EOSC Technical Specification

Federation Services

Monitoring

Version:	1
Status:	EOSC-hub Proposal
Dissemination Level:	Public
Document Link:	https://wiki.eosc-hub.eu/display/EOSCDOC/Monitoring

Abstract

Monitoring allows to quickly detect, correlate, and analyse data for a fast reaction to anomalous behaviour that may affect end-users and ultimately the productivity of the organization. The key functional requirements of a monitoring system are monitoring of services, reporting availability and reliability, visualization of the services status, provide dashboard interfaces and sending real-time alerts.

This document describes the high-level service architecture for an EOSC Monitoring service and presents the main integration and usage use cases for monitoring in EOSC. It proposes interfaces as guidelines to be followed to achieve the interoperability between monitoring systems in EOSC for three envisaged use cases: (1) combine Results of one or more infrastructures in EOSC in a unified UI, (2) add a Service Provider/Infrastructure to EOSC Monitoring and (3) Third-party services exploiting EOSC Monitoring data.

EOSC-HUB RECEIVES FUNDING FROM THE EUROPEAN UNION'S HORIZON 2020 RESEARCH AND INNOVATION PROGRAMME UNDER GRANT AGREEMENT NO. 777536.

COPYRIGHT NOTICE

DELIVERY SLIP

Date	*Name*	*Partner/Activity*
From:	Themis Zamani Kostas Koumantaros Pavel Weber Diego Scardaci	GRNET GRNET KIT EGI Foundation
Reviewed by:	Jens Jensen Catalin Condurache	STFC EGI Foundation

DOCUMENT LOG

Issue	*Date*	*Comment*	*Author*
v.1	25/03/2020	First release ready for public consultation	Themis Zamani (GRNET), Kostas Koumantaros (GRNET), Pavel Weber (KIT), Diego Scardaci (EGI Foundation)

TERMINOLOGY

https://wiki.eosc-hub.eu/display/EOSC/EOSC-hub+Glossary

Contents

1 Introduction

Monitoring is the key service needed to gain insights into an infrastructure. It needs to be continuous and on-demand to quickly detect, correlate, and analyse data for a fast reaction to anomalous behaviour. The challenge of this type of monitoring is how to quickly identify and correlate problems before they affect end-users and ultimately the productivity of the organization. Management teams can monitor the availability and reliability of the services from a high-level view down to individual system metrics and monitor the conformance of multiple SLAs. The key functional requirements are:

- Monitoring of services
- Reporting availability and reliability,
- Visualization of the services status,
- Provide dashboard interfaces,
- Sending real-time alerts.

The dashboard design should enable easy access and visualisation of data for end-users. APIs should also be supported to allow third parties to gather monitoring data from the system through them.

The key requirements of a monitoring system are:

- Support for multiple entry points (***different types of systems can work together***)
- Interoperable
- High availability of the different components of the system
- Loosely coupled: support API's in the full stack so that components are independent in their development cycles
- Support for Multiple Tenants, Configurations, Metrics and profiles to add flexibility and ease of customisation.

2 High-level Service Architecture

The service collects status (metrics) results from one or more monitoring engine(es) and delivers daily and/or monthly availability (A) and reliability (R) results of distributed services. Both status results and A/R metrics are presented through a Web UI, with the ability for a user to drill-down from the availability of a site to individual test results that contributed to the computed figure.

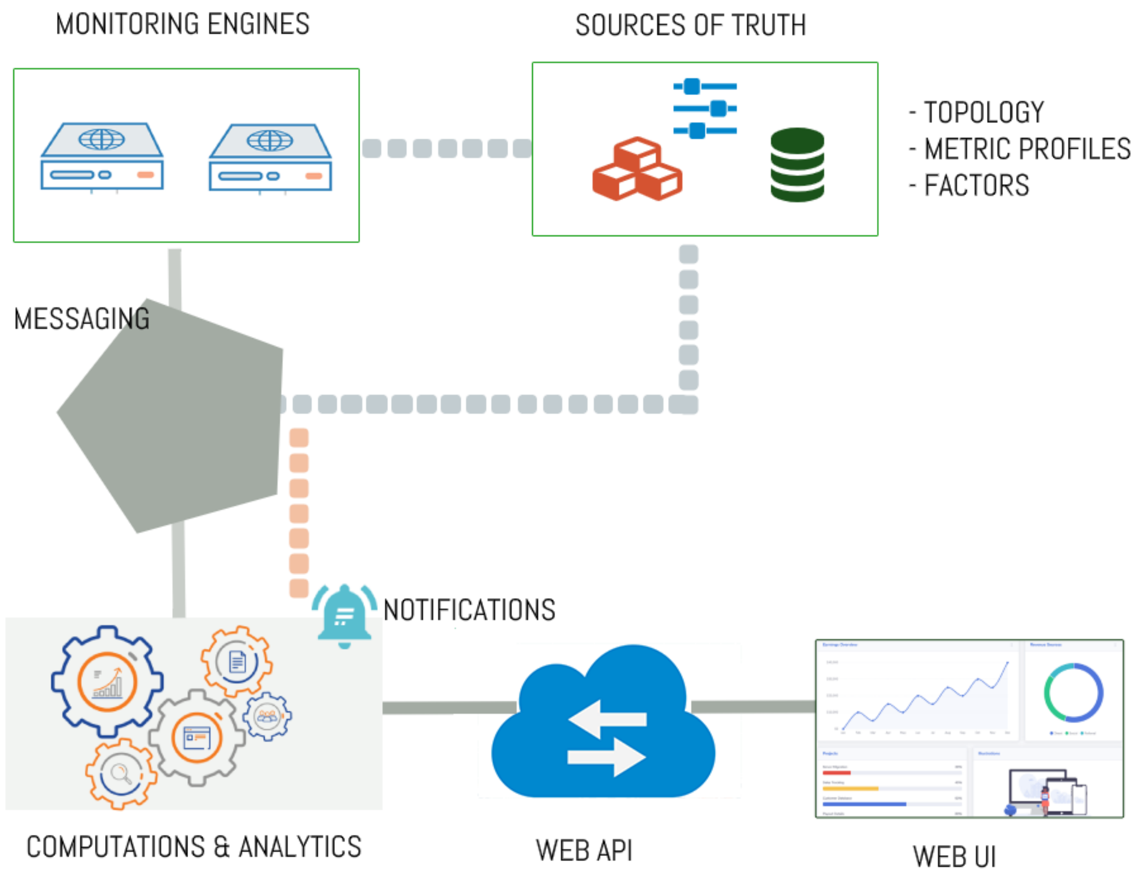

Figure 1. High level architecture of a Monitoring service

Monitoring Engine(s): This service component executes the service checks against the infrastructure and delivers the metric data (probe check results) to the Messaging Service.

Sources of Truth: The Monitoring system should support a number of connector plugins that are able to fetch topology, Metrics and Factors from various sources such as the CMDB and Operations Portal. It also offers a Metric and Profile Management Component which is used in order to define checks (probes) and associate them to service types. Each grouping of checks and service types forms a profile. The authorization and authentication with, and between, the sources of truth are based on X509 wherever needed.

Messaging: The monitoring system depends on a Pub/Sub Messaging Service to be in place, in order to facilitate the communication between its components.

Computations & Analytics: This component of the system should include computational job definitions for ingesting data, calculating status and availability/reliability and a management service to automatically configure, deploy and execute those jobs on a distributed processing engine for stateful computations. At the same time this component analyses the monitoring results and sends notifications based on a set of rules, to inform the users (operators, NGIs) about the status of their services.

The result of the computations should be stored in a distributed file system (in a highly fault-tolerant system). It should provide high throughput access to application data and should be suitable for applications that have large data sets. Apart from the storage of the raw data in a distributed file system, data should also be stored in a document database designed for ease of development and scaling.

WEB API: Rest-like HTTP API service that provides access to status and availability/reliability results. It supports token based authentication and authorization with established roles. Results are provided in JSON Format.

WEB UI: The Web UI is the component used to store, consolidate and "feed" data into the web application. The global information from the primary and heterogeneous data sources is retrieved by means of the different plugins. The collected information is structured and organized within configuration files in the service and, finally, made available to the web application without the need for any further computations. This modular architecture is conceived in order to make it easy to add new data sources and to use cached information if a primary source is unavailable. The resulting data is exposed as XML views through a RESTful web service interface.

3 Adopted standards

The following tables list the standards, API and protocols recommended in this specification.

Table 1. Adopted standards

Standard	Short description	References
REST	Loosely adhere to the REST paradigm.	https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
SAML2	XML based protocol that is used to securely pass the credentials information from Identity provider to Service point (usually web application) that needs it.	https://wiki.oasis-open.org/security/FrontPage
X.509	X.509 is an ITU-T standard for a public key infrastructure (PKI), also known as PKIX (PKI X509) Used for authorization and authentication purposes with the sources of truth.	https://www.rfc-editor.org/info/rfc5280
Apache Avro	Data serialization system	http://avro.apache.org/
JSON API	A specification for building apis in JSON format	https://jsonapi.org/

Table 2. Adopted Protocol/API

Protocol/API	Short description	References
HTTPS	TLS secured HTTP	https://tools.ietf.org/html/rfc2818
HTTP / JMX / Shell / SQL / Ldap ...	All the plugins should be based on standard protocols or formats	http://software.in2p3.fr/lavoisier/adaptors.html
Nagios Plugin API	Nagios API provides a reference for the monitoring plugin developers	https://nagios-plugins.org/doc/guidelines.html
Flink DataStream API	Used to execute live streaming computational jobs on the Flink Streaming platform to produce near real-time results for API & notifications	https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/datastream_api.html
Flink DataSet API	Used to execute batch computational jobs on the Flink Streaming platform to produce status and A/R results for the Web API	https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/batch/
HDFS API	Used to store ingested monitoring data along with supplementary data (topology, downtimes, weights etc) in distributed HDFS storage	https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
ARGO API over REST API	The ARGO Web API provides the Serving Layer of ARGO. It is comprised of a high performance and scalable data store and a multi-tenant REST HTTP API, which is used for retrieving the Status, Availability and Reliability reports and the actual raw metric results	http://argoeu.github.io/guides/api/

4 Interoperability guidelines

This section presents the main integration and usage use cases for monitoring in EOSC and proposes ARGO[1] interfaces as guidelines to be followed to achieve the interoperability between monitoring systems in EOSC.

[1] https://argoeu.github.io/

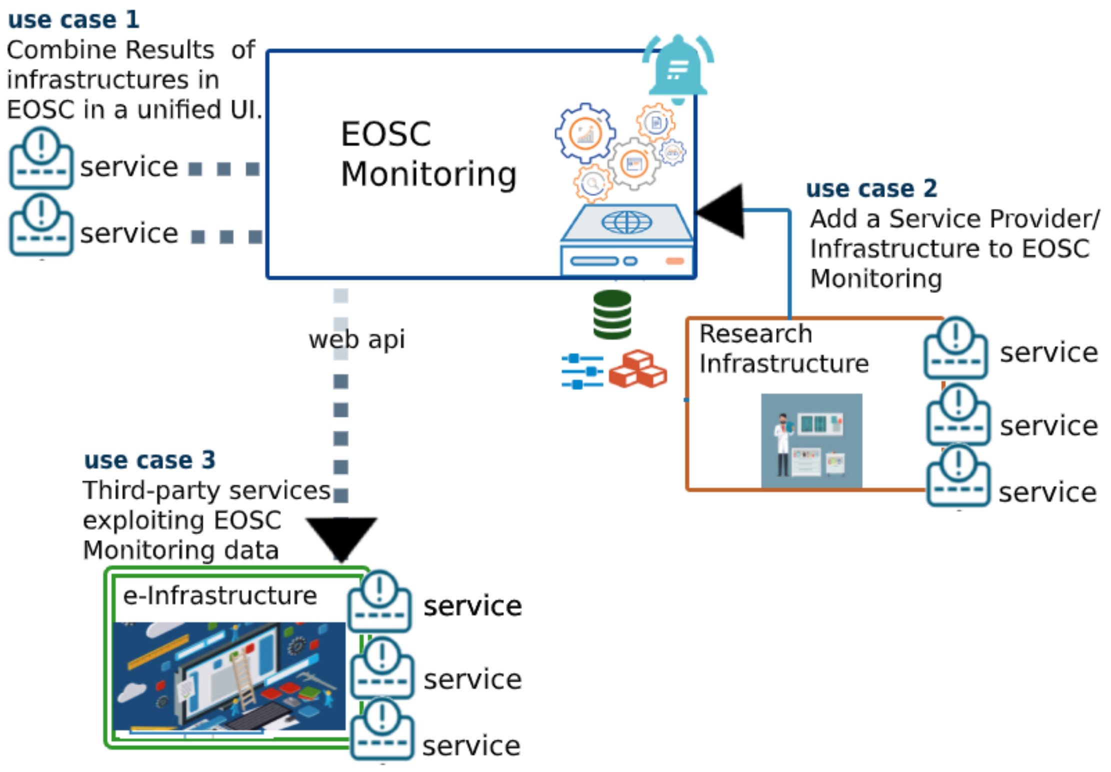
use case 1
Combine Results of infrastructures in EOSC in a unified UI.
service
service
EOSC Monitoring
use case 2
Add a Service Provider/ Infrastructure to EOSC Monitoring
Research Infrastructure
service
service
service
web api
use case 3
Third-party services exploiting EOSC Monitoring data
e-Infrastructure
service
service
service

Figure 2. Integration scenarios in the EOSC Monitoring

Use Case 1: Combine Results of one or more infrastructures in EOSC in a unified UI.

The proposed system should be able to combine the A/R results from different providers / infrastructures in a unified web view. The general goal of "distributed monitoring" is to allow different infrastructures with the same or different environment to scale. There are a number of different options for supporting this. All of them are based on the concept that different sources will publish status and performance data in a predefined form that is read from the core engine of the monitoring service. The data should also be stamped with their source and timestamp. Every metric should be prefixed with [source_type], following the metric naming best practices (based on nagios). Every metric is also labelled with the hostname and service description. These predefined messages should be sent to the Messaging system which is the service responsible to pass them to the computations engine which performs the necessary calculations to produce the reports.

How Argo solves this

The Argo Compute engine uses as source the results of the probes sent by two or more monitoring engines (nagios boxes) via the Argo Messaging Service. Metric data comes in the form of avro files and contains timestamped status information about the hostname, service and specific checks (metrics) that are being monitored. A typical item of information in the metric data avro file contains the field listed in the table below. The compute engine calculates the Availability and Reliability of each service group based on the instructions and mapping given by the Topology and Metric Aggregation & Threshold profiles[2]. This fact allows the compute engine to be flexible enough in

[2] http://argoeu.github.io/guides/argo-compute-engine/input/

order to combine results from a number of sources and produce reports for almost any combination possible. It is therefore able to produce integrated views that combine the topologies of more than one Service Provider or Infrastructure Providers.

Table 3. Description of avro values

Name	Description	Required
hostname	The fqdn address of the host being monitored	YES
service	The name of the specific service being monitored	YES
metric	The name of the specific metric (check) of the service that is being monitored	YES
timestamp	Time of the monitoring check in ZULU time according to ISO-8601 e.g. 2020-03-11T10:39:32Z	YES
status	Status of the metric during the monitoring check	YES
monitoring_host	The fqdn of the monitoring agent	NO
summary	Text containing a summary of the monitoring check	NO
message	Text containing the detailed system output message of the monitoring check probe	NO
tags	Array containing optional user defined tags	NO

Table 4. The avro schema

```
{"namespace": "argo.avro",
"type": "record",
"name": "metric_data",
"fields": [
       {"name": "timestamp", "type": "string"},
       {"name": "service", "type": "string"},
       {"name": "hostname", "type": "string"},
       {"name": "metric", "type": "string"},
       {"name": "status", "type": "string"},
       {"name": "monitoring_host", "type": ["null", "string"]},
       {"name": "summary", "type": ["null", "string"]},
       {"name": "message", "type": ["null", "string"]},
       {"name": "tags", "type" : ["null", {"name" : "Tags",
                                           "type" : "map",
                                           "values" : ["null", "string"]
                                         }]
       }]
}
```

Use Case 2: Add a Service Provider/Infrastructure to EOSC Monitoring

In order to add support for a new Service Provider or Infrastructure in the EOSC Monitoring Service, the provider should only need to provide the topology of the services to be monitored and the equivalent metric and aggregation profiles. The system should take care of all the actions required to probe every endpoint in the topology with the metrics/probes defined and aggregate the results according to the profiles defined and present them in a Web-UI.

How Argo solves this

For each new Tenant ARGO uses as topology input the xml feed by EOSC CMDBs[3] and provides POEM[4], a component to allow management of probes, metrics and profiles. Profiles are the main input required to automatically configure the monitoring engines. The defined checks are executed in regular intervals by the monitoring engines and the results are then passed through the Argo Messaging Service to the Argo Compute Engine[5]. The compute Engine performs the necessary calculations to produce the Availability and Reliability of each endpoint, service and service group defined in the topology according to the Aggregation profiles and then serves the results via the ARGO-Web-API[6] to be presented by the WEB-UI[7].

Use Case 3: Third-party services exploiting EOSC Monitoring data

Any EOSC service should be able to retrieve and use the status information and metrics computed by the EOSC Monitoring system. An API should be provided to allow any authorised third-party service to retrieve such data. This API should return

- Endpoint, service and service group
- Raw data and performance data
- And the data for the sources of truth

How Argo solves this

The ARGO Web API[8] comprises a high performance and scalable data store and a multi-tenant REST HTTP API, which can be used to retrieve the Status, Availability and Reliability reports and the actual raw metric results.

[3] EOSC Configuration Management Databases GOCDB (goc.egi.eu) and DPMT (dp.eudat.eu)
[4] http://argoeu.github.io/guides/poem/
[5] http://argoeu.github.io/guides/argo-compute-engine/
[6] http://argoeu.github.io/guides/api/
[7] http://argoeu.github.io/guides/webui/
[8] http://argoeu.github.io/guides/api/

5 Examples of solutions implementing this specification

List already available Open Source services that fit with the high-level service architecture, Include references to the service web page.

ARGO

ARGO is a flexible and scalable framework for monitoring status, availability and reliability of services provided by infrastructures with medium to high complexity. It can generate multiple reports using customer defined profiles (e.g. for SLA management, operations etc.) and has built-in multi-tenant support in the core framework.

ARGO supports flexible deployment models and its modular design enables ARGO to integrate with external systems (such as CMDBs, Service Catalogues etc.). During the report generation, ARGO can take into account custom factors such as the importance of a specific service endpoint, scheduled or unscheduled downtimes etc.

For the Availability & Reliability monitoring, ARGO relies on a modular architecture comprised of several components described in the next subsections.

The ARGO Monitoring Engine

For status monitoring, ARGO relies on Nagios. All probes developed for ARGO follow the Nagios conventions and can run on any stock Nagios box. ARGO provides an optional set of addons for the stock Nagios that provide features such as auto-configuration from external information sources, publishing results to external Message Brokers etc.

In order to use the new messaging service, the monitoring engine also supports the new AMS Publisher. The AMS publisher is a new component acting as a bridge from Nagios to the ARGO Messaging system. It is an integral part of the software stack running on the ARGO monitoring instance and is responsible for forming and dispatching messages that are results of the Nagios tests. t is running as a UNIX daemon and it consists of two subsystems:

- A queueing mechanism
- A publishing/dispatching part

Messages are cached in a local queue with the help of OCSP Nagios calls and each queue is being monitored by the daemon. After a configurable number of accumulated messages, the publisher that is associated with the queue sends them to the ARGO Messaging system and drains the queue. The argo-nagios-ams-publisher is written in a multiprocessing manner so there is support for multiple queue/publish pairs where for each, a new worker process will be spawned.

The ARGO Connectors

Using custom connectors, ARGO can connect to multiple external Configuration Management Databases and Service Catalogs. Already there are connectors for the EGI and EUDAT e-Infrastructures.

The ARGO Consumer

The ARGO Consumer is ingesting monitoring results in real-time from external Message Brokers. The consumer is responsible for the initial pre-filtering of the monitoring results and for encoding them using the AVRO serialization format before passing them to the Compute Engine.

The ARGO Compute Engine

A powerful and scalable analytics engine built on top of Hadoop and HDFS. The Compute Engine is responsible for the aggregation of the status results and the computation of availability and reliability of composite services using customer defined algorithms. The reorganization of the Compute Engine to support stream processing in real time is one of the key new features. A new streaming layer is introduced. Monitoring results flow through the AMS, to the streaming layer (in parallel to the HDFS). The streaming layer is used in order to push raw metric results to the metric result store and to compute status results and push them to the status store in real-time.

The ARGO Web API

The ARGO Web API provides the Serving Layer of ARGO. It is comprised of a high performance and scalable datastore and a multi-tenant REST HTTP API, which is used for retrieving the Status, Availability and Reliability reports and the actual raw metric results.

The ARGO Web UI

The default web UI is based on the Lavoisier Data Aggregation Framework.

ARGO has been adopted by

- EGI infrastructure
- EUDAT infrastructure

5.1 Procedure to integrate a service with the EOSC Hub Monitoring

For each service specified in the above list, describe the technical steps needed to allow a service to exploit the core service features. Please, be concise and use references if needed.

Follow the steps

1. Open a GGUS ticket on the ARGO/SAM EGI Support Unit with:
    a. Small description of the integration - use of the service
    b. A name for the new project - infrastructure / project / service to monitor
2. The Monitoring team will create a new project into the development infrastructure for testing.
3. If the request refers to a new service type / probe then the probe should follow the guidelines mentioned in the interoperability section:

   https://wiki.eosc-hub.eu/display/EOSC/ARGO+Guidelines+for+monitoring+probes