Docker高级应用实战

讲师：李振良

个人介绍

讲师：李振良

高级运维工程师，6年互联网公司运维经验，掌握中小型Linux平台架构设计及运维管理。

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

Python运维开发群：249171211
Docker技术交流群：516039855

Docker技术学员群(VIP)：397834690
注意：未购买务加，申请加入请说明购买的用户名，谢谢

课程目录

一、Docker Compose
二、多Docker主机网络
三、容器集群管理
四、Docker结合Jenkins构建持续集成环境
五、Docker结合Consul实现服务发现
六、Docker API
七、日志管理

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker Compose

- Compose是什么
- Linux系统安装Compose
- Compose常用命令选项
- YAML文件格式及编写注意事项
- Compose配置文件常用参数
- Compose应用实战
  - 一键部署LNMP网站平台
  - 一键部署Nginx反向代理Tomcat集群
  - 一键部署多节点爬虫程序

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker Compose

Compose是什么

Compose是一个定义和管理多容器的工具，使用Python语言编写。使用Compose配置文件描述多个容器应用的架构，比如使用什么镜像、数据卷、网络、映射端口等；然后一条命令管理所有服务，比如启动、停止、重启等。

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker Compose

Linux系统安装Compose

1.下载二进制文件

```
curl -L https://github.com/docker/compose/releases/download/$dockerComposeVersion/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
```

例如：

```
curl -L https://github.com/docker/compose/releases/download/1.14.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
```

2.对二进制文件添加可执行权限

```
chmod +x /usr/local/bin/docker-compose
```

3.测试安装

```
docker-compose --version
```

也可以使用pip工具安装：pip install docker-compose

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker Compose

Linux系统安装Compose

docker-compose.yml

```
version: '3'
services:
  web:
    build: .
    ports:
      - "8888:80"
```

Dockerfile

```
FROM centos:6
MAINTAINER lizhenliang
RUN yum install -y httpd php php-gd php-mysql
RUN echo "<?php phpinfo()?>" > /var/www/html/index.php
CMD ["/usr/sbin/httpd","-D","FOREGROUND"]
EXPOSE 80
```

Docker Compose

Compose常用选项

```
Usage: docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
选项:
-f 指定Compose配置文件，默认docker-compose.yml
-p 指定项目名称，默认目录名
--verbose 显示更多的输出
```

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker Compose

Compose常用命令

命令	描述	命令	描述
build Usage: build [options] [--build-arg key=val...] [SERVICE...] --no-cache 不使用缓存构建镜像 --build-arg key=val 设置构建时变量	重新构建服务	up Usage: up [options] [--scale SERVICE=NUM...] [SERVICE...] -d 在后台运行容器 -no-deps 不启动连接服务 --no-recreate 如果容器存在，不重建他们 --no-build 不构建镜像，即使它丢失 --build 启动容器下构建镜像 --scale SERVICE=NUM 指定一个服务（容器）的启动数量	创建和启动容器
config Usage: config [options] -q, --quiet 只验证不打印 --services 只打印服务名称，每行一个 --volumes 打印数据卷名称，每行一个	验证和查看Compose文件	stop Usage: stop [SERVICE...]	停止服务
exec Usage: exec [options] SERVICE COMMAND [ARGS...] -d 在后台运行命令 --privileged 给这个进程赋予特权权限 -u, --user USER 作为该用户运行该命令 -T 禁用分配伪终端，默认分配一个终端 --index=index 多个容器时的索引数字，默认1	在运行的容器里执行命令	start Usage: start [SERVICE...]	启动服务
port Usage: port [options] SERVICE PRIVATE_PORT --protocol=proto tcp或udp，默认tcp --index=index 多个容器时的索引数字，默认1	打印绑定的开放端口	restart Usage: restart [options] [SERVICE...]	重启服务
ps Usage: ps [options] [SERVICE...] -q 只显示ID	列出容器	top Usage: top [SERVICE...]	显示容器运行进程
rm Usage: rm [options] [SERVICE...] -f, --force 强制删除 -s, --stop 删除容器时如果需要先停止容器 -v 删除与容器相关的任何匿名卷	删除停止的服务容器	logs -f, --follow 实时输出日志 -t, --timestamps 显示时间戳 --tail="all" 从日志末尾显示行数	显示容器的输出
scale Usage: scale [options] [SERVICE=NUM...]	指定一个服务启动容器数量	down	停止容器和删除容器、网络、数据卷和镜像

Docker Compose

YAML文件格式及编写注意事项

YAML是一种标记语言很直观的数据序列化格式，可读性高。类似于XML数据描述语言，语法比XML简单的很多。
YAML数据结构通过缩进来表示，连续的项目通过减号来表示，键值对用冒号分隔，数组用中括号括起来，hash用花括号括起来。

YAML文件格式注意事项：

1. 不支持制表符tab键缩进，需要使用空格缩进
2. 通常开头缩进2个空格
3. 字符后缩进1个空格，如冒号、逗号、横杆
4. 用井号注释
5. 如果包含特殊字符用单引号引起来
6. 布尔值（true、false、yes、no、on、off）必须用引号括起来，这样分析器会将他们解释为字符串。

Docker Compose

Compose配置文件常用参数

键	描述	键	描述
build	构建镜像上下文路径	external_links	连接Compose之外的容器
dockerfile	指定Dockefile文件名	extra_hosts	添加主机名映射，与—add-host相同
image	来自镜像	logging	记录该服务的日志。与—log-driver相同
args	构建参数。在Dockerfile中指定的参数	network_mode	网络模式，与—net相同
command	覆盖默认命令	networks aliases ipv4_address，ipv6_address	要加入的网络。 在加入网络时为该服务指定容器的静态IP地址
container_name	自定义容器名称。如果自定义名称，则无法将服务scale到1容器之外	pid	将PID模式设置主机PID模式，与宿主机共享PID地址空间。pid: “host”
deploy	指定与部署和运行相关的配置。限版本3	ports	暴露端口，与-p相同。但端口不低于60
depends_on	服务之间的依赖，控制服务启动顺序。正常是按顺序启动服务	sysctls	再容器内设置内核参数，可以是数组或字典
dns	自定义DNS服务器，可以是单个值或列表	ulimits	覆盖容器的默认ulimits
entrypoint	覆盖entrypoint	volumes	挂载一个目录或一个已存在的数据卷容器到容器
env_file	从文件添加环境变量，可以是单个值或列表	restart	默认no，always|on-failure|unless-stopped
environment	添加环境变量，可以是数组或字典。布尔值用引号括起来。	hostname	主机名
expose	声明容器服务端口	working_dir	工作目录
links	连接到另一个容器	hostname	主机名

Docker Compose

Compose应用实战

- 一键部署LNMP网站平台
- 一键部署Nginx反向代理Tomcat集群
- 一键部署多节点爬虫程序

讲师：李振良
主页：http://opsdev.ke.qq.com

多主机网络

- 网络术语概念
- 容器跨主机通信主流方案
- Docker Overlay
- Docker Macvlan
- Weave
- OpenvSwitch

讲师：李振良
主页：http://opsdev.ke.qq.com

网络术语概念

二层交换技术：工作在OSI七层网络模型的第二层，通过MAC地址进行帧转发。

三层交换技术：也称为IP交换技术，工作在OSI七层网络模型的第三层，通过IP地址进行包转发。它解决了局域网中网段划分之后，网段中子网必须依赖路由器进行管理的局面。

网桥（Bridge）：工作在OSI七层网络模型的第二层，根据MAC地址转发，类似于二层交换机。Linux网桥将不同的网络接口连接起来，连接的网络接口可以来自不同的局域网，网桥决定了接收的数据包是转发给同一个局域网内主机还是别的网络上。

VLAN（ Virtual Local Area Network，虚拟局域网）：在物理网络（通常路由器接口）基础上建立一个或多个逻辑子网，将一个大的广播域切分若干小的广播域。一个VLAN就是一个广播域，VLAN之间通信通过三层路由器来完成。

网络术语概念

Overlay Network

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。

Overlay网络有以下三种实现方式：

VXLAN（Virtual Extensible Local Area Network，虚拟可扩展局域网），通过将物理服务器或虚拟机发出的数据包封装到UDP中，并使用物理网络的IP/MAC作为外层报文头进行封装，然后在IP网络上传输，到达目的地后由隧道端点解封装并将数据发送给目标物理服务器或虚拟机，扩展了大规模虚拟机网络通信。

由于VLAN Header头部限制长度是12bit，导致只能分配4095个VLAN，也就是4095个网段，在大规模虚拟网络。VXLAN标准定义Header限制长度24bit，可以支持1600万个VLAN，满足大规模虚拟机网络需求。

VXLAN有以下核心技术组成：

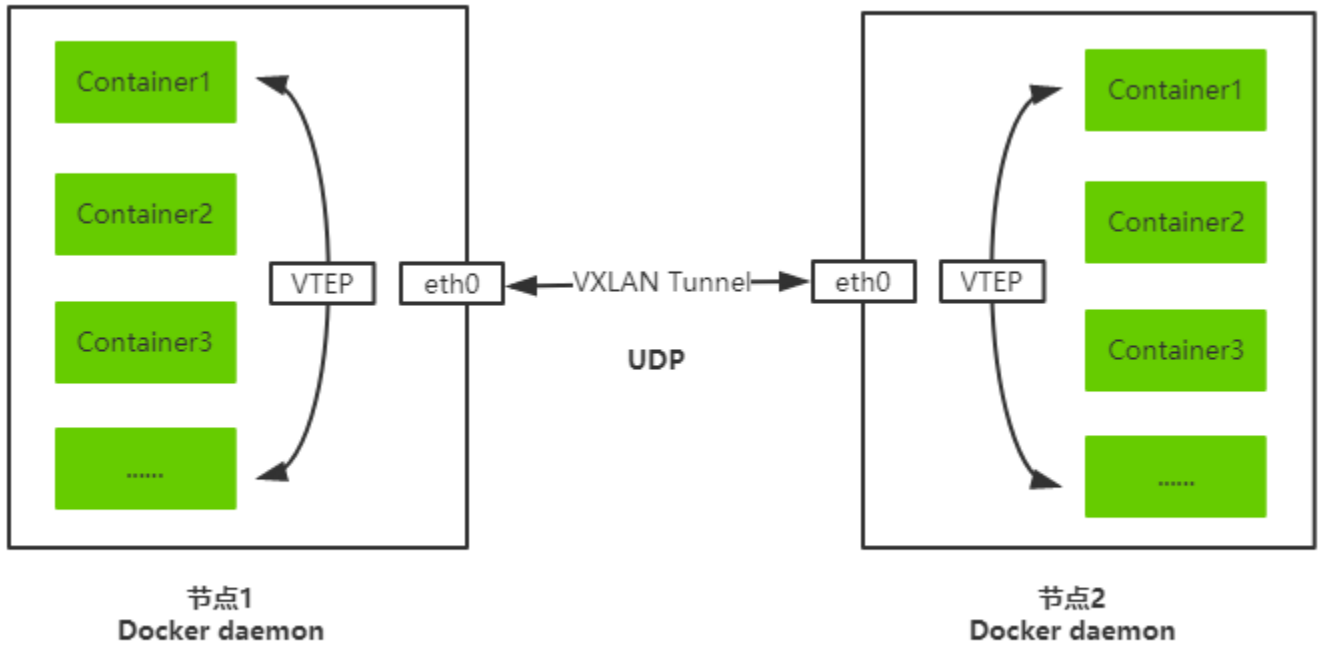
NVE（Network Vritual Endpoint，网络虚拟端点）：实现网络虚拟化功能。报文经过NVE封装转换后，NVE间就可基于三层基础网络建立二层虚拟化网络。

VTEP（VXLAN Tunnel Endpoints，VXLAN隧道端点）：封装在NVE中，用于VXLAN报文的封装和解封装。

VNI（VXLAN Network Identifier，VXLAN网络标识ID）：类似于VLAN ID，用于区分VXLAN段，不同的VXLAN段不能直接二层网络通信。

网络术语概念

Overlay Network



VXLAN网络

网络术语概念

Overlay Network

NVGRE（Network Virtual using Generic Routing Encapsulation，使用GRE虚拟网络）：与VXLAN不同的是，NVGRE没有采用标准传输协议（TCP/UDP），而是借助通用路由封装协议（GRE）。采用24bit标识二层网络分段，与VXLAN一样可以支持1600万个虚拟网络。

STT（Stateless Transport Tunneling，无状态传输隧道）：模拟TCP数据格式进行封装，改造了TCP传输机制，不维护TCP状态信息。

容器跨主机通信主流方案

Docker主机之间容器通信解决方案

- 桥接宿主机网络
- 端口映射
- Docker网络驱动
  - Overlay：基于VXLAN封装实现Docker原生Overlay网络
  - Macvlan：Docker主机网卡接口逻辑上分为多个子接口，每个子接口标识一个VLAN。容器接口直接连接Docker主机网卡接口，通过路由策略转发到另一台Docker主机
- 第三方网络项目

  隧道方案
  - Flannel：支持UDP、VXLAN、Host-gw和AWS-VPC四种工作模式
  - Weave：支持UDP（sleeve模式）和VXLAN（优先fastdp模式）
  - OpenvSwitch：支持VXLAN和GRE协议

  路由方案
  - Calico：支持BGP协议和IPIP隧道。每台宿住主机作为虚拟路由，通过BGP协议实现不同主机容器间通信

Docker Overlay Network

部署前提

Docker通过overlay网络驱动程序支持多主机容器网络通信。

要想使用Docker原生Overlay网络，需要满足以下任意条件：

- Docker运行在Swarm模式
- 使用键值存储的Docker主机集群

我们这里演示第二种，需要满足以下条件：

1. 集群中主机连接到键值存储，Docker支持Consul、Etcd和Zookeeper；
2. 集群中主机运行一个Docker守护进程；
3. 集群中主机必须具有唯一的主机名，因为键值存储使用主机名来标识集群成员；
4. 集群中Linux主机内核版本3.12+，支持VXLAN数据包处理，否则可能无法通信。

Docker Overlay Network

部署

节点1/键值存储：192.168.1.198
节点2：192.168.1.199

1.下载Consul二进制包并启动

```
# wget https://releases.hashicorp.com/consul/0.9.2/consul_0.9.2_linux_amd64.zip
# unzip consul_0.9.2_linux_amd64.zip
# mv consul /usr/bin/consul && chmod +x /usr/bin/consul
# nohup consul agent -server -bootstrap -ui -data-dir /var/lib/consul -client=192.168.1.198 -bind=192.168.1.198
&>/var/log/consul.log &
```

2.节点配置Docker守护进程连接Consul

```
# vi /lib/systemd/system/docker.service
[Service]
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock --cluster-store consul:// 192.168.1.198:8500
--cluster-advertise 192.168.1.198:2375
# systemctl restart docker
```

3.创建overlay网络

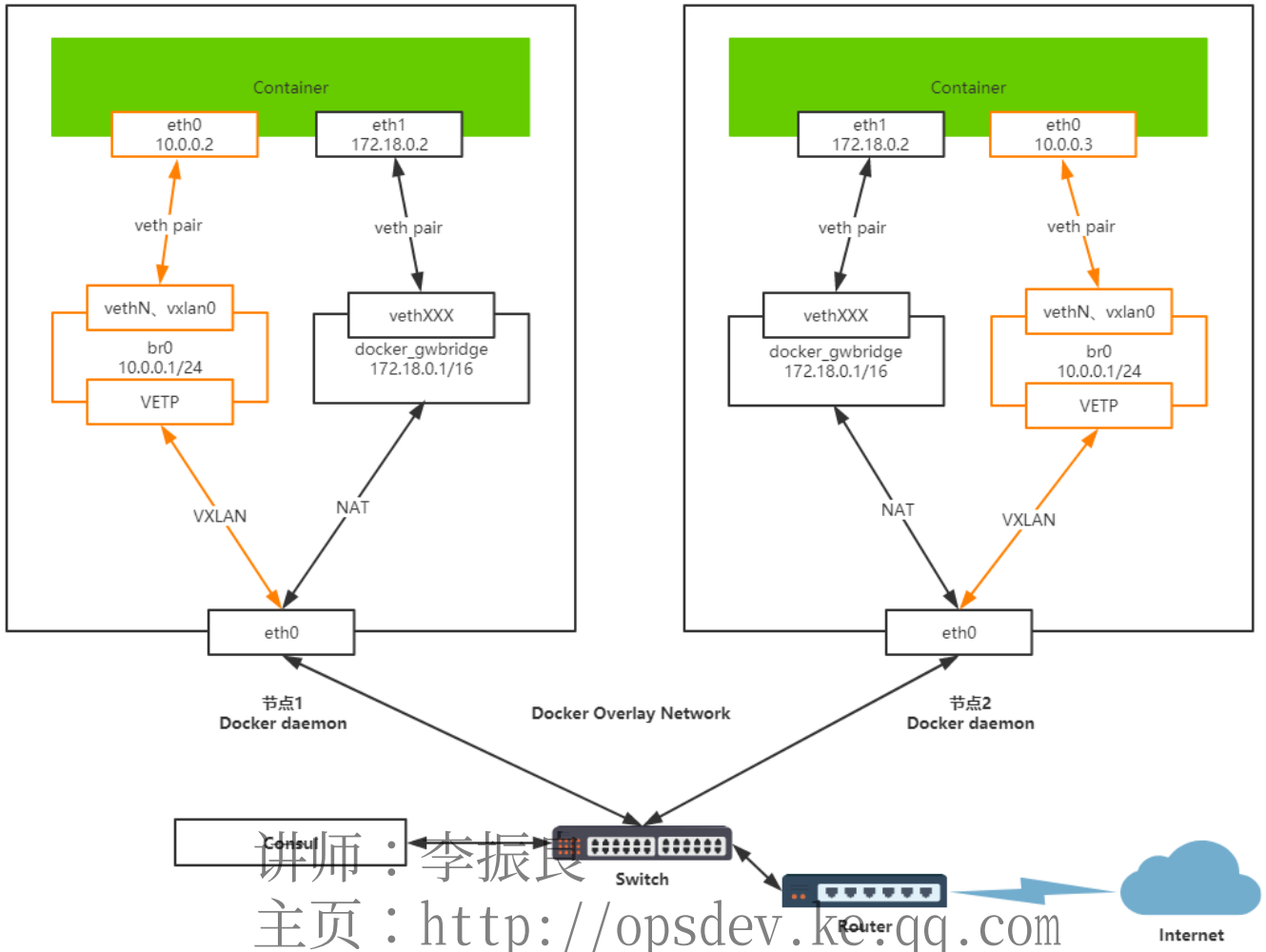
```
# docker network create -d overlay multi_host
```

4.测试互通

```
# docker run -itd --net=multi_host busybox
```

Docker Overlay Network

工作流程



Docker Macvlan Network

部署

Macvlan Bridge模式：

1.创建macvlan网络

```
docker network create -d macvlan --subnet=172.100.1.0/24 --gateway=172.100.1.1 -o parent=eth0 macvlan_net
```

2.测试互通

```
macvlan-01# docker run -it --net macvlan_net --ip=172.100.1.10 busybox
macvlan-02# docker run -it --net macvlan_net --ip=172.100.1.11 busybox ping 172.100.1.10
```

Macvlan VLAN Bridge模式：

1.创建一个VLAN，VLAN ID 50

```
ip link add link eth0 name eth0.50 type vlan id 50
```

2.创建Macvlan网络

```
docker network create -d macvlan --subnet=172.18.50.0/24 --gateway=172.18.50.1 -o parent=eth0.50 macvlan_net50
```

3.测试互通

```
macvlan-01# docker run -it --net macvlan_net50 --ip=172.18.50.10 busybox
macvlan-02# docker run -it --net macvlan_net50 --ip=172.18.50.11 busybox ping 172.18.50.10
```

Weave

介绍

Weave在Docker主机之间实现Overlay网络，使用业界标准VXLAN封装，基于UDP传输，也可以加密传输。
Weave Net创建一个连接多个Docker主机的虚拟网络，类似于一个以太网交换机，所有的容器都连接到这上面，互相通信。
Weave Net由多个peer组成，Weave路由器运行不同Docker主机上，是一个用户空间的进程；每个peer都有一个名称，重启保持不变。它们通过TCP连接彼此，建立后交换拓扑信息。
Weave Net可以在具有编号拓扑的部分连接的网络中路由数据包。例如，在下面网络中，peer1直接连接2和3，但是如果1需要发送数据包到4和5，则必须先将其发送到peer3。

Weave Net中的”fast data path”使用Linux内核的OpenvSwich datapath模块。该模块使Weave Net路由器能够告知内核如何处理数据包。OpenvSwich datapath和VXLAN功能在Linux内核版本3.12+才支持，如果内核不支持，则Weave Net使用”user mode”数据包路径。Weave Net会自动选择两台主机之间最快的路径传输数据，提供近原生吞吐量和延迟。

Weave

部署

官方文档：https://www.weave.works/docs/net/latest/install/installing-weave

使用前提：

1. 确保Linux内核版本3.8+，Docker1.10+。
2. 节点之间如果有防火墙时，必须彼此放行TCP 6783和UDP 6783/6784端口，这是Weave控制和数据端口。
3. 主机名不能相同，通过主机名标识子网。

部署：

1. 安装Weave

```
curl -L git.io/Weave -o /usr/local/bin/Weave
chmod +x /usr/local/bin/Weave
```

2. 启动并与其他主机建立连接

```
weave-01:~# weave launch
weave-02:~# weave launch <ip address>
```

3. 使用Weave网络创建容器

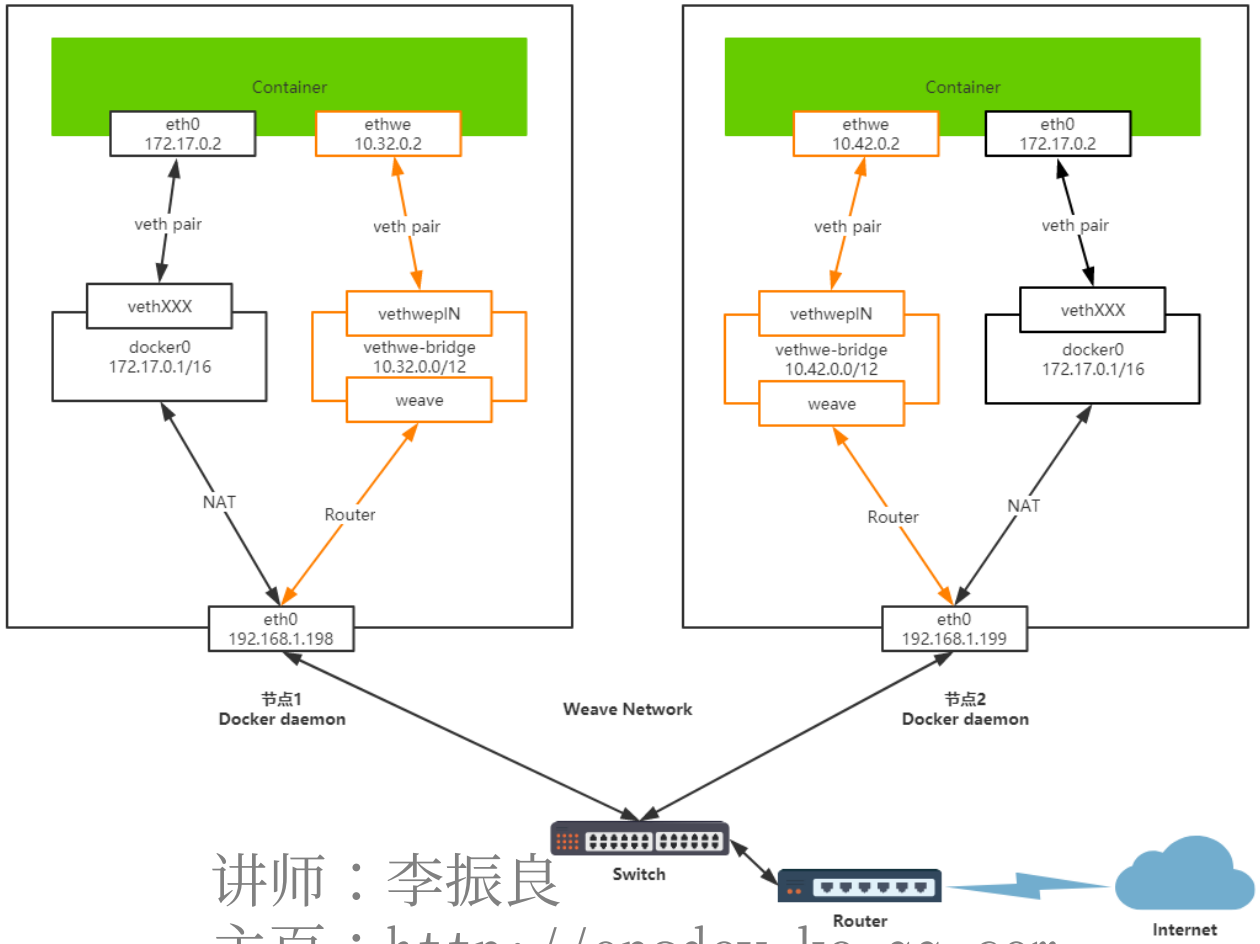
```
方式1：eval $(weave env)
方式2：docker run -it --net=weave  busybox
```

4. 测试互通

```
docker run -it busybox
```

Weave

工作流程



讲师：李振良
主页：http://opsdev.ke.qq.com

Weave

其他功能

- **IP地址管理（IPAM）**
  Weave自动为容器分配唯一的IP地址。可通过weave ps查看
- **命名和发现**
  命名的容器自动会注册到Weave DNS中，并可以通过容器名称访问。
- **负载均衡**
  允许注册多个相同名称的容器，Weave DNS随机为每个请求返回地址，提供基本的负载均衡功能。
- **手动指定IP地址**
  docker run -it -e WEAVE_CIDR=10.32.0.100/24 busybox
- **动态拓扑**
  可以在不停止或重新配置剩余Docker主机的情况下添加主机到Weave网络中或从Weave网络中删除
- **容错**
  weave peer不断交换拓扑信息，监视和建立与其他peer的网络连接。如果有主机或网络出现故障，Weave会绕过这个主机，保证两边容器可以继续通信，当恢复时，恢复完全连接。

OpenvSwitch

OVS介绍

什么是OpenVSwich?

OpenvSwich：开放虚拟交换标准，是一种基于开源Apache2.0许可证的多层软件交换机，专门管理多租赁云计算网络环境，支持KVM、Xen等虚拟化技术。

支持以下功能：

1. 支持标准802.1Q VLAN模块的Trunk和access端口模式；
2. QoS（Quality of Service）配置，及管理；
3. 支持OpenFlow协议；
4. 支持GRE、VXLAN、STT和LISP隧道；
5. 具有C和Python接口配置数据库；
6. 支持内核态和用户态的转发引擎设置；
7. 支持流量控制及监控。

主要组成部分：

ovs-vswitchd	一个实现交换机的守护程序
ovsdb-server	一个轻量级数据库，ovs-vswitchd查询以获取其配置
ovs-dpctl	用于配置交换机的内核模块工具
ovs-vsctl	用于查看和更新ovs-vswitchd的配置工具
ovs-appctl	一个向运行OVS守护程序发送命令的工具

还提供了openflow的工具：

ovs-ofctl	用于查看和控制OpenFlow交换机和控制器
ovs-pki	用于创建和管理公钥
ovs-tcpundump	解析openflow消息

OpenvSwitch

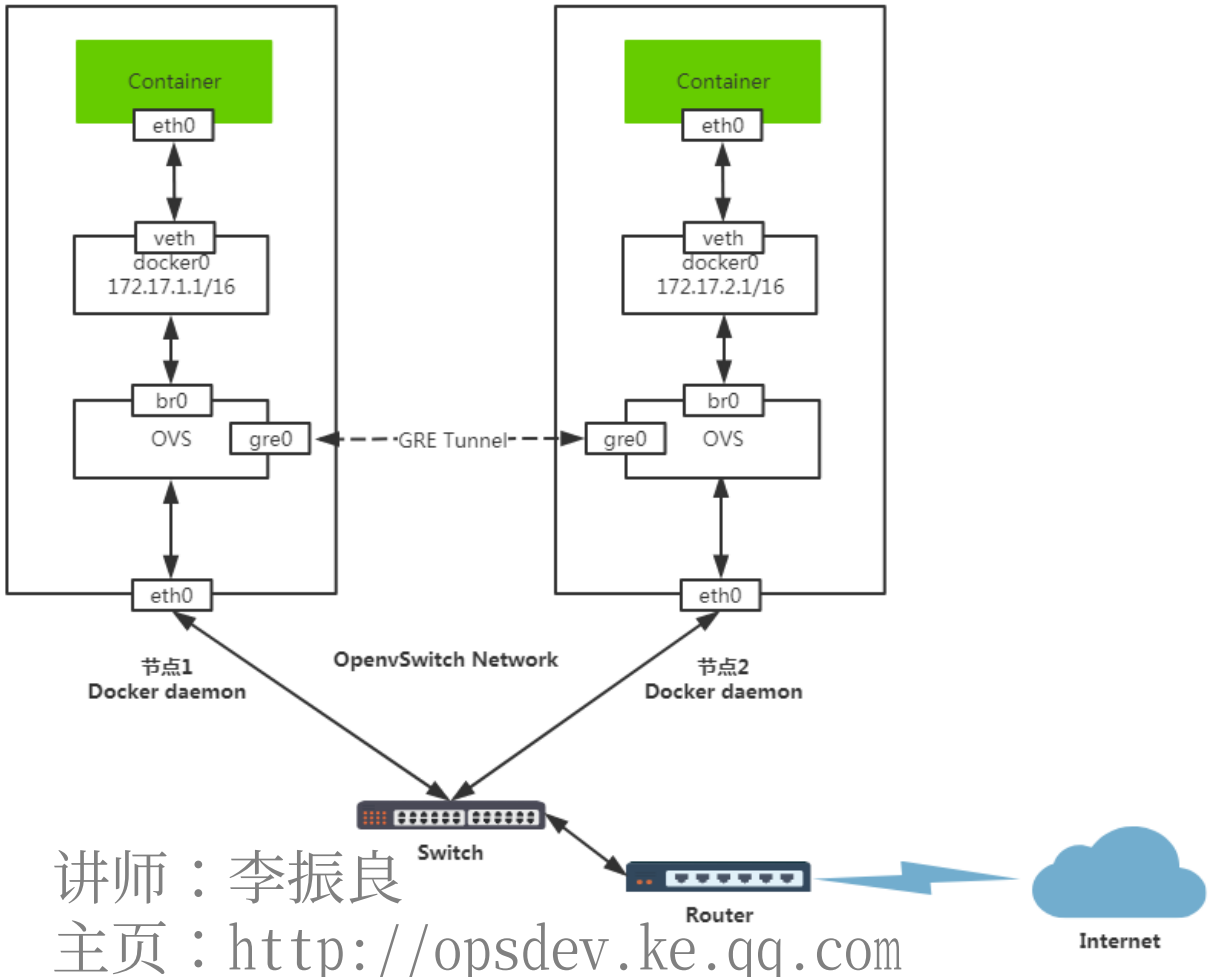
安装部署OVS并建立GRE隧道

节点1:192.168.1.198　　容器网段：172.17.1.0/24
节点2:192.168.1.199　　容器网段：172.17.2.0/24

1.安装OVS
```
# apt-get install openvswitch-switch bridge-utils
```
2.创建网桥并激活
```
# ovs-vsctl add-br br0
# ip link set dev br0 up
```
3.将gre0虚拟接口加入网桥br0，并设置接口类型和对端IP地址（远程IP指定对端）
```
# ovs-vsctl add-port br0 gre0 -- set Interface gre0 type=gre options:remote_ip=192.168.1.199
```
4.添加docker0网桥到OVS网桥br0
```
# brctl addif docker0 br0
```
5.查看网桥信息
```
# ovs-vsctl show
# brctl show
```
6.添加静态路由
```
# ip route add 172.17.0.0/16 dev docker0
```
7.验证互通
```
# docker run -it busybox
```

OpenvSwitch

工作原理



讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理

- Docker容器集群管理主流方案
- Swarm
- Kubernetes

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理

Docker容器集群管理主流方案

- Swarm

Docker公司自研发的集群管理系统。

- Kubernetes

Google开源的一个容器集群管理系统，用于自动化部署、扩展和管理容器应用。也称为K8S

- Mesos

Mesos是一个集群资源调度系统，对集群中的资源进行分配和管理。Marathon是运行在Mesos之上的一个服务管理框架，可管理容器生命周期。

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理

Docker容器集群管理主流方案

Swarm、Kubernetes和Mesos简单比较：

- 复杂性

Swarm使用标准Docker接口，集成于Docker Engine，内置Overlay网络、服务发现、负载均衡，很容易上手，学习成本低。

K8S成熟且复杂，自己的管理体系，服务发现，负载均衡等功能，学习成本高。

Mesos是一个成熟分布式资源管理框架，一个通用的集群管理系统。

- 功能

Swarm支持Docker Compose v3来实现服务编排。

K8S强大的功能，有着一套整体容器解决方案，使用起来更轻松。

- 社区活跃度

K8S社区相比Swarm和Mesos活跃度都高。

容器集群管理之Swarm

Swarm

- Swarm介绍
- 集群部署及节点管理
- 服务管理
- 使用原生Overlay网络
- 数据持久化
- 服务发现与负载均衡
- 高可用性
- 配置文件存储
- 应用实战
  - 手动创建和服务编排部署LNMP网站平台

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Swarm

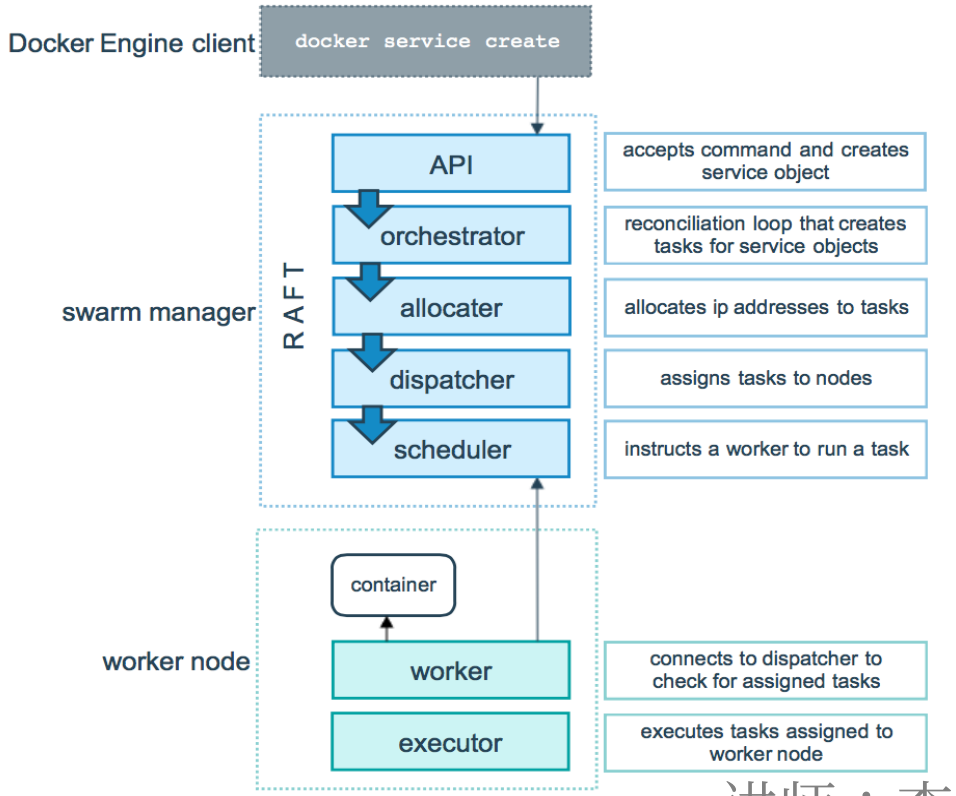
Swarm介绍

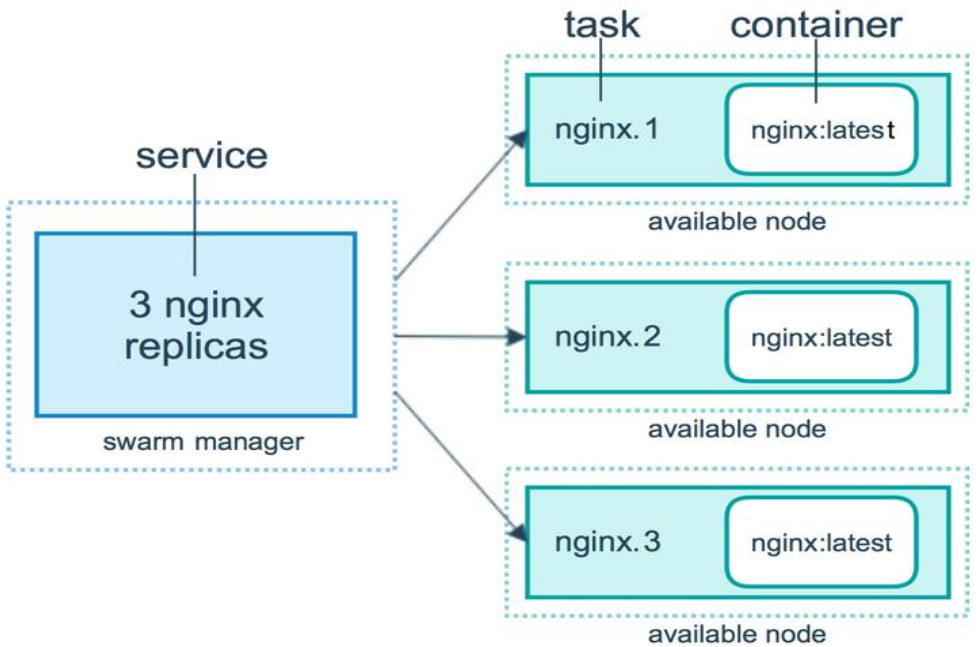
Swarm是什么？
Swarm是Docker公司自研发的容器集群管理系统，Swarm在早期是作为一个独立服务存在，在Docker Engine v1.12中集成了Swarm的集群管理和编排功能。可以通过初始化Swarm或加入现有Swarm来启用Docker引擎的Swarm模式。
Docker Engine CLI和API包括了管理Swarm节点命令，比如添加、删除节点，以及在Swarm中部署和编排服务。
也增加了服务栈（Stack）、服务（Service）、任务（Task）概念。

Swarm两种角色：
Manager：接收客户端服务定义，将任务发送到worker节点；维护集群期望状态和集群管理功能及Leader选举。默认情况下manager节点也会运行任务，也可以配置只做管理任务。
Worker：接收并执行从管理节点分配的任务，并报告任务当前状态，以便管理节点维护每个服务期望状态。

容器集群管理之Swarm

Swarm介绍





讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Swarm

Swarm介绍

Swarm特点：

1. Docker Engine集成集群管理

使用Docker Engine CLI 创建一个Docker Engine的Swarm模式，在集群中部署应用程序服务。

2. 去中心化设计

Swarm角色分为Manager和Worker节点，Manager节点故障不影响应用使用。

3. 扩容缩容

可以声明每个服务运行的容器数量，通过添加或删除容器数自动调整期望的状态。

4. 期望状态协调

Swarm Manager节点不断监视集群状态，并调整当前状态与期望状态之间的差异。例如，设置一个服务运行10个副本容器，如果两个副本的服务器节点崩溃，Manager将创建两个新的副本替代崩溃的副本。并将新的副本分配到可用的worker节点。

5. 多主机网络

可以为服务指定overlay网络。当初始化或更新应用程序时，Swarm manager会自动为overlay网络上的容器分配IP地址。

6. 服务发现

Swarm manager节点为集群中的每个服务分配唯一的DNS记录和负载均衡VIP。可以通过Swarm内置的DNS服务器查询集群中每个运行的容器。

7. 负载均衡

实现服务副本负载均衡，提供入口访问。也可以将服务入口暴露给外部负载均衡器再次负载均衡。

8. 安全传输

Swarm中的每个节点使用TLS相互验证和加密，确保安全的其他节点通信。

9. 滚动更新

升级时，逐步将应用服务更新到节点，如果出现问题，可以将任务回滚到先前版本。

容器集群管理之Swarm

集群部署及节点管理

使用swarm前提：

- Docker版本1.12+
- 集群节点之间保证TCP 2377、TCP/UDP 7946和UDP 4789端口通信

节点规划：

操作系统：Ubuntu16.04_x64
管理节点：192.168.1.110
工作节点：192.168.1.111
工作节点：192.168.1.112

管理节点初始化swarm：

```
docker swarm init --advertise-addr 192.168.1.110
```

工作节点加入swarm：

```
docker swarm join --token SWMTKN-1-XXX 192.168.1.110:2377
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Swarm

服务管理

```
# 创建服务
docker service create --replicas 1 --name hello busybox
# 显示服务详细信息
docker service inspect --pretty hello  # 易于阅读显示
docker service inspect hello  # json格式返回
# 扩展服务实例数
docker service scale hello=3
# 查看服务任务
docker service ls
docker service ps hello
docker service ps -f 'desired-state=running' hello
# 滚动更新服务
docker service create \
--replicas 3 \
--name redis \
--update-delay 10s \
redis:3.0.6
docker service update --image redis:3.0.7 redis
```

容器集群管理之Swarm

服务管理

```
# 创建服务时设定更新策略
docker service create \
--name my_web \
--replicas 10 \
--update-delay 10s \
--update-parallelism 2 \
--update-failure-action continue \
nginx:1.12
# 创建服务时设定回滚策略
docker service create \
--name my_web \
--replicas 10 \
--rollback-parallelism 2 \
--rollback-monitor 20s \
--rollback-max-failure-ratio .2 \
nginx:1.12
# 服务更新
docker service update --image nginx:1.13 my_web
# 手动回滚
docker service update --rollback my_web
```

容器集群管理之Swarm

使用原生Overlay Network

```
# 创建overlay网络
docker network create --driver overlay my-network
# 创建新服务并使用overlay网络
docker service create \
  --replicas 3 \
  --network my-network \
  --name my-web \
  nginx
# 将现有服务连接到overlay网络
docker service update --network-add my-network my-web
# 删除正在运行的服务网络连接
docker service update --network-rm my-network my-web
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Swarm

数据持久化

```
1. volume
# 创建数据卷
docker service create \
  --mount type=volume src=<VOLUME-NAME>,dst=<CONTAINER-PATH> \
  --name myservice \
  <IMAGE>
# 查看数据卷详细信息
docker volume inspect <VOLUME-NAME>
# 使用NFS共享存储作为数据卷
docker service create \
     --mount 'type=volume,src=<VOLUME-NAME>,dst=<CONTAINER-PATH>,volume-driver=local,volume-
opt=type=nfs,volume-opt=device=<nfs-server>:<nfs-path>,"volume-opt=o=addr=<nfs-
address>,vers=4,soft,timeo=180,bg,tcp,rw"' \
    --name myservice \
    <IMAGE>
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Swarm

数据持久化

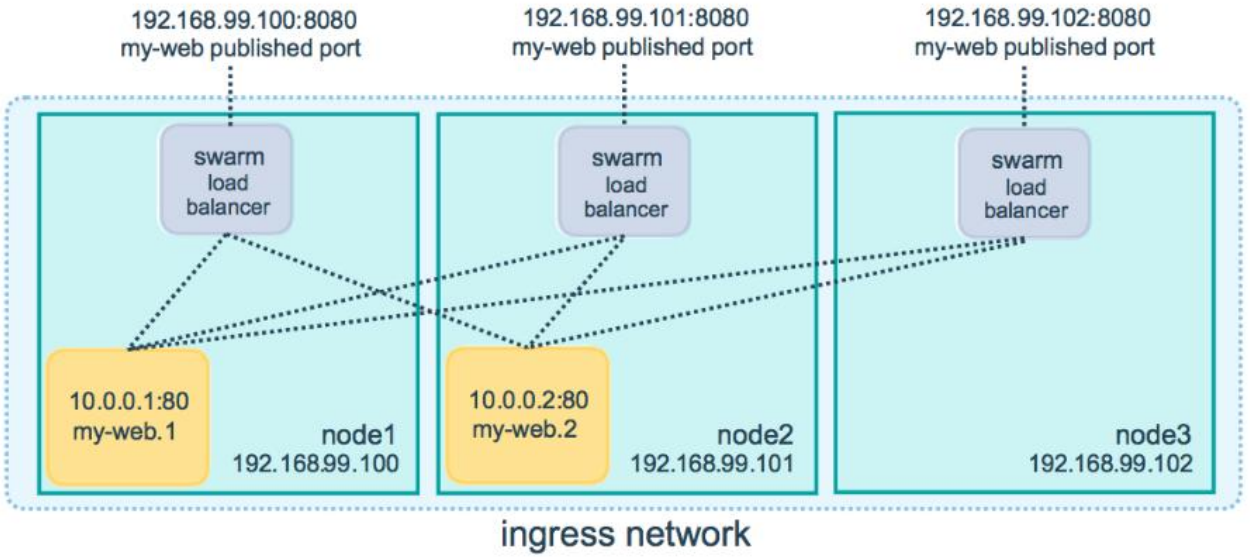
2. bind

```
# 读写挂载
docker service create \
  --mount type=bind,src=<HOST-PATH>,dst=<CONTAINER-PATH> \
  --name myservice \
  <IMAGE>
# 只读挂载
docker service create \
  --mount type=bind,src=<HOST-PATH>,dst=<CONTAINER-PATH>,readonly \
  --name myservice \
  <IMAGE>
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Swarm

服务发现与负载均衡



Swarm模式内置DNS组件，可以自动为集群中的每个服务分配DNS记录。Swarm manager使用内部负载均衡，根据服务的DNS名称在集群内的服务之间分发请求。

Swarm manager使用 ingress load blancing暴露你想从外部访问集群提供的服务。Swarm manager自动为服务分配一个范围30000-32767端口的Published Port,也可以为该服务指定一个Published Port。

ingress network是一个特殊的overlay网络，便于服务的节点直接负载均衡。当任何swarm节点在已发布的端口上接收到请求时，它将该请求转发给调用的IPVS模块，IPVS跟踪参与该服务的所有容器IP地址，选择其中一个，并通过ingress network将请求路由给它。

容器集群管理之Swarm

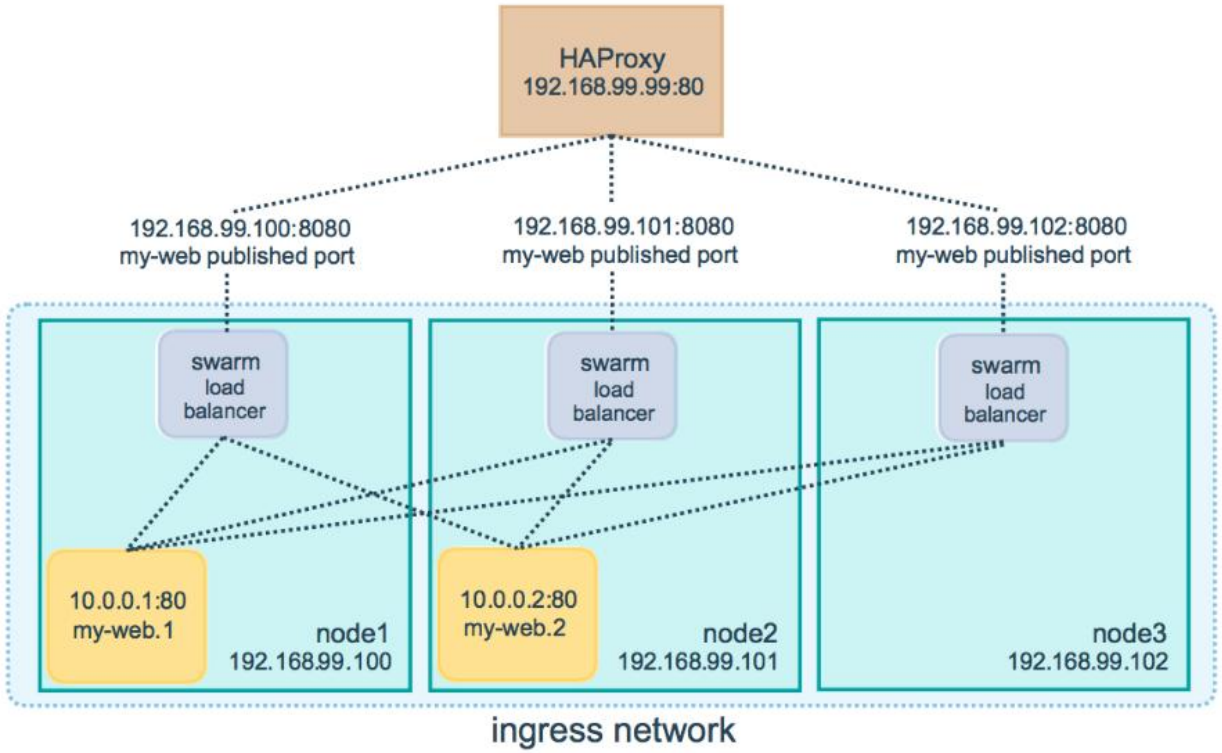
服务发现与负载均衡

```
# 进容器查看DNS记录
nslookup hello
# 获取虚拟IP
docker service inspect -f '{{json .Endpoint.VirtualIPs}}' hello
# 设置DNS轮询模式
docker service create \
--replicas 3 \
--name my-web \
--network my-network \
--endpoint-mode dnsrr \
nginx
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Swarm

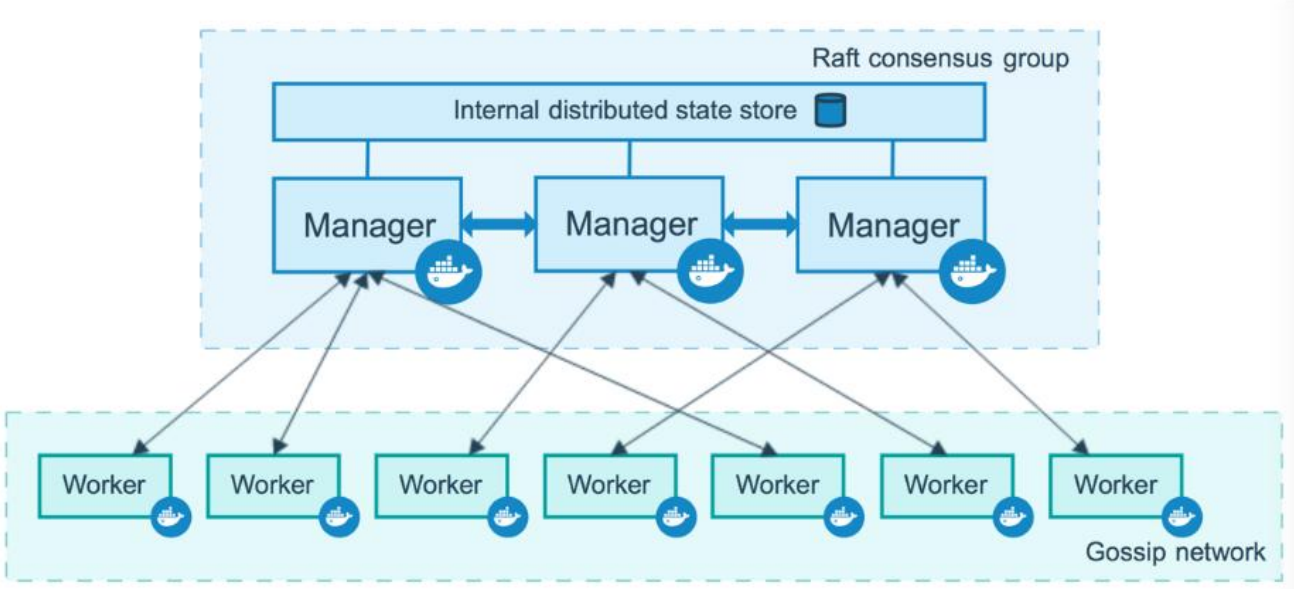
服务发现与负载均衡



讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Swarm

高可用性



Manager节点任务：

1. 维护集群状态
2. 调度服务
3. 提供swarm模式的HTTP API

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Swarm

高可用性

Swarm Size	Majority	Fault Tolerance
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	3
9	5	4

为了利用swarm模式的容错功能，应保持集群中奇数管理员来支持manager节点故障。当leader故障时，会选举新的leader。

故障恢复：

如果swarm失去法定人数，swarm不能自动恢复，工作节点上的任务继续运行，不受影响，但无法执行管理任务，包括扩展或更新服务，加入或删除节点。恢复的最佳方式是将丢失的leader节点重新联机。如果不可能，唯一方法是使用—force-new-cluster管理节点的操作,这将去除本机之外的所有管理器身份。

```
docker swarm init --force-new-cluster --advertise-addr 192.168.1.111:2377
```

容器集群管理之Swarm

高可用性

```
# 以管理节点角色加入swarm
docker swarm join-token manager
# 在管理节点手动改变角色
docker node promote <NAME>
docker node demote <NAME>
# 在管理节点查看角色
docker node ls
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Swarm

配置文件存储

1.生成一个基本的Nginx配置文件

```
site.conf
server {
    listen          80;
    server_name     localhost;
    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }
}
```

2.将site.conf保存到docker配置中

```
docker config create site.conf site.conf
docker config ls
```

3.创建一个Nginx并应用这个配置

```
docker service create \
     --name nginx \
     --config source=site.conf,target=/etc/nginx/conf.d/site.conf \
     --publish 8080:80 \
     nginx:latest
```

容器集群管理之Swarm

应用实战

1.创建overlay网络

```
docker network create -d overlay lnmp
```

2.创建Nginx服务

```
docker service create \
--name nginx \
--replicas 3 \
--network lnmp \
--publish 8888:80 \
--mount
type=volume,source=wwwroot,destination=/usr/local/nginx/html \
192.168.1.110:5000/nginx:v1
```

3.创建PHP服务

```
docker service create \
--name php \
--replicas 3 \
--network lnmp \
--mount
type=volume,source=wwwroot,destination=/usr/local/nginx/html \
192.168.1.110:5000/php:v1
```

4.创建MySQL服务

```
docker service create \
--name mysql \
--replicas 1 \
--network lnmp \
--config src=my.cnf,target="/etc/mysql/conf.d/my.cnf" \
--mount type=volume,source=dbdata,destination=/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456 \
-e MYSQL_USER=wordpress \
-e MYSQL_PASSWORD=wp123456 \
-e MYSQL_DATABASE=wordpress \
mysql:5.6
```

容器集群管理之Swarm

应用实战

NOT SUPPORTED FOR `DOCKER STACK DEPLOY`

The following sub-options (supported for `docker compose up` and `docker compose run`) are *not supported* for `docker stack deploy` or the `deploy` key.

- build
- cgroup_parent
- container_name
- devices
- dns
- dns_search
- tmpfs
- external_links
- links
- network_mode
- security_opt
- stop_signal
- sysctls
- userns_mode

```yaml
version: '3.3'
services:
  nginx:
    image: 192.168.1.197:5000/nginx:v1
    ports:
      - 8888:80
    networks:
      - lnmp_net
    volumes:
      - type: volume
        source: wwwroot
        target: /usr/local/nginx/html
    deploy:
      mode: replicated
      replicas: 3
    depends_on:
      - php
      - mysql

  php:
    image: 192.168.1.197:5000/php:v1
    networks:
      - lnmp_net
    volumes:
      - type: volume
        source: wwwroot
        target: /usr/local/nginx/html
    deploy:
      mode: replicated
      replicas: 3

  mysql:
    image: mysql:5.6
    ports:
      - 3306:3306
    networks:
      - lnmp_net
    volumes:
      - "dbdata:/var/lib/mysql"
    command: --character-set-server=utf8
    environment:
      MYSQL_ROOT_PASSWORD: 123456
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wp123456

networks:
  lnmp_net:
    driver: overlay

volumes:
  wwwroot:
  dbdata:
```

一键部署：docker stack deploy -c service_stack.yml lnmp

容器集群管理之Kubernetes

- Kubernetes介绍
- 基本对象概念
- 系统架构及组件功能
- 集群部署
- Kubectl命令行管理对象
- YAML配置文件管理对象
- Pod管理
- Service
- Volume
- 高可用性设计

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

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2017年9月29日最新稳定版本是v1.8。

Kubernetes目标是让部署容器化应用简单高效。

官方网站：www.kubernetes.io

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

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**
  支持属性访问控制（ABAC）、角色访问控制（RBAC）认证授权策略。

容器集群管理之Kubernetes

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。

容器集群管理之Kubernetes

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

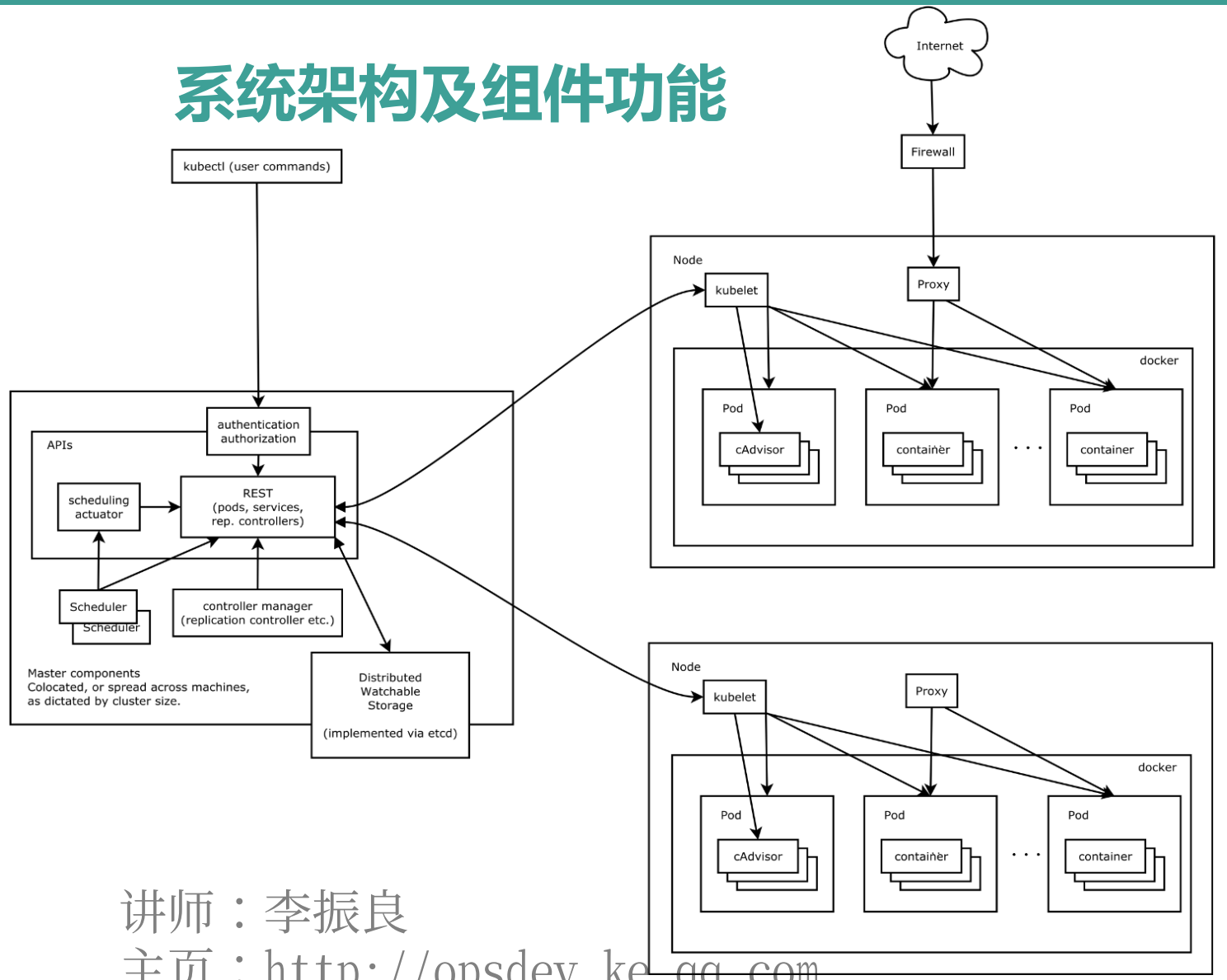
  DaemonSet确保所有（或一些）节点运行同一个Pod。当节点加入Kubernetes集群中，Pod会被调度到该节点上运行，当节点从集群中移除时，DaemonSet的Pod会被删除。删除DaemonSet会清理它所有创建的Pod。

- 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。

容器集群管理之Kubernetes

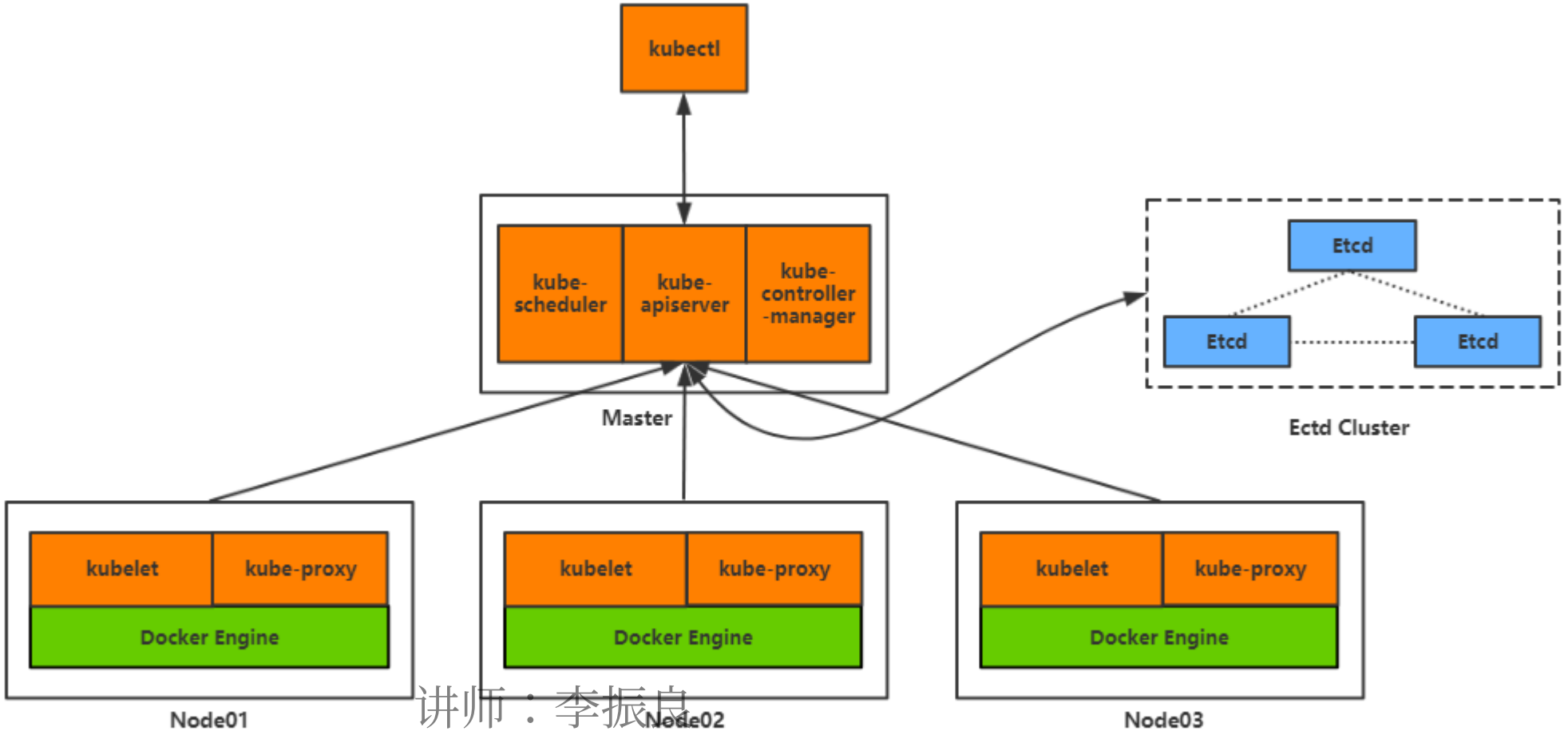
系统架构及组件功能



讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

系统架构及组件功能



容器集群管理之Kubernetes

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。

容器集群管理之Kubernetes

集群部署

环境说明：
操作系统：Ubuntu16.04 or CentOS7
Kubernetes版本：v1.8
Docker版本：v17.06-ce

角色	IP	组件
master	192.168.1.198	etcd kube-apiserver kube-controller-manager kube-scheduler
node01	192.168.1.199	kubelet kube-proxy docker
node02	192.168.1.200	kubelet kube-proxy docker

步骤：

1. 获取Kubernetes二进制包
2. 运行Master组件
3. 运行Node组件
4. 查看集群状态
5. 启动一个示例

容器集群管理之Kubernetes

集群部署

安装Etcd存储服务：

```
# apt-get install etcd -y
# vi /etc/default/etcd
ETCD_DATA_DIR="/var/lib/etcd/default"
ETCD_LISTEN_CLIENT_URLS="http://0.0.0.0:2379"
ETCD_ADVERTISE_CLIENT_URLS=http://0.0.0.0:2379
# service etcd restart
```

运行Master组件：

```
# mkdir -p /opt/kubernetes/{bin,cfg}
# chmod +x *
# mv kube-apiserver kube-controller-manager kube-scheduler kubectl /opt/kubernetes/bin
# ./apiserver.sh 192.168.1.198 http://192.168.1.198:2379
# ./scheduler.sh 192.168.1.198
# ./controller-manager.sh 192.168.1.198
# echo "export PATH=$PATH:/opt/kubernetes/bin" >> /etc/profile
# source /etc/profile
```

运行Node组件：

```
# mkdir -p /opt/kubernetes/{bin,cfg}
# chmod +x *
# mv kubelet kube-proxy /opt/kubernetes/bin
# ./kubelet.sh 192.168.1.198 192.168.1.200 10.10.10.2
# ./proxy.sh 192.168.1.198 192.168.1.200
```

查看集群状态：

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

容器集群管理之Kubernetes

集群部署

示例：

```
# 创建并运行Nginx镜像
kubectl run nginx --image=nginx --replicas=3
# 查看pods
kubectl get pods
# 查看pods及运行节点
kubectl get pods -o wide
# 为deployment资源暴露一个新Service
kubectl expose deployment nginx --port=88 --target-port=80
# 查看Service
kubectl get services nginx
# 查看Service的endpoints
kubectl get ep nginx
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Kubectl命令行管理对象

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Kubernetes服务。
	run	创建并运行一个特定的镜像，可能是副本。 创建一个deployment或job管理创建的容器。
	set	配置应用资源。 修改现有应用程序资源。
	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	执行指定复制控制的滚动更新。
	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	

容器集群管理之Kubernetes

Kubectl命令行管理对象

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pod或指定的资源中容器打印日志。如果pod只有一个容器，容器名称是可选的。
	attach	附加到一个进程到一个已经运行的容器。
	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	为kubernetes API Server启动服务代理。
	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。YAML和JSON格式都接受。
设置命令	label	更新资源上的标签。
	annotate	在一个或多个资源上更新注释。
	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。

容器集群管理之Kubernetes

Kubectl命令行管理对象

示例：

```
# 运行应用程序
kubectl run hello-world --replicas=3 --labels="app=example" --image=nginx:1.10  --port=80
# 显示有关Deployments信息
kubectl get deployments hello-world
kubectl describe deployments hello-world
# 显示有关ReplicaSet信息
kubectl get replicasets
kubectl describe replicasets
# 创建一个Service对象暴露Deployment（在88端口负载TCP流量）
kubectl expose deployment hello-world --port=88 --type=NodePort --target-port=80 --name=example-service
# 创建一个Service对象暴露Deployment（在4100端口负载UDP流量）
kubectl expose deployment hello-world --port=4100 --type=NodePort --protocol=udp --target-port=80 --
name=example-service
# 显示有关Service信息
kubectl describe services example-service
# 使用节点IP和节点端口访问应用程序
curl http://<public-node-ip>:<node-port>
```

容器集群管理之Kubernetes

Kubectl命令行管理对象

示例：

```
# 列出运行应用程序的pod
kubectl get pods --selector="app=example" --output=wide
# 查看pods所有标签
kubectl get pods --show-labels
# 根据标签查看pods
kubectl get pods -l app=example
# 扩容Pod副本数
kubectl scale deployment --replicas=10 hello-world
# 清理应用程序
kubectl delete services example-service
kubectl delete deployment hello-world
```

讲师：李振良
主页：http://opsdev.ke.qq.com

YAML配置文件管理对象

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。

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

YAML配置文件管理对象

创建Deployment对象：

```
# cat nginx-deployment.yaml
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
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

YAML配置文件管理对象

对象管理：

```
# 创建deployment资源
kubectl create -f nginx-deployment.yaml
# 查看deployment
kubectl get deploy
# 查看ReplicaSet
kubectl get rs
# 查看pods所有标签
kubectl get pods --show-labels
# 根据标签查看pods
kubectl get pods -l app=nginx
```

```
# 滚动更新镜像
kubectl set image deployment/nginx-deployment nginx=nginx:1.11
或者
kubectl edit deployment/nginx-deployment
或者
kubectl apply -f nginx-deployment.yaml
# 实时观察发布状态：
kubectl rollout status deployment/nginx-deployment
# 查看deployment历史修订版本
kubectl rollout history deployment/nginx-deployment
kubectl rollout history deployment/nginx-deployment --revision=3
# 回滚到以前版本
kubectl rollout undo deployment/nginx-deployment
kubectl rollout undo deployment/nginx-deployment --to-revision=3
# 扩容deployment的Pod副本数量
kubectl scale deployment nginx-deployment --replicas=10
# 设置启动扩容/缩容
kubectl autoscale deployment nginx-deployment --min=10 --max=15 --cpu-percent=80
```

容器集群管理之Kubernetes

YAML配置文件管理对象

发布服务：

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

清理：

```
# kubectl delete -f nginx-deployment.yaml
# kubectl delete -f nginx-service.yaml
```

容器集群管理之Kubernetes

Pod管理

- Pod状态
- 创建/查询/更新/删除
- 重启策略
- 健康检查
- 数据持久化和共享
- hostPort
- 问题定位

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Pod管理-Pod状态

Pending：Pod已经被系统接收，准备下载镜像。
Running：Pod已经分配到节点，并且所有容器已创建。至少有一个容器仍在运行、正在启动或重新启动。
Successed：Pod中的所有容器已成功终止，不会重新启动。
Failed：Pod中的所有容器已终止，并且至少有一个容器已经终止。也就是说，容器退出非零状态或被系统终止。
Unknown：由于某种原因，无法获得Pod状态，通常是由于与Pod所在主机通信时出现错误。

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Pod管理-创建/查询/更新/删除

创建Pod对象：

```
# cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-test
  labels:
     os: centos
spec:
  containers:
  - name: hello
    image: centos:6
    env:
    - name: Test
      value: "123456"
    command: ["bash","-c","while true;do date;sleep 1;done"]
```

容器集群管理之Kubernetes

Pod管理-创建/查询/更新/删除

基本管理：

```
# 创建pod资源
kubectl create -f pod.yaml
# 查看pods
kubectl get pods pod-test
# 查看pod描述
kubectl describe pod pod-test
# 替换资源
kubectl replace -f pod.yaml -force
# 删除资源
kubectl delete pod pod-test
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Pod管理-重启策略

支持三种策略：
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：当容器终止退出，从不重启容器。

示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-test
  labels:
     test: centos
spec:
  containers:
  - name: hello
    image: centos:6
    command: ["bash","-c","while true;do date;sleep 1;done"]
  # 重启策略
  restartPolicy: OnFailure
```

容器集群管理之Kubernetes

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，然后根据Pod的重启策略来决定是否重启。
- readinessProbe
  如果检查失败，Kubernetes会把Pod从服务代理的分发后端剔除。

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:1.10
    ports:
    - containerPort: 80
    livenessProbe:
      httpGet:
        path: /index.html
        port: 80
```

容器集群管理之Kubernetes

Pod管理-数据持久化和共享

示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-test
  labels:
     test: centos
spec:
  containers:
  # 第一个容器
  - name: hello-write
    image: centos:6
    command: ["bash","-c","for i in {1..1000};do echo $i >> /data/hello;sleep 1;done"]
  # 第二个容器
  - name: hello-read
    image: centos:6
    command: ["bash","-c","for i in {1..1000};do cat /data/hello;sleep 1;done"]
    volumeMounts:
      - name: data
        mountPath: /data
  # 数据卷
  volumes:
  - name: data
    hostPath:
      path: /data
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Pod管理-hostPort

示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.10
    ports:
    - name: http
      containerPort: 80
      hostIP: 0.0.0.0
      hostPort: 80
      protocol: TCP
    - name: https
      containerPort: 443
      hostIP: 0.0.0.0
      hostPort: 443
      protocol: TCP
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Pod管理-问题定位

```
# 查看事件
kubectl get event
# 查看描述信息
kubectl describe pod my-pod
# 查看日志
kubectl logs log-pod -c container1
# 进容器终端
kubectl exec -it my-pod /bin/bash
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Service

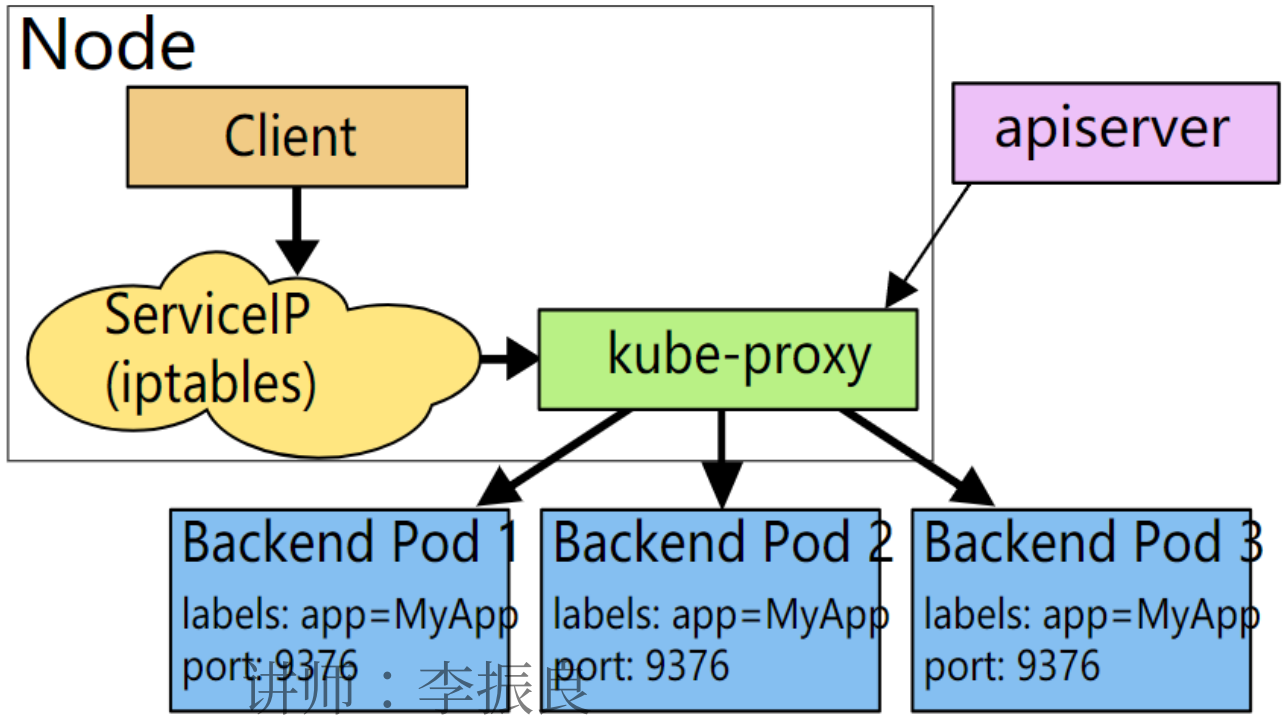
- 网络代理模式
- 服务代理
- 集群IP地址
- 发布服务
- 服务发现

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

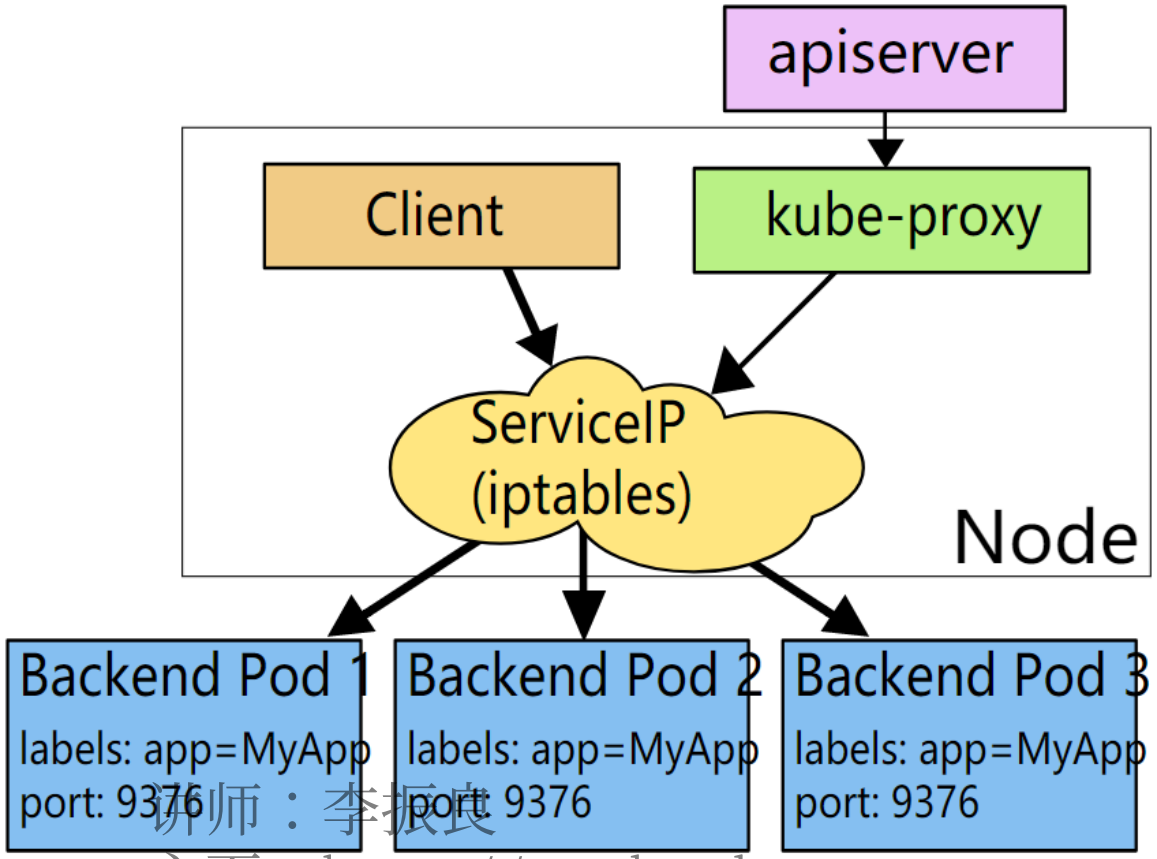
Service-网络代理模式

kube-proxy v1.0中只支持userspace模式，在v1.1中，添加了iptables代理，在v1.2开始iptables是默认的。



容器集群管理之Kubernetes

Service-网络代理模式



容器集群管理之Kubernetes

Service-服务代理

示例：

```
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
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Service-集群IP地址

示例：

```
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
  clusterIP: "10.10.10.123"
```

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

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。
- ExternalName
  通过CNAME将Service与externalName的值映射。要求kube-dns的版本为v1.7+。

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Service-发布服务

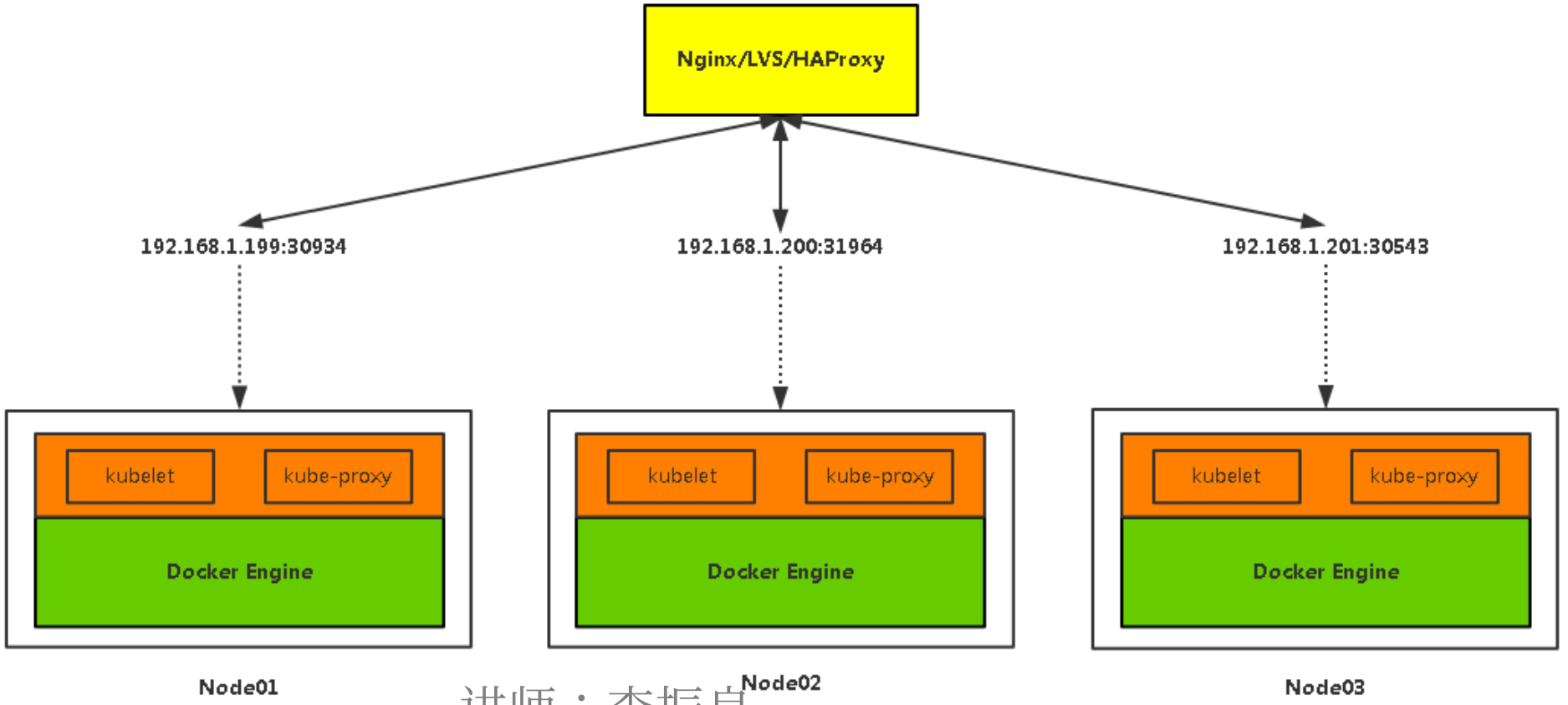
NodePort示例：

```
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
    nodePort: 30001
clusterIP: "10.10.10.123"
type: NodePort
```

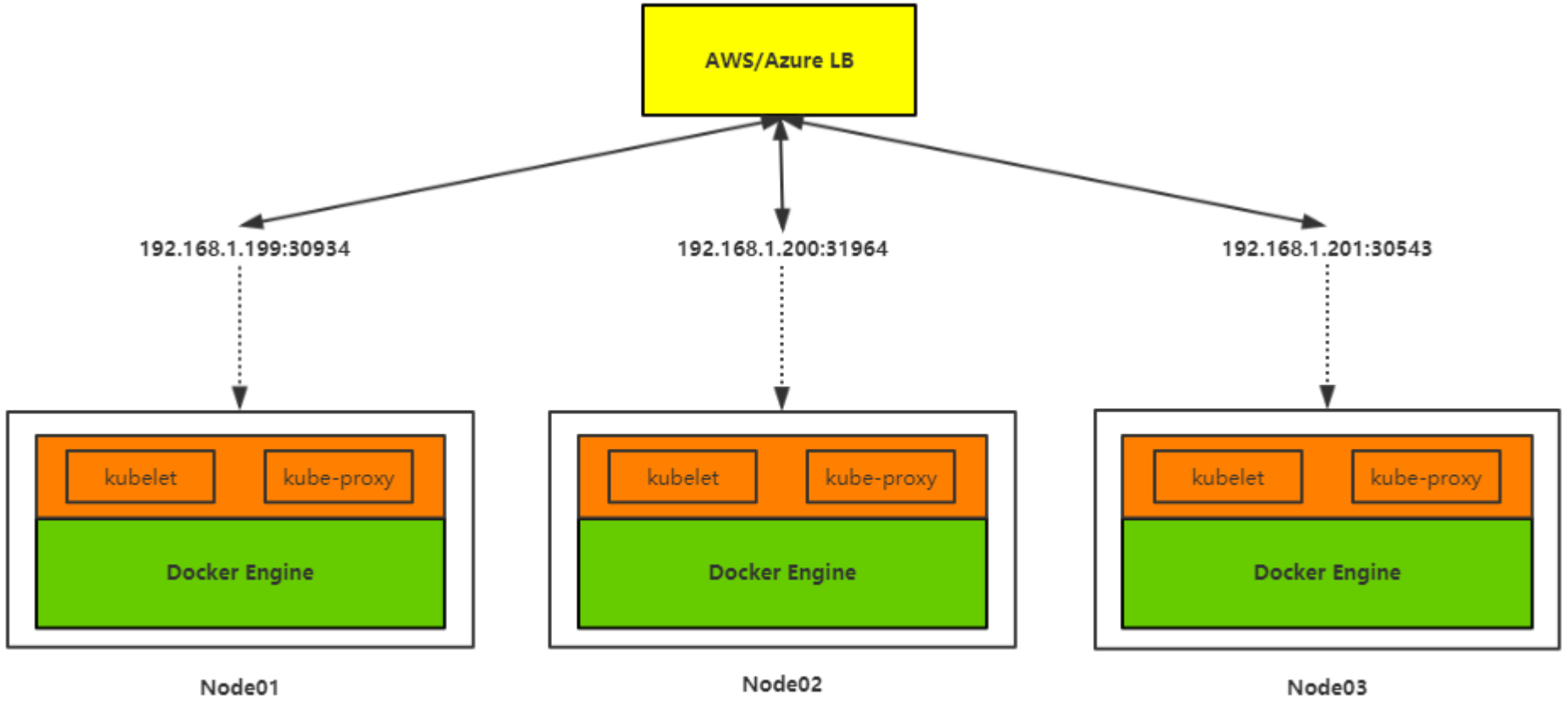
容器集群管理之Kubernetes

Service-发布服务



容器集群管理之Kubernetes

Service-发布服务



讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

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。

容器集群管理之Kubernetes

Volume

- 本地数据卷
- 网络数据卷
- 信息数据卷

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Volume-本地数据卷

- emptyDir
  当Pod分配到Node时，首先创建一个空卷，并挂载到Pod中的容器。Pod中的容器可以读取和写入卷中的文件。
  当Pod从节点中删除emptyDir时，该数据也会被删除。
- hostPath
  一个hostPath卷挂载Node文件系统上的文件或目录到Pod中的容器。

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Volume-本地数据卷

emptyDir示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: gcr.io/google_containers/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

hostPath示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: gcr.io/google_containers/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      path: /data
      type: Directory
```

容器集群管理之Kubernetes

Volume-网络数据卷

- nfs
- iscsi
- glusterfs
- awsElasticBlockStore
- cephfs
- azureFileVolume
- azureDiskVolume
- vsphereVolume
- ...

讲师：李振良
主页：http://opsdev.ke.qq.com

容器集群管理之Kubernetes

Volume-网络数据卷

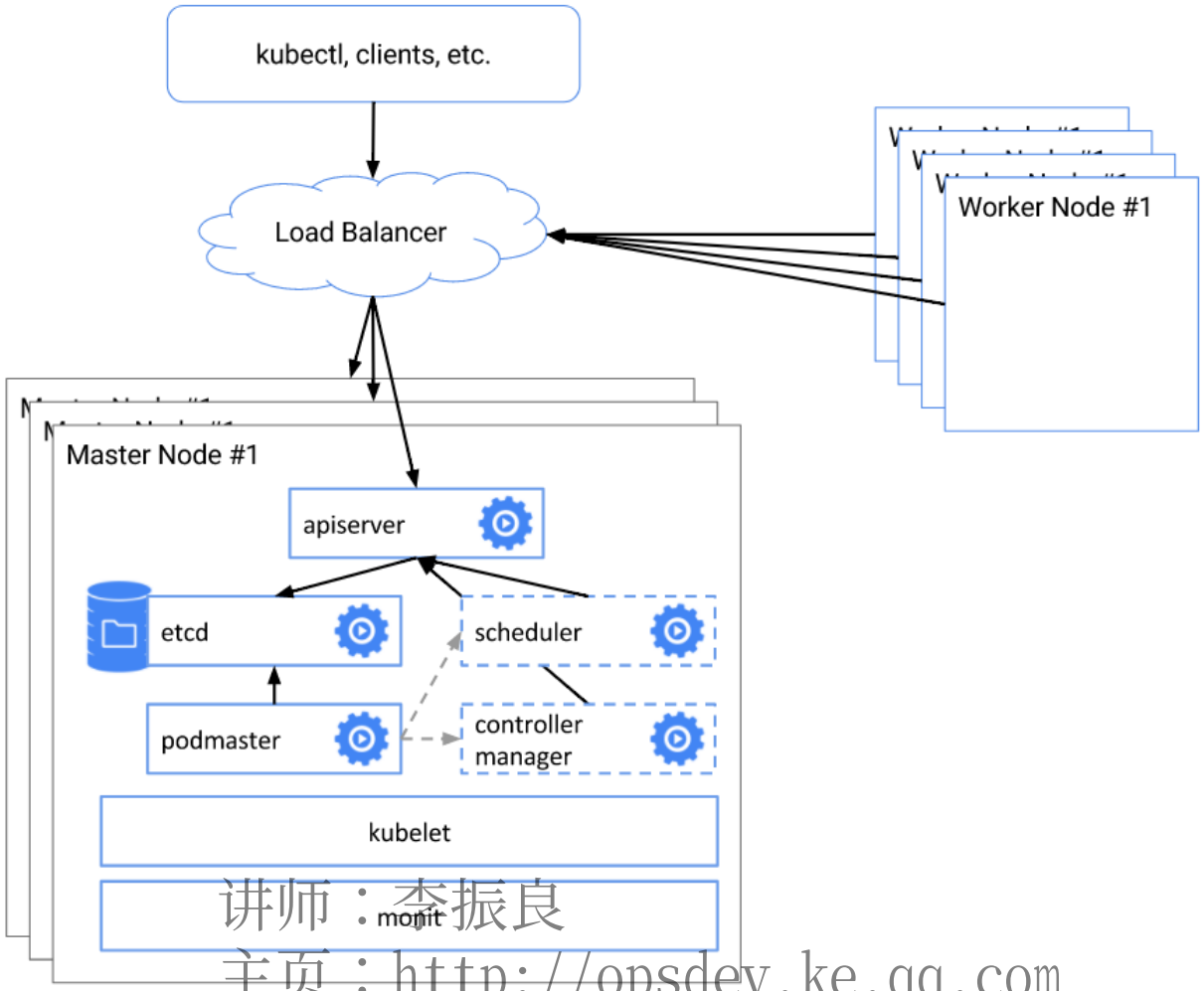
NFS示例：

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.10
        volumeMounts:
        - name: wwwroot
          mountPath: /var/www/html
        ports:
        - containerPort: 80
      volumes:
      - name: wwwroot
        nfs:
          server: 192.168.1.200
          path: /opt/wwwroot
```

讲师：李振良
主页：http://opsdev.ke.qq.com

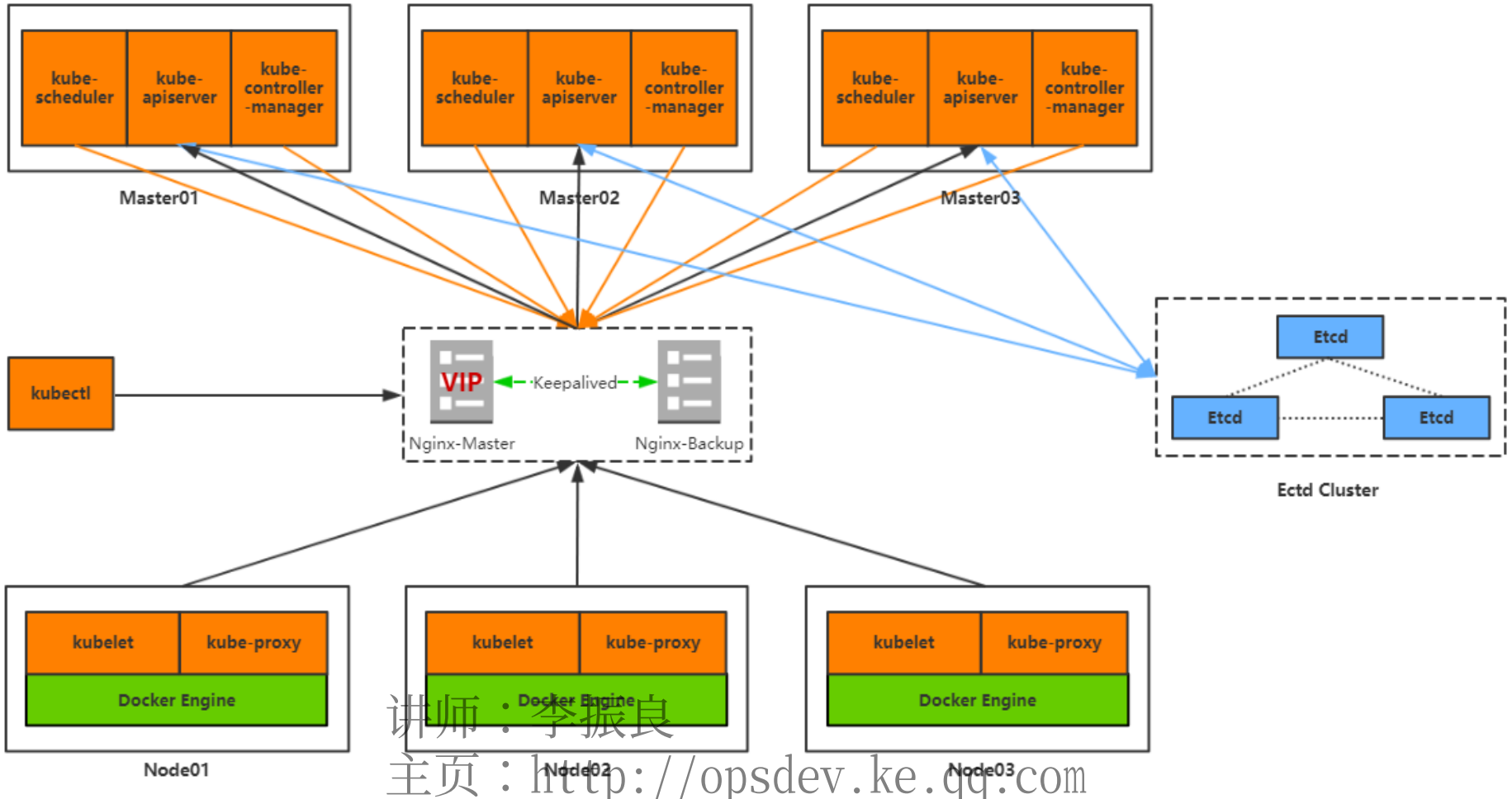
容器集群管理之Kubernetes

高可用性设计



容器集群管理之Kubernetes

高可用性设计



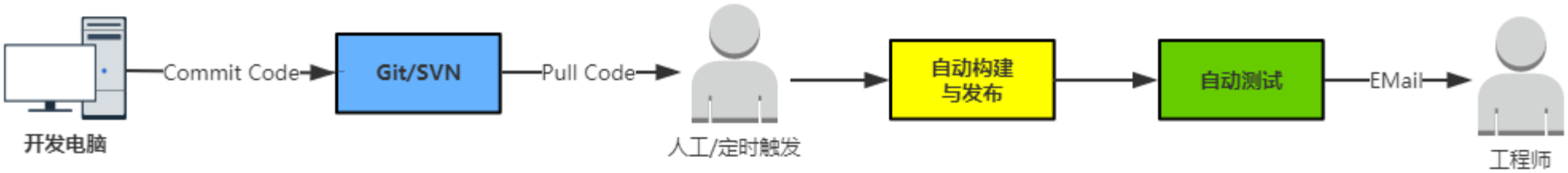
Docker结合Jenkins构建持续集成环境

- 持续集成是什么
- 持续集成相关工具
- Docker+Jenkins+Maven+SVN搭建持续集成环境

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Jenkins构建持续集成环境

持续集成是什么



持续：完成一个新功能就向下一个环节交付，不断发现问题，解决问题。
集成：研发人员提交新代码到主干仓库，进行构建、部署、测试，不断做集成，修正集成结果。
部署：将项目发布到测试环境、预生产环境或生产环境。
交付：将最终产品发布到预生产环境或生产环境，给用户使用。

持续集成（Continuous Integration）：代码合并、构建、部署、测试都在一起，不断地执行这个过程，并对结果反馈。
持续交付（Continuous Delivery）：将最终产品发布到生产环境，给用户使用。
持续部署（Continuous Deployment）：将新需求部署到生产环境。

Docker结合Jenkins构建持续集成环境

持续集成相关工具

Jenkins ：一个开源的持续集成工具，提供软件版本发布、自动测试等一系列流程及丰富的插件。

Maven：一个自动化构建工具，通过一段描述来管理项目的构建，比如编译、打包等逻辑流程。

SVN/Git：源代码版本管理工具。

Docker：容器化技术；打包项目环境与快速部署。

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Jenkins构建持续集成环境

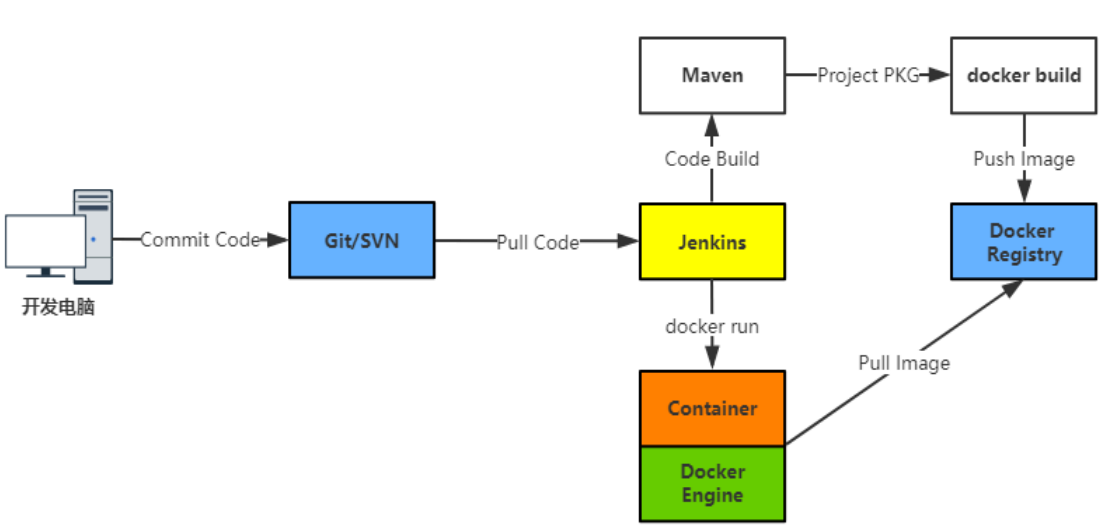
Docker+Jenkins+Maven+SVN搭建持续集成环境

- 发布流程设计
- SVN安装配置及简单使用
- Docker镜像仓库搭建
- 部署节点安装Docker与Docker-Compose及配置普通用户sudo
- Jenkins安装
- Jenkins基本配置
- Jenkins创建项目
- 测试

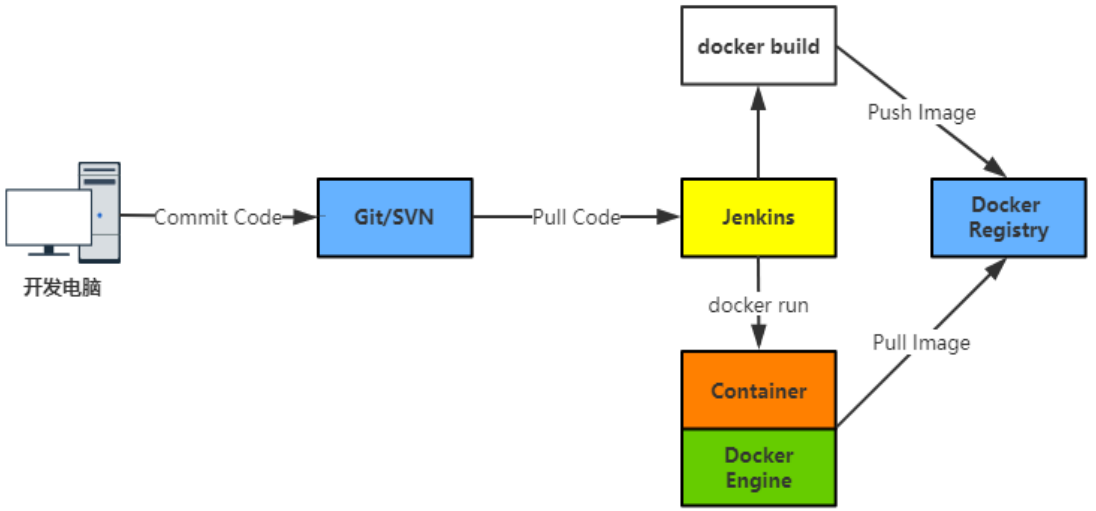
讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Jenkins构建持续集成环境

发布流程设计



JAVA环境CI

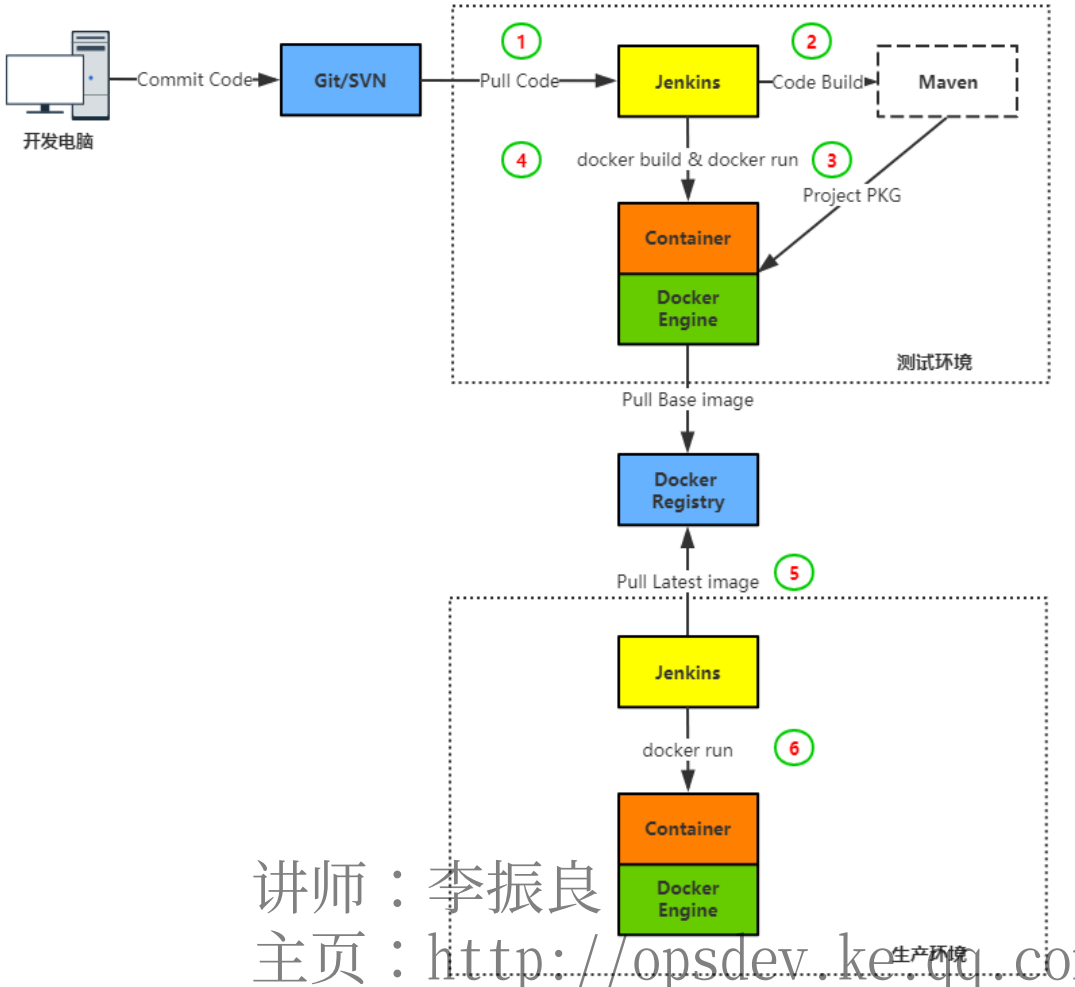


PHP环境CI

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Jenkins构建持续集成环境

发布流程设计



Docker结合Jenkins构建持续集成环境

环境说明

服务器	IP
Jenkins	192.168.0.151
测试环境	192.168.0.152
生产环境	192.168.0.153

操作系统：Ubuntu16.04_x64

Maven3.5

Tomcat8

JDK1.8

Jenkins2.7

Docker CE 17.06

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Jenkins构建持续集成环境

SVN安装配置及简单使用

```
# apt-get install subversion
# mkdir /home/svn
# svnadmin create /home/svn/repos
# vi /home/svn/repos/conf/svnserve.conf
anon-access = none
auth-access = write
password-db = passwd
authz-db = authz
# vi /home/svn/repos/conf/passwd
[users]
test = 123456
# vi /home/svn/repos/conf/authz
[repos:/]
test = rw
# svnserve -d -r /home/svn
```

仓库地址：svn://192.168.1.151/repos

Docker结合Jenkins构建持续集成环境

Docker镜像仓库搭建

```
# docker run -d \
-v /opt/registry:/var/lib/registry \
-p 5000:5000 \
--restart=always \
--name registry \
registry
```

在部署节点配置Docker可信任私有仓库：

```
# vi /etc/docker/daemon.json
{"registry-mirrors": ["http://04be47cf.m.daocloud.io"],"insecure-registries":["192.168.0.151:5000"]}
```

上传基础镜像到私有仓库：

```
# docker build -t 192.168.0.151:5000/lnmp-nginx:base .
# docker build -t 192.168.0.151:5000/lnmp-php:base .
# docker push 192.168.0.151:5000/lnmp-nginx:base
# docker push 192.168.0.151:5000/lnmp-php:base
```

Docker结合Jenkins构建持续集成环境

部署节点安装Docker与Docker-Compose及配置普通用户sudo

1.安装Docker
安装文档：https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu
2.安装Docker-Compose
```
# curl -L https://github.com/docker/compose/releases/download/1.14.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
# chmod +x /usr/local/bin/docker-compose
```
3.赋予用户sudo权限
```
# vi /etc/sudoers
user ALL=(ALL) NOPASSWD:ALL
```

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Jenkins构建持续集成环境

Jenkins安装

安装包下载：
http://mirrors.jenkins.io/war-stable/
http://maven.apache.org/download.cgi
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

配置JDK和Maven环境变量：

```
# tar zxvf jdk-8u45-linux-x64.tar.gz
# mv jdk1.8.0_45 /usr/loca/jdk.18
# tar apache-maven-3.5.0-bin.tar.gz
# mv apache-maven-3.5.0 /usr/local/maven3.5
# vi /etc/profile
JAVA_HOME=/usr/local/jdk1.8
CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
MAVEN_HOME=/usr/local/maven3.5
PATH=$JAVA_HOME/bin:$MAVEN_HOME/bin:$PATH
export JAVA_HOME CLASSPATH MAVEN_HOME PATH
# tar zxvf apache-tomcat-8.0.46.tar.gz
# cd apache-tomcat-8.0.46/webapps
# rm -rf ./*
# unzip /root/jenkins.war -d ROOT
# ../bin/startup.sh
```

Docker结合Jenkins构建持续集成环境

Jenkins基本配置与创建项目

系统管理->系统设置：主要配置workspace目录，全局环境变量，邮件通知，其他插件配置等。
系统管理->Global Tool Configuration：主要配置JDK、Maven等工具。

在系统设置里面先配置好SSH连接各个部署节点信息，在创建项目中使用。

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Consul实现服务发现

- Consul是什么
- Consul集群介绍
- Consul集群部署
- Docker+Registrator+Consul实现Nginx集群节点自动加入

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Consul实现服务发现

Consul是什么

Consul是一个分布式、高可用性，在基础设施中发现和配置服务的工具。
主要功能：

- **服务发现**

  通过DNS或HTTP接口使得消费者发现服务，应用程序可以轻松找到所依赖的服务。

- **健康检查**

  防止将请求转发不健康的主机。

- **键值存储**

  可以使用分层键/值存储，比如功能标记、动态配置等。

- **多数据中心**

  开箱即用，不需要复杂的配置。这就意味这不用建立抽象的逻辑来扩展多个地区。

Docker结合Consul实现服务发现

Consul集群介绍

1）Consul agent是Consul核心工作，分为client和server两种工作模式。默认以client模式运行，提供服务注册、健康检查、转发查询给server leader。server模式启动时使用-server选项指定，用于维护Consul集群状态、Raft协议进行选举。

2）agent必须在每个Consul节点运行，所有运行Consul agent节点构成Consul集群。

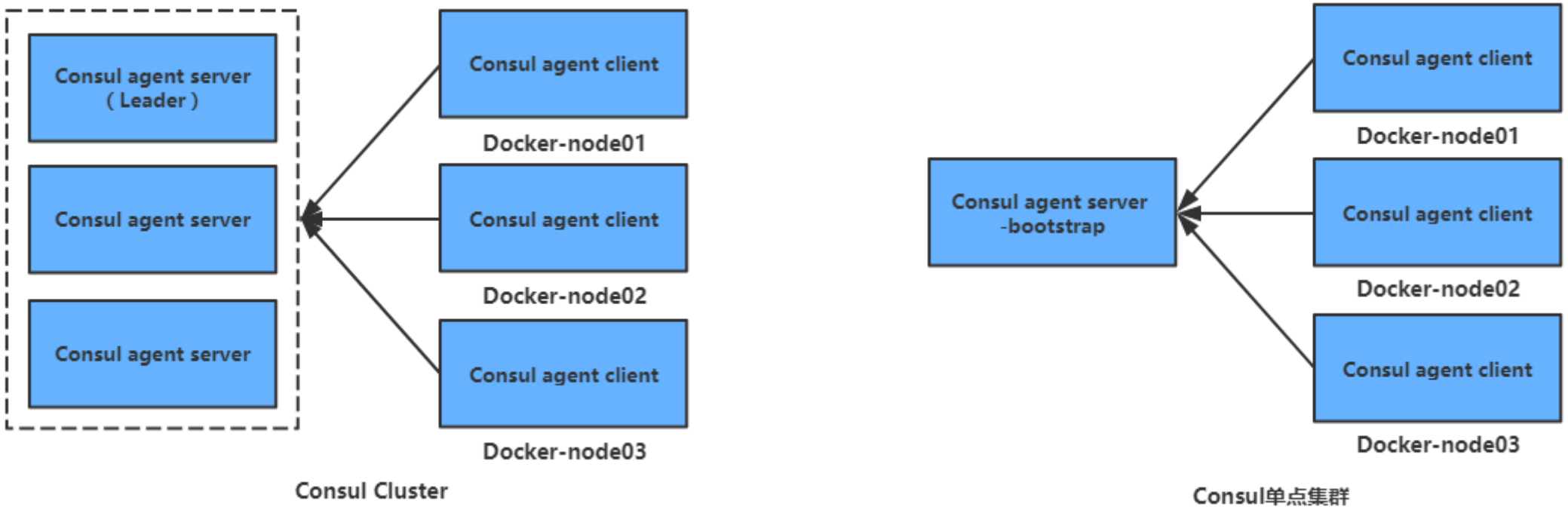
3）官方建议Consul集群至少3或5个节点运行Consul agent server模式，client节点不限。

4）通过join或rejoin选项加入集群。一旦加入，集群信息使用gossip算法同步到整个集群节点。

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Consul实现服务发现

Consul集群部署



讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Consul实现服务发现

Consul集群部署

下载二进制Consul包：https://www.consul.io/downloads.html

安装二进制包：

```
# unzip consul_0.9.2_linux_amd64.zip
# mv consul /usr/bin
```

server模式初始化集群：

```
# consul agent \
-server \
-bootstrap \
-ui \
-data-dir=/var/lib/consul-data \
-bind=192.168.1.198 \
-client=0.0.0.0 \
-node=node01
```

client模式加入集群：

```
# docker run -d \
    --name=consul \
    -p 8301:8301 \
    -p 8301:8301/udp \
    -p 8500:8500 \
    -p 8600:8600 \
    -p 8600:8600/udp \
    --restart=always \
    progrium/consul \
    -join 192.168.1.198 -advertise 192.168.1.199 -client 0.0.0.0 -node=node02
```

Docker结合Consul实现服务发现

Consul集群部署

多个server节点配置

server模式初始化集群：

```
# docker run -d \
    --name=consul \
    -p 8300:8300 \
    -p 8301:8301 \
    -p 8301:8301/udp \
    -p 8500:8500 \
    -p 8600:8600 \
    -p 8600:8600/udp \
    --restart=always \
    progrium/consul \
    -server -bootstrap -ui -advertise
192.168.1.198 -client 0.0.0.0 -node=node01
```

server模式加入集群：

```
# docker run -d \
    --name=consul \
    -p 8301:8301 \
    -p 8301:8301/udp \
    -p 8500:8500 \
    -p 8600:8600 \
    -p 8600:8600/udp \
    --restart=always \
    progrium/consul \
    -server -join 192.168.1.198 -advertise 192.168.1.199 -client 0.0.0.0 -node=node02
```

Docker结合Consul实现服务发现

Consul集群部署

选项	描述
-advertise	通告地址
-bind	集群节点之间通信地址
-bootstrap	设置服务器为bootstrap模式。在一个dc中只有一个server处于bootstrap模式。一般初始化第一台Consul时指定，自选举为leader。
-bootstrap-expect	在一个dc中期望提供server节点数目，consul会一直等到指定的server数目才会引导整个集群，选举leader。不能与boostrap同时用。
-client	设置客户端访问地址，包括RPC、DNS。默认127.0.0.1
-config-file	从JSON配置文件中读取
-data-dir	指定存放agent server集群状态目录，以免系统重启丢失
-dc	数据中心名称，默认dc1
-http-port	HTTP API监听端口
-join	加入一个已经启动的agent，可以指定多个agent地址
-node	节点名称，必须在集群中唯一的。默认是主机名
-rejoin	忽略先前的离开，再次启动后尝试加入集群
-server	切换agent模式到server模式。每个集群至少有一个server
-ui	启用内置的Web UI
-ui-dir	Web UI的资源目录

Docker结合Consul实现服务发现

Consul集群部署

查看集群信息：

```
consul members
consul info |grep leader
consul catalog services
```

通过HTTP API获取集群信息：

```
curl 127.0.0.1:8500/v1/status/peers    # 集群server成员
curl 127.0.0.1:8500/v1/status/leader   # 集群Raft leader
curl 127.0.0.1:8500/v1/catalog/services # 注册的所有服务
curl 127.0.0.1:8500/v1/catalog/services/nginx  # 服务信息
curl 127.0.0.1:8500/v1/catalog/nodes   # 集群节点详细信息
```

类型	描述
kv	键值存储
agent	agent管理
catalog	nodes和services管理
health	健康检查
session	session管理
acl	ACL管理
event	用户事件
status	Consul系统状态

服务注册：

```
curl -X PUT -d \
'{"id": "jetty","name": "service_name","address": "192.168.1.200","port": 8080,"tags": ["test"],"checks":
[{"http": "http://192.168.1.200:8080/","interval": "5s"}]}' \
http://127.0.0.1:8500/v1/agent/service/register
```

Docker结合Consul实现服务发现

Docker+Registrator+Consul实现Nginx集群节点自动加入

https://github.com/hashicorp/consul-template
https://releases.hashicorp.com/consul-template/0.19.3/consul-template_0.19.3_linux_amd64.zip

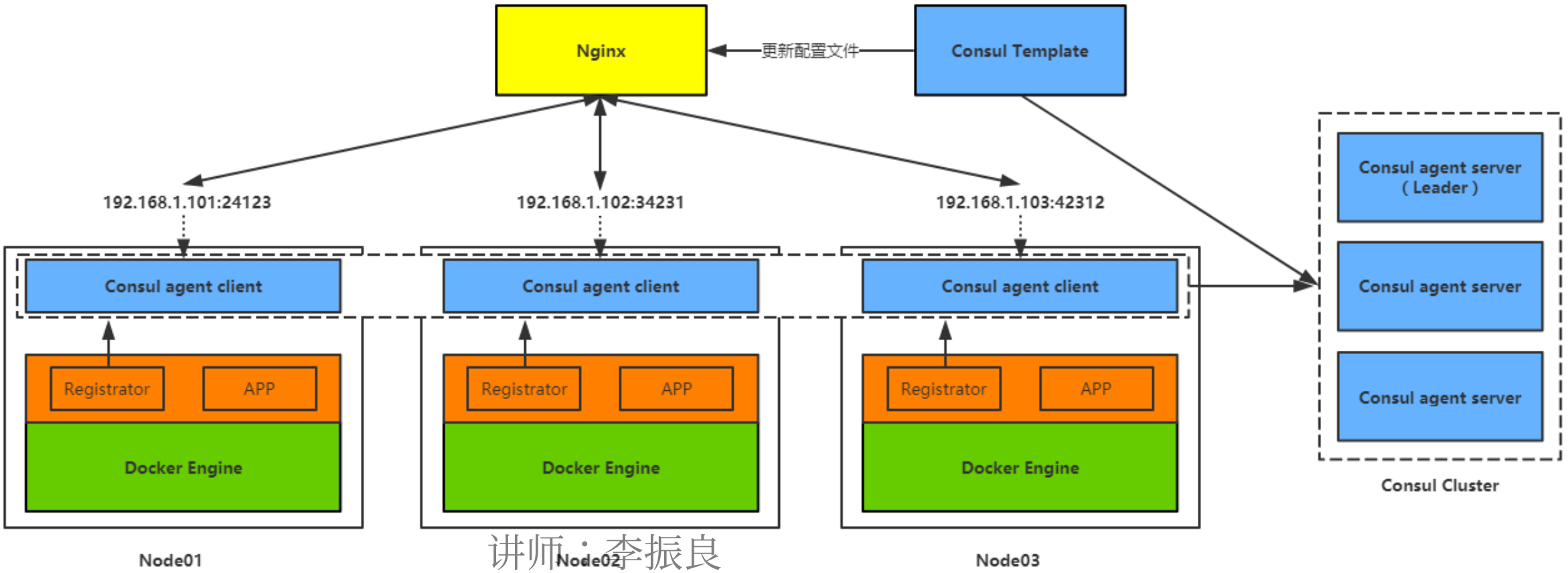
consul-template：一个守护程序，用于实时查询consul集群数据，并更新文件系统上的任意数量的指定模板，生成配置文件，更新完成后可以选择运行任何命令。

gliderlabs/registrator：检查容器运行状态自动注册到服务中心。registrator支持服务注册有consul、etcd、SkyDNS2。

讲师：李振良
主页：http://opsdev.ke.qq.com

Docker结合Consul实现服务发现

Docker+Registrator+Consul实现Nginx集群节点自动加入



Docker结合Consul实现服务发现

Docker+Registrator+Consul实现Nginx集群节点自动加入

Node节点启动注册服务：

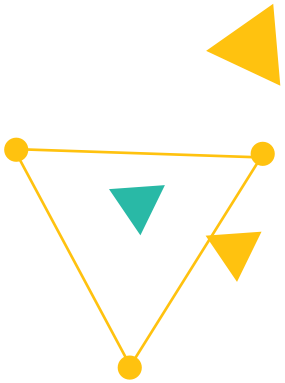
```
# docker run -d \
--name=registrator \
-v /var/run/docker.sock:/tmp/docker.sock \
--restart=always \
gliderlabs/registrator:latest \
consul://<Docker Host IP>:8500
```

讲师：李振良
主页：http://opsdev.ke.qq.com

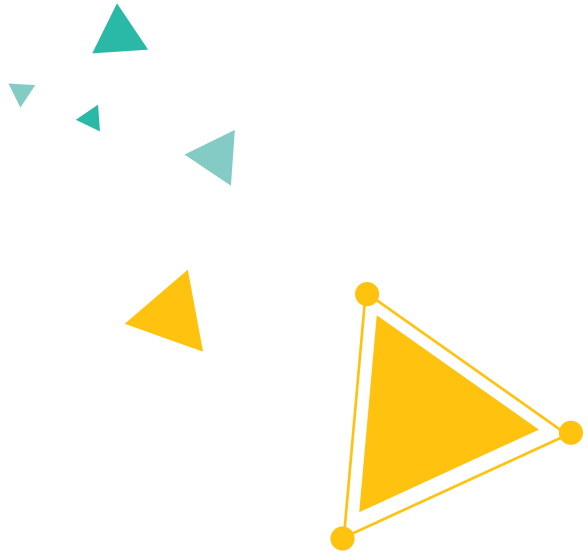
Docker结合Consul实现服务发现

Docker+Registrator+Consul实现Nginx集群节点自动加入

```
# vi nginx.ctmpl
upstream http_backend {
    ip_hash;
    {{range service "nginx"}}
    server {{ .Address }}:{{ .Port }};
    {{ end }}
}
server {
    listen 80;
    server_name localhost;
    location / {
         proxy_pass http://http_backend;
    }
}
# consul-template \
-consul-addr <Consul server>:8500 \
-template "./nginx.ctmpl:/usr/local/nginx/conf/nginx.conf:/usr/local/nginx/sbin/nginx -s reload" \
-log-level=info
```



谢谢



Python运维开发群：323779636
Docker技术交流群：516039855

讲师：李振良
主页：http://opsdev.ke.qq.com