

2. Propuesta técnica – Solución ofertada

2.1. Identificación de las funcionalidades a desarrollar

El proyecto Traza tiene como objetivo centralizar la información ambiental en materia de residuos, controlar su trazabilidad, mejorar la eficiencia del proceso, automatizar y erradicar fallos, impulsar la mejora de planificación del trabajo en toda la cadena de valor, optimizar los recursos y servir de ayuda, según la normativa vigente.

Dado el nivel de detalle de especificación funcional que refleja el PLIEGO y las limitaciones de tamaño del presente documento, hemos optado por realizar un enfoque más técnico en la solución ofertada orientándola a la descripción de la arquitectura propuesta y al encaje en dicha arquitectura de los objetivos funcionales.

2.1.1. Gestión de la Plataforma Traza

Microservicio Gestión de Usuarios

Propósito y Responsabilidades: El microservicio de Gestión de Usuarios se encarga de todas las operaciones relacionadas con la administración de usuarios en una aplicación. Las operaciones que incluye son las siguientes:

- Registro de Usuarios: Permitir a nuevos usuarios crear una cuenta.
- Autenticación: Validar credenciales de usuario para permitir el acceso al sistema.
- Autorización: Determinar los permisos y roles de los usuarios para acceder a recursos específicos.
- Gestión de Perfiles: Permitir a los usuarios actualizar su información personal.
- Manejo de Roles y Permisos: administrar roles y permisos asignados a usuarios.

Interacciones:

API Pública: Provee endpoints RESTful para interactuar con otros servicios y aplicaciones cliente.

BBDD: Se almacena información de usuarios, roles y permisos.

Microservicio Gestión de organizaciones

Propósito y Responsabilidades: El microservicio de Gestión de Organizaciones facilita la gestión de entidades locales y su integración con otros sistemas.

Interacciones:

API Pública: Provee endpoints RESTful para interactuar con otros servicios y aplicaciones cliente.

BBDD: Almacenar información de organizaciones (EELL, Empresas, etc)

Microservicio Parametrización del sistema

Propósito y Responsabilidades: El microservicio de parametrización, tiene como misión ajustar la configuración interna de cada uno de los componentes que forman parte de la plataforma TRAZA de forma centralizada, para que se adapten a las necesidades reales y específicas de cada entidad participante.

Las responsabilidades de este microservicios son:

- **Gestión de Parámetros:** Crear, leer, actualizar y eliminar parámetros del sistema.
- **Validación de Parámetros:** Asegurar que los parámetros cumplan con los requisitos y restricciones definidos.
- **Versionado de Configuraciones:** Mantener un historial de cambios en las configuraciones para auditoría y recuperación.
- **Distribución de Configuraciones:** Proveer configuraciones actualizadas a los microservicios y aplicaciones.

Interacciones:

API Pública: Proveer endpoints RESTful para interactuar con aplicaciones cliente y otros microservicios.

BBDD de Configuraciones: Almacenar parámetros y configuraciones del sistema.

Otros Microservicios: Distribuir configuraciones y recibir solicitudes de configuración.

Microservicio Gestión de maestros

Propósito y Responsabilidades: Es el microservicio encargado de mantener las tablas maestras de la plataforma de forma centralizada. Estas tablas contienen datos comunes y esenciales como el registro de matrículas y compañías de traslado de residuos, listado de organizaciones participantes, lista de orígenes, lista de destinos, ayuntamientos, etc.

Las responsabilidades de este microservicios son:

- **Crud de tablas maestras:** Crear, leer, actualizar o eliminar registros
- **Validación de Datos:** Asegurar que los datos cumplan con los requisitos y restricciones definidas.

Interacciones:

API Pública: Proveer endpoints RESTful para interactuar con aplicaciones cliente y otros microservicios.

Base de Datos de Tablas Maestras: Almacenar datos de referencia.

Otros Microservicios: Distribuir datos de tablas maestras y recibir solicitudes de información.

2.1.2. Parte Operativa de Traza

Microservicio Registro de operaciones

Propósito y Responsabilidades: Este microservicio será el encargado del registro de cada operación que se realice en el proceso para después poder trazarlo.

Las operaciones de TRAZA permitirán la gestión, control y verificación de la presentación de los documentos de traslado (DIs, NT, ...), así como su generación automática previa a los traslados.

También este microservicio, será el encargado de lanzar las alertas y avisos que se definan para cada proceso.

Algunas de las funciones a implementar serían:

- Generación de DIs
- Generación NTs
- Transfronterizos
- Archivo cronológico
- Memoria resumen
- Alertas y avisos

Interacciones

API Pública: Proveer endpoints RESTful para interactuar con aplicaciones cliente y otros microservicios.

BBDD: Almacenar información de entrada y salida

Otros Microservicios: Proveer servicios de alertas y avisos y autorización para otros microservicios.

Microservicio Control de acceso y validaciones.

Propósito y Responsabilidades: La plataforma TRAZA gestionará unas operaciones relativas al acceso de los vehículos a los distintos puntos de la cadena de valor del residuo. Para ello, una funcionalidad de este microservicio será la de “predicción en base al histórico”.

Este microservicio también será el encargado de estimar en el tiempo la cantidad de residuos y su vigencia para que, a posteriori, el servicio de operaciones, genere de forma automática los documentos de traslado (NTs y DIs).

Interacciones:

API Pública: Proveer endpoints RESTful para interactuar con aplicaciones cliente y otros microservicios.

BBDD: Almacenar información de entrada y salida

Otros Microservicios: Proveer servicios de validación y autorización para otros microservicios y de generación.

Microservicio Interfaz para Usuarios y Gestores

Propósito y Responsabilidades: Facilita la interacción de usuarios internos y gestores con el sistema. Tiene la responsabilidad de proveer interfaces intuitivas para usuarios/gestores y gestión de permisos y roles de usuario.

Las funciones a implementar serían:

- Desarrollar interfaces web y móviles
- Implementar gestión de roles y permisos para acceso a diferentes funcionales del sistema (ejemplo: permisos de visibilidad de un botón)

Interacciones:

BBDD: Almacenar información de configuración

Otros Microservicios: Proveer de funcionalidad visual a otros microservicios

Microservicio Gestión de Almacenamiento

Propósito y Responsabilidades: El objetivo de este microservicio es asegurar la disponibilidad y fiabilidad de los datos almacenados. Su responsabilidad es la captura y almacenamiento de datos y gestión de BBDD.

Las funciones a implementar serían:

- Desarrollar mecanismos de captura automatizada de datos
- Administrar bases de datos para asegurar disponibilidad y fiabilidad.

Interacciones:

API Pública: Provee endpoints RESTful para interactuar con otros servicios

BBDD: Utiliza una base de datos para almacenar y proveer de información.

Microservicio Gestión de Almacenamiento de documentos e imágenes

Propósito y Responsabilidades: El objetivo de este microservicio es separar el almacenamiento de documentos e imágenes de la lógica principal de la aplicación. Tiene la responsabilidad de recibir, almacenar y proporcionar el acceso a documentos e imágenes y de implementar medidas de seguridad de los datos almacenados.

Las funciones a implementar serían:

- Recibir los documentos e imágenes
- Proveer acceso a los documentos e imágenes almacenados
- Facilitar la búsqueda y recuperación eficiente
- Almacenar y gestionar información adicional sobre cada documento
- Versionado de documentos
- Limpieza periódica de documentos

Interacciones:

API Pública: Provee endpoints RESTful para interactuar con otros servicios

BBDD: Utiliza una base de datos para almacenar y proveer de información.

Microservicio Certificación con blockchain

Propósito y Responsabilidades: El microservicio de blockchain tiene como objetivo proporcionar una interfaz segura y eficiente para interactuar con una cadena de bloques, permitiendo realizar transacciones, verificar datos y consultar registros de manera confiable y auditable. Este microservicio se utilizará como apoyo certificador de la trazabilidad guardada en nuestro sistema.

Las funciones a implementar serían:

- Registro Transacciones: Registrar en blockchain a través de transacciones.
- Verificación de Datos: Verificar la autenticidad e integridad de los datos en la blockchain.
- Consulta de Registros: Consultar y obtener información almacenada en la blockchain.
- Gestión de Contratos Inteligentes: Desplegar y gestionar contratos inteligentes (smart contracts).
- Integración de Eventos: Escuchar/Responder a eventos emitidos por blockchain.

Interacciones:

API Pública: Provee endpoints RESTful para interactuar con aplicaciones cliente y otros microservicios.

Blockchain: Interactúa con una red blockchain **Ethereum** para registrar y verificar transacciones.

BBDD: Almacena metadatos y referencias a las transacciones en blockchain.

2.1.3. Visualización**Microservicio Consulta de residuos**

Propósito y responsabilidades: El microservicio consulta de residuos tiene como objetivo proveer de información detallada sobre el estado y ciclo de vida de los residuos. Tiene la responsabilidad de obtener información del estado de un residuo y de consultar el ciclo del residuo.

Las funciones a implementar serían:

- Realizar consultas sobre el estado actual del residuo
- Consultar historial del ciclo de vida del residuo

Interacciones:

API Pública: Proveer endpoints RESTful para interactuar con aplicaciones cliente y otros microservicios.

Fuentes de Datos: Conexión a sistemas de recolección de residuos, bases de datos.

BBDD: Almacenar datos procesados y metadatos relacionados con los residuos.

Microservicio Consulta de Situación Contenedores

Propósito y Responsabilidades: Microservicio para la consulta en TRAZA de la situación en tiempo real de los contenedores gestionados por SOGAMA.

Algunas de las funciones a implementar serían:

- Realizar consultas sobre el estado actual del contenedor
- Consultar historial del ciclo de vida del contenedor
- Situación del contenedor

Interacciones:

API Pública: Proveer endpoints RESTful para interactuar con aplicaciones cliente y otros microservicios.

BBDD: Almacenar datos procesados y metadatos relacionados con viajes/contenedores.

Microservicio Cuadro de mando estadístico

Propósito y Responsabilidades: Este microservicio está orientado a la gestión empresarial del proceso y que permitirá medir la evolución y los resultados, desde una perspectiva estratégica y global.

Se tratará de un conjunto de indicadores que aportan información inteligente de forma simplificada a SOGAMA. Los indicadores que conformarán el Cuadro de Mando permitirán evaluar la gestión y eficacia del proceso, detectar amenazas y oportunidades, descubrir posibles incidencias y actuar en consecuencia.

Un Cuadro de Mando está configurado por KPIs acompañados de una representación gráfica, de esta forma se puede acceder a la información de manera muy visual y ágilmente. Este microservicio permitirá optimizar los procesos de toma de decisiones tanto estratégicas como tácticas.

Interacciones:

API Pública: Proveer endpoints RESTful para interactuar con aplicaciones cliente y otros microservicios.

BBDD: Almacenar datos recopilados y procesados, así como configuraciones de informes y dashboards.

Otros Microservicios: Recibir y enviar datos a otros microservicios para procesamiento y análisis.

2.1.4. Interoperabilidad

Microservicio Integración GAIA (DIs y NTs)

Propósito y responsabilidades: El microservicio de Integración con Gaia se encarga de todas las operaciones de integración con la plataforma centralizada de información ambiental en materia de residuos a nivel autonómico.

Esto incluye la generación, actualización y anulación de DIs y NTs e integración con los demás servicios que Gaia Residuos soporte en un futuro (actualmente no existe un servicio para transfronterizos, ni productos finales (energía, compost, etc), aunque existe previsión de desarrollo para ello en la plataforma autonómica.

Interacciones:

API Pública: Provee endpoints RESTful para interactuar con otros servicios y aplicaciones cliente Gaia.

BBDD: Utiliza una base de datos para almacenar información de integración con Gaia.

Microservicio Integración Sistemas Terceros

Propósito y responsabilidades: Este microservicio tiene como propósito abarcar las funcionalidades de integración con cualquier ente externo como MS DYNAMICS 365 BUSINESS CENTRAL, portal del cliente, sistema de pesaje (servipesa) o cualquier otro sistema de forma bidireccional.

Interacciones:

API Pública: Provee endpoints RESTful para interactuar con otros servicios y aplicaciones.

Microservicio Administración de HW e IoT

Propósito y responsabilidades: El microservicio de administración de HW e IoT tiene como objetivo centralizar la gestión, monitorización, y control de dispositivos de hardware y IoT, permitiendo una supervisión eficiente y facilitando la integración con otros sistemas.

Registro y Administración de Dispositivos: Alta, baja y actualización de dispositivos IoT y hardware (básculas, lectores matrículas, lectores rfid, tablets, etc).

Interacciones:

API Pública: Provee endpoints RESTful para interactuar con aplicaciones cliente y otros microservicios.

BBDD: Se almacena la configuración, estado y datos de telemetría de dispositivos.

Mensajería: Utilizará un sistema de mensajería Kafka para comunicación con los dispositivos y gestión de eventos.

Microservicio Generación de Informes y Dashboards

Propósito y responsabilidades: El microservicio de generación de informes y dashboards tiene como objetivo proporcionar una plataforma que centraliza la recolección, procesamiento y visualización de datos, permitiendo a los usuarios crear, publicar informes (PDF, CSV, ...) y dashboards interactivos que muestren datos en tiempo real detallados basados en datos de la aplicación.

Interacciones:

API Pública: Provee endpoints RESTful para interactuar con aplicaciones cliente y otros microservicios.

Fuentes de Datos: Conexión a BBDD, APIs y otros microservicios para recolectar datos.

BBDD: Datos configuraciones de informes, dashboards y metadatos.

2.2. Automatización de Procesos

De los procesos descritos en el pliego se han identificado las siguientes acciones:

Entrada y Salida de Camiones y Contenedores

Definición del Flujo: En la entrada y salida de los contenedores de las instalaciones detalladas en el pliego, incorporaremos un lector de matrículas para identificar el camión y un lector RFID para identificar el contenedor. Estos dispositivos se incorporarán en el mismo lugar que la báscula existente en cada instalación, a excepción de las instalaciones de ADIF, las cuales se instalarán en el mejor lugar después de realizar un estudio de las 3 terminales.

En el caso del CMC que tiene una misma entrada y salida, tiene 2 básculas y puede utilizarse de forma indistinta para entrar y salir, se colocará en cada una un lector de matrículas y un lector rfid. Mediante software detectaremos si el camión está entrando o saliendo de las instalaciones mediante peso y si el camión ya está dentro de las instalaciones o no.

Capturaremos la información enviada por los dispositivos instalados mediante un desarrollo a medida (**1.2.5. Middleware**) que notificará los eventos existentes a TRAZA. La información obtenida, en conjunto con la predictiva generará en la Tablet u móvil asociado el contenido de la entrada o salida. Esta información tendrá que ser confirmada por la persona involucrada. En el caso de que la información no sea acorde a la validación del sistema sobre el origen o tipo de basura, obligaremos a una validación de un ente superior.

Dispositivo IoT: Lector de matrículas, lector RFID, etiquetas RFID, Tablet.

Automatización: Lectura de matrículas, lectura contenedores, lectura de báscula, envío de información de viaje, integración con Tablet, detección de desviaciones de datos(IA).

Notificación Estado Contenedor

Definición del Flujo: Tenemos principalmente 3 automatizaciones destacables:

- Cambio de contenedor en Tolva: Durante el flujo en las plantas de transferencia, será necesario identificar que contenedor está en las tolvas. Colocaremos un lector rfid para ello. Así no perderemos la trazabilidad del residuo.
- Selección de tolva: En las plantas de transferencia con más de una tolva, mediante los lectores rfid identificaremos en que tolva descarga la basura para su compresión el camión.
- Notificación empresa de transporte: A partir de los pesos de cada camión podremos indicar en Traza el estado del contenedor en tolva y también notificar cuántos y cuáles son los contenedores que están llenos.

Dispositivo IoT: Lector rfid, etiquetas RFID, Tablet

Registrar Notificaciones previas y traslados de residuos

Definición del Flujo: Cuando un residuo es trasladado, es necesario la generación de un documento de traslado (DIs) y si procede una notificación previa(NTs). A continuación, detallamos la automatización:

- **NTs:** En el caso de las Notificaciones previas de traslado se generarán en TRAZA siempre y cuando no exista ya una. Las NTs no pueden durar más de tres años, ni excederse de la cantidad de residuo indicada. La creación de la NT será notificado a Gaia a través del servicio web expuesto.

A continuación, se muestra parte de un fichero e3L de Nt integrado con Gaia:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes">
<?e3l:schemaVersion="3.5" xmlns:e3l="http://www.e3l.es/3.0/e3l">
<header>
<test/N/test>
<prepared>2024-05-23T03:16:00.147</prepared>
<sender>
<IDIF:80038096</IDIF>
<center centerCode="150033410" centerDenomination="DESQUACES PEPIÑO"/>
</sender>
</header>
<waste>
<wasteNT NTCode="NT129999999992024275416" NTYear="2024-01-01" NTDate="2024-05-07+02:00" NTStartDate="2024-05-18+02:00" NTEndDate="2027-05-18+02:00" NTFrequency="DIARIA" NTRegional="B" NTEmergency="W"
<NTTransferOperatorData>
<entityId>
<nationalId>870038096</nationalId>
</entityId>
<entityType entityTypeCode="2"/>
<entityName>
<reason>
<reasonName>DESQUACES PEPIÑO</reasonName>
<reasonAssociation associationCode="18" associationDescription="Sociedades de responsabilidad limitada"/>
</reason>
</entityName>
<entityCenter>
<centerId centerCode="150033410" centerDenomination="DESQUACES PEPIÑO"/>
<centerEconomicActivity>W/A</centerEconomicActivity>
<centerAddress>
<spanishAddress>
<vial vialCode="999" vialDescription="Via pública"/>
<address>CTRA. AC-523. P.K. 2.700 A BREA (ARDEMLI)</address>
<CP>15059</CP>
<locality municipalityCode="15059" municipalityName="Ordas"/>
<province provinceCode="15" provinceName="Coruña (A)"/>
<CA CACode="12" CAName="Galicia"/>
<country countryCode="724" countryName="ESPAÑA"/>
</spanishAddress>
</centerAddress>
<centerContact>
<phone>999999495</phone>
<fax>000000000</fax>
<mail>info@desquacespepiño.com</mail>
</centerContact>
<wasteCenterAuthorization>
<authorizationId>
<authorizationIdNumber>12001150003341000</authorizationIdNumber>
</authorizationId>
<authorizationCode>001</authorizationCode>
</wasteCenterAuthorization>
</entityCenter>
<wasteTransferOperatorType operatorTypeCode="001"/>
</NTTransferOperatorData>
<NTProductData>
```

- **DIs:** Los documentos de traslado pueden o no estar vinculados a NTs,

la decisión está vinculada al tipo de residuo que se traslada, Traza automatizará la generación de los DIs para cada transporte/salida de planta de transferencia con residuos.

Traza generará de forma automática a través de un demonio los DIs de las EELL a partir de los datos históricos (De 1 a 3 meses de vigencia). En el momento de cierre se actualizará el peso real del residuo transportado y se notificará a Gaia.

Traza generará los documentos de **traslados transfronterizos**, aunque actualmente en Gaia no existe este servicio web (previsto para nuevos desarrollos dentro del proyecto), aunque la plataforma estará preparada para esta integración.

A continuación, se muestra parte un DI (no peligroso) de un fichero e3L integrado con Gaia:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes">
<?e3l:schemaVersion="3.5" xmlns:e3l="http://www.e3l.es/3.0/e3l">
<header>
<test/N/test>
<prepared>2024-05-23T03:17:27.961</prepared>
<sender>
<IDIF:80038096</IDIF>
<center centerCode="150033410" centerDenomination="DESQUACES PEPIÑO"/>
</sender>
</header>
<waste>
<wasteDCS DCSCode="DCS1299999999920242038935" DCSPhase="D" DCSYear="2024-01-01" DCSDate="2024-04-04+02:00" DCSStartDate="2024-04-04+02:00" DCSEndDate="2024-04-04+02:00" DCSRegional="B" DCSSendtoOrigin="W"
<DCSTransferOperatorData>
<entityId>
<nationalId>870038096</nationalId>
</entityId>
<entityType entityTypeCode="2"/>
<entityName>
<reason>
<reasonName>DESQUACES 24 HORAS SL</reasonName>
<reasonAssociation associationCode="18" associationDescription="Sociedades de responsabilidad limitada"/>
</reason>
</entityName>
<entityCenter>
<centerId centerCode="150033410" centerDenomination="DESQUACES PEPIÑO"/>
<centerEconomicActivity>W/A</centerEconomicActivity>
<centerAddress>
<spanishAddress>
<vial vialCode="999" vialDescription="Via pública"/>
<address>CTRA. AC-523. P.K. 2.700 A BREA (ARDEMLI)</address>
<CP>15059</CP>
<locality municipalityCode="15059" municipalityName="Ordas"/>
<province provinceCode="15" provinceName="Coruña (A)"/>
<CA CACode="12" CAName="Galicia"/>
<country countryCode="724" countryName="ESPAÑA"/>
</spanishAddress>
</centerAddress>
<centerContact>
<phone>999999495</phone>
<fax>000000000</fax>
<mail>info@desquaces24horas.com</mail>
</centerContact>
<wasteCenterAuthorization>
<authorizationId>
<authorizationIdNumber>12001150003341000</authorizationIdNumber>
</authorizationId>
<authorizationCode>001</authorizationCode>
</wasteCenterAuthorization>
</entityCenter>
<wasteTransferOperatorType operatorTypeCode="001"/>
</DCSTransferOperatorData>
<DCSProductData>
```

- **Producto resultante:** El producto resultante (energía generada, compost, etc) como se indica en el pliego se debe de registrar en TRAZA, pero ade-

más debe de ser notificada a Gaia. Actualmente no existe en Gaia ese servicio web (previsto en la planificación para nuevos desarrollos de la plataforma Gaia), de todos modos, TRAZA estará preparada para esta integración.

Automatización de datos de integraciones

Definición del Flujo: Se crearán procesos automáticos para transferir, transformar y sincronizar datos entre Traza y los sistemas de pesaje(báscula), aplicación de PClass y aplicación de PRTEM-PCM.

Automatización Tren

Definición del Flujo: En la entrada de las tres plantas de ADIF, si esta nos lo permite, colocaremos un lector de matrícula y lector RFID para no perder la trazabilidad. A partir de los datos recogidos, en la Tablet, automatizaremos la posibilidad de indicar en que tren va cada contenedor.

Dispositivo IoT: Lector rfid, etiquetas RFID, Tablet

Automatización Cintas Transportadoras

Definición del Flujo: Las cintas transportadoras que salen de PCLAS hacia PRTEM, se supone que con la integración del ERP del gestor de la PCLAS, ya tendremos la identificación del residuo, pero las balas que van por la cinta deberán ser identificadas con una etiqueta, o similar para mantener la trazabilidad.

La cinta transportadora de PRTEM-PCM hacia la PTE en forma de combustible CDR, deberá tener una gestión similar.

Automatización Avisos y Alertas Automáticos

Definición del Flujo: Cuando llegue o vaya a llegar un camión que no está en el maestro para ese origen o tipo de residuo, se generarán alertas y también cuando ya esté en la instalación.

Generaremos avisos para que se autoricen o no los traslados a PTs que no son correspondidas.

Generaremos avisos para que ADIF autorice la salida de residuo una vez lleno el tren.

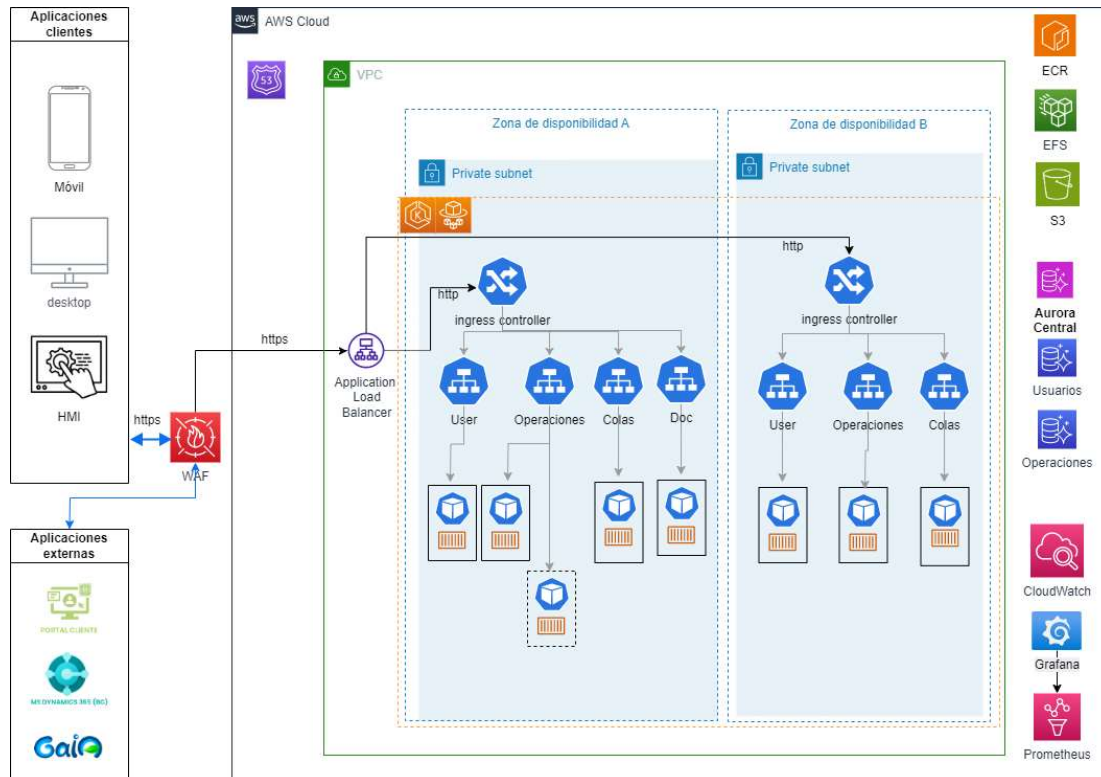
Nota: Como se indica en pliego, todo proceso estará respaldado con la posibilidad de introducir los datos de forma manual.

2.3. Diseño de la plataforma

2.3.1. Descripción general de la Arquitectura

Las aplicaciones que forman parte de la solución TRAZA serán construidas como micro-servicios asociados a contenedores y serán desplegadas en un clúster de Kubernetes

como orquestador con el objetivo de permitir una gran automatización, escalabilidad, portabilidad y resiliencia de la solución de la aplicación.



La selección de Kubernetes se fundamenta en:

- Orquestación de contenedores:
 - Nos permite una gestión automática de la implementación, gestión, escalado y operación con los contenedores.
 - TRAZA aprovecha Kubernetes proporcionando la resiliencia necesaria para permitir reiniciar aquellos contenedores que fallen de forma automática, evitando tener caídas de servicio.
- Escalabilidad:
 - Kubernetes permite escalar aplicaciones de forma rápida, sencilla y automática, agregando o eliminando instancias de los microservicios de la aplicación.
 - Distribuye de forma automática el tráfico entre las diferentes instancias de los microservicios.
- Kubernetes soporta despliegues progresivos (Rolling updates, canary deployments y rollbacks automáticos) facilitando la entrega continua y la implementación de nuevas versiones con el mínimo tiempo de inactividad de la aplicación.
- Kubernetes Proporciona mecanismos de tolerancia a fallos para que los microservicios sigan funcionando en caso de errores.

- Kubernetes facilita la portabilidad de la aplicación entre diferentes entornos cloud y on-premise.

Cada microservicio y el portal web será desplegado en un contenedor.



Cada contenedor será desplegado en un POD de Kubernetes.



Cada microservicio tendrá su propia configuración para escalar de forma horizontal según el consumo de memoria y/o CPU que tenga el contenedor del microservicio.

TRAZA aprovecha el servicio de descubrimiento que Kubernetes ofrece y que permite a los microservicios encontrarse y comunicarse entre ellos, haciendo transparente la necesidad de conocer previamente las ubicaciones de los demás microservicios. Estos servicios escalan de forma dinámica y automática sin la necesidad de la intervención de los administradores del sistema.

Al ser Kubernetes un clúster cerrado, no permite el acceso externo directo a TRAZA. Este acceso se gestiona a través del Ingress Controller. Esta forma de gestionar el acceso externo a los microservicios instalados utiliza un elemento de red que admite peticiones HTTP y/o HTTPS, actuando como un enrutador que redirige el tráfico a los servicios de los microservicios, garantizando así la seguridad de la solución propuesta TRAZA.

Las bases de datos de los microservicios estarán ubicadas fuera del clúster de Kubernetes para obtener un rendimiento óptimo, aprovechando el hardware dedicado a ellas. Para TRAZA esto asegura un rendimiento óptimo y alta disponibilidad, además de reducir la carga operativa de las bases de datos. Al tener las bases de datos fuera del clúster, se facilitan las tareas de mantenimiento y copias de seguridad.

La base de datos de cada microservicio y la base de datos principal estarán instaladas en diferentes servidores con diferentes configuraciones dependiendo de la demanda que vaya a tener. Esto nos va a permitir que los microservicios sean totalmente independientes mejorando la disponibilidad y la resiliencia de los microservicios y su base de datos.

El clúster de Kubernetes se desplegará en Amazon Web Services (AWS) utilizando Elastic Kubernetes Service con Fargate (EKS + Fargate). El servicio de AWS es 100% compatible con el estándar de Kubernetes definido por la Cloud Native Computing Foundation. Al utilizar Fargate, cada pod se despliega en un nodo independiente, lo que beneficia el rendimiento de la aplicación al evitar la competencia de los pods por los recursos de los nodos.

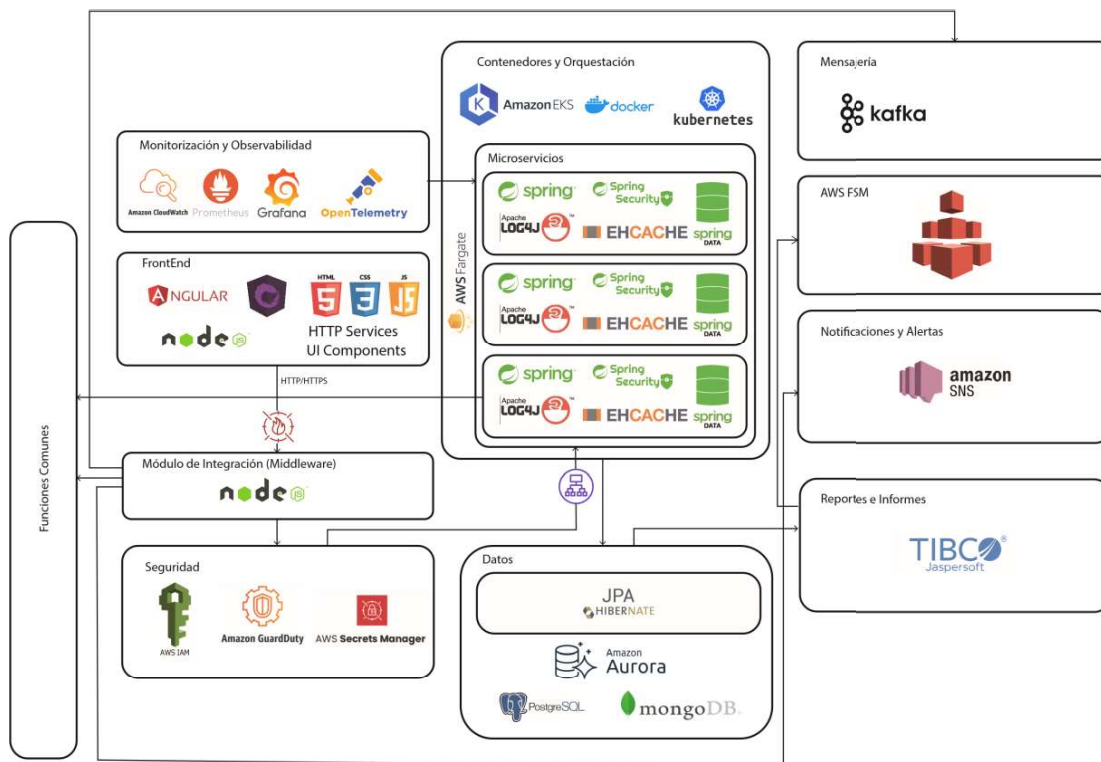
Los informes y/o dashboards serán generados mediante un microservicio que se encargará de la obtención de los datos necesarios para su creación.

TRAZA realizará el almacenamiento de documentos o imágenes de forma permanente en S3 utilizando el ciclo de vida para su optimización de costes. Los documentos e imágenes se almacenarán en S3, mientras que sus metadatos se almacenarán en la base de datos.

Al archivar los metadatos de los documentos e imágenes en una base de datos nos permite tener un registro cronológico de cada documento y sus posibles modificaciones. TRAZA implementará políticas de acceso y control de versiones en los documentos para asegurar la integridad y seguridad de los datos.

Para su uso compartido de los documentos entre los microservicios y antes de su archivo en S3, el almacenamiento se realizará en un EFS compartido con todo el clúster de Kubernetes por medio de un objeto de tipo Persistent Volume. TRAZA proporciona así un almacenamiento compartido entre todos los microservicios en tiempo real, accesible desde todos los microservicios y pensado para el almacenamiento temporal de los documentos.

2.3.2. Tecnologías de Desarrollo / Capas del sistema



A continuación, se detallan algunas de las tecnologías:

- **Angular:** Angular es un framework de desarrollo frontend basado en TypeScript. Usado para el desarrollo de la interfaz de usuario (UI) de la aplicación debido a que proporciona una experiencia de usuario rica y dinámica.

- **HTML, CSS, JavaScript:** HTML, CSS y JavaScript son tecnologías web fundamentales para el desarrollo frontend. Usadas para estructurar y diseñar la interfaz de usuario debido a que crean una UI responsiva y funcional.
- **Node.js:** Node.js es un entorno de ejecución de JavaScript en el servidor. Usado para servir la aplicación frontend y manejar solicitudes HTTP/HTTPS debido a que permite un desarrollo rápido y eficiente del backend para aplicaciones web.
- **Apache Kafka:** Apache Kafka es una plataforma de mensajería distribuida. Usada para la mensajería asíncrona entre microservicios debido a que facilita la comunicación eficiente y desacoplada entre servicios mediante eventos.
- **AWS EKS (Elastic Kubernetes Service):** AWS EKS es un servicio gestionado de Kubernetes en AWS. Usado para la orquestación de contenedores debido a que gestiona el despliegue, la escalabilidad y la operación de aplicaciones en contenedores.
- **Docker:** Docker es una plataforma de contenedorización. Usado para empaquetar aplicaciones y sus dependencias en contenedores debido a que asegura la portabilidad y consistencia del entorno de ejecución.
- **Spring Boot:** Spring Boot es un framework para el desarrollo rápido de aplicaciones Java. Usado para el desarrollo de microservicios debido a que proporciona una plataforma robusta y escalable para construir microservicios.
- **Spring Security:** Spring Security es un framework de seguridad para aplicaciones Java. Usado para la seguridad a nivel de aplicación y microservicios debido a que asegura que la aplicación esté protegida contra accesos no autorizados.
- **EHCACHE:** EHCACHE es una biblioteca de caching para Java. Usado para mejorar el rendimiento y la respuesta de los microservicios debido a que reduce la latencia y aumenta la eficiencia mediante el almacenamiento en caché de datos frecuentemente accedidos.
- **Log4J:** Log4J es un framework de logging para Java. Usado para el registro de actividades y errores en los microservicios debido a que facilita el monitoreo y la depuración de la aplicación.
- **Spring Data:** Spring Data es un framework para facilitar el acceso a bases de datos en aplicaciones Java. Usado para la integración con bases de datos y repositorios de datos debido a que simplifica la interacción con bases de datos relacionales y no relacionales.
- **Spring Cloud:** Spring Cloud es un conjunto de herramientas para desarrollar y desplegar aplicaciones distribuidas en la nube. Usado para el desarrollo y despliegue de microservicios en la nube debido a que facilita la construcción de aplicaciones resilientes y escalables en un entorno de nube.

- **Amazon SNS (Simple Notification Service):** Amazon SNS es un servicio de notificaciones de AWS. Usado para el envío de notificaciones y alertas a diferentes suscriptores debido a que mantiene a los usuarios informados mediante la distribución de mensajes en tiempo real.
- **TIBCO JasperSoft:** TIBCO JasperSoft es una plataforma de informes y analítica. Usado para la generación y gestión de reportes de datos debido a que proporciona herramientas para la creación de informes detallados y análisis de datos.
- **Amazon Redshift:** Amazon Redshift es un almacén de datos en la nube para análisis. Usado para el almacenamiento y análisis de grandes volúmenes de datos debido a que facilita el análisis de datos y la generación de informes a gran escala.
- **MongoDB:** MongoDB es una base de datos no relacional orientada a documentos. Usado para el almacenamiento de datos no estructurados debido a que maneja datos que no se ajustan bien a un esquema relacional.
- **Middleware:** Middleware es software que conecta diferentes aplicaciones y servicios, permitiendo la comunicación y gestión de datos entre ellos. Usado para la integración y comunicación eficiente entre los componentes de la aplicación debido a que facilita la integración y comunicación eficiente entre los componentes de la aplicación.

2.3.3. Integración y Comunicación

Los microservicios se comunicarán entre ellos para obtener o procesar información solicitada por el usuario. Dicha comunicación entre los microservicios será:

- Síncrona, mediante peticiones http entre los distintos microservicios de la aplicación
- Asíncrona, mediante el uso de colas de mensajes

TRAZA mediante el uso colas para la comunicación entre los microservicios asegurará la correcta práctica segura y eficiente del flujo de información entre los componentes. Las colas nos permiten que los microservicios manejen flujos de trabajo asíncronos y distribuyan la carga de manera más equilibrada. Las colas que se utilizarán en el proyecto son de tipo FIFO (First In, First Out).

El uso de colas con los microservicios nos proporciona los siguientes beneficios:

- Separar la lógica de generación de tareas (productores) y la lógica de procesamiento de las tareas (consumidores)
- Facilita el escalado independiente de productores y consumidores según la demanda.
- Mejora la tolerancia a fallos. Las tareas pueden ser procesadas por cualquier consumidor disponible.

TRAZA mantendrá las colas dentro de contenedores POD:

- Nos permite tener un control total sobre la configuración, el ajuste del rendimiento y la integración con los servicios.
- Optimizar el rendimiento ajustando el número de consumidores y la persistencia de los mensajes.
- Tener un entorno más homogéneo teniendo la mayoría de los servicios dentro del clúster de Kubernetes y facilidad de mantener al tener todos los elementos en el mismo entorno.
- Esta implementación de colas propias mediante Kafka, crea una “No dependencia” directa de servicios AWS que condicionen la relación con el proveedor cloud, y permitiendo despliegues más rápidos, además de decisiones estratégicas futuras de en decisión de utilización de una u otra nube.

2.3.4. Seguridad

2.3.4.1 Usuarios

La autenticación de los usuarios (proceso de verificar quien es el usuario) en la aplicación se realizarán mediante los estándares de OpenID Connect para gestionar el acceso seguro a los microservicios.

La autorización de los usuarios (permiso para acceder a los microservicios) se basará en roles. Cada rol tendrá un conjunto predefinido de permisos asociados a determinar qué acciones puede realizar un usuario que tenga ese rol. En la aplicación existirán los siguientes roles:

- El rol de administrador que tendrá acceso a todas las funcionalidades de la aplicación.
- Roles con funcionalidades específicas por operación, por ejemplo, el rol de conductor que tendrá las funcionalidades relacionadas con sus acciones.

Un usuario podrá tener varios roles porque cada role es independiente del otro.

2.3.4.2 Aplicaciones

El acceso a los microservicios por parte del usuario se realizará por medio de un token de tipo Jwttoken, que será validado en cada microservicio. Este token es generado por el OpenId.

Las peticiones que se realicen desde la aplicación móvil, los dispositivos HMI y el portal web se realizará mediante peticiones https, utilizando TLS 1.3 para asegurar la comunicación entre los microservicios y los clientes.

La comunicación entre los microservicios será vía http. Estas peticiones son http porque están dentro de la red del cluster de Kubernetes y sin acceso a Internet.

Las conexiones con la base de datos se realizarán con usuarios con permisos limitados (lectura y escritura)

2.3.4.3 Cluster de Kubernetes

La seguridad en Amazon EKS implica múltiples capas y la integración de diversas herramientas y prácticas de AWS y Kubernetes. Implementa una combinación de controles de acceso, encriptación, monitoreo y buenas prácticas que nos ayudan a proteger la aplicación y los datos.

El acceso al clúster de Kubernetes (administración) se realiza mediante el servicio IAM de AWS.

Los pods tendrán asociado un Role de IAM para asumir permisos específicos, mitigando el alcance de los permisos, para acceder a otros servicios de AWS.

Al utilizar EKS con Fargate, AWS se encarga de mantener todos los nodos del clúster actualizados con los últimos parches de seguridad y actualizaciones del sistema operativo.

Kubernetes nos permite controlar el tráfico de red interno en el clúster de Kubernetes por medio de políticas de red.

Los microservicios desplegados en Pods puede configurar contextos de seguridad que especifica el usuario o el grupo en el que el microservicio trabaja.

La configuración de los datos delicados de los microservicios, como las contraseñas de las cadenas de conexión a bases de datos, se almacena en AWS Secret Manager o en AWS Systems Manager Parameter Store. Estos servicios de AWS almacenan dicha información encriptada, garantizando su seguridad.

Se configurará el servicio GuardDuty para la detección de amenazas y monitoreo de actividades sospechosas en nuestra aplicación. Este servicio no es propio de Kubernetes ni de Eks, pero si es compatible con ellos.

Las imágenes de los contenedores se almacenan en repositorios de ECR que permite escanear las imágenes de los contenedores en búsqueda de vulnerabilidades antes de que sean desplegados en el cluster de EKS. La imagen es escaneada después de subir la imagen al repositorio.

2.4. Escalabilidad de la solución

2.4.1. Estrategias de escalabilidad

2.4.1.1 Microservicios

Los microservicios están desplegados en Pods dentro del clúster de Kubernetes. Estos Pods pueden ser configurados para indicar la cantidad de réplicas máximas y mínimas del Pod según la carga de trabajo. La configuración del escalado se basa en el uso de memoria y CPU que este ejecutando el Pod. Kubernetes realiza el escalado automático de los Pods, sin la necesidad de monitorear la carga del Pod (uso de memoria y CPU) de forma manual.

2.4.1.2 Bases de datos

La aplicación utilizará el servicio de Aurora Postgresql Serverless, que es administrado por Amazon Web Services. Este servicio tiene la capacidad de escalar tanto horizontalmente como verticalmente de manera eficiente.

- El escalado horizontal implica agregar más nodos, que son réplicas de lectura, para distribuir la carga de trabajo, mejorar la disponibilidad y aumentar la capacidad de manejo de consultas.
 - El servidor se ajusta automáticamente la capacidad según las necesidades de la aplicación, eliminando la necesidad de gestionar manualmente los nodos.
 - Aurora Serverless mide la capacidad en unidades llamadas Aurora Capacity Units (ACU). La base de datos escala automáticamente las ACU hacia arriba o hacia abajo en respuesta a las fluctuaciones en la carga de trabajo.
- El escalado vertical implica aumentar los recursos de hardware asignados a una instancia de base de datos, como CPU, memoria y almacenamiento, para mejorar el rendimiento y la capacidad de carga. Gracias a que el escalado propuesto es bastante rápido, el periodo de inactividad o rendimiento reducido mientras se realiza el escalado es minimizado significativamente o imperceptible.

2.4.1.3 Cluster de kubernetes

Amazon EKS con Fargate permite ejecutar Pods de Kubernetes sin gestionar las instancias de servidor sin tener que utilizar los nodos EC2. Esto facilita el proceso de escalado, ya que AWS gestiona la infraestructura subyacente. Fargate ajusta automáticamente los recursos necesarios (memoria y CPU) para ejecutar los Pods configurados basándose en las necesidades de la aplicación.

2.4.2. Tolerancia a Fallos y Recuperación

La tolerancia a fallos y la recuperación son aspectos críticos en la gestión de los clúster de Kubernetes. Este está diseñado para ser resiliente y manejar fallos automáticamente. Una

configuración adecuada y el uso de las buenas prácticas van a ser esenciales para garantizar una alta disponibilidad y rápida recuperación.

2.4.2.1 Tolerancia a fallos

Kubernetes configura los Deployments de los microservicios con Replicasets para asegurar la cantidad mínima de pods en ejecución. Así, si un pod falla, Kubernetes iniciará automáticamente nuevos pods en pocos segundos para mantener la cantidad deseada.

Kubernetes también permite un auto escalado de forma automática:

- Horizontalmente aumentando o disminuyendo el número de réplicas de un pod en función de la carga de CPU, memoria u otras métricas que personalizemos.
- Verticalmente ajustando los recursos asignados a los pod (CPU y memoria) según las necesidades del microservicio instalado en él.

2.4.2.2 Recuperación de fallos

Kubernetes dispone de dos operaciones para comprobar el estado de los Pods:

- Liveness probes: Verifican si los Pods están vivos. Si fallan, se reinicia el Pod.
- Readiness probes: Verifican si los Pods están listos para recibir tráfico, asegurando que sean saludables para recibir solicitudes.

Kubernetes nos permite utilizar varias estrategias de despliegue, pero nosotros utilizaremos la Canary Deployments que nos permite desplegar nuevas versiones a un subconjunto pequeño de usuarios antes de un despliegue completo, permitiendo detectar problemas antes de afectar al resto de usuarios.

2.4.3. Monitorización

La monitorización de los microservicios en EKS con Fargate es crucial debido a la naturaleza dinámica y distribuida de las aplicaciones. Monitorizaremos los microservicios por:

- Estos están distribuidos en múltiples contenedores y nodos para una adecuada monitorización y tener visibilidad completa sobre el estado y el rendimiento de cada servicio.
- EKS gestiona los contenedores de Kubernetes, mientras que Fargate proporciona capacidad de ejecución sin necesidad de gestionar servidores, permitiendo entender mejor el comportamiento de las aplicaciones.
- La integración de la monitorización con gestión de incidencias y problemas ayuda a identificar y resolver la causa raíz rápidamente, minimizando el tiempo de inactividad y mejorando la experiencia de usuario.
- La monitorización de los microservicios nos permite disponer de la información para identificar cuellos de botella y áreas de mejora de rendimiento de la aplicación.
- El uso de métricas de rendimiento permite ajustar la gestión de demanda y recursos eficientemente, evitando el sobre aprovisionamiento o el uso insuficiente.

- Facilitan el seguimiento del comportamiento de los servicios cuando escalan, asegurando que las aplicaciones mantengan un rendimiento óptimo.
- La monitorización puede ayudar a mantener la resiliencia del sistema mediante la identificación de patrones y tendencias que podrían llevar a fallos.
- La monitorización puede ayudar a cumplir con los requisitos de seguridad y regulaciones al proporcionar visibilidad sobre el acceso y el uso de los servicios.
- Detectar actividad inusuales o potencialmente maliciosas en el entorno de microservicios.

Para realizar esta monitorización se usará el estándar de OpenTelemetry. El operador de OpenTelemetry se desplegará en el clúster de EKS para gestionar la instrumentación automática de los microservicios. Se configurarán los exportadores de OpenTelemetry para enviar datos al sistema de backend CloudWatch, y/o Prometheus y Grafana. Utilizaremos las librerías de OpenTelemetry para instrumentar los microservicios, recolectando las trazas, métricas y logs, y se usarán las herramientas de monitoreo para analizar los datos recolectados, identificar patrones y solucionar los posibles problemas.

Con la herramienta CloudWatch monitorizaremos:

- CloudWatch Logs:
 - Configuraremos los logs de los contenedores de Kubernetes para que se envíen a CloudWatch Logs.
 - Utilizaremos los agentes de logs de CloudWatch para recolectar logs de aplicaciones y del sistema.
 - Activaremos los logs de Aurora para PostgreSQL para que sean almacenados en esta herramienta.
- CloudWatch Metrics:
 - Configuraremos la recopilación de métricas del clúster de EKS, incluyendo las métricas de los nodos, pods y contenedores.
 - Con CloudWatch Container Insights para obtener métricas detalladas sobre la infraestructura de contenedores.
 - Activaremos las métricas de Aurora for PostgreSQL para que sean almacenadas y monitorizadas en esta herramienta.
- CloudWatch Alarm:
 - Se configurarán alarmas según las métricas clave para alertarnos sobre problemas potenciales como el uso de cpu o memoria.
 - Se configurarán alarmas según las métricas de Aurora for PostgreSQL para alertarnos sobre problemas de memoria, CPU, espacio ocupado, bloqueos,

Con Prometheus podemos recopilar todas las métricas de los microservicios y con Grafana podremos crear poderosos dashboards de los microservicios de forma individual o de forma colectiva.

- Prometheus:
 - Desplegaremos el operador de Prometheus dentro del clúster de EKS para gestionar la configuración y el despliegue.
 - Utilizaremos el exportador para recolectar las métricas de los nodos y el cAdvisor para las métricas de los contenedores.
- Grafana
 - Configuraremos los dashboards en Grafana para visualizar métricas recolectadas por Prometheus.
 - Utilizaremos dashboards predefinidos para Kubernetes y adaptar esto a nuestras necesidades específicas.

Con la herramienta AWS X-Ray analizaremos las trazas de los microservicios:

- Instrumentaremos los microservicios para enviar datos de las trazas a X-Ray.
- Se configurará el demonio de X-Ray en los contenedores para recolectar y enviar datos de las trazas.

Con OpenTelemetry unificamos la observabilidad.

- Desplegaremos el OpenTelemetry Collector en el clúster para recolectar y exportar datos a sistemas de backend como Prometheus, CloudWatch y X-Ray
- Configuraremos los exportadores para enviar los datos de observabilidad de los microservicios a los servicios de monitoreo y análisis descritos anteriormente.

Con AWS GuardDuty monitorearemos y detectaremos amenazas y actividades maliciosas en el cluster de EKS.

2.5. Middleware

Este componente (que será desplegado en la misma infraestructura detallada en los apartados anteriores) es un desarrollo a medida y sus **objetivos** son:

- Crear un sistema que procese imágenes capturadas por una cámara IP para identificar matrículas vehiculares.
- Integrar un lector RFID para la identificación y gestión de etiquetas RFID.
- Enviar los datos recopilados a una API para su procesamiento y almacenamiento.
- Integración con la plataforma TRAZA asegurando la compatibilidad y correcta comunicación

Tecnologías: Python, Django, Django Rest Framework, PostgreSQL.

Librerías para Integración de Cámara IP y Lector RFID:

- **OpenALPR:** Una herramienta de código abierto especializada en el reconocimiento automático de matrículas (ANPR).
 - **Ventajas:** Facilidad de uso; Alto rendimiento y precisión; Soporta múltiples plataformas y lenguajes de programación.
- **Librerías de RFID:** Librerías específicas de interacción con hardware RFID, como pyrfid o controladores específicos proporcionados por el fabricante del lector RFID.

2.6. Hardware

En este apartado detallamos el hardware seleccionado que debe ser considerado como una propuesta inicial, la cual se validará con el cliente tras el estudio de viabilidad por si fuese necesario algún cambio o mejora.

2.6.1. RFID Antena

RFID Antena es un lector RFID fijo mono canal de alto rendimiento. Emplea un módulo de alto rendimiento de desarrollo propio basado en el chip Impinj E710 / R2000, y trabajando junto con una antena integrada de alto rendimiento.

El lector soporta interfaces de comunicación RS-232, RJ45 y GPIO.

A continuación, se detallan las características:

CARACTERÍSTICAS FÍSICAS

Dimensiones	270 x 270 x 33 mm / 10,63 x 10,63 x 1,30 pulgadas (9 dBi)
	130 x 130 x 50 mm / 5,12 x 5,12 x 1,97 pulgadas (6 dBi)
Peso	2230g / 78.66oz (9dBi)
	620g / 21.87oz (6dBi)
Material	Plástico + Aleación de Aluminio
Voltaje de entrada	DC 9V - 12V
Corriente en espera	< 30mA
Corriente de trabajo	800mA +/-5% @ DC 12V Entrada
Interfaz de comunicaciones	RS-232, RJ45, GPIO
Tasa de Baud	115200bps
Modo de refrigeración	Refrigeración por aire
Comunicación Método	10/100 Base-T Ethernet (RJ45)

RFID UHF

Motor	CM710-1, basado en Impinj E710
	CM2000-1, basado en Impinj R2000
Protocolo	EPC global UHF Clase 1 Gen 2 / ISO 18000-6C
Frecuencia	865-868MHz / 920-925MHz / 902-928MHz
Potencia de salida	2W Opcional (33dBm, soporte +10~+33dBm ajustable, para América Latina, etc.)
Potencia de salida Precisión	+/- 1dB
Potencia de salida Plana	+/- 0.2dB
Sensibilidad de recepción	< -84dBm
Velocidad de lectura más rápida	950+ tags/sec
RSSI	Soporte
Temperatura ambiente Monitor	Soporte

ENTORNO DEL USUARIO

Temp. de funcionamiento	-25°C a 65°C / -13 °F a 149 °F
Temp. almacenamiento	-40°C a 85°C / -40 °F a 185 °F
Sello	IP67 (9dBi) / IP65 (6dBi) según las especificaciones
Humedad	10% RH - 95% RH sin condensación

ANTENA 1

Frecuencia	865-868MHz / 920-925MHz / 902-928MHz
Modo de Polarización	RHCP (polarización circular derecha)
Ganancia	9dBi
Ángulo Horizontal	60°
Ángulo Vertical	55°
VSWR	≤ 1.3
FBR	17dB
Característica Impedancia	50Ω
Entrada máxima Potencia	20w
Rayo Protección	Protección DC Tierra
Puerto de Antena	SMA Male


ANTENA 2

Frecuencia	865-868MHz / 920-925MHz / 902-928MHz
Modo de Polarización	RHCP (polarización circular derecha)
Ganancia	6dBi
Ángulo Horizontal	105°
Ángulo Vertical	95°
VSWR	≤ 1.3

2.6.2. Etiqueta RFID

La solución de etiquetado SL7017-M4 es ideal para el etiquetado de activos con RFID a demanda para todas las superficies.

CARACTERÍSTICAS

Chip	M4
Tamaño de la antena	17*70mm
Imagen de antena	
Gama de lectura	865MHz:13m 920MHz:10
Aplicación	Activo

2.6.3. Tablet

Modelo: Galaxy Tab Active5 5G & WIFI

Pantalla: 8" 1920 x 1200 (WUXGA)

SO: Android 14

Procesador: Exynos 1380 Octa-core, 2.4GHz, 2GHz

Memoria: 6 GB RAM + 128 GB ROM MicroSD(Hasta 1TB)

Peso: 433gramos

Url con especificaciones completas:

<https://www.samsung.com/es/business/tablets/galaxy-tab-active/galaxy-tab-active5-5g-sm-x306bzgaeeb/>

2.6.4. Lector de matrículas

Modelo: Cámara Hikvision DS-2CD4A26FWD-IZS/P

Cámara IP con capacidad de ANPR (Automatic Number Plate Recognition):

- **Resolución:** 2 MP
- **FPS:** Hasta 60 FPS
- **Visión Nocturna:** Hasta 50 metros
- **Conectividad:** PoE, RTSP
- **Características:** WDR, Detección de Movimiento

Url con especificaciones completas:

<https://www.hikvision.com/my/products/IP-Products/Network-Cameras/Ultra-Series-SmartIP-/ds-2cd4a26fwd-lzs-p/>