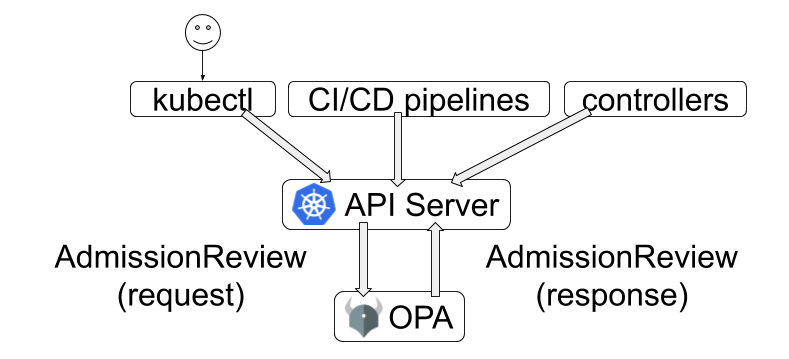
Using Open Policy Agent (OPA)

Using Open Policy Agent (OPA) and Gatekeeper for policy-based control in Amazon EKS

Security and governance is a critical component of configuring and managing fine-grained control for Kubernetes clusters and applications. Amazon EKS provides secure, managed Kubernetes clusters by default, but you still need to ensure that you configure and administer the applications appropriately that you run as part of the cluster.

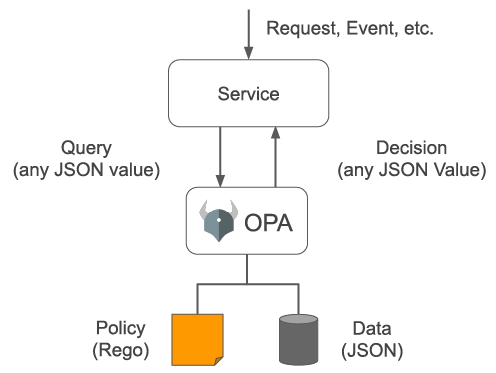
The Open Policy Agent (OPA, pronounced “oh-pa”) is an open source, general-purpose policy engine that unifies policy enforcement across the stack. OPA provides a high-level declarative language that lets you specify policy as code and simple APIs to offload policy decision-making from your software. You can use OPA to enforce policies in microservices, Kubernetes, CI/CD pipelines, API gateways, and more. OPA uses a policy language known as Rego which is a query language which was purpose built to support structured document models such as JSON. To learn more about Rego check out this [link](https://www.openpolicyagent.org/docs/latest/policy-language/).

OPA Gatekeeper is an open-source project that provides a first-class integration between OPA and Kubernetes. What Gatekeeper adds is an extensible parameterized policy library that includes native Kubernetes CRD's for instantiating and extending the OPA policy library. The Kubernetes API Server is configured to query OPA for admission control decisions when objects (e.g., Pods, Services, etc.) are created, updated, or deleted. The API Server sends the entire Kubernetes object in the webhook request to OPA. OPA evaluates the policies it has loaded using the admission review as input. Gatekeeper also provides audit functionality as well. The diagram below shows the flow between a user making a request to the Kube-API server and how AdmissionReview and AdmissionRequests are made through OPA Gatekeeper.



OPA decouples policy decision-making from policy enforcement. When your software needs to make policy decisions it queries OPA and supplies structured data (e.g., JSON) as input. OPA accepts arbitrary structured data as input.In the context of a development platform running on Amazon EKS, platform teams and administrators need a way of being able to set policies to adhere to governance and security requirements for all workloads and teams working on the same cluster. Examples of standard use cases for using policies via OPA Gatekeeper are listed below:

* Which users can access which resources.
* Which subnets egress traffic is allowed to.
* Which clusters a workload must be deployed to.
* Which registries binaries can be downloaded from.
* Which OS capabilities a container can execute with.
* Which times of day the system can be accessed at.



OPA generates policy decisions by evaluating the query input and against policies and data. OPA and Rego are domain-agnostic so you can describe almost any kind of invariant in your policies. Policy decisions are not limited to simple yes/no or allow/deny answers. Like query inputs, your policies can generate arbitrary structured data as output.

**Key Terminology**

* [**OPA Constraint Framework**](https://github.com/open-policy-agent/frameworks/tree/master/constraint) - Framework that enforces CRD-based policies and allow declaratively configured policies to be reliably shareable
* [**Constraint**](https://github.com/open-policy-agent/frameworks/tree/master/constraint#what-is-a-constraint) - A Constraint is a declaration that its author wants a system to meet a given set of requirements. Each Constraint is written with [Rego](https://www.openpolicyagent.org/docs/latest/policy-language/), a declarative query language used by OPA to enumerate instances of data that violate the expected state of the system. All Constraints are evaluated as a logical AND. If one Constraint is not satisfied, then the whole request is rejected.
* [**Enforcement Point**](https://github.com/open-policy-agent/frameworks/tree/master/constraint#what-is-an-enforcement-point) - Places where constraints can be enforced. Examples are Git hooks, Kubernetes admission controllers, and audit systems.
* [**Constraint Template**](https://github.com/open-policy-agent/frameworks/tree/master/constraint#what-is-a-constraint-template) - Templates that allows users to declare new constraints
* [**Target**](https://github.com/open-policy-agent/frameworks/tree/master/constraint#what-is-a-target) - Represents a coherent set of objects sharing a common identification and/or selection scheme, generic purpose, and can be analyzed in the same validation context

**AWS Blogs on OPA:**

* [Using Open Policy Agent on Amazon EKS](https://aws.amazon.com/blogs/opensource/using-open-policy-agent-on-amazon-eks)
* [Realize Policy-as-Code with AWS Cloud Development Kit through Open Policy Agent](https://aws.amazon.com/blogs/opensource/realize-policy-as-code-with-aws-cloud-development-kit-through-open-policy-agent/)
* [Policy-based countermeasures for Kubernetes – Part 1](https://aws.amazon.com/blogs/containers/policy-based-countermeasures-for-kubernetes-part-1/)
* [Policy-based countermeasures for Kubernetes – Part 2](https://aws.amazon.com/blogs/containers/policy-based-countermeasures-for-kubernetes-part-2/)

In this workshop, we will look at how to implement OPA on an Amazon EKS cluster as well as some sample scenarios for enforcing policies in the cluster using an OPA constraint framework.

OPA Gatekeeper setup in EKS

In this section, we will setup OPA Gatekeeper within the cluster.

1. Run the following commands to deploy OPA Gatekeeper using Prebuilt docker images

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

Check Output

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

namespace/gatekeeper-system created

resourcequota/gatekeeper-critical-pods created

customresourcedefinition.apiextensions.k8s.io/assign.mutations.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/assignimage.mutations.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/assignmetadata.mutations.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/configs.config.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/constraintpodstatuses.status.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/constrainttemplatepodstatuses.status.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/constrainttemplates.templates.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/expansiontemplate.expansion.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/expansiontemplatepodstatuses.status.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/modifyset.mutations.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/mutatorpodstatuses.status.gatekeeper.sh created

customresourcedefinition.apiextensions.k8s.io/providers.externaldata.gatekeeper.sh created

serviceaccount/gatekeeper-admin created

role.rbac.authorization.k8s.io/gatekeeper-manager-role created

clusterrole.rbac.authorization.k8s.io/gatekeeper-manager-role created

rolebinding.rbac.authorization.k8s.io/gatekeeper-manager-rolebinding created

clusterrolebinding.rbac.authorization.k8s.io/gatekeeper-manager-rolebinding created

secret/gatekeeper-webhook-server-cert created

service/gatekeeper-webhook-service created

deployment.apps/gatekeeper-audit created

deployment.apps/gatekeeper-controller-manager created

poddisruptionbudget.policy/gatekeeper-controller-manager created

mutatingwebhookconfiguration.admissionregistration.k8s.io/gatekeeper-mutating-webhook-configuration created

validatingwebhookconfiguration.admissionregistration.k8s.io/gatekeeper-validating-webhook-configuration created

1. To validate that OPA Gatekeeper is running within your cluster run the following command:

kubectl get pods -n gatekeeper-system

The output will look like below.

1

2

3

4

5

NAME READY STATUS RESTARTS AGE

gatekeeper-audit-6584df88df-nsf28 1/1 Running 1 (37s ago) 41s

gatekeeper-controller-manager-5ff69b954d-nvzsb 1/1 Running 0 41s

gatekeeper-controller-manager-5ff69b954d-pmm7v 1/1 Running 0 41s

gatekeeper-controller-manager-5ff69b954d-t59kk 1/1 Running 0 41s

If you notice the gatekeeper-audit-6584df88df-nsf28 pod is created when we deploy the OpaGatekeeperAddOn. The audit functionality enables periodic evaluations of replicated resources against the Constraints enforced in the cluster to detect pre-existing misconfigurations. Gatekeeper stores audit results as violations listed in the status field of the relevant Constraint. The gatekeeper-controller-manager is simply there to manage the OpaGatekeeperAddOn.

1. Once OPA Gatekeeper pods are in 'Running' state, monitor Audit controller and Controller manager component logs for webhook requests that are being issued by the Kubernetes API server.

Run the following command in **separate terminal window** to monitor Audit controller logs

kubectl logs -l control-plane=audit-controller -n gatekeeper-system -f

Check Output

1

2

3

4

5

6

7

8

9

10

{"level":"info","ts":1691054570.044209,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054570.0442343,"logger":"controller","msg":"resource","metaKind":"upgrade","kind":"AssignMetadata","group":"mutations.gatekeeper.sh","version":"v1alpha1"}

{"level":"info","ts":1691054570.0488415,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054570.0620747,"logger":"controller","msg":"resource","metaKind":"upgrade","kind":"Provider","group":"externaldata.gatekeeper.sh","version":"v1alpha1"}

{"level":"info","ts":1691054570.065626,"logger":"KubeAPIWarningLogger","msg":"externaldata.gatekeeper.sh/v1alpha1 is deprecated. Use externaldata.gatekeeper.sh/v1beta1 instead."}

{"level":"info","ts":1691054570.065955,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054572.327456,"logger":"readiness-tracker","msg":"readiness satisfied, no further collection"}

{"level":"info","ts":1691054629.8365571,"logger":"controller","msg":"auditing constraints and violations","process":"audit","audit\_id":"2023-08-03T09:23:49Z","event\_type":"audit\_started"}

{"level":"info","ts":1691054629.8529487,"logger":"controller","msg":"no constraint is found with apiversion","process":"audit","audit\_id":"2023-08-03T09:23:49Z","constraint apiversion":"constraints.gatekeeper.sh/v1beta1"}

{"level":"info","ts":1691054629.8529873,"logger":"controller","msg":"auditing is complete","process":"audit","audit\_id":"2023-08-03T09:23:49Z","event\_type":"audit\_finished"}

Run the following command in **separate terminal window** to monitor Controller manager logs

kubectl logs -l control-plane=controller-manager -n gatekeeper-system -f

Check Output

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

{"level":"info","ts":1691054569.154963,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054569.1549916,"logger":"controller","msg":"resource","metaKind":"upgrade","kind":"ModifySet","group":"mutations.gatekeeper.sh","version":"v1alpha1"}

{"level":"info","ts":1691054569.1592152,"msg":"Starting workers","controller":"externaldata-controller","worker count":1}

{"level":"info","ts":1691054569.160231,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054569.1602528,"logger":"controller","msg":"resource","metaKind":"upgrade","kind":"Assign","group":"mutations.gatekeeper.sh","version":"v1alpha1"}

{"level":"info","ts":1691054569.1637428,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054569.213409,"logger":"controller","msg":"resource","metaKind":"upgrade","kind":"Provider","group":"externaldata.gatekeeper.sh","version":"v1alpha1"}

{"level":"info","ts":1691054569.216204,"logger":"KubeAPIWarningLogger","msg":"externaldata.gatekeeper.sh/v1alpha1 is deprecated. Use externaldata.gatekeeper.sh/v1beta1 instead."}

{"level":"info","ts":1691054569.216296,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054570.6133084,"logger":"readiness-tracker","msg":"readiness satisfied, no further collection"}

{"level":"info","ts":1691054570.425286,"logger":"controller","msg":"resource","metaKind":"upgrade","kind":"ModifySet","group":"mutations.gatekeeper.sh","version":"v1alpha1"}

{"level":"info","ts":1691054570.4296567,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054570.4389153,"msg":"Starting workers","controller":"constraint-controller","worker count":1}

{"level":"info","ts":1691054570.438918,"msg":"Starting workers","controller":"externaldata-controller","worker count":1}

{"level":"info","ts":1691054570.4442906,"logger":"controller","msg":"resource","metaKind":"upgrade","kind":"Provider","group":"externaldata.gatekeeper.sh","version":"v1alpha1"}

{"level":"info","ts":1691054570.4474301,"logger":"KubeAPIWarningLogger","msg":"externaldata.gatekeeper.sh/v1alpha1 is deprecated. Use externaldata.gatekeeper.sh/v1beta1 instead."}

{"level":"info","ts":1691054570.447586,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054570.4498425,"msg":"Starting workers","controller":"modifyset-controller","worker count":1}

{"level":"info","ts":1691054570.4542532,"msg":"Starting workers","controller":"assignmetadata-controller","worker count":1}

{"level":"info","ts":1691054571.0831826,"logger":"readiness-tracker","msg":"readiness satisfied, no further collection"}

{"level":"info","ts":1691054569.9443588,"logger":"controller","msg":"resource","metaKind":"upgrade","kind":"AssignImage","group":"mutations.gatekeeper.sh","version":"v1alpha1"}

{"level":"info","ts":1691054569.947655,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054569.9517624,"msg":"Starting workers","controller":"constraint-controller","worker count":1}

{"level":"info","ts":1691054569.9617753,"msg":"Starting workers","controller":"assignmetadata-controller","worker count":1}

{"level":"info","ts":1691054569.9618478,"msg":"Starting workers","controller":"externaldata-controller","worker count":1}

{"level":"info","ts":1691054569.9641025,"msg":"Starting workers","controller":"modifyset-controller","worker count":1}

{"level":"info","ts":1691054569.9651625,"logger":"controller","msg":"resource","metaKind":"upgrade","kind":"Provider","group":"externaldata.gatekeeper.sh","version":"v1alpha1"}

{"level":"info","ts":1691054569.9685936,"logger":"KubeAPIWarningLogger","msg":"externaldata.gatekeeper.sh/v1alpha1 is deprecated. Use externaldata.gatekeeper.sh/v1beta1 instead."}

{"level":"info","ts":1691054569.9686935,"logger":"controller","msg":"resource count","metaKind":"upgrade","count":0}

{"level":"info","ts":1691054570.4463212,"logger":"readiness-tracker","msg":"readiness satisfied, no further collection"}

You can follow the OPA logs to see the webhook requests being issued by the Kubernetes API server.

This completes the OPA Gatekeeper setup on Amazon EKS cluster. In order to define and enforce the policy, OPA Gatekeeper uses a framework [OPA Constraint Framework](https://github.com/open-policy-agent/frameworks/tree/master/constraint)

# Use case #1: Restrict privileged containers in the cluster

In this section, we will define a new constraint template and constraint that will force the cluster to use unprivileged containers.

### [Build Constraint Templates](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/2-opa/2-build-policy-constraint-constraint-template" \l "build-constraint-templates)

ConstraintTemplate describes the Rego that enforces the constraint and the schema of the constraint. The schema constraint allows the author of the constraint (cluster admin) to define the constraint behavior.

In this scenario, the cluster administrator will force the cluster to use unprivileged containers. The OPA Gatekeeper will look for the securitycontext field and determine whether 'privileged=true' is present. If this is the case, the request will fail.

cd ~/environment

cat > constrainttemplate-1.yaml <<EOF

apiVersion: templates.gatekeeper.sh/v1beta1

kind: ConstraintTemplate

metadata:

name: k8spspprivilegedcontainer

spec:

crd:

spec:

names:

kind: K8sPSPPrivilegedContainer

targets:

- target: admission.k8s.gatekeeper.sh

rego: |

package k8spspprivileged

violation[{"msg": msg, "details": {}}] {

c := input\_containers[\_]

c.securityContext.privileged

msg := sprintf("Privileged container is not allowed: %v, securityContext: %v", [c.name, c.securityContext])

}

input\_containers[c] {

c := input.review.object.spec.containers[\_]

}

input\_containers[c] {

c := input.review.object.spec.initContainers[\_]

}

EOF

Create the ConstraintTemplate using the following command

kubectl create -f constrainttemplate-1.yaml

Check Output

1

constrainttemplate.templates.gatekeeper.sh/k8spspprivilegedcontainer created

Ensure that the CRD constrainttemplate is created.

kubectl get constrainttemplate

Check Output

1

2

NAME AGE

k8spspprivilegedcontainer 61s

### [Build Constraint](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/2-opa/2-build-policy-constraint-constraint-template" \l "build-constraint)

To enforce the policy, we will use the constraint below, which will ensure that all newly created pods are not privileged.

cd ~/environment

cat > constraint-1.yaml <<EOF

apiVersion: constraints.gatekeeper.sh/v1beta1

kind: K8sPSPPrivilegedContainer

metadata:

name: psp-privileged-container

spec:

match:

kinds:

- apiGroups: [""]

kinds: ["Pod"]

EOF

Create the Constraint using the following command

kubectl create -f constraint-1.yaml

Check Output

1

k8spspprivilegedcontainer.constraints.gatekeeper.sh/psp-privileged-container created

Ensure that the CRD for constraint is created.

kubectl get constraint

Check Output

1

2

NAME ENFORCEMENT-ACTION TOTAL-VIOLATIONS

psp-privileged-container

### [Test the policy](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/2-opa/2-build-policy-constraint-constraint-template" \l "test-the-policy)

In this section, we will test if the use of unprivileged containers is enforced in the cluster or not,

Let us deploy a privileged nginx pod:

cd ~/environment

cat > example-1.yaml <<EOF

apiVersion: v1

kind: Pod

metadata:

name: bad-nginx

labels:

app: bad-nginx

spec:

containers:

- name: nginx

image: nginx

securityContext:

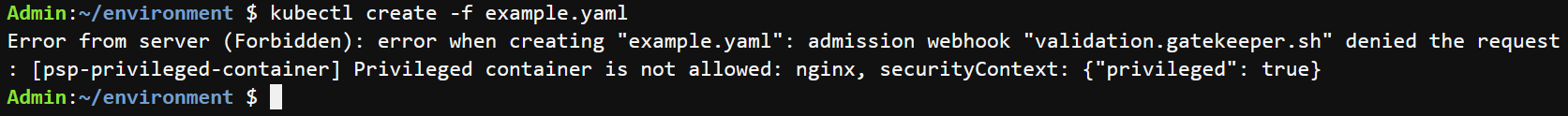
privileged: true

EOF

kubectl create -f example-1.yaml

You should now see an error message similar to below:

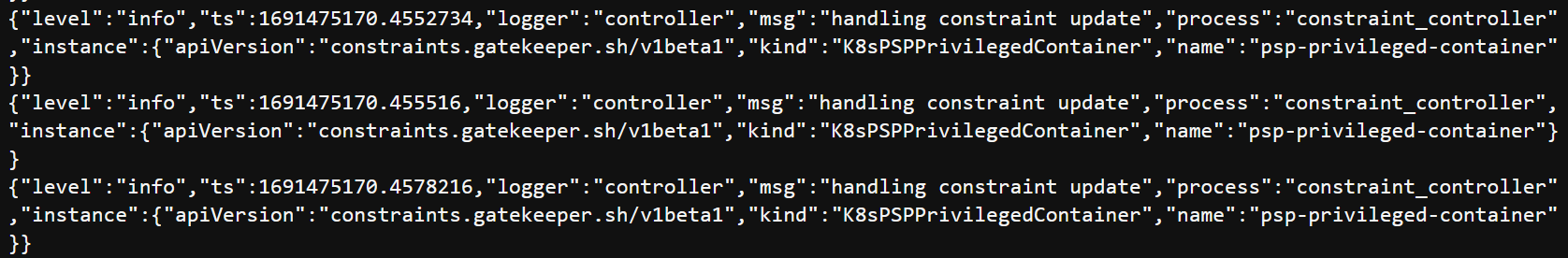
Check Output



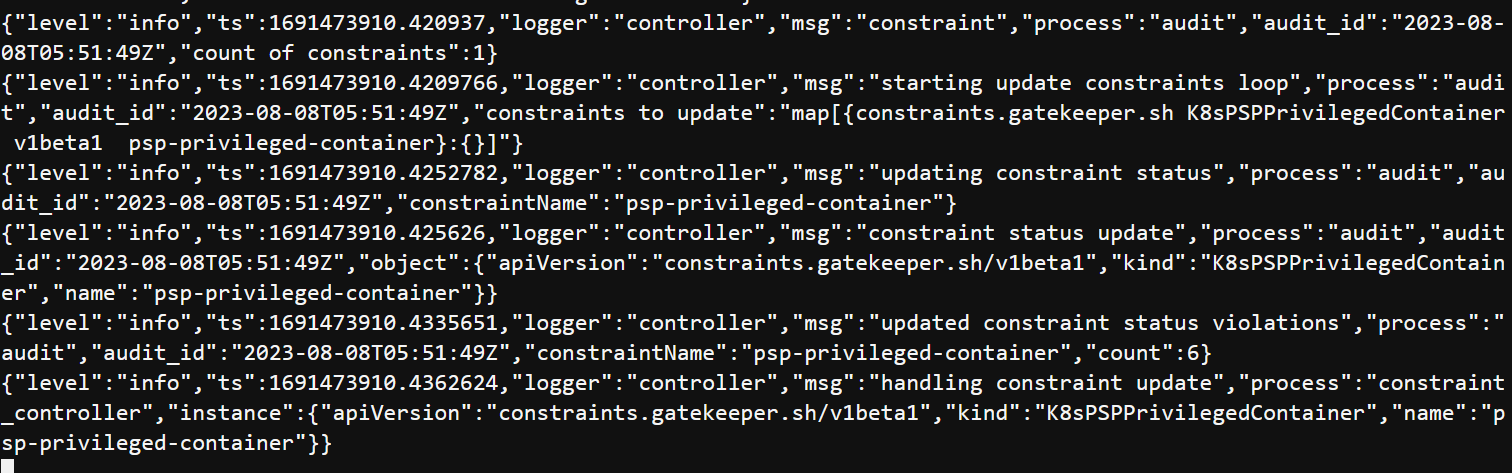
Error from server (Forbidden): error when creating "example-1.yaml": admission webhook "validation.gatekeeper.sh" denied the request: [psp-privileged-container] Privileged container is not allowed: nginx, securityContext: {"privileged": true}

Additionally, check the Controller manager logs to see the webhook requests sent by the Kubernetes API server for validation and mutation, as well as the Audit logs to check for policy compliance on objects that already exist in the cluster.

**Controller Manager Logs**



**Audit Controller Logs**



The request was denied by the Kubernetes API because it did not meet the requirement of unprivileged containers imposed by the OPA Gatekeeper constraint.

# Use case #2: Whitelist only known registry

This section will define a new constraint template and constraint that will verify that every pod's image comes from a known registry on a whitelist.

### [Build Constraint Templates](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/2-opa/3-build-policy-constraint-constraint-template-1" \l "build-constraint-templates)

In the example below, the cluster administrator will mandate that only known image repositories be used in the cluster.

cd ~/environment

cat > constrainttemplate-2.yaml <<EOF

apiVersion: templates.gatekeeper.sh/v1beta1

kind: ConstraintTemplate

metadata:

name: k8swhitelistedimages

spec:

crd:

spec:

names:

kind: k8sWhitelistedImages

validation:

# Schema for the parameters field

openAPIV3Schema:

properties:

images:

type: array

items: string

targets:

- target: admission.k8s.gatekeeper.sh

rego: |

package k8swhitelistedimages

whitelisted\_images = {images |

images = input.parameters.images[\_]

}

images\_whitelisted(str, patterns) {

image\_matches(str, patterns[\_])

}

image\_matches(str, pattern) {

contains(str, pattern)

}

violation[{"msg": msg}] {

input.review.object

image := input.review.object.spec.containers[\_].image

name := input.review.object.metadata.name

not images\_whitelisted(image, whitelisted\_images)

msg := sprintf("pod %q has invalid image %q. Please, contact your DevOps. Follow the whitelisted images %v", [name, image, whitelisted\_images])

}

EOF

Create the ConstraintTemplate using the following command

kubectl create -f constrainttemplate-2.yaml

Check Output

1

constrainttemplate.templates.gatekeeper.sh/k8swhitelistedimages created

Ensure that the CRD for constrainttemplate is created.

kubectl get constrainttemplate

Check Output

1

2

NAME AGE

k8swhitelistedimages 4m15s

### [Build Constraint](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/2-opa/3-build-policy-constraint-constraint-template-1" \l "build-constraint)

To enforce the policy, we will use the constraint below, which will ensure that all newly created pods image comes from a known registry on a whitelist

cd ~/environment

cat > constraint-2.yaml <<EOF

apiVersion: constraints.gatekeeper.sh/v1beta1

kind: k8sWhitelistedImages

metadata:

name: k8senforcewhitelistedimages

spec:

match:

kinds:

- apiGroups: [""]

kinds: ["Pod"]

parameters:

images:

# AWS Internal ECR registries

- 999999999999.dkr.ecr.us-east-1.amazonaws.com/

# AWS Public Registries

- 888888888888.dkr.ecr.us-west-2.amazonaws.com/

- 888888888888.dkr.ecr.us-east-1.amazonaws.com/

# Images used by the infrastructure services inside the kubernetes cluster

- amazon/aws-node-termination-handler

- amazon/aws-alb-ingress-controller

- amazon/aws-efs-csi-driver

- amazon/cloudwatch-agent

- docker.io/amazon/aws-alb-ingress-controller

- nvidia/k8s-device-plugin

- k8s.gcr.io/autoscaling/cluster-autoscaler

- k8s.gcr.io/metrics-server-amd64

- quay.io/coreos/kube-state-metrics

- quay.io/kubernetes-ingress-controller/nginx-ingress-controller

- kubernetesui/dashboard

- kubernetesui/metrics-scraper

- jtblin/kube2iam

- grafana/grafana

- prom/alertmanager

- prom/prometheus

- openpolicyagent/gatekeeper

# Images for support

- amazon/aws-cli

- radial/busyboxplus

- docker.io/radial/busyboxplus

- busybox

EOF

Create the Constraint using the following command

kubectl create -f constraint-2.yaml

Check Output

1

k8swhitelistedimages.constraints.gatekeeper.sh/k8senforcewhitelistedimages created

Ensure that the CRD for constraint is created.

kubectl get constraint

Check Output

1

2

NAME ENFORCEMENT-ACTION TOTAL-VIOLATIONS

k8swhitelistedimages.constraints.gatekeeper.sh/k8senforcewhitelistedimages

### [Test the policy](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/2-opa/3-build-policy-constraint-constraint-template-1" \l "test-the-policy)

Let’s deploy a nginx pod from unknown registry.

cd ~/environment

cat > example-2.yaml <<EOF

apiVersion: v1

kind: Pod

metadata:

name: bad-nginx

labels:

app: bad-nginx

spec:

containers:

- name: nginx

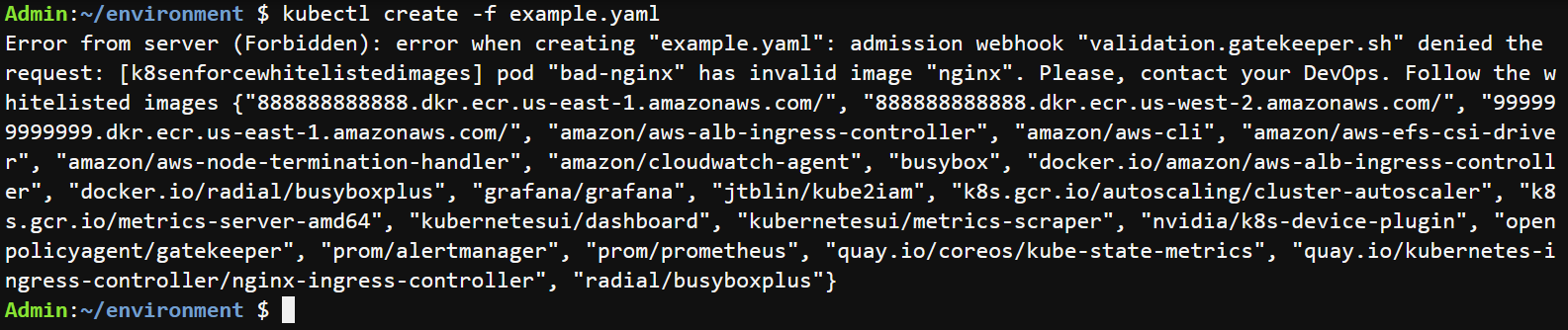
image: nginx

EOF

kubectl create -f example-2.yaml

You should now see an error message similar to below:

Check Output

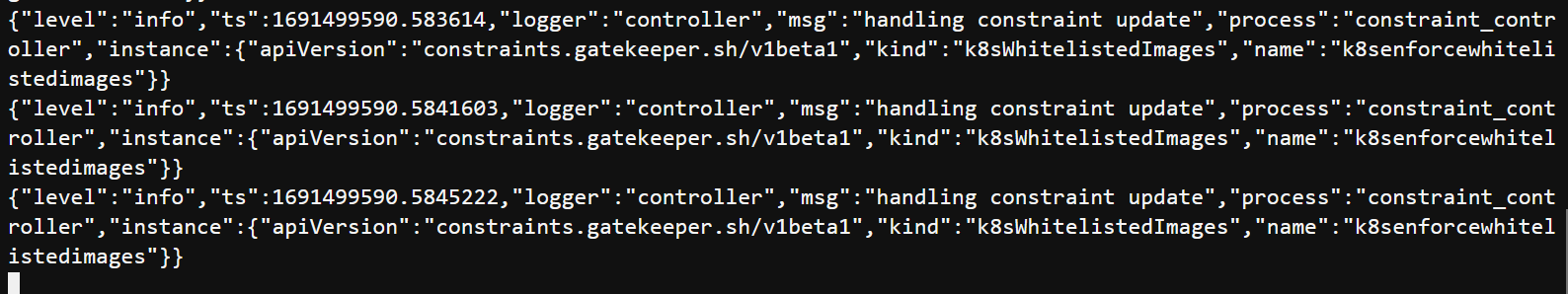


Error from server (Forbidden): error when creating "example-2.yaml": admission webhook "validation.gatekeeper.sh" denied the request: [k8senforcewhitelistedimages] pod "bad-nginx" has invalid image "nginx". Please, contact your DevOps. Follow the whitelisted images {"888888888888.dkr.ecr.us-east-1.amazonaws.com/", "888888888888.dkr.ecr.us-west-2.amazonaws.com/", "999999999999.dkr.ecr.us-east-1.amazonaws.com/", "amazon/aws-alb-ingress-controller", "amazon/aws-cli", "amazon/aws-efs-csi-driver", "amazon/aws-node-termination-handler", "amazon/cloudwatch-agent", "busybox", "docker.io/amazon/aws-alb-ingress-controller", "docker.io/radial/busyboxplus", "grafana/grafana", "jtblin/kube2iam", "k8s.gcr.io/autoscaling/cluster-autoscaler", "k8s.gcr.io/metrics-server-amd64", "kubernetesui/dashboard", "kubernetesui/metrics-scraper", "nvidia/k8s-device-plugin", "openpolicyagent/gatekeeper", "prom/alertmanager", "prom/prometheus", "quay.io/coreos/kube-state-metrics", "quay.io/kubernetes-ingress-controller/nginx-ingress-controller", "radial/busyboxplus"}

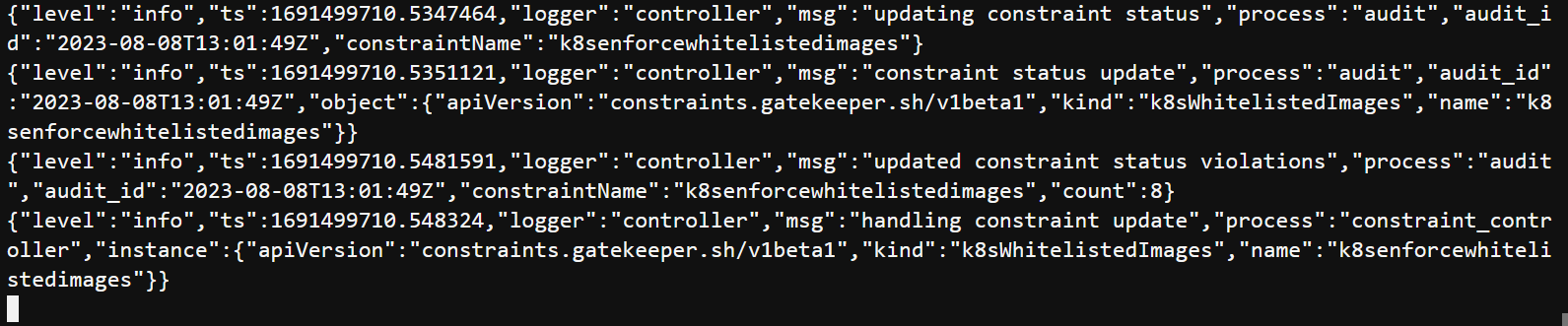
Additionally, check the Controller manager logs to see the webhook requests sent by the Kubernetes API server for validation and mutation, as well as the Audit logs to check for policy compliance on objects that already exist in the cluster.

Check Output

**Controller Manager Logs**



**Audit Controller Logs**



The request was denied by Kubernetes API, because it didn’t meet the requirement of known registries on whitelist imposed by OPA Gatekeeper constraint.

# Use case #3: Enforce labels for objects

In this section, we will define a new constraint template as well as a constraint that enforces the inclusion of labels for namespaces and pods.

### [Build Constraint Templates](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/2-opa/4-build-policy-constraint-constraint-template-2" \l "build-constraint-templates)

The template below defines a general constraint that checks for the presence of labels. Once created, the template can be used to create constraints that require the definition of a specific label or set of labels on an object.

cd ~/environment

cat > constrainttemplate-3.yaml <<EOF

apiVersion: templates.gatekeeper.sh/v1

kind: ConstraintTemplate

metadata:

name: k8srequiredlabels

annotations:

metadata.gatekeeper.sh/title: "Required Labels"

metadata.gatekeeper.sh/version: 1.0.0

description: >-

Requires resources to contain specified labels, with values matching

provided regular expressions.

spec:

crd:

spec:

names:

kind: K8sRequiredLabels

validation:

openAPIV3Schema:

type: object

properties:

message:

type: string

labels:

type: array

description: >-

A list of labels and values the object must specify.

items:

type: object

properties:

key:

type: string

description: >-

The required label.

allowedRegex:

type: string

description: >-

If specified, a regular expression the annotation's value

must match. The value must contain at least one match for

the regular expression.

targets:

- target: admission.k8s.gatekeeper.sh

rego: |

package k8srequiredlabels

get\_message(parameters, \_default) = msg {

not parameters.message

msg := \_default

}

get\_message(parameters, \_default) = msg {

msg := parameters.message

}

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)

}

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)

}

EOF

Create the ConstraintTemplate using the following command

kubectl create -f constrainttemplate-3.yaml

Check Output

1

constrainttemplate.templates.gatekeeper.sh/k8srequiredlabels created

Ensure that the CRD constrainttemplate is created.

kubectl get constrainttemplate

Check Output

1

2

NAME AGE

k8srequiredlabels 2m18s

### [Build Constraint](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/2-opa/4-build-policy-constraint-constraint-template-2" \l "build-constraint)

Below example contraint defines that any namespace objects that are created must have a value set for the owner label.

cd ~/environment

cat > constraint-3.yaml <<EOF

apiVersion: constraints.gatekeeper.sh/v1beta1

kind: K8sRequiredLabels

metadata:

name: all-ns-must-have-owner-label

spec:

match:

kinds:

- apiGroups: [""]

kinds: ["Namespace"]

parameters:

message: "All namespaces must have an owner label"

labels:

- key: owner

EOF

Create the Constraint using the following command

kubectl create -f constraint-3.yaml

Check Output

1

k8srequiredlabels.constraints.gatekeeper.sh/all-ns-must-have-owner-label created

Ensure that the CRD for constraint is created.

kubectl get constraint

Check Output

1

2

NAME ENFORCEMENT-ACTION TOTAL-VIOLATIONS

k8srequiredlabels.constraints.gatekeeper.sh/all-ns-must-have-owner-label

### [Test the policy](https://catalog.us-east-1.prod.workshops.aws/workshops/165b0729-2791-4452-8920-53b734419050/en-US/3-pod-security/2-opa/4-build-policy-constraint-constraint-template-2" \l "test-the-policy)

Let us create namespace without owner label.

cd ~/environment

cat > example-3.yaml <<EOF

apiVersion: v1

kind: Namespace

metadata:

name: test-opa

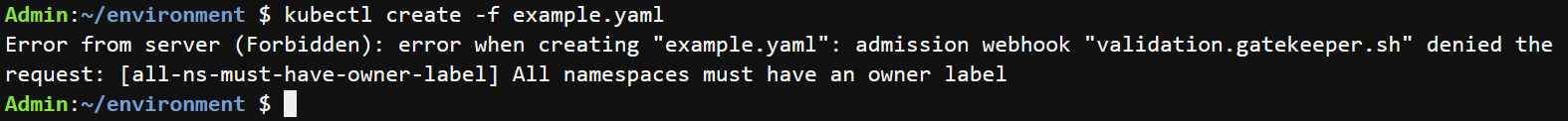
spec: {}

EOF

kubectl create -f example-3.yaml

You should now see an error message similar to below:

Check Output

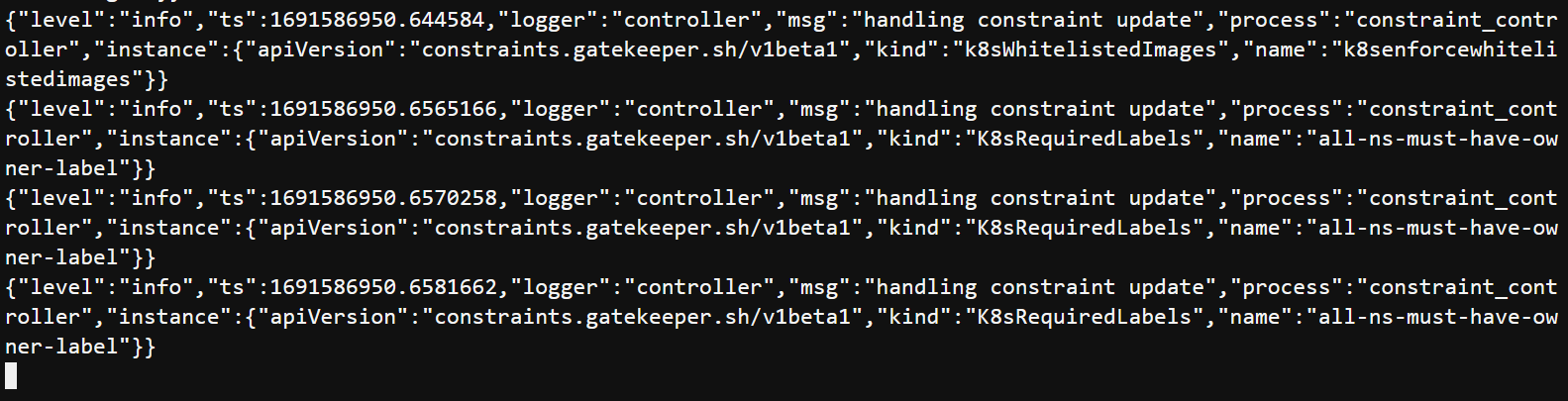


Error from server (Forbidden): error when creating "example-3.yaml": admission webhook "validation.gatekeeper.sh" denied the request: [all-ns-must-have-owner-label] All namespaces must have an owner label

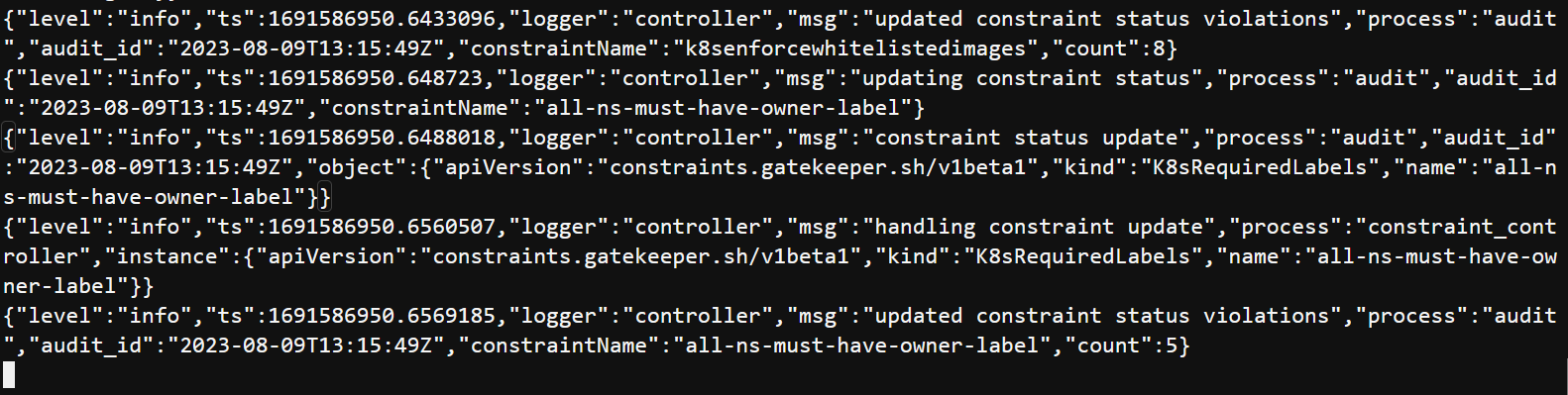
Additionally, check the Controller manager logs to see the webhook requests sent by the Kubernetes API server for validation and mutation, as well as the Audit logs to check for policy compliance on objects that already exist in the cluster.

Check Output

**Controller Manager Logs**



**Audit Controller Logs**



The request was denied by the Kubernetes API because it did not comply with the constraint imposed by OPA Gatekeeper that all namespace objects created must have a value set for the owner label.

**Summary**

Congratulations !!! We learnt how to leverage OPA Gatekeeper to implement fine-grained policies in Kubernetes clusters, enhancing overall security while also simplifying compliance and audit requirements.

# Clean up

Run the following command to clean up the resources.

kubectl delete -f constrainttemplate-1.yaml

kubectl delete -f constrainttemplate-2.yaml

kubectl delete -f constrainttemplate-3.yaml

kubectl delete -f constraint-1.yaml

kubectl delete -f constraint-2.yaml

kubectl delete -f constraint-3.yaml

kubectl delete -f example-1.yaml

kubectl delete -f example-2.yaml

kubectl delete -f example-3.yaml

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