Exercising Caution with Service Accounts in Kubernetes

Service accounts are crucial for enabling applications and services running in your Kubernetes cluster to interact with the API server and perform various tasks. However, due to their privileged nature, it's essential to exercise caution in their usage to minimize security risks. Here's a step-by-step guide on how to achieve this:

1. Disable Default Service Accounts (if applicable):

- Some Kubernetes distributions create default service accounts for specific components like nodes or namespaces.
- These accounts might have broader permissions than necessary, so consider disabling them if you're not actively using them in your deployment.
- Consult your specific Kubernetes distribution documentation for instructions on disabling default service accounts.

2. Minimize Permissions on Newly Created Service Accounts:

- When creating new service accounts, assign them only the **minimum set of permissions** required for their specific tasks.
- Avoid granting broad permissions like "cluster-admin" unless absolutely necessary for specific administrative tasks.
- Utilize tools like kubeval to validate your service account RBAC configs before applying them.

3. Utilize RBAC for Granular Control:

- Leverage Role-Based Access Control (RBAC) to define specific roles and role bindings for service accounts.
- This allows you to grant fine-grained permissions based on the specific needs of each service account.
- Refer to the previous RBAC guide for detailed instructions on creating roles and role bindings.

4. Leverage Namespace Scoping:

- When applying RBAC permissions to service accounts, consider using namespaces to further restrict their access.
- This allows you to limit their access to resources within specific namespaces, enhancing security by preventing unauthorized access to resources in other namespaces.

5. Regularly Review and Update Service Account Permissions:

- Regularly audit and review service account permissions to ensure they remain aligned with the service's current needs.
- Remove unused or excessive permissions to minimize the attack surface and potential damage in case of compromise.

6. Use Secrets for Sensitive Information:

- Avoid storing sensitive information like passwords or tokens directly within service account definitions.
- Utilize Kubernetes Secrets to securely store sensitive data and reference them within your service account configurations using environment variables or volume mounts.

7. Monitor Service Account Activity (Optional):

- Consider implementing tools to monitor service account activity and identify any suspicious or unexpected behavior.
- This can help detect potential unauthorized access attempts or anomalies in service account usage.

Additional Considerations:

- Refer to the official Kubernetes documentation on service accounts for detailed information and best practices: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/

DEMO

```
kubectl get nodes
kubectl get namespaces
kubectl create namespace dev
kubectl get namespaces
kubectl get serviceaccounts -n dev
kubectl get secrets -n dev
kubectl run web --image  nginx -n dev
kubectl get pods -n dev
kubectl get pods -n dev -o yaml
# I have only given name, namespace, and image and all other are created by default (service account, service account name)
#volume mount is also default

kubectl exec -it web -n dev --sh
ls /var/run/secrets/kubernetes.io/serviceaccount
cat /var/run/secrets/kubernetes.io/serviceaccount/token
exit

kubectl get secret -n dev tokenname
kubectl describe secret -n dev tokenname

#we need to disable default
when creating serviceAccountLevel  we have to create and set
automountServiceAccountToken: false

#we will create service account and keep account token to false.
Kubectl create serviceaccount test-svc -n dev --dry-run=client -o yaml > svc.yaml
vim svc.yaml
#copy paste automountServiceAccountToken: false
kubectl apply -f svc.yaml
kubectl get serviceaccounts -n dev
```

kubectl get secret -n dev

#create a pod using this newly created service account

kubectl run web2 –image nginx -n dev --dry-run=client -o yaml < web2.yaml

vim web2.yaml

#under spec put serviceAccountName: test-svc

kubectl apply -f web2.yaml

kubectl get pod -n dev

kubectl get pod -n dev -w

kubectl get pod -n dev web2 -o yaml

#Create Role for service account

kubectl create role test-role -n dev --verb=get,list,watch,create --resource=pods --resource=pods,deployments,services,configmaps

Create RoleBinding for role & service account

kubectl create rolebinding cluster-test-binding -n dev --role=test-role --serviceaccount=dev:test-svc

-