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

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

# Open Policy Agent as authorization engine in the Grid middleware

Federica Agostini

Workshop sul Calcolo nell'INFN, Palau, 20 - 24 maggio 2024

## OPEN POLICY AGENT

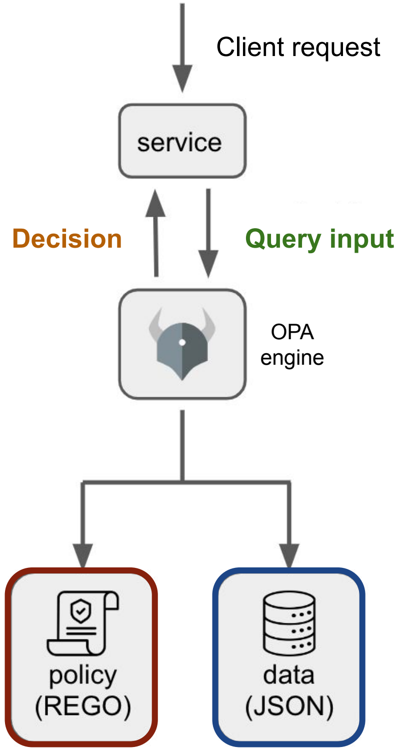
Open Policy Agent (OPA) is an open-source authorization engine

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

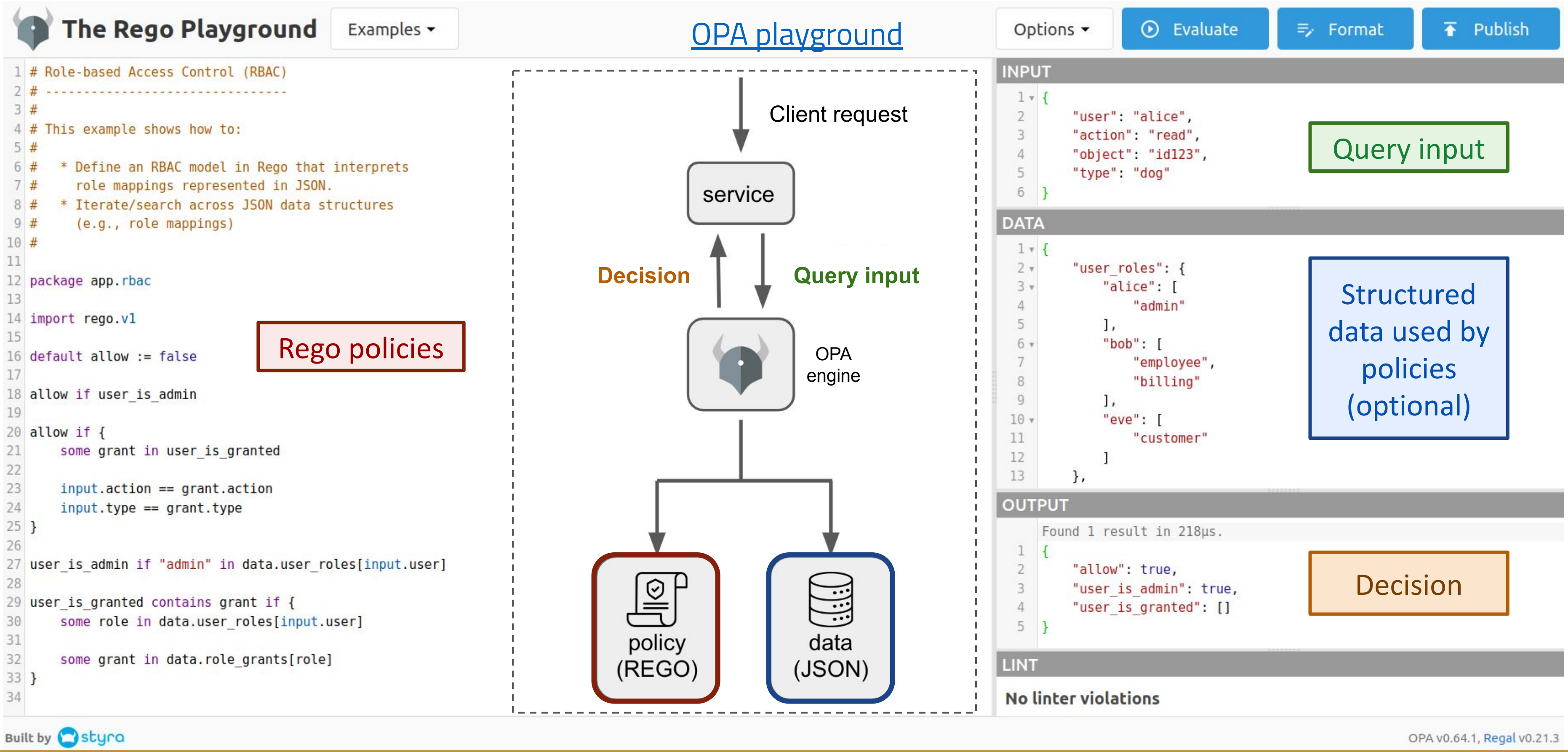
- rego is designed for expressing policies over complex hierarchical data structures
  - great performance thanks to this optimization

A service which needs to take a policy decisions can **query** OPA with arbitrary structured data (e.g., JSON) 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



## OPA playground



```
# 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]
}
```

INPUT

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

DATA

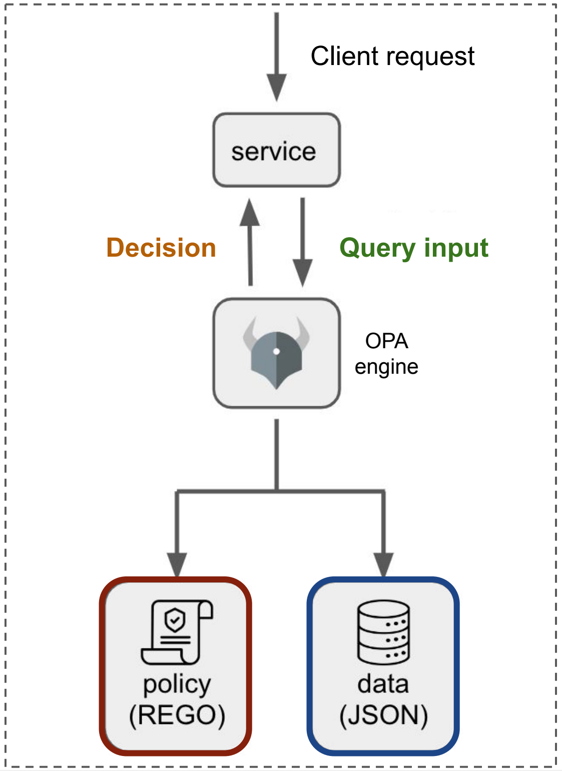
```
{
    "user_roles": {
        "alice": [
            "admin"
        ],
        "bob": [
            "employee",
            "billing"
        ],
        "eve": [
            "customer"
        ]
    },
```

OUTPUT

```
Found 1 result in 218µs.
{
    "allow": true,
    "user_is_admin": true,
    "user_is_granted": []
}
```

LINT

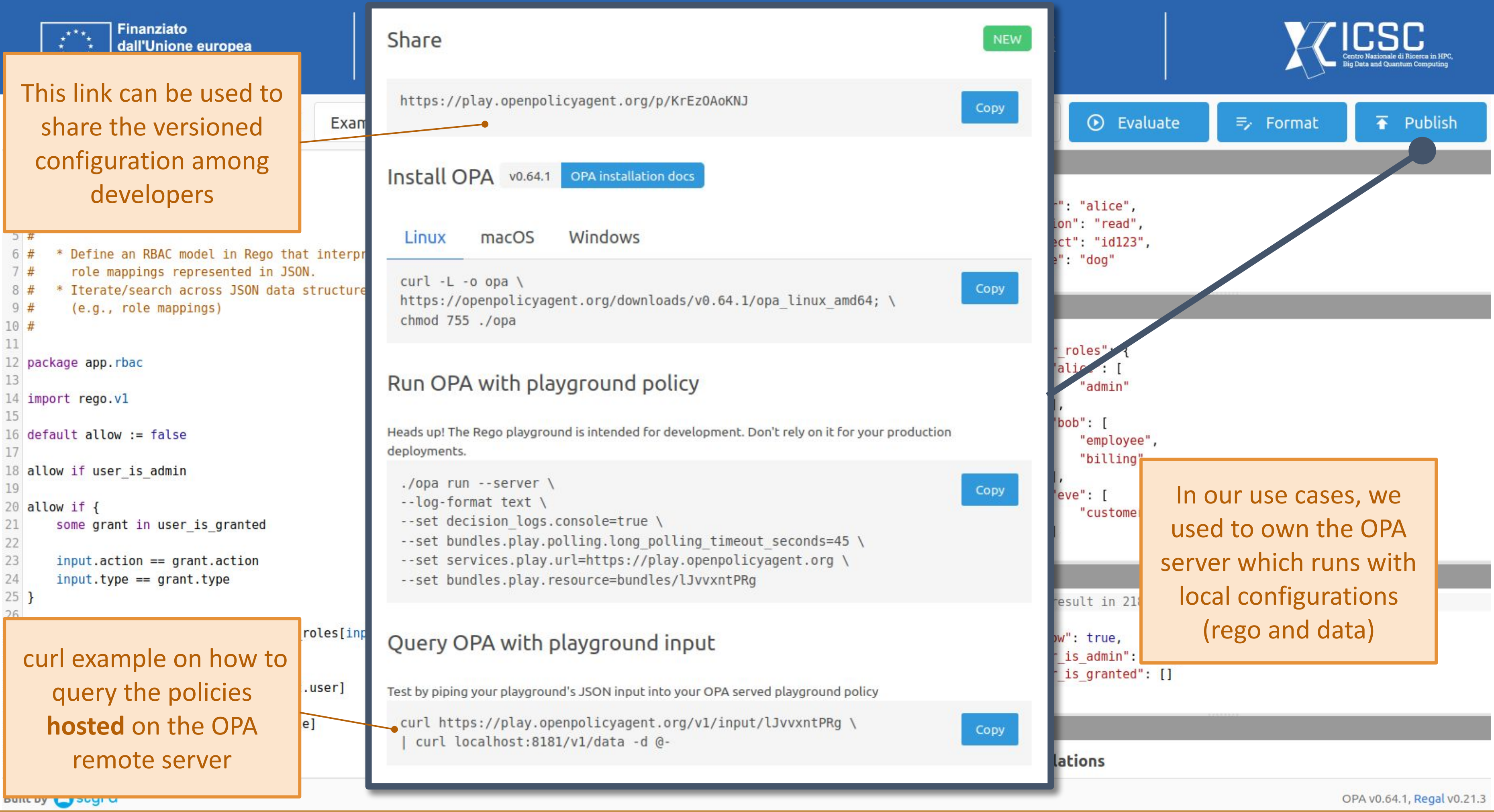
No linter violations



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

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)



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

## Run OPA with playground policy

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

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

```
./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
```

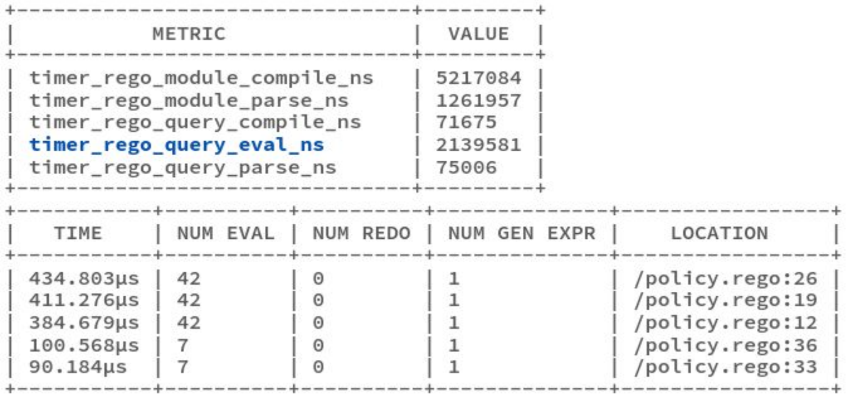
## Query OPA with playground input

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 @-
```

```
#
#   * 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
}
```

## OPA PROFILING and TESTING



```
+---------------------------------+----------+
|             METRIC              |  VALUE   |
+---------------------------------+----------+
| 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            | /policy.rego:26  |
| 411.276µs  | 42       | 0        | 1            | /policy.rego:19  |
| 384.679µs  | 42       | 0        | 1            | /policy.rego:12  |
| 100.568µs  | 7        | 0        | 1            | /policy.rego:36  |
| 90.184µs   | 7        | 0        | 1            | /policy.rego:33  |
+------------+----------+----------+--------------+------------------+
```

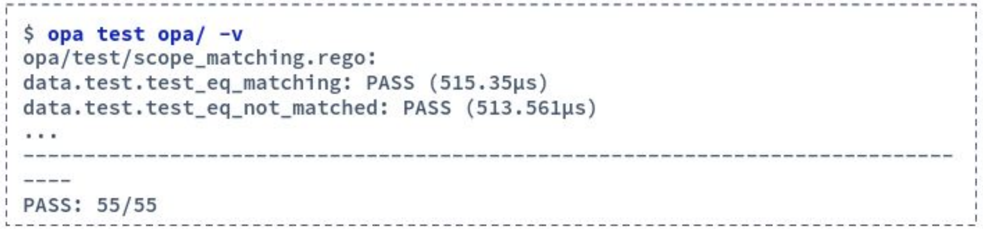
**opa eval** command allows to evaluate a Rego query

The **--profile** option can be used to profile the policies

Some further option can be used to manipulate the output and show statistical informations

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

**opa test** command (plus further optional parameters) allows to run tests, expressed as standard rego rules prefixed with *test_*



```
$ opa test opa/ -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
```

## Integration of OPA into GRID middleware

## INTEGRATION OF OPA WITH GRID MIDDLEWARE

The StoRM Tape REST API

- it is the StoRM implementation of the WLCG Tape REST API, which allows to recall files stored on tape
- OPA is used in this deployment for authorization based on **X509/VOMS proxies** or **JWT tokens**
- only specific DNs, FQANs and scopes are **allowed** to submit the request

The INDIGO IAM service

- IAM Scope Policies provide a mechanism to **control access to OAuth scopes**
- OPA evolves the current IAM PdP logic – e.g. policies are applied to users/groups as in IAM, but also to clients
- the policies definition (on **data** file) is backward compatible with IAM

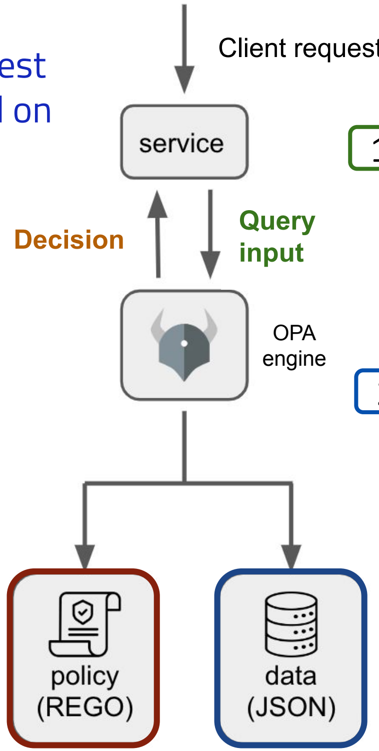
The StoRM Webdav service

- it supports **WLCG JWT scope based authorization**, together with a finer-grained authorization engine
- OPA will replace the current PdP logic, making it also more compliant with the WLCG JWT Profile
- it can potentially be used by any storage service which aims to apply the storage scope rules expressed by WLCG JWT profile

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

# EXAMPLE OF OPA INTEGRATION INTO THE STORM TAPE REST API DEPLOYMENT

OPA authorization example of a stage bulk-request submission done with an allowed JWT (i.e. based on **write scopes**)



*The JWT contains within its scopes a* ***storage.stage:/***

1
```
{
  "method": "POST",
  "path": "/api/v1/stage",
  "access_token":
"eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ..."
}
```

2
```
{
  "scope": {
      "read": [
        "storage.stage:/*",
        "storage.read:/*"
      ],
      "write": [
        "storage.stage:/*"
      ]
   },
   ...
}
```

*In order to read/write one needs to have one of this list of allowed scopes*

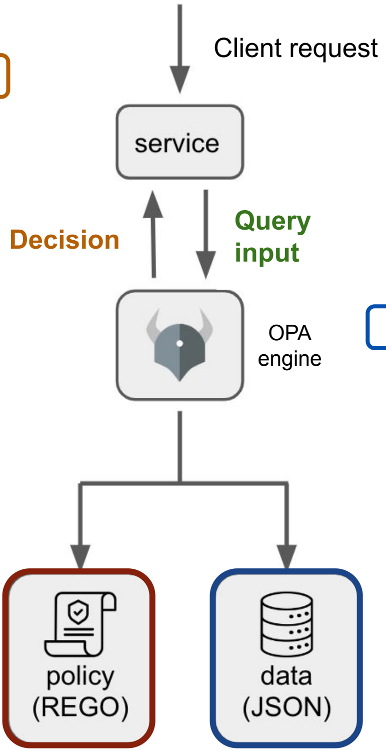
4 Decision
```
{
  "allow": "true"
}
```

*A policy for POST requests to the stage endpoint has been defined*

3
```
# POST /api/v1/stage
allow if {
    input.method == "POST"
    input.path == "/api/v1/stage"

    true in [write_scopes_allowed,
      voms_fqans_allowed]
}
```

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

## EXAMPLE OF OPA QUERY FROM INDIGO IAM SERVICE



1

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

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

2

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

*IAM policies are provided as **data** object. The format can be both the current supported JSON and a new extended format*

3

```
{
  "denied_scopes": [
    "storage.read:/protected/file"
  ],
  "filtered_scopes": [
    "openid"
  ],
  "matched_policy": [
    0
  ],
  "matched_policies_by_scope": {
    "storage.read:/protected/file": [
      {
        "group": "DENY"
      },
  ...
}
```

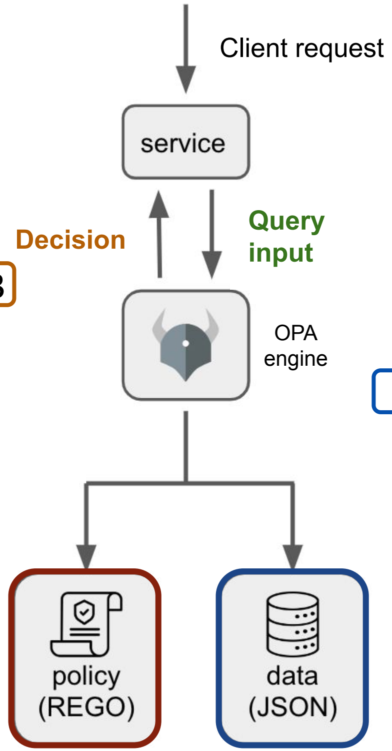
*A list of allowed scopes is returned to IAM, together with other information*

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

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

## PoC OF OPA QUERY FROM STORM WEBDAV

Use case of a client request to StoRM Webdav willing to read the resource `/pippo/pluto`



1
```
{
  "exists": "true",
  "method": "GET",
  "token": {
    "aud": "https://wlcg.cern.ch/jwt/v1/any",
    "scope": "openid storage.read:/pippo",
    "wlcg.groups": [
      "/indigo-iam"
    ],
    ...
  },
  "uri": "https://webdav.example/pippo/pluto"
}
```

*StoRM WebDAV knows if the resource (**uri**) to which the client wants to access exists, thus sends this information to OPA, together with the received header*

2
```
{
  "webdav_hosts": [
    "https://webdav.example"
  ]
  "read_methods": [
    "HEAD",
    "GET",
    "OPTIONS",
    "PROPFIND"
  ],
  ...
}
```

*A list of read, create and modify WebDAV operations matching the input **method** will result in permissions according with the token scopes*

3
```
{
  "allow": true,
  "allowed_read_operation": true,
  "audience_is_present": true,
  "mandatory_claims_are_present": true,
  "resource": "/pippo/pluto",
  "token_scopes": [
    "openid",
    "storage.read:/pippo"
  ],
  "wlcg_groups_are_present": true
}
```

*The answer from OPA is an allow/deny. StoRM Webdav does not need to make further checks, just honour the request*

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

# *Supercomputing shaping the future*

## USEFUL REFERENCES

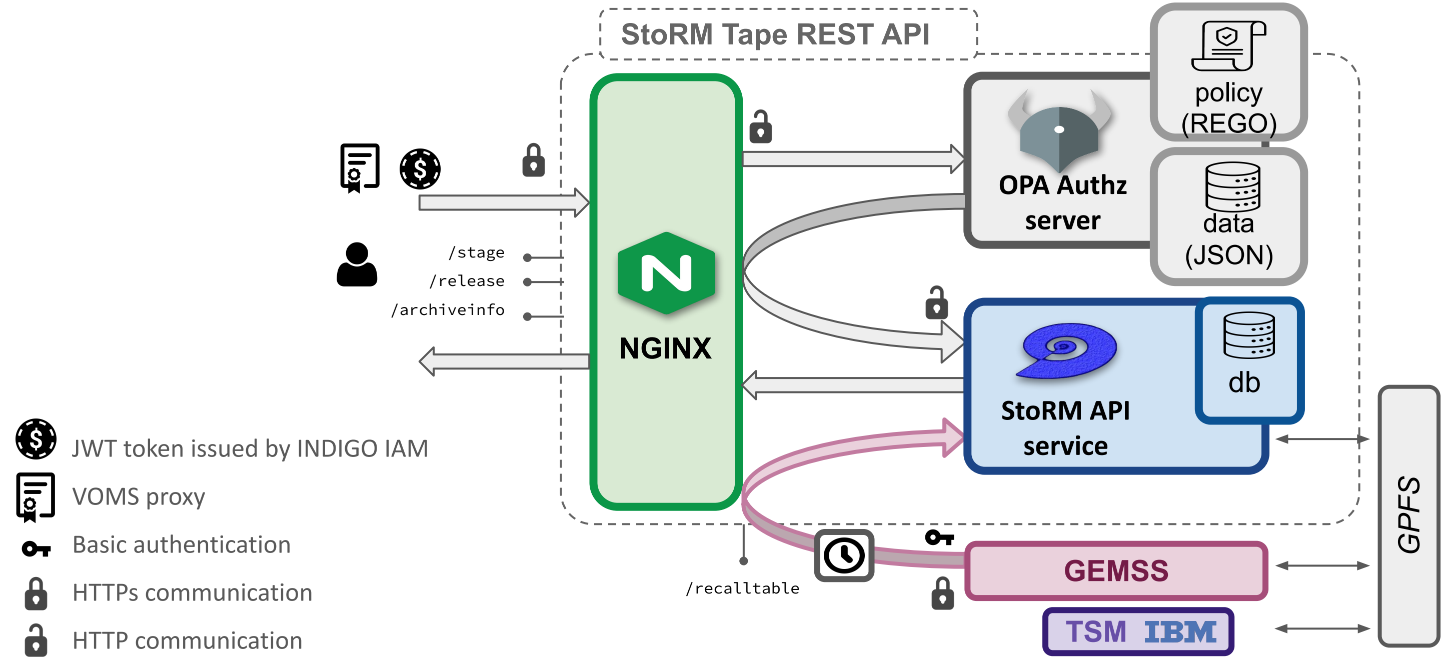
- Open Policy Agent documentation
  - OPA Policy testing
  - OPA Policy performance
  - OPA Playground
- Integration with OPA: source code
  - StoRM Tape REST API deployment
  - INDIGO IAM-OPA integration
  - Compliance with WLCG JWT Profile
- Examples of OPA playgrounds
  - StoRM Tape REST API deployment
  - INDIGO IAM-OPA integration
  - Compliance with WLCG JWT Profile

## Bkp

## StoRM Tape REST API: deployment

The StoRM Tape REST API relies on external components for authN/Z

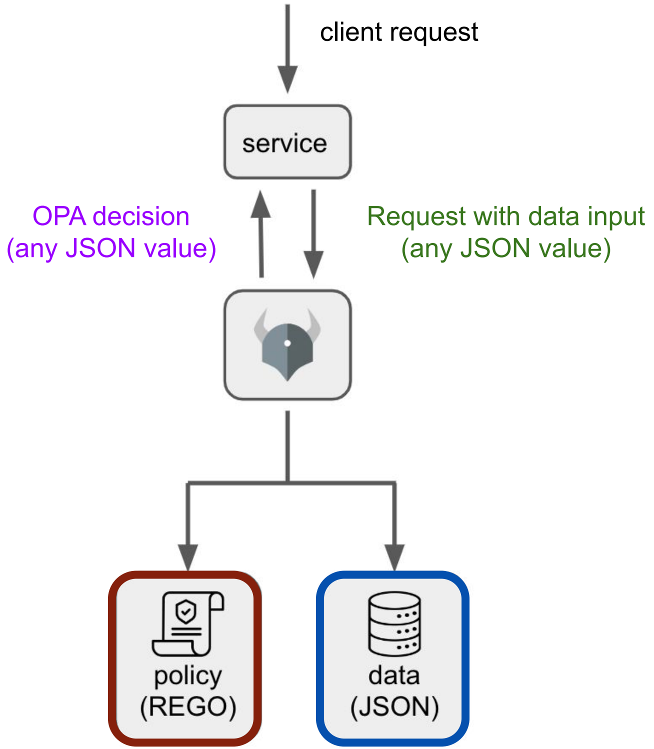
- NGINX → authentication
- OPA → authorization



From CHEP 2023 poster session

## OPA role in the StoRM Tape deployment

- Open Policy Agent (OPA) is an open-source authorization engine that
  - unifies policy enforcement across the stack
  - is based on an high-level declarative language
  - allows the definition of policies as code
- Deployed and tested at INFN-CNAF for **authorization** with X509/VOMS or JWT
- It seems flexible enough to replace other authorization engines
  - *e.g.* Argus



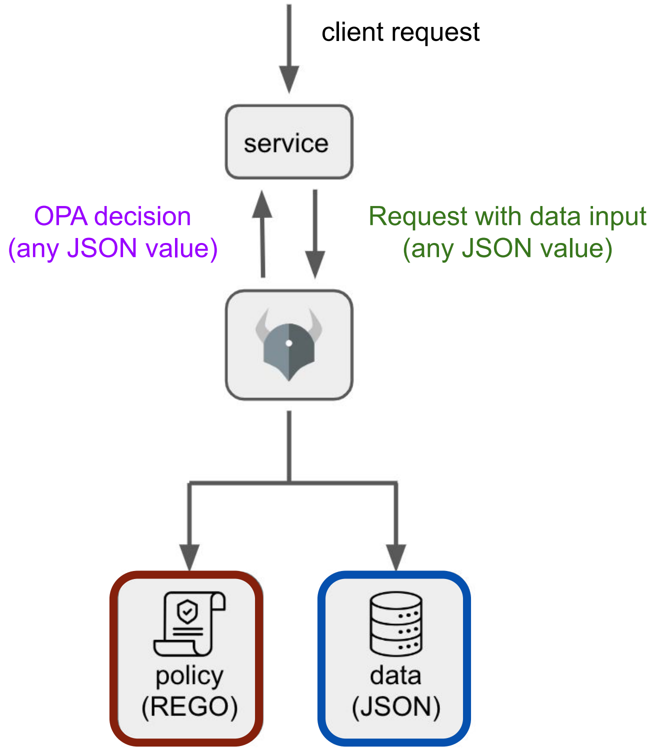
## OPA role in the StoRM Tape deployment: example

```
{
  "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?



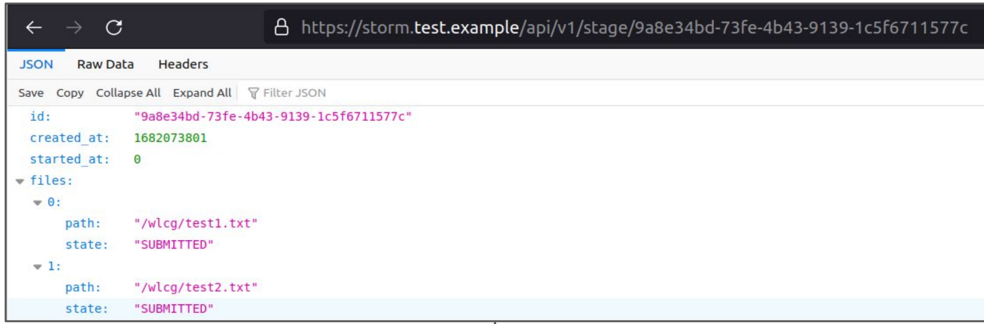
# OPA role in the StoRM Tape deployment: example

```
{
  "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"
}
```

## IAM Scope Policy

IAM Scope policies provide a mechanism to control access to OAuth scopes (documentation).

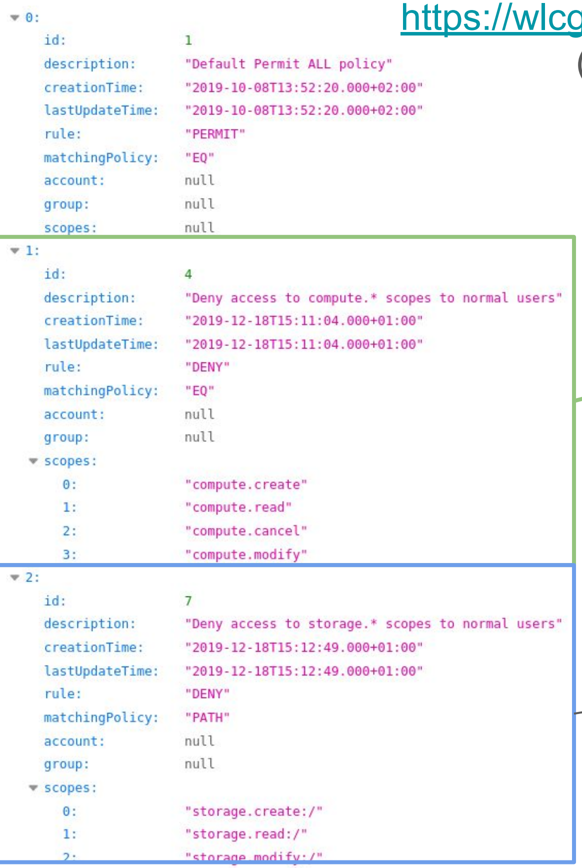
A scope policy defines:

- a *rule* that determines the behaviour of the policy
  - `PERMIT` or `DENY`
- a *scopes selector*, i.e. a set of scopes for which the policy applies
  - e.g. `storage.read:/cms`
- a scope `matchingPolicy` used to determine the scope matching algorithm
  - `EQ`, `PATH` or `REGEXP`
- an *account* or *group selector*, used to determine for which user account or group of accounts the policy should apply

**Order matters**: the account-level policies are applied first, then group-level policies are applied and finally policies that are not bound to any specific account or group are applied

# Example of IAM scope policies

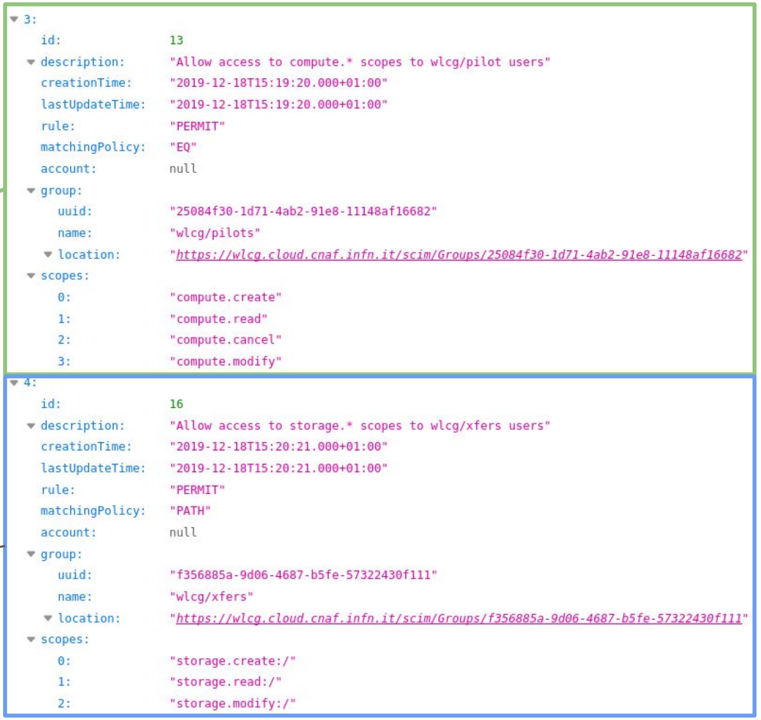
https://wlcg.cloud.cnaf.infn.it/iam/scope_policies
(requires Admin privileges)



```
▼ 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:/"
```

## How OPA policies evolves current IAM PDP logic

- Policies definition (on **data** file) is backward compatible with IAM
    - but a more readable policies definition based on the entity to whom the policy is applied is supported
    - `actor.type` can be “subject” or “group”
        - a “subject” identifies a user or client entity
    - `actor.id` identifies the uuid of the subject, or group
- Added client policies for the use case of `client_credentials` grant (no user is involved)
    - clients are identified by `actor.type=subject` && `actor.id=<client-uuid>`
    - it has same priority as account (i.e. it applies before group entity or policies not bounded to any entity)
- `REGEXP` matching algorithm <u>has been removed</u>
    - we never saw it used in production, and
    - regexps could be dangerous

## 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

# Current 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
```

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

data.json → A list of policy objects

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

One rego file per operation

One test file per rego file

## 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

## 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"
   ]
}
```

## 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` keyword can be used in tests to replace the data document or called functions with mocks
- run tests with: `opa test <file-or-directory>`
  - all rules prefixed with `test_` found in Rego are tested
  - add `-v` option for more verbosity
  - add `--coverage` option to also report coverage for the policies under test

```
$ opa test opa/ -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

- `--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
- etc.

Among other results, 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)
- `timer_rego_query_eval_ns` is the total time OPA took to evaluate the query

**OPA took ~130 ms to parse 10k policies, which in IAM was reaching the `oidc-agent` timeout !**

## 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 |
+-----------+----------+----------+--------------+-------------------------------------------------+
```

## 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.

## 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
- 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!