ARGO Messaging Service - AMS

The ARGO Messaging Service (AMS) is a Publish/Subscribe Service, which implements the Google PubSub protocol. Instead of focusing on a single Messaging API specification for handling the logic of publishing/subscribing to the broker network the API focuses on creating nodes of Publishers and Subscribers as a Service. It provides an HTTP API that enables Users/Systems to implement message oriented service using the Publish/Subscribe Model over plain HTTP.

Presentation:

ARGO-Messaging-Service-GRNET

Documentation:

http://argoeu.github.io/messaging/v1/overview/

Swagger API:

https://api-doc.argo.grnet.gr/argo-messaging/

Features

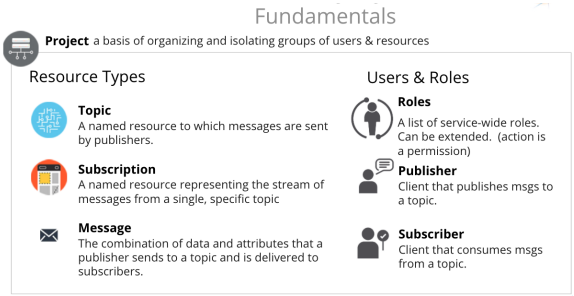
- **Ease of use:** It supports an HTTP API and a python library so as to easily integrate with the AMS.
- **Push Delivery**: S instantly pushes asynchronous event notifications when messages are published to the message topic. Subscribers are notified when a message is available.
- **Replay messages**: replay messages that have been acknowledged by seeking to a timestamp.
- **Schema Support**: on demand mechanism that enables a) the definition of the expected payload schema, b) the definition of the expected set of attributes and values and c) the validation for each message if the requirements are met and immediately notify client
- **Replicate messages on multiple topics**: Republisher script that consumes and publishes messages for specific topics (ex. SITES)

Architectural aspect

- **Durability**: provide very high durability, and at-least-once delivery, by storing copies of the same message on multiple servers.
- **Scalability**: It can handle increases in load without noticeable degradation of latency or availability
- **Latency**: A high performance service that can serve more than 1 billion messages per year
- **Availability**: it deals with different types of issues, gracefully failing over in a way that is unnoticeable to end users. Failures can occur in hardware, in software, and due to load.

Fundamentals

In the Publish/Subscribe paradigm, Publishers are users/systems that can send messages to named-channels called Topics. Subscribers are users/systems that create Subscriptions to specific topics and receive messages.



- **Topics:** Topics are resources that can hold messages. Publishers (users/systems) can create topics on demand and name them (Usually with names that make sense and express the class of messages delivered in the topic)
- **Subscriptions:** In order for a user to be able to consume messages, he must first create a subscription. Subscriptions are resources that can be created by users on demand and are attached to specific topics. Each topic can have multiple subscriptions but each subscription can be attached to just one topic. Subscriptions allows Subscribers to incrementally consume messages, at their own pace, while the progress is automatically tracked for each subscription.
- **Message:** The combination of data and (optional) attributes that a publisher sends to a topic and is eventually delivered to subscribers.
- **Message attribute:** A key-value pair that a publisher can define for a message.

Pull vs Push Subscriptions

AMS supports both push and pull message delivery. In push delivery, the Messaging Service initiates requests to your subscriber application to deliver messages. In pull delivery, your subscription application initiates requests to the Pub/Sub server to retrieve messages.

Pull subscriptions

Pull subscriptions can be configured to require that message deliveries are acknowledged by the Subscribers. If an acknowledgement is made, subscription can resume progressing and send the next available messages. If no acknowledgement is made subscription pauses progressing and re-sends the same messages.

In a pull subscription, the subscribing application explicitly calls the API pull method, which requests delivery of a message in the subscription queue. The Pub/Sub server responds with the message (or an error if the queue is empty), and an ack ID. The subscriber then explicitly calls the acknowledge method, using the returned ack ID, to acknowledge receipt.

Push subscriptions

In a push subscription, the push server sends a request to the subscriber application, at a preconfigured endpoint. The subscriber's HTTP response serves as an implicit acknowledgement: a success response indicates that the message has been successfully processed and the Pub/Sub system can delete it from the subscription; a non-success response indicates that the Pub/Sub server should resend it (implicit "nack"). To ensure that subscribers can handle the message flow, the Pub/Sub dynamically adjusts the flow of requests and uses an algorithm to rate-limit retries.

The push server(s) are an optional set of worker-machines that are needed when the AMS wants to support push enabled subscriptions.

- It allows to decouple the push functionality from AMS api nodes
- They perform the push functionality for the messages of a push enabled subscription (consume->deliverack)/
- Provide a gRPC interface in order to communicate with their api
- Provide subscription runtime status

Apart from all these the Messaging Service supports:

- **Argo-ams-library**: A simple library to interact with the ARGO Messaging Service.
- **Argo-AuthN**: Argo-authn is a new Authentication Service. This service provides the ability to different services to use alternative authentication mechanisms without having to store additional user info or implement new functionalities.The AUTH service holds various information about a service's users, hosts, API urls, etc, and leverages them to provide its functionality.
- **AMS Metrics**: Metrics about the service and the usage.

How can i use/test the AMS ?

Ideas

Some ideas about how to use the service

- **Balancing workloads in network clusters:** For example, a large queue of tasks can be efficiently distributed among multiple workers (e.g. Fedcloud instances).
- **Implementing asynchronous workflows:** For example, an order processing application can place an order on a topic, from which it can be processed by one or more workers.
- **Publishing of monitoring results:** Monitoring Engines can push monitoring results through the ARGO Monitoring Service
- **Publishing of accounting data:** Act as the transport layer for the secure exchange of accounting information
- **Log aggregation**: Log aggregation from different services in your infrastructure

I want to use the service

Follow the steps

1. GGUS ticket at Messaging Support Unit with:
    a. Small description of the integration - use of Messaging service (pull, push option)

    b. An email to associate the new Messaging user
2. The messaging team will create an new project /topic/subscription to the devel infrastructure for testing.
3. The messaging team will close the ticket and will respond to the user email by providing
    a. User token
    b. Service endpoint
    c. URL to the detailed documentation and some examples ready to use
4. When the user is ready we will apply the appropriate configuration to the production infrastructure

Use Cases

The following Services rely on the AMS Service:

- **Operations Portal**: Reads the alarms from predefined topics, store them in a database and displays them in the operations portal. Whenever a service / endpoint changes status (warning, unknown , critical) the ARGO Monitoring engine raises an alarm. These alarms, which are used by the operations portal. ARGO Monitoring engine sends these alarms in a predefined forma as a message to alarms topic of EGI Project in the ARGO Messaging Service. These alarms are consumed by the Operations Portal via alarm subscription.
- **Accounting**: Use of AMS as a transport layer for collecting accounting data from the Sites. The accounting information is gathered from different collectors into a central accounting repository where it is processed to generate statistical summaries that are available through the EGI Accounting Portal. The software used for transferring accounting records (SSM) is using the ARGO Messaging System (in testing mode). The SSM publishes messages to predefined topics.
- **FedCloud** : Use of AMS as a transport layer of the cloud information system. It makes use of the ams-authN. The entry point for users, topics and subscriptions is GOCDB. A utility python script reads the xml feed from GOCDB, creates the respective ams users under the specified project, assigns to the correct project's topic, creates a binding for each user, using the dn from GOCDB and finally creates topics with the schema SITE_sitename_ENDPOINT_id_in_gocdb. Each site publishes the necessary information to the predefined topic and periodically this information is consumed by the corresponding subscription by AppDB info consumer.
- **ARGO Availability and Reliability Monitoring Service**: It uses the AMS service to send the messages from the monitoring engine to other components. he Monitoring Engine is using the AMS and sends all the raw metric data to the metrics topic. These messages are consumed by different components of the ARGO framework but mainly by the compute/analytics engine. The analytics engine - apart from the metrics raw data - is using data from different sources of truth. All these data are published in the AMS by different connectors and are consumed by the analytics engine so as to produce status, availability and reliability reports.