Secure Boot Cloud Technical Details

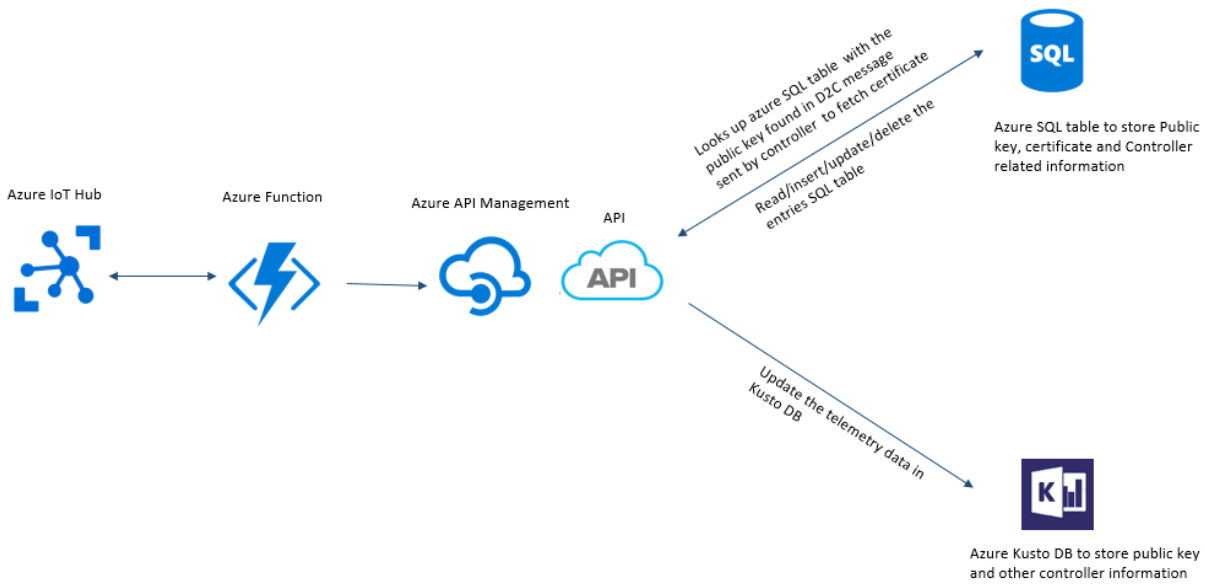
Purpose-The Purpose of this document to provide the technical details of cloud implementation of Secure Boot Project.

Overview- Secure Boot Project is implemented to enable secure booting of IoT Controllers Devices. See [Variscite HAB Wiki] for more details.

Architecture -

The implementation of Secure Boot has 3 components –

1. Azure SQL DB to store Controller Information and Security certificate metadata
2. Rest API to update Azure SQL DB
3. An Azure Function to call API once a D2C message is received



1. A D2C message is received by IoT Hub which is subscribed by an Azure Function.
2. Azure Function will call an API which will check D2C message and decide the next action.
3. If the D2C message contains Public key and Controller related information, the API will fetch the certificate from SQL table based on Public key in the D2C message or update/ insert/delete the entries in the SQL table. The decision to whether update the table or to read the table to fetch IoT Hub Connection string will be made based on D2C message.

Input for API- Public key and Controller related information
Result- read/insert/update/delete the Azure SQL table

4. If the D2C message contains telemetry data, it will be updated in Azure Kusto DB by API.
   Input for API- telemetry data
   Result- insert/update the Azure Kusto DB with telemetry data

Azure SQL Database-

An Azure SQL Database has been which has tables for storing IoT Controller related information and Boot certificate related information.

ControllerInfo Table – This table contains information related to an IoT Controller.

ControllerInfo
ControllerUUID
DeviceID
IoTHubConnString
PublicKey
SerialNo

BootCertificateInfo Table – This table contains information about certificates which are used by Controllers to boot.

BootCertficateInfo
PublicKey
URL
Branch

Measurements Table- Measurements table contains telemetry measurements data.

Tags Table- Tags table contains Tags information.

EventsMeasurements Table- EventsMeasurements table contains information about Events. Currently there is no data flowing for Events so the table is not populated.

Events Table- Events table contains information about each event. Currently there is no data flowing for Events so the table is not populated.

SQL Table Schema-

ControllerInfo	
PartitionKey	String
RowKey	String
TimeStamp	DateTime
ControllerUUID	String
SerialNo	String
IoTHubConnString	String
PublicKey	String
DeviceID	String

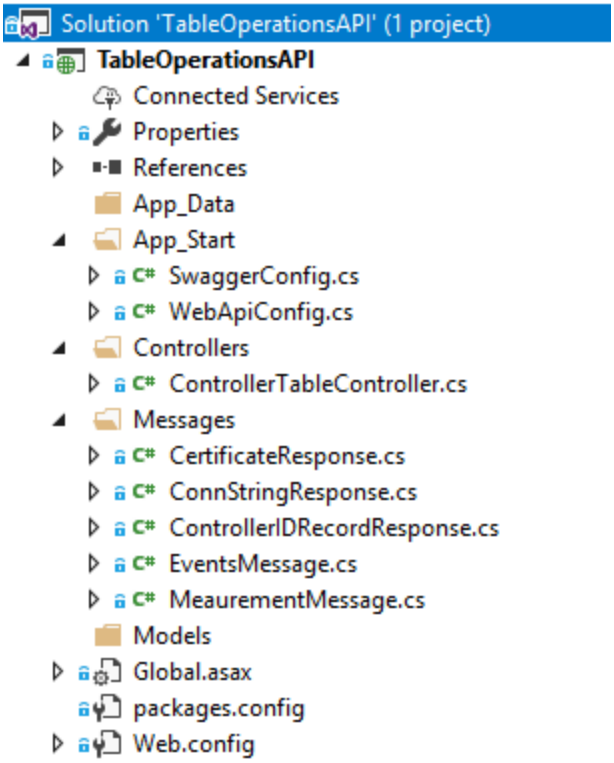
(Not coming from D2C message)

- Partition Key – Public Key of the Certificate used for secure boot
- Row Key – Controller Unique ID
- Timestamp – DateTime when entry was added/modified in table
- Controller UUID – Control unique ID (from D2C message)
- Serial No – som serial (from D2C message)
- IoTHubConnString – Connection String of IoT hub (from D2C message)
- Public Key – Public Key of the Certificate used for secure boot (from D2C message)
- Device ID – Gateway device ID (from D2C message)

REST API-

The *TableOperationsAPI* REST API updates the Azure SQL DB. It is checks the message Type and/or Operation in the request message and updates the DB accordingly. Request and Response bodies are formatted in JSON. The Connection string and Tables related configuration information is stored in web.config file. This API is hosted as API App in Azure cloud.

Solution structure- Solution has a Single Controller (ControllerTableController) which handles all the requests. Required classes to hold messages are in the Messages Folder.



Within the Controller, the Request Body is first parsed and Message Type is extracted.

```
[HttpPost]
public IHttpActionResult Post([FromBody] JObject json)
{
    string MsgType = "";
    string ControllerTableName = ConfigurationManager.AppSettings["ControllerTableName"];
    string PKCertificateTableName = ConfigurationManager.AppSettings["PKCertificateTableName"];
    string TelemetryTableName = ConfigurationManager.AppSettings["TelemetryTableName"];
    string MeasurementTable = ConfigurationManager.AppSettings["MeasurementsTableName"];
    string TagsTable = ConfigurationManager.AppSettings["TagsTableName"];
    string EventsMeasurementTable = ConfigurationManager.AppSettings["EventsMeasurementsTableName"];
    string EventsTagsTable = ConfigurationManager.AppSettings["EventsTagsTableName"];
    string DataSource = ConfigurationManager.AppSettings["DataSource"];
    string UserID = ConfigurationManager.AppSettings["UserID"];
    string Password = ConfigurationManager.AppSettings["Password"];
    string Database = ConfigurationManager.AppSettings["DBName"];
    string jsonResponseToCallerstring = "";
    var jsonResponseToCaller = (dynamic)null;

    if (json.TryGetValue("msg", out JToken msgtype))
    {
        MsgType = (string)msgtype;
    }
```

Message Types accepted-

1. Universal.Measurements-

If message type is Universal.Measurements then API updates the Measurements and Tags tables. INSERT operation is supported for this message type. Response is returned accordingly to the caller in case of success and failure scenarios.

2. Universal.Event-
   If message type is Universal. Event then API updates the EventsMeasurements and Events tables. INSERT operation is supported for this message type. Response is returned accordingly to the caller in case of success and failure scenarios.

3. Certificate.Config-
   If message type is Certificate.Config then API updates/reads the BootCertificateInfo table. INSERT and READ (based on PublicKey field) operations are supported for this message type. Response is returned accordingly to the caller in case of success and failure scenarios. In READ operation, the record from table is returned based on PublicKey in the request.

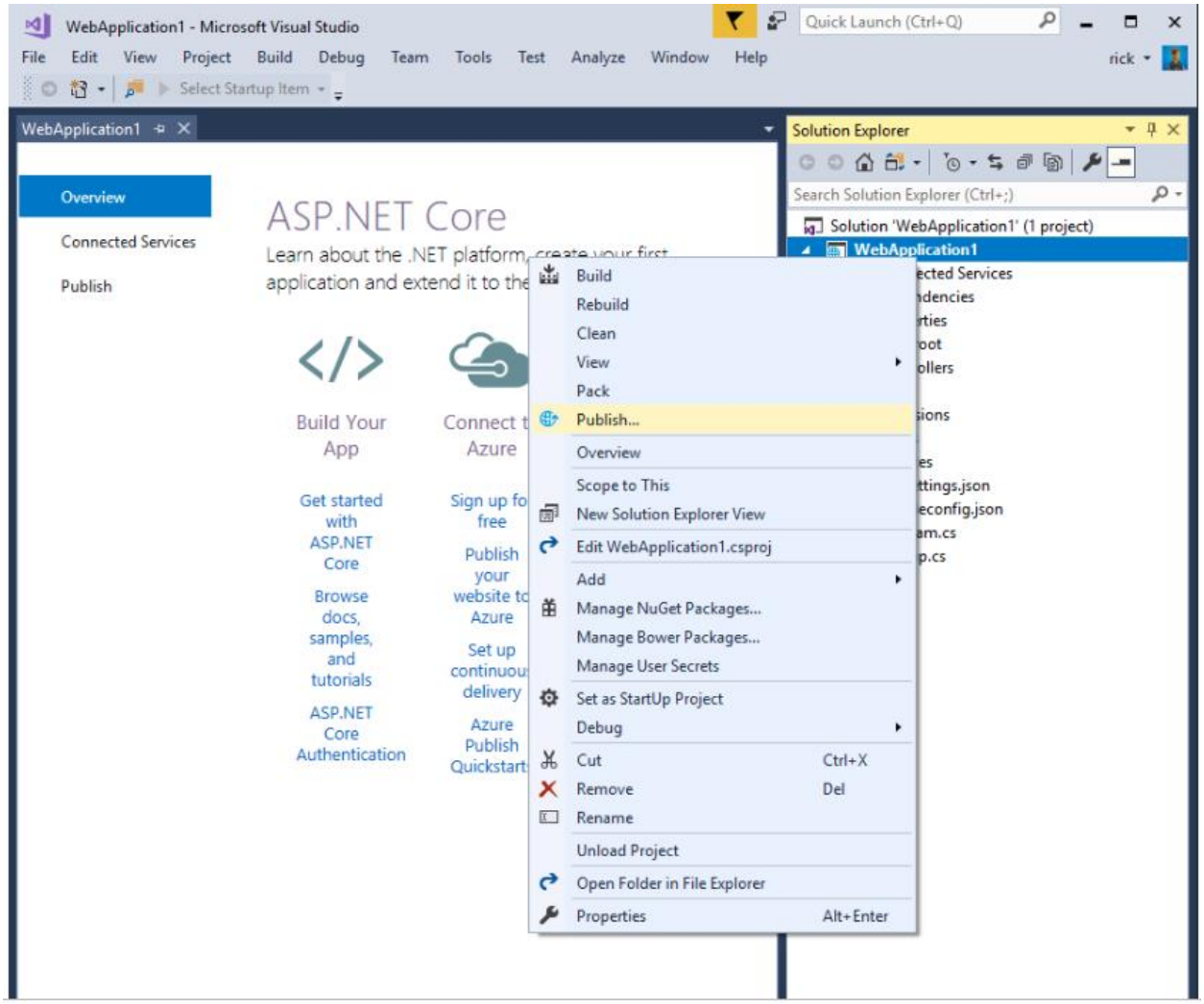
4. Controller.Config- If message type is Controller.Config then API upserts/reads/deletes from the BootCertificateInfo table based on the Operation in the Request message. UPSERT, DELETE and READ operations are supported for this message type. Response is returned accordingly to the caller in case of success and failure scenarios. In READ operation, the record from table is returned based on PublicKey in the request.
   3 types of READ operations are supported-
   Operation- GetConnString
   This operation fetches the IoT Hub connection string and Device ID from ControllerInfo table based on Controller GUID.

   Operation- GetDataByControllerID-
   This operation fetches the complete record from ControllerInfo table based on Controller GUID.

   Operation- GetDataBySerialNo-
   This operation fetches the complete record from ControllerInfo table based on Controller Serial No.

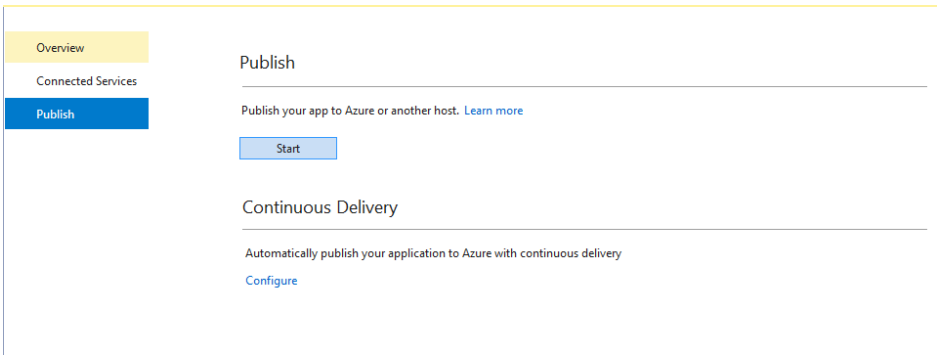
Deploy the app to Azure

1. Right-click on the project in Solution Explorer and select Publish....

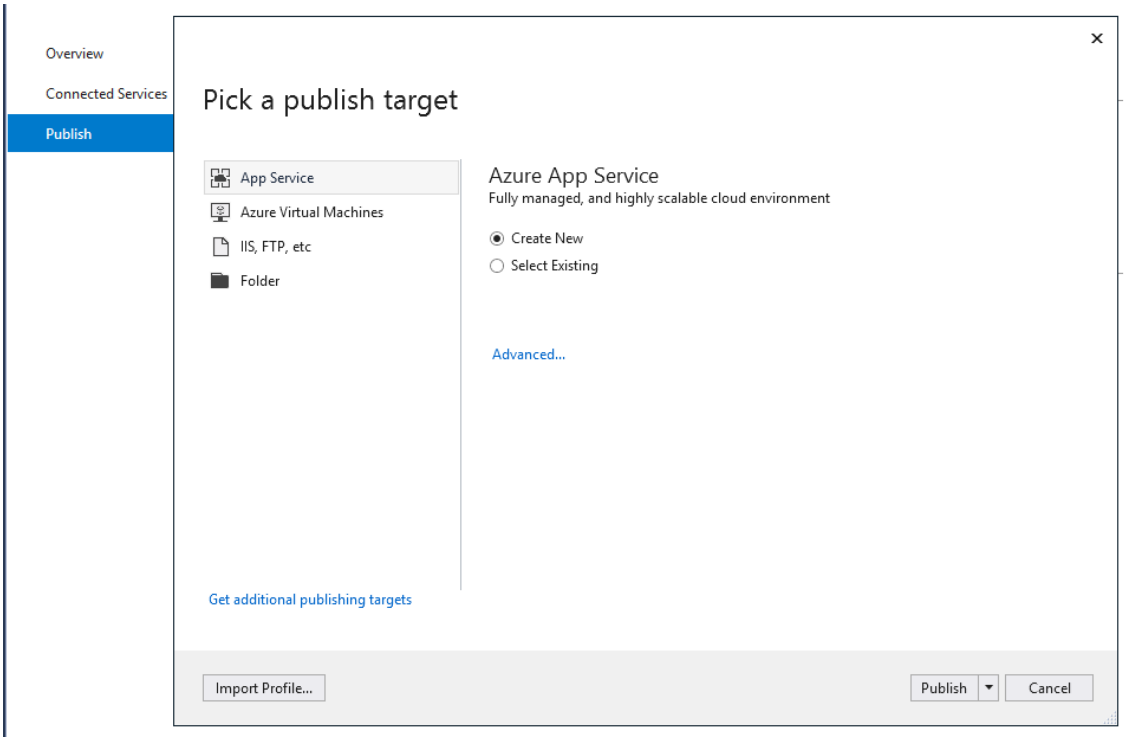


In the **Publish** dialog:

- Click on Start.



- **Pick a publish target** Dialog appears.



- Select **App Service**. Select Publish.

Create Azure resources

The **Create App Service** dialog appears:

- Enter your subscription.
- The **App Name**, **Resource Group**, and **App Service Plan** entry fields are populated. You can keep these names or change them.

- Select **Create** on the **Create App Service** dialog. Visual Studio creates the Web app on Azure and publish the API to the Web App. This step can take a few minutes.

Azure Function- Azure Function *TableOperationsFunc* subscribes to the D2C message from IoT Hub via IotHubTrigger and calls the TableOperationsAPI to update the Azure SQL DB with data in the D2C message. Function checks the TelemetryForm property of D2C message to identify the message and then requests for JWT token to call the API since the API is secured with OAuth 2.0.

```
public static async Task Run(EventData myIoTHubMessage, ILogger log)
{
    log.LogInformation("entered function");
    string reqbody = Encoding.UTF8.GetString(myIoTHubMessage.Body);
    log.LogInformation($"MessageBody: {reqbody}");

    try
    {
        if(myIoTHubMessage.Properties.ContainsKey("TelemetryForm"))
            {
                string MessageType = myIoTHubMessage.Properties["TelemetryForm"].ToString();
                log.LogInformation($"Message Type: {MessageType}");
                //string MessageType = "Controller.Config";
                if((MessageType == "Controller.Config") || (MessageType == "Main") || (MessageType == "Event"))
                {
                    string DeviceID ="";

                 if(myIoTHubMessage.Properties.ContainsKey("DeviceId"))
                 {
                     DeviceID  = myIoTHubMessage.Properties["DeviceId"].ToString();
                     log.LogInformation($"DeviceID: {DeviceID}");

            AccessToken token = GetToken().Result;
            string JWT = token.access_token;
            log.LogInformation($"token: {JWT}");

            string req = Encoding.UTF8.GetString(myIoTHubMessage.Body);
```

This token is requested using Client Credentials Grant Type. This token is requested using an app which is registered in Azure Active Directory and has access to the API. GetToken() in Azure Function requests the JWT token.

```
static async Task<AccessToken> GetToken()
{
    using (var client = new HttpClient())
    {
                string clientId = ClientId;
                string clientSecret = ClientSecret;
                string credentials = String.Format("{0}:{1}", clientId, clientSecret);
                client.DefaultRequestHeaders.Accept.Clear();
                client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

                client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Basic", Convert.ToBase64String(Encoding.UTF8.GetBytes(
                //Prepare Request Body
                List<KeyValuePair<string, string>> requestData = new List<KeyValuePair<string, string>>();

                requestData.Add(new KeyValuePair<string, string>("grant_type", "client_credentials"));
                requestData.Add(new KeyValuePair<string, string>("Scope", Scope));

                FormUrlEncodedContent requestBody = new FormUrlEncodedContent(requestData);

                //Request Token
                var request = await client.PostAsync(TokenEP, requestBody);
                var response = await request.Content.ReadAsStringAsync();
               return JsonConvert.DeserializeObject<AccessToken>(response);
    }
```

Once the token is received this token is passed in header while calling the API.

```
AccessToken token = GetToken().Result;
string JWT = token.access_token;
log.LogInformation($"token: {JWT}");

string req = Encoding.UTF8.GetString(myIoTHubMessage.Body);
var DATA = req;
log.LogInformation($"web req: {req}");
string URL = APIURL;
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(URL);
request.Method = "POST";
request.ContentType = "application/json";
request.Headers.Add("Ocp-Apim-Subscription-Key", APIMSubscriptionKey);
request.Headers.Add("Authorization", "Bearer " + JWT);
request.ContentLength = DATA.Length;
StreamWriter requestWriter = new StreamWriter(request.GetRequestStream(), System.Text.Encoding.ASCII);
requestWriter.Write(DATA);
requestWriter.Close();
try
{
    HttpWebResponse webResponse = (HttpWebResponse)request.GetResponse();
```

The response from API is send back to Device (C2D message) if the D2C message is of Controller.Config type. As per the requirements, the other messages don't expect the response to be sent back to Device.
SendC2D() is used to send API response to the Device.

```
static async Task SendC2D(string deviceID, string message, ILogger log)
        {
            //Deserialize the alertMessage into JSON
                    string MessageJson = JsonConvert.SerializeObject(message);

                    //Get the message bytes
                    var MessageBytes = Encoding.ASCII.GetBytes(MessageJson);

                    //Create an IoT Hub Microsoft.Azure.Devices with our data in it
                    var cloudToDeviceMessage = new Message(MessageBytes);

                     serviceClient = ServiceClient.CreateFromConnectionString(ConnectionString, Microsoft.Azure.Devices.TransportType.Amqp);
                    //Finally, Use the IoT Hub service client to send the cloud-to-device message asynchronously
                    //await serviceClient.SendAsync("MyDotnetDevice", cloudToDeviceMessage);
                    await serviceClient.SendAsync(deviceID, cloudToDeviceMessage);
                    log.LogInformation("sent C2D succesfully");

        }
```

API Management – Table Operations API Overview

Welcome to TableOperations API Documentation. TableOperations API enables user to perform operations on tables in SecureBoot SQL Database which is based in Azure.

Getting Started

TableOperations API is HTTP-based RESTful API that uses OAuth 2.0 for authorization. Request and Response bodies are formatted in JSON.

Authentication and authorization

The TableOperations REST API uses the OAuth 2.0 protocol to authorize calls. OAuth is an open standard that many companies use to provide secure access to protected resources.

Every application that calls the API needs to be registered as an application in customer Azure Active Directory. The registered app's client ID and client secret are then used to request JWT from Azure Active Directory endpoint.

For requesting JWT token below values are required.

Grant Type: Client Credentials

Access Token URL:
https://login.microsoftonline.com/****.microsoftonline.com/oauth2/v2.0/token

ClientID: Application ID of the client App you registered

Client Secret: key value of the key created while registering the client App.

Scope: https://***.microsoftonline.com/TableOperationsAPI/.default

Sample screenshot from Postman:

Once JWT token is acquired, the same should be sent as value of Authorization in the call to API.

```
POST /api/TableOperations HTTP/1.1
Host: localhost:58244
Content-Type: application/json
Ocp-Apim-Subscription-Key:
Authorization: Bearer
cache-control: no-cache
Postman-Token: ebc0a413-983f-45eb-bf00-bbb20f375ba2
{
    "op":"GetConnString",
    "uuid": "testcontrollerid"
}

------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

API requests

Component	Description
The HTTP method	POST- Submits data to a resource to process.
The URL to the API service	https://securebootapim.azure-api.net/api/TableOperations
Query parameters	N/A
HTTP request headers	• The Authorization header with the access token. • Subscription Key of Azure API management instance • Content-Type: application/json
A JSON request body	Required for all the calls.

1. Get Controller Connection String-

This request fetches the connection string and device ID from ControllerInfo table for the Controller ID passed in the request. Both the parameters are mandatory.
Request parameters-

Key	Value
op	GetConnString
uuid	Controller ID

Sample request-

```
{
  "op":"GetConnString",
  "uuid": "ControllerIDvalue"
}
```

Sample response-

```
{
  "DeviceID": "DeviceIDValue",
  "ConnectionString": "ConnectionstringValue"
}
```

2. Get Details for Controller ID-

This request fetches the complete record from ControllerInfo table for the Controller ID passed in the request. All 3 parameters are mandatory.
Request parameters-

Key	Value
op	GetDataByControllerID
uuid	Controller ID
msg	Controller.Config

Sample request-

```
{
  "msg": "Controller.Config",
  "op":" GetDataByControllerID",
  "uuid": "ControllerIDvalue"
}
```

Sample response-

```
{
  "ControllerID": "ControllerIDvalue ",
```

```
  "DeviceID": "DeviceIDValue ",
  "ConnectionString": "ConnStringValue",
  "PublicKey": "PublicKeyValue",
  "SerialNo": "SerialNoValue"
}
```


3. Get Details for Controller Serial Number-

This request fetches the complete record from ControllerInfo table for the Controller Serial Number passed in the request. All 3 parameters are mandatory.
Request parameters-

Key	Value
op	GetDataBySerialNo
serial	SerialNo
msg	Controller.Config

Sample request-


```
{
  "msg": "Controller.Config",
  "op":"GetDataBySerialNo",
  "uuid": "SerialNoValue"
}
```


Sample response-


```
{
  "ControllerID": "ControllerIDvalue ",
  "DeviceID": "DeviceIDValue ",
  "ConnectionString": "ConnStringValue",
  "PublicKey": "PublicKeyValue",
  "SerialNo": "SerialNoValue"
}
```

4. Upsert Operation on ControllerInfo table-

This request inserts a new record in the ControllerInfo table if the record doesn't exist already and updates the record if record exists previously for the Controller ID passed in the request. All the parameters are mandatory.
Request parameters-

Key	Value
op	upsert
msg	Controller.Config
uuid	Controller ID value
hubconn	IoT Hub Connection String value
serial	Serial No value
deviceid	Device ID value

Sample request-

```
{
        "msg": "Controller.Config",
        "op":"upsert",
        "uuid": "ControllerIDvalue",
        "serial":"SerialNoValue",
        "hubconn": "IoTHubConnStringValue",
        "deviceid": "DeviceIDValue"
 }
```

Sample response-

```
{
   "Message": "Table was updated successfully."
}
```

5. Delete Operation on ControllerInfo table-

This request deletes a record in the ControllerInfo table with the Controller ID passed in the request. Operation, Message Type and Controller ID are mandatory.

Request parameters-

Key	Value
op	GetDataBySerialNo
serial	SerialNo

Sample request-

```
{
        "msg": "Controller.Config",
        "op":"delete",
        "uuid": "ControllerIDvalue"
}
```

Sample response-

```
{
   "Message": "Table was updated successfully."
}
```

6. Insert Operation on BootCertificateInfo table-

This request inserts a record in the BootCertificateInfo table with the Public Key, Certificate Location and Branch passed in the request. Operation, Message Type and Public Key are mandatory.

Request parameters-

Key	Value
op	Insert
msg	Certificate.Config

url	URL value
branch	Branch value
hexdump	Public Key Value

Sample request-

```
{
 “op”: ”Insert”
 "msg": "Certificate.Config",
 "url": "URLValue",
 "branch": "Branch value",
 "hexdump": "Public Key Value"
}
```

Sample response-

```
{
   "Message": "Table was updated successfully."
}
```

7. Read Operation on BootCertificateInfo table-

This request gets the Certificate details from the BootCertificateInfo table with the Public Key passed in the request. Operation, Message Type and Public Key are mandatory.

Request parameters-

Key	Value
op	Insert
msg	Certificate.Config
hexdump	Public Key Value

Sample request-

```
{
 “op”: ”Read”
 "msg": "Certificate.Config",
 "hexdump": "Public Key Value"
}
```

Sample response-

```
{
  "PublicKey": "Public Key Value",
  "CertificateURL": "Certificate Location",
  "Branch": "GitHub Branch"
}
```

8. Insert Operation on Measurement data tables-

This request inserts a record in the Measurements table with the TagsID (auto generated GUID), AppID, Apptype, Seq, Timestamp & ControllerUUID and multiple records (depends on number of tags in the message) in Tags table with TagsID (same GUID which was generated for Measurements), Tagname, SCID, RID, V and Timestamp. Messagetype is mandatory.

Sample request-

```
{
        "msg": "Universal.Measurements",
        "agg": "RW382c4abab9e3",
        "seq": 0,
        "meas": {
                "uuid": "2273374e-9192-44e7-9393-ec7211bce696",
                "application": [{
                        "appid": "483c-8605-0482e10d85a3",
                        "apptype": "********",
                        "v": [{
                                "id": "PID 1 log",
```

```
                                    "ts": "2019-02-05T09:44:17-0600",
                                    "tags": [{
                                            "tag": "pterm1",
                                            "v": 3.2113037109375,
                                            "rid": 27258,
                                            "scid": 16180
                                    }, {
                                            "tag": "iterm1",
                                            "v": 0.98745894432067871,
                                            "rid": 27260,
                                            "scid": 16110
                                    }, {
                                            "tag": "pb1",
                                            "v": 140.1300048828125,
                                            "rid": 27257,
                                            "scid": 16160
                                    }]
                            }]
                    }
            }
```


Sample response-


```
{
   "Message": "Measurement data tables were updated successfully."
}
```


9. Insert Operation on Events data tables-

This request inserts a record in the Events table with the TagsID (auto generated GUID), AppID, Apptype, Seq & ControllerUUID and multiple records (depends on number of tags in the message) in Tags table with TagsID (same GUID which was generated for EventsMeasurements),code, T, AddL, itag, irid, iscid, otag, orid, oscid, dtag, drid, dscid, IV and OV. Messagetype is mandatory.

Sample request-

```
{
        "msg": "Universal.Measurements",
        "agg": "",
        "seq": 0,
        "meas":
        {
        "uuid": "GUID",
        "application":
                [
                        {
                                "appid": "GUID",
                                "apptype": "int",
                                "v":
                                [
                                        {
                                                "id": "",
                                                "ts": "2018-11-02T14:24:43-
0500",
                                                "tags":[
                                                        {
                                                        "rid": "int",
                                                        "scid": "int",
                                                        "tag": "" ,
                                                        "t": "",
                                                        "v": 3.14159
                                                        }
                                                ]
                                        }
                                ]
                        }
                ]
        }
}
```


Sample response-


```
{
```

```
    "Message": "Measurement data tables were updated successfully."
}
```