Use Kubernetes service accounts to enable automated kubectl access

Introduction:

In Kubernetes, every API request must be authenticated and authorized. By default, administrators use their own kubeconfig files to authenticate with full privileges — but in many cases, we need to give limited access to specific users, applications, or automation systems.

This is achieved using Service Accounts and RBAC (Role-Based Access Control).

This project demonstrates how to:

- Create a ServiceAccount for limited access.
- Define a Role with specific permissions (to manage pods).
- Bind them with a RoleBinding.
- Create and use a custom kubeconfig that authenticates as that ServiceAccount.

The final result is a dedicated kubeconfig file that can be used to run kubectl commands as that service account with restricted permissions.

Objectives

The main goals of this project are:

- Understand ServiceAccounts and their role in Kubernetes security.
- Configure RBAC using Roles and RoleBindings.
- Generate a service account token and use it for authentication.
- Test permissions using a custom Kubeconfig file.

Key Concepts:

Service Account (SA):

A Service Account is a special type of account used by:

- Pods running inside the cluster (for in-cluster API calls), or
- External automation tools (CI/CD, scripts) to authenticate with the API server.

Unlike normal user accounts, service accounts are namespaced and managed by Kubernetes itself.

Each service account has:

- A name (e.g., myexample-sa)
- A namespace (e.g., myexample)
- An associated token used for authentication

RBAC (Role-Based Access Control)

RBAC defines who can do what within the cluster.

It uses 4 key Kubernetes objects:

1. **Role:** Defines permissions within a specific namespace.
   Ex: View pods, get services, etc.
2. **ClusterRole:** Defines cluster-wide permissions.
   Ex: View all pods in all namespaces.
3. **RoleBinding:** Grants a Role to a user or service account within one namespace.
   Ex: Bind Role to myexample-sa
4. **ClusterRoleBinding:** Grants a ClusterRole to a user or service account cluster-wide.
   Ex: Bind ClusterRole view to myexample-sa.

Example:

- **Role =** What actions are Allowed?
- **RoleBinding =** Who gets those permissions?

Tokens and Kubeconfig

When you create a ServiceAccount, Kubernetes issues a JWT token that allows API access.

By embedding this token in a Kubeconfig file, you can use kubectl commands as that service account — without administrator privileges.

This helps in:

- Running automation tasks securely.
- Delegating limited access to developers or CI/CD pipelines.
- Following the principle of least privilege.

Implementation Steps:

Step1: Create a Namespace

By creating a namespace called myexample, you ensure that your ServiceAccount and Roles are scoped — meaning, they can't accidentally affect other namespaces.

```
controlplane:~$ kubectl create namespace myexample
namespace/myexample created
controlplane:~$
controlplane:~$ kubectl get namespace
NAME                 STATUS   AGE
default              Active   20d
kube-node-lease      Active   20d
kube-public          Active   20d
kube-system          Active   20d
local-path-storage   Active   20d
myexample            Active   7s
controlplane:~$
```

Step2: Create a Service Account

A ServiceAccount represents a non-human user in Kubernetes. When created, Kubernetes associates it with a unique name and optionally a token (a JWT used for authentication).

```
controlplane:~$ vi serviceaccount.yaml
controlplane:~$ cat serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: myexample-sa
  namespace: myexample
controlplane:~$
```

Apply it

kubectl apply -f serviceaccount.yaml

Verify:

Kubectl -n myexample get sa

```
controlplane:~$ kubectl -n myexample get sa
NAME           SECRETS   AGE
default        0         8m10s
myexample-sa   0         7m39s
controlplane:~$
```

Step3: Create a Role

role.yaml

```
controlplane:~$ vi role.yaml
controlplane:~$ cat role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: myexample
  name: myexample-role
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list", "create"]
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods/exec"]
  verbs: ["create"]
controlplane:~$
```

Apply it

```
controlplane:~$ kubectl apply -f role.yaml
role.rbac.authorization.k8s.io/myexample-role created
controlplane:~$
```

A Role defines what actions (verbs) can be performed on which resources in a specific namespace.

Verify:

```
controlplane:~$ kubectl -n myexample get role
NAME             CREATED AT
myexample-role   2025-11-09T12:54:37Z
controlplane:~$
controlplane:~$
```

Step4: Create RoleBinding

rolebinding.yaml

```
controlplane:~$ vi rolebinding.yaml
controlplane:~$ cat rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: myexample-rb
  namespace: myexample
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: myexample-role
subjects:
- kind: ServiceAccount
  name: myexample-sa
  namespace: myexample
controlplane:~$
```

Apply it

```
controlplane:~$ kubectl apply -f rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/myexample-rb created
controlplane:~$
controlplane:~$
```

Verify

```
controlplane:~$ kubectl -n myexample get rolebinding
NAME           ROLE                  AGE
myexample-rb   Role/myexample-role   10m
controlplane:~$
controlplane:~$
```

Step5: Create Token for the Service Account

```
controlplane:~$ kubectl create token -n myexample myexample-sa
eyJhbGciOiJSUzI1NiIsImtpZCI6InN0YXdpc2xNS3NVU3ZqZUlVU3ZRNG00WjdaVDlrX1d6QjVWNzJrN1RuNjQifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZm
F1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNzYyNjkyNzQ5LCJpYXQiOjE3NjI2ODkxNDksImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHV
zdGVyLmxvY2FsIiwianRpIjoiNWYzZTJkMzItNDEyNy00NzMyLWFmODQtMjE2Yzg2N2Q0MGQzIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJteWV4YW1wbGUiLCJz
ZXJ2aWNlYWNjb3VudCI6eyJuYW1lIjoibXlleGFtcGxlLXNhIiwidWlkIjoiMDM3NTc1MGEtYmIwNy00MGE5LThlZDctYTI3OGYzZTc4MTE4In19LCJuYmYiOjE3NjI2ODkxN
DksInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpteWV4YW1wbGU6bXlleGFtcGxlLXNhIn0.P195qxxsTT0_BiNKGcECAyN462jmCO_lwCCO0Y4AdPaC3GfXpkUahWu3VP
l__K4pmV93Vtl4BqIGYOmGSLbfd47yJP2RM2VojtzO2jVYrnA4csq7JRXn9D0ojfz462wdkXaaga4Sg_b393OCTINn-DylVdUB10qiN-V2LKq6JPmQYWQRUYeJQpHgXfLZWRT
iXg6SaFaQBREhBLFoDHp5nZL3bG4313qMhLHq_r0GwtAYG9blS9TZzx9wri5aTJ-Qnu0NJEMyJw4_aTwY-WAeeJilWGjLhKnh0-eCcZWlTejy0ePefNrqv8uxamHqkHHTM7iH
VaT65411EpJmJJWAvg
controlplane:~$
```

Copy this token — it's your authentication key.

Step6: Get Cluster Server URL

```
controlplane:~$ kubectl cluster-info
Kubernetes control plane is running at https://172.30.1.2:6443
CoreDNS is running at https://172.30.1.2:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
controlplane:~$ _
```

Step7: Create Custom Kubeconfig

sa-kubeconfig.yaml

```
controlplane:~$ cat sa-kubeconfig.yaml
apiVersion: v1
clusters:
- cluster:
    insecure-skip-tls-verify: true
    server: https://172.30.1.2:6443
  name: 172.30.1.2:6443
contexts:
- context:
    cluster: 172.30.1.2:6443
    namespace: myexample
    user: myexample-sa
  name: myexample-context
current-context: myexample-context
kind: Config
preferences: {}
users:
- name: myexample-sa
  user:
    token: eyJhbGciOiJSUzI1NiIsImtpZCI6InN0YXdpc2xNS3NVU3ZqZUlVU3ZRNG00WjdaVDlrX1d6QjVWNzJrN1RuNjQifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNzYyNjkyNzQ5LCJpYXQiOjE3NjI2ODkxNDksImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwianRpIjoiNWYzZTJkMzItNDEyNy00NzMyLWFmODQtMjE2Yzg2N2Q0MGQzIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJteWV4YW1wbGUiLCJzZXJ2aWNlYWNjb3VudCI6eyJuYW1lIjoibXlleGFtcGxlLXNhIiwidWlkIjoiMDM3NTc1MGEtYmIwNy00MGE5LThlZDctYTI3OGYzZTc4MTE4In19LCJuYmYiOjE3NjI2ODkxNDksInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpteV4YW1wbGU6bXlleGFtcGxlLXNhIn0.P195qxxsTT0_BiNKGcECAyN462jmCO_lwCCO0Y4AdPaC3GfXpkUahWu3VPl__K4pmV93Vtl4BqIGYOmGSLbfd47yJP2RM2VojtzO2jVYrnA4csq7JRXn9D0ojfz462wdkXaaga4Sg_b393OCTINn-DylVdUB10qiN-V2LKq6JPmQYWQRUYeJQpHgXfLZWRTiXg6SaFaQBREhBLFoDHp5nZL3bG4313qMhLHq_r0GwtAYG9blS9TZzx9wri5aTJ-Qnu0NJEMyJw4_aTwY-WAeeJilWGjLhKnh0-eCcZWlTejy0ePefNrqv8uxamHqkHHTM7iHVaT65411EpJmJJWAvg
controlplane:~$
```

Replace the token value with your real token from Step 5.

Save and exit

Step8:Test the Kubeconfig

```
controlplane:~$ KUBECONFIG=/root/sa-kubeconfig.yaml kubectl config current-context
myexample-context
controlplane:~$
controlplane:~$
```

Lists Pods in the namespace

```
controlplane:~$ KUBECONFIG=/root/sa-kubeconfig.yaml kubectl get pods -n myexample
No resources found in myexample namespace.
controlplane:~$
controlplane:~$
```

Try creating a pod (you're allowed per your Role):

```
controlplane:~$ KUBECONFIG=/root/sa-kubeconfig.yaml kubectl run testpod --image=nginx -n myexample
pod/testpod created
controlplane:~$
controlplane:~$
```

You'll see

pod/testpod created

Step9: Validate Access Control

Check what this SA can do:

```
controlplane:~$ kubectl auth can-i list pods --as=system:serviceaccount:myexample:myexample-sa -n myexample
yes
controlplane:~$
controlplane:~$
```

Output: yes

Try deleting a pod (not permitted):

```
controlplane:~$ kubectl auth can-i delete pods --as=system:serviceaccount:myexample:myexample-sa -n myexample
no
controlplane:~$
controlplane:~$
```

Output: No

Advantages of Namespace-Scoped RBAC

- Least Privilege: Limits access to only needed actions.
- Namespace Isolation: Prevents cross-namespace access.
- Automation Safe: Ideal for scripts or CI/CD jobs.
- Auditable: Each API call is logged under the service account identity.

Conclusion:

This configuration provides a secure, minimal-permission access model for Kubernetes. By combining ServiceAccounts, Roles, and RoleBindings, administrators can precisely control what workloads or users can do within each namespace.

Finally, the custom Kubeconfig allows external tools and automation systems to access Kubernetes APIs safely.