



Comentario técnico: CTC-100
 Componente: **Conexión del ESP32 con Mongoose-OS a Google Cloud Platform**
 Autor: Sergio R. Caprile, Senior R&D Engineer

Revisiones	Fecha	Comentarios
0	26/06/20	

En el [CTC-099](#) analizamos la Google Cloud Platform (GCP) y desarrollaremos la utilización de Cloud IoT Core, el servicio de conectividad, utilizando MQTT. Mongoose-OS incorpora un cliente MQTT y una library con soporte para GCP, por lo que en este Comentario Técnico analizaremos la forma de conectarnos a dicha plataforma IoT con un ESP32 y Mongoose-OS.

Configuración

El proceso de configuración para la utilización de un dispositivo en esta plataforma se desarrolla fundamentalmente en dos partes

1. Primero debemos configurar el proyecto y grabar el dispositivo
2. A continuación debemos generar las credenciales, crear el dispositivo en Cloud IoT Core, y grabar las credenciales (y parámetros de configuración) en el dispositivo

Si estamos desarrollando, existe un tercer paso, opcional, que nos puede ahorrar bastante trabajo. Consiste en obtener las credenciales del dispositivo y guardarlas en el proyecto, de modo de no tener que borrarlo y volverlo a cargar en Cloud IoT Core.

Configuración del proyecto

Configuramos en el archivo YAML que describe el proyecto que vamos a utilizar la library de GCP y nos conectaremos como cliente a una red WiFi.

```
libs:
  - origin: https://github.com/CikaElectronica/gcp2 # GCP library, Cika version
config_schema:
  - ["sys.tz_spec", "<-03>3"] # Timezone spec in tzset format
  # GCP options
  - ["gcp.enable_config", true] # Subscribe to the configuration topic
  - ["gcp.enable_commands", true] # Subscribe to the command topic
```

Como vemos, sólo hace falta indicar que usamos la library de GCP (en este caso una versión extendida desarrollada en Cika)¹ e indicar si vamos a recibir comandos y cambios de configuración, para que el código se suscriba (o no) a los tópicos correspondientes.

La configuración de zona horaria es opcional, para más información remitimos al [CTC-097](#).

Credenciales y Cloud IoT Core

Luego de compilado el código (`mos build`) y grabado el microcontrolador (`mos flash`) mediante `mos tool`, necesitamos realizar una serie de tareas. Afortunadamente todo se resume a una llamada a una aplicación (dos la primera vez). Antes de proseguir es fundamental que hayamos leído el [CTC-099](#) y que hayamos realizado el `quickstart` de Cloud IoT Core. Esto nos dejará con `gcloud` instalado, más un proyecto y una `device registry`

¹ Hemos desarrollado una extensión a la library de GCP de modo de facilitar el uso desde JavaScript (mJS). Al momento de escribir este texto, se halla pendiente el pedido para incorporar estas extensiones en la library del proveedor; por lo tanto utilizaremos nuestra versión directamente desde github. <https://github.com/CikaElectronica/gcp2/>

CTC-100, Conexión del ESP32 con Mongoose-OS a Google Cloud Platform

donde cargar nuestro dispositivo.² También recomendamos no seguir el tutorial de Mongoose-OS sobre este tema.

Las acciones a realizar son:

- cargar el dispositivo en la *device registry*
- generar las clave privada y pública
- subir la clave pública a la *device registry*
- obtener el certificado del broker de Google
- configurar el dispositivo con los datos de conexión de Cloud IoT Core correspondientes
- cargar clave privada y certificado de Google en el dispositivo

Afortunadamente, *mos tool* realiza todo esto por nosotros. Para ello, la primera vez que lo vayamos a hacer debemos autorizar la API que la herramienta utiliza, a través de *gcloud*. Si no lo hacemos, observaremos un mensaje como el siguiente:

```
Error: google: could not find default credentials. See
https://developers.google.com/accounts/docs/application-default-credentials for more
information.
/src/go/src/github.com/mongoose-os/mos/cli/gcp/gcp.go:55: failed to create GCP HTTP
client
```

y si se nos ocurre hacer lo que allí dice, perderemos valioso tiempo de manera innecesaria.

Para generar las credenciales de autorización ejecutamos el siguiente comando:

```
$ gcloud auth application-default login
```

se abrirá nuestro navegador para consultarnos si realmente queremos autorizar a esa aplicación, deberemos ingresar con las credenciales de Google Cloud Platform y confirmarlo, y luego observaremos en el intérprete de comandos un mensaje similar al siguiente:

```
Credentials saved to file: [.../.config/gcloud/application_default_credentials.json]
These credentials will be used by any library that requests Application Default
Credentials (ADC).
```

luego, además, recibiremos un mail de Google avisándonos que

```
Google Auth Library was granted access to your Google Account
```

De ahora en más, lo único que necesitamos hacer para cargar un dispositivo en GCP es ejecutar *mos tool* de la siguiente forma, con el dispositivo conectado:

```
mos gcp-iot-setup --gcp-project PROJECT --gcp-region REGION --gcp-registry REGISTRY
```

donde PROJECT, REGION y REGISTRY corresponden a datos que ingresamos al hacer el *quickstart* de Cloud IoT Core.

El resultado será algo como lo que sigue a continuación:

```
$ mos gcp-iot-setup --gcp-project test-gcp-280019 --gcp-region us-centrall --gcp-registry my-
registry
Using port /dev/ttyUSB0
Connecting to the device...
  esp32 30AEA4807A98 running gcp_
Generating ECDSA private key

Certificate info:
  Subject : CN=esp32_807A98
  Issuer  : CN=esp32_807A98
  Serial  : 0
  Validity: 2020/06/25 - 2030/06/25
```

² El *quickstart* recomienda borrar lo generado, aquí recomendamos no hacerlo de modo de poder utilizarlo para estas pruebas. Lo borraremos al finalizar...

CTC-100, Conexión del ESP32 con Mongoose-OS a Google Cloud Platform

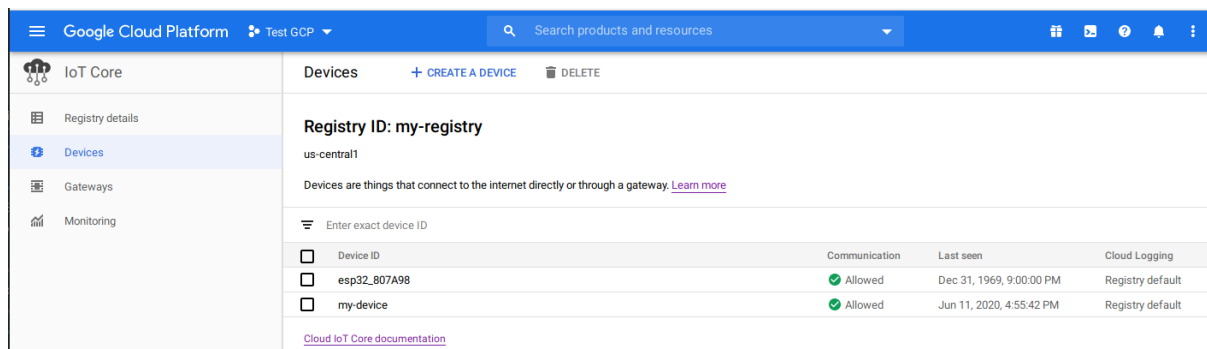
```
Key algo: ECDSA
Sig algo: ECDSA-SHA256
Writing public key to gcp-esp32_807A98.pub.pem...
Writing key to gcp-esp32_807A98.key.pem...
Uploading gcp-esp32_807A98.key.pem (227 bytes)...
Creating the device...
Uploading CA certificate...

Updating config:
gcp.ca_cert = gcp-lts-ca.pem
gcp.device = esp32_807A98
gcp.enable = true
gcp.key = gcp-esp32_807A98.key.pem
gcp.project = test-gcp-280019
gcp.region = us-central1
gcp.registry = my-registry
gcp.server = mqtt.2030.ltsapis.goog:8883
sntp.enable = true
Setting new configuration...
$
```

Podemos entonces observar mediante *gcloud* nuestros dispositivos, y si éste ya se conectó, lo veremos también:

```
$ gcloud iot devices list --region us-central1 --registry my-registry
ID          NUM_ID          BLOCKED
esp32_807A98 2748413172843753
my-device    2792707136353885
$
$ gcloud iot devices configs list --region us-central1 --registry my-registry --device esp32_807A98
VERSION  CLOUD_UPDATE_TIME          DEVICE_ACK_TIME
1        2020-06-25T14:35:38.586759Z 2020-06-25T14:35:53.452257Z
$
```

También podemos observar la consola web. La captura siguiente corresponde al momento de registrar el dispositivo³:



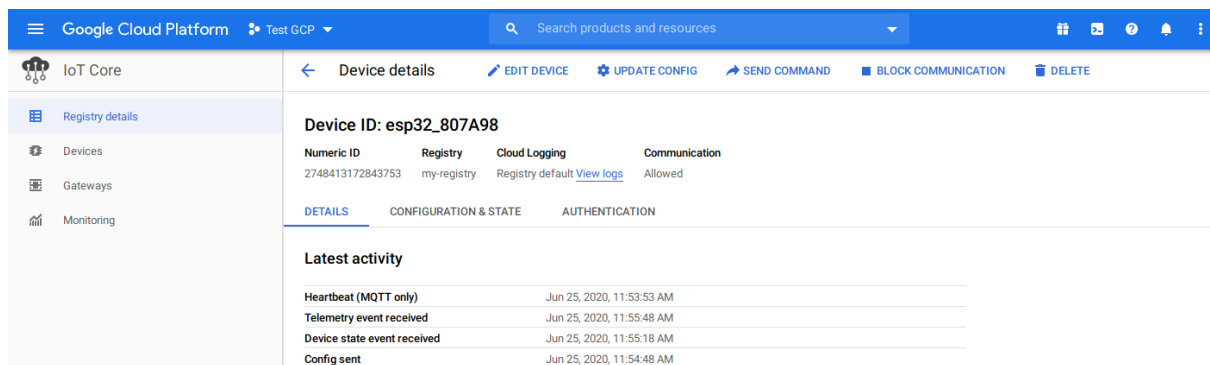
The screenshot shows the Google Cloud Platform IoT Core console. The left sidebar has a menu with 'IoT Core' selected, and sub-items for 'Registry details', 'Devices', 'Gateways', and 'Monitoring'. The main content area is titled 'Devices' and shows 'Registry ID: my-registry' and 'us-central1'. Below this, there is a search bar and a table of devices. The table has columns for 'Device ID', 'Communication', 'Last seen', and 'Cloud Logging'. Two devices are listed: 'esp32_807A98' and 'my-device', both with 'Allowed' communication status and 'Registry default' logging.

Device ID	Communication	Last seen	Cloud Logging
<input type="checkbox"/> esp32_807A98	Allowed	Dec 31, 1969, 9:00:00 PM	Registry default
<input type="checkbox"/> my-device	Allowed	Jun 11, 2020, 4:55:42 PM	Registry default

La siguiente corresponde a un tiempo luego que se conectó, recibió su configuración y manualmente enviamos un reporte de estado:

³ El viaje en el tiempo parece ser la traducción de Google del valor 0 (01/01/1970 00:00) a GMT-3... lo cual no debería suceder...

CTC-100, Conexión del ESP32 con Mongoose-OS a Google Cloud Platform



Durante el desarrollo

Mientras estamos desarrollando código, es común que debamos grabar varias veces el ESP32. Cada vez que lo hagamos perderemos la clave, el certificado, y esta parte de la configuración. Podemos simplemente repetir la operación y *mos tool* generará nuevas claves y actualizará el dispositivo por nosotros. Sin embargo, para evitar sufrir una serie de peripecias⁴, recomendamos hacer lo que sigue a continuación:

- Obtener clave y certificado del dispositivo, copiándolos en el proyecto. Los nombres de archivo los observamos en el listado generado al crear el dispositivo⁵

```
$ mos get gcp-esp32_807A98.key.pem > fs/ gcp-esp32_807A98.key.pem
$ mos get gcp-lts-ca.pem > fs/gcp-lts-ca.pem
```

- Copiar la parte de la configuración que agregó *mos tool* en nuestro archivo YAML, *mos.yml*; modificando el listado generado al crear el dispositivo:

```
- ["gcp.ca_cert", "gcp-lts-ca.pem"]
- ["gcp.device", "esp32_807A98"]
- ["gcp.enable", true]
- ["gcp.key", "gcp-esp32_807A98.key.pem"]
- ["gcp.project", "test-gcp-280019"]
- ["gcp.region", "us-centrall1"]
- ["gcp.registry", "my-registry"]
- ["gcp.server", "mqtt.2030.ltsapis.goog:8883"]
- ["sntp.enable", true]
```

A partir de ahora, cada vez que regrabemos el ESP32, lo haremos con la misma configuración y credenciales que *mos tool* grabó al crear el dispositivo.

Operación

Recordemos que además de lo anterior, debemos configurar las credenciales para conectarnos por WiFi a nuestra red (SSID y clave).

Iniciado el dispositivo, observaremos en el log si todo funciona como debe, o los errores que se hayan producido.

⁴ la herramienta intenta borrar el dispositivo, cosa que por algún motivo Google no permite (probablemente falte alguna autorización). El dispositivo queda reinicializado con la configuración *version 1* activa. Si en algún momento hemos cambiado alguna vez su configuración, existirán otras configuraciones menos recientes pero con numeración mayor (*version 2*, *version 3*, etc.). Esto a GCP no le agrada en lo más mínimo y no nos dejará cambiar la configuración cuando deseemos hacerlo. Debemos entonces borrar manualmente el dispositivo y generarlo otra vez. Pero, algunas veces, por algún motivo, el nuevo dispositivo creado con el mismo device id, aún luego de haber sido borrado y transcurrido un tiempo prudencial, mantiene las viejas configuraciones y debemos repetir la operación hasta tener éxito.

⁵ pero podemos obtenerlos listando los archivos en el ESP32 con `mos call FS.List`

CTC-100, Conexión del ESP32 con Mongoose-OS a Google Cloud Platform

En los archivos adjuntos disponemos de un ejemplo en detalle. El mismo simula información de telemetría enviando periódicamente fecha y hora; podemos enviar mensajes de estado presionando el botón del kit, y observar en el log la recepción de comandos y cambios de configuración:

```
[Jun 25 11:54:48.097] init.js:53           Connected to GCP
[Jun 25 11:54:48.225] init.js:20           Send event:OK msg:{"t":"2020-06-25 11:54:49"}
[Jun 25 11:55:18.225] init.js:20           Send event:OK msg:{"t":"2020-06-25 11:55:19"}
[Jun 25 11:55:18.656] init.js:28           Send state:OK msg:{"total":287792,"free":206180}
[Jun 25 11:55:23.226] init.js:20           Send event:OK msg:{"t":"2020-06-25 11:55:24"}
[... ]
[Jun 25 12:05:13.271] init.js:20           Send event:OK msg:{"t":"2020-06-25 12:05:14"}
[Jun 25 12:05:16.983] init.js:46           Received command: {"cmd": "no haga nada"}
[Jun 25 12:05:18.273] init.js:20           Send event:OK msg:{"t":"2020-06-25 12:05:19"}
[... ]
[Jun 25 12:24:33.219] init.js:20           Send event:OK msg:{"t":"2020-06-25 12:24:34"}
[Jun 25 12:24:36.630] init.js:38           Received config: {"out1": 1} output: ON
[Jun 25 12:24:38.219] init.js:20           Send event:OK msg:{"t":"2020-06-25 12:24:39"}
```

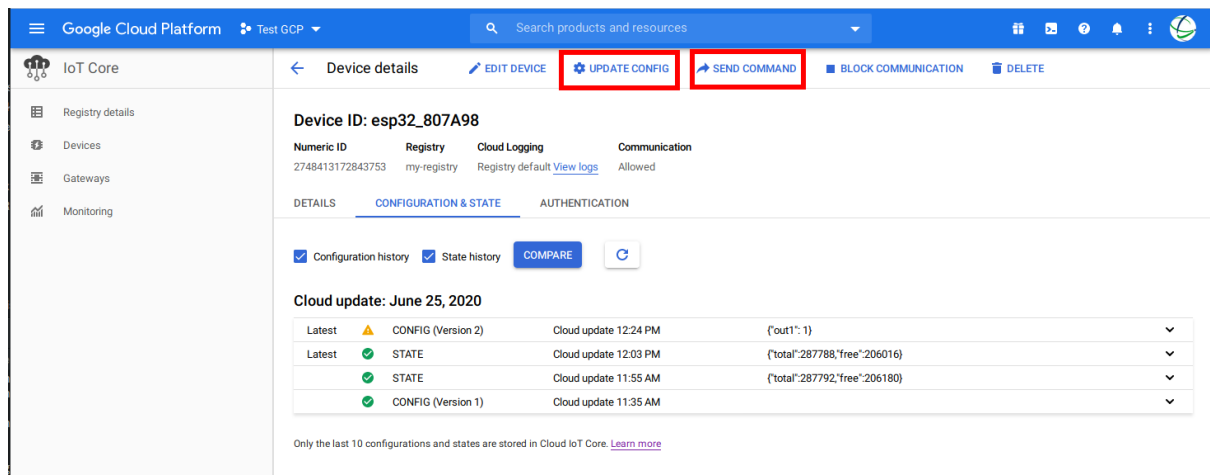
Los datos de telemetría los obtenemos de Cloud Pub/Sub mediante *gcloud*:⁶

```
$ gcloud pubsub subscriptions pull --auto-ack projects/test-gcp-280019/subscriptions/my-subscription
```

DATA	MESSAGE_ID	ATTRIBUTES	DELIVERY_ATTEMPT
{"t":"2020-06-25 11:55:24"}	1300063848206028	deviceId=esp32_807A98 deviceNumId=2748413172843753 deviceRegistryId=my-registry deviceRegistryLocation=us-central1 projectId=test-gcp-280019 subFolder=	

```
$
```

Podemos observar el estado⁷, enviar comandos y actualizar la configuración desde la consola web:



The screenshot shows the Google Cloud Platform IoT Core console. The main content area displays the details for a device with ID `esp32_807A98`. At the top, there are buttons for `EDIT DEVICE`, `UPDATE CONFIG` (highlighted in red), `SEND COMMAND` (highlighted in red), `BLOCK COMMUNICATION`, and `DELETE`. Below these, the device's configuration and state are shown. The `CONFIGURATION & STATE` tab is selected, displaying a table of cloud updates:

Version	State	Time	Output
CONFIG (Version 2)	⚠️	Cloud update 12:24 PM	{"out1": 1}
STATE	✅	Cloud update 12:03 PM	{"total":287788,"free":206016}
STATE	✅	Cloud update 11:55 AM	{"total":287792,"free":206180}
CONFIG (Version 1)	✅	Cloud update 11:35 AM	

o utilizando *gcloud*:

```
$ gcloud iot devices states list --region us-central1 --registry my-registry --device  
esp32_807A98  
UPDATE_TIME  
2020-06-25T15:03:49.242319Z  
2020-06-25T14:55:18.956799Z
```

⁶ Nota: no veremos sólo nuestros datos sino todos los de los dispositivos de esa *device registry* que estén publicando, o de otras que hayamos configurado para publicar en el mismo tópico de Cloud Pub/Sub

⁷ Recordemos que es posible configurar Cloud IoT Core para que publique los mensajes de estado en Cloud Pub/Sub, lo cual no hemos hecho.

CTC-100, Conexión del ESP32 con Mongoose-OS a Google Cloud Platform

```
$
$ gcloud iot devices commands send --command-data='{ "cmd": "no hagas nada"}' --region us-
centrall --registry my-registry --device esp32_807A98
{}
$
$ gcloud iot devices configs update --config-data='{ "out1": 1}' --region us-centrall --registry
my-registry --device esp32_807A98
```

Más información en la documentación de *gcloud*.⁸

Por supuesto, nuestra aplicación IoT final hará todo mediante las APIs correspondientes.

Desarrollo de nuestra aplicación

Describimos brevemente las APIs disponibles. Para más información podemos consultar el código fuente de la library en el repositorio github.⁹

Desarrollo en mJS

Incluimos la API declarando `load('api_gcp.js')`

Sabremos si estamos conectados llamando a `GCP.isConnected()`

Podemos publicar mensajes de telemetría (eventos) utilizando `GCP.sendEvent(msg)`

Podemos publicar mensajes de estado utilizando `GCP.sendState(msg)`

Para recibir la información de configuración, registraremos un handler llamando a `GCP.config(handler, userdata)`

El handler recibirá el mensaje y lo que hayamos puesto en `userdata` como parámetros

Para recibir mensajes de comandos, registraremos un handler llamando a `GCP.command(handler, userdata)`

El handler recibirá el mensaje, el subfolder utilizado (si se utiliza) y lo que hayamos puesto en `userdata` como parámetros. Por ejemplo:

```
GCP.command(function (cmddata, subfolder, ud) {
}, null);
```

Para recibir información de eventos de conexión y desconexión a la red, podemos también registrar los handlers apropiados:

```
Event.addHandler(Event.GCP_CONNECT, function (ev, evdata, ud) {
}, null);
Event.addHandler(Event.GCP_CLOSE, function (ev, evdata, ud) {
}, null);
```

Desarrollo en C

Incluimos la API incluyendo `mgos_gcp.h`

Sabremos si estamos conectados llamando a `bool mgos_gcp_is_connected()`

Podemos publicar mensajes de telemetría utilizando `bool mgos_gcp_send_eventp(struct mg_str *)`

Podemos publicar mensajes de estado utilizando `bool mgos_gcp_send_statep(struct mg_str *)`

La estructura `mg_str` corresponde a un tipo interno de Mongoose-OS para manejar strings.¹⁰

Para recibir la información de configuración, registraremos un handler llamando a `bool mgos_gcp_conf(conf_handler_t, void *ud)`

El handler recibirá el mensaje y lo que hayamos puesto en `ud` como parámetros:

```
typedef void (*conf_handler_t)(const struct mg_str *data, void *ud);
```

Para recibir mensajes de comandos, registraremos un handler llamando a `bool mgos_gcp_cmd(cmd_handler_t, void *ud)`

⁸ <https://cloud.google.com/sdk/gcloud/reference/iot/devices/states/list>
<https://cloud.google.com/sdk/gcloud/reference/iot/devices/commands/send>
<https://cloud.google.com/sdk/gcloud/reference/iot/devices/configs/update>

⁹ <https://github.com/CikaElectronica/gcp2/>

¹⁰ https://mongoose-os.com/docs/mongoose-os/api/core/mg_str.h.md

CTC-100, Conexión del ESP32 con Mongoose-OS a Google Cloud Platform

El handler recibirá el mensaje, el subfolder utilizado (o un string vacío si no se utiliza) y lo que hayamos puesto en `ud` como parámetros:

```
typedef void (*cmd_handler_t)(const struct mg_str *data,  
                             const struct mg_str *subfolder,  
                             void *ud);
```

Para recibir eventos de conexión y desconexión, deberemos registrar un handler de la manera habitual y documentada.¹¹ Los eventos son: `MGOS_GCP_EV_CONNECT` y `MGOS_GCP_EV_CLOSE`,

¹¹ https://mongoose-os.com/docs/mongoose-os/api/core/mgos_event.h.md