BESTCLOUDFOR.ME

Tech Talks

RBAC & OPA/Gatekeeper

30.05.2023

General Information

This document includes a detailed procedure of RBAC & OPA/Gatekeeper

What is RBAC?

Role-Based Access Control (RBAC) is a security framework that defines access permissions based on roles. Roles are assigned to users based on their job functions or responsibilities. This allows organizations to control who has access to what resources, and to ensure that only authorized users can access sensitive data.

RBAC is a powerful tool for managing access control. It is easy to implement and manage, and it can be used to control access to a wide variety of resources, including files, folders, databases, and applications. RBAC is also scalable, so it can be used to manage access in large organizations with complex security requirements. Here are some of the benefits of using RBAC:

- It simplifies access management by centralizing the management of permissions.
- It reduces the risk of unauthorized access by ensuring that only authorized users can access resources. It makes it easier to track and audit access to resources.
- It can be used to enforce security policies.

RBAC DEMO

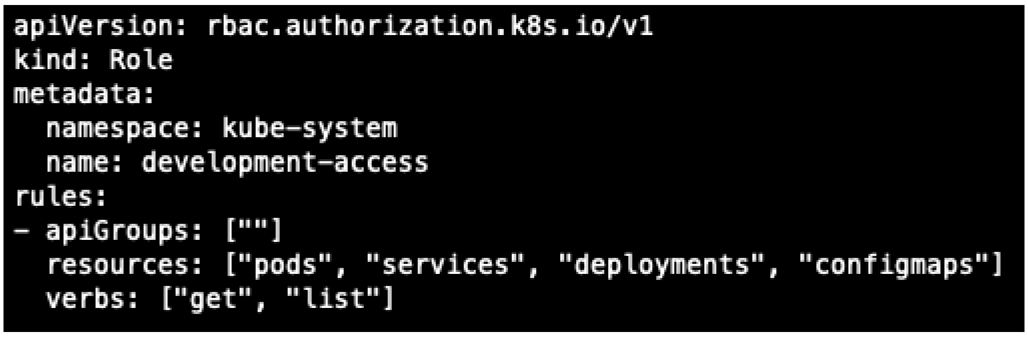
In this demo we have created “Role” and “RoleBinding” in the EKS cluster and updated the “aws-auth” ConfigMap.

The goal is to restrict kubernetes privileges of the IAM user who is attached to “aws-auth” ConfigMap.

Assuming the “developer” group is able to access only “kube-system” namespace resources with read-only privileges.

1) Create “Role” object via manifest:

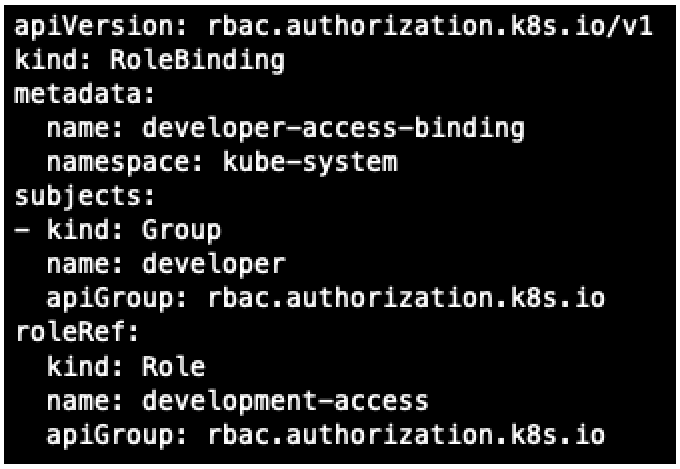
$ vi developer-role.yaml



```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: kube-system
  name: development-access
rules:
- apiGroups: [""]
  resources: ["pods", "services", "deployments", "configmaps"]
  verbs: ["get", "list"]
```

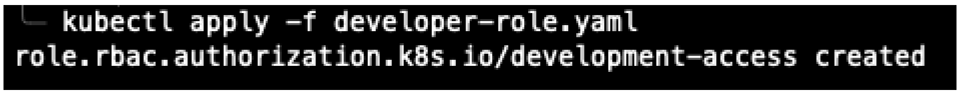
2) Create “RoleBinding” object via manifest:

$ vi developer-role-binding.yaml

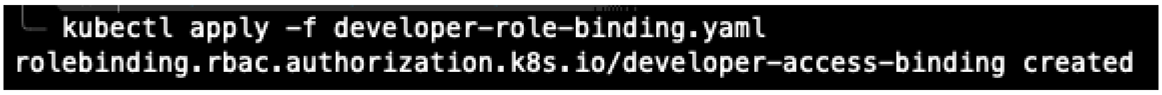


```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: developer-access-binding
  namespace: kube-system
subjects:
- kind: Group
  name: developer
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: development-access
  apiGroup: rbac.authorization.k8s.io
```

3) Create the objects via “Apply” commands:

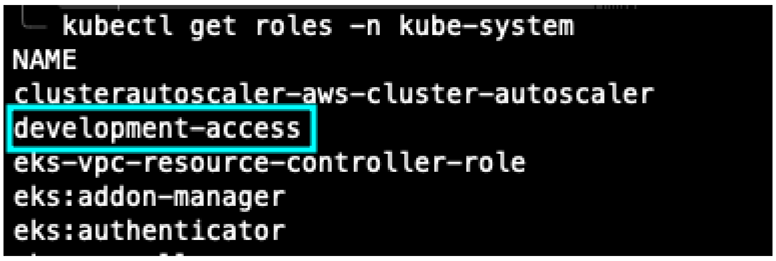


```
└─ kubectl apply -f developer-role.yaml
role.rbac.authorization.k8s.io/development-access created
```

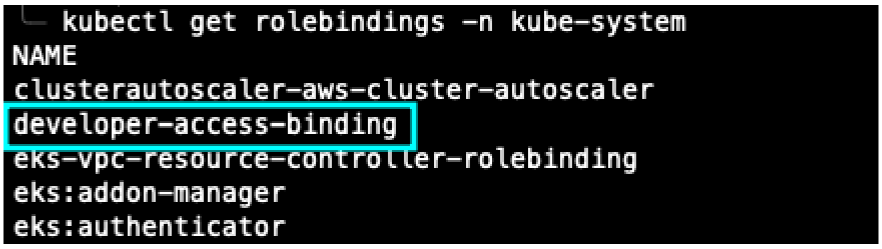


```
└─ kubectl apply -f developer-role-binding.yaml
rolebinding.rbac.authorization.k8s.io/developer-access-binding created
```

4) Assure that objects are created:

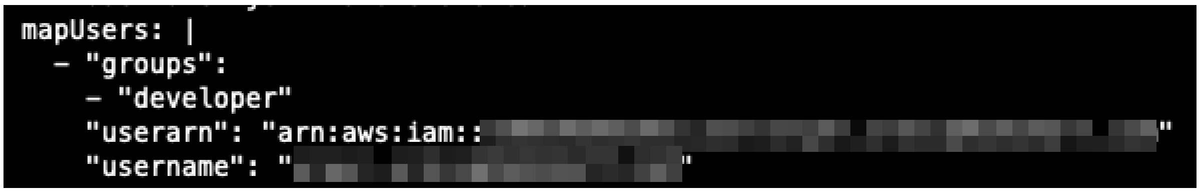


```
└─ kubectl get roles -n kube-system
NAME
clusterautoscaler-aws-cluster-autoscaler
development-access
eks-vpc-resource-controller-role
eks:addon-manager
eks:authenticator
```



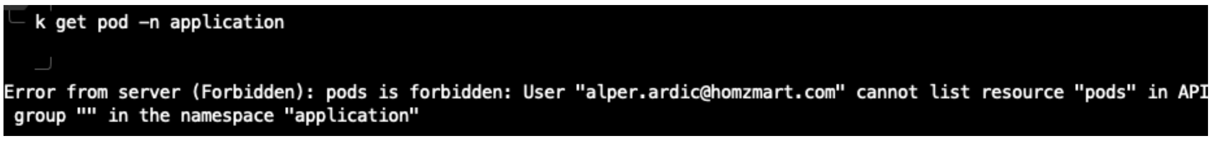
```
└─ kubectl get rolebindings -n kube-system
NAME
clusterautoscaler-aws-cluster-autoscaler
developer-access-binding
eks-vpc-resource-controller-rolebinding
eks:addon-manager
eks:authenticator
```

5) Update the “aws-auth” ConfigMap as “developer” group:

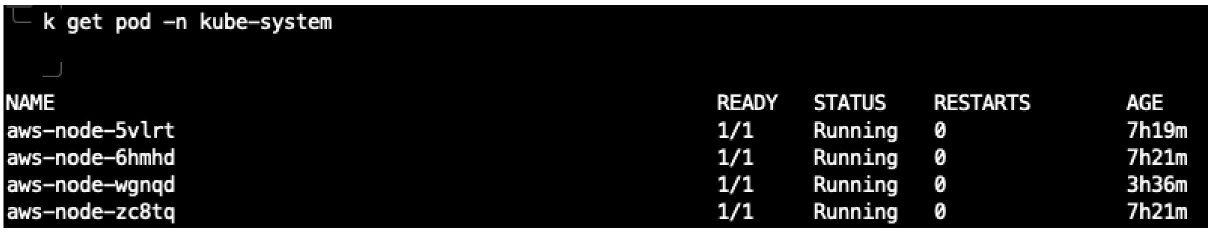


```
mapUsers: |
  - "groups":
    - "developer"
    "userarn": "arn:aws:iam::[redacted]"
    "username": "[redacted]"
```

6) Assuring that roles are working as expected:



```
└ k get pod -n application
  ┘
Error from server (Forbidden): pods is forbidden: User "alper.ardic@homzmart.com" cannot list resource "pods" in API
 group "" in the namespace "application"
```



```
└ k get pod -n kube-system
  ┘
NAME                                                    READY   STATUS    RESTARTS       AGE
aws-node-5vlrt                                          1/1     Running   0              7h19m
aws-node-6hmhd                                          1/1     Running   0              7h21m
aws-node-wgnqd                                          1/1     Running   0              3h36m
aws-node-zc8tq                                          1/1     Running   0              7h21m
```

Summary

By using Role and RoleBinding, we can separate individual IAM users who access the AWS EKS cluster to specific namespaces.

What is OPA/Gatekeeper?

OPA/Gatekeeper is a cloud-native policy controller that aids in enforcing policies and enhancing governance in Kubernetes environments. It serves as a customizable admission controller for Kubernetes, ensuring compliance with policies executed by the Open Policy Agent (OPA), a policy engine for Cloud Native environments hosted by CNCF.

OPA/Gatekeeper can be employed to enforce a wide range of policies, such as:

- Resource quotas
- Pod security policies
- Network policies
- RBAC policies

OPA/Gatekeeper is a potent tool that assists in securing your Kubernetes environment and preventing unauthorized access to your resources. Here are some of the advantages of using OPA/Gatekeeper:

- It is user-friendly and easy to configure.
- It is scalable and can effectively manage large Kubernetes environments.
- It is extensible, allowing enforcement of various policies.
- It is an open-source solution available free of charge.

OPA/Gatekeeper DEMO

In this demo we will restrict the provisioning of Pods which don't include Resources and we set request and limit for both CPU and memory utilization. The purpose of this demo is to avoid unnecessary loading of EKS cluster nodes.

1) We create templates for request and limits seperately;

limits template:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8scontainerlimits
  annotations:
    metadata.gatekeeper.sh/title: "Container Limits"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Requires containers to have memory and CPU limits set and
constrains
      limits to be within the specified maximum values.


https://kubernetes.io/docs/concepts/configuration/manage-resources-conta
iners/
spec:
  crd:
    spec:
      names:
        kind: K8sContainerLimits
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            exemptImages:
              description: >-
                Any container that uses an image that matches an entry
in this list will be excluded
                from enforcement. Prefix-matching can be signified with
`*`. For example: `my-image-*`.

                It is recommended that users use the fully-qualified
Docker image name (e.g. start with a domain name)
                in order to avoid unexpectedly exempting images from an
untrusted repository.
```

```
              type: array
              items:
                type: string
            cpu:
              description: "The maximum allowed cpu limit on a Pod,
exclusive."
              type: string
            memory:
              description: "The maximum allowed memory limit on a Pod,
exclusive."
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8scontainerlimits

        import data.lib.exempt_container.is_exempt

        missing(obj, field) = true {
          not obj[field]
        }

        missing(obj, field) = true {
          obj[field] == ""
        }

        canonify_cpu(orig) = new {
          is_number(orig)
          new := orig * 1000
        }

        canonify_cpu(orig) = new {
          not is_number(orig)
          endswith(orig, "m")
          new := to_number(replace(orig, "m", ""))
        }

        canonify_cpu(orig) = new {
          not is_number(orig)
          not endswith(orig, "m")
          re_match("^[0-9]+(\\.[0-9]+)?$", orig)
          new := to_number(orig) * 1000
        }

        # 10 ** 21
        mem_multiple("E") = 1000000000000000000000 { true }
        # 10 ** 18
        mem_multiple("P") = 1000000000000000000 { true }

        # 10 ** 15
        mem_multiple("T") = 1000000000000000 { true }

        # 10 ** 12
        mem_multiple("G") = 1000000000000 { true }

        # 10 ** 9
        mem_multiple("M") = 1000000000 { true }

        # 10 ** 6
        mem_multiple("k") = 1000000 { true }

        # 10 ** 3
        mem_multiple("") = 1000 { true }

        # Kubernetes accepts millibyte precision when it probably
shouldn't.
        # https://github.com/kubernetes/kubernetes/issues/28741
        # 10 ** 0
        mem_multiple("m") = 1 { true }

        # 1000 * 2 ** 10
        mem_multiple("Ki") = 1024000 { true }

        # 1000 * 2 ** 20
        mem_multiple("Mi") = 1048576000 { true }

        # 1000 * 2 ** 30
        mem_multiple("Gi") = 1073741824000 { true }

        # 1000 * 2 ** 40
        mem_multiple("Ti") = 1099511627776000 { true }

        # 1000 * 2 ** 50
        mem_multiple("Pi") = 1125899906842624000 { true }

        # 1000 * 2 ** 60
        mem_multiple("Ei") = 1152921504606846976000 { true }

        get_suffix(mem) = suffix {
          not is_string(mem)
          suffix := ""
```

```
    }

    get_suffix(mem) = suffix {
      is_string(mem)
      count(mem) > 0
      suffix := substring(mem, count(mem) - 1, -1)
      mem_multiple(suffix)
    }

    get_suffix(mem) = suffix {
      is_string(mem)
      count(mem) > 1
      suffix := substring(mem, count(mem) - 2, -1)
      mem_multiple(suffix)
    }

    get_suffix(mem) = suffix {
      is_string(mem)
      count(mem) > 1
      not mem_multiple(substring(mem, count(mem) - 1, -1))
      not mem_multiple(substring(mem, count(mem) - 2, -1))
      suffix := ""
    }

    get_suffix(mem) = suffix {
      is_string(mem)
      count(mem) == 1
      not mem_multiple(substring(mem, count(mem) - 1, -1))
      suffix := ""
    }

    get_suffix(mem) = suffix {
      is_string(mem)
      count(mem) == 0
      suffix := ""
    }

    canonify_mem(orig) = new {
      is_number(orig)
      new := orig * 1000
    }

    canonify_mem(orig) = new {
      not is_number(orig)
      suffix := get_suffix(orig)
      raw := replace(orig, suffix, "")
```

```
          re_match("^[0-9]+(\\.[0-9]+)?$", raw)
          new := to_number(raw) * mem_multiple(suffix)
        }

        violation[{"msg": msg}] {
          general_violation[{"msg": msg, "field": "containers"}]
        }

        violation[{"msg": msg}] {
          general_violation[{"msg": msg, "field": "initContainers"}]
        }

        # Ephemeral containers not checked as it is not possible to set
field.

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          cpu_orig := container.resources.limits.cpu
          not canonify_cpu(cpu_orig)
          msg := sprintf("container <%v> cpu limit <%v> could not be
parsed", [container.name, cpu_orig])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          mem_orig := container.resources.limits.memory
          not canonify_mem(mem_orig)
          msg := sprintf("container <%v> memory limit <%v> could not be
parsed", [container.name, mem_orig])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          not container.resources
          msg := sprintf("container <%v> has no resource limits",
[container.name])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          not container.resources.limits
          msg := sprintf("container <%v> has no resource limits",
[container.name])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          missing(container.resources.limits, "cpu")
          msg := sprintf("container <%v> has no cpu limit",
[container.name])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          missing(container.resources.limits, "memory")
          msg := sprintf("container <%v> has no memory limit",
[container.name])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          cpu_orig := container.resources.limits.cpu
          cpu := canonify_cpu(cpu_orig)
          max_cpu_orig := input.parameters.cpu
          max_cpu := canonify_cpu(max_cpu_orig)
          cpu > max_cpu
          msg := sprintf("container <%v> cpu limit <%v> is higher than
the maximum allowed of <%v>", [container.name, cpu_orig, max_cpu_orig])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          mem_orig := container.resources.limits.memory
          mem := canonify_mem(mem_orig)
          max_mem_orig := input.parameters.memory
          max_mem := canonify_mem(max_mem_orig)
          mem > max_mem
          msg := sprintf("container <%v> memory limit <%v> is higher
than the maximum allowed of <%v>", [container.name, mem_orig,
max_mem_orig])
        }
      libs:
        - |
          package lib.exempt_container
          is_exempt(container) {
              exempt_images := object.get(object.get(input,
"parameters", {}), "exemptImages", [])
              img := container.image
              exemption := exempt_images[_]
              _matches_exemption(img, exemption)
          }

          _matches_exemption(img, exemption) {
              not endswith(exemption, "*")
              exemption == img
          }

          _matches_exemption(img, exemption) {
              endswith(exemption, "*")
              prefix := trim_suffix(exemption, "*")
              startswith(img, prefix)
          }
```

request template:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8scontainerrequests
  annotations:
    metadata.gatekeeper.sh/title: "Container Requests"
    metadata.gatekeeper.sh/version: 1.0.0
    description: >-
      Requires containers to have memory and CPU requests set and
constrains
      requests to be within the specified maximum values.


https://kubernetes.io/docs/concepts/configuration/manage-resources-conta
iners/
spec:
  crd:
    spec:
      names:
        kind: K8sContainerRequests
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            exemptImages:
              description: >-
                Any container that uses an image that matches an entry
in this list will be excluded
                from enforcement. Prefix-matching can be signified with
`*`. For example: `my-image-*`.

                It is recommended that users use the fully-qualified
Docker image name (e.g. start with a domain name)
                in order to avoid unexpectedly exempting images from an
untrusted repository.
              type: array
              items:
                type: string
            cpu:
              description: "The maximum allowed cpu request on a Pod,
exclusive."
              type: string
            memory:
              description: "The maximum allowed memory request on a Pod,
exclusive."
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8scontainerrequests

        import data.lib.exempt_container.is_exempt

        missing(obj, field) = true {
          not obj[field]
        }

        missing(obj, field) = true {
          obj[field] == ""
        }

        canonify_cpu(orig) = new {
          is_number(orig)
          new := orig * 1000
        }

        canonify_cpu(orig) = new {
          not is_number(orig)
          endswith(orig, "m")
          new := to_number(replace(orig, "m", ""))
        }

        canonify_cpu(orig) = new {
          not is_number(orig)
          not endswith(orig, "m")
          re_match("^[0-9]+(\\.[0-9]+)?$", orig)
          new := to_number(orig) * 1000
        }

        # 10 ** 21
        mem_multiple("E") = 1000000000000000000000 { true }

        # 10 ** 18
        mem_multiple("P") = 1000000000000000000 { true }

        # 10 ** 15
        mem_multiple("T") = 1000000000000000 { true }

        # 10 ** 12
        mem_multiple("G") = 1000000000000 { true }

        # 10 ** 9
        mem_multiple("M") = 1000000000 { true }

        # 10 ** 6
        mem_multiple("k") = 1000000 { true }

        # 10 ** 3
        mem_multiple("") = 1000 { true }

        # Kubernetes accepts millibyte precision when it probably
shouldn't.
        # https://github.com/kubernetes/kubernetes/issues/28741
        # 10 ** 0
        mem_multiple("m") = 1 { true }

        # 1000 * 2 ** 10
        mem_multiple("Ki") = 1024000 { true }

        # 1000 * 2 ** 20
        mem_multiple("Mi") = 1048576000 { true }

        # 1000 * 2 ** 30
        mem_multiple("Gi") = 1073741824000 { true }

        # 1000 * 2 ** 40
        mem_multiple("Ti") = 1099511627776000 { true }

        # 1000 * 2 ** 50
        mem_multiple("Pi") = 1125899906842624000 { true }

        # 1000 * 2 ** 60
        mem_multiple("Ei") = 1152921504606846976000 { true }

        get_suffix(mem) = suffix {
          not is_string(mem)
          suffix := ""
        }

        get_suffix(mem) = suffix {
          is_string(mem)
          count(mem) > 0
          suffix := substring(mem, count(mem) - 1, -1)
          mem_multiple(suffix)
        }
```

```
get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) > 1
  suffix := substring(mem, count(mem) - 2, -1)
  mem_multiple(suffix)
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) > 1
  not mem_multiple(substring(mem, count(mem) - 1, -1))
  not mem_multiple(substring(mem, count(mem) - 2, -1))
  suffix := ""
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) == 1
  not mem_multiple(substring(mem, count(mem) - 1, -1))
  suffix := ""
}

get_suffix(mem) = suffix {
  is_string(mem)
  count(mem) == 0
  suffix := ""
}

canonify_mem(orig) = new {
  is_number(orig)
  new := orig * 1000
}

canonify_mem(orig) = new {
  not is_number(orig)
  suffix := get_suffix(orig)
  raw := replace(orig, suffix, "")
  re_match("^[0-9]+(\\.[0-9]+)?$", raw)
  new := to_number(raw) * mem_multiple(suffix)
}

violation[{"msg": msg}] {
  general_violation[{"msg": msg, "field": "containers"}]
}
```

```
        violation[{"msg": msg}] {
          general_violation[{"msg": msg, "field": "initContainers"}]
        }

        # Ephemeral containers not checked as it is not possible to set
field.

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          cpu_orig := container.resources.requests.cpu
          not canonify_cpu(cpu_orig)
          msg := sprintf("container <%v> cpu request <%v> could not be
parsed", [container.name, cpu_orig])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          mem_orig := container.resources.requests.memory
          not canonify_mem(mem_orig)
          msg := sprintf("container <%v> memory request <%v> could not
be parsed", [container.name, mem_orig])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          not container.resources
          msg := sprintf("container <%v> has no resource requests",
[container.name])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          not container.resources.requests
          msg := sprintf("container <%v> has no resource requests",
[container.name])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          missing(container.resources.requests, "cpu")
          msg := sprintf("container <%v> has no cpu request",
[container.name])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          missing(container.resources.requests, "memory")
          msg := sprintf("container <%v> has no memory request",
[container.name])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          cpu_orig := container.resources.requests.cpu
          cpu := canonify_cpu(cpu_orig)
          max_cpu_orig := input.parameters.cpu
          max_cpu := canonify_cpu(max_cpu_orig)
          cpu > max_cpu
          msg := sprintf("container <%v> cpu request <%v> is higher than
the maximum allowed of <%v>", [container.name, cpu_orig, max_cpu_orig])
        }

        general_violation[{"msg": msg, "field": field}] {
          container := input.review.object.spec[field][_]
          not is_exempt(container)
          mem_orig := container.resources.requests.memory
          mem := canonify_mem(mem_orig)
          max_mem_orig := input.parameters.memory
          max_mem := canonify_mem(max_mem_orig)
          mem > max_mem
          msg := sprintf("container <%v> memory request <%v> is higher
than the maximum allowed of <%v>", [container.name, mem_orig,
max_mem_orig])
        }
      libs:
        - |
          package lib.exempt_container

          is_exempt(container) {
              exempt_images := object.get(object.get(input,
"parameters", {}), "exemptImages", [])
              img := container.image
              exemption := exempt_images[_]
              _matches_exemption(img, exemption)
          }
          _matches_exemption(img, exemption) {
              not endswith(exemption, "*")
              exemption == img
          }

          _matches_exemption(img, exemption) {
              endswith(exemption, "*")
              prefix := trim_suffix(exemption, "*")
              startswith(img, prefix)
          }
```

2) Constraint will be created for each request and limit:

request constraint:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sContainerRequests
metadata:
  name: container-must-have-requests
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    cpu: "200m"
    memory: "10Mi"
```

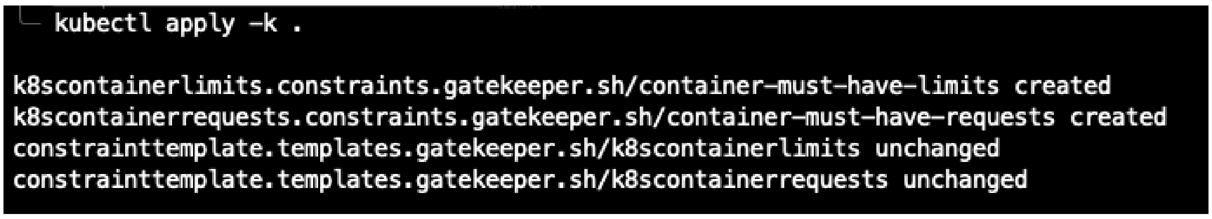
limit constraint:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sContainerLimits
metadata:
  name: container-must-have-limits
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    cpu: "800m"
    memory: "800Mi"
```

3) To apply manifest easily we are using “Kustomization” manifest.

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - restrict-container-limits-constraint.yaml
  - restrict-container-limits-template.yaml
  - restrict-container-requests-constraint.yaml
  - restrict-container-requests-template.yaml
```

4) Apply “Kustomization” manifest:



```
└─ kubectl apply -k .

k8scontainerlimits.constraints.gatekeeper.sh/container-must-have-limits created
k8scontainerrequests.constraints.gatekeeper.sh/container-must-have-requests created
constrainttemplate.templates.gatekeeper.sh/k8scontainerlimits unchanged
constrainttemplate.templates.gatekeeper.sh/k8scontainerrequests unchanged
```

5) Test creating pods who don’t have resources;

```
$ kubectl run nginx --image=nginx

Error from server (Forbidden): admission webhook
"validation.gatekeeper.sh" denied the request:
[container-must-have-limits] container <nginx> has no resource limits
[container-must-have-requests] container <nginx> has no resource
requests
```

6) Test creating pods who are exceeding limit stated in constraint;

limit-exceed.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: opa-disallowed
  labels:
    owner: me.agilebank.demo
spec:
  containers:
    - name: opa
      image: openpolicyagent/opa:0.9.2
      args:
        - "run"
        - "--server"
        - "--addr=localhost:8080"
      resources:
        requests:
          cpu: "100m"
          memory: "2Gi"
```

Results:

```
kubectl apply -f limit-exceed.yaml

Error from server (Forbidden): error when creating "limit-exceed.yaml":
admission webhook "validation.gatekeeper.sh" denied the request:
[container-must-have-limits] container <opa> has no resource limits
[container-must-have-requests] container <opa> memory request <2Gi> is
higher than the maximum allowed of <10Mi>
```

Summary

OPA Gatekeeper is a global unique tool to administrate multi-tenancy organizations in the AWS EKS cluster. By using request and limits restriction, tenants that run in the different namespaces won't have affected the nodes in common use.