EMQX CLOUD | Google Cloud

Migrating from Google Cloud IoT Core to EMQX Cloud

Introduction

By August 16, 2023, Google Cloud IoT Core will no longer be available. All IoT Core users must migrate their applications to alternative IoT services before this date to ensure uninterrupted service for their business.

As an official Google Cloud Certified Partner, EMQ provides two products to aid in the migration process: EMQX Cloud, a fully managed MQTT cloud service; and EMQX Enterprise, an enterprise MQTT platform at scale for private deployment. These solutions boast robust support for extensive connections of IoT devices, as well as reliable message transfer, processing, and storage with exceptional scalability and stability, making them optimal choices for users migrating from Google Cloud IoT Core.

This handbook is designed to offer clear, easy-to-understand, and actionable guidance to existing Google Cloud IoT Core users, providing the migration plan based on the aforementioned EMQX Cloud. Users can utilize this handbook to evaluate their business, select the most suitable solution, and follow the comprehensive steps provided to successfully migrate their services.

Following the completion of migration, users will continue to enjoy Google Cloud's other services while also leveraging the benefits of EMQ's products for their IoT business.

Table of Contents

- Introduction
- Investigation and Evaluation
- Migration Planning
  - Service Construction
  - Device Connection
  - Data Bridging and Integration
  - Summary
- Step-by-Step Migration
- Replacing GCP IoT Core with EMQX Cloud Dedicated
  - Step 1: Creating an EMQX Cloud Deployment on GCP
    - 1. Choose a Specification
    - 2. Choose a Region
    - 3. Choose a Payment Method

- Step 2: Switching Devices Previously Connected to GCP IoT Core to EMQX Cloud Deployment
  - 1. Connection Address and Port
  - 2. Authenticate Device Connections Using JSON Web Token (JWT)
    - 2.1 Parse JWTs in GCP IoT Core
    - 2.2 Migration to EMQX Cloud with Individual Key for Each Client
    - 2.3 Migration to EMQX Cloud with Unified key for All Clients
  - 3. Implement secure MQTT TLS/SSL Authentication
- Step 3: Integrating with GCP Pub/Sub and Third–Party Data Services
  - 1. Bridging Data to GCP Pub/Sub
    - 1.1 Introduction to GCP Pub/Sub
    - 1.2 Creating a GCP Pub/Sub Deployment
    - 1.3 Configuring Data Bridging
    - 1.4 Validating Data Bridging
  - 2. VPC Peering Connection
  - 3. Integrating Data Through EMQX Cloud
    - 3.1 Installing Redis
    - 3.2 Save data to Redis
- Summary

Investigation and Evaluation

Prior to commencing the migration process, it is essential to conduct a comprehensive analysis and evaluation of the existing system.

This will enable users to gain a thorough understanding of the current environment and make an informed decision on the most appropriate migration plan, ultimately ensuring a safe and reliable migration process.

The preliminary investigation is conducted using a questionnaire (see the annex for details of the survey questionnaire), which focuses on three main areas:

Application Overview

Thoroughly understanding the basic situation of applications, such as application scenarios, device types and protocols, concurrency, basic technology stack, application support system, overall application architecture, and other related aspects.

Application Services Relationships

Analyzing the upstream and downstream business correlations, data flow, and data invocation relationships, and investigating the surrounding business and application ecology related to the corresponding business for the application.

Google Cloud Usage

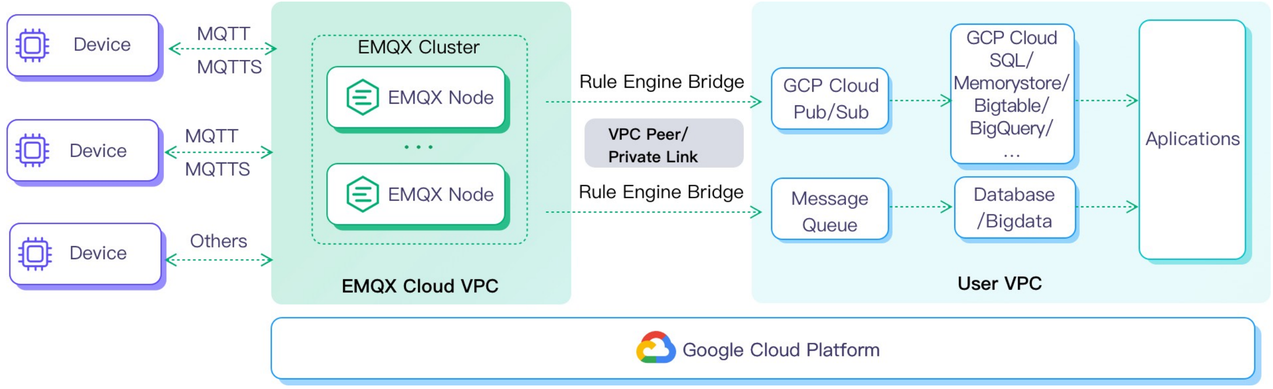
Investigating and sorting out the service usage, deployment architecture, application integration, and resource usage of the application services deployed on Google Cloud that need to be migrated.

Migration Planning

Upon obtaining a thorough understanding of how the business and applications are currently utilized on Google Cloud IoT Core, we will create a comprehensive migration plan that addresses the following three key aspects: service construction, device connection, and data bridging and integration.

Service Construction

EMQ offers a fully managed MQTT Cloud service, EMQX Cloud Service, that allows users a quick and easy setup of an optimally configured EMQX cluster on Google Cloud, utilizing servers and network resources provided by EMQX Cloud. Throughout the usage period, EMQ's global service support team and technical experts provide 24/7 technical support and operation and maintenance services. EMQX clusters are located in a separate VPC network and can be connected to other Google data services or the user's own business applications within their VPC network via VPC peering or private link.



Device Connection

Effective identity authentication for device access is crucial to ensuring system and data security. This manual presents a solution for migrating MQTT devices from GCP IoT Core to EMQX service. EMQX offers robust connection authentication with two levels of security. Firstly, the MQTT protocol itself provides authentication methods such as specifying the username and password in the CONNECT packet or using JWT. Additionally, EMQX supports real-time synchronization of authentication information between internal and external databases. At the transport layer, TLS guarantees client authentication using client certificates and verifies the server certificate to the client for added security.

In GCP IoT Core, devices use a private key to sign a JSON Web Token (JWT) as proof of their identity when communicating with the platform. EMQX offers external authentication support for JWT, allowing for seamless migration via generated password from the original IoT Core key pair to connect to EMQX. Furthermore, EMQX provides 1-way TLS/SSL for IoT Core client connections and supports 2-way TLS/SSL and interfacing with PKI/CA systems to achieve 1-device and 1-key authentication.

Data Bridging and Integration

Both EMQX and Google Cloud IoT Core offer device connectivity and data access in their product designs. In the Google Cloud products, IoT Core is typically connected to GCP Pub/Sub, which bridges device data to data storage and analysis products such as Cloud SQL, Bigtable, and BigQuery. GCP Pub/Sub is an asynchronous, scalable messaging service that decouples message generation from processing.

EMQX provides high concurrent connections and throughput for upstream and downstream data transfer. It also enables seamless integration of device data with third-party databases and data analysis services deployed in the GCP cloud environment. With the EMQX migration solution, data integration is both efficient and stable. By establishing a VPC peering connection between EMQX Cloud and user data service, users can easily bridge device data to GCP Pub/Sub topics with zero code development via the EMQX Management Console.

In GCP IoT Core, devices use a private key to sign a JSON Web Token (JWT) as proof of their identity when communicating with the platform. EMQX offers external authentication support for JWT, allowing for seamless migration via generated password from the original IoT Core key pair to connect to EMQX. Furthermore, EMQX provides 1-way TLS/SSL for IoT Core client connections and supports 2-way TLS/SSL and interfacing with PKI/CA systems to achieve 1-device and 1-key authentication.

Summary

The above key points can guide users in efficiently constructing the solution and provide the necessary architectural guidance for the subsequent migration tasks. If you need additional assistance, please reach out to us at Contact Us .

Step–by–Step Migration: Replacing GCP IoT Core with EMQX Cloud Dedicated

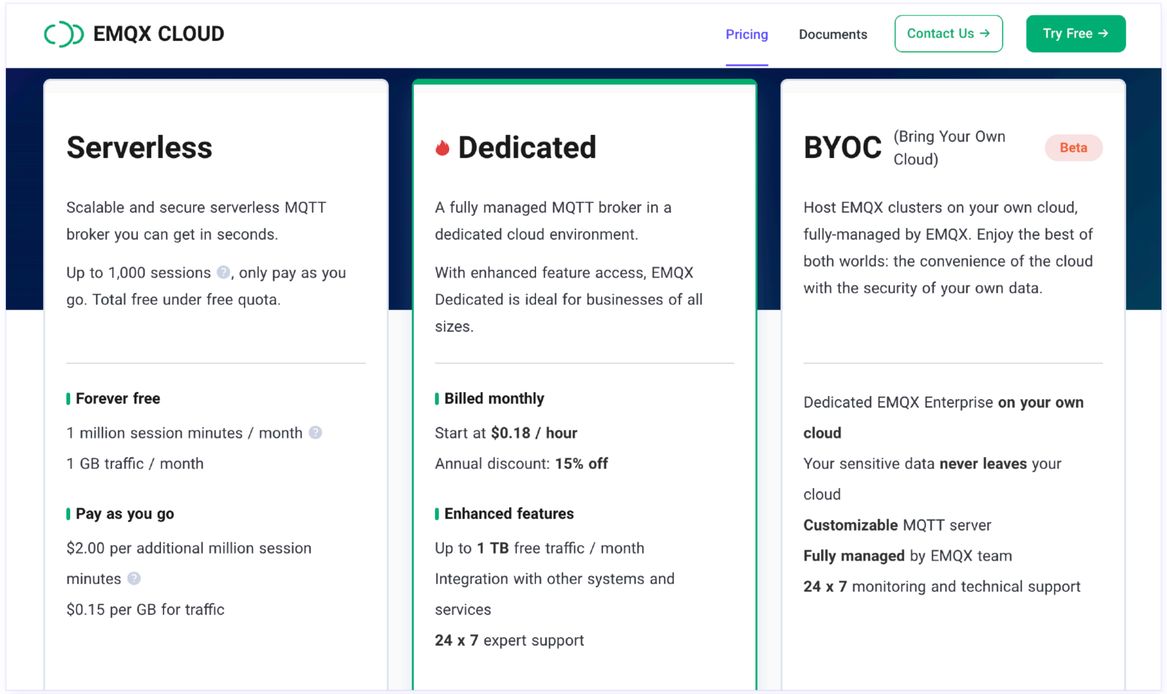
Step 1: Creating an EMQX Cloud Deployment on GCP

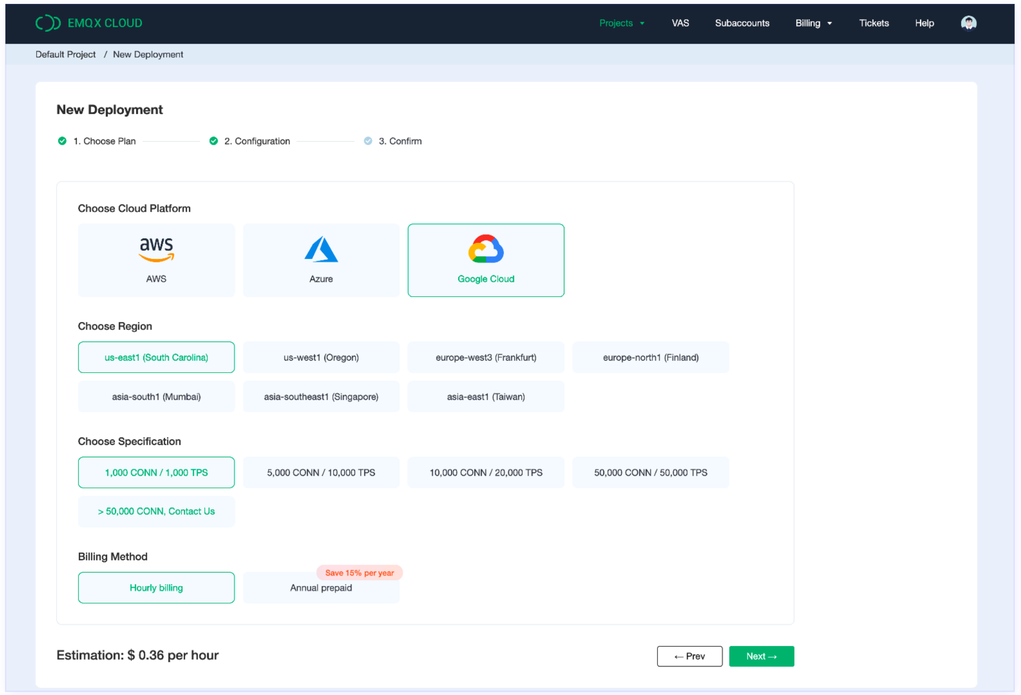
1. Choose a Specification

The initial and critical step is to select the appropriate deployment specification. This requires a meticulous evaluation of business requirements and in–depth preliminary research and analysis to determine the most suitable instance specification. The selected instance must be able to accommodate concurrent connections between devices and applications, satisfy the required message throughput TPS, and support extensive data access and integration. In this section, we have opted for the EMQX Cloud Dedicated – Professional Edition, as it can proficiently meet all these prerequisites.

2. Choose a Region

We suggest selecting the same region as the application services that require data integration.

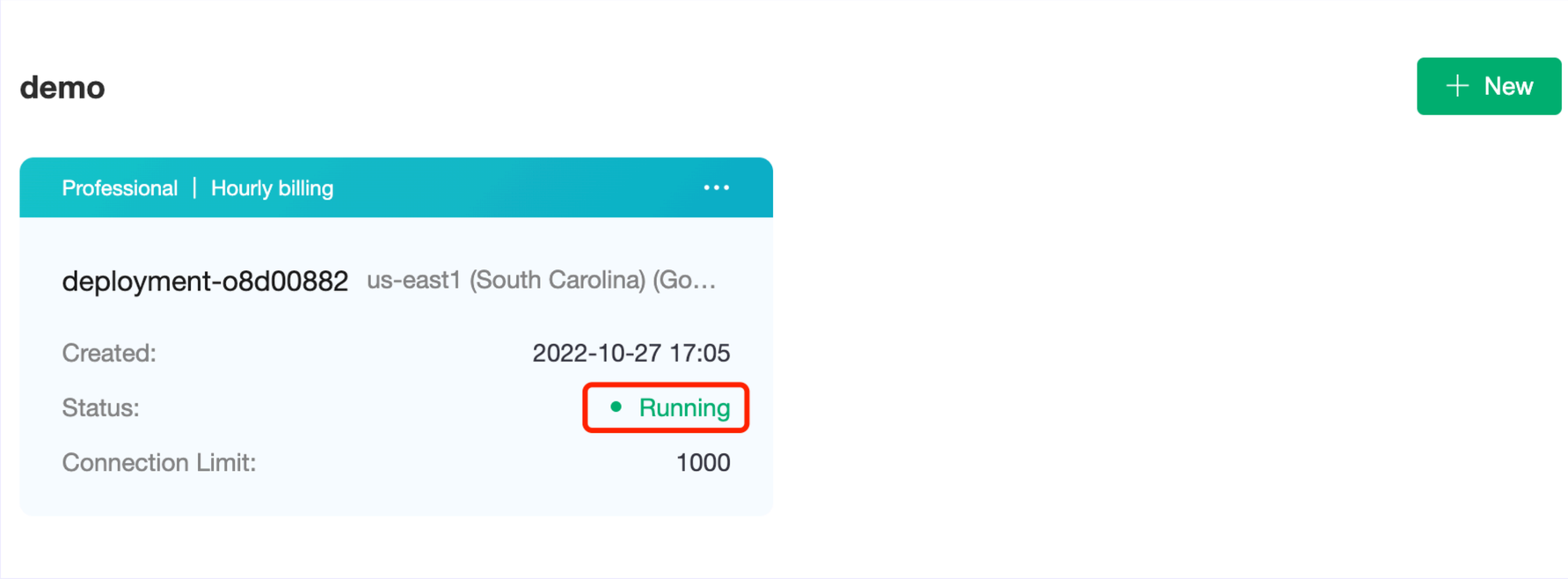




3. Choose a Payment Method

Once you have selected a payment method, click "Next" and EMQX Cloud will automatically create the deployment.

When the status shows "Running,"it is ready to use.

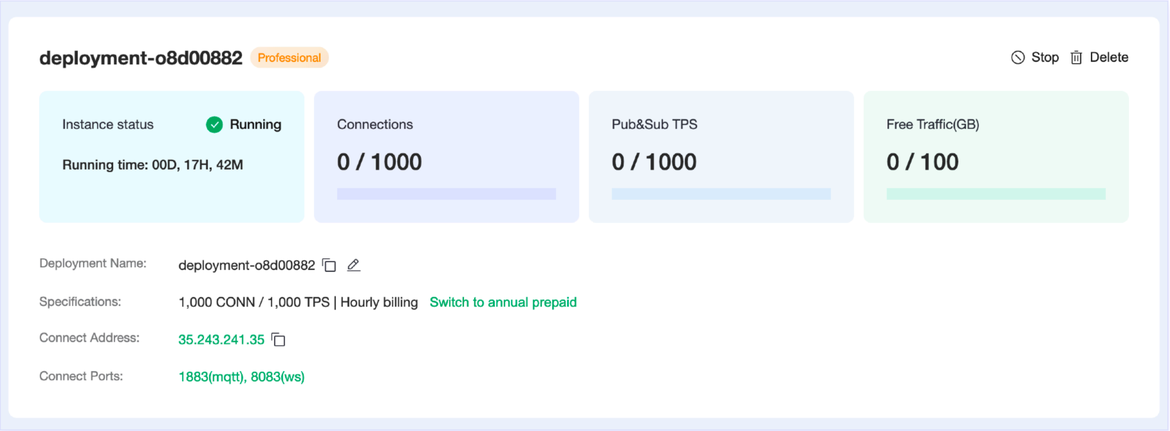


Step 2: Switching Devices Previously Connected to GCP IoT Core to EMQX Cloud Deployment

1. Connection Address and Port

When devices initiate a connection, the connection address and port are the most important pieces of information required. EMQX Cloud's connection address and port are intuitively displayed on the EMQX Cloud console:

- Connection address: IP address for MQTT connection
- Connection port: 1883(mqtt), 8083(ws) are enabled by default



2. Authenticate Device Connections Using JSON Web Token (JWT)

In GCP IoT Core, devices use private keys to sign JSON Web Token (JWT) for secure connection authentication. GCP IoT Core requires each device to prepare a JWT (JWT, RFC 7519) for short–term identity authentication for MQTT connections. When creating an MQTT client connection, the JWT must be passed in the password field of the CONNECT packet. The token is then passed to IoT Core as proof of the device's identity, and the service uses the device's public key (uploaded before sending the JWT) to decrypt and verify the device's identity.

2.1 Parse JWTs in GCP IoT Core

GCP IoT Core uses the public key (or asymmetric) authentication. Devices use a private key to sign a JWT, which is passed to GCP IoT Core as proof of the device's identity.

JWT consists of three parts: header, payload (which includes a set of claims), and signature. The header and payload are JSON objects, which are serialized to UTF-8 bytes and then encoded using base64url. The header, payload, and signature of a JWT are separated by dots. Therefore, a JWT typically looks like the following:

```
{Base64url encoded header}.{Base64url encoded payload}.{Base64url encoded signature}
```

① JWT Header

The header typically consists of two parts: the type of the token and the signing algorithm used. Both fields are required and have only one value each. Cloud IoT Core supports the JWT RS256 and JWT ES256 algorithms.

② JWT Claims

The JWT payload contains a set of claims and is signed using an asymmetric key. JWT claims contain information about the JWT, such as iss (issuer), exp (expiration time), sub (subject), aud (audience), and others. Like the JWT header, the JWT claims set is a JSON object that is used to calculate the signature. Here's an example payload that could be in GCP IoT Core:

```
1.{
2."aud": "your-project",
3."iat": 1509654401,
4."exp": 1612893233
5.}
```

③ JWT Signature

The base64url-encoded header, claims set, and private key (such as the rsa_private.pem file) are signed using the algorithm defined in the header.

The resulting signature is then base64url-encoded to form the JWT. Here's an example of a JWT before base64url encoding:

```
{"alg": "RS256", "typ": "JWT"}.{"aud": "your-project", "iat": 1509654401, "exp": 1612893233}.[signature bytes]
```

Creating public and private keys can be done using the following command:

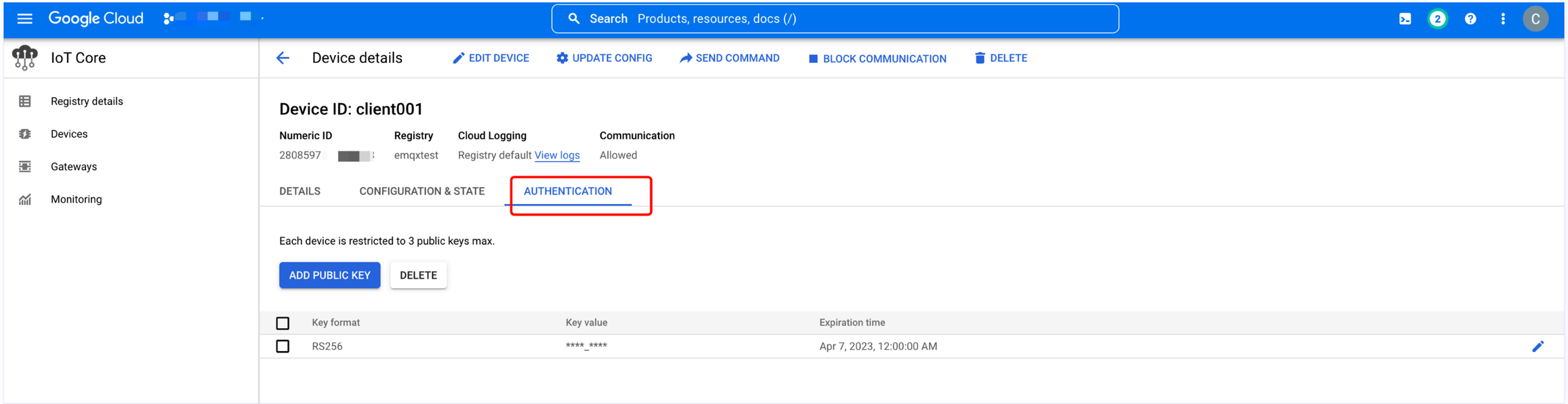
```
1.openssl genpkey -algorithm RSA -out rsa_private.pem -pkeyopt rsa_keygen_bits:2048
2.openssl rsa -in rsa_private.pem -pubout -out rsa_public.pem
```

Where rsa_private.pem is the private key that must be securely stored on the device to sign the JWT of the authentication file, and rsa_public.pem is the public key that must be stored on the server (such as EMQX or Cloud IoT Core) to verify the signature of the JWT.

After the final base64url encoding, the JWT will look like this:

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJlbXEteC1jbG91ZCIsImlhdCI6MTY2NjI0MTU1MSwiZXhwIjoxNjc0MTk3NDQxfQ.SqQsTDZ5kTxRxSnlGs3nHXFRG_Kdjh8GxIsAWyvY4emFB9HbHUH0FHRYbgEbDvFqEPSQWjGKHRz8oXtn2MJFkwQRVfZnhY2fqQuSnFqNc6iGuSqXb-A-v2bc97vuk_x4j-Hguak0sDXzrzj00bwEfOrn4VkkTCPUIgv9EDUZLAbz9PBt0M1WuMrOWKUXgR8Vg8ziIZ0H6T8bT6SfxXqzd94C0APjVP4W8Y5-Vs39SCI1VbtqdKIseg6tQYkLbeJX81gOKC-cFI5MvMchqBrxpUqWahPEmcXYcLo-A6veVmMQ01TK-5uyuABXrsEMUGlE7LjwRbhTTtXfNT8WLMqJOw

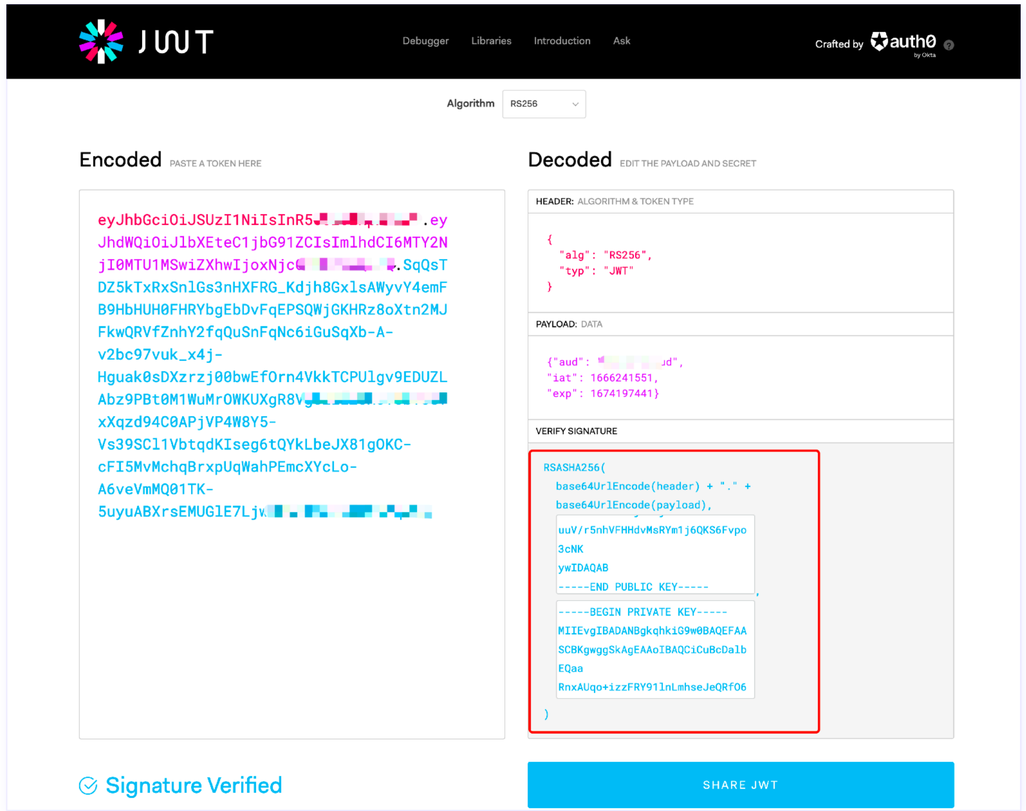
In GCP IoT Core, at least one public key is required for each device connection, which can be obtained from the following page.



GCP IoT Core does not require a specific token generation method. You can use client libraries and tools on JWT.io to generate JWT for connection authentication.

2.2 Migration to EMQX Cloud with Individual Key for Each Client

In GCP IoT Core, each device (ClientID) requires a separate key for JWT authentication. While EMQX Cloud supports JWT as a connection authentication method, it currently only supports a unified key pair for all clients to use. However, EMQX Cloud offers HTTP custom authentication, which allows users to integrate their own JWT authentication service with EMQX Cloud. By leveraging this feature, users can use the JWT generated with the same key pair in GCP IoT Core to authenticate their devices in EMQX Cloud and complete the migration process.



The HTTP custom authentication offered by EMQX Cloud allows users to implement complex authentication logic. In the current scenario, where JWT authentication is required, users can connect EMQX Cloud to their own JWT authentication service, with EMQX Cloud handling the connection and the authentication service responsible for validating the JWT. The authentication process works as follows:

① When a device connects to EMQX, EMQX sends the clientid and password (JWT) carried in the CONNECT packet as parameters to the user's JWT authentication service for verification.

② After receiving the parameters, the JWT authentication service retrieves the public key from the authentication source (which can be a JSON file or Redis) based on the ClientID. The public key is used to decrypt and verify the JWT. If the JWT is valid, the JWT authentication service returns a response status code of 200 to EMQX. If the JWT is invalid, it returns a response status code of 401 to EMQX.

③ EMQX processes the authentication request based on the HTTP response status code returned by the JWT authentication service. If the response status code is 200, the connection is allowed. If it is 401, the connection is rejected.

We recommend the following approach for implementing JWT connection authentication with individual key pairs for each client, to migrate device connections from GCP IoT Core to EMQX:

① Prepare the authentication source in advance for retrieving the client's public key. It is recommended to back up the original client public keys as a public key list that includes the relationship mapping between the clientID and the public key. This list can be saved in a JSON file or Redis. For instance, the public key list is saved in a JSON file as shown below. If the number of devices is large, it is recommended to use Redis to save it.

jwt-public-keys.json

```
{
  "projects/emq-x-cloud/locations/us-central1/registries/emqxtest/devices/client001":  "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAu1SU1LfVLPHCozMxH2Mo\n4lgOEePzNm0tRgeLezV6ffA
  "projects/emq-x-cloud/locations/us-central1/registries/emqxtest/devices/client002":  "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAu1SU1LfVLPHCozMxH2Mo\n4lgOEePzNm0tRgeLezV6ffA
  "projects/emq-x-cloud/locations/us-central1/registries/emqxtest/devices/client003":  "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAu1SU1LfVLPHCozMxH2Mo\n4lgOEePzNm0tRgeLezV6ffA
  "projects/emq-x-cloud/locations/us-central1/registries/emqxtest/devices/client004":  "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAu1SU1LfVLPHCozMxH2Mo\n4lgOEePzNm0tRgeLezV6ffA
  "projects/emq-x-cloud/locations/us-central1/registries/emqxtest/devices/client005":  "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAu1SU1LfVLPHCozMxH2Mo\n4lgOEePzNm0tRgeLezV6ffA
}
```

② Develop a JWT authentication server and deploy it in an environment that can connect to EMQX. The construction of the JWT authentication server can refer to the code example below:

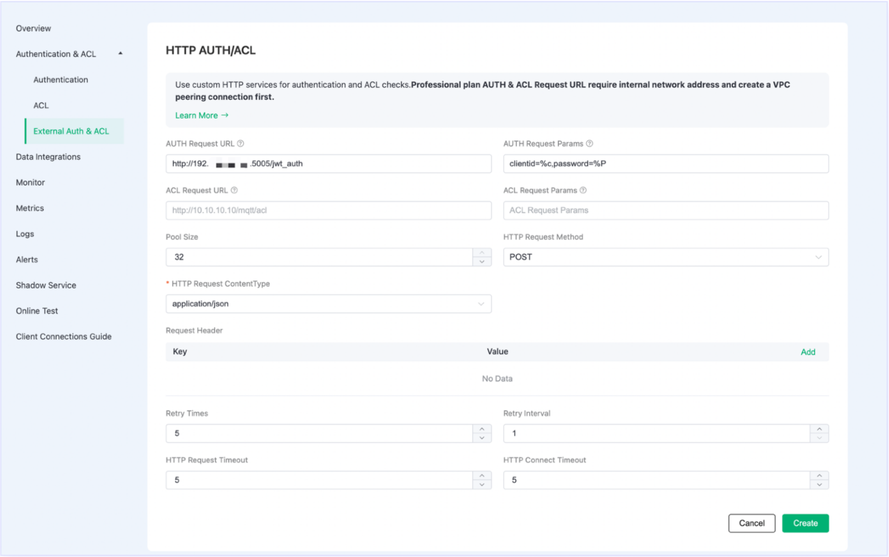
```
import json

import jwt
from flask import Flask, request

app = Flask(__name__)


@app.route("/upload_jwt_public_key", methods=["POST"])
def upload_jwt_public_key():
    # Add the public key to the json file
    request_dict = request.get_json()
    if not request_dict:
        return "bad request", 400

    clientid = request_dict.get("clientid")
    public_key = request_dict.get("publicKey")
    upload_public_key_to_json(clientid, public_key)
    return "", 201
```

```python
@app.route("/jwt_auth", methods=["POST"])
def jwt_auth():
    # Decode the jwt token and return the result
    request_dict = request.get_json()
    if not request_dict:
        return "bad request", 400

    clientid = request_dict.get("clientid")
    password = request_dict.get("password")
    decoded_jwt = decode_jwt_token(clientid, password)
    code = 200 if decoded_jwt else 401
    message = "ok" if decoded_jwt else "auth failed"
    return message, code


def decode_jwt_token(clientid, jwt_token) -> dict:
    # 1. EMQX pass the clientid and password (jwt token) to the jwt server
    # 2. Get the public key from the json file or redis
    # 3. Decode the jwt token with the public key
    # 4. If the jwt token is valid, jwt server will return the 200 code to EMQX
    # 5. If the jwt token is invalid, jwt server will return the 401 code to EMQX
```

```python
# Load the public key
    public_key = read_public_key_from_json(clientid)
    if not public_key:
        return {}
    # Decode the JWT token
    try:
        decoded_jwt = jwt.decode(jwt_token, public_key, algorithms=["RS256"])
    except Exception as e:
        print(e)
        decoded_jwt = {}
    return decoded_jwt


def upload_public_key_to_json(clientid, public_key):
    # Add the public key to the json file or redis?
    with open("./jwt_public_key.json", "r") as f:
        clientid_key_dict = json.load(f)
    clientid_key_dict[clientid] = public_key
    with open("./jwt_public_key.json", "w") as f:
        json.dump(clientid_key_dict, f)
```

```
def read_public_key_from_json(clientid) -> str:
    # Get the public key from the json file or redis?
    with open("./jwt_public_key.json", "r") as f:
        clientid_key_dict = json.load(f)

    return clientid_key_dict.get(clientid)


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5005, debug=True)
```

③ Enable the HTTP custom authentication module in your EMQX Cloud deployment. Go to the Authentication – Custom Authentication. From there, click "Configure authentication" on the starting page to create a new authentication. On the form, fill in the required parameters, such as the authentication request address (which is the address of your JWT authentication server) and the authentication request parameters (which should include the ClientID and Password (JWT)). If there are no special requirements, you can leave the remaining parameters at their default values. You can refer to the following example configuration page for guidance:



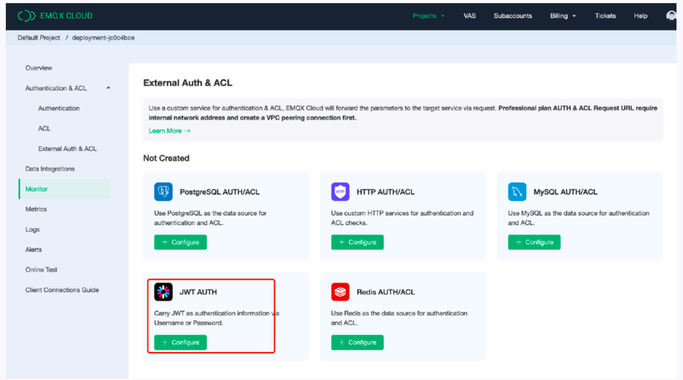
④ After configuring the HTTP custom authentication module and starting it, when a client initiates a connection, EMQX passes the ClientID and JWT carried in the CONNECT packet to the JWT authentication server. The JWT authentication server obtains the public key from the previously prepared public key authentication source (JSON file or Redis) based on the ClientID and then uses the public key to decrypt and verify the JWT. If the JWT authentication server verifies that the JWT is valid, it returns 200 to EMQX, and EMQX allows the client's connection. If the verification fails (possibly due to incorrect clientID or JWT), the JWT authentication server returns 401 to EMQX, and EMQX rejects the connection.

2.3 Migration to EMQX Cloud with Unified key for All Clients

If all clients can use a common key for JWT authentication, the migration process can be more streamlined. Users can enable EMQX's JWT authentication module and configure the public key used to decrypt the JWT to implement JWT authentication. The specific steps are as follows:

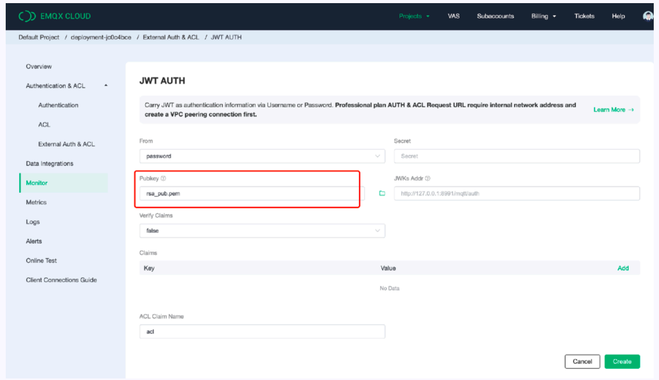
①

Select "Authentication & ACL" –> "External Auth & ACL" on the EMQX console's left sidebar, then select "JWT Auth".



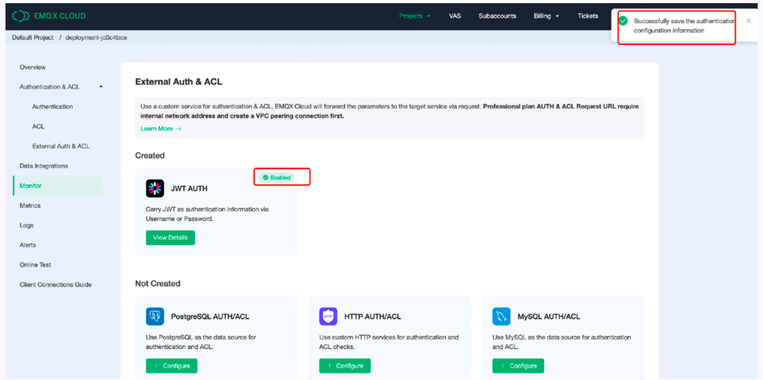
②

Choose "password" in the "From" field and upload the pre–generated public key in the "Pubkey" field. Set other fields to their default values.



③

If the setting is successful, the following pop–up window will appear.



Then use the MQTT X client tool to test the connection authentication:

① Fill in the connection address and port of EMQX Cloud, as well as the ClientID (optional).

② In the Password field, input the encoded JWT.

③ Set other fields to their default values and click "Connect." MQTT X should be in the send/receive state.

④ You can now subscribe to a topic and receive messages without having to register the topic separately, as required in GCP IoT Core.

⑤ To view the connected clients, go to the EMQX Cloud Monitor page.

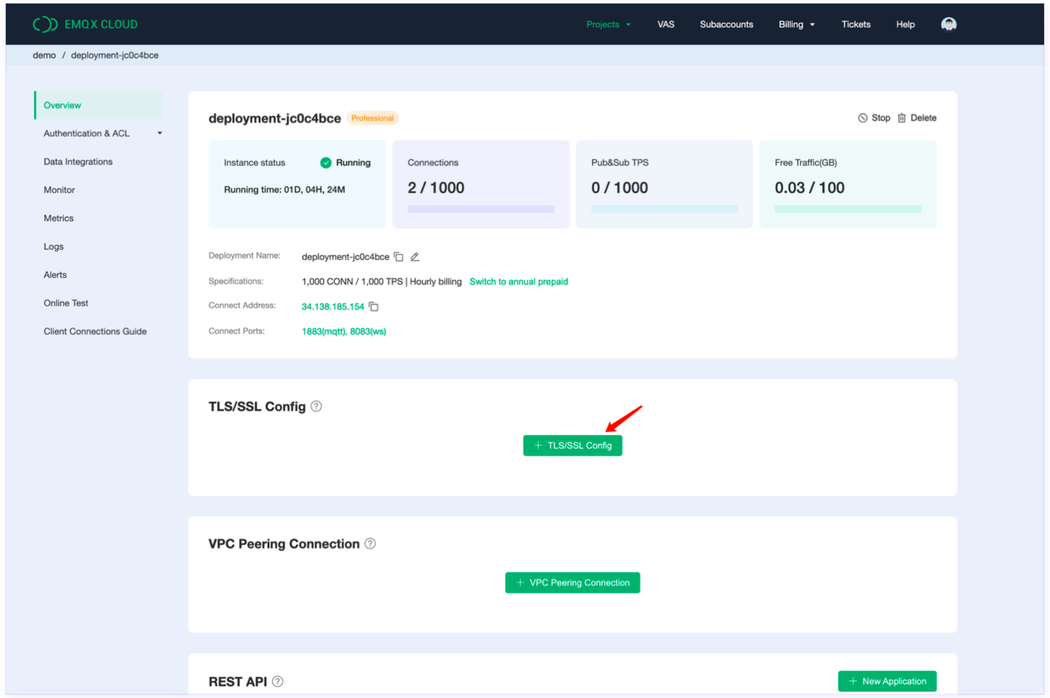
In summary, the EMQX Cloud JWT authentication function can verify device connections using the same pubkey, allowing you to migrate devices easily without having to reset passwords.

3. Implement secure MQTT TLS/SSL Authentication

In GCP IoT Core, client connections use 1-way TLS/SSL, and TLS connections are made through the domain name mqtt.googleapis.com or mqtt.2030.ltsapis.google, using port 8883 or 443. To ensure data security and privacy at the transport layer, EMQX Cloud allows devices to connect to customer-defined TLS/SSL through TLS/SSL ports. The Professional Edition offers 1-way or 2-way TLS/SSL configurations, allowing users to custom certificates. For details please refer to the table shown below:

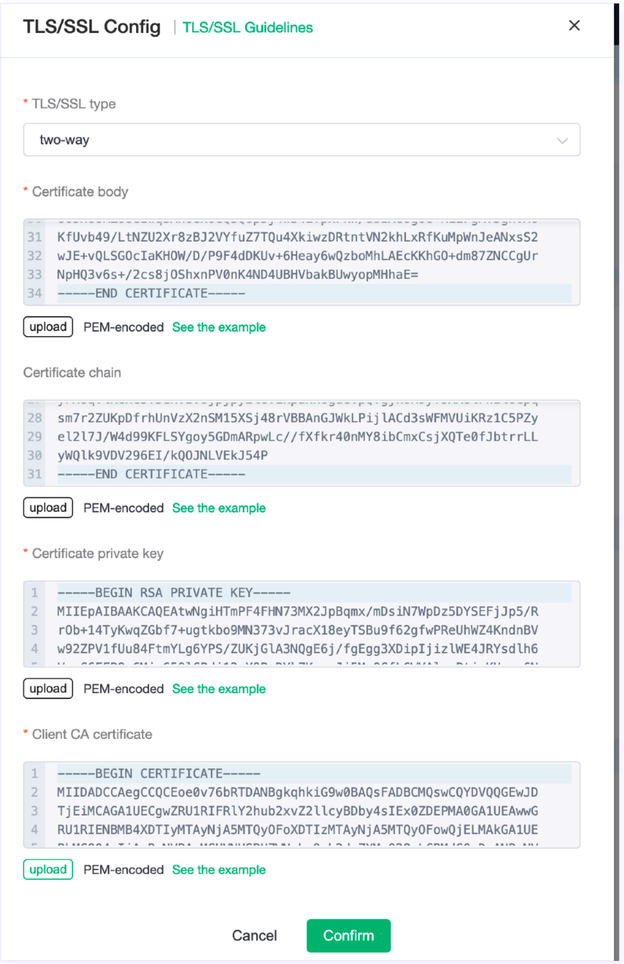
Certification Mode	Support self-signed certificate	Server certificate	Certificate chain	Private key	Client CA certificate
1-way Authentication	Yes	required	required	required	not required
2-way Authentication	Yes	required	required	required	required

To configure TLS/SSL in the EMQX Cloud console, first log in to the console and navigate to the overview page. Click on the "+TLS/SSL Config" button to configure the certificate. You can either upload the file directly or enter the certificate content in the pop-up window.



You will need to fill in the following information:

- Authentication type: You have the option to select either 1-way authentication, which involves solely authenticating the server's certificate on the client side, or 2-way authentication, which necessitates both the client and the server to authenticate each other's certificates.
- Certificate body: Upload the server-side certificate in the appropriate format.
- Certificate chain: If a third party issued the certificate, you may need to provide the certificate chain.
- Certificate private key: Upload the server-side private key in the appropriate format.
- Client CA certificate: This is required for 2-way authentication.



After confirming the TLS/SSL configuration, check if the status is running. If it is, then the configuration is successful. To test and verify MQTT TLS/SSL connections, use the MQTT X client tool by following these steps:

① Create a new connection by providing a name. The client ID is randomly generated.

② Fill in the host information with the connection address and port. For SSL connection, select mqtts:// and port 8883, for WebSocket with SSL, select wss:// and port 8084.

③ Input the authentication information you previously created, including the username and password.

④ Select "true" for SSL/TLS.

⑤ Select a certificate. For third-party authentication certificates, no CA certificate is required. For self-signed certificates, a server-side CA certificate is required. For 2-way authentication, a client certificate and private key are necessary.

⑥ Turn on strict mode.

⑦ Click on the "Connect" button.

Step 3: Integrating with GCP Pub/Sub and Third-Party Data Services

After completing the deployment and client connection steps, you can integrate with GCP Pub/Sub through EMQX's data integration capabilities. You can also integrate with other data storage and analysis services deployed on GCP through VPC peering connections.

1. Bridging Data to GCP Pub/Sub

This section describes how to use EMQX Cloud's data integration capabilities to bridge data to the GCP Pub/Sub and transfer data to other data analysis applications or database systems. We will simulate sending temperature and humidity data to EMQX Cloud and then use data integration to bridge them to GCP Pub/Sub.

(Note: GCP Professional deployment uses a private network by default, so no additional network configuration is needed.)

1.1 Introduction to GCP Pub/Sub

GCP Pub/Sub is an asynchronous and scalable message transport service that effectively decouples message generators from processors. It allows for communication between services asynchronously with a delay of roughly 100 milliseconds. Pub/Sub is widely used for stream analysis and data integration pipelines for acquiring and distributing data. It can be utilized as message-oriented middleware for service integration and as a queue for parallelizing tasks. With Pub/Sub, you can create systems that have event producers and consumers, also known as publishers and subscribers. Publishers broadcast events and communicate asynchronously with subscribers without the need for synchronous remote procedure calls (RPCs). Publishers send events to Pub/Sub without worrying about how or when these events will be processed. Pub/Sub then delivers the events to all services that respond to them. In a system that uses RPC communication, publishers have to wait for subscribers to receive data. Asynchronous integration in Pub/Sub enhances the flexibility of the entire system. To get started quickly with Pub/Sub, you can refer to the Quickstart using Google Cloud console. For more information, please see Building a Pub/Sub messaging system.

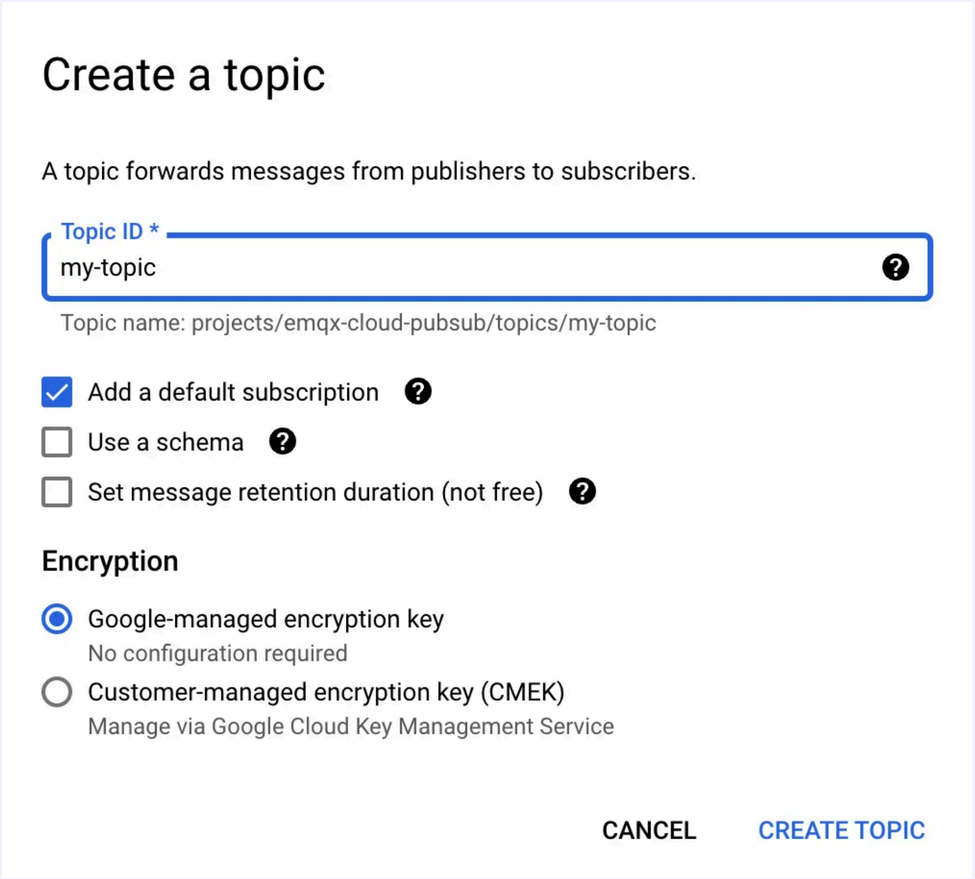
1.2 Creating a GCP Pub/Sub Deployment

① Refer to step 1 to create an EMQX Cloud deployment on GCP.

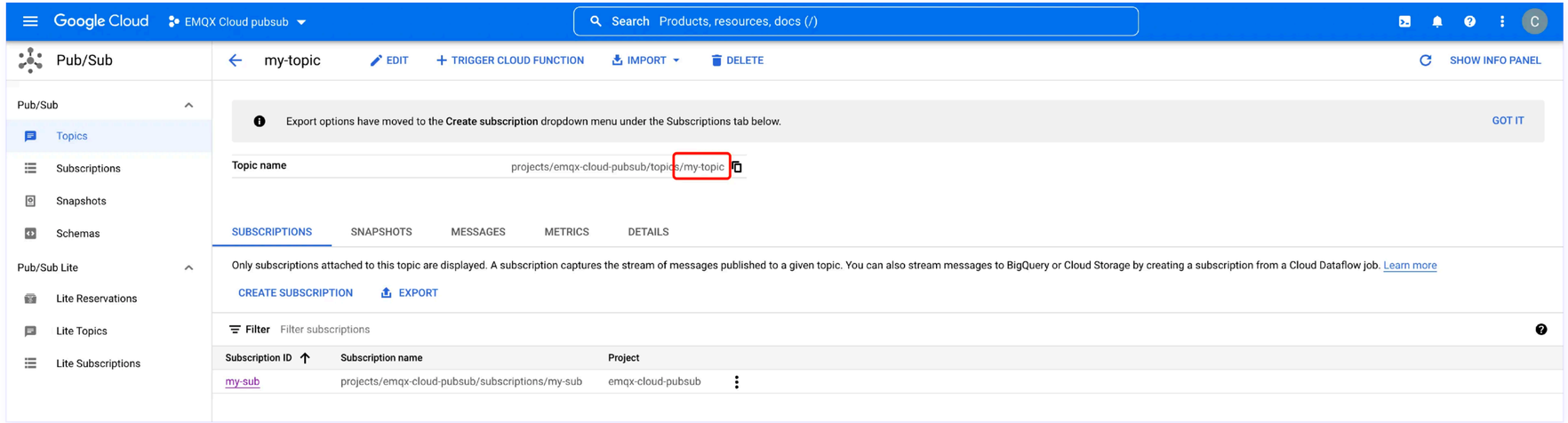
② Create a GCP Pub/Sub instance (if you are using GCP Pub/Sub for the first time, please refer to the documentation).

③ Create a GCP Pub/Sub topic.

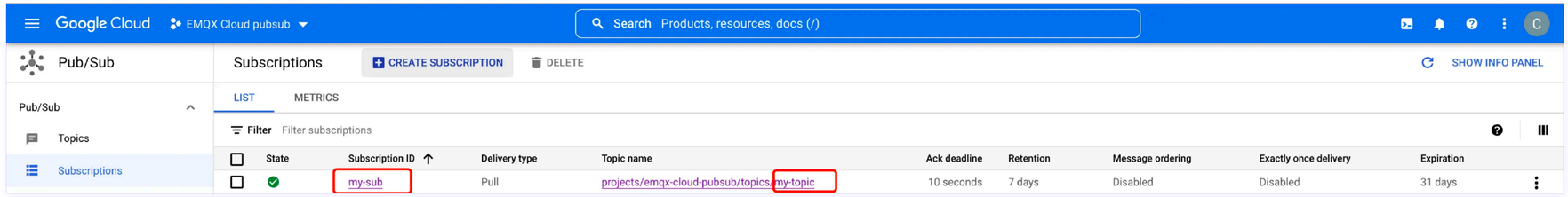
Navigate to the GCP Pub/Sub console and click "Create Topic". Input "my–topic" as the Topic ID to create a new topic.



Click on the Topic ID to view specific information about the topic.



Click on "Subscriptions" to view details about the subscribed topic and the result of message forwarding.



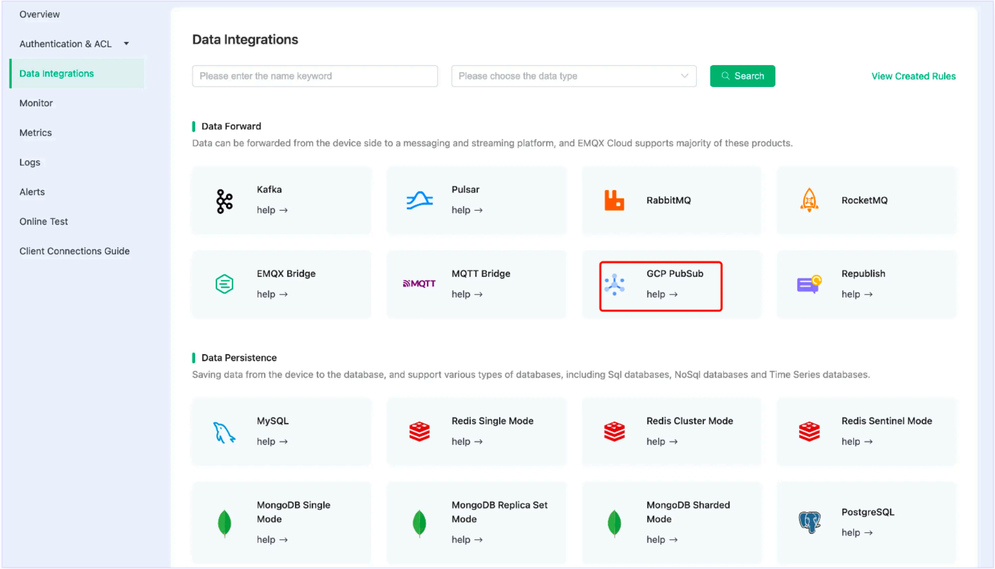
1.3 Configuring Data Bridging

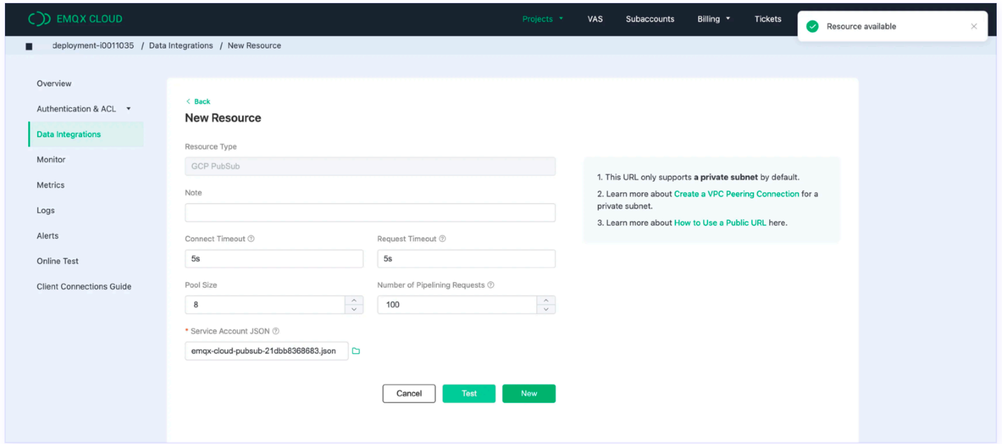
① Create a GCP Pub/Sub resource in EMQX Cloud.

② Go to the Data Integrations page of EMQX Cloud and click on the item "GCP Pub/Sub".

Input the Service Account JSON and other specific information, then click on "Test." If the test fails, please check the status of the GCP Pub/Sub service. (*To obtain the Service Account JSON, go to the GCP console, select the corresponding project, navigate to IAM & Admin, select "Service Accounts," select the email associated with the account, click on "Keys," and generate an authentication JSON file.)

③ Once the test has passed, click on the "New" button, and you will receive a prompt message saying "Resource created successfully."

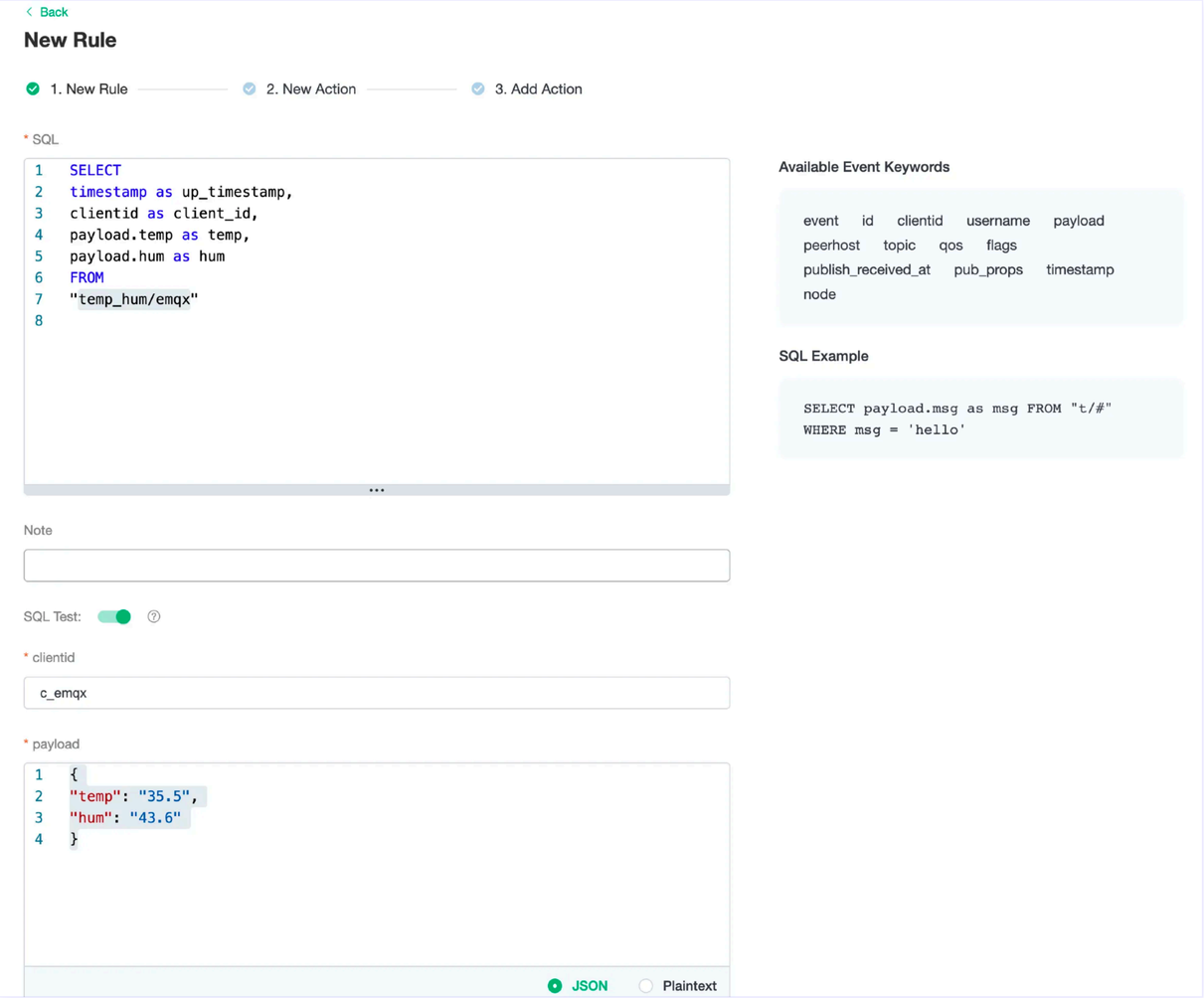




④ Create a rule.

Input the following SQL statement into the SQL input box. The reporting data time (up timestamp), Client ID, and message payload will be extracted from the temp hum/emqx topic in the SQL rule. The device's environmental temperature and humidity data can then be read from the message payload.

```
1.SELECT timestamp as up_timestamp,
3.clientid as client_id,
5.payload.temp as temp,
7.payload.hum as hum
9.FROM
11."temp_hum/emqx"
```



You can enable "SQL Test" below the SQL input box to input specific messages and click "SQL Test" button to view the result.

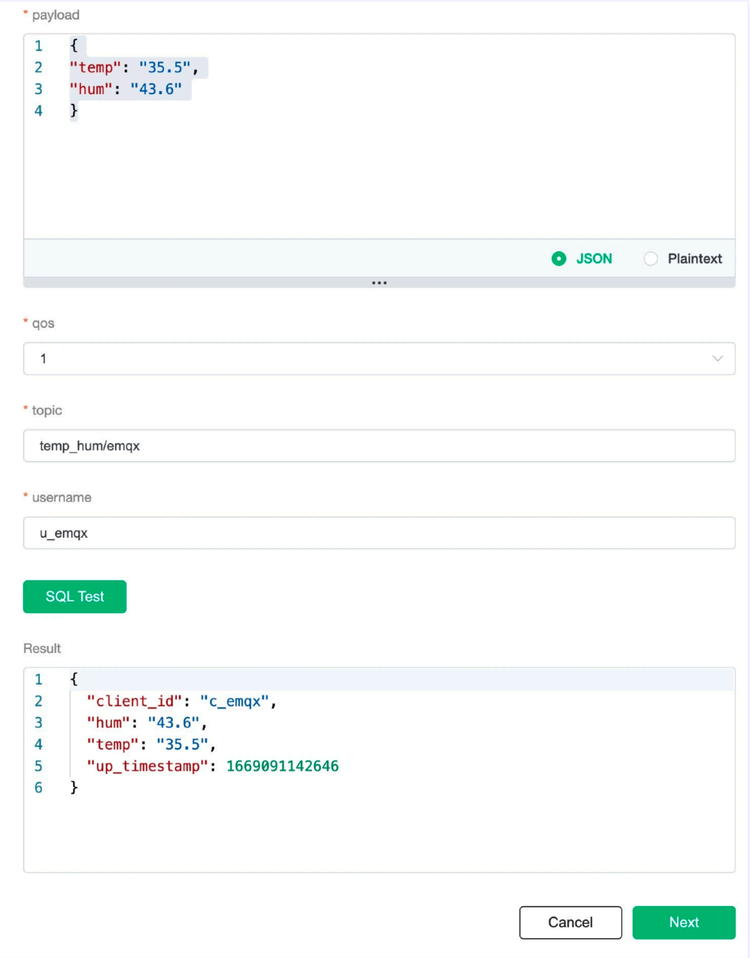
If configured correctly, the "Result" output box will show the complete JSON data as follows.

⑤ Add an action to the rule.

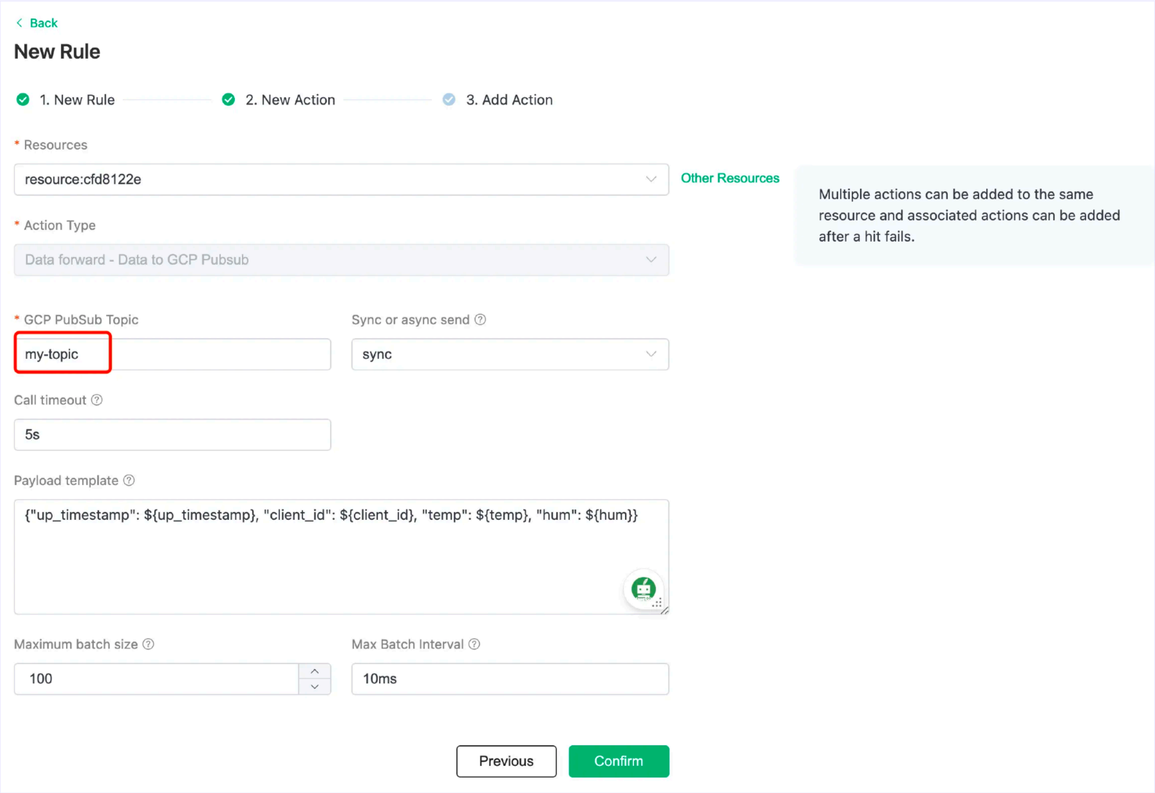
Click on "Next" to set up an action to forward data to GCP Pub/Sub for the SQL rule you just created.

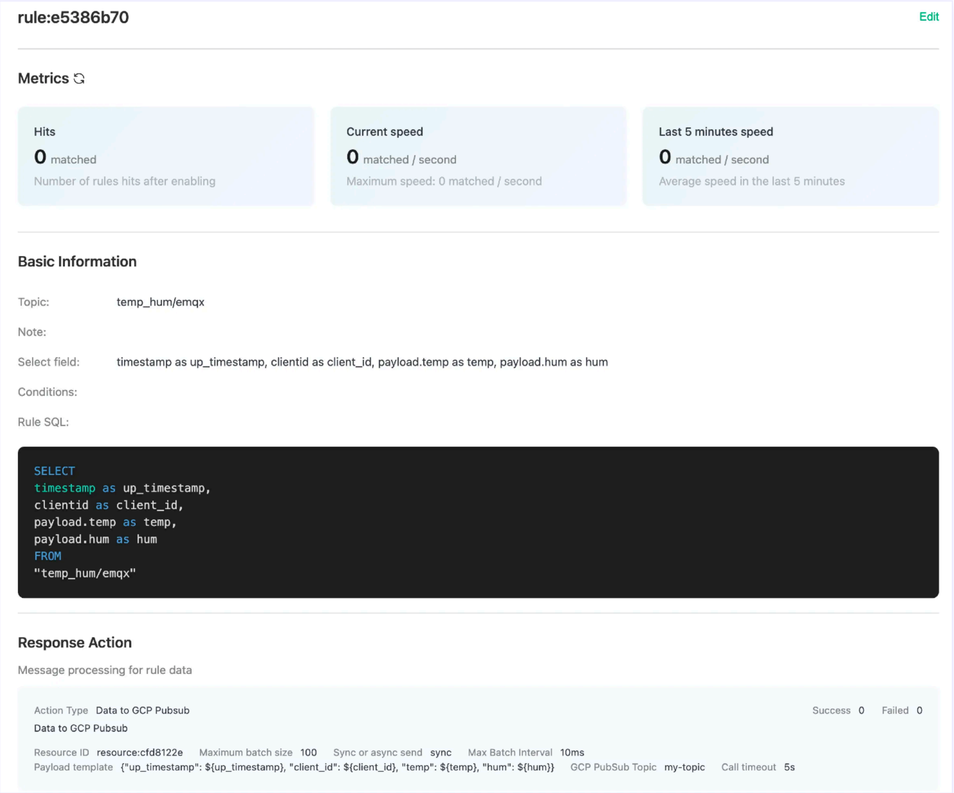
To demonstrate how to forward device-reported data to GCP Pub/Sub, we will use the following GCP Pub/Sub topic and message template.

```
# GCP Pub/Sub message template
{"up_timestamp": ${up_timestamp}, "client_id": ${client_id}, "temp": ${temp}, "hum": ${hum}}
```

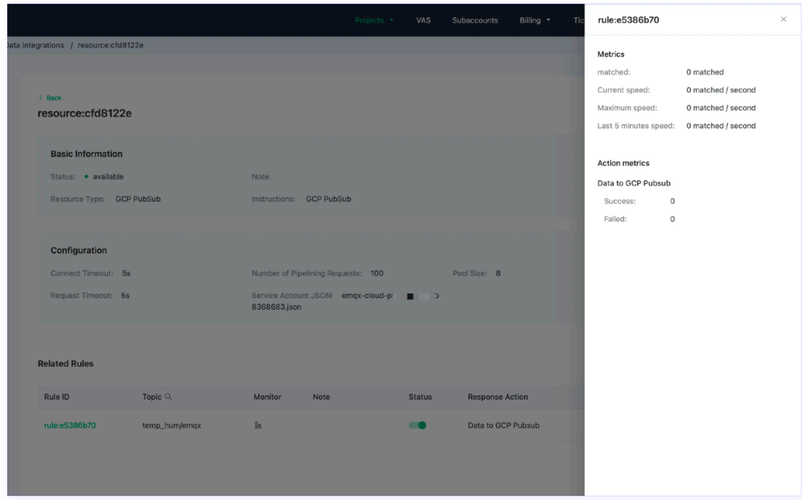


⑥ After successfully binding the action to the rule, you can click on "View Details" to view the SQL rule and corresponding action.





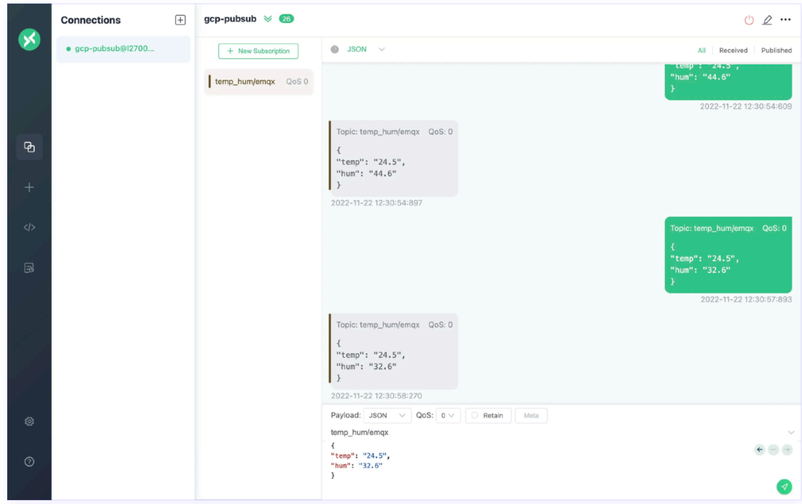
⑦ You can access all created rules through Data Integrations –> View Created Rules. To view detailed information about a specific rule, simply click "Monitor."



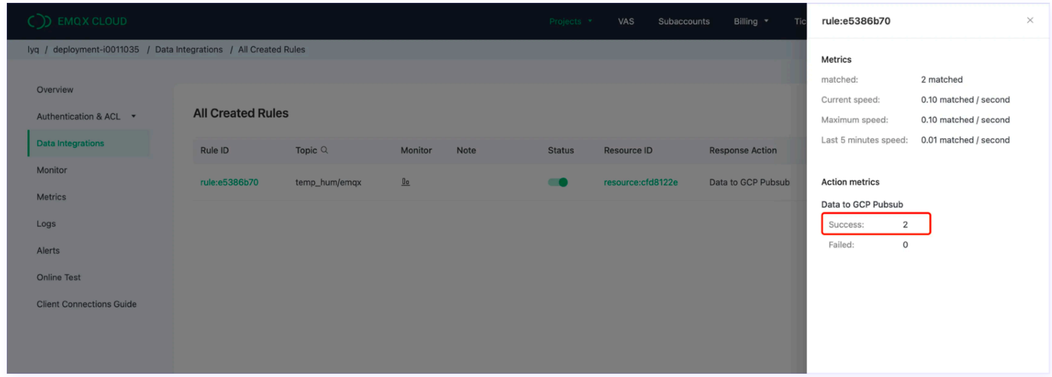
1.4 Validating Data Bridging

① Use the MQTT client tool to simulate reporting data.

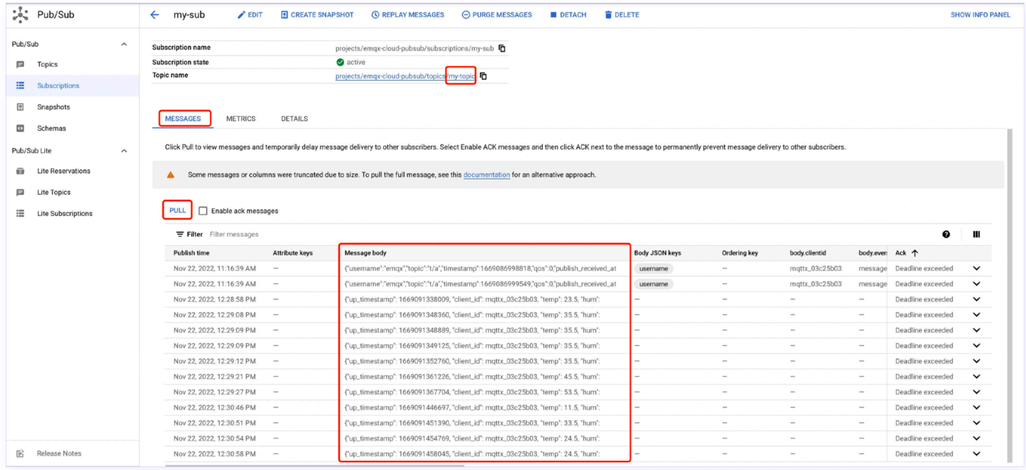
Click the "Add connection" button and input the EMQX Cloud connection information to connect to your deployment. Replace broker.emqx.io with the connection address of your deployment and add client authentication information to the EMQX Cloud console. Once connected, input the topic name and payload to publish the message.



② To ensure proper data forwarding, verify the "Success" count in the rule monitor. If the count has increased correctly, it confirms that EMQX Cloud has successfully bridged and forwarded data as per the defined rule.



③ Go back to the GCP Pub/Sub console, go to the "Subscriptions" page, select "MESSAGES" and GCP Pub/Sub will retrieve and display the results.



2. VPC Peering Connection

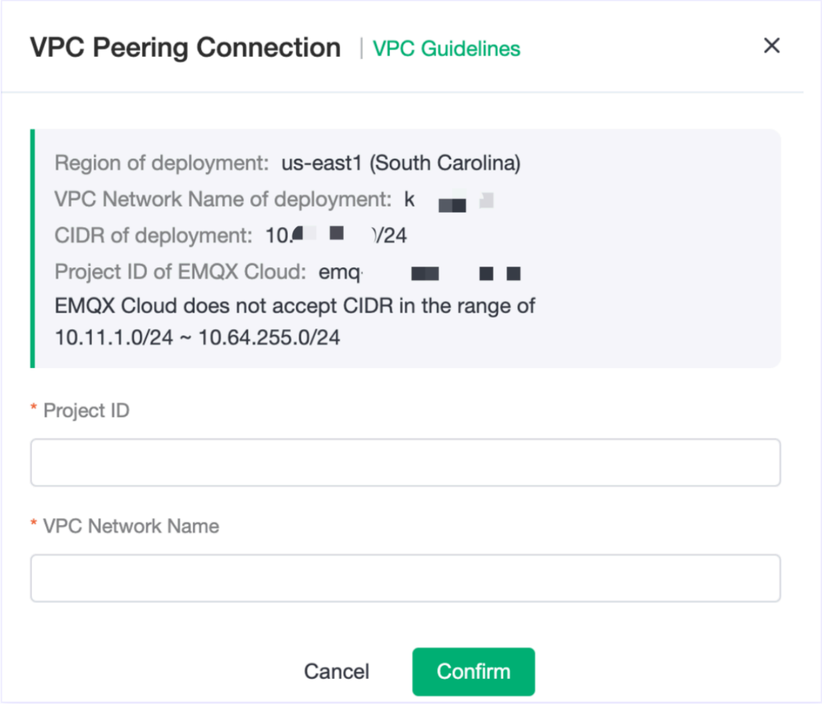
In some production scenarios, it may be necessary to establish VPC peering connections to connect the VPC where the EMQX Cloud deployment is located with the user's VPC. This is done to integrate device data uploaded to EMQX with services and applications deployed in the user's VPC. The following steps demonstrate how to establish VPC peering connections.

After logging in to the EMQX Cloud console, navigate to the deployment overview and click on "+ VPC Peering Connection." Fill in the required information and remember the relevant details for later use.

VPC Network Name of deployment

Project ID of EMQX Cloud

CIDR of deployment



The "Project ID" and "VPC network name" can be found in the "VPC network peering" section of the GCP console.

To create the peering connection in GCP console:

① Go to the "VPC network peering" section of the GCP console.

② Click "Create peering connection" and then click "Continue."

③ Input a name for the peering connection in the "Name" field.

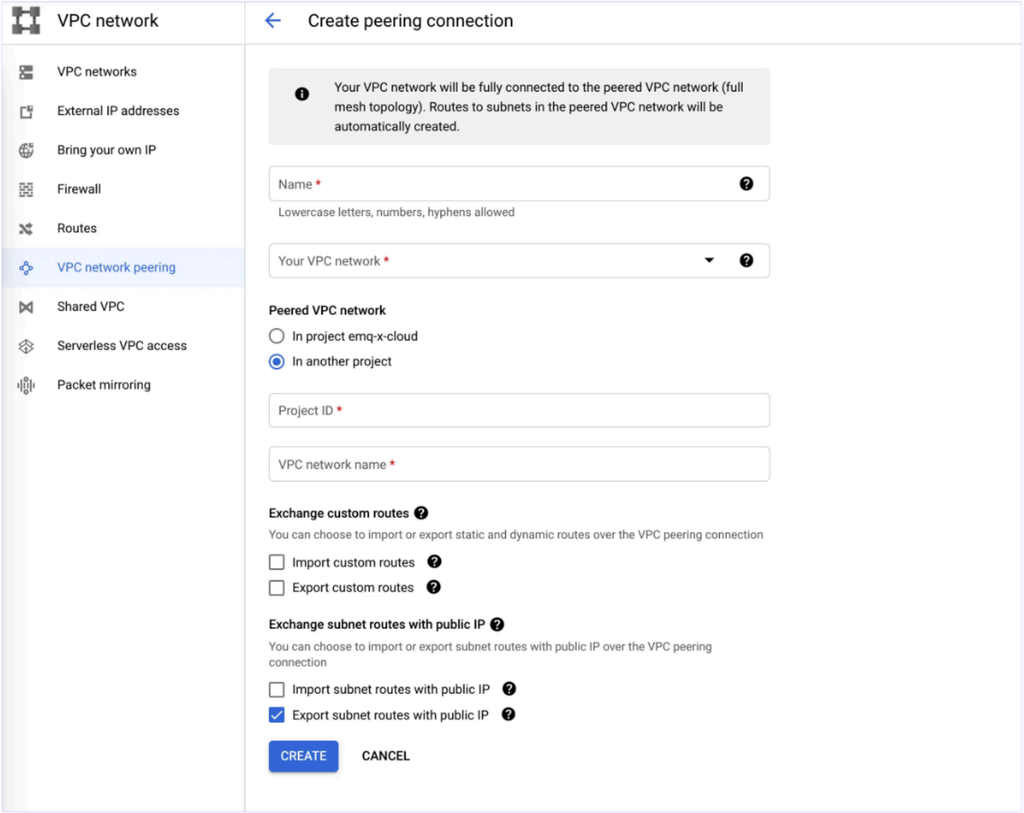
④ In the "Your VPC Network" field, input the name of your GCP VPC network.

⑤ In the "Peered VPC network" field, select "In another project."

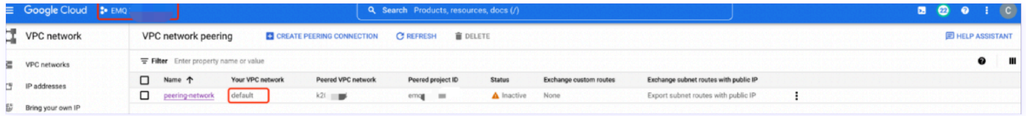
⑥ In the "Project ID" field, input the EMQX Cloud Project ID (which can be found in the EMQX Cloud's "VPC peering view").

⑦ In the "VPC network name" field, input the EMQX Cloud VPC network name (which can be found in the EMQX Cloud's "VPC peering view").

⑧ Click "CREATE."

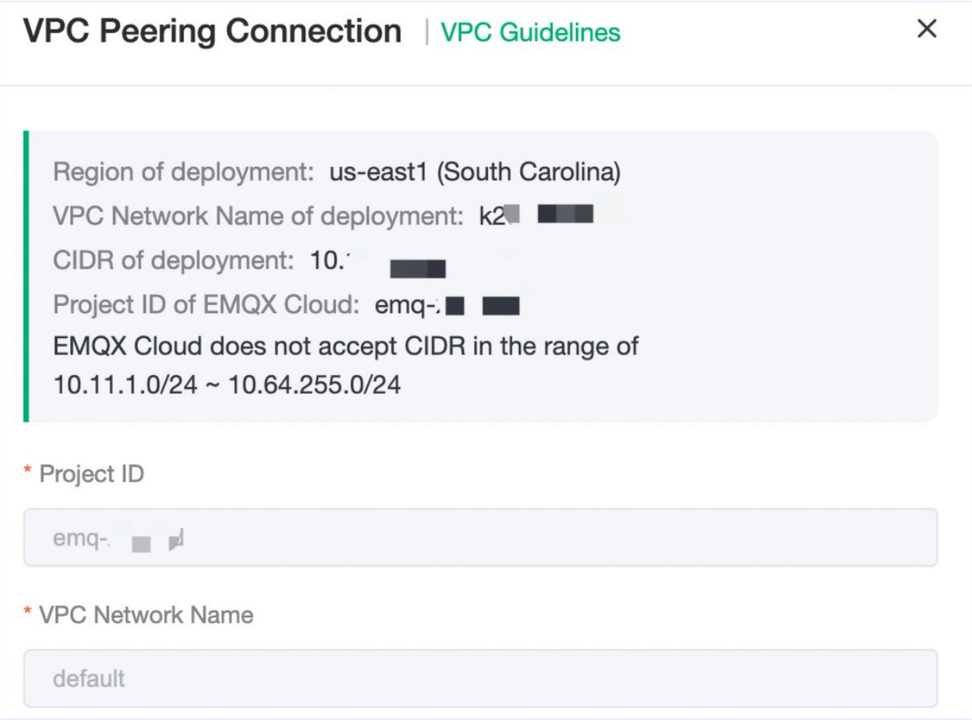


Provide the required information on "VPC network peering" in GCP to the EMQX Cloud VPC configuration panel.

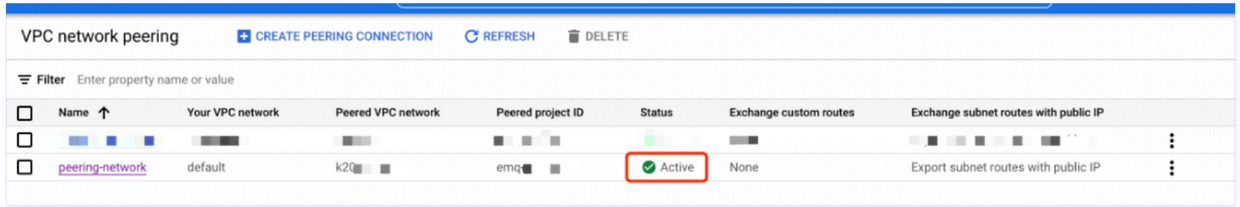


EMQX CLOUD | Google Cloud

- Project ID: GCP Project ID of your peering VPC
- VPC Network Name: Your VPC network



Once all configurations have been correctly established and initiated, the peering connection status on GCP will indicate "Active."



The VPC peering status on the EMQX Cloud will be displayed as "Running."

EMQX CLOUD | Google Cloud

Next, create a firewall rule to allow EMQX Cloud deployment to access the GCP network:

① Click on "Firewall" and then "CREATE FIREWALL RULE."

② In the "Network" field, select your GCP network.

③ In the "Targets" field, select all instances in the network or make selections based on your specific needs.

④ In the "Source IP ranges" field, input the CIDR of the EMQX Cloud VPC you previously recorded.

⑤ Choose the appropriate protocols and ports.

k14 8	Ingress	Apply to all	IP ranges: 10.22. 7.0/24	all	Allow	1000	default	Off

3. Integrating Data Through EMQX Cloud

EMQX Cloud provides rich data integration capabilities, enabling users to connect to third-party data storage and processing services deployed in their VPCs via VPC peering. In the following scenario, we demonstrate how to integrate EMQX Cloud with Redis, a data storage service deployed in the user's VPC.

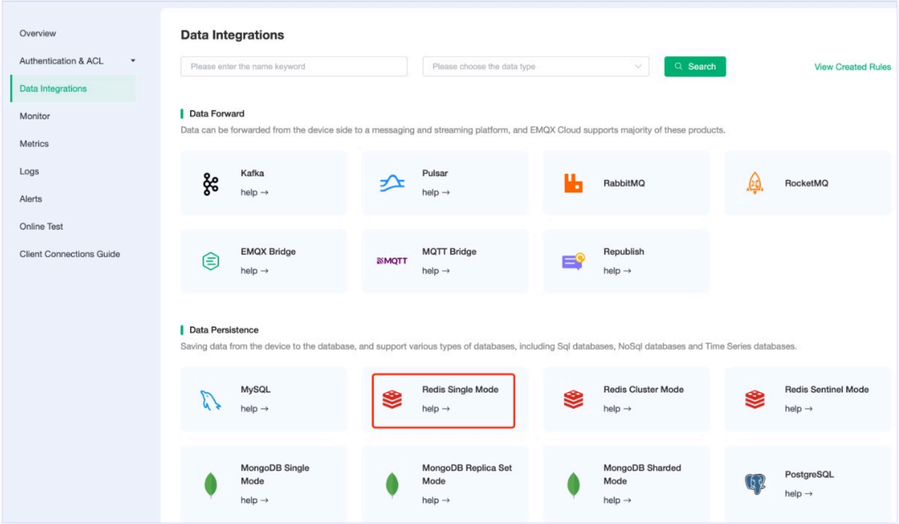
Firstly, we create a VM instance in the GCP console and install Redis using Docker as our data storage.

3.1 Installing Redis

The following command is run to launch a Redis container:

```
1. docker run -itd --name redis -p 6379:6379 redis:latest
2. docker exec -it redis redis-cli
```

After Redis is ready, the next step is to configure the data integration with EMQX Cloud.



3.2 Save data to Redis

Go to EMQX Cloud Data Integrations and select item "Redis Single Mode."

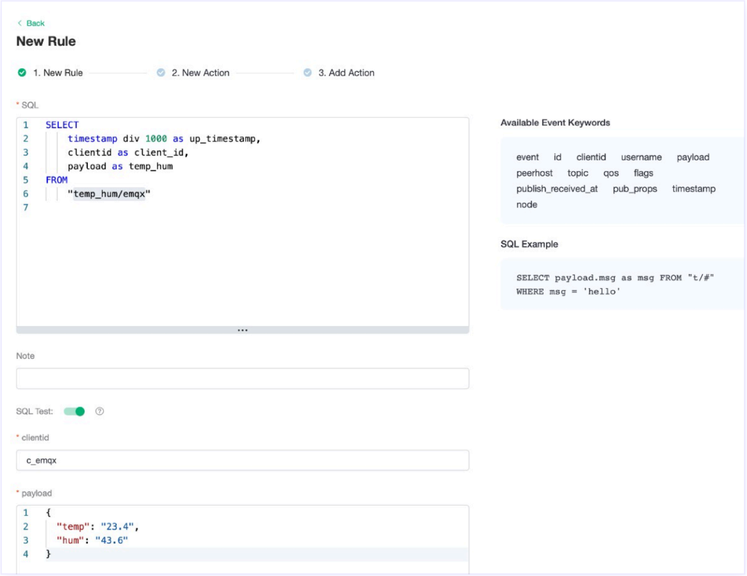
① Create a Redis Single Mode Resource

Configure the Redis server information, including its IP address and port. Click the "Test" button to verify the resource's availability, then click "New" to create the resource. If the resource initialization fails, check your VPC peering connection and firewall configurations. Additional information can be found on the log details page.

② Add a Rule to the Rule Engine

Once the resource is ready, click "New" to enter the rule creation page to add a new rule. The rule is triggered when a client sends a temperature and humidity message to the temp_hum/emqx topic. For example:

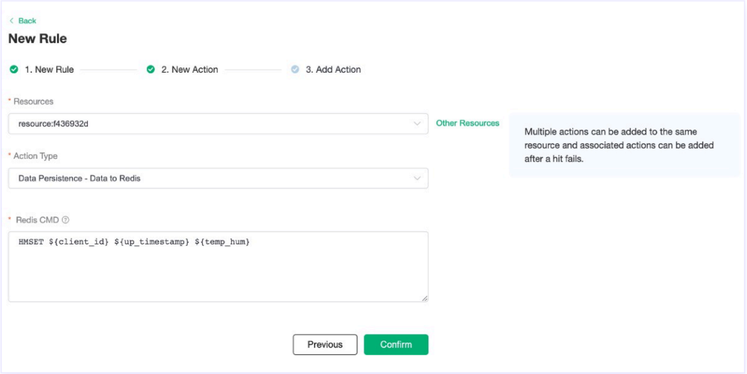
```
SELECT timestamp div 1000 as up_timestamp, clientid as client_id, payload as temp_ FROM
"temp_hum/emqx"
```



③ Create an “Action.”

After configuring the rule, click "Next" to configure and create an “Action.” The objective is to read the up_timestamp, client ID, temperature, and humidity from the topic and save them to Redis.

```
HMSET ${client_id} ${up_timestamp} ${temp_hum}
```



Click "Confirm" to complete the rule configuration. You can use the MQTT X tool to connect and verify the data transfer.

Summary

This handbook provides a comprehensive guide for EMQX Cloud cluster deployment, device access, authentication, data integration and bridging. These steps ensure a seamless migration from the GCP IoT Core service to the EMQX Cloud service with minimal modifications to devices. By implementing these steps, it guarantees access and collection of application data and facilitates a smooth service migration process. If you need additional assistance, please reach out to us at Contact Us .

EMQ Technologies Inc.

The World's Leading Provider of Open Source IoT Data Infrastructure

www.emqx.com Email Twitter YouTube