# Flexible Authorization for Keycloak with Open Policy Agent

## Overview and Use Cases

Thomas Darimont

# About me


- Thomas Darimont
- Principal Consultant codecentric
- Open Source Enthusiast
- Keycloak Contributor for over 9 years
- Official Keycloak Maintainer

# Authentication & Authorization

- Authentication (AuthN)
  - ***who*** the user ***is***
  - ***Identity*** (Alice, Bob, …, Guest)

- **Authorization (AuthZ)**
  - ***what*** the user ***is allowed*** to do
  - ***Entitlements*** (payroll:access, contact:export, etc.)
  - ***Access Control*** (can $user access $client?)

# Authorization in Keycloak

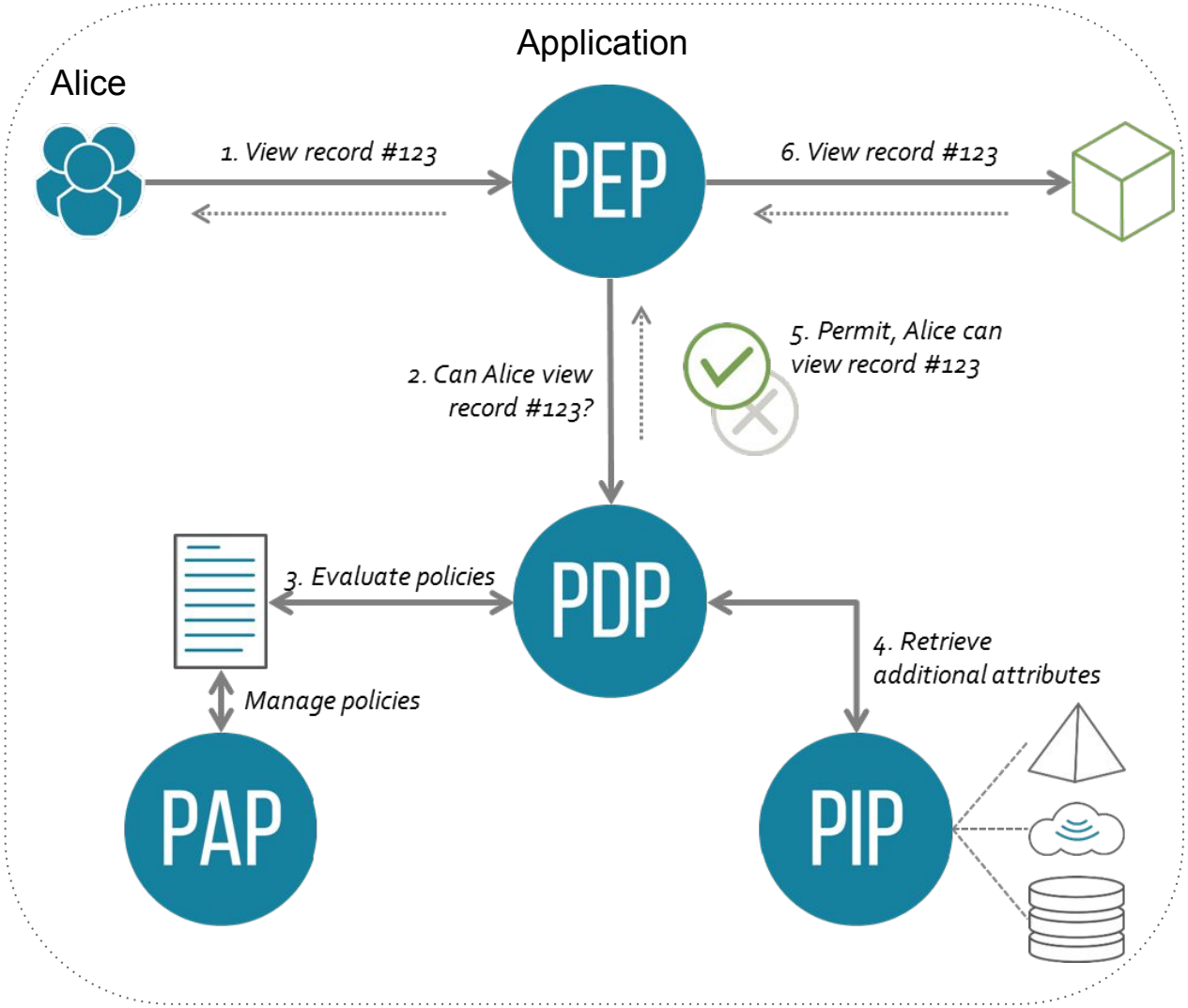
- **Realm Management Roles**
  - Coarse-Grained Roles
- **Admin Permissions**
  - Fine-Grained control over Realm Resources
  - based on *Authorization Services*
- **Authorization Services** (AuthZ Services)
  - Subsystem for flexible Access-Control Policies
  - Built-in Rules, JavaScript, Custom Rules via SPI
- **Custom Extensions**
  - Control Client access via Java / JavaScript
  - Community Extensions, e.g. Restrict Client Auth

# Supported Access-Control Mechanisms

- **Role-based access control (RBAC)**
  - Allow Users with "`Member`" Realm/Client-Role
- **Attribute-based access control (ABAC)**
  - Allow User with Attribute `plan=premium`
- **User-based access control (UBAC)**
  - Allow Users Alex, Cindy & Bert
- **Context-based access control (CBAC)**
  - Access only during work-week 9-5 (Time)
  - Allow access from certain network (Request)
- **Policy-based access control (PBAC)**
  - Access is defined and checked via Policies / Rules

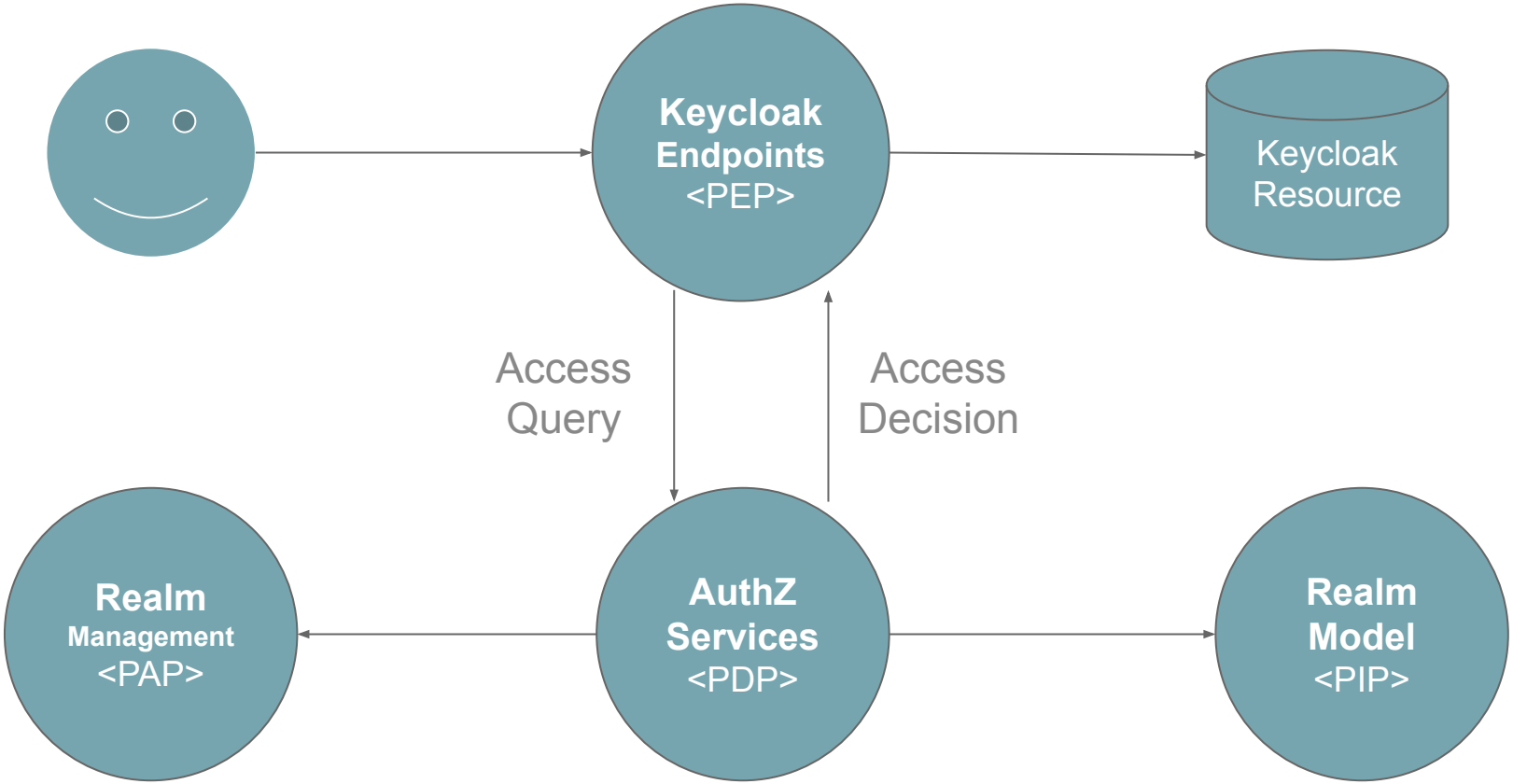
# Policy Based Access-Control Components

- **PEP**
  ***Policy Enforcement Point***
  Requests and applies policy decision.
- **PDP**
  ***Policy Decision Point***
  Evaluates policy with input from request.
- **PAP**
  ***Policy Administration Point***
  Policy life-cycle management.
- **PIP**
  ***Policy Information Point***
  Provides additional information for policy evaluation.




Source

# Keycloak Access-Control Components



# Authorization Services Challenges

- **Only** for **Confidential Clients** (currently)
- Required for **Fine-Grained Admin Permissions**
- Required for **OAuth Token-Exchange**
- **No** built-in **Support for Restricting Client Access**
- In Practice **difficult to use** and hard to maintain
- Hard to manage via Configuration as Code Tools
- Authorization is (currently) not a “First Class Citizen”?

# Authz Services Alternative / Enhancement

We want to:

- Define Policies via Code, e.g. Access Rules
- Validate & Test Policies
- Change Policies easily
- Trace and follow Policy Decisions

… how can we achieve that?

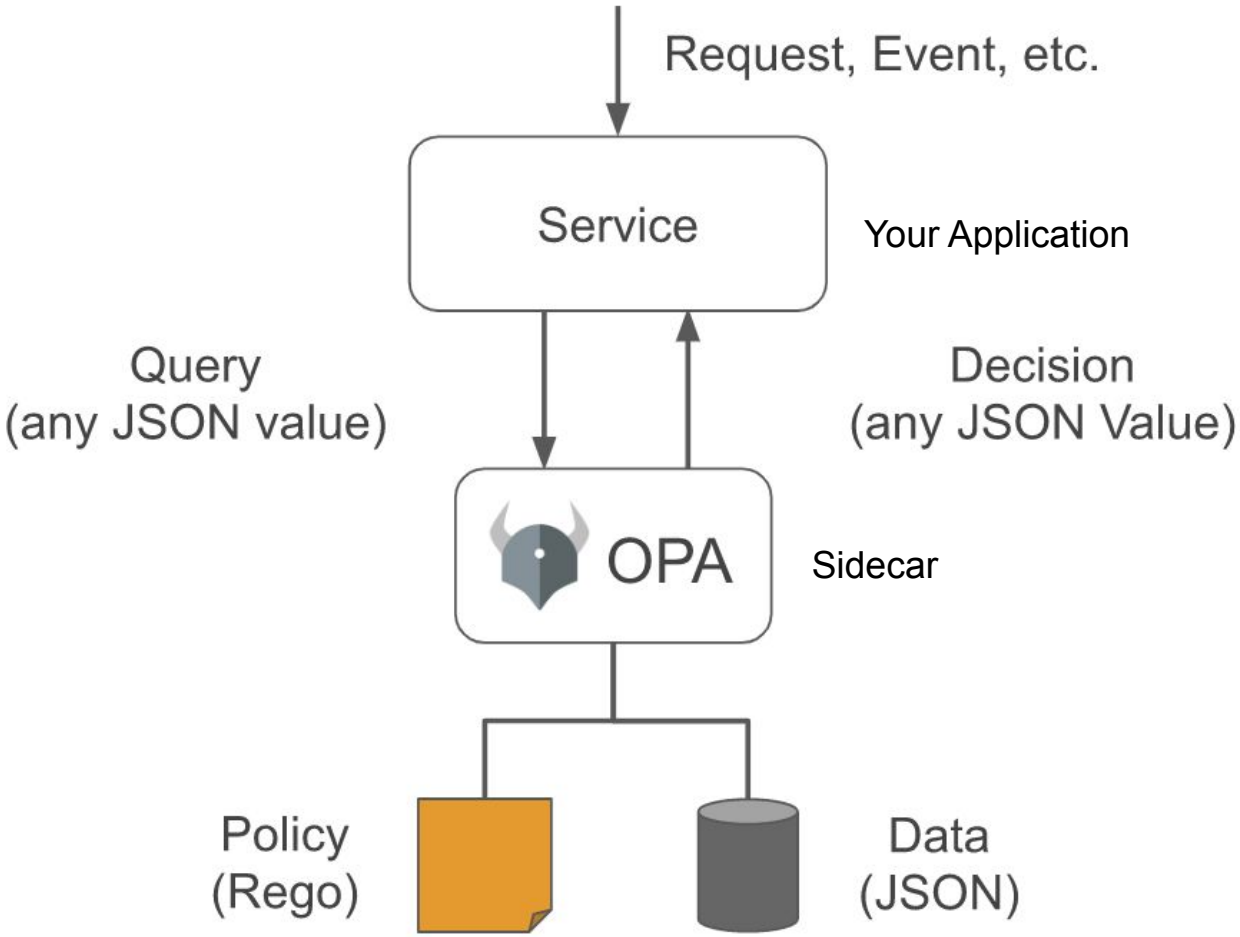
# Open Policy Agent

# Open Policy Agent Overview

- Open-Source **policy engine** written in Go
  - Evaluate Policies, Validate, Supports Pull / Push
- Developed by Styra Inc. and Community (CNCF)
- "***Policy as Code***"
  - Policy / Authorization Logic as Source Code
  - version, lint, test, refactor, audit, run
  - Declarative Policy Language: **Rego**
- **Deployment Options**
  - **Library** - Go library
  - **Sidecar** - Container alongside your App / Gateway

https://www.openpolicyagent.org

# How does OPA work?




Source

# Rego Policy Language

- Declarative DSL, inspired by Datalog
- **Policy** = Collection of **Rules**
- **Rule** = Named Collection of **Conditions**
- **Condition** = Boolean Expressions, Calculations
- Implicit Variables **`input`** and **`data`** for Data Access
- Many built-in Functions (decode JWT, parse JSON, cidr_net, etc.)

Policy File *authz.rego*:

```
1 package myapp.authz
2
3 default allow := false
4
5 allow { # allow IF
6     input.method == "GET"    #     HTTP Method is GET
7     input.path == ["public"]  # AND HTTP Request path is /public
8 }
```

Rules

Conditions

# OPA Policy Queries

- Policy Selection
  - Policy and Rule selected via URL Request Path
  - https://opa/v1/data/path/to/policy/rule

- Policy Input
  - `{ “input”: data }`
  - `data` can be any JSON Object
  - e.g. Identity, Data from Application or Request

- Executing Policy Queries
  - `GRPC / REST API Interface`
  - `POST` https://opa/v1/data/kc/realms/opademo/access/allow
  - `{ “input”: subject | action | resource | context }`

# Demo: Simple Rego Policy

The Rego Playground

Examples ▾ Options ▾ Evaluate Format

```
package app.demo

import future.keywords.in

default allow := false

allow {
    input.method == "GET"
    input.path == "/public"
}

allow {
    input.method == "GET"
    input.path == "/admin"
    "admin" in input.subject.roles
}
```

INPUT

```
{
  "method": "GET",
  "path": "/admin",
  "subject": {
    "name": "bob",
    "roles": ["user","admin"]
  }
}
```

DATA

OUTPUT

```
Found 1 result in 155µs.
{
    "allow": true
}
```

# Open Policy Agent with Keycloak

- **Idea**
  - **Keycloak** as **PEP**, **OPA** as **PDP**
  - Use **Rego** to **define Policies** in Keycloak
  - Not limited to just access control logic!
- **Use-Cases**
  - Fine-Grained Admin Permissions
  - Allow / Reject Identity Brokering
  - OIDC / SAML Artifacts Mapping (Claims / Assertions)
  - Manage Access for Custom Endpoints
  - **Client Access Checks**

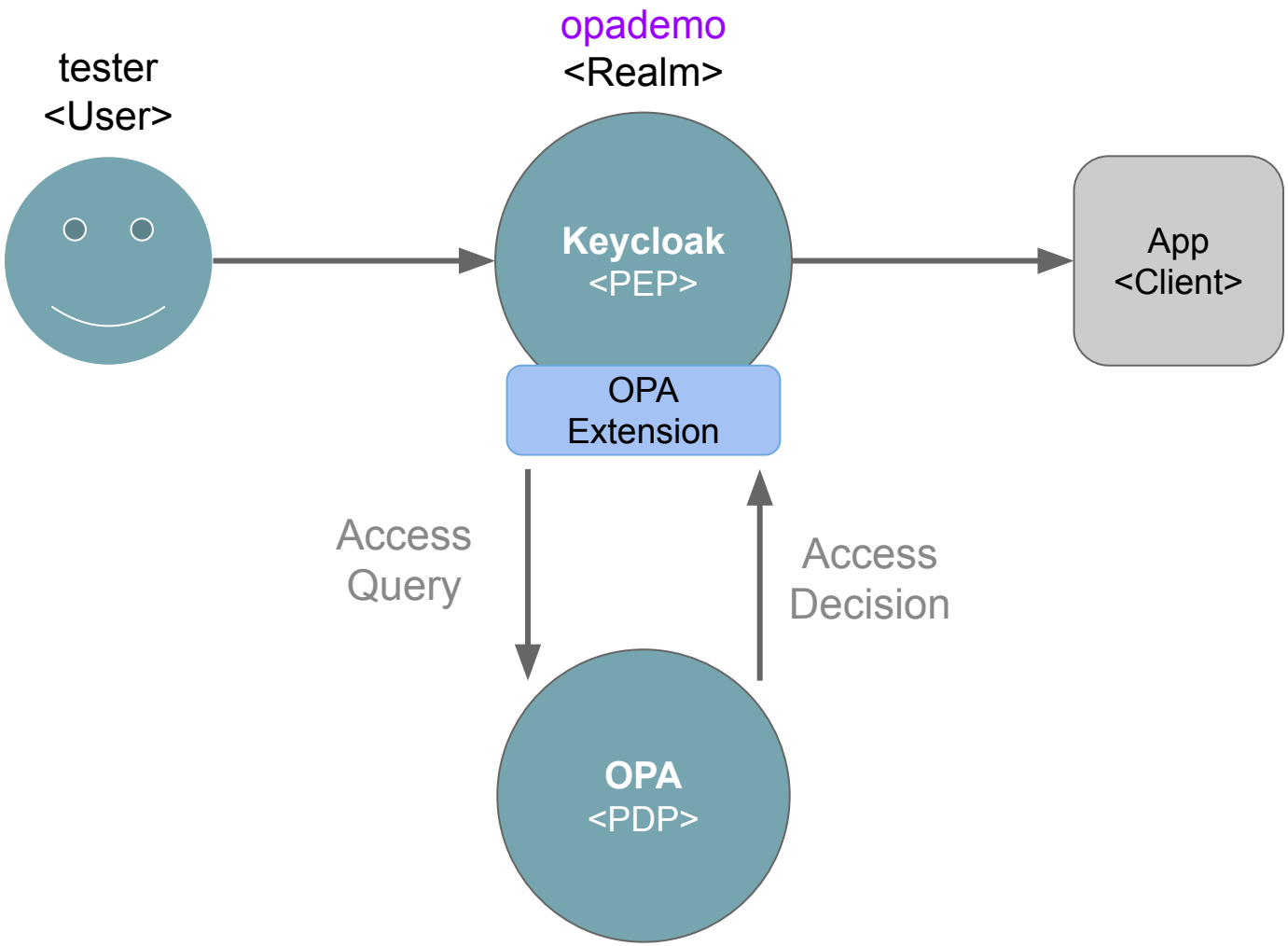
# Integration Options

- Authenticator SPI
- Required Action SPI
- Client Policy SPI
- Protocol Mapper SPI
- AuthZ Service Permission Evaluator SPI
- Extend Community Extensions, e.g. Restrict Client Auth

# How to Manage Client Access with OPA?

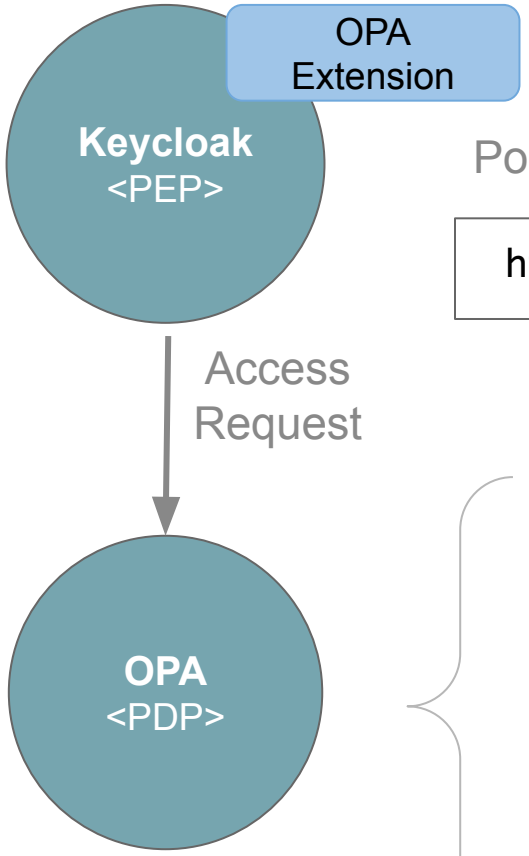
- **Define Access Policy per Realm or Client**
  - Access Rules for all Clients in a Realm
  - Client specific Access Rules
- **Enable OPA Access Checks in Keycloak**
  - Configure OPA Integration via…
    - Custom Authenticator (for selected Auth Flows)
    - Custom Required Action (for all Browser based Auth Flows)
    - Custom Client Profile / Policy (for Client Credentials and Password Grant)
  - Enable and Configure OPA Integration
- **Manage Policies via Policy as Code**
  - ***Push*** Policy ***to OPA*** *or* let *OPA* ***Pull*** **from Source**

# Access Control with Keycloak + OPA (1)



# Access Control with Keycloak + OPA (2)

opademo
<Realm>




Policy URL

Realm Policy Rule

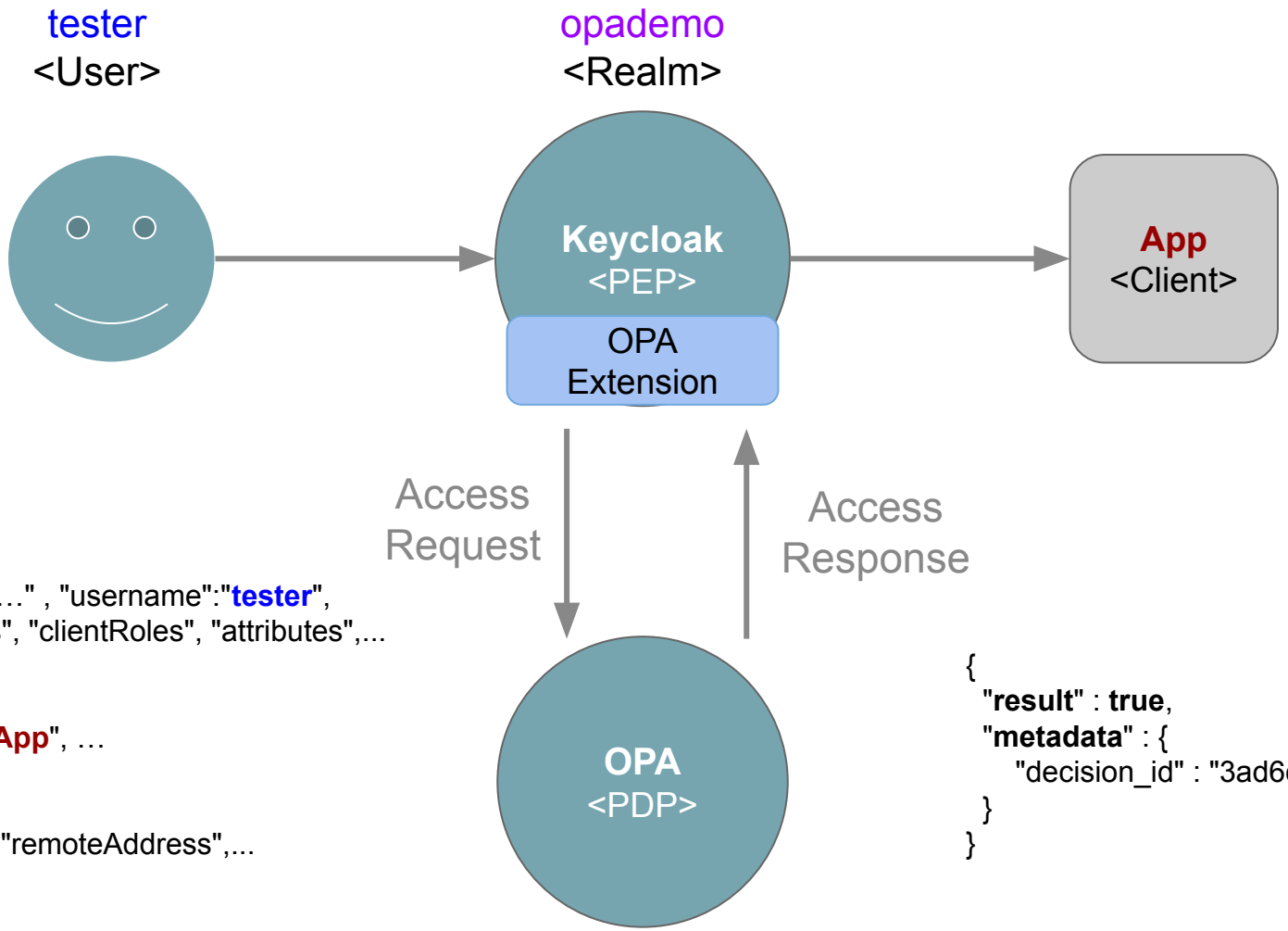
```
https://opa/v1/data/keycloak/realms/opademo/access/allow
```

Access Policy

```
package keycloak.realms.opademo.access
...
import data.keycloak.utils.kc

default allow := false
...
allow if {
   kc.isClient("app")
   kc.hasCurrentClientRole("access")
}
```

# Access Control with Keycloak + OPA (3)




```
{
  "input" : {
    "subject" : {
      "id": "4be7d…" , "username":"tester",
      "realmRoles", "clientRoles", "attributes",...
    },
    "resource" : {
      "clientId" : "App", …
    },
    "context" : {
      "protocol", "remoteAddress",...
    }
  }
}
```

```
{
  "result" : true,
  "metadata" : {
    "decision_id" : "3ad6c…"
  }
}
```

`https://opa/v1/data/keycloak/realms/opademo/access/allow`

# Demo Access Control with Keycloak + OPA

# Access Request / Response Example

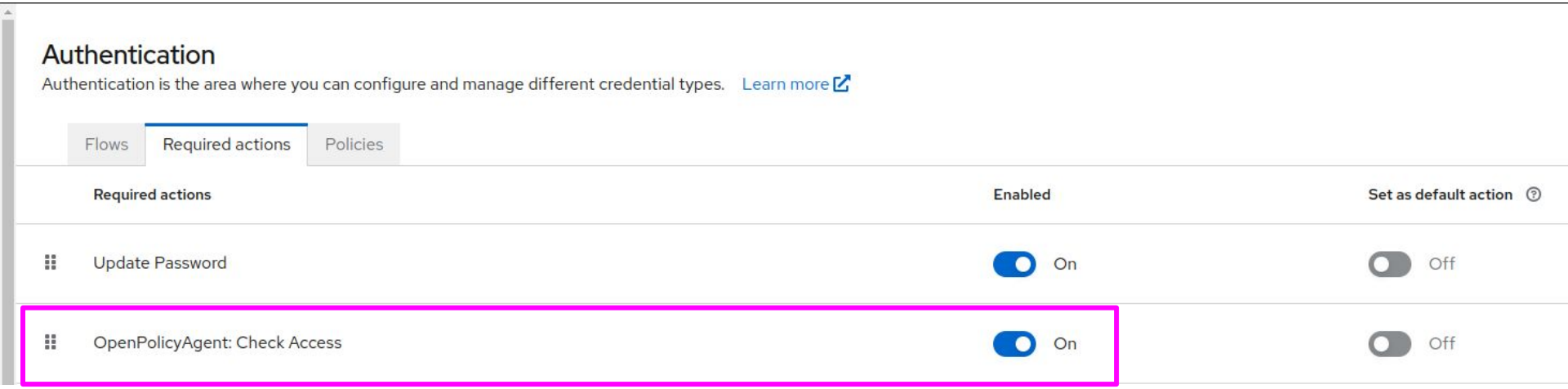
Access Request


```
{
  "input" : {
    "subject" : {
      "id" : "5a69dc8a-fce8-4a4d-a5bb-7a7603984217",
      "username" : "tester",
      "realmRoles" : [ "default-roles-opademo", "offline_access",
      "clientRoles" : [ "account:manage-account", "account:view-p
      "attributes" : {
        "emailVerified" : true,
        "email" : "test@local.test"
      }
    },
    "resource" : {
      "realm" : "opademo",
      "clientId" : "account-console"
    },
    "context" : {
      "attributes" : {
        "protocol" : "openid-connect",
        "remoteAddress" : "172.18.0.1"
      }
    }
  }
}
```


Access Response


```
{
  "result" : false,
  "metadata" : {
    "decision_id" : "7b6554d6-6c50-4c8c-b79b-dbc1eb4b6299"
  }
}
```

# Check Access Required Action



# OPA Client Profile and Policy

Realm settings > Client policies > Client profile details

opa-check-access-profile

Action

**Client profile name** *

opa-check-access-profile

The name must be unique within the realm

**Description**

Save Reload

Executors

Add executor

opa-client-access-policy-enforcer

Realm settings > Client policies > Policy details

opa-client-access-policy

Enabled

Action

**Name** *

opa-client-access-policy

**Description**

Save Reload

Conditions

Add condition

any-client

Client profiles

Add client profile

opa-check-access-profile

# OPA Client Access Policy Enforcer

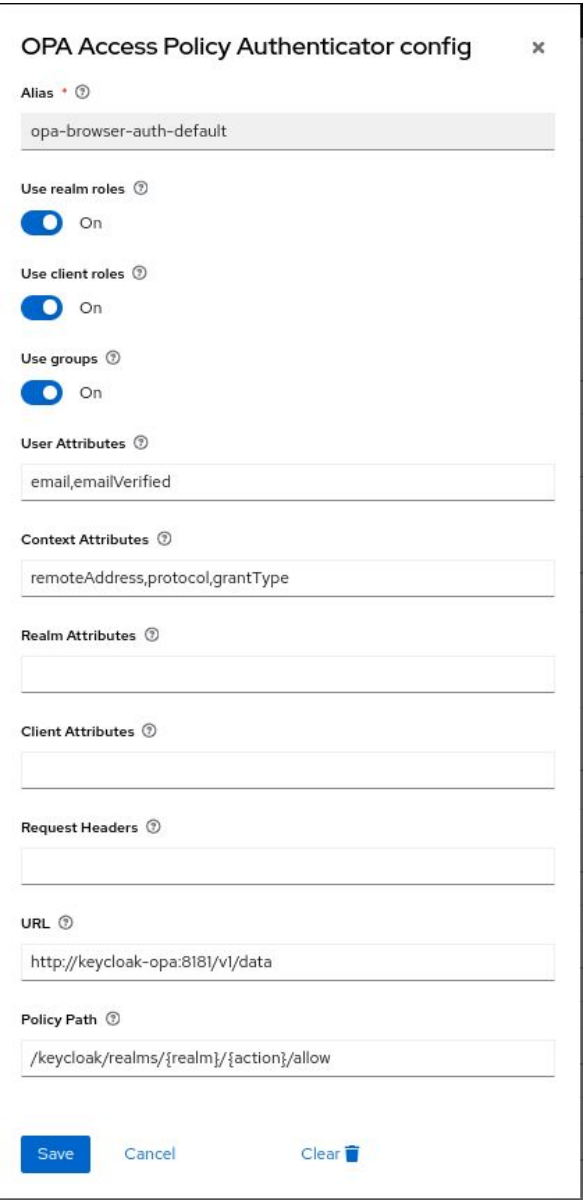
Realm settings > Client policies > Client profile details > Executor details

## opa-client-access-policy-enforcer

| | |
|---|---|
| Executor type | opa-client-access-policy-enforcer |
| Use realm roles | On |
| Use client roles | On |
| Use groups | On |
| User Attributes | email,emailVerified |
| Context Attributes | remoteAddress,protocol,grantType |
| Realm Attributes | |
| Client Attributes | |
| Request Headers | |
| URL | http://keycloak-opa:8181/v1/data |
| Policy Path | /keycloak/realms/{realm}/{action}/allow |

Save Cancel

# OPA Access Policy Authenticator



# Testing Access Policies with OPA

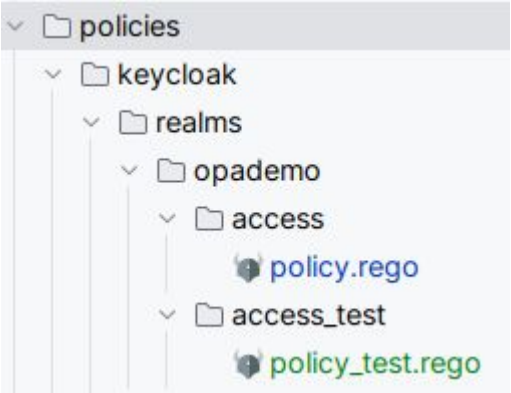
```
package keycloak.realms.opademo.access_test

# See https://www.openpolicyagent.org/docs/latest/policy-testing/
import rego.v1

import data.keycloak.realms.opademo.access

test_access_account_console if {
	access.allow with input as {
		"subject": {"username": "tester", "realmRoles": ["user"]},
		"resource": {"realm": "opademo", "clientId": "account-console"},
	}
}

test_access_app1 if {
	access.allow with input as {
		"subject": {"username": "tester", "clientRoles": ["app1:access"]},
		"resource": {"realm": "opademo", "clientId": "app1"},
	}
}
```




```
$ opa test ./policies -v
policies/keycloak/realms/opademo/access_test/policy_test.rego:
data.keycloak.realms.opademo.access_test.test_access_account_console: PASS (471.053µs)
data.keycloak.realms.opademo.access_test.test_access_app1: PASS (373.272µs)
--------------------------------------------------------------------------------
PASS: 2/2
```

# Regal Linter for Rego

Styra Documentation | Styra DAS | Styra DAS API | Enterprise OPA NEW | Regal NEW | Blog | Styra Academy | Search ctrl K

- Introduction
- Rules
  - Bugs
  - Idiomatic
  - Imports
  - Style
  - Testing
  - Custom
- Custom Rules
- Pre-Commit Hooks

Introduction

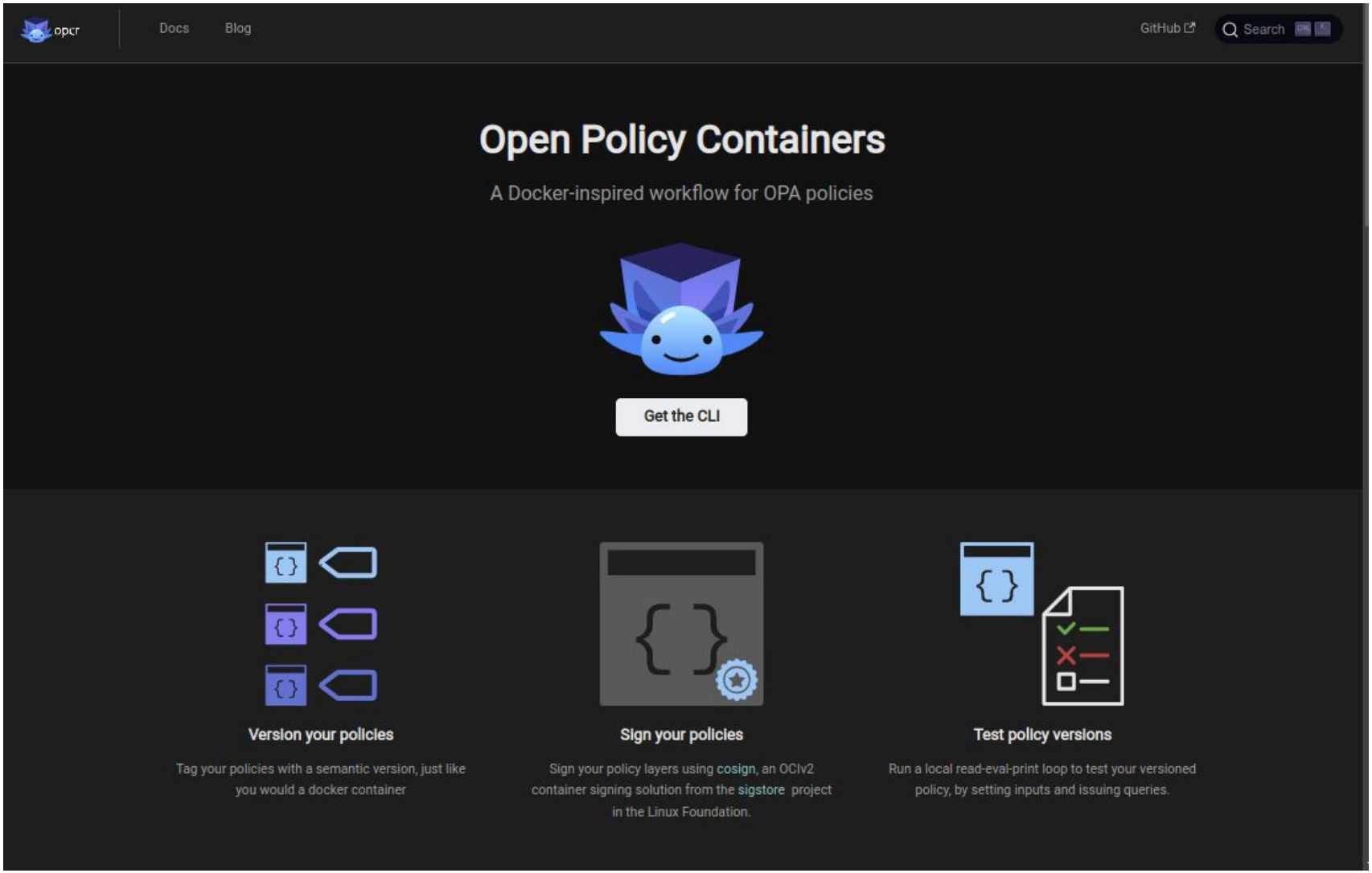
## Regal

Regal is a linter for Rego, with the goal of making your Rego magnificent!

> regal
>
> adj : of notable excellence or magnificence : splendid

- Merriam Webster

# Open Policy Containers



# Summary

- **Authorization Services in Keycloak**
  - Practical for some Use Cases but needs better DX

- **Restricting Client Access**
  - Often requested Feature
  - Possible with JavaScript or Community Extensions
  - Keycloak currently lacks a simple "*Access Policy*" Feature

- **Open Policy Agent + Keycloak**
  - Enables flexible Policy Management
  - **Decisions** can be **delegated** to OPA and **enforced** by Keycloak
  - **Helps** to **consolidate** existing **Access Logic**

## Thank you!

KEYCLOAK

# Questions?

# code & slides

[thomasdarimont/keycloak-opa-authz-demo](https://github.com/thomasdarimont/keycloak-opa-authz-demo)

<github>

# Styra Academy - Free Training

styra | ACADEMY https://academy.styra.com ALL COURSES SIGN IN

Learn how to enforce authorization policy across your cloud-native stack.

## Courses on Unified Policy

**Direct from the creators of Open Policy Agent**

Policy as Code, Infrastructure Policy
OPA for K8s Admission Control
29 Lessons | Free


Policy as Code
OPA by Example
★★★★★ (3)
33 Lessons | Free


Policy as Code, Infrastructure Policy
Terraform Validation with Styra
18 Lessons | Free

# Links

- Open Policy Agent Product
- Open Policy Agent Web Site
- Awesome OPA
- Rego Styleguide
- Gatekeeper
- OPA Gatekeeper Library

# Policy as Code

*"A **programmatic** approach to **uniformly define**, **maintain**, and **enforce authorization policies** throughout cloud-native **applications** and the **infrastructure** they run on."*

Think infrastructure as code, but for AuthZ!

Reusable policy logic for authorization, Kubernetes admission, request processing, CI/CD deployment

Human and machine readable textual description of policies that can be put on source-control.
Policies can be updated and distributed without application restart!

# Open Policy Agent Use-cases

- **Authorization** for Microservices / Applications
  - e.g. REST, GRPC, GraphQL
- **Admission Control** of Kubernetes resources
  - Can the the Deployment be applied?
- **Validation** of Configurations
  - Is this configuration allowed according to company policy?
- **Quality Gates, Stage Propagation** in CI/CD
  - Can we rollout to prod?
- **Feature Flags** in Software Delivery
  - Is this feature available to user X?

# OPA Example Session

**User Bob** *clicks* on a the link in the browser `.../finance/salary/alice`

**Decision**

**Query** POST https://opa:8181/v1/data/app/authz/allow

{ “allow”: true}

{ “input”: { "user": "alice", "method": "GET", "path": ["finance", "salary", "alice"]} }

***Policy***

*Allow users to access their own salary as well as the salary of their direct subordinates.*

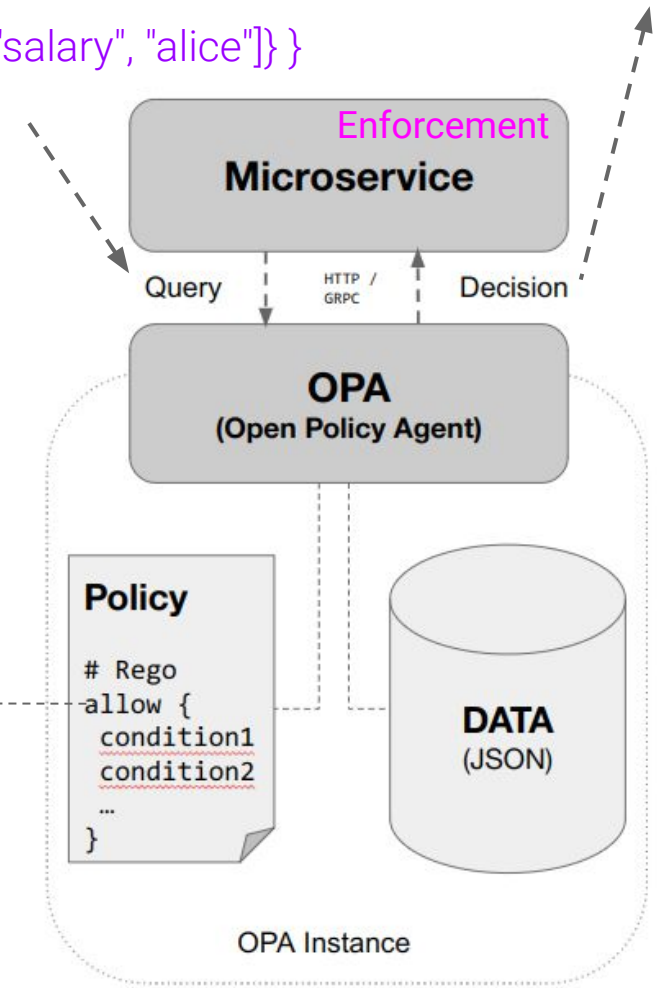
```
package app.auhtz

# bob is alice's manager, and betty is charlie's
import data.subordinates

default allow := false

# Allow users to get their own salaries
allow {
    input.method == "GET"
    input.path == ["finance", "salary", input.user]
}

# Allow managers to get their suborinates salaries
allow {
    some username
    input.method == "GET"
    input.path = ["finance", "salary", username]
    subordinates[input.user][_] == username
}
```



# Demo: RBAC Rego Policy

The Rego Playground

Examples ▾ | ☑ Strict | ☐ Coverage | Evaluate | For

```
package app.access

default allow := false

allow {
    input.required_permission = current_user_permissions[_]
}

user_permissions[user] := perms {
    role = user_roles[user]
    perms := data.role_permissions[role].add[_]
}

current_user_permissions[perm] {
    role = current_user_roles[_]
    perm = data.role_permissions[role].add[_]
}

current_user_roles := roles {
    roles = user_roles[input.user]
}

user_roles[user] := roles {
    direct_roles = data.user_roles[user]
    role_graph[direct_role]
    roles := graph.reachable(role_graph, direct_roles)
}
```

INPUT

```
{
  "user": "alice"
}
```

Input from Request

DATA

```
{
    "roles": {
        "global": {
            "board": {
                "parent": "manager"
            },
            "manager": {
```

Data from memory

OUTPUT

```
Found 1 result in 291.464 µs.
[
    "access:internal",
    "access:public"
]
```

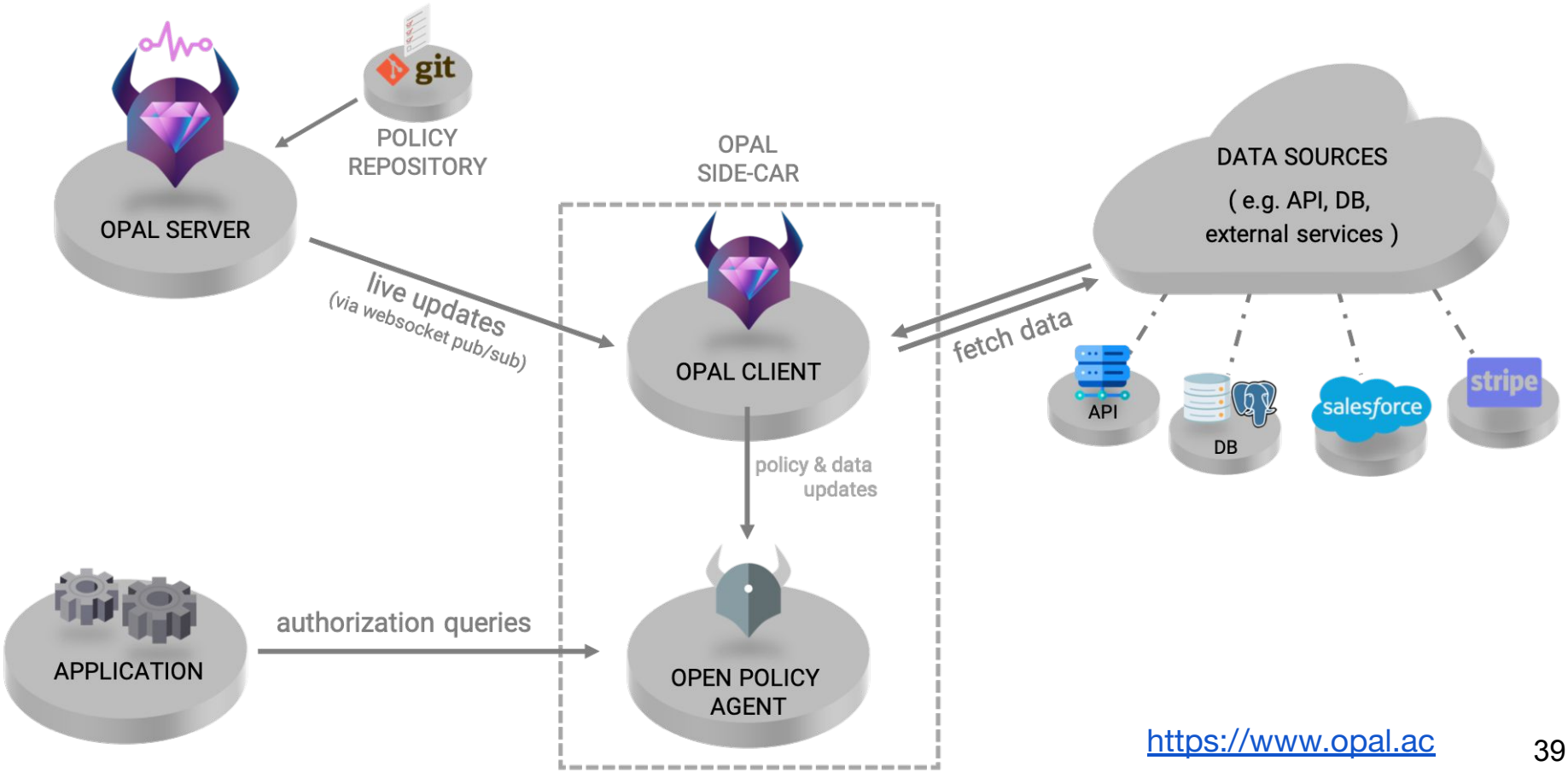
Generated results

Built by styra

OPA v0.44.0

# OPAL Open Policy Administration Layer

- OPAL is an another community project
- Provides control-plane for distributed orchestration of policies




https://www.opal.ac

# OPA as Uniform Interface

- **Combine PBAC and ReBAC Schemes**
- Use OPA as for policy based Authorization
- Use OPA to delegate to ReBAC capable System
- Application only sees OPA!