Безопасность

Авторизация и аутентификация

По умолчанию аутентификация на основе сертификата, но поддерживаются внешние источники.

Аутентификация на основе сертификата.

Авторизация RBAC (пользователь - действие - ресурс). По умолчанию запрещено все что не разрешено. Роли определяют правила, RoleBindings определяют принадлежность пользователей к ролям. Пример настройки ролей:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: shield
  name: read-deployments
rules:
- verbs: ["get", "watch", "list"] <<==== Allowed actions
  apiGroups: ["apps"] <<==== on resources
  resources: ["deployments"] <<==== of this type
```

Пример RoleBinding:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-deployments
  namespace: shield
subjects:
- kind: User
  name: sky <<==== Name of the authenticated user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: read-deployments <<==== This is the Role to bind to the user
  apiGroup: rbac.authorization.k8s.io
```

Свойства правил роли:

verbs ["get", "watch", "list", "create", "update", "patch", "delete"]

ApiGroups (в пределах namespace):

apiGroup	Ресурс
""	pods, secrets
“storage.k8s.io”	storageclass
“apps”	deployments

Полный список API ресурсов:

```
kubectl api-resources --sort-by name -o wide
```

Можно использовать звездочку.

Все роли используются только в контексте namespace!

Кластерные роли и привязки

ClusterRoleBindings используется для создания шаблонов ролей и привязки их к конкретным ролям.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole <<==== Cluster-scoped role
metadata:
  name: read-deployments
rules:
- verbs: ["get", "watch", "list"]
  apiGroups: ["apps"]
  resources: ["deployments"]
```

Пользователи

Интересная статья Еще одна, тоже стоит почитать

Обычных пользователей нельзя добавить через вызовы API. Возможные варианты:

- Базовая аутентификация *(basic auth)*:
  - передача конфигурации API-серверу со следующим (или похожим) содержимым: password, username, uid, group;
- Клиентский сертификат X.509:

    - создание секретного ключа пользователя и запроса на подпись сертификата;
    - заверение его в центре сертификации (Kubernetes CA) для получения сертификата пользователя;
- Bearer-токены (JSON Web Tokens, JWT):
    - OpenID Connect;
    - слой аутентификации поверх OAuth 2.0;
    - веб-хуки *(webhooks)*.

Структура файла ~/.kube/config

- Clusters - список кластеров. Сертификат кластера, адрес и внутреннее имя
- Users - пользователи. Внутреннее имя, сертификат и ключ
- Contexts - объединение пользователя и кластера. Внутреннее имя, внутреннее имя кластера и внутреннее имя пользователя
- Current-context - имя текущего контекста

Важный момент: кубер не управляет членством пользователей в группах. Получить напрямую доступ к спискам пользователей в группе нельзя.

Пример создания пользователя с авторизацией через X.509 сертификат.

Создаем директорию хранения информации о пользователях и генерируем в нее ключ

```
mkdir -p users/sergey/.certs
openssl genrsa -out ~/users/sergey/.certs/sergey.key 2048
```

Генерируем запрос на сертификат

```
openssl req -new -key ~/users/sergey/.certs/sergey.key -out ~/users/sergey/.certs/sergey.csr -subj "/CN=sergey/O=testgroup"
```

Обработка запроса на сертификат

```
openssl x509 -req -in ~/users/sergey/.certs/sergey.csr -CA /etc/kubernetes/pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out ~/users/sergey/.certs/sergey.crt -days 500
```

В некоторых ресурсах говорится, что команда kubectl config set-credentials ... создает пользователя в кластере Kubernetes. Но это не так, команда kubectl config ... создает/модифицирует файл .kube/config, поэтому нужно быть осторожным и не побить свой файл. А Kubernetes авторизует всех пользователей, чей сертификат подписан его центром сертификации.

Добавляем пользователя sergey

```
kubectl config set-credentials sergey \
--client-certificate=/root/users/sergey/.certs/sergey.crt \
--client-key=/root/users/sergey/.certs/sergey.key \
--embed-certs=true
```

Если нужно - создали бы настройки кластера, но у нас есть, поэтому создаем контекст с существующим кластером. Namespace, если нужно, указывается в настройках контекста.

```
kubectl config set-context sergey-context --cluster=kubernetes --user=sergey --namespace=sergey-ns
```

Теперь осталось пользователю определить права.

Например определим роль sergey-ns-full с полным доступом к namespace sergey-ns

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: sergey-ns-full
  namespace: sergey-ns
rules:
  - apiGroups: [ "*" ]
    resources: [ "*" ]
    verbs: [ "*" ]
```

Сейчас вместо привязки пользователя, привяжем группу к роли.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: testgroup-rolebinding  # Название RoleBinding
  namespace: sergey-ns         # Namespace, где применяется
subjects:
- kind: Group                  # Тип субъекта — группа
  name: testgroup              # Название группы
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role                   # Тип привязываемой роли (Role или ClusterRole)
  name: sergey-ns-full         # Название роли
  apiGroup: rbac.authorization.k8s.io
```

Переключаемся на контекст и проверяем

```
kubectl config use-context sergey-context
```

Создаем простой под

```
kind: Pod
apiVersion: v1
metadata:
  name: hello-pod
  labels:
    zone: prod
    version: v1
spec:
  containers:
  - name: hello-ctr
    image: nigelpoulton/k8sbook:1.0
    ports:
    - containerPort: 8080
    resources:
      limits:
        memory: 128Mi
        cpu: 0.5
```

Проверяем факт создания пода

```
kubectl get pods
```

Удалось!

Основные команды управления пользователями

Если при создании ... указать флаг --embed-certs=true то тогда вместо путей к файлам сертификатов, в файл настройки будут встроено содержание сертификатов в Base64.

Команда	Доп. параметры	Описание
kubectl get clusterroles.rbac.authorization.k8s.io --all-namespaces		Список пользователей
kubectl config view		Показать текущую конфигурацию (.kube/config)
kubectl config current-context		Показать текущий активный контекст

Команда	Доп. параметры	Описание
kubectl config get-contexts		Список всех контекстов
kubectl config use-context cont_name		Переключиться на контекст cont_name
kubectl config set-cluster clast_name		Добавить/изменить кластер `kubectl config set-cluster my-new-cluster \ --server=https://10.0.0.1:6443 \ --certificate-authority=./ca.crt`
kubectl config set-credentials		Добавить/изменить учетные данные пользователя `kubectl config set-credentials sergey \ --client-certificate=/root/users/sergey/.certs/sergey.crt \ --client-key=/root/users/sergey/.certs/sergey.key \ --embed-certs=true`
kubectl config set-context cont_name	--cluster=dev-cluster \ --user=dev-user \ --namespace=dev-ns	Создать/изменить контекст
kubectl config delete-context		Удалить контекст
kubectl config delete-cluster		Удалить кластер
kubectl config delete-user user_name		Удалить пользователя
kubectl config rename-context		Переименовать контекст

Безопасность, общая теория

Инструменты

Запрет передачи ключей SA

Каждому поду по умолчанию передаются ключи сервисный аккаунт. Поэтому при получении доступа к поду можно получить доступ вплоть до всего кластера. Обычно подам не нужно управлять кластером. Поэтому можно запретить передачу ключей.

```
apiVersion: v1
kind: Pod
metadata:
  name: service-account-example-pod
spec:
  serviceAccountName: some-service-account
  automountServiceAccountToken: false <<==== This line
<Snip>
```

Также можно передавать временные ключи, но это потом.

Контроль целостности ресурсов

- Ограничьте доступ к серверам, на которых запущены компоненты Kubernetes, особенно к компонентам control plane
- Ограничьте доступ к репозиториям, хранящим конфигурационные файлы Kubernetes
- Передача файлов и управление только через SSH
- Проверка контрольной суммы после скачивания
- Ограничьте доступ к регистру образов и связанным хранилищам

Файловая система пода в read-only режим

```
apiVersion: v1
kind: Pod
metadata:
  name: readonly-test
spec:
  securityContext:
    readOnlyRootFilesystem: true <<==== R/O root filesystem
    allowedHostPaths: <<==== Make anything below
      - pathPrefix: "/test" <<==== this mount point
        readOnly: true <<==== read-only (R/O)
<Snip>
```

Лог действий на кластере и связанной инфраструктуре

Защита данных кластера

Cluster store (обычно etcd) хранит все данные. Необходимо ограничить и контролировать доступ к серверам, на которых работает Cluster store.

DoS

Подвергается API сервер. Должно быть минимум 3 Control plane сервера и 3 worker ноды. Изоляция etcd на сетевом уровне. Ограничения ресурсов для подов и количества подов.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pod-quota
  namespace: skippy
spec:
  hard:
    pods: "100"
```

Доп. опция podPidsLimit ограничивает количество процессов одним подом. Также можно ограничить кол-во подов на одной ноде.

По умолчанию etcd устанавливается на сервер с control plane. На production кластере нужно разделять.

Запретить сетевое взаимодействие между подами и внешние взаимодействия (где это не нужно) при помощи сетевых политик Kubernetes.

Защита подов и контейнеров

Запрет запуска процессов от root

```
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  securityContext: <<==== Applies to all containers in this Pod
    runAsUser: 1000 <<==== Non-root user
  containers:
    - name: demo
      image: example.io/simple:1.0
```

Это запускает все контейнеры от одного непривилегированного пользователя, но позволяет контейнерам использовать общие ресурсы. При запуске нескольких подов, будет

аналогично. Поэтому лучше дополнительно настраивать пользователей контейнера:

```
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  securityContext: <<==== Applies to all containers in this Pod
    runAsUser: 1000 <<==== Non-root user
  containers:
    - name: demo
      image: example.io/simple:1.0
      securityContext:
        runAsUser: 2000 <<==== Overrides the Pod-level setting
```

Рутовые права складываются примерно из 30 capabilities. Простой способ - в тестовом окружении ограничить все и по логам добавлять нужные. Естественно финальное тестирование должно быть максимально всеобъемлющим. Пример разрешений:

```
apiVersion: v1
kind: Pod
metadata:
  name: capability-test
spec:
  containers:
    - name: demo
      image: example.io/simple:1.0
      securityContext:
        capabilities:
          add: ["NET_ADMIN", "CHOWN"]
```

Фильтрация системных вызовов.

Похоже на capabilities, но фильтрует системные вызовы. Способы поиска минимальных разрешений: разрешить все + логгирование, запрет + постепенное разрешение.

Также есть Pod Security Standarts (PSS) и Pod Security Admission (PSA). PSA применяют PSS при старте пода.

Основные команды

Параметр	Описание

kubectl describe clusterrole role_name	Описание роли
kubectl get clusterrolebindings | grep role_name	Список пользователей с такой ролью
kubectl describe clusterrolebindings role_name	Информация по сопоставлению
Аналогично для ролей (clusterrolebindings -> rolebindings)	

Revision #8
Created 28 March 2025 15:09:28 by Admin
Updated 30 March 2025 07:30:27 by Admin