FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

CARRERA DE ESPECIALIZACIÓN EN
INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

Monitoreo ambiental integrado a Enterprise Buildings Integrator de Honeywell

Autor:
Ing. Gonzalo Nahuel Vaca

Director:
Esp. Ing. Pablo Almada (FIUBA-UTN)

Jurados:
Mg. Ing. Christian Yanez Flores (FIUBA)
Esp. Ing. Lucas Fabricio Monzón Languasco (FIUBA-UNNE)
Esp. Ing. Daniel Marquez (FIUBA-UC)

Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires, entre mayo de 2020 y abril de 2021.

Resumen

Esta memoria describe la implementación de una solución realizada para los laboratorios Gador, donde se adapta su sistema de automatización de edificios y gestión empresarial marca Honeywell. La finalidad es integrar una red de sensores que utilizan un protocolo de comunicación que este sistema no puede interpretar.

Se logró cumplir con las necesidades de Gador utilizando contenidos y habilidades desarrollados en las asignaturas de esta especialización. Se creó una arquitectura de datos, se implementaron protocolos de comunicaciones, se programaron servidores y se puso en funcionamiento un sistema de despliegue automático y orquestación de la aplicación.

Índice general

Índice de figuras

Índice de tablas

Dedicado a la memoria del Ing. Valeriy Omelchenko

Capítulo 1

Introducción general

El capítulo presenta las necesidades a satisfacer y una introducción técnica breve, con el objetivo de proveer los conceptos necesarios para comprender el resto de la memoria.

1.1. Motivación

La tendencia tecnológica actual es interconectar los dispositivos a través de la Internet, a tal punto que la cantidad de objetos en el año 2009 superó al número de personas conectadas, y llegado el 2020 la diferencia es de seis veces en favor de las cosas. [1] Los procesos industriales se ven beneficiados con nuevos métodos de control de inventarios y análisis de mediciones, además es posible gestionar los ambientes productivos para lograr una mayor calidad y comodidad. Los datos quedan disponibles para ser procesados por modelos de inteligencia artificial, y la información resultante puede ser vista desde cualquier ubicación y en múltiples plataformas.

Las empresas locales están retrasadas en su progreso tecnológico, muchas no incorporaron sistemas electrónicos en sus procesos o productos, es necesario crear un sistema que logre adaptar la tecnología en uso con el fin de incorporarlas a las nuevas prácticas de negocios. El avance tecnológico modifica el marco normativo de las naciones, para cumplir con los nuevos requerimientos jurídicos, se necesita tener un mínimo de capital. El retraso tecnológico ya no solo genera una pérdida de competitividad, sino que también impide que las empresas coloquen sus mercancías en otros países, por incumplimiento en normas de calidad o de protección del medio ambiente.

La situación de la industria argentina fue la primera razón que impulsó este trabajo, el siguiente paso fue buscar una empresa que quisiera participar de un proyecto adecuado para el pos-grado, y finalmente se logró un acuerdo con los laboratorios Gador S.A. La misión de la compañía es producir medicamentos para la salud humana, con la mejor calidad disponible, y ponerlos al alcance de la comunidad a precios accesibles. [2]

La empresa tiene la necesidad de acceder al mercado estadounidense y para lograrlo se deben satisfacer los requerimientos del *Code of Federal Regulations - Title 21 - Food and Drugs Chapter - Part 11 (21CFR11)*. La norma establece que los registros de las mediciones ambientales de los depósitos y cuartos productivos, se deben almacenar de forma electrónica, pero se debe demostrar que los registros tienen la misma validez y seguridad que aquellos hechos en papel. [3] Esto se

traduce en la necesidad de tener un sistema informático que se encuentre aprobado por la *Food and Drug Administration (FDA)*, que es el organismo encargado de controlar los medicamentos y alimentos que ingresan a los Estados Unidos. Por esa razón Gador tiene comprada una licencia del sistema *Enterprise Buildings Integrator (EBI)* de la marca *Honeywell*, ya que el producto se encuentra aprobado por la FDA. Si bien el programa fue adquirido hace varios años, es indispensable que continúe operando aún cuando los protocolos que utiliza son antiguos. Tampoco es económicamente viable reemplazarlo por un producto moderno que esté aprobado por la FDA, se debe lograr un salto tecnológico manteniendo la plataforma que actualmente está operando.

Los requerimientos que se deben cumplir en las mediciones ambientales de los depósitos y cuartos productivos, estipulan que los sensores se deben someter a un plan de calibración rutinario y realizarles periódicos estudios de perfiles térmicos. Un estudio de perfil térmico se logra tomando una serie de mediciones de temperatura en varios puntos de un ambiente, y con estos datos se procede a calcular las coordenadas de los puntos críticos del cuarto. [4] Los puntos críticos son aquellos lugares donde la temperatura es la más baja o más alta dentro de la habitación. Teniendo los puntos críticos identificados, se procede a colocar sensores de temperatura en esos lugares.

Los periódicos estudios de perfiles térmicos, tienen como consecuencia que cada seis meses se deben mover los sensores de temperatura. La tarea de migrar los dispositivos se vuelve costosa debido a que se encuentran cableados, y además los nuevos recorridos de los cables se deben certificar por el departamento de calidad. El tiempo de migración y de certificación se vería reducido sensiblemente si los equipos fuesen inalámbricos, pero la licencia de EBI que tiene Gador no es compatible con los protocolos de comunicaciones necesarios para lograrlo.

Las plantas de producción de la empresa siguen una arquitectura en donde los sensores reportan sus mediciones a unos controladores lógicos programables o PLC por sus siglas en inglés. Esa comunicación se logra a través de cables que los conectan. Los datos que adquieren los PLCs son entregados al sistema EBI utilizando un protocolo de comunicaciones llamado Modbus TCP. Este protocolo fue creado en el año 1979 y se diseñó teniendo en mente las limitaciones tecnológicas de la época, sin embargo la licencia de EBI que adquirió Gador solo acepta este formato. El modelo lógico de la arquitectura se puede visualizar en la figura 1.1, donde se puede ver una estructura del tipo árbol donde todo converge al sistema EBI.

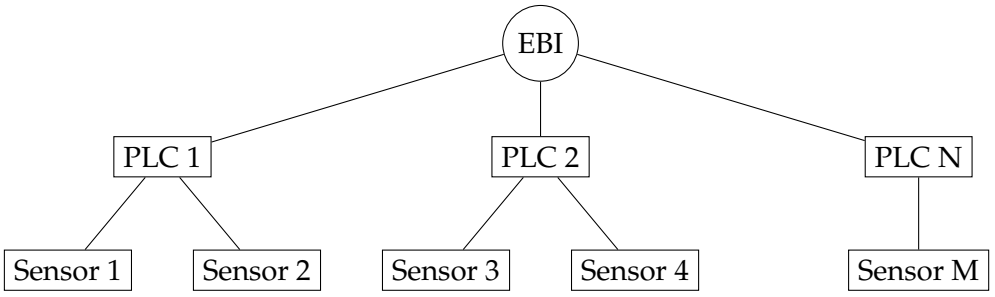

FIGURA 1.1. Red industrial Gador.

1.2. Introducción técnica

El proyecto realizado presentó una serie de desafíos a resolver, siendo el primero de ellos la variedad de tecnologías involucradas, tanto en la problemática a manipular como en la solución a implementada. Para introducir orden en la variedad de conocimientos que forman parte del sistema desarrollado, se introduce un modelo de capas de Internet de las cosas (IoT). El modelo tiene la ventaja que separa los temas en categorías relacionadas con la función o servicio que prestan en la solución, lo que facilita su estudio.

El modelo de capas seleccionado separa los conocimientos en cinco categorías, las cuales son las capas de negocio, aplicación, procesamiento, red y percepción. La capa de negocio agrupa todo lo relacionado con las reglas y el control del sistema, esto incluye supervisar el tráfico de información, verificar el estado de los equipos u otorgar permisos a los usuarios para interactuar con el programa. La capa de aplicación relaciona todas las tecnologías que se encargan de interactuar con el usuario final, es lo que las personas pueden ver como el sistema. La capa de procesamiento agrupa los conocimientos cuya responsabilidad es almacenar y analizar los datos que se generan. La capa de red tiene la finalidad de interconectar los dispositivos para permitir el flujo de datos entre todas las partes involucradas. Finalmente la capa de percepción se refiere a todos los artefactos que manipulan o miden algo que se encuentra en el ambiente, como un sensor o un actuador. Este modelo se encuentra resumido en la tabla 1.1.

TABLA 1.1. Modelo de capas IoT.

Capa	Función
Negocio	Establecer reglas y controlar el sistema
Aplicación	Interactuar con el usuario
Procesamiento	Almacenar y analizar los datos obtenidos
Red	Transportar los datos entre dispositivos
Percepción	Realizar mediciones o acciones en planta

Dependiendo del modelo viabilidad económica de un sistema y de como fue desplegado, la capa de negocio puede tener una funcionalidad contable y calcular los costos de operación. Esta capa puede ser la encargada de determinar y generar la facturación para cobrarle a los usuarios de la aplicación, como así también, de resolver operaciones de transferencia de dinero. La interacción en este nivel es con el personal que administra un sistema, se determina que permisos tiene cada usuario para manipular los servicios ofrecidos y se lleva adelante el registro de acciones y eventos relevantes para el normal funcionamiento del programa.

La experiencia que tiene el usuario al interactuar con la solución pertenece a la capa de aplicación. Aquí se define como se presenta la interfaz gráfica que utilizan las personas, y es común utilizar un formato de sitio web. Las páginas webs tienen la ventaja de ser indiferentes de la plataforma que utiliza el operador, solo importa que pueda ejecutar un navegador. Actualmente, se construyen las interfaces siguiendo un modelo de diseño según el tipo de operación a realizar por el programa, si la solución abarca una interfaz hombre-máquina industrial que debe ser atendida durante toda una jornada laboral, se suele implementar una norma de manejo de situaciones anormales o ASM; si la aplicación es de uso intermitente, se puede usar un esquema de diseño material o *Material Desing* que presenta

una experiencia moderna y fluida, como se puede apreciar en la figura 1.2. Para llevar a delante la interfaz seleccionada se utiliza un servidor que tiene como objetivo proveer los componentes gráficos al dispositivo utilizado, una manera de realizarlo es entregando al cliente una *Single Web Application (SWP)*, logrando que el servidor otorgue todo el código necesario para que el dispositivo del usuario genere por si mismo los componentes gráficos a mostrar. Es importante que el código entregado pueda ser visualizado en múltiples tamaños de pantallas, en la actualidad las personas utilizan ordenadores, tabletas y teléfonos móviles que presentan grandes diferencias en sus dimensiones, cuando una aplicación cumple con este requerimiento se dice que es responsiva.

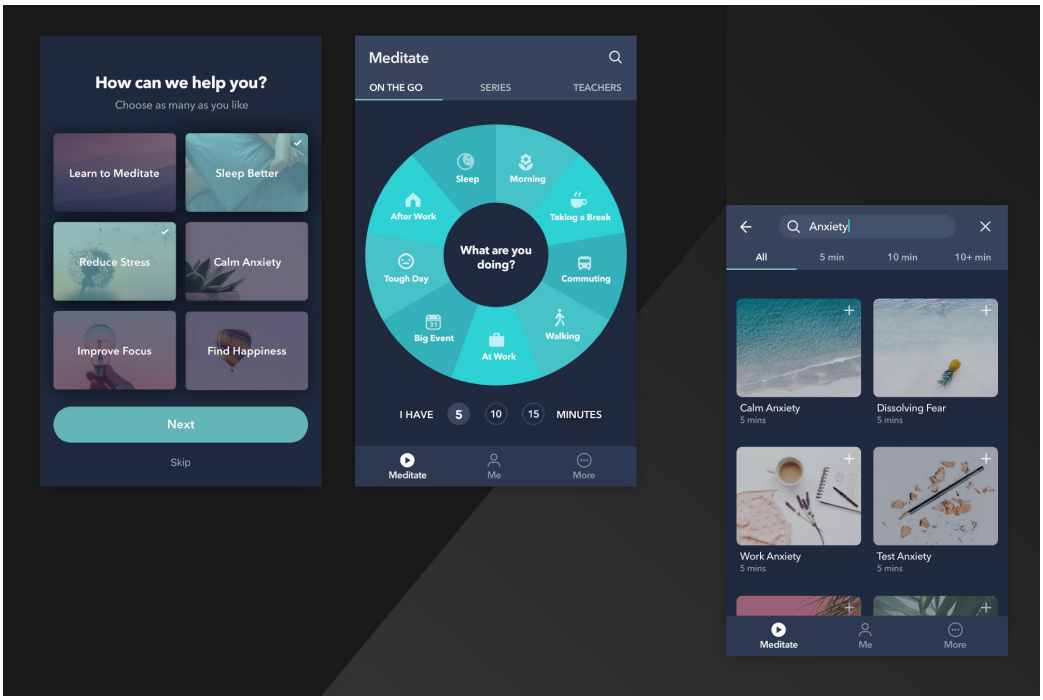

FIGURA 1.2. Ejemplo de interfaz de diseño material. [5]

Para alimentar de datos a la interfaz gráfica se necesita de la capa de procesamiento, que entrega el contenido a mostrar en pantalla. La información puede ser almacenada con distintas tecnologías, siendo una de las principales, las bases de datos relacionales. Este tipo de base de datos se basa en un esquema de tablas que se relacionan entre sí. Estas tecnologías se las suelen llamar SQL, y son utilizadas principalmente en datos de inventarios y sistemas de transacciones de dinero. Existe otro grupo de bases de datos que se denominan no relacionales o NoSQL, esta categoría contienen a las bases de datos tipo clave-valor, documental, de columnas y de grafos.

Las bases de datos clave-valor tratan los datos como una única colección que puede tener campos completamente distintos en cada registro, no existe entonces, ningún tipo de relación entre los miembros de la colección. El uso principal de esta tecnología es gestionar diccionarios dentro de la memoria volátil, ya que se pueden definir tiempos de vida para los datos. La muerte programada de un dato puede ser utilizada para gestionar las sesiones de usuario dentro del programa.

Almacenar los datos de manera documental significa que se agrupa la información siguiendo un criterio de entidades similares, lo cual no significa que exista

una estructura rígida, sino que los datos tienen una naturaleza similar. La persistencia se logra siguiendo un formato de codificación estandar como *XML*, *YAML* y *JSON*.

Las bases de datos orientadas a columnas están pensadas para minimizar el tiempo de búsqueda, principalmente en series temporales. La organización particular de este tipo de tecnologías es afín a los sistemas de IoT ya que los dispositivos de mediciones suelen generar un gran volumen de datos, que se pueden organizar como series temporales.

Una base de datos orientada a grafos presenta la información como nodos que se encuentran relacionados, la diferencia fundamental con los sistemas relacionales es que los nodos no están organizados en tablas, y las relaciones que unen los nodos tienen atributos y no poseen una estructura definida. Este tipo de tecnología permite utilizar la teoría de grafos y posibilita realizar consultas siguiendo modelos matemáticos que forman parte de esa rama de la ciencia.

Se dispone de un repertorio de protocolos pertenecientes a la capa de red para lograr que los dispositivos se comuniquen entre si. Entre los mencionados a lo largo de esta memoria se encuentran el protocolo Modbus, MQTT, HTTP y WebSocket. El manejo de estas tecnologías fue fundamental para lograr que las distintas partes del trabajo interactúen con el exterior.

Modbus es un protocolo que se diseñó teniendo en cuenta su uso para aplicaciones industriales, su prioridad es transmitir los datos manteniendo su integridad aún en ambientes donde el ruido eléctrico es elevado. El protocolo es público y gratuito, lo que provocó que se impusiera en un gran segmento del mercado ya que además es fácil de implementar y requiere poco desarrollo. Los dispositivos de una red Modbus tienen una dirección única y por lo general se asigna un equipo como maestro y el resto como esclavos. La arquitectura descripta presenta varias ventajas, pero la antigüedad del protocolo y su diseño para dispositivos del tipo PLC, hace que no sea adecuado para aplicaciones IoT.

Para interconectar a los dispositivos bajo un esquema de publicación-subscripción se utiliza el protocolo MQTT. El protocolo está diseñado para conexiones en lugares remotos donde los dispositivos funcionan con un ancho de banda limitado. El resultado es que los mensajes son pequeños y consumen poca batería de los equipos involucrados, por lo que se usa frecuentemente en los sistemas de IoT. El tráfico es gestionado por un servidor del tipo broker que decide quienes son los destinatarios de un mensaje en particular, el resto de los dispositivos son clientes del broker. Si un cliente desea transmitir datos, lo hace realizando una publicación a un determinado *topic* y el broker se encarga de determinar quienes deben recibir la información enviada. Quienes quieran obtener los datos publicados a un *topic* en particular, se deben suscribir a él ante el broker. Este al recibir una publicación de un cliente la transmite solo a los clientes que se encuentres subscritos, como se puede ver en el ejemplo de la figura 1.3, donde el cliente 2 no obtiene los datos del sensor porque no se encuentra subscrito.

El protocolo de transferencia de hipertexto (HTTP) está orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. El cliente inicia la comunicación enviando una petición al servidor, este último entrega una respuesta y se cierra el canal. Existe una variante del protocolo llamada HTTPS que agrega una capa de cifrado para que las comunicaciones sean seguras.

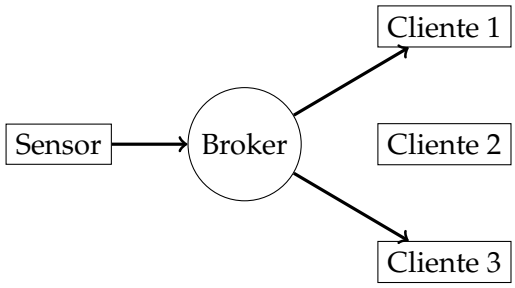

FIGURA 1.3. Ejemplo de comunicación MQTT.

WebSocket es un protocolo similar a HTTP pero con la diferencia que la conexión es bidireccional, esto quiere decir que cuando se logra la conectar al cliente con el servidor, ambos pueden enviar información espontáneamente. Esta cualidad permite realizar transferencias de datos en vivo, con lo que se pueden lograr servicios de *streaming* o *chats*.

Los sensores utilizados en una solución de IoT están incluidos en la capa de percepción, y para que puedan formar parte del sistema se necesita que sean capaces de soportar alguno de los protocolos de comunicaciones mencionados. Dado que el desarrollo de estos dispositivos no formaron parte del proyecto realizado, no se ampliará demasiado en este tema.

Teniendo definido los componentes de las capas, se necesita lograr que todas las partes funcionen como una única entidad. Para lograr este objetivo existen tecnologías de despliegue y orquestación, que cumplen la función de interconectar y mantener los servicios para que trabajen en equipo. Tradicionalmente se solían utilizar máquinas virtuales pero actualmente ese enfoque está quedando en desuso en favor de las tecnologías de contenedores. Una máquina virtual acapara parte de una computadora y funciona como un ordenador independiente, mientras que un contenedor funciona como un sistema operativo independiente pero no acapara los recursos de la computadora principal.

1.3. Estado del arte

En la sección 1.2 se presentó un modelo de capas para analizar las tecnologías. En esta sección se utiliza el mismo esquema para presentar las técnicas que conforman el estado del arte. Es importante mencionar el concepto de nube, ya que las soluciones modernas se basan en utilizar este tipo de plataforma. La nube se refiere a utilizar servicios y servidores provistos por un tercero. Entre los sistemas más representativos se encuentran *Amazon Web Service (AWS)*, *Google Cloud* y *Azure*. Estas empresas ofrecen su infraestructura y una serie de facilidades que promueven un rápido desarrollo y despliegue en el mercado.

Los sensores o actuadores que se utilizan corren un firmware específico para el ecosistema utilizado. Si se decidió utilizar AWS, por ejemplo, lo más probable es que la capa de percepción ejecute *AWS IoT Core* en sus dispositivos. Este esquema es ampliamente utilizado a nivel *enterprise*, por ejemplo, los laboratorios Bayer utilizan el ecosistema de AWS. [6]

En la capa de transporte, los dispositivos se comunican usualmente utilizando los protocolos *LoRaWAN*, *Sigfox*, *ZigBee* o *Bluetooth*. La selección del protocolo

depende de las distancias a cubrir y de las necesidades energéticas. Los sensores convergen luego a un punto de agregación. Desde los puntos de agregación se suelen transmiten los datos al servidor en la nube utilizando el protocolo MQTT.

En la capa de procesamiento se utiliza un esquema de datos de alta disponibilidad. Esto se logra creando réplicas de los datos en distintos servidores. Una de las réplicas se configura como maestro y el resto como esclavos. El servidor maestro es quien se comunica con el exterior de la réplica y retransmite los nuevos datos a los esclavos. Si un servidor maestro sufre un problema, uno de los esclavos se convierte en el nuevo maestro y se mantiene a la réplica funcionando sin interrupciones.

Los datos pueden ser divididos en *shards*, esto se hace para dividir la base de datos según la aplicación. Un ejemplo es separar los datos por región geográfica, de esta manera los clientes de una región en particular pueden tener los servidores con los datos que suelen utilizar cerca de ellos. Para que los *shards* funcionen como una única base de datos, se dispone de un servidor *router* que es la interfaz con el exterior. El *router* recibe las consultas o ingresos de nuevos datos y se encarga de utilizar el *shard* correspondiente. La configuración de este sistema se maneja desde un grupo de servidores destinados para tal fin. Suelen conformar una réplica donde solo se almacenan los datos de configuración. Esta arquitectura de alta disponibilidad se la conoce como granja de datos, se la puede construir con mongoDB [7] y se encuentra visualizada en la figura 1.4.

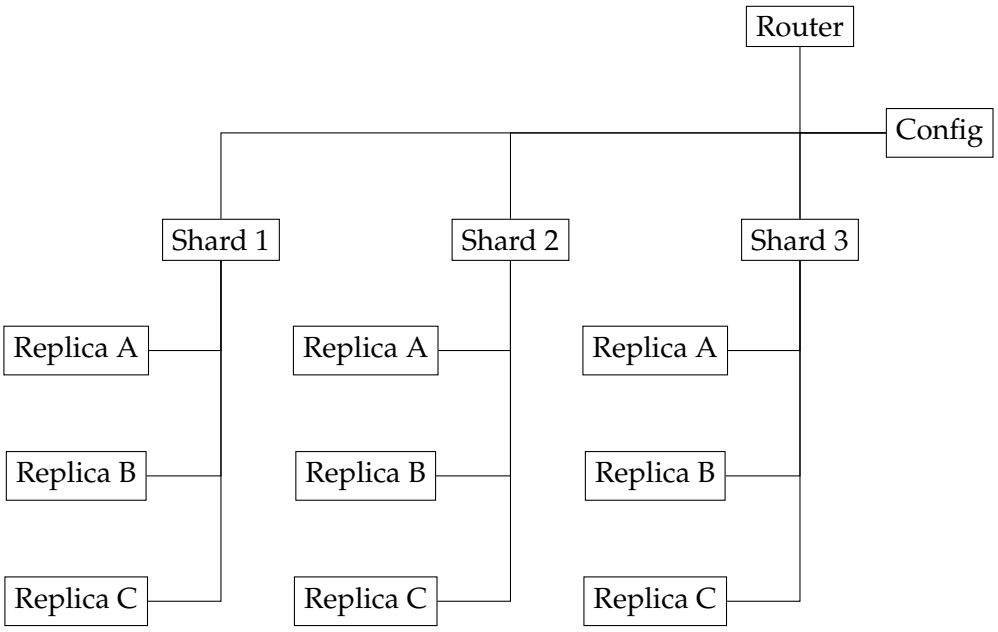

FIGURA 1.4. Arquitectura de datos de alta disponibilidad.

La capa de aplicación se suele diseñar rápidamente con un framework dedicado a la construcción de interfaces gráficas. Uno de los más utilizados es Angular, desarrollado por la empresa Google. El estilo gráfico de diseño material presentado en la sección 1.2 es la más utilizada. Las plataformas de nube ofrecen sus propios sistemas para diseñar la aplicación sin necesidad de escribir demasiado código, pero estas facilidades generan erogaciones adicionales.

La plataforma de nube presenta una capa de negocios donde se puede controlar el tráfico del sistema en ejecución. Desde allí se puede ver la facturación estimada o

el consumo de crédito para mantener en funcionamiento el proyecto. En esta capa se pueden cambiar las variables de entorno del sistema y se pueden controlar el estado de los componentes. Es posible montar servicios que corran programas como *Checkmk* o *Grafana* para visualizar el estado de los dispositivos en campo. O se puede optar por usar los servicios que ofrezca la empresa de nube.

La tecnología que se suele utilizar para orquestar toda la solución es Kubernetes, ya que además de automatizar el despliegue, también permite ajustar la escala. Ajustar la escala se refiere a la capacidad de crear una réplica de un servicio cuando uno de ellos está trabajando cerca de su límite de procesamiento. Es un sistema basado en contenedores y crea un *clúster* a partir de una plantilla donde se definen las reglas de escalamiento.

1.4. Objetivos y alcance

El objetivo principal que cumplió este proyecto fue demostrarle al cliente el potencial de las nuevas tecnologías y la posibilidad de integrarlas a sus actuales sistemas. Se propuso crear una prueba de concepto para evaluar la viabilidad de futuros proyectos. La creación de un sistema que pueda unir equipos que utilizan Modbus con aquellos que usan MQTT, es de relevancia en general para la industria local. Otro objetivo importante fue la de utilizar las técnicas adquiridas durante la cursada de la especialización. Con la finalidad de sementar los conocimientos a través de la práctica.

El proyecto se limitó a desarrollar el software a desplegar en un servidor que fue nombrado Nodos. Esto significa que no se contempló el desarrollo del hardware, en particular los sensores y los puntos de agregación. E esquema del servidor en la red de Gador puede ser visualizado en la figura 1.5. El servidor puede comunicarse con EBI de la misma manera que lo logra un PLC. Además tiene la capacidad de utilizar el protocolo MQTT para conectarse directamente con los sensores o a través de puntos de agregación.

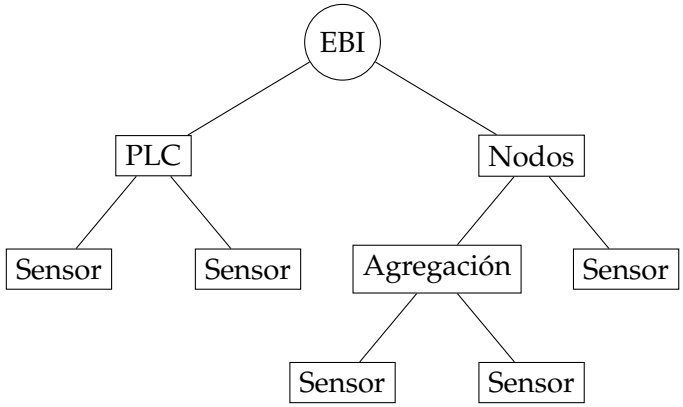

FIGURA 1.5. Red industrial Gador.

Cuando se tuvo definidos los objetivos y el alcance del proyecto, se inició un proceso de negociación con el cliente. Se buscó determinar cuales eran sus necesidades y sus temores respecto del proyecto. Las conversaciones con el cliente dieron como resultado la siguiente lista de requerimientos:

- Debe integrarse a la infraestructura de Gador S.A. sin generar conflictos en otros sistemas.

- Debe crear tramas en el formato *Enterprise Buildings Integrator* y enviarlas al servidor.
- Debe interpretar eventuales mensajes del servidor *Honeywell*.
- Debe interpretar los mensajes de los sensores.
- Debe poder cambiar la frecuencia de lectura de mediciones.
- Debe poseer la capacidad de gestionar los ingresos de usuarios de forma segura.
- Debe permitir que por lo menos cinco usuarios accedan al sistema simultáneamente.
- Debe presentar una interfaz donde se monitoree el estado de los sensores
- Debe permitir elegir un sensor en particular para editarlo.
- Debe poseer un módulo de gestión de usuarios.
- Debe ser compatible con ordenadores de escritorio y smartphones.
- Las contraseñas no persistirán como texto plano.
- Debe persistir todas las modificaciones realizadas a la configuración de los sensores.
- Debe persistir las mediciones obtenidas.

Capítulo 2

Introducción específica

Este capítulo trata sobre los recursos tecnológicos utilizados en el trabajo que fueron desarrollados por terceros.

2.1. Tecnologías utilizadas

Docker es un software que permite el uso y creación de contenedores de Linux. Un contenedor es una unidad que empaqueta el código de un programa junto con sus dependencias. Para crear un contenedor, Docker se vale de el concepto de imagen. Las imágenes son entidades aisladas que corren como contenedores durante el tiempo de ejecución sobre el motor de Docker. Los contenedores son entonces, una abstracción de la capa de aplicación de los sistemas Linux, como se puede visualizar en la figura 2.1. Varios contenedores pueden correr en el mismo ordenador como procesos aislados en el espacio de usuario. La principal diferencia con las máquinas virtuales, es que estas son una abstracción del hardware del ordenador, transformando una única computadora en varios servidores. Los contenedores en cambio, utilizan el kernel del sistema operativo del ordenador físico, no se abstrae un kernel, solo el espacio de aplicación o usuario.

Docker puede ser utilizado para construir imágenes definidas por el usuario. Un Dockerfile es un documento de texto que contiene todos los comandos que un usuario utilizaría para ensamblar una imagen. El programador puede crear automáticamente una serie de comandos en sucesión al ejecutar un único comando sobre el Dockerfile.

Docker Compose, según se define en su documentación [8], es una herramienta para definir y correr aplicaciones de Docker de múltiples contenedores. Permite utilizar un archivo *YAML* para configurar los servicios de la aplicación. Luego, se puede crear y comenzar todos los servicios de la configuración utilizando un único comando.

Nodejs es un servidor asincrónico y orientado a eventos que ejecuta JavaScript. Estas cualidades son una ventaja frente a las aplicaciones de concurrencia de múltiples hilos en el sistema operativo. Ya que no se utilizan candados, no existe la posibilidad de bloquear el servidor. Dado que fue diseñado para construir aplicaciones de red escalables, se eligió para formar parte del trabajo.

Python es un lenguaje de programación interpretado que tiene una gran cantidad de bibliotecas a disposición. Muchas bibliotecas fueron útiles para desarrollar algunos servicios pequeños del proyecto. Se utilizó para acelerar la creación de las partes más livianas del sistema.

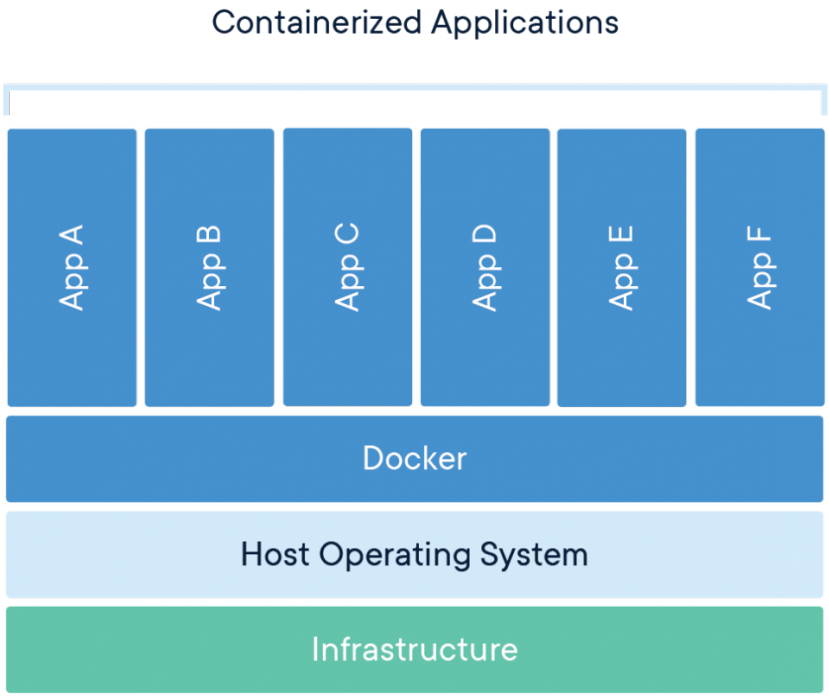

FIGURA 2.1. Arquitectura de Docker. [9]

Eclipse Mosquitto es un broker MQTT. Es liviano y puede ser ejecutado en ordenadores monoplaca. Permite ser configurado con varios niveles de calidad de servicio, se pueden agregar usuarios con distintos permisos y es posible utilizar una capa de seguridad en los mensajes. Uno de sus atractivos es la serie de programas utilitarios que incluye el proyecto y se integran en la terminal del sistema operativo. Estas aplicaciones permiten realizar publicaciones y subscripciones para realizar pruebas, además se pueden crear contraseñas encriptadas para los usuarios.

"MongoDB es una base de datos distribuida, basada en documentos y de uso general diseñada para desarrolladores de aplicaciones modernas y para la era de la nube". [10] Se decidió utilizar esta tecnología para la capa de procesamiento debido a la afinidad que posee para realizar aplicaciones IoT. Además es la base de datos documental más utilizada actualmente [11] y se evaluó como aspecto relevante, ganar experiencia con esta tecnología.

Redis un almacén de estructuras de datos en memoria y puede ser usado como una base de datos, una *caché* de memoria y un broker para crear un bus de eventos. Esta tecnología permite hacer persistencia de datos pero solo en el esquema de clave-valor, permite también establecer tiempos de vida para los datos.

2.2. Bibliotecas y paquetes de terceros

Angular es un framework de diseño de aplicaciones para crear una SWA. Se utiliza codificando en el lenguaje TypeScript que es un superset de JavaScript. Está basado en el paradigma de componentes para generar aplicaciones escalables y poder reutilizar el código escrito en otros proyectos. Pone a disposición una colección de bibliotecas para manejar formularios, comunicaciones cliente-servidor, *routing*, entre otras. Tiene entre sus facilidades ofrecidas, un cliente de terminal que acelera los tiempos de desarrollo. Por todas estas ventajas, se decidió inclinarse por este framework.

Angular Material es una biblioteca de componentes de interfaz gráfica para Angular. Está pensada para construir una experiencia consistente y funcional, adhiriendo con los principios de diseño más utilizados. Estos principios se refieren a la portabilidad entre navegadores y la independencia de dispositivos.

Express es un framework para realizar servidores con Nodejs. Está pensado para construir aplicaciones webs e interfaces de programación de aplicaciones (API). Una API se encarga de definir las interacciones entre múltiples programas o puede intermediar entre componentes de hardware y software. Express es la biblioteca más utilizada para trabajar con Nodejs. [12]

Eclipse Paho es una biblioteca MQTT que está disponible para varios lenguajes. En el trabajo realizado se utilizó para darle conectividad a los servicios programados en Python. La biblioteca provee una clase cliente que habilita la comunicación con un broker MQTT, además provee funciones auxiliares que permiten codificar el código con mayor facilidad.

MQTTjs es una biblioteca que provee de las herramientas para crear un cliente MQTT en Nodejs y en un navegador. Se lo puede instalar de forma global en el sistema para hacer uso de las herramientas de terminal que ofrece. Las herramientas permiten hacer pruebas de subscripción y publicación de mensajes.

ws es una biblioteca de Nodejs para realizar servidores con la capacidad de entablar una conexión WebSocket. Es fácil de utilizar y es altamente configurable. Se decidió usarla debido a que la documentación es completa y simple de entender.

Mongoose es una biblioteca de modelado de datos de objetos(ODM). Se creó para trabajar con MongoDB y está disponible para la plataforma Nodejs. El ODM gestiona las relaciones entre los datos, provee de una validación de esquema y se usa para traducir los objetos del programa en ejecución en una representación dentro de la base de datos.

bcrypt es una biblioteca que contiene funciones para encriptar contraseñas. La encriptación se basa en un esquema de *hashing* e incorpora una *salt* para proteger los datos de un ataque *rainbow table*. Para darle resistencia a los ataques de búsqueda por fuerza bruta, la biblioteca implementa una función adaptativa que por cada iteración se vuelve más lenta. Esto hace que su resistencia sea fuerte aún cuando el ordenador que realiza el ataque sea potente. Por estas razones se decidió usar este recurso en el trabajo para proteger las contraseñas de los usuarios.

jsonwebtoken es un paquete que permite crear un *JavaScript Web Token (JWT)*. JWT es un estandar que se utiliza para la fabricación de *tokens* de acceso que permiten identificar una entidad y determinar cuales son sus privilegios en el sistema. El token está formado por una cabecera, una carga útil y una firma. La cabecera

identifica el algoritmo de encriptación y el tipo de token. La carga útil lleva consigo la información relevante para el funcionamiento. Finalmente la firma cierra el token para certificar la llave privada del servidor. Es relevante mencionar que la biblioteca está diseñada para funcionar con Nodejs

La biblioteca chai ofrece herramientas para hacer pruebas del software escrito. Fue diseñada para Nodejs y puede ser integrada a cualquier framework de JavaScript. En el trabajo fue combinado con Mocha, que es un framework de pruebas para Nodejs. Las pruebas obtenidas al combinar estos dos recursos fueron fundamentales para detectar comportamientos no deseados.

Para que el código escrito en Python pueda utilizar el protocolo Modbus TCP se usó la biblioteca pyModbusTCP. Este recurso permite crear un cliente para acceder a un servidor o bien crear una aplicación que se comporte como esclavo.

oitc/modbus-server es un servidor Modbus TCP realizado en Python y disponible como imagen de Docker. Está configurado para utilizar el puerto 5020 para evitar problemas de permisos con el sistema operativo. [13]

2.3. Sistema propietario del cliente

EBI es un sistema de automatización de edificios y gestión empresarial creado por la firma Honeywell. Ofrece herramientas para dotar a las dependencias de las empresas de inteligencia para incrementar la comodidad, mejorar la seguridad y reducir los costos operativos.

La solución ofrece gestionar una red de edificios a través de una única interfaz gráfica. Pretende reducir los tiempos de respuesta frente a situaciones anormales y mejorar la seguridad. Esta tecnología es compatible con dispositivos y software de terceros, con la idea de ofrecer escalabilidad.

Este software es un ecosistema de módulos que pueden ser adquirido a través de licencias para agregar las siguientes funcionalidades:

- Gestión de consumo energético
- Seguridad de vida
- Control de acceso y intrusión
- Vídeo vigilancia

Las distintas licencias que ofrece Honeywell permiten que EBI utilice los protocolos BACNet, OPC, LonWorks y Modbus TCP. Gador adquirió el producto con la licencia Modbus TCP para conectar sus PLCs de distintas marcas. Como se puede observar en la figura 2.2, se utiliza el software para visualizar los sensores de los distintos cuartos de producción y depósitos. En la figura 2.3 se puede visualizar que EBI está a cargo del control ambiental de las plantas del cliente. El sistema es el corazón de la gestión de edificios de la compañía.

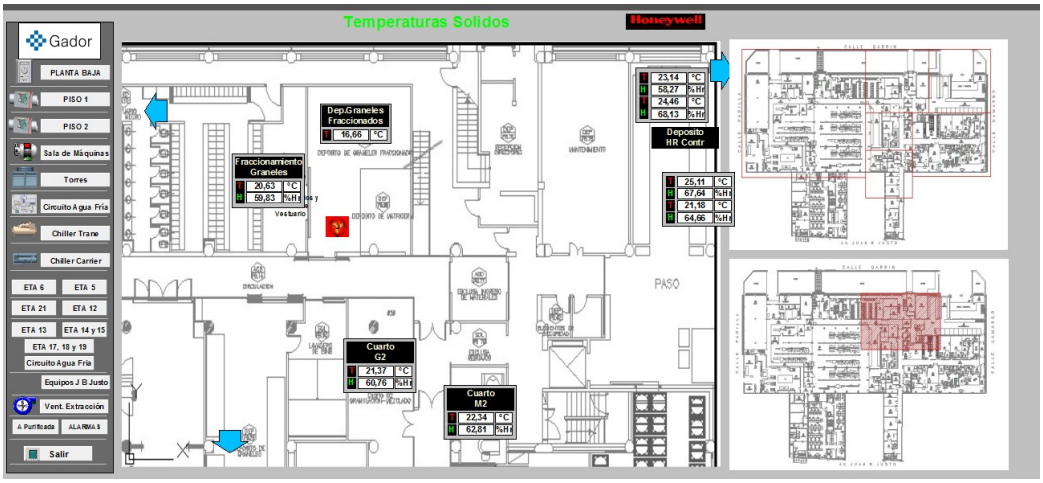

FIGURA 2.2. Control de temperatura de sólidos.

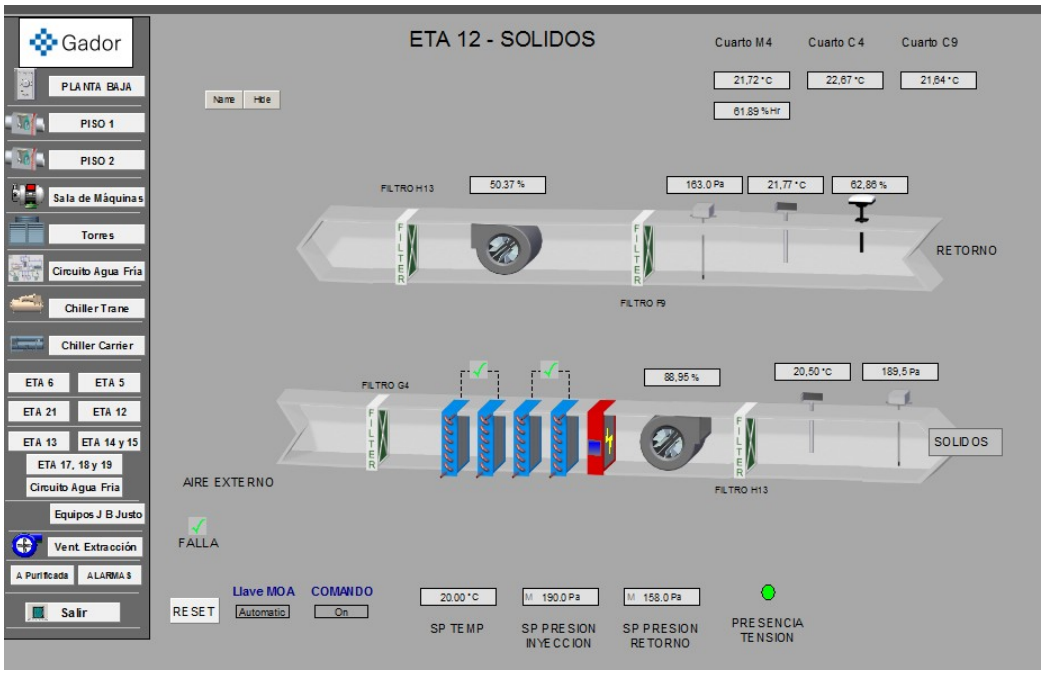

FIGURA 2.3. Unidad de tratamiento de aire de sólidos.

Capítulo 3

Diseño e implementación

Capítulo 4

Ensayos y resultados

Párrafo introductorio del capítulo.

4.1. Pruebas unitarias

4.2. Simulaciones

4.3. Guiones y comandos

4.4. Pruebas del cliente

Capítulo 5

Conclusiones

Este capítulo trata sobre el valor agregado que se le dio al cliente, el aprendizaje adquirido y los siguientes pasos a seguir.

5.1. Resultados obtenidos

El trabajo logró cumplir con los requerimientos que solicitó el cliente, y fueron listados en el capítulo 1. Esta situación sirvió para entablar una relación positiva con el departamento de ingeniería de Gador, ya que el cliente manifestó su conformidad con los resultados obtenidos.

La planificación original del trabajo se pudo cumplir pero solo incrementado la cantidad de horas dedicadas. El principal motivo de retraso que demandó una mayor dedicación horaria fue la poca información sobre el sistema propietario en planta. Además, durante la cursada de la especialización se vieron temas de testeo de software que hicieron visibles ciertas fallas del trabajo. Se dedicó un gran esfuerzo para depurar el código y lograr así una calidad de producción.

La imposibilidad de realizar pruebas dentro de la infraestructura de Gador fue un riesgo que lamentablemente se manifestó. Solo pudo ser sorteado utilizando una máquina virtual con una licencia de uso único de seis horas de duración para verificar la comunicación. El riesgo que por fortuna no se hizo realidad fue que alguna de las personas necesarias para realizar el trabajo se enfermara de *covid-19*, o que se tomaran decisiones de prevención que afectaran la normalidad del desarrollo.

Las técnicas que mejor resultado dieron durante la creación del trabajo fue la automatización y despliegue de contenedores usando *Docker* y *Docker Compose* y el desarrollo orientado a pruebas. La combinación de estos conocimientos genera software de calidad de producción que puede ser desplegado con gran facilidad en múltiples plataformas.

5.2. Trabajo futuro

La principal mejora a realizar es en la capa de negocios, que si bien cumple con los requerimientos del cliente, tendría un salto de valor incorporar *Checkmk* al sistema. Finalmente quedaría incorporar el trabajo a la infraestructura de Gador, para lograrlo se debe crear el hardware necesario. El siguiente paso natural es iniciar un proyecto para diseñar los sensores y puntos de agregación para tener una solución completa.

Bibliografía

[1] Dave Evans. «Hacia adonde se dirige la tecnología». En: *Cisco Internet Business Solutions Group (IBSG)* (2011).

[2] Gador. *Misión Gador*. https://www.gador.com.ar/corporacion/mision-gador/. Abr. de 2021. (Visitado 26-04-2021).

[3] U.S. Department of Health y Human Services. «Guidance for Industry». En: *Center for Drug Evaluation and Research (CDER)* (2003).

[4] Jean Bédard. «Temperature mapping of storage areas». En: *Technical supplement to WHO Technical Report Series, No. 961, 2011* (2014).

[5] Google. *Announcing the Material Design Award Winners for 2020*. https://material.io/blog/mda-2020-winners. Dic. de 2020. (Visitado 04-02-2021).

[6] AWS. *Bayer Crop Science Drives Innovation in Precision Agriculture Using AWS IoT*. https://aws.amazon.com/es/solutions/case-studies/bayer-cropscience/?c=i&sec=cs3. Mar. de 2021. (Visitado 06-03-2021).

[7] mongoDB. *Sharding*. https://docs.mongodb.com/manual/sharding. Mar. de 2021. (Visitado 06-03-2021).

[8] docker docs. *Overview fo Docker Compose*. https://docs.docker.com/compose/. Mar. de 2021. (Visitado 07-03-2021).

[9] Docker. *What is a container*. https://www.docker.com/resources/what-container. Mar. de 2021. (Visitado 07-03-2021).

[10] mongoDB. *La base de datos líder para aplicaciones modernas*. https://www.mongodb.com/es. Mar. de 2021. (Visitado 07-03-2021).

[11] DB-Engines. *DB-Engines Ranking*. https://db-engines.com/en/ranking. Mar. de 2021. (Visitado 07-03-2021).

[12] Paul Serby. *How and why to build a consumer app with Node.js*. https://venturebeat.com/2012/01/07/building-consumer-apps-with-node/. Ene. de 2012. (Visitado 08-03-2021).

[13] Michael Oberdorf. *oitc/modbus-server*. https://hub.docker.com/r/oitc/modbus-server. Mar. de 2021. (Visitado 08-03-2021).