HUAWEI

CloudNative Lives

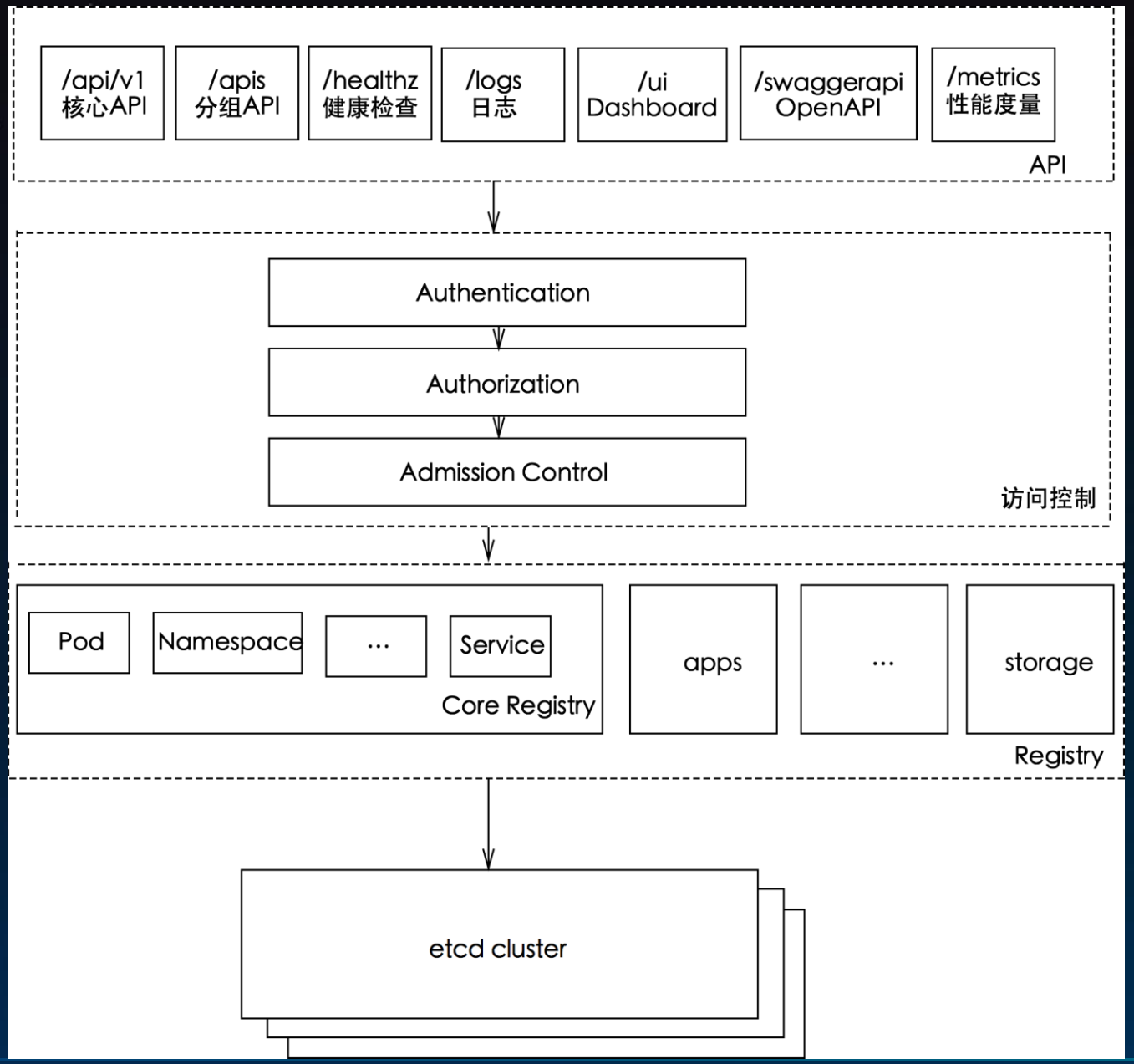
Kubernetes安全原理剖析&实践

华为云容器团队核心架构师 & CNCF社区主要贡献者倾心打造

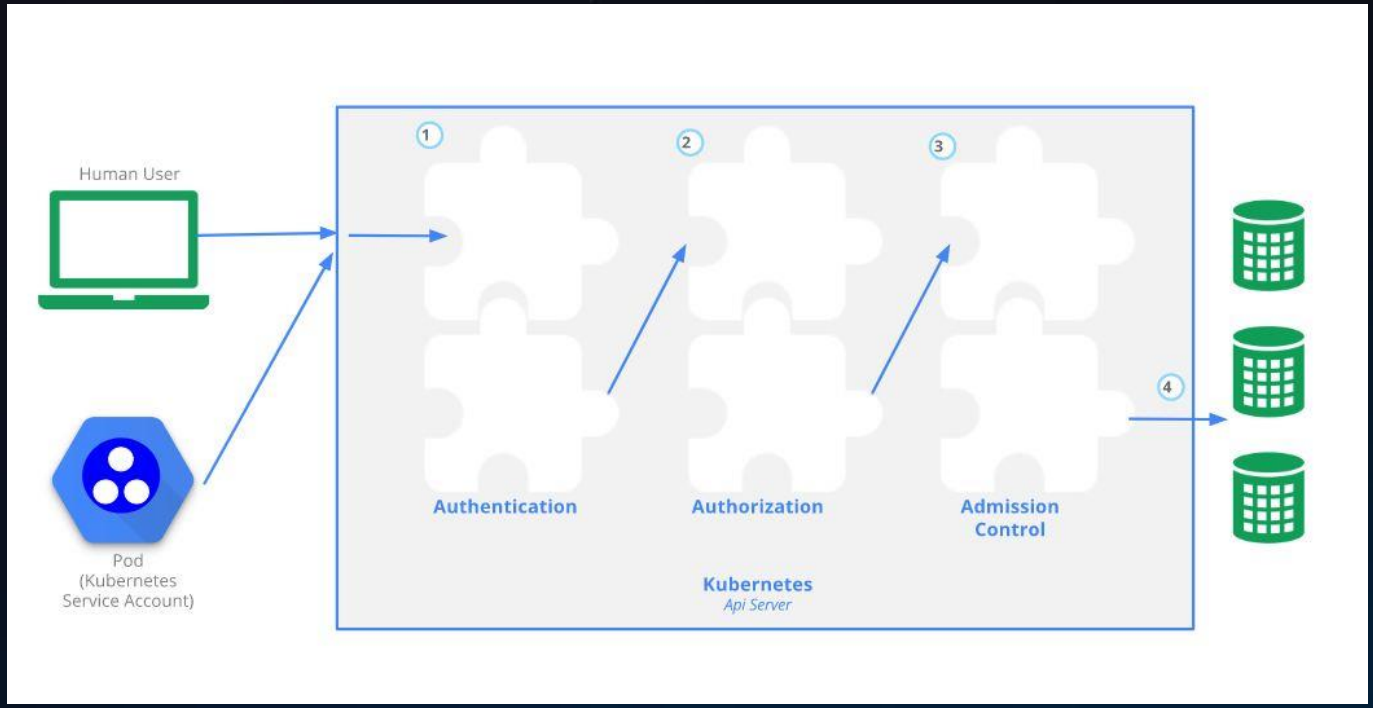
大 纲

- **认证与鉴权**
- 准入控制
- Service Account
- Secret

再看API Server



Kubernetes的安全框架



- 访问K8S集群的资源需要过三关：认证、鉴权、准入控制。
- 普通用户若要安全访问集群API Server，往往需要证书、token、用户名-密码；Pod访问，需要ServiceAccount
- K8S安全控制框架主要由下面3个阶段进行控制，每一个阶段都支持插件方式，通过API Server配置来启用插件
  - ① Authentication
  - ② Authorization
  - ③ Admission Control

认证：告别8080，迎接6443 ☹

三种**客户端**身份认证

认证方式	安全级别	API Server配置	Client配置	访问示例
证书+私钥	高	--client-ca-file：CA根证书 --tls-cert-file：API Server证书文件 --tls-private-key-file：API Server私钥文件	--certificate-authority：CA根证书 --client-certificate：客户端证书文件 --client-key：客户端私钥文件	kubectl --server=https:/192.168.61.100:6443 --certificate-authority=ca.pem --client-certificate=client.crt --client-key=client.key get nodes
Token	中	--token-auth-file：静态Token文件，csv格式： token,user,uid, “group1,group2,group3”	Kubectl --token 或者： Authorization: Bearer ${token}	kubectl –server=https:/192.168.61.100:6443 --token=792c62a1b5f2b07b --insecure-skip-tls-verify=true cluster-info 或者： curl -k --header “Authorization: Bearer 792c62a1b5f2b07b” https:/192.168.61.100:6443/api
用户名+密码	低	--basic_auth_file：basic认证文件，不建议在生产环境使用 格式： password,user,uid,"group1,group2,group3“	Kubectl –username –password 或者： Authorization:Basic BASE64ENCODED(USER:PASSWORD)	kubectl --server=https:/192.168.61.100:6443 --username=admin --password=1234 --insecure-skip-tls-verify=true cluster-info 或者： curl -k --header "Authorization:Basic YWRtaW46MTIzNA==" https:/192.168.61.100:6443/api

授权

- 用户通过认证后，对指定的资源是否有权限访问，还需要经过**授权**环节。授权主要是用于对集群资源的访问控制，通过检查请求包含的**相关属性值**，与相对应的访问策略相比较，API请求必须满足某些策略才能被处理
- Kubernetes授权仅处理以下的请求属性：
  - ① user, group, extra
  - ② API、请求方法（如get、post、update、patch和delete）和请求路径（如/api）
  - ③ 请求资源和子资源
  - ④ Namespace
  - ⑤ API Group
- API Server支持多种授权策略，通过启动参数“--authorization_mode”设置，可以**同时**启用多种策略

授权策略	功能描述
AlwaysAllow	接受所有请求，如果集群不需要授权流程，采用该策略
AlwaysDeny	拒绝所有请求，一般用于测试
ABAC	基于属性的访问控制，使用用户配置的授权规则去匹配用户的请求
RBAC	基于角色的访问控制，RBAC 的授权策略可以利用 kubectl 或者 Kubernetes API 直接进行配置。RBAC 可以授权给用户，让用户有权进行授权管理，这样就可以无需接触节点，直接进行授权管理
Webhook	WebHook 是一种 HTTP 回调，API Server把鉴权请求发给WebHook服务器，由服务器对请求进行鉴权
Node	对Node授权，配合NodeRestriction准入控制来限制kubelet仅可访问node、endpoint、pod、service以及secret、configmap、PV和PVC等相关的资源

授权策略 - ABAC

- ABAC（Attribute-based access control），基于**属性**的访问控制，通过使用将属性组合在一起的策略向用户授予访问权限
- 给API Server指定策略文件，--authorization-policy-file=FILENAME，文件内容是一行一个**Policy对象**的JSON串

```
{
"apiVersion":"abac.authorization.kubernetes.io/v1beta1",
 "kind":"Policy",
 "spec":{
   "user":"bob",
   “group”:”musician”,
   "namespace":"projectDJ",
   "resource":"*",
   "apiGroup":"*",
   "readonly":true
  }
}
```

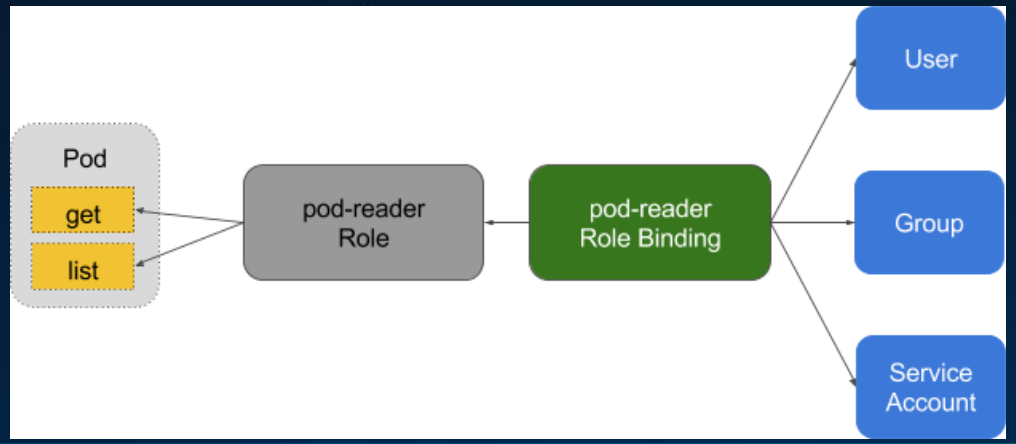
修改策略文件将需要API Server重启

ABAC – 示例

例子	功能描述
Alice可以对所有资源做任何操作	{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "alice", "namespace": "*", "resource": "*", "apiGroup": "*"}}
Kubelet可以读取任何Pod	{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet", "namespace": "*", "resource": "pods", "readonly": true}}
Kubelet可以读写事件	{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet", "namespace": "*", "resource": "events"}}
Bob可以在命名空间projectCaribou中读取Pod	{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "bob", "namespace": "projectCaribou", "resource": "pods", "readonly": true}}
任何人都可以对所有非资源路径进行只读请求	{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"group": "system:authenticated", "readonly": true, "nonResourcePath": "*"}} {"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"group": "system:unauthenticated", "readonly": true, "nonResourcePath": "*"}}

授权策略 - RBAC

- RBAC（Role-Based Access Control），允许通过Kubernetes API 动态配置策略。
- RBAC被映射成四种K8S顶级资源对象：
  - 角色(Role)：Role、ClusterRole
    * 角色表示一组权限的规则，累积规则
    * Role适用带namespace的资源，ClusterRole适用集群资源或非资源API
  - 建立用户与角色的映射/绑定关系：RoleBinding、ClusterRoleBinding
    * RoleBinding和ClusterRoleBinding的区别在于是否是namespace的资源
    * 角色绑定包含了一组相关主体（即subject, 包括用户、用户组、或者Service Account）以及对被授予角色的引用



角色

kind: Role

```
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # 空字符串""表明使用core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

kind: ClusterRole

```
apiVersion:
rbac.authorization.k8s.io/v1beta1
metadata:
  # 无namespace
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

角色绑定

```
# 允许用户"jane"从"default"命名空间读取Pod
kind: RoleBinding
apiVersion:
rbac.authorization.k8s.io/v1beta1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User # Service Account也可以
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

```
# 允许在用户组"manager"中的任何用户都可以读取集群中任何命名空间中的secret
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

Admission Control（准入控制）

- Admission Control实际上是一个准入控制器(Admission Controller)插件列表(又叫“准入控制链”)，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，检查不通过，则API Server拒绝请求
- **会自动修改Pod的配置**
- 官方支持20+个准入控制插件，而且支持自定义扩展
- 1.4以上版本官方推荐使用的插件

  --admission-control=NamespaceLifecycle,LimitRanger,**ServiceAccount**,DefaultStorageClass,ResourceQuota

Secret

Secret 对象类型用来保存敏感信息，例如密码、OAuth 令牌和 ssh key。将这些信息放在 secret 中比放在 Pod 的定义或者 docker 镜像中可以更好地控制它的用途，并降低意外暴露的风险。
Pod 一般有3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
- 环境变量
- 当 Kubelet 为 pod 拉取镜像时使用（imagePullSecret）

类型	功能描述
Opaque	用来存储密码、密钥等，使用base64编码格式
kubernetes.io/dockerconfigjson	也称imagePullSecrets，用来存储私有docker registry的认证信息
kubernetes.io/service-account-token	用于被serviceaccount引用。serviceaccout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

加密存储？

Huawei云CCE支持将Secret数据加密存储到etcd

Secret - Opaque类型定义

Opaque类型的数据是一个map类型，要求value是base64编码格式

以数据库用户名(admin => YWRtaW4=)、密码(1f2d1e2e67df => MWYyZDFlMmU2N2Rm)为例：

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: MWYyZDFlMmU2N2Rm
  username: YWRtaW4=
```

```
# base64 解码
$ echo "MWYyZDFlMmU2N2Rm" | base64 --decode
1f2d1e2e67df
```

Secret挂载到Volume

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    secret:
      secretName: mysecret
      defaultMode: 0400
```

```
# 从volume消费secret
$ ls /etc/foo/
username
password
$ cat /etc/foo/username
admin
$ cat /etc/foo/password
1f2d1e2e67df
```

挂载到volume的secret被更新时，被映射的key也将被更新。
Kubelet在周期性同步时检查被挂载的secret是不是最新的。
- 同步时延：kubelet sync period + ttl。

Secret作为环境变量

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: mycontainer
    image: redis
    env:
      - name: SECRET_USERNAME
        valueFrom:
          secretKeyRef:
            name: mysecret
            key: username
      - name: SECRET_PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysecret
            key: password
  restartPolicy: Never
```

```
# 消费环境变量里的Secret值：
$ echo $SECRET_USERNAME
admin
$ echo $SECRET_PASSWORD
1f2d1e2e67df
```

Secret - dockerconfigjson类型

kubernetes.io/dockerconfigjson类型secret是将包含Docker Registry凭证传递给 Kubelet 的一种方式，可以用来为Pod拉取私有镜像

$ kubectl create secret docker-registry myregistrykey --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL
secret/myregistrykey created.

OR: 从docker配置文件导入
$ kubectl create secret docker-registry myregistrykey –from-file=“~/.dockercfg”

$ kubectl get secret myregistrykey -o yaml
kind: Secret
type: kubernetes.io/dockerconfigjson
data:
.dockerconfigjson:
eyJhdXRocyI6eyJET0NLRVJfUkVHSVNUUllfU0VSVkVSIjp7InVzZXJuYW1lIjoiRE9DS0VSX1VTRVIiLCJwYXNzd29yZCI6IkRPQ0tFUl9QQVNTV09SRCIsImVtYWlsIjoiRE9DS0VSX0VNQUlMIiwiYXV0aCI6IlJFOURTMFZTWDFWVFJWSTZSRTlEUzBWU1gxQkJVMU5YVDFKRSJ9fX0=
metadata:
name: myregistrykey

dockerconfigjson类型Secret使用

```
apiVersion: v1
kind: Pod
metadata:
  name: private-reg
spec:
  containers:
  - name: private-reg-container
    image: <your-private-image>
  imagePullSecrets:
  - name: regcred
```

HUAWEI

service-account-token还没介绍？
不急，先学下Service Account

Service Account

- Service Account用于Pod中的进程访问API Server
  - 相对于客户端使用的user account（全局权限），为Pod内的进程提供身份标识
- 为什么需要Service Account？
  - 客户端授权方式是“全授权”，可以任意操作集群！需要更轻量和精准的方式。
- default Service Account
  - 当namespace创建时，会自动创建一个名为default的Service Account
  - 当default的Service Account创建时，会自动在同namespace下创建一个**default-token**-XXX，并关联到default的Service Account上
  - 创建Pod时，如果没有指定Service Account，K8S的Service Account Admission Controller会自动为该Pod指定default Service Account
- Pod关联Service Account
  - K8S会给Pod创建一个特殊的Volume，该Volume中包含指定Service Account Secret的token，namespace，证书文件，并将Volume挂载到Pod中所有容器的指定目录下(/var/run/secrets/kubernetes.io/serviceaccount)
- 认证环节
  - **用户名** system:serviceaccount:(**NAMESPACE**):(**SERVICEACCOUNT**)
  - **凭证** service account token

ServiceAccount使用

- **让Pod访问API Server**
  - 容器应用读取/var/run/secrets/kubernetes.io/serviceaccount/token文件，使用token认证方式访问API Server
  - client-go做了封装

- **Kubectl使用ServiceAccount token访问API Server**

① 查看指定namespace(如default)下的ServiceAccount，获取Secret
② 查看Secret，获取token
③ 设置kubeconfig中设置token
④ 使用kubectl访问

再看service-account-token类型Secret

创建Service Accout， K8S默认会创建对应的Secret


```
{
    "apiVersion": "v1",
    "data": {
        "ca.crt": "Q2VydGlmaWNhdGU6CiAgICBEYXRhOgogICAgICAgIFZlcnNpb246IDMgKDB4MikKICAgICAgICBTZXJpYWwgT
CAgICAgICAgICAgICAgYTY6ZDY6MWU6ZjU6MDk6NGY6OGE6ZDU6YjI6N2I6NWU6YzU6NTc6MDU6ZTA6CiAgICAgICAgICAgICAgI
W50OiA2NTUzNyAoMHgxMDAwMSkKICAgICAgICBYNTA5djMgZXh0ZW5zaW9uczoKICAgICAgICAgICAgWDUwOXYzIEJhc2ljIENvbnN0c
FNpZ24sIENSTCBTaWduCiAgICAgICAgICAgIFg1MDl2MyBTdWJqZWN0IEtleSBJZGVudGlmaWVyOiAKICAgICAgICAgICAgICAgIDREO
y9uc2daeK83TWlHenJmam9adOdBOONhRnlUZFZQdD2oMHJuRHZCUm92Skx3STYzOTdEWjdiTlJ6ClBGMDlvRmtPcVJOR1NZYW9lcIEz5
0xWRU5MTWRJSQpsSFNzN2l4VE43VjI2QzdFSUhFRDFyVk5QelFJQ2JrVFlGSFlPY05GYktkREZYbzliNnJRSVgzRWY5SGZyaVBzCk5TQ
2R2azhwMDZwWXgra2tRTW5lKy96UHQweTZPegpCa3ZTaSs2YUdXSE5YT2doUlRodTFaSTRBNFJlNjZNbVNmN0JoY2tUT1pxejF1NzF4N
2hJOHg0bms2Z09uUHp6SG9mUTBDZXZETkRjK21iUlNCT3FBPT0KLS0tLS1lFTkQgQ0VSVElGSUNBVEUtLS0tLQo=",
        "namespace": "a3ViZS1zeXN0ZW0=",
        "token": "ZXlKaGJHY2lPaUpTVXpJMU5pSXNJblI1Y0NJNklrcFhWQ0o5LmV5SnBjM0lpT2lKcmRXSmxjbTVsZEdWekwzTm
Z5Ym1WMFpYTXVhVzh2YzJWeWRtbGpaV0ZqWTI5MWJuUXZjMlZqY21WMExtNWhiV1VpT2lKcmRXSmxjbTVsZEdWekxXUmhjMmhpYjJGeV
xkRlZ6TFdSaGMyaGliMkZ5WkNJc0ltdDFZbVZ5Ym1WMFpYTXVhVzh2YzJWeWRtbGpaV0ZqWTI5MWJuUXZjMlZ5ZGlsalpTMWhZMk52ZF
REcHJkV0psTFhONWMzUmxiVHByZFdKbGNtNWxkR1Z6TFdSaGMyaGliMkZ5WkNKOS5Ud3VFNzZvWWVJOUIyaTdORXdjOWk2akVRN3NORz
AtUm0xZ0RzS20tQkx2a2VQSWd4Q0pNajR4TG1KcGwtUzNmV3BUVHBadmZBZXVWVE5nLWJQbjNEbk9NcnRMeUIxWmlRM0NwTEJHVkVaeF
Z3aFZodXdmQ01tY2RJS1ZoU3NTampxUjlWNG5NTzR3bFBFU1E="
    },
    "kind": "Secret",
    "metadata": {
        "annotations": {
            "generator": "JWT",
            "kubernetes.io/service-account.name": "kubernetes-dashboard",
            "kubernetes.io/service-account.uid": "cb5282cb-f52f-11e7-9d18-fa163e51dc68"
        },
        "creationTimestamp": "2018-01-09T11:25:28Z",
        "enable": true,
        "name": "kubernetes-dashboard-token-wnbk2",
        "namespace": "kube-system",
        "resourceVersion": "5399069",
        "selfLink": "/api/v1/namespaces/kube-system/secrets/kubernetes-dashboard-token-wnbk2",
        "uid": "cb54f3dd-f52f-11e7-9d18-fa163e51dc68"
    },
    "type": "kubernetes.io/service-account-token"
}
```


Secret使用：

kubernetes.io/service-account-token类型的Secret对应Pod中的ca.crt（API Server的CA公钥证书）， namespace，token（用API Server私钥签发的bearer token）三个文件：

① /run/sercrets/kubernetes.io/serviceaccount/token

② /run/sercrets/kubernetes.io/serviceaccount/ca.crt

③ /run/sercrets/kubernetes.io/serviceaccount/namespace

扫码加群技术交流

扫码报名厦门meetup