Service Account

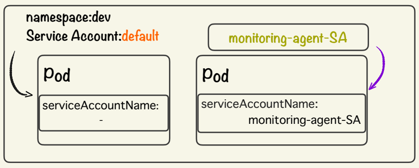
Service accounts in Kubernetes are non-human accounts that provide a unique identity for system components and application pods. These accounts are namespace-specific objects managed within the Kubernetes API server. By default, each Kubernetes namespace includes a service account called "default" which has no special roles or privileges assigned to it. In earlier versions of Kubernetes prior to 1.24, when a service account was created, an associated token would be automatically generated and mounted within the pod's file system. However, from Kubernetes 1.24 onwards, the automatic token generation has been discontinued, and tokens must be acquired through the TokenRequest API or by creating a Secret API object, allowing the token controller to populate it with a service account token.

```
kubectl create namespace <namespace_name>
```

automatically create a default service account

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
  namespace: < namespace_name >
```

Every kubernetes namespace has a default service account named default once being created



If a pod is created without specifying a service account, it will use the default ServiceAccount, default Service Account has limited permissions, but If you need to grant your pod more permissions, you can create a custom Service Account with the necessary roles and assign it to the pod.

Creating and Using a Service Account in a Kubernetes Pod.

1 Create a service account

```
kubectl create serviceaccount monitoring-agent-SA
```

in Kubernetes v1.24 and earlier, when a Service Account is created, a token secret is automatically generated and stored in the same namespace. This token secret is used for authentication and authorization purposes, However, in Kubernetes v1.25 and later, this automatic token creation has been removed. Instead, there are alternative methods for token creation and management. Here are some options:

Secret API object When you create a Secret with the annotation kubernetes.io/service-account.name and specify a ServiceAccount name, the token controller in k8s will automatically populate the Secret with a service account token associated with the referenced ServiceAccount. so you don't need to manually generate or provide the token for the Secret.The token controller takes care of generating and populating the token for you

```
apiVersion: v1
kind: Secret
metadata:
  name: monitoring-agent-SA-token
  annotations:
    kubernetes.io/service-account.name: monitoring-agent-SA
type: kubernetes.io/service-account-token
```

TokenRequest TokenRequest is an API resource in k8s that allows you to request a token for a specific Service Account. It offers a way to dynamically generate short-lived tokens for authentication and authorization purposes. The TokenRequest object has several important fields, with the audience field being one of them. The `audience` field specifies the intended recipient(s) or target audience for the requested token, defining the authorized users or services for token usage. Here are some examples of audiences that can be specified in the audience field

- rbac.authorization.k8s.io: For Role and ClusterRole operations
- metrics.k8s.io: For Metrics API access
- authentication.k8s.io: This specifies use by authentication methods and operators like kubelet
- storage.k8s.io: For Storage operations
- api: This specifies that the token is intended for use against the Kubernetes API server. API access given to service accounts is enforced by this audience

```
apiVersion: authentication.k8s.io/v1
kind: TokenRequest
metadata:
  name: monitoring-agent-SA-token
spec:
  audiences:
  - rbac.authorization.k8s.io
  serviceAccountName: monitoring-agent-SA
```

2 Grant permissions to the ServiceAccount

Create a Cluster Role that grants the necessary permissions for SA and Create a Role Binding that associates the Service Account with the Cluster Role

Code creates a ClusterRole that grants permissions to retrieve and list information about pods and nodes in the k8s cluster using the "get", "list", and "watch" verbs

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: monitoring-agent-role
rules:
- apiGroups: [""]
  resources: ["pods", "nodes"]
  verbs: ["get", "list", "watch"]
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: monitoring-agent-role-binding
subjects:
- kind: ServiceAccount
  name: monitoring-agent-SA
roleRef:
  kind: ClusterRole
  name: monitoring-agent-role
  apiGroup: rbac.authorization.k8s.io
```

To check the permissions of a service account in Kubernetes, execute the following command to list all the available permissions granted to the monitoring-agent-SA Service Account in the default namespace.

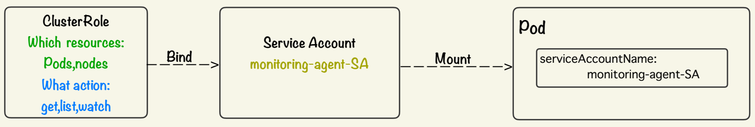
```
kubectl auth can-i --list --as=system:serviceaccount:default:monitoring-agent-SA
```

The --list flag is used to list all the actions and resources that the service account has access to, and the --as flag is used to specify the service account to check the permissions for.

3 Mount the service account token into a pod

When you specify the serviceAccountName field in the Pod spec, Kubernetes mounts the secret containing the Service Account token as a volume in the Pod.

The volume is mounted at /var/run/secrets/kubernetes.io/serviceaccount, and the Service Account token is stored in the token file inside this volume



```
apiVersion: v1
kind: Pod
metadata:
  name: monitoring-agent
spec:
  serviceAccountName: monitoring-agent-SA
  containers:
  - name: monitoring-agent
    image: monitoring-agent-image
    args: ["--kubeconfig=/var/run/secrets/kubernetes.io/serviceaccount/token"]
    volumeMounts:
    - name: sa-token
      mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      readOnly: true
  volumes:
  - name: sa-token
    secret:
      secretName: monitoring-agent-SA-token-xxxxx
```

Using a Service Account to Access a Kubernetes Cluster with kubectl

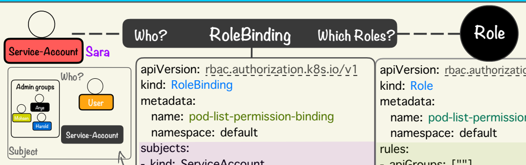
1 Create & Grant permissions to the service account

```
kubectl create serviceaccount sara
```

```
kubectl create role pod-list-permission --verb=get,list,watch --resource=pods --namespace=default
```

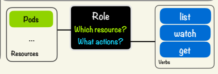
```
kubectl create rolebinding pod-list-permission-binding --role=pod-list-permission --user=sara --namespace=default
```

this command creates a role binding in the default namespace that binds the pod-list-permission role to the user sara



```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-list-permission-binding
  namespace: default
subjects:
- kind: ServiceAccount
  name: sara
  namespace: default
roleRef:
  kind: Role
  name: pod-list-permission
  apiGroup: rbac.authorization.k8s.io
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-list-permission
  namespace: default
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```



2 Retrieve the Service Account token

You can retrieve the Service Account token or recreate it by running the following commands:

```
Kubectl -n default create token sara
eyJhbGciOiJSUzI1NiIsIm...Lz9APOb2rsWHr9HWA
```

3 Set the token as a credential in kubectl

To add the user sara to this .kube/config file, you would need to add the following code configuration under the users section

```
- name: sara
  user:
    token: eyJhbGciOiJSUzI1NiIsIm...Lz9APOb2rsWHr9HWA
```

After adding this configuration, you would then need to create a new context that uses the sara user and the k8s-cluster-1 cluster

```
- context:
    cluster: k8s-cluster-1
    user: sara
  name: sara-k8s-cluster-1
```

Finally, set the current-context field to the newly created context name

```
current-context: sara-k8s-cluster-1
```

This configuration sets the authentication method for the user sara to use a bearer token (token field) instead of the client certificate and client key used by the arye user

.kube/config file

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBD...RVURS0tLS0tCg==
    server: https://127.0.0.1:42995
  name: k8s-cluster-1
contexts:
- context:
    cluster: k8s-cluster-1
    user: sara
  name: sara-k8s-cluster-1
- context:
    cluster: k8s-cluster-1
    user: arye
  name: arye@k8s-cluster-1
current-context: sara-k8s-cluster-1
kind: Config
preferences: {}
users:
- name: arye
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRQUR...S0tLS0tCg==
    client-key-data: LS0tLS1CRUdJTiBd0NibHFxS0t...LS0tCg==
- name: sara
  user:
    token: eyJhbGciOiJSUzI1NiIsIm...Lz9APOb2rsWHr9HWA
```

Using Service Accounts for authentication can be more secure than using user accounts because Service Accounts are automatically created and managed by Kubernetes

The "rules" section in a role specifies the permissions granted by the role. The "rules" section is an array of rules, where each rule specifies the resources and operations that are allowed

The "subjects" section specifies the user or group of users to which the role should be bound. A subject can be a user, a group, or a service account.

How to bind a role to multiple users?

```
...
subjects:
- kind: User
  name: mohsen
  apiGroup: rbac.authorization.k8s.io
- kind: User
  name: arye
  apiGroup: rbac.authorization.k8s.io
- kind: Group
  name: developers
  apiGroup: rbac.authorization.k8s.io
...
```

you can bind a role to multiple users by creating a role binding that specifies multiple users in the "subjects" section

How to specify multiple rules in a Role?

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-viewer
  namespace: my-namespace
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
- apiGroups: [""]
  resources: ["services"]
  verbs: ["get"]
```

You can also specify multiple rules in the "rules" section of a role