FIWARE-Bosun: Bosun

Release

February 13, 2017

Contents

CHAPTER 1

Introduction

Bosun is the reference implementation (GEri) of FIWARE Policy Manager GE, and its component Cloto provides a REST API to create rules associated to servers, subscribe to Context Broker to get information about resources consumption of those servers, and launch actions described in rules when conditions are met.

Policy Manager provides the basic management of cloud resources based on rules, as well as management of the corresponding resources within the FIWARE Cloud instance like actions based on physical monitoring or infrastructure, security monitoring of resources and services or whatever that could be defined by facts, actions and rules. Policy Manager is a easy rule engine designed to be used in the OpenStack ecosystem and, of course, inside the FIWARE Cloud.

IMPORTANT NOTE: This GE reference implementation product is only of interest to potential FIWARE instance providers and therefore has been used to build the basic FIWARE platform core infrastructure of FIWARE Lab. If you are an application developer, you don't need to create a complete FIWARE instance locally in order to start building applications based on FIWARE. You may rely on instances of FIWARE GEris linked to the Data/Media Context Management, the IoT Services Enablement and the Advanced Web-based User Interface chapters, or some GEris of the Applications/Services Ecosystem and Delivery Framework chapter (WireCloud) as well as the Security chapter (Access Control). Those instances are either global instances or instances you can create on FIWARE Lab, but also instances you may create by downloading, installing and configuring the corresponding software in your own premises.

Bosun Policy Manager REST API 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 Programmer Guide for the Policy Manager Generic Enabler. The online documents are being continuously updated and improved, and so will be the most appropriate place to get the most up to date information on using this interface.

Please go to GitHub's README for more documentation.

Background and Detail

This User and Programmers Guide relates to the Policy 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

The Policy Manager GE is a backend component, without user interface. Therefore there is no need to provide a user guide. The Cloud Portal can be used for Web-based interaction (but it is not part of this GE).

2.1.3 Programmer Guide

Policy Manager API is based upon HTTP and therefore all devices, which can handle HTTP traffic, are possible clients.

Accessing Policy Manager from the CLI

To invoke the REST API use the curl program. Curl [1] is a client to get documents/files from or send documents to a server, using any of the supported protocols (HTTP, HTTPS, FTP, LDAP, FILE, etc.) and therefore is also suitable for Policy Manager API. Use either the curl command line tool or libcurl from within your own programs in C. Curl is free and open software that compiles and runs under a wide variety of operating systems.

In order to make a probe of the different functionalities related to the Policy Manager, we make a list of several operations to make a probe of the execution of these GEis.

1. Get a valid token for the tenant that we have (It is not a Policy Manager operation but a IdM operation).

Due to all operations of the Policy Manager are using the security mechanism which is used in the rest of the cloud component, it is needed to provide a security token in order to continue with the rest of operations.

```
curl -d '{"auth": {"tenantName": $TENANT, "passwordCredentials":{"username": $USERNAME,
"password": $PASSWORD}}}' -H "Content-type: application/json"
-H "Accept: application/xml"  http://$KEYSTONE_HOST:$KEYSTONE_PORT/v2.0/tokens
```

Both $TENANT (Project), $USERNAME and $PASSWORD must be values previously created in the OpenStack Keystone. The IP address $KEYSTONE_HOST and the Port $KEYSTONE_PORT are the data of our internal installation of IdM, if you planned to execute it you must changed it by the corresponding IP and Port of the FIWARE Keystone or IdM IP and Port addresses.

We obtained two data from the previous sentence:

- X-Auth-Token

```
<token expires="2012-10-25T16:35:42Z" id="a9a861db6276414094bc1567f664084d">
```

- Tenant-Id

```
<tenant enabled="true" id="c907498615b7456a9513500fe24101e0" name=$TENANT>
```

2. Get tenant information

This is the first real operation about our GEi, by which we can obtain the information about the Policy Manager, together with the information about the window size fixed for the execution of the GEi. For more information about the window size and its meaning.

```
curl -v -H 'X-Auth-Token: a9a861db6276414094bc1567f664084d'
-X GET http://<RULE ENGINE HOST>:8000/v1.0/c907498615b7456a9513500fe24101e0
```

This operation will return the information regarding the tenant details of the execution of the Policy Manager

```
< HTTP/1.0 200 OK
< Date: Wed, 09 Apr 2014 08:25:17 GMT
< Server: WSGIServer/0.1 Python/2.6.6
< Content-Type: text/html; charset=utf-8
{
    "owner": "Telefonica I+D",
    "doc": "http://docs.policymanager.apiary.io",
    "runningfrom": "14/04/09 07:45:22",
    "version": 1.0,
    "windowsize": 10
}
```

3. Create a rule for a server

This operation allows to create a specific rule associate to a server:

```
curl -v -H 'X-Auth-Token: 86e096cd4de5490296fd647e21b7f0b4'
-X POST http://130.206.81.71:8000/v1.0/6571e3422ad84f7d828ce2f30373b3d4/servers
/32c23ac4-230d-42b6-81f2-db9bd7e5b790/rules/
-d '{"action": {"actionName": "notify-scale", "operation": "scaleUp"}, "name": "ScaleUpRule",
"condition": { "cpu": { "value": 98, "operand": "greater" },
"mem": { "value": 95, "operand": "greater equal"}}}'
```

The result of this operation is the following content:

```
< HTTP/1.0 200 OK
< Date: Wed, 09 Apr 2014 10:14:11 GMT
< Server: WSGIServer/0.1 Python/2.6.6
< Content-Type: text/html; charset=utf-8
{
    "serverId": "32c23ac4-230d-42b6-81f2-db9bd7e5b790",
    "ruleId": "68edb416-bfc6-11e3-a8b9-fa163e202949"
}
```

4. Subscribe the server to the rule

Through this operation we can subscribe a rule to be monitored in order to evaluate the rule to be processed.

```
curl -v -H 'X-Auth-Token: a9a861db6276414094bc1567f664084d'
-X POST http://130.206.81.71:8000/v1.0/6571e3422ad84f7d828ce2f30373b3d4/servers
/32c23ac4-230d-42b6-81f2-db9bd7e5b790/subscription
-d '{ "ruleId": "ruleid", "url": "URL to notify any action" }'
```

An the expected result is the following.

```
< HTTP/1.0 200 OK
< Date: Wed, 09 Apr 2014 10:16:11 GMT
< Server: WSGIServer/0.1 Python/2.6.6
< Content-Type: text/html; charset=utf-8
{
    "serverId": "32c23ac4-230d-42b6-81f2-db9bd7e5b790",
    "subscriptionId": "6f231936-bfce-11e3-9a13-fa163e202949"
}
```

5. Manual simulation of data transmission to the server

This operation simulate the operation that the context broker used to send data to the Policy Manager, the normal execution of this process will be automatically once that the Policy Manager subscribes a rule to a specific server. The operation is related to fiware-facts component and it has the following appearance:

```
curl -v -H "Content-Type: application/json"
-X POST http://127.0.0.1:5000/v1.0/6571e3422ad84f7d828ce2f30373b3d4/servers/serverI1
-d '{
"contextResponses": [
    {
        "contextElement": {
           "attributes": [
               {
                   "value": "0.12",
                   "name": "usedMemPct",
                   "type": "string"
               },
               {
                   "value": "0.14",
                   "name": "cpuLoadPct",
                   "type": "string"
```

```
                },
                {
                    "value": "0.856240",
                    "name": "freeSpacePct",
                    "type": "string"
                },
                {
                    "value": "0.8122",
                    "name": "netLoadPct",
                    "type": "string"
                }
            ],
            "id": "Trento:193.205.211.69",
            "isPattern": "false",
            "type": "host"
        },
        "statusCode": {
            "code": "200",
            "reasonPhrase": "OK"
        }
    }]
}'
```

Which produces the following result after the execution:

```
* About to connect() to 127.0.0.1 port 5000 (#0)
*   Trying 127.0.0.1...
* Adding handle: conn: 0x7fa2e2804000
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7fa2e2804000) send_pipe: 1, recv_pipe: 0
* Connected to 127.0.0.1 (127.0.0.1) port 5000 (#0)
> POST /v1.0/33/servers/44 HTTP/1.1
> User-Agent: curl/7.30.0
> Host: 127.0.0.1:5000
> Accept: */*
> Content-Type: application/json
> Content-Length: 1110
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 0
< Date: Wed, 09 Apr 2014 00:11:49 GMT
<
* Connection #0 to host 127.0.0.1 left intact
```

6. Unsubscribe the previous rule

In order to stop the process to evaluate rules, it is needed to unsubscribe the activated rule. We can do it with the following operation:

```
curl -v -H 'X-Auth-Token: a9a861db6276414094bc1567f664084d'
-X DELETE http://130.206.81.71:8000/v1.0/6571e3422ad84f7d828ce2f30373b3d4/servers
/serverI1/subscription/SubscriptionId
```

```
< HTTP/1.0 200 OK
< Date: Wed, 09 Apr 2014 10:16:59 GMT
< Server: WSGIServer/0.1 Python/2.6.6
< Content-Type: text/html; charset=utf-8
```

Accessing Policy Manager from a browser

To send HTTP requests to Policy Manager using a browser, you may use:

- Chrome browser [2] with the Simple REST Client plugin [3]
- Firefox RESTClient add-on [4].

2.2 Installation & Administration Guide

2.2.1 Policy Manager Installation

This guide tries to define the procedure to install the Policy Manager in a machine, including its requirements and possible troubleshooting that we could find during the installation. We have to talk about two applications deployed in a Django server.

Please go to GitHub's README for more documentation.

Requirements

In order to execute the Policy Manager, it is needed to have previously installed the following software of framework in the machine:

- Rule engine dependencies:
  - Python 2.7.6 [1]
  - RabbitMQ 3.3.0 [2]
  - MySQL 5.6.14 or above [3]
- Facts engine dependencies:
  - Python 2.7.6 [1]
  - Redis 2.8.8 [4]

Rule engine installation

There is no need to configure any special options in Django server. Run as default mode.

Step 1: Install Python

If you do not have Python installed by default, please, follow instructions for your Operating System in the official page: https://www.python.org/download/releases/2.7.6/

Step 2: Install RabbitMQ

To install RabbitMQ Server, it is better to refer official installation page and follow instructions for the Operating System you use: http://www.rabbitmq.com/download.html

After installation, you should start RabbitMQ. Note that you only need one instance of RabbitMQ and It could be installed in a different server than fiware-facts or Rule Engine.

Step 3: Install MySQL

To install MySQL Server, it is better to refer official installation page and follow instructions for the Operating System you use: http://dev.mysql.com/downloads/mysql/

You will need four packages:

```
mysql-server
mysql-client
mysql-shared
mysql-devel
```

After installation, you should create a user, create database called ‘cloto’ and give all privileges to the user for this database. The name of that database could be different but should be configured in the config file of fiware-facts and fiware-cloto.

To add a user to the server, please follow official documentation: http://dev.mysql.com/doc/refman/5.5/en/adding-users.html

Step 4: Download and execute the Rule Engine server

1. Installing fiware-cloto

Install the component by executing the following instruction:

```
sudo pip install fiware-cloto
```

It should show something like the following:

```
Installing collected packages: fiware-cloto
    Running setup.py install for fiware-cloto
Successfully installed fiware-cloto
Cleaning up...
```

2. Configuring Rule engine

Before starting the rule engine, you should edit settings file and add it to the default folder located in `/etc/fiware.d/fiware-cloto.cfg`

In addition, user could have a copy of this file in other location and pass its location to the server in running execution defining an environment variable called CLOTO_SETTINGS_FILE.

You can find the reference file here. You should copy this file into default folder and complete all empty keys.

```
[openstack]
# OPENSTACK information about KEYSTONE to validate tokens received
OPENSTACK_URL: http://cloud.lab.fi-ware.org:4731/v2.0
ADM_USER:
ADM_PASS:
ADM_TENANT_ID:
ADM_TENANT_NAME:
USER_DOMAIN_NAME: Default
AUTH_API: v2.0

[policy_manager]
SECURITY_LEVEL: LOW
SETTINGS_TYPE: production
DEFAULT_WINDOW_SIZE: 5
MAX_WINDOW_SIZE: 10
LOGGING_PATH: /var/log/fiware-cloto

[context_broker]
CONTEXT_BROKER_URL: http://130.206.115.92:1026/v1
# Public IP of fiware-facts module
NOTIFICATION_URL: http://127.0.0.1:5000/v1.0
NOTIFICATION_TYPE: ONTIMEINTERVAL
NOTIFICATION_TIME: PT5S

[rabbitmq]
# URL Where RabbitMQ is listening (no port needed, it uses default port)
RABBITMQ_URL: localhost

[mysql]
DB_CHARSET: utf8
DB_HOST: localhost
DB_NAME: cloto
DB_USER:
DB_PASSWD:
```

```
[django]
DEBUG: False
DATABASE_ENGINE: django.db.backends.mysql
ALLOWED_HOSTS: ['127.0.0.1', 'localhost']
### Must be a unique generated value. keep that key safe.
SECRET_KEY: TestingKey+faeogfjksrjgpjaspigjiopsjgvopjsopgvj

[logging]
level: INFO
```

You should also modify `ALLOWED_HOSTS` parameter adding the hosts you want to be accesible from outside, your IP address, the domain name, etc. An example could be like this:

```
ALLOWED_HOSTS: ['127.0.0.1', 'localhost', 'policymanager.host.com','80.71.123.2’]
```

Finally, ensure that folder for logs (`/var/log/fiware-cloto/` by default) has the right permissions and owner.

In 2.5.0 release we added a new parameter called `SECURITY_LEVEL`. This parameter could have three values: `[HIGH | MEDIUM | LOW]` Depending of API version it will store user tokens in memory assuming that a token will be valid for a time period. After this expiration time, token is going to be verified with against keystone.

```
Using v3:
 LOW: user token should be verified after 1h
 MEDIUM: User token should be verified after 30min
 HIGH: user token should be verified after each request

Using v2.0:
 LOW: user tokens should be verified after 24h
 MEDIUM: user token should be verified after 6h
 HIGH: user token should be verified after each request
```

3. Starting the server

To run fiware-cloto, just execute:

```
$ gunicorn fiware_cloto.cloto.wsgi -b BIND_ADDRESS
```

Where BIND_ADDRESS is a valid network interface assigned with a public address. If you execute the command with `127.0.0.1` fiware-cloto won’t be accessible from outside.

To stop fiware-cloto, you can stop gunicorn server, or kill it

NOTE: to enable writing gunicorn log messages to console, please add the option `--log-file=-`; otherwise, if you prefer to write them into a file, just add `--log-file=<log file name>`.

Facts installation

Step 1: Install python

The process will be the same that be see in the previous section.

Step 2: Install Redis

Download, extract and compile Redis with:

```
$ wget http://download.redis.io/releases/redis-2.8.8.tar.gz
$ tar xzf redis-2.8.8.tar.gz
$ cd redis-2.8.8
$ make
```

The binaries that are now compiled are available in the src directory. Run Redis with:

```
$ src/redis-server
```

It execute the redis server on port 6379.

You can interact with Redis using the built-in client:

```
$ src/redis-cli
redis> set foo bar
OK
redis> get foo
"bar"
```

Step 3: Install MySQL

The process is the same as process seen in the previous section. If fiware-facts is being installed in the same system as fiware-cloto, you could omit this step.

Step 4: Download and execute the facts engine server

1. Installing fiware-facts

Using pip Install the component by executing the following instruction:

```
pip install fiware-facts
```

This operation will install the component in your python site-packages folder.

It should shown the following information when it is executed:

```
Installing collected packages: fiware-facts
  Running setup.py install for fiware-facts

Successfully installed fiware-facts
Cleaning up...
```

2. Configuring fiware-facts

The configuration used by fiware-facts component is read from the configuration file located at `/etc/fiware.d/fiware-facts.cfg`

MySQL cloto configuration must be filled before starting fiware-facts component, user and password are empty by default. You can copy the default configuration file `facts_conf/fiware_facts.cfg` to the folder defined for your OS, and complete data about cloto MySQL configuration (user and password).

In addition, user could have a copy of this file in other location and pass its location to the server in running execution defining an environment variable called FACTS_SETTINGS_FILE.

Options that user could define:

```
[common]
brokerPort: 5000       # Port listening fiware-facts
clotoPort:  8000       # Port listening fiware-cloto
redisPort:  6379       # Port listening redis-server
redisHost:  localhost  # Address of redis-server
redisQueue: policymanager
rabbitMQ:   localhost  # Address of RabbitMQ server
cloto:      127.0.0.1  # Address of fiware-cloto
clotoVersion: v1.0
name:       policymanager.facts
maxTimeWindowsize: 10

[mysql]
host: localhost        # address of mysql that fiware-cloto is using
charset:    utf8
db: cloto
user:                  # mysql user
password:              # mysql password

[loggers]
keys: root

[handlers]
keys: console, file

[formatters]
keys: standard

[formatter_standard]
class: logging.Formatter
format: %(asctime)s %(levelname)s policymanager.facts %(message)s

[logger_root]
level: INFO            # Logging level (DEBUG, INFO, WARNING, ERROR, CRITICAL)
handlers: console, file

[handler_console]
level: DEBUG
class: StreamHandler
formatter: standard
args: (sys.stdout,)

[handler_file]
level: DEBUG
class: handlers.RotatingFileHandler
formatter: standard
logFilePath: /var/log/fiware-facts
logFileName: fiware-facts.log
logMaxFiles: 3
logMaxSize: 5*1024*1024  ; 5 MB
args: ('%(logFilePath)s/%(logFileName)s', 'a', %(logMaxSize)s, %(logMaxFiles)s)
```

Finally, ensure that you create a folder for logs `/var/log/fiware-facts/` (by default), with the right permissions to write in that folder.

```
mkdir -p /var/log/fiware-facts
```

3. Starting the server

Execute command:

```
gunicorn facts.server:app -b $IP:5000
```

Where $IP should be the IP assigned to the network interface that should be listening (ej. 192.168.1.33)

You can also execute the server with a different settings file providing an environment variable with the location of the file:

```
gunicorn facts.server:app -b $IP:5000
--env FACTS_SETTINGS_FILE=/home/user/fiware-facts.cfg
```

NOTE: if you want to see gunicorn log if something is going wrong, you could execute the command before adding `--log-file=-` at the end of the command. This option will show the logs in your prompt (standard stderr). If you want to store the log into a file just write `--log-file=<log file name>`.

When you execute the server you can see some information about the server:

```
2015-09-24 16:30:10,845 INFO policymanager.facts policymanager.facts 1.7.0

2015-09-24 16:30:10,846 INFO policymanager.facts Running in stand alone mode
2015-09-24 16:30:10,846 INFO policymanager.facts Port: 5000
2015-09-24 16:30:10,846 INFO policymanager.facts PID: 19472

2015-09-24 16:30:10,846 INFO policymanager.facts
                                          https://github.com/telefonicaid/fiware-facts



2015-09-24 16:30:10,896 INFO policymanager.facts Waiting for windowsizes
```

2.2.2 Sanity check procedures

The Sanity Check Procedures 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

Although one End to End testing must be associated to the Integration Test, we can show here a quick testing to check that everything is up and running. For this purpose we send a request to our API in order to test the credentials that we have from then and obtain a valid token to work with.

In order to make a probe of the different functionalities related to the Policy Manager, we start with the obtention of a valid token for a registered user. Due to all operations of the Policy Manager are using the security mechanism which is used in the rest of the cloud component, it is needed to provide a security token in order to continue with the rest of operations. For this operation we need to execute the following curl sentence.

```
curl -d '{"auth": {"tenantName": $TENANT,
"passwordCredentials":{"username": $USERNAME, "password": $PASSWORD}}}'
-H "Content-type: application/json" -H "Accept: application/xml"
http://<idm.server>:<idm.port)/v2.0/tokens
```

Both $TENANT (Project), $USERNAME and $PASSWORD must be values previously created in the OpenStack Keystone. The <idm.server> and <idm.port> are the data of our installation of IdM, if you planned to execute it you must changed it by the corresponding IP and Port of the FIWARE Keystone or IdM IP and Port values.

We obtained two data from the previous sentence:

- X-Auth-Token

```
<token expires="2012-10-25T16:35:42Z" id="a9a861db6276414094bc1567f664084d">
```

- Tenant-Id

```
<tenant enabled="true" id="c907498615b7456a9513500fe24101e0" name=$TENANT>
```

After it, we can check if the Policy Manager is up and running with a single instruction which is used to return the information of the status of the processes together with the queue size.

```
curl -v -H 'X-Auth-Token: a9a861db6276414094bc1567f664084d' -X GET
http://<fiware.cloto.server>:<fiware.cloto.port>/v1.0/c907498615b7456a9513500fe24101e0
```

This operation will return the information regarding the tenant details of the execution of the Policy Manager

```
< HTTP/1.0 200 OK
< Date: Wed, 09 Apr 2014 08:25:17 GMT
< Server: WSGIServer/0.1 Python/2.6.6
< Content-Type: text/html; charset=utf-8
{
    "owner": "Telefonica I+D",
    "doc": "http://docs.policymanager.apiary.io",
    "runningfrom": "14/04/09 07:45:22",
    "version": 2.7.0,
    "windowsize": 10
}
```

For more details to use this GE, please refer to the User & Programmers Guide.

List of Running Processes

Due to the Policy Manager basically is running over the python process, the list of processes must be only the python and redis in case of the facts engine. If we execute the following command:

```
ps -ewf | grep 'redis\|python' | grep -v grep
```

It should show something similar to the following:

```
2485   599   0 10:09 ?        00:00:01 src/redis-server *:6379
2704   599   0 10:23 ?        00:00:00 /usr/bin/python /usr/bin/gunicorn facts.server:app -b 0.0.0.0:5
```

Where you can see the Redis server, and the run process to launch the Python program.

In case of the rule engine node, if we execute the following command:

```
ps -ewf | grep 'rabbitmq-server\|python\|mysql' | grep -v grep
```

It should show something similar to the following:

```
559    554   0 07:47 ?        00:00:48 /usr/bin/python /usr/bin/gunicorn fiware_cloto.cloto.wsgi -b 0.0
1      0   0 07:45 ?          00:00:00 /bin/sh -e /usr/lib/rabbitmq/bin/rabbitmq-server
1      0   0 07:45 ?          00:00:14 mysqld
```

where we can see the rabbitmq, mysql and gunicorn process.

Network interfaces Up & Open

Taking into account the results of the ps commands in the previous section, we take the PID in order to know the information about the network interfaces up & open. To check the ports in use and listening, execute the command:

```
yum install -y lsof (apt-get for ubuntu or debian)
lsof -i | grep "$PID1\|$PID2"
```

Where $PID1 and $PID2 are the PIDs of Python and Redis server obtained at the ps command described before, in the previous case 5287 (redis-server) and 5604 (Python). The expected results must be something similar to the following:

```
COMMAND    PID USER    FD  TYPE             DEVICE SIZE/OFF NODE NAME
redis-ser 5287  fla    4u  IPv6 0x8a557b63682bb0ef      0t0  TCP *:6379 (LISTEN)
redis-ser 5287  fla    5u  IPv4 0x8a557b636a696637      0t0  TCP *:6379 (LISTEN)
redis-ser 5287  fla    6u  IPv6 0x8a557b63682b9fef      0t0  TCP localhost:6379->
localhost:56046 (ESTABLISHED)
Python    5604  fla    7u  IPv6 0x8a557b63682bacaf      0t0  TCP localhost:56046->
localhost:6379 (ESTABLISHED)
Python    5604  fla    9u  IPv4 0x8a557b6369c90637      0t0  TCP *:commplex-main
(LISTEN)
```

In case of rule engine, the result will we the following:

```
COMMAND    PID USER    FD  TYPE             DEVICE SIZE/OFF NODE NAME
python    2039       root    3u  IPv4  13290      0t0  UDP *:12027
python    2039       root    4u  IPv4  13347      0t0  TCP policymanager.novalocal
:irdmi (LISTEN)
python    2044       root    3u  IPv6  13354      0t0  TCP localhost:38391->localhost
:amqp (ESTABLISHED)
```

Databases

The last step in the sanity check, once that we have identified the processes and ports is to check the database that have to be up and accept queries. For the first one, if we execute the following commands inside the code of the rule engine server:

```
$ mysql -u user -p cloto
```

Where user is the administration user defined for cloto database. The previous command should ask you for the password and after that show you:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 155286
Server version: 5.6.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

In order to show the different tables contained in this database, we should execute the following commands with the result that we show here:

```
mysql> SHOW TABLES FROM cloto;
+----------------------------+
| Tables_in_cloto            |
+----------------------------+
| auth_group                 |
| auth_group_permissions     |
| auth_permission            |
| auth_user                  |
| auth_user_groups           |
| auth_user_user_permissions |
| cloto_entity               |
| cloto_entity_specificrules |
| cloto_entity_subscription  |
| cloto_rule                 |
| cloto_serverinfo           |
| cloto_specificrule         |
| cloto_subscription         |
| cloto_tenantinfo           |
| django_content_type        |
| django_session             |
| django_site                |
+----------------------------+
```

Now, we can execute a simple test query in order to check the content of the table:

```
mysql> select * from cloto.cloto_serverinfo;
```

It should return with the following information:

```
+----+----------------+---------+---------------------+--------+
| id | owner          | version | runningfrom         | doc    |
+----+----------------+---------+---------------------+--------+
|  1 | Telefonica I+D |       1 | 2014-10-02 14:04:41 | {file} |
+----+----------------+---------+---------------------+--------+
```

Where {file} is the path to the OpenSpecification file whose value is http://docs.policymanager.apiary.io

2.2.3 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

The resource availability in the node should be at least 2Gb of RAM and 8GB of Hard disk in order to prevent enabler's bad performance in both nodes. This means that bellow these thresholds the enabler is likely to experience problems or bad performance.

Remote Service Access

We have internally two components to connect, the Rule engine component and the facts engine component. After that two internals component, we should connect with the the IdM GE. An administrator to verify that such links are available will use this information.

The first step is to check that the facts engine is up and running, for this purpose we can execute the following curl command, which is a simple GET operation:

```
root@fiware:~# curl http://<Fact engine HOST>:5000/v1.0
```

The variable will be the IP direction in which we have installed the facts engine. This request should return the status of the server if it is working properly:

```
{"fiware-facts":"Up and running..."}
```

The second step is check that rule engine server is working properly too:

```
root@fiware:~# curl http://<Rule Engine HOST>:8000/info
```

We obtained a json with this content:

```
{
    "owner": "Telefonica I+D",
    "doc": "http://docs.policymanager.apiary.io",
    "runningfrom": "01/01/2016 07:47:06",
    "version": "2.7.0"
}
```

In order to check the connectivity between the rule engine and the IdM GE, due to it must obtain a valid token and tenant for a user and organization with the following curl commands:

```
root@fiware:~# curl
-d '{"auth": {"tenantName": "<MY_ORG_NAME>",
"passwordCredentials":{"username": "<MY_USERNAME>", "password": "<MY_PASS>"}}}'
-H "Content-type: application/json" -H "Accept: application/xml"
http://<KEYSTONE_HOST>:<KEYSTONE_PORT>/v2.0/tokens
```

The will be the name of my Organization/Tenant/Project predefined in the IdM GE (aka Keystone). The and variables will be the user name and password predefined in the IdM GE and finally the and variables will be the IP direction and port in which we can find the IdM GE (aka Keystone). This request should return one valid token for the user credentials together with more information in a xml format:

```
<?xml version="1.0" encoding="UTF-8"?>
<access xmlns="http://docs.openstack.org/identity/api/v2.0">
  <token expires="2012-06-30T15:12:16Z" id="9624f3e042a64b4f980a83afbbb95cd2">
    <tenant enabled="true" id="30c60771b6d144d2861b21e442f0bef9" name="FIWARE">
      <description>FIWARE Cloud Chapter demo project</description>
    </tenant>
  </token>
  <serviceCatalog>
  ...
  </serviceCatalog>
  <user username="fla" id="b988ec50efec4aa4a8ac5089adddbaf9" name="fla">
    <role id="32b6e1e715f14f1dafde24b26cfca310" name="Member"/>
  </user>
</access>
```

With this information (extracting the token id), we can perform a GET operation to the rule engine in order to get the information related to the window size associated to a tenant. For this purpose we can execute the following curl commands:

```
curl -v -H 'X-Auth-Token: a9a861db6276414094bc1567f664084d'
-X GET "http://<Rule Engine HOST$IP>:8000/v1.0/c8da25c7a373473f8e8945f5b0da8217"
```

The variable will be the IP direction in which we have installed the Rule engine API functionality. This request should return the valid info for this tenant in the following json response structure:

```
{
    "owner": "Telefonica I+D",
    "doc": "http://docs.policymanager.apiary.io",
    "runningfrom": "14/04/11 12:32:29",
    "version": "1.0",
    "windowsize": 10
}
```

Resource consumption

State the amount of resources that are abnormally high or low. This applies to RAM, CPU and I/O. For this purpose we have differentiated severals scenarios.

The results were obtained with a top command execution over the following machine configuration: In one of the machines it has been deployed the Bosun Generic Enabler and all his dependencies (Redis, MySQL, RabbitMQ, Orion Context Broker, etc). In the other machine, an Oracle Linux Virtual Machine with Openstack. The load was injected from that machine too.

Table 2.1: Machine Info

Machine	Bosun Generic Enabler	Openstack
Type Machine	Virtual Machine	Virtual Machine
CPU	CPU Intel(R) Xeon(R) CPU E31230. 4 cores @ 3,2Ghz	CPU Intel(R) Xeon(R) CPU E31230. 4 cores @ 3,2Ghz
RAM	4GB	4GB
HDD	128GB	128GB
Operating System	CentOS release 6.7 - 64 bits	CentOS release 6.7 - 64 bits

We have defined three different scenarios to check the resource consumption. These three scenarios consist in a stress scenario with a high load in a short period of time, configuring the "Security" parameter to "High" (token checking in each request), and the log file in debug mode. The second scenario is the same than the first one, but this time the "Security" parameter is configured to "Low", and the log file to info mode. The third one is a stability scenario. The goal of this scenario is to check if the system is degraded with a moderate load for a long period of time (4-6 hours).

The results of requirements both RAM, CPU and HTTP response (average per second) in case of Rule engine node is shown in the following table:

Table 2.2: Resource Consumption

Characteristic	High Usage	Low Usage	Stable
RAM	700Mb used	400Mb used	260Mb used
CPU	7% used	7% used	7% used
HTTP response/sec	19.0	19.6	24.1

And the results of requirements both RAM, CPU and HTTP response in case of Facts node is shown in the following table:

Table 2.3: Resource Consumption

Characteristic	High/Low Usage	Stable Usage
RAM	280Mb	200Mb
CPU	5% used	4.75% used
HTTP response/sec	20.4	27.4

I/O flows

The rule engine application is hearing from port 8000 and the Fact-Gen application (by default) is hearing in the port 5000. Please refer to the installation process in order to know exactly which was the port selected.

2.3 Architecture Description

2.3.1 Legal Notice

Please check the following Legal Notice to understand the rights to use these specifications.

Please go to GitHub's README for more documentation.

2.3.2 Overview

This specification describes the Policy Manager GE, which is a key enabler to scalability and to manage the cloud resources based on defined policies or rules.

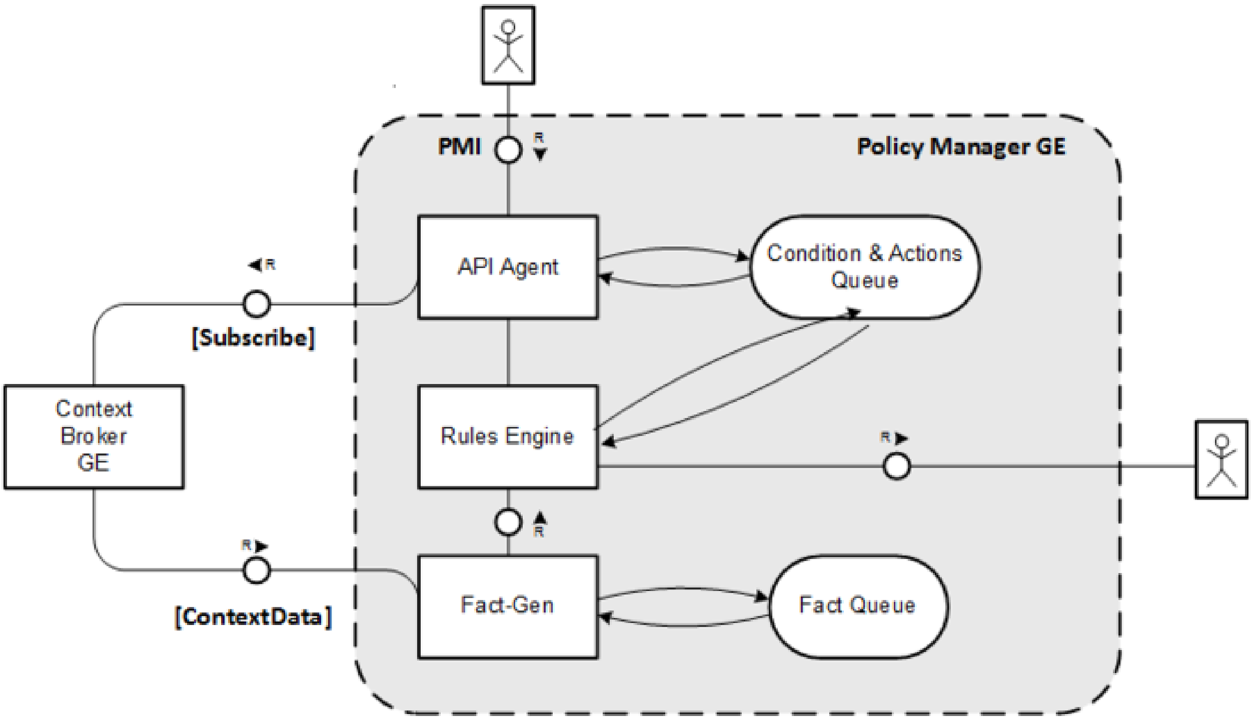
The Policy Manager GE provides the basic management of cloud resources based on rules, as well as management of the corresponding resources within the FIWARE Cloud Instance like actions based on physical monitoring or infrastructure, security monitoring of resources and services or whatever that could be defined by a facts, actions and rules.

The baseline for the Policy Manager GE is PyCLIPS, which is a module to interface CLIPS expert system and python language. The reason to take PyCLIPS is to extend the OpenStack ecosystem with a expert system written in the same language that the rest of the OpenStack services. Hence, Policy Manager offers the decision-making ability, independently of the type of resource (physical/virtual resources, network, service or whatever), able to solve complex problems within the Cloud field by reasoning about the knowledge base, represented by facts and rules.

The main functionality that the Policy Manager GE provides is:

- Management of scalability rules. It is possible to manage rules whose target is not to scale and this is also included in the main functionality of component.
- Management of different facts related to virtual machines and other facts in order to launch actions from the rules whose conditions are met.

The Policy Manager needs interaction with the user who provides the specification of the rules and actions that compound the knowledge system following a CLIPS language format. The facts are received from any producer of information that monitors the different resources of the cloud system. Context Broker GE, like publish/subscribe/notify system, interacts with the Policy Manager GE to suscribe to the information (facts) of Virtual Machines or whatever in order to get updated usage status of resources (e.g. cpu, memory, or disk) or resources that we want to monitor. These facts are used by the inference engine to deduce new facts based on the rules or infer new actions to take by third parties.



Policy Manager architecture specification

Target Usage

The Policy Manager GE is an expert system that provides independent server in the OpenStack ecosystem which evaluates the current state of the knowledge-base, applied the pertinent rules and infers new knowledge into the knowledge-base. Currently, the actions are designed to scale up and down Virtual Machines according to facts received from them (memory, cpu, disk or whatever). There are more kind of usage for these rules and is the user who defines conditions and actions he wants for. It is the user when specify the rule and actions who specify which is the use that we want to give to this GE.

2.3.3 Main concepts

Following the above FMC diagram of the Policy Manager, in this section we introduce the main concepts related to this GE through the definition of their interfaces and components and finally an example of their use.

Basic Concepts

The Policy Manager manages a set of rules which throws actions when certain conditions are activated when some facts are received. These rules can be associated with a specific virtual machine or be a general rule that affects the entire system. The key concepts, components and interfaces associated to the Policy Manager GE and visible to the cloud user, are described in the following subsections.

Entities

The main entities managed by the Policy Manager are as follows:

- **Rules**. They represent the policy that will be used to infer new facts or actions based on the facts received from the Context Broker GE. Usually, rules are some type of statement of the form: if then . The if part is the rule premise or antecedent, and the then part is the consequent. The rule fires when the if part is determined to be true or false. They are compound of 2 types of rules:
  - **General Rules**. They represent a global policy to be considered regardless specific virtual machines. Each rule is compound of a name to identify it and the condition and action which is fired. GeneralRules entities are represented as RuleModel.
  - **Specific Rules**. They represent a policy associated to a specific virtual machine. SpecificRules entities are represented as SpecificRuleModel.
- **Information**. It represent the information about the Policy Manager API and tenant information. Tenant information contains the window size, a modificable value for manage the minimal number of measures to consider a real fact for Rules Engine.
- **Facts**. They represent the measurement of the cloud resources and will be used to infer new facts or actions. an average of measures from a virtual machine trough the Context Broker GE. The are the base of the reasoning process.
- **Actions**, They are the output of the knowledge system related to a sense input and the are the implementation of the response rule or consequent.

Interfaces

The Policy Manager GE is currently composed of two main interfaces:

- **The Policy Manager interface (PMI)** is the exposed REST interface that implements all features of the Policy Manager exposed to the users. The PMI allows to define new rules an actions together with the activation of a specific rule asociated to a resource. Besides, this interface allow to get the information about this GE (url documentation, windows size, owner and time of the last server start). Besides, the PMI implements the NGSI-10 interface in order to receive the facts provided by Context Broker (notification of the context data) related to a virtual server.
- **Context Broker Manager Interface (NGSI)** is invoked in order to subscribe the Policy Manager to a specific monitoring resource. See NGSI-10 Open RESTful API Specification for more details.

Architecture Components

The Policy Manager includes a data repository which keeps the rules stored and information about the server, tenants.

- **API-Agent (PMI)** is responsible of offering a RESTful interface to the Policy Manager GE users. It triggers the appropriate manager to handle the request.
  - **InfoManager**, is responsible for the management of general information about the server running and specific tenant information like the window size.
  - **RuleManager**, is responsible for the management of all related with general rules and rules for specified virtual machines.
- **Rules Engine**. Is responsible for handling when a condition is satisfied based on the facts received and launch the associated actions.
  - **RuleEngineManager**, provides management for access the rule engine based on CLIPS, adding the new facts to the Rule Engine and check rule conditions.

  - **DbManager**, provides connection to the Data Base.
- **Fact-Gen**, provides the mechanisms to insert facts into the rule Engine from context data received.
  - **FactGenManager**, is responsible for the management of all related with data context build facts from this data.
- **Condition & Actions Queue**, which contains all the rules and actions that can be managed by Policy Manager, including the window size for each tenant.
- **Facts Queue**, which represents the actual instantiation of resources for a specific resource. For each element in the inventory (called *-Instance), there is an equivalent in the catalogue. This queue is implemented with a list on a data structure server in order to obtain a rapid response of the system.

Example Scenario

The Policy Manager GE is involved in three different phases:

- Management of the rules provided by users.
- Populate rule engine with facts collected from the data context.
- Management of rules status at runtime.

Rules Management

The management of rules involves several operations to prepare the scalability system working. First of all, the rules have to be defined. The definition of a rule includes the specification of the actions to be launched, the conditions that must be inferred and a descriptive name so user can easily recognize the rule. This rule can also be specified for a single virtual machine.

Secondly, to get facts, it must subscribe the virtual machine to Context Broker GE in order to receive notifications of the resources status. Context Broker GE updates the context of each virtual machined to which we are subscribed and the Policy Manager stores this information in a Queue system in order to get a stable monitored value without temporal oscillation of the signal.

Finally, the rules can be deleted or redefined. When a rule is deleted, Policy Manager unsubscribe the virtual machine from Context Broker if rule is a Specific Rule.

Collecting data

The Context Broker has subscribed a number of virtual machines. Each virtual machine publishes the status of its resources in the Context Broker GE and Policy Manager receives this notifications. After that, Policy Manager is in charge of build facts and insert them into de Rule Engine. When we receive a number of Facts equal to the window size, the Policy Manager calculates the arithmetic mean of the data and insert its value into the Rule Engine. Finally, Policy Manager discards the oldest value in the queue.

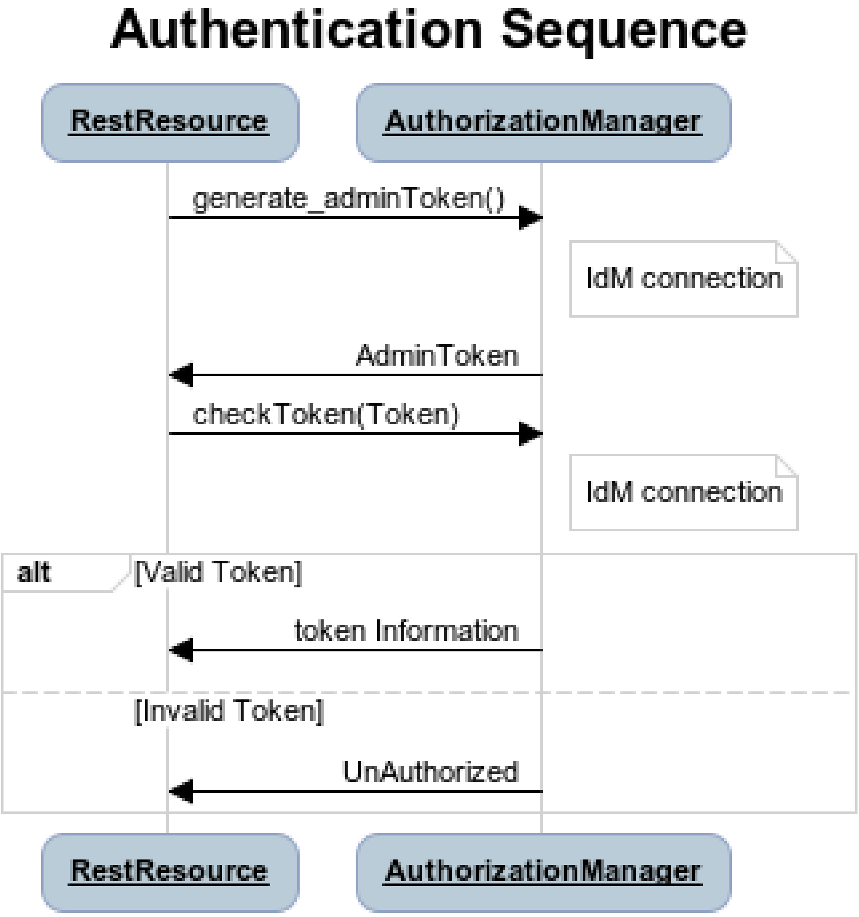
Runtime Management

During the runtime of an application, the Policy Manager can detect if a rule condition is inferred and is in charge of launch actions associated with, this action will be communicated to the users that was subscribed to this specific rule.

2.3.4 Main Interactions

The following pictures depicts some interactions between the Policy Manager, the Cloud Portal as main user in a typical scenario. For more details about the Open REST API of this GE, please refer to the Open Spec API specification.

First of all, every interaction need Authentication sequence before starting. Authentication sequence follows like this:



1. If Policy Manager have requested an administration Token before it will use this token to validate the future token received from the Cloud Portal.
2. If an existing administration token has expired or it is the first initialization, the Policy Manager requests a new administration Token from IdM in order to validate the future token received from the Cloud Portal through **generate_adminToken**() interface.
   (a) The IdM returns a valid administration token that will be used to check the *Token* received from the Cloud Portal requested message through the **checkToken(Token)** interface.
   (b) The IdM could return 2 options:
      i. If the *Token* is valid, the IdM returns the information related to this token.
      ii. If the *Token* is invalid, the IdM returns the message of unauthorized token.

The next interactions gets information about the Policy Manager server:

1. The User through Cloud Portal or CLI sends a GET operation to request information about the Policy Manager through **getInformation**().

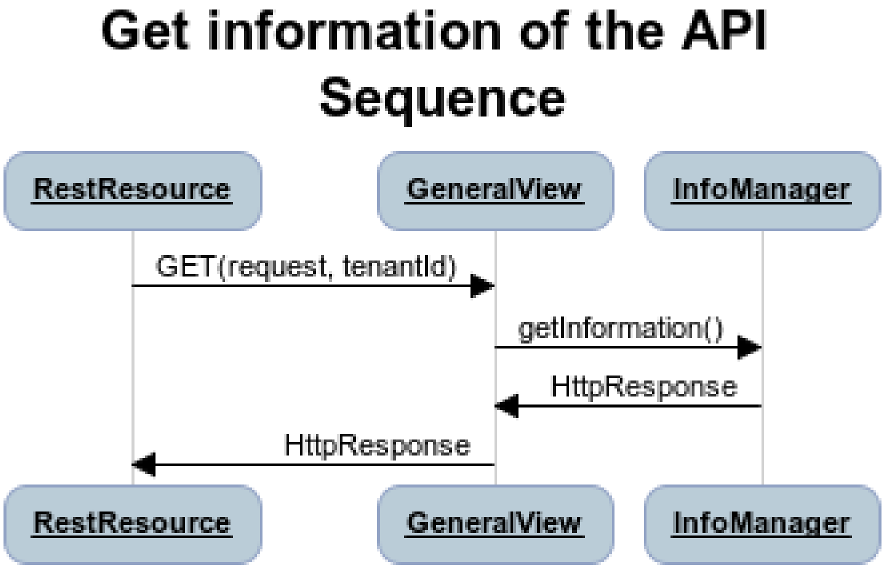


Fig. 2.1: Get Information sequence

2. The InfoManager returns the information related to the Policy Manager GE associated to this tenant.
   (a) Owner of the GEi.
   (b) Time and date of the last wake up of the Policy Manager GE.
   (c) URL of the open specification specification.
   (d) Window size of the facts stabilization queue.

Following, you can see request to update the window size.

1. The User through Cloud Portal or CLI sends a PUT message to the Policy Manager GE to update the window size of the tenantId through the **updateWindowSize()** message.
2. The Policy Manager returns a message with the information associated to this tenantId in order to confirm that the change was made.

Next, you can see the interactions to create general or specific rule sequence

1. The User through Cloud Portal or CLI requests a POST operation to create a new general/specific rule to the Policy Manager.
   (a) In case of general one, the **create_general_rule()** interface is used, with params *tenantId*, the OpenStack identification of the tenant, and the rule description.
   (b) In case of specific one, the **create_specific_rule()** interface is used, with params *tenantId*, the OpenStack identification of the tenant, the *serverId*, the OpenStack identification of the server, and the rule description.
2. The Rule Manager returns the new ruleModel associated to the new requested rule and the Policy Manager returns the respense to the user.
   (a) If something was wrong, due to incorrect representation of the rule, a *HttpResponseServerError* is returned in order to inform to the user that something was wrong.

Afterward, you could see the interactions to get information about already created general rules:

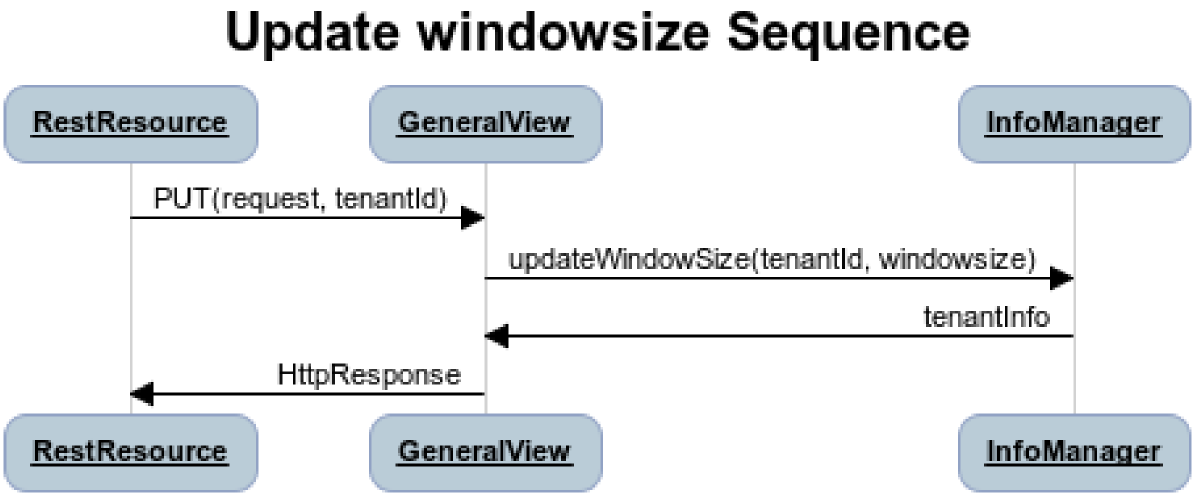


Fig. 2.2: Update Window Size sequence

1. The User through Cloud Portal or CLI requests a GET operation to the Policy Manager in order to receive all the general rules associated to a tenant through **get_all_rules()** interface with parameter *tenantId*
2. The Rule Manager component of the Policy Manager responses with the list of general rules.
3. If the tenant identify is wrong or whatever the Rule Manager responses a HttpResponseServerError.

Following, the interactions to get detailed information about getting general or specific rule sequence.

1. The User through Cloud Portal or CLI requests a GET operation to recover the rules.
   (a) If we decide to recover a general rule, the **get_rule()** interface should be used with *ruleId* parameter
   (b) Otherwise, if you decir to recover a specific rule, the **get_specific_rule()** interface should be used with the *ruleId* parameter.
2. The Rule Manager of the Policy Manager will return the ruleModel that it is stored in the Rule & Action Queue. If something was wrong, Policy Manager will return **HttpResponseServerError** to the user.

Next off, the interactions to delete general or specific rule.

1. The User through Cloud Portal or CLI requests the deletion of a general or specific rule to the Policy Manager with the identity of the tenant and rule.
   (a) The view sends the request to the RuleManager by calling the **delete_rule()** interface with identity of the rule as parameter of this interface to delete it.
   (b) Otherwise, if the rule is specific for a server, the views sends the request to the RuleManager by calling the **delete_specific_rule()** interface, with identity of the rule as parameter of this interface to delete it.
2. If the operation was ok, the RuleManager responses a *HttpResponse* with the ok message, by contrast, if something was wrong, it returns a *HttpResponseServerError* with the details of the problem.

Finally, the interactions to update a specific or general rule

1. The User through Cloud Portal or CLI requests the update of a general or specific rule to the Policy Manager with the identity of the tenant and rule.

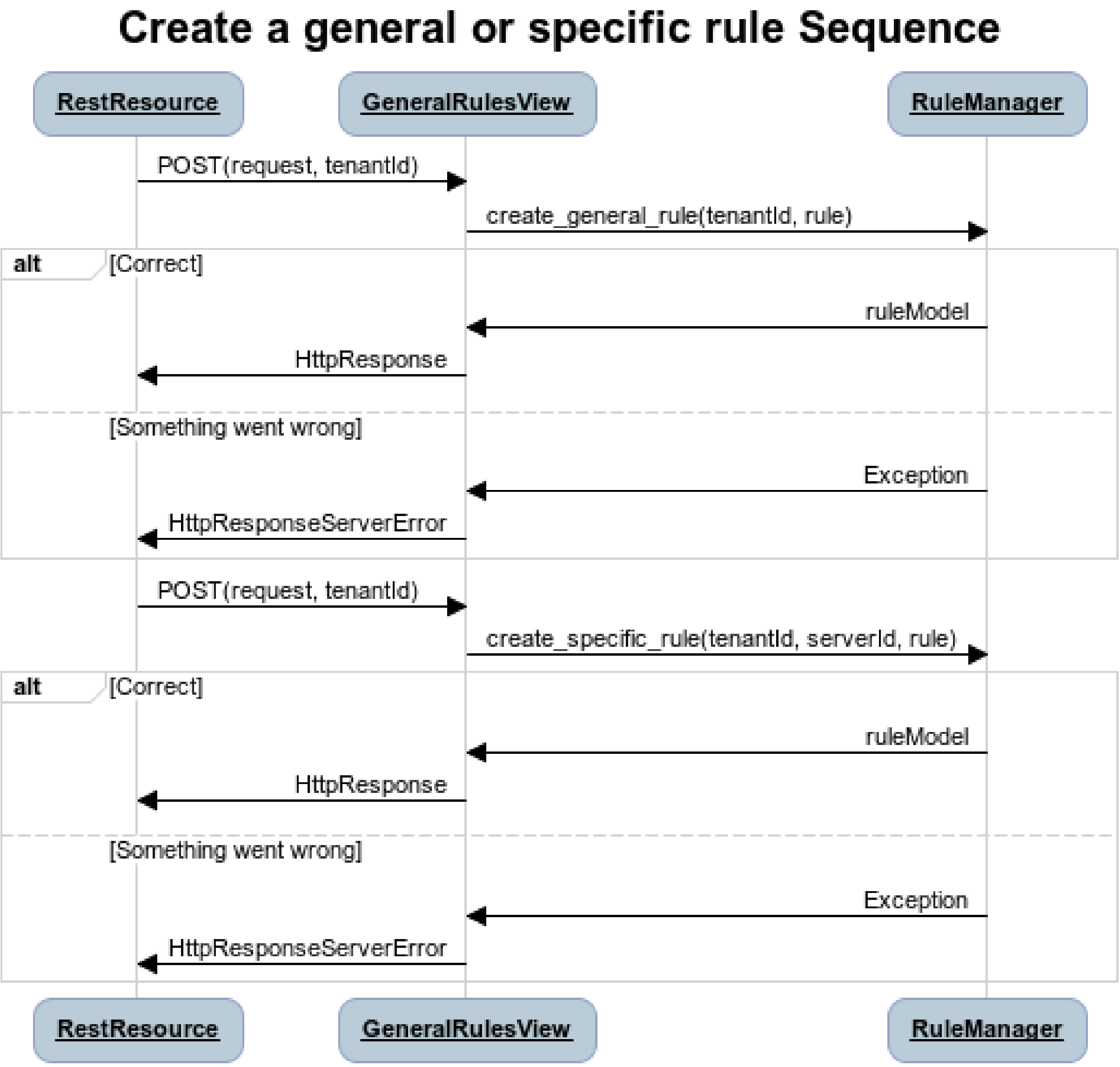


Fig. 2.3: Create general or specific rule sequence

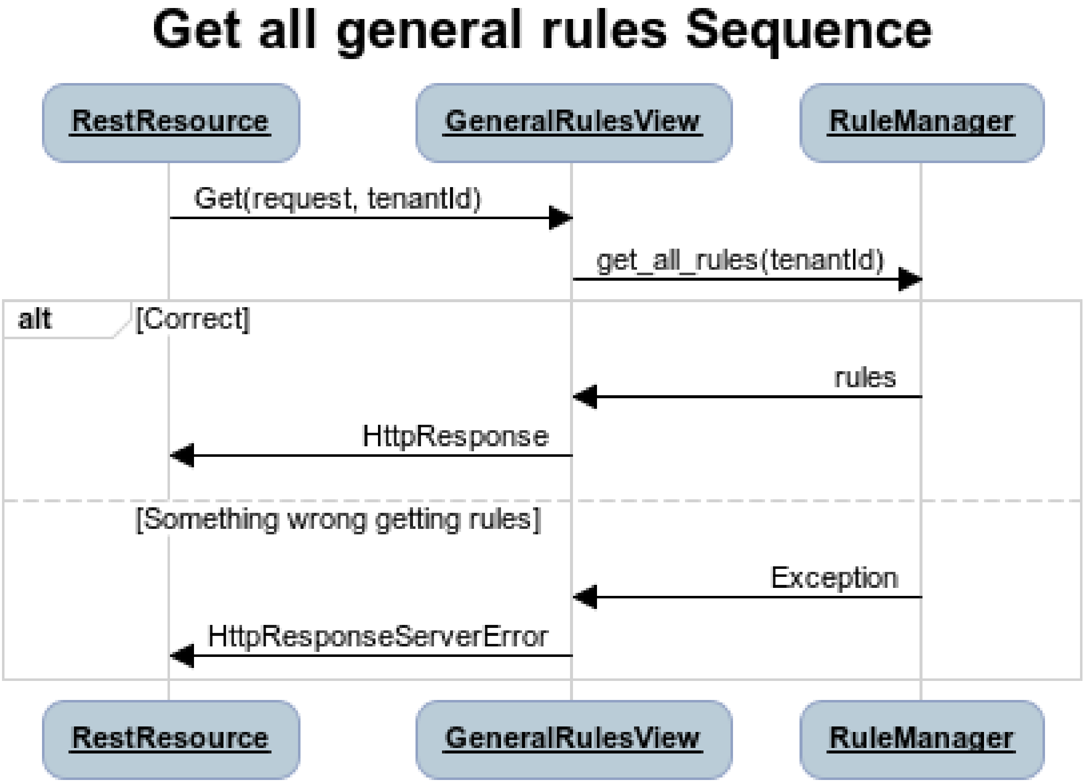


Fig. 2.4: Get all general rules sequence

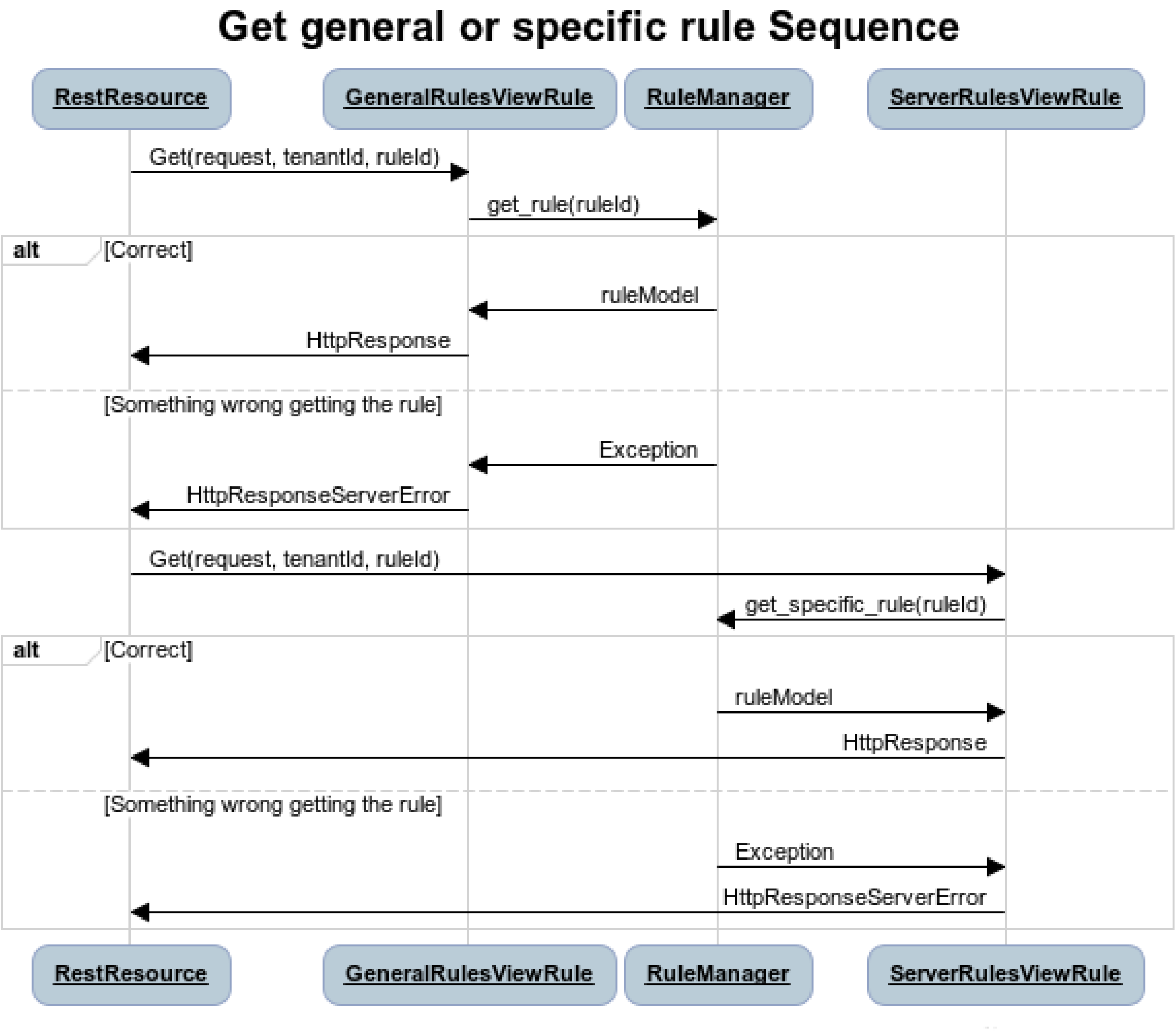


Fig. 2.5: Get general or specific rule sequence

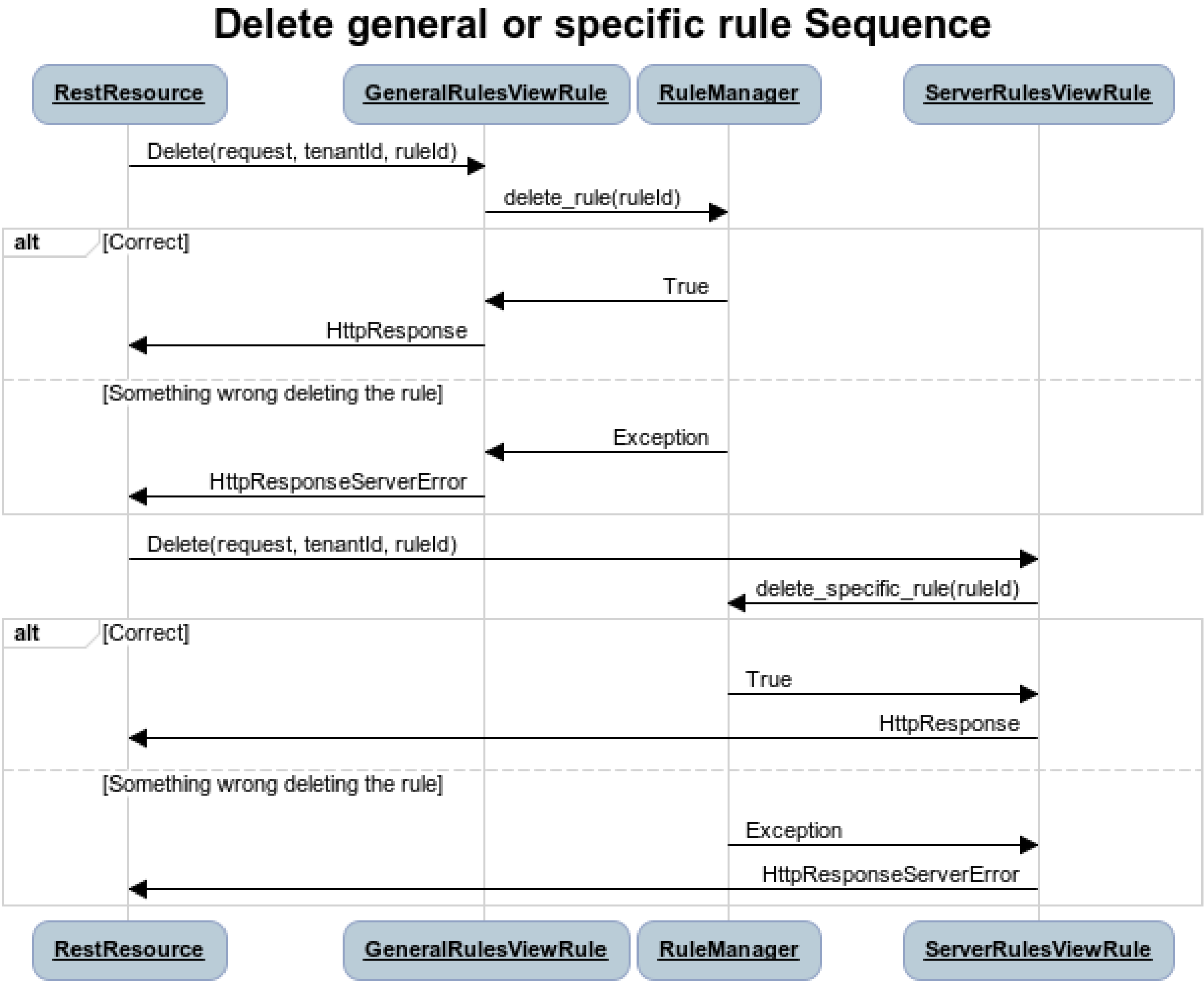


Fig. 2.6: Delete a general or specific rule sequence

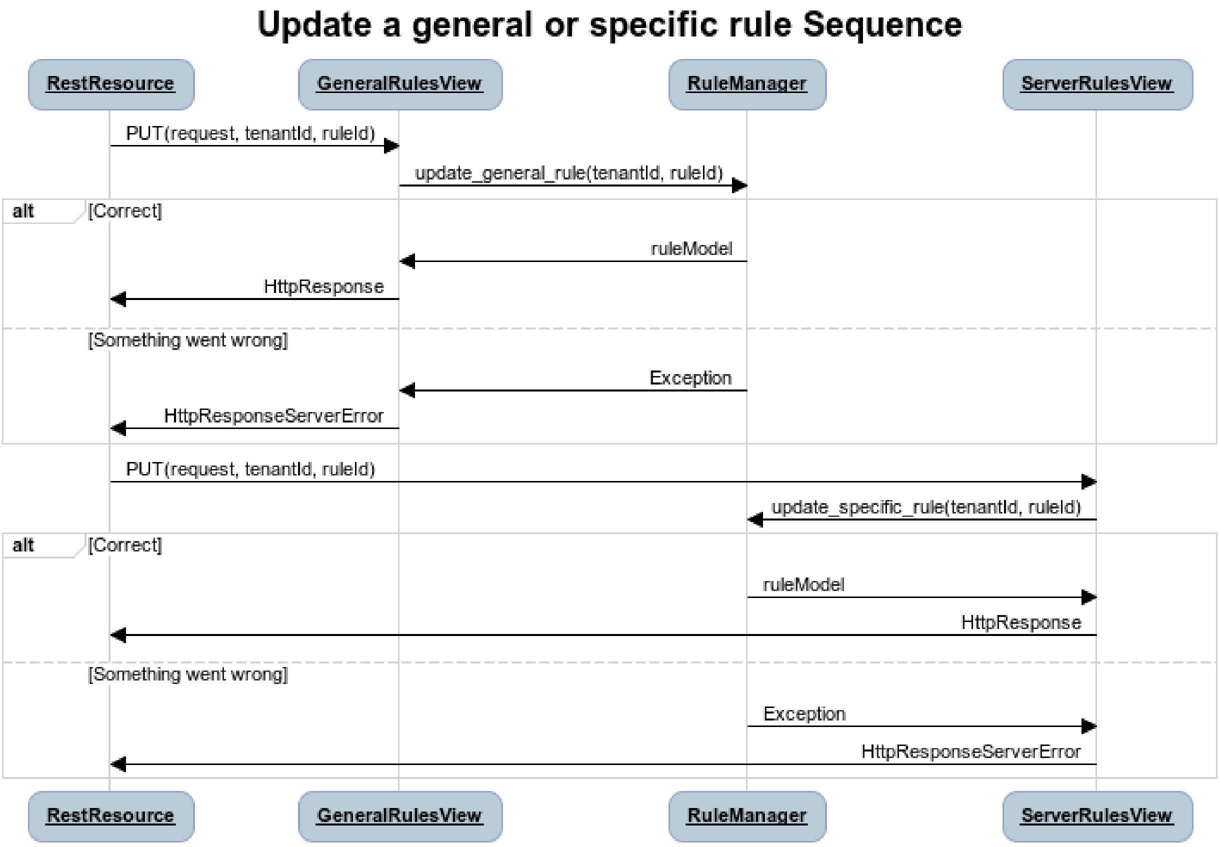


Fig. 2.7: Update a general or specific rule sequence

   (a) The view sends the request to the RuleManager by calling the **update_general_rule()** interface with identity of the tenant and rule as parameters of this interface to delete it.

   (b) Otherwise, if the rule is specific for a server, the views sends the request to the RuleManager by calling the **update_specific_rule()** interface, with identity of the tenant and rule as parameters of this interface to delete it.

2. If the operation was ok, the RuleManager responses with a new ruleModel class created and the API returns a *HttpResponse* with the ok message, by contrast, if something was wrong, it returns a *HttpResponseServerError* with the details of the problem.

2.3.5 Basic Design Principles

Design Principles

The Policy Manager GE has to support the following technical requirements:

- The condition to fire the rule could be formulated on several facts.
- The condition to fire the rule could be formulated on several interrelated facts (the values of certain variables in those facts match).
- User could add facts "in runtime" via API (without stop server).
- User could add rules "in runtime" via API (without stop server).
- That part of the implementation of the rule would:
  - Update facts.
  - Delete facts.
  - Create new facts.
- Actions can use variables used in the condition.
- Actions implementation can invoke REST APIs.
- Actions can send an email.
- The Policy Manager should be integrated into the OpenStack without any problem.
- The Policy Manager should interact with the IdM GE in order to offer authentication functionality to this GE.
- The Policy Manager should interact with the Context Broker GE in order to receive monitoring information from resources.

Resolution of Technical Issues

When applied to Policy Manager GE, the general design principles outlined at Cloud Hosting Architecture can be translated into the following key design goals:

- Rapid Elasticity, capabilities can be quickly elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
- Availability, Policy Manager should be running all the time without interruption of the service due to the nature of itself.
- Reliability, Policy Manager should assure that the activations of rule was produce by correct inference based on facts received from a Context Broker GE.

- Safety, is the Policy Manager has any problem, it should continue working without any catastrophic consequences on the user(s) and the environment.
- Integrity, Policy Manager does not allow the alteration of the facts queue and/or rules and actions queue.
- Confidentiality, Policy Manager does not allow the access to facts, rules and actions associated to a specitic tenant.

Regarding the general design principles not covered by the Cloud Hosting Architecture, they can be translated into the following key design goals:

- REST based interfaces, for rules and facts.
- The Policy Manager GE keeps stored all rules provisioned for each user.
- The Policy Manager GE manage all facts and checks when actions should be fired.