Report: Sending Data from Arduino to Cloud Services

1. Introduction

Arduino boards are widely used in various Internet of Things (IoT) projects and applications. One common requirement is to send data collected by sensors or other inputs to cloud services for storage, analysis, or further processing. There are several ways to achieve this, including MongoDB, Google Firebase, MQTT, and AWS IoT services. In this report, we will explore these options and provide a detailed comparison, along with an example code for sending data from an Arduino board to MongoDB.

2. Detailed Report

MongoDB

MongoDB is a popular NoSQL database that stores data in flexible, JSON-like documents with dynamic schemas. It is designed to handle large volumes of data and provides high availability, scalability, and performance. MongoDB Atlas is a fully managed cloud database service that simplifies the deployment, operation, and scaling of MongoDB databases.

Use Cases:

- IoT applications that require storing and querying large amounts of unstructured or semi-structured data.

- Applications that need to handle frequent updates and real-time data processing.

- Scenarios where data needs to be distributed across multiple data centers or cloud regions.

How it Works:

MongoDB Atlas provides a Data API that allows you to interact with your MongoDB database using standard HTTP requests. This API supports various operations, including inserting, querying, updating, and deleting data. By sending HTTP requests with the appropriate payload, you can store data from your Arduino device in a MongoDB database hosted on Atlas.

Google Firebase

Google Firebase is a comprehensive app development platform that provides a suite of tools and services, including a real-time database, cloud functions, authentication, hosting, and more. The Firebase Realtime Database is a cloud-hosted NoSQL database that allows you to store and sync data in real-time across multiple clients.

Use Cases:

- Mobile and web applications that require real-time data synchronization across multiple clients.

- Applications that need to handle frequent updates and real-time data processing.

- Scenarios where data needs to be accessed and updated from multiple devices or locations.

How it Works:

The Firebase Realtime Database uses a WebSocket-based protocol to establish a persistent connection between the client and the database. Data is stored as JSON objects and synchronized in real-time across all connected clients. Arduino boards can communicate with the Firebase Realtime Database using HTTP or WebSocket requests.

MQTT (Message Queuing Telemetry Transport)

MQTT is a lightweight messaging protocol designed for low-bandwidth, high-latency environments. It follows a publish-subscribe model, where devices (publishers) send messages to a broker, and other devices (subscribers) receive those messages based on their subscriptions.

Use Cases:

- IoT applications that require efficient data transfer with low overhead and minimal bandwidth usage.

- Scenarios where devices need to communicate with each other or with a central server in real-time.

- Applications that require reliable message delivery, even in unreliable network conditions.

How it Works:

MQTT uses a broker (server) to handle the communication between publishers and subscribers. Devices publish messages to specific topics, and subscribers can receive messages by subscribing to those topics. Arduino boards can act as MQTT clients and publish sensor data or other information to an MQTT broker.

AWS IoT Services

AWS IoT Services is a collection of cloud services provided by Amazon Web Services (AWS) for building and managing IoT applications. It includes services like AWS IoT Core, AWS IoT Greengrass, AWS IoT Analytics, and AWS IoT Device Management.

Use Cases:

- IoT applications that require secure communication, device management, and data processing at scale.

- Scenarios where devices need to interact with other AWS services or integrate with existing AWS infrastructure.

- Applications that require advanced analytics, machine learning, or data visualization capabilities.

How it Works:

AWS IoT Core is the central component that allows devices to securely connect and exchange data with the AWS cloud. Devices can publish messages to AWS IoT Core, which can then route the data to other AWS services for storage, processing, or analysis. Arduino boards can connect to AWS IoT Core using MQTT or HTTP protocols.

3. Brief Comparison

Feature	MongoDB	Google Firebase	MQTT	AWS IoT Services
Data Model	Flexible, JSON-like documents	JSON objects	Simple message payloads	Flexible, supports various data formats
Real-time Updates	Supported	Real-time synchronization	Real-time messaging	Supported
Scalability	Highly scalable	Scalable	Scalable with brokers	Highly scalable
Security	Secure with authentication and encryption	Secure with authentication and encryption	Secure with authentication and encryption	Secure with authentication and encryption
Offline Support	Limited	Supported	Limited	Supported
Pricing	Pay-as-you-go	Free tier and pay-as-you-go	Free and open-source brokers available	Pay-as-you-go

4. Example Code: Sending Data from Arduino to MongoDB (DEMO CODE)

```
// Include the necessary libraries
#include <WiFiNINA.h>
#include <WiFiSSLClient.h>

// Your WiFi network credentials
char ssid[] = "YourWiFiSSID"; // Replace with your network SSID
char pass[] = "YourWiFiPassword"; // Replace with your network password

// MongoDB Atlas Data API endpoint
char server[] = "ap-southeast-2.aws.data.mongodb-api.com";
int port = 443; // HTTPS port

void setup() {
  // Initialize serial communication
  Serial.begin(9600);
  while (!Serial);

  // Connect to WiFi network
  WiFi.begin(ssid, pass);
```

```
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi...");
}
Serial.println("Connected to WiFi");

// Create an HTTPS client instance
WiFiSSLClient client;
if (!client.connect(server, port)) {
  Serial.println("Connection failed!");
  return;
}

// Generate a random number between 0 and 100
int randomNumber = random(0, 101);

// HTTP request headers and body
String headers =
  "POST /app/data-bhiqc/endpoint/data/v1/action/insertOne HTTP/1.1\r\n"
  "Host: ap-southeast-2.aws.data.mongodb-api.com\r\n"
  "Content-Type: application/json\r\n"
  "api-key: APIKEY\r\n" // Replace with your API key
  "Content-Length: ";
  String requestBody =
    "{"
    "\"collection\":\"sensordata\","
    "\"database\":\"database\","
    "\"dataSource\":\"sit314\","
    "\"document\":{\"value\": " + String(randomNumber) + "}" // Send the random
number as the data
    "}";

  // Send the HTTP request
  client.print(headers);
  client.print(requestBody.length());
  client.print("\r\n\r\n");
  client.print(requestBody);

  // Read and print the response
  while (client.connected()) {
    String line = client.readStringUntil('\n');
    Serial.println(line);
  }

  // Close the connection
  client.stop();
}


void loop() {
  // You can add other code here if needed
}
```

Explanation:

1. The code includes the necessary libraries for WiFi connectivity (`WiFiNINA.h`) and secure communication over HTTPS (`WiFiSSLClient.h`).

2. It defines the WiFi network credentials (SSID and password) and the MongoDB Atlas Data API endpoint (server and port).

3. In the `setup()` function, the code connects to the specified WiFi network and establishes an HTTPS connection with the MongoDB Atlas Data API endpoint.

4. It generates a random number between 0 and 100 using the `random()` function.

5. The HTTP request headers and body are constructed, including the request method (POST), path (`/app/data-bhiqc/endpoint/data/v1/action/insertOne`), host, content type, API key, and content length.

6. The request body is a JSON document containing the collection name, database name, data source, and the document to be inserted (with the random number as the value of the `"value"` field).

7. The request headers and body are sent to the server using the `WiFiSSLClient` instance.

8. The code reads and prints the response from the server line by line until the connection is closed.

9. Finally, the connection is closed using the `client.stop()` function.

Note: Make sure to replace `"YourWiFiSSID"`, `"YourWiFiPassword"`, and `"APIKEY"` with your actual WiFi credentials and MongoDB Atlas Data API key, respectively, for the code to work correctly.

This example demonstrates how to send data from an Arduino board to a MongoDB database hosted on Atlas using the Data API. The data being sent is a random number, but you can modify the request body to include sensor data or any other relevant information you want to store in the database.

<u>Approach 2</u>

A generic code to push a random number to a MongoDB Atlas database using the Data API. The code includes comments to explain each line:

```c
// Include the necessary libraries
#include <WiFiNINA.h>   // For WiFi connectivity
#include <WiFiSSLClient.h>   // For secure communication over HTTPS
#include <ArduinoJson.h>   // For JSON handling

// WiFi credentials
const char* ssid = "YourWiFiSSID";   // Replace with your WiFi SSID
const char* password = "YourWiFiPassword";   // Replace with your WiFi
password


// MongoDB Atlas Data API endpoint
const char* serverName = "https://your-mongodb-atlas-endpoint.com/app/your-
app-id/endpoint/your-endpoint-path";   // Replace with your MongoDB Atlas Data
API endpoint


WiFiSSLClient wifiClient;   // Create an instance of the WiFiSSLClient class for
secure communication


void setup() {
  Serial.begin(115200);   // Initialize serial communication


  // Connect to WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected to WiFi");
}
```

```
void loop() {
  // Generate a random number between 0 and 100
  int randomNumber = random(0, 101);

  // Create a JSON document with the random number
  StaticJsonDocument<200> doc;   // Create a JSON document with a buffer size of
200 bytes
  doc["value"] = randomNumber;   // Add the random number to the JSON
document

  // Send the data to MongoDB Atlas
  if (sendDataToMongoDB(doc)) {
    Serial.println("Data sent successfully");
  } else {
    Serial.println("Failed to send data");
  }

  delay(2000);   // Wait for 2 seconds before sending the next data
}

bool sendDataToMongoDB(JsonDocument& doc) {
  // Establish a secure connection with the MongoDB Atlas Data API endpoint
  if (!wifiClient.connect(serverName, 443)) {
```

```
  Serial.println("Connection failed");
  return false;
}

// Serialize the JSON document
String jsonPayload;
serializeJson(doc, jsonPayload);

// Construct the HTTP POST request
String httpRequest = String("POST ") + serverName + " HTTP/1.1\r\n" +
                     "Host: " + serverName + "\r\n" +
                     "Content-Type: application/json\r\n" +
                     "Content-Length: " + jsonPayload.length() + "\r\n" +
                     "Connection: close\r\n\r\n" +
                     jsonPayload;

// Send the HTTP request to the server
wifiClient.print(httpRequest);

// Read the response from the server
String response = wifiClient.readStringUntil('\r');
wifiClient.stop();   // Close the connection
```

```
  // Check if the response contains the HTTP status code 200 (OK)
  return (response.indexOf("200") != -1);
}
```

Explanation:

1. The code includes the necessary libraries: `WiFiNINA.h` for WiFi connectivity, `WiFiSSLClient.h` for secure communication over HTTPS, and `ArduinoJson.h` for JSON handling.

2. It defines the WiFi credentials (`ssid` and `password`) and the MongoDB Atlas Data API endpoint (`serverName`). You need to replace these with your actual values.

3. In the `setup()` function, it initializes the serial communication and connects to the WiFi network.

4. In the `loop()` function, it generates a random number between 0 and 100 using the `random()` function.

5. It creates a JSON document using the `ArduinoJson` library and adds the random number to the document.

6. It calls the `sendDataToMongoDB()` function to send the data to the MongoDB Atlas Data API.

7. The `sendDataToMongoDB()` function establishes a secure connection with the MongoDB Atlas Data API endpoint using the `WiFiSSLClient` class.

8. It serializes the JSON document using `serializeJson()` and constructs the HTTP POST request with the appropriate headers and payload.

9. It sends the HTTP request to the server and reads the response.

10. The function returns `true` if the response contains the HTTP status code 200 (OK), indicating successful data insertion.

11. In the `loop()` function, it prints a success or failure message based on the return value of `sendDataToMongoDB()`.

12. The loop repeats after a delay of 2 seconds.

Note:

- Replace `"YourWiFiSSID"` and `"YourWiFiPassword"` with your actual WiFi credentials.

- Replace `"https://your-mongodb-atlas-endpoint.com/app/your-app-id/endpoint/your-endpoint-path"` with the correct MongoDB Atlas Data API endpoint for your application.

This code demonstrates how to generate a random number, create a JSON document with that number, and send it to a MongoDB Atlas database using the Data API. The code is generic and can be easily modified to send different types of data by adjusting the JSON document structure.

More Detailed Research on the use and feature of MongoDB with Arduino

Using MongoDB with Arduino

Arduino boards are widely used in various Internet of Things (IoT) projects and applications. One common requirement is to send data collected by sensors or other inputs to a cloud database for storage, analysis, or further processing. MongoDB Atlas provides a Data API that allows Arduino boards to communicate directly with a MongoDB database hosted on Atlas using standard HTTP requests.

MongoDB Atlas Data API

The MongoDB Atlas Data API is an HTTPS-based API that enables reading and writing data to a MongoDB Atlas database without the need for language-specific drivers. It exposes a set of endpoints that can be used to perform CRUD (Create, Read, Update, Delete) operations and aggregations on the data.

To use the Data API with Arduino, you need to follow these steps:

1. Enable the Data API: From the MongoDB Atlas dashboard, navigate to the "Data API" section and enable the Data API for your desired data source (cluster or database).

2. Create an API Key: Generate a Data API key, which will be used to authenticate your requests to the Data API.

3. Write Arduino Code: Write Arduino code to connect to the WiFi network, construct HTTP requests with the appropriate headers and JSON payloads, and send the requests to the Data API endpoint.

4. Send Data to MongoDB: Use the Data API endpoints to insert, query, update, or delete data in your MongoDB database.

Example Code

Here's an example Arduino code that sends a random number to a MongoDB Atlas database using the Data API:

```cpp
#include <WiFiNINA.h>
#include <WiFiSSLClient.h>
#include <ArduinoJson.h>

// WiFi credentials
const char* ssid = "YourWiFiSSID";
const char* password = "YourWiFiPassword";

// MongoDB Atlas Data API endpoint
const char* serverName = "https://your-mongodb-atlas-endpoint.com/app/your-app-id/endpoint/your-endpoint-path";

WiFiSSLClient wifiClient;

void setup() {
  Serial.begin(115200);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected to WiFi");
}

void loop() {
  // Generate a random number between 0 and 100
  int randomNumber = random(0, 101);

  // Create a JSON document with the random number
  StaticJsonDocument<200> doc;
  doc["value"] = randomNumber;

  // Send the data to MongoDB Atlas
  if (sendDataToMongoDB(doc)) {
    Serial.println("Data sent successfully");
  } else {
    Serial.println("Failed to send data");
  }

  delay(2000);
}

bool sendDataToMongoDB(JsonDocument& doc) {
  // Establish a secure connection with the MongoDB Atlas Data API endpoint
  if (!wifiClient.connect(serverName, 443)) {
    Serial.println("Connection failed");
    return false;
  }

  // Serialize the JSON document
  String jsonPayload;
  serializeJson(doc, jsonPayload);

  // Construct the HTTP POST request
  String httpRequest = String("POST ") + serverName + " HTTP/1.1\r\n" +
                       "Host: " + serverName + "\r\n" +
                       "Content-Type: application/json\r\n" +
                       "Content-Length: " + jsonPayload.length() + "\r\n" +
                        "Connection: close\r\n\r\n" +
                        jsonPayload;

  // Send the HTTP request to the server
  wifiClient.print(httpRequest);

  // Read the response from the server
  String response = wifiClient.readStringUntil('\r');
  wifiClient.stop();

  // Check if the response contains the HTTP status code 200 (OK)
  return (response.indexOf("200") != -1);
}
```

This code generates a random number, creates a JSON document with that number, and sends it to a MongoDB Atlas database using the Data API. The `sendDataToMongoDB()` function establishes a secure connection with the MongoDB Atlas Data API endpoint, constructs the HTTP POST request with the appropriate headers and payload, sends the request, and checks the response for a successful insertion.

Benefits and Limitations

Using the MongoDB Atlas Data API with Arduino offers several benefits:

- No Language-Specific Drivers: The Data API eliminates the need for language-specific drivers, making it easier to integrate with Arduino and other platforms where drivers may not be available or practical.

- Secure Communication: The Data API uses HTTPS for secure communication, ensuring data privacy and integrity.

- Fully Managed: MongoDB Atlas handles the underlying infrastructure, scaling, and maintenance of the database, reducing operational overhead.

However, there are also some limitations:

- Limited Functionality: While the Data API supports CRUD operations and aggregations, it may not provide the full range of functionality available in language-specific drivers.

- Performance Considerations: Sending data over HTTP may introduce additional latency compared to using language-specific drivers, which could be a concern for time-sensitive applications.

- Payload Size Limitations: The Data API has limits on the size of the request and response payloads, which may restrict the amount of data that can be transferred in a single request.

Conclusion

MongoDB Atlas, with its Data API, provides a convenient way to integrate Arduino boards with a scalable and highly available cloud database. By following the steps outlined in this report and using the provided example code, you can start sending data from your Arduino projects to a MongoDB database hosted on Atlas. However, it's important to consider the limitations and performance implications of using the Data API, and evaluate whether it meets the requirements of your specific use case.

Citations:

[1] https://www.mongodb.com/developer/products/atlas/christmas-2021-mongodb-data-api/

[2] https://www.donskytech.com/control-your-arduino-iot-projects-with-a-mongodb-database/

[3] https://www.mongodb.com/community/forums/t/atlas-database-reports/98803

[4] https://www.w3schools.com/mongodb/mongodb_data_api.php

[5] https://www.mongodb.com/blog/post/introducing-mongodb-atlas-data-api-now-available-preview

[6] https://stackoverflow.com/questions/74981620/mongodb-api-document-count

[7] https://www.youtube.com/watch?v=aVRnDu_BYy8