Open Policy Agent - Gatekeeper

Admission Control with Ultimate Expressiveness

Admission Controllers

Open Policy Agent Gatekeeper can do far more than the in-tree Pod Security Standards and Pod Security Policies admission controllers.

https://github.com/open-policy-agent/gatekeeper

OPA Gatekeeper is incredibly expressive, as we'll see. It does have a somewhat steep learning curve, because it uses a domain specific language, so we also look at Kyverno.

OPA Gatekeeper Introduction

Open Policy Agent Gatekeeper is incredibly expressive, allowing you to create any constraints on a cluster that you can code into OPA's Datalog-inspired language, Rego.

Rather than applying a number of Pod Security Policies, it applies governing policies in the form of Constraint Templates and Constraints.

Rego is a custom language created by OPA:

https://www.openpolicyagent.org/docs/latest/policy-language/

Structure of a Governing Policy

A governing policy item for Gatekeeper has two necessary files:

Constraint Template: has Rego to evaluate a proposed resource (pod, namespace…)

Constraint: creates rules by filling in specific values into the Constraint template

Let's look at how this works on the next two slides.

Constraint Template Example

Sections:

- CRD – defines a custom resource
- Rego:
  - Package name
  - Functions to help parse a resource
  - Rules that evaluate the resource and return violations, with reasons in their messages.

```
apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        ...
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        ... functions to help parse ...

        violation[{"msg": msg, "details": {"missing_labels": missing}}] {
          ...
          def_msg := sprintf("you must provide labels: %v", [missing])
          msg := get_message(input.parameters, def_msg)
        }
```

Template Section: Custom Resource (CRD)

In this example, we have a custom resource defined, K8SRequiredLabels.

This template will accept some set of labels that must be present.

It also accepts a regular expression to define acceptable label values.

```
crd:
  spec:
    names:
      kind: K8sRequiredLabels
    validation:
      # Schema for the `parameters` field
      openAPIV3Schema:
        properties:
          message:
            type: string
          labels:
            type: array
            items:
              type: object
              properties:
                key:
                  type: string
                allowedRegex:
                  type: string
```

Template Section: Rego with Rule 1

```
violation[{"msg": msg, "details": {"missing_labels": missing}}] {
  provided := {label | input.review.object.metadata.labels[label]}
  required := {label | label := input.parameters.labels[_].key}
  missing := required - provided
  count(missing) > 0
  def_msg := sprintf("you must provide labels: %v", [missing])
  msg := get_message(input.parameters, def_msg)
}
```

- parse the labels in the resource being evaluated into a set called `provided`
- parse the labels in the specific constraint (rule) into a set called `required`
- if `required` set members aren't in `provided`, store them in a set called `missing`
- if `missing` isn't empty, trigger a violation and print what labels are missing, as well as a dump of the resource that's failing the constraint.

Template Section: Rego with Rule 2

```
violation[{"msg": msg}] {
  value := input.review.object.metadata.labels[key]
  expected := input.parameters.labels[_]
  expected.key == key
  # do not match if allowedRegex is not defined, or is an empty string
  expected.allowedRegex != ""
  not re_match(expected.allowedRegex, value)
  def_msg := sprintf("Label <%v: %v> does not satisfy allowed regex: %v", [key, value, expected.allowedRegex])
  msg := get_message(input.parameters, def_msg)
}
```

- for each label specified (`key`), put its value into `value`.
- if the constraint (specific rule) specifies an regular expression in `allowedRegex`:
  - check for a regular expression match failure
  - alert on a failed match, showing the label (`key`), its `value`, and the regular expression it failed to match (`allowedRegex`)

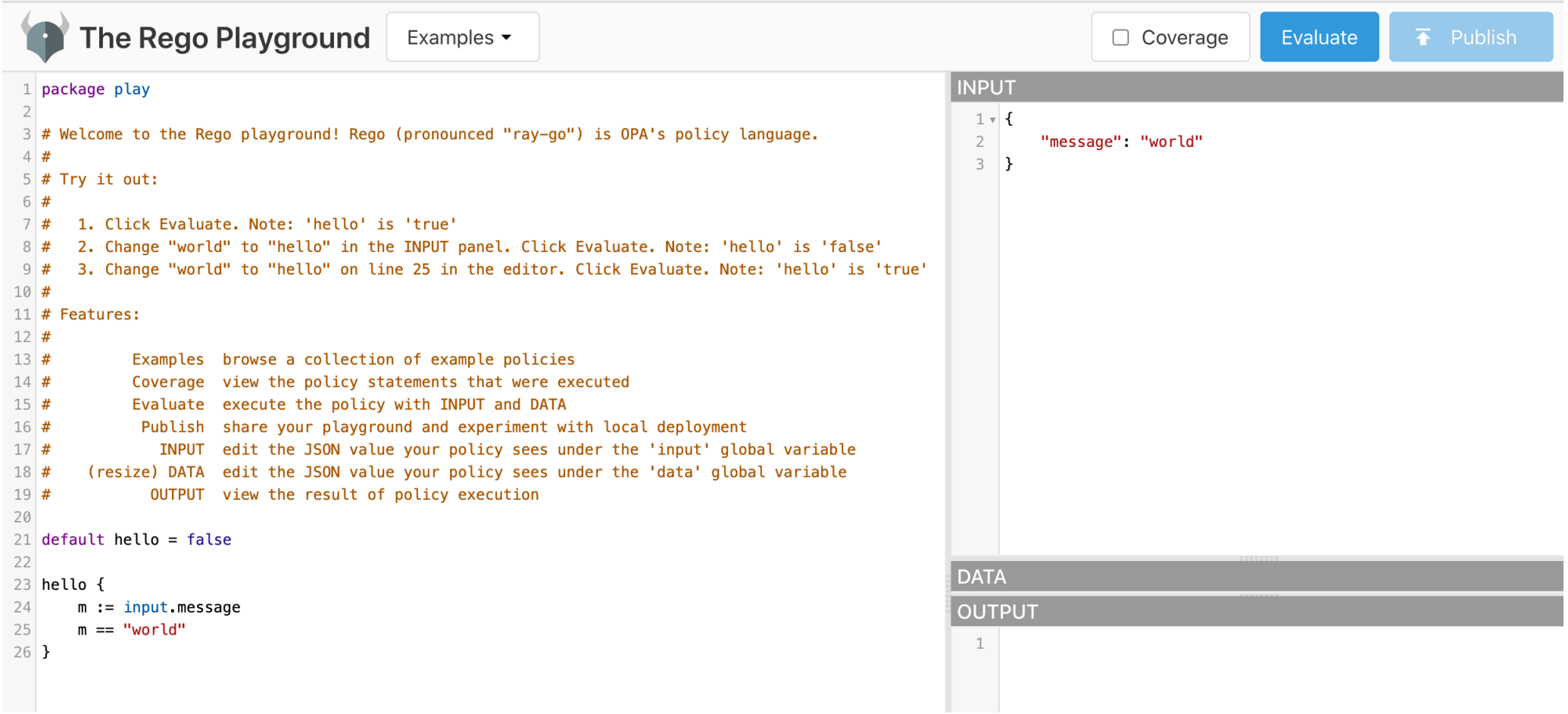
Example Constraint

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: all-must-have-owner
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Namespace"]
  parameters:
    message: "All namespaces must have an `owner` label that points to your company username"
    labels:
      - key: owner
        allowedRegex: "^[a-zA-Z]+.agilebank.demo$"
```

kind is the new CRD

This rule refers only to namespaces.

Namespaces must have an "owner" label, which must end in ".agilebank.demo"



https://play.openpolicyagent.org/

Deploying OPA Gatekeeper

Deploy OPA Gatekeeper from the current master version:

```
kubectl apply -f  https://raw.githubusercontent.com/open-policy-agent/gatekeeper/master/deploy/gatekeeper.yaml
```

Grab a local copy of the Gatekeeper source repository, so we can get the constraint library:

https://github.com/open-policy-agent/gatekeeper/archive/master.zip

OPA Gatekeeper Library of Constraints

Browse the library of pre-written constraint templates here:

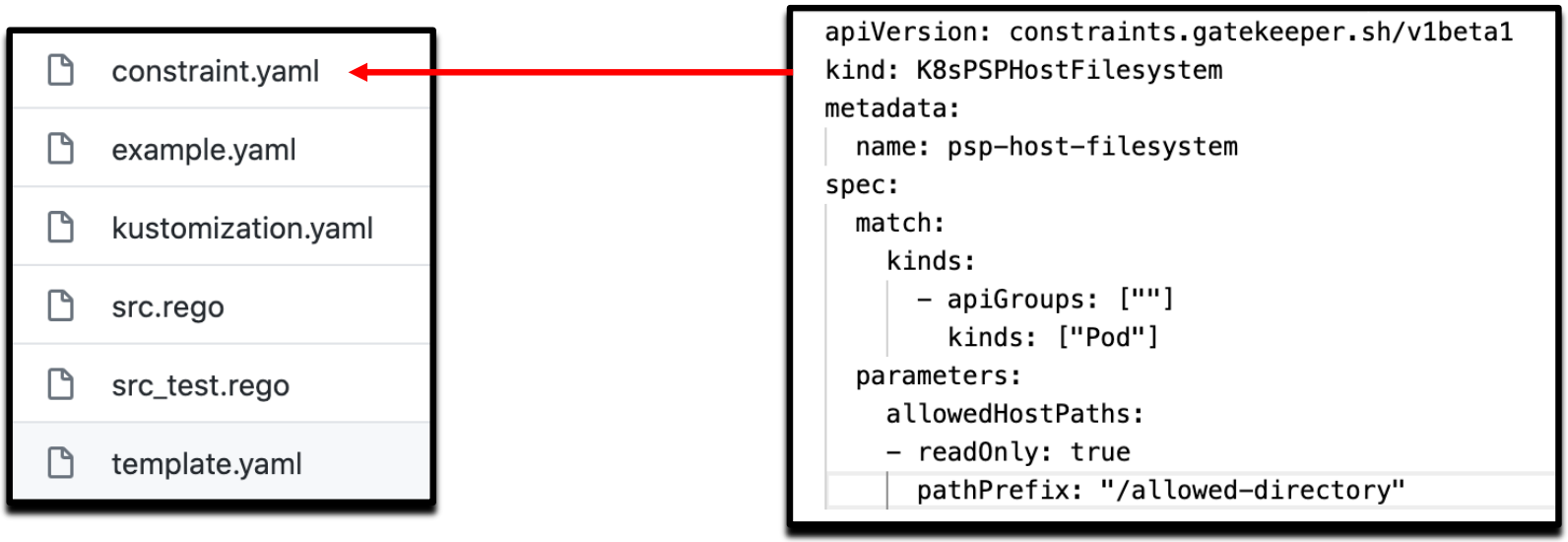
https://github.com/open-policy-agent/gatekeeper-library/

There are constraint templates meant to mirror pod security policies (PSP).

https://github.com/open-policy-agent/gatekeeper-library/tree/master/library/pod-security-policy

OPA Gatekeeper PSP-Equivalent Constraints

To try out OPA Gatekeeper, you could apply the host-filesystem constraint template to the first Bustakube scenario, blocking the attack pod's mounting of the node's root filesystem.



https://github.com/open-policy-agent/gatekeeper-library/tree/master/library/pod-security-policy/host-filesystem

Consideration: Image Provenance

- It's critical that you understand the upstream source of your container images.
- Are your images cryptographically signed?
- Have you scanned them with CoreOS Clair?
- Who created them?

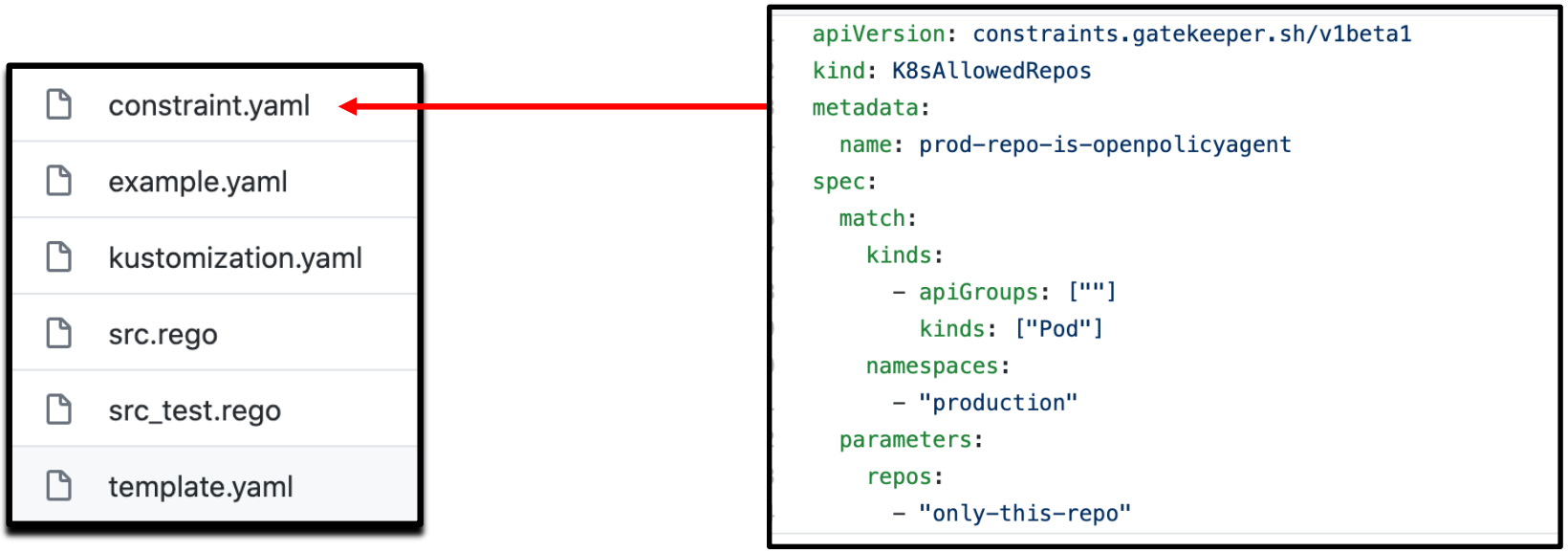
References:

https://docs.docker.com/engine/security/trust/

https://docs.docker.com/engine/security/trust/content_trust/

Use OPA Gatekeeper to Enforce Image Hygeine

Force production namespace pods to use container images from only "only-this-repo".



https://github.com/open-policy-agent/gatekeeper-library/tree/master/library/general/allowedrepos

OPA Gatekeeper

OPA Gatekeeper is powerful and in active development.

Use the Rego Playground to tweak existing library items, creating your own from scratch only where necessary.

If you do create constraint templates from scratch, please consider submitting them to the OPA Gatekeeper repo.

Exercise: OPA Gatekeeper

Please:

Open the Firefox browser on the class machine to:
http://localhost:10000/exercises/kubernetes-opa