Ensuring data integrity with asynchronous programming in a cloud IoT core

Europython 2020

George Zisopoulos

Python Enthusiast, Angular addicted. Currently working as a Full-Stack Engineer at Veturilo.io

Theofanis Petkos

Python Fanatic, Elixir and Ruby fan. Also working as Software Engineer at Veturilo.io

#fleet_management #IoT
#embeded #async #programming

Our Story: Forrest and Lieutenant Dan

Backstage: Two friends working in the same start-up!

Our Story: The fellowship of the core

Backstage: Two friends working in the same start-up!

Mission: Create a fully-operational IoT Core working on fleet management.

IoT (Internet of things): ***A network of Internet connected objects, able to collect and exchange data.***

Requirements’ menace

Requirement 1: Send data packets from device/sensor to a server.

Requirements' menace

Requirement 1: Send data packets from device/sensor to a server.

Component 1: Devices (*OBDII for our use case*) which get signals from vehicles and sends data packets to a server. ***Plenty of devices around the web.***

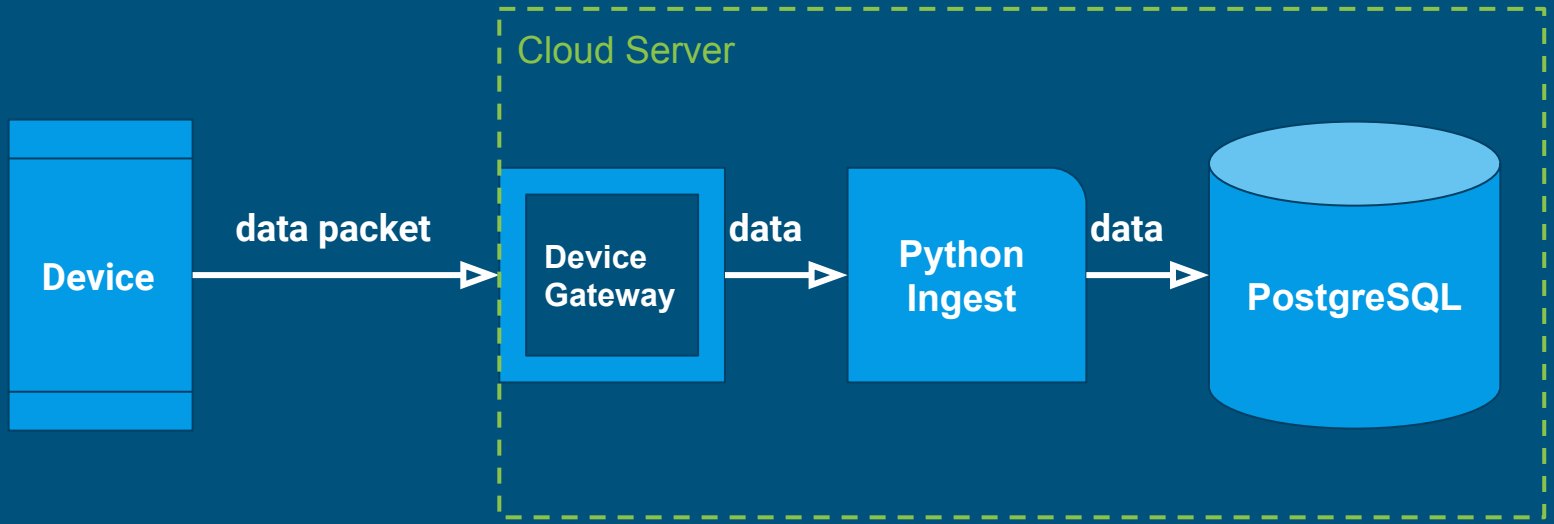
Component 2: An IoT server (*IoT core*) able to save incoming data and provide it to applications. ***Cheap and reliable solutions - cloud servers.***

From theory to Python

Requirement 2: Implement some services, inside IoT core, which will save all incoming data/signals to database.

From theory to Python

Requirement 2: Implement some services, inside IoT core, which will save all incoming data/signals to database.



The dark side of data

Are you sure that your incoming data packets were stored properly and in the desired format?

Scope of Data Integrity

Two basic principles:

1. Correct and not unintended storage
2. Ensure data quality

Two additional principles:

1. Services Integrity
2. Devices Integrity

The dark side of data

Are you sure that your incoming data packets were stored properly and in the desired format?

The dark side of data

Are you sure that your incoming data packets were stored properly and in the desired format?

Idea: Upon failure, use filesystem and temporarily save all signals into files. Then, retry to save all signals to database.

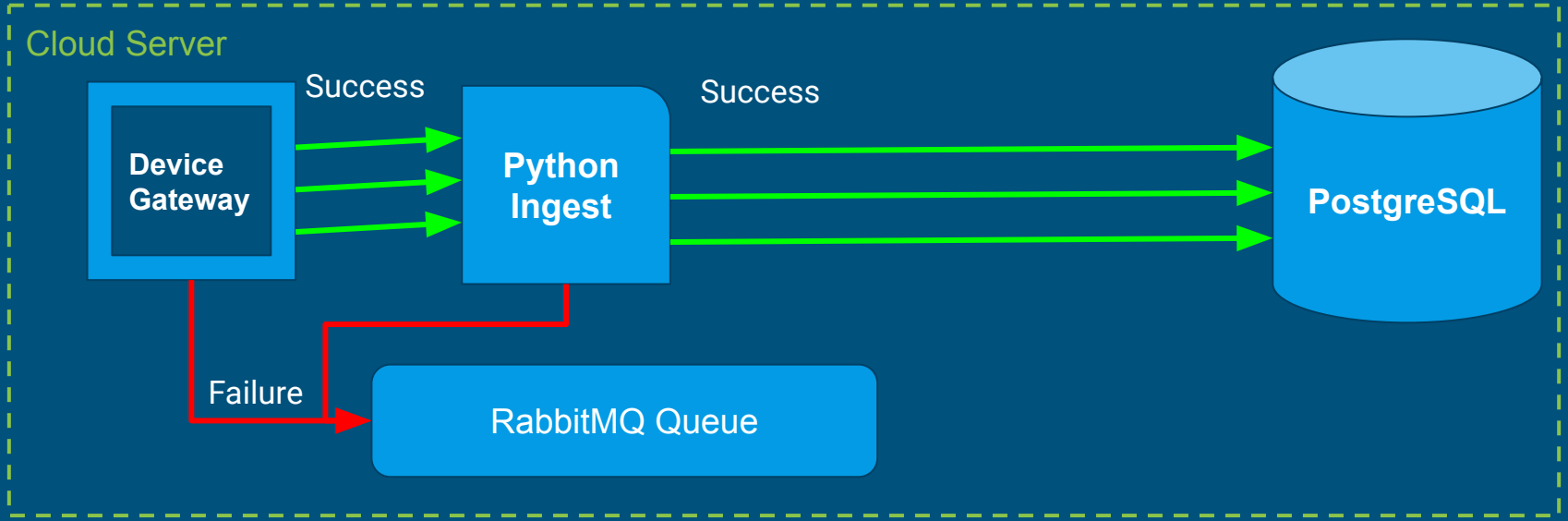
```
while True:
    for filename in os.listdir('/dir/path'):
        with open('/dir/path/' + filename) as f:
            content = f.readlines()
            content = [path.strip('\n') for x in content]
            reader = csv.reader(content)
```

The greatest teacher, failure is.

Master Yoda, The Last Jedi

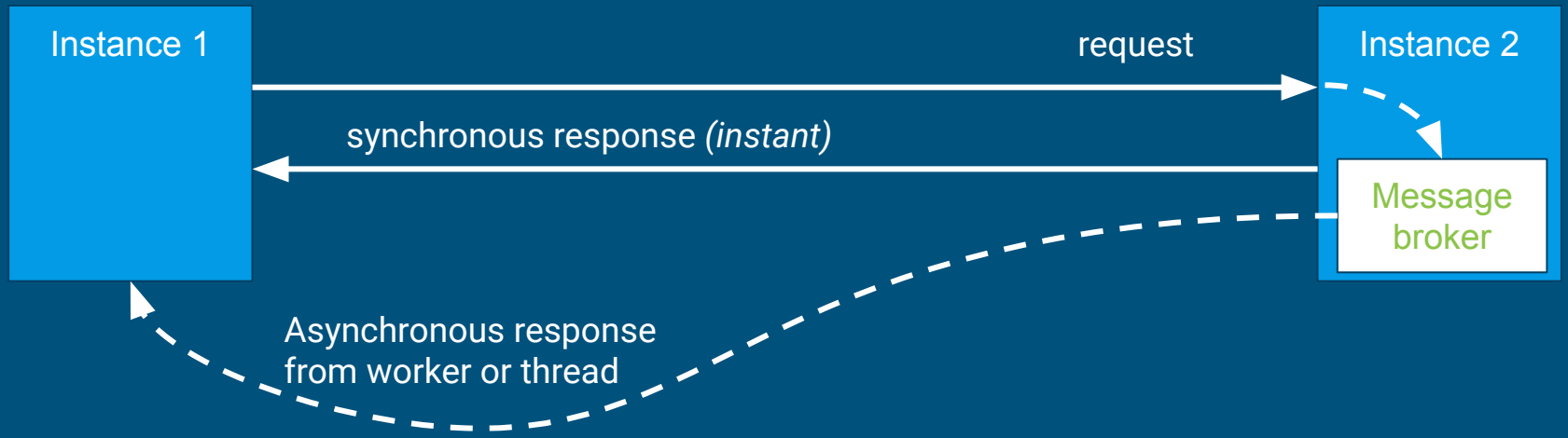
Asynchronous, concurrent days

Ingest Module/Device Gateway: Connected with RabbitMQ with a publisher. If something goes wrong publish signal to *queue*.



Asynchronous ways of Python

Asynchronous: The occurrence of events independent of the main program flow.



Concurrent ways of Python

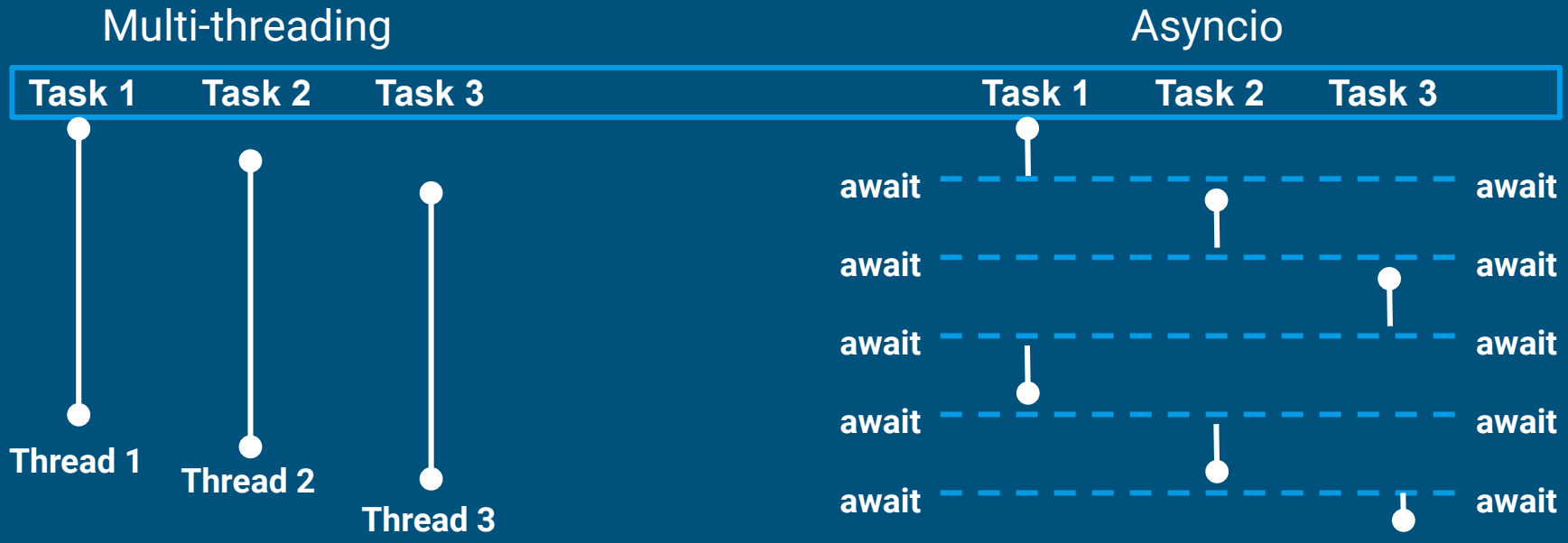
Concurrency: executing multiple tasks at the same time but not necessarily simultaneously *(like example 2)*.

```
# NO CONCURRENCY
# First task
[2020-07-09 14:21:56,030] Received from:('127.0.0.1', 39580)
[2020-07-09 14:21:56,066] Event ('127.0.0.1', 39580) Pushed Successfully to PostgreSQL
# Second task
[2020-07-09 14:21:56,067] Received from:('127.0.0.1', 39584)
[2020-07-09 14:21:56,109] Event ('127.0.0.1', 39584) Pushed Successfully to PostgreSQL

# CONCURRENCY
# First task starts
[2020-07-09 14:21:56,030] Received from:('127.0.0.1', 39580)
# Second task starts
[2020-07-09 14:21:56,031] Received from:('127.0.0.1', 39584)
# First task ends
[2020-07-09 14:21:56,066] Event ('127.0.0.1', 39580) Pushed Successfully to PostgreSQL
# Second task ends
[2020-07-09 14:21:56,083] Event ('127.0.0.1', 39584) Pushed Successfully to PostgreSQL
```

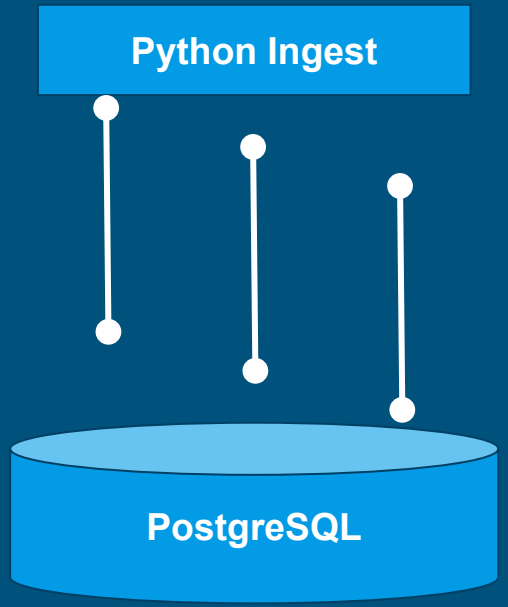
Multi-ways of Python

How to achieve concurrency: Multi-threading vs Asyncio.
Thread: The smallest instance that can be managed independently.



Multi Threading on Ingest

How to achieve concurrency: Multi-threading is important to support concurrency and performance into our Ingesting part.



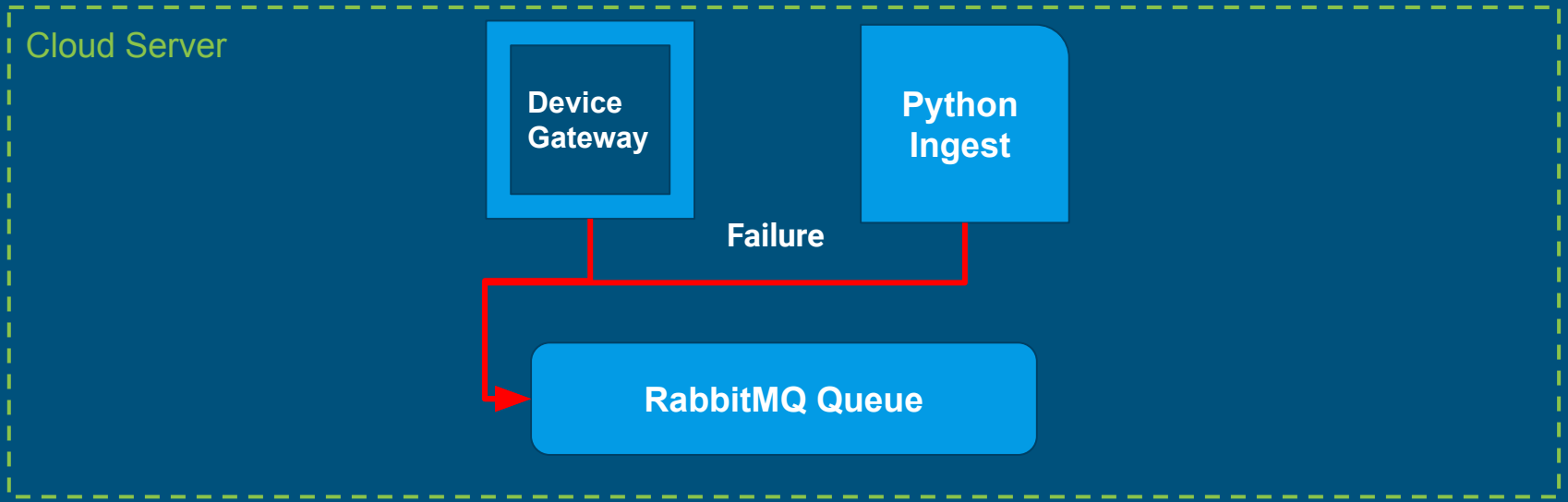

```
# Start a thread pool executor with specific number of workers
# in order to avoid high amount of threads
with cf.ThreadPoolExecutor(max_workers=3) as ingest_executor:
    # signals come to ingest with sockets
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as gw_socket:
        gw_socket.bind((host, port))
        # Wait to port until a a new signal comes
        gw_socket.listen()
        while True:
            # Accept a new signal and save it to db with a new thread.
            connection, address = gw_socket.accept()
            ingest_executor.submit(save_signal_to_db_method, connection, address)
```

Small recap

- **Python Ingest:** Small module which accepts incoming data and parses it to database.
- **Threads - Thread Pool Executor:** Multi-threading is used to our python ingest in order to achieve better performance.
- **Device Gateway:** A module which receives data packets from devices and forwards them as signals to ingest.

RabbitMQ as message broker

RabbitMQ: It gives your applications/modules a common platform to send and receive messages, and your messages a safe place to live until received.



Publisher

Producer: Able to connect with RabbitMQ and publish a message to a specific queue or exchange.

- Connects with RabbitMQ
- Gets connection.channel()
- If not exists, declares queue
- Publishes message

```
import pika

def publish(self, signal):
    """
    We skipped try-except blocks in order to have a very simple code
    """
    # Create connection with pika. Parameters are credentials.
    connection = pika.BlockingConnection(parameters)
    # Get a connection channel.
    channel = connection.channel()
    # Declare a new queue. If it's durable it will be there after a restart.
    channel.queue_declare(queue='queue', durable=True)
    # Publish message to rabbitmq
    properties=pika.BasicProperties(delivery_mode=2)
    channel.basic_publish(exchange='', routing_key='queue' body=signal,
                          properties=properties)
```

Consumer

Consumer: Able to receive/consume all messages inside this queue or exchange. With aioamqp can share thread with other tasks while waiting.

- Connects with RabbitMQ
- Gets connection.channel()
- Awaits for a signal
- Pushes it back to Ingest

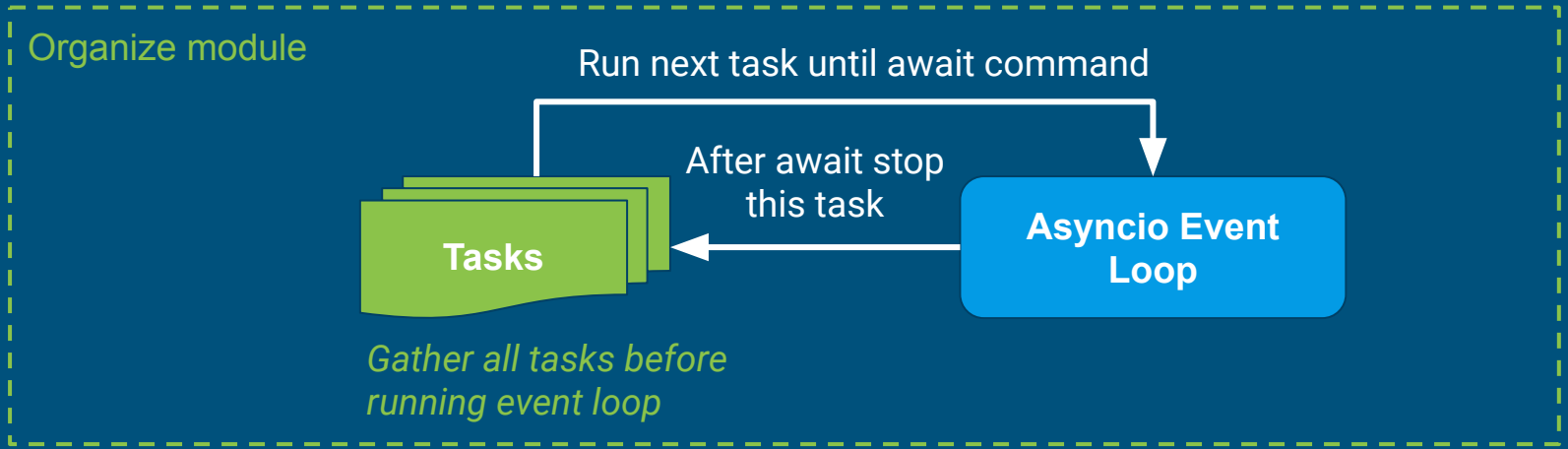
```
async def consume(**kwargs):
    """
    Consumer written with aioamqp in order to work with asyncio.
    """
    transport, connection = await aioamqp.connect(
            host=host, port=port, login=username,
            password=password, login_method='PLAIN')
    # some possible exceptions here
    # except aioamqp.AmqpClosedConnection
    # except ConnectionRefusedError

    # create a channel again in order to receive messages
    channel = await connection.channel()
    # Await for a new signal from queue
    await channel.basic_consume(callback, queue_name='events')
```

Organize module

Duty: Schedule quality/service checks, push back every failed signal. Built with asyncio.

Asyncio: Useful tool which support cooperative multitasking. It gives you the advantage of concurrency inside a single thread.



Organize module

Initialize event loop: Create the event loop, gather all tasks and run it.

Create event loop

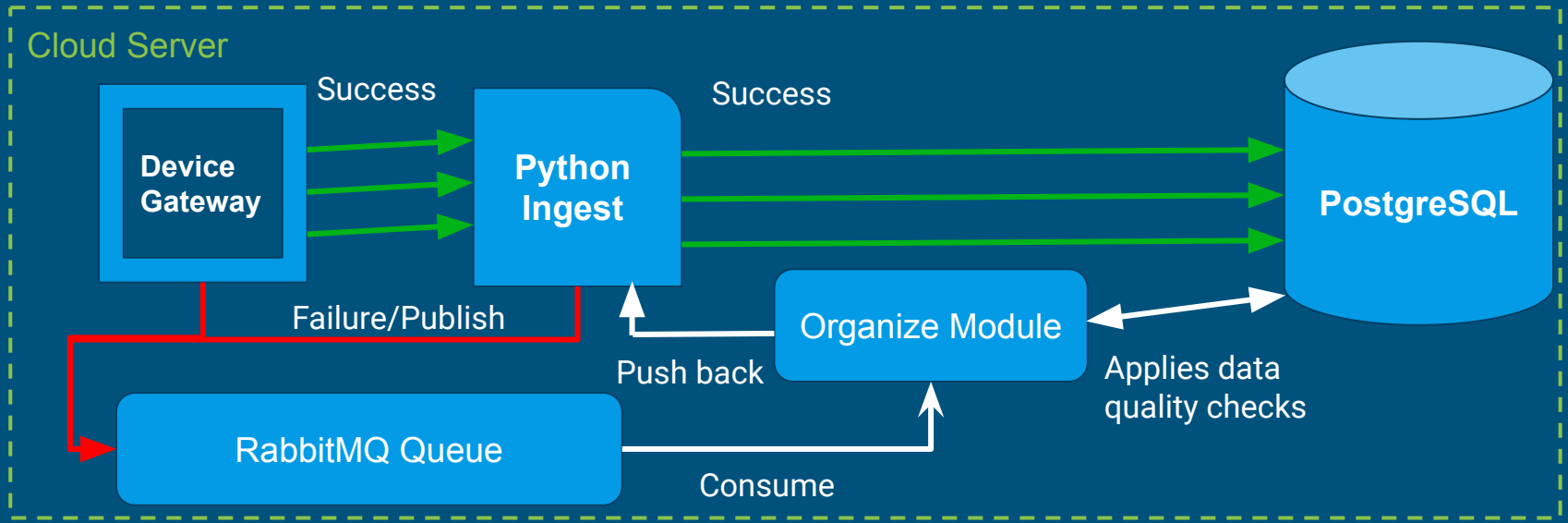
Gather your tasks

Run your event loop

```
# Create new loop
event_loop = asyncio.new_event_loop()
# Set new loop to asyncio
asyncio.set_event_loop(event_loop)
# Gather all tasks
event_loop_tasks = asyncio.gather(
    consumer(),
    periodic_task_1(timeout),
    periodic_task_2(timeout))
 Try:
   # Run the loop
    event_loop.run_forever()
 except KeyboardInterrupt:
    event_loop_tasks.cancel()
```

The rise of asyncio

After the implementation of previous module the flow of our IoT Core would be like this:



Clockwork organizer

Idea 1:

Periodic quality check - Data Quality
Example case - broken gps

Catch 2 - Devices Integrity.
After some errors for the same device, notify for device check.

```
# Fetch a random set of signals from database
# and check if lan,lot values are in correct range
async def periodic_quality_check(timeout):
    while True:
        for signal in list_of_random_signals:
            if wrong_coordinates(signal.longitude, signal.latitude):
            # TODO - Notify user for broken gps.
        # Gives up execution, waits to run after timeout.
        await asyncio.sleep(timeout)


def wrong_coordinates(longitude, latitude):
    """
    Check if longitude and latitude are between correct range
    """
    if (longitude > 90 or longitude < -90 or
       longitude > 180 or longitude < -180):
        return True
    else:
        return False
```

Clockwork organizer

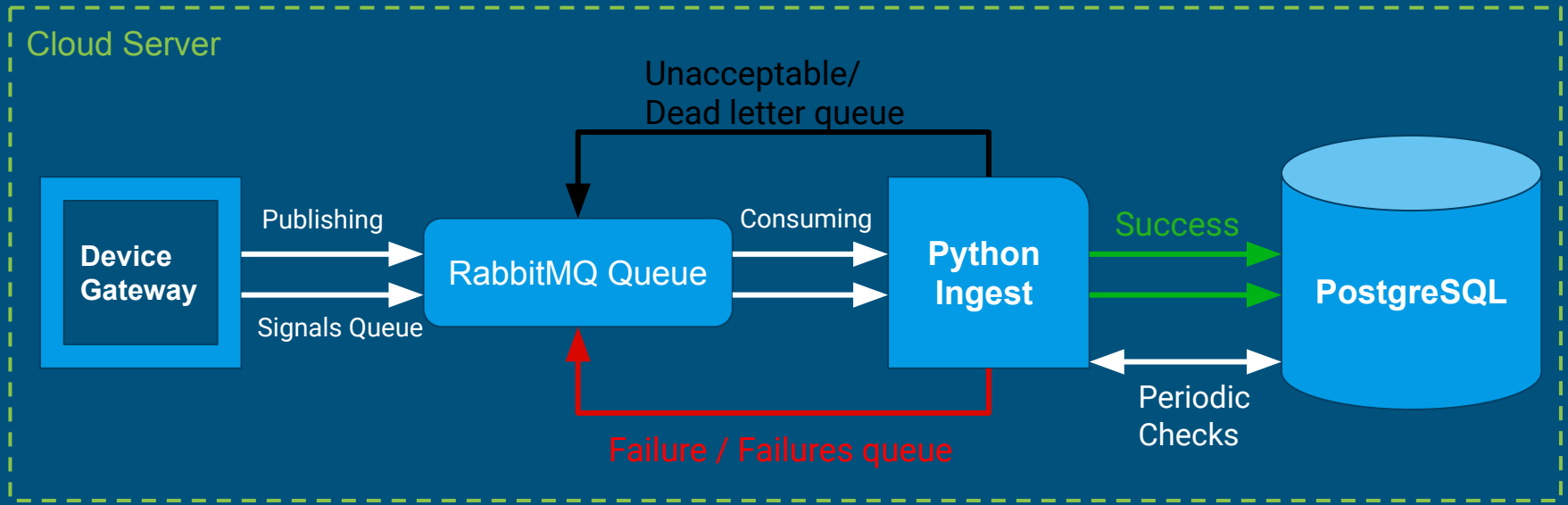
Idea 2:

Periodic check for services' heartbeat.
Pretty simple, though not completed, way to check for services integrity.

```
async def check_heartbeat(timeout):
    “””
    Checks if services are up.
    “””
    services_list = [(‘127.0.0.1’,2006),(‘127.0.0.1’,5432)]]
    while True:
        for address, port in services_list:
            # simplest way to bind with socket to port
            # in order to check if service is up
            running = bind_to_service(address, port)
            if not running:
                # Notify admin that service is down.
            else:
                # Log that everything into IoT core is ok.
        # give up execution for timeout
        await asyncio.sleep(timeout)
```

Reinventing the wheel

Another step forward: Merge everything into python ingest. Make message broker the actual middleman between gateway and ingest.



The artilleryman’s song

Call artillery for help: You could combine this logic with celery or other task queue software.

Before do so: *Code it, break it, smash it and practice!* ***First you have to understand!***

Our Story: Endgame

We would like to thank:

- **Andreas, George and Harry** *(the rest of our **veturilo.io** tech team)*, for their unlimited support.
- **Panos, George and Thanasis** *(from **stackmasters.eu**)*, for their important help and ideas.
- **Bill** (*from **starttech.eu**)*, for his efforts to organize this presentation.
- **The rest of *starttech.eu* wonderful community** for their daily support to our work.

Our Story: Drop us a line!

Github Repo:

1. https://github.com/gzisopoulos/python-iot-data-integrity
2. https://github.com/thepetk/python-ingest

Discord channel:

#talk-data-integrity-with-async