TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Valentin Kirjan 201728IVSB

Facilitation of Kubernetes Role-Based Access Control Implementation and Management with Open-Source Software

Bachelor's thesis

Supervisor: Aleksei Talisainen
MSc.

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Valentin Kirjan 201728IVSB

Kubernetese rollipõhise juurdepääsukontrolli juurutamise ja haldamise hõlbustamine avatud lähtekoodiga tarkvara abil

Bakalaureusetöö

Juhendaja: Aleksei Talisainen
MSc.

Tallinn 2024

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Valentin Kirjan

04.01.2024

Abstract

Kubernetes is the most popular container orchestration system in the world, which makes it one of the common attack surfaces for malicious actors. According to recent studies, its security adoption is not in the best shape, which makes both employees and organizations pay the price for related security incidents. While Kubernetes does provide mechanisms for enhancing security, these features may not be enabled by default and require additional configuration: one of such features is role-based access control (RBAC).

This thesis conducts a detailed study and prototype development to enhance Kubernetes RBAC implementation and management. The research begins with an exploration of Kubernetes RBAC, followed by the selection and comparative study of two open-source modern tools, *Permission Manager* and *RBAC Tool*, in a lab environment replicating real-world scenarios. These tools are examined based on set criteria to identify their functionalities and limitations. Requirements for the prototype are then established based on these limitations, leading to the development of a solution offering improved or additional functionalities. The prototype is designed with a web-based user interface, split into two components, and operates locally, outside a Kubernetes cluster. The research concludes with suggestions for future improvements of the prototype.

The thesis contributes to the field of Kubernetes security and administration by educating and attempting to benefit individuals and organizations adopting role-based access control.

This thesis is written in English and is 30 pages long, including 6 chapters, 28 figures and 2 tables.

List of abbreviations and terms

API	Application Programming Interface
CA	Certificate Authority
CNCF	Cloud Native Computing Foundation
Dockerfile	A text file that contains instructions for the Docker daemon on how to build a container image.
GitHub	A platform and cloud-based service for software development and version control.
JavaScript	An interpreted programming language that is one of the core technologies of the World Wide Web.
JSON	JavaScript Object Notation
MicroK8s	An open-source system for automating deployment, scaling, and management of containerised applications, providing the functionality of core Kubernetes components.
OS	Operating System
RBAC	Role-based Access Control
TypeScript	A strongly typed programming language developed by Microsoft. It adds static typing with type annotations to JavaScript.
UI	User Interface
YAML	Yet Another Markup Language or YAML Ain't Markup Language is a data serialization language.

Table of contents

List of figures

List of tables

1 Introduction

In recent years, software containerization has become a standard for packaging and deploying software code: 93% of CNCF annual survey respondents were using, or were planning to use, containers in production in 2021 [1]. The containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable — called a container — that runs consistently on any infrastructure, adding efficiency to the software development lifecycle [2].

One of the pioneers of containerization is Google, who introduced Borg, their container cluster management system, in 2003. With three people initially, it started as a small-scale project [3] and grew into a full-fledged, high-capacity tool of utmost importance for the organization. For example, huge Google applications, such as Search and Gmail, have been running at extreme scale on containers for years [4]. In 2013, Docker arrived on the scene and revolutionized containerization by providing an uncomplicated way to package, distribute, and deploy applications on a single machine. This development fuelled the demand for a dependable, large-scale container management system. Inspired by this need, Google employees began developing the first prototype of Kubernetes, their open-source version of Borg [5].

Moving forward to the current year, Kubernetes has become the most popular container orchestration system in the world: the CNCF 2022 annual survey shows that 64% of CNCF's end-user organizations are running Kubernetes clusters in production, while 25% are piloting or evaluating the tool. For the non-end user organizations, the percentages are 49% and 20% respectively [6]. The widespread usage of Kubernetes makes it one of the common attack surfaces for malicious actors, which is a major concern for organizations adopting cloud-native environments.

1.1 Research problem

While Kubernetes does provide mechanisms for enhancing security, such as network policies and role-based access control, these features may not be enabled by default and require additional configuration: State of Kubernetes security report 2023 by Red Hat claims that 49% of its 600 respondents had a security incident during runtime, 45% experienced a misconfiguration incident, and another 42% discovered a major vulnerability to remediate. Most importantly, as a result of a container and Kubernetes security incident, 37% of respondents identified revenue or customer loss, 25% said the organization was fined, and 21% experienced employee termination. In retrospect, 38% cited that their company's container strategy was not taking security seriously or investing in security adequately, and 25% thought it was progressing too slowly [7].

The importance of Kubernetes security rises day by day as more organizations integrate container orchestration solutions. The aforementioned role-based access control (RBAC) is a widely adopted access control model that follows guidelines set by modern cybersecurity standards, such as ISO 27001 and NIST. Since Kubernetes RBAC is a key security control to ensure that cluster users and workloads have only the access to resources required to execute their roles [8], it plays a crucial role in hardening the security of Kubernetes clusters: its neglection or misconfiguration can lead to security incidents, which are still common and cause damage to organizations up to this day.

1.2 Research objectives and goal

Third-party open-source software tools exist to assist in working with various aspects of the RBAC in Kubernetes. The objectives of this research are:

1. To examine and outline the functionality of the modern open-source tools dedicated to the implementation and management of RBAC cluster objects.
2. To identify potential areas for improvement or alternative approaches.

The primary goal of this work is to develop a software prototype that incorporates functionalities derived from the identified areas, intending to reduce complexity and enhance efficiency and effectiveness in the RBAC implementation process.

1.3 Scope and target audience

This thesis concentrates solely on the role-based access control security mechanism in Kubernetes and associated open-source software solutions that aid in its implementation and management, which includes the creation and application of RBAC object YAML configurations and the modification of existing objects within a cluster. Importantly, the thesis does not employ specific cybersecurity frameworks, as they are deemed unnecessary for the scope of this research. The focus is primarily on the practical application within Kubernetes environments, where inherent features provide a sufficient basis for exploring and enhancing RBAC security measures, without the need for external cybersecurity frameworks. According to the report papers [7], Kubernetes security adoption is not in the best shape, thus the work may be of interest to cybersecurity experts or other technical roles that work with container orchestration and want to secure their clusters.

2 Role-based access control design in Kubernetes

Starting from its initial stable release in Kubernetes v1.8, RBAC has remained an integral element of Kubernetes security [4], continuing through the latest version, which is v1.29. RBAC is a mechanism that allows Kubernetes administrators to grant permissions within a Kubernetes cluster. It defines a set of roles that can be assigned to different entities, such as users, groups, and service accounts. These roles can be used to authorize access to various resources and operations within the cluster [9].

2.1 Role-based access control objects

The RBAC API declares four kinds of Kubernetes object: Role, ClusterRole, RoleBinding, and ClusterRoleBinding [9].

2.1.1 Role and ClusterRole

An RBAC Role or ClusterRole contains rules that represent a set of permissions. Permissions are purely additive — there are no "deny" rules. A Role always sets permissions within a particular namespace, an abstraction used by Kubernetes to support isolation of groups of resources within a single cluster. When one creates a Role, they must specify the namespace it belongs to. ClusterRole, by contrast, is a non-namespaced resource. The resources have different names, Role and ClusterRole, because a Kubernetes object always must be either namespaced or not namespaced; it cannot be both [9].

Table 1. Role and ClusterRole usage possibilities.

Role	ClusterRole
Defines permissions on namespaced resources and grants access within individual namespace (applies to ClusterRole if it is referenced by RoleBinding; see Table 2. RoleBinding and ClusterRoleBinding access combinations.).	
	Defines permissions on namespaced resources and grants access across all namespaces.
	Defines permissions on cluster-scoped resources.

The minimum viable Role and ClusterRole manifest files follow a similar structure, with the only exception being the specification of the *namespace* field in Role. The main points of interest are *apiGroups*, *resources,* and *verbs* fields. The *apiGroups* field can define either a singular or multiple Kubernetes API groups. The API groups include specific resource types and can be native or custom. A single instance of a resource type is called a resource [10]. In the Kubernetes API, most resources are represented and accessed using a string representation of their object name, such as *pods* for a Pod. The *resources* field declares the types of resources on which actions will be granted, with the applicable actions specified in the *verbs* field. The RBAC refers to resources using the same name that appears in the URL for the relevant API endpoint [9].

The example figure below shows the ClusterRole named *namespace-view*. An empty string in the *apiGroups* field indicates the core API group. The ClusterRole permits get, list, and watch actions on namespaces and pods within the core API group. Since namespaces are cluster-scoped resources, this ClusterRole is designed for use with ClusterRoleBinding to grant cluster-wide permissions.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: namespaces-pods-view
rules:
- apiGroups: [""]
  resources: ["namespaces", "pods"]
  verbs: ["get", "list", "watch"]
```

Figure 1. Example: ClusterRole YAML.

The next figure depicts the Role named *pods-edit-jobs-view*. It permits get, list, and watch actions on jobs within the batch API group in the namespace called *example*. It also permits all the possible actions on pods within the core API group of the same namespace.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
namespace: example
name: pods-edit-jobs-view
rules:
- apiGroups: [""]
resources: ["pods"]
verbs: ["create", "delete", "deletecollection",
"patch", "update", "get", "list", "watch"]
- apiGroups: ["batch"]
resources: ["jobs"]
verbs: ["get", "list", "watch"]
```

Figure 2. Example: Role YAML.

Four user-facing ClusterRoles, namely *cluster-admin*, *admin*, *edit*, and *view*, exist by default in a cluster.

2.1.2 RoleBinding and ClusterRoleBinding

A RoleBinding grants the permissions defined in a role to a user or set of users. It holds a list of *subjects* (users, groups, or service accounts), and a reference to the role being granted. A RoleBinding grants permissions within a specific namespace whereas a ClusterRoleBinding grants that access cluster-wide [9].

A RoleBinding may reference any Role in the same namespace. Alternatively, a RoleBinding can reference a ClusterRole and bind that ClusterRole to the namespace of the RoleBinding. A ClusterRoleBinding is used to bind a ClusterRole to all namespaces in a cluster [9].

Table 2. RoleBinding and ClusterRoleBinding access combinations.

	RoleBinding	**ClusterRoleBinding**
Role	Namespace-wide access	Inapplicable
ClusterRole	Namespace-wide access	Cluster-wide access

The ClusterRoleBinding, named *namespaces-pods-view-binding*, is shown below. It binds the *namespaces-pods-view* ClusterRole to the *IT-personnel* group, thereby granting cluster-wide permissions.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: namespaces-pods-view-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: namespaces-pods-view
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: IT-personnel
```

Figure 3. Example: ClusterRoleBinding YAML.

The following figure illustrates the RoleBinding named *pods-edit-jobs-view-binding*. It binds the *pods-edit-jobs-view* Role to the *developer* user, thereby granting permissions within the *example* namespace.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pods-edit-jobs-view-binding
  namespace: example
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pods-edit-jobs-view
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: developer
```

Figure 4. Example: RoleBinding YAML.

2.2 Kubernetes object management

In Kubernetes, the *kubectl* command-line tool supports three management techniques to create and manage objects. These techniques are imperative commands, imperative object configuration, and declarative object configuration. A Kubernetes object should be managed using only one technique. Mixing and matching techniques for the same object results in undefined behavior [11].

2.2.1 Imperative commands

When using imperative commands, a user operates directly on live objects in a cluster. The user provides operations to the *kubectl* command as arguments or flags. This is the recommended way to get started or to run a one-off task in a cluster. Since this technique operates directly on live objects, it provides no history of previous configurations [11].

```
kubectl run examplepod --image=httpd
```

Figure 5. Imperative command.

Advantages compared to object configuration:

- Commands are expressed as a single action word.
- Commands require only a single step to make changes to the cluster.

Disadvantages compared to object configuration:

- Commands do not integrate with change review processes.
- Commands do not provide an audit trail associated with changes.
- Commands do not provide a source of records except for what is live.
- Commands do not provide a template for creating new objects [11].

2.2.2 Imperative object configuration

In imperative object configuration, the *kubectl* command specifies the operation, optional flags, and at least one file name. The specified locally stored file must contain a full definition of the object in YAML or JSON format [11].

```
kubectl create -f examplerole.yaml
```

Figure 6. Imperative object configuration.

Advantages compared to imperative commands:

- Object configuration can be stored in a source control system such as Git.
- Object configuration can integrate with processes such as reviewing changes before push and audit trails.
- Object configuration provides a template for creating new objects.

Disadvantages compared to imperative commands:

- Object configuration requires basic understanding of the object schema.

- Object configuration requires the additional step of writing a YAML file.

Advantages compared to declarative object configuration:

- Imperative object configuration behavior is simpler and easier to understand.
- As of Kubernetes version 1.5, imperative object configuration is more mature.

Disadvantages compared to declarative object configuration:

- Imperative object configuration works best on files, not directories.
- Updates to live objects must be reflected in configuration files, or they will be lost during the next replacement [11].

2.2.3 Declarative object configuration

When using declarative object configuration, a user operates on object configuration files stored locally, however the user does not define the operations to be taken on the files. *Create*, *update*, and *delete* operations are automatically detected per-object by *kubectl*. This enables working on directories, where different operations might be needed for different objects [11].

```
kubectl apply -f exampledir/
```

Figure 7. Declarative object configuration using directory.

In addition, the declarative object configuration supports singular files.

```
kubectl apply -f examplerole.yaml
```

Figure 8. Declarative object configuration using file.

Advantages compared to imperative object configuration:

- Changes made directly to live objects are retained, even if they are not merged back into the configuration files.
- Declarative object configuration has better support for operating on directories and automatically detecting operation types (*create, patch*, *delete*) per-object.

Disadvantages compared to imperative object configuration:

- Declarative object configuration is harder to debug and understand results when they are unexpected.

- Partial updates using *diff* commands create complex merge and patch operations [11].

2.3 Implementation process of role-based access control

The recommended approach is to use the *kubectl auth reconcile -f FILENAME* object configuration command for managing the RBAC objects [9]. This method is preferred to the *kubectl apply -f FILENAME* command for RBAC resources, as it performs semantically-aware merging of rules and subjects. Moreover, it only reconciles rules for the Role, ClusterRole, RoleBinding, and ClusterRoleBinding objects, preventing unintended changes to other objects [12].

```
kubectl auth reconcile -f multiplerbacobjects.yaml
```

Figure 9. Kubectl auth reconcile command.

If required, the command creates missing objects and containing namespaces for namespaced objects. It updates existing roles to include permissions in input objects and removes extra permissions if the *--remove-extra-permissions* flag is specified. Furthermore, it updates existing bindings to include subjects in input objects and removes extra subjects if the *--remove-extra-subjects* flag is specified [12].

Alternatively, RBAC objects can be applied to a cluster through API calls using Kubernetes client libraries that are developed for multiple programming languages [13].

2.4 Authentication using kubeconfig files

The primary method for user authentication in Kubernetes is through kubeconfig files. These files are used to organize information about clusters, users, namespaces, and authentication mechanisms. The kubeconfig file ensures secure access to the Kubernetes API and contains the following key sections:

- *Clusters*. The section defines the Kubernetes clusters that the user can access. Each cluster entry contains a name and the cluster's API server address.
- *Users*. The section lists the credentials of the users. Each user entry may include a client certificate, client key, username, password, or token.
- *Contexts*. The section combines a cluster with a user and an optional namespace. It defines how *kubectl* or other management tools communicate with clusters.

- *Current-context*. The section specifies the context that is used by default.

3 Methodology

The methodology used to achieve the goal involves qualitative research. It begins with the selection and comparative study of existing tools and concludes with prototype development and its assessment.

3.1 Open-source tools selection

The open-source software is selected based on relevance, which includes timely updates, functionality relevant to the implementation and management of Kubernetes RBAC, and a certain level of popularity. Since all the evaluated open-source software is distributed through GitHub, a solution is chosen if it has had a release within the past 12 months, contained documentation in the README.md file demonstrating relevance to the research topic, and has received 50 stars or more. After evaluating several tools, *RBAC Tool* and *Permission Manager* were identified as the tools that meet these criteria [14] [15].

3.2 Comparative study

Having outlined the selection process for the open-source tools, the next step involves a detailed comparative study to compare and analyze the tools. This analysis is important for understanding the practical application and limitations of the selected tools in a real-world scenario. It provides the basis for the subsequent development of the prototype.

3.2.1 Lab environment setup

The analysis is conducted in a lab environment, which consists of:

- an Ubuntu Server 22.04.3 LTS virtual machine hosting a Kubernetes cluster on MicroK8s 1.28/stable;
- two Ubuntu Desktop 22.04.3 LTS virtual machines: one serving as a cluster administrator and the other as a normal cluster user.

MicroK8s is an open-source system for automating deployment, scaling, and management of containerised applications. It provides the functionality of core Kubernetes components, in a small footprint, scalable from a single node to a high-availability production cluster [16].

The virtual machines are hosted on VirtualBox 7.0.10. The Ubuntu distribution of Linux is chosen because it has wide community support and seamlessly integrates with MicroK8s, both of which are developed by Canonical [17].

The environment is configured to resemble real scenarios where a user requires access to specific resources within a cluster. All cluster permissions are managed by the cluster administrator. Both subjects run *kubectl* commands against the cluster instead of accessing the host machine directly.

3.2.2 Assessment criteria

Each tool is evaluated against the following criteria, which are selected to identify the tools' capabilities and limitations:

- RBAC management capabilities. The criterion is set to examine functionalities such as creating, modifying, and deleting RBAC objects. It directly relates to the functional aspect of the tools and their ability to manage access within a Kubernetes cluster.
- Adherence to the principle of least privilege. The criterion determines how well the tool implements granular access controls to ensure a user has no more privileges than necessary to perform their tasks. It reflects the tools' capability to contribute to a secure Kubernetes environment, which is the reason why RBAC was invented in the first place.
- Ease of use for users with limited Kubernetes expertise. The criterion assesses how beginner-friendly the tool is for individuals with limited or no prior knowledge of Kubernetes RBAC by determining the level of detailed knowledge required to perform basic tasks. It is critical for the adoption of the tools by a broader audience and affects how effectively users can leverage the tool's features without prior training. In addition, a tool with a low entry barrier may help its users acquire the subject faster.

3.3 Prototype development

Moving into the prototype development phase, the approach is structured to address the limitations identified in the previous analysis, which involves setting up specific requirements for the prototype's capabilities based on the findings.

3.3.1 Development process

In the development of the prototype, attention is paid to selecting technologies that not only fulfill the functional requirements but also enhance the overall development process.

TypeScript programming language is chosen for both frontend and backend development due to its capacity to improve code maintainability. As a statically typed extension of JavaScript, TypeScript offers robust support for writing more error-resistant code [18].

For the backend, Node.js runtime environment is selected for its efficient handling of concurrent processes and support of the official client library for using the Kubernetes API [13] [19]. The client library is installed by *npm*, the default package manager for Node.js.

For the frontend, Vite local development server is chosen for its rapid development setup, which noticeably accelerates build times [20]. It is complemented by React, which provides a component-based architecture for modular and maintainable UI development.

3.3.2 Design and conceptualization

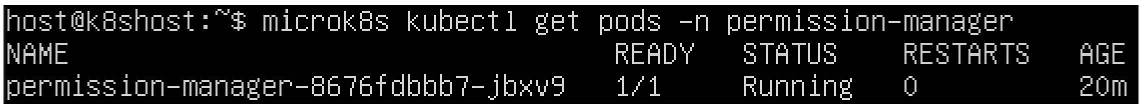
Based on the comparative study of *RBAC Tool* and *Permission Manager*, identified limitations are to be addressed in the prototype by introducing the requirements. To increase efficiency and effectiveness of the underlying work processes, the prototype design focuses on enhancing usability for inexperienced users, ensuring adherence to the principle of least privilege, and expanding RBAC management capabilities.

4 Comparative study of open-source tools

This section delves into the comparative study of *Permission Manager* and *RBAC Tool* in the lab environment (see 3.2.1 Lab environment setup), analyzing their functionalities and how they align with Kubernetes RBAC implementation and management.

4.1 Permission Manager

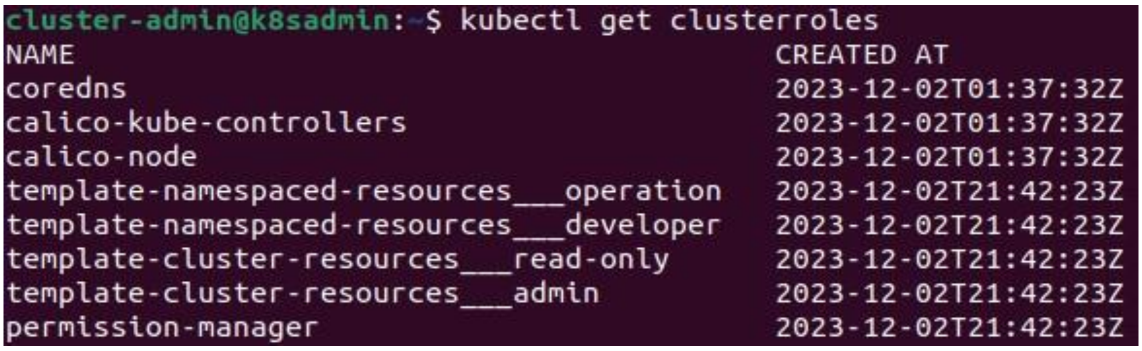
The tool is designed to be installed within a cluster, in a dedicated namespace called *permission-manager*.



```
host@k8shost:~$ microk8s kubectl get pods -n permission-manager
NAME                                   READY   STATUS    RESTARTS   AGE
permission-manager-8676fdbbb7-jbxv9    1/1     Running   0          20m
```

Figure 10. Permission-manager pod running in MicroK8s cluster.

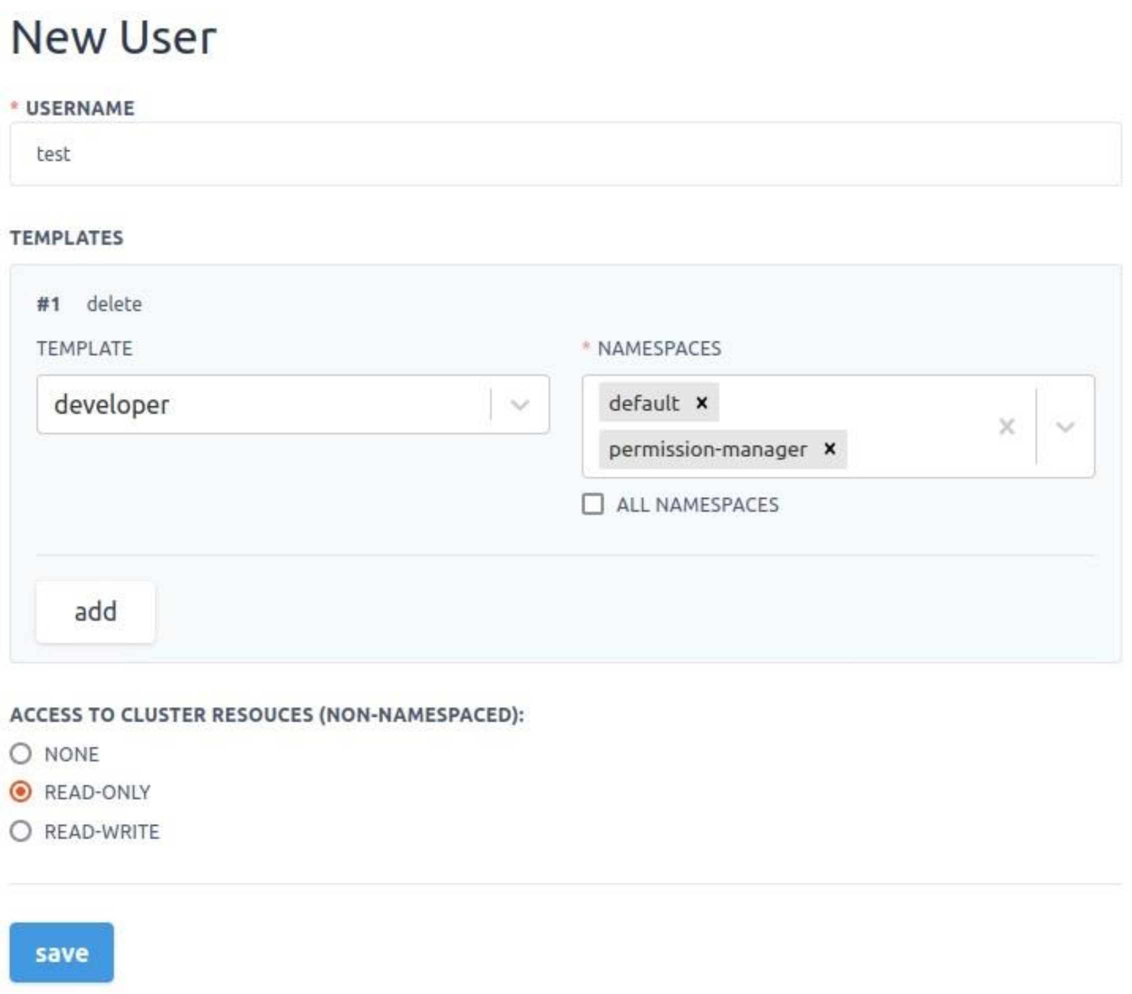
Permission Manager offers a web-based UI for managing Kubernetes RBAC and cluster authentication. It extends existing cluster API groups with the *Permissionmanageruser* CustomResourceDefinition object: the tool provides its own user management system and the ability to generate kubeconfig files for the created users. It uses ClusterRoles, RoleBindings, and ClusterRoleBindings, which are assigned specific names to differentiate them from the ones created normally. The tool comes with four preset ClusterRoles to choose from.



```
cluster-admin@k8sadmin:~$ kubectl get clusterroles
NAME                                            CREATED AT
coredns                                         2023-12-02T01:37:32Z
calico-kube-controllers                         2023-12-02T01:37:32Z
calico-node                                     2023-12-02T01:37:32Z
template-namespaced-resources___operation       2023-12-02T21:42:23Z
template-namespaced-resources___developer       2023-12-02T21:42:23Z
template-cluster-resources___read-only          2023-12-02T21:42:23Z
template-cluster-resources___admin              2023-12-02T21:42:23Z
permission-manager                              2023-12-02T21:42:23Z
```

Figure 11. *Permission Manager* ClusterRoles are prefixed with template-…-resources___name.

A test user is created to assess the capabilities of the tool. The *developer* ClusterRole template is selected to be applied in the *default* and *permission-manager* namespaces. In addition, the *read-only* access to cluster resources is selected.



New User

* USERNAME

test

TEMPLATES

#1 delete

TEMPLATE: developer

* NAMESPACES: default, permission-manager

☐ ALL NAMESPACES

add

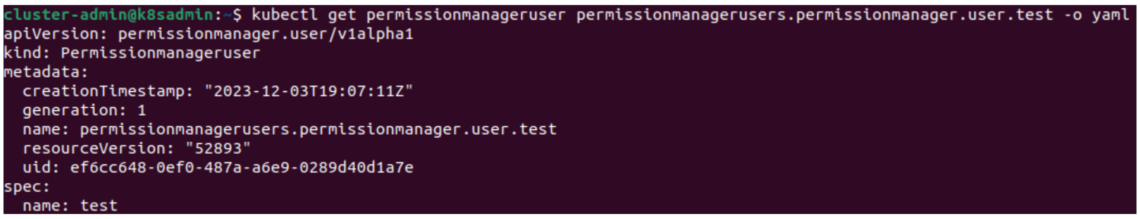
ACCESS TO CLUSTER RESOUCES (NON-NAMESPACED):

NONE [✓ READ-ONLY] READ-WRITE

save

Figure 12. Creating a new user in *Permission manager*.

As a result, two similarly named RoleBindings, *test___template-namespaced-resources___developer___default* and *test___template-namespaced-resources___developer___permission-manager*, are generated in their respective namespaces. The *test___template-cluster-resources___read-only* ClusterRoleBinding is generated. These new RBAC objects bind the ClusterRoles to the test user, granting them namespaced permissions from the *developer* template and cluster-wide permissions from the *read-only* template. The user is recorded as the Permissionmanageruser object.



```
cluster-admin@k8sadmin:~$ kubectl get permissionmanageruser permissionmanagerusers.permissionmanager.user.test -o yaml
apiVersion: permissionmanager.user/v1alpha1
kind: Permissionmanageruser
metadata:
  creationTimestamp: "2023-12-03T19:07:11Z"
  generation: 1
  name: permissionmanagerusers.permissionmanager.user.test
  resourceVersion: "52893"
  uid: ef6cc648-0ef0-487a-a6e9-0289d40d1a7e
spec:
  name: test
```

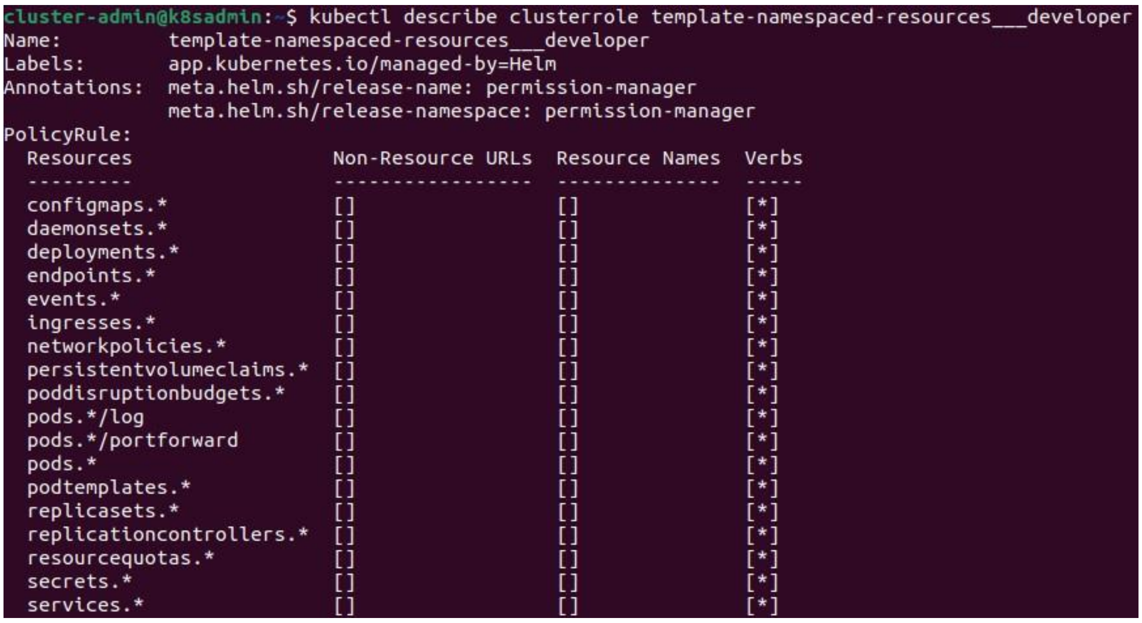
Figure 13. Test user YAML configuration.

4.1.1 RBAC management capabilities

Permission Manager excels in managing users and generating kubeconfig files. However, it limits the authentication method by creating its own users, meaning it does not support users authenticated through identity and access management solutions or certificates. Additionally, the tool relies on manually created ClusterRoles, as it does not assist in their creation. This reliance and a lack of support for more specific Role configurations reveal a gap in its RBAC management capabilities.

4.1.2 Adherence to the principle of least privilege

The tool's primary usage of ClusterRoles and the broad permissions of the provided preset ClusterRoles, even those intended to be more limited, such as the *developer* and *read-only* ClusterRole templates, pose challenges in adhering to the least privilege principle. For instance, the *developer* ClusterRole grants access to a significant number of resources and allows any action on them because it uses wildcards.



```
cluster-admin@k8sadmin:~$ kubectl describe clusterrole template-namespaced-resources___developer
Name:         template-namespaced-resources___developer
Labels:       app.kubernetes.io/managed-by=Helm
Annotations:  meta.helm.sh/release-name: permission-manager
              meta.helm.sh/release-namespace: permission-manager
PolicyRule:
  Resources                 Non-Resource URLs  Resource Names  Verbs
  ---------                 -----------------  --------------  -----
  configmaps.*              []                 []              [*]
  daemonsets.*              []                 []              [*]
  deployments.*             []                 []              [*]
  endpoints.*               []                 []              [*]
  events.*                  []                 []              [*]
  ingresses.*               []                 []              [*]
  networkpolicies.*         []                 []              [*]
  persistentvolumeclaims.*  []                 []              [*]
  poddisruptionbudgets.*    []                 []              [*]
  pods.*/log                []                 []              [*]
  pods.*/portforward        []                 []              [*]
  pods.*                    []                 []              [*]
  podtemplates.*            []                 []              [*]
  replicasets.*             []                 []              [*]
  replicationcontrollers.*  []                 []              [*]
  resourcequotas.*          []                 []              [*]
  secrets.*                 []                 []              [*]
  services.*                []                 []              [*]
```

Figure 14. Policy rules of *developer* ClusterRole template.

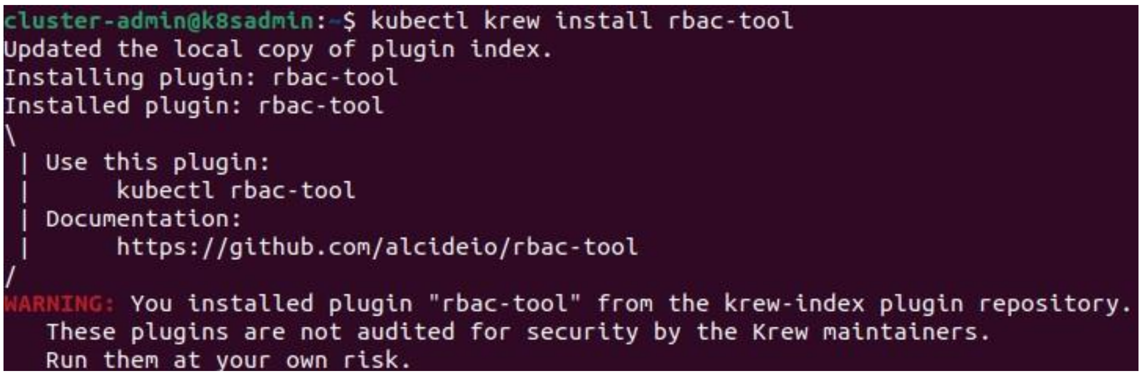
While the tool allows some control through template selection, the lack of support for finetuning ClusterRoles at a granular level may result in potential overprovisioning of permissions. Furthermore, the implementation of numerous ClusterRoles, which are to be namespaced by RoleBindings, pollutes the list of ClusterRoles and is considered a bad practice.

4.1.3 Ease of use for users with limited Kubernetes expertise

The mentioned web-based UI simplifies Kubernetes RBAC management, enhancing usability for users with limited expertise. Its intuitive design facilitates the creation and management of users and RBAC objects. However, the tool's absence of direct Role and ClusterRole management limits its effectiveness for those needing granular control but lacking deep knowledge.

4.2 RBAC Tool

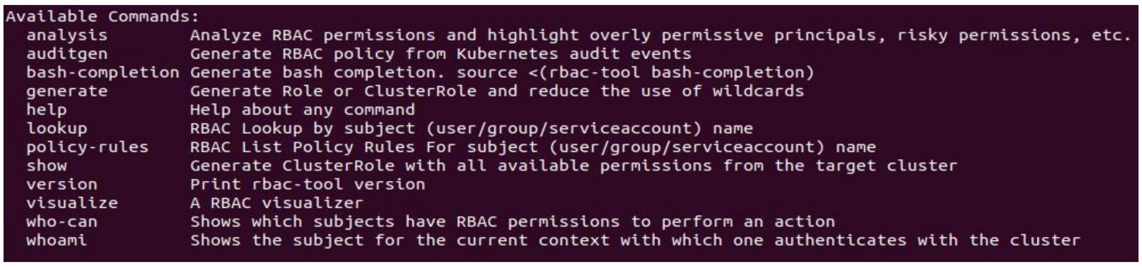
The tool is designed to be installed as a binary package or *kubectl* plugin via Krew.



```
cluster-admin@k8sadmin:~$ kubectl krew install rbac-tool
Updated the local copy of plugin index.
Installing plugin: rbac-tool
Installed plugin: rbac-tool
\
 | Use this plugin:
 |      kubectl rbac-tool
 | Documentation:
 |      https://github.com/alcideio/rbac-tool
/
WARNING: You installed plugin "rbac-tool" from the krew-index plugin repository.
   These plugins are not audited for security by the Krew maintainers.
   Run them at your own risk.
```

Figure 15. *Rbac Tool* installation via Krew.

RBAC Tool offers a command-line utility with a wide range of functionalities for Kubernetes RBAC tasks of different origin.



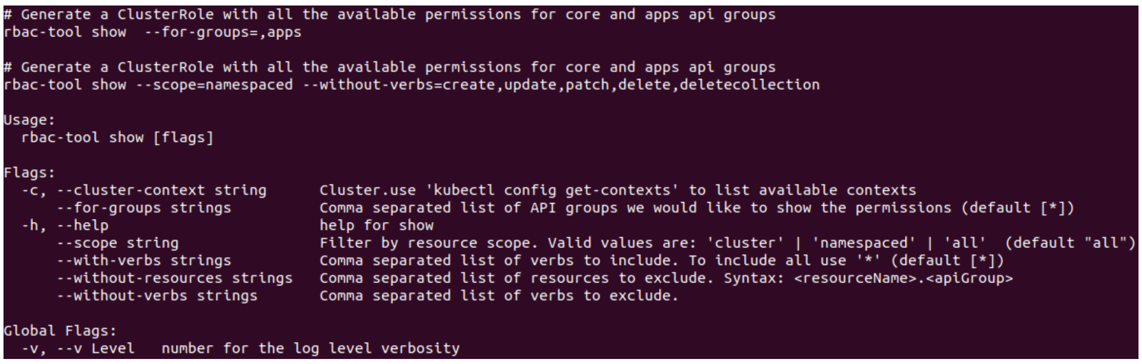
```
Available Commands:
  analysis        Analyze RBAC permissions and highlight overly permissive principals, risky permissions, etc.
  auditgen        Generate RBAC policy from Kubernetes audit events
  bash-completion Generate bash completion. source <(rbac-tool bash-completion)
  generate        Generate Role or ClusterRole and reduce the use of wildcards
  help            Help about any command
  lookup          RBAC Lookup by subject (user/group/serviceaccount) name
  policy-rules    RBAC List Policy Rules For subject (user/group/serviceaccount) name
  show            Generate ClusterRole with all available permissions from the target cluster
  version         Print rbac-tool version
  visualize       A RBAC visualizer
  who-can         Shows which subjects have RBAC permissions to perform an action
  whoami          Shows the subject for the current context with which one authenticates with the cluster
```

Figure 16. Available commands in *RBAC Tool*.

4.2.1 Command selection for analysis

While functionalities such as *rbac-tool lookup*, *rbac-tool viz*, *rbac-tool who-can*, etc., offer value, not all are within the scope of this work (see 1.3 Scope and target audience). The focus is primarily on the *rbac-tool show*, *rbac-tool gen*, and *rbac-tool auditgen*, which help in the creation of detailed RBAC configurations.

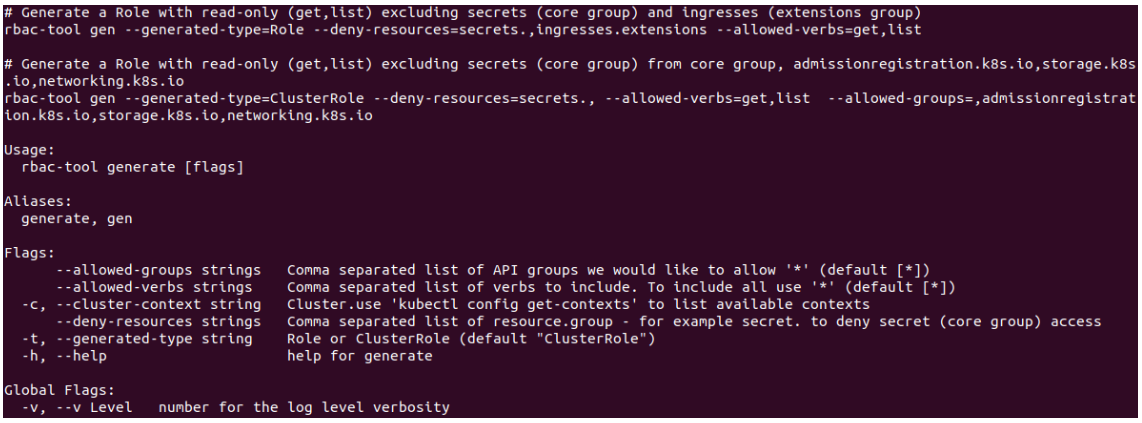
The *rbac-tool show* command generates a sample ClusterRole containing all available permissions from the target cluster. It retrieves available API groups and resource types using the Kubernetes discovery API.



```
# Generate a ClusterRole with all the available permissions for core and apps api groups
rbac-tool show  --for-groups=,apps

# Generate a ClusterRole with all the available permissions for core and apps api groups
rbac-tool show --scope=namespaced --without-verbs=create,update,patch,delete,deletecollection

Usage:
  rbac-tool show [flags]

Flags:
  -c, --cluster-context string      Cluster.use 'kubectl config get-contexts' to list available contexts
      --for-groups strings          Comma separated list of API groups we would like to show the permissions (default [*])
  -h, --help                        help for show
      --scope string                Filter by resource scope. Valid values are: 'cluster' | 'namespaced' | 'all'  (default "all")
      --with-verbs strings          Comma separated list of verbs to include. To include all use '*' (default [*])
      --without-resources strings   Comma separated list of resources to exclude. Syntax: <resourceName>.<apiGroup>
      --without-verbs strings       Comma separated list of verbs to exclude.

Global Flags:
  -v, --v Level   number for the log level verbosity
```

Figure 17. Usage options of *rbac-tool show*.

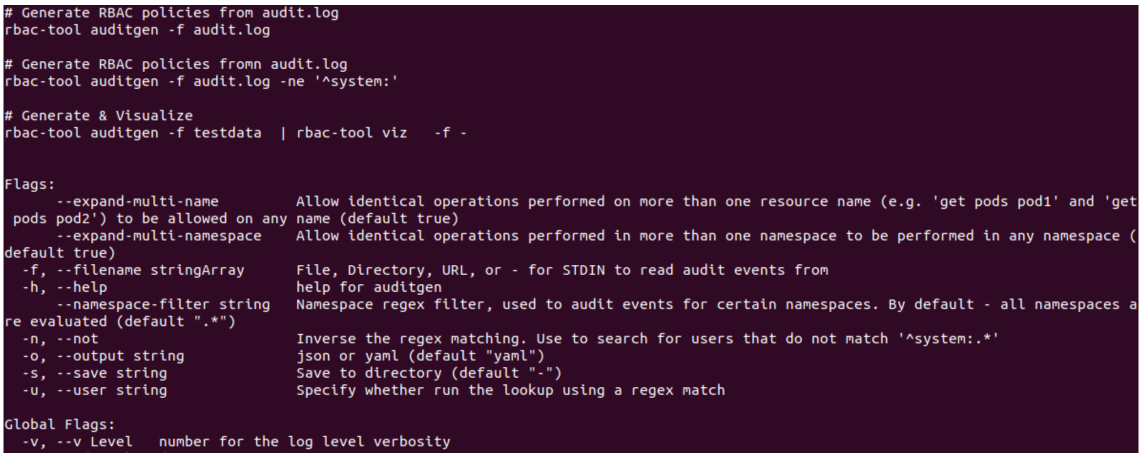
The *rbac-tool gen* command generates Role or ClusterRole objects, aiming to minimize wildcard usage through a variety of flag options.



```
# Generate a Role with read-only (get,list) excluding secrets (core group) and ingresses (extensions group)
rbac-tool gen --generated-type=Role --deny-resources=secrets.,ingresses.extensions --allowed-verbs=get,list

# Generate a Role with read-only (get,list) excluding secrets (core group) from core group, admissionregistration.k8s.io,storage.k8s
.io,networking.k8s.io
rbac-tool gen --generated-type=ClusterRole --deny-resources=secrets., --allowed-verbs=get,list  --allowed-groups=,admissionregistrat
ion.k8s.io,storage.k8s.io,networking.k8s.io

Usage:
  rbac-tool generate [flags]

Aliases:
  generate, gen

Flags:
      --allowed-groups strings   Comma separated list of API groups we would like to allow '*' (default [*])
      --allowed-verbs strings    Comma separated list of verbs to include. To include all use '*' (default [*])
  -c, --cluster-context string   Cluster.use 'kubectl config get-contexts' to list available contexts
      --deny-resources strings   Comma separated list of resource.group - for example secret. to deny secret (core group) access
  -t, --generated-type string    Role or ClusterRole (default "ClusterRole")
  -h, --help                     help for generate

Global Flags:
  -v, --v Level   number for the log level verbosity
```

Figure 18. Usage options of *rbac-tool gen*.

The *rbac-tool auditgen* command generates Role, ClusterRole, RoleBinding, and ClusterRoleBinding objects for different users and service accounts. It does this by analyzing a Kubernetes audit log file, which contains all the API requests made by these entities.



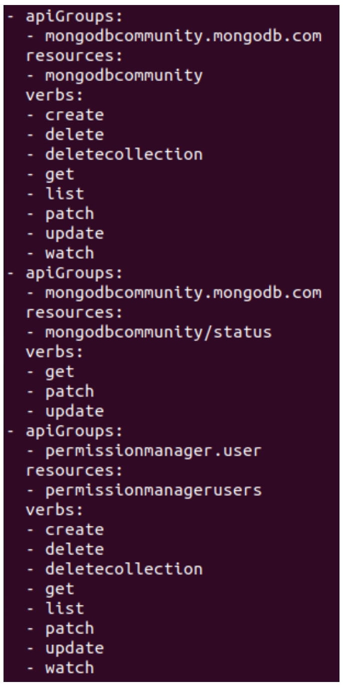
```
# Generate RBAC policies from audit.log
rbac-tool auditgen -f audit.log

# Generate RBAC policies fromn audit.log
rbac-tool auditgen -f audit.log -ne '^system:'

# Generate & Visualize
rbac-tool auditgen -f testdata  | rbac-tool viz   -f -


Flags:
      --expand-multi-name         Allow identical operations performed on more than one resource name (e.g. 'get pods pod1' and 'get
 pods pod2') to be allowed on any name (default true)
      --expand-multi-namespace    Allow identical operations performed in more than one namespace to be performed in any namespace (
default true)
  -f, --filename stringArray      File, Directory, URL, or - for STDIN to read audit events from
  -h, --help                      help for auditgen
      --namespace-filter string   Namespace regex filter, used to audit events for certain namespaces. By default - all namespaces a
re evaluated (default ".*")
  -n, --not                       Inverse the regex matching. Use to search for users that do not match '^system:.*'
  -o, --output string             json or yaml (default "yaml")
  -s, --save string               Save to directory (default "-")
  -u, --user string               Specify whether run the lookup using a regex match

Global Flags:
  -v, --v Level   number for the log level verbosity
```

Figure 19. Usage options of *rbac-tool auditgen*.

4.2.2 RBAC management capabilities

As mentioned above, *RBAC Tool* provides an extensive range of features to assist with different aspects of Kubernetes RBAC. However, the focus is shifted towards the *rbac-tool show*, *rbac-tool gen*, and *rbac-tool auditgen*.

The *rbac-tool show* displays all available API groups, including custom ones, associated resource types, and their verbs within a cluster, offering flexibility through optional flags. This ultimately saves the time needed to examine a cluster and review Kubernetes documentation when writing Role or ClusterRole configurations. Equally, the *rbac-tool gen* increases efficiency by streamlining the creation of these configurations.



```
- apiGroups:
  - mongodbcommunity.mongodb.com
  resources:
  - mongodbcommunity
  verbs:
  - create
  - delete
  - deletecollection
  - get
  - list
  - patch
  - update
  - watch
- apiGroups:
  - mongodbcommunity.mongodb.com
  resources:
  - mongodbcommunity/status
  verbs:
  - get
  - patch
  - update
- apiGroups:
  - permissionmanager.user
  resources:
  - permissionmanagerusers
  verbs:
  - create
  - delete
  - deletecollection
  - get
  - list
  - patch
  - update
  - watch
```

Figure 20. Third-party custom API groups can be discovered by *rbac-tool show*.

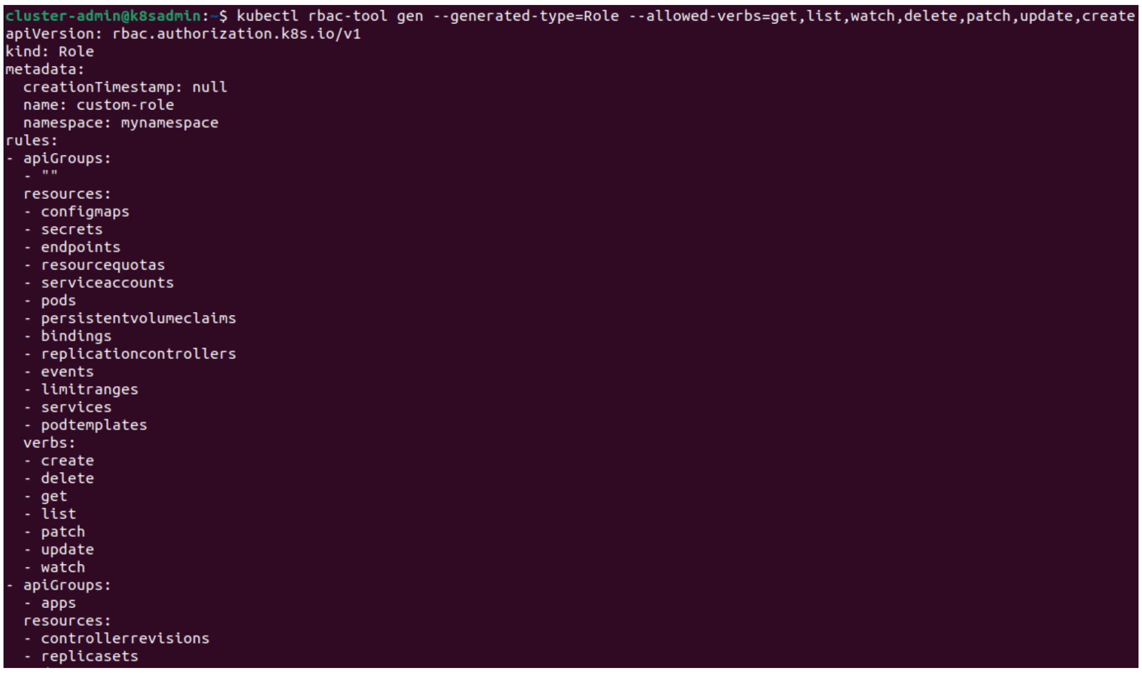
While the *rbac-tool auditgen* supports generation of all four kinds of RBAC objects, it is limited by the information provided in the audit log file. Generally, this means that none of the listed functions support optimized creation of RoleBinding and ClusterRoleBinding objects from the ground up.

4.2.3 Adherence to the principle of least privilege

The *rbac-tool show* and *rbac-tool gen* complement each other, aiming to provide only the necessary permissions and avoid wildcards in configurations. On the other hand, the *rbac-tool auditgen* requires the audit log, implying that the cluster should have been operational for a defined period, either without the principle of least privilege or with only partial implementation of it. Additionally, if the principle of least privilege has not been applied, this means that the command might generate Role and ClusterRole objects with excessive permissions based on logged overprivileged actions. These would then need to be manually tracked and reconfigured.

4.2.4 Ease of use for users with limited Kubernetes expertise

The command-line nature of *RBAC Tool* makes it more geared towards experienced users rather than those with limited Kubernetes expertise. While the *rbac-tool show* offers a straightforward display of possible permissions within a cluster, retrieving and utilizing this information effectively requires a knowledge of *kubectl* commands, RBAC concepts, and their syntax. Similarly, the *rbac-tool gen* demands the same level of knowledge to generate and apply Role and ClusterRole configurations, adhering to the principle of least privilege. Since the *rbac-tool gen* does not support flags for denying API groups and verbs, it can require typing lengthy commands. For example, instead of simply denying the use of the deletecollection verb, a user must specify all other verbs, such as create, get, list, watch, etc. In addition to the mentioned knowledge, using the *rbac-tool auditgen* requires an understanding of how to enable and retrieve cluster audit logs.



```
cluster-admin@k8sadmin:~$ kubectl rbac-tool gen --generated-type=Role --allowed-verbs=get,list,watch,delete,patch,update,create
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null
  name: custom-role
  namespace: mynamespace
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - secrets
  - endpoints
  - resourcequotas
  - serviceaccounts
  - pods
  - persistentvolumeclaims
  - bindings
  - replicationcontrollers
  - events
  - limitranges
  - services
  - podtemplates
  verbs:
  - create
  - delete
  - get
  - list
  - patch
  - update
  - watch
- apiGroups:
  - apps
  resources:
  - controllerrevisions
  - replicasets
```

Figure 21. Example: generating Role configuration without deletecollection verb.

5 Prototype development

The development of the prototype is based on requirements, which are a direct response to the limitations identified in *Permission Manager* and *RBAC Tool*. The design of the prototype takes shape of a software with web user interface, which is operated locally, outside of a cluster, providing an alternative to the examined tools.

5.1 Requirements

The prototype must meet the following requirements to address the limitations:

- Support in configuration of all four kinds of RBAC objects. An ability to apply, delete, and retrieve YAML configurations of desired objects more easily, without the usage of manual commands.
- Granular RBAC management. An ability to define Roles and ClusterRoles with precision, avoiding the use of wildcards and adhering to the least privilege principle.
- Informative user interface. An ability to operate the prototype with minimal Kubernetes RBAC knowledge by providing all relevant information for working with the objects and implementing UI elements that prevent misconfiguration by the users. It reduces the need for constant documentation reference and manual command execution within a cluster.

5.2 Components

The interface and underlying front-end code are split into two components. The components interact with the Kubernetes API via the client library, which is called in the backend code.

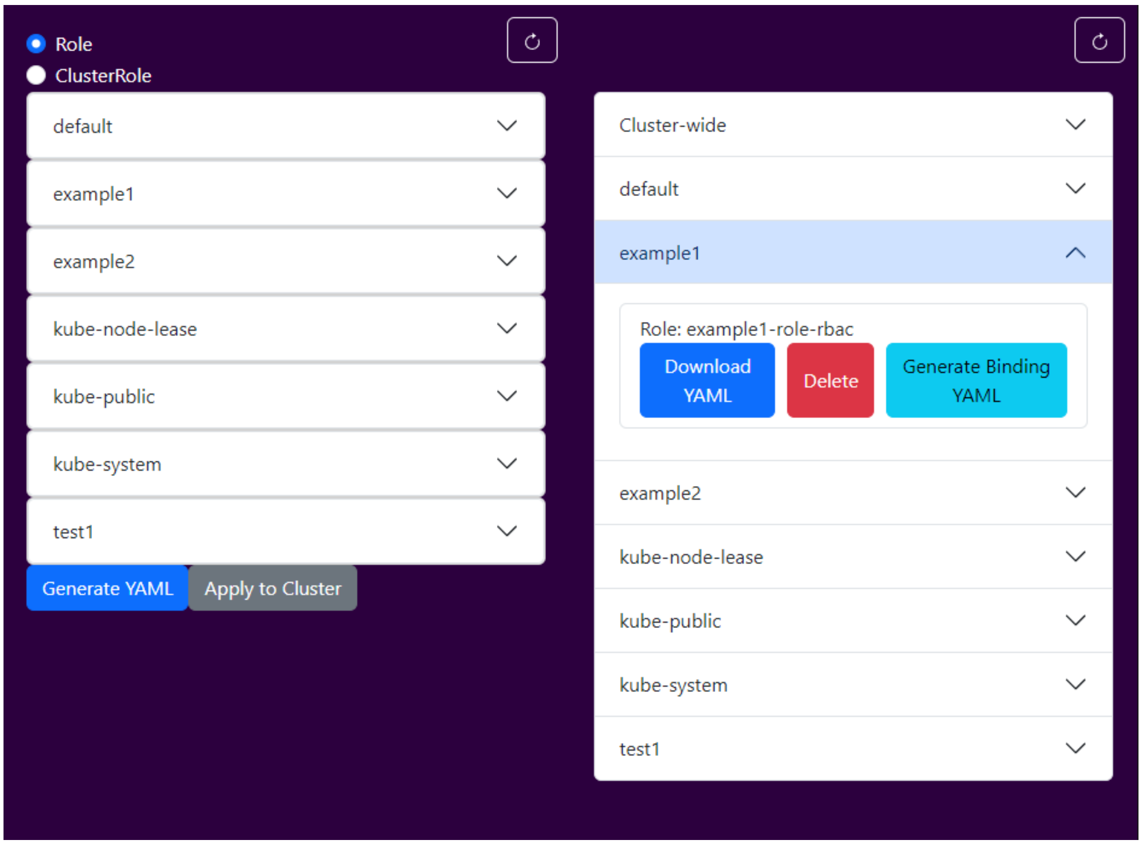


Figure 22. Developed prototype.

5.2.1 Role and ClusterRole generation

The first component of the prototype is a beginner-friendly interface that simplifies the generation of Role and ClusterRole configurations. First, the user must choose between a Role or a ClusterRole. Depending on this choice, the interface dynamically presents the user with a list that includes all available namespaces, API groups, and resource types, along with the checklists of their associated actions.

For example, selecting a Role displays all available namespaces within the cluster. Each namespace has an option to be expanded into a list of API groups, including custom ones provided by third-party tools via API extensions. Each group, in turn, can be expanded into a list of associated resource types. Finally, upon selecting a resource type, a checklist of available verbs for that resource type appears, allowing the user to specify permissions for it.

If a ClusterRole is chosen, the process remains similar, except that namespaces are not being displayed, because a ClusterRole is applied cluster-wide. Instead, the interface immediately shows available API groups.

Key specifications:

- The interface shows only the available cluster resources. For instance, verbs, such as create, delete, deletecollection, and patch cannot be selected for the *services/status* resource type, since it only supports the verbs get, patch, and update.
- The component allows users to download defined YAML configurations locally or apply them directly in a cluster instead of writing them manually.

Key specifications when selecting a Role:

- If an API group contains only non-namespaced resource types that are applied cluster-wide, it is not included in the list. The same applies to non-namespaced resource types within API groups that also have namespaced resource types. The interface makes it impossible to define a cluster-wide resource type for a Role.
- When verbs for resource types within one namespace are selected, the checkboxes for verbs from different namespaces become grayed out, reflecting the fact that a Role is applied only in a single namespace.

The component prevents possible misconfigurations, increases efficiency by saving time that would otherwise be spent on documentation, manual cluster examination, and configuration writing, and increases the effectiveness of granularity in the resulting Roles and ClusterRoles.

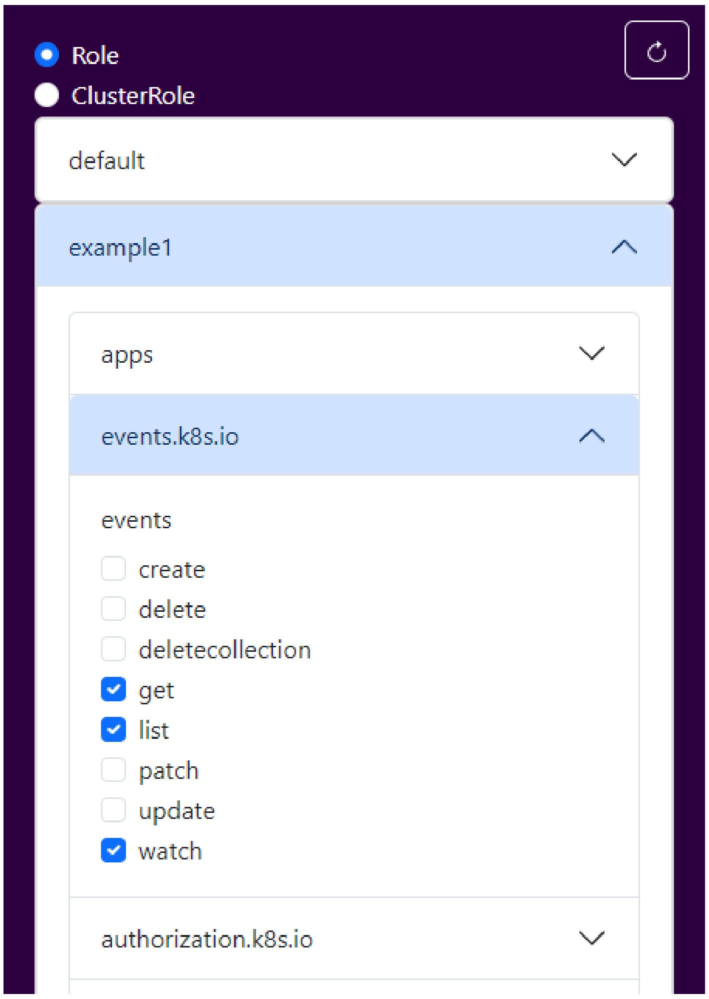


Figure 23. Role and ClusterRole generation component.

5.2.2 RBAC object configuration

The second component of the prototype is an organizational list, which offers an overview of all RBAC objects within the Kubernetes cluster. It provides the functionality to download the YAML configurations of these objects, or to delete the objects directly. The objects are logically separated based on their scope: they are either namespaced, in which case they belong to in-built lists of their respective namespaces, or they are cluster-wide, as in the case of ClusterRoles and ClusterRoleBindings.

Furthermore, the component includes a feature for generating RoleBinding and ClusterRoleBinding: when a Role or ClusterRole is selected from the list, the component gives the option to download the configuration of the corresponding RoleBinding or ClusterRoleBinding. The downloaded configurations have placeholder fields in the *subjects* section, allowing users to assign appropriate entities themselves.

The component enhances overall RBAC management by providing support for all kinds of RBAC object. It increases efficiency by offering visibility into a cluster and complementing the first component, which allows a user to create and delete objects using correct syntax without resorting to manually writing, applying, or deleting configurations and risking a potential misconfiguration.

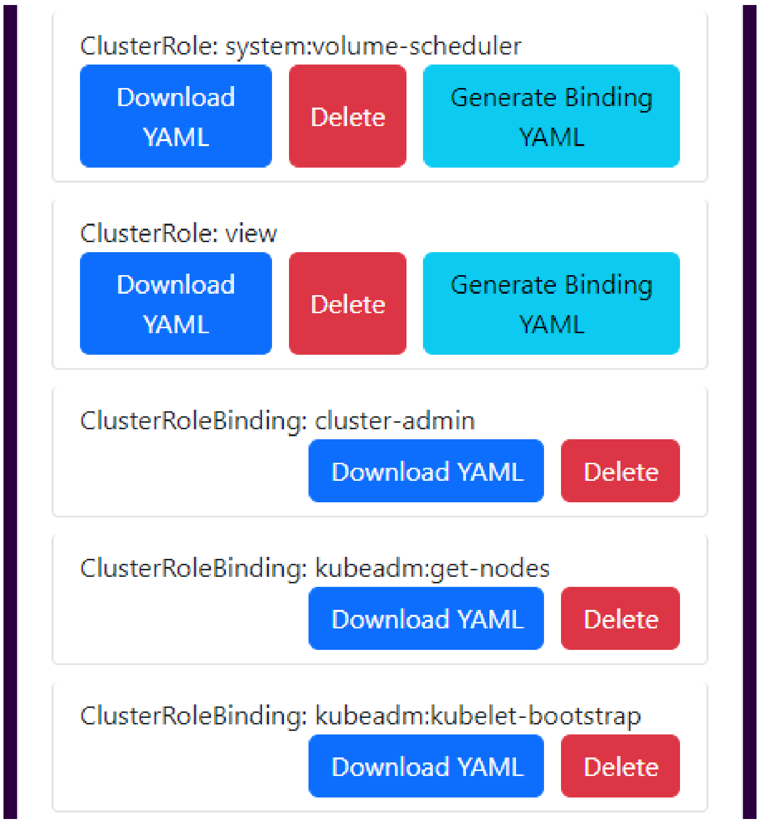
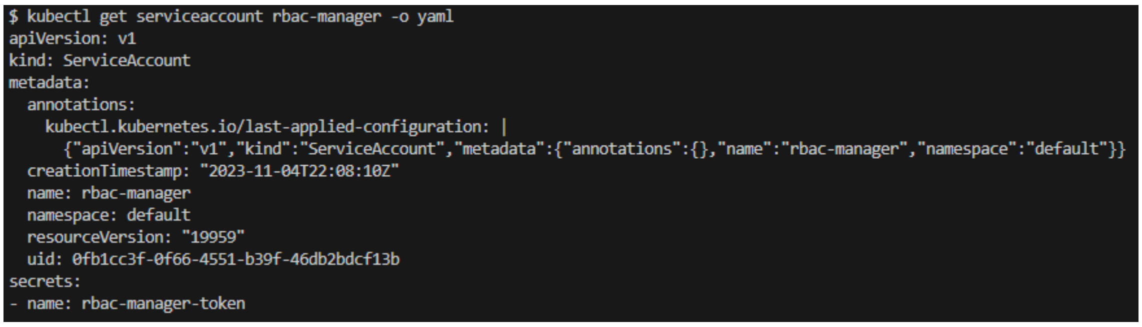


Figure 24. RBAC object configuration component. Example: cluster-wide RBAC objects.

5.3 Configuration

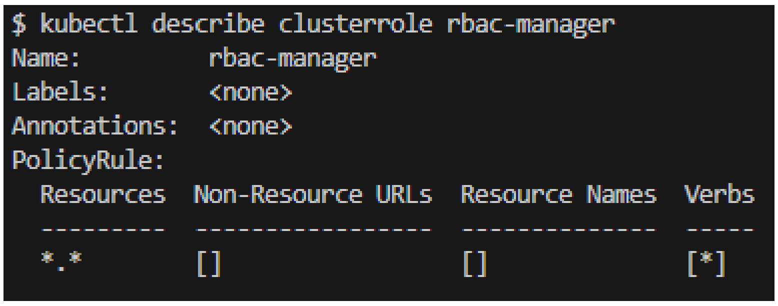
To configure and use the prototype, a service account is created in any namespace within the cluster.



```
$ kubectl get serviceaccount rbac-manager -o yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"ServiceAccount","metadata":{"annotations":{},"name":"rbac-manager","namespace":"default"}}
  creationTimestamp: "2023-11-04T22:08:10Z"
  name: rbac-manager
  namespace: default
  resourceVersion: "19959"
  uid: 0fb1cc3f-0f66-4551-b39f-46db2bdcf13b
secrets:
- name: rbac-manager-token
```

Figure 25. Prototype ServiceAccount YAML configuration.

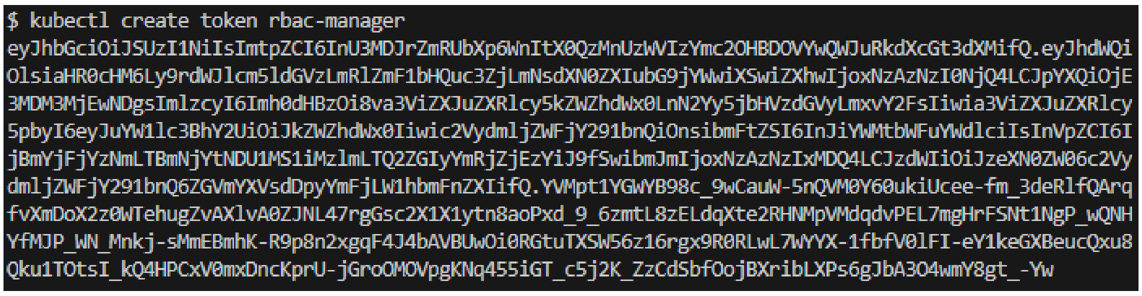
This service account is bound to a permissive ClusterRole via ClusterRoleBinding to allow for privileged actions invoked by API calls.



```
$ kubectl describe clusterrole rbac-manager
Name:         rbac-manager
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  *.*        []                 []              [*]
```

Figure 26. Prototype ClusterRole.

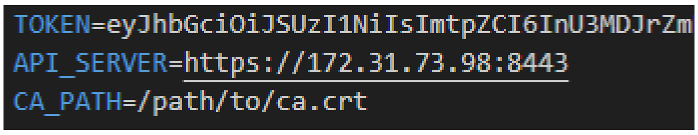
Then, either a short-lived or a long-lived API token is created for the service account. As shown in the figure below, a short-lived token is manually generated.



```
$ kubectl create token rbac-manager
eyJhbGciOiJSUzI1NiIsImtpZCI6InU3MDJrZmRUbXp6WnItX0QzMnUzWVIzYmc2OHBDOVYwQWJuRkdXcGt3dXMifQ.eyJhdWQi
OlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNzAzNzI0NjQ4LCJpYXQiOjE
3MDM3MjEwNDgsImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy
5pbyI6eyJuYW1lc3BhY2UiOiJkZWZhdWx0Iiwic2VydmljZWFjY291bnQiOnsibmFtZSI6InJiYWMtbWFuYWdlciIsInVpZCI6I
jBmYjFjYzNmLTBmNjYtNDU1MS1iMzlmLTQ2ZGIyYmRjZjEzYiJ9fSwibmJmIjoxNzAzNzIxMDQ4LCJzdWIiOiJzeXN0ZW06c2Vy
dmljZWFjY291bnQ6ZGVmYXVsdDpyYmFjLW1hbmFnZXIifQ.YVMpt1YGWYB98c_9wCauW-5nQVM0Y60ukiUcee-fm_3deRlfQArq
fvXmDoX2z0WTehugZvAXlvA0ZJNL47rgGsc2X1X1ytn8aoPxd_9_6zmtL8zELdqXte2RHNMpVMdqdvPEL7mgHrFSNt1NgP_wQNH
YfMJP_WN_Mnkj-sMmEBmhK-R9p8n2xgqF4J4bAVBUwOi0RGtuTXSW56z16rgx9R0RLwL7WYYX-1fbfV0lFI-eY1keGXBeucQxu8
Qku1TOtsI_kQ4HPCxV0mxDncKprU-jGroOMOVpgKNq455iGT_c5j2K_ZzCdSbfOojBXribLXPs6gJbA3O4wmY8gt_-Yw
```

Figure 27. Token generation for service account.

Finally, the cluster API server, the locally stored cluster CA certificate, and the token are declared in the *.env* file located in the prototype's project directory, and the code is executed using *npm* commands.



```
TOKEN=eyJhbGciOiJSUzI1NiIsImtpZCI6InU3MDJrZm
API_SERVER=https://172.31.73.98:8443
CA_PATH=/path/to/ca.crt
```

Figure 28. Environment variables in *.env* file.

6 Summary

This thesis successfully meets its intermediate research objectives: it examines and outlines the functionality of the modern open-source tools dedicated to the implementation and management of RBAC cluster objects and identifies potential areas for improvement via comparative study in a lab environment. To achieve the primary goal of the work, a software prototype was developed. This prototype incorporates functionalities derived from these identified areas, achieved through the introduction of specific requirements aimed at addressing the limitations of selected modern tools, namely *Permission Manager* and *RBAC Tool*.

The prototype successfully fetches the required data from the Kubernetes cluster and parses it for presentation in the UI. It offers improved functionalities and beginner-friendly management capabilities, split into two web UI components, which work robustly. The logic for deploying, manipulating live RBAC objects in the cluster, and downloading their configuration, as well as providing visibility, functions as intended without any issues.

Despite its simplicity, given its proof-of-concept status and limited timeframe for development, the prototype shows potential for more advanced and diverse features: the project would benefit from the inclusion of more cluster connectivity options, such as the ability to use a kubeconfig file instead of environment variables, which would also allow for easier support of multiple clusters simultaneously, in contrast to the current support of only one cluster. In addition, creating a Dockerfile to containerize the application would streamline its deployment, allowing it to run seamlessly on any local machine with a single command, rather than separately executing multiple commands for the back-end and front-end code. The ClusterRole created for the service account should be made less permissive to better adhere to the principle of least privilege.

Considering the functional aspects, the Role and ClusterRole generation component could be enhanced by adding an editable text box that synchronizes with the verb checklists. This would display the resulting YAML configuration, highlight any errors, including those related to available resources within a cluster (for example, signaling an error if a specified namespace in a Role does not exist), and automatically update the verb checklists to match any changes made manually in the text field, and vice versa. The addition would cater to users who prefer writing configurations directly or combining manual writing with UI selection, while still protecting them from misconfigurations. It would also allow for the import of configurations through insertions into the text box. Moreover, enabling users to see the immediate impact of their choices in the graphical interface on the YAML configuration would help them understand the structure and syntax of these configurations more effectively and efficiently than having to download a configuration as an additional step. This approach not only offers a more informative user interface but also enhances the learning opportunity.

Similarly, an editable text box that displays the YAML configurations of objects and highlights errors could be added to the RBAC object configuration component. On top of providing learning opportunities and catering to a diverse user base, this would enable users to edit configurations of existing objects more efficiently than the current process, which requires downloading configurations, editing them, and manually reapplying them with *kubectl* in a cluster.

Overall, the study of RBAC design in Kubernetes, the analysis of existing open-source solutions, and the proposed software prototype, along with its concepts and suggested further development, contribute to the field of Kubernetes security and administration. They can provide valuable insights for individuals already working with, or planning to work with, Kubernetes RBAC. By being both beginner-friendly and cost-free, and by striving to accommodate different needs, the prototype would benefit organizations of any size and level of Kubernetes expertise that are interested in strengthening their Kubernetes security through RBAC implementation.

References

[1] Cloud Native Computing Foundation, "Cloud Native Computing Foundation Annual Survey 2021," 10 February 2022. [Online]. Available: https://www.cncf.io/wp-content/uploads/2022/02/CNCF-AR_FINAL-edits-15.2.21.pdf. [Accessed 8 April 2023].

[2] International Business Machines Corporation (IBM), "What is containerization?," [Online]. Available: https://www.ibm.com/topics/containerization. [Accessed 9 November 2023].

[3] F. Hámori, "The History of Kubernetes on a Timeline," RisingStack, 26 June 2023. [Online]. Available: https://blog.risingstack.com/the-history-of-kubernetes/. [Accessed 24 November 2023].

[4] N. Poulton and P. Joglekar, The Kubernetes Book: 2023 Edition, Kindle Direct Publishing.

[5] B. Burns, "The History of Kubernetes & the Community Behind It," The Linux Foundation, 20 July 2018. [Online]. Available: https://kubernetes.io/blog/2018/07/20/the-history-of-kubernetes-the-community-behind-it/. [Accessed 5 April 2023].

[6] Cloud Native Computing Foundation, "Cloud Native Computing Foundation Annual Survey 2022," November 2022. [Online]. Available: https://www.cncf.io/reports/cncf-annual-survey-2022/. [Accessed 4 April 2023].

[7] Red Hat, Inc., "State of Kubernetes security report 2023," 13 April 2023. [Online]. Available: https://www.redhat.com/en/resources/state-kubernetes-security-report-2023. [Accessed 8 August 2023].

[8] The Linux Foundation, "Role Based Access Control Good Practices," 25 July 2023. [Online]. Available: https://kubernetes.io/docs/concepts/security/rbac-good-practices/. [Accessed 6 August 2023].

[9] The Linux Foundation, "Using RBAC Authorization," 24 August 2023. [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/rbac/. [Accessed 15 September 2023].

[10] The Linux Foundation, "Kubernetes API Concepts," 23 July 2023. [Online]. Available: https://kubernetes.io/docs/reference/using-api/api-concepts/. [Accessed 26 August 2023].

[11] The Linux Foundation, "Kubernetes Object Management," 8 January 2022. [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/object-management/. [Accessed 15 October 2023].

[12] The Linux Foundation, "Kubectl Commands," [Online]. Available: https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands. [Accessed 16 October 2023].

[13] The Linux Foundation, "Client Libraries," 20 November 2023. [Online]. Available: https://kubernetes.io/docs/reference/using-api/client-libraries/. [Accessed 24 November 2023].

[14] Rapid7, "rbac-tool," 26 September 2023. [Online]. Available: https://github.com/alcideio/rbac-tool. [Accessed 24 November 2023].

[15] SIGHUP s.r.l., "permission-manager," 27 March 2023. [Online]. Available: https://github.com/sighupio/permission-manager. [Accessed 24 November 2023].

[16] Canonical Ltd., "MicroK8s documentation - home," September 2023. [Online]. Available: https://microk8s.io/docs. [Accessed 1 October 2023].

[17] Canonical Ltd., "Canonical," 6 December 2023. [Online]. Available: https://canonical.com/. [Accessed 6 December 2023].

[18] Microsoft, "The TypeScript Handbook," 18 May 2023. [Online]. Available: https://www.typescriptlang.org/docs/handbook/intro.html. [Accessed 24 November 2023].

[19] OpenJS Foundation, "About Node.js®," 2023. [Online]. Available: https://nodejs.org/en/about/. [Accessed 24 November 2023].

[20] E. You, "Why Vite," 16 November 2023. [Online]. Available: https://vitejs.dev/guide/why.html. [Accessed 24 November 2023].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Valentin Kirjan

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Facilitation of Kubernetes Role-Based Access Control Implementation and Management with Open-Source Software" supervised by Aleksei Talisainen
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

04.01.2024

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.