International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Enhance the Application Security using Kubernetes Role Based Access Control for Applications

Pallavi Priya Patharlagadda

United States of America
Email: *pallavipriya527.p[at]gmail.com*

Abstract: ***The goal of an identity and access management system is to guarantee that you have control over who can access your information system and what can be accessed by users. Since it is one of the core procedures in security management, it needs to be handled carefully. Identity and user management in Kubernetes should be handled via third-party IAM solutions such as Keycloak, Active Directory, Google's IAM, etc. as these are not incorporated into the platform itself. On the other side, Kubernetes manages authorization and authentication. This article will concentrate on the authorization features of Kubernetes' Identity and Access Management (IAM), particularly on how to use the Role-Based Access Control model to ensure that a user has the appropriate permissions for the appropriate resources.***

Keywords: identity and access management, Kubernetes security, IAM solutions, rolebased access control, user authorization

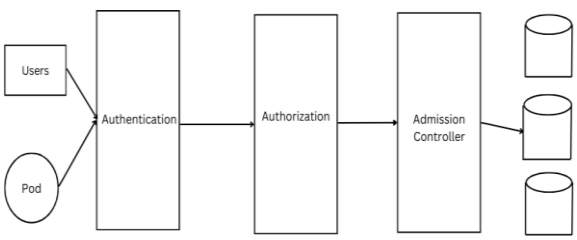
1. Problem Statement

Growing in popularity, Kubernetes has made the ideas of deploying, scaling, and maintaining containerized applications familiar to many developers and administrators. Security is one Kubernetes component that is essential to production deployments that everyone needs to focus on. It's critical to comprehend how the platform handles user and application authorization and authentication. It is necessary to be able to secure a Kubernetes cluster to prevent unauthorized users from having administrator access. A technique for controlling access to computer or network resources based on the responsibilities of specific individuals inside your company is called role-based access control, or RBAC. Kubernetes RBAC authorization enables you to dynamically configure policies via the Kubernetes API. Authorization choices are driven by the rbac.authorization.k8s.io API group. Let’s explore how RBAC would solve the security concerns.

2. Introduction

Before delving into the topic of RBAC, it is imperative to grasp the entire context in which a user or application desires access to Kubernetes objects. Only then can we discuss the role that RBAC plays within these stages. It's a three-stage process all around. While authorization—where RBAC comes into play—will be our primary emphasis, we will touch on admission control and authentication in passing. The below diagram depicts the scenario of a usual request flow.

1) Kubernetes Request Flow:



2) Authentication

Once the request makes it past TLS, it moves on to the authentication stage, where one or more authenticator modules examine the request content.

The administrator configures authentication modules while creating the cluster. If more than one authentication module is configured for a cluster, then each module is attempted one after the other until one of them is successful.

Client certificates, passwords, plain tokens, bootstrap tokens, and JWT tokens (used for service accounts) are a few common authentication modules. The most typical and default circumstance is the use of client certificates. Consult the Kubernetes documentation for a comprehensive inventory of authentication components.

It's critical to realize that Kubernetes lacks a standard user database and profiles for user authentication. Rather, it employs random strings that are taken out of X.509 tokens and certificates and run through the authentication modules. Through one of the authentication modules, Kubernetes can be coupled with external authentication mechanisms from OpenID, Github, or even LDAP.

3) Authorization

The next step after authenticating an API request is to ascertain whether the activity is permitted. The access control pipeline's second stage is where this is completed.

Kubernetes considers three factors when granting a request: the requester's login, the action being asked, and the object that will be impacted by the action. The action is one of the HTTP verbs, such as GET, POST, PUT, and DELETE, mapped to CRUD activities; the object is one of the legitimate Kubernetes objects, such as a pod or a service; and the username is taken from the token encoded in the header.

The authorization is decided by Kubernetes using an established policy. Since Kubernetes adheres to the closed-to-open concept by default, an explicit allow policy is necessary to access the resources.

Like authentication, authorization is set up according to one or more modules, including Webhook, RBAC, and ABAC modes. The authorization modules associated with the API server are configured by the administrator during cluster creation. When many permission modules are being used, Kubernetes verifies each one individually. If a module approves the request, it can move forward. This indicates that the request is refused (HTTP status code 403) if every module rejects it. There is a list of approved permission modules in the Kubernetes documentation.

All requests are processed when you use kubectl in the default configuration since you are regarded as the cluster administrator. However, new users have restricted access by default when they are added.

4) Admission Control

The admission control is the penultimate and last step in the request process. Pluggable modules are the key to admission control, just like they are to authorization and authentication processes.

In contrast to the first two phases, the last step could alter the intended items. Admission control modules don't read objects; instead, they react to changes, deletions, and connections (proxies). To employ a certain storage class, for instance, the request for the formation of a persistent volume claim (PVC) may be modified using an admission control module. The pulling of pictures each time a pod is produced is another policy that a module might impose. See the Kubernetes documentation for a further discussion of the admission control module.

If any admission controller module rejects a request during the access control procedure, it is refused right away. A request is written to the object store after passing through each admission controller and being validated using the associated API object's validation methods.

5) Role-Based Access Control:

In Kubernetes, role-based access control, or RBAC, is a very important but somewhat confusing notion. So, allow me to explain it simply and demystify it. Understanding that there are three components to RBAC is necessary to completely comprehend the concept. Let's learn each of the components in more detail.

a) Subjects:

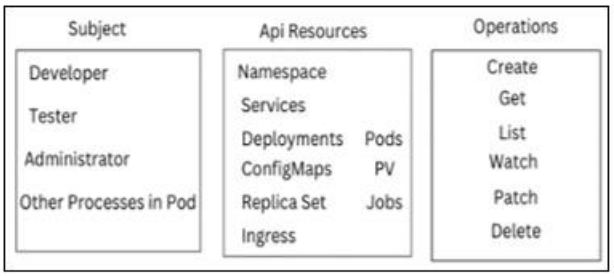
The group of individuals and procedures requesting access to the Kubernetes API.

b) Api Resources:

The collection of cluster-available Kubernetes API Objects. Pods, Deployments, Services, Nodes, and PersistentVolumes are a few examples.

c) Operations:

The group of commands that can be used with the above resources. While several bindings can be used (such as get, watch, create, delete, etc.), they are all ultimately CRUD (create, read, update, or delete) actions.



Keeping these three things in mind, the following is the main concept of RBAC:

Subjects, operations, and API resources should all be connected. Stated otherwise, our goal is to define, for a given user, the actions that can be carried out over a collection of resources.

Thus, by considering the connections between these three categories of things, we may comprehend the many RBAC API Objects that Kubernetes offers.

d) Roles:

Roles Will link verbs with API resources. These can be used again for many topics. These are restricted to a single namespace (we can deploy the same role object in many namespaces, but we cannot use wildcards to represent more than one). The same object is called ClusterRoles if we want the role to be applied throughout the entire cluster.

e) RoleBinding:

RoleBinding Will link the remaining subjects that are entities. We will determine which subjects can use a role that already ties verbs and API Objects. There are ClusterRoleBindings for the non-namespaced, cluster-level equivalent.
Let's learn each of them in more detail.

6) Types of RBAC roles:

Based on the scope, there are two types of Roles in Kubernetes.

a) A Kubernetes role is limited to resources (such as deployment or service) that are located inside a particular namespace.

Below is the sample yaml file for Role

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: examplerole
  namespace: example-ns
rules:
 - apiGroups: [""]
  resources: ["pods"]
  verbs:                    ["list",                    "get"]
```

b) A Kubernetes cluster role is scoped to either namespace-scoped resources, which are present in every namespace, like pods, or cluster-wide resources, like worker nodes. Below is the sample yml file.

```
kind: ClusterRole
apiVersion:rbac.authorization.k8s.io/v1beta1
metadata:
  name: exampleclusterrole
```

```
rules:
 - apiGroups: ["*"]
   resources: ["pods"]
   verbs: ["get", "list"]
```

7) Role Binding and Types of Role Binding:

Role bindings assign cluster or RBAC roles to a designated namespace.

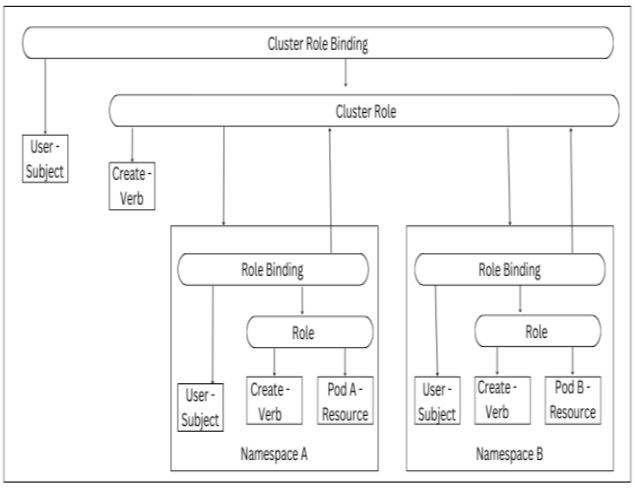
Role Binding:

A user is granted access to a certain resource inside a designated namespace when a role binding is utilized to apply a role. Applying a cluster role through role binding grants a user access to namespace-scoped resources, such as pods, that are present in every namespace but are only accessible within that namespace. Below is the sample yml file.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: example-rolebinding
  namespace: example-ns
subjects:
 - kind: User
   name: Alice
   apiGroup: rbac.authorization.k8s.io
roleRef:
   kind: Role
   name: examplerole
   apiGroup:                    rbac.authorization.k8s.io
```

2. Cluster Role Binding

RBAC cluster roles are applied to every namespace in the cluster through cluster role bindings. A user gains access to resources that are scoped across namespaces, such as pods, or cluster-wide resources, such as worker nodes, when a cluster role binding is used to apply a cluster role.



Below is the yaml file for ClusterRoleBinding.

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: exampleclusterbinding
subjects:
 - kind: User
   name: Alice
   apiGroup: rbac.authorization.k8s.io
roleRef:
   kind: ClusterRole
   name: exampleclusterrole
   apiGroup: rbac.authorization.k8s.io
```

1) Service Account:

A concept that many Kubernetes users find difficult to understand is that of subjects, notably the distinction between ServiceAccounts and normal users. Theoretically, it appears straightforward:

a) Users:

These are aimed at people or processes that reside outside of the cluster and are global.

b) ServiceAccounts:

Designed for intra-cluster processes operating within pods, these are namespaced.

Their domains appear to be well-defined, and they both share the desire to authenticate against the API to carry out a certain set of activities over a certain set of resources (keep in mind the preceding section). In addition, they can be a part of groups, meaning that multiple subjects can be bound by a role binding (albeit ServiceAccounts can only be a part of the "system:serviceaccounts" group).

Because service accounts are API objects, they are separately RBAC managed, allowing each one to have a unique set of RBAC permissions. Moreover, service accounts have the same labeling capabilities as any other Kubernetes object. This indicates that Kubernetes has a somewhat fine-grained RBAC-controlled label artifact available.

However, there are a few things to be mindful of when using service accounts. There can only be one service account associated with each pod. A service account can be compared to a distinct pod identifier.

On the other hand, service accounts—just like any other Kubernetes API object—can also be tagged. This means that, similarly to label selectors for pods, services, and network policies that adhere to normal Kubernetes practice, those labels can be used to construct "groups" of service accounts.

You can restrict which users can alter a service account and its labels using RBAC because service accounts are independent API objects.

This makes it possible to designate trustworthy entities (code and/or persons) that can grant a specific service account the required selectors (service account labels) and enable that service account to be used in a specific namespace. One can create a service account using the below command.

kubectl create serviceaccount example-sa --namespace example-ns

Service Account can also be created using the below yml file.

```
apiVersion: v1
```

```
kind: ServiceAccount
metadata:
  name: sample-sa
  namespace: example-ns
```

The RoleBinding example-role binding links the Role example-role to the ServiceAccount example in the example below:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: example-rolebinding
  namespace: example-ns
subjects:
- kind: ServiceAccount
  name: example-sa
  namespace: mynamespace
roleRef:
  kind: Role
  name: examplerole
  apiGroup: rbac.authorization.k8s.io
```

3. Conclusion

For most platforms, role-based access controls are standard, and Kubernetes is no exception. RBAC policies are required to use Kubernetes in production. These cannot be thought of as a limited list of Kubernetes API Objects administrators need to be aware of. To deploy safe apps and take full advantage of the potential that the Kubernetes API provides for their cloud-native apps, application developers will require them. It's beneficial to thoroughly design your responsibilities to ensure their reusability across many scenarios. We have seen how to utilize the Role-based Access Control model to issue permissions to a service account or a user.

Although it is arguably the most often used model, ABAC (attribute-based access control), the Webhook mode, and the Node Authorization model are alternative effective models that may be used to create authorization in Kubernetes.

Because service accounts allow you to monitor and manage resource access in Kubernetes, they are a very useful tool for cluster administration. They can be used to restrict access to specific namespaces. Pods should only be allowed access to what they require. Understanding which resources were accessed by whom and when gives information about cluster activity. After a user departs the team, it's crucial to delete them as well. By employing these techniques, Authorization can be achieved in Kubernetes.

References

[1] https://thenewstack.io/a-primer-on-kubernetes-access-control/
[2] https://kubernetes.io/docs/reference/access-authn-authz/controlling-access/
[3] https://medium.com/@ishagirdhar/rbac-in-kubernetes-demystified-72424901fcb3
[4] https://theithollow.com/2019/05/20/kubernetes-role-based-access/
[5] https://www.cncf.io/blog/2018/08/01/demystifying-rbac-in-kubernetes/
[6] https://schoolofdevops.github.io/ultimate-kubernetes-bootcamp/configuring_authentication_and_authorization/
[7] https://imti.co/team-kubernetes-remote-access/
[8] https://schoolofdevops.github.io/ultimate-kubernetes-bootcamp/configuring_authentication_and_authorization/
[9] https://kubernetes.io/blog/2017/04/rbac-support-in-kubernetes/
[10] https://thenewstack.io/kubernetes-access-control-exploring-service-accounts/
[11] https://laszlo.cloud/Why-access-control-is-key-for-a-secure-multi-tenant-Kubernetes-deployment
[12] https://kubernetes.io/blog/2017/10/using-rbac-generally-available-18/
[13] https://platform9.com/blog/the-gorilla-guide-to-kubernetes-in-the-enterprise-chapter-4-putting-kubernetes-to-work/#:~:text=The%20types%20of%20Role%20Based,and%20service%20accounts%20to%20Roles).
[14] https://www.ibm.com/docs/en/cloud-private/3.2.0?topic=private-role-based-access-control
[15] https://www.tremolosecurity.com/post/kubernetes-identity-management-part-ii-rbac-and-user-provisioning
[16] https://dev.to/mhausenblas/on-some-defaults-in-kubernetes-rbac-270l
[17] https://software.danielwatrous.com/self-service-access-control-in-kubernetes/
[18] https://techdocs.akamai.com/cloud-computing/docs/secure-a-cluster-with-user-permissions-and-rbac
[19] https://www.yld.io/blog/testing-kubernetes-rbac
[20] https://docs.redhat.com/en/documentation/openshift_container_platform/3.9/html/cluster_administration/admin-guide-manage-rbac#creating-cluster-role
[21] https://www.adaltas.com/en/2019/08/07/users-rbac-kubernetes/