使用kubectl访问Kubernetes集群时的身份验证和授权

Authentication cluster

Tony Bai 2018-06-14 561 阅读

kubectl 是日常访问和管理 Kubernetes集群 最为常用的工具。

当我们使用 kubeadm 成功引导启动(init)一个 Kubernetes集群的控制平面 后，kubeadm会在init的输出结果中给予我们下面这样的“指示”：

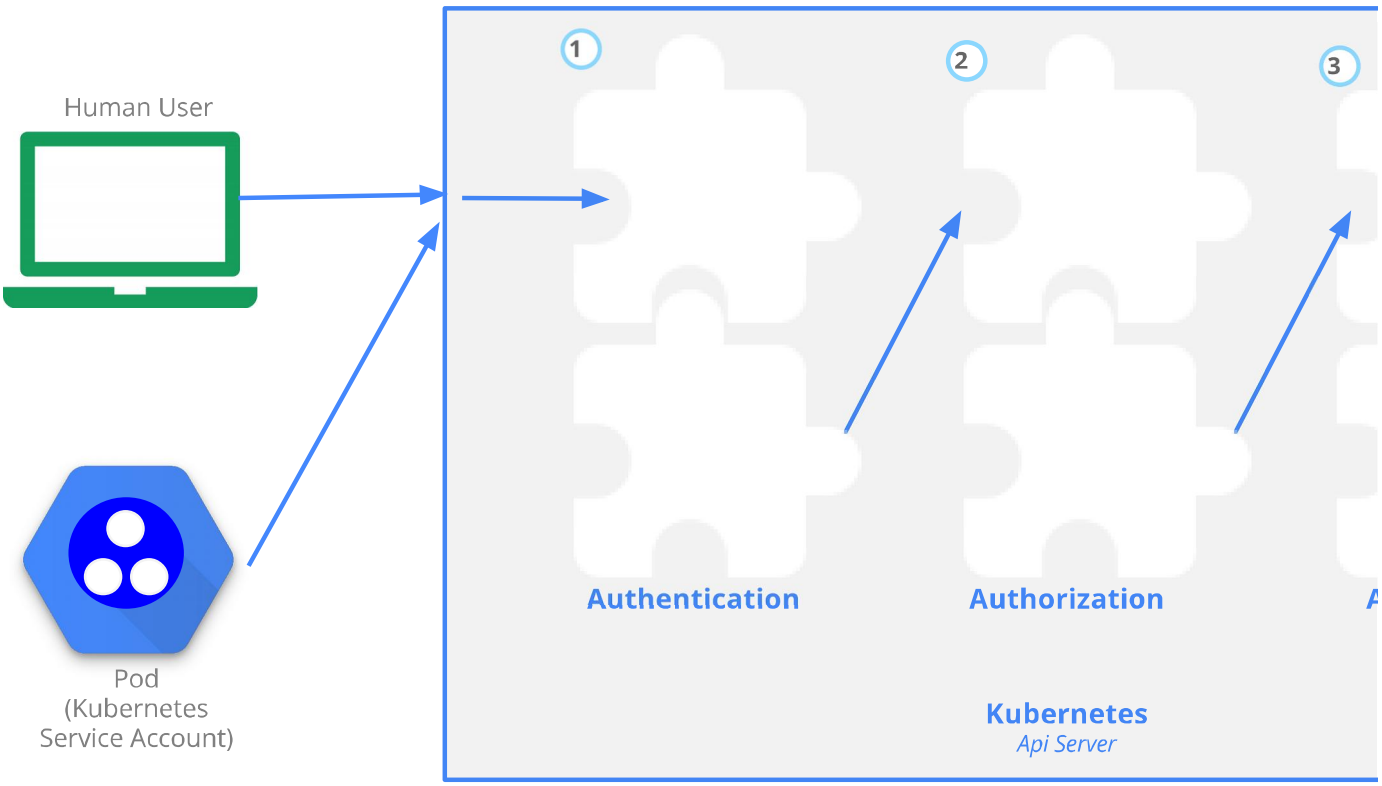
```
... ...
Your Kubernetes master has initialized successfully!
To start using your cluster, you need to run the following as a regular user:
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
... ...
```

kubeadm init 在结尾处输出的这些信息是在告知我们如何配置 **kubeconfig** 文件。按照上述命令配置后，master节点上的kubectl就可以直接使用$HOME/.kube/config的信息访问k8s cluster了。并且，通过这种配置方式，kubectl也拥有了整个集群的 **管理员(root)权限** 。

很多K8s初学者在这里都会有疑问：当kubectl使用这种kubeconfig方式访问集群时，Kubernetes的kube-apiserver是如何对来自kubectl的访问进行 身份验证(authentication)和授权(authorization) 的呢？为什么来自kubectl的请求拥有最高的管理员权限呢？在本文中，我们就来分析说明一下这个过程。

一. Kubernetes API的访问控制原理回顾

在 《Kubernetes的安全设置》 一文中我曾介绍过Kubernetes集群的访问权限控制由kube-apiserver负责，kube-apiserver的访问权限控制由身份验证(authentication)、授权(authorization)和准入控制（admission control）三步骤组成，这三步骤是按序进行的：



要想搞明白kubectl访问Kubernetes集群时的身份验证和授权，就是要弄清kube-apiserver在进行身份验证和授权两个环节都做了什么：

- Authentication：即身份验证，这个环节它面对的输入是整个http request，它负责对来自client的请求进行身份校验，支持的方法包括：client证书验证（https双向验证）、basic auth、普通token以及jwt token(用于serviceaccount)。APIServer启动时，可以指定一种Authentication方法，也可以指定多种方法。如果指定了多种方法，那么APIServer将会逐个使用这些方法对客户端请求进行验证，只要请求数据通过其中一种方法的验证，APIServer就会认为Authentication成功；在较新版本kubeadm引导启动的k8s集群的apiserver初始配置中，默认支持client证书验证和serviceaccount两种身份验证方式。在这个环节，apiserver会通过client证书或http header中的字段(比如serviceaccount的jwt token)来识别出请求的“用户身份”，包括”user”、”group”等，这些信息将在后面的authorization环节用到。

- Authorization：授权。这个环节面对的输入是http request context中的各种属性，包括：user、group、request path（比如：/api/v1、/healthz、/version等）、request verb(比如：get、list、create等)。APIServer会将这些属性值与事先配置好的访问策略(access policy）相比较。APIServer支持多种authorization mode，包括 Node 、 RBAC 、Webhook等。APIServer启动时，可以指定一种authorization mode，也可以指定多种authorization mode，如果是后者，只要Request通过了其中一种mode的授权，那么该环节的最终结果就是授权成功。在较新版本kubeadm引导启动的k8s集群的apiserver初始配置中，authorization-mode的默认配置是”Node,RBAC”。Node授权器主要用于各个node上的kubelet访问apiserver时使用的，其他一般均由RBAC授权器来授权。

RBAC，Role-Based Access Control即Role-Based Access Control，它使用”rbac.authorization.k8s.io”实现授权决策，允许管理员通过Kubernetes API动态配置策略。在RBAC API中，一个角色(Role)包含了一组权限规则。Role有两种：Role和ClusterRole。一个Role对象只能用于授予对某一单一命名空间（namespace）中资源的访问权限。ClusterRole对象可以授予与Role对象相同的权限，但由于它们属于集群范围对象， 也可以使用它们授予对以下几种资源的访问权限：

- 集群范围资源（例如节点，即node）
- 非资源类型endpoint（例如”/healthz”）
- 跨所有命名空间的命名空间范围资源（例如所有命名空间下的pod资源）

rolebinding，角色绑定则是定义了将一个角色的各种权限授予一个或者一组用户。 角色绑定包含了一组相关主体（即subject, 包括用户——User、用户组——Group、或者服务账户——Service Account）以及对被授予角色的引用。 在命名空间中可以通过RoleBinding对象进行用户

授权，而集群范围的用户授权则可以通过ClusterRoleBinding对象完成。

好了，有了上面这些知识基础，要搞清楚kubectl访问集群的身份验证和授权过程，我们只需要逐一解决下面的一些问题即可：

1、authencation中识别出了哪些http request context中的信息？
2、authorization中RBAC authorizer找到的对应的rolebinding或clusterrolebinding是什么？
3、对应的role或clusterrole的权限规则？

二. 在身份验证(authentication)识别出Group

我们先从kubectl使用的kubeconfig入手。kubectl使用的kubeconfig文件实质上就是kubeadm init过程中生成的/etc/kubernetes/admin.conf，我们查看一下该kubeconfig文件的内容：

```
环境k8s 1.10.3:
# kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://172.16.66.101:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

关于kubeconfig文件的解释，可以在 这里 自行脑补。在这些输出信息中，我们着重提取到两个信息：

```
user name: kubernetes-admin
client-certificate-date: XXXX
```

前面提到过apiserver的authentication支持通过tls client certificate、basic auth、token等方式对客户端发起的请求进行身份校验，从kubeconfig信息来看，kubectl显然在请求中使用了tls client certificate的方式，即客户端的证书。另外我们知道Kubernetes是没有 user 这种资源的，通过k8s API也无法创建user。那么kubectl的身份信息就应该“隐藏”在client-certificate的数据中，我们来查看一下。

首先我们将 /etc/kubernetes/admin.conf中client-certificate-data的数据内容保存到一个临时文件admin-client-certificate.txt中：

```
// admin-client-certificate.txt
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM4akNDQWRxZ0F3SUJBZ0lJZjJkVlJqbThFTFF3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhNS2EzVmlaWEp1Wl
```

然后针对该文件数据做base64解码，得到client certificate文件：

```
cat admin-client-certificate.txt | base64 -d > admin-client.crt
# cat admin-client.crt
-----BEGIN CERTIFICATE-----
MIIC8jCCAdqgAwIBAgIIf2dVRjm8ELQwDQYJKoZIhvcNAQELBQAwFTETMBEGA1UE
AxMKa3ViZXJuZXRlczAeFw0xODA1MTQwODE3MTNaFw0xOTA1MTQwODE3MTdaMDQx
FzAVBgNVBAoTDnN5c3RlbTptYXN0ZXJzMRkwFwYDVQQDExBrdWJlcm5ldGVzLWFk
bWluMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxBn3jdw80b1Gfb6s
w2NrqpLotMT4nyAf2HhqMrXjnO+wnaK1AITOw/22mDj0rwIuJwdQIj5/BaF63pPE
pU0vhIPVK4n6JI4dmMzo/lR3jZpGeZW1zdXaCovw9c7clbiHo/mFG4xqytVLfX4
/S8mFp2A9QcieJGIo5S0BR3FZlU1PM7DRbLDVVq1PdyNY2GfsbGrHlGgXvWAKCd/
H79gAqVoTXjSIWCVYuYcoLvdvVXQSIiYlpXFP1jBQLvcU7vrqtb12RmrxnpkW4pl
dGEOX2sLmfYZ5TiFpkRwz2GxsmWyRbt6OuISJFI6RZ0r+Rn4yMDKPrYlEngDVc5K
PZ5zmwIDAQABoycwJTAOBgNVHQ8BAf8EBAMCBaAwEwYDVR0lBAwwCgYIKwYBBQUH
AwIwDQYJKoZIhvcNAQELBQADggEBAEZNTvTz2OgzCUdvMFbrhPsp+mD2vPjMRCxi
BkA10vICOSfdymMn8aw0IbKYz2gQbXqUfqzQmQfa3if+QYBkB+77zfsv9am4EP/
e6Tg52tqV2P7s2eF7tNAe20GyV6yFlQ1QUW5/M4M+JMlV+BUbl9yEyQlENucKf+u
TPyKKTUtzvUYr5E3EJkt84EQINvw2nR2jNveZ1XWOliUrKfjHHtfvO/n56USuI4w
u2LTlICRcj4g+ZWlIjeMFkGyPbJyJAQ65P2sGrZm1klGGH3mzwO5CP1yZWvoUjjP
jzSjMCIaK/fR8eRAJ6q1tT6bG26L+njkKCQDWKpjAWOapuROcbk=
-----END CERTIFICATE-----
```

查看证书内容：

```
# openssl x509 -in ./admin-client.crt -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 9180400125522743476 (0x7f67554639bc10b4)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=kubernetes
        Validity
            Not Before: May 14 08:17:13 2018 GMT
            Not After : May 14 08:17:17 2019 GMT
        Subject: O=system:masters, CN=kubernetes-admin
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
    ... ...
```

从证书输出的信息中，我们看到了下面这行：

```
Subject: O=system:masters, CN=kubernetes-admin
```

k8s apiserver对kubectl的请求进行client certificate验证(通过ca证书–client-ca-file=/etc/kubernetes/pki/ca.crt对其进行校验)，验证通过后kube-apiserver会得到： **group = system:masters** 的http上下文信息，并传给后续的authorizers。

三. 在授权(authorization)时根据Group确定所绑定的角色(Role)

kubeadm在init初始引导集群启动过程中，创建了许多default的role、clusterrole、rolebinding和clusterrolebinding，在k8s有关 RBAC的官方文档 中，我们看到下面一些default clusterrole列表:

Default ClusterRole	Default ClusterRoleBinding	Description
cluster-admin	**system:masters** group	Allows super-user access to perform any action on any re used in a **ClusterRoleBinding**, it gives full control over eve the cluster and in all namespaces. When used in a **RoleBi** full control over every resource in the rolebinding's names the namespace itself.
admin	None	Allows admin access, intended to be granted within a nar **RoleBinding**. If used in a **RoleBinding**, allows read/write a resources in a namespace, including the ability to create rolebindings within the namespace. It does not allow writ resource quota or to the namespace itself.
edit	None	Allows read/write access to most objects in a namespac allow viewing or modifying roles or rolebindings.
view	None	Allows read-only access to see most objects in a namesp allow viewing roles or rolebindings. It does not allow view since those are escalating.

其中第一个 **cluster-admin** 这个cluster role binding绑定了 **system:masters** group，这和authentication环节传递过来的身份信息不谋而合。沿着 **system:masters** group对应的cluster-admin clusterrolebinding“追查”下去，真相就会浮出水面。

我们查看一下这一binding：

```
# kubectl get clusterrolebinding/cluster-admin -n kube-system -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  creationTimestamp: 2018-06-07T06:14:55Z
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: cluster-admin
  resourceVersion: "103"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterrolebindings/cluster-admin
  uid: 18c89690-6a1a-11e8-a0e8-00163e0cd764
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
```

```
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:masters
```

我们看到在kube-system名字空间中，一个名为cluster-admin的clusterrolebinding将cluster-admin cluster role与system:masters Group绑定到了一起，赋予了所有归属于system:masters Group中用户cluster-admin角色所拥有的权限。

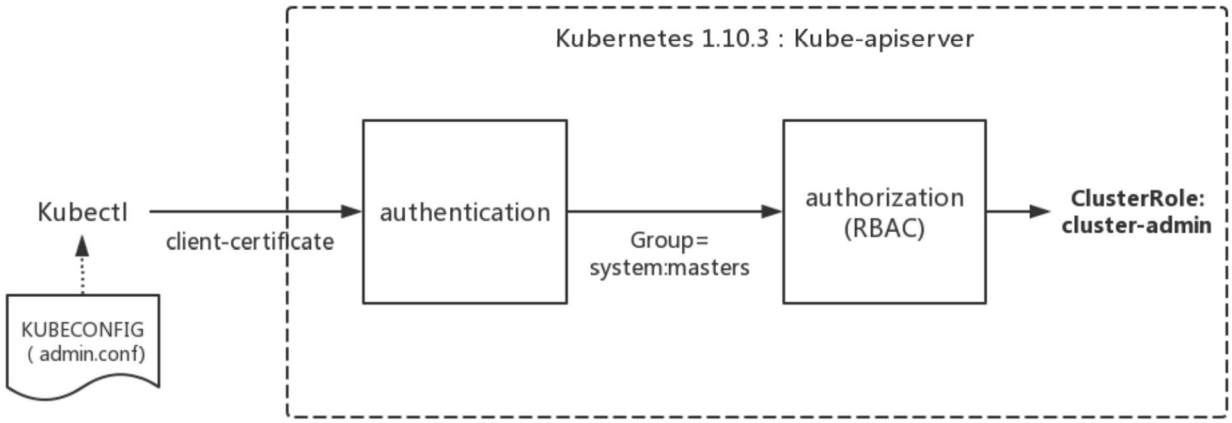
我们再来查看一下cluster-admin这个role的具体权限信息：

```
# kubectl get clusterrole/cluster-admin -n kube-system -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  creationTimestamp: 2018-06-07T06:14:55Z
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: cluster-admin
  resourceVersion: "52"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterroles/cluster-admin
  uid: 18abe535-6a1a-11e8-a0e8-00163e0cd764
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

从rules列表中来看，cluster-admin这个角色对所有resources、verbs、apiGroups均有无限制的操作权限，即整个集群的root权限。于是kubectl的请求就可以操控和管理整个集群了。

四. 小结

至此，我们应该明确了为什么采用了admin.conf kubeconfig的kubectrl拥有root权限了。下面是一幅示意图，简要总结了对kubectl访问请求的身份验证和授权过程：



大家可以结合这幅图，重温一下上面的文字描述，加深一下理解。

更多内容可以通过我在慕课网开设的实战课程 《Kubernetes实战 高可用集群搭建、配置、运维与应用》 学习 。

著名云主机服务厂商DigitalOcean发布最新的主机计划，入门级Droplet配置升级为：1 core CPU、1G内存、25G高速SSD，价格5$/月。有使用DigitalOcean需求的朋友，可以打开这个 链接地址： https://m.do.co/c/bff6eed92687 开启你的DO主机之路。

我的联系方式：

微博：https://weibo.com/bigwhite20xx
微信公众号：iamtonybai
博客：tonybai.com
github: https://github.com/bigwhite

微信赞赏：

商务合作方式：撰稿、出书、培训、在线课程、合伙创业、咨询、广告合作。

Authentication cluster

点赞

作者：Tony Bai

一个程序员的心路历程

原文地址：使用kubectl访问Kubernetes集群时的身份验证和授权, 感谢原作者分享。

←在Kubernetes 1.10.3上以Hard模式搭建EFK日志分析平台

→实践kubernetes ingress controller的四个例子

发表评论

发表评论

您可能感兴趣的博客

您可能感兴趣的代码