# Deploying the ATLAS Metadata Interface (AMI) stack in a Docker Compose or Kubernetes environment

*Fabian* Lambert[1,*], *Jérôme* Odier[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. This paper describes how a renewed architecture and integration with modern technologies ease the usage and deployment of a complete AMI stack. It describes how to deploy AMI in a Docker Compose or Kubernetes environment, with a particular emphasis on the registration of existing databases, the addition of more metadata sources, and the generation of high level Web search interfaces using dedicated wizards.

## 1 Introduction

Originally developed for the ATLAS[1] experiment at the CERN Large Hadron Collider (LHC), the ATLAS Metadata Interface (AMI) is a versatile ecosystem for metadata aggregation, transformation, and database storage. Leveraging nearly 20 years of feedback [2, 3], it offers a wide range of tools (command line tools, lightweight clients) and Web interfaces for data search based on metadata criteria.

The second version of AMI, released in 2018, was specifically designed to ensure scalability, adaptability, and maintainability. It is ideally suited for scientific experiments dealing with big data. The design principles and key features are detailed in [4].

With the upcoming HL-LHC data taking in 2029, the volume of metadata to manage and process will significantly increase, likely leading to activity peaks during certain periods. Deploying AMI within a Docker Compose [5] or Kubernetes [6] environment would enable the allocation of required resources on demand, further enhancing the robustness and scalability of the entire AMI ecosystem.

This paper presents the deployment and administration of the AMI ecosystem in a Docker Compose or Kubernetes environment. Subsequent sections delve into the AMI ecosystem stack within Docker and provide guidance on deploying, configuring and utilizing it.

## 2 Overview of the AMI ecosystem

The AMI ecosystem consists of a set of standalone components packaged into Docker images. (see fig. 1). The complete stack is designed to cover all the metadata needs of scientific collaborations. This section gives an overview of each image in the framework.

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

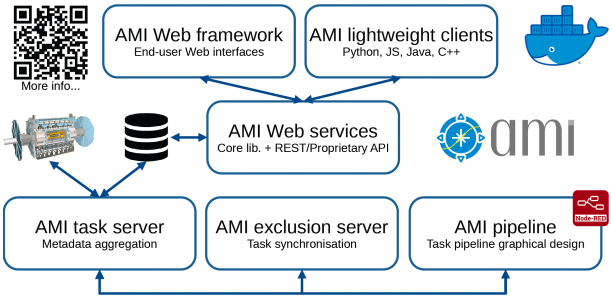


**Figure 1.** The AMI ecosystem docker stack.

### 2.1 AMI Web framework

The AMI Web Framework (AWF) is a JavaScript-based framework for developing metadata-oriented Web applications based on the Twitter Bootstrap CSS framework [7], JQuery [8], and a proprietary JavaScript implementation of the Twig template engine [9]. AWF offers an API [10] for developing graphical controls and applications following the Model-View-Controller (MVC) design pattern to allow code decoupling. It uses the Fetch API (the new replacement for Asynchronous JavaScript And XML (AJAX)) and the Message Queuing Telemetry Transport[11] (MQTT) protocol to interact with the AMI Web Backend. AWF provides the administration and configuration Web interfaces for the whole stack (see section 5) and a collection of ready-to-use metadata-oriented applications such as the multi-criteria search interface or the database schema viewer.

### 2.2 AMI Web services

The AMI Web services are composed of the AMI Java Core Library and two Jakarta EE Web services (implementing REST and proprietary command-oriented APIs). The core library provides a collection of primitives for data connectivity / aggregation / processing / storage. It offers high-level features for data searching by metadata criteria using the AMI Metadata Query Language (MQL) [12], a Domain-Specific Language (DSL), designed for querying databases without requiring any knowledge of the table relationships.

### 2.3 AMI task server

The AMI Task Server is a kind of general-purpose distributed cron[13] used to execute metadata aggregation, processing and storage tasks within a pipeline. It is able to run any program or script but is optimized to use the Java Core Library. Its priority-based lottery scheduler guarantees starvation-free operations. MQTT is used to monitor the global state of a stack : resource usage, number of running tasks, issues, etc. The AMI task server can be deployed in a standalone way. It comes with a dedicated application, running on desktop or Web, that makes it possible to fully control and administer a cluster of AMI task servers (see fig. 2).

### 2.4 AMI exclusion server

The AMI Exclusion Server is an optional zero-configuration server used to pilot the task mutual exclusions (i.e. distributed mutexes to avoid concurrent data modifications) in a cluster of AMI Task Servers.

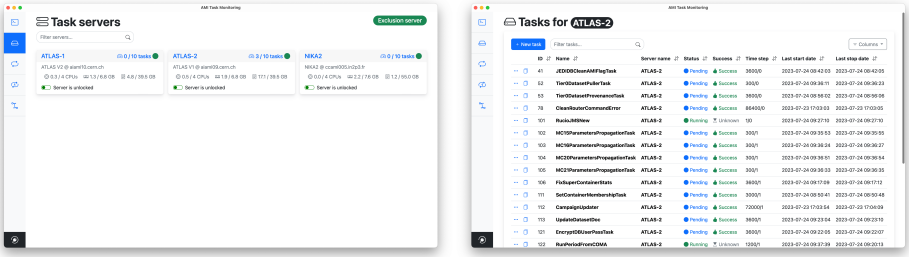


**Figure 2.** Application to administer a cluster of AMI Task Servers. The list of task server nodes is shown on the left, and the list of tasks for the "ATLAS-2" node is shown on the right.

### 2.5 AMI pipeline

Based on Node-RED [14], a flow-based, low-code development tool for visual programming, the AMI Pipeline interface makes it possible to graphically define pipelines of tasks. Pipelines are executed by an AMI Task Server stack and not directly by Node-RED. But its MQTT routing capabilities make the implementation of high-level monitoring and issue treatment easy.

### 2.6 AMI lightweight clients

The AMI ecosystem provides a set of lightweight clients for querying the Web services. The most commonly used within ATLAS is pyAMI [15], a Python client. Additionally, JAMI [16], a Java client, CAMI [17], a C++ client and ami-http-client [18], an NPM JavaScript client are officially supported.

## 3 Containerization of the AMI ecosystem

This section describes the continuous integration process implemented from code updates by developers to the final Docker image made available for end-users.

### 3.1 The continuous integration system

The AMI ecosystem code is versioned in private GitLab repositories. At each commit, the continuous integration system (GitLab CI and Jenkins) performs a static code analysis. If successful, the package is automatically compiled and deployed in a Reposilite[19] artifact server (https://repo.ami-ecosystem.in2p3.fr/). If the version is not a snapshot, the continuous integration triggers an action on the GitHub repository describing the Docker image in order to recompile and deploy it on Docker Hub (see section 3.2).

### 3.2 The AMI ecosystem docker images

All the aforementioned software components have a public Docker image definition on GitHub and Docker Hub [20]. For each image, the exhaustive description and configuration details (environment variables) can be found in the corresponding GitHub "README.md" file.

- *AWF*: https://github.com/ami-team/docker-ami-web-framework,
  Docker Hub : amiteam/web-framework:latest
- *AMI Server*: https://github.com/ami-team/docker-ami-server,
  Docker Hub : amiteam/server:latest
- *AMI Task Server*: https://github.com/ami-team/docker-ami-task-server,
  Docker Hub : amiteam/task-server:latest
- *AMI exclusion server*: https://github.com/ami-team/docker-ami-exclusion-server,
  Docker Hub : amiteam/exclusion-server:latest
- *AMI MQTT server*: https://github.com/ami-team/docker-ami-mosquitto,
  Docker Hub : amiteam/mosquitto:latest
- *AMI pipeline service*: https://github.com/ami-team/docker-ami-pipeline,
  Docker Hub : amiteam/pipeline:latest

# 4 Deploying the AMI ecosystem

## 4.1 Deploy the full AMI ecosystem stack

All the AMI ecosystem docker images can be integrated in a Docker Compose or Kubernetes environnement in order to be deployed together. A "docker-compose"-based demo of the complete AMI ecosystem is accessible at https://github.com/ami-team/AMIDemo. It can be executed locally on any Linux or OSX machine with Git and Docker Compose V2 [21] installed by following the instructions provided:

```
git clone https://github.com/ami-team/AMIDemo.git
cd AMIDemo
docker compose up
```

The AMI Demo "docker-compose.yml" file is a good starting point for deploying a new AMI ecosystem full stack. The "docker-compose.yml" file references all the available Docker images and environment variables to be set up.

On request, the AMI Team can provide a basic profile for deploying AMI in a Kubernetes cluster.

# 5 Configuring the AMI ecosystem

This section describes the configurations to be completed after deploying the AMI ecosystem via Docker.

## 5.1 The AMI setup wizard

The AMI ecosystem has a configuration database (named *router* database) storing global configuration, user profiles, database credentials, other metadata, etc. To initialize this database, an administrator must execute the AMI setup wizard in a Web browser (see fig. 3). It consists of three steps, "server configuration", "*router* database configuration" and "custom extension configuration".

When deploying AMI via Docker, the wizard can be pre-initialized by specifying environment variables (see https://github.com/ami-team/AMIDemo/blob/master/docker-compose.yml).

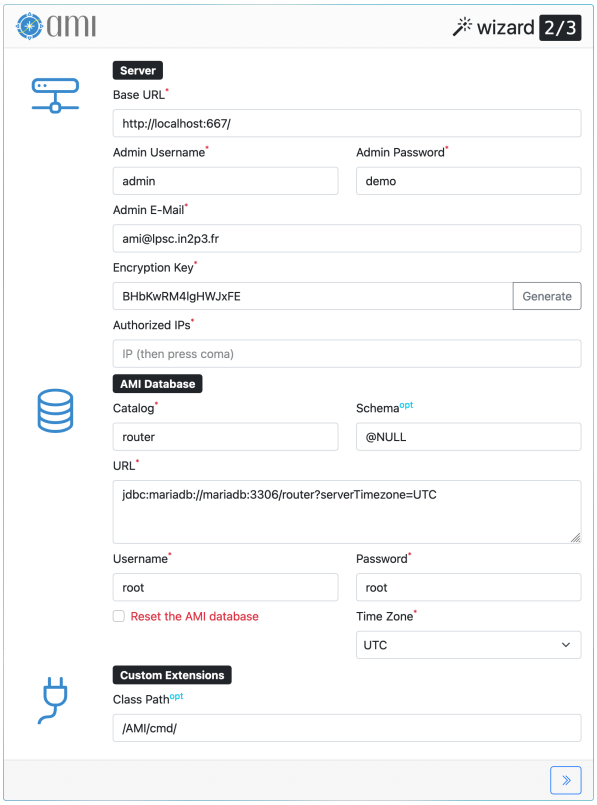


**Figure 3.** The AMI wizard form.

### 5.2 The admin dashboard

After the setup phase, user profiles and database credentials have to be supplied via the dedicated Admin Dashboard (see fig. 4) accessible via the Admin menu. It provides the following interfaces:

- *Conf.*: used for setting up database pools and caches, SMTP credentials, log aggregators, MQTT and Node-Red endpoints, user authentication methods (SSO for instance), etc.
- *Pages*: used to define the home page of the AMI website and add additional pages if needed. It is a kind of embedded Content Management System (CMS).
- *Roles*: used to define roles that will later determine the subset of commands a user can execute.
- *Commands*: used for managing existing commands, adding user-defined ones and their attached roles. Commands are exposed to the end users via the AMI REST and proprietary APIs.
- *Users*: used for managing users and their attached roles.
- *Catalogs*: used for appending new catalogs by providing both a connection string and credentials (see section 5.2.1). AMI automatically extracts the metadata and relation graph of each database (named catalog), table (named entity) and column (named field).
- *Schema Viewer*: A graphical tool displaying all the catalogs registered in AMI. Administrators can define various settings at catalog or field levels (encrypted or hash field, admin only, JSON, groupable, etc.).

- *Search Modeler*: A graphical tool for configuring the interface for searching data by metadata criteria. This tool is described in section 5.2.2.

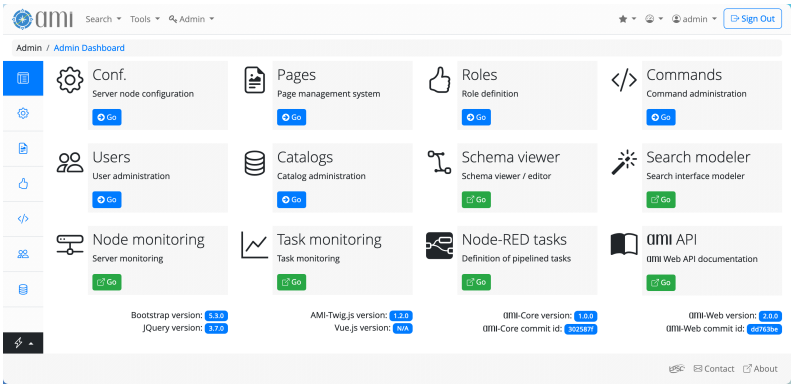


**Figure 4.** The AMI admin dashboard.

### *5.2.1 The catalog administration tool*

The Catalog Interface (see fig. 5) consists of a form where catalogs (aka. databases) are declared by specifying JDBC[22] connection string, credentials, ... The *Foreign key* tab permits the creation of virtual relations between tables or views. From the AMI perspective, these virtual relations are strictly equivalent to SQL ones, but offer additional capabilities such as across catalog relations.

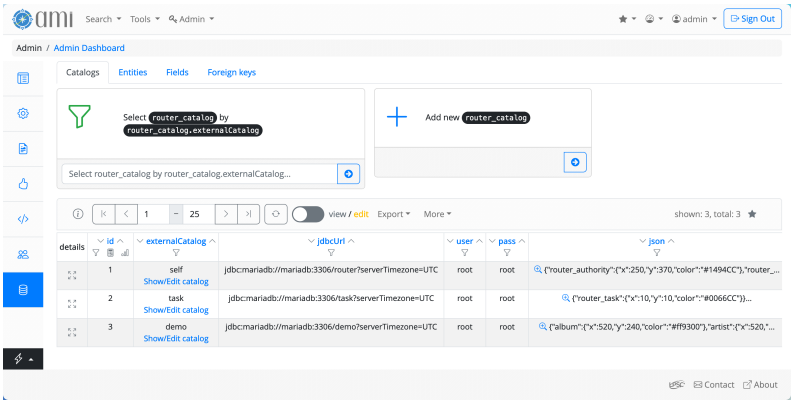


**Figure 5.** The Catalog Interface.

### *5.2.2 The search modeler tool*

For end-users, the AMI ecosystem provides a generic multi-criteria search tool (see fig. 6).

Several instances of this tool can be provided to end-users (for example one per catalog). These instances are configured with the Search Modeler tool (see fig. 7). All searchable catalogs, tables and fields are automatically detected and an administrator just have to specify the desired selection criteria (for example, a dataset identifier or any other information useful for the data selection).

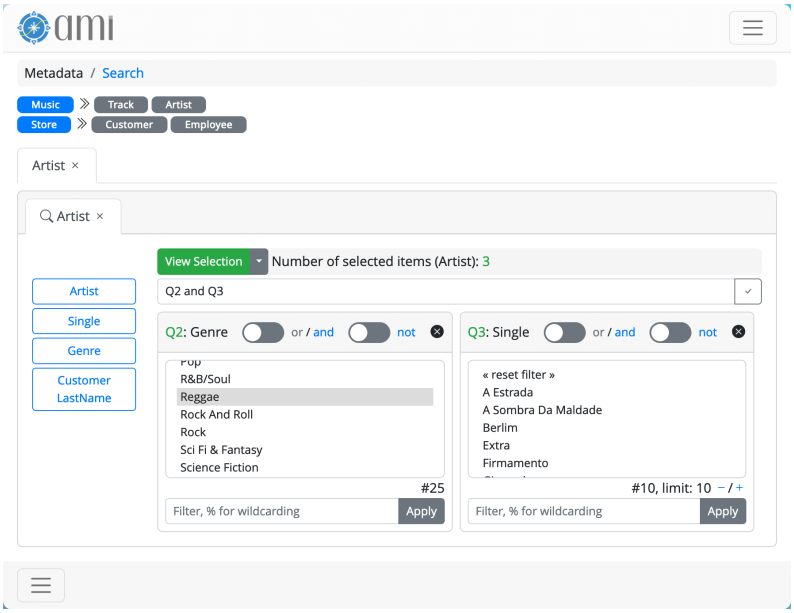


**Figure 6.** The multi-criteria search tool.

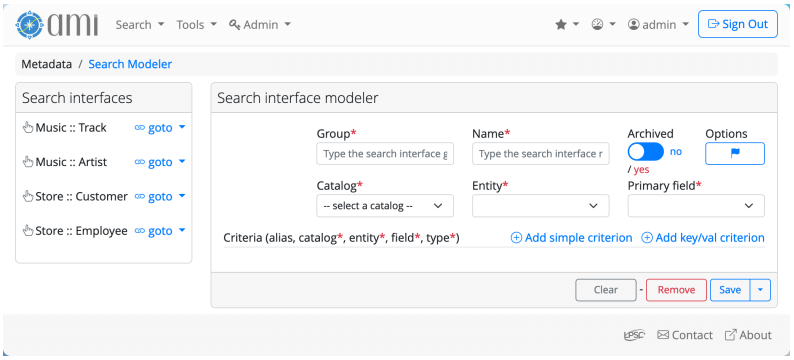


**Figure 7.** The Search Modeler tool.

## 6 Conclusion

AMI is a robust and mature metadata ecosystem that offers a wide range of services and Web applications. This paper shows how to deploy AMI in a Docker environment (Docker Compose, Kubernetes, etc.). These well-established technologies simplify deployment of the AMI stack and make scaling up relatively easy. This approach will in particular address the challenge of HL-LHC data taking by adapting resource usage to the activity peaks of experiments.

## 7 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] 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

[2] 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

[3] 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

[4] 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

[5] Docker [software]: https://www.docker.com/ [accessed 2023-07-03]

[6] Kubernetes [software]: https://kubernetes.io/ [accessed 2023-07-03]

[7] Twitter Bootstrap [software]: https://getbootstrap.com/ [accessed 2023-07-03]

[8] JQuery [software]: https://jquery.com/ [accessed 2023-07-03]

[9] AMI-Twig.js [software]: https://ami.web.cern.ch/twig/ [accessed 2023-07-03]

[10] AWF API [specification]: https://ami-ecosystem.in2p3.fr/doc/technical-guide/ami-web-framework/api/global/ [accessed 2023-07-03]

[11] MQTT [software]: https://mqtt.org/ [accessed 2023-07-03]

[12] J. Odier, F. Lambert, J. Fulachier, 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 Conferences **245**, 04044 (2020). https://doi.org/10.1051/epjconf/202024504044

[13] Crontab [software]: https://man.openbsd.org/crontab.5 [accessed 2023-07-03]

[14] Node-RED [software]: https://nodered.org/ [accessed 2023-07-03]

[15] pyAMI [software]: https://github.com/ami-team/pyAMI-core/ [accessed 2023-07-03]

[16] JAMI [software]: https://github.com/ami-team/jami/ [accessed 2023-07-03]

[17] CAMI [software]: https://github.com/ami-team/cami/ [accessed 2023-07-03]

[18] AMIHTTPClientJS [software]: https://github.com/ami-team/AMIHTTPClientJS/ [accessed 2023-07-03]

[19] Reposilite [software]: https://reposilite.com/[accessed 2023-07-03]

[20] Docker Hub [software]: https://hub.docker.com/ [accessed 2023-07-03]

[21] Docker Compose [software]: https://docs.docker.com/compose/ [accessed 2023-07-03]

[22] Java DataBase Connectivity (JDBC) [software]: https://www.oracle.com/database/technologies/appdev/jdbc.html [accessed 2020-01-10]