学霸君的微服务探索

——基于阿里云容器的架构设计

俞杰
学霸君 运维负责人

目 录
content

扫码观看大会视频

个人简介

- 工作经历
  - 就职公司：蓝汛通信
    - 人民澳客
    - 学霸君
  - 工作内容：Linux系统优化
    - 运维自动化
    - 微服务

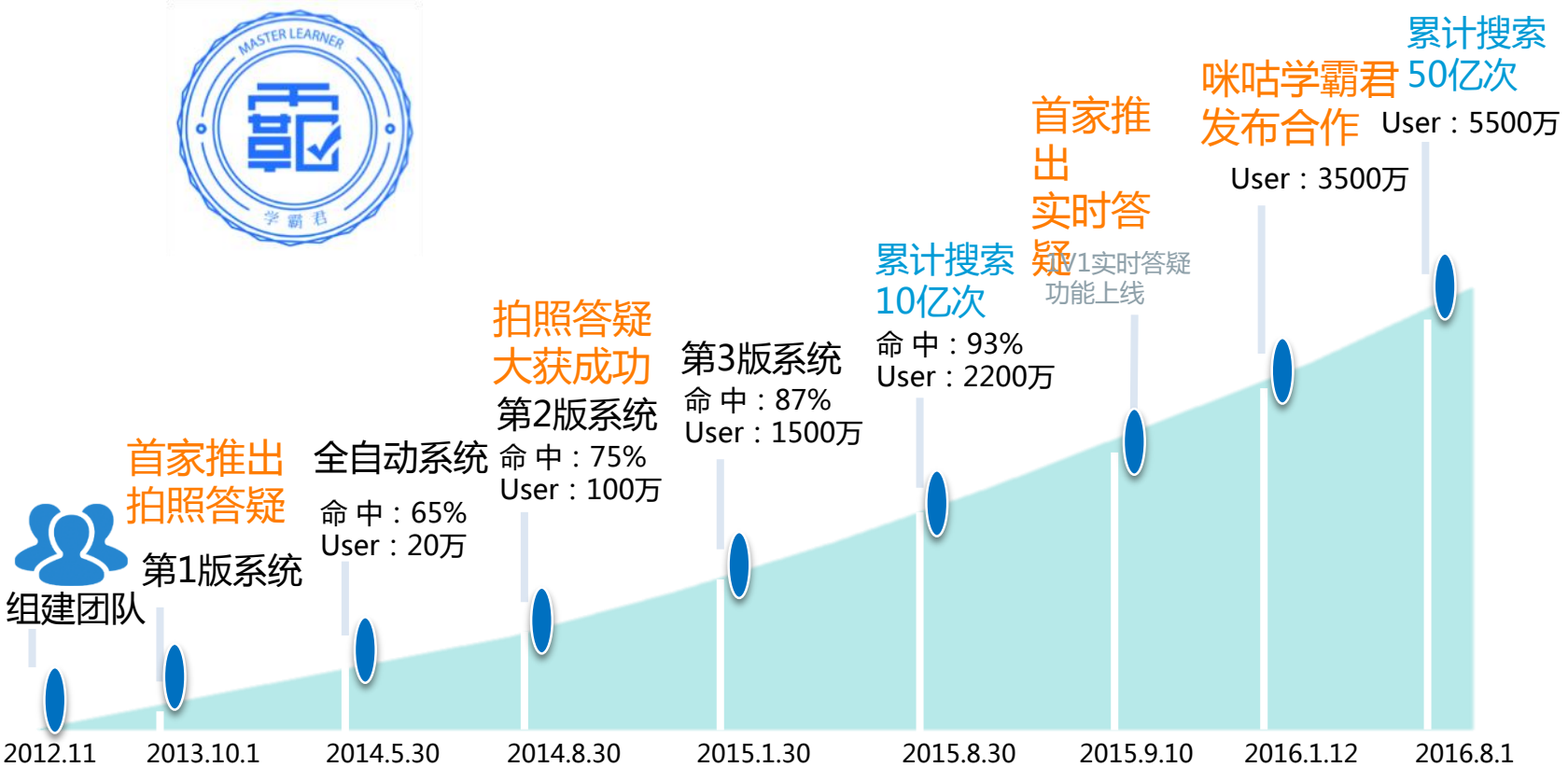
扫码观看大会视频

一、学霸君介绍

问吧科技创立于2012年10月，
是国内领先的在线教育公司，
也是一支典型以技术为驱动的创业公司，
旗下拥有国内首款针对中小学生在线答疑
为核心的产品—“学霸君”APP。

5,500万+	注册用户
50亿+	累计用户搜题
1,800万+	每日用户搜题数
350万+	日活用户

扫码观看大会视频

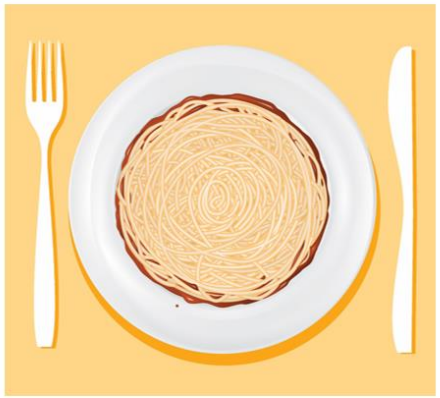

组建团队
第1版系统
首家推出
拍照答疑
全自动系统
命 中：65%
User：20万
拍照答疑
大获成功
第2版系统
命 中：75%
User：100万
第3版系统
命 中：87%
User：1500万
累计搜索
10亿次
命 中：93%
User：2200万
首家推出
实时答疑
1V1实时答疑
功能上线
咪咕学霸君
发布合作
User：3500万
累计搜索
50亿次
User：5500万
2012.11
2013.10.1
2014.5.30
2014.8.30
2015.1.30
2015.8.30
2015.9.10
2016.1.12
2016.8.1

扫码观看大会视频

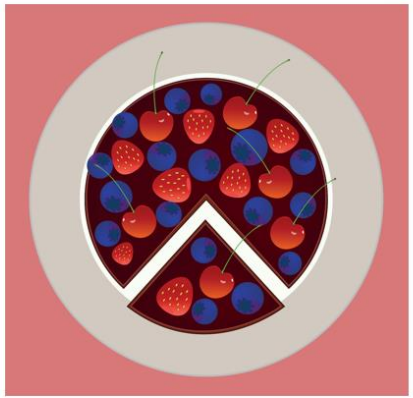
适用人群

- 对微服务架构向往，希望有所实践的公司
- 对微服务和Docker有一些了解，希望获得实际生产实践经验的同行
- 希望借助第三方服务，最大化优化运维效率的中小企业

二、被宣传的微服务



With monolithic, tightly coupled applications, all changes must be pushed at once, making continuous deployment impossible.



Traditional SOA allows you to make changes to individual pieces. But each piece must be carefully altered to fit into the overall design.

With a microservices architecture, developers create, maintain and improve new services independently, linking info through a shared data API.

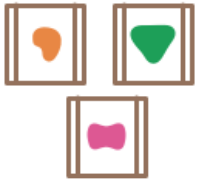
图片来源：http://www.kanbansolutions.com/blog/microservices-architecture-friend-or-foe/

Kanban Solutions @kanbansolutions kanbansolutions.com

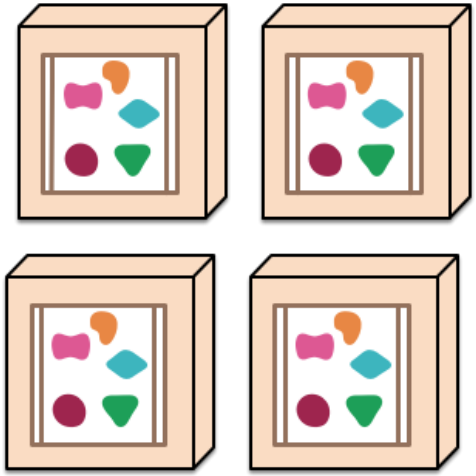
扫码观看大会视频

A monolithic application puts all its functionality into a single process...

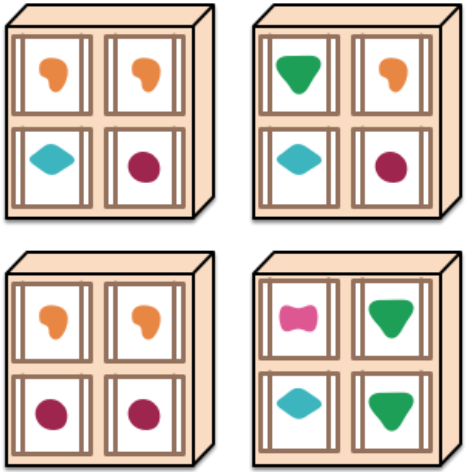
A microservices architecture puts each element of functionality into a separate service...



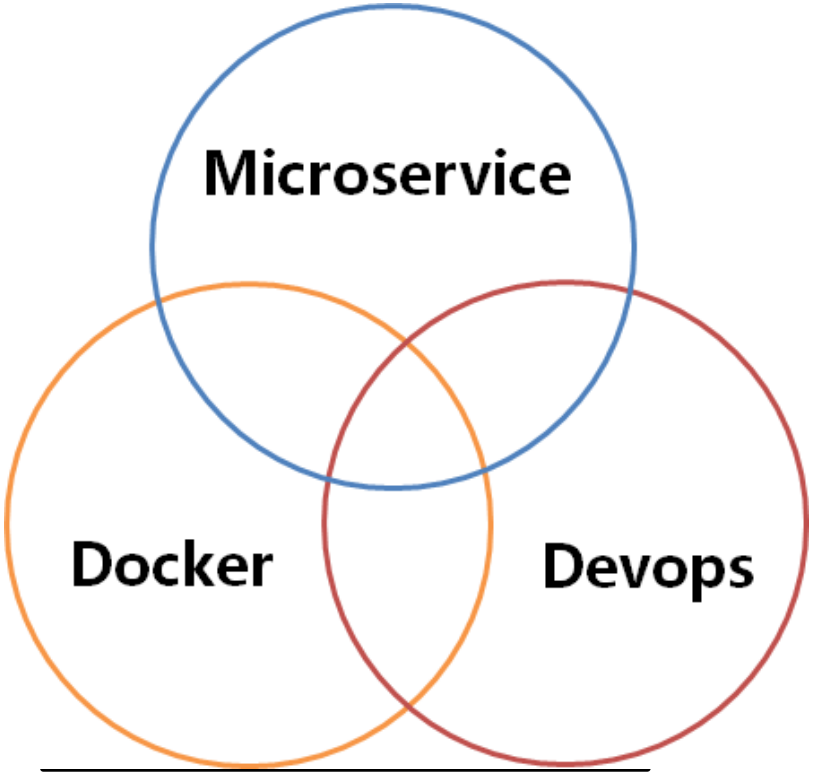
... and scales by replicating the monolith on multiple servers



... and scales by distributing these services across servers, replicating as needed.



图片来源：http://martinfowler.com/articles/microservices.html



图片来源：https://yq.aliyun.com/articles/2764

微服务：是一种架构思想，目的是有效的拆分应用，实现敏捷开发和部署；

Docker：为微服务化架构的实现提供了技术基础；

Devops：是一种重视软件开发、运维和质量保证之间沟通合作的文化、运动或惯例；

三、运维眼中的微服务

如何贯彻Devops流程

减小不同环境之间的差异，流水线式的一体化上线

如何贯彻Devops流程

减小不同环境之间的差异，流水线式的一体化上线

如何实现持续集成/交付

大幅缩短服务上线的时间，并减少人为的干预

如何贯彻Devops流程

减小不同环境之间的差异，流水线式的一体化上线

如何实现持续集成/交付

大幅缩短服务上线的时间，并减少人为的干预

如何快速定位并解决故障

服务链路清晰可感知，服务间调用关系可管理

如何实现细粒度监控

包括链路监控、服务监控等，真切反应业务运行状态

如何实现细粒度监控

包括链路监控、服务监控等，真切反应业务运行状态

如何实现高可用系统架构

服务分布式部署，并可快速的进行灾备切换

如何实现细粒度监控

包括链路监控、服务监控等，真切反应业务运行状态

如何实现高可用系统架构

服务分布式部署，并可快速的进行灾备切换

如何有效提高基础设施资源利用率

及时掌握集群的容量状态，并可以实现自动扩缩容

扫码观看大会视频

四、如何落地微服务

项目背景

如何在缺人缺时间的情况下，保证新业务及时上线

•缺人：团队一共6人，支撑了全公司多条产品线的运维支持，
故新业务最多可以抽出2人支持

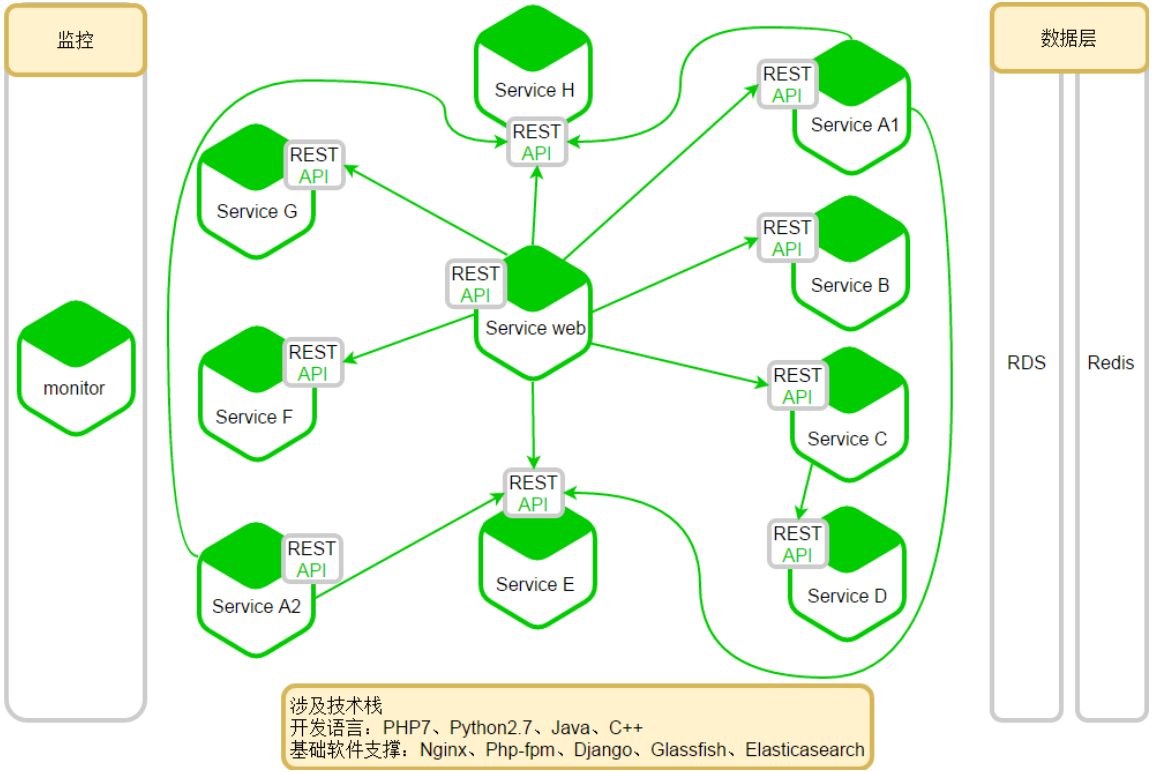
•缺时间：业务需要在1个半月后及时上线

为什么在如此紧迫的情况下尝试新的架构

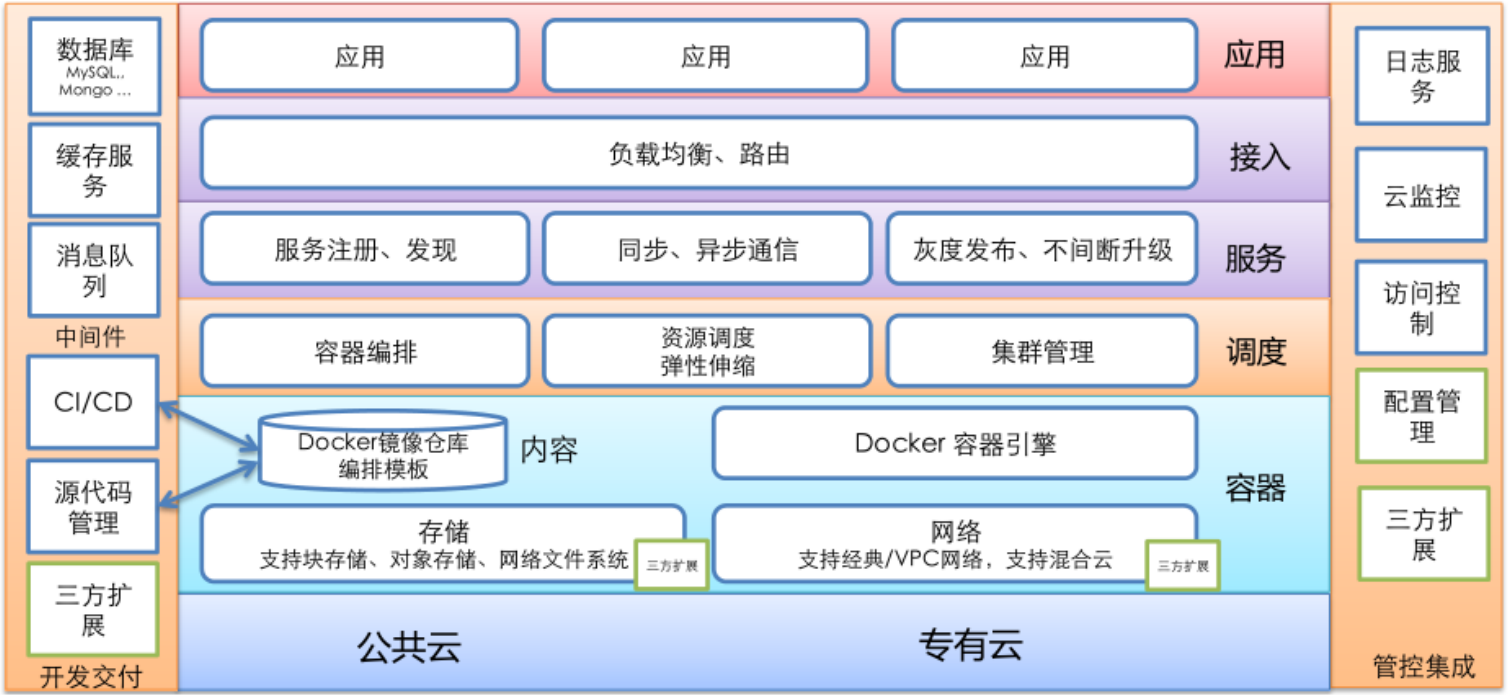
•借助微服务架构和Docker技术，可以很好的实践Devops理念

•新业务对于新的架构接受度较高，可以容忍一定程度的踩坑

项目架构



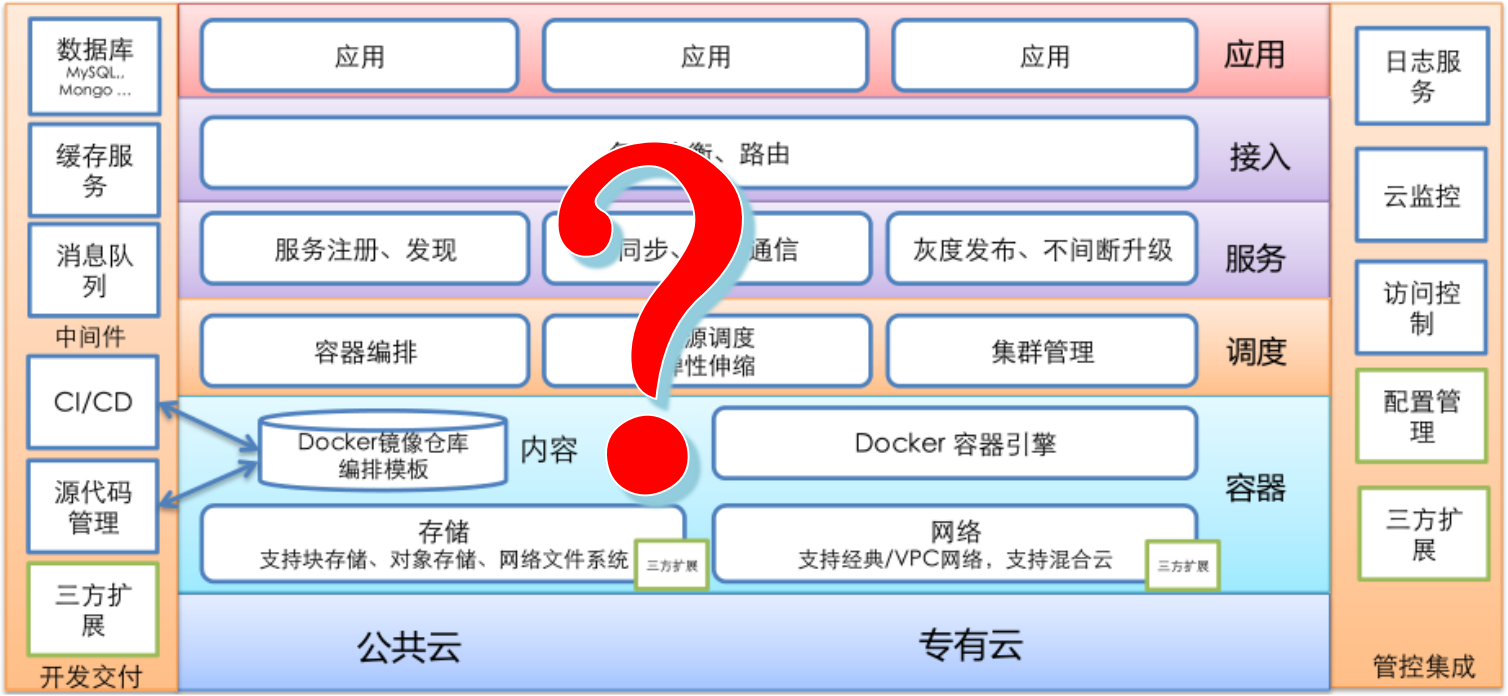
基于Docker的微服务架构技术栈



图片来源：http://chuansong.me/n/2789820

扫码观看大会视频

基于Docker的微服务架构技术栈



图片来源：http://chuansong.me/n/2789820

扫码观看大会视频

微服务架构技术选型

考察点	具体内容
存储	支持哪些存储方式，适用场景具体有哪些？
	存储系统的访问是否是友好型的，如提供域名或服务名来访问，而不是硬编码的IP地址？
网络	支持哪些网络形态，是否支持混合云？
	网络设计的架构结构如何，如何实现对外服务层、对内服务层、数据层相对独立？
	如何保证服务之间的安全性和访问控制？
镜像仓库	镜像仓库是否有足够的有安全保证的镜像使用？
	镜像仓库如何实现跨不同的环境（开发、测试、预发布、生产）进行访问，或无需如此？
	镜像管理的最佳实践是如何的，如镜像版本化管理、镜像精简化等？
资源调度、弹性伸缩	ECS资源、容器资源等的自动扩缩容是否需要人为参与，或是否可以人为指定资源的容量比例（如80%）？
	容器的高可用和故障自动迁移的触发机制，和容器切换的时间？
	容器是否可以灵活指定扩缩容的策略，如根据业务压力动态调整容器实例数量？
集群管理	如何管理上万计的容器，是否具有服务调用管理的拓扑图示？
	容器创建、迁移、销毁、扩缩容等，是否有具体的数据参考，用以评估集群的健康状态？
	集群管理的最佳实践是什么？
服务注册、发现	如何配置服务A的路由规则，并由程序发现？
	修改路由规则后，如何触发程序进行热更新？
	适用场景有哪些？不适用的情况有哪些？
	容器启动后，提供的服务如何被其它服务模块访问到？是否自动注册吗？

微服务架构技术选型

考察点	具体内容
同步、异步通信	是否提供同步、异步的通信机制或产品？用以规范服务模块间的调用关系
	同步、异步通信支持哪些方式和协议，最佳实践是如何的？
灰度发布、不间断升级	是否需要消除任何服务模块的单点。即每个服务模块都至少有2个容器实例在运行，同时规划一套灰度发布组，用以新功能的生产环境小规模测试？
	灰度发布组如何规划，尤其是如何监控相关指标，得到与未升级组相关业务指标的对比测试？
	是否需要将所有服务都设计为无状态服务？（数据都独立进专用的存储中）
负载均衡、路由	内部间的服务模块访问，如何实现以“服务名”访问，而非硬编码IP地址？
	是否一切需对外提供服务的服务都必须对接SLB，便于统一管理和后期扩展？
	是否外部服务根据业务形态需要划分不同的域名方式？
日志收集	日志要以数据流的形式输出，并汇总至统一的日志中心，如何实现？
	日志如何分析、存储、展示，形式是否满足需求？
	配置日志输出方式，对于程序的改造难度如何？
CI/CD	在不同环境（开发、测试、预发布、生产）之间，流通的是代码，还是镜像？
	如何实现自动化单元测试、自动化功能测试、自动化压力测试等？
	源代码如何管理，遵循github分支还是gitlab分支？
	经过CI/CD流程后，可部署至生产环境的输出是原始代码、还是镜像？
	单元测试是否必要，如何在程序设计层次进行接入？

微服务架构技术选型

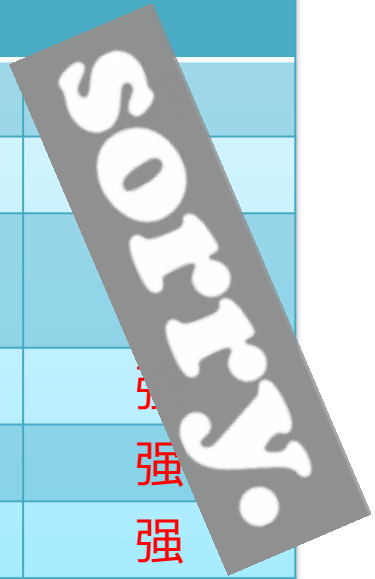
考察点	具体内容
监控	基础资源监控，如ECS资源使用率（CPU、Memory、IO等）、容器资源使用率（CPU、Memory、IO等）能否对我们做到透明？
	如何实施业务和服务模块的监控标准化？判断某个服务或业务是否健康的指标主要参考哪些，最佳实践是什么？
	如果想要实现业务指标采集作为代码的一部分，同时提高指标覆盖率，需要开发如何接入实现？
	业务监控数据的分析、展示的途径有哪些？能否自由集成监控告警和集成趋势图等？（如业务指标趋势图等）
配置管理	配置如何与代码隔离？如何实现集中化管理？
	常见配置有： * 各种开关，比如降级的开关、debug开关、ab测试的开关等
	* 各种可配参数，比如超时时间、并发数、日志级别等
	* 上下游连接信息（与环境的耦合度最高，是最复杂、最多变、最难处理的部分）
部署和变更	部署和变更的版本化管理，以版本为发布单位，版本号以 V1.0_201606131608 是否合适？
	统一的接入流程（gitlab的release分支）和打包规范（构建为docker image，命名为 oneone-ctl_V1.0_201606131608）？
	预发布环境是否必要，主要作用是什么？如何实施？
	如何规划统一的上线流程和检查机制（小流量测试、容灾测试、集成监控告警、集成趋势图）？
	能否实现代码上线由服务模块开发者操作，运维部只要起到核查测试报告并审核操作的角色？
开发习惯	开发环境是只需要一个IDE，还是需要一套线下的类生产环境？
	如何将可维护性带入开发流程中，从开始就追求服务模块的可维护性？

阿里云容器的优势

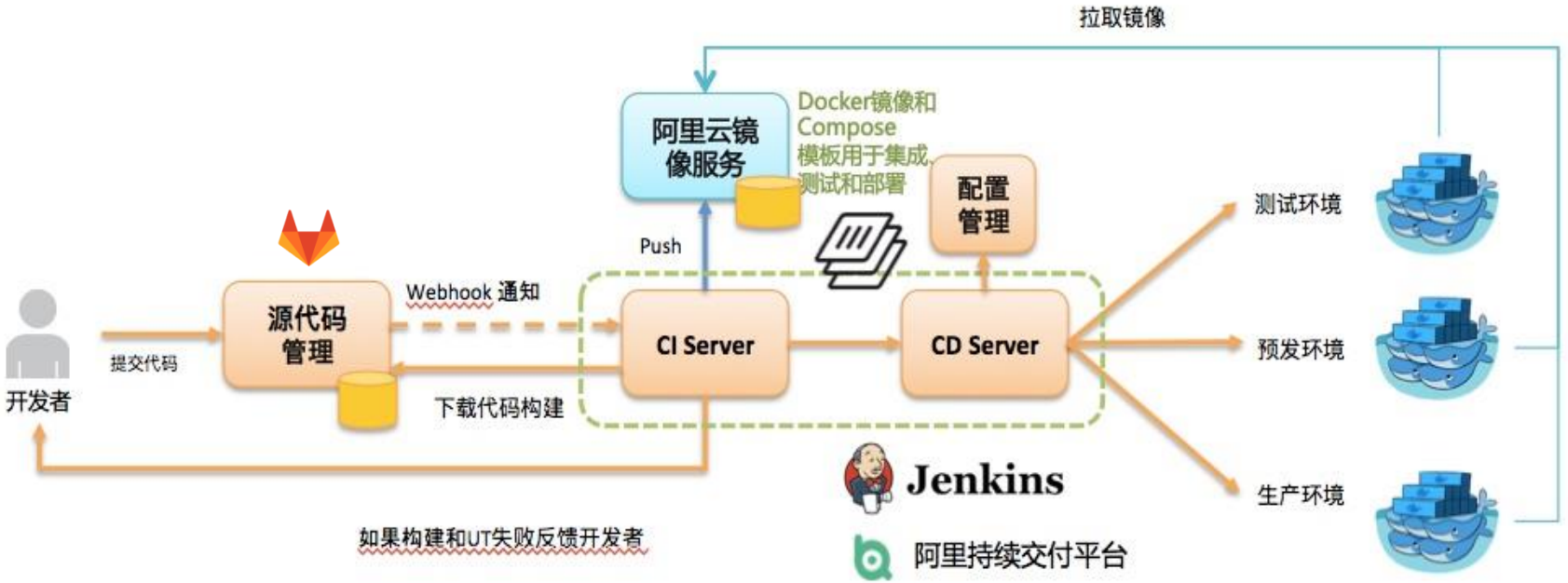
方案对比	阿里云容器	第三方Pass平台	AWS
基础设施支持能力	强	中	强
容器技术成熟度、灵活度	强	弱	强
微服务和devops技术支持	强	中	强
对于第三方系统的支持	强	强	强
蓝绿部署/金丝雀部署	强	弱	强
自动扩缩容	强	强	强
自动识别开发语言并编译	弱	强	弱

阿里云容器的优势

方案对比	阿里云容器	第三方Pass平台	AWS
基础设施支持能力	强	中	
容器技术成熟度、灵活度	强	弱	
微服务和devops技术支持	强	中	
对于第三方系统的支持	强	强	强
蓝绿部署/金丝雀部署	强	弱	强
自动扩缩容	强	强	强
自动识别开发语言并编译	弱	强	弱



持续CI/CD

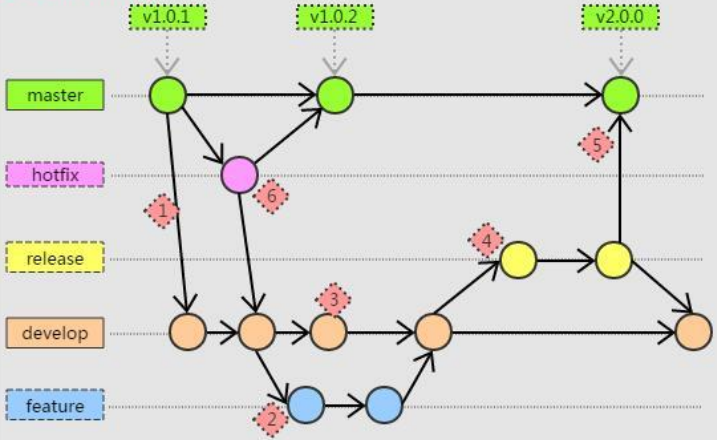


图片来源：https://help.aliyun.com/document_detail/42988.html

扫码观看大会视频

持续CI/CD-Gitlab分支结构及自动化测试

多人开发代码分支流程



1	代码库创建时，master分支和develop需要默认创建	项目负责人负责创建	
2	功能开发需要在feature分支下进行，并最终合并（merge request）至develop分支	模块负责人负责feature分支开发，项目负责人负责代码合并	测试：代码静态检查 gitlab-CI
3	功能开发合并至develop分支后，会自动触发部署至测试（test）环境	项目负责人负责feature分支合并进来的代码质量	测试：单元测试 + 接口测试 + 集成测试
4	经过测试的代码，等待上线，独立出release分支，会自动触发部署至预发布（prerelease）环境	项目负责人负责开辟release分支	测试：组件测试 + 测试
5	通过4的预发布环境的测试（功能测试+压力测试）并符合预期后，合并（merge request）至master分支并打tag，会自动触发部署至生产（production）环境（通知运维部进行人工确认），同时将代码合并（merge request）至develop分支	项目负责人负责将release分支合并至master分支和develop分支，并在mater分支上打tag	
6	hotfix分支仅用于重大的bug紧急修复，一般bug修复还是遵循feature流程	模块负责人负责hotfix分支开发，项目负责人负责代码合并	
!	**对于代码分支不规范，分支名混乱的上线需求，或缺少服务说明文件的程序，运维部有权不予上线！！！**		

扫码观看大会视频

持续CI/CD-服务编排

```
aixue-recommender:
    image: 'registry.cn-hangzhou.aliyuncs.com/[redacted]/aixue-recommender:v1.0.9'
    restart: always
    labels:
        aliyun.scale: 2
        aliyun.routing.port_8085: aixue-recommender.local;aixue-recommender
        aliyun.probe.url: 'tcp://container:8085'
        aliyun.probe.timeout_seconds: '10'
        aliyun.probe.initial_delay_seconds: '3'
        aliyun.rolling_updates: 'true'
        aliyun.log_store_aixue-recommender: stdout
    environment:
        - 'availability:az==~2'
        - DBHOST=[redacted]
        - USERNAME=[redacted]
        - PASSWD=[redacted]
        - DB=[redacted]
    external_links:
        - aixue-superxue-querybyid.local
```

路由

监控检测

日志输出

数据层连接

其他服务连接

扫码观看大会视频

配置

业务级配置：即不会因环境不同而不同的，业务相关，
通常可以以常量形式存在；
需要入版本库，和代码一同发布，或者说本质就是代码；

运维级配置：依据环境会变化，最常见的就是
ip/port/username/password 等；
不入版本库
通过在程序中引用环境变量的key值
通过在compose编排模板中创建对应的环境变量引入value

两种配置分离能简化运维成本，避免污染repo

配置

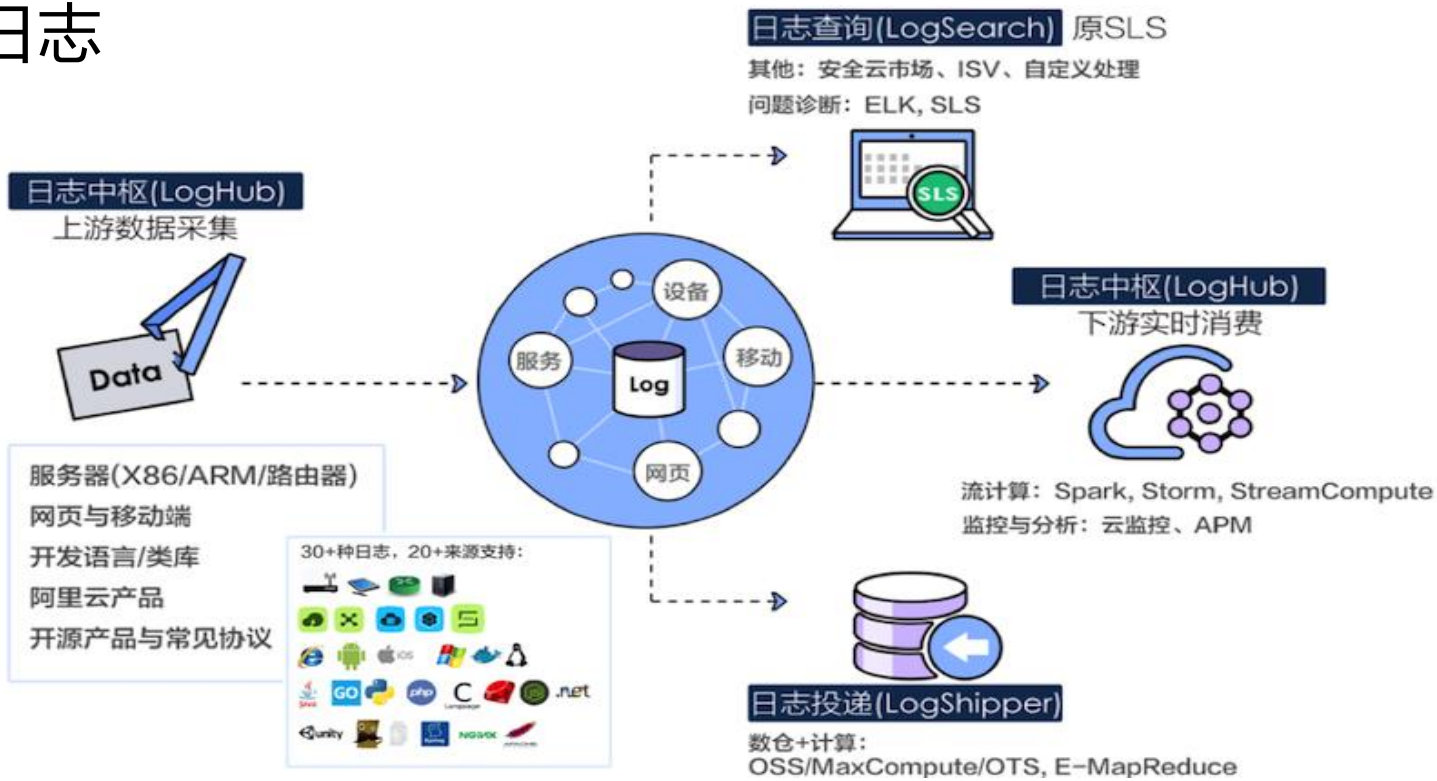
业务级配置：即不会因环境不同而不同的，业务相关，通常可以以常量形式存在；
需要入版本库，和代码一同发布，或者说本质就是代码；

```
//项目配置文件
return array(
    // '配置项' => '配置值'
    'db_type'        => 'mysql'
    'db_host'        => $_SERVER['MYSQL_HOST']
    'db_user'        => $_SERVER['MYSQL_USER']
    'db_pwd'         => $_SERVER['MYSQL_PASSWORD']
    'db_port'        => $_SERVER['MYSQL_PORT']
);
```

ame/password 等；

ue

日志



图片来源：https://help.aliyun.com/document_detail/28960.html

扫码观看大会视频

日志

日志类型：

debug日志：一般是程序的运行日志和debug输出日志，
仅提供日志内容原样展示（单行或多行日志都可）；

性能分析日志：一般是服务模块的性能数据，可进行频次统计、
关键字分析、概率分布分析等多种分析展示方式；

图片来源：http://blog.xebia.fr/2013/12/12/logstash-elasticsearch-kibana-s01e02-analyse-orientee-business-de-vos-logs-applicatifs/

五、微服务是否是救世良药

分布性系统天生的复杂性

如网络延迟、容错性、不可靠的网络、异步机制等

分布性系统天生的复杂性

如网络延迟、容错性、不可靠的网络、异步机制等

运维开销及成本增加

一个单体应用被拆分部署成十几个服务

分布性系统天生的复杂性

如网络延迟、容错性、不可靠的网络、异步机制等

运维开销及成本增加

一个单体应用被拆分部署成十几个服务

隐式接口及接口匹配问题

不同组件间协作需要统一可管理的接口

代码重复

为了尽量实现松耦合，相同功能的底层功能会在不同服务内多次实现

代码重复

为了尽量实现松耦合，相同功能的底层功能会在不同服务内多次实现

异步机制

微服务往往使用异步编程、消息与并行机制

代码重复

为了尽量实现松耦合，相同功能的底层功能会在不同服务内多次实现

异步机制

微服务往往使用异步编程、消息与并行机制

其它

如不可变基础设施、管理难度、微服务架构的推进等

2016 The Computing Conference

THANKS