

Tópicos avanzados de IoT

Proyecto final:

Prototipo solución IoT Capa de computación en el borde



Damian Martínez
Adda Vargas
Andrés Tamayo
Marco Loaiza

Pontificia Universidad Javeriana
Facultad de Ingeniería
Maestría en ingeniería del Internet de las Cosas
Tópicos Avanzados en IoT
Junio 2021

CONTENIDO

DESCRIPCIÓN GENERAL.....	3
OBJETIVOS	4
REQUISITOS DEL SISTEMA	5
2. Raspberry	8
Funcionalidades.....	11
1. Arquitectura del prototipo desarrollado.....	11
2. Funciones usadas en el proyecto:	11
Desarrollo del proyecto.....	14
Prototipo Simulado	18
Prototipo Materializado	23
Conclusiones	28
Bibliografía	28
Anexos.....	29

TABLA DE ILUSTRACIONES

Figura 1 Máquina virtual Azure	5
Figura 2 Host de Docker. Fuente: Azure.....	7
Figura 3 Servidor Docker. Fuente Azure.....	8
Figura 4 Sistema operativo del host. Fuente: Azure	8
Figura 5 Raspberry Pi model B. Fuente: MercadoLibre.....	9
Figura 6 Ventana instalador del SO raspbian. Fuente Pagina oficial Raspberry	9
Figura 7Arquitectura del proyecto. Fuente: Propia	11
Figura 8 Arquitectura IoT en el borde	12
Figura 9 Comunicación en la nube. Fuente: Azure.....	13
Figura 10 Recursos del proyecto en Azure	14
Figura 11 Captura de pantalla del Recurso en Azure	15
Figura 12 Creación del dispositivo en Azure IoT	15
Figura 13 Contenedor del proyecto en Azure	16
Figura 14 Cadena de conexión del proyecto en Azure.....	16
Figura 15 Archivo de configuración del contenedor.	23
Figura 16 Script de ejecución Python.....	23
Figura 17 configuración de despliegue Json.	23
Figura 18 Prototipo físico del sistema	26

DESCRIPCIÓN GENERAL

Los Smart Parking es uno de los proyectos más llamativos para las ciudades en si búsqueda por ser Smart cities, estos proyectos dan la posibilidad de mejorar la movilidad en las ciudades, permitiendo conocer en tiempo real la disponibilidad de parqueaderos públicos o privados que tiene disponible la ciudad.

Para el desarrollo de estos proyectos los parqueaderos cuentan con dispositivos o tecnología IoT en los cuales por sus requerimientos de energía y funcionalidad de larga duración solo pueden recolectar los datos de los sensores y transmitirlos, sin poder analizar la información. Cargan toda esta información directamente a la nube, donde grandes centros de datos la procesan para obtener ciertas conclusiones o activar ciertos eventos como la apertura de puertas, donde también se debe tener en cuenta el ancho de banda del canal disponible para enviar esta gran cantidad de datos e información. Sin embargo, el **Edge Computing**, ofrece una solución para estas circunstancias donde aporta mucha más autonomía a las soluciones de todos esos dispositivos, permitiendo concentrar la información en un dispositivo con alta capacidad de procesamiento y pueda ejecutar el análisis de estos datos para posteriormente enviar la información más relevante a la nube. En este contexto consiste en analizar y entregar resultados de forma local y no en la nube, permite que los datos producidos por los dispositivos de la internet de las cosas **se procesen más cerca de donde se crearon** en lugar de enviarlos a través de largos recorridos para que lleguen a centros de datos y nubes de computación.

Eso tiene una ventaja fundamental, ya que permite, en este caso a los parqueaderos, analizar los datos importantes casi en tiempo real, lo que permite que los dispositivos y sensores situados por todas partes se ocupen no solo de recolectar esos datos para enviarlos a la nube, sino de procesarlos directamente.

Hay otro término muy relacionado con Edge Computing que está usándose cada vez más en este ámbito, y es el de la llamada Fog Computing, permite extender la nube para que esté más cerca de las cosas que producen y se accionan mediante datos de dispositivos IoT. Hay además factores que harán que este tipo de paradigma lo tenga aún más fácil en el futuro: el coste cada vez más reducido de los dispositivos y los sensores se une a la cada vez mayor potencia que tienen incluso dispositivos modestos.

Sin embargo, a la hora de adquirir un proveedor de este servicio es necesario verificar **la seguridad**. Cuantos menos datos hay en un entorno cloud, menos vulnerable es ese entorno si se ve comprometido. Si la seguridad en esos "microcentros de datos" Edge Computing se cuida de la forma adecuada, este apartado podría ganar muchos enteros.

Azure IoT proporciona la cartera de servicios y características más amplia del sector para tener un servicio de perímetro hasta la nube. El enfoque abierto de Azure IoT pone de manifiesto la facilidad de desarrollo y la integración. Conecte, supervise y controle miles de millones de dispositivos, desde el desarrollo soluciones con gran velocidad y simplicidad en una plataforma de aplicaciones totalmente administrada, o soluciones más flexibles con servicios de plataforma muy sólidos. Azure tiene centros de datos en más regiones del planeta, la cartera de cumplimiento normativo más completa que la de cualquier otro proveedor de nube y un audaz compromiso con la sostenibilidad.

Con esta plataforma se puede ejecutar aplicaciones en contenedores y máquinas virtuales directamente en el perímetro, donde se crean y recopilan los datos, tal y como se explicará en este documento. Adicionalmente no requiere ocupar una gran inversión al comenzar, ya que tendrá la

posibilidad de obtener hasta \$200 gratuitos para realizar las pruebas necesarias para convertirse de una implementación más formal.

OBJETIVOS

Para el desarrollo del proyecto final se plantean los siguientes objetivos.

- Desarrollar un prototipo de solución para Smart Parking basado en la capa de computación en el borde.

Para poder implementar el objetivo principal del proyecto se plantea desarrollar los siguientes objetivos secundarios:

- Implementar el ambiente de desarrollo para la solución Edge computing utilizando la solución que plantea Azure.
- Desarrollar una simulación de la solución generando la información de los sensores a partir de programas.
- Implementar la solución en un sistema de desarrollo IoT (RaspberryPi)

REQUISITOS DEL SISTEMA

Con Azure IoT todo podría simularlo desde el programa, la herramienta es tan poderosa que incluso cuenta con un simulador de raspberry¹, puede crear una máquina virtual con sistemas operativo basado en Linux o Windows donde esta acción no le tomara al usuario más de dos minutos en realizarla. Sin embargo, para este proyecto se usará una maquina basa en Linux, Raspbian para raspberry y se usará Docker para simular el perímetro en el borde. En esta sección se describirán cada una de las partes usadas con una descripción y características fundamentales.

Azure IoT Edge para Linux en Windows permite ejecutar cargas de trabajo de Linux en contenedores junto con aplicaciones de Windows en implementaciones de Windows IoT. Para Linux en Windows funciona ejecutando una máquina virtual Linux en un dispositivo Windows. La máquina virtual Linux viene preinstalada con el tiempo de ejecución de IoT Edge. Todos los módulos de IoT Edge implementados en el dispositivo se ejecutan dentro de la máquina virtual. Mientras tanto, las aplicaciones de Windows que se ejecutan en el dispositivo host de Windows pueden comunicarse con los módulos que se ejecutan en la máquina virtual de Linux.

IoT Edge para Linux en Windows utiliza los siguientes componentes para permitir que las cargas de trabajo de Linux y Windows se ejecuten juntas y se comuniquen sin problemas:

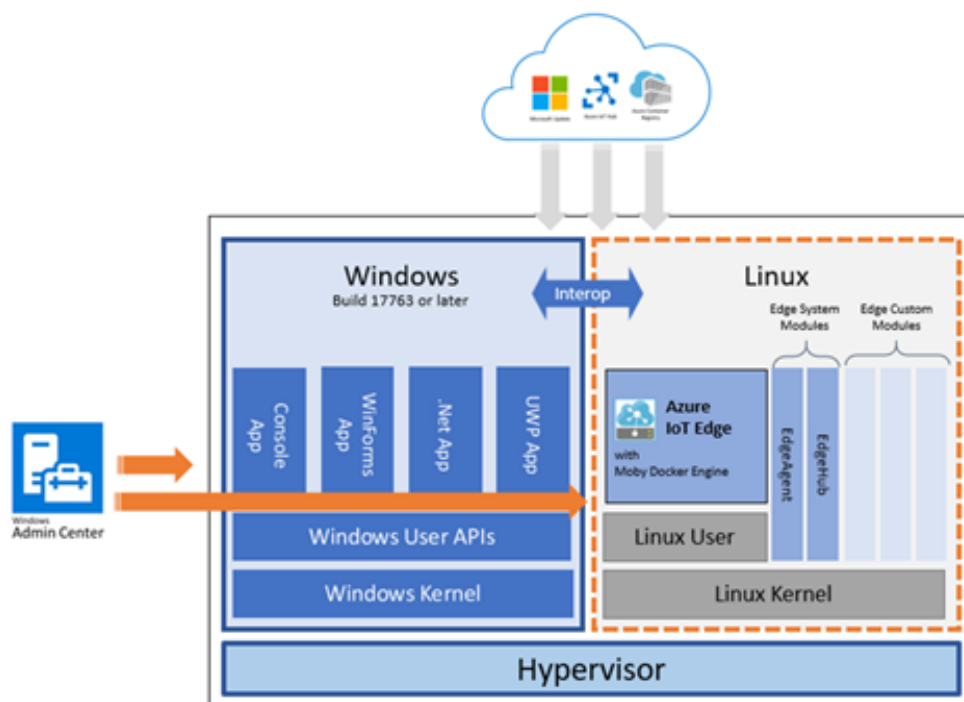


Figura 1 Máquina virtual Azure

- **Una máquina virtual Linux que ejecuta Azure IoT Edge:** una máquina virtual Linux, basada en el sistema operativo Mariner de Microsoft , se crea con el tiempo de ejecución de

¹ <https://azure-samples.github.io/raspberry-pi-web-simulator/#getstarted>

IoT Edge y se valida como un entorno compatible de nivel 1 para cargas de trabajo de IoT Edge.

- **Centro de administración de Windows:** una extensión de IoT Edge para el Centro de administración de Windows facilita la instalación, configuración y diagnóstico de IoT Edge en la máquina virtual de Linux. El Centro de administración de Windows puede implementar IoT Edge para Linux en Windows en el dispositivo local, o puede conectarse a los dispositivos de destino y administrarlos de forma remota.
- **Microsoft Update:** la integración con Microsoft Update mantiene actualizados los componentes de tiempo de ejecución de Windows, la máquina virtual Mariner Linux e IoT Edge.

La comunicación bidireccional entre el proceso de Windows y la máquina virtual de Linux significa que los procesos de Windows pueden proporcionar interfaces de usuario o servidores proxy de hardware para las cargas de trabajo que se ejecutan en los contenedores de Linux.

Del mismo modo que en el sector del transporte se usan contenedores físicos para aislar diferentes cargas (por ejemplo, para el transporte en buques y en trenes), las tecnologías de desarrollo de software usan cada vez más un método denominado **contenerización**, el cual es un paquete de software estándar (conocido como “contenedor”) agrupa el código de una aplicación con las bibliotecas y los archivos de configuración asociados, junto con las dependencias necesarias para que la aplicación se ejecute.

El problema de que una aplicación no se ejecute correctamente cuando se mueve de un entorno a otro es tan antiguo como el propio desarrollo de software. Estos problemas suelen deberse a diferencias de configuración en los requisitos de las bibliotecas subyacentes y otras dependencias. Los contenedores solucionan este problema proporcionando una infraestructura ligera e inmutable para el empaquetado y la implementación de aplicaciones. Una aplicación o un servicio, sus dependencias y su configuración se empaquetan como una imagen de contenedor. La aplicación en contenedor se puede probar como una unidad e implementarse como una instancia de imagen de contenedor en el sistema operativo host.

1. Docker

Docker es una plataforma de creación de contenedores que se usa para desarrollar, distribuir y ejecutar contenedores. La plataforma Docker consta de varios componentes que se usan para desarrollar, ejecutar y administrar nuestras aplicaciones en contenedores.

- **Motor de Docker**

El motor de Docker está formado por varios componentes configurados como una implementación de cliente y servidor en la que ambos se ejecutan simultáneamente en el mismo host. El cliente se comunica con el servidor mediante una API REST, la cual le permite comunicarse también con una instancia del servidor remoto.

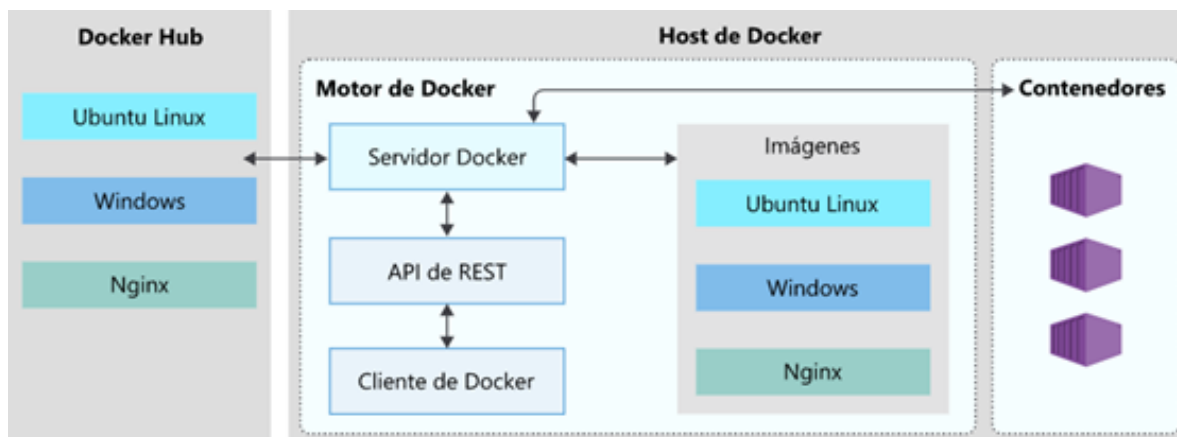


Figura 2 Host de Docker. Fuente: Azure

En el diagrama se muestra un cuadrado que representa Docker Hub con imágenes de contenedor y un cuadrado independiente que representa un host de Docker. Una flecha muestra la comunicación entre Docker Hub y el host de Docker.

El host de Docker contiene dos objetos. Uno representa el motor de Docker y el segundo los contenedores de Docker en ejecución. El objeto de host de Docker contiene cuatro objetos. Son el servidor de Docker, la API de REST de Docker, el cliente de Docker y las imágenes de contenedor almacenadas.

Algunas flechas muestran la comunicación entre el servidor de Docker, la API de REST y el cliente de Docker. Estas flechas indican cómo el usuario se comunica con el servidor de Docker a través de la API de REST.

- **Imagen del contenedor**

Una imagen de contenedor es un paquete portátil que contiene software. Cuando se ejecuta esta imagen, se convierte en nuestro contenedor. El contenedor es la instancia en memoria de una imagen. Las imágenes de contenedor son inmutables. Una vez que se crea una imagen, no se puede cambiar. La única forma de cambiar una imagen es crear una nueva. Esta característica es nuestra garantía de que la imagen que usamos en el equipo de producción es la misma que se usa en los equipos de desarrollo y de control de calidad².

El sistema operativo del host es el sistema operativo en el que se ejecuta el motor de Docker. Los contenedores de Docker que se ejecutan en Linux comparten el kernel del sistema operativo del host y no requieren un sistema operativo de contenedor, siempre que el archivo binario pueda acceder directamente al kernel del sistema operativo.

² <https://docs.microsoft.com/es-es/learn/modules/intro-to-docker-containers/3-how-docker-images-work>

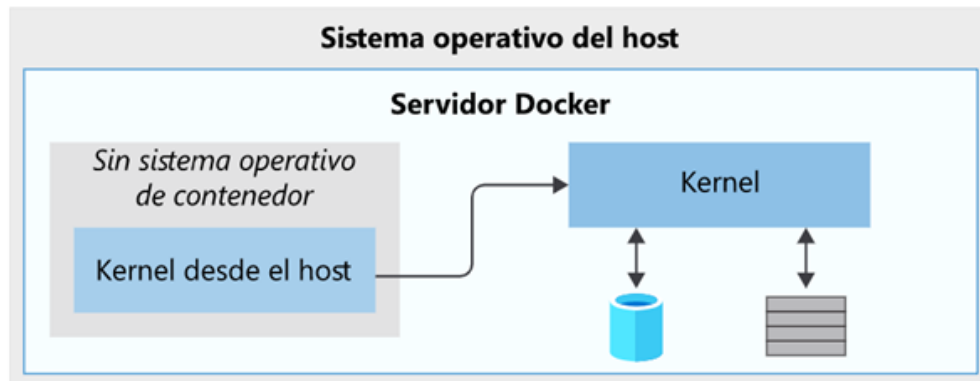


Figura 3 Servidor Docker. Fuente Azure

Pero los contenedores de Windows necesitan un sistema operativo de contenedor. El sistema operativo del contenedor es el sistema operativo que forma parte de la imagen empaquetada. Tenemos flexibilidad para incluir diferentes versiones de los sistemas operativos de Linux o Windows en un contenedor. Esta flexibilidad nos permite tener acceso a determinadas características del sistema operativo o instalar software adicional que pueden usar nuestras aplicaciones.

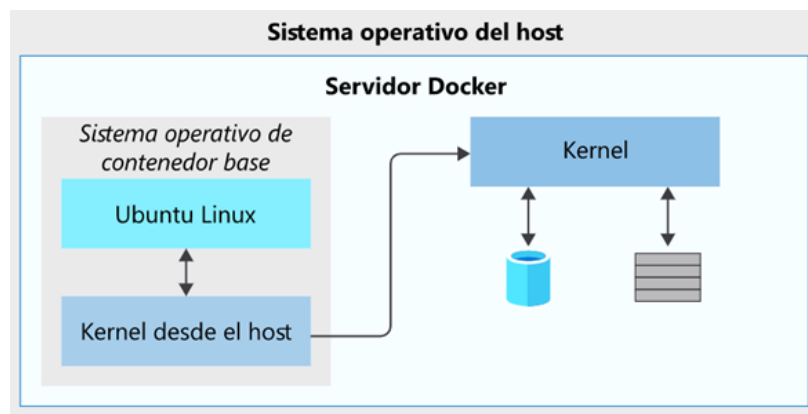


Figura 4 Sistema operativo del host. Fuente: Azure

El sistema operativo del contenedor está aislado del sistema operativo del host y es el entorno en el que se implementa y ejecuta la aplicación. Si se combina con la inmutabilidad de la imagen, este aislamiento implica que el entorno de la aplicación que se ejecuta en el departamento de desarrollo es el mismo que el del departamento de producción. En nuestro ejemplo, usamos Ubuntu Linux como sistema operativo del contenedor y este sistema operativo no cambia en el proceso de desarrollo ni en el de producción. La imagen que usamos es siempre la misma.

2. Raspberry

Raspberry Pi es un mini ordenador de pequeñas dimensiones y precio destinado principalmente al desarrollo de pequeños prototipos y a estimular la enseñanza de las ciencias de la computación en los centros educativos. Desarrollado en hardware libre cuenta con sistemas operativos GNU/Linux como Raspbian aunque podemos encontrar otros sistemas operativos optimizados para el hardware de la Raspberry Pi.

Raspberry Pi utiliza una arquitectura para el procesador ARM distinta a la que estamos acostumbrados a utilizar en nuestros ordenadores de sobremesa o portátiles. Esta arquitectura es de

tipo RISC (Reduced Instruction Set Computer), es decir, utiliza un sistema de instrucciones realmente simple lo que le permite ejecutar tareas con un mínimo consumo de energía.



Figura 5 Raspberry Pi model B. Fuente: MercadoLibre

Raspbian es el sistema operativo recomendado para Raspberry Pi (al estar optimizado para su hardware) y se basa en una distribución de GNU/Linux llamada Debian. Existen numerosos métodos para instalar Raspbian o cualquier otro sistema operativo en la Raspberry Pi. No obstante, el método más sencillo que conozco es usar Raspberry Pi Imager, un software multiplataforma que de forma totalmente automática descargará e instalará la imagen de Raspbian a la tarjeta SD.

- **Material mínimo necesario para instalar Raspbian**

Una tarjeta micro SD debidamente formateada con el sistema de archivos FAT32. Recuerden que la tarjeta micro SD tiene que tener un almacenamiento de al menos 16GB.

Un adaptador de tarjetas de Micro SD a SD. De esta forma podré conectar la tarjeta al lector de tarjetas SD de mi ordenador.

- **Instalar Raspberry Pi Imager**

Se puede instalar Raspberry Pi Imager en Windows, Linux y MacOS. Para instalarlo tan solo tenemos que acceder a la siguiente página web. Una vez dentro descargamos el archivo binario de instalación para nuestro sistema operativo.



Figura 6 Ventana instalador del SO raspbian. Fuente Pagina oficial Raspberry

Seguidamente clicamos sobre el botón CHOSSE OS y seleccionamos el sistema operativo que queremos instalar. En mi caso selecciono la opción recomendada clicando sobre la opción Raspbian.

A continuación, clicamos sobre el botón CHOSSE SD CARD y seleccionamos la tarjeta Micro SD en que queremos instalar Raspbian. Finalmente, tan solo tenemos que clicar sobre el botón WRITE. Acto seguido, de forma totalmente automática se descargará e instalará el sistema operativo seleccionado en nuestra tarjeta Micro SD. Una vez finalizado el proceso de descarga e instalación obtendremos el siguiente mensaje: “Instalación de Raspbian finalizada”.

FUNCIONALIDADES

El dispositivo estará en la capacidad de

1. Realizar la captura de datos del sensor desde el dispositivo el cual indica el estado del parqueadero.
2. Filtra la información de eventos donde solo se toman los valores cuando el carro ingresa o sale del parqueadero.
3. Envío de información del sensor a la nube

1. Arquitectura del prototipo desarrollado

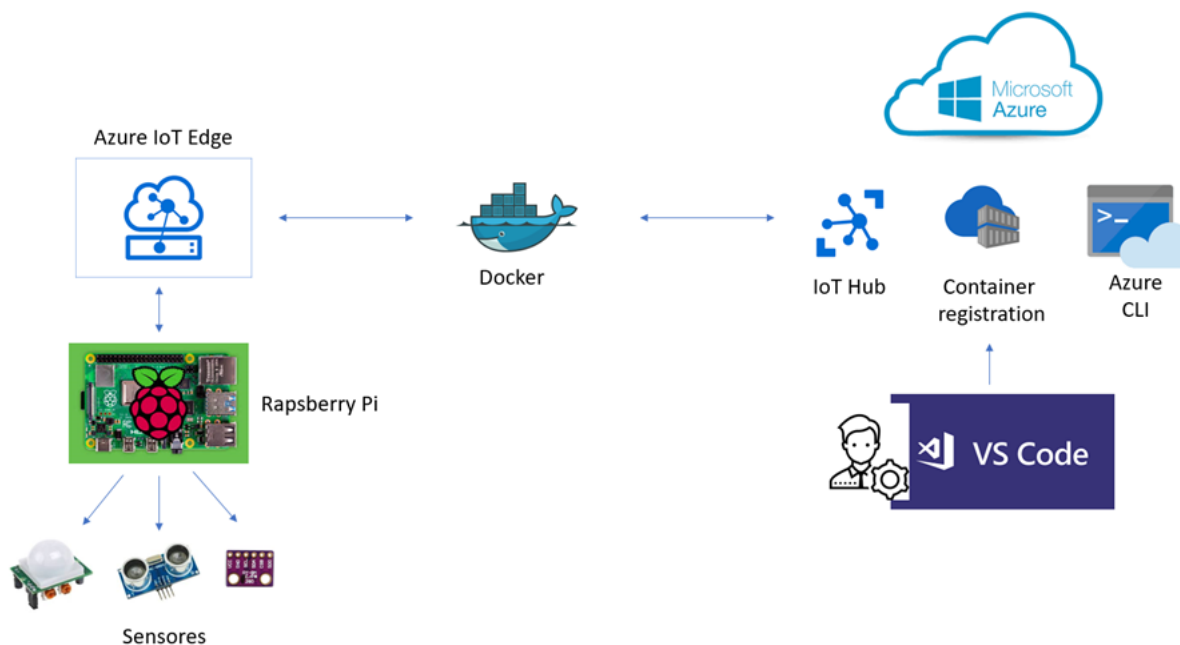


Figura 7Arquitectura del proyecto. Fuente: Propia

2. Funciones usadas en el proyecto:

IoT Hub

IoT Hub es un servicio administrado alojado en la nube que actúa como un centro de mensajes central para las comunicaciones en ambas direcciones entre una aplicación de IoT y sus dispositivos conectados. Puede conectar millones de dispositivos y sus soluciones de backend de manera confiable y segura. Casi cualquier dispositivo se puede conectar a un IoT Hub.

Se admiten varios patrones de mensajería, incluida la telemetría de dispositivo a la nube, la carga de archivos desde los dispositivos y los métodos de solicitud y respuesta para controlar sus dispositivos desde la nube. IoT Hub también admite el monitoreo para ayudarlo a rastrear la creación de dispositivos, la conexión de dispositivos y las fallas de dispositivos.

Con las capacidades de IoT Hub, puede crear soluciones de IoT escalables y con todas las funciones, como la administración de equipos industriales utilizados en la fabricación, el seguimiento de activos valiosos en el cuidado de la salud y el monitoreo del uso de edificios de oficinas.

Para el desarrollo proyecto el iot hub se encarga de gestionar todos los dispositivos.

Agente de IoT edge Este es un servicio que nos permite desplegar diferentes cargas de trabajo en los diferentes dispositivos. Permite desplegar contenedores Docker en diferentes máquinas que previamente hemos configurado con el runtime de IoT Edge.

En el proyecto se realizará la creación de un docker el cual contiene el sistema operativo arm23v7 y el código de la aplicación desarrollado en python.

Run Time Edge

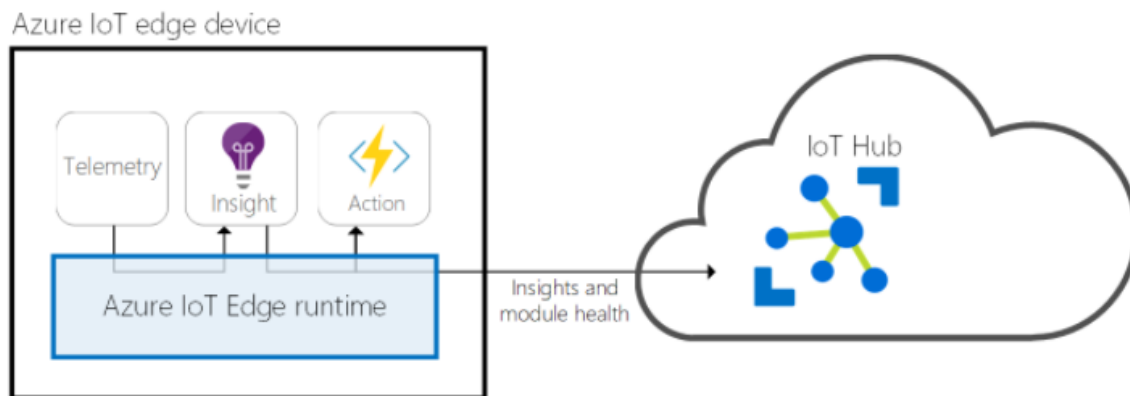


Figura 8 Arquitectura IoT en el borde

El agente de IoT Edge es uno de los dos módulos que componen el entorno de ejecución de Azure IoT Edge. Es responsable de crear instancias de los módulos, lo que garantiza que continúen ejecutándose y notificando el estado de los módulos a IoT Hub. Estos datos de configuración se escriben como una propiedad del módulo gemelo del agente de IoT Edge.

El demonio de seguridad de IoT Edge inicia el agente de IoT Edge durante el inicio del dispositivo. El agente recupera su módulo gemelo de IoT Hub e inspecciona el manifiesto de implementación. El manifiesto de implementación es un archivo JSON que declara al módulo que debe iniciarse.

Cada elemento del manifiesto de implementación contiene información específica de un módulo y el agente de IoT Edge lo usa para controlar el ciclo de vida del módulo. Para obtener más información acerca de todas las propiedades que usa el agente de IoT Edge para controlar los módulos, lea acerca de las propiedades del agente de IoT Edge y los módulos gemelos del centro de IoT Edge.

El agente de IoT Edge envía la respuesta de entorno de ejecución a IoT Hub. A continuación, se ofrece una lista de las posibles respuestas:

- 200 - CORRECTO
- 400 - La configuración de implementación tiene un formato incorrecto o no es válida.
- 417 - El dispositivo no tiene un conjunto de configuración de implementación.

- 412 - La versión del esquema de la configuración de implementación no es válida.
- 406 - El dispositivo de IoT Edge está sin conexión o no envía informes de estado.
- 500 - Error en el entorno en tiempo de ejecución de Azure IoT Edge.

En el desarrollo del proyecto de este módulo se usó para poder realizar la conexión desde la Raspberry instalando el agente en este y poder comunicar el código de la aplicación con la nube de Azure.

Comunicación en la nube

Para reducir el ancho de banda que usa la solución IoT Edge, el centro de IoT Edge optimiza el número real de conexiones a la nube. El centro de IoT Edge toma las conexiones lógicas de módulos o dispositivos de bajada y las combina para crear una sola conexión física a la nube. Los detalles de este proceso son transparentes para el resto de la solución. Los clientes creen que tienen su propia conexión a la nube, aunque todos los datos van a enviarse a través de la misma. El centro de IoT Edge puede usar el protocolo AMQP o MQTT para comunicarse con la nube, independientemente de los protocolos que usen los dispositivos de nivel inferior. No obstante, el centro de IoT Edge actualmente solo admite la combinación de conexiones lógicas en una única conexión física mediante el uso de AMQP como protocolo de nivel superior y sus capacidades de multiplexación. AMQP es el protocolo de nivel superior predeterminado.

Para esta aplicación se usó el protocolo de comunicación mqtt.

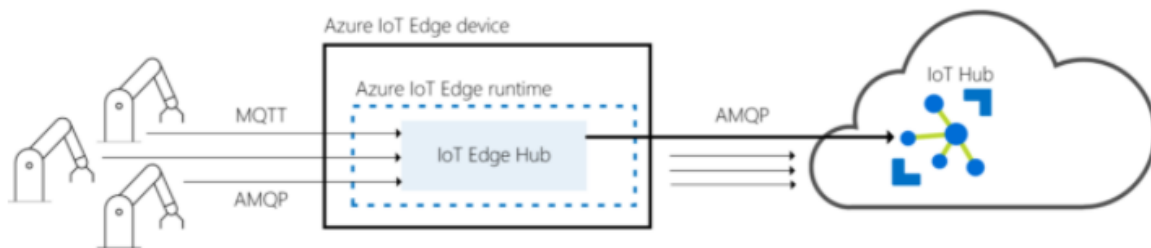


Figura 9 Comunicación en la nube. Fuente: Azure

Extensión de IoT para Azure Cli

Es una extensión de IoT de código abierto que se suma a las capacidades de la CLI de Azure. La extensión de IoT brinda a los desarrolladores de IoT acceso de línea de comandos a todas las capacidades del servicio de aprovisionamiento de dispositivos de IoT Hub, IoT Edge y IoT Hub.

Para la aplicación se usó la interfaz de comandos de Azure (Azure Cli) para realizar la ejecución de las líneas de configuración de las máquinas, almacenamiento, sensor y Docker durante los pasos del desarrollo del proyecto.

DESARROLLO DEL PROYECTO

Implementación del ambiente de desarrollo Azure Iot Edge

A continuación, se muestran los pasos para crear las conexiones de los dispositivos Edge a la plataforma central de Iot Hub en Azure.

Para el desarrollo del proyecto en la cuenta de azure se debe crear:

- Grupo de recursos del proyecto
- Cloud storage
- Centro de Azure IoT (IoTHub)
- Dispositivo que ejecuta Azure IoT Edge
- Registros de contenedores

Azure permite realizar la configuración desde la interfaz gráfica de usuario, o por la interfaz de línea de comandos utilizando el AzureCLI, por practicidad y facilidad de configuración utilizamos la opción de los comandos para crear y configurar los requerimientos.

Al inicializar el CLI de Azure este crea automáticamente un centro de recursos para almacenar la información de los comandos ejecutados en un Cloud-shell-Storage.

El grupo de recursos del proyecto permite administrar y almacenar todos los recursos que se utilizan en el proyecto. Para crearlo se utiliza el siguiente comando

```
- az group create --name IoTEdgeResources --location eastus2
```

De esta manera se crea en los datacenter de Microsoft ubicados al este de Estados Unidos el grupo de recursos IoTEdgeResources.

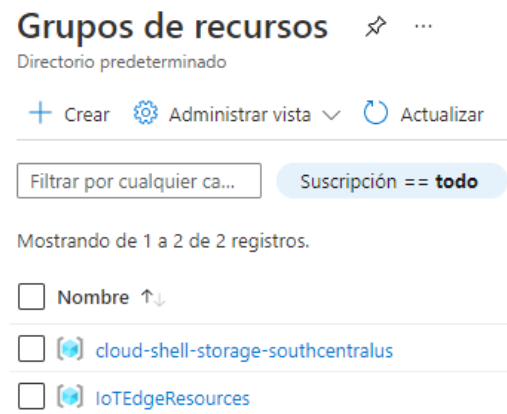


Figura 10 Recursos del proyecto en Azure

Continuando con la implementación se crea el centro IoT Hub junto con el dispositivo IoT Edge con los siguientes comandos.

Los parámetros son el nombre del grupo de recursos, el nombre del IoT Hub, la opción de nivel precios donde F1 corresponde a la versión gratuita, y el número de partición el cual es un registro de confirmación cuando se recibe un evento donde se recomienda por parte de azure que por lo menos se tengan 2 particiones.³

```
- az iot hub create --resource-group IoTEdgeResources --name HubParking --sku F1 --partition-count 2
```



Figura 11 Captura de pantalla del Recurso en Azure

Para la creación del dispositivo Edge se agrega como parámetros el nombre del dispositivo y el nombre del IoT Hub.

```
- az iot hub device-identity create --device-id ParkingEdge --edge-enabled --hub-name HubParking
```

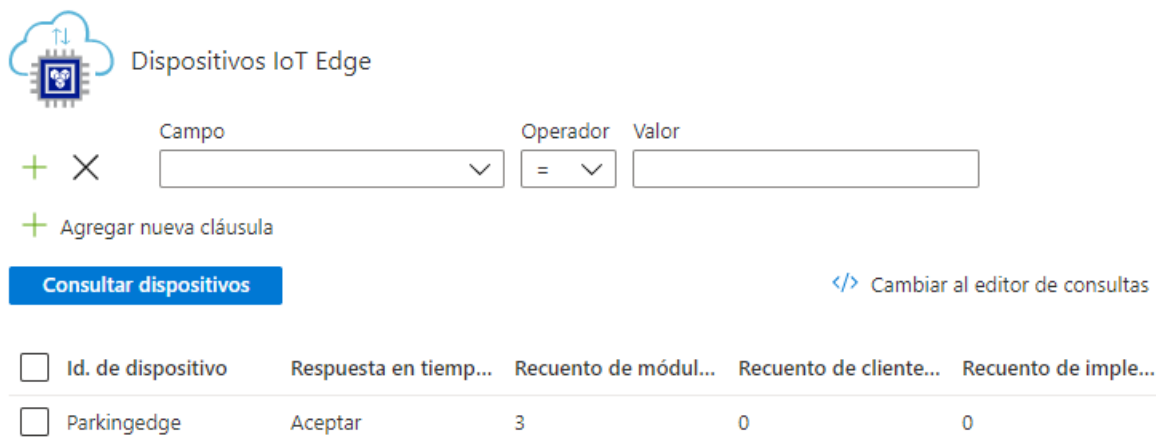


Figura 12 Creación del dispositivo en Azure IoT

Por último, se crea el registro de contenedores donde se almacenarán los contenedores de las aplicaciones que se envíen al dispositivo Edge

³ <https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-features>

- `az acr create --resource-group IoTEdgeResources --name RPicontainer --sku F1`

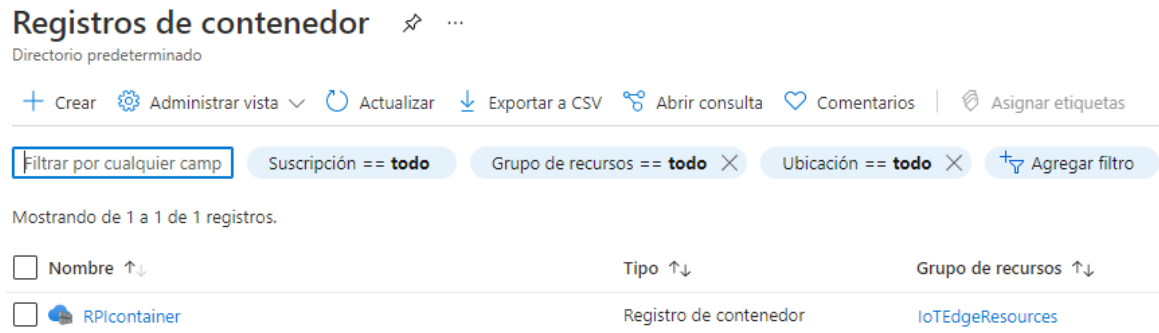


Figura 13 Contenedor del proyecto en Azure

Para la simulación se utiliza una máquina virtual que trabaja como IoT Edge Device, donde los datos se generan con funciones de Python. Posteriormente se realiza el montaje de la solución en una Raspberry Pi y un sensor de proximidad el cual emulara la detección de entrada y salida de un auto al parqueadero.

Para relacionar el dispositivo ya sea la maquina virtual o la Raspberry Pi con el IoT Hub se utiliza una cadena de conexión la cual se genera con el siguiente comando

- `az iot hub device-identity connection-string show --device-id Parkingedg --hub-name HubParking`

o se puede conseguir desde la interfaz gráfica en el espacio de cadena de conexión principal como se muestra a continuación.

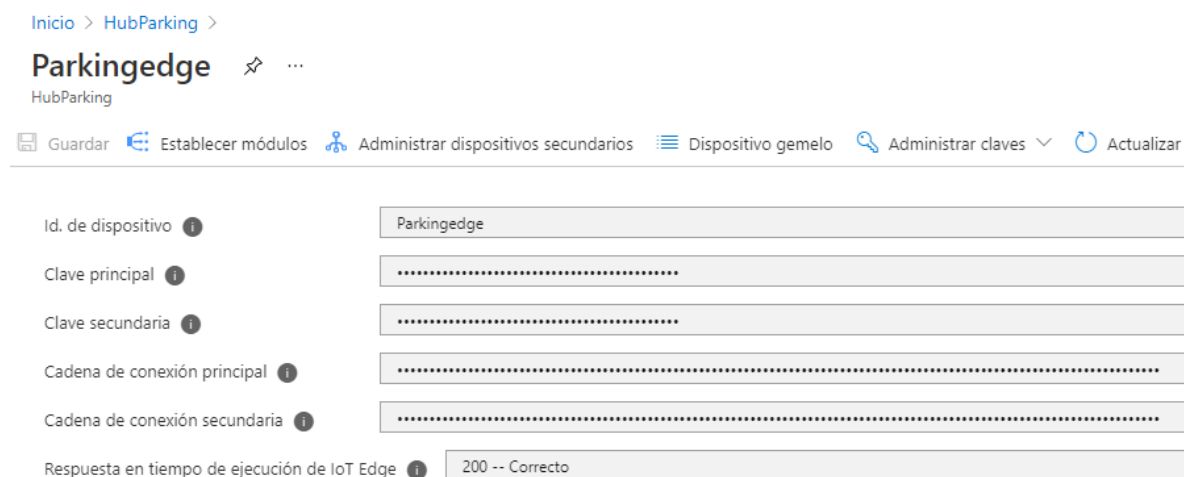


Figura 14 Cadena de conexión del proyecto en Azure

Teniendo ya configurados estos parámetros se procede a realizar la creación de la solución.

Cuando se enlaza el dispositivo Edge con el IoTHub se despliega el Azure IoT Edge runtime el cual es una colección de aplicaciones que transforman un dispositivo en un IoT Edge device.

Estos componentes permiten al dispositivo Edge recibir código utilizando contenedores para correr en el borde y comunicar los resultados al IoT Hub, estos contenedores o aplicaciones Azure las denomina módulos.

Inicialmente el Edge runtime contiene dos componentes principales que se encarga un módulo para la comunicación, y el otro modulo que se encarga de desplegar y monitorear las demás aplicaciones o módulos que se envían al dispositivo, estos son el IoT Edge Hub y el IoT Edge agent respectivamente.

Para la creación del módulo de aplicación Azure presenta algunos tutoriales para generar los contenedores y cargarlos al IoT Edge device utilizando Visual Studio.

Para esto, se debe configurar un archivo Json con la información de los módulos que se desplegaran sobre el dispositivo, un archivo Docker con la información del ambiente de aplicación del contenedor y el archivo de Python el cual ejecuta el programa del dispositivo.

A continuación, se procede a ejecutar dos soluciones, una solución completamente simulada utilizando como dispositivo de borde una máquina virtual y la aplicación de un módulo que simula todos los sensores y la otra con un dispositivo raspberry pi y el uso de un sensor para modelar el funcionamiento de identificación de un carro y posterior envío de la información a la nube.

PROTOTIPO SIMULADO

Para el prototipo simulado se creó una máquina virtual Linux configurada para ser un dispositivo IoT Edge, en el cual

Para empezar lo primero es crear un grupo de recursos para administrar todos los recursos que se van a usar en este proyecto.

- Crear un centro de IoT Hub.

El siguiente paso es crear un centro de IoT Hub , el siguiente código crea un centro F1 gratis en el grupo de recursos IoTEdgeResources con el nombre smartPark.

```
- az iot hub create --resource-group IoTEdgeResources --name smartPark --sku F1 --partition-count 2
```

- Registre un dispositivo IoT Edge en la instancia de IoT Hub.

Se crea una identidad para el dispositivo de IoT Edge, con el fin que se pueda comunicar con el IoT Hub. Se usa una cadena de conexión de dispositivo única para asociar un dispositivo físico a una identidad de dispositivo, con el siguiente comando vamos a crear un dispositivo llamado myEdgeDevice.

```
- az iot hub device-identity create --device-id myEdgeDevice --edge-enabled --hub-name smartPark
```

La siguiente es la cadena de conexión del dispositivo, que vincula el dispositivo físico con la identidad en IoT Hub. Contiene el nombre del centro de IoT, el nombre del dispositivo y, después, una clave compartida que autentica las conexiones entre los dos

Cadena de conexión:

```
damian@Azure:~$ az iot hub device-identity connection-string show --device-id myEdgeDevice --hub-name smartPark
{
  "connectionString": "HostName=smartPark.azure-devices.net;DeviceId=myEdgeDevice;SharedAccessKey=Ib5kAo3g838Q/428uE4ohch3SDiEY0WnGti+ZVZx1Ko="
}
```

- Configuración de un dispositivo de IoT Edge

Para el prototipo simulado se creó una máquina virtual con el runtime de Azure IoT Edge el cual consta de tres componentes el demonio de seguridad de IoT Edge se inicia cada vez que se inicia un dispositivo IoT Edge y arranca el dispositivo mediante el inicio del agente de este. El agente de IoT Edge facilita la implementación y supervisión de los módulos en el dispositivo IoT Edge, incluido el

centro de IoT Edge. El centro de IoT Edge administra las comunicaciones entre los módulos del dispositivo de IoT Edge y entre el dispositivo y la instancia de IoT Hub.

Se usará la plantilla de Azure Resource Manager para crear la máquina virtual con el siguiente comando.

```
az deployment group create \
--resource-group IoTEdgeResources \
--template-uri "https://aka.ms/iotedge-vm-deploy" \
--parameters dnsLabelPrefix='vmpark' \
--parameters adminUsername='azureUser' \
--parameters deviceConnectionString=$(az iot hub device-identity
connection-string show --device-id myEdgeDevice --hub-name smartPark -
o tsv) \
--parameters authenticationType='password' \
--parameters adminPasswordOrKey="p@rking"
```

^ Información esencial

Grupo de recursos [\(cambiar\)](#)
IoTEdgeResources

Estado
En ejecución

Ubicación
Oeste de EE. UU. 2

Suscripción [\(cambiar\)](#)
Azure subscription 1

Id. de suscripción
db6dc2e0-b7a4-4063-a8a0-76c8d598a0d6

Etiquetas [\(cambiar\)](#)
[Haga clic aquí para agregar etiquetas.](#)

Sistema operativo
Linux

Tamaño
Estándar DS1 v2 (1 vcpu, 3.5 GiB de memoria)

Dirección IP pública
52.156.71.96

Red virtual/subred
vnet-omiwnrefrmdj4/subnet-omiwnrefrmdj4

Nombre DNS
vmpark.westus2.cloudapp.azure.com

Propiedades

Supervisión

Funcionalidades (7)

Recomendaciones

Tutoriales



Máquina virtual

Nombre del equipo	vmpark
Sistema operativo	Linux
Publicador	Canonical
Oferta	UbuntuServer
Plan	18.04-LTS
Generación de VM	V1
Grupo host	Ninguno
Host	-
Grupo con ubicación por proximidad	
Estado de ubicación	N/D



Disponibilidad y escalado



Redes

Dirección IP pública	ip-vmpark
Dirección IP pública (IPv6)	-
Dirección IP privada	10.0.0.4
Dirección IP privada (IPv6)	-
Red virtual/subred	vnet-omiwnrefrmdj4,
Nombre DNS	vmpark.westus2.clou



Tamaño

Tamaño	Estándar DS1 v2
vCPU	1
RAM	3.5 GiB

- Visualización del estado del entorno de ejecución de Azure IoT Edge

Creada la máquina se realiza la conexión

```

damian@Azure:~$ ssh azureUser@vmpark.westus2.cloudapp.azure.com
Warning: Permanently added the ECDSA host key for IP address '52.156.71.96' to the list of known hosts.
azureUser@vmpark.westus2.cloudapp.azure.com's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1047-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Wed Jun  2 15:59:43 UTC 2021

System load:  0.32           Users logged in:           0
Usage of /:   11.7% of 28.90GB IP address for eth0:       10.0.0.4
Memory usage: 17%           IP address for br-0e304575d8db: 172.18.0.1
Swap usage:   0%            IP address for docker0:    172.17.0.1
Processes:   138

```

A continuación se valida que el runtime se ha instalado y configurado correctamente en el dispositivo.

```
sudo iotedge list
```

```

azureUser@vmpark:~$ sudo iotedge list
NAME                STATUS      DESCRIPTION          CONFIG
edgeAgent           running     Up 2 hours           mcr.microsoft.com/azureiotedge-agent:1.1
edgeHub             running     Up 2 hours           mcr.microsoft.com/azureiotedge-hub:1.1
azureUser@vmpark:~$

```

- Implemente de módulo en el dispositivo IoT Edge.

Una de las funcionalidades clave de Azure IoT Edge es la implementación de código en dispositivos de IoT Edge desde la nube. Los módulos de IoT Edge son paquetes ejecutables que se implementan como contenedores. El módulo que se implementa simula un sensor y envía los datos generados.

Establecer módulos en el dispositivo: myEdgeDevice ...

smartPark

Módulos [Rutas](#) [Revisar y crear](#)

Credenciales de Container Registry

Puede especificar las credenciales en los registros de contenedor que hospedan las imágenes de módulo. Las credenciales especificadas se usan para recuperar los módulos con una dirección URL correspondiente. El agente de Edge devolverá el código de error 500 si no se encuentra una configuración de registro de contenedor para un módulo.

NOMBRE	DIRECCIÓN	NOMBRE DE USUARIO	CONTRASEÑA
contenedoredgeparki	contenedoredgeparking.azurecr.io	ContenedorEdgePa...
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="password"/>
Nombre	Dirección	Nombre de usuario	Contraseña

Módulos de IoT Edge

Un módulo IoT Edge es un contenedor de Docker que puede implementar en dispositivos IoT Edge. Se comunica con otros módulos y envía datos al entorno en tiempo de ejecución de IoT Edge. Mediante esta interfaz de usuario, puede importar módulos del servicio de Azure IoT Edge o especificar la configuración para un módulo IoT Edge. La configuración de módulos en cada dispositivo se aplicará al recuento de la cuota y se limitará en función del nivel y las unidades de IoT Hub. Por ejemplo, para el nivel S1, pueden establecerse módulos diez veces por segundo si no se están produciendo otras actualizaciones en la instancia de IoT Hub. [Más información](#)

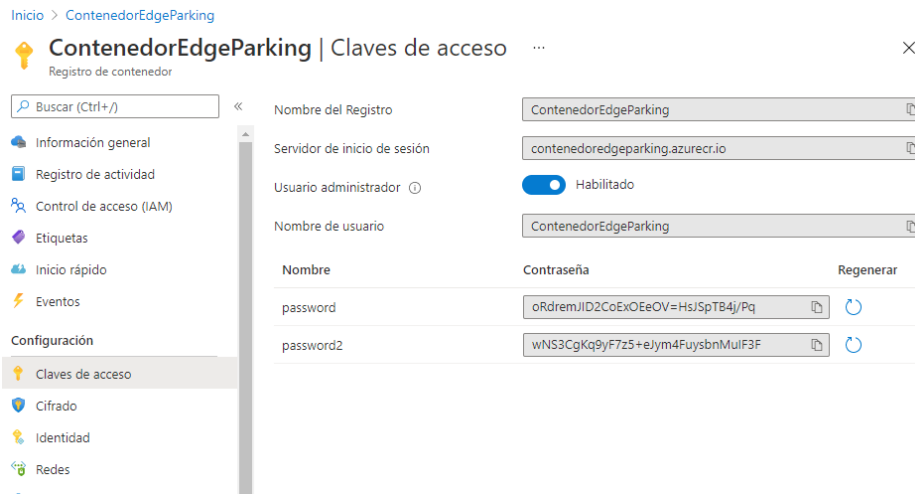
[+](#) Agregar [⚙](#) Configuración del entorno de ejecución

NOMBRE	ESTADO DESEADO
SimulatedTemperatureSensor	running

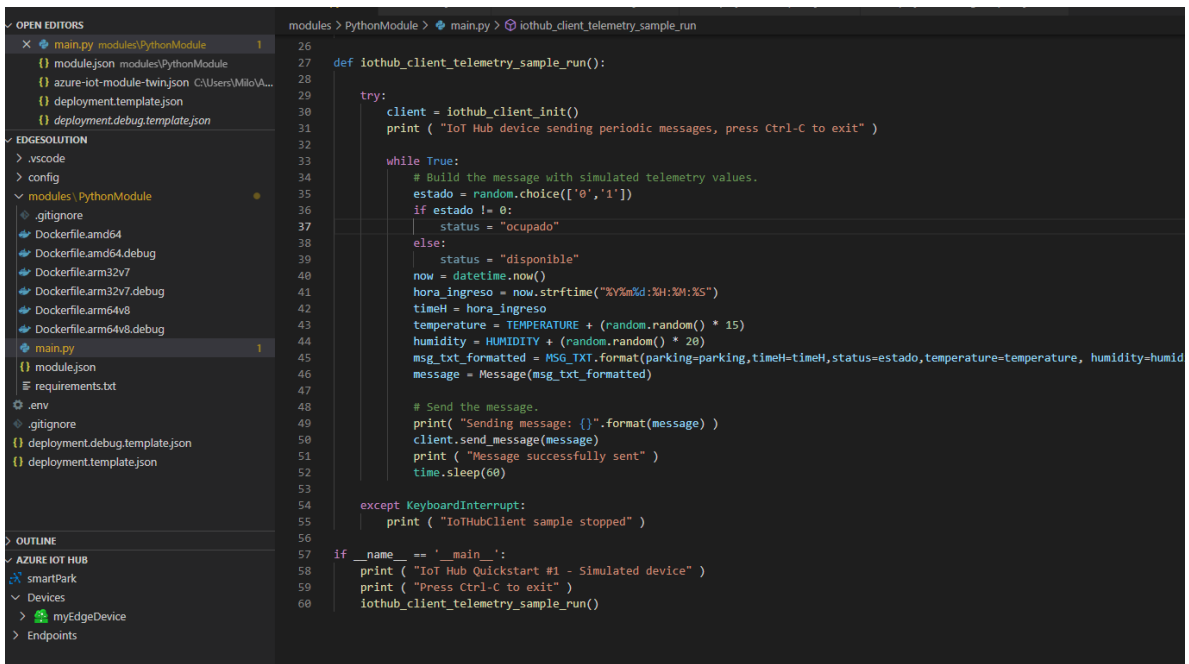
En la máquina virtual se pueden revisar los mensajes que genera el sensor de temperatura

```
[Information]: Successfully initialized module client of transport type [Amqp_Tcp_Only].
06/02/2021 15:58:00> Sending message: 1, Body: [{"machine":{"temperature":21.05979735698076,"pressure":1.0068
06/02/2021 15:58:05> Sending message: 2, Body: [{"machine":{"temperature":21.619437831742427,"pressure":1.070
06/02/2021 15:58:10> Sending message: 3, Body: [{"machine":{"temperature":22.307550721130124,"pressure":1.148
06/02/2021 15:58:15> Sending message: 4, Body: [{"machine":{"temperature":23.111454557679338,"pressure":1.240
06/02/2021 15:58:20> Sending message: 5, Body: [{"machine":{"temperature":24.00396721286418,"pressure":1.3422
06/02/2021 15:58:25> Sending message: 6, Body: [{"machine":{"temperature":24.52926808364189,"pressure":1.4020
```

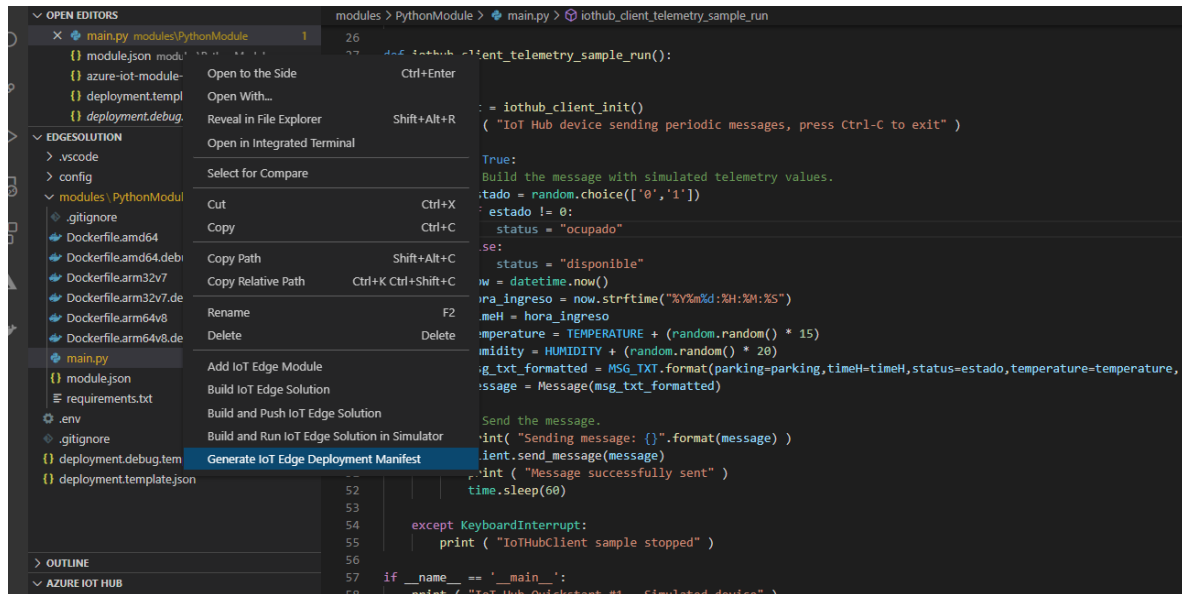
Ahora implementamos nuestro propio código en el dispositivo IoT Edge para lo cual se requiere de un Registro de Contenedores en la cual se insertaran la imagen de los archivos para su almacenamiento y su ejecución en el dispositivo IoT Edge.



Se crea un nuevo modulo tomando como referencia la plantilla proporciona por Azure IoT Edge: New IoT Edge solution, seleccionando la arquitectura de inicio en nuestro caso simulado amd64 y modificando el archivo main en la carpeta modules/PythonModule con nuestra implementación de los sensores.

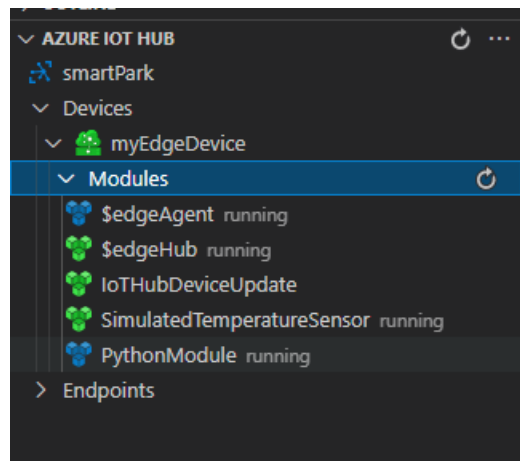


Ahora para compilar e insertar el código de la solución en una imagen de contenedor se realiza dando click derecho en el archivo deployment.template.json y selecciona Build and Push IoT Edge solution.

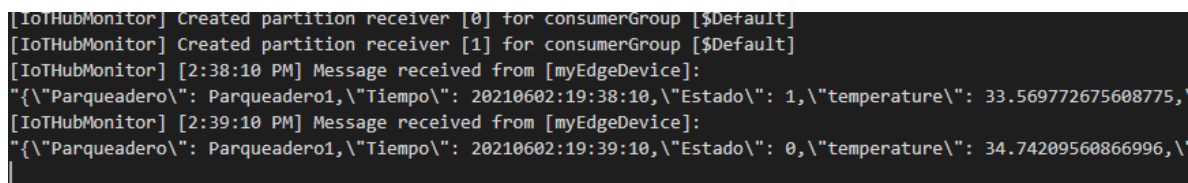


Creado el contenedor solo resta implementarla en el dispositivo Edge en el cual se selecciona Create Deployment for IoT Edge device (Crear una implementación para un dispositivo individual) y en la carpeta config y selecciona el archivo deployment.amd64.json.

Finalmente, al actualizar el nuevo módulo iniciara su ejecución en el dispositivo Edge.



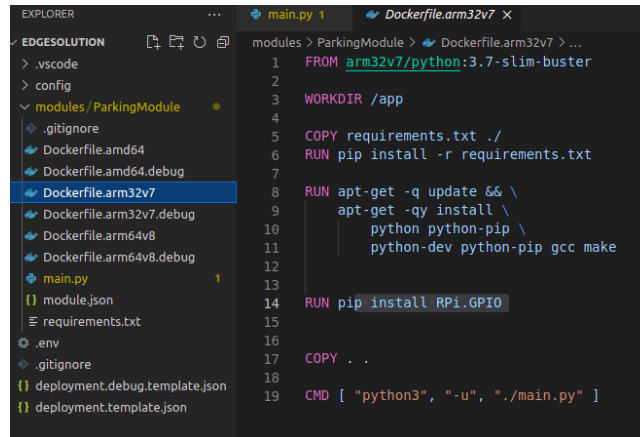
El código PythonModule se encarga de simular los diferentes sensores y contener la lógica referente al estado del sitio de parqueo según la información generada, como se observa en la siguiente imagen.



PROTOTIPO MATERIALIZADO

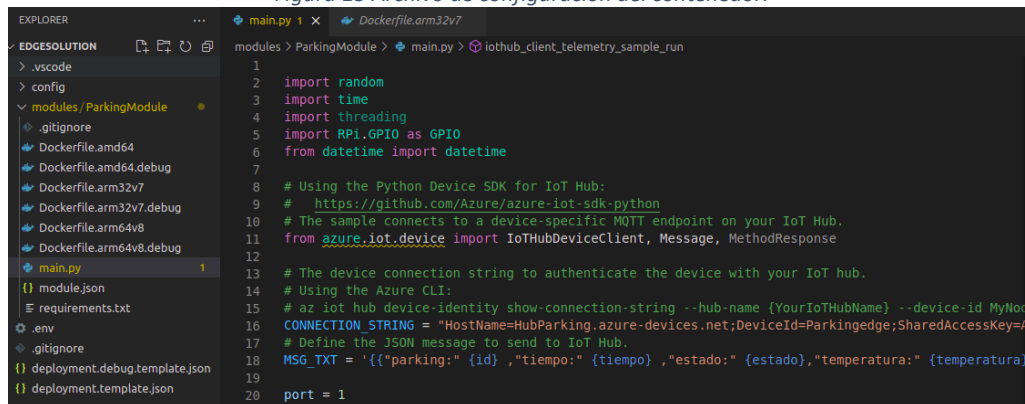
En el segundo caso se implementa la solución del borde en una raspberry Pi la cual recibe información de un sensor de proximidad, el cual emula el comportamiento del parqueadero cuando un carro es detectado entrando o saliendo de una plaza de parqueo.

Para esto se procede a realizar la configuración de los tres archivos, código Python, archivo de configuración Docker, archivo Json de despliegue de solución.



```
1 FROM arm32v7/python:3.7-slim-buster
2
3 WORKDIR /app
4
5 COPY requirements.txt ./
6 RUN pip install -r requirements.txt
7
8 RUN apt-get -q update && \
9     apt-get -qy install \
10         python python-pip \
11         python-dev python-pip gcc make
12
13
14 RUN pip install RPi.GPIO
15
16
17 COPY . .
18
19 CMD [ "python3", "-u", "./main.py" ]
```

Figura 15 Archivo de configuración del contenedor.



```
1 import random
2 import time
3 import threading
4 import RPi.GPIO as GPIO
5 from datetime import datetime
6
7
8 # Using the Python Device SDK for IoT Hub:
9 # https://github.com/Azure/azure-iot-sdk-python
10 # The sample connects to a device-specific MQTT endpoint on your IoT Hub.
11 from azure.iot.device import IoTHubDeviceClient, Message, MethodResponse
12
13 # The device connection string to authenticate the device with your IoT hub.
14 # Using the Azure CLI:
15 # az iot hub device-identity show-connection-string --hub-name {YourIoTHubName} --device-id MyNode
16 CONNECTION_STRING = "HostName=HubParking.azure-devices.net;DeviceId=Parkingedge;SharedAccessKey=A"
17 # Define the JSON message to send to IoT Hub.
18 MSG_TXT = '{"parking": { "id": "1", "tiempo": "1", "estado": "estado", "temperatura": "temperatura" } }'
19
20 port = 1
```

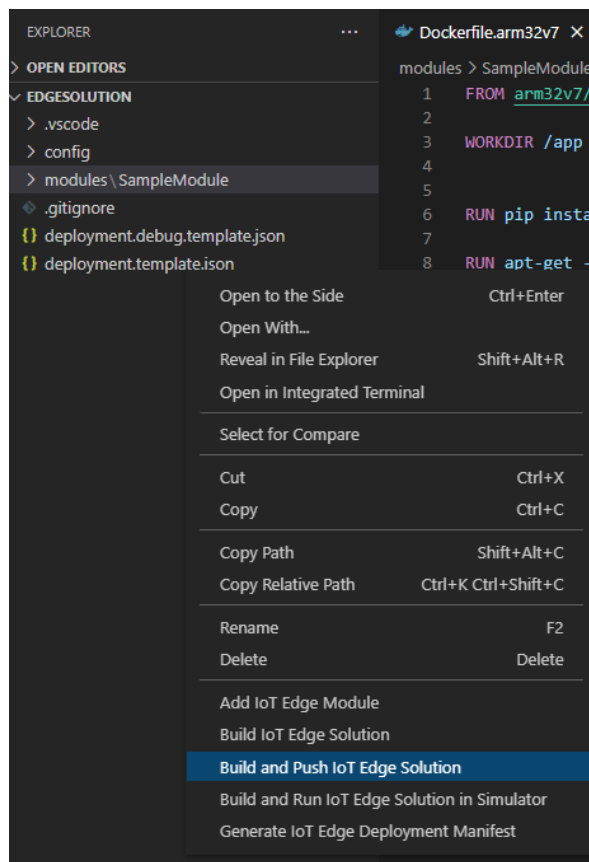
Figura 16 Script de ejecución Python.



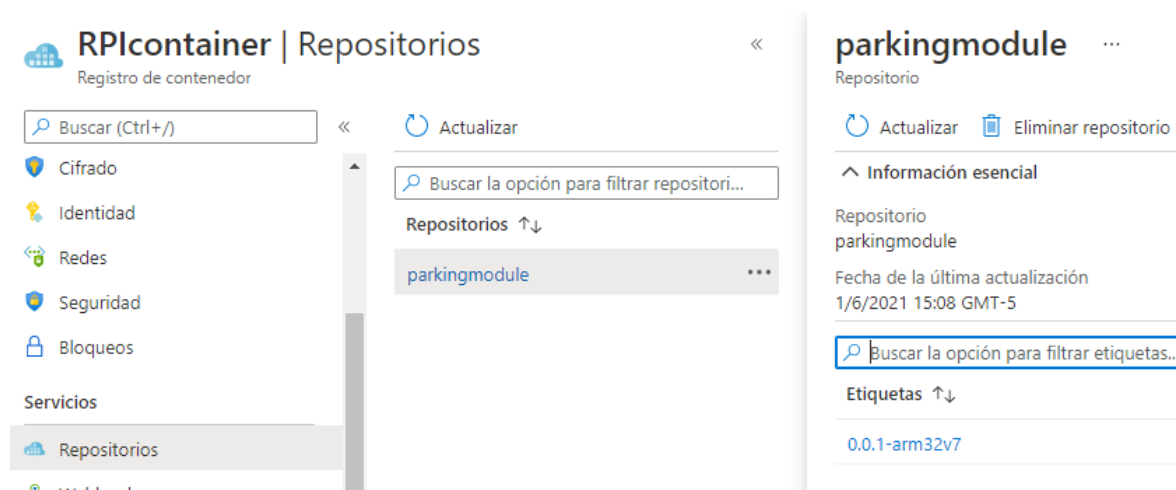
```
1 {
2   "$schema-template": "2.0.0",
3   "modulesContent": {
4     "sedgeAgent": {
5       "properties.desired": {
6         "schemaVersion": "1.0",
7         "runtime": {
8           "type": "docker",
9           "settings": {
10             "minDockerVersion": "v1.25",
11             "loggingOptions": "",
12             "registryCredentials": {
13               "rpcontainer": {
14                 "username": "$CONTAINER_REGISTRY_USERNAME_rpcontainer",
15                 "password": "$CONTAINER_REGISTRY_PASSWORD_rpcontainer",
16                 "address": "rpcontainer.azurecr.io"
17               }
18             }
19           }
20         }
21       }
22     }
23   }
24 }
```

Figura 17 configuración de despliegue Json.

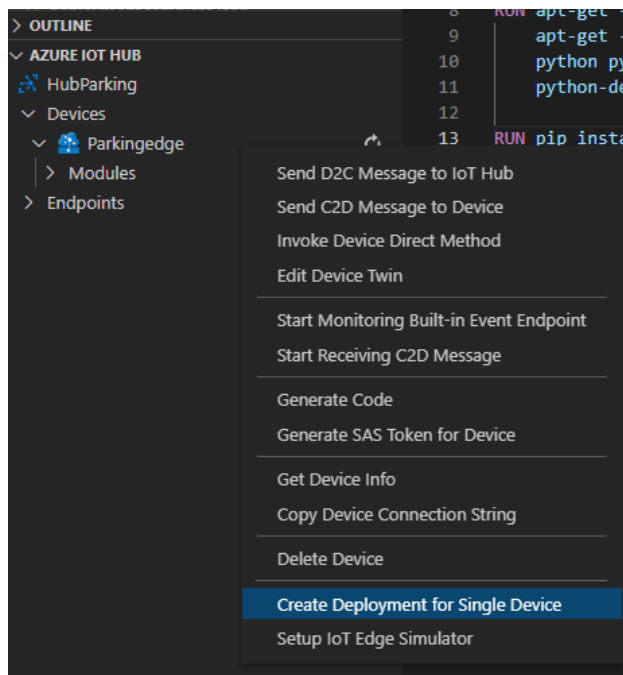
Para cargar el módulo al dispositivo primero se carga el contenedor en el repositorio de contenedores de Azure dando click derecho sobre el archivo deployment.template.json y seleccionando la opción de Build and Push IoT Edge Solution.



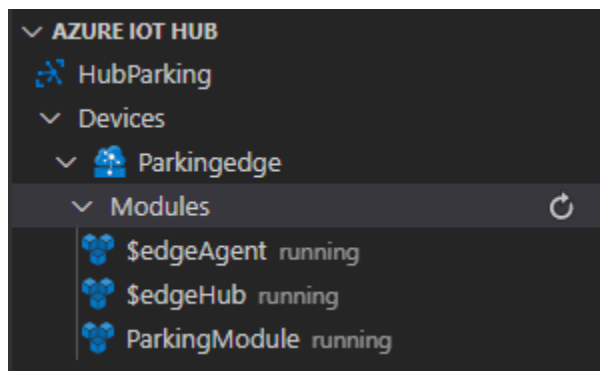
Se puede confirmar la carga del contenedor en el repositorio del registro de contenedor de Azure.



Y finalmente se despliega el contenedor en el dispositivo Edge dando click sobre el dispositivo y seleccionando la opción Create deployment for single Device.



Se puede verificar desplegando la opción de módulos donde se ven corriendo las aplicaciones.



Esto también se puede verificar desde la plataforma



Mensajes recibidos en el IoT Hub

```
[IoTHubMonitor] Start monitoring message arrived in built-in endpoint for device [Parkingedg] ...
[IoTHubMonitor] Created partition receiver [0] for consumerGroup [$Default]
[IoTHubMonitor] Created partition receiver [1] for consumerGroup [$Default]
[IoTHubMonitor] [11:32:45 AM] Message received from [Parkingedg]:
{"\parking:\" Parqueadero1 ,\"tiempo:\" 20210531:16:32:44 ,\"estado:\" disponible,\"temperatura:\" 21.
37009022508118}"
[IoTHubMonitor] [11:32:45 AM] Message received from [Parkingedg]:
{"\parking:\" Parqueadero1 ,\"tiempo:\" 20210531:16:32:45 ,\"estado:\" ocupado,\"temperatura:\" 21.
968527521784207}"
[IoTHubMonitor] [11:32:55 AM] Message received from [Parkingedg]:
{"\parking:\" Parqueadero1 ,\"tiempo:\" 20210531:16:32:55 ,\"estado:\" disponible,\"temperatura:\" 26.
010746785886223}"
[IoTHubMonitor] [11:33:16 AM] Message received from [Parkingedg]:
{"\parking:\" Parqueadero1 ,\"tiempo:\" 20210531:16:33:16 ,\"estado:\" ocupado,\"temperatura:\" 33.
65569361867175}"
[IoTHubMonitor] [11:33:26 AM] Message received from [Parkingedg]:
{"\parking:\" Parqueadero1 ,\"tiempo:\" 20210531:16:33:26 ,\"estado:\" disponible,\"temperatura:\" 24.
745685008138175}"
```

CONCLUSIONES

- Almacenamiento en la nube azure restringido por generación de costos por parte del proveedor.
- Azure tiene y permite desarrollar varios códigos en C#, Java, Node.js y Python.
- El desarrollo del proyecto presento menos inconvenientes realizando el proceso desde una maquina linux
- Azure brinda una facilidad de replicación de un modelo funcionalidad por medio contenedores en varios dispositivos
- Iot Edge Soporta tanto linux como windows ampliando la capacidad de soporte de dispositivos

BIBLIOGRAFÍA

- Kurt, G. (30 de Noviembre de 2018). *hackster.io*. Obtenido de <https://www.hackster.io/ngkurt/azure-iot-edge-with-sense-hat-and-raspberry-pi-06791b>
- Microsoft. (04 de 08 de 2020). *docs.microsoft*. Obtenido de <https://docs.microsoft.com/es-es/azure/iot-edge/tutorial-python-module?view=iotedge-2020-11>
- Microsoft. (05 de 03 de 2021). *docs.microsoft*. Obtenido de <https://docs.microsoft.com/en-us/azure/iot-hub/about-iot-hub>
- Microsoft Azure. (2020). *Azure IoT Hub*. Obtenido de <https://azure.microsoft.com/es-es/services/iot-hub/>

ANEXOS

Códigos implementados.

Implementación del contenedor.

```
FROM arm32v7/python:3.7-slim-buster

WORKDIR /app
RUN pip install azure-iot-device~=2.0.0
RUN apt-get -q update && \
    apt-get -qy install \
    python python-pip \
    python-dev python-pip gcc make
RUN pip install RPi.GPIO

COPY . .
CMD [ "python3", "-u", "./main.py" ]
```

Archivo de configuración Json.

```
{
  "$schema-template": "2.0.0",
  "modulesContent": {
    "$edgeAgent": {
      "properties.desired": {
        "schemaVersion": "1.0",
        "runtime": {
          "type": "docker",
          "settings": {
            "minDockerVersion": "v1.25",
            "loggingOptions": "",
            "registryCredentials": {
              "rpicontainer": {
                "username": "$CONTAINER_REGISTRY_USERNAME_rpicontainer",
                "password": "$CONTAINER_REGISTRY_PASSWORD_rpicontainer",
                "address": "rpicontainer.azurecr.io"
              }
            }
          }
        }
      }
    }
  }
}
```

```

    },
    "systemModules": {
      "edgeAgent": {
        "type": "docker",
        "settings": {
          "image": "mcr.microsoft.com/azureiotedge-agent:1.1",
          "createOptions": {}
        }
      },
    },
    "edgeHub": {
      "type": "docker",
      "status": "running",
      "restartPolicy": "always",
      "settings": {
        "image": "mcr.microsoft.com/azureiotedge-hub:1.1",
        "createOptions": {
          "HostConfig": {
            "PortBindings": {
              "5671/tcp": [
                {
                  "HostPort": "5671"
                }
              ],
              "8883/tcp": [
                {
                  "HostPort": "8883"
                }
              ],
              "443/tcp": [
                {
                  "HostPort": "443"
                }
              ]
            }
          }
        }
      }
    },
  },
  "modules": {
    "ParkingModule": {
      "version": "1.0",
      "type": "docker",
      "status": "running",
      "restartPolicy": "always",

```

```

        "settings": {
            "image": "${MODULES.ParkingModule.arm32v7}",
            "createOptions": "{\"HostConfig\":{\"Privileged\": true}}"
        }
    }
}
},
"$edgeHub": {
    "properties.desired": {
        "schemaVersion": "1.0",
        "routes": {
            "ParkingModuleToIoTHub": "FROM /messages/modules/ParkingModule/out
puts/* INTO $upstream",
            "sensorToParkingModule": "FROM /messages/modules/SimulatedTemperat
ureSensor/outputs/temperatureOutput INTO BrokeredEndpoint(\"/modules/Parking
Module/inputs/input1\")"
        },
        "storeAndForwardConfiguration": {
            "timeToLiveSecs": 7200
        }
    }
}
}
}
}
}

```

Codigo Python

```

import random
import time
import threading
import RPi.GPIO as GPIO
from datetime import datetime

# Using the Python Device SDK for IoT Hub:

# The sample connects to a device-specific MQTT endpoint on your IoT Hub.
from azure.iot.device import IoTHubDeviceClient, Message, MethodResponse

# The device connection string to authenticate the device with your IoT hub.
# Using the Azure CLI:
# az iot hub device-identity show-connection-string --hub-
name {YourIoTHubName} --device-id MyNodeDevice --output table

```

```

CONNECTION_STRING = "HostName=HubParking.azure-
devices.net;DeviceId=Parkededge;SharedAccessKey=Ai1AFw7iT8Jn8UbPfoMJ1Y1510Q
/U2sZCagnRExvnSk="

# Define the JSON message to send to IoT Hub.

MSG_TXT = '{"parking:" {id} ,"tiempo:" {tiempo} ,"estado:" {estado},"temper
atura:" {temperatura}}}'

port = 1
id="Parquadero1"
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.IN) #Read output from PIR motion sensor
GPIO.setup(13, GPIO.OUT) #LED output pin
TEMPERATURE = 20.0
HUMIDITY = 60
Accion = 0

def iothub_client_init():
    # Create an IoT Hub client
    client = IoTHubDeviceClient.create_from_connection_string(CONNECTION_STR
ING)
    return client

def iothub_client_telemetry_sample_run():

    try:
        client = iothub_client_init()
        print( "IoT Hub device sending periodic messages, press Ctrl-
C to exit" )

        while True:

            i=GPIO.input(11)
            if i==0:

                estado="evaluando"
                print(estado)
                GPIO.output(13,0)
                time.sleep(1)

```



```

elif i==1:

    estado="ocupado"
    print(estado)
    GPIO.output(13,1)
    now = datetime.now()
    hora_ingreso = now.strftime("%Y%m%d:%H:%M:%S")
    tiempo=hora_ingreso
    print("Hora de deteccion", hora_ingreso)
    temperatura = TEMPERATURE + (random.random() * 15)
    msg_txt_formatted = MSG_TXT.format(id=id,tiempo=tiempo,estado=estado,temperatura=temperatura)
    message = Message(msg_txt_formatted)
    print( "Sending message: {}".format(message) )
    client.send_message(message)
    print ( "Message successfully sent" )

    time.sleep(5)

    estado="disponible"
    GPIO.output(13,0)
    now = datetime.now()
    hora_salida = now.strftime("%Y%m%d:%H:%M:%S")
    tiempo=hora_salida
    temperatura = TEMPERATURE + (random.random() * 15)
    print("Hora de salida", hora_salida)
    print("temperatura salida",temperatura)
    msg_txt_formatted = MSG_TXT.format(id=id,tiempo=tiempo,estado=estado,temperatura=temperatura)
    message = Message(msg_txt_formatted)
    print( "Sending message: {}".format(message) )
    client.send_message(message)
    print ( "Message successfully sent")

except KeyboardInterrupt:

    print ( "IoTHubClient sample stopped" )

if __name__ == '__main__':

    print ( "IoT Hub Quickstart #2 - Simulated device" )
    print ( "Press Ctrl-C to exit" )
    iothub_client_telemetry_sample_run()

```