Kubernetes容器云平台入门与进阶

讲师介绍

阿良

资深运维工程师，51CTO知名博主。曾就职在IDC，大数据，金融行业，现任职奇虎360公司负责PC浏览器业务。经重重磨炼，具有丰富的运维实战经验。

技术博客：http://blog.51cto.com/lizhenliang

阿良微信

入门须知

- 熟悉Linux基础命令
- 熟悉Docker基本管理
- 了解SSL证书工作原理
- 了解负载均衡工作原理（L4/L7）
- 了解集群，分布式概念
- 了解域名解析原理
- 了解网络协议

第 1 章 Kubernetes概述

1. Kubernetes是什么
2. Kubernetes特性
3. Kubernetes集群架构与组件
4. Kubernetes核心概念

Kubernetes是什么

- Kubernetes是Google在2014年开源的一个容器集群管理系统，Kubernetes简称K8S。
- K8S用于容器化应用程序的部署，扩展和管理。
- K8S提供了容器编排，资源调度，弹性伸缩，部署管理，服务发现等一系列功能。
- Kubernetes目标是让部署容器化应用简单高效。

官方网站：http://www.kubernetes.io

Kubernetes特性

- **自我修复**

在节点故障时重新启动失败的容器，替换和重新部署，保证预期的副本数量；杀死健康检查失败的容器，并且在未准备好之前不会处理客户端请求，确保线上服务不中断。

- **弹性伸缩**

使用命令、UI或者基于CPU使用情况自动快速扩容和缩容应用程序实例，保证应用业务高峰并发时的高可用性；业务低峰时回收资源，以最小成本运行服务。

- **自动部署和回滚**

K8S采用滚动更新策略更新应用，一次更新一个Pod，而不是同时删除所有Pod，如果更新过程中出现问题，将回滚更改，确保升级不受影响业务。

- **服务发现和负载均衡**

K8S为多个容器提供一个统一访问入口（内部IP地址和一个DNS名称），并且负载均衡关联的所有容器，使得用户无需考虑容器IP问题。

- **机密和配置管理**

管理机密数据和应用程序配置，而不需要把敏感数据暴露在镜像里，提高敏感数据安全性。并可以将一些常用的配置存储在K8S中，方便应用程序使用。

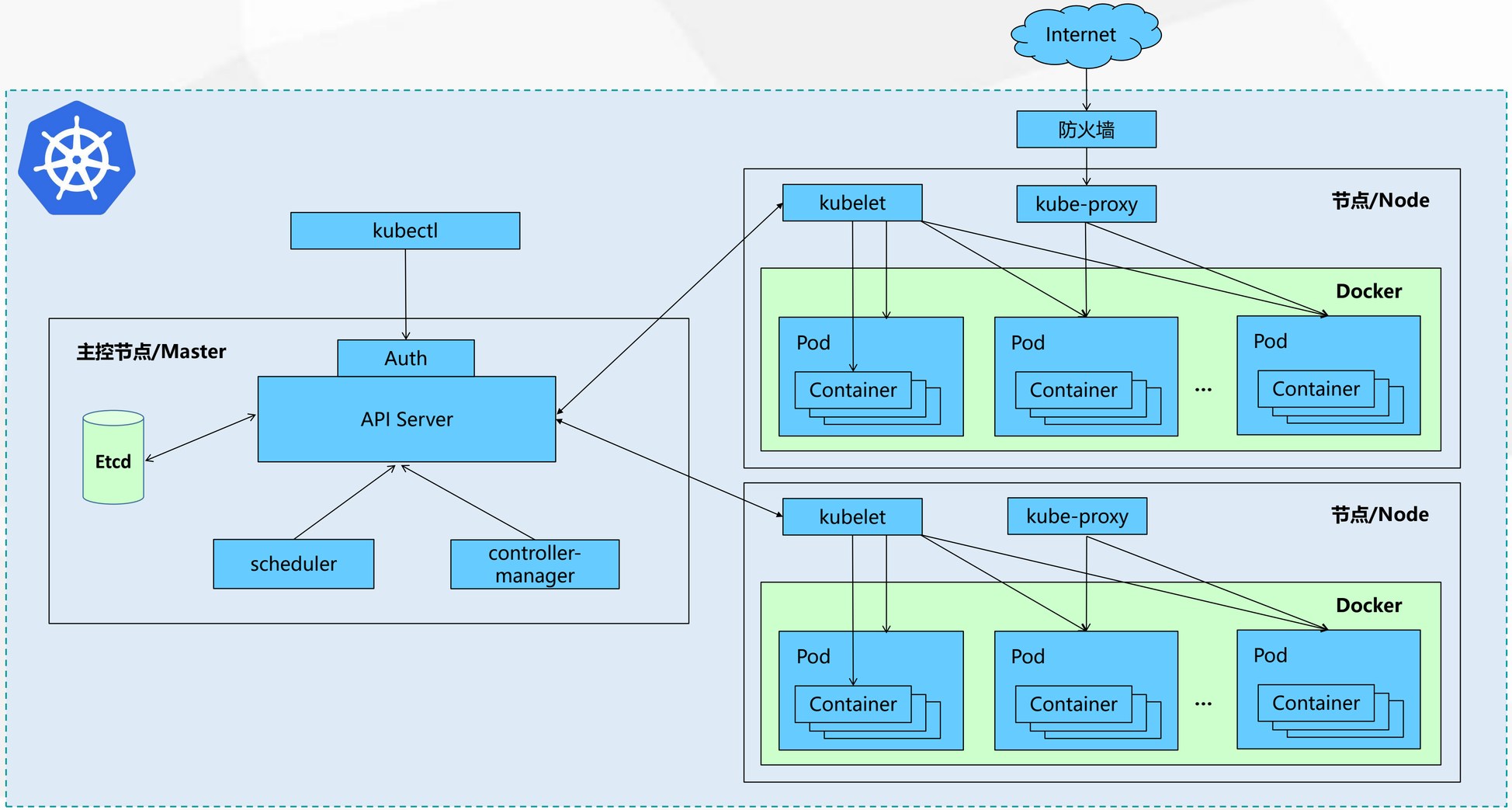
- **存储编排**

挂载外部存储系统，无论是来自本地存储，公有云（如AWS），还是网络存储（如NFS、GlusterFS、Ceph）都作为集群资源的一部分使用，极大提高存储使用灵活性。

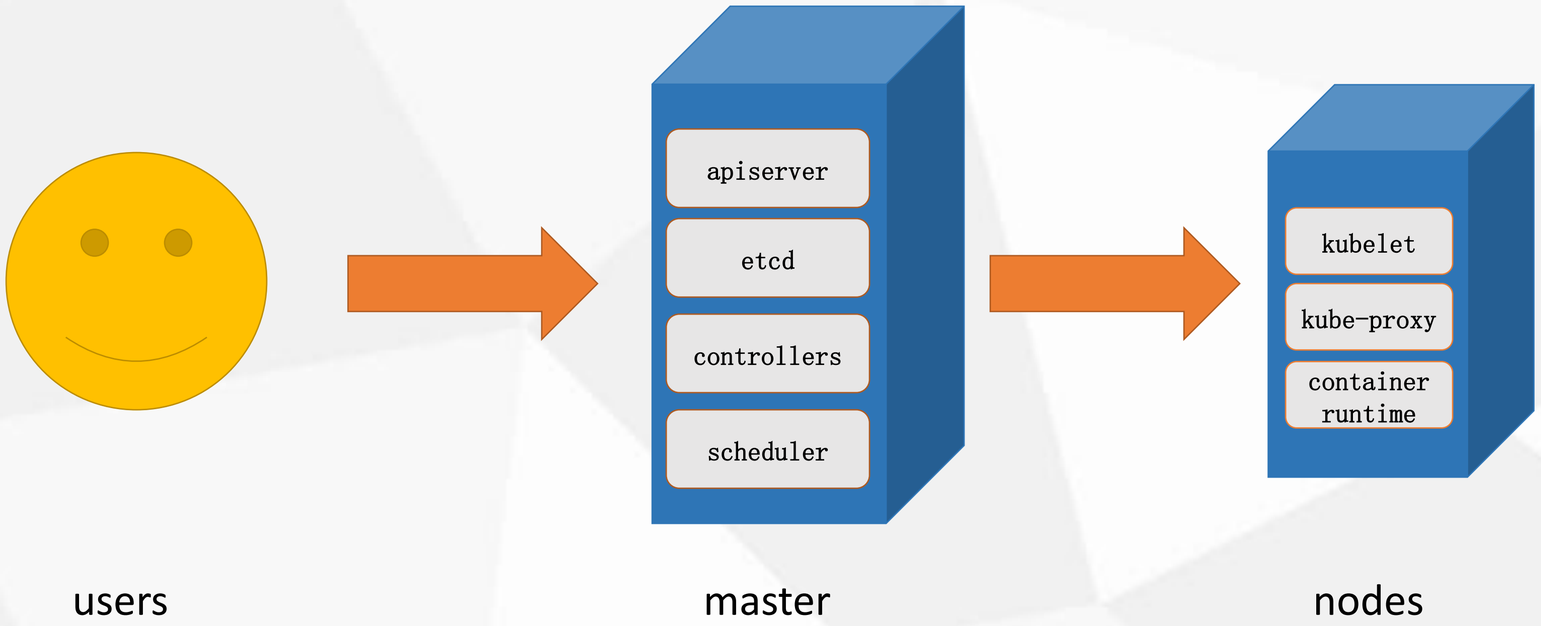
- **批处理**

提供一次性任务，定时任务；满足批量数据处理和分析的场景。

Kubernetes集群架构与组件



Kubernetes集群架构与组件



Kubernetes集群架构与组件

Master组件

- **kube-apiserver**

Kubernetes API，集群的统一入口，各组件协调者，以RESTful API提供接口服务，所有对象资源的增删改查和监听操作都交给APIServer处理后再提交给Etcd存储。

- **kube-controller-manager**

处理集群中常规后台任务，一个资源对应一个控制器，而ControllerManager就是负责管理这些控制器的。

- **kube-scheduler**

根据调度算法为新创建的Pod选择一个Node节点，可以任意部署,可以部署在同一个节点上,也可以部署在不同的节点上。

- **etcd**

分布式键值存储系统。用于保存集群状态数据，比如Pod、Service等对象信息。

Node组件

- **kubelet**

kubelet是Master在Node节点上的Agent，管理本机运行容器的生命周期，比如创建容器、Pod挂载数据卷、下载secret、获取容器和节点状态等工作。kubelet将每个Pod转换成一组容器。

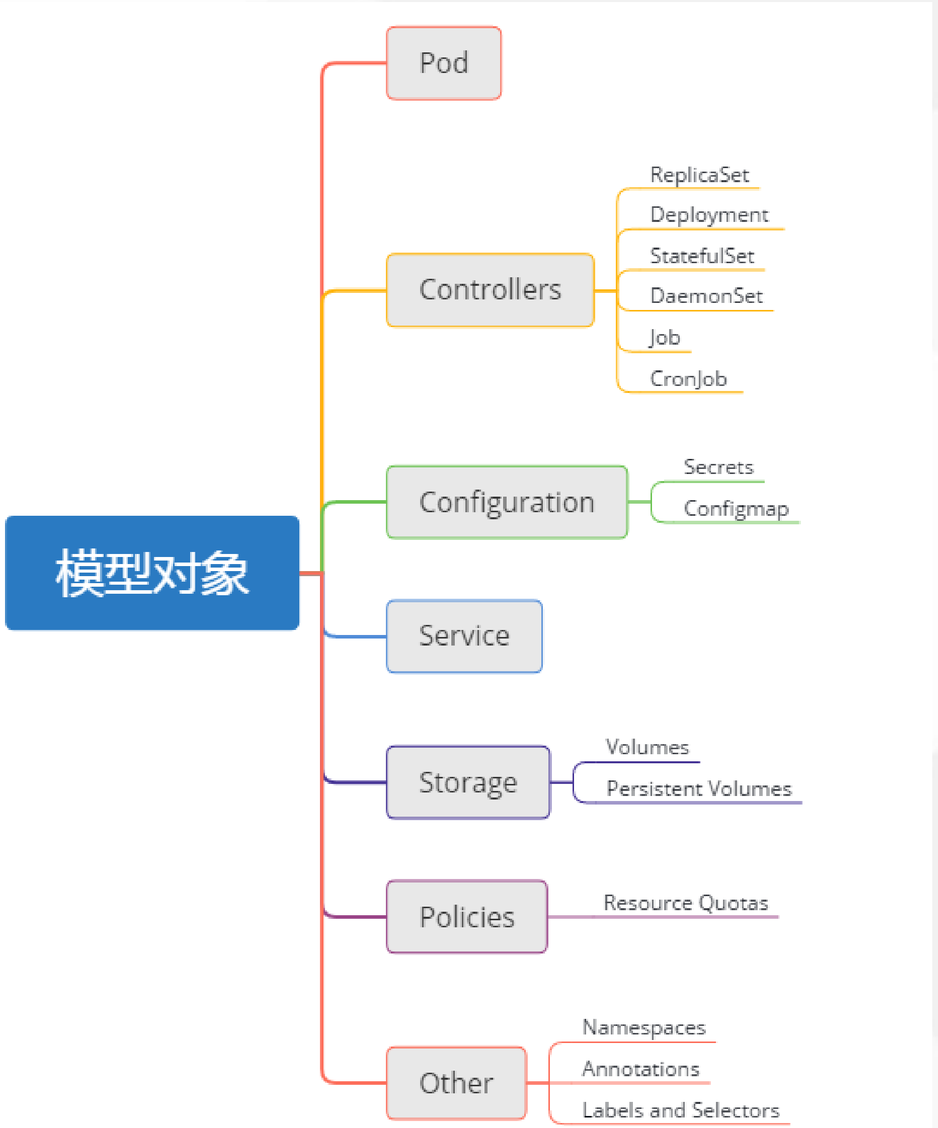
- **kube-proxy**

在Node节点上实现Pod网络代理，维护网络规则和四层负载均衡工作。

- **docker或rocket**

容器引擎，运行容器。

Kubernetes核心概念



Kubernetes核心概念

- **Pod**
  - 最小部署单元
  - 一组容器的集合
  - 一个Pod中的容器共享网络命名空间
  - Pod是短暂的

- **Controllers**
  - ReplicaSet： 确保预期的Pod副本数量
  - Deployment： 无状态应用部署
  - StatefulSet： 有状态应用部署
  - DaemonSet： 确保所有Node运行同一个Pod
  - Job： 一次性任务
  - Cronjob： 定时任务

  更高级层次对象，部署和管理Pod

- **Service**
  - 防止Pod失联
  - 定义一组Pod的访问策略

Kubernetes核心概念

- Label： 标签，附加到某个资源上，用于关联对象、查询和筛选
- Namespaces： 命名空间，将对象逻辑上隔离
- Annotations： 注释

第 2 章 Kubernetes集群部署

1. 官方提供的三种部署方式
2. Kubernetes平台环境规划
3. 自签SSL证书
4. Etcd数据库集群部署
5. Node安装Docker
6. Flannel容器集群网络部署
7. 部署Master组件
8. 部署Node组件
9. 部署一个测试示例
10. 部署Web UI（Dashboard）
11. 部署集群内部DNS解析服务（CoreDNS）

Kubernetes平台环境规划

软件	版本
Linux操作系统	CentOS7.5_x64
Kubernetes	1.12
Docker	18.xx-ce
Etcd	3.x
Flannel	0.10

角色	IP	组件	推荐配置
master01	192.168.31.63	kube-apiserver kube-controller-manager kube-scheduler etcd	CPU：2C+ 内存：4G+
master02	192.168.31.64	kube-apiserver kube-controller-manager kube-scheduler etcd	
node01	192.168.31.65	kubelet kube-proxy docker flannel etcd	
node02	192.168.31.66	kubelet kube-proxy docker flannel	
Load Balancer（Master）	192.168.31.61 192.168.31.60 (VIP)	Nginx L4	
Load Balancer（Backup）	192.168.31.62	Nginx L4	
Registry	192.168.31.66	Harbor	

官方提供的三种部署方式

- **minikube**

Minikube是一个工具，可以在本地快速运行一个单点的Kubernetes，仅用于尝试Kubernetes或日常开发的用户使用。

部署地址：https://kubernetes.io/docs/setup/minikube/

- **kubeadm**

Kubeadm也是一个工具，提供kubeadm init和kubeadm join，用于快速部署Kubernetes集群。

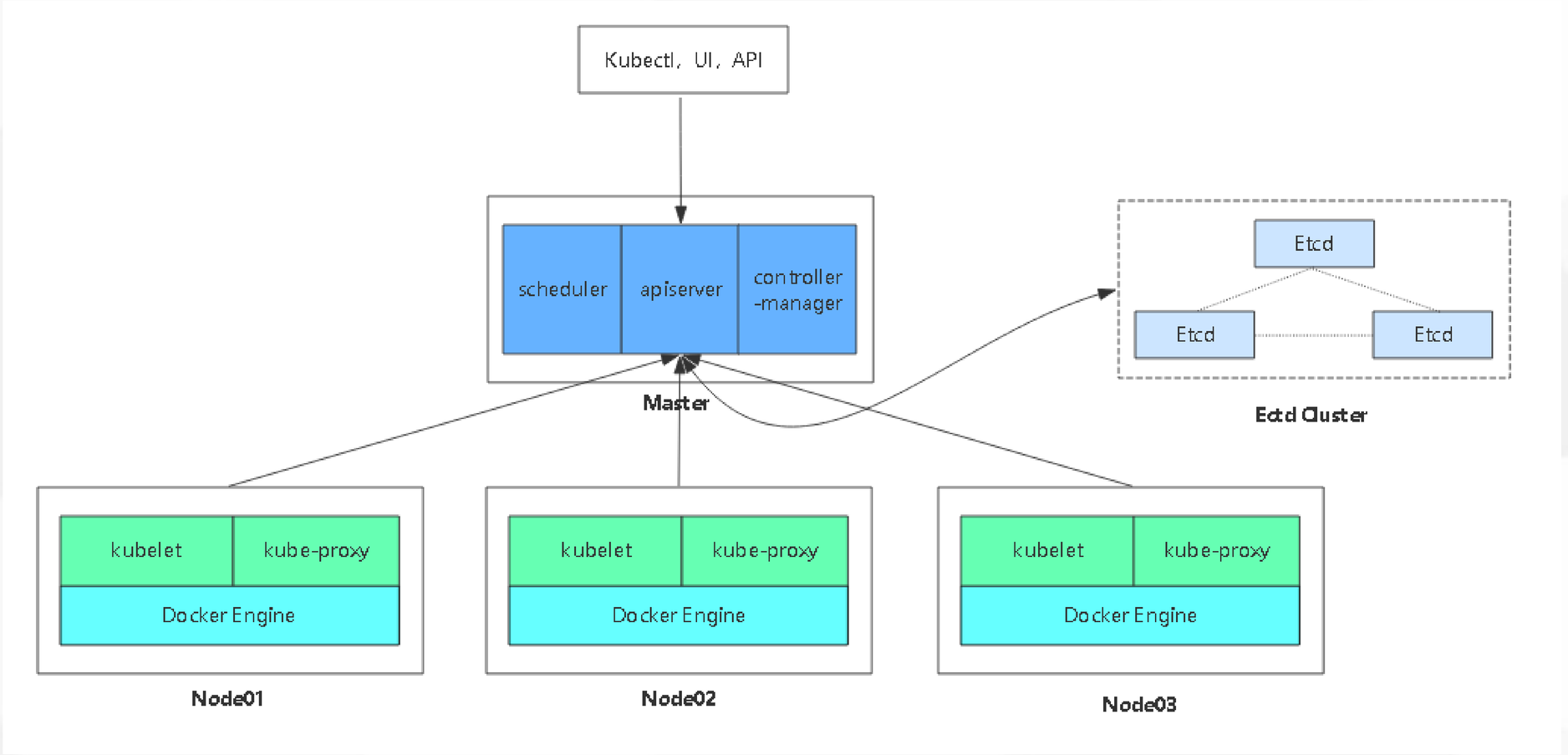
部署地址：https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm/

- **二进制包**

推荐，从官方下载发行版的二进制包，手动部署每个组件，组成Kubernetes集群。

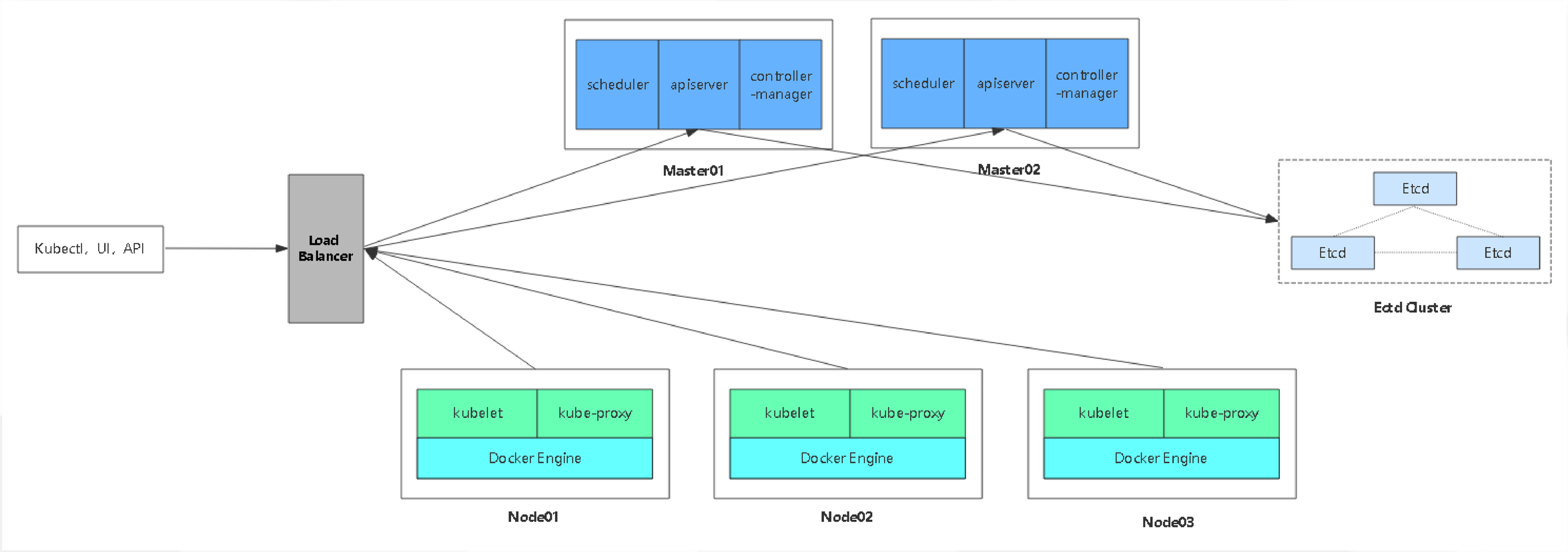
下载地址：https://github.com/kubernetes/kubernetes/releases

Kubernetes平台环境规划



单Master集群架构图

Kubernetes平台环境规划



多Master集群架构图

自签SSL证书

组件	使用的证书
etcd	ca.pem，server.pem，server-key.pem
flannel	ca.pem，server.pem，server-key.pem
kube-apiserver	ca.pem，server.pem，server-key.pem
kubelet	ca.pem，ca-key.pem
kube-proxy	ca.pem，kube-proxy.pem，kube-proxy-key.pem
kubectl	ca.pem，admin.pem，admin-key.pem

Etcd数据库集群部署

- **二进制包下载地址**

https://github.com/etcd-io/etcd/releases

- **查看集群状态**

```
/opt/etcd/bin/etcdctl \
--ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem \
--endpoints="https://192.168.0.x:2379,https://192.168.0.x:2379,https://192.168.0.x:2379" \
cluster-health
```

Node安装Docker

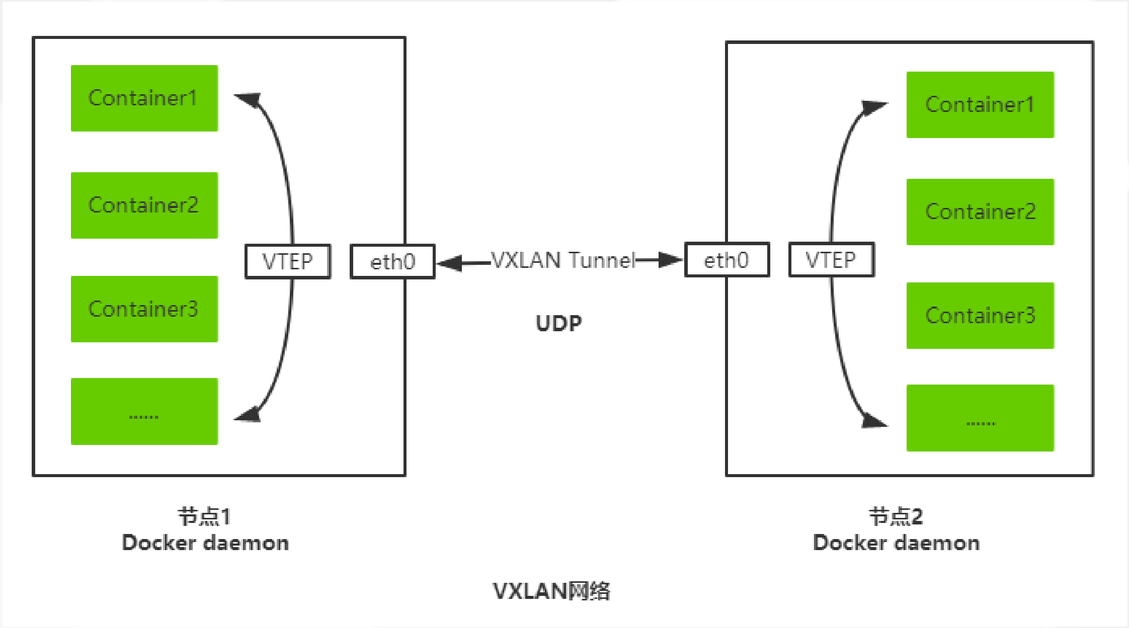
访问和工具层	帮助用户更高效的完成任务，包括Web控制台、RESTful API、CI/CD、监控管理、日志管理等...
PaaS服务层	为开发、测试和运维人员提供统一的平台服务
容器编排层	 Kubernetes
容器引擎层	 Docker
IaaS基础设施层	提供基础运行环境，例如虚拟机、物理机、网络、存储、数据库、公有云等...

Flannel容器集群网络部署

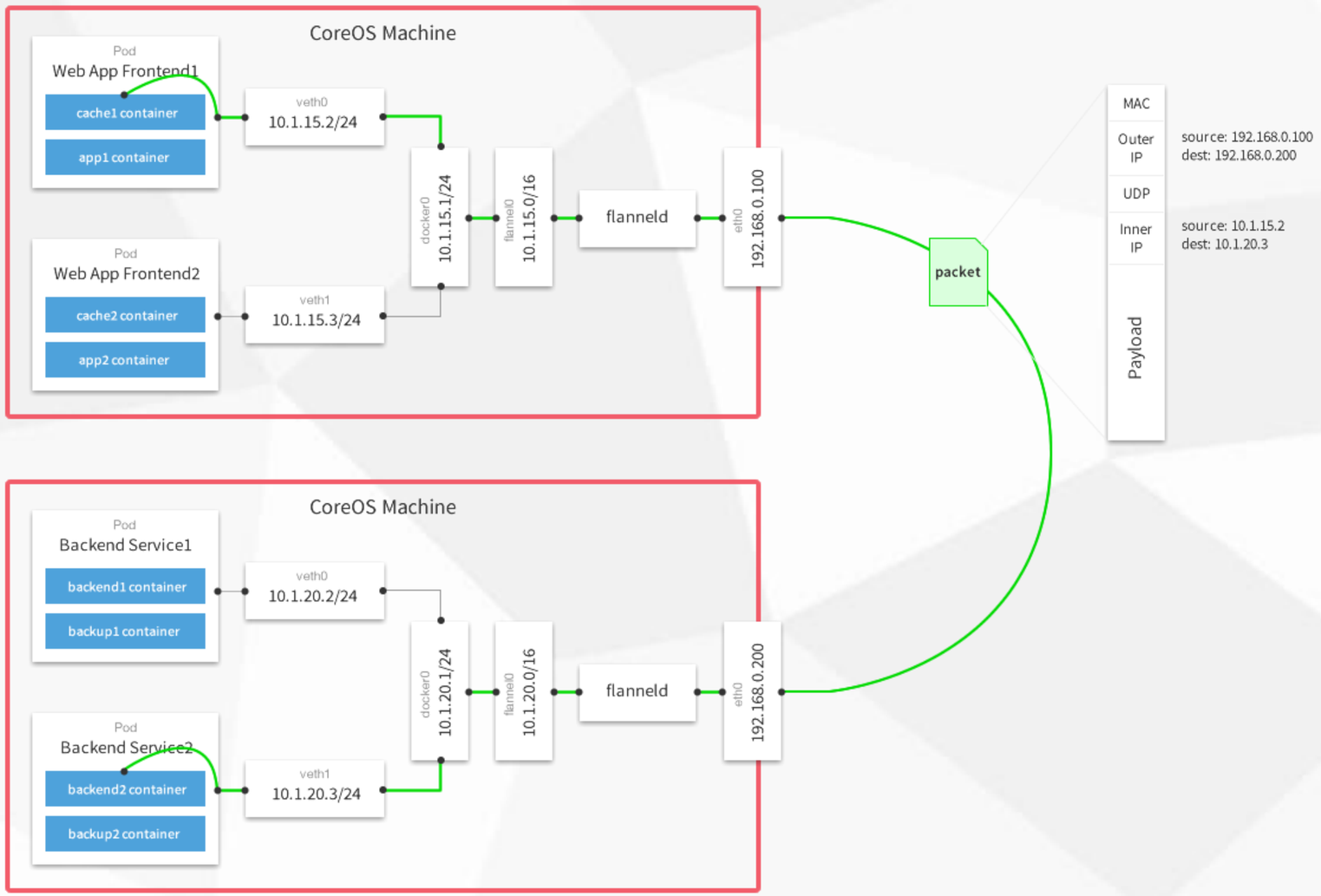
Overlay Network：覆盖网络，在基础网络上叠加的一种虚拟网络技术模式，该网络中的主机通过虚拟链路连接起来。

VXLAN：将源数据包封装到UDP中，并使用基础网络的IP/MAC作为外层报文头进行封装，然后在以太网上传输，到达目的地后由隧道端点解封装并将数据发送给目标地址。

Flannel：是Overlay网络的一种，也是将源数据包封装在另一种网络包里面进行路由转发和通信，目前已经支持UDP、VXLAN、AWS VPC和GCE路由等数据转发方式。



Flannel容器集群网络部署



Flannel容器集群网络部署

1. 写入分配的子网段到etcd，供flanneld使用

```
/opt/etcd/bin/etcdctl \
--ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem \
--endpoints="https://192.168.0.x:2379,https://192.168.0.x:2379,https://192.168.0.x:2379" \
set /coreos.com/network/config  '{ "Network": "172.17.0.0/16", "Backend": {"Type": "vxlan"}}'
```

2. 下载二进制包

https://github.com/coreos/flannel/releases

3. 部署与配置Flannel

4. systemd管理Flannel

5. 配置Docker使用Flannel生成的子网

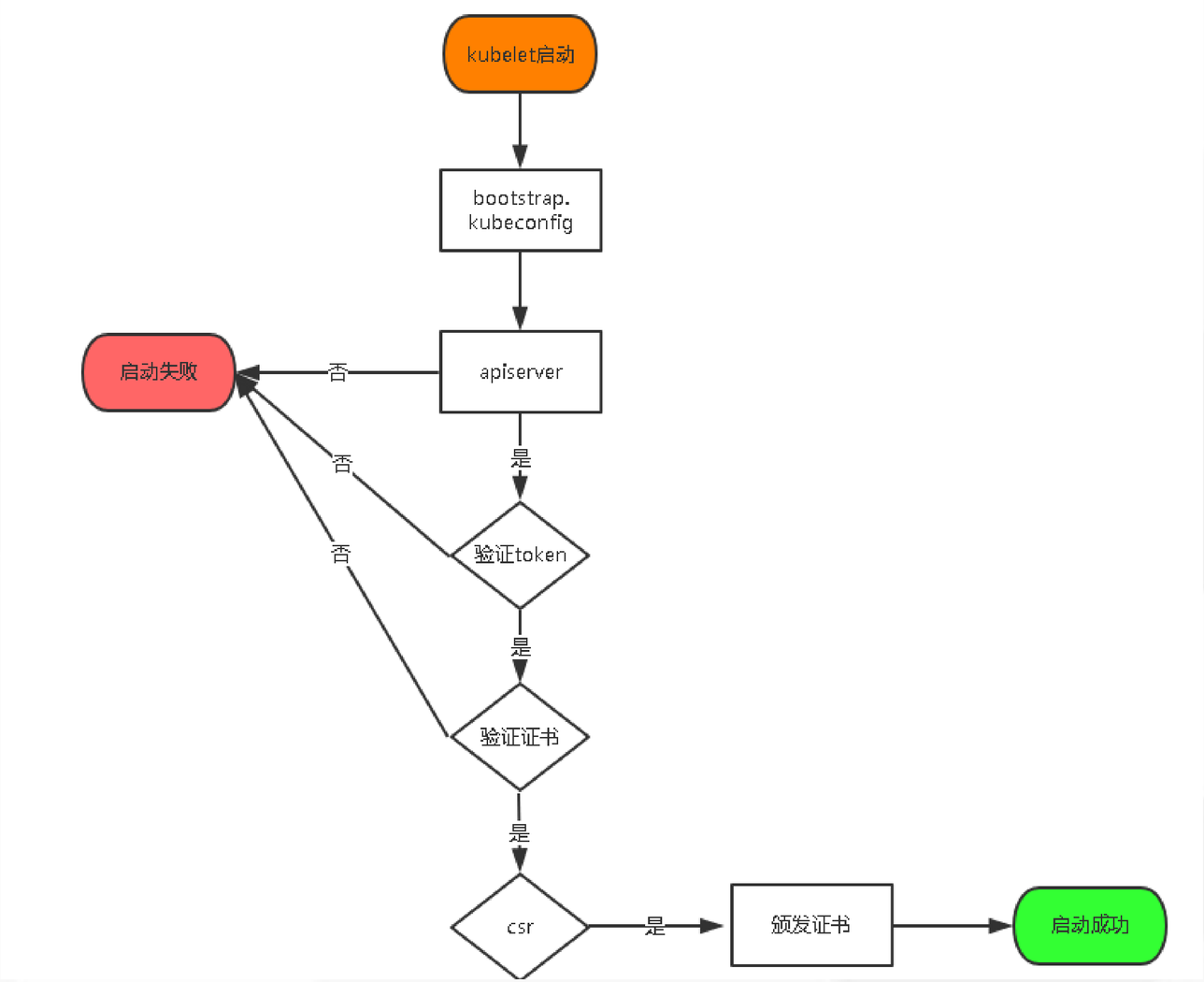
6. 启动Flannel

部署Master组件

1. kube-apiserver
2. kube-controller-manager
3. kube-scheduler

配置文件 -> systemd管理组件 -> 启动

部署Node组件



部署Node组件

1. 将kubelet-bootstrap用户绑定到系统集群角色

```
kubectl create clusterrolebinding kubelet-bootstrap \
--clusterrole=system:node-bootstrapper \
--user=kubelet-bootstrap
```

2. 创建kubeconfig文件

3. 部署kubelet，kube-proxy组件

部署一个测试示例


```
# kubectl run nginx --image=nginx --replicas=3
# kubectl get pod
# kubectl expose deployment nginx --port=88 --target-port=80 --type=NodePort
# kubectl get svc nginx
```

部署Web UI（Dashboard）

https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/dashboard

kubectl与YAML

1. kubectl命令行工具管理集群
2. YAML配置文件管理资源

kubectl命令行管理工具

类型	命令	描述
基础命令	create	通过文件名或标准输入创建资源
	expose	将一个资源公开为一个新的Service
	run	在集群中运行一个特定的镜像
	set	在对象上设置特定的功能
	get	显示一个或多个资源
	explain	文档参考资料
	edit	使用默认的编辑器编辑一个资源。
	delete	通过文件名、标准输入、资源名称或标签选择器来删除资源。
部署命令	rollout	管理资源的发布
	rolling-update	对给定的复制控制器滚动更新
	scale	扩容或缩容Pod数量，Deployment、ReplicaSet、RC或Job
	autoscale	创建一个自动选择扩容或缩容并设置Pod数量
集群管理命令	certificate	修改证书资源
	cluster-info	显示集群信息
	top	显示资源（CPU/Memory/Storage）使用。需要Heapster运行
	cordon	标记节点不可调度
	uncordon	标记节点可调度
	drain	驱逐节点上的应用，准备下线维护
	taint	修改节点taint标记

类型	命令	描述
故障诊断和调试命令	describe	显示特定资源或资源组的详细信息
	logs	在一个Pod中打印一个容器日志。如果Pod只有一个容器，容器名称是可选的
	attach	附加到一个运行的容器
	exec	执行命令到容器
	port-forward	转发一个或多个本地端口到一个pod
	proxy	运行一个proxy到Kubernetes API server
	cp	拷贝文件或目录到容器中
	auth	检查授权
高级命令	apply	通过文件名或标准输入对资源应用配置
	patch	使用补丁修改、更新资源的字段
	replace	通过文件名或标准输入替换一个资源
	convert	不同的API版本之间转换配置文件
设置命令	label	更新资源上的标签
	annotate	更新资源上的注释
	completion	用于实现kubectl工具自动补全
其他命令	api-versions	打印受支持的API版本
	config	修改kubeconfig文件（用于访问API，比如配置认证信息）
	help	所有命令帮助
	plugin	运行一个命令行插件
	version	打印客户端和服务版本信息

kubectl命令行管理工具

1、创建

kubectl run nginx --replicas=3 --image=nginx:1.14 --port=80
kubectl get deploy,pods

2、发布

kubectl expose deployment nginx --port=80 --type=NodePort --target-port=80 --name=nginx-service

kubectl get service

3、更新

kubectl set image deployment/nginx nginx=nginx:1.15

4、回滚

kubectl rollout history deployment/nginx
kubectl rollout undo deployment/nginx

5、删除

kubectl delete deploy/nginx
kubectl delete svc/nginx-service

YAML配置文件管理资源

YAML 是一种简洁的非标记语言。

语法格式：

- 缩进表示层级关系
- 不支持制表符“tab”缩进，使用空格缩进
- 通常开头缩进 2 个空格
- 字符后缩进 1 个空格，如冒号、逗号等
- “---” 表示YAML格式，一个文件的开始
- “#”注释

YAML配置文件管理资源

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15
        ports:
        - containerPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  labels:
    app: nginx
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: nginx
```

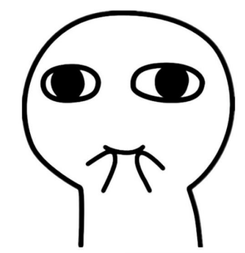
YAML配置文件管理资源

控制器定义

被控制对象

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

YAML配置文件管理资源



- **用run命令生成**

kubectl run --image=nginx my-deploy -o yaml --dry-run > my-deploy.yaml

- **用get命令导出**

kubectl get my-deploy/nginx -o=yaml --export > my-deploy.yaml

- **Pod容器的字段拼写忘记了**

kubectl explain pods.spec.containers

深入理解Pod对象

1. Pod容器分类
2. 镜像拉取策略
3. 资源限制
4. 重启策略
5. 健康检查
6. 调度约束
7. 故障排查

Pod

- 最小部署单元
- 一组容器的集合
- 一个Pod中的容器共享网络命名空间
- Pod是短暂的

Pod容器分类

- **Infrastructure Container**：基础容器
  - 维护整个Pod网络空间
- **InitContainers**：初始化容器
  - 先于业务容器开始执行
- **Containers**：业务容器
  - 并行启动

镜像拉取策略（imagePullPolicy）

- IfNotPresent：默认值，镜像在宿主机上不存在时才拉取
- Always：每次创建 Pod 都会重新拉取一次镜像
- Never： Pod 永远不会主动拉取这个镜像

https://kubernetes.io/docs/concepts/containers/images/

镜像拉取策略（imagePullPolicy）

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: foo
  namespace: awesomeapps
spec:
  containers:
    - name: foo
      image: janedoe/awesomeapp:v1
      imagePullPolicy: IfNotPresent
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: foo
  namespace: awesomeapps
spec:
  containers:
    - name: foo
      image: janedoe/awesomeapp:v1
  imagePullSecrets:
    - name: myregistrykey
```

https://kubernetes.io/docs/concepts/containers/images/

资源限制

Pod和Container的资源请求和限制：

- spec.containers[].resources.limits.cpu
- spec.containers[].resources.limits.memory
- spec.containers[].resources.requests.cpu
- spec.containers[].resources.requests.memory

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/

重启策略（restartPolicy）

- Always：当容器终止退出后，总是重启容器，默认策略。
- OnFailure：当容器异常退出（退出状态码非0）时，才重启容器。
- Never:：当容器终止推出，从不重启容器。

```
apiVersion: v1
kind: Pod
metadata:
  name: foo
  namespace: awesomeapps
spec:
  containers:
    - name: foo
      image: janedoe/awesomeapp:v1
  restartPolicy: Always
```

健康检查（Probe）

Probe有以下两种类型：

- livenessProbe

  如果检查失败，将杀死容器，根据Pod的restartPolicy来操作。

- readinessProbe

  如果检查失败，Kubernetes会把Pod从service endpoints中剔除。

Probe支持以下三种检查方法：

- httpGet

  发送HTTP请求，返回200-400范围状态码为成功。

- exec

  执行Shell命令返回状态码是0为成功。

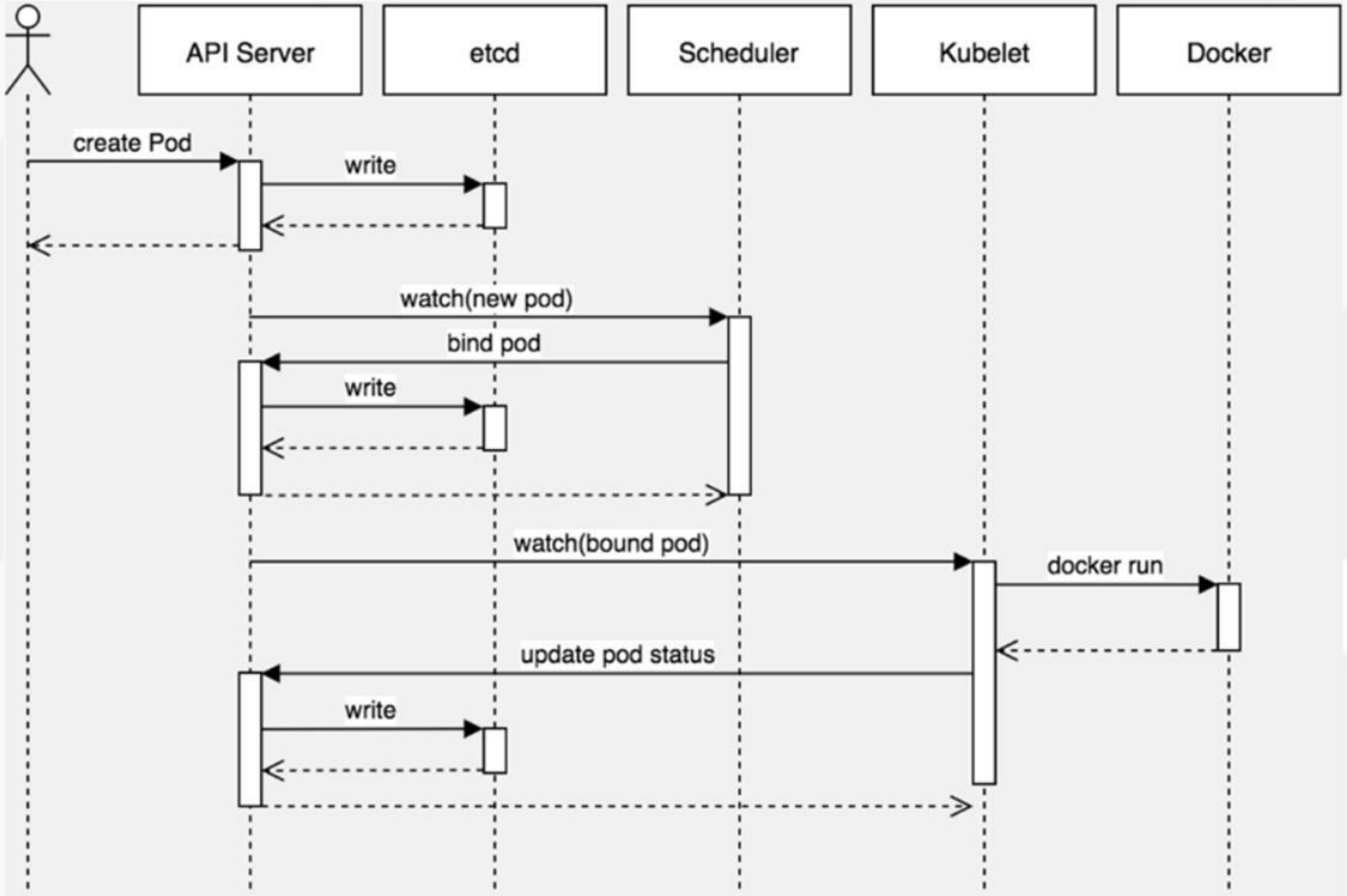
- tcpSocket

  发起TCP Socket建立成功。

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/

调度约束


API Server
etcd
Scheduler
Kubelet
Docker
create Pod
write
watch(new pod)
bind pod
write
watch(bound pod)
docker run
update pod status
write

调度约束

- nodeName用于将Pod调度到指定的Node名称上
- nodeSelector用于将Pod调度到匹配Label的Node上

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
  labels:
    app: nginx
spec:
  nodeName: 192.168.31.65
  containers:
  - name: nginx
    image: nginx:1.15
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  nodeSelector:
    env_role: dev
  containers:
  - name: nginx
    image: nginx:1.15
```

故障排查

值	描述
Pending	Pod创建已经提交到Kubernetes。但是，因为某种原因而不能顺利创建。例如下载镜像慢，调度不成功。
Running	Pod已经绑定到一个节点，并且已经创建了所有容器。至少有一个容器正在运行中，或正在启动或重新启动。
Succeeded	Pod中的所有容器都已成功终止，不会重新启动。
Failed	Pod的所有容器均已终止，且至少有一个容器已在故障中终止。也就是说，容器要么以非零状态退出，要么被系统终止。
Unknown	由于某种原因apiserver无法获得Pod的状态，通常是由于Master与Pod所在主机kubelet通信时出错。

```
kubectl describe TYPE/NAME
kubectl logs TYPE/NAME [-c CONTAINER]
kubectl exec POD [-c CONTAINER] -- COMMAND [args...]
```

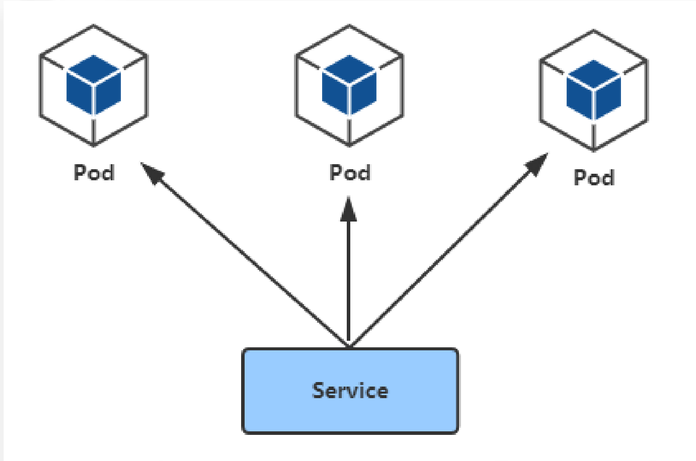
https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/

Service

- 防止Pod失联
- 定义一组Pod的访问策略
- 支持ClusterIP，NodePort以及LoadBalancer三种类型
- Service的底层实现主要有Iptables和IPVS二种网络模式

Pod与Service的关系

- 通过label-selector相关联
- 通过Service实现Pod的负载均衡（ TCP/UDP 4层）



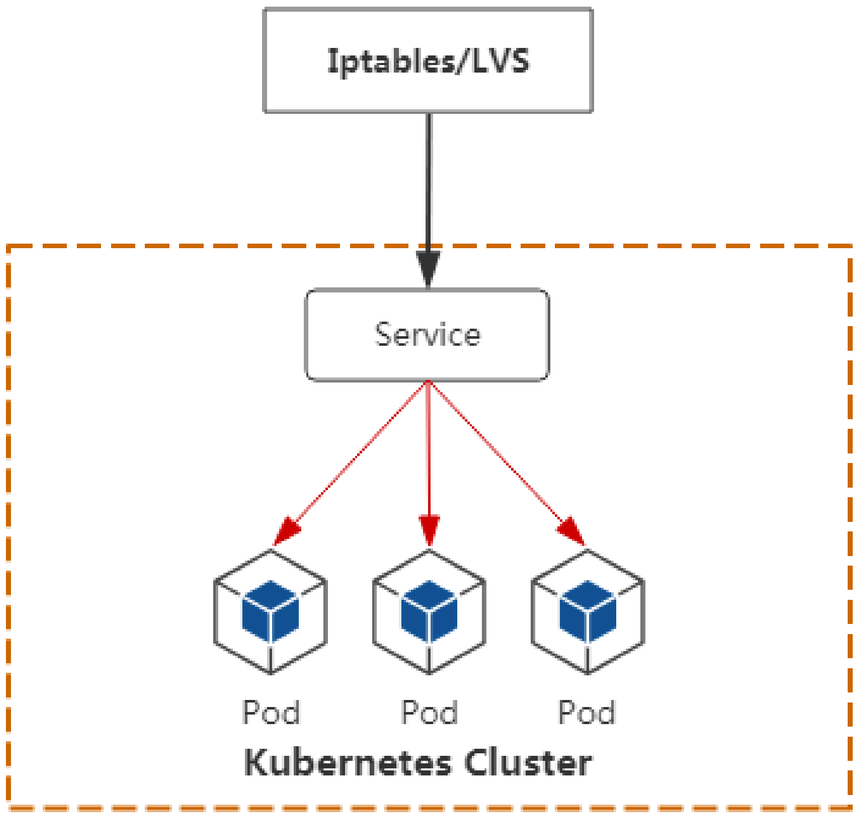
labels:
  app: nginx

selector:
  app: nginx

Service 类型

- ClusterIP：默认，分配一个集群内部可以访问的虚拟IP（VIP）
- NodePort：在每个Node上分配一个端口作为外部访问入口
- LoadBalancer：工作在特定的Cloud Provider上，例如Google Cloud，AWS，OpenStack

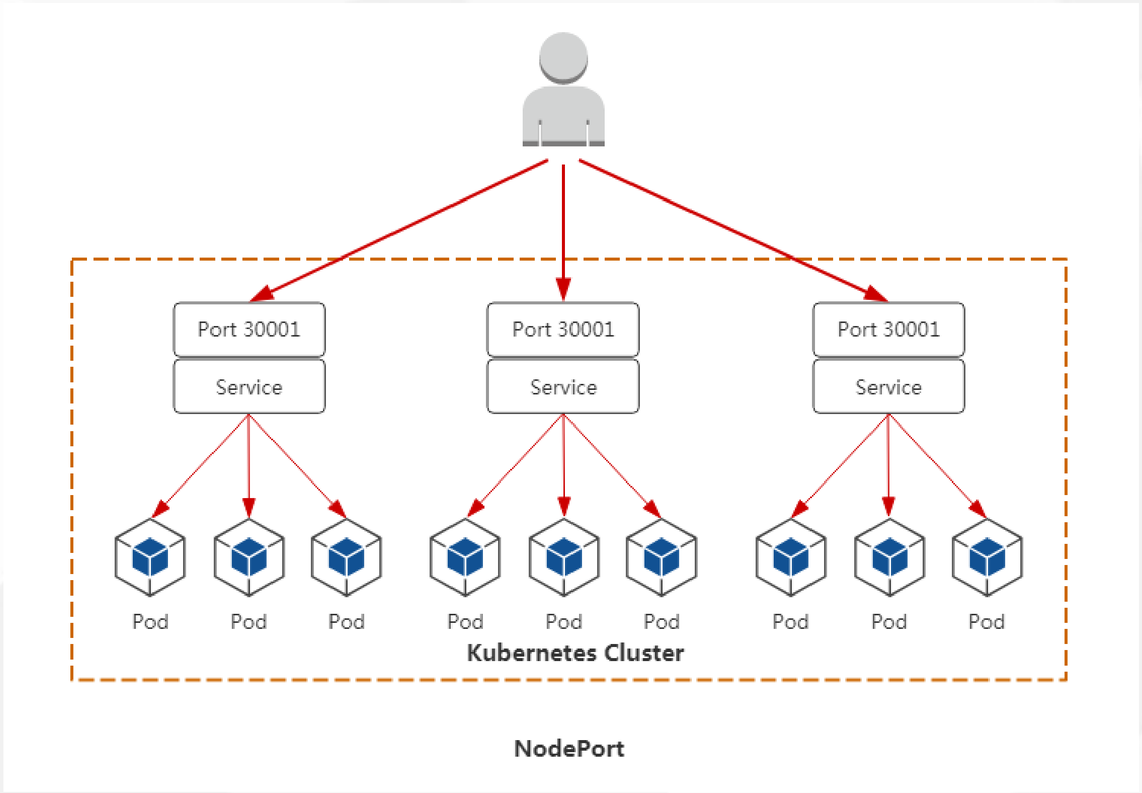
Service 类型



ClusterIP

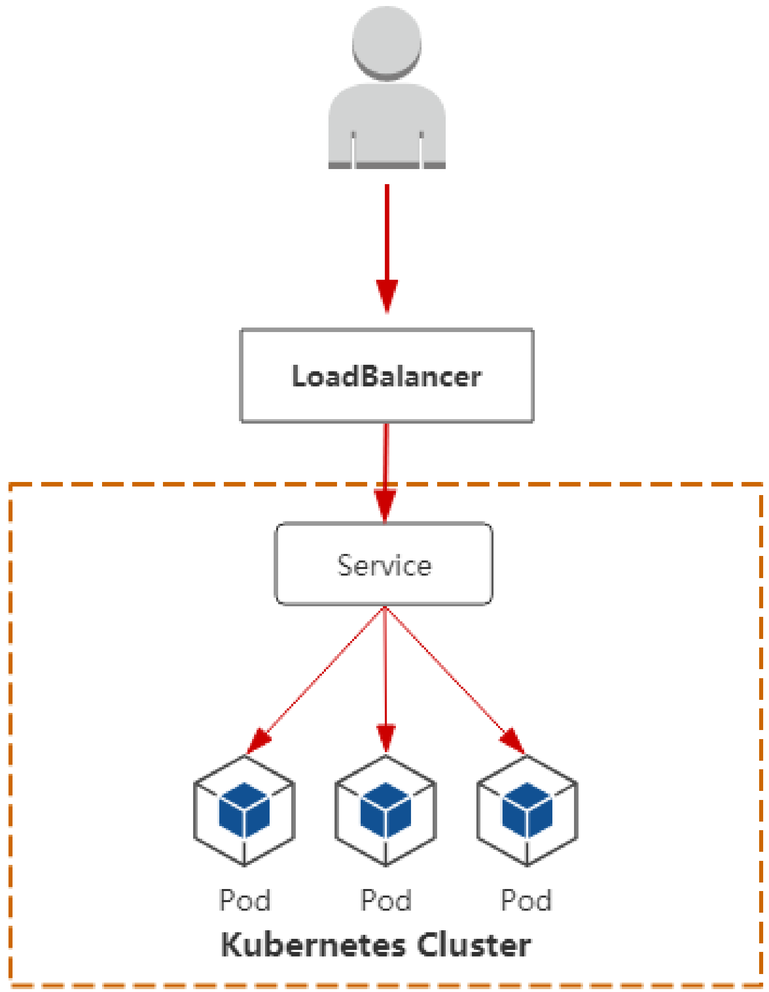
```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: A
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Service 类型




```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: A
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
      nodePort: 30001
  type: NodePort
```

Service 类型



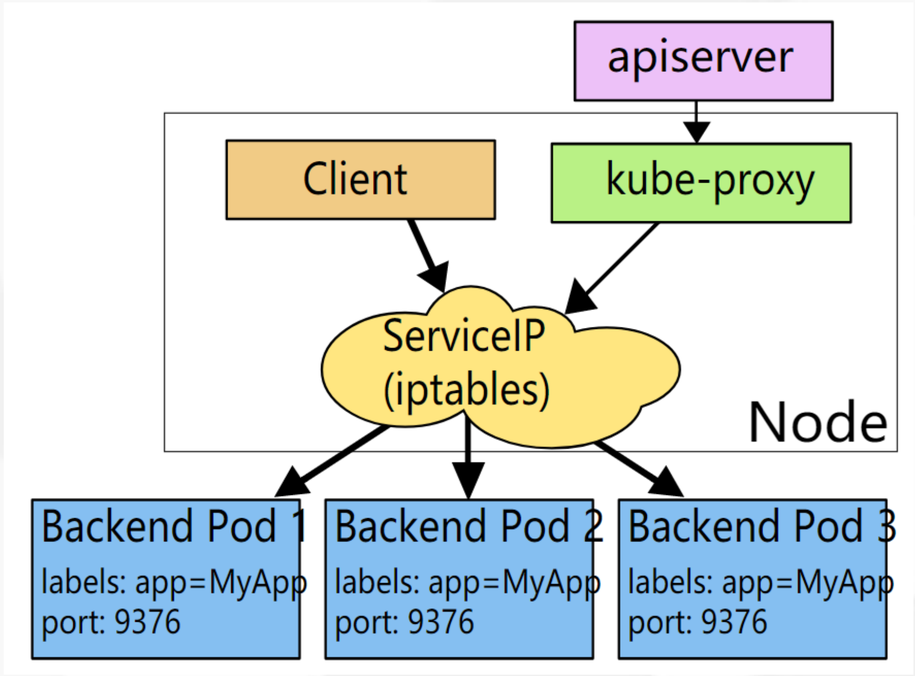
LoadBalancer

Service 代理模式

底层流量转发与负载均衡实现：

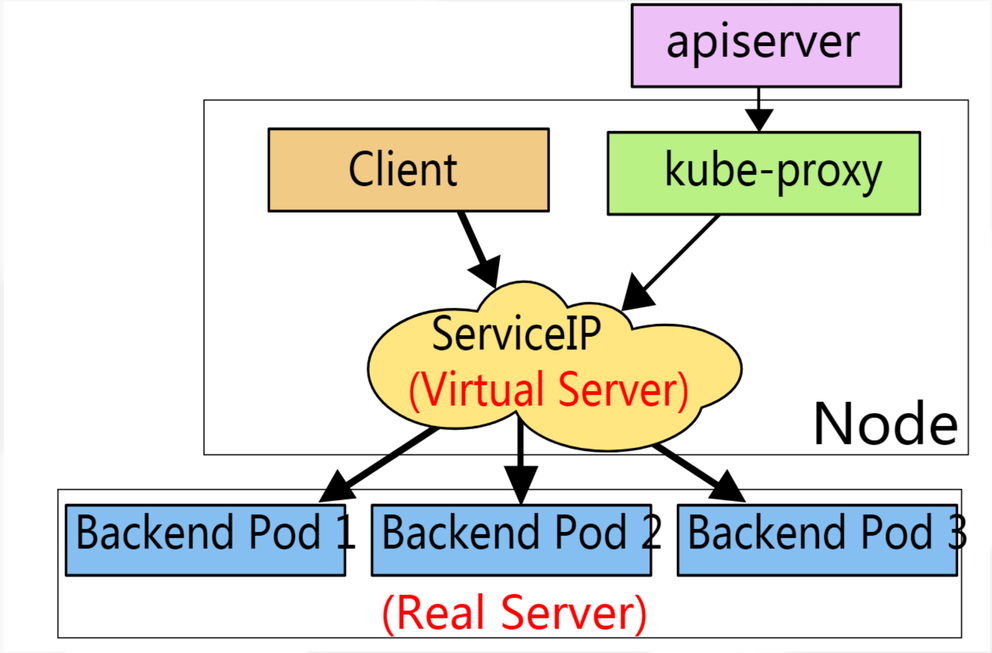
- Iptables
- IPVS

Service 代理模式



Iptables

Service 代理模式



IPVS

Service 代理模式

Iptables VS IPVS

Iptables：

- 灵活，功能强大（可以在数据包不同阶段对包进行操作）
- 规则遍历匹配和更新，呈线性时延

IPVS：

- 工作在内核态，有更好的性能
- 调度算法丰富：rr，wrr，lc，wlc，ip hash...

DNS

DNS服务监视Kubernetes API，为每一个Service创建DNS记录用于域名解析。

ClusterIP A记录格式：<service-name>.<namespace-name>.svc.cluster.local
示例：my-svc.my-namespace.svc.cluster.local

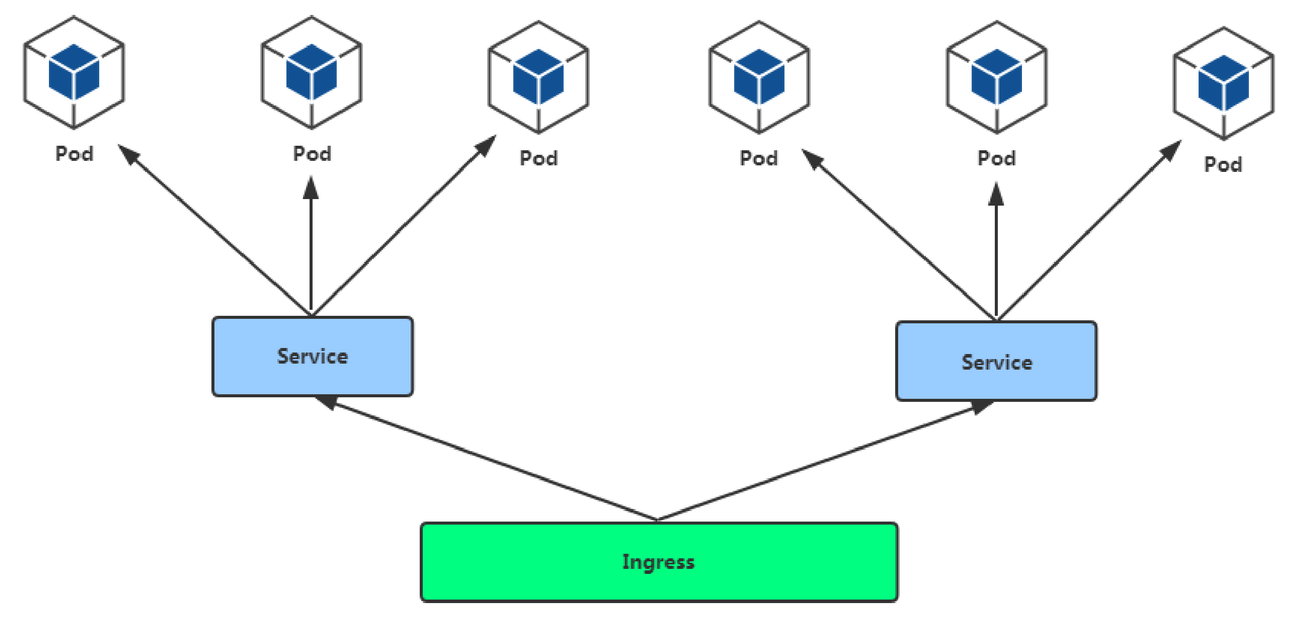
https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/dns/coredns

Ingress

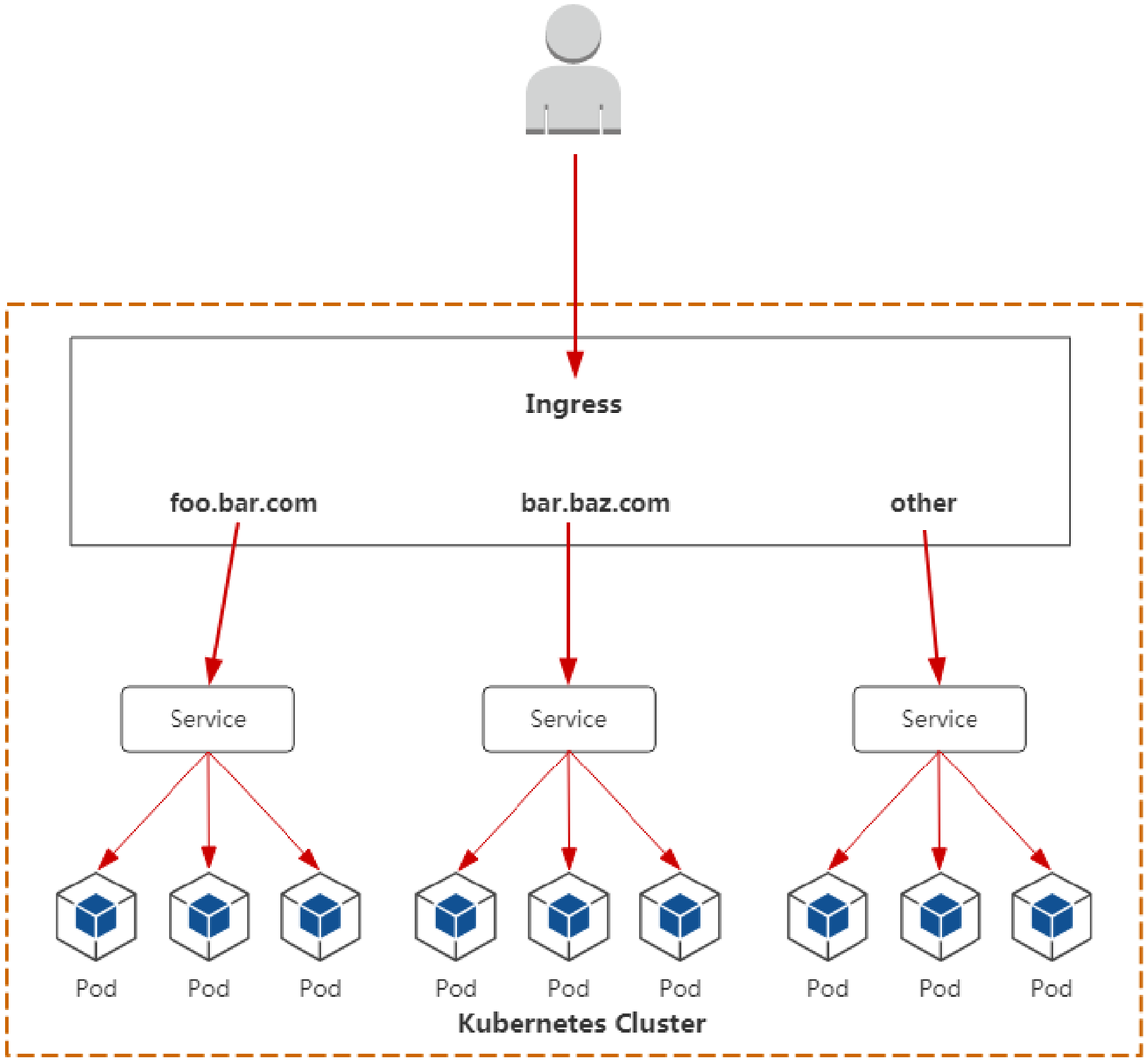
1. Pod与Ingress的关系
2. Ingress Controller
3. Ingress（HTTP与HTTPS）

Pod与Ingress的关系

- 通过service相关联
- 通过Ingress Controller实现Pod的负载均衡
  - 支持TCP/UDP 4层和HTTP 7层



Ingress Controller



Ingress Controller

部署文档：https://github.com/kubernetes/ingress-nginx/blob/master/docs/deploy/index.md

注意事项：

- 镜像地址修改成国内的：lizhenliang/nginx-ingress-controller:0.20.0
- 使用宿主机网络：hostNetwork: true

其他控制器：

- Contour is an Envoy based ingress controller provided and supported by Heptio.
- F5 Networks provides support and maintenance for the F5 BIG-IP Controller for Kubernetes.
- HAProxy based ingress controller jcmoraisjr/haproxy-ingress which is mentioned on the blog post HAProxy Ingress Controller for Kubernetes. HAProxy Technologies offers support and maintenance for HAProxy Enterprise and the ingress controller jcmoraisjr/haproxy-ingress.
- Istio based ingress controller Control Ingress Traffic.
- Kong offers community or commercial support and maintenance for the Kong Ingress Controllerfor Kubernetes.
- NGINX, Inc. offers support and maintenance for the NGINX Ingress Controller for Kubernetes.
- Traefik is a fully featured ingress controller (Let's Encrypt, secrets, http2, websocket), and it also comes with commercial support by Containous.

https://kubernetes.io/docs/concepts/services-networking/ingress/

Ingress


```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
spec:
  rules:
  - host: example.foo.com
    http:
      paths:
      - backend:
          serviceName: service1
          servicePort: 80
```

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: tls-example-ingress
spec:
  tls:
  - hosts:
    - sslexample.foo.com
    secretName: testsecret-tls
  rules:
    - host: sslexample.foo.com
      http:
        paths:
        - path: /
          backend:
            serviceName: service1
            servicePort: 80
```

Volume & PersistentVolume

1. Volume
2. PersistentVolume
3. PersistentVolume 动态供给

Volume

- Kubernetes中的Volume提供了在容器中挂载外部存储的能力
- Pod需要设置卷来源（spec.volume）和挂载点（spec.containers.volumeMounts）两个信息后才可以使用相应的Volume

https://kubernetes.io/docs/concepts/storage/volumes/

emptyDir

创建一个空卷，挂载到Pod中的容器。Pod删除该卷也会被删除。

应用场景：Pod中容器之间数据共享

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: write
    image: busybox
    command: ["bash","-c","for i in {1..100};do echo
$i >> /data/hello;sleep 1;done"]
    volumeMounts:
      - name: data
        mountPath: /data

  - name: read
    image: centos
    command: ["bash","-c","tail -f /data/hello"]
    volumeMounts:
      - name: data
        mountPath: /data

  volumes:
  - name: data
    emptyDir: {}
```

hostPath

挂载Node文件系统上文件或者目录到Pod中的容器。

应用场景：Pod中容器需要访问宿主机文件

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - /bin/sh
    - -c
    - sleep 36000
    volumeMounts:
    - name: data
      mountPath: /data
  volumes:
  - name: data
    hostPath:
      path: /tmp
      type: Directory
```

NFS


```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
        - name: wwwroot
          mountPath: /usr/share/nginx/html
        ports:
        - containerPort: 80
      volumes:
      - name: wwwroot
        nfs:
          server: 192.168.0.200
          path: /data/nfs
```

将公司项目部署到Kubernetes平台中

1. 准备工作与注意事项
2. 准备基础镜像并推送到镜像仓库
3. 部署PHP/Java项目

Kubernetes集群资源监控

1. Kubernetes监控指标
2. Kubernetes监控方案
3. Heapster+InfluxDB+Grafana

Kubernetes监控指标

集群监控

- 节点资源利用率
- 节点数
- 运行Pods

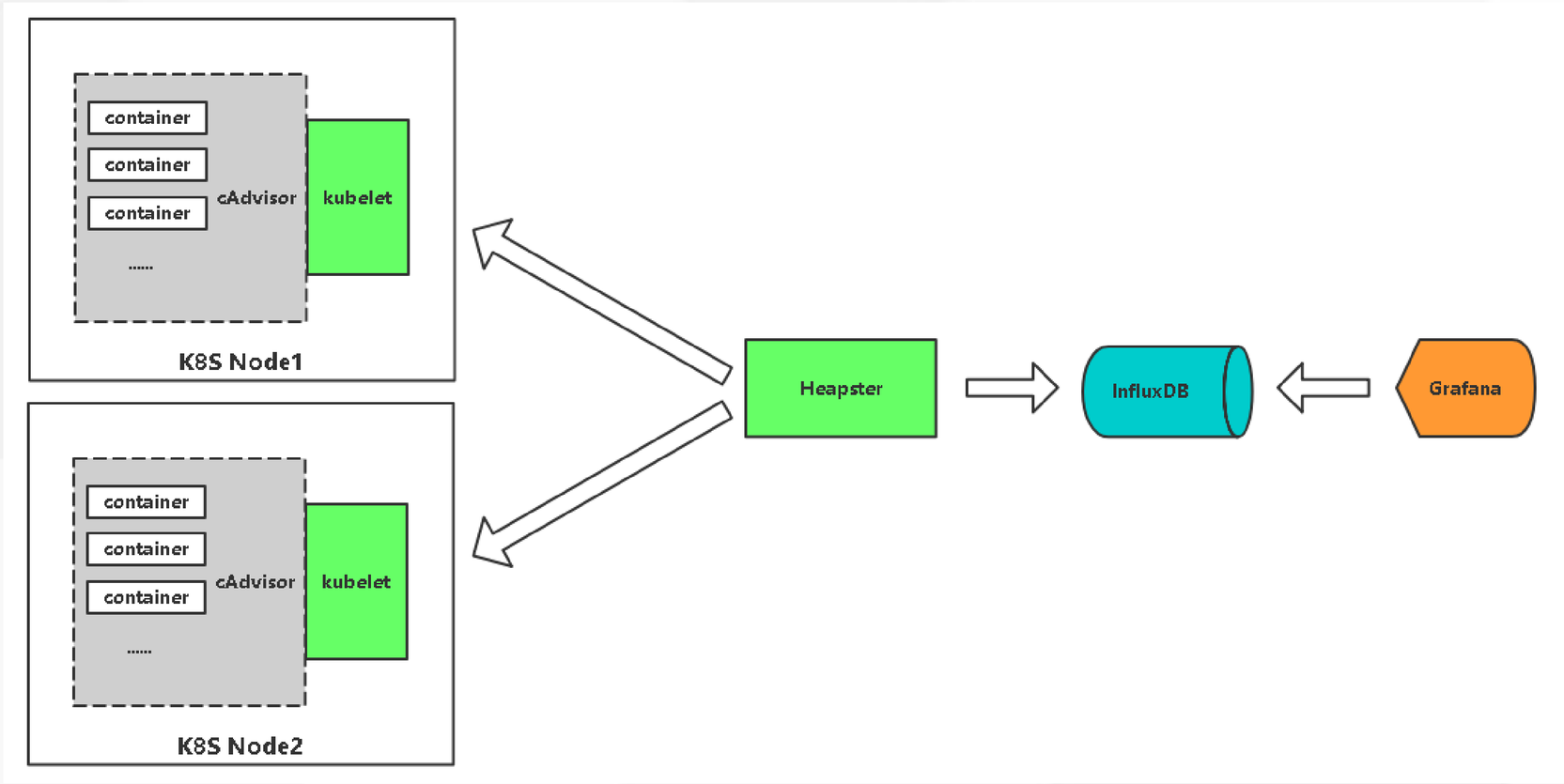
Pod监控

- Kubernetes指标
- 容器指标
- 应用程序

Kubernetes监控方案

监控方案	特点	适用
Zabbix	大量定制工作	大部分的互联网公司
open-falcon	功能模块分解比较细，显得更复杂	系统和应用监控
cAdvisor+InfluxDB+Grafana	简单	容器监控
cAdvisor/exporter+Prometheus+Grafana	扩展性好	容器，应用，主机全方面监控

Heapster+InfluxDB+Grafana



Kubernetes平台中日志收集

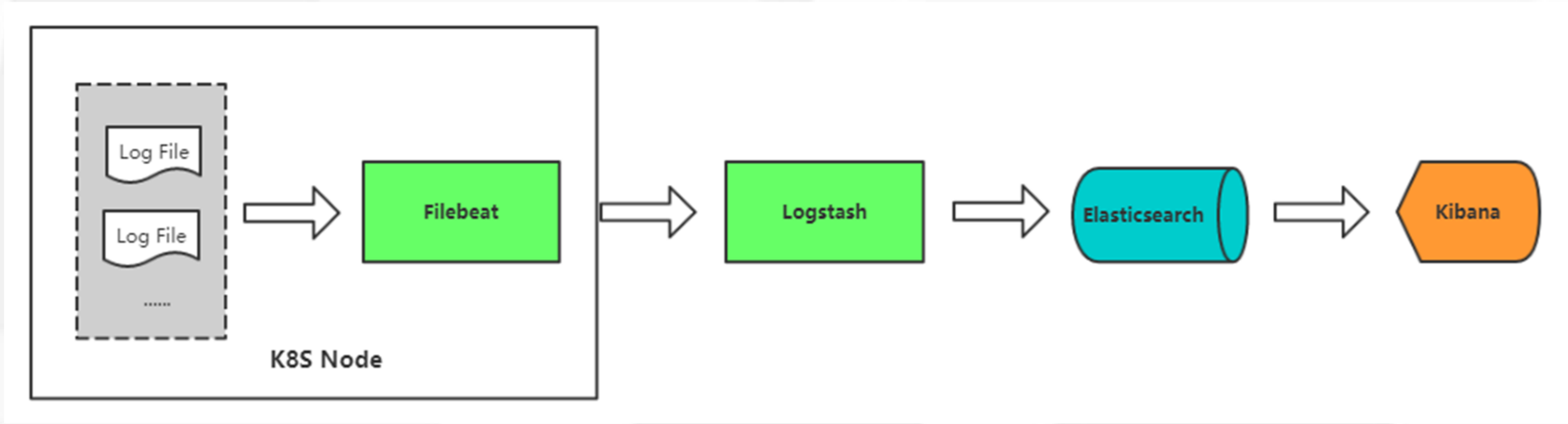
1. 收集哪些日志
2. 日志方案
3. 容器中的日志怎么收集

收集哪些日志

- K8S系统的组件日志
- K8S Cluster里面部署的应用程序日志

日志方案

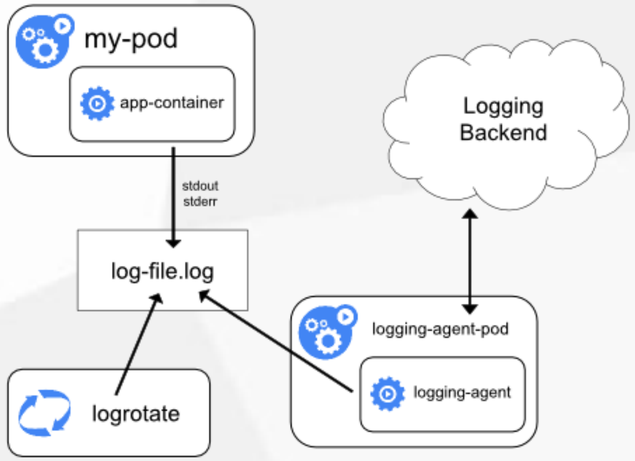
Filebeat+ELK



容器中的日志怎么收集

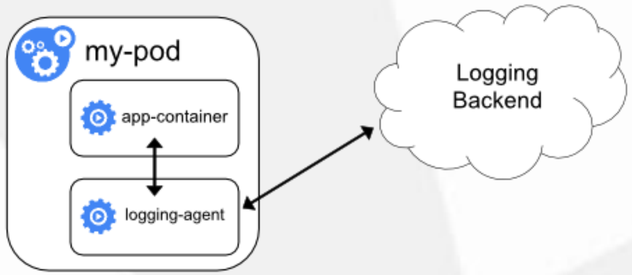
方案一：Node上部署一个日志收集程序

- DaemonSet方式部署日志收集程序
- 对本节点/var/log和 /var/lib/docker/containers/ 两个目录下的日志进行采集



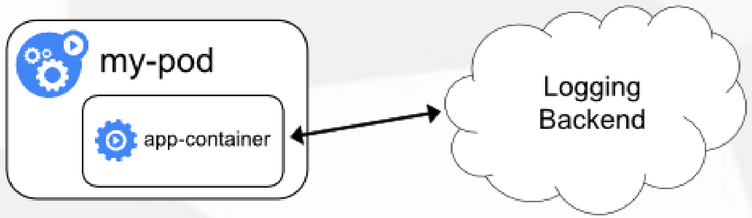
方案二：Pod中附加专用日志收集的容器

- 每个运行应用程序的Pod中增加一个日志收集容器，使用emtyDir共享日志目录让日志收集程序读取到。



方案三：应用程序直接推送日志

- 超出Kubernetes范围



容器中的日志怎么收集

方式	优点	缺点
方案一：Node上部署一个日志收集程序	每个Node仅需部署一个日志收集程序，资源消耗少，对应用无侵入	应用程序日志需要写到标准输出和标准错误输出，不支持多行日志
方案二：Pod中附加专用日志收集的容器	低耦合	每个Pod启动一个日志收集代理，增加资源消耗，并增加运维维护成本
方案三：应用程序直接推送日志	无需额外收集工具	浸入应用，增加应用复杂度

方案二：Pod中附加专用日志收集的容器


```
containers:
 - name: web
   image: reg.example.com/project/web:1.1
   ports:
   - containerPort: 8080
   volumeMounts:
   - name: tomcat-catalina
     mountPath: /usr/local/tomcat/logs

 - name: filebeat
   image: reg.example.com/ops/filebeat:6.4.1
   args: [
     "-c", "/etc/filebeat.yml",
     "-e",
   ]
   volumeMounts:
   - name: filebeat-config
     mountPath: /etc/filebeat.yml
     subPath: filebeat.yml
   - name: tomcat-catalina
     mountPath: /usr/local/tomcat/logs

 volumes:
 - name: tomcat-catalina
   emptyDir: {}
 - name: filebeat-config
   configMap:
     name: filebeat-config
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: filebeat-config

data:
  filebeat.yml: |-
    filebeat.prospectors:
      - type: log
        paths:
          - /usr/local/tomcat/logs/catalina*.log
        fields:
          app: tomcat
          type: project-catalina
        fields_under_root: true
        multiline:
          pattern: '^\['
          negate: true
          match: after

    output.redis:
      hosts: ["10.213.94.202"]
      password: "elk"
      key: "filebeat"
      db: 0
      datatype: list
```

谢谢