

## 在 10 章: k8s 认证、授权、准入控制

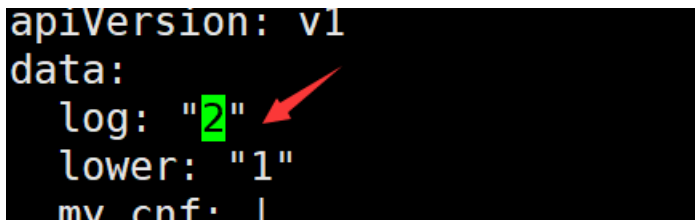
本节所讲内容:

- 10.1 k8s 安全管理: 认证、授权、准入控制概述
- 10.2 通过 token 令牌登录 dashboard
- 10.3 通过 kubeconfig 登录 dashboard
- 10.4 限制用户操作 k8s 资源

### 实战 1、Configmap 热更新

```
[root@xuegod63 ~]# kubectl edit configmap mysql
```

把 logs: "1" 变成 log: "2"



```
apiVersion: v1
data:
  log: "2"
  lower: "1"
  my.cnf: |
```

保存退出

```
[root@xuegod63 ~]# kubectl exec -it mysql-pod-volume -- /bin/sh
```

```
/ # cat /tmp/config/log
```

```
2
```

#发现 log 值变成了 2, 更新生效了

注意:

更新 ConfigMap 后:

使用该 ConfigMap 挂载的 Env 不会同步更新

使用该 ConfigMap 挂载的 Volume 中的数据需要一段时间 (实测大概 10 秒) 才能同步更新

ENV 是在容器启动的时候注入的, 启动之后 kubernetes 就不会再改变环境变量的值, 且同一个 namespace 中的 pod 的环境变量是不断累加的, 为了更新容器中使用 ConfigMap 挂载的配置, 可以通过滚动更新 pod 的方式来强制重新挂载 ConfigMap, 也可以在更新了 ConfigMap 后, 先将副本数设置为 0, 再扩容 pod

### 实战 2、Secret 概述

#### 1 Secret 是什么?

上面我们学习的 Configmap 一般是用来存放明文数据的, 如配置文件, 对于一些敏感数据, 如密码、私钥等数据时, 要用 secret 类型。

Secret 解决了密码、token、秘钥等敏感数据的配置问题, 而不需要把这些敏感数据暴露到镜像或

者 Pod Spec 中。Secret 可以以 Volume 或者环境变量的方式使用。

要使用 secret, pod 需要引用 secret。Pod 可以用两种方式使用 secret: 作为 volume 中的文件被挂载到 pod 中的一个或者多个容器里, 或者当 kubelet 为 pod 拉取镜像时使用。

secret 可选参数有三种:

generic: 通用类型, 通常用于存储密码数据。

tls: 此类型仅用于存储私钥和证书。

docker-registry: 若保存 docker 仓库的认证信息的话, 就必须使用此种类型来创建。

Secret 类型:

Service Account: 用于被 serviceaccount 引用。serviceaccount 创建时 Kubernetes 会默认创建对应的 secret。Pod 如果使用了 serviceaccount, 对应的 secret 会自动挂载到 Pod 的 /run/secrets/kubernetes.io/serviceaccount 目录中。

Opaque: base64 编码格式的 Secret, 用来存储密码、秘钥等。可以通过 base64 --decode 解码获得原始数据, 因此安全性弱

kubernetes.io/dockerconfigjson: 用来存储私有 docker registry 的认证信息。

## 2 使用 Secret

### 1、通过环境变量引入 Secret

#把 mysql 的 root 用户的 password 创建成 secret

```
[root@xuegod63 ~]# kubectl create secret generic mysql-password --from-literal=password=xuegodpod**lucky66
```

```
[root@xuegod63 ~]# kubectl get secret
```

| NAME           | TYPE   | DATA | AGE |
|----------------|--------|------|-----|
| mysql-password | Opaque | 1    | 30s |

```
[root@xuegod63 ~]# kubectl describe secret mysql-password
```

Name: mysql-password

Namespace: default

Labels: <none>

Annotations: <none>

Type: Opaque

Data

====

password: 18 bytes

#password 的值是加密的,

#但 secret 的加密是一种伪加密, 它仅仅是将数据做了 base64 的编码。

#创建 pod, 引用 secret

```
[root@xuegod63 ~]# cat pod-secret.yaml
```

apiVersion: v1

```
kind: Pod
metadata:
  name: pod-secret
  labels:
    app: myapp
spec:
  containers:
  - name: myapp
    image: ikubernetes/myapp:v1
    ports:
    - name: http
      containerPort: 80
    env:
    - name: MYSQL_ROOT_PASSWORD #它是 Pod 启动成功后,Pod 中容器的环境变量名.
      valueFrom:
        secretKeyRef:
          name: mysql-password #这是 secret 的对象名
          key: password #它是 secret 中的 key 名
[root@xuegod63 ~]# kubectl apply -f pod-secret.yaml
[root@xuegod63 ~]# kubectl exec -it pod-secret -- /bin/sh
/ # printenv
MYSQL_ROOT_PASSWORD=xuegodpod**lucky66
```

## 2、通过 volume 挂载 Secret

### 1) 创建 Secret

手动加密, 基于 base64 加密

```
[root@xuegod63 ~]# echo -n 'admin' | base64
```

YWRtaW4=

```
[root@xuegod63 ~]# echo -n 'xuegod123456f' | base64
```

eHVIZ29kMTIzNDU2Zg==

解码:

```
[root@xuegod63 ~]# echo eHVIZ29kMTIzNDU2Zg== | base64 -d
```

### 2) 创建 yaml 文件

```
[root@xuegod63 ~]# vim secret.yaml
```

apiVersion: v1

kind: Secret

metadata:

name: mysecret

type: Opaque

data:

username: YWRtaW4=

password: eHVIZ29kMTIzNDU2Zg==

```
[root@xuegod63 ~]# kubectl apply -f secret.yaml
```

3) 将 Secret 挂载到 Volume 中

```
[root@xuegod63 ~]# vim pod_secret_volume.yaml
```

apiVersion: v1

kind: Pod

metadata:

name: pod-secret-volume

spec:

containers:

- name: myapp

image: registry.cn-beijing.aliyuncs.com/google\_registry/myapp:v1

volumeMounts:

- name: secret-volume

mountPath: /etc/secret

readOnly: true

volumes:

- name: secret-volume

secret:

secretName: mysecret

```
[root@xuegod63 ~]# kubectl apply -f pod_secret_volume.yaml
```

```
[root@xuegod63 ~]# kubectl exec -it pod-secret-volume -- /bin/sh
```

```
/ # ls /etc/secret
```

```
password username
```

```
/ #
```

```
/ # cat /etc/secret/username
```

```
admin/ #
```

```
/ #
```

```
/ # cat /etc/secret/password
```

```
xuegod123456f/ #
```

#由上可见, 在 pod 中的 secret 信息实际已经被解密。

## 10.1 k8s 安全管理: 认证、授权、准入控制概述

k8s 对我们整个系统的认证, 授权, 访问控制做了精密的设置; 对于 k8s 集群来说, apiserver 是整个就集群访问控制的唯一入口, 我们在 k8s 集群之上部署应用程序的时候, 也可以通过宿主机的 NodePort 暴露的端口访问里面的程序, 用户访问 kubernetes 集群需要经历如下认证过程: 认证 -> 授权 -> 准入控制 (admissioncontroller)

1.认证(Authenticating)是对客户端的认证,通俗点就是用户名密码验证

2.授权(Authorization)是对资源的授权, k8s 中的资源无非是容器,最终其实就是容器的计算,网络,存储资源,当一个请求经过认证后,需要访问某一个资源(比如创建一个 pod),授权检查会根据授权规则判定该资源(比如某 namespace 下的 pod)是否是该客户可访问的。

3.准入(Admission Control)机制:

准入控制器 (Admission Controller) 位于 API Server 中,在对象被持久化之前,准入控制器拦截对 API Server 的请求,一般用来做身份验证和授权。其中包含两个特殊的控制器:

MutatingAdmissionWebhook 和 ValidatingAdmissionWebhook。分别作为配置的变异和验证准入控制 webhook。

变更 (Mutating) 准入控制: 修改请求的对象

验证 (Validating) 准入控制: 验证请求的对象

准入控制器是在 API Server 的启动参数重配置的。一个准入控制器可能属于以上两者中的一种,也可能两者都属于。当请求到达 API Server 的时候首先执行变更准入控制,然后再执行验证准入控制。

我们在部署 Kubernetes 集群的时候都会默认开启一系列准入控制器,如果没有设置这些准入控制器的话可以说你的 Kubernetes 集群就是在裸奔,应该只有集群管理员可以修改集群的准入控制器。

例如我会默认开启如下的准入控制器。

--admission-

control=ServiceAccount,NamespaceLifecycle,NamespaceExists,LimitRanger,ResourceQuota,MutatingAdmissionWebhook,ValidatingAdmissionWebhook

k8s 的整体架构也是一个微服务的架构,所有的请求都是通过一个 GateWay,也就是 kube-apiserver 这个组件(对外提供 REST 服务),k8s 中客户端有两类,一种是普通用户,一种是集群内的 Pod,这两种客户端的认证机制略有不同,但无论是哪一种,都需要依次经过认证,授权,准入这三个机制。

#### 10.1.1 认证

##### 1、认证支持多种插件

###### (1) 令牌 (token) 认证:

双方有一个共享密钥,服务器上先创建一个密码,存下来,客户端登陆的时候拿这个密码登陆即可,这个就是对称密钥认证方式; k8s 提供了一个 restful 风格的接口,它的所有服务都是通过 http 协议提供的,因此认证信息只能经由 http 协议的认证首部进行传递,这种认证首部进行传递通常叫做令牌;

###### (2) ssl 认证:

对于 k8s 访问来讲,ssl 认证能让客户端确认服务器的认证身份,我们在跟服务器通信的时候,需要服务器发过来一个证书,我们需要确认这个证书是不是 ca 签署的,如果是我们认可的 ca 签署的,里面的 subj 信息与我们访问的目标主机信息保持一致,没有问题,那么我们就认为服务器的身份得

到认证了, k8s 中最重要的是服务器还需要认证客户端的信息, kubectl 也应该有一个证书, 这个证书也是 server 所认可的 ca 签署的证书, 双方需要互相认证, 实现加密通信, 这就是 ssl 认证。

## 2、kubernetes 上的账号

客户端对 apiserver 发起请求, apiserver 要识别这个用户是否有请求的权限, 要识别用户本身能否通过 apiserver 执行相应的操作, 那么需要哪些信息才能识别用户信息来完成对用户的相关的访问控制呢?

kubectl explain pods.spec 可以看到有一个字段 serviceAccountName (服务账号名称), 这个就是我们 pod 连接 apiserver 时使用的账号, 因此整个 kubernetes 集群中的账号有两类, ServiceAccount (服务账号), User account (用户账号)

User account: 实实在在现实中的人, 人可以登陆的账号, 客户端想要对 apiserver 发起请求, apiserver 要识别这个客户端是否有请求的权限, 那么不同的用户就会有不同的权限, 靠用户账号表示, 叫做 username

ServiceAccount: 方便 Pod 里面的进程调用 Kubernetes API 或其他外部服务而设计的, 是 kubernetes 中的一种资源

sa 账号: 登陆 dashboard 使用的账号

user account: 这个是登陆 k8s 物理机器的用户

### 1.ServiceAccount

Service account 是为了方便 Pod 里面的进程调用 Kubernetes API 或其他外部服务而设计的。它与 User account 不同, User account 是为人设计的, 而 service account 则是为 Pod 中的进程调用 Kubernetes API 而设计; User account 是跨 namespace 的, 而 service account 则是仅局限它所在的 namespace; 每个 namespace 都会自动创建一个 default service account; 开启 ServiceAccount Admission Controller 后

1) 每个 Pod 在创建后都会自动设置 spec.serviceAccount 为 default (除非指定了其他 ServiceAccount)

2) 验证 Pod 引用的 service account 已经存在, 否则拒绝创建;

当创建 pod 的时候, 如果没有指定一个 serviceaccount, 系统会自动在与该 pod 相同的 namespace 下为其指派一个 default service account。而 pod 和 apiserver 之间进行通信的账号, 称为 serviceAccountName。如下:

kubectl get pods

[root@xuegod63 ~]# kubectl get pods

| NAME                            | READY | STATUS  | RESTARTS | AGE |
|---------------------------------|-------|---------|----------|-----|
| mysql-pod-volume                | 1/1   | Running | 0        | 43m |
| nfs-provisioner-cd5589cfc-c8vc5 | 1/1   | Running | 0        | 13h |
| pod-secret                      | 1/1   | Running | 0        | 18m |
| web-0                           | 1/1   | Running | 0        | 13h |

web-1                      1/1      Running      0              13h

```
[root@xuegod63 ~]# kubectl get pods web-0 -o yaml | grep "serviceAccountName"
serviceAccountName: default
```

```
[root@xuegod63 ~]# kubectl describe pods web-0
```

Volumes:

www:

Type: PersistentVolumeClaim

ClaimName: www-web-0

default-token-cq5qp:

Type: Secret (a volume populated by a Secret)

SecretName: default-token-cq5qp

Optional: false

从上面可以看到每个 Pod 无论定义与否都会有个存储卷，这个存储卷为 default-token-\*\*\*的 token 令牌，这就是 pod 和 serviceaccount 认证信息。通过 secret 进行定义，由于认证信息属于敏感信息，所以需要保存在 secret 资源当中，并以存储卷的方式挂载到 Pod 当中。从而让 Pod 内运行的应用通过对应的 secret 中的信息来连接 apiserver，并完成认证。每个 namespace 中都有一个默认的叫 default 的 serviceaccount 资源。进行查看名称空间内的 secret，也可以看到对应的 default-token。让当前名称空间中所有的 pod 在连接 apiserver 时可以使用的预制认证信息，从而保证 pod 之间的通信。

```
[root@xuegod63 ~]# kubectl get sa
```

| NAME    | SECRETS | AGE |
|---------|---------|-----|
| default | 1       | 12d |

```
[root@xuegod63 ~]# kubectl get secret
```

| NAME                | TYPE                                | DATA | AGE |
|---------------------|-------------------------------------|------|-----|
| default-token-cq5qp | kubernetes.io/service-account-token | 3    | 12d |

默认的 service account 仅仅只能获取当前 Pod 自身的相关属性，无法观察到其他名称空间 Pod 的相关属性信息。如果想要扩展 Pod，假设有一个 Pod 需要用于管理其他 Pod 或者是其他资源对象，是无法通过自身的名称空间的 serviceaccount 进行获取其他 Pod 的相关属性信息的，此时就需要进行手动创建一个 serviceaccount，并在创建 Pod 时进行定义。那么 serviceaccount 该如何进行定义呢？实际上，service account 也属于一个 k8s 资源，serviceAccount 也属于标准的 k8s 资源，可以创建一个 serviceAccount，创建之后由我们创建的 pod 使用 serviceAccountName 去加载自己定义的 serviceAccount 就可以了，如下查看 service account 的定义方式：

(1) 创建一个 serviceaccount:

```
kubectl create serviceaccount test
```

```
kubectl get sa 显示如下:
```

|      |   |     |
|------|---|-----|
| test | 1 | 31s |
|------|---|-----|

可以看到已经创建了 test 的 serviceaccount



kubectl describe sa test 查看 test 这个账号的详细信息

```
Name: test
Namespace: default
Labels: <none>
Annotations: <none>
Image pull secrets: <none>
Mountable secrets: test-token-hnc57
Tokens: test-token-hnc57
Events: <none>
```

上面可以看到生成了一个 test-token-hnc57 的 secret 和 test-token-hnc57 的 token

kubectl get secret 显示如下:

```
test-token-hnc57          kubernetes.io/service-account-token    3          4m3s
```

kubectl describe secret test-token-hnc57 显示如下:

```
Name: test-token-hnc57
Namespace: default
Labels: <none>
Annotations: kubernetes.io/service-account.name: test
              kubernetes.io/service-account.uid: 35a40baf-67f5-11e9-a2e8-005056010488

Type: kubernetes.io/service-account-token

Data
====
ca.crt: 1025 bytes
namespace: 7 bytes
token: eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJ1cm5ldGVzL3N1cnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWN1YWNjb3VudC9zZW50IiwiaWF0Ij0iY2VhY2NvdW50LnVpZCI6IjM1YTQwYmFmLTU3ZjU0MTF0S1hMmU4LTAwNTA1NjAxMDQ0OCIsInN1YiI6InN5c3R1bTpsZXJ2aWN1YWNjb3VudDpkZWZhdWx0OnRlc3QifQ.Vwaohzx4p0rxI20uGnJ6g9ack5rbvdOKjLUIIE9U-c7y92XtHRXNgejbFIWmx1h5K5TnWvuqmTCA5sF_yjD85NwNth7t1TsL0Jf0AW4BVF9g1b1K7_j3h0250y0Zy6jvAWa-SY4AGVyhUzTdrtQI78Qz9QWNnO7tMmPghyFt-Psh3XftVthIGb5V5v4h2N7U4nzdRtKQONyZypIfIGtw5n3bIaShbBf-2BN6QBgyUNjE242j5ny4yc9_Ae5pD6gJzesvVXzt5aAPuGOPL1s6EL4a3a_awYIrXQheJ-4UIdh0zTm7MCZxhI7wJrT_QdaPcH_CF3CJMGsqOHIFwuj9Q
```

上面可以看到生成了 test-token-hnc57 的 token 详细信息, 这个账号就是 sa 连接 apiserver 的账号, 这个 token 是登陆 k8s 的 token, 这些是一个认证信息, 能够登陆 k8s, 能认证到 k8s, 但是不能做别的事情, 不代表权限, 想要做其他事情, 需要授权

## 2、kubeconfig 文件

在 K8S 集群当中, 每一个用户对资源的访问都是需要通过 apiserver 进行通信认证才能进行访问的, 那么在此机制当中, 对资源的访问可以是 token, 也可以是通过配置文件的方式进行保存和使用认证信息, 可以通过 kubectl config 进行查看配置, 如下:

kubectl config view

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.40.199:6443    #apiserver 的地址
    name: kubernetes                      #集群的名字
contexts:
- context:
```



```
cluster: kubernetes
user: kubernetes-admin
name: kubernetes-admin@kubernetes    #上下文的名字
current-context: kubernetes-admin@kubernetes    #当前上下文的名字
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

在上面的配置文件当中，定义了集群、上下文以及用户。其中 Config 也是 K8S 的标准资源之一，在该配置文件当中定义了一个集群列表，指定的集群可以有多个；用户列表也可以有多个，指明集群中的用户；而在上下文列表当中，是进行定义可以使用哪个用户对哪个集群进行访问，以及当前使用的上下文是什么。

```
kubectl get pods --kubeconfig=./config
```

#### 10.1.2 授权

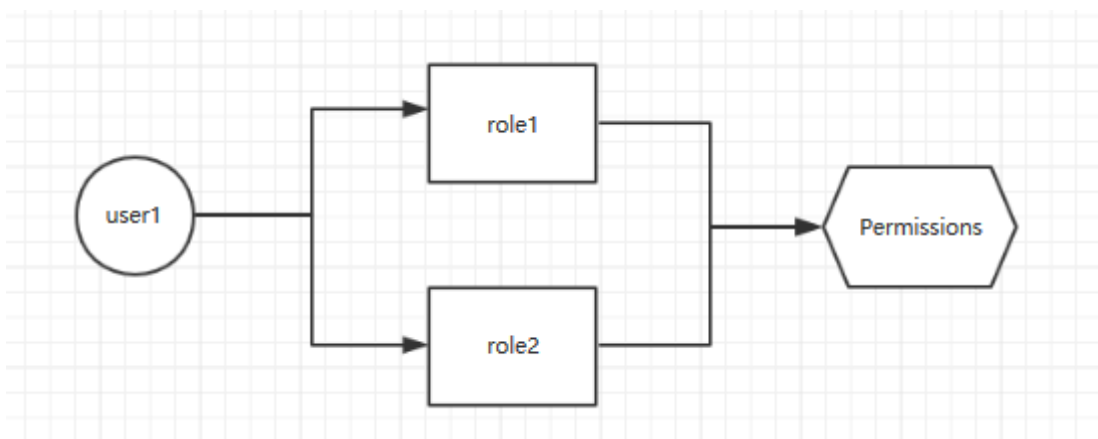
如果用户通过认证，什么权限都没有，需要一些后续的授权操作，如对资源的增删该查等，kubernetes1.6 之后开始有 RBAC（基于角色的访问控制机制）授权检查机制。

Kubernetes 的授权是基于插件形成的，其常用的授权插件有以下几种：

- 1) Node（节点认证）
- 2) ABAC(基于属性的访问控制)
- 3) RBAC（基于角色的访问控制）\*\*\*
- 4) Webhook（基于 http 回调机制的访问控制）

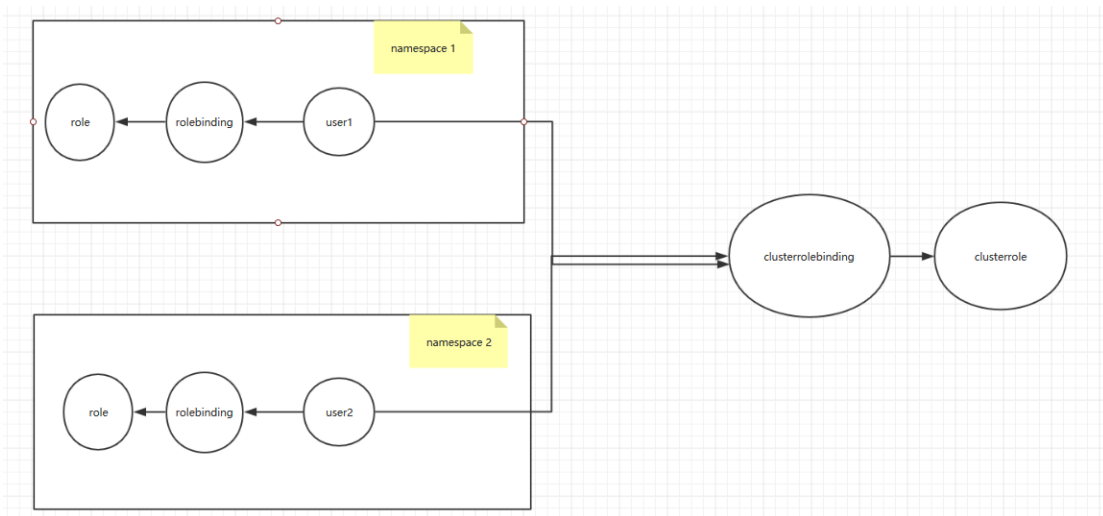
##### 什么是 RBAC（基于角色的访问控制）？

让一个用户（Users）扮演一个角色（Role），角色拥有权限，从而让用户拥有这样的权限，随后在授权机制当中，只需要将权限授予某个角色，此时用户将获取对应角色的权限，从而实现角色的访问控制。如图：



在 k8s 的授权机制当中, 采用 RBAC 的方式进行授权, 其工作逻辑是, 把对对象的操作权限定义到一个角色当中, 再将用户绑定到该角色, 从而使用户得到对应角色的权限。如果通过 rolebinding 绑定 role, 只能对 rolebinding 所在的名称空间的资源有限, 上图 user1 这个用户绑定到 role1 上, 只对 role1 这个名称空间的资源有权限, 对其他名称空间资源没有权限, 属于名称空间级别的;

另外, k8s 为此还有一种集群级别的授权机制, 就是定义一个集群角色 (ClusterRole), 对集群内的所有资源都有可操作的权限, 从而将 User2 通过 ClusterRoleBinding 到 ClusterRole, 从而使 User2 拥有集群的操作权限。Role、RoleBinding、ClusterRole 和 ClusterRoleBinding 的关系如下图:



通过上图可以看到, 可以通过 rolebinding 绑定 role, rolebinding 绑定 clusterrole, clusterrolebinding 绑定 clusterrole。

上面我们说了两个角色绑定:

- (1) 用户通过 rolebinding 绑定 role
- (2) 用户通过 clusterrolebinding 绑定 clusterrole

还有一种: rolebinding 绑定 clusterrole

假如有 6 个名称空间, 每个名称空间的用户都需要对自己的名称空间有管理员权限, 那么需要定义 6 个 role 和 rolebinding, 然后依次绑定, 如果名称空间更多, 我们需要定义更多的 role, 这个是很麻烦的, 所以我们引入 clusterrole, 定义一个 clusterrole, 对 clusterrole 授予所有权限, 然后用户通过 rolebinding 绑定到 clusterrole, 就会拥有自己名称空间的管理员权限了

注: RoleBinding 仅仅对当前名称空间有对应的权限。

### 10.1.3 准入控制

一般而言, 准入控制只是用来定义我们授权检查完成之后的后续的其他安全检查操作的, 进一步补充了授权机制, 由多个插件组合实行, 一般而言在创建, 删除, 修改或者做代理时做补充;

Kubernetes 的 Admission Control 实际上是一个准入控制器(Admission Controller)插件列表, 发送到 API Server 的请求都需要经过这个列表中的每个准入控制器插件的检查, 如果某一个控制器插件准入失败, 就准入失败。

控制器插件如下:

AlwaysAdmit: 允许所有请求通过

**AlwaysPullImages:** 在启动容器之前总是去下载镜像, 相当于每当容器启动前做一次用于是否有权使用该容器镜像的检查

**AlwaysDeny:** 禁止所有请求通过, 用于测试

**DenyEscalatingExec:** 拒绝 exec 和 attach 命令到有升级特权的 Pod 的终端用户访问。如果集中包含升级特权的容器, 而要限制终端用户在这些容器中执行命令的能力, 推荐使用此插件

**ImagePolicyWebhook**

**ServiceAccount:** 这个插件实现了 serviceAccounts 等等自动化, 如果使用 ServiceAccount 对象, 强烈推荐使用该插件

**SecurityContextDeny:** 将 Pod 定义中定义了的 SecurityContext 选项全部失效。

**SecurityContext** 包含在容器中定义了操作系统级别的安全选型如 fsGroup, selinux 等选项

**ResourceQuota:** 用于 namespace 上的配额管理, 它会观察进入的请求, 确保在 namespace 上的配额不超标。推荐将这个插件放到准入控制器列表的最后一个。ResourceQuota 准入控制器既可以限制某个 namespace 中创建资源的数量, 又可以限制某个 namespace 中被 Pod 请求的资源总量。ResourceQuota 准入控制器和 ResourceQuota 资源对象一起可以实现资源配额管理。

**LimitRanger:** 用于 Pod 和容器上的配额管理, 它会观察进入的请求, 确保 Pod 和容器上的配额不会超标。准入控制器 LimitRanger 和资源对象 LimitRange 一起实现资源限制管理

**NamespaceLifecycle:** 当一个请求是在一个不存在的 namespace 下创建资源对象时, 该请求会被拒绝。当删除一个 namespace 时, 将会删除该 namespace 下的所有资源对象

**DefaultStorageClass**

**DefaultTolerationSeconds**

**PodSecurityPolicy**

当 Kubernetes 版本  $\geq 1.6.0$ , 官方建议使用这些插件:

`--admission-`

`control=NamespaceLifecycle,LimitRanger,ServiceAccount,PersistentVolumeLabel,DefaultStorageClass,ResourceQuota,DefaultTolerationSeconds`

当 Kubernetes 版本  $\geq 1.4.0$ , 官方建议使用这些插件:

`--admission-`

`control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota`

以上是标准的准入插件, 如果是自己定制的话, k8s1.7 版 出了两个 alpha features, Initializers 和 External Admission Webhooks

## 实战-安装 kubernetes-dashboard-2.0

把安装 kubernetes-dashboard 需要的镜像上传到工作节点, 在工作节点节点按照如下方法通过 `docker load -i` 解压, 如果你的环境有多个工作节点, 需要把镜像上传到每个工作节点上, 然后 `docker load -i` 解压, 或者修改 kubernetes-dashboard.yaml 文件, 增加一个字段 `nodeName: xuegod64`, 让这个 dashboard 只调度到 xuegod64 节点, 镜像压缩包在课件里, 用如下方法手动解压:

```
[root@xuegod64~]# docker load -i dashboard_2_0_0.tar.gz
```

```
[root@xuegod64 ~]# docker load -i metrics-scraper-1-0-1.tar.gz
```

注意: 解压出来的镜像是 kubernetesui/dashboard:v2.0.0-beta8 和 kubernetesui/metrics-scraper:v1.0.1

### 1、安装 dashboard 组件

在 xuegod63 节点操作如下命令:

```
[root@xuegod63 ~]# kubectl apply -f kubernetes-dashboard.yaml
```

kubernetes-dashboard.yaml 文件在课件, 可在课件下载

## 2、查看 dashboard 的状态

```
[root@xuegod63 ~]# kubectl get pods -n kubernetes-dashboard
```

显示如下, 说明 dashboard 安装成功了

| NAME                                       | READY | STATUS  | RESTARTS | AGE |
|--|-------|---------|----------|-----|
| dashboard-metrics-scraper-7445d59dfd-vz7d6 | 1/1   | Running | 0        | 79s |
| kubernetes-dashboard-54f5b6dc4b-z5q9s      | 1/1   | Running | 0        | 79s |

## 3、查看 dashboard 前端的 service

```
[root@xuegod63 ~]# kubectl get svc -n kubernetes-dashboard
```

显示如下:

| NAME                      | TYPE      | CLUSTER-IP   | EXTERNAL-IP | PORT(S)  |
|---------------------------|-----------|--------------|-------------|----------|
| dashboard-metrics-scraper | ClusterIP | 10.10.26.97  | <none>      | 8000/TCP |
| kubernetes-dashboard      | ClusterIP | 10.10.103.49 | <none>      | 443/TCP  |

## 4、修改 service type 类型变成 NodePort

```
[root@xuegod63 ~]# kubectl edit svc kubernetes-dashboard -n kubernetes-dashboard
```

把 type: ClusterIP 变成 type: NodePort, 保存退出即可。

```
[root@xuegod63 ~]# kubectl get svc -n kubernetes-dashboard
```

显示如下:

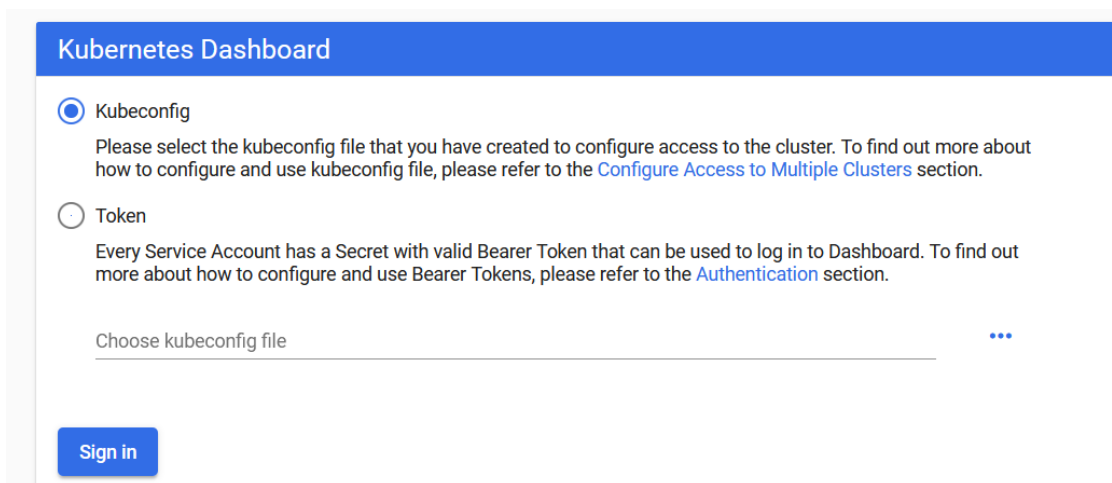
| NAME                      | TYPE      | CLUSTER-IP   | EXTERNAL-IP | PORT(S)       |
|---------------------------|-----------|--------------|-------------|---------------|
| dashboard-metrics-scraper | ClusterIP | 10.10.26.97  | <none>      | 8000/TCP      |
| kubernetes-dashboard      | NodePort  | 10.10.103.49 | <none>      | 443:30517/TCP |

上面可看到 service 类型是 NodePort, 访问 xuegod63 节点 ip:30517 端口即可访问 kubernetes dashboard, 在浏览器(最好使用火狐浏览器)访问如下地址:

<https://192.168.1.63:30517/>



可看到出现了 dashboard 界面



## 10.2 通过管理员 Token 登陆 dashboard

创建管理员 token，具有查看任何空间的权限，可以管理所有资源对象

```
[root@xuegod63 ~]# kubectl create clusterrolebinding dashboard-cluster-admin --  
clusterrole=cluster-admin --serviceaccount=kubernetes-dashboard:kubernetes-dashboard
```

查看 kubernetes-dashboard 名称空间下的 secret

```
[root@xuegod63 ~]# kubectl get secret -n kubernetes-dashboard
```

显示如下：

| NAME                            | TYPE                                | DATA |
|---------------------------------|-------------------------------------|------|
| default-token-pzpzl             | kubernetes.io/service-account-token | 3    |
| kubernetes-dashboard-certs      | Opaque                              | 0    |
| kubernetes-dashboard-csrf       | Opaque                              | 1    |
| kubernetes-dashboard-key-holder | Opaque                              | 2    |

kubernetes-dashboard-token-pdfj9 kubernetes.io/service-account-token 3

找到对应的带有 token 的 kubernetes-dashboard-token-ngcmg

```
[root@xuegod63 ~]# kubectl describe secret kubernetes-dashboard-token-pdfj9 -
```

n kubernetes-dashboard

显示如下:

...

...

token:

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImpfUDZGektXZmRIQ2luS1BKamlkNFpTMzgxMTdpR0Y2eXBla3U3UzR3V2MifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWVudC9uYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIslmt1YmVybmV0ZXMuaW8vc2VydmljZWJyY291bnQvc2VjcmV0Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZC10b2t1bi1qc2xnZCIslmt1YmVybmV0ZXMuaW8vc2VydmljZWJyY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIslmt1YmVybmV0ZXMuaW8vc2VydmljZWJyY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImYyNjUwMmJhLWE3OGEtNGFkYy1iYjRILTMxYzVmNDgzODBmZCIslmN1Yil6InN5c3RlbTppZXJ2aWVudC9uYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZDprdWJlcm5ldGVzLWRhc2hib2FyZCJ9.PMeFznMelrgMAWsCPYC-
```

```
q8Ooqzz2vcDbbZdqF7VwZl8w3gGafbm5cdYXXMiR7UM9Otj8gOrB6qjKNPmjFb799HrDKJ6Z5C2CX11Bj6HOVqjL6sQg_a8UqnUi8PxOI3HwAlyyixa8KvA-yzf-EqCWYyTdTRMkmFEMkan2bQ7Lr3ip0B1LxUSHhCRp_5M0xGaN6nem6WnAmN87xw5JF5uRz4sRmgZvEsG_2MEbFozq1W6PMX0MaUOQ5EpzYzQ1zvdgo6c26MwGTDf2UzlkeTEXyEaLLxBx9nUPLTzYFJ3D_QxMn_L4VgHEGQyPdEUFv5MK8Yb4jBXFMsX4fotJY32Q
```

记住 token 后面的值, 把下面的 token 值复制到浏览器 token 登陆处即可登陆:

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImpfUDZGektXZmRIQ2luS1BKamlkNFpTMzgxMTdpR0Y2eXBla3U3UzR3V2MifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWVudC9uYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIslmt1YmVybmV0ZXMuaW8vc2VydmljZWJyY291bnQvc2VjcmV0Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZC10b2t1bi1qc2xnZCIslmt1YmVybmV0ZXMuaW8vc2VydmljZWJyY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIslmt1YmVybmV0ZXMuaW8vc2VydmljZWJyY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImYyNjUwMmJhLWE3OGEtNGFkYy1iYjRILTMxYzVmNDgzODBmZCIslmN1Yil6InN5c3RlbTppZXJ2aWVudC9uYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZDprdWJlcm5ldGVzLWRhc2hib2FyZCJ9.PMeFznMelrgMAWsCPYC-
```

```
q8Ooqzz2vcDbbZdqF7VwZl8w3gGafbm5cdYXXMiR7UM9Otj8gOrB6qjKNPmjFb799HrDKJ6Z5C2CX11Bj6HOVqjL6sQg_a8UqnUi8PxOI3HwAlyyixa8KvA-yzf-EqCWYyTdTRMkmFEMkan2bQ7Lr3ip0B1LxUSHhCRp_5M0xGaN6nem6WnAmN87xw5JF5uRz4sRmgZvEsG_2MEbFozq1W6PMX0MaUOQ5EpzYzQ1zvdgo6c26MwGTDf2UzlkeTEXyEaLLxBx9nUPLTzYFJ3D_QxMn_L4VgHEGQyPdEUFv5MK8Yb4jBXFMsX4fotJY32Q
```

点击 sing in 登陆, 显示如下, 这次就可以看到和操作任何名称空间的资源了





## 10.4 通过 kubeconfig 登录 dashboard

把 token 令牌封装成 kubeconfig, 通过 kubeconfig 登陆 dashboard

1. 创建一个只能管理指定名称空间的 kubeconfig 文件

以下步骤在 k8s 的 master 节点操作

`cd /etc/kubernetes/pki`

(1) 创建 cluster

```
kubecttl config set-cluster kubernetes --certificate-authority=./ca.crt --
server="https://192.168.40.63:6443" --embed-certs=true --kubeconfig=/root/lucky-
admin.conf
```

(2) 创建 credentials 时需要使用上面我们创建的 token 信息

```
kubecttl get secret -n lucky
```

```
DEF_NS_ADMIN_TOKEN=$(kubecttl get secret lucky-admin-token-qrrrc -n lucky -o
jsonpath={.data.token})|base64 -d)
```

(3) 开始创建 credentials

```
kubecttl config set-credentials lucky --token=$DEF_NS_ADMIN_TOKEN --
kubeconfig=/root/lucky-admin.conf
```

(4) 创建 context

```
kubecttl config set-context lucky@kubernetes --cluster=kubernetes --user=lucky --
kubeconfig=/root/lucky-admin.conf
```

(5) 切换 context 的 current-context 是 lucky@kubernetes

```
kubecttl config use-context lucky@kubernetes --kubeconfig=/root/lucky-admin.conf
```

(6) 把刚才的 kubeconfig 文件 lucky-admin.conf 复制到桌面

浏览器访问时使用 kubeconfig 认证, 把刚才的 lucky-admin.conf 导入到 web 界面, 那么就可以登陆了

2. 创建一个能管理所有名称空间的 kubeconfig 文件, 步骤同上, 略

## 10.5 限制用户操作 k8s 资源

ssl 认证

生成一个证书

(1) 生成一个私钥

```
cd /etc/kubernetes/pki/
```

```
(umask 077; openssl genrsa -out lucky.key 2048)
```

(2) 生成一个证书请求

```
openssl req -new -key lucky.key -out lucky.csr -subj "/CN=lucky"
```

(3) 生成一个证书

```
openssl x509 -req -in lucky.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out lucky.crt -
days 3650
```

在 kubeconfig 下新增加一个 lucky 这个用户

(1) 把 lucky 这个用户添加到 kubernetes 集群中, 可以用来认证 apiserver 的连接

```
kubecttl config set-credentials lucky --client-certificate=./lucky.crt --client-  
key=./lucky.key --embed-certs=true
```

(2) 在 kubeconfig 下新增加一个 lucky 这个账号

```
kubecttl config set-context lucky@kubernetes --cluster=kubernetes --user=lucky
```

(3) 切换账号到 lucky, 默认没有任何权限

```
kubecttl config use-context lucky@kubernetes
```

```
kubecttl config use-context kubernetes-admin@kubernetes 这个是集群用户, 有任何权限
```

把 user 这个用户通过 rolebinding 绑定到 clusterrole 上, 授予权限, 权限只是在 lucky 这个名称空间有效

(1) 把 lucky 这个用户通过 rolebinding 绑定到 clusterrole 上

```
kubecttl create rolebinding lucky -n lucky --clusterrole=cluster-admin --user=lucky
```

(2) 切换到 lucky 这个用户

```
kubecttl config use-context lucky@kubernetes
```

(3) 测试是否有权限

```
kubecttl get pods -n lucky
```

有权限操作这个名称空间

```
kubecttl get pods
```

没有权限操作其他名称空间

添加一个 lucky 的普通用户

```
useradd lucky
```

```
cp -ar /root/.kube/ /home/lucky/
```

```
chown -R lucky.lucky /home/lucky/
```

```
su -lucky
```

```
kubecttl get pods -n lucky
```

#可以操作 lucky 名称空间

```
exit
```

#切换到 kubernetes-admin@kubernetes

```
kubecttl config use-context kubernetes-admin@kubernetes
```

## 总结:

实战 1 Configmap 热更新

实战 2、Secret 概述

10.1 k8s 安全管理: 认证、授权、准入控制概述

10.2 通过 token 令牌登录 dashboard

10.3 通过 kubeconfig 登录 dashboard

10.4 限制用户操作 k8s 资源