INTEGRACIÓN DE SERVICIOS Y TECNOLOGÍAS IOT: DETECCIÓN DE VEHÍCULOS Y PREDICCIÓN DE NIVELES DE TRÁFICO / IOT SERVICES AND TECHNOLOGIES INTEGRATION: VEHICLES DETECTION AND TRAFFIC LEVELS PREDICTION

TRABAJO FIN DE MÁSTER
CURSO 2018-2019

AUTOR
VINCENZO ANTONIO CASCHETTO SIRACUSA

DIRECTOR
GONZALO PAJARES MARTINSANZ

MÁSTER EN INTERNET DE LAS COSAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

INTEGRACIÓN DE SERVICIOS Y TECNOLOGÍAS IOT: DETECCIÓN DE VEHÍCULOS Y PREDICCIÓN DE NIVELES DE TRÁFICO
IOT SERVICES AND TECHNOLOGIES INTEGRATION: VEHICLES DETECTION AND TRAFFIC LEVELS PREDICTION

TRABAJO DE FIN DE MÁSTER EN INTERNET DE LAS COSAS
DEPARTAMENTO DE INGENIERÍA DEL SOFTWARE E INTELIGENCIA ARTIFICIAL

AUTOR
VINCENZO ANTONIO CASCHETTO SIRACUSA

DIRECTOR
GONZALO PAJARES MARTINSANZ

CONVOCATORIA: SEPTIEMBRE 2019
CALIFICACIÓN: 9

MÁSTER EN INTERNET DE LAS COSAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

30 DE SEPTIEMBRE DE 2019

Autorización de Difusión

El abajo firmante, matriculado en el Máster en Internet de las Cosas de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "INTEGRACIÓN DE SERVICIOS Y TECNOLOGÍAS IoT: DETECCIÓN DE VEHÍCULOS Y PREDICCIÓN DE NIVELES DE TRÁFICO", realizado durante el curso académico 2018-2019 bajo la dirección de Gonzalo Pajares Martinsanz en el Departamento de Ingeniería del Software e Inteligencia, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

VINCENZO ANTONIO CASCHETTO SIRACUSA

Septiembre, 2019

Dedicatoria

A mis papás.

Agradecimientos

Gracias a Dios por tantos favores recibidos, por tener una familia tan bella como la que tengo, por encontrarme con mi novia y prometida, por guiarme hasta acá y por permitirme tener una oportunidad tan buena como haber tenido esta experiencia dentro de la universidad.

Agradezco a mis padres, abuelos, hermanas, tíos y primos por sus consejos tan buenos, agradezco también a mis amigos y compañeros de clases por tanto apoyo.

Agradezco a mi tutor Gonzalo Pajares por haberme orientado en la realización de este trabajo.

Y sobre todo agradezco a Vane por haberme ayudado tanto a lo largo de este Máster.

A todos ellos, muchas gracias.

Resumen

Las congestiones de tráfico vehicular son un problema que sufren muchas ciudades en el mundo, cada día es mayor la cantidad de personas que pierden valiosas horas de su tiempo atascadas en el tráfico, lo que se traduce en pérdidas económicas y de otra índole. Una posible solución para reducir el tiempo perdido de los viajeros por congestión y la contaminación creada por los estancamientos, es alertar acerca del flujo de tráfico presente en las vías, de modo que un conductor pueda escoger su ruta conociendo previamente el tráfico existente por zonas. El presente trabajo se focaliza en una solución para monitorizar el flujo vehicular en las vías con el fin de que estos datos sean accesibles por parte de los usuarios, de igual modo, podrán obtener información sobre el tráfico estimado en cierto momento de cualquier día para dichas vías. Esta solución está basada en la integración de servicios y tecnologías de Internet de las Cosas, IoT (Internet of Things), y además está pensada para ser totalmente escalable, de manera que sirva para sensorizar no solo una sino todas las vías que se deseen. El planteamiento de la solución consiste en el uso de cámaras inteligentes con procesamiento de borde (*edge computing*) que implementan contenedores Docker, éstos son capaces de ejecutar códigos en Python con modelos de redes neuronales entrenadas con TensorFlow para reconocimiento de vehículos, luego este código envía por MQTT a Azure IoT Hub los resultados del análisis y, posteriormente, estos datos se almacenan en una base de datos MongoDB hospedada en Azure. Una vez recolectados estos datos, una función Lambda programada en Python y publicada en AWS, a través de Amazon API Gateway, permite actuar como API Rest para consultas HTTP desde múltiples dispositivos y aplicaciones, que sirven para preguntar el nivel de tráfico actual o estimado en cierta fecha y hora. Para predecir el tráfico en otra fecha se utiliza un modelo de regresión lineal realizada con Apache Spark a partir de los datos obtenidos. En este trabajo se explica el proceso evolutivo de todas las pruebas y cambios realizados para llegar a diseñar la solución que se propone, junto con las evidencias de su viabilidad en base a los resultados obtenidos.

Palabras clave

IoT, Redes Neuronales, Tráfico, Azure IoT Hub, Apache Spark, TensorFlow, MongoDB, AWS Lambda, Amazon API Gateway

Abstract

Traffic congestion is a problem that many cities in the world, every day, more people lose valuable hours of their time stuck in traffic, which translates into economic and other losses. A possible solution to reduce the lost time of the travelers due to congestion and the pollution created by the vehicular stagnations is to warn about the traffic flow present on the roads, so that a driver can choose his route knowing previously the existing traffic by zones. The present work focuses on a solution to monitor vehicular flow on the roads so that these data are accessible by users, in the same way, they can obtain information on the estimated traffic at a certain time of any day for such roads. This solution is based on the integration of services and technologies of Internet of Things, IoT, and is also designed to be fully scalable, so that it serves to sensorize not only one, but all the desired routes. The solution is based on the use of intelligent cameras with edge processing (edge computing) that implement Docker containers, they are able for executing Python codes with models of neural networks trained with TensorFlow for vehicle recognition, then this code sends by MQTT to Azure IoT Hub the results of the analysis and, subsequently, this data is stored in a MongoDB database hosted in Azure. Once this data has been collected, a Lambda function programmed in Python and published in AWS, through Amazon API Gateway, allows acting as a Rest API for HTTP queries from multiple devices and applications, which are used to ask the current or estimated traffic level at an specific date and time. To predict traffic on another date, a linear regression model using Apache Spark is used based on the obtained data. This work explains the evolutionary process of all the tests and changes done in order to get the design of the proposed solution, along with the evidence of its feasibility based on the results obtained.

Keywords

IoT, Neural Networks, Traffic, Azure IoT Hub, Apache Spark, TensorFlow, MongoDB, AWS Lambda, Amazon API Gateway

Lista de Acrónimos

ACR Azure Container Registry

ADB Android Debug Bridge

AWS Amazon Web Services

BBDD Base de Datos

CNN Convolutional Neural Networks (Redes Neuronales Convolucionales)

Faas Function as a service (Funciones como servicio)

IA Inteligencia artificial

Iaas Infrastructure as a service (Infraestructura como servicio)

IAM Identity and Access Management

IDE Integrated Development Environment (Entorno de desarrollo integrado)

IoT Internet of Things (Internet de las cosas)

IP Internet Protocol

MAE Mean Absolute Error (Error medio absoluto)

MSE Mean Square Error (Error cuadrático medio)

ML Machine Learning (Aprendizaje automático)

MQTT Message Queue Telemetry Transport

NN Neural Network (Red Neuronal)

Paas Platform as a service (Plataforma como servicio)

RFID Radio Frequency Identification (Identificación por Radiofrecuencia)

RMSE Root Mean Square Error (Error de la raíz cuadrada de la media)

RNN Recurrent Neural Networks (Redes Neuronales Recurrentes)

Saas Software as a service (Software como servicio)

TPD Tránsito Promedio Diario

TPDS Tránsito Promedio Diario Semanal

TPDM Tránsito Promedio Diario Mensual

TPDA Tránsito Promedio Diario Anual

UFW Uncomplicated Firewall

VHP Volumen de la Hora Pico

VOIP Voice Over IP (Voz sobre IP)

VPS Virtual Private Server (Servidor Privado Virtual)

Índice de contenidos

Índice de Tablas

Capítulo 2

Índice de Figuras

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 5

Capítulo 1 Introducción

1.1 Problemática

Las tecnologías desarrolladas en el ámbito de Internet de las Cosas (IoT, Internet of Things) proporcionan soluciones factibles en diversos ámbitos de la actividad humana. Una de ellas se basa en lo que se refiere a la monitorización y control del tráfico de vehículos en vías urbanas e interurbanas, que es donde se focaliza el presente trabajo.

El ritmo de vida actual ha llevado a las personas a estar en constante movimiento, ya que, en ocasiones, se hacen cortas las horas del día para llevar a cabo todas las actividades previstas para una jornada. Teniendo esto en cuenta, un factor de gran importancia en la rutina diaria es el hecho de poder aprovechar al máximo el tiempo, por lo que es necesario evitar aquellas situaciones donde no se realizan actividades provechosas.

Por otra parte, es posible comparar y relacionar dos factores importantes: tiempo y dinero, ya que, al pasar horas sin realizar actividades fructíferas, se está dejando de utilizar ese tiempo en actividades que pueden, por ejemplo, generar ingresos.

Un problema cotidiano que afecta el cumplimiento de este objetivo son los atascos en el tráfico ya que, según estudios realizados por la Comisión Europea, en 2016 los conductores en Barcelona (Prieto y col., 2018) perdieron de media 119 horas al año en atascos, pero este también es un gran problema presente en otras ciudades como Madrid, Sevilla y Valencia, ya que en la primera se pierden hasta 105 horas al año (Prieto y col. 2018)

Todas estas horas desperdiciadas se traducen en pérdidas económicas, ya que equivalen a cerca de 15 jornadas diarias de trabajo que se pierden dentro de un coche. Tomando en cuenta todas estas consideraciones y los lugares donde ocurren estos problemas, en España, al cabo de un año se pierden alrededor de 5.500 millones de euros anuales IPTS (2019).

Por otra parte, este problema ha ido creciendo con el tiempo, ya que en los últimos tres años se puede decir que, al menos en Madrid, la congestión casi se ha

duplicado, según Prieto y col. (2018).

Este problema expuesto no solo puede producir un efecto negativo en la economía del país, sino que además significa un problema para el medio ambiente, lo que termina trayendo como consecuencia complicaciones para la salud y siniestros en las carreteras.

La congestión de tráfico produce una alta concentración de gases tóxicos en el aire, lo que produce como resultado un aumento en la contaminación atmosférica; este hecho afecta a la población en general, pero de manera más pronunciada, a aquellos que ya sufren de problemas respiratorios o cardiovasculares, sin dejar de tomar en cuenta a la población de la tercera edad que es más vulnerable.

Estudios realizados en las Comunidades Autónomas de España, revelan que la contaminación en algunas regiones españolas supera los límites anuales establecidos por Organización Mundial de la Salud (OMS, 2005). Siendo Madrid una ciudad donde se superan notablemente los límites máximos aconsejables de la contaminación urbana, relacionados específicamente con la congestión del tráfico.

Según los datos proporcionados por la consultora ambientalista A. T. Kearney, es necesario actuar en la reducción del tráfico ya que éste es uno de los principales causantes de contaminación atmosférica, que se estima en más del 40% de su peso; además, dichos estudios señalan que este hecho puede ser causante también de aumentos en la mortalidad a edades tempranas. (Prieto, Fanjul, & González, 2018)

Otros estudios que demuestran la gravedad de este problema son los realizados por la Escuela Nacional de Sanidad, que estiman que la cifra de muertes anuales por contaminación en España llega casi a 2.700, mientras que la OMS establece esta cifra en 7.000 y el Instituto Global de Barcelona a un valor de 21.000. Sin embargo, la mayor cifra calculada la proporciona la Agencia Europea de Medio Ambiente, que supera las 31.500 muertes al año debidas a la contaminación atmosférica.

No solo se genera contaminación atmosférica, también se produce contaminación sónica, se agrava el cambio climático y además se producen efectos negativos sobre la naturaleza.

Sumando todo lo anteriormente expuesto, el impacto económico producido

aumenta y se estima que supera los 141 millones de euros anuales en pérdidas solo en Madrid, mientras que para España en general, esta cifra puede superar los 500 millones de euros (DGT, 2016a) (TomTom, 2017).

1.2 Motivación y planteamiento con IoT

A pesar de lo desalentadoras que se ven las cifras expuestas anteriormente, existen soluciones tecnológicas apropiadas que sirven para disminuir los efectos producidos por los atascos en el tráfico, la solución a este problema va ligada al hecho de disminuir de alguna manera el tiempo que se pasa a diario en dichos atascos debido al exceso de tráfico circulando por una misma vía.

Gracias a los avances tecnológicos, es posible encontrar soluciones que puedan ayudar a reducir el tráfico o al menos reducir la rapidez con la que este aumenta. La manera de encontrar esta solución se basa en ofrecer información oportuna a los usuarios de las vías, que indique la cantidad de vehículos que se encuentran en las mismas y así evitarlas, así como a las autoridades encargadas de regular el tráfico para una gestión eficiente de las vías de circulación y espacios de aparcamiento.

Este hecho produciría una mejor distribución del tráfico y con ello se evitaría la generación y acumulación de gases como el CO_2, uno de los principales causantes de la contaminación atmosférica; además se evitaría el uso de bocinas y la producción de esmog que afea la ciudad.

Una solución a este problema puede desarrollarse con la integración de distintas tecnologías, por un lado, es posible utilizar dispositivos IoT que sean capaces de medir el flujo de vehículos y que posteriormente envíen estos datos a otros dispositivos, de manera que se procese la información necesaria para reducir el tráfico.

Otro desarrollo importante de la tecnología, que además es el que se utiliza en este trabajo, es el uso de tecnologías IoT dotadas de Inteligencia Artificial, lo que puede ser útil para diseñar y utilizar sistemas eficientes, en este caso mediante el reconocimiento de imágenes. Tomando esto en cuenta, se busca implementar un sistema que reconozca imágenes de vehículos, los cuente y envíe estos datos para ser almacenados en la nube, así como otros datos de ubicación, tiempo, fecha y hora que

permitirán el acceso a los mismos de manera pública, lo que servirá para realizar modelos de predicción del tráfico, siendo de utilidad tanto para toda la comunidad desarrolladora de software a nivel técnico como para las entidades gestoras del tráfico y los propios usuarios de los vehículos.

Todo este sistema proporciona una solución para la detección de vehículos basada en el marco de IoT, donde se integran distintas tecnologías, protocolos y servicios que permiten hacer de ésta una solución escalable y viable para poder monitorizar y predecir flujos de tráfico en momentos determinados.

La contribución realizada en este trabajo se basa en el hecho de que, luego de realizar un estudio exhaustivo de las diversas tecnologías disponibles, se escogieron aquellas que presentan un alto potencial basado en conceptos relevantes como la escalabilidad, precio, facilidad de mantenimiento del código, aprovisionamiento de los dispositivos y, especialmente en la confiabilidad de funcionamiento del sistema.

Por otra parte, cabe destacar que se utilizaron algunas tecnologías nuevas con errores por encontrarse muchas de ellas en versiones beta. Sin embargo, esto junto con el hecho de que se utilizara software libre, hizo posible que se propusiera la correcciones a la comunidad de desarrolladores a través de Pull Requests en Github, las cuales fueron aceptadas y ahora son accesibles al resto de usuarios.

Al ser tecnologías nuevas y poco estudiadas, trae como consecuencia que exista poca documentación y parte de ella, con errores, lo que provoca fallos en su aplicación y complicaciones en su desarrollo.

En resumen, las contribuciones específicas aportadas en el presente trabajo son:

1) Estudio comparativo de técnicas nuevas, no suficientemente desarrolladas.
2) Integración de diferentes tecnologías seleccionadas para llegar a la solución modular global propuesta.
3) Inserción de métodos de reconocimiento de imágenes mediante técnicas de aprendizaje profundo basadas en Redes Neuronales Convolucionales.
4) Realización de manuales paso a paso para implementación e integración de servicios en la Nube.

1.3 Objetivos

1.3.1 Objetivo General

Diseñar un sistema IoT escalable que, mediante la integración de múltiples tecnologías y servicios, permita medir y almacenar datos del tráfico en las vías de circulación terrestre y que, a partir del entrenamiento de un modelo predictivo con los datos obtenidos, sea capaz de predecir los niveles de tráfico. El sistema debe permitir la difusión de la información obtenida no solo para el momento en que se mide el flujo vehicular sino para cualquier otro momento.

1.3.2 Objetivos Específicos

- Seleccionar el dispositivo de entrada para medir el tráfico.
- Seleccionar el tipo de red neuronal para reconocer las imágenes e implementar su código.
- Seleccionar el hardware y el sistema operativo que se utilizará en los dispositivos de entrada.
- Decidir qué dispositivos o servicios van a realizar el cómputo de la red neuronal.
- Seleccionar el tipo de almacenamiento para los datos obtenidos.
- Seleccionar el tipo de regresión que se va a usar para predecir los niveles de tráfico.
- Seleccionar en qué infraestructuras o servicios se va a implementar el regresor, para entrenar el modelo o para predecir con él.
- Seleccionar los medios de difusión de la información.
- Integrar todas las tecnologías utilizadas para que funcionen en conjunto.

1.4 Organización de la Memoria

Esta memoria se encuentra dividida en cinco capítulos:

- **Capítulo 1:** En este capítulo se encuentra el planteamiento del problema, lo cual justifica o motiva la realización de este proyecto, así como además se plantean los

objetivos generales y específicos que rigen el mismo.

- **Capítulo 2:** En este capítulo se muestran algunas aplicaciones similares que existen actualmente. Además, se explican las bases teóricas que en las que se basan las redes neuronales, las mejoras que se han hecho para aumentar su rapidez en el cómputo, las maneras de detectar más de un elemento de interés a la vez en una foto, las maneras de realizar un seguimiento de un mismo objeto en un video, entre otros. También se mencionan todos los servicios en la Nube que se utilizan a lo largo del trabajo.
- **Capítulo 3**: En este capítulo se presenta el orden en el que se desarrollaron las pruebas de cada tecnología y los análisis o conclusiones obtenidas en cada una de estas pruebas.
- **Capítulo 4:** Se explica el diseño propuesto para llevar a cabo la solución, se presentan las tecnologías, plataformas y servicios que se utilizaron para poder consumir datos masivamente en la nube y la interrelación que hay entre ellos. Por último, se presentan unos instructivos para poder implementar la solución diseñada con los códigos desarrollados en la Nube y así poder reproducir la aplicación finalmente desarrollada.
- **Capítulo 5:** Se muestran los resultados obtenidos al probar la solución diseñada una vez todos los servicios ya se han integrado. Se muestra su funcionamiento y resultados de tiempos de respuesta obtenidos.
- **Capítulo 6:** En este capítulo se presentan las conclusiones a las que se llegaron gracias a la elaboración de este proyecto y recomendaciones de trabajo futuro posible para darle continuidad a este proyecto y mejorarlo.

Capítulo 2 Marco Teórico

Actualmente, una de las maneras más efectivas para medir el tráfico de vehículos en las vías públicas es el uso de contadores vehiculares, lo que permite la elección de vías con menor cantidad de vehículos circulando y, al mismo tiempo, llevar a cabo tareas eficientes como el control de la distribución de semáforos o gestión de plazas de parking, entre otras.

En este capítulo se exponen las distintas posibilidades tecnológicas existentes hasta la actualidad para resolver el problema antes planteado, todas ellas enfocadas hacia el reconocimiento o conteo vehicular pero hecho de diversas formas, con el fin de que esto sirva como revisión general de los métodos existentes.

Para ello, previamente se realiza una revisión de los conceptos importantes referentes al tránsito y al control, así como al estudio del mismo, que comprende la sección 2.1.

Seguidamente, se definen extensamente distintos conceptos, tecnologías, servicios, sistemas y herramientas utilizados para llevar a cabo los objetivos propuestos y llegar al diseño que se presenta. En la sección 2.2 se realiza una descripción conceptual relativa al paradigma IoT. En la sección 2.3 se describe un elemento sensor importante, cual es la captura de imágenes mediante una cámara con capacidad de procesamiento de borde. En la sección 2.4 se introducen los aspectos más relevantes relacionados con aprendizaje automático, que constituye un elemento clave del diseño inteligente que se propone como solución y que permite la detección de los vehículos en movimiento mediante el uso de redes neuronales convolucionales y también se introduce el concepto de regresión, el cual puede servir para la predicción de los niveles de tráfico. En la sección 2.5 se describen aspectos relevantes relacionados con el procesamiento en la nube, que constituye uno de los elementos fundamentales de IoT. Finalmente, en la sección 2.6 se analizan las herramientas útiles para el desarrollo, acordes con las tecnologías y métodos analizados.

2.1 Ingeniería de Tránsito

Existe un área de la Ingeniería Civil encargada de este tipo de estudios, denominada Ingeniería de Tránsito, que se encarga de obtener y analizar información concerniente al volumen y velocidad de vehículos, además de la capacidad de las vías y todos los datos referentes al movimiento vehicular y peatonal.

Es posible agrupar estos estudios en diferentes categorías: Inventario, volúmenes, velocidades, demoras, estacionamientos, transporte público, accidentes, entre otros. Para el presente trabajo se definen las tres categorías mejor relacionadas con el tema objeto de estudio y que abarcan los distintos aspectos de manera más general.

- **Inventario:** es la primera etapa de cualquier estudio de tránsito. Recoge toda la información relacionada con la vía bajo estudio, esto incluye el ancho de los carriles, espacios de aparcamiento, clasificación, mantenimiento y señalización. Este estudio consta de una planificación, un trabajo de campo y un resumen de los resultados obtenidos. Es necesario actualizar esta información periódicamente.
- **Estudios administrativos:** se refieren a la información perteneciente a las oficinas de gobierno, donde se realizan encuestas, fotografías y mediciones de campo y que sirven como soporte para los inventarios.
- **Estudios dinámicos**: información referente a la operación de una vía, incluyendo volúmenes de tránsito vehicular, velocidades y tiempos de viaje (Sisalima, 2018).

Dentro de todos estos estudios, uno de los más importantes es el estudio del volumen, ya que permite determinar la cantidad de vehículos o peatones en tramos específicos de una zona vial en un intervalo de tiempo dado. Este factor no es determinante ya que depende de variables que pueden dar resultados diferentes, por ejemplo, el día de la semana en que se realiza el estudio o las horas más o menos transitadas, entre otros. Sin embargo, permite estimar de manera muy parecida a la realidad, el volumen de tránsito de una zona en particular.

Dependiendo del tiempo en el que se realice el estudio, el volumen vehicular se clasifica en:

- **Tránsito Promedio Diario (TPD):** número de vehículos que pasan por una vía durante un día. Indica la demanda diaria en una calle o carretera, con él se puede determinar si es necesario crear y mejorar vialidades.
- **Tránsito Promedio Diario Semanal (TPDS):** número de vehículos que transitan una vía durante una semana. Tiene las mismas funciones que el TPD.
- **Tránsito Promedio Diario Mensual (TPDM):** cantidad de vehículos que transitan una vía durante un mes.
- **Tránsito Promedio Diario Anual (TPDA):** cantidad total de vehículos que transitan un punto vial específico durante un año. Con ello se determinan variaciones del volumen mediante peajes, lo que indica patrones de viajes en las zonas que se quieren estudiar.
- **Volumen de la hora pico (VHP):** se mide a través de la Tasa de Flujo, que indica el total de coches observados durante periodos inferiores a 60 min. Sirve para analizar flujos en horas pico y analizar las características de los flujos o volúmenes máximos.

El estudio del volumen de tránsito se puede medir de manera manual, pero también se han desarrollado métodos automáticos que facilitan este conteo, de manera que puedan determinarse no sólo la cantidad de coches que transitan una vía sino también, la velocidad con la que éstos la recorren.

Los métodos actualmente utilizados van desde circuitos inductivos, magnetómetros, radares de microondas, infrarrojos, ultrasonidos, hasta la detección por procesamiento de imágenes.

Actualmente la Dirección General de Tráfico (DGT) permite la utilización de una aplicación de móvil con la que se puede medir el tráfico en tiempo real, monitorizando el movimiento de peatones y vehículos a través de las señales de Wi-Fi y Bluetooth emitidas por los teléfonos. Esto se puede hacer a cualquier hora del día y en cualquier lugar que se quiera estudiar (DGT, 2016b).

Para el uso de este dispositivo es necesario que haya corriente eléctrica y conexión

a internet, mientras que no es necesario realizar obras para su utilización ya que se puede colocar fácilmente incluso en un semáforo. Los resultados de este estudio se pueden observar a través de la red social Twitter (DGT, 2016b).

Por otra parte, existen dos grandes aplicaciones muy conocidas y utilizadas que son Google Maps (Google Maps, 2019) y Waze (Waze, 2019), con ambas es posible compartir información en tiempo real referente al tráfico en una zona específica.

En el caso de la aplicación Waze, con solo tener la aplicación abierta es posible emitir información ya que a través de la conexión a internet ésta puede medir parámetros como la ubicación en una zona vial y la velocidad a la que esta se recorre. Al mismo tiempo, la aplicación proporciona información de otros usuarios que están o han estado en ese lugar, generando así una red entrelazada de información.

Hay muchas maneras de medir el tráfico, una de las propuestas existentes consiste en un mecanismo que toma en cuenta el número de *beacons* recibido por vehículo a través de comunicaciones inalámbricas de las operadoras móviles y, al mismo tiempo, los relaciona con la ubicación donde se recibe esta señal, de esta manera puede definirse el flujo de coches que circulan por una vía. Este método tiene como limitación que las cantidades indicadas pueden presentar grandes errores, ya que generalmente en el vehículo no va solo el conductor y, para el caso de transporte público, las señales darían mediciones erróneas.

Un método que no se debe dejar de lado es el que ha sido más utilizado a lo largo del tiempo, que consiste en sensores a nivel de calzada o dentro del pavimento. El funcionamiento de los mismos se basa en las características de medición y el tipo de tránsito vehicular. La limitación de este método es que no cuenta vehículos sino pisadas, haciendo que la medida dependa del peso del coche y de la cantidad de ejes o ruedas, por lo que, si pasa, por ejemplo, un autobús es posible que lo tome como varios vehículos y no como uno solo.

No solo es importante medir la cantidad de coches que circulan por una vía, sino también la velocidad a la que la circulan, ya que es posible encontrar una vía muy transitada pero fluida o una vía poco transitada, pero con atascos. Por esta razón, es

necesario encontrar un método que sea capaz de medir todos estos parámetros y que además sea capaz de distinguir el tipo de vehículo que está transitando, para evitar confusiones en el conteo.

Una manera de cumplir con estos requisitos es hacer uso de la Inteligencia Artificial, ya que ésta permite utilizar técnicas capaces de detectar a través de imágenes, la cantidad, el tipo y la velocidad de vehículos que transitan una vía, recolectando así todos los parámetros que hacen del conteo vehicular un proceso eficiente (Sisalima, 2018)

Todos estos métodos posibles, en general, incluyen el uso de sensores para el conteo de tráfico, por lo tanto, es posible decir que se trata de un tema relacionado con IoT.

2.2 Internet de las Cosas

Actualmente, gracias al avance tecnológico, una herramienta muy efectiva para resolver problemas cotidianos es internet. A partir de 1999 se introdujo el concepto de Internet de las Cosas (Internet of Things, IoT) por Kevin Ashton, experto en identificación por radiofrecuencia (RFID), que lo define como una red o sistema de objetos físicos que son capaces de comunicarse con otros equipos, dispositivos y máquinas, compartiendo información y conexión a internet mediante sensores e interconectados por un protocolo de internet (IP) público o privado (Somayya Madakam, 2015).

Este término se desarrolló luego de lograr conectar procesos que cotidianamente se realizaban mediante intervención humana a internet, facilitando así el seguimiento de mercancías y disminuyendo costos de mano de obra.

Bajo este concepto se destaca la presencia de una variedad de objetos que son capaces de comunicarse y colaborar mutuamente, lo que da como resultado el desarrollo de nuevas aplicaciones y servicios de valor añadido, creando un entorno inteligente. Se hace de forma que los sistemas puedan trabajar de forma autónoma y adecuada, reconociendo eventos y cambios.

El objetivo de esta nueva tendencia es romper las barreras físicas entre dispositivos, de manera que la comunicación entre los mismos esté menos limitada y que no solo se puedan conectar máquinas sino dispositivos de distintos tamaños y utilidades,

como vehículos, sistemas industriales, personas, equipos médicos, entre otros. Lo que da como resultado una mejora notable en la calidad de vida de las personas y empresas (Stanescu, 2019).

El uso de internet permite que la interacción sea posible de distintas maneras, interpersonal, entre máquinas, pero también entre personas y máquinas, lo que proporciona una red de colaboración global.

Esta metodología se ha visto impulsada por diversos factores que sirven como empuje a su crecimiento, entre ellos se destaca la conectividad en todas partes, ya que actualmente es posible "conectar" todo gracias a las tecnologías inalámbricas, entre otras.

Por otra parte, el hecho de que ahora se pueda almacenar una cantidad grande de información en dispositivos muy pequeños, ha impulsado el desarrollo de sensores de tamaño reducido y de bajo costo con tecnología incorporada que permite cómputo y comunicación.

El protocolo IP pasó a ser un factor dominante a la hora de crear redes y permite la utilización de una plataforma con una alta definición y de fácil implementación en software. Esto permite que se adopte de forma generalizada el uso de redes basadas en este protocolo, lo que provee herramientas fáciles de usar y de bajo costo.

Para analizar datos, se han desarrollado nuevos algoritmos con altas capacidades para el cálculo, almacenamiento de datos y servicios en la nube, que permiten la interacción y análisis de grandes cantidades de datos, con ello además se extraen grandes cantidades de información y abundante conocimiento. Con respecto a este tema ha habido grandes avances impulsados por las inversiones de la industria en temas relacionados con la investigación, el desarrollo, la fabricación y ciertas leyes que garantizan poder llevarlos a cabo con precios bajos y menor consumo de energía.

Por último, IoT se ha visto impulsado por el surgimiento de la computación en la nube, con lo que se aprovechan los recursos informáticos conectados en red, logrando así el procesamiento, gestión y almacenamiento de datos. Así, dispositivos de pequeño tamaño pueden interactuar con sistemas más potentes y mejorar las capacidades analíticas y de control (Sisalima, 2018).

2.3 EIC MS Vision 500: Dispositivo con cámara y procesamiento de borde

El dispositivo EIC MS Vision 500 posee una cámara y tiene la particularidad de que posee un procesador Qualcomm QCS603 capaz de ejecutar modelos pre-entrenados para procesar las imágenes que obtiene y enviar a Azure los datos procesados (véase sección 2.5.11). Lo interesante de esto es que estos modelos se ejecutan en contenedores Docker. Esto permite que sea muy fácil desplegar código/modelos en estos dispositivos, figura 2.1.

Figura 2.1 Dispositivo EIC MS Vision 500 y características.

La implementación de estos contenedores se puede hacer directamente desde IoT Edge (véase sección 2.5.11.2), lo cual permite actualizar los modelos muy fácilmente a través de internet sin tener que acudir presencialmente al lugar de ubicación del dispositivo.

La utilización de la cámara con procesamiento de borde EIC MS Vision 500 tiene muchas ventajas por encima de otras opciones como la Raspberry Pi 3 o prácticamente cualquier otra opción, destacando las siguientes:

- Se procesan las imágenes en el propio dispositivo, no siendo necesario enviar las imágenes a través de internet, lo cual aumenta la seguridad, reduce costos de procesamiento en la nube.
- Está perfectamente integrado con Azure IoT Edge, lo cual permite gobernar los

dispositivos, actualizarlos, saber si están en línea o no, entre otros, de una manera muy sencilla a través del portal de Azure.

- El dispositivo es robusto, su construcción está hecha para quedar de esa forma, a diferencia de por ejemplo una Raspberry Pi 3 con cámara, que dicha adaptación requiere un cuadro personalizado que puede no quedar tan bien construido.
- Las imágenes tienen muy buena calidad.

También posee algunas desventajas:

- El equipo es mucho más costoso que una cámara IP o una raspberry Pi 3 o muchas otras opciones.

Al ser un dispositivo que está disponible solo para pruebas actualmente, existen muy pocos desarrollos disponibles que permitan comenzar a utilizar la cámara de forma fácil y eficiente, lo cual en el caso de este trabajo ha resultado ser una tarea muy complicada. De hecho, muchos enlaces de la propia documentación de Microsoft tienen información incompleta o enlaces web no disponibles.

2.4 Aprendizaje Automático

Existen varias definiciones acerca de qué es Aprendizaje Automático o *Machine Learning*. Una de las definiciones más antiguas e informal fue dada por Arthur Samuel, según él, Machine Learning es "el campo de estudio que le da a las computadoras la capacidad de aprender sin ser programadas explícitamente" (1959). Por otra parte, Tom Mitchell proporciona una definición más moderna: “Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas T y la medida de desempeño P, si su desempeño en tareas en T, medido por P, mejora con la experiencia E." (Géron, 2017).

2.4.1 Tipos de Aprendizaje Automático

En general, se distinguen dos tipos de aprendizaje automático (Das, 2017).

- **Aprendizaje supervisado:** Se tiene un conjunto de datos y ya se sabe cómo es la salida correcta, teniendo la idea de que existe una relación entre la entrada y la

salida. Los problemas de regresión y clasificación abordados en este trabajo se enmarcan dentro de esta categoría. En un problema de regresión, se trata de predecir resultados dentro de una salida continua, lo que significa que se está tratando de asignar variables de entrada a alguna función continua. En un problema de clasificación, se trata de asignar variables de entrada a categorías concretas.

- **Aprendizaje no supervisado:** El aprendizaje no supervisado permite abordar problemas con poca o ninguna idea de cómo deberían ser los resultados. Se puede derivar la estructura de datos donde no necesariamente se conoce el efecto de las variables. Se puede derivar esta estructura agrupando (*clustering*) los datos en función de las relaciones entre las variables existentes.

2.4.2 Redes Neuronales

Tras entrenar al sistema, es decir, tras encontrar la relación o los patrones en los datos de entrada y la salida, se obtiene un modelo que servirá para realizar la clasificación.

Una de las mejores formas para reconocer imágenes es con redes neuronales (Neural Networks, NN), según Matich (2001) existen varias definiciones de qué es una red neuronal:

- Un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles.
- Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico.

Las redes neuronales pueden aprender a reconocer ciertos patrones y calcular la probabilidad de que algo sea una cosa u otra. Por ejemplo, si se entrena un clasificador de fotos de perros y gatos, la red neuronal será capaz de determinar qué probabilidad existe de que esa foto sea un perro o un gato.

Lo más interesante de las redes neuronales es su capacidad de aprender sin tener que enseñarles o indicarles cómo aprender o en qué fijarse para aprender. Lo que se necesita es tener un conjunto de fotos etiquetadas (identificando si pertenecen a un gato o a un perro) para que la NN sea capaz de aprender a reconocerlos.

Un problema grande ocurre cuando el conjunto de datos de entrenamiento no es lo suficientemente general o parecido predecirá los datos a clasificar, por ejemplo, puede ocurrir que se entrene solo con imágenes de animales completos y se quiera determinar la clasificación con imágenes de gatos o perros donde sólo se encuentre la cara y parte del cuerpo. O que se haya entrenado con imágenes donde gran parte de la imagen sea ocupada por el animal y cuando se quiera identificar, la imagen tenga mucho fondo y el animal muy pequeño en proporción a la imagen. Para estos casos, es mejor utilizar Redes Neuronales Convolucionales (Convolutional Neural Networks, CNN) (LeCun, 1989). Las CNN se utilizan en casos con imágenes, en cambio las NN en general son más genéricas y pueden servir para otros casos, como los de tablas con datos no relacionados con sus columnas vecinas. En la figura 2.2 se muestra un esquema de operación de convolución, que constituye la base de las CNN.

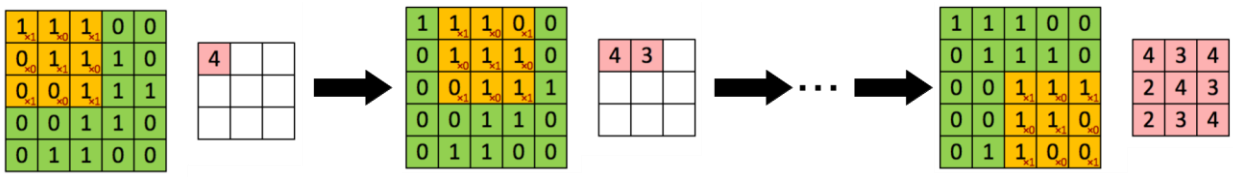

Figura 2.2 Esquema de convolución en una imagen.

El problema de las CNN es que realizan muchos cálculos, que se traducen en un elevado tiempo de cómputo, así como una gran cantidad de recursos utilizados como memoria, GPU o CPU. Es por esto que se crearon técnicas menos costosas computacionalmente como lo son las R-CNN (Ren y col. 2015), las cuales realizan una primera inspección de la imagen, donde se seleccionan las regiones interesantes donde posiblemente esté el elemento que se desea clasificar. Con esto, se reduce considerablemente el tiempo de cómputo ya que la CNN no recorre toda la foto, sino

todas las regiones propuestas por la R-CNN (aproximadamente 2000 por foto), un esquema general de este modelo de red se muestra en la figura 2.3 donde aparece la parte relativa a la extracción de características de la imagen, donde además se propone la región de interés, que se pasa a la parte específica de clasificación de objetos con las correspondientes capas de clasificación, junto con la región espacial donde se contiene el objeto (*Bounding Box*).

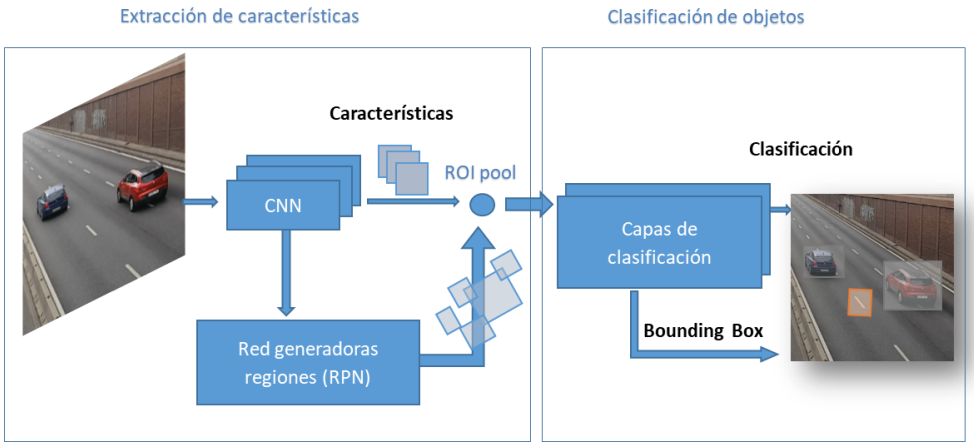

Figura 2.3 Esquema de red neuronal del tipo R-CNN.

Posteriormente, el mismo creador de R-CNN publicó un nuevo método más rápido llamado Fast R-CNN para poder resolver los inconvenientes del anterior. El Fast R-CNN tiene un enfoque similar al del algoritmo de R-CNN pero, en lugar de alimentar las regiones propuestas hacia el CNN, se alimenta la imagen entrante al CNN para generar un mapa de caracterización convolucional, a partir del cual se identifica la región propuesta y se transforman en recuadros (*Bounding Box*) y, posteriormente, usando una capa de agrupación, se redimensionan a un tamaño fijo de manera que pueden ser alimentadas en una capa completamente conectada. En el vector de caracterización, se usa una capa *softmax* para predecir la clase de región propuesta y además los valores de compensación del recuadro.

La razón que hace que "Fast R-CNN" sea más rápido que R-CNN es porque no es necesario alimentar 2000 regiones propuestas a la red neural convolucional en cada momento. En lugar de ello, la operación de convolución se hace solo una vez por imagen y

a partir de ella se genera un mapa de caracterización.

En definitiva, Fast R-CNN es significativamente más rápido en las sesiones de entrenamiento y pruebas que R-CNN. Cuando se observa la forma de actuar de Fast R-CNN durante el tiempo de entrenamiento, incluyendo las regiones propuestas se hace significativamente más lento el algoritmo comparado con cuando no se usan las regiones propuestas. Por lo tanto, las regiones propuestas serían el paso lento en el algoritmo de Fast R-CNN, afectando a su funcionamiento.

Tanto el algoritmo de R-CNN como el de Fast R-CNN usan una búsqueda selectiva para encontrar las regiones propuestas. La búsqueda selectiva es un proceso lento que consume tiempo y afecta al funcionamiento de la red. Por lo tanto, Ren y col. (2015) presentaron un algoritmo de detección de objetos que eliminara el algoritmo de búsqueda selectiva y permitiera que la red aprendiera las regiones propuestas.

De forma similar a como funciona Fast R-CNN, la imagen se proporciona como una salida a una CNN que provee un mapa de caracterización. En lugar de usar el algoritmo de búsqueda selectiva en el mapa de caracterización para identificar las regiones propuestas, se usa una red separada para predecirlas. Las regiones propuestas predichas son posteriormente redimensionadas usando una capa de agrupación que luego es usada para clasificar la imagen dentro de la región propuesta y predecir los valores de compensación para los recuadros redimensionados.

El modelo Faster R-CNN es mucho más rápido que sus predecesores. Por lo tanto, puede también ser usado para la detección de objetos en tiempo real.

Todos los métodos anteriormente nombrados para la detección de objetos utilizan regiones para localizar el objeto dentro de la imagen. La red no ve la imagen completa, sino sólo partes de la imagen que podrían contener el objeto. YOLO o You Only Look Once (Solo lo ves una vez) es un algoritmo de detección de objetos muy diferente basado en los algoritmos anteriores. En YOLO, una única red convolucional predice el recuadro y las probabilidades de clases de esos recuadros (Redmon y col., 2016).

Lo que hace YOLO es tomar una imagen y dividirla en una cuadrícula de dimensión SxS, dentro de cada cuadrícula se toman *m* recuadros. Por cada recuadro, la red produce

una probabilidad de recuadros y valores para los mismos. Los recuadros que tienen la probabilidad de clase por encima de un valor máximo se seleccionan y se usan para ubicar el objeto dentro de la imagen. La limitación de este algoritmo es que tiene problemas con objetos pequeños dentro de la imagen, por ejemplo, puede tener dificultades para detectar una bandada de pájaros. Esto se debe a las restricciones espaciales del algoritmo.

2.4.3 Regresión

El análisis de Regresión es un subcampo de "machine learning supervisado". Su propósito es establecer un modelo para la relación entre un cierto número de características y una variable objetivo continua (Román, 2019).

En los problemas de regresión perseguimos obtener una respuesta cuantitativa, como por ejemplo, predicciones sobre precios de inmuebles o el número de segundos que alguien dedicará a visualizar un vídeo.

Existen varios tipos de regresión, entre ellos se encuentran:

- Regresión lineal
- Regresión logística (Clasificación)
- Árboles de decisión

Cada uno tiene sus ventajas y desventajas, no cualquier tipo de regresión funciona, dependiendo del comportamiento de los datos, puede ser más conveniente uno u otro.

2.4.4 Frameworks para desarrollar aprendizaje automático: Apache Spark

Apache Spark es un framework de programación para procesamiento de datos distribuidos diseñado para ser rápido y de propósito general. Como su propio nombre indica, ha sido desarrollada en el marco del proyecto Apache, lo que hace que su licencia sea Open Source (Apache Spark, 2019) (Yudego, 2018).

Dado que está soportado por Apache, se puede contar con que su mantenimiento y evolución se llevarán a cabo por grupos de trabajo de gran prestigio, y existirá una gran flexibilidad e interconexión con otros módulos de Apache como Hadoop, Hive o Kafka.

Parte de la esencia de Spark es su carácter generalista. Consta de diferentes APIs y

módulos que hacen que pueda ser utilizado por muchos profesionales en todas las etapas del ciclo de vida del dato.

Dichas etapas pueden incluir desde soporte para análisis interactivo de datos con SQL hasta la creación de complejos pipelines de aprendizaje automático y procesamiento en streaming, todo usando el mismo motor de procesamiento y las mismas APIs.

Apache Spark es un motor de procesamiento que se encarga de realizar tres tareas importantes: orquestar, distribuir y monitorear aplicaciones que están conformadas por un gran número de tareas de procesamiento de datos sobre varias máquinas de trabajo, todo esto da lugar a la formación de un *cluster*.

Los datos se pueden leer a través de Amazon S3 o Google Storage, que son soluciones de almacenamiento, pero también es posible hacerlo desde sistemas de almacenamiento distribuido como HDFS, otros sistemas de clave-valor tal como Apache Cassandra o incluso buses de mensajes como Kafka (Amazon, 2019) (Cloud, 2019) (Riccio & Guizado, 2018).

Aún con esto, Apache Spark no está diseñado para almacenar datos dentro de sí, sino que su funcionamiento está enfocado al procesamiento; lo que muestra la gran diferencia del mismo con Hadoop, que presenta de manera integrada por un lado un almacenamiento persistente (HDFS) y por el otro un sistema de procesamiento (MapReduce) (González, 2019).

Otro punto importante en este tema es la velocidad de procesamiento, en este caso es relevante el hecho de que Spark ofrece la posibilidad de realizar el procesamiento en memoria, lo que,con la funcionalidad de MapReduce, puede permitir de manera eficiente otros tipos de operaciones como son Queries interactivas y Procesamiento en Streaming.

Por otro lado, el objetivo general de este motor es su capacidad que tiene para lograr cubrir un amplio catálogo de cargas de trabajo que anteriormente necesitaban sistemas distribuidos diferentes.

Otras características que presentan los sistemas anteriormente mencionados son: procesamiento batch, algoritmos iterativos, Queries interactivas, procesamiento

streaming, entre otras. Estas características generalmente se emplean en un *pipeline* usado frecuentemente para el análisis de datos.

Hemos resaltado también que Apache Spark permite distintos usos, esto se debe a la cantidad de APIs que ofrece, que hacen que los usuarios que tiene diferentes *backgrounds* sean capaces de utilizarlo. Entre las APIs que contiene destacan las de Python, Java, Scala, SQL y R, que presentan funciones integradas además de un desempeño notablemente bueno en cada una de ellas.

Por otra parte, Spark se adapta a las necesidades y preferencias del consumidor, por lo que se puede trabajar con datos más o menos estructurados, como es el caso de RDDs, dataframes y datasets. Además, tiene facilidad de integración con otras herramientas Big Data, especialmente las que provienen del proyecto Apache.

Es posible además la integración del motor con Hadoop, ejecutándose en sus *clusters* y accediendo a los datos que se han almacenado en HDFS y algunas otras fuentes de datos que también pertenecen a Hadoop como Cassandra, Hbase, Kafka, entre otras.

2.5 Computación en la nube

La computación en la nube consiste en un modelo que permite el acceso de red desde cualquier parte y en cualquier momento, a un grupo de recursos informáticos configurables tales como redes, servidores, almacenamiento o aplicaciones, que posteriormente pueden ser aprovisionados y liberados de forma rápida y fácil por parte del administrador o del proveedor de servicios.

Este modelo de computación en la nube, según NIST (2011) se compone de cinco características esenciales:

- **Autoservicio bajo demanda:** el consumidor puede aprovisionar, de manera unilateral y automática, capacidades informáticas como la fecha y la hora del servidor y el almacenamiento en red, de acuerdo a las necesidades sin interacción humana con el proveedor de servicio.
- **Acceso amplio a la red:** las capacidades de cómputo están disponibles en toda la red al igual que el acceso a mecanismos estándar capaces de aprovisionar su uso por plataformas heterogéneas. Por ejemplo, teléfonos móviles, tablets, portátiles,

entre otros.

- **Agrupación de recursos:** los recursos computacionales del proveedor están agrupados para servir múltiples consumidores, utilizando un modelo multi-cuenta, con diferentes recursos físicos y virtuales dinámicamente asignados y reasignados de acuerdo a la demanda del consumidor.
- **Elasticidad rápida:** las capacidades pueden ser elásticamente aprovisionadas y publicadas, en algunos casos automáticamente, para escalar rápidamente hacia afuera y hacia adentro de acuerdo con la demanda. Para el consumidor, las capacidades disponibles para el aprovisionamiento a menudo suelen parecer ilimitadas y suelen ser apropiadas en cualquier cantidad y momento.
- **Servicio medido:** los sistemas de la nube controlan y optimizan automáticamente el uso de recursos por aprovechamiento de una capacidad de almacenamiento a cierto nivel de abstracción apropiado al tipo de servicio, por ejemplo, almacenamiento, procesamiento, ancho de banda y cuentas de usuarios activas. El uso de recursos puede ser monitorizado, controlado y reportado, proporcionando transparencia para cada proveedor y consumidor del servicio utilizado.

2.5.1 Modelos de servicio

La computación en la nube presenta tres modelos de servicio:

- **Infraestructura como servicio o *Infrastructure as a Service* (Iaas):** la capacidad proporcionada al consumidor es para aprovisionar procesamiento, almacenamiento, redes y otros recursos computacionales donde el consumidor está autorizado para desplegar y ejecutar procesos arbitrarios, lo que puede incluir sistemas operativos y aplicaciones. El consumidor no maneja o controla la infraestructura de la nube subyacente, pero tiene el control sobre los sistemas operativos, almacenamiento y aplicaciones desplegadas; y posiblemente control limitado de los componentes de la red seleccionada.
- **Plataforma como servicio o *Platform as a Service* (Paas):** la capacidad proporcionada al consumidor es para implementar en la infraestructura de la nube aplicaciones creadas o adquiridas por el consumidor usando lenguajes de

programación, librerías, servicios y herramientas compatibles con el proveedor. El consumidor no administra ni controla la infraestructura de la nube subyacente, incluida la red, los servidores, los sistemas operativos o el almacenamiento, pero tiene control sobre las aplicaciones implementadas y posiblemente las configuraciones para el entorno de alojamiento de aplicaciones.

- **Software como servicio o *Software as a Service* (Saas):** la capacidad proporcionada al consumidor es para ejecutar sus aplicaciones en la infraestructura de una nube. Las aplicaciones son accesibles desde varios dispositivos como un navegador web o una interfaz. El consumidor no administra la estructura de la nube subyacente que incluye la red, servicios, sistemas operativos, almacenamiento o las capacidades de aplicaciones individuales, con la posible excepción de límites en los ajustes de configuración de aplicación específicos de usuarios.

La figura 2.4 muestra un esquema general de lo expuesto anteriormente (Watts & Raza, 2019).

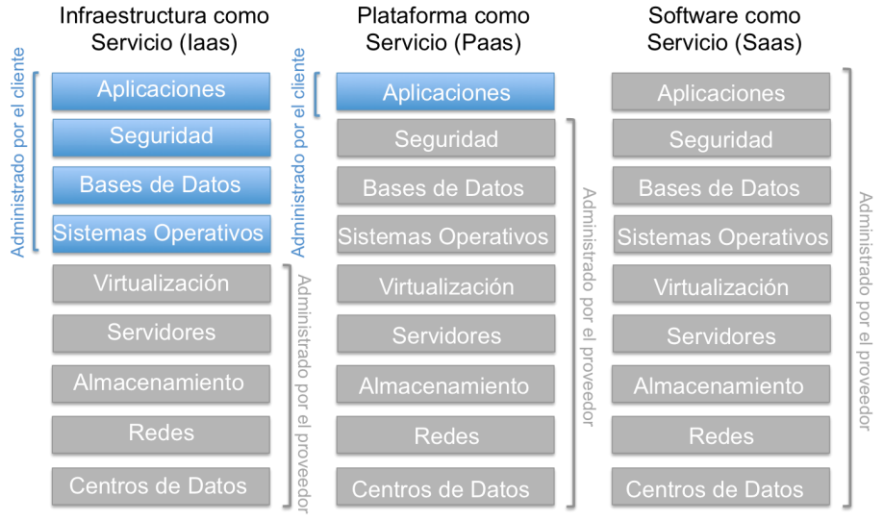

Figura 2.4 Gestiones realizadas en cada modelo de servicio en la nube.

2.5.2 Modelos de implementación

El cómputo en la Nube se presenta en cuatro modelos de implementación diferentes:

- **Nube privada:** la infraestructura de la nube es aprovisionada para uso exclusivo de una única organización que comprende múltiples consumidores, como las unidades de negocio. Esto puede ser propiedad de la organización y ser administrado y operado por la misma, por un tercero o por una combinación de los dos y, además, pueden existir dentro o fuera de las organizaciones.
- **Nube comunitaria:** la infraestructura de la nube es aprovisionada para uso exclusivo de una comunidad de consumidores de organizaciones que comparten preocupaciones, como por ejemplo misión, requerimientos de seguridad, políticas y consideraciones de cumplimiento. Esto puede pertenecer a una o más organizaciones en la comunidad y puede ser administrado y operado por la misma, por terceros o una combinación de ambos, y también puede existir dentro y fuera de las organizaciones.
- **Nube pública:** la infraestructura de la nube puede ser aprovisionada para su uso abierto al público en general. Esta puede pertenecer, ser administrada y operada por una organización comercial, académica o gubernamental, e incluso una combinación de ellas, ubicándose en las instalaciones del proveedor de la nube.
- **Nube híbrida:** la infraestructura de la nube es una composición de dos o más infraestructuras de nube, entre la privada, comunitaria o pública, que continúan siendo entidades únicas, pero no están vinculadas juntas por tecnologías estandarizadas o patentadas que permitan datos y portabilidad de aplicación, por ejemplo, fraccionamiento de nubes para equilibrar la carga entre las mismas.

2.5.3 Informática sin servidor

Actualmente, la tendencia está dirigida hacia el uso de servicios sin servidor o *serverless*. Este tipo de tecnología se refiere a la arquitectura propia de la nube, que permite el traslado de nuevas responsabilidades operativas al proveedor de cómputo, lo que hace que se produzca un aumento en la agilidad y la innovación de la nube. La tecnología *serverless* permite la creación y ejecución de aplicaciones y servicios sin que sea necesario tomar en cuenta los servidores, por lo que se pueden eliminar las tareas de administración de la infraestructura, entre las que destacan el aprovisionamiento de

servidores o nubes, parches, mantenimiento del sistema operativo y capacidad para aprovisionar. Es posible la creación de las mismas sin importar el tipo de aplicación o el servicio de *backend* y también se puede administrar todo lo referente a la ejecución de la aplicación y hacerla altamente disponible para escalarla (AWS-01, 2019).

La información obtenida a partir de tecnologías sin servidor permite la creación de aplicaciones cuyo costo total de propiedad es más bajo, lo que a su vez implica que el enfoque principal por parte de los desarrolladores va dirigido al producto principal y no a la administración y funcionamiento de los servidores y tiempos de ejecución tanto en la nube como de forma local. El hecho de que los costos sean menores permite a los desarrolladores el empleo del tiempo y energía en el desarrollo de productos confiables y escalables.

Entonces, el detalle es que cada tipo de servicio tiene sus pros y sus contras, en general la tendencia actual es que los servicios Saas se vuelven cada vez más rentables porque por lo general no se paga una suscripción, sino que se paga por uso y al ser *serverless*, son ampliamente escalables, pueden procesar millones de peticiones en paralelo sin aumentar el tiempo de respuesta.

2.5.4 Proveedores

Existen muchos proveedores de servicios de cómputo en la nube, sin embargo, el mercado está encabezado por Amazon Web Services (AWS), Microsoft Azure, IBM, Salesforce, SAP, Oracle, Google Cloud Platform (GCP).

2.5.5 IoT en la Nube

Otra ventaja interesante de utilizar algunos de estos proveedores de Cloud, es que también ofrecen servicios para IoT, entre los cuales, tanto AWS como Azure ofrecen los productos y servicios que se muestran en la tabla 2.1 (AWS-02, 2019) (Azure-01, 2019).

Tabla 2.1 Servicios en nubes de AWS y Azure para desarrollar aplicaciones IoT.

	AWS	Microsoft Azure
Software/Sistema operativo para instalar en dispositivos IoT	**Amazon FreeRTOS:** es un sistema operativo para microcontroladores que facilita la programación, implementación, protección, conexión y administración de los dispositivos de borde pequeños y de poca potencia. **AWS IoT Greengrass:** es un software que le permite ejecutar capacidades de informática local, mensajería, almacenamiento de datos en caché, sincronización e inferencias de aprendizaje automático en dispositivos conectados de manera segura.	**Windows 10 IoT Core:** Es un sistema operativo diseñado para dispositivos inteligentes protegidos. Ofrece soporte para arquitectura ARM. Diseñado para ser ligero e integrado con otros servicios de Azure IoT
Servicios de Control	**AWS IoT Core:** permite que los dispositivos conectados interactúen de manera sencilla y segura con las aplicaciones en la nube y otros dispositivos. **AWS IoT Device Defender:** monitoriza y audita continuamente sus configuraciones de IoT para garantizar que no se aparten de las prácticas recomendadas de seguridad. **AWS IoT Device Management:** facilita la incorporación, la organización, la monitorización y la administración remota de dispositivos IoT a gran escala y de forma segura. **AWS IoT Things Graph:** facilita la conexión de diferentes dispositivos y servicios en la nube para crear aplicaciones IoT.	**Azure IoT Central:** Permite crear implementaciones de IoT de una manera completamente administrable. **Azure IoT Hub:** Véase sección 2.5.11 **Azure IoT Edge:** Véase sección 2.5.11.2
Servicios de Datos	**AWS IoT Analytics:** facilita la ejecución de análisis sofisticados en volúmenes masivos de datos de IoT. **AWS IoT Events:** facilita las tareas de detección y respuesta a eventos que provienen de grandes cantidades de aplicaciones y sensores compatibles con IoT. **AWS IoT SiteWise:** facilita la recopilación, estructuración y búsqueda de datos de IoT que provienen de bases de datos de instalaciones industriales, que luego utiliza para analizar el rendimiento de los equipos y procesos.	**Azure Time Series Insights:** Permite el análisis, almacenamiento y administración de datos. **Azure Maps:** Permite agregar análisis especiales a los datos, relacionar los datos con mapas y mostrar la ubicación geográfica de los dispositivos que generan dicha información.

Estos servicios resultan muy útiles si se pretende utilizar a gran escala esta solución, ya que permiten tener un gobierno de los dispositivos, aprovisionar nuevos dispositivos sin que sea muy complicado, así como ver en un dashboard centralizado la información proveniente de los dispositivos, tener un control de versiones y actualizarlos, entre otras ventajas.

2.5.6 Funciones como servicio

Existen servicios de informática sin servidor que permiten ejecutar código, algunos llaman a esto Faas (Function as a service), tanto AWS como Azure o GCP brindan este servicio; a continuación, se mencionarán datos importantes acerca de las dos principales opciones actuales del mercado.

2.5.6.1 AWS Lambda

Tal como indica la descripción de su página web, AWS Lambda es un servicio de informática sin servidor que ejecuta código en respuesta a eventos y administra automáticamente los recursos informáticos subyacentes. Se puede usar AWS Lambda para ampliar la funcionalidad de otros productos de AWS con lógica personalizada o bien crear servicios *back-end* propios que funcionen con el nivel de seguridad, rendimiento y escala de AWS. AWS Lambda puede ejecutar código automáticamente en respuesta a varios eventos, como solicitudes HTTP a través de Amazon API Gateway, modificaciones realizadas en objetos en buckets de Amazon S3, actualizaciones de tablas en Amazon DynamoDB y transiciones de estado en AWS Step Functions (AWS-03, 2019).

AWS tiene la particularidad de que es muy segura ya que por defecto están prohibidas todas las solicitudes a menos que se autorice explícitamente con un permiso IAM (véase 2.4.7.3). Otro detalle a favor de la seguridad es que ningún AWS Lambda tiene acceso directamente a través de internet. Posee las siguientes características clave:

- Económico, siendo de pago por uso, en función del número de solicitudes para sus funciones y la duración, el tiempo necesario para ejecutar el código. El precio varía dependiendo de la cantidad de peticiones al mes. El primer millón de peticiones del mes es gratuito, luego se cobra 0,2 USD por cada millón de solicitudes

posteriores, es decir, 0,0000002 USD por solicitud (AWS-04, 2019).

- Se puede ampliar la funcionalidad de otros productos de AWS con lógica personalizada.
- Se pueden crear servicios *back-end* personalizados.
- Se puede utilizar propio código, no son funciones predefinidas que sólo se pueden parametrizar. Se puede programar de forma nativa con Java, Go, PowerShell, Node.js, C#, Python y Ruby.
- Administración totalmente automatizada.
- Tolerancia a errores integrada.
- Escalado automático
- Modelo de seguridad integrado

2.5.6.2 Microsoft Azure Functions

Microsoft Azure Functions es el servicio informático sin servidor alojado en la nube de Microsoft Azure. Azure Functions y la informática sin servidor, en general, están diseñadas para acelerar y simplificar el desarrollo de aplicaciones (Azure-02, 2019).

Con Azure Functions, un usuario puede simplemente crear y cargar código, y luego definir los disparadores o eventos que ejecutarán el código. Los disparadores pueden provenir de una amplia gama de fuentes, incluida la aplicación de otro usuario u otros servicios en la nube, como bases de datos y centros de eventos y notificaciones.

En cuanto al uso de Azure Functions conviene señalar que las funciones informáticas sin servidor no suelen ser aplicaciones completas con todas las funciones. En cambio, las funciones manejan tareas específicas de corta duración. La mayoría de las funciones implican algún tipo de procesamiento de datos, como el procesamiento de imágenes o pedidos, así como el mantenimiento de archivos o la recopilación de datos de dispositivos de IoT.

Algunos ejemplos de cómo un equipo de TI usa Azure Functions incluyen:

- Azure Event Hub puede ofrecer diferentes eventos que activan funciones relacionadas con condiciones en el entorno de la nube o una cuenta de usuario.
- Un enlace web genérico puede procesar solicitudes HTTP, lo que permite que los

disparadores provengan de sitios web o repositorios de GitHub.

- El tráfico de mensajes puede activar funciones. Por ejemplo, los mensajes que llegan a una cola de Azure Storage pueden activar funciones.
- Los temporizadores también pueden activar funciones, que permiten a los usuarios ejecutar tareas regulares, como la limpieza de archivos, en un horario regular (Rouse, 2017).

Los usuarios también pueden encadenar funciones, o asociar una con otra, para crear interfaces de programa de aplicación (API) y aplicaciones de microservicios más completas.

Azure Functions se integra con las ofertas de software como servicio (SaaS) de Azure, incluidas Azure Cosmos DB, Azure Mobile Apps y Azure Service Bus.

Azure Functions se puede desarrollar en varios lenguajes de programación: C #, F #, Node.js, Python, PHP, batch, bash y cualquier formato de archivo ejecutable. Azure Functions también admite el administrador de paquetes de código abierto NuGet y el Administrador de paquetes de nodos para JavaScript, lo que permite a los desarrolladores usar bibliotecas populares. No todos los lenguajes se encuentran bien soportados, por ejemplo, Python aún no cuenta con soporte, pero se espera que llegue al estado de disponibilidad general (totalmente compatible y aprobado para su uso en producción) en el futuro (Azure-03, 2019).

Azure Functions tiene un sistema de cobro por consumo donde se factura en función del consumo de recursos y las ejecuciones por segundo. Los precios del plan de consumo incluyen una concesión gratuita mensual de 1 millones de solicitudes y 400.000 GB-segundos de consumo de recursos por suscripción en el modelo de precios de pago por uso, para todas las aplicaciones de funciones de esa suscripción.

Una vez superada la cantidad gratuita al mes, será necesario pagar los precios establecidos en la tabla 2.2 (Azure-04, 2019).

.

Tabla 2.2 Precios Azure Functions.

Medidor	Precio	Concesión Gratuita (Por mes)
Tiempo de ejecución	€ 0,000014/GB/s	400.000 GB/s
Total de ejecuciones	€ 0,169 por millón de ejecuciones	1 millón de ejecuciones

Algo muy atractivo es que Azure Functions se puede utilizar con Azure IoT Edge (en la sección 2.4.12.2 se profundizará sobre este tema) sin cargo alguno (Azure-04, 2019).

2.5.7 Otros servicios Saas de AWS:

Existen otros servicios que se pueden utilizar en la Nube para alcanzar los objetivos que están asociados a servicios mencionados anteriormente, en particular, para poder utilizar las funciones Lambda de AWS.

2.5.7.1 AWS ARN

Cada AWS Lambda, así como casi cualquier otro recurso generado en AWS contiene un ARN. Los nombres de recursos de Amazon (ARN) identifican de forma exclusiva los recursos de AWS. Se requiere un ARN cuando sea preciso especificar un recurso de forma inequívoca para todo AWS, como en las políticas de IAM, las etiquetas de Amazon Relational Database Service (Amazon RDS) y las llamadas a la API. (AWS-05, 2019)

2.5.7.2 Amazon API Gateway

Amazon API Gateway es un servicio completamente administrado que facilita a los desarrolladores la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala. Con tan solo unos clics en la consola de administración de AWS, puede crear API REST y API WebSocket que actúen como "puerta delantera" para que las aplicaciones obtengan acceso a datos, lógica de negocio o funcionalidades desde sus servicios *backend*, tales como cargas de trabajo ejecutadas en Amazon Elastic Compute

Cloud (Amazon EC2), código ejecutado en AWS Lambda, cualquier aplicación web o aplicaciones de comunicación en tiempo real (AWS-06, 2019).

API Gateway gestiona todas las tareas implicadas en la aceptación y el procesamiento de hasta cientos de miles de llamadas a API simultáneas, entre ellas, la administración del tráfico, el control de autorizaciones y acceso, el monitoreo y la administración de versiones de API. API Gateway no requiere pagos mínimos ni costos iniciales. Solamente se paga por las llamadas a las API que se reciben y por la cantidad de datos salientes transferidos; además, con el modelo de precios por niveles de API Gateway, puede reducir sus costos a medida que cambie la escala de uso de las API.

La capa gratuita de API Gateway incluye un millón de llamadas a API, un millón de mensajes y 750.000 minutos de conexión al mes durante un máximo de doce meses. Una vez superado este tiempo o esa cantidad de mensajes en un mes aplican las tarifas habituales. Las tarifas habituales dependen del número de peticiones, como se muestra en la tabla 2.3 (AWS-07, 2019).

Tabla 2.3 Tarifas de uso de API Gateway.

Llamadas a la API	
Número de Solicitudes (Por mes)	**Precio (Por millón)**
Primeros 333 millones	3,5 USD
Próximos 667 millones	2,8 USD
Próximos 19 mil millones	2,38 USD
Más de 20.000 millones	1,51 USD

2.5.7.3 AWS IAM

Identity and Access Management es una herramienta que permite la administración de acceso a servicios y recursos con los que se está trabajando en AWS, de

manera segura. Permite además la creación de grupos AWS, así como la creación y administración de usuarios. Por otra parte, es posible utilizar recursos que permitan autorizar o no la entrada de otros a los recursos AWS (AWS-08, 2019).

2.5.8 Servicios IAAS

Entre los distintos tipos de modelos servicio mencionados, se encuentra Iaas, Iaas se podría resumir en pocas palabras en "Máquina Virtual", es decir, el servicio brindado es una máquina virtual con ciertas capacidades de procesamiento, almacenamiento, ancho de banda, entre otros.

A continuación se presenta una tabla comparativa (Tabla 2.4) de precios de distintos proveedores de servicios IaaS según (OVH), (AWS-09, 2019) y (Azure-05, 2019).

Tabla 2.4 Comparación de precios de los servicios IaaS.

Características	Azure	AWS	OVH
2 GB RAM, 1 VCPU	14,24	16,55 *	2,99
4 GB RAM, 2 VCPU	28,23	33,11 *	5,99 **

* Convertido a euros a tasa 1 USD= 0,90 Euros (27/08/19)

** 1 VCPU, 40 GB SSD

Existen muchísimas opciones a escoger, ya sea con más procesador, más memoria RAM, mayor ancho de banda, sin embargo, se colocaron dos de las opciones más económicas de cada uno de estos. En caso de que se desee obtener información acerca de estas opciones, se pueden consultar las páginas web respectivas.

2.5.9 Twilio

Twilio es una plataforma que permite comunicar aplicaciones en la nube y sistemas web. Las API de Twilio permiten comunicar las aplicaciones con los usuarios a través de mensajes de voz, videollamadas, mensajes de texto, mensajes por Whatsapp y más. Se puede contratar un número telefónico para enviar SMS, recibir llamadas o enviar mensajes por Whatsapp (Twilio-01, 2019) (Twilio-02, 2019; Twilio-03, 2019).

Twilio tiene SDK desarrollados para poder ser utilizado con varios lenguajes de programación, en caso de que se desee utilizar un lenguaje no soportado, se puede utilizar una API Rest a través de consultas HTTP, lo que lo hace muy versátil e integrable.

2.5.10 Telegram

Telegram es una aplicación de telefonía móvil que permite la comunicación a través de mensajería instantánea y VOIP. A diferencia de Whatsapp, esta tiene una plataforma abierta y gratuita a través de la cual los desarrolladores son capaces de conectar con la red, por medio de la API que este presenta.

Para poder enlazar una aplicación con telegram, es necesario registrar un bot, que se puede hacer fácilmente a través de un bot ya existente en telegram llamado "Bot Father" (Telegram, 2019).

2.5.11 Azure IoT Hub

IoT Hub es un servicio administrado, hospedado en la nube, que actúa como centro de mensajes para comunicaciones bidireccionales entre la aplicación de IoT y los dispositivos que administra. Se puede usar Azure IoT Hub para compilar soluciones de IoT con comunicaciones confiables y seguras entre millones de dispositivos de IoT y un *back-end* de solución hospedado en la nube. Se puede conectar virtualmente casi cualquier dispositivo a IoT Hub (Azure-06, 2019).

IoT Hub es compatible con comunicaciones del dispositivo a la nube y viceversa. IoT Hub es compatible con varios patrones de mensajes, como telemetría de dispositivo a la nube, carga de archivos desde dispositivos y métodos de respuesta a solicitudes para controlar los dispositivos desde la nube. La supervisión de IoT Hub ayuda a conservar el estado de la solución, ya que realiza el seguimiento de eventos, como la creación, los errores y las conexiones de dispositivos (Azure-07, 2019).

Las funcionalidades de IoT Hub ayudan a compilar soluciones de IoT escalables y completas, como la administración de equipos industriales que se usan en fabricación, el seguimiento de activos valiosos en el sector sanitario y la supervisión del uso de compilaciones en oficina.

Entre los lenguajes compatibles con IoT Hub se incluyen C, C#, Java, Python y Node.js.

IoT Hub y los SDK de dispositivo admiten los siguientes protocolos de conexión de dispositivos:

- HTTPS
- AMQP
- AMQP sobre WebSockets
- MQTT
- MQTT sobre WebSockets

2.5.11.1 IoT Hub Device Provisioning

El servicio IoT Hub Device Provisioning es un servicio auxiliar de IoT Hub que ofrece un aprovisionamiento *just-in-time*, sin intervención del usuario, de la instancia apropiada de IoT Hub, que permite a los clientes aprovisionar millones de dispositivos de forma segura y escalable.

IoT Hub ofrece otros servicios como los mencionados en la tabla 2.5:

Tabla 2.5 Servicios IoT Hub.

CARACTERÍSTICA	BASIC	ESTÁNDAR/GRATIS
Telemetría del dispositivo a la nube	✓	✓
Identidad por dispositivo	✓	✓
Enrutamiento de mensajes, integración con Event Grid	✓	✓
Protocolos HTTP, AMQP, MQTT	✓	✓
Compatibilidad con DPS	✓	✓
Supervisión y diagnóstico	✓	✓
Flujos de dispositivos VERSIÓN PRELIMINAR		✓
Mensajería de la nube al dispositivo		✓
Administración de dispositivos, dispositivos gemelos, módulos gemelos		✓
IoT Edge		✓

Los servicios anteriores permiten aumentar el potencial de IoT Hub, entre ellos a continuación se presenta información más detallada acerca de IoT Edge.

2.5.11.2 IoT Edge

Azure IoT Edge mueve el análisis en la nube y lógica de negocios personalizada a los dispositivos para que la organización pueda centrarse en la información empresarial en lugar de en la administración de los datos. Se puede escalar horizontalmente la solución de IoT mediante el empaquetado de la lógica de negocio en contenedores estándar y, después, podrá implementar esos contenedores en cualquiera de los dispositivos y supervisarlo todo desde la nube (Azure-08, 2019).

Los análisis son un valor añadido empresarial para las soluciones de IoT, pero no es necesario que todos los análisis estén en la nube. Si se quiere responder a emergencias lo antes posible, se pueden ejecutar las cargas de trabajo de detección de anomalías en el borde. Si se quiere reducir los costos de ancho de banda y evitar la transferencia de terabytes de datos sin procesar, puede limpiar y agregar los datos localmente y enviar solo la información a la nube para su análisis.

Azure IoT Edge está formado por tres componentes:

- Los módulos de IoT Edge son contenedores que ejecutan servicios de Azure, de terceros o código propio del usuario. Se implementan en los dispositivos de IoT Edge y se ejecutan en ellos.
- El entorno en tiempo de ejecución de IoT Edge se ejecuta en todos los dispositivos de IoT Edge y administra los módulos que se implementan en cada dispositivo.
- Una interfaz basada en la nube permite supervisar y administrar los dispositivos de IoT Edge de forma remota.

Módulos de IoT Edge

Los módulos de IoT Edge son unidades de ejecución implementadas como contenedores compatibles con Docker, que ejecutan la lógica de negocios en los dispositivos perimetrales. Se pueden configurar varios módulos para que se comuniquen entre sí al crear una canalización de procesamiento de datos. Se pueden desarrollar módulos personalizados o empaquetar determinados servicios de Azure en módulos que proporcionen información sin conexión y en el dispositivo perimetral.

Inteligencia artificial perimetral

Azure IoT Edge permite implementar el procesamiento de eventos complejos, el aprendizaje automático, el reconocimiento de imágenes y otros tipos de inteligencia artificial de gran valor sin necesidad de escribirla internamente. Los servicios de Azure, como Azure Functions, Azure Stream Analytics y Azure Machine Learning, se pueden ejecutar de manera local mediante Azure IoT Edge. Sin embargo, esto no se limita únicamente a los servicios de Azure. Cualquier desarrollador puede crear módulos de inteligencia artificial y ponerlos a disposición de la comunidad mediante Azure Marketplace.

Cuando se desee implementar código propio en los dispositivos, Azure IoT Edge también lo admite. Azure IoT Edge aplica el mismo modelo de programación que los demás servicios de IoT de Azure. El mismo código se puede ejecutar en un dispositivo o en la nube. Azure IoT Edge es compatible con Windows y Linux, por lo que se podrá codificar para múltiples plataformas. Admite Java, .NET Core 2.0, Node.js, C y Python, por lo que los desarrolladores pueden crear código en un lenguaje que ya conozcan y usar la lógica de negocios existente.

Precios

IoT Hub cobra por unidad de IoT, en la tabla 2.6 se muestra la cantidad de mensajes permitida por unidad de IoT al día, así como sus precios actuales (2019) (Azure-09, 2019)

Tabla 2.6 Precios IoT Hub.

TIPO DE EDICIÓN	PRECIO POR UNIDAD DE IOT HUB (AL MES)	NÚMERO TOTAL DE MENSAJES AL DÍA POR UNIDAD DE IOT HUB	TAMAÑO DEL MEDIDOR DE MENSAJES
B1	€8,433	400.000	4 KB
B2	€42,165	6.000.000	4 KB
B3	€421,650	300.000.000	4 KB

Nivel Standard

TIPO DE EDICIÓN	PRECIO POR UNIDAD DE IOT HUB (AL MES)	NÚMERO TOTAL DE MENSAJES AL DÍA POR UNIDAD DE IOT HUB	TAMAÑO DEL MEDIDOR DE MENSAJES
Gratis	Gratis	8.000	0,5 KB
S1	€21,083	400.000	4 KB
S2	€210,825	6.000.000	4 KB
S3	€2108,25	300.000.000	4 KB

Una unidad de IoT Hub representa una instancia de IoT Hub. El número de unidades necesarias depende del número de mensajes necesarios para la solución de IoT. Por ejemplo, cada unidad de las instancias S1 o B1 de IoT Hub puede controlar 400.000 mensajes al día, si se necesita enviar hasta 800.000 mensajes al día, serían necesarias 2 instancias S1 o dos instancias B1. El Nivel se debe escoger en base a los servicios necesarios, basándose en la tabla 2.6.

2.5.12 Azure Container Registry

Azure Container Registry es un servicio privado, de código abierto, administrado del Registro de Docker que usa Docker Registry 2.0. Permite crear y mantener los registros de Azure Container para almacenar y administrar las imágenes privadas de contenedores Docker (Azure-10, 2019).

Azure Container Registry se puede usar para desarrollo de contenedores y canalizar implementaciones existentes, o bien se puede usar Azure Container Registry Tasks para compilar imágenes de contenedor en Azure. Se puede compile a petición o automatizar completamente las compilaciones con desencadenadores como las

confirmaciones del código fuente y las actualizaciones de la imagen de base.

2.5.13 Azure Logic Apps

Azure Logic Apps consiste en una plataforma de integración en la nube que permite la programación, automatización y organización de tareas y procesos. Se utiliza cuando es necesario integrar aplicaciones, datos y servicios. Es un servicio que facilita el diseño y compilación de soluciones escalables (Azure-11, 2019).

Un factor a resaltar dentro de este servicio es la utilización de conectores, ya que presenta una gran galería entre la que se puede elegir cuál usar. Estos permiten el uso de desencadenadores o acciones, o una unión de ambos, lo que ayuda a la creación de aplicaciones lógicas capaces de acceder y procesar datos en tiempo real y de forma segura (Rodríguez, 2018).

2.5.14 Bases de datos o plataformas de almacenamiento

Uno de los objetivos planteados en el presente trabajo es poder utilizar los datos de las cámaras para entrenar modelos de redes del tipo R-CNN que permitan predecir el tráfico en las vías donde se encuentran ubicadas estas cámaras; para lograr entrenar los modelos es necesario disponer de los datos, que se utilizarán para el entrenamiento de los modelos y que provienen de las propias cámaras, es por esto que es necesario almacenar la información recibida de las cámaras en una base de datos.

Entonces, surge la pregunta de ¿Qué motor de bases de datos utilizar? Existen muchas opciones disponibles en el mercado, unas gratuitas, otras económicas y otras no tan económicas. A continuación, se analizan algunas de las opciones disponibles.

2.5.14.1 Azure Cosmos DB

Azure Cosmos DB es un servicio de base de datos totalmente administrado con distribución global llave en mano y replicación de arquitectura multi-maestro transparente. Con latencias de lectura y escritura inferiores a diez milisegundos en el percentil 99, escalado automático y elástico del procesamiento y el almacenamiento en todo el mundo, alta disponibilidad del 99,999 % y cinco opciones de coherencia bien definidas, todo ello con el respaldo de acuerdos de nivel de servicio extraordinarios.

Precio mínimo: ~$5.84/mes +$0.25 por GB almacenado por mes (Azure-12, 2019).

2.5.14.2 Amazon DynamoDB

Es una base de datos de claves-valor y documentos que ofrece rendimiento en milisegundos de un solo dígito a cualquier escala. Se trata de una base de datos multi-región y multi-maestro completamente administrada, con seguridad integrada, copia de seguridad y restauración, y almacenamiento de caché en memoria para aplicaciones a escala de Internet. DynamoDB puede gestionar más de 10 billones de solicitudes por día y admite picos de más de 20 millones de solicitudes por segundo (AWS-10, 2019).

2.5.14.3 MongoDB Atlas

Servicio MongoDB alojado en la nube en AWS, Azure y GCP. Se puede implementar, operar y escalar una base de datos MongoDB con solo unos pocos clics. Dado que se puede implantar en distintas nubes, los precios varían, sin embargo, este ofrece una opción gratuita con 512 MB de almacenamiento en cualquiera de las 3 nubes (MongoDB, 2019).

2.5.14.4 Azure Blob Storage o AWS S3

Tanto Azure Blob Storage como AWS S3 son servicios que permiten almacenar ficheros en la nube; se paga por GB almacenado y por petición de lectura en ambos casos, es decir, cargar información no cuesta nada, pero descargarla sí. Dado que la mayoría de los modelos se pueden entrenar a partir de ficheros txt o csv, en realidad no es necesario utilizar una base de datos. No es una solución práctica dado que la idea es ir añadiendo datos continuamente a los archivos y que se puedan consultar bien sea por fecha, lugar, cantidad de tráfico entre otros (Azure-13, 2019) (Amazon, 2019).

2.5.15 Azure Custom Vision

Es un servicio de Azure que permite a los desarrolladores la opción de acceder a algoritmos capaces de lograr el reconocimiento de una imagen y dar como respuesta información referente a la misma. Computer Vision permite utilizar la dirección URL de una imagen o cargarla directamente para poderla analizar, pero, al mismo tiempo,

permite que el reconocimiento vaya más allá de características simples de la imagen, ya que no solo es capaz de definir solamente el objeto sino alguna propiedad más explícita de la misma. Por ejemplo, puede reconocer entre varios objetos, cuál de ellos es un coche, pero también puede ser capaz de reconocer que el coche tiene un color específico con respecto a otro (Azure-14, 2019).

A través del uso de un SDK nativo o un API REST es posible usar Computer Vision. El uso de este servicio permite la realización de diversas actividades:

- **Análisis de imágenes para obtener información:** es posible el análisis completo de imágenes para poder obtener de ellas información detallada sobre todas sus características y propiedades.
    - Etiquetar características visuales.
    - Detectar objetos.
    - Detección de las marcas.
    - Clasificar una imagen.
    - Describir una imagen.
    - Detectar tipos de imagen.
    - Detectar contenido específico del dominio.
    - Detectar la combinación de colores.
    - Generar una miniatura.
    - Obtener el área de interés.
- **Extracción de texto en las imágenes:** es capaz de reconocer, mediante modelos actualizados, texto que se encuentra sobre superficies de otro color o fondo. Es posible detectar hasta 25 idiomas.
- **Moderación del contenido de las imágenes:** es posible su utilización para reconocer imágenes que están fuera de lo permitido, aplicando filtros capaces de detectar contenido no apropiado.
- **Uso de contenedores:** permite la utilización de contenedores Docker para reconocer texto impreso y manuscrito localmente, para ello el Docker debe ser el más cercano posible a los datos.

- **Requisitos de imágenes:** es necesario que las imágenes procesadas cumplan una serie de requisitos como tamaño (menor de 4 MB), formato (JPEG, PNG, GIF o BMP) y dimensiones (mayores de 50 x 50 píxeles).
- **Seguridad y privacidad de los datos:** es necesario, como en todo procedimiento, estar al tanto de las leyes que rigen el uso de este servicio.

2.6 Herramientas útiles de desarrollo

Para el desarrollo de la propuesta que se plantea se han evaluado las herramientas que se mencionan a continuación.

2.6.1 Visual Studio Code

Visual Studio Code es un editor de código fuente ligero pero potente que se ejecuta en su escritorio y está disponible para Windows, macOS y Linux. Viene con soporte incorporado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros lenguajes (como C ++, C #, Java, Python, PHP, Go) y tiempos de ejecución (como .NET y Unity) (VisualStudio-01, 2019).

Entre las extensiones, se encuentran:

- **Azure IoT Edge Tools:** Facilita la codificación, creación, implementación y depuración de sus soluciones IoT Edge, al proporcionar un amplio conjunto de funcionalidades:
    - Permite crear nuevo proyectos Azure IoT Edge dirigido a diferentes plataformas (Linux amd64, Linux arm32v7, Windows amd64)
    - Permite agregar nuevos módulos IoT Edge a la solución.
    - Permite editar, crear y depurar módulos de IoT Edge localmente en Visual Studio Code.
    - Permite crear e implementar imágenes acoplables de módulos IoT Edge.
    - Permite ejecutar módulos de IoT Edge en un simulador local o remoto.
    - Permite administrar dispositivos y módulos de IoT Edge en IoT Hub

(con Cloud Explorer) (VisualStudio-02, 2019).

- **AWS Toolkit for Visual Studio Code:** Facilita la creación, depuración e implementación de aplicaciones en Amazon Web Services. El kit de herramientas proporciona una experiencia integrada para desarrollar aplicaciones sin servidor, incluida la asistencia para comenzar, depurar paso a paso e implementar desde el IDE (AWS-11, 2019).
- **Azure Functions for Visual Studio Code:** Permite crear, depurar, administrar e implementar Azure Functions directamente desde VS Code.Contiene proyectos de muestra que sirven como plantilla para empezar fácilmente el desarrollo (VisualStudio-03, 2019).

2.6.2 Visual Studio

Visual Studio es un entorno de desarrollo integrado, IDE por sus siglas en inglés. Es un editor redefinido y optimizado de códigos que permite construir y depurar webs y aplicaciones en la nube. Este software libre constituye un conjunto de herramientas y otras tecnologías de desarrollo que se basa en componentes, de manera que sea posible la creación de aplicaciones eficaces y de alto rendimiento (Microsoft, 2019).

2.6.3 Jupyter Notebooks

Jupyter Notebook es una aplicación web de código abierto que permite crear y compartir documentos (cuadernos) que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo. Los usos incluyen: limpieza y transformación de datos, simulación numérica, modelado estadístico, visualización de datos, aprendizaje automático entre otros muchos. Jupyter admite más de 40 lenguajes de programación, incluidos Python, R, Julia y Scala (Jupyter, 2019)

Los cuadernos se pueden compartir con otras personas mediante correo electrónico, Dropbox, GitHub y Jupyter Notebook Viewer. El código puede producir una salida rica e interactiva: HTML, imágenes, videos, LaTeX y tipos MIME personalizados lo cual hace que sean muy cómodos para enseñar o explicar códigos.

Jupyter Notebooks permite usar herramientas de big data, como Apache Spark, de

Python, R y Scala y explore esos mismos datos con pandas, scikit-learn, ggplot2, TensorFlow de una manera muy cómoda ya que se puede ejecutar el código de manera interactiva por partes.

Jupyter Notebooks se volvió un estándar, los principales proveedores de cómputo en la nube tienen servicios de instancias de Jupyter Notebooks para realizar muchos procesos, especialmente los orientados a Machine Learning.

2.6.4 Node Red

Node Red es una herramienta de programación que permite el enlace de dispositivos, APIs y servicios online, de manera que éstos puedan conectarse entre sí para lograr una comunicación entre ellos. Las relaciones se muestran de forma visual y se establecen a través de editores basados en navegadores (Node-Red, 2019).

Node Red permite la instalación de diferentes paquetes o librerías de procesamiento y gestión y cuenta con una comunidad de desarrolladores muy amplia. Estos paquetes se instalan con “npm install”. Entre ellos cabe mencionar Red-Bot, este permite crear bots integrados con unos diversos proveedores destacando Telegram, Slack, FaceBook o Alexa, entre otros, sin una alta dificultad de escritura de códigos. Gracias a que Node Red se programa de manera gráfica, es posible desarrollar de forma muy sencilla un bot conversacional. Además, gracias a Red-Bot, es posible integrar múltiples plataformas utilizando la misma lógica.

2.6.5 Atajos de Siri

Son aplicaciones que permiten la configuración de acciones que se desencadenarán a través de Siri en dispositivos con iOS. Permiten además realizar una gran cantidad de acciones diferentes, entre ellas hacer consultas HTTP y, una vez configurado el atajo, se puede compartir con otros dispositivos (Apple, 2019).

2.6.6 Ngrok

Ngrok es una herramienta que permite exponer hacia el exterior (internet público) los servidores locales detrás de túneles y firewalls por medio de túneles seguros. Esta

herramienta es útil para realizar demostraciones o pruebas de sitios web sin implementar o desplegar, además es posible la construcción de consumidores webhook en máquinas de desarrollo, la prueba de aplicaciones que se conectan a su backend y se ejecutan de manera local. Otros de los beneficios del uso de esta herramienta son la ejecución de servicios personales en la nube y que provee direcciones estables para dispositivos conectados (Ngrok, 2019).

Capítulo 3 Metodología y análisis

Tras la revisión de técnicas, metodologías, procedimientos y herramientas realizada en el capítulo previo, en éste se lleva a cabo la selección de las mismas con el objetivo de diseñar el sistema propuesto, siendo necesario identificar correctamente cada uno de los puntos de la arquitectura del sistema. A continuación, se explica la evolución de forma secuencial del proceso desde su inicio hasta su finalización, las pruebas realizadas y los análisis necesarios para cada prueba con vistas a proporcionar la solución integrada. Siguiendo el paradigma IoT, en la sección 3.1 se analizan las cuestiones relacionadas con la captura y procesamiento de imágenes bajo la consideración de las redes neuronales convolucionales, así como su envío a la nube. A la vez se analizan los dispositivos de captura compatibles con los requisitos de proceso y su integración con fines de procesamiento. En la sección 3.2 se analiza el almacenamiento de datos en función del repositorio. Tras lo cual, el siguiente paso consiste en decidir el tipo y estructura de datos a almacenar, cuestión ésta que se aborda en la sección 3.3. En la sección 3.4 se analiza la arquitectura API REST (Representational State Transfer) con el fin de proporcionar la adecuada accesibilidad a los datos desde distintas aplicaciones y plataformas. Tras el almacenamiento y disposición de los datos, en la sección 3.5 se aborda la problemática relacionada con su utilización, principalmente en lo que se refiere a los modelos de regresión logística, con tal propósito se analizan las posibilidades de configuración, contratación y seguridad de la máquina virtual IaaS. En la sección 3.6 se plantea la integración del módulo de análisis de datos por regresión en el paradigma IoT, una vez que éstos se encuentran disponibles y accesibles. En la sección 3.7, tras la disposición de los datos y la integración del módulo de predicción para su análisis, se realiza el correspondiente estudio para avanzar en la mejora de la API Rest con tecnologías *serverless*. Finalmente, en la sección 3.8 se aborda la implantación del correspondiente servicio para ofrecer información útil y amigable a nivel de usuario, cerrando así el ciclo de la aplicación IoT.

3.1 Captura y procesamiento de datos

Como primer paso fue necesario definir cómo obtener los datos y dónde procesarlos, el proceso fue llevado a cabo realizando diversas pruebas de concepto.

Se decidió utilizar como dispositivo de entrada una cámara dispuesta de modo tal que sea capaz de identificar los vehículos en las vías, para lo cual se plantea una red neuronal capaz de detectar los vehículos. A continuación, se presenta el orden en el que se fue probando tanto la cámara, como el dispositivo donde se procesarán los datos, su sistema operativo y la red neuronal a utilizar.

Red Neuronal Convolucional Alexnet con Matlab

El proceso para elegir la red neuronal a utilizar requirió una gran cantidad de tiempo; se comenzó con la realización de pruebas con el modelo pre-entrenado AlexNet con MATLAB R2019b en Windows 10. Con esta red en un equipo con procesador Intel Core i7 8550U @1.80 GHz y 8 GB de RAM, analizar un frame podía llevar hasta 80 segundos. Se realizaron varias pruebas sin observar mejoras significativas.

Red Neuronal Mask RCNN con Python en Windows 10

Posteriormente se probó en Python con una red neuronal convolucional basada en Mask-RCNN en el mismo equipo con Windows 10, se lograron tiempos de procesamiento de 40 segundos, lo cual supone una mejora significativa al conseguir la mitad de tiempo que con AlexNet en MATLAB. La figura 3.1 muestra una captura de pantalla bajo el procesamiento indicado, observándose sobre cada vehículo el porcentaje de acierto respecto de su identificación. Este código es de código abierto con licencia MIT, disponible en https://github.com/matterport/Mask_RCNN)

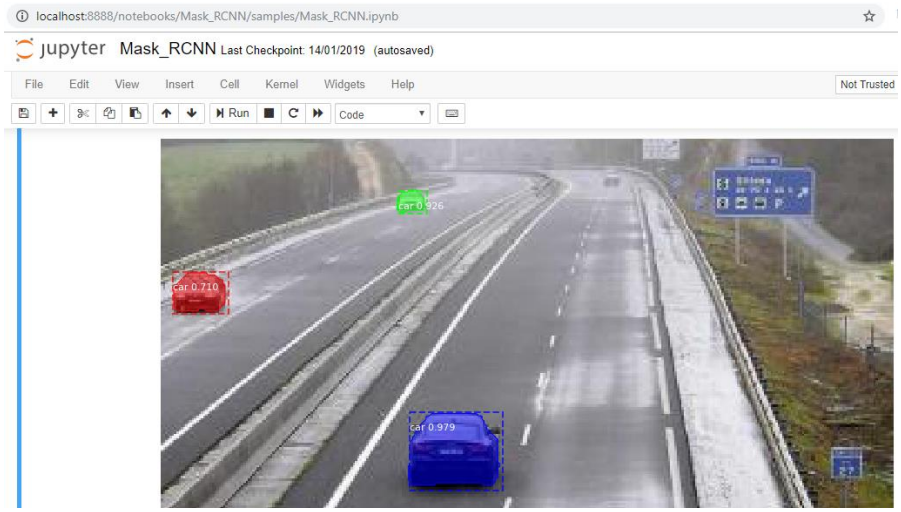

Figura 3.1 Resultados de identificación mediante una R-CNN.

Red Neuronal Mask RCNN con Python en Ubuntu 16.04

Luego se decidió probar con la misma red neuronal (Mask RCNN) en el mismo equipo (Procesador Intel Core i7 8550U @1.80 GHz y 8 GB de RAM), pero utilizando el sistema operativo Ubuntu 16.04, con el mismo código y en el mismo equipo, pero con Ubuntu se lograran tiempos de procesamiento del orden de 10 segundos, lo cual resulta en una mejora significativa.

Red Neuronal Mask RCNN con Python en Google Colab

Como el resultado en Ubuntu de 10 segundos seguía pareciendo mucho tiempo, se decidió probar el procesamiento en la nube y, dado que el programa se ejecutaba en Python, se decidió probar con Google Colab. Gracias al procesamiento en GPU que ofrece Google Colab, se pudo procesar con prácticamente el mismo código en tiempos menores a 1 segundo por *frame*, lo que se puede ver en la figura 3.2.

Figura 3.2 Tiempos de procesamiento de una R-CNN en Google Colab.

Cabe destacar que todas las pruebas se hicieron con un mismo video grabado en el que circulan varios vehículos por una vía, es decir, las comparaciones no se hicieron con videos tomados en vivo con alguna cámara web.

El uso de Google Colab (en febrero del 2019) llevó a la conclusión de que claramente el problema de la tardanza en el cómputo de las imágenes se debe a la limitación de hardware del equipo utilizado para probar, es importante aclarar que Google Colab no permite ejecuciones de más de 12 horas continuas y si el *kernel* está ocioso por cierto tiempo, se pueden borrar los datos almacenados tanto en memoria como en disco. Es decir, tanto las variables como todos los archivos y dependencias instaladas con "pip install" pueden eliminarse, es por esto que, si se va a ejecutar el código en Google Colab, vale la pena dedicar el principio del código a la instalación de todas las dependencias *pip* utilizando el comando "!pip install" dentro del mismo código en Python. Recuérdese que el comando pip install verifica si ya está instalado el módulo y en caso de que ya esté instalado, no lo vuelve a instalar, así que no se pierde mucho tiempo al ejecutar estos comandos.

El hecho de que se borre todo lo que está en disco en Google Colab no es un problema mayor dado que la velocidad de descarga de las dependencias puede alcanzar

fácilmente los 500 Mbps, consiguiendo así velocidades de descarga suficientemente rápidas.

Envío de imágenes desde la cámara a la Nube

Como hasta el momento, la mejor opción parecía ser utilizar la Nube para procesar las imágenes, se decidió seguir intentando con Google Colab, pero esta vez no con videos grabados previamente, sino con videos en tiempo real bajo la modalidad de *streaming*.

Se probó con videos alojados en YouTube y con algunas cámaras IP públicas, el problema principal observado aquí fue que al trabajar con videos en *streaming*, el búfer puede colapsar si no se programa con cuidado ya que un segundo de video puede contener hasta 60 fps y un *frame* puede demorar hasta 3 segundos en analizarse, es decir, que por cada 3 segundos se podrían encolar hasta 179 *frames*, como cada *frame* tiene todos los píxeles en sus tres colores (RGB), si el video es de alta definición, es posible que colapse la memoria en muy poco tiempo. Es por esto que fue necesario decidir entre dos opciones:

A. Analizar *frame* por *frame* sin importar si se retrasa el análisis con respecto al tiempo real. En este caso habría que grabar el video en un archivo y enviar ese archivo guardado a la red neuronal.
B. Analizar únicamente el último *frame*, en este caso, habría que limitar el tamaño del búfer para que pueda almacenar únicamente 1 *frame*. Esto impide que se puedan contar todos los coches, así como el poder hacer un seguimiento (*tracking*) de los vehículos.

El resultado de las pruebas con ambas opciones fue exitoso, por lo que es simplemente cuestión de decidir qué se prefiere, esto es, analizar videos en streaming en tiempo real o analizar por lotes cada video, asumiendo que en este caso se requiere mucho más tiempo de cómputo que en el video. Se decidió optar por la opción de analizar el último *frame* recibido para no retrasar el análisis, teniendo en cuenta que no se pretende identificar cada vehículo de forma individual ni se pretende realizar un sistema de conteo de todos los vehículos sino de medición de densidad vehicular en general.

Descarga en paralelo del último frame disponible

En algunas de las pruebas realizadas, el código en Python se encargaba de analizar el *frame* y posteriormente descargarlo de la cámara IP. Se observó que esto no era conveniente, ya que no es recomendable esperar a que se haya terminado de analizar una imagen para empezar a descargar la siguiente, por lo que se decidió desarrollar un código específico que, en paralelo, descargue el último *frame* y éste se guarde, de modo que el código que analiza las imágenes con la red neuronal sólo se encargue de esto. Con este código, se alcanzaron de nuevo los tiempos de proceso próximos a 1 segundo por frame.

Creación de una API Rest para poder acceder a los datos

Dado que se deseaba poder acceder a los datos desde cualquier aplicación, se decidió crear una API Rest. Como el código es en Python, se probó con Flask para publicar los recursos a través de consultas HTTP. Utilizar Flask fue muy sencillo, realmente lo complicado fue saber cómo se debían asignar los puertos y los certificados ssl para que la comunicación fuera cifrada con SSL.

Debido a que se desarrolló una API, se realizaron pruebas utilizando una cámara web previamente usada localmente sin una IP pública accesible desde internet, es decir, se decidió probar que en lugar de ser el código Python en el dispositivo el encargado de realizar la solicitud de descarga, sea el dispositivo con la cámara el que envíe la imagen a la API.

La idea, entonces, consiste en crear una API que reciba las fotos de la cámara para procesar las imágenes en Google Colab y posteriormente, se pueda preguntar a la API la cantidad de tráfico medida; un esquema que representa la solución propuesta hasta el momento se puede observar en la figura 3.3.

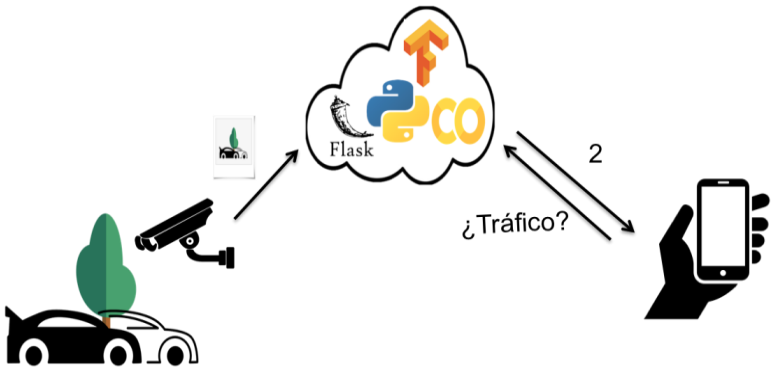

Figura 3.3 Esquema de consulta de datos desde una aplicación.

Pruebas con varias instancias de la red neuronal en Google Colab

Una vez obtenida una solución suficientemente completa para recolectar los datos, enviarlos y analizarlos, se decidió probar la escalabilidad de esta solución. Los resultados obtenidos no se adecuaron a las expectativas, ya que como máximo era posible analizar 3 imágenes sin causar problemas de colapso de memoria RAM en Google Colab, esto provocó que se buscaran opciones, no sólo para fines académicos o de desarrollo, sino como una opción para entornos reales productivos.

Un dato importante a considerar en este sentido, es que a pesar de que Google Colab solo pudo analizar hasta 3 imágenes a la vez, los tiempos de análisis por *frame* llegaron a alcanzar los 8-10 segundos por *frame* en cada video. Es decir, suponiendo que con Google Colab se pudieran analizar videos de varias cámaras, cada análisis individual demorará mucho más tiempo, lo cual no es conveniente plantear esta opción desde una perspectiva de procesamiento en tiempo real.

Investigación de opciones en la Nube para procesamiento de imágenes

Dado que no se ve factible utilizar Google Colab en un entorno real, se decidió buscar otras opciones. Se encontraron varias opciones en la Nube, el problema de la mayoría de estos servicios es que son muy costosos, la mayoría cobra por tiempo de uso o por peticiones. En un entorno IoT como el planteado en este trabajo y con vistas a su implantación en un entorno real, no sería muy factible por temas de costo. Una opción que se podría utilizar es reducir la frecuencia de muestreo y no medir el tráfico cada segundo ni cada minuto, sino cada cierto intervalo de tiempo que cumpliera con los

requisitos de la instalación, lo cual evitaría la medición correcta del TPD, el TPDA, el VHP y los otros valores de interés. Entre las opciones que se consideraron al respecto para realizar este procesamiento se incluyen las prestaciones proporcionadas por AWS SageMaker y Azure ML Studio (AWS-12, 2019) (Azure-15, 2019).

Uso de dispositivos con capacidad de procesamiento

A pesar de que utilizar la Nube parece ser una opción apropiada por la capacidad de procesamiento que se puede conseguir, al analizar los costos que puede conllevar deja de parecer tan buena opción, es por esto que se vuelve a la opción inicial de procesar de forma local las imágenes y enviar los datos a la nube, sólo que esta vez se plantea de una manera diferente, a saber: cada dispositivo con cámara procesa las imágenes y envía sus resultados a la nube para almacenar los datos y ser servidos posteriormente.

Por lo expuesto en este sentido, se decidió tomar como aceptable el tiempo que sea necesario para procesar las imágenes en cada dispositivo; como cada dispositivo se encarga de analizar sus propias imágenes, no va a haber retardo a la hora de aumentar la cantidad de dispositivos, es decir; si un dispositivo demora 10 segundos para analizar una imagen, 1000 dispositivos también demorarán 10 segundos para analizar sus propias 1000 imágenes.

Tras este planteamiento no es necesario decidir cómo y dónde guardar los datos obtenidos de manera escalable, en cualquier caso, más adelante se ofrece una solución a esta cuestión.

Uso de Azure IoT Edge en dispositivos con procesamiento de borde

Como se mencionó anteriormente, procesar las imágenes en cada dispositivo también es algo factible, aunque el procesamiento no es tan rápido como en la nube, sin embargo, sí podría resultar más económico. Para ello se necesitaría un Hardware con ciertas prestaciones para poder realizar procesamiento de redes neuronales en él. Es decir, tener cierto procesamiento de borde y no enviar la imagen, sino únicamente el análisis de la misma. Teniendo en mente todo lo anterior, se decidió buscar la manera de poder aprovisionar, supervisar y actualizar de una manera cómoda los dispositivos que

van a estar enviando datos, por esta razón se decidió probar con Azure IoT Edge.

Como primer paso se planteó la idea de utilizar una raspberry Pi 3 con una cámara conectada a ella, pero, como el hardware de este dispositivo es tan limitado, ejecutar modelos de redes neuronales no es recomendable, por lo que se descartó completamente esta idea sin ni siquiera intentarlo.

Posteriormente, se encontró otra opción basada en el dispositivo EIC MS Vision 500, que, al momento de hacer las pruebas, aún no estaba disponible en el mercado (Azure-16, 2019).

Se logró tener acceso anticipado a un dispositivo EIC MS Vision 500 y, una vez lograda la implementación de un contenedor Docker con un modelo entrenado, se obtuvieron resultados interesantes. Se comprobó que realmente ésta es una buena elección para incluirla en la arquitectura de la solución finalmente propuesta.

Otro punto a favor que tiene esta cámara es que, en realidad, el modelo que se ejecuta está implementado en TensorFlow ejecutado con Python, lo cual permite que se programen acciones con cierta lógica en la misma cámara. Es decir, no sólo se utiliza el modelo para clasificar o detectar la presencia de objetos en las imágenes, sino que se puede utilizar esta información para enviar a Azure IoT Edge o a una API Rest los datos que realmente sean útiles.

Utilizar esta cámara con Azure IoT Edge requirió una gran cantidad de tiempo de programación, investigación, configuración y pruebas, debido a que:

- El producto aún no se encontraba en el mercado para la venta.
- Existe muy poca documentación.
- La documentación que existe tiene muchos errores y muchos enlaces caídos.
- Muchos códigos no funcionaban en su versión beta, requiriendo correcciones. De hecho, se realizaron varios Pull Request en Github con propuestas de modificación, de las cuales varias fueron aceptadas. Estos Pull Request se hicieron con el usuario vincenzocaschetto en el GitHub de microsoft/vision-ai-developer-kit.
- La cámara quedó varias veces en un estado bloqueado del que costó muchísimo salir.

- Su uso requiere la integración de diversas tecnologías y servicios, entre ellos, Docker, Azure Container Registry, Azure IoT Hub y Azure IoT Edge.

Para utilizar esta cámara fue necesario utilizar Visual Studio Code y Docker para poder crear un contenedor con el código en Python y el modelo de red neuronal a utilizar. La cámara implementa contenedores Docker almacenados en Azure Container Registry.

Para lograr esto, fue necesario crear un grupo de recursos en Azure. Dentro de este grupo se creó el Azure Container Registry y se utilizó la contraseña para poder entrar en Visual Studio Code en Docker con el usuario y contraseña generado con ACR.

En primera instancia, se utilizó un modelo creado en Azure Custom Vision. Entrenar este modelo fue muy sencillo ya que se hizo todo desde el portal web de este servicio donde tan solo hubo que colocar imágenes e indicar sus ubicaciones. La ejecución de este modelo fue muy rápida obteniéndose resultados excelentes, alcanzando tiempos de menos de un segundo por frame.

Esta solución aparentemente se ve como buena opción, siendo necesario determinar en qué lugar almacenar los datos, dónde y cómo analizarlos y servirlos. En las siguientes secciones se aborda esta cuestión.

3.2 Almacenamiento de datos

Dado que existen muchas formas de almacenar los datos, tales como bases de datos relacionales, bases de datos no relacionales, ficheros de texto plano, entre otras, es necesario decidir cómo se desea almacenar y determinar qué servicio o proveedor utilizar.

Si los datos se almacenan en un fichero en un servicio como AWS S3 o Azure Blob Storage, donde no se cobra por su almacenamiento, pero sí por su lectura, se pagaría sólo en este último caso. El problema es que como es un fichero, cada vez que se quiera consultar, habrá que descargar todo el archivo y pagar cada vez más por consultar, debido a que cada vez el fichero contendrá más datos, lo cual no resulta factible como una solución a largo plazo.

Si se desea realmente almacenar en ficheros en algunos de estos servicios, una manera de reducir los gastos es crear ficheros nuevos cada día, semana o mes, lo cual a

pesar de la economía resulta un proceso incómodo y quizás poco eficiente.

Por todas dichas razones, resumidas en practicidad y economía, no conviene almacenar los datos en ficheros, sino en una base de datos ya sea relacional o no. Utilizar una base de datos relacional puede ser muy conveniente para desarrollar el código fácilmente, ya que las consultas se harían con cursores que recorren la tabla. En cambio, utilizar una base de datos no relacional no es tan sencillo para programar el código que va a utilizar los datos consultados. Sin embargo, el hecho de que se pueda tener un esquema flexible en las BBDD no relacionales es muy conveniente a largo plazo en soluciones IoT.

Como en el presente trabajo se presenta una solución IoT y se prefiere dedicar un poco más de tiempo al desarrollo del código en pro de tener una solución que permita ser integrable y evolucionable, se optó, tomando en cuenta los costos, por utilizar la capa gratuita de MongoDB Atlas en Azure con 512 MB de almacenamiento gratuito. En caso de que se requiera más capacidad, MongoDB Atlas puede escalar perfectamente, tanto en capacidad de almacenamiento, como en capacidad de procesamiento, en RAM y en nodos del *cluster* donde se encuentra replicada la información. De esta forma, posiblemente no es necesario cambiar esta opción para su uso en producción.

La elección para utilizar MongoDB Atlas en Azure no se debe a alguna razón en particular. En realidad, como se mencionó, se puede utilizar en GCP y AWS también.

En consecuencia, se creó una BBDD y una colección en ella; en la figura 3.4 se observa una captura de pantalla de la interfaz gráfica para gestionar la base de datos Mongo en el portal correspondiente.

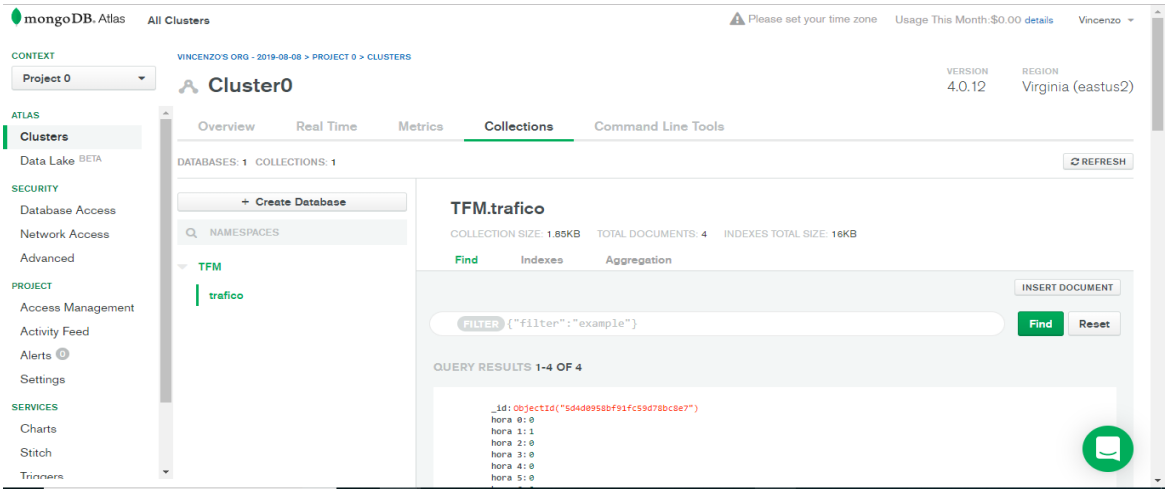

Figura 3.4 Captura de pantalla para gestión de BBDD MongoDB.

3.3 Selección de datos para la predicción de los niveles de tráfico

Decidido el almacenamiento, el siguiente paso consiste en decidir qué datos almacenar y de qué forma. Como se decidió utilizar una base de datos MongoDB, la cual es NoSQL sin estructura estática, se hace más fácil la planificación debido a que se puede cambiar el esquema sobre la marcha y almacenar nuevos campos que antes no se hayan estado almacenando.

Es necesario almacenar datos que sean pertinentes para poder predecir el flujo de tráfico de vehículos, dado que lo primordial es almacenar la fecha, hora y cantidad de vehículos contados. Como el almacenamiento es muy económico, incluso más económico que el procesamiento, se decidió almacenar los datos de manera que se hiciera sencillo el entrenamiento del modelo de R-CNN durante la fase correspondiente. Como consecuencia de lo cual, se decidió pre-procesar los datos antes de almacenarlos y almacenarlos ya categorizados en distintos campos.

El problema de almacenar el número de vehículos contados es que no sólo basta saber la cantidad de los mismos sino los niveles de tráfico, que se pueden medir en base a la densidad vehicular. Por esta razón, es necesario conocer la cantidad de vehículos máxima soportada por el espacio de vía monitorizada por la cámara. Una observación presencial podría servir para evaluar las cantidades máximas aceptables para considerar

que el tráfico se encuentra en un nivel bajo, medio o alto.

Después de todos estos análisis se decidió almacenar la información con el siguiente esquema:


```
{     "_id": "5d4d095dbf91fc59d78bc8e9",
      "hora 0": "0",
      "hora 1": "1",
      "hora 2": "0",
      "hora 3": "0",
      "hora 4": "0",
      "hora 5": "0",
      "hora 6": "0",
      "hora 7": "0",
      "hora 8": "0",
      "hora 9": "0",
      "hora 10": "0",
      "hora 11": "0",
      "hora 12": "0",
      "hora 13": "0",
      "hora 14": "0",
      "hora 15": "0",
      "hora 16": "0",
      "hora 17": "0",
      "hora 18": "0",
      "hora 19": "0",
      "hora 20": "0",
      "hora 21": "0",
      "hora 22": "0",
      "hora 23": "0",
      "L": "0",
      "M": "0",
      "X": "0",
      "J": "0",
      "V": "1",
      "S": "0",
      "D": "0",
      "traficoactual": "3",
      "trafico": ["3", "10", "20"],
      "calle": "Paseo La Castellana",
      "fecha": "1565336957755”}
```


Desde el punto de vista del análisis predictivo que se propone en el presente trabajo, no son necesarios todos los datos mostrados. No obstante, se ha optado por mantener y almacenar todos los mostrados en la estructura anterior con el fin de poder utilizarlos en modelos más complejos cuya dependencia de otras variables tales como la fecha o la hora exacta sea crucial o bien por si se desea graficar la evolución del tráfico a lo largo del año. Es decir, se mantiene la estructura de datos en base a al diseño modular de la aplicación, para poder incorporar en el futuro nuevas funcionalidades.

La elección de MongoDB Atlas como motor de base de datos parece una decisión acertada debido a su bajo costo, a su garantía de seguridad de los datos, tanto en acceso a ellos como en que no se van a borrar accidentalmente y a que se mantiene la información replicada en al menos 3 sitios, garantizando así que no se perderá información. Por otra parte, las lecturas y escrituras se realizan en cuestión de milisegundos cuando son documentos como el mencionado en esta sección, en caso de consultar muchos documentos puede demorar, sin embargo, las consultas realizadas con datos de prueba para el presente trabajo con 120 documentos han producido demoras del orden de milisegundos, que resulta ser un resultado aceptable.

3.4 API Rest

Para poder hacer que los datos sean accesibles de manera libre y desde distintas aplicaciones de diferentes plataformas en múltiples lenguajes, se decidió la accesibilidad de los datos a través de una API Rest, la cual permitiría:

- Registrar los datos en la base de datos MongoDB
- Brindar información en tiempo real (actual) de los niveles de tráfico en cierta vía.
- Brindar información del pronóstico del tráfico en cierta vía en cierto momento a futuro.

La primera de las tres acciones anteriores debe ser privada ya que solo los dispositivos permitidos deben poder registrar datos, en cambio, las otras dos serán públicas para poder ser accedidas por cualquiera.

Para realizar la API Rest primero se probó con Flask en Python, sin embargo, el problema de Flask es que, si se ejecuta en un servidor, este servidor podría colapsar si se realizan muchas peticiones simultáneamente. Sin embargo, fue muy útil como ejercicio para probar de manera local el desarrollo de las aplicaciones.

Para probar las API se utilizó Postman v7.8.0, el cual permitió de manera muy cómoda definir el método, la URL, los encabezados, parámetros y cuerpo de las peticiones. En la figura 3.5 se muestra una captura de pantalla relativa a una solicitud en Postman para insertar una medición; la respuesta de la solicitud devuelve el documento

insertado en la base de datos, tal como se indicó en el apartado anterior.

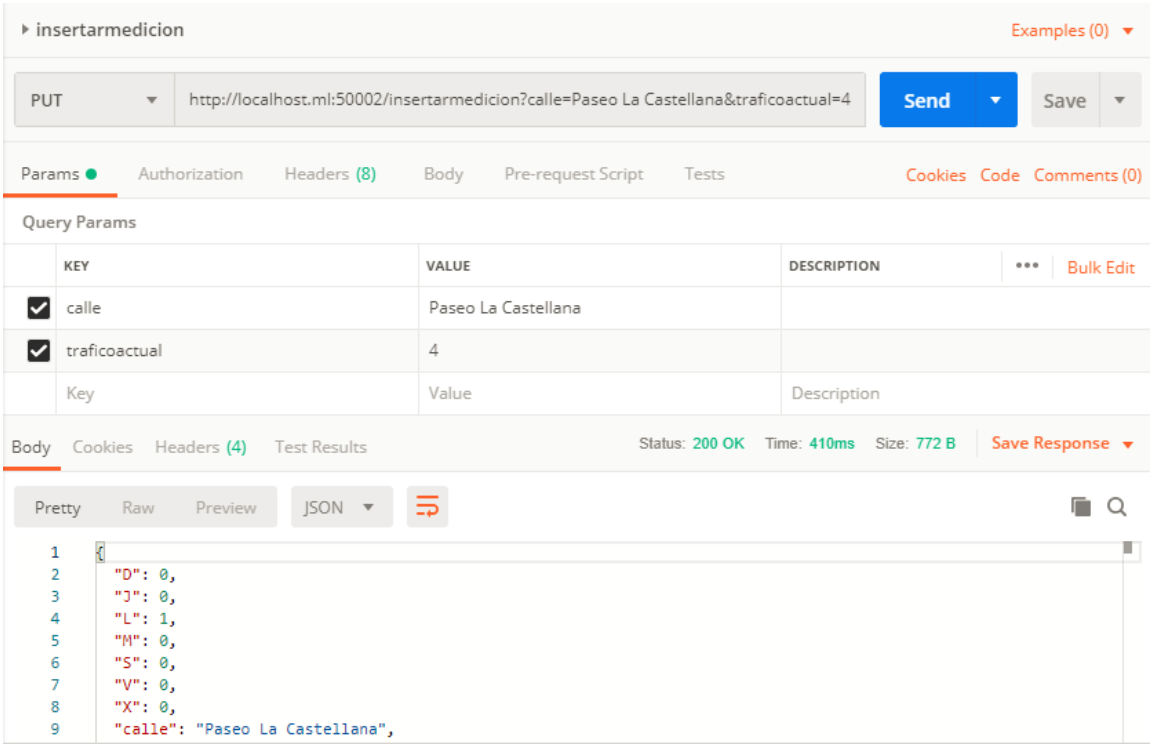

Figura 3.5 Captura de pantalla de una solicitud en Postman.

3.5 Máquina Virtual IaaS

A pesar de que en la mayoría de los procedimientos se utilizan servicios *serverless*, se decidió utilizar un servicio Iaas para entrenar los modelos de regresión logística, por dos razones:

1. Se quiere integrar la mayor cantidad de servicios diferentes, y ya se utilizó tanto SaaS como PaaS.
2. Se desea disminuir los costos. Lamentablemente los servicios de entrenamiento de Inteligencia Artificial en la nube son muy costosos, también es muy costosa la virtualización. Una ventaja de servicios de Inteligencia Artificial en la nube para entrenamiento como AWS SageMaker es que cobran por tiempo utilizado para entrenar, estos cobran por implementación de la máquina, carga de datos, entrenamiento y descarga de resultados, pero luego, cobran por seguir utilizando ese modelo para predecir, lo cual se vuelve muy costoso a la larga. Hay opciones

de virtualización no tan costosas como AWS SageMaker o Azure ML Studio que pueden servir para entrenar modelos y predecir con ellos.

3.5.1 Contratación y configuración

A pesar de que la virtualización en la nube es costosa, existen servicios de virtualización en la nube que no lo son tanto. Por ejemplo, AWS tiene una capa gratuita de máquinas virtuales muy limitadas, sin embargo, para ejecutar Apache Spark se quedaría muy pequeña.

Dado que se desea la disponibilidad de un servicio de predicción a bajo costo, se decidió, por ahora, implementar una máquina virtual en OVH; en este tipo de máquinas el precio es fijo, a pesar de lo cual, no es tan alto como un servicio de virtualización o de entrenamiento en las nubes de Azure o AWS.

Implementar un IaaS implica muchísimo más trabajo que un PaaS o un SaaS. Para poder poner IaaS en funcionamiento fue necesario realizar los siguientes pasos

1. Contratar un servicio de VPS, que crea una instancia de máquina virtual y asigna una IP pública estática, figura 3.6.

Figura 3.6 Servicio de VPS.

2. Registrar/comprar un nombre de dominio en un DNS, esto se hizo con Freenom, el

cual asigna nombres de dominio gratuitos, figura 3.7

Figura 3.7 Registro de dominio DNS.

3. Asociar en Freenom de la IP asignada por OVH al nombre de dominio comprado, figura 3.8.

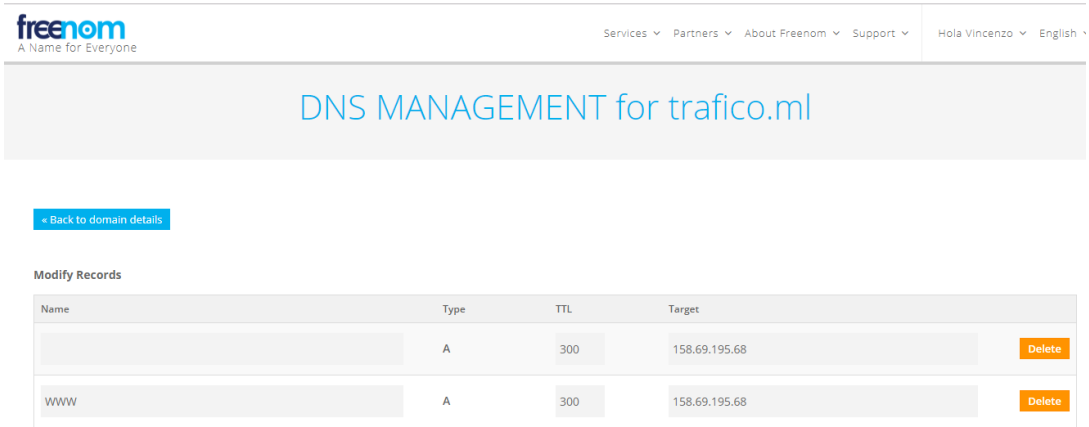

Figura 3.8 Asociación en Freenom de la IP asignada por OVH.

4. Crear un certificado SSL con Let's Encrypt y Certbot (Rahul, 2018).
5. Instalar Jupyter Notebooks con entornos virtuales para evitar problemas de

dependencias a futuro. Esto conlleva también: a) configurar Jupyter Notebooks para que se inicie como servicio cada vez que se reinicie la máquina virtual. Permitir que el certificado SSL otorgado por Let's Encrypt sea accesible por Jupyter Notebooks y otros programas; b) configurar Jupyter Notebooks para que use el certificado SSL otorgado por Let's Encrypt y c) asignar una contraseña para que se pueda acceder a través de internet de manera segura (Andrade, 2016).

6. Instalar el resto de las dependencias: JDK, Apache Spark, Scala, algunas dependencias instalables con pip install como requests, ufw, etc. (Roseindia, 2019).
7. Permitir los accesos a través del firewall con ufw al puerto 8888 (De Jupyter Notebook) y 50002 (puerto que se configuró en Flask, se pudo configurar cualquier otro no utilizado). Para esto se tienen que ejecutar los comandos “ufw enable; ufw allow 8888; ufw allow 50002”

Una vez instalado y configurado el entorno, se pudo empezar a ejecutar el código tal como se usó en local. Fue necesario hacer una pequeña modificación al código, que consistió en indicar a Flask que iba a utilizar *https* con el mismo certificado otorgado por LetsEncrypt. Cabe destacar que a pesar de que esto se puede implementar en un ordenador de sobremesa, el hecho de tener una IP pública y que un servicio activo permanentemente (24/7) permite que el sistema sea mucho más estable y accesible.

En la figura 3.9 se muestra una captura de pantalla con el código siendo editado directamente en Jupyter Notebook en el servidor virtual privado.

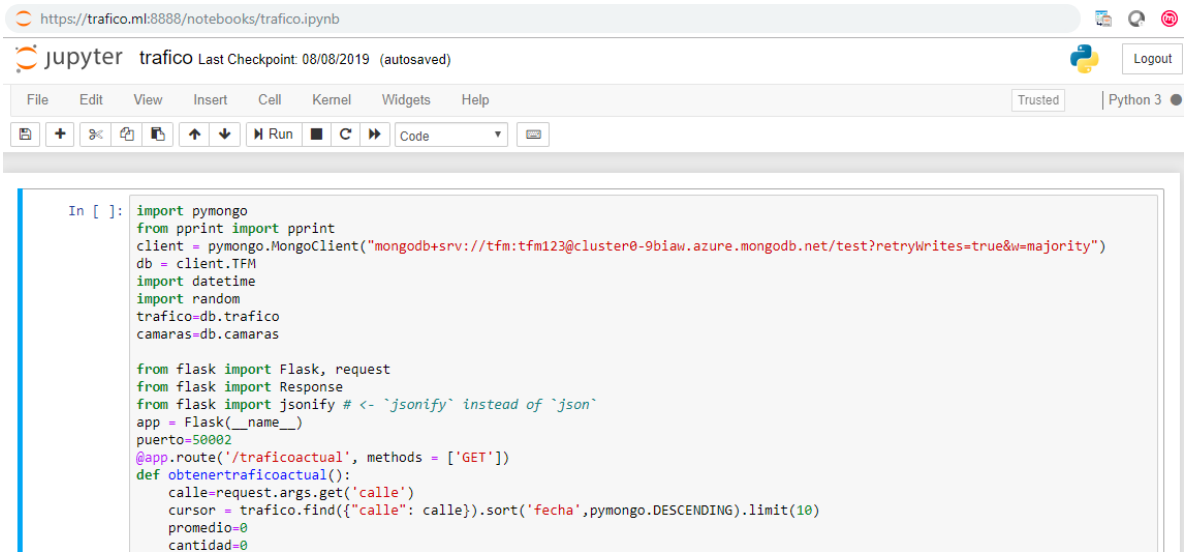

Figura 3.9 Captura de pantalla con edición de código en Jupyter Notebook.

Una vez realizadas las pruebas pertinentes se obtuvo el resultado en Postman, mostrado en la figura 3.10.

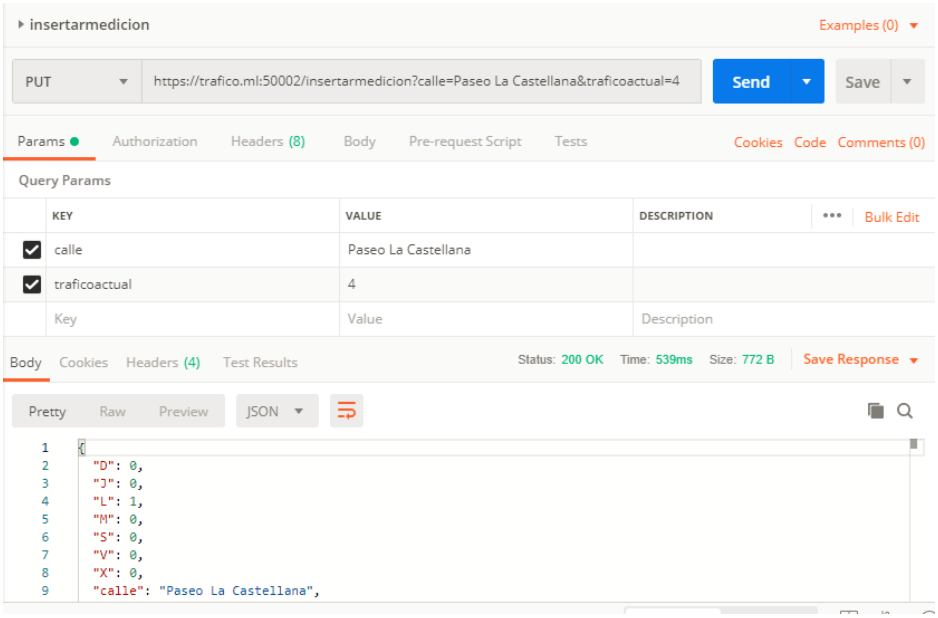

Figura 3.10 Resultados de las pruebas en Postman.

Para conectar la máquina virtual de OVH se realizaron las conexiones por SSH a

través de Putty. Además, con el fin de aumentar la seguridad se realizaron las siguientes acciones principales, entre otras: actualizar todos los paquetes de Ubuntu con apt-get upgrade, crear un usuario con permisos de super usuario diferente al que se crea por defecto, prohibir el acceso por SSH al usuario root, instalar fail2ban para evitar los ataques de denegación de servicio.

La idea de este servidor sería usarlo también en el futuro para proporcionar información del pronóstico del tráfico en cierta vía y en cierto momento, de esta forma, se utilizará tanto para entrenar el modelo como para predecir con el modelo pre-entrenado.

3.6 Modelo predictivo de los niveles de tráfico basado en regresión

Una vez se dispone de los datos convenientemente almacenados y estructurados, se decidió utilizar un modelo de regresión para predecir los niveles de tráfico. Cabe destacar que esto es posible ya que se consideran como variables independientes las horas de cada día y los días de la semana, es decir, se almacenan y evalúan como variables categóricas.

Cabe destacar en este sentido que el flujo vehicular es muy complejo pudiendo haber optado por diferentes modelos de regresión para el análisis, si bien el objetivo del presente trabajo se sitúa fuera de esta consideración ya que pretenden realidad se trata de demostrar que cualquier modelo de predicción resulta perfectamente integrable bajo el paradigma IoT con todas las tecnologías disponibles y que se pueden interconectar para lograr abarcar desde la captura de los datos hasta la predicción. El esquema categórico de los datos que se utilizó es el siguiente:

```
root
|-- D: long (nullable = true)
|-- J: long (nullable = true)
|-- L: long (nullable = true)
|-- M: long (nullable = true)
|-- S: long (nullable = true)
|-- V: long (nullable = true)
|-- X: long (nullable = true)
|-- calle: string (nullable = true)
|-- hora_0: long (nullable = true)
|-- hora_1: long (nullable = true)
|-- hora_10: long (nullable = true)
|-- hora_11: long (nullable = true)
|-- hora_12: long (nullable = true)
|-- hora_13: long (nullable = true)
|-- hora_14: long (nullable = true)
|-- hora_15: long (nullable = true)
|-- hora_16: long (nullable = true)
```

```
|-- hora_17: long (nullable = true)
|-- hora_18: long (nullable = true)
|-- hora_19: long (nullable = true)
|-- hora_2: long (nullable = true)
|-- hora_20: long (nullable = true)
|-- hora_21: long (nullable = true)
|-- hora_22: long (nullable = true)
|-- hora_23: long (nullable = true)
|-- hora_3: long (nullable = true)
|-- hora_4: long (nullable = true)
|-- hora_5: long (nullable = true)
|-- hora_6: long (nullable = true)
|-- hora_7: long (nullable = true)
|-- hora_8: long (nullable = true)
|-- hora_9: long (nullable = true)
|-- traficoactual: long (nullable = true)
```

Se hicieron varias pruebas diferentes utilizando datos simulados y distintos tipos de regresores, de entre todos los probados, el que mejores resultados obtuvo en términos de MAE, MSE y RMSE fue el regresor Lineal. Este modelo se implementó con Apache Spark y Python en el mismo VPS en OVH de modo que está siempre disponible con fines de predicción. Cabe destacar que, dado que actualmente el predictor consiste en un modelo de regresión lineal, no es necesario utilizar Apache Spark para predecir. Es decir, se podría usar Apache Spark para entrenar el modelo, obtener sus coeficientes y utilizar un código que tome los datos de entrada, le aplique el preprocesamiento a los datos y simplemente utilice los coeficientes obtenidos del modelo para multiplicar los vectores y, en base a esa información, obtener la predicción; es decir, no sería necesario utilizar una VPS, siendo posible utilizar una función serverless sencilla como una AWS Lambda o una Microsoft Azure Function, sin embargo, dado que el modelo podría no ser una simple regresión lineal, se deja la infraestructura preparada para escenarios con un modelo predictivo más complejo.

3.7 Mejoras en la API Rest con tecnología serverless

Debido a que la tarea más común realizada corresponde al registro de las mediciones a partir de las imágenes captadas por las cámaras y la solicitud de los niveles de tráfico, se decidió utilizar un servicio *serverless* para gestionar estas peticiones, para esto se empezó probando con Azure Functions en Python y luego con AWS Lambda Functions en Python.

3.7.1 Azure Functions en Python

Implementar una Azure Function en Python desde Visual Studio Code es relativamente sencillo, y permite gestionar las dependencias a instalar en un archivo llamado *requirements.txt* donde se establecen dichas dependencias que finalmente se instalarán con "pip install", e incluso se puede indicar la versión que se desea instalar. Las Azure Functions se pueden probar en local y no hacen uso de Flask, sino que utilizan un esquema ligeramente diferente, pero partiendo de la plantilla "Hello World" se puede entender fácilmente, de modo que prácticamente el mismo código utilizado con Flask en Python se puede utilizar en una Azure Function en Python.

El problema de estas funciones es su inestabilidad, encontrándose actualmente en versión preliminar (beta). Por lo que se pudo verificar, durante la realización de las pruebas, existen problemas con la estabilidad de estas funciones de forma que a veces pueden no ejecutarse o ejecutarse incorrectamente.

3.7.2 Lambda Functions en Python

Desarrollar una función Lambda en Python con AWS resulta mucho más sencillo que una Azure Function en caso de que no haya que instalar ninguna dependencia, es decir, un ejemplo sencillo como el de un código "Hello World" es más fácil de probar en AWS. El portal de la consola de AWS permite probar la ejecución de las Lambda, además, permite configurar eventos de prueba con el fin de dejar guardado lo que se va a enviar en la solicitud HTTP para poder probar cada vez. Permite guardar varios eventos de prueba diferente para probar la misma Lambda.

La complicación principal ocurre cuando es necesario utilizar dependencias instaladas con "pip install", estas dependencias hay que instalarlas en el equipo en el que se está desarrollando, guardarlas en una carpeta y en la misma carpeta donde se encuentran todos los módulos se debe colocar el código que se va a ejecutar. Posteriormente, todo esto hay que comprimirlo y en caso de que el archivo .zip supere los 50 MB, será necesario subirlo a un *bucket* de AWS S3, lo que fue necesario hacer en este trabajo. Desarrollar la primera Lambda con dependencias de "pip install" trajo grandes problemas para la publicación correcta de Lambda con la documentación en español por

la imprecisión en la traducción con respecto a la versión inglesa, con la que finalmente se logró correctamente.

Otra ventaja que presentan las funciones Lambda con Python es que se puede ver y editar el código directamente desde la consola de AWS, algo que no ocurre en formato *.zip*. En cualquier caso, durante el desarrollo resulta muy conveniente controlar la evolución de las versiones para evitar sobreescrituras.

Las funciones Lambda en Python permiten ser probadas desde la consola de AWS, tal como se acaba de mencionar, lamentablemente, la depuración no puede realizarse de manera interactiva, es decir, no puede probarse línea a línea el código, sino que se pueden ver los *logs* de todos los *print* disponibles, junto con las trazas de error impresas en su caso. Sin embargo, la programación de las Lambda es mucho más fácil de probar e integrar con otros editores y depuradores de código ya que en las Lambda de Python se programan dentro de una función de Python definida con "def", esto permite que el código completo se pueda probar por ejemplo en Jupyter Notebook y sin tener que realizar modificaciones, se pueda publicar en AWS.

En la figura 3.11 se puede observar una captura de pantalla de los *logs* que se pueden ver en la consola de AWS al ejecutar una Lambda.

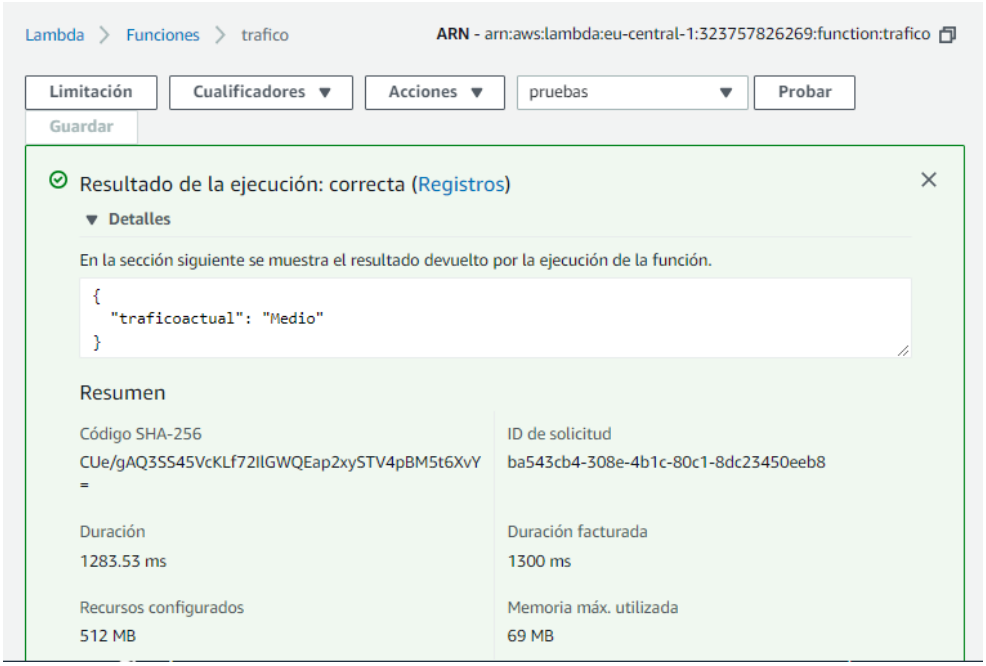

Figura 3.11 Visualización de los logs en la consola AWS.

3.8 Servicio para ofrecer información a nivel de usuario

A pesar de que la información es accesible a través de las correspondientes aplicaciones y plataformas, es necesario conseguir que sea lo más amigable posible "*human friendly*", para cualquier persona en general. Para conseguir este objetivo se analizaron las opciones que se describen a continuación.

3.8.1 Uso de Atajos de Siri (Siri Shortcuts)

Los atajos de Siri permiten programar una serie de acciones, en particular, incluso consultas HTTP y luego, mostrar la información tanto en modo escrito como hablado al usuario.

Algo interesante es que se puede compartir muy fácilmente entre dispositivos Apple con iOS. Se realizaron las pruebas pertinentes con resultados satisfactorios. En la captura de pantalla mostrada en la figura 3.12 se observa la facilidad para programar estos atajos.

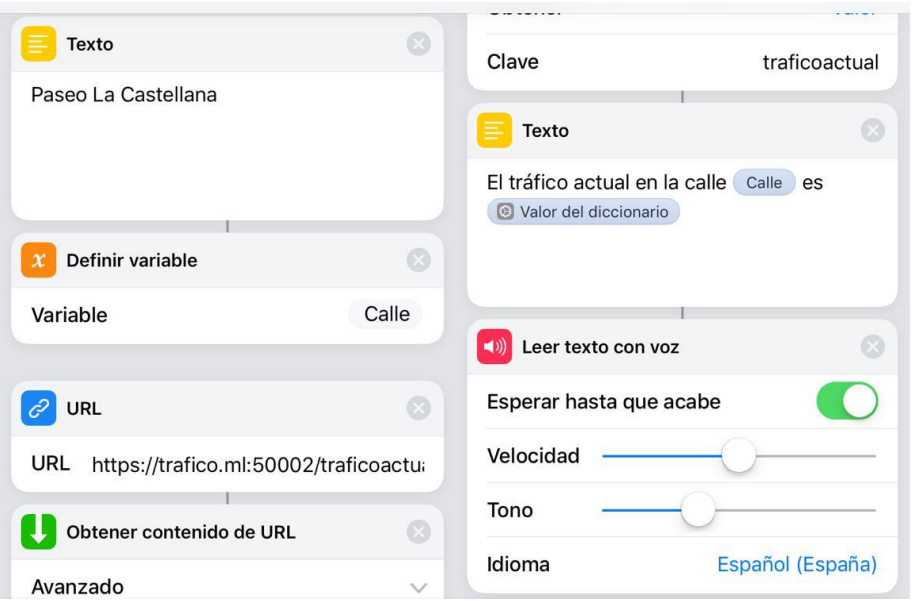

Figura 3.12 Visualización de los atajos en Siri.

3.8.2 Uso de Telegram a través de un Bot y Node-red

Otra opción factible consiste en utilizar un Bot a través de Telegram para hacer las pertinentes solicitudes. Como ya se disponía de una instancia de VPS en OVH, se podría fácilmente ejecutar un código que estuviese disponible para responder mensajes de

Telegram, a pesar de que también se pudo utilizar Python, se decidió utilizar Node-Red, dado que se puede contar con los conectores preprogramados que sólo necesitan ser configurados. Con tal propósito, fue necesario realizar los siguientes pasos para lograr esto:

- Instalar Node-Red.
- Permitir el acceso a los puertos a través del firewall de Ubuntu con *ufw*.
- Configurar Node-Red para que utilice los mismos certificados SSL otorgados por LetsEncrypt.
- Instalar las dependencias: Red-Bot
- Crear un Bot en Telegram con @BotFather
- Programar el Flujo para que recibiera mensajes, evaluara la intención y respondiera

En la imagen de la figura 3.13 se observa una captura de pantalla relativa a una de las pruebas 3.8.3 Uso de WhatsApp para empresas a través de Twilio y Azure Logic Apps

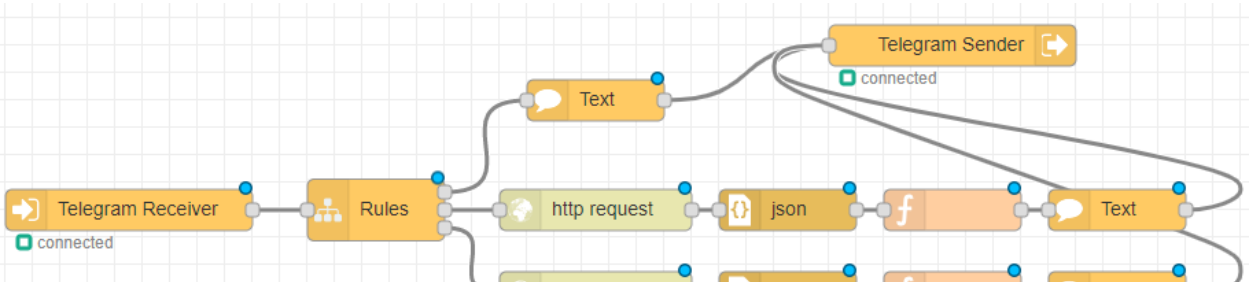

Figura 3.13 Configuración de WhatsApp con Twilio y Azure Logic Apps.

3.8.3 Uso de Whatsapp a través de Twilio y Azure Logic Apps

Utilizar Telegram es muy útil y fácil, sin embargo, esta aplicación es menos utilizada en comparación con otras por ejemplo WhatsApp a pesar de ser gratuito para su integración con una aplicación.

Dado que Twilio permite utilizar Whatsapp para Empresas, pudiéndose integrar con una aplicación con su correspondiente API, se decidió probar con ella. Esta vez, para programar la API, se decidió utilizar Azure Logic Apps.

Utilizar Azure Logic Apps no fue muy complicado ya que se puede observar el registro de los *Logs*, lo que permite detectar las ejecuciones que fallaron y saber por qué. Además, la programación en bloques de acción de manera gráfica hace que sea muy intuitivo desde el punto de vista de la programación.

Para la programación de la Logic App fue necesario implementar un bucle que recorriera el cuerpo enviado por Twilio, en ese momento se observó que los bucles se realizan muy lentamente en las Logic Apps. Se prefirió utilizar una Azure Function App que tomara el cuerpo recibido de Twilio y lo serializara en un JSON para poder trabajarlo más cómodamente, bien sea en la Logic App o directamente en la Azure Function App.

La Azure Function App se programó en C# debido a que se quiso realmente utilizar Azure Function Apps en la solución completa. Por consiguiente, dado que Python no era una opción conveniente porque Azure Function Apps para Python se encuentra actualmente en versión preliminar, se utilizó C# que está completamente soportado.

Esta integración de Whatsapp para Empresas, Twilio, Azure Logic Apps y Azure Function Apps en C#, a pesar de que los códigos no sean muy extensos, no resulta trivial desde el punto de vista de la integración dada la falta de documentación y su dispersión, habiendo sido necesario la realización de diferentes y exhaustivas pruebas.

En la secuencia de figuras 3.14 a 3.16 se muestran capturas de pantallas de los portales de estos servicios.

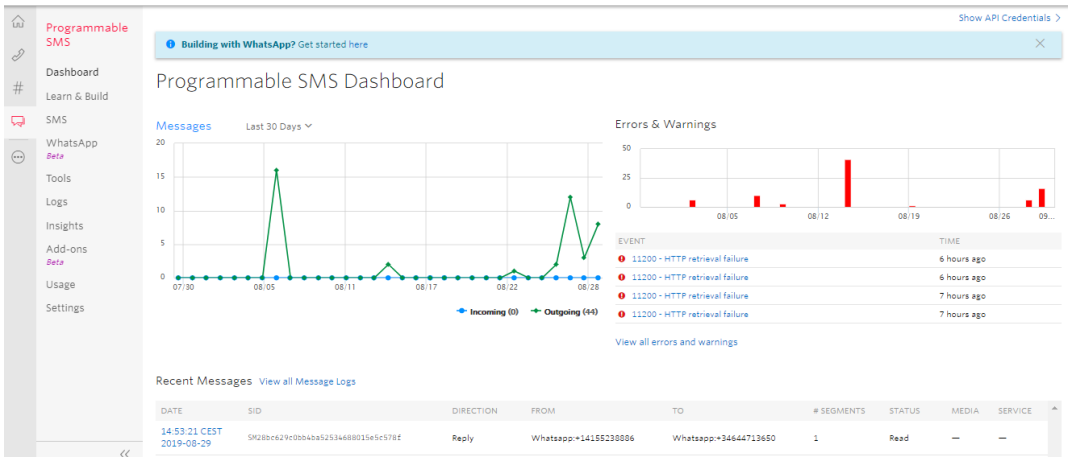

Figura 3. 14 Dashboard de Logs de Twilio.

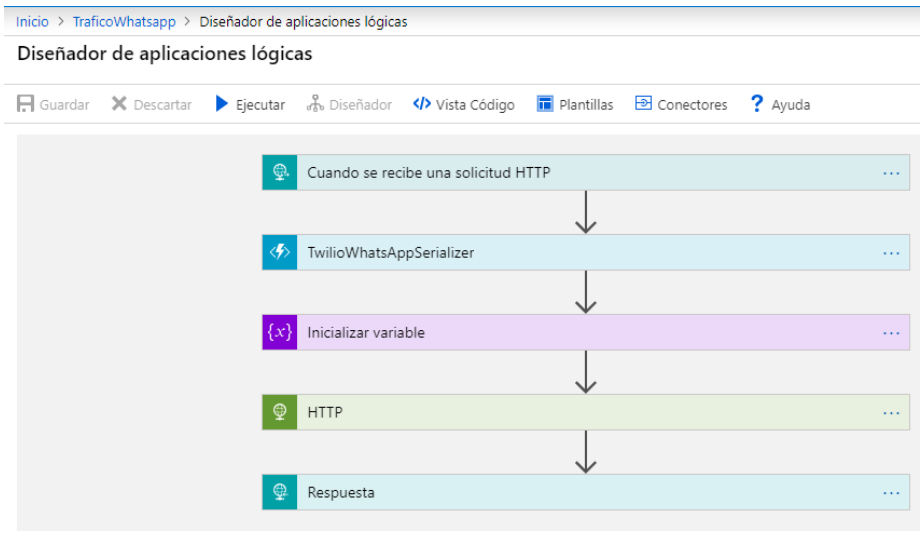

Figura 3. 15 Azure Logic App que gestiona los mensajes de WhatsApp.

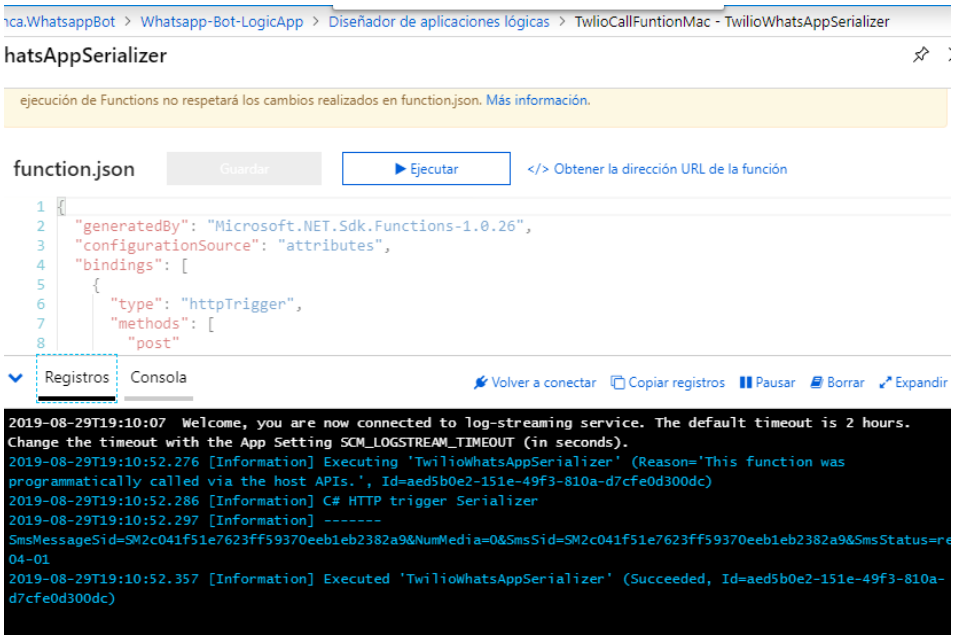

Figura 3. 16 Logs de ejecución de una Azure Function.

3.8.4 Uso de SMS a través de Twilio

Dado que ya se utilizó Twilio y se analizaron sus opciones para enviar SMS, se pensó que otra opción interesante podría ser que algunos usuarios quieran ser avisados acerca de los niveles de tráfico existentes en las vías que utilizan a diario, ésta puede ser

una manera de cobrar una suscripción a los usuarios que utilizan este servicio y así sufragar algunos gastos para mantener el sistema operativo.

Por tanto, se decidió crear una API con capacidad para enviar los mensajes a través de Twilio. En este caso se siguió utilizando Azure Logic Apps, dado que posee un conector de Twilio que permite de una manera muy sencilla enviar SMS. Esta Logic App no se integró con ningún programa, sólo se dejó en forma de API Rest para solicitar la información del tráfico a la API previamente creada, de forma que posteriormente se envía el SMS al número enviado y recibido en el cuerpo de la solicitud inicial.

En la imagen de la figura 3.17 se observa el registro de los *Logs* de solicitudes previas en Logic Apps.

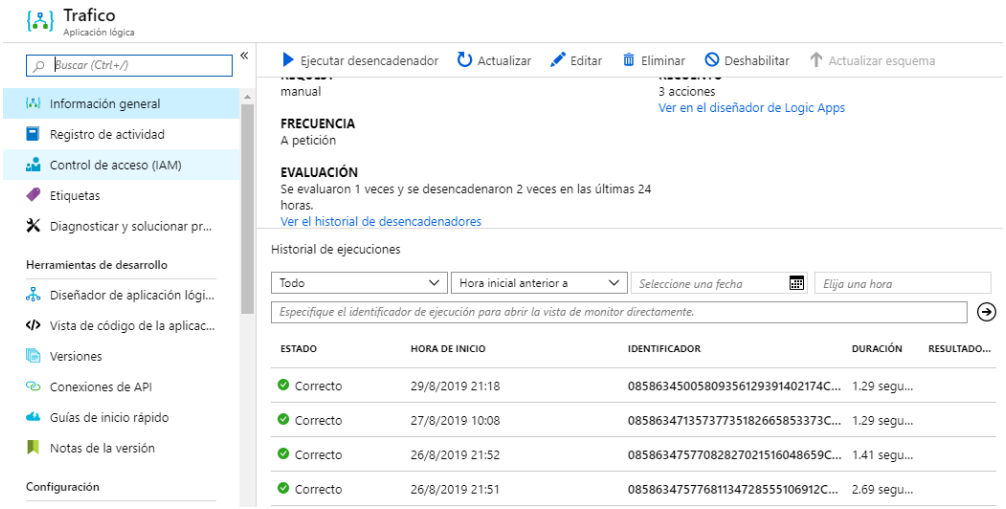

Figura 3.17 Visualización de Logs de solicitudes previas en Logic Apps.

Capítulo 4 Diseño de la solución

En el capítulo tres se ha descrito la evolución tecnológica, que ha permitido analizar distintas opciones técnicas al objeto de conseguir una aplicación lo más eficiente posible. En base a ello, en este capítulo se aborda y explica el diseño de la solución finalmente propuesta, donde se mencionan las tecnologías utilizadas, de qué modo se integran y cómo es el procedimiento para implementar la infraestructura y los códigos de la solución diseñada.

4.1 Arquitectura del sistema propuesto

El sistema propuesto se compone, a grandes rasgos, de los siguientes cinco componentes fundamentales:

1. **Captura de imágenes:** Para la captura de imágenes se utiliza un dispositivo modelo EIC MS Vision 500 de la marca Altek.
2. **Procesamiento de imágenes:** El procesamiento de las imágenes se realiza dentro del mismo dispositivo EIC MS Vision 500, dada su potencialidad de procesamiento de borde. Los códigos implementados son programados en Python, que ejecutan una red neuronal entrenada con Azure Custom Vision. Tanto el código en Python como el modelo de la red neuronal implementado en TensorFlow se ejecutan en un contenedor Docker.
3. **BBDD:** La base de datos donde se almacenan los datos provenientes de los dispositivos es MongoDB Atlas, hospedada en Azure.
4. **Gestor de dispositivos IoT:** Se utiliza Azure IoT Hub para gestionar los dispositivos y, con la implementación de Azure IoT Edge, se gestiona el aprovisionamiento, monitorización, parametrización, actualización y control de los dispositivos. Para la actualización e instalación del código de los dispositivos, se utiliza Azure Container Registry, donde se almacenan las imágenes que se envían a los dispositivos.
5. **APIs Rest:** Se utilizan varias API Rest, cada una realiza una tarea en específico. Son las siguientes:

a. **API para el registro de la medición:** Se cuenta con una función Lambda de AWS programada en Python que inserta las mediciones provenientes de los dispositivos en la base de datos MongoDB.
b. **API para la solicitud de los niveles de tráfico actual:** Se cuenta con una función Lambda de AWS programada en Python que consulta la base de datos MongoDB y responde con información del tráfico actual en la vía de circulación objeto de monitorización. Para poder hacer accesible esta API desde internet, fue necesario utilizar Amazon API Gateway, gestionada por Amazon API Gateway que a su vez redirige la función Lambda.
c. **API para la solicitud de envío de SMS con la información del tráfico:** Se utiliza una Azure Logic App que consulta la API mencionada en el punto (b) y envía un SMS al número que se haya solicitado a través de Twilio.
d. **Aplicación para la contestación de mensajes a través de Telegram:** Se emplea una aplicación en Node-Red que utiliza Red-Bot para gestión de Bots, la misma lógica sirve para implementar bots en Telegram, Facebook, Alexa o Slack. Según el caso sólo es necesario registrar los *bots* en cada una de las distintas plataformas y configurar los nodos de conexión respectivos. Esta aplicación también utiliza la API del punto (b) y un bot registrado con @BotFather en Telegram.
e. **Atajo de Siri para consultar el flujo de tráfico:** Con la API del punto (b), se programa una implementación de un atajo de Siri que realiza una llamada a dicha API y pregunta por el tráfico.
f. **Aplicación para la contestación de mensajes a través de Whatsapp:** Gracias al uso de Twilio, se establece un procedimiento de recibo y respuesta de mensajes de Whatsapp. Se dispone de una Logic App que gestiona las consultas HTTP provenientes de Twilio y,

dado que las Logic Apps no son para realizar cómputos complejos, se envía el mensaje recibido a una Azure Function programada en C# que serializa en un JSON el mensaje previamente recibido de Twilio (que viene con Content-Type application/x-www-form-urlencoded), posteriormente utiliza la API del punto (b) y responde con la información al usuario de Whatsapp que le haya escrito.

g. **Aplicación para entrenamiento y API para la predicción de los niveles de tráfico:** Se cuenta con una aplicación desarrollada en Python que se ejecuta en una instancia Iaas de un VPS de OVH. Esta aplicación utiliza Apache Spark para entrenar modelos y predecir valores de los niveles de tráfico con datos almacenados en la BBDD del punto (3).

En la figura 4.1 se representa el esquema de la solución diseñada, donde se observan los logos de las tecnologías, herramientas, servicios y entes involucrados.

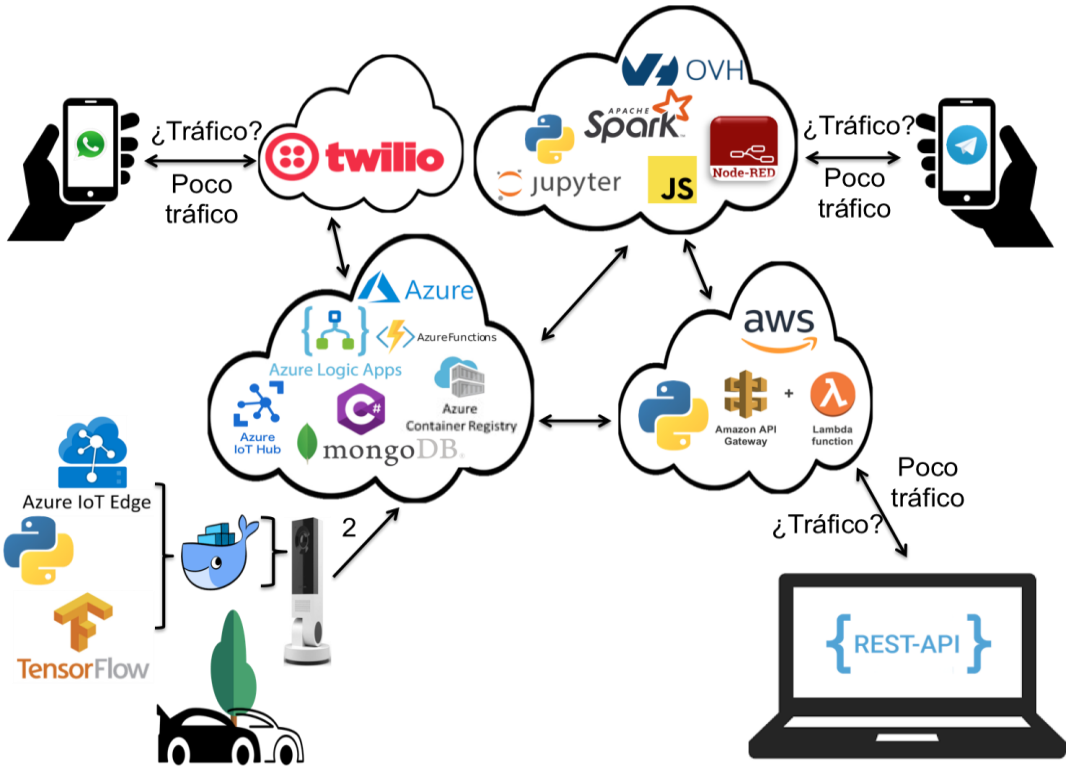

Figura 4.1 Esquema de la solución diseñada.

4.2 Configuración y puesta en funcionamiento de la solución

Los códigos se encuentran en https://github.com/vincenzocaschetto/trafico

A continuación, se indica cómo ejecutar y desplegar cada uno de los códigos:

4.2.1 Requisitos

Es necesario disponer de:

1. Cuenta de Azure.
2. Cuenta de AWS.
3. Cuenta de Twilio.
4. Ordenador o máquina virtual con acceso desde internet a todos los puertos con Node-Red, Jupyter Notebook, JDK y Apache Spark instalado y configurado.
5. Ordenador con Visual Studio Code, Azure IoT Edge Tools for Visual Studio Code, Docker Engine y Visual Studio instalados
6. Móvil con IOS, Whatsapp y Telegram instalados.

4.2.2 Configuración de entorno en Azure

- Crear un grupo de recursos.
- Crear recurso de IoT Hub.
- Crear recurso de Azure Container Registry.

4.2.3 Configuración inicial de dispositivos EIC MS Vision 500

1. Haber realizado los pasos definidos en el punto 4.2.2.
2. Registrar un nuevo dispositivo en Azure IoT Edge desde el portal de Azure en la sección del recurso de Azure IoT Edge creado previamente, siendo necesario tomar nota de la cadena de conexión principal.
3. Encender el dispositivo y conectar un ordenador a la red Wi-Fi generado por el dispositivo con el nombre y contraseña indicados en la caja del dispositivo

4. Se abrirá un navegador web, es necesario iniciar el setup, conectar a una red Wi-Fi con acceso a internet e indicar la cadena de conexión principal obtenida en el punto (2) en el campo “ConnectionString”

4.2.4 Registro de contenedores Docker Azure Container Registry

1. Haber realizado los pasos del punto (4.2.2).
2. Iniciar Docker Engine en dicho ordenador.
3. Abrir el código vision-ai-developer-kit con Visual Studio Code en un ordenador (código abierto descargado de https://github.com/microsoft/vision-ai-developer-kit, con algunas modificaciones)
4. Abrir el archivo samples\official\ai-vision-devkit-get-started\.env y agregar los datos obtenidos durante la creación del Azure Container Registry. Se pueden obtener en el Portal de Azure.
5. Hacer clic derecho en el archivo samples\official\ai-vision-devkit-get-started\deployment.template.json y hacer clic en “Build and Push IoT Edge Solution”.

4.2.5 Despliegue de contenedores en dispositivos con Azure IoT Edge

1. Iniciar Docker Engine en dicho ordenador
2. Abrir el código vision-ai-developer-kit con Visual Studio Code en un ordenador.
3. Haber realizado los pasos de los puntos 4.2.3 y 4.2.4.
4. Hacer clic con el botón derecho en el archivo samples\official\ai-vision-devkit-get-started\config\deployment.amd64.json y hacer clic en “Create Deployment for Single Device”.
5. Seleccionar el dispositivo en el que se desea desplegar el contenedor.
6. Esperar unos minutos con el dispositivo conectado a internet, la primera vez en descargar el contenedor tiene que descargar unos 900 MB, para futuras actualizaciones del contenedor por modificaciones en el código,

descarga sólo unos 25 MB, todas ellas correspondientes al código en Python y sus dependencias.

7. Verificar en el portal de Azure, en la sección de Azure IoT Edge, que el dispositivo donde se hizo el despliegue se obtienen resultados como los mostrados en la figura 4.2.

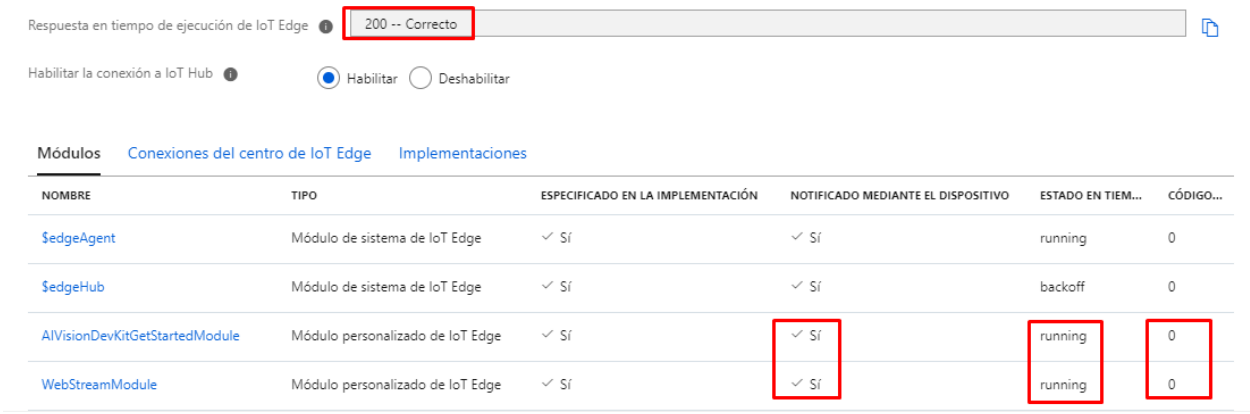

Figura 4.2 Dispositivo en IoT Edge con contenedores Docker en ejecución.

4.2.6 Indicación de valores personalizados para cada dispositivo

Para la configuración de parámetros diferentes de cada dispositivo, se deben realizar los siguientes pasos

1. Seleccionar el módulo (Contenedor) que se desea personalizar.
2. Hacer clic en Identidad de módulo gemela.
3. Modificar el valor que se desee cambiar.
4. Hacer clic en guardar.
5. Esperar unos segundos a que el cambio se implemente en el dispositivo.

4.2.7 Acceso SSH al dispositivo

El acceso por SSH al dispositivo puede resultar de gran utilidad para depurar errores o visualizar los *logs*, también puede ser útil para ver si los contenedores se están ejecutando o no, o si las imágenes Docker ya se descargaron en el dispositivo.

Para acceder por SSH es necesario tener un ordenador conectado a la misma red Wi-Fi y conocer la IP del dispositivo. Si no se conoce la IP, se puede utilizar algún programa

como Advanced IP Scanner para identificarla. Una vez conocida la IP se puede acceder a través de un terminal con SSH instalado o con Putty. Es necesario utilizar el usuario y contraseña establecidos en el *setup* de configuración indicado en el punto 4.2.3.

A continuación, se enumeran las acciones que se pueden opcionalmente realizar y los comandos requeridos para su ejecución:

- **Ver imágenes Docker descargadas:** docker images
- **Ver contenedores en ejecución:** docker ps
- **Ver logs de algún contenedor:** docker logs <nombre del contenedor>
- **Eliminar algún contenedor:** docker rm <identificador del contenedor>
- **Eliminar alguna imagen:** docker rmi <identificador de la imagen>

4.2.8 Errores conocidos:

1. Es posible que en algún momento el dispositivo EIC MS Vision 500 presente los 3 leds encendidos en color naranja. Para este caso, no se ha encontrado información adecuada en la documentación, sin embargo, si esto ocurre, es necesario conectar por cable USB-C y utilizar ADB ejecutando "adb shell" (desde un terminal abierto en la ruta donde se encuentra descargado ADB), esto se hace para acceder al *shell* y desde ahí escribir "reboot". Se observó que, si los 3 leds están encendidos en color naranja, el botón de encendido o el de Reset no realiza ninguna actividad, el equipo no reacciona, pero sí conecta a través de ADB.
2. Es posible que algún contenedor esté teniendo muchos *logs*, si un código realiza muchos *prints*, es posible que se sature la memoria de los *logs* de Docker. Para este caso, los contenedores dejarán de ejecutarse sin informar cuál es el problema. Si por alguna razón no se puede iniciar el *docker engine* en el dispositivo, es posible que sea por falta de espacio en el propio dispositivo, en este caso es necesario ubicar el archivo con el *log* grande, borrarlo y reiniciar el dispositivo.
3. En caso de que se desee partir de una plantilla original del código que se encuentra en el repositorio microsoft/vision-ai-developer-kit de Github, si hay errores de despliegue, se recomienda que se determinen las diferencias entre las carpetas,

dado que se hicieron unas pequeñas modificaciones porque la versión original no está configurada para utilizar el procesador con la arquitectura de este dispositivo.

4.2.9 Despliegue de Azure Function App:

En la aplicación se utiliza una Azure Function App que sirve para serializar en un JSON el cuerpo recibido de Twilio. Para publicar esta función en Azure es necesario realizar los siguientes pasos:

1. Haber realizado los pasos descritos en el punto 4.2.2.
2. Abrir el archivo Trafico/Trafico.sln en Visual Studio.
3. Hacer clic derecho sobre el archivo TwilioSerializerFunctionApp en el explorador de soluciones.
4. Hacer clic en *Publicar*.
5. Seleccionar el grupo de recursos en el que se desea publicar.
6. Hacer clic en *Publicar*.
7. Esperar a que se publique, puede demorar unos minutos.

4.2.10 Despliegue de Azure Logic App

En este trabajo se utilizan tres Logic Apps. A continuación, se explica cómo desplegar la que envía los SMS, siendo necesario realizar los siguientes pasos:

1. Haber aplicado los pasos contemplados en el punto 4.2.2.
2. Abrir el archivo Trafico/Trafico.sln en Visual Studio
3. Hacer clic derecho sobre el archivo TraficoSMS en el explorador de soluciones.
4. Hacer clic en *Validar*.
5. Hacer clic en *Parámetros*.
6. Agregar las cadenas de conexión y *secret* de Twilio en los campos respectivos.
7. Hacer clic en *Aceptar*.
8. Seleccionar el grupo de recursos en el que se desea desplegar la Logic App.
9. Hacer clic en *Aceptar*.
10. Hacer clic derecho sobre el archivo TraficoSMS en el explorador de soluciones.

11. Colocar el cursor sobre *Implementar* y esperar a que se abra un submenú a la derecha.
12. Hacer clic en el nombre del grupo de recursos donde se va a desplegar.
13. Esperar que se despliegue, puede demorar unos minutos.

En el caso de las otras Logic Apps, no es necesario enviar SMS a través de Twilio, por lo que el punto 6 no es necesario aplicarlo y, las acciones sobre los puntos 3 y 10, se realizarán sobre el proyecto correspondiente.

Cabe destacar que los códigos, tal como están programados actualmente, realizan consultas a una API que será la que indique los valores del flujo de tráfico. No obstante, como se está implementado todo desde 0, será necesario haber implementado el código de dicha API para tener una URL que se utilice en las consultas HTTP de cada Logic App, según se describe en el punto 4.2.11.

4.2.11 Registro de Bot en Telegram

Para registrar un *bot* en Telegram es necesario realizar los siguientes pasos:

1. Abrir Telegram en el dispositivo móvil.
2. Abrir una conversación con @BotFather.
3. Escribir el comando "/newbot".
4. Indicar el nombre que se le desee dar al *bot*, que debe terminar en *bot*.
5. Tomar nota del *token* generado para poder hacer uso de la API de Telegram.

4.2.12 Implementación del código en Node Red

Para implementar el código en Node-Red es necesario seguir los siguientes pasos:

1. Ejecutar Node-Red y abrir el portal desde un navegador web.
2. Instalar Red-Bot:
    a. Hacer clic en el menú (3 barras horizontales en la esquina superior derecha).
    b. Hacer clic en "Manage Pallete".
    c. Seleccionar la pestaña "Install".

d. Buscar “node-red-contrib-chatbot” y presionar instalar
e. Hacer clic en “Close”

3. Abrir el archivo flows.json y copiar el contenido de todo el archivo.
4. En la web de Node-Red abrir el Menú.
5. En “Import” hacer clic en “Clipboard”.
6. Pegar los datos copiados.
7. Hacer clic en “Import”.
8. Hacer doble clic en el nodo “Telegram Receiver”.
9. En “Bot configuration (development)”, hacer clic en “Add new chatbot-telegram-node…”, luego hacer clic en el botón a la derecha que tiene dibujado un lápiz.
10. Asignar un nombre en “Bot Name”.
11. Colocar el token conseguido en el punto (4.2.11) en el campo “Token”.
12. Hacer clic en “Add”.
13. Seleccionar el *bot* configurado en “Bot configuration (production)”.
14. Hacer clic en “Done”.
15. Seleccionar el bot configurado tal como en el punto 13 en nodo “Telegram Sender” tanto en “Bot configuration (production)” como en “Bot configuration (development)”.
16. Hacer clic en “Deploy”.
17. Probar el *bot* escribiendo por un chat nuevo en una conversación con el *bot* creado desde Telegram, se debería recibir una respuesta.

Cabe destacar que este módulo también hace uso de la API que informa el tráfico, por lo que habría que cambiar la URL en los nodos “http request” por la *url* correcta, según se indica en el punto 4.2.11.

4.2.13 Configuración de entorno en AWS

Crear un Api Gateway que permita acceso desde internet.

4.2.14 Implementación del código en AWS Lambda

Existe un código incluido en la API, que consiste en insertar mediciones en la base de datos y otro que indica el tráfico de vehículos actual, ambas APIs son muy utilizadas, y los mencionados códigos están desarrollados para ser implementados en Lambdas de AWS. A continuación, se explican los pasos a seguir para lograr esta implementación:

1. Abrir la consola de administración de AWS desde el portal web.
2. Abrir el servicio *Lambda.*
3. Hacer clic en *Crear una función*.
4. Asignar un nombre de función.
5. Seleccionar *Python 3.7* en la lista desplegable de lenguajes.
6. Crear un nuevo rol de ejecución con permisos básicos de Lambda si no se dispone de un rol con estos permisos, en caso de que se disponga de un rol, se puede seleccionar dicho rol. Asignarle un nombre.
7. Hacer clic en *Crear función*.
8. Esperar a que se haya creado, puede demorarse uno o dos minutos.
9. Una vez creada la Lambda, acceder a ella.
10. En la sección *Código de la función*, seleccionar *Cargar un archivo .zip* en *Tipo de entrada de código*.
11. Hacer clic en *Cargar*.
12. Seleccionar el archivo lambdatraficoactual.zip.
13. Hacer clic en *Guardar* (esquina superior derecha)
14. Habilitar el acceso a través del API Gateway haciendo clic en *Habilitar* en la sección *API Gateway*.
15. Hacer clic en *Guardar*.
16. Tomar nota de la *url* generada, que se utiliza para llamar a la API desde Internet o desde las Logic Apps o Node-Red o Atajos de Siri.

4.2.15 Implementación de los atajos de Siri

Para utilizar el atajo de Siri en dispositivos con iOS y Siri, es necesario realizar los siguientes pasos:

1. Abrir el enlace de la url https://www.icloud.com/shortcuts/be68a420ae52412395f5d381a537c47d
2. importar el atajo en el móvil.
3. Abrir el atajo y entrar en el modo edición.
4. Escribir el nombre de la calle sobre la que se desee consultar el flujo de tráfico y siempre en el atajo.
5. Colocar la *url* de la API de la función *lambda* obtenida en el punto 4.2.11.

4.2.16 Implementación del predictor de tráfico

El código que implementa el modelo predictivo en Apache Spark está pensado para ser ejecutado en una VPS de OVH, sin embargo, no es necesario hacer esto, se puede ejecutar en un ordenador que cumpla los requisitos mencionados en el punto 4 de la sección 4.2.1. Para ejecutar este código es necesario realizar los siguientes pasos:

1. Ejecutar Jupyter-Notebook y abrir un navegador web con el portal de dicho programa.
2. Abrir el archivo API Predictor.ipynb
3. Ejecutar el código.
4. Se podría utilizar *ngrok* para hacer accesible desde internet el puerto 50002 con *https*, para esto sería necesario:
    a. Descargar *ngrok*
    b. Abrir un terminal en la ruta donde esté descargado *ngrok* y ejecutar el comando: *ngrok http 50002 -host-header="localhost:50002"*
    c. Tomar nota de la *url* generada por *ngrok*. Ésta es la *url* que se puede utilizar para hacer las consultas http a la API desde internet.

Capítulo 5 Resultados

En este capítulo se presentan los resultados obtenidos con el fin de evaluar el diseño final desde dos puntos de vista: uso y rendimiento. El objetivo desde el punto de vista del uso es verificar el nivel de integración obtenido, considerando que una de las aportaciones más importantes realizadas ha sido precisamente la conjunción de una serie de módulos que conforman la aplicación global. Por otra parte, desde el punto de vista del rendimiento la finalidad es determinar el desempeño de la aplicación, donde los tiempos de ejecución de cada uno de los módulos y de la aplicación en su conjunto, constituye un elemento esencial, teniendo en cuenta que el objetivo final es su implantación en contextos eficientes para tiempo real.

5.1 Funcionamiento y uso de la solución

En esta sección se explica cómo se utiliza cada una de las aplicaciones modulares o servicios que integran la solución global.

La verificación de las diferentes pruebas se realiza mediante la comprobación de la respuesta obtenida en cada aplicación modular al requerimiento formulado.

5.1.1 Uso de la API Rest para obtener la información del tráfico actual

Para utilizar la API Rest que se encuentra en AWS Lambda, es necesario llevar a cabo una consulta con los siguientes datos:

- Método: POST
- URL: **https://grnqm12udh.execute-api.eu-central-1.amazonaws.com/ApiGfi/trafico**
- Encabezado: Content-Type:application/json
- Cuerpo:

```
{
   "calle": "Paseo La Castellana"
}
```

La respuesta recibida tiene la siguiente forma:

- Encabezado: Content-Type:application/json
- Cuerpo:

```
{
   "traficoactual": "Bajo"
}
```


5.1.2 Uso de la API Rest para registrar datos en la base de datos

Para utilizar la API Rest que se encuentra en AWS Lambda, es necesario realizar una consulta con los siguientes datos:

- Método: POST
- URL:**https://grnqm12udh.execute-api.eu-central-1.amazonaws.com/ApiGfi/traficoinsertarmedicion**
- Encabezado: Content-Type:application/json
- Cuerpo:


```
{
   "calle": "Paseo La Castellana",
   "traficoactual": 4
}
```


El valor 4 se reemplaza dinámicamente por la cantidad de vehículos contados, sin embargo, desde el punto de vista de las pruebas de test, se puede utilizar cualquier valor numérico. En el caso que nos ocupa y para la consulta realizada, la respuesta recibida tiene la siguiente forma:

- Encabezado: Content-Type:application/json
- Cuerpo:


```
{
   "hora 0": 0,
   "hora 1": 0,
   "hora 2": 0,
   "hora 3": 0,
   "hora 4": 0,
   "hora 5": 0,
   "hora 6": 0,
   "hora 7": 0,
   "hora 8": 0,
   "hora 9": 0,
   "hora 10": 0,
   "hora 11": 0,
   "hora 12": 0,
   "hora 13": 0,
```

```
    "hora 14": 0,
    "hora 15": 1,
    "hora 16": 0,
    "hora 17": 0,
    "hora 18": 0,
    "hora 19": 0,
    "hora 20": 0,
    "hora 21": 0,
    "hora 22": 0,
    "hora 23": 0,
    "L": 0,
    "M": 0,
    "X": 0,
    "J": 0,
    "V": 0,
    "S": 1,
    "D": 0,
    "traficoactual": 4,
    "trafico": [
        3,
        10,
        20
    ],
    "calle": "Paseo La Castellana",
    "fecha": "2019-07-31 21:03:32.270665"
}
```


5.1.3 Uso de la API Rest para recibir por SMS los niveles de tráfico

Para utilizar la API Rest que se encuentra en Microsoft Azure Logic Apps, es necesario hacer una consulta con los siguientes datos:

- Método: POST
- URL:**https://prod-02.centralus.logic.azure.com:443/workflows/a7474db97eca4931b42b7873169d1fad/triggers/manual/paths/invoke?api-version=2016-10-01&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=smV4L1OCNW-vvXwcHReniahQwcUnGs3bMvJw43nFXk4**
- Encabezado: Content-Type:application/json
- Cuerpo:


```
{
    "calle": "Paseo La Castellana",
    "telefono": "644444444"
```

```
}
```

El valor del teléfono debe ser reemplazado por el número del móvil en el que se desea recibir el SMS

La respuesta recibida tiene la siguiente forma:

- Encabezado: Content-Type:application/json
- Cuerpo:

```
{
        "traficoactual": "Bajo"
}
```

Y en el móvil se recibirá un SMS con un texto como el siguiente: “El tráfico actual en la calle Paseo La Castellana es Bajo”. En este caso se hace uso de la API del punto 5.1.1.

5.1.4 Uso de la API Rest para consultar los niveles de tráfico predichos

Para utilizar la API Rest que se encuentra en Python con Flask en una instancia VPS en OVH, que a su vez ejecuta un modelo implementado con Apache Spark, es necesario hacer una consulta con los siguientes datos:

- Método: POST
- URL: **https://trafico.ml:50002/prediccion**
- Encabezado: Content-Type:application/json
- Cuerpo:

```
{
   "calle": "Paseo La Castellana",
   "fecha": "31/08/19 20:00"
}
```

El valor 4 se reemplaza dinámicamente por la cantidad de vehículos contados, sin embargo, desde el punto de vista de las pruebas, se puede utilizar cualquier número.

La respuesta recibida tiene la siguiente forma:

- Encabezado: Content-Type:application/json
- Cuerpo:

```
{
   "calle": "Paseo La Castellana",
   "fecha": "31/08/19 20:00",
   "prediccion": "Medio" }
```

5.1.5 Uso de Whatsapp para utilizar el servicio de consulta de tráfico

Para utilizar el servicio de consulta a través de Whatsapp es necesario realizar la secuencia de pasos que aparece a continuación:

1. Agregar al número telefónico de Whatsapp asignado por Twilio en el móvil.
2. Abrir un chat en Whatsapp con dicho número.
3. La primera vez hay que escribir “join place-clean” para comenzar a usar el bot.
4. Escribir el “Tráfico en XXXX” donde XXXX es el nombre de la calle, por ejemplo: “Paseo La Castellana”
5. Esperar la respuesta. Se recibirá un mensaje con un texto como el siguiente: “El tráfico actual en la calle Paseo La Castellana es Bajo”

Vale la pena recordar que este *bot* está utilizando una cuenta de Whatsapp para Empresas gestionada por Twilio, que hace una llamada HTTP a una instancia de Microsoft Azure Logic Apps, la cual utiliza una Microsoft Azure Function programada en C# para serializar el cuerpo obtenido desde Twilio, posteriormente se utiliza la API del punto 5.1.1.

También se puede escribir cualquier otro mensaje, de forma que en este caso el *bot* responde con una información de cómo utilizarlo.

5.1.6 Uso de Telegram para utilizar el servicio de consulta de tráfico

Para utilizar el servicio de consulta a través de Telegram es necesario realizar las acciones que se sintetizan en los siguientes pasos:

1. Abrir un chat en Telegram con el bot @Trafico_Madrid_bot.
2. Escribir “/calles” para obtener una lista de las calles que están monitorizadas.
3. Escribir “/trafico XXXX”, donde XXXX es el nombre de la calle, por ejemplo: “Paseo La Castellana”.
4. Esperar la respuesta,

Vale la pena recordar que este *bot* está programado en Node-Red en una instancia VPS de OVH que hace uso de Red-Bot y de la API del punto 5.1.1.

También se puede escribir cualquier otro mensaje, en cuyo caso responde con una

información de cómo utilizarlo.

5.2 Rendimiento y tiempos de ejecución obtenidos

Como se ha indicado previamente, uno de los objetivos prioritarios es determinar la posibilidad de implantación de la aplicación para tiempo real. Con tal propósito en esta sección se presentan los tiempos de ejecución obtenidos al usar cada uno de los servicios utilizados.

5.2.1 Análisis de las imágenes en el dispositivo EIC MS Vision 500 con NN

Para la realización de las pruebas se han utilizado 65 vídeos obtenidos con el dispositivo EIC MS Vision 500 obteniendo en total un conjunto de 4500 imágenes que son las finalmente analizadas. En promedio, se requieren 0.95 segundos para el procesamiento de cada frame, que representa un valor aceptable teniendo en cuenta la carga computacional que conlleva este tipo de procesos, principalmente por comparaciones similares, tal y como se describe en el capítulo dos. El objetivo del análisis es la identificación de los vehículos para proceder al cómputo del flujo de los mismos en la vía de tránsito correspondiente. En cualquier caso, el reconocimiento sólo resulta efectivo, alcanzando, en promedio, el 98% de éxito, para vehículos cuya imagen en el frame tenga una resolución mínima de 80x80 píxeles, sobre imágenes cuya resolución es de 800x1200 píxeles. En cualquier caso, el tipo de vehículos reconocido es: Coche (*car*), camión (*truck*) y autobús (*bus*). En cualquier caso, para compatibilizar tamaños de los objetos según el tipo de R-CNN utilizado siempre cabe la posibilidad de adaptar el sistema óptico para conseguir las resoluciones en tamaño óptimas para el reconocimiento.

Con carácter ilustrativo, en la figura 5.1 se observan los resultados del análisis de una imágen con el modelo R-CNN utilizado e integrado, donde se puede apreciar que todos los coches se reconocen correctamente, excepto uno que se encuentra en la esquina superior izquierda, que lo reconoce como un bote debido a la resolución espacial del objeto en la imagen.

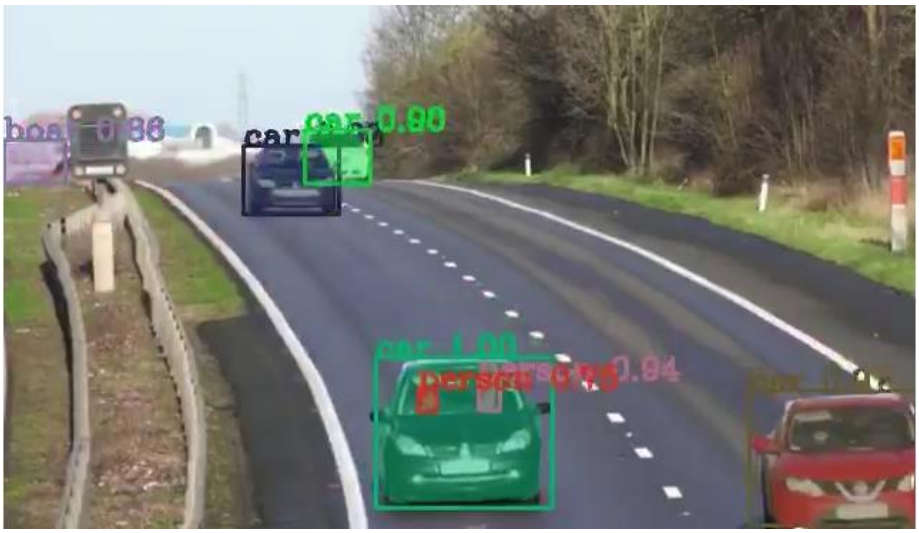

Figura 5.1 Detección de vehículos con la R-CNN integrada.

5.2.2 Despliegue de contenedores en los dispositivos con Azure IoT Edge

La asignación de variables de entorno diferentes a cada dispositivo, actualización de imágenes y contenedores docker se gestiona a través de Azure IoT Edge.

Dado que las imágenes docker se pueden ver como la aplicación de muchas imágenes docker, cuando se actualiza el código en Python que implementa la lógica, sólo se cambia la imágen que se encuentra en lo alto de la pila, lo cual permite que la actualización en los dispositivos se realice de manera suficientemente rápida. Cuando se realiza un cambio de este tipo en una imágen desde Visual Studio Code, primero es necesario registrar el contenedor en ACR y luego implementar el contenedor en los dispositivos. Los tiempos de ejecución de estas dos tareas se sitúan por debajo de un minuto, dependiendo del tipo y grado de conexión a internet, dado que en el caso actual, los cambios sólo son de unos 27 MB. En la figura 5.2 se muestra una captura de pantalla donde se observa el registro de una nueva versión de una imagen en ACR desde Visual Studio Code, pudiéndose apreciar que la única capa actualizada es la superior.

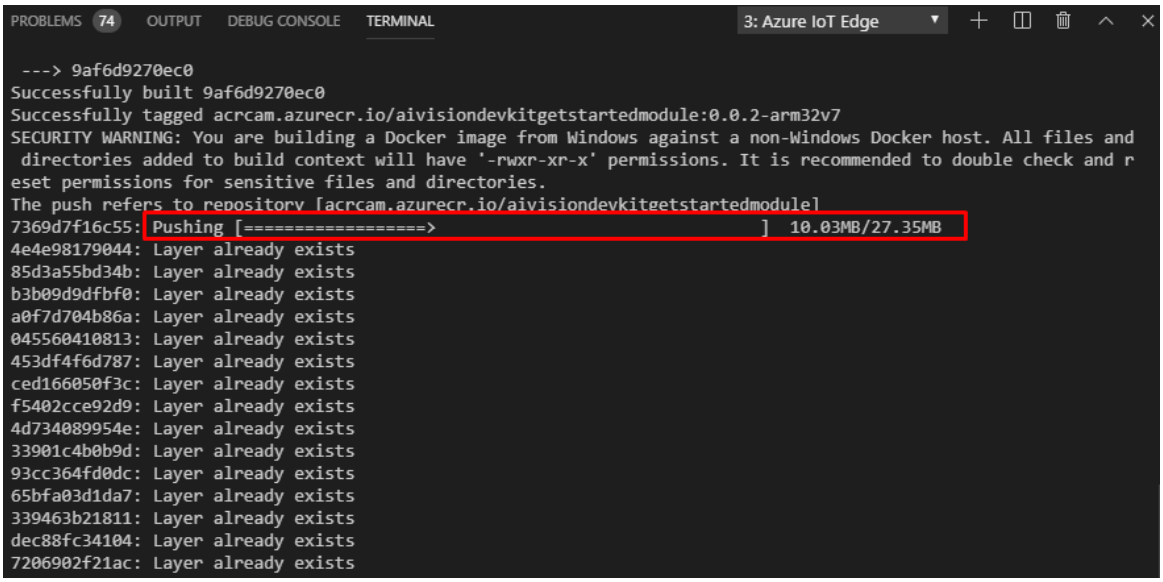

Figura 5.2 Captura de pantalla de actualización del contenedor Docker en ACR.

5.2.3 Modelo predictivo del tráfico

Como se ha mencionado previamente, el modelo predictivo del tráfico implementado con Apache Spark en Python seleccionado es una regresión lineal, con datos simulados se logró conseguir los resultados que se muestran en la figura 5.3.

```
errors = [ (name, eval.evaluate(res_test)) for name,eval in evaluators]
print(errors)
```

```
[('r2', 0.8897935897074931), ('mae', 0.604552336031624), ('mse', 12.26360940060961), ('rmse', 3.501943660399123)]
```

Figura 5.3 Resultados de la regresión lineal utilizando datos simulados.

Como se ha indicado previamente, el diseño de aplicación modular permite la integración de módulos diferentes según los requerimientos del momento. Así, en el caso de sustituir el módulo de regresión lineal por otro de regresión con Árbol de decisión, el resultado es:

```
[('r2', -315.7120418848168), ('mae', 1.4791666666666667), ('mse', 105.02083333333333), ('rmse', 10.247967278115857)]
```

Si se comparan ambos resultados, claramente conviene elegir como modelo predictivo el primer caso, sin embargo, dado que estos son datos simulados, se dejó implementada la infraestructura lista para implementar tanto el modelo de regresión, como cualquier otro soportado por Apache Spark.

Dado que Apache Spark consume para cada solicitud una gran cantidad de tiempo en asignar y distribuir las tareas de *map* y *reduce*, independientemente del número de tareas, no conviene utilizar el modelo para predecir un sólo valor. Si se intenta predecir un único valor, se pueden conseguir resultados como los mostrados en la figura 5.4, donde se puede observar que una única predicción puede demorar 26.64 segundos.

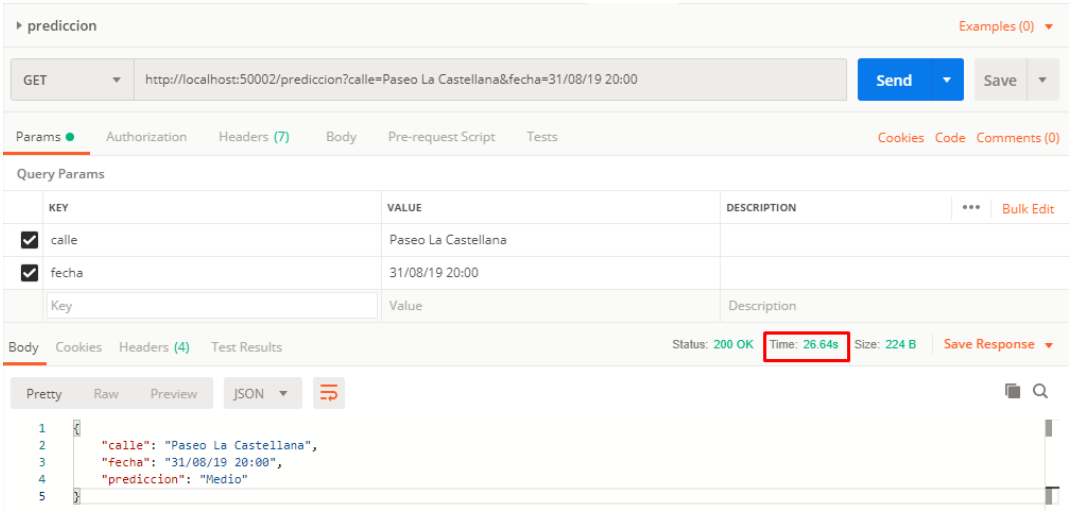

Figura 5.4 Uso de la API de predicción del tráfico desde Postman.

Es por esto, por lo que lo mejor es realizar predicciones para cada hora y cada día en una sola ejecución y almacenar los datos y resultados en una base de datos, de modo que cada vez que se realiza una predicción, no sea necesario utilizar el modelo predictivo, por lo tanto, siendo esto así, podría ahorrarse la necesidad de la instancia Iaas VPS en OVH.

5.2.4 Uso de la API con AWS Lambda

El uso de las AWS Lambda es una decisión muy acertada debido a su escalabilidad, el único detalle en contra detectado es que las Lambda se "enfrían" siendo necesario "calentarlas" para su reactivación. Estos términos se refieren a que, si una Lambda no se usa durante un cierto tiempo, dejan de ser accesibles con la rapidez deseada, así que la primera petición (cuando está "fría") puede demorar unos dos o tres segundos más que el resto (cuando ya está "caliente"). A modo ilustrativo, a continuación, se muestran dos

capturas de pantalla, en la primera (figura 5.5) la lambda está fría y en la segunda (figura 5.6) caliente, observándose en los correspondientes recuadros, marcados en rojo, las diferencias sustanciales de tiempo empleado en ambos casos.

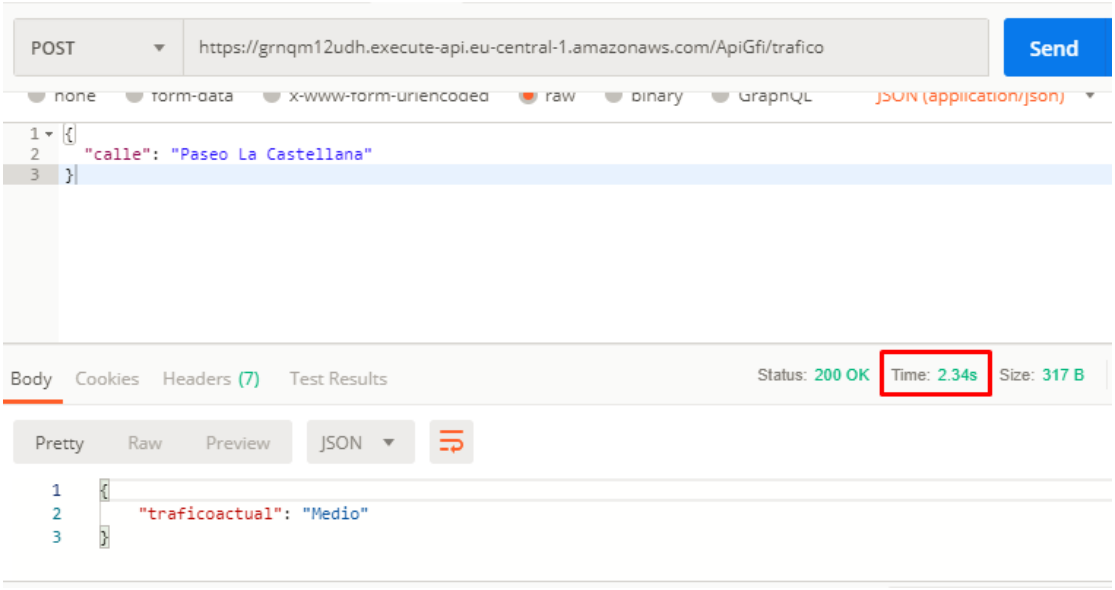

Figura 5.5 Uso de API en AWS Lambda después de más de 15 min sin uso.

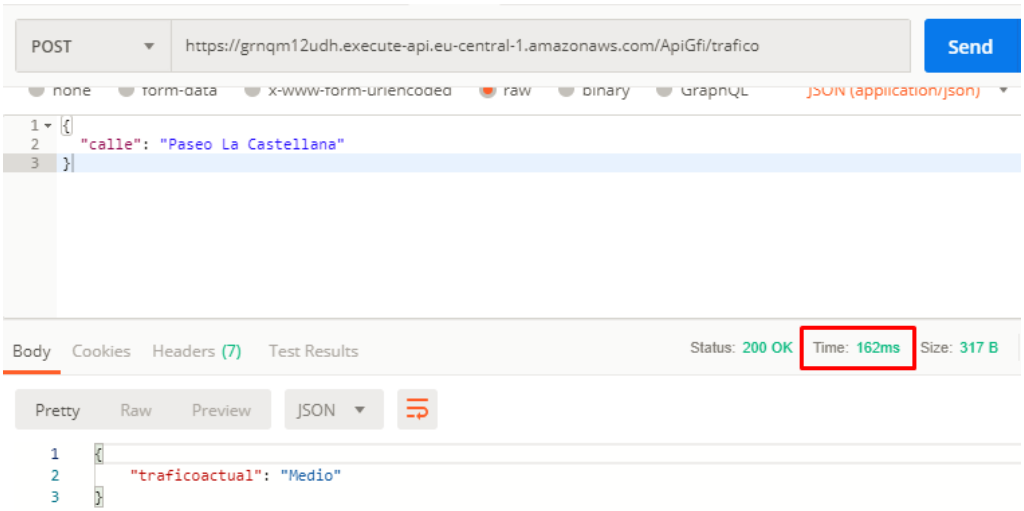

Figura 5.6 Uso de API en AWS Lambda después de menos de 15 min sin uso.

Es decir, si se le da un alto uso a la API, funciona mejor que si se usa poco, lo cual es favorable ya que va a tener tiempos de respuesta siempre de alrededor de 160 milisegundos, sin importar la cantidad de peticiones simultáneas realizadas.

5.2.5 Uso de MongoDB Atlas

La base de datos utilizada en MongoDB Atlas, hospedada en Azure, tiene unos tiempos de lectura inferiores a 10 milisegundos, en la figura 5.7 se muestra una captura del dashboard de MongoDB Atlas donde se puede observar una gráfica que indica que el tiempo máximo alcanzado para lecturas o escrituras fue de 7 milisegundos.

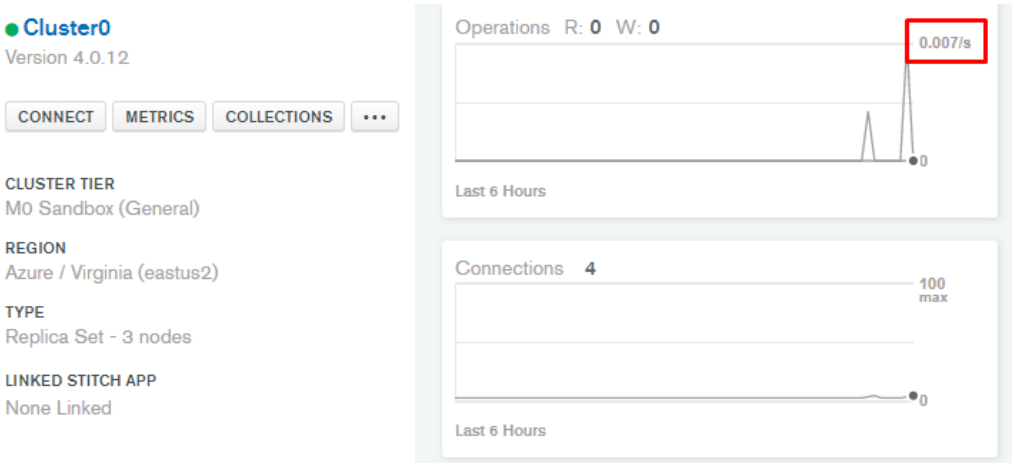

Figura 5.7 Dashboard de MongoDB Atlas.

5.2.6 Uso de Telegram para conocer el tráfico actual

La aplicación desarrollada con Node-Red donde se implementa el uso de Red-Bot para gestionar un *bot* de Telegram tiene respuestas muy rápidas, a

En la figura 5.8 se puede muestra una conversación con el bot donde se observa que responde en menos de 3 segundos, e incluso, a veces, aun utilizando la API de AWS Lambda, se obtienen tiempos de respuesta inferiores a 1 segundo.

Figura 5.8 Conversación con bot de Telegram implementado en Node-Red.

5.2.7 Uso de Whatsapp

El uso de Whatsapp se ha integrado entre Azure Logic App, Azure Functions y Twilio. En la figura 5.9 se muestra una conversación con el bot.

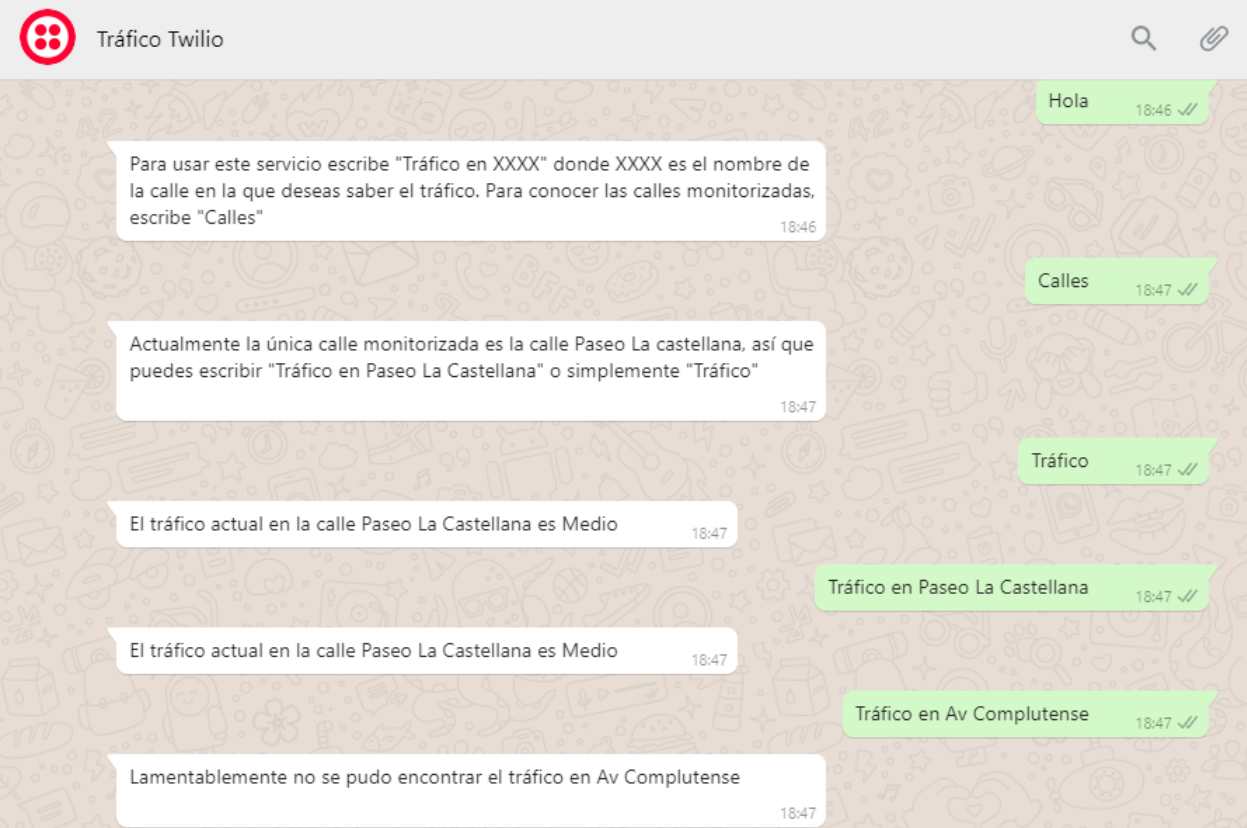

Figura 5.9 Bot en Whatsapp con Twilio, Azure Logic Apps y Azure Functions.

Como se evidencia, a partir de los tiempos que aparecen en los mensajes, las respuestas duran menos de un minuto, sin embargo, lo interesante está en ver los logs tanto en Logic Apps como en Twilio, a pesar de que los errores están manejados dentro de la Logic App programada, se forzó a que provocara errores para que se pueda ver el log. En la figura 5.10 se muestra la duración de cada ejecución de las LogicApps, junto con el estado de la ejecución

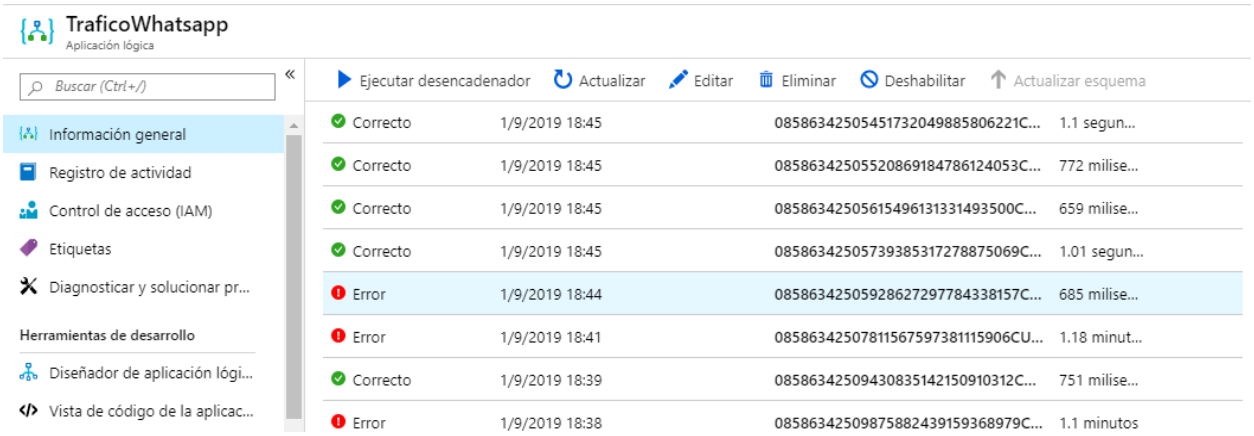

Figura 5.10 Captura de pantalla de logs de ejecuciones en Azure Logic Apps.

En la figura 5.11 se muestra una captura de pantalla donde se puede observar el log de los errores en Twilio obtenidos por las mismas ejecuciones con los errores de la figura 5.10.

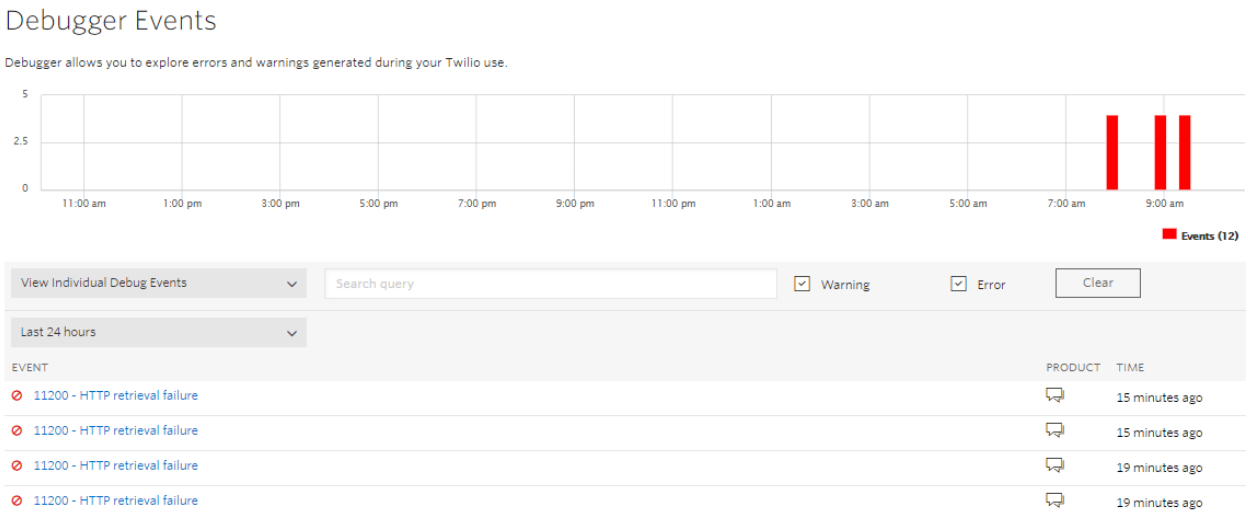

Figura 5.11 Captura de pantalla de logs de eventos con error en Twilio.

Capítulo 6 Conclusiones y Trabajo futuro

A continuación, se exponen las conclusiones y trabajo futuro en relación al desarrollo del presente trabajo. En este sentido se ha de tener en cuenta el momento de su desarrollo bajo el paradigma IoT donde existen diversas tecnologías, muchas de las cuales se encuentran en continua evolución, de suerte que en el futuro algunos de los problemas actuales, se habrán superado con toda probabilidad, siendo este trabajo en sí mismo una contribución al respecto.

6.1 Conclusiones

Las conclusiones destacables más relevantes llevadas a cabo en la elaboración del presente trabajo se pueden concretar, bajo la perspectiva de análisis de viabilidad, en dos aspectos concretos:

a) Eficiencia y soporte tecnológico
   - A pesar de que algún problema pueda tener diversas soluciones, es necesario buscar apropiadamente cuál es la óptima, ya que no necesariamente las soluciones más sencillas son las correctas, sin embargo, esto tampoco implica que una solución más compleja sea la mejor.
   - El uso de tecnologías muy nuevas puede ser problemático a la hora de desarrollar, ya que es posible que la documentación sea muy escasa, además de que puede presentar problemas de confiabilidad o de soporte a futuro.
   - Cuando se utilizan tecnologías nuevas, muchas en fase experimental, la comunidad de desarrolladores resulta de gran ayuda a la hora de buscar soluciones a los problemas encontrados.
   - Así como la comunidad de desarrolladores sirve de apoyo, es necesario y conveniente contribuir en sentido inverso proporcionando soluciones a la comunidad. Por esta razón, se realizaron 4 Pull Request en GitHub con correcciones de errores encontrados.
   - Crear un sistema IoT es algo complejo que requiere mucho tiempo de

diseño, investigación, configuración, programación y corrección de errores.

b) Soluciones técnicas

- Se ha diseñado un sistema que integra distintas tecnologías y servicios IoT y que permite el conteo del flujo de tráfico vehicular, el almacenamiento de datos obtenidos y la predicción de flujos en distintos momentos.
- Se ha diseñado una solución basada en reconocimiento de imágenes para conteo vehicular mediante técnicas de aprendizaje profundo basadas en Redes Neuronales Convolucionales.
- Se han utilizado tecnologías de implantación en la Nube con el fin de poder desarrollar sistemas escalables.
- Debido a que el proceso de virtualización en la Nube puede ser muy costoso, además de poco escalable, se han utilizado tecnologías del tipo *serverless* en la Nube, que además hacen posible la disminución de costos, con fines de su implantación en el futuro.
- Es de gran utilidad probar diferentes tecnologías antes de decidir un diseño, ya que, gracias a esta filosofía, se logró pasar de 80 segundos a menos de 1 segundo por *frame* analizado luego de cambiar de Sistema Operativo, Lenguaje de Programación, Tipo de Red Neuronal, algoritmo de procesamiento. Cada cambio produjo mejoras en el rendimiento, siendo unas más significativas que otras.
- El uso de Azure IoT Hub resulta muy conveniente para poder aprovisionar, actualizar y mantener controlada la flota de dispositivos IoT.
- El uso de contenedores Docker con los códigos desarrollados permite, de manera muy sencilla, mantener un control de versiones de imágenes y códigos, además ayuda a implementar cambios en dispositivos de distintas arquitecturas de manera muy rápida ya que sólo se descarga en los dispositivos la capa que haya cambiado.
- Actualmente, AWS tiene mayor confiabilidad al momento de ejecutar

códigos en Python en forma de *serverless*, con sus funciones Lambda, que Azure con las Function Apps. Esto se debe a que, en AWS, las Lambdas tienen soporte mientras que en Azure no. Como muestra de este hecho, se presenció un problema en Azure Functions que hizo que un código que funcionaba, dejara de funcionar repentinamente y luego, funcionara correctamente de nuevo. Finalmente se comprobó que fue un problema de Azure porque varios usuarios reportaron el mismo problema.

- El uso de una base de datos NoSQL como lo es MongoDB al principio representó un problema debido a que, posiblemente, una base de datos relacional hubiese sido más cómoda para entrenar un modelo de regresión. Sin embargo, el uso de bases de datos NoSQL pueden ser una gran ventaja en sistemas IoT dado que existe una gran posibilidad de que cada dispositivo esté en circunstancias diferentes y esto podría ocasionar que unos dispositivos envíen más datos que otros sin generar un conflicto en la base de datos por cambio de esquema.
- El uso de API Rest para ofrecer la información de manera abierta a los usuarios permite que este servicio sea integrado de una manera muy cómoda y fácil con distintas aplicaciones desarrolladas en distintos lenguajes, en especial porque se usa el formato JSON para presentar el contenido.
- El desarrollo de aplicaciones separadas como microservicios permiten reducir tiempos de ejecución, así como aumentar la interoperabilidad.
- Probablemente la solución planteada en el presente trabajo no sea la más sencilla posible, pero sirve de ejercicio didáctico para conocer distintas tecnologías. En resumen, entre otras cosas, en este trabajo se utilizó lo siguiente:
    - Frameworks:
        - TensorFlow
        - .Net

        - Apache Spark
    - Redes neuronales
        - AlexNet
        - MsCoco
        - MaskRCNN
        - computervision.ai
        - ResNet
    - Lenguajes de Programación:
        - Shell
        - Python
        - C#
        - JavaScript
    - Bases de datos:
        - MongoDB
    - Servicios informáticos en la Nube:
        - Azure:
            - IoT Hub
            - IoT Edge
            - Azure Container Registry
            - Azure Function Apps
            - MongoDB
        - AWS:
            - API Gateway
            - Lambda Functions
            - Amazon S3
        - Google Cloud Platform
            - Google Colab
        - Twilio
            - Programmable SMS

        - Whatsapp API
    - Herramientas y Software:
        - Docker
        - Visual Studio Code
        - Jupyter Notebooks
        - ADB
        - Node-Red
        - Ngrok
        - Visual Studio
        - Freenom
        - Certbot
        - Let´s Encrypt
        - UFW

6.2 Trabajo futuro

Es necesario seguir investigando sobre nuevas tecnologías y soluciones, que pueden integrase o incluso sustituirse por las propuestas dado el carácter modular integrado de la propuesta.

De forma más específica, se recomienda desarrollar con más profundidad el modelo de predicción con datos reales para lograr mayor precisión en la previsión de los niveles de tráfico. En este sentido conviene señalar que el presente trabajo se ha orientado principalmente a diseñar la arquitectura para recolectar y publicar la información que a desarrollar un algoritmo preciso de predicción del tráfico. Dado que existen muchos factores que influyen en el tráfico como lo son el clima, las huelgas de transportistas del transporte público, de los eventos, temporada del año, entre otros, es necesario desarrollar sistemas que tengan esto en cuenta.

El presente trabajo sirve como inicio para otros sistemas IoT basados en reconocimiento de imágenes. Realmente esta arquitectura propuesta no sólo funciona para detectar coches en las vías, puede ser utilizado para el reconocimiento de otros objetos y puede ser utilizado para predecir o servir cualquier otro tipo de información. En

este sentido, conviene señalar que el valor más relevante que ofrece el trabajo es proporcionar una solución IoT integrada aplicable a diferentes ámbitos y sectores.

Se recomienda, como acción de futuro, ya sea para dar continuidad a este proyecto o para un proyecto similar o uno totalmente diferente, el uso de servicios *serverless* en la Nube con el fin de aumentar la agilidad y la escalabilidad, así como disminuir los tiempos de desarrollo y costos de implementación y mantenimiento.

En línea con lo expresado anteriormente, en relación a la evolución en el tiempo de las diferentes tecnologías, señalar que al inicio de este trabajo la documentación de la cámara utilizada era muy escasa, de hecho, la propia cámara no estaba disponible en el mercado, fue justo un mes antes de la finalización de esta memoria que salió oficialmente al mercado, así que muy posiblemente la documentación de la cámara mejore en el futuro, así que se recomienda revisar periódicamente la documentación oficial ante las previsibles actualizaciones.

Capítulo 7 Introduction

7.1 Problem statement

The technologies developed in the field of Internet of Things, IoT, provide feasible solutions in different areas of human activity. One of them is related to vehicle traffic monitoring and control in urban and interurban roads, which is where the present work is focused.

The current pace of life has led people to be in constant motion, since, at times, the hours of the day are few to carry out all the activities planned for a day. Taking this into account, a factor of great importance in the daily routine is the fact of being able to make the most of the time, so it is necessary to avoid those situations where valuable activities are not carried out.

On the other hand, it is possible to compare and relate two important factors: time and money, since, by spending hours without carrying out fruitful activities, that time is being stopped in activities that can, for example, generate income.

A daily problem affecting the fulfillment of this objective is traffic jams, since, according to studies carried out by the European Commission, in 2016 drivers in Barcelona lost on average 119 hours a year in traffic jams, but this is also a big problem present in other cities such as Madrid, Seville and Valencia, since in the first one they lose up to 105.

All these wasted hours translate into economic losses, as they amount to about 15 daily workdays lost inside a car. Taking into account all these considerations and the places where these problems occur, in Spain around 5,500 million euros are lost annually.

On the other hand, this problem has been growing over time, since in the last three years it can be said that, at least in Madrid, congestion has almost doubled. This exposed problem can not only produce a negative effect on the economy of the country, but also means a problem for the environment, which ends up resulting in health complications and accidents on the roads.

Traffic congestion produces a high concentration of toxic gases in the air, which results in an increase in air pollution. This fact affects the general population but, more

pronouncedly, those who already suffer from respiratory or cardiovascular problems, while taking into account the elderly population that is most vulnerable.

Studies carried out in the Autonomous Communities of Spain reveal that pollution in some Spanish regions exceeds the annual limits established by the World Health Organization (WHO). Madrid is a city where the maximum recommended limits of urban pollution, specifically related to traffic congestion, are sometimes exceeded.

According to the data provided by the environmental consultant A. T. Kearney, it is necessary to act in the reduction of traffic since this is one of the main causes of air pollution, estimated at more than 40% of its weight. In addition, these studies indicate that this fact may also cause increases in mortality at an early age.

Other studies that demonstrate the seriousness of this problem are those carried out by the National School of Health, which estimates that the number of annual deaths from pollution in Spain reaches almost 2,700, while WHO sets this figure at 7,000 and the Global Institute from Barcelona at a value of 21,000. However, the highest calculated figure is provided by the European Environment Agency, which exceeds 31,500 deaths per year due to air pollution.

Not only is air pollution generated, sonic pollution also occurs, climate change is aggravated and negative effects on nature also occur.

Adding all of the above, the economic impact produced increases and is estimated to exceed 10,000 million euros per year in losses.

7.2 Motivation and approach with IoT

Despite the discouraging data exposed above, there are appropriate technological solutions that serve to reduce the effects caused by traffic jams, the solution to this problem is linked to the fact that somehow decreases the time spent daily in these traffic jams due to excess traffic circulating in the same way.

Thanks to technological advances, it is possible to find solutions that can help reduce traffic or at least reduce how quickly it increases. The way to find this solution is based on offering timely information to the users of the roads, indicating the number of vehicles that are on them and thus avoiding them, as well as the authorities responsible

for regulating traffic for efficient management of the traffic lanes and parking spaces.

This fact would produce a better distribution of traffic and thereby avoid the generation and accumulation of gases such as CO_2, one of the main causes of air pollution. In addition, the use of horns and the production of tux that affect the city would be avoided.

A solution to this problem can be developed with the integration of different technologies, on the one hand it is possible to use IoT devices that are capable of measuring the flow of vehicles and then send this data to other devices, so that the necessary information is processed to reduce traffic flow.

Another important development of technology, which is also the one used in this work, is the use of IoT technologies equipped with Artificial Intelligence, which can be useful for designing and using efficient systems, in this case through image recognition. Taking this into account, we seek to implement a system that recognizes images of vehicles, counts them and sends this data to be stored in the cloud, as well as other location, time, date and time data that will allow access to them in a manner public, which will serve to make traffic prediction models, being useful both for the entire software development community at a technical level and for the traffic management entities and the users of the vehicles themselves.

This whole system provides a solution for the detection of vehicles based on the IoT framework, where different technologies, protocols and services are integrated, making it a scalable and feasible solution to monitor and predict traffic flows at certain times.

The contribution made in this work is based on the fact that, after an exhaustive study of the various available technologies, those with a high potential based on relevant concepts such as scalability, price, ease of maintenance of the code, provisioning of devices and, especially in the reliability of system operation were selected for integration.

On the other hand, it should be noted that very new technologies were used with a high number of errors because many of them were found in beta versions. However, along with the fact that free software was used, it made possible the proposal of

corrections to the developer community, which were accepted and are now accessible to other users.

Being new and poorly studied technologies, it results in documentation being very scarce and presenting errors, which causes failures in its application and complications in its development.

In summary, the specific contributions made in this work are:

1) Comparative study of novel techniques not sufficiently developed.
2) Integration of the different technologies selected to arrive at the proposed global modular solution.
3) Insertion of image recognition methods through deep learning techniques based on Convolutionary Neural Networks.

7.3 Objectives

7.3.1 General Objective

The main goal is to design a scalable IoT system that, through the integration of multiple technologies and services, allows measuring and storing traffic data on land traffic routes and that, from the training of a predictive model with the data obtained, is able to predict the traffic levels. The system must allow the dissemination of the information obtained not only for the moment at which the vehicle flow is measured but for any other time.

7.3.2 Specific Objectives

- Select the input device to measure traffic.
- Select the type of neural network to recognize the images and implement their code.
- Select the hardware and operating system that will be used in the input devices.
- Decide which devices or services are going to perform the calculation of the neural network.

- Select the type of storage for the data obtained.
- Select the type of regression to be used to predict traffic levels.
- Select in which infrastructure or services the regressor will be implemented, to train the model or to predict with it.
- Select the means of dissemination of information.
- Integrate all the technologies used to work together.

Capítulo 8 Conclusions and Future Work

The conclusions and future work related to the development of this work are presented below. In this way, the present time of its development must be taken into account under the IoT paradigm where there are several technologies, many of which are in continuous evolution and development, so that in the future some of the current problems will have been overcome with all probability, this work itself makes another contribution in this regard.

8.1 Conclusions

The most relevant outstanding conclusions carried out in the preparation of this work can be specified, under the perspective of feasibility analysis, in two specific aspects:

a) Efficiency and technological support
    - Although some problem may have different solutions, it is necessary to properly search for the most optimal one, since the simplest solutions are not necessarily the right ones, however, this does not imply that a more complex solution is the best one.
    - The use of very new technologies can be problematic at the time of development, since it is possible that the documentation is very scarce, in addition to that it can present problems of reliability or support in the future.
    - When new technologies are used, many in the experimental phase, the developer community is very helpful in finding solutions to the problems encountered.
    - Just as the developer community serves as support, it is necessary and convenient to contribute in the opposite direction by providing solutions to the community. For this reason, 4 Pull Request were performed on GitHub with corrections of errors found.
    - Creating an IoT system is something complex that requires a lot of design, research, configuration, programming and error correction time.

b) Technical Solutions

- A system has been designed that integrates different IoT technologies and services and allows the counting of vehicular traffic flow, the storage of data obtained and the prediction of flows at different times.
- A solution based on image recognition for vehicle counting has been designed using deep learning techniques based on Convolutional Neural Networks.
- Cloud deployment technologies have been used in order to develop scalable systems.
- Because the process of virtualization in the Cloud can be very expensive, as well as not very scalable, serverless technologies have been used in the Cloud, which also make it possible to reduce costs, with the purpose of implementing them in the future.
- It is very useful to try different technologies before deciding on a design, because thanks to this philosophy, it was possible to go from 80 seconds to less than 1 second per frame analyzed after changing the Operating System, Programming Language, Neural Network Type, processing algorithm. Each change produced improvements in performance, some being more significant than others.
- The use of Azure IoT Hub is very suitable to be able to provision, update and control the fleet of IoT devices.
- The use of Docker containers with the developed codes allows, in a very simple way, to maintain a control of versions of images and codes, in addition it helps to implement changes in devices of different architectures very quickly since only the layer is downloaded in the devices that has changed.
- Currently, AWS has greater reliability when executing codes in Python in the form of serverless, with its Lambda functions, than Azure with Function

Apps. This is because in AWS, Lambdas have support while in Azure they don't. As proof of this fact, there was a problem in Azure Functions that caused a code that worked, stopped working suddenly and then, worked correctly again. Finally, it was verified that it was an Azure problem because several users reported the same problem.

- The use of a NoSQL database such as MongoDB at the beginning represented a problem because, possibly, a relational database would have been more convenient to train a regression model. However, the use of NoSQL databases can be a great advantage in IoT systems since there is a great possibility that each device is in different circumstances and this could cause some devices to send more data than others without generating a conflict in the database by schema change.
- The use of API Rest to offer information openly to users allows this service to be integrated in a very comfortable and easy way with different applications developed in different languages, especially because the JSON format is used to present the content.
- The development of separate applications such as micro-services can reduce execution times, as well as increase interoperability.
- The solution proposed in this work is probably not the simplest possible, but it serves as a didactic exercise to learn about different technologies. In summary, among other things, the following was used in this work:
    - Frameworks:
        - TensorFlow
        - .Net
        - Apache Spark
    - Neural Networks
        - AlexNet
        - MsCoco
        - MaskRCNN

        - computervision.ai
        - ResNet
    - Programming Languages:
        - Shell
        - Python
        - C#
        - JavaScript
    - Databases:
        - MongoDB
    - Cloud computing services:
        - Azure:
            - IoT Hub
            - IoT Edge
            - Azure Container Registry
            - Azure Function Apps
            - MongoDB
        - AWS:
            - API Gateway
            - Lambda Functions
            - Amazon S3
        - Google Cloud Platform
            - Google Colab
        - Twilio
            - Programmable SMS
            - Whatsapp API
    - Tools and Software:
        - Docker
        - Visual Studio Code
        - Jupyter Notebooks

- ADB
- Ngrok
- Node-Red
- Visual Studio
- Freenom
- Certbot
- Let´s Encrypt
- UFW

8.2 Future Work

It is necessary to continue investigating new technologies and solutions, which can be integrated or even replaced by the proposals given the integrated modular nature of the proposal.

More specifically, it is recommended that the prediction model be further developed to achieve greater accuracy in forecasting traffic levels. In this regard, it should be noted that the present work has been oriented mainly to design the architecture to collect and publish the information that will develop a precise traffic prediction algorithm. Since there are many factors that influence traffic such as the weather, public transport carrier strikes, events, season of the year, among others, it is necessary to develop systems that take this into account.

This work serves as a starting point for other IoT systems based on image recognition. Currently the proposed architecture not only works to detect cars on the tracks, it can be used for the recognition of other objects and can be used to predict or serve any other type of information. In this regard, it should be noted that the most relevant value offered by the work is to provide an integrated IoT solution applicable to different fields and sectors.

It is recommended, as a future action, either to give continuity to this project or for a similar or a fully different project, the use of serverless services in the Cloud in order to increase agility and scalability, as well as decrease times of development and

implementation and maintenance costs.

In line with the above, in relation to the evolution in time of the different technologies, note that at the beginning of this work the documentation of the camera used was very scarce, in fact, the camera itself was not available in the market. It was just one month before the end of this report that it officially went on the market, so it is very possible that the camera documentation will improve in the future, so it is recommended to periodically review the official documentation for the expected updates.

Bibliografía

Amazon. (2019). *Amazon S3*. Recuperado el 09 de 2019, de https://aws.amazon.com/es/s3/

Andrade, A. (06 de 2016). *How To Set Up a Jupyter Notebook to Run IPython on Ubuntu 16.04.* Recuperado el 02 de 2019, de https://www.digitalocean.com/community/tutorials/how-to-set-up-a-jupyter-notebook-to-run-ipython-on-ubuntu-16-04

Apache Spark. (2019). Recuperado el 09 de 2019, de https://spark.apache.org/

Apple. (2019). *Usar Atajos de Siri.* Recuperado el 09 de 2019, de https://support.apple.com/es-es/HT209055

AWS-01. (2019). *Sin servidor.* Recuperado el 09 de 2019, de https://aws.amazon.com/es/serverless/

AWS-02. (2019). *AWS IoT.* Recuperado el 09 de 2019, de https://aws.amazon.com/es/iot/?nc2=h_m1

AWS-03. (2019). *Características de AWS Lambda.* Recuperado el 09 de 2019, de https://aws.amazon.com/es/lambda/features/

AWS-04. (2019). *Precios de AWS Lambda.* Recuperado el 09 de 2019, de https://aws.amazon.com/es/lambda/pricing/

AWS-05. (2019). *Nombres de recursos de Amazon (ARN) y espacios de nombres de servicios de AWS.* Recuperado el 09 de 2019, de https://docs.aws.amazon.com/es_es/general/latest/gr/aws-arns-and-namespaces.html

AWS-06. (2019). *Amazon API Gateway.* Obtenido de https://aws.amazon.com/es/api-gateway/

AWS-07. (2019). *Precios de Amazon API Gateway.* Obtenido de https://aws.amazon.com/es/api-gateway/pricing/

AWS-08. (2019). *AWS Identity and Access Management (IAM).* Recuperado el 09 de 2019, de https://aws.amazon.com/es/iam/

AWS-09. (2019). *Precios de Amazon EC2.* Recuperado el 09 de 2019, de https://aws.amazon.com/es/ec2/pricing/on-demand/

AWS-10. (2019). *Amazon DynamoDB.* Recuperado el 09 de 2019, de https://aws.amazon.com/es/dynamodb/

AWS-11. (2019). *AWS Toolkit for Visual Studio Code.* Recuperado el 09 de 2019, de https://aws.amazon.com/es/visualstudiocode/

AWS-12. (2019). *Amazon SageMaker.* Recuperado el 09 de 2019, de https://aws.amazon.com/es/sagemaker/

Azure-01. (2019). *Azure IoT.* Recuperado el 09 de 2019, de https://azure.microsoft.com/es-es/overview/iot/

Azure-02. (2019). *Azure Functions.* Recuperado el 09 de 2019, de https://azure.microsoft.com/es-es/services/functions/?&ef_id=CjwKCAjw-7LrBRB6EiwAhh1yX5DCypceYq2j-GhLp2KHeX4j6BqnOKOSr5lG3Hkb8YW5jGiMsg_YSRoCla8QAvD_BwE:G:s&OCID=AID2000115_SEM_iL5Q3KPM&MarinID=iL5Q3KPM_337948230646_azure%20functions_e_c__61529306777_kwd-

Azure-03. (2019). *Supported languages in Azure Functions.* Recuperado el 09 de 2019, de https://docs.microsoft.com/es-es/azure/azure-functions/supported-languages#languages-in-runtime-1x-and-2x

Azure-04. (2019). *Precios de Azure Functions.* Recuperado el 09 de 2019, de https://azure.microsoft.com/es-es/pricing/details/functions/

Azure-05. (2019). *Precios de Azure.* Recuperado el 09 de 2019, de https://azure.microsoft.com/es-es/pricing/

Azure-06. (2019). *Azure IoT Hub.* Recuperado el 09 de 2019, de https://azure.microsoft.com/es-es/services/iot-hub/

Azure-07. (2019). *What is Azure IoT Hub?* Recuperado el 09 de 2019, de https://docs.microsoft.com/es-es/azure/iot-hub/about-iot-hub

Azure-08. (2019). *What is Azure IoT Edge.* Recuperado el 09 de 2019, de https://docs.microsoft.com/es-es/azure/iot-edge/about-iot-edge

Azure-09. (2019). *Precios de Azure IoT Hub.* Obtenido de https://azure.microsoft.com/es-es/pricing/details/iot-hub/

Azure-10. (2019). *Container Registry.* Recuperado el 09 de 2019, de https://azure.microsoft.com/es-es/services/container-registry/

Azure-11. (2019). *What is Azure Logic Apps?* Recuperado el 09 de 2019, de https://docs.microsoft.com/es-es/azure/logic-apps/logic-apps-overview

Azure-12. (2019). *Precios de Azure Cosmos DB.* Recuperado el 09 de 2019, de https://azure.microsoft.com/es-es/pricing/details/cosmos-db/

Azure-13. (2019). *Blob Storage.* Recuperado el 09 de 2019, de https://azure.microsoft.com/es-es/services/storage/blobs/

Azure-14. (2019). *What is Computer Vision?* Recuperado el 09 de 2019, de https://docs.microsoft.com/es-es/azure/cognitive-services/computer-vision/home

Azure-15. (2019). *Precios de Machine Learning Studio.* Recuperado el 09 de 2019, de https://azure.microsoft.com/es-es/pricing/details/machine-learning-studio/

Azure-16. (2019). *A Smart Camera for the Intelligent Edge.* Recuperado el 09 de 2019, de https://aka.ms/visionaidevkit

Civantos, M. (16 de 02 de 2018). *Trybalite Technologies.* Obtenido de ¿Qué son las APIs REST?: https://tech.tribalyte.eu/blog-que-es-una-api-rest

Cloud, G. (2019). *Google Cloud Storage*. Recuperado el 09 de 2019, de https://cloud.google.com/storage/

Das, C. (05 de 2017). *What is machine learning and types of machine learning — Part-1*. Recuperado el 09 de 2019, de https://towardsdatascience.com/what-is-machine-learning-and-types-of-machine-learning-andrews-machine-learning-part-1-9cd9755bc647

DGT. (2016a). *Los atascos cuestan más de 140 millones de euros*. Recuperado el 02 de 2019 de 09, de Revista DGT: http://revista.dgt.es/es/noticias/nacional/2016/12DICIEMBRE/1205atascos-lo-que-cuestan.shtml#.XW0mNJMzat8

DGT. (2016b). *Medir el tráfico mediante los móviles*. Recuperado el 09 de 2019, de Revista

DGT.

Géron, A. (2017). Aprendizaje automático con Scikit-Learn y TensorFlow por Aurélien Géron (O'Reilly). *Aurélien Géron*, 978-1-491-96229-9.

González, P. (03 de 2019). *Apache Spark VS Hadoop Map Reduce*. Recuperado el 09 de 2019, de https://openwebinars.net/blog/apache-spark-vs-hadoop-map-reduce/

Google Maps. (2019). Recuperado el 09 de 2019, de https://www.google.es/maps/?hl=es

Jupyter. (2019). *Jupyter.* Recuperado el 09 de 2019, de https://jupyter.org/

LeCun, Y. (1989). *Generalization and network design strategies. Technical Report CRG-TR-89-4.* Toronto: University of Toronto. Recuperado el 09 de 2019, de http://yann.lecun.com/exdb/publis/pdf/lecun-89.pdf

Matich, D. J. (2001). *Redes Neuronales: Conceptos Básicos y Aplicaciones.* Rosario: Universidad Tecnológica Nacional.

McCarthy, J. &. (1990). Pioneer in Machine Learning. *AI Magazine*. Obtenido de https://doi.org/10.1609/aimag.v11i3.840

Microsoft. (2019). *Visual Studio.* Recuperado el 09 de 2019, de https://visualstudio.microsoft.com/es/

MongoDB. (2019). *MongoDB Atlas.* Recuperado el 09 de 2019, de https://www.mongodb.com/cloud/atlas/lp/general/try?jmp=search&utm_source=google&utm_campaign=GS_EMEA_Spain_Search_Brand_Atlas_Desktop&utm_term=mongodb%20cloud&utm_device=c&utm_network=g&utm_medium=cpc_paid_search&utm_matchtype=e&utm_cid=1718986525&utm_asa

Ngrok. (2019). *What is ngrok?* Recuperado el 09 de 2019, de https://ngrok.com/product

NIST. (09 de 2011). *The NIST Definition of Cloud.* Recuperado el 09 de 2019, de http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

Node-Red. (2019). *Node-RED.* Recuperado el 09 de 2019, de https://nodered.org/

OMS. (2005). *Guías de calidad del aire de la OMS relativas al material particulado, el ozono, el dióxido de nitrógeno y el dióxido de azufre Actualización mundial 2005*. Recuperado el 09 de 2019, de https://apps.who.int/iris/bitstream/handle/10665/69478/WHO_SDE_PHE_OEH_0

6.02_spa.pdf;jsessionid=8373A6173F7AF3D03464C59D3C6FA0A4?sequence=1

OVH. (s.f.). *Servidores privados virtuales SSD.* Recuperado el 09 de 2019, de https://www.ovh.es/vps/vps-ssd.xml

Prieto, E., Fanjul, M., & González, J. p. (2018). *Hacia un modelo social y sostenible de infraestructuras viarias en España.* Recuperado el 02 de 09 de 2019, de https://seopan.es/wp-content/uploads/2016/09/AT-KEARNEY_Hacia-un-modelo-social-y-sostenible-de-infraestructuras-viarias-en-Espa%C3%B1a.pdf

Rahul, K. (10 de 2018). *How to Setup Let's Encrypt SSL on Ubuntu 18.04 & 16.04 LTS.* Recuperado el 02 de 2019, de https://tecadmin.net/install-lets-encrypt-create-ssl-ubuntu/

Redmon, J. D. (2016). *You Only Look Once: Unified, Real-Time Object Detection.*

Riccio, F., & Guizado, J. (02 de 2018). *GoldenGate for Big Data - Streaming de Datos de Oracle Database a Kafka Server*. Recuperado el 09 de 2019, de https://www.oracle.com/technetwork/es/articles/database-performance/goldengate-for-bigdata-4408708-esa.html

Rodríguez, J. (02 de 2018). *Azure Logic Apps, ¿la versión más "Pro" de Microsoft Flow?* Recuperado el 09 de 2019, de https://itblogsogeti.com/2018/02/15/azure-logic-apps-la-version-mas-pro-de-microsoft-flow/

Román, V. (02 de 2019). *Machine Learning Supervisado: Fundamentos de la Regresión Lineal.* Recuperado el 09 de 2019, de https://medium.com/datos-y-ciencia/machine-learning-supervisado-fundamentos-de-la-regresi%C3%B3n-lineal-bbcb07fe7fd

Roseindia. (2019). *Install Spark on Ubuntu 18.04.* Recuperado el 02 de 2019, de https://www.roseindia.net/spark/install-spark-on-ubuntu-18.04.shtml

Rouse, M. (07 de 2017). *Microsoft Azure Functions.* Recuperado el 09 de 2019, de https://searchcloudcomputing.techtarget.com/definition/Microsoft-Azure-Functions

S. Ren, K. H. (2015). *Faster R-CNN: Towards real-time object detection with region proposal networks. .*

Sisalima, F. (2018). *Sistema para detección y conteo vehicular aplicando técnicas de visión artificial.* Loja: Universidad Nacional de Loja.

Somayya Madakam, R. R. (2015). Internet of Things (IoT): A Literature Review. *Journal of Computer and Communications*, 3, 164-173.

Stanescu, E. G. (2019). *Sistema de iluminación inteligente en carreteras.* Madrid: Universidad Complutense de Madrid. Recuperado el 09 de 2019, de https://eprints.ucm.es/51612/1/428601_EMANUEL_GEORGIAN_STANESCU_Memoria_Final_TFM_Sistema_de_Iluminacion_Inteligente_en_Carreteras_3735426_1737660455.pdf

Telegram. (2019). *Bots: An introduction for developers.* Recuperado el 09 de 2019, de https://core.telegram.org/bots

TomTom. (12 de 2017). *El coste de los atascos para las empresas*. Recuperado el 09 de 2019, de TomTom Telematics: https://telematics.tomtom.com/es_es/webfleet/blog/coste-los-atascos-las-empresas/

Twilio-01. (2019). *Twilio API for WhatsApp.* Recuperado el 09 de 2019, de https://www.twilio.com/whatsapp

Twilio-02. (2019). *Programmable SMS.* Recuperado el 09 de 2019, de https://www.twilio.com/sms

Twilio-03. (2019). *Programmable Voice.* Recuperado el 09 de 2019, de https://www.twilio.com/voice

VisualStudio-01. (2019). *Visual Studio Code.* Recuperado el 09 de 2019, de https://code.visualstudio.com/docs

VisualStudio-02. (2019). *Azure IoT Edge Tools.* Recuperado el 09 de 2019, de https://marketplace.visualstudio.com/items?itemName=vsc-iot.vsiotedgetools

VisualStudio-03. (2019). *Azure Functions.* Recuperado el 09 de 2019, de https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions

Watts, S., & Raza, M. (06 de 2019). *SaaS vs PaaS vs IaaS: What's The Difference and How*

To Choose. Recuperado el 09 de 2019, de Multi-Cloud Blog: https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/

Waze. (2019). *Cómo funciona Waze*. Recuperado el 09 de 2019, de https://support.google.com/waze/answer/6078702?hl=es

Yudego, M. M. (10 de 2018). *Apache Spark: Introducción, qué es y cómo funciona*. Recuperado el 09 de 2019, de https://www.icemd.com/digital-knowledge/articulos/apache-spark-introduccion-que-es-y-como-funciona/