ServiceAccount и RBAC в Kubernetes

Теоретическая и практическая часть

Что такое ServiceAccount

Факты:

- Учётная запись *для приложений* внутри кластера.
- Используется Pod’ами и контроллерами для обращения к Kubernetes API.
- Позволяет задать чёткие права.
- Каждый Pod по умолчанию получает `default` ServiceAccount своего namespace.

ServiceAccount содержит:

- имя и namespace
- токен для доступа к API
- привязки через RoleBinding/ClusterRoleBinding

Зачем нужны ServiceAccount

- Изоляция прав разных приложений.
- Ограничение доступа из Pod к API.
- Реализация принципа минимально достаточных прав.
- Audit логирование действий сервисов.
- Исключение использования персональных логинов разработчиков.

Что такое RBAC

RBAC — механизм контроля доступа в Kubernetes.

Определяет:

- **кто** выполняет действие
- **что** ему разрешено делать

Состоит из объектов:

- Role
- ClusterRole
- RoleBinding
- ClusterRoleBinding

Принцип:

1. Субъект делает запрос.
2. Kubernetes сверяет с RBAC.
3. Разрешает или блокирует действие.

Role vs ClusterRole

Role

- Действует **в пределах одного namespace**.
- Обычно для приложений и отдельных окружений.

ClusterRole

- Действует **на весь кластер**.
- Используется для cluster-wide ресурсов: nodes, namespaces, PV, CRD.

Что описывает Role / ClusterRole

Компоненты правила:

- **apiGroups** — группа API
- **resources** — ресурсы (`pods` , `deployments` , `secrets`)
- **verbs** — разрешённые действия

Примеры verbs:

- `get` , `list` , `watch`
- `create` , `update` , `patch`
- `delete`

RoleBinding и ClusterRoleBinding

RoleBinding

- Привязывает Role к субъекту.
- Работает внутри namespace.

ClusterRoleBinding

- Привязывает ClusterRole к субъекту на уровне всего кластера.

Субъекты:

- User
- Group
- ServiceAccount

Как работает связка ServiceAccount + RBAC

1. Pod запускается с ServiceAccount.
2. SA содержит токен для обращения в API.
3. RBAC проверяет разрешённые действия.
4. Pod может выполнять только то, что указано в Role/ClusterRole.

Практика: Создаём ServiceAccount

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app-sa
  namespace: demo
```

Результат:

- создаётся SA
- создаётся токен (или временный API токен)
- Pod может ссылаться на него через `serviceAccountName`

Практика: Создаём Role

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

Назначение: дать минимальные права только на чтение Pod’ов.

Практика: RoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: demo
subjects:
- kind: ServiceAccount
  name: app-sa
  namespace: demo
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Результат:

- ServiceAccount получает права на чтение Pod’ов.

Использование ServiceAccount в Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: demo
spec:
  serviceAccountName: app-sa
  containers:
  - name: nginx
    image: nginx
```

Pod работает от имени `app-sa`.

Важные нюансы

- SA не наследует права автоматически.
- Несколько RoleBinding могут суммировать права.
- Без разрешённых verbs — доступ запрещён.
- Доступ к Secret — критичный и должен ограничиваться.
- ClusterRoleBinding даёт слишком широкие права.

Типичные ошибки

- Использование ClusterRoleBinding вместо RoleBinding.
- Назначение `*` в verbs/resources → полный доступ.
- Запуск всех Pod'ов под `default` SA.
- Лишний доступ к Secret.