Finanziato dall'Unione europea
NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA

ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

Exploring authorization in Grid and Cloud middleware with Open Policy Agent

Federica Agostini

EGI2024, Lecce (Italy), October 3rd 2024

OPEN POLICY AGENT

Open Policy Agent (OPA) is an open-source authorization engine

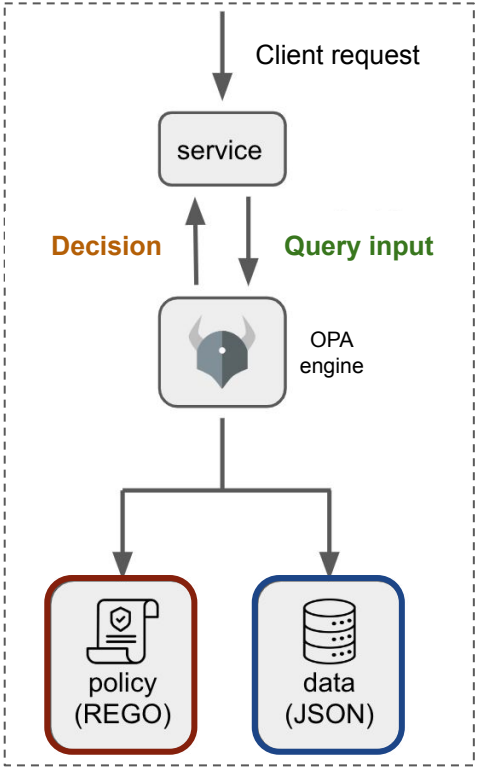
OPA is based on an high-level declarative language (***Rego***) that allows the definition of policies as code

Rego is designed for expressing policies over complex hierarchical data structures

- policy authors can focus on *what* queries should return rather than *how* they should be executed
- Rego ensures high performance policy decisions, even with increasing number of rules

A service which needs to take a policy decisions can **query** OPA with arbitrary structured data (JSON or YAML) as **input**

- OPA evaluates the query input against **policies** and optionally **data**
- OPA **decision** is not limited by simple *allow/deny* answer, but can generate arbitrary structured data as output



The Rego Playground

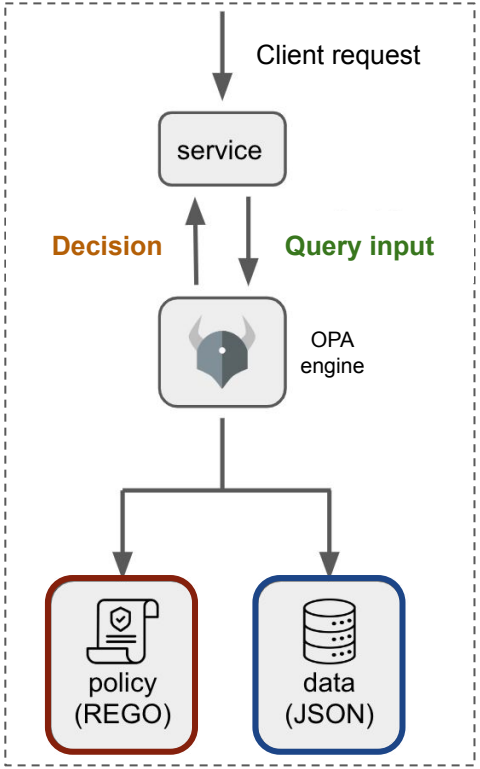
Examples

OPA playground

Options | Evaluate | Format | Publish

```
# Role-based Access Control (RBAC)
# -------------------------------
#
# This example shows how to:
#
#   * Define an RBAC model in Rego that interprets
#     role mappings represented in JSON.
#   * Iterate/search across JSON data structures
#     (e.g., role mappings)
#

package app.rbac

import rego.v1

default allow := false

allow if user_is_admin

allow if {
    some grant in user_is_granted

    input.action == grant.action
    input.type == grant.type
}

user_is_admin if "admin" in data.user_roles[input.user]

user_is_granted contains grant if {
    some role in data.user_roles[input.user]

    some grant in data.role_grants[role]
}
```

Rego policies



INPUT

```
{
    "user": "alice",
    "action": "read",
    "object": "id123",
    "type": "dog"
}
```

Query input

DATA

```
{
    "user_roles": {
        "alice": [
            "admin"
        ],
        "bob": [
            "employee",
            "billing"
        ],
        "eve": [
            "customer"
        ]
    },
```

Structured data used by policies (optional)

OUTPUT

Found 1 result in 218µs.

```
{
    "allow": true,
    "user_is_admin": true,
    "user_is_granted": []
}
```

Decision

LINT

No linter violations

Built by styra

OPA v0.64.1, Regal v0.21.3

This link can be used to share the versioned configuration among developers

Share

NEW

https://play.openpolicyagent.org/p/KrEzOAoKNJ

Copy

Install OPA

v0.64.1 OPA installation docs

Linux macOS Windows

```
curl -L -o opa \
https://openpolicyagent.org/downloads/v0.64.1/opa_linux_amd64; \
chmod 755 ./opa
```

Copy

Run OPA with playground policy

Heads up! The Rego playground is intended for development. Don't rely on it for your production deployments.

```
./opa run --server \
--log-format text \
--set decision_logs.console=true \
--set bundles.play.polling.long_polling_timeout_seconds=45 \
--set services.play.url=https://play.openpolicyagent.org \
--set bundles.play.resource=bundles/lJvvxntPRg
```

Copy

Query OPA with playground input

Test by piping your playground's JSON input into your OPA served playground policy

```
curl https://play.openpolicyagent.org/v1/input/lJvvxntPRg \
| curl localhost:8181/v1/data -d @-
```

Copy

```
#
#   * Define an RBAC model in Rego that interpr
#     role mappings represented in JSON.
#   * Iterate/search across JSON data structure
#     (e.g., role mappings)
#

package app.rbac

import rego.v1

default allow := false

allow if user_is_admin

allow if {
    some grant in user_is_granted

    input.action == grant.action
    input.type == grant.type
}
```

Evaluate Format Publish

```
": "alice",
ion": "read",
ect": "id123",
e": "dog"

roles": {
alice": [
    "admin"
],
bob": [
    "employee",
    "billing"
],
eve": [
    "custome
]

result in 21

ow": true,
_is_admin":
_is_granted": []
```

lations

curl example on how to query the policies hosted on the OPA remote server

In our use cases, we used to own the OPA server which runs with local configurations (**Rego** and **data**)

Built by segrd

OPA v0.64.1, Regal v0.21.3

Usage of OPA for the Grid and Cloud middleware

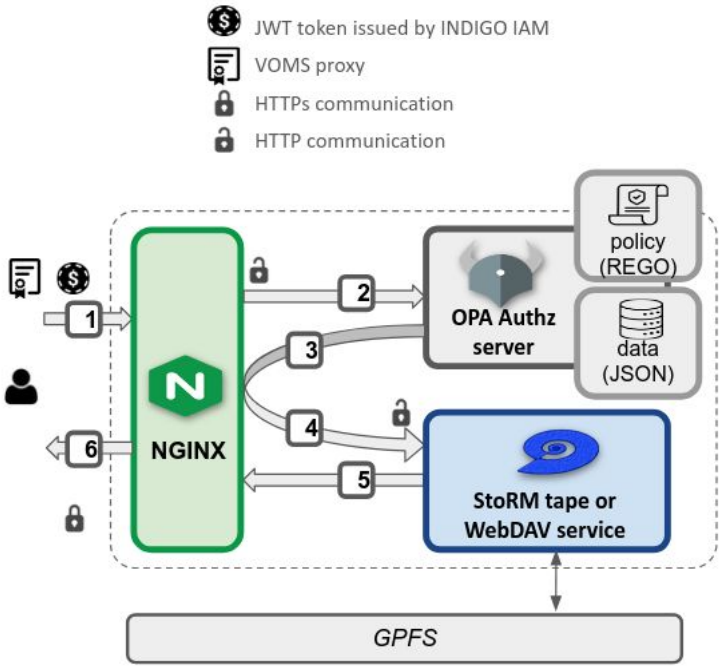
STORM WEBDAV AND STORM TAPE

The StoRM WebDAV service provides HTTP/WebDAV access to resources shared on a filesystem

The StoRM tape REST API is the implementation of the WLCG Tape REST API spec, which allows to recall files stored on tape

OPA is used for authN/Z, as follows:

1. the user submits a stage request, by presenting a **X509/VOMS proxy** or **JWT token**. The request is VOMS/TLS terminated by NGINX
2. NGINX sends the request to the OPA engine for JWT authN and JWT/VOMS authZ
3. OPA makes the authZ decision using its policies and data. In case of negative authZ, NGINX returns 403 Forbidden
4. in case of successful authZ, the request is forwarded to the StoRM WebDAV or StoRM tape service
5. (and 6.) the response from the service is relayed to the user via NGINX



STORM WEBDAV AND STORM TAPE POLICIES DEFINED WITH OPA

OPA will replace the current StoRM WebDAV Policy decision Point (PdP) logic, making it also more compliant with the WLCG JWT Profile

The same rules are applied to the StoRM tape service, almost available in production

The OPA rules (**rego** files) are versioned and published as a bundle. The service operators just need to update the policies (**data** file)

The rules can potentially be used by any storage service which aims to be compliant with the WLCG JWT profile !

An OPA policy (contained in a **`data.yaml`** file) is defined by:

`actions` list of actions the policy is authorizing. Possible values: *list*, *read*, *write*, *delete*, *stage*, *all*

`paths` list of paths the policy applies to. Use '**' to match a directory and all its content. Default to '**'

`principals` list of principals the policy applies to. Possible values:

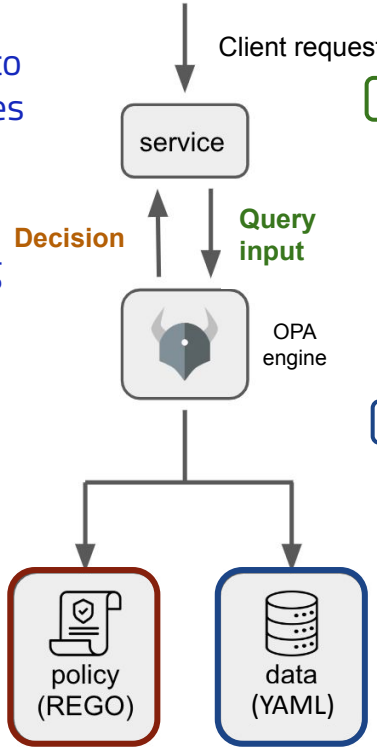
- **`vo`** string of the VO name
- **`fqan`** allowed VOMS FQAN
- **`x509-subject`** certificate subject
- **`jwt-issuer`** token issuer which authorizes the storage operation
- **`jwt-group`** object of the allowed token issuer (*iss*) and group name (*group*)
- **`jwt-subject`** object of the allowed token issuer (*iss*) and subject (*sub*)
- **`jwt-scope`** object of the allowed token issuer (*iss*) and scope (*scope*), which may include a path

OPA source code

QUERYING OPA WITH STORM WEBDAV AND STORM TAPE

A stage bulk-request submission is forwarded to OPA through NGINX in order to compute the files allowed for staging

The original request body (list of *paths*) is also forwarded, in order to match the corresponding storage area



*NGINX performs a POST request to OPA with information about which **files** to recall and which credential is provided. Here a JWT containing **storage.stage:/tape** within its scopes is presented*

1

```
{
  "method": "POST",
  "path": "/api/v1/stage",
  "paths": [
    "/atlas/tape/file",
    "/cms/tape/file"
  ],
  "access_token": "eyJraWQiOiJyc2ExIiwiY..."
}
```

2

```
storage-areas:
- name: cms
  root: /tmp/disk/cms
  access-point: /cms
  policies:
  - actions:
      - stage
    paths:
    - /tape/**
    principals:
    - jwt-scope:
        iss: https://indigo-iam.example/
        scope: storage.stage:/
...
```

*StoRM policies are provided as **data** object. The following policy requires a parametric **storage.stage:/** scope in the JWT in order to submit stage requests to the cms endpoint*

3

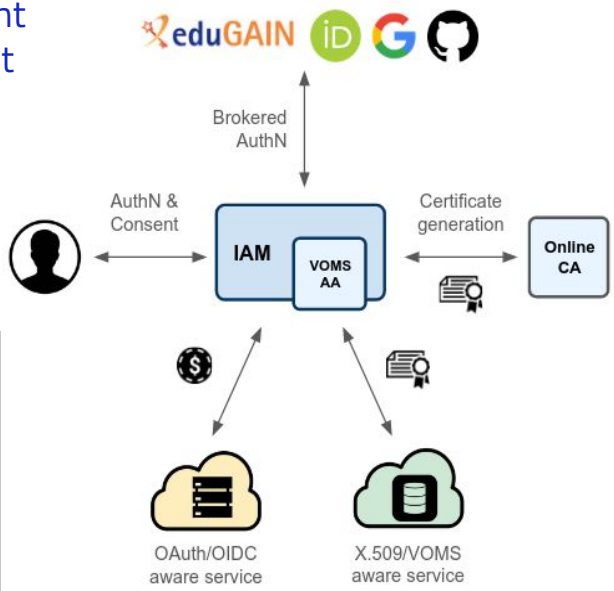
```
{
  "allowed_files": [
    "/cms/tape/file"
  ],
  "denied_files": [
    "/atlas/tape/file"
  ],
  ...
}
```

*A list of **allowed files** is returned to StoRM, together with other information*

INDIGO IAM

INDIGO IAM is an authentication and authorization service which manages user identities, enrollments, group memberships, etc. It simplifies the management of user credentials leveraging on standard and secure OAuth/OpenID Connect protocols

INDIGO IAM issues **JWT tokens** and X.509 Attribute Certificates with identity and membership information, attributes and capabilities



Token capabilities determine the privileges granted to a Client application, expressed as OAuth **scopes**

The IAM Scope Policies provide a mechanism to control access to token scopes

https://wlcg.cloud.cnaf.infn.it/iam/scope_policies
(requires Admin privileges)


```
▼ description:      "Allow access to storage.* scopes to wlcg/xfers users"
  creationTime:     "2019-12-18T15:20:21.000+01:00"
  lastUpdateTime:   "2019-12-18T15:20:21.000+01:00"
  rule:             "PERMIT"
  matchingPolicy:   "PATH"
  account:          null
▼ group:
    uuid:           "f356885a-9d06-4687-b5fe-57322430f111"
    name:           "wlcg/xfers"
  ▼ location:       "https://wlcg.cloud.cnaf.infn.it/scim/Groups/f356885a-9d06-4687-b5fe-57322430f111"
▼ scopes:
    0:              "storage.create:/"
    1:              "storage.read:/"
    2:              "storage.modify:/"
```


Example of a scope policy defined in IAM

INDIGO IAM POLICIES DEFINED WITH OPA

OPA implements and evolves the current IAM PdP logic

- more readable policy definition based on the entity the policy applies to
- policies are also applied to clients, such to support the OAuth *client credentials* flow (not bounded to a user)

The policies definition (on **data** file) is backward compatible with IAM

The **`opa eval --profile`** command (plus further options) has been used to profile the scope policies

An OPA query took **~130 ms** to parse 10k policies, which in IAM reached the client timeout !

METRIC	VALUE
timer_rego_module_compile_ns	52170843
timer_rego_module_parse_ns	12619578
timer_rego_query_compile_ns	716752
timer_rego_query_eval_ns	129958182
timer_rego_query_parse_ns	750061

An OPA policy (contained in a **`data.json`** file) is defined by:

`scopes` list of scopes the policy applies to

`matchingPolicy` algorithm used to compare the requested scopes wrt the ones defined in the policy. Possible values are EQ (string matching), or PATH (parametric scope matching, as described in the WLCG JWT Profile)

`rule` determines the behavior of the policy. Possible values are *PERMIT*, or *DENY*

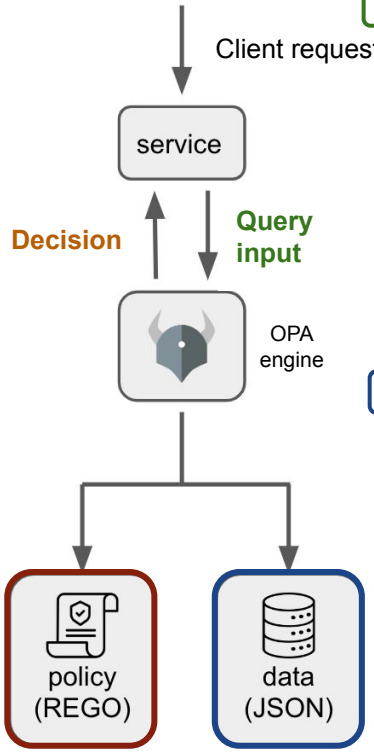
`actor` an object describing the entity the policy applies to (if missing, the policy applies to everyone), identified by

- **`type`** can be *subject* (matching a user or a client entity), or *group*
- **`id`** unique identifier for the subject or group

OPA source code

QUERYING OPA WITH INDIGO IAM

A request for the token scopes *openid* and *storage.read:/protected/file* is forwarded to OPA in order to compute the allowed capabilities (*filtered_scopes*)



1

```
{
  "actor": {
    "groups": [
      "1234"
    ],
    "subject": "999"
  },
  "scopes": [
    "openid",
    "storage.read:/protected/file"
  ]
}
```

*IAM performs a POST request to OPA with information about **who** requested the token and which **scopes** wants to receive*

2

```
[
  {
    "actor": {
      "id": "1234",
      "name": "test",
      "type": "group"
    },
    "matchingPolicy": "PATH",
    "rule": "DENY",
    "scopes": [
      "storage.read:/protected"
    ]
  }
]
```

*IAM policies are provided as **data** object. The following policy denies access to the **storage.read:/protected** parametric scope to the "test" group*

3

```
{
  "denied_scopes": [
    "storage.read:/protected/file"
  ],
  "filtered_scopes": [
    "openid"
  ],
  ...
}
```

*A list of **allowed scopes** is returned to IAM, together with other information*

ICSC

Centro Nazionale di Ricerca in HPC
Big Data and Quantum Computing

Thanks for your attention

USEFUL REFERENCES

- Open Policy Agent documentation
  - OPA Policy testing
  - OPA Policy performance
  - OPA Playground
- OPA source code
  - StoRM Tape AuthN/Z
  - IAM OPA integration
- VS Code plugin for OPA

Bkp

Query OPA

A simulation of IAM call-out to OPA can be done with `curl`

```
$ curl http://localhost:8181 -s -d@assets/opa/input-example.json | jq
{
  "denied_scopes": [
       "storage.modify:/slash/",
       "storage.read:/cms/pippo",
       "storage.read:/slash/pippo"
  ],
  "matched_policy": [
              0
  ],
  "filtered_scopes": [
       "compute.read:/slash/pippo",
       "openid",
       "wlcg.groups:/pippo"
  ],
  …
}
```

IAM performs a POST request with JSON-formatted input data

input-example.json

```
{
   "actor": {
       "subject": "30559491-17b8-4bc8-84b6-7825fb7c89e5",
       "groups": [
           "1234"
       ]
   },
   "scopes": [
       "openid",
       "compute.read:/slash/pippo",
       "storage.read:/slash/pippo",
       "storage.read:/cms/pippo",
       "storage.modify:/slash/",
       "wlcg.groups:/pippo"
   ]
}
```

OPA hierarchical data structure

OPA reorders the rego packages (with variables and rules), data/policies, tests and configuration within a data object

```
$ curl http://localhost:8181/v1/data | jq .result
{
  "default_decision": "rules",
  "policies": [
      {
            "actor": {
                  "id": "1234",
                  "name": "/indigoiam",
                  "type": "group"
            },
            "description": "Deny storage scopes to indigoiam group",
            "matchingPolicy": "PATH",
            "rule": "DENY",
            "scopes": [
                  "storage.read:/",
                  "storage.create:/",
                  "storage.modify:/"
            ]
      },
      ...
}
```

- the dot notation is used to descend through the hierarchy, in order to access the requested variable
- all values generated by rules can be queried via the global **`data`** variable
- **`input`** is a reserved, global variable which binds data provided in the query

OPA testing

OPA also provides a framework that one can use to write tests

- tests are expressed as standard Rego rules where the rule name is prefixed with `test_`
- the `with/as` keywords are used to mock input, data, rules or functions
- run tests with: `opa test <file-or-directory>`, plus optional flags
  - `-v` gives more verbosity
  - `--coverage` reports coverage for the policies under test
  - `--var-values` shows the value of variables causing failures
  - etc.

```
$ opa test /etc/opa/test --var-values -v
opa/test/scope_matching.rego:
data.test.test_eq_matching: PASS (515.35µs)
data.test.test_eq_not_matched: PASS (513.561µs)
...
--------------------------------------------------
PASS: 55/55
```

OPA profiling

`opa eval` command allows to evaluate a Rego query

The `--profile` option can be use to profile the policies, together with further flags, e.g.

- `--profile-sort` option sorts the output by the total time the query has been computed, in nanoseconds (this option includes --`profile`)
- `--format=pretty` enables the output as table format (default is JSON)
- `--count=10` repeats the policy evaluation 10 time and enables statistics results
- `--profile-limit=5` shows 5 lines of profiling results

Among other parameters, the output shows:

- `NUM EVAL` is the number of times an expression is evaluated
- `NUM REDO` is the number of times an expression is re-evaluated(redo)
- `NUM GEN EXPR` is the number of expression generated for a given statement in a particular line
- `timer_rego_query_eval_ns` is the total time OPA took to evaluate the query

OPA profiling example

```
$ opa eval -i assets/opa/input-example.json -d opa/rules -d assets/opa/data-example.json
"data.rules.filtered_scopes" --profile-sort total_time_ns --format=pretty
[
  "openid",
  "wlcg.groups:/pippo"
]
+--------------------------------+---------+
|           METRIC               |  VALUE  |
+--------------------------------+---------+
| timer_rego_data_parse_ns       | 10414   |
| timer_rego_external_resolve_ns | 790     |
| timer_rego_load_files_ns       | 1502719 |
| timer_rego_module_compile_ns   | 5217084 |
| timer_rego_module_parse_ns     | 1261957 |
| timer_rego_query_compile_ns    | 71675   |
| timer_rego_query_eval_ns       | 2139581 |
| timer_rego_query_parse_ns      | 75006   |
+--------------------------------+---------+
+-----------+----------+----------+--------------+-----------------------------------------------+
|   TIME    | NUM EVAL | NUM REDO | NUM GEN EXPR |                LOCATION                       |
+-----------+----------+----------+--------------+-----------------------------------------------+
| 434.803µs | 42       | 0        | 1            | /etc/opa/rules/policy_evaluation_order.rego:26 |
| 411.276µs | 42       | 0        | 1            | /etc/opa/rules/policy_evaluation_order.rego:19 |
| 384.679µs | 42       | 0        | 1            | /etc/opa/rules/policy_evaluation_order.rego:12 |
| 100.568µs | 7        | 0        | 1            | /etc/opa/rules/policy_evaluation_order.rego:36 |
| 90.184µs  | 7        | 0        | 1            | /etc/opa/rules/policy_evaluation_order.rego:33 |
| 89.251µs  | 7        | 0        | 1            | /etc/opa/rules/policy_evaluation_order.rego:50 |
| 77.387µs  | 14       | 14       | 2            | /etc/opa/rules/policy.rego:12                  |
| 76.434µs  | 7        | 0        | 1            | /etc/opa/rules/policy_evaluation_order.rego:40 |
| 71.61µs   | 7        | 0        | 1            | /etc/opa/rules/policy_evaluation_order.rego:56 |
| 65.831µs  | 7        | 0        | 1            | /etc/opa/rules/policy_evaluation_order.rego:47 |
+-----------+----------+----------+--------------+-----------------------------------------------+
```

Update the policies

OPA supports the JSON Patch operation to update a document, as for RFC 6902.
For instance, in order to upload a policy which denies access to IAM admin scopes to the client identified by `1234`, one should submit the following request:

```
$ curl https://opa.test.example/v1/data/policies -k -XPATCH -H "Content-Type:
application/json-patch+json" -d '[{"op": "add", "path": "-", "value": {
    "actor": {
        "id": "1234",
        "name": "client-credentials",
        "type": "subject"
    },
    "description": "Deny access to admin scopes to client 1234",
    "matchingPolicy": "EQ",
    "rule": "DENY",
    "scopes": [
        "iam:admin.read",
        "iam:admin.write"
    ]
  }
}]'
```

Now, the client-vetting policy is appended to the previous ones

Pros & counts

Pros

- very powerful tool !
- easy policy definition language – also for basic developers
- very fast, even without caching
- a lot of documentation
- OPA playground service very useful to start coding and sharing policies among colleagues
- a VS Code plugin (supporting also the Language Server Protocol through Regal) is available to help the development phase
- used in industry
- very well maintained

Cons

- not so many examples in stack overflow for instance, and blogs just apply the documentation
    - but, I have found many suggestion into GitHub issues
    - let’s start all together!

NGINX role in the StoRM Tape deployment

- NGINX is an open-source HTTP server and reverse proxy known for
  - high performance
  - high stability
  - rich feature set
  - simple configuration
  - low resource consumption
- The service has been chosen as part of this deployment for
  - VOMS/TLS termination
  - Authentication with JWT

nginx.conf

```
load_module modules/ngx_http_voms_module.so;
load_module modules/ngx_http_js_module.so;
…
server{
   …
   location /api/v1 {
   auth_request /authz;
   proxy_set_header        X-SSL-Client-S-Dn $ssl_client_s_dn;
   proxy_set_header        x-voms_fqans $voms_fqans;
   …
   proxy_pass              http://storm-tape:8080;
 }
 location /authz {
   internal;
   js_var $trusted_issuers
"https://wlcg.cloud.cnaf.infn.it/,https://cms-auth.web.cern.ch/";
   js_content auth_engine.authorize_operation;
 }
 location /_opa {
   internal;
   …
   proxy_pass http://opa:8181/;
 }
}
```

E.g.: POST https://storm-tape.test.example**/api/v1/stage**

NGINX role in the StoRM Tape deployment

An **auth_engine.js** module has been written at CNAF in order to

- check the presence of a JWT in the HTTP Header and, in case, validate it
- check the presence of X.509/VOMS variables (**voms_fqans**, **ssl_client_s_dn**)
- pass the above data with a POST request to OPA and handle its response

auth_engine.js

```
async function authorize_operation(r) {
  …
  r.subrequest("/_opa", opts, function (opa_res) {

    const body = JSON.parse(opa_res.responseText);

    if (!body || !body.allow) {
      r.return(403);
      return;
    }
    r.return(200);
  }
}
```

nginx.conf

```
load_module modules/ngx_http_voms_module.so;
load_module modules/ngx_http_js_module.so;
…
server{
   …
   location /api/v1 {
   auth_request /authz;
   proxy_set_header        X-SSL-Client-S-Dn $ssl_client_s_dn;
   proxy_set_header        x-voms_fqans $voms_fqans;
   …
   proxy_pass              http://storm-tape:8080;
 }
 location /authz {
   internal;
   js_var $trusted_issuers
"https://wlcg.cloud.cnaf.infn.it/,https://cms-auth.web.cern.ch/";
   js_content auth_engine.authorize_operation;
 }
 location /_opa {
   internal;
   …
   proxy_pass http://opa:8181/;
 }
}
```

NGINX+VOMS role in the StoRM Tape deployment

- **ngx_http_voms_module** is a module for NGINX which
  - enables client-side authentication based on X.509 proxy certificates
  - developed at INFN-CNAF
- it defines a set of embedded variables whose values are extracted from the Attribute Certificate
  - *e.g.* the **`voms_fqans`**

```
subject   : /DC=org/DC=terena/DC
issuer      : /DC=org/DC=terena/
identity  : /DC=org/DC=terena/DC
type        : RFC3820 compliant
strength  : 2048
path        : /tmp/x509up_u1000
timeleft  : 00:59:35
key usage : Digital Signature, K
=== VO wlcg extension informatic
VO          : wlcg
subject   : /DC=org/DC=terena/DC
issuer      : /DC=org/DC=terena/
attribute : /wlcg
attribute : /wlcg/mc
attribute : /wlcg/pilots
attribute : /wlcg/xfers
timeleft  : 11:59:53
uri         : wlcg-voms.cloud.cr
```

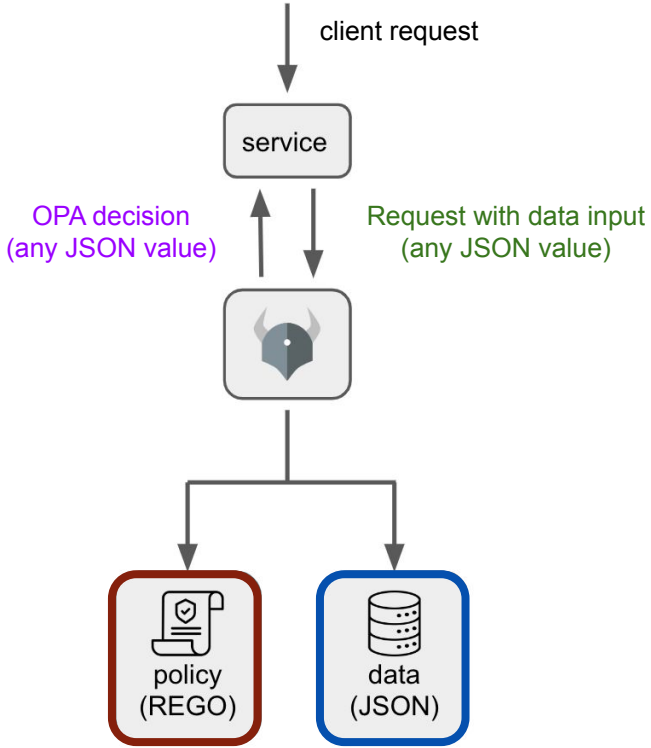
OPA role in the StoRM Tape deployment: example

```
{
  "method": "GET",
  "path": "/api/v1/stage/9a8e34bd-73fe-4b43-9139-1c5f6711577c",
  "client_s_dn": "CN=test0,O=IGI,C=IT"
}
```

```
{
  "allowed_dn": [
  "CN=John Doe jhondoe@infn.it,O=Istituto Nazionale di Fisica
Nucleare,C=IT,DC=tcs,DC=terena,DC=org",
  "CN=test0,O=IGI,C=IT"
  ],
  …
}
```

```
# GET /api/v1/stage/<id>
allow if {
    input.method == "GET"
    glob.match("/api/v1/stage/*", ["/"], input.path)

    any([read_scopes_allowed, voms_fqans_allowed, certificate_dn_allowed])
}
```

has allowed WLCG scopes? OR has allowed FQANs? OR has allowed DN?



OPA role in the StoRM Tape deployment: example

```
{
  "method": "GET",
  "path": "/api/v1/stage/9a8e34bd-73fe-4b43-9139-1c5f6711577c",
  "client_s_dn": "CN=test0,O=IGI,C=IT"
}
```

```
{
  "allowed_dn": [
  "CN=John Doe jhondoe@infn.it,O=Istituto Nazionale di Fisica
Nucleare,C=IT,DC=tcs,DC=terena,DC=org",
  "CN=test0,O=IGI,C=IT"
  ],
  …
}
```

```
# GET /api/v1/stage/<id>
allow if {
    input.method == "GET"
    glob.match("/api/v1/stage/*", ["/"], input.path)

    any([read_scopes_allowed, voms_fqans_allowed, certificate_dn_allowed])
}
```

has allowed WLCG scopes? ✗ OR has allowed FQANs? ✗ OR has allowed DN? ✓

https://storm.test.example/api/v1/stage/9a8e34bd-73fe-4b43-9139-1c5f6711577c

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

```
id:          "9a8e34bd-73fe-4b43-9139-1c5f6711577c"
created_at:  1682073801
started_at:  0
files:
  0:
    path:    "/wlcg/test1.txt"
    state:   "SUBMITTED"
  1:
    path:    "/wlcg/test2.txt"
    state:   "SUBMITTED"
```

```
{
  "allow": "true"
}
```

Example of IAM scope policies

https://wlcg.cloud.cnaf.infn.it/iam/scope_policies

(requires Admin privileges)


```
0:
  id:              1
  description:     "Default Permit ALL policy"
  creationTime:    "2019-10-08T13:52:20.000+02:00"
  lastUpdateTime:  "2019-10-08T13:52:20.000+02:00"
  rule:            "PERMIT"
  matchingPolicy:  "EQ"
  account:         null
  group:           null
  scopes:          null
1:
  id:              4
  description:     "Deny access to compute.* scopes to normal users"
  creationTime:    "2019-12-18T15:11:04.000+01:00"
  lastUpdateTime:  "2019-12-18T15:11:04.000+01:00"
  rule:            "DENY"
  matchingPolicy:  "EQ"
  account:         null
  group:           null
  scopes:
    0:             "compute.create"
    1:             "compute.read"
    2:             "compute.cancel"
    3:             "compute.modify"
2:
  id:              7
  description:     "Deny access to storage.* scopes to normal users"
  creationTime:    "2019-12-18T15:12:49.000+01:00"
  lastUpdateTime:  "2019-12-18T15:12:49.000+01:00"
  rule:            "DENY"
  matchingPolicy:  "PATH"
  account:         null
  group:           null
  scopes:
    0:             "storage.create:/"
    1:             "storage.read:/"
    2:             "storage.modify:/"
```


`compute` scopes allowed only to `wlcg/pilot` group

`storage` scopes allowed only to `wlcg/xfer` group


```
3:
  id:              13
  description:     "Allow access to compute.* scopes to wlcg/pilot users"
  creationTime:    "2019-12-18T15:19:20.000+01:00"
  lastUpdateTime:  "2019-12-18T15:19:20.000+01:00"
  rule:            "PERMIT"
  matchingPolicy:  "EQ"
  account:         null
  group:
    uuid:          "25084f30-1d71-4ab2-91e8-11148af16682"
    name:          "wlcg/pilots"
    location:      "https://wlcg.cloud.cnaf.infn.it/scim/Groups/25084f30-1d71-4ab2-91e8-11148af16682"
  scopes:
    0:             "compute.create"
    1:             "compute.read"
    2:             "compute.cancel"
    3:             "compute.modify"
4:
  id:              16
  description:     "Allow access to storage.* scopes to wlcg/xfers users"
  creationTime:    "2019-12-18T15:20:21.000+01:00"
  lastUpdateTime:  "2019-12-18T15:20:21.000+01:00"
  rule:            "PERMIT"
  matchingPolicy:  "PATH"
  account:         null
  group:
    uuid:          "f356885a-9d06-4687-b5fe-57322430f111"
    name:          "wlcg/xfers"
    location:      "https://wlcg.cloud.cnaf.infn.it/scim/Groups/f356885a-9d06-4687-b5fe-57322430f111"
  scopes:
    0:             "storage.create:/"
    1:             "storage.read:/"
    2:             "storage.modify:/"
```

Project folder tree

Source code

```
opa
├── config.yaml
├── policies
│   └── data.json
├── rules
│   ├── entity_matching
│   │   └── entity_matching.rego
│   ├── entity_type.rego
│   ├── policy_evaluation_order.rego
│   ├── policy.rego
│   └── scope_matching.rego
└── test
    ├── entity_matching
    │   └── entity_matching.rego
    ├── entity_type.rego
    ├── policy_evaluation_order.rego
    ├── policy.rego
    └── scope_matching.rego
```

Contains OPA configuration: default decision, authentication to OPA, enable caching results, etc.

A list of policy objects

Policy entity matcher (based on `actor.id/type`)

One rego file per operation

One test file per rego file

To do

Development:

- add **audience** policies:
  - e.g. the `https://wlcg.cern.ch/jwt/v1/any` audience can be obtained only by a certain group
- implement a real path algorithm to match path-parametric scopes
  - it is now just a prefix match of the requested scope
  - only scopes that matched a prefix plus "/" should be allowed
  - the rule matching the longest path wins
    - e.g. a policy on the `storage.read:/home` overrides the one defined for the `storage.read:/` scope

Deployment:

- deploy a test IAM instance which supports OPA
  - deployment model is now only based on docker-compose and includes only OPA
  - play with OPA configuration (e.g. caching) to enhance performances
- decide which authentication mechanism apply to whom operates OPA (e.g. for adding policies)
  - OPA supports Bearer Authentication, Basic Authentication, etc.