EPJ Web of Conferences **295**, 01055 (2024) https://doi.org/10.1051/epjconf/202429501055
CHEP 2023

Using MQTT and Node-RED to monitor the ATLAS Metadata Interface (AMI) stack and define metadata aggregation tasks in a pipelined way

Jérôme Odier[1,*], *Fabian* Lambert[1], *Jérôme* Fulachier[1], *Maxime* Jaume[1], and *Pierre-Antoine* Delsart[1]

[1]Centre National de la Recherche Scientifique (CNRS) / Laboratoire de Physique Subatomique et de Cosmologie (LPSC), Grenoble FRANCE

Abstract. ATLAS Metadata Interface (AMI) is a generic ecosystem for metadata aggregation, transformation and cataloging. Each sub-system of the stack has recently been improved in order to acquire messaging/telemetry capabilities. This paper describes the whole stack monitoring with the Message Queuing Telemetry Transport (MQTT) protocol and Node-RED, a tool for wiring together hardware/software devices. Finally, this paper shows how Node-RED is used to graphically define metadata aggregation tasks, in a pipelined way, without introducing any single point of failure.

1 Introduction

ATLAS Metadata Interface[1][2][3] (AMI) is a generic ecosystem for metadata aggregation, transformation and cataloging. Benefiting from more than 20 years of feedback in the LHC context, the second major version was released in 2018.

AMI provides a wide array of tools (command line tools, lightweight clients) and Web interfaces for searching scientific data by metadata criteria. It was designed to guarantee scalability, flexibility and maintainability. It perfectly fits the needs of scientific experiments in a big data context.

AMI is currently used by scientific collaborations such as ATLAS[4], NIKA2[5] and n2EDM[6].

We recently added, for each subsystem of the ecosystem, telemetry capabilities via the Message Queuing Telemetry Transport[8][9] (MQTT) protocol. This article describes how MQTT is used to improve the ecosystem monitoring. It also describes how MQTT is used to define and monitor the execution of pipeline of metadata aggregation tasks.

*e-mail: ami@lpsc.in2p3.fr

2 Overview of the AMI ecosystem

The AMI ecosystem consists in:

- AMI Core Lib: A high-level Java library to manage connectivity to heterogeneous data sources (SQL, noSQL, files, ...), to extract, manipulate and transform metadata, and to manage database persistence. AMI Core Lib implements the Metadata Query Language[10] (MQL) making it possible to perform queries without knowledge of the underlying database structure.
- AMI Core Web : Jakarta EE[11] Web services based on the HTTPS protocol and implementing a REST interface as well as, for historical reasons, a proprietary interface.
- AMI Task Server : A distributed server, a kind of "super cron[12]", to perform metadata aggregation, transformation and persistence tasks.
- AMI Exclusion Server : An optional zero-configuration server used to pilot the task exclusions in a cluster of AMI Task Servers.
- AMI Pipeline : A flow-based and low-code development tool for visual programming permitting pipelines of tasks to be defined graphically.
- AMI Web Framework : JavaScript-based development tool for developing metadata-driven Web applications, benefiting from the features of the AMI backend.

Each major subsystem has telemetry capabilities, via the MQTT protocol, and can connect to an MQTT V3.X or v5.0 broker (for instance Eclipse Mosquitto[13] or RabbitMQ[14]), see figure 1.

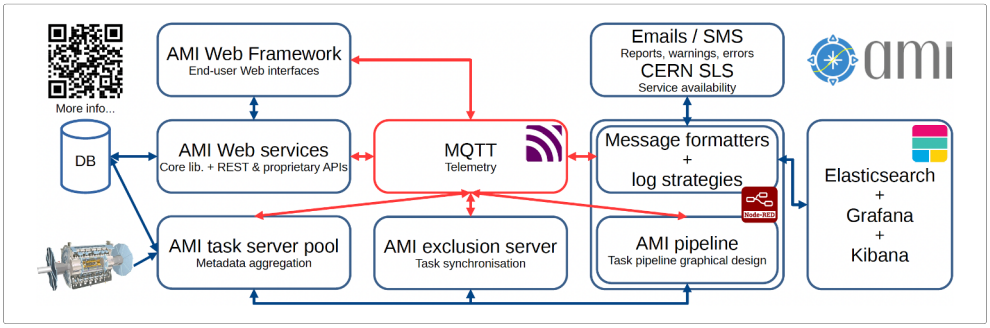

Figure 1. Overview of the AMI ecosystem stack. Each major subsystem has telemetry capabilities via the MQTT protocol.

This new telemetry architecture is still in the testing phase and is part of the process of redesigning the ATLAS experiment's metadata aggregation tasks.

3 Implementing MQTT telemetry in AMI

MQTT is a lightweight network protocol that facilitates machine-to-machine communication using a publish-subscribe model. It is especially useful for message queuing services. This protocol is designed for operating in remote locations that have devices with limited resources or network bandwidth, making it ideal for Internet of Things (IoT) applications.

MQTT requires a transport protocol that ensures ordered, lossless, and bi-directional connections, typically TCP/IP. It is recognized as an open OASIS[7] standard and also recommended by ISO (ISO/IEC 20922).

3.1 MQTT clients

Developed by the Eclipse foundation[15], the Paho[16] project provides MQTT client libraries for a variety of programming languages (Java, JavaScript, Python, C/C++) with support of asynchronous (non-blocking) communication via TCP or via WebSocket[17] connections. The libraries also have built-in automatic reconnect functionality to guarantee high service availability. Paho clients support MQTT versions 3.1, 3.1.1 (often referred to as MQTT 4), and 5.0.

The AMI ecosystem use the Java and JavaScript implementations of Eclipse Paho.

3.2 MQTT server

Server-side, AMI uses Eclipse Mosquitto[13] 2. This MQTT broker is lightweight and suitable for use on all devices from low power single board computers to full servers. Mosquitto can bridge to other MQTT servers, making it possible to create a network of MQTT servers for distributed architectures. It can persist messages to disk when they cannot be delivered immediately, allowing it to handle temporary network failures or subscriber downtime.

Mosquitto supports SSL-secured WebSocket connections. Unfortunately, unlike the AMI ecosystem, it does not natively offer JSON Web Token[18] (JWT) authentication, so a generic plugin[19] has been developed.

3.3 MQTT-based commands

The traditional way of communicating with the AMI Web services is to exchange messages (aka AMI commands) via a dedicated servelet. As an example, the AMI command to find out information about the ATLAS dataset "`data23_calib.00441648.calibration_LArElec-Ramp-7s-Low-Em.daq.RAW`" is :

```
GetDatasetInfo -logicalDatasetName="data23_calib.00441648.calib..."
```

To preserve this way of communicating with AMI via MQTT, a protocol overlay has been developed. It permits asynchronous and bidirectional exchanges of AMI commands between a server component (AMIMQTTServerJava project) and a client component (AMIMQTTClientJava or AMIMQTTClientJS[20] projects). It should be noted that the servers, like the clients, are themselves clients of the MQTT broker.

This command system allows direct interrogation of any subsystem of the AMI ecosystem to perform administration operations (reboot, check CPU and RAM usage, ...) without having to open dedicated TCP ports.

One example is the new interface for managing AMI Task Server clusters and metadata aggregation tasks (see figure 2).

3.4 MQTT message routing with Node-RED

Originally designed by IBM, Node-RED[21] is a flow-based, low-code development tool that permits visual programming. This tool was primarily created for the purpose of connecting hardware devices, APIs, and online services within the IoT landscape.

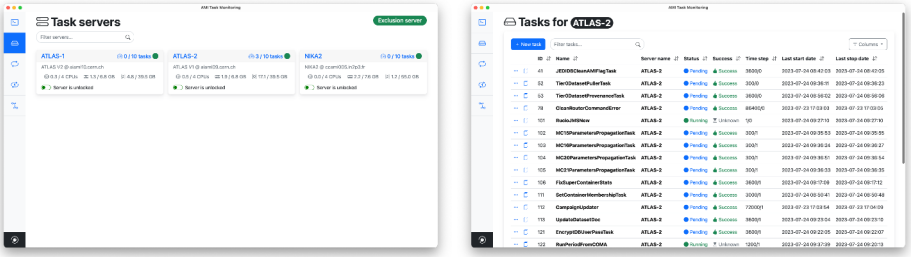

Figure 2. Application (Web and desktop) to administrate a cluster of AMI Task Servers. The list of task server nodes is shown at left, and the list of tasks for the "ATLAS-2" node is shown at right.

Node-RED is often used as a graphical editor offering an intuitive way to design and manage data flows, including routing MQTT messages and defining actions based on those messages. It permits the creation of complex workflows, by connecting and configuring nodes, in a way that even users with minimal coding skills can leverage.

In 2016, Node-RED was transitioned into an open-source project, now managed by the OpenJS Foundation, thanks to a contribution from IBM.

AMI offers a Node-RED plugin for executing AMI commands via the HTTPS protocol (to access the traditional metadata Web services) and via the MQTT protocol (to control the ecosystem's subsystems).

The critical logs are also broadcast to a dedicated MQTT topic. This makes Node-RED an effective tool for implementing strategies to deal with machine or service problems. For example, email or SMS alerts can be triggered.

Figure 3 shows how to graphically define two slots, one for sending emails and one for sending SMSs. If sending the SMS fails, the `"/opt/send_sms"` node triggers the sending of a fallback email.

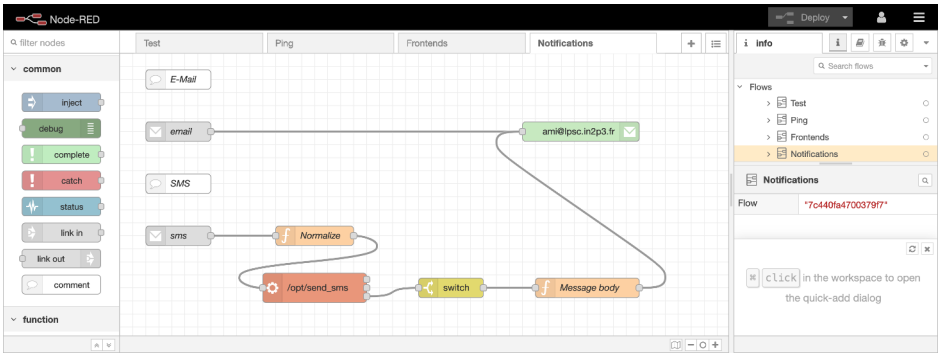

Figure 3. Definition of two slots for sending emails and SMSs with Node-RED.

4 Defining and monitoring MQTT-based task pipelines

In a move to modernize the ATLAS experiment's metadata aggregation system, it was decided to introduce a pipeline paradigm to divide monolithic tasks into smaller ones. Pipelines defined in this way must be able to be restarted on the fly after any failure.

The idea is to use Node-RED to define the pipeline topology and perform monitoring without introducing any single point of failure.

4.1 The AMI Task Server

The AMI Task Server is a kind of general-purpose distributed cron. It is able to execute any program or script but is optimized to use the Java Core Library. Its priority-based lottery scheduler guarantees starvation-free operations. It can launch recurring or pipelined tasks (triggered by recurring tasks) whose definition are stored in the database.

Each event - server startup, task startup / success / failure, etc. - gives rise to an MQTT task report message (in addition to standard logs) for fine-grained monitoring.

4.2 Task pipeline definition with Node-Red

Node-RED is a handy tool for graphically modeling flows. AMI provides a plugin for defining the topology of pipelines of tasks.

Node-RED is employed for the design, validation, and persistence of pipeline definitions within the AMI Task Servers database (as illustrated in Figures 4 and 5). However, it does not serve to trigger the execution of any tasks.

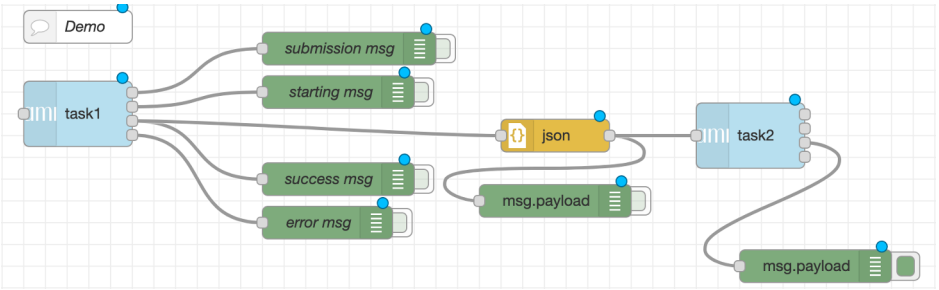

Figure 4. Minimal example of pipeline. In case of success, a task report document is propagated from *task1* to *task2*, causing the latter to start.

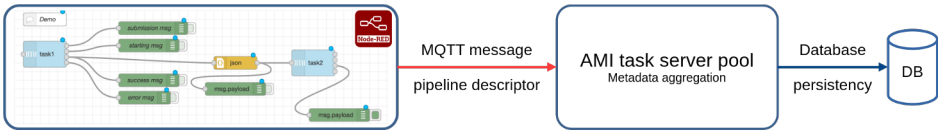

Figure 5. The AMI ecosystem provides a Node-RED plugin to graphically define metadata aggregation task pipelines. Each individual pipeline description is sent to the task server pool which validates and persists the description.

The task server cluster scheduling system is solely responsible for execution. Node-RED receives MQTT messages for monitoring purposes only. No single point of failure is introduced (see figure 6).

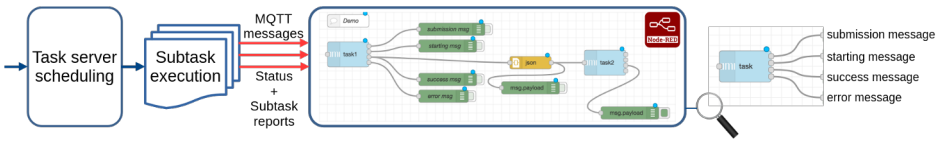

Figure 6. A task server executes the pipeline. At each stage, messages are sent to Node-RED for monitoring purposes. In case of issues, an error report is returned and the pipeline stops. Otherwise, a subtask report is propagated to the next subtask. The final status can be sent to the Elasticsearch[22] full-text search engine.

5 Conclusion

MQTT is a TCP/IP-based publish-subscribe messaging protocol that became very popular in the IoT landscape. It is particularly well suited for controlling and monitoring the various components of a software ecosystem.

MQTT has been integrated into each of the AMI subsystems. The Node-RED MQTT message router allows administrators to graphically set up issue management strategies, such as sending emails and SMSs, or creating new messages for an Elasticsearch stack.

Node-RED makes it possible to graphically define pipelines of metadata aggregation tasks without introducing any single point of failure.

These developments are part of the process of modernizing the ATLAS metadata aggregation tasks.

6 Acknowledgements

Over the years, we have been helped and supported by many people at CC-IN2P3 and in the ATLAS collaboration, in particular: Osman Aidel, Luca Canali, Philippe Cheynet, Benoît Delaunay, Elizabeth Gallas, Pierre-Etienne Macchi, Mattieu Puel and Jean-René Rouet.

References

[1] J. Fulachier, O. Aidel, S. Albrand, F. Lambert, Looking back on 10 years of the ATLAS Metadata Interface. Proceedings of the 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP) J. Phys.: Conf. Ser. **513**, 042019 (2013). https://doi.org/10.1088/1742-6596/513/4/042019

[2] J. Odier, O. Aidel, S. Albrand, J. Fulachier, F. Lambert, Evolution of the architecture of the ATLAS Metadata Interface (AMI). Proceedings of the 21st International Conference on Computing in High Energy and Nuclear Physics (CHEP) J. Phys.: Conf. Ser. **664**, 042040 (2015). https://doi.org/10.1088/1742-6596/664/4/042040

[3] J. Odier, F. Lambert, J. Fulachier, The ATLAS Metadata Interface (AMI) 2.0 metadata ecosystem: new design principles and features. Proceedings of the 23rd International Conference on Computing in High Energy and Nuclear Physics (CHEP) EPJ Web of Conf.: Conf. Ser. **214**, 05046 (2019). https://doi.org/10.1051/epjconf/201921405046

[4] The ATLAS Collaboration et al., The ATLAS Experiment at the CERN Large Hadron Collider. JINST **3**, S08003 (2008). https://doi.org/10.1088/1748-0221/3/08/S08003

[5] Calvo, M., Benoît, A., Catalano, A. et al., The NIKA2 Instrument, A Dual-Band Kilopixel KID Array for Millimetric Astronomy. J Low Temp Phys **184**, 816–823 (2016). https://doi.org/10.1007/s10909-016-1582-0

[6] Ayres, N.J., Ban, G., Bienstman, L. et al., The design of the n2EDM experiment. Eur. Phys. J. C **81**, 512 (2021). https://doi.org/10.1140/epjc/s10052-021-09298-z

[7] Organization for the Advancement of Structured Information Standards [consortium]: https://www.oasis-open.org/ [accessed 2023-07-03]
[8] Message Queuing Telemetry Transport (MQTT) 3.X [specifications]: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html [accessed 2023-07-03]
[9] Message Queuing Telemetry Transport (MQTT) 5.0 [specifications]: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html [accessed 2023-07-03]
[10] J. Odier, F. Lambert, J. Fulachier, This Design principles of the Metadata Querying Language (MQL) implemented in the ATLAS Metadata Interface (AMI) ecosystem. Proceedings of the 24th International Conference on Computing in High Energy and Nuclear Physics (CHEP) EPJ Web of Conf.: **245**, 04044 (2020). https://doi.org/10.1051/epjconf/202024504044
[11] Java Servlet [software]: https://jakarta.ee/ [accessed 2023-07-03]
[12] Crontab [software]: https://man.openbsd.org/crontab.5 [accessed 2023-07-03]
[13] Eclipse Mosquitto [open source MQTT broker]: https://mosquitto.org/ [accessed 2023-07-03]
[14] RabbitMQ [open source MQTT broker]: https://www.rabbitmq.com/ [accessed 2023-07-03]
[15] Eclipse foundation [organisation]: https://www.eclipse.org/ [accessed 2023-07-03]
[16] Eclipse Paho [open source MQTT client]: https://mosquitto.org/ [accessed 2023-07-03]
[17] WebSocket [specifications]: https://datatracker.ietf.org/doc/html/rfc6455 [accessed 2023-07-03]
[18] JSON Web Token [specifications]: https://datatracker.ietf.org/doc/html/rfc7519 [accessed 2023-07-03]
[19] Mosquitto IP and JWT authentication plugin [open source software plugin]: https://github.com/odier-io/mosquitto-ip-jwt-auth [accessed 2023-07-03]
[20] AMIMQTTClientJS [open source software library]: https://github.com/ami-team/AMIMQTTClientJS [accessed 2023-07-03]
[21] Node-RED [open source flow-based and low-code development tool]: https://nodered.org/ [accessed 2023-07-03]
[22] Elasticsearch [open source full-text search engine]: https://www.elastic.co [accessed 2023-07-03]