



Connecting the IDK to the Google Cloud IoT Core

INTRODUCTION

This application note provides details on connecting the IoT Development Kit (IDK) to the Google Cloud (GCP). An environmental sensing (temperature, humidity, pressure, ambient light) use case provided as an example within the IDK Software will be used to demonstrate connectivity to GCP.

Sensor data are transmitted over secure connection to AWS IoT Core service. Connection is secured by TLS encryption and the measured sensor data are periodically sent in JSON format over MQTT.

REFERENCES

[Ref 1] IoT IDK User Guide: <https://www.onsemi.com/pub/Collateral/AND9666-D.PDF>

[Ref 2] Google Cloud IoT Core documentation: <https://cloud.google.com/iot/docs/>

[Ref 3] Getting started: <https://cloud.google.com/iot/docs/how-tos/getting-started>

[Ref 4] Creating registries and devices: <https://cloud.google.com/iot/docs/how-tos/devices>

[Ref 5] PUB/SUB service documentation: <https://cloud.google.com/pubsub>

PREREQUISITES

The below items are needed in order to successfully configure and run this use case.

Hardware

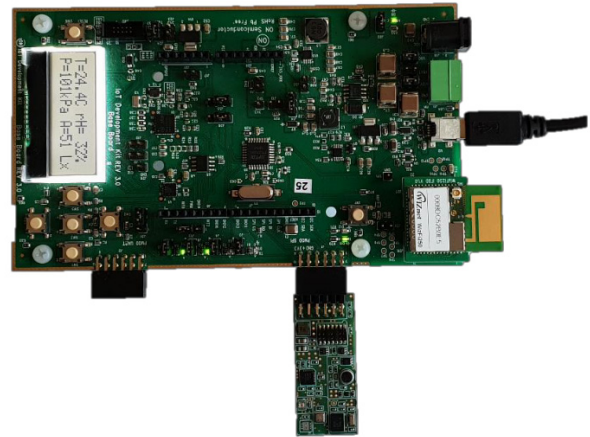
- IDK Baseboard ([BB-GEVK](#)) with WizFi250 Wi-Fi Module
- Multi-Sensor Shield ([MULTI-SENSE-GEVB](#))

The Multi-Sensor shield should be plugged into one of the two available PMOD connectors of the IDK Baseboard. The IDK Baseboard should then be connected to PC to provide power and for downloading the example use case.

ON Semiconductor®

www.onsemi.com

APPLICATION NOTE



Software

- IoT IDK Software (Version 4.3 & above): [BB-GEVK Software](#)
Refer to IoT IDK User Guide (Ref 1) for installation details
- openssl
Optional utility for generation of public/private key pairs for IoT devices on local machine

Google Cloud account

- Google Account with Access to Google Cloud Console: <https://cloud.google.com/>
(Free trial account is sufficient for this example)
- Cloud IoT Core Service Enabled
This service is used to connect and manage IoT devices. Please refer to the reference documents listed above.

GCP IOT CORE CONFIGURATION

This section covers the creation of Device Registry and IoT devices using web based Cloud Console admin UI.

Create Device Registry

Device Registry is a container of devices with shared properties. At least one registry must be created to connect devices to GCP. Follow the instructions in the [Creating registries and devices](#) documentation page to create a Device Registry. The registry created for this example uses the parameters listed below and the filled registry creation page is shown in Figure 1.

- Registry ID: *idk-examples-registry*
- Region: *europa-west1*
- Protocol: *MQTT*
- Default telemetry topic: *projects/{project-id}/topics/events*
- Device state topic: *None*
- CA certificate: *None*
- Stack Driver Logging: *Info*

Google Cloud Platform IDK Project

IoT Core Create a registry

Define how devices in this registry will send data to Cloud IoT Core. After you create your registry, you can start adding devices to it. [Learn more](#)

Registry ID
Permanent identifier for your registry. Start with a letter.
idk-examples-registry

Region
Determines where data is stored for devices in this registry. Choice is permanent.
europa-west1

Protocol
Select the protocols your devices will use to connect to Cloud IoT Core. [Learn more](#)
 MQTT
 HTTP

Cloud Pub/Sub topics
Cloud IoT Core routes device messages to Cloud Pub/Sub for aggregation. You can route messages to different topics and subfolders in Cloud Pub/Sub based on the type of data in the messages. [Learn more](#)

Default telemetry topic
Device telemetry events will be published to this topic by default. Add more topics if you want these events to be published to separate topics.
projects/sublime-state-230114/topics/events

Add more telemetry topics

Device state topic (Optional)
State events published by MQTT devices are stored in the registry by default. You can also select a Cloud Pub/Sub topic where these messages will be published on a best-effort basis. [Learn more](#)
Select a Cloud Pub/Sub topic

Add CA certificate

Stackdriver Logging
Set the default logging for all devices in this registry. You can apply a different setting or debug at the device level. [Learn more](#)
 None
 Error
 Info
 Debug

Create Cancel

Figure 1.

Generate Public/Private Key Pair for Device

Before creating an IoT device under a device registry it is required to generate a public/private key pair for the device that will be used to authenticate the device when it connects to the cloud. The public key will be uploaded to Cloud Console during device creation. Private Key will be stored in the device itself as part of firmware.

The IDK supports the use of RSA–256 private key with 2048–bit key size.

The key pair can be generated using the *openssl* utility program as described in [Creating public/private key pairs](#) documentation of IoT Core service.

The commands listed in Figure 2 can be used to create private key (rsa_private.pem) and then extract the public key (rsa_public.pem).

```
openssl genrsa-out rsa_private.pem 2048
openssl rsa-in rsa_private.pem-pubout -out rsa_public.pem
```

Figure 2. Openssl Commands used to Generate Public/Private Key Pair

If openssl is not installed on your machine as a standalone program it can still be distributed as part of other software packages like Git Tools or Cygwin. If neither of that is applicable to generate keys locally it is possible to use Cloud Shell from the Cloud Console UI to generate the keys directly in Web browser as described in next section.

Using Google Cloud Shell to Generate Public/Private Key Pair

The Google Cloud UI provides Cloud Shell console interface that can be used to access all GCP resources directly from Web Browser. The Cloud Shell environment has openssl utility preinstalled and can be used to generate and download public/private key pairs:

1. Activate Cloud Shell from the toolbar of Cloud UI web page as shown in Figure 3 step 1
2. Enter the openssl commands from Figure 2 into the console to generate public/private key pair
3. Use Download file button to download generated keys from shell to your local machine. When asked for file name enter *rsa_private.pem* and then repeat for *rsa_public.pem* file
4. After downloading the files clean up the shell environment by deleting the generated keys using *rm* command followed by the names of files you want to remove

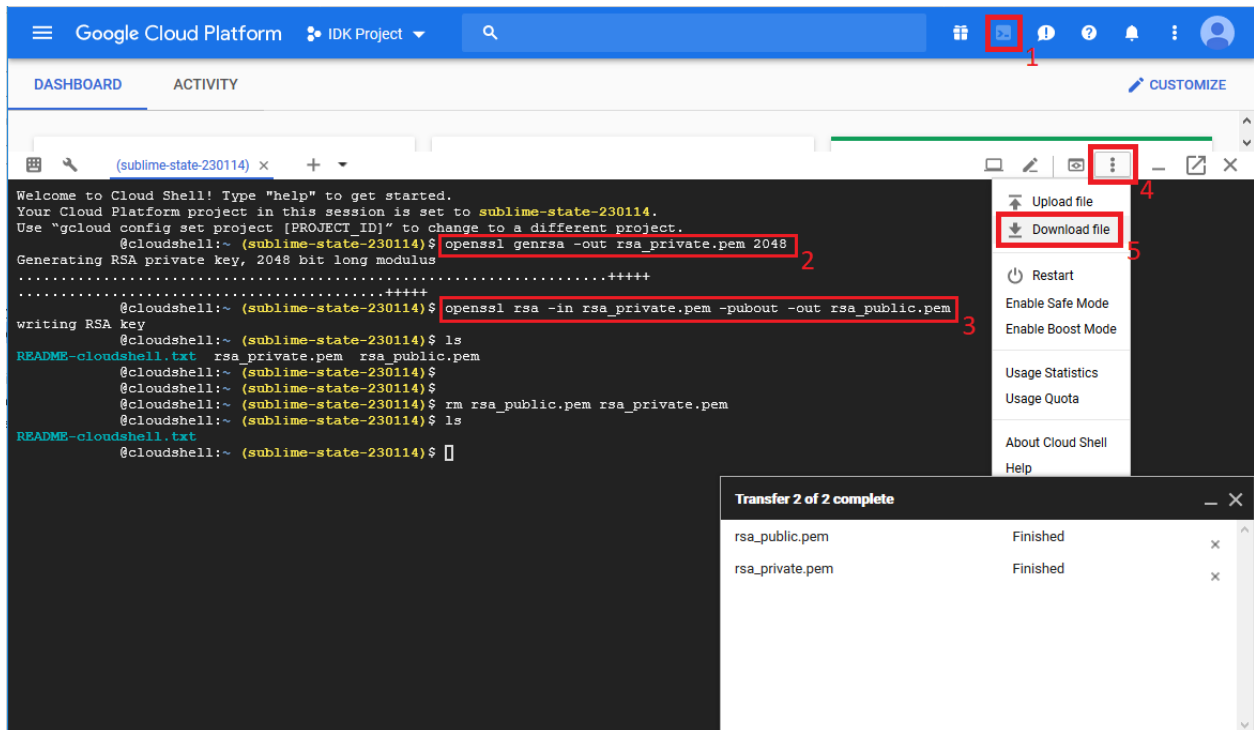


Figure 3. Steps for Generation of Public/Private Key Pair using Google Cloud Shell

Create Device

To add new device to your device registry follow the instructions in [Creating or editing a device](#) section of IoT Core service documentation. This example will use the following device parameters:

- Device ID: *idk-device*
- Device communication: *Allow*
- Authentication Input Method: *Upload*
- Public Key Format: *RS256* – Depends on type of generated public/private key pair. IDK supports only RS256
- Public key value: *rsa_public.pem* – Use Browse button to select pem file containing generated public key
- Public key expiration date: *None*
- Device metadata: *None*
- Stackdriver Logging: *Use registry default settings*

Google Cloud Platform | IDK Project

IoT Core | Create a device

Create a device in registry idk-examples-registry.

Device ID [?]

Device communication [?]
 Allow
 Block

Authentication (Optional) [?]
Input method
 Enter manually
 Upload

Public key format
 RS256 [?]
 ES256 [?]
 RS256_X509 [?]
 ES256_X509 [?]

Public key value

Public key expiration date (Optional)
 Expires on:

Device metadata (Optional) [?]
 Key must contain only letters, numbers, hyphens, and underscores, and be no longer than 128 characters

Stackdriver Logging
 Choose a log setting for this device. Will override the registry default for this device only.
[Learn more](#)
 Use registry default setting
 None [?]
 Error [?]
 Info [?]
 Debug [?]

Figure 4. Example of Device Creation Page

CONFIGURING IDK EXAMPLE PROJECT

With the device created in cloud it is now time to prepare the IDK to connect to cloud using these credentials.

Importing Project

The example code for this application is bundled with the IoT IDK software package and can be imported using the Examples toolbar menu. The project is listed in the menu as *Complex* → *Secure Cloud Connectivity (TLS) by Ubiquios* → *Multi Sensor Node* → *Google Cloud* as shown in Figure 5.

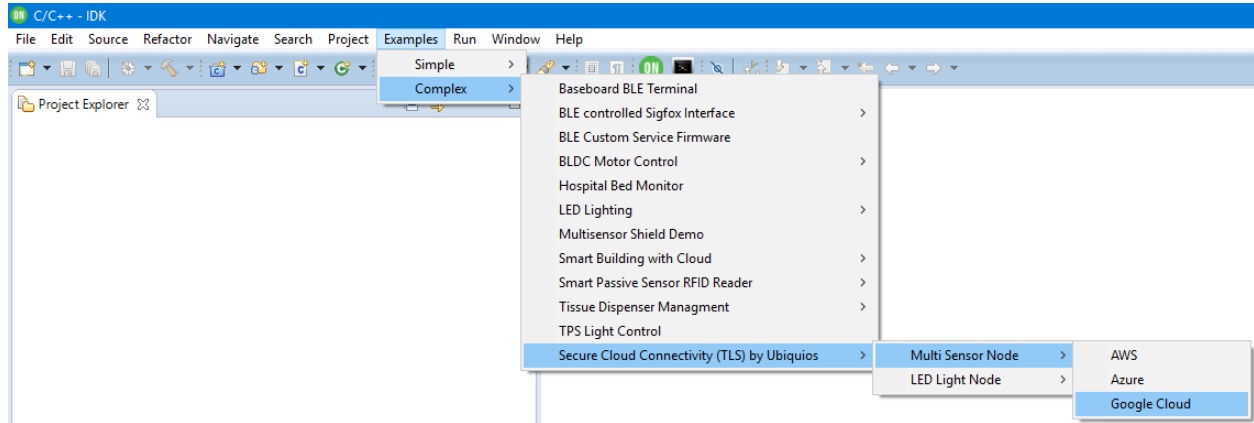


Figure 5. Examples Import Menu in IoT IDK

Project structure

After importing the project will appear in the project explorer view.

The internal structure of the project is as follows:

- src – Folder containing all source code files
 - ◆ configuration.h – Contains macro definitions for Wi-Fi credentials and cloud related variables that need to be modified for each user / environment
 - ◆ gcp_device_keystore.def – Contains preprocessed device certificates that are used to authenticate the device when it connects to cloud. This file needs to be generated and replaced for each device (steps in next section)
 - ◆ gcp_iot.c – Main application file that handler Wi-Fi connectivity, MQTT session to GCP and timers for periodic publishing of sensor data to cloud
 - ◆ jwt.* – Function for generation of signed JWT tokens used for authentication
 - ◆ sensor_node.* – Abstraction layer for the IDK sensor libraries used in this example
 - ◆ platform.* – Part of Ubiquios TLS stack. Should not be modified
 - ◆ pmalloc_pools.* – Part of Ubiquios TLS stack. Should not be modified
 - ◆ hal – Folder containing abstraction layers for the Ubiquios TLS stack. Should not be modified

Entering Credentials in Configuration.h File

The example program contains file **configuration.h** that is located inside of the **src** folder. This file is divided into three sections used to configure Wi-Fi, GCP connectivity and application related variables which need to be adjusted for proper operation of the application

1. Wi-Fi configuration consists of entering valid Wi-Fi access point credentials consisting of SSID and passphrase. To do this modify the **AP_SSID** and **AP_PASSPHRASE** macros

```
#define AP_SSID "my_ap_ssid"

#define AP_PASSPHRASE "my_ap_passphrase"
```

2. To configure GCP connectivity settings modify *GCP_PROJECT_ID*, *GCP_REGISTRY_REGION*, *GCP_REGISTRY_ID*, *GCP_DEVICE_ID* and *GCP_TELEMETRY_TOPIC* with values used to create Device Registry and device in the GCP IoT Core configuration section

```
#define GCP_PROJECT_ID          "sublime-state-230114"

#define GCP_REGISTRY_REGION    "europe-west1"

#define GCP_REGISTRY_ID       "idk-examples-registry"

#define GCP_DEVICE_ID         "idk-device"

#define GCP_TELEMETRY_TOPIC    "/devices/" GCP_DEVICE_ID "/events"
```

3. Optionally modify application specific configuration macros to change sensor data publish interval or to use simulated data if MULTI-SENSE-GEVK shield is not used

```
/** Time to wait after a message is sent before sending the next message. */
#define SEND_INTERVAL          (UB_SECONDS(12))

/** Time to wait before retrying when send fails. */
#define SEND_RETRY_INTERVAL    (UB_SECONDS(5))

/** Change to true if simulated data are to be used.
 *
 * When set to true it is not required to connect the MULTI -SENSE-GEVK board to IDK
 * Baseboard.
 */
#define USE_SIMULATED_SENSOR_DATA (false)
```

Generating Device Keystore from Private Key

The last configuration step is to generate a keystore that contains device's private key used to authenticate the device when connecting to GCP. To overcome the low RAM footprint of the processor located on IDK Baseboard the downloaded certificates cannot be used directly but have to be preprocessed and included as a source code file into the project.

The following steps need to be done to get preprocessed keystore header file that can be used in the project:

1. Open IDK installation directory and navigate to `C:\IDK_INSTALL_DIR\external\ubiquios\bin` directory.
2. Copy the downloaded private key file `rsa_private.pem` into this directory
3. Open Command line terminal (`cmd.exe`) and change working directory to this folder.

This can be done from File Explorer by typing `cmd.exe` into the navigation bar and hitting enter as shown in Figure 6

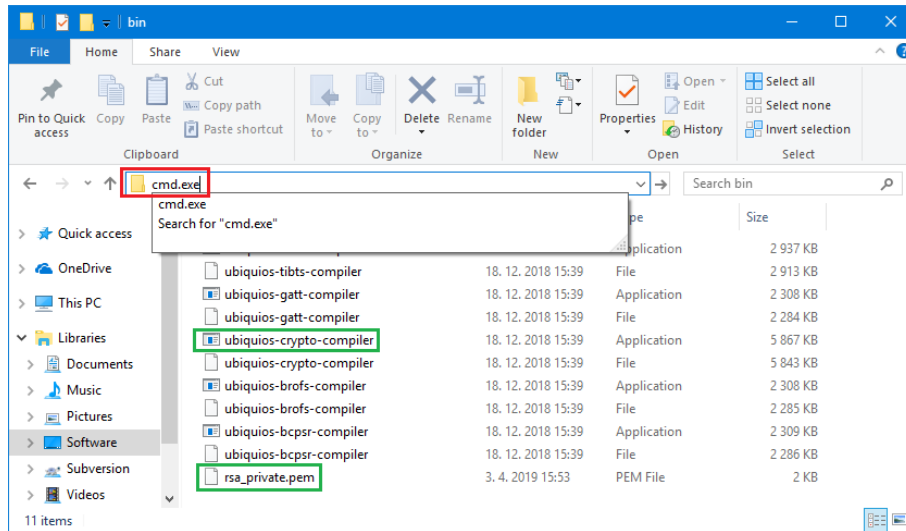


Figure 6. Entering `cmd.exe` into the Navigation Bar in File Explorer to Open Command Line with Working Directory Set to this Folder

4. In command line enter the following command to generate keystore for this application:

```
ubiquios-crypto-compiler.exe keystore -n gcp_device_keystore -o gcp_device_keystore.def
rsa_private.pem
```

This command will extract the private key encoded in `rsa_private.pem` and turn it into C header file named `gcp_device_keystore.def` that contains keystore structure named `gcp_device_keystore`. This structure is then used by application when operations involving private key are executed.

5. Copy the generated `gcp_device_keystore.def` file into `src` subdirectory of the project, replacing the original placeholder file
6. Compile project to verify that no errors are generated from the new keystore header

VERIFYING DEMO FUNCTIONALITY

Once all configuration macros in configuration.h header file were modified to valid values and keystore with private key generated the application can be compiled and flashed into the IDK Baseboard.

The application will print status messages to serial terminal showing sensor data and connectivity status. Serial terminal output can be accessed either from the Flash Utility if it is left open after flash process is finished or from any other serial terminal application. The serial terminal uses 115200 baud rate, 8 data bits, no parity, 1 stop bit and no flow control.

Working application will show the following status messages:

- ◆ Connection to Wi-Fi network was successful and assigned IP address of the kit is shown
- ◆ NTP request for current network time succeeds and current time in UTC is shown
- ◆ Signed JWT authentication token is generated and MQTT connection is established
- ◆ Sensor data are measured and successfully published to cloud periodically according to configured *SEND_INTERVAL*
- ◆ Payload of all packets received on one of the */devices/{device_name}/commands/* sub topic is printed to terminal

```

opening port : \\.\COM6
open_serial_port : opening of serial port successful
This version of ubiquios is for evaluation purposes only and must not be distributed.
SENS: Sensor node initialization success
GCP IoT Core - Device Example
Welcome to the Internet of Things - powered by ubiquios!
ubiquios build version: ubiquios-7.7.5-641 20181214052914

WLAN: Attempting to connect to network.
WLAN: SSID = test_ap
WLAN: passphrase = *****
WLAN: Connecting...
TCP/IP: Link came up with IP address 192.168.43.53
NTP: Requesting time from 'time.google.com'
NTP: Request successful.
NTP: Time is 2019-4-4 12:38:9
MQTT: Creating session to 'mqtt.googleapis.com'
JWT: Generating token with claims: '{"iat":1554381489,"exp":1554467889,"aud":"sublime-state-230114"}'
JWT: Token generated in 16787 ms.
MQTT: Session created
MQTT: Subscription to '/devices/idk-device/commands/#' added.
MQTT: Connected to MQTT server
SENS: Generated data payload:
{"timestamp":30842,"temp":27.09,"hum":23.73,"pres":98632,"als":279.00}
MQTT: Publishing to topic '/devices/idk-device/events'.
MQTT: Subscribed to topic /devices/idk-device/commands/#
SENS: Generated data payload:
{"timestamp":43047,"temp":27.17,"hum":23.67,"pres":98633,"als":280.00}
MQTT: Publishing to topic '/devices/idk-device/events'.
MQTT: Message received ( 53368): topic="/devices/idk-device/commands", message="Hello world!", qos=0, retained=false
SENS: Generated data payload:
{"timestamp":55327,"temp":27.24,"hum":23.52,"pres":98634,"als":282.00}
MQTT: Publishing to topic '/devices/idk-device/events'.
SENS: Generated data payload:
{"timestamp":67488,"temp":27.32,"hum":23.38,"pres":98634,"als":276.00}
MQTT: Publishing to topic '/devices/idk-device/events'.

```

Figure 7. Terminal Output from Baseboard showing Periodically Published Data and Received Messages

Sending Commands to Device from Cloud UI

The device is subscribed to the `/devices/{device-id}/commands/#` wildcard topic which allows it to receive messages from all its sub topics, e.g. `/devices/idk-device/commands/my-command`

The example firmware set up to print payload of all such messages to serial terminal as can be seen in Figure 7 where message with the payload “Hello world!” was received. Additional functionality can be added in the receive handler in the application.

Commands can be send from device page of Cloud UI as described in the [Sending a command](#) section of Cloud IoT Core documentation and illustrated in Figure 8.

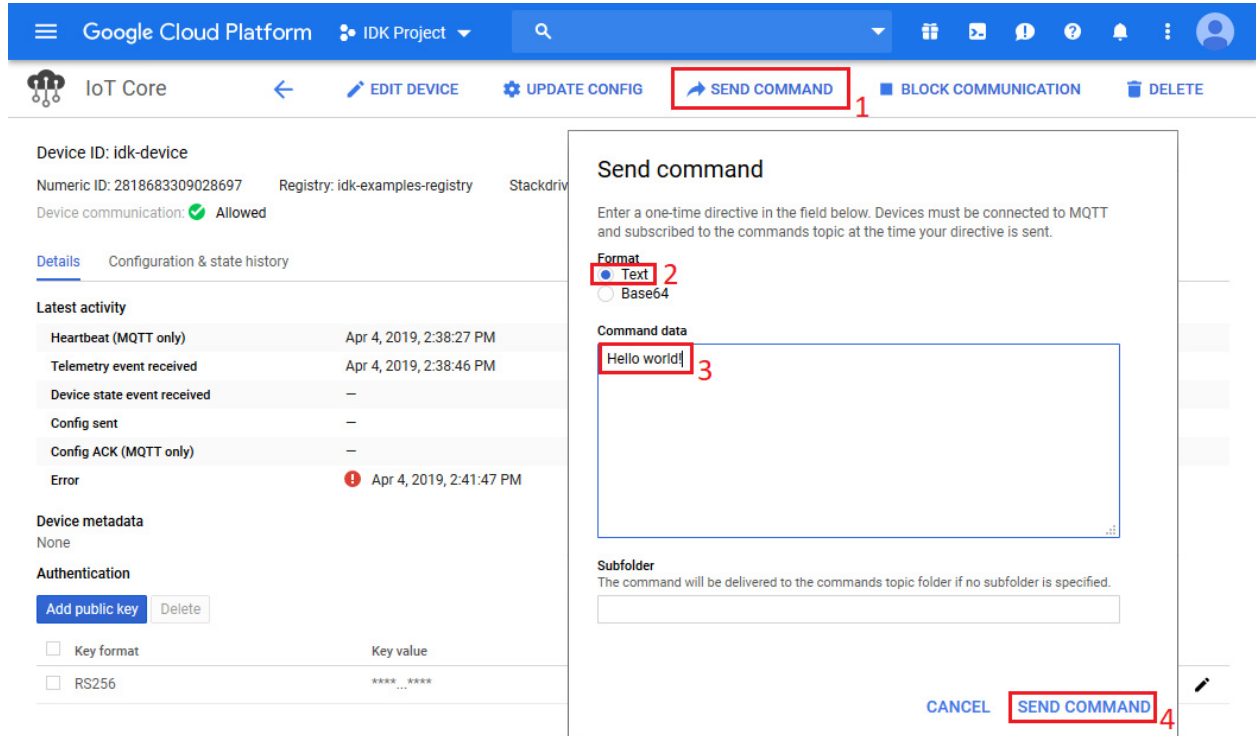


Figure 8. Sending a Command to Device from Cloud UI

Reading Telemetry Data using PUB/SUB Service

Google Cloud UI does not offer direct method for viewing of payloads of published device messages. To view telemetry data it is necessary to create a subscription to devices default telemetry topic using the [PUB/SUB](#) service and then pull the messages using the gcloud command–line tool.

Following the instructions from [PUB/SUB Quick start guide](#) create a new subscription to the telemetry topic which was created when first device was added to device registry. Once the subscription is created all new messages that are published to the telemetry topic can be pulled using the gcloud command line tool that is available in the Cloud Shell.

In this case the topic *projects/sublime–state–230144/topics/events* was selected as default telemetry topic for IoT devices. A subscription named *projects/sublime–state–230144/subscriptions/sensors* was created to receive messages published to events topic. Using the Cloud Shell environment it is possible to pull messages with the gcloud command line tool as illustrated in Figure 9.

```
gcloud pubsub subscriptions pull --auto-ack sensors
```

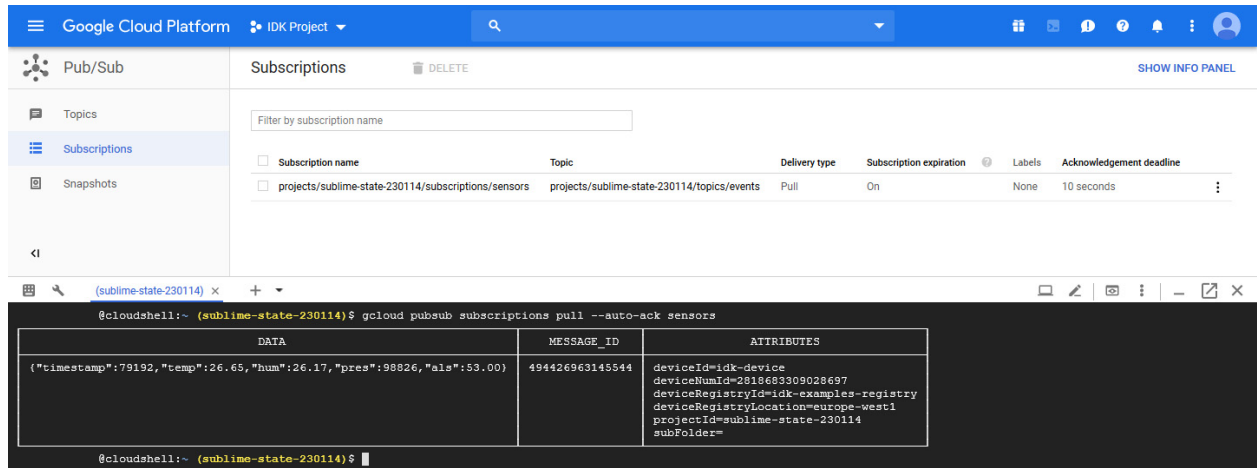


Figure 9. Using Cloud Command Line Tool in Cloud Shell to Display Payloads of Messages Published by IoT Devices

ON Semiconductor and are trademarks of Semiconductor Components Industries, LLC dba ON Semiconductor or its subsidiaries in the United States and/or other countries. ON Semiconductor owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of ON Semiconductor’s product/patent coverage may be accessed at www.onsemi.com/site/pdf/Patent-Marking.pdf. ON Semiconductor reserves the right to make changes without further notice to any products herein. ON Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does ON Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. Buyer is responsible for its products and applications using ON Semiconductor products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by ON Semiconductor. "Typical" parameters which may be provided in ON Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer’s technical experts. ON Semiconductor does not convey any license under its patent rights nor the rights of others. ON Semiconductor products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use ON Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold ON Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that ON Semiconductor was negligent regarding the design or manufacture of the part. ON Semiconductor is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303-675-2175 or 800-344-3860 Toll Free USA/Canada
Fax: 303-675-2176 or 800-344-3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800-282-9855 Toll Free
USA/Canada
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910

ON Semiconductor Website: www.onsemi.com
Order Literature: <http://www.onsemi.com/orderlit>

For additional information, please contact your local Sales Representative