# RabbitMQ

# Overview Guide

| | |
|---|---|
| **Solution** | Cypress VUE |
| **Revision** | Rev 1.0 |

Overview: RabbitMQ and Cypress VUE
3066 Beta Avenue | Burnaby, B.C. | V5G 4K4

# Revision Control

| Description | Revision | Date |
|---|---|---|
| Initial Release | Rev 1.0 | 2021-11-17 |

# Overview

RabbitMQ is a message broker that allows quick and easy access to sending messages to a third party external to Cypress VUE. This is the recommended approach by Cypress VUE for third parties to collect large volumes of data.

The RabbitMQ system consists of three main parts:

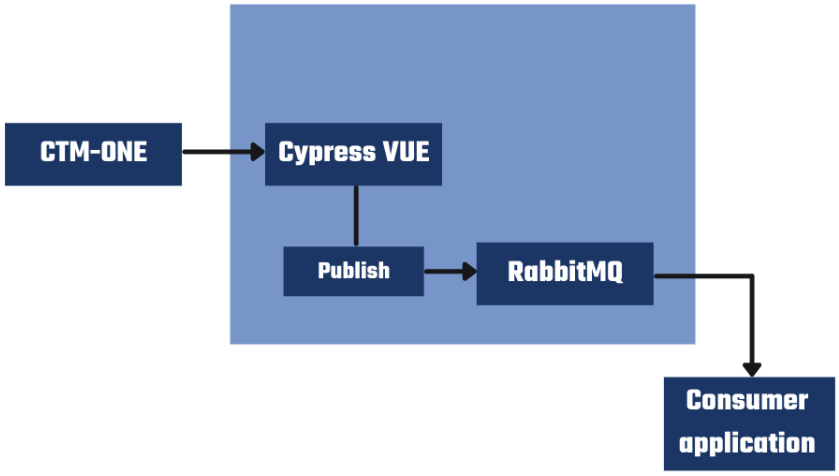
1. **Producer** - the item that creates the messages. In our case, Cypress VUE publishes device-mapped data into the queue.
2. **Queue** - list of messages waiting to be read.
3. **Consumer** – the application that reads messages out of the queue (this is an application that you would write).


As data arrives at our system, it is published by Cypress VUE into the Queue. The consumer (Customer Application) is notified that there is a message(s). Those messages are removed from the queue and are processed on the consumer's side. Once removed from the queue, the message is considered "acknowledged".

The queue can be set up with a TTL (time to live) of one hour. This means that a message will sit in the queue for up to one hour waiting to be processed. If there is no processing activity (i.e., the consumer goes offline), the message will be purged. This prevents the queue from growing too large if the consumer goes down for an extended period.

The data queues are "Durable". This means that if our server crashes unexpectedly, the messages are not lost. When the server starts again, the queue will have the same messages prior to the crash.

For this process, the CTM-ONE wireless gateway communicates with Cypress VUE. Consequently, Cypress VUE performs Device Mapping on the data and publishes data to the Queue, which in turn informs the consumer that data is available for consumption.




## Access Protocol

RabbitMQ supports several messaging protocols, directly and via the use of plugins.

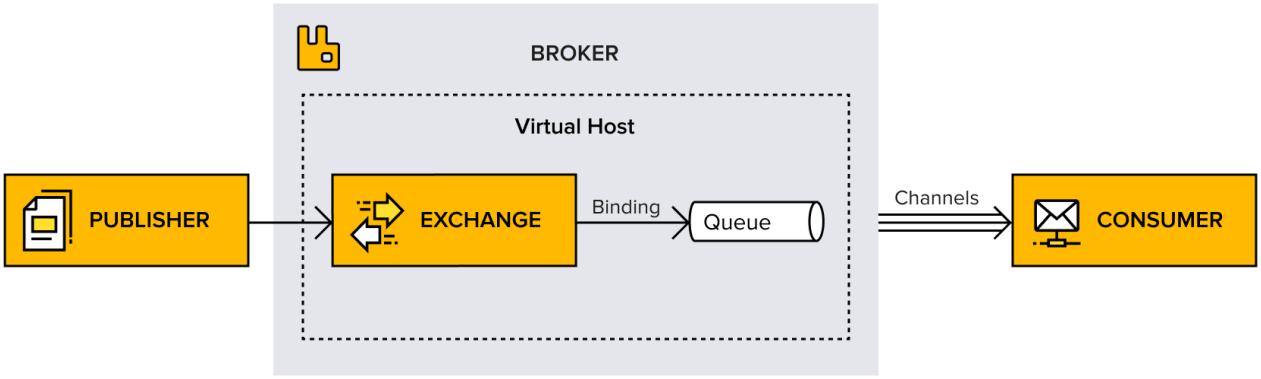
RabbitMQ was originally developed to support AMQP 0-9-1. AMQP 0-9-1 is a binary protocol and defines strong messaging semantics. For clients, it's a reasonably easy protocol to implement, and as such, there are many client libraries available for different programming languages and environments.

AMQP 0-9-1 is the protocol used by RabbitMQ. The data gets pushed to the consumer for an acknowledgment before it gets removed from the queue.

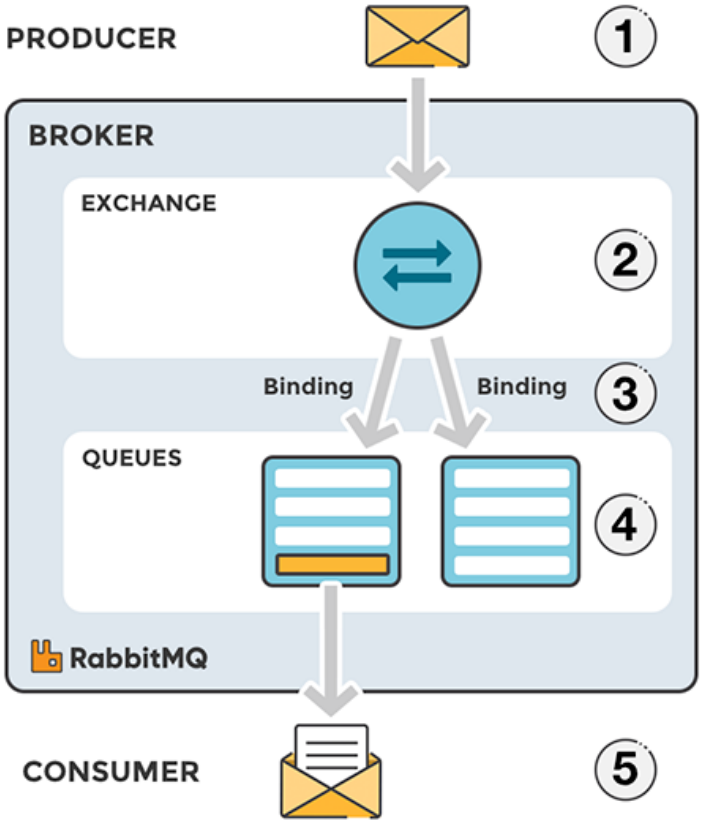
## Access Security

A virtual host is set up for each client on RabbitMQ. A user will be created with a username and password to connect to this virtual host. Exchange is set up to manage the different queues.

For this process, the device reports to Cypress VUE. The data is routed to the appropriate virtual host and exchange/queue by using user credentials and routing keys. Therefore, each client has their secure queues to work with, and that way, security is maintained in RabbitMQ.




If you have different types of data that you would like to separate, multiple queues can be made.



## Provided Data Format

The data that is sent from the queue(s) to the consumer can either be serialized to **JSON** or **Protocol Buffer** format.

Cypress Solutions will provide you with the following information for the consumer application to request data from the queue:

- RabbitMQ server URL
- Port Number
- Virtual Hostname
- Credentials for user setup in RabbitMQ virtual server
- Queue Name

**Note:** For consuming data on the consumer end, various ways can be used to fetch data from the queue. For example, you can use python as shown in **this** tutorial.

## Example: Python Source Code

This is a simple consumer that will dump out the messages read from the queue. The consumer application consists of:

- Connect to the RabbitMQ service (using the provided credentials)
- Create a callback that will handle incoming data
- Start consuming messages

When run, this application will connect and print out messages as they are published to the queue.

Please refer to the below screenshot.

```python
import pika, sys, os

def main():
    credentials = pika.PlainCredentials(username, password)

    connection = pika.BlockingConnection(
        pika.ConnectionParameters(rabbit_host, 5672, virtual_host_name, credentials))
    channel = connection.channel()

    def callback(ch, method, properties, body):
        print(" [x] Received %r" % body)

    channel.basic_consume(queue=queue_name, on_message_callback=callback, auto_ack=True)
    print(' [*] Waiting for messages. To exit press CTRL+C')

    channel.start_consuming()


if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
```

**Example of JSON data message:**


```
{
  "imei": "00000000000000",
  "userID": 00,
  "utc": 1637174125000,
  "valid": true,
  "heading": 181,
  "speed": 8.642666668625672,
  "lng": 90,
  "lat": 40,
  "alt": 304,
  "fields": {
    "OdometerAcc": 11111.006,
    "Speed": 8.642666668625672,
    "Total Engine Hours": 0,
    "Total Fuel Used": 0,
    "Total Idle Time": 0,
    "Battery Voltage": 28.323926,
    "SYS_BATT_STATUS": "",
    "SYS_FW_VER": "",
    "VIN": "",
    "Ignition": true,
    "IDLE_STATE": false,
    "Idling": false,
    "COMMON_BEACON_LIGHT": false,
    "GRADER_FRONT_BLADE_DOWN": false,
    "GRADER_MOLDBOARD_LEFT": false,
    "Input1": false,
    "Input2": false,
    "Input3": false,
    "Input4": false,
    "GRADING": false
  }
}
```


## Technical Support

**Cypress Solutions Service Support Group**

1.877.985.2878 or 1.604.294.4465

9.00am to 5.00pm PST

support@cypress.bc.ca