

[Open in app](#)**Jonathan**[Follow](#)

29 Followers

[About](#)

# Certified Kubernetes Security Specialist (CKS) Preparation Part 6 — Open Policy Agent (OPA)

**Jonathan** Mar 10 · 4 min read

If you have not yet checked the previous parts of this series, please go ahead and check [Part1](#), [Part2](#), [Part3](#), [Part4](#) and [Part5](#).

In this article, I would focus on the preparation around **open policy agent (OPA)** in CKS certification exam. The whole article would be split into 3 parts, installation, testing and customization.

*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.*

— Quoted from [Open Policy Agent | Documentation](#)

In my own definition, OPA is the yet another custom resource deployed in K8s environment to ensure the whole deployment and operation process are under strict security principles. OPA is written in Rego, a query language that is easy to read and write. For more information around Rego, please check out its official documentation [here](#).

[Open in app](#)


This article walks you through OPA installation, testing and customization. The installation is as simple as executing the one-liner,

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

After installation completes, we could verify by looking into the specific namespace.

- `kubectl get ns`
- `kubectl get all -n gatekeeper-system`

```
jonw@CKS-Master:~$ kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/master/deploy/gatekeeper.yaml
namespace/gatekeeper-system created
Warning: apiextensions.k8s.io/v1beta1 CustomResourceDefinition is deprecated in v1.16+, unavailable in v1.22+; use apiextensions.k8s.io/v1 CustomResourceDefinition
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
serviceaccount/gatekeeper-admin created
podsecuritypolicy.policy/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
Warning: admissionregistration.k8s.io/v1beta1 ValidatingWebhookConfiguration is deprecated in v1.16+, unavailable in v1.22+; use admissionregistration.k8s.io/v1 ValidatingWebhookConfiguration
validatingwebhookconfiguration.admissionregistration.k8s.io/gatekeeper-validating-webhook-configuration created
jonw@CKS-Master:~$ kubectl get ns
NAME                STATUS   AGE
allowingress        Active   21d
default              Active   22d
gatekeeper-system    Active   6m19s
ingress-nginx        Active   18d
kube-node-lease      Active   22d
kube-public          Active   22d
kube-system          Active   22d
jonw@CKS-Master:~$ kubectl get all -n gatekeeper-system
NAME                                                    READY   STATUS    RESTARTS   AGE
pod/gatekeeper-audit-66764cbbb5-7zfrj                 1/1     Running   0           6m22s
pod/gatekeeper-controller-manager-c9fdc8c4-75fgr       1/1     Running   0           6m22s
pod/gatekeeper-controller-manager-c9fdc8c4-9mzmt       1/1     Running   0           6m22s
pod/gatekeeper-controller-manager-c9fdc8c4-q3wqj       1/1     Running   0           6m22s

NAME                                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/gatekeeper-webhook-service  ClusterIP   10.103.143.117  <none>        443/TCP    6m22s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/gatekeeper-audit    1/1     1             1           6m22s
deployment.apps/gatekeeper-controller-manager  3/3     3             3           6m22s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/gatekeeper-audit-66764cbbb5  1         1         1       6m22s
replicaset.apps/gatekeeper-controller-manager-c9fdc8c4  3         3         3       6m22s
```

Also, we could check on the custom resource definitions (CRD) being created.

```
jonw@CKS-Master:~$ kubectl get crd
NAME                                                    CREATED AT
configs.config.gatekeeper.sh                          2021-03-08T21:23:15Z
constraintpodstatuses.status.gatekeeper.sh             2021-03-08T21:23:15Z
constrainttemplatepodstatuses.status.gatekeeper.sh     2021-03-08T21:23:15Z
constrainttemplates.templates.gatekeeper.sh            2021-03-08T21:23:15Z
```

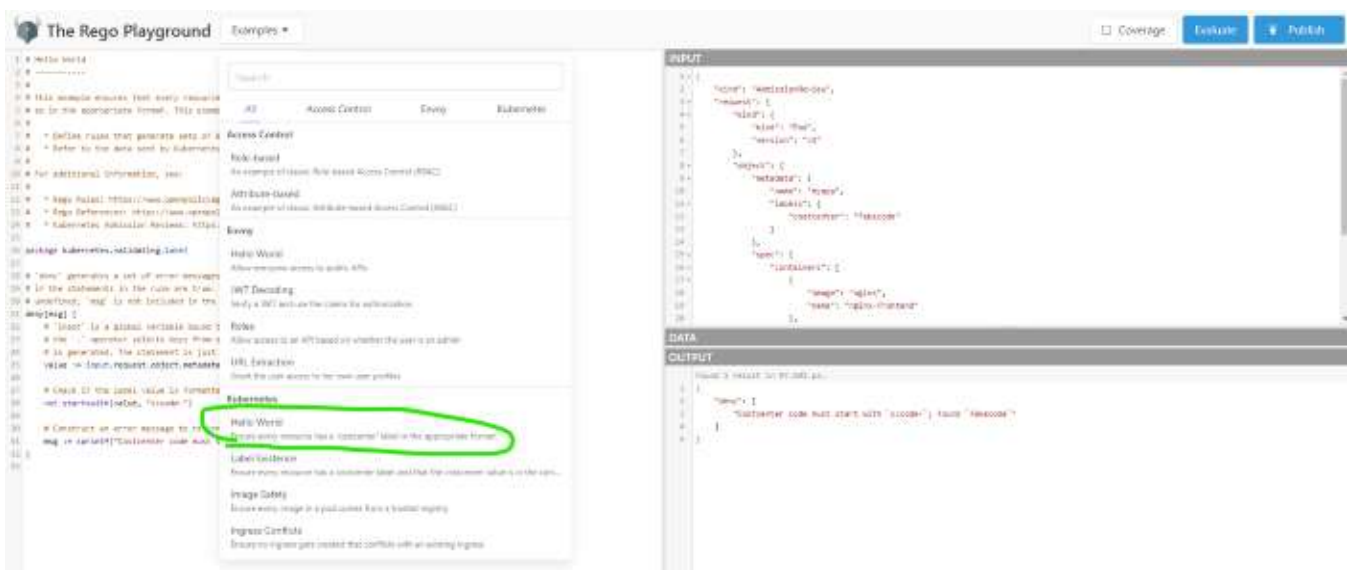
[Open in app](#)


the specific details. For example, constraint template defines every resource needs a label and constraint defines resource “namespace” needs to have label “OPA”.

## Testing

We could either directly go through our running environment to see how OPA works in practice or we could head over to [the Rego playground](#) to verify some of the OPA constraint logics and testify whether those fit our needs.

On the top left, there is a drop-down menu that gives you some pre-defined constraint templates when you are using OPA in different environment. In this case, we choose Kubernetes’ constraint template example, “Hello World”. The constraint template is not “Hello World” but the fact that some resources within K8s require labels.



In the constraint itself (after clicking, the content would show on the left of the web browser), you could see the incoming request would be denied if the label of “costcenter” does not start with “cccode-”.

```
deny[msg] {
  # `input` is a global variable bound to the data sent to OPA by
  # Kubernetes. In Rego,
  # the `.` operator selects keys from objects. If a key is missing,
  # no error
```

[Open in app](#)

```
# Check if the label value is formatted correctly.
not startswith(value, "cccode-")

# Construct an error message to return to the user.
msg := sprintf("Costcenter code must start with `cccode-`; found
`%v`, [value])
}
```

The result of not having “cccode-” in the beginning of label “costcenter” would result in the following deny message.

```
{
  "deny": [
    "Costcenter code must start with `cccode-`; found
`fakecode`"
  ]
}
```

Pretty straightforward, right? Let’s now take a look how it works in a real cluster. First things first, execute the one-liner to get constraint template installed.

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/master/demo/basic/templates/k8srequiredlabels\_template.yaml
```

Ensure the constraint template is installed by executing

```
kubectl get crd
```

```
jonw@CKS-Master:~$ kubectl get crd
NAME                                     CREATED AT
configs.config.gatekeeper.sh           2021-03-08T21:23:15Z
constraintpodstatuses.status.gatekeeper.sh 2021-03-08T21:23:15Z
constrainttemplatepodstatuses.status.gatekeeper.sh 2021-03-08T21:23:15Z
constrainttemplates.templates.gatekeeper.sh 2021-03-08T21:23:15Z
k8srequiredlabels.constraints.gatekeeper.sh 2021-03-08T21:57:59Z
```

[Open in app](#)

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/master/demo/basic/constraints/all_ns_must_have_gatekeeper.yaml
```

Ensure the constraint is installed by executing

```
kubectl get k8srequiredlabels
```

```
jonw@CKS-Master:~$ kubectl get k8srequiredlabels
NAME                AGE
ns-must-have-gk     29s
```

Lastly, try to create a namespace without label “gatekeeper” and see what it would respond.

```
kubectl create ns test
```

```
jonw@CKS-Master:~$ kubectl create ns test
Error from server ([denied by ns-must-have-gk] you must provide labels: {"gatekeeper": ""}): admission webhook "validation.gatekeeper.sh" denied the request: [denied by ns-must-have-gk] you must provide labels: {"gatekeeper": ""}]
```

## Customization

Now, I believe all of you that have understood what is going on in the first 2 parts would be ready for some customizations in OPA, meaning you could create your own constraint templates and constraints. As this is thoroughly covered by Mohamed’s article, [Create Your Own Constraint Step by Step](#), I would just briefly go through the key points.

- Ensure the JSON object is in the correct format for both constraint templates and constraints

[Open in app](#)

Deploy in the actual running environment

That is it! Now all of you have some knowledge around what is OPA and how we could use it in the real-life situation. Happy learning!

[Kubernetes Security](#)[Preparation](#)[Open Policy Agent](#)[Rego](#)[Constraints](#)[About](#) [Help](#) [Legal](#)

Get the Medium app

