

Managing your Policies and Standards

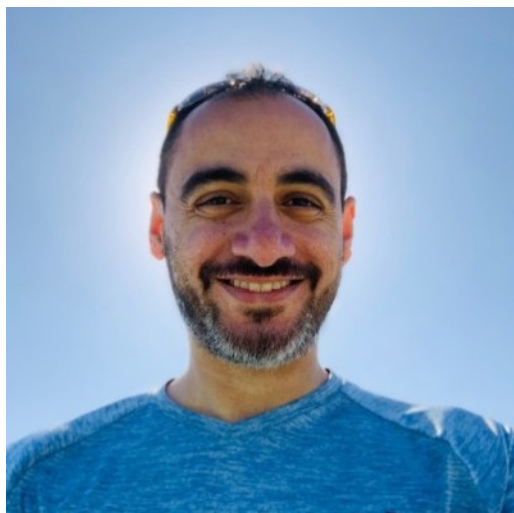
How to Create Policies with Rego and OPA

Ahmed Badran

CTO



About Me



Ahmed Badran

CTO



MAGALIX



Language
Technologies
Institute

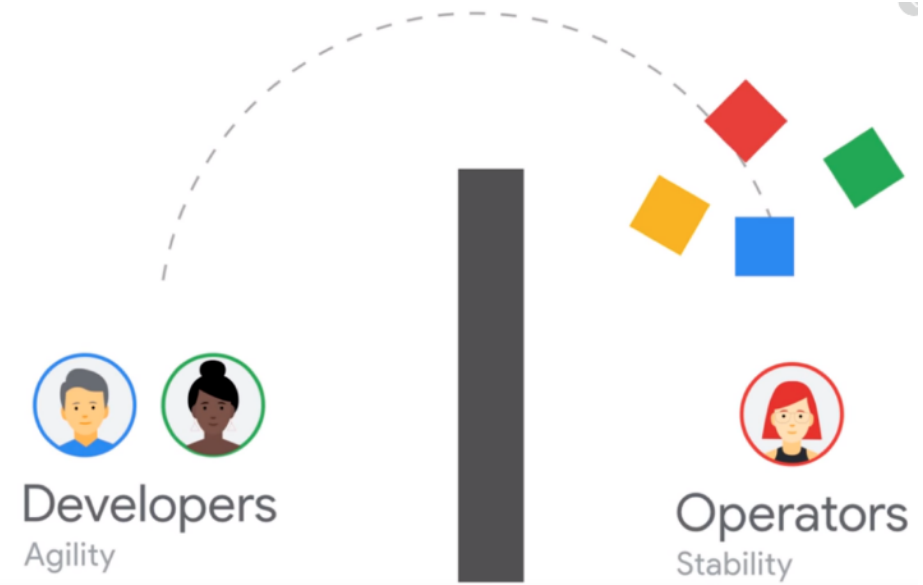


The
WALT DISNEY
Company



The Cloud-Native Challenge

Agility vs. Stability



You are not alone!

DocuSign

Barracuda

Pinterest

Expedia

NETFLIX

MONSANTO

NORDSTROM

whitepages

Objectives

- What is governance and why it is important
- How to establish a governance framework
- How Open Policy Agent and the Rego language could help
- Example policies for Kubernetes

“Based on a True Story”



Amr Emam 2:52 PM

@channel on dev, whose workload is this?

namespace: application-system

application-controller-manager-0

Issue

- Could this be malicious?
- Who should we contact?
- Can I update it safely?

Solution?

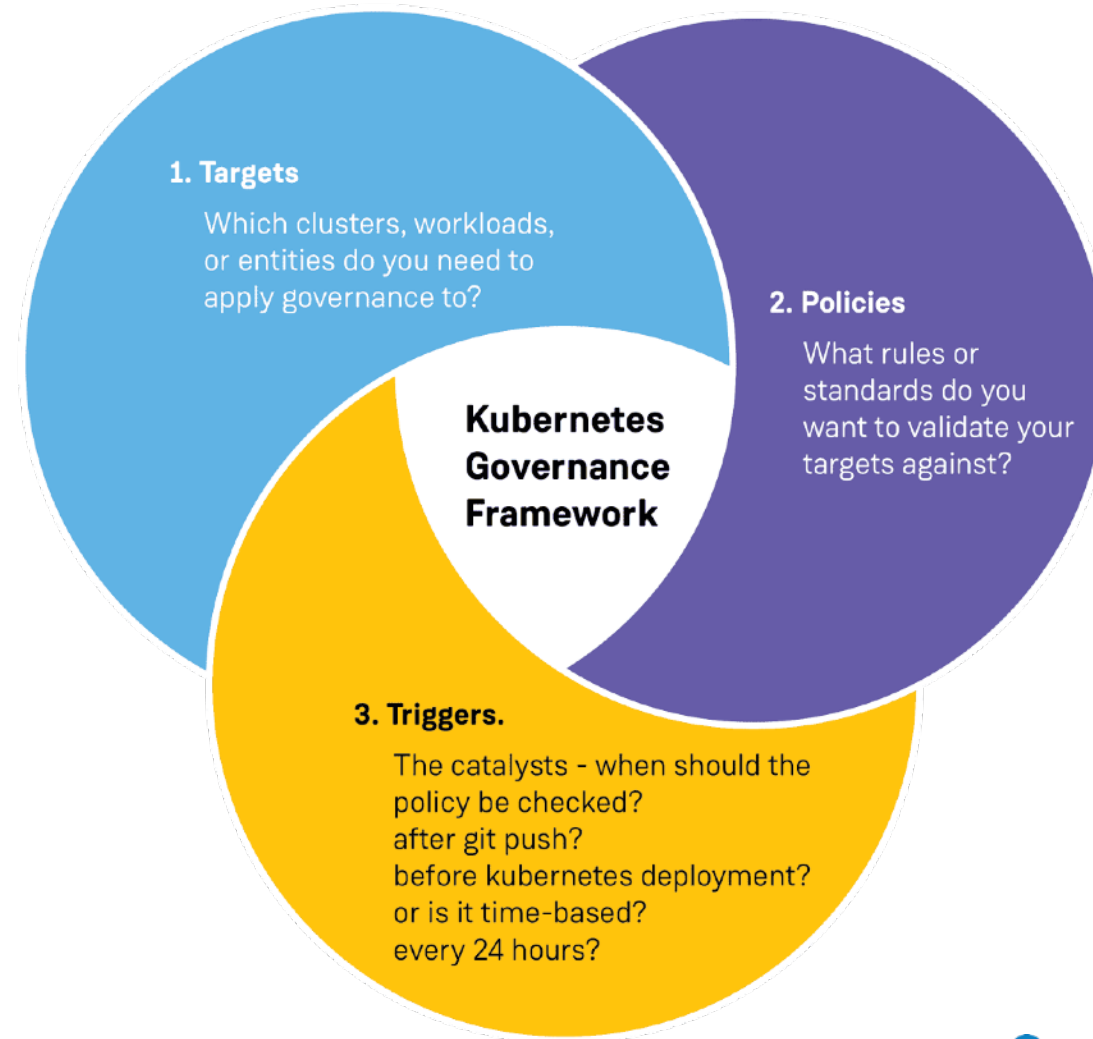
- Block all dev deployments?
- Force PR on all production changes?
- There is gotta be a better way!

Governance Framework - Policy as Code

Governance: The ability of the operations team to verify and enforce certain policies and standards across the entire organization or within specific clusters

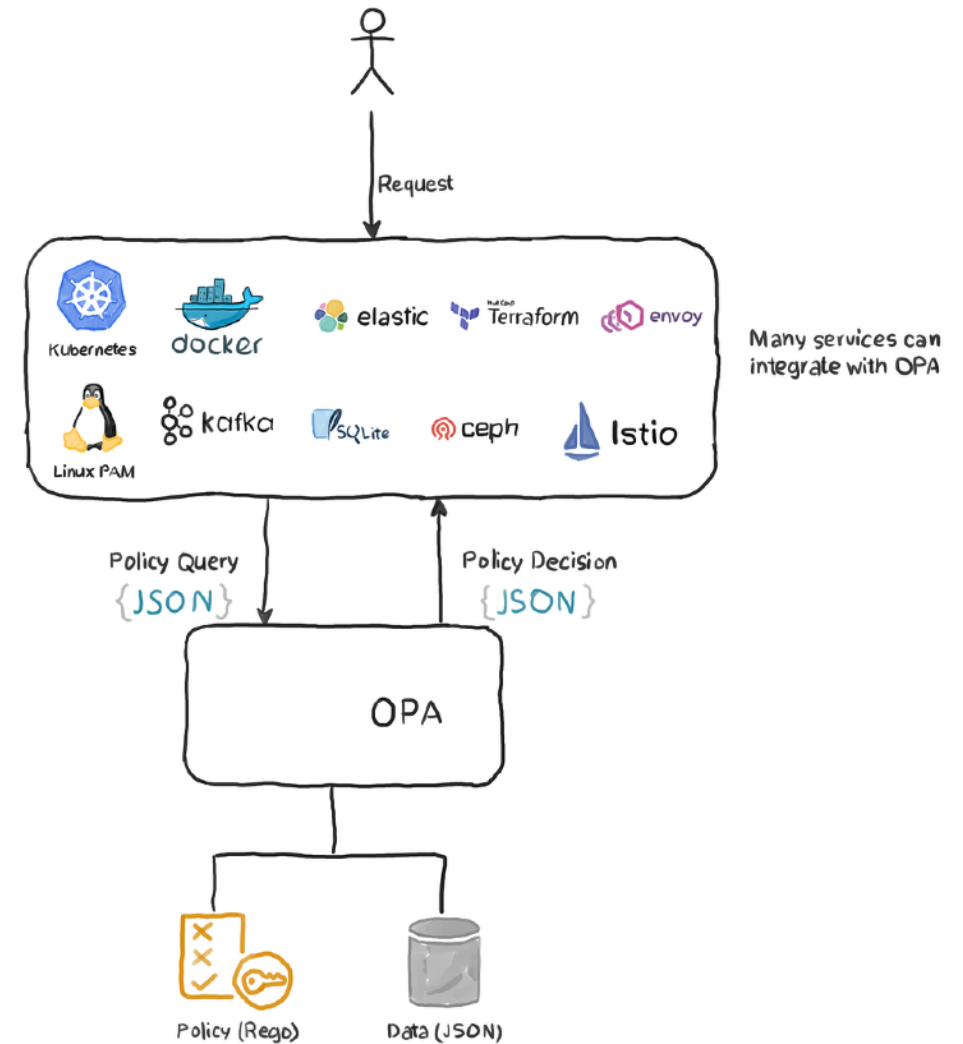
Governance Framework

- **Targets**
which entities
- **Policies**
what rules
- **Triggers**
when to run



Open Policy Agent

OPA: as part of the CNCF project, the Open Policy Agent (OPA) is a great tool that allows organizations to easily define custom policies for their Kubernetes environments. Open Policy Agent policies are written in a declarative policy language called [Rego](#)

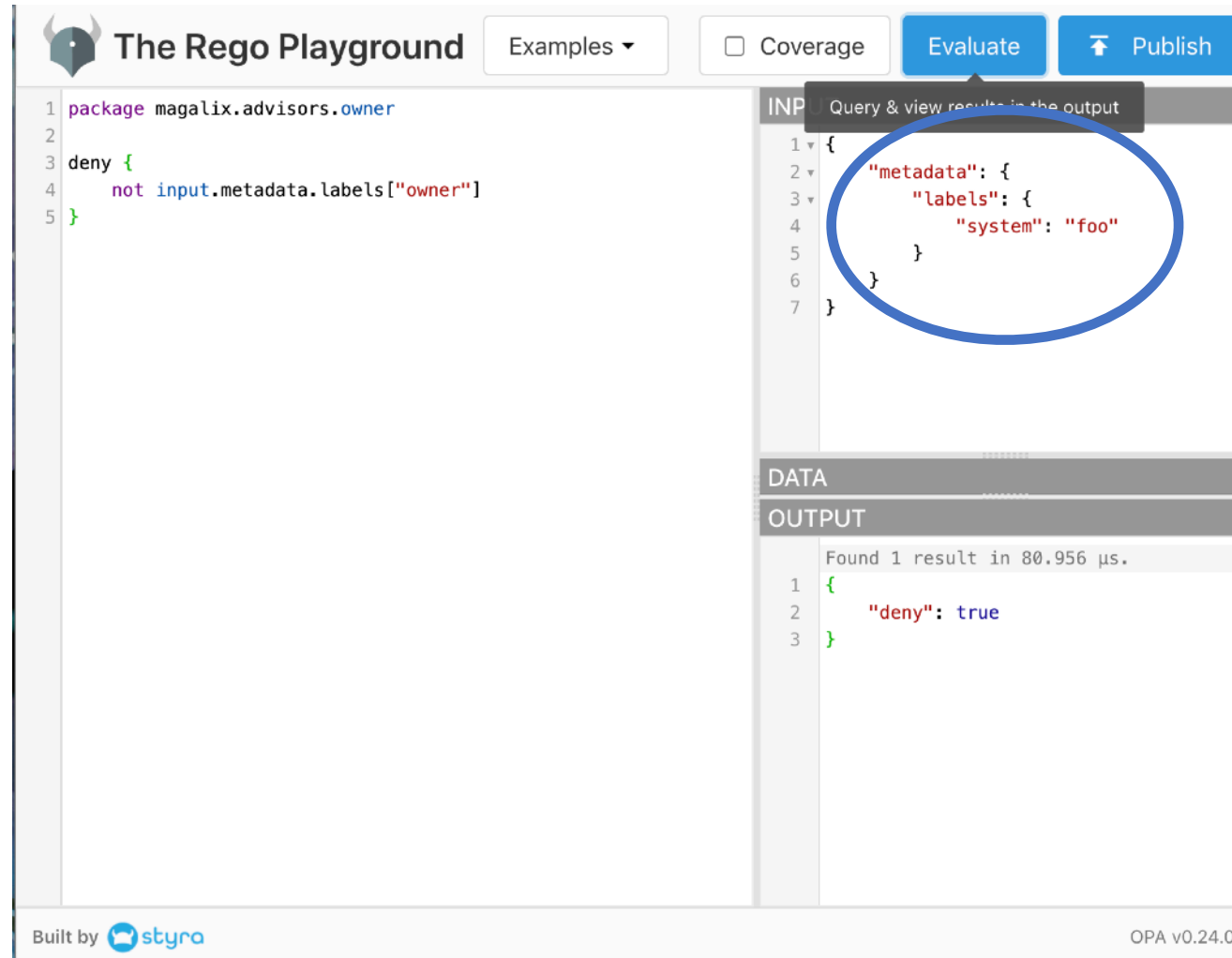


Rego Language

Rego: a declarative language to define policies where statements are assertions that evaluate to true or false.

```
1 package magalix.advisors.owner
2
3 deny {
4     not input.metadata.labels["owner"]
5 }
```

Rego Playground



The screenshot displays the 'The Rego Playground' interface. On the left, a code editor contains a Rego policy:

```
1 package magalix.advisors.owner
2
3 deny {
4   not input.metadata.labels["owner"]
5 }
```

On the right, the 'Evaluate' button is highlighted. Below it, the 'INPUT' section shows a JSON object with a blue circle around the 'labels' field:


```
1 {
2   "metadata": {
3     "labels": {
4       "system": "foo"
5     }
6   }
7 }
```

Below the input, the 'DATA' and 'OUTPUT' sections are visible. The 'OUTPUT' section shows the result of the evaluation:

```
Found 1 result in 80.956 µs.
1 {
2   "deny": true
3 }
```

At the bottom, the footer indicates 'Built by styra' and 'OPA v0.24.0'.


Rego Playground

 The Rego Playground

Examples ▾

☐ Coverage

Evaluate

 Publish

```
1 package magalix.advisors.owner
2
3 deny {
4   not input.metadata.labels["owner"]
5 }
```


INPUT

1 {
2 "metadata": {
3 "labels": {
4 "owner": "foo"
5 }
6 }
7 }

DATA

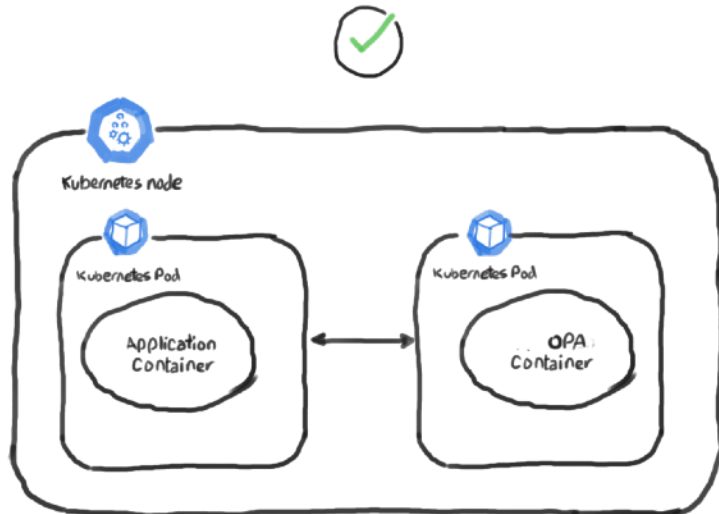
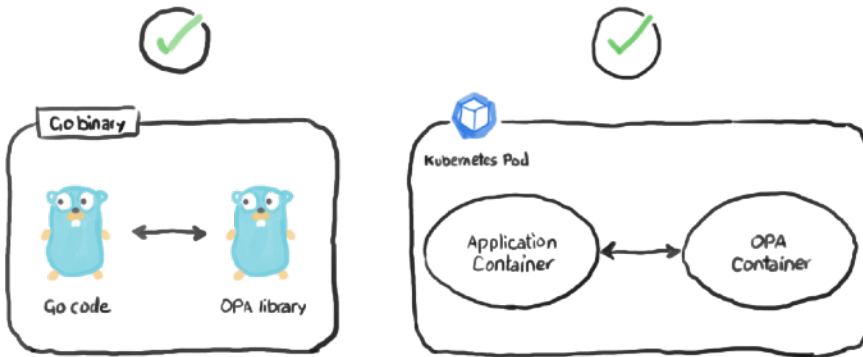
OUTPUT

1 Found 1 result in 80.859 μs.
2 {}

Built by  styra

OPA v0.24.0

Deployment



```
1 import "github.com/open-policy-agent/opa/rego"
2
3 query, err := rego.New(
4     rego.Query("x = magalix.adivosr.owner.deny"),
5     rego.Module("example.rego", module),
6     ).PrepareForEval(ctx)
```

Options

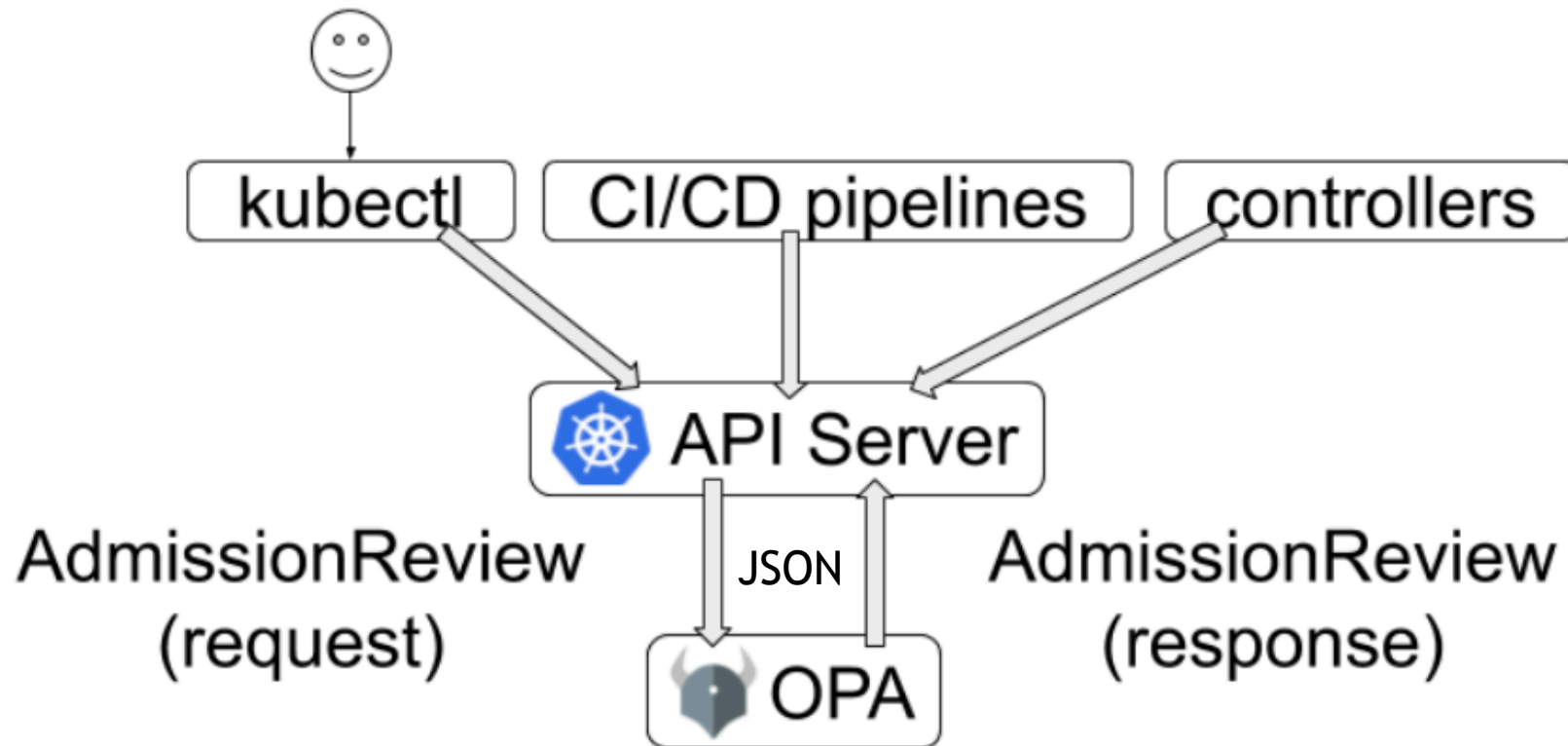
- As Go library
- As REST API
- As Sidecar container (kube-mgmt)

Gatekeeper

An extensible, parameterized policy library with native Kubernetes CRDs that support audit functionalities and instantiating and extending the policy library.



Gatekeeper 3.0 Architecture



Gatekeeper - Native Kubernetes CRDs

```
1  apiVersion: templates.gatekeeper.sh/v1beta1
2  kind: ConstraintTemplate
3  metadata:
4    name: advisorowner
5  spec:
6    crd:
7      spec:
8        names:
9          kind: AdvisorOwner
10   targets:
11     - target: admission.k8s.gatekeeper.sh
12       rego: |
13         package advisorowner
14
15         violation[{"msg": msg, "details": {"missing_labels": missing}}] {
16           provided := {label | input.review.object.metadata.labels[label]}
17           required := {label | label := input.parameters.labels[_]}
18           missing := required - provided
19           count(missing) > 0
20           msg := sprintf("Deployment pods must contain labels: %v", [missing])
21         }
```

Template

```
1  apiVersion: constraints.gatekeeper.sh/v1beta1
2  kind: AdvisorOwner
3  metadata:
4    name: must-have-owner
5  spec:
6    match:
7      kinds:
8        apiGroups: ["apps"]
9        kinds: ["Deployment"]
10   parameters:
11     labels: ["owner"]
```

Constraints define Scope and Intent

Real Time Enforcement

Applying Deployment Manifest

- No owner label
- Governance in real-time

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demoservice
  labels:
    app.kubernetes.io/name: demoservice
    app: demoservice
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demoservice
```

```
# kubectl apply -f bad-deployment.yml
Error from server ([denied by must-have-owner] Deployment pods must contain labels: {"owner"}): error when creating
"bad-deployment.yml": admission webhook "validation.gatekeeper.sh" denied the request: [denied by must-have-owner]
Deployment pods must contain labels: {"owner"}
```


Example Policies

- Check that readinessProbe and livenessProbe are defined in your containers spec to guarantee that only healthy pods get traffic
- Enforce the settings of allowPrivilegeEscalation=false and mustRunAsNonRoot so the container and its child process cannot escalate their privilege
- Verify that the spec's replicas count is 2 or greater, to ensure redundancy in your ReplicaSets for fault tolerance.
- Ensure that affinity.podAntiAffinity is set in your deployment spec to avoid having multiple pods - from the same deployment - running on the same node.
- Check that no RoleBinding objects give patch access to users that you haven't approved.
- Check that container.image in all your specs are using a trusted container registry.
- Check for rules.apiGroups, rules.resources, and rules.verbs combinations that might violate any of your access control policies.
- Avoid using hostPort and hostNetwork for any pod since this could limit the number of places the pod could run, since *hostIP.hostPort.protocol* must be unique.

Audit

The ability to see what resources are currently violating any given policy.

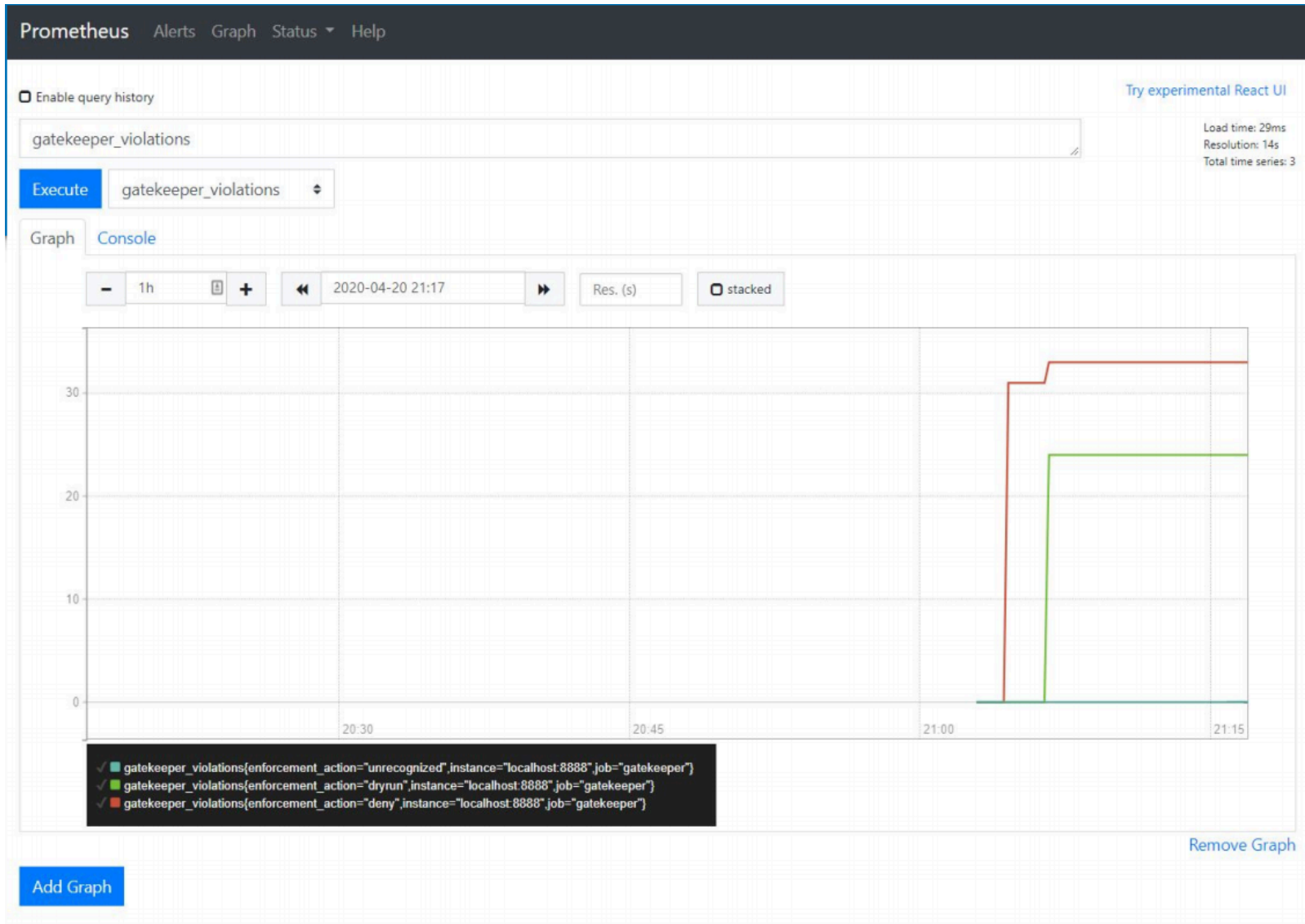
Status

```
status:
  auditTimestamp: "2020-10-26T05:58:00Z"
  byPod:
    - constraintUID: df1620fa-5ba6-4932-90e4-d9e81539fd6f
      enforced: true
      id: gatekeeper-audit-59d5559c8d-t97t4
      observedGeneration: 1
      operations:
        - audit
        - status
    - constraintUID: df1620fa-5ba6-4932-90e4-d9e81539fd6f
      enforced: true
      id: gatekeeper-controller-manager-7ffc874b4c-58v4p
      observedGeneration: 1
      operations:
        - webhook
    - constraintUID: df1620fa-5ba6-4932-90e4-d9e81539fd6f
      enforced: true
      id: gatekeeper-controller-manager-7ffc874b4c-d4db7
      observedGeneration: 1
      operations:
        - webhook
    - constraintUID: df1620fa-5ba6-4932-90e4-d9e81539fd6f
      enforced: true
      id: gatekeeper-controller-manager-7ffc874b4c-jhqts
      observedGeneration: 1
      operations:
        - webhook
  totalViolations: 4
  violations:
    - enforcementAction: deny
      kind: Deployment
      message: 'Deployment pods must contain labels: {"owner"}'
      name: demoservice
      namespace: default
    - enforcementAction: deny
      kind: Deployment
      message: 'Deployment pods must contain labels: {"owner"}'
      name: gatekeeper-audit
      namespace: gatekeeper-system
    - enforcementAction: deny
      kind: Deployment
      message: 'Deployment pods must contain labels: {"owner"}'
      name: gatekeeper-controller-manager
      namespace: gatekeeper-system
    - enforcementAction: deny
      kind: Deployment
      message: 'Deployment pods must contain labels: {"owner"}'
      name: coredns
      namespace: kube-system
```

Metrics

An HTTP endpoint used to collect performance metrics

Prometheus



Thanks

ahmed.badran@magalix.com

@a3badran