FIWARE Monitoring

Release

September 28, 2016

Contents

CHAPTER 1

Introduction

FIWARE Monitoring GEri is the key component to allow incorporating monitoring and metering mechanisms in order be able to constantly check the performance of the system, but the architecture should be easily extended to collect data for other required needs. Monitoring involves gathering operational data in a running system. Collected information can be used for several purposes:

- Cloud users to track the performance of their own instances.
- SLA management, in order to check adherence to agreement terms.
- Optimization of virtual machines.

The monitoring system is used by different Cloud GEs in order to track the status of the resources. They use gathered data to take decisions about elasticity or for SLA management. Whenever a new resource is deployed in the cloud, the proper monitoring probe is set up and configured to start providing monitoring data.

FIWARE Monitoring source code can be found here.

CHAPTER 2

Documentation

GitHub's README provides a good documentation summary, and the following cover more advanced topics:

2.1 User & Programmers Guide

2.1.1 Introduction

Welcome the User and Programmers Guide for the Monitoring Generic Enabler. This GE is built up from different distributed components, as depicted in the following figure:

Background and Detail

This User and Programmers Guide relates to the Scalability Manager GE which is part of the Cloud Hosting Chapter. Please find more information about this Generic Enabler in the following Open Specification.

2.1.2 User Guide

This GE does not provide an interactive user interface, hence there is no User Guide. The following section elaborates on programmatic usage.

2.1.3 Programmer Guide

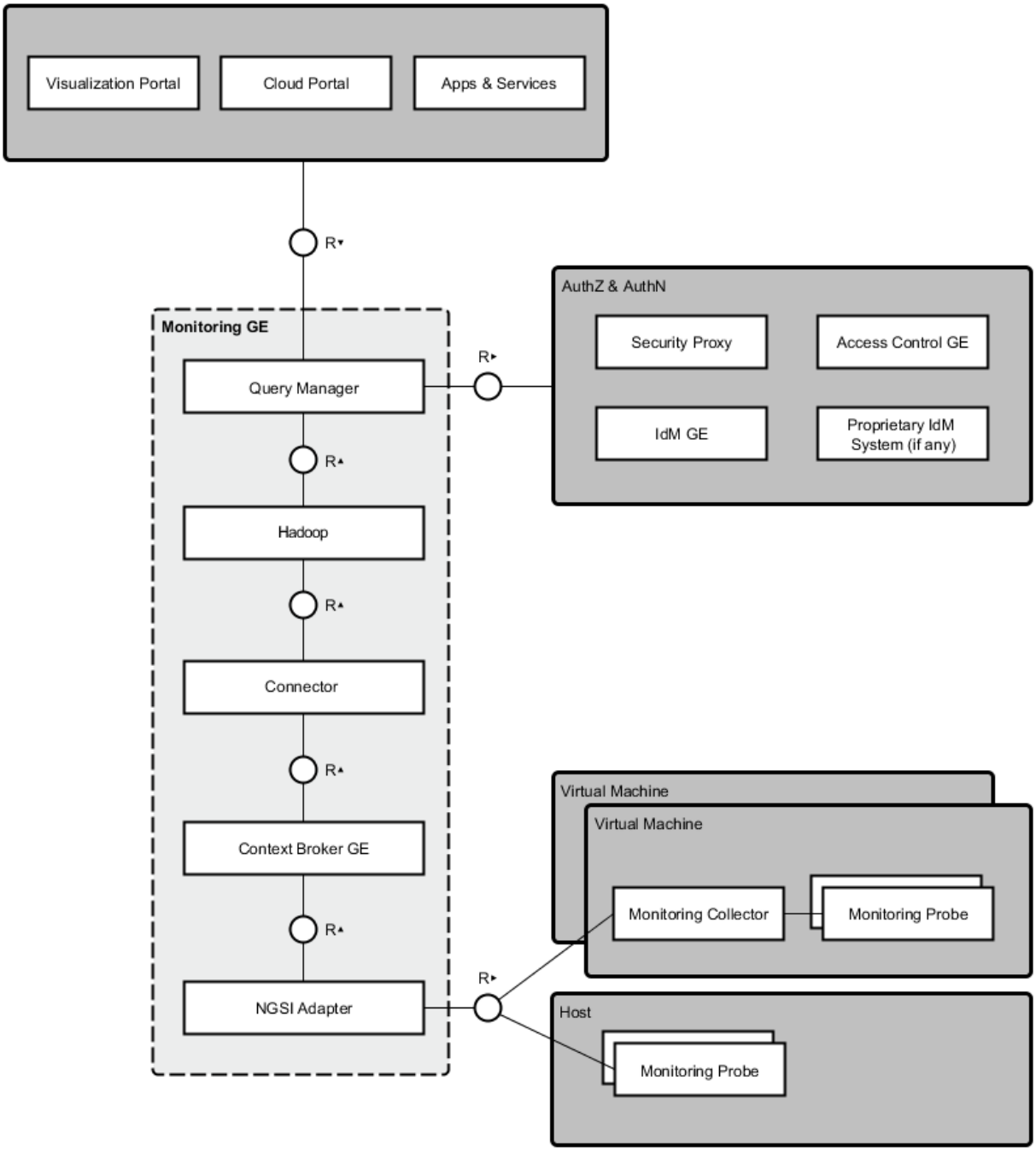
According to the architecture aforementioned, there are several APIs involved in the monitoring process:

- NGSI Adapter API (HTTP)
- NGSI Adapter API (UDP)
- Context Broker API
- Monitoring (*Query Manager*) API

NGSI Adapter API (HTTP)

Probe raw data should be sent as body of a POST request to the adapter, identifying the source entity being monitored in the query parameters. For example, given the following scenario:

Monitored host 178.23.5.23


Visualization Portal
Cloud Portal
Apps & Services
R
Monitoring GE
Query Manager
R
Hadoop
R
Connector
R
Context Broker GE
R
NGSI Adapter
R
AuthZ & AuthN
Security Proxy
Access Control GE
IdM GE
Proprietary IdM System (if any)
Virtual Machine
Virtual Machine
Monitoring Collector
Monitoring Probe
Host
Monitoring Probe

Monitoring tool Nagios

Monitoring probe name check_load

NGSI Adapter endpoint http://adapterhost:1337

then requests would look like:

```
HTTP POST http://adapterhost:1337/check_load?id=178.23.5.23&type=host
Content-Type: text/plain
OK - load average: 0.36, 0.25, 0.24|load1=0.360;1.000;1.000;0;
load5=0.250;5.000;5.000;0; load15=0.240;15.000;15.000;0;
```

Please take into account that NGSI standard identify entities (in this case, the resources being monitored) using a pair *<entityId,entityType>*. This identification of the monitored resource has to be provided as the query parameters `id` and `type`, respectively. The probe name included in the URL lets NGSI Adapter know the originating monitoring probe, therefore selecting the proper parser for it. This API is fully described in Apiary.

Monitoring framework is expected to schedule the execution of probes and send the raw data been gathered to the NGSI Adapter. Depending on the tool that has been chosen, this would require the development of a custom component (a kind of **monitoring collector**) to automatically forward such data to the adaptation layer.

NGSI Adapter API (UDP)

In case UDP endpoints are defined (specifying the target parser to be loaded), probe raw data should be sent as UDP request to the adapter. Such message is expected to include both the id and the type of the NGSI Entity whose data is about to be parsed.

NGSI Adapter parsers

NGSI Adapter processes requests asynchronously, trying to load a valid parser named after the originating probe, located at any of the directories specified (see *Installation and Administration Guide*). If probe is unknown (parser not found), HTTP response status will be `404`; otherwise, response status will be `200`, parser will be dynamically loaded, and then its `parseRequest()` and `getContextAttrs()` methods will be called. The attribute list returned by the latter will be used to invoke Context Broker.

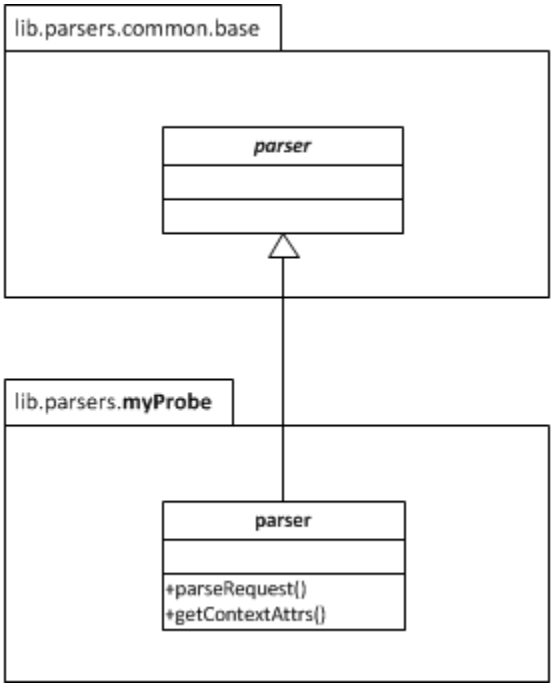
Custom parsers for new probes may be easily added to NGSI Adapter, just extending a base abstract object and implementing the aforementioned methods. For example, suppose we want to support a new "*myProbe*" whose data is a comma-separated list of values of two attributes *myAttr0* and *myAttr1*:

```
/**
 * module "myProbe" at any directory included in ADAPTER_PARSERS_PATH
 */

// @augments base parser (must redefine parseRequest and getContextAttrs)
var myParser = Object.create(null);

// @param Domain object including context, timestamp, id, type & body
myParser.parseRequest = function (reqDomain) {
    var reqDataContent = this.doSomeParsing(reqDomain.body);
    return { data: reqDataContent };
};

// @param EntityData object including data attribute
myParser.getContextAttrs = function (entityData) {
    var items = this.doMoreParsing(entityData.data);
    return { myAttr0: items[0], myAttr1: items[1] };
};
```




```
exports.parser = myParser;
```

Custom parsers for UDP request **must** also set the attributes `entityId` and `entityType` of the input object `reqDomain` on return, given that such information is part of the UDP message itself being parsed:

```
// @param Domain object
myParser.parseRequest = function (reqDomain) {
    var identification = this.doSomeParsing(reqDomain.body),
        reqDataContent = this.doMoreParsing(reqDomain.body);
    reqDomain.entityId   = identification['id'];
    reqDomain.entityType = identification['type'];
    return { data: reqDataContent };
};
```

Context Broker API

Please refer to Context Broker documentation. This will give us access to the last updates of monitoring data available, but not to historical data.

Monitoring API

Retrieval of historical data stored at a distributed filesystem (e.g. Hadoop) is handled by the *Query Manager* component, whose API is described in this preliminary specification.

2.2 Installation & Administration Guide

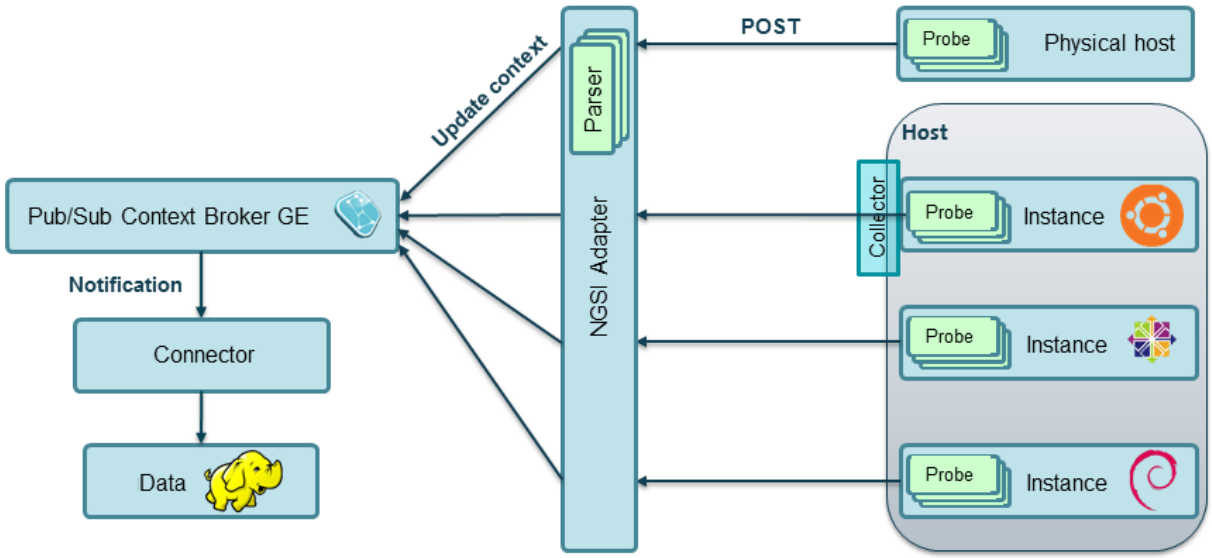
2.2.1 Introduction

This guide defines the procedure to install the different components that build up the Monitoring GE, including its requirements and possible troubleshooting.

For general information, please refer to GitHub's README.

2.2.2 Installation

Monitoring infrastructure comprises several elements distributed across different hosts, as depicted in the following figure:



1. **Probes** gather raw monitoring data, which a **Collector** (for Nagios, this is *NGSI Event Broker*) forwards to *NGSI Adapter*.
2. **NGSI Adapter**, responsible for translating probe raw data into a common format (NGSI).
3. **Parsers** at NGSI Adapter, specific for the different probes that generate monitoring data.
4. **Context Broker**, where monitoring data (transformed into NGSI context updates) will be published.
5. **Hadoop**, for storing historical context data.
6. **Connector** between Context Broker and data storage (for example, this could be *Cygnus*).

Installation of probes

Monitoring GE is agnostic to the framework used to gather monitoring data. It just assumes there are several probes collecting such data, which somehow will be forwarded to the adaptation layer (NGSI Adapter).

It is up to the infrastructure owner which tool (like Nagios, Zabbix, openNMS, perfSONAR, etc.) is installed for this purpose.

Installation of collector

Probes must "publish" their data to NGSI Adapter. Depending on the exact monitoring tool installed, a kind of *collector* has to be deployed in order to send data to the adapter:

- **NGSI Event Broker** is an example specific for Nagios, implemented as a loadable module. Description and installation details can be found here.

Installation of NGSI Adapter

Requirements

NGSI Adapter should work on a variety of operating systems, particularly on the majority of GNU/Linux distributions (e.g. Debian, Ubuntu, CentOS), as it only requires a V8 JavaScript Engine to run a Node.js server.

Hardware Requirements The minimal requirements are:

- RAM: 2 GB

Software Requirements NGSI Adapter is a standalone Node.js process, so `node` and its package manager `npm` should be installed previously. These requirements are automatically checked when installing the `fiware-monitoring-ngsi-adapter` package. However, for manual installation please visit NodeSource.

Downloads

Please refer to this document for details.

Additional parsers

NGSI Adapter currently includes a predefined set of parsers for Nagios probes at `lib/parsers/nagios` directory, each named after its corresponding probe.

This can be extended with additional parsers found at additional directories. To do so, please configure `--parsersPath` command line option (or set the variable `ADAPTER_PARSERS_PATH`) with a colon-separated list of absolute (or relative to Adapter root) directories where parsers are located.

Installation of Context Broker

Please refer to Orion documentation.

Installation of the connector

This component subscribes to changes at Context Broker and writes data into a distributed filesystem storage (usually HDFS from Hadoop). Historically the **ngsi2cosmos** connector implementation has been used (installation details here), although from March 2014 this component is deprecated and a brand new **Cygnus** implementation (installation details here) is available.

2.2.3 Running the monitoring components

As stated before, there are a number of distributed components involved in the monitoring. Please refer to their respective installation manuals for execution details (this applies to probes & monitoring software, Context Broker, Hadoop, etc.). This section focuses on NGSI Adapter specific instructions.

Running NGSI Adapter

Once installed, there are two ways of running NGSI Adapter: manually from the command line or as a system service. It is not recommended to mix both ways (e.g. start it manually but using the service scripts to stop it).

As system service

When installed from its package distribution, a Linux service `ngsi_adapter` is configured (but not started). Please refer to this document for details.

From the command line

You can run the adapter just typing the following command at the installation directory (usually `/opt/fiware/ngsi_adapter/`):

```
$ adapter
```

You can use these command line options (available typing `adapter --help`):

-l, --logLevel=LEVEL Verbosity of log messages

-H, --listenHost=NAME The hostname or address at which NGSI Adapter listens

-p, --listenPort=PORT The port number at which NGSI Adapter listens

-u, --udpEndpoints=LIST Optional list of UDP endpoints (host:port:parser)

-P, --parsersPath=PATH Colon-separated path with directories to look for parsers

-b, --brokerUrl=URL The URL of the Context Broker instance to publish data to

-m, --maxRequests=VALUE Maximum number of simultaneous outgoing requests to Context Broker

-r, --retries=VALUE Number of times a request to Context Broker is retried, in case of error

2.2.4 Sanity check procedures

These are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

End to end testing

Use the commands of the monitoring framework being used (for example, Nagios) to reschedule some probe execution and force the generation of new monitoring data:

- Check the logs of the framework (i.e. `/var/log/nagios/nagios.log`) for a new probe execution detected by the *collector*:

```
$ cat /var/log/nagios/nagios.log
[1439283831] lvl=INFO | trans=rdPmJ/uHE62a |
               comp=fiware-monitoring-ngsi-event-broker | op=NGSIAdapter |
               msg=Request sent to http://host:1337/check_xxx?id=xxx&type=host
```

- Check NGSI Adapter logs for incoming requests with raw data, and for the corresponding updateContext() request to Context Broker:

```
$ cat /var/log/ngsi_adapter/ngsi_adapter.log
time=... | lvl=INFO | trans=rdPmJ/uHE62a | op=POST |
               msg=Request on resource /check_xxx with params id=xxx&type=xxx

time=... | lvl=INFO | trans=rdPmJ/uHE62a | op=POST | msg=Response status 200 OK
time=... | lvl=INFO | trans=rdPmJ/uHE62a | op=UpdateContext |
               msg=Request to ContextBroker at http://host:1026/...
```

- Finally, query Context Broker API to check whether entity attributes have been updated according to the new monitoring data (see details here)

List of Running Processes

A `node` process running the "adapter" server should be up and running, e.g.:

```
$ ps -C node -f | grep adapter
fiware   21930     1  0 Mar28 ?        00:06:06 node /opt/fiware/ngsi_adapter/adapter
```

Alternatively, we can check if service is running, e.g.:

```
$ service ngsi_adapter status
* ngsi_adapter is running
```

Network interfaces Up & Open

NGSI Adapter uses TCP 1337 as default port, although it can be changed using the `--listenPort` command line option.

Additionally, a list of UDP listen ports may be specified by `--udpEndpoints` command line option.

Databases

This component does not persist any data, and no database engine is needed.

2.2.5 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

Resource availability

Although we haven't done yet a precise profiling on NGSI Adapter, tests done in our development and testing environment show that a host with 2 CPU cores and 4 GB RAM is fine to run server.

Remote service access

- Probes at monitored hosts should have access to NGSI Adapter listen port (TCP 1337, by default)
- NGSI Adapter should have access to Context Broker listen port (TCP 1026, by default)
- Connector should have access to Context Broker listen port in order to subscribe to context changes
- Context Broker should have access to Connector callback port to notify changes

Resource consumption

No issues related to resources consumption have been detected neither with the NGSI Adapter server nor with the NGSI Event Broker loaded as a "pluggable" module on Nagios startup.

I/O flows

Figure at *installation section* shows the I/O flows among the different monitoring components:

- Probes send requests to NGSI Adapter with raw monitoring data, by means of a custom *collector* component (for example, NGSI Event Broker)
- NGSI Adapter sends request to Context Broker in terms of context updates of the monitored resources
- Context Broker notifies Connector with every context change
- Connector writes changes to storage

CHAPTER 3

See also

- Monitoring Federation OpenStack Infrastructure presentation summarises the development of this component as a joint task between FIWARE and XIFI projects.