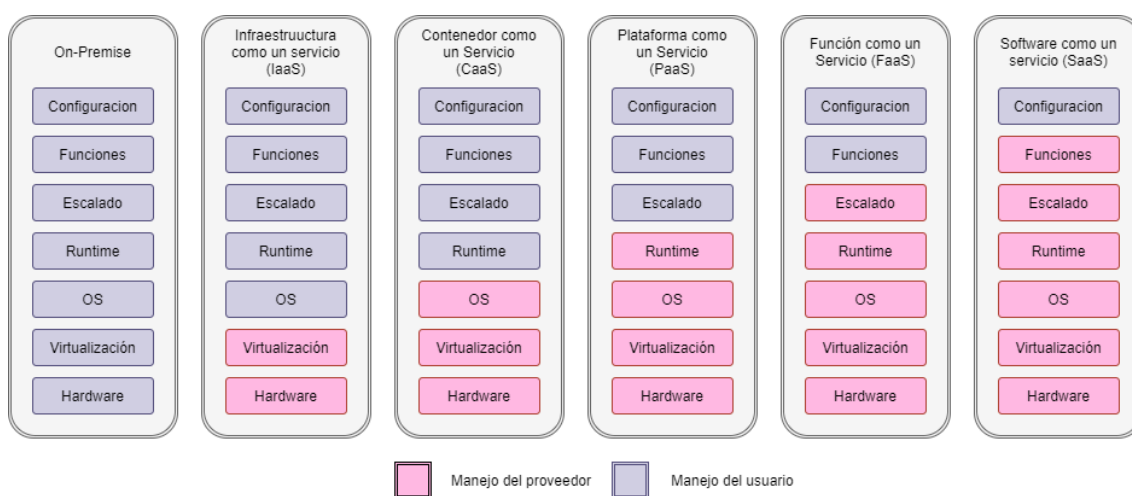


Figura 1. Representación de servicios

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

La opción más básica que puede ofrecer un proveedor de *cloud computing* es la Infraestructura como Servicio, IaaS por sus siglas en inglés, que consisten en el alquiler de equipos informáticos que permitan a sus usuarios utilizar sus recursos. La contratación de estos servicios requiere que el usuario o empresa cuente con personal con conocimientos de gestión, ya que necesita instalar el sistema operativo, configurar opciones de red o almacenamiento, etc. Por ello, existen herramientas que ayudan a su gestión, como puede ser Cloudify, que permite la integración con herramientas de automatización como Kubernetes, Ansible o Terraform.

Puede darse un escenario en el que el usuario que contrata el servicio no tenga conocimientos de configuración, poco tiempo para gestionar sus infraestructuras, o que requiera de una plataforma o entorno sobre el que poder operar directamente. Aquí es donde aparece el modelo denominado Plataforma como Servicio (PaaS). Este modelo ofrece a los usuarios un entorno gestionado donde simplemente tiene que desplegar sus aplicaciones. De la misma forma que en los IaaS, también existen herramientas para facilitar su uso, como CloudFoundry, una herramienta de despliegue que facilita la integración de *Frameworks* para desarrollo de aplicaciones.

Como última opción tenemos la posibilidad de tener una aplicación ya desplegada donde el usuario solo tiene que configurar pequeñas cosas. Este modelo se conoce como Software como Servicio (SaaS) y, aunque es el más sencillo de utilizar, también resulta ser el más restrictivo, ya que el usuario solo puede acceder a aplicaciones que ofrezca el proveedor. Algunos de los proveedores más conocidos de SaaS son Google con GSuite, y Microsoft, con Office 365.

Procesos específicos para su implementación, pasando de servicios on-Premise a *cloud*, como la migración del software, suele ser sencilla. Los proveedores ofrecen servicios específicos que facilitan el proceso de traslado del software de las máquinas propias del usuario a las del proveedor *cloud*.

2.2 Contenedores y Orquestadores

Gracias al desarrollo de la virtualización, el uso de contenedores se ha convertido en uno de los pilares dentro de la distribución, despliegue y mantenimiento de aplicaciones. La herramienta más popularizada es Docker, aunque encontramos otras como OpenVZ o LXD. Estos contenedores se usan para sacar al mercado servicios centrados en la arquitectura de microservicios, y resultan muy útiles al permitir gestionar y desplegar fácilmente, de forma independiente al sistema, pequeños servicios con sus

propias dependencias. Aquí aparecen los Contenedores como servicio (CaaS), que permiten a los usuarios cargar, organizar, iniciar y administrar contenedores.

Cuando tratamos con una gran cantidad de microservicios en numerosos contenedores, puede elevarse bastante el coste para el usuario. Así aparece la orquestación de los servicios, que permite auto escalar o generar equilibrio de carga entre los contenedores facilitando la gestión de estas arquitecturas. Los principales orquestadores de código abierto son:

- **Kubernetes:** Un sistema de código abierto que orquesta contenedores Docker, desarrollado originalmente por Google. Es el estándar por defecto para la orquestación tanto en nubes públicas como privadas. Esto es debido a su fiabilidad, auto escalado y equilibrio de carga. Tanto es así que es el orquestador que usa empresas como Amazon o Microsoft (Kubernetes, 2019).
- **Docker Swarm:** Desarrollado por la empresa de Docker, ofrece los servicios de orquestación que ofrece Kubernetes. Docker Swarm es gestionado por el mismo motor que usa Docker, por lo que existen muchas facilidades para el usuario que usa el cliente de Docker (Docker, 2020).
- **OpenShift:** Se trata de una versión open-source de RedHat para la distribución de Kubernetes que se ejecuta de forma nativa en AWS. Al usar el mismo entorno de Kubernetes, está listo para que muchas empresas puedan trasladar las cargas de trabajo a la nube con mayor facilidad. (RedHat, 2020)

2.3 Serverless computing

La rápida evolución de aplicaciones para web o móviles, llevaron a los desarrolladores de software a separar el frontend del backend en las aplicaciones, y fruto de esto aparece el modelo de Backend como Servicio (BaaS). Este nuevo servicio es parecido al modelo SaaS, pero este se centra en el desarrollo de la aplicación final para el usuario. Uno de los servicios más populares es *Firebase* de Google. Gracias a él, los proveedores pueden ofrecer interfaces APIs o desarrollos de software que ya implementen funciones como las redes sociales, notificaciones o permitir la conexión con servicios en la nube y autenticación (Jonas et al., 2019).

Paralelamente, surge también la necesidad de ejecutar tareas más ligeras en la nube y que permitan ahorrar tiempo y dinero. Se crea así el modelo de Funciones como Servicio o *FaaS*. Este servicio está destinado a aquellos desarrolladores que únicamente se centran en ofrecer un trozo de código o función, mientras que el proveedor del servicio se encarga del aprovisionamiento, el escalado, etc. Hay que tener

en cuenta que los códigos de programación que puede usar el desarrollador están limitados a la disponibilidad que le ofrece el proveedor. Alguno de los proveedores de *FaaS* más conocidos son *AWS Lambda*, *Microsoft Azure Functions* o *Apache OpenWhisk*, este último de código abierto.

Tras la definición de *BaaS* y *FaaS*, la Fundación de Computación en la Nube (CNCF) definió un nuevo concepto llamado *serverless computing* (Allen et al., 2018). El paradigma de *serverless computing* hace referencia a una arquitectura o modelo de programación basado en la ejecución de porciones de código sin que el desarrollador tenga ningún control de los recursos sobre el sistema en el que se lanza. Por tanto, usamos el termino *serverless* para hablar de aplicaciones que dependen de servicios auto-gestionados, dentro del cual también podremos enmarcar este servicio *AaaS*.

En el *AaaS*, para que todo este engranaje de herramientas funcione conjuntamente y facilite al usuario insertar o utilizar los diferentes algoritmos, es necesario un servicio que se encargará de gestionar toda la lógica de la aplicación, y realizará los cambios que sean necesarios para poder ejecutar el código para el usuario final. Para entender el desarrollo necesario para este servicio, utilizaremos como referencia el funcionamiento de los servicios *FaaS*, que veremos en el siguiente apartado.

2.4 Estructura de un *FaaS*

Como hemos visto anteriormente, *FaaS* trata de ofrecer trozos de código bajo demanda en una plataforma ajena al usuario sin la necesidad de que este trate directamente con la infraestructura. El código que se almacena no requerirá ninguna dependencia, ya que estará aislado y es independiente. Para que un servicio se considere *FaaS* deberá de cumplir dos funciones: primero, recibir el código de la función que se tiene que ejecutar y, segundo, poder lanzarlo. Para ello debe contar con un mecanismo que permita al usuario comunicarse directamente y que, además, englobe todas las acciones de envío, invocación y obtención de resultado de las funciones. También debe contar con un entorno de ejecución que, con la configuración necesaria, permita recibir toda la función, ejecutarla y obtener los resultados.

Para poder entender mejor cómo funciona un servicio *FaaS*, tenemos que ver los diferentes actores que pueden usar esta herramienta. Por un lado, el desarrollador o usuario y por otro el proveedor del servicio. El usuario o desarrollador es el que crea el código, añade todas las dependencias que requiere para su funcionamiento y aparte, integra un *trigger* o disparador que se activa cuando otra herramienta del ecosistema lo

llama (una solicitud POST, una llamada a la BD, etc...). Por otro lado, tenemos el proveedor del *FaaS*, que se encargará de lanzar el código proporcionado por el desarrollador, controlando los tiempos y adaptándose a la carga que reciba. Para que esto funcione, el proveedor habilitará un punto de acceso teniendo siempre en cuenta la seguridad, eficiencia y latencia, entre otras opciones. Estos actores serán los mismos que encontraremos en nuestro *AaaS*.

La arquitectura de *FaaS* consta en tres partes:

- **Una API** que sirve como punto de acceso variable, ya que puede ser por medio de interfaz gráfica o por llamadas CLI entre la plataforma y el usuario. Debido a que estas herramientas se encuentran integradas en un *serverless*, estas comunicaciones de la API se lanzan a partir de disparadores o *triggers* que realizan la invocación debido a algún evento generado.
- **Contenedores** donde se realiza el entorno de ejecución, permitiendo la abstracción, protección de la infraestructura y proporcionando aislamiento entre las diferentes instancias. Además, debido a que su ciclo de vida es ágil, posee una gran versatilidad que permite lanzar tanto funciones pesadas como ligeras.
- **Diferentes estructuras internas** que facilitan el apoyo al flujo de información. Es decir, que contienen al menos una base de datos, que permite obtener persistencia de datos, y unos mecanismos que ayudan a repartir el trabajo, como pueden ser los procesos de invocación, balanceadores de carga, etc. Aunque esto último, en muchas ocasiones, se externaliza hacia otros servicios.

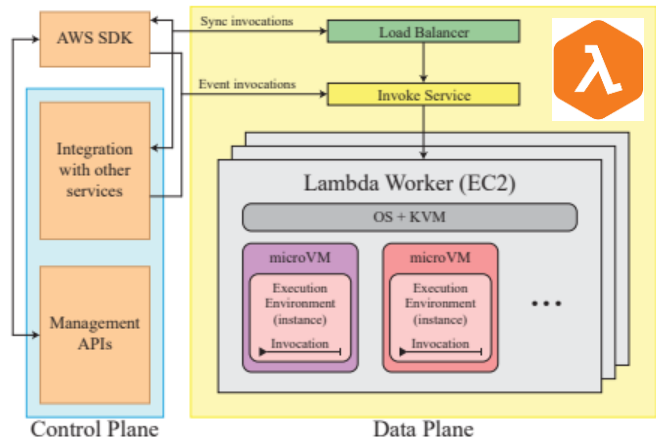
A partir de este punto, analizaremos las diferentes arquitecturas aplicadas en varias arquitecturas *FaaS*, tanto de código abierto como privadas. Con ello podremos, posteriormente, valorar las ventajas y desventajas de cada una de ellas de cara a un servicio *FaaS*.

2.5 Plataformas *FaaS* y su arquitectura

Debido a las facilidades que genera esta arquitectura, han aparecido muchos *Frameworks* o proveedores que ofrecen este servicio. A continuación, veremos algunos de los servicios privados y públicos de *FaaS* más importantes (Barcelona-Pons & García-López, 2020).

2.5.1 AWS Lambda

AWS Lambda es el servicio informático *serverless* de Amazon. Permite administrar los servidores, crear la lógica de escalado, mantener las integraciones de eventos y administrar su tiempo de ejecución. La arquitectura (figura 2) de este servicio se divide en dos bloques:



- *Control Plane*. Se encarga de toda la gestión de la plataforma, incluyendo la integración con los servicios y la administración de la API (gracias a esta API podemos crear y actualizar las funciones).
- *Data Plane*. Se encarga de supervisar los recursos y las invocaciones que recibe la aplicación.

Figura 2. Arquitectura AWS Lambda

AWS Lambda incluye un servicio denominado *Invoke* que controla y asigna los entornos de ejecución a las invocaciones de las funciones. Las invocaciones activadas por eventos van directamente al *Invoke*, donde pueden ponerse en cola; hay que tener en cuenta que las invocaciones síncronas necesitan una gestión adicional para responder a las personas que las llaman. Para gestionar esta asincronía, utiliza un componente de equilibrio de carga de trabajo.

En AWS Lambda, el usuario despliega las funciones individualmente, la API de gestión permite la creación y configuración de funciones (por ejemplo, la memoria o el tiempo de ejecución de la función), el código de la función se carga en el servicio en paquetes comprimidos y la configuración se actualiza directamente con peticiones HTTP de las funciones. Aunque las funciones pueden invocarse directamente con una petición HTTP, lo habitual es vincularlas a eventos o *triggers*. Los desencadenantes establecen vínculos con otros servicios en la nube que producen nuevos eventos, y permiten habilitar la invocación de funciones en respuesta a esos eventos.

2.5.2 Azure Functions

Se trata de la opción de servicio bajo demanda por eventos de la compañía de Microsoft. La arquitectura (figura 3) de este servicio invoca instancias de las Funciones

en respuesta a un evento. Para la escalada de todas estas instancias utiliza un componente que monitoriza el estado del servicio, denominado *Scale Controller*.

El funcionamiento de *Azure Functions* se basa en un paquete de funciones que gestiona un conjunto de recursos (instancias de las funciones). Este paquete compilado lo sube el usuario, e incluye las dependencias, el código y la configuración (anotaciones del código y los archivos relacionados con la aplicación).

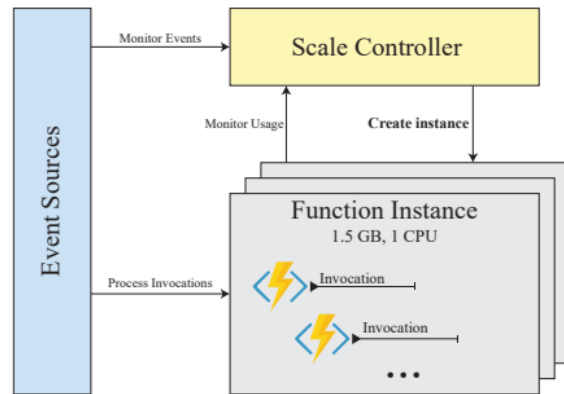


Figura 3. Arquitectura Azure Functions

Cada definición de función es un trozo del código añadido como una función de Azure. Contiene un *trigger* y un enlace a la misma, que define los eventos generados cuando se invoca, permitiendo que las funciones puedan operar con un flujo de entrada y salida; además, crea puntos de invocaciones finales por medio de una API de acceso por HTTP. Mediante los parámetros de configuración avanzada que tiene *Azure Functions*, se puede permitir algunas políticas de escalado y gestión, creando una interfaz intuitiva para el usuario.

2.5.3 Google Cloud Functions

Google Cloud Funcions (figura 4) es el servicio informático *serverless* de Google. Dado que no detalla la arquitectura de su servicio y componentes, es decir, cómo trata las peticiones o cómo gestiona los recursos, no se puede presentar un esquema de cómo funciona esta plataforma.



Figura 4. Logo Google Cloud Functions

Con respecto a su funcionamiento, el *FaaS* de Google lanza las funciones de forma individual, aunque todo el código esté en el mismo paquete. Para desplegarla, carga el directorio donde se encuentra el código y detecta las funciones basándose en la estructura del proyecto; estas funciones también pueden ser invocadas por HTTP, o asociarse a eventos mediante eventos o como respuesta a otros servicios. La configuración se actualiza por medio de consultas HTTP hacia una API. Solo se paga por su uso y permite la conexión con servicios para compilar aplicaciones complejas utilizando siempre tecnología abierta.



2.5.4 IBM Cloud Functions

Se trata de un *FaaS* de código abierto, aunque cuenta con opciones premium de pago. Este proyecto fue iniciado por IBM y cedido a Apache Software Foundation, de ahí que IBM Cloud Functions se despliegue con Apache OpenWhisk.

La arquitectura (figura 5) de ese servicio está formada por cuatro componentes:

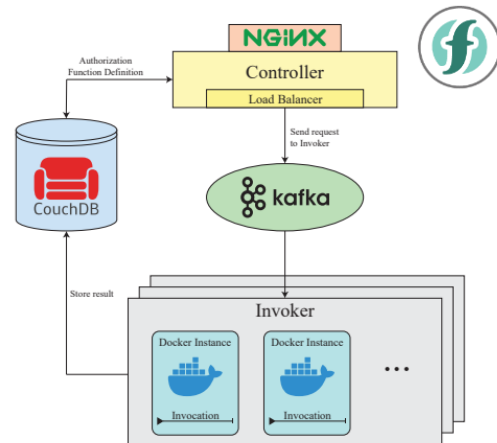


Figura 5. Arquitectura IBM Cloud Functions

- *El controlador*. Actúa como equilibrador de la carga de peticiones y gestiona los recursos de las instancias.
- Las máquinas *Invoker*. Son los orquestadores que ejecutan varios contenedores Docker. También tiene acciones como *triggers* o reglas, a las cuales se puede acceder por HTTP.
- Canal de comunicación Kafka entre el *invoker* y el controlador.
- Almacenamiento. Las peticiones y los datos de petición se guardan en una base de datos CouchDB.

Apache OpenWisk, del cual hablaremos más adelante, denomina las funciones como “*Actions*” y las despliega individualmente en los servidores. Estas acciones tienen unos nombres asignados, así como un grupo y pueden estar organizados en paquetes. Todas las acciones se realizan a través del controlador que expone una API de acceso por HTTP.

Una de las alternativas a los proveedores de servicios *FaaS* son las *Open-Source FaaS platforms*. Estas plataformas van dirigidas a aquellas empresas o usuarios que requieran de mayor control sobre sus datos, ya que permiten aplicarse o adaptarse según las necesidades. Cabe destacar que estas alternativas no requieren de un despliegue únicamente en la nube pública, sino también en la privada, ya que estos sistemas se basan en orquestadores de contenedores como Kubernetes. Consecuentemente, se genera un inconveniente a la hora de gestionarlo y desplegarlo, haciendo que sea necesario un administrador de sistemas que comprenda el uso de algunas de las plataformas *FaaS*. Algunos ejemplos *open-source* son:

2.5.5 Kubeless

Kubeless (figura 6) nace de Bitnami en 2016 con la creación de una plataforma propia *FaaS*. Esta plataforma se divide en tres elementos: *Runtimes*, *Triggers* y *Functions*.



Figura 6. Logo Kubeless

- Las *runtimes* son los entornos preparados para inyectar código de la función antes de llamarla.
- Los *trigger* se encargan de activar las llamadas a las funciones. Kubeless trabaja con *triggers* HTTP, gestionados en *cronjobs*.
- Las *Functions* son el código ejecutable que se correrá una vez se le llame.

Kubeless trata con varios lenguajes diferentes de programación por defecto, pero se puede usar otros para las *runtimes* siempre y cuando mantengan el modelo que usa. Kubeless utiliza Kubernetes como orquestador y entre otras características ofrece escalabilidad de funciones, a través de escalabilidad horizontal de Kubernetes, y monitorización mediante Prometheus. Para el almacenamiento de *runtimes* como funciones se utiliza Docker Registry (repositorio de imágenes de Docker), que puede ser tanto privado como público. Para trabajar con las funciones, tiene una entrada en JSON y devuelve el resultado del mismo modo. Las dependencias de la función van ligadas al *runtimes* (se necesitarán crear runtimes para aquellas librerías propias o no soportadas por Kubeless).

2.5.6 Knative

Knative (figura 7) nace de Google y permite gestionar y desplegar cargas de trabajo *serverless* gracias al desarrollo de una plataforma basada en Kubernetes. A partir de la construcción de Kubernetes, esta plataforma crea funciones en contenedores aprovechando las funcionalidades de Kubernetes. Esta plataforma basa su funcionamiento en tres componentes (*build*, *eventing* y *serving*).



Figura 7. Logo Knative

- Build proporciona métodos de creación de imágenes de contenedores para su posterior uso en Kubernetes.
- Serving se basa en Kubernetes a través de Istio que permite la implementación del servicio de aplicaciones sin el servidor.
- Eventing es el sistema diseñado para gestionar y entregar los eventos.

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

Para construir y desplegar las funciones en Kubernetes, Knative utiliza Teekton Pipelines, un marco de código abierto para crear sistemas de integración continua CI/CD que permite a los desarrolladores construir, probar e implementar a través de controles de versiones.

2.5.7 Apache OpenWhisk

Apache OpenWhisk (figura 8) es una plataforma *serverless FaaS* creada por IBM y gestionada por la Fundación Apache de código abierto. Gestiona automáticamente toda la infraestructura, servicios y escalado de aplicaciones. Esta plataforma se divide en tres elementos: *Action*, *Trigger* and *Rules*.



Figura 8. Logo Apache OpenWhisk

- *Action* es la función que se va a ejecutar.
- El *tigger* es una clase de evento que provienen de una variedad de fuentes.
- *Rules* permite asignar a un trigger una acción.

Apache OpenWhisk ofrece una Interfaz de Línea de Comandos (CLI) llamada “wsk” para crear, ejecutar y administrar entidades OpenWisk fácilmente, permitiendo instalar cualquier sistema operativo que facilite a los desarrolladores las implementación e interacción con las plataformas. Apache OpenWhisk se puede implementar y configurar utilizando contenedores y desplegarse tanto en Kubernetes como en OpenShift (Quevedo et al., 2019). Es utilizado por IBM Cloud Functions, por lo que es una de las soluciones más estables y probadas. Cuenta con una versión de pago con más funcionalidades.

2.5.8 Fission

Fission (figura 9) es una plataforma *serverless* para Kubernetes centrándose en la productividad y el alto rendimiento bajo la licencia Apache. Esta plataforma proporciona herramientas para la gestión y facilita el despliegue de código. Docker y Kubernetes se abstraen bajo su funcionamiento, pero se puede extender a Fisión ya que facilita la posibilidad de crear diferentes imágenes de contenedores o mejorar entornos ya existentes. Proporciona una rápida inicialización gracias a que mantiene los contenedores precargados en un pool.



Figura 9. Logo Fission

Está desarrollado con Go, es extensible a cualquier idioma y las partes específicas del lenguaje están aisladas. Actualmente, soporta entornos de desarrollo en Python, NodeJS, PHP, C#, Go, Ruby, Bash y entornos Linux (Fission, 2021b). Fission se integra con Istio, que es una plataforma abierta para conectar, administrar y asegurar microservicios, donde los usuarios también tienen la capacidad de monitorizar el uso de funciones y rastrear la latencia de las solicitudes a través de paneles (Fission, 2021a).

2.5.9 OpenFaaS

OpenFaaS (figura 10) es la plataforma FaaS basada en contenedores Docker liderada por Alex Ellis. Su desarrollo incluye la participación de toda una comunidad de desarrolladores y soporta diferentes orquestadores, como son Docker Swarm, Kubernetes o OpenShift. Esta plataforma permite a los usuarios ejecutar cualquier tipo de código dentro de una imagen Docker ofreciendo capacidades de auto escalado y métricas por medio de un servicio API GETWAY y Prometheus respectivamente. Soporta entorno de desarrollo en NodeJS, CoeScript, Go, Python, Java, .NET Core, R, y Shell Script.



Figura 10. Logo OpenFaaS

Tiene la posibilidad de realizar peticiones HTTP a cualquier función desplegada en Docker a través de un servidor HTTP dentro de cada contenedor. Cuando una función finaliza dentro de esta plataforma, se transforma en una petición HTTP y se devuelve.

2.6 Análisis de la arquitectura serverless y el concepto FaaS

La arquitectura *serverless* y el concepto de FaaS han sido explotados por diferentes empresas líderes del mercado, que proporcionan una gran variedad de plataformas que siguen estos modelos y que se encuentran en un estado de madurez avanzado, siendo una opción accesible para los usuarios finales. Cada una de las opciones que encontramos tienen sus peculiaridades ya sea por la tecnología utilizada, su implementación o el modelo de programación. Sin embargo, encontramos cuatro aspectos en los que todos intentan centrarse, que son los siguientes (Shahrad et al., 2019):

Facilidad de Uso. Se trata de una de las características esenciales de un FaaS, y de cualquier servicio en general, ya que, por definición, una arquitectura *serverless* debe facilitar al usuario la ejecución del código sin necesidad de realizar ningún tipo de gestión de la infraestructura. Todas ellas proporcionan facilidad a la hora de poder migrar su lógica a nuevas plataformas, ofreciendo herramientas y mecanismos para que

la transición sea sencilla para el usuario. Por otro lado, se intenta facilitar una guía de uso para su gestión y así reducir la dificultad para aquellos usuarios que no tengan preparación específica para estos procesos.

Rendimiento. Es otro de los aspectos prioritarios de un *FaaS*, y uno de los más variables, ya que depende de diferentes factores dentro del contexto de cada despliegue, porque está sujeto a la infraestructura donde se instale. Para que un *FaaS* tenga un buen rendimiento con respecto a otro, debemos analizar las necesidades de hardware idóneas de acuerdo con el coste teórico de ejecución. De esta forma, el tiempo de respuesta dependerá tanto de la fluidez de la interacción con el *FaaS* como el coste que tiene ejecutar el código.

Alta disponibilidad. Esta es una de las características base que destaca en las soluciones *open-source*. Si se desea desplegar el servicio en un entorno local, *a priori* no sería una de las características más importantes, pero siempre debemos tener en cuenta la necesidad de soportar una carga masiva de peticiones en cualquier servicio. Debemos recordar que la alta disponibilidad va sujeta a la facilidad de replicación, la redundancia y la tolerancia a fallos de los diferentes componentes que encontramos en el sistema.

Por último, uno de los aspectos que tratan varias soluciones es **el uso eficiente de los recursos**. En muchas ocasiones entra en contraposición con el rendimiento, ya que la aplicación de una reducción de consumo suele asociarse con una reducción del rendimiento. Esta característica se acentúa según aumenta la escalabilidad del sistema.

2.7 Industria 4.0 y cloud computing

El objetivo de integrar las soluciones *cloud* dentro de la industria han supuesto uno de los puntos de evolución más destacados en los últimos años. La premisa de la que parte es la aplicación de estas arquitecturas y su estructura de servicios dentro de los sistemas de fabricación. Este concepto se ha dado a conocer como *Manufacturing Cloud* (MCloud) (Lu et al., 2014) y evolucionó a *Cloud Based Manufacturing* (CBM)(Wu et al., 2014). Encontramos tres áreas donde el *cloud computing* se puede aplicar en empresas de fabricación.

- **Recopilación y análisis de datos:** Se centra en las operaciones y la gestión de equipos, aplicación de Big Data en la fabricación y el refuerzo proporcionado por el uso de un modelo de datos para combinar datos comerciales (como transacciones de inventario o financieras) con datos operativos (como alarmas, parámetros de proceso y eventos de calidad) y datos internos y externos (como

datos de clientes, proveedores, Web y máquinas) a través de herramientas analíticas avanzadas.

- **Inter-factory collaboration:** Interoperabilidad a nivel de cadena de suministros, gestión del transporte, negociación de proveedores o contratos. Los socios pueden crear módulos de *cloud computing* para abordar otros problemas de fabricación como, por ejemplo, la ejecución de la cadena de suministro y programación de la producción.

Aquí encontramos el modelo SCOR. Este modelo se centra en la función de gestión de la cadena de suministro desde una perspectiva de proceso operativo e incluye interacciones con el cliente, transacciones físicas e interacciones con el mercado (Zhou et al., 2011).

- **Computación y simulación:** Utilizar modelos digitales para probar virtualmente los sistemas de fabricación, comprender mejor el entorno empresarial a través de la inteligencia empresarial y tomar decisiones.

Los sistemas de producción tienen una estructura jerárquica, donde los equipos de fabricación se organizan a diferentes niveles, desde el nivel de máquina, hasta el de fábrica (Unver, 2013). Apoyándose en esta organización en niveles, se han desarrollado diferentes modelos de referencia, como RAMI 4.0 (Electro Industria, 2016) que son la base para definir la arquitectura de sistemas basados en modelos IaaS, PaaS.

La interoperabilidad es un requisito esencial tanto para los proveedores de servicios como para las empresas. Los servicios deben guardar un mínimo de interoperabilidad para garantizar su adaptación, transferencia o conexión con otros servicios, plataformas o aplicaciones tanto dentro del propio sistema como con los sistemas de los diferentes proveedores. Entre otras ventajas, esta interoperabilidad garantiza una fácil migración, una mejor integración con otras aplicaciones que se utilicen y un mejor intercambio de datos entre diferentes proveedores de servicios en la nube (Pedone & Mezgár, 2018).

Debido a la explotación del paradigma *cloud* en la industria 4.0, esta tecnología nos ofrece servicios a través de la red proporcionando información en la cadena de producción. La creación de redes globales de recursos y procesos de producción, así como el uso de aplicaciones interconectadas globalmente, demandan del uso de una serie de características o pautas. A continuación, veremos algunas iniciativas relevantes y arquitecturas de referencia en la creación de organizaciones empresariales dentro del ámbito industrial.

2.8 Iniciativas relevantes y arquitecturas de referencia dentro de la industria 4.0

Para el desarrollo de una arquitectura dentro del ámbito de industrial, debemos conocer el concepto de *Industrial Internet* y su contexto, también conocido como industria 4.0. Estas arquitecturas de referencia (AR) sirven de enlace entre los arquitectos de sistemas y los diferentes participantes en la cadena de fabricación, como puede ser el personal de planta, los diferentes ingenieros, consultores de negocio, etc.

Existen varios grupos de trabajo que se centran en ofrecer diferentes modelos de arquitectura de referencia, teniendo en cuenta diversas perspectivas o *viewpoints*. Por un lado, Industrial Internet Consortium (IIC) que propone la Arquitectura de Referencia de Internet Industrial (IIRA) (Lin, Miller, et al., 2017) y, por otro lado, Open Fog. En paralelo, un grupo de trabajadores del sector industrial 4.0 presentan el Modelo de Arquitectura de Referencia para la Industria 4.0 (RAMI 4.0). Todas estas arquitecturas son hoy las principales fuentes de información a la hora de desarrollar y diseñar sistemas en entornos de industria 4.0 o Internet Industrial (IIS). No obstante, la base de estos modelos es desarrollar un entorno colaborativo, interoperable y que permita la obtención de datos para mejorar la eficiencia de todos los procesos dentro del ámbito industrial.

Cualquiera de los dos modelos presentados anteriormente, apoyan la digitalización de la industria por medio de dispositivos IoT. Utiliza los servicios, el personal y las máquinas como componentes centrales para descomponer las funciones y los procesos en subprocesos más sencillos, intuitivos y funcionales. A continuación, los definiremos.

2.8.1 Industrial Internet Reference Architecture (IIRA)

IIC nace como una asociación global sin ánimo de lucro de la industria, el mundo académico y el gobierno, en marzo de 2014. Lo forman un conjunto de miembros de diferentes perfiles, desde pequeños y grandes innovadores tecnológicos, líderes del mercado, hasta investigadores de universidades y organizaciones gubernamentales.

El objetivo principal es reunir a todas las organizaciones y tecnologías necesarias para acelerar el crecimiento del IIS juntando, testeando y promoviendo mejores prácticas. En este contexto, todos sus miembros trabajan en colaboración para acelerar el uso comercial de todas estas tecnologías avanzadas, ofreciendo ayuda a los usuarios de la tecnología, los proveedores y los integradores de sistemas para conseguir resultados tangibles. Por otro lado, también se centran en dar a las diferentes

organizaciones toda la orientación necesaria para aplicar estratégicamente las tecnologías digitales con la intención de acelerar la transformación digital del sector.

Industrial Internet Reference Architecture (IIRA) fue publicado por el *Industrial Internet Consortium* (IIC) en el documento "*The Industrial Internet of Things Volume G1: Reference Architecture*" (Lin, Miller, et al., 2017) y contiene conceptos arquitectónicos, vocabulario, estructuras, patrones y una metodología para abordar los problemas de diseño. Este documento define e identifica la arquitectura fundamental y los problemas de diseño de diferentes puntos arquitectónicos basándose en la norma *ISO 42010-2011* (ISO.ORG, 2011).

IIRA trata de identificar los problemas de arquitectura más importantes y comunes. Para ello proporciona una plantilla arquitectónica y una metodología que los desarrolladores pueden usar para resolver los problemas de diseño. Por otro lado, sugiere una metodología a seguir para abordar las principales preocupaciones que tienen los diseñadores a la hora de obtener la información mediante sistemas de Internet Industrial de las Cosas (IIoT) y así evitar que se pierdan importantes pilares de la arquitectura. Es decir, IIRA nos presenta una plantilla arquitectónica que nos permite categorizar los requisitos del sistema IIoT y diseñar una arquitectura propia para abordarlos. Además de hacer hincapié en la interoperabilidad y el despliegue de las tecnologías IoT.

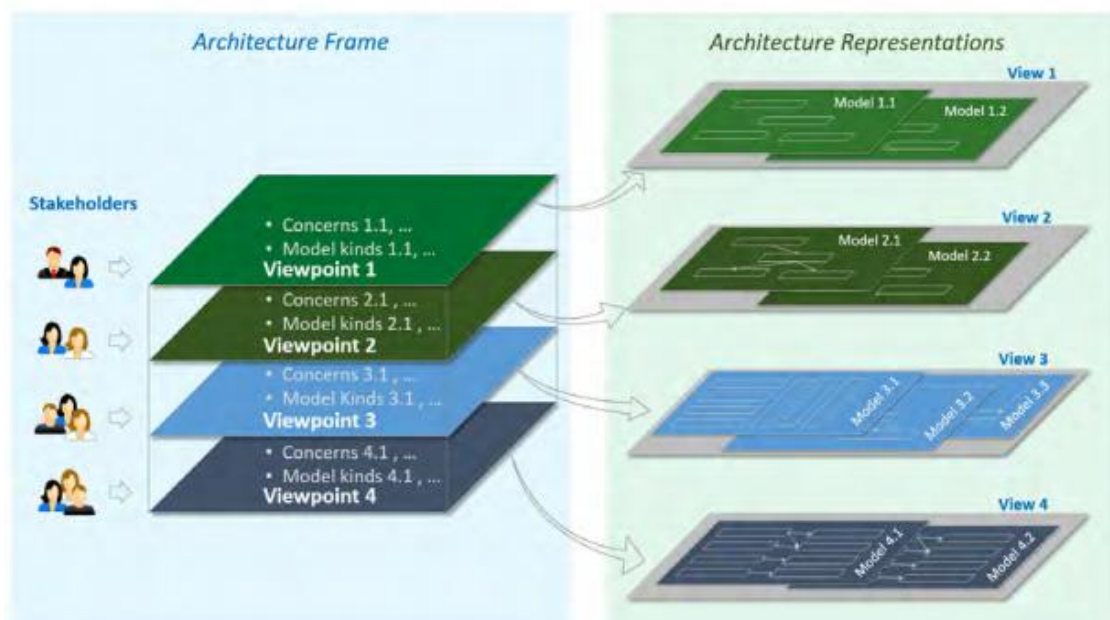


Figura 11. IIRA Architecture Framework (Lin, Miller, et al., 2017)

El núcleo de la metodología IIRA está formado por un conjunto de herramientas denominados viewpoint (perspectivas) que permiten a los arquitectos resolver e

identificar los problemas clave del diseño. Para ello, comienza con la definición de las formas de una arquitectura de IIC a partir de los *viewpoints* de las partes interesadas que están dispuestas en un orden determinado. De esta forma, las decisiones de un punto de vista marcan los requisitos de los *viewpoints* posteriores.

Como podemos ver en la figura 11, IIRA nos presenta un modelo por capas que se divide en cuatro perspectivas: negocio, uso, funcional e implementación. Para definirlos, se centra en las capacidades desde la perspectiva del proceso de negocio y la del software (Lin, Murphy, et al., 2017):

- **Perspectiva empresarial:** Identifica las partes interesadas que participan en el desarrollo, despliegue y funcionamiento del sistema IoT incluyendo todos los objetivos empresariales. Este *viewpoint* tendrá en cuenta el contexto empresarial y normativo en el que opera el sistema IoT.
- **Perspectiva de uso:** Se centra en el uso de IoT, donde se ilustra en base a secuencias de actividades que pueden ser realizadas por los diferentes *stakeholders*.
- **Perspectiva funcional:** Especifica la funcionalidad de los sistemas IoT. Ilustra los componentes funcionales que forman todo el sistema junto con sus interfaces e interacciones, incluyendo interacciones con módulos externos.
- **Perspectiva implementación:** Esta formado por las tecnologías que deben utilizarse para la implementación de los componentes funcionales junto con información sobre su ciclo de vida y las conexiones entre ellos.

Aunque las cuatro perspectivas (*viewpoints*) son de gran trascendencia, el más importante de ellos en la realización de un sistema IoT es el funcional. IIRA especifica una serie de características denominadas dominios funcionales que a su vez se construyen mediante agrupaciones en bloques verticales. Estos se dividen en cinco dominios funcionales (control, operaciones, información, aplicación y negocio). También existen funciones transversales que son independientes del dominio, entre las que encontramos la conectividad, la gestión de datos distribuidos, el análisis industrial y el control inteligente.

Por otro lado, IIRA proporciona unas pautas de cómo debe ser la implementación segura, protegida a la hora de realizarla. Esta arquitectura la divide en tres niveles (*Edge*, plataforma y empresa), que desarrollaremos más adelante.

2.8.2 OpenFog

OpenFog Consortium es una iniciativa público-privada nacida en 2016 y que comparte muchas similitudes con el IIC. Entre algunos de los miembros fundadores que encontramos está Cisco, Dell o Microsoft. El objetivo es encontrar una solución ante los problemas que aparecían a la hora de desarrollar y desplegar de *Edge Computing* y *Cloud Computing*, así como problemas con la latencia y los assets en el *Edge*.

El desarrollo del Internet de las Cosas (IoT) ha llevado a la necesidad de procesar los datos allí donde se crean debido al retraso de respuesta que puede generar la red. Todos los datos generados por los dispositivos de los usuarios se transfieren a la nube para ser almacenados o procesados. Este modelo de computación no es práctico debido a que aumentaría las latencias de comunicación cuando se conecten millones de dispositivos a Internet en el futuro (Hong & Varghese, 2019). En muchos casos, los sistemas requieren una interacción de baja latencia con su entorno o con los usuarios ya que el posible aumento de esta latencia disminuye la calidad de servicio (QoS) y la calidad de la experiencia (QoE). Aquí es donde nace el concepto del modelo alternativo de *Fog/Edge Computing*.

El *Edge computing* se define como los procesos computacionales que se realizan dentro de los dispositivos de borde (dispositivos IoT) con capacidad de análisis y procesamiento, como los Pc compactos, las Raspberrys, los routers o las pasarelas de red. Al procesar la información obtenida cerca de donde se creó, se reducen las latencias y se consume menos ancho de banda. Sin embargo, la computación de estos dispositivos es limitada en comparación con la nube. La computación de borde se caracteriza por su baja latencia, su densa distribución geográfica, la información de contexto de la red, el conocimiento de la ubicación y la proximidad (Hassan et al., 2018).

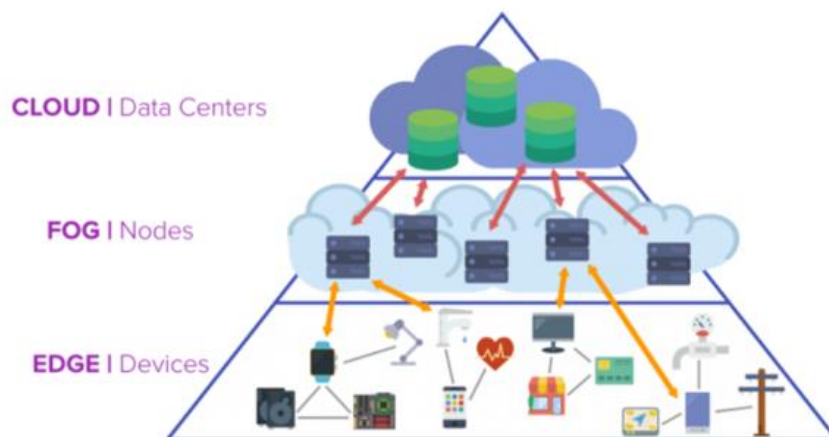


Figura 12. Fog/Edge computing (ErpNews, 2018)

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

El concepto de *Fog Computing* se refiere a una estructura de red descentralizada en la que los recursos (datos, aplicaciones, etc.) se sitúan en un lugar lógico entre la nube y la fuente de datos generada (Gedeon et al., 2019). Es decir, acerca los servicios que analizan dichos datos a la fuente. El *Fog Computing* surge como una extensión del *Cloud Computing* (que consiste en un servidor distante donde almacenamos datos, aplicaciones, etc.) para complementarse satisfaciendo esas necesidades entre los dispositivos de los usuarios y los centros de *Cloud Computing*. *Fog Computing* se caracteriza por la baja latencia, la interoperabilidad, la interacción en tiempo real, el conocimiento de la ubicación y el apoyo a la interacción en línea con la nube.

Su arquitectura sigue una estructura de pirámide (figura 12). La nube está en la cima de la pirámide, donde se alojan muchos servidores. Ofrece capacidades de computación o almacenamiento. Entre la computación de borde y la de nube, se encuentra la plataforma *Fog*, que está compuesta por nodos en diferentes áreas. En la base se encuentran los gadgets de los usuarios finales (ordenadores portátiles, smartwatches, etc.) y diferentes dispositivos IoT que pueden llegar a un nodo *Fog* con baja latencia (Confais et al., 2017).



Figura 13. Pilares OpenFog

La arquitectura de referencia de *OpenFog* pretende ayudar a los arquitectos ingenieros y líderes de las empresas a entender los requisitos específicos que pueden aplicarse en los nodos *fog* en un escenario determinado. Esta arquitectura se rige por un conjunto de Pilares (figura 13), que serán los atributos clave necesarios para

conseguir la definición de *OpenFog*. A partir de ellos se genera una arquitectura horizontal, a nivel de sistema distribuido de las funciones de computación, almacenamiento, control y red más cerca de la fuente de datos (usuarios, dispositivos, sensores, etc.) a lo largo de la nube (*OpenFog*, 2017).

Los pilares que conforman la arquitectura horizontal de *OpenFog* son:

- **Pilar de seguridad:** La seguridad es uno de los temas importantes en cualquier solución de IoT. Este pilar describe todos los mecanismos que pueden aplicarse para que un nodo de *fog* sea seguro.
- **Pilar de la escalabilidad:** Este pilar aborda las necesidades técnicas y empresariales dinámicas que hay detrás de los despliegues de *fog*. La escalabilidad se centra en los nodos *fog* internos, a través de la adición de hardware o software, así como en los externos, donde las redes de *fog* deben ser escalables a través de la adición de nuevos nodos de *fog* para ayudar en la operación de carga pesada o el almacenamiento y la conectividad de red, para ampliar la red *fog* en general
- **Pilar de apertura:** La apertura es esencial para el éxito de las soluciones informáticas de *fog* ubicuas. El bloqueo de la propiedad o de los proveedores puede tener como resultado que sólo se pueda disponer de proveedores limitados, lo que influye negativamente en los costes, la calidad y la innovación del sistema.
- **Pilar de la autonomía:** La autonomía permite a los nodos de la *fog* seguir proporcionando la funcionalidad designada incluso durante los fallos de los servicios externos. La toma de decisiones se basará en todos los niveles del sistema, incluso cerca del dispositivo.
- **Pilar de la programabilidad:** La programabilidad permite desplegar diferentes aplicaciones en los nodos de *fog* y la posibilidad de modificar el sistema tanto en las capas de software como en las de hardware.
- **Pilar de fiabilidad, disponibilidad y capacidad de servicio (RAS):** La fiabilidad, la disponibilidad y la capacidad de servicio (RAS) son de vital importancia. Estos temas se refieren al hardware, al software y a las aplicaciones que se ejecutan en el nodo. La fiabilidad define que el nodo de *fog* continuará ofreciendo su funcionalidad designada tanto en comportamientos normales como inesperados. La disponibilidad define el funcionamiento continuo (gestión, control, orquestación, etc.) del sistema, que a menudo se mide en tiempo de actividad. Por último, la



Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

servicialidad se refiere a proporcionar un funcionamiento correcto al sistema, lo que significa que la aplicación hace lo que se supone que debe hacer.

- **Pilar de agilidad:** El pilar de la agilidad aborda las decisiones operativas del negocio para un despliegue de *OpenFog* RA. La agilidad se centra en la transformación de los datos en información necesaria para las acciones dentro del sistema. También aborda la naturaleza altamente dinámica de los despliegues de fog y la necesidad de responder rápidamente a los cambios tanto dentro de la red, como en el despliegue o al aportarse nuevos datos o nuevas solicitudes.
- **Pilar de la jerarquía:** La jerarquía computacional y del sistema no es necesaria para todas las arquitecturas de *OpenFog*, pero aún se expresa en la mayoría de los despliegues. Muchos despliegues tienen una jerarquía diferente, que varía desde el nivel de la nube hasta el nivel del taller. La jerarquía depende completamente de la aplicación en la que se desplegarán los nodos de *fog*, pero es importante para la construcción del sistema.

2.8.3 Modelo de Arquitectura de Referencia para la Industria 4.0 (RAMI 4.0)

El modelo de Arquitectura de Referencia para la Industria 4.0 (RAMI 4.0) surge a partir de un grupo de colaboradores de diferentes empresas e instituciones de I+D europeas denominadas "*Plattform Industrie 4.0*" con el objetivo de agrupar y representar los diferentes aspectos de la industria en un único modelo común, es decir, que contenga la integración vertical, de extremo a extremo y horizontal (Pisching et al., 2018). Este modelo desarrolló su primera versión en abril 2015, convirtiéndose en una de las referencias para el uso de la industria 4.0. Surgió casi al mismo tiempo que cuando ICC promovió la IIRA.

El objetivo principal de la arquitectura RAMI 4.0 es proporcionar servicios a otros componentes a través de un protocolo de comunicación dentro de la red, es decir, una arquitectura orientada a servicios (SOA), proporcionando una disponibilidad de toda la información en tiempo real a través de la conexión en red de todas las instancias durante su ciclo de vida. Este proceso de administración se denomina *Asset Administration Shell* (AAS) (Danielle, 2020) y es un modelo que intenta crear un puente entre el usuario y los dispositivos IoT integrando *assets*. Un *asset* es todo aquello que puede conectarse para implementar una solución dentro de la industria 4.0

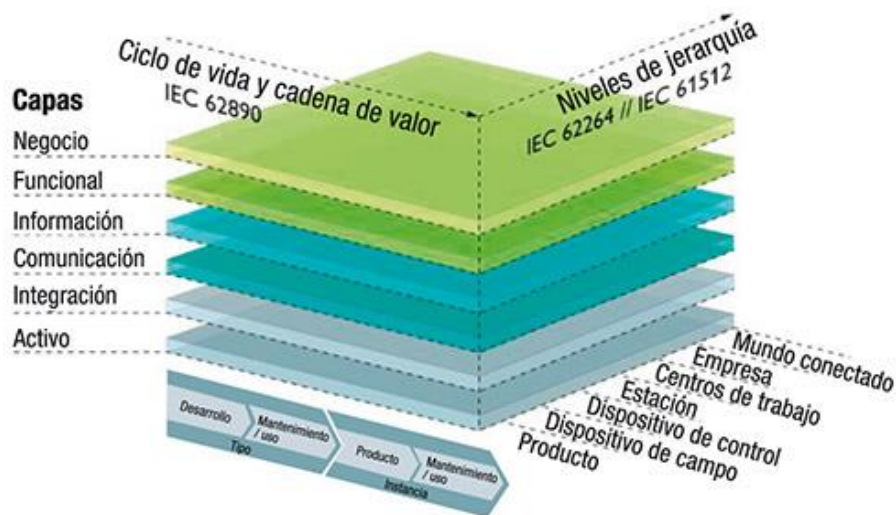


Figura 14. Arquitectura RAMI 4.0

RAMI 4.0 es un modelo tridimensional que combina el ciclo de vida y la cadena de valor estructurándolo de forma jerárquica para los diferentes componentes TI dentro de la industria 4.0 (figura 14). Este concepto nos agrupa la lógica de funciones y el mapeo de interfaces y estándares, permitiendo implementar soluciones flexibles usando este modelo de arquitectura de referencia.

En la parte superior, en el eje horizontal izquierdo, encontramos el ciclo de vida y cadena de valor. Esta parte del modelo representa el ciclo de vida de los productos o sistemas dentro de la cadena de valor haciendo distinción en la parte inferior entre “Tipo” e “Instancia”. Adopta e integra los modelos de referencia de IEC 62890 que definen la gestión de los ciclos de vida en productos y sistemas, así como en su control, medición y automatización dentro del proceso industrial. El eje horizontal derecho de este modelo está formado por cinco niveles de automatización más dos niveles que agrega, el “Mundo conectado” en la parte superior y el “producto” en la capa inferior. De esta forma adopta todas las definiciones de referencia de las normas IEC 61512 y IEC 62264.

En el eje horizontal nos encontramos con seis capas o niveles llamados: activo, integración, comunicación, información, funcional y de negocio. La función de cada una de las capas es:

- **La capa de activos:** En esta capa nos encontramos todos los elementos físicos de un sistema. Desde las piezas del producto, hasta los sensores, documentos, equipos de producción incluyendo los seres humanos. Cada valor activo representado en esta capa tiene una representación virtual en las capas

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

superiores. Esta capa incluirá la interfaz con la que interactúan los trabajadores y la relación con la capa superior (integración).

- **La capa de integración:** Esta capa se encarga de integrar la información del mundo real al informático.
- **La capa de comunicación:** Esta capa es la responsable de la comunicación estandarizada entre la capa de integración y la de información. Por tanto, su función es transmitir archivos y datos desde la integración, proporcionando formato de datos a la capa de información. También proporciona servicios para controlar la capa de integración.
- **La capa de información:** Esta capa se encarga de mantener los datos necesarios de forma estructurada e integrada, proporcionando las interfaces para acceder a los datos de la capa funcional. Además, es responsable de procesar, integrar y mantener la persistencia de datos, así como de describir la relación que existe entre los datos y la funcionalidad técnica del activo.
- **La capa funcional:** En esta capa se describen las funciones lógicas y técnicas del activo de forma digital mediante una plataforma. Aparte describe el mapeo del modelo de la capa de negocio que debe ajustarse a las entradas de la capa funcional.
- **La capa de negocio:** Esta capa es la encargada de orquestar los servicios proporcionados por la capa funcional. Modela las reglas de negocio y las restricciones legales del sistema.

2.8.4 ZDMP

Zero Defect Manufacturing Platform (ZDMP) (figura 15) es un proyecto que busca proporcionar una plataforma que ayude a las fábricas a alcanzar el objetivo de cero defectos en su producción, con un alto nivel de interoperabilidad para hacer frente al concepto de fábricas

conectadas. Para ello, combina tecnologías y software de código abierto, de forma que los usuarios finales puedan conectar sus sistemas para beneficiarse de ello. Por otro lado, simplifica parte del proceso mediante conexiones preexistentes y nuevas con sensores y dispositivos, y con el sistema, de forma que se pueda monitorizar cada parte y gestionar la plataforma. Esto permite conectar los sistemas de planta y ERPs para mejorar las características de la plataforma y aumentar la calidad de los productos y de la producción.



Figura 15. Logo ZDMP

Centrándonos en ofrecer un servicio de Algoritmos dentro de la industria, vamos a focalizarnos en la solución que realizamos en el CIGIP. Este componente es la de “Prediction and Optimisation Run-time” (ZDMP, 2021) y proporciona en tiempo de ejecución los algoritmos de procesamiento de datos y los integra en la plataforma ZDMP. Utiliza las herramientas de gestión de datos del entorno propios y proporciona una API fácil de usar a través de los otros componentes de ZDMP (ICE, 2019).

2.8.5 AI4EU

AI4EU (figura 16) es un consorcio creado en enero de 2019 con la intención de construir la primera plataforma y ecosistema europeo de inteligencia artificial bajo demanda. Para ello propone compartir los recursos de IA producidos en proyectos europeos incluidos servicios de alto nivel, experiencia en investigación e innovación de IA, componentes y conjuntos de datos de IA y recursos informáticos de alta potencia. Todos estos recursos estarán almacenados en un único lugar para su libre acceso (plataforma web) con el objetivo de acelerar las innovaciones basadas en IA (Cortés et al., 2019).



Figura 16. Logo AI4EU

Centrándonos en ofrecer un servicio de Algoritmos dentro de la industria, de esta plataforma podemos extraer diferentes algoritmos de inteligencia artificial que luego podemos ofrecer como servicio dentro de la cadena de producción.

2.8.6 Comparativa de las arquitecturas de referencia.

Las diferentes arquitecturas de referencia en el despliegue de *Frameworks* dentro de la industria 4.0 ha sido desarrollada por un conjunto de organizaciones, ya sean privadas o públicas líderes en el sector, que buscan innovar, desarrollar e investigar los mejores métodos para la implementación y el desarrollo de sistemas dentro de la industria. Cada una de las opciones que encontramos tienen sus peculiaridades ya sea por los servicios que puede ofrecer, el objetivo central de la cadena de producción o las metodologías para abordar los diferentes problemas. Sin embargo, debemos centrarnos en alguna de estas opciones para tomar una decisión sobre la arquitectura de referencia que queremos utilizar para desarrollar nuestro servicio.

La decisión tomada será *Industrial Internet Reference Architecture*, debido a que IIRA nos presenta una plantilla arquitectónica que nos permite categorizar los requisitos del sistema y diseñar una arquitectura propia para abordarlos. Aunque, debido a la interoperabilidad que busca la industria 4.0, podemos encontrar muchas similitudes con la arquitectura referencial de RAMI 4.0 (figura 17).

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

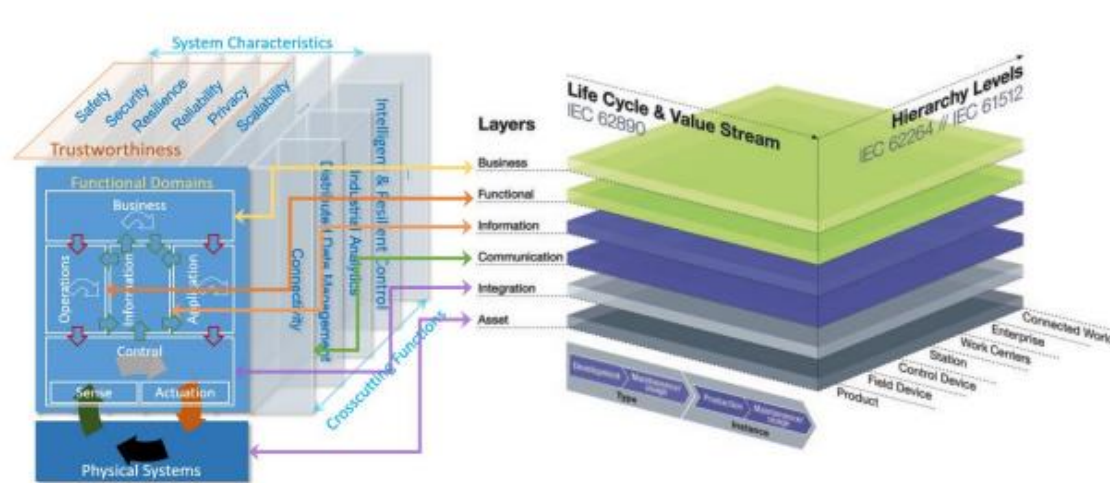


Figura 17. IIRA vs RAMI 4.0

Para nuestra arquitectura, analizaremos los diferentes *viewpoints* de la capa de negocio en los que destacaremos tanto los *stakeholders* como los requisitos de estos; veremos los diferentes puntos de vista mediante un diagrama de uso, la parte funcional y su implementación.

2.9 Arquitectura de referencia

La arquitectura de referencia que tendrá nuestro servicio será Industrial Internet Reference Architecture (IIRA) la cual se divide en cuatro *viewpoints* (figura 18), con la intención de mostrar una arquitectura iterativa entre los diferentes procesos que pueden ir implicados. Estos *viewpoints* son: *business viewpoint*, *usage viewpoint*, *functional viewpoint* e *implementation viewpoint*.

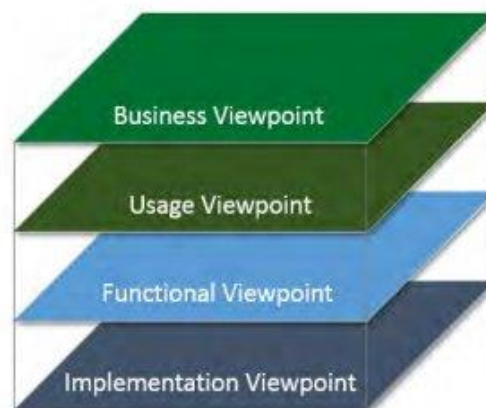


Figura 18. IIRA: Arquitectura

2.9.1 Business Viewpoint

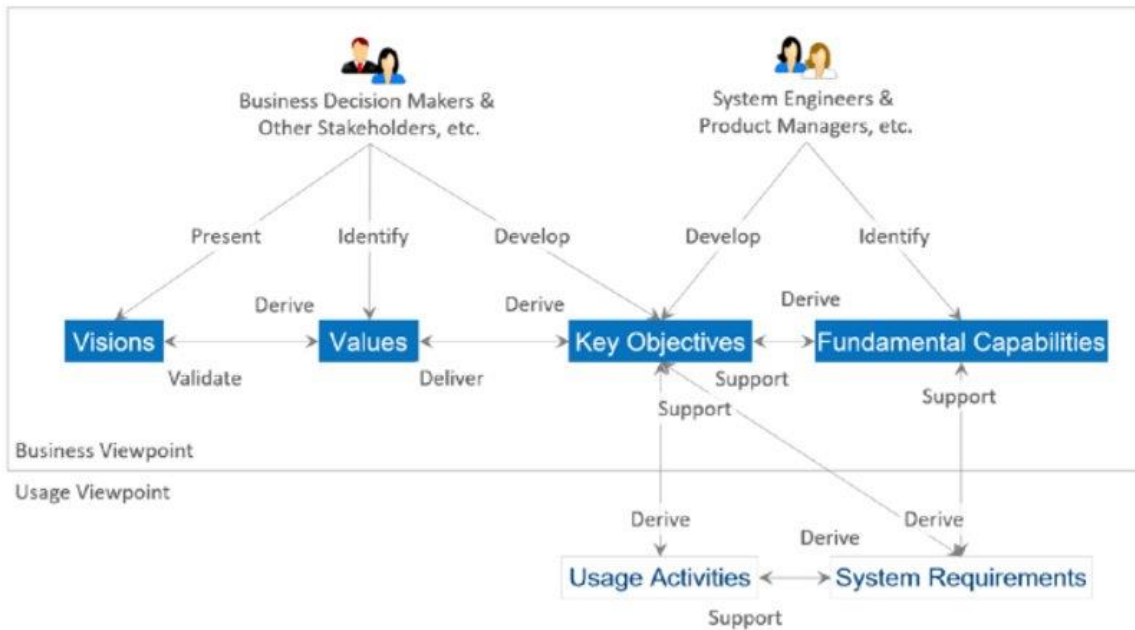


Figura 19. IIRA: Business viewpoint

Uno de los aspectos principales de este *viewpoint* es el análisis desde la perspectiva empresarial. Para ello, es necesario incorporar en el diseño los requisitos que se acercan a las necesidades reales que surgen a la hora de realizar una operación desde el punto de vista de la empresa. Basándonos en el documento de IIRA, definiremos los elementos que debemos considerar a la hora de desarrollar nuestra nueva arquitectura (figura 19) los cuales son:

- **Stakeholders:** Son los actores que tienen un interés en el negocio y una fuerte influencia en su dirección, incluyen a quienes impulsan la concepción y el desarrollo de los sistemas dentro de la organización.
- **Visión:** Describe el futuro de una organización y define la dirección del negocio
- **Valor:** Justifica por qué la visión tiene importancia para las partes interesadas, así como para los usuarios del sistema resultante.
- **Objetivos clave:** Son los resultados cuantificables de alto nivel técnico.
- **Procesos en los que centrarse:** Detallan las capacidades esenciales del sistema para completar las tareas específicas de la empresa.

2.9.2 Usage Viewpoint

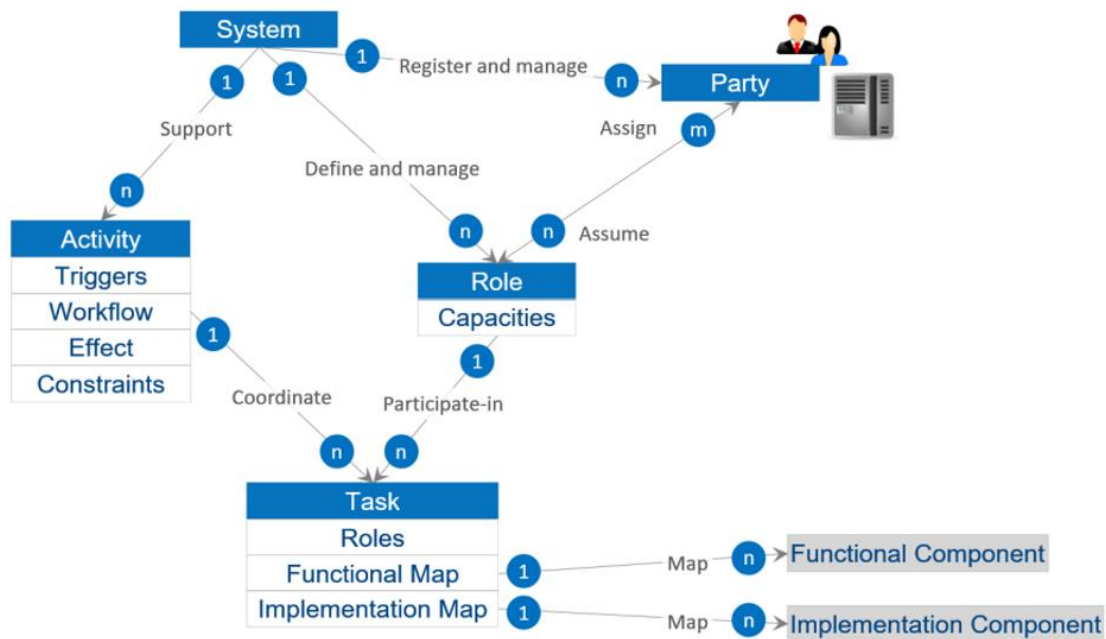


Figura 20. IIRA: Usage viewpoint

En esta capa de la arquitectura (figura 20) definiremos los requisitos del sistema, así como las tareas de los diferentes *stakeholders*. Esto nos servirá de referencia tanto en el desarrollo del *functional viewpoint*, como en el diseño del *implementation viewpoint*. Los *stakeholders* se definen, según el documento de IIRA, como todos los agentes, tanto humanos como automatizados, que tienen una autonomía, interés y responsabilidad en la ejecución de tareas.

El *Usage viewpoint* desarrolla los siguientes conceptos:

- **Definición de tareas:** Son las piezas de trabajo que componen una actividad y que son ejecutadas por una parte con un rol.
- **Definición de roles:** Es el conjunto de capacidades asumidas por una o varias partes.
- **Diseño de partes:** Los agentes involucrados.
- **Diseño de actividad:** Define cómo se utiliza el sistema y de qué se componen las tareas. Se inicia por un disparador, sigue con un flujo de trabajo que causa un efecto en el estado del sistema y tiene restricciones (integridad de los datos, confidencialidad, etc.).
- **Diseño de los sistemas:** El sistema que estamos desarrollando.

Este *viewpoint* sirve de punto de entrada para el *functional viewpoint*.

2.9.3 Functional viewpoint

Este punto se centra en los componentes funcionales de nuestro servicio. Considera su estructura, interfaz e interacción del servicio con sus usos y actividades. Estos elementos están coordinados con el *usage viewpoint* y apoyado por el *business viewpoint*.

El punto de vista funcional es un punto de vista de la arquitectura que enmarca las consideraciones relacionadas con las capacidades funcionales y la estructura de un sistema del Internet Industrial de las cosas (IIoT) y sus componentes. Este *viewpoint* lo podemos dividir en cinco dominios funcionales.

Los flujos de datos y los flujos de control tienen lugar entre estos dominios funcionales. La figura 21 ilustra cómo los dominios funcionales se relacionan entre sí con respecto a los flujos de datos y control. Las flechas verdes muestran cómo circulan los flujos de datos entre dominios. Las flechas rojas muestran cómo circulan los flujos de control a través de los dominios. Otras flechas horizontales ilustran algún proceso que tiene lugar dentro de cada dominio.

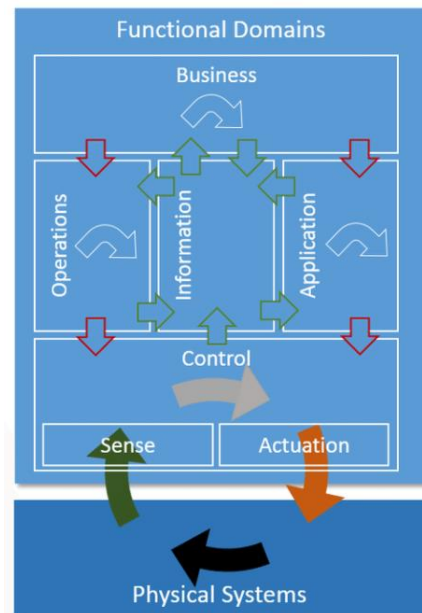


Figura 21. IIRA: Functional Viewpoint

1. **Dominio de Control:** Es un conjunto de funciones que se realizan en los sistemas de control industrial. Las funciones se implementan cerca de los sistemas físicos que lo controlan.
2. **Dominio de operaciones:** Son soluciones cuya función consiste en aprovisionar, gestionar, supervisar y optimizar las soluciones del dominio de control.
3. **Dominio de la información:** Son soluciones cuya función es recoger datos de varios dominios, especialmente del dominio de control, y analizarlos para adquirir inteligencia de alto nivel sobre el sistema global.
4. **Dominio de aplicación:** Conjunto de funciones que implementan la lógica, las reglas y los modelos de aplicación a un nivel alto y detallado para la optimización de la aplicación.
5. **Dominio de negocio:** se trata de un conjunto de funciones centradas en las operaciones de extremo a extremo que utiliza el servicio.

2.9.4 Implementation viewpoint

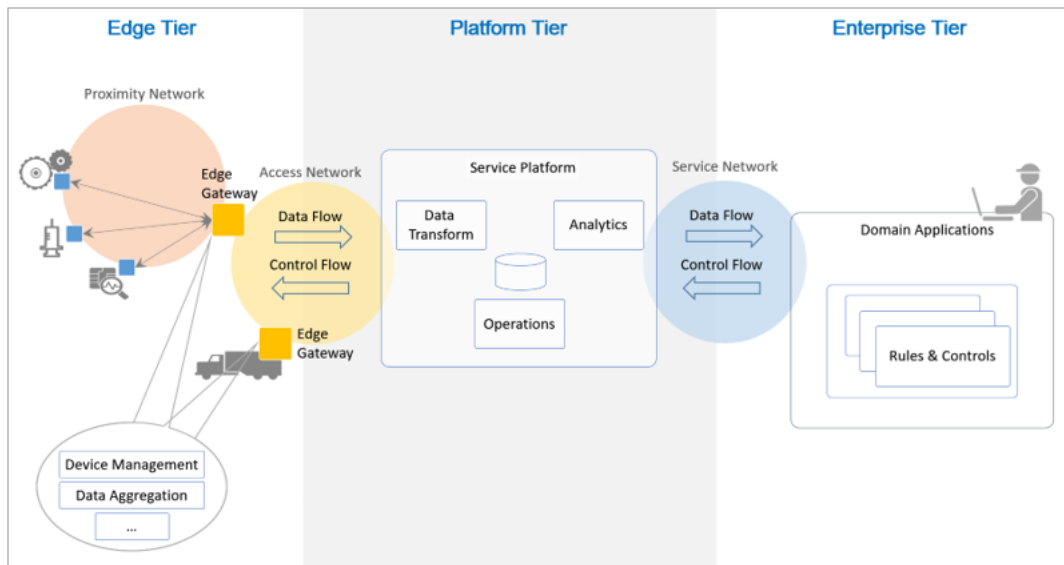


Figura 22. IIRA: Implementation viewpoint

En este punto de vista se define técnicamente y en detalle toda la arquitectura del sistema. Para ello deberemos ver los componentes necesarios para dichas tecnologías, los esquemas de comunicación entre los procedimientos de su ciclo de vida y la relación entre nuestro sistema y los elementos del entorno. Estos elementos están coordinados por el usage viewpoint y apoyado por el business viewpoint.

Como podemos ver en la figura 22, el *implementation viewpoint* comprende diferentes niveles que desempeñan funciones específicas en el procesamiento del flujo de datos y el control de las actividades implicadas. Estos niveles son tres:

- **Edge tier:** Recoge todos los datos de los nodos que se encuentran en el *Edge*, y utilizan la red de proximidad.
- **Platform tier:** Recibe, procesa y envía los comandos de control del Enterprise tier al *Edge* tier.
- **Enterprise tier:** Implementa las aplicaciones específicas del dominio, los sistemas de apoyo para la toma de decisiones y proporciona las interfaces a los usuarios finales.

Estos tres niveles están conectados por tres redes:

- **Proximity Network:** Conecta los diferentes sensores o dispositivos.
- **Access Network:** Conecta los flujos de datos con el flujo de datos de control entre los Tier de *Edge* y *Platform*.

- **Service Network:** Conecta los servicios en los Tier de Platform, Enterprise y los de dentro de cada nivel.

La red de servicios permite la conectividad entre los servicios en el nivel de plataforma y el nivel empresarial, y los servicios dentro de cada nivel. Puede ser una red privada superpuesta sobre la Internet pública o la propia Internet, lo que permite el grado de seguridad empresarial entre los usuarios finales y varios servicios. Como podemos ver en la figura 23, los tres niveles de la implementación se combinan con los niveles funcionales comentados anteriormente.

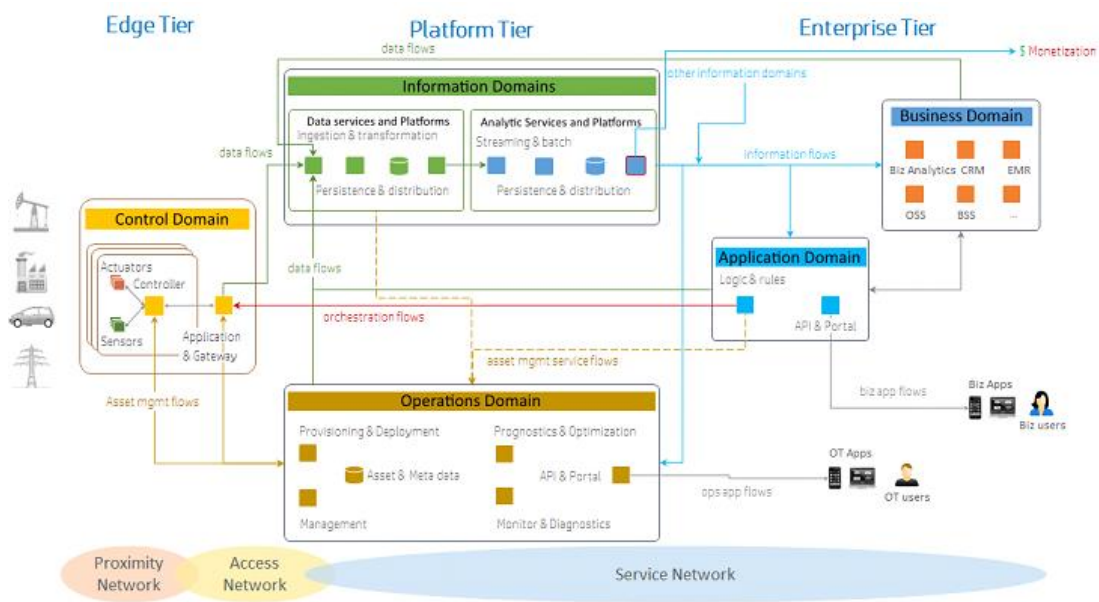


Figura 23. IIRA: Viewpoints

2.10 Análisis de la arquitectura para AaaS

La industria 4.0 está diseñada para proporcionar soluciones software capaces de analizar en tiempo real diferentes procesos dentro de la cadena de producción que generen datos. Este volumen de datos masivos, también conocidos como Big Data, se generan y almacenan en bases de datos para su análisis. Gracias a la interconexión que nos presenta la industria 4.0 podemos obtener datos de los distintos puntos de la línea de producción, analizarlos y, a partir de los resultados, tomar decisiones. Los algoritmos son una herramienta que permite automatizar la optimización de procesos dentro de la industria a partir de los datos obtenidos por los sensores.

Un algoritmo se define como un conjunto de instrucciones estructuradas que siguen un orden lógico y cuyos pasos permiten resolver un problema. Hasta el momento

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

encontramos los algoritmos como ficheros alojados en los proyectos en los que se va a utilizar de forma específica y que son actualizados según las necesidades de los usuarios o del mismo proyecto. Como ya hemos mencionado, estos algoritmos son fundamentales a la hora de resolver problemas de optimización, de calidad o de tiempo, ya que muchos de ellos pueden ser utilizados en diferentes situaciones o máquinas. Aquí es donde nace la idea de este proyecto, que busca facilitar su uso e implementación dentro de la Industria.

Con el fin de proporcionar un servicio en donde todos los científicos de datos puedan alojar sus algoritmos y ofrecerlos a todo el ecosistema de la industrial, en este trabajo plantearemos una arquitectura óptima para la implementación de este tipo de servicios. Gracias a la Fabricación en la nube, podemos tratar de ofertar este servicio, de la misma manera que lo hemos visto con los *FaaS*, pero centrándonos en algoritmos. Este servicio proporcionará una API fácil de usar a través de la cual otros componentes puedan utilizar los algoritmos incorporados. El usuario podrá desplegar un algoritmo en el sistema, y este se alojará de forma escalable para que esté disponible.

3. Solución Propuesta

Basándonos en el estado del arte, a continuación, se desarrolla la arquitectura de sistema a través de las diferentes perspectivas (viewpoints) indicados en la sección anterior. Este proyecto trata de diseñar la integración de un sistema de Algoritmos como Servicios (AaaS) para la optimización basándonos en la arquitectura de diseño *Industrial Internet Reference Architecture* (IIRA) que hemos visto anteriormente y teniendo en cuenta el trabajo realizado por el CIGIP en el desarrollo del componente “*Prediction and Optimisation Run-time*”.

Este sistema estará alojado en la nube y ofrecido a las empresas interesadas como un recurso para la optimización de planes de aprovisionamiento, producción y distribución. Las empresas que deseen acceder a este servicio podrán realizarlo mediante un acceso web y podrán ejecutar el algoritmo que tengan desplegado. Los desarrolladores, usuarios o empresas que desplieguen el servicio podrán añadir algoritmos propios en la aplicación siguiendo un modelo y que, por medio de automatización, generará solo los recursos necesarios para desplegarlos en el servicio. Estos algoritmos podrán ser instanciados en diferentes clientes (Bases de datos, aplicaciones de mensajes, almacenamiento de ficheros, etc.) para acceder a los resultados por parte de los usuarios finales de diferentes modos.

El resultado que se busca con este trabajo es desarrollar la arquitectura de este servicio teniendo en cuenta diferentes enfoques como las perspectivas de negocio, de aplicación, de información y tecnología; y preparar el entorno de integración continua para la automatización de integración de algoritmos para poder usarse en el AaaS.

3.1 Plan de trabajo

La amplitud que conlleva desarrollar una arquitectura requiere que se realice un plan de trabajo por fases para poder desarrollar todos los aspectos de forma ordenada. Por ello vamos a plantear el proyecto en 3 etapas de forma que en cada una de ellas haya un conjunto claro de objetivos a cumplir. A continuación, se esboza el contenido de las 3 etapas para posteriormente detallar su alcance.

- **Etapas 1:** Captura de Requerimientos: En esta fase capturaremos los requerimientos que debe tener una empresa para poder desarrollar el sistema de Algoritmos como Servicios.

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

- **Etapa 2:** Vista de Negocio y de Operación. En esta etapa se definirá la vista del negocio y la vista del usuario del Sistema de Algoritmos como Servicio, teniendo en cuenta los requerimientos de la Etapa 1.
- **Etapa 3:** Vista Funcional y de implementación. En esta parte se definirán la vista funcional y la vista de implementación del sistema de Algoritmos como Servicio, teniendo en cuenta los requerimientos de la Etapa 1.

3.2 Plan detallado

Etapa 1: Captura de Requerimientos

Esta Etapa tiene como objetivo la documentación de todos los requerimientos que tiene una empresa sobre las funcionalidades a implementar con el sistema de Algoritmos como Servicio. Para ello lo dividiremos en tres fases:

1. **Definición de requerimientos Funcionales:** Esta fase tiene como tarea definir los requerimientos funcionales tanto a alto nivel como a bajo. Por ello, el objetivo es entender el alcance de las funcionalidades, analizar los requerimientos que las empresas demandan de nuestro servicio y tener una visión clara de los distintos sistemas de información con los que el sistema desarrollado debe interactuar.
2. **Definición de Requerimientos No Funcionales:** Esta fase tiene como tarea definir los requerimientos no funcionales tanto a alto nivel como a bajo. Por ello, el objetivo será entender el alcance de las características no funcionales solicitadas por la empresa y analizar los requerimientos no funcionales que las empresas demandan para así definir las estrategias de desarrollo del software.
3. **Definir casos de uso:** Esta fase tiene como tarea definir los casos de uso a los que va enfocado el sistema.

Etapa 2: Vista de Negocio y de Operación

Esta etapa tiene como objetivo desarrollar el punto de vista empresarial para evitar el riesgo de desarrollar una tecnología centrada en el desarrollador. Esto permite incorporar en el diseño los requisitos y necesidades operativas del mundo real. Para ello, esta etapa se centrará en enmarcar la visión, los valores y los objetivos clave del AaaS desde el punto de vista empresarial y de las partes interesadas. Una vez definida esta vista de negocio se concretarán las tareas, roles y responsables y los mapas funcionales y de implementación, completando la vista de Operación. Para conseguirlo lo dividiremos a su vez en dos fases:

1. **Vista de negocio:** En esta fase definiremos la visión, los valores y los objetivos clave del AaaS. El objetivo es incorporar en el diseño del AaaS los requisitos y necesidades operativas del mundo real.
2. **Vista de Operación:** En esta fase definiremos las tareas, roles y responsables, mapas funcionales y de implementación. El objetivo es asegurar una correcta planificación para el desarrollo en implementación del AaaS.

Etapas 3: Vista funcional y de implementación

El objetivo de esta etapa es descomponer el sistema AaaS en sus dominios de control, operaciones, información, aplicaciones y negocios e identificar los flujos de datos, decisiones y solicitudes que circulen entre ellos. Una vez realizada esta descomposición, se profundizará en los controles, coordinación y orquestación ejercidos desde cada uno de estos dominios, así como en las diferentes operaciones típicas de estos dominios. Para ello dividiremos esta etapa en otras dos fases:

1. **Vista funcional:** En esta fase diseñaremos el dominio de control, operación, información, aplicación y de negocio e identificaremos los flujos de datos. El objetivo es descomponer el sistema de Algoritmos como Servicios en sus dominios.
2. **Vista de Implementación:** En esta fase diseñaremos los controles, los sistemas de coordinación y el sistema de control. El objetivo es definir el diseño para que la empresa pueda iniciar el desarrollo e implementación de nuestro sistema.



4. Diseño de la Solución

En esta sección vamos a diseñar toda la solución siguiendo el plan de trabajo detallado en el apartado anterior.

4.1 Etapa 1: Captura de Requerimientos

En esta etapa capturaremos los requerimientos que debe tener nuestro servicio para la empresa que quiera integrar el sistema de Algoritmos como Servicios. Como hemos visto anteriormente, lo dividiremos en tres fases.

4.1.1 Fase 1: Definición de requerimientos Funcionales

Un requerimiento funcional de un sistema es una función multidisciplinar que se establece entre los dominios del adquisidor y del proveedor o desarrollador para establecer y mantener los requisitos que debe cumplir el sistema, software o servicio de interés. Los requerimientos funcionales se ocupan de descubrir, obtener, desarrollar, analizar, verificar (incluidos los métodos y la estrategia de verificación), validar, comunicar, documentar y gestionar los requisitos (ISO 29148, 2018). En esta fase, vamos a especificar los requisitos funcionales teniendo en cuenta los derivados de nuestro sistema de Algoritmos como Servicio, el análisis del estado de arte que hemos realizado, los conocimientos que hemos aprendido y la experiencia propia que tenemos.

Siguiendo las recomendaciones de la ISO 29148, adaptaremos todos los requisitos para que sean comprensibles y entendibles en cualquier contexto de nuestro sistema. Es por ello, que todos los requerimientos se definirán con la siguiente estructura para tener una forma coherente.

[Asunto] + [Acción] + [Restricción de la acción]

Teniendo en cuenta esta estructura, los requerimientos funcionales (RF) del sistema son:

- **RF1:** El sistema de Algoritmos como Servicios (AaaS) [Asunto], contará con niveles de seguridad [Acción], para poder acceder a cada una de las partes del servicio [Restricción de la acción].

- **RF2:** Cuando un usuario desarrolle un algoritmo [Asunto] deberá contar con una plantilla que le proporcione el esqueleto de un modelo o algoritmo [Acción] para que este sea compatible con la plataforma [Restricción de la acción].
- **RF3:** Cuando un usuario desee desarrollar un algoritmo [Asunto] deberá disponer de la documentación necesaria para poder implementar un nuevo modelo o algoritmo [Acción] para que este sea compatible con la plataforma [Restricción de la acción].
- **RF4:** Cuando un usuario desarrolle, edite o elimine un algoritmo [Asunto], este se desplegará, eliminará o actualizará automáticamente el servicio en el servidor donde se encuentran almacenados los algoritmos [Acción] para que puedan acceder todos los usuarios finales [Restricción de la acción].
- **RF5:** Antes de desplegar “añadir el algoritmo al servicio” [Asunto], el sistema debe ser capaz de analizar el código y comprobar la calidad de este [Acción], para evitar errores al ser utilizados por el usuario final [Restricción de la acción].
- **RF6:** Todos los algoritmos desarrollados [Asunto] deben contener un archivo README.MD [Acción] para entender por parte de la empresa y el usuario final cual es la finalidad de este [Restricción de la acción].
- **RF7:** El sistema final debe tener actualizado el servicio [Asunto], por lo que los servidores cargarán siempre las últimas versiones [Acción], para que tengan todos los algoritmos actualizados [Restricción de la acción].
- **RF8:** Cuando una empresa desee realizar un despliegue de alguno de los algoritmos [Asunto] deberá poder acceder a la imagen [Acción] para poder ser desplegada en la plataforma deseada [Restricción de la acción].
- **RF9:** Cuando un usuario final desea acceder a un Algoritmo desplegado por la empresa [Asunto], deberá acceder desde su red mediante una petición a un API [Acción] para tener una respuesta con el resultado del algoritmo [Restricción de la acción].
- **RF10:** Todos los algoritmos generados [Asunto] deben guardarse en un repositorio de contenedores [Acción] para que se pueda acceder a ellos desde cualquier punto [Restricción de la acción].
- **RF11:** Cuando un usuario quiera información de un Algoritmo [Asunto], debe poder tener acceso mediante API [Acción] para obtener información más específica de qué hace y cómo se utiliza [Restricción de la acción].



Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

- **RF12:** Cuando un usuario final utiliza un algoritmo [Asunto], debe poder conectarlo a otros componentes del sistema [Acción] para poder aumentar las posibilidades de uso del algoritmo [Restricción de la acción].
- **RF13:** La empresa que instala el servicio en la organización [Asunto] debe poder conocer el funcionamiento de la línea de producción y maquinaria [Acción] para poder desplegar los mejores algoritmos para cada circunstancia [Restricción de la acción].
- **RF14:** Cuando un usuario final utilice un algoritmo [Asunto] podrá testearlo en un entorno de pruebas [Acción] para ver si el impacto en la industria es efectivo [Restricción de la acción].
- **RF15:** La empresa que quiera desarrollar algoritmos [Asunto] debe poder instanciarlos directamente en el servicio siguiendo unas pautas [Acción] para poder ofrecer mejoras a los usuarios finales en el mismo entorno ya desplegado [Restricción de la acción].
- **RF16:** Cuando un usuario termine un algoritmo [Asunto], este se deberá construir solo por medio de integración continua [Acción] para ganar interoperabilidad entre sistemas de diferentes organizaciones [Restricción de la acción].
- **RF17:** Cuando un desarrollador desee crear un algoritmo para el sistema [Asunto] debe poder tener acceso a una guía interactiva e intuitiva [Acción] para que le permita entender cómo funciona y desarrollar el algoritmo de forma correcta [Restricción de la acción].
- **RF18:** La empresa que va a implementar el servicio [Asunto] debe proporcionar un documento técnico al usuario final [Acción] para que pueda tener acceso a todas las utilidades del algoritmo como servicio [Restricción de la acción].
- **RF19:** La empresa que implementa el servicio [Asunto] debe proporcionar todas las medidas de seguridad [Acción] para un correcto y seguro intercambio de datos [Restricción de la acción].
- **RF20:** La empresa que implementa el servicio [Asunto] debe poder extender el servicio a cualquier plataforma que pueda desplegar contenedores [Acción] para mayor capacidad de atracción de las empresas [Restricción de la acción].

4.1.2 Fase 2: Definición de requerimientos No Funcionales

Los requisitos no funcionales son aquellas funcionalidades que especifican criterios para evaluar las operaciones de un servicio de tecnología de la información, en contraposición al requerimiento funcional, que trata de especificar el comportamiento que tiene una aplicación. La definición de estos requisitos no funcionales está basada en la ISO 9126 (Behkamal et al., 2009). Estas ISOS nacen de la búsqueda de la calidad del software partiendo de la vista del producto. Es decir, nos presenta un conjunto de modelos de calidad en seis categorías: Funcionalidad, Fiabilidad, Usabilidad, Eficiencia, Facilidad de mantenimiento y portabilidad (figura 24).

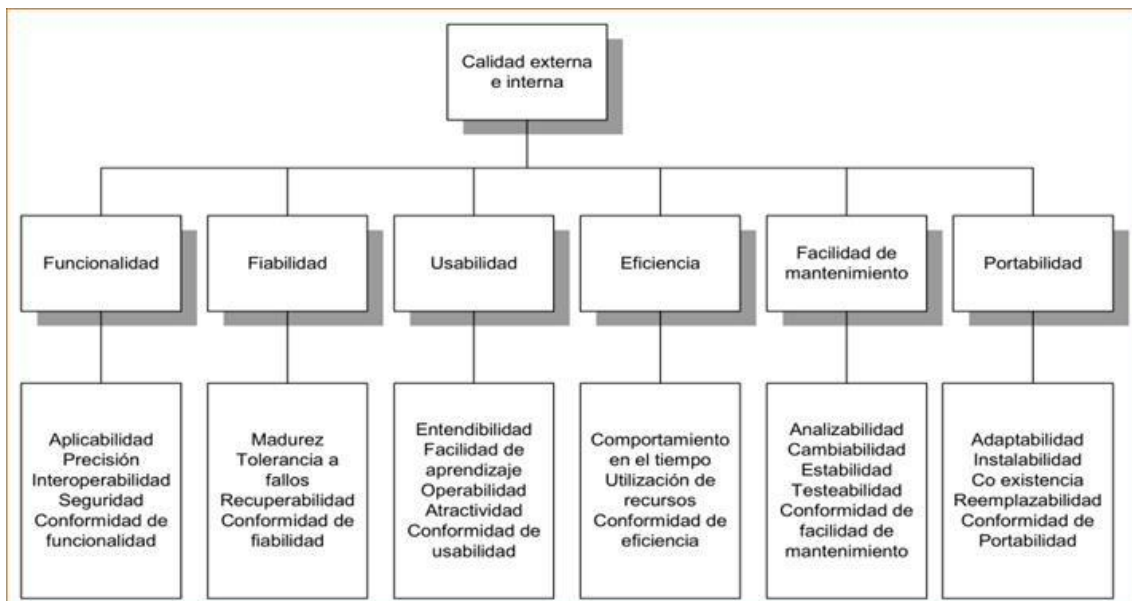


Figura 24. Requisitos no funcionales de la ISO 9126:2001

La finalidad de la ISO 9126:2001 es ofrecer una ventana de evaluación de la calidad del software desde la perspectiva del usuario y los prejuicios humanos que pueden afectar en la percepción de un proyecto. Los requerimientos no funcionales de nuestro sistema que debemos tener en cuenta para nuestro servicio son los siguientes:

- **Funcionalidad:** Nuestro servicio debe presentar seguridad e interoperabilidad. La interoperabilidad mide la capacidad para integrar la aplicación en el resto del sistema de industria aplicando la arquitectura de IIRA. Por otro lado, la seguridad mide la habilidad para evitar el acceso no autorizado.
- **Fiabilidad:** Nuestro servicio debe ser tolerante a fallos y que tenga recuperabilidad, es decir, debe poder levantarse de nuevo en caso de caída o fallo. Así mismo debe tener la capacidad de recuperarse ante la pérdida de datos.

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

- **Usabilidad:** Nuestro sistema debe ofrecer una interfaz sencilla e intuitiva, además de manuales para la fácil implementación de los algoritmos, de modo que permita la facilidad de aprendizaje y entendibilidad. Por otro lado, debe tener un diseño atractivo y que desprenda conformidad de usabilidad, para ello, el desarrollo de la plataforma se deberá realizar junto a desarrolladores con conocimientos de *user experience* (UX).
- **Eficiencia:** Nuestro sistema debe estar formado por un conjunto de atributos que relacionen el desempeño del software con la cantidad de recursos que necesita (es decir, la relación acción/recursos consumidos). Este requisito mide las operaciones por segundo, el consumo de recursos de memoria, procesador o espacio en la memoria. Cuanto menor sean las necesidades del software para toda la funcionalidad que posee, mayor será su eficiencia.
- **Facilidad de Mantenimiento:** En este punto se mide la facilidad con la que podemos dar mantenimiento a nuestro servicio, la facilidad para integrar nuevos mecanismos o requerimientos a la aplicación y corregir diferentes fallos sin dejar de dar servicio. Para ello se desarrollará un plan de integración continua, que permita no solo el despliegue automático, si no que testee la calidad del servicio.
- **Portabilidad:** Define la posibilidad de que el sistema tenga facilidad para su crecimiento en el futuro, es decir, la posibilidad que tengamos de desarrollar más componentes dentro de nuestro sistema. Tendrá en cuenta la escalabilidad del servicio, para que pueda crecer ante un aumento de la carga de trabajo o la necesidad de que lo utilicen más usuarios de forma concurrente. Para ello, levantaremos los servicios con Docker, que pueden ser derivados de forma sencilla a Kubernetes y ahí permitir tanto la escalabilidad, como la disponibilidad de este.

4.1.3 Fase 3: Casos de Uso

Para finalizar la etapa de captura de requerimientos, desarrollaremos los casos de uso (figura 25) definidos en las fases anteriores. Estos diagramas nos proporcionaran una mayor comprensión del sistema, ya que representan las operaciones o tareas que realiza un sistema. Para poder desarrollarlo correctamente, es necesario definir los actores, que en este caso serán tres:

- **Desarrollador:** Se encarga del desarrollo e implementación de los algoritmos.
- **Proveedor tecnológico:** Se trata de una empresa que se encarga de implementar soluciones, en este caso instalar nuestro sistema de Algoritmos como Servicio

para la optimización. Tiene acceso al repositorio con las imágenes de cada uno de los algoritmos.

- **Usuario final:** El usuario que utiliza el resultado del algoritmo dentro de la organización.

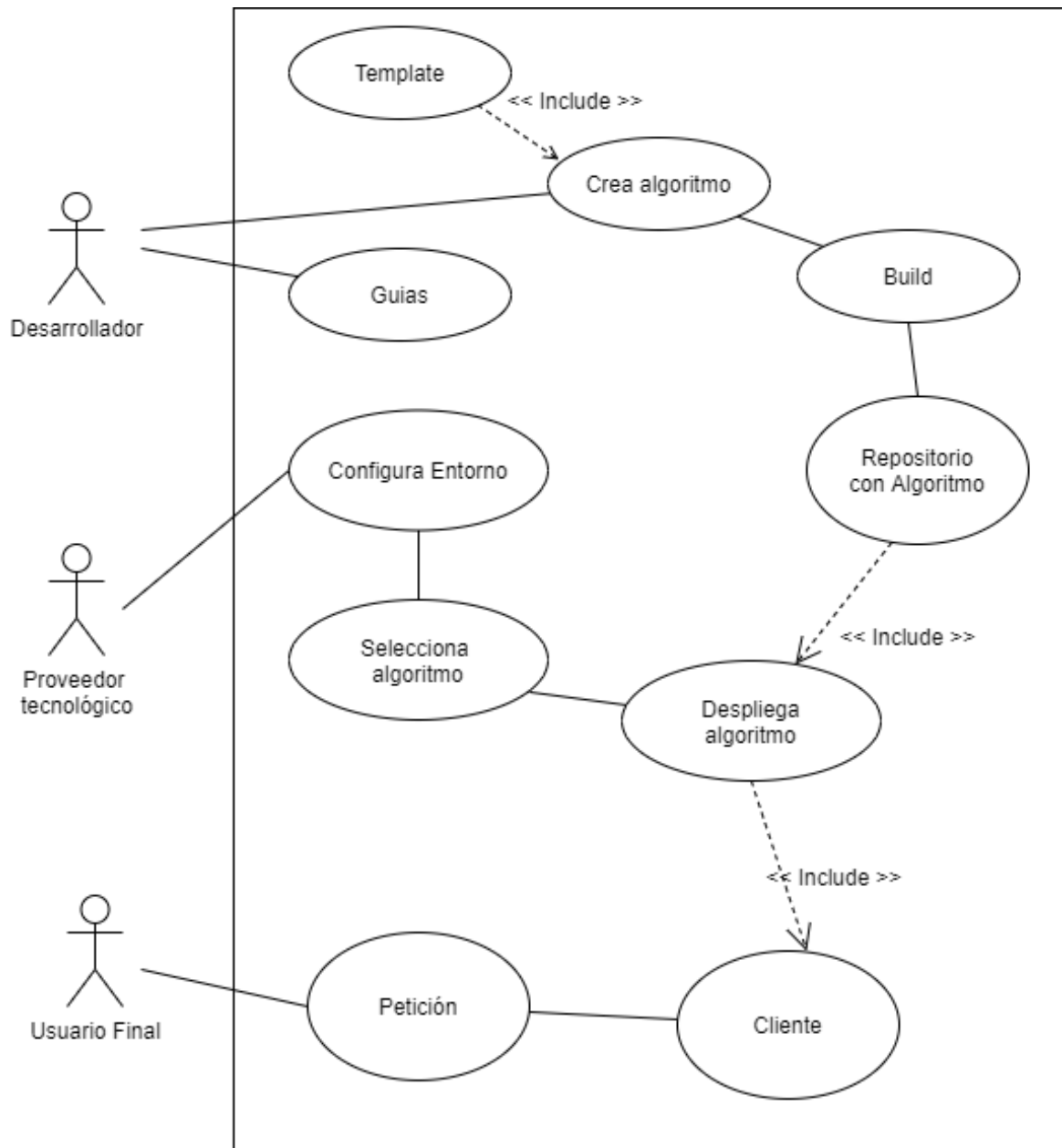


Figura 25. Diagrama de casos de uso

Para entender mejor el funcionamiento de este caso de uso, se puede diseñar un diagrama de proceso de negocio (BPM). Este modelo nos permite describir el entorno en el que un sistema será implantado sin ofrecer detalles de la implementación,

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

permitiendo unir todos los sistemas, métodos, herramientas, técnicas de desarrollo de procesos y la gestión de estos en un sistema estructurado (Garimella et al., 2008).

Para el desarrollo del modelado del sistema de algoritmos como servicio, hemos identificado tres carriles (figura 26). Cada uno de estos carriles identifica a los actores vistos anteriormente.

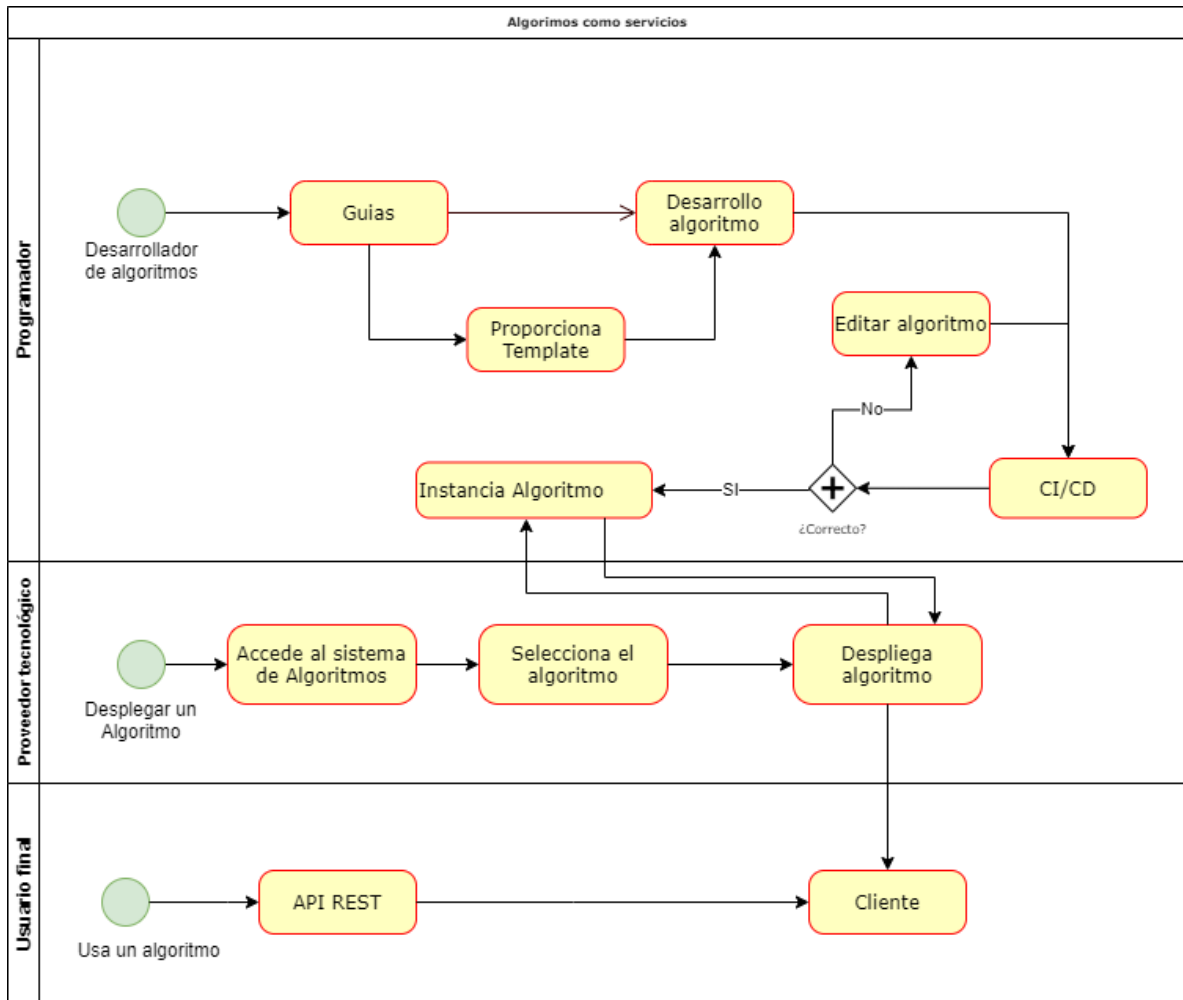


Figura 26. BPM

Teniendo en cuenta el diagrama de casos de uso y el diagrama de proceso de negocio, vamos a tratar de describir cada uno de los casos de uso (tabla 1) y a continuación, con más detalle en las tablas de la 2 a la 11.

Tabla 1. Casos de uso

| Ref. Caso de Uso | DESCRIPCIÓN |
|------------------|--|
| Cu-1 | Guías: Un desarrollador puede acceder a una guía para informarse en que consiste el desarrollo de algoritmos para ofrecerlos de servicio y entender la estructura que debe seguir el proyecto. |
| Cu-2 | Crear Algoritmo: El usuario desarrollará un algoritmo teniendo en cuenta la estructura y funciones necesarias que debe tener dicho algoritmo para poder integrarlo en la plataforma. |
| Cu-3 | Template: Cuando un usuario desarrolle un algoritmo lo debe elaborar a partir del template proporcionado, ya que este viene integrado con las instrucciones necesarias que deberá realizar el sistema a la hora de construir la solución por medio de integración continua. |
| Cu-4 | Build: En el momento que el usuario ha finalizado el algoritmo y sube la solución a su repositorio, se lanza un fichero de integración continua que construye, mide la calidad y crea un contenedor con dicho algoritmo. |
| Cu-5 | Repositorio del algoritmo: Es el lugar donde se almacenará el algoritmo y al que podrán acceder las empresas para desplegarlo como servicio. |
| Cu-6 | Configuración del entorno: La empresa debe preparar y configurar el entorno dentro de la industria donde se vaya a desplegar este servicio. |
| Cu-7 | Selección del algoritmo: Obtiene la dirección el contenedor o un fichero zip con el algoritmo. |
| Cu-8 | Despliegue del algoritmo: Desplegaremos el algoritmo en el servidor. Si en algún momento existe una actualización del algoritmo en el repositorio, este se actualizará automáticamente. |
| Cu-9 | Petición: Acceso del usuario final al resultado del servicio de algoritmos. |
| Cu-10 | Cliente: Es el servicio al que acceden los usuarios finales para obtener los resultados almacenados de los algoritmos. Este cliente puede ser una base de datos, un directorio de almacenamiento o una herramienta de análisis de mensajes Cliente/Servidor, etc.. |



Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

Tabla 2. Caso de uso – Guías

| Cu-1 | | Guías |
|----------------------|--|---|
| Actores asociados | Programador | |
| Requisitos asociados | RF2, RF3, RF15, RF17 | |
| Descripción | Un desarrollador puede acceder a una guía para informarse en que consiste el desarrollo de algoritmos para ofrecerlos de servicio y entender la estructura que debe seguir el proyecto. | |
| Precondición | -- | |
| Secuencia Normal | Paso | Acceso a la plataforma de AaaS. |
| | 1 | El usuario entra en la wiki de AaaS. |
| | 2 | El usuario consulta la documentación para elaborar un algoritmo para un sistema AaaS. |
| Postcondición | -- | |
| Excepciones | Paso | Error |
| | 1 | La web o el manual no está disponible. |
| Comentarios | Este proceso es de documentación para que el usuario se familiarice con todas las posibilidades que tiene el servicio y pueda proporcionar la posibilidad de resolver las dudas que aparezcan. | |

Tabla 3. Caso de uso – Crear algoritmo

| Cu-2 | | Crear algoritmo |
|----------------------|--|---|
| Actores asociados | Programador | |
| Requisitos asociados | | |
| Descripción | El usuario se dispone a desarrollar un algoritmo teniendo en cuenta la estructura y funciones necesarias que debe tener el algoritmo para ser integrado en la plataforma. | |
| Precondición | -- | |
| Secuencia Normal | Paso | Crear algoritmo |
| | 1 | Crea una estructura de ficheros predefinida en la guía o utiliza el template. |
| | 2 | Una vez generado, sube el código a su repositorio. |
| Postcondición | Cu-4 | |
| Excepciones | Paso | Error |
| | 2 | Tiene un fallo de conexión y no puede subir el proyecto al repositorio. |
| Comentarios | Si el desarrollador es nuevo y tiene dudas puede acceder al Cu-3 para obtener un Template y conocer la estructura que debe seguir para añadir el algoritmo. Al finalizar, debe subir al repositorio el código. | |

Tabla 4. Caso de uso – Template

| Cu-3 | | Template |
|----------------------|---|---|
| Actores asociados | Programador | |
| Requisitos asociados | RF2, RF3, RF6 | |
| Descripción | Cuando un usuario desarrolle un algoritmo lo debe elaborar a partir del template proporcionado, ya que este viene integrado con las instrucciones necesarias que deberá realizar el sistema a la hora de construir la solución por medio de integración continua. | |
| Precondición | Cu-2 | |
| Secuencia Normal | Paso | Copiar Template |
| | 1 | Puede descargar de un repositorio o un punto de descarga de ficheros el template que debe utilizar como base para rellenar y crear su algoritmo para el servicio. |
| Postcondición | -- | |
| Excepciones | Paso | Error |
| | 1 | No estar disponible el template o utiliza una versión desactualizada. |
| Comentarios | -- | |

Tabla 5. Caso de uso – Build

| Cu-4 | | Build |
|----------------------|---|---|
| Actores asociados | Programador | |
| Requisitos asociados | RF4, RF5, RF16 | |
| Descripción | En el momento que el usuario ha finalizado el algoritmo y sube la solución a su repositorio, se lanza un fichero de integración continua que construye, mide la calidad y crea un contenedor con dicho algoritmo. | |
| Precondición | Cu-2 | |
| Secuencia Normal | Paso | Build |
| | 1 | Se activa el fichero de Integración continua. |
| | 2 | Pasa una prueba de calidad del código. |
| | 3 | Construye una imagen del algoritmo que guarda en el <i>container repository</i> . |
| Postcondición | Cu-5 | |
| Excepciones | Paso | Error |
| | 2 | No pasa el proceso de calidad |
| | 3 | No pasa el proceso de construcción de la imagen |
| Comentarios | Automatiza todo el desarrollo de despliegue de imágenes para las empresas y proveedores del sistema de algoritmos como servicio. | |

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

Tabla 6. Caso de uso – Repositorio del algoritmo

| Cu-5 | | Repositorio del algoritmo | |
|----------------------|---|---|--|
| Actores asociados | Programador | | |
| Requisitos asociados | RF7, RF10 | | |
| Descripción | Es el lugar donde se almacena el algoritmo y al que podrán acceder las empresas para desplegarlo como servicio. | | |
| Precondición | Cu-4 | | |
| Secuencia Normal | Paso | Almacenamiento del código | |
| | 1 | Proporciona un enlace de acceso a la imagen Docker. | |
| | 2 | Proporciona un enlace de descarga del paquete con el algoritmo. | |
| Postcondición | Cu-8 | | |
| Excepciones | Paso | -- | |
| Comentarios | Es el punto final del caso de uso que afecta al desarrollador de algoritmos. | | |

Tabla 7. Caso de uso – Configuración del entorno

| Cu-6 | | Configuración del entorno | |
|----------------------|--|--|--|
| Actores asociados | Proveedor tecnológico | | |
| Requisitos asociados | RF13, RF20 | | |
| Descripción | La empresa debe preparar y configurar el entorno dentro de la industria donde se vaya a desplegar este servicio. | | |
| Precondición | -- | | |
| Secuencia Normal | Paso | Instanciación | |
| | 1 | Debe ser un entorno que permita el lanzamiento de contenedores Docker. Algunos ejemplos son: Azure, Google, AWS, ZDPM. | |
| Postcondición | Cu-7 | | |
| Excepciones | Paso | Error | |
| | 1 | Que no pueda desplegarse un entorno de contenedores dentro de la organización donde se desea instalar un sistema para proporcionar Algoritmos como Servicio. | |
| Comentarios | -- | | |

Tabla 8. Caso de uso – Selección del algoritmo

| Cu-7 | | Selección del algoritmo |
|----------------------|---|--|
| Actores asociados | Proveedor tecnológico | |
| Requisitos asociados | RF7, RF8 | |
| Descripción | Obtiene la dirección del contenedor con el algoritmo. | |
| Precondición | Cu-6 | |
| Secuencia Normal | Paso | Accede |
| | 1 | Obtiene ya sea por fichero o contenedor, el algoritmo con la estructura definida para ofrecerlo como servicio. |
| Postcondición | Cu-7 | |
| Excepciones | Paso | Error |
| | 1 | Que el fichero o contenedor no esté con la estructura adecuada o falle. |
| Comentarios | | |

Tabla 9. Caso de uso – Despliegue algoritmo

| Cu-8 | | Despliegue del algoritmo |
|----------------------|---|--|
| Actores asociados | Proveedor tecnológico | |
| Requisitos asociados | RF14 | |
| Descripción | Despliegue del algoritmo en el servidor. Si en algún momento existe una actualización del algoritmo en el repositorio, este se actualizará automáticamente. | |
| Precondición | Cu-7, Cu-5 | |
| Secuencia Normal | Paso | Despliegue |
| | 1 | Añade al contenedor la imagen de Docker y le proporciona una puerta de enlace para el usuario final. |
| Postcondición | -- | |
| Excepciones | Paso | Error |
| | 1 | No funciona la imagen con el algoritmo. |
| Comentarios | Este es el proceso en el que la empresa proporciona el algoritmo como servicio en las diferentes plataformas, generando un punto de acceso para el usuario final. | |

Tabla 10. Caso de uso – Petición

| Cu-9 | | Petición |
|----------------------|--|--|
| Actores asociados | Usuario Final | |
| Requisitos asociados | RF9, RF11 | |
| Descripción | Es el servicio al que acceden los usuarios finales para obtener los resultados de los algoritmos, almacenados. | |
| Precondición | | |
| Secuencia Normal | Paso | Petición |
| | 1 | El usuario final realiza una llamada a la API con la configuración del cliente, el algoritmo deseado y los datos de entrada. |
| | 2 | El usuario final recibe los datos. |
| Postcondición | Cu-10 | |
| Excepciones | Paso | Error |
| | 1 | Error en los datos enviados. |
| Comentarios | | |

Tabla 11. Caso de uso – Cliente

| Cu-10 | | Cliente |
|----------------------|--|---|
| Actores asociados | Usuario Final | |
| Requisitos asociados | RF9, RF12, | |
| Descripción | Es el servicio al que acceden los usuarios finales para obtener los resultados almacenados de los algoritmos. Este cliente puede ser una base de datos, un directorio de almacenamiento o una herramienta de análisis de mensajes Cliente/Servidor, etc. | |
| Precondición | Cu-9, Cu-8 | |
| Secuencia Normal | Paso | actualización |
| | 1 | Accede al resultado de los datos procesados por parte del Algoritmo. Este cliente puede depender de como el algoritmo devuelva los resultados, ya sea por mensaje, archivos, bases de datos, etc. |
| Postcondición | Cu-10 | |
| Excepciones | Paso | |
| | 1 | Que la configuración y selección del cliente no sea el adecuado para el algoritmo utilizado. |
| Comentarios | | |

4.2 Etapa 2: Vista de Negocio y de Operación

En esta etapa analizaremos la vista de negocio y operaciones que tiene este servicio de Algoritmos, que van alineados con las capas superiores de los *viewpoint* de la arquitectura de referencia IIRA.

4.2.1 Fase 1: Vista de negocio

En esta fase definiremos la visión, los valores y los objetivos clave del AaaS. El fin es incorporar en el diseño del AaaS los requisitos y necesidades operativas del mundo real.

En primer lugar, es necesario definir los *stakeholders*. Un *Stakeholder* es cualquier grupo o individuo que pueda afectar o resultar afectado por el logro de los objetivos de una organización. Los *stakeholders* incluyen a empleados, clientes, proveedores, accionistas, bancos y otros grupos que puedan ayudar o dañar la organización (R. Edward, 2010).

Debido a que este sistema de Algoritmo como servicio (AaaS) va orientado a la industria, se utilizará el estándar ISA-95 para entender cómo está organizada dicha industria. Este estándar fue desarrollado por el *International Society of Automation* en 1990, buscando unir las funciones empresariales con los sistemas de control en un intento de reducir los costes y errores a la hora de implementar interfaces entre diferentes sistemas.

El estándar ISA-95 separa la funcionalidad (figura 27) de la empresa dividiéndola en tres capas (Junín Durán de Leon et al., 2016):

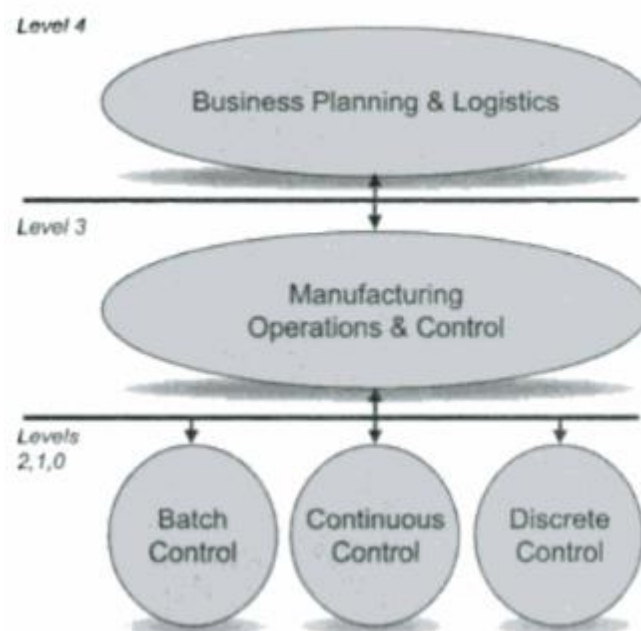


Figura 27. ISA-95 Model

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

- *Business Planning and Logistics*. Se trata de la capa superior (nivel de Planificación), que estructura toda la información Empresarial y Logística.
- *Manufacturing, Operation and Control*. Es la capa intermedia (nivel de Ejecución), que integra todas las operaciones de manufactura y control de la información.
- *Control level*. La última capa (Nivel de Control) que integra todos los elementos de la cadena de producción.

Teniendo en cuenta la distribución de las capas de ISA-95, este sistema de Algoritmos como Sistemas lo podemos localizar en el nivel de control y operaciones de fabricación e información. En este nivel (donde encontramos actividades como la programación de la producción de la planta, la gestión de operaciones, etc.) el estándar ISA-95 especifica un modelo funcional completo para integrar las capas de negocio y de fabricación y define la información a intercambiar entre los niveles 3 y 4 de la pirámide (Prades et al., 2013). A partir de este modelo, podemos describir 4 categorías que se muestran en la zona sombreada de la figura 28:

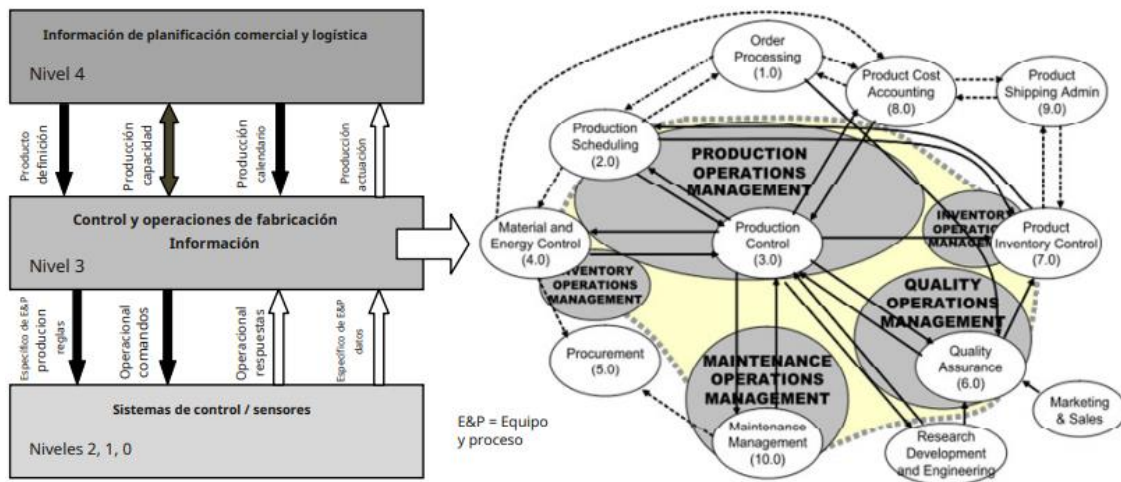


Figura 28. Isa-95 y Stakeholders

- **Gestión de las operaciones de producción:** incluye las actividades de control de la producción (3.0) que operan como funciones de Nivel 3 y el subconjunto de la programación de la producción (2.0) que operan como funciones de Nivel 3 y que se muestran en la Figura 1.
- **Gestión de las operaciones de mantenimiento:** incluye las actividades de gestión del mantenimiento (10.0) que operan como funciones de Nivel 3.

- **Gestión de operaciones de calidad:** incluye las actividades de garantía de calidad (6.0) que operan como funciones de Nivel 3.
- **Gestión de las operaciones de inventario:** incluye las actividades de gestión del inventario y del material, el control del inventario de productos (7.0) y las actividades de control del material y la energía (4.0) que se definen como funciones de Nivel 3.

No obstante, la estructura del modelo no refleja una estructura organizativa empresarial dentro de una compañía, sino que es un modelo de actividades. Diferentes empresas asignan las responsabilidades de las actividades o subactividades a diferentes grupos organizativos empresariales (ISO British Standards Institution, 2007). Además, pueden existir otras categorías dentro de una organización. Teniendo en cuenta estos modelos, se va a realizar una estructura organizativa empresarial dentro de una compañía donde utilizarán este sistema de Algoritmos como servicio.

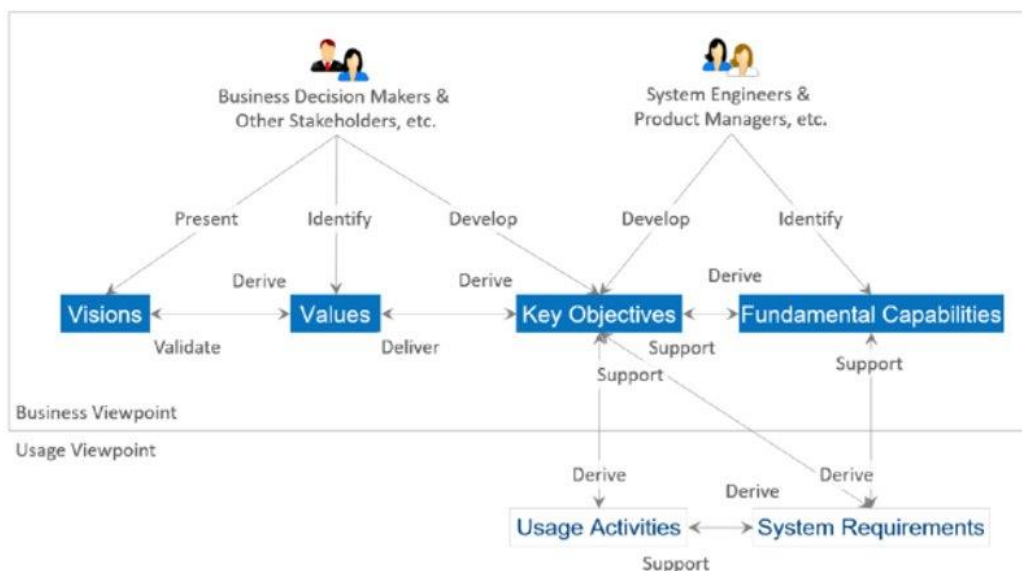


Figura 29. IIRA: Business viewpoint

Siguiendo la arquitectura referencial de IIRA (figura 29), se pueden identificar dos tipos de *stakeholders*. Por un lado, los primarios, que son las personas y organizaciones que buscan, reciben y gestionan las aplicaciones para mejorar la calidad; por otro, los secundarios, que son los usuarios que proveen de información o servicios al grupo primario.

Dentro del grupo de primarios, se encuentran dos categorías. Por un lado, están los que toman decisiones dentro de la organización industrial. De este grupo debemos analizar la visión y los valores y sus objetivos. Por otro lado, el personal técnico, formado por ingenieros de sistemas, producto managers, etc. De este segundo grupo debemos

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

ver sus objetivos, el escenario en el que trabajan y capacidades fundamentales. Este resultado lo podemos ver en la tabla 12.

Tabla 12. Tipos Stakeholders

| Tipos | Stakeholders |
|--------------------|--------------------------------------|
| Primarios | Responsable de la toma de decisiones |
| | Ingeniero de sistemas |
| | Operario de planta |
| | Supervisor |
| | Operador de mantenimiento |
| | Servicios de mantenimiento |
| | Project Manager |
| | Desarrolladores de algoritmos |
| | Programadores |
| | Técnicos |
| Secundarios | Proveedores de tecnología |
| | Proveedores |
| | Cientes |
| | Desarrolladores de software |

Una vez definidos los *Stakeholder*, se ha desarrollado un modelo de actividades para tener una imagen clara de la distribución de los *Stakeholder* (Figura 30) dentro del entorno de la fabricación en el que introduciremos nuestros algoritmos.

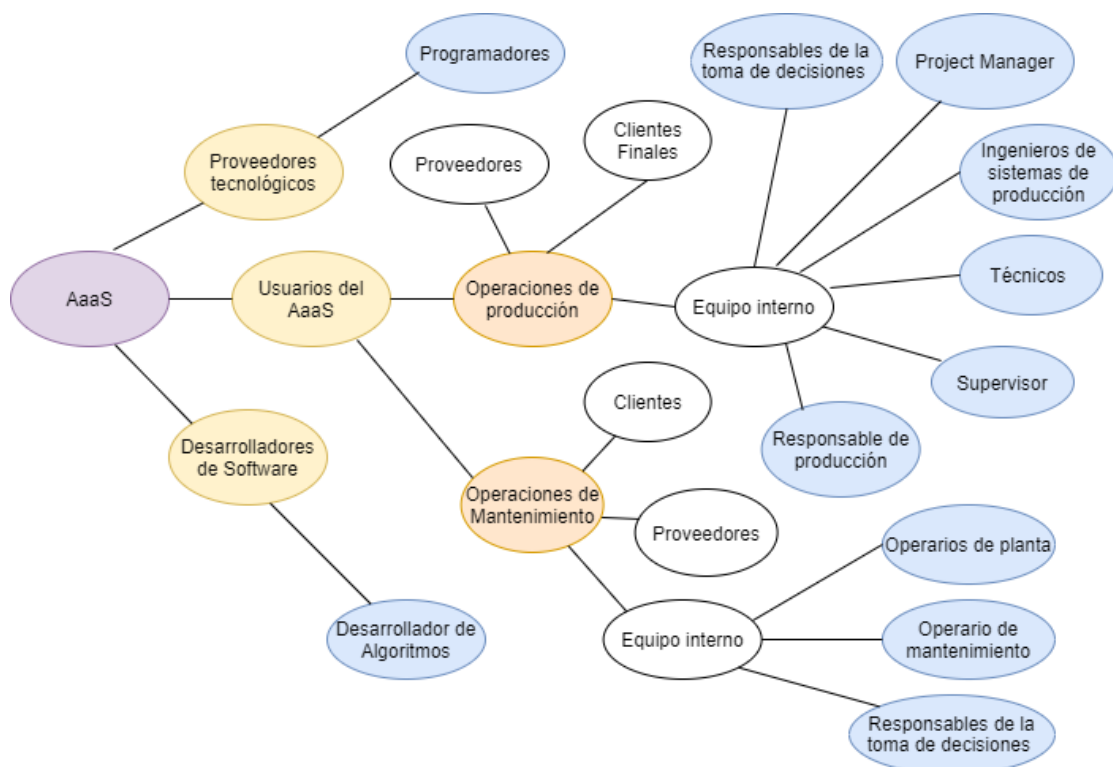


Figura 30. Modelo de distribución de los Stakeholders

Teniendo en cuenta los *stakeholders*, se identificará desde el punto empresarial, la visión, los valores y experiencias, los objetivos clave de negocio y los procesos en los que deben focalizarse. Podemos ver el resultado de las tablas 13 a la 22.

Tabla 13. Stakeholder: Responsable de la toma de decisiones

| Stakeholder | Responsables de la toma de decisiones |
|--|--|
| Visión | Ser reconocido como un fabricante que ofrece una producción de alto nivel sin fallos. |
| Valores y experiencias | La empresa podría beneficiarse de los servicios basados en datos que contribuyen a la identificación temprana de problemas en los productos finales. Para ello, podría ser útil integrar herramientas que permitan mejorar la calidad del proceso de producción. De este modo, la empresa podría reducir los costes relacionados con los desechos y las piezas que hay que volver a realizar. Todo esto podría aumentar la satisfacción del cliente final. |
| Objetivos clave | Desde el punto de vista de negocio: <ul style="list-style-type: none"> • Minimizar los residuos. • Reducir los costes de las actividades de reelaboración. • Mejorar la calidad del producto final. • Aumentar la satisfacción del cliente. |
| Procesos en los que focalizarse | <ul style="list-style-type: none"> • Control de la producción. • Garantía de calidad. |

Tabla 14. Stakeholder: Responsable de producción

| Stakeholder | Responsable de producción |
|----------------------------------|--|
| Escenario | Actividades de control de la producción |
| Objetivos clave | Desde un punto de vista de negocio: <ul style="list-style-type: none"> • Conocer las actividades que realizan. • Minimiza gastos y reducir costes. Desde un punto de vista técnico: <ul style="list-style-type: none"> • Anticiparse posibles problemas en la producción. |
| Capacidades fundamentales | <ul style="list-style-type: none"> • Control de la trazabilidad de los datos. • Conocimiento de las necesidades de la cadena de producción. • Control de tareas y actividades que ocurren dentro de la organización. • Predicción de posibles problemas. |

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

Tabla 15. Stakeholder: Operarios de planta

| Stakeholder | Operarios de Planta |
|---------------------------|---|
| Escenario | Actividades de control de la producción |
| Objetivos clave | <p>Desde un punto de vista de negocio:</p> <ul style="list-style-type: none"> • Conocer las necesidades de la maquinaria • Minimizar gastos y reducir costes. <p>Desde un punto de vista técnico:</p> <ul style="list-style-type: none"> • Anticipación a posibles problemas en la producción. • Conocimientos previos para la reconfiguración correcta de la maquinaria. |
| Capacidades fundamentales | <ul style="list-style-type: none"> • Conocimiento de los parámetros del proceso de producción, verificando el rango de funcionamiento. • Conocimiento de los procesos de reconfiguración de los parámetros del proceso. • Propone autoajustes y sugerencias si se produce un problema. |

Tabla 16. Stakeholder: Operarios de Mantenimiento

| Stakeholder | Operarios de Mantenimiento |
|---------------------------|---|
| Escenario | Actividades de control de la producción |
| Objetivos clave | <p>Desde un punto de vista de negocio:</p> <ul style="list-style-type: none"> • Conocer el estado de la maquinaria. • Conocer los recursos que posee la empresa. <p>Desde un punto de vista técnico:</p> <ul style="list-style-type: none"> • Anticipar posibles problemas en la maquinaria. |
| Capacidades fundamentales | <ul style="list-style-type: none"> • Conocimiento de las necesidades de los equipos de producción. • Predicción de posibles problemas. |

Tabla 17. Stakeholder: Programadores

| Stakeholder | Programadores [Proveedores tecnológicos] |
|---------------------------|--|
| Escenario | Actividades de desarrollo para las funciones de lógica, reglas y los modelos de aplicación a un nivel alto y detallado. |
| Objetivos clave | <p>Desde un punto de vista de negocio:</p> <ul style="list-style-type: none"> • Desarrollar de algoritmos y soluciones para empresas. • Conocer los recursos informáticos necesarios para implementar un algoritmo. <p>Desde un punto de vista técnico:</p> <ul style="list-style-type: none"> • Desarrollar programas y algoritmos de calidad. |
| Capacidades fundamentales | <ul style="list-style-type: none"> • Conocimiento de las necesidades de software de la cadena de producción. • Proponen mejoras y sugerencias a los diferentes problemas. |

Tabla 18. Stakeholder: Supervisor

| Stakeholder | Supervisores |
|----------------------------------|--|
| Escenario | Actividades de operaciones. Gestionar, supervisar y optimizar soluciones del dominio de control de la producción. |
| Objetivos clave | Desde un punto de vista de negocio: <ul style="list-style-type: none"> • Conocer las actividades que se realizan. • Minimizar gastos y reducir costes. Desde un punto de vista técnico: <ul style="list-style-type: none"> • Anticiparse a fallos de calidad. |
| Capacidades fundamentales | <ul style="list-style-type: none"> • Conocimiento de las necesidades de la cadena de producción. • Conocimiento de los posibles fallos en producción. |

Tabla 19. Stakeholder: Ingenieros de Sistemas de producción

| Stakeholder | Ingenieros de Sistemas de producción |
|----------------------------------|--|
| Escenario | Actividades de desarrollo para las funciones de lógica, reglas y los modelos de aplicación a un nivel alto y detallado. |
| Objetivos clave | Desde un punto de vista de negocio: <ul style="list-style-type: none"> • Conocer toda la estructura lógica del sistema. • Conocer todos los servicios de la producción. Desde un punto de vista técnico: <ul style="list-style-type: none"> • Conocer el flujo de necesidades de la cadena de producción. • Solucionar problemas de integración. |
| Capacidades fundamentales | <ul style="list-style-type: none"> • Control de los datos. • Control de las necesidades del sistema. • Control de fallos y necesidades. |

Tabla 20. Stakeholder: Project Manager

| Stakeholder | Project Manager |
|----------------------------------|---|
| Escenario | Actividades de operaciones. Gestionar, supervisar y optimizar soluciones del dominio de control de la producción. |
| Objetivos clave | Desde un punto de vista de negocio: <ul style="list-style-type: none"> • Encargarse de la integración de nuevos servicios • Supervisar el resto del equipo interno Desde un punto de vista técnico: <ul style="list-style-type: none"> • Conocer todas las áreas de la empresa y necesidades hasta los usuarios finales. |
| Capacidades fundamentales | <ul style="list-style-type: none"> • Control de personal. • Control de programas e integraciones. |

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

Tabla 21. Stakeholder: Desarrolladores de algoritmos

| Stakeholder | Desarrolladores de algoritmos [Desarrolladores de Software] |
|----------------------------------|--|
| Escenario | Actividades de desarrollo para las funciones de lógica, reglas y los modelos de aplicación a un nivel alto y detallado. |
| Objetivos clave | <p>Desde un punto de vista de negocio:</p> <ul style="list-style-type: none"> • Conocer las actividades que se realizan. • Minimizar gastos y reducir costes. • Conocer los recursos y necesidades de una máquina. <p>Desde un punto de vista técnico:</p> <ul style="list-style-type: none"> • Solucionar problemas a bajo nivel. • Anticiparse a posibles problemas en la producción. • Obtener calidad. |
| Capacidades fundamentales | <ul style="list-style-type: none"> • Conocimiento de las necesidades de la cadena de producción. • Control de tareas y actividades que ocurren dentro de la organización. |

Tabla 22. Stakeholder: Técnicos

| Stakeholder | Técnico |
|----------------------------------|--|
| Escenario | Actividades relacionadas con el proceso productivo de la línea de producción |
| Objetivos clave | <p>Desde un punto de vista de negocio:</p> <ul style="list-style-type: none"> • Adecua la estructura y la organización de la línea para aumentar la eficiencia. • Realiza tareas de mantenimiento y reparación de la línea. • Cumple con los planes de producción KPIs. <p>Desde un punto de vista técnico:</p> <ul style="list-style-type: none"> • Anticiparse posibles problemas en la producción. • Participa en proyectos de mejora tecnológica de la línea de producción. |
| Capacidades fundamentales | <ul style="list-style-type: none"> • Control de la trazabilidad de los datos. • Conocimiento de las necesidades de la cadena de producción con respecto a equipos y maquinaria. • Colabora con la oficina técnica para diseñar equipos y componentes. • Predicción de posibles problemas. |

4.2.2 Fase 2: Vista de operaciones

En esta fase se definen tanto las actividades (tareas o roles) de cada uno de los *Stakeholder*, como los requisitos de los sistemas involucrados, teniendo en cuenta las referencias que nos proporciona IIRA (Figura 31). Para ello, se recogen los *Stakeholder* definidos en la fase anterior y se definen sus roles y tareas dentro de la industria. Seguidamente, se detallarán las actividades dentro del desarrollo de este servicio, teniendo en cuenta las tareas planteadas en los casos de uso. Finalmente, estos dos análisis proporcionarán un mapa funcional donde se relacionarán los *Stakeholder* con las actividades del sistema que se tendrá en cuenta en la siguiente etapa.

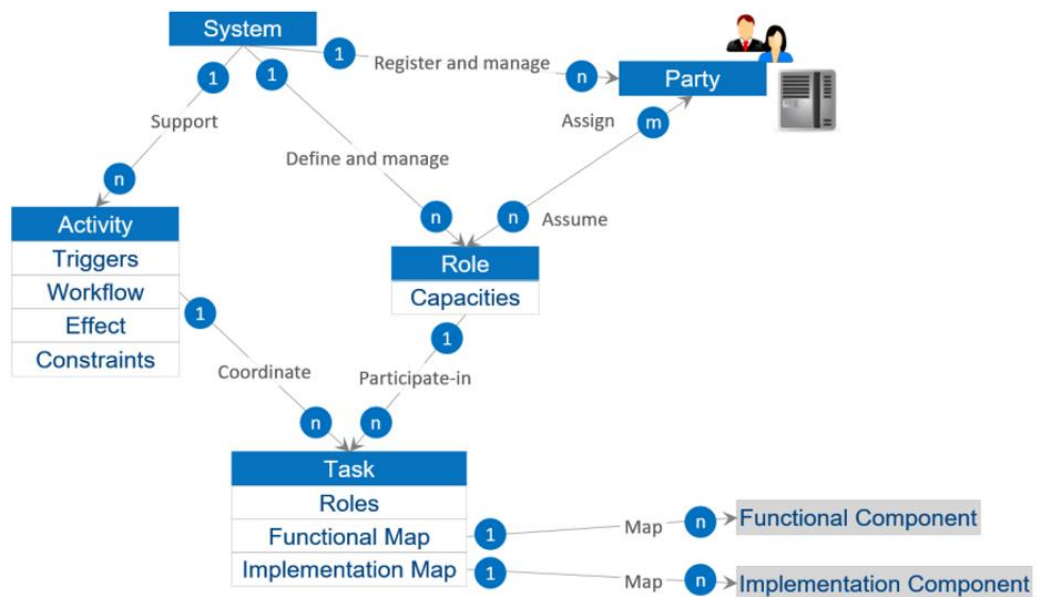


Figura 31. IIRA: Usage Viewpoint

Teniendo en cuenta las tareas a realizar en esta fase, se empezará por definir para cada *Stakeholder* visto anteriormente sus tareas y roles en una organización. Pero antes, hay que determinar los tipos de roles que existen.

Un rol define las tareas específicas que realiza un usuario en un entorno laboral para asegurar su productividad y resultados. Esto permite generar una identificación personal a cada uno de los *stakeholders* potenciando los puntos fuertes y débiles. Conocer los roles de una organización es esencial para lograr un equilibrio que favorezca las dinámicas laborales de cada uno de los *Stakeholder*. Estos roles se dividen en tres grupos:

- **Rol de acción:** Se trata de los usuarios que se encargan de realizar las tareas más técnicas.

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

- **Rol coordinador:** Se trata de los usuarios encargados de coordinar los equipos de trabajo e investigar nuevas necesidades de la organización.
- **Rol responsable:** Se trata de los usuarios que se encargan de realizar tomas de decisiones en base a su experiencia y conocimientos de las operaciones.

Una vez vistos los roles y tareas, se verán las actividades ligadas al sistema. Una actividad es una tarea específica que se realiza dentro de un entorno o sistema. Cada actividad tiene los siguientes elementos:

- **Un desencadenante:** Son las condiciones por las que se inicia la actividad. Puede estar asociada a uno o más roles responsables.
- **Un flujo de trabajo** consiste en una organización secuencial, paralela o condicional de las tareas.
- **Las restricciones** son características del sistema que deben preservarse durante la ejecución y después del nuevo estado. Estas características pueden verse afectadas por la promulgación de las tareas más allá de lo que es exigible por el diseño del sistema o sus componentes funcionales por sí solos.

No obstante, en esta fase del desarrollo de la arquitectura, basta con una descripción básica de las actividades. Ya que, durante el diseño, las actividades sirven como entradas para los requisitos del sistema, guiándonos en el diseño funcional de la siguiente etapa. Para poder ver las actividades, primero tenemos que definir una serie de tareas que se pueden encontrar dentro de la industria y que se utilizan para definir las actividades. Estas tareas las podemos encontrar detalladas en las tablas de la 23 a la 34.

Tabla 23. Tareas: Producir un artículo

| | |
|--------------------|---|
| ID | T01 |
| Nombre | Producir un artículo |
| Descripción | El proceso de producción de un artículo o de cualquiera de las partes que componen un producto. |

Tabla 24. Tareas: Predecir calidad

| | |
|--------------------|---|
| ID | T02 |
| Nombre | Predecir la calidad de un artículo en tiempo de producción. |
| Descripción | Combinando la información recogida por los sensores con modelos de inteligencia artificial o de aprendizaje automático, es posible predecir la calidad de un artículo antes de que se complete el proceso de fabricación. |

Tabla 25. Taras: Comprobar manualmente

| | |
|--------------------|---|
| ID | T03 |
| Nombre | Comprobar manualmente la calidad de un artículo. |
| Descripción | Evaluación manual de la calidad de un artículo para encontrar cualquier posible defecto o imperfección. |

Tabla 26. Taras: Decidir modificación parámetros

| | |
|--------------------|---|
| ID | T04 |
| Nombre | Decidir si hay que modificar los parámetros de la máquina. |
| Descripción | Hay diferentes razones por las que puede ser necesario modificar la configuración de una máquina. Las razones para modificar esta configuración pueden estar relacionadas con los datos recogidos por los sensores que controlan la máquina, la proporción de artículos admitidos y rechazados o el análisis de la calidad de los artículos producidos, por ejemplo. En función de estos u otros criterios, el responsable del funcionamiento de la máquina puede decidir modificar estos parámetros. |

Tabla 27. Taras: Proponer nuevos parámetros

| | |
|--------------------|--|
| ID | T05 |
| Nombre | Proponer nuevos valores para los parámetros de la máquina. |
| Descripción | Si se considera que es necesario modificar los parámetros de una máquina, hay que proponer nuevos valores. Esta propuesta puede basarse en aspectos como los datos recogidos por los sensores que controlan la máquina, la proporción de artículos admitidos y rechazados o el análisis de la calidad de los artículos producidos. |

Tabla 28. Taras: Decidir método para resolver

| | |
|--------------------|---|
| ID | T06 |
| Nombre | Decidir si el problema puede resolverse con cambios automáticos. |
| Descripción | Normalmente, los problemas que surgen durante el proceso de producción son conocidos por los operarios. Basándose en la experiencia de los usuarios y en los conocimientos de los expertos, es posible clasificar estos problemas y decidir cuáles pueden resolverse de forma automática. |

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

Tabla 29. Taras: Modificar automáticamente

| | |
|--------------------|--|
| ID | T07 |
| Nombre | Modificar automáticamente los parámetros de la máquina. |
| Descripción | Se puede suponer que un problema de producción es rectificable de forma automática. En este caso, la máquina asociada o un agente externo automatizado puede medir y evaluar algunas métricas a través de una serie de sensores y, mediante técnicas de control, actuar sobre las entradas de la máquina para corregir el proceso de producción. |

Tabla 30. Taras: Modificar manualmente

| | |
|--------------------|---|
| ID | T08 |
| Nombre | Modificar manualmente los parámetros de la máquina. |
| Descripción | Cuando un proceso no puede ser rectificado de forma automática, tiene que intervenir un actor humano para modificar manualmente los parámetros de la máquina. |

Tabla 31. Taras: Admitir artículo

| | |
|--------------------|---|
| ID | T09 |
| Nombre | Admitir el artículo. |
| Descripción | Cuando se inspecciona un artículo y cumple los requisitos de calidad. |

Tabla 32. Taras: Rechazar artículo

| | |
|--------------------|--|
| ID | T10 |
| Nombre | Rechazar el artículo. |
| Descripción | Cuando se inspecciona un artículo y no se admite por sus defectos, se rechaza. |

Tabla 33. Taras: Corregir artículo

| | |
|--------------------|---|
| ID | T11 |
| Nombre | Corregir un artículo. |
| Descripción | Dependiendo de las razones por las que no se admita un artículo, puede considerarse su corrección en lugar de su rechazo. |

Tabla 34. Detectar datos

| | |
|--------------------|---|
| ID | T12 |
| Nombre | Detectar los datos durante el proceso de producción del artículo. |
| Descripción | El proceso captura los datos de la línea de producción. |

Para tener una mejor visual del flujo de tareas, este mapa muestra el conjunto de tareas para facilitar su interpretación (figura 32):

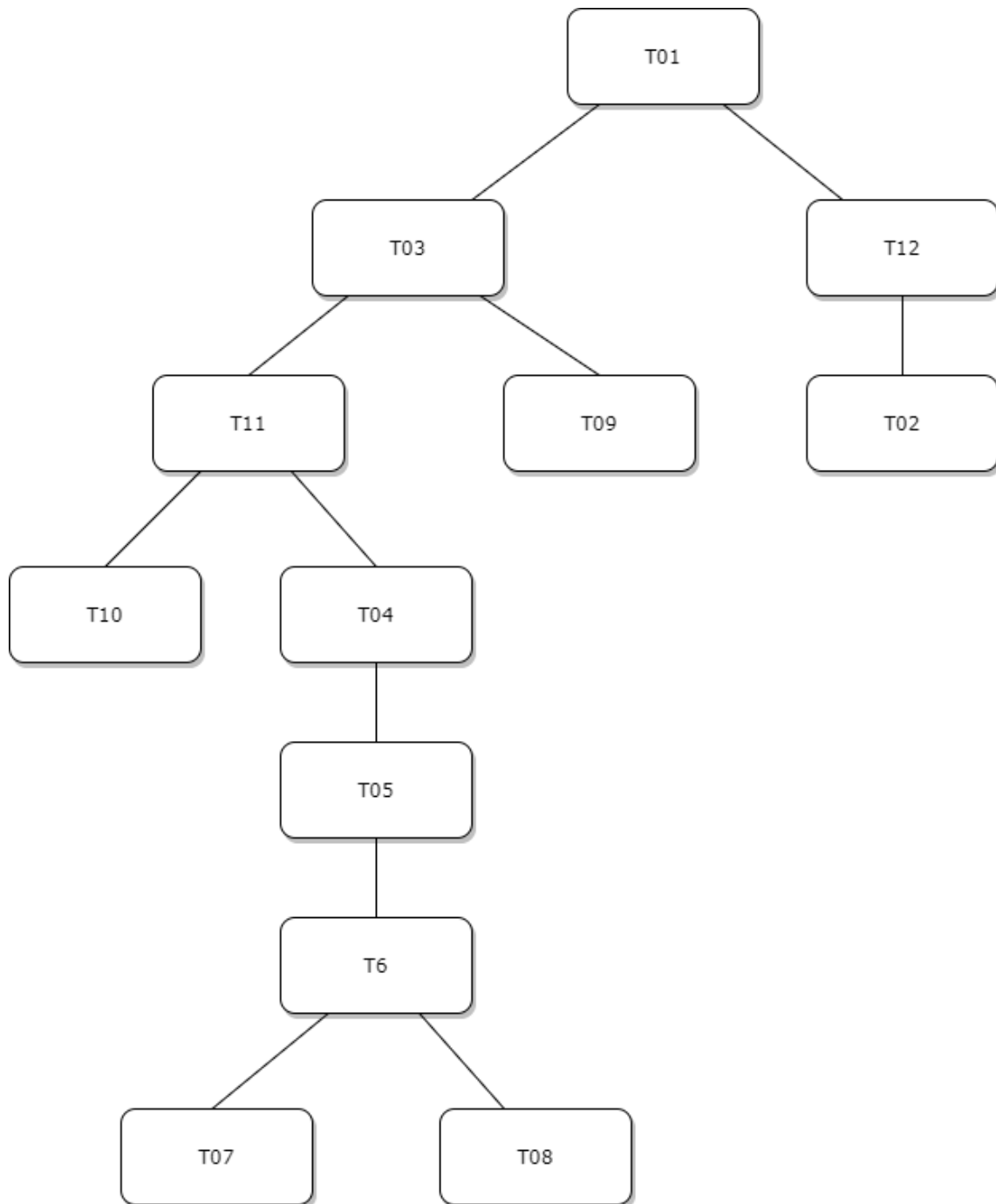


Figura 32. Flujo de tareas

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

Teniendo en cuenta las tareas definidas anteriormente dentro de la industria, las actividades definidas son las siguientes (tablas de la 35 a la 39):

Tabla 35. Actividad: Producir

| | |
|--------------------|--|
| ID | A01 |
| Nombre | Producir un objeto y aceptarlo. |
| Descripción | Esta actividad incluye tareas directamente relacionadas con la producción. |
| Tareas | T01, T03, T09 |

Tabla 36. Actividad: Producir objeto de baja calidad

| | |
|--------------------|--|
| ID | A02 |
| Nombre | Producir un objeto de baja calidad y retirarlo. |
| Descripción | Esta actividad incluye tareas directamente relacionadas con la producción. |
| Tareas | T01, T03, T11, T10 |

Tabla 37. Actividad: Corregir automáticamente

| | |
|--------------------|---|
| ID | A03 |
| Nombre | Detectar un fallo en un objeto de baja calidad y corregir la línea de producción automáticamente. |
| Descripción | Esta actividad incluye tareas directamente relacionadas con la calidad del producto. |
| Tareas | T01, T03, T11, T04, T05, T06, T07 |

Tabla 38. Actividad: Corregir manualmente

| | |
|--------------------|--|
| ID | A04 |
| Nombre | Producir un objeto de baja calidad y corregir la línea de producción manualmente. |
| Descripción | Esta actividad incluye tareas directamente relacionadas con la calidad del producto. |
| Tareas | T01, T03, T11, T04, T05, T06, T08 |

Tabla 39. Actividad: Predecir calidad

| | |
|--------------------|--|
| ID | A05 |
| Nombre | Predecir la calidad de un producto. |
| Descripción | Esta actividad incluye tareas directamente relacionadas con la calidad del producto. |
| Tareas | T01, T12, T02 |

Aunque estas son algunas de las actividades definidas en este caso de uso, que busca la optimización de parámetros de la máquina, también encontramos muchas más actividades dentro de la industria que utilicen algoritmos, como puede ser:

- Planificar y secuenciar las operaciones de producción
- Planificar y secuenciar las operaciones de mantenimiento
- Detectar defectos en los productos
- Detectar defectos o fallos en el proceso productivo

A continuación, se van a ver más en profundidad todas las tareas específicas en base a los objetivos de los diferentes *Stakeholder* y a definir el rol en el que se encuentran (tablas de la 40 a la 48):

Tabla 40. Tareas Stakeholder: Ingeniero de Sistema

| Stakeholder | Responsables de la toma de decisiones |
|--------------------|---|
| Tareas | <ul style="list-style-type: none"> • Ver y conocer las necesidades de los sistemas. • Intentar reducir los costes asociados a fallos o problemas • Realizar tareas para mejorar la calidad final de los productos. |
| Roles | Rol responsable |

Tabla 41. Tareas Stakeholder: Ingeniero de Sistema

| Stakeholder | Supervisores |
|--------------------|---|
| Tareas | <ul style="list-style-type: none"> • Comprobar todas las actividades que ocurren en la planta de producción. • Comprobar los equipos de desarrollo. • Anticiparse a problemas que puedan surgir. |
| Roles | Rol coordinador |

Tabla 42. Tareas Stakeholder: Ingeniero de Sistema

| Stakeholder | Técnico |
|--------------------|---|
| Tareas | <ul style="list-style-type: none"> • Comprobar el funcionamiento de los sistemas • Arreglar los problemas que aparezcan |
| Roles | Rol de acción |

Tabla 43. Tareas Stakeholder: Ingeniero de Sistema

| Stakeholder | Operarios de Planta |
|--------------------|--|
| Tareas | <ul style="list-style-type: none"> • Utilizar las máquinas en la línea de producción. • Comprobar si hay fallos en el producto que se está elaborando. |
| Roles | Rol de acción |

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

Tabla 44. Tareas Stakeholder: Ingeniero de Sistema

| Stakeholder | Operarios de Mantenimiento |
|-------------|--|
| Tareas | <ul style="list-style-type: none"> Ver el estado de las máquinas y repararlas cuando sea necesario. |
| Roles | Rol de acción |

Tabla 45. Tareas Stakeholder: Ingeniero de Sistema

| Stakeholder | Programadores |
|-------------|--|
| Tareas | <ul style="list-style-type: none"> Desarrollar programas internos y mejoras para los sistemas implementados. Comprobar la calidad de los programas introducidos. |
| Roles | Rol de acción |

Tabla 46. Tareas Stakeholder: Ingeniero de Sistema

| Stakeholder | Ingenieros de Sistemas |
|-------------|--|
| Tareas | <ul style="list-style-type: none"> Crear, diseñar y mantener los sistemas industriales. Administrar redes y sistemas de información. Optimizar los datos que utilice la empresa. Investigar para mejorar el software y hardware de la empresa. Mantener servicios de la red que se usen dentro de la industria. |
| Roles | Rol coordinador |

Tabla 47. Tareas Stakeholder: Project Manager

| Stakeholder | Project Manager |
|-------------|---|
| Tareas | <ul style="list-style-type: none"> Planificar de proyectos. Supervisar tareas. Implementar cambios y soluciones. |
| Roles | Rol responsable. |

Tabla 48. Tareas Stakeholder: Desarrollador de algoritmos

| Stakeholder | Desarrolladores de Algoritmos |
|-------------|---|
| Tareas | <ul style="list-style-type: none"> Recopilar y almacenar de datos. Transformar datos recopilados para un mejor análisis. Resolver problemas técnicos. Plantear y proponer soluciones. |
| Roles | Rol de acción. |

Para el conjunto de tareas definidas podemos relacionar los siguientes *stakeholders* con las actividades vistas (tabla 49):

Tabla 49. Tabla de Actividades

| | |
|------------|---|
| A01 | Operarios de Planta, Operarios de Mantenimiento. |
| A02 | Operarios de Planta, Operarios de Mantenimiento. |
| A03 | Responsables de la toma de decisiones, Supervisores, Ingenieros del Sistema, Project Manager. |
| A04 | Responsables de la toma de decisiones, Supervisores, Técnicos. |
| A05 | Ingenieros del Sistema, Project Manager, Desarrolladores de algoritmos, Programadores. |

Después de este proceso de identificación de los diferentes *stakeholders* que se pueden encontrar dentro del entorno industrial donde se integre el sistema de Algoritmos como Servicio, centraremos la atención en los *stakeholders* que afectan a las actividades de “Predecir la calidad de un producto” y “Producir un objeto de baja calidad y corregir la línea de producción automáticamente”. Es precisamente en estas actividades en las que se puede utilizar nuestro servicio porque, por medio de algoritmos, se puede predecir la calidad del producto o automatizar la corrección de la línea de producción de forma inmediata. Los *stakeholders* afectados son:

- Los programadores y desarrolladores de algoritmos, que crean los algoritmos necesarios y que trabajan tanto dentro de la empresa, como desde fuera siendo proveedor o desarrollador de software.
- Los ingenieros de sistemas, que son los encargados de implementar el sistema dentro de la organización.
- El Project Manager y responsables de la toma de decisiones que se encargará de indicar qué algoritmos debe usar el usuario final.

4.3 Etapa 3: Vista funcional y de implementación

En esta fase se desarrollan los dominios funcionales, se identifican los flujos de datos, decisiones y solicitudes que circulen entre ellos. Una vez realizada esta descomposición del funcionamiento del sistema de Algoritmos como Servicio, se profundizará en el flujo de operaciones que realiza cada uno de los *stakeholders* identificados anteriormente. Finalmente, se diseñará un proceso de implementación y se analizarán las tecnologías necesarias para desplegarlo.



4.3.1 Fase 1: Vista funcional

Uno de los puntos más importantes en el desarrollo de software es la vista funcional, ya que es necesaria para desarrollar e implementar cualquier software. Gracias a este punto, se consigue una imagen general del proyecto para su correcta implementación. En ella se tiene en cuenta tanto la arquitectura y la organización de los sistemas, como sus componentes y la relación entre ellos. Para encaminar bien esta fase, primero hay que modelar, mediante diagramas funcionales, después definir los componentes y, por último, desarrollar un diagrama de secuencia, para facilitar la comprensión del funcionamiento del servicio.

Antes de desarrollar el diagrama funcional, se definirá el funcionamiento del sistema de Algoritmos como servicio:

1. Un usuario podrá acceder a una API del servicio.
2. En la Api tendrá que ir definido si desea realizar una consulta sobre el algoritmo implementado, el cual responderá con toda la información sobre el algoritmo, incluyendo inputs y outputs; o lanzar el algoritmo.
3. Cuando se lanza el algoritmo, este deberá contener una serie de datos para su correcto uso. Por otro lado, también podrá subscribirse a un algoritmo, para que esté ejecutándose de forma continua y pueda obtener los datos analizados por el algoritmo de forma constante.
4. Una vez el servicio reconoce los datos de entrada, los ejecuta según el algoritmo y enviará el resultado a un cliente.

Como se puede ver en la imagen del diagrama funcional (figura 33), se ha desarrollado una primera versión del funcionamiento de los algoritmos como Servicio.

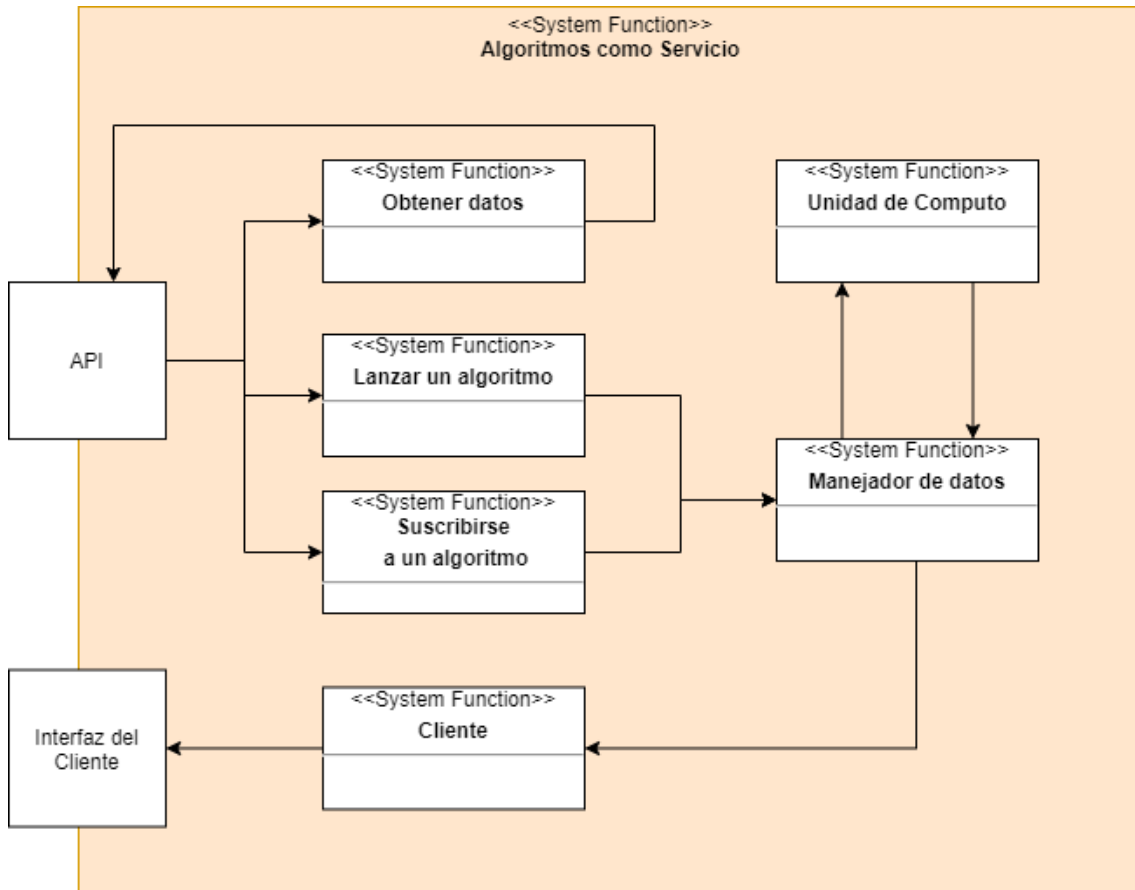


Figura 33. Diagrama funcional

Para entender cada uno de los componentes, a continuación, se explicarán con mayor profundidad (tablas de la 50 a la 55).

Tabla 50. Componente: Obtener datos

| Nombre | Obtener datos |
|-------------|---|
| Descripción | Este componente procesa la información y devuelve todos los datos del algoritmo. En estos datos incluye desde una descripción en profundidad del algoritmo, hasta los inputs y outputs de este. |
| Input | Requiere de una llamada a la API sin pasarle ninguna información específica. |
| Output | Devuelve un objeto con el nombre del algoritmo, información detallada de su funcionamiento, información de los inputs que requiere para su funcionamiento y la salida que devuelve. |

Tabla 51. Componente: Lanzar un algoritmo

| Nombre | Lanzar un algoritmo |
|-------------|--|
| Descripción | Este componente recoge la configuración que le envía la API para lanzar un dato. |
| Input | Requiere de una llamada a la API que le envía la información necesaria. |
| Output | Le envía los datos configurados al manejador de datos. |

Tabla 52. Componente: Suscribirse a un algoritmo

| Nombre | Suscribirse a un algoritmo |
|-------------|---|
| Descripción | Este componente recoge la configuración que le envía la API para que un usuario o máquina se suscriba a un algoritmo. |
| Input | Requiere de una llamada a la API que le envía la información. Esta información consta de una identificación, datos de configuración del cliente y la información necesaria. |
| Output | Le envía los datos configurados al manejador de datos. |

Tabla 53. Componente: Manejador de datos

| Nombre | Manejador de datos |
|-------------|--|
| Descripción | Identifica los datos de configuración que recibe, ya sea de las suscripciones o de lanzamiento de un algoritmo. Conecta con la unidad de cómputo que se encarga de lanzar los algoritmos y con los clientes donde se vuelcan los resultados finales. |
| Input | Recibe la configuración del cliente y los datos necesarios para ejecutar el algoritmo. |
| Output | Envía los datos analizados al cliente, con la configuración pertinente. |

Tabla 54. Componente: Unidad de computo

| Nombre | Unidad de Computo |
|-------------|---|
| Descripción | Ejecuta el algoritmo según los datos recibidos. |
| Input | Recibe los datos necesarios para lanzar el algoritmo. |
| Output | Devuelve el resultado del algoritmo lanzado. |

Tabla 55. Componente: Cliente

| Nombre | Cliente |
|-------------|--|
| Descripción | El cliente es el punto final donde se almacenan los datos ejecutados. Esto puede ser un fichero, una base de datos, una base de datos no relacional o sistema de comunicación por mensajes constantes. Este es el punto al que accede el usuario final |
| Input | Los datos. |
| Output | El resultado final. |

Teniendo en cuenta esta definición de cada uno de los componentes, se desarrolla un diagrama de secuencia (figura 34) donde se define como un usuario final utiliza los componentes del servicio.

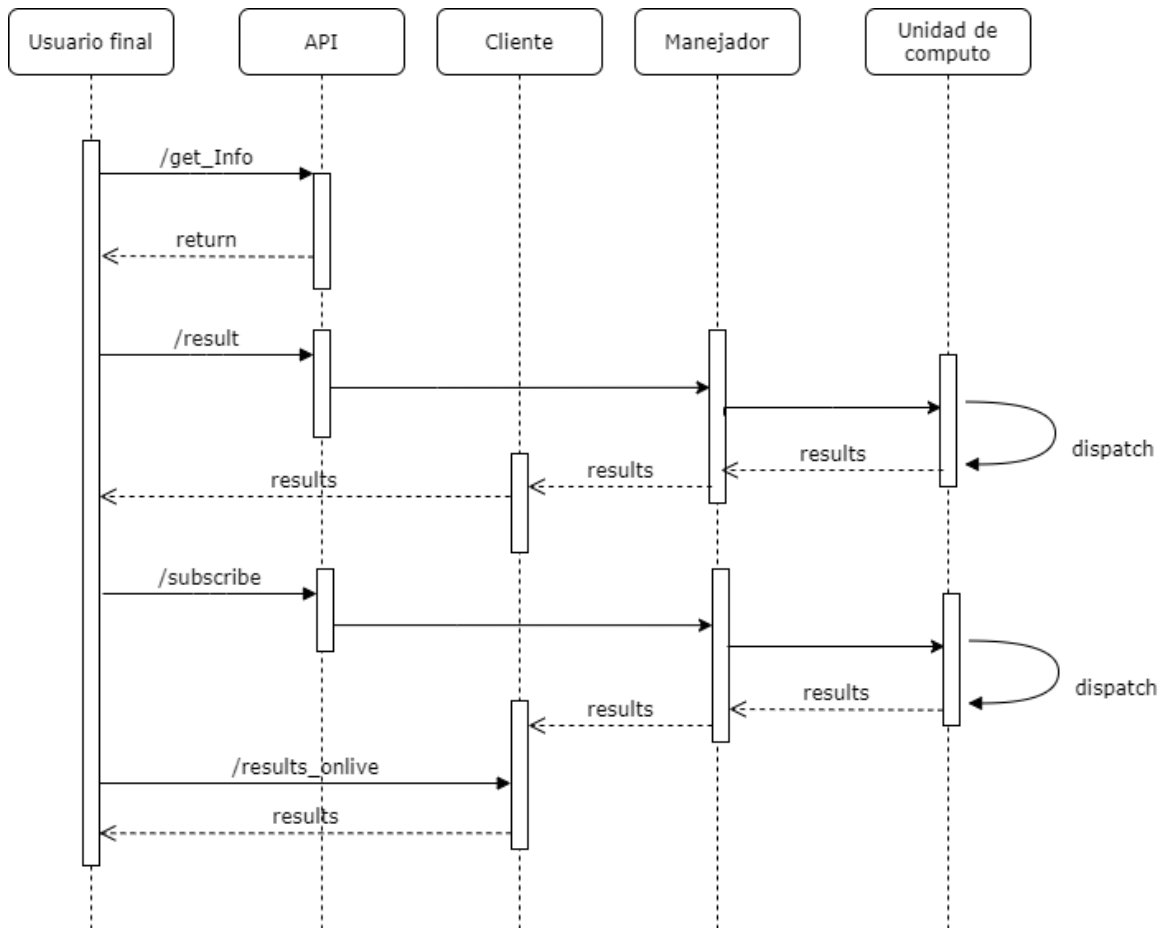


Figura 34. Diagrama de secuencia

4.3.2 Fase 2: Vista de implementación

La vista de implementación expone todos los elementos físicos que podemos encontrar en el sistema, ya sean componentes, interfaces o dependencias entre los componentes. Para comprender la solución propuesta, primero se detallarán las diferentes tecnologías que se utilizarán para integrar el servicio y el motivo de su elección.

La tecnología principal es Docker, ya que permite ofrecer un servicio contenerizado adaptado a las diferentes necesidades que pudieran surgir. Para el desarrollo de la imagen Docker es necesario utilizar un repositorio donde guardar el código y lanzar el fichero de integración continua. Este fichero utilizará Kaniko para generar la imagen con el servicio y un *package* donde se guarde la solución del algoritmo. Además, se integrará la posibilidad de desplegar un SonarQube propio para analizar la calidad del código.

Para desplegar el sistema de Algoritmos como servicio será necesario tener un entorno contenerizado y utilizar alguna herramienta *user-friendly* como Portainer. Para mejorar los requisitos no funcionales, se transformará este contenedor Docker en Kubernetes mejorando así su escalabilidad, y proporcionando las instrucciones para generar los ficheros para este orquestador. También se incluye una herramienta *user-friendly* para gestionar Kubernetes llamada Rancher.

A continuación, se tratarán de definir y resumir las herramientas y tecnologías nombradas anteriormente.

Docker

Docker (figura 35) nació en 2013 y es un proyecto de código abierto que permite encapsular aplicaciones software con sus propias dependencias sin que afecten al sistema operativo que lo aloja. Utilizar esta tecnología proporciona muchas ventajas a la hora de ofrecer un servicio ya que se pueden encapsular en un mismo contenedor todas las librerías y dependencias, por lo que proporciona una gran capacidad para migrar los servicios. Funciona de un modo parecido a las máquinas virtuales, aunque Docker separa el software en diferentes contenedores sobre una única máquina, dando la posibilidad de poder gestionar los recursos de esta (figura 36).



Figura 35. Logo Docker

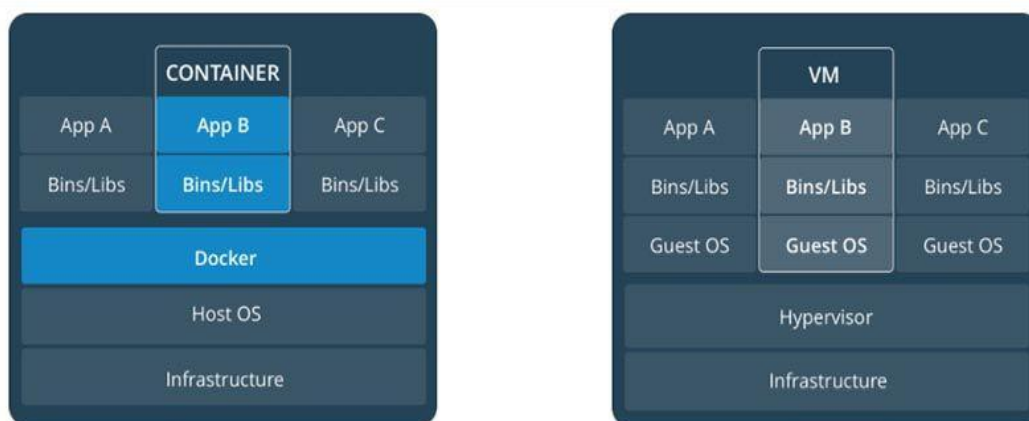


Figura 36. Docker y Máquina Virtual

Para preparar y ofrecer un contenedor Docker existen dos opciones:

- **Repositorio de imágenes:** Un repositorio es un punto en el que se pueden almacenar las imágenes de Docker preparadas para poder acceder a ellas desde cualquier sitio. El repositorio más conocido es Docker Hub.
- **DockerFile:** Es un documento que define la estructura y configuración del contenedor. Mediante este archivo se puede replicar el contenedor en diferentes equipos.

Para este proyecto se utilizará Docker, ya que es el estándar de la industria, como medio para encapsular los algoritmos y así poder instanciarlos. Además, es la tecnología que mejor funciona con Kubernetes.

Kubernetes

De la ventaja de utilizar Docker en un entorno de desarrollo para ofrecer servicios, nace Kubernetes (figura 37). Gracias a esta tecnología se pueden controlar todos contenedores ofreciendo abstracción para su manejo. Aunque en el estado del arte se ha hablado de esta tecnología, a continuación, se explicará de forma básica como trabajan entre sí los diferentes componentes de Kubernetes.



Figura 37. Logo Kubernetes

En primer lugar, los objetos (figura 38):

- **Pods:** Un pod es un elemento en Kubernetes que hace referencia a uno o varios contenedores, que comparten una red y una máquina local.



Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

- **Replica set.** Es el componente que se encarga de gestionar los pods y mantenerlos activos. Garantiza que exista siempre un número de pods, siempre que sea posible. Además, autogestiona los pods caídos por fallos y aumenta el número si es necesario.
- **Deployment:** Trabaja igual que Replica Set y maneja de forma declarativa los pods permitiendo al usuario organizarlos de modo sencillo.
- **Node:** Es la máquina que ejecuta una aplicación.

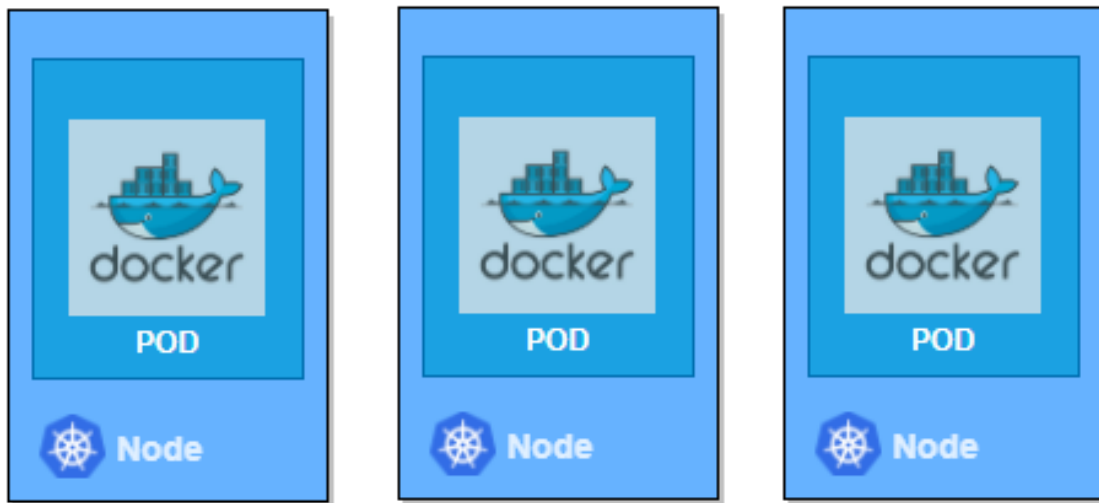


Figura 38. Nodos Kubernetes

A continuación, se verán los servicios que proporciona Kubernetes:

- **ClusterIP:** Proporciona una IP interna que da acceso a los procesos que quieran conectarse al clúster.
- **NodePort:** Este servicio proporciona un puerto estático al nodo que se encuentra ejecutando un servicio dentro del clúster. Para que podamos acceder al servicio, únicamente necesitaremos la IP del nodo y el puerto asignado.
- **LoadBalancer:** Este servicio utiliza tanto el NodePort como de ClusterIP para poder exponer a la red una aplicación accesible desde fuera del clúster.
- **Namespace:** Es la forma de dividir los recursos del clúster entre varios usuarios. Todos ellos están lógicamente aislados unos de otros.

Para que Kubernetes pueda usar todos elementos del clúster y manejar los diferentes pods, despliega dentro de cada nodo unos componentes específicos que son:

- **Kubelet:** Es el encargado de controlar, parar e iniciar un nodo.

- **Kube-proxy:** Es el que se encarga de configurar el servicio de red del nodo dentro del clúster.

Para finalizar, es necesario utilizar **Kubectl**, una herramienta para poder trabajar en un clúster de Kubernetes. Se utiliza para controlar por la línea de comando todo el clúster de Kubernetes, proporcionando la posibilidad de observar, definir y desplegar tanto los nodos, como los pods y/o los servicios del clúster.

Para este proyecto se utilizará Kubernetes para montar nuestro clúster y ofrecer los Algoritmos como Servicios (AaaS).

Gitlab y CI/CD

Gitlab (figura 39) es un sistema de control de versiones de código abierto basado en Git, que proporciona más servicios, desde el alojamiento de contenedores pasando por la integración continua, analítica de uso, seguridad, etc. Uno de los puntos de mayor interés para este proyecto es la facilidad para aplicar integración continua que se ejecuta mediante ficheros *gitlab-ci.yml* en el repositorio. Gracias a esta herramienta se pueden detectar y solucionar problemas por medio de automatización de pruebas a la hora de subir al repositorio el algoritmo. Por otro lado, es posible ejecutar órdenes para automatizar la construcción de imágenes Docker, preparar subidas a repositorios de estas imágenes y cualquier operación que queramos lanzar mediante un *script*.



Figura 39. Logo Gitlab

SonarQube

SonarQube (figura 40) es un software gratuito que permite el análisis de código, informe de errores y malas prácticas dentro del código, como la duplicidad, *Hotspots*, etc. Los resultados se agrupan en 8 categorías:



Figura 40. Logo SonarQube

- **Complejidad:** Hace referencia a las métricas cognitivas y ciclomáticas. Estas referencias definen la complejidad de entender el código fuente para un programador ajeno. Para ello, la herramienta traduce el código en frases y define la dificultad que tiene entender dichas frases debido a las acciones o condiciones.
- **Duplicidad:** Mide la cantidad de código repetido que aparece en el proyecto. Tener duplicidad implica más tiempo para compilar o aumentar el tamaño de este.



Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

- **Fiabilidad:** Hace referencia a bugs encontrados, es decir, fallos dentro de la aplicación que pueden provocar interrupciones.
- **Mantenibilidad:** Hace referencia a la métrica de *code smells*. Estas referencias definen trozos de código fuente que no están bien formulados con respecto al framework o código utilizado y, por tanto, no se ajustan al estándar.
- **Seguridad:** Hace referencia a las vulneraciones que tiene nuestro programa y que pueden provocar que falle.
- **Tamaño:** Nos muestra el número de líneas, comentarios o clases utilizadas por el programa.
- **Test:** SonarQube no ejecuta test, pero sí que recoge los informes y los muestra. Además, te informa del porcentaje de código que está cubierto por estas pruebas.

Para mejorar la calidad del código se usará la herramienta sonarLint implementada en el IDE. Se trata de una extensión gratuita para cualquier IDE que permite corregir los problemas de codificación antes de que existan. Para ello, alerta de los errores igual que los correctores ortográficos, resaltándolos y marcando las vulnerabilidades de seguridad a medida que se escribe el código.

Kaniko

Kaniko (figura 41) es una herramienta que se puede insertar en el CI/CD para crear de forma automática un contenedor con el algoritmo a partir de un fichero base de DockerFile (figura 42). Esta herramienta no necesita permisos del administrador, como otras herramientas, para crear una imagen y permite subirlo automáticamente al registro de contenedores. Para este trabajo, se utilizará Kaniko para crear las imágenes del servicio y así poder ser utilizado por una plataforma Docker o Kubernetes (Priya Wadhwa, 2018).



Figura 41. Logo Kaniko

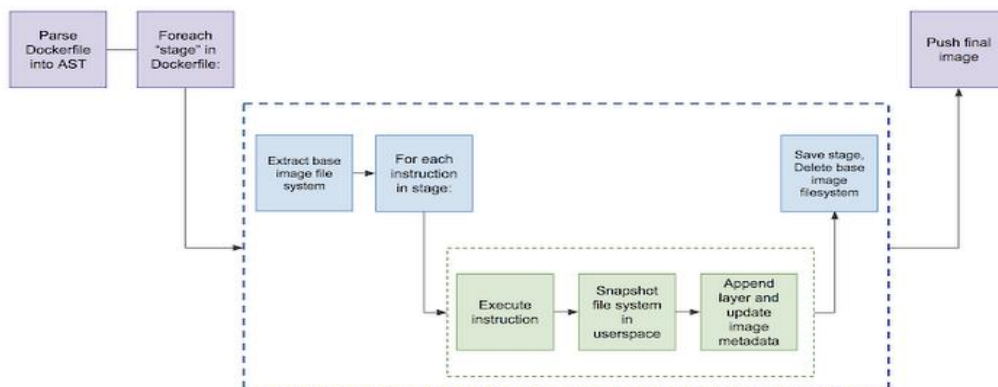


Figura 42. Arquitectura Kaniko

Docker-compose

Docker-compose (figura 43) permite definir diferentes contenedores Docker en uno solo, conectándolos entre sí a partir de un archivo YAML. De esta forma se puede tener centralizado en un solo contenedor todos los contenedores con servicios que requiere una aplicación o microservicio (por ejemplo, un contenedor de BD, uno de front y otro de back).



Figura 43. Logo docker-compose

Kompose

Kompose es una herramienta para ayudar a los usuarios familiarizados con Docker-Compose a migrar a Kubernetes. Toma un archivo de Docker Compose y lo traduce a recursos de Kubernetes (Kompose, 2020). Su uso es muy sencillo, únicamente es necesario introducir el comando “kompose convert -f docker-compose.yaml” y generará los archivos de Kubernetes preparados para su uso.

Rancher

Rancher (figura 44) es un software que permite, por medio de una interfaz gráfica, manejar un servidor o cluster de Kubernetes. Facilita la gestión y control de todos los clusters de una organización, incluso si están alojados en un servidor privado. Por tanto, se trata de una herramienta muy potente para la gestión de servicios para la empresa.



Figura 44. Logo Rancher

Estructura implementación

En las figuras mostradas a continuación (figuras 45, 46 y 47) se puede ver un mapa tecnológico con los diferentes niveles de integración del servicio y las tecnologías a utilizar.

1. El primer nivel afecta a cómo se prepara el algoritmo para el servicio. Como se ha comentado anteriormente, se utilizará un archivo de gitlab-ci para generar automáticamente la imagen de Docker. Este proceso de automatización también pasará una prueba de calidad de SonarQube.

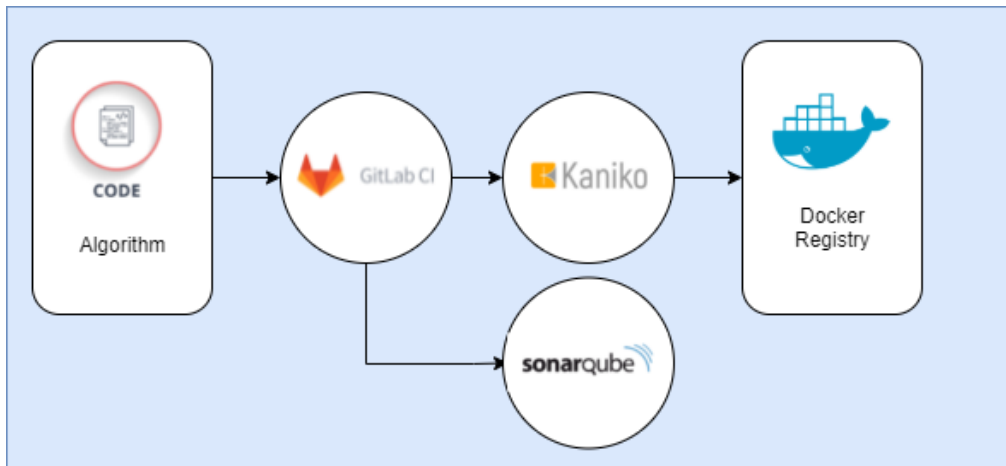


Figura 45. Implementación de gitlab-ci

2. El segundo nivel, es la preparación del algoritmo para ofrecerlo como servicio. Para ello, con la imagen generada en el paso anterior, preparamos un archivo Docker-compose, lanzamos el Kompose y generamos los archivos necesarios para poder usar el servicio en Kubernetes.

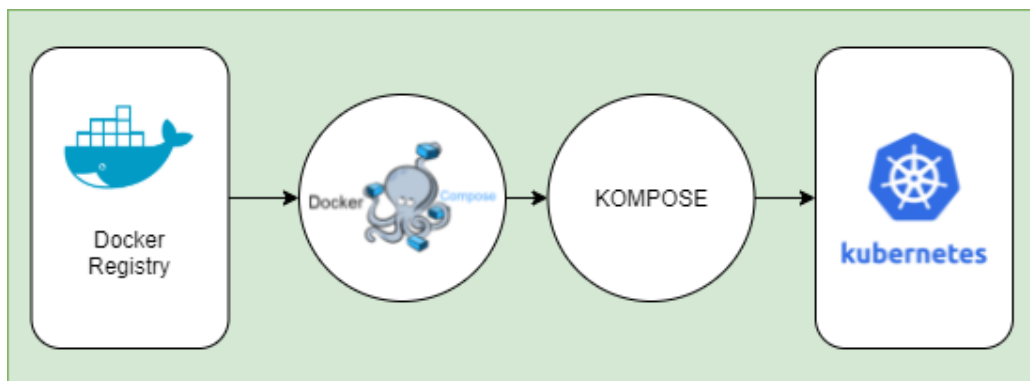


Figura 46. Implementación de docker-compose

3. Por último, el tercer nivel es la implementación del servicio dentro de una organización o empresa. Para ello utilizamos el gestor de servicios de Kubernetes llamado rancher. Cargamos nuestros ficheros Kubernetes y proporcionamos el algoritmo como servicio HTTP.

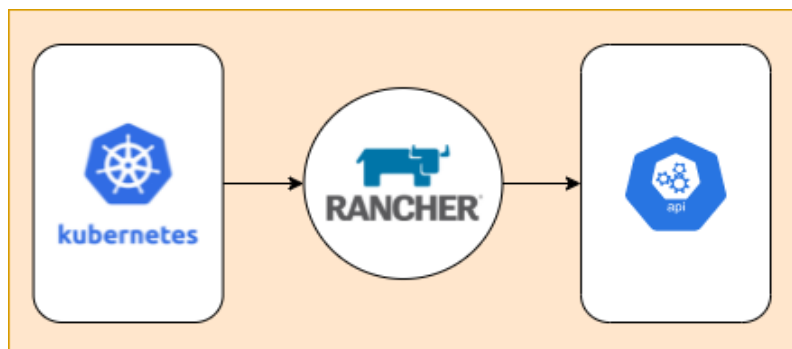


Figura 47. Implementación de Rancher

5. Implementación

Partiendo de la arquitectura definida en el apartado anterior, en este punto se va a desarrollar el fichero de integración continua para la implementación de un AaaS, y se explicará el funcionamiento de la API. A partir de este punto, se irá mostrando su funcionalidad desde el punto de vista de cada uno de los actores implicados.

Para la implantación se definen los siguientes pasos:

1. “Desarrollo de un fichero CI/CD”. En esta sección se partirá del *template* para los algoritmos del PO_runtime de ZDMP, y se desarrollará un fichero de integración continua. Con él se creará la imagen preparada para ofrecer el algoritmo como servicio.
2. “Desarrollo de un algoritmo”. Se define con el rol de un desarrollador que desea preparar un algoritmo para el sistema de Algoritmos como Servicio en un entorno industrial. Tras generarlo, lanza el fichero de integración continua y se generará la imagen con el algoritmo desarrollado y preparado para el servicio.
3. “Preparación del entorno”. Se define con el rol de un proveedor de solución o ingeniero de sistema, que preparará un entorno de Kubernetes gestionado con Rancher donde se desplegará el algoritmo como servicio generado en la fase anterior.
4. “Implementación del algoritmo como servicio”. En esta sección veremos como insertar el algoritmo como servicio dentro de nuestro entorno y conocer el punto de acceso que se generará para usar el AaaS.

5.1 Desarrollo de fichero de integración continua

El primer paso de esta fase es el estudio de la estructura del template que deben utilizar los desarrolladores; seguidamente, se explicará el funcionamiento de las API y el desarrollo del fichero gitlab-ci para la generación automática de la imagen y un test de calidad del programa.

La estructura de directorios del template es el siguiente (figura 48):

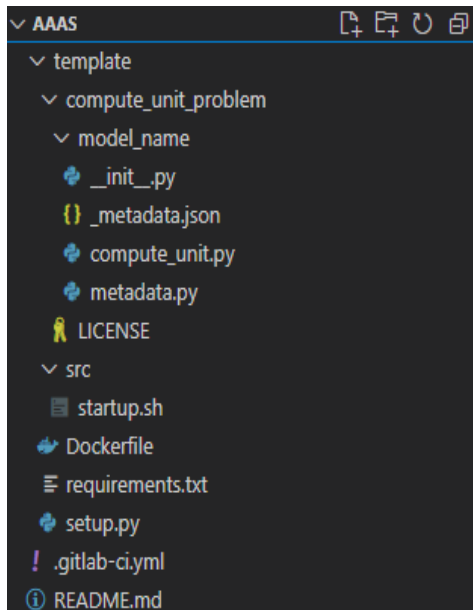


Figura 48. Estructura Template AaaS

Fichero gitlab-ci con las instrucciones para generar una imagen Docker: un directorio template con un DockerFile con las dependencias necesarias, un directorio src con un script que lanzará los clientes necesarios; y un directorio llamado compute_unit_problem que contiene la licencia de uso gratuito y un directorio con el algoritmo. Este directorio deberá contener un fichero inicializador de Python que lanza el metadata, el cual devuelve la información del algoritmo; y el compute_unit que ejecuta el algoritmo.

Para generar la imagen de Docker y pasar por el sonarqube, debemos configurar el archivo gitlab-ci.yml.

```
stage: build
image:
  name: gcr.io/kaniko-project/executor:debug
script:
  - mkdir -p /kaniko/.docker
  - echo
  - '{"auths":{"$CI_REGISTRY":{"username":"$CI_REGISTRY_USER","password":"$CI_REGISTRY_PASSWORD"}}}' > /kaniko/.docker/config.json
  - echo "Building and shipping image to $CI_REGISTRY_IMAGE"
  - echo $CI_PROJECT_DIR
  - echo $CI_COMMIT_TAG
  - /kaniko/executor --context $CI_PROJECT_DIR --dockerfile
  $CI_PROJECT_DIR/template/Dockerfile --destination
  $CI_REGISTRY_IMAGE:$CI_COMMIT_TAG
```

Como se puede ver en el código, el fichero de integración continua a partir de una imagen de Kaniko y la configuración descrita en el DockerFile dentro del template, generará una imagen que guardará en el registro de contenedores del proyecto de gitlab y proporcionará una url de acceso a él. Este Dockerfile de configuración del template contendrá los requisitos mínimos para el correcto funcionamiento del algoritmo y tendrá de base la imagen desarrollada previamente con el backend del AaaS.

```
stage: sonar
image:
  name: sonarsource/sonar-scanner-cli:latest
```

```

variables:
  TOKEN_SONAR: "${TOKEN_SONAR}"
  SONAR_URL: "${SONAR_URL}"
script:
  - sonar-scanner -Dsonar.qualitygate.wait=true -Dsonar.projectKey=Mes -Dsonar.sources=. -
Dsonar.host.url=${SONAR_URL} -Dsonar.login=${TOKEN_SONAR}

```

El segundo proceso es un test de SonarQube que enviará los resultados al servidor donde esté instalado. El desarrollador deberá configurar las variables de entorno para poder usar correctamente el servicio.

Gracias a la API integrada (tabla 56) en el desarrollo del *backend*, se podrá acceder al servicio. Las llamadas serían las siguientes.

Tabla 56. Estructura API

| API | POST | GET |
|----------------------------------|---|--|
| /metadata | -- | Proporciona una descripción del componente de tiempo de ejecución, incluida una descripción de las entradas y salidas. |
| /result | Ejecuta el modelo con la configuración de entrada y salida definida en el cuerpo de la solicitud. | -- |
| /getSubscriptions | -- | Devuelve la lista de suscripciones. |
| /addSubscription | Agrega una "suscripción" a la lista de suscripciones. | -- |
| /deleteSubscription /{id} | -- | Elimina la suscripción con ID de la lista de suscripciones. |

Los parámetros de configuración de entrada se proporcionan como el cuerpo (JSON) de la solicitud POST a / result con las siguientes propiedades:

- **id:** Identificador del parámetro de entrada o salida. Especificado en los metadatos del componente.
- **format:** Especifica el formato del parámetro de datos de entrada o salida. Los valores de esta clave pueden ser "nombre de archivo", "colección-mongo", "colección", "mqtt".
- **<format>:** Esta clave dependerá del tipo de dato que enviemos ya sea un archivo, colección de mongo, base de datos, mensajes, etc. Dependiendo del formato, se indicará con una clave asociada a este cliente.



5.2 Desarrollo de un algoritmo

En esta fase se probará el servicio desde la perspectiva de un desarrollador de algoritmos que va a crear un pequeño algoritmo de prueba para ver su funcionamiento. Este algoritmo es un pequeño reflejo de la ISO 22400. Esta ISO define un conjunto de indicadores clave de rendimiento (KPI) para evaluar el rendimiento de la operación de fabricación (Zhu et al., 2018). Se partirá del supuesto de que un valor de temperatura de un sensor debe estar por debajo de los 50º y que en caso de que no se cumpla, enviará un valor 0 indicando el peligro. El *script* en Python quedaría así:

```
def comprobar_temperatura(i):
    temperatura_maxima = 50
    if i >= temperatura_maxima:
        return 0
    else:
        return 1
```

Una vez generado, será añadido sobre el *template* del AaaS. Para ello, en el fichero metadata.json se añadirá la información correspondiente al algoritmo:

```
{ "name": "Comprobar temperatura",
  "resource_id": "http...",
  "inputs": [
    { "id": "Identificador",
      "description": "Este algoritmo avisa si se supera la temperatura de 50º",
      "attributes": [
        { "id": "1",
          "name": "i",
          "description": "temperatura actual",
          "type": "integer"
        }
      ]
    }
  ],
  "outputs": [
    { "id": "1",
      "description": "Resultado de comprobar el valor",
      "attributes": [
        { "id": "1",
          "description": "0 si supera los 50º, 1 si está por debajo",
          "type": "integer"
        }
      ]
    }
  ]
}
```

En el fichero compute_unit se añadirá el algoritmo:

```
class ComputeUnit:
    def compute(self, temperature):
        max_temperature = 50
        if temperature >= max_temperature:
```

```
return 0
else:
return 1
```

Para poder utilizar el servicio, se ha preparado la aplicación, previamente, en una imagen con la estructura y dependencia necesaria y así se define en el DockerFile que creará el servicio. Esta imagen ya contendrá los ficheros de configuración para que funcione la API la cual apuntará a los ficheros que añadimos con el *template*. El fichero quedará de la siguiente forma:

```
FROM registry.gitlab.com/mimaca1/aaas:service

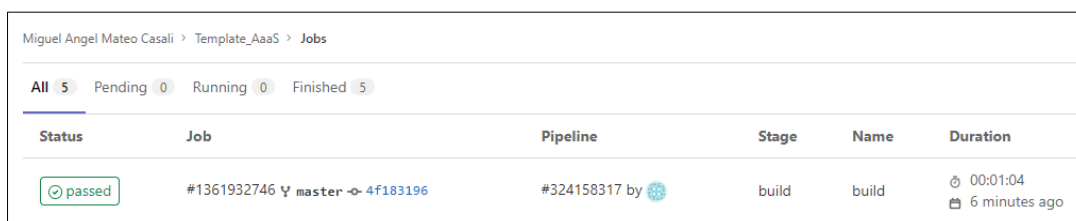
# Set up working directory
WORKDIR /usr/local/src/layers

# Copy files
COPY ./template/requirements.txt .
COPY ./template/compute_unit_problem ./compute_unit_problem

#Install requirements
RUN pip install --no-cache-dir -r requirements.txt
RUN pip install ./compute_unit_problem/

#Reset WORKDIR (TODO: Check if we actually need to reset the WORKDIR
WORKDIR /usr/local/app/
```

Tras esto, se comiteará el resultado. En ese momento se lanzará la tarea automatizada por medio del fichero de integración continua y se iniciará el proceso (también conocido como job) de creación del contenedor y testeo de calidad dentro de GITLAB (figura 49). Si el fichero pasa correctamente, se generará una imagen con este nuevo algoritmo como servicio dentro del *Container Registry* (figura 50).



The screenshot shows the GitLab CI/CD interface for a project named 'Template_AaaS'. The 'Jobs' section is active, showing a table with one job that has passed. The job details include the commit hash #1361932746, the pipeline ID #324158317, the stage 'build', and the job name 'build'. The duration of the job was 00:01:04, and it was completed 6 minutes ago.

| Status | Job | Pipeline | Stage | Name | Duration |
|--------|---|---------------|-------|-------|---------------------------|
| passed | #1361932746 <small>✓</small> master <small>⇄</small> 4f183196 | #324158317 by | build | build | 00:01:04 6 minutes ago |

Figura 49. Jobs CI/CD

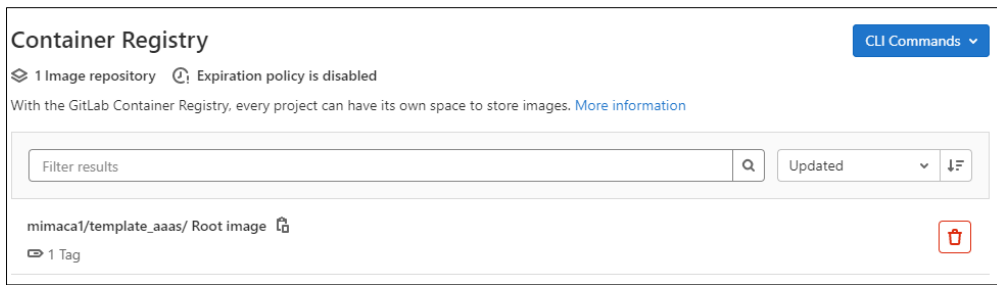


Figura 50. Container Registry

5.3 Preparación del entorno

El entorno donde se va a preparar el sistema de Algoritmos como Servicio será un servidor con Kubernetes gestionado con Rancher. A lo largo de este apartado, se verán las instrucciones necesarias para preparar este entorno y finalmente cargar el algoritmo generado en la fase anterior.

Primero, es necesario un entorno con Ubuntu que permita instalar Docker y Kubernetes. Las instrucciones que seguir para la instalación (siempre desde CLI) son las siguientes:

- Instalar Kubernetes (La distribución de kubernetes que instalo es la K3S. Esto es debido a que se trata de una distribución más liviana y consume menos recursos, por lo que para laboratorios o equipos de testeo es más adecuado):

```
#Instalamos K3S:

curl -sL https://get.k3s.io | sh -s - --write-kubeconfig-mode 644 --node-ip {YOUR-IP} --node-external {YOUR-IP}
# ejemplo:
curl -sL https://get.k3s.io | sh -s - --write-kubeconfig-mode 644 --node-ip 192.168.1.160 --node-external-ip 192.168.1.160
sudo chmod 644 /etc/rancher/k3s/k3s.yaml

#Comprobamos la instalación
kubectl version
kubectl get nodes
kubectl cluster-info

#A continuación instalamos Helm, que es una herramienta para gestionar paquetes
Kubernetes
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml

#Comprobamos la instalación
helm version
```



```
helm ls

#Instalamos Kompose, que es una herramienta que permite de forma automatica generar los
charts a partir de un docker-compose.
curl -L https://github.com/kubernetes/kompose/releases/download/v1.22.0/kompose-
linux-amd64 -o kompose
chmod +x kompose
sudo mv ./kompose /usr/local/bin/kompose

#Comprobamos la instalación
kompose versión
```

- Una vez instalado Kubernetes, se instala Docker para poder trabajar con las imágenes del registro de Docker:

```
#Eliminamos cualquier version Antigua instalada
sudo apt-get remove docker docker-engine docker.io containerd runc

#Actualizamos el índice de paquetes e instalamos las dependencias necesarias
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-
properties-common

#Descargamos la clave GPG oficial de Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"

#Configuramos el repositorio y actualizamos
sudo apt-get update

#Instalamos Docker y añadimos el usuario Docker al grupo
sudo apt-get install docker-ce docker-ce-cli containerd.io
sudo usermod -aG docker {USERNAME}

#Comprobamos la instalación
docker version
docker ps

#Instalamos docker-compose
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

#Comprobamos la instalación
Docker-compose version
```

Una vez instaladas ambas, Kubernetes y Docker, en el servidor, se procederá a instalar Rancher también en el servidor para facilitar la gestión de este por parte de los



Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

usuarios. Esta instalación se hará con Helm, que es una herramienta para gestionar paquetes de Kubernetes.

```
#Instalar certmanager que es un Controlador de Gestión de Certificados Nativo de
Kubernetes.
kubectl apply --validate=false -f https://github.com/jetstack/cert-
manager/releases/download/v0.15.0/cert-manager.crds.yaml
  kubectl create namespace cert-manager
  helm repo add jetstack https://charts.jetstack.io
  helm repo update
  helm install cert-manager jetstack/cert-manager --namespace cert-manager --
version v0.15.0
  export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
  kubectl get pods --namespace cert-manager

#Instalar Rancher
  helm repo add rancher-stable https://releases.rancher.com/server-charts/stable
  helm fetch rancher-stable/rancher --version=2.4.8
  kubectl create namespace cattle-system
  helm repo update
  helm install rancher rancher-stable/rancher --version=2.4.8 --namespace cattle-system --
set hostname=rancher.localhost

# Comprobamos la instalación
  kubectl -n cattle-system rollout status deploy/rancher
  kubectl -n cattle-system get deploy rancher
  kubectl get all --all-namespaces
```

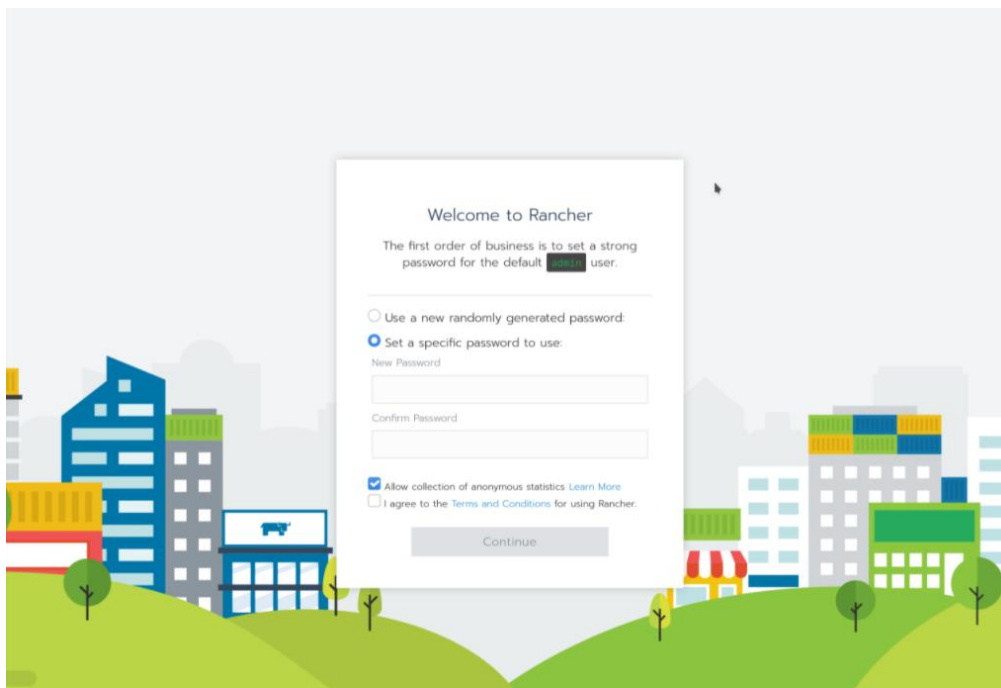


Figura 51. Pagina bienvenida Rancher

La primera vez que se accede a Rancher (figura 51) es necesario asignar una contraseña. El usuario por defecto es admin.

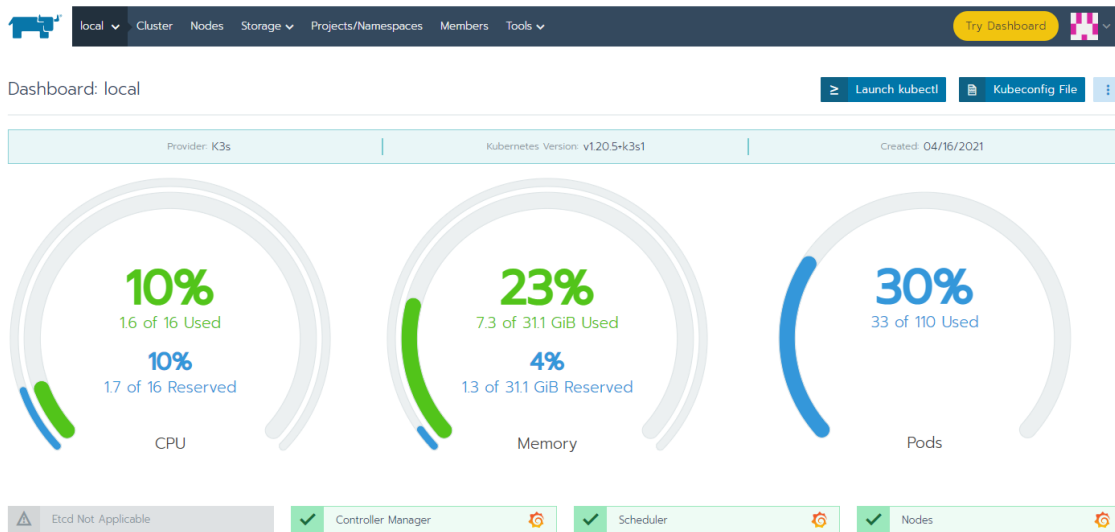


Figura 52. Dashboard Rancher

Si todo ha funcionado correctamente, tendremos nuestra aplicación de Rancher monitorizando el clúster de kubernetes (figura 52).

5.4 Preparar nuestro algoritmo como servicio

Una vez que el entorno esté listo, con la instalación completa de Kubernetes gestionado con Rancher, se prepararán los Helm charts para poder utilizar la imagen con el algoritmo en un entorno de Kubernetes. Para ello, se preparará el siguiente archivo Docker-compose.yml:

```
version: '3.3'
services:
  aaas:
    image: registry.gitlab.com/mimaca1/template_aaas
```

Una vez preparado, se utilizará Kompose para generar los Helms mediante la siguiente instrucción:

```
kompose convert -f docker-compose.yml -c --replicas=1
```

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

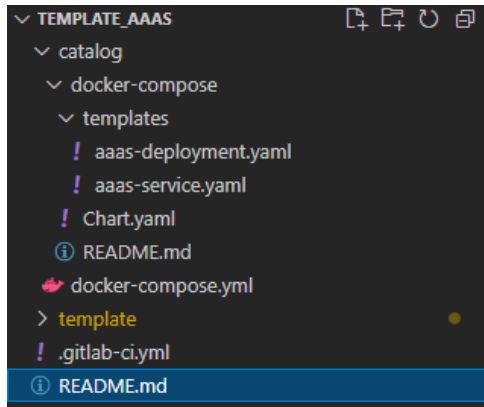


Figura 53. Estructura ficheros Helm

Esto generará una serie de ficheros (figura 53) dentro del proyecto, los cuales se podrán comitear y guardar en el repositorio para poder acceder a ellos desde Rancher. Una vez que el algoritmo está preparado, ya puede ser usado en un entorno Kubernetes. Se subirán estos ficheros a un entorno privado al que podremos acceder desde Rancher para desplegar el algoritmo.

Una vez subido, se añade al catálogo para poder acceder al servicio (figura 54):

Figura 54. Añadir catálogo Rancher

Una vez cargado y preparado, Rancher nos proporcionará un punto de acceso (*Endpoint*) al que acceder al servicio por medio de http, y ya estaría levantado:

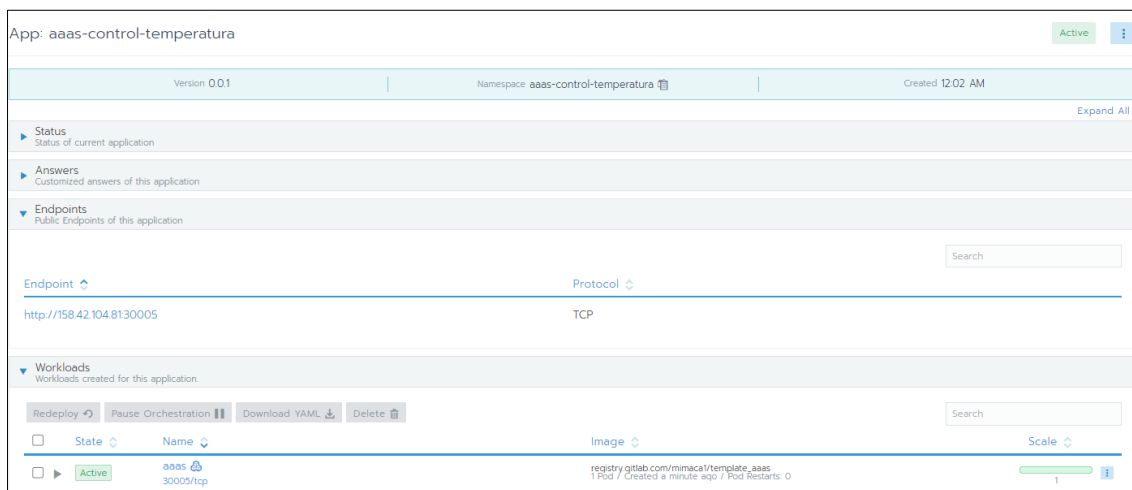


Figura 55. AaaS Desplegado

6. Conclusiones

El objetivo general de este trabajo de final de máster era desarrollar una arquitectura para la implantación de Algoritmos como Servicio (AaaS) en entornos de Industria 4.0. Teniendo esto en cuenta, el desarrollo del trabajo ha seguido distintos apartados y objetivos específicos que se han ido resolviendo de forma satisfactoria.

El primer paso ha sido realizar un estudio del concepto *cloud computing* y el uso de contenedores y orquestadores, con el fin de conocer el concepto de *serverless computing* y definir la estructura de un FaaS con el fin de entender su arquitectura. Llegados a este punto se han analizado las diferentes opciones de mercado existentes sobre plataformas FaaS y hemos analizado los puntos clave de este tipo de servicios. Por otro lado, hemos ahondado en el concepto de Fábrica en la nube y su relación con el *cloud computing*. A partir de aquí, se han ido revisando las iniciativas más relevantes y las arquitecturas de referencia que existen para la integración de soluciones técnicas dentro de la industria. Finalmente, tras todo este proceso, se ha seleccionado la arquitectura de IIRA como modelo de referencia a la hora de desarrollar la arquitectura para la implantación de Algoritmos como Servicios.

Una vez concluido el análisis previo, se ha desarrollado el proceso de implantación en tres fases bien definidas. Una primera que analiza los requisitos del sistema y un caso de uso para ellos, una segunda fase que consiste en examinar las capas de negocio y la usabilidad y una tercera fase en la que se ha desarrollado el punto de vista funcional y de implementación del servicio. En último lugar, con el fin de mostrar de forma práctica la arquitectura definida para este servicio, se ha desarrollado un proceso de implantación a modo de demostración del AaaS. Para el desarrollo de esta demostración, se han generado una serie de ficheros de integración continua para seguir una estructura definida y poder desplegar correctamente los algoritmos como servicio.

Durante el desarrollo de esta arquitectura he puesto en práctica conocimientos aprendidos durante el Máster Universitario en Ingeniería Informática, mi experiencia laboral previa y trabajando de forma autodidacta según iban surgiendo algunos problemas. Mi cotutor, Francisco Fraile Gil, como experto en tecnología asociada a la Industria 4.0, me ha ido guiando ante las dificultades que han ido surgiendo y proporcionándome la bibliografía necesaria, lo que me ha permitido poder acceder a ISOS, reglamentos y manuales que, de otra manera, no habría sido posible.

Desarrollo de una arquitectura para la implantación de Algoritmos como servicio (AaaS) en entornos de Industria 4.0

Para concluir, hay que indicar que este proyecto tiene un gran apartado de investigación y comprensión de las estructuras técnicas asociadas al campo industrial, en lo cual creo que los ingenieros informáticos tenemos la oportunidad de proporcionar muchas mejoras. Además, cabe destacar que este trabajo de final de máster tiene muchas posibilidades de futuro: desde desarrollar una herramienta más focalizada en la facilidad de integración de algoritmos por parte del usuario (ya sea añadiendo una interfaz gráfica o generado un sistema completo e intuitivo en red), hasta desarrollar un modelo de madurez para analizar el estado de implementación de los algoritmos dentro de la industria, pasando incluso por la posibilidad de generar un servicio donde podamos almacenar muchos más algoritmos de forma sencilla.

7. Referencias

- Allen, S., Browning, B., Calcote, L., Chaudhry, A., Davis, D., Fourie, L., Gulli, A., Haviv, Y., Krook, D., Nissan-Messing, O., Munns, C., Owens, K., Peek, M., Zhang, C., A., C., Amin, K., Chaudhry, A., Conway, S., Corleissen, Z., ... Zhao, P. (2018). CNCF WG-Serverless Whitepaper. *Cloud Native Computing Foundation*, 1–39.
- Barcelona-Pons, D., & García-López, P. (2020). *Benchmarking Parallelism in FaaS Platforms*. <http://arxiv.org/abs/2010.15032>
- Baur, C., & Wee, D. (2015). Manufacturing's next act - McKinsey and Company. *McKinsey and Company*, June, 1–5. https://www.timereaction.com/papers/manufacturing_next_act.pdf
- Behkamal, B., Kahani, M., & Akbari, M. K. (2009). Customizing ISO 9126 quality model for evaluation of B2B applications. *Information and Software Technology*, 51(3), 599–609. <https://doi.org/10.1016/j.infsof.2008.08.001>
- Confais, B., Lebre, A., & Parrein, B. (2017). Performance analysis of object store systems in a fog and edge computing infrastructure. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10430, 40–79. https://doi.org/10.1007/978-3-662-55696-2_2
- CORDIS. (2019). *Zero Defect Manufacturing Platform*. <https://cordis.europa.eu/project/id/825631>
- CORDIS. (2021). *Industrial Data Services for Quality Control in Smart Manufacturing*. <https://cordis.europa.eu/project/id/958205>
- Cortés, U., Cortés, A., & Barrué, C. (2019). Trustworthy AI. The AI4EU approach. *Proceedings of Science*, 372, 0–11.
- Danielle, C. (2020). *What are RAMI 4.0 and asset administration shells in the context of Industry 4.0?* <https://www.motioncontroltips.com/what-are-rami40-and-asset-administration-shells-in-the-context-of-industry40/>
- Docker. (2020). *Swarm mode overview | Docker Documentation*. Docker. <https://docs.docker.com/engine/swarm/>
- Electro Industria. (2016). *ARQUITECTURAS DE REFERENCIA RAMI 4.0 E IIRA*. <http://marefateadyan.nashriyat.ir/node/150>
- Fission. (2021a). *Fission: Key Features*. <https://fission.io/features/>
- Fission. (2021b). *Fission: Serverless Functions for Kubernetes*. <https://github.com/fission/fission>
- Garimella, K., Lees, M., & Williams, B. (2008). BPM (GERENCIA DE PROCESOS DE NEGOCIO). In *Introducción a BPM*. <https://doi.org/10.1161/01.RES.13.6.537>

- Gedeon, J., Brandherm, F., Egert, R., Grube, T., & Mühlhäuser, M. (2019). What the Fog? Edge Computing Revisited: Promises, Applications and Future Challenges. In *IEEE Access* (Vol. 7). <https://doi.org/10.1109/ACCESS.2019.2948399>
- Hassan, N., Gillani, S., Ahmed, E., Yaqoob, I., & Imran, M. (2018). The Role of Edge Computing in Internet of Things. *IEEE Communications Magazine*, 56(11), 110–115. <https://doi.org/10.1109/MCOM.2018.1700906>
- Hong, C.-H., & Varghese, B. (2019). Resource Management in Fog/Edge Computing: A Survey on Architectures, .. *ACM Computing Surveys*, 52(5), 1–37. <http://dl.acm.org/citation.cfm?doid=3362097.3326066>
- ICE, I. C. (2019). *Zero Defects Manufacturing Platform*. <https://es.zdmp.eu/>
- ISO 29148. (2018). *Systems and software engineering - Life cycle processes - Requirements engineering*.
- ISO British Standards Institution. (2007). *Enterprise-control system integration, Part 3: Activity models of manufacturing operations management (BS EN 62264-3:2007)* (Issue June).
- ISO.ORG. (2011). *ISO/IEC/IEEE 42010:2011 Systems and software engineering — Architecture description*. <https://www.iso.org/standard/50508.html>
- Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N., Gonzalez, J. E., Popa, R. A., Stoica, I., & Patterson, D. A. (2019). Cloud programming simplified: A Berkeley view on serverless computing. *ArXiv*, 1–33.
- Junín Durán de Leon, A., Cruz Rentería, J. R., Muñoz Zamora, G., Garcia-alva, S., Gutierrez-torres, L., & Sanchezz Hernández, Z. (2016). *Desarrollo de Software basado en el estándar ISA-95*. March.
- Kubernetes. (2019). *Kubernetes Documentation - Kubernetes*. Kubernetes. <https://kubernetes.io/docs/home/>
- Lin, S.-W., Miller, B., Durand, J., Bleakley, G., Amine, C., Robert, M., Brett, M., & Crawford, M. (2017). The Industrial Internet of Things Volume G1: Reference Architecture. *Industrial Internet Consortium White Paper, Version 1.*, 58 Seiten.
- Lin, S.-W., Murphy, B., Clauser, E., Loewen, U., Neubert, R., Bachmann, G., Pai, M., & Hankel, M. (2017). Architecture Alignment and Interoperability. *Plattform Industrie 4.0*, 19. https://www.iiconsortium.org/pdf/JTG2_Whitepaper_final_20171205.pdf
http://www.iiconsortium.org/pdf/JTG2_Whitepaper_final_20171205.pdf
- Lu, Y., Xu, X., & Xu, J. (2014). Development of a Hybrid Manufacturing Cloud. *Journal of Manufacturing Systems*, 33(4), 551–566. <https://doi.org/10.1016/j.jmsy.2014.05.003>

- Mateo-Casali, M. A., Boza, A., & Fraile Gil, F. (2019). *Implementación de una aplicación para la guía en la implantación de sistemas de fabricación (MES)*. <https://riunet.upv.es/handle/10251/124983>
- Mehrpouya, M., Dehghanghadikolaei, A., Fotovvati, B., Vosooghnia, A., Emamian, S. S., & Gisario, A. (2019). The potential of additive manufacturing in the smart factory industrial 4.0: A review. *Applied Sciences (Switzerland)*, 9(18). <https://doi.org/10.3390/app9183865>
- Michael, H., Fang, L., Annie, S., & Jin, T. (2011). The NIST definition of cloud computing. *NIST Cloud Computing Standards RoadMap, First Edit*. <https://doi.org/10.6028/NIST.SP.800-145>
- Pedone, G., & Mezgár, I. (2018). Model similarity evidence and interoperability affinity in cloud-ready Industry 4.0 technologies. *Computers in Industry*, 100(May), 278–286. <https://doi.org/10.1016/j.compind.2018.05.003>
- Pisching, M. A., Pessoa, M. A. O., Junqueira, F., dos Santos Filho, D. J., & Miyagi, P. E. (2018). An architecture based on RAMI 4.0 to discover equipment to process operations required by products. *Computers and Industrial Engineering*, 125(January), 574–591. <https://doi.org/10.1016/j.cie.2017.12.029>
- Popkova, E. G., Ragulina, Y. v., & Bogoviz, A. v. (2018). Industry 4.0: Industrial Revolution of the 21st Century. In *Studies in Systems, Decision and Control* (Vol. 169). https://doi.org/10.1007/978-3-319-94310-7_6
- Prades, L., Romero, F., Estruch, A., García-Dominguez, A., & Serrano, J. (2013). Defining a methodology to design and implement business process models in BPMN according to the standard ANSI/ISA-95 in a manufacturing enterprise. *Procedia Engineering*, 63(2004), 115–122. <https://doi.org/10.1016/j.proeng.2013.08.283>
- Priya Wadhwa. (2018). *Introducing kaniko: Build container images in Kubernetes and Google Container Builder without privileges*. <https://cloud.google.com/blog/products/containers-kubernetes/introducing-kaniko-build-container-images-in-kubernetes-and-google-container-builder-even-without-root-access>
- Quevedo, S., Merchan, F., Rivadeneira, R., & Dominguez, F. X. (2019). Evaluating Apache OpenWhisk - FaaS. *2019 IEEE 4th Ecuador Technical Chapters Meeting, ETCM 2019*. <https://doi.org/10.1109/ETCM48019.2019.9014867>
- R. Edward, F. (2010). *Strategic management: A stakeholder approach*.
- RedHat. (2020). *OpenShift Container Platform 4.1*. <https://docs.openshift.com/container-platform/4.7/welcome/index.html>
- Shahrad, M., Balkind, J., & Wentzlaff, D. (2019). Architectural implications of function-as-a-service computing. *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, 1063–1075. <https://doi.org/10.1145/3352460.3358296>

- Unver, H. O. (2013). An ISA-95-based manufacturing intelligence system in support of lean initiatives. *International Journal of Advanced Manufacturing Technology*, 65(5–8), 853–866. <https://doi.org/10.1007/s00170-012-4223-z>
- Wu, D., Rosen, D. W., Wang, L., & Schaefera, D. (2014). Cloud-based manufacturing: Old wine in new bottles? *Procedia CIRP*, 17, 94–99. <https://doi.org/10.1016/j.procir.2014.01.035>
- ZDMP. (2021). *Prediction and Optimisation Run-Time*. <https://angry-spence-1ecc9e.netlify.app/docs/components/platform-tier/prediction-and-optimisation-runtime/>
- Zhou, H., Benton, W. C., Schilling, D. A., & Milligan, G. W. (2011). *Integración de la cadena de suministro y modelo SCOR*. 32(4), 332–344.
- Zhu, L., Johnsson, C., Varisco, M., & Schiraldi, M. M. (2018). Key performance indicators for manufacturing operations management - Gap analysis between process industrial needs and ISO 22400 standard. *Procedia Manufacturing*, 25, 82–88. <https://doi.org/10.1016/j.promfg.2018.06.060>