Policy Engine for OSDU Entitlements

A Comparison of

Open Policy Agent (OPA) and Common

Expression Language (CEL) Policy Templates

July 2020

Notices

Contents

Abstract

This whitepaper presents a comparison of two candidate software packages to implement granular authorization and data access controls to Open Subsurface Data Universe™ (OSDU) Data Platform APIs and data. It compares the two policy engines and the ways in which they can offload policy decision-making from the services. The two packages compared are Open Policy Agent (OPA) and the CEL Policy templates framework. This whitepaper documents the findings from this study.

This study is strictly scoped to the suitability of these two packages for the specific needs of the OSDU data platform and is not intended to convey any opinion about the suitability of these packages for use in other contexts.

Introduction

The OSDU Data Platform is an industry-specific system for managing Oil and Gas data. It enables organizations in the Oil and Gas industry to collect, describe and serve data to various applications and services. Historically, this data has been highly segmented, provided in different formats and was spread across different systems. Major companies have been working redundantly to build their own data platforms to integrate data silos and simplify access to it across the organization and to third-parties. The OSDU data platform provides a reference implementation for the industry. It standardizes on data schemas and a set of unified APIs for bringing data into the data platform, describing, validating, finding and retrieving data elements and their metadata. Application developers can use standard APIs to create applications that are directly connected to the operator's data sets. Its goal is to become a system of record for subsurface and wells data.

The OSDU data platform consists of multiple services that are typically deployed to the operator's cloud account/subscription. Subsurface data is prepared and loaded into the data platform and, finally, various applications are deployed on top of the platform to utilize this data via OSDU APIs and using common data types. Platform APIs are made available for applications in the form of HTTP APIs. The OSDU entitlements service handles the user management and permissions for access to the services and data.

Open Policy Agent: Overview

The Open Policy Agent (OPA) is an open source, general-purpose policy engine. It decouples policy decision-making from policy enforcement. When a web-based application needs to make a policy decision, attributes can be passed to OPA as structured data (e.g. JSON). OPA makes policy decisions by evaluating the input against policies and context data. For OPA, policies are written using a high-level declarative language *Rego*. OPA provides several ways to integrate with microservices, Kubernetes, CI/CD pipelines, API gateways, and more.

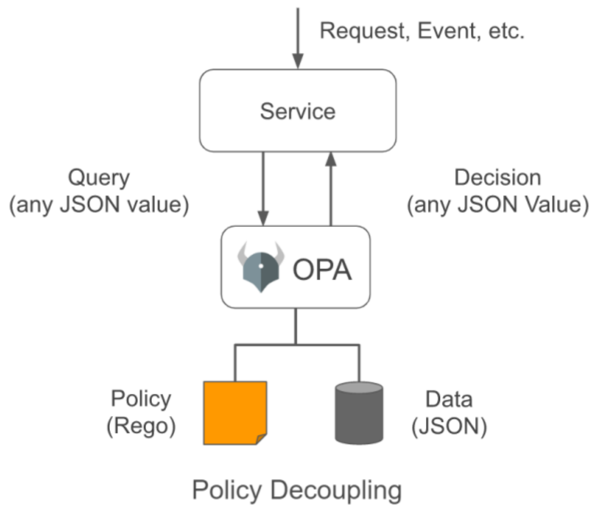

Figure 1- OPA Policy Decoupling[1]

CEL Policy Templates: Overview

[Common expression language](#) (CEL) policy templates is an open source framework written in Go. CEL consists of four key components: templates, instances, evaluators and environments.

The *templates* define the shape of the policy. Templates are written in YAML and use Open API Schema v3 terminology. *Instances* can be used to provide context data to the policy for decision making. The instance content is validated against the template definition. The expressions within the *evaluator* are written in CEL and are checked for syntactic correctness, cycles, and type-agreement. The *CEL environment* specifies an engine's configuration, variables and functions that are available to the engine to evaluate.

For more information on cel-policy-templates, see https://github.com/google/cel-policy-templates-go.

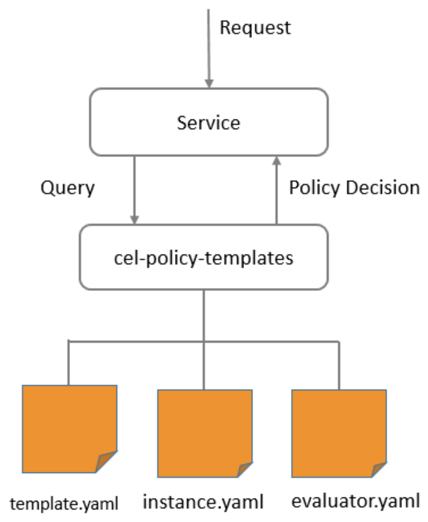

Figure 2 - CEL-Policy-templates

Comparison

Environment

To enable comparison, 10 OSDU use cases (see below) were identified by the OSDU forum members. Rules implementing all 10 of these use cases were written in both OPA and CEL.

A serverless implementation pattern was established. Each OPA/CEL policy use case was written as AWS Lambda function running in a Go 1.X runtime. The policy and a minimal set of context data was packaged along with the Lambda function and deployed with the default AWS Lambda configuration using the AWS Cloud Development Kit. The Lambda functions were exposed as REST APIs with Amazon API Gateway.

The ten OSDU data platform use cases that were evaluated can be found at: https://community.opengroup.org/osdu/documentation/-/wikis/OSDU-(C)/Architecture-Exploration-Topics/OSDU-R3-Architecture/Entitlements-User-Stories.The formal descriptions are available on the web site. The titles of the use cases are:

- DataPurchase
- DataRoomFarmIn
- DataSubscription

- EconomicSanctions
- ExportRestrictions
- Joint Operating Agreement
- MultiClientData
- Production Sharing Contract
- StateSecret

Comparison Summary

Table 1- High-level evaluation summary

	OPA	**CEL-policy-templates-go**
Open Source	Yes	Yes
Policy Language	Rego	CEL
Policy Definition Format	Rego	YAML
Policy Context Data Format	JSON	YAML
OSDU Use Case Compatibility	Yes	Yes
Professional Support	Yes	No
GitHub Project Usage	3300 stars, 392 forks	10 stars, 5 forks

	OPA	**CEL-policy-templates-go**
Logical Operators	No Logical OR for rules. This requires writing multiple rules with the same name.	Logical operators such as && and || supported making rules shorter
Development Tools	The Rego playground Interactive shell	
Testing Tools	Unit testing framework	-
Integration Support	As a host-level-daemon As a Go library	As a Go library
Development Support	Offline policy testing using ‘opa eval’.	
JWT support	Has specialized JWT support	
APIs	Management APIs Status Service APIs Decision LogService APIs	
Extensibility	Yes. Can be extended by adding custom Go functions and building from source	Yes. Can be extended by adding new Go functions and building from source

	OPA	**CEL-policy-templates-go**
Product Documentation	Extensive documentation.	Limited Documentation through github project. Visit cel-spec, cel-go, cel-policy-templates-go.

Performance Testing

A third-party, cloud-based, continuous testing platform named BlazeMeter was used for performance tests. The following table illustrates the results of high-level performance test cases and response times. The load testing parameters included total users, duration, ramp-up time, ramp up steps (RPS). The results are shown below.

Table 1: Performance test summary

Test #	**Policy Engine**	**Total Users**	**Duration(min)**	**Avg. Response Time**
1	OPA	10	10	25.58 ms
2	CEL	10	10	25 ms
3	OPA	25	20	26.87 ms
4	CEL	25	20	25.75 ms

External Data and Policy Updates

Over a period of time, the policies to be enforced and the data in external data sources such as databases, will change. The possible set of inputs that can be evaluated and the amount of context data, which is the data required to make policy decisions, that can be included in a JWT token both have size limits. There will always be policies that depend on external context data to make decisions.

For example, in case of the *EconomicSanctions* use case, a country may be added or removed from the restricted destinations list. In the case of *DataRoom* use case, access to resources is allowed after the agreement expiration date, but only to users whose job role is *Data.DataSteward* or *Data.DelegatedSteward*. The policy rule itself could change allowing users with *Data.DataAdmin* role to access the resources as well. It is also possible that a new user is assigned a *Data.DataSteward* role or another user stripped off of the role. In these cases, the updated policy and the context data will have to be made available to the policy engine.

The following sections discuss the methods available to inject external data and policy updates to OPA and CEL-Policy-Templates that are relevant to OSDU entitlements.

OPA: Bundle API

OPA's bundle feature allows injecting policies and external data packaged as an archive file. A bundle server needs to be implemented to aggregate context data and policies and package them as a bundle. OPA's bundle feature will then periodically download this bundle from the bundle server. Since context data and policies are packaged together, every time OPA gets updated policies, it gets updated context data too. This strategy can be used in scenarios where the external data changes infrequently. Since OPA stores it in memory all at once, the bundle has size limitations. In case of large datasets, context data should be pruned to only what is needed for the policy to decide. Delta based bundle protocol is a feature currently under design at OPA. With the bundle feature, since data and policies arrive together, policy-data consistency is guaranteed.

The strategy of how frequently the bundle should be updated and be downloaded by OPA depends on the use case and the tolerable lag time. The total lag for the updated policy and data becoming available to OPA will be the sum of lag for the bundle server to get updated data and policy packaged as a bundle and the lag for OPA to download the bundle from the bundle server into memory. These elements are shown in Figure 3 below, Total Lag = Lag (B) + Lag (C).

For more information on OPA Bundle API, see https://www.openpolicyagent.org/docs/latest/external-data

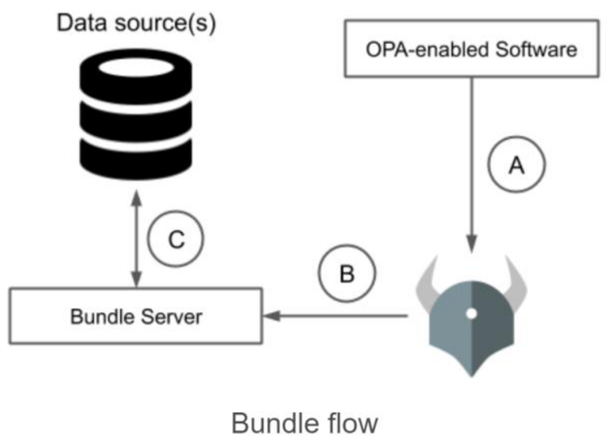

Figure 3: OPA Bundle API flow

A long lag will perform better, allowing caches to work and requiring less overhead to parse new bundles and prepare them for use. The risk is that decisions are made with stale policy data and data is exposed to unauthorized users as a result. A short lag reduces the risk of incorrect decisions because of stale rules, but it increases the overhead by introducing frequent updates.

OPA: Push Context Data

To achieve more frequent updates to context data, OPA offers a push API to load arbitrary JSON data into OPA. A data replicator process needs to be written to extract data out of external data sources (source of truth) such as databases, and then to push that information in OPA through its API.

As shown in Figure 4, although steps B and C can be decoupled, depending on the sensitivity of data, they can be combined to make every update to data available to OPA. The decoupling lets you optimize for latency and network traffic. In this case, Total lag = Lag (B) + Lag (C).

Unlike Bundle API, this approach lets you make updates to data alone. This strategy has size limitations too since the data is held in memory all at once by OPA. Instead of downloading an entire bundle, only the changes in data (deltas) will be pushed to OPA.

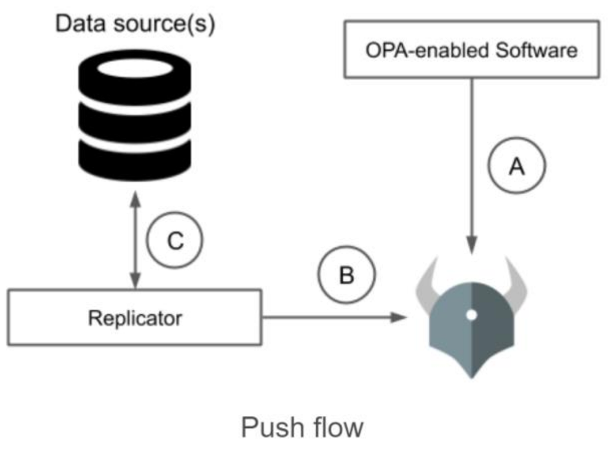

Figure 4: OPA Push Flow

OPA: Pull Data

For the scenarios where a lag in data update is not acceptable by the system or there are too many frequent updates to the policy or data, OPA has an experimental capability to reach out to the source of truth. With this approach, there are no limitations for size since data is stored outside of OPA. The data will always be up to date. The downside to this approach is, if OPA is unable to reach the external data source or if the connection is slow, OPA may not be able to return a decision. In this case, the OPA-enabled service will have to handle the case. This architecture is shown below in Figure 5.

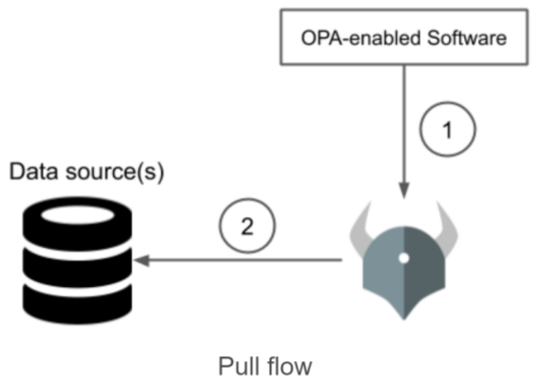

Figure 5: OPA Pull Data flow

CEL-Policy-templates: Push Data

In the CEL-policy-templates framework, policy context data is stored in an `instance.yaml` file. For more information, see CEL Policy Templates: Overview. There are no management APIs documented to push updates to templates, instances or evaluators. Custom functionality will need to be written to add this capability to the framework. If a new policy rule is created that requires a new set of inputs to evaluate, these variables need to be supplied in the CEL Environments as well.

CEL-Policy-templates: Pull Data

CEL allows *Function Overloads* (external functions written in Go) to be called from within the evaluator. This function can be used to query external data sources. Latency will depend on the network and the external data source performance.

Conclusion

Considering the different OSDU use cases, external, policy-relevant data may come in multiple forms. For example, for some API authorizations the user's identity and job role can come in through a JWT token and may be sufficient to decide. In cases where the JWT token doesn't or can't contain all the information needed, external context data may be required, for example, lists of operators allowed, list of restricted countries etc. from other sources. In such cases, data needs to be available at lowest latency. To reduce latency for a policy decision, the policy engine and required external context data could be cached with the service as a host-level daemon, side-car or a library.

When external data is updated, a 'Push Context Data' approach can be applied to inject only the relevant context data required by that service. This will keep the policy engine light and modular.

A context data aggregator needs to be built to abstract only the relevant data needed by the policy and pushed to the policy engine. This could be a periodic update or an atomic update. For example, when a user's role is updated to provide additional access to data or stripping of a role to remove access, this role update should not be complete unless this update is pushed to policy engine's context data.

The performance testing results have revealed no significant differences in the response time of OPA and CEL-Policy-templates considering the current implementation of OPA and CEL policies for OSDU use cases and assumptions regarding context data.

Contributors

Contributors to this document include:

- Rucha Deshpande, Solutions Developer, Amazon Web Services
- Srihari Prabaharan, Cloud Application Architect, Amazon Web Services

Document Revisions

Date	Description
July 2020	First publication

Notes

1 https://www.openpolicyagent.org/docs/latest