Adding policies to your apps

Marcin Wyszynski, spacelift.io

Background

- Designed and built sales tools at Google;
- Built tooling for cold storage (tape backup) systems at Google;
- Built cluster management tools at Facebook;
- Former CTO of a static code analysis startup (codebeat.co);
- Founded a startup (spacelift.io) providing a specialized CI/CD platform for infra-as-code;

Common theme: building things for **power users**.

What do power users want?

- APIs;
- Event notifications (eg. webhooks);
- Lots of options, tweaks and toggles;

The last bit is a **nightmare to build and maintain**, and will never cover *all* the use cases. We can do better.

Exhibit 1: endless forms

*Require status checks to pass **unless** you're an admin **and** one of the other admins approved your PR, **or** it's outside of business hours and you're only tweaking documentation **and** you have linear history.*

Lesson #1 : GUI forms suck at implementing complex logic.

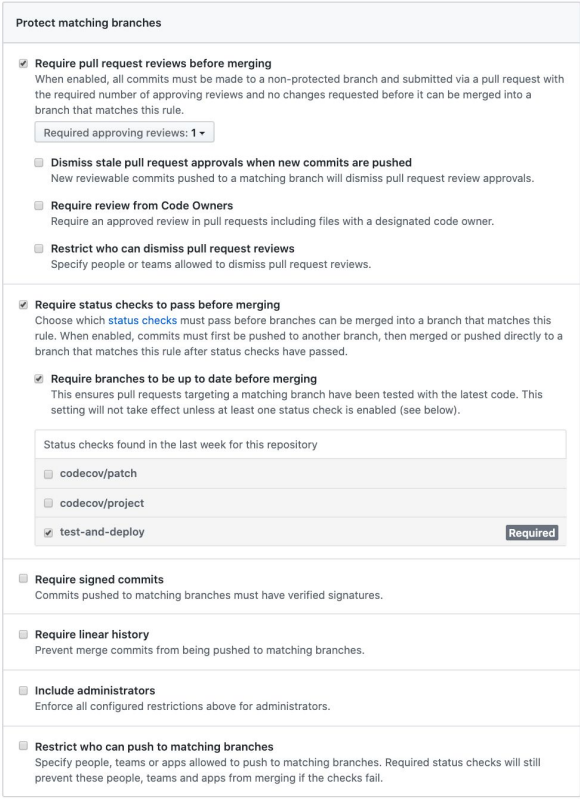


Exhibit 2: markup languages

config.yml — backend

config.yml

.circleci > config.yml

```yaml
workflows:
  version: 2
  test-and-deploy:
    jobs:
      - test-code
      - test-build: { filters: { branches: { ignore: ["master"] } } }
      - deploy: { filters: { branches: { only: ["master"] } } }
```

*Test build of every branch that starts with "test" **and all** tags that conform to semantic versioning format.*

Lesson #2: markup languages still suck at implementing complex logic, but require much less work.

Exhibit 3: user code snippets

I know: let's embed a Lua* interpreter and allow users to write snippets of custom code that are executed with the input representing **all the possibly relevant data**, have it return one of the **predetermined values** (*decisions*), and have your app act on this decision.

Yay, we've just **decoupled** decision **making** from decision **enforcement**.

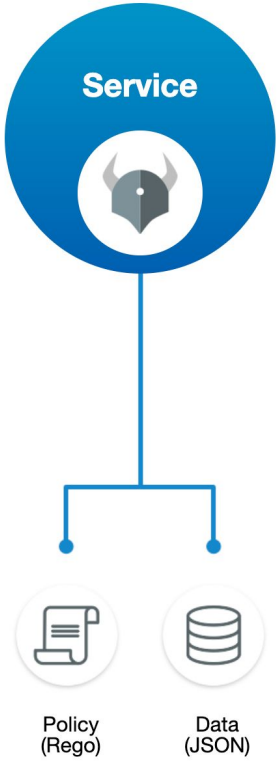
But is there a simpler, safer way?

* or any other interpreted language...

What do we **actually** need?

- a query / transformation language;
- for loosely structured data;
- with JSON support;

Meet Open Policy Agent and **Rego**.



Wait, a *policy*?

"A set of ideas or a plan of **what to do** in **particular situations** that has been **agreed to officially** by a group of people, a business organization, a government, or a political party".

In the context of your application, a policy would mean a **predefined query** on **structured data** with a **constrained set of possible results**.

Simple examples: AWS IAM policies, Kubernetes RBAC policies.

Exhibit #1 - user logging in

Structured data:

- remote IP;
- current timestamp;
- login (username);
- full name;
- group/org memberships;

Possible decision(s):

- allow;
- deny;

Exhibit #2 - user level of access

Structured data: see Exhibit #1

Possible decisions:

- allow;
- deny;
- **admin**;
- **deny admin** - useful to define exceptions;

Exhibit #3

Challenge: *make configuration changes only during business hours* ***AND*** *from the office IP.*

Because if things go wrong, we want you to be available, and have all the help you need.

Make changes from the office

Stack access policy

```
1  package spacelift
2
3  now     := input.request.timestamp_ns
4  clock   := time.clock([now, "America/Los_Angeles"])
5  weekend := { "Saturday", "Sunday" }
6  weekday := time.weekday(now)
7  office  := "12.34.56.0/24"
8  ip      := input.request.remote_ip
9
10 write      { input.session.teams[_] == "Product team" }
11 deny_write { weekend[weekday] }
12 deny_write { clock[0] < 9 }
13 deny_write { clock[0] > 17 }
14 deny_write { not net.cidr_contains(office, ip) }
15
```

Why use Rego?

- it makes running custom code **safe**:
  - **non Turing-complete**, eg. no loops or conditionals;
  - guaranteed to **terminate**;
  - ensures that **queries are correct and unambiguous**;
- it is **declarative** - you focus on the outcome, not the process;
- supports **arbitrarily nested** documents (eg. graphs);
- is **readable** and easier for non-programmers than eg. Lua;

Rego vs. OPA

- **Rego** is a **query language** that can be directly embedded in a Go application using a library;
- **Open Policy Agent** is an **executable** deployed as a separate service or a sidecar to provide policy evaluation as an **HTTP service**;
- **Open Policy Agent** uses **Rego** as its query language;
- But today we're going to use the **Rego** library *directly*;

Simple (Re)Go example

```go
const timeout = 20 * time.Millisecond

// Evaluate evaluates input against a policy.
func Evaluate(policyBody string, input interface{}) (rego.ResultSet, error) {
	ctx, cancel := context.WithTimeout(context.Background(), timeout)
	defer cancel()

	base := rego.New(
		rego.Query("data.spacelift.result"),
		rego.Module("policy", policyBody),
	)

	preparedQuery, err := base.PrepareForEval(ctx)
	if err != nil {
		return nil, err
	}

	// Note: you will still need to dig a bit in the returned structure to
	// get your desired value.
	return preparedQuery.Eval(ctx, rego.EvalInput(input))
}
```

Challenge #1: does the query make sense?

```
const defaults = `
package spacelift

# This is what Spacelift will query for when evaluating policies.
result = {
        "deny": deny,
        "deny_write": deny_write,
        "read": read,
        "write": write,
}

default deny = false
default deny_write = false
default read = false
default write = false
`

// Evaluate evaluates input against a policy.
func Evaluate(policyBody string, input interface{}) (rego.ResultSet, error) {
        ctx, cancel := context.WithTimeout(context.Background(), timeout)
        defer cancel()

        base := rego.New(
                rego.Query("data.spacelift.result")
                rego.Module("defaults", defaults),
                rego.Module("policy", policyBody),
        )
```

Now whatever the user does in their part of the policy, the query will always return what we expect.

Challenge #2: ensuring that the decision is safe

```go
var unsafeBuiltins = map[string]struct{}{
	"http.send":         {},
	"opa.runtime":       {},
	"rego.parse_module": {},
	"time.now_ns":       {},
	"trace":             {},
}

// Evaluate evaluates input against a policy.
func Evaluate(policyBody string, input interface{}) (rego.ResultSet, error) {
	ctx, cancel := context.WithTimeout(context.Background(), timeout)
	defer cancel()

	base := rego.New(
		rego.Query("data.spacelift.result"),
		rego.Module("defaults", defaults),
		rego.Module("policy", policyBody),
		rego.UnsafeBuiltins(unsafeBuiltins),
	)
```

Now the decision will only be made **based on the input**.

Notice what we consider unsafe. Why?

Rego and OPA in the wild

- AWS API Gateway
- Boomerang Bosun Policy Gating
- Ceph Object Storage Authorization
- Kubernetes Admission Control using Vulnerability Scanning
- Cloudflare Worker Enforcement of OPA Policies Using WASM
- Conftest -- Configuration checking
- Custom Application Authorization
- Library-based Microservice Authorization
- HTTP API Authorization in Dart
- Docker controls via OPA Policies
- Elasticsearch Data Filtering
- Container Network Authorization with Envoy
- GCP audit with Forseti
- Gloo API Gateway
- Gradle Build Plugin
- IPTables
- Container Network Authorization with Istio (at the Edge)
- Container Network Authorization with Istio (as part of Mixer)
- Jenkins Job Trigger Policy Enforcement
- Kafka Topic Authorization
- API Gateway Authorization with Kong
- Kubernetes Authorization
- Kubernetes Provisioning
- Kubernetes Admission Control
- Secure Kubernetes using eBPF & Open Policy Agent
- SSH and Sudo Authorization with Linux
- Minio API Authorization
- OpenFaaS Serverless Function Authorization
- HTTP API Authorization in PHP
- Spinnaker Pipeline Policy Enforcment
- Authorization for Java Spring Security
- SQL Database Data Filtering
- Terraform Authorization
- Traefik API Gateway

Further reading

- openpolicyagent.org
- play.openpolicyagent.org
- docs.spacelift.io/concepts/policy

Questions?

marcinw@spacelift.io