Requirements Document

Prog-X Asset Tracker

Version: **1.01.** Date: **Nov-16-2021**

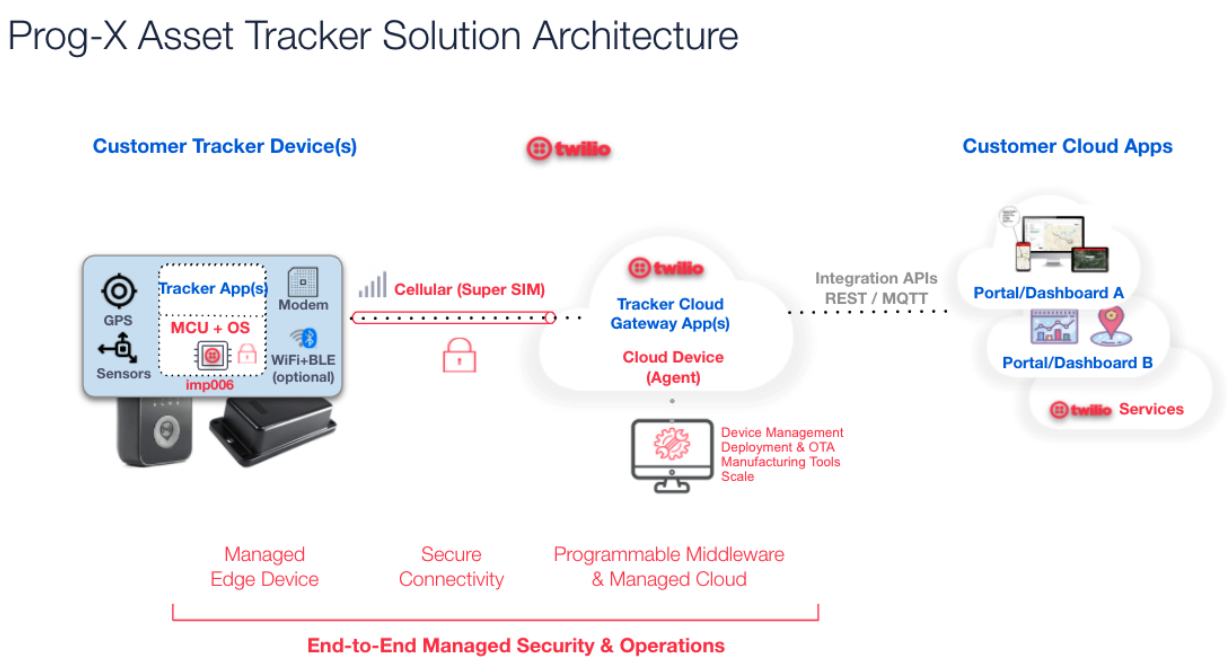
EXTERNAL REVIEW

CONFIDENTIAL

Solution Architecture

The Twilio Electric Imp platform allows to readily define a solution architecture with the following components:

- Tracker device with sensors and communication interfaces, based on imp006 module with impOS
- Updatable tracker device application(s)
- Global cellular cloud connectivity via Twilio Super SIM
- Twilio Electric Imp managed cloud and programmable middleware (cloud agent)
- Updatable cloud agent (gateway application) with integration one or more cloud tracking portals/dashboards
- Tracker portals/dashboards (outside of project scope)



Project Timeline & Deliverables

Proposed Timeline*

Sep-Oct 21	● Agreement on engagement ● Initial requirements definition ● Non-binding commitment based on success criteria ● Work on PoC readiness starts ● Scoping & preparation
Nov 21	PoC development: ● Design & development work
Dec 21	PoC phase [“Will it work”]: ● PoC hardware (based on imp006 breakout) ● PoC device software and simplified cloud integration (joint) ● PoC testing by Lead Customer (doesn’t include *HW* durability, approx. 2 weeks duration)
Jan 22	Field trial development: ● Finalized requirements ● Design & development work ● Final simplified cloud integration (joint) ● Pre-production hardware design & prototypes
Feb 22	Field Trial phase [“Does it Work”]: ● Initial device software and cloud integration blueprints ● If necessary, blueprint customization for Lead Customer ● Device manufacturing & testing software blueprint ● Pilot production run with selected ODM ● Field Trial testing by Lead Customer ○ Field trial size: Approx. 100 units ○ We ask Lead Customer purchase field trial units at cost ○ Approx. 4-6 weeks duration ○ Note: With 100 units, this may already count as pilot rollout
Apr 22	Contract and Production Pilot phase: ● Final production hardware design + certifications
Jun 22	Production Ramp and Rollout: ● Production run with Lead Customer CM ● Sale and rollout through Lead Customer

* Production schedule is dependent on hardware availability/lead times and ODM schedule

High-Level Product User Story (Informative)

This is the high-level user story on how a typical end user (tracking customer) would interact with the product, focused on the device/IoT platform portion of the functionality:

1. Manufacturing / storage / shipping
    - Tracker is tested, batteries inserted, pre-provisioned, and set to ‘shipping mode’ (permanent sleep)
    - Tracker is stored in warehouse, then shipped to customer (user)
2. User installation
    - User unpacks tracker and mounts it on asset
    - User pushes user button to activate tracker (exits ‘shipping mode’) **(not in PoC)**
    - User uses mobile/web app (note: part of customer solution) to enter serial number or scan QR code and associate tracker to account and asset
    - Note: Mechanism to associate tracker with asset is still being investigated - this is not part of project scope.
3. User uses tracking system via mobile/web app, with the following possible operations:
4. Device configuration (based on tracker ID/serial number)
    - User views last configuration values in tracking portal. User is able to force re-reading configuration values from device to ensure the tracking portal reflects the actual configuration state of the device.
    - User sets individual configuration values in tracking portal. Possible configuration values include (list may be incomplete):
        - Sensor reading period
        - Normal reporting period
        - Alerts on/off & thresholds (for tamper detect, sensors, battery, etc)
        - **(not in PoC)** On-device geofence on/off & center/radius
        - Tracking mode on/off & reporting period
        - **(not in PoC)** Repo/theft mode on/off & reporting period/schedule
        - **(not in PoC)** Known BLE beacons and locations
    - User saves configuration state in tracking portal. All changes are queued for the device and marked as pending in tracking portal. When device connects, changes are pushed and changes confirmed as applied by device are marked as confirmed.
5. Normal reporting mode
    - User views tracker data (tracker ID, last connection, position, sensor values, battery level) on tracking portal. Values are updated on tracking portal when tracker reports periodically.

6. Alerts
    - User configures alerts in tracking portal (tamper detect, sensor alerts, battery alerts, geofence alerts, etc)
    - If alerts trigger on device they are shown in tracking portal in semi-real time
7. **(not in PoC)** On-device geofencing
    - User enables on-device geofencing in tracking portal by configuring geofence (center, radius)
    - If device enters/exits geofence, alert is shown in tracking portal
8. Tracking mode
    - User enables tracking mode in tracking portal by configuring tracking (reporting period)
    - If device enters/exits tracking mode, alert is shown in tracking portal along with position updates
9. **(not in PoC)** Repo/theft mode
    - User enables repo/theft mode in tracking portal by configuring repo/theft (schedule)
    - If device enters/exits repo/theft mode, alert is shown in tracking portal along with position updates

Hardware Requirements

PoC Phase Hardware

imp006 breakout board

- BG96 (LTE-M1 + 2G fallback), embedded Super SIM (Note: BG96 GNSS used for only for initial PoC)
- Added ublox M8N module incl. antenna, attached via UART
- Temp/humidity sensor (humidity not used)
- Accelerometer
- Light sensor (BlinkUp sensor)
- WiFi/BLE: Passive positioning via WiFi APs (2.4 GHz) and BLE beacons
    - ~~Initially use on-board 1MW~~
    - Added ESP32-WROOM-32 attached via UART, e.g. https://www.mikroe.com/wifi-ble-click
- LEDs
    - BlinkUp status
    - Programmable status LED for user notifications
- No wake button
- Rechargeable 2000 mAh LiPo battery
- Battery level monitoring (initially via voltage level. If not accurate enough, need to consider battery gauge via coulomb counter)

- Antennas: Cellular, GNSS, WiFi/BLE (for passive reading)
- Existing NewAge IP67 enclosure

Field Trial/Production Hardware

- ~~Based on imp006, BG96 (LTE-M1 + 2G fallback), embedded Super SIM, BG96 GNSS~~
- Based on tracker design, BG95-M3 (LTE-M1 + 2G fallback), embedded Super SIM, (Note: BG95-M3 GNSS not used)
- ublox GNSS module incl. antenna, via UART
- On-board sensors:
    - Using temperature sensor in LIS2DH12 (LISDH12TR) (Note: This is just for basic on-board temperature sensing, so limited accuracy is OK. No HTS221 on device)
    - Accelerometer (LIS2DH12)
    - Light sensor at bottom (light/dark detection, to detect tampering/removal of unit). Plan to use `hardware.lightlevel()`
- WiFi/BLE: Passive positioning via WiFi APs (2.4 GHz) and BLE beacons
    - ESP32 module with ESP-AT
- LEDs
  Note: In this order: BlinkUp status LED should be middle LED to simplify identification
    - Programmable status LED for user notifications
    - BlinkUp status LED (middle)
    - Modem netlight LED
- User Button
    - Programmable button for user input (wake), connected to imp006 ‘wake’ pin
- Operation on 3xAA or 4xAA batteries (Energizer Ultimate Lithium) for up to 5 years (based on battery type and application behavior)
- Battery level monitoring (method to be confirmed, e.g. voltage or coulomb)
- Antennas: Cellular, GNSS, WiFi/BLE (for passive reading)
- Designed for high-volume manufacturing
- Proposed housing (IP6X+ water/dust resistance)
  http://www.kingwoiot.com/index.php/Product/detail/cat_en_name/nt07e.html

Target Hardware BOM Cost
Estimates at 1k units: $40-$50, based on design, manufacturing, and other factors

Device Software Requirements

Note that the listed application behavior is meant as a description of the Software Blueprint Twilio is providing, specific additional functionality required by the customer can be added as a separate project.

General software requirements

- Modular design and source code to support easy customization
- Initially, the lead customer portal is chosen for integration, but the integration must be modular as the open source version will likely use a different portal (e.g. a demo portal)
- Software must be free of 3rd party rights or -- if 3rd party code is used -- such code must be clearly marked and must use MIT license
- For production software **(not PoC or Field Trial)**
    - basic developer documentation of high-level design, modules, and public APIs
    - must be ready to be published with MIT license on github

Quality Metrics

Issue Level Definition

For the project, the following simplified definitions of issues will be used:

- **P1: Critical.** Business critical functionality unavailable/unusable. No workaround possible.
- **P2: Degraded**. Key functionality impacted or unreliable. Workaround may be available but impacts product quality/user experience.
- **P3: General.** Other functionality impacted
- **P4: Minor.** Minor or cosmetic.

PoC Phase

- PoC hardware (imp006 breakout)
- PoC-level functionality: See requirements
- Basic manual testing of key functionality
- <= 2 P1 issues, <= 7 P2 issues

Field Trial Phase

- Initially imp006 breakout, then field trial/production hardware
- Field Trial-level functionality: See requirements

- Testing of core functionality: API automated unit testing, manual integration testing, end-to-end system and usability testing
- Zero P1 issues, <= 3 P2 issues, <= 10 P3 issues

Production Phase

- Production hardware
- Production-level functionality: See requirements
- Comprehensive testing: API automated unit testing, manual integration testing, end-to-end system and usability testing
- Zero P1 issues, <= 1 P2 issues, <= 5 P3 issues

Application Functionality

Phase	Description
PoC Field Trial Production	**Device configuration state** ● Device can be remotely configured with a set of parameters ● Configuration changes are queued on cloud agent. When device connects, changes are pushed to device and all applied configuration changes are acknowledged back to cloud agent (to ensure device configuration is in sync with cloud agent). ● Ability to query complete state from cloud agent (to confirm current configuration) ● Possible device configuration values include (TBC): ○ Sensor reading period (TS) ○ Normal reporting period (TR) ○ Tamper detect on/off ○ Movement detection (motion threshold, continuous motion duration) ○ Alerts on/off & thresholds (for sensors, shock, battery, etc) ○ **(not in PoC)** On-device geofence on/off & center/radius ○ Tracking mode on/off & reporting period (TT) ○ **(not in PoC)** Repo/theft mode on/off & reporting period/schedule ○ **(not in PoC)** Known BLE beacons and locations ○ **(not in PoC)** Log level

PoC Field Trial Production	**Battery-efficient operation** ● Sleeps between operations to maximize battery life ● **(not in PoC)** Shallow sleep only (to preserve black box recorder) ● Cellular on only for cloud connection ● Peripherals on only when needed (sensors, WiFi/BLE, etc.)

Field Trial Production	**Robust/defensive design & operation** ● Battery safeguards ○ Back-off for power-intensive operations: If a power-intensive operation (i.e. cellular connection or acquiring a GNSS fix) fails multiple times in a row then back off and retry less frequently ○ Prevent crash loop from draining battery: If the application repeatedly restarts by accident (e.g. due to an error) then enter recovery mode. Recovery mode halts regular application operation and periodically checks until new application version is available. (maybe also define as “empty configuration mode” where the app does nothing except for periodically check the connection) ● Connection safeguards ○ Implement a minimum and maximum cloud agent connection period to ensure that misconfiguration does not result in very frequent connections or device being inaccessible for long periods. ● Support Super SIM OTA updates ○ Initial support for OTA updates by receiving a flag on the device that instructs the device to use forceSuperSimOTA() and then stay online for 1 minute to allow an OTA update to complete ● Logging ○ Device and cloud code must provide logging output of key information, warnings, and errors, enabled by log level (INFO, WARN, ERROR) to adjust verbosity ○ Log level can be set by configuration parameter

PoC Field Trial Production	**Cloud connectivity** ● Cellular only

Field Trial Production	**WiFi/BLE support** ● ~~On PoC hardware, initially~~ ○ ~~1MW (with impOS support)~~ ● PoC hardware and production hardware ○ ESP module (with ESP-AT driver) ● Need application-level ESP-AT driver, see https://docs.espressif.com/projects/esp-at/en/latest/AT_Command_Set

Field Trial Production	**Shipping mode** ● Device sleeps in ‘shipping mode’ after production to conserve battery ● Upon pressing of user wake button exits ‘shipping mode’ and begins normal operation. This includes a first report with sensor values, location, battery level, etc.

PoC Field Trial Production	**Positioning** ● Positioning is needed indoors (buildings, warehouses, or places with poor GNSS reception) as well as outdoors. ● Position typically only needs to be determined when device is moving, to avoid battery drain (by repeated useless positioning of a stationary device) ● Determine position in this order: ○ **(not in PoC)** BLE beacons (known list of beacons + positions. Works offline. Approx. accuracy: 5m to 15m) ○ GNSS (Works offline. Approx. accuracy: 3m to 20m) ○ WiFi (APs & Google Location API. Approx. accuracy: 10m to 100m) ○ **(not in PoC)** Cellular trilateration (tower(s) & Google Location AP. Approx. accuracy: 100m to 3,000m) ○ Note: Type of positioning needs to be provided with the report ● If a position is found with a method, skip the following methods for that cycle

PoC Field Trial Production	**GNSS** ● Use assist to minimize fix time. Load/cache valid assist data.

PoC Field Trial Production	**Shock/Motion** ● Wake on motion above configured motion threshold ● Send shock alert if acceleration above configured shock threshold ● Ongoing motion means position can/should be reacquired. Ignore sporadic/brief motion, i.e. no position change ● Tracker should not continuously be awake during motion (to preserve battery)

PoC Field Trial Production	**Offline operation** ● If no network connection, data & events are stored for a max of X days on device and reported to cloud agent when connection is regained

Field Trial Production	**On-device geofencing mode** ● Goal of on-device geofence is to preserve battery by avoiding unnecessary position reporting while inside the geofence ● Enabled via configuration setting from cloud agent ● One geofence (center + radius) is sent from cloud agent to device to activate geofence mode ● Geofencing must support offline positioning modes, i.e. GNSS positioning as well as BLE beacon positioning **(not in PoC)**. (Note: This may mean the geofence must be implemented in the application rather than leveraging GNSS built-in geofence). ● If geofence mode is enabled, a position report is sent if the device enters or exits the geofence. Note: Tracking mode is not automatically enabled … see also tracking mode. ● Note: Additional (off-device) geofencing can still be done on tracking portal based on tracking reports

PoC Field Trial Production	**Periodic sensor reading** ● Periodic (every TS seconds, configurable) wake ● Read sensor values (temperature, battery level, etc) ● Check for alerts/thresholds. If alert is required, report to cloud agent (see alerts) ● If no alert is required, queue report for next connection

PoC Field Trial Production	**Alerts** ● Values are checked on every wake ● If alert condition is detected (sensor value is exceeded, tampering (light sensor is triggered), send alert to cloud agent immediately (semi-real time) ● Send further alerts to cloud agent only if conditions have changed (i.e. do not automatically repeat previous alerts)

PoC Field Trial Production	**Normal reporting mode** ● Periodic (every TR seconds, configurable) report of queued information to cloud agent ● Includes current position (use last position if no movement occurred)

PoC Field Trial Production	**Tracking mode** ● Enabled via configuration setting from cloud agent. Upon enabling, acquire position and send a position report to report the current position (even if the device is not moving) ● On movement, periodically (every TT seconds) acquire position and send position report ● When movement stops, send another position report to report the current position ● If on-device geofence is enabled, do not send position report while inside geofence

Field Trial Production	**Repo ("Repossession") / Theft mode** ● Repo / theft mode is like tracking mode, but activated by a previously defined schedule, e.g. automatically activate tracking mode on May 1st. ● (Use case: Asset is expected to be returned on April 30, so starting May 1st it is automatically tracked)

PoC Field Trial Production	**Northbound API & application data** (from cloud agent to tracking portal) ● Periodic reporting: Temperature, battery level, position, etc ● Alerts: Tamper, temperature, battery level, position, etc ● Current device configuration state ● APIs & JSON schema: See section "Cloud Agent Software" below

PoC Field Trial Production	**Southbound API & application data** (from tracking portal to cloud agent) ● Configuration values ● APIs & JSON schema: See section “Cloud Agent Software” below

PoC Field Trial Production	**Device setup/commissioning** ● **(PoC only)** Electric Imp blinkup app to associate user ● **(Field Trial/Production)** tracker ID/QR code + app to associate user

Library functionality

Squirrel libraries to be created/updated as part of the work:

- New: ESP-AT driver
- New: Library for tracking portal integration, including simple REST API simulator (exact functionality/portal to be defined)
- ~~New: Library to manage Super SIM OTA updates~~
- ~~Update: BG96 GNSS library~~
- Update: ublox GNSS library, using ublox assist (as needed)
- Update: Google Maps library (cellular positioning)
- Update: ReplayMessenger library (as needed)
- ~~Update: Low-power manager library (to support shallow sleep)~~

Note: Same quality requirements as the rest of the code.

Out of Scope

- Companion mobile app
  We’re trying to avoid a dedicated mobile (BlinkUp) app since all devices have cellular and can be pre-provisioned in the factory. So the only thing left in the field (by the customer/user) is to associate a tracker with a user. This can be done via tracker ID on label or QR code + web app (hopefully a dedicated mobile app is not needed).

Cloud Agent Software

Initially, the lead customer portal is chosen for integration, but the integration must be modular as the open source version will likely use a different portal (e.g. a demo portal). I.e.:

- Integration with lead customer tracking portal: APIs & schema see below “Simplified Proposal for Cloud Integration”. Note: If AWS integration is required it is expected that existing Electric Imp AWS libraries are leveraged.

- Note: Customer currently has a node.js application running to handle incoming device data and forward to DynamoDB/Postgres. Customer is in the process of reworking the architecture and redefining the APIs and data model, so adjustments are possible. Customer to provide example APIs.
- For open source version: TBD. Possible candidate: https://www.traccar.org

Simplified Proposal for Cloud Integration

Simplified proposal for initial integration, either into existing v1 cloud or upcoming v2 cloud.

Data sending to the cloud (Northbound API & JSON data model)

REST API on the cloud. To be implemented by the customer.

- One endpoint should be enough for the beginning (exact endpoint name can be changed): https://<cloud_api_url>/data
- POST request from imp to send new portion of data.
- Basic authentication (username/password). Can be substituted by an API Key later.
- Body with data in text (JSON) format.
- Response code 200 from the cloud if the request/data is accepted.
- Some fields (keys) in the JSON can be mandatory, some - optional (do not exist in every message) - depends on the current configuration and/or on the event which triggered the message sending.

An example of data msg (all fields are just for example. The exact format/content/names TBD during design/implementation):

```
{
   "trackerId": "c0010c2a69f088a4",
   "timestamp": 1617900077,
   "status": {
      "movement": true,
      "tracking": true
   },
   "location": {
      "type": "gnss",
      "accuracy": 3,
      "lng": 30.571465,
      "lat": 59.749069
   },
   "sensors": {
      "batteryLevel": 64.32,
```

```
    "temperature": 42.191177
  },
  "alerts": {
    "temperatureHigh": true
  }
}
```

Configuration (Southbound API & JSON Data Model)

REST API on imp-agent. To be implemented by Twilio.

- URL of every imp-agent is known after the imp enrollment.
- A couple of requests/endpoints for beginning (exact number and names can be changed):
  - PATCH/POST https://<imp_agent_url>/cfg - pushes a configuration update from the cloud to the imp.
  - GET https://<imp_agent_url>/cfg - returns the latest known actual configuration from the imp to the cloud.
- Response code 202 (TBD) from the imp-agent if the request is accepted (does not mean it is executed).
- If/when imp-device is online, cfg update requests are forwarded from imp-agent to imp-device for execution.
- The cloud can periodically poll the imp-agent (GET request) to obtain the actual known configuration.
- Basic authentication (username/password).
- Body with data in text (JSON) format.
- Some fields (keys) in the JSON can be mandatory, some - optional.
- Configuration update requests can contain full configuration or only parts/fields of configuration which should be updated (assuming these are logically consistent parts).

An example of cfg update request (all fields are just for example. The exact format/content/names TBD during design/implementation):

```
{
  "cfgFormatVersion": "1.0",
  "readingPeriod": 600,
  "reportingPeriod": 43200,
  "temperatureHigh": 40,
  "tracking": false
}
```

- The actual cfg returned to the cloud is always the full configuration.

Device Manufacturing & Test Software Requirements

The manufacturing and test software leverages the Electric Imp connected manufacturing process.

Field Trial Production	**Manufacturing & Test software to be developed** ● Device hardware testing code & PASS/FAIL recording ● Printing of individual label with device information: ○ IMEI ○ ICCIDs ○ Tracker ID serial number (imp deviceID) ○ Tracker ID QR code ● Device blessing (hardware activation) -> ready to ship

Appendix: Planned Testing Checking by Customer

- ❏ Device successfully registers on cell network
- ❏ Device successfully communicates with our server
- ❏ ~~Device accepts and responds to all SMS commands~~ (no SMS support)
- ❏ Device accepts and responds to all ~~TCP~~ commands
- ❏ Device accepts configuration modifications
- ❏ Device retains configuration modifications after a soft reboot
- ❏ Device retains configuration modifications after a hard reboot
- ❏ Device reports consistently at the defined interval
- ❏ Device backfills all buffered data when a broken connection is resumed
- ❏ Device isn't prone to random reboots or crashing firmware
- ❏ Device is able to report a GPS location when installed on a trailer
- ❏ Device is able to report a GPS location when installed underneath a vehicle
- ❏ Device is able to maintain a cellular connection consistently during a shipment
- ❏ Device is able to maintain a consistent cellular connection when indoors
- ❏ Device is able to detect nearby wifi access points and successfully reports MAC addresses to server
- ❏ Device consistently sends alerts ~~via SMS and/or TCP~~ for each configured event
- ❏ Device wakes to ~~SOS event~~ wake button
- ❏ LED indicator functions properly
- ❏ Device reports at least as accurately and consistently as the Micron Bolt Mini when devices are installed on the same asset

- ❏ Battery level can be accurately estimated
- ❏ Battery life isn't affected by poor connectivity

Parking Lot

Parking lot for previous content that has been replaced. This section is out of scope and for information only.

~~Initial thoughts on cloud integration~~

~~Northbound: Data sending to the cloud~~

~~The first choice would be to use~~

~~a) either AWS Kinesis Firehose~~
~~https://aws.amazon.com/kinesis/data-firehose/~~
~~Amazon Kinesis Data Firehose is the easiest way to reliably load streaming data into data lakes, data stores, and analytics services.~~

~~b) or AWS Kinesis Streams~~
~~https://aws.amazon.com/kinesis/data-streams/~~
~~Amazon Kinesis Data Streams (KDS) is a massively scalable and durable real-time data streaming service.~~

~~Firehose might seem a bit easy for the cloud side, at least at the beginning.~~
~~It should not be a problem for the imp side to switch from one to another later, if needed. But, of course, it will take some minor additional efforts.~~

~~c) Alternatively, other AWS services can be used:~~
~~- For example, AWS SQS~~
~~https://docs.aws.amazon.com/sqs/~~
~~Amazon Simple Queue Service (Amazon SQS) is a fully managed message queuing service that makes it easy to decouple and scale microservices, distributed systems, and serverless applications.~~
~~- Maybe together with AWS SNS~~
~~https://docs.aws.amazon.com/sns/~~
~~Amazon Simple Notification Service (Amazon SNS) is a web service that enables applications, end-users, and devices to instantly send and receive notifications from the cloud.~~

~~This may make sense if SNS and SQS are utilized for the configuration feature (see below).~~

But this option requires more investigation and more efforts for overall cloud-device design and implementation.

A concrete AWS service should be finally chosen based on customer needs - taking into account the planned amount of devices/data, pricing and limitations of the services, etc.

In all options the cloud agent will send data as records/messages. One record/message contains one completed data message from the device in text (JSON) format.

All possible fields (keys) in the JSON message will be strictly specified. Some fields can be mandatory, some - optional, do not exist in every message - depends on the current configuration and/or on the event which triggered the message sending.

An example of data msg (all fields are just for example. The exact format/content/names TBD during design/implementation):

```
{
   "trackerId": "c0010e2a69f088a4",
   "timestamp": 1617900077,
   "status": {
      "movement": true,
      "tracking": true
   },
   "location": {
      "type": "gnss",
      "accuracy": 3,
      "lng": 30.571465,
      "lat": 59.749069
   },
   "sensors": {
      "batteryLevel": 64.32,
      "temperature": 42.191177
   },
   "alerts": {
      "temperatureHigh": true
   }
}
```

Southbound: Configuration Settings

Option 1: AWS services are not required.

Cloud agent exposes HTTPS endpoint(s). URL of every cloud agent is known after the device enrollment.

GET request returns the latest known actual configuration from the device to the cloud.
PUT request pushes a configuration update from the cloud to the device.

All cfg update requests are queued in the cloud agent and sent to the device when the device is online. The device sends the full actual configuration to the cloud agent every time the actual configuration is changed. The cloud should periodically poll the cloud agent (GET requests) to obtain the actual configuration.

Basic HTTPS authentication will be used.
Cfg update requests can be encrypted on the cloud side by a private key and decrypted on the imp side by the public key.

Option 2: Utilize AWS services.

Potentially AWS SQS / AWS SNS and/or some other services can be used for the configuration transmission/returning. This option requires more investigation and more efforts for overall cloud-device design and implementation.

A concrete variant should be finally chosen based on customer needs.

In any option - configuration is in text (JSON) format.

Configuration update requests can contain the full configuration or only the parts/fields of configuration which should be updated. Some fields (keys) can be mandatory, some - optional.

An example of cfg update request (all fields are just for example. The exact format/content/names TBD during design/implementation):

```
{
    "cfgFormatVersion": "1.0",
    "readingPeriod": 600,
    "reportingPeriod": 43200,
    "temperatureHigh": 40,
    "tracking": false
}
```

The actual cfg returned to the cloud is always the full cfg.