Flexible Authorization for Java Applications with Open Policy Agent

Overview and Use-cases

Thomas Darimont

About me

- Thomas Darimont
- Principal Consultant codecentric
- Open Source Enthusiast
- Spring Team Alumni
- Keycloak Contributor for over 8 years
- Official Keycloak Maintainer
- Extism Java SDK (WASM for Java)
- Java User Group Saarland Organizer

@thomasdarimont
@jugsaar
thomas.darimont@codecentric.de

“Every Non-Trivial Application Uses Some Form of ***Access Control****.”*

Access Control

$\in$

Identity & <u>Access Management</u>

IAM: Identity & Access Management

Identity Management

- Identity Proofing
- Identity Life-cycle
- Identity Self-services
- …
- **Authentication**

Access Management

- Policy Administration
- Entitlements Management
- Auditing
- …
- **Authorization**

Authentication & Authorization

- Authentication (AuthN)
  - ***who*** the user ***is***
  - → ***Identity*** (Alice, Bob, …, Guest)
- Authorization (AuthZ)
  - ***what*** the user ***is allowed*** to do
  - usually happens AFTER authentication
  - Rights, Permissions, Privileges
  - → ***Entitlements*** (payroll:access, contact:export, etc.)

Typical Authorization Use Cases

- *Restrict access based on HTTP endpoints*
  - Can $user perform HTTP GET /salaries/bob ?
  - Can $user perform HTTP POST /salaries/bob/raise ?
- *Restrict access on functional level*
  - Can $user export customer contact information?
  - Can $user with role support impersonate users?
  - Can $user see the delete account button in the UI?
- *Restrict access on resource level*
  - Can $user access the salary information of $user2?
  - Can $user cancel subscription for customer 42?
  - Can $user access the finance system?

Authorization in Applications

- Access Control Types
  - RBAC: Role based access control User has role “Manager”
  - ABAC: Attribute based access control User has plan:premium
  - PBAC: Policy based access control Allow helpdesk from 9am-5pm
  - ReBAC: Relation based access control Org users can access doc
- Access Control within Application code
  - Explicit checks for user / roles / group / permissions / attributes
  - Imperative: Method calls integrated with business code
  - Declarative: Annotations augmenting business code
  - Checks before calls (Guarded Commands, Filter, `PreAuthorize`)
  - Checks after calls (Filter Results, `PostAuthorize`)
- Access Control outside of Application code
  - Policies describe access control logic
  - Textual Policy Languages (XACML, ALFA, Rego, etc.)

Java Frameworks / Libraries for Authorization

- JAAS: Java Authentication and Authorization Service
- Java Authorization Frameworks
  - Spring Security
    - @Secured, @PreAuthorize, @PostAuthorize
  - Apache Shiro
    - subject.hasRole("admin");
    - subject.isPermitted("user:delete:jsmith")
  - OACC Framework
- Policy based authorization (XACLM)
  - WSo2 Balana
  - Sun XACML
- Externalized Authorization
  - Casbin, Cerbos, Open Policy Agent

Policy based Authorization

Policies contain ***Rules*** with ***Conditions*** to ***decide*** if a ***Subject*** (user) is **allowed** to perform an ***Action*** with a ***Resource*** in a given ***Context***.

- XACML
  - eXtensible Access Control Markup Language
  - WSo2 Balana
  - Sun XACML
- Alternatives
  - Casbin,
  - Cerbos
  - ***Open Policy Agent***

History: Policies with XACML

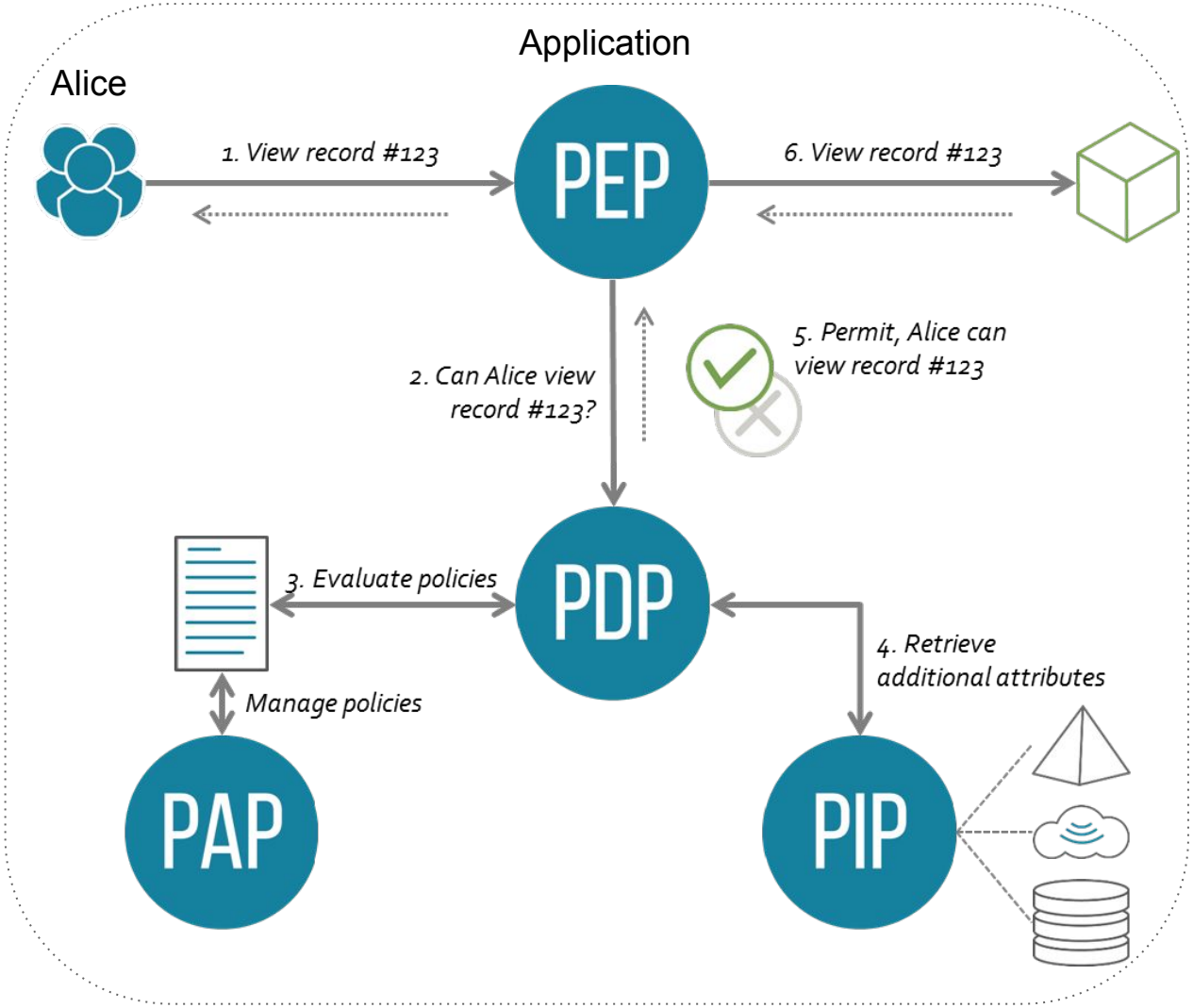
- eXtensible Access Control Markup Language
- XML-based language for access control policies
- Quite dated OASIS standard (V1 2003; V2 2005)
- Java Implementations Sun XACML and WSo2 Balana

Policies containing ***Rules*** with ***Conditions*** to ***decide*** if a **Subject** (user) is **allowed** to perform an **Action** with a **Resource**.

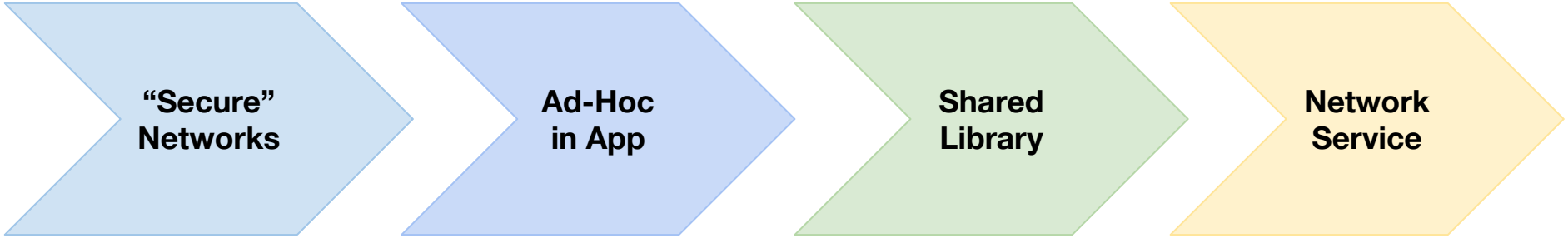
Example XACML Policy

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.



Evolution of Authorization Systems



Usage of “right” network serves as authorization

Programmatic / declarative role & permission checks in every app code

Authorization logic shared across multiple teams with same technology stack

Authorization logic shared across multiple teams with arbitrary technology stack

Authorization Variants for Microservices

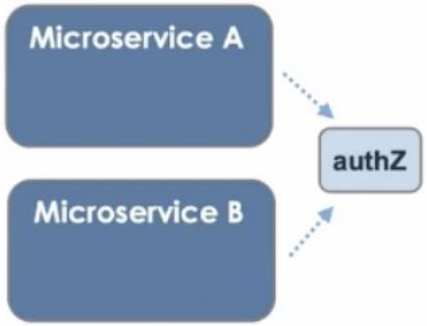
Authorization Variants for Microservices

Hardcoded

ADVANTAGES
- Available and performant

DRAWBACKS
- Risk of breakage
- More repeated work
- Inconsistent implementation and logging

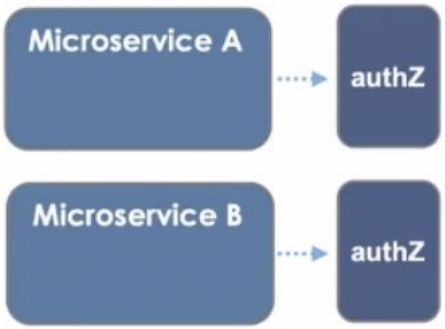
Centralized Service



ADVANTAGES
- No repeated work
- Consistent policies
- Hot-patching
- Fast security reviews

DRAWBACKS
- Availability and performance suffer
- Inconsistent enforcement

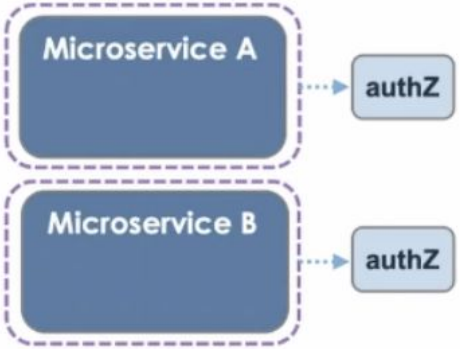
Sidecar



ADVANTAGES
- No repeated work
- Consistent policies
- Hot-patching
- Fast security reviews
- Available, performant

DRAWBACKS
- Inconsistent enforcement

Service Mesh



ADVANTAGES
- No repeated work
- Consistent policies
- Hot-patching
- Fast security reviews
- Available, performant
- Consistent logging

DRAWBACKS
- Deploy proxy

Open Policy Agent

Open Policy Agent Overview

- Open-Source **policy engine** written in Go
  - Pull / Push, Validate, Eval
- Developed by Styra Inc. and community
- Propagates "***Policy as Code***"
  - Authorization logic as source
  - version, lint, test, refactor, audit
  - **Rego**: Declarative policy language
- **Sidecar deployment**
  - **Idea: decouple** policy **decision** from **enforcement**

https://www.openpolicyagent.org

Open Policy Agent Use Cases

- Microservice **authorization**
  - e.g. REST, GRPC, GraphQL
- Kubernetes **admission control**
  - can we run the deployment?
- Configuration **validation**
- CI/CD **quality gates, stage propagation**
- Software Delivery **Feature flags**

Policy as Code

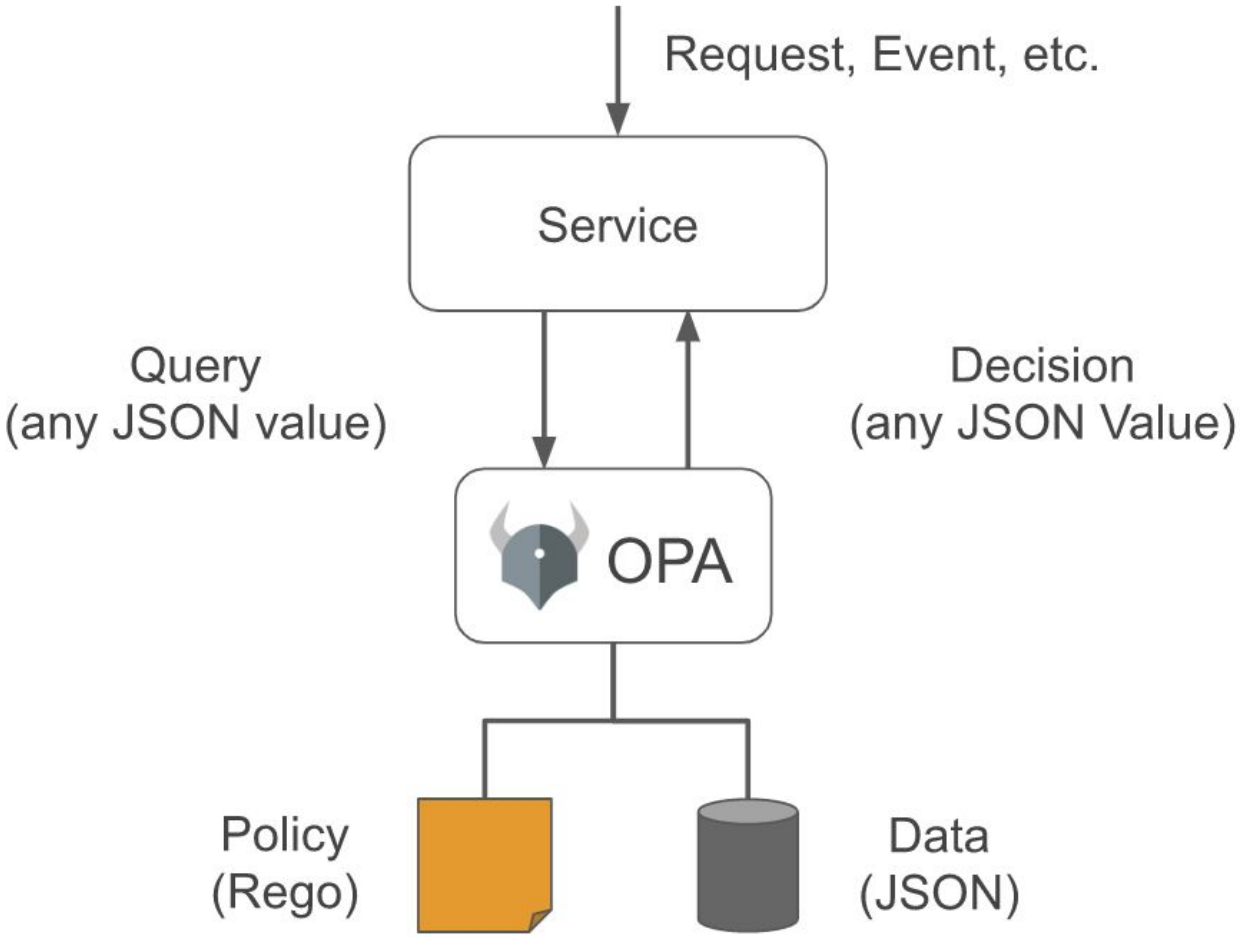
*"A **programmatic** approach to **uniformly define**, **maintain**, and **enforce authorization policies** throughout cloud-native **applications** and the **infrastructure** they run on."*

Think infrastructure as code, but for AuthZ!

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

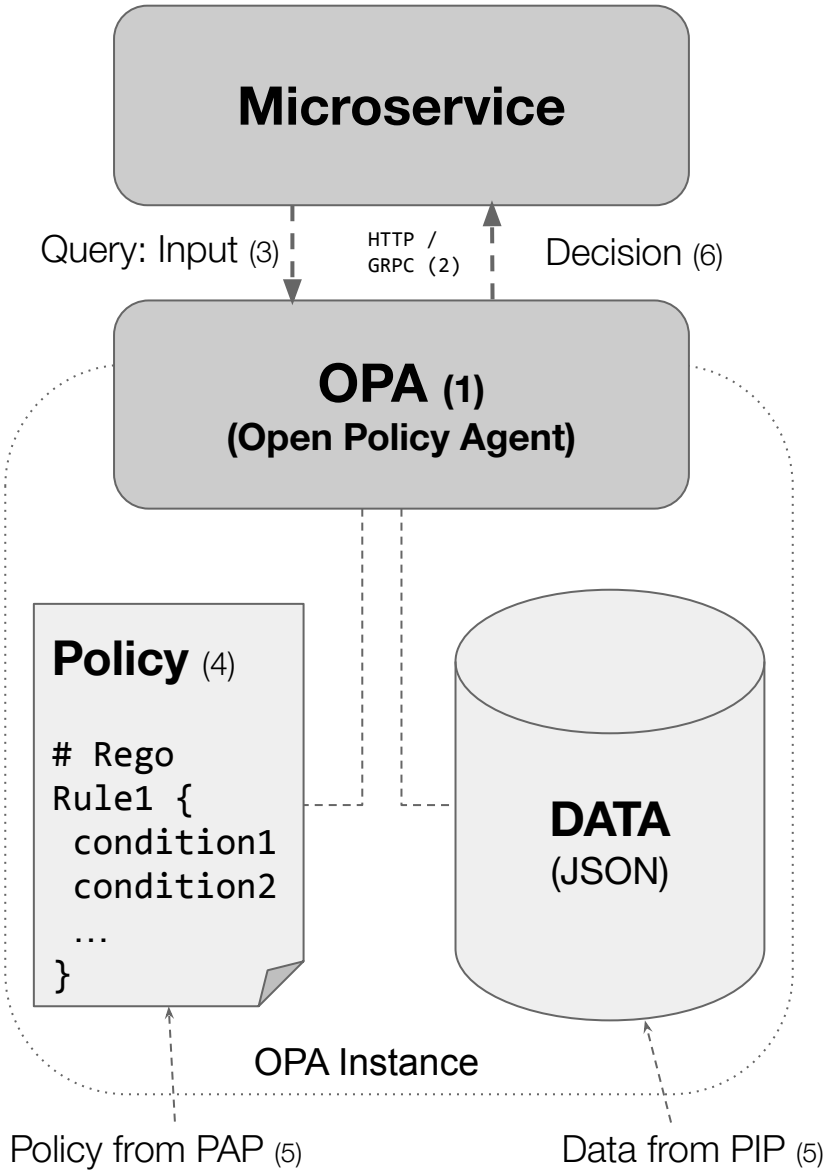
Human and machine readable textual description of policies that can be put on source-control.
Policies can be updated and distributed without application restart!

How does OPA work?



Source

OPA Usage with Microservice



1. OPA instance runs besides Microservice (colocated)
2. OPA instance provides REST / GRPC API Input & Result (Decision) JSON / ProtoBuf
3. Microservice generates authz query and delegates authorization decision to OPA
4. OPA uses policies to make decision (OPA acts as *Policy Decision Point*, ***PDP***)
5. Policies and data loaded into memory via push / pull (policy bundles)
6. Policy decision enforced by Microservice

OPA Policy Queries

- REST / GRPC API Interface
  - `POST` https://opa-service/v1/policies/path/to/policy/rule
  - Policy and Rule selected via request path

- `{ “input”: data }`
  - Context data from request / application
  - `data` can be any JSON Object

OPA Policy Query Styles

- Example: Subject, Action, Ressource
  - Subject: User / Service-Account / System
  - Action: Operation
  - Resource: Object type / Object Entity (Type+ID) / Instance
- Example: Subject, Permission
  - Subject: User / Service-Account / System
  - Permission: logical access right

OPA Example Session

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

Decision

Query POST https://127.0.0.1:8181/v1/policies/app/authz/allow

{ “allow”: true}

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

Policy

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

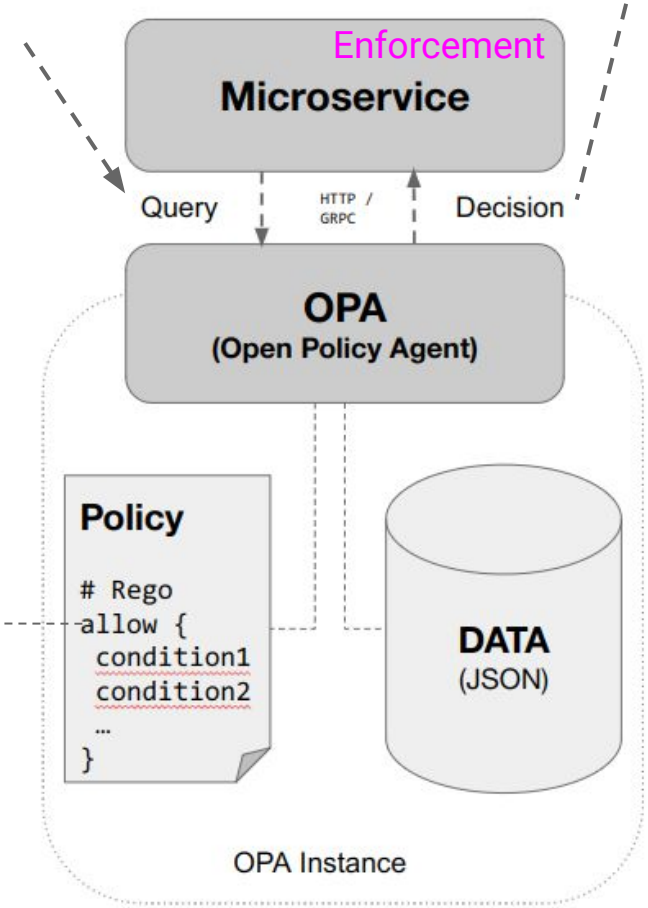
```
package app.auhtz

# 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
}
```



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, etc.)

```
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

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
}
```

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

Demo: OPA with Quarkus

```
/**
 * Provides a coarse grained interface for the greeting service.
 */
2 usages   Thomas Darimont *
@ApplicationScoped
@RequiredArgsConstructor
class GreetingFacade {

    1 usage
    private final GreetingService greetingService;

    /**
     * <pre>{@code
     * @PreAuthz(resource = "#name") --> {"action": "greet", "resource": "#name"}
     * }
     * </pre>
     */
    1 usage   Thomas Darimont *
    @PreAuthz(resource = "#name")
    public String greet(String name) {
        return greetingService.greet(name);
    }
}
```

→ Declarative Authorization for easy use and integration!

Demo: OPA with Spring Boot

```
1 usage   Thomas Darimont
@SpringBootApplication
@EnableConfigurationProperties(OpaProperties.class)
public class SpringOpaApp {

    Thomas Darimont
    public static void main(String[] args) {
        SpringApplication.run(SpringOpaApp.class, args);
    }

}
```

Demo: OPA with custom Policy Server

```
1 usage   Thomas Darimont *
@SpringBootApplication
public class SpringOpaPolicyServerApp {

    Thomas Darimont *
    public static void main(String[] args) {
        SpringApplication.run(SpringOpaPolicyServerApp.class, args);
    }

}
```

→ Generate Policies & Data dynamically based on your Application data!

Demo: OPA for Access Control in Keycloak

Authentication Flows › Browser Identity First › opa-auth

Opa-auth

ID	8c67d724-84ed-4c54-b8dd-a35433a2b4b7
Alias	opa-auth
Authz Server URL	http://acme-opa:8181/v1/data/iam/keycloak/allow
Use user attributes	ON
User Attributes	
Use realm attributes	ON
Realm Attributes	acme.greeting
Use client attributes	OFF
Client Attributes	
Use realm roles	ON
Use client roles	ON
Use groups	ON

Save Cancel

→ Access Decisions with more powerful policy language!

Demo: Validate configuration with OPA

jugsaar / jugsaar-58-opa-for-java Public

Unwatch Fork Star 0

Code Issues Pull requests Actions Projects Wiki 0 Settings

main jugsaar-58-opa-for-java / keycloak-opa-config-validation /

Thomas Darimont Add opa keycloak-config validation example b1dfa29 7 days ago History

..

policies	Add opa keycloak-config validation example	7 days ago
readme.md	Add opa keycloak-config validation example	7 days ago

readme.md

Keycloak OPA Config Validation

PoC for validating Keycloak configuration with Open Policy Agent. In this example we use OPA's "rego" policy definition language and the configtest tool to validate a client configuration.

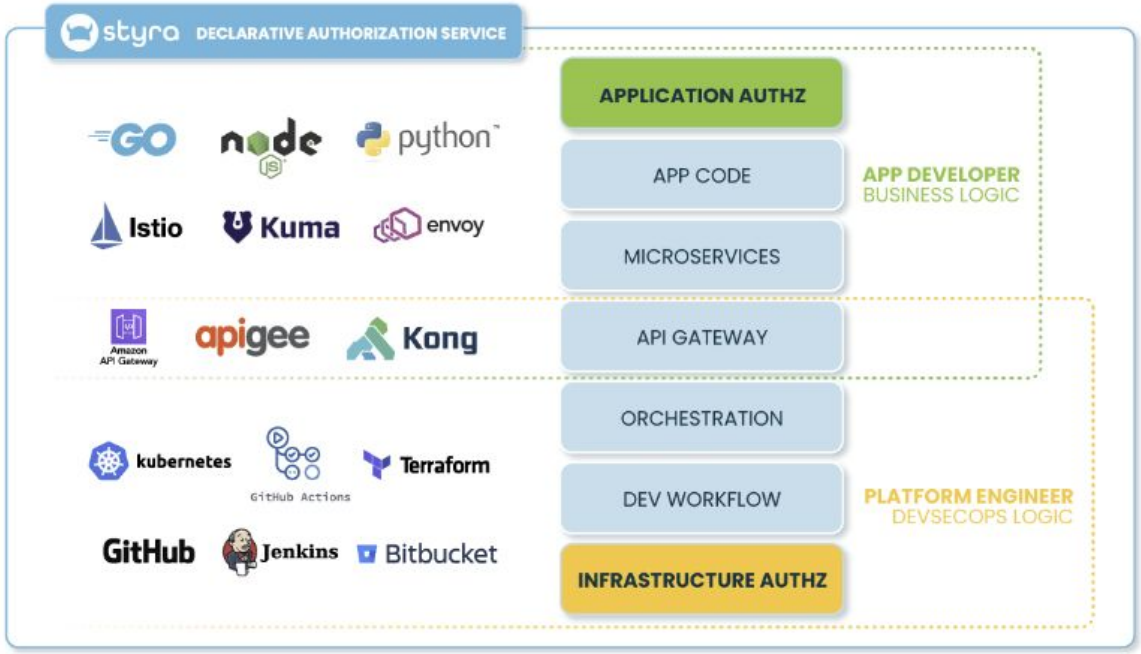
The example policy can be found in the policies folder.

Styra DAS - Declarative Authorization Service

https://www.styra.com/styra-das/

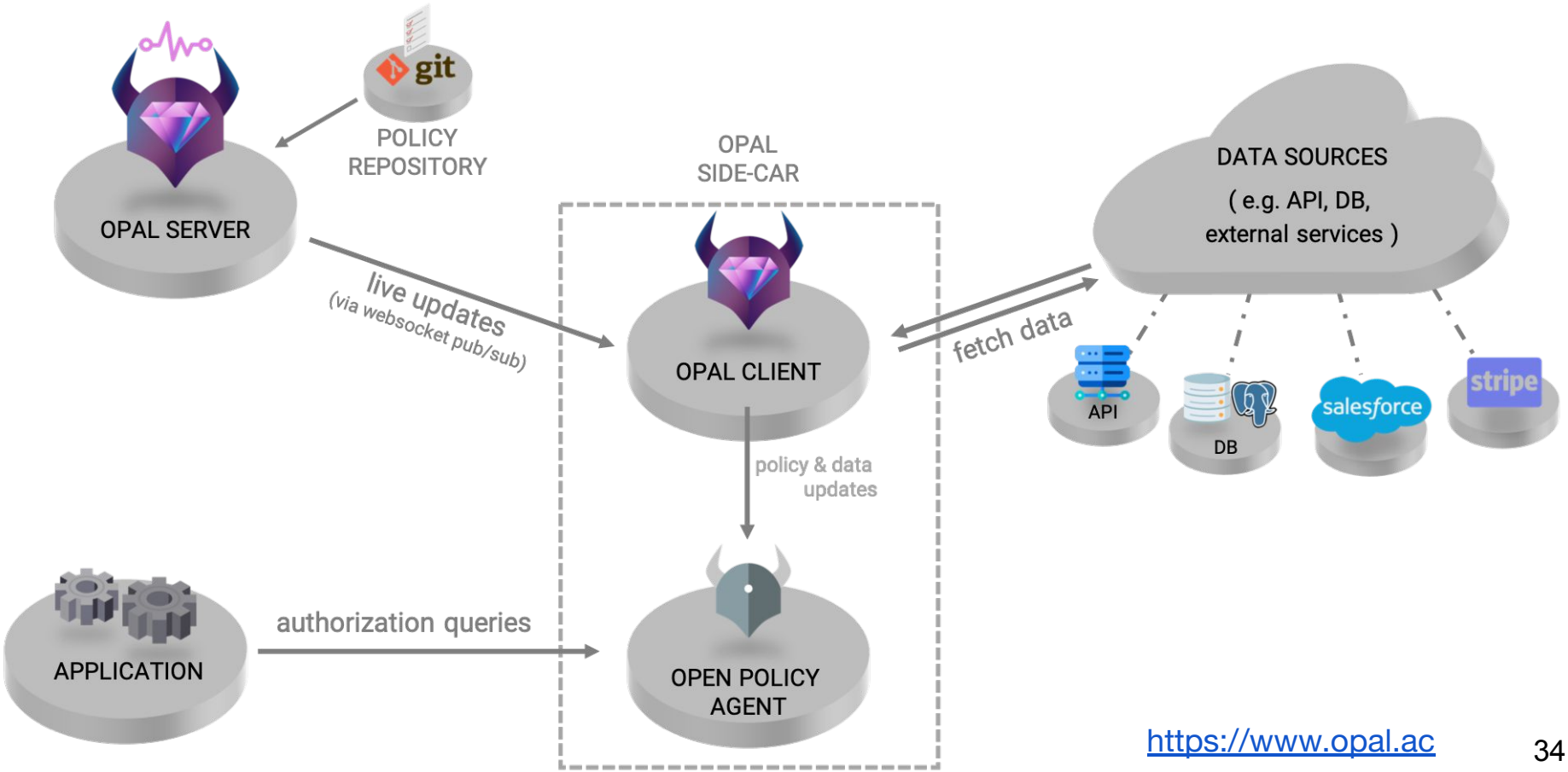
Policy-based Access Management

The Styra Declarative Authorization Service (DAS), built on top of the open-source project Open Policy Agent (OPA), provides authorization through policy management across the cloud-native ecosystem. Styra DAS allows least-privilege access through APIs, identities, systems and services for context-rich authorization.



OPAL Open Policy Administration Layer

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



https://www.opal.ac

Where does OPA fit well?

- **Logic driven** authorization
- Function / Endpoint level authorization / RBAC / ABAC
- Temporal access restrictions (9am-5pm)
- Uniform cardinality low/medium data

→ Policy Style Authorization (OPA, Ory Keto, Cerbos)

Where does OPA NOT fit well?

- **Data driven** authorization
- Ownership / relational based authorization
- Annotated direct / indirect relations
- Heterogeneous / (very) high-cardinality data

→ ReBAC Style Authorization (OpenFGA, Authzed, …)

Summary

- **Open Policy Agent**
  - **Flexible policy management** and **easy to integrate**
  - **Decisions** can be **delegated** and **enforced** in app
  - **Helps** to **consolidate** existing **access logic**
  - **Commercial products: Styra DAS, Styra Load, Aserto, Topaz**

- **Policies as Code**
  - Allows **policy sharing** and **lifecycle** management
  - **Rego** is a powerful policy language

- **OPA enables "Modern Authorization" for existing applications**

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
styra | ACADEMY
OPA for K8s Admission Control

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

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

Thank you

Questions?

About me

- Thomas Darimont
- Principal Consultant codecentric
- Open Source Enthusiast
- Spring Team Alumni
- Keycloak Contributor for over 8 years
- Official Keycloak Maintainer
- Extism Java SDK (WASM for Java)
- Java User Group Saarland Organizer

@thomasdarimont
@jugsaar
t.darimont@codecentric.de

Slides & Code

Github

thomasdarimont/javaland2023-authz-for-java-devs-with-opa

Links

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

Google Research

Philosophy | Research Areas | Publications | People | Tools & Downloads | Outreach | Careers | Blog

PUBLICATIONS ›

Zanzibar: Google's Consistent, Global Authorization System

Ruoming Pang, Ramon Caceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner, Jeffrey L. Korn, Abhishek Parmar, Christina D. Richards, Mengzhi Wang

2019 USENIX Annual Technical Conference (USENIX ATC '19), Renton, WA

Download | Google Scholar | Copy Bibtex

Abstract

Determining whether online users are authorized to access digital objects is central to preserving privacy. This paper presents the design, implementation, and deployment of Zanzibar, a global system for storing and evaluating access control lists. Zanzibar provides a uniform data model and configuration language for expressing a wide range of access control policies from hundreds of client services at Google, including Calendar, Cloud, Drive, Maps, Photos, and YouTube. Its authorization decisions respect causal ordering of user actions and thus provide external consistency amid changes to access control lists and object contents. Zanzibar scales to trillions of access control lists and millions of authorization requests per second to support services used by billions of people. It has maintained 95th-percentile latency of less than 10 milliseconds and availability of greater than 99.999% over 3 years of production

Zanzibar Implementations

- ory/keto
- Auth0 FGA
- OpenFGA
- Speedle
- AuthzDB