Zaptec

Integrating with Zaptec charging systems

ZapCloud 6.5.8, revision 2022.06.15

Table of Contents

Zaptec Pro and Home charging stations are cloud connected EV charging stations. Cloud connection allows multiple charging stations to work in collaboration to maximize efficiency. It also enables a range of other options like payment solutions and smart house integration.

Currently there are four ways to interact with, and receive information, from charging stations that will be detailed in this document:

1. ZapCloud API
2. ZapCloud web hooks
3. Message subscription
4. OCPP-J 1.6

To enable the integration options discussed in this document, your charging stations need to be connected to the internet. This can be done in multiple ways, please refer to the appropriate installation manual for available options. It also needs to be configured as part of an EV-charger installation in Zaptec Portal (https://portal.zaptec.com). The installation reflects the physical circuits and other limitations enforced for the group of chargers. When a charger is in use, this configuration and data received from other charging stations in the installation are used to continuously monitor, control, and optimize the installation.

ZapCloud API

The ZapCloud API contains methods to request information related to Zaptec charging stations and their installations. It also contains methods to allow 3rd parties to adjust some runtime parameters. The API is REST based and uses OAuth for authentication.

Authentication

The API methods need to be called with a valid OAuth bearer token. This token is obtained by posting the following as `application/x-www-form-urlencoded` data to: https://api.zaptec.com/oauth/token:

- `grant_type` = `password`
- `username` = `{your username}`
- `password` = `{your password}`

For a successful request an `access_token` will be provided. This token needs to be provided through the Authorization header for any API requests:

```
Authorization: Bearer {access_token}
```

Most methods in the API requires a user with owner permissions for the installation and/or charger. If you're not the owner of the installation, and need to use these methods, any user with owner role for the installation can grant access.

Refresh tokens

To use refresh tokens, **offline_access** scope must be used.
Ldapwiki: Refresh Token

These two requests can be used to fetch refresh_token and access_token, and refresh the access_token

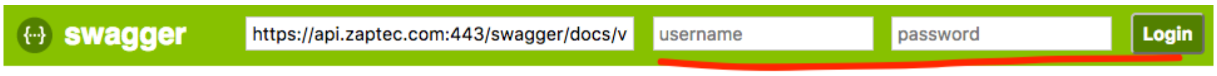
```
###
# These requests can be opened in VS code with REST Client addon installed.
# https://marketplace.visualstudio.com/items?itemName=humao.rest-client
###

@baseUrl = https://api.zaptec.com
@userName = <Insert username>
@password = <Insert password>

# @name auth
POST {{baseUrl}}/oauth/token
Content-Type: application/x-www-form-urlencoded

grant_type=password
&username={{userName}}
&password={{password}}
&scope=offline_access

###

@accessToken = {{auth.response.body.$.access_token}}
@refreshToken = {{auth.response.body.$.refresh_token}}

# @name refresh

POST {{baseUrl}}/oauth/token
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token
&refresh_token={{refreshToken}}
```

API documentation

The current API documentation is maintained at: https://api.zaptec.com/help/index.

Models returned by the API are populated based on the requesting users' roles. Some properties may only be populated for users with certain roles, and some of the documented model properties may never be populated when requested from through the public API endpoints. Refer to the actual returned data to see what your user can access.

The ZapCloud API documentation is interactive, and after logging in using the login field at the top of the webpage, API methods can be executed through the Swagger UI.

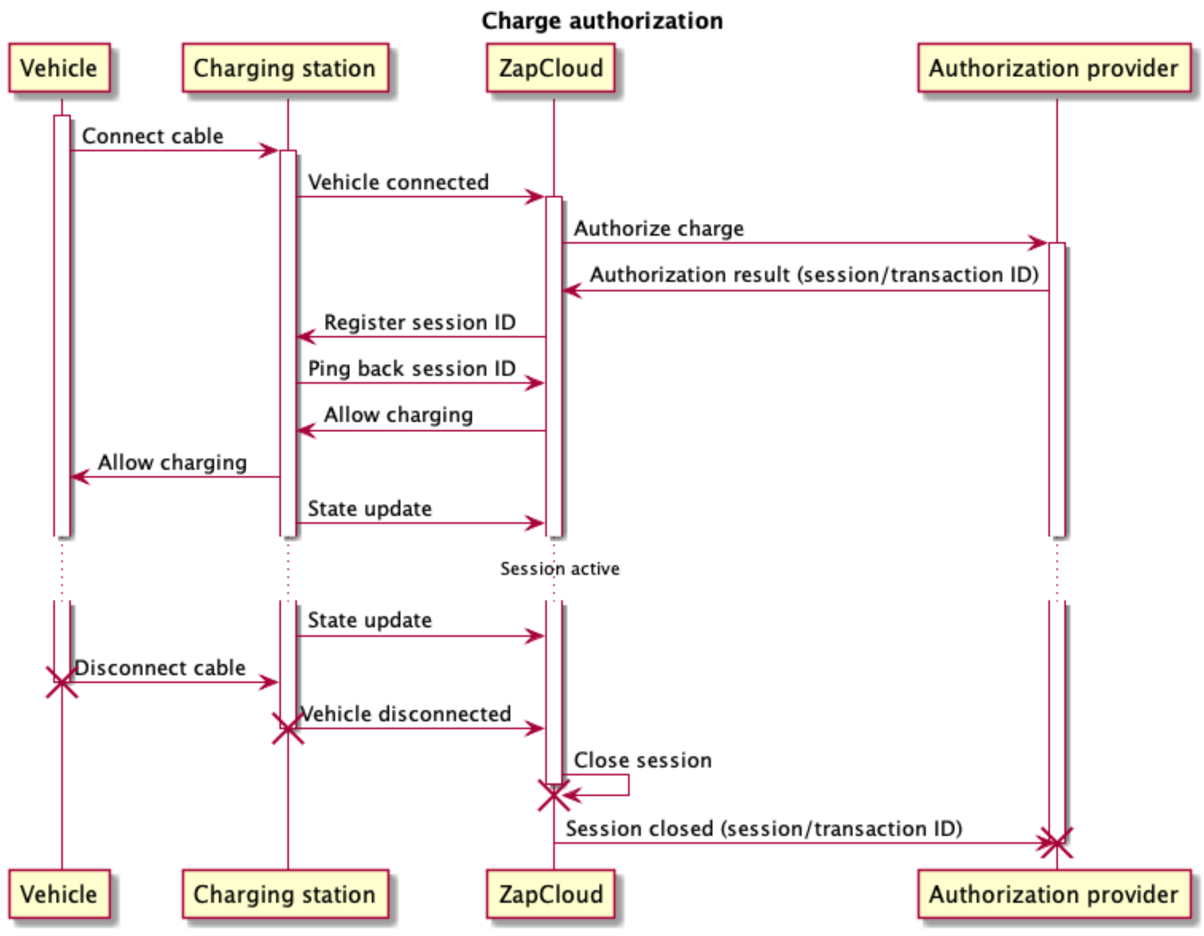


Part of the API is marked as **deprecated**. Do not create integrations using any deprecated methods as these will be removed without warning. For the same reason, if you are already using deprecated method, it is required that you refactor your code as soon as possible.

For API methods returning charger state observations, see the list of supported observation/state IDs in chapter State observation reference. Constants, e.g. valid warnings, state observation and setting ID's etc. can be programmatically resolved from the API's constants file: http://api.zaptec.com/api/constants.

Authorizing charge requests

Before a vehicle connected to a charging station will be allowed to charge, it must be authorized. Currently there are 3 authentication and authorization providers: Zaptec's internal authorization, Web hooks and OCPP-J 1.6. Independent of the authorization method selected for your installation, the following sequence describes communication between vehicle, charging station, ZapCloud and authorization provider:



Authorization providers

- **Internal authorization**
  This is the default authorization option, and by default internal authorization will authorize all charge requests as anonymous charge sessions. It is possible to configure authorization which will prevent charge for all but authorized users. If authorization is required, the user needs to authenticate using RFID token or Zaptec app and needs to have *user*-permission to the charging station or installation, before charge will start.
- **Web hook authorization**
  Using this option, it is possible to allow 3rd parties to authorize charge requests. External authorization and 3rd party payment solutions can be implemented using these web hooks (see more details on the Web hook authorization chapter). Two web hooks can be configured:

    - *Session start* – will be called before a session can start. The 3rd party can control if the session is allowed to start.
    - *Session end* – will be called after a session has completed, with information like session start/end times, and energy consumed.
- **OCPP-J 1.6 Core**
  Authorization is done using OCPP-J 1.6 Core. This allows the installation to be integrated with other OCPP enabled cloud solutions.

Web hook authorization

Web hooks is a simple way to integrate external authorization providers. Before a charge is allowed, a web hook at the remote provider is called (HTTP POST) and charge is only allowed if remote provider authorize the request.

Configuration

Web hook configuration is found in the installation details page when the installation is configured with web hooks authorization.

- **Authentication URL**
  This configuration option is used to configure the URL for an **OAuth** token service. If provided, before posting data to a web hook, an OAuth bearer token will be obtained from this URL. The bearer token will be provided through the Authorization header when posting to the web hooks, i.e.: `Authorization: Bearer {access_token}`. This option is only applicable if calls to web hooks must be authenticated using OAuth.
- **Authentication payload**
  If authentication URL is configured, the payload is provided to the authentication URL to obtain the OAuth bearer token. Format of the payload must match your OAuth token service. The payload is posted to the authentication URL with content type. `application/x-www-form-urlencoded`. Example of a plain OAuth payload: `grant_type=password&username={username}&password={password}`

  If authentication URL is not provided, username and password from the payload will be provided in the Authorization header when posting data to the web hooks (**HTTP basic**). For HTTP basic mode, the payload must be provided as a query string with username and password, i.e.: `username={username}&password={password}`
- **Session start URL**
  The web hook URL to call before a session is authorized to start. Sessions can be denied starting, depending on the result of the request. If external authorization is enabled and this web hook is not provided, the installation will use ZapCloud internal authorization.
- **Session end URL**
  The URL to call after a charge session is finished (vehicle was disconnected from the charging station)

> Though webhook authentication can be omitted, production web hooks should always require authentication and be served from an HTTPS endpoint.

Web hooks

Web hooks are configurable HTTP methods that is called by ZapCloud at certain points in the charge process. The individual web hooks are detailed below. Please note that all web hooks should respond with the appropriate JSON payload, as detailed below, using `content-type: application/json`.

Session start

The session start web hook will be called when a user initiates a charge by connecting an EV to a charging station. The hook will be called with information on what charger and installation is requesting charge, and an optionally scanned RFID token code to identify the user. The 3rd party web hook implementation must look up the user based on the provided RFID token, or by linking the user to charger using other options, e.g. through a 3rd party app, SMS or similar. If charge is authorized the 3rd party system must create and return a unique UUID[1] session identifier to authorize the session. This identifier will be used to reference the session in the ZapCloud database and in further communication. If the request is not authorized, the service should return a HTTP 401 response. For other failure situations not related to authorization or authentication, the service should send an appropriate failure status code (>=400).

For as long as the service returns HTTP 401 responses, the web hook will be polled every 10 seconds until one of these events occur:

- First failed authorization attempt with RFID token
- Poll timeout of 5 minutes have elapsed
- Charger is disconnected from vehicle

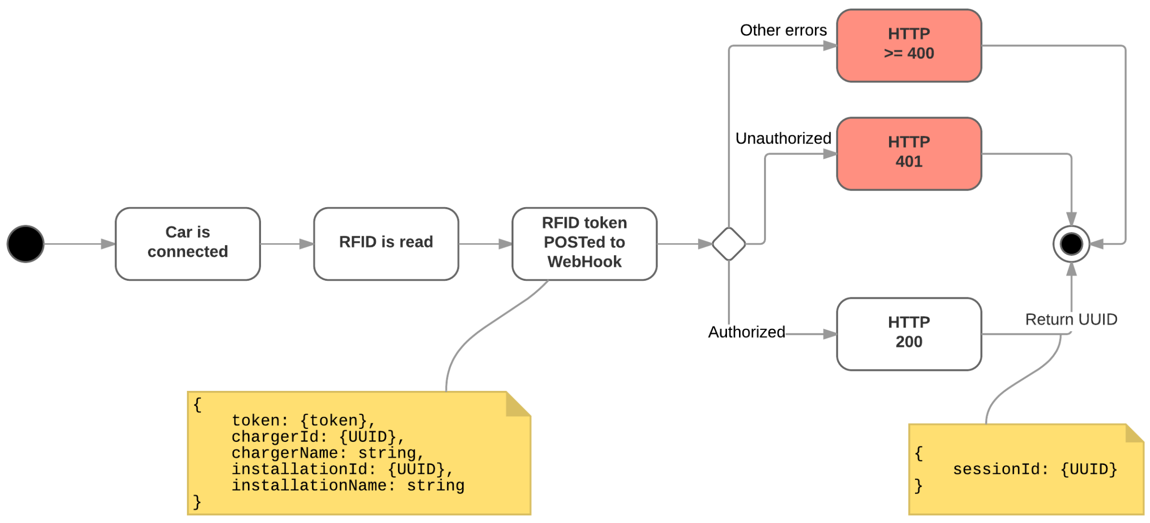
The user must scan RFID token after connecting to the charger. Because of this 3rd parties requiring RFID tokens will get session start requests without token code until this have been scanned. If RFID token is required, the 3rd part web hook should return 401 unauthorized to allow the authorization to be retried.

Any HTTP error code other than 401 will immediately deny charge and stop authorization. Because of this, status codes other than 200 and 401 should only be used in abnormal error situations.

There is no display on the charging station and we have no way of informing about rejection-causes, other than flashing the LED Z-logo. Hence there is no need for the web hook to return user messages. A message can however be returned, and Zaptec will log these error messages for debugging purposes. For failed authorization attempts the charger Z-logo will flash red. 3rd parties authorizing users through their own apps, can provide more detailed messages in the app.

Web hook communication flow:

[1] https://en.wikipedia.org/wiki/Universally_unique_identifier



```
{
    token: {token},
    chargerId: {UUID},
    chargerName: string,
    installationId: {UUID},
    installationName: string
}
```

```
{
    sessionId: {UUID}
}
```

Session end

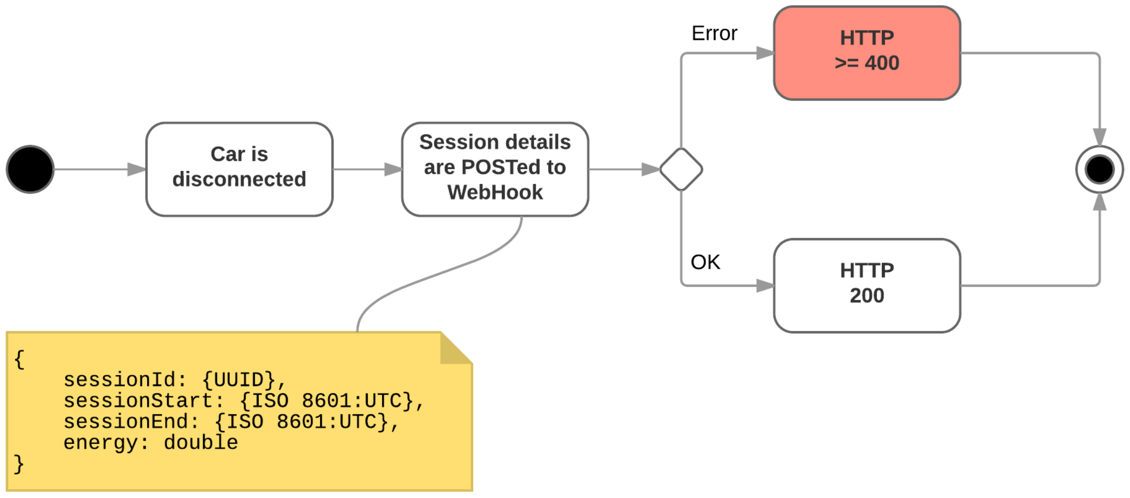
For pure authorization purposes, the charge start web hook is all that is required. If, in addition, payment should be processed after a charge session or a 3rd party needs to maintain a charge log, more information is required to be posted to the 3rd party system. This is done using the session end web hook.

After a charge session is completed (i.e. vehicle is disconnected), information about the session is posted to the configured web hook. The web hook will be provided the following details:

- **Session ID**
  This is the UUID created when authorizing the session, either by 3rd party or ZapCloud.
- **Session start time**
- **Session end time**
  Note: session ends when vehicle is disconnected, not when it was fully charged. The reason for this is that after a charge is initially completed, the car may at any point request more power, e.g. for cabin or battery heating.
- **Consumed energy**
  kWh

Web hook communication flow:

Zaptec



```
{
    sessionId: {UUID},
    sessionStart: {ISO 8601:UTC},
    sessionEnd: {ISO 8601:UTC},
    energy: double
}
```

The web hook is called after the session have been closed in the Zaptec database. If session end call fail, retries will be done periodically until success or timeout. Session status can at any point be requested from the API, see more details in section API: Session end.

For failures, we recommend that a detailed error message be returned in addition to the HTTP status code. This will be logged and better allow us to debug failure scenarios.

API: Session end

In addition to the below web hooks, an API method is provided to check the status of any given session. This can be used to check the status of a session, or request details of sessions whose session end requests have failed.

Sessions are referenced by their UUID session ID and the API method returns the same data as provided to the session end web hook.

More details about the session API can be found here:
https://api.zaptec.com/help/index#!/Session/Session_Get

Other details

- Serialization:
  - Dates are provided as ISO 8601 UTC strings: `2017-02-06T09:56:25Z`
  - UUID's are provided and expected as strings in the following format (.NET GUID): `123e4567-e89b-12d3-a456-426655440000`
- We do not expect users to be common across ZapCloud and the integrating party. The solution allows 3rd parties to integrate and authorize their own users, without any user synchronization between systems.
- For load balancing ZapCloud needs to know the topology of the installation, it's circuits and chargers. Even though, for authorization purposes, this may also need to be partly maintained in a 3rd party system, these details must exist in ZapCloud.
- Charge sessions authorized through 3rd parties will be created and stored in the ZapCloud charge session database. These sessions will be visible in ZapCloud as part of the statistics/charge history for the installation. Because we have no details of the users, **these sessions will be anonymous**. If there is a need for user specific statistics or logs, this will have to be provided by the 3rd party system.

OCPP authorization

Enabling OCPP authorization allows charging stations in the installation to connect to an OCPP cloud for authorization and other OCPP features. Currently most of OCPP-J 1.6 Core is supported.

OCPP integration is mainly aimed at external payment operators, and all methods required for session management and payment are implemented.

Supported OCPP methods

- Charge point to cloud:
  - Authorize
  - BootNotification
  - Heartbeat
  - MeterValues
    - By default, meter values are sent every 2 minutes when charger has an active transaction. Interval can be changed using configuration key `MeterValueSampleInterval`. Set to 0 to disable meter values.
  - StartTransaction
  - StopTransaction
  - StatusNotification
- Cloud to charge point
  - ChangeAvailability
  - ChangeConfiguration
  - ClearCache
  - GetConfiguration
  - RemoteStartTransaction
    - IdTag set using this method will time out after 120s

    - RemoteStopTransaction
    - Reset
    - UnlockConnector
        - Provided but non-functional (returns UnlockStatus.NotSupported)
    - GetDiagnostics
        - Charger will upload a set of pre-defined observations to the provided FTP or HTTP POST location
    - UpdateFirmware
        - Location is ignored and firmware will always be downloaded from ZapCloud. RetrieveDate and Retries are ignored, and firmware update will be instantly triggered
    - SendLocalList
    - GetLocalListVersion

Custom configuration keys

Zaptec charger's support the following custom OCPP configuration keys:

- FreeCharging
  If this config option is set to true, the charger will skip Authorize-call before starting a transaction and will go directly to StartTransaction. When free charging is enabled, the StartTransaction's idTag will be populated with an empty string, or the configured default idTag. The default idTag can be set from Zaptec Portal, or through the FreeChargingIdTag configuration key.
- FreeChargingIdTag
  The idTag to use in free charging mode. See more in the FreeCharging custom configuration key.
- LockCablePermanently
  If this config is set to true, the connected cable will permanently lock to the charging station. It will stay permanently locked until the setting is set to false, or it is unlocked in Zaptec Portal or app.

Configuring Zaptec Pro/Home chargers for OCPP

OCPP configuration is found in the installation details page when the installation is configured with OCPP authorization.

- **URL**
  The websocket URL used by chargers for connecting to the OCPP cloud. `{deviceId}` will be replaced with the charger's lowercase device Id (serial no.) when connecting.

  Example:
  For charger ZCS000143, `ws://ocpp.example.com/devices/{deviceId}` will at runtime be expanded to `ws://ocpp.example.com/devices/zcs000143`.
- **Initial device password**
  For large installations it may not be practical to manually set external password on each charging station. The initial password is a common password used by all charging stations in the installation, and allows the OCPP cloud to remotely set a new

password after the initial connection. This process is described in chapter **6.2.2. Charge point authentication** of the OCPP-J 1.6 specification.

- **Default id tag**
  ID tag used in StartTransaction when authentication is disabled. If no default id tag is provided an empty string is used. *Please note that this option is currently only used for installations using the beta backend.*

When OCPP is enabled for the installation, additional configuration options are enabled for charge points:

- **URL**
  Used when the device serial no. cannot be used when connecting to the OCPP cloud. Specify the full URL the charge point will use when connecting.
- **Password**
  The password used when connecting the charging station to OCPP

Passwords for OCPP (installation *Initial device password*, and charger *External password*) are required to be 20 bytes. The password need to be provided as a hex string, with a maximum length of 40 characters. Shorted passwords are allowed, and will be 0-padded to 40 characters.

WebSocket ping/pong

By default, WebSocket ping is disabled from charger. If connecting to an OCPP cloud that does not provide WebSocket ping functionality, charger ping should be enabled through setting OCPP configuration key `WebSocketPingInterval` to a value greater than 0 (ping interval in seconds).

Charging station ping is currently sent as a unsolicited pong frame, see chapter 5.5.3 Pong in rfc6455.

Message subscription

An installation can be configured with an AMQP (Azure Service Bus) message subscription. 3rd parties can connect to subscriptions to receive continuous push notifications when charging station's state changes. State observations are details such as charge mode, charge current, session energy etc. A list is observations provided to messaging subscriptions can be found in chapter State observation reference.

Message subscriptions must be activated per installation. This is done through the "Message subscription" option under advanced installation settings in Zaptec Portal. Please note that subscriptions are automatically removed after 14 days without use.

Enabling a subscription will configure an Azure Service Bus Topic for your installation. Messages received from installation chargers will be broadcast to this topic.

There is a wide range of options available for receiving messages from Azure Service Bus: https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-how-to-use-topics-subscriptions. In addition to using Microsoft's Service Bus libraries, it is also possible to consume messages using the standard protocol AMQP 1.0 (https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-amqp-overview).

Credentials

After messaging subscription is enabled connection details can be retrieved either:

- **For production purposes:**
  - Connection details **must** be retrieved from this API method: https://api.zaptec.com/help/index#!/Installation/Installation_GetMessagingConnectionDetails; accessible by users with installation owner or service roles.

As connection credentials and host can change at any time, it is important to query message subscription connection details from our API whenever you are connecting!

- For development or testing purposes:
  - Connection details can be manually retrieved in Zaptec Portal advanced installation settings; accessible by users with installation owner or service roles.

Connection details can be combined as a Service Bus connection string:

Endpoint=sb://**{Host}**/;SharedAccessKeyName=**{UserName}**;SharedAccessKey=**{Password}**;EntityPath=**{Topic}**

Messages are published to a topic, but can only be received from the topic subscription[2]. Depending on the library used to connect to the service bus topic, you may need to combine the topic and subscription name in your connection settings. The full name of the topic subscriptions is: `{Topic}/subscriptions/{Subscription}`

> Note that message subscriptions will be disabled after 14 days without messages being sent to the subscription (e.g. chargers are offline), **or** without messages being consumed.
>
> If your subscription has been disabled you need to manually re-enable this though Zaptec Portal or by requesting credentials from our public API: http://api.zaptec.com/help/index#!/Installation/Installation_GetMessagingConnectionDetails. Please note that connection details will be reset when subscription is disabled.

> Message TTL is 5 minutes. Meaning that any state observation not consumed within 5 minutes will be lost.

Message format

Messages are provided as JSON-serialized `ChargerState` objects (**https://api.zaptec.com/help/index#!/Charger/Charger_ChargerState**), containing:

- **`ChargerId`**
  The UUID of the charger providing the message
- **`StateId`**
  The ID of the changed state observation – list of supported state observation Id's can be found in chapter State observation
- **`Timestamp`**
  The UTC timestamp when the state observation was changed, provided as an ISO 86013 string
- **`ValueAsString`**
  The new state value, serialized as a string
    - Boolean: `true` = `1`, `false` = `0`

[2] https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-nodejs-how-to-use-topics-subscriptions#what-are-service-bus-topics-and-subscriptions

[3] https://en.wikipedia.org/wiki/ISO_8601

State observation reference

List of available states and valid value ranges for enum's can be found in the API's constants definition: https://api.zaptec.com/api/constants. Please also note that there are some differences in state observations provided for Zaptec Smart-family of chargers (Pro/Home), and Zaptec Go. The full list of observations available for these chargers can be found in enum's `Schema.Smart.ObservationIds` (Zaptec Pro/Home) and `Schema.Apollo.ObservationIds` (Zaptec Go).

Id	*Description*
-2	**`IsOnline`** • 1: charger is online • 0: charger is offline
150	**`Network communication mode`** Wifi, LTE, PLC or none
151	**`PermanentCableLock`** • 0: cable is not permanently locked to charging station • 1: cable is permanently locked to charging station
201	**`TemperatureInternal5`** Internal temperature sensor 5 in degrees centigrade
202	**`TemperatureInternal6`** Internal temperature sensor 6 in degrees centigrade
270	**`Humidity`** Internal humidity in percent
501	**`VoltagePhase1`** Output voltage phase 1 in volts
502	**`VoltagePhase2`** Output voltage phase 2 in volts
503	**`VoltagePhase3`** Output voltage phase 3 in volts
507	**`CurrentPhase1`** Output current phase 1 in amperes
508	**`CurrentPhase2`** Output current phase 2 in amperes
509	**`CurrentPhase3`** Output current phase 3 in amperes
513	**`TotalChargePower`** Total instant charge power in watts
519	**`SetPhases`** • 1: TN phase 1 • 2: TN phase 2 • 3: TN phase 3 • 4: TN phase 1/2/3 • 8: IT phase 1 • 6: IT phase 2 • 5: IT phase 3

553 **`TotalChargePowerSession`**
Total **energy** delivered in the current charge session in kWh

554 **`SignedMeterValue`**
Aggregated meter value for the charging station.

708 **`ChargeCurrentSet`**
The allocated charge current for SetPhases in amperes

710 **`ChargerOperationMode`**

- 1: no vehicle connected to charging station
- 2: vehicle connected; requesting to charge
- 3: vehicle connected; charging
- 5: vehicle connected; finished

711 **`IsEnabled`**

- 1: charger is enabled
- 0: charger is disabled

712 **`IsStandAlone`**

- 1: charger is in system mode and is controlled by Zaptec's backend
- 0: charger is in stand-alone mode and is not backend controlled

714 **`CableType`**
Ampere rating of the connected cable.

715 **`NetworkType`** (electrical grid)

- 1: IT 1 phase
- 2: IT 3 phase
- 3: TN 1 phase
- 4: TN 3 phase

721 **`SessionIdentifier`**
The current session Id (GUID/UUID)

804 **`Warnings`**

809 **`CommunicationSignalStrength`**
For LTE signal strength is provided in signal percent, of Wifi RSSI:

- >= -70: reliable network connection (recommended) [4]
- < -70: minimal signal for connection, packet loss may occur
- < -80: unreliable connection, charger may randomly disconnect
- < -90: unusable

911 **`SmartComputerSoftwareApplicationVersion`**
Charger firmware version

[4] https://support.metageek.com/hc/en-us/articles/201955754-Understanding-WiFi-Signal-Strength

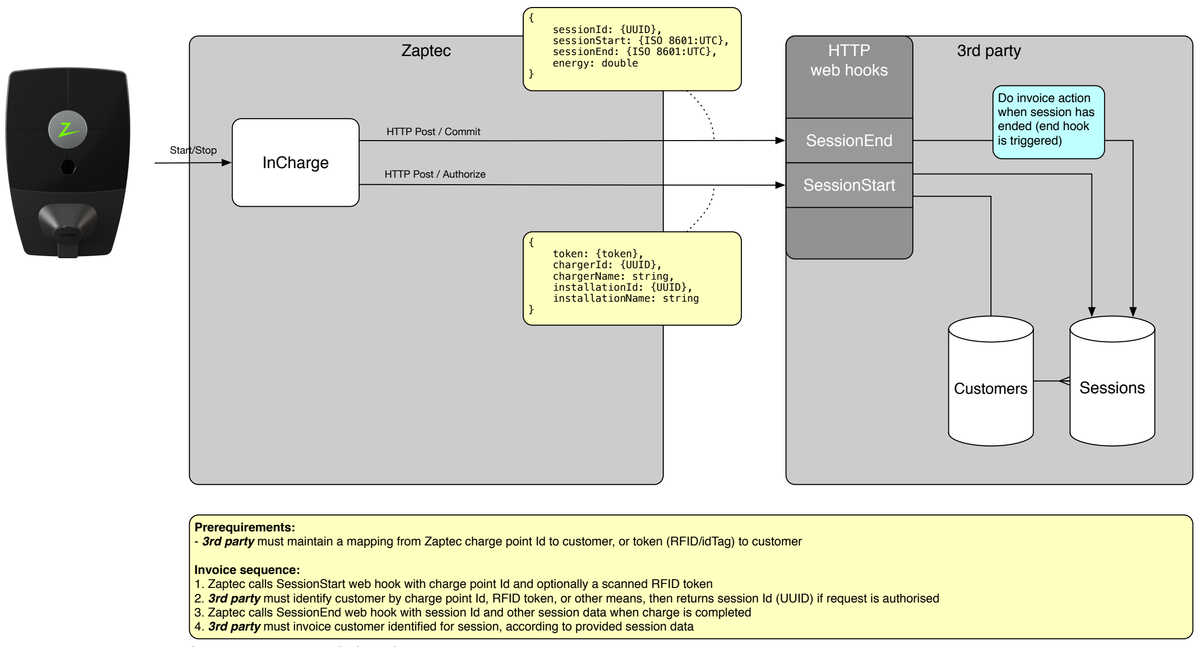
Common use cases

Authorization and payment solutions

Payment solutions, or solutions where an external system should authorize users, can be built using the ZapCloud web hooks discussed in chapter Web hook authorization, or by using ZapCloud's OCPP-J 1.6 Core integration option.

- Web hooks
  - The integration should expose two OAuth 2.0 or HTTP basic authenticated HTTPS end points:
    - **Session start**
      Will be called before charge is authorized. The 3rd party system decides whether the user is allowed to charge
    - **Session end**
      Called after a charge session has ended, with details on the charge session. Data can be used for calculating a cost for the charge session.
- 3rd party payment solution using ZapCloud native users/authentication
  - User and session are maintained in ZapCloud
  - 3rd party can periodically call Zaptec's session/charge history API to query completed sessions for invoicing
- OCPP-J 1.6
  - Build an OCPP-J 1.6 enabled cloud solution for authorization or payment, then configure the Zaptec installation to authorize using your solution
  - You assume control over session lifecycle and can do invoicing

Below we outline pre-requirements and invoice process for native and web hooks authorization.



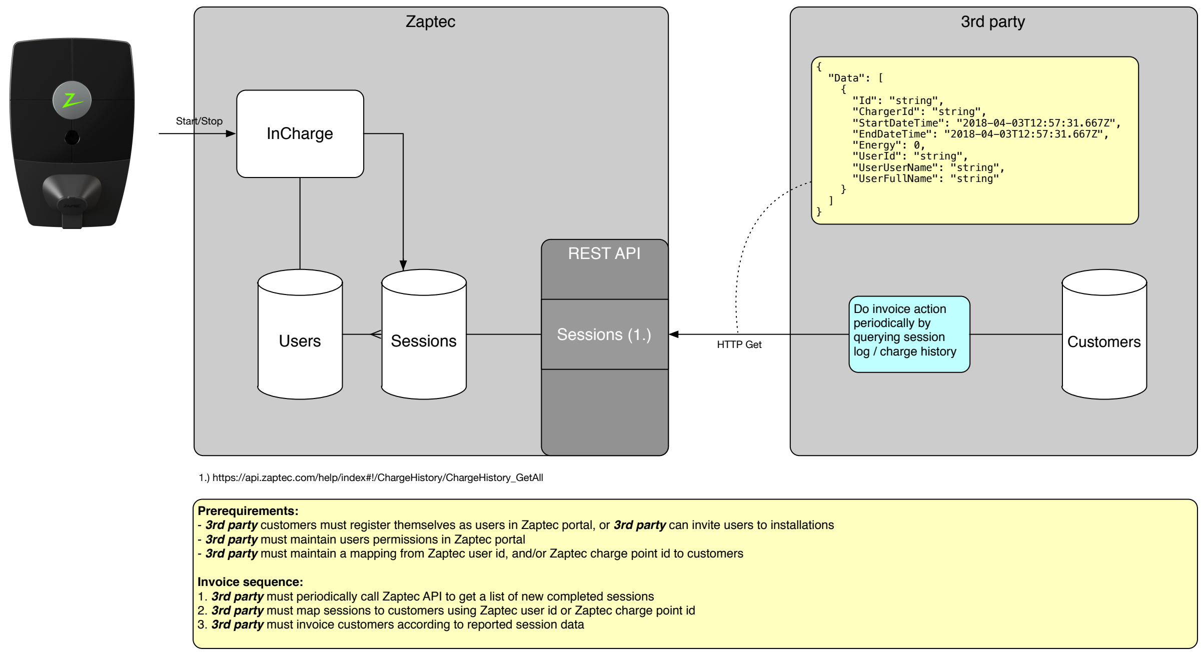
```
{
    sessionId: {UUID},
    sessionStart: {ISO 8601:UTC},
    sessionEnd: {ISO 8601:UTC},
    energy: double
}
```

```
{
    token: {token},
    chargerId: {UUID},
    chargerName: string,
    installationId: {UUID},
    installationName: string
}
```

Prerequirements:
- ***3rd party*** must maintain a mapping from Zaptec charge point Id to customer, or token (RFID/idTag) to customer

Invoice sequence:
1. Zaptec calls SessionStart web hook with charge point Id and optionally a scanned RFID token
2. ***3rd party*** must identify customer by charge point Id, RFID token, or other means, then returns session Id (UUID) if request is authorised
3. Zaptec calls SessionEnd web hook with session Id and other session data when charge is completed
4. ***3rd party*** must invoice customer identified for session, according to provided session data

Figure 1; payment solution using web hooks



```
{
  "Data": [
    {
      "Id": "string",
      "ChargerId": "string",
      "StartDateTime": "2018-04-03T12:57:31.667Z",
      "EndDateTime": "2018-04-03T12:57:31.667Z",
      "Energy": 0,
      "UserId": "string",
      "UserUserName": "string",
      "UserFullName": "string"
    }
  ]
}
```

1.) https://api.zaptec.com/help/index#!/ChargeHistory/ChargeHistory_GetAll

Prerequirements:
- ***3rd party*** customers must register themselves as users in Zaptec portal, or ***3rd party*** can invite users to installations
- ***3rd party*** must maintain users permissions in Zaptec portal
- ***3rd party*** must maintain a mapping from Zaptec user id, and/or Zaptec charge point id to customers

Invoice sequence:
1. ***3rd party*** must periodically call Zaptec API to get a list of new completed sessions
2. ***3rd party*** must map sessions to customers using Zaptec user id or Zaptec charge point id
3. ***3rd party*** must invoice customers according to reported session data

Figure 2; payment solution using native authentication

Dynamic load balancing

A charging station's charging current and phase is dynamically controlled by ZapCloud. The optimal charge configuration is calculated based on the charger's installation properties and runtime state of other charger's is the installation. This is done transparently, always ensuring that as many chargers as possible is providing as much charging power as possible.

In some scenarios 3rd parties may want to limit the charge power available for an installation. E.g. to limit EV-charge power during costly peak hours, or prevent circuit breakers tripping in other high load scenarios. This can be done through setting `AvailableCurrent` using the installation update API method: https://api.zaptec.com/help/index#!/Installation/Installation_ExternalUpdate.

Dashboards

Using Message subscriptions it is possible to build rich live dashboards and widgets that aggregate and present key metrics of one or more Zaptec installations. Messages can be received using Azure Service Bus libraries or AMQP 1.0 and data can be integrated with a wide range of programming languages and solutions.

The live data can be used together with API methods like https://api.zaptec.com/help/index#!/Charger/Charger_ChargerState, that provide an instantaneous snapshot of the chargers current state, to provide a complete live representation of the installation and chargers.