UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Integración y monitorización de controles de soldadura en plataforma IIoT para automatización y control de la calidad y mantenimiento predictivo en líneas de soldadura.

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Cárcel Montero, Álvaro

Tutor/a: López Patiño, José Enrique

CURSO ACADÉMICO: 2022/2023

Resumen

En este proyecto se va a desarrollar un sistema que centralice la monitorización de los distintos controles de soldadura de la marca Harms&Wende.

Dichos controles son los utilizados en las líneas de montaje de la planta de carrocería de la factoría de Ford en Almussafes.

Con esta centralización se facilitará y optimizará el mantenimiento de estos sistemas.

Como herramienta principal de trabajo se utilizará una solución de código abierto denominada ELK Stack, integrada por tres tecnologías: Elasticsearch, Logstash y Kibana.

La información en bruto de la monitorización se encuentra distribuida en bases de datos PostgreSQL alojadas en los PCs conectados a los controles de soldadura.

En un primer paso esta información se centralizará en una instancia de Elastic en Google cloud mediante el uso de la herramienta Logstash.

Posteriormente se empleará la herramienta Kibana para poder generar dashboards donde poder visualizar de manera ágil dichos datos.

La centralización de todos estos datos permitirá que posteriormente se puedan utilizar técnicas de Machine Learning y control de tendencias con el fin de optimizar la prevención, el control y corrección de errores y averías.

Resum

En aquest projecte es desenvoluparà un sistema que centralitze el monitoratge dels diferents controls de soldadura de la marca Harms&Wende.

Aquests controls són els utilitzats en les línies de muntatge de la planta de carrosseria de la factoria de Ford a Almussafes.

Amb aquesta centralització es facilitarà i optimitzarà el manteniment d'aquests sistemes.

Com a eina principal de treball s'utilitzarà una solució de codi obert denominada ELK Stack, integrada per tres tecnologies: Elasticsearch, Logstash i Kibana.

La informació en brut del monitoratge es troba distribuïda en bases de dades PostgreSQL allotjades en els PCs connectats als controls de soldadura.

En un primer pas aquesta informació se centralitzarà en una instància de Elastic en Google cloud mitjançant l'ús de l'eina Logstash.

Posteriorment s'emprarà l'eina Kibana per a poder generar dashboards on poder visualitzar de manera àgil aquests dades.

La centralització de totes aquestes dades permetrà que posteriorment es puguen utilitzar tècniques de Machine Learning i control de tendències amb la finalitat d'optimitzar la prevenció, el control i correcció d'errors i avaries.

Abstract

This project is going to develop a system that will centralise the monitoring of the different welding modules of the Harms&Wende brand.

These modules are used in the assembly lines of the bodywork plant of the Ford factory in Almussafes.

This centralisation will simplify and optimise the maintenance of these systems.

ELK Stack is an open source solution that will be used as the main working tool. It is made up of three technologies: Elasticsearch, Logstash and Kibana.

The raw monitoring information is distributed in PostgreSQL databases hosted on the PCs connected to the welding controls.

In a first step, this information will be centralised in an instance of Elastic in the Google cloud using the Logstash tool.

Subsequently, the Kibana tool will be used to generate dashboards where this data can be visualised in an agile way.

The centralisation of all this data will enable Machine Learning and trend monitoring techniques to be used to optimise the prevention, control and correction of errors and faults.

Índice de contenidos

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1 Introducción a la compañía

La fábrica de Ford en Almussafes, Valencia, España, fue inaugurada en 1976 y ha sido una de las fábricas de automóviles más importantes y productivas de España, con una capacidad de producción anual de alrededor de 400.000 vehículos. A lo largo de los años, ha experimentado varias expansiones y actualizaciones para adaptarse a las demandas del mercado y ha sido un importante empleador en la región. En la actualidad, la fábrica se encuentra en medio de un plan de reestructuración con el fin de ser la primera factoría en Europa donde se fabrique el primer modelo totalmente eléctrico de la compañía[1] en vistas de adaptarse a los nuevos tiempos dentro del mundo del automóvil.

1.1.1 Organización de la empresa

La factoría de Ford en Almussafes se divide en las siguientes zonas:

- **V.A.O (Vehicles Assembly Operations):** Es donde se realiza todo el proceso de fabricación del automóvil y está compuesta por las plantas de **Prensas y Carrocerías**, **Pinturas** y **Montaje**
- **Planta de Motores:** Recoge todo el proceso de fabricación del motor, desde el mecanizado de las diferentes piezas hasta el montaje de este.
- **Planta de recambios:** Almacenaje de piezas.
- **Planta motriz:** Suministra energía a la factoría.
- **Planta piloto:** Se encarga de llevar a cabo las operaciones necesarias para el lanzamiento de nuevos modelos y prototipos de vehículos.
- **Servicios:** Incluye **oficinas centrales**, **centro de formación**, **área médica**, **instalaciones deportivas** y **área de bomberos.**

Figura 1. Plano de la factoría.

Este proyecto se desarrolla en la zona de **Prensas y Carrocerías**, concretamente en las plantas Body 1 y Body 2, resaltadas en la figura anterior.

[1] Reestructuración de Ford para el coche eléctrico https://elpais.com/economia/2022-06-22/ford-elige-la-factoria-de-almussafes-para-fabricar-sus-nuevos-coches-electricos.html

1.2 Introducción al proyecto

Este proyecto se focaliza en la parte de la fabricación de la **carrocería** dentro del proceso de manufactura del automóvil. Una de las etapas más importantes en esta parte es la **soldadura por puntos**, que se lleva a cabo en las líneas de producción utilizando equipos especializados para unir las diferentes partes de la carrocería con precisión y fiabilidad. La soldadura por puntos, o soldadura de resistencia por puntos, es un método de unión de dos piezas de metal mediante la aplicación de una corriente eléctrica a un punto específico de las piezas, fundiendo el metal y formando una pequeña protuberancia conocida como "punto". La corriente eléctrica se aplica a través de electrodos que presionan las piezas juntas y crean una conexión sólida entre ellas [1].

La factoría cuenta con distintos proveedores para obtener equipos que puedan realizar el proceso de soldadura por puntos y, de todos ellos, el trabajo está centrado en los equipos Harms & Wende[2].

1.3 Objetivos

La soldadura por puntos es una parte crucial dentro del proceso de manufactura de la carrocería automóvil y, por tanto, es primordial cuidar la calidad con la que se realiza esta etapa al detalle. Es por ello por lo que se pretende **optimizar**, **centralizar** y **automatizar** el proceso de monitorización de calidad y mantenimiento de los equipos de soldadura por puntos del fabricante Harms & Wende.

El objetivo principal consiste en la centralización en una base de datos NoSQL de los parámetros de soldadura almacenados en bases de datos SQL, las cuales están alojadas en distintos ordenadores de soldadura a lo largo y ancho de las plantas Body 1 y Body 2.

Una vez conseguido el objetivo de centralizar los datos se pretende, por un lado, obtener información relevante mediante *dashboards* de visualización, teniendo un seguimiento histórico y en tiempo real de los parámetros almacenados gracias a la ingesta en permanente funcionamiento con el objetivo de optimizar la detección de averías y el monitoreo de la calidad de los puntos de soldadura. Por otro lado, se pretende (para un futuro posterior a este proyecto) procesar estos datos mediante *Machine Learning*, de esta manera no solo se tiene un control gráfico histórico o a tiempo real del estado de los equipos y de la calidad de los puntos sino también un control preventivo consiguiendo así evitar las averías prediciéndolas y no detectarlas una vez ya han sucedido.

[2] Harms & Wende https://harms-wende.de/en/

Capítulo 2. Metodología

2.1 Distribución de tareas

El desarrollo del proyecto se ha dividido en las siguientes tareas de forma cronológica:

1. Estudio de la infraestructura de red y de la distribución de computadoras de soldadura (Weld PC) de la planta de carrocerías.
2. Estudio del funcionamiento del Software de control de soldadura.
3. Estudio de estructura y posibilidad de almacenaje de datos en la base de datos creada por el software de soldadura.
4. Configuración de la base de datos y conexión de esta con servidor a través de la red interna de uno de los ordenadores de soldadura para realizar pruebas posteriores.
5. Creación de archivo de configuración de Logstash de prueba para uno de los ordenadores con consecutivas ejecuciones para llevar a cabo su depuración con el fin de modelar la extracción de datos al completo.
6. Reuniones periódicas con el equipo de soldadura para establecer una selección de parámetros que aporte la información necesaria.
7. Estudio de la herramienta kibana para obtener una representación de datos útil.
8. Diseño y creación de dashboards funcionales.
9. Habilitación de parte de los ordenadores de soldadura para posibilitar su conexión a la red donde se ubica el servidor.
10. Creación de todos los archivos de configuración para la extracción de todos los ordenadores a partir de archivo piloto e implementación de estos en servidor de producción para extracción masiva de datos de todos los ordenadores de soldadura.
11. Verificación de funcionamiento.

2.2 Cronograma

Mediante el siguiente diagrama de Gantt podemos ver el tiempo empleado para llevar a cabo cada una de las tareas:

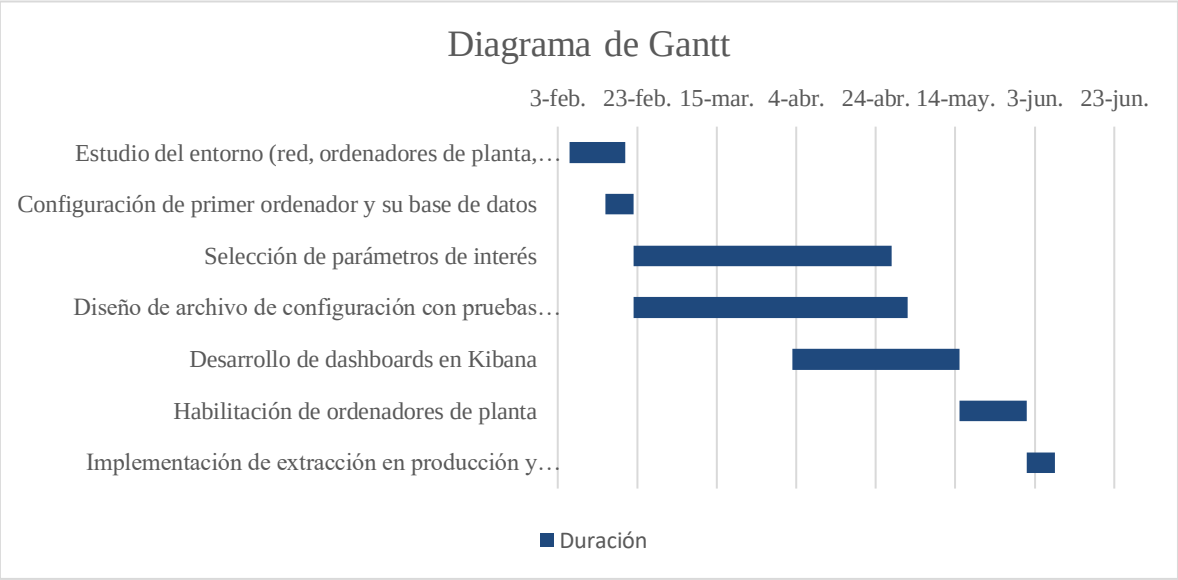


Figura 2. Diagrama de Gantt

Capítulo 3. Tecnologías y herramientas

3.1 ETL

ETL son las siglas de **Extract, Transform y Load**. Son tres funciones que combinadas entre sí y ejecutadas de forma secuencial nos permitirán extraer datos de una fuente de datos, transformarlos para que tengan el formato deseado y guardarlos en otra base de datos [2].

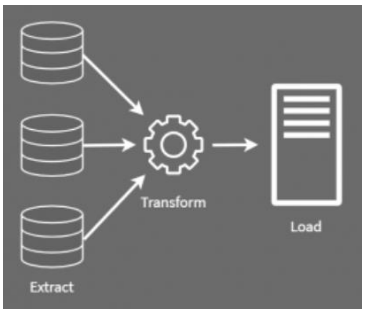


Figura 3. ETL

Este es un proceso muy extendido y utilizado en la actualidad en un afán de obtener un control total sobre los datos que se generan dentro de las empresas provenientes de distintos procesos o actividades relativos a su producción.

Una ETL tiene como objetivo final poder aportar un mayor sentido y obtener toda la información posible de las fuentes de datos dentro de una empresa como pueden ser logs, reportes, mensajes de error, registros de una base de datos SQL… La forma en la que se consigue esto es gracias a los dos principales beneficios aportados por esta operación: una ubicación única en la que almacenar los datos y un formato estándar en la que presentarlos. Esto permite filtrar y buscar la información que se necesita entre la amalgama de datos almacenados de la manera más eficiente posible.

3.2 ELK Stack

Existen múltiples soluciones para llevar a cabo un proceso como el explicado anteriormente y, de entre todas ellas, se ha considerado que la mejor es la ELK Stack.

"ELK" es la sigla de tres herramientas de trabajo de código abierto: Elasticsearch, Kibana y Logstash.

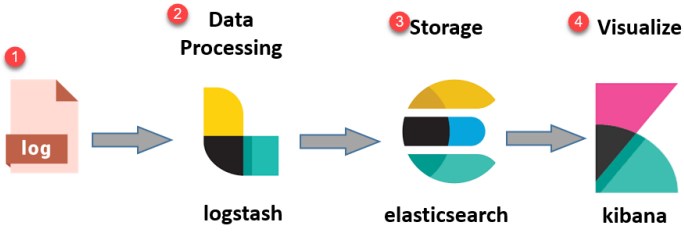


Figura 4. ELK Stack

3.2.1 Elasticsearch

Elasticsearch es un motor de búsqueda y análisis distribuido y de código abierto basado en la biblioteca Apache Lucene que utiliza una base de datos NoSQL [3]. Proporciona una plataforma potente y flexible para analizar y buscar grandes cantidades de datos casi en tiempo real. Elasticsearch puede almacenar y buscar varios tipos de datos, como estructurados, no estructurados, geográficos y métricos.

Una de las principales características de Elasticsearch es su capacidad para escalar horizontalmente, lo que significa que puede distribuir los datos y el procesamiento a través de muchos servidores con el fin de manejar grandes cantidades de datos de manera eficiente. Elasticsearch también admite consultas de búsqueda complejas, filtrado, agregaciones, sugerencias y mucho más.

En Elasticsearch, un *cluster* [4] es un conjunto de uno o más nodos que trabajan juntos para almacenar y procesar datos. Los nodos en un *cluster* se comunican entre sí y comparten información sobre los datos que están almacenando, para garantizar que todos los nodos tengan una copia del mismo conjunto de datos y puedan responder a las consultas de manera coherente.

Dentro de un *cluster*, los datos se organizan en **índices**, que son colecciones de documentos relacionados. Cada documento es un objeto JSON que contiene uno o más campos que describen alguna entidad o dato. Los índices se dividen en fragmentos (*shards*) para permitir la distribución y el procesamiento de datos en diferentes nodos del *cluster*. Cada fragmento es una porción independiente del índice que se almacena y procesa en un nodo específico.

3.2.2 Logstash

Logstash es lo que se conoce como una canal de procesado de datos gratuita y de código abierto que ingesta datos desde múltiples fuentes y los transforma acorde a unos requisitos para después enviarlos al destino deseado [5].

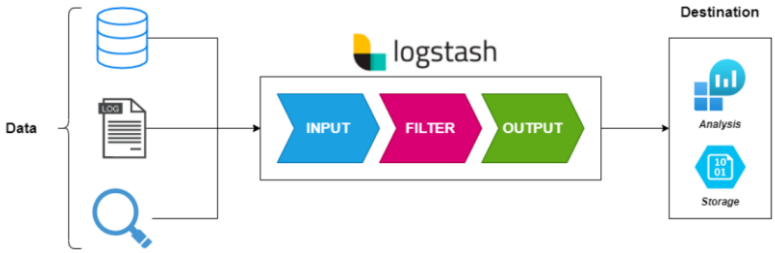


Figura 5. Ejemplo visual de canal de Logstash

Internamente, un canal de Logstash consta de tres componentes principales: **entradas**, **filtros** y **salidas**. Las **entradas** se encargan de recopilar datos de diversas fuentes, como archivos de registro, colas de mensajes y otros sistemas externos. Los **filtros** se utilizan para preprocesar los datos aplicando transformaciones y modificaciones como el análisis sintáctico, el enriquecimiento y la eliminación de campos. Por último, las **salidas** se utilizan para enviar los datos preprocesados a Elasticsearch, otros sistemas de almacenamiento o servicios externos. El canal se estructura de forma modular, esto quiere decir que en cada parte se añaden las funcionalidades de manera separada con lo que se conoce como *plugins*[3] [5]

[3] Un plugin en Logstash es un componente de software que se utiliza para darle funcionalidad a la plataforma proporcionando una forma modular y escalable para realizar tareas específicas de procesamiento de datos.

Ruby es el lenguaje de programación utilizado para crear Logstash, que se ejecuta en una máquina virtual Java (JVM). Las distintas partes del canal de Logstash deben definirse en un archivo de configuración .conf antes de que Logstash pueda ponerse en funcionamiento. Los filtros y transformaciones definidos en el archivo de configuración se aplican durante la ejecución, ya que Logstash supervisa continuamente el flujo de entrada. El programa también ofrece capacidades para la supervisión del procesamiento en tiempo real y la detección de errores o problemas de rendimiento [6].

3.2.3 Kibana

Kibana es una herramienta de visualización de datos que se utiliza para analizar y explorar grandes y complejas colecciones de datos almacenados en Elasticsearch. proporciona una interfaz de usuario basada en la web para crear y compartir visualizaciones personalizadas, paneles y redes de comandos. Kibana se utiliza para rastrear y analizar registros, métricas, aplicaciones, seguridad y otros datos.

Kibana se divide en diferentes módulos, cada uno con un propósito diferente, como visualizaciones, cuadros de mando y aprendizaje automático. El módulo de visualización de Kibana permite crear representaciones visuales de los datos de muchas maneras diferentes, incluyendo gráficos de líneas, gráficos de barras, gráficos circulares, tablas y muchos más. Un usuario también puede crear cuadros de mando personalizados que muestren múltiples visualizaciones a la vez.

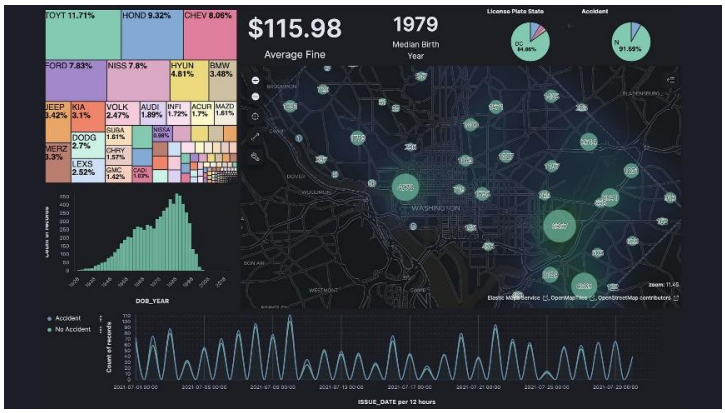
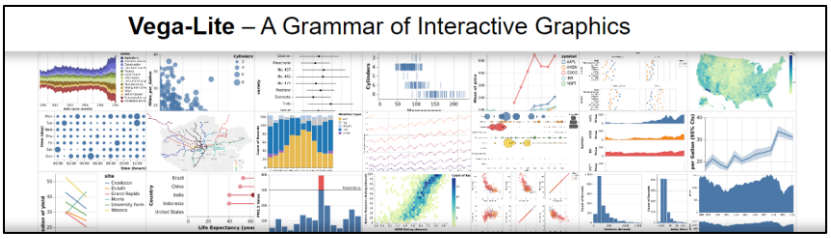


Figura 6. Pantallazo ejemplo de Kibana

Kibana utiliza Elasticsearch Query DSL para construir consultas y buscar datos dentro de los clusters de Elasticsearch. El DSL de Elasticsearch es un lenguaje específico del dominio basado en JSON que permite definir consultas complejas como consultas bool, consultas de coincidencia, consultas de rango y muchas más [7].

3.2.3.1 Vega Lite

Vega lite [8] es una librería de visualización de datos de alto nivel que permite a los usuarios crear gráficos personalizados e interactivos utilizando una sintaxis simplificada y declarativa. Está integrado en Kibana por lo que permite crear visualizaciones a partir de datos almacenados en Elastic directamente [9].



3.3 Google Cloud

Google Cloud es una plataforma de servicios en la nube que permite a los usuarios crear, alojar y construir aplicaciones y sitios web, así como almacenar, procesar y analizar datos en línea. Esta tecnología ofrece una amplia variedad de servicios, incluyendo almacenamiento de datos, análisis de datos, inteligencia artificial y aprendizaje automático, servicios de Internet de las cosas (IoT), herramientas de desarrollo de aplicaciones y más.

Figura 7. Google cloud

Cloud Storage es el servicio de almacenamiento de objetos de Google que ofrece almacenamiento de alta disponibilidad y escalabilidad para datos no estructurados. Ofrece funciones como la georredundancia, la replicación automática de datos y políticas de gestión del ciclo de vida que permiten a los clientes almacenar, recuperar y gestionar sus datos de forma rentable y sencilla [10].

3.4 XPegasus

XPegasus es la interfaz de usuario global para los controladores GeniusMFI, GeniusHWI, HWI 28XX y HWI 22XX y Ratia 73. Combina un funcionamiento uniforme bajo un mismo techo sin importar qué control está utilizando [11].

Figura 8. Controles de soldadura Harms & Wende

Es un software diseñado para controlar y monitorizar el proceso de soldadura por puntos en tiempo real. De las distintas soluciones con las que cuenta el programa el proyecto se centra en la monitorización de parámetros por cada punto soldado. El software cuenta con una base de datos PostgreSQL para almacenar estos parámetros.

3.5 PostgreSQL

PostgreSQL es un sistema gestor de bases de datos relacionales de código abierto que hace uso del lenguaje SQL. Es un sistema de bases de datos relacional alta disponibilidad. Una de sus características más valiosas es su robustez y estabilidad, atributos muy solicitados por las empresas [12]. Proporciona soporte para JSON, XML y datos de tipo de matriz, así como soporte para transacciones ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad).

Figura 9. PostgreSQL

3.6 Ruby

Ruby es un lenguaje de programación dinámico y de código abierto enfocado en la simplicidad y productividad. El objetivo de Ruby es ser fácil de leer y escribir. Las estructuras de control de flujo de Ruby, como las condicionales y los bucles, son similares a los de otros lenguajes de programación y son fáciles de aprender [13].

Estas características y el hecho de que es un lenguaje ya utilizado en el mundo del procesamiento de datos y la automatización es lo que lo han hecho el lenguaje en el que está escrito la herramienta previamente explicada Logstash.

3.7 HeidiSQL

HeidiSQL es una herramienta de administración de sistemas de gestión de bases de datos libre y de código abierto [14]. Proporciona una interfaz gráfica fácil de usar para trabajar con PostgreSQL y otras bases de datos, lo que facilitará en este proyecto la visualización y verificación de los datos a trabajar.

Figura 10. HeidiSQL

Capítulo 4. Desarrollo del proyecto

En base al desglose de las tareas realizado en el capítulo de metodología podemos dividir el desarrollo del proyecto en cuatro fases diferenciales: preparación del entorno, configuración de canal de logstash, diseño de dashboards en kibana e implementación en planta.

4.1 Preparación del entorno

4.1.1 Estudio del escenario actual

En primer lugar, es necesario plantear la situación de la cual partimos, a partir de ahí establecer una hoja de ruta mediante la cual proceder para conseguir el objetivo.

Las pinzas de soldadura del fabricante Harms & Wende están conectadas a controles realizados por el mismo fabricante y estos equipos se encuentran repartidos por líneas de las plantas Body 1 y Body 2 del área de carrocerías. Hay un máximo de tres pinzas conectadas a cada control, no obstante, lo más habitual es ver una pinza por cada control. Asimismo, los distintos controles de soldadura están conectados a ordenadores denominados "Weld PC", donde se encuentra el software XPegasus, desde el cual el operario de línea deberá configurar los controles para que las pinzas ejecuten los puntos sobre las piezas de chapa según las necesidades y los requisitos establecidos. El software también es capaz de obtener parámetros de las soldaduras realizadas gracias a los sistemas de monitorización equipados en los equipos y en los controles.

La cantidad de controles conectados a cada Weld PC dependerá de las necesidades de la línea donde se ubiquen los equipos, donde influyen factores como el tipo de piezas a soldar o cuestiones de distribución del espacio físico compartido con el resto de maquinaria en cada línea.

Actualmente, el control y monitoreo de la calidad de los puntos de soldadura se realiza de manera local mediante un cuaderno de Excel alojado en cada uno de los Weld PC. Estos albergan valores y parámetros cargados desde archivos de *backup* generados automáticamente por el software XPegasus y alojados en una carpeta local de ordenador. Una vez realizado el volcado periódico de estos datos sobre el cuaderno de Excel local, el operario encargado de ello podrá ver si dichos parámetros se encuentran dentro de los umbrales requeridos para llegar al estándar de calidad requerida y actuar en consecuencia.

4.1.2 Planteamiento general

La meta final del proyecto es optimizar el proceso recién explicado y para ejemplificar la forma en la que se llevará a cabo se ha realizado el siguiente esquema:

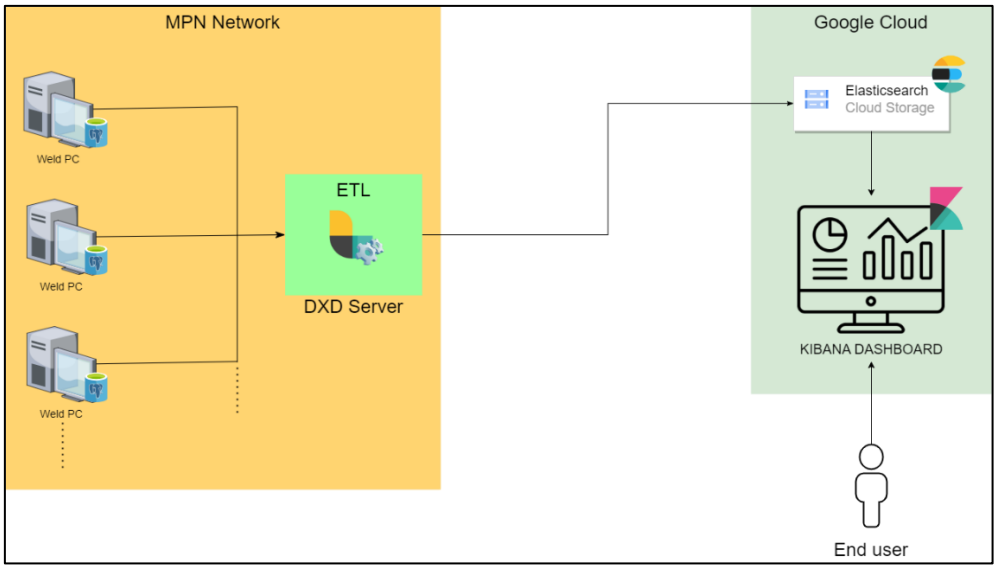


Figura 11. Esquema general del proyecto

- En primer lugar, vemos el conjunto de Weld PC con sus respectivas bases de datos PostgreSQL en las que se almacenan los parámetros de cada punto en tiempo real.
- Se conectan dichas bases de datos, a través de la red MPN[4] de la empresa con el servidor DXD[5].
- En el servidor DXD se encuentra en permanente ejecución el archivo de configuración de Logstash encargado de **extraer**, **modificar** acorde con las necesidades establecidas y **cargar**, mediante archivos JSON, a la base de datos NoSQL utilizada por Elasticsearch, alojada en servidor de Google Cloud (ETL).
- Por último, mediante la herramienta Kibana, se diseñan dashboards que aporten información de relevancia para el proceso de monitorización de la calidad de la soldadura por puntos.

Como hemos visto, disponemos de un conjunto de ordenadores de soldadura repartidos por las plantas Body 1 y Body 2. A continuación se desplegará un listado de estos con sus características de interés:

GIng/ARMARIO	Controles	Conexión a Red MPN
5B LH/CX-482_V-408/SOLDADURA	7 Equipos	SÍ
5B RH/CX-482_V-408/SOLDADURA	7 Equipos	SÍ
7X LH/CX-482/SOLDADURA	22 Equipos	NO
7X RH/CX-482/SOLDADURA	22 Equipos	NO
8XY/CX-482/SOLDADURA	26 Equipos	NO
8A/CX-482/SOLDADURA	16 Equipos	NO
8N(2)/CX-482/SOLDADURA	10 Equipos	NO
9XY/CX-482/SOLDADURA	13 Equipos	NO
9A LH/CX-482/SOLDADURA	10 Equipos	NO
6G/CX-482/SOLDADURA	22 Equipos	SÍ

[4] Consiste en una red interna del área de carrocerías que conecta los equipos en producción de planta con otras áreas de interés como la oficina de ingeniería o el departamento de IT entre otras.

[5] Se trata de un servidor utilizado por el departamento de ingeniería preparado para llevar a cabo procesos en ejecución permanente.

6X/CX-482/SOLDADURA	23 Equipos	SÍ
6Y/CX-482/SOLDADURA	20 Equipos	SÍ
8R/CX-482/SOLDADURA	5 Equipos	NO
7F3-K-L LH/CX-482/SOLDADURA	22 Equipos	SÍ
7F3-K-L RH/CX-482/SOLDADURA	22 Equipos	NO
8J/CX-482/SOLDADURA	9 Equipos+5 Equipos	NO
9ASUB LH/CX-482/SOLDADURA	3 Equipos	NO
9ASUB RH/CX-482/SOLDADURA	4 Equipos	NO
8F LH/CX-482/SOLDADURA	2 Equipos	NO
8C/CX-482/SOLDADURA	1 Equipo	NO
8Apre/CX-482/SOLDADURA	4 Equipos +1 8H	NO
9A RH/CX-482/SOLDADURA	9 Equipos	NO
8H/CX-482/SOLDADURA	1 Equipo	NO
9C/CX-482/SOLDADURA	3 Equipos	NO
9B/CX-482/SOLDADURA	3 Equipos	NO
5J/CX-482/SOLDADURA	1 Equipo	NO

Tabla 1. Listado de Weld PC

La columna de Grupo de Ingeniería indica la ubicación en la que se encuentra el ordenador y, por tanto, la parte de la carrocería que sueldan los equipos de la línea conectados al Weld PC en cuestión; en la segunda columna se indican el número de controles conectados a cada ordenador y, por último, se nos indica si tienen conexión a la red MPN.

4.1.3 Conexión de la primera línea

Una vez se tiene un plano general del escenario en el que se desarrollará el proyecto se procederá a seleccionar los ordenadores de soldadura de únicamente una de las líneas y a establecer una "conexión de prueba" al portátil de trabajo. Mediante esta conexión podremos configurar y perfilar adecuadamente la etapa de **transformación** de datos dentro del proceso ETL depurando los datos de salida de la ingesta que se ejecutará en local dentro del portátil. Una vez se obtenga el resultado deseado, se desplegará la extracción en el servidor DXD donde el proceso se encontrará en permanente ejecución.

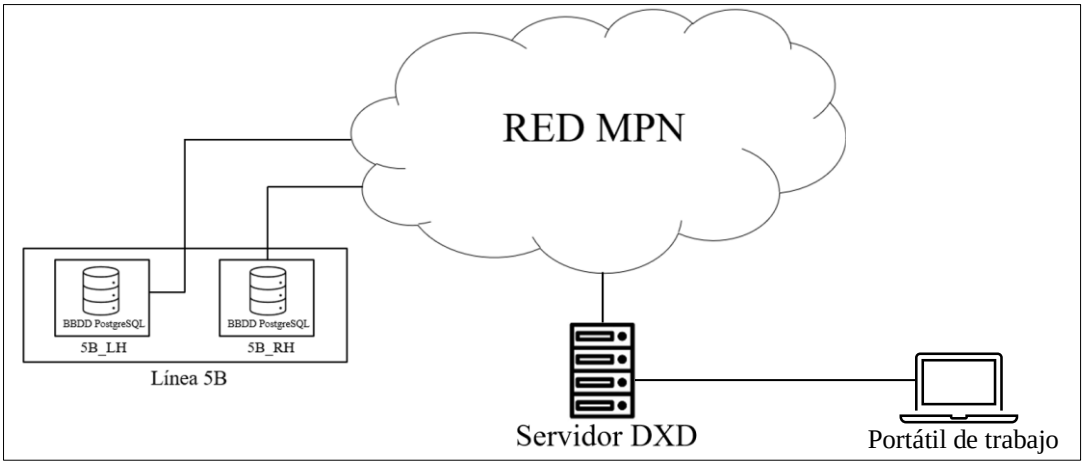


Figura 12. Esquema de conexión[6]

Se pretende establecer una conexión desde el ordenador de trabajo a las bases de datos y esto se hace posible gracias a la realización de un reenvío de puertos mediante el comando de Windows NETSH para cuyo uso, es necesario tener privilegios administrativos sobre la máquina en la que se utiliza. Dicho comando tiene una propiedad llamada *portproxy* y es la que permite la comunicación entre puertos en la red IP [15].

```
netsh interface portproxy add v4tov4 listenport=<Puerto-Local> listenaddress=<IP-
Local> connectport=<Puerto-Remoto> connectaddress=<IP-Remota>
```

Figura 13. Comando NETSH *portproxy*

El comando es introducido en el servidor DXD para que así las bases de datos de ambos ordenadores de soldadura sean visibles desde el portátil de trabajo conectándose únicamente al servidor. La base de datos de cada ordenador se asignará a un puerto distinto del servidor.

Para que la comunicación de puertos sea efectiva será necesario comprobar la correcta configuración de los ordenadores de soldadura, esta consiste en lo siguiente:

- Cortafuegos deshabilitado
- Asignar el tipo de ubicación privado a las redes no identificadas en la directiva de seguridad local.

[6] Para simplificar nos referiremos a la 5B como una única línea, no obstante, físicamente se trata de dos líneas separadas: 5B_LH (*Left*) y 5B_RH (*Right*), es decir, en ambas se elabora la misma parte de la carrocería, pero se realiza el lado izquierdo en una y el lado derecho en la otra. Es por ello por lo que tenemos dos ordenadores de soldadura distintos y no solamente uno.

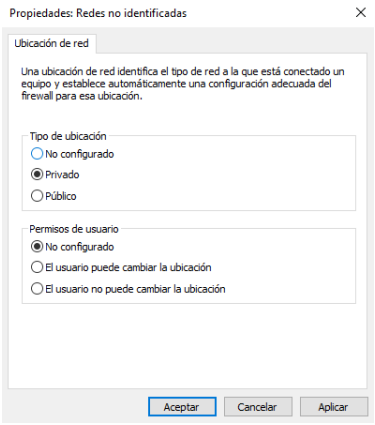


Figura 14. Asignación redes no identificadas

- Modificar archivo de configuración "pg_hba.conf" de la base de datos ubicado en la carpeta del programa XPegasus con el objetivo de permitir la conexión a la base de datos desde cualquier host, añadiendo la siguiente línea:

```
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
host    all             all             192.168.0.0/16          md5
host    all             all             0.0.0.0/0               md5
```

Figura 15. Configuración base de datos

Es necesario reiniciar el servicio que mantiene actualizada la base de datos desde el programa XPegasus para poder hacer efectivo este último cambio en la configuración.

Una vez se ha configurado el ordenador y su respectiva base de datos correctamente, los puertos quedan comunicados y es posible acceder a las bases de datos desde el portátil de trabajo.

4.1.4 Estudio de la base de datos

Se hace uso de la herramienta HeidiSQL para establecer la conexión con las bases de datos y poder visualizar las distintas tablas que las componen. Estas constan de una estructura idéntica para todos los ordenadores.

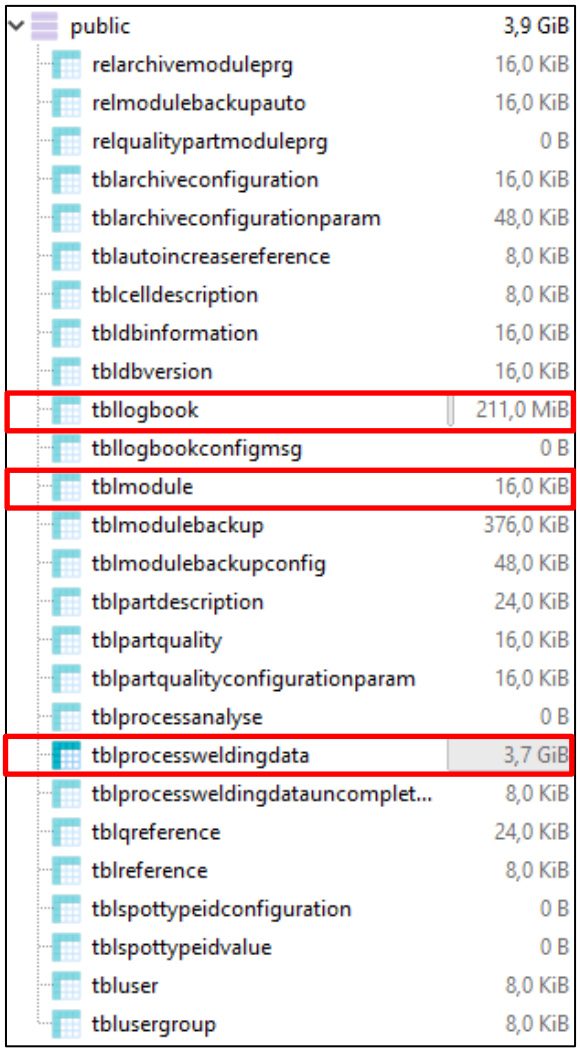


Figura 16. Lista de tablas de la base de datos SQL

La tabla principal donde se guarda el valor de los parámetros de cada punto de soldadura es "*tblprocessweldingdata*" y, por lo tanto, se considerará como la tabla primaria de la ingesta. Adicionalmente, serán de interés las tablas "*tblmodule*" y "*tbllogbook*". La primera es una tabla relacionada a la tabla primaria mediante el campo "*moduleid*" y nos dice el control de soldadura que ha realizado el punto mediante el campo "*modulename*" y su línea con el campo "*modulegroup*".

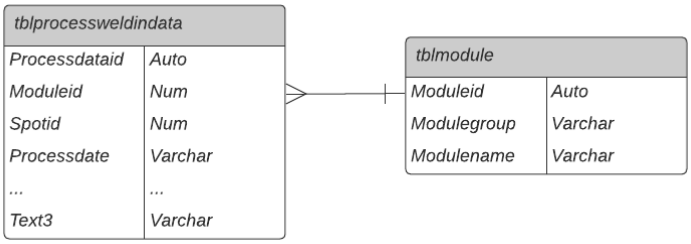


Figura 17. Relación de tabla primaria con "*tblmodule*"

La tabla "*tbllogbook*" almacena el diario de registro[7] al completo y se rige por los siguientes campos:

Campo	Descripción
Logid	Campo clave ID del registro.
Logtype	Tipo de registro. 0 = fallos, 1 = alarma, 2 = información.
Logcategoryid	
Modulename	Identificador del control de soldadura.
Logtextid	Identificador del texto del registro.
Logdate	Fecha del registro.
Logtime	Hora del registro.
Logparam1	%p0. Mismo valor que campo Logtype.
Logparam2	%p1.
Logparam3	%p2.
…	…
Logparam20	%p19
Userid	ID del usuario que se encontraba manipulando el programa XPegasus cuando se dio el registro.
Username	Nombre del usuario que se encontraba manipulando el programa XPegasus cuando se dio el registro.

Tabla 2. Campos de la tabla "*tbllogbook*"

Se puede ver que existe una serie de campos llamados "*Logparams*" cuya razón de ser es dotar de significado al texto del registro.

A continuación, se encuentra un listado de los campos de la tabla primaria y su correspondiente descripción con el fin de hacer visible el alcance de la información que es posible obtener de ella:

[7] Lugar donde se anotan los sucesos o incidencias relevantes como una modificación en la configuración o una parada en la línea

Campo	Descripción
Processdataid	Campo clave ID del punto soldado.
Moduleid	Clave foránea de la tabla "***tblModule***".
Partid	Clave foránea de la tabla "*tblPartDescription*".
Spotid	<Campo inutilizado>
Processdate	Fecha en la que se ha soldado el punto.
Processtime	Hora en la que se ha soldado el punto.
Parttempnameint	<Campo inutilizado>
Partname	Nombre de pieza generado por XPegasus para el seguimiento de piezas. Solo es posible con un PLC y la asignación de componentes
Partok	Indica si se soldó correctamente la pieza. Solo es posible con un PLC y la asignación de componentes.
Programnumber	Número de programa[8].
Spotname	Nombre del punto.
Spotok	<Campo inutilizado>
Gunname	Nombre de la pinza.
Gunnumber	Número de pinza.
Actualcounter	Contador de puntos.
Boolvalue1	Parámetro booleano 1
…	…
Boolvalue5	Parámetro booleano 5
HwhIdBoolvalue1	Clave identificadora del parámetro booleano 1.
…	…
HwhIdBoolvalue5	Clave identificadora del parámetro booleano 5.
Value1	Parámetro 1
…	…
Value15	Parámetro 15
HwhIdValue1	Clave identificadora del parámetro 1.

[8] Identifica una configuración concreta diseñada para un tipo de punto específico de la pieza que se suelda.

…	…
HwhIdValue15	Clave identificadora del parámetro 15.
Curve1	Curva 1.
…	…
Curve7	Curva 7.
HwhIdCurve1	Clave identificadora de la curva 1.
…	…
HwhIdCurve7	Clave identificadora de la curva 7.
Text1	Texto 1.
Text2	Texto 2.
Text3	Texto 3.
HwhIdText1	Clave identificadora del texto 1.
HwhIdText2	Clave identificadora del texto 2.
HwhIdText3	Clave identificadora del texto 3.

Tabla 3. Campos de la tabla "*tblprocessweldingdata*".

En primer lugar, existe una serie de campos fijos, es decir, siempre van a aportar el mismo tipo de información. Esta información suele ser sobre el control, la pinza o demás generalidades que son permanentes tras la instalación de los equipos. Posteriores a estos campos, se encuentran aquellos que nos aportan parámetros e información sobre la soldadura.

En el proceso de soldadura de un punto entran en juego una inmensa cantidad de factores que pueden poner en riesgo la calidad en la ejecución del punto, sin embargo, la capacidad del ordenador de soldadura y de la base de datos donde se almacenan estos parámetros es limitada. Es por ello por lo que el software XPegasus solo permite seleccionar una combinación limitada de estos parámetros. No obstante, esta selección es más que suficiente para abordar las métricas más importantes que rigen la calidad del punto.

4.1.5 Selección de parámetros

Como se ha podido ver en el subapartado anterior, existe una serie de campos en la base de datos a los que se les podría llamar "seleccionables". Estos se dividen en tres tipos: valores (*values*), valores booleanos (*boolvalues*) y textos (*texts*).

Para abordar estos campos es necesario, en primer lugar, entender las necesidades del equipo de soldadura y, en segundo lugar, tener en cuenta qué permite configurar y cambiar el software XPegasus, ya que, no se hace uso de todos los parámetros disponibles para guardar debido a que no influyen a las necesidades concretas de la empresa.

4.1.5.1 Conceptos previos

Antes de continuar es necesario explicar, a grandes rasgos, ciertos aspectos del proceso de fabricación de la carrocería de un coche y, en concreto, sobre el procedimiento de la soldadura por puntos que se sigue a nivel de equipamiento del proveedor Harms & Wende y contexto concreto dentro de las plantas para poder abordar los requisitos del equipo de soldadura.

Los equipos de soldadura por puntos constan de una pinza con unos electrodos a través de los cuales cruza la corriente mientras se presiona la chapa de carrocería, el aumento del calor debido al aumento de la resistencia por el paso de la corriente (siguiendo la ley de Joule [16]) es el que provoca que las dos piezas de chapa se fundan y se solidifiquen en un mismo punto unido [17]. Este fenómeno provoca que los electrodos que se mencionaban anteriormente vayan desgastándose y adquiriendo fragmentos residuales de chapa e impurezas con el paso del tiempo y, por tanto, es necesario fresarlos[9] periódicamente hasta que sea necesario cambiar los electrodos de la pinza.

Como se ha mencionado en el subapartado anterior, cada pinza suelda distintos puntos a lo largo y ancho de una pieza, cada uno de estos puntos distintos están configurados de una manera y esta configuración se la identifica como **número de programa**. Existen programas o configuraciones cuya función no es soldar un punto sino comprobar el estado de los electrodos tras haber sido fresados o repuestos, a estos se les conoce como programas de "*autochequeo*". Esto sirve para verificar que el funcionamiento del equipo es correcto ya que, si se obtiene un resultado insatisfactorio del programa de "*autochequeo*" significa que el descenso en calidad de la soldadura que ha llevado previamente al fresado o reposición de los electrodos en un primer momento no era debido a estos, sino a alguna avería en la pinza o a cualquier otro aspecto ajeno a los electrodos [18].

Por otro lado, el control se nutre de información acerca de la efectividad de cada punto gracias a las herramientas de monitorización sobre el equipamiento que tiene conectadas a su bus de entradas y salidas, como una sonda de corriente y otra de tensión (entre otras). Esta información recopilada por el control sobre cada punto sirve para que este actúe en consecuencia tras haber sido recibida, dependiendo también de la configuración realizada desde el software de XPegasus. Esto quiere decir que el control es capaz de regular la soldadura y/o detener el funcionamiento según la información que haya recibido durante el punto o tras el último punto soldado. Aquí entra en juego un parámetro muy importante conocido como **modo de regulación**. Existen tres modos de regulación acorde con la guía de funcionamiento [18]:

- **SKT**: en este modo el control no regula, únicamente dejará pasar cierta cantidad de corriente en función del valor *SKT* introducido desde el software. Este valor *SKT* funciona como si de un potenciómetro se tratase, siendo 0 *SKT* dejar pasar el mínimo de corriente y 999 *SKT* dejar pasar el máximo flujo de corriente posible [18]. Este es un mecanismo heredado de otro anterior en el que se realizaba una modulación de ancho de pulso (PWM) a la señal de corriente alterna previa al transformador del control.

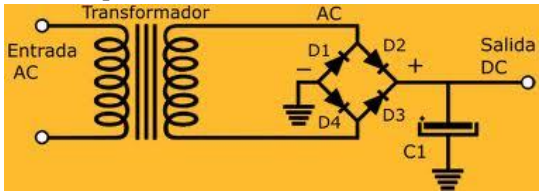


Figura 18. Transformador de corriente a la salida del control.

  Mediante este procedimiento, se modificaba el valor medio a la salida del transformador y, por tanto, el flujo de corriente continua que atravesaba los electrodos era mayor o

[9] El fresado es un proceso por el cual se lima la punta de los electrodos sujetos a la pinza, con el fin de eliminar fragmentos de chapa e impurezas, consiguiendo así mejorar la superficie de contacto de los electrodos con la chapa y, en consecuencia, un mejor resultado en la soldadura.

menor en función de los grados de fase que se introdujesen en el parámetro, siendo 0º dejar pasar todo el flujo de la corriente y 180º no dejar pasar nada [19].

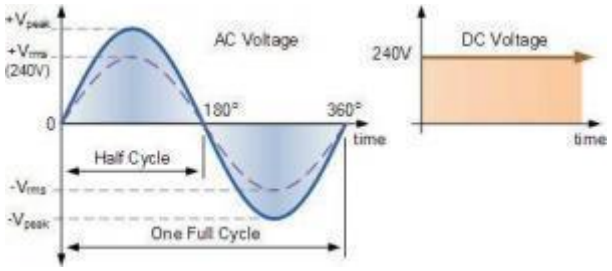


Figura 19. Transformación AC a DC.

- **KSR**: Modo de regulación por corriente. El operario define en el software los Kiloamperios que debe alcanzar en un tiempo establecido (generalmente en el orden de milisegundos). El control fuerza al equipo a llegar a la corriente marcada en el tiempo establecido. Actualmente este modo se encuentra en **desuso** [18].
- **IQR**: Modo de regulación por resistencia. Se trata del modo de regulación más preciso de los tres ya que trata de perfeccionar la curva de resistencia durante la soldadura de un punto. Este es el método que mejor palia las imperfecciones debidas a factores externos a los parámetros configurados, como pueden ser las impurezas de los electrodos o a una ligera desviación en la pinza que afecte a la superficie de contacto debido a que corrige el flujo de corriente en función de la curva de resistencia obtenida.

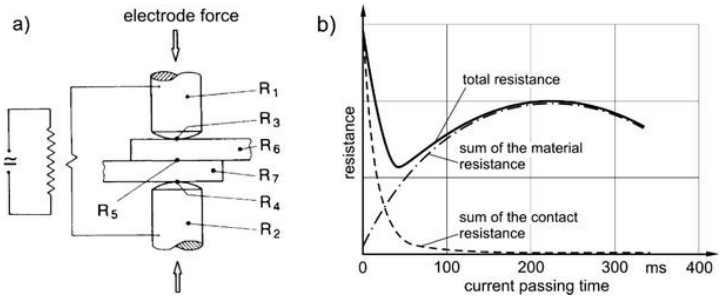


Figura 20. Curva de resistencia en soldadura por puntos.

Otro indicador muy importante del descenso de la calidad en la soldadura es la aparición de proyecciones. Se entiende por proyección una protuberancia que puede ser provocada por diversos motivos, como un exceso de corriente o de tiempo de soldadura, así como una presión incorrecta de la pinza o impurezas o desperfectos en los electrodos que afecten a la superficie de contacto [20].

4.1.5.2 Selección de parámetros

Habiendo visto ciertos conceptos previos, es posible entender las necesidades del equipo de soldadura respecto al proyecto, siendo estas las siguientes:

- Control del porcentaje de proyecciones que se dan sobre el total de puntos soldados.
- Verificación de que todos los programas de "*autochequeo*" se encuentran en el modo de regulación SKT y que todos los programas de soldadura se encuentran en el modo de regulación IQR.
- Verificación de que las tolerancias de corriente de los programas de "*autochequeo*" se encuentran en un porcentaje concreto.
- Conteo del número de piezas soldadas.

- Detalle de parámetros (corriente, voltaje, resistencia…) de los puntos soldados.

A partir de aquí, es necesario ver qué parámetros nos permite seleccionar XPegasus para completar estos campos "seleccionables" de la base de datos explicados anteriormente y deliberar con el equipo de soldadura para conseguir una selección que se ajuste lo máximo posible a las necesidades expuestas.

Se ha visto que solo son necesarios los parámetros de los campos *Values* y *Curves*, por tanto, por razones de optimización, se van a dejar los campos de *boolvalues* y *texts* deseleccionados.

La lista de parámetros que son seleccionables para los campos *Values* se divide en distintos apartados en función del tipo de información que estos aportan:

- General.
- Parámetros IQR.
- Parámetros IQFlex.
- Monitoreo de corriente.
- Monitoreo de tensión.
- Monitoreo de salpicaduras.
- Monitoreo de carrera de regulación.
- Medición de recorrido.
- Monitoreo de fuerza.
- Monitoreo de resistencia.
- Deriva de proceso.
- Monitoreo de AMF.

Dentro de cada familia existen una multitud de campos y, mediante ensayo y error se realizarán distintas combinaciones de 15 parámetros hasta conseguir una selección ajustada al equipo de soldadura.

No se abordará cada campo disponible ni su significado ya que excede la motivación esencial y las competencias de este proyecto y el peso de su discusión recae sobre el equipo de soldadura que son los expertos en la materia.

Por otro lado, la selección de *curves* consiste básicamente en: Señales de voltaje, corriente y resistencia y la curva de carrera de regulación[10].

Por tanto, la selección final de parámetros es la siguiente:

[10] Esta curva es información relevante en la aplicación del valor de SKT en los programas de "*autochequeo*"

	Evolución de las señales
1	Curva de I
2	Curva de tensión electrodos
3	R-Inspector: curva de resistencia filtrada
4	Curva de carrera de regulación
5	Libre
6	Libre
7	Libre
	Parámetros
1	Contador ciclos fresado
2	Cont. pun. de fre.
3	T soldadura efectivo
4	U-Inspector: tensión de electrodos efectiva promedio
5	I-Inspector: I secundaria efectiva promedio
6	Corriente nominal promedio
7	I-Inspector: tolerancia + (promedio)
8	I-Inspector: tolerancia - (promedio)
9	R-Inspector: resistencia efectiva promedio
10	R-Inspector: resistencia nominal promedio
11	R-Inspector: tolerancia + (promedio)
12	R-Inspector: tolerancia - (promedio)
13	Modo de regulación
14	Momento de salpicadura
15	Ptos. Por Pie.
	Valores Si/No
1	Libre
2	Libre
3	Libre
4	Libre
5	Libre
	Texto
1	Libre
2	Libre
3	Libre

Figura 21. Selección de parámetros.

A continuación, se explicará brevemente la información que aporta cada parámetro:

Parámetro	Descripción
Contador ciclos fresado	Número de fresados a los electrodos antes de que estos sean repuestos.
Cont. Punt. Fre.	Número de puntos soldados entre fresados de los electrodos.
T soldadura efectivo	Tiempo que dura el punto.
U-Inspector: tensión de electrodos efectiva promedio	Monitoreo de tensión: promedio de tensión medida en los electrodos durante la ejecución del punto.
I-Inspector: I secundaria efectiva promedio	Monitoreo de corriente: promedio de la corriente medida durante la ejecución del punto.
Corriente nominal promedio	Media de la corriente nominal.
I-inspector: tolerancia + (promedio)	Monitoreo de corriente: tolerancia superior.
I-inspector: tolerancia – (promedio)	Monitoreo de corriente: tolerancia inferior.
R-inspector: resistencia efectiva promedio	Monitoreo de resistencia: promedio de la resistencia medida durante la ejecución del punto.
R-inspector: resistencia nominal promedio	Monitoreo de resistencia: promedio de la resistencia nominal.
R-inspector: tolerancia + (promedio)	Monitoreo de resistencia: tolerancia superior.
R-inspector: tolerancia - (promedio)	Monitoreo de resistencia: tolerancia inferior.
Modo de regulación	Modo de regulación bajo el que se ha soldado el punto.
Momento de salpicadura	Momento en el que se ha detectado proyección a lo largo de la ejecución del punto (en milisegundos). En caso de que el valor <1 significa que no ha existido proyección en el punto.
Ptos. Por Pieza	Número de puntos que realiza una pinza sobre cada pieza.

Tabla 4. Descripción de parámetros.

Una vez obtenidos y verificados los parámetros deseados es momento de comenzar la extracción.

4.2 Configuración del canal de Logstash

Como se ha explicado previamente, un canal de Logstash consiste en un archivo de configuración escrito en lenguaje *Ruby* formado por tres partes fundamentales: **entradas**, **filtros** y **salidas** [5].

```
input {
  # Aquí se especifica la fuente del mensaje de registro. Puede ser un archivo, una base de datos,
una cola, etc.
}

filter {
  # Aquí se definen las acciones que se deben realizar sobre los mensajes de registro: análisis,
conversión de formato, etc.
}

output {
  # Aquí se especifica el destino del mensaje de registro. Puede ser un archivo, una base de
datos, un servicio de mensajería, etc.
}
```

Figura 22. Esqueleto de archivo de configuración de Logstash.

A continuación, se explicará cómo se ha configurado el canal de Logstash que ha efectuado la ETL desglosada en las tres partes que configuran dicho canal.

4.2.1 Entradas

Ya se ha visto que nuestra única fuente consiste en bases de datos SQL, por lo tanto, el módulo fundamental que utilizaremos en esta parte de **entradas** es el *plugin JDBC* [21]. Este plugin fue creado como una forma de ingerir datos desde cualquier base de datos con una interfaz JDBC en Logstash. Es posible programar periódicamente la ingesta utilizando una sintaxis *cron*[11] o ejecutar la consulta una vez para cargar los datos en Logstash. Cada fila del conjunto de resultados se convierte en un único evento[12]. Las columnas del conjunto de resultados se convierten en campos del evento.

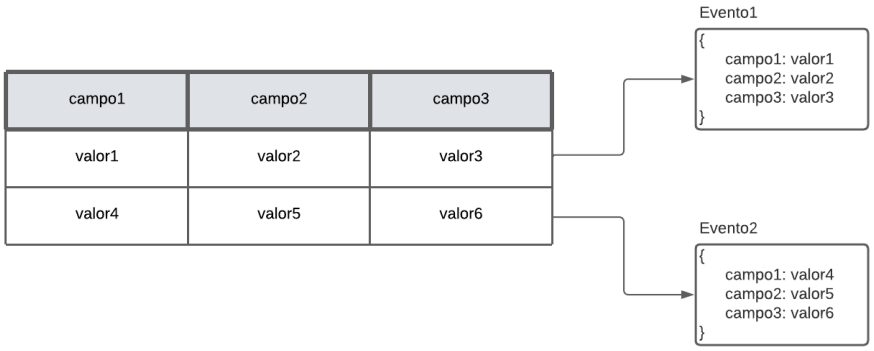


Figura 23. Funcionamiento *plugin JDBC*.

Es necesario definir un *plugin JDBC* por cada consulta que se realice a una tabla, por lo que, en este caso serán necesarios dos *plugins JDBC*, uno para obtener datos de la tabla "*tblprocessweldingdata*" y otro para obtener datos de la tabla "*tbllogbook*" (recordemos que la tabla "*tblmodule*" aporta información complementaria a la tabla "*tblprocessweldingdata*" y no por sí misma.

[11] La sintaxis c*ron* implica cinco o seis campos que representan un conjunto de tiempos, los cuales están separados por espacios en blanco. El formato habitual de la sintaxis Cron es (minuto) (hora) (día del mes) (mes) (día de la semana)
[12] Un evento es un único registro individual de datos, en este caso consiste en un archivo JSON. Es decir, cada fila de la base de datos se convierte en un archivo JSON en el que cada campo es una columna de la tabla en cuestión con su respectivo valor

A continuación, se dará una explicación detallada de la configuración de cada *plugin JDBC*.

```
jdbc {
  id =>"5B_RH"
    jdbc_connection_string => "jdbc:postgresql://19.174.129.12:55551/"
    jdbc_driver_class => "org.postgresql.Driver"
    jdbc_user => "xpe[illegible]"
    jdbc_password => "[illegible]"
    jdbc_driver_library => "C:\IngEq\ELK\LogStash\postgresql\postgresql-42.5.4.jar"
    statement => "
  SELECT  tblprocessweldingdata.*
  FROM tblprocessweldingdata
  WHERE
  processdataid > :sql_last_value AND
  TO_TIMESTAMP((processdate||' '||processtime), 'yyyy-MM-dd HH24:MI:SS') >= CURRENT_TIMESTAMP
  AT TIME ZONE 'CEST' - INTERVAL '2 minutes'
  ORDER BY processdataid ASC
  "
  tracking_column => "processdataid"
  use_column_value => true
  last_run_metadata_path => "C:/IngEq/ELK/LogStash/data/.ExtWeldMeasureProt_HW_weld_5B_RH_pcweld"
  add_field => { "table_name" => "tblprocessweldingdata"}
  add_field => { "database_name" => "public"}
  add_field => { "plant" => "body1" }
  add_field => { "factory" => "0145a" }
  add_field => {
    "index" => "hwh-curvas-prueba"
  }
  add_field => {
    "policy" => "ilm_PowerTools_Body__data"
  }
  tags => ["Prueba","body1","5B_RH","b1_5BRH_pcweld", "parametros"]
  jdbc_fetch_size => 1
  schedule => "* * * * mon-fri Europe/Madrid"
}
```

Figura 24. *Plugin JDBC* para tabla "*tblprocessweldingdata*".

El *plugin* está compuesto por un **ID** que identifica el flujo de entrada de datos, por una serie de **opciones** que podemos dividir en **requeridas** y **complementarias** y por un conjunto de **parámetros** asociados a dichas opciones.

Las opciones **requeridas** son aquellas que son necesarias para que el flujo de entrada exista, si no se definiesen sería imposible tener una ingesta como tal. Las opciones requeridas son:

- ***jdbc_connection_string***: establece la conexión con la base de datos indicando la dirección ip de la máquina donde se encuentra alojada y su puerto[13].
- ***jdbc_driver_class***: indica el nombre de la clase del controlador *JDBC* que se utilizará para conectarse a la base de datos, que en nuestro caso será el de PostgreSQL.
- ***jdbc_driver_library***: apunta a la ruta donde se encuentra la librería[14] del controlador *JDBC* que se utilizará para establecer conexión con la base de datos, obtenida de la página oficial de PostgreSQL.
- ***jdbc_user***: nombre de usuario con el que autenticarse para acceder a la base de datos.
- ***jdbc_password***[15]: contraseña con la que autenticarse para acceder a la base de datos.
- ***statement***: petición SQL realizada a la base de datos. En este caso se extraen todos los campos de la tabla "*tblprocessweldingdata*". Se establecen dos condiciones con el fin de no extraer todas las filas de la tabla, ya que esta contiene del orden de 4 Gb de datos y una petición de este calibre haría colapsar la ingesta. Las dos condiciones son:
    - ***Temporal***: Se establece, gracias a los campos de "*processdate*" y "*processtime*" (ver descripción de la tabla), que el registro más antiguo que se obtenga no sea anterior a dos minutos desde que se realiza la consulta.

[13] En este caso, al estar en la conexión de prueba, se está apuntando a la dirección ip del servidor DXD y al puerto asignado mediante la redirección de puertos.

[14] La librería contiene los archivos del controlador JDBC (por ejemplo, el archivo JAR) y otros recursos requeridos por el controlador (como archivos de configuración).

[15] Omitida por motivos de confidencialidad.

- ***Parámetro "sql_last_value"***: este es uno de los **parámetros** intrínsecos a las opciones del *plugin* que se mencionaban anteriormente. De este parámetro dependen dos más, auxiliares. El primero es "***tracking_column***", en este se apunta al campo "***processdataid***" ya que es el campo clave que diferencia de manera única a cada registro. En segundo lugar, el parámetro "***last_run_metadata_path***" apunta la ruta donde se almacena el valor del parámetro "***sql_last_value***" y de donde se leerá cuando se tenga que hacer uso de este. Con esto, este parámetro apuntará en cada consulta el último valor de la columna asignada en un archivo ubicado en la ruta correspondiente, de esta manera, cada vez que haga una consulta obtendrá registros posteriores al último valor apuntado. En el caso de que este haya sido hace más de dos minutos solo se obtendrán los registros guardados en la base de datos durante ese tiempo (recordar que el programa XPegasus registra los parámetros de cada punto en tiempo real).

- ***jdbc_fetch_size***: establece la cantidad de registros que se extraen por cada operación de consulta. En caso de no definirlo se le asigna un valor por defecto que depende del controlador *JDBC* que se esté utilizando.
- ***schedule***: consiste en una sintaxis *cron* que programa la ejecución de la consulta periódicamente. Permite automatizar el proceso de importación de datos. El formato consiste en una cadena de tiempo en segundos, minutos, horas, días, semanas y meses [22]. En este caso se le indica que debe realizar la consulta cada minuto de lunes a viernes.

Por otra parte, respecto a las opciones **complementarias** encontramos un conjunto de ***add_field*** que consisten en añadir campos nuevos que contextualizarán cada evento desde el inicio, así como el conjunto de etiquetas añadido desde la opción ***tags***.

```
jdbc {
  id =>"5B_RH_logbook"
      jdbc_connection_string => "jdbc:postgresql://19.174.129.12:55551/"
      jdbc_driver_class => "org.postgresql.Driver"
      jdbc_user => "xpegasus"
      jdbc_password => "hwhpegasus"
      jdbc_driver_library => "C:\IngEq\ELK\LogStash\postgresql\postgresql-42.5.4.jar"
      statement => "
    SELECT  tbllogbook.*
    FROM tbllogbook
    WHERE
    logid > :sql_last_value AND
    TO_TIMESTAMP((logdate||' '||logtime), 'yyyy-MM-dd HH24:MI:SS') >= CURRENT_TIMESTAMP
    AT TIME ZONE 'CEST' - INTERVAL '10 minutes'
    ORDER BY logid ASC
  "
  tracking_column => "logid"
  use_column_value => true
  last_run_metadata_path => "C:/IngEq/ELK/LogStash/data/.ExtWeldMeasureProt_HW_weld_5B_RH_logbook_pcweld"
  add_field => { "table_name" => "tbllogbook"}
  add_field => { "database_name" => "public"}
  add_field => { "plant" => "body1" }
  add_field => { "factory" => "0145a" }
  add_field => {
    "index" => "hwh-logbook-prueba"
  }
  add_field => {
    "policy" => "ilm_PowerTools_Body__data"
  }
  tags => ["body1","5B_RH","b1_5BRH_pcweld", "logbook"]
  jdbc_fetch_size => 1
  schedule => "* * * * mon-fri Europe/Madrid"
}
```

Figura 25. *Plugin JDBC* para tabla "*tbllogbook*".

Mediante el último *plugin JDBC* que extrae la tabla "*tbllogbook*" se completa la parte de **entradas**. Esta, como se puede ver, es exactamente igual que la anterior a excepción de la tabla a la que se dirige la consulta, algunos campos de contextualización y algún ***tag***.

Recordando el funcionamiento de Elasticsearch, es notoria la importancia de los campo ***index*** y ***policy***, los cuales cobrarán mayor significado más adelante en el archivo de configuración.

Por lo tanto, la estructura de la entrada del archivo de configuración sería la siguiente:

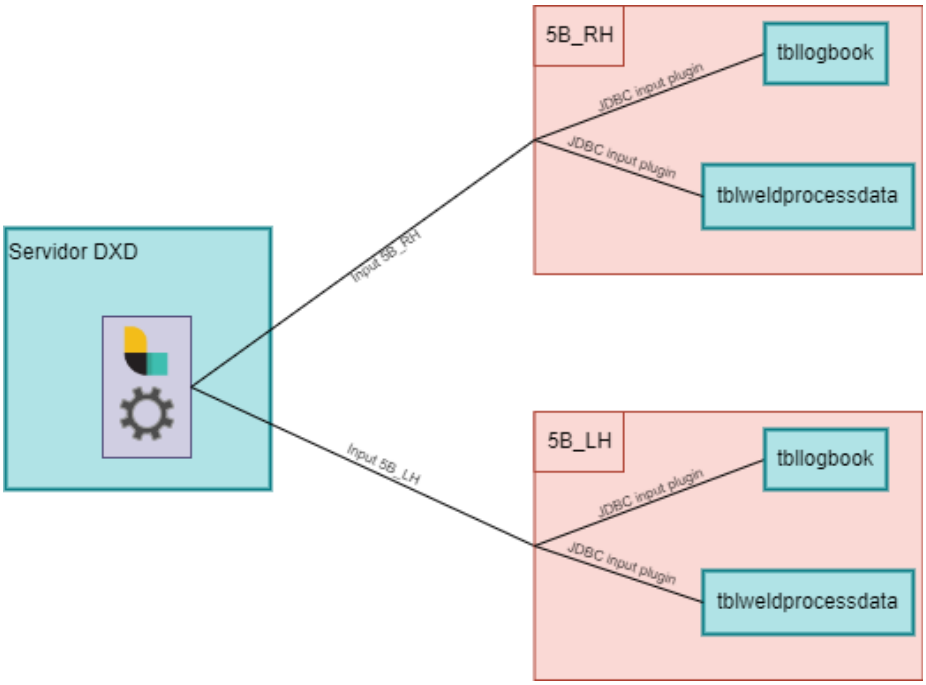


Figura 26. Esquema input.

Cada archivo de configuración se encarga del proceso de ETL de cada una de las bases de datos, es decir, se crea un archivo de configuración distinto para cada ordenador y estos son idénticamente iguales entre sí a excepción de todos los campos que identifican la fuente de los datos (obviamente).

4.2.2 Filtros

Esta es la parte que se encarga de procesar y **transformar** los datos a los que podríamos llamar "crudos" que entran directamente desde la fuente con el fin de darles un mayor sentido y acomodarlos a la visualización que necesitamos de cada uno de ellos. Mediante esta parte es posible *parsear*[16] y manipular los campos obtenidos en el evento, afinando así su significado y obteniendo información más precisa a partir de ellos.

Esta es una parte extensa que contiene un amplio conjunto de *plugins*. Cabe destacar que los estos se encuentran divididos por una gran cláusula *if_else* que aplica según qué filtros en función de si el evento nace de la consulta a la tabla "*tblprocessweldingdata*" o a la tabla "*tbllogbook*" ya que la naturaleza los datos de cada tabla es diferente, así como las visualizaciones que necesitaremos de ellos. El elemento diferenciador de un tipo de evento a otro serán dos etiquetas (*tag*) con el valor "*logbook*" o "*parámetros*".

```
filter{
if "parametros" in [tags] {
```

Figura 27. Claúsula *if* parámetros.

[16] Término adoptado del inglés que se refiere al proceso de analizar una cadena de texto o datos en busca de patrones para extraer información específica.

```
if "logbook" in [tags] {
```

Figura 28. Claúsula *if logbook*.

A continuación, se desglosará la descripción de los *plugins* en base a la división recién explicada.

4.2.2.1 Filtros "tblprocessweldingdata"

Este conjunto de *filter plugins* se encuentra dividido en subgrupos divididos en subgrupos en función del campo del evento que se pretende **transformar**, creando de esta manera campos que son producto de la unión o derivación de los campos del evento original. El desglose por secciones es el siguiente:

- ***Timestamp***: todo evento se crea automáticamente con el campo *timestamp* o "sello temporal". Este indica el momento en el que el evento fue creado, es decir, ingerido en el canal y utiliza el formato "yyyy-MM-ddTHH:mm:ss.SSSZ" y se considera un campo estándar que es de gran utilidad a la hora de representar datos históricos en visualizaciones. No obstante, el evento creado proviene de un registro que ya posee su propio sello temporal dividido en los campos *processdate* y *processtime* (como se ha visto en la explicación de la estructura de la base de datos), por lo tanto, en esta sección se pretende diferenciar ambos sellos temporales. El objetivo es asignar al campo estándar de Logstash *timestamp* el sello temporal original de la base de datos unificando los campos mencionados anteriormente y que el sello temporal predefinido automáticamente pase a llamarse *processed_timestamp*, de esta manera en visualizaciones futuras, los parámetros históricos serán fieles al momento exacto en el que se ha soldado un punto y no de cuando estos han sido ingeridos en la base de datos.

```
mutate {
    add_field => {
        "timestamp" => "%{processdate} %{processtime}"
    }
    rename => {
        "@timestamp" => "processed_timestamp"
    }
}
date {
    match => ["timestamp", "yyyy-MM-dd HH:mm:ss"]
    remove_field => ["timestamp", "processdate", "processtime"]
}
```

Figura 29. Sección *timestamp* de la tabla "*tblprocessweldingdata*".

Como es visible, se hace uso de dos *plugins*:

  - ***mutate*** [23]: Permite realizar mutaciones generales en los campos como renombrar, reemplazar y modificar campos a través de diversas opciones. En este se hace uso de la opción genérica[17] *add_field* creando un campo auxiliar homónimo[18] al campo estándar *timestamp* donde se concatenan los campos *processdate* y *processtime* y se renombra el sello temporal estándar a *processed_timestamp*.
  - ***Date*** [24]: Gracias a este plugin es posible redefinir el campo estándar a partir del campo auxiliar creado previamente mediante la opción *match*, introduciendo como parámetros de dicha opción el nuevo sello temporal y el formato en el que se encuentra escrito, para que el *plugin* automáticamente guarde el valor con el

[17] Opción que se puede utilizar dentro de casi cualquier filtro.
[18] El sello temporal estándar se diferencia mediante el carácter "@" al principio.

formato correcto en el campo estándar. Finalmente se eliminan los campos innecesarios.

- ***plugin de Ruby***: consiste en utilizar código Ruby personalizado en los eventos que se procesan mediante Logstash. Esta es una manera de crear *plugins* propios ya que permite modificar los eventos a bajo nivel [25]. En esta sección se procesan los datos referentes a los parámetros "seleccionables" ya que, como se ha visto, cada uno de ellos tiene asociado un campo identificativo el cual nos da el nombre de dicho parámetro (recordar explicación base de datos).

  El proveedor aporta en su documentación un archivo de texto donde encontramos todos los pares clave-parámetro a modo de diccionario.

  Este filtro acepta código Ruby insertado directamente o un archivo Ruby externo. Las dos opciones se excluyen mutuamente y tienen modos de funcionamiento ligeramente diferentes. Por un lado, la **inserción directa de código** consiste en la ejecución directa de aquello que se escriba dentro del *plugin.*

```
filter {
  ruby {
    # Cancel 90% of events
    code => "event.cancel if rand <= 0.90"
  }
}
```

Figura 30. Ejemplo de inserción de código ruby[19].

  Por otro lado, cuando el código a ejecutar es más complejo, conviene hacer uso de un **archivo *script*[20] de Ruby**, utilizando la opción *path.*

```
filter {
  ruby {
    # Cancel 90% of events
    path => "/etc/logstash/drop_percentage.rb"
    script_params => { "percentage" => 0.9 }
  }
}
```

Figura 31. Introducción de un archivo *script*.

  Como se ve, mediante esta opción se indica la ruta donde se encuentra el archivo a ejecutar. Asimismo, mediante la opción *script_params* es posible definir parámetros a utilizar dentro del *script*.

  El archivo de *script* Ruby debe definir los siguientes métodos:

  - ***register_params***: Método de registro opcional que recibe el *hash*[21] clave/valor pasado en la opción de configuración *script_params*.
  - ***filter(event)***: Método Ruby obligatorio que acepta un evento Logstash y debe devolver un vector de eventos.

[19] Mediante este código se cancelan el noventa por ciento de los eventos creados.
[20] Se refiere a un programa o conjunto de instrucciones que se escriben para automatizar tareas o realizar una secuencia de operaciones específicas.
[21] Estructura de datos que permite almacenar información en pares clave-valor, en la que se puede acceder a los valores a través de una clave única.

```ruby
# the value of `params` is the value of the hash passed to `script_params`
# in the logstash configuration
def register(params)
        @drop_percentage = params["percentage"]
end

# the filter method receives an event and must return a list of events.
# Dropping an event means not including it in the return array,
# while creating new ones only requires you to add a new instance of
# LogStash::Event to the returned array
def filter(event)
        if rand >= @drop_percentage
                return [event]
        else
                return [] # return empty array to cancel event
        end
end
```

Figura 32. Archivo *script* de Ruby.

Esta sección está dividida en dos partes: una para el procesado e identificación de las curvas (*curves*) mediante el archivo *script* de Ruby ***curva.rb*** y la otra para la identificación de los parámetros (*values*) mediante el archivo *script* de Ruby ***parameters.rb***:

- ***curves***: las curvas de soldadura se guardan en un formato específico en la base de datos SQL y al extraerlas, es necesario transformarlas para que estas sean representables en la herramienta de visualización. El software de XPegasus obtiene los datos de las curvas de soldadura del punto que se acaba de realizar en el tiempo establecido en el parámetro "tiempo de soldadura efectivo" (ver listado de parámetros) y los almacena en el parámetro correspondiente como una cadena de texto en hexadecimal, en palabras de 2 bytes y *little-endian*[22]. Por tanto, es necesario transformar dicha cadena en un vector, reordenando cada elemento para deshacer el formato *little-endian*:

```ruby
def filter(event)
  (1..7).each do |i|
    unless event.get("curve"+i.to_s).nil?
      array = event.get("curve"+i.to_s).chars.each_slice(2).map(&:join)
      array2 = []
      x=1
      array3 = []
      (18..array.size-2).step(2).each do |j|
        array2.push((array[j+1].concat(array[j])).to_i(16))
        array3.push(x)
        x += 1
      end
```

Figura 33. *Parseo* de las curvas.

Se define un bucle de siete iteraciones ya que, como se ha visto en la descripción de la base de datos, es el máximo número de curvas que el programa permite alojar. Dentro del bucle, antes que nada y siempre que el campo no sea nulo, se definen las siguientes variables: el vector ***array***, siendo cada elemento del vector un fragmento de dos caracteres de la cadena de texto de la curva que se está procesando, el vector ***array2***, el vector ***array3*** y el entero ***x***. Posteriormente se define un segundo bucle dentro del anterior

[22] Formato de almacenamiento de datos donde el byte menos significativo se almacena primero en la memoria seguido por los bytes más significativos.

donde, en cada iteración (empezando por la posición decimoctava[23] de ***array***):

- Se escribe una palabra reordenada en ***array2*** y se transforma cada byte a decimal.
- Se escribe la variable ***x*** en la posición correspondiente de ***array3***.
- Se incrementa la variable ***x***.

Por tanto, se deduce que: ***array*** sirve como vector auxiliar para aplicar las transformaciones pertinentes, ***array2*** es el vector final que obtendremos de la curva en cuestión y ***array3*** es el vector de tiempo, donde cada elemento representa un milisegundo del tiempo de soldadura, formado gracias al incremento iterativo de la variable ***x***.

Posteriormente, dentro del mismo bucle inicial, se identifica el nombre de la curva a partir del campo identificativo (*hwhidcurve*) y se define el campo en el evento:

```
case event.get("hwhidcurve"+i.to_s)
when 14055
  event.set("curr_curve_unit", "A")
  event.set("curr_curve", array2)
when 14057
  event.set("vol_curve_unit","mV")
  event.set("vol_curve", array2)
when 14740
  event.set("filtered_r_unit","uOhms")
  event.set("filtered_r", array2)
when 14095
  event.set("Actuating_value_curve_unit","Skt")
  event.set("Actuating_value_curve", array2)
when 0
  #do nothing
else
  event.set("curve_"+i.to_s+"_error", "hwhid not found")
end
end
```

Figura 34. Identificación de curva.

Mediante el uso de una cláusula case se asigna el nombre a la curva procesada en la iteración en función del valor del campo *hwhidcurve* correspondiente. Como se ha visto previamente, esto es posible gracias al archivo de texto proporcionado pro Harms & Wende donde se asocia cada valor posible de los campos identificativos al campo que corresponde.

Por último, se eliminan los campos correspondientes a la curva y a su identificativo previos al procesado y se define el vector de tiempo en el evento.

```
    event.remove("curve"+i.to_s)
    event.remove("hwhidcurve"+i.to_s)
    event.set("time_vector"+i.to_s, array3)
  end
  return[event]
end
```

Figura 35. Final de *curva.rb*

[23] Esto se debe a que todas las curvas poseen un prefijo identificativo que resulta inútil para el cometido de este proceso.

Por lo tanto, dentro del archivo de configuración solo será necesario definir la opción *path* del *plugin* de Ruby.

```
ruby {
    path => "C:\IngEq\ELK\Logstash\scripts\curva.rb"
}
```

Figura 36. *Plugin* de Ruby en archivo de configuración.

values: este *script* es idéntico a la segunda parte del *script* anterior. Se resume en una cláusula case donde se asocia cada posible *hwhidvalue* con el nombre del parámetro en cuestión. Es reutilizable en caso de que se quiera obtener valores de alguna de las dos familias de parámetros inutilizadas (*boolvalues* y *texts*) en un futuro, por lo tanto, será necesario indicar el número de iteraciones del bucle como *script_param* para después utilizarlo como variable local dentro del *script*. El número de iteraciones viene definido por la cantidad de campos y esta es distinta según la familia de los parámetros, como se ha visto en la descripción de la base de datos. Se ha optado por esta opción debido a que se está tratando con un número reducido de datos y, por tanto, una cláusula case permite realizar la asociación rápidamente sin ocupar excesivo espacio de almacenamiento.

```
def register(params)
  @iter = params["iter"]
  @params = params["param"]
end
```

Figura 37. Definición de *script_params.*

En primer lugar, se define la función *register(params)*, mediante la que se obtiene el número de iteraciones y la familia de parámetros que se va a procesar.

```
def filter(event)
  (1..@iter).each do |i|
    case event.get("hwhid"+@params+i.to_s)
    when 608
      event.set("mil_cyc_coun", event.get(@params+i.to_s))
    when 14031
      event.set("mil_spt_coun", event.get(@params+i.to_s))
    when 14219
      event.set("act_wel_time", event.get(@params+i.to_s))
    when 14185
      event.set("mean_act_vol", event.get(@params+i.to_s))
    when 14181
      event.set("mean_isi", event.get(@params+i.to_s))
    when 14216
      event.set("mean_nom_i", event.get(@params+i.to_s))
    when 14234
      event.set("i_tol_+_mv", event.get(@params+i.to_s))
    when 14235
      event.set("i_tol_-_mv", event.get(@params+i.to_s))
    when 14745
      event.set("resistance", event.get(@params+i.to_s))
    when 16196
      event.set("mean_nom_r", event.get(@params+i.to_s))
    when 16197
      event.set("r_tol_+_mv", event.get(@params+i.to_s))
    when 16198
      event.set("r_tol_-_mv", event.get(@params+i.to_s))
    when 14459
      event.set("sptimstm", event.get(@params+i.to_s))
    when 14090
      event.set("pts_piece1", event.get(@params+i.to_s))
    when 14049
      event.set("reg_num", event.get(@params+i.to_s))
    when 0
      #do nothing
    else
      event.set(@params+"_"+i.to_s+"_error", "hwhid_not_found")
    end
    event.remove("hwhid"+@params+i.to_s)
    event.remove(@params+i.to_s)
  end
  return[event]
end
```

Figura 38. Cláusula case de *parameters.rb*.

Por último, se define la asignación de la misma forma que en el *script* explicado anteriormente gracias al archivo de texto del proveedor ya mencionado. Por lo tanto, el *plugin* quedará definido de la siguiente manera dentro del archivo de configuración:

```
ruby {
    script_params => {
        "iter" => 15
        "param" => "value"
    }
    path => "C:\IngEq\ELK\Logstash\scripts\parameters.rb"
}
```

Figura 39. Definición de *plugin* de Ruby de *values* en archivo de configuración.

- ***Enriquecimiento con tabla auxiliar tblmodule*** [26]: Para esta sección se hará uso del *plugin JDBC_static*. Este es el más adecuado para enriquecer eventos con datos de referencia que son estáticos o no cambian muy a menudo como, en este caso, sucede con la tabla *tblmodule*, ya que esta únicamente indica el nombre de control y grupo de controles conectados al ordenador de soldadura. Funciona obteniendo datos de una base de datos remota, almacenándolos en caché en una base de datos local Apache Derby en memoria, y utilizando búsquedas para enriquecer eventos con datos almacenados en caché en la base de datos local. Para definir el filtro es necesario tres secciones principales:
    - ***local_db_objects***: Esta opción se encarga de definir las columnas, tipos e índices utilizados para construir la estructura de la base de datos local. Los nombres y tipos de las columnas deben coincidir con los de la base de datos externa.

```
local_db_objects => [
  {
    name => "operations"
    index_columns => ["moduleid"]
    columns => [
      ["moduleid", "INTEGER"],
      ["modulename", "varchar(255)"],
      ["modulegroup", "varchar(255)"],
      ["connectionstring", "varchar(50)"]
    ]
  }
]
```

Figura 40. Opción *local_db_objects*.

    - ***loaders***: esta opción se encarga de realizar la consulta a la base de datos externa para obtener el conjunto de datos que se almacenará en la caché local.

```
loaders => [
  {
    id => "operation_data"
    query => "
      SELECT moduleid, modulename, modulegroup, connectionstring from tblmodule
    "
    local_table => "operations"
  }
]
```

Figura 41. Opción *loaders*.

    - ***local_lookups***: mediante esta opción se realizan consultas a la base de datos local, ya cargada con los datos de la base de datos externa, para así enriquecer los eventos.

```
local_lookups => [
  {
    id => "local-operations"
    query => "SELECT * FROM operations WHERE moduleid = :moduleid"
    parameters =>{
      moduleid => "[moduleid]"
    }
    target => "operation"
  }
]
```

Figura 42. Opción *local_lookups.*

Como se ve, la clave foránea *moduleid* es la que asocia los datos correspondientes de la tabla *tblmodule* con el evento pertinente.

- ***Opciones adicionales***: Por último, se toman los datos del objeto JSON y se almacenan en campos del evento de nivel superior para facilitar el análisis en Kibana y se eliminan los campos innecesarios.

```
add_field => { modulename => "%{[operation][0][modulename]}" }
add_field => { modulegroup => "%{[operation][0][modulegroup]}" }
add_field => { connectionstring => "%{[operation][0][connectionstring]}" }
remove_field => ["operation", "moduleid"]
```

Figura 43. Opciones adicionales.

- ***Ejecución de la consulta***: a continuación, se definen las opciones necesarias para obtener los datos que nutrirán la base de datos local de forma similar a como se ha visto en *JDBC input plugin*.

```
staging_directory => "C:/IngEq/ELK/LogStash/data/import_data.B1_5B_RH"
# run loaders every 2 hours
loader_schedule => "* */2 * * *"
jdbc_connection_string => "jdbc:postgresql://19.172.160.133:5550/"
    jdbc_driver_class => "org.postgresql.Driver"
    jdbc_driver_library => "C:\IngEq\ELK\LogStash\postgresql\postgresql-42.5.4.jar"
    jdbc_user => "xpegasus"
    jdbc_password => "hwhpegasus"
```

Figura 44. Opciones para ejecutar consulta.

Además de las opciones ya vistas, también se define la ruta donde se almacenará la caché local mediante *staging_directory.*

- ***Clasificación del modo de regulación***: si se revisa la selección de parámetros se aprecia que existe un parámetro llamado "modo de regulación", cuyo funcionamiento y explicación se ha abordado en los conceptos previos a la selección de parámetros. El valor asignado por el software de soldadura es numérico, por lo que será necesario procesar este campo para asignar a este campo una cadena de texto que indique exactamente el modo de regulación con el que se ha soldado el punto en cuestión.

  Mediante comprobaciones en planta junto con el equipo de soldadura se obtiene que:

  - Si el valor del campo que se ha guardado es "1" el punto se soldó en modo SKT.
  - Si el valor del campo que se ha guardado es "2" el punto se soldó en modo KSR.
  - Si el valor del campo que se ha guardado es "4" el punto se soldó en modo IQR.

  Para conseguir intercambiar el valor numérico es necesario seguir dos pasos:

  - En primer lugar, es necesario obtener el número como entero. Esto se debe a que, aunque el valor del campo sea numérico, este es del tipo cadena de texto. Para ello se hará uso del *plugin grok* [27], cuya función es identificar la cadena de

texto como un número y guardarlo en un campo a parte para después poder transformar el tipo de ese campo a entero.

```
grok {
  match => ["reg_num","%{NUMBER:num}" ]
}
mutate {
  convert => { "num" => "integer" }
}
```

Figura 45. Obtención del número de modo de regulación.

- Una vez obtenido el número y sabiendo a qué modo se corresponde cada valor, es sencillo, mediante una cláusula *if_else* obtener un campo del tipo cadena de texto que indique el modo de regulación correct.

```
if [num] == 4 {
  mutate {
    add_field => {
      "reg" => "IQR"
    }
  }
}
else if [num] == 1{
  mutate {
    add_field => {
      "reg" => "SKT"
    }
  }
}
else{
  mutate {
    add_field => {
      "reg" => "KSR"
    }
  }
}
```

Figura 46. Cláusula *if_else* del modo de regulación.

- ***Clonación*** [28]: en esta sección se hace uso del *plugin clone* que sirve para duplicar eventos. Con esto se busca crear dos flujos de salida distintos a partir de la misma ingesta con el objetivo de, a partir del mismo evento, crear dos distintos donde se separen curvas del resto de parámetros. Esto se debe a que las curvas ocupan una gran cantidad de espacio, por lo que, se busca no saturar de información el servidor, a la vez que se pretende mantener a disposición aquellos parámetros distintos a las curvas el máximo tiempo posible. Gracias este *plugin* es posible almacenar los eventos con curvas y los eventos sin curvas en dos índices distintos (ver explicación de Elasticsearch), sobre los cuales, se aplicará una política de almacenamiento distinta. No obstante, se profundizará sobre este aspecto al final de este apartado del desarrollo.

  En primer lugar, es necesario establecer los dos flujos mediante la lista de clonación:

```
clone {
    clones => ["hwh-curvas","hwh-no-curvas"]
    ecs_compatibility => "disabled"
}
```

Figura 47. *plugin clone*[24].

[24] La opción *ecs_compatibility* afecta al formato en el que se guardan los eventos y se deja desactivada por cuestiones de configuración de la base de datos central.

La opción *clones* establece una lista mediante la cual se diferenciará un flujo del otro, añadiendo el campo *type* a cada evento clonado con el valor correspondiente en cada uno de ellos.

Posteriormente, mediante una cláusula *if_else* se aplicarán las modificaciones pertinentes en función del flujo, es decir, en función del campo *type* asignado al evento:

```
if [type] == "hwh-no-curvas" {
    prune {
        blacklist_names => ["curve","curr_curve","vol_curve", "filtered_r", "Actuating_value", "time_vector"]
    }
    mutate {
        replace => {
            "index" => "hwh-no-curves"
        }
    }
}
else{
    mutate {
        replace => {
            "policy" => "ilm_hwh_curves"
        }
    }
}
```

Figura 48. Modificaciones a los clones.

En el caso del flujo correspondiente a los parámetros sin curvas se hace uso del *plugin prune* [29], el cual, mediante la opción *blacklist_names*, se encarga de eliminar los campos de la lista. En este caso vemos como se eliminan todos los campos correspondientes a curvas y al vector de tiempos. Asimismo, se cambia el valor del campo *index*, añadido en la entrada, el cual indica el índice de Elasticsearch al que se pretende enviar el evento.

Por otro lado, si el evento corresponde al flujo de las curvas, únicamente se cambia el valor del campo añadido en la entrada *policy*, el cuales deben ser las políticas de almacenamiento de las que se ha hablado recientemente y que se explicarán al final de este apartado del desarrollo.

- ***Campo "línea"***: se pretende dividir la cadena de texto del campo *modulegroup*, ya que esta identifica un grupo de controles de soldadura a partir de un prefijo, que indica la línea, y un identificativo del grupo (irrelevante a nivel de visualización por sí solo). El objetivo es obtener un campo donde únicamente se indique la línea.

```
grok {
    match => {
        "modulegroup" => "/%{WORD:linea}/%{WORD:control}"
    }
}
```

Figura 49. Obtención del campo línea

Mediante el *plugin grok* se analiza el patrón del campo *modulegroup* y se extrae la línea como un campo a parte gracias a la opción *match*.

Con esto concluye la estructura de la parte de ***filtros*** para la ingesta de la tabla "***tblprocessweldingdata***". A continuación, se verá la sección correspondiente a la ingesta de la tabla "***tbllogbook***".

4.2.2.2 Filtros "tbllogbook"

En esta parte del condicional de los **filtros** que afectan a la tabla "*tbllogbook*" se sigue la misma lógica que en el subapartado anterior de dividir el conjunto de *plugins* en secciones en función de la transformación o procesamiento realizado. A continuación, se describe esta parte desglosada en secciones:

- ***Timestamp***: Como se ve a continuación, la sección referente al sello temporal es idéntica a la explicada en el subapartado anterior con la única diferencia del nombre de los campos originales utilizados:

```
mutate {
    add_field => {
        "timestamp" => "%{logdate} %{logtime}"
    }
    rename => {
        "@timestamp" => "processed_timestamp"
    }
}
date {
    match => ["timestamp", "yyyy-MM-dd HH:mm:ss"]
    remove_field => ["timestamp", "logdate", "logtime"]
}
```

Figura 50. Sección *timestamp* de la tabla "*tbllogbook*"

- ***Estandarización del formato de los campos***: Recordando la descripción de la tabla "*tbllogbook*", se tiene los campos "*logtype*", el cual nos dirá el tipo de mensaje de registro que se trata, y "*logtext*", donde viene el mensaje de registro en cuestión. Asimismo, tenemos el conjunto de veinte campos "*logparam*". Con esto, tenemos el grueso de la información que se necesita del diario de registro. Todos los campos mencionados vienen especificados por valores numéricos que, por lo general, representan un código, a excepción (en el caso de los "*logparam*") de que este valor numérico se trate de una magnitud o tenga significado por sí mismo, cosa que se sabe en función del tipo de mensaje de registro (es decir, del campo "*logtype*"). Estos códigos funcionan de manera idéntica a los campos de identificación de los parámetros "seleccionables" vistos en la descripción de campos de la tabla "*tblprocessweldingdata*", es decir, del mismo archivo de texto (mencionado anteriormente) donde se identifican estos parámetros "seleccionables", se identifican también estos códigos de la tabla "*tbllogbook*". En este caso específico, será necesario que los campos descritos sean una cadena de texto y contengan únicamente caracteres numéricos para poder llevar a cabo la identificación de los mismos a partir del código obtenido. Por ello se requiere una estandarización realizada como se ve a continuación:

```
mutate {
    gsub => [
        "logparam1","@","",
        "logparam2","@","",
        "logparam3","@","",
        "logparam4","@","",
        "logparam5","@","",
        "logparam6","@","",
        "logparam7","@","",
        "logparam8","@","",
        "logparam9","@","",
        "logparam10","@","",
        "logparam11","@","",
        "logparam12","@","",
        "logparam13","@","",
        "logparam14","@","",
        "logparam15","@","",
        "logparam16","@","",
        "logparam17","@","",
        "logparam18","@","",
        "logparam19","@","",
        "logparam20","@",""
    ]
}
```

Figura 51. Estandarización de formato de los campos *logparam*.

Se hace uso de la opción *gsub* [23] del *plugin mutate*, que compara una expresión regular[25] (en este caso se trata del carácter "@") con el valor de un campo y sustituye todas las coincidencias por una cadena de sustitución (en este caso se elimina el carácter no incluyendo nada entre las comillas). Esto se debe a que, con frecuencia, los campos "*logparam*" contienen este carácter al principio del código con el fin de indicar que se trata de una referencia a dicho código, no obstante, es necesario eliminarlo para poder realizar el procesado que identificará el código con el texto pertinente.

```
mutate {
    convert => {
        "logtype" => "string"
    }
    convert => {
        "logtextid" => "string"
    }
}
```

Figura 52. Conversión de los campos *logtype* y *logtextid*

Mediante la opción *convert* [23] del *plugin mutate* se convierten los campos *logtype* y *logtextid* en cadenas de textos ya que se obtienen como enteros desde base de datos original.

- ***Identificación de campos***: se emplea el *plugin* de Ruby introduciendo la ruta a un *script* para cada proceso de identificación como se ha explicado previamente.

```
ruby {
    path => "C:\IngEq\ELK\Logstash\scripts\logreader.rb"
}

ruby {
    path => "C:\IngEq\ELK\Logstash\scripts\logparams.rb"
}
```

Figura 53. Definición de *plugins* de Ruby para identificación de campos de "*tbllogbook*"

Se deberá tener en cuenta todos los posibles mensajes de registro que se pueden dar, por lo que el número de códigos a identificar asciende considerablemente y la opción de utilizar una cláusula *case* en los *scripts* de Ruby como en el caso anterior es claramente inviable.

Para llevar a cabo el proceso de identificación se necesitará hacer uso del archivo de texto proporcionado por Harms & Wende en su totalidad, transformándolo en un *hash*. Para conseguirlo es necesario transformar este archivo de texto en un archivo *yaml*[26] que sirva como *hash* en nuestro *script*. Existe un paso intermedio para conseguir esto y es transformar en primer lugar el archivo de texto a un archivo *csv*[27], para posteriormente ejecutar el siguiente *script* de Ruby consiguiendo así el archivo *yaml* deseado.

[25] Una expresión regular es una secuencia de caracteres que se utiliza para describir un patrón en un conjunto de texto. Se utilizan comúnmente para buscar, reemplazar, validar o extraer ciertas porciones de texto en una cadena de caracteres.

[26] El formato YAML se utiliza comúnmente para escribir archivos de configuración, pero también se puede utilizar para almacenar y transferir datos estructurados.

[27] Valores separados por comas.

```
require 'csv'
require 'yaml'

csv_file = 'C:/IngEq/ELK/Logstash/scripts/diccionario.csv'
yaml_file = 'C:/IngEq/ELK/Logstash/scripts/diccionario.yaml'

csv_data = CSV.read(csv_file, headers: true).map(&:to_h)
File.open(yaml_file, 'w') { |file| file.write(YAML.dump(csv_data)) }
```

Figura 54. *Script* de obtención de archivo *yaml*[28].

Como se ve en la imagen anterior el proceso de identificación de todos estos parámetros se divide en dos *scripts*.

```
require 'yaml'
def filter(event)
    diccionario = YAML.load_file('C:/Users/ACARCELM/Documents/logstash-8.5.1/config/diccionario.yaml')
    event.set("logtype", diccionario[event.get("logtype")])
    event.set("logtextid", diccionario[event.get("logtextid")])
    return[event]
end
```

Figura 55. *Script logreader.rb*.

En el *script logreader.rb* se abordan los campos *logtext* y *logtype*. El funcionamiento es bastante sencillo: se carga el archivo *yaml* obtenido y se obtiene el texto introduciendo el valor del campo correspondiente.

```
require 'yaml'
def filter(event)
  diccionario = YAML.load_file('C:/Users/ACARCELM/Documents/logstash-8.5.1/config/diccionario.yaml')
    case event.get("logtype")
    when " Message"
        event.set("logparam3", diccionario[event.get("logparam3")])
    when " Parameter modification"
        event.set("logparam2", diccionario[event.get("logparam2")])
        event.set("logparam9", diccionario[event.get("logparam9")])
        event.set("logparam8", diccionario[event.get("logparam8")])
    when " Debug"
        event.set("logparam2", diccionario[event.get("logparam2")])
        event.set("logparam3", diccionario[event.get("logparam3")])
        event.set("logparam4", diccionario[event.get("logparam4")])
    else
    end
  return[event]
end
```

Figura 56. *Script logparams.rb*.

El *script logparams.rb* funciona exactamente igual, pero añadiendo una cláusula *case*, ya que, en este caso, no todos los campos representarán un código, a pesar de ser un valor numérico, sino que, en función del tipo de mensaje, algunos serán valores numéricos por sí mismos de magnitudes o parámetros.

- ***Campo "línea"***: tanto su objetivo como su funcionamiento es exactamente igual a la sección homónima de la ingesta de la tabla "*tblprocessweldingdata*" cambiando el campo *modulegroup* por el campo *modulename*.

4.2.3 Salidas

Los documentos JSON, también llamados eventos, se envían directamente a una instancia de Elasticsearch del servicio de Google Cloud contratado por la empresa. Para ello, en la parte de salidas se hace uso únicamente del *plugin* Elasticsearch [30]. En este es necesario definir las siguientes opciones:

- ***ID de la ingesta***: es, sencillamente, un identificativo del origen de los datos que se envían.

[28] Este *script* solo es necesario ejecutarlo una sola vez, ya que una vez obtenido el archivo *yaml* este se puede utilizar para todos los *scripts* que se necesite.

- ***ECS compatibility***: deshabilitada.
- ***Manage_template***: esta opción determina si Logstash configura las plantillas[29] de Elasticsearch automáticamente. Para este caso, la opción se encuentra **deshabilitada** ya que se busca obtener mayor control sobre los índices creados.
- ***Host***: establece la dirección y el puerto destino de la instancia de Elasticsearch sobre la que se enviarán los documentos.
- ***User y password***[30]: autenticación necesaria para poder enviar los archivos.
- ***ILM (gestión del ciclo de vida de los índices)***: Elasticsearch permite aplicar políticas sobre el ciclo de vida de los índices para optimizar el mantenimiento del *cluster* y no sobrepasar la capacidad de almacenamiento contratada. Este último bloque de opciones es clave para llevar a cabo una gestión correcta y aplicar las políticas correspondientes:
    - ***ilm_rollover_alias***: alias en el que se escribirán los índices gestionados mediante las políticas de ILM definidas. Se trata, en definitiva, el índice, definido por el campo *index* visto en las secciones de **entrada** y **filtros**.
    - ***ilm_pattern***: el valor especificado en el patrón se añadirá al alias de escritura y se incrementará automáticamente cuando ILM cree un nuevo índice.
    - ***ilm_policy***: define la política de ILM asignada al *ilm_rollover_alias*. Existen dos políticas distintas de gestión del ciclo de vida de los índices, la primera para los índices de los eventos con curvas, más agresiva con la renovación de índices, y la segunda para los índices de los eventos sin curvas, más permisiva. La confección de estas políticas viene dada y no se ha abordado su confección en este proyecto. La asignación de esta opción depende del campo *policy* visto en las secciones de **entrada** y **filtros**.

a

```
output{

  elasticsearch {
  id => "hwh_5BLH_to_gcp"
  ecs_compatibility => disabled
  manage_template => false
  hosts => ["https://fc48f47a40d145328e1aa5fe1dd9db4c.psc.us-central1.gcp.cloud.es.io:9243"]
    user => "prx_acarcelm"
    password => "G0_wPfj;697xQ4F0IkG3G9d5-_Mq$b0X"
  ilm_rollover_alias => "%{index}"
    ilm_pattern => "000001"
    ilm_policy => "%{policy}"
  }
}
```

Figura 57. Sección de salida del archivo de configuración.

[29] Una plantilla es un archivo en Elasticsearch que define cómo se indexarán los documentos para un índice determinado.

[30] Censurados por confidencialidad.

4.3 Desarrollo del *Dashboard*

Llegados a este punto se tiene un proceso de ETL satisfactorio en la conexión de prueba establecida, por lo que únicamente restaría obtener una representación visual de los datos que resolviese la problemática planteada en los objetivos del proyecto para, finalmente, establecer todas las conexiones posibles y desplegar el sistema ideado en producción.

Dentro de este apartado se verá cómo y qué herramientas se han utilizado para la confección del *dashboard* de visualización, pero no se abordará su estructura general y su explicación detallada ya que eso se reserva para el apartado de resultados.

4.3.1 Vega lite

Se ha utilizado **Vega lite** para visualizar las curvas de soldadura dado el vector de valores de cada curva y un vector de tiempo. Esto permite ver en detalle las diferentes curvas de cada punto realizado en tiempo real. El procedimiento es el siguiente:

```
  format: {property: "hits.hits[0]_source"}


}

"transform": [
  {"flatten": ["curr_curve", "vol_curve", "filtered_r",
    "Actuating_value_curve", "time_vector1"]}
  {"fold": ["curr_curve"], "as": ["colcur", "current"]},
  {"fold": ["vol_curve"], "as": ["colvol", "voltage"]},
  {"fold": ["filtered_r"], "as": ["colfil", "filtered res"]},
  {"fold": ["Actuating_value_curve"], "as": ["colact",
    "actuating value"]},
  {"calculate": "slice(datum.colcur, -255)", "as": "colc"},
  {"calculate": "slice(datum.colvol, -255)", "as": "colv"},
  {"calculate": "slice(datum.colfil, -255)", "as": "colf"},
  {"calculate": "slice(datum.colact, -255)", "as": "cola"},
]
```

Figura 58. Declaración inicial Vega Lite.

Como se puede ver, se describe la representación de datos mediante la confección de un documento JSON. En primer lugar, mediante el objeto *format*, se define la fuente de la que obtener los datos a representar, en este caso, se trata del objeto *_source* dentro de los archivos JSON alojados en la base de datos ya que es donde se encuentran los datos **extraídos** y **transformados**. A partir de ahí, se declaran las curvas a representar a partir de los vectores, es decir, de los campos presentes en el objeto *_source* (datos obtenidos por el proceso de ETL). De esta manera la propia herramienta de visualización se encargará de representar los cuatro vectores como curvas distintas asignándoles un color de manera automática.

```
"layer":[
  {
    mark: line
    // "encoding" tells the "mark" what data to use and in
      what way.  See https://vega.github.io/vega-lite/docs/
      encoding.html
    encoding: {
      x: {
        field: time_vector1
        type: quantitative
      }
      y: {
        // The "doc_count" is the count per bucket.  Use it
          for Y axis.
        field: curr_curve
        type: quantitative
      //type: quantitative
        axis: {title: false}
      }
      color: {
        field: colc
        type: nominal
      }
    }
  }
```

Figura 59. Descripción de una de las curvas Vega Lite.

Posterior a esto, se define cada una de las curvas dentro del vector *layer* para que las cuatro aparezcan superpuestas en la misma gráfica y se establece como eje x el vector de tiempo. Se repite el mismo proceso hasta completar el vector *layer*, obteniendo el siguiente resultado:

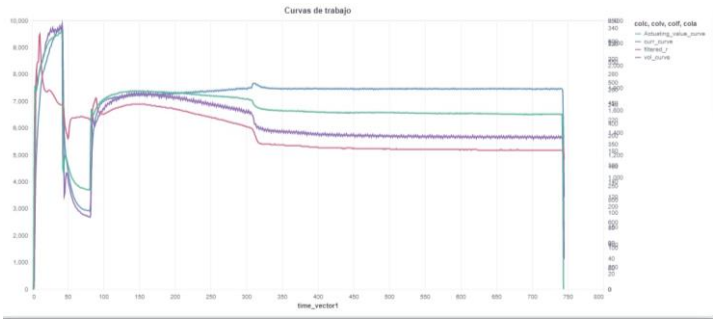


Figura 60. Representación de las curvas.

Adicionalmente, mediante lenguaje *markdown*[31], se obtiene una tabla con las métricas más significativas de cada curva:

Curve	Mean	StDev	qua1	qua2	qua3
Current	8,057.971	1,580.108	7,567.602	8,323.431	8,785.677
Voltage	1,416.285	367.372	1,309.46	1,458.418	1,601.21
Filtered R	175.678	41.343	152.008	175	200.991
ActVal	465.528	84.952	449	469	497.771

Figura 61. Tabla de métricas de las curvas.

4.3.2 Gráficos de líneas

Es necesario el uso de **gráficos de líneas** para estudiar la tendencia de ciertos campos a lo largo del tiempo. Se configuran de manera muy sencilla, estableciendo como eje x el campo *@timestamp* y como eje y el campo que se pretende representar. De esta manera se podrá obtener el valor que se haya obtenido de dicho campo cada vez que un punto ha sido soldado. La precisión en la representación dependerá del rango de tiempo escogido ya que, el eje x, se dividirá en tramos de cada vez más tiempo cuanto mayor sea el rango de tiempo aplicado y, por tanto, el tiempo

[31] Lenguaje de marcado ligero y sencillo que permite dar formato a textos sin utilizar caracteres especiales.

establecido en cada punto no será el tiempo concreto sino un redondeo definido por el rango de tiempo y su división en tramos.

Figura 62. Gráfico de líneas en un rango de 24 horas.

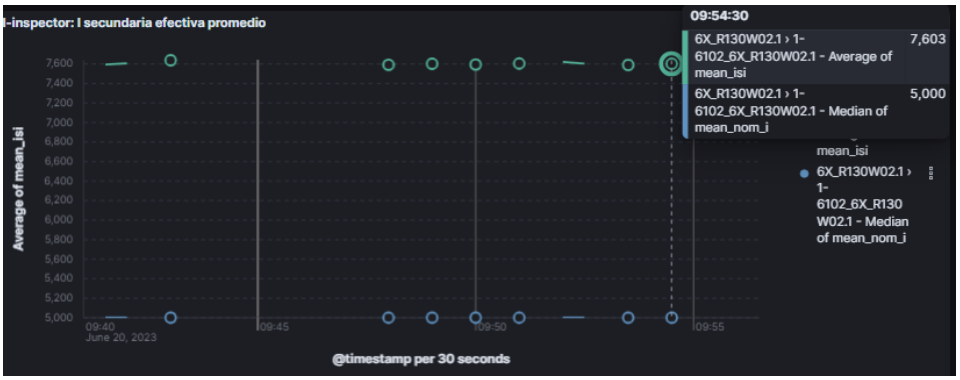


Figura 63. Gráfico de líneas en un rango de 15 minutos.

4.3.3 Tablas

Resulta fundamental, por otra parte, el uso de **tablas** como elemento de visualización en las vistas del *dashboard*. Estas nos permiten obtener información sobre el estado de uno o un conjunto reducido de parámetros en todos los ordenadores de soldadura que se encuentran dentro del proceso de ETL, es decir, para obtener una foto general del estado de los ordenadores analizados sobre un aspecto concreto. Se configuran estableciendo, en primer lugar, la definición de las **filas**, y para esto se tienen las siguientes opciones:

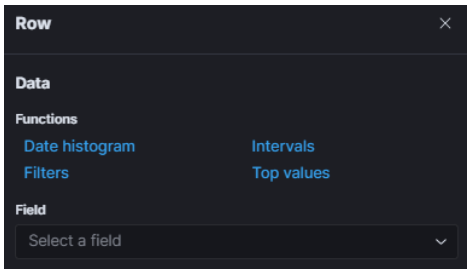


Figura 64. Opciones para selector de filas en tabla.

En primer lugar, se tiene la opción de **histograma,** solo permite el campo *@timestamp* y organiza la tabla a modo de histórico donde las filas se dividen por rangos de tiempo más o menos espaciados en función del rango total seleccionado.

La opción **intervalos** permite desglosar un campo numérico en intervalos numéricos, haciendo que cada fila represente un valor de este campo y que cada fila esté separada por el intervalo definido.

La opción ***Top values*** permite desglosar un campo cualquiera en un número específico de valores distintos. Además, permite el añadido de filtrar incluyendo o excluyendo ciertos valores específicos, determinando así si estos definirán una fila de la tabla o no.

Por último, la opción ***Filters*** permite establecer un filtro mediante una petición KQL (*Kibana Query Language*) [31]. Este es un lenguaje muy sencillo definido por la propia herramienta Kibana que establece condiciones para filtrar las filas que se quiere obtener de manera más compleja o avanzada.

Combinando varias opciones se pueden obtener tablas de visualización muy útiles que aporten información a nivel visual muy potente.

Finalmente quedaría definir las **métricas**. Se puede realizar de dos formas a nivel general:

- Obteniendo una **función** concreta de un campo seleccionado de manera directa.

Figura 65. Posibles funciones sobre campo seleccionado en tabla.

- Describiendo una **fórmula** a mano cuando se requieren cálculos más complejos o con varios campos.

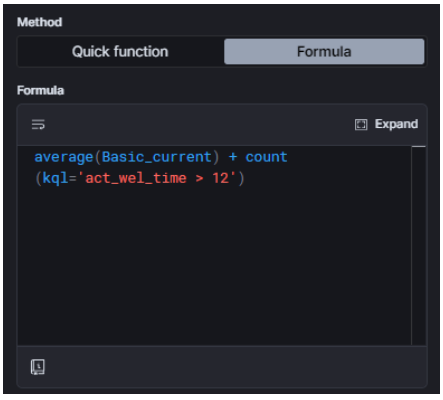


Figura 66. Ejemplo de fórmula para tabla.

Estas métricas se pueden filtrar mediante peticiones KQL, de la misma forma que se ha visto con las filas, para filtrar su visualización.

De manera opcional, es posible dividir las **columnas** de la misma forma que se ha realizado con las **filas** para añadir dimensionalidad a la tabla. Asimismo, permite también la asignación de

colores sobre celdas que albergan valores numéricos en función de este valor o de su porcentaje sobre el total. De esta manera se obtiene más información a nivel visual.

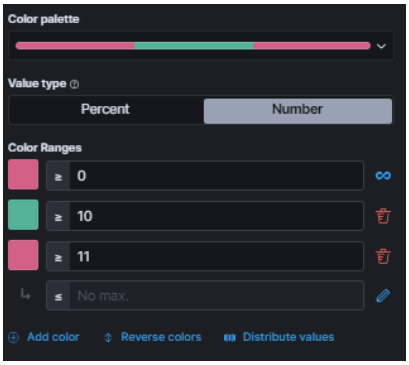


Figura 67. Ejemplo de asignación de colores sobre celdas en tablas.

4.3.4 Resto de visualizaciones

Este apartado comprende visualizaciones como gráficos de donut, mapas de árbol o mapas de calor. Estas proporcionan valor a nivel visual ya que mediante el uso de colores y formas permiten obtener una idea rápida sobre el estado del apartado estudiado en concreto. Se reúnen todas ellas en el mismo punto debido a que la configuración de todas ellas es exactamente igual a la de una tabla y, por tanto, su interés es mayor a nivel de resultados ofrecidos más que en su confección, que, como ya se ha visto, es bien simple.

4.4 Implementación en planta

Si se observa la Tabla 1. Listado de Weld PC se obtiene el conjunto de ordenadores que se encuentran físicamente conectados a la red MPN mencionada en el esquema de conexión de la Figura 11. Esquema general del proyecto. Por tanto, el último paso que se pretende llevar a cabo, una vez se ha desarrollado por completo el sistema, es una implementación en planta a escala reducida a modo de prueba de concepto (PoC) [32], mediante la cual, se obtendrán los resultados que indicarán el camino a seguir.

Capítulo 5. Resultados

El resultado, en general, es lo propuesto más atrás en objetivos: un sistema de control y monitorización de la calidad de soldadura y mantenimiento de los equipos centralizado y optimizado. Este, como se ha visto, se basa en un *dashboard* de visualización con diferentes vistas, desglosadas a continuación.

5.1 Organización del *dashboard*

Se pretende obtener un sistema de monitorización que permita llevar un riguroso control de ciertos parámetros básicos, cuyo seguimiento es crítico para la soldadura de la carrocería, estos son: valor de las **tolerancias** en la corriente en los puntos de ***autocheck***, correcta asignación del **modo de regulación** en los puntos de ***autocheck*** (ver más atrás), porcentaje de **salpicaduras** y **conteo de piezas** hasta el cambio de electrodos (también llamado *Tip Life*[32]).

Adicionalmente, se tienen varias vistas que proporcionan información más detallada del proceso de cada punto donde se muestran datos que permiten realizar un análisis más exhaustivo del estado de la maquinaria y la calidad del proceso y no un control de mínimos, enfocado a que el operario o responsable de soldadura obtenga información detallada de un apartado específico.

5.2 Control de aspectos críticos

Para controlar los aspectos críticos del proceso de soldadura el especialista de soldadura debe acudir a dos vistas del *dashboard*, consideradas las **vistas principales**. Estas son las vistas **auditoría** y **alarmas**.

5.2.1 Auditoría

Desde esta vista el usuario (especialista de soldadura) tiene a disposición una visualización que le aporta información y control sobre cada uno de los aspectos críticos mencionados anteriormente y puede encontrar errores de **configuración** y parámetros de **calidad** gracias al uso de **filtros**.

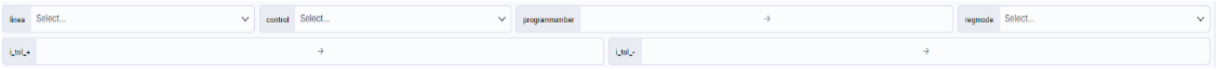


Figura 68. Menú de filtros en la vista auditoría.

Gracias a este menú de filtrado es posible ver qué configuración en los puntos de *Autocheck* se encuentra fuera del estándar[33]. Rápidamente se puede seleccionar una combinación de parámetros mediante los filtros que proporcione esos errores de configuración críticos que se buscan. A continuación, se verá mediante dos ejemplos.

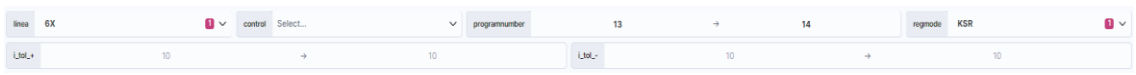


Figura 69. Obtención de modo de regulación de *autocheck* fuera del estándar.

[32] Término que se traduce como vida útil de los electrodos.

[33] El estándar es el protocolo que rige la metodología de trabajo en los distintos procesos industriales llevados a cabo en la fábrica. En este caso siempre que se mencione este término, se hará referencia al estándar de soldadura con la instrumentación del proveedor Harms & Wende.

Tras el lanzamiento[34] y acorde con el estándar, todos los números de programa de *autocheck* deben encontrarse en el modo de regulación SKT (recordar lo visto en el apartado de conceptos previos), por tanto, bajo esta combinación de filtros obtendremos donde NO se está cumpliendo el estándar de soldadura.

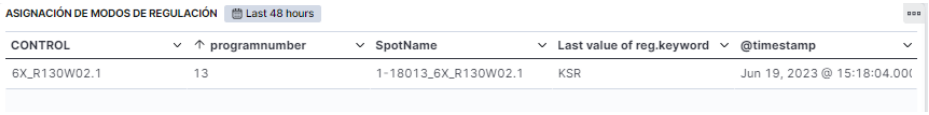


Figura 70. Programas de *Autocheck* fuera del estándar.

Para esta visualización se ha optado por una tabla donde se especifique: el control de soldadura, el número de programa, el nombre del punto, modo de regulación y el sello temporal del último punto de *autocheck* realizado bajo los parámetros seleccionados en el filtro, es decir, cuyo modo de regulación se encuentra fuera del estándar.

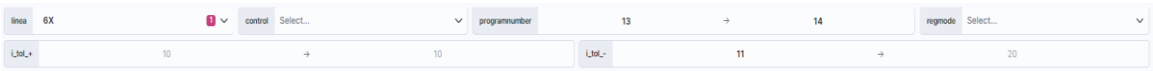


Figura 71. Obtención de tolerancias de *autocheck* fuera del estándar.

Para este segundo ejemplo la dinámica es igual que en el anterior, no obstante, ahora se busca un programa de *autocheck* al que se le haya configurado un valor de tolerancia de corriente que no sea el estipulado en el estándar (el valor estipulado es diez).

Figura 72. Programas de *autocheck* con valores de tolerancia fuera del estándar.

Para esta visualización se ha optado por una tabla con una estructura prácticamente igual que la anterior, pero para el parámetro estudiado en este caso.

Hasta este punto se obtiene control de los errores **de configuración**. El control sobre la calidad se realiza sobre otras dos visualizaciones que se verán a continuación.

[34] El lanzamiento es un término que hace referencia a la puesta a punto para el funcionamiento de maquinaria o de un proceso cualquiera. En este caso se hace referencia a la instrumentación de soldadura de Harms & Wende.

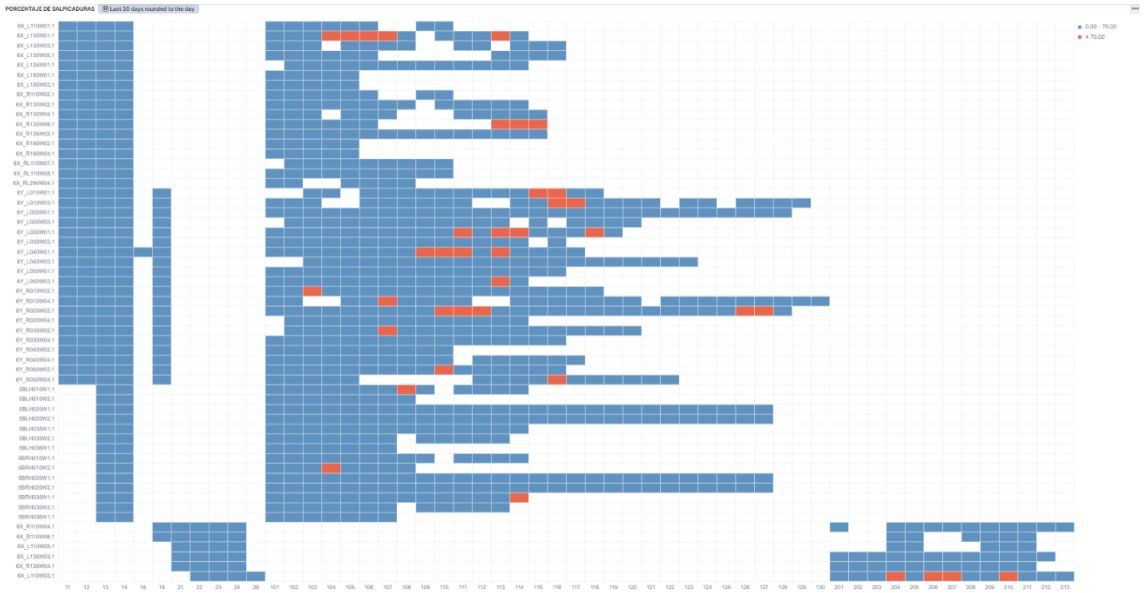


Figura 73. Mapa de calor de salpicaduras.

En primer lugar, se ha optado por un mapa de calor para controlar la tasa de salpicaduras de cada programa en cada uno de los controles de los que se ingieren datos, definiendo así en cada celda de la cuadrícula un tipo de punto distinto. El color de cada celda viene definido por un porcentaje umbral de salpicaduras. Para esta prueba de concepto se ha definido como umbral el setenta por ciento de las salpicaduras, sin embargo, este es ajustable siempre que se requiera. Clicando en una de las celdas se obtiene el control y el número de programa en cuestión, de manera que se puede identificar el programa y el control que está dando problemas de calidad a la distancia de un solo clic.

CONTROL	Ciclos de fresado	Puntos por fresado	puntos totales	Puntos x pieza	Piezas
6Y_L040W01.1	50	88	4,488	17	264
6X_L130W01.1	40	91	3,731	13	287
6Y_L030W01.1	60	80	4,880	16	305
5BLH020W2.1	28	226	6,554	18	364
6Y_L020W01.1	76	94	7,238	19	380
6X_R135W02.1	60	100	6,100	16	381
6Y_L010W01.1	75	84	6,384	16	399
6Y_R030W02.1	70	93	6,603	16	412
6X_L110W01.1	40	90	3,690	8	461
5BRH036W1.1	11	122	1,464	3	488
5BRH020W2.1	39	226	9,040	18	502
5BLH020W1.1	40	226	9,266	18	514
6Y_R010W02.1	95	108	10,368	19	545
6Y_R050W02.1	89	99	8,910	16	556
6X_R130W04.1	90	75	6,825	12	568
6Y_R030W04.1	95	95	9,120	16	570
6Y_L050W01.1	90	101	9,191	16	574
5BRH020W1.1	43	235	10,340	18	574
6Y_R050W04.1	85	87	7,482	13	575
6Y_R020W02.1	80	114	9,234	16	577
6Y_L030W03.1	83	104	8,736	15	582
6Y_R010W04.1	95	110	10,560	18	586
6Y_L010W03.1	100	101	10,201	17	600
6Y_L040W03.1	75	97	7,372	12	614
6X_L135W03.1	80	95	7,695	12	641
5BLH036W1.1	19	98	1,960	3	653
6Y_L020W03.1	120	101	12,221	17	718
6Y_R020W04.1	120	95	11,495	16	718
6X_R130W02.1	42	235	10,105	14	721
6X_L130W03.1	120	80	9,680	13	744
6Y_R040W04.1	110	108	11,988	16	749
6Y_R040W02.1	102	90	9,270	11	842
6Y_L050W03.1	105	116	12,296	14	878
6X_R135W04.1	100	98	9,898	11	899

Por último, se tiene la tabla de control de *Tip life* o vida útil de los electrodos. Esta contiene, como filas, todos los controles conectados al proceso de ETL, y las siguientes columnas: **ciclos de fresado**, cuenta el número de fresados entre cada cambio de electrodos (recordar en qué consiste el fresado de electrodos); **puntos por fresado**, cuenta el número de puntos soldados entre cada fresado; **puntos totales** entre cambios de electrodos, obtenidos mediante:

$$(\boldsymbol{Ciclos_{fresado}} + \mathbf{1}) \times \boldsymbol{Puntos_{fresado}} \quad \textbf{(5.1)}$$

Y, por último, se tiene el número de **piezas totales** producidas entre cambios de electrodos, obtenido mediante:

$$\boldsymbol{Puntos_{pieza}} \times \boldsymbol{Puntos_{totales}} \quad \textbf{(5.2)}$$

Siendo el número de **puntos por pieza** un parámetro configurado.

El objetivo marcado durante esta prueba de concepto es de ochocientas piezas entre cambios de fresado, siendo este umbral configurable, de la misma manera que con la visualización de salpicaduras.

5.2.2 Alarmas

En esta vista se reflejan de manera directa discrepancias de **configuración** con el estándar de soldadura. No existe un menú de filtrado y consta de dos tablas: la primera muestra las tolerancias de los puntos de *autocheck* cuyo valor no es el estipulado por el estándar y la segunda muestra los puntos de *autocheck* que no se encuentran en el modo de regulación correcto. La estructura de ambas es exactamente igual a las explicadas en la vista de auditoría y sus filas únicamente muestran los puntos considerados "erróneos". La diferencia fundamental de esta vista con respecto a la anterior es que esta no dispone de un menú de filtrado ya que su cometido es mostrar únicamente las discrepancias, por lo que el filtrado esta preestablecido en la tabla.

Esta vista trabaja de manera conjunta con un sistema de notificaciones a través del servicio de mensajería y video conferencia utilizado por la empresa: Cisco Webex, por lo que, al especialista de soldadura pertinente le llegarán notificaciones diarias de manera automática en caso de que existan errores de configuración en los ordenadores de soldadura donde esté funcionando el software XPegasus.

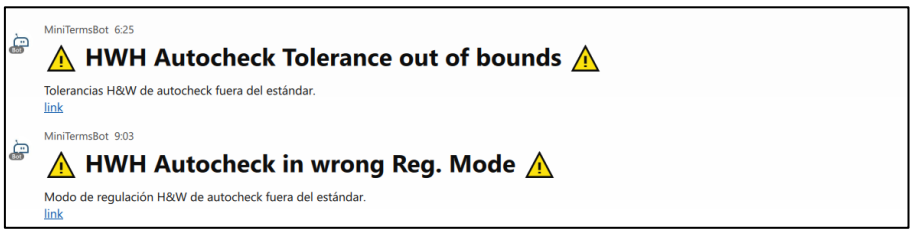


Figura 74. Notificaciones de error en la configuración.

Al clicar en el enlace, el trabajador de soldadura accederá directamente a la vista donde se muestran los errores que han hecho saltar la alarma. De esta manera se evita tener que hacer una inspección activa de este tipo de errores, sino que es el sistema el que, automáticamente, detecta estos errores en todas las líneas integradas en el sistema.

TOLERANCIAS AUTOCHECK Last 48 hours

control	programnumber	tol_+	tol_-	Last value of @timestamp
6Y_L010W01.1	14	100	10	Jun 20, 2023 @ 09:58:22.0(
6Y_L010W03.1	14	100	10	Jun 20, 2023 @ 10:35:22.0(
6Y_R010W02.1	14	100	10	Jun 20, 2023 @ 09:54:21.0(
6Y_R010W04.1	14	100	10	Jun 20, 2023 @ 09:55:57.0(
5BLH020W1.1	13	25	10	Jun 21, 2023 @ 09:24:06.0(
5BLH020W1.1	14	15	10	Jun 20, 2023 @ 09:40:21.0(
5BLH036W1.1	14	20	10	Jun 20, 2023 @ 09:25:09.0(
5BRH035W1.1	13	12	10	Jun 20, 2023 @ 18:32:47.0(
6X_L110W03.1	24	10	20	Jun 21, 2023 @ 10:04:19.0(
6X_L135W03.1	24	10	15	Jun 21, 2023 @ 10:04:36.0(
6X_R110W02.1	13	10	15	Jun 21, 2023 @ 08:56:06.0(
6X_R110W02.1	14	10	20	Jun 21, 2023 @ 09:56:33.0(
6X_R135W04.1	24	10	15	Jun 21, 2023 @ 10:04:36.0(
5BLH010W1.1	13	5	5	Jun 19, 2023 @ 18:09:46.0(

Figura 75. Tabla de discrepancias de tolerancias en *autocheck*.

ASIGNACIÓN DE MODOS DE REGULACIÓN Last 48 hours

CONTROL	programnumber	ModoReg	@timestamp
5BRH020W1.1	14	KSR	Jun 21, 2023 @ 03:11:16.000
6Y_R030W02.1	13	KSR	Jun 19, 2023 @ 20:41:45.000
6X_R130W02.1	13	KSR	Jun 19, 2023 @ 15:18:04.000

Figura 76. Tabla de discrepancias de modo de regulación en *autocheck*.

5.3 Análisis detallado

5.3.1 Análisis de salpicaduras

En esta vista el trabajador de soldadura obtiene una visión más amplia dentro del aspecto de las salpicaduras en la soldadura que le permitirá indagar más allá de qué valores están por encima o por debajo del umbral preestablecido previamente.

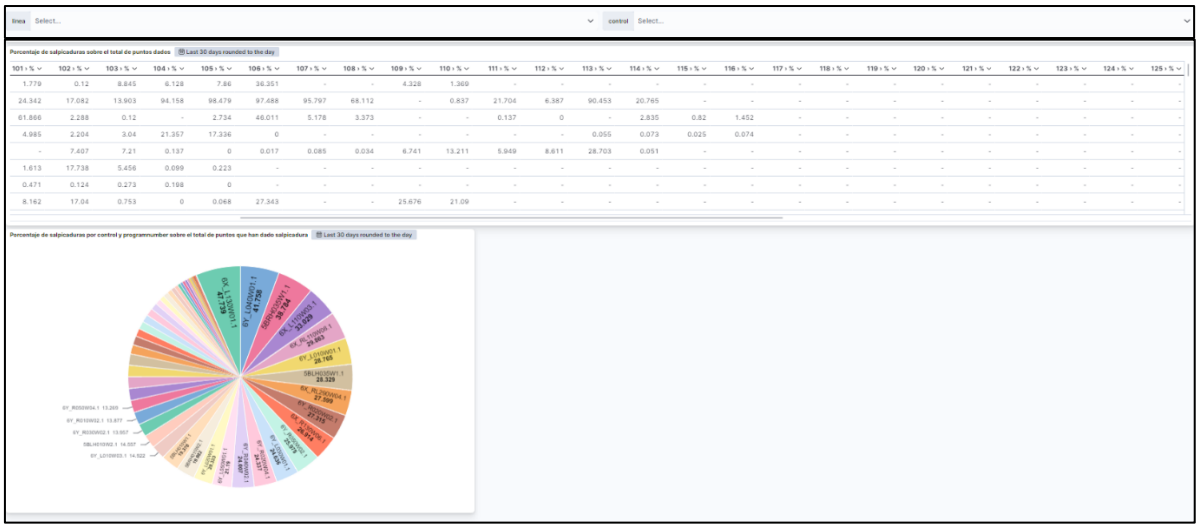


Figura 77. Vista análisis de salpicaduras.

En esta vista se muestra, por un lado, un gráfico de donut donde se desglosan todos los controles registrados en el sistema y su porcentaje de salpicaduras, y, por otro lado, una tabla, donde se define el porcentaje de salpicadura. También dispone de un menú de filtrado para seleccionar entre las distintas líneas registradas.

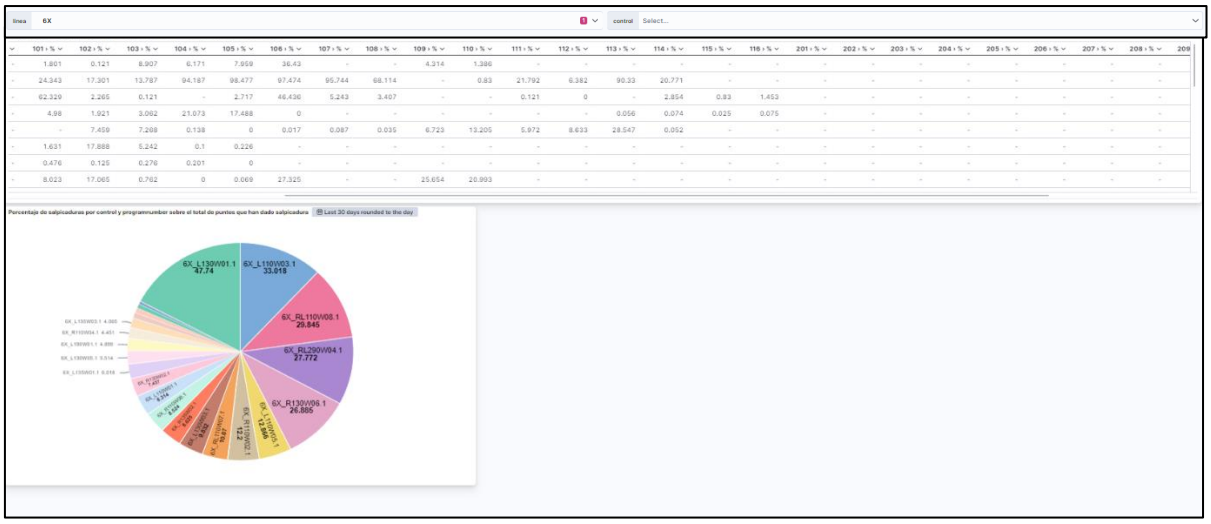


Figura 78. Vista de análisis de salpicaduras filtrada por línea.

Si se selecciona una línea en el filtrado, el número de controles se reduce al de dicha línea. El gráfico de donut permite que al clicar sobre una de las secciones (cada sección corresponde a un control) se filtre por ese control.

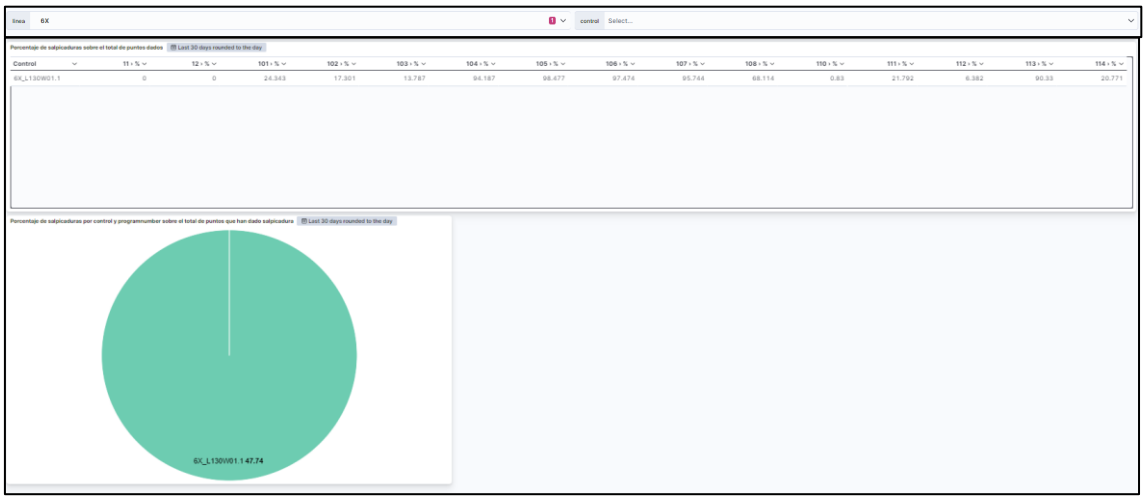


Figura 79. Vista análisis de salpicaduras filtrada por control.

Una vez filtrada por control, es sencillo ver en la tabla los programas que mayor porcentaje de salpicadura tienen.

5.3.2 Vista de análisis de modo regulación

Mediante un panel donde se muestra cada control y la cantidad de programas que ejecuta para soldar puntos, el operario puede seleccionar uno de estos obteniendo así una tabla con características de la configuración de cada programa.

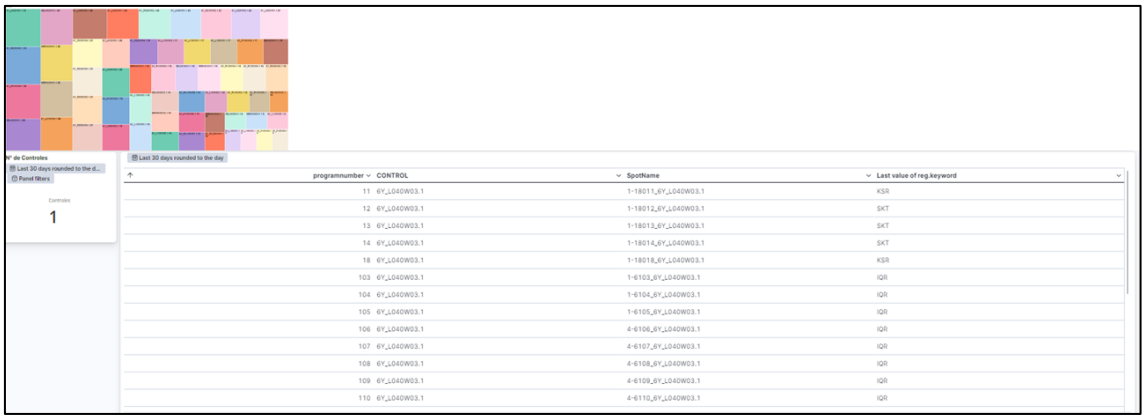


Figura 80. Vista de análisis de modo de regulación.

5.3.3 Vista de análisis de puntos autocheck

En esta vista el especialista en soldadura puede analizar los parámetros fundamentales (voltaje, resistencia y corriente) registrados de los últimos puntos de *autocheck* realizado, además de sus tolerancias, para inspeccionar más allá la posible degradación en la calidad de la soldadura.

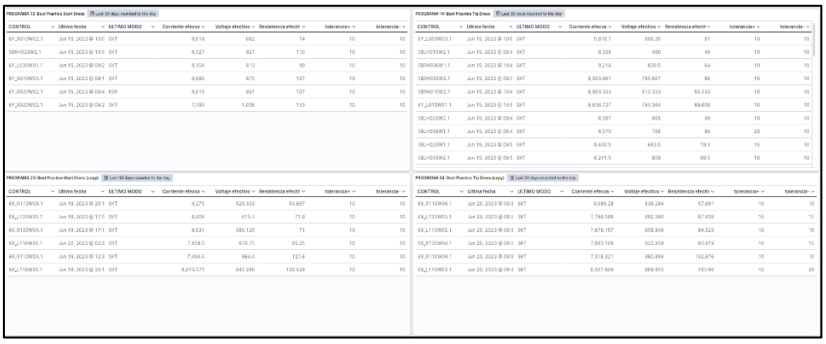


Figura 81. Vista de análisis de puntos *autocheck.*

5.3.4 Vista de análisis del "inspector"

Como se ha visto previamente, el *inspector* es un sistema de monitoreo aplicable a **corriente**, **tensión** y **resistencia**. Por lo tanto, en esta vista se obtiene un histórico de la media de estos parámetros a lo largo de distintos puntos, con el objetivo de estudiar su tendencia y detectar anomalías.

Figura 82. Vista de análisis de "*inspector*".

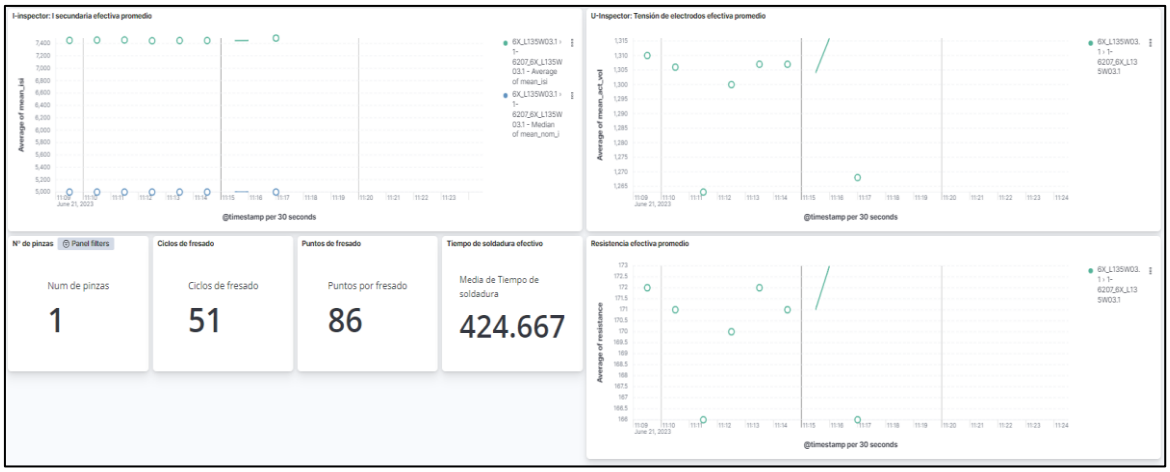


Figura 83. Vista de análisis de "*inspector*" filtrado por punto.

5.3.5 Vista de análisis de curvas de soldadura

Por último, se tiene la vista que aporta la representación gráfica de las curvas de los parámetros básicos del proceso de soldadura por puntos. Cada gráfica como la que se ve en la figura muestra las curvas obtenidas por los sistemas de monitorización en cada uno de los puntos. El eje x de la gráfica representa el tiempo (en milisegundos) de duración del punto. Mediante el menú de filtrado disponible se puede seleccionar el punto específico a analizar o el último punto del control seleccionado.

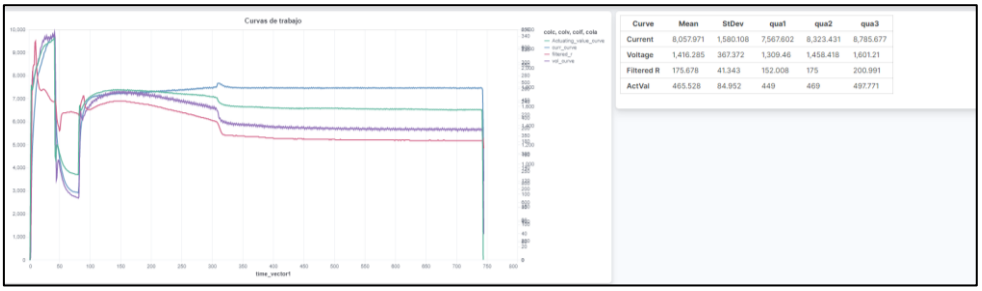


Figura 84. Análisis de curvas de soldadura.

5.4 Control del diario de registro

De manera independiente a los apartados de control explicados anteriormente, se tiene una vista que centraliza los diarios de registro de todos los ordenadores de soldadura conectados al proceso de ETL en una sola vista. Representa los datos obtenidos de la ingesta de la tabla *tbllogbook* y permite visualizar cuándo y quién realizó un cambio en la configuración de los controles, así como la activación de alarmas o mensajes de emergencia.

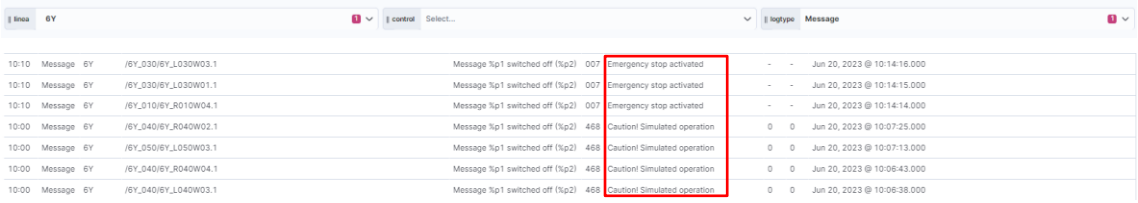


Figura 85. Diario de registro filtrado en la línea 6Y.

Capítulo 6. Conclusiones y propuesta de trabajo futuro

Se ha diseñado un sistema de monitoreo del proceso de soldadura por puntos en las líneas con maquinara de Harms & Wende en las plantas Body 1 y Body 2 de carrocerías de Ford España S.L. renovado y completamente distinto al anterior que centraliza los datos de monitorización y optimiza el control de la calidad y el mantenimiento. Para ello se han requerido conocimientos obtenidos durante el **grado** en bases de datos y redes y se han adquirido conocimientos **nuevos** como el uso de la pila de trabajo ELK (codificación de archivos de configuración en Logstash, peticiones API REST en Elastic y diseño de *dashboards* en Kibana), mantenimiento de Google Cloud y programación en Ruby.

Actualmente se encuentran implementados y en funcionamiento un grupo reducido de ordenadores dentro de este nuevo sistema, por lo que, la primera propuesta de trabajo futuro sería implementar el sistema en el resto de las líneas, procedimiento que solo requiere de la habilitación, mediante la tarjeta de red correspondiente, de los equipos a la red MPN de la empresa.

Asimismo, gracias a la centralización de datos se crea una base de trabajo para, en proyectos futuros, implementar las siguientes herramientas y funcionalidades:

- Procesamiento de los datos mediante **Machine Learning** para ser capaces de predecir averías gracias al estudio de tendencias en los parámetros centralizados.
- Integración de nuevas alarmas en el sistema de notificaciones de Cisco Webex ya desarrollado.
- Trazabilidad de cada coche gracias a la asociación de su código identificativo con los datos de soldadura (proyecto en proceso dentro de la compañía).

Bibliografía

[1] TWI, «WHAT IS SPOT WELDING? (A COMPLETE WELDING PROCESS GUIDE),» [En línea]. Available: https://www.twi-global.com/technical-knowledge/faqs/what-is-spot-welding. [Último acceso: mayo 2023].

[2] M. P. Esteso, «Geek Theory,» 1 Marzo 2021. [En línea]. Available: https://geekytheory.com/que-es-una-etl-y-como-funciona/. [Último acceso: mayo 2023].

[3] «AWS,» [En línea]. Available: https://aws.amazon.com/what-is/elasticsearch/. [Último acceso: mayo 2023].

[4] R. H. Gómez, «BASES DE DATOS NOSQL: ARQUITECTURA Y EJEMPLOS DE APLICACIÓN,» Leganés, 2014.

[5] Elastic, «Logstash reference,» [En línea]. Available: https://www.elastic.co/logstash/. [Último acceso: mayo 2023].

[6] Elastic, «Logstash reference: getting started with Logstash,» [En línea]. Available: https://www.elastic.co/guide/en/logstash/current/getting-started-with-logstash.html. [Último acceso: mayo 2023].

[7] Elastic, «Kibana reference,» [En línea]. Available: https://www.elastic.co/es/kibana/. [Último acceso: mayo 2023].

[8] «Vega doc,» [En línea]. Available: https://vega.github.io/. [Último acceso: mayo 2023].

[9] Elastic, «Kibana reference: vega,» [En línea]. Available: https://www.elastic.co/guide/en/kibana/current/vega.html. [Último acceso: mayo 2023].

[10] Google, «¿Qué es Cloud Storage?,» [En línea]. Available: https://cloud.google.com/learn/what-is-cloud-storage?hl=es. [Último acceso: mayo 2023].

[11] «Harms&Wende,» [En línea]. Available: https://harms-wende.de/en/products/xpegasus-bediensoftware/. [Último acceso: mayo 2023].

[12] Arsys, «¿Qué es PostgreSQL y por que llevarlo a Cloud?,» [En línea]. Available: https://www.arsys.es/blog/postgresql-servidores. [Último acceso: mayo 2023].

[13] «Ruby,» [En línea]. Available: https://www.ruby-lang.org/es/. [Último acceso: mayo 2023].

[14] «HeidiSQL,» [En línea]. Available: https://www.heidisql.com/. [Último acceso: mayo 2023].

[15] A. Lois, «Zona System,» 24 Febrero 2019. [En línea]. Available: https://www.zonasystem.com/2019/02/netsh-portproxy-port-forwarding-local-windows.html. [Último acceso: mayo 2023].

[16] J. P. Joule, On the Production of Heat by Voltaic Electricity, 1841.

[17] Z. Hongyan y J. Senkara, Resistance Welding: Fundamentals and Applications, 2011.

[18] FORD ESPAÑA S.L. y HARMS & WENDE, «ESTÁNDAR DE SOLDADURA EQUIPOS HARMS & WENDE,» mayo, 2023.

[19] J. J. M. Durango, J. J. Ordoñez y L. F. M. Machado, «Diseño y construcción de un convertidor dc/dc tipo Boost con PWM ajustable,» *Scientia et Technica*, vol. 22, nº 1, XXII.

[20] «Doctor Welding,» 5 junio 2020. [En línea]. Available: https://doctorwelding.com/las-salpicaduras-en-soldadura-por-que-se-presentan/. [Último acceso: mayo 2023].

[21] Elastic, «Logstash reference: jdbc input plugin,» [En línea]. Available: https://www.elastic.co/guide/en/logstash/8.8/plugins-inputs-jdbc.html. [Último acceso: mayo 2023].

[22] Varios, «Github,» [En línea]. Available: https://github.com/jmettraux/rufus-scheduler. [Último acceso: mayo 2023].

[23] Elastic, «Logstash reference: mutate filter plugin,» [En línea]. Available: https://www.elastic.co/guide/en/logstash/current/plugins-filters-mutate.html. [Último acceso: mayo 2023].

[24] Elastic, «Logstash reference: date filter plugin,» [En línea]. Available: https://www.elastic.co/guide/en/logstash/current/plugins-filters-date.html. [Último acceso: mayo 2023].

[25] Elastic, «Logstash reference: ruby filter plugin,» [En línea]. Available: https://www.elastic.co/guide/en/logstash/current/plugins-filters-ruby.html. [Último acceso: mayo 2023].

[26] Elastic, «Logstash reference: Jdbc_static filter plugin,» [En línea]. Available: https://www.elastic.co/guide/en/logstash/current/plugins-filters-jdbc_static.html. [Último acceso: mayo 2023].

[27] Elastic, «Logstash reference: grok filter plugin,» [En línea]. Available: https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html. [Último acceso: mayo 2023].

[28] Elastic, «Logstash reference: clone filter plugin.,» [En línea]. Available: https://www.elastic.co/guide/en/logstash/current/plugins-filters-clone.html. [Último acceso: mayo 2023].

[29] Elastic, «Logstash reference: prune filter plugin.,» [En línea]. Available: https://www.elastic.co/guide/en/logstash/current/plugins-filters-prune.html. [Último acceso: mayo 2023].

[30] Elastic, «Logstash reference: Elasticsearch output plugin,» [En línea]. Available: https://www.elastic.co/guide/en/logstash/current/plugins-outputs-elasticsearch.html. [Último acceso: mayo 2023].

[31] Elastic, «Kibana reference: Kibana Query Language,» [En línea]. Available: https://www.elastic.co/guide/en/kibana/current/kuery-query.html. [Último acceso: mayo 2023].

[32] T. Asana, «Asana,» 5 enero 2022. [En línea]. Available: https://asana.com/es/resources/proof-of-concept. [Último acceso: junio 2023].