CroMFlaG2

An Open Source Software Stack for Building IoT Solutions

March 27, 2020
Revision 1.00

Table Of Contents

Executive Summary

This whitepaper showcases the hows and whys of a simple, flexible open source Internet of Things (IoT) software stack: CroMFlaG2 (CROn, Mqtt, FLAsk, GridDB, Grafana). This stack gives developers multiple options for ingesting, analyzing, and visualizing data required to build useful IoT applications.

Introduction

With the popularity of the LAMP[1] (Linux, Apache, MySQL, PHP/Perl/Python) stack that powered the commoditization of the internet in the 2000s, many vendors have pushed their products into neat, tightly coupled stacks of software that ease usage and deployment.

Some of these stacks have some adverse side effects: they promote vendor lock-in and force the solution to fit the model that is intended by the stack making improvements, changes, or new features difficult. CroMFlaG2 is intended to behave like the original LAMP stack by being open source, extensively configurable and user modifiable.

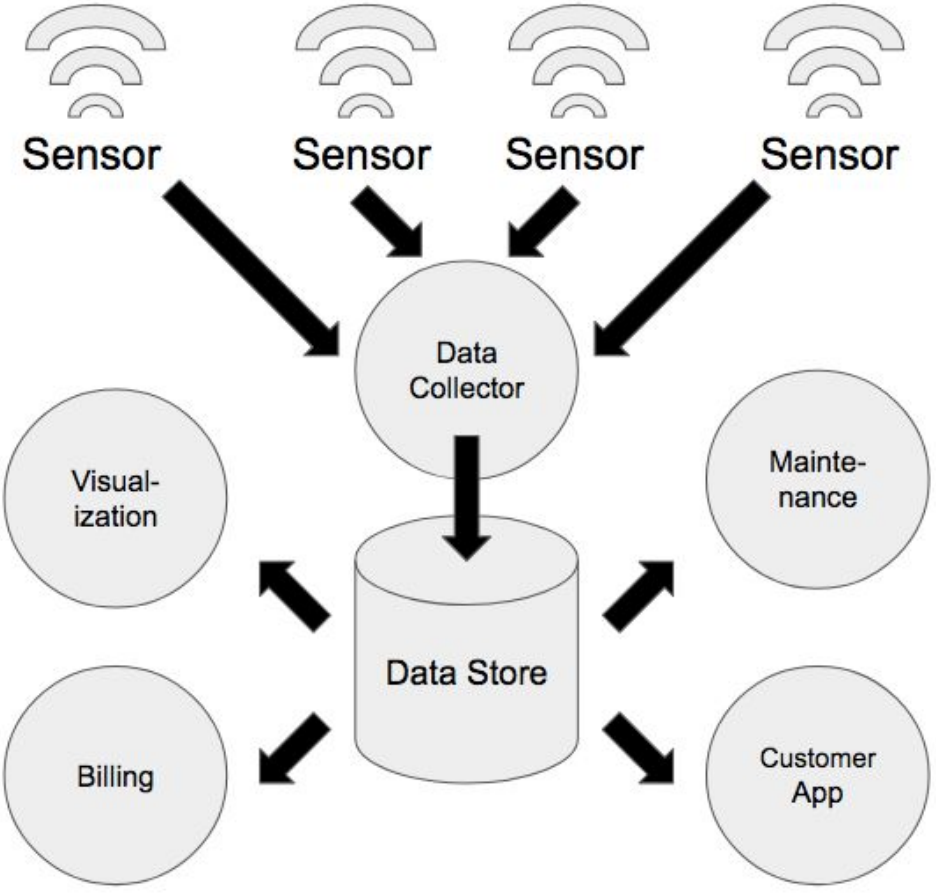
[1] https://en.wikipedia.org/wiki/LAMP_(software_bundle)

CroMFlaG2 has also been proven to be stable, lightweight and scalable to hundreds of thousand devices and is easy to implement traditionally or with new practices such as DevOps and Containerization; its components are well proven to work well with each other. They consist of Cron, Mqtt, Flask, GridDB, and Grafana.

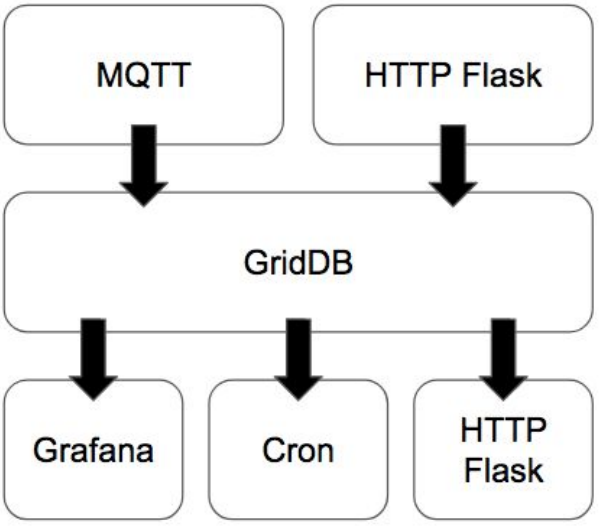
IoT Architecture

Most IoT solutions consist of edge devices that contain sensors or other data generators and communicate with centralized infrastructure either directly or through a local gateway.

The centralized infrastructure can be hosted either on a public cloud or private on-premises servers. The devices usually communicate with a separate data collector and not directly with a database but there are certain instances where it is not required.



Once the data is stored in the primary data store, other processes can perform analysis or visualize the stored data either in real time or in batches.

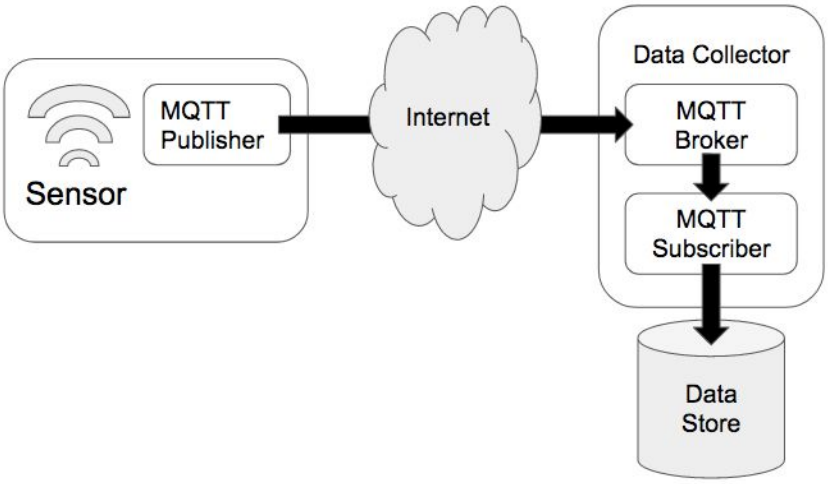


In CroMFlaG2, the MQTT Broker and Subscriber or HTTP Flask Application act as the data collector while GridDB is the data store. HTTP Flask can also operate as an endpoint to provide real time data to untrusted applications, for example, those developed or executed by third parties outside of the data hosting organization. Meanwhile Grafana provides a platform to visualize the data and Cron is used to generate reports, invoices, etc. on predefined intervals.

Sending and Collecting Data

MQTT

MQTT is a lightweight publish/subscribe protocol that is intended for machine to machine (M2M) communication. The key advantages of MQTT are its persistent connection, low overhead (2 bytes) and its three QoS levels that ensure either that a message is sent, a message is received, or a message is received at exactly once.



The MQTT broker is a service that runs on the data collector and passes data between the MQTT publisher that is executed within the sensor or edge device and the MQTT Subscriber that reads the publisher's data and writes it into the data store. Popular brokers include Mosquitto[2], HiveMQ[3], and RabbitMQ[4], any of which can be used within CroMFlaG2.

Data can be sent in any format but is usually binary blobs or more universal formats such as MessagePack, JSON, or XML. MessagePack is a very lightweight, binary format that can be easily serialized through data type conversion or custom packing. JSON and XML are both used extensively within industry with excellent libraries available for generation and parsing.

HTTP/Flask

HTTP is the most commonly used network protocol, powering nearly all web traffic. HTTP has more network overhead than MQTT and does not feature QoS features but has some other advantages. First of all, HTTP is rarely blocked by network firewalls, it is simple to test and well understood by many developers, features many libraries and frameworks to build solutions with it, and is more reliable if wanting to transmit data from the centralized infrastructure to the edge device.

On the edge or device side, libcurl[5] or more abstracted libraries such as Microsoft's C++ Rest SDK[6], can be used to send data. There are also a variety of ways to process incoming data on the data collector, one of which is Flask, a lightweight web application framework.

The following sample code shows how curl can be used to simulate a device inserting a record, simple scripts allow for easy DevOps deployment and Continuous Integration practices.

---

[2] https://mosquitto.org/
[3] https://www.hivemq.com/
[4] https://www.rabbitmq.com/
[5] https://curl.haxx.se/libcurl/
[6] https://github.com/microsoft/cpprestsdk

```
$ curl -X POST --header "Content-Type: application/json" \
  http://localhost:8000/insert/sample --data '{
    "deviceinfo" : { "deviceid": "sample", "fw_ver" : "12345.009", "batt_lvl": 78 },
    "tsdata": [
            { "day": 18, "dayofweek": 2, "hour": 17, "humidity": 75.0,
              "illuminance": 74.0, "month": 2, "motion": false, "temperature": 78.0,
              "timestamp": 1582046738255 },
            { "day": 18, "dayofweek": 2, "hour": 17, "humidity": 66.0,
              "illuminance": 96.0, "month": 2, "motion": false, "temperature": 82.0,
              "timestamp": 1582046748255 },
            { "day": 18, "dayofweek": 2, "hour": 17, "humidity": 67.0,
              "illuminance": 93.0, "month": 2, "motion": false, "temperature": 81.0,
              "timestamp": 1582046758255 },
            { "day": 18, "dayofweek": 2, "hour": 17, "humidity": 77.0,
              "illuminance": 95.0, "month": 2, "motion": true , "temperature": 80.0,
              "timestamp": 1582046768255 },
            { "day": 18, "dayofweek": 2, "hour": 17, "humidity": 70.0,
               "illuminance": 90.0, "month": 2, "motion": false, "temperature": 79.0,
               "timestamp": 1582046778255 },
            { "day": 18, "dayofweek": 2, "hour": 17, "humidity": 66.0,
               "illuminance": 86.0, "month": 2, "motion": false, "temperature": 77.0,
               "timestamp": 1582046788255 }
    ]
}'
```

An example app built with the Flask[7] framework that enables recording of data to GridDB follows:

```
#!/usr/bin/python3 -u

from flask import Flask, request, abort
from flask_cors import CORS, cross_origin
from datetime import datetime
import griddb_python
import json

griddb = griddb_python
factory = griddb.StoreFactory.get_instance()
app = Flask(__name__)
cors = CORS(app)

@app.route('/insert/<device>', methods=['POST'])
def post(device):

    try:
        data = json.loads(request.data)

        if data['deviceinfo']['deviceid'] != device:
            abort(500, "malformed request")
```

[7] https://palletsprojects.com/p/flask/

```
        conInfo = griddb.ContainerInfo("devices",
                [["deviceid", griddb.Type.STRING],
                ["batt_lvl", griddb.Type.INTEGER],
                ["fw_ver", griddb.Type.STRING]],
                griddb.ContainerType.COLLECTION, True)

        devConInfo = griddb.ContainerInfo(device,
                [["timestamp", griddb.Type.TIMESTAMP],
                ["motion", griddb.Type.BOOL],
                ["temperature", griddb.Type.FLOAT],
                ["humidity", griddb.Type.FLOAT],
                ["illuminance", griddb.Type.FLOAT],
                ["month", griddb.Type.LONG],
                ["day", griddb.Type.LONG],
                ["dayofweek", griddb.Type.LONG],
                ["hour", griddb.Type.LONG]],
                griddb.ContainerType.TIME_SERIES, True)

        col = gridstore.put_container(conInfo)
        devCol = gridstore.put_container(devConInfo)
        col.set_auto_commit(False)
        devCol.set_auto_commit(False)

        col.put([data['deviceinfo']['deviceid'], data['deviceinfo']['batt_lvl'],
                 data['deviceinfo']['fw_ver']])

        tsdata = []
        for row in data['tsdata']:
            tsdata.append([datetime.fromtimestamp(row['timestamp']/1000), row['motion'],
                           row['temperature'], row['humidity'], row['illuminance'],
                           row['month'], row['day'], row['dayofweek'], row['hour']])
        devCol.multi_put(tsdata)
        col.commit()
        devCol.commit()

    except:
        abort(500, "Insert failed")

    return "True"

if __name__ == "__main__":
    gridstore = factory.get_store(
        host="239.0.0.1",
        port=31999,
        cluster_name="defaultCluster",
        username="admin",
        password="admin"
    )

    app.run(host='0.0.0.0', port=8000 )
```

The device sends a HTTP POST request to the endpoint which parses the JSON and inserts the data into multiple GridDB containers.

While not included in the above code, one big advantage of using a web framework such as Flask is the ease of authenticating devices; you can include libraries such as OAuth2 that provide a proven framework for authentication.

Storing Data

GridDB

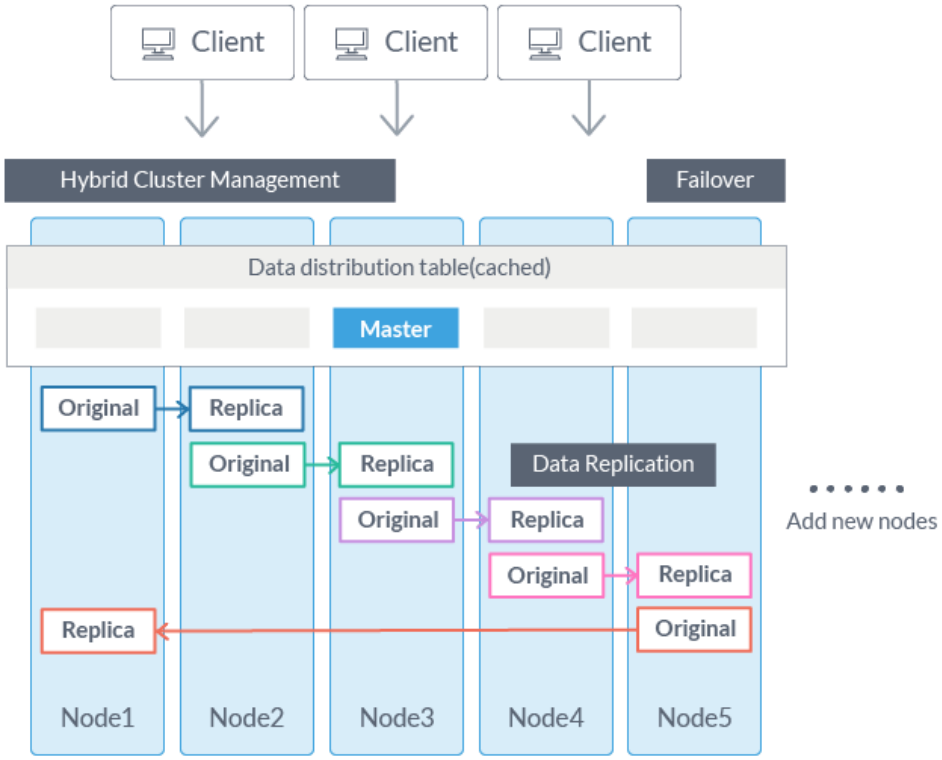
GridDB is a highly scalable, in-memory NoSQL time series database optimized for IoT and Big Data developed by Toshiba Digital Solutions Corporation.

The Key-Container data model of GridDB extends the typical NoSQL Key-Value store. The Key-Container model represents data in the form of collections that are referenced by keys. The key and container are rough equivalents of the table name and table data in Relational Databases (RDB). Data modeling in GridDB is easier than with other NoSQL databases as we can define the schema and design the data similar to that of an RDB.

GridDB's hybrid composition of In-Memory and Disk architecture is designed for maximum performance as I/O is a common bottleneck in any DBMS that can cause the CPU to be under-utilized. GridDB overcomes this bottleneck with the 'Memory first, Storage second' structure where the 'primary' data that is frequently accessed resides in memory and the rest is passed on to disks (SSD and HDD).

GridDB scales linearly and horizontally on commodity hardware maintaining excellent performance as shown in YCSB benchmarks against Cassandra.[8] Traditional RDBMS are built on Scale-Up architecture (add more capacity to existing server/node). Transactions and data consistency are excellent on RDBMS. On the other hand, NoSQL databases focus on Scale-Out architecture but fare poorly on transactions and data consistency.

[8] https://griddb.net/en/blog/griddb-and-cassandra-ycsb-benchmarks/



Hybrid cluster management and high fault-tolerant system of GridDB is exceptional for mission-critical applications. Network partitions, node failures, and maintaining consistency are some of the major problems that arise when data is distributed across nodes. Typically, distributed systems adopt ‘Master-Slave’ or ‘Peer-to-Peer’ architectures. Master-Slave option is good at maintaining data consistency but a master node redundancy is required to avoid having a Single Point of Failure (SPOF). Peer-to-Peer, though avoids SPOF, has a huge problem of communication overhead among the nodes.

Processing and Using Data

HTTP/Flask

Like using HTTP and Flask for ingesting data, HTTP and Flask are ideal for delivering data to other applications, especially those used by third parties. The benefits of using HTTP and Flask is the ease of development, the flexibility HTTP provides, and the number of different libraries and frameworks available to build the application.

```
@app.route('/fetch/<device>', methods=['GET'])
def fetch(device):
```

```
    try:
        ts = gridstore.get_container(device)

        tql = "select *"
        first = True;

        if device != "devices":
            for arg in request.args:
                if first:
                    tql = tql + " WHERE"
                else:
                    tql = tql + " AND"
                if arg == "from":
                    tql = tql + " timestamp >= TO_TIMESTAMP_MS("+request.args['from']+") "
                if arg == "to":
                    tql = tql + " timestamp < TO_TIMESTAMP_MS("+request.args['to']+") "
                first = False
        query = ts.query(tql)
        rs = query.fetch(False)
        columns = rs.get_column_names()
        datadict = {}

        retval=[]
        while rs.has_next():
            data = rs.next()
            for col in columns:
                if col == "timestamp":
                    datadict[col] = int(data[columns.index(col)].timestamp()*1000)
                else:
                    datadict[col] = data[columns.index(col)]
            retval.append(datadict.copy())

        return json.dumps(retval)
    except:
        abort(500, "Fetch failed")
```

The above is a simple endpoint that allows querying of time series data optionally between timestamps. Further features are easy to add as required with the numerous data analysis tools available in python and useful additions to the Flask webframe like OAuth2 authentication to ensure data privacy or WebSockets to build real-time streaming applications.

Grafana

Grafana[9] is an open source analytics and interactive visualization platform that can read data from many different databases including GridDB. There are two options for using Grafana with GridDB, the first using a custom endpoint with Flask and then using JSON Datasource or using the GridDB WebAPI and the native Grafana GridDB Datasource.

[9] https://grafana.com/

With Grafana, not only can you visualize real-time time series data in a variety of chart formats, but it is also possible to add database driven annotations or display aggregations in a table.

It is the ideal tool for operations teams to monitor their equipment with little upfront setup.

Cron

Cron[10] is one of the oldest utilities used to manage software and should not be overlooked for its simplicity and effectiveness. With Cron, the desired application that performs any task can be run hourly, weekly, or monthly. In terms of the CroMFlaG2 stack, these applications usually generate daily or weekly reports or monthly invoices based on data that has been ingested.

Cron is not only included in most Linux distributions, but it is also possible to deploy applications with Kubernetes container orchestration[11].

Conclusion

With CroMFlaG2 developers can build IoT solutions with open source components that adapt to their infrastructure rather than adapting their infrastructure to the IoT software stack. With MQTT and HTTP Flask, both high velocity data streams and recurring batched data can

[10] https://en.wikipedia.org/wiki/Cron
[11] https://kubernetes.io/docs/tasks/job/automated-tasks-with-cron-jobs/

be ingested into the GridDB data store while HTTP Flask, Cron, and Grafana provide different methods to furnish, process and visualize IoT data.

The HTTP Flask samples included in this whitepaper are available at https://griddb.net/en/resources/ and as a Docker image at https://hub.docker.com/r/griddbnet/cromflag2.