容器技术在云平台大规模后台系统的应用实践

UCloud 刘晓晖

2015/10/17

自我介绍

刘晓晖

UCloud 资深研发工程师

2010年浙江大学研究生毕业后，加入百度从事自动化运维平台相关的研发工作。目前就职于UCloud，负责容器相关技术的开发和运维工作。对运维自动化平台、运维安全和PaaS相关技术有较丰富的研发和运维经验

1. UCloud云平台后台系统介绍

2. UCloud的容器技术应用实践

3. 容器化的实践经验总结

4. 后续计划

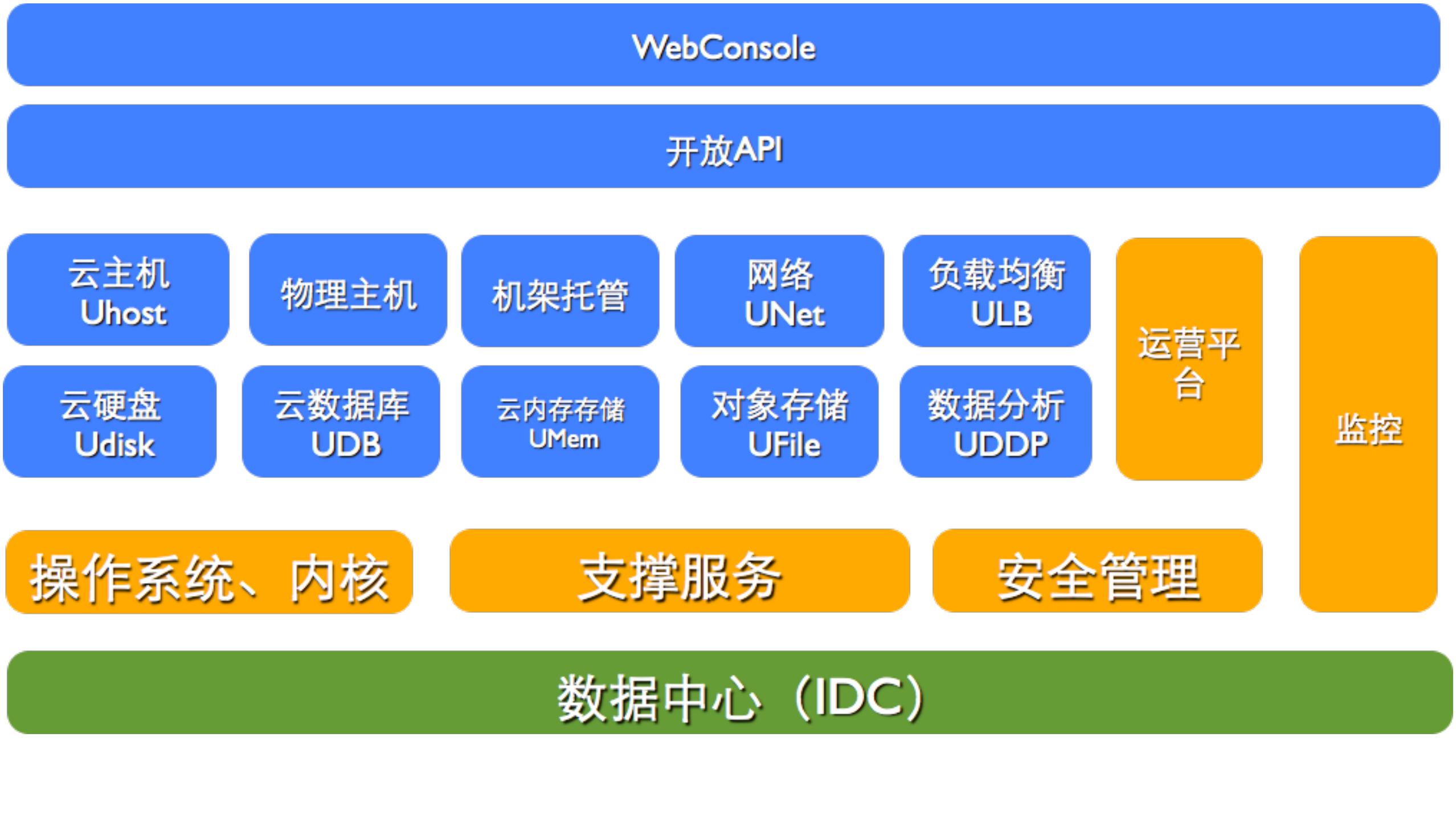
1. UCloud云平台后台系统介绍

2. UCloud的容器技术应用实践

3. 容器化的实践经验总结

4. 后续计划

云平台后台系统介绍


WebConsole
开放API
云主机 Uhost
物理主机
机架托管
网络 UNet
负载均衡 ULB
运营平台
监控
云硬盘 Udisk
云数据库 UDB
云内存存储 UMem
对象存储 UFile
数据分析 UDDP
操作系统、内核
支撑服务
安全管理
数据中心（IDC）

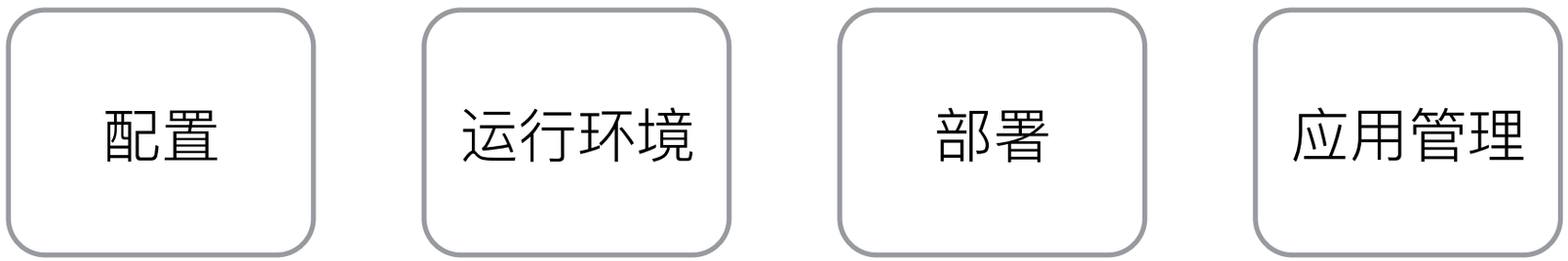
面临的问题和挑战

- 模块数量多，模块间的关联关系复杂，维护和部署成本高
- 多环境，多IDC部署，应用配置维护难
- 迭代速度快，部署交付效率低
- 服务器和应用规模增加迅速，应用管理和运维成本高

例如:监控系统

问题

- 统一内部系统和云主机监控，支持10w级设备节点的监控
- 20+模块部署在开发、测试、预发布环境，在10多个IDC进行部署，配置复杂
- 迭代速度快，每周交付2~3次，每次交付近2小时
- 新建一个IDC，模块完整部署、测试、运行需要近1周时间



1. UCloud云平台后台系统介绍

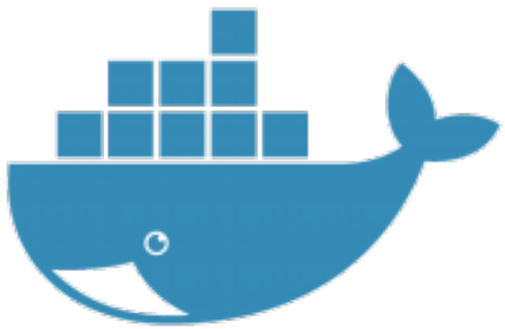
2. UCloud的容器技术应用实践

3. 容器化的实践经验总结

4. 后续计划

主要问题类型和解决思路

- 模块配置
  - 模块上下游关系，后端服务
  - 运行环境，机房的差异性配置等
- 一致性和依赖
  - 开发、测试、运行环境的不一致性
  - 依赖于不同的基础库



- 部署
  - 部署效率低下，步骤多，耗时长
  - 部署状态缺少检查机制
- 应用管理
  - 大量容器实例的管理、扩容、缩容成本高
  - 程序构建、打包、运行和运维统一管理
  - 监控、日志分析

容器-docker

Build

通过容器构建app，不管语言工具链

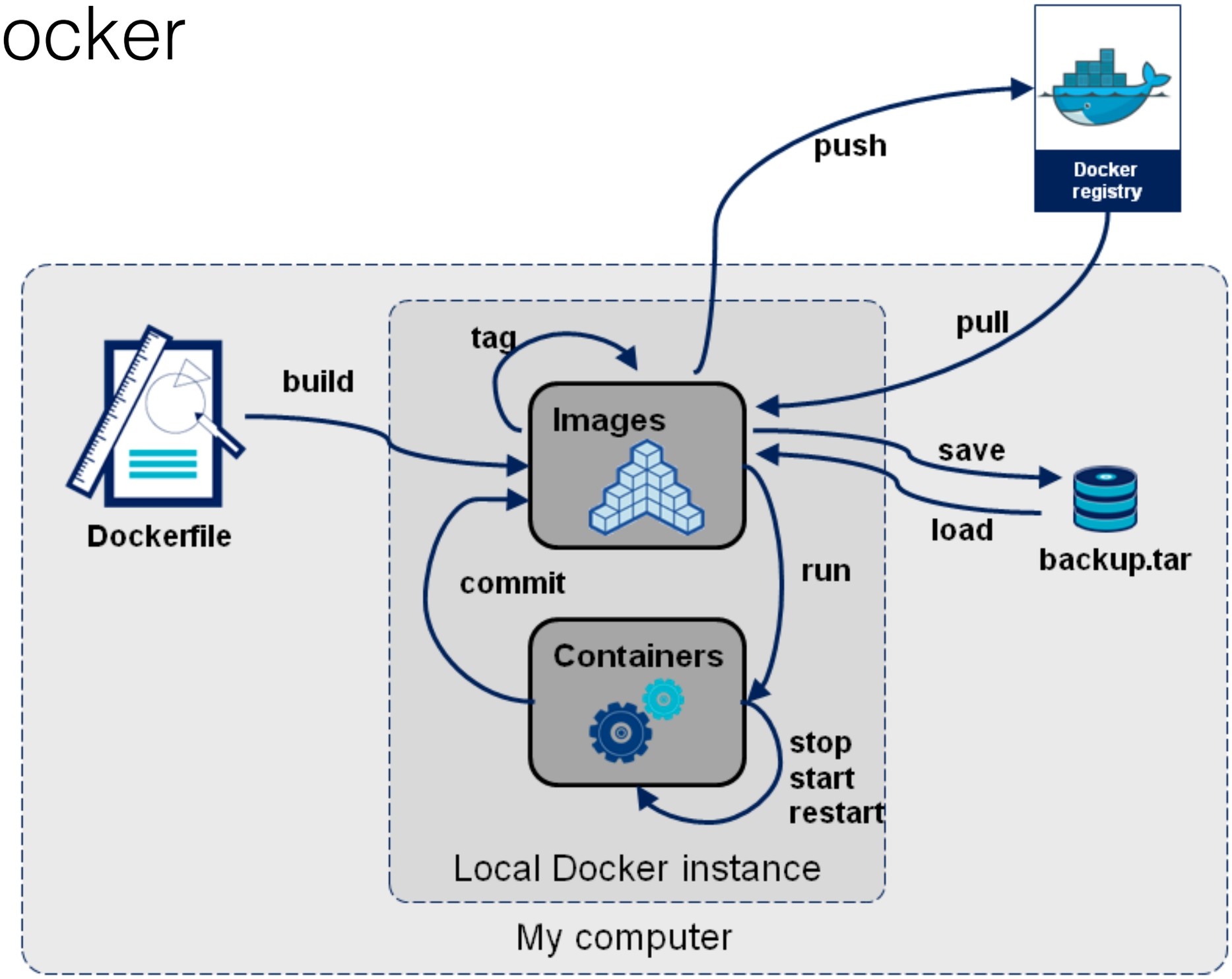
Ship

完整地ship容器化的app到任何地方-QA、云平台

Run

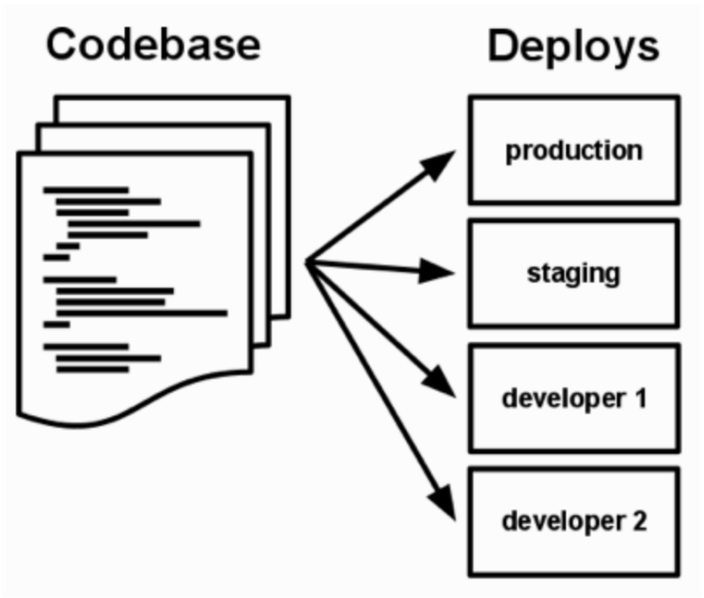
随时run容器化的app在各类云平台、虚拟机、个人PC、移动设备上

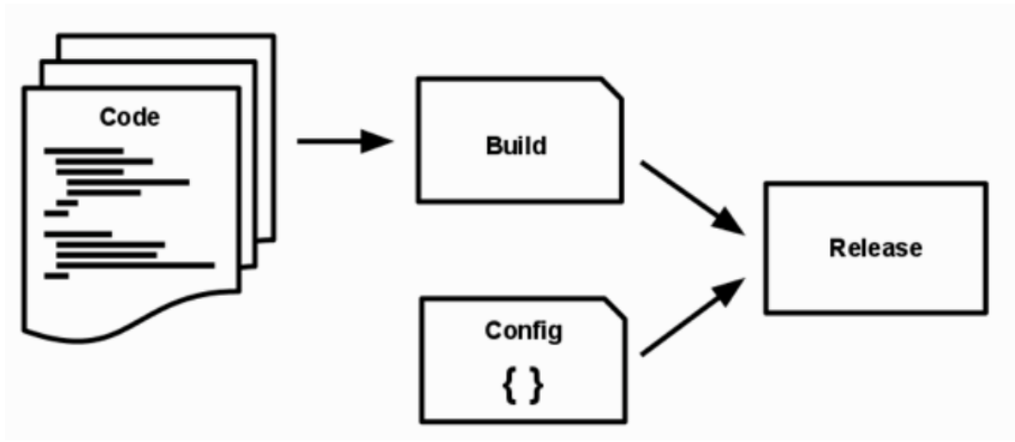
容器-docker


push
Docker registry
pull
tag
build
Images
save
load
backup.tar
Dockerfile
commit
run
Containers
stop
start
restart
Local Docker instance
My computer

模块配置

- 解决办法:
  1. 分离环境、IDC、资源类等差异化的配置项信息
  2. 配置模板，提交到codebase进行版本化管理
  3. 对不同deploys派生不同配置值，填充模板, 启动脚本
  4. 运行在不同的deploys中，只需通过环境变量传递给container即可





例子:

```
[root@19866b545665 ~]# ls
[redacted]  run.sh  templates  test.sh  umonitor2  [redacted]
```

run.sh

```
#################### Generating conf ####################
cd /root
file=$module".conf"
cp -f "templates/"$file".template" $fi
if [[ "$instanceid" != "0" ]]; then
        old=$file
        file=$module"_"$instanceid".cc
        mv -f $old $file
fi

sed -i "s#%{region}#$region#" $file
sed -i "s#%{regionid}#$regionid#" $fil
```

```
case "$
pre)

jh)
```

```
"Env": [
    "LOGLEVEL=information",
    "affinity:container==03431c1c
    "SERVICEPORT=6508",
    "NODESTARTID=200",
    "SERVICEIP=[redacted]"
],
"Cmd": [
    "/root/run.sh",
    "--region=jh",
    "--module=[redacted]"
],
```

docker镜像解决一致性和依赖问题

- 解决方法
  - 开发、测试、线上运行环境均采用docker生成的镜像，保证一致；
  - 基础系统、基本工具、框架，分层构建；
  - 基础镜像在开发、测试、线上环境统一预部署；
- 私有镜像库
  - V2版本
  - 支持UFile驱动
  - 定时pull最新镜像

后台系统容器应用实践-概述

docker应用业务:	监控、UDB、UNet、UDDP
docker容器数:	数千个实例
docker版本:	1.1.1、1.8.2
kernel version:	2.6.32-431(定制化)、4.1.10
发行版:	centos 6.x
kubernetes version:	1.0.6

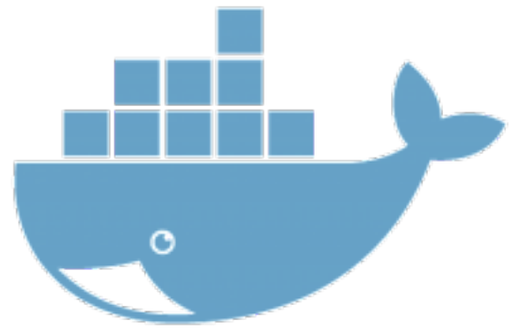
主要问题类型和解决思路

模块配置

模块上下游关系，后端服务
运行环境，机房的差异性配置等

一致性和依赖

开发、测试、运行环境的不一致性
依赖于不同的基础库



部署

- 部署效率低下，步骤多，耗时长
- 部署状态缺少检查机制

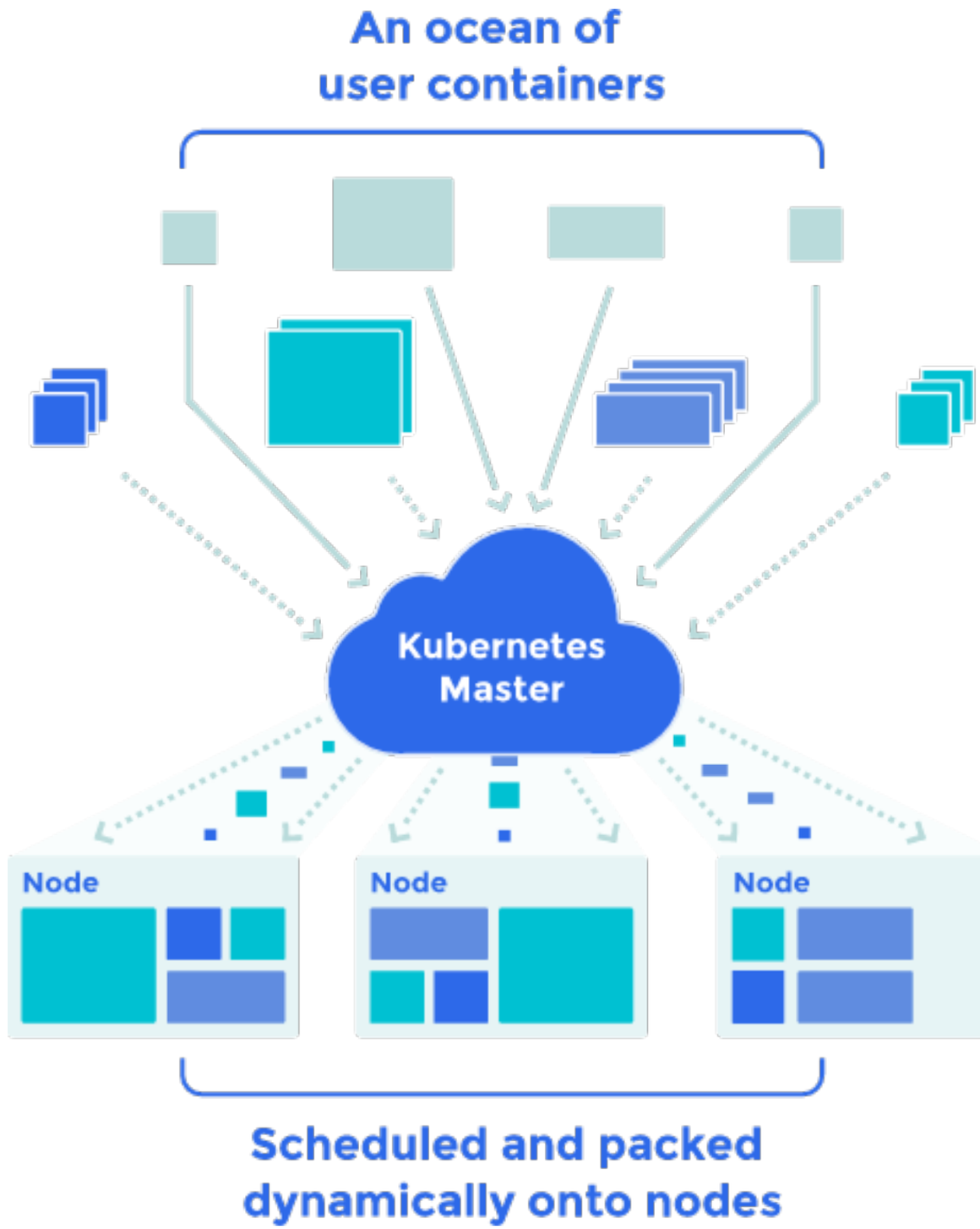
应用管理

- 大量容器实例的管理、扩容、缩容成本高
- 程序构建、打包、运行和运维统一管理
- 监控、日志分析

容器集群-kubernetes

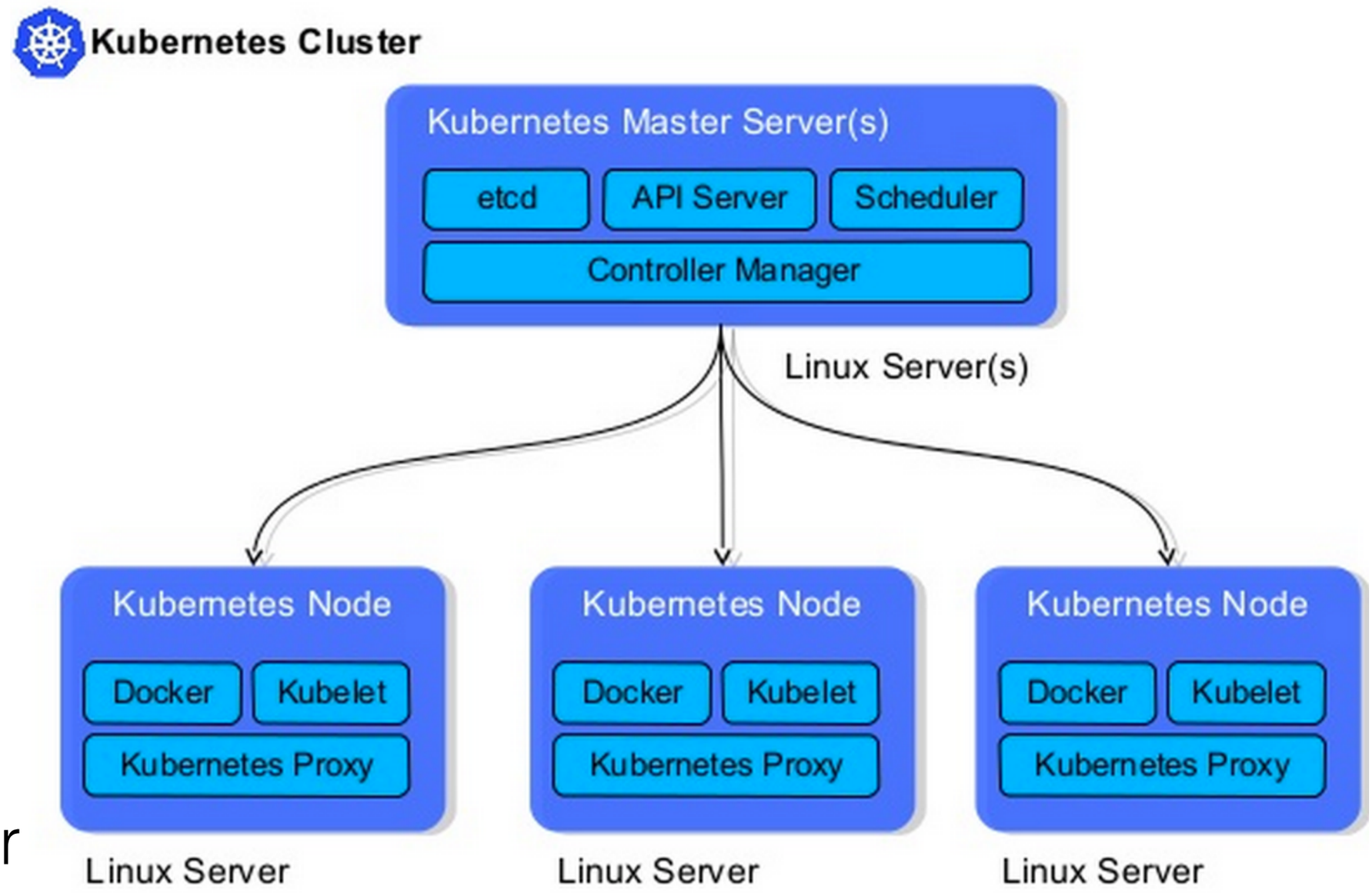
集群基本需求

- [x] 资源调度
- [x] 负载均衡
- [x] 健康检查
- [x] 实例伸缩
- [x] 服务发现
- [x] 生命周期
- [ ] 监控日志(插件)
- [ ] 程序构建
- [ ] 镜像管理
- [ ] 存储和网络



kubenetes

- 模块化
- 可扩展
- 轻量级



- container
- pod
- controller
- service
- labels & selector
- scheduler

容器集群管理收益明显

- **部署效率提升**
  - 部署时间减少：小时级别到分钟级
  - 部署效果能有效保证，运行副本数满足期望
- **容器生命周期管理**
  - 自动故障恢复，减少故障处理运维成本

1. UCloud云平台后台系统介绍

2. UCloud的容器技术应用实践

3. 容器化的实践经验

4. 后续计划

一些经验(1)

- **docker日志**
  - 日志打印耗费性能
  - 最好关闭logdriver，将日志打印在后台
- **docker daemon**
  - 退出kill container，升级docker daemon,kill可选
  - centos 6.3 service stop耗时长，需要5min，init-scripts的bug
- **docker 网络**
  - NAT模式下会启用启用nf_conntrack造成性能下降: 调节内核参数

一些经验(2)

- **合理设置ulimit**
- **docker镜像**
  - 制作镜像时候，commit的信息要简单明了
  - 编写dockerfile规范、减少镜像层数，基础部分放前面
  - 分地域部署镜像registry

1. UCloud云平台后台系统介绍

2. UCloud的容器技术应用实践

3. 容器化的实践经验

4. 后续计划

后续计划

- **容器集群化管理**
  - 提高集群的稳定性和可靠性，进一步推进容器化和微服务化改造
  - 推广kubernetes集群的使用，完善集群功能
  - 推广从研发、测试、运维全流程化，持续发布
- **容器相关技术产品化**

THANKS

xiaohui.zju@gmail.com