LF NETWORKING

Virtual Technical Meetings

LF NETWORKING

Virtual Technical Meetings

Policy Decisions and Enforcement

Pamela Dragosh

Agenda

- How to design a Policy Type
  - Inheriting from out-of-box ONAP Policy Types that are already available
  - How to create a new Policy Type that is new to the Platform.
- How to use the Policy Lifecycle API to load your Policy Type and create Policies.
- How to create an application in the XACML PDP to translate your Policy Type into a native policy for usage with the Policy Decision API.
- How to use the Policy Administrative API to push your policies to a running XACML PDP
- How to build enforcement into your application
  - Using the Decision API during runtime to retrieve your policy decision.
  - Responding to Dmaap Policy Notifications when a new Policy is available (or unavailable)

Policy Type Design

Design the Policy Type first and what you expect the policies to look like

There are many examples already available in ONAP to reference

https://github.com/onap/policy-models/tree/master/models-examples/src/main/resources

Decision application building

XACML PDP Guilin Documentation

- https://docs.onap.org/projects/onap-policy-parent/en/master/xacml/xacml.html

Clone the XACML PDP codebase

- https://github.com/onap/policy-xacml-pdp/tree/master
- Study the “applications” sub-module to see how the current set of applications are built.
  - Monitoring
  - Naming
  - Guard
    - Blacklist, Frequency Limiter, Min/Max, Filter (new to Guilin)
  - Optimization
  - Native (Frankfurt)
  - Match (new to Guilin)

Continued…

Each application translates policies and responds to the Decision API appropriately.
If your PEP has different needs from out-of-box, then you can build a custom XACML PDP Application.

- Wish to add another layer of algorithm's on top of XACML PDP engine decision.
  - Eg. Extend the existing translators or override some of their methods and change the behavior
- Prefer to translate TOSCA Policies differently
  - Eg. Write your own translator
- Desire to segregate "actions" into your own application
  - An "action" is used in the Decision API payload to forward a Decision request to an application.
  - Applications can have more than one action associated with their application

Implementation details

XacmlApplicationServiceProvider interface

- This is the interface that an application is required to implement
- The XACML PDP uses java.service to find implementations of this service interface to load into the PDP and make available
- **StdXacmlApplicationServiceProvider** is an implementation of this interface that performs a lot of common work that the packaged applications utilize.
    - Strongly recommend using this interface and overriding methods as appropriate

Continued…

ToscaPolicyTranslator interface

- Must be provided to the implementation of the XacmlApplicationServiceProvider for policy translation
- There are some implementations available for use:
  - StdCombinedPolicyResultsTranslator – very basic, not recommended
  - StdMatchableTranslator – recommended, allows flexibility of attributes to be used in policy design
- Possible to create your own translator if desired

Step #1 – build the decision app

Let's see some code….

NOTE: Previous tutorials existing for Frankfurt

- Code is uploaded into this wiki: https://wiki.onap.org/pages/viewpage.action?pageId=84654893
- Tutorial is documented in readthedocs and the code is available there as well
  - https://docs.onap.org/projects/onap-policy-parent/en/frankfurt/xacml/xacml-tutorial.html

Step #2 - testing

Docker Compose script

There is a docker-compose.yml script saved in the source code tar file:

```
src/main/docker/docker-compose.yml
```

To start it run these commands:

```
docker-compose -f src/main/docker/docker-compose.yml run --rm start_dependencies
docker-compose -f src/main/docker/ docker-compose.yml run --rm start_all
```

NOTE: You may have to tweak the script to get it running in your environment.

MariaDb

Dmaap simulator

- For Guilin, the policy team now builds a version for developer testing use

Continue...

Docker Compose script

```
docker-compose -f src/main/docker/ docker-compose.yml run --rm start_all
```

This will start the api and the pap

Step #3 – deploy policies

LF NETWORKING Virtual Technical Meetings

Some derived policy types can be automatically deployed

- onap.policies.monitoring.*
- onap.policies.optimization.*
- onap.policies.match.*

For new/custom Policy Types, you must declare it as supported in your PDP group

Step #4 – build enforcement app

Current ONAP components that are PEP’s (Policy Enforcement Points):

- DCAE analytics/collectors – support Monitoring Policy Types
- OOF – support Optimization Policy Types
- SDNC – support Naming Policy Types

What code do I need to develop?

Your application should be able to make a RESTful API call

Your application should be able to parse the JSON payload of the RESTful API call

If your application needs notification of a policy change, it should be able to connect to Dmaap to receive Policy Update notifications

Is there an SDK in Policy to support this?

- Not exactly an SDK, but Java-based applications can use our java artifacts in our policy/common and policy/models repositories if they do not have code in their codebase that can perform RESTful API calls

Starting from scratch…

Eclipse can be used to build a new Java Maven application

Create an empty maven Java project

- Ensure using JDK 11 for compilation as the Policy Framework only supports JDK 11

Find the latest ONAP Policy Frameworks java artifacts and Docker images by consulting our wiki page:

- https://wiki.onap.org/display/DW/Policy+Framework+Project%3A+Component+Versions
- This tutorial will use policy/common and policy/models java artifacts

More information

Developer Documentation is located here:

https://docs.onap.org/projects/onap-policy-parent/en/master/development/development.html