



Unit testing your Kubernetes configuration Using Open Policy Agent

Gareth Rushgrove | Director, Product Management | Snyk

May, 2019



Agenda

- A quick introduction to **Open Policy Agent**
- Shift-left testing
- Introducing **conftest**
- **Rego** as a language
- Portability between different Kubernetes solutions
- Not just Kubernetes

Open Policy Agent

A policy enforcement engine for configuration

What is Open Policy Agent?



Open Policy Agent

[Documentation](#) [Tutorials](#) [Playground](#)



Policy-based control for cloud native environments

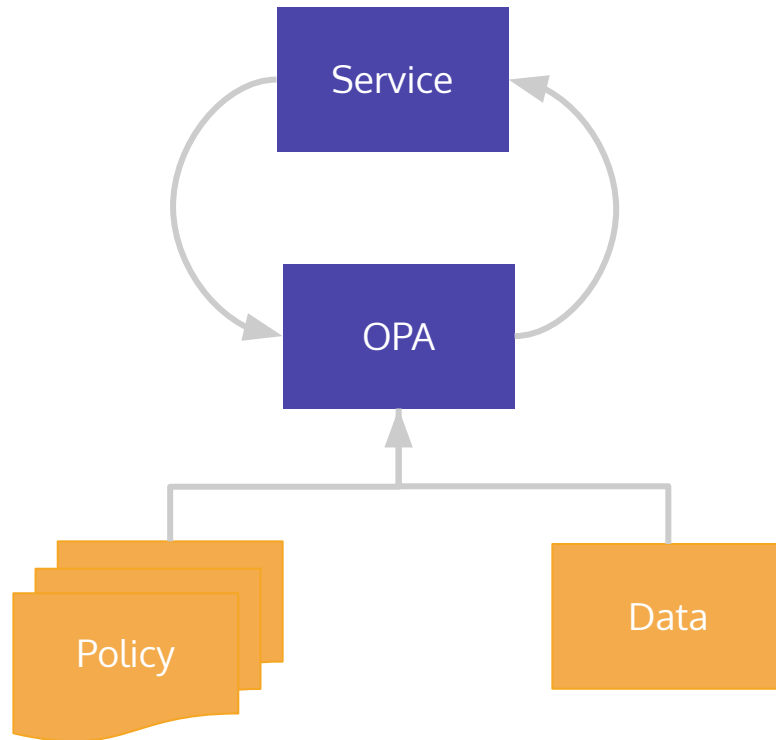
Empower your administrators with flexible, fine-grained control across your entire stack.

[Get started](#)

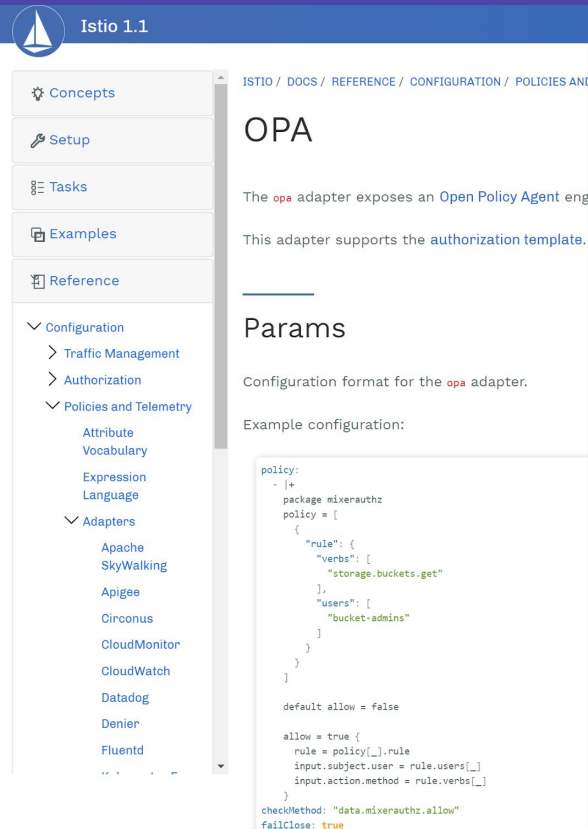
[Learn more](#)



How Open Policy Agent works



A growing ecosystem



Istio 1.1

- Concepts
- Setup
- Tasks
- Examples
- Reference
- Configuration
 - Traffic Management
 - Authorization
 - Policies and Telemetry
 - Attribute Vocabulary
 - Expression Language
 - Adapters
 - Apache
 - SkyWalking
 - Apigee
 - Cirrus
 - CloudMonitor
 - CloudWatch
 - Datadog
 - Denier
 - Fluentd

ISTIO / DOCS / REFERENCE / CONFIGURATION / POLICIES AND AUTHORIZATION

OPA

The `opa` adapter exposes an `Open Policy Agent` engine.

This adapter supports the `authorization template`.

Params

Configuration format for the `opa` adapter.

Example configuration:

```
policy:
- |+
  package mixerauthz
  policy = {
  {
    "rule": {
      "verbs": [
        "storage.buckets.get"
      ],
      "users": [
        "bucket-admins"
      ]
    }
  }
  }
  default allow = false

  allow = true {
    rule = policy[_].rule
    input.subject.user = rule.users[_]
    input.action.method = rule.verbs[_]
  }
  checkMethod: "data.mixerauthz.allow"
  failClose: true
```



Ceph Documentation » Ceph Object Gateway »

ceph

TABLE OF CONTENTS

- Intro to Ceph
- Installation (ceph-deploy)
- Installation (Manual)
- Installation (Kubernetes + Helm)
- Ceph Storage Cluster
- Ceph Filesystem
- Ceph Block Device
- Ceph Object Gateway
 - Manual Install w/Civetweb
 - HTTP Frontends
 - Pool Placement and Storage Classes
 - Multisite Configuration
 - Configuring Pools
 - Config Reference
 - Admin Guide
 - S3 API
 - Swift API
 - Admin Ops API
 - Python binding
 - Export over NFS
 - OpenStack Keystone Integration
 - OpenStack Barbican Integration
 - Open Policy Agent Integration
 - Configure OPA
 - Configure the Ceph Object Gateway
 - How does the RGW-OPA integration work
 - Multi-tenancy
 - Compression
 - LDAP Authentication
 - Server-Side Encryption
 - Bucket Policy
 - Dynamic bucket index resharding
 - Multi factor authentication
 - Sync Modules
 - Data Layout in RADOS
 - STS Lite
 - Role
 - Troubleshooting
 - Manpage radosgw
 - Manpage radosgw-admin

Notice: This document is for a development version of Ceph.

OPEN POLICY AGENT INTEGRATION

Open Policy Agent (OPA) is a lightweight general-purpose policy engine that can be integrated as a sidecar, host-level daemon, or library.

Services can offload policy decisions to OPA by executing queries against OPA's RESTful APIs.

CONFIGURE OPA

To configure OPA, load custom policies into OPA that control the access to the Ceph Object Gateway. Policies can also be loaded into OPA to make decisions.

Policies and data can be loaded into OPA in the following ways:

- OPA's RESTful APIs
- OPA's `bundle` feature that downloads policies and data from a remote location
- Filesystem

CONFIGURE THE CEPH OBJECT GATEWAY

The following configuration options are available for OPA integration:

```
rgw use opa authz = {use opa server to authorize requests}
rgw opa url = {opa server url:opa server port}
rgw opa token = {opa bearer token}
rgw opa verify ssl = {verify opa server ssl}
```

HOW DOES THE RGW-OPA INTEGRATION WORK

After a user is authenticated, OPA can be used to check if the user is allowed to perform the requested action. OPA responds with an allow or deny decision which is sent back to the Ceph Object Gateway.



styra

Declarative & Secure

Compliance guardrails from tribal knowledge

Req



styra

testerr-1lead007.styra.com
4 systems

WORKSPACE

testerr-1lead007.styra.com

SYSTEMS

- Analytics Cluster
 - Admission Control
 - Rules
 - Rules
 - Tests
- Development Cluster
- Production Cluster
- Test Cluster

Decisions

1D 1W 1M 3M

1.6K
1.2K
800
400
0



Vincent Janelle

@randomfrequency

Replying to [@garethr](#)

It's my new favourite hammer.

7:15 PM - 5 May 2019

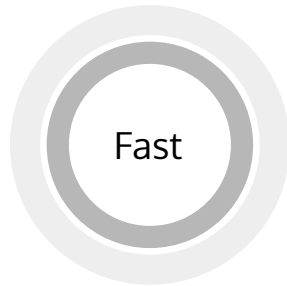
Shift left

Faster feedback

Shift-left testing is an approach to software testing and system testing in which testing is performed earlier in the lifecycle (i.e., moved left on the project timeline).

Wikipedia

Development cycle

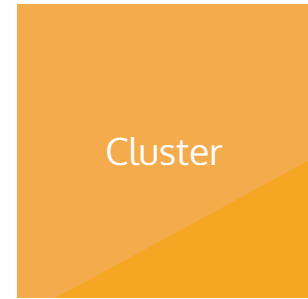
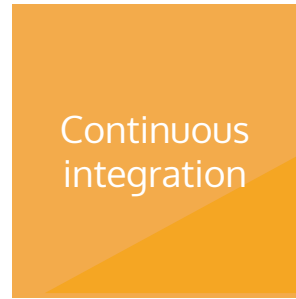


Development cycle



Open Policy Agent is normally used here

Development cycle



What if we could use Open
Policy Agent here **as well?**

Introducing **conftest**

Test your configuration locally, and in CI

Introducing conftest

instrumenta / **conftest**

Unwatch 2 Star 10 Fork 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Write tests against structured configuration data using the Rego query language













Edit

kubernetes testing rego openpolicyagent instrumenta Manage topics

39 commits 1 branch 7 releases 1 contributor View license

Branch: master New pull request

Create new file Upload files Find File Clone or download

 garethr Remove vendored pkg from oras ...	Latest commit 66275f8 7 hours ago
 .github	added automatic releases using goreleaser a month ago
 examples	Added Docker Compose example a day ago
 .gitignore	added automatic releases using goreleaser a month ago
 .goreleaser.yml	added scoop package publishing 18 days ago
 Dockerfile	convert to using Go modules a month ago
 LICENSE	added an explicit license a month ago
 README.md	Support configuration of the namespace to find rules 3 days ago
 acceptance.bats	Added acceptance tests demonstraing configuration file usage 3 days ago
 conftest.go	Remove vendored pkg from oras 7 hours ago
 go.mod	Remove vendored pkg from oras 7 hours ago
 go.sum	Remove vendored pkg from oras 7 hours ago

Write your policies

```
package main
```

```
deny[msg] {  
  input.kind = "Deployment"  
  not input.spec.template.spec.securityContext.runAsNonRoot = true  
  msg = "Containers must not run as root"  
}
```

```
deny[msg] {  
  input.kind = "Deployment"  
  not input.spec.selector.matchLabels.app  
  msg = "Containers must provide app label for pod selectors"  
}
```


Explaining what we just wrote

```
deny[msg] {  
  input.kind = "Deployment"  
  not input.spec.template.spec.securityContext.runAsNonRoot = true  
  msg = "Containers must not run as root"  
}
```

We should **deny** any input for which

Deployment is the value for *kind*

and

When **runAsNonRoot** is set to **false**

Run tests locally with conftest

```
$ conftest test deployment.yaml
```

```
deployment.yaml
```

```
Containers must not run as root
```

```
Deployments are not allowed
```

```
$ echo $status
```

```
1
```

Demo

Rego as a language

The usual pros and cons of a DSL

Good documentation



Open Policy Agent

v0.10.7 latest ▾

DOCUMENTATION

- Introduction
- How Does OPA Work?
- [How Do I Write Policies?](#)
- What is Rego?
- Why use Rego?
- The Basics
- Scalar Values
- Strings
- Composite Values
- Variables
- References
- Comprehensions
- Rules
- Negation
- Modules
- With Keyword

How Do I Write Policies?

OPA is purpose built for reasoning about information represented in structured documents. The data that your service and its users publish can be inspected and transformed using OPA's native query language Rego.

What is Rego?

Rego was inspired by [Datalog](#), which is a well understood, decades old query language. Rego extends Datalog to support structured document models such as JSON.

Rego queries are assertions on data stored in OPA. These queries can be used to define policies that enumerate instances of data that violate the expected state of the system.

Why use Rego?

Use Rego for defining policy that is easy to read and write.

Rego focuses on providing powerful support for referencing nested documents and ensuring that queries are correct and unambiguous.

Rego is declarative so policy authors can focus on what queries should return rather than how queries should be executed. These queries are simpler and more concise than the equivalent in an imperative language.

The Rego playground

```
1 package main
2 |
3 version {
4   to_number(input.version)
5 }
6
7 deny[msg] {
8   endswith(input.services[_].image, ":latest")
9   msg = "No images tagged latest"
10 }
11
12 deny[msg] {
13   version < 3.5
14   msg = "Must be using at least version 3.5 of the Compose file format"
15 }
```

Input

```
1 {
2   "version": "3.4",
3   "services": {
4     "web": {
5       "build": ".",
6       "ports": [
7         "5000:5000"
8       ]
9     },
10    "redis": {
11      "image": "redis:latest"
```

Output

```
1 # Evaluated package in 69.69 µs.
2 {
3   "result": {
4     "deny": [
5       "No images tagged latest",
6       "Must be using at least version 3.5 of the Compose file format"
7     ],
8     "version": true
9   }
10 }
```

Built-in testing tools

```
package main

test_deployment_without_security_context {
  deny["Containers must not run as root"] with input as {"kind": "Deployment"}
}

test_deployment_with_security_context {
  no_violations with input as {"kind": "Deployment", "spec": {
    "selector": { "matchLabels": { "app": "something", "release": "something" }},
    "template": { "spec": { "securityContext": { "runAsNonRoot": true }}}}
}

test_services_not_denied {
  no_violations with input as {"kind": "Service"}
}

test_services_issue_warning {
  warn["Services are not allowed"] with input as {"kind": "Service"}
}
```

Open Policy Agent test runner

```
$ opa test --verbose .
```

```
data.main.test_deployment_without_security_context: PASS (1.029μs)
```

```
data.main.test_deployment_with_security_context: PASS (1.058μs)
```

```
data.main.test_services_not_denied: PASS (701ns)
```

```
data.main.test_services_issue_warning: PASS (614ns)
```

```
-----  
PASS: 4/4
```


Not much public rego code yet

```
In:path .rego extension:rego
```

546 results

Open Policy Agent Bundles



Open Policy Agent

v0.10.7 latest ▾

DOCUMENTATION

- Introduction
- How Does OPA Work?
- How Do I Write Policies?
- How Do I Test Policies?
- Language Reference
- Configuration Reference
- REST API
- Deployments
- Bundles**
- Bundle Service API
- Bundle File Format
- Multiple Sources of Policy and Data
- Debugging Your Bundles

Status

Decision Logs

Bundles

OPA can periodically download bundles of policy and data from remote HTTP servers. The policies and data are loaded on the fly without requiring a restart of OPA. Once the policies and data have been loaded, they are enforced immediately. Policies and data loaded from bundles are accessible via the standard OPA [REST API](#).

Bundles provide an alternative to pushing policies into OPA via the REST APIs. By configuring OPA to download bundles from a remote HTTP server, you can ensure that OPA has an up-to-date copy of policies and data required for enforcement at all times.

OPA can only be configured to download one bundle at a time. You cannot configure OPA to download multiple bundles. By default, the OPA REST APIs will prevent you from modifying policy and data loaded via bundles. If you need to load policy and data from multiple sources, see the section below.

See the [Configuration Reference](#) for configuration details.

Bundle Service API

OPA expects the service to expose an API endpoint that serves bundles. The bundle API should allow clients to download named bundles.

```
GET /<bundle_prefix>/<name> HTTP/1.1
```

Reusing OCI registries

STEVE LASKER

[Home](#)

[Contact](#)

stevelas / January 25, 2019 / CI/CD, Container Registry, DevOps, Docker, Docker Registry

Cloud Native Artifact Registries evolve from Docker Container Registries

Search



Proposed OCI media types of OPA

open-policy-agent / opa

Watch 65

Star 2,031

Fork 186

Code

Issues 88

Pull requests 4

Projects 1

Wiki

Insights

Feedback on mediaTypes for storing OPA bundles as OCI images #1413

Edit

New issue

Open

garethr opened this issue 5 hours ago · 0 comments



garethr commented 5 hours ago

+ 👤 ⋮

As discussed briefly with @tsandall. Posting for visibility and to widen the discussion.

I've been hacking on <https://github.com/instrumenta/confest>, which uses OPA/rego but presents an interface for local unit testing of configuration. I'll be talking at KubeCon in a few weeks about this and why I think it's useful.

One thing I've added recently is the ability to share rego files on OCI registries. Basically you can do the following and download existing rules or other bits.

```
conftest pull instrumenta.azurecr.io/kubernetes-helpers
```

That spun off work I was doing with @SteveLasker (product manager for Azure Container Registry at Microsoft) before I left Docker. Basically better support in registries for other types of content than just Docker images. Steve has a proposal up at:

<https://github.com/SteveLasker/RegistryArtifactTypes/blob/master/mediaTypes.md>

The rationale for sharing things via OCI images is described in this blog post <https://stevlasker.blog/2019/01/25/cloud-native-artifact-stores-evolve-from-container-registries/>. But in short, everyone already has one (whether cloud provider, public/private, self-hosted, geo-replicated, etc.)

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe

You're receiving notifications because you authored the thread.

1 participant

Using conftest to share policy

```
$ ls policy
```

```
$ conftest pull instrumenta.azurecr.io/kubernetes-helpers
```

```
$ ls policy
```

```
kubernetes.rego
```

General helpers

```
package main
```

```
# has_field returns whether an object has a field
```

```
has_field(object, field) {  
    object[field]  
}
```

```
# False is a tricky special case, as false responses would create an undefined  
# document unless they are explicitly tested for
```

```
has_field(object, field) {  
    object[field] == false  
}
```

```
has_field(object, field) = false {
```

Domain-specific packages

```
package kubernetes
```

```
is_service {  
    input.kind = "Service"  
}
```

```
is_deployment {  
    input.kind = "Deployment"  
}
```

Test helpers

```
package main
```

```
empty(value) {  
    count(value) == 0  
}
```

```
no_violations {  
    empty(deny)  
}
```

```
no_warnings {  
    empty(warn)  
}
```


Dependencies in conftest.toml

```
# You can override the directory in which to store and look for policies  
policy = "tests"
```

```
# You can override the namespace which to search for rules  
namespace = "conftest"
```

```
# An array of individual policies to download. Only the repository  
# key is required. If tag is omitted then latest will be used
```

```
[[policies]]  
repository = "instrumenta.azurecr.io/test"  
tag = "latest"
```

Using conftest to stay updated

```
$ ls policy
$ conftest update
$ ls policy
kubernetes.rego
```

Portability

Helping to move between different Kubernetes tools

Demo

Kustomize

```
$ kustomize build | conftest test -
```

Helm

```
$ helm template | conftest test -
```

Typescript

```
import {Pod} from 'kubernetes-types/core/v1'
import {ObjectMeta} from 'kubernetes-types/meta/v1'
import * as yaml from 'js-yaml'

let metadata: ObjectMeta = {name: 'example', labels: {}}

let pod: Pod = {
  apiVersion: 'v1',
  kind: 'Pod',
  metadata,
  spec: {
    containers: [
      /* ... */
    ],
  },
}

console.log(yaml.safeDump(pod))
```

Typescript

```
$ npx ts-node pod.ts | conftest test -
```


Kubectl (look away now)

```
$ kubectl get all -o json \  
  | jq -cj '.items[] | tostring+"\u0000"' \  
  | xargs -n1 -0 -I@ bash -c "echo '@' | conftest test -"
```

Kubectl plugin

```
$ kubectl krew install conftest
```

```
$ kubectl conftest deployment some-deployment
```

Cue

```
package kubernetes

deployment "hello-kubernetes": {
  apiVersion: "apps/v1"
  spec: {
    replicas: 3
    template spec containers: [{
      image: "paulbouwer/hello-kubernetes:1.5"
      ports: [{
        containerPort: 8080
      }]
    }]
  }
}
```

Cue

```
package kubernetes

import "encoding/yaml"

command test: {
  task conftest: {
    kind: "exec"
    cmd: "conftest test -"
    stdin: yaml.MarshalStream(objects)
  }
}
```

Cue

```
$ cue test
```

```
Containers must not run as root
```

```
--- .
```

```
command "conftest test -" failed: exit status 1  
terminating because of errors
```

Not just Kubernetes

Lots of other configurations to care about

Lots of Kubernetes-like docs

```
apiVersion: kind: extension:yaml
```

1,054,453 results

Lots more YAML

```
in:path .yaml language:YAML
```

69,822,306 results

Demo

Serverless framework

```
service: aws-python-scheduled-cron

frameworkVersion: ">=1.2.0 <2.0.0"

provider:
  name: aws
  runtime: python2.7
  tags:
    author: "this field is required"

functions:
  cron:
    handler: handler.run
    runtime: python2.7
    events:
      - schedule: cron(0/2 * ? * MON-FRI *)
```

Serverless framework

```
package main

deny[msg] {
  input.provider.runtime = "python2.7"
  msg = "Python 2.7 cannot be the default provider runtime"
}

runtime[name] {
  input.functions[i].runtime = name
}

deny[msg] {
  runtime["python2.7"]
  msg = "Python 2.7 cannot be used as the runtime for functions"
}

deny[msg] {
  not has_field(input.provider.tags, "author")
  msg = "Should set provider tags for author"
}
```

Terraform

```
$ terraform plan -out config.tfplan
```

```
$ terraform show -json config.tfplan | configtest test -
```

Terraform

```
package main

blacklist = [
    "google_iam",
    "google_container"
]

deny[msg] {
    check_resources(input.resource_changes, blacklist)
    banned := concat(", ", blacklist)
    msg = sprintf("Terraform plan will change prohibited resources in: %v", [banned])
}

# Checks whether the plan will cause resources with certain prefixes to change
check_resources(resources, disallowed_prefixes) {
    startswith(resources[_].type, disallowed_prefixes[_])
}
}
```

Docker Compose

```
version: "3.4"  
services:  
  web:  
    build: .  
    ports:  
      - "5000:5000"  
  redis:  
image: "redis:latest"
```

Docker Compose

```
package main

version {
  to_number(input.version)
}

deny[msg] {
  endswith(input.services[_].image, ":latest")
  msg = "No images tagged latest"
}

deny[msg] {
  version < 3.5
  msg = "Must be using at least version 3.5 of the Compose file format"
}
```

Conclusions

If all you remember is...

Summary

- **Open Policy Agent** is incredibly flexible
- Expect lots more integrations in the future
- Policy is a good starting point for conversations
- Managing configuration as code needs better tools

Come talk to **Snyk** at booth #S41

Questions?

And thanks for listening