UNIVERSIDAD DE LA FRONTERA
FACULTAD DE INGENIERÍA Y CIENCIAS
DEPTO. DE CIENCIAS DE LA COMPUTACIÓN E INFORMÁTICA

Documentación de Integración de FIWARE para Soluciones de SmartCities.

Integrantes:
Manuel Espinoza
Felipe Quezada
Profesor:
Mg. Jorge Diaz
Asignatura:
Gestión de Proyectos Informáticos

Contenidos

Objetivos

Objetivo general.

- Realizar un mapeo general de la herramienta FIWARE y documentar el proceso de integración de esta abarcando diferentes tecnologías.

Objetivo específico.

- Explorar y documentar los componentes de FIWARE.
- Diseñar, implementar y documentar los componentes básicos de FIWARE.
- Diseñar, implementar y documentar un ejemplo integrado de IoT con visualización web.
- Diseñar, implementar y documentar un ejemplo de reconocimiento de patentes (machine learning).
- Diseñar, implementar y documentar la comunicación entre web y una aplicación móvil.
- Integrar y documentar el proyecto en Amazon Web Services.

Introducción

FIWARE es una iniciativa de código abierto que define una serie de estándares para el manejo de contexto de datos en diferentes dominios, en los cuales se destaca el de ciudades inteligentes. En cualquier solución inteligente se necesita manejar el contexto de la información, procesando y reportando dicho contexto a actores externos, siendo estos los que actúan sobre el contexto y se encargan de enriquecerlo. Son variadas las ciudades que han apostado por FIWARE, especialmente en Europa en lugares como Valencia y Santander, además de organizaciones que han aportado a generar comunidad en torno a esta herramienta, en específico Atos, Engineering, Orange y Telefónica, las cuales se encargaron de constituir la FIWARE Foundation. Si bien la FIWARE Foundation cuenta con un buen número de miembros, proyectos de código abierto, soluciones y partners estratégicos, existe una sensación generalizada de que a día de hoy queda mucho que avanzar en relación a la formación de una comunidad realmente consolidada, fundamentalmente si se hace foco en lugares como Chile y Latinoamérica. Respecto a lo último, la razón de ser del presente documento consiste en explorar FIWARE, dando a conocer sus componentes y mostrando el flujo de trabajo necesario para implementar soluciones en el contexto de ciudades inteligentes.

Además se plantea un ejemplo completo de implementación de los componentes de FIWARE y la visualización de datos en la web que son enviados desde un dispositivo IOT, para este caso una Raspberry Pi 3 modelo B con un sensor de temperatura y humedad DHT22.

Fiware

FIWARE es un framework de código abierto de componentes de plataformas para acelerar el desarrollo de soluciones inteligentes, combina componentes que permiten la coneccion para el IOT con la gestión de la información contextual y los servicios de Big Data en la Nube, también posee un conjuntos de APIs estándar para la gestión y el intercambio de datos así también como modelos de datos armonizados, es de facil integracion con otro tipo de soluciones y servicios. Además existe la fundación que ayuda activamente a promover la adopción de FIWARE, está apoya a la comunidad proporcionando recursos compartidos y valida las tecnologías, es una Comunidad Abierta independiente cuyos miembros se comprometen a materializar la misión de FIWARE, es decir: "construir un ecosistema abierto y sostenible en torno a estándares de plataformas de software públicos, libres de regalías y basados en la implementación que faciliten el desarrollo de nuevas aplicaciones inteligentes en múltiples sectores". La comunidad FIWARE no sólo está formada por contribuidores a la tecnología (la plataforma FIWARE) sino también por aquellos que contribuyen a construir el ecosistema FIWARE y a hacerlo sostenible en el tiempo. Como tal, las personas y organizaciones que comprometen recursos relevantes en las actividades del Laboratorio de FIWARE o en las actividades de los programas FIWARE Accelerator, FIWARE Mundus o FIWARE iHubs también se consideran miembros de la comunidad FIWARE.

FIWARE como framework para la implementación de soluciones Smart está compuesto por un gran abanico de componentes de software que van desde bases de datos relacionales como MySql, Postgres, Crate-db, bases de datos no relacionales como MongoDB, servicios para persistencia de datos como Cygnus, servicios para gestión de contextos como Orion Context Broker, y también herramientas para visualización de datos como Grafana, además de poder facilitar la integración con soluciones ad hoc gracias a que funciona bajo una arquitectura orientada a microservicios.

El principal y único componente obligatorio de cualquier plataforma o solución "Powered by FIWARE" es el FIWARE Orion Context Broker Generic Enabler, que aporta una función fundamental en cualquier solución inteligente: la necesidad de gestionar la información contextual, permitiendo realizar actualizaciones y acceder al contexto.

Construido alrededor del Context Broker de FIWARE, se encuentra un diverso conjunto de componentes complementarios de FIWARE están disponibles para su uso:

Interfaz con la Internet de las cosas (IOT), los robots y los sistemas de terceros, para capturar las actualizaciones de la información de contexto y traducir las actuaciones necesarias.

Gestión, publicación y monetización de datos contextuales/API, lo que brinda soporte para el control del uso y la oportunidad de publicar y monetizar parte de los datos contextuales gestionados.

Procesamiento, análisis y visualización de la información contextual implementando el comportamiento inteligente esperado de las aplicaciones y/o ayudando a los usuarios finales a tomar decisiones inteligentes.

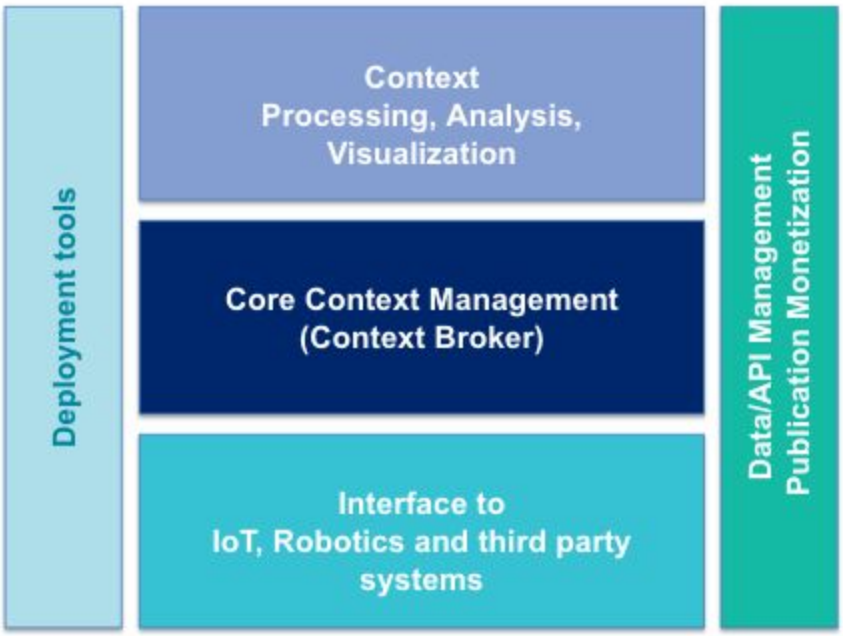

Fig. 1 Componentes Generales de FIWARE..

Componentes de Fiware

Los componentes de FIWARE se puede apreciar con mayor claridad en la Fig.2 ya que se encuentran ordenados por categorías generales en base a sus funcionalidades, a continuación se detalla cada uno de los componentes que componen el framework de FIWARE, la documentación oficial se encuentra disponible en: https://github.com/FIWARE/catalogue

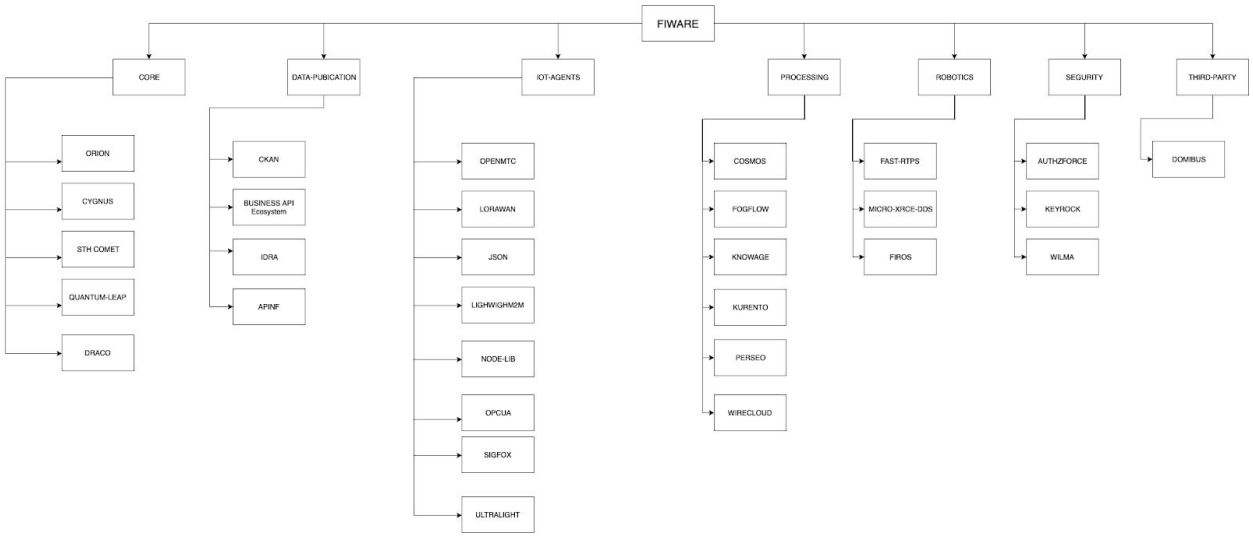

Fig. 2 Mapa de Componentes de FIWARE.

CORE

Documentación oficial disponible en:
https://github.com/FIWARE/catalogue/tree/master/core

ORION

Orion Context Broker permite administrar todo el ciclo de vida de la información de contexto, entre ello las actualizaciones, consultas, registros y suscripciones. Usando el Orion Context Broker, puede crear elementos de contexto y administrarlos a través de actualizaciones y consultas. Además, puede suscribirse a la información de contexto para que cuando ocurra alguna condición (por ejemplo, los elementos de contexto hayan cambiado) reciba una notificación.

CYGNUS

Cygnus es un conector encargado de mantener ciertas fuentes de datos en ciertos almacenes de terceros configurados, creando una vista histórica de dichos datos. Internamente, Cygnus se basa en Apache Flume, una tecnología que aborda el diseño y la ejecución de la recolección de datos y los agentes de persistencia.

STH COMET

Short Time Historic (STH o bien Comet) es un componente del ecosistema FIWARE a cargo de administrar (almacenar y recuperar) información histórica de series de tiempo crudas y agregadas sobre la evolución en el tiempo de los datos de contexto (es decir, valores de atributos de entidad) registrados en una instancia de Orion Context Broker.

QUANTUM-LEAP

QuantumLeap es la primera implementación de una API que admite el almacenamiento de datos NGSI FIWARE NGSIv2 en una base de datos de series temporales, conocida como ngsi-tsdb.

Al final, tiene objetivos similares a los del cometa STH de FIWARE. Sin embargo, Comet aún no admite NGSIv2, está fuertemente vinculado a MongoDB, y algunas de las condiciones y restricciones bajo las cuales se desarrolló ya no se mantienen. Dicho esto, no tiene nada de malo; esto es solo una exploración de una nueva forma de proporcionar datos históricos para FIWARE NGSIv2 con diferentes bases de datos de series de tiempo como backend.

La idea es mantener la base de datos de series temporales intercambiables para esperar tener soporte para diferentes. Comenzamos a probar InfluxDB, RethinkDB y CrateDB. Sin embargo, se ha centrado el desarrollo en el traductor para CrateDB debido a las siguientes ventajas:

- Escalabilidad sencilla con el clúster de base de datos en contenedores listo para usar.
- Soporte de consultas geográficas listo para usar.
- Agradable lenguaje de consulta similar a SQL para trabajar.
- Integración compatible con herramientas de visualización como Grafana

DRACO

Draco es un sistema fácil de usar, potente y confiable para procesar y distribuir datos. Internamente, Draco se basa en Apache NiFi, NiFi es un sistema de flujo de datos basado en los conceptos de programación basada en flujo. Admite gráficos dirigidos potentes y escalables de enrutamiento de datos, transformación y lógica de mediación del sistema. Fue construido para automatizar el flujo de datos entre sistemas. Si bien el término 'flujo de datos' se usa en una variedad de contextos, aquí se refiere al flujo automatizado y administrado de información entre sistemas.

Data-Publication

Documentación oficial disponible en:
https://github.com/FIWARE/catalogue/tree/master/data-publication

CKAN

CKAN es una plataforma de publicación de datos abiertos, ampliamente extendida, que permite la publicación, búsqueda, descubrimiento y consumo de conjuntos de datos abiertos.

En este contexto, las extensiones FIWARE CKAN mejoran las características predeterminadas de CKAN para integrar esta herramienta dentro del ecosistema FIWARE, apoyando la publicación, gestión y visualización de datos contextuales en el momento adecuado, al tiempo que mejora el control de acceso y permite la monetización de datos.

Business API Ecosystem

Business API Ecosystem proporciona a los vendedores los medios para administrar, publicar y generar ingresos de sus productos, aplicaciones, datos y servicios. Business API Ecosystem permite la monetización de diferentes tipos de activos (tanto digitales como físicos) a lo largo de todo el ciclo de vida del servicio, desde la creación de ofertas hasta la facturación, la contabilidad y la liquidación y distribución de ingresos.

Idra

Idra es una aplicación web capaz de federar los sistemas de gestión de datos abiertos (ODMS) existentes basados en diferentes tecnologías que proporcionan un punto de acceso único para buscar y descubrir conjuntos de datos abiertos procedentes de fuentes heterogéneas. Idra uniforma la representación de los conjuntos de datos abiertos recopilados, gracias a la adopción de estándares internacionales (DCAT-AP) y proporciona un conjunto de API RESTful para ser utilizadas por aplicaciones de terceros.

APINF

El APInf API Management Framework es un orquestador de Smart City que se utiliza junto con otros habilitadores de FIWARE. APInf se integra con las tecnologías centrales de FIWARE, como Identity Management, NGSI v2 y Business API Ecosystem junto con API Umbrella para ofrecer un conjunto de herramientas integral que permita a varios propietarios de API operar negocios con sus API.

IOT-AGENTS

Documentación oficial disponible en:
https://github.com/FIWARE/catalogue/tree/master/iot-agents

Un agente de IoT es un componente que permite que grupos de dispositivos envíen sus datos y sean administrados desde un agente de contexto FIWARE NGSI utilizando sus propios protocolos nativos. Los Agentes de IoT también deberían poder abordar los aspectos de seguridad de la plataforma FIWARE (autenticación y autorización del canal) y proporcionar otros servicios comunes al programador del dispositivo.

OPENMTC

OpenMTC es un middleware IoT que consta de funcionalidades de puerta de enlace y backend. La puerta de enlace aloja adaptadores de protocolo que interconectan varios dispositivos y decodifica información específica del dispositivo. Además, las aplicaciones locales pueden preprocesar los datos antes de proporcionarlos a niveles superiores. El Backend agrega información de múltiples puertas de enlace y proporciona información de conexión a los dispositivos. OpenMTC unifica la información de fuentes de datos heterogéneas al agregar semántica y almacenar los datos en una estructura basada en árbol. Mediante el uso de metainformación, los datos se pueden descubrir fácilmente.

LORAWAN

El Agente de Internet de las cosas FIWARE para el protocolo LoRaWAN permite el intercambio de datos y comandos entre dispositivos IoT y los corredores de contexto FIWARE NGSI utilizando el protocolo LoRaWAN. Se basa en la biblioteca FIWARE IoT Agent Node.js.

JSON

Este Agente de Internet de las Cosas es un puente que se puede utilizar para comunicar dispositivos utilizando un protocolo JSON simple y NGSI Context Brokers (como Orion). Este protocolo se basa en simples pares de codificación de objetos JSON de un solo nivel (atributo, valor). Este protocolo tiene como objetivo la simplicidad y la facilidad de uso, para aquellos escenarios donde no hay restricciones de recursos duros.

LIGHTWEIGHT M2M

Este Agente de Internet de las cosas es un puente que se puede utilizar para comunicar dispositivos utilizando el protocolo OMA Lightweight M2M y NGSI Context Brokers (como Orion). Lighteweight M2M es un protocolo liviano destinado a dispositivos y comunicaciones restringidos donde el ancho de banda y la memoria del dispositivo pueden ser recursos limitados.

NODE-LIB

Este proyecto tiene como objetivo proporcionar un módulo Node.js para permitir a los desarrolladores de IoT Agent crear agentes personalizados para sus dispositivos que puedan conectarse fácilmente a NGSI Context Brokers (como Orion).

OPC UA

El agente F4I IDAS OPC UA es un componente de código abierto destinado a permitir la captura de datos de dispositivos OPC UA en el taller y proporcionarlos a los niveles superiores de un sistema basado en FIWARE. Por lo tanto, el enfoque principal de este componente está en la comunicación desde dispositivos de campo que implementan un servidor OPC UA a FIWARE, permitiendo la comunicación al FIWARE Orion Context Broker.

SIGFOX

Este Agente de IoT está diseñado para ser un puente entre el protocolo de devolución de llamada de Sigfox y el protocolo OMA NGSI utilizado por Orion Context Broker, así como por otros componentes del ecosistema FIWARE.

Para cada dispositivo, el backend de Sigfox puede proporcionar un mecanismo de devolución de llamada que se puede utilizar para enviar dos tipos de información:

Atributos definidos por el backend de Sigfox (incluyendo id, marca de tiempo, etc.).
Un formato de datos libre, cuya estructura se puede definir en el tipo de dispositivo.

ULTRALIGHT

Este agente de Internet de las cosas es un puente que se puede utilizar para comunicar dispositivos utilizando el protocolo Ultralight 2.0 y los corredores de contexto NGSI (como Orion). Ultralight 2.0 es un protocolo ligero basado en texto dirigido a dispositivos y comunicaciones restringidos donde el ancho de banda y la memoria del dispositivo pueden ser recursos limitados.

PROCESSING

Documentación oficial disponible en:
https://github.com/FIWARE/catalogue/tree/master/processing

COSMOS

El Cosmos Big Data Analysis GE es un conjunto de herramientas que ayudan a realizar las tareas de Streaming y procesamiento por lotes sobre datos contextuales. Estas herramientas son, Apache Flink Processing Engine y Apache Spark Processing Engine.

FOGFLOW

FogFlow es un marco de computación de borde de IOT para orquestar flujos de procesamiento dinámico sobre la nubes y bordes. Puede componer de forma dinámica y automática múltiples tareas de procesamiento de datos basadas en NGSI para formar servicios de IOT de alto nivel y, a continuación, orquestar y optimizar el despliegue de esos servicios dentro de un entorno de nube compartido, en lo que se refiere a la disponibilidad, la localidad y la movilidad de los dispositivos de IOT.

KNOWAGE

Knowage es una suite profesional de código abierto para el análisis moderno de negocios sobre fuentes tradicionales y grandes sistemas de datos.

Knowage se compone de varios módulos, cada uno concebido para un dominio analítico específico. Pueden utilizarse individualmente o combinarse entre sí para garantizar la cobertura total de las necesidades del usuario:

- Big Data - para analizar datos almacenados en grandes clusters de datos o bases de datos NoSQL.
- Inteligencia Inteligente - la inteligencia de negocio habitual sobre datos estructurados, pero más orientada a las capacidades de autoservicio y a la creación ágil de prototipos.
- Enterprise Reporting - para producir y distribuir informes estáticos.
- Location Intelligence - para relacionar datos de negocio con información espacial o geográfica.
- Gestión del Desempeño - para administrar KPIs y organizar scorecards.
- Análisis Predictivo - para análisis más avanzados
- Embedded Intelligence - para vincular el conocimiento con soluciones externas proporcionadas por el cliente o terceros.

KURENTO

Stream Oriented GE es un framework de desarrollo que proporciona una capa de abstracción para las capacidades multimedia, permitiendo a los desarrolladores no expertos incluir componentes de medios interactivos en sus aplicaciones. La API abierta está en el corazón de este habilitador: una API similar a REST, basada en JSON RPC 2.0, que expone una caja de herramientas de elementos de medios que se pueden encadenar para crear complejas tuberías de procesamiento de medios. The Stream Oriented GE proporciona varias implementaciones de cliente de la API abierta. El cliente Java permite a los desarrolladores incluir capacidades de medios para aplicaciones Java o JEE. Un cliente JavaScript también está listo para ser usado con Node.js o directamente en aplicaciones de navegador. Gracias a ello, Stream Oriented GE proporciona a los desarrolladores un conjunto de robustas capacidades de comunicación multimedia interoperables de extremo a extremo para hacer frente a la complejidad de las tareas de transporte, codificación/decodificación, procesamiento y renderizado de una manera fácil y eficiente.

PERSEO

El Perseo Context-aware Complex Event Processing (Context-aware CEP) GE es un módulo que escucha eventos a partir de datos contextuales (procedentes de Orion Context Broker o de cualquier otro sistema o servicio compatible con NGSI) en tiempo real, y genera una visión inmediata, permitiendo así una respuesta instantánea a condiciones cambiantes.

WIRECLOUD

Wirecloud es una herramienta de web mashup (aplicación web híbrida) diseñada para facilitar el desarrollo de cuadros de mando operativos. Esto permite a los usuarios finales crear fácilmente aplicaciones web y cuadros de mando sin conocimientos de programación y visualizar datos de interés y controlar su entorno.

Los mashups de aplicaciones Web integran datos heterogéneos, lógica de aplicaciones y componentes de interfaz de usuario (widgets) procedentes de la Web para crear nuevas aplicaciones compuestas coherentes y de valor añadido. Su objetivo es aprovechar la "larga cola" de la Web de Servicios (también conocida como la Web Programable) explotando el desarrollo rápido, el bricolaje y la capacidad de compartir.

ROBOTIC

Documentación oficial disponible en:
https://github.com/FIWARE/catalogue/tree/master/robotics

eFAST-RTPS

eProsima Fast RTPS es una implementación en C++ del protocolo RTPS (Real Time Publish Subscribe), que proporciona comunicaciones entre el editor y el suscriptor sobre transportes poco fiables como UDP, según lo definido y mantenido por el consorcio Object Management Group (OMG). RTPS es también el protocolo de interoperabilidad por cable definido para el estándar Data Distribution Service (DDS), también por la OMG. eProsima Fast RTPS tiene el beneficio de ser independiente y actualizado, ya que la mayoría de las soluciones de los proveedores implementan RTPS como una herramienta para implementar DDS o utilizan versiones anteriores de la especificación.

eMICRO-XRCE-DDS

eProsima Micro XRCE-DDS es un middleware orientado a la IO basado en un patrón de mensajería de publicación-suscripción. El Micro XRCE-DDS sigue una arquitectura cliente-servidor donde los dispositivos de bajo recurso (Micro XRCE-DDS Clients) están conectados a un servidor (Micro XRCE-DDS Agent).

Por un lado, el Micro XRCE-DDS Client (librería C) se centra en abordar los retos de los entornos con recursos limitados, como un microcontrolador. Por esta razón, esta librería está diseñada para ofrecer una implementación completamente libre de memoria dinámica y un uso de memoria realmente bajo (~2.5 KB de uso de pila para una simple aplicación de suscripción de editor).

Por otro lado, el Micro XRCE-DDS Agent (aplicación C++) es una aplicación autónoma que se encarga de proporcionar presencia a los Clientes en el espacio de datos global DDS (Data Distribution Service). Esta aplicación multiplataforma (Linux y Windows) ofrece a los Clientes un acceso de alto rendimiento al espacio DDS gracias al framework Fast RTPS. Micro XRCE-DDS es una implementación autónoma C/C++ del OMG (Object Management Group) DDS-XRCE (DDS for eXtremely Resource Constrained Environments Specification).

FIROS

FIROS es una herramienta que ayuda a conectar robots a la nube. Para ello utiliza el Sistema Operativo del Robot (ROS) y el FIWARE Context Broker como una forma de publicar y escuchar los datos del robot. FIROS trabaja como intérprete entre el campo de la robótica y el mundo de la nube, transformando los mensajes ROS en NGSI para publicarlos en la nube, y viceversa.

SECURITY

Documentación oficial disponible en:
https://github.com/FIWARE/catalogue/tree/master/security

AUTHZFORCE

Se obtiene la implementación de referencia del Authorization PDP Generic Enabler (anteriormente llamado Access Control GE). De hecho, según lo dispuesto en la especificación de GE, esta implementación proporciona una API para obtener decisiones de autorización basadas en políticas de autorización y solicitudes de autorización de las PEP. La API sigue el estilo de la arquitectura REST y cumple con XACML v3.0. XACML (eXtensible Access Control Markup Language) es un estándar OASIS para el formato de política de autorización y la lógica de evaluación, así como para el formato de solicitud/respuesta de decisión de autorización. Los términos PDP (Policy Decision Point) y PEP (Policy Enforcement Point) están definidos en el estándar XACML. Este GEri juega el papel de un PDP.

Para cumplir con la arquitectura XACML, es posible que necesite un PEP (Policy Enforcement Point) para proteger su aplicación, que no se proporciona aquí. Para las APIs de REST, le recomendamos que utilice el PEP Proxy de UPM disponible en el catálogo.

KEYROCK

La gestión de la identidad cubre una serie de aspectos relacionados con el acceso de los usuarios a redes, servicios y aplicaciones, incluida la autenticación segura y privada de los usuarios a dispositivos, redes y servicios, la gestión de autorizaciones y confianza, la gestión de perfiles de usuario, la disposición de los datos personales para preservar la privacidad, el Single Sign-On (SSO) a los dominios de servicio y la Identity Federation a las aplicaciones. El Gestor de identidades es el componente central que proporciona un puente entre los sistemas IdM a nivel de conectividad y de aplicación. Además, la gestión de identidades se utiliza para autorizar a los servicios extranjeros a acceder a los datos personales almacenados en un entorno seguro. Por lo general, el propietario de los datos debe dar su consentimiento para acceder a ellos; el procedimiento de consentimiento también implica cierta autenticación del usuario.

WILMA

Obtendrá la implementación de referencia de PEP Proxy Generic Enabler. Gracias a este componente y junto con los GEs de Gestión de Identidades y Autorización de PDP, podrá añadir autenticación y seguridad de autorización a sus aplicaciones backend. Por lo tanto, sólo los usuarios de FIWARE podrán acceder a sus servicios de GEs o REST. Pero también podrá gestionar permisos y políticas específicas para sus recursos, permitiendo diferentes niveles de acceso a sus usuarios.

THIRD-PARTY

Documentación oficial disponible en:
https://github.com/FIWARE/catalogue/tree/master/third-party

DOMIBUS

Domibus es un ejemplo de implementación de un Punto de Acceso CEF eDelivery. CEF eDelivery ayuda a los usuarios a intercambiar datos y documentos electrónicos entre sí de forma fiable y fiable. La solución CEF eDelivery se basa en un modelo distribuido llamado "modelo de 4 esquinas". En este modelo, los sistemas backend de los usuarios no intercambian datos directamente entre sí, sino que lo hacen a través de puntos de acceso. Estos puntos de acceso cumplen las mismas especificaciones técnicas y, por lo tanto, son capaces de comunicarse entre sí. Como resultado, los usuarios que adoptan CEF eDelivery pueden intercambiar datos de forma fácil y segura, incluso si sus sistemas de TI se han desarrollado de forma independiente.

Implementación Básica de Componentes en un Ejemplo Localhost.

Se implementó un ejemplo básico utilizando los principales componentes de FIWARE. Lo primero que se debe entender es el funcionamiento de DOCKER y para ello se facilita un tutorial realizado por Felipe Quezada en donde se explican todas las funcionalidades y características de DOCKER, este tutorial se encuentra disponible en el repositorio público de github (**https://github.com/MRROOX/docker-la-guia-ici**). Además se utiliza DOCKER-COMPOSE como orquestador de los contenedores.

Una vez comprendidas las características y el funcionamiento de DOCKER se procede a realizar la implementación básica utilizando FIWARE.

Diagrama de Arquitectura

El proyecto completo y también parte de la documentación se encuentra disponible en el siguiente repositorio del proyecto: **https://github.com/MRROOX/gpi-fiware.git**

A continuación se presenta el diagrama de arquitectura en donde se puede apreciar la los componentes principales de implementación. Quantum Leap es un servicio para persistir datos en series de datos, Cygnus para persistir datos en bases de datos relacionales como Mysql, Postgres, Orion Context Broker es el broker de contextos de información y es la pieza fundamental de una solución FIWARE que persiste los datos de forma temporal en una base de datos NoSql en este caso MongoDB, Iot Agent es un servicio para gestionar dispositivos IOT, los dispositivos IOT en este caso una raspberry pi en la cual se encuentra un sensor DHT22 de temperatura y humedad y por último Grafana es una herramienta en la cual se pueden crear distintas vistas para visualización de datos en series de tiempo.

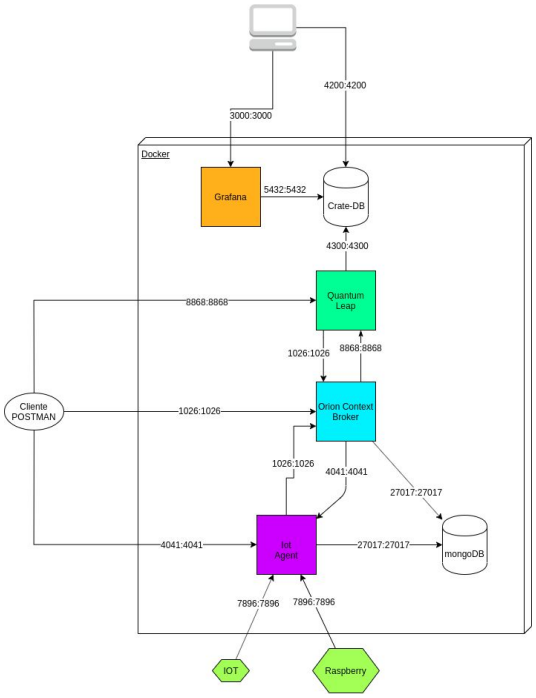

Fig. 1. Arquitectura de Ejemplo Localhost

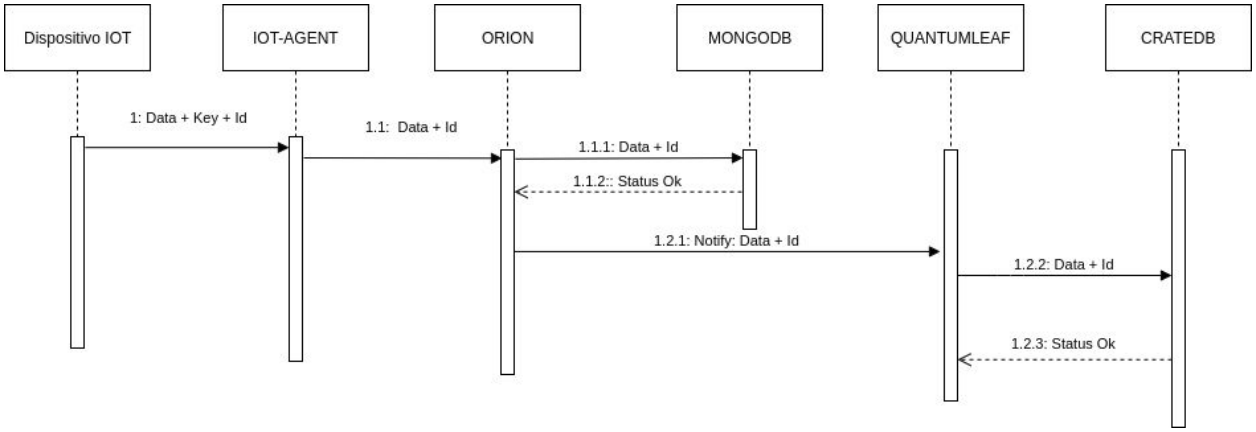

Fig 2. Diagrama de Secuencia

En la Fig 2 se puede apreciar como los datos son obtenidos por el Dispositivo IOT y luego son enviados al servicios de IOT-AGENT, este valída la Key y luego envía los datos al servicio de ORION, éste servicio actualiza los datos datos en MONGODB y luego notifica al servicio de QUANTUMLEAF éste servicio persiste los datos en la base de datos CRATEDB. Los datos en CRATEDB serán utilizados por GRAFANA para su posterior visualización y utilización.

Configuración Servicios, Redes y Volumenes.

Si se comprende el funcionamiento de DOCKER y DOCKER COMPOSE se puede entender a mayor cabalidad las configuraciones es por eso a continuación de describen los servicios las redes y Volúmenes empleados para realizar este ejemplo.

Servicios

Orion Context Broker:

```
# Orion Context Broker
orion:
  image: fiware/orion
  hostname: orion
  container_name: fiware-orion
  depends_on:
    - mongo-db
  networks:
    - fiware_orion_net
  ports:
    - "1026:1026"
  command: -dbhost mongo-db -logLevel DEBUG -noCache
  healthcheck:
    test: curl --fail -s http://orion:1026/version || exit 1
```

Iot Agent:

```
# Angente IOT intermediario entre Orion y Dispositivo IOT
iot-agent:
  image: fiware/iotagent-ul
  hostname: iot-agent
  container_name: fiware-iot-agent
  depends_on:
    - mongo-db
```

```
    ports:
      - "4041:4041"
      - "7896:7896"
    environment:
      - "IOTA_CB_HOST=orion"
      - "IOTA_CB_PORT=1026"
      - "IOTA_NORTH_PORT=4041"
      - "IOTA_REGISTRY_TYPE=mongodb"
      - "IOTA_MONGO_HOST=mongo-db"
      - "IOTA_MONGO_PORT=27017"
      - "IOTA_MONGO_DB=iotagentul"
      - "IOTA_HTTP_PORT=7896"
      - "IOTA_PROVIDER_URL=http://iot-agent:4041"
    networks:
      - fiware_orion_net
```

Quantum leap:

```
# Quantum Leap is persisting Short Term History to Crate-DB
  quantumleap:
    image: smartsdk/quantumleap:0.7.5
    container_name: fiware-quantumleap
    ports:
      - "8668:8668"
    networks:
      - fiware_grafana_net
      - fiware_orion_net
    depends_on:
      - crate-db
      - orion
      - mongo-db
    environment:
      - "CRATE_HOST=http://crate-db"
    healthcheck:
      test: curl --fail -s http://localhost:8668/v2/version || exit 1
```

Grafana:

```
# Monitorización utilizando Grafana
  grafana:
    image: grafana/grafana:master
    depends_on:
        - crate-db
    ports:
        - "3000:3000"
    environment:
       # -
GF_INSTALL_PLUGINS=https://github.com/raintank/crate-datasource/archive/
master.zip;crate-datasource,grafana-clock-panel,grafana-worldmap-panel
       #- GF_INSTALL_PLUGINS=grafana-clock-panel,grafana-worldmap-panel
      -
GF_INSTALL_PLUGINS=https://github.com/orchestracities/grafana-map-plugin
/archive/master.zip;grafana-map-plugin,grafana-clock-panel,grafana-world
map-panel
    networks:
      - fiware_grafana_net
    volumes:
      - type: volume
        source: vol_grafana-conf
        target: /var/lib/grafana
```

MongoDB:

```
# MONGO DB solo para servicio de Orion
  mongo-db:
    image: mongo:3.6
    hostname: mongo-db
    container_name: db-mongo
    ports:
      - "27017:27017"
```

```
    command: --bind_ip_all --smallfiles
    networks:
      - fiware_orion_net
    volumes:
      - type: volume
        source: vol_mongo-db
        target: /data
```

CreateDB:

```
 # Crate para utilizar Grafana
  crate-db:
    image: crate:3.3.2
    hostname: crate-db
    container_name: db-crate
    ports:
      # Admin UI
      - "4200:4200"
      # Transport protocol
      - "4300:4300"
   # command: crate -Clicense.enterprise=false
-Cauth.host_based.enabled=false  -Ccluster.name=democluster
-Chttp.cors.enabled=true -Chttp.cors.allow-origin="*"
```

Redes

```
# Redes
networks:
  fiware_orion_net:
  fiware_cygnus_net:
  fiware_grafana_net:
```

Volúmenes

```
# Volúmenes para persistir la data.
volumes:
  vol_portainer_data:
  vol_mongo-db:
  vol_mysql-db:
  vol_crate-db:
  vol_grafana-conf:
```

Configuración Dispositivo IOT, Raspberry Pi 3 modelo b y Sensor DHT22

Se realiza la configuración de la raspberry y el sensor dht22 siguiendo el siguiente diagrama de configuracion y luego siguiente la intalacion de las dependencias siguiedo la siguiente guia disponible en: https://github.com/MRROOX/dht22-raspberry-pi-model-b esta informacion tambien se encuentra disponible dentro del repositorio de trabajo.

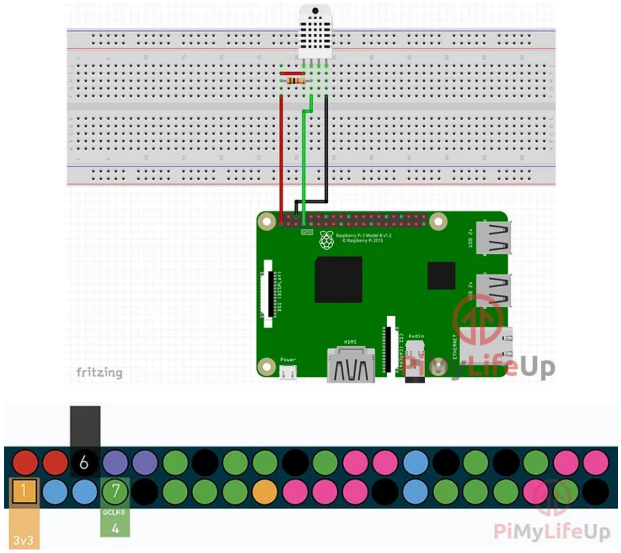

Fig 3. Diagrama de Configuración de Sensor Dht22.

Dentro del proyecto se dispone de un script en python que se encuentra en el siguiente directorio: **gpi-fiware/TUTORIALES/LOCALHOST/IOT/FIWARE-DHT22/**dh22-fiware+config.py

En este script en python se implementa la obtención de datos desde el sensor DHT22 y posteriormente son enviados al servicio de IOT-AGENT mediante el protocolo PDI-IoTA-UltraLight.

En el archivo de configuración se define el token de acceso y el **id** del dispositivo iot.

Esto se puede apreciar con mayor detalle en la documentación definida en el repositorio de trabajo.

Entidad de calidad del aire en Orion Context Broker.

En primer lugar se ejecuta el docker-compose.yml por medio del comando “docker-compose up -d”, por el cual inicialmente se van a descargar y levantar todas las dependencias, servicios y comandos indicados en el archivo.

De ahora en adelante se utiliza Postman para realizar las peticiones correspondientes. Para crear la entidad que monitoree la humedad y temperatura, se enviará la siguiente petición:

```
POST http://localhost:1026/v2/entities
Header: Content-Type: "application/json"
        fiware-service: openiot
        fiware-servicepath: /

En el body, utilizamos el contenido de dht22.json, que se ve a continuación:

{
   "id":"DHT22002",
   "type":"DHT22",
   "dateObserved":{
      "type":"DateTime",
      "value":"2019-09-30T18:00:00-05:00"
   },
   "address": {
      "type": "StructuredValue",
      "value": {
         "addressCountry": "CL",
```

```
            "addressLocality": "Temuco",
            "streetAddress": "UFRO"
        }
    },
    "location": {
        "value": {
            "type": "Point",
            "coordinates": [-38.751398,-72.605476],
            "latitude": -38.751398,
            "longitude": -72.605476
        },
        "type": "geo:json"
    },
    "temperature":{
        "type":"Number",
        "value":"0"
    },
    "relativeHumidity":{
        "type":"Number",
        "value":"0"
    }
}
```

En este caso se utiliza una máquina virtual, por lo que en Postman se utiliza la dirección IP de la máquina; en caso de utilizar la misma máquina, se debe utilizar Localhost. En las figuras 4 y 5 se ve el ejemplo de cómo se configuran los Headers y el Body para la petición.

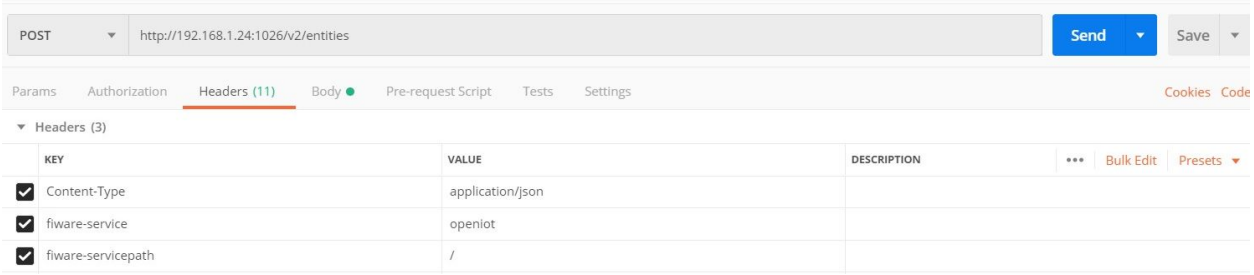

Fig 4. Configuración de Headers en Postman

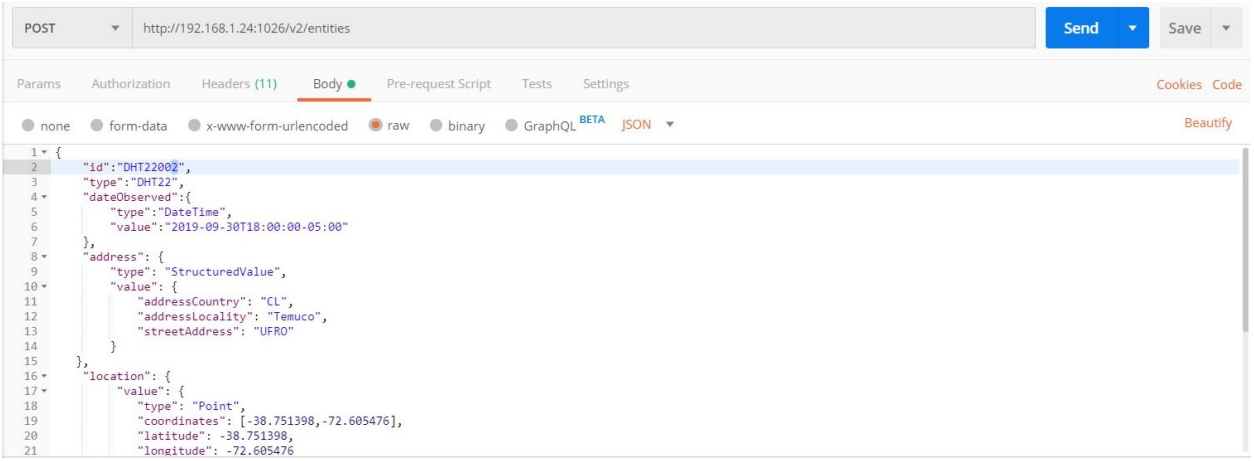

Fig 5. Configuración de Body en Postman

Suscripción a entidad por medio de Quantumleap.

La suscripción a la entidad anteriormente creada servirá para recibir una notificación cada vez que dicha entidad sea actualizada, esta notificación la recibirá Quantumleap y este se encargará de persistir los nuevos datos en la base de datos CRATE DB. La petición para suscribirse se encuentra a continuación:

POST http://localhost:1026/v2/subscriptions/
Header: Content-Type: "application/json"
fiware-service: openiot
fiware-servicepath: /

En body, utilizamos el contenido de dht22-subcription.json, que se ve a continuación:

```
{
   "description": "Notify QuantumLeap on luminosity changes on any DHT22",
   "subject": {
     "entities": [
       {
         "idPattern": "DHT22.*",
         "type": "DHT22"
       }
     ],
     "condition": {
       "attrs": [
         "temperature",
         "relativeHumidity"
       ]
     }
```

```
  },
  "notification": {
    "attrs": [
      "id",
      "temperature",
      "relativeHumidity",
      "dateObserved",
      "location"
    ],
    "http": {
      "url": "http://quantumleap:8668/v2/notify"
    },
      "metadata": ["dateCreated", "dateModified"]
  },
  "throttling": 1
}
```

En Postman la parte de los Headers se mantiene idéntico, lo que cambia será el Body que quedará como se ve en la imagen:

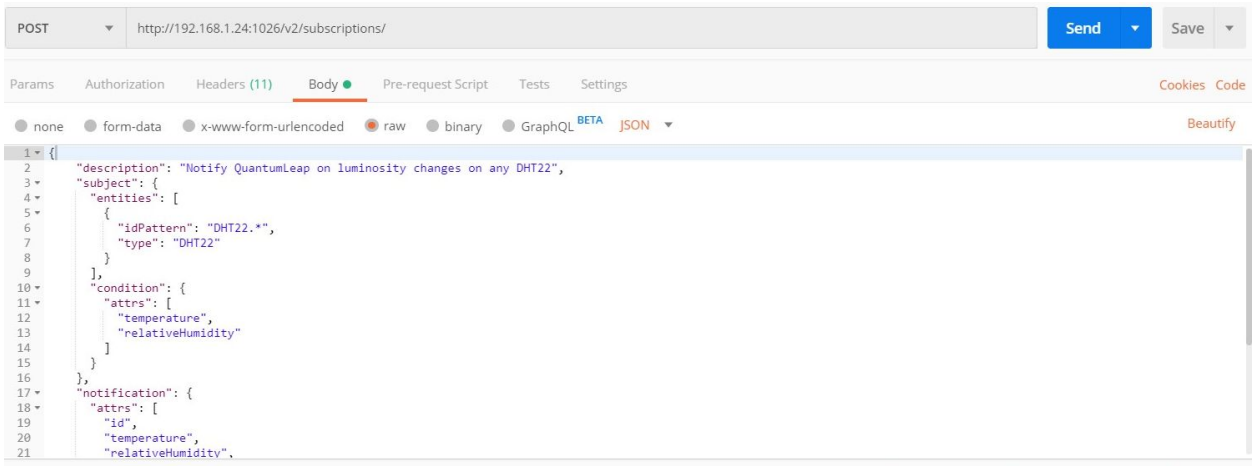

Fig 6. Contenido del Body en Postman.

Configuración de CRATE DB.

Quantumleap hará que los datos se persistan en la base de datos CRATE DB. Para verificar que todo se encuentre funcionando como debe, se debe ingresar a la url: localhost:4200. Se puede acceder a la base de datos llamada (de forma automática) mtopeniot y se puede hacer la consulta que se ve en la figura 8 para verificar que funciona correctamente.

```
select * from mtopeniot.etdht22 limit 100;
```

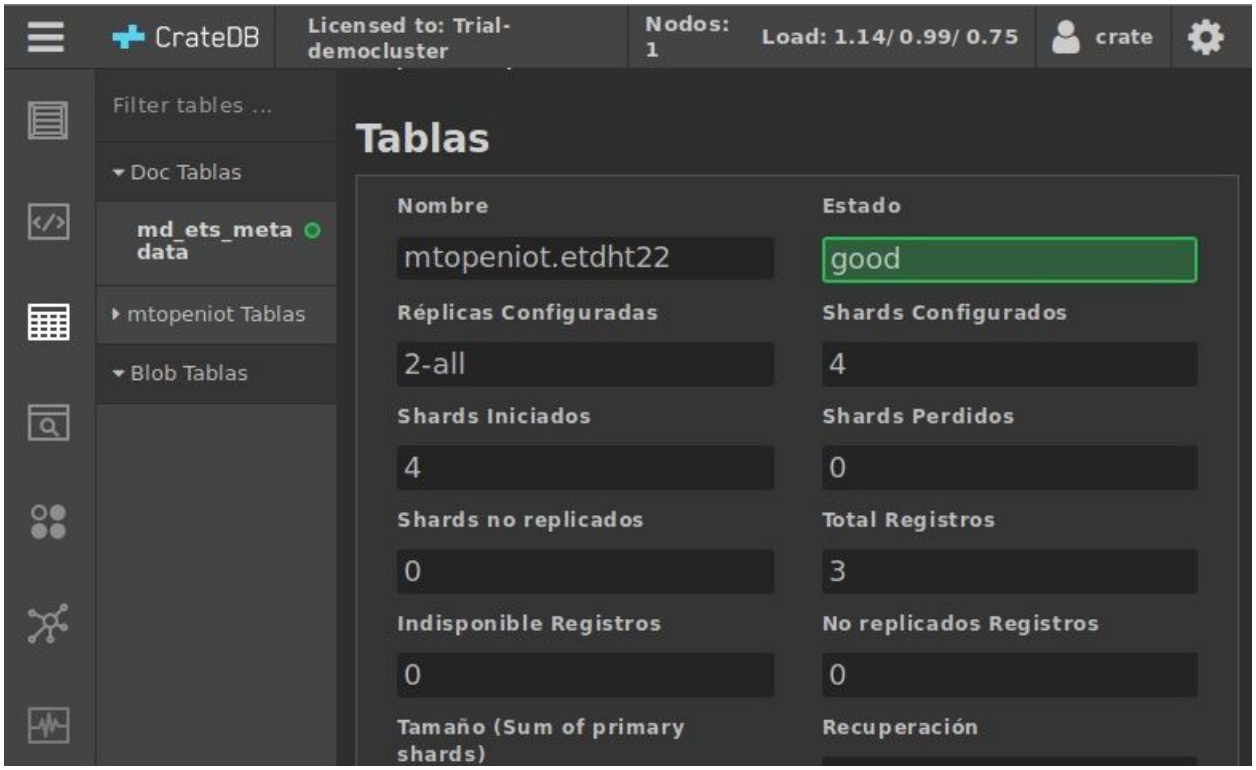

Fig 7. Panel de administración CrateDB.

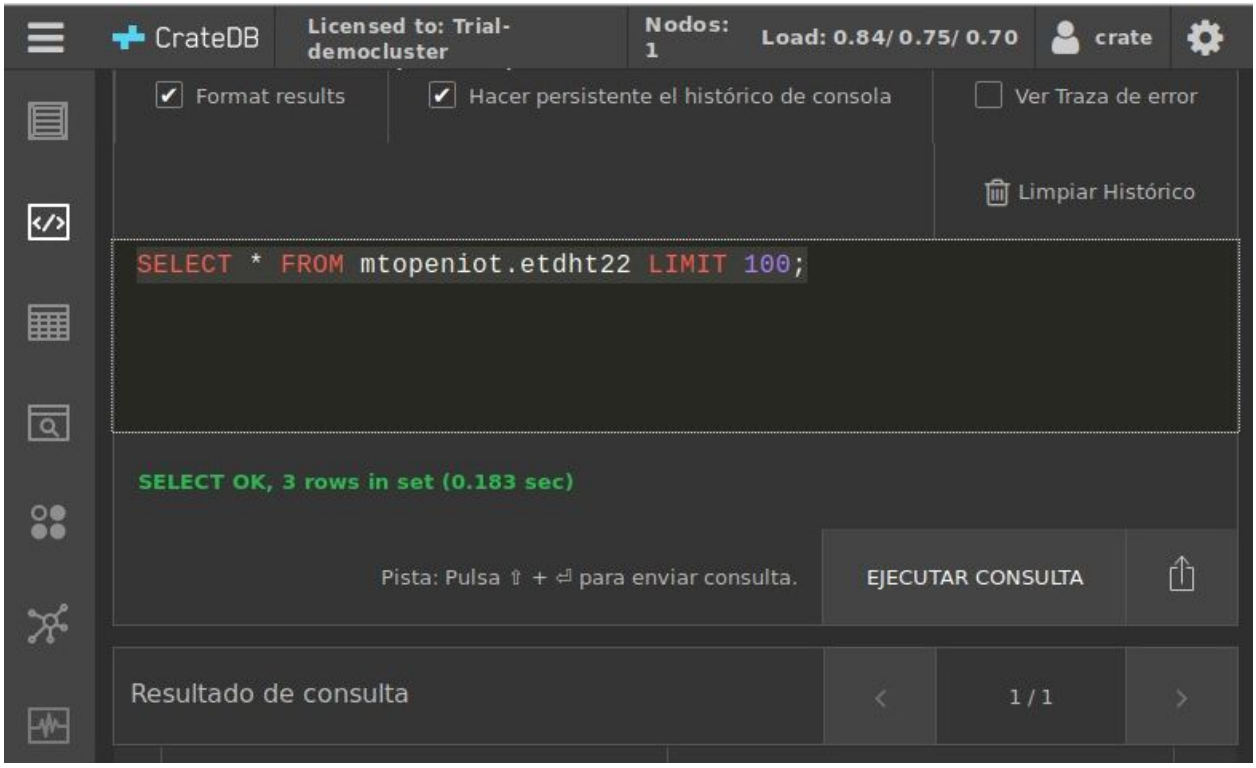

Fig 8. Query de prueba en consola de CrateDB.

Visualización web de datos con Grafana.

Para acceder a Grafana se hace por medio de localhost:3000. Lo primero que se debe hacer es acceder a la página con las credenciales por defecto (user: admin, password: admin), para luego cambiar de contraseña y acceder al panel principal de Grafana. Luego, se debe conectar a una fuente de datos (data source), por lo que se accede a esa opción, se elige la configuración de PostgreSQL y se rellenan los campos de la forma que se indica a continuación:

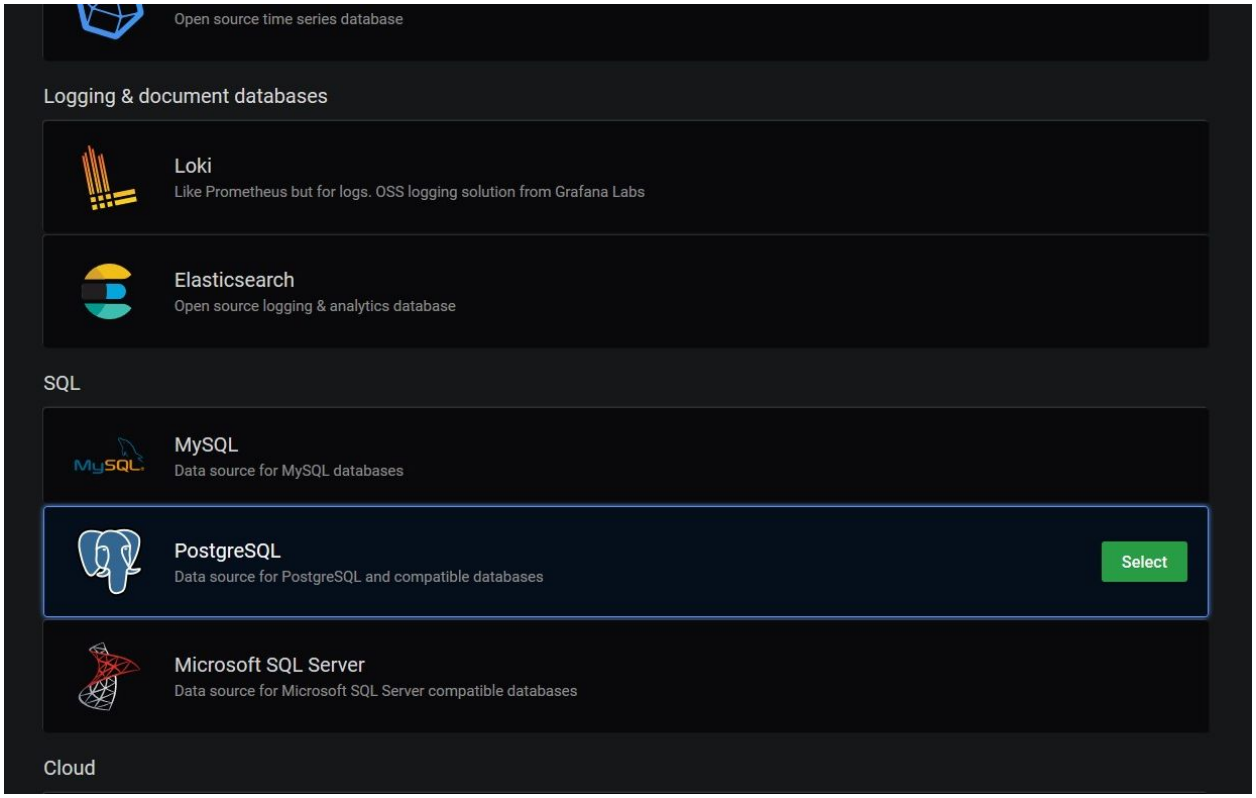

Fig 9. Configuración de fuente de datos en Grafana.

Name	CrateDB
Host	crate-db:5432
Database	mtopeniot
User	crate
SSL Mode	disable

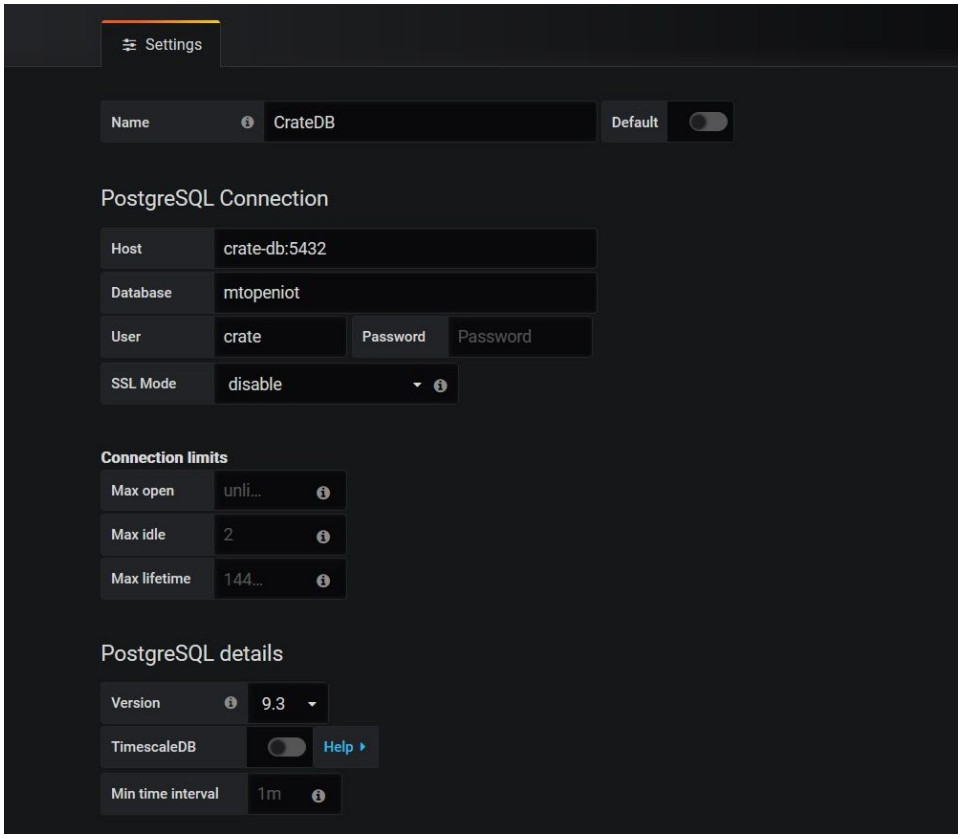

Fig 10. Conexión a la base de datos CrateDB en Grafana.

Se guardan los cambios y se prueba la conexión con la base de datos, posteriormente se crea una nueva Dashboard donde se visualizarán los datos de humedad y temperatura por medio de una query que se indica en las siguientes imágenes:

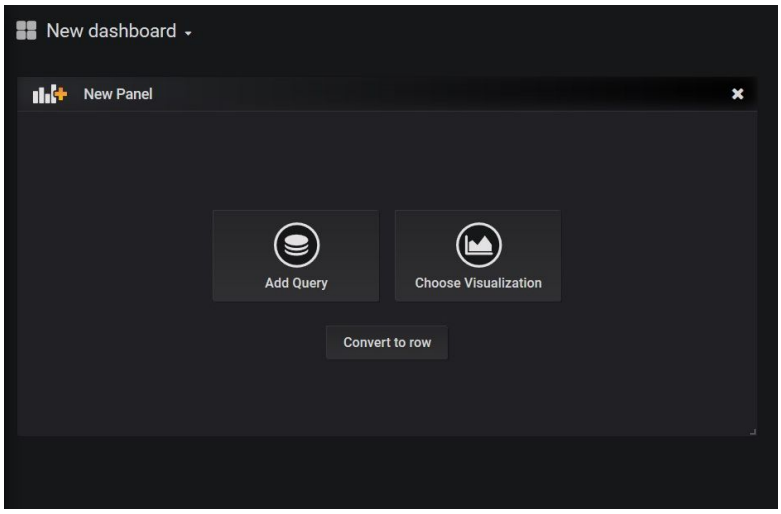

Fig 11. Panel para agregar un nuevo dashboard.

La query que se muestra es la siguiente:

FROM etdht22 Time column time_index Metric column entity_id SELECT Column: temperature Column: relativehumidity

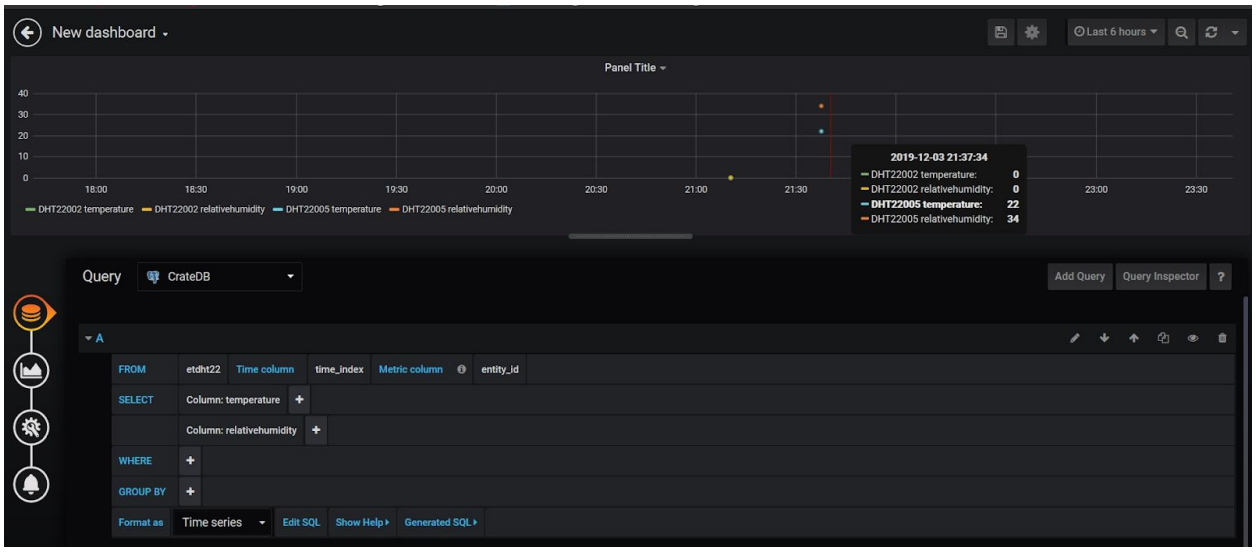

Fig 12. Visualización de los datos por medio de la Query.

Los cambios en los valores de humedad y temperatura se visualizarán en tiempo real en el Dashboard recientemente configurado, para poder hacer actualizaciones en los datos, se utiliza Postman con los mismos Headers de antes y con los siguientes datos en el Body (se usa la petición PATCH):

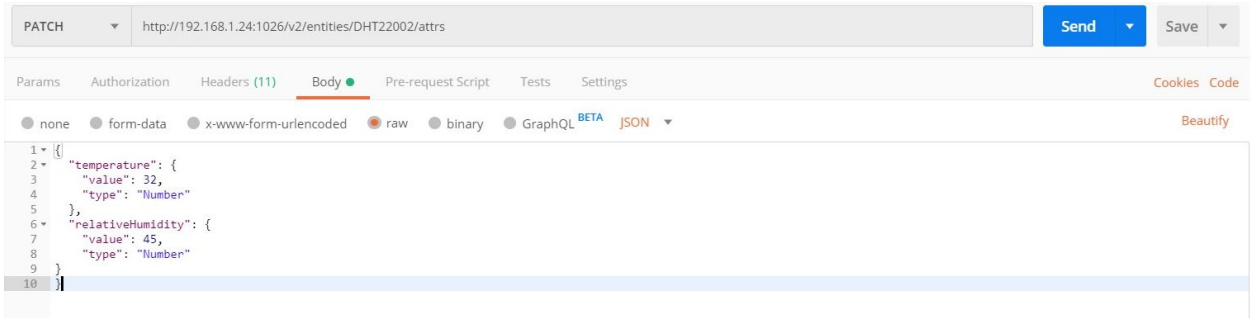

Fig 13. Actualización de datos mediante Postman.

Finalmente los cambios se visualizarán al instante en el Dashboard.

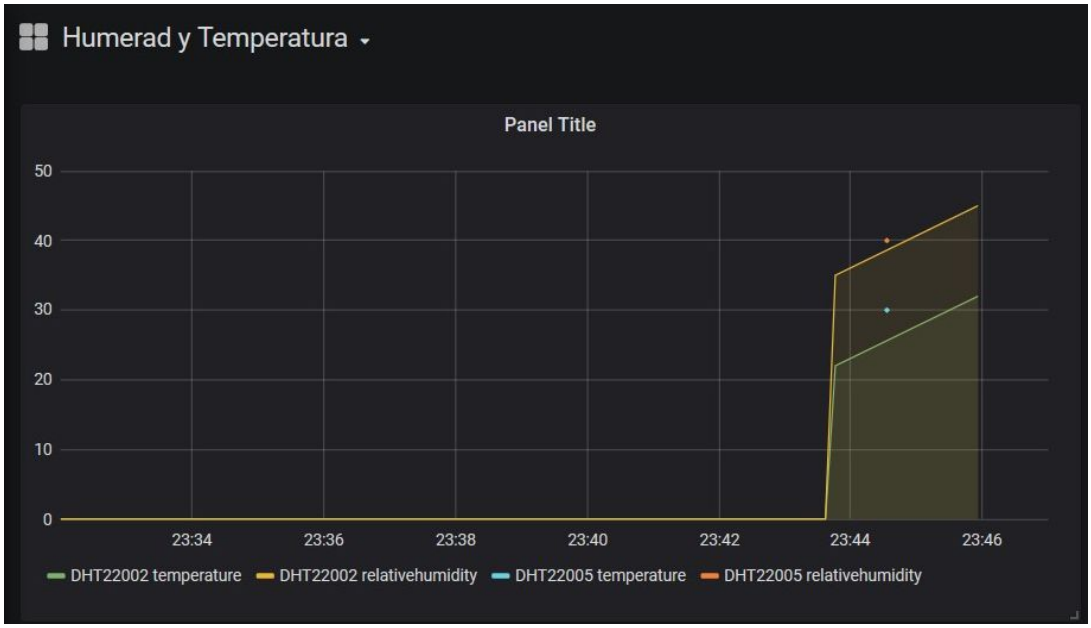

Fig 14. Visualización de datos en dashboard de Grafana.

Configuración IOT-Agent.

El IOT-Agent sirve de intermediario entre Orion Context Broker y un Dispositivo IOT. Cada dispositivo puede tener sus propios protocolos de comunicación y en este caso se utilizará uno llamado “PDI-IoTA-UltraLight”.

Configuración de SERVICE.

```
POST http://localhost:4041/iot/services
Header: Content-type: application/json
        Fiware-service: openiot
        fiware-servicepath: /

En el body se agrea el json iot-agent.service.json que se muestra a continuación:
{
   "services": [
     {
       "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
       "cbroker":     "http://orion:1026",
```

```
      "entity_type": "DHT22",
      "resource":    "/iot/d"
    }
  ]
 }
```

Configuración de DEVICE.

POST http://localhost:4041/iot/devices

Header: Content-type: application/json
Fiware-service: openiot
Fiware-servicepath: /

En el body se agrega el json iot-agent.device.json que se muestra a continuación:

```
{

   "devices": [
      {
        "device_id": "DHT22003",
        "entity_name": "DHT22003",
        "entity_type": "DHT22",
        "protocol": "PDI-IoTA-UltraLight",
        "transport": "HTTP",

        "attributes": [
          {"object_id": "t", "name": "temperature", "type":"Number"},
          {"object_id": "h", "name": "relativeHumidity", "type":"Number"}
         ]
      }
    ]
}
```

Adicionalmente se deben agregar los siguientes dos parámetros a la petición:

k=4jggokgpepnvsb2uv4s40d59ov
i=DHT22005

Finalmente, para actualizar datos del dispositivo IOT, se hace la siguiente petición por medio de Postman (notar que el Header ha cambiado a texto plano y el Body pasó a ser de una sola línea):

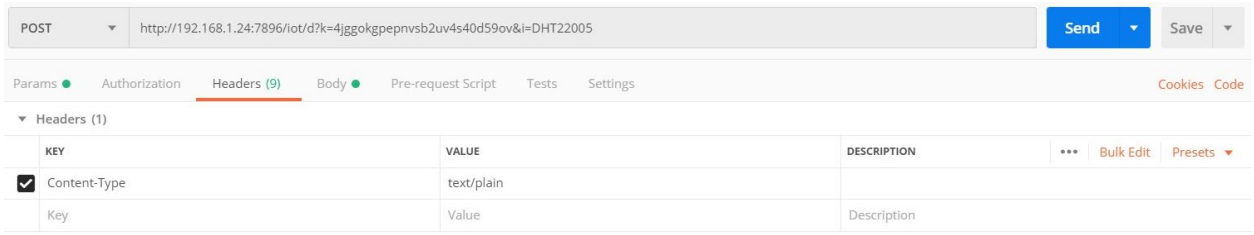

Fig 15. Configuración de Header en Postman.

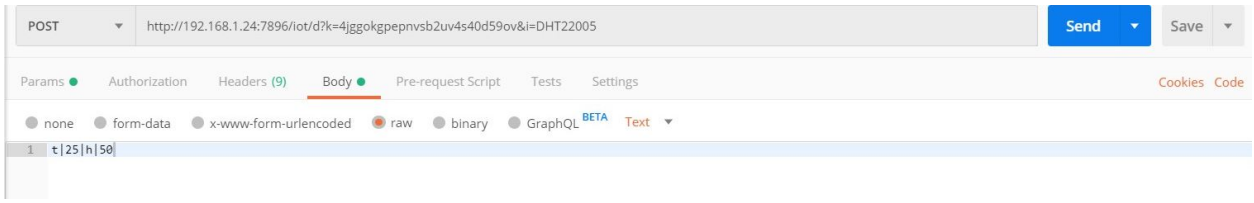

Fig 16. Contenido del Body en Postman.

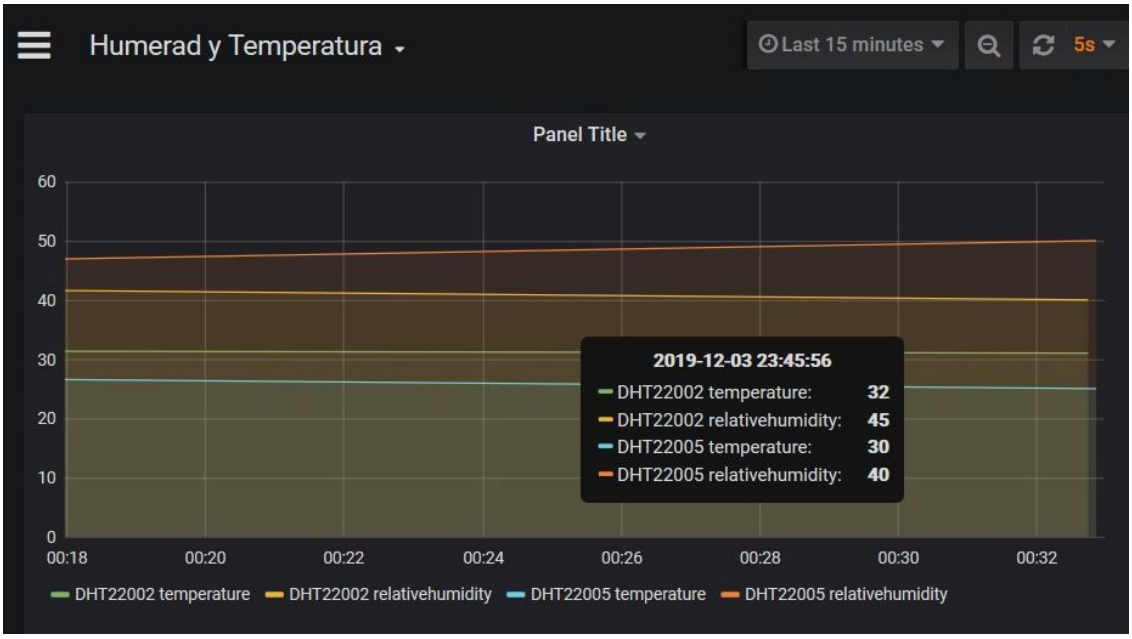

Fig 17. Visualización de los dos dispositivos en Grafana.

Finalmente se puede notar en el dashboard principal de Grafana que se actualizan los datos en tiempo real tanto de la forma directa (DHT22002) como por medio del Agente IOT (DHT22005).

Implementación Sistema ALPR para reconocimiento de Patentes Vehiculares.

Los sistemas ALPR (Reconocimiento Automático de Placa de Matrículas) son una tecnología que utiliza reconocimiento óptico de caracteres (OCR) para leer automáticamente los caracteres de la placa de matrícula de automóviles.

Para este proyecto se ha utilizado OpenALPR que se define como una biblioteca de código abierto de Reconocimiento Automático de Matrículas escrita en C++ con cliente en C#, Java, Node.js, Go y Python. La biblioteca analiza imágenes y secuencias de vídeo para identificar matrículas. La salida es la representación de texto de cualquier carácter de la patente de un vehículo.

Para este proyecto se ha creado una imagen docker completamente nueva utilizando python3 ya que la versión de python2 acaba de ser deprecada. Esta imagen contiene todos los elementos básicos para el funcionamiento, tanto como OpenALPR, y las dependencias de python3 para ejecutar un servicio web para el reconocimiento de matrículas a partir de imágenes. Los datos generados son enviados a la plataforma de Fiware implementada en la primera iteración de este proyecto.

La imagen docker generada se ha añadido al archivo de docker-compose.yml para que pueda ser desplegada en conjunto con los otros servicios de del proyecto, con el nombre de **openalpr.**

Además se agrega un nuevo servicio llamado **Cygnus,** este servicio es utilizado para persistir los datos generados por el servicios de **openalpr** en un base de datos mongodb. Para que sea capaz de persistir los datos de un contexto, es necesario realizar la suscripción del cambio de estado de los datos en el servicio de **orion** para que notifique dichos cambios en a **Cygnus** y éste los persista en la base de datos. Esto se puede ver claramente el diagrama de secuencia de la Fig. 19 Diagrama de Secuencia + ALPR.

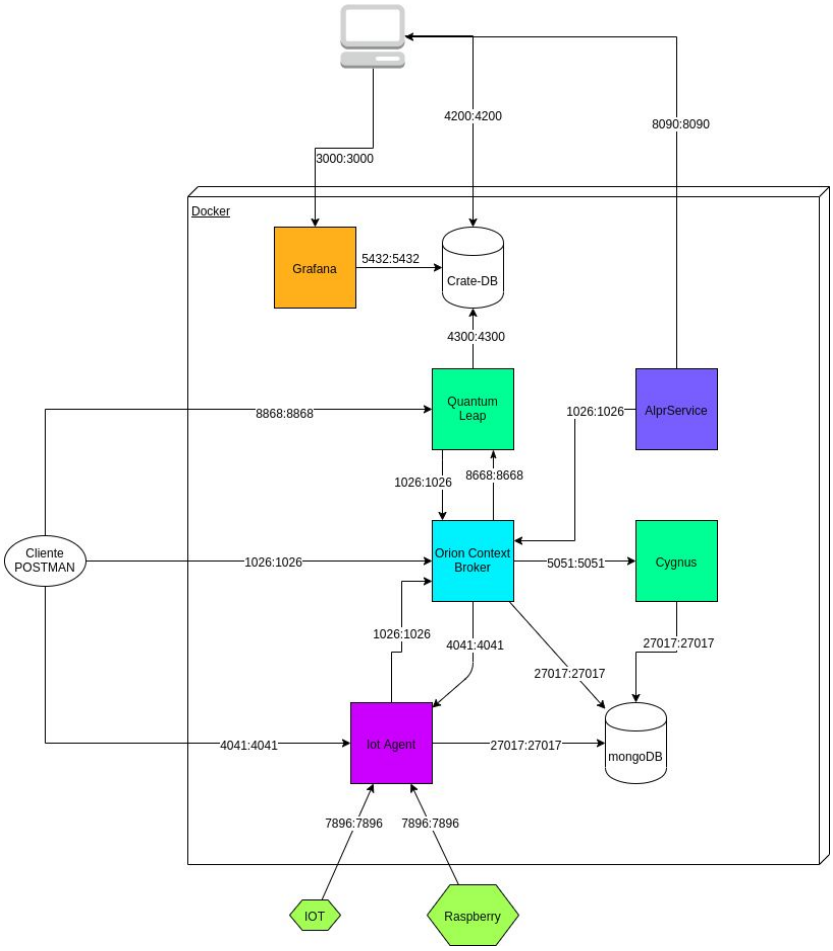

Fig 18. Diagrama de Arquitectura Sistema ALPR

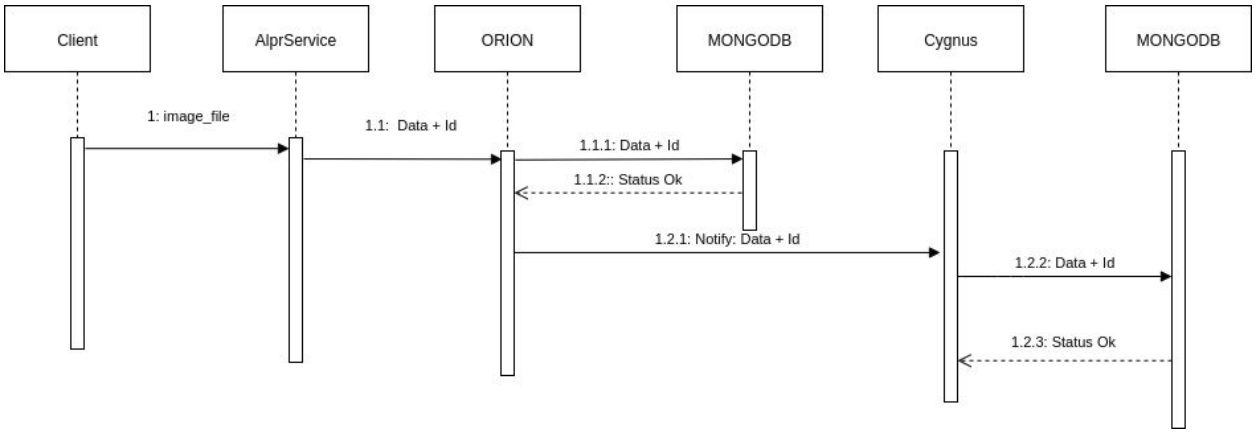

Fig 19. Diagrama de Secuencia + ALPR

En primer lugar, se deben levantar los servicios de Fiware disponibles en el repositorio (https://github.com/MRROOX/gpi-fiware/tree/master/TUTORIALES/LOCALHOST) utilizando el comando “docker-compose up -d”, donde aparece integrado el servicio Open ALPR con Fiware. Una vez realizada dicha acción, suponiendo que ya se hizo el ejemplo en LocalHost anterior, se deben realizar los pasos que se explicarán a continuación.

El primer paso consiste en hacer el registro de la entidad en el Orion Context Broker, recordando que es la piedra angular de Fiware. Para esto, se envía una petición por medio de Postman con el siguiente contenido:

```
POST http://localhost:1026/v2/entities
Header: Content-Type: "application/json"
        fiware-service: openalpr
        fiware-servicepath: /alpr

Body:

{
   "id":"ALPR0004",
   "type":"ALPR",
   "dateObserved":{
      "type":"DateTime",
      "value":"2019-09-30T18:00:00-05:00"
   },
   "address": {
      "type": "StructuredValue",
      "value": {
         "addressCountry": "CL",
         "addressLocality": "Temuco",
         "streetAddress": "UFRO"
      }
  },
   "location": {
       "value": {
         "type": "Point",
         "coordinates": [-38.751398,-72.605476],
         "latitude": -38.751398,
         "longitude": -72.605476
      },
      "type": "geo:json"
 },
   "data":{
         "type":"objetc",
         "value":{}
```

```
    }
}
```

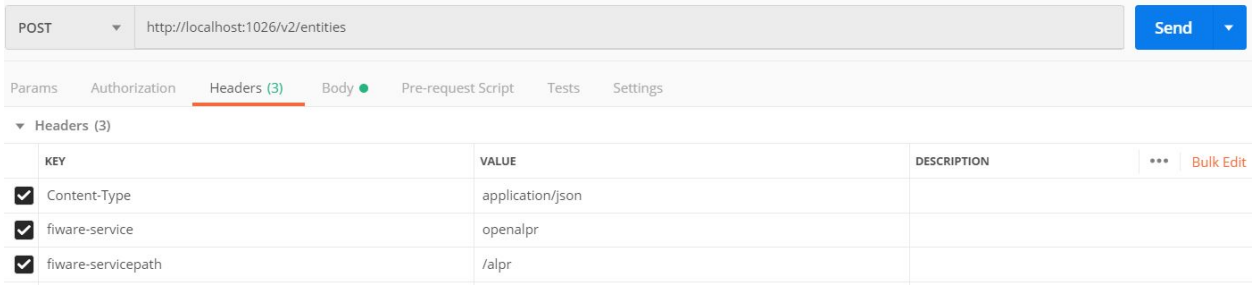

Fig 20. Configuración de Header en Postman.

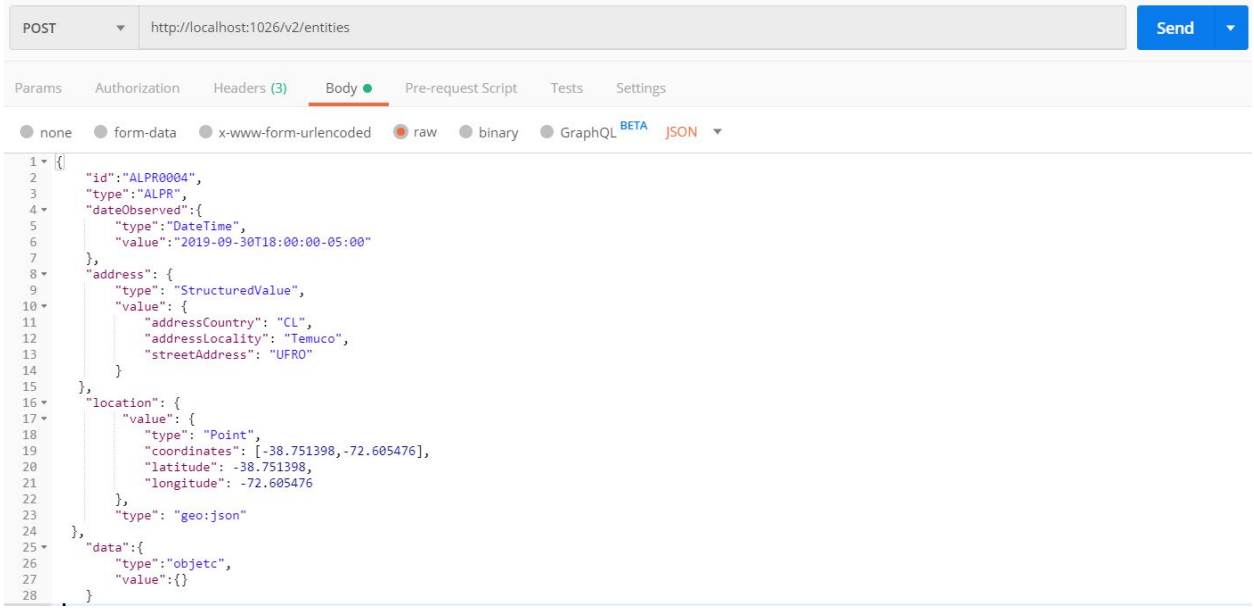

Fig 21. Contenido del Body en Postman.

En caso de necesitar actualizar el contexto, se envía una petición con los siguientes detalles:

```
PATCH http://localhost:1026/v2/entities/ALPR0004/attrs
Header: Content-Type: "application/json"
        fiware-service: openalpr
        fiware-servicepath: /alpr

En el body se indicarán los cambios por medio de parámetros.
```

El siguiente paso consiste en crear la suscripción mediante el servicio de Cygnus, el cual se encargará de persistir los datos mediante reciba actualizaciones de la entidad.

POST http://localhost:1026/v2/subscriptions/
Header: Content-Type: "application/json"
fiware-service: openalpr
fiware-servicepath: /alpr

En el body, se utiliza el contenido de alpr-cygnus.json

```
{
   "description": "Notify Cygnus of all ALPR context changes",
   "subject": {
     "entities": [
       {
         "idPattern": "ALPR.*"
       }
     ]
   },
   "notification": {
     "http": {
       "url": "http://cygnus:5051/notify"
     },
     "attrsFormat": "legacy"
   },
   "throttling": 5
 }
```

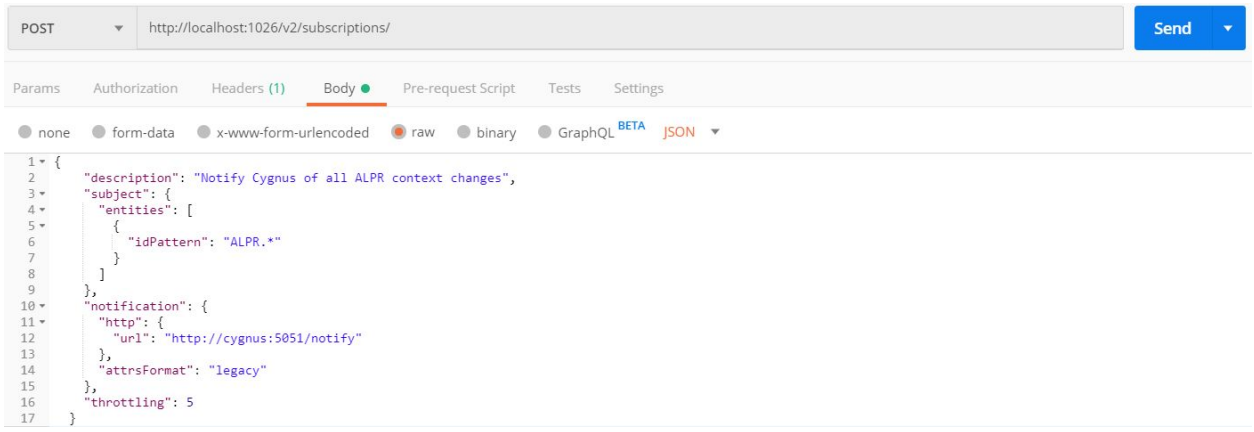

Fig 22. Contenido del Body en Postman para la suscripción mediante Cygnus.

Al hacer esto, lo único que queda por hacer es probar el servicio pasándole una imagen que contenga una patente chilena, donde el servicio responderá con un Array donde lo que destaca es el texto de la patente transcrita a texto, además de la confianza de que la predicción sea correcta.

POST http://localhost:8090/v1/identify/plate?country=us

Header: Content-Type: "application/x-www-form-urlencoded"

En el body, se utiliza un form-data, image = <imagen.jpg>

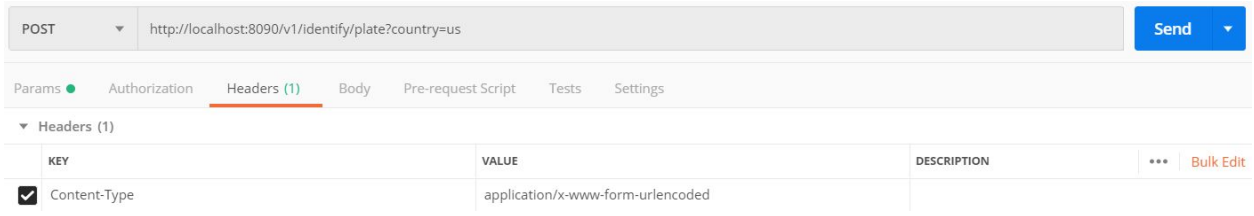

Fig. 23. Configuración del Header en Postman para el envío de la imagen.

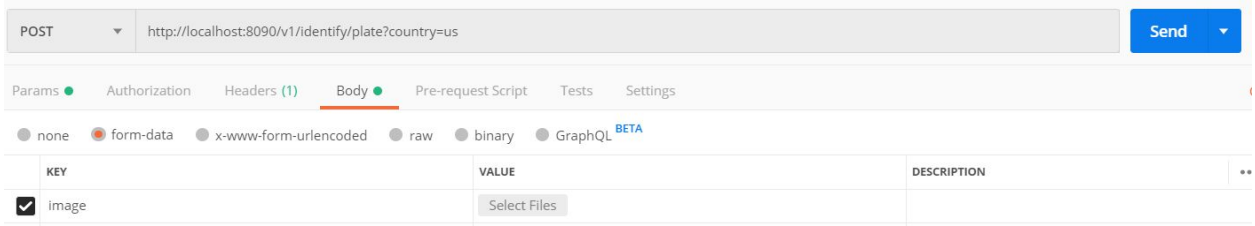

Fig. 24. Contenido del Body en Postman para enviar la imagen.

Al seleccionar una imagen desde el dispositivo y se presiona enviar, lo que se recibe son un Array con las predicciones correspondientes, las cuales se pueden comparar con la imagen enviada.

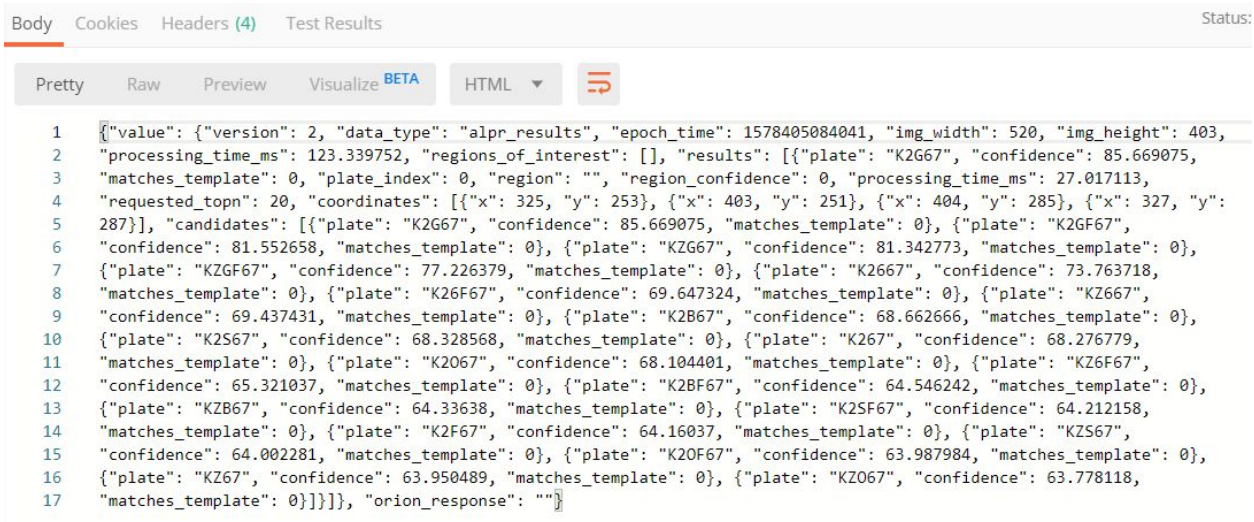

Fig. 25. Respuesta del servicio Open ALPR con las predicciones de la patente.

Fig. 26 Imagen de prueba.

Implementación de Fiware en Amazon Web Services.

Creación y configuración de la instancia.

En primer lugar, se debe crear una instancia EC2 en la consola de Amazon Web Services (AWS desde ahora), que servirá como nuestro propio servidor privado. Para esto, vamos a la consola de EC2, nos dirigimos a la pestaña Instances y para crear una nueva instancia, escogemos la opción Launch Instance (ambas cosas resaltadas en la imagen siguiente).

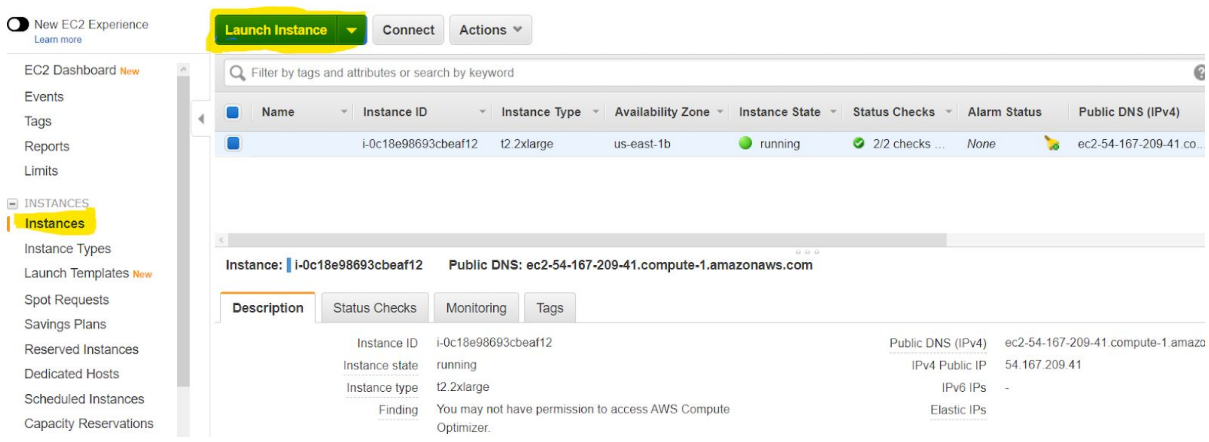

Fig. 27. Dashboard de instancias en la consola AWS.

Se desplegará un panel donde da elegir entre los Amazon Machine Image (AMI) disponibles en la plataforma, en este caso se eligió el “Ubuntu Server 18.04 LTS (HVM), SSD Volume Type”, que se muestra en la imagen.

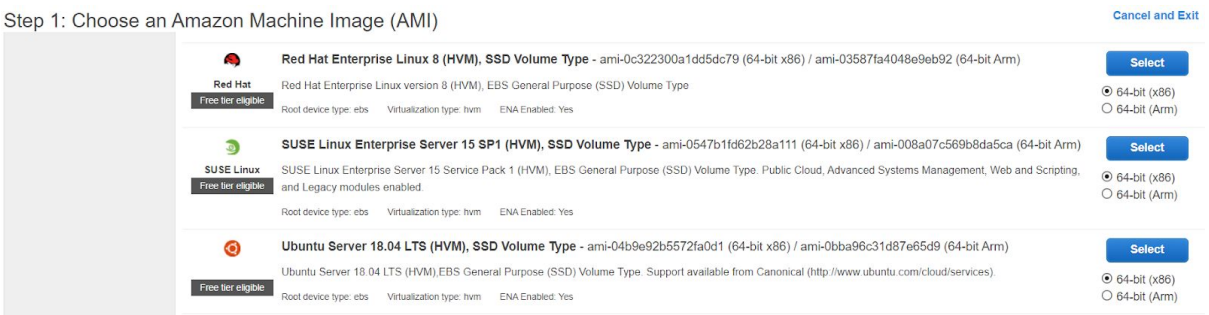

Fig. 28. Selección de AMI para crear una instancia.

En el siguiente paso, se pide escoger el tipo de instancia que se desea crear, en este caso se debe priorizar la memoria RAM, debido a la cantidad de servicios que se desplegarán, para esto se selecciona uno de 8 Gb. de RAM, como se muestra en la imagen.

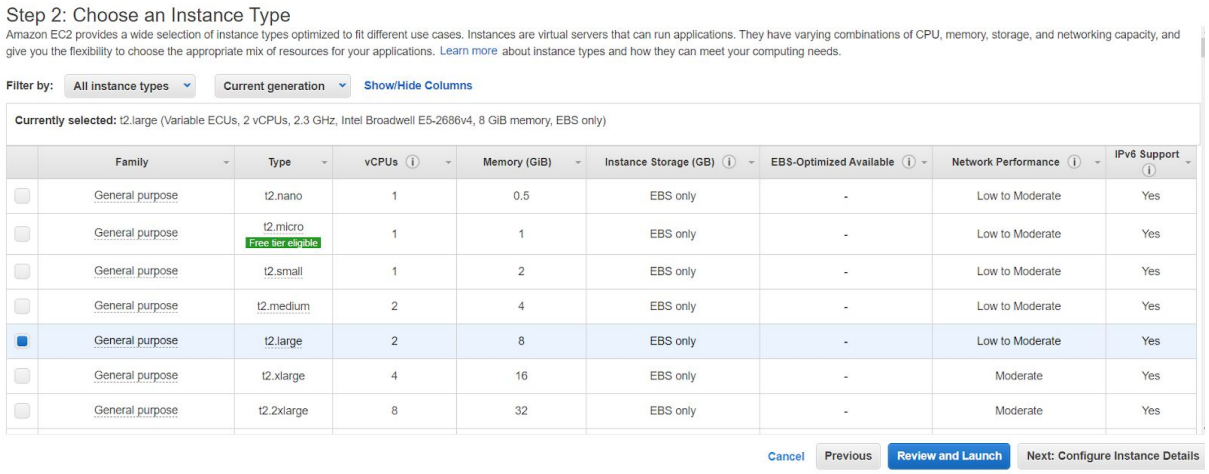

Fig. 29. Elección de tipo de instancia.

En el paso 3, se configuran los detalles de la configuración de la instancia, en este caso se dejarán los valores por defecto.

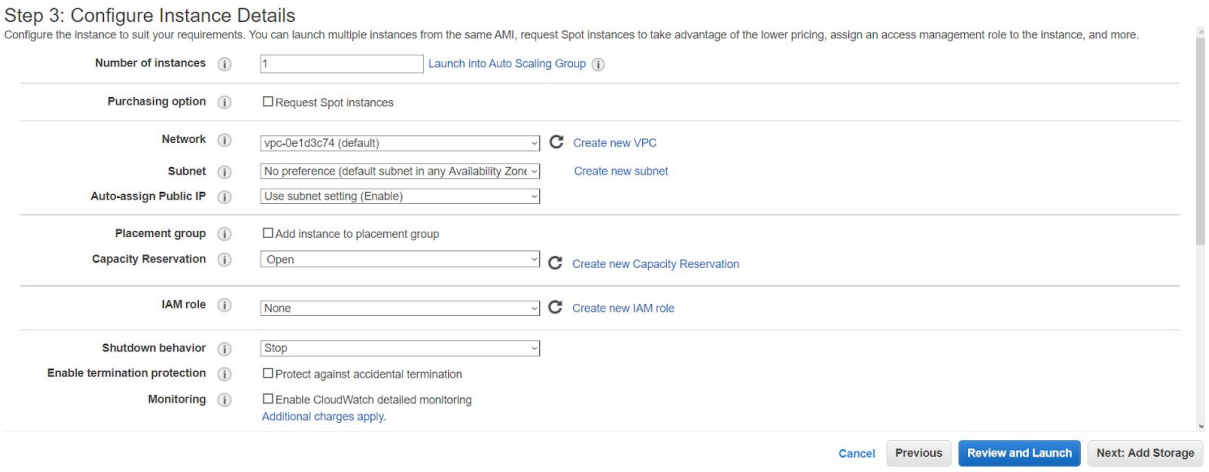

Fig. 30. Detalles de configuración de la instancia EC2.

En el paso 4, se pide configurar la memoria de disco duro de la instancia, la cual será de 15 Gb. en este caso para no tener problemas de memoria a futuro.

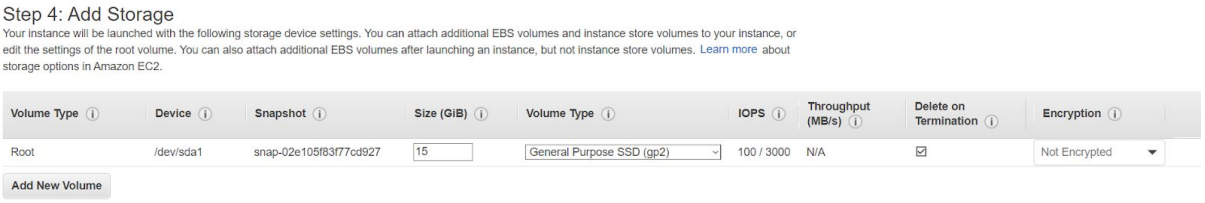

Fig. 31. Configuración de la capacidad del almacenamiento.

No es necesario realizar el paso 5, por lo que se puede saltar al siguiente paso.

El paso 6 consiste en configurar la seguridad de la instancia, en este caso es necesario abrir los puertos que se utilizarán en la solución Fiware a implementar. En la siguiente imagen se muestran los puertos que se abren, escogiendo la opción de crear un nuevo Security Group.

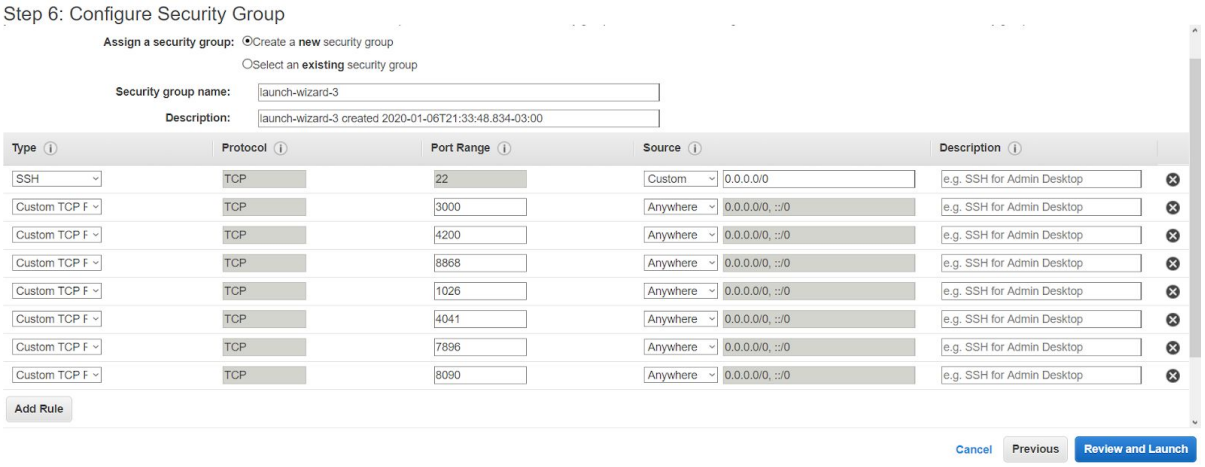

Fig. 32. Configuración de puertos para servicios de Fiware.

Para finalizar la creación de la instancia, se muestra un Review para verificar que todo está correcto y finalmente se le da al botón Launch.

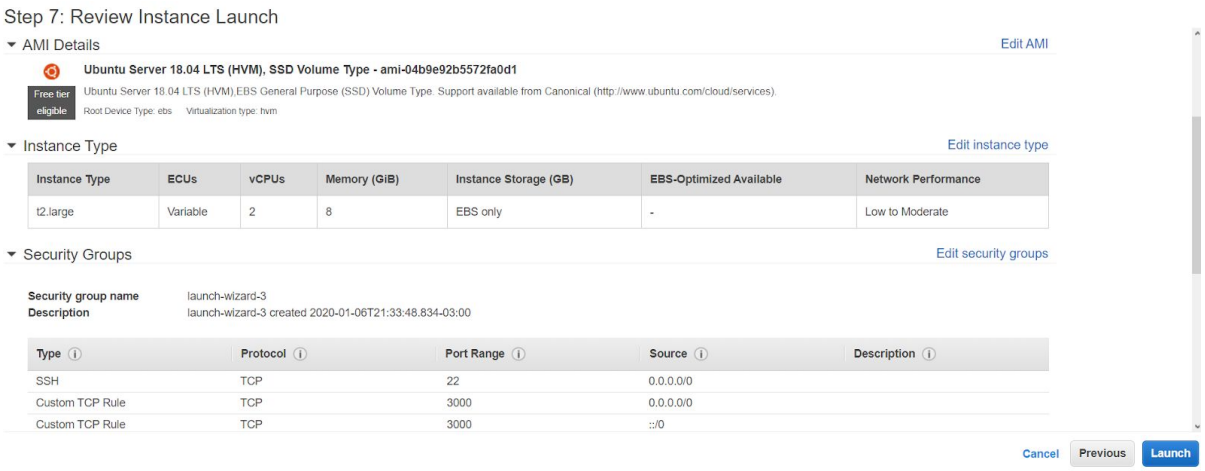

Fig. 33. Review de la instancia antes de lanzarla.

Al lanzar la instancia, emergerá una ventana indicando que se debe crear una par de llaves para poder acceder a la instancia por medio de SSH. Se escoge la opción de crear un nuevo par de llaves, se ingresa el nombre que tendrá la llave y se descarga un archivo del tipo <“nombre_llave”.pem>, donde “nombre_llave” corresponde al nombre que se ingresó previamente. Luego de esto, se mostrará un panel o dashboard en el que aparecen las propias instancias, donde se puede ver la información de esta, como la dirección IP y DNS públicos para acceder a éstas vía SSH.

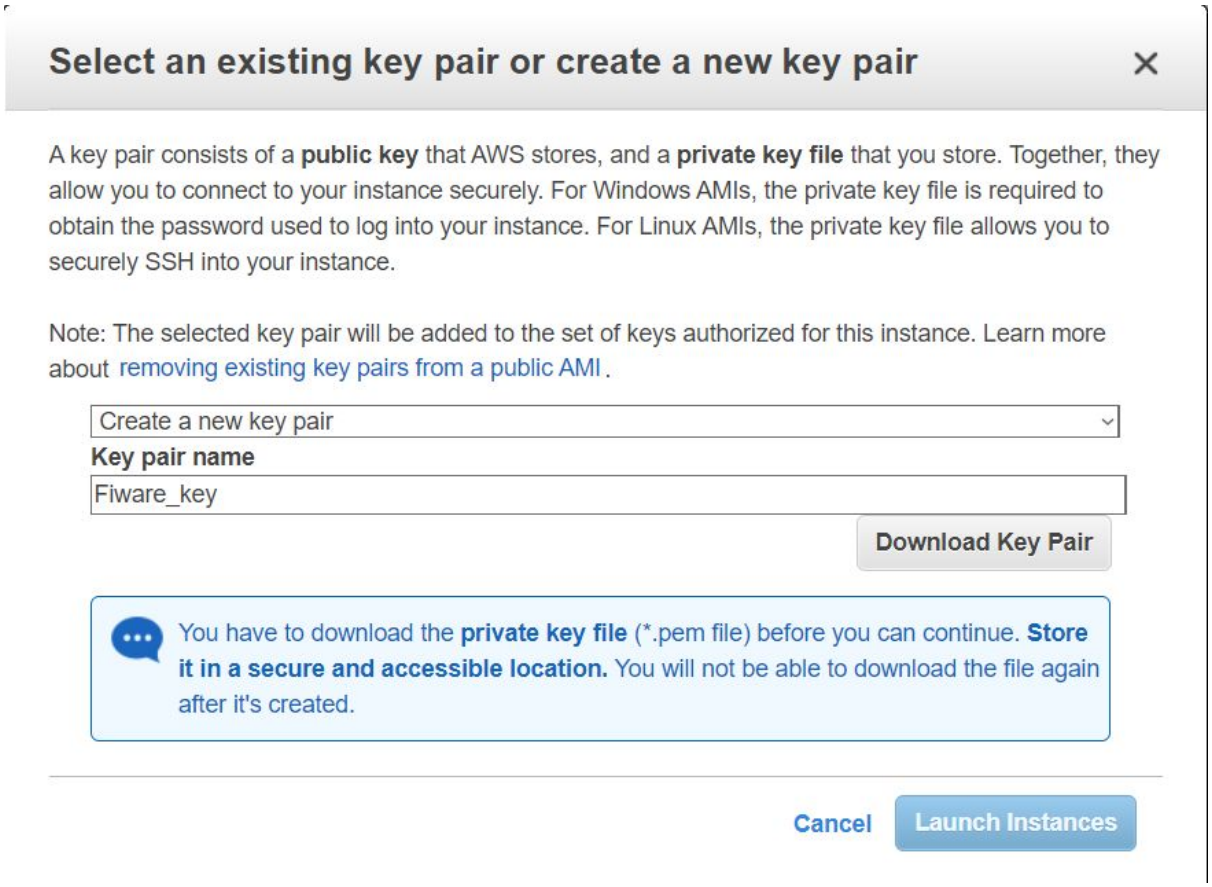

Fig. 34. Ventana para descargar la llave privada para conectarse a la instancia por SSH.

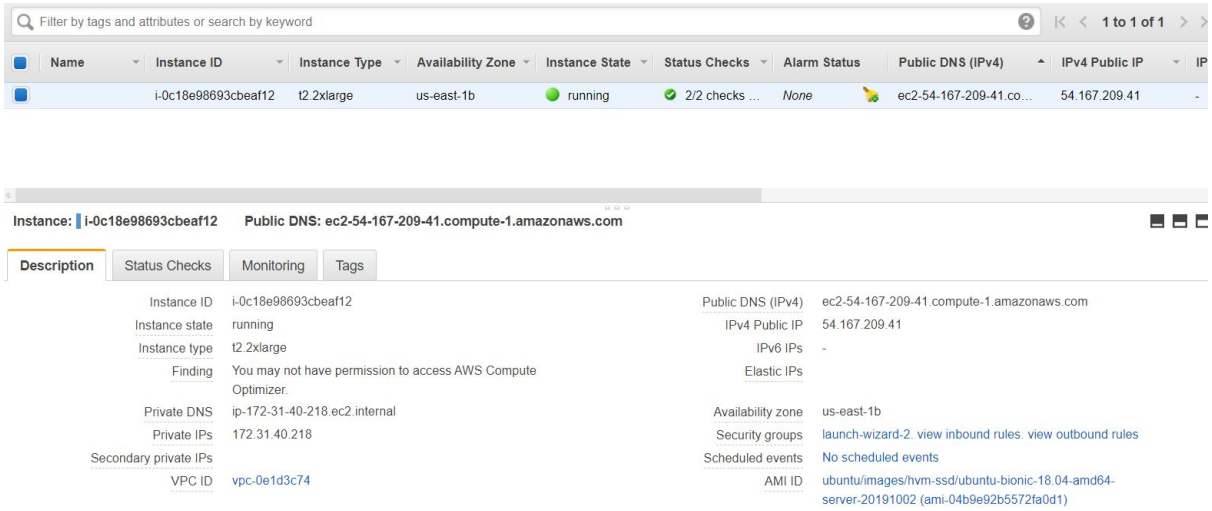

Fig. 35. Detalles de la instancia creada recientemente.

Para acceder al servidor vía SSH, se deben dar ciertos permisos al archivo que se descargó hace unos momentos (la llave), de la siguiente manera:

```
chmod 400 nombre_llave.pem
```

Utilizando este archivo, se conecta por SSH con el siguiente comando (utilizando el DNS público):

```
ssh -i "nombre_llave.pem" ubuntu@DIRECCION_DNS_PUBLICA
```

Seguido estos pasos, se loguea a la instancia por medio del usuario llamado ubuntu, por defecto.

Prerrequisitos para implementar Fiware.

Es necesario que la máquina posea Git, Docker y Docker-compose. Para aquello, se siguen los siguientes pasos:

Instalación de Docker

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

```
sudo apt-get update

sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common


curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -



sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

sudo usermod -aG docker <tu usuario>
```

Instalación de Docker-compose

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.25.0/docker-compose-$(uname
-s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Adicionalmente, para evitar futuros problemas de memoria, se debe escribir el siguiente comando:

```
sudo sysctl -w vm.max_map_count=262144
```

Para que los cambios persistan en el sistema debemos modificar el archivo /etc/sysctl.conf y agregar la siguiente línea al final del archivo:

```
vm.max_map_count=262144
```

Despliegue de servicios Fiware.

Idéntico al ejemplo en LocalHost que está documentado anteriormente, se levantan los servicios mediante el comando “docker-compose up -d”, repitiendo todo salvo que la dirección a la que hacer peticiones, en este caso se debe utilizar la dirección DNS pública de la instancia.

Conclusiones

FIWARE es un framework para el desarrollo e implementación de soluciones smart, el principal componente y que define una solución power by FIWARE es el Orion Context Broker y alrededor de él se implementan todos los componentes que sean requeridos para la solución, en este documento se han definido cada uno de ellos de acuerdo a las categorías definidas por funcionalidad, esta información ha sido obtenida de la documentación oficial del framework de FIWARE disponible en la web.

Se plantea la configuración y puesta en marcha de dos ejemplos de integración e implantación utilizando los componentes de FIWARE, la visualización de datos en tiempo real, y la configuración y puesta en marcha de un dispositivo IOT para la captura y envío de datos a al servicios de IOT-agent mediante el protocolo PDI-IoTA-UltraLight. Además se entrega la implementación de un servicio ALPR para la identificación de matrículas de vehículos integrado a la plataforma de FIWARE. Además se realiza la migración del proyecto completo a la plataforma Amazon Web Service, en una instancia de EC2.

Al finalizar este proyecto se entrega la documentación del paso a paso para dos ejemplos que reflejan casos de necesidad real, esta documentación y experiencia se disponen para que se continúe el trabajo con la plataforma de FIWARE en aplicaciones reales, dentro de la región o de Chile.