讲师：李振良（阿良）

今天课题：《K8s集群强化》

学院官网：www.aliangedu.cn

阿良个人微信

DevOps技术栈公众号

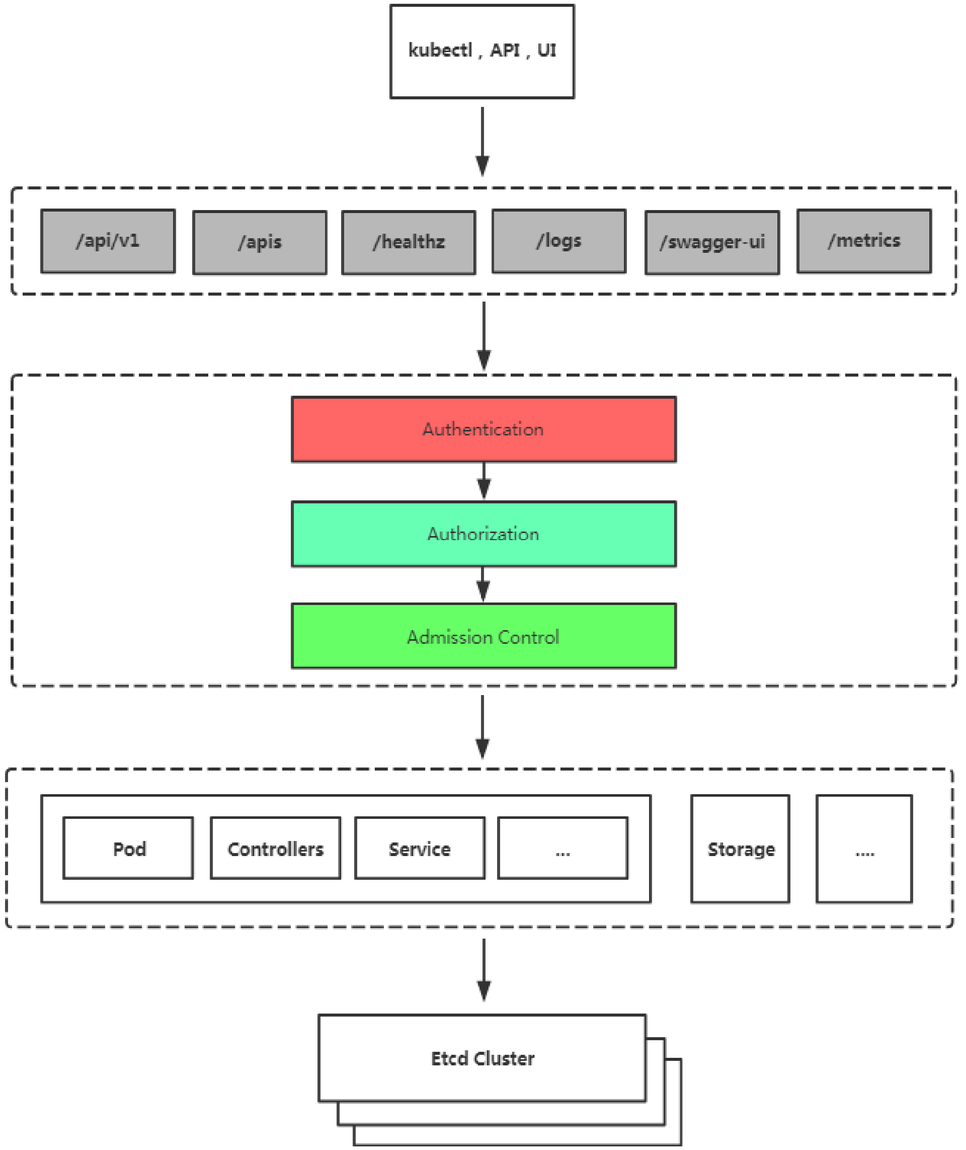
第二章 集群强化

- K8s安全框架
- RBAC认证授权案例
- 资源配额 ResourceQuota
- 资源限制 LimitRange

Kubernetes 安全框架

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（鉴权）
2. Authorization（授权）
3. Admission Control（准入控制）



Kubernetes 安全框架：鉴权（Authentication）

K8s Apiserver提供三种客户端身份认证：

- HTTPS 证书认证：基于CA证书签名的数字证书认证（kubeconfig）
- HTTP Token认证：通过一个Token来识别用户（serviceaccount）
- HTTP Base认证：用户名+密码的方式认证（1.19版本弃用）

Kubernetes 安全框架：授权（Authorization）

RBAC（Role-Based Access Control，基于角色的访问控制）：负责完成授权（Authorization）工作。

RBAC根据API请求属性，决定允许还是拒绝。

比较常见的授权维度：

- user：用户名
- group：用户分组
- 资源，例如pod、deployment
- 资源操作方法：get，list，create，update，patch，watch，delete
- 命名空间
- API组

Kubernetes 安全框架：准入控制（Admission Control）

Adminssion Control实际上是一个准入控制器插件列表，发送到API Server的请求都需要经过这个列表中的每个准入控制器插件的检查，检查不通过，则拒绝请求。

启用一个准入控制器：

```
kube-apiserver --enable-admission-plugins=NamespaceLifecycle,LimitRanger ...
```

关闭一个准入控制器：

```
kube-apiserver --disable-admission-plugins=PodNodeSelector,AlwaysDeny ...
```

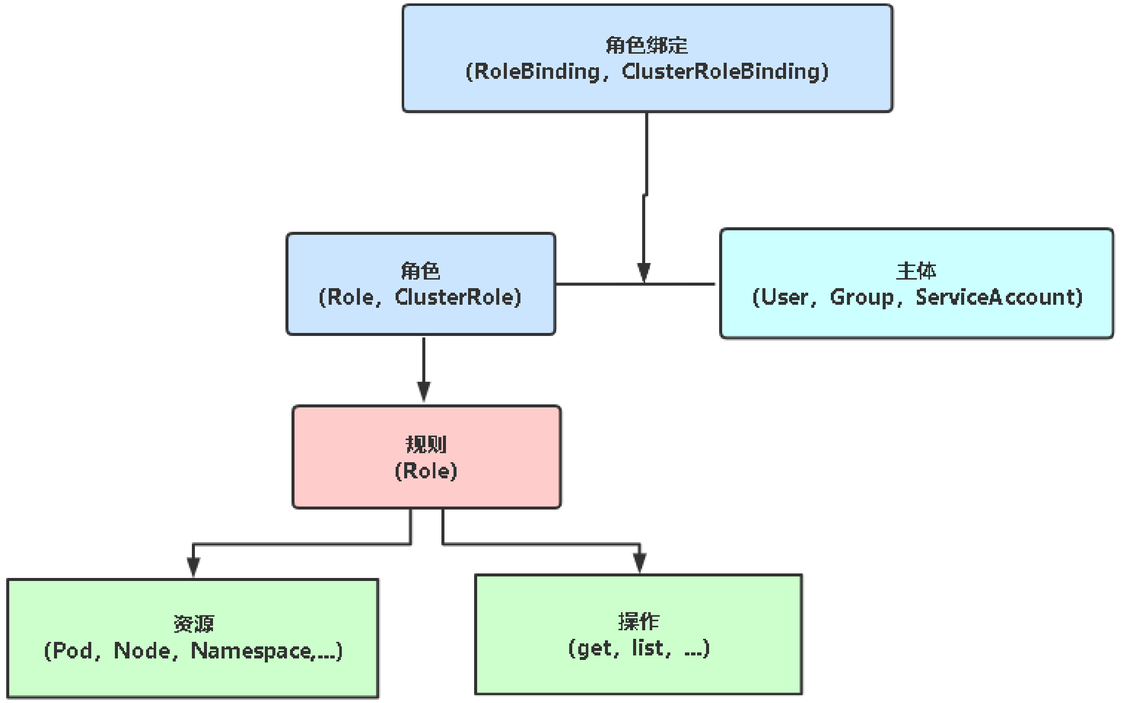
查看默认启用：

```
kubectl exec kube-apiserver-k8s-master -n kube-system -- kube-apiserver -h | grep enable-admission-plugins
```

基于角色的权限访问控制：RBAC

RBAC（Role-Based Access Control，基于角色的访问控制），

是K8s默认授权策略，并且是动态配置策略（修改即时生效）。



主体（subject）

- User：用户
- Group：用户组
- ServiceAccount：服务账号

角色

- Role：授权特定命名空间的访问权限
- ClusterRole：授权所有命名空间的访问权限

角色绑定

- RoleBinding：将角色绑定到主体（即subject）
- ClusterRoleBinding：将集群角色绑定到主体

注：RoleBinding在指定命名空间中执行授权，ClusterRoleBinding在集群范围执行授权。

基于角色的权限访问控制：RBAC

k8s预定好了四个集群角色供用户使用，使用kubectl get clusterrole查看，其中systemd:开头的为系统内部使用。

内置集群角色	描述
cluster-admin	超级管理员，对集群所有权限
admin	主要用于授权命名空间所有读写权限
edit	允许对命名空间大多数对象读写操作，不允许查看或者修改角色、角色绑定。
view	允许对命名空间大多数对象只读权限，不允许查看角色、角色绑定和Secret

案例1：对用户授权访问K8s（TLS证书）

需求：为指定用户授权访问不同命名空间权限，例如新入职一个小弟，希望让他先熟悉K8s集群，为了安全性，先不能给他太大权限，因此先给他授权访问default命名空间Pod读取权限。

实施大致步骤：

1. 用K8S CA签发客户端证书
2. 生成kubeconfig授权文件
3. 创建RBAC权限策略
4. 指定kubeconfig文件测试权限：kubectl get pods --kubeconfig=./aliang.kubeconfig

案例1：对用户授权访问K8s（TLS证书）

角色权限分配：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [“”] # api组，例如apps组，空值表示是核心API组，像namespace、pod、service、pv、pvc都在里面
  resources: [“pods”] #资源名称（复数），例如pods、deployments、services
  verbs: [“get”, “watch”, “list”] # 资源操作方法
```

将主体与角色绑定：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User  # 主体
  name: jane  # 主体名称
  apiGroup: rbac.authorization.k8s.io
roleRef:  # 绑定的角色
  kind: Role
  name: pod-reader  # 角色名称
  apiGroup: rbac.authorization.k8s.io
```

案例1：对用户授权访问K8s（TLS证书）

用户组：用户组的好处是无需单独为某个用户创建权限，统一为这个组名进行授权，所有的用户都以组的身份访问资源。

例如：为dev用户组统一授权

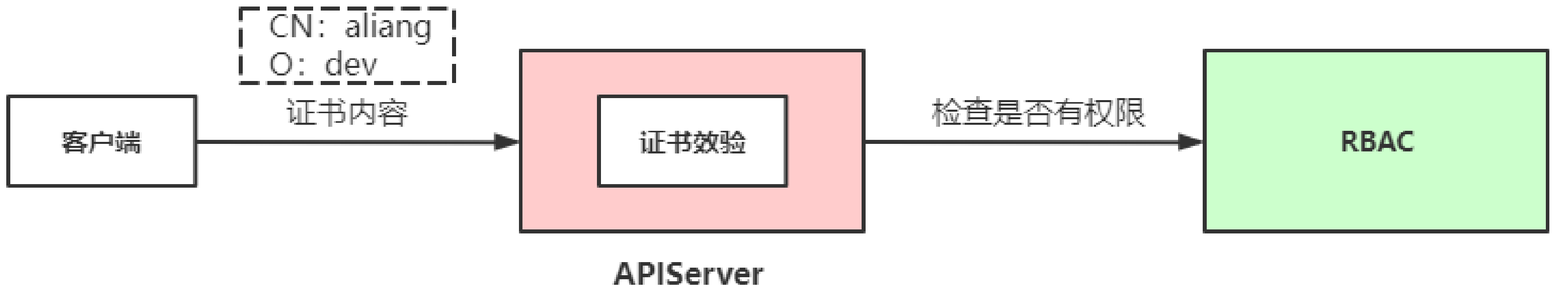
1、将certs.sh文件中的aliang-csr.json下的O字段改为dev，并重新生成证书和kubeconfig文件

2、将dev用户组绑定Role（pod-reader）

3、测试，只要O字段都是dev，这些用户持有的kubeconfig文件都拥有相同的权限

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: Group
  name: dev
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

案例1：对用户授权访问K8s（TLS证书）



TLS证书认证流程

案例2：对应用程序授权访问K8s（ServiceAccount）

先了解下ServiceAccount，简称SA，是一种用于让程序访问K8s API的服务账号。

- 当创建namespace时，会自动创建一个名为default的SA，这个SA没有绑定任何权限
- 当default SA创建时，会自动创建一个default-token-xxx的secret，并自动关联到SA
- 当创建Pod时，如果没有指定SA，会自动为pod以volume方式挂载这个default SA，在容器目录：/var/run/secrets/kubernetes.io/serviceaccount

验证默认SA权限：kubectl --as=system:serviceaccount:default:default get pods

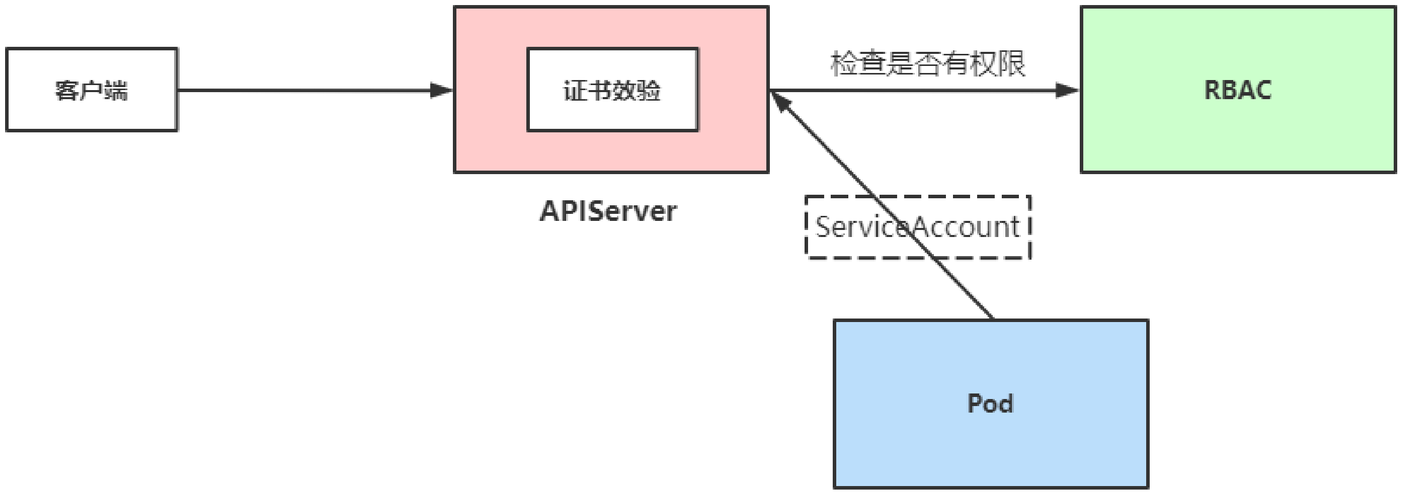
案例2：对应用程序授权访问K8s（ServiceAccount）

需求：授权容器中Python程序对K8s API访问权限

实施大致步骤：

1. 创建Role
2. 创建ServiceAccount
3. 将ServiceAccount与Role绑定
4. 为Pod指定自定义的SA
5. 进入容器里执行Python程序测试操作K8s API权限

案例2：对应用程序授权访问K8s（ServiceAccount）



ServiceAccount认证流程

案例2：对应用程序授权访问K8s（ServiceAccount）

命令行使用：授权SA只能查看test命名空间控制器的权限

```
# 创建角色
kubectl create role role-test --verb=get,list \
--resource=deployments,daemonsets,statefulsets -n test
# 创建服务账号
kubectl create serviceaccount app-demo -n test
# 将服务账号绑定角色
kubectl create rolebinding role-test:app-demo \
--serviceaccount=test:app-demo --role=role-test -n test
# 测试
kubectl --as=system:serviceaccount:test:app-demo \
get pods -n test
```

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app-demo
  namespace: test
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: role-test
  namespace: test
rules:
- apiGroups: ["apps"]
  resources: ["deployments","daemonsets","statefulsets"]
  verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: role-test:app-demo
  namespace: test
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: role-test
subjects:
- kind: ServiceAccount
  name: app-demo
  namespace: test
```

命令对应YAML

资源配额 ResourceQuota

当多个团队、多个用户共享使用K8s集群时，会出现不均匀资源使用，默认情况下先到先得，这时可以通过ResourceQuota来对命名空间资源使用总量做限制，从而解决这个问题。

使用流程：k8s管理员为每个命名空间创建一个或多个ResourceQuota对象，定义资源使用总量，K8s会跟踪命名空间资源使用情况，当超过定义的资源配额会返回拒绝。

资源配额 ResourceQuota

ResourceQuota功能是一个准入控制插件，默认已经启用。

支持的资源	描述
limits.cpu/memory	所有Pod上限资源配置总量不超过该值（所有非终止状态的Pod）
requests.cpu/memory	所有Pod请求资源配置总量不超过该值（所有非终止状态的Pod）
cpu/memory	等同于requests.cpu/requests.memory
requests.storage	所有PVC请求容量总和不超过该值
persistentvolumeclaims	所有PVC数量总和不超过该值
<storage-class-name>.storageclass.storage.k8s.io/requests.storage	所有与<storage-class-name>相关的PVC请求容量总和不超过该值
<storage-class-name>.storageclass.storage.k8s.io/persistentvolumeclaims	所有与<storage-class-name>相关的PVC数量总和不超过该值
pods、count/deployments.apps、count/statfulsets.apps、count/services（services.loadbalancers、services.nodeports）、count/secrets、count/configmaps、count/job.batch、count/cronjobs.batch	创建资源数量不超过该值

资源配额 ResourceQuota

计算资源配额：

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
  namespace: test
spec:
  hard:
    requests.cpu: "4"
    requests.memory: 10Gi
    limits.cpu: "6"
    limits.memory: 12Gi
```

存储资源配额：

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage-resources
  namespace: test
spec:
  hard:
    requests.storage: "10G"
    managed-nfs-
storage.storageclass.storage.k8s.io/requ
ests.storage: "5G"
```

对象数量配额：

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: object-counts
  namespace: test
spec:
  hard:
    pods: "10"
    count/deployments.apps: "3"
    count/services: "3"
```

查看配额：

```
kubectl get quota -n test
```

资源限制 LimitRange

默认情况下，K8s集群上的容器对计算资源没有任何限制，可能会导致个别容器资源过大导致影响其他容器正常工作，这时可以使用LimitRange定义容器默认CPU和内存请求值或者最大上限。

LimitRange限制维度：

- 限制容器配置requests.cpu/memory，limits.cpu/memory的最小、最大值
- 限制容器配置requests.cpu/memory，limits.cpu/memory的默认值
- 限制PVC配置requests.storage的最小、最大值

资源限制 LimitRange

计算资源最大、最小限制：

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-memory-min-max
  namespace: test
spec:
  limits:
  - max:   # 容器能设置limit的最大值
      cpu: 1
      memory: 1Gi
    min:  # 容器能设置request的最小值
      cpu: 200m
      memory: 200Mi
    type: Container
```

计算资源默认值限制：

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-memory-min-max
  namespace: test
spec:
  limits:
  - default:
      cpu: 500m
      memory: 500Mi
    defaultRequest:
      cpu: 300m
      memory: 300Mi
    type: Container
```

存储资源最大、最小限制：

```
apiVersion: v1
kind: LimitRange
metadata:
  name: storage-min-max
  namespace: test
spec:
  limits:
  - type: PersistentVolumeClaim
    max:
      storage: 10Gi
    min:
      storage: 1Gi
```

查看限制：

```
kubectl get limits -n test
kubectl describe limits -n test
```

课后作业

1、创建一个名为backend-sa的serviceaccount，授权只能查看default命名空间下pod，再创建一个deployment使用这个serviceaccount。

2、为default命名空间下创建的容器默认请求值（resources.requests）cpu=200m，memory=200Mi

谢谢

阿良个人微信

DevOps技术栈公众号

阿良教育：www.aliangedu.cn