# Kubernetes.
# Модель безопасности

# Namespaces

# Namespaces

- Можно создать несколько виртуальных кластеров в рамках одного физического кластера.
- **Namespace** - один виртуальный кластер

# Namespaces

- Namespaces часто применяются для:
  - Обеспечения **multitenancy** (множественная аренда)
  - разграничения прав между командами
  - делегирования части административных функций доверенным пользователям
  - лимитирования ресурсов на проект с помощью квот (cpu, memory, storage)

# Namespaces

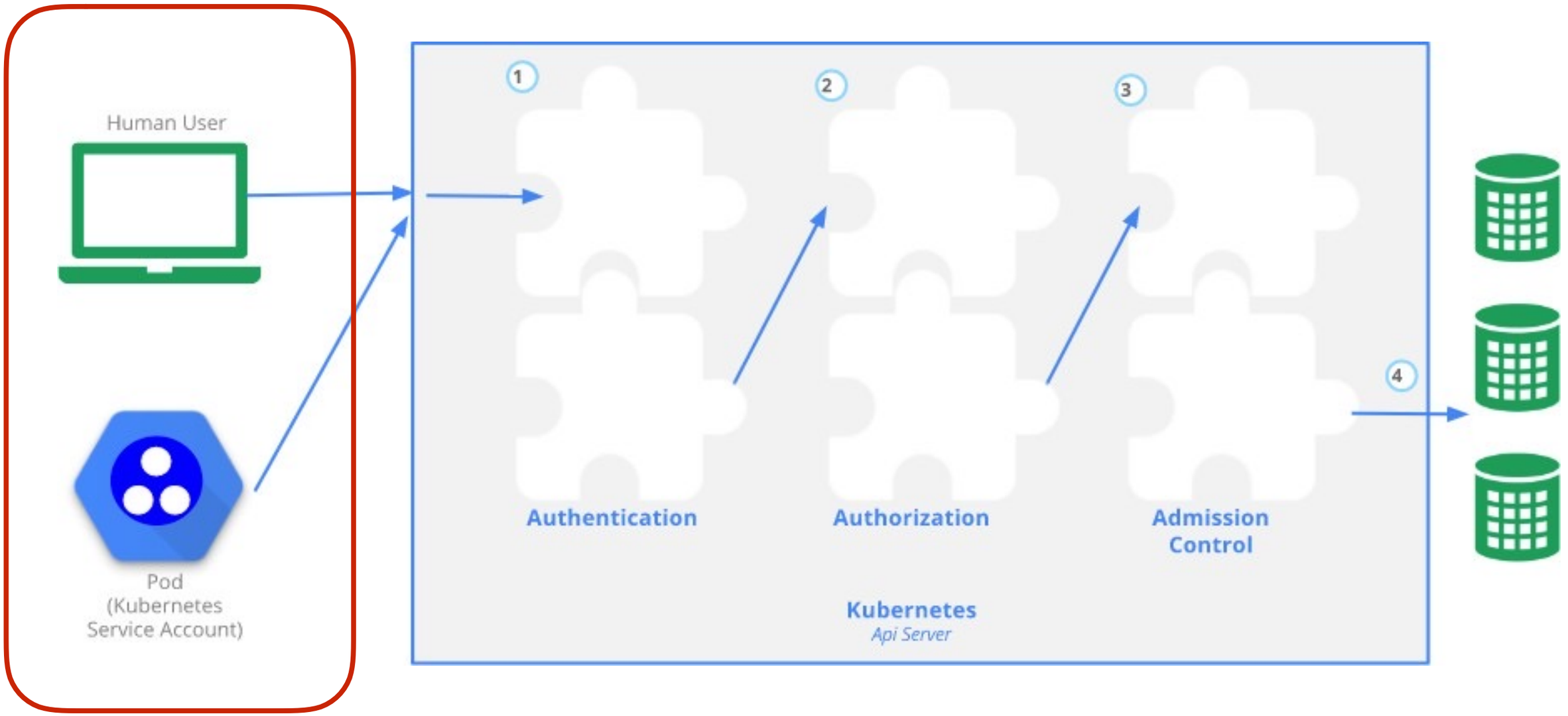
- Достигается это путем:
  - Создания области видимости имен (Names)
    - Имя (Name) - минимальный объект Kubernetes API (/apis/v1/namespaces/<namespace>/pods/<name>)
  - Подключения политик безопасности и авторизации к выделенной части кластера

# Identity management

# Accounts

- **User account** - учетная запись под которой работает пользователь
- **Service account** - учетная запись под которой процесс внутри pod'a работает с Kubernetes (например, с API)

# Accounts


Human User
Pod
(Kubernetes
Service Account)
1
2
3
4
Authentication
Authorization
Admission
Control
Kubernetes
Api Server

# Service accounts vs User accounts

## Service Accounts

- Для задач в Pod-ах
- разделены по Namespace’ам
- Управляются Kubernetes
- Конфигурация приложения могут включать Service Account

## User Accounts

- Для людей
- Работают во всем кластере
- Управляются внешними сервисами

# User accounts

- User Accounts не могут быть созданы через API Kubernetes (нет такого объекта в Kubernetes API)
- В качестве источников информации могут использоваться
  - Файлы (токены, сертификаты)
  - LDAP
  - SAML (Например, Google, Yandex, ADFS)
  - Kerberos

# Service accounts: Автоматизация

- Service Account Admission Controller
- Token Controller
- Service Account Controller

# Service Account Admission Controller

- Если у Pod'а не установлен Service Account, то устанавливает Service Account "default"
- Проверяет, что Service Account Pod'а существует
- Если у Pod'а нет своих ImagePullSecrets, то устанавливает ImagePullSecrets Service Account'а
- Подключает к Pod'у volume с ключом для API **/var/run/secrets/kubernetes.io/serviceaccount**

# Service Account

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: system-default
```

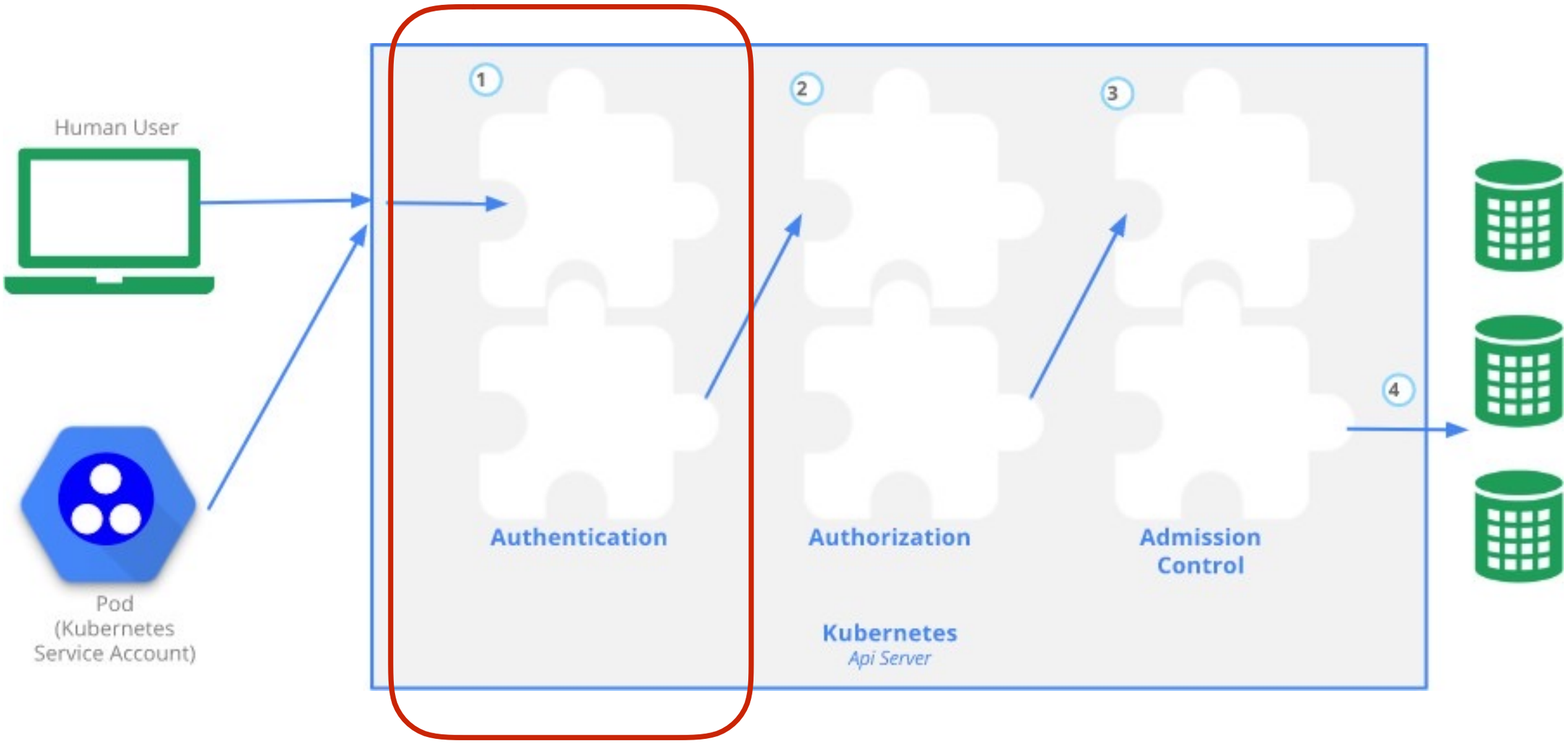
```
...
    - job_name: 'kubernetes-nodes'
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
```

# Service Account

```
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: prometheus-core
  namespace: system-default
spec:
 replicas: 1
  template:
    metadata:
      name: prometheus-main
      labels:
        k8s-app: prometheus
        component: core
    spec:
     serviceAccountName: prometheus
…
```

# Аутентификация

# Аутентификация



# Аутентификация

- X509 сертификаты
- Статичные токены
- Bootstrap токены
- Статичные файлы с паролями
- Authenticate Proxy

Могут использоваться сразу несколько методов

# Аутентификация

Основные поля:

- **Username** - идентификация конечного пользователя
- **UID** - уникальный идентификатор конечного пользователя
- **Groups** - объединение нескольких пользователей в 1 группу

# x509

- сертификаты должны быть подписаны корневым сертификатом кластера

```
kube-apiserver --client-ca-file=ca.crt
```

Создать запрос

```
openssl req -new -key jbeda.pem -out jbeda-csr.pem \
-subj "/CN=jbeda/O=app1/O=app2"
```

# Tokens

- Статичные токены
    - лежат в папке (чаще всего прямо на master-ноде)

```
kube-apiserver --token-auth-file=SOMEFILE
```

```
$ cat SOMEFILE
token,user,uid,”group1,group2,group3"
…
31ada4fd-adec-460c-809a-9e56ceb75269,admin,42,"group1,group2,group3"
```

# Token Controller

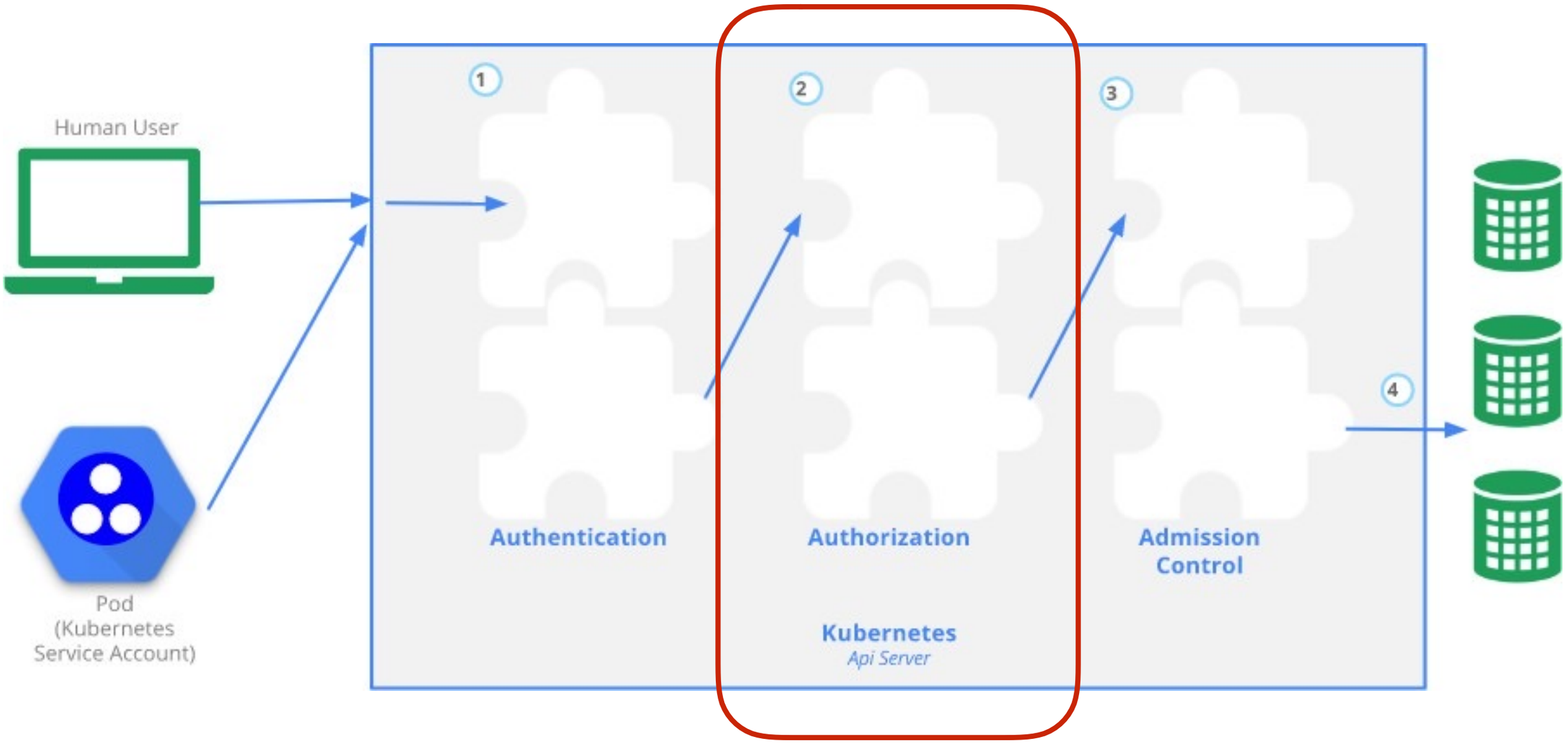
- Следит за созданием и удалением токенов для Service Account-ов

- Создает и удаляет соответствующие ключи доступа к API

- Следит, чтобы ключи всегда соответствовали существующему Service Account'y

# Авторизация

# Авторизация



# Авторизация

- Attribute-based access control (ABAC)
- **Role-based access control (RBAC)**
- Node
- Webhook

# ABAC

- Авторизирует действия пользователя на основе набора политик
- Политики состоят наборов атрибутов по которым происходит авторизация
- Виды атрибутов: пользовательские атрибуты, атрибуты ресурсов, объектов, окружений и т.д.

# ABAC

- Сервер стартует с флагом --authorization-policy-file=Policy.json

Policy.json

Одна запись (атрибут)

```
{"apiVersion":
    "abac.authorization.kubernetes.io/v1beta1",
    "kind": "Policy",
    "spec": {
    "user": "alice",
    "namespace": "*",
    "resource": "*",
    "apiGroup": “*"}
}
```

# RBAC

- Авторизация происходит на основании роли пользователя

- **Роль** - набор правил, задающих разрешения

- ClusterRole - роль в рамках всего кластера

- Role - роль в рамках одного namespace

- Роли пользователям назначаются с помощью привязок (RoleBindings)

# RBAC

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

Правило для чтения информации о pod-ах

1) Группа ресурсов
2) ресурсы
3) действия

# RBAC

**ClusterRole** = Role + :

- кластерные ресурсы (nodes, PersistentVolumes)
- нересурсные эндпоинты ( “/healthz”)
- ресурсы из нескольких или всех namespaces

# RBAC

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources:
  - deployments
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
```

блок правил

# RBAC-правила

```
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
```

блок ресурсов

# RBAC-правила

```
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
verbs: ["get", "list", "watch"]
```

блок разрешенных действий

# RBAC-правила

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps"]
  resources:
  - deployments
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
```

# RoleBinding

Назначаем роль:

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: services
```

# RoleBinding

Назначаем роль:

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: services
```

Что привязываем?

# RoleBinding

Назначаем роль:

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: services
```

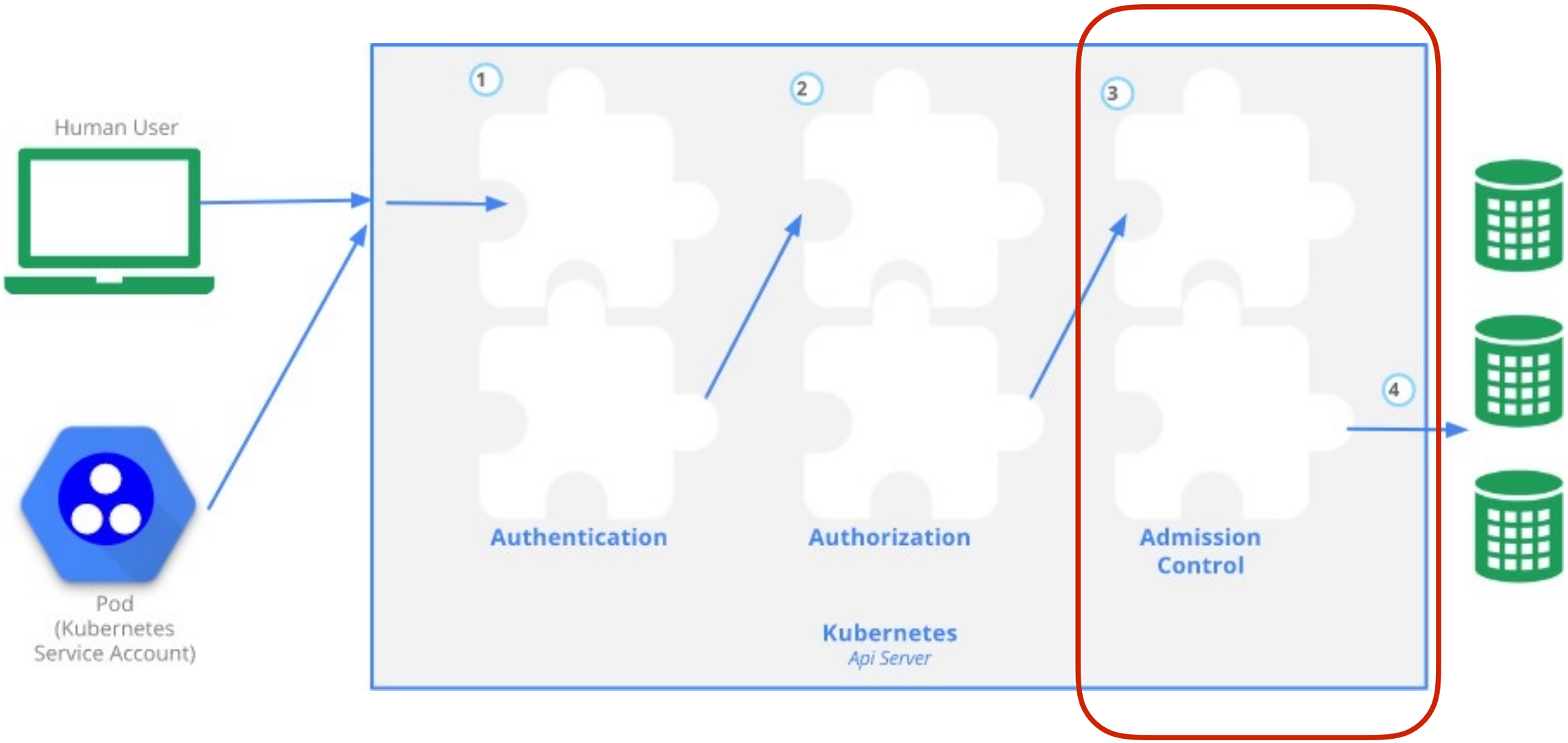
Кому привязываем?

# RoleBinding

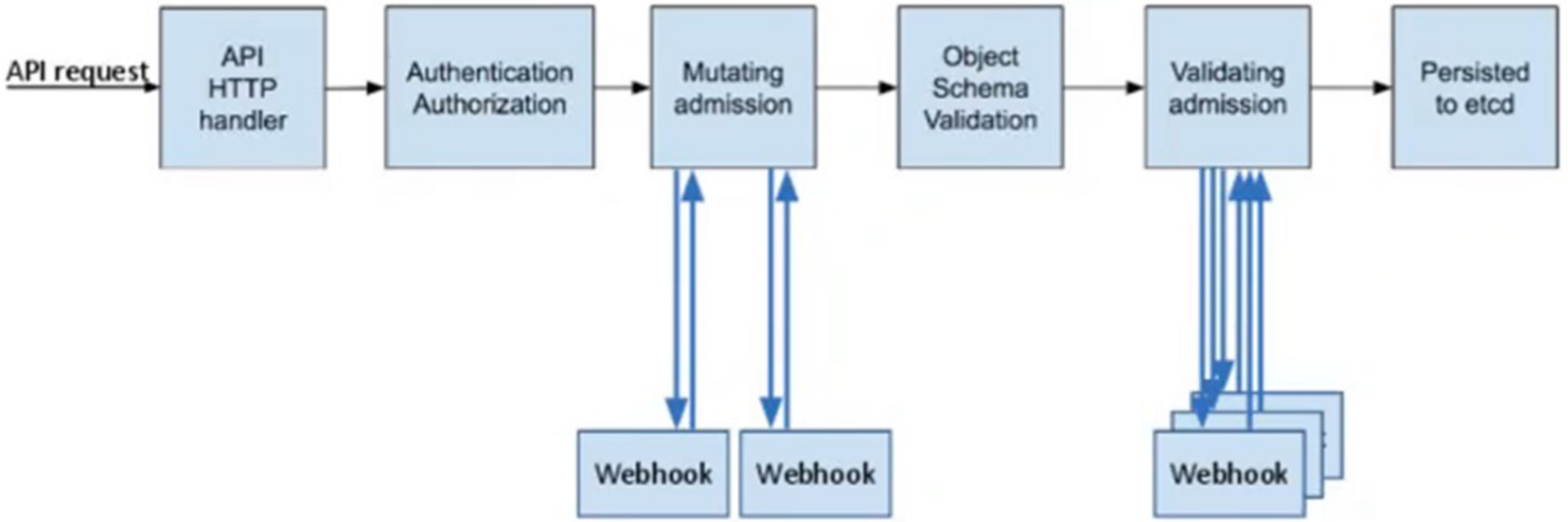
## Назначаем роль:

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: services
- kind: User
  name: John
  apiGroup: rbac.authorization.k8s.io
- kind: Group
  name: admins
  apiGroup: rbac.authorization.k8s.io
```

# Admission control



# Admission controllers



# Admission controllers

Работает на основе действий с ресурсами (отклоняет или разрешает):

```
--enable-admission-plugins=
NamespaceLifecycle,LimitRanger,
DefaultStorageClass,ResourceQuota,PodSecurityPolicy..
```