Project Acronym	**Fed4FIRE**
Project Title	**Federation for FIRE**
Instrument	**Large scale integrating project (IP)**
Call identifier	**FP7-ICT-2011-8**
Project number	**318389**
Project website	**www.fed4fire.eu**

D6.3 – Report on first cycle development regarding measuring and monitoring

Work package	WP6
Task	T6.1, T6.2, T6.3
Due date	01/04/2014
Submission date	12/05/2014
Deliverable lead	Yahya Al-Hazmi (TUB)
Version	Final
Authors	Yahya Al-Hazmi (TUB) Wim Vandenberghe (iMinds)
Reviewers	Mikhail Smirnov (Fraunhofer) and Carlos Bermudo (i2CAT)

Abstract	This deliverable reports on the developments of the first cycle regarding monitoring and measurements across the Fed4FIRE federation and used monitoring tools.

Keywords	Measurement, Monitoring, OML, First Level Support.

Nature of the deliverable	R	Report	X
	P	Prototype	
	D	Demonstrator	
	O	Other	
Dissemination level	PU	Public	X
	PP	Restricted to other programme participants (including the Commission)	
	RE	Restricted to a group specified by the consortium (including the Commission)	
	CO	Confidential, only for members of the consortium (including the Commission)	

Disclaimer

The Fed4FIRE project received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no FP7-ICT-318389.

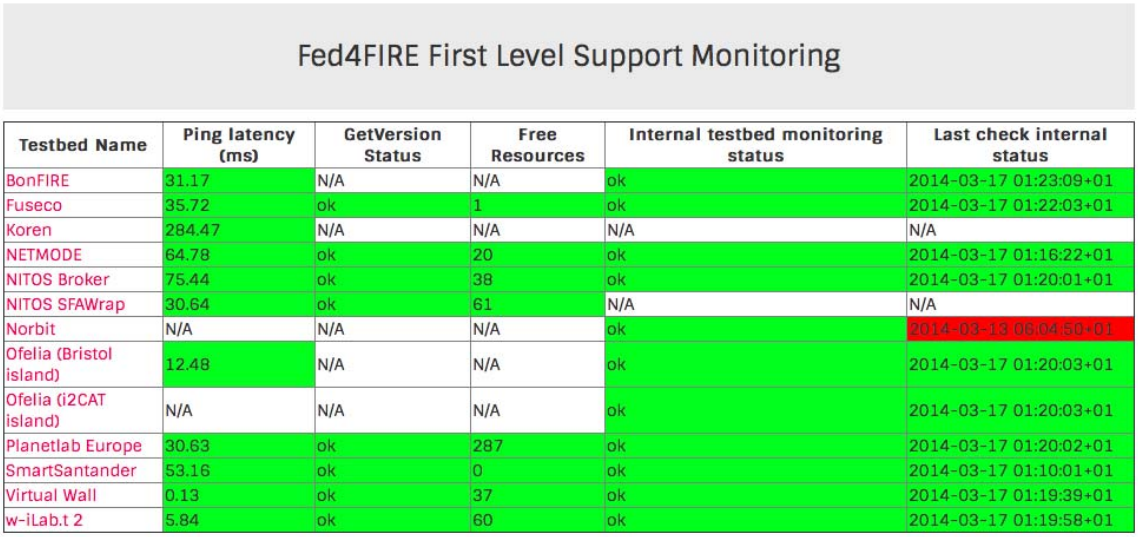
Executive Summary

This deliverable reports on the first cycle development of the Fed4FIRE monitoring and measurement architecture covered by the work package 6. The architecture defined in the D2.1 [3] groups monitoring and measurement services into three main types: facility monitoring, infrastructure monitoring and experiment measuring. The first implementation cycle covers only the first type, namely the facility monitoring, across the federation as it was decided after having the specification for the first cycle ready. Facility monitoring provides a fundamental monitoring service that is of a major concern in federated environments. It provides high level monitoring information about the availability of the testbed facilities as well as their provided resources and services. It allows the federation services as well as end-users to regularly and automatically track whether a testbed belonging to the federation is operational and, on the other hand, users can learn or inquire from which testbed they could request resources and how many of them are still available.

It was decided during the specification phase of the first cycle that the facility monitoring would be implemented using OML [1], a collection and reporting framework that also allows providing the data in a common way. To this end, the testbed should have some means to provide the relevant monitoring data about the concerned metrics transported as OML streams.

However, one of the main architectural principles we followed is that it is not needed by the testbed infrastructures to use special monitoring / measurement solutions to produce the measures. But it is left to them to use whatever they prefer as long as they provide the relevant OML compliant data. Although some testbeds already had solutions in place, the most commonly used, and thus indicated for others are, in the order of preference, Zabbix, Nagios or Collectd. This allowed all Fed4FIRE testbeds to implement facility monitoring in the short timeframe of the first development cycle of the project.

At the federation level, an OML server along with a PostgreSQL database was deployed as a central collecting resource for the facility monitoring service. It is now used by the other components to calculate and show the useful and human readable information to the users through the First Level Support (FLS) monitoring dashboard, as depicted in Figure 1. Through the FLS monitoring dashboard, users can track the overall status of all the individual Fed4FIRE testbeds (whether they are up and running, in risk or down), the availability of resources, the number of free resources, and the last timestamp of the provided information. This was made possible by the facility monitoring design and implementation that took place in cycle 1 of the project.



Fed4FIRE First Level Support Monitoring

Testbed Name	Ping latency (ms)	GetVersion Status	Free Resources	Internal testbed monitoring status	Last check internal status
BonFIRE	31.17	N/A	N/A	ok	2014-03-17 01:23:09+01
Fuseco	35.72	ok	1	ok	2014-03-17 01:22:03+01
Koren	284.47	N/A	N/A	N/A	N/A
NETMODE	64.78	ok	20	ok	2014-03-17 01:16:22+01
NITOS Broker	75.44	ok	38	ok	2014-03-17 01:20:01+01
NITOS SFAWrap	30.64	ok	61	N/A	N/A
Norbit	N/A	N/A	N/A	ok	2014-03-13 06:04:50+01
Ofelia (Bristol island)	12.48	N/A	N/A	ok	2014-03-17 01:20:03+01
Ofelia (i2CAT island)	N/A	N/A	N/A	ok	2014-03-17 01:20:03+01
Planetlab Europe	30.63	ok	287	ok	2014-03-17 01:20:02+01
SmartSantander	53.16	ok	0	ok	2014-03-17 01:10:01+01
Virtual Wall	0.13	ok	37	ok	2014-03-17 01:19:39+01
w-iLab.t 2	5.84	ok	60	ok	2014-03-17 01:19:58+01

Figure 1: Screenshot of the Fed4FIRE First Level Support monitoring dashboard, which provides an overall view on the federation's health status thanks to the facility monitoring work of WP6

Acronyms and Abbreviations

API	Application Programming Interface
AM	Aggregate Manager
DB	Database
FLS	First Level Support
GENI	Global Environment for Network Innovations
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ICMP	Internet Control Message Protocol
FIRE	Future Internet Research and Experimentation
OML	Measurement Library: an instrumentation system allowing for remote collection of any software-produced metrics, with in line filtering and multiple SQL back-ends.
SQL	Structured Query Language
SSH	Secure Shell

Table of Contents

List of Figures

List of Tables

1 Introduction

This deliverable reports on the first cycle development of the monitoring and measurement architecture that was described in the D6.1 “Detailed specifications for first cycle ready” [2]. The architecture covers three main types of monitoring, facility monitoring, infrastructure monitoring and experiment measuring. However, as it was also stated in the D6.1, in the first implementation cycle, only the facility monitoring is implemented. It was supported by all testbeds involved in Fed4FIRE to ensure the first important step in the federation by being Fed4FIRE federation complaint. This deliverable will not provide any specifications but rather reports on what has been implemented and how. It is therefore structured as follows: Section 2 briefly represents requirements from multiple parties in Fed4FIRE that have different needs and concerns. These requirements, which are relevant to facility monitoring, are retrieved from previous Fed4FIRE deliverables. The implementation of the architecture in the first cycle is discussed in Section 3. It includes the used tools and the implementation at the testbed level as well as at the federation level. Furthermore, it shows how the monitoring data is being utilized to provide a first level support to the users. The deliverable is concluded in Section 4 that also sheds light on the current and future work.

2 Input to this deliverable

This section gives a brief summary on requirements from different stakeholders in Fed4FIRE that are relevant to monitoring development in the first cycle.

2.1 Architecture

The monitoring and measurement architecture shown in Figure 2 as identified in D2.1 "First Federation Architecture" [3] includes three types of monitoring: facility monitoring, infrastructure monitoring and experiment measuring. This deliverable focuses only on the facility monitoring that is implemented in the first cycle. This service is mainly about monitoring the availability and health status of the testbeds involved in the Fed4FIRE federation. It is defined by the architecture (D2.1) as follows:

"Facility monitoring: this monitoring is used in the first level support to see if the testbed facilities are still up and running. The most straight forward way for this, is that there is a common distributed tool which monitors each facility (Zabbix [5], Nagios [6] or similar tools). The interface on top of this facility monitoring should be the same and will further be specified in WP6 (it seems in this case more straightforward to use all the same monitoring tool, then to define and implement new interfaces)."

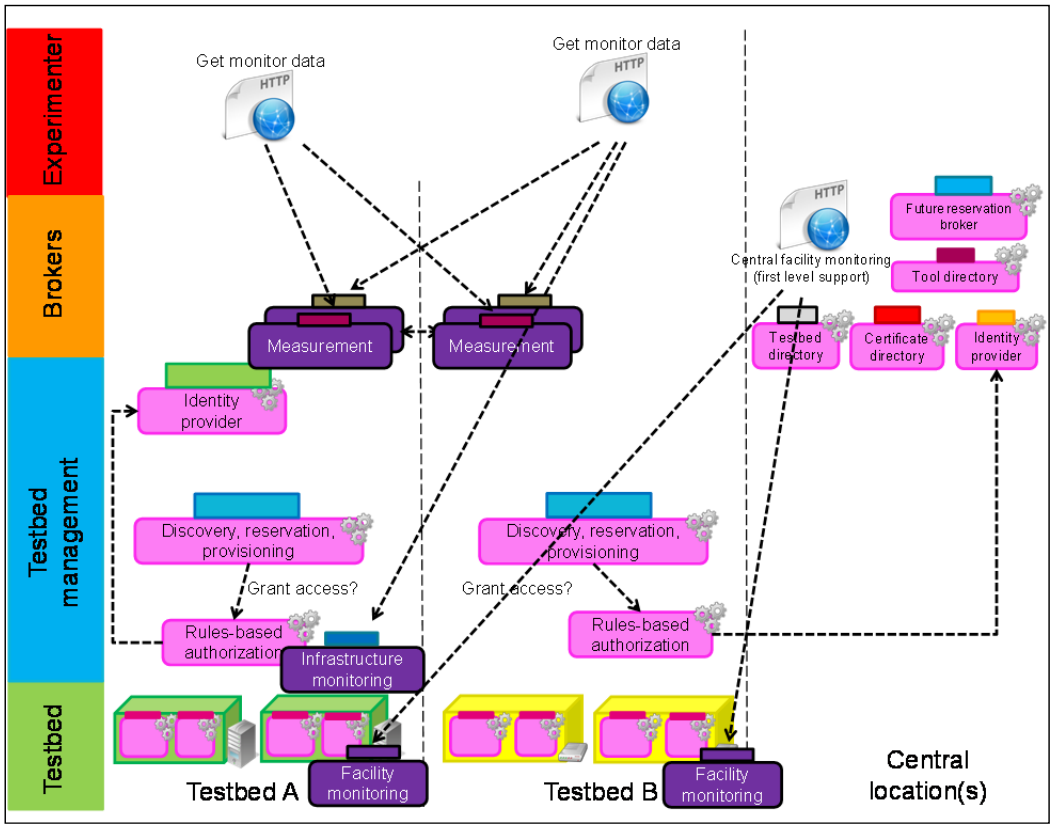


Figure 2: Monitoring and measurement architecture for cycle 1

2.2 High priority requirements of shared support services

This section recalls the requirements relevant to first cycle development of WP6 set forth in D8.1 [4] that is indicated by the FLS.

Table 1 - First Level Support requirements are satisfied by the Facility Monitoring

Req. ID	Description	Remark
FLS.1	Facility monitoring should push RAG (Red, Amber, Green) status to a central dashboard for FLS reactive monitoring	This is achieved at the federation level where a central collection resource receives reported data about the key components from the individual facilities and calculates the overall RAG status of each and finally shows the data through the FLS monitoring dashboard. All testbed facilities already supported.
FLS.2	The Facility RAG status should be based upon the monitoring of key components of each facility that indicate the overall availability status of the facility	Each testbed reports the status of its predefined key components and the overall status is calculated based on specific logic that differs from one testbed to another based on their nature. All testbed facilities already supported.
FLS.3	The FLS should be able to drill down from the facility RAG status to see which components are degraded or down	As each testbed provides monitoring data about the individual key components in a regular basis, the FLS still have the ability to identify those that are not healthy. This is satisfied by all testbeds as well.
FLS.4	The key components monitored for facility monitoring, should be standardised across all facilities as much as possible	Some are standardized such as the AM server, and some not due to the heterogeneity of the testbeds participating in Fed4FIRE.
FLS.5	A commitment is required from each testbed to maintain the quality of monitoring information (FLS is “monitoring the monitoring” and the information FLS has is only as good as the facility monitoring data)	It is supported where each reported data includes their last timestamps, thus it’s clear whether the data is up-to-date or not. The FLS monitoring dashboard notify testbeds owner once there is deviations.
FLS.6	Any central federation-level systems/components that may be implemented will need to be monitored by FLS (e.g. a central directory)	Not yet considered but probably in further releases.
FLS.7	FLS requires visibility of planned outages on a push basis from testbeds and administrators of central systems	This is also supported since we are using OML for reporting that allow providing not only measurement data but also human readable messages that can also be

		considered at the federation level to take proper actions. It is not yet visible to the end-users but can be done in further releases.
FLS.8	Exception alerts from both testbeds and central systems should be filtered prior to reaching the FLS, to avoid reacting to alerts unnecessarily.	This is done through the local monitoring used at the testbed level. To give an example, in FUSECO PG Zabbix is used for facility monitoring and it sends alerts once a component is off or not reachable. This is done immediately after the problem occurs. Admins take care to solve the issue.

2.3 Additional WP6 requirements

We have addressed several requirements in WP6 that are also taken into consideration during the development cycle. Examples include:

- Testbeds that have some local monitoring tools in place could remain using them as long as they fulfil the requirements, while preferable tools are recommended for the others.
- Monitoring data should be provided through a common API across the federation.
- To reduce the complexity and the configuration efforts at the federation level, the testbed should provide the data in a push manner to the central collection resource at the federation. Since otherwise, the collection resource at the federation level would need to know and invoke the individual sources of data at the testbed infrastructures, and has to take care also about their reachability and so on.

2.4 Deviations from specifications in D6.1

In the first development cycle we have met those requirements from the architecture as well as other stakeholders concerning the facility monitoring as well as the specifications defined for the first cycle. Thus, there are no deviations from the D6.1 [2].

3 Implementation of the monitoring

This section presents the implementation steps that have been taken during the first development cycle to realise the facility monitoring service at both the testbed infrastructure and at the federation levels.

In the first cycle, all testbeds implemented support for facility monitoring, which is used to steer the First Level Support dashboard [7]. The detailed specifications of how testbeds can do this were defined after delivery of the first specification deliverable of WP6 (D6.1) [1]. These details as well as the implementation are presented in this section.

3.1 Architecture

Facility monitoring comprises several measurement services that are in charge of realising specific measurements. There are currently five services deployed in both the testbed infrastructure and in the federation. These are indicated in the facility monitoring architecture shown in Figure 3.

In this perspective, the five main services monitored at each testbed facility are as follow:

1. ICMP ping to the AM server or some other testbed server: this checks connectivity over the internet to the testbed. If this fails, testbed can likely not be used.
2. AM API GetVersion call: tests the AM component (no credential is needed).
3. AM API ListResources call: to know the number of free resources. If the returned value is 0, then new experiments cannot be created.
4. Green/Amber/Red aggregation state of what the testbed provider monitors itself. This is custom per testbed. Each testbed provider identifies a list of key components / resources, that should be monitored and based on their status the overall status of the testbed is calculated as mentioned in Table **1**. The testbed provider is responsible for measuring and reporting their status. These differ from one testbed to another depending on the testbed type and nature.
5. Last timestamp of the information monitored at the testbed itself and injected into the OML server: if this timestamp is too old, the testbed monitor doesn't provide any information anymore.

The implemented architecture supports live visualization at the Fed4FIRE FLS monitoring dashboard [7] and provides email alarms and long term statistics helping to dive into the monitoring information of the past.

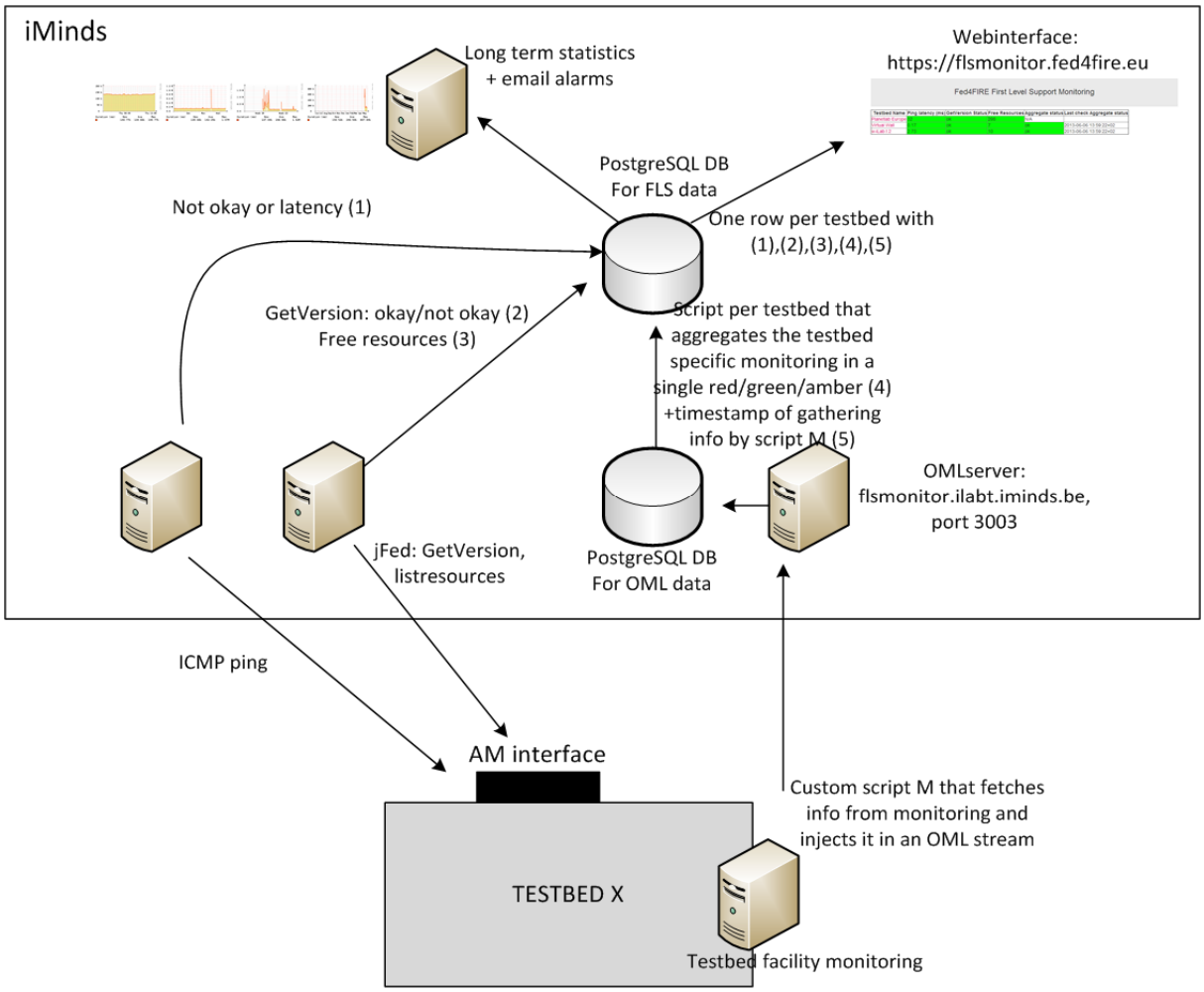


Figure 3: Facility Monitoring Architecture

By browsing the Fed4FIRE FLS monitoring dashboard [7]) the user will have the ability to see the health status about the individual testbeds involved in Fed4FIRE federation.

This dashboard shows the results of the aforementioned five measurement services that are regularly monitored. The status of a testbed is indicated and the user can track the results reported by these services individually. Different reported results visualised by the screenshots of the FLS monitoring dashboard are discussed below.

Figure 4 shows the status of three testbeds where everything goes well. That means that these testbeds are fully healthy.

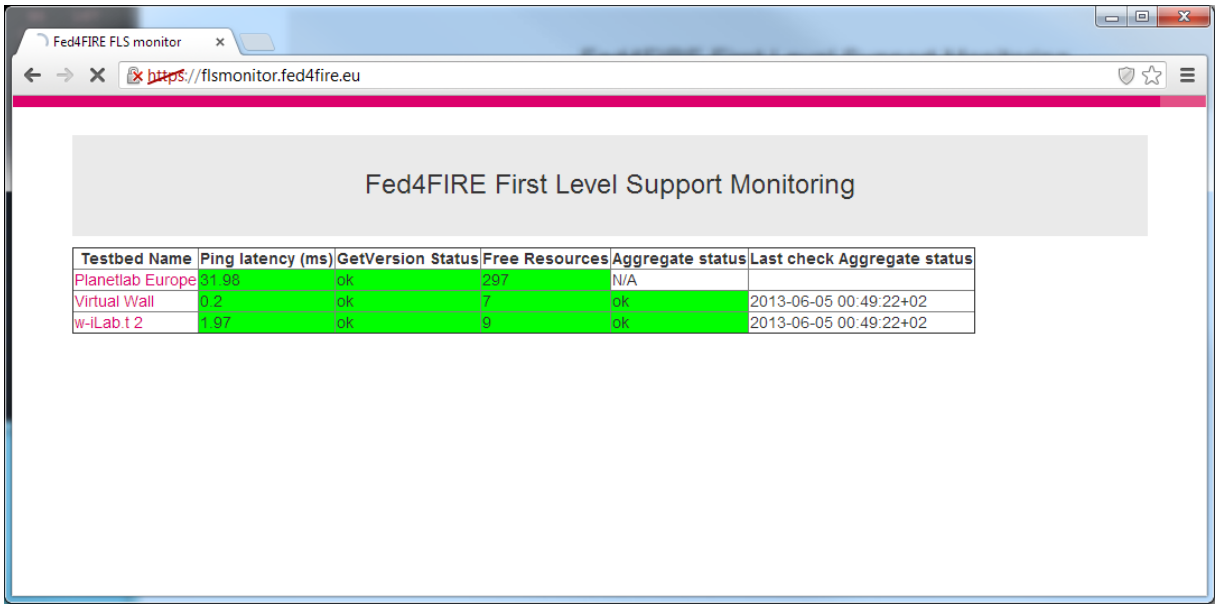


Figure 4: Fed4FIRE FLS Monitoring (where everything goes well)

Figure 5 represents the case in which the testbed "W-iLab.t 2" goes offline. The network is still okay (ping is okay), but the AM is not reachable and the testbed internal monitoring says that it is partially up and running. However, Figure 6 shows another case in which the same testbed goes fully offline and the internal monitoring says that it is down. The difference between both is that in the former case some components are up and running and some others are down, while in the later all the testbed components are down.

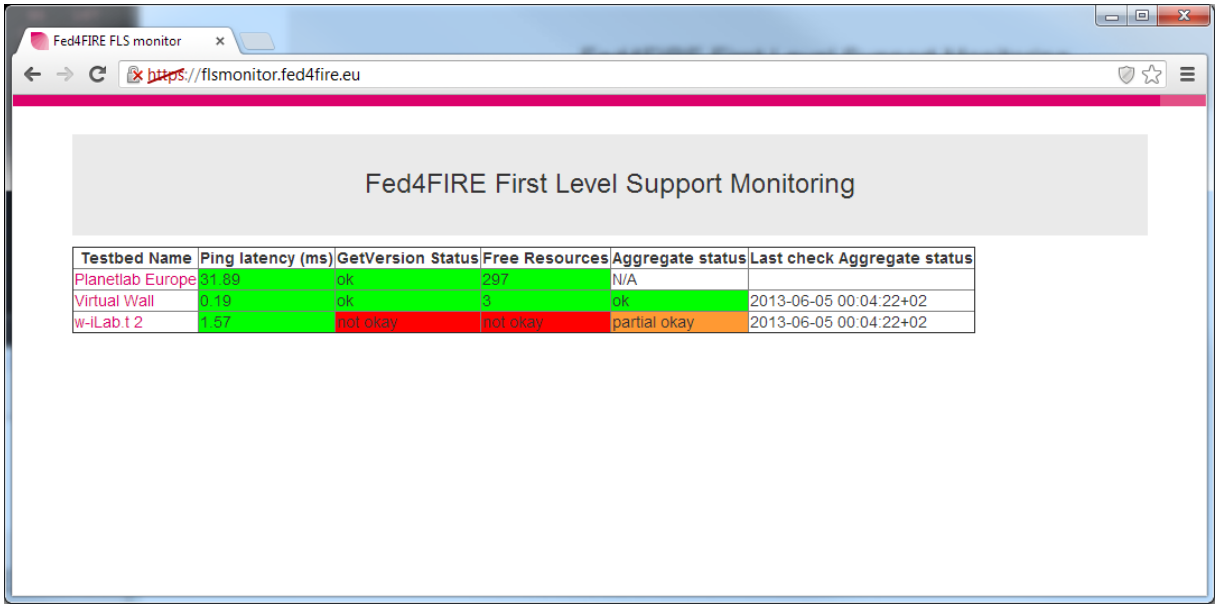


Figure 5 - Fed4FIRE FLS Monitoring (where a testbed goes offline)

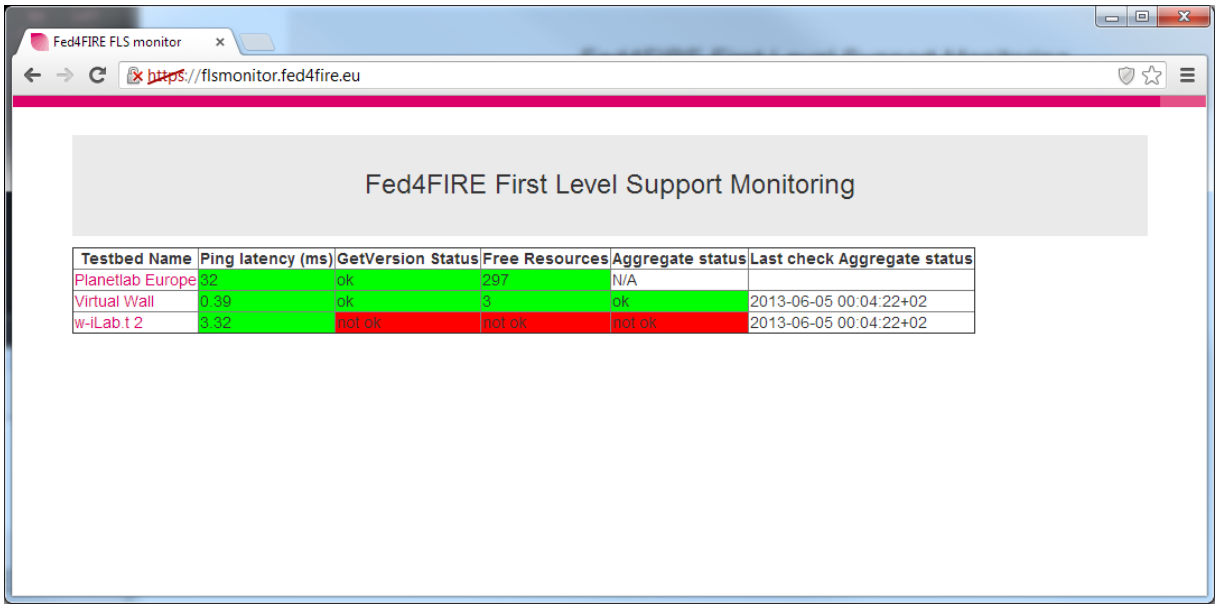


Figure 6 - Fed4FIRE FLS Monitoring (where a testbed goes fully offline)

Another possible result is shown in Figure 7 where the "Virtual Wall" testbed shows zero free resources. An experimenter might not have noticed this and call FLS (amber state).

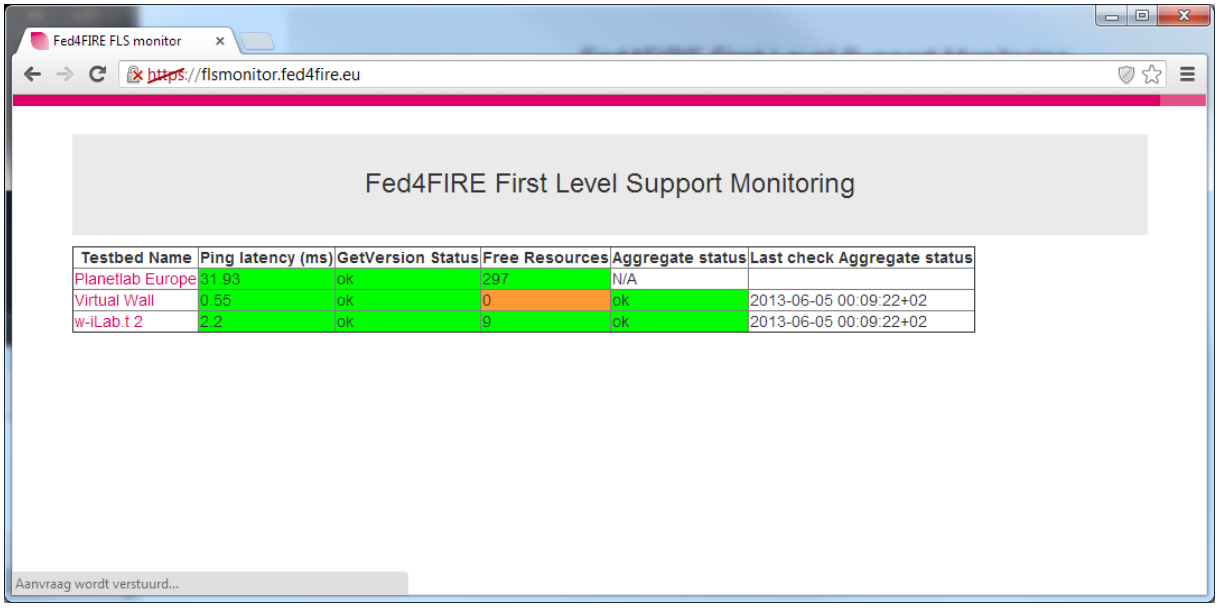


Figure 7 - Fed4FIRE FLS Monitoring (where zero free resources are shown)

3.2 Implementation at the testbed level

This section reports on the used tools as well as implementations undertaken at the testbed infrastructure level.

According to the aforementioned five measurement services, the first three are managed by a component at the federation level (discussed in Section 3.3), while in this section we present how the other two services (Green/Amber/Red aggregation state and last timestamp of monitoring information) are supported at the testbed level.

3.2.1 Used measurement tools

The specifications reported in D6.1 [2] include also the selected tools to be used in the first cycle. Accordingly, most of the testbeds have used Zabbix [5] or Nagios [6] for facility monitoring purpose. It was mandatory to use OML for data collection and reporting to provide the monitoring data in a common way across the testbeds.

The installations of these are out of the scope of this deliverable but to be found at their corresponding documentation websites (Zabbix [5], Nagios [6], and OML [8]).

3.2.2 Publishing monitoring data

Using any of the aforementioned tools to monitor the testbed facility, the monitoring data about the key components is available locally. It is then reported by each testbed compliant with OML (also as OML streams) to the central collection resource at the federation level.

There are several reference implementations in Fed4FIRE, at least one per testbed on how to provide monitoring data as OML streams. These are not presented or discussed here but rather we shed light on how this implementation has been undertaken in a form of a very simple example with only one single measured metric.

In the following we assume that the local monitoring tool used is Zabbix. We show an example on how to query a Zabbix server through the Zabbix API and insert the results into an OML database.

This example in a form of a Ruby script that can be run on any machine, as long as it has connectivity to both Zabbix and OML server. It should be run by the testbed provider.

Pre-conditions and Requirements:

- Install guide :
    - Ruby packages: ruby1.9.1, ruby1.9.1-dev, rubygems
    - OML client library: liboml2 (detailed instructions at [8])
    - Rubygems: zabbixapi, oml4r (install with 'gem install <gemName>')

- Fed4Fire FLS OML Server details :
  - Version: OML 2.10 with PostgreSQL back-end
  - Connection: flsmonitor.ilabt.iminds.be port 3003
- The example script in "Appendix A: Oml4r-zabbix.rb for monitoring ssh: to be adapted and run by testbed provider" (original template from [9]) shows how to query the Zabbix API to check if the SSH service is running.
  - In the oml_opts hash, please change the :domain and :nodeID values to reflect your own testbed.
  - Change the zabbix_opts to connect to your own Zabbix server.
  - Define your own measurement point. An example measurement point is given to check the SSH service. Multiple measurement points can be defined (e.g. ICMP, HTTP, SSH).
  - Add more Zabbix queries if needed. Change the :name value to query different measurement data from the Zabbix server.
  - Send the Zabbix data to the monitoring server by calling <mpName>.inject(<dbFields>).
  - Add the host names as command line arguments and start the script (-i specifies the query interval :

ruby oml4r-zabbix.rb <hostname1 hostname2 …>

This example script can be extended in order to report more than a single metric. As mentioned above there are several other implementations that are up and running in the Fed4FIRE testbeds.

3.3 Implementation at the federation level

The first three of the five measurement services presented in Section 3.1 are managed only at the federation level without any extra efforts required from the testbeds.

A component at the federation level is in charge of realising these measurement services. It periodically checks the reachability of a testbed through sending ICMP ping to the AM server or some other testbed server. The second and third measurements are done by invoking the AM API (AM API GetVersion call and AM API ListResources call) that must exist in the testbed infrastructure supporting SFA across the federation. The technology used to drive these SFA calls from this server side component is jFed[1] probe, part of the jFed software framework that was especially developed within Fed4FIRE (and partially within WP6) to support testbed federation [10]. There is no need for extra efforts from the testbeds to support these three measurements.

As aforementioned the first three measurement services do not require extra implementation efforts from the testbeds, while the last two (namely the Green/Amber/Red aggregation state and last timestamp of monitoring information) still require implementation efforts at both levels. Support at

[1] http://jfed.iminds.be/

the testbed level is already discussed in Section 3.2, while the following section reports on how the reported data from the individual testbeds are processed.

3.3.1 Details on aggregating the status of the internal testbed monitoring

A testbed periodically reports measurement data about its individual key components as OML streams. An OML stream includes specific information about a component being monitored and values being reported. This means if a testbed reports monitoring information about 5 key components, each push should include 5 OML streams. Based on these, the overall status of the testbed is calculated according to a predefined and specific logic. This logic differs from testbed to another depending on their natures and types.

Therefore, there are several implementations, but in the following through a simple example we explain how the aggregation status is generated.

The script provided in the "Appendix B: Aggregation script: example for virtual wall, script run by iMinds based on information provided by testbed provider" (in Ruby) is used as an example implementation to calculate the aggregated testbed status. In this example script is for the Virtual Wall; the script first checks if the three core servers of the Virtual Wall can be pinged, after which it will check if it is possible to SSH to the user server and verify if the webserver of the testbed is up and running.
The following states can be shown:

- 0 (green): everything is OK, testbed is up and running.
- 1 (amber): some servers or switches cannot be reached, but part of the testbed may still be up.
- 2 (red): one or more of the core servers/switches is down, the testbed cannot be used.

The script updates the aggregate status on the FLS monitoring database every minute and is run by iMinds at the FLS servers (one script per testbed which provides monitoring information over OML).

4 Conclusion and Future Work

This deliverable reports on the first development cycle of Fed4FIRE regarding the measurement and monitoring services. Following the specifications reported in D6.1 [2] for the first cycle implementation, the facility monitoring (which is one of the main three monitoring types identified in Fed4FIRE monitoring and measurement architecture) has been implemented and rolled out as it was also stated in D6.1.

In this deliverable we have shown how the facility monitoring has been implemented. It explains the implementation steps undertaken in both the testbed infrastructure and federation levels. This discussion is supported with simple implementation examples. Starting from such examples, several advanced implementations that are specific to each testbed in Fed4FIRE have been made available. In fact, every testbed of the Fed4FIRE federation currently supports facility monitoring as described in this deliverable[2].

We have shown how the facility monitoring has been implemented through five main measurement services that provide high level informative results through the Fed4FIRE monitoring dashboard [7]. Users can track the overall status of the individual testbeds (whether they are up and running, in risk, or down), the availability of resources, the number of free resources, and the last timestamp of the provided information.

In the second implementation cycle we implement the other two monitoring services, namely the infrastructure monitoring and experiment measuring. This is ongoing work that will be continued in cycle 2, and will take the specifications and implementation plan reported in D6.2 “Detailed specifications regarding monitoring and measurement for second cycle” into account.

[2] Except the testbeds that only joined the project after the first open call, since they only joined near the end of the first cycle on which we report in this deliverable.

References

[1] O. Mehani, G. Jourjon, T. Rakotoarivelo, and M. Ott, "An instrumentation framework for the critical task of measurement collection in the future Internet," Computer Networks, vol. 63, pp. 68-83, Apr. 2014.

[2] Fed4FIRE D6.1 Detailed specifications for first cycle ready. Available online at: http://www.fed4fire.eu/fileadmin/documents/public_deliverables/D6-1_Fed4FIRE_Detailed_specifications_for_first_cycle_ready.pdf.

[3] Fed4FIRE D2.1 First Federation Architecture. Available online at: http://www.fed4fire.eu/fileadmin/documents/public_deliverables/D2-1_Fed4FIRE_First_federation_architecture.pdf.

[4] Fed4FIRE D8.1 First input to WP2 concerning First Level Support. Available online at: http://www.fed4fire.eu/publications/deliverables.html

[5] Zabbix – open source monitoring system. Available online at www.zabbix.com, last accessed on May 7, 2014.

[6] Nagios: systems and network monitoring; 2nd ed., by Wolfgang Barth, San Francisco: No Startch Press (2008).

[7] Fed4FIRE First Level Support Dashboard. Available online at: https://flsmonitor.fed4fire.eu.

[8] OML WiKi. Available online at http://oml.mytestbed.net/projects/oml/wiki/Installation, last accessed on May 7, 2014.

[9] OML wrapper for Ruby. Available online at: https://github.com/mytestbed/oml4r/blob/master/examples/oml4r-zabbix.rb, last accessed on May 7, 2014.

[10]jFed: Java based framework to support SFA testbed federation client tools. Available online at: http://jfed.iminds.be, last accessed on May 8, 2014.

Appendix A: Oml4r-zabbix.rb for monitoring ssh: to be adapted and run by testbed provider

```
#!/usr/bin/env ruby
# example script for FLS Testbed monitoring with Zabbix
# make sure you install these gems

require "zabbixapi"
require "oml4r"
require "date"

# # Zabbix node names
# nodes = ["10.129.16.11", "10.129.16.12", "10.129.16.13"]

# Define your own Measurement Point
class SSH_MP < OML4R::MPBase
  name :ssh
  param :insert_time, :type => :string
  param :node, :type => :string
  param :up, :type => :double
  param :last_check, :type => :string
end

# Initialise the OML4R module for your application
oml_opts = {
  :appName => 'zabbix',
  :domain => 'iMinds',
  :nodeID => 'iMindsTestbeds',
  :collect => 'tcp:flsmonitor.ilabt.iminds.be:3003'
}

zabbix_opts = {
  :url => 'https://xxxxx/api_jsonrpc.php',
  :user => 'xxxxx',
  :password => 'xxxxx'
}

interval = 10

nodes = OML4R::init(ARGV, oml_opts) do |op|
  op.banner = "Usage: #{$0} [options] host1 host2 ...\n"
  op.on( '-i', '--interval SEC', "Query interval in seconds [#{interval}]"
) do |i|
    interval = i.to_i
  end
  op.on(    '-s',    '--service-url    URL',    "Zabbix    service    url
[#{zabbix_opts[:url]}]" ) do |u|
    zabbix_opts[:url] = p
  end
```

```
  op.on(      '-p',      '--password      PW',      "Zabbix      password
[#{zabbix_opts[:password]}]" ) do |p|
    zabbix_opts[:password] = p
  end

  op.on( '-u', '--user USER', "Zabbix user name [#{zabbix_opts[:user]}]" )
do |u|
    zabbix_opts[:user] = u
  end
end

if nodes.empty?
  OML4R.logger.error "Missing host list"
  OML4R::close()
  exit(-1)
end



# connect to Zabbix JSON API
zbx = ZabbixApi.connect(zabbix_opts)
# catch CTRL-C
exit_requested = false
Kernel.trap( "INT" ) { exit_requested = true }

# poll Zabbix API
while !exit_requested
  nodes.each{|n|
    results = zbx.query(
      :method => "item.get",
      :params => {

        :output => "extend",
        :host => "#{n}",
        # only interested in SSH
        :search => {
          :name => "SSH service is running"
        }
      }
    )

    unless results.empty?
      up = results[0]["lastvalue"]
      # injecting measurements into OML
      SSH_MP.inject(Time.now.to_s,                    n,                    up,
Time.at(results[0]["lastclock"].to_i).to_s)
    else
      OML4R.logger.warn   "Empty   result   usually   means   misspelled   host
address"
    end
  }
```

```
  sleep interval
end


OML4R::close()
puts "Exiting"
```

Appendix B: Aggregation script: example for virtual wall, script run by iMinds based on information provided by testbed provider

```
#!/usr/bin/ruby
require "pg"
#open DB connections
flsmon = PGconn.open('dbname=flsmonitoring')
#query iMinds testbeds
iminds = PGconn.open('dbname=iMinds')

#wall3 - aggregated testbed info
res = iminds.exec("select node,up,max(last_check) as last_check from
zabbix_icmp where node='boss.wall3.test.ibbt.be' or
node='ops.wall3.test.ibbt.be' \
                or node='fs.wall3.test.ibbt.be' group by node,up ;")

$up=res[0]['up'].to_i==1 && res[1]['up'].to_i==1 && res[2]['up'].to_i==1

#only if ping tests succeed -> test SSH and HTTP
if $up
  sshres = iminds.exec("select up from zabbix_ssh where
node='ops.wall3.test.ibbt.be' order by insert_time desc limit 1;")
  httpres = iminds.exec("select up from zabbix_http where
node='boss.wall3.test.ibbt.be' order by insert_time desc limit 1;")
  $up = $up && sshres[0]['up'].to_i && httpres[0]['up'].to_i
end

if $up==1 then $result=0 else $result=2 end
#if something failed, insert 2 (=testbed is not usable!)
update = flsmon.exec("update flstestbeds set
aggregatetestbedstate=#{$result} , last_check='#{res[0]['last_check']}'
where testbedid=1 ; ")
iminds.close()
flsmon.close()
```