AuthZForce Documentation

Release 5.1.2

Cyril Dangerville, Thales Services

May 06, 2016

CHAPTER 1

Introduction

AuthZForce is the reference implementation of the Authorization PDP Generic Enabler (formerly called Access Control GE). Indeed, as mandated by the GE specification, this implementation provides an API to get authorization decisions based on authorization policies, and authorization requests from PEPs. The API follows the REST architecture style, and complies with XACML v3.0. XACML (eXtensible Access Control Markup Language) is a OASIS standard for authorization policy format and evaluation logic, as well as for the authorization decision request/response format. The PDP (Policy Decision Point) and the PEP (Policy Enforcement Point) terms are defined in the XACML standard. This GEri plays the role of a PDP.

To fulfill the XACML architecture, you may need a PEP (Policy Enforcement Point) to protect your application, which is not provided here. For REST APIs, we can use the PEP Proxy (Wilma) available in the FIWARE catalogue.

CHAPTER 2

Contents

2.1 AuthZForce - Installation and Administration Guide

This guide provides the procedure to install the AuthZForce server, including system requirements and troubleshooting instructions.

2.1.1 System Requirements

- CPU frequency: 2.6 GHz min
- CPU architecture: i686/x86_64
- RAM: 4GB min
- Disk space: 10 GB min
- File system: ext4
- Operating System: Ubuntu 14.04 LTS
- Java environment:
  - JDK 7 either from OpenJDK or Oracle;
  - Tomcat 7.x.

2.1.2 Installation

If you are still using AuthZForce 4.2.0 and want to upgrade, please proceed with the *Minimal setup* below, to install the new version; then the *Upgrade* section that follows, to transfer data from the old version.

Minimal setup

1. Install a JDK 7 if you don't have one already, using either of these two methods depending on your JDK preference:
   - If you prefer OpenJDK: `$ sudo aptitude install openjdk-7-jdk`
   - If you prefer Oracle JDK, follow the instructions from WEB UPD8. In the end, you should have the package `oracle-java7-installer` installed.
2. Install Tomcat 7: `$ sudo aptitude install tomcat7`.

3. Download the binary (Ubuntu package with `.deb` extension) release of AuthZForce from Maven Central Repository. You get a file called `authzforce-ce-server-dist-5.1.2.deb`.
4. Copy this file to the host where you want to install the software.
5. On the host, from the directory where you copied this file, run the following commands:

```
    $ sudo aptitude install gdebi curl
    $ sudo gdebi authzforce-ce-server-dist-5.1.2.deb
```

6. At the end, you will see a message giving optional instructions to go through. Please follow them as necessary.

Note that Tomcat default configuration may specify a very low value for the Java `Xmx` flag, causing the Authzforce webapp startup to fail. In that case, make sure Tomcat with `Xmx` at 1Go or more (2 Go recommended). For example, for Ubuntu 12.04, Tomcat default `Xmx` used to be 128m. You can fix it as follows:

```
$ sudo sed -i "s/-Xmx128m/-Xmx1024m/" /etc/default/tomcat
$ sudo service tomcat7 restart
```

Upgrade

If you are still using AuthZForce 4.2.0 and wish to upgrade, please download AuthZForce server upgrader tool distribution and follow the instructions from the upgrader tool README, starting with step 3. You may also find the `README` inside the downloaded `tar.gz` file itself.

Advanced setup

The *Minimal setup* gave you minimal installation steps to get started testing the features of the GE API. This may be enough for testing purposes, but barely for production. If you are targeting a production environment, you have to carry out extra installation and configuration steps to address non-functional aspects: security (including availability), performance, etc. The *Appendix* also gives some recommendations on what you should do.

2.1.3 Administration

Tomcat

For configuring and managing Tomcat, please refer to the official user guide.

Authzforce webapp

The Authzforce webapp configuration directory is located here: `/opt/authzforce-ce-server/conf`.

In particular, the file `logback.xml` configures the logging for the webapp (independently from Tomcat). By default, Authzforce-specific logs go to `/var/log/tomcat7/authzforce-ce/error.log`.

Restart Tomcat to apply any configuration change:

```
$ sudo service tomcat7 restart
```

Fast Infoset mode

Fast Infoset is an ITU-T/ISO standard for representing XML (XML Information Set to be accurate) using binary encodings, designed for use cases to provide smaller encoding sizes and faster processing than a W3C XML representation as text. The open source Fast Infoset project provide some performance results and more information about the standardisation status. There are several use cases at the origin of Fast Infoset. A major one comes from the Web3D consortium that is responsible for open standards in real-time 3D communication, and that adopted Fast Infoset for the serialization and compression of X3D documents. X3D is a standard for representing 3D scenes and objects using XML.

AuthZForce Server offers experimental support for Fast Infoset (use with caution). This feature is disabled by default. To enable Fast Infoset support, change the value of the parameter `spring.profiles.active` to `+fastinfoset` in the webapp context configuration file `/etc/tomcat7/Catalina/localhost/authzforce-ce.xml`; then restart Tomcat as shown in the previous section in order to apply changes.

Policy Domain Administration

The Concept of Policy Domain

The application is multi-tenant, i.e. it allows users or organizations to work on authorization policies in complete isolation from each other. In this document, we use the term *domain* instead of *tenant*. In this context, a policy domain consists of:

- Various metadata about the domain: ID assigned by the Authzforce API, external ID (assigned by the provisioning client), description, reference to the (root) active policy in the domain;
- A policy repository;
- Attribute Providers configuration: attribute providers provide attributes that the PEP does NOT directly provide in the XACML <Request>. For example, an attribute provider may get attribute values from an external database.

The reasons for creating different domains:

- Users or organizations do not want others to access their data, or even be impacted by others working on the same application.
- The same user or organization may want to work on different domains for different use cases; e.g. work with one policy for production environment, another for testing, another for a specific use case project, etc.

Default Domain Settings

Administrators can set default settings for all domains to make sure domains are created in a proper configuration according to an administrative policy, or, in more simple terms, the administrator's preferences. The administrator may change these settings in the various XML files inside the folder `/opt/authzforce-ce-server/conf/domain.tmpl`:

- `pdp.xml`:
  - `maxVariableRefDepth`: optional, positive integer that indicates the maximum depth of Variable reference chaining allowed in policies: `VariableDefinition` 1 -> `VariableDefinition` 2 -> ..., where -> represents a XACML VariableReference. No limit if undefined. This property applies only to policies loaded by the PDP, i.e. the root policy and policies referenced from it directly or indirectly via XACML PolicySetIdReference.
  - `maxPolicyRefDepth`: optional, positive integer that indicates the maximum depth of Policy(Set) reference chaining: `PolicySet` 1 -> `PolicySet` 2 -> ... -> `PolicySet` N; where -> represents a XACML PolicySetIdReference. No limit if undefined. This property applies only to policies loaded by the PDP, i.e. the root policy and policies referenced from it directly or indirectly via XACML PolicySetIdReference.

- policies/cm9vdA/0.1.0.xml: the default root XACML PolicySet enforced by the PDP on the domain. As an administrator, you may change the content of this policy on two conditions:

  1. You **must not** change the PolicySetId.
  2. If you change the Version (e.g. to 1.2.3), you **must** change the filename prefix (before .xsd extension) to the same value (e.g. 1.2.3.xsd).

- properties.xml: other domain properties, more specifically:

  - maxPolicyCount: optional, stricly positive integer that indicates the maximum number of policies on a domain, no limit if undefined.
  - maxVersionCountPerPolicy: optional, stricly positive integer that indicates the maximum number of versions per policy, no limit if undefined.
  - versionRollingEnabled: boolean, true if and only if policy versions should be rolled over, i.e. when maxVersionCountPerPolicy has been reached, oldest versions are automatically removed to make place.

Domain Creation

You create a domain by doing a HTTP POST request with XML payload to URL: http://${SERVER_NAME}:${PORT}/authzforce-ce/domains. Replace ${SERVER_NAME} and ${PORT} with your server hostname and port for HTTP. You can do it with curl tool:

```
$ export domainProperties="<?xml version="1.0" encoding="UTF-8" standalone="yes"?> \
  <ns4:domainProperties \
    xmlns:ns4="http://authzforce.github.io/rest-api-model/xmlns/authz/5" \
    externalId="external0"> \
    <description>This is my domain</description> \
  </ns4:domainProperties>"

$ curl --verbose --request POST \
  --header "Content-Type: application/xml;charset=UTF-8" \
  --data "$domainProperties" \
  --header "Accept: application/xml" \
  http://${SERVER_NAME}:${PORT}/authzforce-ce/domains

...
> POST /authzforce-ce/domains HTTP/1.1
> Content-Type: application/xml;charset=UTF-8
> Accept: application/xml
> Content-Length: 227
>
...
< HTTP/1.1 200 OK
< Server: Authorization System
< Date: Mon, 04 Aug 2014 13:00:12 GMT
< Content-Type: application/xml
< Transfer-Encoding: chunked
<
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <link xmlns="http://www.w3.org/2005/Atom"
  rel="item" href="h_D23LsDEeWFwqVFFMDLTQ"
  title="h_D23LsDEeWFwqVFFMDLTQ"/>
```

WARNING: Mind the leading and trailing single quotes for the --data argument. Do not use double quotes instead of these single quotes, otherwise curl will remove the double quotes in the XML payload itself, and send invalid XML which will be rejected by the server. The `--trace-ascii -` argument (the last dash here means *stdout*) is indeed a way to check the actual request body sent by `curl`. So use it only if you need to dump the outgoing (and incoming) data, in particular the request body, on *stdout*.

The `href` value in the response above gives you the domain ID (in the form of a Base64-encoded UUID) assigned by the API. You need this ID for any further operation on the domain.

Domain Removal

You remove a domain by doing a HTTP DELETE request with XML payload to URL: `http://${SERVER_NAME}:${PORT}/authzforce-ce/domains/{domain_ID}`. For example with `curl` tool:

```
$ curl --verbose --request DELETE \
 --header "Content-Type: application/xml;charset=UTF-8" \
 --header "Accept: application/xml" \
 http://${SERVER_NAME}:${PORT}/authzforce-ce/domains/h_D23LsDEeWFwqVFFMDLTQ
```

Policy administration is part of the Authorization Server API, addressed more extensively in the *Programmer Guide*.

2.1.4 High Availability

In order to achieve high availability with multiple AuthZForce Server instances (AuthZForce Server cluster), you need to make sure that the following directories are synchronized on all instances:

- Configuration directory: `/opt/authzforce-ce-server/conf`. This directory is not modified by the API but only by administrators having access to the directory, and any change to it requires restarting Tomcat to apply. Therefore, this directory requires synchronization only after a manual change by a server admin, which should not occur very often. When it occurs, the server administrators may reproduce the changes on each instance manually; or, if there are too many instances for this to be practical, they may use automatic file synchronization solutions, or a distributed filesystems (e.g. NFS) combined with file monitoring solutions. Both kinds of solutions must be capable of executing a specific command, to restart Tomcat in this case, whenever a filesystem change in the directory is detected on a instance node. For example, csync2 is a solution of the first kind that is free and open source.
- Data directory: `/opt/authzforce-ce-server/data`. This is where the Server API persists and retrieves domain data such as policies. Therefore, it is critical to keep this directory synchronized across all the nodes in the high availability cluster, using either file synchronization solutions such as csync2, or distributed file systems such as NFS. Besides, for usability and performance reasons, the AuthZForce server caches certain objects in memory such as domains' PDPs and ID-externalId mappings (more info in the *Programmer Guide*). Therefore, it is also critical to re-sync the AuthZForce Server cache after certain changes done directly by aforementioned solutions to the local data directory. There are two ways to do that:
  - **REST API**: you can keep the server in sync with the data directory by calling the following API operations, dependending on the type of change:
    - HEAD `/domains`: to be used after any global change to the data directory. Inappropriate and largely suboptimal if there are many domains but changes concern only one or a few of them, in which case the next operations should be preferred.
    - HEAD `/domains/{domainId}`: to be used after a specific domain directory `/opt/authzforce-ce-server/data/domains/{domainId}` is created.
    - DELETE `/domains/{domainId}`: to be used after a specific domain directory `/opt/authzforce-ce-server/data/domains/{domainId}` is deleted.

- HEAD /domains/{domainId}/properties: to be used after a specific domain's properties file /opt/authzforce-ce-server/data/domains/{domainId}/properties.xml is modified (especially the externalId property).
- HEAD /domains/{domainId}/pap/pdp.properties: to be used after a specific domain's PDP configuration file /opt/authzforce-ce-server/data/domains/{domainId}/pdp.xml or policies directory /opt/authzforce-ce-server/data/domains/{domainId}/policies is modified.

In these operations, you may use GET method instead of HEAD as well. However, HEAD is recommended for better performances as it does not return any content (response body), on the contrary to GET. Beware that the Content-Length returned by a HEAD is still the same as would be returned by the GET equivalent. In any case, if you opt for the file synchronization solution as mentioned earlier, you would have to make it call one of these operations depending on the type of change detected. If you opt for the distributed file system, you would need a file monitoring solution to detect changes and make such calls.

- **Embedded file monitoring threads**: it is possible to enable file monitoring threads embedded in AuthZForce Server. These threads check for changes to the local data directory periodically, and synchronize the cache automatically. This feature is disabled by default. To enable it, change the value of the parameter org.ow2.authzforce.domains.sync.interval to a strictly positive integer in the webapp context configuration file /etc/tomcat7/Catalina/localhost/authzforce-ce.xml. The parameter value indicates the period between two checks for changes, in seconds. Beware that this feature creates one extra thread per domain. Therefore, the impact on memory and CPU usage increases with the number of domains. Last but not least, **use this feature only on filesystems that support millisecond or higher resolution of file timestamps**, such as ext4 (supports nanosecond resolution). Indeed, Authzforce file monitoring threads use file timestamps to detect changes. As a result, if the resolution of the filesystem is coarser than the millisecond, and a file change occurred in less than a second after the last check, it will go undetected (the file's *mtime* timestamp is not updated), and synchronization will not work as expected.

2.1.5 Sanity check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that the 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

To check the proper deployment and operation of the Authorization Server, perform the following steps:

1. Get the list of policy administration domains by doing the following HTTP request, replacing ${host} with the server hostname, and ${port} with the HTTP port of the server, for example with curl tool:

```
$ curl --verbose --show-error --write-out '\n' \
  --request GET http://${host}:${port}/authzforce-ce/domains
```

2. Check the response which should have the following headers and body (there may be more headers which do not require checking here):

```
Status Code: 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:resources
  xmlns:ns2="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
```

```
    ... list of links to policy domains omitted here...
  </ns2:resources>
```

You can check the exact body format in the representation element of response code 200 for method `getDomains`, and all other API resources and operations in general, in the WADL (Web Application Description Language) document available at the following URL:

```
http://${host}:${port}/authzforce-ce/?_wadl
```

List of Running Processes

- One or more `java` processes for Tomcat.

Network interfaces Up & Open

- TCP 22;
- TCP 8080.

The port 8080 can be replaced by any other port Tomcat is listening to for HTTP connections to the webapp.

Databases

None.

2.1.6 Diagnosis Procedures

1. Perform the test described in *End to End testing*.
2. If you get a Connection Refused/Error, check whether Tomcat is started:

```
  $ sudo service tomcat7 status
```

3. If status stopped, start Tomcat:

```
  $ sudo service tomcat7 start
```

4. If Tomcat fails to start, check for any Tomcat high-level error in Tomcat log directory: `/var/log/tomcat7`
5. If Tomcat is successfully started (no error in server logs), perform the test described in *End to End testing* again.
6. If you still get a Connection Refused/error, check whether Tomcat is not listening on a different port:

```
  $ sudo netstat -lataupen|grep java
```

7. If you still get a connection refused/error, especially if you are connecting remotely, check whether you are able to connect locally, then check the network link, i.e. whether any network filtering is in place on the host or on the access network, or other network issue: network interface status, DNS/IP adress resolution, routing, etc.
8. If you get an error `404 Not Found`, make sure the webapp is deployed and enabled in Tomcat. Check for any webapp deployment error in file: `/var/log/tomcat7/authzforce-ce/error.log`.

Resource availability

To have a healthy enabler, the resource requirements listed in *System Requirements* must be satisfied, in particular:

- Minimum RAM: 4GB;
- Minimum CPU: 2.6 GHz;
- Minimum Disk space: 10 GB.

Remote Service Access

None.

Resource consumption

The resource consumption strongly depends on the number of concurrent clients and requests per client, the number of policy domains (a.k.a. tenants in this context) managed by the Authorization Server, and the complexity of the policies defined by administrators of each domain.

The memory consumption shall remain under 80% of allocated RAM. See *System Requirements* for the minimum required RAM.

The CPU usage shall remain under 80% of allocated CPU. See *System Requirements* for the minimum required CPU.

As for disk usage, at any time, there should be 1GB free space left on the disk.

I/O flows

- HTTPS flows with possibly large XML payloads to port 8443 or whatever port Tomcat is listening to for HTTPS connections to the webapp;
- HTTP flows with possibly large XML payloads to port 8080 or whatever port Tomcat is listening to for HTTP connections to the webapp.

2.1.7 Appendix

Security setup for production

You have to secure the environment of the application server and the server itself. Securing the environment of a server in general will not be addressed here, because it is a large subject for which you can find a lot of public documentation. You will learn about perimeter security, network and transport-level security (firewall, IDS/IPS...), OS security, application-level security (Web Application Firewall), etc. For instance, the *NIST Guide to General Server Security* (SP 800-123) is a good start.

Server Security Setup

For more Tomcat-specific security guidelines, please read Tomcat 7 Security considerations.

For security of communications (confidentiality, integrity, client/server authentication), it is also recommended to enable SSL/TLS with PKI certificates. The first step to set up this is to have your Certification Authority (PKI) issue a server certificate for your AuthZForce instance. You can also issue certificates for clients if you want to require client certificate authentication to access the AuthZForce server/API. If you don't have such a CA at hand, you can create your own (a basic one) with instructions given in the next section.

Certificate Authority Setup

If you have a CA already, you can skip this section. So this section is about creating a basic local Certificate Authority (CA) for internal use. This CA will be in charge of issuing certificates to the Authorization Server and clients, for authentication, integrity and confidentiality purposes. **This procedure requires using a JDK 1.7 or later.** (For the sake of simplicity, we do not use a subordinate CA, although you should for production, see keytool command example, use the `pathlen` parameter to restrict number of subordinate CA, `pathlen=0` means no subordinate.)

1. Generate the CA keypair and certificate on the platform where the Authorization Server is to be deployed (change the validity argument to your security requirements, example here is 365 days):

```
$ keytool -genkeypair -keystore taz-ca-keystore.jks -alias taz-ca \
  -dname "CN=My Organization CA, O=FIWARE" -keyalg RSA -keysize 2048 \
  -validity 365 -ext bc:c="ca:true,pathlen:0"
```

2. Export the CA certificate to PEM format for easier distribution to clients:

```
$ keytool -keystore taz-ca-keystore.jks -alias taz-ca \
  -exportcert -rfc > taz-ca-cert.pem
```

Server SSL Certificate Setup

For Tomcat 7, refer to the Tomcat 7 SSL/TLS Configuration HOW-TO.

Web Application Secutity

The AuthZForce web application exposes a XML-based API. Therefore it is vulnerable to XML denial-of-service attacks. To mitigate these attacks, there are two solutions:

- **Authzforce native protection**: you can add the following Environment entries in Authzfoce webapp context file `/etc/tomcat7/Catalina/localhost/authzforce-ce.xml` (if an entry is absent or its value is negative, the default value is used):

```
<Environment
 name="org.apache.cxf.stax.maxChildElements"
 description="Maximum number of child elements in an input XML element. Default: 50000."
 type="java.lang.Integer"
 value="1000"
 override="false" />

<Environment
 name="org.apache.cxf.stax.maxElementDepth"
 description="Maximum depth of an element in input XML. Default: 100."
 type="java.lang.Integer"
 value="100"
 override="false" />

<!--Following entries are not supported in Fast Infoset mode
(more info: https://issues.apache.org/jira/browse/CXF-6848) -->
<Environment
 name="org.apache.cxf.stax.maxAttributeCount"
 description="Maximum number of attributes per element in input XML. Default: 500."
 type="java.lang.Integer"
 value="100"
 override="false" />
```

```
    <Environment
     name="org.apache.cxf.stax.maxAttributeSize"
     description="Maximum size of a single attribute in input XML. Default: 65536 (= 64*1024)."
     type="java.lang.Integer"
     value="1000"
     override="false" />

    <Environment
     name="org.apache.cxf.stax.maxTextLength"
     description="Maximum size of XML text node in input XML. Default: 134217728 (= 128*1024*1024)."
     type="java.lang.Integer"
     value="1000"
     override="false" />
```

Restart Tomcat to apply changes.

- **Dedicated WAF**: for better mitigation, we recommend using a WAF (Web Application Firewall) with XML attack mitigation features in front of the Authzforce server.

There are commercial as well as open source WAFs available on the market. However, beware that this solution is not compatible with Fast Infoset, unless the WAF itself supports Fast Infoset. Similarly, if you want to use TLS, then the WAF or some proxy in front of it must support TLS to be the TLS server endpoint.

User and Role Management Setup

In production, access to the API must be restricted and explicitly authorized. To control which clients can do what on which resources, we need to have access to user identity and attributes and assign proper roles to them. These user and role management features are no longer supported by the AuthZForce server itself, but should be delegated to the Identity Management GE.

Domain Role Assignment

In production, access to the API must be restricted and explicitly authorized. To control which clients can do what on what parts of API, we need to have access to user identity and attributes and assign proper roles to them. These user role assignment features are no longer supported by the AuthZForce server itself, but should be delegated to the Identity Management GE.

Performance Tuning

For Tomcat and JVM tuning, we strongly recommend reading and applying - when relevant - the guidelines from the following links:

- Performance tuning best practices for VMware Apache Tomcat;
- Tuning Tomcat Performance For Optimum Speed;
- How to optimize tomcat performance in production;
- Apache Tomcat Tuning Guide for REST/HTTP APIs.

Last but not least, consider tuning the OS, hardware, network, using load-balancing, high-availability solutions, and so on.

2.2 AuthZForce - User and Programmers Guide

AuthZForce is the reference implementation of the Authorization PDP GE. In this regard, it provides an API to manage XACML-based access control policies and provide authorization decisions based on such policies and the context of a given access request. This guide explains how to use the API.

If you have been using a previous version of AuthZForce, check the release notes **to know what is changed and what is new.**

2.2.1 Background and Detail

This User and Programmers Guide relates to the reference implementation of the Authorization PDP GE which is part of FIWARE Security Architecture. Please find more information about this Generic Enabler in the following Open Specification.

2.2.2 User Guide

Since the Authorization PDP is a Generic Enabler which provides backend functionality to other applications (e.g. Generic Enablers or end user facing applications) and security administrators, we do not distinguish between the User and Programmers Guide. Please refer to the Programmers Guide section for more information.

2.2.3 Programmer Guide

AuthZForce provides the following APIs:

- PDP API (PDP = Policy Decision Point in the XACML terminology): provides an API for getting authorization decisions computed by a XACML-compliant access control engine;
- PAP API (PAP = Policy Administration Point in XACML terminology): provides API for managing XACML policies to be handled by the Authorization Service PDP.

The full API (RESTful) is described by a document written in the Web Application Description Language format (WADL) and associated XML schema files available in Authzforce rest-api-model project files.

XACML is the main international OASIS standard for access control language and request-response formats, that addresses most use cases of access control. AuthZForce supports the full core XACML 3.0 language; therefore it allows to enforce generic and complex access control policies.

Attribute-Based Access Control

AuthZForce provides Attribute-Based Access Control. To understand what is meant by *attribute* in the context of access control, below is the list of standard categories of attributes identified by the XACML standard:

- Subject attributes: the subject is an actor (human, program, device, etc.) requesting access to a resource; attributes may be user ID, Organization, Role, Clearance, etc. In fact, XACML enables you to be more specific about the type of subject, e.g. intermediary subject, requesting machine, etc.
- Resource attributes: the resource is a passive entity (from the access control perspective) on which subject requests to act upon (e.g. data but also human, device, application, etc.); resource attributes may be resource ID, URL, classification, etc.
- Action attributes: the action is the action that the subject requests to perform on the resource (e.g. create, read, delete); attributes may be action ID, parameter A, parameter B, etc.
- Environment attributes: anything else, e.g. current time, CPU load of the PEP/PDP, global threat level, etc.

If this is not enough, XACML enables you to make your own attribute categories as well.

Domain Management API

The API allows AuthZForce application administrators or administration interfaces to create domains for the users, and remove domains once they are no longer used. This part of the API is described in the section *Policy Domain Administration*. The API also allows users to update certain properties of the domain allocated to them:

- An **externalId** (optional) for the domain, which users/clients can modify and more easily use as reference, as opposed to the unique and read-only domain ID assigned by the API - once and for all - when the domain is created;
- A **description** of the domain (optional).

You may retrieve the current domain properties as follows:

- Method: GET
- Path: `/domains/{domainId}/properties`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example, this request gets the properties of domain `iMnxv7sDEeWFwqVFFMDLTQ`. In this case, there is no specific property, which is the case by default:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/properties
HTTP/1.1
Accept: application/xml; charset=UTF-8
```

The response goes:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:domainProperties
   xmlns:ns4="http://authzforce.github.io/rest-api-model/xmlns/authz/5" />
```

You may update the domain properties as follows:

- Method: PUT
- Path: `/domains/{domainId}/properties`
- Headers:
  - Content-Type: `application/xml; charset=UTF-8`
  - Accept: `application/xml; charset=UTF-8`
- Body: new properties.

For example, this request sets the `externalId` property to `my-domain-123`:

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/properties
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:domainProperties
 xmlns:ns4="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
 externalId="my-domain-123" />
```

The response is the new properties.

As a result, the domain's external ID `my-domain-123` points to the domain `iMnxv7sDEeWFwqVFFMDLTQ`. Clients may only rely on the externalId under their control to recover the API-defined domain ID, before they begin to use other API operations that require the API-defined domain ID. Indeed, clients may look up the API-defined ID corresponding to a given `externalId` as follows:

```
GET /domains?externalId=my-domain-123
HTTP/1.1
Accept: application/xml; charset=UTF-8
```

The response gives the corresponding domain ID in a link `href` attribute:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:resources
  xmlns:ns2="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
  xmlns:ns3="http://www.w3.org/2005/Atom">
  <ns3:link rel="item" href="iMnxv7sDEeWFwqVFFMDLTQ" title="iMnxv7sDEeWFwqVFFMDLTQ"/>
</ns2:resources>
```

Policy Administration API

The PAP is used by policy administrators to manage the policy repository from which the PDP loads the enforced policies. The PAP supports multi-tenancy in the form of generic administration domains that are separate from each other. Each policy administrsator (except the Superadmin) is in fact a domain administrator, insofar as he is allowed to manage the policy for one or more specific domains. Domains are typically used to support isolation of tenants (one domain per tenant).

Adding and updating Policies

The PAP provides a RESTful API for adding and updating policies to a specific domain. HTTP requests to this API must be formatted as follows:

- Method: POST
- Path: `/domains/{domainId}/pap/policies`
- Headers:
  - Content-Type: `application/xml; charset=UTF-8`
  - Accept: `application/xml; charset=UTF-8`
- Body: XACML PolicySet as defined in the XACML 3.0 schema.

Example of request given below:

```
POST /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet
 xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 PolicySetId="P1"
 Version="1.0"
 PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
 <Description>Sample PolicySet</Description>
```

```
<Target />
<Policy
 PolicyId="MissionManagementApp"
 Version="1.0"
 RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit">
 <Description>Policy for MissionManagementApp</Description>
 <Target>
  <AnyOf>
   <AllOf>
    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">MissionManagementApp</AttributeValue>
     <AttributeDesignator
      Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string"
      MustBePresent="true" />
    </Match>
   </AllOf>
  </AnyOf>
 </Target>
 <Rule RuleId="MissionManager_role_can_manage_team" Effect="Permit">
  <Description>Only MissionManager role authorized to manage the mission team</Description>
  <Target>
   <AnyOf>
    <AllOf>
     <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">Team</AttributeValue>
      <AttributeDesignator
       Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
       AttributeId="urn:thales:xacml:2.0:resource:sub-resource-id"
       DataType="http://www.w3.org/2001/XMLSchema#string"
       MustBePresent="true" />
     </Match>
    </AllOf>
   </AnyOf>
   <AnyOf>
    <AllOf>
     <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">manage</AttributeValue>
      <AttributeDesignator
       Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
       AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
       DataType="http://www.w3.org/2001/XMLSchema#string"
       MustBePresent="true" />
     </Match>
    </AllOf>
   </AnyOf>
  </Target>
  <Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">
    <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
     <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">MissionManager</AttributeValue>
     <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
      DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"
```

```
       Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" />
    </Apply>
   </Condition>
  </Rule>
 </Policy>
</PolicySet>
```

The HTTP response status is 200 with a link to manage the new policy, if the request was successful. The link is made of the policy ID and version separated by '/'.

Response:

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:link xmlns:ns3="http://www.w3.org/2005/Atom"
  rel="item" href="P1/1.0" title="Policy 'P1' v1.0"/>
```

To update a policy, you add a new version of the policy, i.e. you send the same request as above, but with a higher `Version` value.

Getting Policies and Policy Versions

Once added to the domain as shown previously, you can get the policy by its ID as follows:

- Method: GET
- Path: `/domains/{domainId}/pap/policies/{policyId}`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies/P1
HTTP/1.1
Accept: application/xml; charset=UTF-8
```

The response is the list of links to the versions of the policy `P1` available in the domain `iMnxv7sDEeWFwqVFFMDLTQ`:

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:resources
  xmlns:ns2="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
  xmlns:ns3="http://www.w3.org/2005/Atom">
    <ns3:link rel="item" href="1.0"/>
    <ns3:link rel="item" href="1.1"/>
    <ns3:link rel="item" href="2.0"/>
    <ns3:link rel="item" href="2.1"/>
    <ns3:link rel="item" href="2.2"/>
    ...
</ns2:resources>
```

As the `href` values are telling you, you may get a specific version of the policy as follows:

- Method: GET
- Path: `/domains/{domainId}/pap/policies/{policyId}/{version}`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies/P1/1.0
HTTP/1.1
Accept: application/xml; charset=UTF-8
```

The response is the policy document (XACML PolicySet) in this version.

You may use the special keyword `latest` as version here to get the latest version of a given policy; e.g. URL path `/domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies/P1/latest` points to the latest version of the policy `P1` in domain `iMnxv7sDEeWFwqVFFMDLTQ`.

Last but not least, you may get all policies in the domain as follows:

- Method: GET
- Path: `/domains/{domainId}/pap/policies`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies
HTTP/1.1
Accept: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:resources
  xmlns:ns2="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
  xmlns:ns3="http://www.w3.org/2005/Atom">
    <ns3:link rel="item" href="root"/>
    <ns3:link rel="item" href="P1"/>
    <ns3:link rel="item" href="P2"/>
    ...
</ns2:resources>
```

Removing Policies and Policy Versions

You may remove a policy version from the domain as follows:

- Method: DELETE
- Path: `/domains/{domainId}/pap/policies/{policyId}/{version}`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example:

```
DELETE /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies/P1/1.0
HTTP/1.1
Accept: application/xml; charset=UTF-8
```

The response is the removed policy document (XACML PolicySet) in this version.

You may remove a policy, i.e. all versions of a policy from the domain as follows:

- Method: DELETE
- Path: `/domains/{domainId}/pap/policies/{policyId}`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example:

```
DELETE /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies/P1
HTTP/1.1
Accept: application/xml; charset=UTF-8
```

The response is the list of links to all the removed versions of the policy, similar to the the GET request on the same URL.

Policy Repository (PRP) Properties

Administrators (global or domain-specific) may configure the policy repository with the following properties:

- `maxPolicyCount`: optional, stricly positive integer that indicates the maximum number of policies on a domain, no limit if undefined.
- `maxVersionCountPerPolicy`: optional, stricly positive integer that indicates the maximum number of versions per policy, no limit if undefined.
- `versionRollingEnabled`: boolean, true if and only if policy versions should be rolled over, i.e. when `maxVersionCountPerPolicy` has been reached, oldest versions are automatically removed to make place.

For example, below is a HTTP GET request and response for the policy repository properties of domain `iMnxv7sDEeWFwqVFFMDLTQ`:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/prp.properties
Accept: application/xml

-

HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:prpProperties xmlns:ns2="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
   <maxPolicyCount>10</maxPolicyCount>
   <maxVersionCountPerPolicy>10</maxVersionCountPerPolicy>
   <versionRollingEnabled>true</versionRollingEnabled>
</ns2:prpProperties>
```

The HTTP PUT request to update the properties has a body that is similar to the GET response:

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/prp.properties
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:prpProperties xmlns:ns2="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
   <maxPolicyCount>4</maxPolicyCount>
   <maxVersionCountPerPolicy>2</maxVersionCountPerPolicy>
   <versionRollingEnabled>true</versionRollingEnabled>
</ns2:prpProperties>
```

The response format is the same as for the GET request.

Policy Decision (PDP) Properties

Administrators (global or domain-specific) may configure the PDP engine with the following properties:

- `rootPolicyRefExpression`: reference - in the form of a XACML PolicySetIdReference - to the root policy. The root policy is the policy from which the PDP starts the evaluation. A policy matching this reference must exist on the domain, therefore it must have been added in the way described in *Adding and updating Policies*. If there is no specific `Version` in the reference, the latest matching policy version is selected.
- `feature`: optional, enables a particular PDP feature. Currently, only one value is accepted for this property: `urn:oasis:names:tc:xacml:3.0:profile:multiple:repeated-attribute-categories`. This enables support for the functionality identified by *urn:oasis:names:tc:xacml:3.0:profile:multiple:repeated-attribute-categories* in XACML v3.0 Multiple Decision Profile Version 1.0. Else (if this property is omitted) the functionality is disabled.

For example, below is a HTTP GET request and response for the PDP properties in domain `iMnxv7sDEeWFwqVFFMDLTQ`:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/pdp.properties
Accept: application/xml

-

HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:pdpProperties
 xmlns:ns2="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
 lastModifiedTime="2016-04-30T22:47:39.163Z">
 <rootPolicyRefExpression>default</rootPolicyRefExpression>
 <applicablePolicies>
   <rootPolicyRef Version="1.0">default</rootPolicyRef>
   <refPolicyRef Version="1.0">PPS:Employee</refPolicyRef>
 </applicablePolicies>
</ns2:pdpProperties>
```

As you can see, the GET response provides extra information such as:

- `lastModifiedTime`: the last time the PDP was reloaded (due to a change of root policy for instance);
- `applicablePolicies`: the actual root policy version selected for evaluation according to the `rootPolicyRefExpression`, and any policy referenced from it directly or indirectly via `PolicySetIdReference`.

The HTTP PUT request to update the properties goes as follows:

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/pdp.properties
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:pdpPropertiesUpdate xmlns:ns2="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
   <feature>urn:oasis:names:tc:xacml:3.0:profile:multiple:repeated-attribute-categories</feature>
   <rootPolicyRefExpression>root</rootPolicyRefExpression>
</ns2:pdpPropertiesUpdate>
```

This request sets the root policy reference to the latest version of the policy with `PolicySetId = 'root'` that must exist in the domain (see *Adding and updating Policies*), and enables support for repeated `<Attributes>` categories as part of the XACML v3.0 Multiple Decision Profile Version 1.0. You may remove the `feature` element if you want to disable it.

Re-usable Policies (e.g. for Hierarchical RBAC)

The PAP API supports policies that have references to other policies existing in the domain. This allows to include/reuse a given policy from multiple policies, or multiple parts of the same policy, by means of XACML `<PolicySetIdReference>` elements. One major application of this is Hierarchical RBAC. You can refer to the XACML v3.0 Core and Hierarchical Role Based Access Control (RBAC) Profile specification for how to achieve hierarchical RBAC with `<PolicySetIdReference>` elements.

For example, I want to define a role *Employee* and a role *Manager* derived from *Employee*. In other words, permissions of an *Employee* are included in the permissions of a *Manager*. In order to create this role hierarchy, we first add the Employee's *Permission PolicySet*:

```
POST /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<PolicySet
 PolicySetId="PPS:Employee"
 Version="1.0"
 PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
 <Description>Permissions specific to the Employee role</Description>
 <Target />
 <Policy
  PolicyId="PP:Employee"
  Version="1.0"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit">
  <Target />
  <Rule RuleId="Permission_to_create_issue_ticket" Effect="Permit">
   <Target>
    <AnyOf>
     <AllOf>
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">https://acme.com/tickets</AttributeValue>
       <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
        DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
      </Match>
     </AllOf>
    </AnyOf>
    <AnyOf>
     <AllOf>
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">POST</AttributeValue>
       <AttributeDesignator
        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string"
        MustBePresent="true" />
      </Match>
     </AllOf>
    </AnyOf>
   </Target>
  </Rule>
 </Policy>
</PolicySet>
```

Then we add the role-based hierarchical policy defining the Employee role and the Manager role, both with a reference (`<PolicySetIdReference>`) to the Employee's *Permission PolicySet* added previously. The Manager role has one policy more, so more permissions:

```
POST /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicySetId="rbac:policyset"
 Version="1.0"
 PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
 <Description>Root PolicySet</Description>
 <Target />
 <PolicySet PolicySetId="RPS:Employee" Version="1.0"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
  <Description>Employee Role PolicySet</Description>
  <Target>
   <AnyOf>
    <AllOf>
     <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">Employee</AttributeValue>
      <AttributeDesignator
       Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
       AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
       DataType="http://www.w3.org/2001/XMLSchema#string"
       MustBePresent="true" />
     </Match>
    </AllOf>
   </AnyOf>
  </Target>
  <PolicySetIdReference>PPS:Employee</PolicySetIdReference>
 </PolicySet>
 <PolicySet PolicySetId="RPS:Manager" Version="1.0"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
  <Description>Manager Role PolicySet</Description>
  <Target>
   <AnyOf>
    <AllOf>
     <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Manager</AttributeValue>
      <AttributeDesignator
       Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
       AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
       DataType="http://www.w3.org/2001/XMLSchema#string"
       MustBePresent="true" />
     </Match>
    </AllOf>
   </AnyOf>
  </Target>
  <Policy PolicyId="PP1:Manager" Version="1.0"
   RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit">
   <Description>Permissions specific to Manager Role</Description>
   <Target />
   <Rule
    RuleId="Permission_to_create_new_project" Effect="Permit">
    <Target>
     <AnyOf>
      <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
         DataType="http://www.w3.org/2001/XMLSchema#string">https://acme.com/projects</AttributeValue
        <AttributeDesignator
         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
         AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
         DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
       </Match>
      </AllOf>
     </AnyOf>
     <AnyOf>
      <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">POST</AttributeValue>
        <AttributeDesignator
         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
         AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
         DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
       </Match>
      </AllOf>
     </AnyOf>
    </Target>
   </Rule>
  </Policy>
  <!-- This role is senior to the Employee role, therefore includes the Employee role Permission
   PolicySet -->
  <PolicySetIdReference>PPS:Employee</PolicySetIdReference>
 </PolicySet>
</PolicySet>
```

You may add more policies for more roles as you wish. Once you are satisfied with your role hierarchy, you may apply your new RBAC policy by updating the domain's root policy reference (this may not be necessary if you reused the same root policy ID as before, in which case your policy is already active by now):

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/pdp.properties
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:domainProperties xmlns:ns4="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
 <rootPolicyRefExpression>rbac:policyset</rootPolicyRefExpression>
</ns4:domainProperties>
```

The policy is now enforced by the PDP as described in the next section.

Attribute Providers

The API allows to manage PDP attribute providers. These are PDP extensions that enable the PDP to get attributes from other sources than PEPs' requests. Such sources may be remote services, databases, etc. The AuthZForce Server distribution does not provide attribute providers out of the box, but allows you to plug in custom-made one(s) from your own invention or from third parties. The AuthZForce project also provides a separate Attribute Provider example, for testing and documentation purposes only. If you wish to make your own attribute provider, read on the next section. If you wish to test the example provided by AuthZForce or have another one ready for use, you may jump to the section *Integrating an Attribute Provider into AuthZForce Server*.

Making an Attribute Provider The steps to make your own PDP Attribute Provider extension for AuthZForce go as follows:

1. Create a Maven project with `jar` packaging type.
2. Create an XML schema file with `.xsd` extension in the `src/main/resources` folder of your Maven project. Make sure this filename is potentially unique on a Java classpath, like your usual Java class names. One way to make sure is to use a filename prefix following the same conventions as the Java package naming conventions. In this schema file, define an XML type for your attribute provider configuration format. This type must extend `AbstractAttributeProvider` from namespace `http://authzforce.github.io/xmlns/pdp/ext/3`. You may use the schema of AuthZForce Test Attribute Provider (used for AuthZForce unit tests only) as an example. In this example, the XSD filename is `org.ow2.authzforce.core.test.xsd` and the defined XML type extending `AbstractAttributeProvider` is `TestAttributeProvider`.
3. Copy the files `bindings.xjb` and `catalog.xml` from Authzforce source code into the `src/main/jaxb` folder (you have to create this folder first) of your Maven project.
4. Add the following Maven dependency and build plugin configuration to your Maven POM:

```
...
<dependencies>
 <dependency>
  <groupId>org.ow2.authzforce</groupId>
  <artifactId>authzforce-ce-core-pdp-api</artifactId>
  <version>3.7.0</version>
 </dependency>
 ...
</dependencies>
...

<build>
 ...
 <plugins>
  <plugin>
   <groupId>org.jvnet.jaxb2.maven2</groupId>
   <artifactId>maven-jaxb2-plugin</artifactId>
   <version>0.13.0</version>
   <configuration>
    <debug>false</debug>
    <strict>false</strict>
    <verbose>false</verbose>
    <removeOldOutput>true</removeOldOutput>
    <extension>true</extension>
    <useDependenciesAsEpisodes>false</useDependenciesAsEpisodes>
    <episodes>
     <episode>
      <groupId>org.ow2.authzforce</groupId>
      <artifactId>authzforce-ce-pdp-ext-model</artifactId>
      <version>3.3.7</version>
     </episode>
    </episodes>
    <catalog>src/main/jaxb/catalog.xml</catalog>
    <bindingDirectory>src/main/jaxb</bindingDirectory>
    <schemaDirectory>src/main/resources</schemaDirectory>
   </configuration>
  </plugin>
  ...
 </plugins>
</build>
...
```

5. Run Maven `generate-sources`. This will generate the JAXB-annotated class(es) from the XML schema into the folder `target/generated-sources/xjc`, one of which corresponds to your attribute provider XML type defined in the second step, therefore has the same name and also extends `org.ow2.authzforce.xmlns.pdp.ext.AbstractAttributeProvider` class corresponding to `AbstractAttributeProvider` type in the XML schema. For example, in the case of the Authzforce *Test Attribute Provider* aforementioned, the corresponding generated class is `org.ow2.authzforce.core.xmlns.test.TestAttributeProvider`. In your case and in general, we will refer to it as your *Attribute Provider Model Class*.

6. Create your Attribute Provider factory and concrete implementation class (as in the *Factory* design pattern). The Java class must extend `org.ow2.authzforce.core.pdp.api.CloseableAttributeProviderModule.FactoryBuilder<APM>`, where `APM` stands for your *Attribute Provider Model Class*. You may use the AuthZForce TestAttributeProviderModule class (used for AuthZForce unit tests only) as an example. In this example, the static nested class `Factory` is the one extending `CloseableAttributeProviderModule.FactoryBuilder<TestAttributeProvider>`. Such a class has a factory method `getInstance(APM configuration)` (`getInstance(TestAttributeProvider conf)` in the example) that, from an instance of your `APM` representing the XML input (`TestAttributeProvider` in the example), creates an instance of your Attribute Provider implementation class (`TestAttributeProviderModule` in the example). The latter must implement a method `get(attributeGUID, attributeDatatype, context))` in charge of actually retrieving the extra attributes (`TestAttributeProviderModule#get(...)` in the example). The `attributeGUID` identifies an XACML attribute category, ID and Issuer that the PDP is requesting from your attribute provider; the `attributeDatatype` is the expected attribute datatype; and `context` is the request context, including the content from the current XACML Request and possibly extra attributes retrieved so far by other Attribute Providers.

7. When your implementation class is ready, create a text file `org.ow2.authzforce.core.pdp.api.PdpExtension` in folder `src/main/resources/META-INF/services` (you have to create the folder first) and put the fully qualified name of your implementation class on the first line of this file, like in the example from Authzforce source code.

8. Run Maven `package` to produce a JAR from the Maven project.

Now you have an Attribute Provider extension ready for integration into AuthZForce Server, as explained in the next section.

Integrating an Attribute Provider into AuthZForce Server This section assumes you have an Attribute Provider extension in form of a JAR, typically produced by the process in the previous section. You may use AuthZForce Test

Attribute Provider JAR if you only wish to test the examples in this documentation. This JAR is available on Maven Central.

The steps to integrate the extension into the AuthZForce Server go as follows:

1. Make the JAR - and any extra dependency - visible from the AuthZForce webapp in Tomcat. One way to do it consists to copy the JAR (e.g. `authzforce-ce-core-3.8.3-tests.jar` in our example) into `/opt/authzforce-ce-server/webapp/WEB-INF/lib`. For other ways, please refer to Tomcat HowTo.
2. Import your attribute provider XML schema in the XML schema file `/opt/authzforce-ce-server/conf/authzforce-ext.xsd`, using `namespace` **only** (no `schemaLocation`), like in the example from Authzforce code with this schema import for Authzforce `TestAttributeProvider`:

```
<xs:import namespace="http://authzforce.github.io/core/xmlns/test/3" />
```

3. Add a `uri` element to XML catalog file `/opt/authzforce-ce-server/conf/catalog.xml`, with your attribute Provider XML namespace as `name` attribute value, and, the location of your XML schema file within the JAR, as `uri` attribute value, prefixed by `classpath:`. For example, in the sample XML catalog from Authzforce source code, we add the following line for Authzforce `TestAttributeProvider`:

```
<uri name="http://authzforce.github.io/core/xmlns/test/3" uri="classpath:org.ow2.authzforce.core
```

4. Finally, restart Tomcat to apply changes.

Managing attribute providers configuration Once you have deployed a new attribute provider extension on Authzforce, following previous instructions, you are ready to use it on a domain:

- Method: PUT
- Path: `/domains/{domainId}/pap/attribute.providers`
- Headers:
  - Content-Type: `application/xml; charset=UTF-8`
  - Accept: `application/xml; charset=UTF-8`
- Body: new attribute providers.

For example, this request instantiates a specific `TestAttributeProvider` configuration on domain `iMnxv7sDEeWFwqVFFMDLTQ` (as mentioned in the previous section, `TestAttributeProvider` is merely an example for testing and documentation purposes, it is not available in a default installation of Authzforce):

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/attribute.providers
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:attributeProviders
 xmlns:ns4="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
 xmlns:ns3="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
 <attributeProvider
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ns5="http://authzforce.github.io/core/xmlns/test/3"
  xsi:type="ns5:TestAttributeProvider" id="test">
  <ns3:Attributes
  Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
   <ns3:Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:example:attribute:role"
```

```
    IncludeInResult="false">
    <ns3:AttributeValue
     DataType="http://www.w3.org/2001/XMLSchema#string">Physician</ns3:AttributeValue>
   </ns3:Attribute>
  </ns3:Attributes>
 </attributeProvider>
</ns4:attributeProviders>
```

The response is the new attribute provider configuration from the request.

In this second example, we disable all PDP attribute providers of domain `iMnxv7sDEeWFwqVFFMDLTQ` by sending an empty element:

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/attribute.providers
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:attributeProviders xmlns:ns4="http://authzforce.github.io/rest-api-model/xmlns/authz/5" />
```

Finally, you may get the current attribute providers anytime as follows:

- Method: GET
- Path: `/domains/{domainId}/pap/attribute.providers`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example, this request gets the PDP attribute providers of domain `iMnxv7sDEeWFwqVFFMDLTQ`:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/attribute.providers
HTTP/1.1
Accept: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:attributeProviders xmlns:ns4="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
  ...
</ns4:attributeProviders>
```

Policy Decision API

The PDP API returns an authorization decision based on the currently enforced policy, access control attributes provided in the request and possibly other attributes resolved by the PDP itself. The Authorization decision is typically `Permit` or `Deny`. The PDP is able to resolve extra attributes not provided directly in the request, such as the current date/time (environment attribute).

The PDP provides an HTTP RESTful API for requesting authorization decisions. The HTTP request must be formatted as follows:

- Method: POST
- Path: `/domains/{domainId}/pdp`
- Headers:
  - Content-Type: `application/xml; charset=UTF-8`
  - Accept: `application/xml; charset=UTF-8`

- Body: XACML Request as defined in the XACML 3.0 schema.

The HTTP response body is a XACML Response as defined in the XACML 3.0 schema.

Example of request given below:

```
POST /domains/iMnxv7sDEeWFwqVFFMDLTQ/pdp
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<Request xmlns='urn:oasis:names:tc:xacml:3.0:core:schema:wd-17'
 CombinedDecision="false" ReturnPolicyIdList="false">
 <Attributes
  Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
  <Attribute
   AttributeId='urn:oasis:names:tc:xacml:1.0:subject:subject-id'
   IncludeInResult="false"> <AttributeValue
   DataType='http://www.w3.org/2001/XMLSchema#string'>joe</AttributeValue>
  </Attribute>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
   IncludeInResult="false"> <AttributeValue
   DataType='http://www.w3.org/2001/XMLSchema#string'>Manager</AttributeValue>
  </Attribute>
 </Attributes>
 <Attributes
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
  <Attribute
   AttributeId='urn:oasis:names:tc:xacml:1.0:resource:resource-id'
   IncludeInResult="false">
   <AttributeValue
    DataType='http://www.w3.org/2001/XMLSchema#string'>MissionManagementApp</AttributeValue>
  </Attribute>
  <Attribute
   AttributeId='urn:thales:xacml:2.0:resource:sub-resource-id' IncludeInResult="false">
   <AttributeValue
    DataType='http://www.w3.org/2001/XMLSchema#string'>Team</AttributeValue>
  </Attribute>
 </Attributes>
 <Attributes
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
  <Attribute
   AttributeId='urn:oasis:names:tc:xacml:1.0:action:action-id'
   IncludeInResult="false">
   <AttributeValue
    DataType='http://www.w3.org/2001/XMLSchema#string'>manage</AttributeValue>
  </Attribute>
 </Attributes>
 <Attributes
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment" />
</Request>
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns1:Response xmlns:ns1="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" ...>
```

```
   <ns1:Result>
       <ns1:Decision>Permit</ns1:Decision>
   </ns1:Result>
</ns1:Response>
```

Note for developers parsing XML manually or with namespace-UNaware parsers: the namespace prefix of the `Response` *element -* `ns1` *in this example - might vary from a run time to another, but it is always the same XML element as the prefix is always mapped to* `urn:oasis:names:tc:xacml:3.0:core:schema:wd-17` *(XACML 3.0 namespace). Therefore, any valid (namespace-aware) XML parser will handle it equally, no matter the namespace prefix.*

Fast Infoset

Fast Infoset is an ITU-T/ISO standard for representing XML (XML Information Set to be accurate) using binary encodings, designed for use cases to provide smaller encoding sizes and faster processing than a W3C XML representation as text. The open source Fast Infoset project provide some performance results and more information about the standardisation status. There are several use cases at the origin of Fast Infoset. A major one comes from the Web3D consortium that is responsible for open standards in real-time 3D communication, and that adopted Fast Infoset for the serialization and compression of X3D documents. X3D is a standard for representing 3D scenes and objects using XML.

AuthZForce Server API offers experimental support for Fast Infoset (use with caution). This feature is disabled by default and needs to be enabled explicitly by the administrator as told in the *Fast Infoset mode*. When it is enabled, provided that your API client supports Fast Infoset as well, you may use Fast Infoset on the server API by replacing the media type `application/xml` with `application/fastinfoset` in your API request headers (*Accept*/*Content-Type*).

Integration with the IdM and PEP Proxy GEs (e.g. for OAuth)

AuthZForce integrates with the Identity Management (KeyRock) and PEP Proxy GE (Wilma) reference implementations. For an overview of the main interactions, please refer to the Basic and Advanced sections of Wilma programmer guide.

After you installed and configured KeyRock, to connect it to Authzforce, you modify the properties with names prefixed by `ACCESS_CONTROL_` in the configuration file `fiware-idm/horizon/openstack_dashboard/local/local_settings.py` (example on KeyRock Github repository) according to your AuthZForce instance properties. Then go to IdM web interface, and check that the permissions and roles are well configured for your application. You may have to 'trigger' the policy generation in IdM by going to your application > *Manage roles* and click *Save* to trigger the XACML generation. More information in KeyRock installation and administration guide.

Then, after you installed and configured Wilma, to connect it to Authzforce, you modify the settings in `config.azf` object of configuration file `config.js` (example) according to your AuthZForce instance properties. More information in Wilma installation and administration guide.

Software Libraries for clients of AuthZForce or other Authorization PDP GEis

The full API (RESTful) is described by a document written in the Web Application Description Language format (WADL) and associated XML schema files available in Authzforce rest-api-model project files. Therefore, you can use any WADL-supporting REST framework for clients; for instance in Java: Jersey, Apache CXF. From that, you can use WADL-to-code generators to generate your client code. For example in Java, 'wadl2java' tools allow to generate code for JAX-RS compatible frameworks such as Apache CXF and Jersey. Actually, we can provide a CXF-based Java library created with this tool to facilitate the development of clients.