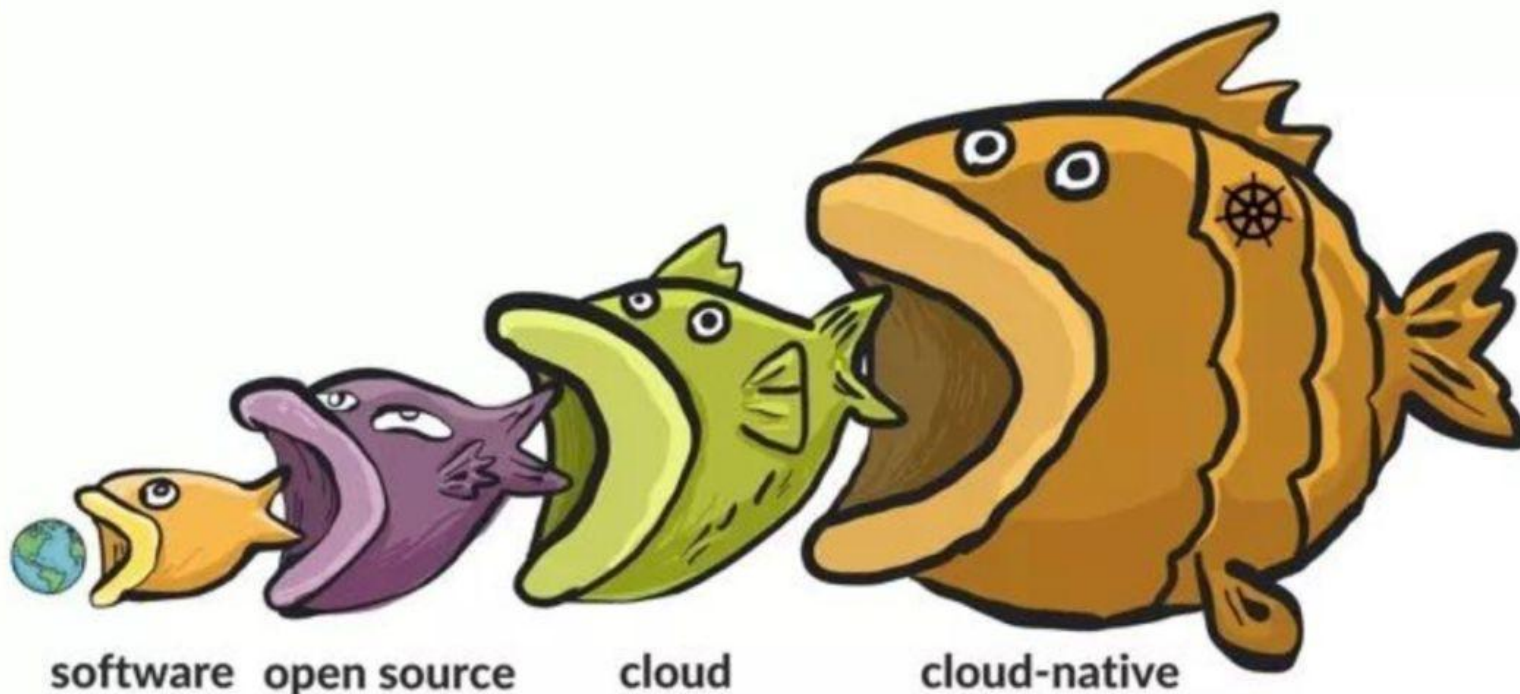


## 云原生简介:

- 2004年开始, Google已在内部大规模地使用容器技术。
- 2008年, Google将 Cgroups合并进入了Linux内核。
- 2013年, Docker项目正式发布。
- 2014年, Kubernetes项目正式发布。
- 2015年, 由Google、Redhat 以及微软等大型云计算厂商以及一些开源公司共同牵头成立了CNCF(Cloud Native Computing Foundation)云原生计算基金会。
- 2017年, CNCF达到170个成员和14个基金项目;
- 2018年, CNCF成立三周年有了195个成员, 19个基金会项目和11个孵化项目。

云原生定义:

<https://github.com/cncf/toc/blob/main/DEFINITION.md#%E4%B8%AD%E6%96%87%E7%89%88%E6%9C%AC>



## 云原生技术栈:

- 容器: 以docker为代表的容器运行技术。
- 服务网格: 比如Service Mesh等。
- 微服务: 在微服务体系结构中, 一个项目是由多个松耦合且可独立部署的较小组件或服务组成。
- 不可变基础设施: 不可变基础设施可以理解为一个应用运行所需要的基本运行需求, 不可变最基本的就是指运行服务的服务器在完成部署后, 就不在进行更改, 比如镜像等。
- 声明式API: 描述应用程序的运行状态, 并且由系统来决定如何来创建这个环境, 例如声明一个pod, 会有k8s执行创建并维持副本。

## 云原生特征:

- 符合 12 因素应用,12要素应用程序是一种构建应用程序的方法.
- 面向微服务架构.
- 自服务敏捷架构.
- 基于 API 的协作.
- 抗脆弱性

## 12 因素应用:

- 1、基准代码: 一份基准代码, 多份部署(用同一个代码库进行版本控制, 并可进行多次部署)。
- 2、依赖: 显式地声明和隔离相互之间的依赖。
- 3、配置: 在环境中存储配置。
- 4、后端服务: 把后端服务当作一种附加资源。
- 5、构建, 发布, 运行: 对程序执行构建或打包, 并严格分离构建和运行。
- 6、进程: 以一个或多个无状态进程运行应用。
- 7、端口绑定: 通过端口绑定提供服务。
- 8、并发: 通过进程模型进行扩展。
- 9、易处理: 快速地启动, 优雅地终止, 最大程度上保持健壮性。
- 10、开发环境与线上环境等价: 尽可能的保持开发, 预发布, 线上环境相同。
- 11、日志: 将所有运行中进程和后端服务的输出流按照时间顺序统一收集存储和展示。
- 12、管理进程: 一次性管理进程(数据备份等)应该和正常的常驻进程使用同样的运行环境。

云原生景

观图:

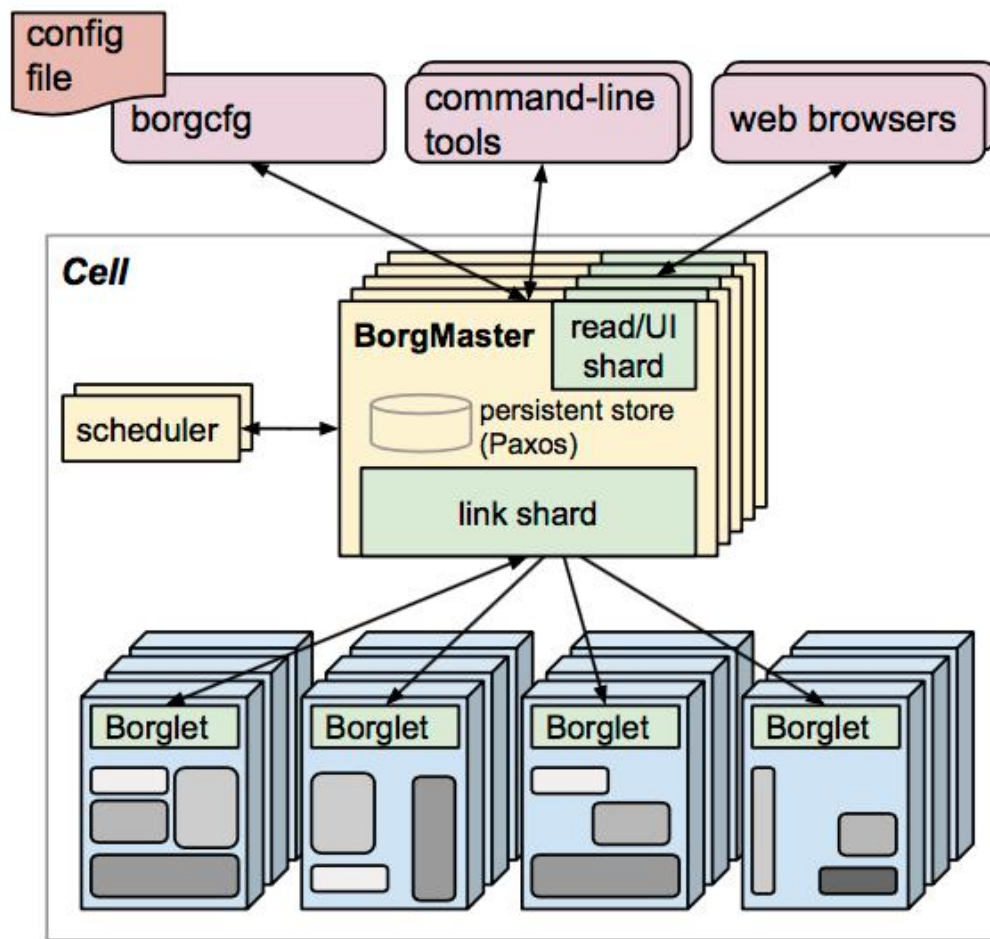
<https://landscape.cncf.io/>

## 云原生项目分类:



# kubernetes 简介:

- Kubernetes最初源于谷歌内部的Borg，Borg是谷歌内部的大规模集群管理系统，负责对谷歌内部很多核心服务的调度和管理,Borg的目的是让用户能够不必操心资源管理的问题，让他们专注于自己的核心业务，并且做到跨多个数据中心的资源利用率最大化。
- Borg主要由BorgMaster、Borglet、borgcfg和Scheduler组成:
- <https://kubernetes.io/zh/> #官网
- <https://github.com/kubernetes/kubernetes> #github





## kubernetes 组件简介:

### ➤ kube-apiserver:

<https://kubernetes.io/zh/docs/reference/command-line-tools-reference/kube-apiserver/>

Kubernetes API server 提供了k8s各类资源对象的增删改查及watch等HTTP Rest接口，这些对象包括pods、services、replicationcontrollers等，API Server为REST操作提供服务，并为集群的共享状态提供前端，所有其他组件都通过该前端进行交互。

### ➤ RESTful API:

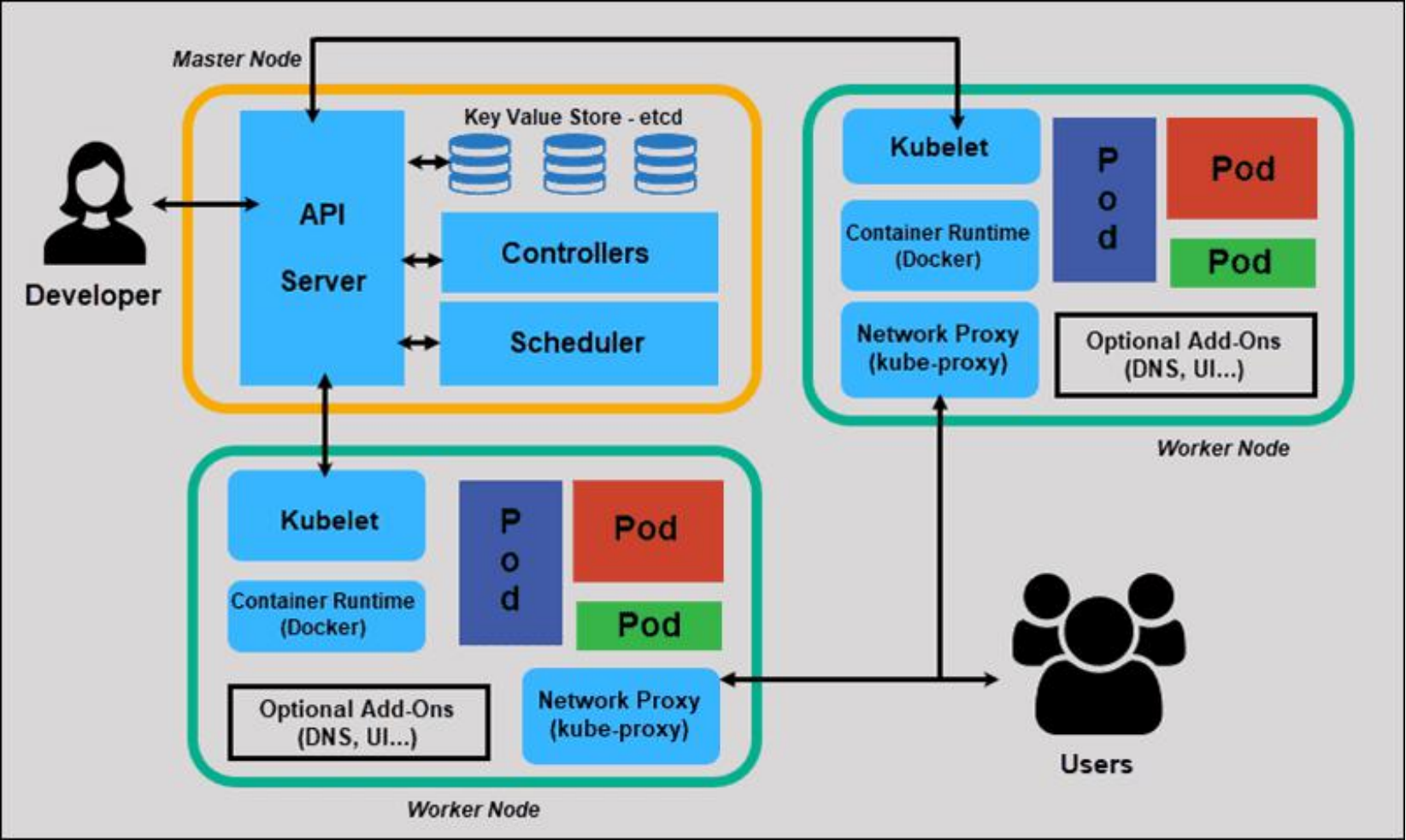
➤ 是REST风格的网络接口，REST描述的是在网络中client和server的一种交互形式

### ➤ REST:

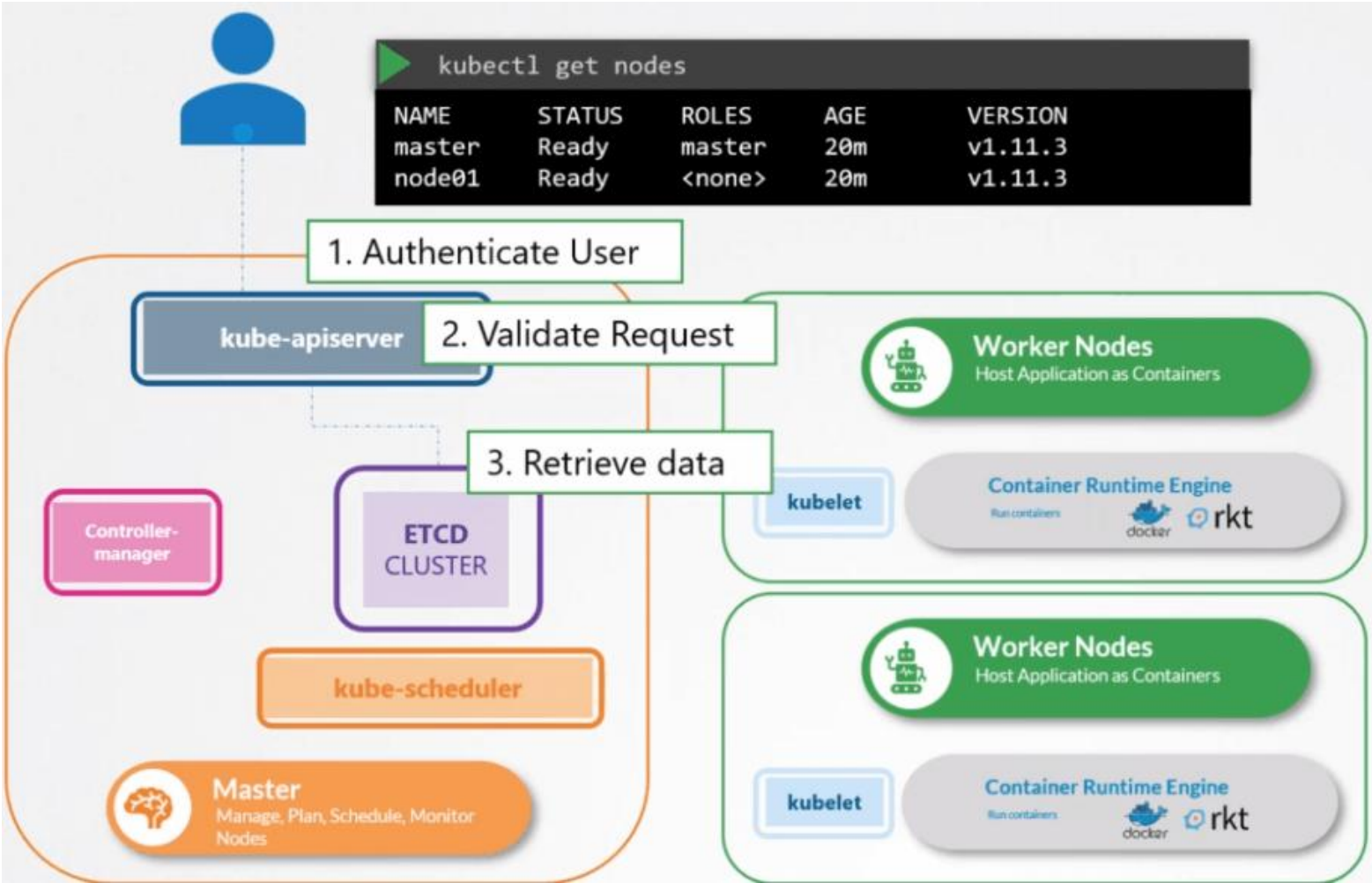
➤ 是一种软件架构风格，或者说是一种规范，其强调HTTP应当以资源为中心，并且规范了URI的风格，规范了HTTP请求动作(GET/PUT/POST/DELETE/HEAD/OPTIONS)的使用,具有对应的语义。

<https://github.com/Arachni/arachni/wiki/REST-API>

kubernetes 组件简介:



# kubernetes 简介:



## kubernetes API Server简介:

- 该端口默认值为6443，可通过启动参数 “`--secure-port`” 的值来修改默认值。
- 默认IP地址为非本地（Non-Localhost）网络端口，通过启动参数 “`--bind-address`” 设置该值。
- 该端口用于接收客户端、dashboard等外部HTTPS请求。
- 用于基于Token文件或客户端证书及HTTP Base的认证。
- 用于基于策略的授权。

## kubernetes API Server简介:

### ➤ kubernetes API测试:

- # curl --cacert /etc/kubernetes/ssl/ca.pem -H "Authorization: Bearer eyxxx https://127.0.0.1:6443
- # curl 127.0.0.1:6443/ #返回所有的API列表
- # curl 127.0.0.1:6443/apis #分组API
- # curl 127.0.0.1:6443/api/v1 #带具体版本号的API
- # curl 127.0.0.1:6443/version #API版本信息
- # curl 127.0.0.1:6443/healthz/etcd #与etcd的心跳监测
- # curl 127.0.0.1:6443/apis/autoscaling/v1 #API的详细信息
- # curl 127.0.0.1:6443/metrics #指标数据

# kubernetes 组件简介:

➤ kube-scheduler:

<https://kubernetes.io/zh/docs/reference/command-line-tools-reference/kube-scheduler/>

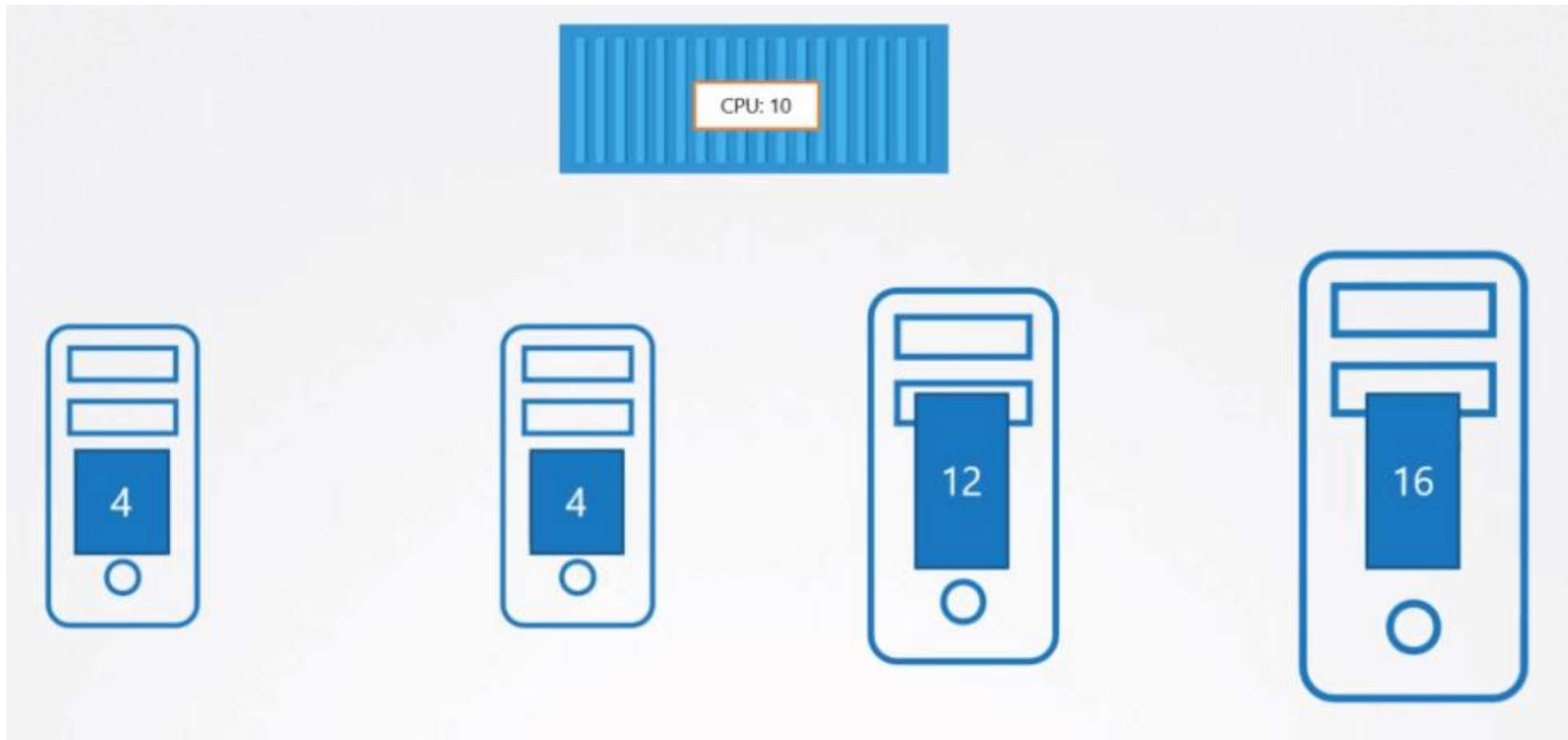
➤ Kubernetes 调度器是一个控制面进程，负责将 Pods 指派到节点上。

## kube-scheduler简介:

- 通过调度算法为待调度Pod列表的每个Pod从可用Node列表中选择一个最适合的Node，并将信息写入etcd中。
  - node节点上的kubelet通过API Server监听到kubernetes Scheduler产生的Pod绑定信息，然后获取对应的Pod清单，下载Image，并启动容器。
  - 策略:
    - LeastRequestedPriority
      - 优先从备选节点列表中选择资源消耗最小的节点（CPU+内存）。
    - CalculateNodeLabelPriority
      - 优先选择含有指定Label的节点。
    - BalancedResourceAllocation
      - 优先从备选节点列表中选择各项资源使用率最均衡的节点。
- 1.先排除不符合条件的节点
  - 2.在剩余的可用选出一个最符合条件的节点

## kube-scheduler简介:

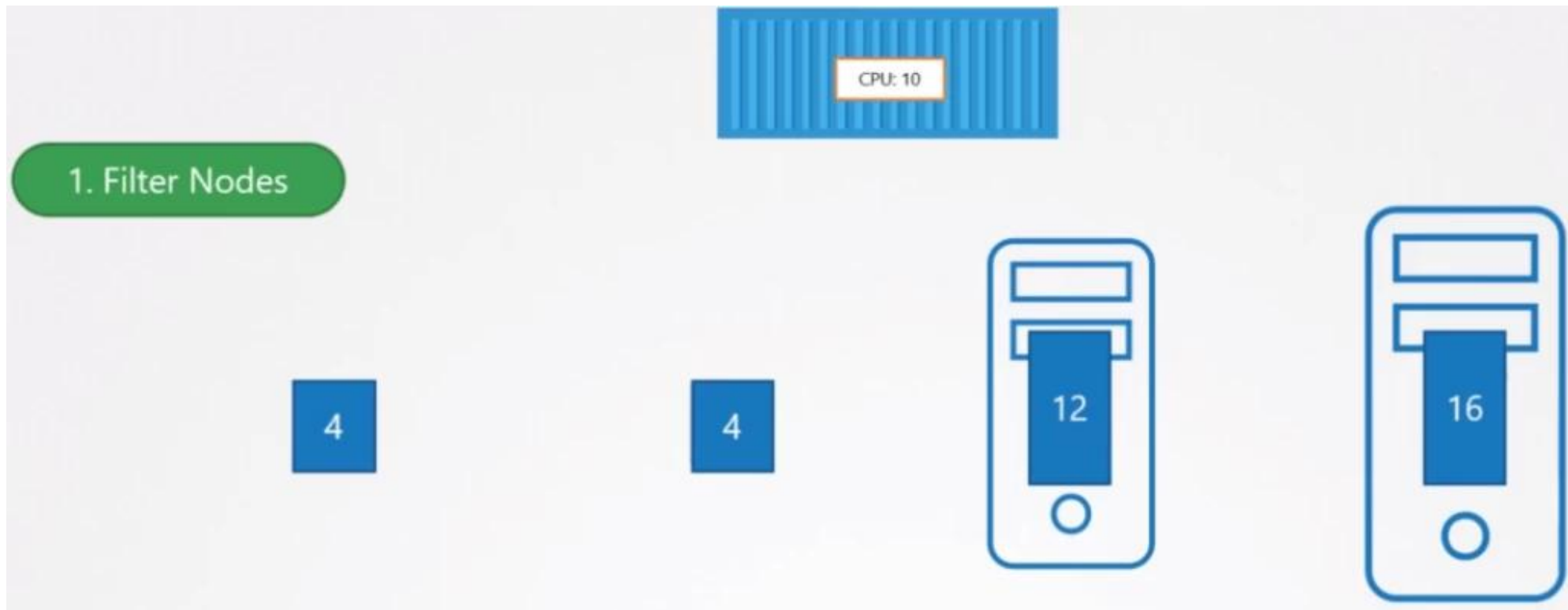
### 第一步: 创建pod





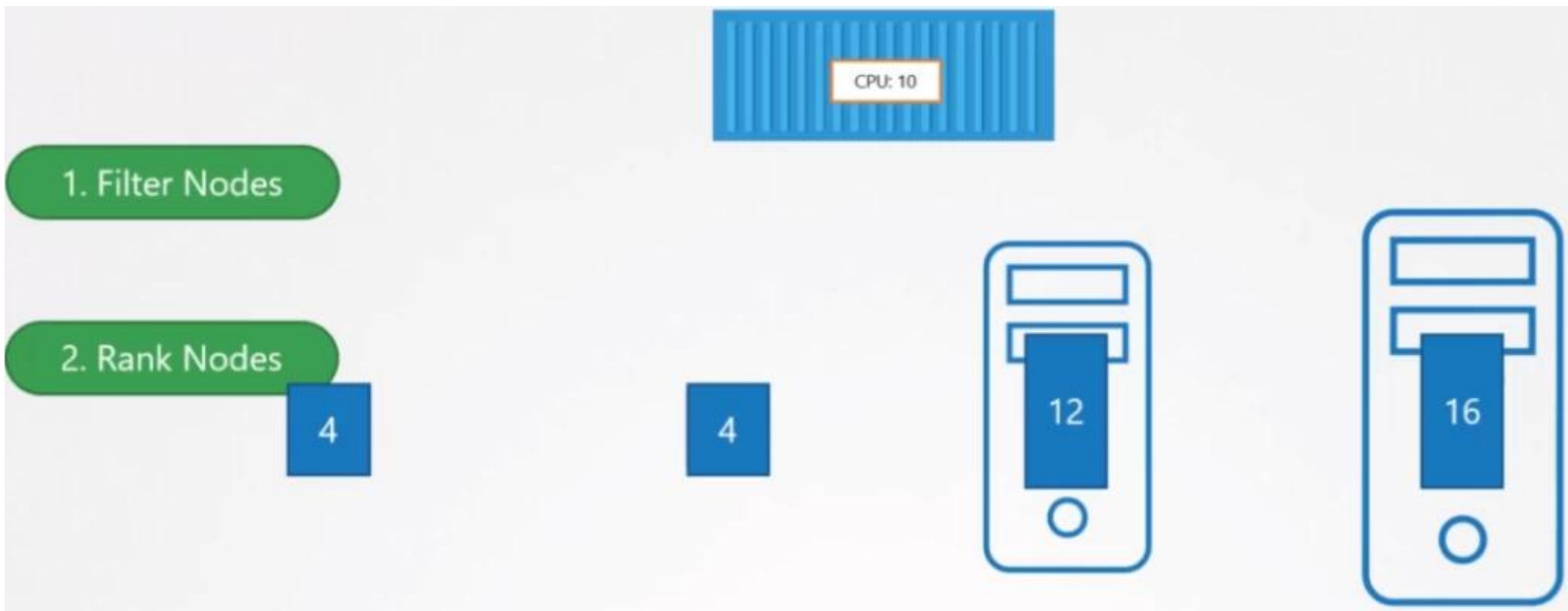
## kube-scheduler简介:

第二步：过滤掉资源不足的节点



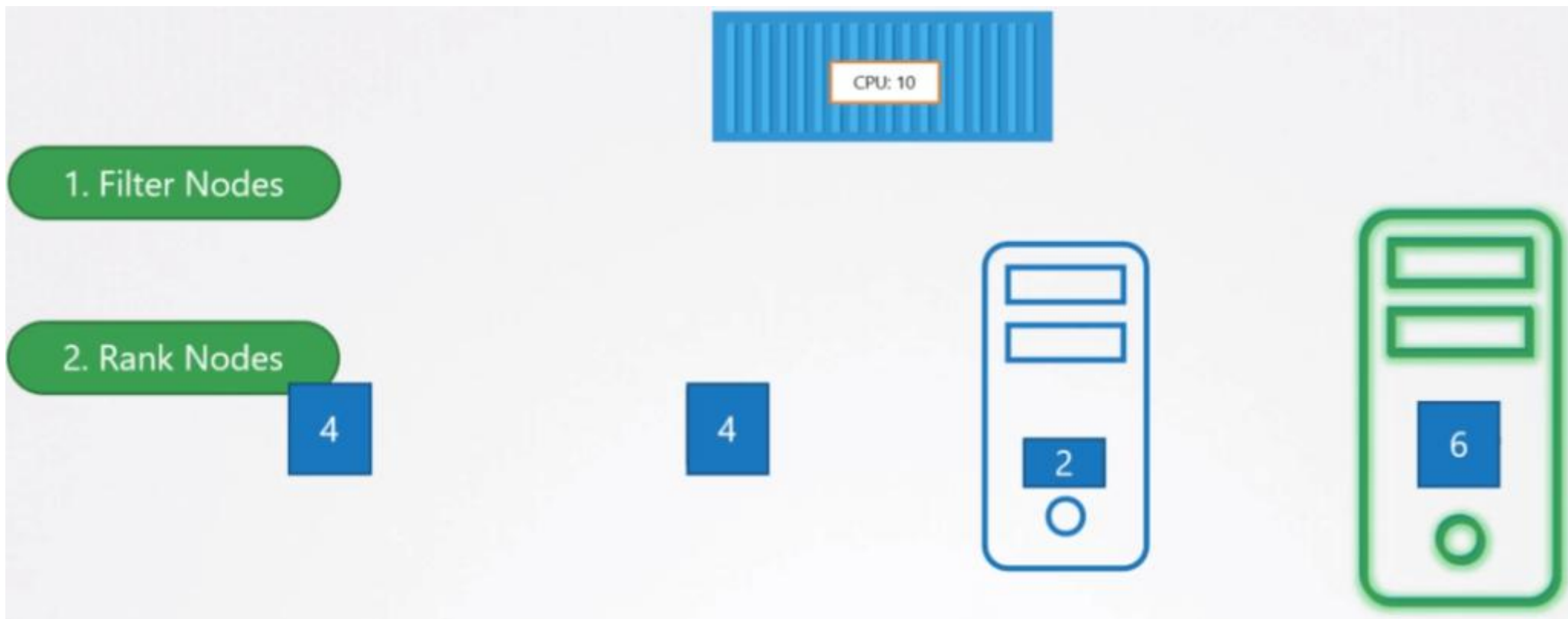
## kube-scheduler简介:

第三步：在剩余可用的节点中进行删选

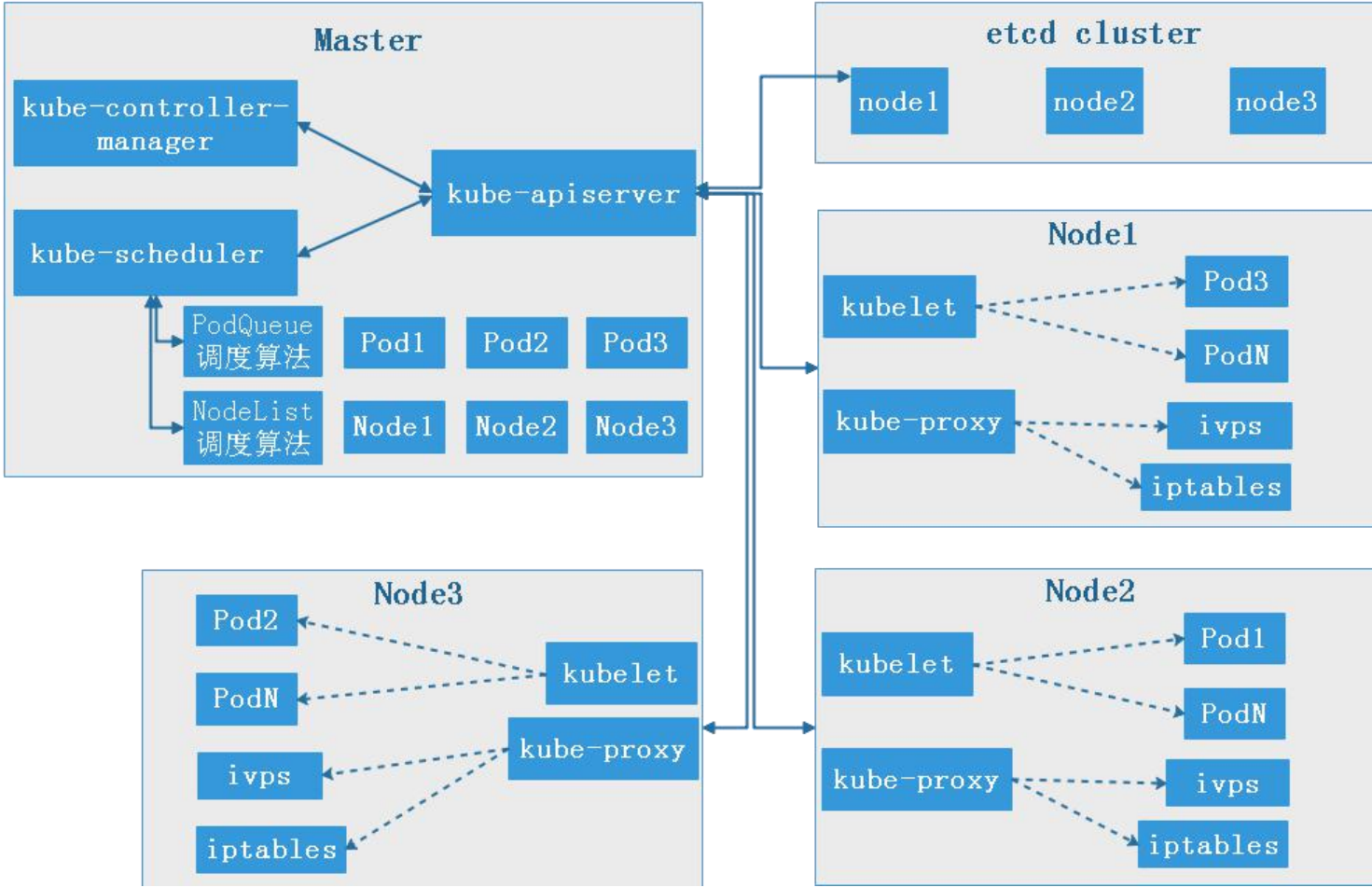


## kube-scheduler简介:

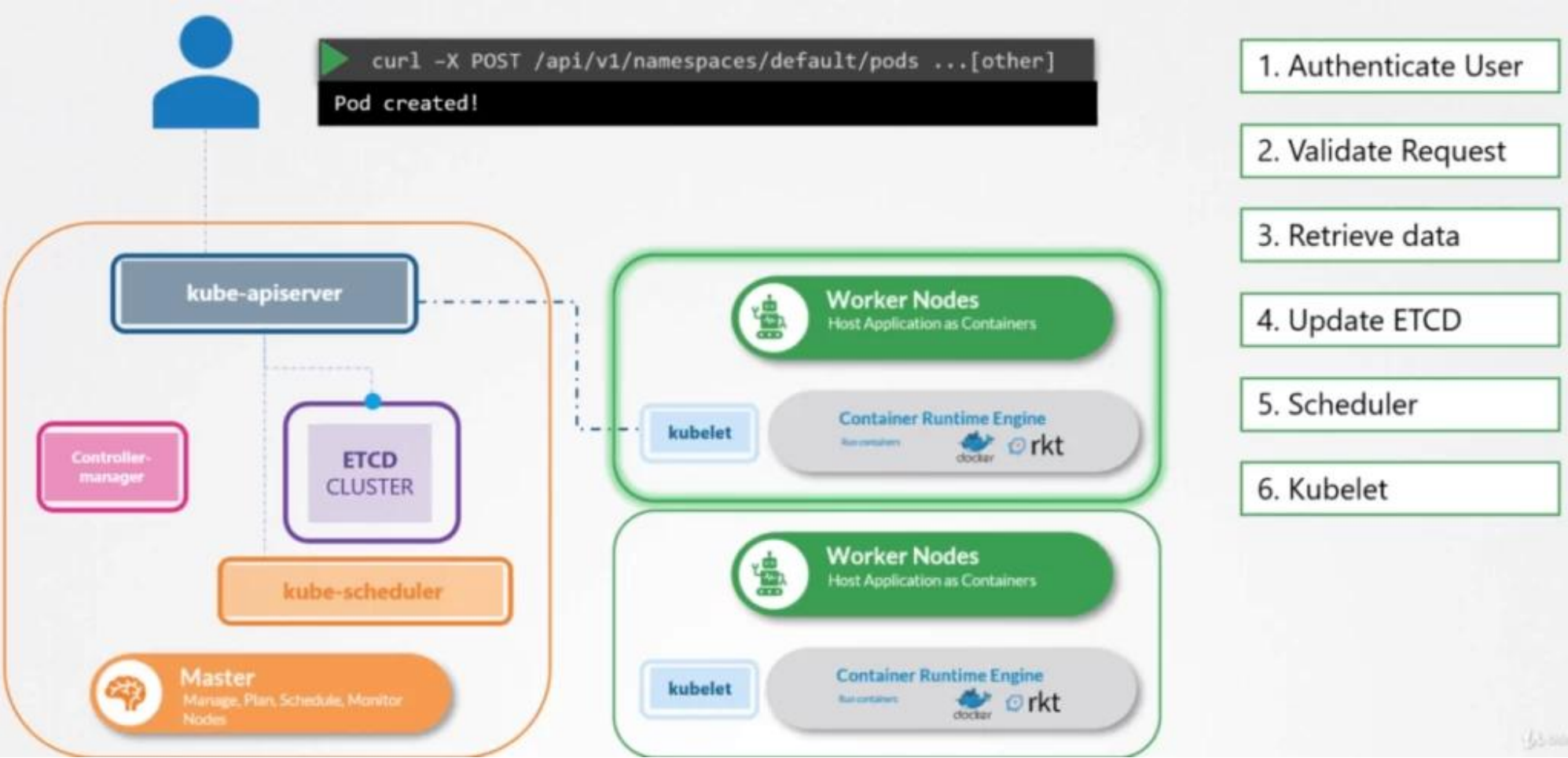
### 第四步：选中节点



kube-scheduler简介:



kube-scheduler简介:



## kubernetes 组件简介:

### ➤ kube-controller-manager:

<https://kubernetes.io/zh/docs/reference/command-line-tools-reference/kube-controller-manager/>

- kube-controller-manager: Controller Manager还包括一些子控制器(副本控制器、节点控制器、命名空间控制器和服务账号控制器等), 控制器作为集群内部的管理控制中心, 负责集群内的Node、Pod副本、服务端点 (Endpoint)、命名空间 (Namespace)、服务账号 (ServiceAccount)、资源定额 (ResourceQuota) 的管理, 当某个Node意外宕机时, Controller Manager会及时发现并执行自动化修复流程, 确保集群中的pod副本始终处于预期的工作状态。
- controller-manager控制器每间隔5秒检查一次节点的状态。
- 如果controller-manager控制器没有收到自节点的心跳, 则将该node节点被标记为不可达。
- controller-manager将在标记为无法访问之前等待40秒。
- 如果该node节点被标记为无法访问后5分钟还没有恢复, controller-manager会删除当前node节点的所有pod并在其它可用节点重建这些pod。

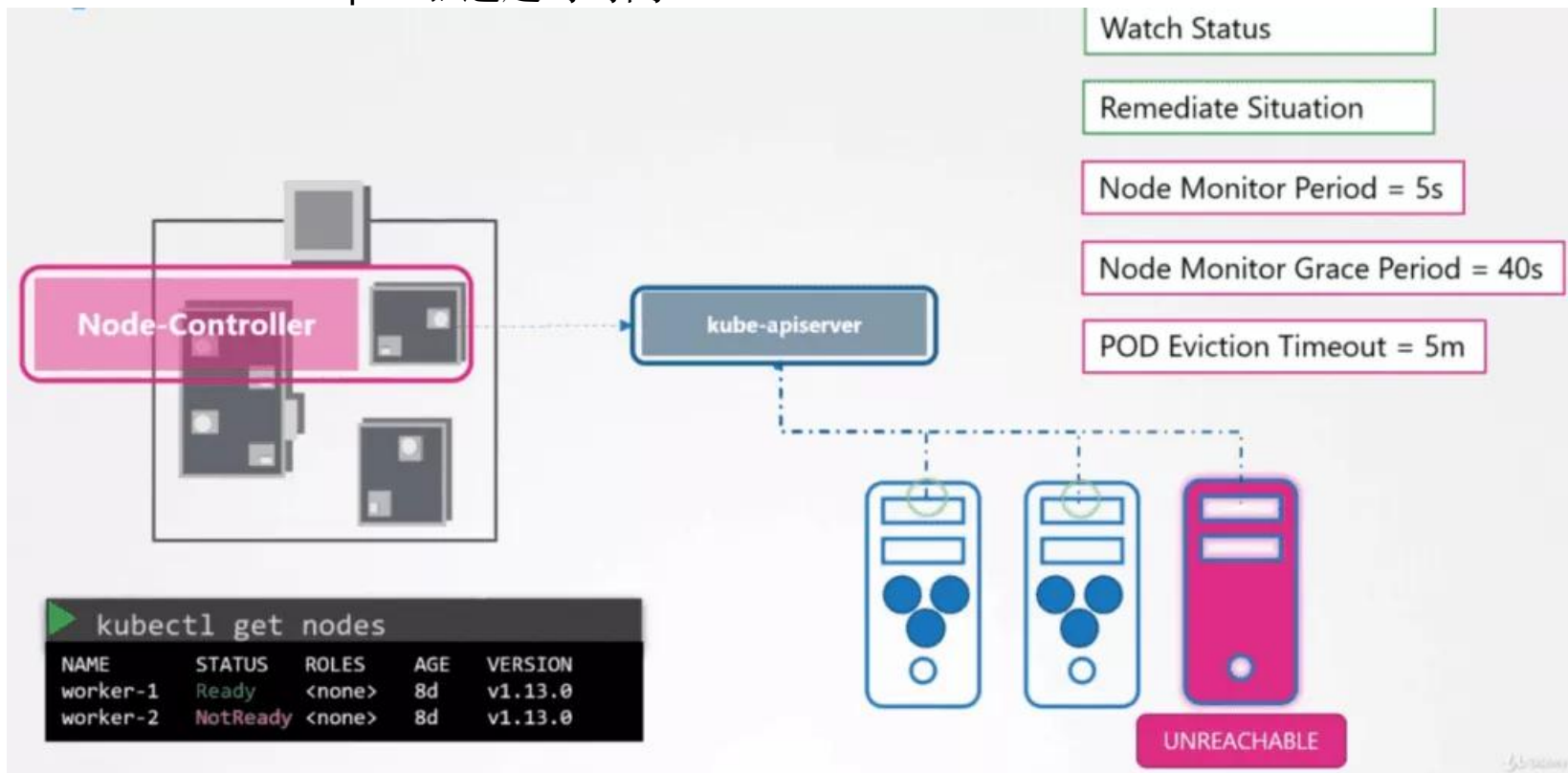
# kube-controller-manager简介:

pod 高可用机制:

node monitor period: 节点监视周期

node monitor grace period: 节点监视器宽限期

pod eviction timeout: pod驱逐超时时间



## kubernetes 组件简介:

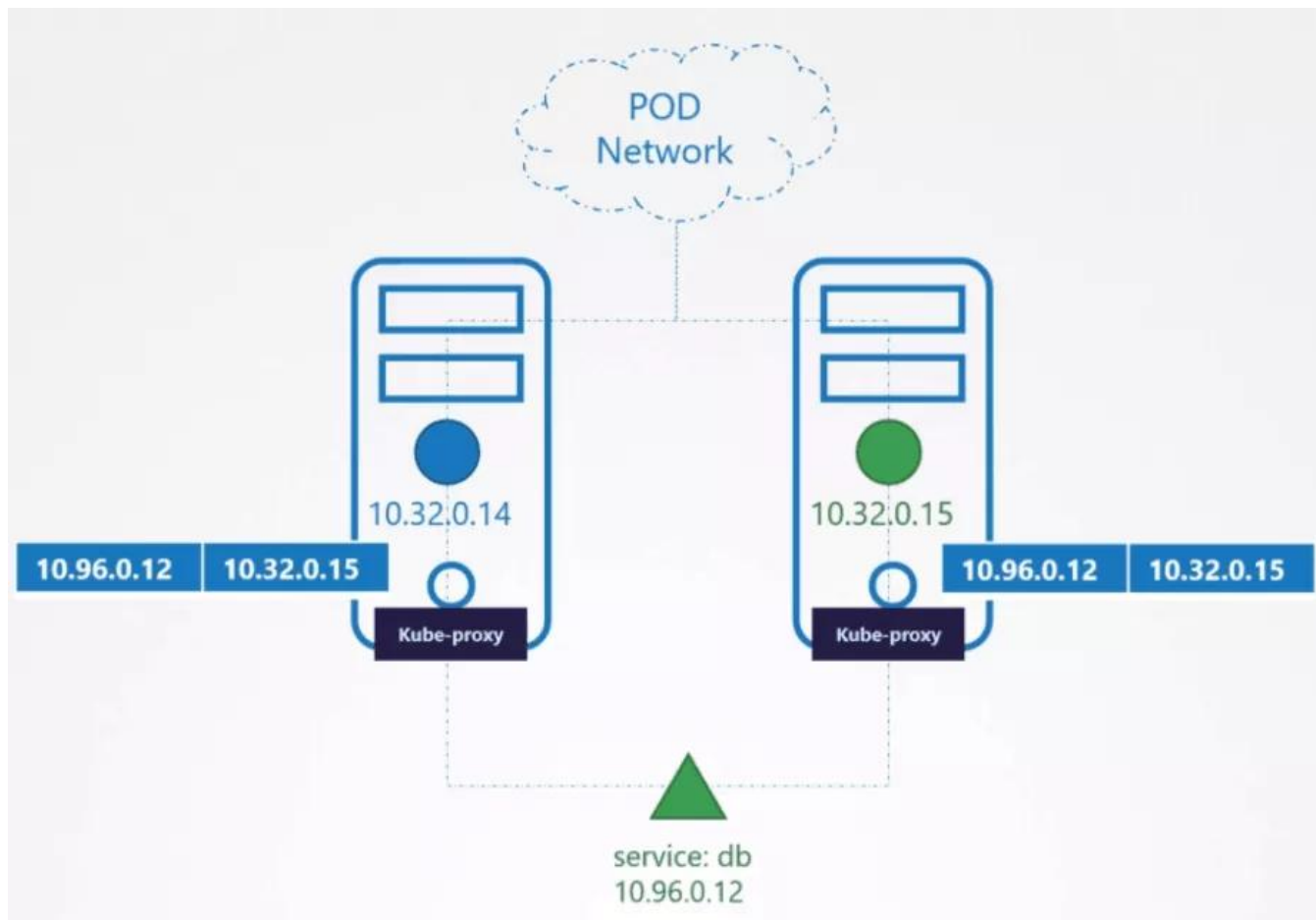
### ➤ kube-proxy:

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

- kube-proxy: Kubernetes 网络代理运行在 node 上, 它反映了 node 上 Kubernetes API 中定义的服务, 并可以通过一组后端进行简单的 TCP、UDP 和 SCTP 流转发或者在一组后端进行循环 TCP、UDP 和 SCTP 转发, 用户必须使用 apiserver API 创建一个服务来配置代理, 其实就是kube-proxy通过在主机上维护网络规则并执行连接转发来实现Kubernetes 服务访问。
- kube-proxy 运行在每个节点上, 监听 API Server 中服务对象的变化, 再通过管理 IPtables 或者IPVS规则 来实现网络的转发。
- Kube-Proxy 不同的版本可支持三种工作模式:
  - UserSpace: k8s v1.1之前使用,k8s 1.2及以后就已经淘汰
  - IPtables : k8s 1.1版本开始支持, 1.2开始为默认
  - IPVS: k8s 1.9引入到1.11为正式版本, 需要安装ipvsadm、ipset 工具包和加载 ip\_vs 内核模块



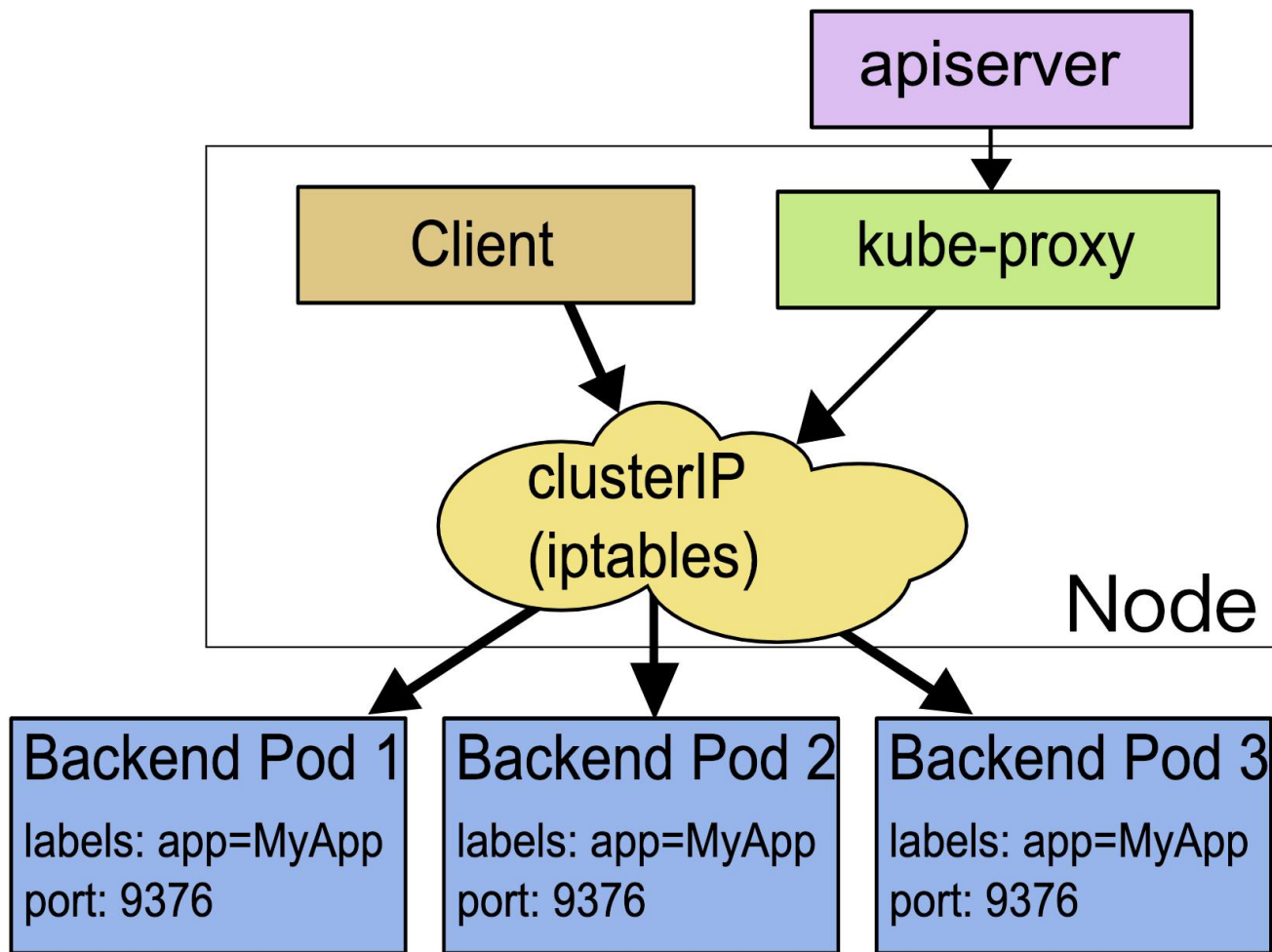
## kube-proxy简介:



- 1.node 网络
- 2.pod网络
- 3.service 网络

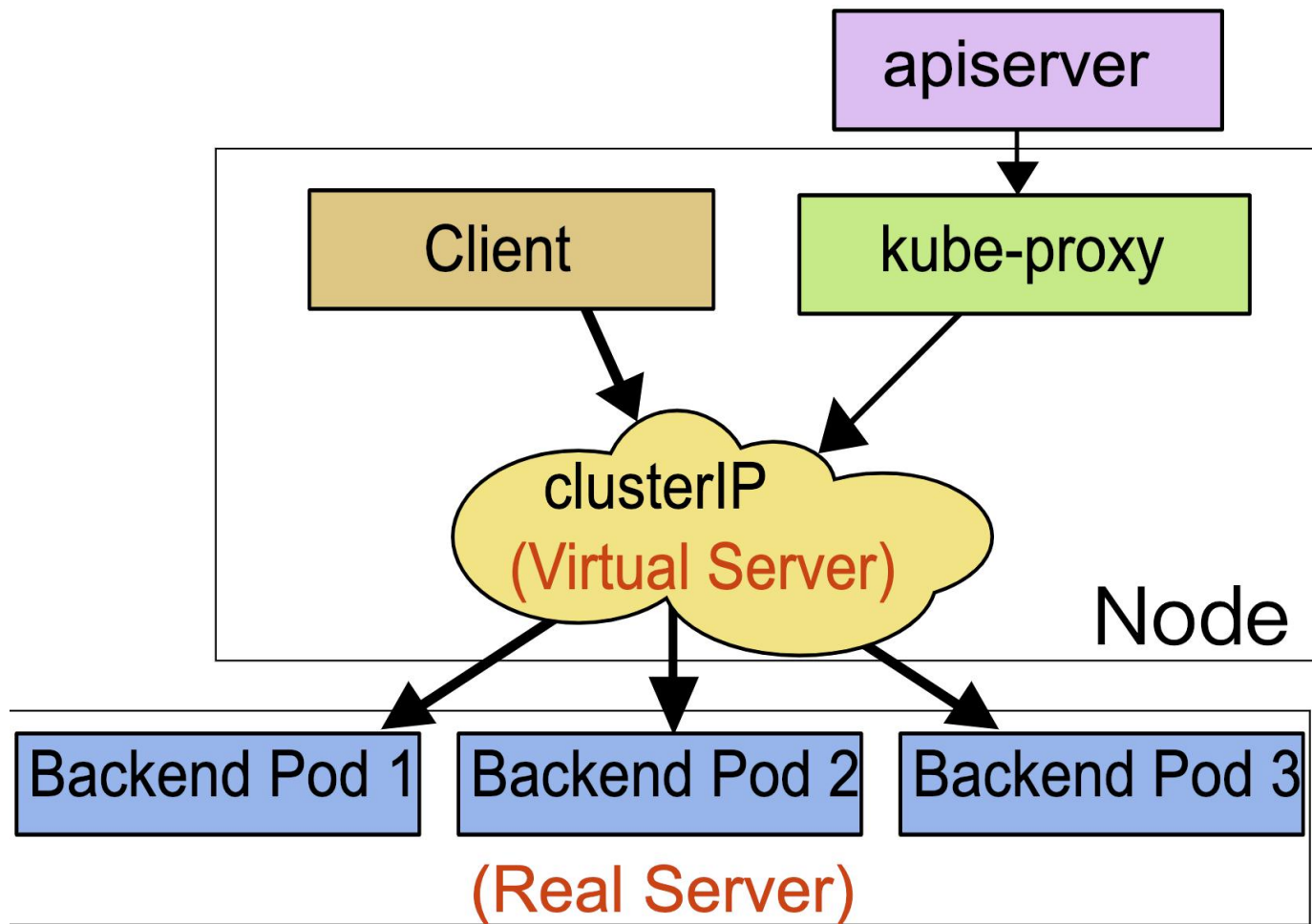
kube-proxy简介:

iptables 模式:



kube-proxy简介:

ipvs 模式:



## kube-proxy简介:

- IPVS 相对 IPtables 效率会更高一些，使用 IPVS 模式需要在运行 Kube-Proxy 的节点上安装 ipvsadm、ipset 工具包和加载 ip\_vs 内核模块，当 Kube-Proxy 以 IPVS 代理模式启动时，Kube-Proxy 将验证节点上是否安装了 IPVS 模块，如果未安装，则 Kube-Proxy 将回退到 IPtables 代理模式。
- 使用IPVS模式，Kube-Proxy会监视Kubernetes Service对象和Endpoints，调用宿主机内核 Netlink接口以相应地创建IPVS规则并定期与Kubernetes Service对象 Endpoints对象同步IPVS规则，以确保IPVS状态与期望一致，访问服务时，流量将被重定向到其中一个后端 Pod,IPVS使用哈希表作为底层数据结构并在内核空间中工作，这意味着IPVS可以更快地重定向流量，并且在同步代理规则时具有更好的性能，此外，IPVS 为负载均衡算法提供了更多选项，例如：rr (轮询调度)、lc (最小连接数)、dh (目标哈希)、sh (源哈希)、sed (最短期望延迟)、nq(不排队调度)等。

# kube-proxy简介:

配置使用IPVS及指定调度算法:

<https://kubernetes.io/zh/docs/reference/config-api/kube-proxy-config.v1alpha1/#ClientConnectionConfiguration>

```
# cat /var/lib/kube-proxy/kube-proxy-config.yaml
kind: KubeProxyConfiguration
apiVersion: kubeproxy.config.k8s.io/v1alpha1
bindAddress: 172.31.7.111
clientConnection:
  kubeconfig: "/etc/kubernetes/kube-proxy.kubeconfig"
clusterCIDR: "10.100.0.0/16"
conntrack:
  maxPerCore: 32768
  min: 131072
  tcpCloseWaitTimeout: 1h0m0s
  tcpEstablishedTimeout: 24h0m0s
healthzBindAddress: 172.31.7.111:10256
hostnameOverride: "172.31.7.111"
metricsBindAddress: 172.31.7.111:10249
mode: "ipvs" #指定使用ipvs及调度算法
ipvs:
  scheduler: sh
```

## kube-proxy简介:

开启会话保持

```
#cat nginx-service.yaml
```

```
kind: Service
```

```
apiVersion: v1
```

```
metadata:
```

```
  labels:
```

```
    app: magedu-nginx-service-label
```

```
  name: magedu-nginx-service
```

```
  namespace: magedu
```

```
spec:
```

```
  type: NodePort
```

```
  ports:
```

```
    - name: http
```

```
      port: 80
```

```
      protocol: TCP
```

```
      targetPort: 80
```

```
      nodePort: 30004
```

```
  selector:
```

```
    app: nginx
```

```
  sessionAffinity: ClientIP
```

```
  sessionAffinityConfig:
```

```
    clientIP:
```

```
      timeoutSeconds: 1800
```

## kubernetes 组件简介:

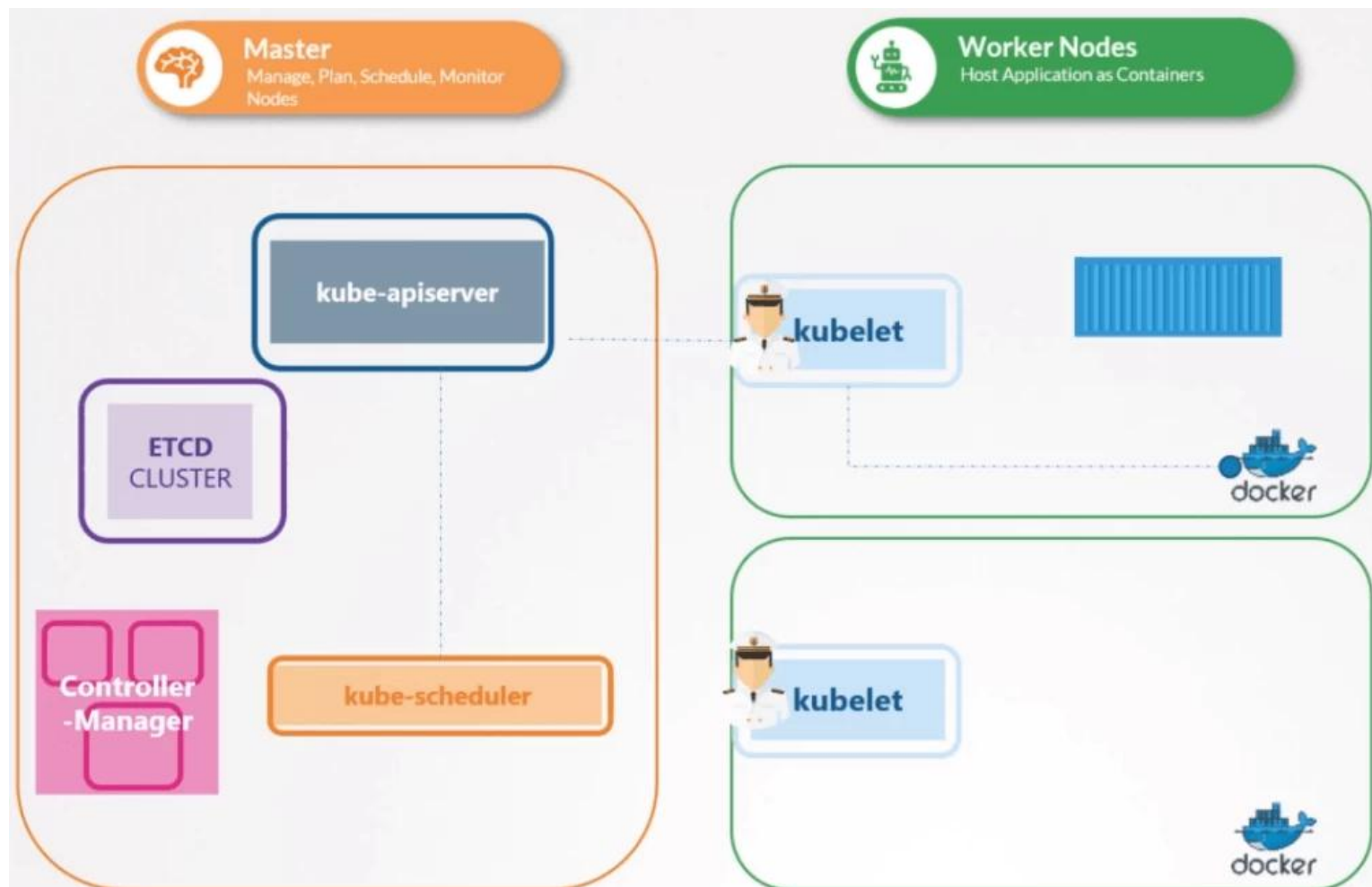
### ➤ kubelet:

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

kubelet是运行在每个worker节点的代理组件，它会监视已分配给节点的pod，具体功能如下：

- 向master汇报node节点的状态信息
- 接受指令并在Pod中创建 docker容器
- 准备Pod所需的数据卷
- 返回pod的运行状态
- 在node节点执行容器健康检查

# kubelet简介:





# kubernetes 组件简介:

## ➤ kubectl:

<https://kubernetes.io/zh/docs/reference/kubectl/kubectl/>

- 是一个通过命令行对kubernetes集群进行管理的客户端工具。

## kubernetes 组件简介:

### ➤ etcd:

<https://kubernetes.io/zh/docs/tasks/administer-cluster/configure-upgrade-etcd/>

- etcd 是CoreOS公司开发目前是Kubernetes默认使用的key-value数据存储系统，用于保存kubernetes的所有集群数据，etcd支持分布式集群功能，生产环境使用时需要为etcd数据提供定期备份机制。
- <https://etcd.io/> #官网
- <https://github.com/etcd-io/etcd> #github

etcd简介:

Key	Value
Name	John Doe
Age	45
Location	New York
Salary	5000

Key	Value
Name	Dave Smith
Age	34
Location	New York
Salary	4000

Key	Value
Name	Aryan Kumar
Age	10
Location	New York
Grade	A

Key	Value
Name	Lauren Rob
Age	13
Location	Bangalore
Grade	C

Key	Value
Name	Lily Oliver
Age	15
Location	Bangalore
Grade	B

# kubernetes 组件简介:

## ➤ DNS:

<https://kubernetes.io/zh/docs/tasks/administer-cluster/dns-custom-nameservers/>

- DNS负责为整个集群提供DNS服务，从而实现服务之间的访问。

coredns

kube-dns: 1.18

sky-dns

## kubernetes 组件简介:

Dashboard:


<https://kubernetes.io/zh/docs/tasks/access-application-cluster/web-ui-dashboard/>

- Dashboard是基于网页的Kubernetes用户界面，可以使用Dashboard获取运行在集群中的应用的概览信息，也可以创建或者修改Kubernetes资源（如 Deployment，Job，DaemonSet 等等），也可以对Deployment实现弹性伸缩、发起滚动升级、重启 Pod 或者使用向导创建新的应用。

Firefox文件 编辑 查看 历史 书签 工具 窗口 帮助

Kubernetes Dashboard

https://172.31.7.111:30002/#/pod?namespace=\_all

kubernetes

全部命名空间

搜索

Workloads > Pods

Pods

Replica Sets

Replication Controllers

Stateful Sets

服务 N

Ingresses

Services

配置和存储

Config Maps N

Persistent Volume Claims N

Secrets N

Storage Classes

集群

Cluster Role Bindings

Cluster Roles






命名空间

网络策略 N

Nodes

Persistent Volumes

Pods

名称	命名空间	镜像	标签	节点	状态	重启
 <a href="#">kubernetes-dashboard-79b875f7f8-zb8xs</a>	kubernetes-dashboard	kubernetesui/dashboard:v2.3.1	k8s-app: kubernetes-dashboar d pod-template-hash: 79b875f7f8	172.31.7.111	Running	0
 <a href="#">dashboard-metrics-scraper-856586f554-mqxn timer</a>	kubernetes-dashboard	kubernetesui/metrics-scraper:v1.0.6	k8s-app: dashboard-metrics-scraper pod-template-hash: 856586f554	172.31.7.113	Running	0
 <a href="#">coredns-f97dc456d-pj ktp</a>	kube-system	coredns/coredns:1.8.3	k8s-app: kube-dns pod-template-hash: f97dc456d	172.31.7.111	Running	0
 <a href="#">kube-flannel-ds-amd64-lxw9g</a>	kube-system	easylab/flannel:v0.13.0-amd64	app: flannel controller-revision-hash: 76f84785b5 pod-template-generation: 1 <a href="#">显示所有</a>	172.31.7.103	Running	0
 <a href="#">kube-flannel-ds-amd64-t4xml</a>	kube-system	easylab/flannel:v0.13.0-amd64	app: flannel controller-revision-hash: 76f84785b5 pod-template-generation: 1 <a href="#">显示所有</a>	172.31.7.113	Running	0

kubernetes 高可用集群部署:

## kubectl 常用命令:

```
# kubectl get service --all-namespaces -o wide
# kubectl get pods --all-namespaces -o wide
# kubectl get nodes --all-namespaces -o wide
# kubectl get deployment --all-namespaces
# kubectl get deployment -n magedu -o wide #更改显示格式
# kubectl describe pods magedu-tomcat-app1-deployment -n magedu #查看某个资源详细信息
# kubectl create -f tomcat-app1.yaml
# kubectl apply -f tomcat-app1.yaml
# kubectl delete -f tomcat-app1.yaml
# kubectl create -f tomcat-app1.yaml --save-config --record
# kubectl apply -f tomcat-app1.yaml --record #推荐命令
# kubectl exec -it magedu-tomcat-app1-deployment-6bccd8f9c7-g76s5 bash -n magedu
# kubectl logs magedu-tomcat-app1-deployment-6bccd8f9c7-g76s5 -n magedu
# kubectl delete pods magedu-tomcat-app1-deployment-6bccd8f9c7-g76s5 -n magedu
```