Full Stack IOT Cloud Application

# 1. Introduction

The theme and basic function of full stack IoT cloud applications is to keep track of sensor data. IoT stands for Internet of Things, is a group of physical devices connected over the network to exchange sensor data. IoT devices are used in homes, offices and industrial places. There are 10 billion connected IoT devices according to 2020 data and it is expected that these devices would reach 22 billion by 2025 [1].

The concept which makes IoT powerful is the communication inside the IoT network. There are 4 different types of communication models which IoT uses to communicate.

1. Device to Device Communications
2. Device to Cloud Communications
3. Device to Gateway Model
4. Backend Sharing Model

The device to device communication model consists of two or more IoT devices which communicate with each other without third party device or service. These devices can use simple IP protocol, bluetooth, Z-Wave or ZigBee for communication [2]. Generally in this communication between devices, data rate is very small. This model is used in home automation for devices like light bulbs, smoke sensors, door sensors or thermostats.

In device to cloud applications, devices don’t directly communicate with each other, they communicate with the cloud. For this purpose, they have to connect to the internet. Devices can be connected to the internet using wifi or ethernet cable. The process of this communication model can be illustrated visually by Figure 1.

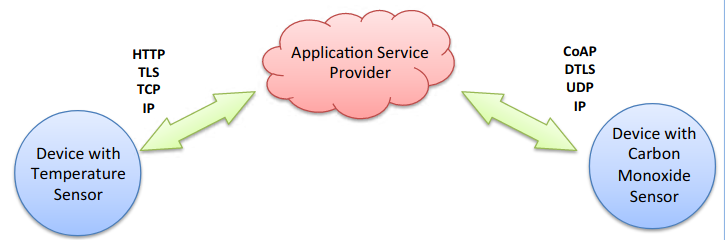


Figure 1: Device to Cloud Communication Model [3]

In the device to gateway model, devices don’t directly communicate with the cloud, but they communicate with a software which then communicates with the cloud. The benefit of this method is that it provides extra security to IoT devices. It is a general case that devices and cloud don’t follow the same protocol to communicate, in these cases, these gateway softwares translate device protol to that protocol which is supported by the cloud. One example of this model is to communicate with a smartphone which then would communicate with the cloud.

In the backend sharing model, the devices share their data to the cloud which then makes it available to the end users. Devices can also send data directly to the end users. Different analysis algorithms with the usage of machine learning and statistical methods can detect patterns which can be helpful for user understanding. This model can further understood by the following diagram

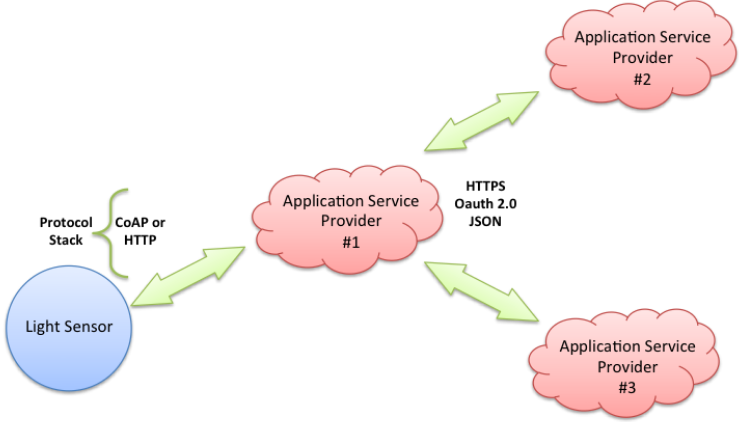


Figure 2: Backend Data Sharing Model [3]

For the purpose of this application, a backend sharing model is used. Data from different types of sensors is pushed to the cloud and made available to the user with a web application. Users can see previous and current data from the application from anywhere. They can draw conclusions by looking at the track of value changes of sensors which can help to better understand the problems in the system and give a reason to solve them.

# 2. Cloud Architecture Diagram

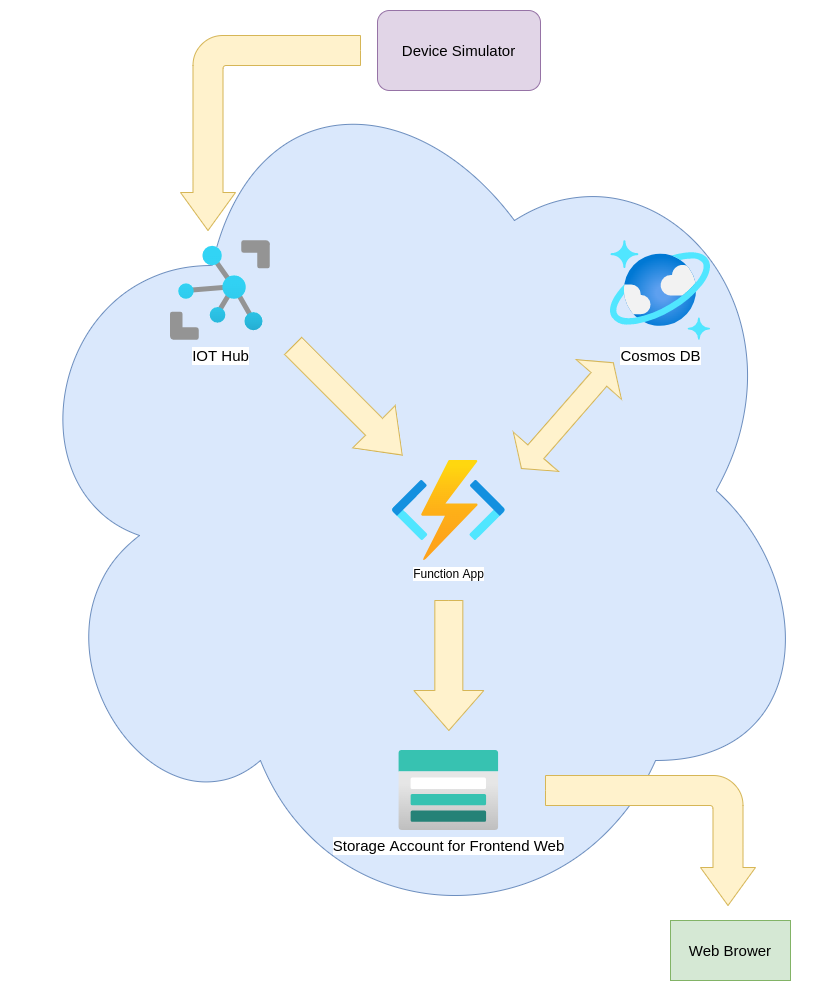


Figure 3: Cloud Architecture Diagram

# 3. Cloud Services

Azure is a platform for development and deployment of different cloud applications. There are around 50 Azure cloud data centers around the world. It offers many services like analytics, storage, networking and computation. It offers different tools like databases, virtual machines, web, Mixed Reality and blockchain which are helpful in these services.

In 2008, Microsoft announced the introduction of Windows Azure and made preview versions available. The first commercial launch was in 2010. The earlier versions of Azure were not good compared to its competitors like AWS. The main reason behind that was Azure at that time supported only windows frameworks. By time, the management at Microsoft realized that cloud computing is not limited to windows and there is a whole new world. So in 2014, Windows Azure was rebranded as Microsoft Azure offering all types of operating system which are important for cloud [4].

Azure also offers some IoT specific services which make IoT development and deployment easier. Services used in building full-stack iot cloud applications with their purpose and use is given below.

## 3.1. IoT Hub

IoT Hub is a recommended tool by Microsoft for the connection of IoT devices to the cloud. They are capable of generating a huge number of events per day which can be very important in integration with other services like Stream Analytics, Databricks, and HDInsight. IoT uses Event Hub to keep records of reading from sensors.

To transmit and queue data, IoT Hub uses many protocols. Some of these protocols are HTTPS, AMQP, MQTT and web sockets [5]. Other protocols can also be used but for them protocol conversion needs to be done. Protocol conversion can also be done in Microsoft Azure by using Azure IoT Edge or deploying custom Azure IoT protocol gateway. Custom protocol gateway can be deployed by using the services like Azure Service Fabric, Azure Cloud Services Worker roles or windows virtual machines [5].

It is necessary to understand the builtin protocols which IoT hubs use for communication. HTTPS is on top of them all. HTTPS is the advanced version of HTTP which is the hypertext transfer protocol. Almost all websites use HTTP/HTTPS for browsing. HTTP is an application layer protocol for transmitting html documents [6]. It was originally designed for communication between server and a web application. But this protocol is used in other applications too, IoT hub is one example.

HTTPS is the abbreviation of hypertext transfer protocol secure. It adds security to the HTTP protocol by encrypting the document it shares. Due to its security benefit, almost all popular browsers recommend using HTTPS instead of HTTP. The reason is that HTTP data would be plain and anybody can read it who is snooping on traffic. Which is a great security hazard. HTTPS apply encryption using TLS/SSL certificates [7].

Advanced Message Queuing Protocol also known as AMQP is another important protocol for communication with IoT Hub. There is a publisher and a consumer in this protocol. The publisher sends a message and the consumer consumes it. There is a broker in between them whose responsibility is to send messages to the right consumer. Broker which is generally a server uses two important components: exchange and queue. Publisher sends a message to the broker where It is saved in the exchange and the consumer requests data from the broker which transfers data from the queue to the consumer. Then data from the exchange is passed to the queue. This is the same cycle AMQP follows again and again for communication [8]. This process can be further explained by the following diagram.

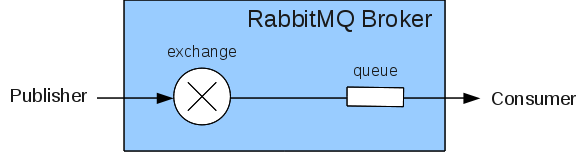


Figure 4: AMQP protocol diagram [8]

MQTT is another protocol which IoT hub uses for messaging. It is the standard messaging protocol used for IoT. It is a lightweight publish/subscribe message transport which is ideal for device communication where a very light data bandwidth is required.

Iot Hub gives a landing spot to the IoT devices which means IoT messages first go to the IoT Hub. It is the central point which controls devices by storing and synchronizing data. It also enables querying of device meta dta and state information using device twins. It can be used to set the state of one or multiple IoT devices which have the same characteristics. Automatic responses can also be generated to device reported states using message routing integration.

Monitoring device identity operations, device telemetry and diagnostics, cloud-to-device commands, and connections determines the IoT Hub's status. To connect to the IoT Hub, each device has its own security key. Each device can be individually whitelisted or blacklisted, giving complete control over device access. Device applications can read and receive notifications when desired properties in the device twin in the IoT Hub change. The IoT solution backend can change the desired properties. Device applications can modify reported properties in the device twin, and changes are read by the backend [5].

The IoT Hub can also be viewed as a gateway to several key Microsoft backend cloud-based services. Among the services offered are

* Azure Stream Analytics
* Azure Time Series Insights
* Azure Databricks
* Apache Spark
* Azure Functions
* Azure Logic Apps

Azure Functions can be used to create and deploy actions that contain custom code written in C#, Python, JavaScript or Java. Azure Logic Apps are a set of predefined actions that can be orchestrated using a graphical user interface-based development environment. Both of these workloads are deployed as serverless workloads.

## 3.2. Cosmos Database

Nowadays applications must be highly available and constantly available to be successful.

In order to achieve low latency and high availability, instances of these applications must be placed in data centers near their customers. Applications must adapt to massive changes in demand in real time during peak hours, store ever-increasing amounts of data, and make this data available to users in milliseconds.

Azure Cosmos DB is a NoSQL database that is completely managed for contemporary app development. Speed at any scale is guaranteed by single-digit millisecond response rates and automated and rapid scaling. SLA-backed availability and enterprise-grade security ensure business continuity. Because of turnkey multi-region data dissemination everywhere in the globe, open source APIs and SDKs for major languages, app development is faster and more productive. Azure Cosmos DB, as a fully managed service, relieves database administration by automating management, updates, and patching. It also manages capacity using cost-effective serverless and automated scaling solutions that adapt to application requirements to match capacity with demand [9].

Cosmos DB assures high availability, high throughput, low latency, and configurable consistency which benefit mobile, web, or IoT applications. As they have to manage enormous volumes of data, reads, and writes with high speed for a variety of data. Cosmos DB can be used to construct IoT and telematics apps, retail and marketing applications, gaming applications, and online and mobile applications.

Some features Cosmos DB provide out of the the box are following

* A very low latency of fewer than 10 milliseconds while receiving data and less than 15 milliseconds when writing data is practically assured.
* Because data is automatically indexed, users can access it using any API of their choosing. SQL, Gremlin, JavaScript, Azure Table Storage, and MongoDB may be used to see their data.
* It provides a comprehensive solution that is Azure-powered and can be automatically duplicated in data centers around the world.
* It employs five distinct degrees of consistency: bounded staleness, strong, session, eventual, and consistent-prefix.

## 3.3. Functions App

Azure Functions is a serverless tool for which less code is required. It is a great tool which enables us to control less infrastructure, and save cost. Instead of constantly worrying about deploying and maintaining servers, the cloud architecture delivers all of the current resources required to keep apps running.

Efficients systems are required to respond to a succession of crucial events. Whether creating a web API, responding to database updates, processing IoT data streams, or even managing message queues, every application needs a mechanism to run code when certain events happen. Azure Functions delivers "compute on-demand" in two important ways to address this need.

1. Azure Functions convert system's logic into easily accessible units of code. These code chunks are referred to as "functions." Different functions might be activated to respond to a vital occurrence.
2. As the number of requests grows, Azure Functions scales to match the demand with as many resources and function instances as are required - but only while they are required. As demands decrease, any surplus resources and application instances are immediately decommissioned [10].

A single Azure Function execution has a maximum execution time of 5 minutes by default. If the Function runs longer than the maximum timeout, the Azure Functions runtime provides the ability to terminate the process at any point after the maximum timeout has been reached. This execution time can also be changed to increase from 5 minutes.

Azure function supports a variety of programming languages depending upon the runtime version. In general Azure functions support C#, JavaScript, Java, Python, TypeScript and Power shell.

When events from other services trigger, Azure Functions can be invoked. Because it is event-driven, the application platform may execute code that is triggered by events in any third-party service or on-premise system. Azure Functions also provide key templates to deal with different situations. These templates are following

* **HTTPTrigger** - Uses an HTTP request to initiate the execution of code.
* **TimerTrigger** - Run activities on a set schedule.
* **CosmosDBTrigger** - Process Azure Cosmos DB documents when they are added or changed in a NoSQL database collection.
* **BlobTigger -** Process Azure Storage blobs when they are introduced to containers. This can be used to resize images.
* **QueueTrigger -** Respond to messages as they come in an Azure Storage queue.
* **EventGridTrigger -** Responds to events supplied to an Azure Event Grid subscription. Filtering is supported as part of a subscription-based mechanism for receiving events. A solid choice for creating event-driven structures.
* **EventHubTrigger -** Respond to events supplied to an Azure Event Hub. Application instrumentation, user experience or workflow processing, and internet-of-things (IoT) situations are all very beneficial.
* **ServiceBusQueueTrigger -** Listen to message queues to connect code to other Azure or on-premises services.
* **ServiceBusTopicTrigger -** By subscribing to topics, connect code to other Azure or on-premises services.
* **IoTHubTrigger -** Listen to messages and trigger code whenever new messages come to the IoT hub [11].

For the purpose of this application, Function App is configured with two functions. One function listens to IoT hub messages and the code of this function is executed whenever a new message is passed to the Iot Hub. The other function that is added to the Functions App is HttpTrigger function. The code of this function is executed whenever http request is given.

## 3.4 Static Storage Account

All of Azure Storage data objects, including blobs, file shares, queues, tables, and discs, are stored in an Azure storage account. The storage account creates a distinct namespace for Azure Storage data, which is accessible through HTTP or HTTPS from anywhere in the globe. Data in storage accounts are long-lasting and highly accessible, as well as secure and enormously expandable.

A storage account in Azure gives data a unique namespace. Every item saved to Azure Storage has a unique address that includes the account name. The endpoints for storage accounts are formed by the account name and the Azure Storage service endpoint.

Static material (HTML, CSS, JavaScript, and picture files) can be served straight from the $web storage container. By hosting content in Azure Storage, serverless architectures such as Azure Functions and other Platform as a Service (PaaS) technologies can be leveraged. Azure Storage static website hosting is an excellent solution for simple HTML, CSS and JavaScript websites [12].

# 4. Development

By looking at Figure 3, it can be seen that the device simulator is the IoT end device which communicates with the IoT hub by sending messages. The Functions App has two functions, the purpose of one function is to listen to the messages on the IoT hub and save messages on Cosmos DB. The other function listens to HTTP requests and on HTTP requests it extracts data from the Cosmos DB and sends it back in JSON format. Static storage account is hosting static website files. This website uses Rest API’s to get data from Functions App.

## 4.1 Device Simulator

In this workflow, no real time physical devices are used to work as IoT devices. To replace these devices a python script is developed to work as a device simulator. The basic flow of this flow can be understood by Figure 5.

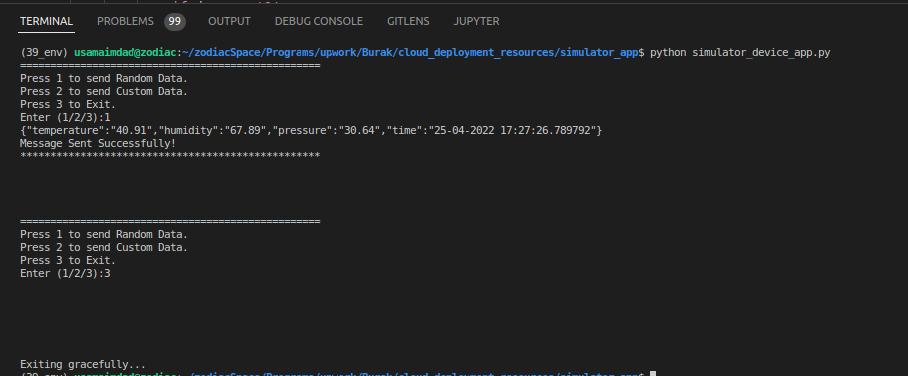


Figure 5: Simulator Device Flow

There are 3 options to select from. This simulator sends three different types of sensor data to the IoT hub along with the time stamp. These sensors are temperature, humidity and pressure. The form in which this simulator sends data to the IoT hub can be seen in Figure 5. By selecting 1 in Figure 5, the simulator randomly selects values and sends them to the IoT hub. By selecting 2, it gives the option to manually input sensor values. By selecting 3, the simulator exits gracefully.

## 4.2 IoT Hub

IoT hub is the place where the cloud receives messages from IoT devices. To create the IoT hub click on add new resource at azure portal and search for IoT Hub. Then click create button, it will redirect to another page where it will ask for a resource group and hub name.

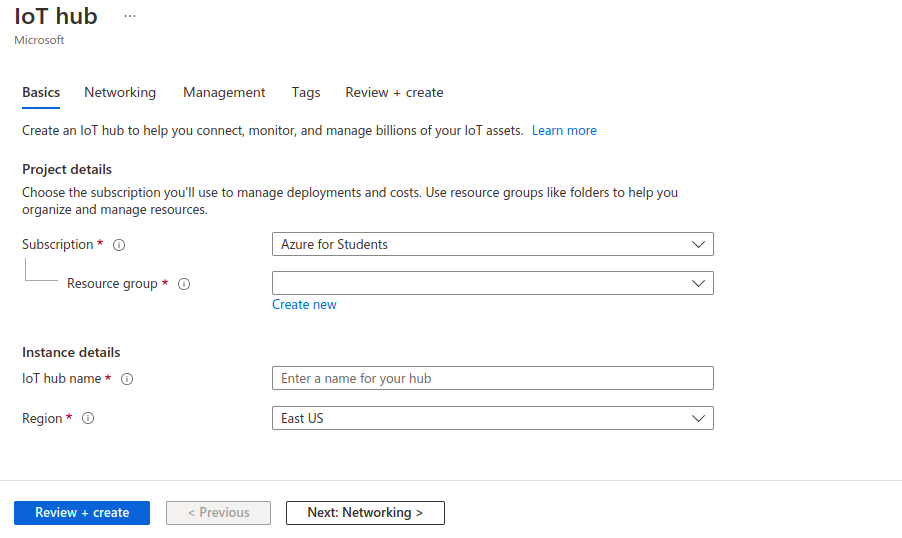


Figure 6: IoT hub creation

Select the resource group from the drop down menu and input the available hub name in its own section. Then click on “Review + create” for further process.

Once the IoT hub is created, It is now time to add the devices into the IoT hub. To create the device in the IoT hub, from the left pane click on device and then click on Add device. This will redirect to the creation page.

Look at Figure 7 for reference. Enter the device name of liking in the Device Id field and click on save to create the device.

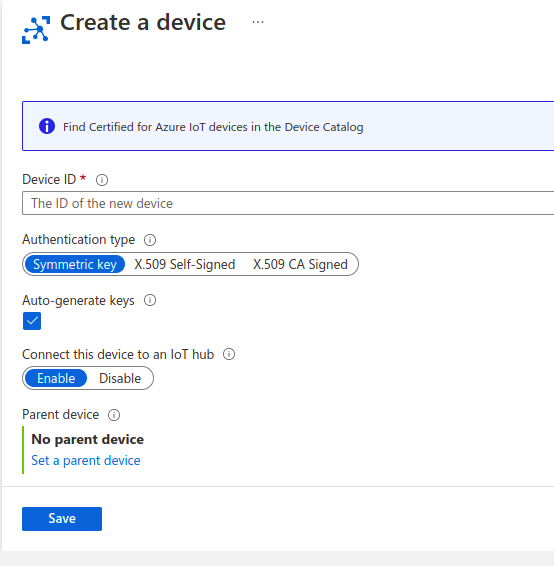


Figure 7: Device Creation

Once the device is added to the IoT hub, this is the time to use the simulator app as this device. To do this, copy the primary connection string and use it in the device simulator for connection.

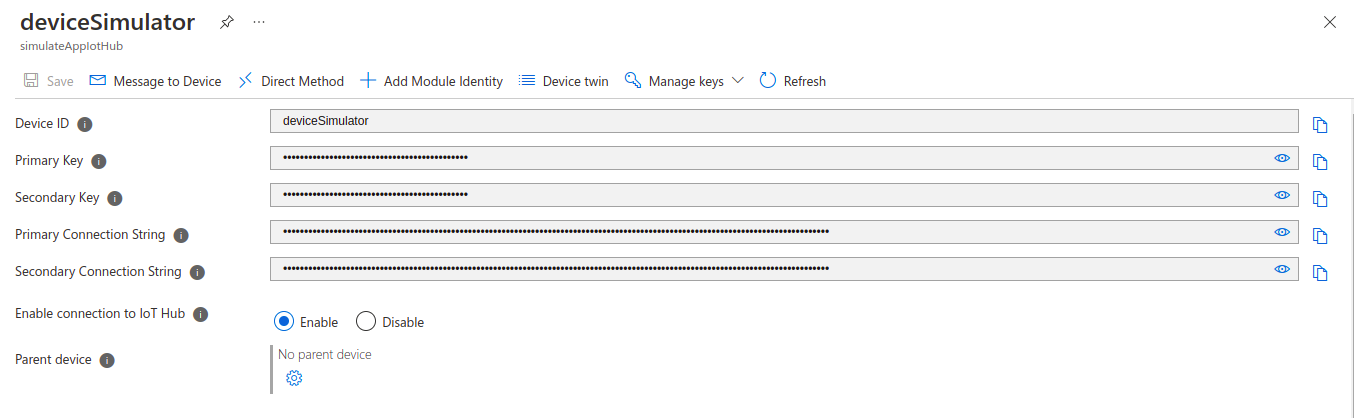


Figure 8: Device keys

## 4.3 Cosmos DB

To use the Cosmos DB in flow, it is necessary to first make a resource of the Cosmos DB. To do that, click on “Add new Resource” and search for Cosmos DB. Select Azure Cosmos DB and click on the “create” button. It redirects to a page to select the API’s which is Core (SQL) in our case.

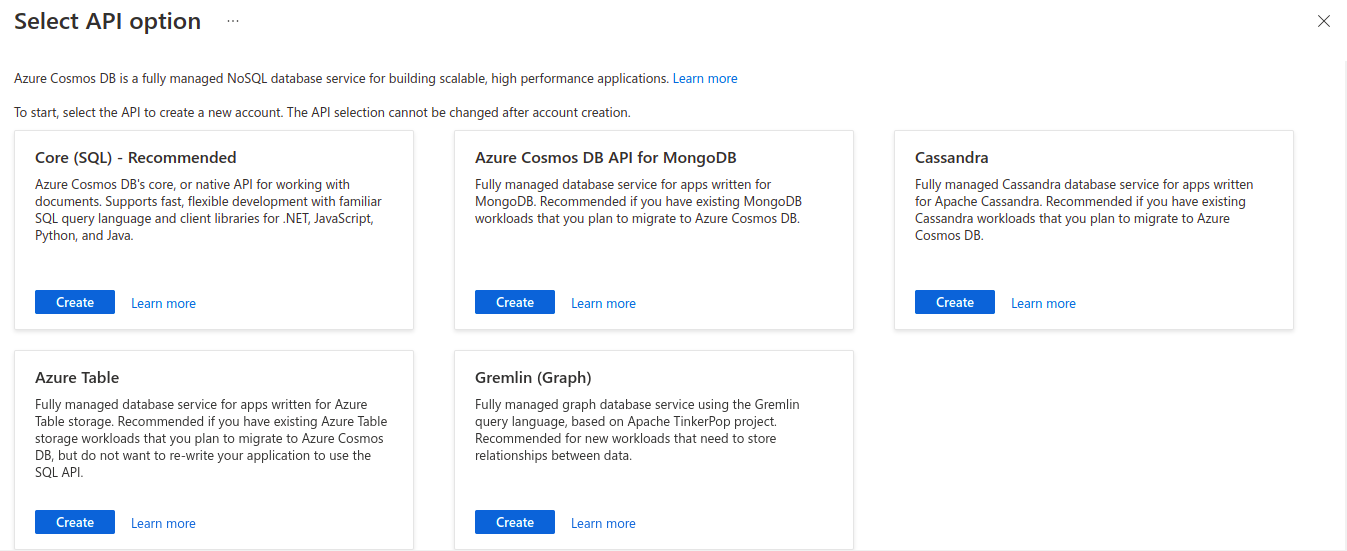


Figure 9: Select API page for Cosmos DB

Then it redirects to the SQL creation page where it is necessary to enter resource group and account name. Look at Figure 10 for reference.

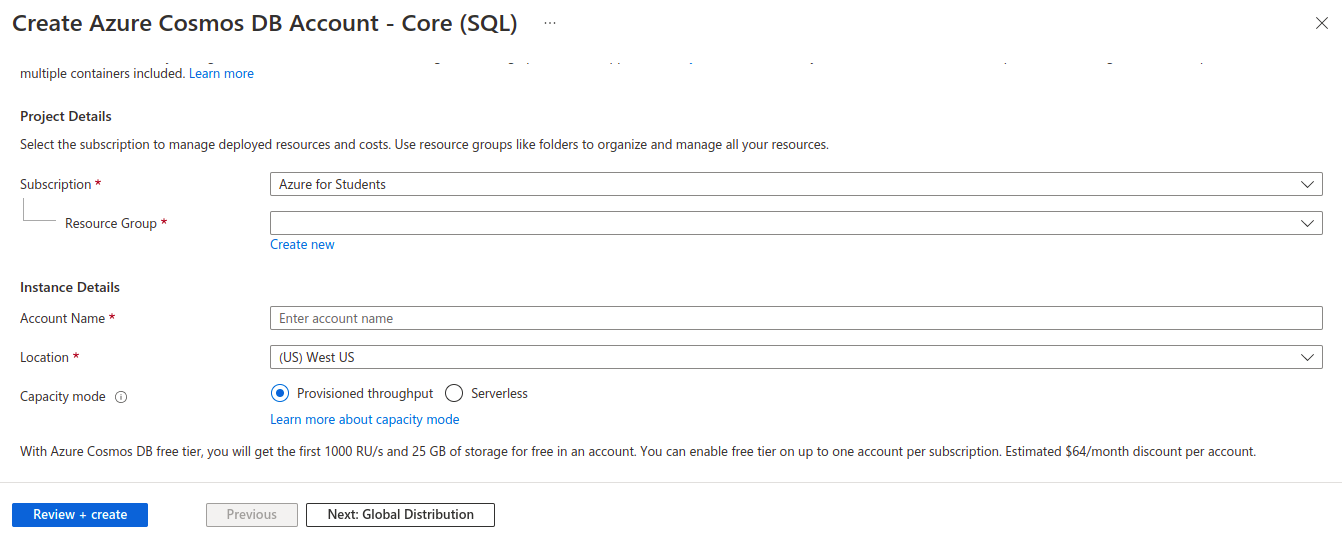


Figure 10: SQL creation page

After the creation of SQL API in Cosmos DB it is time to create a new database and add a container to that database. For this application the database name is “iot” and the container is “messages”. The structure can be further understood by looking at Figure 11.

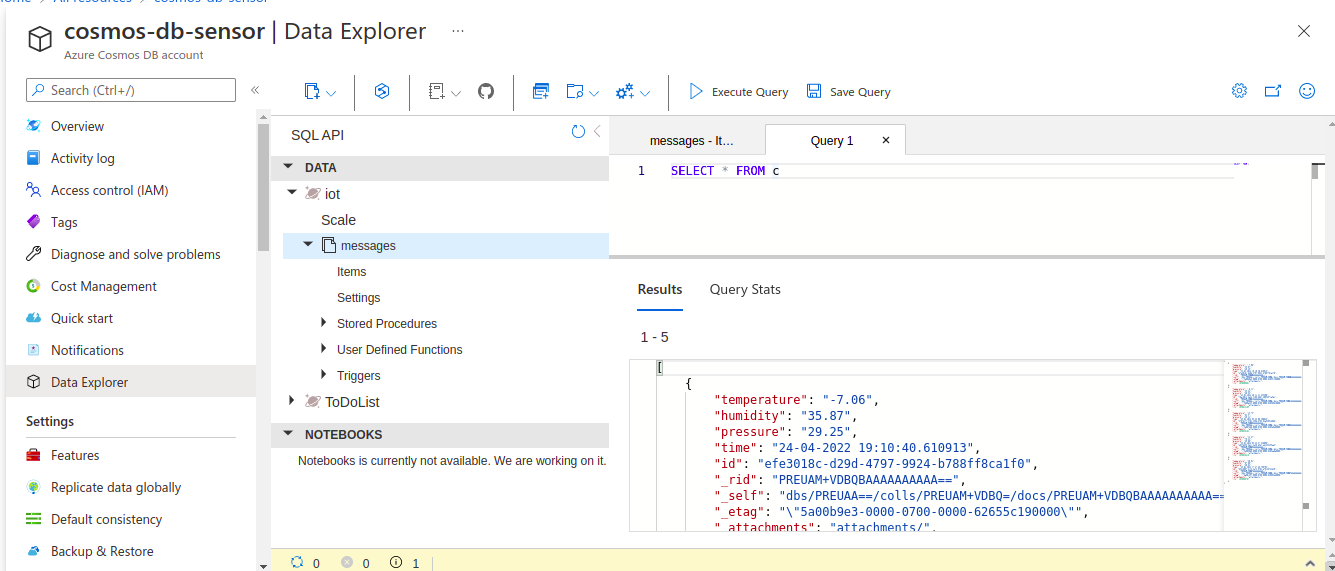


Figure 11: Cosmos DB database and container example

## 4.4 Functions App

The next step is to make the function app and add two functions in it for IoTHubTrigger and HTTP Trigger. The creation part is the same as discussed in section 4.2 and 4.3. The stack used for Function App is JavaScript and Node version is 16.

To add new functions in this function app, the prerequisite is to install Azure Function extension in Visual Studio Code. This extension add Azure logo to the side bar of vscode. By clicking on this logo we can add new functions.

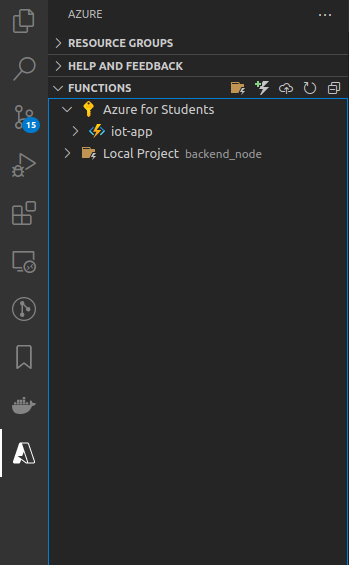


Figure 12: Azure Function App extension in vscode

This extension helps to create different kinds of functions by availing templates for them at start. Use HttpTrigger and IoTHubTrigger templates for two respective functions.

IoTHubTrigger is a function which is triggered when a new message is passed from the device simulator. This function actually saves the data to the Cosmos DB.

HttpTrigger function is triggered when an http api is called. This function extracts data from Cosmos DB and returns it in JSON format.

## 4.5 Static Storage Account

The creation of a static storage account is also not different from other resources which are created and discussed thoroughly in section 4.2 and 4.3. For the purpose of this application, a static storage account is used to host frontend files like HTML, CSS and JavaScript. So to enable it for this purpose, a small change needs to be done. To do this, on the left pane look for data management and then click on the static website. Then enable the static website on the next page. For reference look at Figure 13.

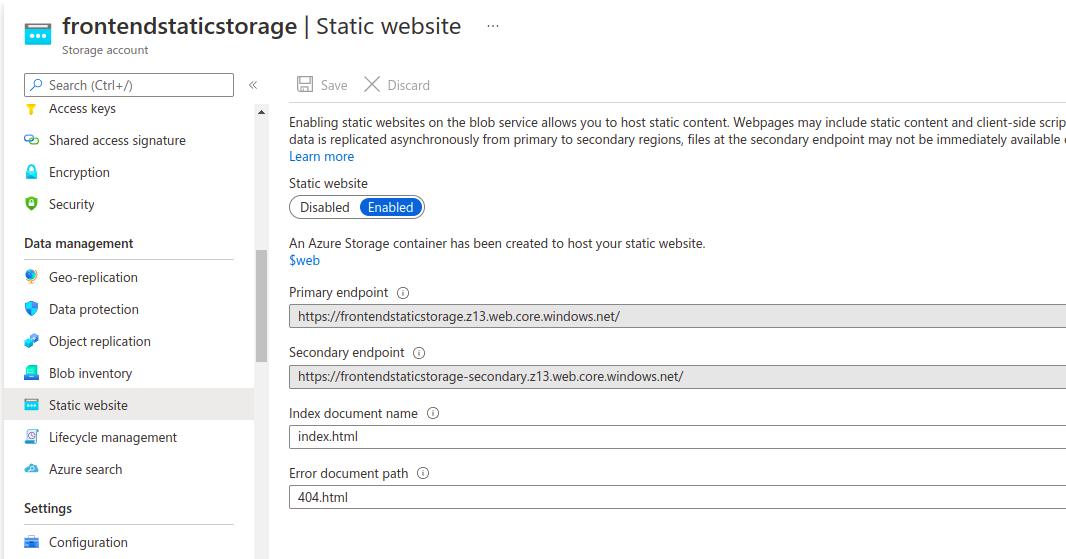


Figure 13: Static Storage Account

## 4.6 Frontend App

The frontend app is made with html, css and javascript which consist of a table and refresh button. Table shows the record of all previous sensor data with timestamp. Refresh button is the button which triggers the HttpTrigger function of the Functions app. This app uses JSON data from the HttpTrigger function and renders it into the table. The basic design of this app can be seen in Figure 14.

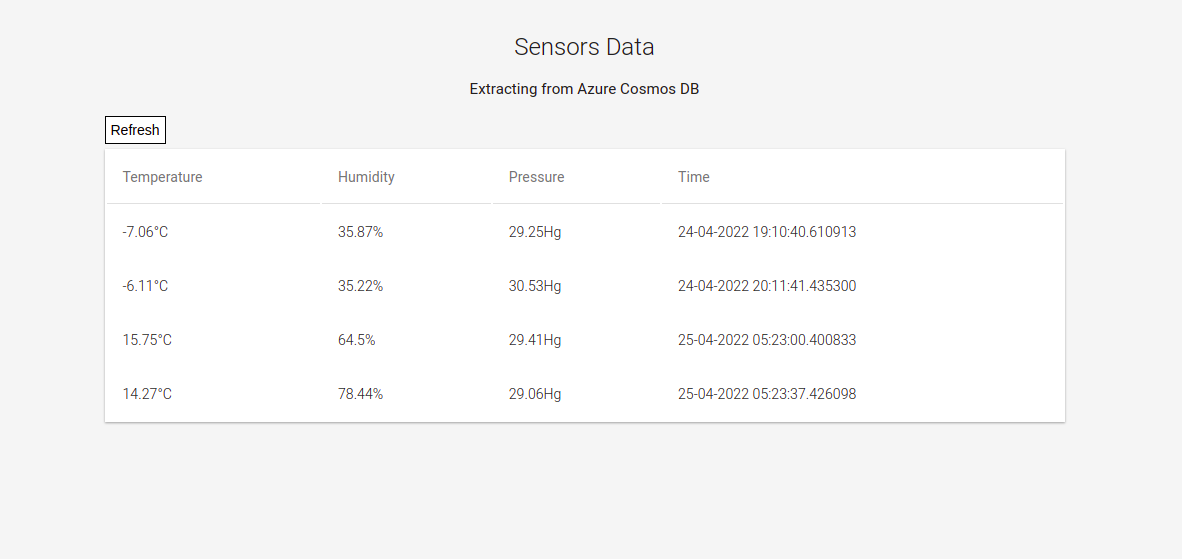


Figure 14: Frontend App

# 5. DevOps/Testing

The device simulator is developed and deployed locally on the system. It uses azure-iot-device library to make connections with the IoTHub and send messages.

For the deployment of IoT hub, add azure-iot extension to the Azure cloud. This extension can be installed by using Azure bash terminal, look at Figure 15.

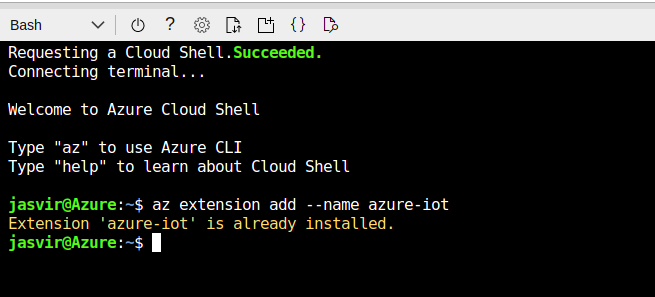


Figure 15: Installation of azure-iot extension

The command of installation of azure-iot extension is as follows.

***az extension add --name azure-iot***

When the connection is made between the simulator device and IoT hub, it is necessary to monitor the messages which are sent from the simulator device on bash terminal. This can be done by running the following command.

***az iot hub monitor-events --hub-name simulatedAppIotHub --device-id deviceSimulator***

This command monitors messages from the simulated device in the bash terminal and shows them inside the terminal.

Once the functions are developed for Function App. These functions can be deployed using Azure Functions extension by clicking on deploy button. Then vscode prompts to select the function app on which these functions need to be deployed.

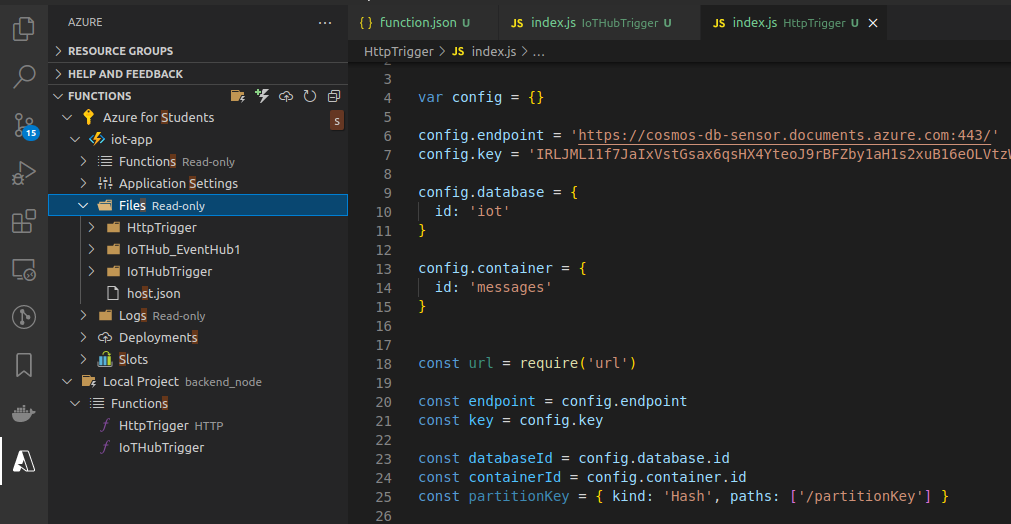


Figure 16: Deployment on Functions app

To connect to Cosmos DB copy the connection string and primary key from it and use in both functions HttpTrigger and IoTHubTrigger, as both functions save or retrieve data to or from the Cosmos DB.

The final step is to deploy the static website on the static storage account. For this purpose, there is also a Azure extension, which is called “Azure Storage”. Using this extension, static websites can be deployed to the cloud the same way Functions App is deployed.

The last step is to remove the CORS error during the Functions App http call. This can be done in Function app settings and going to CORS listed at the left panel of the portal and adding the url which should be allowed to access the API.

# 6. References

[1] <https://www.oracle.com/internet-of-things/what-is-iot/>

[2] Rose, Karen, Scott Eldridge, and Lyman Chapin. "The internet of things: An overview." *The internet society (ISOC)* 80 (2015): 1-50.

[3] Tschofenig, H., et. al., Architectural Considerations in Smart Object Networking. Tech. no. RFC 7452. Internet Architecture Board, Mar. 2015. Web.

[4] <https://www.techtarget.com/searchcloudcomputing/definition/Windows-Azure>

[5] Stackowiak, Robert. "Azure IoT solutions overview." *Azure Internet of Things Revealed*. Apress, Berkeley, CA, 2019. 29-54.

[6] <https://developer.mozilla.org/en-US/docs/Web/HTTP>

[7] <https://www.cloudflare.com/learning/ssl/what-is-https/>

[8] <https://spring.io/blog/2010/06/14/understanding-amqp-the-protocol-used-by-rabbitmq/>

[9] <https://docs.microsoft.com/en-us/azure/cosmos-db/introduction>

[10] <https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview>

[11] <https://www.serverless360.com/azure-functions>

[12] <https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blob-static-website>