01 K8s系统核心组件

1.1 Master和Node

> 官网：https://kubernetes.io/zh/docs/concepts/architecture/master-node-communication/

```
K8S集群中的控制节点，负责整个集群的管理和控制，可以做成高可用，防止一台Master打竞技或者不可用。
其中有一些关键的组件：比如API Server，Controller Manager，Scheduler等
```

> Node

```
Node会被Master分配一些工作负载，当某个Node不可用时，会将工作负载转移到其他Node节点上。
Node上有一些关键的进程：kubelet，kube-proxy，docker等
```

> 查看集群中的Node

```
kubectl get nodes
kubectl describe node node-name
```

1.2 kubeadm

1.2.1 kubeadm init

```
[root@master-kubeadm-k8s ~]# kubeadm init --kubernetes-version=1.14.0 --apiserver-advertise-addres
s=192.168.0.51 --pod-network-cidr=10.150.0.0/16
[init] Using Kubernetes version: v1.14.0
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flag
s.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [master-kubeadm-k8s localhost] and IPs [192
.168.0.51 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [master-kubeadm-k8s localhost] and IPs [1
92.168.0.51 127.0.0.1 ::1]
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [master-kubeadm-k8s kubernetes kubernetes.d
efault kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 192.168.0.5
1]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[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 dire
ctory "/etc/kubernetes/manifests". This can take up to 4m0s
[apiclient] All control plane components are healthy after 21.502688 seconds
[upload-config] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system"
Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.14" in namespace kube-system with the configurati
on for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --experimental-upload-certs
[mark-control-plane] Marking the node master-kubeadm-k8s as control-plane by adding the label "nod
e-role.kubernetes.io/master=''"
[mark-control-plane] Marking the node master-kubeadm-k8s as control-plane by adding the taints [no
de-role.kubernetes.io/master:NoSchedule]
[bootstrap-token] Using token: e0j0ci.d3fn5tx9wwske1e1
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for n
odes to get long term certificate credentials
[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve
CSRs from a Node Bootstrap Token
[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certific
ates in the cluster
[bootstrap-token] creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
```

```
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.0.51:6443 --token e0j0ci.d3fn5tx9wwske1e1 \
    --discovery-token-ca-cert-hash sha256:dfba74f6df5a403208b3d180911bc27eb8b08ad154ed8e2017e80121
7ffb29f2
[root@master-kubeadm-k8s ~]#
```

01-进行一系列检查[init之前的检查]，以确定这台机器可以部署kubernetes

kubeadm init pre-flight check:

(1)kubeadm版本与要安装的kubernetes版本的检查

(2)kubernetes安装的系统需求检查[centos版本、cgroup、docker等]

(3)用户、主机、端口、swap等

02-生成kubernetes对外提供服务所需要的各种证书可对应目录，也就是生成私钥和数字证书

/etc/kubernetes/pki/*

(1)自建ca，生成ca.key和ca.crt

(2)apiserver的私钥与公钥证书

(3)apiserver访问kubelet使用的客户端私钥与证书

(4)sa.key和sa.pub

(5)etcd相关私钥和数字证书

03-为其他组件生成访问kube-ApiServer所需的配置文件xxx.conf

```
ls /etc/kubernetes/
admin.conf  controller-manager.conf  kubelet.conf  scheduler.conf
```

【

```
(1)# 有了$HOME/.kube/config就可以使用kubectl和K8s集群打交道了，这个文件是来自于admin.config
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

(2)kubeconfig中包含了cluster、user和context信息:kubectl config view

(3)允许kubectl快速切换context，管理多集群

】

04-为master生成Pod配置文件，这些组件会被master节点上的kubelet读取到，并且创建对应资源

```
ls /etc/kubernetes/manifests/*.yaml
kube-apiserver.yaml
kube-controller-manager.yaml
kube-scheduler.yaml
```

【

这些pod由kubelet直接管理，是静态pod，直接使用主机网络

kubelet读取manifests目录并管理各控制平台组件pod的启动与停止

要想修改这些pod，直接修改manifests下的yaml文件即可

】

05-下载镜像[这里的镜像我们已经提前准备好了]，等待控制平面启动

k8s.gcr.io下载不了，所以我们先提前下载并且tag好了

06-一旦这些 YAML 文件出现在被 kubelet 监视的/etc/kubernetes/manifests/目录下，kubelet就会自动创建这些yaml文件定义的pod，即master组件的容器。master容器启动后，kubeadm会通过检查localhost:6443/healthz这个master组件的健康状态检查URL，等待master组件完全运行起来

【 cat kube-apiserver.yaml里面有健康检查的配置】

07-为集群生成一个bootstrap token，设定当前node为master，master节点将不承担工作负载

08-将ca.crt等 Master节点的重要信息，通过ConfigMap的方式保存在etcd中，工后续部署node节点使用

09-安装默认插件，kubernetes默认kube-proxy和DNS两个插件是必须安装的，dns插件安装了会出于pending状态，要等网络插件安装完成，比如calico

【
kubectl get daemonset -n kube-system
可以看到kube-proxy和calico[或者其他网络插件]
】

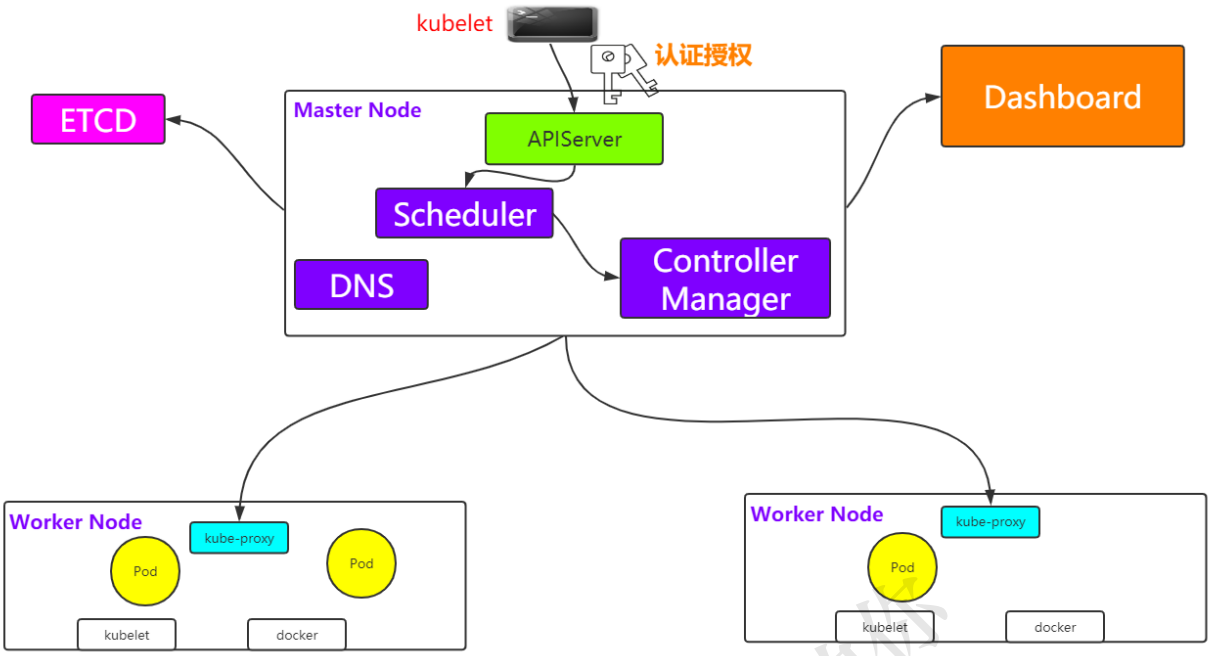
1.2.2 kubeadm join

kubeadm join 192.168.0.51:6443 --token yu1ak0.2dcecvmpozsy8loh \ --discovery-token-ca-cert-hash sha256:5c4a69b3bb05b81b675db5559b0e4d7972f1d0a61195f217161522f464c307b0

```
01 join前检查

02 discovery-token-ca-cert-hash用于验证master身份
    # 可以计算出来，在w节点上执行
    openssl x509 -in /etc/kubernetes/pki/ca.crt -noout -pubkey | openssl rsa -pubin -
outform DER 2>/dev/null | sha256sum | cut -d' ' -f1
    # 最终hash的值
    909adc03d6e4bd676cfc4e04b864530dd19927d8ed9d84e224101ef38ed0bb96

03 token用于master验证node
    # 在master上节点上，可以查看对应的token
    kubectl get secret -n kube-system | grep bootstrap-token
    # 得到token的值
    kubectl get secret/bootstrap-token-kggzhc -n kube-system -o yaml
    # 对token的值进行解码
    echo NHRzZHpOY2RidDRmd2U5dw==|base64 -d    --->4tsdztcdbt4fwe9w
    # 最终token的值
    kggzhc.4tsdztcdbt4fwe9w

04 实在忘了怎么办?
    https://gper.club/articles/7e7e7f7ff7g5bgc0g68  ---> 07
```

1.3 先把核心组件总体过一遍

不妨查看一下之前的K8s架构图，勾起回忆



对之前理解的优化，先是整体

> **01 kubectl**
>
> 总得要有一个操作集群的客户端，也就是和集群打交道

> **02 kube-apiserver**
>
> 整个集群的中枢纽带，负责的事情很多

```
(1)/etc/kubernetes/manifests/kube-apiserver.yaml    # kubelet管理的静态pod
(2)--insecure-port=0    # 默认使用http非安全协议访问
(3)安全验证的一些文件
(4)准入策略的拦截器
(5)--authorization-mode=Node,RBAC
(6)--etcd     # 配置apiserver与etcd通信
```

> **03 kube-scheduler**
>
> 单纯地调度pod，按照特定的调度算法和策略，将待调度Pod绑定到集群中某个适合的Node，并写入绑定信息，由对应节点的kubelet服务创建pod。

```
(1)/etc/kubernetes/manifests/kube-scheduler.yaml    # kubelet管理的静态pod
(2)--address表示只在master节点上提供服务，不对外
(3)kubeconfig表示
```

> **04 kube-controller-manager**
>
> 负责集群中Node、Pod副本、服务的endpoint、命名空间、Service Account、资源配合等管理
>
> 会划分成不同类型的controller，每个controller都是一个死循环，在循环中controller通过apiserver监视自己控制资源的状态，一旦状态发生变化就会努力改变状态，直到变成期望状态

```
(1)/etc/kubernetes/manifests/kube-controller-manager.yaml   # kubelet管理的静态pod
(2)参数设置ca-file
(3)多个manager，是否需要进行leader选举
```

> **05 kubelet** 集群中的所有节点都有运行，用于管理pod和container，每个kubelet会向apiserver注册本节点的信息，并向master节点上报本节点资源使用的情况

```
(1)kubelet由操作系统init[systemd]进行启动
(2)ls /lib/systemd/system/kubelet.service
(3)systemctl daemon-reload & systemctl restart kubelet
```

> **06 kube-proxy**
>
> 集群中的所有节点都有运行，像service的操作都是由kube-proxy代理的，对于客户端是透明的

```
(1)kube-proxy由daemonset控制器在各个节点上启动唯一实例
(2)配置参数: /var/lib/kube-proxy/config.conf(pod内)  # 不是静态pod
(3)kubectl get pods -n kube-system
(4)kubectl exec kube-proxy-jt9n4 -n kube-system -- cat /var/lib/kube-proxy/config.conf
(5)mode:""   --->iptables
```

> **07 DNS**
>
> 域名解析的问题

> **08 dashboard**
>
> 需要有监控面板能够监测整个集群的状态

> **09 etcd**
>
> 整个集群的配置中心，所有集群的状态数据，对象数据都存储在etcd中
>
> kubeadm引导启动的K8s集群，默认只启动一个etcd节点

```
(1)/etc/kubernetes/manifests/etcd.yaml   # kubelet管理的静态pod
(2)etcd所使用的相关秘钥在/etc/kubernetes/pki/etcd里面
(3)etcd挂载master节点本地路径/var/lib/etcd用于运行时数据存储，tree
```

1.4 Kubernetes源码查看方式

> 源码地址：https://github.com/kubernetes/kubernetes
>
> https://github.com/kubernetes/kubernetes/tree/release-1.14

1.5 kubectl

> 语法：kubectl [command] [TYPE] [NAME] [flag]
>
> 官网：https://kubernetes.io/docs/reference/kubectl/overview/

```
command:用于操作k8s集资源对象的命令，比如apply、delete、describe、get等
TYPE:要操作资源对象的类型，区分大小写，比如pod[pods/po]、deployment
NAME:要操作对象的具体名称，若不指定，则返回该资源类型的全部对象[是默认命名空间下的]
flags:可选
```

demo

- 查看集群信息

```
# 查看集群的信息
kubectl config view

# 查看cluster的信息
kubectl config get-clusters
```

- 创建资源

```
kubectl apply -f xxx.yaml
kubectl apply -f <directory>
```

- 查看资源对象

```
# 查看Pod
kubectl get pods
# 查看Service
kubectl get svc
```

- 问题查看调试

```
kubectl describe pod pod-name
kubectl exec -it pod-name
kubectl logs -f pod-name
kubectl attach
```


1.6 API Server

> 官网：https://kubernetes.io/zh/docs/reference/command-line-tools-reference/kube-apiserver/
>
> APIServer提供了K8S各类资源对象的操作，是集群内各个功能模块之间数据交互和通信的中心枢纽，是整个系统的数据总线和数据中心。通常我们通过kubectl与APIServer进行交互。
>
> APIServer通过**kube-apiserver**的进程提供服务，运行在master节点上
>
> kubectl与APIServer之间是REST调用
>
> ```
> The Kubernetes API server validates and configures data for the api objects which include pods, services, replicationcontrollers, and others. The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.
> ```

(1)查看yaml文件中的apiVersion

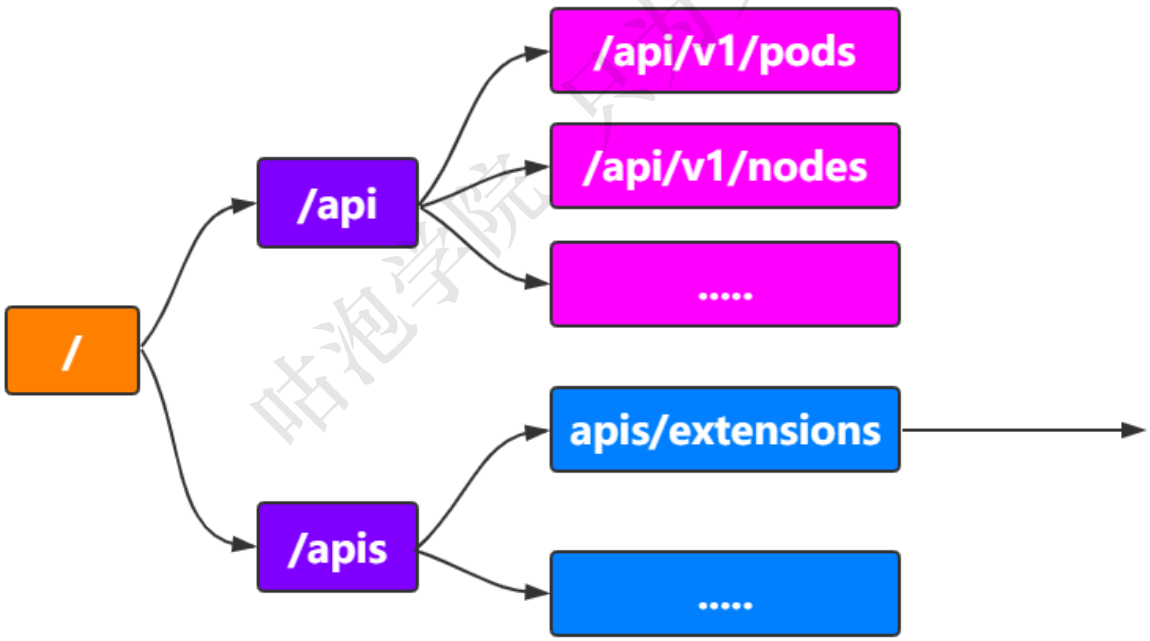
```
grep -r "apiVersion" .
```

```
./pod_nginx.yaml:apiVersion: apps/v1
./my-tomcat.yaml:apiVersion: apps/v1
./my-tomcat.yaml:apiVersion: v1
./mandatory.yaml:apiVersion: v1
./mandatory.yaml:apiVersion: v1
./mandatory.yaml:apiVersion: v1
./mandatory.yaml:apiVersion: v1
./mandatory.yaml:apiVersion: v1
./mandatory.yaml:apiVersion: rbac.authorization.k8s.io/v1beta1
./mandatory.yaml:apiVersion: rbac.authorization.k8s.io/v1beta1
./mandatory.yaml:apiVersion: rbac.authorization.k8s.io/v1beta1
...
```

(2)REST API设计

`api官网`：https://kubernetes.io/docs/concepts/overview/kubernetes-api/

`v1.14`：https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.14/



(3)想要写Pod的yaml文件

```
https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.14/#pod-v1-core
```

(4)kube-apiserver

```
lsof -i tcp:8080
vi /etc/kubernetes/manifests/kube-apiserver.yaml    [kubeadm安装方式]
# 查询insecure-port，并将修改端口为8080
insecure-port=8080
# kubect apply生效，需要等待一会
kubectl apply -f kube-apiserver.yaml
```

> (5)查看端口以及访问测试
>
> `可以发现结果和kubectl使用一样`

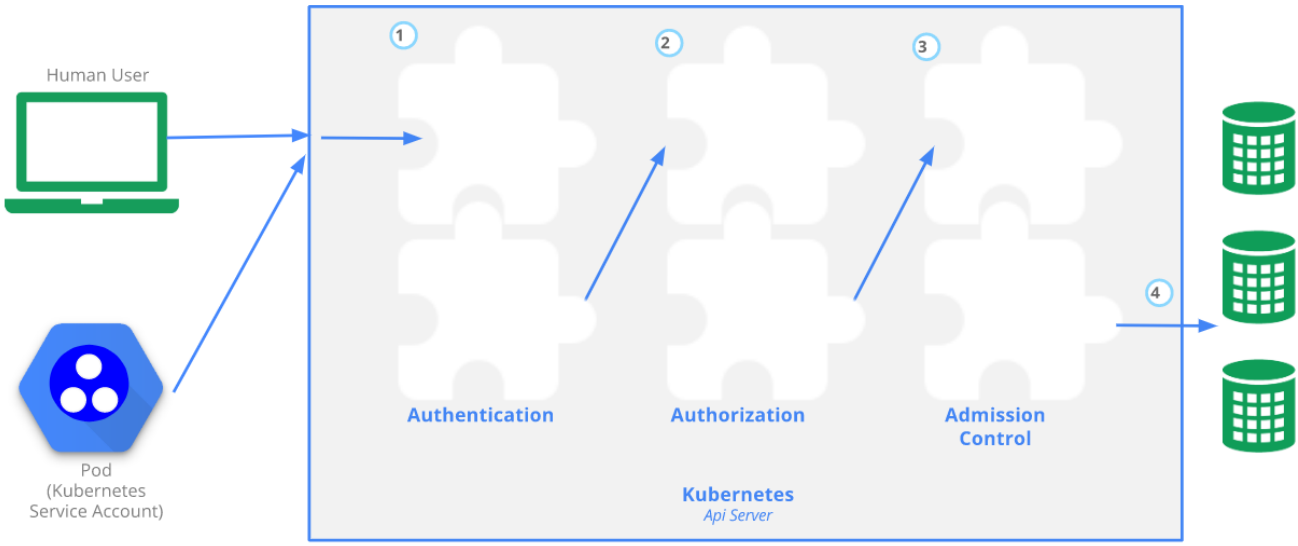
```
lsof -i tcp:8080
curl localhost:8080
curl localhost:8080/api
curl localhost:8080/api/v1
curl localhost:8080/api/v1/pods
curl localhost:8080/api/v1/services
```

> (6)设计一个Pod的url请求
>
> https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.14/#-strong-write-operations-pod-v1-core-strong-

> (7)这种操作还是相对比较麻烦的，哪怕使用kubectl，怎么办？好事之者
>
> https://github.com/kubernetes-client
>
> `Java`：https://github.com/kubernetes-client/java
>
> `Go`：https://github.com/kubernetes/client-go

1.7 集群安全机制之API Server

> `官网`：https://v1-12.docs.kubernetes.io/docs/reference/access-authn-authz/controlling-access/
>
> 对于k8s集群的访问操作，都是通过api server的rest api来实现的，难道所有的操作都允许吗？当然不行，这里就涉及到认证、授权和准入等操作。



1.7.1 API Server认证(Authentication)

> 官网：
>
> https://v1-12.docs.kubernetes.io/docs/reference/access-authn-authz/controlling-access/#authentication
>
> **说白了，就是如何来识别客户端的身份，K8s集群提供了3种识别客户端身份的方式**

- HTTPS证书认证

基于CA根证书签名的双向数字证书认证方式

- HTTP Token认证

通过一个Token来识别合法用户

- HTTP Base认证

通过用户名+密码的方式认证

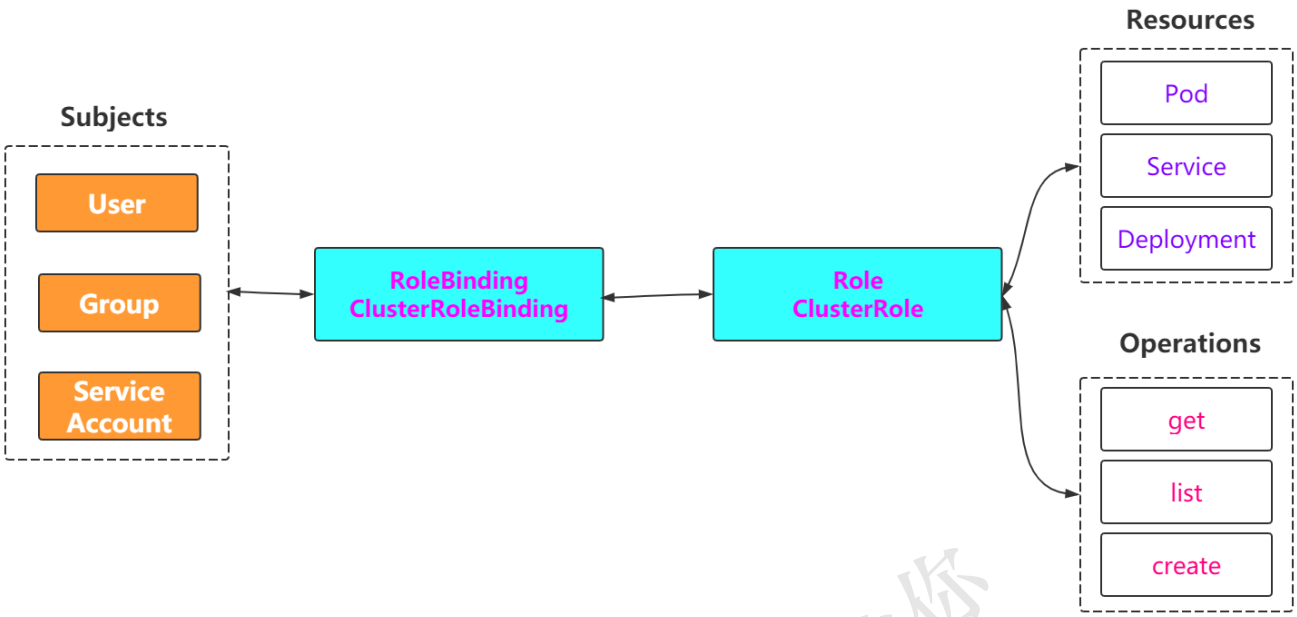
1.7.2 API Server授权(Authorization)

> 官网：
>
> https://v1-12.docs.kubernetes.io/docs/reference/access-authn-authz/controlling-access/#authorization

- ABAC授权模式
- Webhook授权模式
- RBRC授权模式

Role、ClusterRole、RoleBinding和ClusterRoleBinding

用户可以使用kubectl或者API调用等方式操作这些资源对象。



7.7.3 Admission Control(准入控制)

> 官网：
>
> https://v1-12.docs.kubernetes.io/docs/reference/access-authn-authz/controlling-access/#admission-control

- Always

允许所有请求

- AlwaysPullImages

在启动容器之前总是尝试重新下载镜像

- AlwaysDeny

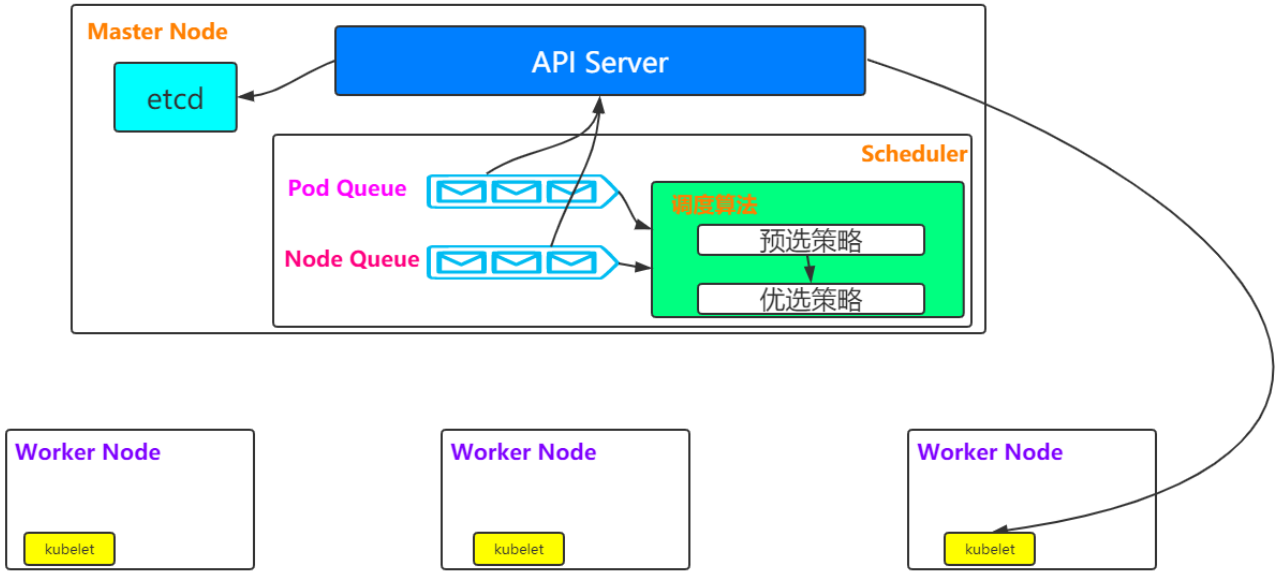
禁止所有请求

.....

1.8 Scheduler

> 官网：https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/
>
> ```
> In Kubernetes, scheduling refers to making sure that Pods are matched to Nodes so that Kubelet can run them.
> ```
>
> 通过调度算法，为待调度Pod列表的每个Pod，从Node列表中选择一个最合适的Node。
>
> 然后，目标节点上的kubelet通过API Server监听到Kubernetes Scheduler产生的Pod绑定事件，获取对应的Pod清单，下载Image镜像，并启动容器。

1.8.1 架构图



1.8.2 流程描述

> https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/#kube-scheduler-implementation
>
> ```
> 01-Filtering
> 02-Scoring
> ```

（1）预选调度策略：遍历所有目标Node，刷选出符合Pod要求的候选节点

（2）优选调度策略：在(1)的基础上，采用优选策略算法计算出每个候选节点的积分，积分最高者胜出

1.8.3 预选策略和优选策略

1.8.3.1 预选策略

> https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/#filtering

- PodFitsHostPorts

```
Checks if a Node has free ports (the network protocol kind) for the Pod ports the Pod is requesting.
```

- PodFitsHost

```
Checks if a Pod specifies a specific Node by it hostname.
```

- PodFitsResources

```
Checks if the Node has free resources (eg, CPU and Memory) to meet the requirement of the Pod.
```

1.8.3.2 优选策略

> https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/#scoring

- SelectorSpreadPriority

```
Spreads Pods across hosts, considering Pods that belonging to the same Service, StatefulSet
or ReplicaSet
```

- InterPodAffinityPriority

```
Computes a sum by iterating through the elements of weightedPodAffinityTerm and adding
“weight” to the sum if the corresponding PodAffinityTerm is satisfied for that node; the
node(s) with the highest sum are the most preferred.
```

1.8.4 实战

1.8.4.1 Node

> (1)正常创建pod，网盘/课堂源码/scheduler-node-origin.yaml

```
kubectl apply -f scheduler-node-origin.yaml
kubectl get pods
kubectl describe pod pod-name
```

> (2)使用网盘/课堂源码/scheduler-/scheduler-node.yaml
>
> 主要是体现node的调度

```
kubectl get nodes w1 -o yaml
找到labels
```

```
[root@master-kubeadm-k8s ~]# kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
scheduler-node-84845c99d4-fvw9d   0/1     Pending   0          7s
```

```
kubectl describe pod scheduler-node-84845c99d4-fvw9d
Events:
  Type     Reason            Age                From               Message
  ----     ------            ----               ----               -------
  Warning  FailedScheduling  37s (x2 over 37s)  default-scheduler  0/3 nodes are available:
3 node(s) didn't match node selector.
```

1.8.4.2 Pod

```
affinity:
       podAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
         - labelSelector:
             matchExpressions:
             - key: app
               operator: In
               values:
               - k8s
           topologyKey: kubernetes.io/hostname
```

1.9 kubelet

> 官网：https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/

在k8s集群中，每个Node上都会启动一个kubelet服务进程，用于处理master节点下发到本节点的任务。

管理Pod及Pod中的容器，每个kubelet进程会在API Server上注册节点自身信息，定期向Master节点汇报节点资源的使用情况，并通过cAdvisor监控容器和节点资源。

1.10 kube-proxy

> 官网：https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/

在k8s集群中，每个Node上都会运行一个kube-proxy进行，它是Service的透明代理兼负载均衡器，核心功能是将某个Service的访问请求转发到后端的多个Pod实例上。