# OPA Gatekeeper: ConstraintTemplate vs Constraint Explained

In OPA Gatekeeper, Kubernetes policy enforcement is implemented using two main building blocks: ConstraintTemplates and Constraints. These allow for the definition and enforcement of custom policies across Kubernetes resources.

## 1. ConstraintTemplate

A ConstraintTemplate defines the policy logic using Rego, the policy language of Open Policy Agent (OPA). It allows the creation of reusable constraints that define how Kubernetes resources should behave.

Example:

apiVersion: templates.gatekeeper.sh/v1beta1  
kind: ConstraintTemplate  
metadata:  
 name: k8srequiredresources  
spec:  
 crd:  
 spec:  
 names:  
 kind: K8sRequiredResources  
 targets:  
 - target: admission.k8s.gatekeeper.sh  
 rego: |  
 package k8srequiredresources  
  
 violation[{"msg": msg}] {  
 container := input.review.object.spec.containers[\_]  
 not container.resources.requests.cpu  
 msg := sprintf("Container %v is missing CPU requests", [container.name])  
 }

## 2. Constraint

A Constraint is a custom resource that references a ConstraintTemplate. It defines which Kubernetes resources the policy should be enforced on, and can include specific parameters.

Example:

apiVersion: constraints.gatekeeper.sh/v1beta1  
kind: K8sRequiredResources  
metadata:  
 name: enforce-resource-limits  
spec:  
 match:  
 kinds:  
 - apiGroups: [""]  
 kinds: ["Pod"]

## Why Both ConstraintTemplate and Constraint Are Needed

The use of both ConstraintTemplates and Constraints provides clear separation between policy logic (what to check) and policy application (where to check). This enhances modularity and reusability.  
- ConstraintTemplate = Defines the logic (what to check)  
- Constraint = Applies the logic (where to check)