Что стоит знать о безопасности и управлении доступом в Kubernetes

Не забудь включить запись!

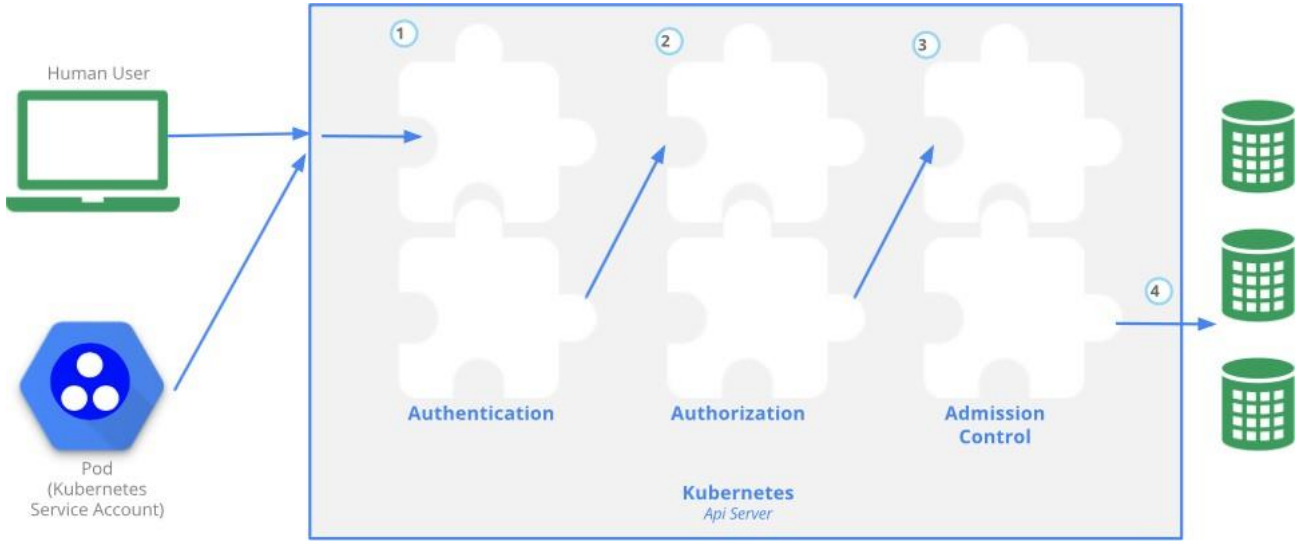
План

- Нужные примитивы и понятия
- Аутентификация
- Авторизация
- Admission Controllers

Нужные примитивы и понятия

AAA против AA

Помним о том, что Kubernetes разделяет эти понятия (и правильно делает):



AAA против AA

- **Аутентификация**
  - ответ на вопрос "**Кто** ты"? 👽
- **Авторизация**
  - ответ на вопрос "**Можно** ли мне вот это сделать?" 🔥
- **Admission Controllers (AC)**
  - **валидация** и **изменение** запросов перед помещением в кластер 🤖

Еще есть четвертая **A**:

- **Accounting/Auditing** - ответ на вопрос "Кто, что и когда сделал?" 🙀

Namespaces

Позволяют разруливать видимость объектов в кластере

- То есть, в одном физическом кластере может быть несколько виртуальных
- Ну, не совсем кластеров и не совсем виртуальных 😊

Не все вещи уходят в namespaces (nodes, persistentVolumes, например, нет)

- `kubectl api-resources` покажет в колонке namespaced - уходит ли оно в Namespace

Demo 1: Namespaces

Аутентификация

Виды пользователей

В Kubernetes API нет сущностей связанных с конкретными пользователями:

- объектов типа "Пользователь"
- объектов типа "Группа"
- паролей

Вот в каком LDAPe пользователя делали - вот туда и идите!

Группы или имена пользователей имеют смысл **только для** механизмов **авторизации**

Виды пользователей

- **Обычные пользователи**
  - Это люди, которые отдают команды кластеру
  - Глобальны в рамках кластера
  - Не управляются из API
- **Service Accounts**
  - Привязаны к жизни ресурса или процесса в кластере
  - Локальны в Namespace
  - Управляются из API
  - Привязаны к токену из `Secrets`, позволяют элементам кластера общаться с API

"Не управляются" это как?

Вы должны каким-то образом объяснить K8s, как аутентифицировать "человеков":

- X509 Client Certs
- Static Token File / Static Password File
- Bootstrap Tokens / Service Account Tokens
- OpenID Connect Tokens
- Webhook Token Authentication
- Authenticating Proxy
- Анонимный запрос

А что реально используется?

- **X509 Client Certs**
  - Kubernetes не поддерживает отзыв сертификатов
  - Можно делать их короткоживущими, но это много телодвижений
- **Static Password/Token File**
  - Не имеют срока жизни
  - Требуют рестарта API Server
- **OpenID Connect Tokens**
  - Внешняя система рулит этим вопросом

А в моем кластере?

- Помним, что способы аутентификации задаются при помощи параметров командной строки `kube-apiserver`
  - Формально нужно ее посмотреть и понять, что там
- К размышлению:
  - `--basic-auth-file=string`
  - `--token-auth-file=string`
  - `--anonymous-auth=true` - дефолт с 1.6+

Параметры kube-apiserver

Demo 2: x509 client cert

Выводы

- Судя по всему, аутентификация нам удалась
- Не очень удалась авторизация, то есть нам нельзя делать то, что мы хотим

Service Account

- Аккаунты пользователей - для пользователей, то есть для людей
  - Service Account для процессов в Pod
- Аккаунты пользователей глобальны в рамках кластера
  - Service Accounts живут в рамках namespace (это удобно для разных приложений внутри кластера)

Demo 3: Service Account

Авторизация

А что дают?

- Существует 4 модуля авторизации:
  - Node
  - ABAC
  - RBAC
  - Webhook

А что у меня в кластере?

Смотрим так (осторожно, МНОГО букв):

```
kubectl cluster-info dump | grep authorization-mode
```

Или так:

```
kubectl -n kube-system describe pod kube-apiserver-***

# .. CUT ..
--authorization-mode=Node,RBAC
# ..
```

Если что, вот так делать не стоит:

- `--authorization-mode=AlwaysAllow`
- `--authorization-mode=AlwaysDeny`

Node Authorizer

Этот модуль дает возможность kubelet совершать определенные действия с API, то есть:

Читать:

- Services
- Endpoints
- Nodes
- Pods
- Secrets, ConfigMaps, PersistentVolumes, PVC

которые относятся к Pod на этой Node

Node

Записывать:

- Nodes и их статус
- Pods их статус
- События

NodeRestriction бежит на помощь в ограничении власти kubelet при помощи этого модуля

ABAC

- Attribute-based access control (ABAC) строится на парадигме, где права доступа выдаются через использование политик
- Политика связывает вместе права и набор атрибутов

```
{
    "user": "alice",
    "namespace": "*",
    "resource": "*",
    "apiGroup": "*"
},
{
    "user": "bob",
    "namespace": "projectCaribou",
    "resource": "pods",
    "readonly": true
}
```

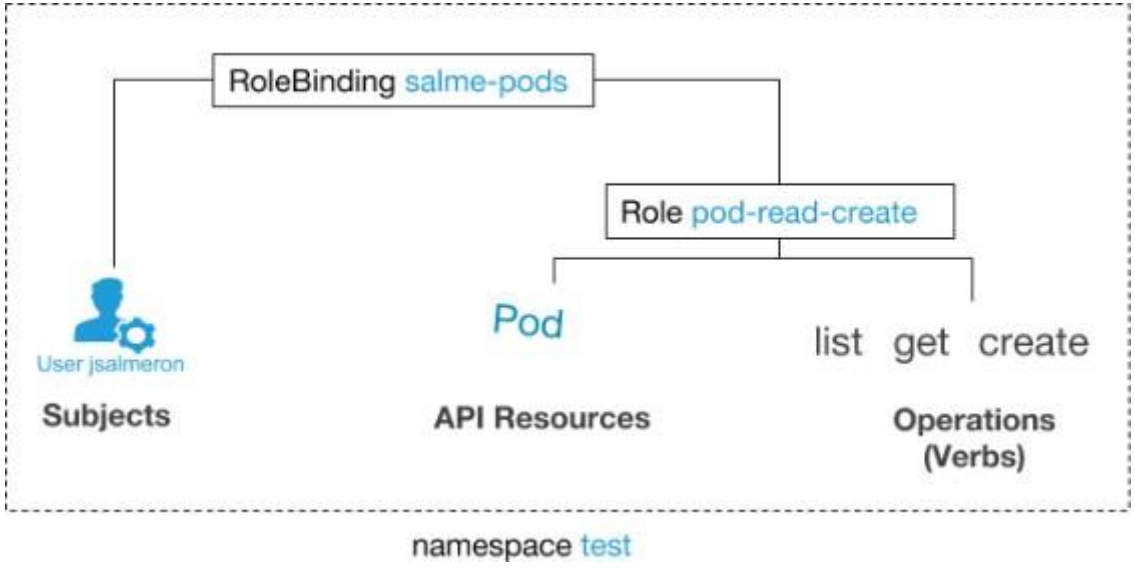
Webhook модуль

- Apiserver нужно передать вот этот параметр с путем к файлу настроек (куда ломиться для авторизации хотелок пользователя)

```
--authorization-webhook-config-file string
```

RBAC || ККК - Кто, кого и как?

- **RBAC** = Субъект + (Операция + Ресурс)



- **Role** = Операция + Ресурс
- **RoleBinding** = Субъект <-> Роль

Роли

- Начнем с ролей, которые по сути своей связывают набор операций и ресурсов.
- Роли могут быть ограничены namespace (Role), так и простираться на весь кластер (ClusterRole)

```
1 apiVersion: rbac.authorization.k8s.io/v1
2 kind: Role
3 metadata:
4   namespace: default
5   name: pod-reader
6 rules:
7 - apiGroups: [""] # "" означает apiGroup под именем core или legacy
8   resources: ["pods"]
9   verbs: ["get", "watch", "list"]
```

Роли

- Основное - это `apiGroups`, группы ресурсов, к которым мы даем доступ
- Параметр `namespace` не применяется в ClusterRole по понятным причинам

apiGroups

`kubectl api-resources` покажет нам и apiGroup и имена ресурсов

Например:

- `batch`
- `apps`
- `rbac.authorization.k8s.io`

Хорошо, как теперь эту роль применить?

- Роль "накатывается" при помощи RoleBinding (или ClusterRoleBinding)

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods      # под этим именем мы потом увидим этот RoleBinding
  namespace: default
subjects:
- kind: User           # Group, ServiceAccount
  name: jane           # имя чувствительно к регистру
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role           # явно указываем Role или ClusterRole
  name: pod-reader     # а тут имя той Role или ClusterRole к которой мы биндимся
  apiGroup: rbac.authorization.k8s.io
```

Решаем проблему

- Заставим работать пользователя и sa, которых мы ранее создали

Demo 4: Разбираем по кускам

Demo 5: Вся власть

Продолжение

Demo 5: Вся власть

Остальные роли?

- Мы пока посмотрели только роль cluster-admin, но из коробки их чуть больше
  - admin
  - edit
  - view
- Они по-умолчанию никому не выданы, их анализ — самостоятельная работа
- Ну и создавать свои роли никто, разумеется, не мешает :)

Admission Controllers

Admission Controller (AC)

- Kubernetes предлагает еще одну функцию ограничения доступа
- AC может делать две важные функции:
  - Изменять запросы к API (JSON Patch)
  - Пропускать или отклонять запросы к API
- Каждый контроллер может делать обе вещи, если захочет

> ❗Но сначала - мутаторы, потом - валидаторы

Admission Controller (AC)

- Какие-то AC включены сразу же из коробки
- А как это посмотреть?

```
kubectl cluster-info dump | grep enable-admission`

--enable-admission-
plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTolerat
ionSeconds,NodeRestriction,MutatingAdmissionWebhook,ValidatingAdmissionWebhook,ResourceQ
uota
```

NamespaceLifecycle

- Запрещает создавать новые объекты в удаляемых Namespaces
- Не допускает указания несуществующих Namespaces
- Не дает удалить системные Namespaces
  - `default`
  - `kube-system`
  - `kube-public`

ResourceQuota

- Определяется для namespace
- Ограничивает
  - кол-во объектов
  - общий объем ресурсов
  - объем дискового пространства для volumes

LimitRanger

- Определяется для namespace
- Возможность принудительно установить ограничения по ресурсам pod-а

И еще

- NodeRestriction
  - Ограничивает возможности kubelet по редактированию Node и Pod
- ServiceAccount
  - Автоматически подсовывает в Pod необходимые секреты для функционирования Service Accounts
- Mutating + Validating AdmissionWebhook
  - Позволяют внешним обработчикам вмешиваться в обработку запросов, идущих через AC

[Pod]SecurityContext

- Есть два объекта, которые контролируют фичи (без)опасности Container или Pod
- Свойство `securityContext` его содержит
- Наиболее часто используемое:
  - UID, GID
  - Возможность привелигированного запуска (доступ к устройствам)
  - Linux Capabilities 
  - AppArmor и SELinux

PodSecurityPolicy

- Но отсюда сразу же возникает вопрос - как контролировать [Pod]SecurityContext в рамках кластера?
- `PodSecurityPolicy` позволяет нам контролировать [Pod]SecurityContext
- Его читает и применяет нужный PodSecurityPolicy AC
- Достаточно граблеопасен, поскольку в RBAC-среде требует вдумчивого анализа
  - И внимательного включения

Как это работает?

- Мы создаем `PodSecurityPolicy`, в которой описываем параметры Pod
- Даем право пользоваться этой `PodSecurityPolicy` при помощи Role + RoleBinding
- Любой создаваемый Pod должен быть одобрен AC
- Если мы в политике для пользователя сделали `privileged: false`, а он пытается сделать `privileged: true`, то Pod не создастся
- Так мы можем гранулярно решать, кому какие (не)безопасные фичи можно использовать

На практике

- Если мы без настройки включим `PodSecurityPolicy` AC, то мы приложим наш кластер
  - Нет ни одной политики, поэтому нельзя создавать Pod
- Поэтому сначала создаем наиболее разрешительную политику
- Всем даем право ее использовать
- Дальше начинаем создавать более запретительные политики, вкатывая их по мере необходимости

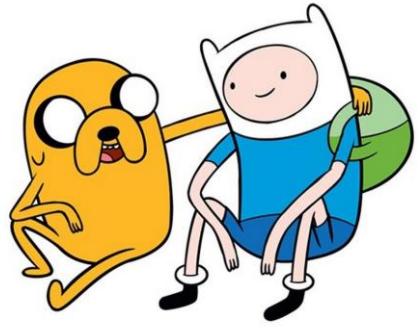
RIP

Kubernetes has officially deprecated PodSecurityPolicy in version 1.21, although it will likely continue to be available in Kubernetes until version 1.25.

https://kubernetes.io/blog/2021/04/06/podsecuritypolicy-deprecation-past-present-and-future/

Ссылки

- Статья в блоге CNCF
- Как начать жить с RBAC + полезные инструменты на The New Stack
- Дружим RBAC и PSP click!
- Расширяем возможности kubectl при помощи плагинов



Спасибо за внимание!

Время для ваших вопросов!