

Cognizant Technology Solutions

Cognizant EV Charging Protocol Gateway 1.0: Technical Design Document

Version: 1.0

Contents

[1. Introduction 3](#_Toc61515160)

[1.1. Purpose of the document 3](#_Toc61515161)

[1.2 Intended audience 3](#_Toc61515162)

[1.3 Definitions 3](#_Toc61515163)

[1.4 Overview of OCPP & EV Charging 3](#_Toc61515164)

[2. Technical architecture 4](#_Toc61515165)

[2.1 Technical architecture diagram 4](#_Toc61515166)

[2.2 Components used 4](#_Toc61515167)

[3. How to configure and use 6](#_Toc61515168)

[3.1 Steps to follow for configuration 6](#_Toc61515169)

[3.2 Flow of actions 9](#_Toc61515170)

**Version History**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description** |
| 1.0 | 28-December-2020 | First version of the document |

# 1. Introduction

## 1.1. Purpose of the document

This document describes technical design details of Cognizant EV Charging Protocol Gateway that simplifies development of EV Charge Management solution using Azure IoT services.

## 1.2 Intended audience

This document is intended for developers who want to develop EV Charge Management solution using Azure IoT Hub and other Azure services.

## 1.3 Definitions

This section contains the terminology used throughout this document in context of the Cognizant EV Charging Protocol Gateway component.

|  |  |
| --- | --- |
| OCPP | Open Charge Point Protocol. Please refer: <https://www.openchargealliance.org/> for details |
| EV Charge Management solution | The EV Charge Management solution that the developer user of Cognizant EV Charging Protocol Gateway intends to develop. |
| Azure portal | Microsoft provided portal using which you can manage your Azure subscription and various cloud based components. Please refer: <https://portal.azure.com/> |

## 1.4 Overview of OCPP & EV Charging

The Open Charge Point Protocol ([OCPP](https://www.openchargealliance.org/)) is an open standard application protocol for communication between Electric vehicle chargers (EVSE) and OCPP Central System (or only “Central System”). The standard elaborates several standard operations to enable standardized communication between EVSE & Central System. OCPP protocol defines interfacing operations & messages standards / guidelines. It consists of two conceptual entities

* EVSE (the charger)
* OCPP Central System

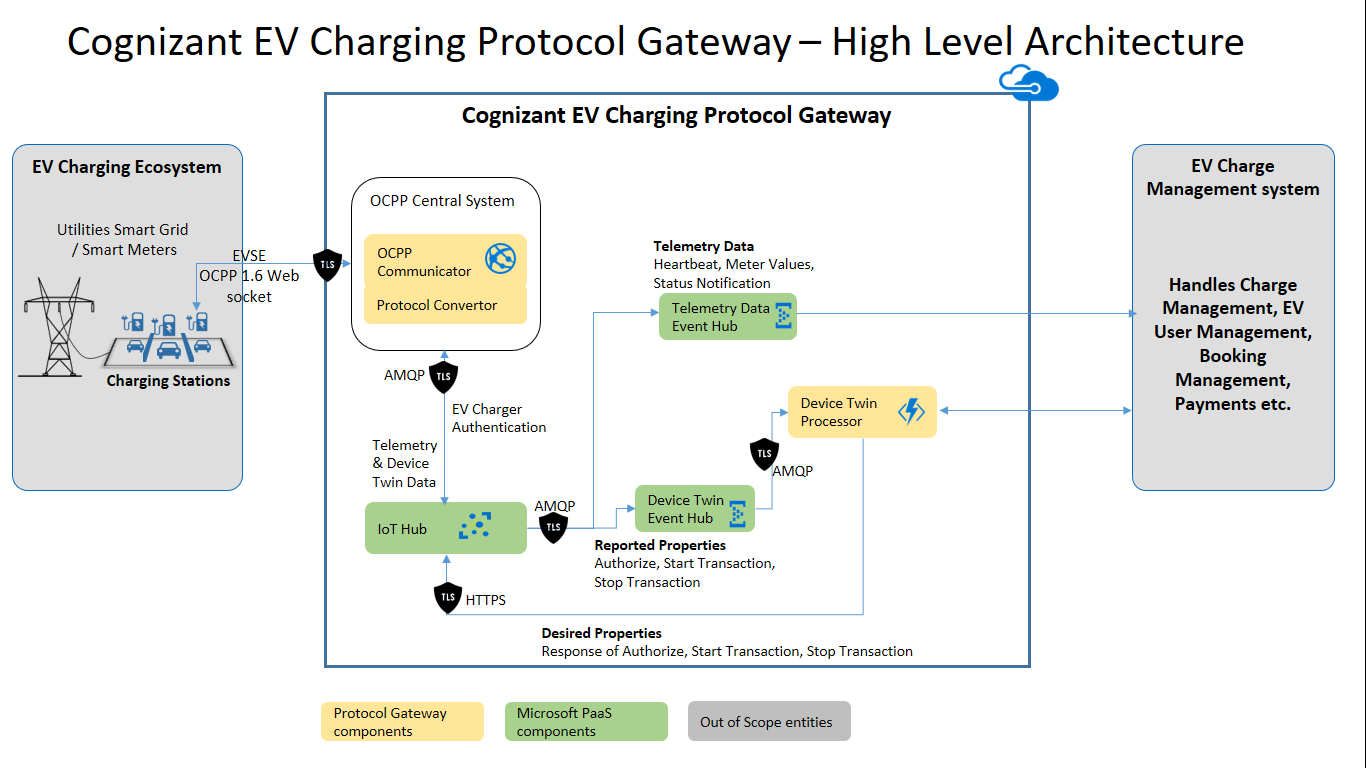
EVSE side implementation is done by OEMs (ABB, Schneider, Siemens etc.). In order to use these EVSE hardware, Charge Point Operator (CPO) needs a software handle (OCPP Central System). Central System receives OCPP commands from EV Charger. It then communicates with the EV Charge Management system for getting response of OCPP commands.

To develop an IoT solution like EV Charge Management in Azure, developers would prefer to use specialized service like Azure IoT Hub. However, OCPP being a specialized protocol for EV Charging, it is not supported by Azure IoT Hub. Cognizant EV Charging Protocol Gateway enabled communication between Azure IoT Hub and OCPP Central System by translating OCPP protocol into protocol supported by Azure IoT Hub like AMQP. Using this Gateway, EV Chargers data can be routed to Azure IoT Hub to fully utilize Azure IoT Hub capabilities. Thus, it allows the developers to develop EV Charge Management solution with familiar and powerful Azure IoT services.

# 2. Technical architecture

## 2.1 Technical architecture diagram

Below is the technical architecture diagram explaining individual components used as part of the Cognizant EV Charging Protocol Gateway.



## 2.2 Components used

* OCPP Central System

This is the Charge Point Management System that communicates with Charge Points (EV Chargers) through various OCPP commands. It also communicates with EV Charge Management system for getting response of OCPP commands. (Note: EV Charge Management system is responsible for processing OCPP commands, Charge Management, EV User Management, Booking Management, Payments etc. Cognizant EV Charging Protocol Gateway can communicate with EV Charge Management system using RESTful API endpoints. It is out of scope for the Cognizant EV Charging Protocol Gateway).

* + OCPP Communicator

It works as primary point of communication between EV Charger and OCPP Central System. It provides a secure way of communication to EV Charger using WSS (Web Socket over SSL/TLS). It exposes a Web Socket URL using which EV Chargers establishes connection with the Central System. It supports JSON implementation of OCPP 1.6 protocol. Please refer ocpp-j-1.6-specifications from <https://www.openchargealliance.org/>

* + Protocol Convertor

It converts OCPP protocol into IoT Hub supported protocol (AMQP) and establishes communication with IoT Hub. It performs following activities:

1. Protocol conversion from OCPP to AMQP and vice versa
2. Sending Telemetry Data to IoT Hub
3. Setting Reported Properties (using IoT Hub Device Twin)
4. Listening to Desired Properties (using IoT Hub Device Twin)

* Azure IoT Hub

It is used to receive the ingested data i.e. telemetry data and Device Twin updates. OCPP Central System sends telemetry data and Device Twin updates to IoT Hub. Separate message routes are required to process each one of them. These message routes are created in IoT Hub using Azure portal. One message route transfers telemetry data to Telemetry Data Event Hub endpoint. Moreover, another message routes transfers Device Twin updates to Device Twin Event Hub endpoint.

* Telemetry Data Event Hub

This Event Hub is used to receive ingested Telemetry data, which is forwarded by IoT Hub message route. Telemetry data can be consumed by writing Even Hub listener code. Creation of Event Hub Listener code is out of scope for Cognizant EV Charging Protocol Gateway.

* Device Twin Event Hub

This Event Hub is used to receive ingested Device Twin updates, which are forwarded by IoT Hub message route.

* Device Twin Processor

It is an Azure Function that perform following tasks:

* + Read Reported Properties from the Device Twin updates forwarded by the IoT Hub message route to Device Twin Event Hub. Convert them into format required by the EV Charge Management system APIs and pass them to the EV Charge Management system.
  + Read the response from the EV Charge Management system APIs and update Desired Properties accordingly in IoT Hub Device Twin.

# 3. How to configure and use

## 3.1 Steps to follow for configuration

Please follow below steps to configure the Cognizant EV Charging Protocol Gateway.

1. Download the complete solution from location: <https://github.com/CognizantIoT/Cognizant-EV-Charging-Protocol-Gateway-1.0>
2. Create an Azure IoT Hub from Azure portal. Copy the IoT Hub connection string from Azure portal.
3. In the downloaded solution, unzip the file “OCPP Central System.zip”. Go to the folder “OCPP Central System” This is the OCPP Central System component. Open the file *OCPP Central System\ChargePointOperator\appsettings.json* in your favorite editor. It supports .Net Core 2.2. Update the value of key *IoTHubString* with IoT Hub connection string copied in above step.
4. Please refer key *OCPPBootInterval.* If the OCPP Central System rejects Boot Notification command, EV Charger should send the same command again after the time interval (in seconds) mentioned in value of this key. If the OCPP Central System accepts Boot Notification command, EV Charger should send Heartbeat command after the time interval mentioned in value of this key (ex. After every 120 seconds).
5. Deploy the OCPP Central System component on your Azure subscription as App Service (Web app).
6. There are two ways of logging various events in the OCPP Central system components. Either of those can be used.
7. There is inbuilt logging mechanism in OCPP Central System component, which saves the logs on Azure portal under the App Service, created in above steps. To View the logs, go to the deployed App Service in Azure portal. From the left panel, under heading “*Development Tools*”, click on “*Advanced Tools*”. It opens a new window with URL as: *https://<AppServiceName>.scm.azurewebsites.net/*. From top menu, click on “*Debug Console*” and then on “*CMD*”. It shows the Azure hosted file system of App Service. Click on *site* => *wwwroot* => *Logs*. It will show the various log files.
8. Using custom API: You need to create a custom API and configure deployed API URL in value of key *LogURL* in the file *OCPP Central System\ChargePointOperator\appsettings.json.* After redeployment, OCPP Central System will send required data for logging, to this custom API. Following is the required schema for this API

API type: POST

Input: {

"StationChargerId": <integer, The charger id from which OCPP command is received>,

"Input": <string, Input of the OCPP command>,

"Command": <string, OCPP command name>,

"IsRequest": <Boolean, True: if it is OCPP command request. False: if it is OCPP command response>

}

1. Get the WSS endpoint URL of the published App Service from Azure portal. It should be *wss://<AppServiceName>.azurewebsites.net*. Configure this URL in the EV Charger as base URL for central system endpoint.
2. Create required IoT devices in IoT Hub. For each EV charger you want to connect with the Gateway, create a separate IoT device in IoT Hub.
3. Create two Azure Event hubs (Telemetry Data Event Hub and Device Twin event Hub) using Azure portal.
4. Create a message route in the IoT Hub. This route should forward telemetry data i.e. “Device Telemetry Messages” to an Event Hub endpoint. Use Telemetry Data Event Hub created in above step, as this endpoint. Telemetry data will be sent to this Event Hub. Create a listener for this Event Hub to further process the Telemetry data as required. Creating the listener is out of scope of this document.
5. Create another message route in the IoT Hub. This route should forward Device Twin updates i.e. “Device Twin Change Events” to an Event Hub endpoint. Use Device Twin event Hub created in above step, as this endpoint.
6. Create and deploy custom APIs to accept data from Device Twin Processor. Below are the required API signatures. Please refer ocpp-j-1.6-specifications from <https://www.openchargealliance.org/> for specific parameter info.

* **Authorize**:

API type: POST

Input: {

"StationChargerId": <integer, The charger id from which OCPP command is received >,

"idTag": <string>

}

Output: {

"idTagInfo": {

"expiryDate": null, <as per OCPP 1.6 guidelines, this parameter is optional>

"parentIdTag": null, <as per OCPP 1.6 guidelines, this parameter is optional>

"status": <string>

}

}

* **StartChargeTransaction**:

API type: POST

Input: {

"IdTag": <string>,

"ConnectorId": <integer>,

"MeterStart": <integer>,

"StationChargerId": <integer>,

"timestamp": <UTC timestamp in string format ex. 2020-09-03T11:21:00.962Z>

}

Output: {

"idTagInfo": {

"expiryDate": null, <as per OCPP 1.6 guidelines, this parameter is optional>

"parentIdTag": null, <as per OCPP 1.6 guidelines, this parameter is optional>

"status": <string>

},

"transactionId": <integer>

}

* **StopChargeTransaction**

API type: POST

Input: {

"idTag": <string>,

"MeterStop": <integer>,

"timestamp": <UTC timestamp in string format ex. 2020-09-03T11:21:00.962Z>,

"TransactionId": <integer>,

"Reason": <string>,

"MeterValue": <integer>

}

Output: {

"IdTagInfo": {

"expiryDate": null, <as per OCPP 1.6 guidelines, this parameter is optional>

"parentIdTag": null, <as per OCPP 1.6 guidelines, this parameter is optional>

"status": <string>

}

}

1. From the downloaded solution, unzip the file “Device Twin Processor.zip”. Go to folder “Device Twin Processor”. This is the Device Twin Processor component. It supports .Net Core 2.2. It is Azure Function. Create a new Azure Function App using Azure portal. Go to the Function App on Azure portal. In the left panel, under heading “*Settings*”, click “*Configuration*”. Create following new Application settings keys
2. AzureWebJobsStorage: Connection string of Azure Storage Account. It is created automatically when you create Azure Function App on Azure portal.
3. EventHubConnectionString: Connection String of input Event Hub (Device Twin event Hub - used for sending Device Twin updates) for this Azure Function. The Azure Function is triggered when this Event Hub receives any data.
4. IOTHubConnectionString: Connection string of IoT hub, which was created in earlier steps.
5. AuthorizeChargerAPIURL: URL of the EV Charge Management system API to process OCPP **Authorize** command
6. StartChargeTransactionAPIURL: URL of the EV Charge Management system API to process OCPP **Start Transaction** command**.**
7. StopChargeTransactionAPIURL: URL of the EV Charge Management system API to process OCPP **Stop Transaction** command.
8. In the left panel, click “*Overview*”. At the top right bar, click on “*Get publish profile*”. It will download a “*<FunctionAppName>.PublishSettings*” file. Deploy Device Twin Processor as Azure Function using the downloaded publish settings file.
9. The Cognizant EV Charging Protocol Gateway is ready to use now. Start your EV Charger and Telemetry data will start flowing in the Telemetry Data Event Hub.

## 3.2 Flow of actions

This section describes a general happy path flow of actions/data, which flows from EV Charger to EV Charge Management system, through the Cognizant EV Charging Protocol Gateway. For the exact data used in any of the OCPP command, please refer ocpp-j-1.6-specifications from <https://www.openchargealliance.org/>

1. **Connection to the OCPP Central System**: A charging cycle starts with an EV Charger connection with OCPP Central System. As explained in above sections, the OCPP Central System component exposes a WSS URL (Web Socket) to which an EV Charger can connect.
2. **Boot Notification**: After successful connection, the EV Charger sends BootNotification.req along with the necessary input data. OCPP Central system reads the input OCPP command and its associated data. The Protocol Convertor then checks with IoT Hub whether the requesting EV Charger is registered in the IoT Hub as an IOT Device or not. Based on the response from IoT Hub, the OCPP Central system responds the EV Charger with BootNotification.conf
3. **Heartbeat**: EV Charger sends Heartbeat.req to OCPP Central System. Protocol Convertor sends it to IoT Hub. As it is telemetry data, it is routed to Event Hub through IoT Hub message route.
4. **Meter Values**: EV Charger sends MeterValues.req to OCPP Central System. Protocol Convertor sends it to IoT Hub. As it is telemetry data, it is routed to Event Hub through IoT Hub message route.
5. **Status Notification**: EV Charger sends StatusNotification.req to OCPP Central System. Protocol Convertor sends it to IoT Hub. As it is telemetry data, it is routed to Event Hub through IoT Hub message route.
6. **Authorize**: EV Charger sends Authorize.req to OCPP Central System. OCPP Convertor sets IoT Hub Reported Property with necessary data for this command. As it is Device Twin updates, it is routed to Event Hub through IoT Hub message route. Device Twin Processor reads the data and sends to the EV Charge Management system. EV Charge Management system processes the command and provides output. This output should be sent to the EV Charger. Therefore, Device Twin Processor updates the IoT Hub Desired Property for this command. The update in the Desired Property is listened by the OCPP Convertor through IoT Hub. Based on the updated Desired Property, OCPP Central System constructs. Authorize.conf and sends the response to EV Charger.
7. **Start Transaction**: In response to successful Authorize.conf, the EV Charger sends StartTransaction.req to OCPP Central System. OCPP Convertor sets IoT Hub Reported Property with necessary data for this command. As it is Device Twin updates, it is routed to Event Hub through IoT Hub message route. Device Twin Processor reads the data and sends to the EV Charge Management system. EV Charge Management system processes the command and provides output. This output should be sent to the EV Charger. Therefore, Device Twin Processor updates the IoT Hub Desired Property for this command. The update in the Desired Property is listened by the OCPP Convertor through IoT Hub. Based on the updated Desired Property, OCPP Central System constructs StartTransaction.conf and sends the response to EV Charger.
8. **Stop Transaction**: EV Charger sends StopTransaction.req to OCPP Central System. OCPP Convertor sets IoT Hub Reported Property with necessary data for this command. As it is Device Twin updates, it is routed to Event Hub through IoT Hub message route. Device Twin Processor reads the data and sends to the EV Charge Management system. EV Charge Management system processes the command and provides output. This output should be sent to the EV Charger. Therefore, the Device Twin Processor updates the IoT Hub Desired Property for this command. The update in the Desired Property is listened by the OCPP Convertor through IoT Hub. Based on the updated Desired Property, OCPP Central System constructs StopTransaction.conf and sends the response to EV Charger.