UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de un catálogo de recursos hardware para federación de sistemas de computación

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: González Altimiras, Javier

Tutor/a: Muñoz Escoí, Francisco Daniel

Cotutor/a externo: Arjona Aroca, Jorge

CURSO ACADÉMICO: 2023/2024

Resum

L'objectiu d'aquest TFG és el desenvolupament d'un catàleg de recursos hardware que permeta la federació de servicis informàtics de computació; és a dir, la integració de recursos de diferents entorns i proveïdors per a una presentació de manera conjunta i coordinada. En aquest treball es realitzarà una anàlisi dels paràmetres més rellevants sobre diferents recursos hardware i es seleccionaran aquells més adequats per a la implementació del catàleg. A més, es definiran els requisits funcionals i no funcionals del mateix. Amb aquest treball s'espera contribuir al desenvolupament de solucions que permeten una gestió més eficient i flexible dels sistemes informàtics.

Paraules clau: Catàleg, Federació, Recursos hardware

Resumen

El objetivo de este TFG es el desarrollo de un catálogo de recursos hardware que permita la federación de servicios informáticos de computación; es decir, la integración de recursos de diferentes entornos y proveedores para su presentación de manera conjunta y coordinada. En este trabajo se realizará un análisis de los parámetros más relevantes acerca de distintos recursos hardware y se seleccionarán aquellos más adecuados para la implementación del catálogo. Además, se definirán los requisitos funcionales y no funcionales del mismo. Con este trabajo se espera contribuir al desarrollo de soluciones que permitan una gestión más eficiente y flexible de los sistemas informáticos.

Palabras clave: Catálogo, Federación, Recursos hardware

Abstract

The objective of this Final Degree Project is the development of a catalogue of hardware resources which allows the federations of computing services; that is, the integration of resources from different environments and providers for their joint and coordinated presentation. In this work, an analysis of the most relevant parameters about different hardware resources will be carried out and the most appropriate ones for the implementation of the catalogue will be selected. In addition, the functional and non-functional requirements of the same will be defined. This work is expected to contribute to the development of solutions that allow a more efficient and flexible management of computer systems.

Key words: Catalogue, Federation, Hardware resources

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

La computación en la nube ha tenido un impacto significativo en la forma en que empresas y organizaciones gestionan sus recursos informáticos. En vez de invertir en *hardware* costoso y recursos humanos para administrarlo y mantener su infraestructura, cada vez más organizaciones se están moviendo hacia la nube para aprovechar ventajas como la escalabilidad, la flexibilidad y la eficiencia. Sin embargo, aunque la nube ofrece muchas ventajas, también presenta desafíos únicos en términos de gestión y coordinación de recursos. En particular, la integración de estos sistemas de diferentes proveedores y entornos puede ser un desafío significativo. En este trabajo se afrontará este problema mediante el desarrollo de un catálogo de recursos *hardware* de manera que permita la federación de recursos informáticos de computación. El catálogo desarrollado permitirá la integración de dispositivos de diferentes entornos y proveedores para su presentación de manera conjunta y coordinada. Para ello, en este trabajo se presentará un estudio acerca del estado del arte en este campo, posteriormente un análisis detallado de los requisitos funcionales y no funcionales del catálogo para garantizar su operatividad y rendimiento óptimos, además de los detalles de la implementación. Incluyendo también las pruebas y la implantación del sistema en un entorno real.

En particular, este trabajo se enfoca en el desarrollo de un catálogo. Este catálogo se desarrollará utilizando tecnologías y herramientas actuales y se enfocará en la escalabilidad y la eficiencia. Se espera que el resultado de este trabajo sea una solución práctica y escalable que permita a las organizaciones integrar y coordinar de manera efectiva los recursos informáticos de diferentes entornos y proveedores. Una vez definidos los requisitos, se procederá a la implementación del catálogo, de un *script* para recoger los datos y de una ontología para organizarlos. Para la recolección de los datos que conformarán el catálogo, se hará uso de un *script* en lenguaje *bash*. Se implementará el catálogo haciendo uso de la herramienta Apache Kafka, utilizándola como sistema de mensajería para transmitir los detalles de los servicios, además de utilizar su persistencia para guardar estos detalles. Y finalmente, la ontología se llevará a la práctica mediante un JSON Schema. Esta ontología tendrá como objetivo describir, de todos los parámetros que se pueden extraer de una máquina, cuáles son los más relevantes. Estos serán los que posteriormente serán publicados en el catálogo.

Finalmente, se espera que este TFG sea de interés para estudiantes, investigadores, profesionales y empresas que trabajan en el campo de la gestión de recursos informáticos en entornos de nube. El catálogo de recursos *hardware* desarrollado en este trabajo representa una solución innovadora y eficiente para dicha gestión de recursos en entornos de nube, y su implementación y pruebas buscarán demostrar su utilidad y eficacia en situaciones reales.

1.1 Motivación

Este trabajo se ha desarrollado bajo el marco de unas prácticas curriculares de empresa, dicha empresa es el Instituto Tecnológico de Informática (ITI). Por lo tanto, los objetivos del proyecto están marcados por tal entidad.

El desarrollo de un catálogo de recursos *hardware* para la federación de recursos de computación es una tarea crucial para optimizar la utilización de recursos en la computación distribuida. Este proyecto de TFG busca abordar los desafíos de la gestión de recursos en entornos distribuidos, proporcionando una herramienta que permita gestionar de manera más eficiente los recursos *hardware* y, por lo tanto, que sea capaz de ofrecer recursos de distintos proveedores. Además, este proyecto ofrece una oportunidad única para explorar la tecnología de la computación distribuida y comprender su impacto en la gestión de recursos, lo cual es cada vez más importante en el mundo actual, donde encontramos grandes cantidades de datos y demandas de procesamiento.

1.2 Objetivos

El principal objetivo del trabajo consiste en el desarrollo de un gestor de recursos *hardware* en un entorno federal, para lo cual es necesario decidir qué información habrá que recoger de cada uno de los recursos federados. Para la organización y la recogida de todos estos datos requeriremos del uso de una ontología.

Tal como hemos dicho, desarrollar la ontología será el objetivo inicial. El desarrollo de esta ontología cuenta para ello con dos propósitos, en primer lugar diferenciar del resto aquellos parámetros que son considerados relevantes a la hora de decidir en la elección de conectarse a un sistema u otro, para lo cual se habrá de hacer un estudio minucioso de todas estas características de las máquinas, para no dejarse ningún parámetro crucial. El segundo de los propósitos tiene la intención de que contemos con los mismos parámetros de todas las máquinas, tratando de evitar en la medida de lo posible valores nulos en el catálogo.

Otro de los objetivos principales consiste en conocer, utilizar y aprender acerca del *middleware* de mensajería Apache Kafka. Se trata de una tecnología de intercambio de mensajes de tipo publicador-suscriptor.

También es importante destacar como objetivo principal la creación de una federación de recursos. El término de «federación», según la RAE, se refiere a la unión o asociación entre personas o grupos sociales para lograr un fin común [1]. En este caso, el fin común consiste en el desarrollo de una nube conformada por distintos proveedores y cada uno de los sistemas que ofertan.

1.3 Impacto esperado

El presente trabajo busca ofrecer una solución práctica y concreta a un problema creciente en el ámbito de la computación, donde la diversidad de proveedores y tecnologías puede dificultar la gestión unificada de recursos. Por lo tanto, proporcionando un catálogo que identifique y clasifique los recursos *hardware* más adecuados para la federación de sistemas, se espera facilitar la toma de decisiones por parte de los gestores de IT y promover un uso más eficiente de los recursos disponibles.

Además, el impacto de este trabajo no se limita únicamente al ámbito académico, sino que también se puede trasladar al mundo empresarial, donde la optimización de

recursos y la mejora en la prestación de servicios informáticos son aspectos esenciales para la competitividad y el éxito empresarial.

El desarrollo de un catálogo de recursos *hardware* para la federación de sistemas de computación tendrá un impacto significativo en varios niveles, tanto para los usuarios directos como para la sociedad en general. En primer lugar, los usuarios finales de este catálogo (que pueden ser empresas, organizaciones gubernamentales o investigadores) experimentarán una mejora notable en la eficiencia y flexibilidad de sus sistemas informáticos. Al poder integrar recursos de diferentes proveedores y entornos de manera coordinada, se podrán aprovechar al máximo los recursos disponibles, optimizando costos y tiempos de procesamiento, tanto por parte de los proveedores como de los usuarios finales de estos recursos de cómputo.

Por otro lado, los proveedores de servicios informáticos también se verán beneficiados, ya que podrán ofrecer soluciones más completas y adaptadas a las necesidades específicas de sus clientes. Esto podría traducirse en una mayor competitividad en el mercado y en la posibilidad de abrir nuevas oportunidades de negocio.

A nivel más amplio, el desarrollo de este catálogo cuenta como una contribución en el campo de la computación distribuida y la gestión de recursos en la nube, áreas que son fundamentales en la actualidad y que previsiblemente lo seguirán siendo en el futuro. La optimización de recursos informáticos es crucial para abordar desafíos contemporáneos, como el procesamiento masivo de datos, el desarrollo de inteligencia artificial y la investigación en áreas científicas y tecnológicas, así como para tratar de reducir la huella de carbono generada.

En términos de los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas, este trabajo puede estar relacionado con varios de ellos. Por ejemplo, contribuiría al ODS 9 (Industria, innovación e infraestructura) al promover el desarrollo de infraestructuras tecnológicas más eficientes y accesibles. También podría tener un impacto positivo en el ODS 11 (Ciudades y comunidades sostenibles) al facilitar el acceso a recursos informáticos avanzados en áreas urbanas y rurales. Además, al optimizar el uso de recursos, podría contribuir al ODS 12 (Producción y consumo responsables) al reducir el desperdicio de recursos informáticos y energéticos.

En resumen, el desarrollo de este catálogo de recursos *hardware* para la federación de sistemas de computación tiene el potencial de generar beneficios tangibles para los usuarios directos, los proveedores de servicios y la sociedad en su conjunto, al tiempo que contribuye a la consecución de los Objetivos de Desarrollo Sostenible.

1.4 Metodología

El proyecto presentó una serie de desafíos significativos, principalmente porque tanto el problema a resolver como las tecnologías necesarias para implementar la solución no habían sido abordados a lo largo del Grado de Informática. Para superar este obstáculo, fue necesario emplear cierto tiempo en formación específica en esas nuevas tecnologías, como por ejemplo las ontologías relacionadas con equipos informáticos, el uso de sistemas de coordenadas de red y los sistemas de mensajería. Esta formación se basó en dos enfoques principales: el estudio de artículos académicos relevantes para temas más teóricos como las ontologías y las coordenadas, y la consulta de tutoriales y de la documentación técnica disponible online para comprender y aplicar herramientas prácticas, como los sistemas de mensajería.

La metodología de trabajo adoptada fue iterativa, lo que implicó un proceso cíclico de desarrollo y refinamiento de la solución. Sin embargo, este enfoque también presen-

tó sus propios retos. Normalmente, el desarrollo de este tipo de proyectos se realiza en equipo, lo que facilita el intercambio de ideas y la resolución conjunta de problemas. En este caso, trabajar de manera individual significó enfrentarse solo a los desafíos técnicos y conceptuales. Aunque se contaba con reuniones periódicas de supervisión, propias de una metodología iterativa, pero quizá esas reuniones deberían haber proporcionado mayor guía o ayuda.

1.5 Estructura de la memoria

En este trabajo los contenidos están divididos en capítulos estructurados de la siguiente manera:

1. Introducción: se presenta el punto de partida del trabajo, así como la motivación y los objetivos

2. Estado del arte: se presenta un estudio de la literatura existente acerca de las ontologías para describir recursos *hardware*, así como catálogos de recursos de tipo *hardware*.

3. Análisis del problema: se detallan los requisitos funcionales y no funcionales del catálogo, así como los casos de uso del mismo.

4. Diseño de la solución: se presenta la arquitectura del proyecto, incluyendo las tecnologías y herramientas utilizadas para su implementación, así como todas las decisiones que se han tenido que tomar.

5. Desarrollo de la solución: se detallará el progreso desde el diseño de la solución hasta su implementación teniendo en cuenta los problemas que han surgido, las decisiones que se han tenido que tomar durante este desarrollo al igual que las particularidades de esta solución.

6. Implantación: se expondrán la puesta en marcha del proyecto en un entorno real

7. Pruebas: se detallan las pruebas realizadas para verificar el correcto funcionamiento de cada uno de los componentes del sistema.

8. Conclusiones: por último, se exponen las conclusiones, comparándolas con los objetivos inicialmente fijados y se discuten las posibles mejoras y trabajos futuros, así como la relación con las asignaturas cursadas durante el grado.

1.6 Convenciones

En el presente Trabajo de Fin de Grado se utilizará a nivel estilístico la cursiva para indicar palabras extranjeras, como cloud o hardware. Asimismo se entrecomillan los nombres de archivos mencionados a lo largo de la memoria. Del mismo modo, se entrecomillan también los comandos utilizados para el desarrollo del proyecto.

CAPÍTULO 2
Estado del arte

Actualmente, la gestión de recursos en la nube se trata de un aspecto de gran importancia en el campo de la informática. A diferencia de Estados Unidos, en nuestro continente no contamos con una empresa que posea una nube propia y centralizada. Por lo tanto, es más conveniente contar con la colaboración de varios proveedores aislados que unan sus recursos para presentarlos de manera coordinada en lugar de esperar a que una empresa construya su propia red unificada. En este contexto, surge la idea de este trabajo de fin de grado, ya que podría ser aprovechado por distintos proyectos europeos, al tratarse de inteligencias artificiales distribuidas podrían requerir una federación como la que se propone.

En este capítulo se van a incluir las siguientes secciones. En primer lugar, en la sección 2.1 se van a comentar varios proyectos de investigación, posteriormente en la sección 2.2 se disertará acerca de la ontología y los lenguajes que se pueden utilizar en ella. A continuación, a lo largo de la sección 2.3 se hablará sobre diferentes sistemas de coordenadas de red, NCS según sus siglas en inglés. Más tarde se comentarán diferentes sistemas de mensajería 2.4 y seguidamente explicará la tecnología utilizada y se justificará la decisión. Por último se realizará la crítica al estado del arte y la propuesta en la que se enmarca este trabajo.

2.1 Proyectos de investigación relacionados

Entre ellos, destaca el proyecto GAIA-X [2], que se centra en la creación de una infraestructura de datos abierta y segura. Cuenta con diferentes proyectos dentro de lo que se engloba como GAIA-X. Todos ellos teniendo en común el uso de una IA distribuida que tras una recolecta de múltiples datos, se entrena utilizándolos para así tomar la mejor decisión posible respecto a diferentes temas. Otro de sus objetivos es garantizar la seguridad de los datos publicados por los usuarios a las nubes federadas, garantizando que estos no pierden el control sobre su información. Dentro de este marco, que busca potenciar la comunidad del dato, también se busca incorporar soluciones o servicios que puedan desarrollarse siguiendo este marco de protección del dato del usuario final. Estos temas o casos de uso son de diferentes asuntos, estos pueden ser del campo médico como en el caso del *CARECOL Project*. Este proyecto se centra en el cáncer de estómago, y consiste en la creación de una base de datos (de lesiones precancerosas, datos clínicos, datos diagnósticos...) que puedan servir para el entrenamiento de una IA con el objetivo de poder detectar y tratar antes este tipo de cáncer y también mejorar la investigación en este campo. Otro ejemplo de caso de uso de GAIA-X podría ser el proyecto de mantenimiento predictivo de la red de carreteras del sur del Tirol. Teniendo en cuenta que no hay recursos ilimitados por parte del gobierno para realizar el mantenimiento de las carrete-

ras, a partir de datos como el presupuesto disponible, la última fecha de mantenimiento, la calidad de la carretera, sus características o el uso, se entrena una IA para gestionar este mantenimiento, de la manera más efectiva y eficiente posible.

Por otro lado, otro proyecto europeo también relacionado es *Artificial Intelligence for Europe*[3], en el cual se está desarrollando una plataforma denominada *AI-on-demand* (AIOD). ¿Y qué significa este AIOD? Se refiere a la capacidad de solicitar y obtener acceso a recursos de inteligencia artificial típicamente por medio de *cloud* sin la necesidad de desarrollar una infraestructura propia. Estos recursos pueden ser inteligencias de procesamiento del lenguaje natural. Esta plataforma destaca en cuanto al ahorro, concretamente en la cantidad de la inversión inicial requerida para acceder a este tipo de infraestructura.

También hay una gran cantidad de artículos de investigación relacionados de una u otra manera con el campo de desarrollo del TFG. Cabe destacar que esta es tan solo una pequeña recopilación de todos los artículos científicos sobre el tema. Los más relevantes de entre los encontrados, son los siguientes:

En primer lugar, Castañé et al.[4] presentan una propuesta de ontología para mejorar la interoperabilidad en infraestructuras de nube heterogéneas. En entornos de computación en la nube, la creciente diversidad de recursos está generando la necesidad de gestionar eficientemente tanto los recursos heterogéneos como los tradicionales, lo que a su vez plantea desafíos de interoperabilidad. En este artículo se propone una ontología, denominada CloudLightning Ontology (CL-Ontology), la cual está basada en la ontología *cloud* mOSAIC[5], la cual es el fundamental pilar del estándar IEEE 2302 para la Interoperabilidad y Federación entre Nubes. Esta extensión de la ontología busca abordar específicamente la incorporación de recursos heterogéneos y entornos de computación de alto rendimiento (HPC) en la nube. Además, se presenta una arquitectura genérica para gestionar la heterogeneidad en la nube, ofreciendo un enfoque unificado para resolver los problemas de interoperabilidad. El artículo destaca la importancia de abordar estos problemas desde una perspectiva de una nube unificada, considerando tanto la interoperabilidad entre nubes como la integración de recursos heterogéneos en entornos de nube. Se menciona que la ontología y la arquitectura propuestas se integran en el proyecto CloudLightning, diseñado para entornos de computación en la nube a gran escala. Este trabajo es relevante para el desarrollo del catálogo de recursos *hardware* en el contexto de la federación de sistemas de computación, ya que aborda la gestión de la heterogeneidad de recursos y está enfocado sobre todo en los recursos de alto rendimiento, en entornos de nube. La adopción de ontologías y arquitecturas específicas puede ayudar a facilitar la integración y gestión eficiente de los recursos en sistemas informáticos federados, contribuyendo así a una gestión más flexible y eficaz de los sistemas informáticos.

Por su parte, Kousiouris et al.[6] abordan un tema muy relevante del ámbito de la federación de servicios informáticos en la nube: la gestión de datos, en concreto los datos personales, focalizado para entornos de *clouds* privadas así como de las federadas. El texto destaca los desafíos que enfrentan las organizaciones debido a la legislación existente, que impone restricciones y límites sobre el uso de los datos por parte de proveedores de nube externos. La propuesta presentada en el artículo consiste en un esquema de descripción y mecanismo de uso que incluye varios niveles de información legal necesaria para automatizar el proceso de selección de proveedores de nube y externalización de datos. Este mecanismo se llama Cloud Provider Description Schema(CPDS), y en nuestro escenario este esquema de descripción mencionado será la ontología. El CPDS tiene como objetivo permitir que las organizaciones verifiquen los requisitos legales de manera automatizada y comprensible para las máquinas, aprovechando así el potencial creado por los avances en la computación en la nube, como la federación dinámica. Identifica lagunas legales y acciones necesarias para que la automatización evite pasos manuales y burocráticos que actualmente son necesarios. Esto es fundamental para garantizar una

federación de nubes efectiva y cumplir con las normativas legales relacionadas con la gestión de datos en entornos de nube, que al estar almacenados en la federación deberán cumplir todos sus miembros.

Dumss et al.[7] proponen una arquitectura de sistema industrial diseñada para satisfacer las necesidades de los sistemas de producción impulsados por datos. Actualmente, la mayoría de las soluciones de *IoT* para el ecosistema de producción se derivan de tendencias que primero se establecieron en el mercado y que estaban dedicadas al consumidor. Aunque muchos de estos conceptos se han adaptado bien en el entorno industrial, se relata como en bastantes casos han dado lugar a soluciones fragmentadas que requieren interfaces complejas. Por lo que es una propuesta que busca mejorar los entornos de producción mediante la creación de un ecosistema de datos industriales. La reciente introducción de GAIA-X[2] abre la posibilidad de desarrollar soluciones de *IoT* independientes de plataformas, adaptadas específicamente a las necesidades de los fabricantes. Hasta ahora, GAIA-X era solo un concepto propuesto por gobiernos y consejos asesores económicos. Este artículo extiende el concepto hacia una arquitectura de sistema industrial que permite el intercambio confiable de información entre la cadena de suministro de una red de producción altamente distribuida. El enfoque propuesto busca abordar la complejidad y fragmentación en las soluciones de *IoT* para la industria, permitiendo una integración más fluida y eficiente de los sistemas de producción basados en datos. La arquitectura propuesta tiene como objetivo facilitar la interoperabilidad y el intercambio de información en entornos de producción distribuidos, lo que puede conducir a una mayor eficiencia y agilidad en las operaciones industriales. A lo largo de este trabajo se presenta una perspectiva valiosa para el desarrollo del catálogo de recursos *hardware* en el contexto de la federación de sistemas de computación, ya que se destaca la importancia de una arquitectura sólida y adaptable para apoyar los sistemas de producción centrados en la economía del dato. La integración de soluciones como las propuestas en este artículo puede contribuir significativamente a la mejora de la gestión y operación de los sistemas informáticos federados en entornos industriales. La implementación inicial del proyecto está enfocada al alojamiento de servicios que proporcionen flujos de datos básicos desde dispositivos de *hardware* muy simples en el campo y con el objetivo principal de agregar valor al enriquecer los datos desde el principio.

De manera relacionada con su proyecto, Attardi et al.[8] abordan una problemática. En concreto, viene derivada de la reducción de presupuestos para la investigación en muchas universidades públicas, debido a la crisis económica global. Esta reducción presupuestaria mencionada obliga a las universidades a ajustar sus gastos tanto en investigación como en administración. En este contexto, la posibilidad de compartir recursos ofrecidos por las nubes federadas representa una oportunidad para reducir los gastos en recursos de *hardware* y *software*, al tiempo que mejora la colaboración entre estas instituciones educativas, de manera que pueden mantener su capacidad de cómputo y almacenamiento dentro de unos niveles aceptables. En este trabajo, se explotan los servicios ofrecidos por la GARR Federated Cloud para implementar un sistema eficiente de respaldo para la documentación producida por la oficina administrativa de la Universidad Vanvitelli, situada en Italia. Este enfoque se presenta como una alternativa más económica y confiable, así como más rápida en comparación con el sistema actual basado en NAS, es un sistema de almacenamiento en red ampliamente utilizado. Dentro del artículo se destaca la utilidad de las nubes federadas en el ámbito académico para optimizar los recursos y mejorar la colaboración entre universidades. Al utilizar servicios de nube federada, las universidades pueden reducir costos operativos y mejorar la eficiencia en la gestión de la documentación. Este trabajo proporciona un ejemplo práctico de cómo las universidades pueden aprovechar las tecnologías de nube federada para optimizar sus recursos y mejorar sus servicios de investigación y administración. La implementación

exitosa de esta solución respaldada por la GARR Federated Cloud puede servir como un modelo para otras instituciones públicas que enfrentan desafíos similares en la gestión de recursos informáticos.

2.2 Ontología

Es importante remarcar la necesidad de la utilización de una ontología para el desarrollo del proyecto. Fundamentalmente por los siguientes motivos: en primer lugar, la organización de los datos recogidos de manera coherente, y en segundo lugar también es importante saber qué detalles de las máquinas son los requeridos para incorporarse a una federación, o también aquellos considerados relevantes para que el cliente pueda tomar una decisión. En los próximos párrafos vamos a comentar acerca de diferentes mecanismos que se pueden utilizar para la organización de los datos más destacados.

Por un lado tenemos *GLUE Schema*. Es una funcionalidad de la que hace uso la tecnología AWS que permite descubrir y controlar diferentes esquemas, siendo estos esquemas una definición del formato y la estructura de un conjunto de datos, es decir, de una ontología. El hecho de registrar un nuevo conjunto de datos correspondiente a una ontología determinada conlleva varios pasos: en primer lugar, es necesario registrar el esquema. Posteriormente, al incluir un conjunto de datos relacionado con esa ontología, se verifica que coincide la estructura del esquema registrado con la estructura del conjunto de datos que se quiere incluir. En caso de que se verifique el paso anterior, se serializa el conjunto de datos y se puede comprimir (a decisión del usuario) y se envía al destinatario. Más tarde el consumidor o destinatario, en caso de no conocer el esquema lo solicita y deserializa los datos (en caso que sea necesario), y, por último la aplicación se encarga de procesar estos datos.

Como alternativas de lenguajes para el desarrollo de la ontología podemos encontrar las siguientes opciones: en primer lugar tenemos RDF (Resource Description Framework), se trata de un *framework* utilizado para representar e intercambiar información en la web. Este está basado en tripletas formadas del modo sujeto-predicado-objeto, de manera que cada sujeto representa un recurso, cada predicado indica una relación y cada objeto representa otro recurso. Por lo tanto, es muy útil a la hora de conformar relaciones entre recursos. Similar a RDF aunque no igual, tenemos RDFS (RDF Schema), que se trata de un lenguaje construido sobre RDF, es decir, una extensión de RDF. Esta permite la creación de ontologías y aporta un vocabulario básico que permite definir clases, propiedades de estas clases y relaciones entre recursos. Es más simple que OWL (Web Ontology Language) pero útil para ontologías y razonamientos sencillos. Está diseñado para definir ontologías en la web semántica. ¿Qué es la web semántica? Es la visión desde el punto de vista de los ordenadores de la web, es decir, un conjunto de metadatos, que luego son utilizados por las *search engines* a la hora de conseguir mejores resultados en las consultas. Por lo que, en resumen, RDFS añade vocabulario a RDF. Otra herramienta disponible alternativa a JSON Schema es OWL. Que es un lenguaje que sirve para crear ontologías en la web. Así como RDFS, también está diseñado para definir ontologías en la web semántica. OWL añade funcionalidades a RDFS de manera que mejora las capacidades de modelado de este así como también posibilita un razonamiento con un mayor nivel de complejidad en la ontología, como por ejemplo permitiendo restricciones entre recursos o agregando la propiedad de la cardinalidad. OWL está basado en DL (Description Logics), este último se trata de un una lógica que se utiliza en ontologías para el modelado de conceptos, propiedades y relaciones entre diferentes recursos, así como entre diferentes ontologías. Por último, tenemos JSON Schema, que comparte muchas características de estilo con el conocido formato de intercambio de datos JSON (JavaScript Object No-

tation) y permite añadir diferentes tipos de restricciones, como por ejemplo propiedades requeridas o especificar los diferentes tipos de los datos.

2.3 Sistema de coordenadas de red

Para la parte de la interfaz del proyecto se ha realizado un estudio acerca de distintos sistemas de coordenadas de la red. El objetivo de incorporar alguno de estos sistemas al proyecto reside en implantar alguno de ellos para conseguir una mejora en la calidad de la decisión a la hora de desplegar servicios que incluyan varios centros de datos. Entre los sistemas de coordenadas destacan los siguientes. Como «fundador» de este tipo de tecnología, tenemos a las coordenadas llamadas como Vivaldi[9]. Es un sistema que data del 2004, está desarrollado por Frank Dabek, Russ Cox, Frans Kaashoek y Robert Morris, investigadores del MIT, y está orientado a sistemas distribuidos. Esta tecnología forma un mapa de coordenadas que indica lo cercano o lejano que está otro nodo. Esto se realiza mediante la técnica conocida como *piggybacking*, por lo que no añade más tráfico al sistema. En cada uno de los nodos se muestra un mapa de coordenadas euclídeo, por lo que para calcular las latencias basta con realizar una simple operación de resta de coordenadas y su posterior raíz en función de cuántas dimensiones tenga el mapa de coordenadas. El hecho de aumentar el número de dimensiones añade grados de dificultad al cálculo de la distancia, pero reduce el error relativo, aunque el aumentar la cantidad de dimensiones para lograr mayor precisión tiene un límite, en función del sistema con el que se trabaje. Para esbozar el mapa, se colocan todos los nodos del sistema en los ejes de coordenadas, y a medida que se comunican entre ellos se va esbozando el mapa, su aspecto al inicio se podría comparar con el Big Bang. El esbozo del mapa de coordenadas toma como modelo un muelle, en el cual las medidas van variando, se van acercando o alejando hasta que llegan a un punto de estabilidad. Los sistemas de coordenadas euclideos cuentan con un problema, las Violaciones de la Desigualdad del Triángulo (TIVs), estas, son situaciones en las que las estimaciones de latencia no son calculadas correctamente, dado que dados 3 nodos, como se puede ver en la figura 2.1, la distancia de A a C sea menor que la de A a B sumado a la de B a C. Esto puede ocurrir debido a la dinámica naturaleza de Internet. En relación con los sistemas NC, las TIVs tienen un impacto negativo en la precisión de las estimaciones de latencia realizadas por Vivaldi, lo que lleva a errores en la predicción de distancias entre nodos en la red. Por lo tanto, abordar las TIVs es esencial para mejorar la eficacia de sistemas como Vivaldi en la estimación de latencia en entornos de red complejos.

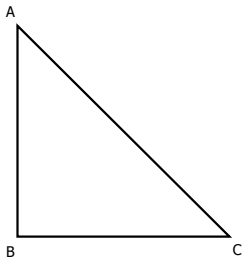

Figura 2.1: TIV

Por otro lado tenemos otra propuesta, realizada en el 2007 por Jonathan Ledlie, Paul Gardner, y Margo Seltzer, de Harvard, que es conocida como Vivaldi 2. Refiere al artículo científico *Network Coordinates in the Wild*[10]. Este trabajo refinó varios aspectos de la versión original de Vivaldi. En primer lugar, tomó una red más grande, mientras que el conjunto de nodos estudiados por Vivaldi contaba, en primer lugar con 192 hosts de PlanetLab, y posteriormente 1740 servidores DNS, para Vivaldi 2, se utilizó como entorno de pruebas la red de BitTorrent de Azureus, que contaba con más de un millón de nodos. En segundo lugar, en Vivaldi 2 se añade un parámetro de confiabilidad en el nodo, el cual determina cuan precisa será una medición de ese nodo, de manera que se podrá actualizar más correctamente el mapa de coordenadas. Por último, en esta nueva versión de Vivaldi se filtran las mediciones de manera que se eliminan los datos anómalos, que son calculados según la mediana.

Otro sistema que se conoce en este campo es Meridian[11]. Este sistema fue diseñado con el propósito de abordar diversos escenarios, la búsqueda del nodo más cercano o la elección de líder central. La característica distintiva de Meridian es su capacidad para formar una red organizada en modo de anillos que utiliza mediciones directas de RTT, lo que facilita la identificación de nodos de cada anillo de manera eficiente. Cada nodo mantiene una lista de anillos, lo que permite independencia de cada nodo, al haber eliminado un sistema centralizado.

Por otro lado está también Pharos[12]. El artículo comienza discutiendo la causa de la falta de precisión de otros sistemas de coordenadas en la red(NC). En primer lugar, diferencia dos tipos de sistemas de coordenadas. En primer lugar LBA (Landmark-based Algorithm) que mide retardos entre cada nodo y un punto de referencia, dividiendo todos los nodos en grupos, de manera que las mediciones entre nodos cercanos son muy acertadas, pero el error relativo en mediciones de mayor distancia aumenta considerablemente. En segundo lugar los SBA(Simulation-Based Algorithm), como Vivaldi, están basados en su mayoría en representaciones euclídeas. Como principales problemas de estos destacan la falta de precisión en las mediciones más pequeñas en contra de lo que podría imaginarse a primera vista. De manera que se propone Pharos para corregir este problema e intentar conseguir un sistema NC que funcione adecuadamente para calcular

retardos tanto grandes como pequeños. Pharos se trata de un sistema jerárquico descentralizado. La mejora de Pharos reside en organizar los nodos en clústeres de igual manera que en los LBA, pero sin eliminar las mediciones entre nodos alejados de diferentes clústeres para así tener también alta precisión en los enlaces más alejados. En comparación a Vivaldi, agrega carga de computación, pero mejora las prestaciones considerablemente.

Siguiendo con Tarantula[13], en este artículo se presenta una nueva propuesta de sistema de coordenadas, que introduce una forma innovadora de dividir el espacio. Tarantula clasifica las TIVs en tres categorías y demuestra que es capaz de manejar una parte considerablemente más grande de las TIVs existentes en comparación con los sistemas de dos capas. Además, este artículo presenta dos técnicas que hacen todavía más potente el sistema Tarantula: en primer lugar ajusta el tamaño del paso de actualización en el algoritmo Vivaldi utilizado en Tarantula para mejorar la predicción de enlaces cortos y proponer la Optimización Dinámica de Clusters para ajustar dinámicamente la agrupación de hosts. Los resultados experimentales muestran que Tarantula supera significativamente a Pharos y Vivaldi en términos de precisión en las estimaciones de latencia, lo que lo convierte en una opción destacada para aplicaciones como la selección de servidores y la búsqueda de desvíos en redes, lo que busca evitar las ya nombradas TIV.

Por otro lado está Phoenix[14], es un sistema de coordenadas de red que se ha destacado en el campo de la predicción de latencias en Internet. Su enfoque se basa en la idea de mapear las ubicaciones geográficas de los nodos de la red utilizando una representación en un espacio de coordenadas multidimensional. A través de mediciones de latencia entre pares de nodos, Phoenix calcula y ajusta continuamente las coordenadas de los nodos en este espacio multidimensional, lo que permite estimar las distancias entre nodos de manera eficiente y precisa. Uno de los aspectos destacados de Phoenix es su capacidad para lidiar con las Violaciones de la Desigualdad del Triángulo (TIVs), situaciones en las que las estimaciones de latencia no cumplen con la desigualdad del triángulo, lo que puede llevar a errores en la predicción de latencias. Phoenix ha demostrado ser eficaz en la predicción de latencias en una amplia gama de entornos de red, lo que lo convierte en una herramienta valiosa para la optimización de la entrega de contenido y la selección de servidores en Internet. Además de su capacidad para abordar TIVs, Phoenix se ha destacado por su escalabilidad y eficiencia. A medida que la red crece y se vuelve más compleja, Phoenix puede adaptarse para manejar grandes conjuntos de datos y mantener la precisión en sus estimaciones de latencia. Su enfoque descentralizado y la capacidad de trabajar en tiempo real lo hacen valioso en aplicaciones como la entrega de contenido y la planificación de la infraestructura de red, donde la precisión en la estimación de latencias es esencial para una operación eficiente y confiable. En resumen, Phoenix representa una contribución significativa al campo de las coordenadas de red y la predicción de latencias en Internet, abordando los desafíos de TIVs y brindando un enfoque escalable y preciso para estimar latencias en entornos de red dinámicos.

2.4 Sistema de mensajería

Cabe destacar que dentro del ámbito del proyecto hay que resolver la cuestión de la comunicación entre los diferentes proveedores de sistemas y el sistema federado del catálogo. Para ello en los siguientes párrafos se realizará un resumen acerca de distintos sistemas de comunicación que se podrían emplear para este propósito.

2.4.1. RabbitMQ

En primer lugar tenemos RabbitMQ, es un robusto y confiable sistema de mensajería de código abierto diseñado para facilitar la comunicación entre aplicaciones distribuidas, desarrollado en Erlang. Destaca dentro del mercado de la mensajería empresarial gracias a su escalabilidad, rendimiento y facilidad de uso. Utiliza el protocolo AMQP (Advanced Message Queuing Protocol), que define un estándar abierto para la mensajería avanzada, lo que permite a las aplicaciones comunicarse de manera eficiente y confiable, independientemente del lenguaje de programación en el que estén escritas, por lo que se podría decir que es bastante sencillo de integrar en una variopinta cantidad de ecosistemas. Otra de las características de esta tecnología es su flexibilidad en la gestión de colas de mensajes y enrutamiento. Las aplicaciones pueden publicar mensajes en colas y suscribirse a ellas para consumir los mensajes de manera eficiente. Esto facilita la implementación de patrones de comunicación como el *publish/subscribe* y el *request/reply*. Además, RabbitMQ es altamente escalable, lo que significa que puede manejar cargas de trabajo crecientes mediante la adición de nodos y la distribución de las colas de mensajes. Esto lo convierte en una opción ideal para aplicaciones que requieren una escalabilidad horizontal y un alto rendimiento en entornos empresariales. RabbitMQ se ha convertido en una pieza fundamental en la arquitectura de sistemas distribuidos y microservicios, facilitando la comunicación entre componentes y la construcción de sistemas altamente disponibles y tolerantes a fallos. RabbitMQ se destaca en el mundo de los sistemas de mensajería por colas debido a varias características que lo diferencian de otros sistemas similares. Una de las características más notables es su soporte para múltiples protocolos de mensajería, siendo el principal el protocolo AMQP (Advanced Message Queuing Protocol). AMQP es un protocolo de mensajería avanzada que define un estándar abierto para la comunicación entre sistemas distribuidos. Proporciona un marco para el intercambio eficiente de mensajes entre aplicaciones, independientemente del lenguaje de programación o la plataforma en la que estén implementadas. AMQP se utiliza en sistemas de mensajería, como RabbitMQ, para facilitar la comunicación asíncrona y la gestión de colas de mensajes. RabbitMQ también es compatible con otros protocolos como STOMP, MQTT, y HTTP. Por lo tanto, se puede afirmar que RabbitMQ es un sistema altamente versátil y que puede comunicarse con una amplia variedad de aplicaciones y servicios, lo que lo hace ideal para entornos heterogéneos. Además, RabbitMQ se destaca por su capacidad de gestión y monitoreo, proporcionando una interfaz de administración web intuitiva que permite supervisar el estado de las colas, gestionar usuarios y permisos, y realizar un seguimiento de las métricas de rendimiento. Esta característica facilita la administración y la resolución de problemas en entornos empresariales de alto tráfico. En resumen, la versatilidad, la flexibilidad en el enrutamiento y la robusta capacidad de administración hacen de RabbitMQ una elección sólida para una amplia gama de aplicaciones que requieren una mensajería confiable y escalable.

2.4.2. Apache Kafka

Otra opción de sistema de mensajería que necesita ser considerado para el proyecto es Apache Kafka. Este es un sistema de mensajería de código abierto basado en un enfoque del tipo publicación y suscripción que permite a las aplicaciones enviar y recibir flujos de datos en tiempo real de manera segura. Fue diseñada originalmente por LinkedIn. Lo que distingue a Kafka de otros sistemas de mensajería es su capacidad para manejar grandes volúmenes de datos así como su gran capacidad para la escalabilidad y la posibilidad de mantener persistencia de los mensajes.

Una de las características más notables de Kafka es su arquitectura distribuida. Como ya se ha mencionado, utiliza un modelo de publicación y suscripción en el que los productores envían mensajes a *topics* o temas, y los consumidores se suscriben a estos temas para recibir los datos. Kafka puede operar en clústers de servidores, lo que permite una escalabilidad horizontal para satisfacer las demandas de aplicaciones de alto rendimiento. Además, Kafka es capaz de almacenar los mensajes de manera duradera, lo que significa que los datos no se pierden incluso en caso de fallo del sistema, lo que lo convierte en una opción sólida para casos de uso críticos en tiempo real y análisis de datos a gran escala. En resumen, Apache Kafka es una herramienta esencial para la transmisión de datos y el procesamiento en tiempo real, y su arquitectura distribuida y su escalabilidad lo hacen adecuado para una amplia variedad de aplicaciones en diversas industrias. Apache Kafka se destaca entre los sistemas de mensajería por colas debido a diversas características. Una de las diferencias clave es su capacidad de procesamiento de transmisión de datos a gran escala y en tiempo real. A diferencia de los sistemas de mensajería tradicionales que se centran en el almacenamiento y entrega de mensajes, Kafka está diseñado para el procesamiento de flujos de datos, lo que lo hace especialmente adecuado para casos de uso de transmisión de datos, como análisis en tiempo real, seguimiento de eventos y procesamiento de registros. En resumen, se trata de una herramienta que despliega todo su potencial en entornos de comunicación entre aplicaciones dentro de infraestructuras distribuidas. Finalmente, se ha optado por el uso de esta tecnología en el proyecto por delante de otras debido a que cuenta una curva de aprendizaje menos pronunciada que otras alternativas así como por haber tratado con él en una asignatura del grado.

2.4.3. NATS

NATS es un sistema de mensajería desarrollado en GoLang, que se caracteriza por ser ligero y consiste en un esquema de código abierto que está diseñado para la comunicación entre microservicios y aplicaciones distribuidas. Este sistema es conocido por su simplicidad, velocidad y eficiencia, y apto para entornos que requieren baja latencia y alto rendimiento.

Una de las características principales de NATS es su arquitectura basada en un modelo de publicación y suscripción (pub/sub), similar a otros sistemas de mensajería como Apache Kafka, aunque también admite patrones como *request/reply* (req/rep) o los *queue groups*. En el modelo pub/sub, los productores (publicadores) envían mensajes a un tema (topic), y los consumidores (suscriptores) reciben esos mensajes si están suscritos a ese tema. Esta arquitectura permite una comunicación eficiente y desacoplada entre diferentes componentes de un sistema distribuido. Además, NATS es altamente escalable, ya que se puede añadir nodos adicionales al clúster para distribuir la carga de trabajo de manera eficiente. Otra ventaja de NATS es su capacidad para mantener conexiones persistentes y ligeras, lo que reduce la sobrecarga de recursos y mejora el rendimiento general del sistema. En resumen, NATS es una opción adecuada para aplicaciones que requieren una comunicación rápida y fiable.

2.5 Crítica al estado del arte

Durante la búsqueda de información relacionada hemos encontrado proyectos con objetivos similares, pero ninguno acaba de gestionar de manera precisa y eficiente la gestión del catálogo. Con este estudio se ha obtenido una base suficiente para el desarrollo de la ontología, que abordaremos detenidamente en capítulos posteriores. También he-

mos observado que los sistemas de coordenadas de red existentes se basan en sistemas *peer-to-peer* y necesitan esos algoritmos de sistemas P2P para asegurarse un intercambio frecuente y continuo de mensajes y en un entorno federado de sistemas de IA ese entorno no se dará y debido a ello resultará muy difícil o incluso imposible utilizar un sistema de coordenadas de red en el prototipo que se va a desarrollar.

Tras considerar las diferentes opciones de sistemas de mensajería, incluyendo RabbitMQ y NATS, se ha decidido utilizar Apache Kafka para este proyecto. Esta elección se basa en varias razones fundamentales:

- Capacidad de Manejo de Datos a Gran Escala: Apache Kafka está diseñado para manejar grandes volúmenes de datos y flujos de datos en tiempo real. Su capacidad para procesar y almacenar grandes cantidades de datos de manera eficiente es interesante para un sistema de mensajería como el del catálogo.
- Persistencia de Mensajes: Kafka proporciona una robusta persistencia de mensajes, garantizando que los datos no se pierdan incluso en caso de fallos del sistema. Esta característica es crucial para aplicaciones que requieren alta disponibilidad y confiabilidad en la entrega de mensajes.
- Arquitectura Distribuida y Escalabilidad: La arquitectura distribuida de Kafka permite una escalabilidad horizontal, lo que significa que puede crecer fácilmente añadiendo más nodos al clúster. Esto asegura que el sistema pueda manejar un aumento en la carga de trabajo sin degradar el rendimiento.
- Ecosistema Maduro y Herramientas: Kafka cuenta con un ecosistema amplio y maduro, incluyendo diversas herramientas y bibliotecas que facilitan su integración y uso en diferentes entornos. La comunidad activa y el soporte continuo también aseguran que Kafka evolucione y mejore constantemente.
- Flexibilidad y Versatilidad: Kafka es extremadamente versátil y puede ser utilizado en una variedad de casos de uso, desde la transmisión de datos en tiempo real hasta el procesamiento de eventos y análisis de datos a gran escala. Esta flexibilidad es un valor añadido significativo para el proyecto.

Apache Kafka ha sido elegido como el sistema de mensajería para el proyecto debido a su capacidad para manejar grandes volúmenes de datos, su robusta persistencia de mensajes, su escalabilidad y su ecosistema bien desarrollado. Estas características hacen de Kafka la opción más adecuada para satisfacer las necesidades actuales y futuras del sistema de mensajería del catálogo.

2.6 Propuesta

En conclusión, la finalidad de este trabajo es la creación de una federación de recursos *hardware* que busca introducirse en el marco de diferentes proyectos de Inteligencia Artificial de la Unión Europea, lo que resulta un enfoque novedoso y propone retos tecnológicos para la gestión de estos servicios.

CAPÍTULO 3
Análisis del problema

Como ya se ha mencionado, es muy diferente el mercado de las *cloud* americano al europeo, porque mientras que en el primero podemos encontrar una gran empresa, que ofrezca servicios de nube como por ejemplo la de Amazon, la de Microsoft o la de Google, en el europeo carecemos de una gran organización que cumpla este cometido. Por lo que, al no haber una gran infraestructura dispuesta para ello, el mercado está predispuesto a que entidades más pequeñas o particulares que dispongan de recursos de este tipo y quieran comercializarlos puedan publicarlos. Por lo que es un punto a tener en cuenta: es un importante paso adelante el unificar todos estos sistemas de manera que se ubiquen todos en una misma plataforma. De esta manera, un usuario o cliente del catálogo se vería beneficiado de esta medida al poder consultar todo lo que requiere en una misma plataforma. Teniendo una visión más a largo plazo deberíamos remarcar que el proyecto podría orientarse hacia la gestión de federaciones para aplicaciones de IA (Inteligencia Artificial) europeas como las mencionadas en el capítulo anterior.

3.1 Especificación de requisitos

Respecto a los requisitos los tenemos de dos tipos, los requisitos funcionales y los requisitos no funcionales. A continuación vamos a explicar cada uno de los tipos.

En primer lugar, los requisitos funcionales describen lo que el sistema debe hacer. Estos requisitos definen las funciones y características que el sistema debe proporcionar a los usuarios. Son específicos y se pueden traducir directamente en funcionalidades del *software*. Los requisitos funcionales son esenciales para garantizar que el sistema cumple con las necesidades y expectativas de los usuarios finales. Por otro lado, los requisitos no funcionales describen cómo el sistema debe comportarse. Estos requisitos se enfocan en atributos de calidad del sistema, como la usabilidad, el rendimiento o la escalabilidad. Aunque no se refieren a funciones específicas del sistema, son igualmente importantes porque afectan la experiencia del usuario y la eficiencia del sistema.

A continuación, se presentan los requisitos funcionales y no funcionales que podrían ser relevantes para el desarrollo de un catálogo de recursos *hardware* para la federación de recursos de computación:

Requisitos Funcionales:

1. La interfaz del catálogo debe permitir la búsqueda y filtrado de recursos de *hardware* según diferentes criterios (procesador, memoria, capacidad de almacenamiento, etc.).

2. Los recursos de *hardware* deben poder ser reservados y liberados por los usuarios a través del catálogo.

3. Los usuarios deben poder visualizar información detallada sobre cada recurso *hardware*, como su disponibilidad o estado.

4. Los administradores del sistema deben poder agregar, editar y eliminar recursos de *hardware* del catálogo.

5. Se deberá poder añadir nuevos sistemas al catálogo mediante la ejecución de un *script*.

Requisitos No Funcionales:

1. El sistema debe ser altamente disponible y escalable para soportar usuarios concurrentes, así como un gran conjunto de recursos de *hardware*.

2. La seguridad de los datos debe ser una prioridad, incluyendo la autenticación y autorización de usuarios y la protección contra ataques maliciosos.

3. La interfaz de usuario del catálogo debe ser intuitiva y fácil de usar para fomentar la adopción del sistema por parte de los usuarios.

4. El catálogo debe ser capaz de manejar múltiples tipos de recursos de *hardware* y debe ser compatible con diferentes plataformas y sistemas operativos.

5. El rendimiento del catálogo debe ser óptimo, minimizando los tiempos de búsqueda y reserva de recursos de *hardware*.

3.2 Modelado conceptual

Contamos con diferentes componentes del sistema. Por un lado tenemos los *providers*, esta parte del sistema la conforman los propietarios de los servicios, los cuales son diferentes organizaciones que cuentan con clústeres de computación de distintas características y capacidades. Asimismo tenemos el propio catálogo de recursos, en el cual se almacena registro de todos los sistemas que hay incluidos en la federación, así como su disponibilidad. Por último se encuentra el *frontend*, que se trata de una aplicación web, esta estará disponible para su consulta por parte de los clientes.

En el catálogo habrá dos roles diferenciados, cada uno de los cuales tendrá una manera de interactuar con el catálogo y en la figura 3.1 se describen las acciones realizables por cada uno de ellos. Por una parte, el rol de usuario, que podrá visualizar el catálogo junto con todas las máquinas que lo incluyen, y podrá realizar acciones como listar las máquinas y sus características sin ninguna limitación. También tendremos el rol de administrador del sistema, el cual tendrá funciones para modificar las máquinas incorporadas al catálogo, de manera que tendrá tanto permisos de lectura (como los miembros del rol usuario) como de escritura, pudiendo añadir, modificar o eliminar cualquier máquina perteneciente al catálogo.

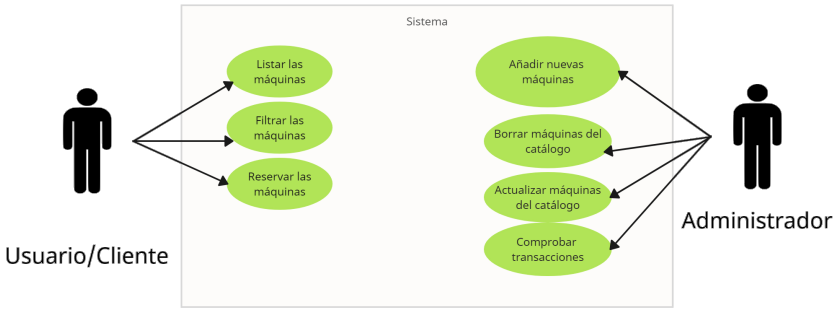

Figura 3.1: Modelado conceptual

3.3 Análisis de seguridad

Respecto al análisis de seguridad, lo podemos separar en dos campos claramente diferenciados: el cumplimiento de los plazos fijados, y la protección de los datos del proyecto, de manera que cada colaborador tan sólo pueda acceder a la parte que le corresponda.

Es importante tener en cuenta la seguridad de las conexiones de los servicios de Kafka, así como la manera de persistir el catálogo. También es importante considerar los requisitos de conexión como productor al entorno de Apache Kafka.

3.4 Solución propuesta

Para resolver esta problemática se establecerá una estructura de nodos centralizada. Este será un catálogo, que agrupará las características y parámetros de los diferentes recursos que se quieren comercializar. Para la composición de este catálogo se hará uso de Apache Kafka, una tecnología de tipo *publish-subscribe* altamente escalable, así como tolerante a fallos y de baja latencia [15]. Mediante el uso de Kafka se comunicarán los distintos nodos de cómputo con el catálogo. Esta comunicación podrá ser de dos tipos. En primer lugar, para publicar nuevos recursos que se quieran añadir al catálogo. En segundo lugar, para recibir información de los recursos publicados en el sistema en el momento que se solicite la información almacenada en el catálogo.

Dada la cantidad de sistemas diferentes que se podrían añadir al catálogo, será necesario desarrollar una ontología que especifique los parámetros que serán considerados como los más relevantes. Estos parámetros relevantes serán aquellos que serán publicados en el catálogo. De esta manera, con la ontología, se busca unificar mediante los mismos atributos todos los sistemas para así poder compararlos de una manera visual y sencilla.

Se utilizará un *script* con el objetivo recoger todas aquellas propiedades descritas en la ontología, el cual se habrá de ejecutar en cada una de las máquinas que se desee añadir al catálogo.

Para almacenar, gestionar y poder publicar el catálogo se va a utilizar una aplicación Java, que será la encargada de gestionar toda la lógica que se engloba dentro del catálogo, de manera que el *frontend* pueda solicitar los datos a esta aplicación para así poder exponerlos a los usuarios.

Por otro lado, para la visualización del catálogo se utilizará una interfaz web, lo que es un enfoque idóneo para el problema. Por la parte del usuario, se trata de una solu-

ción que destaca por su facilidad de uso, ya que no se requiere la instalación de ninguna aplicación, y se adquiere la seguridad incorporada en el protocolo *https* con respecto al cifrado de los datos. Por la parte del programador, es una opción que se caracteriza por ser fácilmente desplegable y actualizable, igualmente, no requiere un alto mantenimiento ya que son tecnologías asentadas y con poca evolución. Asimismo se trata de una tecnología conocida y que ha sido utilizada a lo largo del grado, en asignaturas como por ejemplo DEW.

Respecto a la implantación del proyecto, el catálogo y la aplicación se van a desplegar sobre contenedores Docker, mientras que para la interfaz se va a hacer uso del servidor web de páginas personales que ofrece la Universidad. Para la validación del proyecto se harán pruebas de incorporación de diferentes máquinas al catálogo, con su consiguiente visualización desde la interfaz web.

3.5 Plan de trabajo

A la hora de comenzar el proyecto, encontramos distintas tareas a realizar. A lo largo de esta sección vamos a introducir una figura con un diagrama de Gantt. Las unidades de este diagrama son las horas de trabajo. En este diagrama se expondrán las diferentes tareas que conformarán el proyecto así como una pequeña descripción de cada una de ellas, su duración y las dependencias que puedan existir entre ellas.

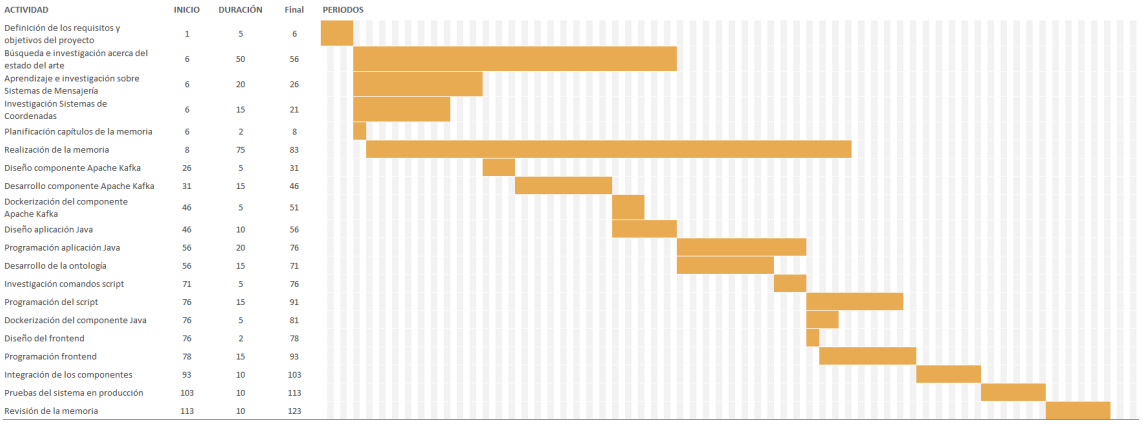

Figura 3.2: Diagrama de Gantt

La cantidad de horas totales que se prevé es de 309 horas según lo estipulado en el diagrama de Gantt de la figura de encima.

1. **Definición de los requisitos y objetivos del proyecto** (5 horas): En primer lugar necesitaremos definir los objetivos del proyecto, así como una propuesta que defina los requisitos del mismo. Las tareas 2, 3, 4 y 5 tienen una dependencia de fin a inicio con esta tarea descrita.

2. **Búsqueda e investigación acerca del estado del arte** (50 horas): Una vez definidos los objetivos y requisitos del trabajo, tendremos que documentarnos acerca de la tecnología existente en el campo y cómo se están proponiendo las soluciones a problemas similares, además de investigar acerca de las características de las máquinas más útiles para poder elegir entre ellas en el catálogo. Esta tarea tendrá una larga duración, ya que es de alta importancia para el desarrollo, a parte de para formarse en el campo de estudio.

3. **Aprendizaje e investigación sobre Sistemas de Mensajería** (20 horas): Haremos una investigación acerca de los diferentes sistemas de mensajería, de esta manera se podrá tomar la mejor decisión posible al respecto. Posteriormente necesitaremos formarnos en la opción elegida.

4. **Investigación Sistemas de Coordenadas** (15 horas): Se investigará sobre los diferentes sistemas de coordenadas para estudiar si es viable su incorporación a la capa de visualización del proyecto.

5. **Planificación capítulos de la memoria** (2 horas): Consiste en la organización de los distintos capítulos que formarán parte de la memoria, junto con un breve resumen de lo que debería aparecer en cada uno.

6. **Realización de la memoria** (75 horas): Esta será la tarea más larga del proyecto y cuenta con una dependencia fin a fin con todo el resto de tareas de desarrollo, ya que habrá que completar todas esas tareas antes de acabar la memoria. Esta tarea tiene una dependencia fin a inicio con la tarea 5.

7. **Diseño componente Apache Kafka** (5 horas): Esta tarea tiene una dependencia fin a inicio con la tarea 3. En esta tarea se decidirá la configuración utilizada por el componente de mensajería y se definirán los *topics* utilizados.

8. **Desarrollo componente Apache Kafka** (15 horas): Esta tarea cuenta con una dependencia de fin a inicio con la tarea 7 y consiste en la construcción de esta parte de la solución.

9. **Dockerización del componente Apache Kafka** (5 horas): Existe una dependencia de fin a inicio entre esta tarea y la 8. En concreto esta tarea consiste en hacer uso del sistema de mensajería pero utilizando Docker.

10. **Diseño aplicación Java** (10 horas): Consiste en el diseño conceptual de las clases y la funcionalidad que deberá cumplir dicha aplicación. Tiene dependencia de fin a inicio con el desarrollo del apartado de Kafka.

11. **Programación del componente Java** (20 horas): Proceso de desarrollar la solución Java, este proceso tiene dependencia de fin a inicio con el diseño de la aplicación.

12. **Desarrollo de la ontología** (15 horas): Es la realización de un esquema en el que se recogen las características más importantes de las máquinas. Tiene una dependencia de fin a inicio con la segunda tarea.

13. **Investigación comandos para el script** (5 horas): Se investigará acerca de los posibles comandos que se pueden utilizar para la extracción de las características descritas en la ontología. Por ello, tiene una dependencia de fin a inicio con la tarea del desarrollo de la ontología.

14. **Programación del script** (15 horas): Tras conseguir los comandos que satisfacen los contenidos requeridos para completar el esquema propuesto en la ontología se programa el *script*. Esta tarea tiene una dependencia con la tarea 13.

15. **Dockerización del componente Java** (5 horas): Al finalizar el desarrollo de la aplicación Java se dockeriza para facilitar su despliegue en producción así como su escalabilidad. Esta tarea depende de la tarea 11.

16. **Diseño del frontend** (2 horas). En esta tarea se diseñan las diferentes partes del *frontend*. Esta tarea depende tanto de la tarea 12 como de la tarea 11.

17. **Programación del frontend** (15 horas): Tras diseñar la interfaz se pasará a su implementación, por lo tanto, esta tarea tendrá que comenzar después de que finalice el diseño.

18. **Integración de los componentes del proyecto** (10 horas): Consiste en unir todos los elementos de la solución de manera que se ofrezca como un sistema único y homogéneo y no como un conjunto de piezas por separado. Esta tarea depende de todas las que incluyen el desarrollo de algún componente.

19. **Pruebas del sistema en fase de producción** (10 horas): En esta tarea se probarán los diferentes casos de uso propuestos para comprobar la eficacia de la solución en todas las coyunturas. Esta tarea depende de la tarea 18.

20. **Revisión de la memoria** (10 horas): Tras completar todo el desarrollo y la memoria se deberá revisar el contenido de esta. Esta será la última tarea del proyecto.

3.6 Presupuesto

A partir del plan de trabajo descrito anteriormente se deriva el presupuesto. Estimando un precio de 11 € por hora[16], y la cantidad de 309 horas de trabajo se estima un coste del proyecto de 3399€ por la compleción del trabajo de fin de grado.

El sistema resultante tendría unos costes recurrentes una vez puesto en marcha, ya que habría que contabilizar los costes del *hardware* y/o *software*, ya sean compradas o alquiladas, así como la electricidad que consumen esos sistemas así como sus administradores, en caso que se haya optado por comprarlos. También sería interesante tener en cuenta el salario de los administradores del catálogo. Sin embargo, estos costes recurrentes no son objeto de análisis en el campo del proyecto, puesto que las tareas desarrolladas dentro de las prácticas de empresa en las cuales se adscribe el proyecto se limitaban al análisis, diseño y desarrollo de un primer prototipo. De este modo, se deja el despliegue definitivo y la puesta en producción como tareas a realizar por otros investigadores de la empresa o por personal del resto de organizaciones que participan en el proyecto del catálogo. Dentro de este contexto hemos decidido utilizar Docker en la solución para facilitar así las subsiguientes tareas, pues los contenedores simplificarán en gran medida el despliegue definitivo, ya sea en la nube o en centros de datos de las organizaciones participantes en el catálogo y por tanto en el proyecto de investigación.

CAPÍTULO 4
Diseño de la solución

Durante este capítulo se especificarán todas las etapas por las que se ha pasado a lo largo del diseño de este proyecto.

Contamos con diferentes componentes del sistema. Por un lado tenemos los *providers*, esta parte del sistema la conforman los propietarios de los servicios. Por otro lado tenemos el propio catálogo de recursos, en el cual se almacena registro de todos los sistemas que hay incluidos en la federación, así como su disponibilidad. Por último se encuentra el *frontend*, que se trata de una aplicación web, esta estará disponible para su consulta por parte de los clientes.

4.1 Arquitectura del sistema

La arquitectura del proyecto se puede dividir en varias partes. Como se muestra en la figura 4.1.

En primer lugar tenemos los proveedores, que a partir del *script* mencionado consiguen listar los detalles de cada una de sus máquinas, de manera que una vez conocen toda la información, le aportan un identificador único para cada proveedor. Una vez con el identificador, se envían esos datos al clúster mediante un publicador (utilizando Apache Kafka). Gracias al identificador, el clúster es capaz de conocer la procedencia de cada máquina para así poder agruparla posteriormente en subgrupos en función de su proveedor. En este punto del proyecto se podrían tomar varias soluciones para resolver el tema de cómo se puede persistir el catálogo. En primer lugar podríamos hacer uso de la propia persistencia de Kafka, ya que hay diversos métodos para asegurar la persistencia de los mensajes. Uno de esos métodos consiste en configurar un atributo de Apache Kafka para que los mensajes sean persistentes.

Por otra parte una herramienta que gestione los mensajes enviados y recibidos por el sistema de mensajería y actúe como base de datos, en la cual, a medida que el nodo del catálogo va recibiendo mensajes de los diferentes *providers*, se va incluyendo esa nueva información o modificando la existente en caso de cambios en el estado de las máquinas en esta. Esta solución es a priori más conveniente en el ámbito de la gestión de los datos En el momento que el clúster contiene esa información, se almacena en el sistema gestor para que, cada vez que algún cliente solicite el catálogo, se extraiga esta información de la base de datos. Para exponer al *frontend* los datos se puede dejar abierto un *endpoint* que sea una API REST.

La manera de exponer el catálogo a los clientes es mediante una aplicación web, que conecta con el sistema gestor, mediante esta web el cliente puede consultar los datos actualizados acerca de los sistemas y su disponibilidad. Dentro de esta aplicación web se

podrá visualizar la información en forma de tabla, en la cual se podrán filtrar las máquinas según sus características.

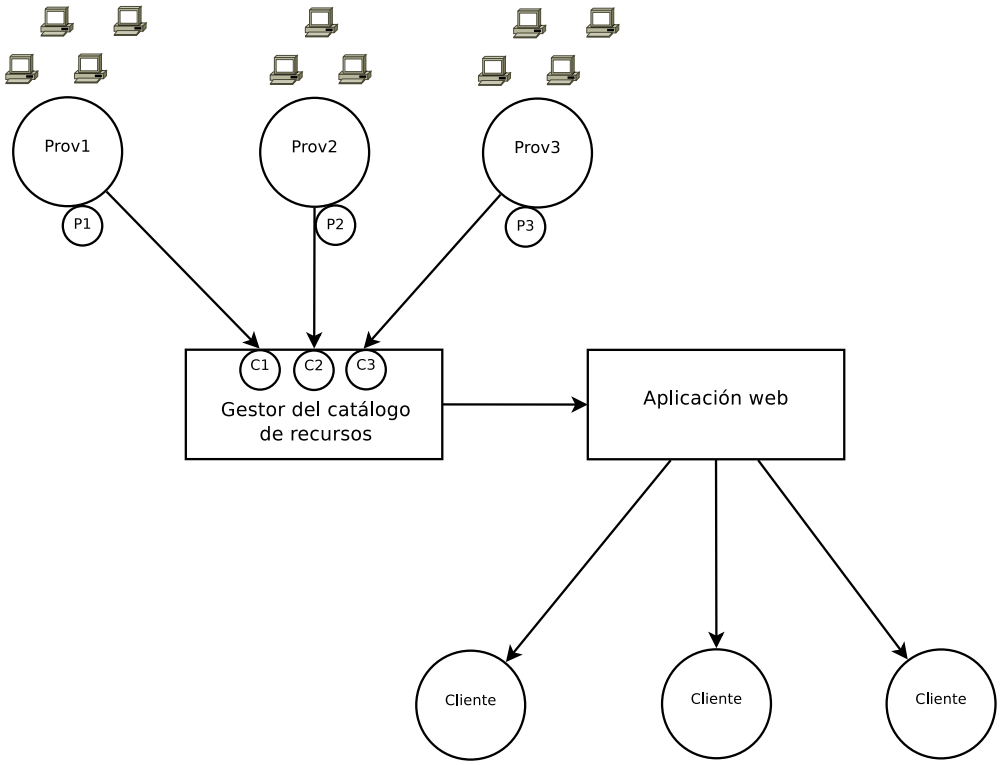

Figura 4.1: Arquitectura del sistema

De la figura 4.1, Prov1, Prov2 y Prov3 representan a los diferentes *providers* que publican sus máquinas en la federación. ¿Cómo las publican? Estos sistemas son publicados utilizando la herramienta Apache Kafka, de manera que los *providers* actúan como publicadores de sistemas al catálogo. Por otro lado el catálogo está conformado por diferentes réplicas del broker, para así poder escalar el sistema en caso de ser necesario, lo que aumentaría su disponibilidad y su tolerancia a fallos. Al realizar un despliegue con Docker, en caso de ser necesario, se pueden aumentar fácilmente la cantidad de réplicas. Por su parte, el *frontend* se trata de una aplicación web que extrae los datos del gestor del catálogo de recursos.

4.2 Diseño detallado

Comenzaremos esta sección de diseño comentando y especificando cada una de las diferentes partes que se describen en dicha figura 4.1 de la arquitectura.

En primer lugar, comenzaremos por la parte de los *providers*. Se definen como tal los miembros de la federación que sean poseedores de recursos *hardware* de cómputo.

Como punto de partida, es necesario detectar, de todas las características que se pueden extraer de un dispositivo *hardware*, cuáles deberían ser las más relevantes a la hora de decidirse entre un sistema u otro en un despliegue en la nube. Para lo cual se ha realizado un estudio minucioso acerca de cuáles de estas características serán mostradas al usuario para que pueda tomar su decisión entre todas las opciones. Una vez realizada la recopilación de estos atributos, procederemos a introducirlos y organizarlos. Para ello se ha requerido del uso de una ontología. En ella vemos claramente diferenciados dos aspectos que se habrán de recoger de cada máquina. Por un lado, el *hardware*, y por otro

lado el *software*. Una vez realizada esa diferenciación, ya podemos profundizar en cada uno de ellos. La parte del *hardware* está conformado por cinco subcategorías.La primera de ellas es el sistema, que se centra en los detalles de este, estos son el fabricante y el modelo, la segunda es el disco, luego la cpu, la memoria ram y por útlimo la red, se trata de un *array* que contiene las diferentes interfaces de red del sistema. Por su parte, la parte *software* consiste en el sistema operativo de la máquina así como su versión del kernel.

En el ámbito del catálogo y su visualización encontramos dos maneras de interactuar con el mismo, los denominaremos roles. En primer lugar, el usuario base puede acceder al dominio en el que está publicado el catálogo para consultar todos los sistemas dados de alta en la federación. También tendrá permisos para filtrar el catálogo según alguno de los diferentes parámetros de la ontología o un conjunto de ellos. Por otro lado, los usuarios autenticados serán capaces de reservar máquinas desde la interfaz web.

El segundo rol es el de administrador del sistema, este otro tipo de usuario será capaz de realizar todas las acciones de visualización de los usuarios base. Además podrá añadir nuevas máquinas al catálogo de manera manual, así como borrarlas y actualizar las características de ellas. De esta manera los administradores pueden realizar todas las operaciones CRUD(*Create, Read, Update and Delete*), que son las cuatro operaciones básicas que se pueden realizar sobre elementos guardados sobre almacenamiento persistente. Asimismo podrán comprobar el registro de las operaciones del catálogo, para así poder detectar casos de uso fraudulento del mismo igualmente, podrán monitorizar algunas operaciones del sistema.

4.3 Tecnología utilizada

A continuación se expondrá cada una de las tecnologías utilizadas para el desarrollo del proyecto, contextualizando en cada caso el uso que se hace de ella y justificando su elección frente a otras alternativas.

4.3.1. Bash

Bash ha sido la tecnología elegida para escribir el *script* encargado de extraer información acerca de los diferentes servicios *hardware*. La selección de *Bash* se debe a su gran compatibilidad con entornos Unix/Linux y su robustez en la ejecución de *scripts* complejos. Aunque existen alternativas como *sh*, *Bash* ofrece características avanzadas que resultan fundamentales para este proyecto. Por ejemplo, el uso de dobles corchetes "([[" y "]])" permite una mayor flexibilidad en las condiciones lógicas, y los *arrays* facilitan la manipulación y el almacenamiento de datos de manera eficiente. Estas características son esenciales para la extracción detallada y precisa de la información del *hardware*, ya que permiten manejar operaciones complejas y condiciones específicas que otros *shells* como *sh* no soportan. Además, *Bash* cuenta con una comunidad activa, lo que asegura acceso a una vasta cantidad de recursos y documentación. Esta portabilidad y el soporte comunitario hacen de *Bash* una opción confiable para el desarrollo de *scripts* en proyectos de esta naturaleza. La elección de *Bash* también se justifica por su familiaridad y uso extendido entre los administradores de sistemas y desarrolladores, lo que reduce la curva de aprendizaje y facilita la integración y mantenimiento del código en el entorno de producción. Su robustez y eficiencia en la gestión de tareas de automatización hacen que *Bash* sea la herramienta ideal para la extracción de datos en este proyecto.

4.3.2. JSON Schema

JSON Schema es una herramienta habitualmente utilizada para definir y validar la estructura de documentos JSON. Actúa como un contrato que especifica el formato que los datos deben cumplir, incluyendo los tipos de datos, las dependencias, los valores por defecto e incluso las restricciones de longitud. Al utilizar JSON Schema, se puede asegurar que los datos intercambiados entre sistemas sean consistentes y conformes a un formato predefinido, lo que facilita la integración y reduce la probabilidad de los errores relacionados con el formato de los datos.

Se ha utilizado esta tecnología para el formato de los datos de la ontología. Hay una gran cantidad de lenguajes disponibles a la hora de desarrollar una ontología, como por ejemplo RDFS, OWL o DL. Se ha elegido JSON Schema por varios motivos. En primer lugar, por la orientación de la tecnología a la validación de la estructura de los datos, es decir, se centra en organizar datos, frente a OWL o RDFS, en los que también se busca hallar inferencias de nuevos hechos tomando como base las ontologías definidas, por lo que no se han considerado necesarias esas funcionalidades para el desarrollo de la ontología. En última instancia, dada la popularidad y familiaridad con el formato JSON, cosa que ha reducido la curva de aprendizaje y, por lo tanto, facilitado el desarrollo de esta ontología.

En el proyecto, JSON Schema ha sido empleado para especificar la estructura de los mensajes que se intercambian entre los diferentes componentes del sistema, en concreto los mensajes entre los proveedores y el sistema gestor del catálogo. Esto ha permitido establecer un estándar claro para los datos que deben ser enviados y recibidos, facilitando la implementación y el mantenimiento del sistema. Al validar los datos entrantes con JSON Schema, se asegura que solo los mensajes correctamente estructurados sean procesados, lo que ayuda a prevenir errores y fallos. Además, al documentar las estructuras de datos mediante JSON Schema, se mejora la comprensión del sistema tanto para los desarrolladores actuales como para aquellos que puedan unirse en el futuro. Esta práctica no solo incrementa la robustez y la fiabilidad del sistema, sino que también agiliza el desarrollo y la colaboración entre los equipos.

4.3.3. Apache Kafka

Se hace uso de este sistema de mensajería, como ya se ha descrito en la sección 2.4.2 que se utilizará para la publicación y/o actualización del estado de las máquinas, así como para gestionar la API REST con la cual se pueden consultar todos los sistemas incorporados en la federación.

En el contexto de este proyecto, Apache Kafka ha sido elegido como el sistema de mensajería para gestionar la comunicación entre los diferentes proveedores y el sistema gestor del catálogo. Su capacidad para manejar grandes volúmenes de datos y su arquitectura escalable son características que permiten que el sistema crezca sin comprometer en gran manera el rendimiento del sistema. Kafka asegura que los mensajes, son entregados y procesados en tiempo real. Esto es relativamente importante en nuestro contexto para mantener la integridad y la actualidad del catálogo de recursos. Además, la durabilidad y la replicación de Kafka garantizan que los mensajes no se pierdan, incluso en situaciones de fallo del sistema, proporcionando una alta confiabilidad y disponibilidad. La utilización de Kafka facilita también la implementación de análisis de datos en tiempo real, permitiendo que el sistema responda rápidamente a las condiciones cambiantes y mejore continuamente su servicio.

4.3.4. Docker

Se ha dockerizado el despliegue del Apache Kafka para así poder admitir múltiples réplicas del broker gestor del catálogo y así aumentar la disponibilidad y la tolerancia a fallos, al reducir la cantidad de fallos ocasionados porque pueda fallar el broker. Principalmente aporta al proyecto portabilidad, por su independencia sobre el sistema operativo sobre el que se ejecuta. Igualmente, permite de manera muy sencilla escalar el servicio desplegado para replicar componentes del tipo que se requiera.

Docker es una plataforma de contenedores que permite a los desarrolladores empaquetar aplicaciones y sus respectivas dependencias en contenedores ligeros y portátiles. Docker Compose, una herramienta de Docker, facilita la definición y ejecución de aplicaciones multicontenedor. Con Docker Compose, se puede utilizar un archivo YAML para configurar los servicios de una aplicación, lo que simplifica el proceso de configuración y despliegue. Esta herramienta permite iniciar, detener y gestionar todos los contenedores de una aplicación con un solo comando, facilitando la orquestación de entornos complejos y asegurando que todas las partes del sistema funcionen correctamente en conjunto.

En el contexto de este proyecto, Docker Compose se utiliza para desplegar los diferentes componentes del sistema de manera eficiente y cómoda. Al definir todos los servicios necesarios en un archivo "docker-compose.yml", se puede garantizar que cada componente tenga las configuraciones y dependencias correctas. Esto no solo acelera el proceso de desarrollo y pruebas, sino que también simplifica el despliegue en diferentes entornos, asegurando que el sistema se comporte de la misma manera en cada uno de ellos. Docker Compose permite a los desarrolladores concentrarse en el desarrollo de funcionalidades sin preocuparse por los problemas de configuración y compatibilidad. Por otro lado, permite escalar y desescalar el servicio, en función de la carga de trabajo que soporta, para ello se pueden utilizar mediciones del tiempo de respuesta de diferentes *endpoints* clave de la aplicación, aunque esta funcionalidad, para el ámbito del proyecto está limitada ya que no está automatizada, por lo que requiere de una persona consultando los datos para decidir sobre el escalado de cada uno de los componentes.

4.3.5. Maven y Spring Boot

Spring Boot y Maven son dos tecnologías fundamentales que han desempeñado un papel crucial en el desarrollo de este proyecto, proporcionando una base sólida y escalable para la construcción de aplicaciones robustas y eficientes.

Spring Boot es un *framework open source* de Java basado en Spring que simplifica el desarrollo de aplicaciones Java, permitiendo la creación de aplicaciones independientes y de producción listas para ejecutarse con una configuración mínima. En este proyecto, Spring Boot ha sido utilizado para desarrollar los servicios *backend*, aprovechando su capacidad para crear aplicaciones escalables rápidamente. Con su enfoque en la convención sobre la configuración, Spring Boot ha reducido significativamente la cantidad de configuración y repetición de fragmentos del código, permitiendo centrarse en la lógica de negocio. Además, las características integradas de Spring Boot, como su servidor web embebido y su soporte para diferentes perfiles de configuración, han facilitado el desarrollo, las pruebas y el despliegue de los servicios *backend*, asegurando que el proyecto pueda evolucionar rápidamente y adaptarse a nuevas necesidades sin comprometer la calidad del código. En el contexto del proyecto, Spring Boot ha permitido crear una aplicación *backend* sólida y escalable que se comunica eficazmente con Apache Kafka para la gestión de mensajes y su posterior envío al *frontend*.

Maven es una herramienta de gestión y comprensión de proyectos, utilizada principalmente para proyectos Java, que proporciona un marco estandarizado para la construcción de proyectos, la gestión de dependencias y la ejecución de pruebas. En el contexto de este proyecto, Maven ha sido fundamental para gestionar las dependencias, asegurando que todas las bibliotecas y *frameworks* necesarios estén disponibles y actualizados en el entorno de desarrollo. Al definir todas las dependencias en un archivo "pom.xml", Maven permite agregar, actualizar y gestionar fácilmente las bibliotecas sin preocuparse por conflictos de versiones o la necesidad de manejar manualmente los archivos JAR. Esto ha facilitado la integración de diversas tecnologías, como las dependencias de Apache Kafka, y ha simplificado la integración continua y la implementación automatizada, permitiendo a los desarrolladores compilar, probar y desplegar el código de manera eficiente. La combinación de Spring Boot y Maven ha proporcionado una solución integral que ha mejorado la productividad del equipo de desarrollo y ha garantizado la entrega de un producto final de alta calidad.

4.3.6. Visual Studio Code

Se ha optado por utilizar esta IDE (entorno de desarrollo integrado) por estar familiarizado con ella desde hace tiempo. Además de que se trata de una interfaz amigable, aparte de ligera y rápida.

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft que se ha convertido en una herramienta muy popular entre desarrolladores de todo el mundo gracias a su flexibilidad, extensibilidad y alto rendimiento. VS Code es un editor ligero pero potente que ofrece características avanzadas como depuración, resaltado de sintaxis, autocompletado inteligente y un terminal integrado. Su arquitectura basada en extensiones permite a los desarrolladores personalizar y mejorar el editor con una amplia gama de plugins y extensiones, que cubren prácticamente cualquier lenguaje de programación y marco de trabajo existente. A continuación se comentarán algunas de las utilizadas, en primer lugar una extensión para mejorar la organización de los archivos CSS, la extensión Maven for Java que, entre otras cosas se puede utilizar para editar el archivo POM de una manera más cómoda. También tiene extensiones interesantes como la de Docker y Docker Compose, o una para procesar yaml de manera más cómoda.

En el contexto de este proyecto, VS Code ha sido fundamental para gestionar y desarrollar el código de manera eficiente. Con sus capacidades de integración de Git, se ha podido realizar un seguimiento de los cambios. Además, la posibilidad de añadir extensiones específicas ha permitido a los desarrolladores trabajar con tecnologías como Docker, CSS y los lenguajes utilizados en el proyecto. Las características de depuración han sido esenciales para identificar y solucionar errores rápidamente, mientras que el terminal integrado ha permitido ejecutar comandos y *scripts* directamente desde el editor, agilizando el flujo de trabajo. En resumen, VS Code ha proporcionado un entorno de desarrollo integrado (IDE) robusto y flexible que ha optimizado significativamente la productividad y la calidad del desarrollo del proyecto.

CAPÍTULO 5
Desarrollo de la solución

En este capítulo se definirá y explicará el modo de resolver los retos que propone cada una de las partes que compone este trabajo.

5.1 Ontología

Para el desarrollo de la ontología, es necesario realizar un análisis acerca de esos parámetros que son más relevantes que el resto para así exponerlos en el catálogo de la federación y facilitar de esta manera la elección entre las máquinas. Mediante el uso de la herramienta *Google Scholar* se han encontrado varios artículos que destacan diferentes puntos de vista para afrontar esta elección.

En primer lugar, un artículo de Castañé et al. [4]. En la introducción de este artículo, se destaca cómo los entornos de computación en la nube ofrecen una amplia gama de servicios a los clientes, atrayendo a las aplicaciones de entornos de clúster tradicionales debido a características como la disponibilidad bajo demanda, la capacidad de almacenamiento y el rendimiento de nivel de servicio. Se menciona que la nube se ha vuelto heterogénea debido a la creciente demanda de servicios específicos, y se enfatiza que la computación de alto rendimiento (HPC) en la nube presenta nuevos desafíos acerca del rendimiento y la complejidad. Así mismo, también expone que las soluciones de HPC basadas en la nube se ofrecen mediante varios proveedores comerciales, pero su complejidad aumenta debido a la variedad de recursos disponibles en la nube y la falta de compatibilidad entre diferentes abstracciones de recursos. El artículo propone abordar estos desafíos mediante la creación de una ontología llamada CloudLightning Ontology (CL-Ontology) que respalda la heterogeneidad y el rendimiento en la nube. Esta ontología se extiende a los estándares IEEE-2302 y IEEE-2301 para la interoperabilidad en la nube y se utiliza en una arquitectura de servicios orientada a la semántica para gestionar recursos de manera dinámica y facilitar la toma de decisiones. Además, se presenta la arquitectura del sistema CloudLightning como un caso de uso para demostrar la gestión de recursos heterogéneos a gran escala. El artículo subraya la importancia de abordar estos desafíos en la nube heterogénea y resalta la eliminación de la configuración manual mediante el uso de ontologías semánticas.

Por otro lado, este otro artículo[17], escrito por Hwaitat et al. que propone una clasificación de los atributos. En primer hace una división que separa los componentes *hardware* en tres partes: CPU, memoria y dispositivos de entrada/salida. A su vez, divide cada clase en subclases, por ejemplo, la memoria la divide en principal y secundaria como se puede ver en la siguiente figura 5.1. En nuestro caso, toda la clase de los dispositivos de entrada y salida no se considera relevante para el proyecto. La ontología se basa en la idea de que un sistema informático consta de componentes de *hardware* que interactúan

entre sí, y busca formalizar conceptos que representen el conocimiento en este campo .El artículo presenta ejemplos razonamientos en el contexto de esta ontología, lo que proporciona una base sólida para el entendimiento y la representación de los componentes *hardware* de un sistema informático. Pero también queda abierto a que los conceptos de la ontología cambien, es decir, a que esta evolucione, de manera que pueda seguir siendo útil aun cuando no está completa.

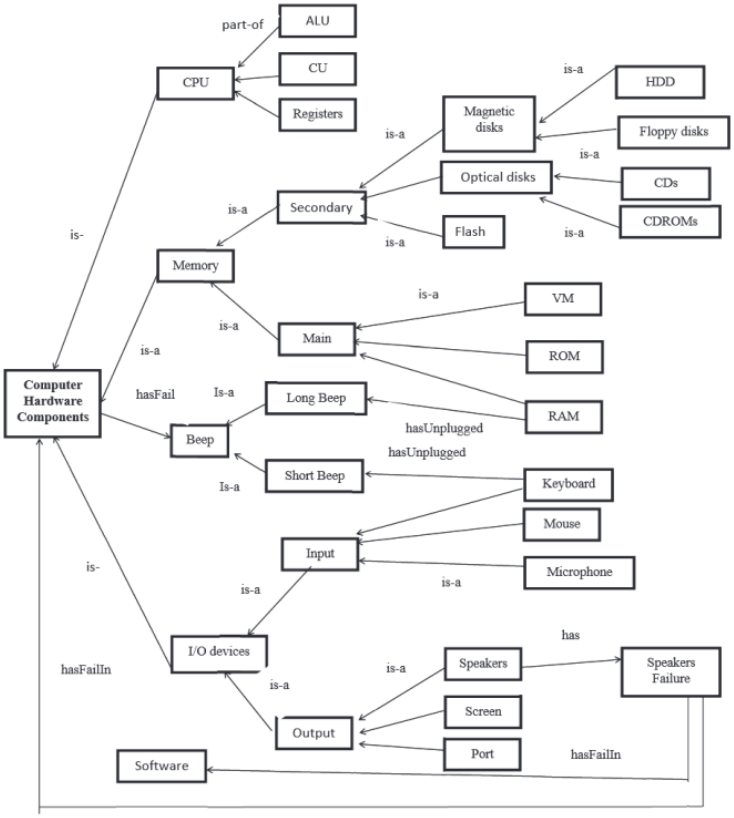

Figura 5.1: Jerarquía de la ontología

5.2 Extracción de los parámetros

Para poder contar con un catálogo son necesarios los datos que lo componen, en esta sección se profundizará en los comandos utilizados para obtenerlos

Por lo que, tras determinar los aspectos a incluir en la ontología, se ha realizado una investigación sobre los diferentes comandos que extraen información desde la consola de comandos del sistema operativo de Linux[18][19][20][21]. Para esta extracción de los datos de las diferentes máquinas se ha optado por el uso de un *script* que se pueda ejecutar en la consola de Linux. Éste recoge los datos, los guarda en variables para más tarde separar los datos por comas, emulando un archivo del tipo csv, ya que, de esta manera se facilita considerablemente su posterior procesamiento. Tiene ciertas dependencias ya que no todos los comandos utilizados son nativos de este sistema operativo. En concreto, los comandos de los que se hace uso para extraer esos datos del sistema para poder cumplir con las especificaciones de la ontología son los siguientes:

1. hostnamectl: Este comando de Linux permite a los usuarios visualizar y en algunos casos modificar diferentes configuraciones del sistema, como la configuración del nombre de host del sistema, el nombre de máquina, el kernel, el sistema operativo y la arquitectura del sistema operativo.

2. hwinfo: Proporciona información detallada sobre el *hardware* del sistema, incluyendo información sobre la CPU, la memoria, los dispositivos de entrada/salida y otros componentes del *hardware*. En el proyecto se utiliza para obtener un informe completo acerca de las propiedades del disco

3. lscpu: Muestra información detallada sobre la unidad central de procesamiento (CPU) del sistema, incluyendo el número de núcleos, la velocidad del reloj, la arquitectura y otras características relacionadas con la CPU. Es útil para obtener información rápida y específica sobre la CPU de un sistema Linux. Su utilidad para el trabajo reside en aportar información sobre el número de CPUs, la cantidad total de sockets, la cantidad de sockets por core y la familia de la CPU.[19]

4. cpupower: Se utiliza para gestionar la configuración y el rendimiento de la CPU. Permite controlar la frecuencia de la CPU, los estados de energía y otras opciones relacionadas con la administración de la CPU. En el proyecto, se emplea para determinar las frecuencias máxima y mínima de la CPU.[20]

5. /proc/cpuinfo: El archivo proporciona información detallada sobre la CPU del sistema, incluyendo el modelo, la velocidad, el número de núcleos y otras características técnicas de la CPU. Para el ámbito del proyecto, con este comando se extrae información de la CPU como la cantidad de CPUs o la cantidad de núcleos.

6. /proc/meminfo: El archivo muestra información detallada sobre el uso de la memoria del sistema, incluyendo datos sobre la memoria total, libre, utilizada y disponible, así como estadísticas sobre el uso de la memoria por parte del sistema operativo y las aplicaciones. Se trata de un fichero que se va actualizando para mostrar este uso de la memoria. Se utiliza para extraer información sobre la RAM.[18]

7. lshw: Es un comando en Linux que proporciona información detallada sobre el *hardware* del sistema, incluyendo la CPU, la memoria, la tarjeta madre, las tarjetas de red, los dispositivos USB y otros componentes. Es útil para obtener una visión general de la configuración del *hardware* en un sistema Linux. En el proyecto, se utiliza para extraer información sobre la GPU del sistema.

8. dmidecode: Se trata de un comando bastante utilizado por administradores de sistemas cuyo objetivo es obtener información acerca de cualquier tipo de especificación del *hardware* de un sistema. Permite acceder a los datos del Sistema de Gestión de Datos de Equipamiento (DMI) presente en la BIOS de un sistema.[21]

Para evitar tener que acceder a cada una de las máquinas para ejecutar el *script*, ya que podría tratarse de que un mismo proveedor tuviera centenares de sistemas que quisiera publicar, se utiliza un nivel más de abstracción. ¿Qué quiere decir esto? Esto significa que un nivel por encima de este código tenemos un manager que se encarga de ejecutar el *script* en todas las máquinas que se desee (es necesario contar con la dirección IP de cada una de las máquinas) mediante conexiones ssh a cada una de las máquinas en cuestión. Este manager es el que posteriormente se conecta al servicio de Kafka como publicador y envía esta información al broker del sistema de mensajería, en el cual se almacenará posteriormente.

El uso de un *script* lanzador para la recolección de datos presenta varias ventajas significativas:

- Automatización: Permite la automatización del proceso de recolección de datos, reduciendo la necesidad de intervención manual y minimizando errores.
- Escalabilidad: El sistema puede manejar la recolección de datos de un gran número de máquinas, lo que es especialmente útil cuando se trata de grandes infraestructuras de computación.
- Eficiencia: La ejecución síncrona y la publicación directa en Kafka optimizan el flujo de datos, asegurando una recolección y almacenamiento rápidos y fiables.
- Centralización: Centraliza la gestión de la recolección de datos, facilitando el monitoreo y el control del proceso.

5.3 Apache Kafka

Para conectar los *providers* con el servicio del catálogo, se emplea el sistema de mensajería Apache Kafka. A lo largo de los próximos párrafos se expondrá el proceso del desarrollo en el ámbito del sistema que se ocupa de la comunicación entre proveedores e infraestructura del catálogo.

Tras consultar con las diferentes opciones disponibles en cuanto al sistema de mensajería, se ha optado por contar con Apache Kafka, por la familiaridad con el sistema y su escalabilidad, ya que el proyecto está preparado para aumentar las instancias del servidor. Para el desarrollo de esta parte de la solución, se ha escogido su versión de Java, mediante un proyecto *SpringBoot*. De manera que en el proyecto se ha desarrollado una aplicación en Java que se encarga de gestionar y almacenar las comunicaciones entre los diferentes componentes del sistema. En concreto, como se puede ver en la figura de la arquitectura4.1, Apache Kafka hace la función de gestionar el catálogo de recursos. Y por lo tanto, se ocupa de la comunicación con los diferentes *providers*, así como la gestión de la publicación en la página web.

Apache Kafka necesita *topics* para realizar las comunicaciones, se ha creado un topic para las comunicaciones con los proveedores de recursos, de manera que envían todos envían mensajes al mismo topic. Hay varios motivos por los cuales un proveedor puede mandar mensajes al servicio de Apache Kafka. En primer lugar para dar de alta un conjunto de máquinas que quieren aportar a la federación. Por otro lado, para actualizar el estado de esas máquinas, que puede ser el caso si hay alguna avería o también en el caso de que un proveedor quiera incorporar alguna nueva máquina al sistema, aunque también sirve para actualizar el estado de las máquinas, es decir, si están o no en uso. El objetivo es que los proveedores de sistemas realicen estas actualizaciones periódicamente hayan o no habido cambios, de manera que se pueda tener en cuenta y de alguna manera anticipar la caída de alguno de los centros de datos, para poder notificarlo a los consumidores.

5.4 Frontend

Los *mockups* de la interfaz web son representaciones visuales ampliamente utilizadas en el proceso de diseño y desarrollo de aplicaciones e interfaces. Estas maquetas permiten visualizar la estructura y el diseño inicial de la interfaz de usuario antes de que comience la fase de implementación. En este apartado, se presentan los *mockups* del catálogo de recursos de *hardware*, que han sido diseñados para asegurar una experiencia de usuario intuitiva y eficiente. A través de estos *mockups*, se busca anticipar el flujo de navegación, la disposición de los elementos interactivos y la estética general de la plataforma.

Para comenzar, la página principal del catálogo de recursos *hardware* será una tabla en la cual se podrán consultar las máquinas adheridas al mismo. Cada una de las columnas se podrá ordenar, para así facilitar la filtración de todos los sistemas.

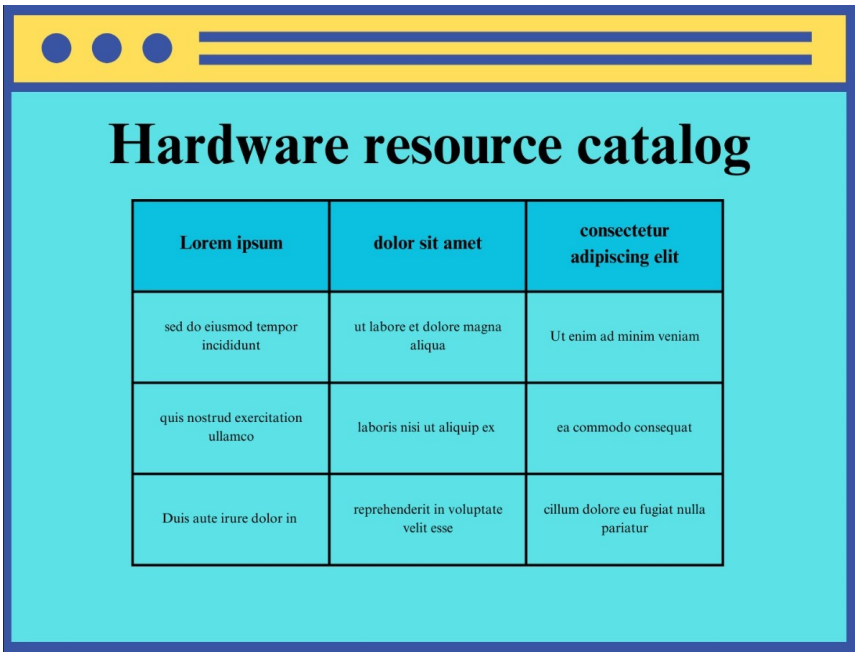

Figura 5.2: Interfaz del catálogo

Habrá otra interfaz que nos permitirá decidir qué parámetros acerca de las máquinas estarán visibles, se podrá seleccionar cualquier conjunto de parámetros que deberán ser mostrados por la interfaz principal del catálogo.

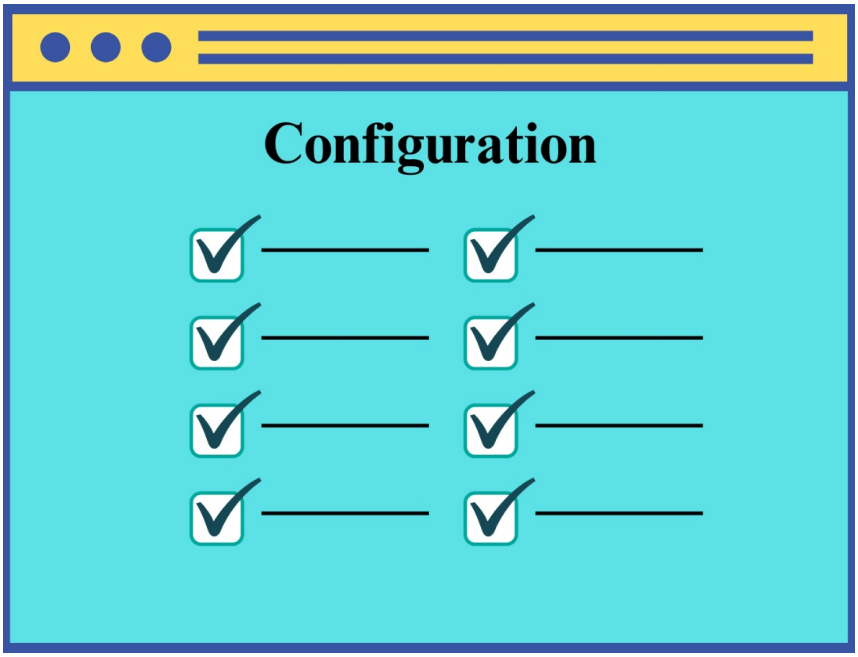

Figura 5.3: Interfaz de configuración

Para exponer los datos de los que dispone el sistema a los diferentes usuarios se ha optado por utilizar una aplicación web como *frontend*. En dicha aplicación web se ha utilizado HTML acompañado de CSS para mejorar y facilitar la interfaz tanto por la parte del diseño, como de la visibilidad y facilidad de uso de la misma.

Para el desarrollo del *frontend*, hemos utilizado tres tecnologías fundamentales en la creación de interfaces web: HTML, JavaScript y CSS. Cada una de estas tecnologías cumple un rol específico en la estructura, funcionalidad y presentación de la página, permitiéndonos crear una agradable experiencia de usuario.

HTML (HyperText Markup Language): HTML es el lenguaje de marcado elegido que se ha utilizado para estructurar el contenido de nuestra página web. A través de etiquetas específicas, definimos los diferentes elementos que componen la interfaz, como tablas, enlaces y formularios. La estructura proporcionada por HTML no solo facilita la organización del contenido, sino que también mejora la accesibilidad por parte de las otras partes de la interfaz, como es el caso de Javascript con el DOM.

CSS (Cascading Style Sheets): CSS se encarga de la presentación visual de la página web. Mediante reglas de estilo, definimos la apariencia de los elementos HTML, especificando propiedades como colores, márgenes, bordes y disposición en la pantalla. Utilizando técnicas de *responsive design*, se garantiza que la página se adapte correctamente a una gran cantidad de tamaños de pantalla y dispositivos, proporcionando una experiencia de usuario consistente en todos ordenadores de escritorio e incluso en dispositivos móviles.

JavaScript: JavaScript añade interactividad y dinamismo a la página web. Con este lenguaje de programación, implementamos funcionalidades que permiten a los usuarios interactuar con la página de manera más efectiva. Esto incluye la validación de formularios en tiempo real, la manipulación del DOM (Document Object Model) para actualizar el contenido sin recargar la página, y la integración de APIs externas para obtener datos adicionales. Además, utilizamos bibliotecas y *frameworks* populares que facilitan el desarrollo y mejoran el rendimiento de las aplicaciones web.

La integración de estas tecnologías(HTML, CSS y JavaScript) se realiza de manera que cada una de las tecnologías complementa a las otras. De manera que HTML proporciona la estructura sobre la que se aplican los estilos definidos en CSS. Por su parte, JavaScript interactúa con el HTML a través del DOM, en nuestro caso se encarga de rellenar la tabla con todas las características de los dispositivos que forman parte de la federación, así como agregar la opción de poder ordenar cada una de las columnas de la tabla.

CAPÍTULO 6
Implantación

La etapa de implantación es una de las fases en el desarrollo de cualquier sistema informático, ya que es el momento en que se lleva a cabo la transición del entorno de desarrollo al entorno de producción. En este apartado se detalla el proceso de instalación, configuración y puesta en marcha del sistema desarrollado. La implantación busca verificar la correcta puesta en marcha del sistema. Además, esta fase incluye la obtención de resultados que demuestran la eficacia de la solución propuesta, asegurando que cumple con los requisitos establecidos y que está preparada para su explotación en el ámbito para el cual fue diseñada.

6.1 Recopilación de los datos

Se ha diseñado un *script bash* que actuará de *manager* o lanzador de otro *script bash* que se encarga propiamente de recoger los parámetros de la máquina en cuestión. Este lanzador de *scripts* funciona de la siguiente manera. Se envía mediante ssh la ejecución del *script* a cada una de las IPs que se envíe como parámetro de ejecución. Se trata de un proceso síncrono, y a medida que recibe resultados los publica en un topic de Apache Kafka.

Proceso de Ejecución

- Preparación de la lista de IPs: El *script* lanzador recibe como parámetro una lista de direcciones IP correspondientes a las máquinas de las cuales se desea recopilar datos. Estas IPs pueden estar almacenadas en un archivo o ser proporcionadas directamente al *script*.

- Establecimiento de conexión SSH: Utilizando SSH, el *script* lanzador se conecta a cada una de las máquinas especificadas en la lista de IPs. Para ello, es necesario contar con las credenciales adecuadas y configurar el acceso SSH en cada máquina destino.

- Ejecución remota del *script*: Una vez establecida la conexión, el *script* lanzador envía el comando para ejecutar el *script* de recolección de datos en la máquina remota. Este segundo *script* es el encargado de recopilar los parámetros de *hardware* y otros datos relevantes de la máquina.

- Recolección de resultados: A medida que el *script* de recolección finaliza su ejecución en cada máquina, los datos obtenidos son enviados de vuelta al *script* lanzador.

Este proceso se realiza de manera síncrona, asegurando que los datos de una máquina se reciben y procesan antes de pasar a la siguiente.

- Publicación en Apache Kafka: Los resultados recopilados por el *script* lanzador son publicados en un topic de Apache Kafka. Kafka actúa como un intermediario de mensajería, almacenando y gestionando los datos recibidos para su posterior procesamiento y análisis. Esto permite una integración eficiente y escalable de los datos en el sistema de catálogo de recursos.

6.2 Apache Kafka

Para poder contar con un sistema de mensajería, se levanta el servicio de Apache Kafka utilizando su versión docker de confluentinc. Concretamente "confluentinc/cp-zookeeper"[22] y "confluentinc/cp-kafka"[23], configurando el servicio haciendo uso de los parámetros expuestos desde el docker compose, el hacer uso de la tecnología docker permite poder escalar y desescalar de manera sencilla y elástica.

Para la implantación de este elemento de la solución se ha creado un archivo "docker-compose.yml", el cual define los servicios necesarios para Apache Kafka. Este archivo incluye la definición de los servicios: ZooKeeper y Kafka por un lado y la aplicación por el otro. En este archivo se configuran parámetros como los puertos escuchados por cada servicio, o el tiempo máximo de espera para cerrar una conexión activa.

6.3 Aplicación

Se trata de una aplicación Java encargada de gestionar el servicio Apache Kafka desplegado y las consultas por parte del *frontend*. Esta aplicación se levanta mediante una imagen docker, por lo que se permite su escalado en caso de ser necesario, ya sea por un aumento en el tráfico de solicitudes de tipo GET por parte de solicitudes de posibles clientes, que accedan al sistema para consultar y/o contratar los dispositivos miembros de la federación. Otro caso para el cual se requiere el escalado es garantizar la alta disponibilidad, al tener múltiples réplicas del servicio, los fallos son más asumibles, ya que no caería todo el sistema.

Después de levantar los contenedores, es necesario configurar Apache Kafka para que se integre adecuadamente con el sistema desarrollado. Esto incluye la creación de los *topics* necesarios para la comunicación entre los proveedores de recursos y el catálogo. Esta configuración se encarga la aplicación. Creación de Topics: Se crean los *topics* necesarios utilizando las herramientas de línea de comandos de Kafka. Un topic es una categoría o *feed* donde se publican los registros. En este caso, se crea un *topic* para recibir los datos de los proveedores de recursos.

6.4 Frontend

Para la sección del *frontend*, se ha optado por utilizar el servidor web de páginas personales que ofrece la Universidad[24]. Esta elección ha proporcionado una solución conveniente, ya que al utilizar una plataforma ya existente, se ha simplificado considerablemente el proceso de integración y despliegue del proyecto. Al no tener que preocuparnos por la configuración y el mantenimiento de un entorno de desarrollo independiente, hemos podido concentrar nuestros esfuerzos en otros aspectos del proyecto. Sin embargo,

esta solución también presenta ciertas limitaciones, principalmente en lo que respecta al desarrollo del *backend*, ya que el servicio proporcionado por la Universidad no permite la implementación de funcionalidades del lado del servidor. Esta restricción ha condicionado el alcance del proyecto, impidiendo la adición de características más complejas que podrían haber enriquecido la funcionalidad del sistema.

A pesar de estas limitaciones, la solución ofrecida por la Universidad ha sido adecuada para los objetivos del proyecto, permitiendo la implementación de un *frontend* funcional sin mayores complicaciones técnicas. Esta solución temporal ha sido suficiente para las necesidades actuales del proyecto, pero sería recomendable explorar otras alternativas algo más avanzadas si se busca ampliar la funcionalidad o mejorar el rendimiento del sistema en fases posteriores del desarrollo. Esta cuestión, así como otras posibles mejoras, se discuten con mayor detalle en el capítulo 9.

CAPÍTULO 7
Pruebas

En este capítulo se detallarán las pruebas realizadas para asegurar que la solución desarrollada cumple con los requisitos y expectativas definidos. Se presentarán pruebas de verificación, destinadas a garantizar que el sistema funciona correctamente desde un punto de vista técnico y funcional. En las próximas secciones vamos a comprobar cada una de las partes del proyecto.

7.1 Script

El propósito del *script* dentro del proyecto es la extracción de los parámetros y características de una máquina, por lo tanto, para la comprobación de su funcionamiento bastará con probarlo localmente.

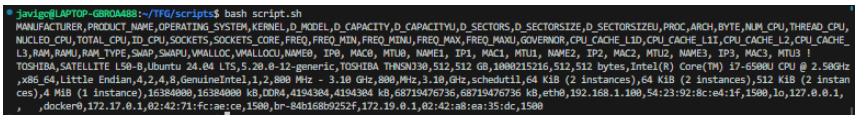

Figura 7.1: Ejecución script

Hemos comprobado que el *script* funciona adecuadamente en local para una sola máquina, como se puede ver en la figura, en la que se extraen los parámetros de la máquina que ejecuta el código.

7.2 Docker Compose

Con el archivo docker-compose.yml definido, se inicia el servicio ejecutando el comando “docker-compose up -d”. Este comando descarga las imágenes necesarias y levanta los contenedores de ZooKeeper y Kafka.

Una vez que los contenedores están en funcionamiento, se verifica que los servicios estén activos y funcionando correctamente. Esto se puede hacer utilizando comandos como “docker ps” para listar los contenedores en ejecución y “docker logs” para ver los registros de los contenedores. A continuación vamos a introducir una figura, en la que se puede observar el despliegue de los tres contenedores docker mediante los cuales se pone en marcha el proyecto.

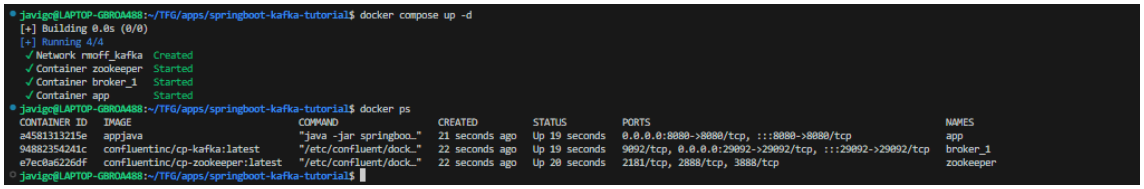

Figura 7.2: Despliegue Docker

7.3 API REST

Para verificar que este apartado de la solución funciona adecuadamente se ha utilizado un add-on de Mozilla Firefox, concretamente un cliente REST[25] para hacer peticiones, permite peticiones de tipo GET que es lo que se necesita para esta prueba.

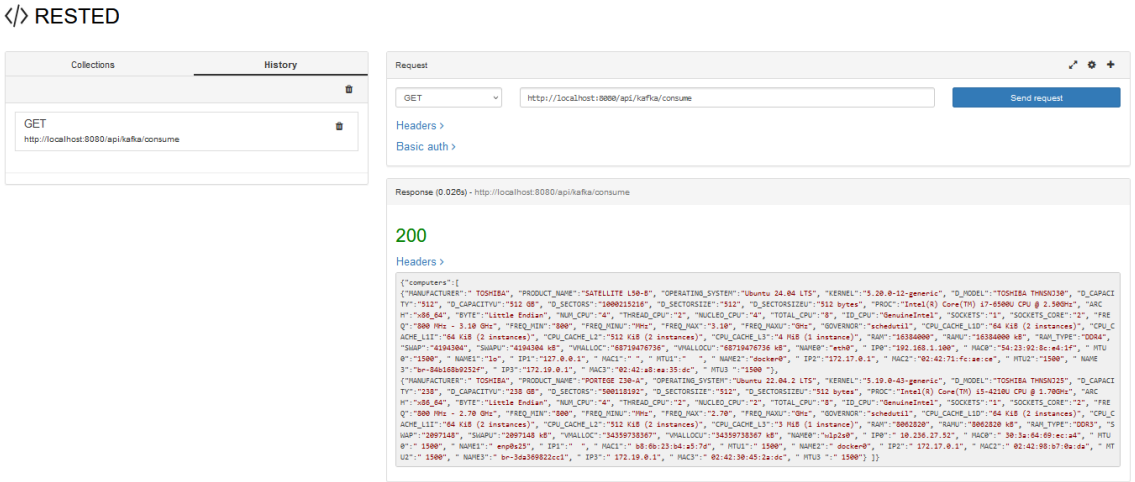

Figura 7.3: Operación GET api

A continuación se muestra una captura del contenido organizado de este fragmento JSON.

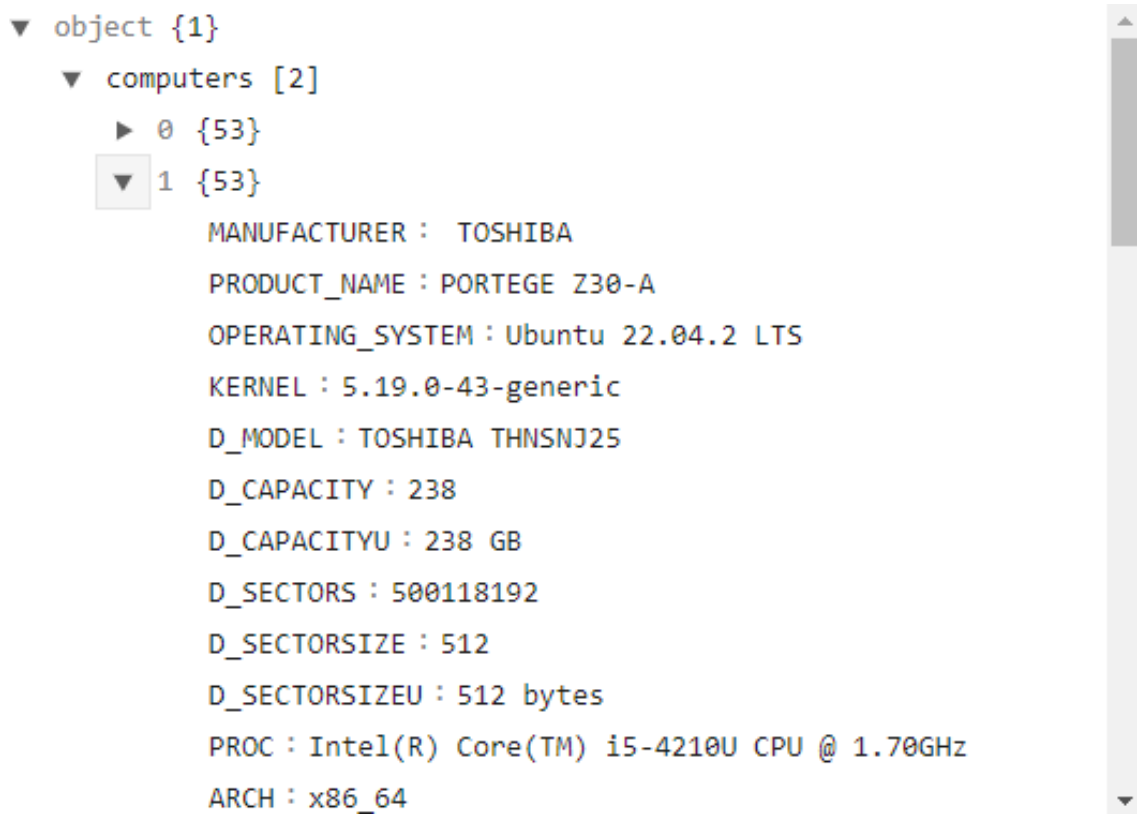

Figura 7.4: Respuesta API get organizado

Como se puede observar hay dos objetos incluidos dentro del campo computers, cada uno de ellos es un computador distinto, y en esta captura se pueden observar las primeras características del segundo de ellos.

7.4 Frontend

Inicialmente al abrir la página web se presenta la interfaz de la figura 7.5, la cual se trata de un menú de configuración en el cual puedes elegir mostrar los parámetros en cualquiera de las combinaciones posibles, para mayor comodidad, existe la opción de seleccionar y deseleccionar todos con un solo clic.

Figura 7.5: Interfaz de configuración

Una vez elegidos los parámetros deseados, en este caso “Manufacturer”, “Product Name”, “Operating System” y “Proc”, tras pulsar seleccionar se puede ver una tabla creada con esos parámetros seleccionados.

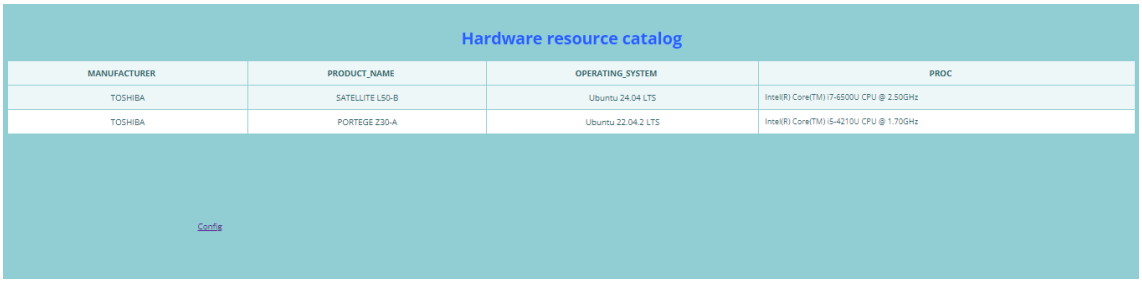

Figura 7.6: Interfaz de visualización

CAPÍTULO 8
Conclusiones

El capítulo de conclusiones cierra el ciclo de este proyecto, sintetizando los principales hallazgos, logros y aprendizajes obtenidos a lo largo del desarrollo del proyecto. En esta sección se destacan los objetivos alcanzados, se reflexiona sobre los desafíos enfrentados, y se valoran las implicaciones del proyecto tanto desde una perspectiva técnica como de su impacto potencial. Además, se discuten las limitaciones del trabajo y posteriormente se sugerirán posibles mejoras y desarrollos futuros que podrían ampliar y perfeccionar los resultados obtenidos. Finalmente, se ofrece una visión global del valor que este proyecto aporta al campo de estudio y su relevancia para futuras investigaciones o aplicaciones prácticas.

8.1 Cumplimiento de los objetivos

El objetivo principal del proyecto era el de desarrollar un gestor de recursos *hardware* en un entorno federal, lo que implicaba decidir qué información había que recopilar de cada recurso federado. Para lograrlo, se ha diseñado y desarrollado una ontología capaz de organizar y estructurar la información de manera eficiente. Este objetivo se ha cumplido satisfactoriamente, permitiendo una representación uniforme y detallada de los recursos, lo cual facilita la integración y gestión dentro del sistema federado.

El desarrollo de la ontología, que constituía el primer objetivo específico, también fue alcanzado con éxito. Se realizó un estudio exhaustivo para identificar y seleccionar los parámetros más relevantes que influirían en la elección de conectarse a un sistema u otro. Este proceso permitió crear una ontología robusta, evitando la omisión de parámetros importantes y asegurando que los recursos estuvieran descritos con un nivel de detalle adecuado. Además, se logró estandarizar los parámetros para todos los recursos, el objetivo era la minimización de los valores nulos y garantizar la coherencia y capacidad de comparar los datos en el catálogo.

Otro objetivo importante del proyecto era adquirir conocimientos y experiencia en el uso de Apache Kafka, una tecnología determinante para la implementación de la mensajería basada en el modelo publicador-suscriptor. Este objetivo también se ha cumplido. Durante el proyecto, se integró Apache Kafka con éxito en el sistema, lo que no solo ha permitido un intercambio eficiente de mensajes, sino que también ha proporcionado una plataforma de aprendizaje valiosa para profundizar en esta tecnología. La experiencia adquirida en la configuración, manejo y optimización de Kafka será de gran utilidad en proyectos futuros que requieran un manejo avanzado de un sistema de mensajería en el ámbito de los sistemas distribuidos.

En resumen, todos los objetivos propuestos se han alcanzado de manera satisfactoria, con resultados que cumplen y, en algunos aspectos, superan las expectativas iniciales. El proyecto ha logrado tanto el desarrollo técnico del sistema como el enriquecimiento del conocimiento y habilidades en tecnologías altamente utilizadas hoy en día en la industria.

8.2 Relación del trabajo desarrollado con los estudios cursados

El trabajo se puede relacionar con las siguientes asignaturas

CSD: tiene relación con toda la parte de sistemas distribuidos acerca de la replicación de los brokers y temas del Apache Kafka.

TSR: esta misma parte de Apache Kafka y la dockerización de la aplicación en contenedores.

GPR: este trabajo guarda relación con esta asignatura en el campo del desarrollo de un proyecto, las tareas, tiempo asociado a cada una y la metodología seguida en el desarrollo de la aplicación y de cada una de ellas.

DEW: en cuanto a la parte del *frontend*, que será una aplicación web. Desarrollo Web es una asignatura que aporta las bases necesarias que se requieren para la realización de esta tarea.

CAPÍTULO 9
Trabajos futuros

En este capítulo se detallan posibles líneas de desarrollo y mejora que podrían abordarse en trabajos futuros relacionados con este proyecto. Se discutirán áreas en las que se podrían incorporar nuevas funcionalidades, mejorar la eficiencia, optimizar la experiencia del usuario, o incluso integrar tecnologías emergentes. Además, se propondrán posibles enfoques para superar las limitaciones identificadas durante el desarrollo del proyecto, con el objetivo de orientar investigaciones y desarrollos futuros que enriquezcan y amplíen el alcance del sistema.

En primer lugar, sería altamente recomendable migrar el *frontend* a un desarrollo independiente, desvinculándolo de las páginas personales de la UPV, que actualmente se utilizan como una solución provisional. Esta migración permitiría una mayor flexibilidad en el diseño y desarrollo de la interfaz de usuario. Además, al independizar el *frontend*, se facilitaría su escalabilidad y mantenimiento a largo plazo, permitiendo actualizaciones más frecuentes y una mejor integración con servicios externos. También abriría la puerta a una mayor personalización y optimización del rendimiento, aspectos clave para mejorar la experiencia del usuario y asegurar que la aplicación pueda crecer y adaptarse a las futuras necesidades del proyecto. No se ha realizado esta migración dentro de este trabajo de fin de grado ya que nos el objetivo en este apartado era el de tener un prototipo de la interfaz web, para así poder aplicarlo independientemente del servidor web que cada organización de la federación prefiera utilizar.

En el marco del proyecto en el que se desarrollaron estas tareas había actividades centradas en la gestión de identidades en la federación resultante. Por ello, este TFG no ha incluido ningún esfuerzo relacionado con esa área, pero para llegar a efectuar un despliegue serio, debería complementarse con una gestión de identidades y autenticación suficientemente robusto. Este sistema permitiría la creación de perfiles para cada usuario, facilitando el control de acceso y la asignación de roles específicos dentro de la plataforma. Por ejemplo, mediante un sistema de autenticación con contraseña abriría la posibilidad de implementar funcionalidades avanzadas, como la recuperación de contraseñas, autenticación multifactor (MFA) para aumentar la seguridad, y la integración con servicios de inicio de sesión único (SSO) o autenticación basada en OAuth, facilitando la interoperabilidad con otras plataformas. Esta mejora no solo garantizaría la protección de los datos y recursos, sino que también mejoraría la experiencia del usuario al proporcionar un entorno más seguro y personalizado.

CAPÍTULO 10
Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas aquellas personas e instituciones que, de una u otra manera, han hecho posible la realización de este Trabajo de Fin de Grado.

En primer lugar, quisiera agradecer a mi tutor de TFG, Francisco Daniel Muñoz Escoí, por su constante apoyo, orientación y valiosos consejos a lo largo de todo el proceso. Su experiencia y conocimientos han sido fundamentales para la compleción de este proyecto.

Extiendo también mi agradecimiento al ITI, cuya colaboración ha sido clave en el desarrollo de este trabajo. Su apoyo y recursos han enriquecido significativamente el proyecto, permitiendo un enfoque más aplicado y profesional.

Agradezco también a los profesores y profesoras del Grado de Ingeniería Informática, gracias a ellos he sido capaz de aprender sobre una enorme cantidad de temas fundamentales para mi desarrollo académico y profesional.

Agradezco también a mis compañeros y compañeras de clase, con quienes he compartido innumerables horas de estudio y aprendizaje

Bibliografía

[1] Definición de federación. https://dpej.rae.es/lema/federaci%C3%B3n, 2018. Accedido 19-05-2023.

[2] Gaia-x Home page. https://gaia-x.eu/, 2018. Accedido 18-05-2023.

[3] AI on Demand Home page. https://www.ai4europe.eu/, 2018. Accedido 18-05-2023.

[4] Gabriel G Castañé, Huanhuan Xiong, Dapeng Dong, and John P Morrison. An ontology for heterogeneous resources management interoperability and hpc in the cloud. *Future Generation Computer Systems*, 88:373–384, 2018.

[5] Francesco Moscato, Rocco Aversa, Beniamino Di Martino, Dana Petcu, Massimiliano Rak, Salvatore Venticinque, et al. An ontology for the cloud in mosaic. In *Cloud Computing: Methodology, System, and Applications*, volume 1, pages 467–485. CRC Press, Taylor & Francis Group, 2011.

[6] George Kousiouris, George Vafiadis, and Marcelo Corrales. A cloud provider description schema for meeting legal requirements in cloud federation scenarios. In Christos Douligeris, Nineta Polemi, Athanasios Karantjias, and Winfried Lamersdorf, editors, *Collaborative, Trusted and Privacy-Aware e/m-Services - 12th IFIP WG 6.11 Conference on e-Business, e-Services, and e-Society, I3E 2013, Athens, Greece, April 25-26, 2013. Proceedings*, volume 399 of *IFIP Advances in Information and Communication Technology*, pages 61–72. Springer, 2013.

[7] Stefan Dumss, Markus Weber, Clemens Schwaiger, Clemens Sulz, Patrick Rosenberger, Friedrich Bleicher, Manfred Grafinger, and Matthias Weigold. Euprogigant–a concept towards an industrial system architecture for data-driven production systems. *Procedia CIRP*, 104:324–329, 2021.

[8] Giuseppe Attardi, Beniamino Di Martino, Antonio Esposito, and Michele Mastroianni. Using federated cloud platform to implement academia services for research and administration. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 413–418. IEEE, 2018.

[9] Frank Dabek, Russ Cox, M. Frans Kaashoek, and Robert Tappan Morris. Vivaldi: a decentralized network coordinate system. In Raj Yavatkar, Ellen W. Zegura, and Jennifer Rexford, editors, *Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 30 - September 3, 2004, Portland, Oregon, USA*, pages 15–26. ACM, 2004.

[10] Jonathan Ledlie, Paul Gardner, and Margo I. Seltzer. Network coordinates in the wild. In Hari Balakrishnan and Peter Druschel, editors, *4th Symposium on Networked Systems Design and Implementation (NSDI 2007), April 11-13, 2007, Cambridge, Massachusetts, USA, Proceedings*. USENIX, 2007.

[11] Bernard Wong, Aleksandrs Slivkins, and Emin Gün Sirer. Meridian: A lightweight network location service without virtual coordinates. *ACM SIGCOMM Computer Communication Review*, 35(4):85–96, 2005.

[12] Yang Chen, Yongqiang Xiong, Xiaohui Shi, Beixing Deng, and Xing Li. Pharos: A decentralized and hierarchical network coordinate system for internet distance prediction. In *IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference*, pages 421–426. IEEE, 2007.

[13] Zhuo Chen, Yang Chen, Yibo Zhu, Cong Ding, Beixing Deng, and Xing Li. Tarantula: Towards an accurate network coordinate system by handling major portion of tivs. In *2011 IEEE Global Telecommunications Conference-GLOBECOM 2011*, pages 1–6. IEEE, 2011.

[14] Yang Chen, Xiao Wang, Xiaoxiao Song, Eng Keong Lua, Cong Shi, Xiaohan Zhao, Beixing Deng, and Xing Li. Phoenix: Towards an accurate, practical and decentralized network coordinate system. In *NETWORKING 2009: 8th International IFIP-TC 6 Networking Conference, Aachen, Germany, May 11-15, 2009. Proceedings 8*, pages 313–325. Springer, 2009.

[15] Khin Me Me Thein. Apache Kafka: Next generation distributed messaging system. *International Journal of Scientific Engineering and Technology Research*, 3(47):9478–9483, 2014.

[16] ¿Cuánto gana un ingeniero informático? https://www.uax.com/blog/ingenieria/cuanto-cobra-un-ingeniero-informatico, 2024. Accedido 04-09-2024.

[17] Ahmad K AL Hwaitat, Ameen Shaheen, Khalid Adhim, Enad N Arkebat, Aezz Aldain AL Hwiatat, et al. Computer hardware components ontology. *Modern Applied Science*, 12(3):35–40, 2018.

[18] The /proc/meminfo File in Linux. https://www.baeldung.com/linux/proc-meminfo#:~:text=The%20%2Fproc%2Fmeminfo%20file%20inside,can%20analyze%20this%20file's%20contents., 2022. Accedido 27-07-2023.

[19] How to Use Linux lscpu Command Tutorial. https://linuxhint.com/lscpu-command/, 2021. Accedido 27-07-2023.

[20] CPU frequency scaling. https://wiki.archlinux.org/title/CPU_frequency_scaling, 2023. Accedido 27-07-2023.

[21] dmidecode command in Linux with Examples. https://data-flair.training/blogs/dmidecode-linux/, 2023. Accedido 27-07-2023.

[22] Image confluentinc/cp-zookeeper. https://hub.docker.com/r/confluentinc/cp-zookeeper, 2024. Accedido 15-05-2024.

[23] Image confluentinc/cp-kafka. https://hub.docker.com/r/confluentinc/cp-kafka, 2024. Accedido 15-05-2024.

[24] Servicio de Publicación Estándar (webs Personales). https://wiki.upv.es/confluence/pages/viewpage.action?pageId=390136139, 2024. Accedido 20-07-2024.

[25] Firefox RESTED. https://addons.mozilla.org/en-US/firefox/addon/rested/, 2024. Accedido 22-08-2024.

APÉNDICE A
Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10.**Reducción de las desigualdades.**			X	
ODS 11.**Ciudades y comunidades sostenibles.**				X
ODS 12.**Producción y consumo responsables.**				X
ODS 13.**Acción por el clima.**			X	
ODS 14.**Vida submarina.**				X
ODS 15.**Vida de ecosistemas terrestres.**				X
ODS 16.**Paz, justicia e instituciones sólidas.**				X
ODS 17.**Alianzas para lograr objetivos.**		X		

Tabla A.1: Tabla de los ODS

Reflexión acerca de la relación del tfg con los ODS más relacionados

- ODS 9: Industria, innovación e infraestructuras.

  El TFG se centra en la creación y mejora de infraestructuras tecnológicas para la federación de recursos de computación, lo que impulsa la innovación y el desarrollo de nuevas tecnologías en la industria. Este proyecto contribuye directamente a la construcción de infraestructuras resilientes y a la promoción de la innovación, apoyando así el crecimiento sostenible en el sector tecnológico.

- ODS 10: Reducción de las desigualdades.

  La federación de recursos y la posibilidad de compartir infraestructuras tecnológicas avanzadas pueden ayudar a reducir las desigualdades en el acceso a la tecnología. Este enfoque permite que comunidades y organizaciones con recursos limitados puedan acceder a tecnologías de alto rendimiento, lo que contribuye a la reducción de brechas tecnológicas y promueve una distribución más equitativa de los recursos.

- ODS 13. Acción por el clima

  El proyecto contribuye al ODS 13 al fomentar la creación y utilización de infraestructuras de computación más eficientes y sostenibles. Al desarrollar un sistema de federación de recursos *hardware*, se optimiza el uso de estos recursos, lo que puede reducir la necesidad de construir nuevas instalaciones y, por tanto, disminuir el impacto ambiental. Además, la federación de recursos facilita la realización de proyectos de investigación y desarrollo enfocados en la mitigación y adaptación al cambio climático, permitiendo un mejor uso de la tecnología en la lucha contra el calentamiento global.

- ODS 17. Alianzas para lograr los objetivos

  El proyecto apoya el ODS 17 al facilitar la colaboración y cooperación entre diferentes entidades y proyectos de la Unión Europea mediante la federación de recursos *hardware*. Esta iniciativa fomenta la construcción de alianzas estratégicas entre instituciones de investigación, empresas y gobiernos, permitiendo compartir recursos y conocimientos. La creación de una plataforma federada de recursos de computación no solo impulsa la innovación, sino que también fortalece las capacidades técnicas y científicas de las organizaciones, promoviendo un desarrollo más equitativo y sostenible a nivel global.