从Docker到Kubernetes企业应用实战

个人介绍

讲师：李振良

资深运维工程师，曾混在IDC，大数据，金融行业。下到搬服务器，上到Linux平台架构设计。经重重磨练，具备各方综合能力。

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

DevOps技术栈

专注于分享DevOps工具链及经验总结。

微信扫一扫

Docker/K8s技术学员群：397834690

目 录

Kubernetes介绍

Kubernetes是Google在2014年6月开源的一个容器集群管理系统，使用Go语言开发，Kubernetes也叫K8S。

K8S是Google内部一个叫Borg的容器集群管理系统衍生出来的，Borg已经在Google大规模生产运行十年之久。

K8S主要用于自动化部署、扩展和管理容器应用，提供了资源调度、部署管理、服务发现、扩容缩容、监控等一整套功能。

2015年7月，Kubernetes v1.0正式发布，截止到2018年1月27日最新稳定版本是v1.9.2。

Kubernetes目标是让部署容器化应用简单高效。

官方网站：www.kubernetes.io

Kubernetes介绍

Kubernetes主要功能：

- **数据卷**

  Pod中容器之间共享数据，可以使用数据卷。

- **应用程序健康检查**

  容器内服务可能进程堵塞无法处理请求，可以设置监控检查策略保证应用健壮性。

- **复制应用程序实例**

  控制器维护着Pod副本数量，保证一个Pod或一组同类的Pod数量始终可用。

- **弹性伸缩**

  根据设定的指标（CPU利用率）自动缩放Pod副本数。

- **服务发现**

  使用环境变量或DNS服务插件保证容器中程序发现Pod入口访问地址。

- **负载均衡**

  一组Pod副本分配一个私有的集群IP地址，负载均衡转发请求到后端容器。在集群内部其他Pod可通过这个ClusterIP访问应用。

- **滚动更新**

  更新服务不中断，一次更新一个Pod，而不是同时删除整个服务。

- **服务编排**

  通过文件描述部署服务，使得应用程序部署变得更高效。

- **资源监控**

  Node节点组件集成cAdvisor资源收集工具，可通过Heapster汇总整个集群节点资源数据，然后存储到InfluxDB时序数据库，再由Grafana展示。

- **提供认证和授权**

  支持角色访问控制（RBAC）认证授权等策略。

基本对象概念

基本对象：

- Pod

  Pod是最小部署单元，一个Pod有一个或多个容器组成，Pod中容器共享存储和网络，在同一台Docker主机上运行。

- Service

  Service一个应用服务抽象，定义了Pod逻辑集合和访问这个Pod集合的策略。

  Service代理Pod集合对外表现是为一个访问入口，分配一个集群IP地址，来自这个IP的请求将负载均衡转发后端Pod中的容器。

  Service通过Lable Selector选择一组Pod提供服务。

- Volume

  数据卷，共享Pod中容器使用的数据。

- Namespace

  命名空间将对象逻辑上分配到不同Namespace，可以是不同的项目、用户等区分管理，并设定控制策略，从而实现多租户。

  命名空间也称为虚拟集群。

- Lable

  标签用于区分对象（比如Pod、Service），键/值对存在；每个对象可以有多个标签，通过标签关联对象。

基本对象概念

基于基本对象更高层次抽象：

- ReplicaSet

  下一代Replication Controller。确保任何给定时间指定的Pod副本数量，并提供声明式更新等功能。

  RC与RS唯一区别就是lable selector支持不同，RS支持新的基于集合的标签，RC仅支持基于等式的标签。

- Deployment

  Deployment是一个更高层次的API对象，它管理ReplicaSets和Pod，并提供声明式更新等功能。

  官方建议使用Deployment管理ReplicaSets，而不是直接使用ReplicaSets，这就意味着可能永远不需要直接操作ReplicaSet对象。

- StatefulSet

  StatefulSet适合持久性的应用程序，有唯一的网络标识符（IP），持久存储，有序的部署、扩展、删除和滚动更新。

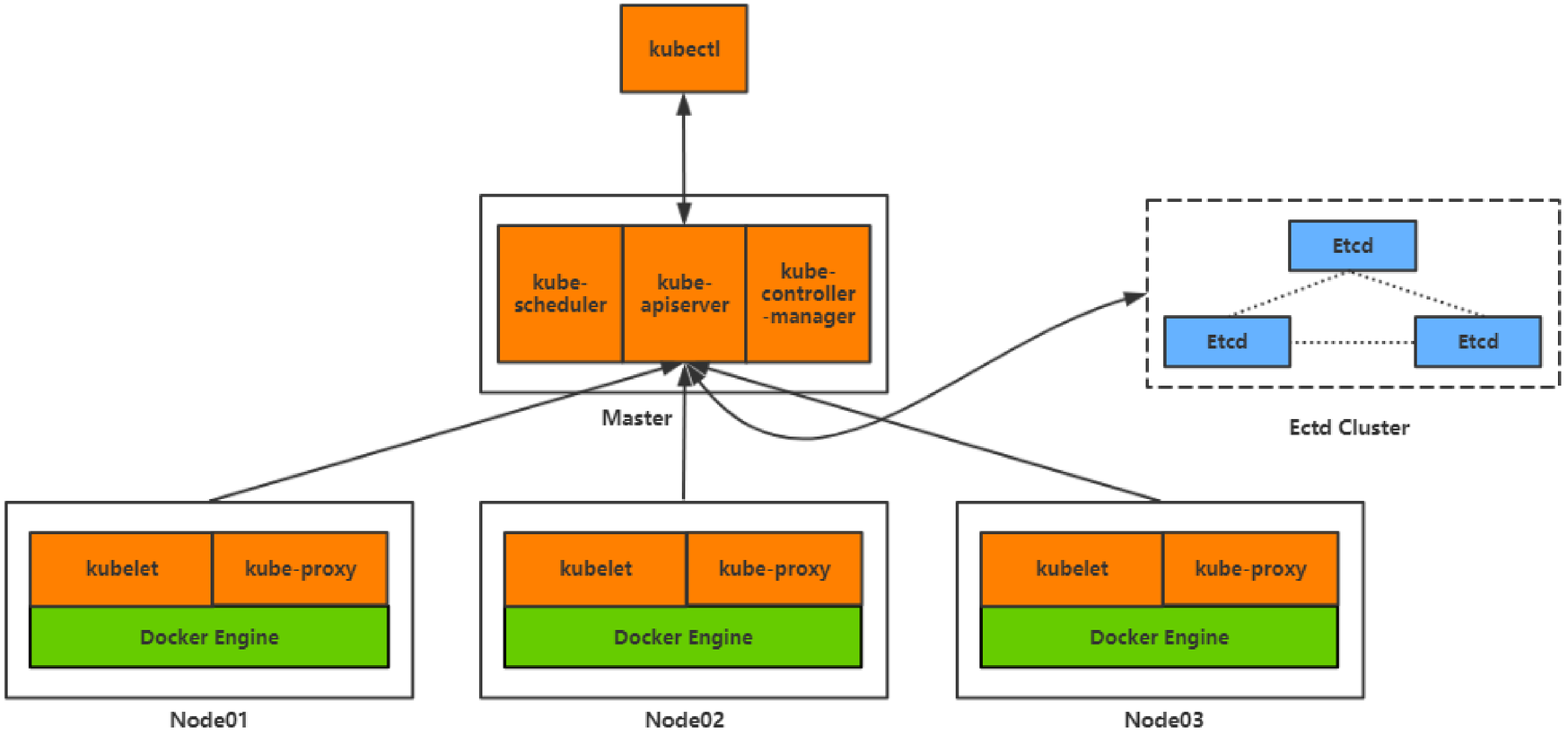
- DaemonSet

  DaemonSet确保所有（或一些）节点运行同一个Pod。当节点加入Kubernetes集群中，Pod会被调度到该节点上运行，当节点从集群中移除时，DaemonSet的Pod会被删除。删除DaemonSet会清理它所有创建的Pod。

- Job

  一次性任务，运行完成后Pod销毁，不再重新启动新容器。还可以任务定时运行。

系统架构及组件功能



系统架构及组件功能

Master组件：

- **kube-apiserver**

  Kubernetes API，集群的统一入口，各组件协调者，以HTTP 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。

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/rkt**

  运行容器。

第三方服务：

- **etcd**

  分布式键值存储系统。用于保持集群状态，比如Pod、Service等对象信息。

集群部署

1、环境规划

2、安装Docker

3、自签TLS证书

4、部署Etcd集群

5、部署Flannel网络

6、创建Node节点kubeconfig文件

7、获取K8S二进制包

8、运行Master组件

9、运行Node组件

10、查询集群状态

11、启动一个测试示例

12、部署Web UI (Dashboard)

集群部署 – 环境规划

软件	版本
Linux操作系统	CentOS7.4_x64
Kubernetes	1.9
Docker	17.12-ce
Etcd	3.0

关闭selinux

角色	IP	组件	推荐配置
master	192.168.0.211	kube-apiserver kube-controller-manager kube-scheduler etcd	CPU 2核+ 2G内存+
node01	192.168.0.212	kubelet kube-proxy docker flannel etcd	
node02	192.168.0.213	kubelet kube-proxy docker flannel etcd	

集群部署 – 安装Docker

```
# yum install -y yum-utils device-mapper-persistent-data lvm2
# yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
# yum install docker-ce
# cat << EOF > /etc/docker/daemon.json
{
  "registry-mirrors": [ "https://registry.docker-cn.com"],
   "insecure-registries":["192.168.0.210:5000"]
}
EOF
# systemctl start docker
# systemctl enable docker
```

集群部署 – 自签TLS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

安装证书生成工具cfssl：

```
wget https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
wget https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64
chmod +x cfssl_linux-amd64 cfssljson_linux-amd64 cfssl-certinfo_linux-amd64
mv cfssl_linux-amd64 /usr/local/bin/cfssl
mv cfssljson_linux-amd64 /usr/local/bin/cfssljson
mv cfssl-certinfo_linux-amd64 /usr/bin/cfssl-certinfo
```

集群部署 – 部署Etcd集群

二进制包下载地址：https://github.com/coreos/etcd/releases/tag/v3.2.12

查看集群状态：

```
# /opt/kubernetes/bin/etcdctl \
--ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem \
--endpoints="https://192.168.0.211:2379,https://192.168.0.212:2379,https://192.168.0.213:2379" \
cluster-health
```

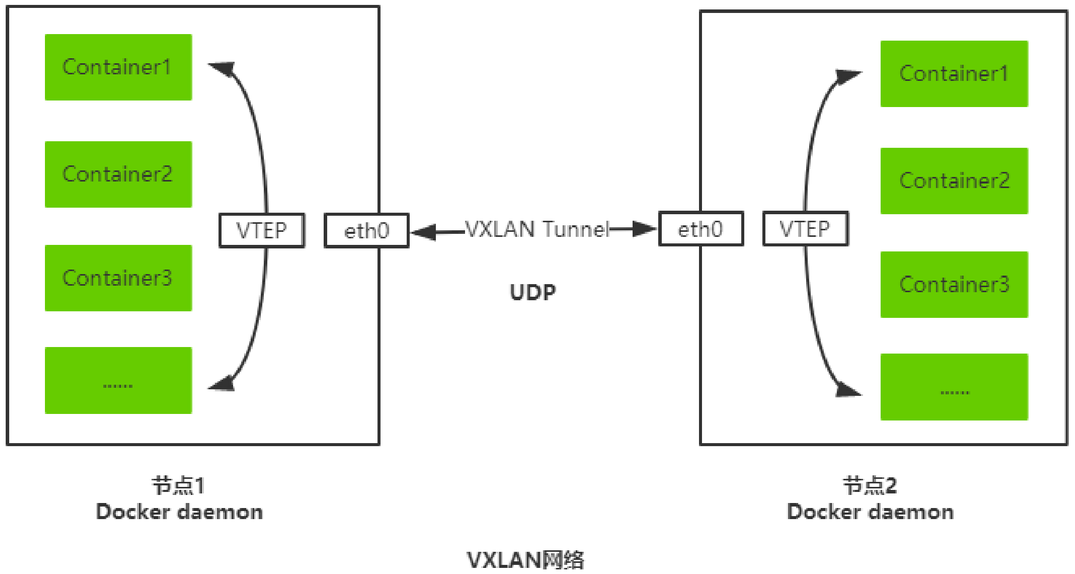
集群部署 – 部署Flannel网络

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。

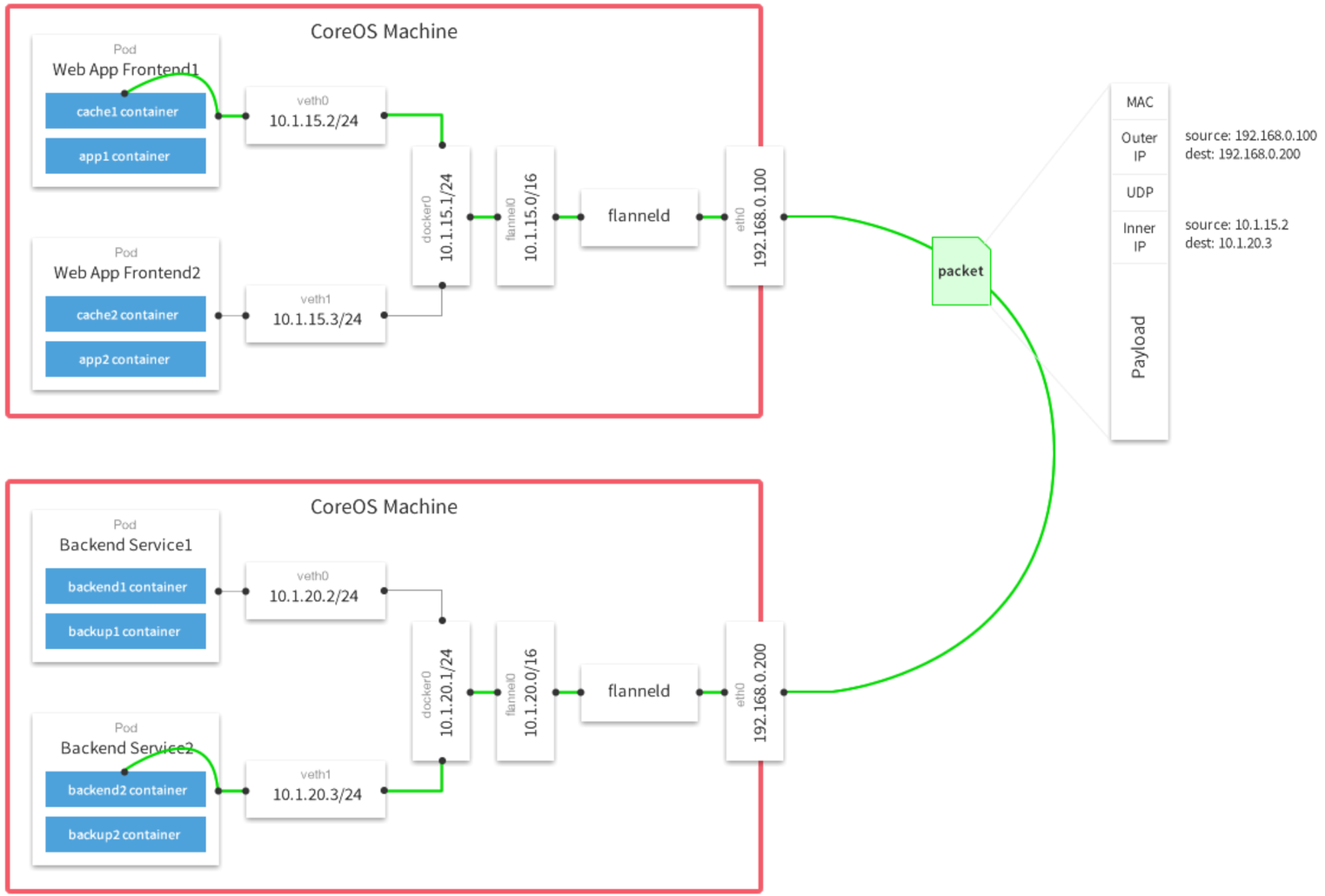
VXLAN：将源数据包封装到UDP中，并使用基础网络的IP/MAC作为外层报文头进行封装，然后在以太网上传输，到达目的地后由隧道端点解封装并将数据发送给目标地址。

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。

多主机容器网络通信其他主流方案：隧道方案（ Weave、OpenvSwitch ），路由方案（Calico）等。



集群部署 – 部署Flannel网络



集群部署 – 部署Flannel网络

1）写入分配的子网段到etcd，供flanneld使用

```
# /opt/kubernetes/bin/etcdctl \
--ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem \
--endpoints="https://192.168.0.211:2379,https://192.168.0.212:2379,https://192.168.0.213:2379" \
set /coreos.com/network/config  '{ "Network": "172.17.0.0/16", "Backend": {"Type": "vxlan"}}'
```

2）下载二进制包

```
# wget https://github.com/coreos/flannel/releases/download/v0.9.1/flannel-v0.9.1-linux-amd64.tar.gz
```

3）配置Flannel

4）systemd管理Flannel

5）配置Docker启动指定子网段

6）启动

集群部署 – 创建Node节点kubeconfig文件

1、创建TLS Bootstrapping Token

2、创建kubelet kubeconfig

3、创建kube-proxy kubeconfig

集群部署 – 获取K8S二进制包

https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG-1.9.md

Kubernetes容器集群管理

集群部署 – 运行Master组件

```
# mkdir -p /opt/kubernetes/{bin,cfg,ssl}
# mv kube-apiserver kube-controller-manager kube-scheduler kubectl /opt/kubernetes/bin
# chmod +x /opt/kubernetes/bin/* && chmod +x *.sh
# cp ssl/ca*pem ssl/server*pem /opt/kubernetes/ssl/
# cp ssl/token.csv /opt/kubernetes/cfg/
# ./apiserver.sh 192.168.0.211 https://192.168.0.211:2379,https://192.168.0.212:2379,https://192.168.0.213:2379
# ./scheduler.sh 127.0.0.1
# ./controller-manager.sh 127.0.0.1
# echo "export PATH=$PATH:/opt/kubernetes/bin" >> /etc/profile
# source /etc/profile
# iptables -I INPUT -s 192.168.0.0/24 -j ACCEPT
```

集群部署 – 运行Node组件

```
# mv kubelet kube-proxy /opt/kubernetes/bin
# chmod +x /opt/kubernetes/bin/* && chmod +x *.sh
# ./kubelet.sh 192.168.0.212 10.10.10.2
# ./proxy.sh 192.168.0.212
```

集群部署 – 查询集群状态

```
# kubectl get node
# kubectl get componentstatus
```

集群部署 – 启动一个测试示例

```
# 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
```

集群部署 – 部署Web UI (Dashboard)

```
# kubectl create -f .
# kubectl create -f dashboard-rbac.yaml
# kubectl create -f dashboard-deployment.yaml
# kubectl create -f dashboard-service.yaml
```

Kubernetes容器集群管理

Kubectl管理工具

```
# 设置集群项中名为kubernetes的apiserver地址与根证书
kubectl config set-cluster kubernetes --server=https://192.168.1.195:6443 --certificate-authority=ca.pem
# 设置用户项中cluster-admin用户证书认证字段
kubectl config set-credentials cluster-admin --certificate-authority=ca.pem --client-key=admin-key.pem --client-certificate=admin.pem
# 设置环境项中名为default的默认集群和用户
kubectl config set-context default --cluster=kubernetes --user=cluster-admin
# 设置默认环境项为default
kubectl config use-context default
```

Kubectl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	维护期间排除节点
	taint	

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

Kubernetes容器集群管理

Kubectl管理工具

1、创建

```
kubectl run nginx --replicas=3 --labels="app=example" --image=nginx:1.10  --port=80
```

2、查看

```
kubectl get deploy
kubectl get pods --show-labels
kubectl get pods -l app=example
kubectl get pods -o wide
```

3、发布

```
kubectl expose deployment nginx --port=88 --type=NodePort --target-port=80 --name=nginx-service

kubectl describe service nginx-service
```

4、故障排查

```
kubectl describe TYPE NAME_PREFIX
kubectl logs nginx-xxx
kubectl exec  -it nginx-xxx bash
```

5、更新

```
kubectl set image deployment/nginx nginx=nginx:1.11
or
kubectl edit deployment/nginx
```

资源发布管理：

```
kubectl rollout status deployment/nginx
kubectl rollout history deployment/nginx
kubectl rollout history deployment/nginx --revision=3

kubectl scale deployment nginx --replicas=10
```

6、回滚

```
kubectl rollout undo deployment/nginx-deployment
kubectl rollout undo deployment/nginx-deployment --to-revision=3
```

7、删除

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

YAML配置文件管理资源

配置文件说明：

- 定义配置时，指定最新稳定版API（当前为v1）；
- 配置文件应该存储在集群之外的版本控制仓库中。如果需要，可以快速回滚配置、重新创建和恢复；
- 应该使用YAML格式编写配置文件，而不是JSON。尽管这些格式都可以使用，但YAML对用户更加友好；
- 可以将相关对象组合成单个文件，通常会更容易管理；
- 不要没必要的指定默认值，简单和最小配置减少错误；
- 在注释中说明一个对象描述更好维护。

Kubernetes容器集群管理

YAML配置文件管理资源

deployment：

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.10
        ports:
        - containerPort: 80
```

service：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  labels:
    app: nginx
spec:
  ports:
  - port: 88
    targetPort: 80
  selector:
    app: nginx
```

Kubernetes容器集群管理

Pod管理

- 创建/查询/更新/删除
- 资源限制
- 调度约束
- 重启策略
- 健康检查
- 问题定位

Pod管理 – 创建/查询/更新/删除

创建Pod对象：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
     app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
```

基本管理：

```
# 创建pod资源
kubectl create -f pod.yaml
# 查看pods
kubectl get pods nginx-pod
# 查看pod描述
kubectl describe pod nginx-pod
# 更新资源
kubectl apply -f pod.yaml
# 删除资源
kubectl delete pod nginx-pod
```

Pod管理 – 资源限制

示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
     app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

Pod管理 – 调度约束

Pod.spec.nodeName	强制约束Pod调度到指定Node节点上
Pod.spec.nodeSelector	通过lable-selector机制选择节点

示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
     app: nginx
spec:
  # nodeName: node01
  nodeSelector:
    env_role: dev
  containers:
  - name: nginx
    image: nginx
```

Pod管理 – 重启策略

三种重启策略：

Always：当容器停止，总是重建容器，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：当容器终止退出，从不重启容器。

示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
     app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
  restartPolicy: OnFailure
```

Pod管理 – 健康检查

提供Probe机制，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。

示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    livenessProbe:
      httpGet:
        path: /index.html
        port: 80
```

Pod管理 – 问题定位

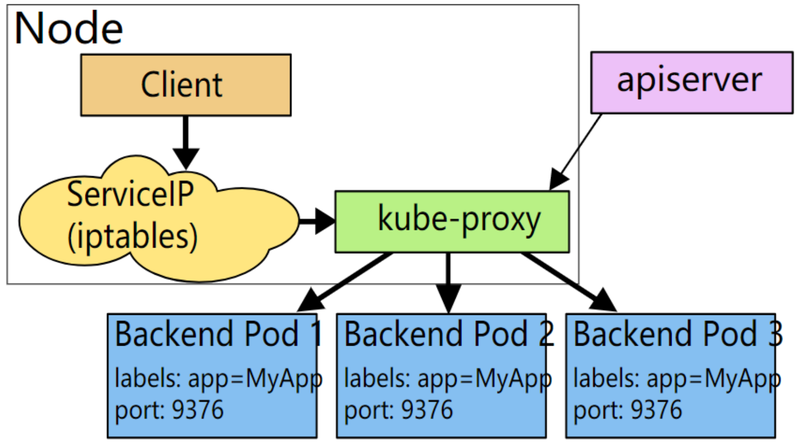
```
kubectl describe TYPE NAME_PREFIX
kubectl logs nginx-xxx
kubectl exec  -it nginx-xxx bash
```

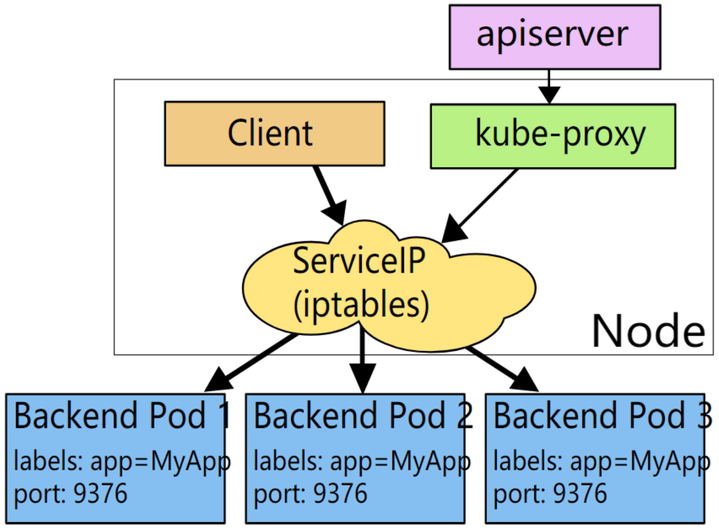
Service

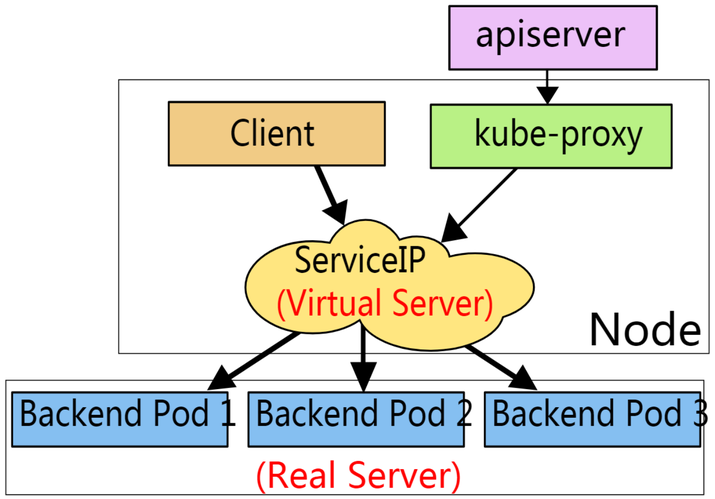
- 网络代理模式
- 服务代理
- 服务发现
- 发布服务

Service – 网络代理模式

三种代理模式：userspace、iptables和ipvs
官方文档：https://kubernetes.io/docs/concepts/services-networking/service







Service – 服务代理

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 9376
  - name: https
    protocol: TCP
    port: 443
    targetPort: 9377
```

Service – 服务发现

服务发现支持Service环境变量和DNS两种模式：

- **环境变量**

  当一个Pod运行到Node，kubelet会为每个容器添加一组环境变量，Pod容器中程序就可以使用这些环境变量发现Service。

  环境变量名格式如下：

  {SVCNAME}_SERVICE_HOST

  {SVCNAME}_SERVICE_PORT

  其中服务名和端口名转为大写，连字符转换为下划线。

  限制：

  1）Pod和Service的创建顺序是有要求的，Service必须在Pod创建之前被创建，否则环境变量不会设置到Pod中。

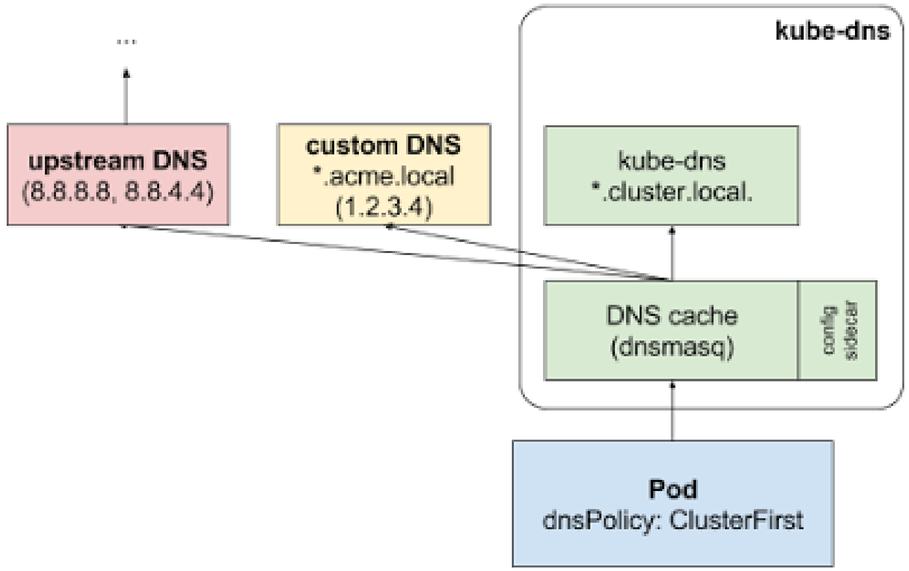
  2）Pod只能获取同Namespace中的Service环境变量。

- **DNS**

  DNS服务监视Kubernetes API，为每一个Service创建DNS记录用于域名解析。这样Pod中就可以通过DNS域名获取Service的访问地址。

Kubernetes容器集群管理

Service – 服务发现




```
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  namespace: default
spec:
  containers:
  - image: busybox
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
    name: busybox
  restartPolicy: Always
```

测试解析：

```
# kubectl exec -ti busybox -- nslookup kubernetes.default
```

Service – 发布服务

服务类型：

- ClusterIP

  分配一个内部集群IP地址，只能在集群内部访问（同Namespace内的Pod），默认ServiceType。

- NodePort

  分配一个内部集群IP地址，并在每个节点上启用一个端口来暴露服务，可以在集群外部访问。

  访问地址：<NodeIP>:<NodePort>

- LoadBalancer

  分配一个内部集群IP地址，并在每个节点上启用一个端口来暴露服务。

  除此之外，Kubernetes会请求底层云平台上的负载均衡器，将每个Node（[NodeIP]:[NodePort]）作为后端添加进去。

Service – 发布服务

NodePort示例：

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
        role: web
    spec:
      containers:
      - name: nginx
        image: nginx:1.10
        ports:
        - containerPort: 80
```

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  labels:
    app: nginx
spec:
  selector:
    app: nginx
    role: web
  ports:
  - name: http
    port: 8080
    protocol: TCP
    targetPort: 80
  type: NodePort
```

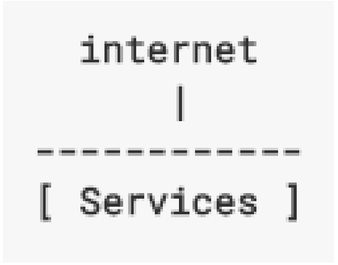
Kubernetes容器集群管理

Service – 发布服务

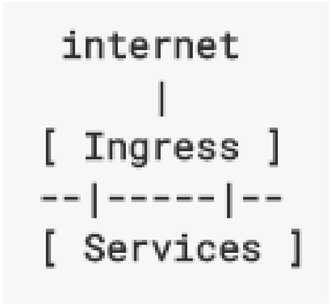
Ingress

- Ingress介绍
- 部署Default Backend
- 部署Ingress Controller
- 部署Ingress
- 部署Ingress TLS

Ingress – 介绍



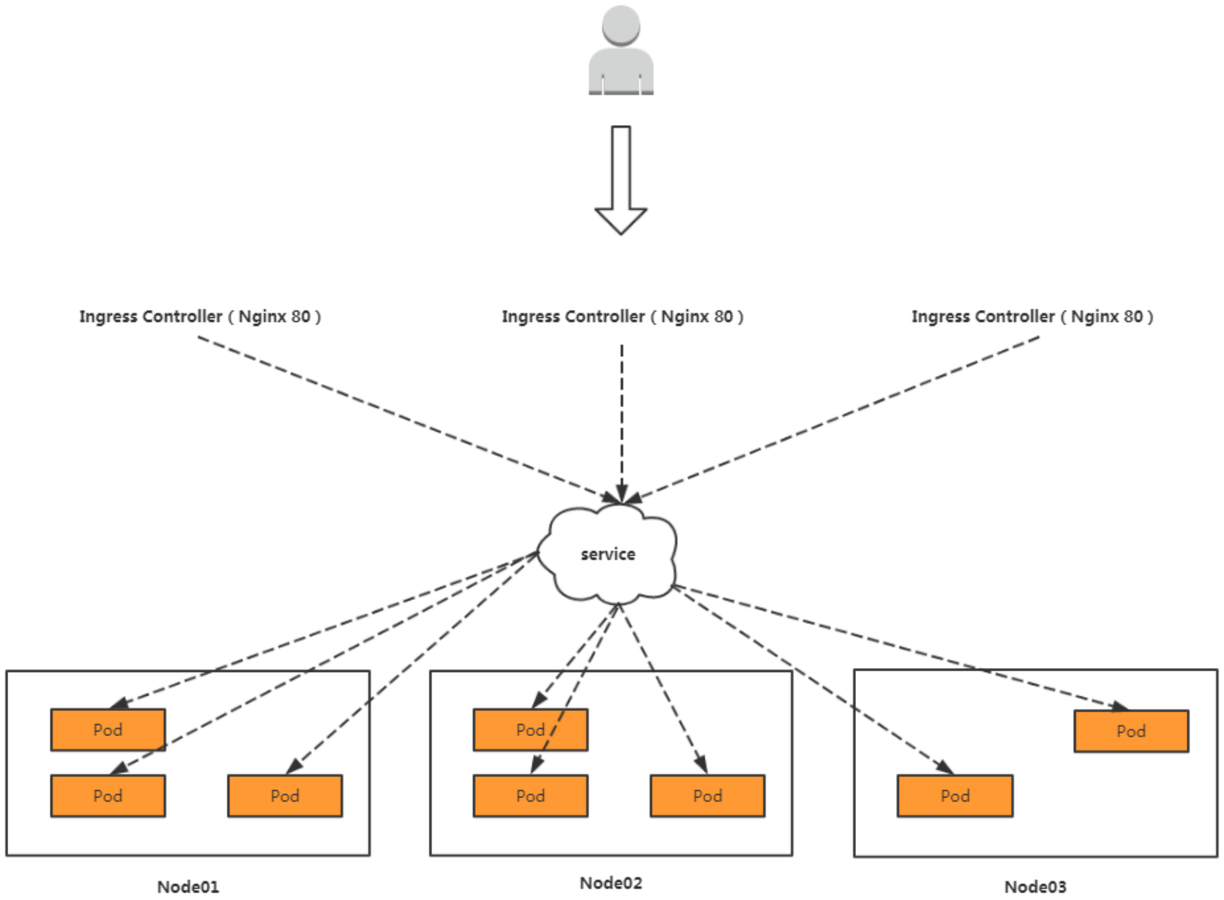
service发布服务



Ingress发布服务

- Ingress Examples
- Dummy controller backend
- HAProxy Ingress controller
- Linkerd
- traefik
- AWS Application Load Balancer Ingress Controller
- kube-ingress-aws-controller
- Voyager: HAProxy Ingress Controller

Ingress – 介绍



- Nginx
- Ingress Controller
- Ingress

Ingress – 部署

```
curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/namespace.yaml \
    | kubectl apply -f -

curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/default-backend.yaml \
    | kubectl apply -f -

curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/tcp-services-configmap.yaml \
    | kubectl apply -f -

curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/udp-services-configmap.yaml \
    | kubectl apply -f -

curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/rbac.yaml \
    | kubectl apply -f -

curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/with-rbac.yaml \
    | kubectl apply -f  -
```

注意：需修改国内镜像下载地址

官方文档：https://github.com/kubernetes/ingress-nginx/tree/master/deploy

Ingress – 部署

Ingress测试：

```
kubectl run --image=nginx nginx --replicas=2
kubectl expose deployment nginx --port=80

kubectl run --image=httpd httpd --replicas=2
kubectl expose deployment httpd --port=80

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: http-test
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: nginx
          servicePort: 80
  - host: bar.baz.com
    http:
      paths:
      - backend:
          serviceName: httpd
          servicePort: 80
```

Ingress TLS测试：

```
kubectl create secret tls ingress-secret --key aliangedu-key.pem --cert aliangedu.pem

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: https-test
spec:
  tls:
  - hosts:
    - www.aliangedu.com
    secretName: ingress-secret
  rules:
  - host: www.aliangedu.com
    http:
      paths:
      - backend:
          serviceName: nginx
          servicePort: 88
```

Volume

- emptyDir
- hostPath
- nfs
- glusterfs

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

Volume – emptyDir

```
apiVersion: v1
kind: Pod
metadata:
  name: redis-pod
spec:
  containers:
  - image: redis
    name: redis
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

Volume – hostPath

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: nginx
    name: test-container
    volumeMounts:
    - mountPath: /tmp-test
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      path: /tmp
      type: Directory
```

Volume – nfs

```
apiVersion: extensions/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: /opt/wwwroot
```

Volume – glusterfs

GlusterFS部署：http://docs.gluster.org/en/latest/Quick-Start-Guide/Quickstart/

https://github.com/kubernetes/kubernetes/tree/8fd414537b5143ab039cb910590237cabf4af783/examples/volumes/glusterfs

```
# kubectl create -f glusterfs-endpoints.json
# kubectl create -f glusterfs-service.json
```

PersistentVolumes

PersistentVolume（PV，持久卷）：对存储抽象实现，使得存储作为集群中的资源。

PersistentVolumeClaim（PVC，持久卷申请）：PVC消费PV的资源。

Pod申请PVC作为卷来使用，集群通过PVC查找绑定的PV，并Mount给Pod。

PersistentVolumes - PV

PV类型：

- GCEPersistentDisk
- AWSElasticBlockStore
- AzureFile
- AzureDisk
- FC (Fibre Channel)
- FlexVolume
- Flocker
- NFS
- iSCSI
- RBD (Ceph Block Device)
- CephFS
- Cinder (OpenStack block storage)
- Glusterfs
- VsphereVolume
- Quobyte Volumes
- HostPath
- VMware Photon
- Portworx Volumes
- ScaleIO Volumes
- StorageOS

PersistentVolumes - PV

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /opt/nfs/data
    server: 192.168.0.200
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gluster-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  glusterfs:
    endpoints: "glusterfs-cluster"
    path: "gv0"
    readOnly: false
```

accessModes：

- ReadWriteOnce
- ReadOnlyMany
- ReadWriteMany

Recycling Policy：

- Retain
- Recycle
- Delete

Phase：

- Available
- Bound
- Released
- Failed

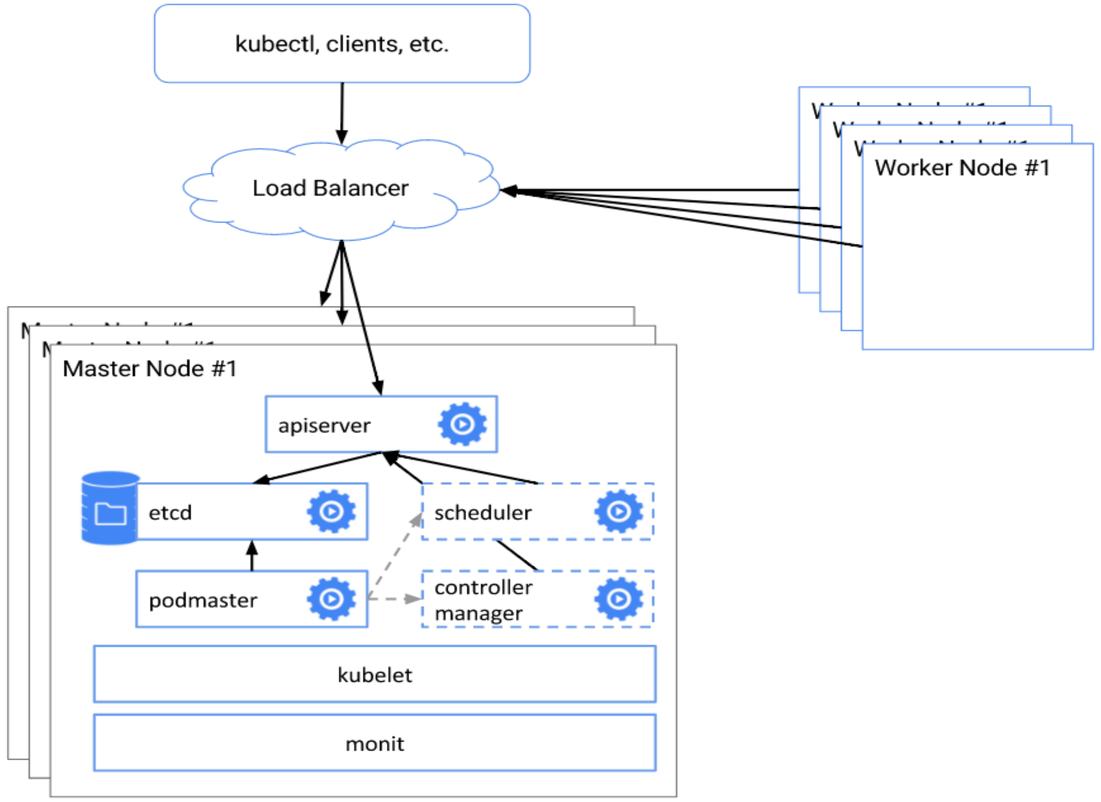
PersistentVolumes - PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc001
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

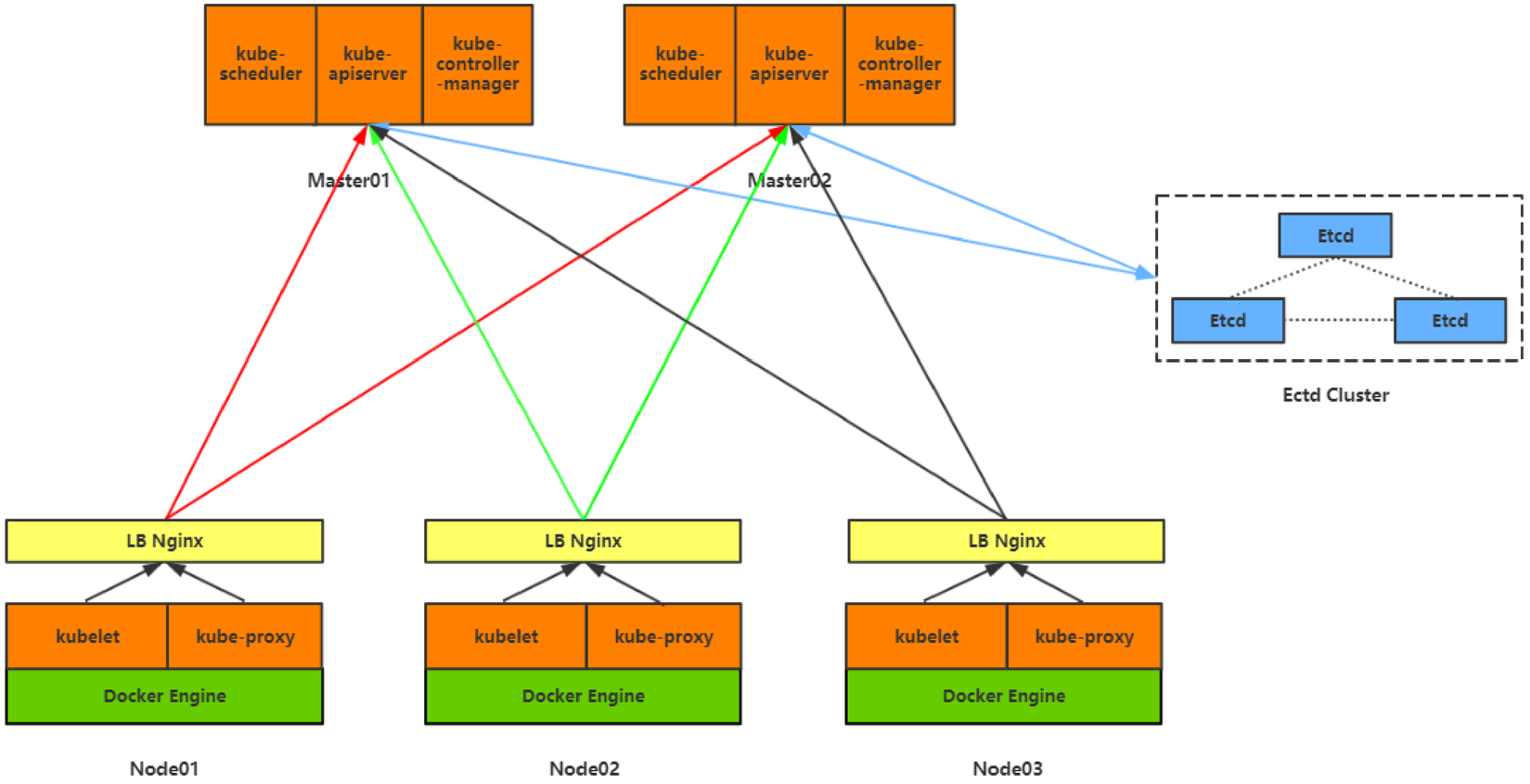
```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: nginx
      image: nginx
      volumeMounts:
      - mountPath: "/usr/share/nginx/html"
        name: wwwroot
  volumes:
    - name: wwwroot
      persistentVolumeClaim:
        claimName: pvc001
```

Kubernetes容器集群管理

高可用架构



高可用架构



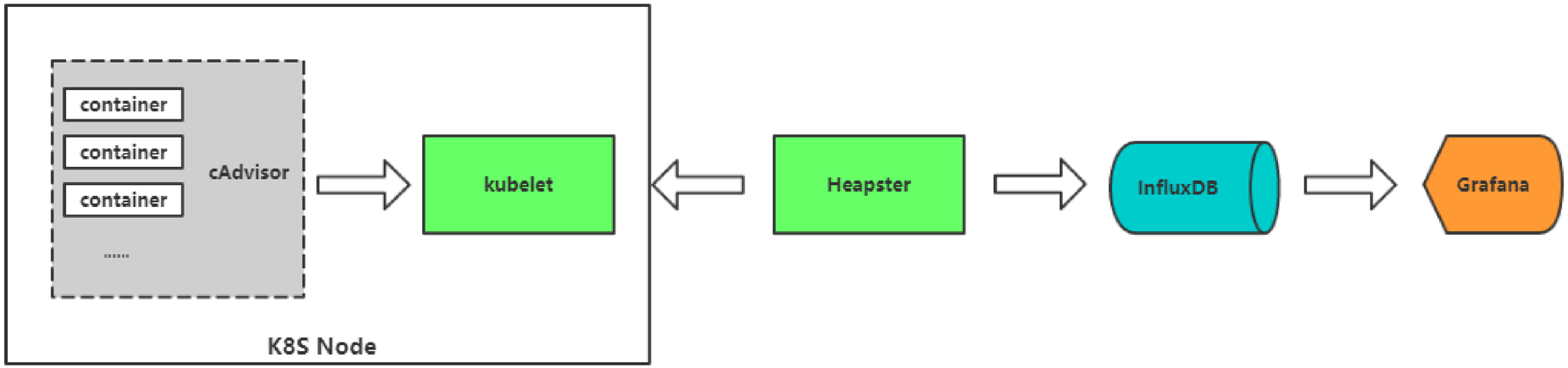
高可用架构

Node节点安装Nginx：

```
# cat > /etc/yum.repos.d/nginx.repo << EOF
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/7/\$basearch/
gpgcheck=0
EOF
# yum install nginx -y
```

Kubernetes容器集群管理

集群监控（cAdvisor+Heapster+InfluxDB+Grafana）



cAdvisor：容器数据收集。

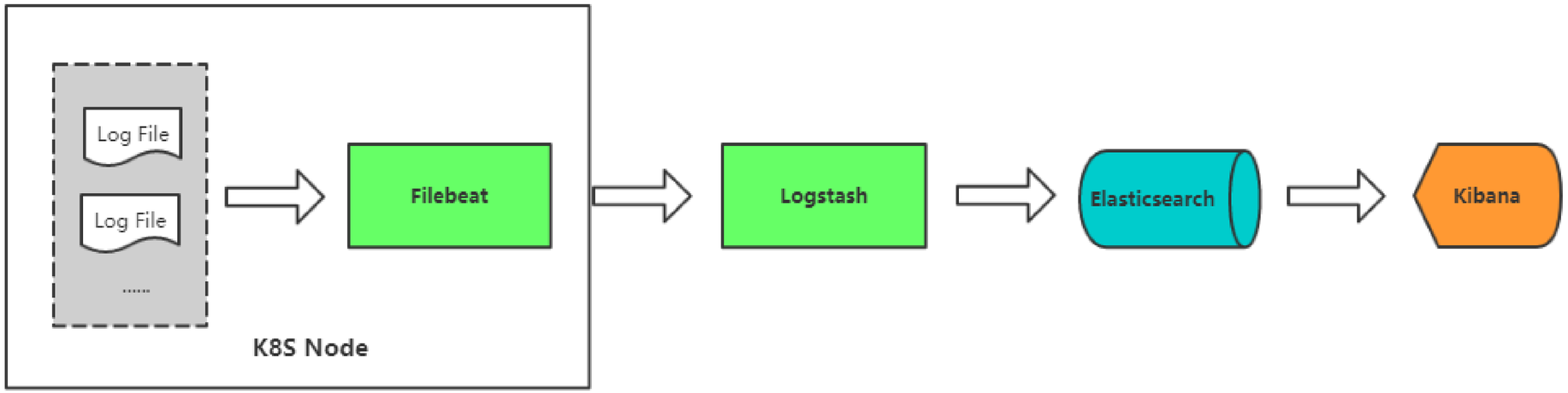
Heapster：集群监控数据收集，汇总所有节点监控数据。

InfluxDB：时序数据库，存储监控数据。

Grafana：可视化展示。

Kubernetes容器集群管理

应用日志收集（Filebeat+ELK)



Filebeat：日志采集工具。

Logstash：数据处理引擎。支持从各种数据源收集数据，并对数据进行过滤、分析等操作。

Elasticsearch：分布式搜索引擎。用于全文检索。

Kibana：可视化平台。能够搜索、展示存储在ES中索引数据。可以很方面以图表、表格、地图形式展示。

谢谢

DevOps技术栈

专注于分享DevOps工具链及经验总结。

微信扫一扫

Docker/K8s技术学员群：397834690