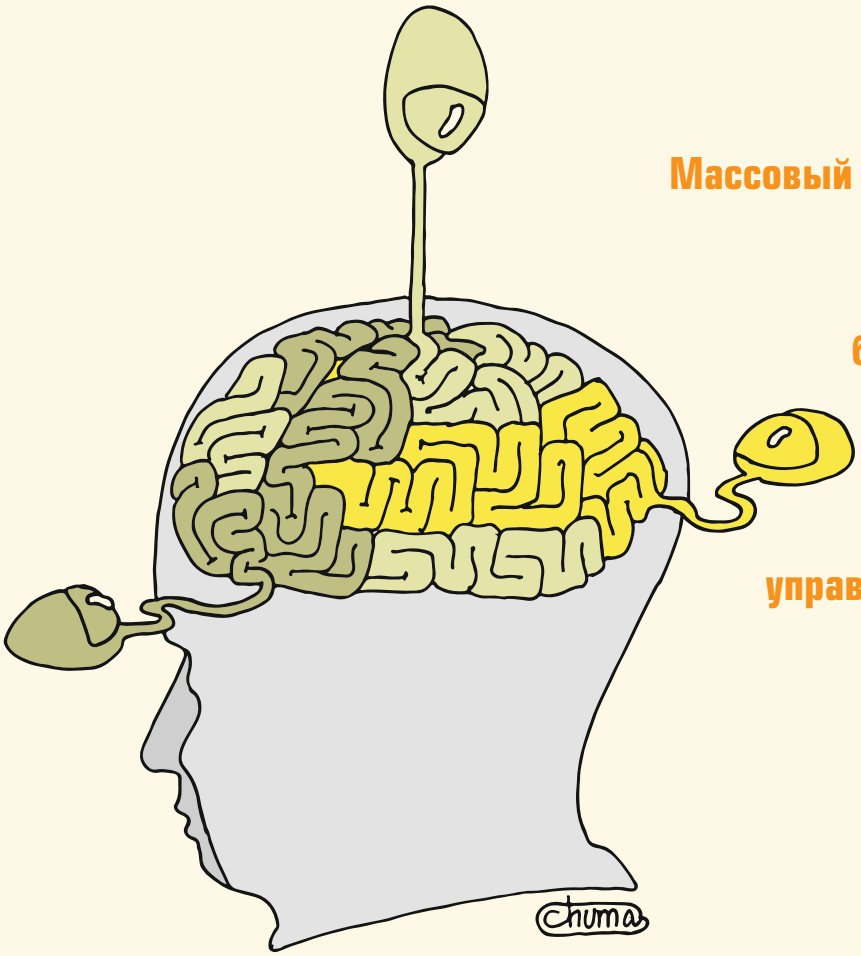
«СА» – 20 лет: наши темы, наши авторы

ISSN 1813-5579 22010
9 771813 557005

Системный администратор

ежемесячный журнал www.samag.ru

№10(239)
2022



Поиск пользователей Active Directory и экспорт

Массовый сброс паролей через PowerShell

Как управлять сегодня

большим количеством серверов?

Openshift и все вокруг него, часть 7:

управление пользователями и ролями

Ключевые преимущества

использования STIX/TAXII

Компрометация ключа электронной подписи

Почему она происходит, и что нужно делать?

Робоистория

Все, что нужно знать о роботах и робототехнике

Наука и технологии

Наука и технологии

16+

Разработка программных средств для моделирования распространения вирусных заболеваний

Программирование аналитического моделирования системы массового обслуживания на языках программирования Fantom и Pike

Визитка

СЕРГЕЙ ГОЛОВАШОВ,
ведущий инженер DevOps, руководитель центра компетенций, компания Bell Integrator

Визитка

НИКОЛАЙ СИТНИКОВ,
инженер DevOps, компания Bell Integrator

Openshift и все вокруг него,
часть 7: управление пользователями и ролями

Администрирование пользователей и ролей необходимо для управления пользователями, их доступом и контролем над различными проектами (неймспейсами). Пока мы рассмотрим функционал и примеры Openshift'а, а потом коснемся и самого Кубера.

Создание пользователя

Предопределенные ниже шаблоны могут использоваться для создания новых пользователей в OpenShift.

Используйте oc create --f <имя файла> для создания пользователей.

```
$ oc create --f vipin.yaml
```

Используйте следующую команду, чтобы удалить пользователя в OpenShift.

```
$ oc delete user <user name>
```

Ограничение доступа пользователя

ResourceQuotas и LimitRanges используются для ограничения уровней доступа пользователей. Они также используются для ограничения Pod'ов и контейнеров в кластере.

```
kind: "Template" apiVersion: "v1" parameters:
   name: vipin required: true
objects:
   kind: "User" apiVersion: "v1" metadata:
name: "${email}"

   kind: "Identity" apiVersion: "v1" metadata:
name: "vipin:${email}" providerName: "SAML" providerUserName:
"${email}"
   kind: "UserIdentityMapping" apiVersion: "v1"
identity:
name: "vipin:${email}" user:
name: "${email}"

apiVersion: v1 kind: ResourceQuota metadata:
name: resources-utilization spec:
hard:
pods: "10"
```

Создание цитаты с использованием вышеуказанной конфигурации

```
$ oc create -f resource-quota.yaml --n --Openshift-sample
$ oc describe quota resource-quota -n Openshift-sample
```

Описание ресурса

```
Name:                   resource-quota
Namespace:              Openshift-sample
Resource                Used                Hard
--------                ----                ----
pods                    3                   10
```

Определение ограничений контейнера может применяться для ограничения ресурсов, которые будут использоваться развернутыми контейнерами. Те используются для определения максимальных и минимальных ограничений определенных объектов.

Пользовательские граничения проекта

Это в основном используется для количества проектов, которые пользователь может иметь в любой момент времени. Они в основном сделаны путем определения пользовательских уровней в категориях бронза, серебро и золото.

Сначала нам нужно определить объект, который содержит значение того, сколько проектов может иметь бронзовая, серебряная и золотая категории. Это нужно сделать в файле master-config.yaml.

```
admissionConfig: pluginConfig:
   ProjectRequestLimit: configuration:
      apiVersion: v1
      kind: ProjectRequestLimitConfig limits:
         selector: level: platinum
         selector: level: gold
      maxProjects: 15
         selector: level: silver
      maxProjects: 10
         selector: level: bronze
      maxProjects: 5
```

Перезагрузите главный сервер.

Назначение пользователя определенному уровню.

```
$ oc label user vipin level = gold
```

Перемещение пользователя из метки, если требуется.

```
$ oc label user <user_name> level-
```

Добавление ролей пользователю.

```
$ oadm policy add-role-to-user <user_name>
```

Удаление роли от пользователя.

```
$ oadm policy remove-role-from-user <user_name>
```

Добавление роли кластера для пользователя.

```
$ oadm policy add-cluster-role-to-user <user_name>
```

Удаление роли кластера от пользователя. Добавление роли в группу.

```
$ oadm policy add-role-to-user <user_name>
```

Удаление роли из группы.

```
$ oadm policy remove-cluster-role-from-user <user_name>
```

Добавление роли кластера в группу.

```
$ oadm policy add-cluster-role-to-group <groupname>
```

Удаление роли кластера из группы.

```
$ oadm policy remove-cluster-role-from-group <role> <groupname>
```

Пользователь для администрирования кластера

Это одна из самых мощных ролей, где пользователь имеет возможность управлять целым кластером, начиная с создания и заканчивая удалением кластера.

Openshift – это хорошо, а теперь давайте поговорим про роли Kubernetes, потому что этот вопрос тоже очень актуален для нас сейчас.

Определение ограничений контейнера может применяться для ограничения ресурсов, которые будут использоваться развернутыми контейнерами. Те используются для определения максимальных и минимальных ограничений определенных объектов

Эталонным примером стал Helm: простой запуск helm init + helm install больше не работал. Внезапно нам потребовалось добавлять «странные» элементы вроде ServiceAccounts или RoleBindings ещё до того, как разворачивать чарт с WordPress или Redis

Многие опытные пользователи Kubernetes могут вспомнить релиз Kubernetes 1.6, когда авторизация на основе Role-Based Access Control (RBAC) получила статус бета-версии. Так появился альтернативный механизм аутентификации, который дополнил уже существующий, но трудный в управлении и понимании, – Attribute-Based Access Control (ABAC).

Все с восторгом приветствовали новую фичу, однако в то же время бесчисленное число пользователей были разочарованы. StackOverflow и GitHub изобиловали сообщениями об ограничениях RBAC, потому что большая часть документации и примеров не учитывали RBAC (но сейчас уже всё в порядке).

Эталонным примером стал Helm: простой запуск helm init + helm install больше не работал. Внезапно нам потребовалось добавлять «странные» элементы вроде ServiceAccounts или RoleBindings ещё до того, как разворачивать чарт с WordPress или Redis.

Оставив же эти неудачные первые попытки в стороне, нельзя отрицать тот огромный вклад, что внёс RBAC в превращение Kubernetes в готовую к production платформу. Многие из нас успели поиграть с Kubernetes с полными привилегиями администратора, и мы прекрасно понимаем, что в реальном окружении необходимо:

- Иметь множество пользователей с разными свойствами, обеспечивающими нужный механизм аутентификации.
- Иметь полный контроль над тем, какие операции может исполнять каждый пользователь или группа пользователей.
- Иметь полный контроль над тем, какие операции может исполнять каждый процесс в поде.
- Ограничивать видимость определённых ресурсов в пространствах имён.

И в этом отношении RBAC – ключевой элемент, предоставляющий столь необходимые возможности.

Чтобы полностью осознать идею RBAC, нужно понимать, что к ней причастны три элемента:

- Subjects (субъекты) – совокупность пользователей и процессов, которые хотят иметь доступ в Kubernetes API;
- Resources (ресурсы) – совокупность объектов Kubernetes API, доступных в кластере. Их примерами (среди прочих) являются Pods, Deployments, Services, Nodes, PersistentVolumes;

- Verbs (глаголы) – совокупность операций, которые могут быть выполнены над ресурсами. Существуют различные verbs (get, watch, create, delete и т.п.), но все они в конечном счёте являются операциями из разряда CRUD (Create, Read, Update, Delete).

Если помнить об этих трёх элементах, ключевая идея RBAC звучит так: мы хотим соединить субъекты, ресурсы API и операции. Другими словами, мы хотим указать для заданного пользователя, какие операции могут быть исполнены на множестве ресурсов. А теперь давайте перейдем к примерам.

Пример RoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
   name: edit
   namespace: project-a-devel
subjects:
- kind: Group
   name: bell
   apiGroup: rbac.authorization.k8s.io
roleRef:
   kind: ClusterRole
   name: edit
   apiGroup: rbac.authorization.k8s.io
```

Эта роль:

- Действует в рамках namespace project-a-devel - (metadata:)
- Применяется к пользователям в группе bell (/O=bell) - (subjects:)
- Разрешает действия, описанные в созданной по умолчанию кластерной роли (ClusterRole) edit - (roleRef:)

Какие права предоставляет ClusterRole edit, можно посмотреть через kubectl get clusterrole edit -o yaml (ниже приведена часть вывода)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
aggregationRule:
   clusterRoleSelectors:
   - matchLabels:
        rbac.authorization.k8s.io/aggregate-to-edit: "true"
metadata:
   annotations:
      rbac.authorization.kubernetes.io/autoupdate: "true"
   labels:
      kubernetes.io/bootstrapping: rbac-defaults
   name: edit
rules:
- apiGroups:
   - ""
   resources:
   - pods
   - pods/attach
   - pods/exec
   - pods/portforward
   - pods/proxy
   verbs:
   - create
   - delete
   - deletecollection
   - get
   - list
   - patch
   - update
   - watch
...
```

В секции rules описано, в каких группах api (apiGroups:), над какими ресурсами (resources:), какие действия (verbs:) может совершать данная роль.

Пример выдачи прав пользователю через x509 сертификата

При использовании сертификатов пользователя и группу как таковую создавать не надо, т.к. аутентификация происходит по параметрам CN(CommonName) и O(Organization), указанным в подписанном сертификате, который означает пользователя и группу(ы) соответственно, а дальнейшая авторизация действия пользователя происходит уже по описанным для этих CN,O в Role и RoleBinding.

> Чтобы работать с Kubernetes в production или Openshift, политики RBAC не являются опциональными. Их нельзя рассматривать как набор объектов API, которые должны знать только администраторы

В случае использования в k8s-кластере самоподписанного CA, процедура выглядит примерно так:

1) Создать ключ и сгенерировать csr с соответсвующими CN, O

```
openssl genrsa -out vpupkin.key 4096
openssl req -new -key vpupkin.key -out vpupkin.csr -subj
"/CN=v.pupkin/O=devops/O=bell"
Зайти на k8s master и подписать csr с помощью своего CA
openssl x509 -req -in vpupkin.csr -CA /etc/kubernetes
/ssl/ca.pem -CAkey /etc/kubernetes/ssl/ca-key.pem -
CAcreateserial -out vpupkin.crt -days <requiredDays>
```

2) Полученный сертификат и сгенерированный ранее ключ добавить в конфиг kubectl в ${HOME}/.kube/config

Вместо заключения

Чтобы работать с Kubernetes в production или Openshift, политики RBAC не являются опциональными. Их нельзя рассматривать как набор объектов API, которые должны знать только администраторы. Они на самом деле нужны разработчикам для развёртывания безопасных приложений и полного использования потенциала, предлагаемого Kubernetes API для облачных (cloud native) приложений.

Ну. а мы выходим на финишную прямую и начинаем говорить про безопасность внутри кластера. Следующие статьи будут завершать цикл про Openshift и касаться уже только безопасности самой среды. EOF

Ключевые слова: управление пользователями и ролями, openshift, kubernetes