Chapter 6: IOT

6.1 Introduction

Internet of Things (**IoT**) is a system of interrelated computing devices which enables the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

The definition of the Internet of things has evolved due to the **convergence** of multiple technologies:

- Real-time analytics
- Machine learning
- Commodity sensors
- Embedded systems

Traditional fields of **embedded systems**, **wireless sensor networks**, **control systems**, **automation** (including home and building automation), and others all contribute to enabling the Internet of Things.

In the consumer market, IoT technology is most synonymous with products pertaining to the concept of the "smart home", covering devices and appliances (such as lighting fixtures, thermostats, home security systems and cameras, and other home appliances) that support one or more common ecosystems, and can be controlled via devices associated with that ecosystem, such as smartphones and smart speakers. [98]

Fig.6.1: IOT integrations.

6.2 Architecture

IOT system architecture, in its simplistic view, consists of three tiers:

6.2.1 Devices

Include networked things, such as the sensors and actuators found in IIOT (Industrial internet of things) equipment, particularly those that use protocols such as **Modbus**, **Zigbee**, or **proprietary protocols**, to connect to an Edge Gateway. [99]

Fig.6.2: ZigBee.

6.2.2 The Gateway

consists of sensor data aggregation systems called Edge Gateways that provide functionality, such as pre-processing of the data, securing connectivity to cloud, using systems such as Web-Sockets, the event hub, and, even in some cases, edge analytics or fog computing.

- A device that connects between the cloud and the controller.
- IoT Gateway feature set

A versatile IoT Gateway may perform any of the following:

- Facilitating communication with legacy or non-internet connected devices.
- Data caching, buffering and streaming
- Data pre-processing, cleansing, filtering and optimization
- Some data aggregation
- Device to Device communications/M2M
- Networking features and hosting live data
- Data visualization and basic data analytics via IoT Gateway applications
- Short term data historian features
- Security – manage user access and network security features
- Device configuration management
- System diagnostics

Some sensors generate tens of thousands of data points per second. A gateway provides a place to preprocess that data locally at the edge before sending it on to the cloud. When data is aggregated, summarized and tactically analyzed at the edge, it minimizes the volume of data that needs to be forwarded on to the cloud, which can have a big impact on response times and network transmission costs. [99]

Another benefit of an IoT gateway is that it can provide additional security for the IoT network and the data it transports. Because the gateway manages information moving in both directions, it can protect data moving to the cloud from leaks and IoT devices from being compromised by malicious outside attacks with features such as tamper detection, encryption, hardware random number generators and crypto engines. [99]

Examples

- Arduino
- ESP8266
- Raspberry pi 3
- PLC

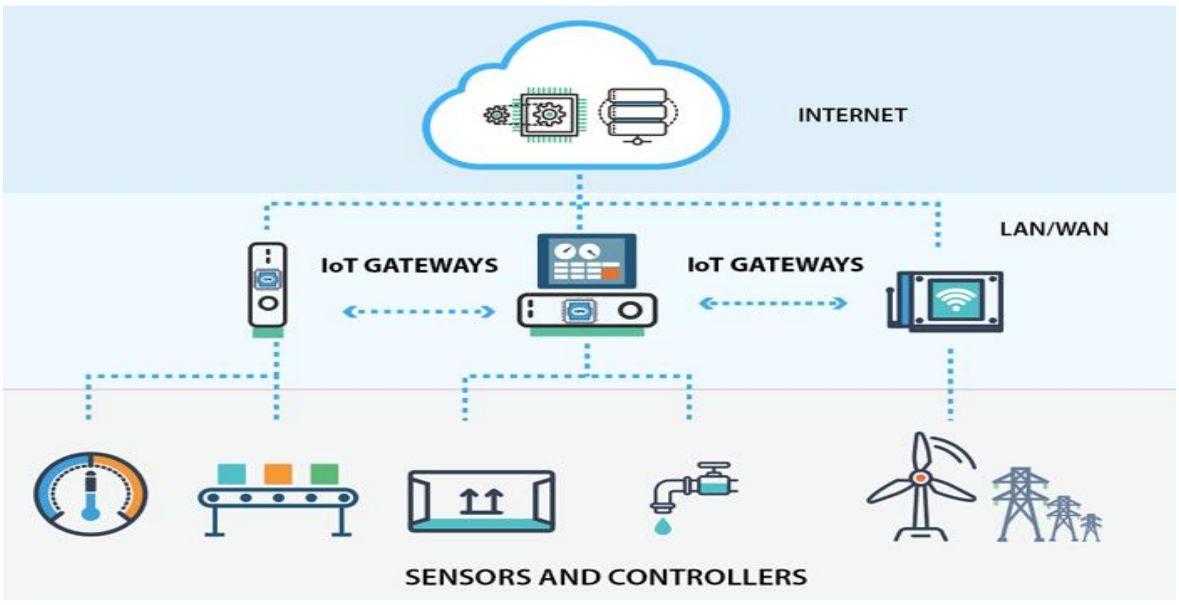


Fig.6.3: The Gateways.

6.2.3 The Cloud

The final tier includes the cloud application built for IIoT using the micro-services architecture, which are usually polyglot and inherently secure in nature using HTTPS/OAuth. It includes various database systems that store sensor data, such as time series databases or asset stores using backend data storage systems

(e.g. Cassandra, Postgres). The cloud tier in most cloud based IoT system features event queuing and messaging system that handles communication that
transpires in all tiers. Some experts classified the three-tiers in the IIoT system as edge, platform, and enterprise and these are connected by proximity network, access network, and service network, respectively.
Cloud is a platform that is designed to store and process Internet of Things (IoT) data. The platform is built to take in the massive volumes of data generated by devices, sensors, websites, applications, customers and partners and initiate actions for real-time responses. For example, wind turbines could adjust their behavior based on current weather data; airline passengers whose connecting flights are delayed or cancelled could be rebooked before the planes they are on have landed. [99]
An IoT cloud platform may be built on top of generic clouds such as those from Microsoft, Amazon, Google or IBM.
Network operators such as AT&T, Vodafone and Verizon may offer their own IoT platforms with stronger focus on network connectivity. Platforms could be vertically integrated for specific industries such as oil and gas, logistics and transportation, etc. Device manufacturers such as Samsung (ARTIK Cloud) are also offering their own IoT cloud platforms. [99]

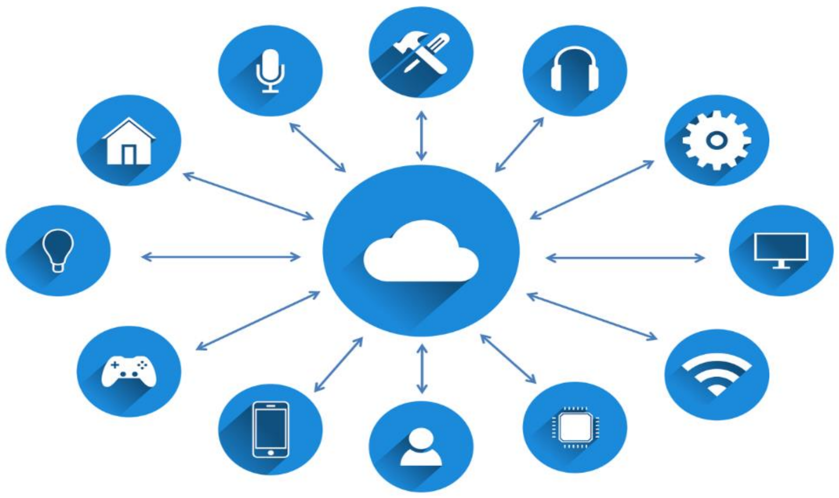


Fig.6.4: The Cloud.

6.2.4 dashboard

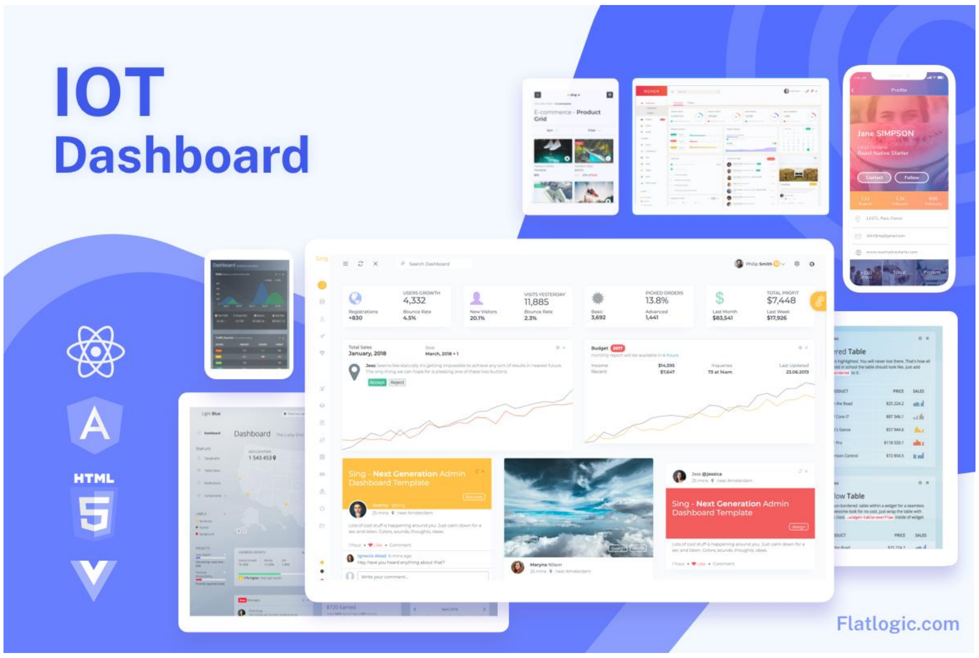


Fig.6.5: IOT Dashboard.

For the Internet of Things, or any control system, the dashboard or IoT dashboard is the key HMI (Human-Machine Interface) component that organizes and presents digital information from our physical world into a simply understood display on a computer or mobile device. With the help of IoT Dashboards, users and operators can (remotely) monitor and control specific assets and processes, and depending on safety requirements, access and control an environment from anywhere in the world. [99]

Attributes and Advantages of IoT Dashboards

They monitor and control physical assets. The IoT can be simply understood as the digitalization of our physical world; while many common data sources are purely digital like financial stock prices, the Internet of Things utilizes input/output devices and sensors from our physical world to display insights from an environment or its systems. IoT dashboards populated with graphs, charts, control switches, maps, tables, and countless other widgets are the digitals tools we use to visualize and display data coming from the physical world to our computers. [100]

They're used by both businesses and individuals. While the use of dashboard to view company, stats is not new for businesses, making the data readily available to employees,

management, and customers at the same time is. Businesses are now adopting the IoT to incorporate cloud data-analytics to improve operating efficiency and worker safety, then relay this data to customers or vendors for increased product transparency. Similarly, individuals are adopting the IoT to improve the efficiency of one's own mind and body thanks to fitness trackers and home alarm systems. In both scenarios, end-users have access to specific IoT dashboards as HMIs that present data relating to the status of a system or events (door open, smoke detected, 4 miles traversed); all the while not requiring programming skills to operate either system. [100]

They're cloud-based and global. The Internet of Things (IoT) adoption is – in part – thanks to the expansion of cloud computing and its proficient data collection, processing, and analysis capabilities. With the global accessibility of cloud data-storage platforms like AWS, Azure, Blumix, and Google Cloud no longer do businesses or private users need servers' rooms to store data nor the on-hand IT engineer to run it. With the global architecture of most cloud or IoT service providers, IoT dashboards can be accessed simply with a URL and any standard browser or mobile application – anywhere in the world. [100]

Examples of IoT Dashboards

- Real-time GPS asset tracking and condition monitoring
- Energy and Environment monitoring
- Ubidots IoT dashboards

6.3 Development Kits

6.3.1 Esp 8266 NodeMCU

6.3.1.1 Features

- Open source
- Interactive
- Programmable
- Low cost
- Simple
- Smart
- WI-FI enabled
- Arduino like hardware

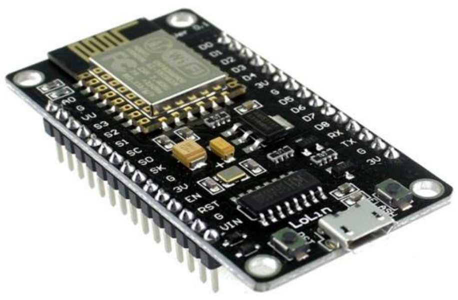


Fig.6.6: Esp 8266 NodeMCU

Advanced API for hardware IO, which can dramatically reduce the redundant work for configuring and manipulating hardware. Code like arduino, but interactively in Lua script.[101]

- Nodejs style network API

Event-driven API for network applications, which facilitates developers writing code running on a 5mm*5mm sized MCU in Nodejs style. Greatly speed up your IOT application developing process.

- Lowest cost WI-FI

Less than $2 WI-FI MCU ESP8266 integrated and easy to prototype development kit. We provide the best platform for IOT application development at the lowest cost. [102]

6.3.1.2 Pinout Diagram

The Development Kit based on ESP8266, integrates GPIO, PWM, IIC, 1-Wire and ADC all in one board. Power your development in the fastest way combination with NodeMcu Firmware.

- USB-TTL included, plug&play
- 10 GPIO, every GPIO can be PWM, I2C, 1-wire
- FCC CERTIFIED WI-FI module, PCB antenna. [102]

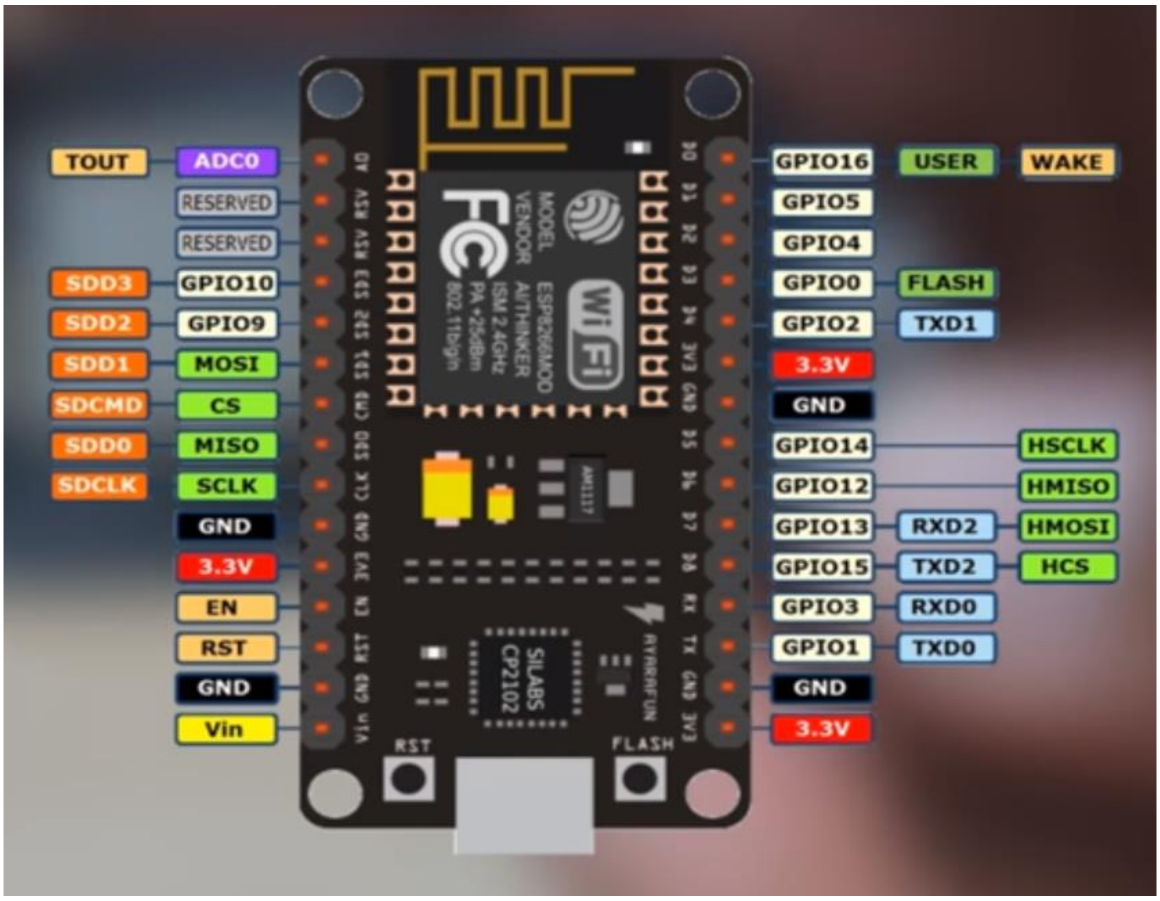


Fig.6.7: ESP 8266 NodeMCU Pinout diagram

6.3.1.3 CPU

The ESP8266EX integrates a Tensilica L106 32-bit RISC processor, which achieves extra low power consumption and reaches a maximum clock speed of 160 MHz. The Real-Time Operating System (RTOS) and Wi-Fi stack allow 80% of the processing power to be available for user application programming and development. The CPU includes the interfaces as below:

• Programmable RAM/ROM interfaces (iBus), which can be connected with memory controller, and can also be used to visit flash.

• Data RAM interface (dBus), which can connect with memory controller.

• AHB interface which can be used to visit the register. [103]

6.3.1.4 Memory

ESP8266EX Wi-Fi SoC integrates memory controller and memory units including SRAM and ROM. MCU can access the memory units through iBus, dBus, and AHB interfaces. All memory units can be accessed upon request, while a memory arbiter will decide the running sequence according to the time when these requests are received by the processor.

According to our current version of SDK, SRAM space available to users is assigned as below.

• RAM size < 50 kB, that is, when ESP8266EX is working under the Station mode and connects to the router, the maximum programmable space accessible in Heap + Data section is around 50 kB.

• There is no programmable ROM in the SoC. Therefore, user program must be stored in an external SPI flash. [103]

6.3.1.5 External Flash

ESP8266EX uses external SPI flash to store user programs and supports up to 16 MB memory capacity theoretically.

The minimum flash memory of ESP8266EX is shown below:

• OTA disabled: 512 kB at least

• OTA enabled: 1 MB at least.

6.3.1.6 Wi-Fi

ESP8266EX implements TCP/IP and full 802.11 b/g/n WLAN MAC protocol. It supports Basic Service Set (BSS) STA and SoftAP operations under the Distributed Control Function (DCF). Power management is handled with minimum host interaction to minimize active duty period.

6.3.1.7 Wi-Fi Radio and Baseband

The ESP8266EX Wi-Fi Radio and Baseband support the following features:

• 802.11b and 802.11g

• 802.11n MCS0-7 in 20 MHz bandwidth

• 802.11n 0.4 µs guard-interval.

• Up to 72.2 Mbps of data rate**.**

• Receiving STBC 2x1

• Up to 20.5 dBm of transmitting power

• Adjustable transmitting power

• Antenna diversity**.** [103]

6.3.2 Esp32 DEVKIT:

ESP32 Wifi chip is a successor of the famous ESP8266.

6.3.2.1 Features

Compared to it, every feature is enhanced (speed up to 240 MHz, two cores, 520 kiB RAM, number of GPIOs, variety of peripherals, etc.) and there are some new ones (Bluetooth: legacy/BLE, 4 MiB-flash memory encryption capability, cryptographic hardware acceleration: AES, SHA-2, RSA, ECC, RNG). There are a lot of resources on the web concerning ESP32.

- The ESP32 is dual core, this means it has 2 processors.
- It has Wi-Fi and Bluetooth built in. It runs 32-bit programs.
- The clock frequency can go up to 240MHz and it has a 512 kB RAM.
- This board has 30 or 36 pins, 15 in each row.

- It also has wide variety of peripherals available, like capacitive touch, ADCs, DACs, UART, SPI, I2C and much more.
- It comes with built-in hall effect sensor and built-in temperature sensor. [104]

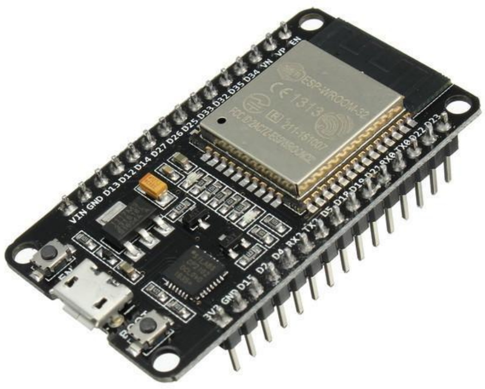
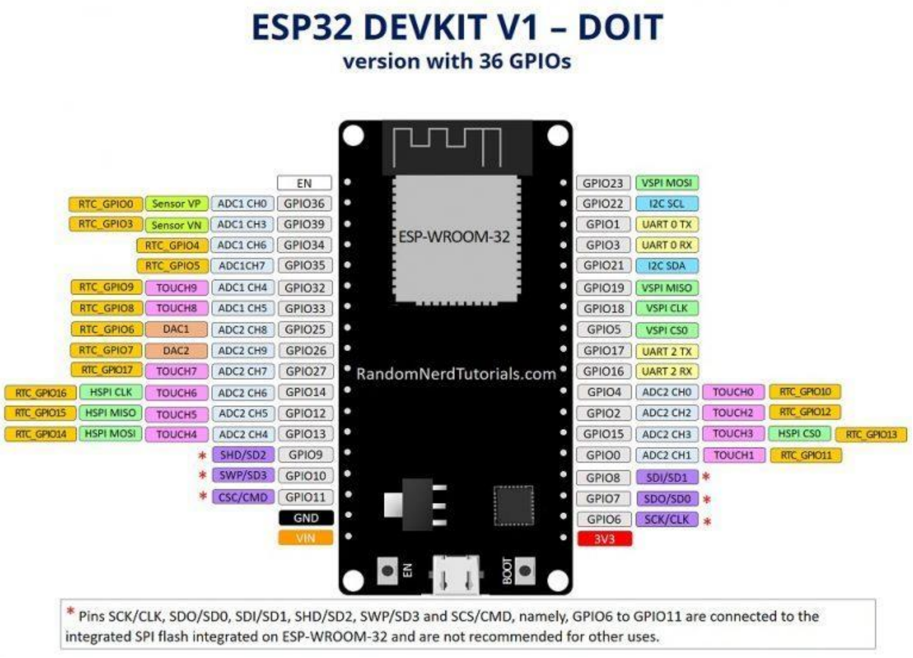


Fig.6.8: Esp32

6.3.2.2 Pinout Diagram

With the ESP32 you can decide which pins are UART, I2C, or SPI – you just need to set that on the code. This is possible due to the ESP32 chip's multiplexing feature that allows to assign multiple functions to the same pin. If you don't set them on the code, the pins will be used as default – as shown in the figure below (the pin location can change depending on the manufacturer). [105]


ESP32 DEVKIT V1 – DOIT
version with 36 GPIOs
ESP-WROOM-32
RandomNerdTutorials.com
EN
RTC_GPIO0 Sensor VP ADC1 CH0 GPIO36
RTC_GPIO3 Sensor VN ADC1 CH3 GPIO39
RTC_GPIO4 ADC1 CH6 GPIO34
RTC_GPIO5 ADC1CH7 GPIO35
RTC_GPIO9 TOUCH9 ADC1 CH4 GPIO32
RTC_GPIO8 TOUCH8 ADC1 CH5 GPIO33
RTC_GPIO6 DAC1 ADC2 CH8 GPIO25
RTC_GPIO7 DAC2 ADC2 CH9 GPIO26
RTC_GPIO17 TOUCH7 ADC2 CH7 GPIO27
RTC_GPIO16 HSPI CLK TOUCH6 ADC2 CH6 GPIO14
RTC_GPIO15 HSPI MISO TOUCH5 ADC2 CH5 GPIO12
RTC_GPIO14 HSPI MOSI TOUCH4 ADC2 CH4 GPIO13
* SHD/SD2 GPIO9
* SWP/SD3 GPIO10
* CSC/CMD GPIO11
GND
VIN
GPIO23 VSPI MOSI
GPIO22 I2C SCL
GPIO1 UART 0 TX
GPIO3 UART 0 RX
GPIO21 I2C SDA
GPIO19 VSPI MISO
GPIO18 VSPI CLK
GPIO5 VSPI CS0
GPIO17 UART 2 TX
GPIO16 UART 2 RX
GPIO4 ADC2 CH0 TOUCH0 RTC_GPIO10
GPIO2 ADC2 CH2 TOUCH2 RTC_GPIO12
GPIO15 ADC2 CH3 TOUCH3 HSPI CS0 RTC_GPIO13
GPIO0 ADC2 CH1 TOUCH1 RTC_GPIO11
GPIO8 SDI/SD1 *
GPIO7 SDO/SD0 *
GPIO6 SCK/CLK *
3V3
EN
BOOT

* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

Fig.6.9: ESP32 Pinout diagram

This is the ESP32 development board with the incredible module ESP32 from Espressif. This is very powerful module which exceds by far the previous famous module ESP8266. The ESP32 has both WIFI and Bluetooth 4 capabilities (BLE). It also has 32 bit double core CPU, one dedicated for the wireless (WIFI & Bluetooth) and the other dedicated for the logic and control. [105]

<u>The ESP32 also have 36 PIN as follows:</u>

- **Analog-to-Digital Converter (ADC**) – Up to 16 channels of 12-bit SAR ADC's. The ADC range can be set, in firmware, to either 0-1V, 0-1.4V, 0-2V, or 0-4V – no weirder 0-1V ADC.
- **Digital-to-Analog Converter (DAC)** – Two 8-bit DAC's to produce true analog voltages
- **Pulse-Width Modulation (PWM)** – Up to 16 channels of PWM-capable pins for dimming LEDs or controlling motors.
- **Touch Sensor** – 10 GPIOs feature capacitive sensing; make a 10-key buttonpad.
- **UART** - Two UART interfaces, one is used to load code serially. They feature flow control, and support IrDA too.
- **I2C, SPI, I2S** – There are two I^{2}C and four SPI interfaces to hook up all sorts of sensors and peripherals, plus two I2S interfaces if you want to add sound to your project. [105]

6.4 Google cloud platform

Fig.6.10: Google Cloud

Google Cloud Platform (GCP), offered by Google, is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as

Google Search and YouTube. Alongside a set of management tools, it provides a series of modular cloud services including computing, data storage, data analytics and machine learning.

Registration requires a credit card or bank account details. **Google Cloud Platform** provides **infrastructure as a service**, **platform as a service**, and **serverless computing environments.**

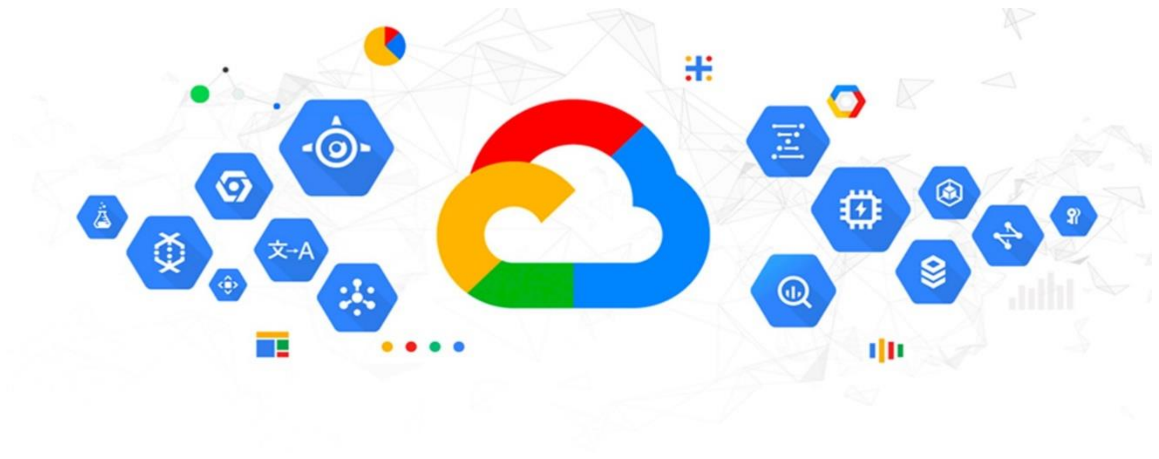


Fig.6.11: Google Cloud

Google Cloud Platform is a part of Google Cloud, which includes the Google Cloud Platform public cloud infrastructure, as well as G Suite, enterprise versions of Android and Chrome OS, and application programming interfaces (APIs) for machine learning and enterprise mapping services. [106]

6.4.1 Compute

- App Engine - Platform as a Service to deploy Java, PHP, Node.js, Python, C#, .Net, Ruby and Go applications.
- Compute Engine - Infrastructure as a Service to run Microsoft Windows and Linux virtual machines.
- Kubernetes Engine (GKE) or GKE on-prem offered as part of Anthos platform - Containers as a Service based on Kubernetes.
- Cloud Functions - Functions as a Service to run event-driven code written in Node.js, Python or Go.
- Cloud Run - Compute execution environment based on K-native. Offered as Cloud Run (fully managed) or as Cloud Run for Anthos. [106]

6.4.2 Storage & Databases

- Cloud Storage - Object storage with integrated edge caching to store unstructured data.
- Cloud SQL - Database as a Service based on MySQL and PostgreSQL.

- Cloud Bigtable - Managed NoSQL database service.
- Cloud Spanner - Horizontally scalable, strongly consistent, relational database service.
- Cloud Datastore - NoSQL database for web and mobile applications.
- Persistent Disk - Block storage for Compute Engine virtual machines.
- Cloud Memory Store - Managed in-memory data store based on Redis.

6.4.3 Networking

- VPC - Virtual Private Cloud for managing the software defined network of cloud resources.
- Cloud Load Balancing - Software-defined, managed service for load balancing the traffic.
- Cloud Armor - Web application firewall to protect workloads from DDoS attacks.
- Cloud CDN - Content Delivery Network based on Google's globally distributed edge points of presence.
- Cloud Interconnect - Service to connect a data center with Google Cloud Platform
- Cloud DNS - Managed, authoritative DNS service running on the same infrastructure as Google.
- Network Service Tiers - Option to choose Premium vs Standard network tier for higher performing network.[106]

6.4.4 Big Data

- Big Query - Scalable, managed enterprise data warehouse for analytics.
- Cloud Dataflow - Managed service based on Apache Beam for stream and batch data processing.
- Cloud Dataproc - Big data platform for running Apache Hadoop and Apache Spark jobs.
- Cloud Composer - Managed workflow orchestration service built on Apache Airflow.
- Cloud Datalab - Tool for data exploration, analysis, visualization and machine learning. This is a fully managed Jupiter Notebook service.
- Cloud Dataprep - Data service based on Trifecta to visually explore, clean, and prepare data for analysis.
- Cloud Pub/Sub - Scalable event ingestion service based on message queues.
- Cloud Data Studio - Business intelligence tool to visualize data through dashboards and reports. [106]

6.4.5 Cloud AI

- Cloud AutoML - Service to train and deploy custom machine learning models. As of September 2018, the service is in Beta.

- Cloud TPU - Accelerators used by Google to train machine learning models.
- Cloud Machine Learning Engine - Managed service for training and building machine learning models based on mainstream frameworks.
- Cloud Job Discovery - Service based on Google's search and machine learning capabilities for recruiting ecosystem.
- Dialogflow Enterprise - Development environment based on Google's machine learning for building conversational interfaces.
- Cloud Natural Language - Text analysis service based on Google Deep Learning models.
- Cloud Speech-to-Text - Speech to text conversion service based on machine learning.
- Cloud Text-to-Speech - Text to speech conversion service based on machine learning.
- Cloud Translation API - Service to dynamically translate between thousands of available language pairs
- Cloud Vision API - Image analysis service based on machine learning
- Cloud Video Intelligence - Video analysis service based on machine learning. [106]

6.4.6 Management Tools

- Stackdriver - Monitoring, logging, and diagnostics for applications on Google Cloud Platform and AWS.
- Cloud Deployment Manager - Tool to deploy Google Cloud Platform resources defined in templates created in YAML, Python or Jinja2.
- Cloud Console - Web interface to manage Google Cloud Platform resources.
- Cloud Shell - Browser-based shell command line access to manage Google Cloud Platform resources.
- Cloud Console Mobile App - Android and iOS application to manage Google Cloud Platform resources.
- Cloud APIs - APIs to programmatically access Google Cloud Platform resources. [106]

6.4.7 Identity & Security

- Cloud Identity - Single sign-on (SSO) service based on SAML 2.0 and OpenID.
- Cloud IAM - Identity & Access Management (IAM) service for defining policies based on role-based access control.
- Cloud Identity-Aware Proxy - Service to control access to cloud applications running on Google Cloud Platform without using a VPN.
- Cloud Data Loss Prevention API - Service to automatically discover, classify, and redact sensitive data. [106]

- Security Key Enforcement - Two-step verification service based on a security key.
- Cloud Key Management Service - Cloud-hosted key management service integrated with IAM and audit logging.
- Cloud Resource Manager - Service to manage resources by project, folder, and organization based on the hierarchy.
- Cloud Security Command Center - Security and data risk platform for data and services running in Google Cloud Platform.
- Cloud Security Scanner - Automated vulnerability scanning service for applications deployed in App Engine.
- Access Transparency - Near real-time audit logs providing visibility to Google Cloud Platform administrators.
- VPC Service Controls - Service to manage security perimeters for sensitive data in Google Cloud Platform services. [106]

6.4.8 IoT

- Cloud IoT Core - Secure device connection and management service for Internet of Things.
- Edge TPU - Purpose-built ASIC designed to run inference at the edge. As of September 2018, this product is in private beta.
- Cloud IoT Edge - Brings AI to the edge computing layer.

6.4.9 API Platform

- Maps Platform - APIs for maps, routes, and places based on Google Maps.
- Apigee API Platform - Lifecycle management platform to design, secure, deploy, monitor, and scale APIs.
- API Monetization - Solution for API providers to create revenue models, reports, payment gateways, and developer portal integrations.
- Developer Portal - Self-service platform for developers to publish and manage APIs.
- API Analytics - Service to analyze API-driven programs through monitoring, measuring, and managing APIs.
- Apigee Sense - Enables API security by identifying and alerting administrators to suspicious API behaviors.
- Cloud Endpoints - A NGINX-based proxy to deploy and manage APIs.
- Service Infrastructure - A set of foundational services for building Google Cloud products.[106]

6.5 Google Firebase

Fig.6.12: Google Firebase

Firebase's first product was the Firebase Real-time Database, an API that synchronizes application data across iOS, Android, and Web devices, and stores it on Firebase's cloud. The product assists software developers in building real-time, collaborative applications. [107]

Services provided by firebase

Fig.6.13: Firebase services

6.5.1Firebase Analytics

Firebase Analytics is a cost-free app measurement solution that provides insights on app usage and user engagement.

6.5.2 Develop

6.5.2.1 Firebase Cloud Messaging

Formerly known as Google Cloud Messaging (GCM), Firebase Cloud Messaging (FCM) is a cross-platform solution for messages and notifications for Android, iOS, and web applications, which as of 2016 can be used at no cost. [107]

6.5.2.3 Firebase Auth

Firebase Auth is a service that can authenticate users using only client-side code. It supports social login providers Facebook, GitHub, Twitter and Google (and Google Play Games). Additionally, it includes a user management system whereby developers can enable user authentication with email and password login stored with Firebase. [107]

6.5.2.4 Firebase Realtime Database

Firebase provides a real-time database and back-end as a service. The service provides application developers an API that allows application data to be synchronized across clients and stored on Firebase's cloud.

The company provides client libraries that enable integration with Android, iOS, JavaScript, Java, Objective-C, Swift and Node.js applications. The database is also accessible through a REST API and bindings for several JavaScript frameworks such as AngularJS, React, Ember.js and Backbone.js. The REST API uses the Server-Sent Events protocol, which is an API for creating HTTP connections for receiving push notifications from a server. Developers using the realtime database can secure their data by using the company's server-side-enforced security rules. [107]

6.5.2.5 Cloud Firestore

On January 31, 2019, Cloud Firestore was officially brought out of beta, making it an official product of the Firebase lineup. It is the successor to Firebase's original databasing system, Real-time Database, and allows for nested documents and fields rather than the tree-view provided in the Real-time Database. [107]

6.2.5.6 Firebase Storage

Firebase Storage provides secure file uploads and downloads for Firebase apps, regardless of network quality, to be used for storing images, audio, video, or other user-generated content. It is backed by Google Cloud Storage. [107]

6.2.5.7 Firebase Hosting

Firebase Hosting is a static and dynamic web hosting service that launched on May 13, 2014. It supports hosting static files such as CSS, HTML, JavaScript and other files, as well as support through Cloud Functions. The service delivers files over a content delivery network (CDN) through HTTP Secure (HTTPS) and Secure Sockets Layer encryption (SSL). Firebase partners with Fastly, a CDN, to provide the CDN backing Firebase Hosting. The company states that Firebase Hosting grew out of customer requests; developers were using Firebase for its real-time database but needed a place to host their content.[107]

6.2.5.8 ML Kit

ML Kit is a mobile machine learning system for developers launched on May 8, 2018, in beta during the Google I/O 2018. ML Kit APIs feature a variety of features including optical character recognition, detecting faces, scanning barcodes, labelling images and recognising landmarks. It is currently available for iOS or Android developers. You may also import your own TensorFlow Lite models, if the given APIs are not enough. The APIs can be used on-device or on-cloud. [107]

6.5.3 Stability

6.5.3.1 Crashlytics

Crash Reporting creates detailed reports of the errors in the app. Errors are grouped into clusters of similar stack traces and triaged by the severity of impact on app users. In addition to automatic reports, the developer can log custom events to help capture the steps leading up to a crash. Before acquiring Crashlytics, Firebase was using its own Firebase Crash Reporting.[107]

6.5.3.2 Performance

Firebase Performance provides insights into an app's performance and the latencies the app's users experience.[107]

6.5.3.3 Firebase Test Lab

Firebase Test Lab provides cloud-based infrastructure for testing Android and iOS apps in one operation. Developers can test their apps across a wide variety of devices and device configurations. Test results—including logs, videos, and screenshots—are made available in the Firebase console. Even if a developer hasn't written any test code for their app, Test Lab can exercise the app automatically, looking for crashes. Test Lab for iOS is currently in beta. [107]

6.5.3.4 Admob

Admob is a Google product that integrates with Firebase audience. [107]

6.5.4 Grow

6.5.4.1 Firebase Dynamic Links

Dynamic Firebase links are smart URLs that dynamically change their behavior to provide "the best available experience" across multiple platforms, including desktop web browsers, iOS, and Android, and in-depth links to mobile apps. Dynamic Links work in all app installs: if the user opens Dynamic Link on iOS or Android and the application is not installed, the user will be prompted to install the app first. Once installed, the application will start running and can access the link. [107]

6.6 Test project

we conducted two test projects to help us understand how IOT works

6.6.1 check the weather using GCP-Cloud IoT Core with ESP32 and Mongoose OS

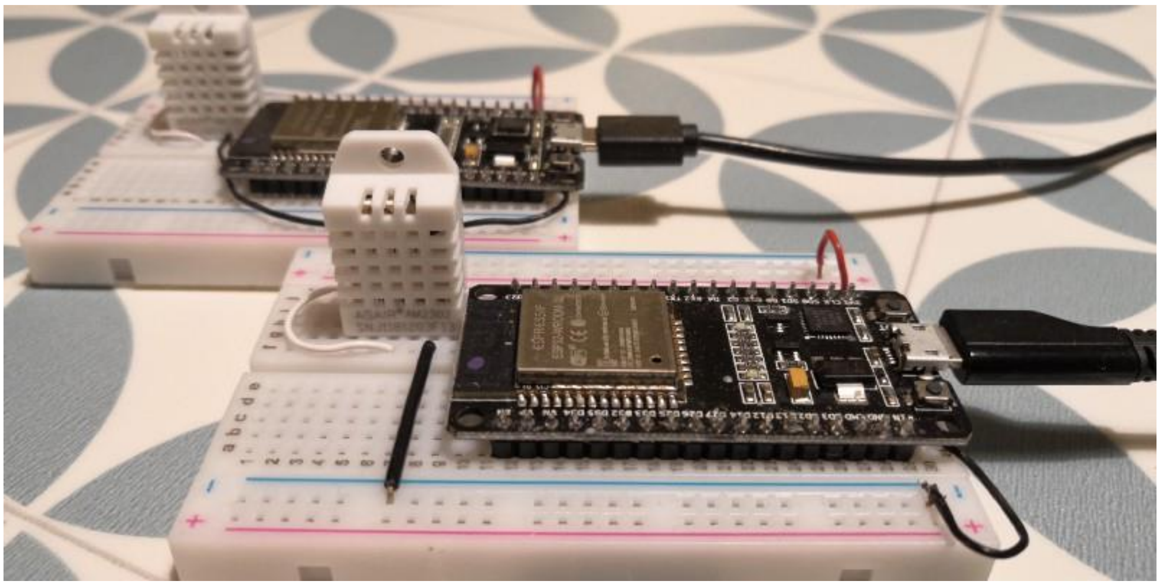


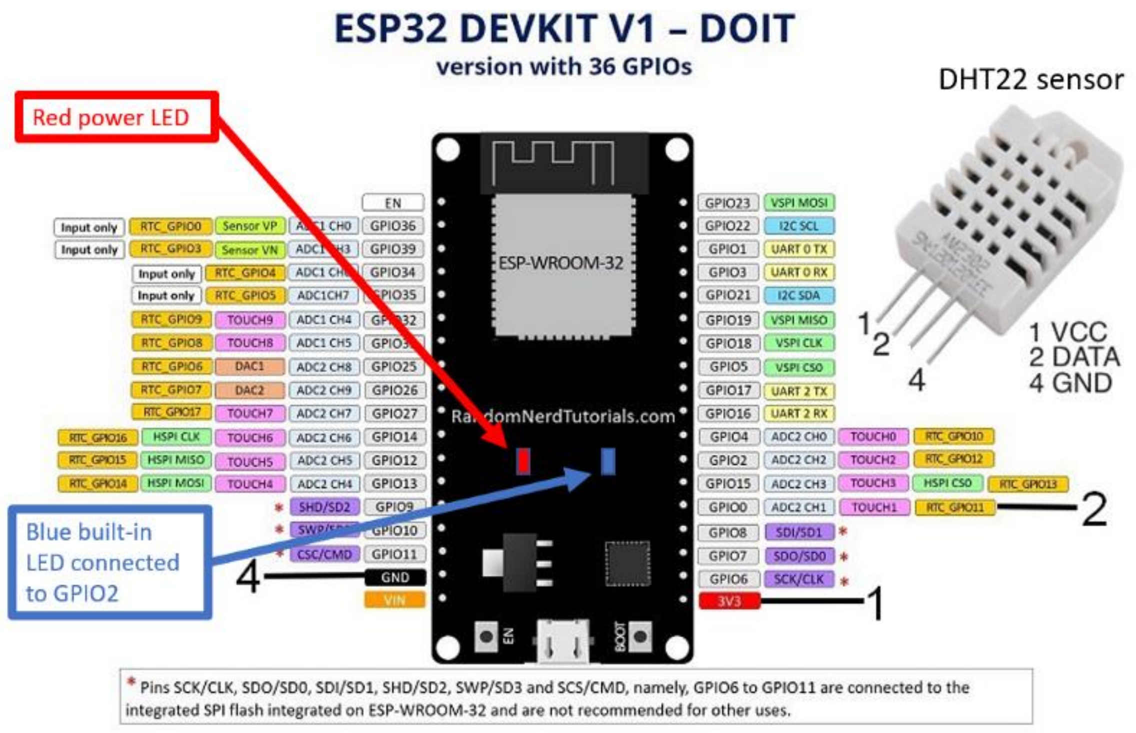
Fig.6.14: DHT sensor and NodeMCU

6.6.1.2 Project hardware: ESP32 & DHT22

- **DHT22**
- ➢ Low cost.
- ➢ 3 to 5V power and I/O.
- ➢ 2.5mA max current use during conversion (while requesting data).
- ➢ Good for 0-100% humidity readings with 2-5% accuracy.
- ➢ Good for -40 to 80℃ temperature readings ±0.5℃ accuracy.
- ➢ No more than 0.5 Hz sampling rate (once every 2 seconds)
- ➢ Body size 15.1mm x 25mm x 7.7mm

➢ 4 pins with 0.1 inch spacing

We can already build the following assembly twice (one for indoor and one for outdoor). For now, power will come from the USB connector connected to our host machine. In production, power may come from a power bank. [108]



* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

Fig.6.15: Connection diagram

6.6.1.3 Project architect

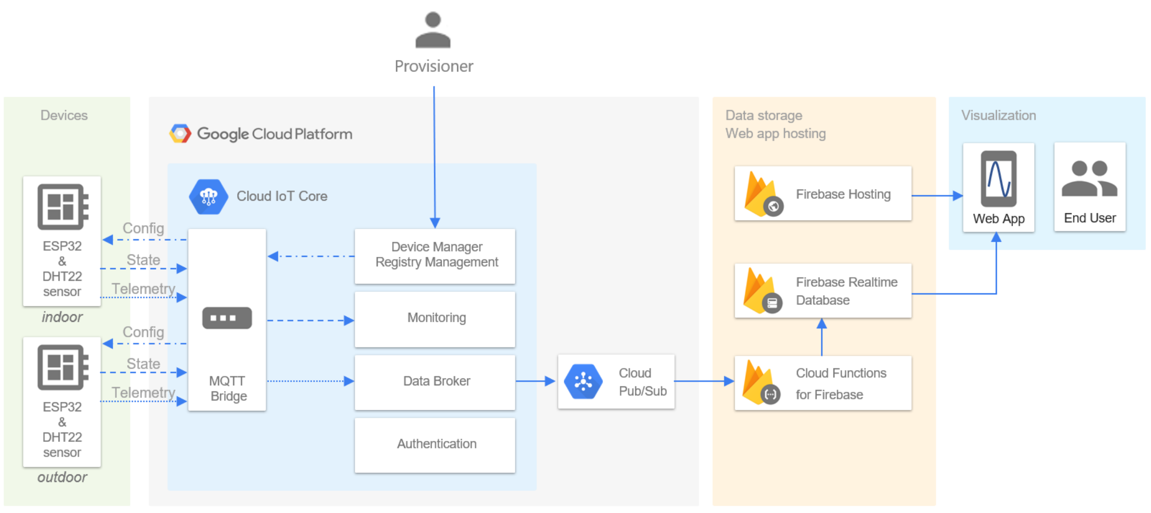


Fig.6.16: Connection diagram

There is no gateway between our devices and Cloud IoT Core because they “speak” **MQTT**.

Devices can also communicate with Cloud IoT Core via its HTTP bridge. As it is less performant than the MQTT bridge. [108]

6.6.1.4 Let’s explain this architecture in three sections

- “From Devices to Cloud Pub/Sub” describes the classical Google IoT architecture.
- “From Cloud Pub/Sub to data storage and visualization”, describes the choices we made to exploit data.
- “Additional config and state topics” complete this architecture presentation.

1. From Devices to Cloud Pub/Sub

- **Cloud IoT Core**

Cloud IoT Core is the Google Cloud Platform service to which each of our registered devices will send temperature/humidity data. When such a data is sent, we say that the device publishes a telemetry event (sometimes also called a “telemetry message”).

- **MQTT**

This publication is done through a MQTT connection. MQTT is a publish/subscribe-based message protocol; most of the time it lies over TCP (or better: over TLS, itself being over TCP). The telemetry message has to be published by the device (a MQTT client) to the Cloud Iot Core “MQTT bridge” (a MQTT server) in a MQTT topic whose name imperatively respects this format:

/devices/{device-id}/events

- **Quality of Service (QoS)**

The MQTT specification describes three Quality of Service (QoS) levels, when publishing to a topic

> QoS 0, the message is delivered at most once;

> QoS 1, the message is delivered at least once;

> QoS 2, the message is delivered exactly once.

Cloud IoT Core does not support QoS 2. And QoS 1 is better than QoS 0. So QoS 1 is the one we will adopt. Mongoose OS can do that.

- **Registry**

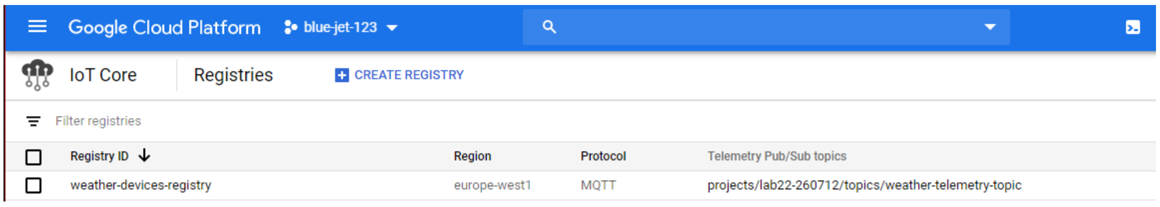
Devices sharing the same purpose are regrouped within a registry.

- **Cloud pub/sub**

Telemetry data from all devices belonging to the same registry is then forwarded to a Cloud Pub/Sub topic (Cloud Pub/Sub is a GCP product, not specifically a Cloud IoT Core one) [108]. The name of the Cloud Pub/Sub topic follows this pattern:

projects/id-of-google-cloud-project/topics/name-of-telemetry-topic

So, if we call our Google Cloud project blue-jet-123, if we choose weather-telemetry-topic for the name of our Pub/Sub telemetry topic and finally our registry is called weather-devices-registry, we'll get sooner or later that kind of view in Google Cloud Console:



Registry ID	Region	Protocol	Telemetry Pub/Sub topics
weather-devices-registry	europe-west1	MQTT	projects/lab22-260712/topics/weather-telemetry-topic

Fig.6.17: IOT Core

2. From Cloud Pub/Sub to data storage and visualization

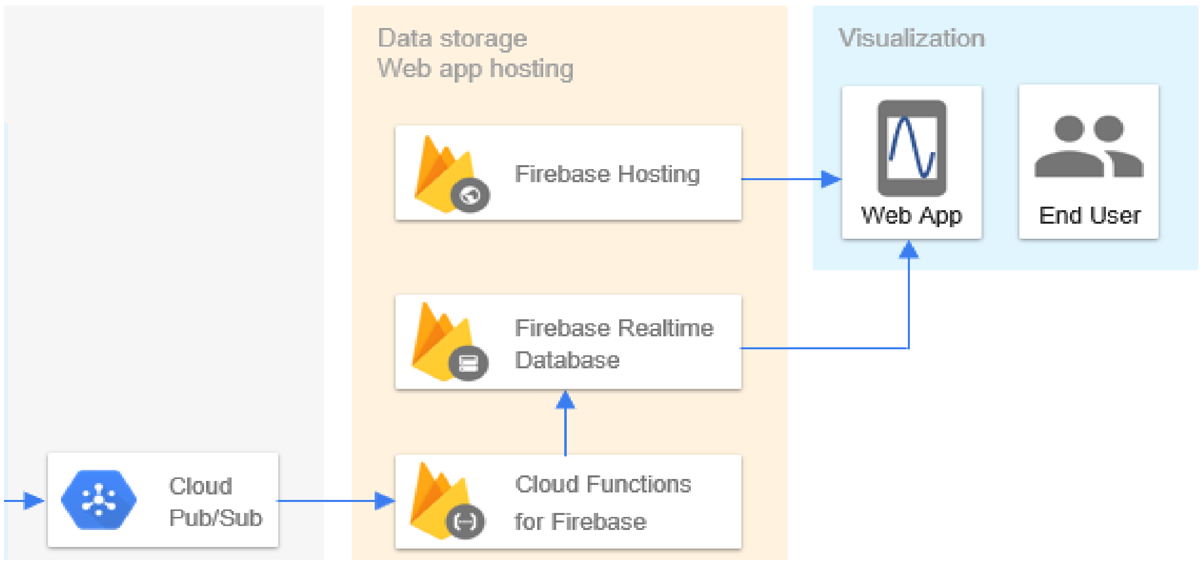


Fig.6.18: Pub/Sub to data storage

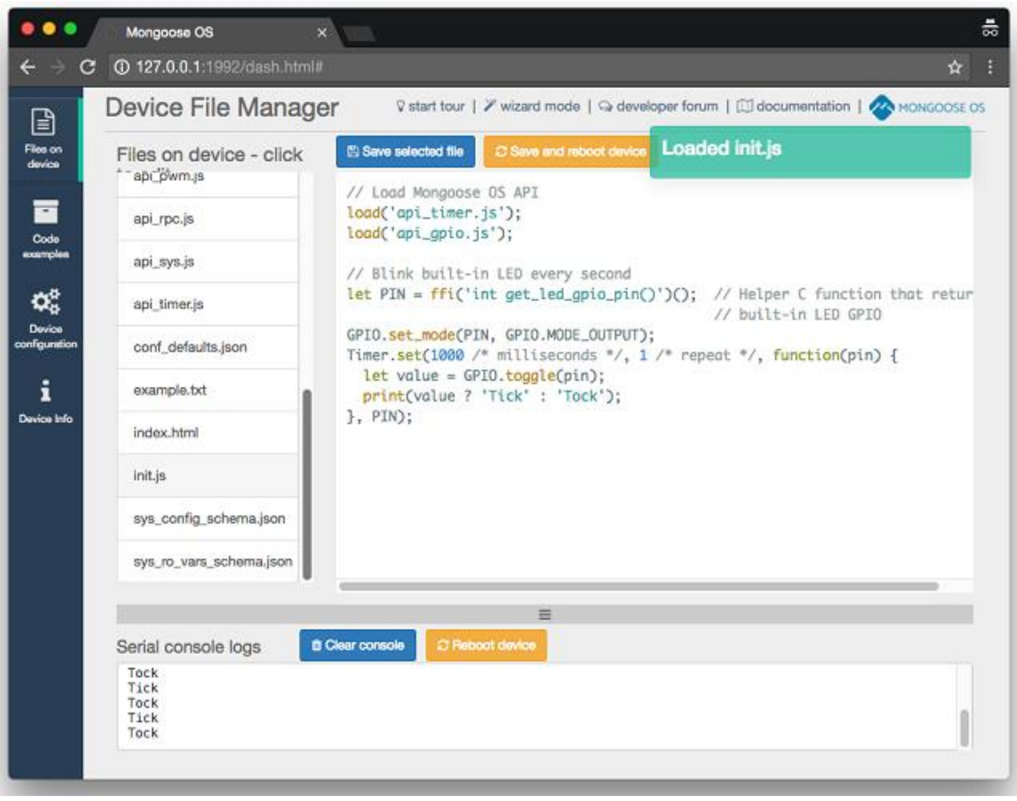
publication to the Cloud Pub/Sub topic will trigger a Firebase Cloud Function that will itself fulfill a Firebase Realtime Database with the new data. A web app hosted by Firebase Hosting will lively plot data from the Firebase Realtime Database. [108]

6.6.1.5 Mongoose OS

Mongoose OS is a smart IoT-oriented OS, runnable on several chips, including ESP8266 and ESP32. Mongoose OS is in partnership with the major actors in IoT.

Fig.6.19: Mongoose OS

It comes with a development tool called mos., working either in a UI or with a Command Line Terminal (like cmd.exe in Windows). In either cases, we'll write mos commands. There is also a device management app called mDash but we didn't try it. Numerous APIs dealing with most of the network and sensor protocols are provided. Programs can be written in both C/C++ and JS. [108]



```
// Load Mongoose OS API
load('api_timer.js');
load('api_gpio.js');

// Blink built-in LED every second
let PIN = ffi('int get_led_gpio_pin()')();  // Helper C function that retur
                                             // built-in LED GPIO
GPIO.set_mode(PIN, GPIO.MODE_OUTPUT);
Timer.set(1000 /* milliseconds */, 1 /* repeat */, function(pin) {
  let value = GPIO.toggle(pin);
  print(value ? 'Tick' : 'Tock');
}, PIN);
```

```
Tock
Tick
Tock
Tick
Tock
```

Fig.6.20: Mongoose OS UI

The following code has two functions

- The first it sends the DHT sensor readings to google cloud which will be connected later with firebase
- The second it receives a number (1&0) from google cloud configuration to control a LED from google cloud

```
load('api_config.js');
load('api_dht.js');
load('api_timer.js');
load('api_mqtt.js');
load('api_gpio.js');
let led = 2;
let topic = '/devices/' + Cfg.get('device.id') + '/config';
GPIO.set_mode(led, GPIO.MODE_OUTPUT);
```

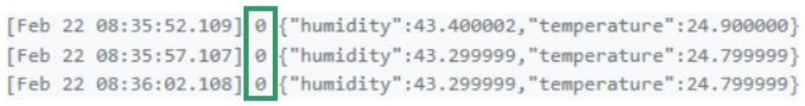
```
MQTT.sub(topic, function(conn, topic, msg) {
    print('Topic:', topic, 'message:', msg);
    let obj = JSON.parse(msg) || {led: 1};
    GPIO.write(led, obj.led);
    }, null);

// Telemetry topic must have this name:
let topic = '/devices/' + Cfg.get('device.id') + '/events';
let pin = 4;
let dht = DHT.create(pin, DHT.DHT22);

Timer.set(10000, true, function() { // timer period is in ms
    let msg = JSON.stringify({temperature: dht.getTemp(), humidity:
dht.getHumidity()});
    // Publish message with a QoS 1
    // MQTT.pub() returns 1 in case of success, 0 otherwise.
    let ok = MQTT.pub(topic, msg, 1);
    print(ok, msg);
    }, null);
```

We name this file **init.js**, upload it to Mongoose file system, then provoke a reset:

mos put fs/init.js

mos call Sys.Reboot

When running, this last program prints data to console but it fails to publish data to the MQTT bridge of Cloud IoT Core (MQTT.pub() returns 0):



```
[Feb 22 08:35:52.109] 0 {"humidity":43.400002,"temperature":24.900000}
[Feb 22 08:35:57.107] 0 {"humidity":43.299999,"temperature":24.799999}
[Feb 22 08:36:02.108] 0 {"humidity":43.299999,"temperature":24.799999}
```

Fig.6.21: MQTT returns 0

6.6.1.6 Cloud IoT Core project setup

Google Cloud SDK installation

Firstly, we need to install Google Cloud SDK because we will have to type some gcloud commands in a Command Line Terminal. At the time of writing, it requires Python 2.7. It won't work with Python 3.5. The Google Cloud SDK download page offers versions of the SDK with Python bundled inside (if you're sure you don't have Python already installed and don't want to handle this Python point). [108]

Then, Cloud IoT Core requires some Beta versions of gcloud commands. So in a Command Line Terminal, from any folder, we type:

gcloud components install beta. These two previous steps have to be done just one time!

Note: Most of the following actions on Google IoT Core can be performed in three ways:

- with Google Cloud Console (on the web)
- with some APIs in different languages, and
- with Command Line Interface in a terminal, typing gcloud commands.

We will use the latter to configure things and we'll check facts with Google Cloud Console (on the web).

Google Cloud project setup

```
# Get authenticated with Google Cloud
gcloud auth login


# Create cloud project. We chose hello-cloud-iot-core as PROJECT_ID
gcloud projects create lab22-260712


# Give Cloud IoT Core permission to publish to Pub/Sub topics
gcloud projects add-iam-policy-binding lab22-260712 --member=serviceAccount:cloud-iot@system.gserviceaccount.com --role=roles/pubsub.publisher


# Set default project for gcloud
gcloud config set project lab22-260712


# Create Pub/Sub topic for device telemetry
```

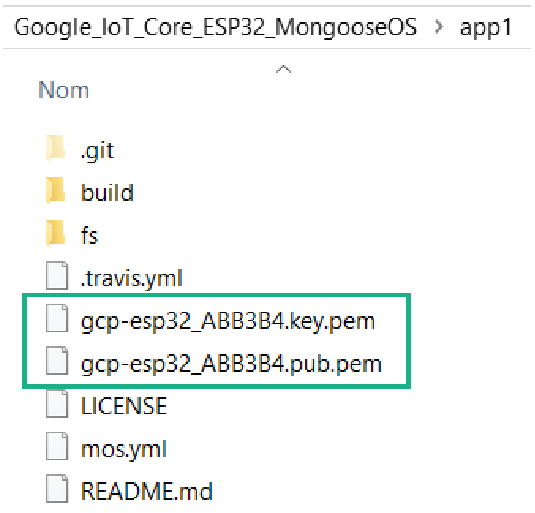
```
gcloud beta pubsub topics create weather-telemetry-topic

# Create a Pub/Sub subscription to the just created topic
gcloud beta pubsub subscriptions create --topic weather-telemetry-topic weather-telemetry-subscription

# Create devices registry (we call it weather-devices-registry)
# Precise Pub/Sub topic name for event notifications
# Disallow device connections to the HTTP bridge
gcloud beta iot registries create weather-devices-registry --region europe-west1 --no-enable-http-config --event-notification-config=topic=weather-telemetry-topic
# Say 'yes' to enable API (if prompted).
# But the last command may not work all the same
# if you don't enable billing.
# So, follow the link to enable billing and retry last command.
# It should end up to "Created registry [weather-devices-registry]."
```

Device registration within the Cloud IoT Core project

Let’s now register the devices to the project! One at a time of course. mos tool is really helpful for this task. From mos tool launched in its UI or from Command Line Terminal, placed in our app1 folder, we type the following command (project id and registry name are involved, as you see): # Register device with Cloud IoT Core

```
mos gcp-iot-setup --gcp-project lab22-260712 --gcp-region europe-west1 --gcp-registry weather-devices-registry
```

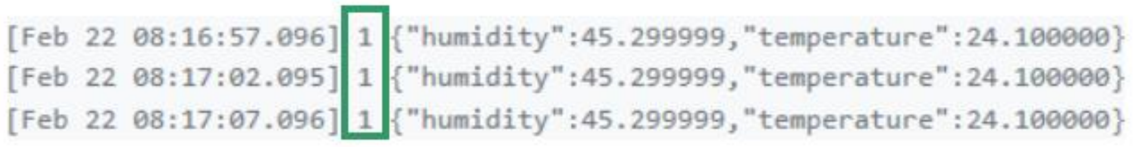
This command is a mos command that will itself use gcloud commands. The device about to be registered must be connected via the serial port to our host computer because some information will be uploaded to it just like keys, MQTT bridge address, etc. Indeed, we see on mos console that two keys (one private, one public) are generated. We can inspect them in app1 project folder. The private one is for ESP32 and the public one is for Google IoT Core. They are used during the authentication process involving the JSON Web Token we mentioned earlier. [108]



```
Google_IoT_Core_ESP32_MongooseOS > app1
Nom
.git
build
fs
.travis.yml
gcp-esp32_ABB3B4.key.pem
gcp-esp32_ABB3B4.pub.pem
LICENSE
mos.yml
README.md
```

Fig.6.22: MQTT returns 1

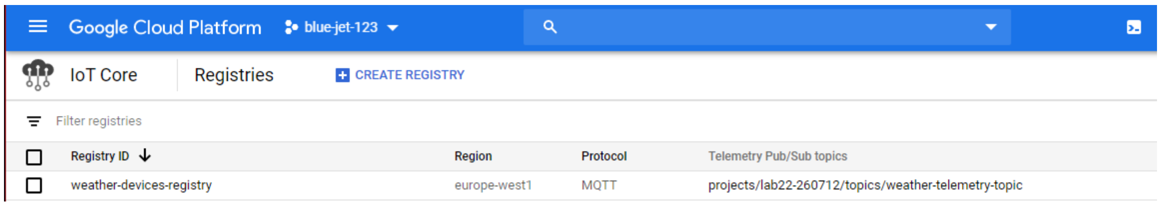
When the device reboots, we see in the console that it successfully connects to the Google MQTT bridge and publishes telemetry messages (MQTT.pub() returns 1):



```
[Feb 22 08:16:57.096] 1 {"humidity":45.299999,"temperature":24.100000}
[Feb 22 08:17:02.095] 1 {"humidity":45.299999,"temperature":24.100000}
[Feb 22 08:17:07.096] 1 {"humidity":45.299999,"temperature":24.100000}
```

Fig.6.23: mos console keys

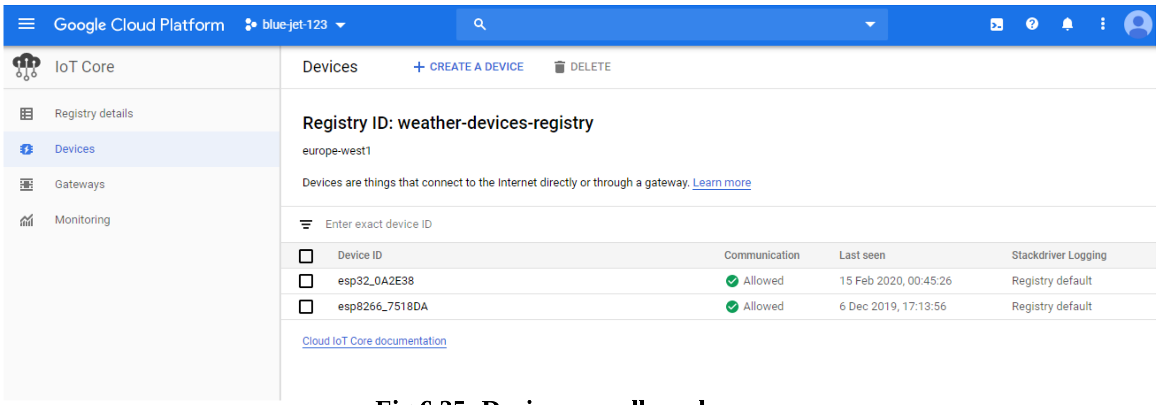
Checking the project setup in Google Cloud Console



Registry ID	Region	Protocol	Telemetry Pub/Sub topics
weather-devices-registry	europe-west1	MQTT	projects/lab22-260712/topics/weather-telemetry-topic

Fig.6.24: Registry ID created

Clicking on the Registry ID weather-devices-registry reaches another screen. Clicking on "Devices" on this new screen lists provisioned devices and gives details like the last time they were seen (but this is not a live update, we have to refresh the page):



Google Cloud Platform · blue-jet-123

IoT Core — Devices · + CREATE A DEVICE · DELETE

Registry details · Devices · Gateways · Monitoring

Registry ID: weather-devices-registry

europe-west1

Devices are things that connect to the Internet directly or through a gateway. Learn more

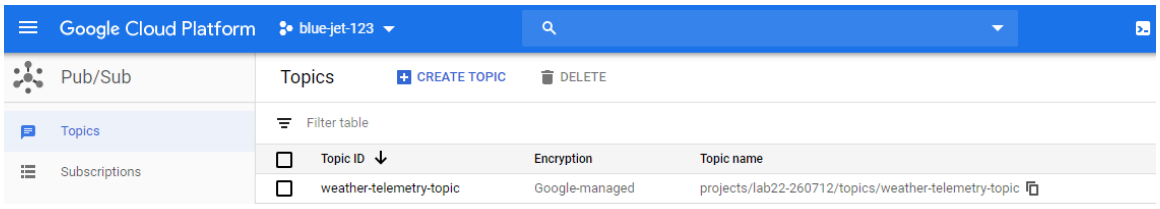
Enter exact device ID

Device ID	Communication	Last seen	Stackdriver Logging
esp32_0A2E38	✓ Allowed	15 Feb 2020, 00:45:26	Registry default
esp8266_7518DA	✓ Allowed	6 Dec 2019, 17:13:56	Registry default

Cloud IoT Core documentation

Fig.6.25: Devices are allowed

Clicking on the Telemetry Pub/Sub topic name goes to Pub/Sub console to show the subscription we created before, i.e. the one related to the telemetry topic:



Google Cloud Platform · blue-jet-123

Pub/Sub — Topics · CREATE TOPIC · DELETE

Topics · Subscriptions

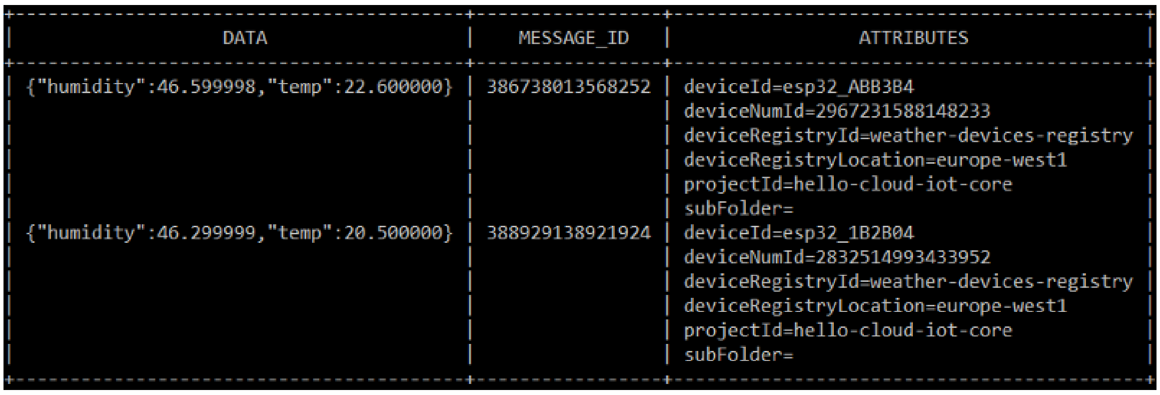
Filter table

Topic ID	Encryption	Topic name
weather-telemetry-topic	Google-managed	projects/lab22-260712/topics/weather-telemetry-topic

Fig.6.26: Pub/Sub topic subscribtion

Viewing at last some telemetry data

gcloud beta pubsub subscriptions pull --auto-ack weather-telemetry-subscription --limit=2



```
+----------------------------------------+-----------------+--------------------------------------------+
|                  DATA                  |   MESSAGE_ID    |                 ATTRIBUTES                 |
+----------------------------------------+-----------------+--------------------------------------------+
| {"humidity":46.599998,"temp":22.600000} | 386738013568252 | deviceId=esp32_ABB3B4                      |
|                                        |                 | deviceNumId=2967231588148233               |
|                                        |                 | deviceRegistryId=weather-devices-registry |
|                                        |                 | deviceRegistryLocation=europe-west1        |
|                                        |                 | projectId=hello-cloud-iot-core             |
|                                        |                 | subFolder=                                 |
| {"humidity":46.299999,"temp":20.500000} | 388929138921924 | deviceId=esp32_1B2B04                      |
|                                        |                 | deviceNumId=2832514993433952               |
|                                        |                 | deviceRegistryId=weather-devices-registry |
|                                        |                 | deviceRegistryLocation=europe-west1        |
|                                        |                 | projectId=hello-cloud-iot-core             |
|                                        |                 | subFolder=                                 |
+----------------------------------------+-----------------+--------------------------------------------+
```

Fig.6.27: humidity and temperature readings

Logging, storing and visualizing weather data with Firebase

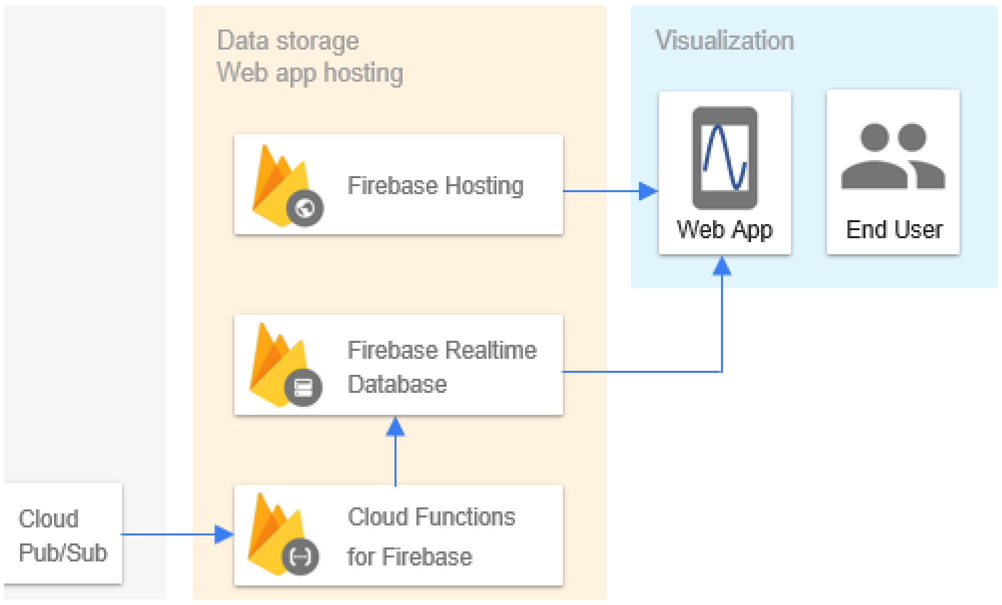
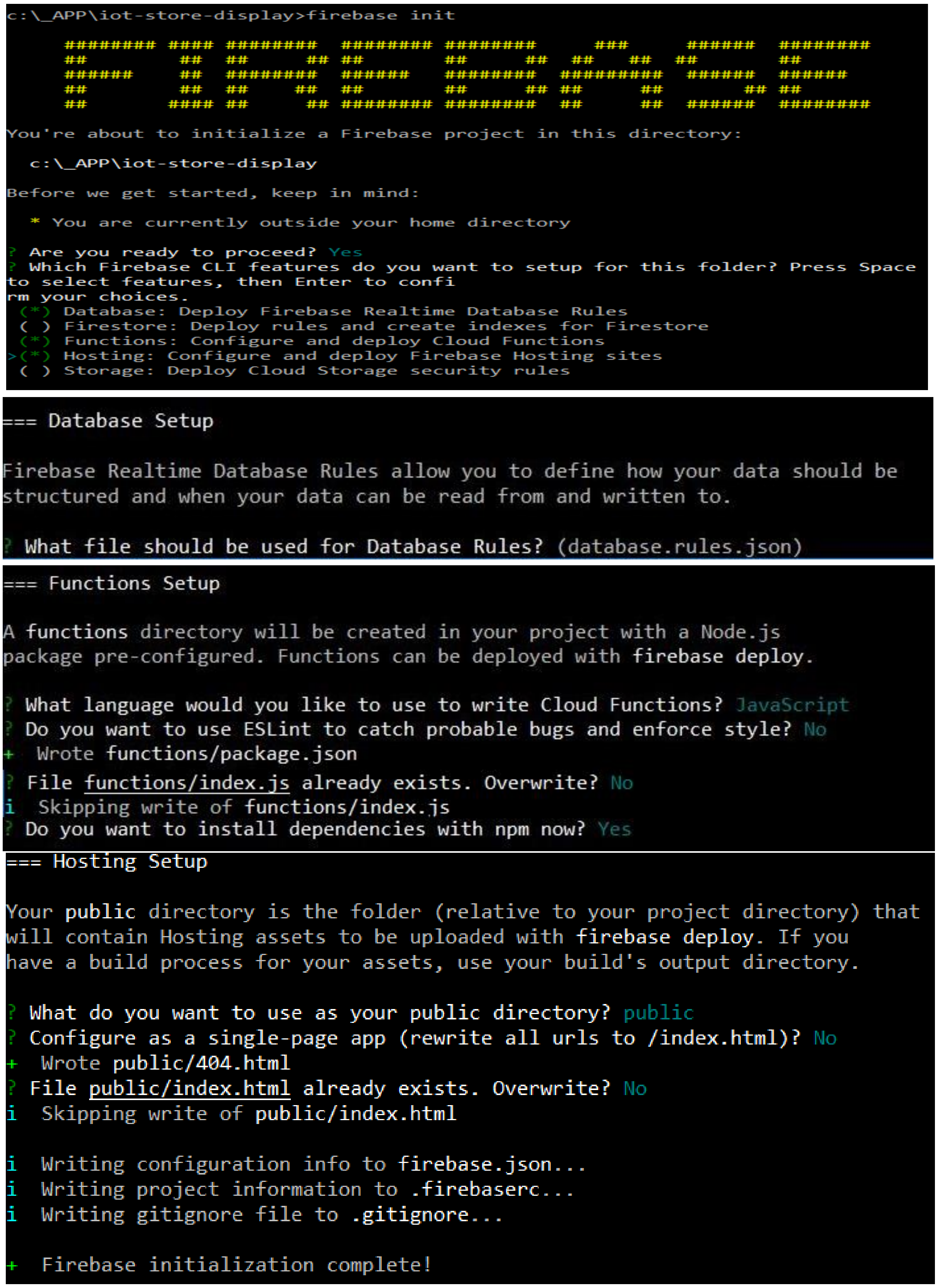


Fig.6.28: Firebase products

On that diagram, we see that our project needs 3 Firebase products: A Firebase Cloud Function (more exactly "a Cloud Function for Firebase") must react to any publication to the telemetry topic in order to store the weather data of this publication to a Firebase Realtime Database. This storage allows weather data persistence and is used to feed a web app hosted by Firebase Hosting. This web app draws live plots of this weather data across time. [108]



```
c:\_APP\iot-store-display>firebase init

     ######## #### ########  ######## ########     ###     ######  ########
     ##        ##  ##     ## ##       ##     ##  ##   ##  ##       ##
     ######    ##  ########  ######   ########  #########  ######  ######
     ##        ##  ##    ##  ##       ##     ## ##     ##       ## ##
     ##       #### ##     ## ######## ########  ##     ##  ######  ########

You're about to initialize a Firebase project in this directory:

  c:\_APP\iot-store-display

Before we get started, keep in mind:

  * You are currently outside your home directory

? Are you ready to proceed? Yes
? Which Firebase CLI features do you want to setup for this folder? Press Space
to select features, then Enter to confi
rm your choices.
 (*) Database: Deploy Firebase Realtime Database Rules
 ( ) Firestore: Deploy rules and create indexes for Firestore
 (*) Functions: Configure and deploy Cloud Functions
>(*) Hosting: Configure and deploy Firebase Hosting sites
 ( ) Storage: Deploy Cloud Storage security rules
```

```
=== Database Setup

Firebase Realtime Database Rules allow you to define how your data should be
structured and when your data can be read from and written to.

? What file should be used for Database Rules? (database.rules.json)
```

```
=== Functions Setup

A functions directory will be created in your project with a Node.js
package pre-configured. Functions can be deployed with firebase deploy.

? What language would you like to use to write Cloud Functions? JavaScript
? Do you want to use ESLint to catch probable bugs and enforce style? No
+  Wrote functions/package.json
? File functions/index.js already exists. Overwrite? No
i  Skipping write of functions/index.js
? Do you want to install dependencies with npm now? Yes
```

```
=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? public
? Configure as a single-page app (rewrite all urls to /index.html)? No
+  Wrote public/404.html
? File public/index.html already exists. Overwrite? No
i  Skipping write of public/index.html

i  Writing configuration info to firebase.json...
i  Writing project information to .firebaserc...
i  Writing gitignore file to .gitignore...

+  Firebase initialization complete!
```

Fig.6.29: Firebase initiation

6.6.1.7 Hardware

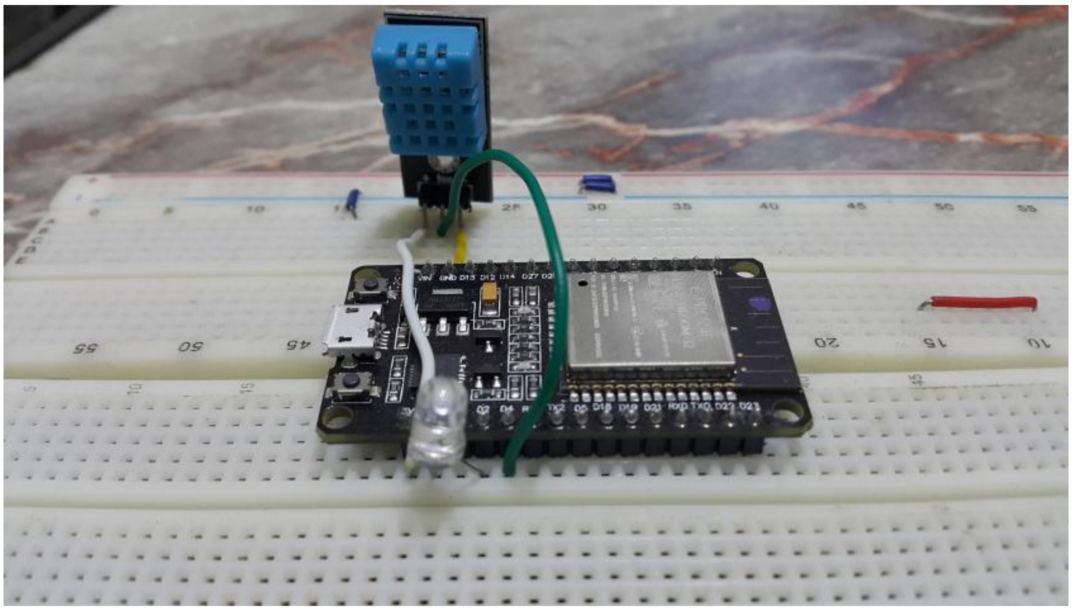
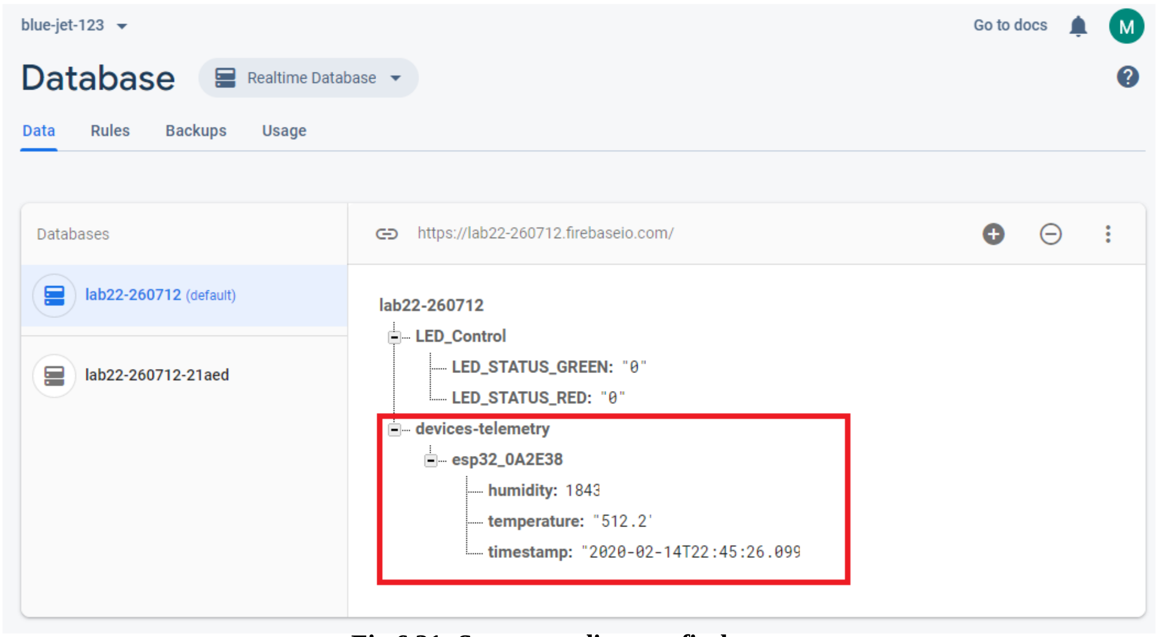


Fig.6.30: DHT Hardware

6.6.1.8 After deploying to firebase



```
lab22-260712
  LED_Control
    LED_STATUS_GREEN: "0"
    LED_STATUS_RED: "0"
  devices-telemetry
    esp32_0A2E38
      humidity: 1843
      temperature: "512.2'
      timestamp: "2020-02-14T22:45:26.099
```

Fig.6.31: Sensor readings on firebase

6.6.2 Sending commands from unity to firebase to control a hardware as Switch

In this project we used ESP 82666 NodeMCU to make things way easier. [109]

6.6.2.1 Code written on Arduino IDE

```
#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>
// Set these to run example.
```

```
#define FIREBASE_HOST "lab22-260712.firebaseio.com"
// the project name address from firebase id
#define FIREBASE_AUTH "9fYKbOLrzPi6K6G31m1UglfieWUvD2DXROSFG47f"
// the secret key generated from firebase
#define WIFI_SSID "*********"
// input your home or public wifi name
#define WIFI_PASSWORD "********"
//password of wifi ssid
#define green "LED_Control/LED_STATUS_GREEN"
#define red "LED_Control/LED_STATUS_RED"

void setup() {
Serial.begin(9600);
pinMode(D1, OUTPUT);
pinMode(D3, OUTPUT);

// connect to wifi.
WiFi.setSleepMode(WIFI_NONE_SLEEP);
WiFi.mode(WIFI_STA);
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
}
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
}
String n = "0" ;
String m = "0" ;
void loop() {
// get value
n = Firebase.getString(green);
if (n == "1") {
digitalWrite(D1,HIGH);
delay(1000);
  }
else {
digitalWrite(D1,LOW);
}
m = Firebase.getString(red);
// handle error
if (m == "1") {
digitalWrite(D3,HIGH);
delay(1000);
  }
else {
digitalWrite(D3,LOW);
}
}
```

6.6.6.2 Hardware

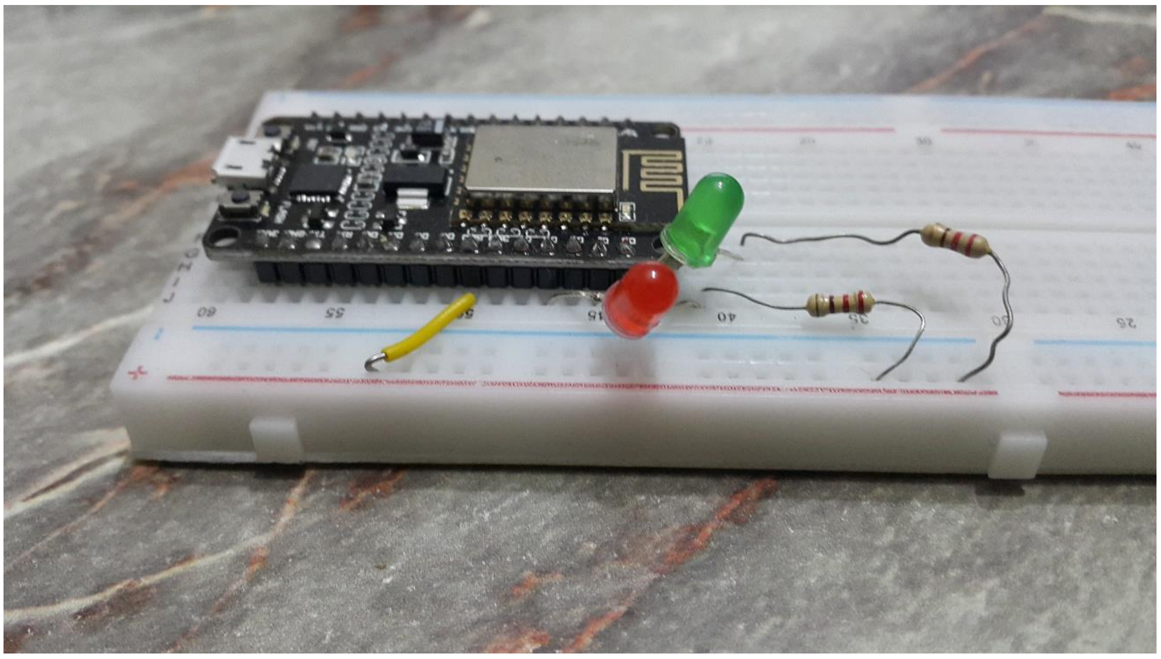
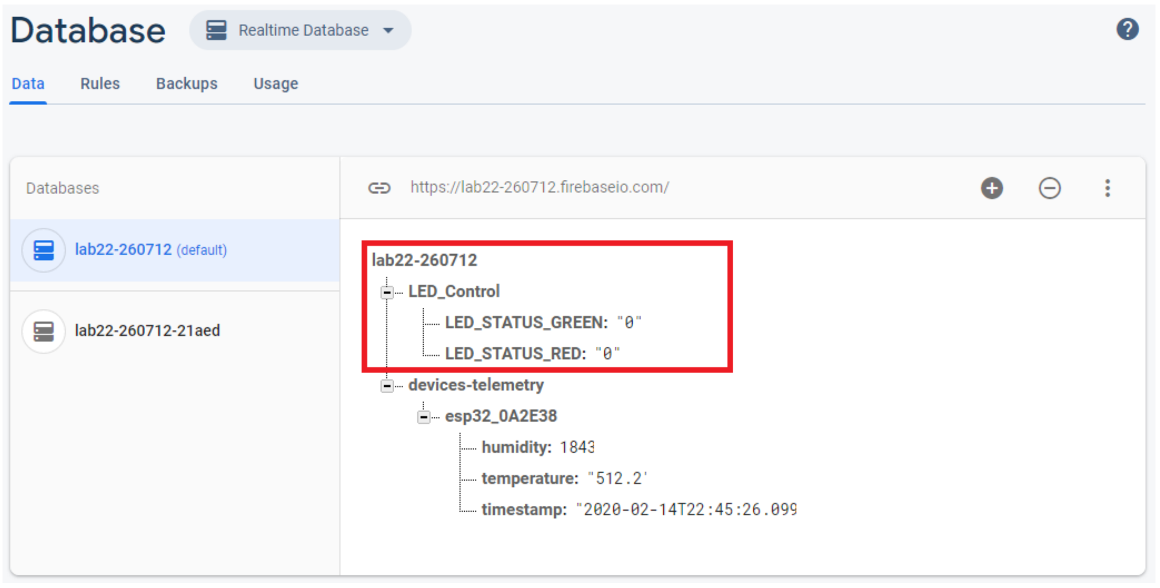


Fig.6.32: Firebase switch hardware

6.6.6.3After deploying to firebase



```
lab22-260712
  LED_Control
    LED_STATUS_GREEN: "0"
    LED_STATUS_RED: "0"
  devices-telemetry
    esp32_0A2E38
      humidity: 1843
      temperature: "512.2'
      timestamp: "2020-02-14T22:45:26.099
```

Fig.6. 33: Firebase switch

6.7 The actual system of our project:

The first test project humidity sensor would have been replaced with the motor's sensors and it would act as an eye for our actual system through the digital twin model for predictive maintenance.

The second test project, however, would have acted as a remote-control switch to turn on/off our system remotely in case of maintenance or faults