ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE APLICACIONES DE SUPERVISIÓN INDUSTRIAL DE CÓDIGO ABIERTO PARA EL LABORATORIO DE REDES INDUSTRIALES

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE HISTORIADOR DE PROCESO INDUSTRIAL DE CÓDIGO ABIERTO

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN INGENIERÍA ELECTRÓNICA Y AUTOMATIZACIÓN

CHRISTOPHER ALEXIS CASTRO GARCÉS

christopher.castro@epn.edu.ec

DIRECTOR: DRA.- ING. SILVANA DEL PILAR GAMBOA BENÍTEZ

silvana.gamboa@epn.edu.ec

DMQ, abril 2023

CERTIFICACIONES

Yo Christopher Alexis Castro Garcés, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Firmado electrónicamente por:
CHRISTOPHER ALEXIS CASTRO GARCES

CHRISTOPHER ALEXIS CASTRO GARCÉS

Certifico que el presente trabajo de integración curricular fue desarrollado por Christopher Alexis Castro Garcés, bajo mi supervisión.

Firmado electrónicamente por:
SILVANA DEL PILAR GAMBOA BENITEZ

DRA.- ING. SILVANA DEL PILAR GAMBOA BENÍTEZ
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Christopher Alexis Castro Garcés

DRA.- ING. SILVANA DEL PILAR GAMBOA BENÍTEZ

DEDICATORIA

Dedico este trabajo a mis padres y a mis tutores que han tenido la ferocidad de poder guiarme de manera adecuada y haberme permitido poder llegar hasta este punto de mi vida.

AGRADECIMIENTO

Agradezco a mis padres por haberme ayudado y apoyado con lo que ha estado dentro de su alcance para poder conseguir todos y cada uno de mis logros.

ÍNDICE DE CONTENIDO

Contenido

RESUMEN

La necesidad de recopilar datos resultantes de la operación de un determinado proceso, así como la necesidad de poder realizar registros de los mismos, para la generación de históricos que ayuden a la optimización del proceso en cuestión, obliga a que el sector industrial se surta de un software Historiador de Datos; de manera práctica también conocido como Historiador de Procesos. Actualmente hay varias herramientas que buscan satisfacer dichas necesidades, sin embargo, el costo de la adquisición de su licencia puede alcanzar valores elevados dependiendo de sus prestaciones.

En el presente trabajo se diseña e implementa un software de aplicación, mismo que hace las veces de Historiador de procesos, utilizando, principalmente, el lenguaje de software libre JavaScript y el motor de base de datos PostgreSQL. Se incorpora un servidor de datos que permite un fácil acoplamiento entre el aplicativo y los diferentes dispositivos clientes utilizando consultas de tipo SQL, así como los protocolos MQTT y Modbus TCP. Se cuenta con una interfaz gráfica en donde se centraliza toda la información asociada a la configuración, monitoreo y mantenimiento del flujo de datos.

Todo el software desarrollado, así como las dependencias del mismo y el motor de base de datos se encuentran contenidos en un entorno virtual de ejecución.

De esta forma, se busca beneficiar a las PYMES al proporcionarles una alternativa de código abierto y de menor costo a los aplicativos historiadores comerciales.

La validación del funcionamiento del aplicativo se desarrolla en base a las pruebas de funcionamiento y despliegue realizadas.

PALABRAS CLAVE: Historiador, Procesos, PostgreSQL, JavaScript, Node, Docker, SQL, MQTT, Modbus TCP

ABSTRACT

The need to collect data resulting from the operation of a certain process, as well as the need to be able to record them, for the generation of historical data that help optimize the process in question, forces the industrial sector to supply itself with a software known as Process Historian. Currently there are several tools that seek to satisfy these needs, however, the cost of acquiring the license of said tool can reach high values.

In this work, an application software is designed and implemented, which acts as a Process Historian, mainly using the free software language JavaScript and the PostgreSQL database engine. A data server is also incorporated that allows an easy coupling between the application and the different client devices using SQL-type queries as well as the MQTT and Modbus TCP protocols. Additionally, a graphical interface was developed where all the information associated with the configuration, monitoring and maintenance of the data flow is centralized.

All the software developed, as well as its dependencies and the database engine, are contained in a virtual environment created with Docker and Docker-compose.

In this way, it seeks to benefit SMEs by providing them with an open source and lower cost alternative to commercial historian applications.

The validation of the application's operation is developed based on the operation and deployment tests carried out.

KEYWORDS: Historian, Processes, PostgreSQL, JavaScript, API, Docker, Docker-compose, MQTT, Modbus TCP.

1 INTRODUCCIÓN

La necesidad de recopilar datos resultantes de la operación de un determinado proceso, así como la necesidad de poder respaldar los mismos para la generación de históricos que ayuden al análisis y optimización del proceso en cuestión, obliga a que el sector industrial requiera de aplicativos que se encarguen de la adquisición y almacenamiento de datos de los diferentes dispositivos del sistema de automatización y control industrial (IACS). Dichas aplicaciones toman el nombre de Historiador de procesos [1].

Un historiador de procesos es un software que se encarga de registrar datos de una planta y guardarlos en una base de datos utilizando marcas de tiempo con el fin de generar información sobre patrones o tendencias en el comportamiento de variables de procesos industriales [1]. Es frecuente encontrarse a un historiador de procesos como una herramienta complementaria, es decir como un servicio adicional que puede ser acoplado a aplicaciones de monitoreo y control de un entorno de producción pero que también puede operar como un módulo completamente autónomo [2]. De esta forma, el modelo de negocio de muchos de los desarrolladores de historiadores de procesos se enfoca principalmente en ofrecer un servicio adaptable y escalable, en donde el costo del mismo es proporcional a la cantidad de datos que puede mantener en su registro.

En la actualidad existen varios desarrolladores de software que buscan satisfacer esta necesidad y que incorporan herramientas que facilitan el trabajo del operador, sin embargo, la mayoría de ellos requieren de una licencia que usualmente es de un costo relativamente elevado. Por esta razón es muy común que el sector industrial, específicamente aquellos que no pueden realizar fuertes inversiones como es el caso de la micro, pequeña y mediana industria (MIPYMES [3]), opten por no utilizar un sistema de registro o, en su defecto, utilizar software comercial sin su licencia, lo que limita las capacidades del software en cuestión, esto siempre y cuando el desarrollador lo permita. Por otro lado, existen aplicaciones con licencia de libre acceso para respaldo de datos, pero no están enfocadas para funcionar en un ambiente industrial o, por el contrario, tienen un enfoque industrial pero su código, así como sus funciones no son accesibles en su totalidad lo que dificulta su adaptación a la realidad de los diferentes procesos industriales locales. En este sentido se plantea el desarrollo local de una aplicación historiador de proceso, que traerá ventajas tales como que el soporte se podrá realizar a nivel nacional, además que la disponibilidad del código base permitirá futuras mejoras acorde con las necesidades del entorno industrial ecuatoriano.

Por las razones mencionadas, se propone desarrollar, utilizando software libre, una aplicación que funcione como historiador de procesos, misma que constará de un “back-end” [4] encargado de administrar el tráfico de información de entrada o salida, así como de su respectivo respaldo; y de un “front-end” [4] conformado por una interfaz de usuario que permitirá al operador modificar su despliegue de servicios y visualizar los registros de los procesos.

1.1. Objetivo general

El objetivo general de este Proyecto Técnico es:

- Diseñar e implementar un historiador de procesos basado en software de código abierto.

1.2. Objetivos específicos

Los objetivos específicos del Proyecto Técnico son:

1. Realizar una recopilación bibliográfica referente a las características, la implementación y el uso de historiadores en procesos industriales.
2. Determinar los principales requerimientos con los que debe cumplir un historiador de procesos industrial.
3. Seleccionar el software libre en base al cual se trabajará el historiador de procesos.
4. Diseñar e implementar un historiador de procesos de código abierto, con una interfaz de visualización y configuración para el operador.
5. Realizar pruebas de validación de funcionamiento del historiador de procesos implementado.

1.3. Alcance

Se realizará una recopilación bibliográfica de documentos técnicos referentes a historiadores en procesos industriales.

Se establecerán los requerimientos que debe cumplir el historiador de proceso en base a un análisis de las características de aplicaciones comerciales enfatizando sus capacidades de gestión de flujo de datos y las herramientas visuales para el acceso a los datos en cuestión.

Se seleccionará el motor de base de datos a utilizar para perdurar los registros de datos generados por el historiador de procesos.

Se seleccionará el entorno de programación, basado en código abierto, a usarse en la implementación en función de los requerimientos establecidos al estudiar aplicaciones comerciales.

En base a los requerimientos establecidos previamente, se diseñará los componentes que deben integrar el historiador de proceso procurando una arquitectura modular, de tal forma que facilite el desarrollo y, especialmente, posibles modificaciones a futuro.

Se diseñará e implementará una herramienta que permita la inicialización de una base de datos, es decir, la configuración de seguridad y acceso; la definición de sus respectivos atributos y la definición de los vínculos que los interrelacionan.

Se diseñará e implementará una aplicación que permita establecer múltiples conexiones con la base de datos para que múltiples dispositivos puedan conectarse en tiempo real con el objeto de enviar información para su respectivo almacenamiento en la base de datos.

Se diseñará e implementará una aplicación que permitirá la conexión a clientes para realizar consultas de información, tanto histórica como en tiempo real.

Se diseñará e implementará una aplicación que satisfaga los requerimientos que un operador de procesos necesita de una interfaz visual para garantizar un adecuado control y monitoreo de un determinado proceso en base a sus datos históricos.

Una vez desarrollado el software, se realizarán pruebas enfocadas en la verificación de su funcionamiento, así como su facilidad de operación utilizando como base de pruebas la emulación de un proceso industrial que cuente con los dispositivos básicos que se encuentran en los sistemas de automatización y control industrial prácticos.

1.4. Marco teórico

En esta sección se revisan conceptos relacionados a los componentes utilizados para el historiador de procesos.

1.4.1. Software historiadores de procesos comerciales

El software historiador de procesos es una de las partes que forman un sistema de control industrial. De manera que su uso es obligatorio para los casos en los que se desea mantener un registro de históricos del estado del proceso gestionado a través del sistema de control en cuestión.

Para solventar esta necesidad varias entidades se han dedicado al desarrollo de aplicaciones que cumple este cometido. Dichas aplicaciones tienen diferentes tipos de enfoques dependiendo de sus requerimientos, por ejemplo, mientras que un desarrollador busca facilitar la operabilidad de la interfaz visual, otro busca tener una amplia compatibilidad con dispositivos externos.

En la actualidad existen varios desarrolladores que proporcionan un software de pago, que, en su mayoría, el costo está en función de la cantidad de información que se puede manejar proveniente de múltiples dispositivos simultáneos.

A continuación, en la Tabla 1.1, se mencionan los desarrolladores de historiadores de procesos más populares y sus principales características.

Tabla 1.1. Estructura de la entidad "Agente" dentro de la base de datos.

Empresa de desarrollo	Nombre del software	Características destacables	Costo de la licencia
Schneider Electric	Wonderware Historian	Se trata de una aplicación capaz de manejar datos en tiempo real y datos históricos. Opera sobre el servidor Microsoft SQL	El precio está en función de la cantidad de tags que puede manejar el aplicativo. Yendo desde 100 tags a un precio estimado de $2000 hasta los 2000000 de tags con un costo estimado de $10000 [5]
Rockwell Automation	FactoryTalk Historian	Se trata de una aplicación que cuenta con 2 versiones. Machine edition (ME): proporciona acceso a las funcionalidades en un equipo cualquiera; Site Edition (SE): proporciona acceso a las funcionalidades a través de una red de área local, necesita ser desplegado en Windows Server	Cuenta con 3 variantes de licencias que permiten el uso del aplicativo durante un año. Iniciado con un estimado de $35000 para la versión ME. Los estimados de las otras licencias requieren de una consultoría con los proveedores. [6]
Siemens	SIMATIC Process Historian	Cuenta con una serie de mecanismos de seguridad para garantizar que los datos se archiven de forma fiable. Si se produce una avería del sistema, la base de datos se puede recuperar mediante copia de seguridad. Está diseñado para trabajar en sincronía	El aplicativo cuenta con 3 tipos de licencias: Basic, Redundancy y Openness (conexión OPC UA). Estas licencias se pueden comprar como alquiler, tipo individual o flotante dependiendo de la aplicación. La licencia de alquiler permite el uso de Process Historian

		con WinCC RT Professional, WinCC V7, SIMATIC Information Server y SIMATIC PCS7	durante un año completo en una sola computadora y almacenamiento local; la licencia única permite el uso ilimitado en una sola computadora y almacenamiento local; y la licencia flotante permite el uso ilimitado en varios equipos donde se requiere o se prefiere el almacenamiento local y del servidor. El precio parte desde los $3000 en función del tipo de licencia. [7]

Con respecto a la lista de precios disponible de cada software, es importante mencionar que la mayoría de desarrolladores ofrecen un servicio personalizado en función de las necesidades de operación y que cuyo costo varía acorde a las funcionalidades disponibles, como en el caso de SIMATIC Process Historian de Siemens, que cuenta con una variedad de formas para adquirir el mismo aplicativo; por esta razón resulta complicada una estimación exacta del precio final.

De manera adicional es importante considerar que los historiadores comerciales están desarrollados de manera que se pueda integrar con otras aplicaciones del mismo fabricante. Por lo que si en una industria se opta por trabajar con software de la empresa Schneider Electric deberá considerar adquirir el aplicativo Wonderware Historian que tiene total integración con toda la suite de aplicaciones, de lo contrario no se garantiza la funcionalidad y operabilidad del mismo.

Es por esta razón que se requiere de un software historiador de procesos de código abierto que pueda integrarse con facilidad a la mayoría de sistema de control industrial.

1.4.2. Aplicación web

Una aplicación web es una herramienta que funciona acorde a una lógica programada que se ejecuta en un ordenador, conocido como servidor, y que puede ser accedido a través de internet o intranet haciendo uso de un navegador web [8].

Este formato de aplicación es el más popular en la actualidad debido a su accesibilidad y operabilidad en comparación con el formato clásico de aplicaciones de escritorio.

El funcionamiento de una aplicación web depende de 3 apartados en principio [8]:

- Una base de datos en donde se registre información asociada al funcionamiento del aplicativo.
- La lógica de funcionamiento o código de operación, que toma el nombre de back-end, puede ser accedida a través del internet o de la intranet.
- Una interfaz gráfica que se ejecuta en un navegador web y que consume las funcionalidades del código de operación para permitir el intercambio de información con el usuario. Este componente toma el nombre de front-end.

1.4.3. Lenguaje de programación JavaScript

Existen varios caminos a tomar cuando se trata de desarrollar código para back-end o para front-end.

Javascript es un lenguaje de programación interpretado creado a partir del lenguaje Java que permite desarrolla aplicaciones con mayor interactividad y dinamismo, es decir que cuentas con herramientas para realizar animaciones de texto e imágenes, botones y pantallas.

JavaScript utiliza archivos de texto denominados scripts. Estos scripts operan conjuntamente con HTML, que es el lenguaje que se utiliza para el desarrollo de páginas web, para ejecutarse y renderizarse desde un navegador web [9] [10]

1.4.4. Node.js

Se trata de un entorno de ejecución para el código desarrollado en JavaScript, el mismo que contienen herramientas y funcionalidades orientadas al asincronismo de eventos en un ordenador, es decir que procura la ejecución tareas en segundo plano permitiendo que el flujo del código avance hasta que dichas tareas hayan finalizado y retornen información.

Al igual que JavaScript, Node utiliza el mismo motor de ejecución JavaScript V8, que se encarga de convertir el código en JavaScript a código de máquina o ensamblador con el objeto de optimizar sus tiempos de ejecución.

Como ya se mencionó Node.js se basa en un modelo de manejo de entrada y salida de datos controlados por eventos. Un evento se refiere a cualquier acción o modificación que puede englobar lectura, escritura y solicitudes HTTP. Todo esto con el objeto de garantizar un funcionamiento asíncrono de un entorno liviano y rápido.

Node.js tiene muchas utilidades, pero la que más destaca para el presente proyecto está asociada a la facilidad de crear aplicaciones que se comunican por la web y que son capaces de manejar conexiones de clientes simultáneos con gran facilidad [11].

Con respecto a las facilidades y ventajas que otorga Node.js se tiene:

- Alto rendimiento, gracias su funcionamiento basado en eventos.
- Cuenta con un gestor de paquetes o librerías remoto denominado NPM (Node Packet Manager) que alberga una gran cantidad de repositorios. Adicionalmente permite la automatización de adquisición e instalación de dependencias.
- Es multiplataforma, puede ejecutarse sin ningún problema en casi todos los sistemas operativos.
- La comunidad de desarrolladores es muy grande por lo que existe gran variedad de proyectos realizados sobre Node.js y por lo tanto resulta sencillo el intercambio de ideas entre desarrolladores.

A continuación, en la Figura 1.1, se muestra un script con la forma en la que se puede inicializar un servidor web basado en http utilizando Node,js que escucha peticiones http en la URL http://127.0.0.1:8000. El servidor responderá con la cadena de texto “Hola mundo” ante cualquier petición. Una vez iniciado el servidor, se imprime por consola la URL para su acceso.

```
1  // Se carga el módulo de HTTP
2  var http = require("http");
3
4  // Creación del servidor HTTP, y se define la escucha
5  // de peticiones en el puerto 8000
6  http.createServer(function(request, response) {
7
8     // Se define la cabecera HTTP, con el estado HTTP (OK: 200) y el tipo de contenido
9     response.writeHead(200, {'Content-Type': 'text/plain'});
10
11    // Se responde, en el cuerpo de la respuesta con el mensaje "Hello World"
12    response.end('Hola Mundo!\n');
13 }).listen(8000);
14
15 // Se escribe la URL para el acceso al servidor
16 console.log('Servidor en la url http://127.0.0.1:8000/');
```

Figura 1.1. Script para la creación de un servidor web basado en http utilizando Node.js

1.4.5. Exress.js

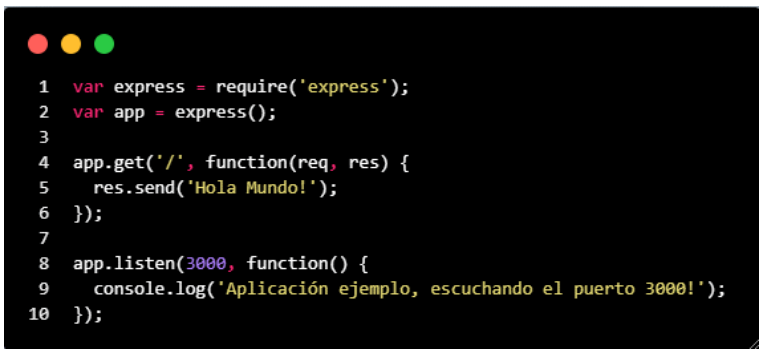
Si bien Node.js permite la creación de servidores basados en http, en la mayoría de ocasiones se requiere agregar soporte a otro tipo de funcionalidades por ejemplo el manejo específico de los diferentes tipos de peticiones HTTP (GET, POST, PUT, etc), la definición y gestión de rutas, el manejo de plantillas y ficheros estáticos dinámicamente, entre otros. Una solución ante este requerimiento es el framework express.js [12].

Un framework es un conjunto de conceptos, buenas prácticas y criterios aplicados a un proyecto, que funcionan como una plantilla para el desarrollo de nuevos proyectos.

Exress.js se encuentra disponible en manejador de paquetes NPM y es uno de los frameworks más populares de Node puesto que proporciona herramientas para:

- Estructurar peticiones personalizadas basadas en http sobre rutas definidas.
- Integrar motores de renderización que facilitan las respuestas en función de plantillas definidas
- Definir la operación del host, sus puertos y la localización en memoria de ficheros y plantillas.
- Facilitar la creación de peticiones intermediarias que permitan la interacción con otras entidades o dispositivos por HTTP.
- Invocar funciones o métodos a partir de una petición http personalizada.

Con express.js fácilmente se puede inicializar un servidor. Como se muestra en la Figura 1.2. Primero se debe invocar al framework, como se visualiza en la línea 1. A continuación, en la línea dos, se crea la aplicación de express bajo el nombre “app” que es la que cuenta con los métodos de enrutamiento de peticiones HTTP, configuraciones del host, renderización de vistas, manejo de plantilla y demás funcionalidades del framework. En las líneas 4, 5 y 6 se visualiza la forma en la que se define una ruta, en este caso la ruta “/” asociada una petición “GET” que, al ser invocada, ejecuta una función que recibe la petición y la respuesta de dicha petición para finalmente retornar una cadena de texto “Hola mundo”. En las líneas restantes se define la configuración del servidor, es este caso se utiliza el puerto 3000 y se imprime un mensaje en la consola una vez que se ha ejecutado la aplicación.



```
1  var express = require('express');
2  var app = express();
3
4  app.get('/', function(req, res) {
5    res.send('Hola Mundo!');
6  });
7
8  app.listen(3000, function() {
9    console.log('Aplicación ejemplo, escuchando el puerto 3000!');
10 });
```

Figura 1.2. Script para la creación de un servidor web basado en http utilizando Express.js

1.4.6. Vue.js

Se trata de un framework para Node.js que facilita la creación de interfaces gráficas de usuario (GUI). Se encuentra disponible en el gestor de paquetes NPM.

Una de las características que diferencia a Vue.js de otros frameworks es su flujo de trabajo basado en componentes, que se trata de un código encapsulado que representa un elemento visual el cual es reutilizable en todo el proyecto [13].

Como se observa en la Figura 1.3, un componente de Vue.js puede estar escrito haciendo uso de 3 lenguajes de programación:

- JavaScript, para describir la lógica de operación y funcionamiento del proyecto
- HTML, para definir la estructura de los componentes gráficos que mostrarán en pantalla
- CSS, para definir el formato de los componentes gráficos

Cada uno de los componentes tiene la capacidad de invocar otros componentes, así como de reutilizar los datos resultantes de la ejecución de su lógica. Esto le otorga al proyecto una capacidad enorme de escalamiento progresivo

a) b)

Figura 1.3. Del lado izquierdo Figura a, el script para la creación de un componente de Vue.js. Del lado derecho, Figura b, la renderización de dicho componente en el navegador web

1.4.7. Base de datos

Una base de datos se define como una entidad encargada de manejar, almacenar y mantener a disposición información en un espacio de memoria físico [14].

En la actualidad existen muchas variantes de bases de datos que pueden ser englobadas en 2 grandes grupos en función de la manera en la que se maneja la información contenida:

- Bases de datos relacionales, su información se organiza sobre tablas. Cada tabla cuenta con columnas que corresponden a un atributo que representan un tipo de información. A medida que se ingresa información a la base de datos, se van agregando filas a las tablas mismas que contienen la información agrupada acorde a los atributos definidos. Cada tabla puede relacionarse con otra tabla haciendo uso de referencias a partir de los atributos de una tabla, esto gracias al modelo de datos que se maneja. De ahí su nombre de “relacional”

- Bases de datos no relacionales, almacenan su información en archivos con un modelo de diccionario que no tienen como tal una forma de agrupamiento definida por defecto. En este caso el usuario debe definir el modelo sobre de almacenamiento en donde no es necesario que exista relación de datos. Esto facilita en gran medida el manejo de gran cantidad de información no estructurada o que no tiene relación entre sí. De ahí el nombre de “No relacional”.

Figura 1.4. Componentes básicos de una base de datos.

Tal como se muestra en la Figura 1.4, los tres aspectos más importantes para una base de datos son:

- El lenguaje, siendo el más utilizado las consultas SQL
- El motor de base de datos, encargado de la gestión de información en el espacio de memoria asignado.
- El servidor, o host que es la entidad que permite el flujo de información tanto de entrada como de salida

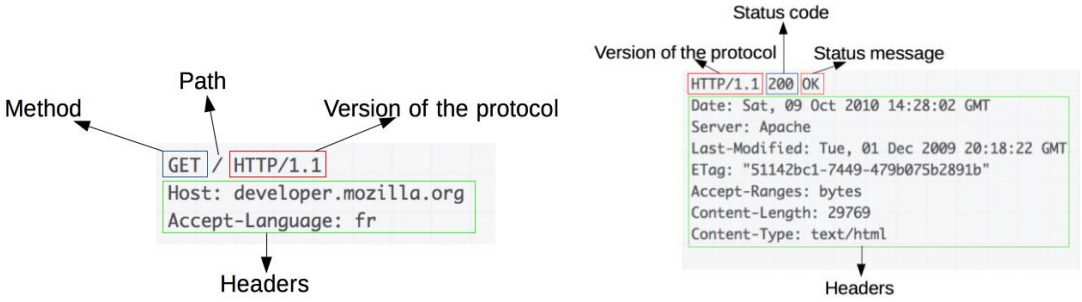
1.4.8. Protocolos de comunicación

1.4.8.1. HTTP

Hypertext Transfer Protocol o HTTP es un protocolo de comunicación enfocado en el intercambio de datos y recursos, utilizando ficheros HTML y CSS, por ejemplo.

El protocolo HTTP es la base de la mayor parte de tráfico de información de la web. Tiene una estructura basada en comunicación cliente-servidor es decir que el cliente es el encargado de inicializar la petición de información, generalmente a través de un navegador web o de un programa.

El intercambio de información se estructura en mensajes individuales. Los mensajes emitidos por un cliente se conocen como “peticiones” conformados por: Un método que corresponde a la operación que se desea realizar (GET, PUT, POST, entre otros), la dirección del recursos pedido o URL, la versión del protocolo, la cabecera y el cuerpo del mensaje ;y, por otro lado, los mensajes emitidos por el servidor se conocen como “respuestas” cuya arquitectura está conformada por: La versión del protocolo que se usa, un código de estado que contiene el resultado de la petición, un mensaje de estado que detalla el código de estado, las cabeceras del mensaje y opcionalmente el recurso o dato solicitado. [15]



a) **b)**

Figura 1.5. Del lado izquierdo, en la Figura a, se muestra la Arquitectura de una petición HTTP. De lado derecho, en la Figura b, se muestra la Arquitectura de una respuesta HTTP

1.4.8.2. MQTT

MQTT o MQ Telemetry Transport, se trata de un protocolo de comunicación de máquina a máquina (M2M), está basado en TCP/IP [15]. Utiliza el puerto 1883 por defecto.

Es un protocolo muy popular en el campo del internet de las cosas IoT debido a sus bajos requerimientos de procesamiento y operación basado en el paradigma de mensajes encolados con patrones pub-sub. Es decir que permite que clientes puedan publicar mensajes y suscribirse a tópicos para recibir los mensajes que han publicado otros dispositivos.

Una de sus principales ventajas es su interoperabilidad ya que se trata de un protocolo basado en TCP/IP es más sencillo permitir el acceso a aplicaciones que utilicen distintos lenguajes de programación dejando de lado problemas de compatibilidad.

Un tópico permite la organización de canales de comunicación de manera que en un servidor MQTT o también llamado broker puede disponer de diferentes tópicos sobre los cuales se intercambian mensajes.

Un mensaje MQTT puede ser dividido en 3 apartados:

- Una cabecera obligatoria que ocupa de 2 a 5 bytes. Un byte corresponde a la identificación del tipo de mensaje, su código de control y la longitud del mensaje.
- Una cabecera opcional que está a disposición para la definición de cierto tipo de mensajes, situaciones o eventos.
- El contenido del mensaje que puede tener un máximo de 256 Mb, sin embargo, en la práctica el máximo valor permitido son 4kB.

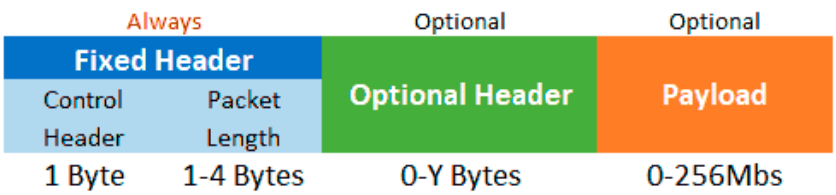


Figura 1.6. Arquitectura de una respuesta HTTP

1.4.8.3. Modbus TCP

Se trata de una variante del protocolo de Modbus como protocolo de aplicación orientado a la comunicación desarrollado para operar sobre TCP/IP para el transporte de datos utilizando el puerto por defecto 502 en un entorno de ejecución a través de internet o de intranet. Está publicada como un estándar de automatización [16].

La principal diferencia con el protocolo Modbus RTU radica en la arquitectura del mensaje que cuenta con un campo de dirección adicional correspondiente a la dirección IP del dispositivo objetivo.

De esta forma, un mensaje está conformado por:

- 2 bytes correspondientes al identificador de transacción utilizado para el emparejamiento de transacciones en los casos de solicitudes de conexiones múltiples.
- 2 bytes de identificador de protocolo, en este caso siempre corresponde al valor 0 para Modbus
- 2 bites correspondientes a la longitud que posee la trama de datos desde el identificador de unidad hasta el final de la trama.
- 1 byte de identificador de unidad que corresponde al identificador del dispositivo esclavo Modbus en el caso de requerir comunicar una red TCP y una red serial. Por defecto lleva el valor 0xFF. Se debe mencionar que este campo resulta irrelevante puesto que la dirección se la obtiene directamente con la dirección IP

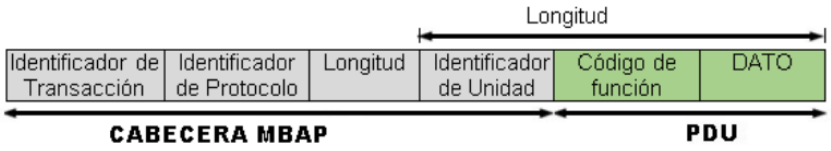


Figura 1.7. Arquitectura de un mensaje Modbus TCP

1.4.8.4. Consultas SQL

Se trata de un lenguaje estructurado de consultas y antes que ser un protocolo de comunicación, se trata de un lenguaje de programación que tiene como objetivo facilitar la definición, creación y modificación de la información almacenada en una base de datos de tipo relacional.

SQL cuenta con instrucciones o comandos que facilitan:

- La definición de tipo de datos que facilita el modelamiento de información dentro de la base de datos.
- La manipulación de datos como inserciones, consulta, actualizaciones, eliminaciones y peticiones.
- Mantener la integridad con la ayuda de restricciones personalizables.
- La definición de vistas de tablas
- El agrupamiento de datos
- La autorización de transacciones

2. METODOLOGÍA

En este apartado se describirá los requerimientos del historiador de procesos, el diseño de la propuesta para cumplir con el requerimiento establecido y la arquitectura para el mismo.

2.1. Diseño del historiador de procesos

Las características que definen a un historiador de procesos han ido variando a lo largo de los años gracias a los avances tecnológicos de la época, por lo que resulta complicado definir lo que representa un software de este tipo con exactitud; en la actualidad podemos encontrar variantes comerciales que cuentan con diferentes enfoques para sus aplicativos, mismos que pueden ser: de operabilidad, de conectividad y compatibilidad; todo en función de su modelo de negocio.

En este apartado se busca definir los requerimientos básicos que debe cumplir un aplicativo para poder tomar el nombre de Historiador de procesos.

2.1.1. Definición de los requerimientos básicos de un historiador de procesos

La definición actual de un software Historiador de Procesos varia de desarrollador a desarrollador dependiendo de su respectivo modelo de negocios. Por esta razón, para el desarrollo del presente proyecto se definirán requerimientos enfatizando las necesidades básicas con las que debe cumplir para poder operar e integrarse a un proceso industrial.

De manera que, el historiador de procesos propuesto se caracteriza por los requerimientos mencionados a continuación:

- El software desarrollado debe ser de código abierto, de manera que cualquier persona tenga acceso al código fuente del aplicativo para poder replicarlo, desplegarlo e incluso modificarlo.
- El software debe ser capaz de gestionar y mantener comunicaciones con múltiples clientes utilizando protocolos de comunicación tanto para el manejo de información de entrada como para la información de salida.
- El software debe permitir el registro de información previamente recolectada y organizada utilizando marcas de tiempo en una base de datos. De manera que se pueda definir el tiempo que dura la recolección de datos, así como el tiempo entre cada toma de datos.
- El usuario debe poder solicitar información generada en los registros históricos en el historiador, así como la información en tiempo real de las comunicaciones activas.
- El software debe contar con una interfaz gráfica que facilite la operabilidad del aplicativo.
- La lógica del manejo de información en los canales de entrada y de salida de datos del aplicativo debe permitir que entidades externas al historiador de procesos puedan consumir los servicios existentes en el mismo con el objeto de facilitar su integración.
- El aplicativo debe permitir la exportación de datos utilizando archivos de texto de valores separados por comas (.csv).
- El aplicativo debe contar con un fácil manejo, distribución e instalación de manera que pueda integrarse en la mayoría de sistemas operativos utilizados a nivel industrial.

En función de los requerimientos mencionados, a continuación, se establecen las herramientas de desarrollo que se utilizarán para la creación del aplicativo.

2.1.2. Elección de las herramientas de desarrollo a utilizar

De acuerdo a los requerimientos mencionados previamente, se debe elegir un lenguaje de programación de código abierto, el ambiente de ejecución del software y las herramientas de desarrollo.

2.1.2.1. Lenguaje de programación y entorno de ejecución

Existen varios caminos a tomar; en la actualidad, la tendencia del desarrollo de aplicativos se inclina al desarrollo de aplicaciones web debido a la facilidad que tiene el usuario de acceder al mismo mediante un navegador web de manera local o remota desde cualquier dispositivo móvil o de escritorio con acceso a internet sin la necesidad de realizar una instalación de software facilitando así la conexión de múltiples dispositivos que es uno de los requerimientos establecidos previamente.

Existen varios lenguajes de programación que cuentan con herramientas que facilitan el desarrollo de aplicaciones web. Para el presente proyecto se elige el lenguaje de código abierto, interpretado y orientado a objetos: JavaScript puesto que es el lenguaje de programación por defecto para este tipo de aplicaciones ya que corre de manera nativa en todos los navegadores web.

En consecuencia, el entorno de ejecución por defecto del software en JavaScript será node.js mismo que permitirá desarrollar, compilar y ejecutar el código del lado del servidor, permitiendo que cualquier dispositivo que conozca su dirección pueda conectarse y consumir sus servicios. Node.js se caracteriza por ser una de las alternativas de desarrollo, implementación y despliegue más rápidas en la actualidad.

Una vez definido el lenguaje de programación y el entorno de ejecución se procede a determinar las herramientas o también llamados frameworks que facilitarán el desarrollo de la aplicación. Toda aplicación web, bajo el estándar actual, puede ser dividida en dos partes principales: el front-end: que corresponde a la interfaz que permite la interacción del usuario con la lógica del aplicativo; y el back-end, que corresponde a toda la lógica del aplicativo que permite canalizar y organizar la información de acuerdo a un modelo establecido.

El back-end utilizará el framework express.js que contiene un conjunto de herramientas de desarrollo que facilitan el enrutamiento, gestión de memoria, conectividad, entre otros; de un software.

Por otro lado, el front-end funcionará con Vue.js, que se trata de un conjunto de herramientas para el desarrollo de interfaces visuales de aplicación y de usuario.

2.1.2.2. Comunicaciones

En función de los requerimientos establecidos, el aplicativo debe ser capaz de mantener comunicación y permitir el tráfico de datos de entrada y salida del mismo proveniente de dispositivos externos que se encuentren en un nivel de campo, de control o de supervisión dentro de la pirámide de automatización.

La información de entrada al aplicativo se caracteriza por:

- Ser ligera, es decir que la unidad de memoria que ocupa una muestra o registro de datos es poca.
- No ser compleja, es decir que el tipo de dato que utiliza para un registro se limita a un texto o un número.
- Tener un gran volumen, es decir que un dispositivo puede enviar un registro tras otro al aplicativo a la par que hay otros dispositivos realizando la misma acción. Es importante considerar que a pesar de ser abundante sigue siendo ligera.

De esta forma para el tráfico de entrada al aplicativo se elige el protocolo MQTT que es un protocolo de comunicaciones Pub/Sub basado en una cola de mensajes de máquina a máquina (M2M) muy popular en el internet de las cosas (IoT). Este protocolo permite cumplir con el requerimiento de conexiones simultáneas múltiples establecido para el aplicativo ya que se acopla perfectamente con el modelo de información que se maneja.

Con el objeto de cumplir el requerimiento de extensibilidad del aplicativo para con aplicativos externos, se establece que el apartado del aplicativo encargado del tráfico de datos de entrada será desarrollado de tal forma que permita una extensión hacia otros protocolos de comunicación. Dentro del alcance del presente proyecto se establece agregar los protocolos Modbus TCP y las consultas SQL.

Por el otro lado, con respecto a la información de salida del aplicativo se considera que:

- El usuario debe poder solicitar la información tanto histórica como en tiempo real de las conexiones activas.
- El flujo de datos de salida dependerá de las conexiones de entrada activas, así como de la cantidad de información en el caso de solicitar registros históricos existentes.
- Los datos de salida deben poder ser visualizados gráficamente en una interfaz web desde un navegador web.

De esta forma, el protocolo de comunicación que facilidad el manejo de datos de salida de la aplicación es el protocolo HTTP que es uno de los protocolos más populares y utilizados para el intercambio de información en la World Wide Web.

2.1.2.3. Empaquetado del software

El desarrollo del aplicativo se lo plantea basado en un paradigma modular, es decir que se procura dividir al aplicativo en “módulos” que son auto funcionales y que al ser integrados cumplan con el cometido del historiador de procesos propuesto.

Cada uno de los módulos tendrá su lógica funcional y podrá comunicarse con otros módulos a través de TCP/IP.

De esta forma, para el empaquetado del software, que facilitará su distribución e instalación, se utilizará Docker que se trata de una herramienta de automatización de despliegue de software utilizando contenedores mismos que cuentan con todo lo necesario para la ejecución del mismo. De esta forma cada uno de los módulos que conforman el historiador de procesos ocuparían un contenedor dentro de Docker.

Finalmente, para facilitar aún más el despliegue y configuración de las comunicaciones entre contenedores de Docker se utilizará la herramienta Docker-compose que facilita la definición del ambiente en dónde se encuentran funcionando los contenedores a través de un archivo de texto, mismo que se considerará como el “instalador” del proyecto.

2.1.3. Arquitectura preliminar del aplicativo

El aplicativo está divido en 5 apartados, entidades o módulos y son:

1. Un módulo de persistencia de datos conformado por la base de datos PostgreSQL y una funcionalidad que permita automatizar la definición e inicialización de la misma.

2. Un módulo de comunicaciones de entrada, que se trata de un servidor web encargado de permitir a dispositivos externos comunicarse con el historiador de procesos utilizando el protocolo MQTT y transferir la información respectiva al contenedor de la base de datos.

3. Un agente de comunicaciones que haciendo uso del módulo de comunicaciones de entrada, será el encargado de traducir al protocolo MQTT otras consultas que se basen en los protocolos Modbus TCP y SQL. El agente contará con una interfaz gráfica web que permita configurar dichas conexiones.

4. Un módulo de comunicaciones de salida, que permite realizar consultas HTTP para posteriormente traducirlas a consultas SQL hacia el módulo de la base de datos para extraer los históricos de la misma en función de las marcas de tiempo deseadas.

5. Una interfaz gráfica web que permitirá monitorear los datos históricos y en tiempo real del historiador de procesos.

En la Figura 2.1 se puede observar el esquema gráfico de la aplicación propuesta:

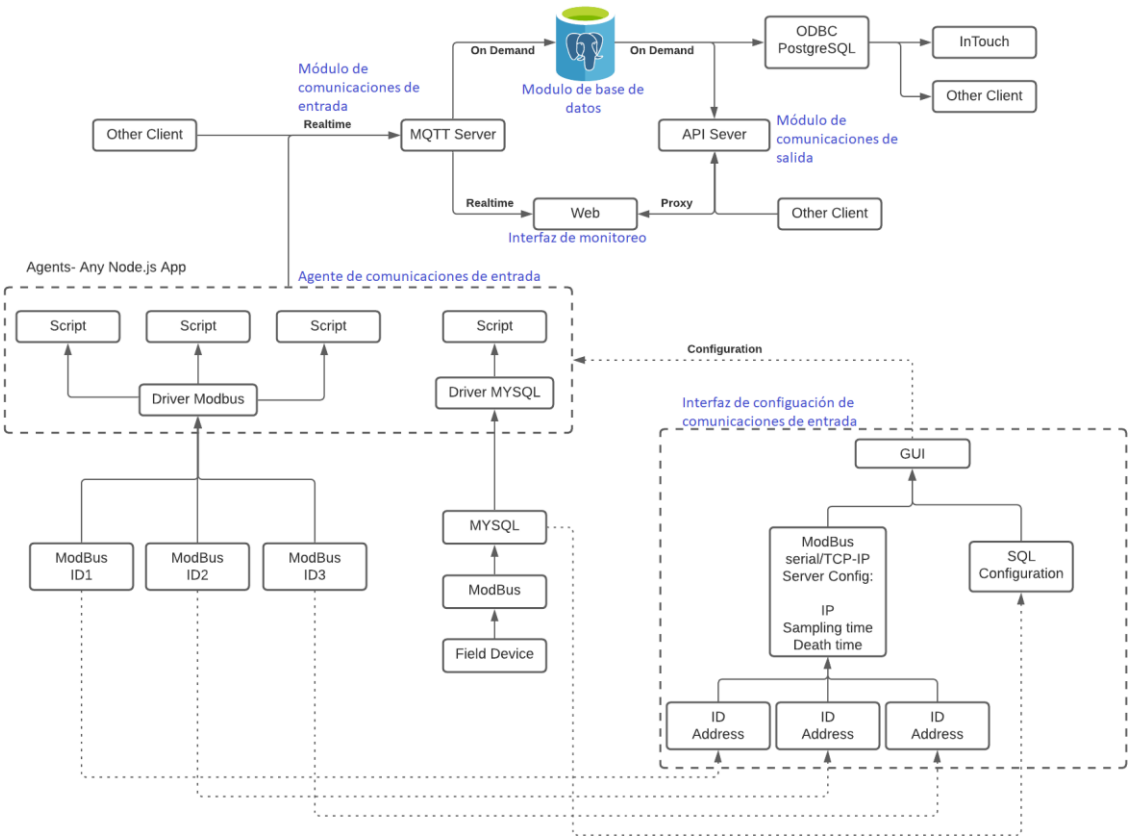


Figura 2.1. Esquema gráfico preliminar del aplicativo descrito previamente

El diseño del aplicativo está pensado de manera que cada módulo sea independiente de los demás. De esta forma se los puede empaquetar dentro de contenedores de Docker con el fin de automatizar el despliegue y la distribución del software.

2.2. Arquitectura de la base de datos

En este apartado se define el módulo de la base de datos, nombrado como “historiador-db”, que se utilizará para respaldar los históricos generados por la aplicación.

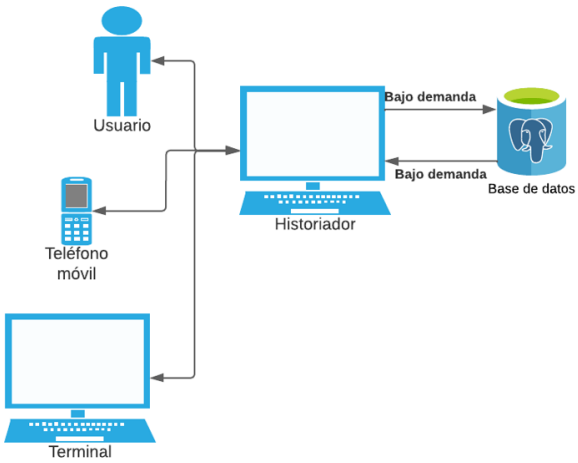


Figura 2.2. Esquematización de la arquitectura del módulo de base de datos

2.2.1. Selección de la base de datos

Para la elección de la base de datos se parte del modelo de datos que debe manejar el historiador de procesos, existen dos tipos de modelos principalmente, relacionales y no relacionales.

Los datos del historiador de procesos corresponden a un conjunto de valores de texto y números que representan el estado de una variable definida dentro de una lógica de control asociados a una entidad que emite dichos datos. Es decir que existen datos asociados o “relacionados” a una o varias entidades.

De esta forma el modelo de datos que concuerda con el modelo del aplicativo es el modelo relacional. Por esta razón se opta por la utilización de un motor de base de datos de tipo relacional.

La base de datos elegida es PostgreSQL, que se trata de uno de los motores de tipo relacional y de código abierto más potentes y populares de la actualidad, considerado como un motor de base de datos idéntico a MySQL pero con funcionalidades mucho mas variadas y complejas debido a sus capacidades de manejo de datos y a su capacidad de extensibilidad nativa con muchos de los lenguajes de programación de medio/alto nivel.

Cada uno de los módulos de la aplicación web podrá interactuar con la base de datos utilizando la técnica de mapeo objeto-relacional (ORM [17]), que básicamente permite la conversión de los tipos de datos utilizados en el lenguaje de desarrollo de la aplicación, en este caso el lenguaje orientado a objetos JavaScript, y los tipos de datos que maneja el motor de la base de datos.

2.2.2. Definición de las entidades de la base de datos

Debido a que PostgreSQL es una base de datos de tipo relacional se deben definir las tablas asociadas a las entidades que la conforman.

Para este caso se propone utilizar dos entidades: Agentes y Métricas.

Un agente sería todo aquel dispositivo o entidad que se conecte al historiador de procesos, más específicamente al módulo de comunicaciones de entrada, con el propósito de transferir información para que perdure en la base de datos. La estructura de esta entidad se la representa en la Tabla 2.1.

Tabla 2.1. Estructura de la entidad “Agente” dentro de la base de datos

Nombre del atributo en la base de datos	Descripción
id	Es la llave primaria auto incremental con la que se identificará a cada agente.
uuid	Es un código único generado aleatoriamente para cada agente.
name	Nombre del agente (dispositivo o entidad).
username	Nombre de usuario dueño del agente que permitirá definir permisos internos de funcionalidad.
hostname	Dirección del nodo del agente.
pid	Es un identificador de procesos que permite agrupar a agentes.
connected	Se trata de una bandera que indica el estado de conectado o desconectado del agente hacia el historiador de procesos.
topic	Es un identificador de propósito general que permite organizar y agrupar agentes.
createdAt	Fecha de la creación del agente.
updatedAt	Fecha de la última actualización del agente

Una métrica puede ser cualquier unidad de información, dato o valor transmitido por un agente. Sus atributos se describen en la Tabla 2.2.

Tabla 2.2. Estructura de la entidad “Métrica” dentro de la base de datos

Nombre del atributo en la base de datos	Descripción
id	Es la llave primaria auto incremental con la que se identificará a cada métrica.
agentId	Es la llave de referencia que permite relacionar cada métrica con su correspondiente agente.
type	Es un campo de tipo texto que permite nombrar una métrica. Adicionalmente este campo permite diferenciar entre el tipo de

	dato cargado en el atributo “value”, de manera que si la cadena de caracteres ingresada inicia con un “bool_”, se considera que el valor de la métrica será de tipo booleano o digital
value	Unidad de información que se desea perdurar en la base de datos.
createdAt	Fecha de la creación del agente.
updatedAt	Fecha de la última actualización del agente

La relación entre agentes y métricas es de cero a muchas dado que un agente puede llegar a tener o cero métricas o muchas métricas. De esta manera se puede esquematizar gráficamente las entidades propuestas tal como se muestra en la Figura 2.1.

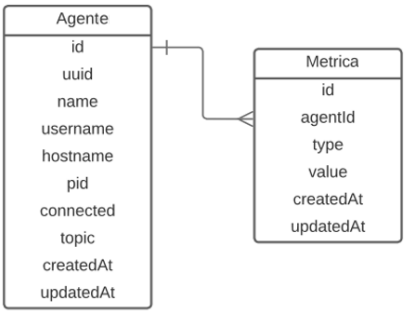


Figura 2.3. Resultados de las pruebas realizadas

Existen básicamente dos formas con las que se pueden definir y modificar las entidades mencionadas en la base de datos: Utilizando consultas SQL planas, es decir realizando manualmente cada instrucción hacia la base datos; o se puede hacer uso de herramientas de software, comúnmente llamadas librerías, con las que se puede automatizar la definición y modificación de las mismas.

Para el presente proyecto se opta por la segunda opción, donde la herramienta de software a utilizarse es conocida como “sequelize”, que es una librería que recopila las abstracciones de varias bases de datos, entre ellas PostgreSQL, en objetos manejables por el lenguaje de programación lo que nos permite definir la base de datos y sus entidades utilizando código de JavaScript.

2.2.3. Creación e inicialización de la instancia de base de datos

Lo primero que se debe realizar es la instalación del motor de base de datos. En la documentación oficial se pueden encontrar las instrucciones a seguir para instalar PostgreSQL en diferentes sistemas operativos. Para el presente proyecto se utilizará un

contenedor de Docker de PostgreSQL en su versión 13. El proceso de instalación del contenedor se lo puede encontrar en la sección 2.7.2 del presente documento.

Como se mencionaba anteriormente, se automatizará la inicialización de la base de datos utilizando abstracciones de código en JavaScript que representan las entidades y sus correspondientes atributos. De manera que se desarrolla un módulo que brinde el servicio de inicialización y, adicionalmente, de modificación de registros; de tal forma que cualquiera de los otros módulos que conforman el historiador de procesos pueda realizar consultas de creación y modificación hacia los registros de la base de datos.

El flujo de la funcionalidad que automatiza la creación, definición e inicialización de la base de datos inicia validando si existe una conexión activa con dicha entidad, si no existe se crea una nueva conexión hacia la base de datos utilizando los parámetros que permiten inicializar una nueva conexión con el host (Endpoint, puerto, las credenciales de usuario, el nombre de la base de datos); en el caso de existir algún error durante la conexión se presenta un mensaje indicando que la conexión falló y se termina el flujo. Si la conexión con la base de datos es estable, se procede a validar la existencia de las tablas “Agents” y “Metrics”, si existen entonces borra todo lo existente. Finalmente crea las tablas “Agents” y “Metrics”, sus atributos y relaciones acorde al esquema definido en el apartado 2.2.2.

A continuación, se presenta el diagrama de flujo que representa la funcionalidad descrita desarrollada en JavaScript:

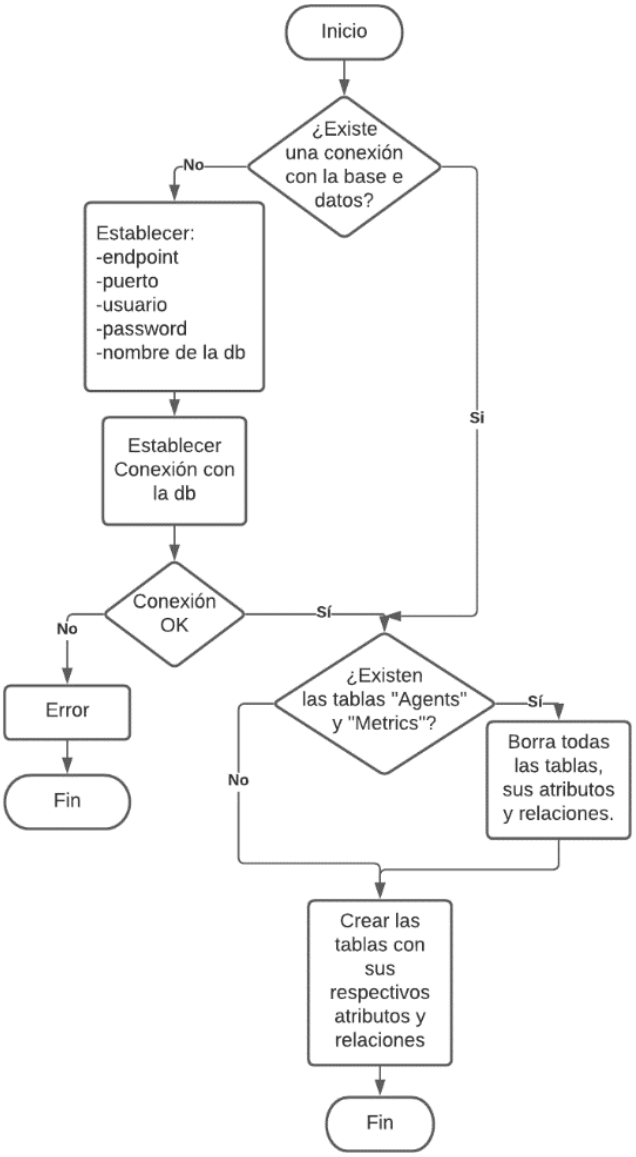


Figura 2.4. Diagrama de flujo de inicialización de la base de datos.

2.2.4. Lectura y escritura de información en la base de datos

Con el objeto de realizar consultas para la creación y modificación de registros en la entidad de la base de datos definida se establece un banco de funciones utilizando la librería “sequelize” de manera que una entidad pueda hacer uso de las mismas con el objeto de ingresar información o se realizar consultas sobre los registros existentes. Se pueden agrupar las funcionalidades de lectura y escritura de información acorde a las entidades que se dispone en la base de datos.

Las funcionalidades definidas de lectura y escritura en la tabla de Agentes son las siguientes:

- createOrUpdate(): es una función que recibe los parámetros asociados a los atributos de la entidad “Agents” y se encarga de crear un nuevo registro dentro de la tabla de agentes utilizando dichos parámetros de entrada. En el caso de que ya exista un registro que coincida con los parámetros de entrada en lugar de crear un nuevo registro, actualiza el existente.
- finById(): es una función que recibe un número entero como parámetro y que permite filtrar los agentes por su atributo “id” y retornar uno de ellos en función del parámetro de entrada.
- findAll(): es una función que nos permite filtrar un conjunto de agentes en función de su estado de conexión
- findConnected(): esta función nos permite filtrar un conjunto de agentes que tengan un estado de “conectado”.
- findByUsername(): Esta función permite filtrar un conjunto de agentes en función de su campo “username”.

Por otro lado, las funciones de lectura y escritura definidas para la entidad “métricas” son:

- create(): esta función crea un nuevo registro de métrica asociada a un agente. Para ello recibe dos parámetros de entrada, el uuid del agente y el objeto de métrica que contiene la información.
- findByAgentUuid(): la función permite filtrar un grupo de métricas en función de su atributo “agentId”. De manera que permite filtrar todas las métricas asociadas a un agente determinado.
- findByTypeAgentUuid(): esta función permite filtrar un grupo de métricas en función de sus atributos “type” y ”uuid”, de manera que retorna la métrica asociada a un agente con determinado “type”.
- findByAgentDate(): esta función permite filtrar un grupo de métricas en función de sus marcas de tiempo.

Todas estas funcionalidades son públicas para todo el proyecto, de manera que cualquier otro módulo lo puede importar con el objeto de poder utilizarlas a conveniencia.

2.3. Módulo de comunicaciones de entrada

En este apartado se define la estructura del módulo de comunicaciones de entrada hacia el historiador de procesos, nombrado como “historiador-mqtt”.

El principal objetivo de este módulo es recibir y respaldar en la base de datos la información que proviene de los diferentes dispositivos agentes utilizando el protocolo MQTT.

Este módulo se caracteriza por ser una integración directa a un broker MQTT, es decir que el módulo se construye sobre un servidor que hace las veces de un broker MQTT y, por otro lado, se caracteriza por hacer uso de los eventos ejecutados por los clientes que se subscriben o publican mensajes en los tópicos del broker MQTT.

Su enfoque no se centra en la redistribución de mensajes publicados en un determinado tópico hacia todos los clientes subscritos a dicho tópico, sino más bien en la interpretación del mensaje publicado en un tópico por los clientes para, posteriormente, perdurar dicho mensaje en la base de datos definida en 2.2.

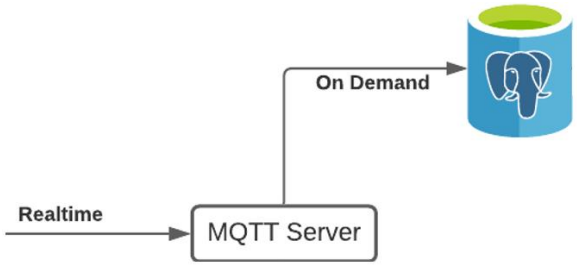


Figura 2.5. Esquematización del módulo de comunicaciones de entrada (historiador-mqtt)

2.3.1. Broker MQTT

De acuerdo con lo que se mencionó en el apartado 1.4.8.2, un broker MQTT permite la conexión de múltiples dispositivos y se encarga de redistribuir los mensajes publicados en tópicos hacia todos los clientes subscritos a dichos tópicos. Cada vez que se conecta o desconecta un cliente al broker, se suscribe o se des-suscribe un cliente a un tópico o se publica un mensaje en un tópico, se emite un evento determinado que permite la ejecución de funciones de código.

En este caso se utilizará la herramienta de libre acceso “Mosca” [18], misma que nos permitirá definir funciones de back-end que opere como un emisor de eventos MQTT, tal como se describió previamente, y poder definir un flujo de eventos que se deben cumplir para enviar información hacia la base de datos definidos en 2.2.

Se definirán 3 tópicos en los cuales se publicarán los diferentes tipos de mensaje:

- agent/connected
- agent/disconnected
- agent/message

Los eventos desencadenados estarán directamente asociados a los mensajes MQTT de los diferentes clientes que se publiquen en los diferentes tópicos. De esta forma la aplicación operará sobre los siguientes eventos:

- Evento de conexión
- Evento de desconexión
- Evento de mensaje publicado

Cada uno de los eventos está asociado a los diferentes tipos de mensajes que se propone utilizar y se encargará de ejecutar una o varias de las funciones de lectura y escritura mencionadas en el apartado 2.2.4, asociadas a la creación de nuevos registros en la base de datos.

2.3.2. Definición de los mensajes MQTT

El módulo de comunicaciones de entrada propuesto admite 3 tipos de mensajes MQTT:

- Estado de conexión: Con este mensaje un dispositivo indica que va a dar inicio a la transferencia de información hacia el historiador de procesos.

  Lo que hace este mensaje es modificar el atributo “connected” de la entidad correspondiente al Agente en la base de datos. Para ello lleva todos los parámetros asociados a un determinado Agente, es decir su uuid, name, hostname y pid.

  Este mensaje debe ser publicado en el tópico “agent/connected” del broker MQTT y debe tener la siguiente estructura:

```
{
    agent: {
        uuid, // identificador único, se genera automáticamente
        username, // nombre del dueño del agente, definido por el usuario
        name, // nombre del agente, definido por el usuario
        hostname, // endpoint del sistema operativo
        pid // identificador de proceso, definido por el usuario
        topic  // identificador de tópico, definido por el usuario
    }
}
```

- Estado de desconexión: Mediante este mensaje la aplicación puede identificar el estado de desconexión de un dispositivo.

  Al igual que el mensaje de estado de conexión, este mensaje modifica el atributo “connected” de la entidad correspondiente al Agente en la base de datos. En este caso el mensaje debe llevar como parámetro únicamente el identificador único (uuid) del agente previamente registrado como conectado en la base de datos.

  Este mensaje debe ser publicado en el tópico “agent/disconnected” del broker MQTT y debe tener la siguiente estructura:

```
{
    agent: {
        uuid, // identificador único, se genera automáticamente
    }
}
```

- Mensaje de métrica: Este mensaje incluirá la información de un agente que se desea perdurar en el historiador de procesos siempre y cuando dicho agente se encuentre con un estado de conexión al historiador.

  Este mensaje lleva como parámetros la información respectiva de un determinado agente, así como las métricas a registraste. La aplicación se encargará de generar las marcas de tiempo en función del instante en el que re recibe e interpreta este mensaje.

  Este mensaje debe ser publicado en el tópico “agent/message” del broker MQTT y debe tener la siguiente estructura:

```
{
    agent, // identificador del agente
    metrics: [
        {
            type, // nombre de la métrica
            value //valor de la métrica
        }
    ], // lista de métricas a publicar
```

timestamp, *// marcas de tiempo, se generan automáticamente*

}

2.3.3. Integración de los eventos MQTT con el módulo de la Base de Datos

Como se mencionó previamente, el broker se encontrará a la espera de que ocurra un evento que ejecute una o varias de las funciones definidas en el módulo de la base de datos. De manera que el módulo de comunicaciones hace uso de la lógica definida en el módulo de la base de datos para establecer comunicación y ser capaz de realizar consultas de escritura y lectura de información.

Por otro lado, cada vez que un dispositivo publica un determinado tipo de mensaje MQTT en un tópico específico del broker MQTT se emitirá un determinado evento.

A continuación, se muestra un diagrama de flujo que describe el comportamiento del módulo de comunicaciones de entrada.

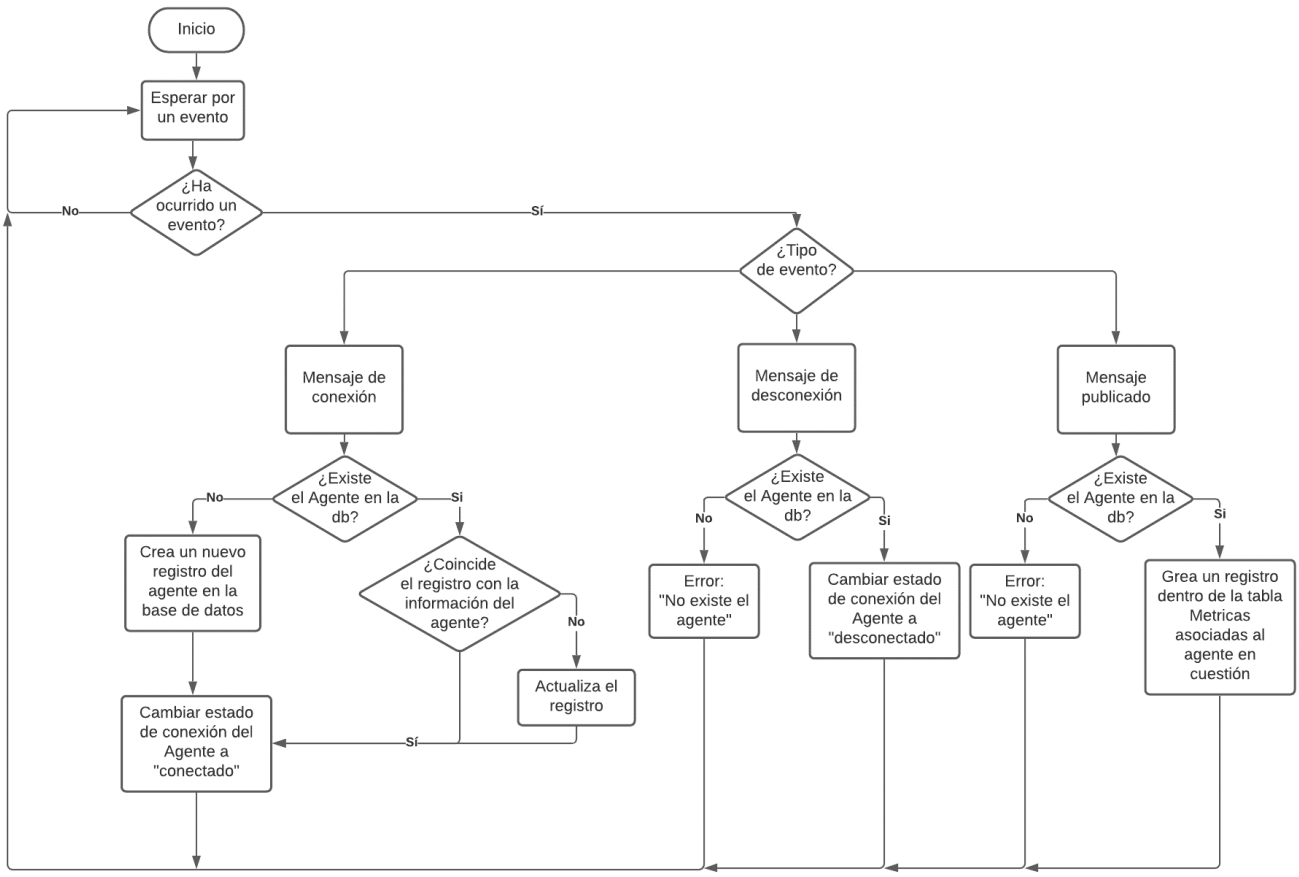


Figura 2.6. Diagrama de flujo de los eventos MQTT de conexión, desconexión y mensaje publicado.

De esta forma, cada vez que llega un mensaje a uno de los 3 tópicos definidos en el broker, se realiza un flujo de validación para identificar el tipo de mensaje, su contenido y a su vez, el tipo de evento desencadenado por el mismo de manera que se tienen 3 casos posibles:

- Si es un evento de conexión, se valida la existencia del registro del agente que se desea conectar para, en función de ello, crear el registro o actualizarlo con el estado de “Conectado”.
- Si se trata de un evento de desconexión, se valida la existencia del agente en cuestión para actualizar su registro y cambiar su estado a “Desconectado”
- Si es un evento de mensaje publicado, se valida la existencia del agente para posteriormente crear un nuevo registro de métricas asociadas a dicho agente.

Bajo este paradigma, cualquier dispositivo o agente que conozca las credenciales del broker, los tópicos definidos, los tipos de mensaje y el contenido que deben tener, puede hacer uso de las funcionalidades descritas en este módulo.

2.4. Agente de comunicaciones de entrada

En el presente apartado se describirá un módulo dentro de la aplicación que funcionará como un agente de comunicaciones, por esta razón se lo nombró como “historiador-agent”.

Como se mencionó en el apartado anterior, cualquier dispositivo que conozca los parámetros y credenciales de conexión al broker, así como los tópicos y las estructuras de los mensajes MQTT, puede consumir los servicios del módulo “historiador-mqtt” y técnicamente ser considerado como un agente dentro del modelo de la aplicación puesto a que será capaz de detonar los eventos MQTT descritos en 2.3. Sin embargo, ese flujo no está considerado como un flujo válido debido a que no se tendría control sobre quién o cuándo ha detonado un evento y por esa razón se requiere de un agente de comunicación que controle el tráfico de datos de entrada al historiador de procesos a la par que cumpla con el flujo de la lógica de eventos MQTT.

El módulo “historiador-agent” tiene como objetivo satisfacer la necesidad previamente mencionada de manera que es capaz de recolectar datos de diferentes fuentes de información haciendo uso de un driver de comunicación que le permita comunicarse por diferentes protocolos de comunicación a la par de cumplir con el flujo de eventos MQTT definido para el módulo de comunicaciones de entrada descrito en 2.3.

La operabilidad del módulo de agente de comunicaciones se basa en un conjunto de funcionalidades que pueden ser reutilizadas para cumplir con el flujo de datos definido en el módulo “historiador-mqtt”. Así, se puede incorporar la capacidad de comunicación con dispositivos que operan sobre:

- El protocolo MQTT.
- El protocolo Modbus TCP
- El lenguaje de consulta estructurado (SQL).

De esta forma, dentro de la lógica propuesta, un Agente puede ser dividido en 3 partes:

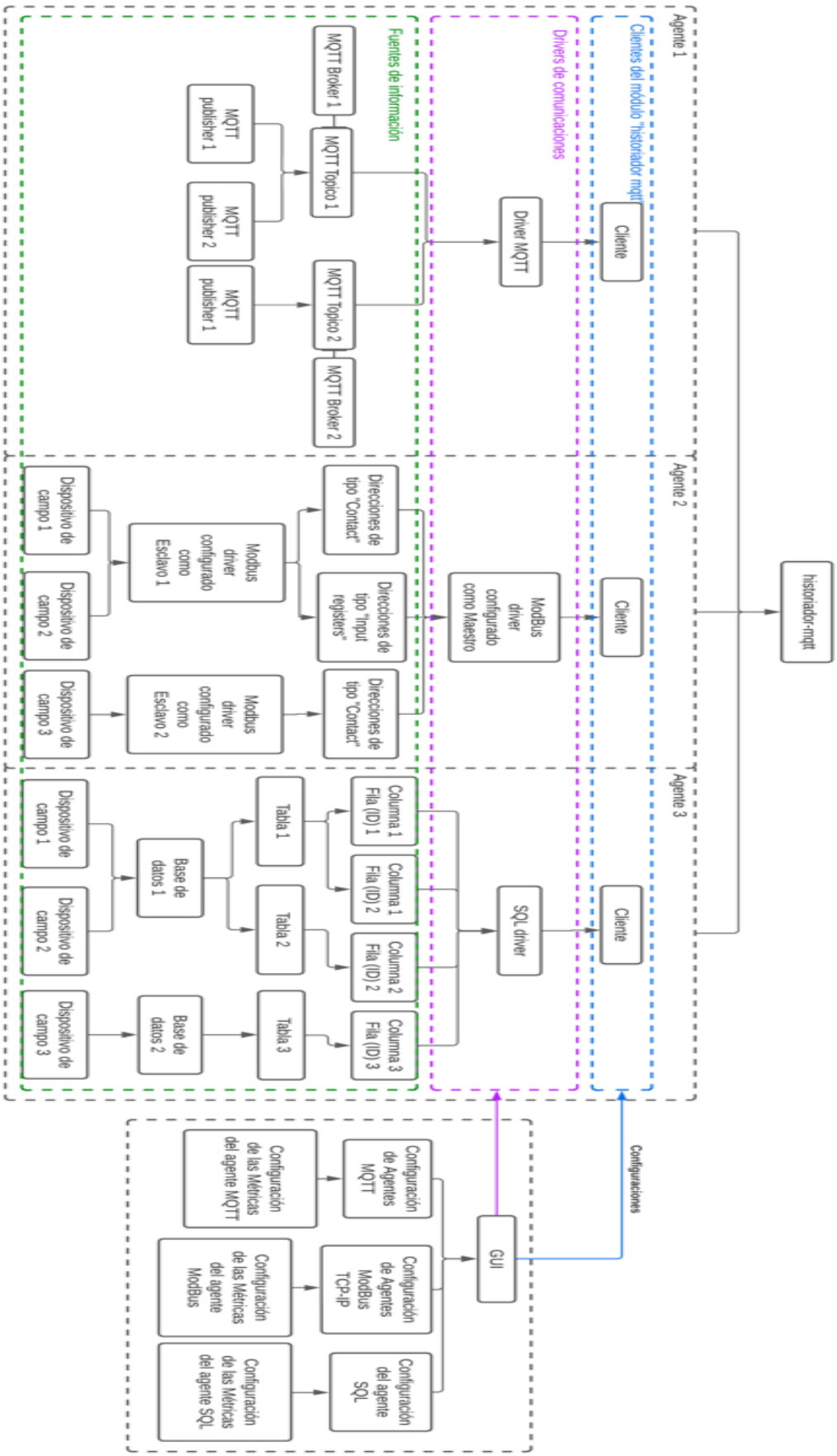
- La fuente de información: se trata de un dispositivo, sistema embebido, unidad de procesamiento o entidad que pueda comunicarse mediante los protocolos descritos previamente.
- Un driver de comunicaciones: que habilite la capacidad de comunicación mediante los protocolos mencionados al agente de comunicaciones de entrada.
- Un cliente del módulo “historiador-mqtt”: Se trata de un script construido en JavaScript que se encarga de tomar datos desde la fuente de información haciendo uso del driver de comunicaciones pertinente y consumir el flujo de eventos requeridos por el módulo “historiador-mqtt”.

Con el objeto de facilitar la configuración de las comunicaciones al operador del historiador de procesos se incorpora una interfaz web en donde se puede establecer las credenciales de conexión e información inherente a las entidades definidas en la lógica de la aplicación para los Agentes, previamente descritos, y para las Métricas, publicadas por dichos Agentes.

Así, se puede representar la arquitectura de este módulo mediante el siguiente esquema gráfico de ejemplo para tres agentes de tipo: MQTT, Modbus y SQL, mismos que se encuentran encerrados en un recuadro negro del lado izquierdo de la Figura 2.7. De manera adicional se puede divisar las partes que conforman cada uno de los agentes: De color azul se encuentran encerrados los clientes del módulo “historiador-mqtt”, de color morado se encuentran encerrados los drivers de comunicación respectivos para cada protocolo utilizado por la fuente de información y de color verde se encuentran encerradas las diferentes fuentes de información que operan sobre diferentes protocolos de comunicación ya mencionados. Del lado derecho de la figura 2.7 se puede visualizar el

esquema de la interfaz gráfica encargada de recopilar la información de configuración de los Agentes y de las métricas respectivas de cada agente.

Figura 2.7. Esquematización del módulo “historiador-agent”



2.4.1. Cliente del módulo “historiador-mqtt”

Lo que se busca es poder definir un cliente del módulo “historiador-mqtt” como una clase reutilizable de JavaScript de manera que sea el puente entre uno o varios drivers de comunicación.

El cliente del módulo “historiador-mqtt” tiene la capacidad de emitir eventos MQTT acorde a una lógica de tiempos establecida de manera que el usuario pueda definir el tiempo de retardo entre mensaje y mensaje, denominado tiempo de muestreo, y el tiempo total que dura la recolección de los mensajes desde el driver de comunicaciones, denominado tiempo de registro.

Considerando que pueden existir varios agentes y que cada agente puede publicar varias métricas, se opera de manera asíncrona, con el objeto de que los tiempos de registro y los tiempos de muestreo no se vean afectados al tener múltiples fuentes de datos.

La detonación de los diferentes eventos MQTT por parte de un agente de comunicación se detalla a continuación:

- Evento de conexión del agente: se ejecuta una vez se hayan proporcionado la información de configuración del driver pertinente de cada protocolo de comunicación además de la variable que representa el tiempo total que durará la conexión (tiempo de registro), y servirá para establecer un timer que una vez haya cumplido su tiempo, emita el evento de desconexión del agente. Durante el lapso definido, el cliente construirá mensajes MQTT que se enviarán uno tras otro con un retardo de tiempo o también llamado tiempo de muestreo.

  Internamente se construye el mensaje de “Estado de Conexión”, tal como se describe en 2.3.2, que será enviado al módulo “historiador-mqtt” con el objeto de crear un nuevo registro en la tabla de Agentes en la base de datos con el estado “connected” y así dar inicio al envío de mensajes de tipo métrica.

- Evento de desconexión del agente: este evento no recibe ningún parámetro de entrada y es desencadenado por el timer definido en el evento de conexión del agente haciendo que se detenga la generación y envío de mensajes de tipo métrica hacia el módulo historiador-mqtt.

  Una vez detenido el envío de mensajes de tipo métrica, internamente se construye el mensaje de “Estado de Desconexión”, tal como se describe en 2.3.2, de tal manera que en el atributo de conexión en el registro en la tabla de agentes correspondiente sea modificado a “desconnected”.

- Evento de mensaje publicado del agente: este evento recibe como parámetros de entrada los atributos “type” y “value” con los que se construirá los mensajes de “Métrica” que serán enviados al módulo “historiador-mqtt”, adicional recibe el tiempo de muestreo que definirá el tiempo entre cada uno de los mensajes de tipo métrica que serán construidos y enviados.

  Este evento es desencadenado una vez que se haya dado inicio al evento de conexión del agente y se detendrá cuando el evento de desconexión del agente haya sido desencadenado.

2.4.2. Reutilización del cliente del módulo “historiador-mqtt”

Como ya se mencionó previamente, el cliente del módulo “historiador-mqtt” es una entidad encargada de facilitar el envío de información desde un driver de comunicaciones hacia la base de datos.

Todo agente que use el cliente del módulo “historiador-mqtt” deberá seguir un flujo de manera que se consuman los eventos definidos en el mismo en el orden requerido. De esta forma, primero se deberá invocar e instanciar el cliente, posteriormente se deberá consumir el evento de conexión del agente enviándole sus respectivos parámetros de entrada, tal como se define en el punto 2.4.1. Ocurrido el evento de conexión, se dará inicio al evento de mensaje publicado del agente en donde se envía una serie de mensajes con un tiempo de espera entre mensajes (tiempo de muestreo) con los parámetros “Type” y “Value” en donde se debe colocar la información que se desee perdurar en el historiador. En cualquier momento se puede desencadenar el evento de desconexión del agente lo que ocasiona un alto en la generación y emisión de mensajes publicados del agente.

A continuación, en la figura 2.8, se muestra el flujo de proceso descrito previamente, que un módulo, entidad o aplicación deberá ejecutar para hacer uso del cliente MQTT y sus funcionalidades:

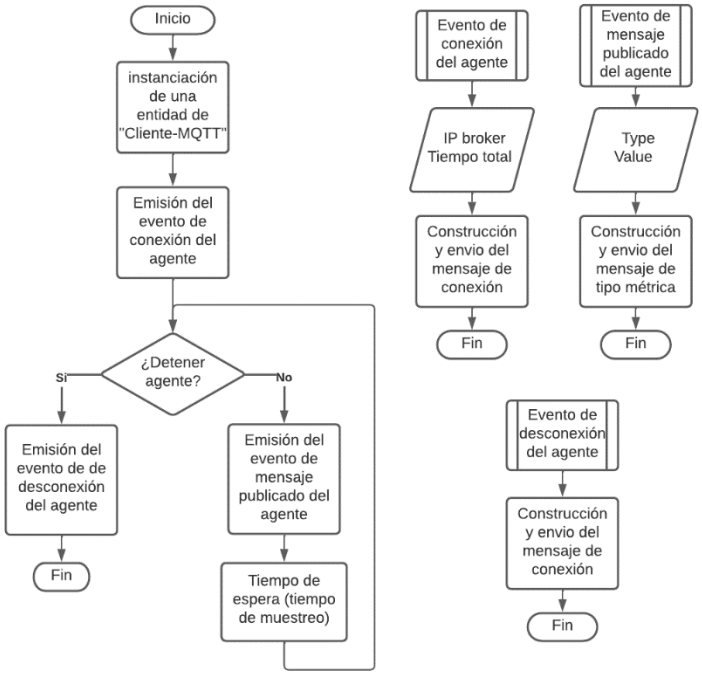


Figura 2.8. Esquematización del módulo “historiador-agent”

2.4.3. Driver de comunicación Modbus TCP

Tal como se mencionó previamente, cualquier aplicación escrita en Javascript puede hacer uso de las funcionalidades del cliente del módulo “historiador-mqtt” al invocarlo como si de una librería se tratase.

Tal como se menciona en el inicio del presente capítulo, se desarrolla un driver que se encarga de establecer conexión, entre el cliente del módulo “historiador-mqtt” y cualquier dispositivo o entidad configurada como esclavo Modbus TCP utilizando la librería de comunicaciones “Modbus-serial”, que incorpora funcionalidades para establecer comunicación mediante Modbus RTU serial y Modbus TCP [19]. Dicho driver hará uso de las funcionalidades del cliente del módulo “historiador-mqtt”, descritas en 2.4.1 y 2.4.2, para enviar la información recolectada a través del protocolo Modbus TCP.

La librería de comunicaciones “Modbus-serial” facilita la creación de una entidad que hace las veces de servidor Modbus TCP de tipo maestro de base 1 con un tamaño de registro de 5 bits utilizando código de Java Script, que permite tomar información de un dispositivo configurado previamente como esclavo. La entidad maestra requiere la información asociada a la dirección IP del esclavo, su identificador único Modbus, y la dirección de los “Contacts” e “Input register” de interés. Con dicha información puede iniciar una conexión con el dispositivo objetivo para enviar mensajes de código de función 4 (FC4). Cada uno

de los mensajes FC4 será enviado una vez que el cliente MQTT emita el “evento de mensaje publicado del agente” tal como se describe en 2.4.1.

De esta forma, se puede definir un Agente en función del esquema mencionado en 2.4.2, es decir como un conjunto de 3 entidades:

- Un esclavo Modbus, que es el dispositivo objetivo de donde se toma la información a perdurar en el historiador de procesos.
- La entidad que hace las veces de un Maestro Modbus encargada de solicitar información del esclavo a través de mensajes FC4, es decir el driver de comunicaciones y poner dicha información a disposición del cliente MQTT.
- El cliente del módulo “historiador-mqtt”, cuyas funcionalidades serán consumidas por la entidad Modbus maestro para el envío de información tomada del eslavo y enviarla a la base de datos.

Con el objeto de facilitar la configuración y establecimiento de la conexión al operador del historiador de procesos se diseña la entidad como un servicio que puede ser ejecutado a través de una interfaz gráfica (que será descrita en el apartado 2.4.6). Dicha entidad deberá seguir el flujo descrito en 2.4.2 y este apartado para inicializar un nuevo Agente.

2.4.4. Driver de comunicación de Consultas SQL

Al igual que con el Agente basado en el driver de comunicaciones Modbus TCP, se desarrolla otro agente basado en un Driver que permite establecer comunicación con una base de datos de tipo relacional que admita consultas SQL.

En este caso se utiliza la librería denominada “mysql2” que permite realizar consultas de tipo SQL utilizando funciones del lenguaje JavaScript. Para ello se requiere la información de conexión a la base de datos: su dirección y credenciales de acceso, el nombre de la base de datos, el nombre de la tabla y el nombre de las propiedades (columnas) de donde se requiere leer información.

De esta forma el Agente basado en el driver de comunicaciones de consultas SQL puede definirse como la suma de:

- Una base de datos basada en el motor MySQL que cuente con al menos una tabla de destino con dos columnas. Una de ellas corresponde a la columna de donde se requiere leer la información a perdurar en el historiador de procesos y la otra columna corresponde a la llave primaria que es el identificador del número de registros con los que cuenta la tabla.

- Un driver de comunicaciones que apunta a la base de datos objetivo utilizando los parámetros de conexión necesarios y que toma la información de la tabla objetivo para ponerla a disposición para el cliente del módulo “historiador-mqtt”.

  Es importante mencionar que la lectura realizada será del último registro creado en la tabla objetivo dentro de la base de datos en cuestión. Cada lectura será realizada una vez que el “evento de mensaje publicado del agente” sea desencadenado, acorde a lo descrito en 2.4.2.

- El cliente del módulo “historiador-mqtt” que toma la información proveniente del enlace de comunicaciones con la base de datos y la envía hacia la base de datos.

Al igual que con el Agente Modbus, se procura definir el inicializar el agente a través de una interfaz gráfica (que será descrita en el apartado 2.4.6).

2.4.5. Driver de comunicación MQTT

Al igual que con los agentes anteriores, se incorpora un agente basado en un driver de comunicaciones MQTT. Este driver permite a un cliente subscribirse a un determinado tópico de un determinado broker (definidos por el operador), identificar la fuente cada uno de los mensajes publicados en dicho tópico para, finalmente, usar dichos mensajes dentro del flujo de eventos del cliente del módulo “historiador-mqtt”.

Para ello se requiere la información de conexión hacia el broker de interés: su dirección ip, y número de puerto lógico. Por otro lado, se requiere el nombre del tópico MQTT sobre el cual se están publicando mensajes y se requiere los nombres de las métricas publicadas en dicho mensaje.

De esta forma el Agente basado en el driver de comunicaciones MQTT puede definirse como la suma de:

- Uno o varios tópicos pertenecientes a uno o varios brokers MQTT.
- Un driver de comunicaciones que apunta al broker MQTT objetivo utilizando los parámetros de conexión necesarios y que toma la información de un tópico para ponerla a disposición para el cliente del módulo “historiador-mqtt”.
- El cliente del módulo “historiador-mqtt” que toma la información proveniente del enlace de comunicaciones y la envía a la base de datos.

Al igual que los casos anteriores, se procura definir el inicializar el agente a través de una interfaz gráfica (que será descrita en el apartado 2.4.6).

2.4.6. Interfaz web para la configuración de comunicaciones

Tal como se mencionó en 2.1.2.1, se propone utilizar una interfaz gráfica que facilite al operador del historiador de procesos la configuración de las comunicaciones que gestionan el tráfico de entrada al aplicativo.

Se propone una interfaz gráfica basada en una aplicación web y será desarrollada utilizando el framework Vue.js.

Con el objeto de facilitar la descripción del aplicativo que representa la interfaz web, se la puede dividir en 2 apartados:

- La parte lógica, desarrollada en el lenguaje JavaScript, contiene un conjunto de instrucciones que tienen como objetivo ejecutar uno o varios de los flujos descritos en 2.4.2, 2.4.3, 2.4.4 y 2.4.5. Es decir, que será la encargada de iniciar la ejecución del cliente del módulo “historiador-mqtt” y del driver de comunicación apuntando a una determinada fuente de información (dispositivo externo) en función de la información ingresada por el operador desde la parte gráfica.
- La parte gráfica, desarrollada en los lenguajes HTML y CSS5, corresponde a un formulario que el operador debe llenar en función del tipo de Agente que desea inicializar.

La información solicitados por los diferentes formularios coincide en los campos:

- Nombre del agente que corresponde al atributo “name” de la tabla “Agents” según lo descrito en el apartado 2.2.2.
- Grupo del agente, que corresponde al atributo “pid” según el apartado 2.2.2.
- Tópico del agente, que corresponde a un campo que nos ayuda a identificar la fuente de información. Para el caso del Agente MQTT este valor es reutilizado para apuntar al Tópico MQTT deseado.
- Tiempo de muestreo, que es el tiempo de retardo entre mensaje y mensaje que se enviará hacia el módulo historiador mqtt.
- Tiempo de recolección: permite alternar entre las opciones
    - “Termina en”, que habilita un campo para ingresar el tiempo en el que se desea terminar la recolección de datos una vez iniciado el Agente.

- “Interminable”, que indica que el tiempo de recolección nunca se detendrá y que habilita dos opciones de guardado
    - “Todos los datos”, que mantiene todos los datos recolectados desde el inicio del Agente hasta el presente.
    - “Datos de los últimos”, que habilita un campo para ingresar un rango de tiempo t con sus unidades de tiempo, que indica que solo se guardarán los datos de las últimas t unidades de tiempo. Esta opción resulta útil para cuidar el uso de memoria de la base de datos.

Y sus demás campos varía en función del tipo de conexión a utilizarse, de esta forma:

- Para configurar un agente tipo Modbus TCP se solicitan los campos:
    - Dirección IP de la fuente de información, es decir del esclavo Modbus.
    - Identificador (Id) del esclavo Modbus.
    - Dirección del Input Register del esclavo Modbus de donde se desea tomar la información para ingresarla al historiador de datos.
    - Nombre de la métrica, corresponde al atributo “type” de los registros generados en la tabla “Métricas” en la base de datos. Tal como se describe en 2.2.2
- Para configurar un agente de consultas SQL se solicita lo siguiente:
    - Dirección IP del host de la base de datos
    - El nombre de la credencial de acceso del usuario
    - La clave de la credencial de acceso del usuario
    - Nombre de la base de datos
    - Nombre de la tabla
    - Nombre de la columna que contiene la información, este caso es de donde el driver tomará un dato y lo enviará al cliente MQTT. Corresponde al atributo “Value”, tal como se lo describe en 2.2.2.
    - Nombre de la columna que corresponde a la llave primaria de la tabla, este campo permite validar constantemente el último registro creado en la tabla.

    - Nombre de la métrica, corresponde al campo “Type” tal como se lo describe en 2.2.2.

- Para configurar un agente de tipo MQTT se solicitan los campos:
    - Dirección IP del broker MQTT.
    - Puerto del broker.
    - Tópico sobre el cual se publica una o varias métricas
    - Nombre de la métrica, que permite al driver MQTT apuntar al valor correspondiente dentro de los mensajes que se han publicado en el Tópico sobre el que se trabaja.

Finalmente, para todos casos, se presenta un campo de selección adicional que lleva el título de “Tipo de Métrica” y permite elegir entre “Analógico” y “Digital”. Lo que hace este campo es concatenar la cadena de caracteres “bool ” al nombre de la métrica siempre y se haya seleccionado la opción “Digital”, esto con el objeto de facilitar a la interfaz de monitoreo la impresión de datos. Ejemplo, si el operador llena el campo “Nombre de la métrica” como: ”pump_in_flow” y selecciona el campo “Tipo de métrica” como: “Digital”, entonces el campo “Nombre de la métrica” será modificado a “bool pump_in_flow”.

En la Figura 2.9, se muestra el esquemático de la interfaz de configuración. De izquierda a derecha se observa:

- El formulario de configuración de un agente que utiliza el driver de consultas SQL
- El formulario de configuración de un agente que utiliza el driver Modbus TCP
- El formulario de configuración de un agente que utiliza el driver MQTT.

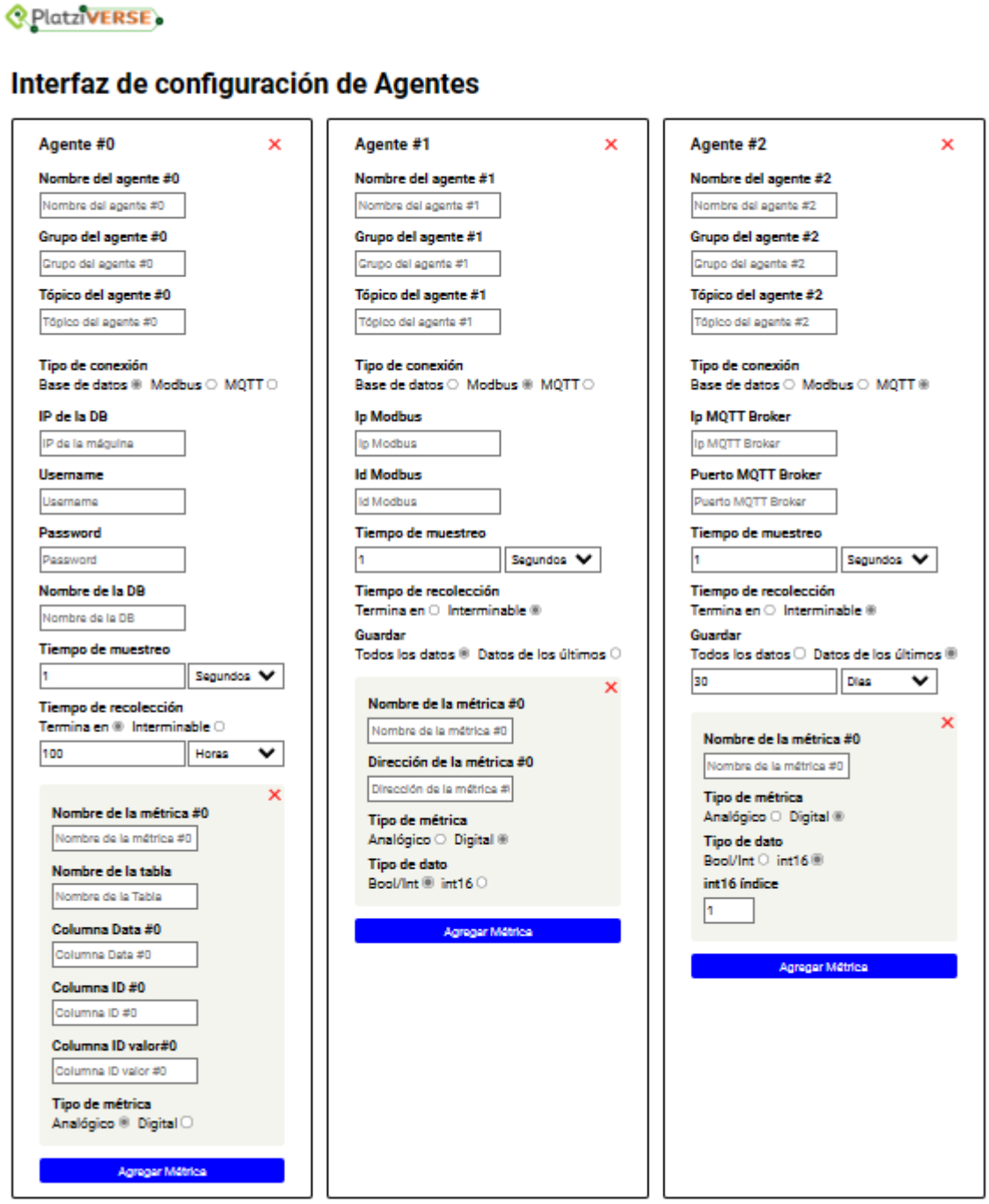


Figura 2.9. Esquematización de los diferentes formularios disponibles en la interfaz de configuración de comunicaciones de entrada.

2.5. Módulo de comunicaciones de salida

En este apartado se describe el módulo de comunicaciones de salida que se encuentra conformado por una única entidad misma que toma el nombre de “historiador-api” y cuyo objetivo es entregar información asociada a los agentes y sus respectivas métricas que ha

sido recolectadas previamente por el módulo de comunicaciones de entrada hacia el historiador de procesos.

El módulo se encarga de recibir diferentes solicitudes de información hacia las tablas “Agents” y “Metrics” de la base de datos del historiador de procesos, provenientes de clientes a través del protocolo HTTP.

En la Figura 2.10 se puede visualizar el apartado que ocupa el presente módulo dentro de la arquitectura de la aplicación.

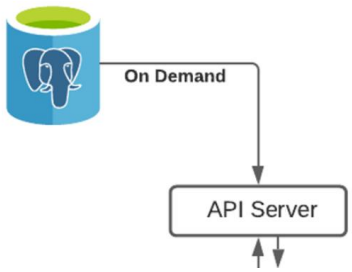


Figura 2.10. Esquematización del módulo de comunicaciones de salida (historiador-api)

2.5.1. Servidor API

El módulo “historiador-api” es un servidor web API (interfaz de programación de aplicaciones) que opera como una biblioteca de subrutinas, funciones y procedimientos que puede ser utilizada por cualquier cliente. El módulo está desarrollado bajo el paradigma API REST, de manera que cualquier cliente puede enviar una consulta o solicitud a través de HTTP utilizando el formato JSON (Javascript Object Notation), el módulo validará la solicitud, ejecutará una determinada acción y retornará una respuesta al cliente en formato JSON [20].

El módulo básicamente es un puente que traduce las peticiones HTTP recibidas a consultas SQL hacia la base de datos del historiador de procesos utilizando el módulo “historiador-db” y su grupo de funciones de lectura y escritura definidas tal como se lo describe en 2.2.4.

Las subrutinas, funciones y procedimientos del servidor se las puede organizar por rutas construidas sobre una URL. Como se observa en la Figura 2.11, una URL puede ser dividida en 3 partes, dos de ellas: el protocolo y el dominio, se consideran como la raíz de la dirección y la parte variable es la ruta de acceso. El cliente debe construir un mensaje JSON y enviarlo al servidor para realizar peticiones sobre las diferentes variantes de la URL con el fin de solicitar información.

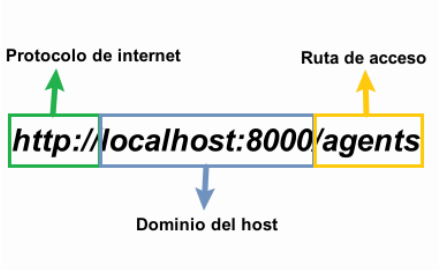


Figura 2.11. Esquema de las partes que conforma una URL definida para el módulo historiador-api

A continuación, se describen las rutas implementadas:

- '/agents': Se trata de una petición de tipo GET que tiene como objetivo retornar los registros de la tabla "Agents" existentes en la base de datos. Dichos registros son filtrados en función de los argumentos de entrada que recibe la petición.

  Los argumentos de entrada corresponden a un usuario y contraseña, mismos que son validados por la lógica para poder identificar al cliente y otorgarle un rol de acceso. Dicho roll de acceso limitará a los agentes sobre las búsquedas permitidas. Existen 3 roles de acceso definidos:

  - Usuario: es rol con menos jerarquía, permite retornar únicamente los agentes que se encuentren como conectados, es decir con el atributo "connected" con valor verdadero en los registros correspondientes de la tabla "Agents"; y que adicionalmente su atributo "username" de la tabla "Agents" coincida con el nombre de usuario ingresado en la petición.
  - Admin: este rol permite retornar todos los agentes que se encuentren con estado conectado independientemente de su valor en el atributo "username".
  - Superadmin: es el rol con mayor jerarquía, permite retornar todos los agentes que se encuentren con estado conectado y adicionalmente todos los agentes que alguna vez se llegaron a conectar y que se encuentren con estado desconectado, es decir con el atributo "connected" con valor falso en los registros correspondientes de la tabla "Agents".

  Si los argumentos de entrada no corresponden a ninguno de las credenciales definidas para los usuarios entonces el sistema arroja un mensaje de error señalando que el acceso es no autorizado.

Esta petición es de importancia puesto que todas las demás peticiones que se describirán a continuación se basan en ella de manera que toda la información que manejan se filtra en función de la lógica de roles ya mencionada.

- '/agent/:uuid': se trata de una petición de tipo GET, recibe como parámetro un identificador único (uuid) que permite filtrar los agentes existentes en la tabla "Agents" en función de su atributo "uuid".
- '/metrics/:uuid': se trata de una petición de tipo GET, que recibe como parámetro un identificador único y permite filtrar las métricas asociadas a un agente cuyo atributo "uuid" coincide con el parámetro de entrada.
- '/metrics/:uuid/:type': se trata de una petición de tipo GET, que recibe como parámetro un identificador único y un "type"; permite filtrar las métricas cuyo atributo "type" coincide con el correspondiente parámetro de entrada ingresado y que además pertenezcan a un agente cuyo atributo "uuid" coincide con el otro parámetro de entrada ingresado.
- '/metrics/date/:uuid/:type': se trata de una petición de tipo GET, que recibe como parámetro un "uuid", un "type" y un campo que contiene un rango de fecha, "date"; permite filtrar las métricas cuyo atributo "type" coincide con el correspondiente parámetro de entrada ingresado y que además pertenezcan a un agente cuyo atributo "uuid" coincide con el correspondiente parámetro de entrada. Adicionalmente se cuenta con un tercer filtro basado en el parámetro de entrada "date" que recibe una fecha de origen y una fecha fin y filtra todas las métricas que se encuentren dentro de dicho rango en función de su atributo "createdAt"

Es importante mencionar que si la URL ingresada para la petición HTTP es incorrecta entonces el servidor dará aviso al cliente retornando dicho error como respuesta a la petición.

De esta forma cualquier cliente que tenga acceso al URL del servidor y que pueda construir las diferentes solicitudes HTTP y ejecutarlas; será capaz de obtener información del historiador de procesos a través del módulo historiador-api.

2.6. Módulo de monitoreo y visualización

En este apartado se describe el módulo que corresponde a la interfaz de monitoreo y visualización del historiador de procesos y que toma el nombre de "historiador-web".

La entidad funciona como un cliente del módulo historiador-api, es decir que solicita información de la base de datos del historiador de procesos a través del módulo historiador-api mediante peticiones HTTP. Una vez obtenida dicha información la organiza y la muestra en una interfaz a través de una gráfica de amplitud en función del tiempo.

La vista que tiene el operador de la interfaz se divide en 2 bloques:

- Del lado izquierdo de la ventana se muestra a los diferentes agentes con sus correspondientes métricas que se encuentran registrados en el historiador de procesos. Cada agente con sus métricas asociadas es agrupado en tarjetas. Cada tarjeta muestra los atributos “uuid” y “conected” del agente, así como los nombres del agente y de sus métricas.

  Cada una de las métricas de un agente pueden ser ocultada o no y, adicionalmente, se cuenta con una herramienta para cambiar el color de línea utilizada en la gráfica amplitud vs tiempo.

Figura 2.12. Grupo de tarjetas que contiene la información de un agente y sus métricas en la interfaz de monitoreo y visualización

- Del lado derecho de la ventana se muestra la gráfica de amplitud en función del tiempo en donde se graficarán todas las métricas seleccionadas de cada una de las tarjetas de Agente.

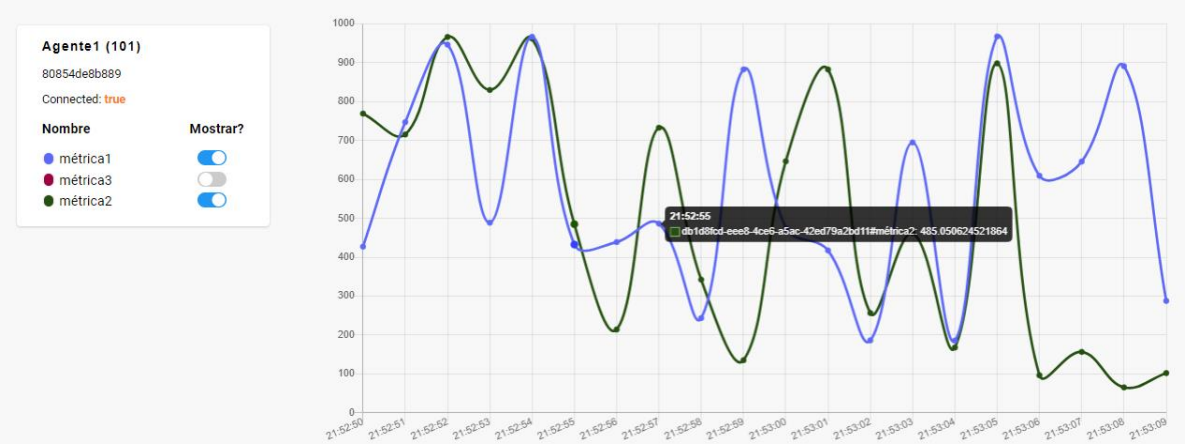


Figura 2.12. Gráfica de amplitud en función del tiempo en la interfaz de monitoreo y visualización.

La interfaz gráfica tiene dos formas de mostrar la información:

- Datos en tiempo real: muestras una gráfica dinámica que se actualiza cada segundo en función de los datos que se están ingresando en ese momento a la base de datos.

  La gráfica permite visualizar los últimos 20 registros ingresados al historiador, mismos que viajan directamente del módulo historiador-mqtt haciendo uso del "Evento de mensaje de tipo métrica publicado" en el broker tal como se describe en 2.4.1.

  El operador puede seleccionar esta vista utilizando el botón "Modo Live".

- Datos históricos: muestra una gráfica estática con la información existente en la base de datos desde una fecha de inicio hasta una fecha fin. Para ello se cuenta con los campos pertinentes para que el operador ingrese el rango de fecha que desea para la toma de información. Una vez definido el rango, el operador puede alternar a esta vista utilizando el botón "Filtrar"

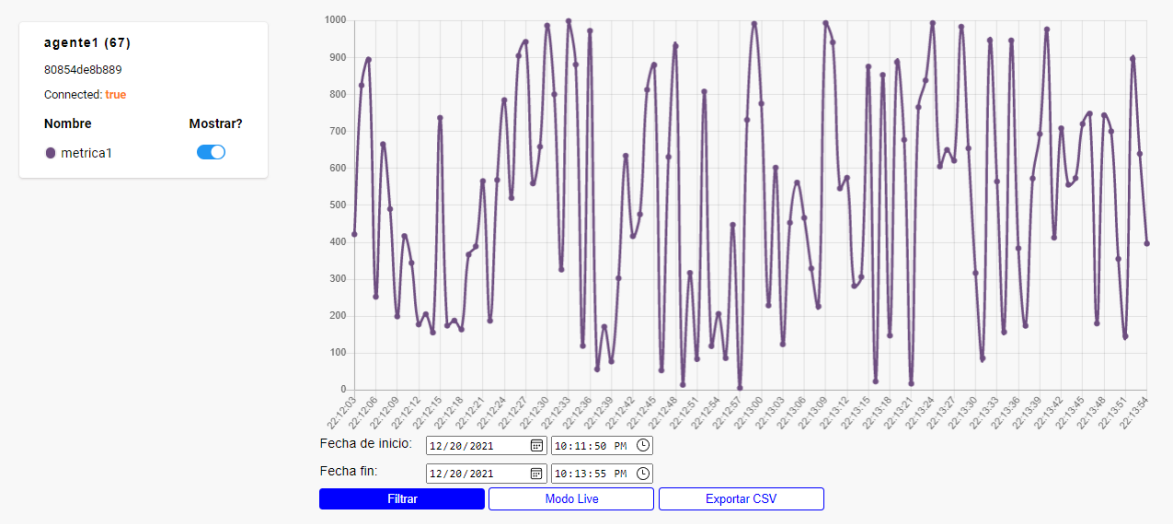


Figura 2.13. Gráfica de amplitud en función del tiempo para el caso de Datos históricos en la interfaz de monitoreo y visualización.

Finalmente se cuenta con un botón “Exportar CSV” que se encarga de crear un archivo .csv [21] para posteriormente descargarlo utilizando el navegador con los datos que se estén mostrando en pantalla en el momento de utilizar la funcionalidad.

2.7. Integración de los módulos que conforman el historiador de procesos

2.7.1. Puertos utilizados para la comunicación entre contenedores

En el presente apartado se define la forma en la que cada uno de los módulos descritos previamente se interrelaciona con los demás con el objeto de formar un todo que representa el historiador de procesos.

Cada uno de los módulos fue creado basado en un servidor cuyos servicios fueron definidos y configurados según su correspondiente framework: express.js para el back-end y Vue.js para el front-end. De esta forma cada uno de los módulos cuenta con uno o más puertos de comunicaciones mismos que operan sobre el protocolo IP por los cuales intercambiarán información ya sea con uno de los módulos que conforman el historiador de procesos o con alguna entidad externa al mismo en función de la lógica descrita previamente en cada uno de sus correspondientes apartados. A continuación, se describen los puertos utilizados por cada módulo:

- Módulo historiador-db: Para este caso se trata de la entidad de la base de datos que utiliza el puerto por defecto de PostgreSQL (5432) para permitir solicitudes de consulta, creación o modificación de los registros existentes en la misma.
- Módulo historiador-mqtt: El servidor MQTT permite que clientes se conecten al broker utilizando el puerto por defecto definido para el protocolo mqtt (1883) con el objeto de realizar consultas al módulo historiador-db a través del puerto 5432.
- Módulo historiador-agent: Tal como se lo expuso en 2.4, un agente puede dividirse en 3 partes de tal forma que:
    - Cliente del módulo “historiador-mqtt”: utiliza el puerto 1883 para comunicarse con el módulo historiador-mqtt haciendo uso del emisor de eventos MQTT.
    - Driver de comunicaciones: utiliza el puerto 502 para comunicarse a través del protocolo Modbus TCP o el puerto 3306 para acceder a una entidad de base de datos externa basada en el motor MySQL.
    - Fuente de información: el driver de comunicaciones utiliza sus puertos definidos para iniciar conexión con algún dispositivo, entidad o aplicación externa a través de Modbus TCP (502), MQTT (1883) o las consultas SQL a un motor de base de datos MySQL (3306).

  Adicionalmente, la interfaz gráfica web de configuración de agentes hace uso del puerto 8000 para mostrar su contenido en el navegador web.
- Módulo historiador-api: Utiliza el puerto 3000 para recibir consultas HTTP y traducirlas a consultas hacia la base de datos utilizando el puerto 5432.
- Módulo historiador-web: hace uso del puerto 8001 para mostrar toda la información que es solicitada al módulo historiador api a través del puerto 3000 en el navegador web.

2.7.2. Creación de imágenes en Docker a partir de los módulos que conforman el historiador de procesos

Docker facilita la creación de contenedores que representan un ambiente virtual de tal forma que, si de una máquina virtual se tratase, cuyas capacidades de procesamiento y comunicaciones pueden ser definidas por el usuario.

Lo que se busca es poder crear un contenedor por cada módulo desarrollado para el historiador de procesos, en donde se ejecuta una imagen previamente construida utilizando un archivo DockerFile, que no es más que un archivo de texto que contienen un conjunto de instrucciones que le indican a Docker como crear una imagen para que pueda ser ejecutada en un contenedor. Revisar DockerFile en el apartado de anexos A3.

Docker pone a disponibilidad una nube de acceso gratuito para publicar y descargar imágenes creadas por los desarrolladores oficiales de una determinada herramienta o por la comunidad. La nube toma el nombre de Docker-Hub y es básicamente una librería de imágenes para la creación de contenedores de Docker. Las imágenes de interés para este proyecto son:

- Ubuntu-20.04, es una imagen oficial creada a partir del sistema operativo Ubuntu en su versión 20.04. Sobre esta imagen se crearán las diferentes imágenes correspondientes a los módulos: historiador-mqtt, historiador-agent, historiador-api e historiador-web.
- PostgresSQL-13, es una imagen creada a partir del sistema operativo Alpine en su versión 14.1 sobre la cual se ejecuta el motor de base de datos PostgreSQL en su versión 14. Sobre esta imagen se construye la imagen correspondiente al módulo historiador-db
- PgAdmin-4, se trata de un gestor visual de bases de datos desarrollado en Python y optimizado para funcionar con PostgreSQL. Esta imagen no corresponde a ninguno de los módulos desarrollados, pero resulta de utilidad en el supuesto caso de requerir modificaciones manuales hacia la base de datos.

La construcción de las imágenes que representan los módulos del historiador de procesos puede ser realizada manualmente siguiendo uno de los flujos detallados a continuación.

Para el caso de los módulos: historiador-mqtt, historiador-agent, historiador-api e historiador-web:

1) Se parte de la imagen oficial de Ubuntu-20.04.
2) Se copia todo el código que corresponde al módulo en cuestión a una ruta de memoria del sistema operativo.
3) Se instalan todas las librerías de dependencia del módulo.
4) Se ejecuta la imagen sobre el contenedor

Para el caso del módulo historiador-db:

1) Se parte de la imagen oficial de PostgreSQL-13
2) Se define las credenciales de acceso de la base datos
3) Se ejecuta la imagen sobre el contenedor

Una vez creadas las imágenes, Docker nos permite cargarlas a la nube Docker-Hub con el objeto de facilitar el acceso al público de las mismas y que cualquier persona pueda descargarlas y usarlas a conveniencia. De esta forma el usuario tendrá dos caminos a tomar para poder ejecutar los diferentes módulos:

- Crear manualmente las imágenes correspondientes a cada módulo siguiendo el flujo mencionado previamente. Para ello el usuario debe disponer del código completo de cada uno de los módulos del historiador de procesos en el ambiente en el que se realizará la construcción de las imágenes disponible en Github: https://github.com/Christopher-Castro/Historiador.

  Esta opción resulta útil siempre y cuando el usuario haya modificado directamente el código desarrollado para este proyecto.

- Descargar las imágenes ya creadas a la par del desarrollo del presente proyecto mismas que se encuentran disponibles en https://hub.docker.com/u/christopher57

  Esta opción resulta útil si el usuario desea ejecutar el historiador de procesos sin aplicar ninguna modificación a la lógica definida en el presente proyecto.

2.7.3. Ejecución de las imágenes creadas en contenedores de Docker

Con el objeto de facilitar el despliegue de cada una de las imágenes creadas en su correspondiente contenedor se utiliza la herramienta Docker-compose, que permite la creación y definición de: uno o varios contenedores, el espacio de memoria físico que ocuparán los contenedores virtuales y una red de comunicaciones de tipo IP por la que los contenedores se comunican haciendo uso de sus puertos tal como se definió en 2.7.1. Revisar el archivo Docker-compose en el apartado de anexos A3.

Docker-compose se ayuda únicamente de un archivo de texto con extensión .yml, en donde se describe:

- El nombre del contenedor a crearse.
- El nombre de la imagen previamente construida o, en su defecto, el espacio de memoria donde se encuentra el DockerFile para la creación de una imagen.

En este apartado se puede agregar la información del repositorio de Docker-Hub con el fin de utilizar la imagen previamente construida y cargada a la nube.

- Los puertos que el contenedor utilizará para comunicarse con otros contenedores o para comunicarse con usuarios externos.
- Las variables de entorno a utilizarse, este campo resulta útil para la configuración del contenedor de base de datos. En este apartado se definen las credenciales de usuario que un cliente deberá utilizar para autenticar su conexión a la base de datos.

3. RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1. Resultados

Las pruebas estarán divididas en función del tipo de agente que se utiliza para la generación de métricas.

3.1.1. Agente MQTT

En este caso se procura la conexión de una aplicación en JavaScript externa al historiador de procesos que es capaz de invocar al módulo del cliente-mqtt para hacer uso de su set de eventos y así crear registros en la base de datos del historiador de procesos.

Como se lo describió en 2.4.5, en el módulo historiador-agent cuenta con la capacidad de conectarse a uno o varios brokers para suscribirse en un determinado tópico y de ahí obtener las métricas de interés.

A continuación, en la Figura 3.1, se muestra la configuración del agente desde la interfaz web. Se lo nombra como “AgenteMQTT” y se lo destina al grupo “Grupo1” y se apunta al “Topico1”. Adicionalmente se agrega la configuración de conexión al broker y se define también el tiempo de muestreo de las métricas en 1 segundo, así como el tiempo de recolección del agente que termina en 100 horas. El agente publicará dos métricas una de tipo digital bajo el nombre “metrica1” y otra de tipo analógica bajo el nombre “metrica2”.

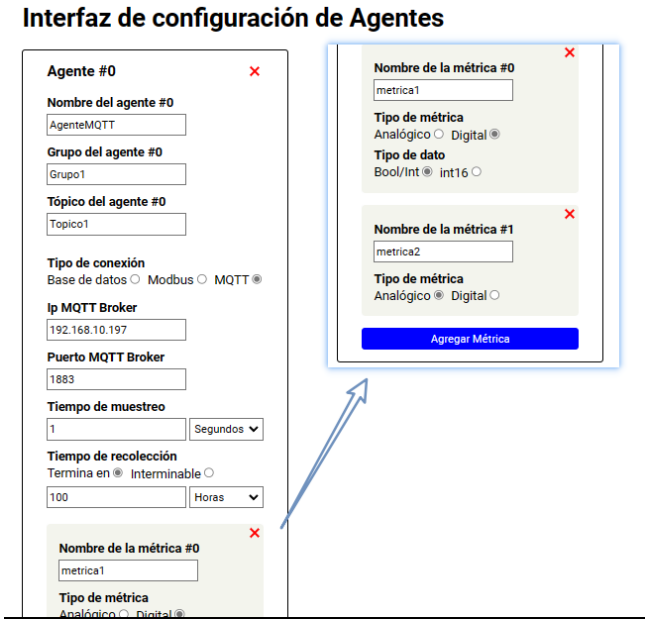


Figura 3.1. Configuración de un agente MQTT desde la interfaz de comunicaciones de entrada.

Una vez generado el agente nos dirigimos a la interfaz de monitoreo y visualización, en donde se puede acceder a los registros en tiempo real del agente creado, así como sus registros históricos. En color azul se visualiza la métrica “metrica1” que es de tipo analógico. De color violeta se visualiza la métrica “metrica2” que es de tipo digital.

A continuación, se puede visualizar la interfaz de monitoreo para el caso de un agente MQTT conectado al historiador de procesos. En la Figura 3.2, se visualiza la información en tiempo real de ambas métricas correspondientes a “AgenteMQTT”. Por otro lado, en la Figura 3.3, se visualiza los históricos de la métrica analógica dentro de un rango de tiempo.

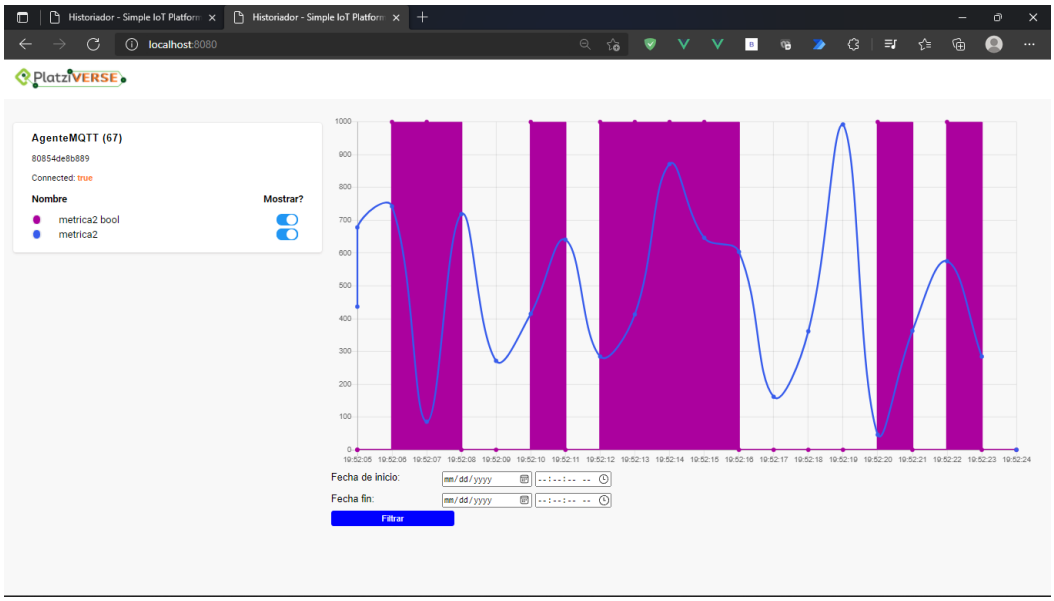


Figura 3.2. Interfaz de monitoreo para el caso de un agente MQTT.

Figura 3.3. Interfaz de monitoreo para el caso de un agente MQTT.

3.1.2. Agente de consultas SQL hacia una base de datos MySQL

Para esta prueba se crea una base de datos MySQL externa al historiador de procesos.

Se propone el caso de una aplicación externa que toma datos del funcionamiento de una bomba centrífuga y los almacena en una base de datos MySQL. La base de datos lleva el nombre “db_test” y cuenta con una tabla nombrada como “pump” misma que tiene 3 columnas:

- Id: que es la llave primaria que permite identificar el número de registros que contiene la tabla. En otras palabras nos permite fijar una fila determinada.
- Power: esta columna contiene el estado de encendido o apagado de la bomba centrífuga
- Rpms: esta columna contiene el registro de las revoluciones por minuto a las que opera la bomba centrífuga

id	Power	Rpms
1	0	0
2	0	0
3	0	0

Figura 3.4. Representación gráfica de la tabla en la base de datos. Inicialmente tiene 3 registros.

En función de esta información se realiza la configuración del correspondiente agente desde la interfaz de configuración de comunicaciones de entrada.

Se define el nombre y el grupo del agente como “AgenteSQL” y “Grupo2” respectivamente. A continuación, se configuran los parámetros de conexión con el host cada una de las métricas del agente. El tiempo de muestreo se lo establece en 1 segundo y el tiempo total de recolección de datos se lo setea en “Interminable”. Finalmente, para cada métrica se requiere el nombre de la tabla, el nombre de la columna que contiene el dato, el nombre de la columna que nos permite seleccionar una fila (Id) y, finalmente, el valor que debe tener la fila (Id) deseado. En este caso se configura una variable digital a ser tomada se la columna “Power” con Id 1 y por otro lado se configura una variable analógica a ser tomada se la columna “Rpms” con Id 3.

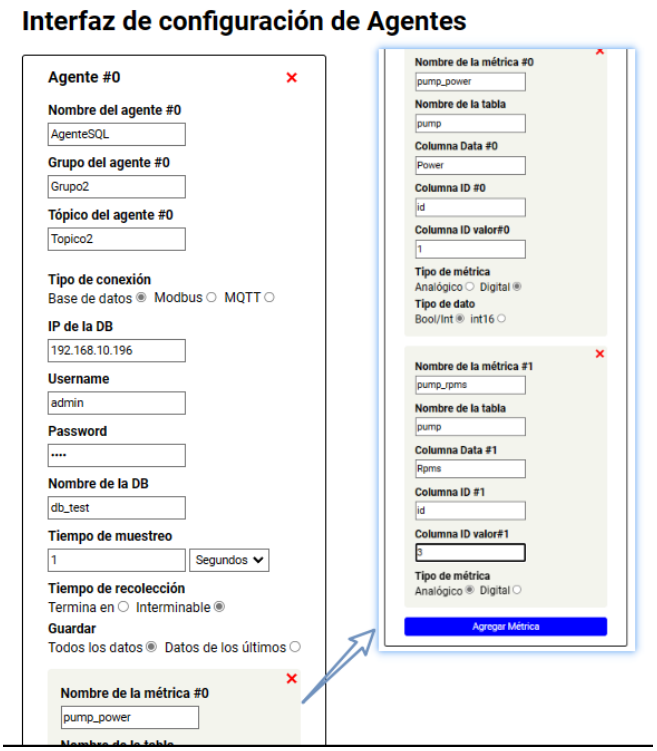


Figura 3.5. Interfaz de configuración para el caso de un agente SQL que apunta a las filas con id 1 y 3 además de apuntar a 2 columnas “Power” y “Rpms” de una tabla en una base de datos MySQL externa.

Una vez configurado el agente, se puede visualizar la información en la interfaz de monitoreo. De esta forma se visualiza la gráfica realizada a partir del último registro ingresado en la tabla, es decir, cada 1 segundo durante un tiempo de recolección interminable, el agente lee el último valor de una determinada celda de la tabla “pump”.

A continuación, se muestra la gráfica resultante para el caso en el que la tabla tiene los registros mostrados en la Figura 3.4. Como e observa cada 1 segundo se lee el último registro de las celdas a las que se apunta, en este caso la celda de la columna “Rpms” con Id 3 corresponde a un valor 0 y un valor falso para la celda “Power” con Id 1.

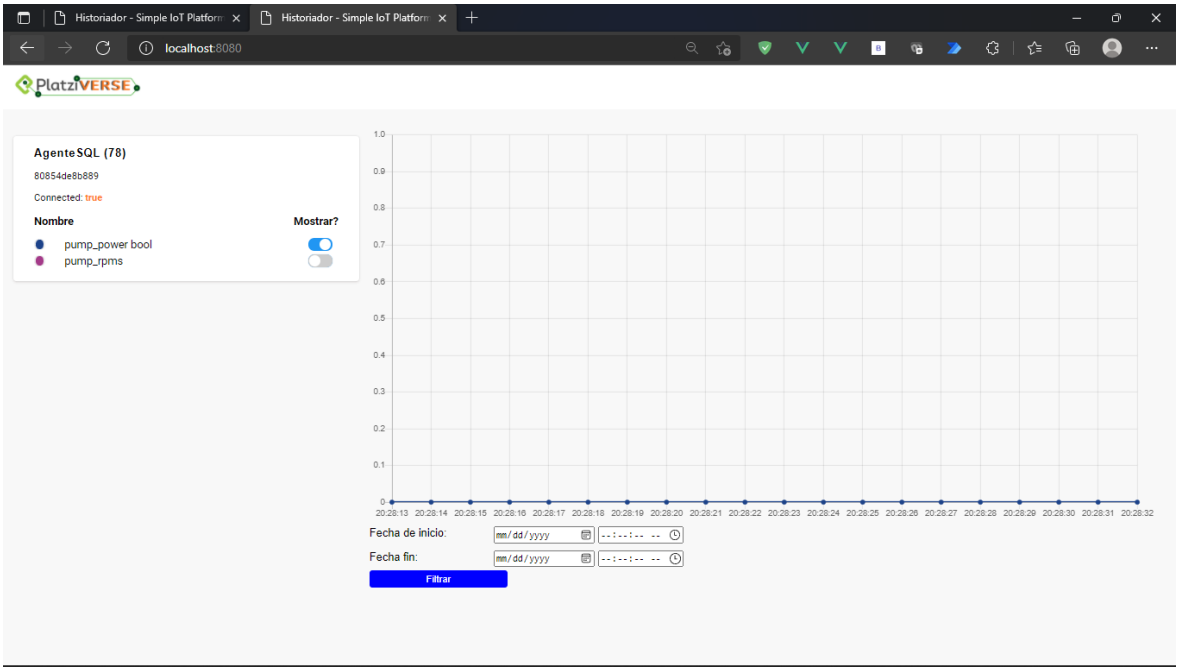


Figura 3.6. Interfaz de monitoreo para el caso de un agente SQL que apunta a 2 columnas de una tabla en una base de datos MySQL externa.

A medida que se generan registros en la tabla de la base de datos se van modificando, las métricas correspondientes mismas que pueden ser visualizadas en la interfaz de monitoreo del historiador de procesos. En la Figura 3.7 se puede visualizar la progresión de cambios sobre la tabla de la base de datos externa resultante tras a lo largo del tiempo, mientras que en la Figura 3.8 se visualiza su representación en el historiador de procesos.

id	Power	Rpms
1	0	0
2	0	0
3	0	0

id	Power	Rpms
1	1	0
2	0	0
3	0	10

id	Power	Rpms
1	1	0
2	0	0
3	0	100

id	Power	Rpms
1	1	0
2	0	0
3	0	1000

id	Power	Rpms
1	1	0
2	0	0
3	0	2500

id	Power	Rpms
1	1	0
2	0	0
3	0	100

id	Power	Rpms
1	1	0
2	0	0
3	0	50

id	Power	Rpms
1	0	0
2	0	0
3	0	0

Figura 3.7. Representación gráfica de la progresión de cambios sobre las celdas de la tabla tabla en la base de datos sobre la que se ha configurado el Agente.

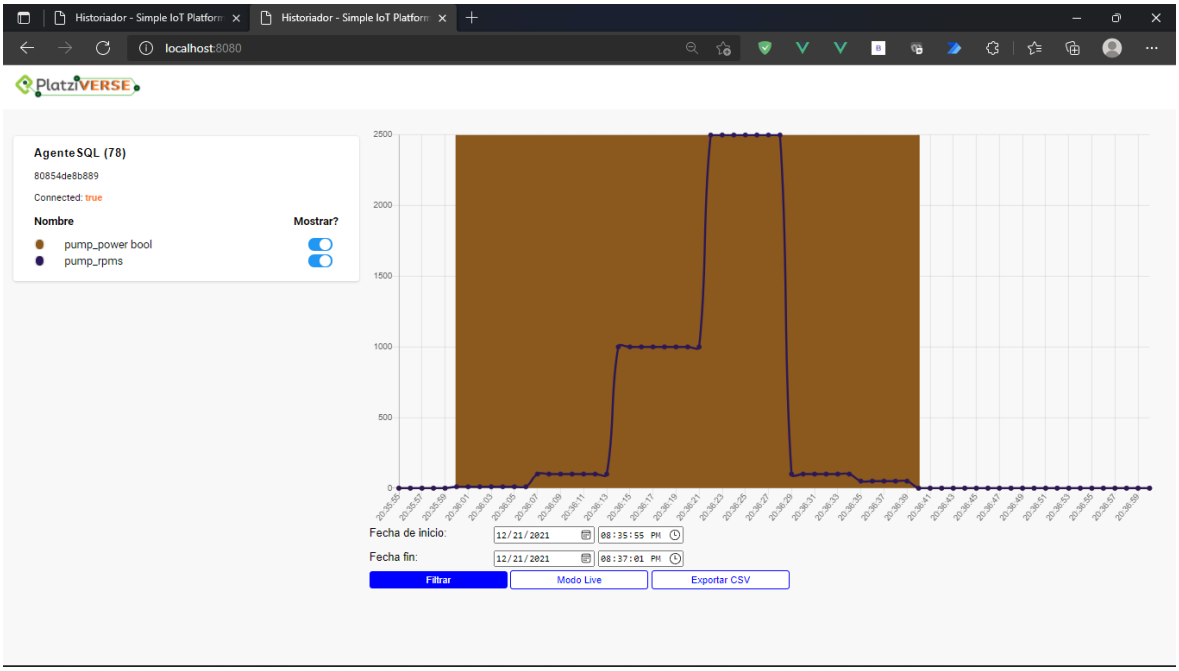


Figura 3.8. Interfaz de monitoreo para el caso de un agente SQL que apunta a 2 celdas de una tabla en una base de datos MySQL externa.

3.1.3. Agente Modbus

Para la prueba de comunicaciones de entrada basadas en un agente Modbus se propone el siguiente subproceso:

Dentro de una planta de pasteurización, existe un subproceso denominado “recepción y almacenamiento”, como su nombre lo indica, se trata de una etapa en la que se recibe el producto por una tubería pasando por una válvula de ingreso y se lo almacena en un tanque. El producto lácteo puede ser drenado del tanque mediante una tubería a través de una válvula de salida. Existen 3 dispositivos lógicos recolectando datos de manera que: Uno de ellos monitorea el nivel, la temperatura y la presión interna del tanque; el otro dispositivo se encarga de monitorear el estado de las válvulas de ingreso y de salida del tanque; finalmente un dispositivo se encuentra monitoreando el máximo y mínimo nivel de seguridad con la ayuda de dos detectores de nivel. Cada dispositivo está configurado como esclavo Modbus TCP y ponen a disposición de un dispositivo maestro sus datos a través de Input Registers.

Debido a que propone utilizar 3 dispositivos Modbus TCP, la prueba se la realiza con la ayuda de un simulador de esclavo Modbus. La herramienta se denomina “Modbus Slave” [22] y es el simulador Modbus más popular para ordenadores Windows. De esta forma, se ejecutarán 3 instancias Modbus configuradas como esclavos base 1 de 5 dígitos, cada una con un endpoint diferente acorde a lo mostrado en la tabla 3.1:

Tabla 3.1. Mapa de memoria de las instancias Modbus emuladas

Dispositivo emulado	End Point	Id Modbus	Nombre de variables	Dirección Modbus
1	192.168.100.110	1	tank_lvl	30001
			tank_temp	30002
			tank_preasure	30003
2	192.168.100.89	2	valve_in	10001
			valve_out	10002
3	192.168.100.90	3	tank_LLS_alarm	10001
			tank_HLS_alarm	10002

Todas las instancias operan en el puerto por defecto 502.

En la Figura 3.9 se muestra la configuración de uno de los 3 emuladores Modbus TCP utilizando la herramienta "Modbus Slave" para Windows. En la ventana nombrada "Slave Definition" se puede visualizar la configuración del esclavo Modbus: su Identificador, el código de función a utilizar, las direcciones en función de la base seleccionada. Del lado izquierdo se puede visualizar el mapa de memoria mediante una tabla cuya primera columna corresponde a un nombre que facilita la identificación del registro de memoria y la segunda columna corresponde al valor que maneja el respectivo registro Modbus.

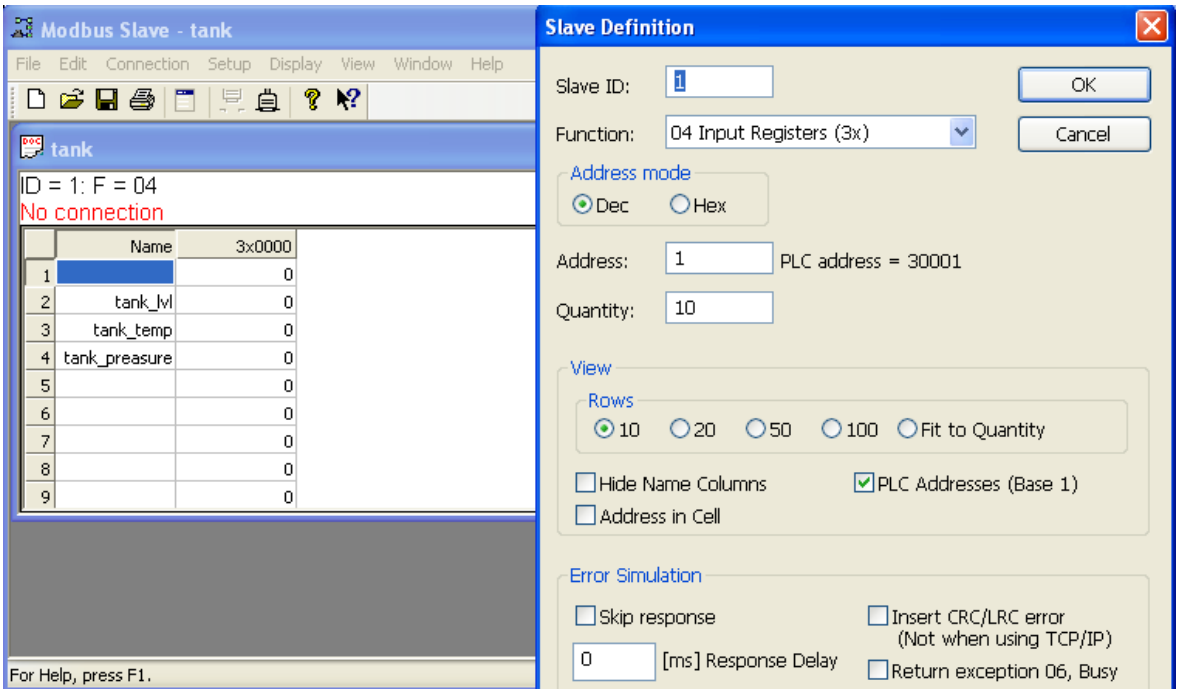


Figura 3.9. Configuración del Agente Modbus1 emulado con la aplicación "Modbus Slave" para windows.

De esta forma se puede iniciar con la configuración de las comunicaciones de entrada al historiador de procesos. Se parte realizando la configuración del conector a través de la interfaz de configuración en donde se generan 3 agentes, cada agente corresponde a cada uno de los dispositivos Modbus. Se definen los nombres, grupos, tiempo de muestreo y tiempo total para cada agente. Finalmente, se completan los parámetros de la conexión Modbus con la información de la Tabla 3.10

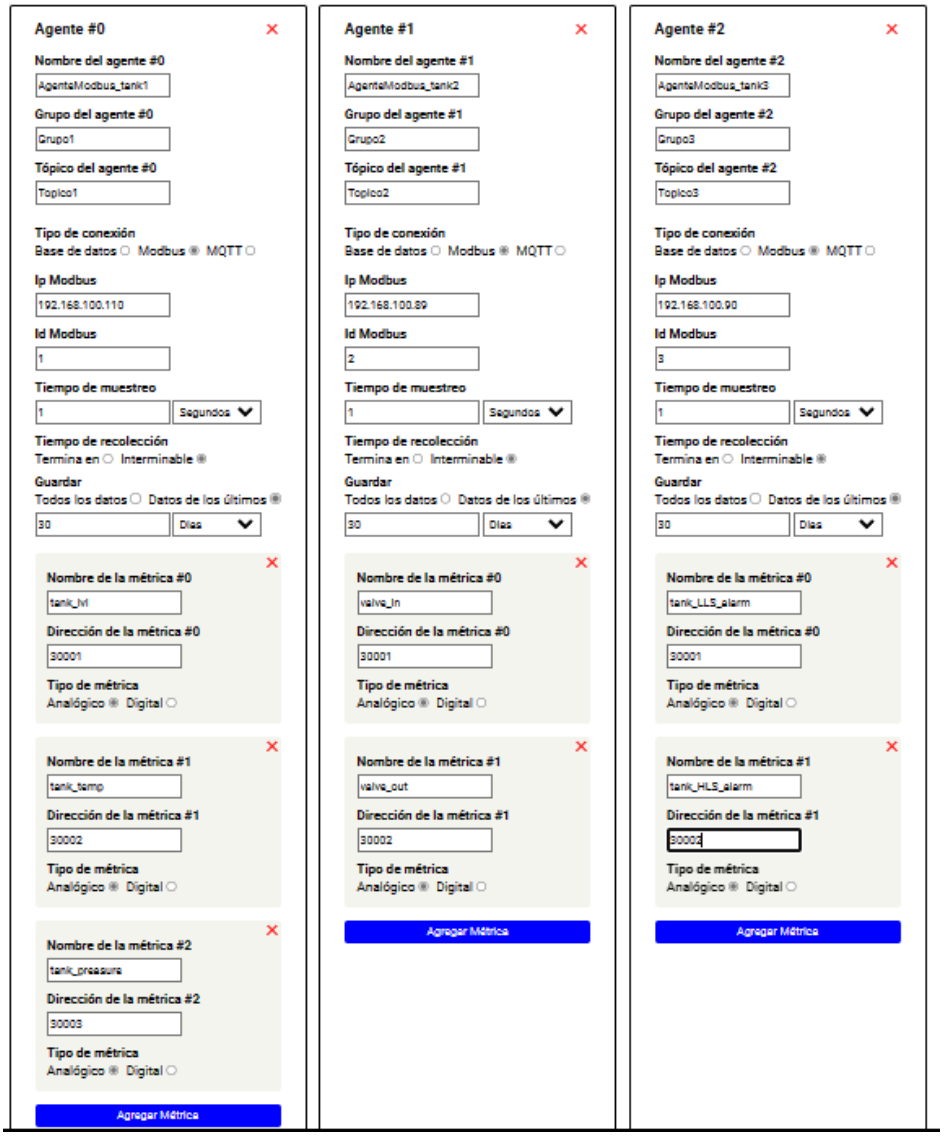


Figura 3.10. Interfaz de configuración para el caso de 3 agentes Modbus.

Adicionalmente, se muestran los pasos realizados para cada una de las métricas en cuestión. Se realizan 10 pasos que representan diferentes instantes de tiempo en los cuales cada uno de los dispositivos Modbus emulados escriben información en los diferentes registros utilizados para la comunicación, tal como se muestra en la Tabla 3.2.

Tabla 3.2. Valores enviados por los esclavos Modbus emulados en la prueba propuesta.

Nombre de variables	Dirección Modbus	p0	p1	p2	p3	p4	p5	p6	p7	p8	p9
tank_lvl	30001	0	20	30	40	50	60	70	80	70	60
tank_temp	30002	0	3	6	7	8	9	11	10	11	10
tank_preasure	30003	0	3	4	3	4	3	3	3	3	3
valve_in	10001	0	1	1	1	1	1	1	0	0	0
valve_out	10002	0	0	0	0	0	0	0	0	1	1
tank_LLS_alarm	10001	0	1	1	1	1	1	1	1	1	1
tank_HLS_alarm	10002	0	0	0	0	0	0	0	1	0	0

En la interfaz de monitoreo se puede visualizar el flujo de datos que las 3 instancias ingresan al historiador de procesos, como se muestra en las Figuras 3.11 a 3.14. Con el objeto de validar las métricas de cada uno de los 3 agentes generados se procura visualizarlos en grupos.

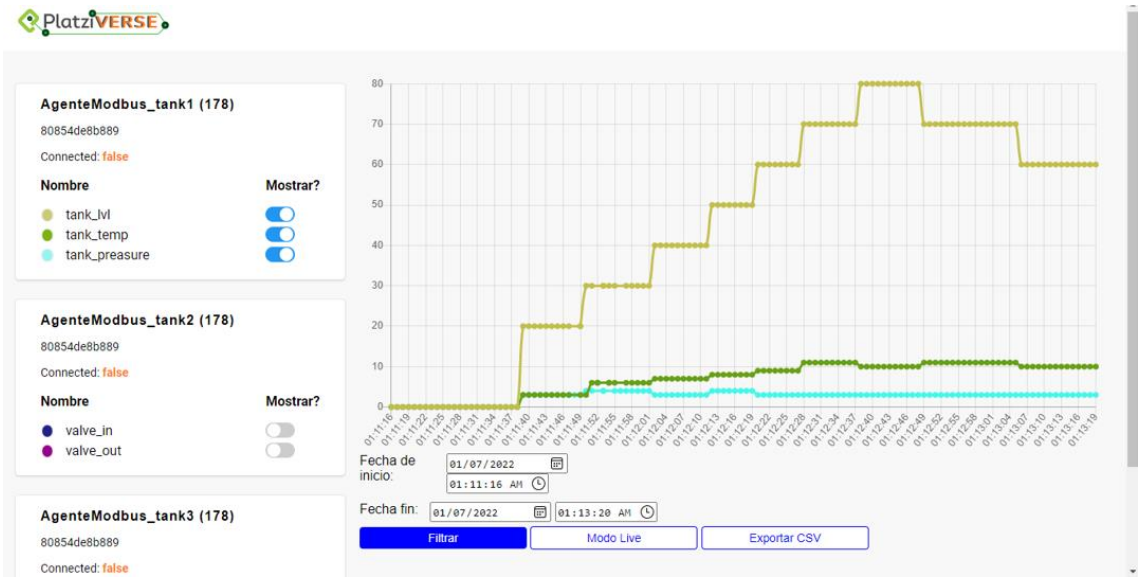


Figura 3.11. Históricos de las métricas: “tank_lvl”, “tank_temp”, “tank_preasure” del Agente Modbus 1.

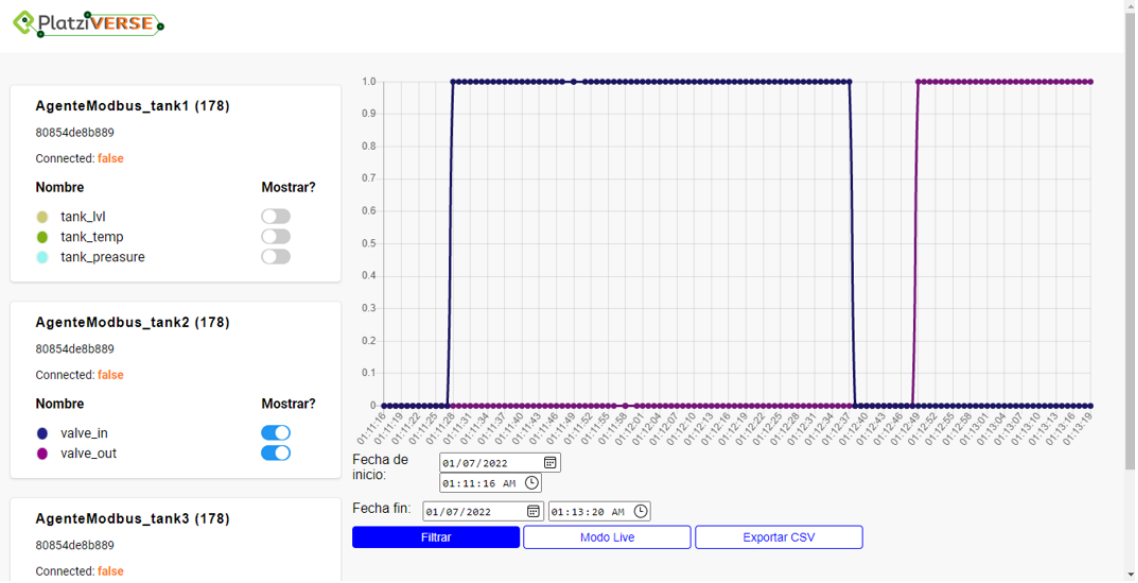


Figura 3.12. Históricos de las métricas: “valve_in”, “valve_out” del Agente Modbus 2.

Figura 3.13. Históricos de las métricas: “tank_LLS_alarm”, “tank_HLS_alarm” del Agente Modbus 3.

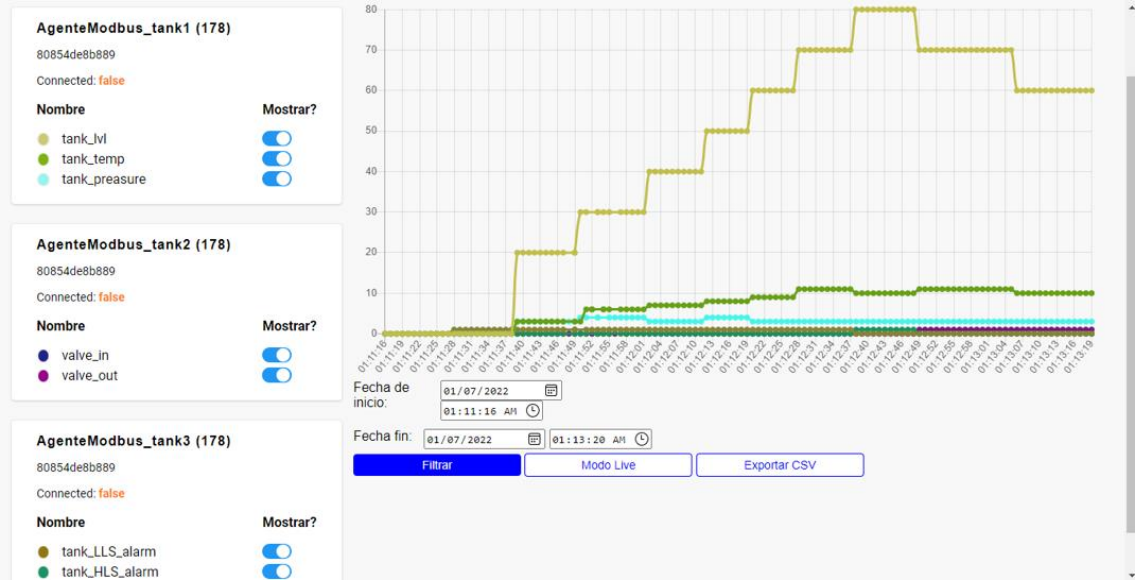


Figura 3.14. Históricos de las métricas de los 3 Agentes Modbus correspondientes.

Finalmente, con los registros históricos seleccionados se procede a exportar la información correspondiente mediante un archivo CSV, Tal como se muestra en las Figuras 3.15 y 3.16.

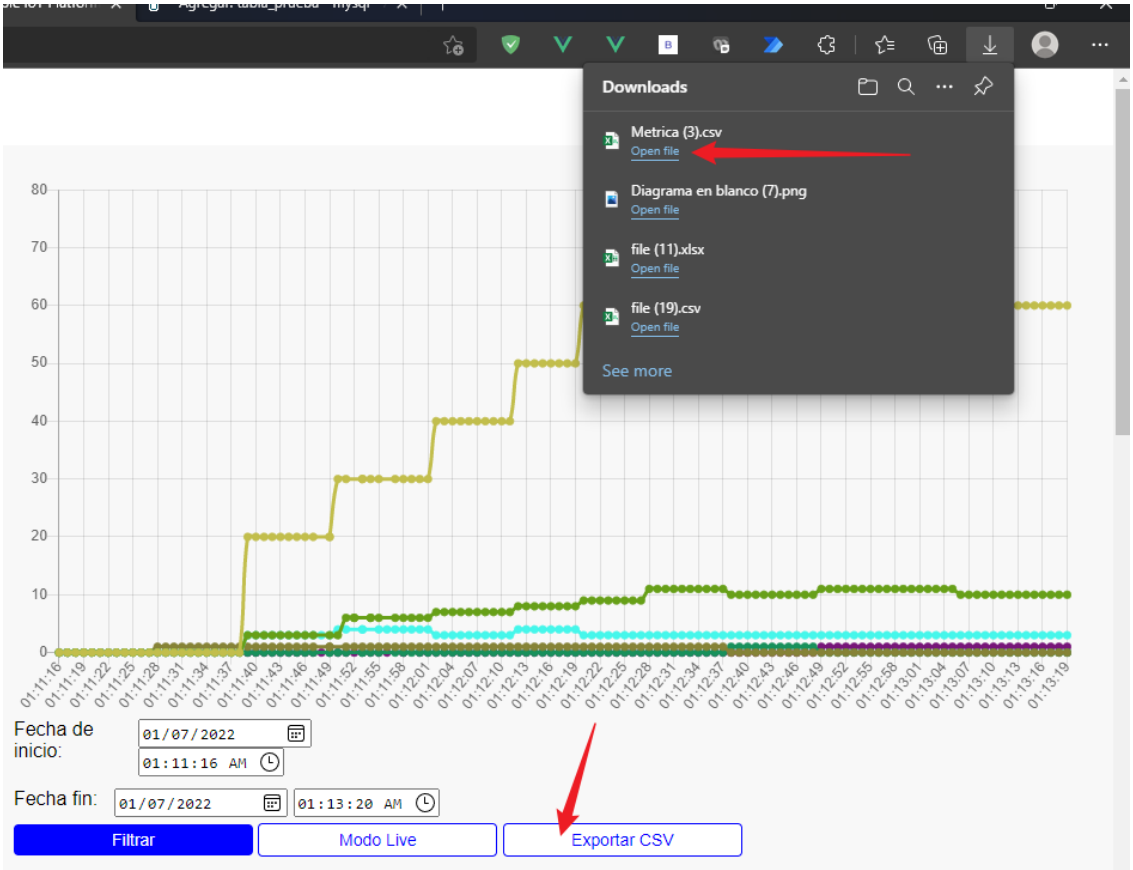
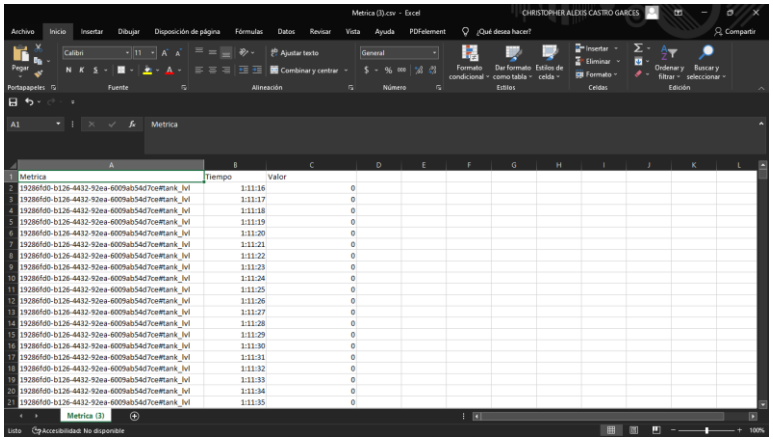


Figura 3.15. Exportación de datos históricos mediante un archivo .csv



Metrica	Tiempo	Valor
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:16	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:17	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:18	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:19	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:20	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:21	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:22	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:23	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:24	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:25	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:26	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:27	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:28	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:29	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:30	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:31	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:32	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:33	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:34	0
19286fd0-b126-4432-92ea-6009ab54d7ce#tank_lvl	1:11:35	0

Figura 3.16. Archivo .csv exportado que contiene la información de la consulta realizada en la interfaz de monitoreo de las métricas de los 3 agentes generados para la presente prueba

3.2. Conclusiones

En función de los resultados obtenidos en el apartado 3.1 se determinan las siguientes conclusiones:

- Se desarrolló un historiador de procesos industriales basado en código abierto capaz de comunicarse utilizando los protocolos MQTT, HTTP, Modbus TCP y consultas SQL y presentar los resultados del intercambio de datos de manera gráfica.

- Las capacidades de comunicación simultanea con múltiples fuentes de información, la incorporación de protocolos de comunicación, el control sobre flujo de datos de entrada, el manejo de información basado en marcas de tiempo, la capacidad de monitoreo de datos históricos y en tiempo real y la posibilidad de realizar configuraciones y control sobre el comportamiento del aplicativo de manera gráfica, son las características más representativas que permiten definir a la aplicación desarrollada como un Historiador de Procesos.

- Para el desarrollo del Historiador de Procesos, particularmente para el desarrollo de sus apartados visuales, JavaScript resulta ser una buena alternativa de código abierto puesto que proporciona varias herramientas que facilitan la maquetación, construcción y despliegue de aplicaciones web multiplataforma, así como el monitoreo y corrección de errores. Para el apartado lógico y de procesamiento de datos, JavaScript responde de manera adecuada, sin embargo presenta ciertas dificultades al trabajar con grandes cantidades de datos por lo que se requiere

aplicar ciertas estrategias de optimización como por ejemplo solo considerar con grupos de datos que se sabe que serán mostrados en el apartado visual.

- Las interfaces visuales desarrolladas incrementan la operabilidad del aplicativo. La interfaz de configuración facilita la creación de Agentes y Métricas en función del tipo de protocolo de comunicación a utilizar y la interfaz de monitoreo permite acceder a los registros de la base de datos del historiador de procesos para mostrar la información que se maneja en tiempo real, así como de los registros históricos existentes.

- Al abrir un canal de flujo de entrada de datos al Historiador de Procesos, es importante tener el control de quién, cuándo y cómo puede hacer uso de dicho canal. En este caso el módulo “historiador-mqtt” es el encargado de validar la adecuada ejecución de los eventos definidos para validar la conexión/desconexión de clientes, el tiempo de muestreo de datos, el tiempo de recolección de datos, el tipo de dato y la información pertinente del cliente que solicita el registro de datos en el Historiador de Procesos.

- Al manejar grandes cantidades de datos enviados por múltiples clientes, es importante definir una lógica de recepción de datos de tal forma que se consiga una operación optima y con el mínimo de retardos. En el caso del presente proyecto se trabajó en una lógica basada en eventos con un esquema de datos definido de manera que los datos de un cliente no interfieren con los de los demás. De esta forma, se garantizan tiempos de retardo mínimos, consiguiendo una operación asíncrona,

- Una correcta organización y administración de los puertos lógicos disponibles de un servidor es crucial para evitar problemas de conexión entre instancias que operan sobre protocolos basados en internet. El uso de redes y subredes facilita en gran medida esta tarea.

- Al realizar las pruebas descritas en el apartado de 3.1 se consiguió verificar el funcionamiento del historiador de procesos de manera que se puede afirmar que el resultado del desarrollo del presente proyecto es una aplicación web de libre acceso y multiplataforma que hace las veces de un historiador de procesos industriales accesible para las PYMES.

- La característica más importante de la aplicación desarrollada es su modularidad. Puesto que facilita la modificación de su lógica de funcionamiento dando paso a

que diferentes fuentes puedan aportar el código fuente del mismo con el objeto de expandir sus capacidades de funcionamiento y/o acoplamiento con otros sistemas.

3.3. Recomendaciones

- Se recomienda expandir las capacidades de comunicación de entrada del aplicativo con el objeto de incrementar la accesibilidad de la aplicación, esto debido a que existen otros protocolos de comunicación que son igual de utilizados que Modbus TCP y las consultas SQL, como, por ejemplo, las comunicaciones basadas en comunicación serial, Profinet, Profibus e Industrial Ethernet.

- Si se desea hacer uso del historiador de procesos desde otra aplicación desarrollada en JavaScript, se recomienda hacer uso de las librerías de funciones y eventos creadas con el objeto de seguir al pie de la letra los diferentes flujos necesarios para la invocación, instanciación y ejecución de las capacidades que pone a disposición el historiador de procesos.

- Si bien cada uno de los módulos puede ser ejecutado directamente en el sistema operativo de un ordenador de escritorio, se recomienda realizar su ejecución a través de los contenedores de Docker definidos, puesto que facilita la configuración del espacio de memoria asignado, así como la distribución de puertos que se utilizan para su funcionamiento.

4. REFERENCIAS BIBLIOGRÁFICAS

[1] I. Y. a. H. Eren, «Data historian,» 2012. [En línea]. Available: https://www.researchgate.net/profile/Halit-Eren-2/publication/294885293_Data_Historian/links/5c1a075ba6fdccfc7058bae6/Data-Historian.pdf. [Último acceso: 28 4 2021].

[2] A. D. a. W. Salter, «cern,» 4 28 2021. [En línea]. Available: https://cds.cern.ch/record/532624/files/mc1i01.pdf.

[3] S. E. d. Normalización, Servicio Ecuatoriano de Normalización, [En línea]. Available: https://www.normalizacion.gob.ec/mipymes-y-organizaciones-de-economia-popular-y-solidaria-son-una-pieza-clave-para-la-economia-del-pais/.

[4] P. G. Smith, «Professional website performance: optimizing the front-end and back-end,» John Wiley & Sons, 2012. [En línea].

[5] S. E. E. Marketplace, «Citect SCADA to Wonderware Historian Translator by Schneider Electric | Schneider Electric Exchange,» [En línea]. [Último acceso: 22 Diciembre 2021].

[6] R. Automation, «Software Operativo Historian,» [En línea]. Available: www.rockwellautomation.com/es-es/products/software/factorytalk/operationsuite/historian.html. [Último acceso: 15 Diciembre 2021].

[7] Mall.industry.siemens.com, «SIMATIC Process Historian,» [En línea]. Available: mall.industry.siemens.com/mall/es/WW/Catalog/Products/10205212. [Último acceso: 22 Diciembre 2021].

[8] M. Jazayeri, «Some Trends in Web Application Development,» Mayo 2007. [En línea]. Available: ieeexplore.ieee.org/document/4221621, 10.1109/fose.2007.26.. [Último acceso: 22 Diciembre 2021].

[9] J. E. PÉREZ, «Introduccion a JavaScript,» 2019. [En línea]. [Último acceso: 22 Diciembtre 2021].

[10] T. Navarrete, «El lenguaje JavaScript,» [En línea].

[11] Node.js, «Acerca de,» [En línea]. Available: nodejs.org/es/about/. [Último acceso: 20 Diciembre 2021].

[12] Node.js, «Express - Infraestructura de Aplicaciones Web Node.js,» [En línea]. Available: expressjs.com/es/. [Último acceso: 22 Diciembre 2021].

[13] Vue.js, «About,» [En línea]. Available: es.vuejs.org/.

[14] R. G. J. &. G. J. Ramakrishnan, «Database management systems (Vol. 3),» [En línea]. [Último acceso: 22 Diciembre 2021].

[15] R. A. Light, «Mosquitto: server and client implementation of the MQTT protocol,» *Journal of Open Source Software,* vol. 2, nº 13, p. 265, 2017.

[16] G. Thomas, « Introduction to Modbus serial and Modbus tcp,» *The extension: A technical Supplement to Control Network,* nº 5, p. 9, 2008.

[17] P. S. B. &. N. A. Ziemniak, «Object oriented application cooperation methods with relational database (ORM) based on J2EE Technology,» *International Conference-The Experience of Designing and Applications of CAD Systems in Microelectronics,* nº 327, p. 330, 2007.

[18] npm, «mosca,» [En línea]. Available: https://www.npmjs.com/package/mosca. [Último acceso: 22 Diciembre 2021].

[19] npm, «Modbus-serial,» [En línea]. Available: https://www.npmjs.com/package/Modbus-serial. [Último acceso: 22 Diciemnre 2021].

[20] A. B. K. K. S. M. W. P. M. R. G. R. .. &. F. P. Yates, «The Ensembl REST API: Ensembl data for any language. Bioinformatics,» vol. 1, nº 31, pp. 143-145, 2015.

[21] J. N. S. U. J. &. P. A. Mitlöhner, «Characteristics of open data CSV files.,» *International Conference on Open and Big Data (OBD) ,* pp. 72-79, 2016.

[22] www.Modbustools.com, «Modbus Slave Simulator,» [En línea]. Available: https://www.Modbustools.com/Modbus_slave.html. [Último acceso: 22 Diciembre 2021].

[23] L. Carvajal, Metodología de la Investgación Científica. Curso general y aplicado, 28 ed., Santiago de Cali: U.S.C., 2006, p. 139.

5. ANEXOS

ANEXO A: MANUAL DE USUARIO

ANEXO A

A.1. Introducción

El software desarrollado corresponde a una aplicación multiplataforma que se puede ejecutar en cualquier sistema operativo donde Docker y Docker-compose hayan sido instalado previamente.

La aplicación corresponde a un historiador de procesos industriales que puede comunicarse utilizando los protocolos Modbus TCP y las consultas SQL a bases de datos externas. Cuenta con una interfaz gráfica que permite la configuración de las comunicaciones y otra que permite la visualización y monitoreo de la información ingresada al historiador.

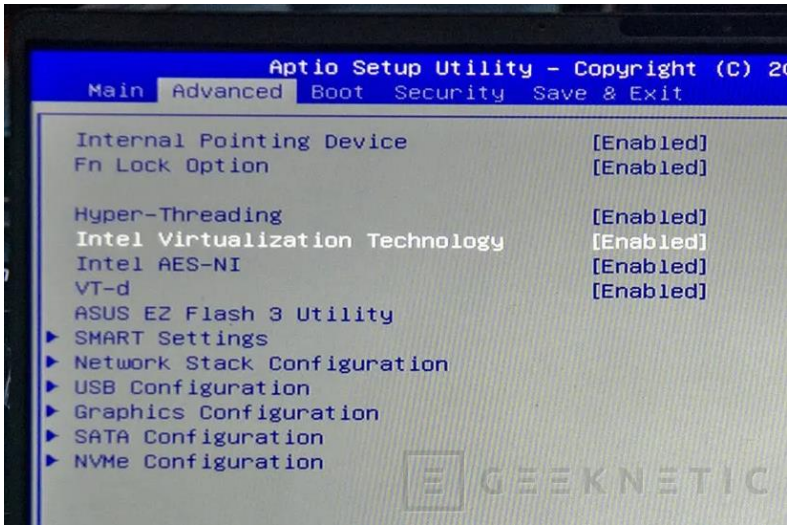
En las siguientes secciones del manual se describen los pasos a seguir para la instalación, el despliegue y operación de la aplicación.

A.2. Requisitos previos

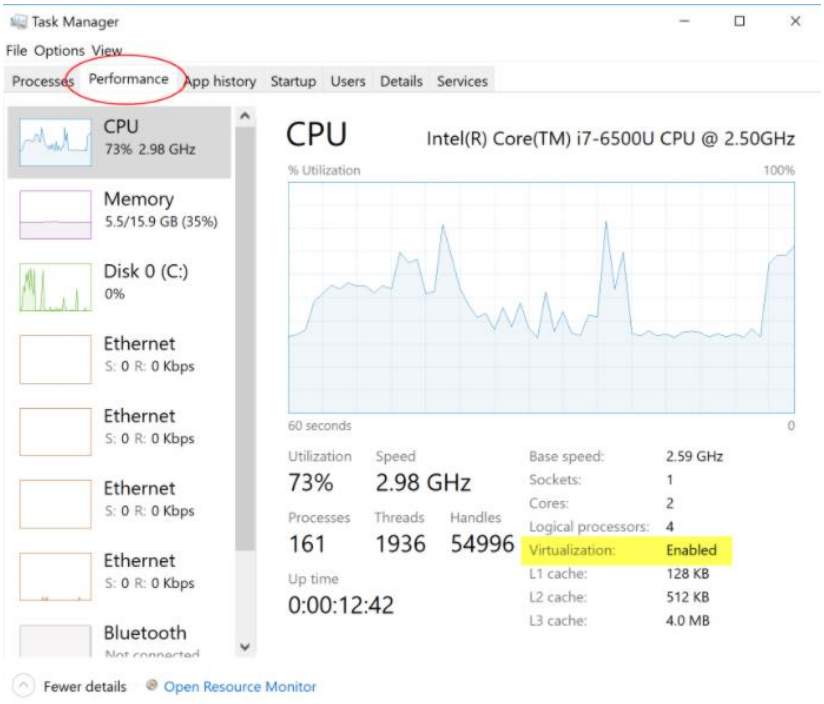
Debido a que se propone la utilización de la herramienta de virtualización Docker y Docker-compose se deben considerar de manera primordial sus requerimientos de instalación. Tal como se menciona en su documentación oficial Get Docker | Docker Documentation, Docker tiene compatibilidad con los sistemas operativos Mac, Windows y Linux; para cada uno de ellos se debe cumplir con un conjunto de requerimientos, así como de un flujo de instalación. En la presente guía se va a seguir el flujo pertinente para el sistema operativo Windows.

A.2.1. Virtualización

La virtualización de hardware en la configuración de la BIOS debe estar activada. El flujo de configuración vería dependiendo del tipo de placa madre y de procesador con los que el ordenador cuente. A continuación, se muestra un ejemplo para el caso de un ordenador con un procesador Intel Core i5 con una placa madre de marca Asus H410.



Se puede verificar dicha configuración con la ayuda del asistente de tareas de Windows, tal como se muestra en la imagen:



Una vez activada la tecnología de virtualización, para el caso de Windows, se debe cumplir con uno de los dos siguientes requerimientos adicionales para una adecuada instalación de Docker Desktop:

- WSL: Windows Subsystem for Linux es una capa de compatibilidad desarrollada por Microsoft para poder ejecutar Linux de manera nativa en Windows.

- Hyper-v: se trata de una tecnología de virtualización desarrollada por Microsoft. Esta tecnología de visualización está disponible únicamente para la versión del sistema operativo Windows PRO.

Preferentemente se busca la instalación de WSL sobre la activación de Hyper-v por motivos de compatibilidad con otras herramientas de virtualización como lo es el caso de VMware.

A continuación, se describe el flujo de instalación de las herramientas de virtualización mencionadas previamente.

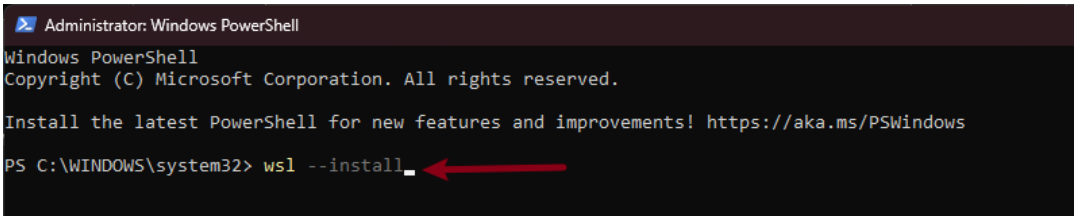
A.2.2. Windows Subsystem for Linux (WSL)

Actualmente hay dos formas de instalar WSL dependiendo de la versión del sistema operativo.

Se recomienda intentar el flujo por defecto que corresponde para las versiones de Windows 10 versión 2004 o mayor (Build 19041 o mayor) y Windows 11. Esl flujo instala y configura automáticamente wsl en su versión 2:

1. Dentro de la terminal de Windows Power Shell como administrador ejecutar:

   wsl --install

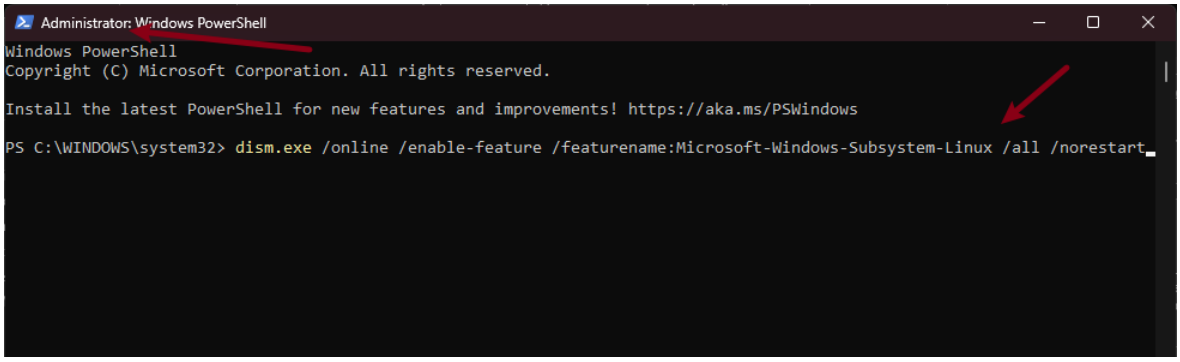


Por otro lado, si se posee una versión de Windows diferente, se requiere de una instalación manual. Los pasos mostrados a continuación son un resumen del tutorial oficial de Microsoft para la instalación de WSL de manera manual: [Manual installation steps for older versions of WSL | Microsoft Docs]

1. Habilitar “Windows Subsystem for Linux”

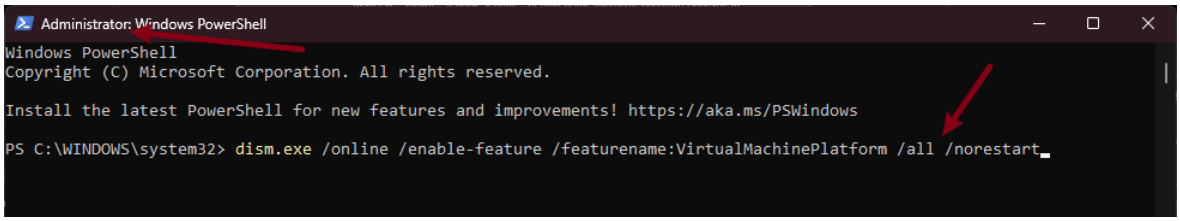
   En una terminal de Windows Power Shell como administrador ejecutar:

   dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart



2. Habilitar la opción de manejo de máquinas virtuales en Windows
   En una terminal de Windows Power Shell como administrador ejecutar:

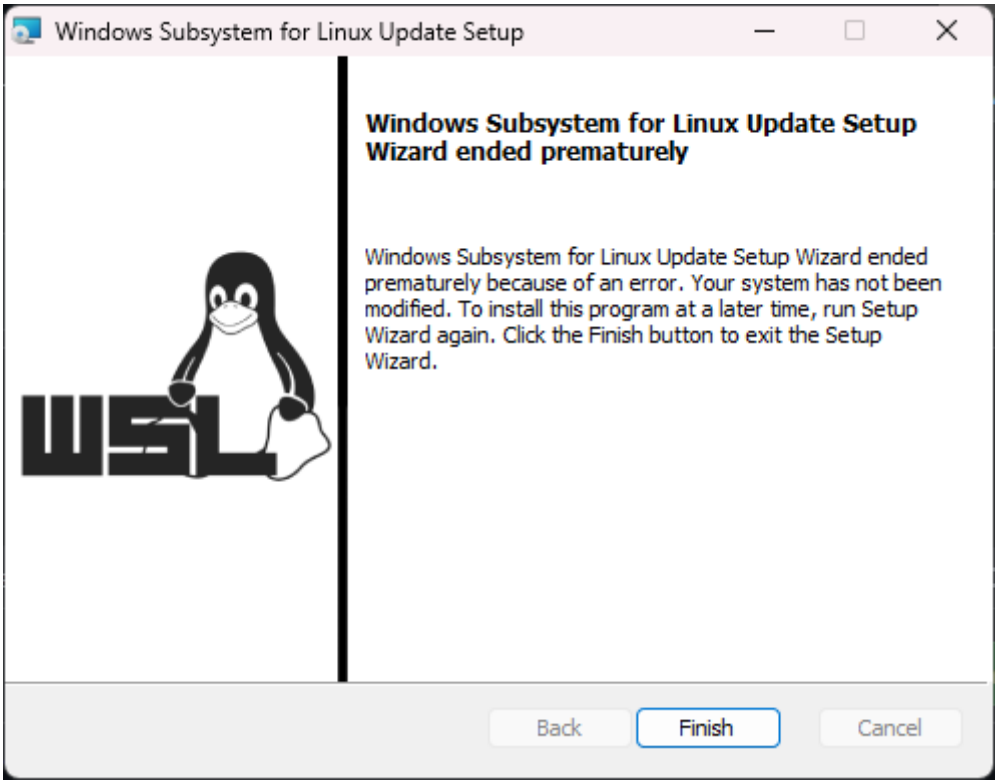
   dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart



   Reiniciar el Sistema para garantizar la instalación/actualización de WSL

3. Descargar e instalar la actualización del kernel de Linux para Windows

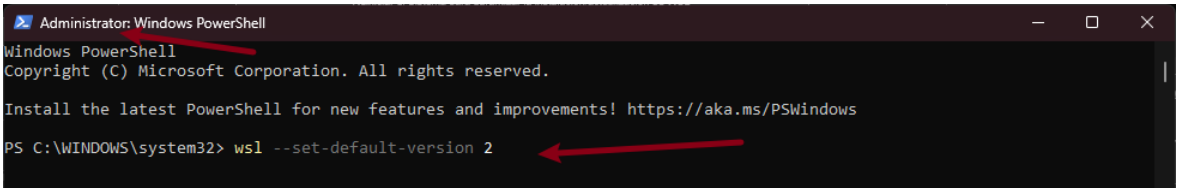
   El enlace se encuentra disponible en el siguiente url: [Download Linux Kernel]



4. Establecer WSL 2 por defecto

   En una terminal de Windows Power Shell como administrador ejecutar:

   wsl --set-default-version 2

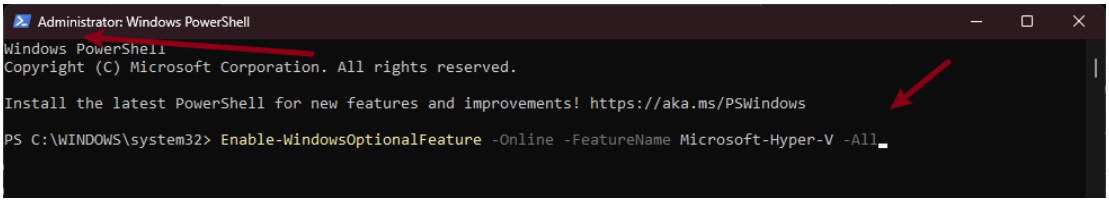


A.2.3. Hyper-V

Los pasos mostrados a continuación son un resumen del tutorial oficial de Microsoft para habilitar Hyper-V en el caso de contar con la versión Windows 10 PRO: [Enable Hyper-V on Windows 10 | Microsoft Docs]

1. En una terminal de Windows Power Shell como administrador ejecutar:

   Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All



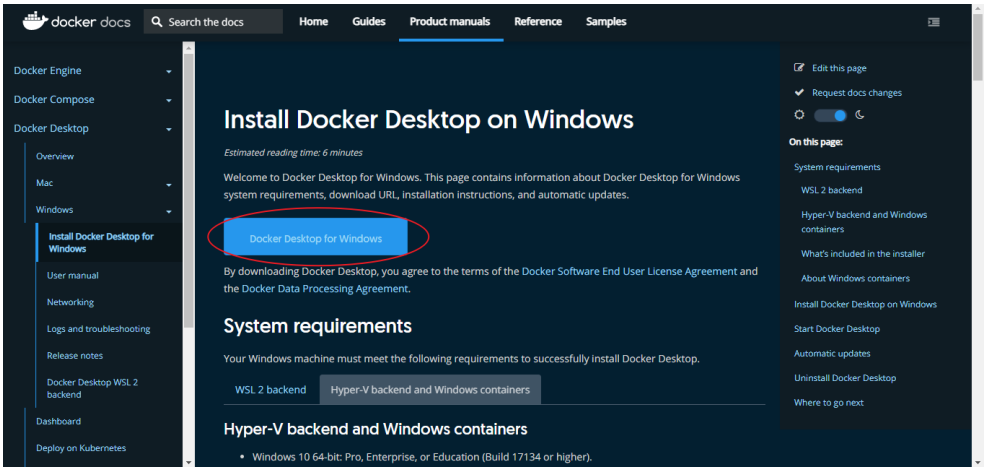
2. Reiniciar el sistema

Es importante mencionar que los pasos mencionados en este documento para instalar y habilitar WSL y Hyper-V pueden variar dependiendo de la versión de Windows. Toda la información oficial se encuentra en los enlaces previamente proporcionados.

A.3. Instalación de Docker

La información detallada del proceso de instalación de Docker en Windows es detallada en la página oficial de Docker Hub: [Install Docker Desktop on Windows | Docker Documentation]

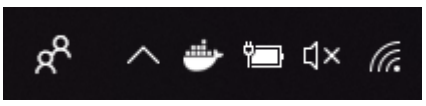
1. Descargar Docker Desktop para Windows del enlace proporcionado



2. Realizar una instalación típica de Windows

3. Si durante la instalación se solicita instalar complementos y/o drivers adicionales, se debe aceptar y continuar.

4. Una vez que se observe el siguiente ícono en la barra de inicio, Docker está listo para su uso.

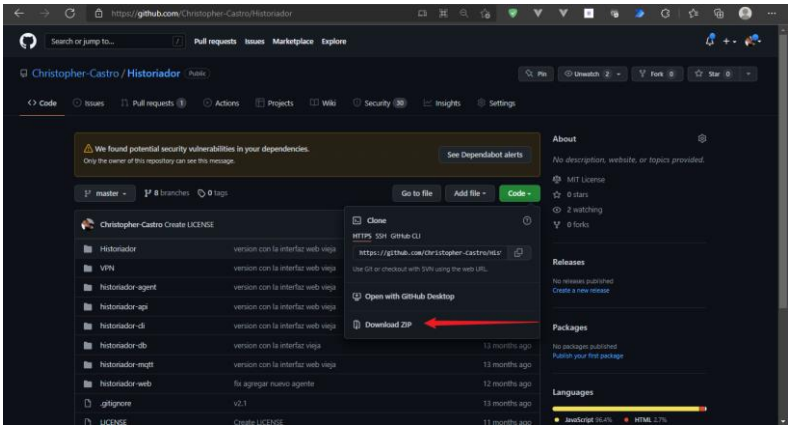


Es importante mencionar que el flujo detallado en este apartado corresponde específicamente para la instalación de Docker Desktop para Windows, que básicamente

instala Docker, Docker-compose y una interfaz gráfica que facilita el manejo de los contenedores.

A.4. Instalación del historiador de procesos

El código fuente se encuentra disponible en el siguiente repositorio de GitHub: [Christopher-Castro/Historiador (github.com)](). Se recomienda realizar una copia del repositorio completo utilizando Git o en su defecto descargando el archivo comprimido tal como se muestra en la siguiente imagen.



Dentro de la carpeta se pueden encontrar los 2 archivos de mayor importancia para la ejecución del aplicativo: DockerFile, para la creación de las imágenes de los contenedores y Docker-compose.yml, para la creación del cluster de ejecución del proyecto.

A.4.1. Creación de la imagen de Docker

Abrir un terminal de Windows en la ruta raíz de la carpeta donde se encuentra el código fuente descargado y ejecutar el siguiente comando:

'docker build -t christopher57/historiador_image .'

Se iniciará el proceso de creación de la imagen que contiene el código fuente y las dependencias necesarias para la ejecución de cada uno de los 5 módulos que conforman el historiador de procesos.

A continuación, se muestra el contenido del archivo DockerFile:

```
FROM node:16

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
```

```
# A wildcard is used to ensure both package.json AND package-lock.json are
copied
# where available (npm@5+)
COPY . .

RUN cd /usr/src/app/historiador-db && npm install

RUN cd /usr/src/app/historiador-mqtt && npm install

RUN cd /usr/src/app/historiador-api && npm install

RUN cd /usr/src/app/historiador-agent && npm install

RUN cd /usr/src/app/historiador-agent/gui && npm install

RUN cd /usr/src/app/historiador-web && npm install
```

El construir manualmente las imágenes resulta de ayuda cuando se requiere visualizar alguna modificación que se ha realizado sobre el código fuente.

Por otro lado, si solo se desea ejecutar las imágenes de Docker sin la realización de ninguna modificación sobre el código fuente, gracias a Dockerhub, se pone a disposición la imagen del contenedor previamente construida para su descarga utilizando utilizando una conexión a internet. Si se opta por este camino no es necesario la ejecución del comando para hacer la construcción de la imagen de Docker.

Es importante mencionar que este proceso corresponde únicamente para la creación de la imagen que contiene el código fuente del proyecto. Hay otras imágenes de contenedores como la de la base de datos PostgreSQL, la de la herramienta de gestión de base de datos pgadmin y la imagen de la base de datos no relacional MongoDB que utiliza el contenedor historiador-mqtt para su funcionamiento que son propias de cada desarrollador y de igual forma están disponibles para su descarga desde DockerHub.

A.4.2. Ejecución de los contenedores mediante Docker-compose

Las imágenes de los contenedores de Docker se encuentran en el siguiente repositorio de DockerHub: Docker Hub

La instalación del historiador se la realiza con la ayuda de Docker-compose que permite la creación y ejecución de contenedores a través de un archivo de texto bajo el nombre “Docker-compose.yml” mostrado a continuación y que también se encuentra disponible en el repositorio de Github.

```yaml
version: "3.8"

services:
  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_DB: Historiador
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: example
      PGDATA: /var/lib/postgresql/data
    volumes:
      - db-data:/var/lib/postgresql/data
    ports:
      - "5432:5432"


  pgadmin:
    image: dpage/pgadmin4
    restart: always
    environment:
      PGADMIN_DEFAULT_EMAIL: user@domain.com
      PGADMIN_DEFAULT_PASSWORD: example
      PGADMIN_LISTEN_PORT: 80
    ports:
      - "80:80"
    volumes:
      - pgadmin-data:/var/lib/pgadmin
    links:
      - "db:pgsql-server"
    depends_on:
      - db

  mqtt:
    image: christopher57/historiador_image
    working_dir: /usr/src/app/historiador-mqtt
    command: "npm run start-dev"
    ports:
      - "1883:1883"
    depends_on:
      - mongo
      - db
```

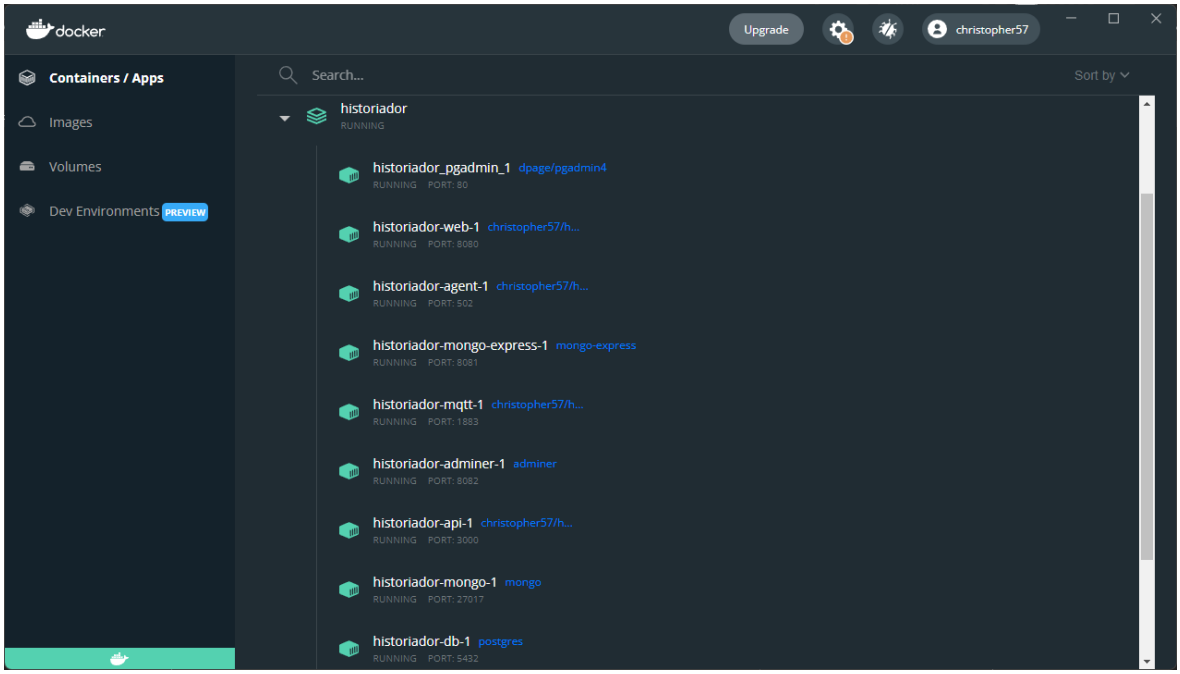
```yaml
agent:
  image: christopher57/historiador_image
  working_dir: /usr/src/app/historiador-agent/gui
  command: "npm run start-dev"
  ports:
    - "8000:8000"
    - "502:502"
  depends_on:
    - mqtt
    - api

api:
  image: christopher57/historiador_image
  working_dir: /usr/src/app/historiador-api
  command: "npm run start-dev"
  ports:
    - "3000:3000"
  depends_on:
    - db

web:
  image: christopher57/historiador_image
  working_dir: /usr/src/app/historiador-web
  command: "npm run start-dev"
  ports:
    - "8080:8080"
  depends_on:
    - api
    - mqtt

mongo:
  image: mongo
  restart: always
  environment:
    MONGO_INITDB_ROOT_USERNAME: root
    MONGO_INITDB_ROOT_PASSWORD: example
  ports:
    - 27017:27017
  volumes:
    - mongodb-data:/data/db


mongo-express:
  image: mongo-express
  restart: always
  ports:
    - 8081:8081
```

```
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: example

    depends_on:
      - mongo

  mysql:
    image: mysql
    command: --default-authentication-plugin=mysql_native_password
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: example
    volumes:
      - mysql-data:/var/lib/mysql

  adminer:
    image: adminer
    restart: always
    ports:
      - 8082:8080

volumes:
  db-data:
  pgadmin-data:
  openvpn-data:
  noip-config:
  noip-etc:

  mongodb-data:
  mysql-data:
```

Para ejecutar el Docker-compose.yml se debe ejecutar el siguiente comando desde la terminal de Windows desde la ruta en la que se encuentra el archivo.

docker-compose up -d

De esta forma se descargan las imágenes desde el repositorio de DockerHub y se crean los contenedores respectivos.



5. Puertos de comunicación utilizados

En el presente apartado se define la forma en la que cada uno de los módulos descritos previamente se interrelaciona con los demás con el objeto de formar un todo que representa el historiador de procesos.

Cada uno de los módulos fue creado basado en un servidor cuyos servicios fueron definidos y configurados según su correspondiente framework: express.js para el back-end y Vue.js para el front-end. De esta forma cada uno de los módulos cuenta con uno o más puertos de comunicaciones mismos que operan sobre el protocolo IP por los cuales intercambiarán información ya sea con uno de los módulos que conforman el historiador de procesos o con alguna entidad externa al mismo en función de la lógica descrita previamente en cada uno de sus correspondientes apartados. A continuación, se describen los puertos utilizados por cada módulo:

- Módulo historiador-db: Para este caso se trata de la entidad de la base de datos que utiliza el puerto por defecto de PostgreSQL (5432) para permitir solicitudes de consulta, creación o modificación de los registros existentes en la misma.
- Módulo historiador-mqtt: El servidor MQTT permite que clientes se conecten al broker utilizando el puerto por defecto definido para el protocolo mqtt (1883) con el objeto de realizar consultas al módulo historiador-db a través del puerto 5432.

- Módulo historiador-agent: Tal como se lo expuso en 2.4, un agente puede dividirse en 3 partes de tal forma que:
    - Cliente MQTT: utiliza el puerto 1883 para comunicarse con el módulo historiador-mqtt.
    - Driver de comunicaciones: utiliza el puerto 502 para comunicarse a través del protocolo Modbus TCP o el puerto 3306 para acceder a una entidad de base de datos externa basada en el motor MySQL.
    - Fuente de información: el driver de comunicaciones utiliza sus puertos definidos para iniciar conexión con algún dispositivo, entidad o aplicación externa a través de Modbus TCP (502) o las consultas SQL a un motor de base de datos MySQL (3306).

  Adicionalmente, la interfaz gráfica web de configuración de agentes hace uso del puerto 8000 para mostrar su contenido en el navegador web.
- Módulo historiador-api: Utiliza el puerto 3000 para recibir consultas HTTP y traducirlas a consultas hacia la base de datos utilizando el puerto 5432.
- Módulo historiador-web: hace uso del puerto 8080 para mostrar toda la información que es solicitada al módulo historiador api a través del puerto 3000 en el navegador web.

6. Uso del aplicativo

Para la operación y monitoreo del historiador de procesos de dispone principalmente de las interfaces de usuario proporcionadas por los módulos “historiador-agent” (http://localhost:8000), que permite la configuración de agentes de comunicación que son los encargados de la recopilación de datos utilizando Modbus TCP y las consultas SQL; e “historiador-web” (http://localhost:8080), que es la interfaz de monitoreo del historiado en donde se puede visualizar tanto la información en tiempo real como los históricos de datos a lo largo del tiempo de los diferentes agentes que mantienen conexión con el aplicativo.

ORDEN DE EMPASTADO