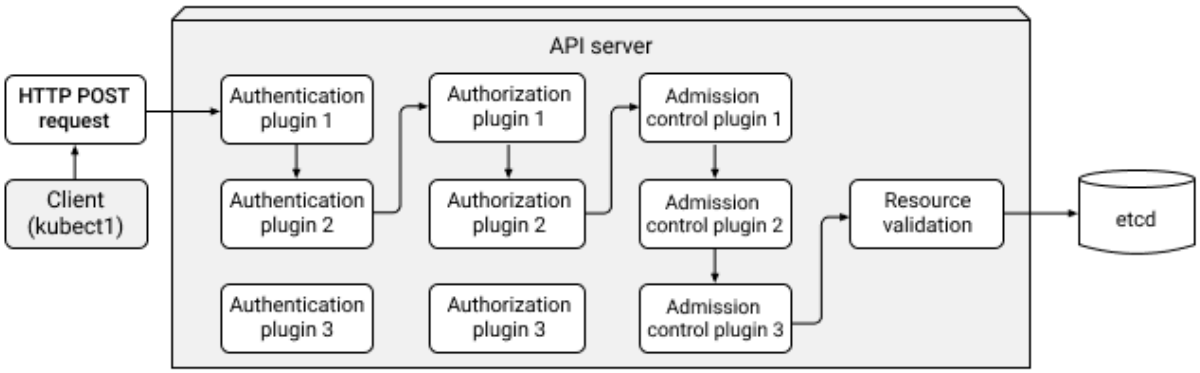
KUB 26: Role Based Access Control в Kubernetes

Описание:

Для доступа к ресурсам kubernetes задействуется три шага:

- authentication — осуществляет логин для пользователя;
- authorization — авторизует доступ к ресурсу;
- admission control — может изменить или проверить объект на основе дополнительных правил (например, таких, как квоты).



Authentication

В kubernetes нет таких объектов, как пользователь или пароль. Но и без этого kubernetes может использовать имена пользователя для доступа к объектам. Есть два типа пользователей: люди и служебные пользователи.

- Люди — это пользователи вне kubernetes, которые получают к нему доступ, авторизуясь через сертификат, куку и подобное.
- Служебные пользователи — Service Accounts — это пользователи внутри кластера. Большинство из них создается автоматически при разворачивании кластера, но мы можем сделать дополнительные в каждом namespace. При определенной настройке kubernetes может обслуживать запросы и от анонимных пользователей.

Типы авторизации для внешних пользователей (могут активироваться при помощи передачи параметров запуска к Kubernetes API Server):

- client certificate — для включения авторизации по сертификатам нужно прибавить в параметры запуска API сервера --client-ca-file=FILE. CA в этом файле будут валидировать клиентские сертификаты.
- static token file — для включения авторизации по токенам нужно прибавить в параметры запуска API сервера --token-auth-file=FILE. В этом файле будут

записаны токены и их нельзя будет изменить без перезапуска API сервера. Также срок их действия не будет ограничен.

- bootstrap token — используется для установки нового кластера, токен генерируется автоматически.
- static password file — --basic-auth-file=SOMEFILE, содержит пользователей и пароли. Изменить без рестарта API сервера нельзя, срок действия не ограничен.
- service account token — это автоматически генерируемый токен для служебного пользователя. Он добавляется к pod через serviceaccount admissioncontroller и позволяет взаимодействовать с kubernetes изнутри pod.
- openid connect token — oauth2 авторизация, примеры — ActiveDirectory, Google, Facebook. Переносит процесс авторизации на внешний сервис.
- webhook token auth — переносит верификацию токенов на внешний сервис.
- keystone password: --experimental-keystone-url=<AuthURL> -- авторизация через альфа-версию keystone.
- auth proxy — авторизация через прокси, который реализует свою дополнительную логику.

Одновременно можно включить несколько видов авторизации. После прохождения аутентификации можно взаимодействовать с API kubernetes. В дело вступают авторизаторы.

Authorization

Это модули, которые запрещают или позволяют определенное действие. Примеры:

- Node Authorize — специальный авторизатор. Служит для проверки запросов, отправленных с kubelet рабочих узлов (https://kubernetes.io/docs/reference/access-authn-authz/node/).
- ABAC (Attribute based control authorizer) — контроль доступа на основе атрибутов. Устаревший авторизатор, редко используется. Включается через --authorization-mode=ABAC --authorization-policy-file=PolicyFile.json. Пример:

```
{
  "apiVersion": "abac.authorization.kubernetes.io/v1beta1",
  "kind": "Policy",
  "spec": {
    "user": "nkhare",
    "namespace": "lfs158",
    "resource": "pods",
    "readonly": true
  }
}
```

-

- WebHookAuthorizer — авторизатор через веб-хук. Делает запрос на внешний сервис. Очень простой, принимает ответ только true или false. Включается --authorization-webhook-config-file=SOME_FILENAME.
- RBAC (Role based authorizer) — доступ с разделением по ролям. В данный момент — основной используемый авторизатор по умолчанию. Включается через --authorization-mode=RBAC.

Admission control

Эти опции позволяет обрабатывать запросы, когда они, казалось бы, прошли этап аутентификации и авторизации. Как правило, они используются для проверки.

Есть 2 типа admission controller:

- валидирующие (validating) — подтверждают, что переданные данные верны и валидны;
- изменяющие (mutating) — изменяют входные данные.

На самом деле, любой контроллер может выполнять обе роли, если это требуется. И, соответственно, одновременно может быть включено несколько контроллеров, и если на любом из них запрос не проходит проверку, то весь запрос считается не валидным.

Для включения конкретных Admission Controller требуется передать параметр --enable-admission-plugins=X,Y,.. в параметры Kubernetes API Server, где X, Y — это имена требуемых контроллеров, и разделяться они должны запятой без пробелов до и после. Для отключения же используется опция --disable-admission-plugins с таким же синтаксисом.

Если говорить о примерах, то LimitRanger отвечает за то, чтобы объекты типа LimitRange обрабатывались, а PersistentVolumeClaimResize отвечает за возможность расширять PVC, как мы обсуждали ранее.

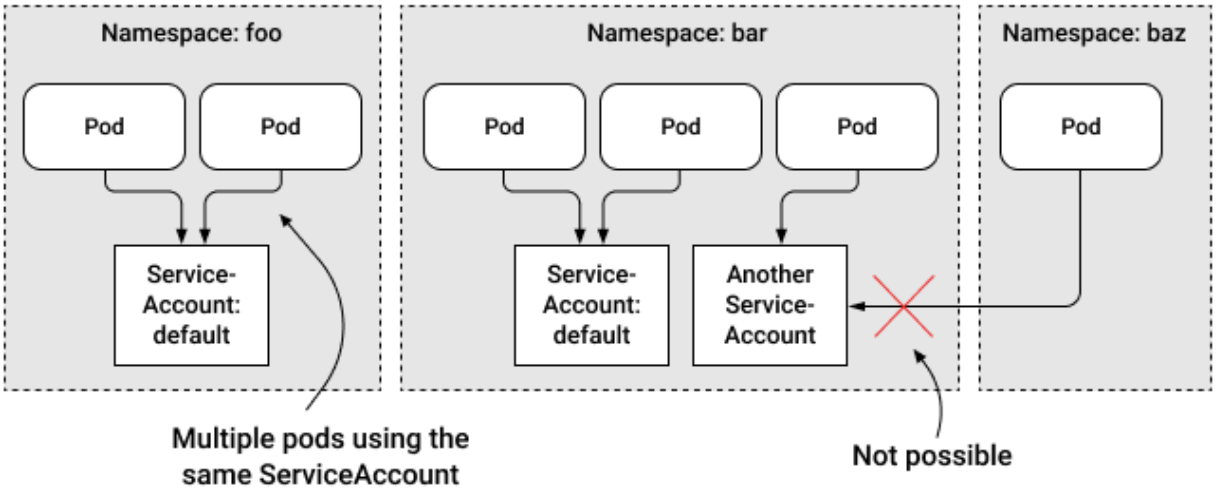
Когда мы получаем доступ к кластеру (например, с помощью kubectl), мы аутентифицируемся через apiserver с помощью определенной учетной записи пользователя (User Account), которой соответствует определенный набор прав на взаимодействие с кластером. Процессы в контейнерах также могут связываться с сервером для взаимодействия с кластером. Когда они это делают, они аутентифицируются под определенной сервисной учетной записью (Service Account, SA).

При создании pod, если вы не указали конкретный Service Account, ему будет присвоен 'default' service account, который по умолчанию создается в каждом namespace. Если же получить описание созданного пода, то мы увидим, что поле ServiceAccount будет содержать 'default' (например, с помощью kubectl get po <pod's-name> -o yaml).

При этом каждому SA соответствует токен доступа, который создается одновременно с SA и хранится в соответствующем секрете (в случае необходимости, его можно предварительно создать вручную).

RBAC — это предоставление определенных прав соответствующим аккаунтам (User Account, Service Account). Если описывать простым языком, то в K8S существуют поды, которым соответствуют определенные SA. Этим SA может соответствовать описание разрешенных действий (к примеру, create/get/update/patch/delete) над рядом сущностей (это могут быть другие pods/rs/deployments или определенные объекты, или процессы, как pod/exec, и т.д.).

Для начала рассмотрим объект ServiceAccount. Это внутренний объект kubernetes, формируемый по маске system:serviceaccount::. API-сервер передает это имя пользователя плагинам авторизации, которые сопоставляют его с группой, получая список доступных прав. По умолчанию уже есть несколько serviceaccount. SA default существует в каждом namespace kubectl get sa.



Этот аккаунт автоматически монтируется в каждый pod /var/run/secrets/kubernetes.io/serviceaccount/token. Внутри пода можно делать запросы к API server с этим токеном. Однако данный аккаунт ограничен своим namespace, получить информацию о ресурсах другого namespace под ним невозможно.
Есть два типа ролей: Role — указанный класс описывает права только в рамках указанного namespace. ClusterRole — описывает права в рамках всего кластера. Сначала разберемся с Role. Посмотрим пример описания роли:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: lfs158
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

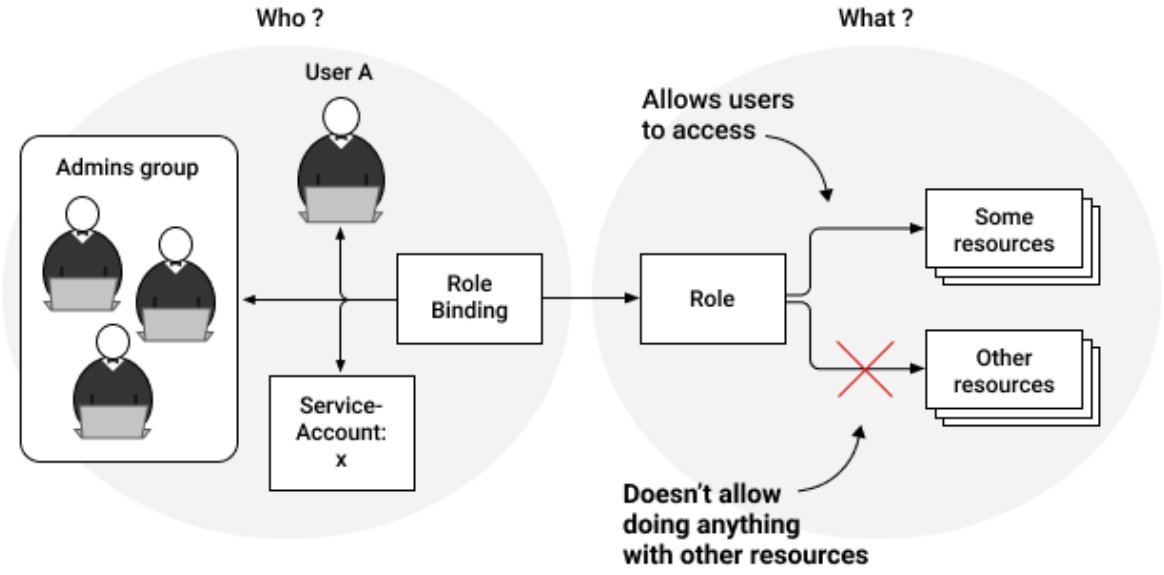
Это описание создаст роль в namespace lfs158, и в нем разрешается доступ к операциям get/watch/list над ресурсами pods в api-group 'core'.
Кластерная роль описывается похожим образом, но не указывает namespace, так как распространяется на весь кластер.

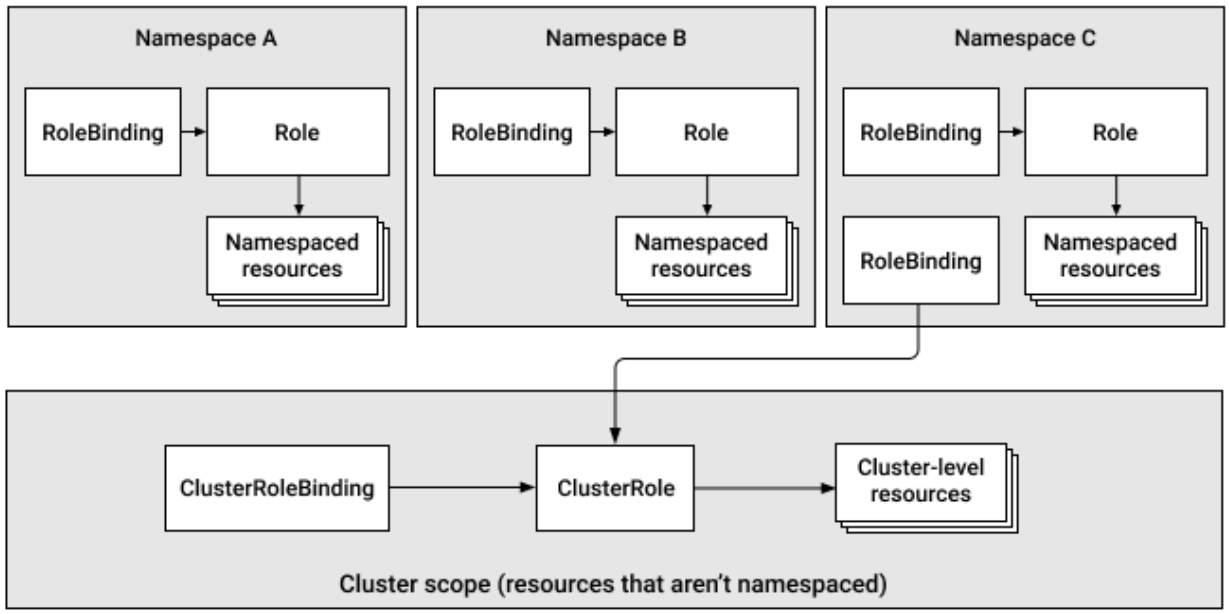
```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
  # "namespace" comitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

HTTP method	Verb for sibgle resource	Verb for collection
GET, HEAD	get (and watch for watching)	list (and watch)
POST	create	n/a
PUT	update	n/a
PATCH	patch	n/a
DELETE	delete	deletecollection

Роль прикрепляется при помощи RoleBindindg и ClusterRoleBinding. Отличия понятны из названия. В первом случае связывание роли и SA происходит в рамках одного namespace. Во втором — в рамках кластера.

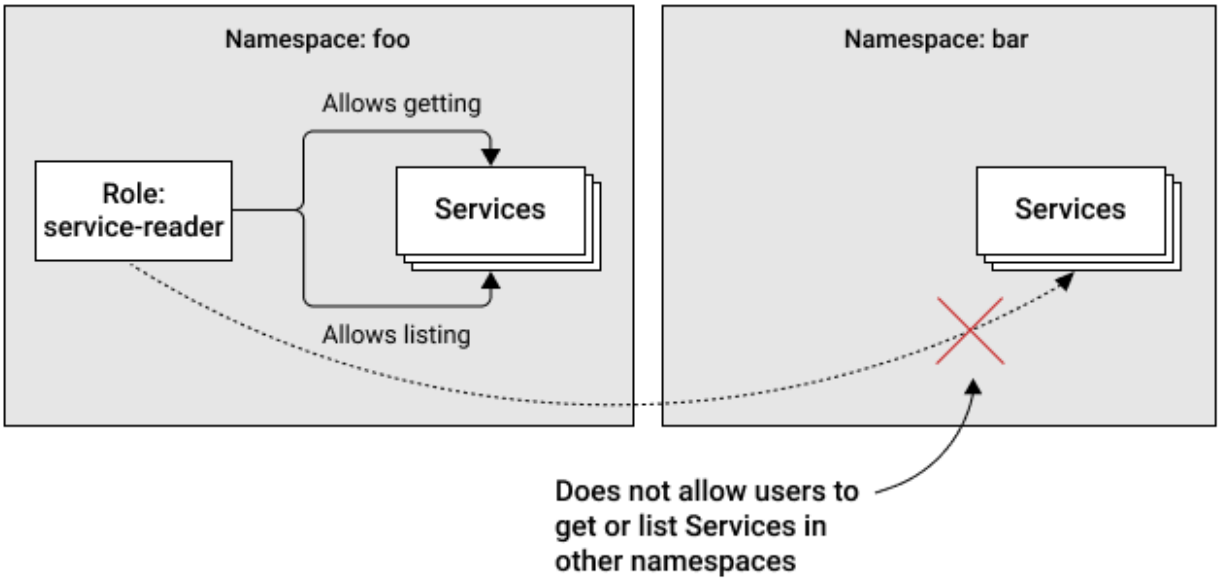




Пример привязки RoleBinding:

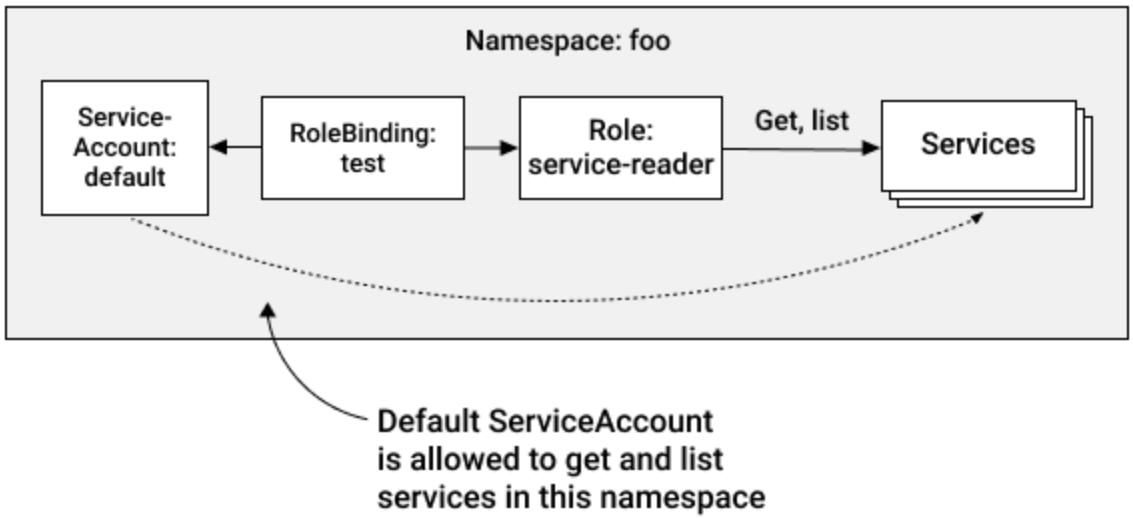
```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: foo
  name: service-reader
rules:
- apiGroups: [""]
  verbs: ["get", "list"]
  resources: ["services"]
```

Эта роль будет создана в namespace foo и разрешит только два действия с сервисами в нем.



Для привязки роли к аккаунту:

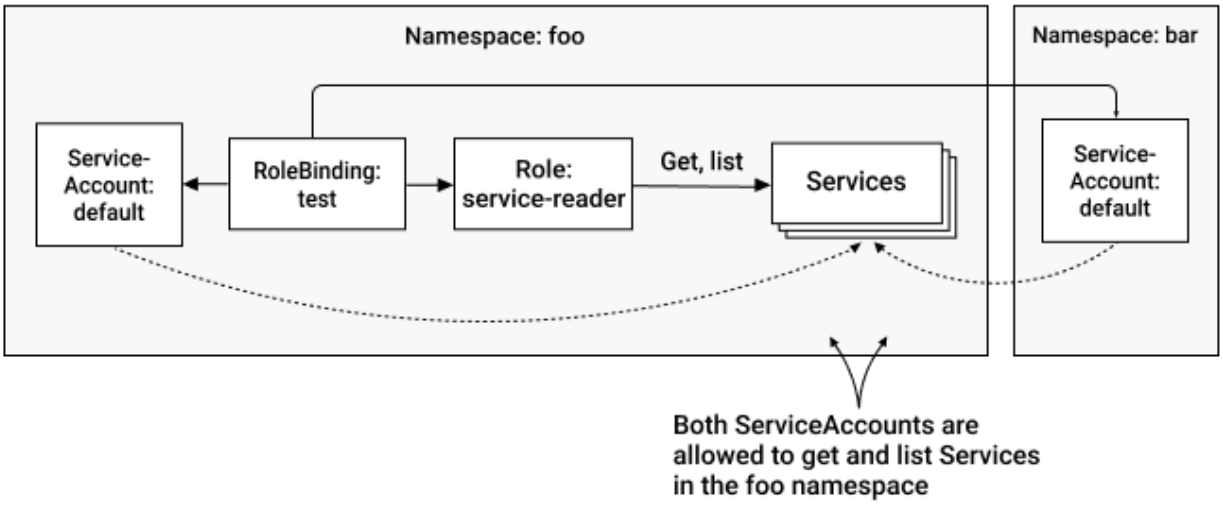
```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test
  namespace: foo
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: service-reader
subjects:
- kind: ServiceAccount
  name: default
  namespace: foo
```



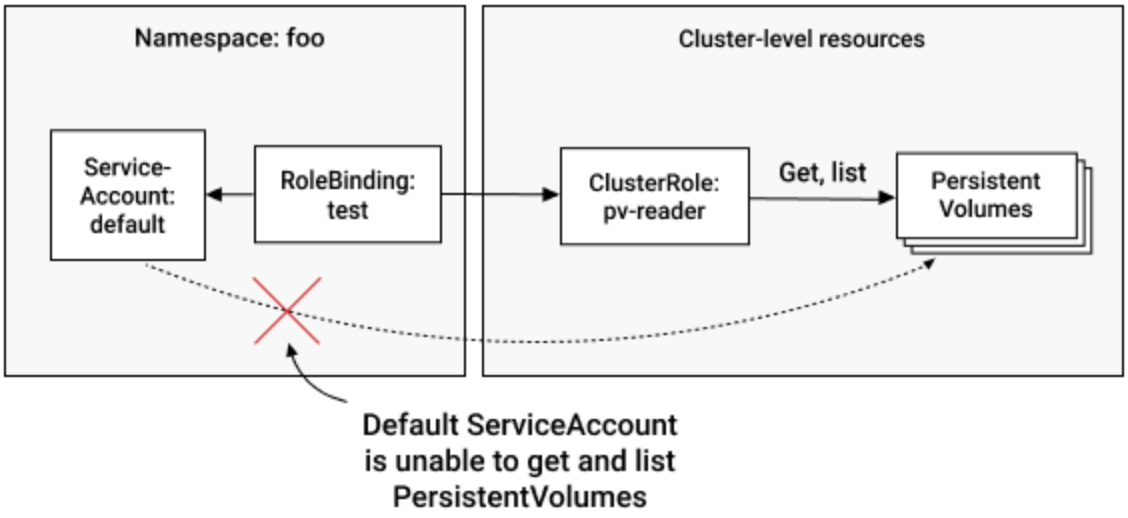
Также роль можно привязать к аккаунту из другого namespace:

```
subjects:
- kind: ServiceAccount
  name: default
  namespace: bar
```

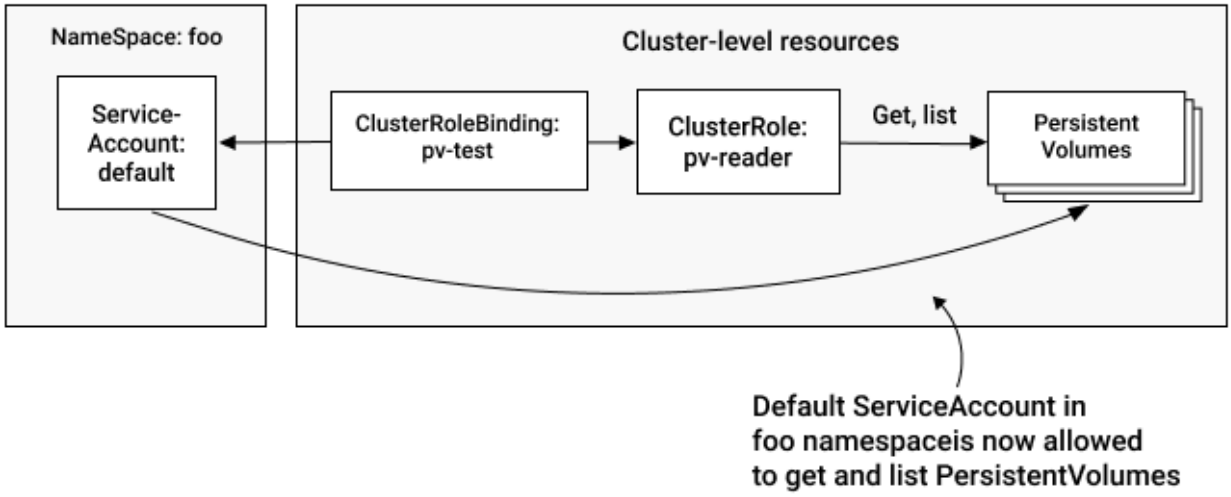
В этом случае sa default из ns bar получит возможность читать сервисы ns foo.



Похожим образом создаются ClusterRoleBinding.



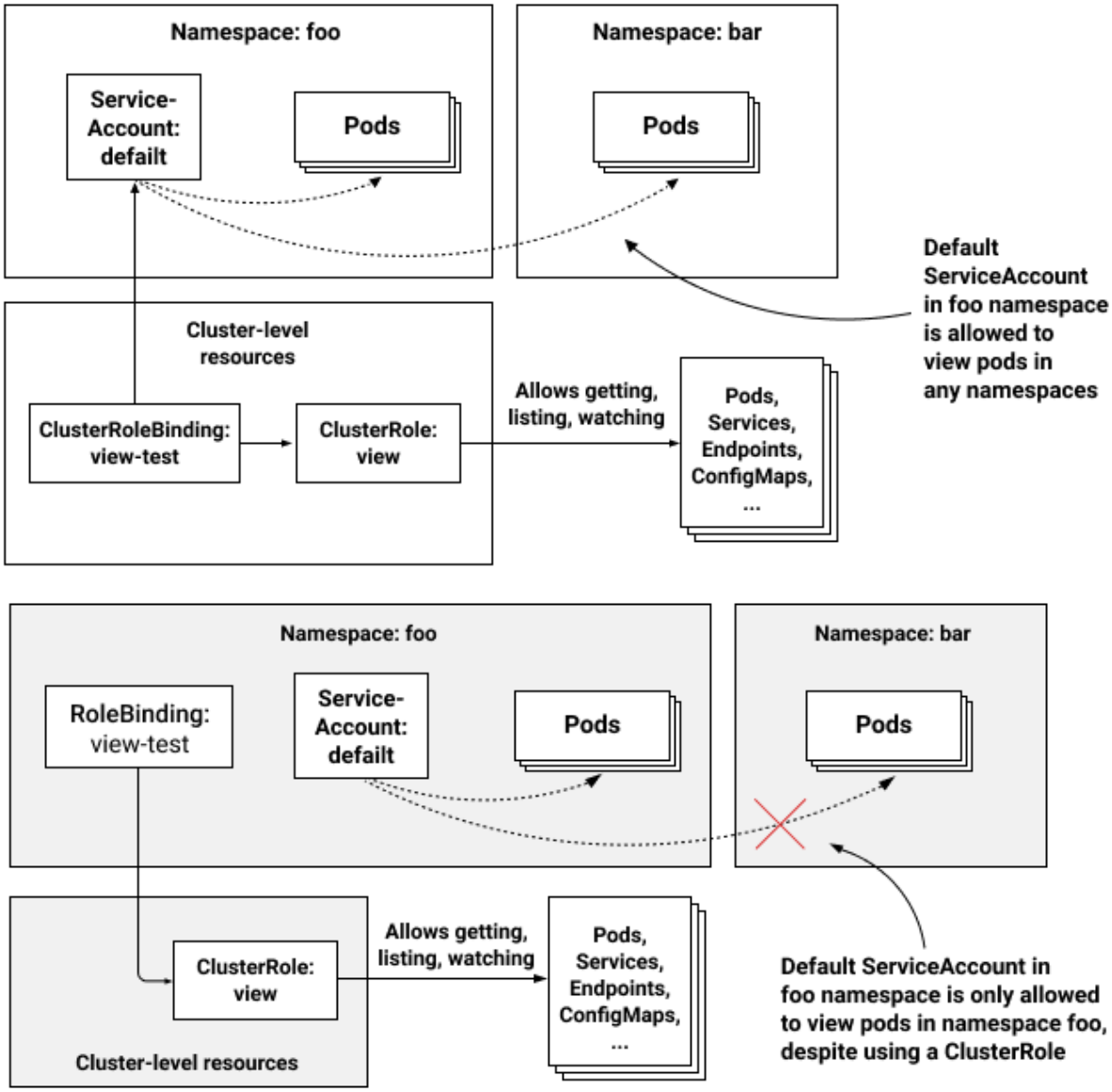
Используйте ClusterRoleBinding.



For accessing	Role type to use	Binding type to use
Cluster-level resources (Nodes, PersistentVolumes, ...)	ClusterRole	ClusterRoleBinding
Non-resource URLs (/api, /healtthz, ...)	ClusterRole	ClusterRoleBinding
Namespaced resources in any namespace (and across all namespaces)	ClusterRole	ClusterRoleBinding
Namespaced resources in a specific namespace (reusing the same ClusterRole in multiple namespaces)	ClusterRole	RoleBinding
Namespaced resources in a specific namespace (Role must be defined in each namespace)	Role	RoleBinding

Взгляните на роль kubectl get clusterrole system:discovery -o yaml. Вместо указания объектов доступа прописаны URL. Подобным способом также можно создавать роли и

предоставлять доступ. Только немного изменятся действия — get, post, patch. Конечно, для этой роли есть binding — kubectl get clusterrolebinding system:discovery -o yaml. Посмотрите, что в списке субъектов привязки. Group system:authenticated и Group system:unauthenticated. Даже неавторизованный пользователь может получить доступ к этим URL. Проверьте curl -k https://<MASTER IP>:8443/api. ClusterRole не обязательно предоставляет доступ к ресурсам кластера, также может описывать namespaced ресурсы. Например, kubectl get clusterrole view -o yaml. Эта кластерная роль ведет себя по-разному в зависимости от привязки. Попробуйте выполнить из пода curl localhost:8001/api/v1/pods или curl localhost:8001/api/v1/namespaces/foo/pods. А теперь привяжите роль kubectl create clusterrolebinding view-test --clusterrole=view --serviceaccount=foo:default. Что вы получите? Попробуйте заменить clusterrolebinding на rolebinding, какой будет результат?



Полезные ссылки:

- Using RBAC authorization

- Configure Service Accounts
- Service Accounts Admin
- Controlling Access to the Kubernetes API (official docs)
- Authenticating (official docs)
- Authorization Overview (official docs)
- Using Admission Controllers (official docs)
- Securing the Configuration of Kubernetes Cluster Components

Задание:

1. Создайте сервисный аккаунт sa-dp в namespace default.
2. Создайте кластерную роль cr-dp, которая будет позволять создавать deployments.
3. Привяжите аккаунт sa-dp к роли cr-dp в namespace default, используя rolebinding sa-dp-cr-dp.
4. Отправьте задание на проверку.