安全机制

认证基础

基础知识

学习目标

这一节，我们从 场景基础、认证简介、小结 三个方面来学习。

场景基础

应用场景

对于任何一种应用场景，其权限的认证管理都是非常重要的，对于linux系统来说，selinux、防火墙、pam、sudo等等，其核心的目的都是为了保证我们的环境是一个安全的。
对于k8s的这种大型的任务编排系统来说，设计到的认证远远超出了一般平台对认证的技术要求，在k8s平台中，陆陆续续的产生了一些列对平台的权限认证、对业务的权限认证、对网络的安全认证等等一些了认证体系。由于篇幅限制，我们这里专门针对平台和应用的权限认证来进行详细的学习。

业务流程

用户访问k8s业务应用的流程：
方法一：无需api_server认证
    用户 -- ingress|service -- pod
方法二：基于api_server认证
    管理k8s平台上各种应用对象

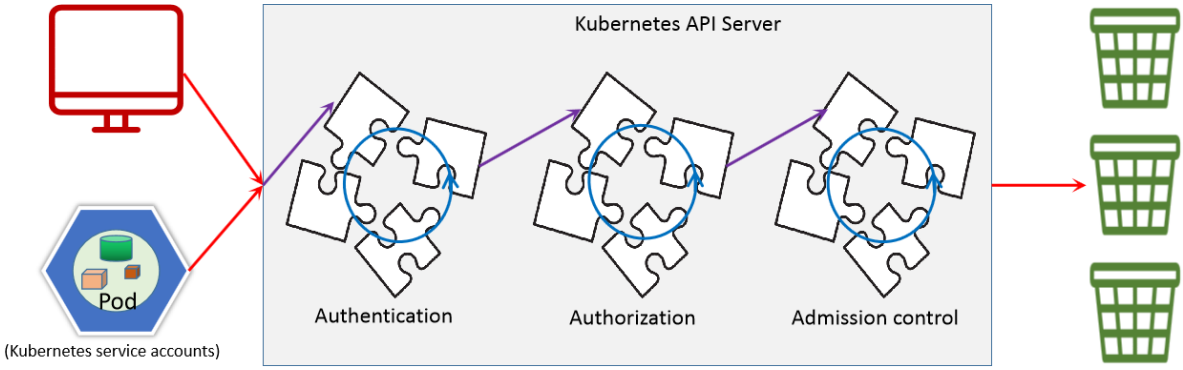
对于Kubernetes平台来说，几乎所有的操作基本上都是通过kube-apiserver这个组件进行的，该组件提供HTTP RESTful形式的API供集群内外客户端调用，对于kubernetes集群的部署样式主要有两种：http形式和https形式。我们采用kubeadm部署的形式默认对外是基于https的方式，而内部组件的通信时基于http方式，而k8s的认证授权机制仅仅存在于https形式的api访问中，也就是说，如果客户端使用HTTP连接到kube-apiserver，那么是不会进行认证授权的，这样既增加了安全性，也不至于太复杂。

认证简介

基本流程

对APIServer的访问一般都要经过的三个步骤，认证(Authn)+授权(Authz)、准入控制(Admission)，它也能在一定程度上提高安全性，不过更多是资源管理方面的作用。
注意：认证和授权功能都是以插件化的方式来实现的，这样可以最大化的用户自定义效果。

步骤	解析
认证(Authn)	对用户身份进行基本的认证，只允许被当前系统许可的人进入集群内部
授权(Authz)	不同的用户可以获取不同的资源操作权限，比如普通用户、超级用户、等
准入控制(Admission)	类似于审计，主要侧重于 操作动作的校验、语法规范的矫正等写操作场景。



左侧：
对于k8s来说，它主要面对两种用户：
普通人类用户(交互模式) - User Account
集群内部的pod用户(服务进程) - Service Account
中间：
每个部分都是以插件的方式来整合到 k8s 集群环境中：
- 认证和授权是按照 插件的顺序进行依次性的检测，并且遵循 "短路模型" 的认证方式
所谓的"短路"即，失败的时候，到此结束,后续的规则不会进行。
- 准入控制 由于仅用户写操作场景，所以它不遵循 "短路模型"，而是会全部检测
原因在于，它要记录为什么不执行，原因是什么，方便后续审计等动作。

小结

原理解析

学习目标

这一节，我们从 认证、授权、准入控制 三个方面来学习。

认证

认证用户

认证用户，在我们的认证范围中，有一个术语叫Subject，它表示我们在集群中基于某些规则和动作尝试操作的对象，在k8s集群中定义了两种类型的subject资源：

用户种类	解析
User Account	这是有外部独立服务进行管理的，对于用户的管理集群内部没有一个关联的资源对象，所以用户不能通过集群内部的 API 来进行管理。常见的管理方式就是 openssl等
Service Account	通过Kubernetes API 来管理的一些用户帐号，和 namespace 进行关联的，适用于集群内部运行的应用程序，需要通过 API 来完成权限认证，所以在集群内部进行权限操作，我们都需要使用到 ServiceAccount，这也是我们这节课的重点

用户组

在k8s集群中，为了更方便的对某一类用户进行细节化的管理，我们一般会通过用户组的方式来进行管理，常见的用户组有以下四类：

用户组	解析
system:unauthenticated	未能通过任何一个授权插件检验的账号的所有未通过认证测试的用户统一隶属的用户组；
system:authenticated	认证成功后的用户自动加入的一个专用组，用于快捷引用所有正常通过认证的用户账号；
system:serviceaccounts	所有名称空间中的所有ServiceAccount对象
system:serviceaccounts:	特定名称空间内所有的ServiceAccount对象

认证插件

kubernetes提供了多种认证方式，我们可以同时使用一种或多种认证方式，只要通过任何一个都被认作是认证通过。常见的认证方式如下：

认证方式	解析
客户端证书认证	客户端证书认证叫作TLS双向认证，也就是服务器客户端互相验证证书的正确性，在都正确的情况下协调通信加密方案。,最常见的方式X509数字证书，证书中的Subject中的 CommonName 或 Orgnization等信息进行校验 对于这种客户的账号，k8s的平台一般是无法管理的。为了使用这个方案，api-server需要用--client-ca-file、--tls-private-key-file、--tls-cert-file选项来开启。
令牌认证(Token)	在结点数量非常多的时候，大量手动配置TLS认证比较麻烦，我们可以通过在api-server开启 experimental-bootstrap-token-auth 特性，通过对客户端的和k8s平台预先定义的token信息进行匹配，认证通过后，自动为结点颁发证书，可以大大减轻我们的工作压力，而且应用场景非常广。
代理认证	一般借助于中间代理的方式来进行统用的认证方式，样式不固定

常见的令牌

引导令牌：

kubeadm创建集群环境的时候，自动创建好的令牌，用于其他节点加入到集群环境中

静态令牌：

存储于API Server进程可直接加载到的文件中保存的令牌，该文件内容会由API Server缓存于内存中

比如我们在使用kubelet的时候，需要依赖的token文件

静态密码：

存储于API Server进程可直接加载到的文件中保存的账户和密码令牌，该文件内容会由API Server缓存于内存中

比如我们在使用kubelet的时候，需要依赖的.kube/config文件

SA令牌：

SA 就是 ServiceAccount，集群内部账号之间操作相关资源的一些认证方式。

OIDC令牌：

OIDC 就是 OpenID Connect，一般应用于集成第三方认证的一种集中式认证方式，一般遵循OAuth 2协议。

尤其是第三方云服务商的认证。

Webhook令牌：

常应用于触发第三个动作时候的一些认证机制，主要侧重于http协议场景。

授权

简介

授权主要是在认证的基础上，用于对集群资源的访问控制权限设置，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。

授权方式

对于k8s集群来说，它会根据实际的业务应用场景，采用不同级别的授权方式，有时候也称为授权策略。这些授权策略很多，常见的方式有：

授权方式	解析
Node	主要针对节点的基本通信授权认证，比如kubelet的正常通信。
ABAC	(Attribute Based Access Control)：基于属性的访问控制，在apiserver本地的某一个文件里写入策略规则，如果满足其中一条，就算授权通过。现阶段如果想新增规则，那么必须重启apiserver，在生产环境中使用几率较小，但未来可能会使用API动态管理。 更多的关于动态属性控制可以参考 OPA(open policy agent)相关资料。
RBAC	(Role Based Access Control)：基于角色的访问控制，这是1.6+版本主推的授权策略，可以使用API自定义角色和集群角色，并将角色和特定的用户，用户组，Service Account关联起来，可以用来实现多租户隔离功能（基于namespace资源）
Webhook	使用第三方授权组件，以 HTTP Callback 的方式，利用外部授权接口对于已有访问控制组件实现权限控制的无缝衔接
AlwaysDeny	此标志阻止所有请求. 仅使用此标志进行测试
AlwaysAllow	此标志允许所有请求. 只有在您不需要API请求授权的情况下才能使用此标志

对于k8s集群来说，默认的认证插件有 Node 和 RBAC，其他的都是使用大量的证书来进行的。

```
root@master1:/etc/kubernetes/manifests# cat kube-apiserver.yaml
```

```
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=10.0.0.12
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
    - --secure-port=6443
    - --service-account-issuer=https://kubernetes.default.svc.cluster.local
    - --service-account-key-file=/etc/kubernetes/pki/sa.pub
    - --service-account-signing-key-file=/etc/kubernetes/pki/sa.key
    - --service-cluster-ip-range=10.96.0.0/12
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
```

准入控制

简介

> 准入控制(Admission Control)，实际上是一个准入控制器(Admission Controller)插件列表，发送到APIServer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。
> 它主要涉及到pod和容器对特定用户和特定权限之间的关联关系。

> 对于k8s来说，他支持的准入控制器有数十种，分别应用于不同的场景功能。一般情况下，k8s的不同版本的默认准入控制器相差不大。但是其他的功能场景的准入控制器相差还是比较大的，尤其是最近两三年版本中所支持的准入控制器。

> 到目前位置，k8s由于其灵活性和云原生的特性，它在安全层面做的不是太完善，每个版本更新的时候，都会在安全层面上面发现新的问题并进行解决。希望以后会好一点。

常见控制器

控制器	解析
LimitRanger	对一些没有做资源限制的pod，赋予一些默认的资源配置属性，主要针对的是pod个体。
ResourceQuota	对一些没有做资源限额的pod，赋予一些默认的资源限额，主要针对的是ns的范围限制。
PSP	PodSecurityPolicy，在集群层面来限制，pod内部使用的某些用户特权级别资源。

小结

认证解析

SA认证

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

SA简介

```
K8S自动为每个Pod注入一个 ServiceAccount 及 配套的令牌
在每个名称空间中，会自动存在(由ServiceAccount准入控制器负责)一个ServiceAccount，将被该空间下的每个Pod共享使用。
认证令牌保存于该空间下的一个Secret对象中，该对象中共有三个信息：
    namespace、ca.crt、token
```

认证示例

```
当我们在k8s环境中创建任意一个pod的时候，它都会帮我们自动创建一个属性信息：

样式1：
    Volumes:
      ...
      default-token-9vs4d:
        Type:        Secret (a volume populated by a Secret)
        SecretName:  default-token-9vs4d
        Optional:    false

    这里的SecretName就是一个认证信息，来源于我们创建k8s的时候，默认创建的一个secret
    ]# kubectl get secrets
    NAME                  TYPE                                  DATA   AGE
    default-token-9vs4d   kubernetes.io/service-account-token   3      3d
        这是k8s提供的一种认证模式，只不过默认的认证权限不是太大。

token是有有限期限的，如果过期的话，token就没有了，那么就会出现第二种方式。

样式2：
    Volumes:
      ...
      kube-api-access-x84dp:
        Type:                    Projected (a volume that contains injected data
from multiple sources)
        TokenExpirationSeconds:  3607
        ConfigMapName:           kube-root-ca.crt
        ConfigMapOptional:       <nil>
        DownwardAPI:             true
```

资源属性

```
apiVersion: v1                          # ServiceAccount所属的API群组及版本
```

```
kind: ServiceAccount                        # 资源类型标识
metadata:
  name <string>                             # 资源名称
  namespace <string>                        # ServiceAccount是名称空间级别的资源
automountServiceAccountToken <boolean>      # 是否让Pod自动挂载API令牌
secrets <[]Object>                          # 以该SA运行的Pod所要使用的Secret对象组成的列
表
  apiVersion <string>                       # 引用的Secret对象所属的API群组及版本，可省略
  kind <string>                             # 引用的资源的类型，这里是指Secret，可省略
  name <string>                             # 引用的Secret对象的名称，通常仅给出该字段即可
  namespace <string>                        # 引用的Secret对象所属的名称空间
  uid  <string>                             # 引用的Secret对象的标识符；
imagePullSecrets <[]Object>                 # 引用的用于下载Pod中容器镜像的Secret对象列表
  name <string>                             # docker-registry类型的Secret资源的名称
```

简单实践

- 信息查看

普通用户

```
查看普通用户空间的信息
]# kubectl get sa
NAME      SECRETS   AGE
default   1         3d20h
注意：
    每个命名空间都会有一个默认的default的sa账号名称

查看kube-system的sa信息
]# kubectl get sa -n kube-system | grep -v SECR| wc -l
37
可以看到：
    在kube-system的namespace中sa的条目多大31条
```

sa属性

```
]# kubectl get sa default -o yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: "2021-10-03T03:07:30Z"
  name: default
  namespace: default
  resourceVersion: "423"
  uid: c64f01cd-2a5b-4bbd-9bf6-8f78fd800f7f
secrets:
- name: default-token-pl9wp
结果显示：
    对于一个serviceaccount来说，其属性信息在metadata部分，在这里比较重要的是对于sa的信息交流还是需要通过secrets的认证信息进行通信，而这个认证信息它是自动帮我们附加的，我们无需关心。
```

secrets信息

```
root@master1:~/deployment# kubectl get secrets
NAME                  TYPE                                  DATA   AGE
```

```
default-token-pl9wp   kubernetes.io/service-account-token   3      2d11h
root@master1:~/deployment# kubectl get secrets default-token-pl9wp -o yaml
apiVersion: v1
data:
  ca.crt: LS0...tCg==
  namespace: ZGVmYXVsdA==
  token: ZXl...nBB
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: default
    kubernetes.io/service-account.uid: c64f01cd-2a5b-4bbd-9bf6-8f78fd800f7f
  creationTimestamp: "2021-10-03T03:07:30Z"
  name: default-token-pl9wp
  namespace: default
  resourceVersion: "421"
  uid: 1ef774f5-61ce-4442-a0e4-52133f498e20
type: kubernetes.io/service-account-token

结果显示：
    这个secrets是一个token类型的
```

- 创建sa账号

创建方法

```
命令格式：kubectl create serviceaccount NAME [--dry-run] [options]
作用：创建一个"服务账号"
参数详解
    --dry-run=false                     模拟创建模式
    --generator='serviceaccount/v1'     设定api版本信息
    -o, --output=''                     设定输出信息格式，常见的有：
                                            json|yaml|name|template|...
    --save-config=false                 保存配置信息
    --template=''                       设定配置模板文件

创建资源对象的一种简单方法
]# kubectl create serviceaccount mysa --dry-run -o yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: null
  name: mysa
```

创建SA

```
简单实践
root@master1:~/deployment# kubectl create serviceaccount admin
serviceaccount/admin created
root@master1:~/deployment# kubectl get sa
NAME      SECRETS   AGE
admin     1         4s                      生成了一个sa账号
default   1         2d11h
root@master1:~/deployment# kubectl describe sa admin
Name:                admin
Namespace:           default
Labels:              <none>
```

```
Annotations:         <none>
Image pull secrets:  <none>                  下载镜像时候的认证信息
Mountable secrets:   admin-token-4497k       挂载数据时候的认证信息
Tokens:              admin-token-4497k       原始的账户认证信息
Events:              <none>

sa账号配套的token是在secrets对象中
root@master1:~/deployment# kubectl get secrets
NAME                  TYPE                                  DATA   AGE
admin-token-4497k     kubernetes.io/service-account-token   3      79s
default-token-pl9wp   kubernetes.io/service-account-token   3      2d11h
注意：
    token是有准入控制器自动为用户配套创建的，无需手工干预
```

应用SA

```
在我们创建pod的时候，里面会有一条属性，专门来设置该资源属于哪个sa管理
kubectl explain pod.spec.serviceAccountName

创建pod使用自定义的sa
cat > 01-security-sa-admin.yaml << EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-sa-admin
spec:
  containers:
  - name: pod-sa-admin
    image: 10.0.0.19:80/mykubernetes/pod_test:v0.1
    imagePullPolicy: IfNotPresent
  serviceAccountName: admin
EOF
实例化资源对象
kubectl apply -f 01-security-sa-admin.yaml
检查效果
[root@master ~]# kubectl describe pod pod-sa-admin
Name:               pod-sa-admin
...
Containers:
  pod-sa-admin:
    ...
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-sxhhw
(ro)
Volumes:
  admin-token-w6pv9:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  admin-token-29w8g
    Optional:    false
...
```

结果显示：

由于我们的pod中是被admin的serviceaccount管理的，所以该资源对象自动附加了admin账号的认证信息。

这些信息挂载到容器的 /var/run/secrets/kubernetes.io/serviceaccount 目录下了

确认ca相关的信息

```
进入到容器中确认信息
root@master1:~/sa# kubectl exec -it pod-sa-admin -- /bin/sh
[root@pod-sa-admin /]# ls /var/run/secrets/kubernetes.io/serviceaccount/
ca.crt     namespace  token

查看相关信息
cat ca.crt
cat namespace
cat token
```

小结

UA基础

学习目标

这一节，我们从 证书信息、权限关系、小结 三个方面来学习。

证书信息

简介

我们知道，通过kubeadm在创建集群的时候，其中有一步就是：生成kubernetes控制组件的kubeconfig文件及相关的启动配置文件,通过各种conf文件，让不同的组件具备操作相关资源的权限。

```
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can tak
e up to 4m0s
[apiclient] All control plane components are healthy after 21.300208 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.22" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
```

kubeconfig 文件专门为 kubectl命令使用的，那么这个文件是如何被kubectl命令直接使用的呢？

配置信息

```
kubernetes 配置文件管理
[root@master ~]# kubectl config -h
Modify kubeconfig files using subcommands like "kubectl config set current-
context my-context"

The loading order follows these rules:

  1. If the --kubeconfig flag is set, then only that file is loaded. The flag
may only be set once and no merging takes place.
```

```
  2. If $KUBECONFIG environment variable is set, then it is used as a list of
paths (normal path delimitting rules for your system). These paths are merged.
When a value is modified, it is modified in the file that defines the stanza.
When a value is created, it is created in the first file that exists. If no
files in the chain exist, then it creates the last file in the list.
  3. Otherwise, ${HOME}/.kube/config is used and no merging takes place.
# 配置文件的基本信息
    1 使用 --kubeconfig 参数管理某个指定的配置文件内容
    2 该文件可以使用 $KUBECONFIG 变量来管理
    3 其他情况下，配置文件指的就是 ${HOME}/.kube/config
    优先级： 1 > 2 > 3
Available Commands:
  current-context   显示 current_context
  delete-cluster    删除 kubeconfig 文件中指定的集群
  delete-context    删除 kubeconfig 文件中指定的 context
  get-clusters      显示 kubeconfig 文件中定义的集群
  get-contexts      描述一个或多个 contexts
  rename-context    Renames a context from the kubeconfig file.
  set                   设置 kubeconfig 文件中的一个单个值
  set-cluster       设置 kubeconfig 文件中的一个集群条目
  set-context       设置 kubeconfig 文件中的一个 context 条目
  set-credentials   设置 kubeconfig 文件中的一个用户条目
  unset             取消设置 kubeconfig 文件中的一个单个值
  use-context       设置 kubeconfig 文件中的当前上下文
  view              显示合并的 kubeconfig 配置或一个指定的 kubeconfig 文件

Usage:
  kubectl config SUBCOMMAND [options]
结果显示：
    对于一个用户账号来说，至少包含了三部分：
        用户条目-credentials 设定具体的user account名称
        集群-cluster 设定该user account所工作的区域
        上下文环境-context 设定用户和集群的关系
```

配置内容

```
查看配置文件内容
[root@master ~]# kubectl config view
apiVersion: v1
clusters:                                       # 集群列表
- cluster:
    certificate-authority-data: DATA+OMITTED    # 证书的认证方式
    server: https://10.0.0.12:6443              # api_server 的地址
  name: kubernetes                              # 当前集群的名称
contexts:                                       # 上下文列表，一般指的是多集群间用户的切换
所需的环境属性
- context:
    cluster: kubernetes                         # 集群名称kubernetes
    user: kubernetes-admin                      # 使用kubernetes-admin用户来访问集
群kubernetes
  name: kubernetes-admin@kubernetes             # 该context的名称标准写法
current-context: kubernetes-admin@kubernetes    # 当前上下文的名称
kind: Config
preferences: {}
users:                                          # 用户列表
- name: kubernetes-admin                            # 用户名称
  user:                                             # 用户自己认证属性
```

```
    client-certificate-data: REDACTED                # 客户端证书
    client-key-data: REDACTED                        # 客户端私钥
```

结果显示：

一个config主要包含了三部分内容：users、clusters、contexts，每个部分都有两部分组成：name和user|cluster|context

对于cluster，对外的地址-server 和 基本的认证方式-certificate-authority-data

对于context，连接到的集群-cluster 和 连接集群的用户-user

对于user，连接集群的认证方式-client-certificate-data 和 私钥信息-client-key-data

current-context表明我们是处于哪一个环境中。

默认方式

```
root@master1:~# kubectl config --help
    ...
  3.  Otherwise, ${HOME}/.kube/config is used and no merging takes place.
```

结果显示：

默认使用的kubeconfig 就是 当前用户家目录下的 .kube/config 文件

而这个文件是我们在创建完毕集群后，自动复制过来的

```
root@master1:~# diff .kube/config /etc/kubernetes/admin.conf
root@master1:~#
```

命令方式：

```
root@master1:~# kubectl options | grep kubeconfig
      --cluster='': The name of the kubeconfig cluster to use
      --context='': The name of the kubeconfig context to use
      --kubeconfig='': Path to the kubeconfig file to use for CLI requests.
      --user='': The name of the kubeconfig user to use
```

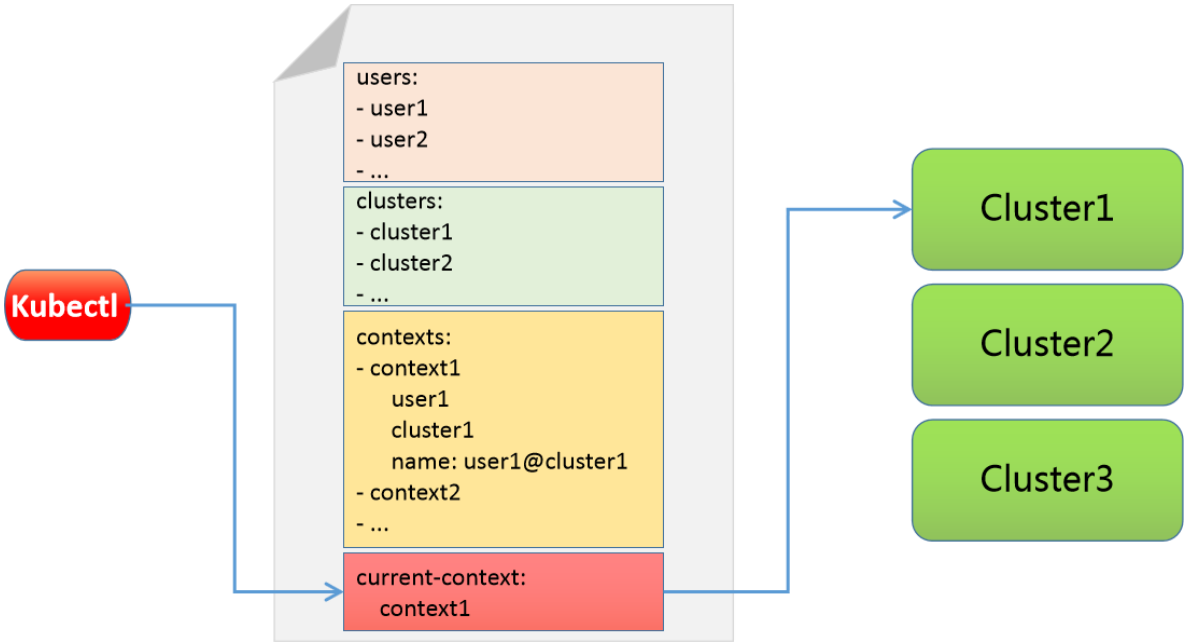
结果显示：

如果我们有自己的config文件的话，可以使用 --kubeconfig 选项来覆盖默认的配置文件

环境变量方式

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

```
root@master1:~# kubectl --kubeconfig=/etc/kubernetes/admin.conf get nodes
NAME      STATUS   ROLES                  AGE     VERSION
master1   Ready    control-plane,master   2d12h   v1.22.2
node1     Ready    <none>                 2d12h   v1.22.2
node2     Ready    <none>                 2d12h   v1.22.2
```

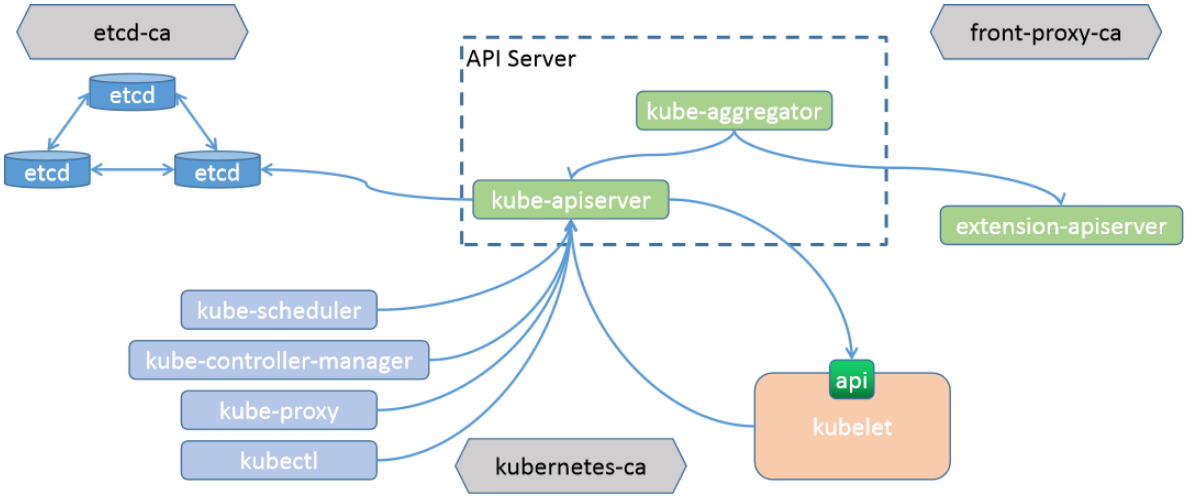


ca相关信息

```
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=10.0.0.12
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
    - --secure-port=6443
    - --service-account-issuer=https://kubernetes.default.svc.cluster.local
    - --service-account-key-file=/etc/kubernetes/pki/sa.pub
    - --service-account-signing-key-file=/etc/kubernetes/pki/sa.key
    - --service-cluster-ip-range=10.96.0.0/12
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
```

```
root@master1:~# ls /etc/kubernetes/pki/
apiserver.crt                apiserver-kubelet-client.key  front-proxy-ca.key
apiserver-etcd-client.crt    ca.crt                        front-proxy-
client.crt
apiserver-etcd-client.key    ca.key                        front-proxy-
client.key
apiserver.key                etcd                          sa.key
apiserver-kubelet-client.crt  front-proxy-ca.crt           sa.pub
root@master1:~# ls /etc/kubernetes/pki/etcd/
ca.crt  healthcheck-client.crt  peer.crt  server.crt
ca.key  healthcheck-client.key  peer.key  server.key
```

证书体系



对于一个自建的kubernetes集群来说，它依赖一套CA相关的证书

默认 CN	父级 CA	O (位于 Subject 中)	类型	主机 (SAN)
kube-etcd	etcd-ca		server, client	`localhost`, `127.0.0.1`
kube-etcd-peer	etcd-ca		server, client	`<hostname>`, `<Host_IP>`, `localhost`, `127.0.0.1`
kube-etcd-healthcheck-client	etcd-ca		client	
kube-apiserver-etcd-client	etcd-ca	system:masters	client	
kube-apiserver	kubernetes-ca		server	`<hostname>`, `<Host_IP>`, `<advertise_IP>`, `[1]`
kube-apiserver-kubelet-client	kubernetes-ca	system:masters	client	
front-proxy-client	kubernetes-front-proxy-ca		client	

资料来源：`https://kubernetes.io/zh/docs/setup/best-practices/certificates/`

权限关系

用户组

我们知道所有的资源操作，其实都是node结点上的kubelet和master结点上的apiserver中间的通信，而在kubernetes的认证目录中尤其专用的通信认证证书 apiserver-kubelet-client.crt，我们可以通过该文件来检查一下这两者之间是一个怎样的关系。

```
]# cd /etc/kubernetes/pki/
]# openssl x509 -in ./apiserver-kubelet-client.crt  -text -noout
Certificate:
    ...
        Issuer: CN = kubernetes
        Validity
            Not Before: Oct  3 03:06:43 2021 GMT
            Not After : Oct  3 03:06:43 2022 GMT
        Subject: O=system:masters, CN=kube-apiserver-kubelet-client
...
```

结果显示：

对于kubelet来说，他的用户名是kube-apiserver-kubelet-client，而且属于system:master的组，这两者的关系是我们在基于openssl或者csffl工具创建用户时候基于CN和O来设定的信息。

绑定关系

在我们的集群环境中有一个资源叫clusterrolebindings，该资源会基于用户的名称或者组信息将其绑定到一个权限列表中(即clusterrole)，在这个绑定关系中有一个cluster-admin，在这里面定义了group和clusterrole之间的关系。

```
]# kubectl describe clusterrolebindings cluster-admin
Name:         cluster-admin
...
Role:
  Kind:  ClusterRole
  Name:  cluster-admin
Subjects:
  Kind   Name            Namespace
  ----   ----            ---------
  Group  system:masters

]# kubectl get clusterrolebindings cluster-admin -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
...
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:masters
```

结果显示：

只要是"system:masters"组中的成员，都具有cluster-admin的所有权限

角色权限

我们可以通过clusterrole 资源来查看cluster-admin的所有权限信息

```
]# kubectl describe clusterrole cluster-admin
Name:         cluster-admin
...
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  *.*        []                 []              [*]
             [*]                []              [*]
```

结果显示：

对于cluster-admin来说，它拥有所有资源在所有空间的所有权限。

小结

UA实践

学习目标

这一节，我们从 实践思路、简单实践、小结 三个方面来学习。

实践思路

基本流程

根据我们对config和openssl的了解，完成一个整体的用户认证，其创建流程主要分七步：

1 创建私钥文件
2 基于私钥文件创建证书签名请求
3 基于私钥和签名请求生成证书文件
4 基于tls文件在k8s上创建用户
5 创建工作区域-cluster
6 将cluster和user关联起来-context
7 验证效果

技术关键点

这七步各有操作关键点，我们在操作之前必须提前认识清楚，否则的话，在执行过程中，会遇到各种意外。

1 创建私钥文件

对于用户名和用户组需要提前规划好，如果用户多权限集中的情况下，一定要规划好用户组信息

2 基于私钥文件创建证书签名请求

要基于我们自建的私钥来创建签证请求文件

3 基于私钥和签名请求生成证书文件

因为我们的生成的证书要应用在kubernetes环境中，所以必须由kubernetes的全局证书来认证

4 基于tls文件在k8s上创建用户

我们说过kubernetes平台上的普通用户需要基于秘钥认证通信

5 创建工作区域-cluster

所谓的工作区域是用户的工作场景，必须定制好，一个cluster可以被多个用户使用

6 将cluster和user关联起来-context

关联的作用就是，将用户和区域整合在一起，我们使用资源的时候便于调用

7 验证效果

因为我们主要做的是认证，而用户的操作涉及到资源权限，这部分是需要结合授权机制来进行的

默认情况下，基于创建好的文件来获取资源是被forbidden的

简单实践

1 创建私钥文件

```
给用户 wangss 创建一个私钥，命名成：wangss.key(无加密)
cd /etc/kubernetes/pki/
(umask 077; openssl genrsa -out wangss.key 2048)
命令解析：
    genrsa          该子命令用于生成RSA私钥，不会生成公钥，因为公钥提取自私钥
    -out filename   生成的私钥保存至filename文件，若未指定输出文件，则为标准输出
    -numbits        指定私钥的长度，默认1024，该项必须为命令行的最后一项参数

注意：
    为了避免对当前环境产生不必要的影响，我们这里使用子shell的方式创建私钥
    由于我们后面需要k8s的私钥对用户进行认证，所以我们在k8s的认证文件目录中创建
```

2 签名请求

```
用刚创建的私钥创建一个证书签名请求文件：wangss.csr
openssl req -new -key  wangss.key -out wangss.csr -subj "/CN=wangss/O=wangss"
参数说明：
    -new    生成证书请求文件
    -key    指定已有的秘钥文件生成签名请求，必须与-new配合使用
    -out    输出证书文件名称
    -subj   输入证书拥有者信息，这里指定 CN 以及 O 的值，/表示内容分隔
        CN以及O的值对于kubernetes很重要，因为kubernetes会从证书这两个值对应获取相关信息：
            "CN"：Common Name，用于从证书中提取该字段作为请求的用户名 (User Name)；
                浏览器使用该字段验证网站是否合法；
            "O"：Organization，用于分组认证
注意：
    证书名称必须设计好，在我们这里，用户是wangss、组是wangss。

检查效果：
]# ls wangss.*
wangss.csr  wangss.key
结果显示：
    *.key 是我们的私钥，*.csr是我们的签名请求文件
```

3 生成证书

```
刚才的私钥和认证并没有被我们的Kubernetes集群纳入到管理体系，我们需要基于kubeadm集群的CA相关证书来进行认证，CA相关文件位于/etc/kubernetes/pki/目录下面，我们会利用该目录下面的ca.crt和ca.key两个文件来批准上面的证书请求
]# openssl x509 -req -in wangss.csr -CA ./ca.crt  -CAkey ./ca.key  -CAcreateserial -out wangss.crt -days 365
Signature ok
subject=/CN=wangss/O=wangss
Getting CA Private Key
参数说明：
    -req                产生证书签发申请命令
    -in                 指定需要签名的请求文件
    -CA                 指定CA证书文件
    -CAkey          指定CA证书的秘钥文件
    -CAcreateserial 生成唯一的证书序列号
    -x509               表示输出一个X509格式的证书
    -days               指定证书过期时间为365天
```

```
    -out                输出证书文件
检查文件效果
]# ls wangss.*
wangss.crt  wangss.csr  wangss.key
结果显示：
    *.crt就是我们最终生成的签证证书
```

```
提取信息效果
]# openssl x509 -in wangss.crt -text -noout
Certificate:
   ...
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=kubernetes
        Validity
            Not Before: Oct  6 01:32:46 2021 GMT
            Not After : Oct  6 01:32:46 2022 GMT
        Subject: CN=wangss, O=wangss
结果显示：
    Issuer: 表示是哪个CA机构帮我们认证的
    我们关注的重点在于Subject内容中的请求用户所属的组信息
```

4 创建k8s用户

```
创建用户信息
kubectl config set-credentials wangss --client-certificate=./wangss.crt --
client-key=./wangss.key  --embed-certs=true --kubeconfig=/tmp/sswang.conf
参数详解：
    set-credentials      子命令的作用就是给kubeconfig认证文件创建一个用户条目
    --client-certificate=path/to/certfile    指定用户的签证证书文件
    --client-key=path/to/keyfile                  指定用户的私钥文件
    --embed-certs=true，在kubeconfig中为用户条目嵌入客户端证书/密钥，默认值是false，
    --kubeconfig=/path/to/other_config.file 表示将属性信息单独输出到一个文件，不指定的
话，表示存到默认的 ~/.kube/config文件中

检查效果
]# kubectl config view --kubeconfig=/tmp/sswang.conf
apiVersion: v1
clusters: []
contexts: []
current-context: ""
kind: Config
preferences: {}
users:
- name: wangss
  user:                                          --embed-certs=false后的证书效果
    client-certificate-data: REDACTED       client-certificate:
/etc/kubernetes/pki/wangss.crt
    client-key-data: REDACTED               client-key:
/etc/kubernetes/pki/wangss.key
结果显示：
    在指定配置文件的users部分增加了一个wangss的普通用户
```

5 创建集群

创建一个新的集群mycluster
```
kubectl config set-cluster mycluster  --server="https://10.0.0.12:6443" --
certificate-authority=/etc/kubernetes/pki/ca.crt --embed-certs=true --
kubeconfig=/tmp/sswang.conf
```
参数详解：
```
    --server=cluster_api_server
    --certificate-authority=path/to/certificate/authority
```
注意：
    这里使用到的证书，必须是kubernetes的ca证书。

检查效果
```
]# kubectl config view --kubeconfig=/tmp/sswang.conf
                           apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.0.0.12:6443
  name: mycluster
...
```
属性解析
    如果不用--embed-certs=true 效果如下
```
    certificate-authority: /etc/kubernetes/pki/ca.crt
```

6 关联用户和集群

配置上下文信息
```
kubectl  config set-context wangss@mycluster --cluster=mycluster --user=wangss
--kubeconfig=/tmp/sswang.conf
```
属性详解
```
    --cluster=cluster_nickname      关联的集群名称
    --user=user_nickname            关联的用户名称
    --namespace=namespace           可以设置该生效的命名空间
```

最终效果
```
]# kubectl config view --kubeconfig=/tmp/sswang.conf
        apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.0.0.12:6443
  name: mycluster
contexts:
- context:
    cluster: mycluster
    user: wangss
  name: wangss@mycluster
current-context: ""
kind: Config
preferences: {}
users:
- name: wangss
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

7 验证效果

根据刚才的信息显示，current-context的信息是空，那么我们切换一下用户
更改用户
```
kubectl config use-context wangss@mycluster --kubeconfig=/tmp/sswang.conf
```

检查效果
```
]# kubectl config view --kubeconfig=/tmp/sswang.conf
...
contexts:
...
current-context: wangss@mycluster
...
```

检查权限
```
]# kubectl get pod --kubeconfig=/tmp/sswang.conf
Error from server (Forbidden): pods is forbidden: User "wangss" cannot list
resource "pods" in API group "" in the namespace "default"
```
结果显示：
    虽然认证信息我们配置好了，但是由于权限相关的内容没有设置，所以我们看不了任何资源信息

简便方法
```
kubectl get pods --context=wangss@mycluster --kubeconfig=/tmp/sswang.conf
```

小结

UA操作

学习目标

这一节，我们从 简单实践、其他操作、小结 三个方面来学习。

简单实践

使用默认的kubectl配置文件

```
root@master1:/etc/kubernetes/pki# kubectl get sa
NAME      SECRETS   AGE
admin     1         10h
default   1         2d22h
```

注意：
    这里使用的配置文件是 ~/.kube/config 文件

使用--kubeconfig 参数

```
root@master1:~# kubectl get sa --kubeconfig=/etc/kubernetes/admin.conf
NAME      SECRETS   AGE
admin     1         10h
default   1         2d22h

root@master1:/etc/kubernetes/pki# kubectl get sa --kubeconfig=/tmp/sswang.conf
Error from server (Forbidden): serviceaccounts is forbidden: User "wangss"
cannot list resource "serviceaccounts" in API group "" in the namespace
"default"

结果显示：
    --kubeconfig 参数的优先级要高于 默认的 .kube/config 配置文件
```

使用环境变量

```
root@master1:~# export KUBECONFIG='/etc/kubernetes/admin.conf'
root@master1:~# kubectl get sa
NAME      SECRETS   AGE
admin     1         10h
default   1         2d22h

root@master1:~# export KUBECONFIG='/tmp/sswang.conf'
root@master1:~# kubectl get sa
Error from server (Forbidden): serviceaccounts is forbidden: User "wangss"
cannot list resource "serviceaccounts" in API group "" in the namespace
"default"

root@master1:~# kubectl get sa --kubeconfig=/etc/kubernetes/admin.conf
NAME      SECRETS   AGE
admin     1         10h
default   1         2d22h
结果显示：
    环境变量KUBECONFIG 的优先级要高于 ~/.kube/config
    --kubeconfig 的优先级要高于 环境变量KUBECONFIG
```

其他操作

为了能让所有动作可以正常执行，我们合并两个配置文件

```
环境变量方式
root@master1:~# export KUBECONFIG='/etc/kubernetes/admin.conf:/tmp/sswang.conf'
root@master1:~# kubectl get sa
NAME      SECRETS   AGE
admin     1         11h
default   1         2d23h
注意：
    多个文件可以通过冒号隔开

查看多个配置文件的合并统一查询效果
root@master1:~# kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
```

```
    server: https://10.0.0.12:6443
  name: kubernetes
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.0.0.12:6443
  name: mycluster
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
- context:
    cluster: mycluster
    user: wangss
  name: wangss@mycluster
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
- name: wangss
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

采用 merge 参数合并文件

```
kubectl config view --merge --flatten > newconfig.conf
```

属性解析：

    --flatten 用于显示加密后的内容

使用效果

```
root@master1:~# kubectl get sa --kubeconfig=/tmp/newconfig.conf
NAME      SECRETS   AGE
admin     1         11h
default   1         2d23h
```

结果显示：

    我们可以在一个config文件中，配置多个用户和集群及配套的上下文环境。

查看当前的上下文

```
# kubectl config view --kubeconfig=/tmp/newconfig.conf | grep current
current-context: kubernetes-admin@kubernetes
```

切换上下文

```
kubectl config use-context wangss@mycluster --kubeconfig=/tmp/newkubeconfig.conf
root@master1:/etc/kubernetes/pki# kubectl get sa --
kubeconfig=/tmp/newkubeconfig.conf        Error from server (Forbidden):
serviceaccounts is forbidden: User "wangss" cannot list resource
"serviceaccounts" in API group "" in the namespace "default"
```

小结

授权解析

RBAC基础

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

- 基本简介

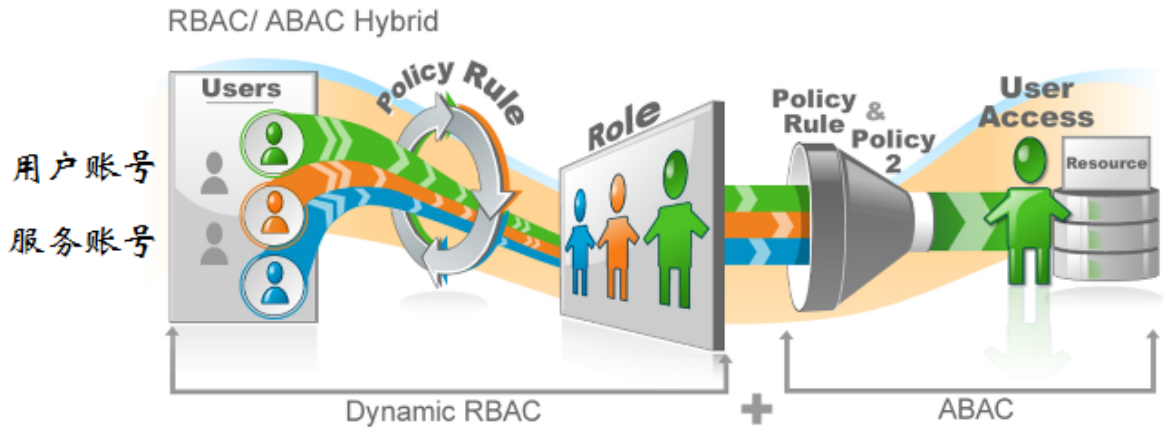
简介

RBAC使用rbac.authorization.k8s.io API Group 来实现授权决策，允许管理员通过 Kubernetes API 动态配置策略，要启用RBAC，需要在 apiserver 中添加参数--authorization-mode=RBAC，kubeadm安装的集群默认开启了RBAC，我们可以通过查看 Master 节点上 apiserver 的静态Pod定义文件：

```
$ cat /etc/kubernetes/manifests/kube-apiserver.yaml
...
    - --authorization-mode=Node,RBAC
...
```

Kubernetes的基本特性就是它的所有资源对象都是模型化的 API 对象，我们可以基于api-server对各种资源进行增、删、改、查等操作，但是这些操作涉及到的不仅仅是资源本身和动作，而且还涉及到资源和动作之间的内容，比如api的分组版本和资源和api的关联即权限授权等。

授权



所谓的授权，其实指的是，将某些subject对象赋予执行某些资源动作的权限。我们有时候会将其称为Group(权限组)，而这个组其实是有两部分组成：组名和组关联(也称绑定)。
简单来说：所谓的授权，其实是为用户授予xx角色

组成	解析
组名	其实是附加在某些资源上的一系列权限，对于k8s来说，它主要有两类：Role和clusterRole，其中Role主要是作用于namespace，而clusterRole主要作用于多个namespace，它们之间是一对多的关系。 我们为了将权限应用和具体权限列表分开描述，我们一般称权限列表为规则-rules
组关联(绑定)	所谓的组关联其实是将我们之前定义的Subject和对应的权限组关联在一起，表示某个Subject具有执行某个资源的一系列动作权限。它主要涉及到两个RoleBingding和ClusterRoleBingding。

- 属性解析

> 为了更好的解释这四个属性，我们按照其工作的范围将其划分为三类进行描述：
>
> `namespace`级别、`cluster`级别、混合级别。

namespace级别

术语	解析
rules	规则，是一组属于不同 API Group 资源上的权限操作的集合
role	表示在一个namespace中基于rules使用资源的权限，属于集群内部的 API 资源，主要涉及到操作和对象
RoleBingding	将Subject和Role绑定在一起，表示Subject可以在namespace中使用指定资源的role角色权限

cluster级别

术语	解析
ClusterRole	表示在一个cluster中基于rules使用资源的权限，属于集群内部的 API 资源，一个cluster有多个namespace即有多个role
ClusterRoleBingding	将Subject和ClusterRole绑定在一起，表示Subject可以在cluster中使用指定资源的ClusterRole角色权限

混合级别

术语	解析
RoleBingding	将Subject基于RoleBingding与ClusterRole绑定在一起，表示Subject可以使用所有namespace中指定资源的role角色，从而避免了多次role和user的RoleBingding。 同样的操作，站在ClusterRole的角度，我们可以理解为，用户得到的权限仅是ClusterRole的权限在Rolebinding所属的名称空间上的一个子集，也就是所谓的"权限降级"

场景1：
多个namespace中的role角色都一致，如果都使用内部的RoleBingding的话，每个namespace都必须单独创建role，而使用ClusterRole的话，只需要一个就可以了，大大的减轻批量使用namespace中的 RoleBingding 操作。
场景2：
我们对A用户要提升权限，但是，由于A处于考察期，那么我们暂时给他分配一个区域，测试一下它的运行效果。生活中的场景，提升张三为公司副总裁，但是由于是新手，所以加了一个限制 -- 主管销售范围的副总裁。

简单实践

属性解析

因为角色由于级别不一样，作用的范围也不同，所以我们这一节先从namespace级别的Role来学习，关于Role的属性，我们可以使用 kubectl explain role 的方式来查看一下：

```
apiVersion <string>
kind <string>
metadata     <Object>
rules        <[]Object>
  apiGroups             <[]string>
  nonResourceURLs       <[]string>
  resourceNames         <[]string>
  resources             <[]string>
  verbs                 <[]string> -required-
```

结果显示：
对于role来说，其核心的内容主要是rules的权限规则
在这么多rules属性中，最重要的是verbs就是权限条目，而且所有的属性都是可以以列表的形式累加存在

查看pod角色具有get、list权限的role的资源定义文件格式

```
]# kubectl create role pods-reader --verb=get,list --resource=pods --dry-run -o
yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null       时间信息
  name: pods-reader             role的名称
rules:                          授权规则
- apiGroups:                    操作的对象
  - ""                               所有权限
  resources:                    资源对象
  - pods                             pod的对象
  verbs:                             对pod允许的权限
  - get                              获取
  - list                                  查看
```

结果显示：
对于一个role必备的rules来说，他主要有三部分组成：apiGroup、resources、verbs
apiGroups 设定包含资源的api组，如果是多个，表示只要属于api组范围中的任意资源都可以操作
resources 位于apiGroup范围中的某些具体的资源对象
verbs 针对具体资源对象的一些具体操作
注意：
关于api组的信息获取，可以参照https://kubernetes.io/docs/reference/#api-reference

简单实践

按照上面的role格式，我们写一个role资源文件,允许用户操作 Deployment、Pod、RS 的所有权限

```
cat > 02-security-myrole.yaml <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: myrole
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["pods", "deployments", "replicasets"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
EOF
```

属性解析：

Pod属于 core 的 API Group，在YAML中用空字符就可以，Deployment 属于 apps 的 API Group，ReplicaSets属于extensions这个 API Group，所以 rules 下面的 apiGroups 的内容：["", "extensions", "apps"]

verbs是可以对这些资源对象执行的操作，如果是所有动作，也可以使用["*"]来代替。

初始化实例对象

```
kubectl apply -f 02-security-role-demo.yaml
```

查看效果

```
]# kubectl get role
NAME     CREATED AT
myrole   2021-10-06T03:07:54Z

]# kubectl describe role myrole
```

```
root@master1:~/role# kubectl describe role myrole
Name:         myrole
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources              Non-Resource URLs  Resource Names  Verbs
  ---------              -----------------  --------------  -----
  deployments            []                 []              [get list watch create update patch delete]
  pods                   []                 []              [get list watch create update patch delete]
  replicasets            []                 []              [get list watch create update patch delete]
  deployments.apps       []                 []              [get list watch create update patch delete]
  pods.apps              []                 []              [get list watch create update patch delete]
  replicasets.apps       []                 []              [get list watch create update patch delete]
  deployments.extensions  []                []              [get list watch create update patch delete]
  pods.extensions        []                 []              [get list watch create update patch delete]
  replicasets.extensions  []                []              [get list watch create update patch delete]
```

小结

RBAC绑定

学习目标

这一节，我们从 用户绑定、集群绑定、小结 三个方面来学习。

用户绑定

- UA绑定

属性简介

我们通过explain的命令来看一下rolebinding的属性信息

```
#] kubectl explain rolebinding
apiVersion   <string>
kind <string>
metadata     <Object>
roleRef      <Object> -required-
subjects     <[]Object>
```

结果显示：

对于角色绑定来说，主要涉及到两点：subject和对应的role权限列表，其中roleRef是必选项。

配置文件

我们以pod-sa-admin或者sswang的subject来与myrole进行一次模拟绑定查看属性效果

```
]# kubectl create rolebinding wangss-myrole --role=myrole --user=wangss  -o yaml
--dry-run
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: null
  name: wangss-myrole
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: myrole
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: wangss
```

结果显示：

对于roleRef和subjects分别写好其对应的资源属性信息即可

创建配置文件

```
kubectl create rolebinding wangss-myrole --role=myrole --user=wangss  -o yaml --
dry-run > 03-security-rolebinding.yaml
```

简单实践

创建rolebinding

```
kubectl create rolebinding myrolebinding --role=myrole --user=wangss
```

或者

```
kubectl  apply -f 03-security-rolebinding.yaml
```

注意：

如果是多个用户的话，可以多来几个 --user=subject 彼此间使用空格隔开即可

检查效果

```
]# kubectl get rolebindings
NAME            AGE
myrolebinding   10s
]# kubectl describe rolebindings. wangss-myrole
Name:         wangss-myrole
...
Subjects:
  Kind  Name          Namespace
  ----  ----          ---------
  User  wangss
```

可以看到：

```
    sswang 已经加入到 myrolebinding 角色绑定组中了
    这里的namespace没有写，表示默认的default命名空间


查看当前namespace中资源效果
]# kubectl get pod --context=wangss@mycluster --kubeconfig=/tmp/sswang.conf
NAME           READY   STATUS    RESTARTS   AGE
pod-sa-admin   1/1     Running   0          90m
注意：
    我们在创建wangss的时候没有指定命名空间，表示默认的default

查看default命名空间中，非pod、deployment、rs之外的资源
]# kubectl get secrets
NAME                  TYPE                                  DATA   AGE
admin-token-29w8g     kubernetes.io/service-account-token   3      156m
default-token-slqsq   kubernetes.io/service-account-token   3      3d23h
]# kubectl get secrets --context=wangss@mycluster --kubeconfig=/tmp/sswang.conf
Error from server (Forbidden): secrets is forbidden: User "wangss" cannot list
resource "secrets" in API group "" in the namespace "default"
结果显示：
    在我们role之外的内容我们都没有权限查看

查看非当前namespace中的资源
]# kubectl get pod -n kube-system --context=wangss@mycluster --
kubeconfig=/tmp/sswang.conf
Error from server (Forbidden): pods is forbidden: User "wangss" cannot list
resource "pods" in API group "" in the namespace "kube-system"
结果显示：
    在当前role权限范围内容，其它namespace中的资源，我们生效不了，因为role只是针对某个具体的
ns
```

- SA绑定

命令格式

```
查看SA的角色绑定格式
kubectl create rolebinding NAME --role=NAME [--
serviceaccount=namespace:serviceaccountname]
属性解析：
    我们在基于服务账号进行关联的时候，需要关注一下该SA所属的namespace信息。
```

简单实践

```
我们的自建的sa是admin而且是属于default空间，我们先将admin和myrole进行绑定，查看一下效果
kubectl create rolebinding myrolebinding1 --role=myrole --
serviceaccount=default:admin

检查效果
]# kubectl describe rolebindings myrolebinding1
Name:         myrolebinding1
...
Subjects:
  Kind            Name   Namespace
  ----            ----   ---------
  ServiceAccount  admin  default
结果显示
```

我们的admin加入到了myrolebinding1的绑定组中了，而且显示了特定的命名空间
由于sa账号在应用程序中的信息检查，我们没有办法直接命令行测试。所以这块我们在后续的服务认证中验证

集群绑定

简介

所谓的cluster级别的实践主要涉及到clusterRole和ClusterRoleBingding之间的操作，也就是说我们可以操作多个namespace空间的资源。

属性简介

```
#] kubectl explain clusterrole
aggregationRule      <Object>
apiVersion <string>
kind <string>
metadata     <Object>
rules        <[]Object>
    apiGroups               <[]string>
    nonResourceURLs         <[]string>
    resourceNames           <[]string>
    resources               <[]string>
    verbs                   <[]string> -required-
```

结果显示：
clusterrole相对于role的属性多了一个集中控制器的属性aggregationRule，而这是一个可选的属性

查看一个简单的配置格式

```
]# kubectl create clusterrole myclusterrole --verb=get,list --resource=pods -o
yaml --dry-run=client
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: myclusterrole
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  - list
```

结果显示：
单从模板的资源配置样式来说，他的配置信息与role的配置信息几乎一样。

角色创建

```
创建资源定义文件 - root用户操作
cat > 04-security-myclusterrole.yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

```
metadata:
  name: myclusterrole
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
EOF

初始化资源对象
kubectl apply -f 04-security-myclusterrole.yaml

检查效果
]$ kubectl get clusterrole | egrep 'NA|my'
NAME                                                   CREATED AT
myclasterrole                                          2021-10-06T23:28:08Z

]$ kubectl describe clusterrole myclasterrole
Name:         myclasterrole
...
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  pods       []                 []              [get list watch]
结果显示：
    在myclasterrole仅仅有pod资源的三种权限
```

角色绑定

```
命令格式
kubectl create clusterrolebinding NAME --clusterrole=NAME [--user=username] ...
属性解析
    对于clusterrolebinding来说，仅仅允许集群角色进行和其进行绑定，对于普通的role来说就无效了

我们将wangss用户和myclasterrole进行角色绑定，查看资源配置效果
]# kubectl create clusterrolebinding myclusterrolebinding --
clusterrole=myclusterrole --user=wangss --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: myclusterrolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole                          用户绑定类型
  name: myclasterrole                        用户绑定名称
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User                                 用户类型
  name: wangss                               用户名称
属性解析：
    这里的属性配置与我们之前的role和rolebinding的方法几乎一样
    区别就是kind和--clusterrole的不同

确定cluserrolebinding绑定
命令行方式
```

```
kubectl create clusterrolebinding myclusterrolebinding --
clusterrole=myclusterrole --user=wangss

配置文件方式 05-security-myclusterrolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: myclusterrolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: myclusterrole
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: wangss

检查效果
]$ kubectl get clusterrolebinding | egrep 'NA|my'
NAME                                                   AGE
myclusterrolebinding                                   23s

]$ kubectl describe clusterrolebinding myclusterrolebinding
Name:         myclusterrolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  myclasterrole
Subjects:
  Kind  Name    Namespace
  ----  ----    ---------
  User  wangss
结果显示：
    我们将wangss的账号，赋予了操作集群角色的权限
```

简单实践

```
切换账号查看效果
]$ kubectl get pod --context=wangss@mycluster  --kubeconfig=/tmp/sswang.conf
NAME           READY   STATUS    RESTARTS   AGE
pod-sa-admin   1/1     Running   0          165m

]$ kubectl get pod --context=wangss@mycluster -n kube-system  --
kubeconfig=/tmp/sswang.conf
NAME                              READY   STATUS    RESTARTS   AGE
etcd-master                       1/1     Running   9          4d
...
kube-scheduler-master             1/1     Running   1          4d
结果显示：
    对于clusterrolebinding来说，绑定后的user account不但可以看到当前namespace中的资源对象，还能看到其他namespace中的资源对象
```

小结

RBAC混合

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

功能需求

所谓的cluster级别的实践主要涉及到clusterRole和rolebinding之间的操作，也就是说虽然我们可以操作很大范围的权限，但是由于某个特殊的场景我们需要避免大的权限，采用rolebinding的方式来主动降低可行的执行权限，即，在混合级别实践的场景中，无论我们的用户是哪一个，我们只需要关联同一个clusterrole，那么这个用户只能在指定的namespace空间中进行响应的操作。

准备工作

为了避免之前所做的rolebinding和clusterrolebinding对混合实践的影响，我们需要将之前所做的操作全部清空，对subject账号和clusterrole、role无需变动

还原环境

```
kubectl delete rolebinding myrolebinding
kubectl delete rolebinding myrolebinding1
kubectl delete clusterrolebinding myclusterrolebinding
```

当前subject和clusterrole现状

```
]$ kubectl get clusterrole | egrep 'NA|my'
NAME                                                                   CREATED AT
myclasterrole                                                          2021-10-06T23:28:08Z

]$ kubectl config get-contexts
CURRENT   NAME                          CLUSTER      AUTHINFO
NAMESPACE
*         kubernetes-admin@kubernetes   kubernetes   kubernetes-admin
```

简单实践

- 混合实践

命令格式

查看混合实践格式

```
kubectl create rolebinding NAME --clusterrole=NAME|--role=NAME [--user=username]
```

属性解析：

对于rolebinding来说，我可以基于clusterrole和role两种角色进行绑定操作。

角色绑定

绑定角色

```
kubectl create rolebinding hunherole --clusterrole=myclusterrole --user=wangss
--dry-run=client -o yaml > 06-security-hunhebinding.yaml
```

定制该资源定义文件

```
# cat 06-security-hunhebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: RoleBinding
metadata:
  name: hunherole
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: myclusterrole
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: wangss
```

属性解析：

我们将自动生成的配置文件中的时间戳删除，然后给其设定了一个namespace的权限范围属性

应用资源对象

```
kubectl apply -f 06-security-hunhebinding.yaml
```

检查效果

```
]# kubectl get rolebindings
NAME        ROLE                         AGE
hunherole   ClusterRole/myclasterrole   7s
]# kubectl describe rolebindings hunherole
Name:         hunherole
...
Role:
  Kind:  ClusterRole
  Name:  myclusterrole
Subjects:
  Kind  Name    Namespace
  ----  ----    ---------
  User  wangss
```

结果显示：

我们已经将wangss采用rolebinding的方式赋予其clusterrole的角色权限

检查效果

```
]# kubectl get pod --context=wangss@mycluster  --kubeconfig=/tmp/sswang.conf
NAME           READY   STATUS    RESTARTS   AGE
pod-sa-admin   1/1     Running   0          3h8m

]# kubectl get pod --context=wangss@mycluster -n kube-system
Error from server (Forbidden): pods is forbidden: User "wangss" cannot list
resource "pods" in API group "" in the namespace "kube-system"
```

结果显示：

对于rolebinding的clusterrole来说，我们只能访问指定的default空间中的资源了。

- 权限实践

简介

我们知道对于自定义的myclusterrole来说，它对于pod只有查看的权限，没有删除的权限，查看效果

```
]# kubectl delete pod pod-sa-admin --context=wangss@mycluster --
kubeconfig=/tmp/sswang.conf
Error from server (Forbidden): pods "pod-sa-admin" is forbidden: User "wangss"
cannot delete resource "pods" in API group "" in the namespace "default"
```

结果显示：

我们无法对dafault命名空间的pod资源做权限之外的事情。

隐患

我们知道官方也给default空间提供了很多权限，admin的权限就是极大的

```
]# kubectl get clusterrole admin -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
...
- apiGroups:
  - ""
  resources:
  - pods
  - pods/attach
  - pods/exec
  - pods/portforward
  - pods/proxy
  verbs:
  - create
  - delete
  - deletecollection
  - patch
  - update
...
```

可以看到：

admin的clusterrole它是在default命名空间中的，而且对pod有各种操作权限

简单实践

定制混合绑定

```
kubectl create rolebinding default-ns-admin  --clusterrole=admin --user=wangss
kubectl delete rolebindings hunherole
```

查看效果

```
]# kubectl describe rolebindings default-ns-admin
Name:         default-ns-admin
...
Subjects:
  Kind  Name    Namespace
  ----  ----    ---------
  User  wangss
```

资源操作

```
]# kubectl get pod --context=wangss@mycluster --kubeconfig=/tmp/sswang.conf
NAME           READY   STATUS    RESTARTS   AGE
pod-sa-admin   1/1     Running   0          3h22m

]# kubectl delete pod pod-sa-admin --context=wangss@mycluster --
kubeconfig=/tmp/sswang.conf
pod "pod-sa-admin" deleted

]# kubectl get pod --context=wangss@mycluster -n kube-system --
kubeconfig=/tmp/sswang.conf
Error from server (Forbidden): pods is forbidden: User "wangss" cannot list
resource "pods" in API group "" in the namespace "kube-system"
```

结果显示：

我们只能管理当前命名空间的资源，其他命名空间的资源是管理不了的。

小结

服务认证

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

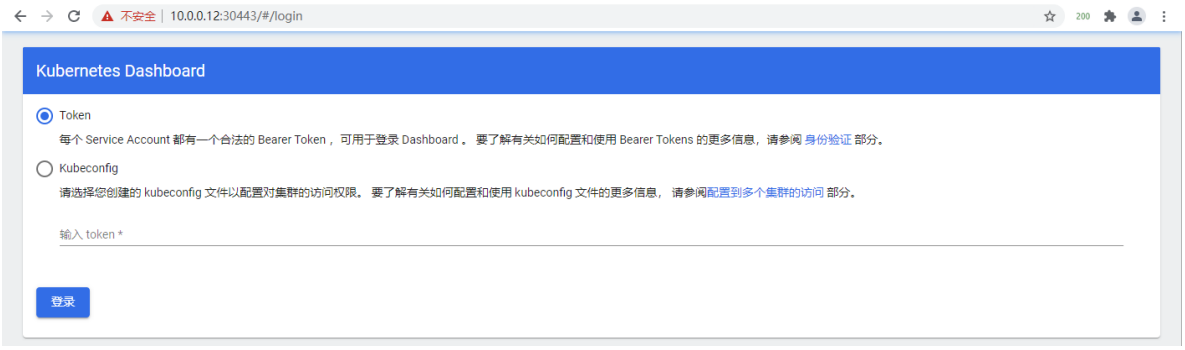
需求

> 关于用户、组相关的认证以及授权我们已经学习了，而且还进行了相当的知识点实践，尤其是user account的认证和授权及其准入控制，但是我们对serviceaccount的准入控制涉及的不太深，那么剩下的就是关于服务认证的知识了，kubernetes软件给我们提供了非常多的组件，这些组件在应用的过程中就涉及到和服务认证相关的知识，尤其是dashboard组件部署的后期阶段，就用到了很多和服务认证相关的内容，而且只有先配置serviceaccount才能继续执行下去。
>
> 那么接下来我们就在部署dashboard的时候，掌握服务认证相关的知识

基本环境

> 获取官方的yaml文件
> wget https://raw.githubusercontent.com/kubernetes/dashboard/v2.3.1/aio/deploy/recommended.yaml
>
> 修改配置
>
> ```
> kind: Service
> apiVersion: v1
> metadata:
>   ...
> spec:
>   ports:
>     - port: 443
>       targetPort: 8443
>       nodePort: 30443               # 添加此项
>   type: NodePort                    # 添加此项
>   selector:
>     k8s-app: kubernetes-dashboard
> ```
>
> 注意：
>
> 我们在这里做的修改主要是镜像源的修改和启动端口的配置，采用NodePort 可以在所有的节点上都开启一个端口用于dashboard访问
>
> 依赖两个镜像：kubernetesui/dashboard:v2.3.1 和 kubernetesui/metrics-scraper:v1.0.6
>
> 将这两个镜像都改为 10.0.0.19:80/google_containers
>
> 应用配置文件
> kubectl apply -f recommended.yaml

在Firefox浏览器输入Dashboard访问地址：https://10.0.0.12:30443



认证解析

对于dashboard来说，他的认证方法主要有两种：令牌认证和文件认证。由于涉及面比较多，我们通过两节的内容来进行讲解，在这里我们用令牌的方式来学习完整的服务认证流程。

令牌认证
- 基于认证用户的唯一令牌来进行认证，有默认的超时机制，一会儿就失效了

文件认证
- 基于账号的token信息，创建kubeconfig文件，实现长久的认证方式。

根据我们之前对serviceaccount的工作流程的学习，对于dashboard的配置也应该遵循相应的操作流程：

1、创建专用的serviceaccount
2、创建对应的权限角色
3、将serviceaccount和权限角色进行关联绑定

清理之前的记录

```
kubectl delete clusterrolebindings kubernetes-dashboard
kubectl delete serviceaccount dashboard-admin -n kube-system
```

简单实践

- 令牌认证

认证授权

命令方式：
创建专用的服务账户

```
kubectl create serviceaccount dashboard-admin -n kube-system
```

使用集群角色绑定

```
kubectl create clusterrolebinding  dashboard-admin --clusterrole=cluster-admin --serviceaccount=kube-system:dashboard-admin
```

参数详解：

```
--clusterrole=集群角色名称
--serviceaccount=命名空间:serviceaccountname
```

由于我们这里配置的是全局用户，所有命名空间是kube-system
dashboard需要操作各种资源的所有权限，所以我们需要和cluster-admin的clusterrole进行绑定

资源定义文件方式 07-security-dashboard-admin.yaml

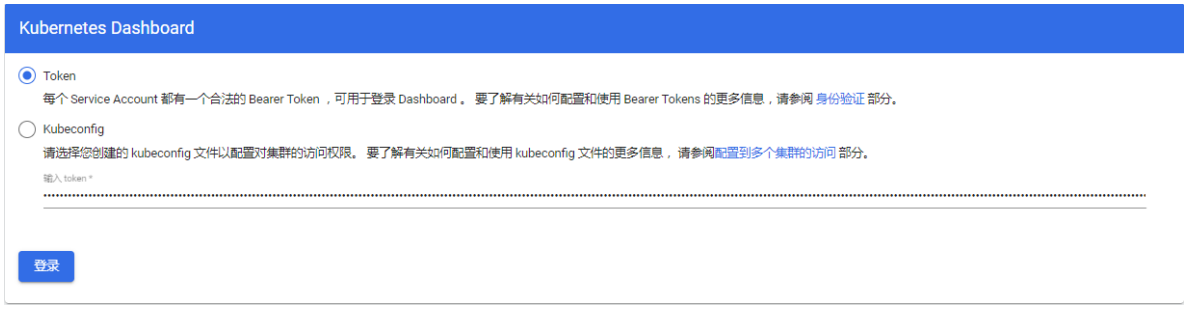
```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: dashboard-admin
```

```
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: dashboard-admin
  namespace: kube-system

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dashboard-admin
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
```

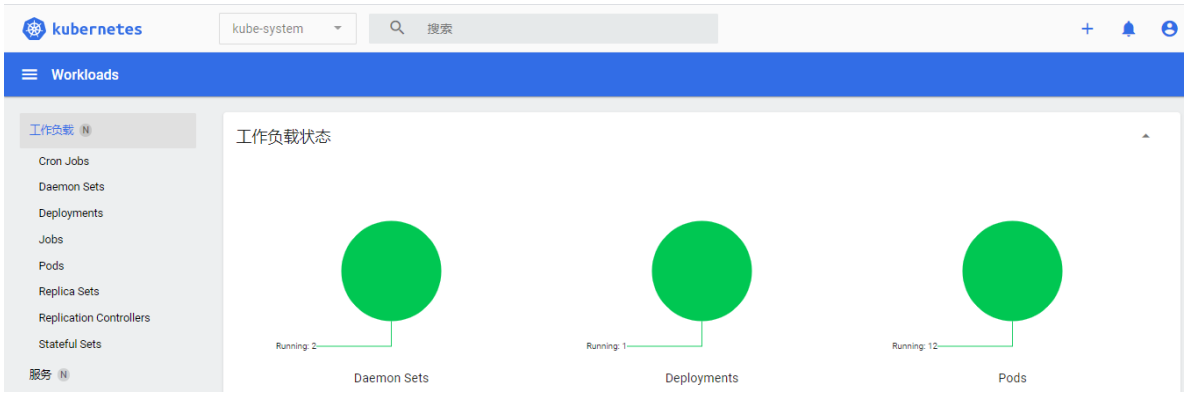
token信息

```
我们要使用token的方式来登录dashboard，而token保存在secrets中，所以我们需要基于该
serviceaccount账号的配套secrets中获取全局令牌信息
方法一：
kubectl describe secrets -n kube-system $(kubectl -n kube-system get secret |
awk '/dashboard-admin/{print $1}')
方法二：
]# kubectl -n kube-system get secret | grep dashboard-admin
dashboard-admin-token-5sv88                      kubernetes.io/service-account-
token   3      3d5h

]# kubectl describe secrets -n kube-system dashboard-admin-token-5sv88
Name:         dashboard-admin-token-5sv88
Namespace:    kube-system
...
Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1025 bytes
namespace:  11 bytes
token:      eyJhbGciOiJSU...vrJaQ
结果显示：
    配置好serviceaccount之后，会自动帮我们生成一个dashboard-admin-token-xxx格式的
secret信息
    token：属性后面的加密信息，就是我们需要的内容，复制该内容到dashboard界面中的令牌输入框
```

使用输出的token登录Dashboard。



认证通过后，登录Dashboard首页如图



结果显示：

我们这里可以对任何用户空间进行管理操作

有时候我们需要不同的用户查看不同的资源内容，那么接下来我们需要对serviceaccount进行某个特定namespace环境中的实践。

- 命名空间用户

需求

所谓命名空间的用户实践，其实就是我们所说的用户权限缩小的混合实践，即在将subject和clusterrole使用rolebinding的方式进行关联。

命令方式

```
创建专用的服务账户
kubectl create serviceaccount def-ns-admin -n default

使用集群角色绑定
kubectl create rolebinding def-ns-admin  --clusterrole=admin --
serviceaccount=default:def-ns-admin
```

文件方式

```
资源配置文件 08-security-dashboard-admin-ns.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: def-ns-admin
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
```

```
metadata:
  name: def-ns-admin
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- kind: ServiceAccount
  name: def-ns-admin
  namespace: default
```

获取全局令牌信息

```
]# kubectl get secret | grep def-ns-admin
def-ns-admin-token-f7qcl   kubernetes.io/service-account-token   3      47m

]# kubectl describe secret def-ns-admin-token-f7qcl
Name:         def-ns-admin-token-f7qcl
Namespace:    default
...
Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1025 bytes
namespace:  7 bytes
token:      eyJhbG...1aHVQ
```

结果显示：

token：属性后面的加密信息，就是我们需要的内容，复制该内容到dashboard界面中的令牌输入框

认证通过后，查看效果

结果显示：

当前的dashboard，仅仅能管理default命名空间下的资源

文件认证

学习目标

这一节，我们从 通用认证、单用户认证、小结 三个方面来学习。

通用认证

服务账号

参考 服务认证中令牌认证当中的serviceaccount创建

```
root@master1:~# kubectl get sa dashboard-admin -n kube-system
NAME              SECRETS   AGE
dashboard-admin   1         7m7s
root@master1:~# kubectl get sa def-ns-admin
NAME           SECRETS   AGE
def-ns-admin   1         3m18s
```

集群配置

```
设置集群信息
kubectl config set-cluster kubernetes --certificate-
authority=/etc/kubernetes/pki/ca.crt  --server="https://10.0.0.12:6443" --embed-
certs=true --kubeconfig=/root/kubeadmin.conf
注意：
    设定claster信息的时候，必须基于集群初始化时候的认证信息--ca证书：
    证书认证：--certificate-authority

查看效果
]# kubectl config view --kubeconfig=/root/kubeadmin.conf
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.0.0.12:6443
  name: kubernetes
contexts: []
current-context: ""
kind: Config
preferences: {}
users: []
```

用户配置

```
获取认证信息
]#  kubectl get secrets -n kube-system | grep admin-token
dashboard-admin-token-5sv88                      kubernetes.io/service-account-
token   3      3d6h

]# kubectl get secrets dashboard-admin-token-5sv88 -n kube-system -o
jsonpath='{.data.token}' | base64 -d
eyJhbGciO...CaYoZTug

]# KUBEADMIN_TOKEN=$(kubectl get secret dashboard-admin-token-5sv88 -n kube-
system -o jsonpath='{.data.token}' |base64 -d)
注意：
    是 kube-system 命名空间的认证信息

设置用户信息
kubectl config set-credentials kubeadmin --token=$KUBEADMIN_TOKEN --
kubeconfig=/root/kubeadmin.conf
检查效果
]# kubectl config view --kubeconfig=/root/kubeadmin.conf
apiVersion: v1
...
```

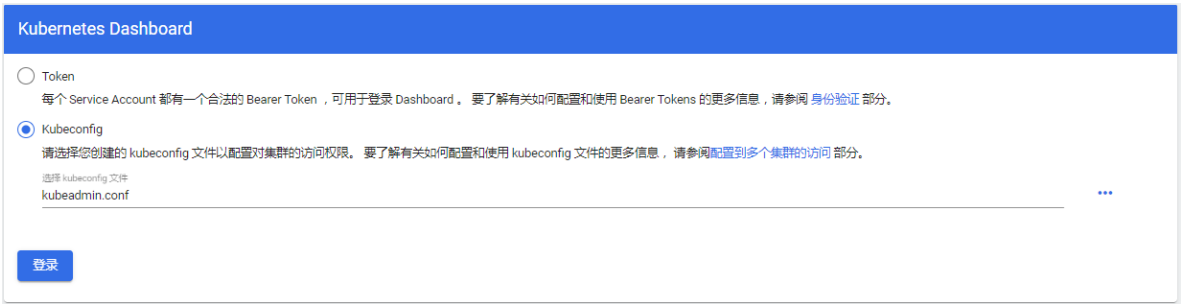
```
contexts: []
current-context: ""
kind: Config
preferences: {}
users:
- name: kubeadmin
  user:
    token: eyJhbGciO...GQaCaYoZTug
```

上下文配置

```
设置上下文信息
kubectl config set-context kubeadmin@kubernetes --cluster=kubernetes --
user=kubeadmin --kubeconfig=/root/kubeadmin.conf
检查效果
]# kubectl config view --kubeconfig=/root/kubeadmin.conf
                                    apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.0.0.12:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubeadmin
  name: kubeadmin@kubernetes
current-context: ""
kind: Config
preferences: {}
users:
- name: kubeadmin
  user:
    token: eyJhbGciO...GQaCaYoZTug

切换用户
kubectl config use-context kubeadmin@kubernetes --
kubeconfig=/root/kubeadmin.conf
kubectl config view --kubeconfig=/root/kubeadmin.conf

确认效果
root@master1:~# kubectl get sa --kubeconfig=/root/kubeadmin.conf
NAME           SECRETS   AGE
admin          1         34h
def-ns-admin   1         8m40s
default        1         3d22h
```

登录效果

```
下载kubeadmin.conf
sz kubeadmin.conf
浏览器上，采用kubeconfig方式上传kubeadmin.conf文件，查看效果
```



结果显示：
　　可以看到所有用户空间效果

单用户认证

服务账户

参考 服务认证中命名空间用户当中的 serviceaccount 创建

集群配置

```
设置集群信息
kubectl config set-cluster kubernetes --certificate-
authority=/etc/kubernetes/pki/ca.crt  --server="https://10.0.0.12:6443" --embed-
certs=true --kubeconfig=/root/def-ns-admin.conf

kubectl config view --kubeconfig=/root/def-ns-admin.conf
```

用户配置

```
获取秘钥信息
kubectl get secret
kubectl describe secret def-ns-admin-token-f7qcl
DEF_NS_ADMIN_TOKEN=$(kubectl get secret  def-ns-admin-token-f7qcl  -o jsonpath=
{.data.token} |base64 -d)

设置用户信息
kubectl config set-credentials def-ns-admin --token=$DEF_NS_ADMIN_TOKEN --
kubeconfig=/root/def-ns-admin.conf
kubectl config view --kubeconfig=/root/def-ns-admin.conf
```

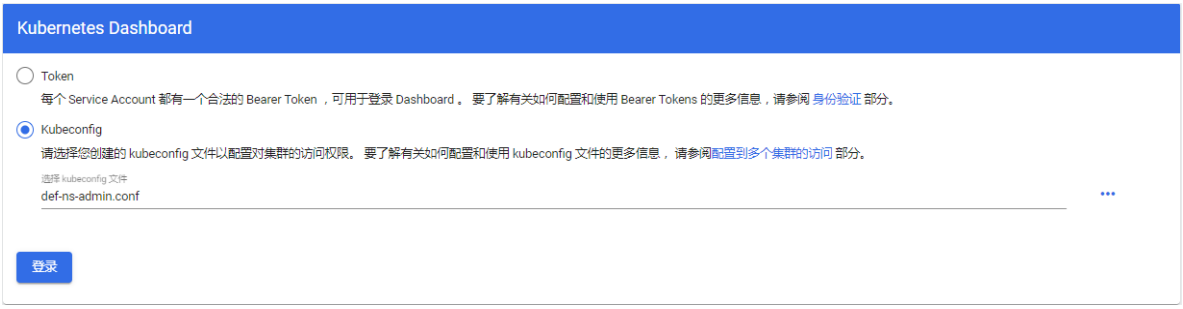
上下文配置

```
配置上下文信息
kubectl config set-context def-ns-admin@kubernetes --cluster=kubernetes --
user=def-ns-admin --kubeconfig=/root/def-ns-admin.conf
kubectl config view --kubeconfig=/root/def-ns-admin.conf
切换用户
kubectl config use-context def-ns-admin@kubernetes --kubeconfig=/root/def-ns-
admin.conf
kubectl config view --kubeconfig=/root/def-ns-admin.conf

查看效果
root@master1:~# kubectl get sa --kubeconfig=def-ns-admin.conf
```

```
NAME           SECRETS   AGE
admin          1         34h
def-ns-admin   1         13m
default        1         3d22h
root@master1:~# kubectl get sa --kubeconfig=def-ns-admin.conf -n kube-system
Error from server (Forbidden): serviceaccounts is forbidden: User
"system:serviceaccount:default:def-ns-admin" cannot list resource
"serviceaccounts" in API group "" in the namespace "kube-system"
```

登录效果

```
下载def-ns-admin.conf
sz def-ns-admin.conf
浏览器上，采用kubeconfig方式上传def-ns-admin.conf文件，查看效果
```




```
结果显示：
    当前的dashboard，仅仅能管理default命名空间下的资源
```

小结

dashboard进阶

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

简介

```
    我们在搭建K8S环境的时候，涉及到一个私有容器仓库Harbor，Harbor是VMware开源的一个私有容器镜像仓库，对于k8s默认的可视化界面dashboard虽然功能很好，但是在有些场景中，它并不太美观和便捷。VMware为此开源了一个Octant工具，它看来它更像一个Dashboard的代替品，不仅仅外观漂亮了很多，而且功能也丰富了很多，尤其是便捷性。
```

```
官方代码：https://octant.dev/
github：https://github.com/vmware-tanzu/octant
最新版本：v0.24.0 (20210910)
```

官方定位

```
官网介绍：
Octant is an open source developer-centric(以开发人员为中心) web interface for
Kubernetes that lets you inspect(检查) a Kubernetes cluster and its applications.

github介绍：
Octant is a tool for developers to understand how applications run on a
Kubernetes cluster. It aims(主旨) to be part of the developer's toolkit for
gaining(增加) insight(深入了解) and approaching(接近) complexity found in
Kubernetes. Octant offers a combination(组合) of introspective(内省、反省)
tooling, cluster navigation(导航), and object management along with a plugin
system to further(进一步) extend its capabilities.
```

工具功能

特点	解析
Resource Viewer	Graphically visualize relationships between objects in a Kubernetes cluster. The status of individual objects are represented by color to show workload performance.
Summary View	Consolidated status and configuration information in a single page aggregated from output typically found using multiple kubectl commands.
Port Forward	Forward a local port to a running pod with a single button for debugging applications and even port forward multiple pods across namespaces.
Log Stream	View log streams of pod and container activity for troubleshooting or monitoring without holding multiple terminals open.
Label Filter	Organize workloads with label filtering for inspecting clusters with a high volume of objects in a namespace.
Cluster Navigation	Easily change between namespaces or contexts across different clusters. Multiple kubeconfig files are also supported.
Plugin System	Highly extensible plugin system for users to provide additional functionality through gRPC. Plugin authors can add components on top of existing views.

简单实践

- 环境搭建

获取软件

```
wget https://github.com/vmware-
tanzu/octant/releases/download/v0.24.0/octant_0.24.0_Linux-64bit.deb
```

安装软件

```
dpkg -i ./octant_0.24.0_Linux-64bit.deb
```

检查效果

```
root@master1:~# octant version
Version:  0.24.0
Git commit:  5a8648921cc2779eb62a0ac11147f12aa29f831c
Built:  2021-09-09T01:54:00Z
```

命令帮助

安装完成后，使用octant命令启动即可，启动前需要指定一些参数，使用octant --help可查命令帮助

```
root@master1:~# octant --help
octant is a dashboard for high bandwidth cluster analysis operations

Usage:
  octant [flags]
  octant [command]

Available Commands:
  completion  generate the autocompletion script for the specified shell
  help        Help about any command
  version     Show version

Flags:
      --context string                 initial context
      --disable-cluster-overview       disable cluster overview
      --enable-feature-applications    enable applications feature
      --kubeconfig string              absolute path to kubeConfig file
                                        指定k8s集群的config文件
  -n, --namespace string               initial namespace
      --namespace-list strings         a list of namespaces to use on start
      --plugin-path string             plugin path
  -v, --verbose                        turn on debug logging
      --client-max-recv-msg-size int   client max receiver message size (default
16777216)
      --accepted-hosts string          accepted hosts list [DEV]
      --client-qps float32             maximum QPS for client [DEV] (default
200)
      --client-burst int               maximum burst for client throttle [DEV]
(default 400)
      --disable-open-browser           disable automatic launching of the
browser [DEV]
      --disable-origin-check           disable cross origin resource check
  -c, --enable-opencensus              enable open census [DEV]
      --klog-verbosity int             klog verbosity level [DEV]
      --listener-addr string           listener address for the octant frontend
[DEV]
                                        指定访问监听的地址及端口
      --local-content string           local content path [DEV]
      --proxy-frontend string          url to send frontend request to [DEV]
      --ui-url string                  dashboard url [DEV]
                                        Dashboard面板访问域名
      --browser-path string            the browser path to open the browser on
                                        访问路径
      --memstats string                log memory usage to this file
      --meminterval string             interval to poll memory usage (requires -
-memstats), valid time units are "ns", "us" (or "µs"), "ms", "s", "m", "h".
(default "100ms")
```

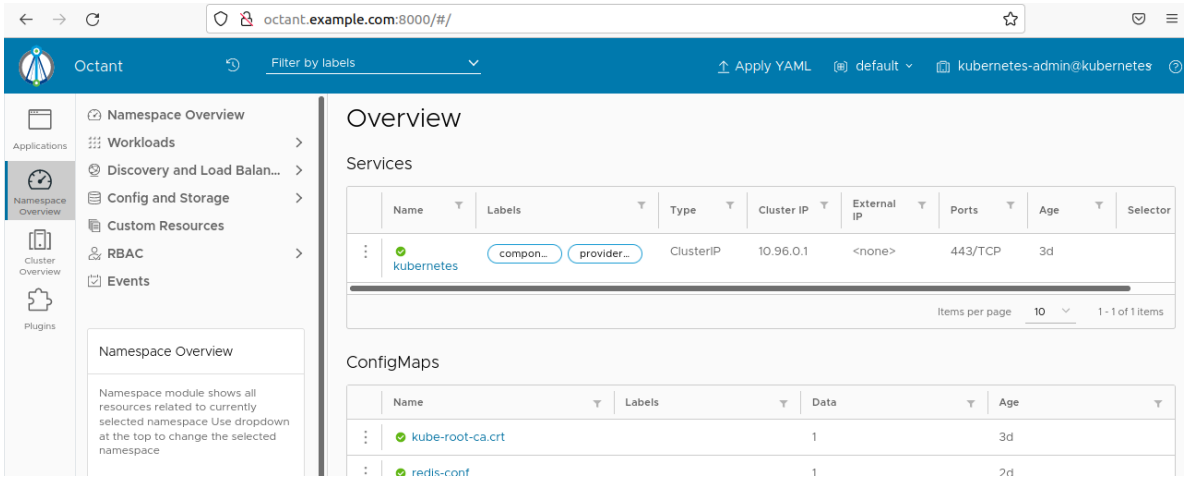
```
  -h, --help                              help for octant

Use "octant [command] --help" for more information about a command.
```

程序启动

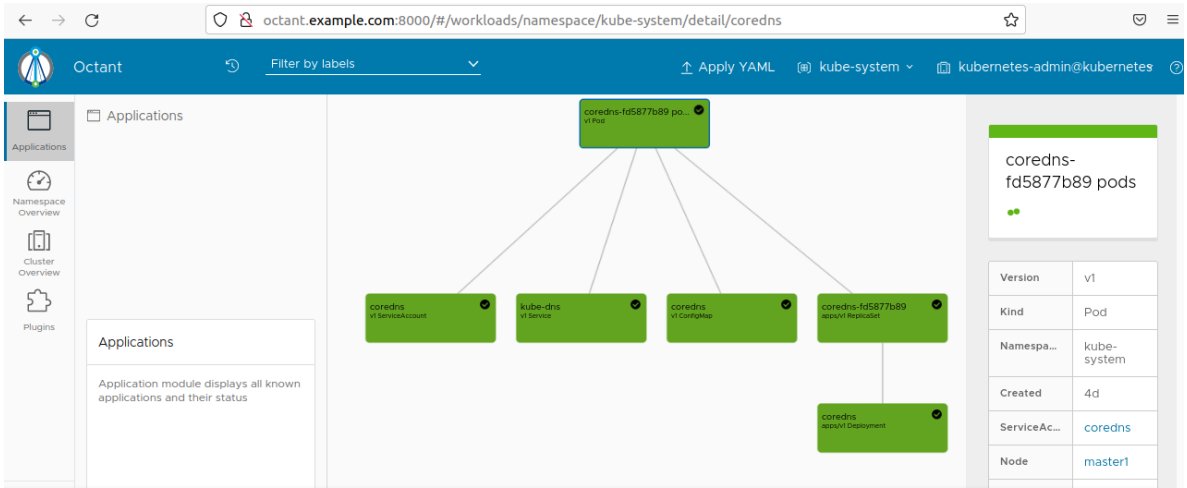
```
octant --kubeconfig=/etc/kubernetes/admin.conf --listener-addr=0.0.0.0:8000 --ui-url=octant.example.com --namespace-list kube-system,default,kubernetes-dashboard

注意：
    它会自动弹出一个可视化界面，默认是火狐浏览器
```

页面访问

```
添加域名解析
10.0.0.12 octant.example.com

在浏览器中输入 octant.example.com:8000
```



点击首页



小结

准入解析

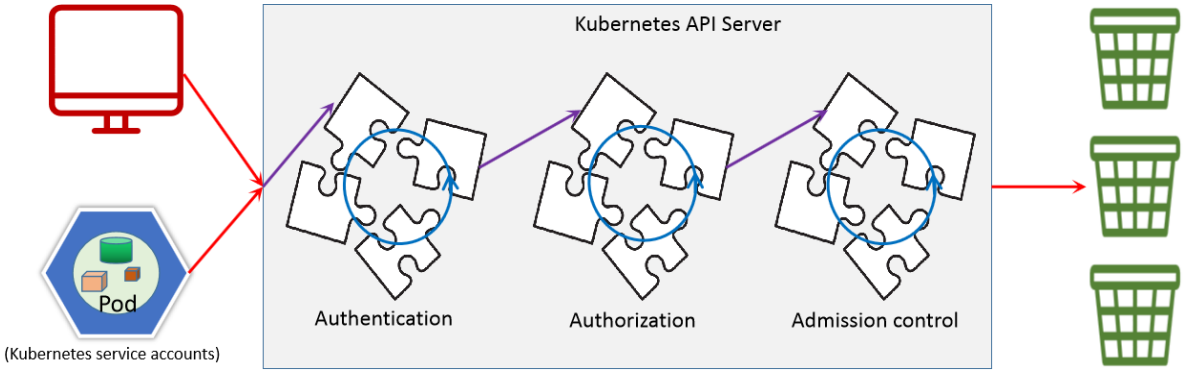
准入基础

学习目标

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

准入机制



所谓的"准入机制"，指的是经过了 用户认证、角色授权之后，当进行一些写操作的时候，需要遵循的一些原则性要求。准入机制有一大堆的 "准入控制器" 组成，这些准入控制器编译进 kube-apiserver 二进制文件，由集群管理员进行配置。

这些控制器中，最主要的就是：MutatingAdmissionWebhook(变更) 和 ValidatingAdmissionWebhook(验证)，变更（mutating）控制器可以修改被其接受的对象；验证（validating）控制器则不行。

准入控制过程分为两个阶段，运行变更准入控制器 和 运行验证准入控制器。 实际上，某些控制器既是变更准入控制器又是验证准入控制器。如果任何一个阶段的任何控制器拒绝了该请求，则整个请求将立即被拒绝，并向终端用户返回一个错误。

最后，除了对对象进行变更外，准入控制器还可以有其它作用：将相关资源作为请求处理的一部分进行变更。增加使用配额就是一个典型的示例，说明了这样做的必要性。 此类用法都需要相应的回收或回调过程，因为任一准入控制器都无法确定某个请求能否通过所有其它准入控制器。

准入控制功能启用

根据我们刚才的学习，准入控制器需要在kube-apiserver中进行配置，我们的kubeadm的kube-apiserver的配置文件中就通过属性开启了准入控制器：

```
root@master1:~# grep -ni 'admission' /etc/kubernetes/manifests/kube-apiserver.yaml
20:    - --enable-admission-plugins=NodeRestriction
```

根据上面的内容显示，开启用enbable，那么关闭控制器也比较方便：

```
    --disable-admission-plugins=控制器列表
```

查看默认开启的控制器

```
kubectl -n kube-system exec -it kube-apiserver-master1 -- kube-apiserver -h |
grep 'enable-admission-plugins strings'
```

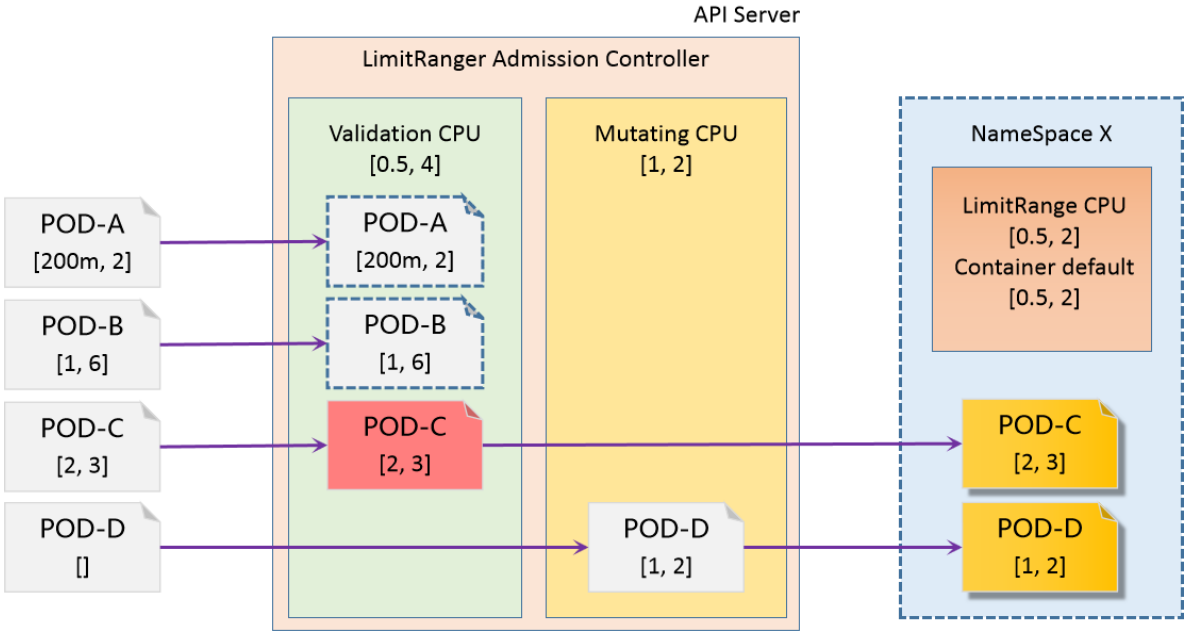
```
root@master1:~# kubectl -n kube-system exec -it kube-apiserver-master1 -- kube-apiserver -h | grep 'enable-admission-plugins string
s'
      --enable-admission-plugins strings       admission plugins that should be enabled in addition to default enabled ones (Namesp
aceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageCl
ass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateS
ubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota). Comma-delimited list
of admission plugins: AlwaysAdmit, AlwaysDeny, AlwaysPullImages, CertificateApproval, CertificateSigning, CertificateSubjectRestric
tion, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, DenyServiceExternalIPs, EventRateLimit, ExtendedResourceT
oleration, ImagePolicyWebhook, LimitPodHardAntiAffinityTopology, LimitRanger, MutatingAdmissionWebhook, NamespaceAutoProvision, Nam
espaceExists, NamespaceLifecycle, NodeRestriction, OwnerReferencesPermissionEnforcement, PersistentVolumeClaimResize, PersistentVol
umeLabel, PodNodeSelector, PodSecurity, PodSecurityPolicy, PodTolerationRestriction, Priority, ResourceQuota, RuntimeClass, Securit
yContextDeny, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionWebhook. The order of plugins
 in this flag does not matter.
```

结果显示：默认启用了 18个 准入控制器。而可以启用的准入控制器有 35 个。

这些控制器有很多，与我们的pod资源控制相关的准入控制器主要有以下几个

控制器	解析
limitranger	为Pod添加默认的计算资源需求和计算资源限制；以及存储资源需求和存储资源限制； 支持分别在容器和Pod级别进行限制；
resourcequota	限制资源数量，限制计算资源总量，存储资源总量；资源类型名称ResourceQuota
podsecuritpolicy	在集群级别限制用户能够在Pod上可配置使用的所有securityContext。由于RBAC的加强，该功能在 Kubernetes v1.21 版本中被弃用，将在 v1.25 中删除

limitranger示意图



psp资源属性

```
apiVersion: policy/v1beta1                  # PSP资源所属的API群组及版本
kind: PodSecurityPolicy                     # 资源类型标识
metadata:
  name <string>                             # 资源名称
spec:
  allowPrivilegeEscalation  <boolean>       # 是否允许权限升级
```

```
  allowedCSIDrivers <[]Object>                #内联CSI驱动程序列表，必须在Pod规范中显式
定义
  allowedCapabilities <[]string>              # 允许使用的内核能力列表，“*”表示all
  allowedFlexVolumes <[]Object>               # 允许使用的Flexvolume列表，空值表示“all
  allowedHostPaths <[]Object>                 # 允许使用的主机路径列表，空值表示all
  allowedProcMountTypes <[]string>            # 允许使用的ProcMountType列表，空值表示
默认
  allowedUnsafeSysctls <[]string>             # 允许使用的非安全sysctl参数，空值表示不允
许
  defaultAddCapabilities  <[]string>          # 默认即添加到Pod对象的内核能力，可被drop
  defaultAllowPrivilegeEscalation <boolean> # 是否默认允许内核权限升级
  forbiddenSysctls  <[]string>                # 禁止使用的sysctl参数，空表示不禁用
  fsGroup <Object>                            # 允许在SecurityContext中使用的
fsgroup，必选字段
    rule <string>                             # 允许使用的FSGroup的规则，支持RunAsAny
和MustRunAs
    ranges <[]Object>                         # 允许使用的组ID范围，需要与MustRunAs规则
一同使用
      max  <integer>                          # 最大组ID号
      min  <integer>                          # 最小组ID号
  hostIPC <boolean>                           # 是否允许Pod使用hostIPC
  hostNetwork <boolean>                       # 是否允许Pod使用hostNetwork
  hostPID <boolean>                           # 是否允许Pod使用hostPID
  hostPorts <[]Object>                        # 允许Pod使用的主机端口暴露其服务的范围
    max  <integer>                            # 最大端口号，必选字段
    min  <integer>                            # 最小端口号，必选字段
  privileged  <boolean>                       # 是否允许运行特权Pod
  readOnlyRootFilesystem  <boolean>           # 是否设定容器的根文件系统为“只读”
  requiredDropCapabilities <[]string>         # 必须要禁用的内核能力列表
  runAsGroup  <Object>                        # 允许Pod在runAsGroup中使用值列表，未定
义表示不限制
  runAsUser <Object>                          # 允许Pod在runAsUser中使用的值列表，必选
字段
    rule <string>                             # 支持RunAsAny、MustRunAs和
MustRunAsNonRoot
    ranges <[]Object>                         # 允许使用的组ID范围，需要
跟“MustRunAs”规则一同使用
      max  <integer>                          # 最大组ID号
      min  <integer>                          # 最小组ID号
  runtimeClass <Object>                       # 允许Pod使用的运行类，未定义表示不限制
    allowedRuntimeClassNames <[]string>       # 可使用的runtimeClass列表，“*”表示all
    defaultRuntimeClassName <string>          # 默认使用的runtimeClass
  seLinux <Object>                            # 允许Pod使用的selinux标签，必选字段
    rule <string>           # MustRunAs表示使用seLinuxOptions定义的值；RunAsAny表示
可使用任意值
    seLinuxOptions  <Object>                  # 自定义seLinux选项对象，与MustRunAs协作
生效
  supplementalGroups  <Object>                # 允许Pod在SecurityContext中使用附加
组，必选字段
  volumes <[]string>                          # 允许Pod使用的存储卷插件列表，空表示禁
用，*表示全部
```

注意：

由于PSP的准入控制非常严格，而且默认情况下，不允许任何未被k8s集群所承认的资源正常使用，所以，k8s集群默认情况下，没有正常启用。而且psp资源也不会被生效

如果需要使用psp资源的话，需要自己来进行相关功能的启用。

简单实践

- 资源策略

limitranger实践

```
apiVersion: v1
kind: LimitRange
metadata:
  name: storagelimits
spec:
  limits:
  - type: PersistentVolumeClaim
    max:
      storage: "10Gi"
    min:
      storage: "1Gi"
---
apiVersion: v1
kind: LimitRange
metadata:
  name: core-resource-limits
spec:
  limits:
    - type: Pod
      max:
        cpu: "4"
        memory: "4Gi"
      min:
        cpu: "500m"
        memory: "16Mi"
    - type: Container
      max:
        cpu: "4"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "2"
        memory: "512Mi"
      defaultRequest:
        cpu: "500m"
        memory: "64Mi"
      maxLimitRequestRatio:
        cpu: "4"
    - type: PersistentVolumeClaim
      max:
        storage: "10Gi"
      min:
        storage: "1Gi"
      default:
        storage: "5Gi"
      defaultRequest:
        storage: "1Gi"
      maxLimitRequestRatio:
```

```
        storage: "5"
```

属性解析：

maxLimitRequestRatio 用于设定 上阈值和下阈值之间的比例。

应用资源定义文件

```
kubectl apply -f 09-security-limitrange-test.yaml
```

确认效果

```
root@master1:~/admission# kubectl get limitranges
NAME                  CREATED AT
pod-resource-limits   2021-10-07T04:36:55Z
root@master1:~/admission# kubectl describe limitranges core-resource-limits
Name:                   pod-resource-limits
Namespace:              default
Type                    Resource  Min   Max   Default Request  Default Limit  Max
Limit/Request Ratio
----                    --------  ---   ---   ---------------  -------------  ---
-
Pod                     cpu       500m  4     -                -              -
Pod                     memory    16Mi  4Gi   -                -              -
Container               cpu       100m  4     500m             2              4
Container               memory    4Mi   1Gi   64Mi             512Mi          -
PersistentVolumeClaim   storage   1Gi   10Gi  1Gi              5Gi            5

root@master1:~/admission# kubectl describe limitranges storagelimits
Name:                   storagelimits
Namespace:              default
Type                    Resource  Min  Max   Default Request  Default Limit  Max
Limit/Request Ratio
----                    --------  ---  ---   ---------------  -------------  ---
PersistentVolumeClaim   storage   1Gi  10Gi  -                -              -
```

```
root@master1:~/admission# kubectl run pod --
image=10.0.0.19:80/mykubernetes/pod_test:v0.1
pod/pod created
root@master1:~/admission# kubectl describe pod pod
...
    Limits:
      cpu:     2
      memory:  512Mi
    Requests:
      cpu:        500m
      memory:     64Mi
      ...
```

定制测试pod

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-test
spec:
  containers:
  - name: pod-test
    image: 10.0.0.19:80/mykubernetes/pod_test:v0.1
    resources:
      requests:
        memory: 128Mi
```

```
        cpu: 1
      limits:
        memory: 1Gi
        cpu: 2

应用资源文件
kubectl apply -f 10-security-limitrange-pod-test.yaml

确认效果
root@master1:~/admission# kubectl get pod pod-test
NAME       READY   STATUS    RESTARTS   AGE
pod-test   0/1     Pending   0          42s
root@master1:~/admission# kubectl describe pod pod-test
...
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  18s   default-scheduler  0/3 nodes are available: 3
Insufficient cpu.
```

resourcequota

```
资源定义文件
apiVersion: v1
kind: ResourceQuota
metadata:
  name: resourcequota-test
spec:
  hard:
    pods: "5"
    count/services: "5"
    count/configmaps: "5"
    count/secrets: "5"
    count/cronjobs.batch: "2"
    requests.cpu: "2"
    requests.memory: "4Gi"
    limits.cpu: "4"
    limits.memory: "8Gi"
    count/deployments.apps: "2"
    count/statefulsets.apps: "2"
    persistentvolumeclaims: "6"
    requests.storage: "20Gi"

应用资源文件
kubectl apply -f 11-security-resourcequota-test.yaml

确认效果
kubectl get resourcequotas
kubectl describe resourcequotas
```

```
root@master1:~/admission# kubectl get resourcequotas
NAME                AGE   REQUEST

                          LIMIT
resourcequota-test   5s    count/configmaps: 2/5, count/cronjobs.batch: 0/2, count/deployments.apps: 0/2, count/secrets: 3/5, count
/services: 1/5, count/statefulsets.apps: 0/2, persistentvolumeclaims: 3/6, pods: 0/5, requests.cpu: 0/2, requests.memory: 0/4Gi, re
quests.storage: 6Gi/20Gi   limits.cpu: 0/4, limits.memory: 0/8Gi
root@master1:~/admission# kubectl describe resourcequotas resourcequota-test
Name:                   resourcequota-test
Namespace:              default
Resource                Used  Hard
--------                ----  ----
count/configmaps        2     5
count/cronjobs.batch    0     2
count/deployments.apps  0     2
count/secrets           3     5
count/services          1     5
count/statefulsets.apps 0     2
limits.cpu              0     4
limits.memory           0     8Gi
persistentvolumeclaims  3     6
pods                    0     5
requests.cpu            0     2
requests.memory         0     4Gi
requests.storage        6Gi   20Gi
```

尝试创建多个pod,确认一下，pod数量多于5个的时候，看看是否会限制pod的创建

- 安全策略

podsecuritpolicy实践

启用psp功能

```
kubectl -n kube-system exec -it kube-apiserver-master1 -- kube-apiserver --help
| grep 'enable-admission'
```

```
root@master1:~/admission# kubectl -n kube-system exec -it kube-apiserver-master1 -- kube-apiserver --help | grep 'enable-admission'
      --admission-control strings              Admission is divided into two phases. In the first phase, only mutating admission pl
ugins run. In the second phase, only validating admission plugins run. The names in the below list may represent a validating plugi
n, a mutating plugin, or both. The order of plugins in which they are passed to this flag does not matter. Comma-delimited list of:
 AlwaysAdmit, AlwaysDeny, AlwaysPullImages, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressC
lass, DefaultStorageClass, DefaultTolerationSeconds, DenyServiceExternalIPs, EventRateLimit, ExtendedResourceToleration, ImagePolic
yWebhook, LimitPodHardAntiAffinityTopology, LimitRanger, MutatingAdmissionWebhook, NamespaceAutoProvision, NamespaceExists, Namespa
ceLifecycle, NodeRestriction, OwnerReferencesPermissionEnforcement, PersistentVolumeClaimResize, PersistentVolumeLabel, PodNodeSele
ctor, PodSecurity, PodSecurityPolicy, PodTolerationRestriction, Priority, ResourceQuota, RuntimeClass, SecurityContextDeny, Service
Account, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionWebhook. (DEPRECATED: Use --enable-admission-plugi
ns or --disable-admission-plugins instead. Will be removed in a future version.)
      --enable-admission-plugins strings       admission plugins that should be enabled in addition to default enabled ones (Namesp
aceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageCl
ass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateS
ubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota). Comma-delimited list
of admission plugins: AlwaysAdmit, AlwaysDeny, AlwaysPullImages, CertificateApproval, CertificateSigning, CertificateSubjectRestric
tion, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, DenyServiceExternalIPs, EventRateLimit, ExtendedResourceT
oleration, ImagePolicyWebhook, LimitPodHardAntiAffinityTopology, LimitRanger, MutatingAdmissionWebhook, NamespaceAutoProvision, Nam
espaceExists, NamespaceLifecycle, NodeRestriction, OwnerReferencesPermissionEnforcement, PersistentVolumeClaimResize, PersistentVol
umeLabel, PodNodeSelector, PodSecurity, PodSecurityPolicy, PodTolerationRestriction, Priority, ResourceQuota, RuntimeClass, Securit
yContextDeny, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionWebhook. The order of plugins
 in this flag does not matter.
```

注意：

由于默认psp是拒绝所有pod的，所以我们在启用psp的时候，需要额外做一些措施 -- 即提前做好psp相关的策略，然后再开启PSP功能。

Policy本身并不会产生实际作用，需要将其与用户或者serviceaccount绑定才可以完成授权。所以PSP的基本的操作步骤是：

1 定义psp相关策略

2 绑定psp资源的角色

3 集群启用PSP功能

如果要在生产环境中使用，必须要提前测试一下，否则不推荐使用，因为它的门槛较多。

1 定义psp相关策略

定制资源配置文件

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: privileged
  annotations:
```

```
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  allowedUnsafeSysctls:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 0
    max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  runAsGroup:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'

---
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default'
    apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
    seccomp.security.alpha.kubernetes.io/defaultProfileName:  'docker/default'
    apparmor.security.beta.kubernetes.io/defaultProfileName:  'runtime/default'
spec:
  privileged: false
  allowPrivilegeEscalation: false
  allowedUnsafeSysctls: []
  requiredDropCapabilities:
    - ALL
  # Allow core volume types.
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    - 'persistentVolumeClaim'
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    rule: 'MustRunAsNonRoot'
  seLinux:
    rule: 'RunAsAny'
```

```
  supplementalGroups:
    rule: 'MustRunAs'
    ranges:
      # Forbid adding the root group.
      - min: 1
        max: 65535
  fsGroup:
    rule: 'MustRunAs'
    ranges:
      # Forbid adding the root group.
      - min: 1
        max: 65535
  readOnlyRootFilesystem: false

应用资源定义文件
kubectl apply -f 12-security-podsecuritypolicy-base.yaml

确认效果
kubectl get podsecuritypolicies.policy
```

2 绑定psp资源的角色

```
定义资源配置文件
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: psp-restricted
rules:
- apiGroups: ['policy']
  resources: ['podsecuritypolicies']
  verbs:     ['use']
  resourceNames:
  - restricted
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: psp-privileged
rules:
- apiGroups: ['policy']
  resources: ['podsecuritypolicies']
  verbs:     ['use']
  resourceNames:
  - privileged
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: privileged-psp-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: psp-privileged
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:masters
```

```
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:node
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts:kube-system
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: restricted-psp-user
roleRef:
  kind: ClusterRole
  name: psp-restricted
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:authenticated

应用资源定义文件
kubectl apply -f 13-security-podsecuritypolicy-binding.yaml

确认效果
root@master1:~/admission# kubectl get clusterrole | grep psp
psp-privileged                          2021-10-07T06:10:41Z
psp-restricted                          2021-10-07T06:10:41Z
root@master1:~/admission# kubectl get clusterrolebindings | grep psp
privileged-psp-user             ClusterRole/psp-privileged       56s
restricted-psp-user             ClusterRole/psp-restricted       56s
```

3 集群启用PSP功能

```
配置apiserver增加admission plugin PodSecurityPolicy即可。
编辑 /etc/kubernetes/manifests/kube-apiserver.yaml 文件，添加如下配置
    --enable-admission-plugins=NodeRestriction,PodSecurityPolicy

由于kubeadm集群中，api-server是以静态pod的方式来进行管控的，所以我们不用重启，稍等一会，环境自然就开启了PSP功能
```

4 测试效果

```
设置资源配置文件 14-security-pod-test.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: 10.0.0.19:80/mykubernetes/nginx:1.21.3
```

小结