The 8th China Cloud Computing Conference
第八届中国云计算大会
技术融合 应用创新

# 用友基于Docker构建高可用系统的实践

綦玉冰
qiyb@yonyou.com
用友企业互联网开放平台


iuap.yonyou.com
yonyou
iUAP
支持新一代计算架构的
企业互联网开放平台
开放的
互联网
技术平台
通用的
应用平台
DevOps
运维平台
混合云
集成平台
SaaS服务
iWEB前端技术框架
互联网中间件
支持多端
支撑互联网应用、
SaaS类应用、
运营服务
互联网类运维工具
智能、开放、可视化
Portal、ESB、BPM、
MDM、IdM等
互联网连接服务
云审
OpenAPI

# 传统企业进军互联网面临的问题与挑战

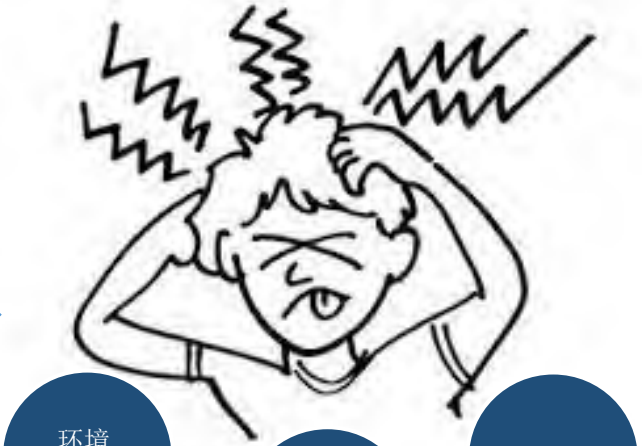
模块解耦

横向扩展

调用灵活

容错
高可用

可独立部署

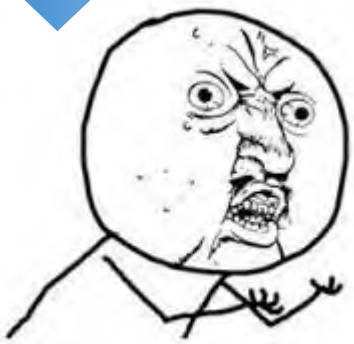
环境复杂

模块多

依赖多

频繁升级

路由混乱



服务挂掉

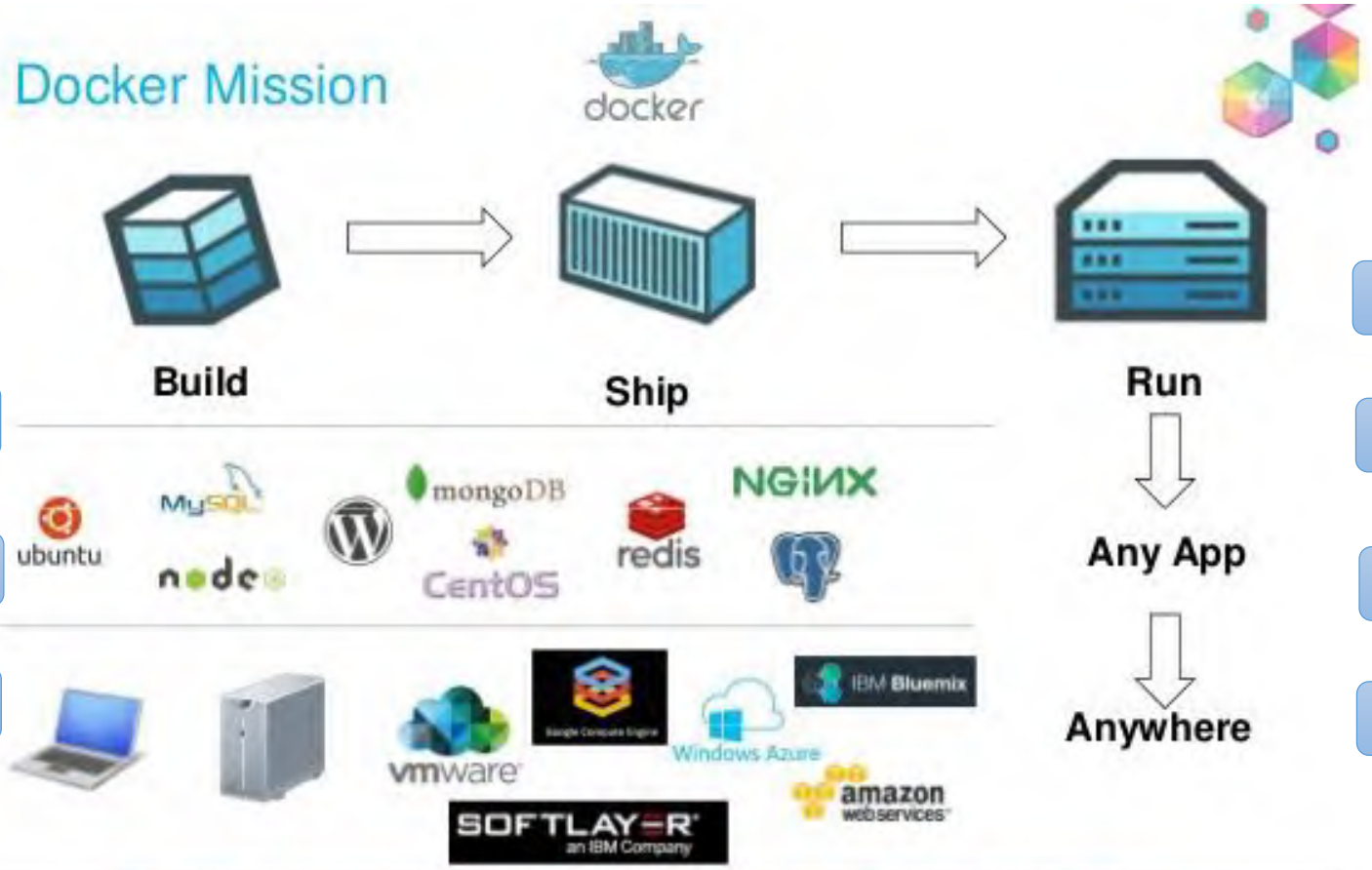
## Docker带来了什么？

Lxc虚拟化

启动速度快

镜像分层

CI环境构建




提升性能，节约成本

快速部署

更快的扩容缩容

测试生产环境一致

# 为何基于Docker可以实现微服务架构

镜像是应用的唯一单元

- 应用包本身
- 配置环境
- 运行环境
- 运行依赖包
- 操作系统发行版
- 操作系统内核

# 高可用系统需要具备哪些能力？

## 弹性伸缩

自动伸缩 服务发现 负载均衡

Docker模式

- 镜像文件
- 服务编排
- 镜像仓库

Native模式

- 中间件
- 网络管理
- 应用构建仓库

## 运维监控

系统监控

- 主机管理
- 集群管理
- 中间件安装
- 配置管理
- 批量命令
- 发布部署

异常报警

## 性能分析

日志收集

- 应用日志
- 中间件日志
- 系统日志
- 安全日志

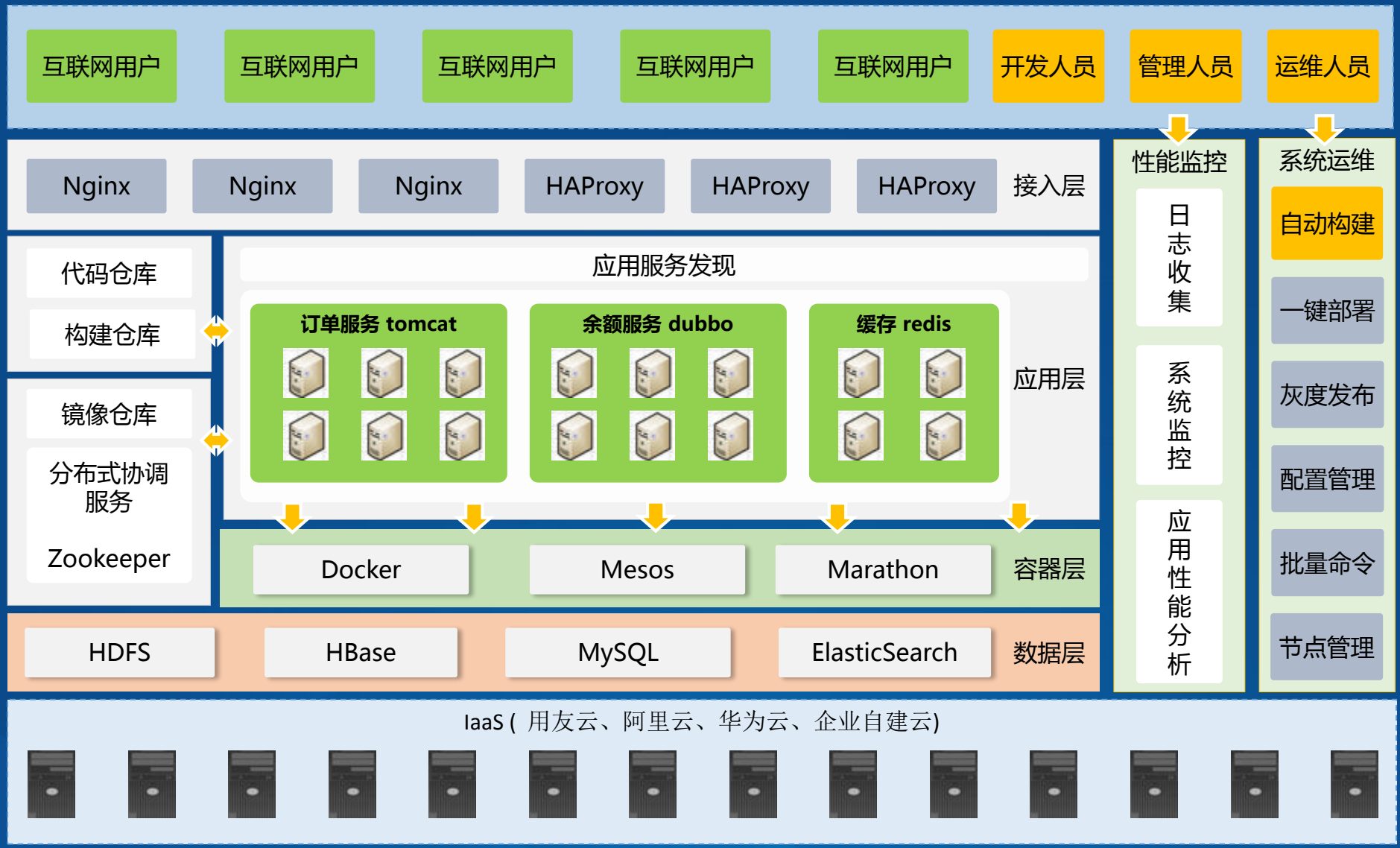
海量索引

应用性能

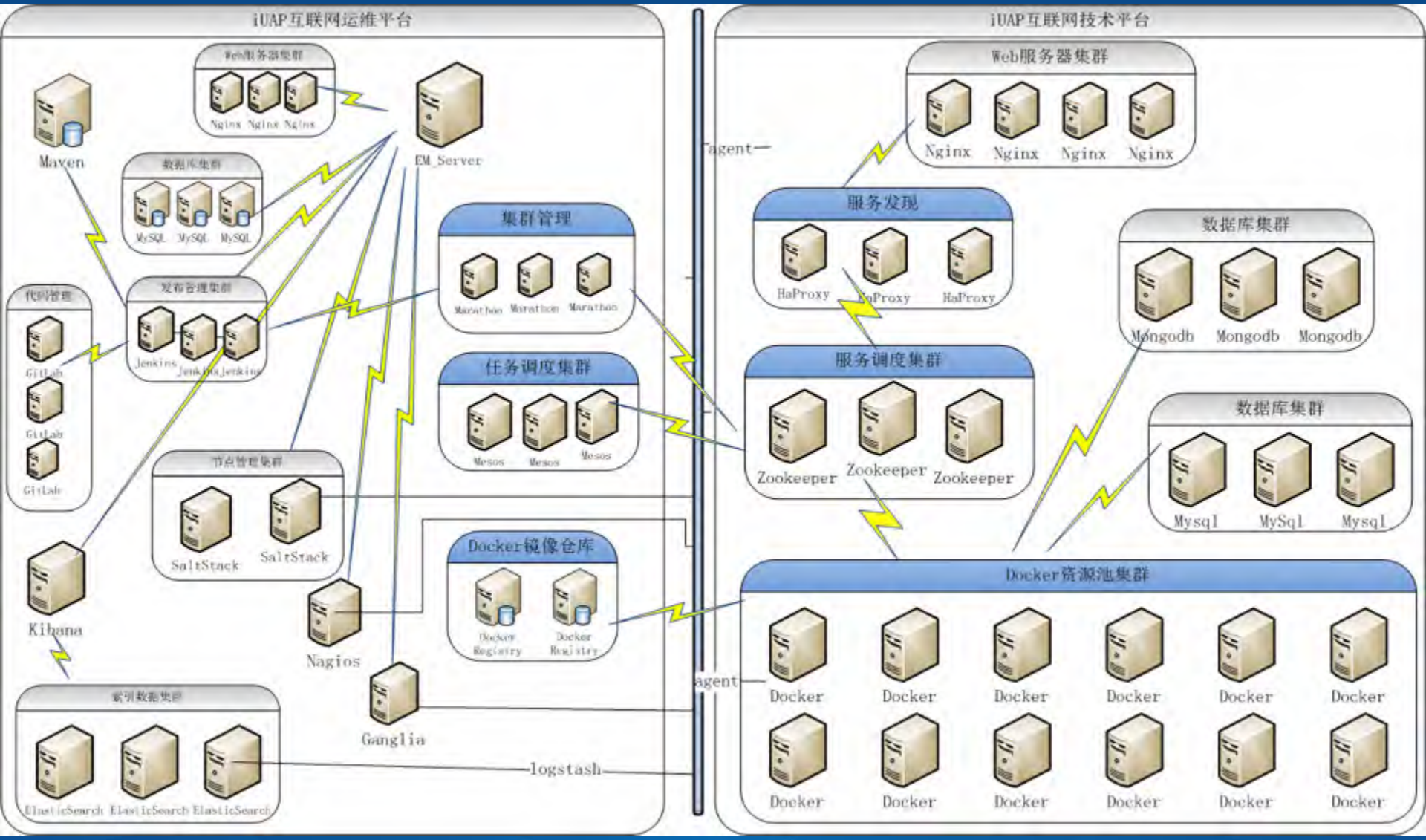
- 应用拓扑
- 调用链路
- 错误详情
- 内存快照

大数据存储

# 总体架构设计


互联网用户
互联网用户
互联网用户
互联网用户
互联网用户
开发人员
管理人员
运维人员
Nginx
Nginx
Nginx
HAProxy
HAProxy
HAProxy
接入层
代码仓库
构建仓库
镜像仓库
分布式协调服务
Zookeeper
应用服务发现
订单服务 tomcat
余额服务 dubbo
缓存 redis
应用层
Docker
Mesos
Marathon
容器层
HDFS
HBase
MySQL
ElasticSearch
数据层
性能监控
日志收集
系统监控
应用性能分析
系统运维
自动构建
一键部署
灰度发布
配置管理
批量命令
节点管理
IaaS ( 用友云、阿里云、华为云、企业自建云)

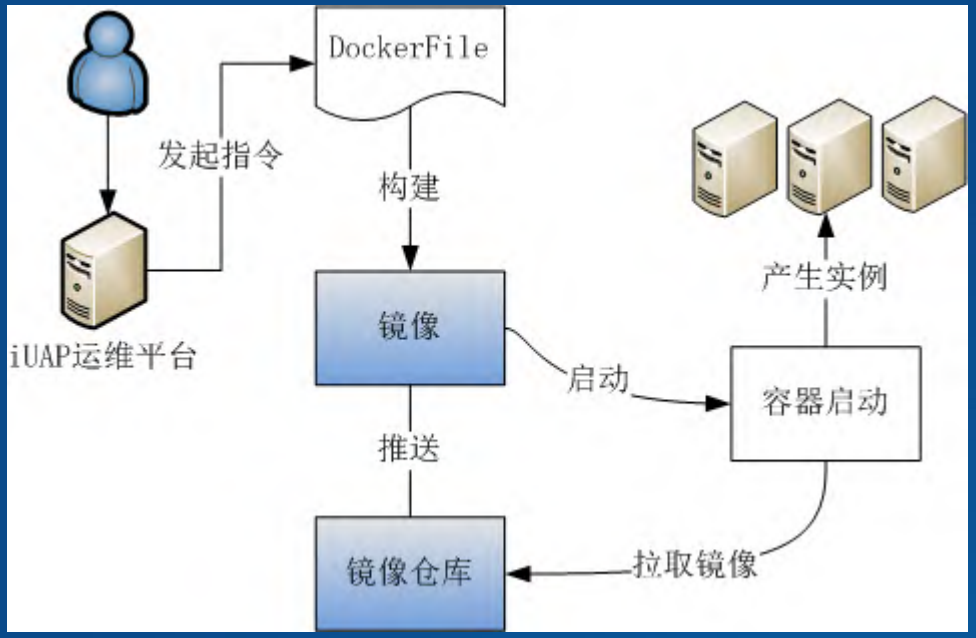
# 物理架构设计


iUAP互联网运维平台
Web服务器集群
Nginx Nginx Nginx
Maven
EM_Server
数据库集群
MySQL MySQL MySQL
集群管理
Marathon Marathon Marathon
发布管理集群
Jenkins Jenkins Jenkins
代码管理
GitLab
GitLab
GitLab
任务调度集群
Mesos Mesos Mesos
节点管理集群
SaltStack SaltStack
Docker镜像仓库
Docker Registry
Docker Registry
Kibana
Nagios
Ganglia
索引数据集群
ElasticSearch ElasticSearch ElasticSearch
logstash
agent
iUAP互联网技术平台
Web服务器集群
Nginx Nginx Nginx Nginx
服务发现
HaProxy HaProxy HaProxy
数据库集群
Mongodb Mongodb Mongodb
服务调度集群
Zookeeper Zookeeper Zookeeper
数据库集群
Mysql MySql Mysql
Docker资源池集群
Docker Docker Docker Docker Docker Docker
Docker Docker Docker Docker Docker Docker

# 总体流程设计


Maven
构建
仓库
Git
Jenkins
代码管理
发布管理
镜像仓库
任务调度集群
Mesos Master
Mesos Master
Mesos Master
系统监控 Ganglia
应用监控
业务监控
日志分析
统一收集 分布式存储
服务注册与发现
Ganglia Nagios

# 基于Docker模式的构建流程




DockerFile

```
# VERSION 0.0.1
# 默认ubuntu server长期支持版本，当前是12.04
FROM ubuntu
# 作者信息
MAINTAINER yonyou "enterprise_manager@yonyou.com"

# 更新源，安装ssh server
RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc
RUN apt-get update
RUN apt-get install -y openssh-server
RUN mkdir -p /var/run/sshd

# 设置root ssh远程登录密码为123456
RUN echo "root:123456" | chpasswd

# 添加orache java7源，一次性安装vim, wget, curl, java7, tomcat7等必备软件
RUN apt-get install python-software-properties
RUN add-apt-repository ppa:webupd8team/java
RUN apt-get update
RUN apt-get install -y vim wget curl oracle-java7-installer tomcat7

# 设置JAVA_HOME环境变量
RUN update-alternatives --display java
RUN echo "JAVA_HOME=/usr/lib/jvm/java-7-oracle" >> /etc/environment
RUN echo "JAVA_HOME=/usr/lib/jvm/java-7-oracle" >> /etc/default/tomcat7
```

build & tag

```
[root@uap-01~]# docker build -t  yonyou/tomcat  .
```

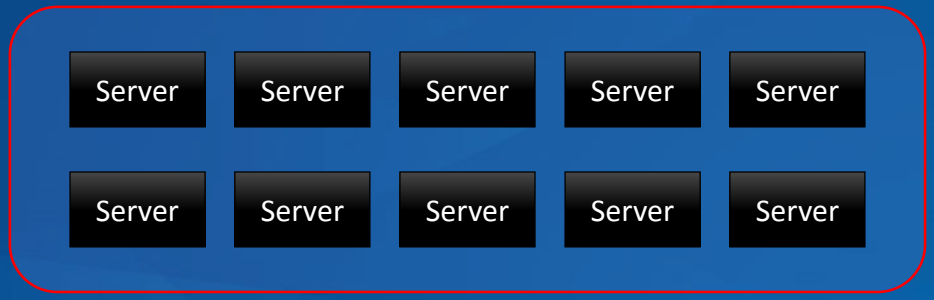
push

```
[root@uap-01~]# docker push  yonyou/tomcat :1.0
```

run & pull

```
[root@uap-01~]# docker run -d -p 8080:8080 yonyou/tomcat
[root@uap-01~]# docker run -d -p 8081:8080 yonyou/tomcat
... ...
[root@uap-01~]# docker run -d -p 8090:8080 yonyou/tomcat
```

由1台服务器，产生了10个新“服务器”（容器）



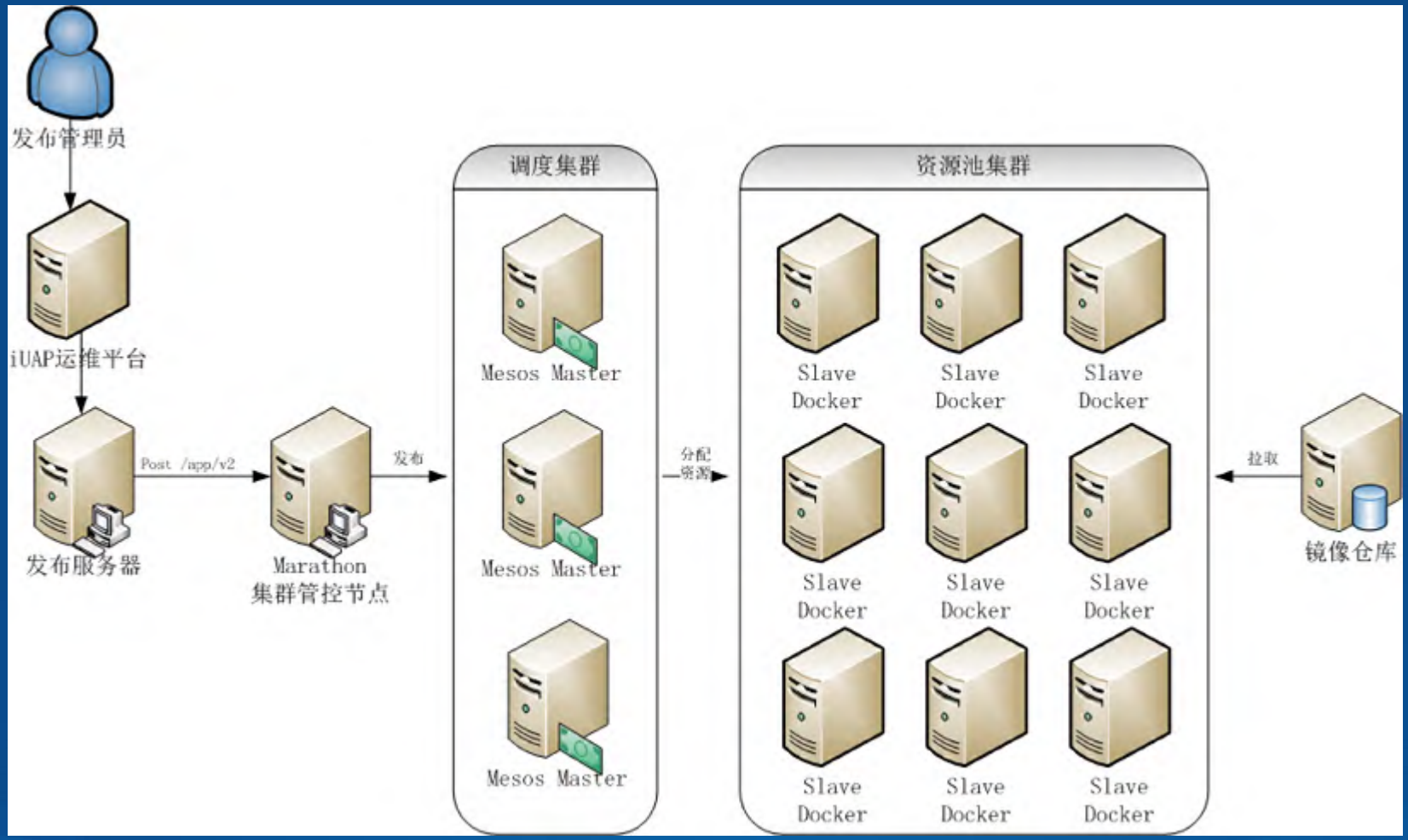
# 基于Docker模式的构建脚本

```
base_path=`pwd` #默认镜像文件存放地址
warName="$fileName-$version.war" #war包名称
warPATH="$base_path/$warName" #war包所在绝对路径
fileName="myapp" #应用名称
version="1.2.7" #war包版本号
tomcatName="apache-tomcat-8.0.28" #tomcat版本
dockerRegistry="172.20.13.70:5000" # 镜像仓库地址
app_path='/data/apps'  #制作镜像时的临时文件夹
resultFileName=$fileName  #制作镜像时候的app的临时文件夹
requestPath="$resultFileName"
imageName="$resultFileName" #app对应镜像名
targetNAME="$app_path/$resultFileName/$fileName" # war包解压到的目录
date_string=`date '+%Y%m%d'` #镜像制作时间戳

#============开始制作镜像==========
if [ ! -x  "$warPATH" ]; then
    echo "$warPATH not exist!!!!"
    echo -e "\033[31m=====================================镜像制作失败!....`date`\033[0m"
    exit 1
fi
echo "tar from $warPATH to $targetNAME"
unzip   -o -d $targetNAME $warPATH
#================================================docker开始制作镜像
docker build -t $imageName  "$app_path/$resultFileName/"
docker tag -f $imageName $dockerRegistry/$imageName
docker push $dockerRegistry/$imageName

tag_name=$dockerRegistry/$imageName:$date_string-$imageName
docker tag -f $imageName $tag_name
docker push $tag_name
```

# 应用向资源池发布如何做？



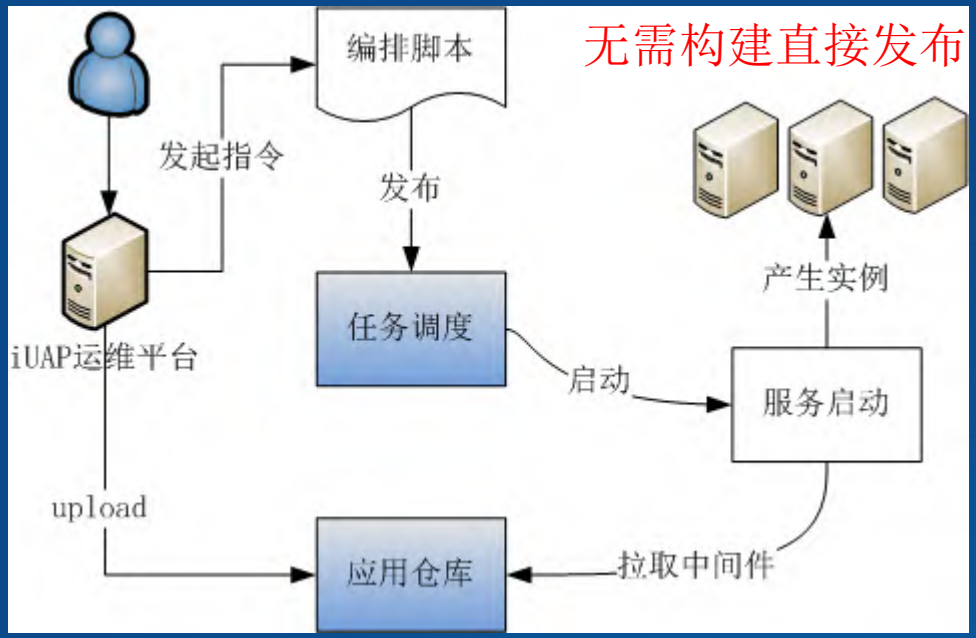
# 基于Docker模式的服务发布脚本

```
curl -X POST -d @- ${publish_server}/v2/apps?force=true --header "Content-Type:application/json" <<EOF
{
    "id": "/product/service/tomcat-example",
    "cpus": 0.1,
    "mem":512,
    "instances": 2,
    "container": {
      "type": "DOCKER",
      "volumes": [],
      "docker": {
        "image": "${docker_registry}/tomcat-example",
        "network": "BRIDGE",
        "portMappings": [
          {
            "containerPort": 8080,
            "hostPort": 0,
            "servicePort": 8888,
            "protocol": "tcp"
          }
        ],
      "privileged": false,
      "parameters": [],
      "forcePullImage": false
    }
  }
}
EOF
```

# 持续交付怎么做？


代码提交
Git/SVN
代码构建
Maven仓库
应用war包
容器编排
服务端口设定
应用版本号
镜像仓库
发布脚本
开始发布
Marathon 分发
Mesos Master
Mesos Slave
服务启动
服务发现
用户引流
Nginx 反向代理
域名接入
用户可访问

# 基于Native模式应用构建和发布流程




服务编排脚本

```
{
  "id": "tomcat",
  "cmd": "export JAVA_HOME=../jre-7u76-linux-x64 && mv *.war
  "mem": 256,
  "cpus": 0.1,
  "instances": 2,
  "ports": [
      9998,0,0
  ],
  "uris": [
    "http://172.20.13.224/repo/apache-tomcat-7.0.68.tar.gz",
    "http://172.20.13.224/repo/jre-7u76-linux-x64.tar.gz",
    "http://172.20.13.224/repo/new-app.war"
  ]
```

wget

```
[root@01~]# wget app.war  tomcat.tar.gz  jdk.tar.gz
```

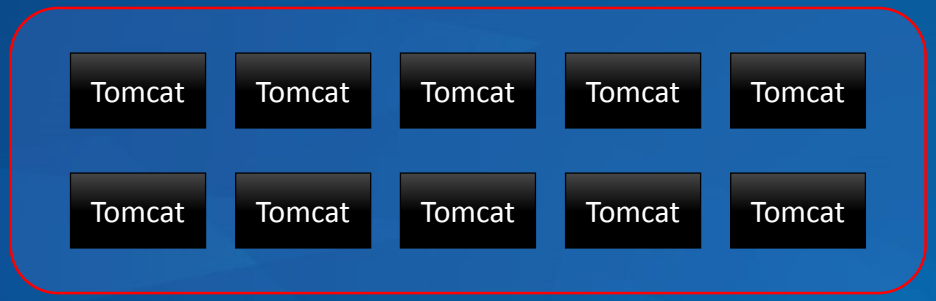
unzip

```
[root@01~]# tar –zvxf tomcat.tar.gz jdk.tar.gz
```

start

```
[root@01~]# ./bin/catalina.sh run -config ./conf/mesos.xml
[root@01~]# ./bin/catalina.sh run -config ./conf/mesos.xml
[root@01~]# ./bin/catalina.sh run -config ./conf/mesos.xml
```

由1台服务器，产生了10个新“中间件”（原生）



# Native模式应用发布的利与弊

简单，无需DockerFile

不用服务编排

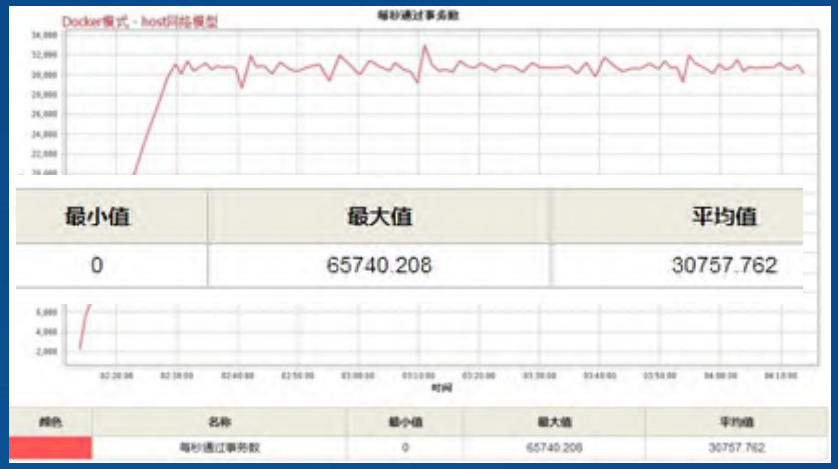
基于Framework性能优势

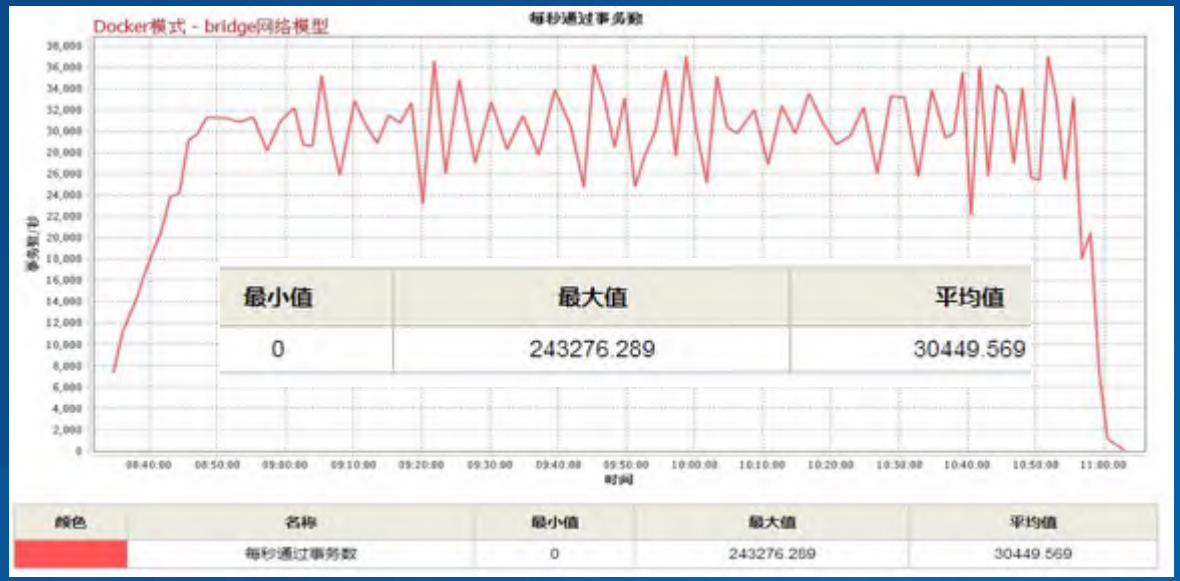
环境冲突

版本管理

端口管理

# 百万并发压测 – Docker不同模式的性能对比

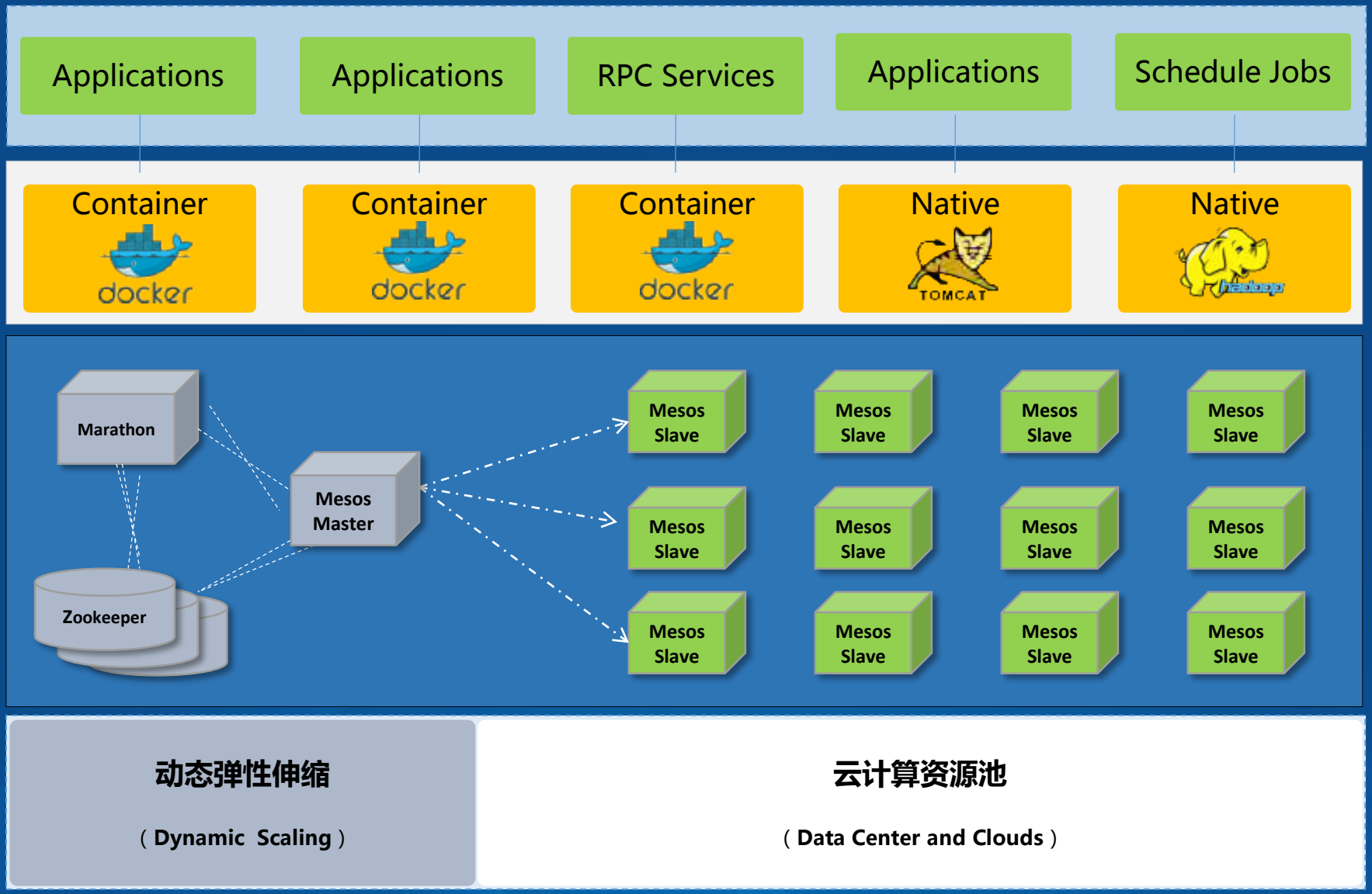




**测试场景：**物理机+Docker

**结论：**Docker bridge网络模型，性能最差，但是相比Docker host模型损耗不大，1%左右。

Docker host网络模型和Native方式，性能几乎无差异。

# 弹性高可用的设计


Applications
Applications
RPC Services
Applications
Schedule Jobs
Container
docker
Container
docker
Container
docker
Native
TOMCAT
Native
Marathon
Mesos Master
Zookeeper
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Slave
动态弹性伸缩
( Dynamic Scaling )
云计算资源池
( Data Center and Clouds )

# 健康状态检查保证高可用

```
"healthChecks" : [
 {
     "path":"$healthCheckPath",
     "portIndex": 0,
     "protocol": "HTTP",
     "gracePeriodSeconds": 300,
     "intervalSeconds": 60,
     "timeoutSeconds": 10,
     "maxConsecutiveFailures": 3,
     "ignoreHttplxx": false
   }
 ],
```

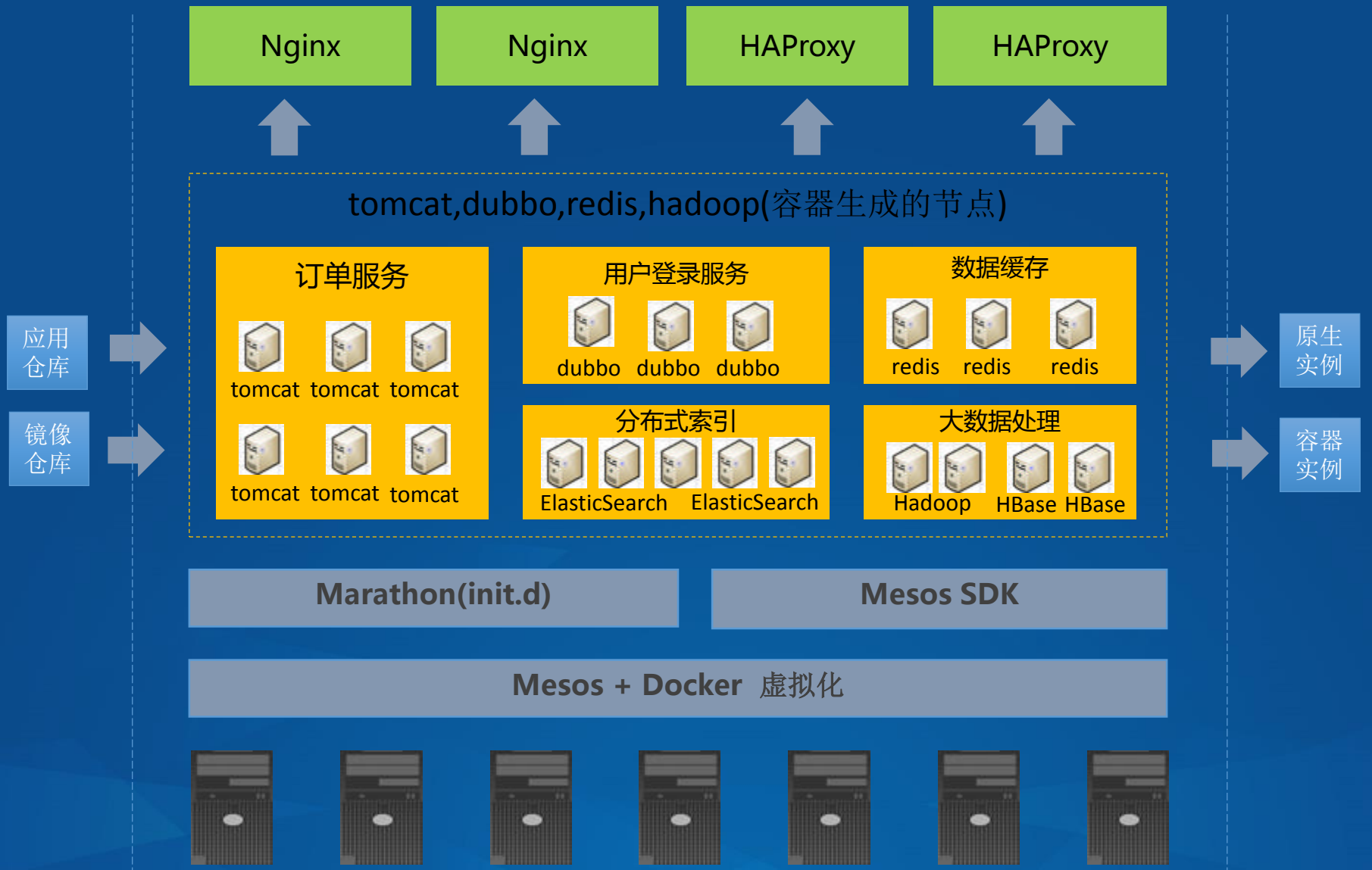
```
"healthChecks" : [
   {
     "protocol": "COMMAND",
     "command": { "value": "curl -f -X GET http://$HOST:$PORT0/health" },
     "gracePeriodSeconds": 60,
     "intervalSeconds": 60,
     "timeoutSeconds": 30,
     "maxConsecutiveFailures": 3
   }
 ],
```

```
"healthChecks" : [
   {
     "protocol": "TCP",
     "portIndex": 0,
     "gracePeriodSeconds": 60,
     "intervalSeconds": 60,
     "timeoutSeconds": 30,
     "maxConsecutiveFailures": 3
   }
 ],
```

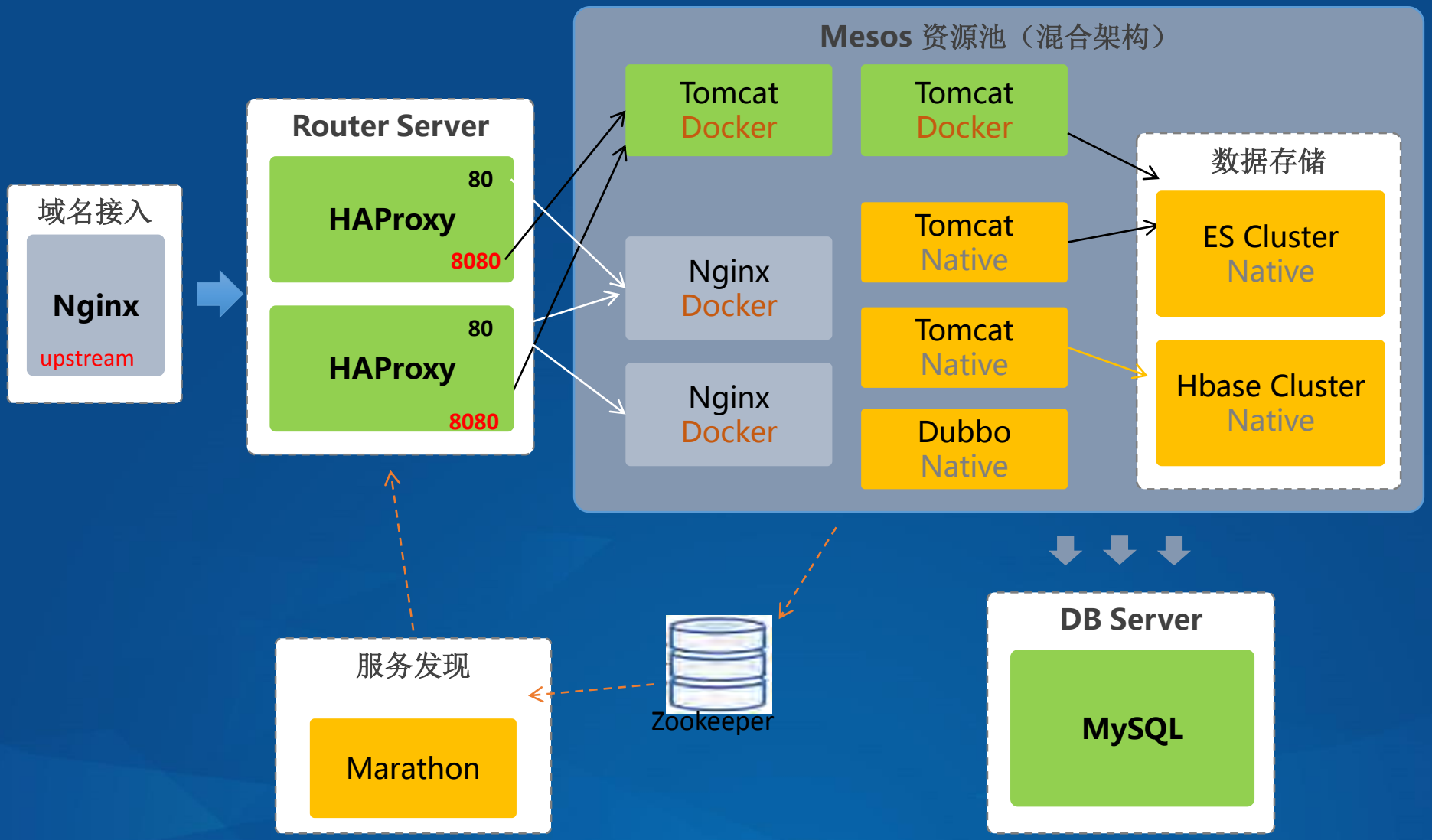
| | | | | |
|---|---|---|---|---|
| elasticsearch... | 1.2 | 2 GiB | | 6 of 6 |
| hbase-mesos | 0.1 | 256 MiB | | 1 of 1 |
| hdfs-mesos | 0.1 | 256 MiB | | 1 of 1 |
| zookeeper-2 | 2.4 | 2 GiB | Running | 3 of 3 |

| | | | | | | |
|---|---|---|---|---|---|---|
| elasticsearch-2.1.1_es-querynode-cluster.2137ea49-0176-11e6-b744-024265972527<br>172.20.13.224:[31868, 31869, 31870] | Healthy | Started | stderr | stdout | 7 days ago | 2016/4/13 下午 8:49:18 |
| elasticsearch-2.1.1_es-querynode-cluster.2136ffe7-0176-11e6-b744-024265972527<br>172.20.13.98:[31373, 31374, 31375] | Healthy | Started | stderr | stdout | 7 days ago | 2016/4/13 下午 8:49:21 |

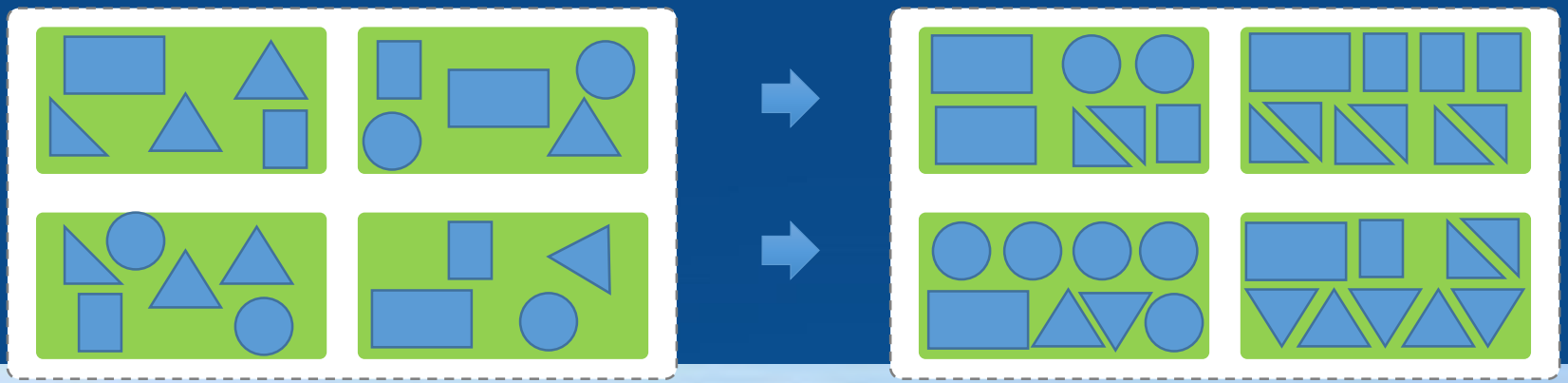
## 应用运行的架构


Nginx
Nginx
HAProxy
HAProxy
tomcat,dubbo,redis,hadoop(容器生成的节点)
订单服务
tomcat tomcat tomcat
tomcat tomcat tomcat
用户登录服务
dubbo dubbo dubbo
数据缓存
redis redis redis
分布式索引
ElasticSearch ElasticSearch
大数据处理
Hadoop HBase HBase
应用仓库
镜像仓库
原生实例
容器实例
Marathon(init.d)
Mesos SDK
Mesos + Docker 虚拟化

# 应用访问的数据流向

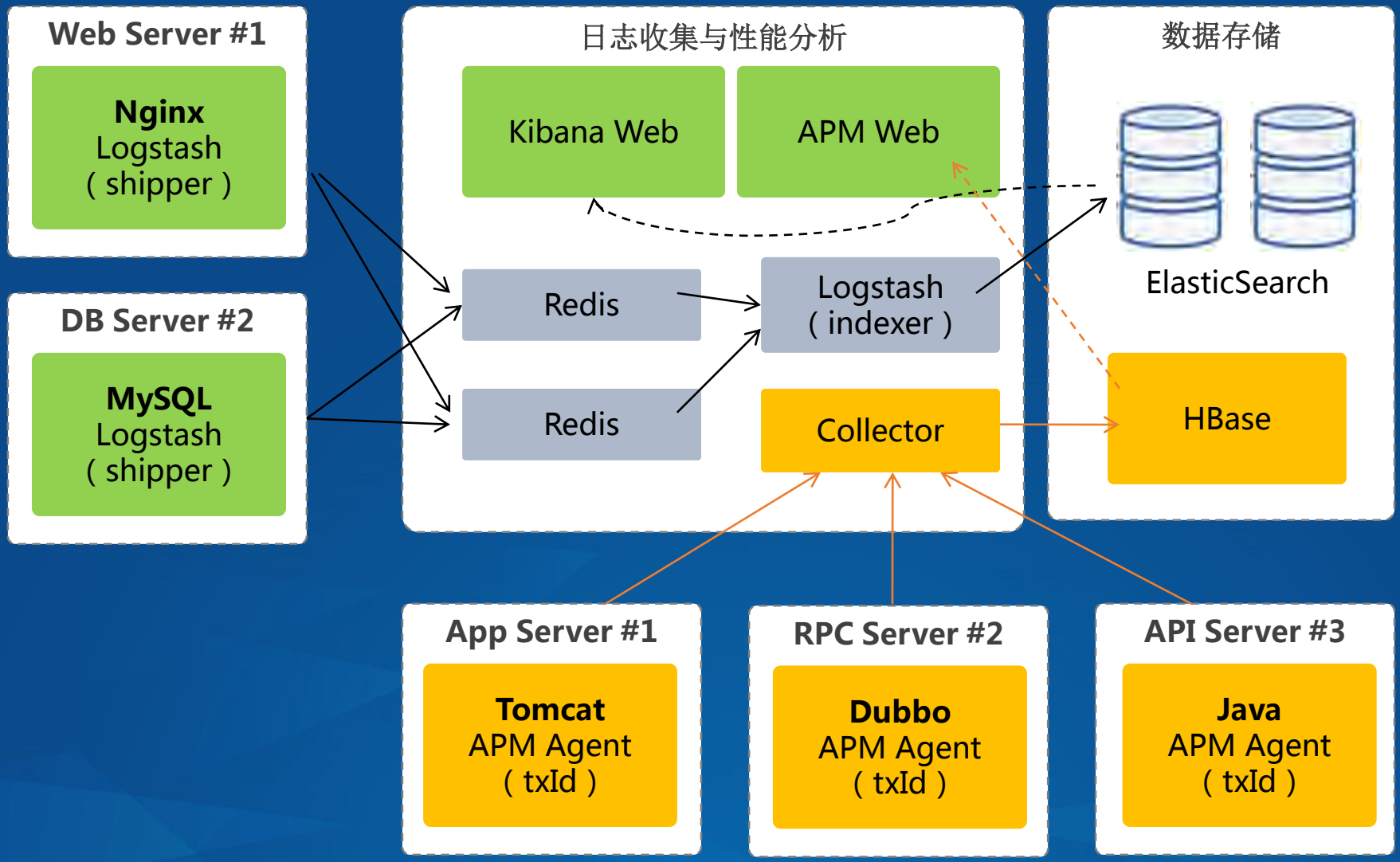

域名接入
Nginx
upstream
Router Server
80
HAProxy
8080
80
HAProxy
8080
Mesos 资源池（混合架构）
Tomcat
Docker
Tomcat
Docker
Nginx
Docker
Nginx
Docker
Tomcat
Native
Tomcat
Native
Dubbo
Native
数据存储
ES Cluster
Native
Hbase Cluster
Native
服务发现
Marathon
Zookeeper
DB Server
MySQL

## 资源的合理利用保证服务的高性能

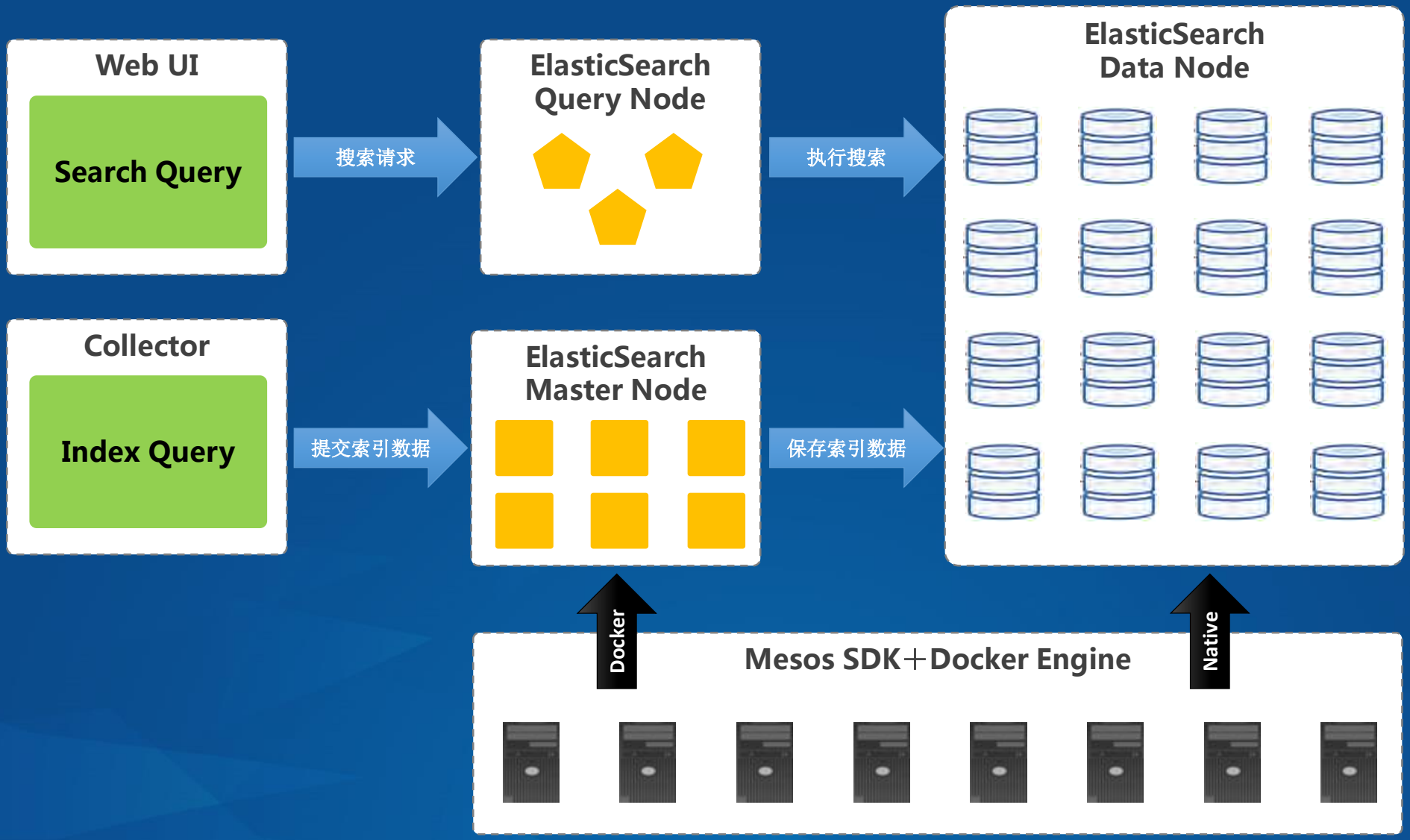


CPU
I/O
NetWork
Mem

# 业务监控和应用性能监控怎么做？



# 更大数据量的Easticsearch集群



# 应用云化的一些规范与准则

- 代码与拓扑无关
- 应用无状态化
- 不要写本地磁盘
- 日志统一管理
- DockerFile +镜像化

- 不要依赖基础设施
- 使用通用的REST API
- 禁止手工部署应用
- Port资源必须规划
- 禁用不稳定的源

# 平台建设过程中用到的技术栈

Jenkins
docker
MESOS
MARATHON
python
SALTSTACK
MySQL
CHRONOS
elasticsearch.
logstash
kibana
APACHE HBASE
Nagios
Apache Zookeeper
hadoop
Maven
Ganglia
HA PROXY
redis
NGINX

# 企业互联网开放平台的总体构成

## 行业化/领域化应用支撑平台

### 电商应用支撑平台

| | |
|---|---|
| 营销规则引擎 | 根据不同用户群、不同商品，配置不同折扣的价格规则 |
| 电商主数据 | 包括客户、商品、供应商、员工等 |
| 其他支撑 | 包括购物车、电商类UI模板等 |

其他领域应用支撑

## 公共应用支撑

### 互联网连接器

| | | |
|---|---|---|
| 社交网站适配 | 支付平台适配 | 电商平台适配 |
| IM适配 | 短信适配 | 消息推送适配 |
| ..... | | |

### 应用支撑组件

| | | | |
|---|---|---|---|
| 认证 | 调度任务 | 打印 | 编码规则 |
| 权限 | 业务日志 | 规则引擎 | ..... |

## 技术平台

开发工具

前端控件、框架

服务端中间件、框架

## 云运维平台

- 持续集成
- 镜像仓库
- 弹性伸缩
- 日志管理
- 性能分析
- 运维监控


用友iUAP

iuap.yonyou.com

The 8th China Cloud Computing Conference

# Thank you