

k8s-进阶篇-RBAC

文档: <https://kubernetes.io/zh/docs/reference/access-authn-authz/rbac/>

一、什么是RBAC?

基于角色 (Role) 的访问控制 (RBAC) 是一种基于组织中用户的角色来调节控制对 计算机或网络资源的访问的方法。

RBAC 鉴权机制使用 `rbac.authorization.k8s.io` API 组 来驱动鉴权决定, 允许你通过 Kubernetes API 动态配置策略。

要启用 RBAC, 在启动 API 服务器 时将 `--authorization-mode` 参数设置为一个逗号分隔的列表并确保其中包含 RBAC。

二、RBAC分类介绍 (API 对象)

2.1、RBAC有4种顶级资源

- Role
- ClusterRole
- RoleBinding
- ClusterRoleBinding

2.2、资源介绍

Role: 角色, 包含一组权限的规则。没有拒绝规则, 只是附加允许。Namespace隔离, 只作用于命名空间内!

ClusterRole: 集群角色, 和Role一样。和Role的区别, Role是只作用于命名空间内, ClusterRole作用于整个集群!

RoleBinding: 作用于命名空间内, 将ClusterRole或者Role绑定到User、Group、ServiceAccount!

ClusterRoleBinding: 作用于整个集群。

中文官方文档: <https://kubernetes.io/zh/docs/reference/access-authn-authz/rbac/>

三、示例

3.1、Role 示例

Role 总是用来在某个名字空间内设置访问权限; 在你创建 Role 时, 你必须指定该 Role 所属的名字空间

```

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

```

3.2、ClusterRole 示例

ClusterRole 有若干用法。你可以用它来：

1. 定义对某名字空间域对象的访问权限，并将在各个名字空间内完成授权；
2. 为名字空间作用域的对象设置访问权限，并跨所有名字空间执行授权；
3. 为集群作用域的资源定义访问权限。

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" 被忽略，因为 ClusterRoles 不受名字空间限制
  name: secret-reader
rules:
- apiGroups: ["" ]
  # 在 HTTP 层面，用来访问 Secret 对象的资源的名称为 "secrets"
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]

```

3.3、RoleBinding 示例

下面的例子中的 RoleBinding 将 "pod-reader" Role 授予在 "default" 名字空间中的用户 "jane"。这样，用户 "jane" 就具有了读取 "default" 名字空间中 pods 的权限。

```

apiVersion: rbac.authorization.k8s.io/v1
# 此角色绑定允许 "jane" 读取 "default" 名字空间中的 Pods
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
# 你可以指定不止一个“subject（主体）”
- kind: User # 这里可以是User,Group,ServiceAccount
  name: jane # "name" 是不区分大小写的
  apiGroup: rbac.authorization.k8s.io
roleRef:
# "roleRef" 指定与某 Role 或 ClusterRole 的绑定关系
kind: Role # 此字段必须是 Role 或 ClusterRole

```

```
name: pod-reader      # 此字段必须与你要绑定的 Role 或 ClusterRole 的名称匹配
apiGroup: rbac.authorization.k8s.io
```

3.3.1、RoleBinding 引用 ClusterRole

RoleBinding 也可以引用 ClusterRole，以将对应 ClusterRole 中定义的访问权限授予 RoleBinding 所在名字空间的资源。这种引用使得你可以跨整个集群定义一组通用的角色，之后在多个名字空间中复用！

下面的例子中的 RoleBinding，将"secret-reader" ClusterRole授予在"development" namespace中的用户"dave"。这样，用户 "dave" 就具有了读取 "development" 名字空间中 pods 的权限。

```
apiVersion: rbac.authorization.k8s.io/v1
# 此角色绑定使得用户 "dave" 能够读取 "default" 名字空间中的 Secrets
# 你需要一个名为 "secret-reader" 的 ClusterRole
kind: RoleBinding
metadata:
  name: read-secrets
  # RoleBinding 的名字空间决定了访问权限的授予范围。
  # 这里仅授权在 "development" 名字空间内的访问权限。
  namespace: development
subjects:
- kind: User
  name: dave # 'name' 是不区分大小写的
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

3.4、ClusterRoleBinding 示例

要跨整个集群完成访问权限的授予，你可以使用一个 ClusterRoleBinding。下面的 ClusterRoleBinding 允许 "manager" 组内的所有用户访问任何名字空间中的 Secrets。

```
apiVersion: rbac.authorization.k8s.io/v1
# 此集群角色绑定允许 "manager" 组中的任何人访问任何名字空间中的 secrets
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager # 'name' 是不区分大小写的
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

注意：

创建了绑定之后，你不能再修改绑定对象所引用的 Role 或 ClusterRole。试图改变绑定对象的 `roleRef` 将导致合法性检查错误。如果你想要改变现有绑定对象中 `roleRef` 字段的内容，必须删除重新创建绑定对象。

这种限制有两个主要原因：

1. 针对不同角色的绑定是完全不一样的绑定。要求通过删除/重建绑定来更改 `roleRef`，这样可以确保要赋予绑定的所有主体会被授予新的角色（而不是在允许修改 `roleRef` 的情况下导致所有现有主体胃镜验证即被授予新角色对应的权限）。
2. 将 `roleRef` 设置为不可以改变，这使得可以为用户授予对现有绑定对象的 `update` 权限，这样可以让他们管理主体列表，同时不能更改被授予这些主体的角色。

命令 `kubectl auth reconcile` 可以创建或者更新包含 RBAC 对象的清单文件，并且在必要的情况下删除和重新创建绑定对象，以改变所引用的角色

3.5、聚合的 ClusterRole

下面是一个聚合 ClusterRole 的示例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: monitoring
aggregationRule:
  clusterRoleSelectors:
  - matchLabels:
      rbac.example.com/aggregate-to-monitoring: "true"
rules: [] # 控制面自动填充这里的规则

-----
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: monitoring-endpoints
  labels:
    rbac.example.com/aggregate-to-monitoring: "true"
# 当你创建 "monitoring-endpoints" ClusterRole 时,
# 下面的规则会被添加到 "monitoring" ClusterRole 中
rules:
- apiGroups: ["" ]
  resources: ["services", "endpoints", "pods"]
  verbs: ["get", "list", "watch"]
```

下面的 ClusterRoles 让默认角色 "admin" 和 "edit" 拥有管理自定义资源 "CronTabs" 的权限，"view" 角色对 CronTab 资源拥有读操作权限。你可以假定 CronTab 对象在 API 服务器所看到的 URL 中被命名为 `"crontabs"`。

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

```
name: aggregate-cron-tabs-edit
labels:
  # 添加以下权限到默认角色 "admin" 和 "edit" 中
  rbac.authorization.k8s.io/aggregate-to-admin: "true"
  rbac.authorization.k8s.io/aggregate-to-edit: "true"
rules:
- apiGroups: ["stable.example.com"]
  resources: ["crontabs"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: aggregate-cron-tabs-view
labels:
  # 添加以下权限到 "view" 默认角色中
  rbac.authorization.k8s.io/aggregate-to-view: "true"
rules:
- apiGroups: ["stable.example.com"]
  resources: ["crontabs"]
  verbs: ["get", "list", "watch"]
```

Dashboard使用用户名密码形式登录

1、基本说明

在生产环境使用k8s以后，大部分应用都实现了高可用，不仅降低了维护成本，也简化了很多应用的部署成本，但是同时也带来了诸多问题。比如开发可能需要查看自己的应用状态、连接信息、日志、执行命令等。

使用k8s后，业务应用以Pod为单位，不像之前的以服务器为单位，可以直接通过登录服务器进行相关操作。当业务应用使用k8s部署后，k8s官方的dashboard虽然可以进行查看日志、执行命令等基本操作，但是作为运维人员，不想让开发操作或查看自己范围之外的Pod，此时就要使用RBAC进行相关的权限配置。

本文章主要讲解两方面的问题：

-
- - 使用用户名密码登录Dashboard
 - 对已登录用户进行权限配置，实现只能操作自己Namespace的Pod，不能进入到未授权的其他Namespace

2、更改Dashboard认证方式

为了方便开发和运维人员登录Dashboard，需要将Dashboard登录方式用户名密码认证（用户名密码和Token可以同时开启）。

使用Ratel将kubernetes-dashboard的deployment的--authentication-mode改成basic即可，未安装Ratel的可以使用kubectl edit进行更改，更改完成会自动重启。

	kubernetes-dashboard
	10.120.116.178/system-images/kubernetes-dashboard-amd64:v2.0.0-beta1
	多个Command逗号分隔, 如: sh,-c,/usr/sbin/nginx -g 'daemon off;'
	:server.key,--tls-cert-file=server.pem,--token-ttl=21600,--authentication-mode=basic,--namespace=kubern
est&limit	Request: 0Limit: 0

之后更改kube-apiserver配置添加

```
vim /usr/lib/systemd/system/kube-apiserver.service

#增加如下行

--basic-auth-file=/etc/kubernetes/basic_auth_file
```

basic_auth_file为存储账号密码的文件，格式如下：

```
xxx1_2019,xxx1,3,"system:authentication"
xxx2_2019,xxx2,4,"system:authentication"
xxx3_2019,xxx3,5,"system:authentication"
xxx4_2019,xxx4,6,"system:authentication"
```

依次是密码、用户名、ID号、用户组，因为下面会为已登录的用户进行授权，所以把组设置成了system:authentication，按需更改。

```
systemctl daemon-reload
systemctl restart kube-apiserver
```

3、添加默认权限

首先配置一个system:authentication组允许查询namespace列表（因为进入到指定namespace，必须能list该集群的namespace）：

```
mkdir -p /root/rbac && cd /root/rbac
vim namespaceReadonly.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
    rbac.authorization.k8s.io/aggregate-to-edit: "true"
  name: ratel-namespace-readonly
rules:
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - metrics.k8s.io
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: ratel-namespace-readonly
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ratel-namespace-readonly
subjects:
- apiGroup: rbac.authorization.k8s.io
```

```
kind: Group
name: system:authentication
```

```
kubectl create -f namespaceReadonly.yaml
```

创建查看namespace资源的权限

```
vim view-resource.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: ratel-resource-readonly
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - endpoints
  - persistentvolumeclaims
  - pods
  - replicationcontrollers
  - replicationcontrollers/scale
  - serviceaccounts
  - services
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - bindings
  - events
  - limitranges
  - namespaces/status
  - pods/log
  - pods/status
  - replicationcontrollers/status
  - resourcequotas
  - resourcequotas/status
  verbs:
  - get
  - list
```



```
- watch
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - apps
  resources:
  - controllerrevisions
  - daemonsets
  - deployments
  - deployments/scale
  - replicaset
  - replicaset/scale
  - statefulset
  - statefulset/scale
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - autoscaling
  resources:
  - horizontalpodautoscalers
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - batch
  resources:
  - cronjobs
  - jobs
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - extensions
  resources:
  - daemonsets
  - deployments
  - deployments/scale
  - ingresses
  - networkpolicies
```

```

- replicaset
- replicaset/scale
- replicationcontrollers/scale
verbs:
- get
- list
- watch
- apiGroups:
  - policy
resources:
- poddisruptionbudgets
verbs:
- get
- list
- watch
- apiGroups:
  - networking.k8s.io
resources:
- networkpolicies
verbs:
- get
- list
- watch
- apiGroups:
  - metrics.k8s.io
resources:
- pods
verbs:
- get
- list
- watch

```

```
kubectl create -f view-resource.yaml
```

创建Pod执行权限

```
vim podExec.yaml
```

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: ratel-pod-exec
rules:
- apiGroups:
  - ""
resources:
- pods
- pods/log

```

```
verbs:
- get
- list
- apiGroups:
- ""
resources:
- pods/exec
verbs:
- create
```

创建Pod删除权限

```
vim podDelete.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: ratel-pod-delete
rules:
- apiGroups:
- ""
resources:
- pods
verbs:
- get
- list
- delete
```

上述权限创建完成后，只需要将对应的用户绑定对应的权限即可实现不同的用户在不同的namespace实现不同的权限。

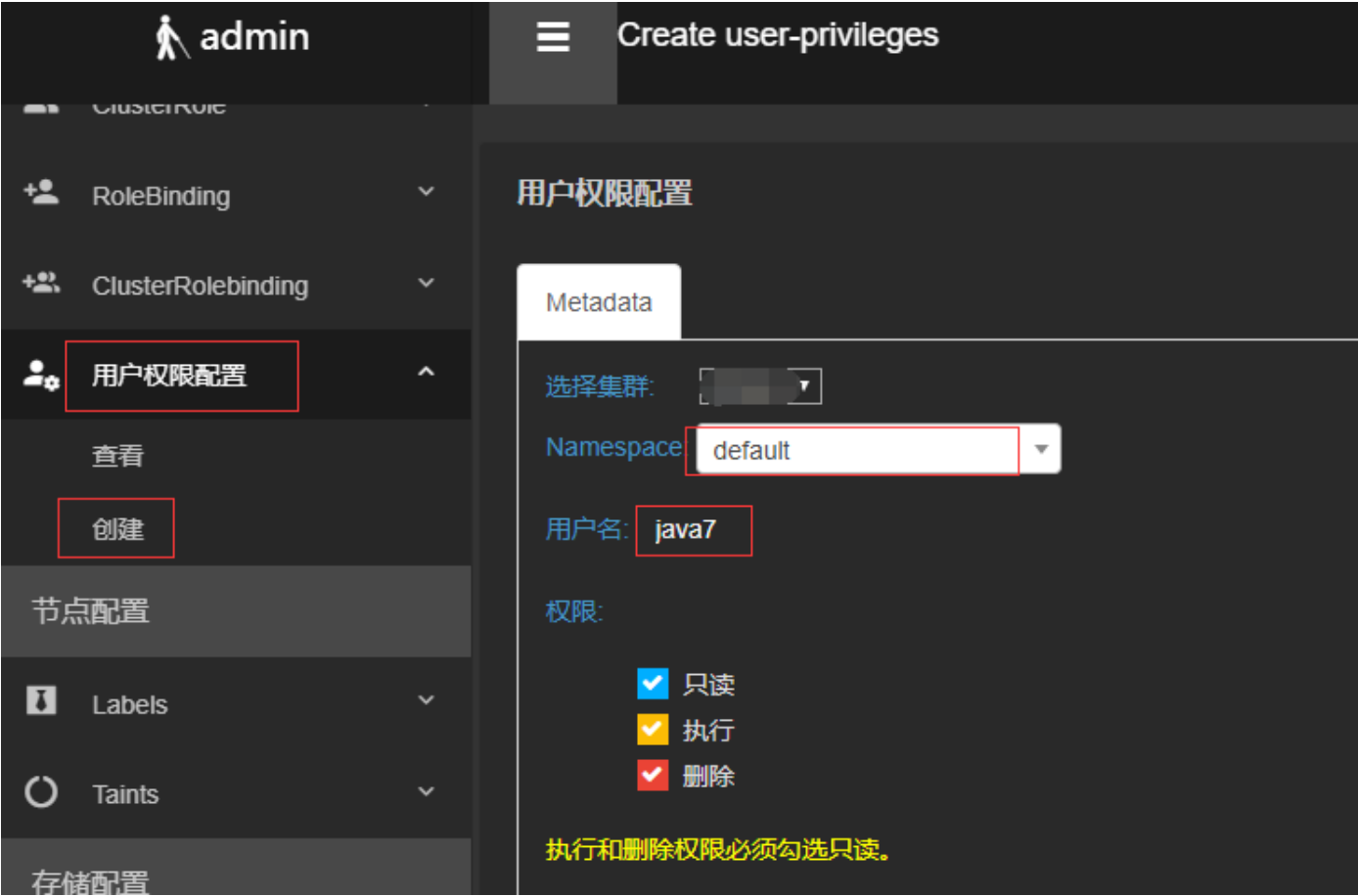
对RBAC不熟悉的可以参考<https://www.cnblogs.com/dukuan/p/9948063.html>

4、配置权限

案例：假设有一个用户叫java7，需要访问default命名空间下的资源，可以在容器执行命令和查看日志
添加权限之前是不能查看任何信息的：

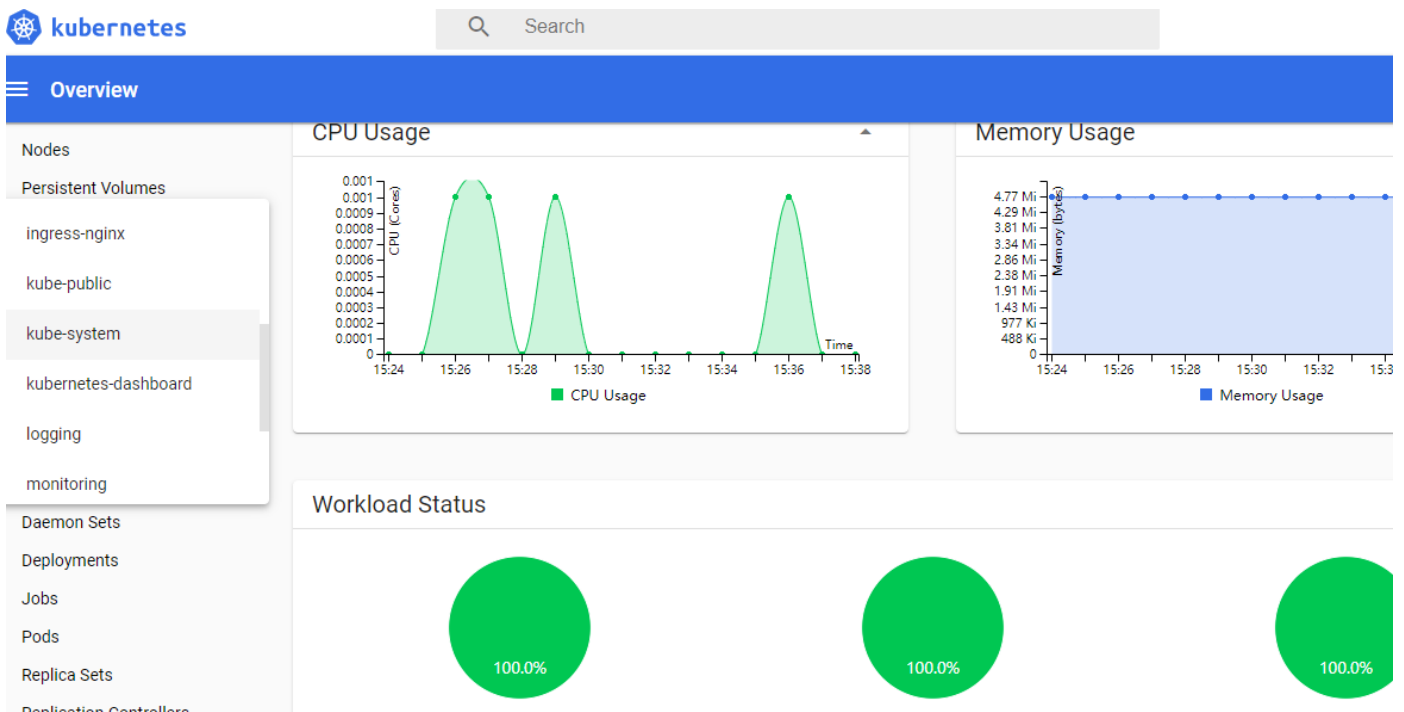
方式1

方式一：使用Ratel一键配置，选择对应的集群、Namespace、用户名、勾选权限点击创建即可。





创建成功后再次登录，即可查看该Namespace的信息



查看日志：

Logs from fortio ▾ in fortio-deploy-8ddb4b96d-cckv5 ▾

```
Fortio 1.0.1 grpc 'ping' server listening on [::]:8079
Fortio 1.0.1 https redirector server listening on [::]:8081
Fortio 1.0.1 echo server listening on [::]:8080
UI started - visit:
http://localhost:8080/fortio/ (or any host/ip reachable on this server)
04:21:40 I fortio_main.go:212> All fortio 1.0.1 2018-06-28 22:11 1e338e499d8ae134dcc32ad63825e7b972158687 go1.10.3 servers started!
```

执行命令：

Shell in

```
bash-4.3$ pwd
/home/tomcat
bash-4.3$
```

同时也不能查看其他namespace的资源

Overview

Nodes

Persistent Volumes

Storage Classes

Namespace

kube-system ▾

Overview

Workloads

There is nothing to display here

You can [deploy a containerized app](#), select other namespace or [take the Dashboard Tour](#) to learn more.

方式2

使用yaml文件配置

使用Ratel进行权限配置，在配置权限后在对应的namespace下创建对应的RoleBinding，如下：

```
[root@k8s-master01 ~]# kubectl get rolebinding
NAME                                     AGE
gitlab                                 112d
ratel-pod-delete-java7                 11m
ratel-pod-exec-java7                  11m
ratel-resource-readonly-java7         11m
```

内容如下:

```
source-readonly-java7 -o yaml
apiVersion: v1
items:
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2019-12-03T07:34:24Z"
    name: ratel-pod-delete-java7
    namespace: default
    resourceVersion: "35887290"
    selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/default/rolebindings/ratel-
pod-delete-java7
    uid: 547f5d42-159f-11ea-b1b5-001e674e3dd6
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: ratel-pod-delete
  subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: java7
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2019-12-03T07:34:24Z"
    name: ratel-pod-exec-java7
    namespace: default
    resourceVersion: "35887289"
    selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/default/rolebindings/ratel-
pod-exec-java7
    uid: 547c5768-159f-11ea-b1b5-001e674e3dd6
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: ratel-pod-exec
  subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: java7
```

```

- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    creationTimestamp: "2019-12-03T07:34:24Z"
    name: ratel-resource-readonly-java7
    namespace: default
    resourceVersion: "35887288"
    selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/default/rolebindings/ratel-
resource-readonly-java7
    uid: 5476577f-159f-11ea-b1b5-001e674e3dd6
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: ratel-resource-readonly
  subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: java7
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""

```

在没有安装Ratel的情况下，可以使用上述yaml内容直接创建至对应的namespace下即可完成权限配置。

上述只是实现了对常用资源的权限控制，其他权限控制类似。

Kubernetes多集群资源管理平台Ratel安装可以参考：<https://github.com/dotbalo/ratel-doc>