



Apollo架构和实践

讲师：杨波

研发总监/资深架构师

课程大纲

- 01 课程概述
- 02 业务需求
- 03 配置定义和场景
- 04 开关驱动开发原理和实践
- 05 Apollo简介
- 06 Apollo核心概念
- 07 Apollo快速起步(Lab1)
- 08 Apollo架构设计之服务器端

课程大纲

- 09 Apollo架构设计之客户端
- 10 Apollo架构设计之高可用和监控
- 11 Apollo分布式部署指南
- 12 Apollo Java客户端配置和多语言接入
- 13 Apollo Client API实操 (Lab2)
- 14 Apollo Client和Spring集成~XML方式 (Lab3)
- 15 Apollo Client和Spring集成~代码方式 (Lab4)
- 16 Apollo Client和Spring Boot集成 (Lab5)

课程大纲

- 17 Apollo开放平台接入实操 (Lab6)
- 18 Spring Cloud Config简介
- 19 Apollo vs Spring Cloud Config
- 20 Apollo FAQ和开发常见问题
- 21 参考资料和后续课程预览

第

1

部分

课程概述

课程概述和亮点



1

杨波的微服务基础架构体系的**第二个**模块

2

携程开源配置中心**Apollo**深度剖析

3

Spring Cloud Config简介

4

面向**业务场景、原理和架构**

5

面向**微服务**

6

结合**案例和实操(Apollo)**

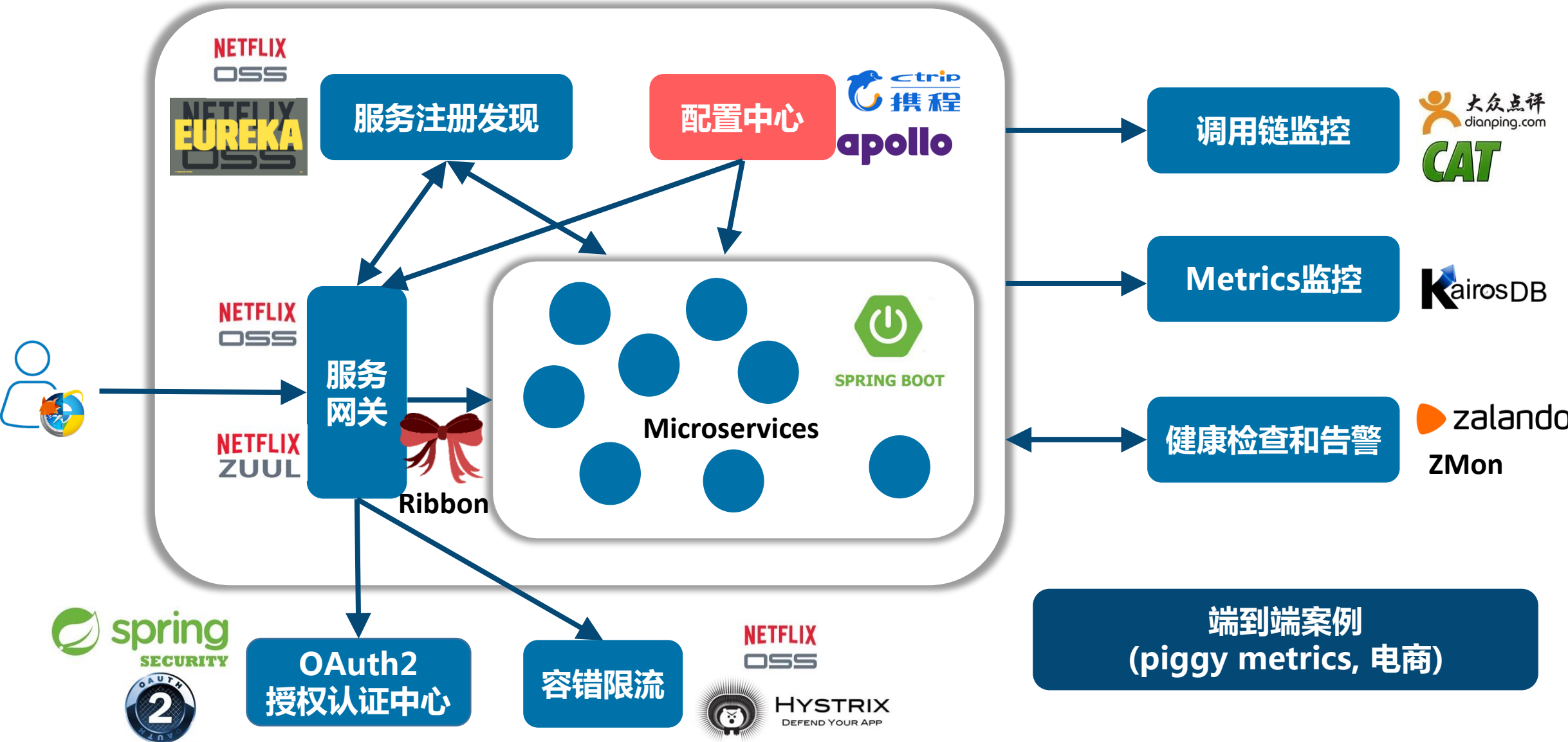
7

结合**生产最佳实践**

杨波的微服务基础架构体系2018预览(draft)



架构和技术栈预览



第

2

部分

业务需求

研发小故事



小D是研发工程师，某天产品说要开发一个双十一商品促销功能

由于产品无法预估促销商品的需求量，于是拍脑袋说，每个用户限购10个！

于是小D在代码里头这么写

```
// 产品需求每个用户限购10个商品
private static final int MAX_QTY_PER_USER = 10;

public boolean purchase(int quality) {
    if (quality > MAX_QTY_PER_USER) {
        throw new IllegalArgumentException(
            String.format("每个用户限购%d个商品", MAX_QTY_PER_USER));
    }
    // 商品购买逻辑
    return true;
}
```

研发小故事

促销当日中午，销售火爆出乎产品预料，产品匆忙跑到小D处，要求赶紧改成每人限购2个！



小D只好放弃午饭，
改代码、回归测试、
上线，整整花了1个
多小时才搞定.....

研发小故事



小S是研发工程师，某天产品说要开发一个双十一商品促销功能

由于产品无法预估促销商品的需求量，于是拍脑袋说，每个用户限购10个！

小S吸取了小D的教训，直接把配置写在了集中式配置中心里头



Key	Value	备注
max-qty-per-user	10	产品需求限购10个

研发小故事

促销当日中午，销售火爆出乎产品预料，产品匆忙跑到小S处，要求赶紧改成每人限购2个！

10秒钟就搞定~~



研发小故事



为什么要配置中心

传统应用配置问题



主要采用本地文件静态配置

本地静态配置导致在运行时无法动态修改

配置散乱格式不标准

有的用xml格式，有的用properties，有的存DB

易引发生产事故

发布的时候容易将非生产的配置带到生产上，引发事故

配置修改麻烦，周期长

当部署的服务器很多时，修改配置费时费力

配置信息缺少安全审计和版本控制功能

事后无法追溯，谁改的？改了什么？什么时候改的？

当出现问题无法及时回滚

配置中心解决办法

主要采用本地文件静态配置

- 集中式配置，所有配置信息都存放配置中心

易引发生产事故

- 环境隔离，不同的环境对应不同的配置，互不干扰
- 配置错误，可以立即修改，即时生效

配置信息缺少安全审计和版本控制功能

- 所有修改有历史记录，方便查找谁和时间
- 可按需回退到历史版本

配置散乱格式不标准

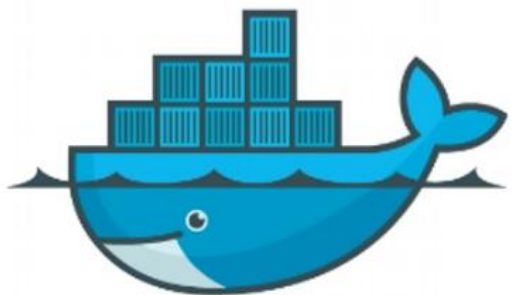
- 配置中心统一管理格式，用户不必关注格式

配置修改麻烦，周期长

- 配置集中一次修改，实时通知到所有客户端

现代交付需求

Immutable Infrastructure

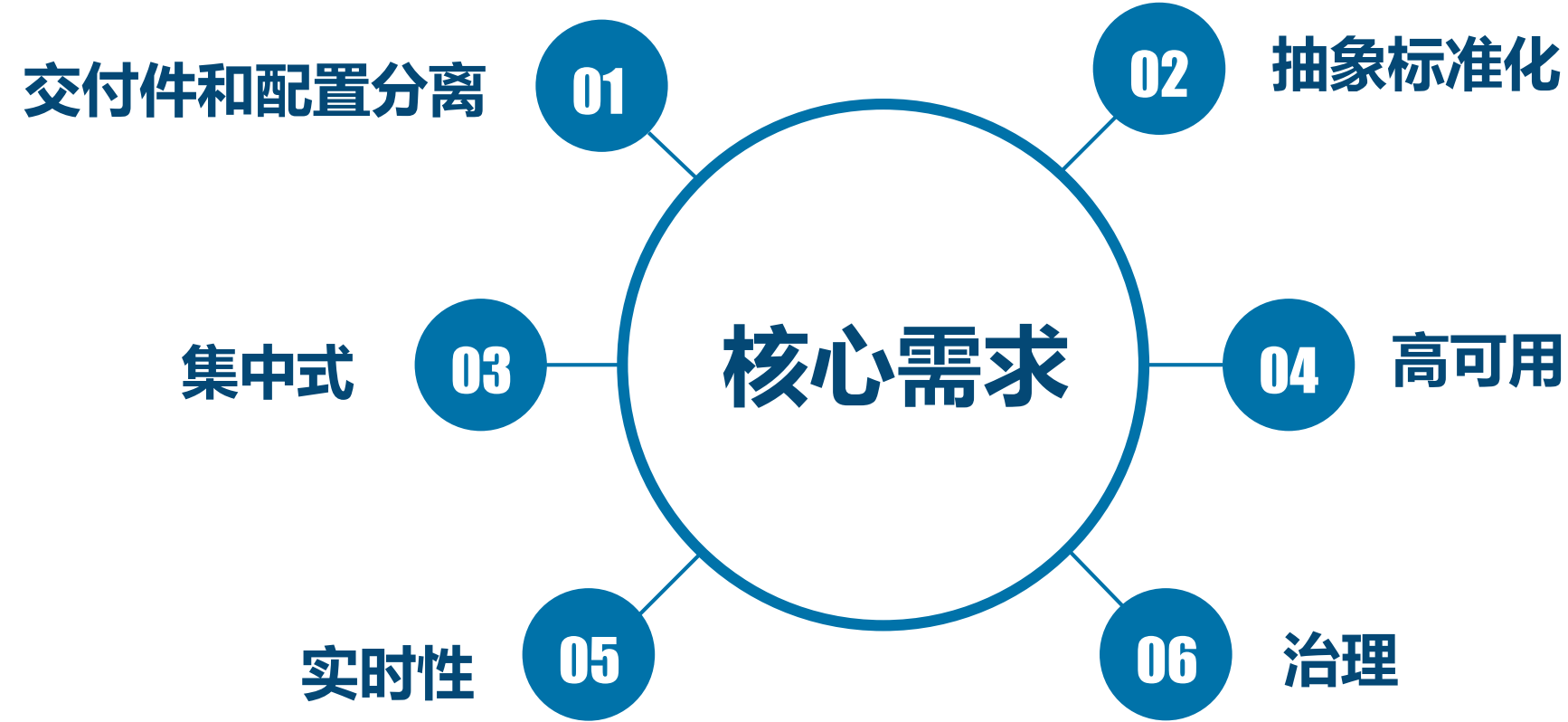


MicroServices

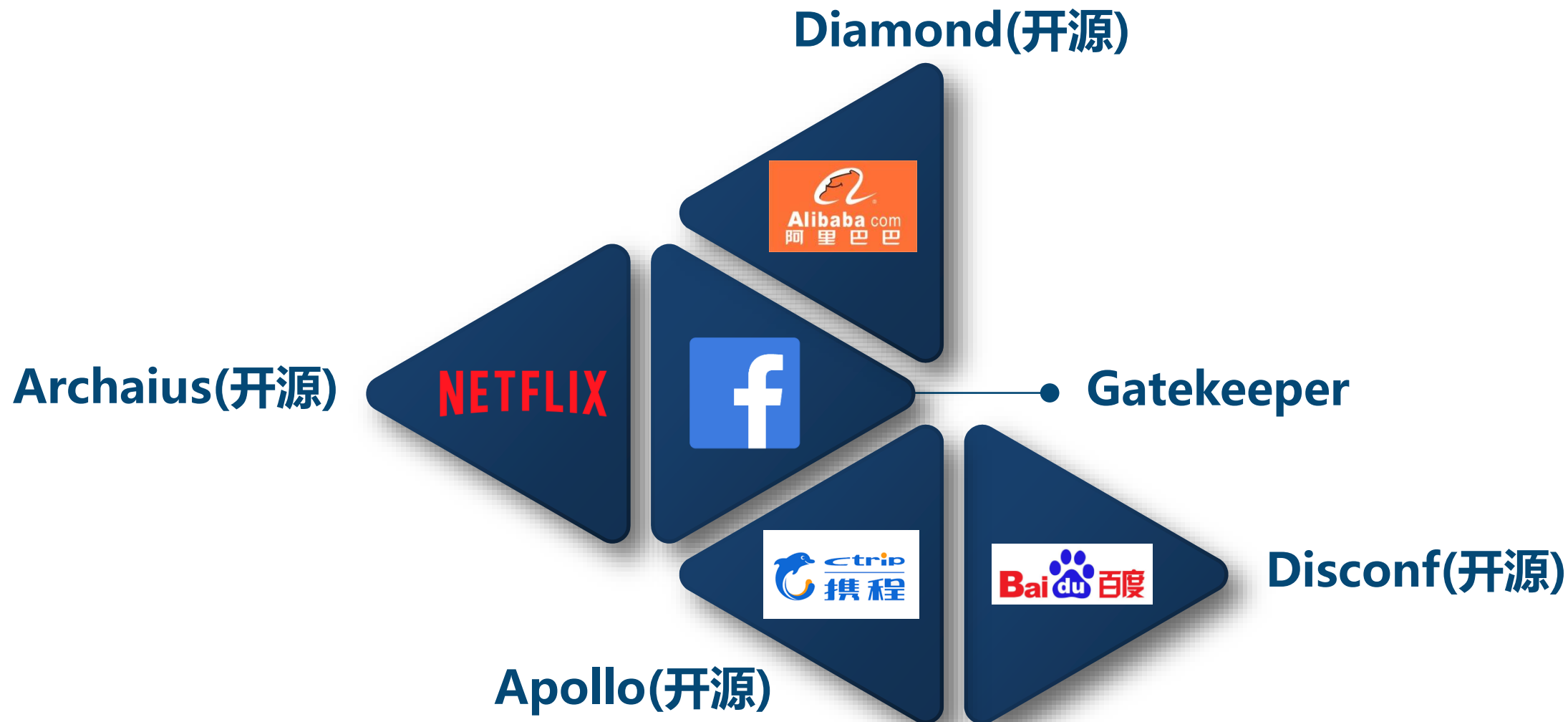


CLOUD NATIVE
COMPUTING FOUNDATION

现代配置核心需求



公司案例和产品



第

3

部分

配置定义和场景

配置基本概念



配置定义

- 可独立于程序的可配变量
- 同一份程序在不同配置下会有不同行为
- 连接字符串，应用配置，业务配置

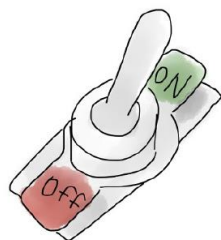
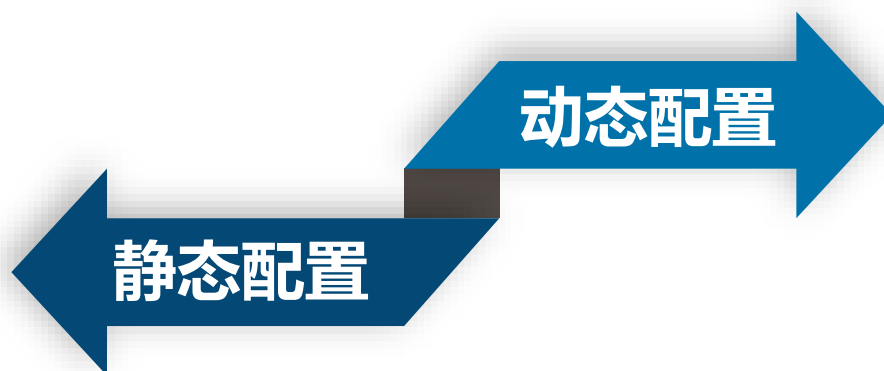
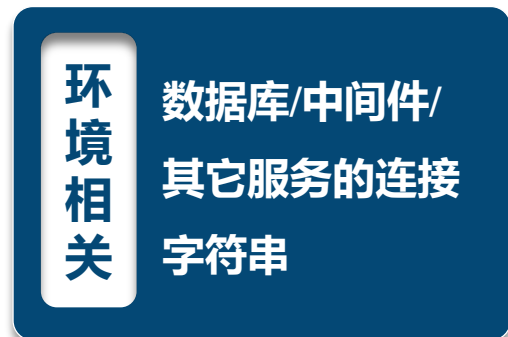
配置形态

- 程序内部hardcode (反模式，不建议！)
- 配置文件
- 环境变量
- 启动参数
- 基于数据库

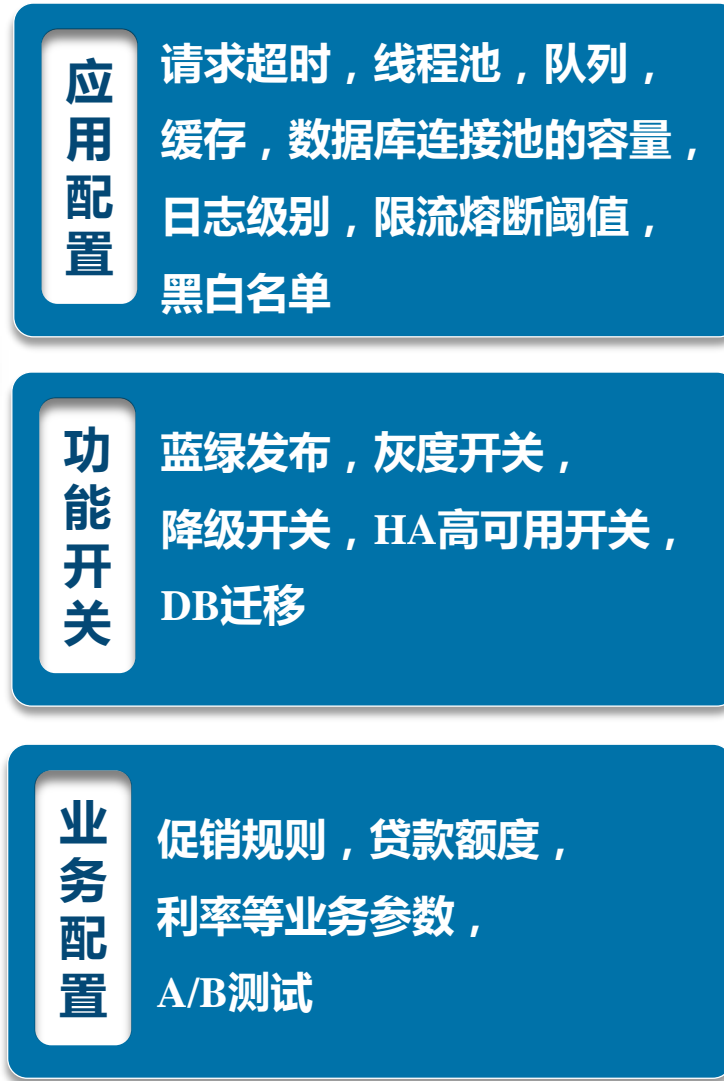
配置治理

- 权限控制和审计
- 不同环境、集群配置管理
- 框架类组件配置管理
- 灰度发布

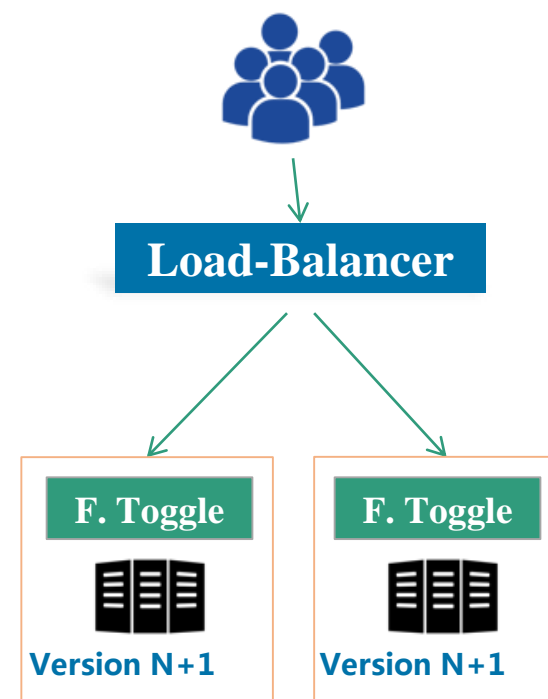
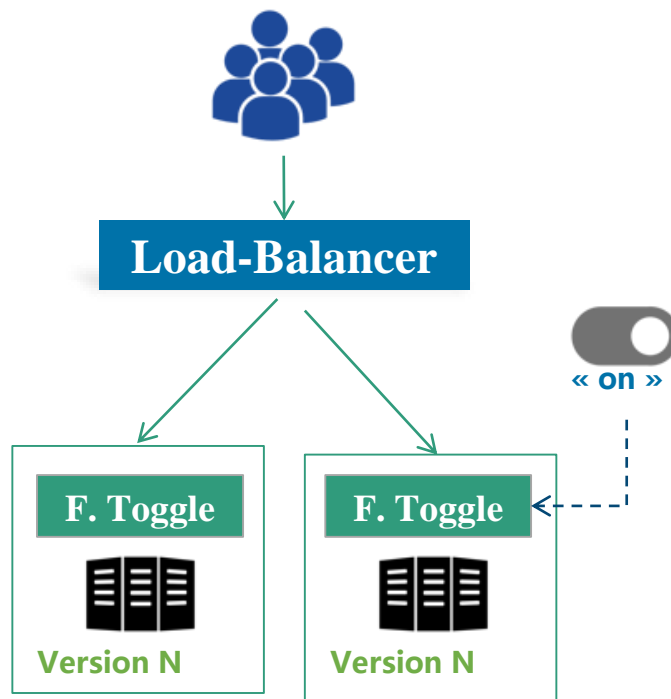
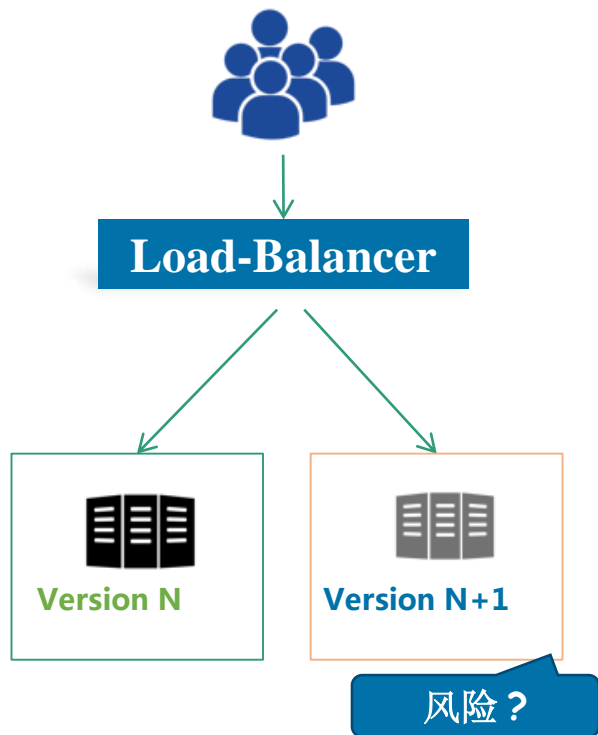
配置分类和场景



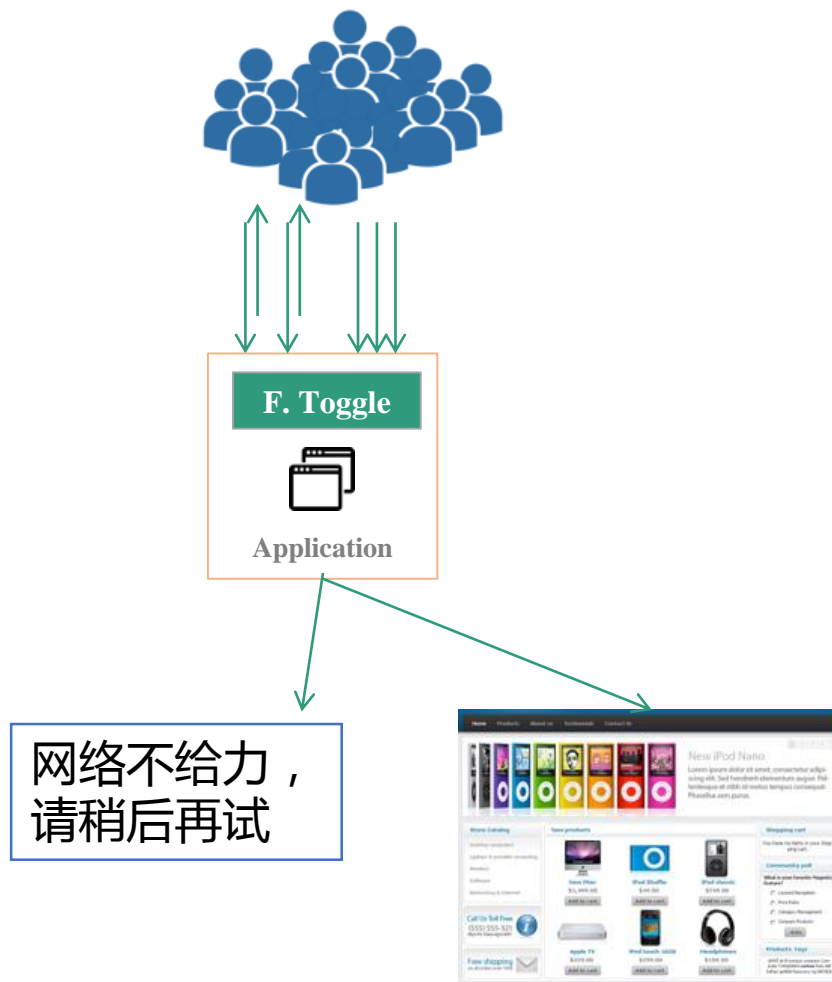
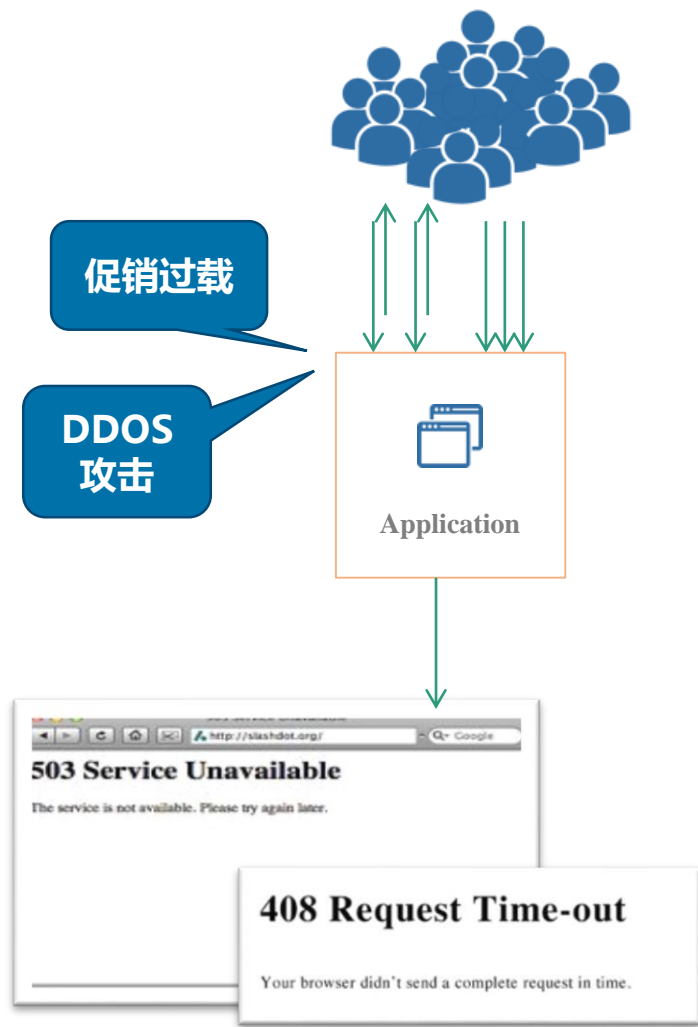
Feature Flag/Toggle/Switch



蓝绿发布



功能降级



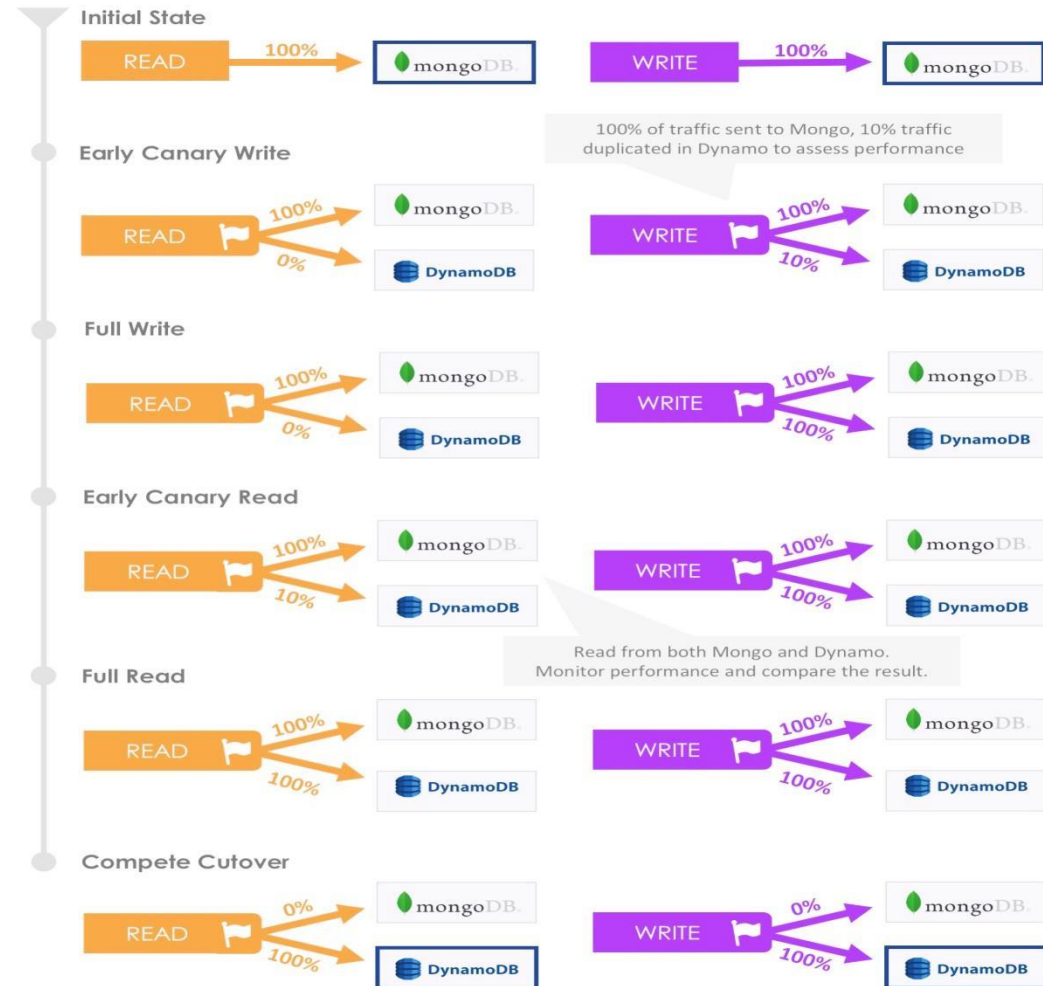
DB/Schema迁移

<https://blog.launchdarkly.com/feature-flagging-to-mitigate-risk-in-database-migration/>



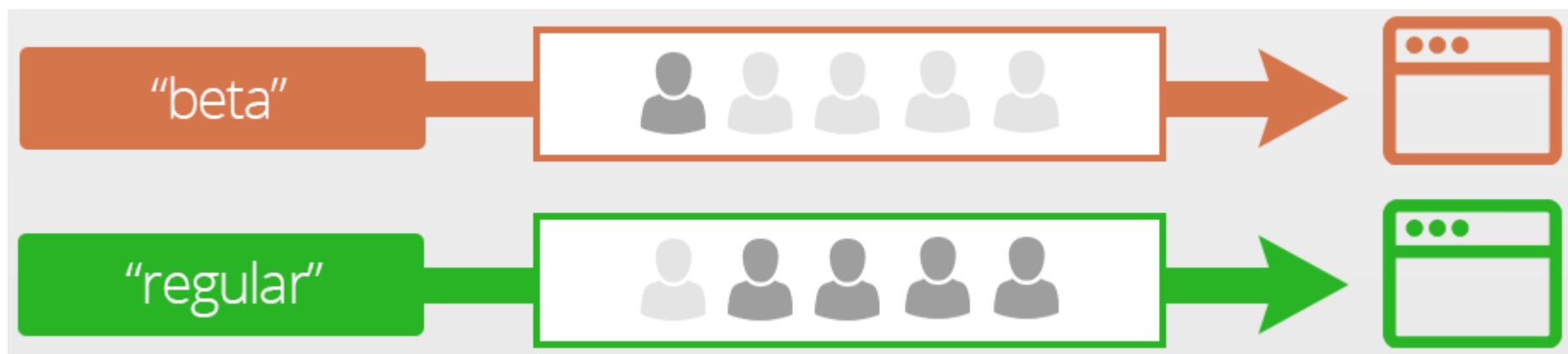
Database Migration Using Feature Flags

Migrating from Mongo to Dynamo



电商结账功能的A/B测试

```
if ((ab_test_flag==true  
    && user==beta)  
    || ab_test_flag==false)
```



```
if (ab_test_flag==true  
    && user==regular)
```

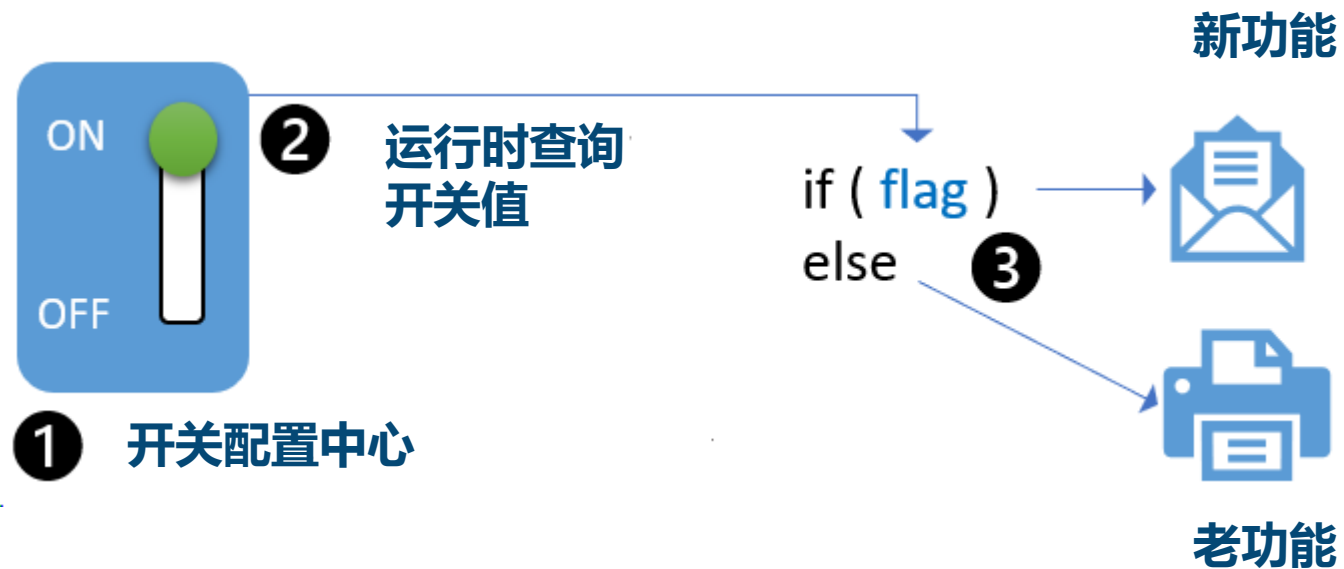
第

4

部分

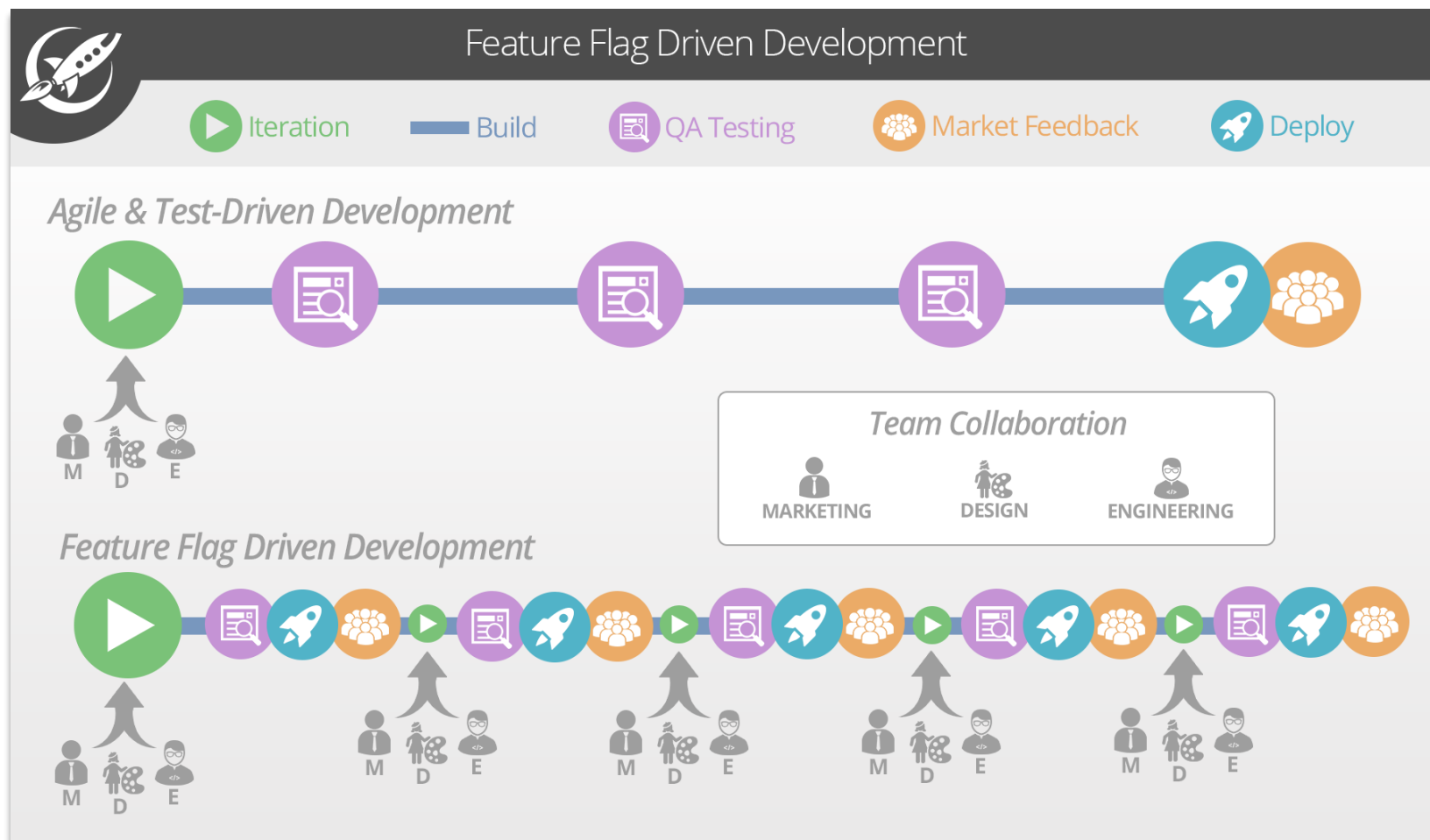
开关驱动开发原理和实践

开关驱动开发



- Feature Flag Driven Development
- 可结合A/B测试
- DevOps最佳实践

基于开关的持续交付



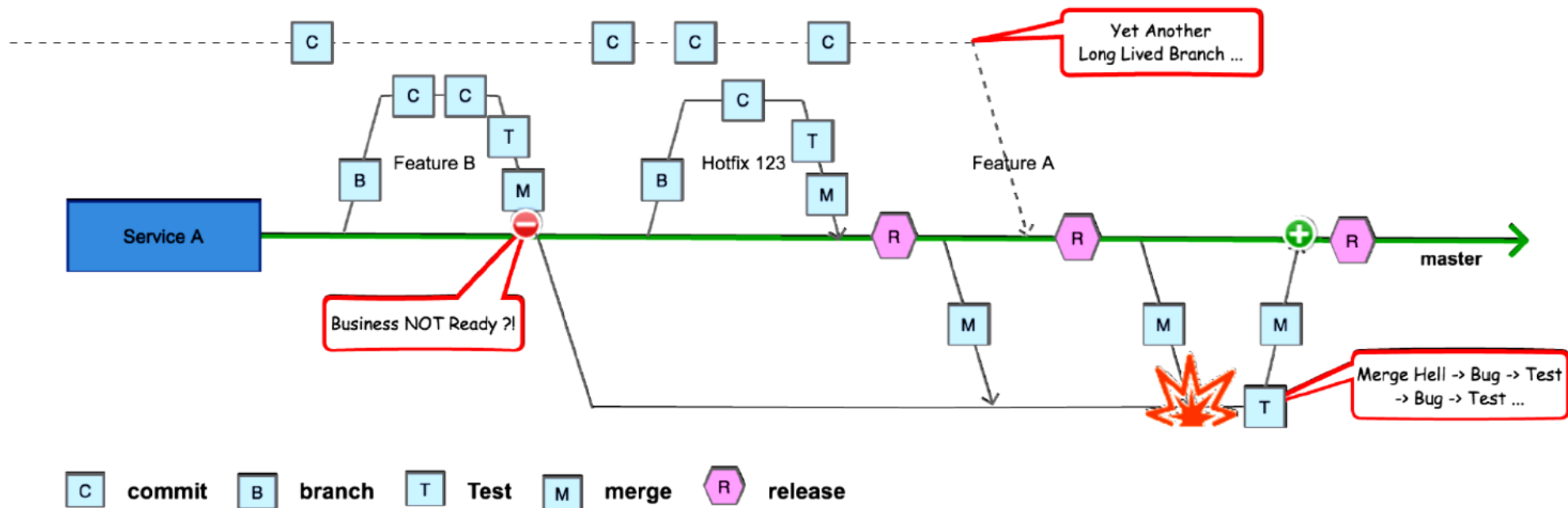
图片来自：<https://blog.launchdarkly.com/feature-flag-driven-development/>

长生命期分支的问题

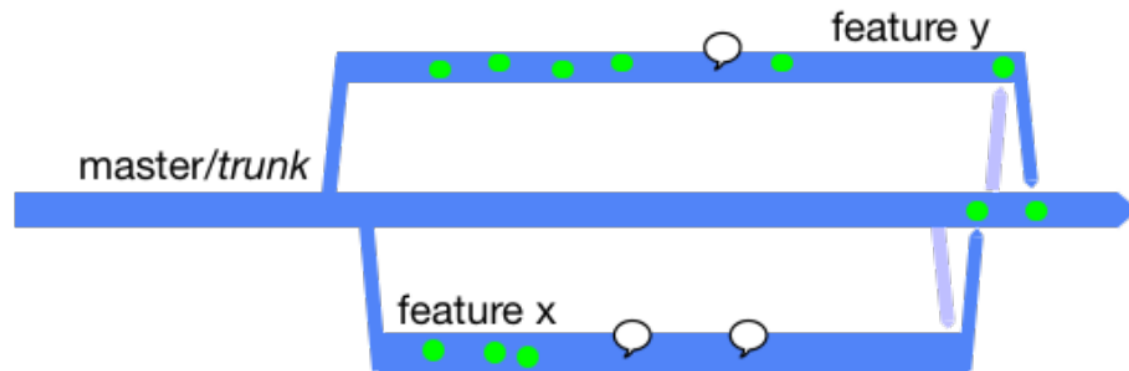
Merge hell痛点

开销大效率低

微服务

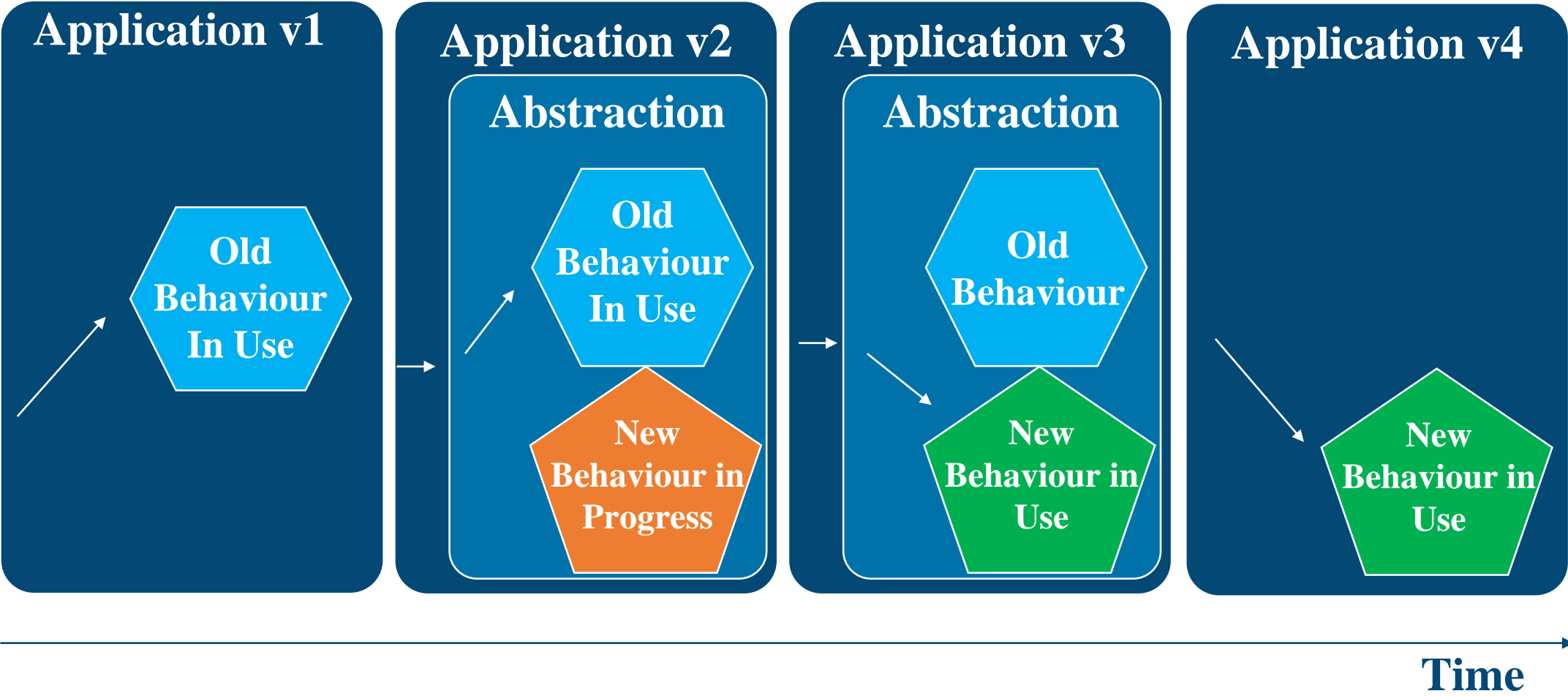


Trunk based Development(TBD)



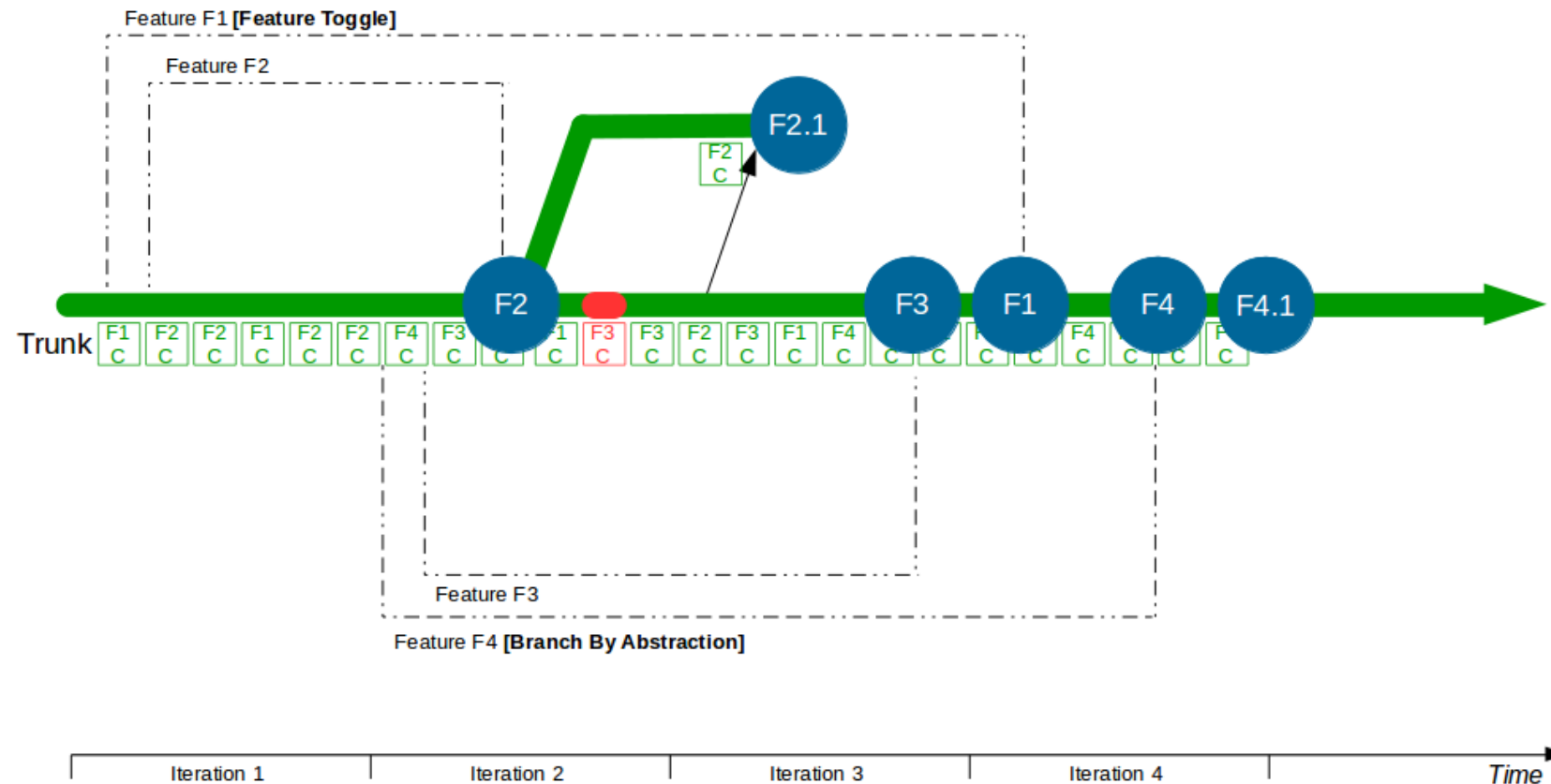
- 每天至少向**主干Trunk**提交一次
- Trunk**始终在可发布**状态
- 将新功能代码(或者未完成代码)影藏在**功能开关**后面
- 使用**branch by abstraction**进行重构
- **可以开分支branch** , **但是一般**不超过2天

Branch by Abstraction重构



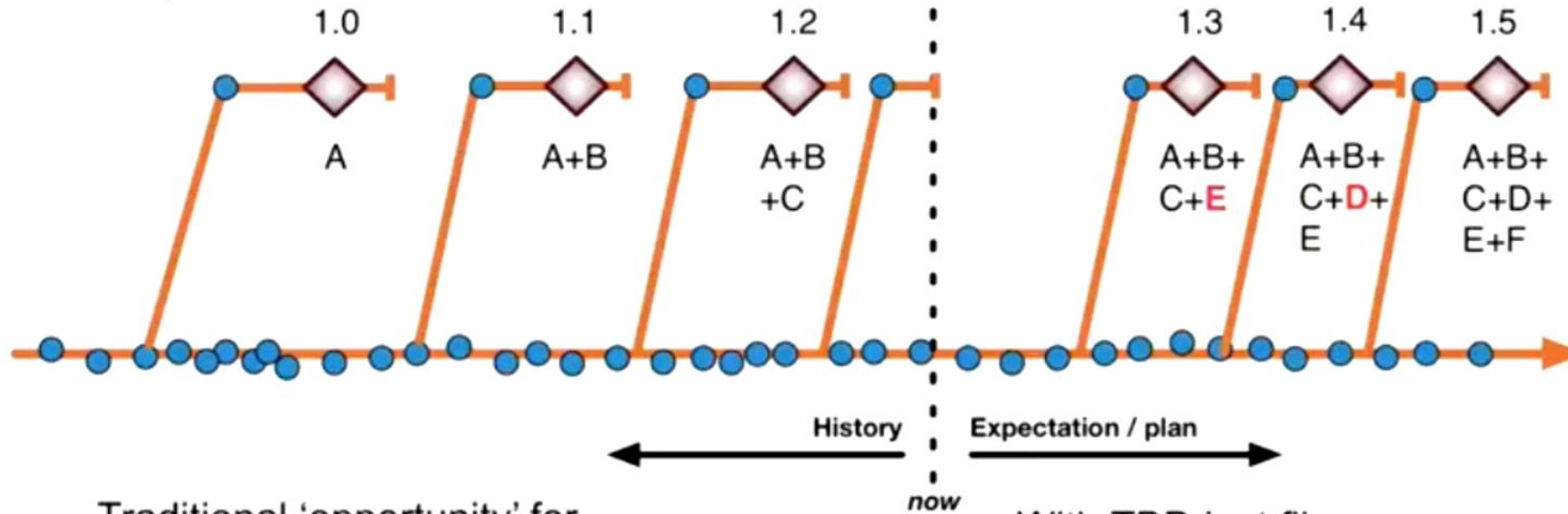
TBD案例

<https://www.continuousdeliveryconsulting.com/blog/organisation-pattern-trunk-based-development/>



Re-planning

Re-plan needed ..



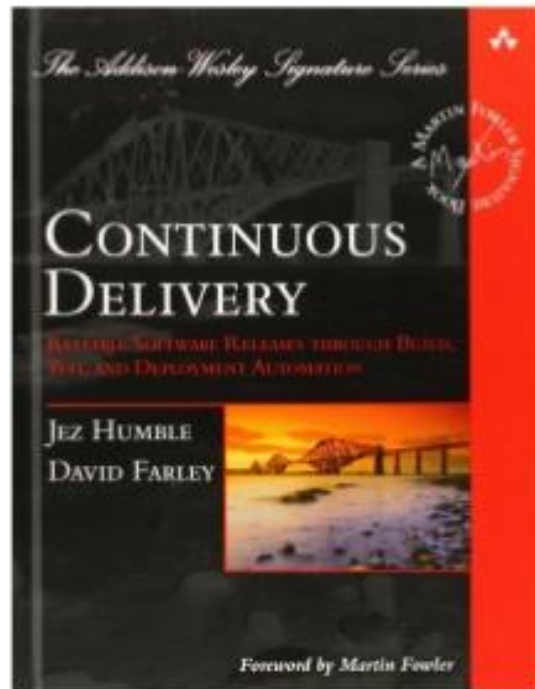
Traditional 'opportunity' for

- Unmerge
- Comment-out
- **No real developer work**

With TBD just flip some toggles/flags,
and make a new CI pipeline,
work through failing tests.

TBD & CD

We can't emphasize enough how important this practice is in enabling continuous delivery of valuable, working software.



Dave Farley
@davefarley77



Jez Humble
@jezhumble

优劣权衡

优

- 新功能和代码发布分离，减轻发布风险
- 迭代速度快，快速创新实验
- 可定制高级A/B测试
- 相比复杂发布系统，投入成本相对低
- 没有分支开发(Feature Branch)的合并冲突问题



劣

- 代码侵入，技术债，需要定期清理
- 需要开关配置中心配合
- 需要DevOps文化和流程配合

VS

第

5

部分

携程Apollo配置中心简介

背景



携程框架研发
部研发

作者
宋顺（研发）
吴其敏（架构）

<https://github.com/ctripcorp/apollo>
>4.8k Stars
>1.8k Forks

2016年研发
并开源

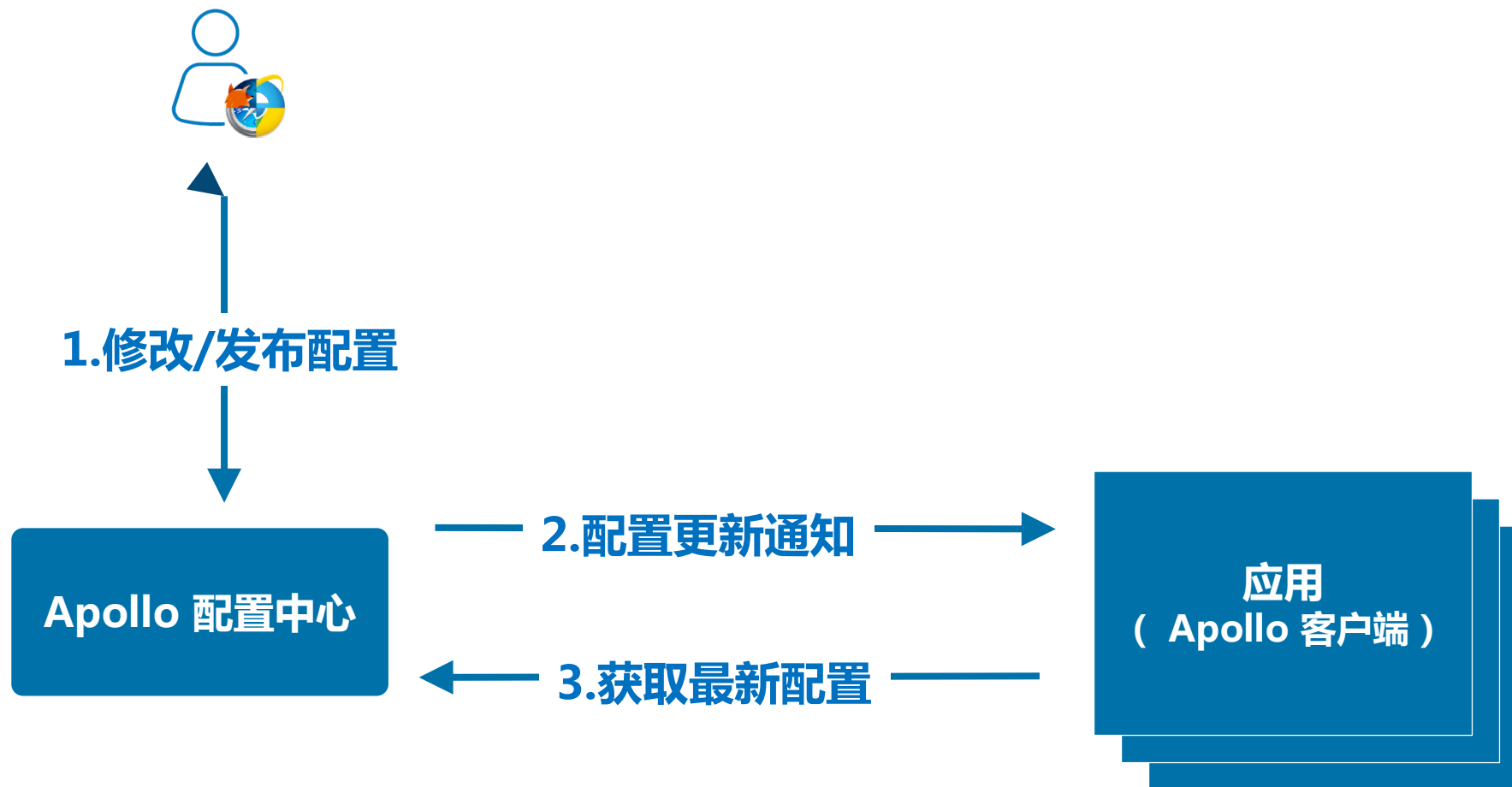
在携程稳定运行，
服务10万+实例

功能亮点



- 统一管理不同环境、不同集群的配置
- 配置修改实时生效（热发布）
- 版本发布管理
- 灰度发布
- 权限管理、发布审核、操作审计
- 客户端配置信息监控
- 提供Java和.Net原生客户端
- 提供开放平台API
- 部署简单
- 文档完善

简化架构



Apollo主界面

Apollo 配置中心

搜索项目(AppId、项目名)

帮助

song_s

环境列表

FAT

default

dev

UAT

PRO

default

SHAOY

SHAJQ

项目信息

AppId: 100004458

应用名: apollo-demo

部门: 框架(FX)

负责人: song_s

邮箱: song_s@ctrip.com

管理项目

添加集群

添加Namespace

私有

application properties

发布 回滚 发布历史 授权 灰度 更多

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
已发布	timeout	3000		song_s	2017-02-16 13:24:58	编辑 删除
已发布	kibana.url	http://1.1.1.2:5600		song_s	2016-11-25 20:57:27	编辑 删除
已发布	elastic.document.type	biz1		song_s	2017-01-11 19:14:06	编辑 删除
已发布	elastic.cluster.name	es-cluster		song_s	2016-10-18 19:57:29	编辑 删除
已发布	elastic.cluster	2.2.2.2:9300,4.4.4.4:9300		zhanglea	2016-12-08 14:19:43	编辑 删除
已发布	page.size	20		song_s	2016-12-27 14:58:56	编辑 删除
已发布	zookeeper.address	10.1.12.2		song_s	2016-10-19 11:33:50	编辑 删除

关联

FX.apollo properties

发布 回滚 发布历史 授权 灰度 更多

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
已发布	servers	3.3.3.3,4.4.4.4		song_s	2017-02-16 13:26:27	编辑 删除

公共的配置 (AppId:100003173, Cluster:default)

Key	Value	备注	最后修改人	最后修改时间	操作
batch	2000	样例项目会使用到，勿删。	song_s	2017-02-16 13:27:07	删除
servers	1.1.1.1,2.2.2.2	样例项目会使用到，勿删。	song_s	2016-10-12 14:03:34	

已知接入公司



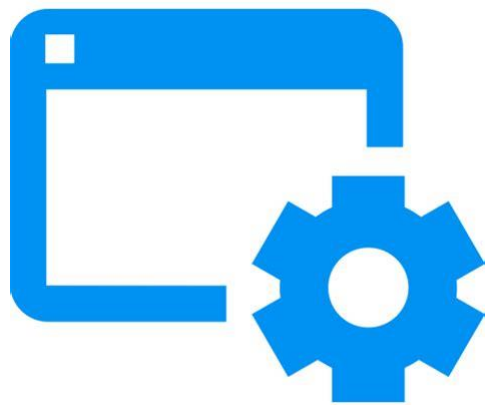
第

6

部分

Apollo核心概念

核心概念~应用(application)



使用配置的应用

有唯一标识appId :

- Java: classpath:/META-INF/app.properties -> appid
- .Net: app.config -> AppID

核心概念~环境(environment)

配置对应的环境

DEV, FAT, UAT, PRO

- server.properties -> env
- C:\opt\settings\server.properties或/opt/settings/server.properties

管理中心

DEV

app1

app2

appx

FAT

app1

app2

appx

UAT

app1

app2

appx

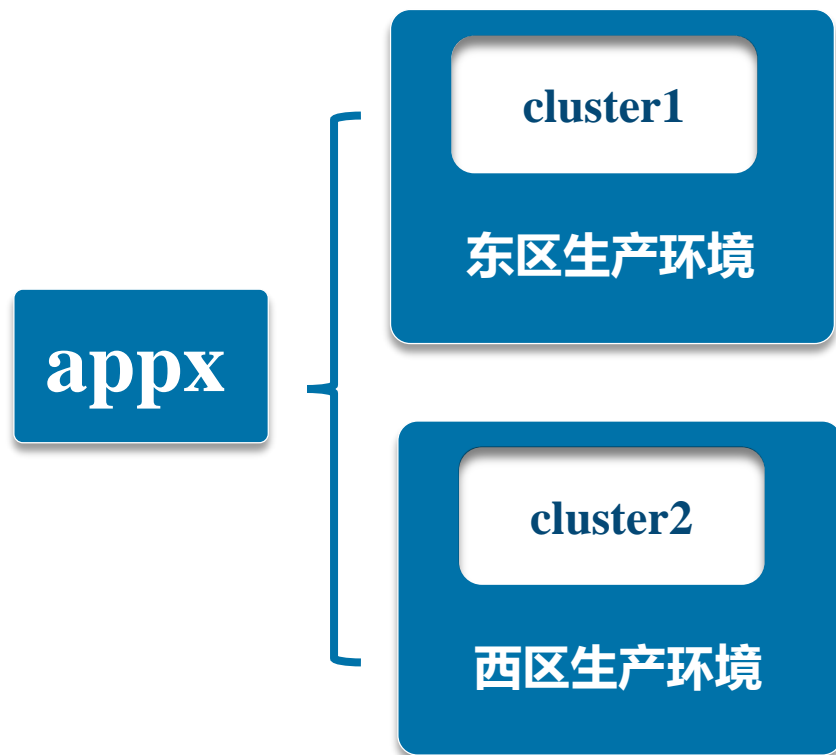
PRO

app1

app2

appx

核心概念~集群(cluster)



一个应用下不同实例的分组

对不同的cluster，可以有不同的配置

- 比如kafka地址针对上海机房和成都机房可以有不一样的配置

默认数据中心作为cluster

- server.properties -> idc
- C:\opt\settings\server.properties或
/opt/settings/server.properties

核心概念~名字空间(namespace)

一个应用下不同配置的分组

- 数据库配置
- 服务框架配置
- 应用元数据配置

01

应用默认有自己的配置
namespace – application

02

03

也可以使用公共组件的配置
namespace

- 如服务框架，MQ客户端等
- 可以通过继承方式对公共组件的配置做调整，如MQ消费者线程数

名字空间类型

私有(Private)类型

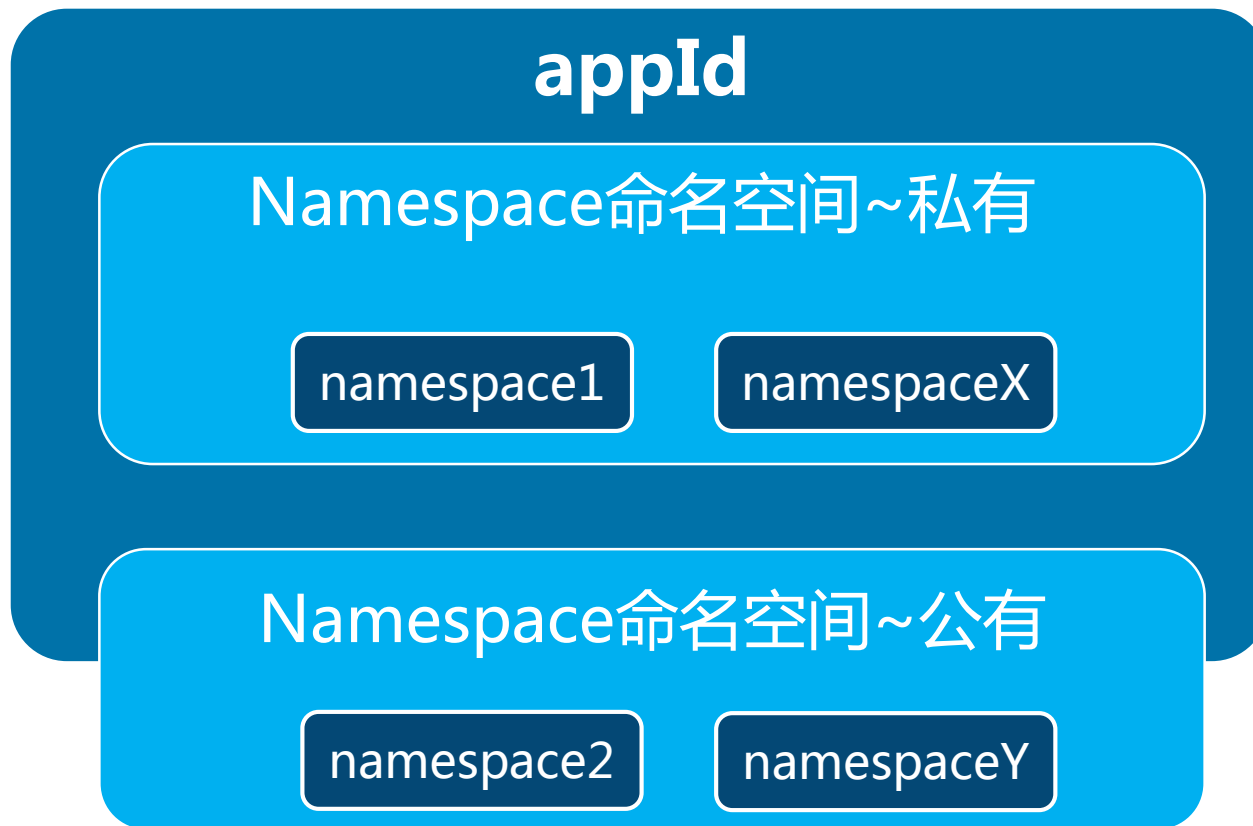
只能被所属应用获取

公有(Public)类型

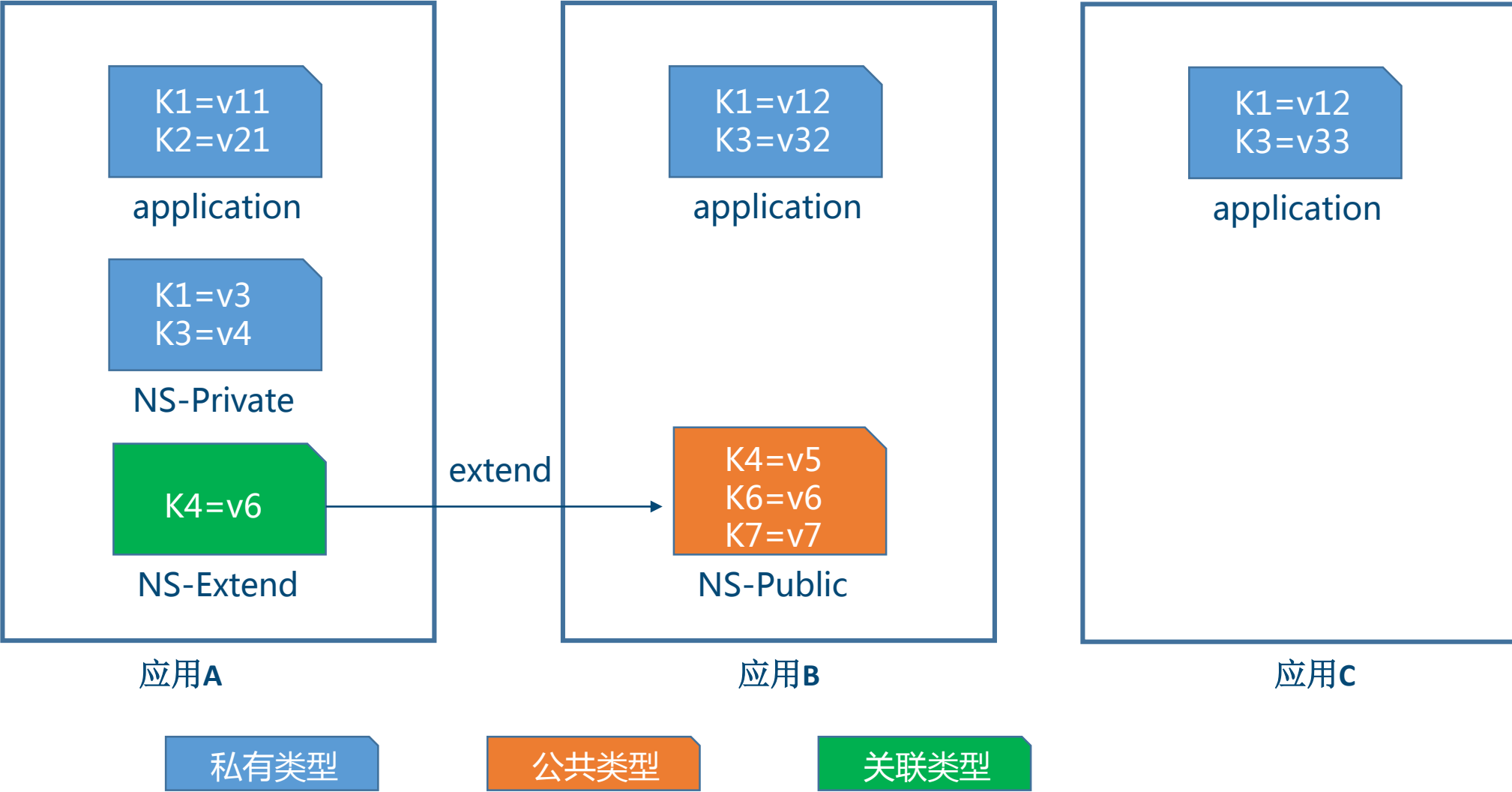
- 公有场景
 - 部门级别共享配置
 - 小组级别共享配置
 - 中间件客户端的配置
- 必须全局唯一

关联类型(继承类型)

- 私有继承公有并覆盖
- 定制公共组件配置场景



关联类型案例



核心概念~配置项(item)

表示可配置项

支持properties/json/xml格式

定位方式

- 私有配置env+app+cluster+namespace+item_key
- 公有配置env+cluster+namespace+item_key



{ 'key': 'value' }

核心概念~权限



- 系统管理员拥有所有的权限
- 创建者可以代为创建项目，责任人是默认的项目管理员，一般创建者=责任人
- 项目管理员可以创建Namespace，集群，管理项目和Namespace权限
- 编辑权限只能编辑不能发布
- 发布权限只能发布不能编辑
- 查看，普通用户可以搜索查看所有项目配置，但不能做相关操作

第

7

部分

Apollo快速起步（ Lab ）

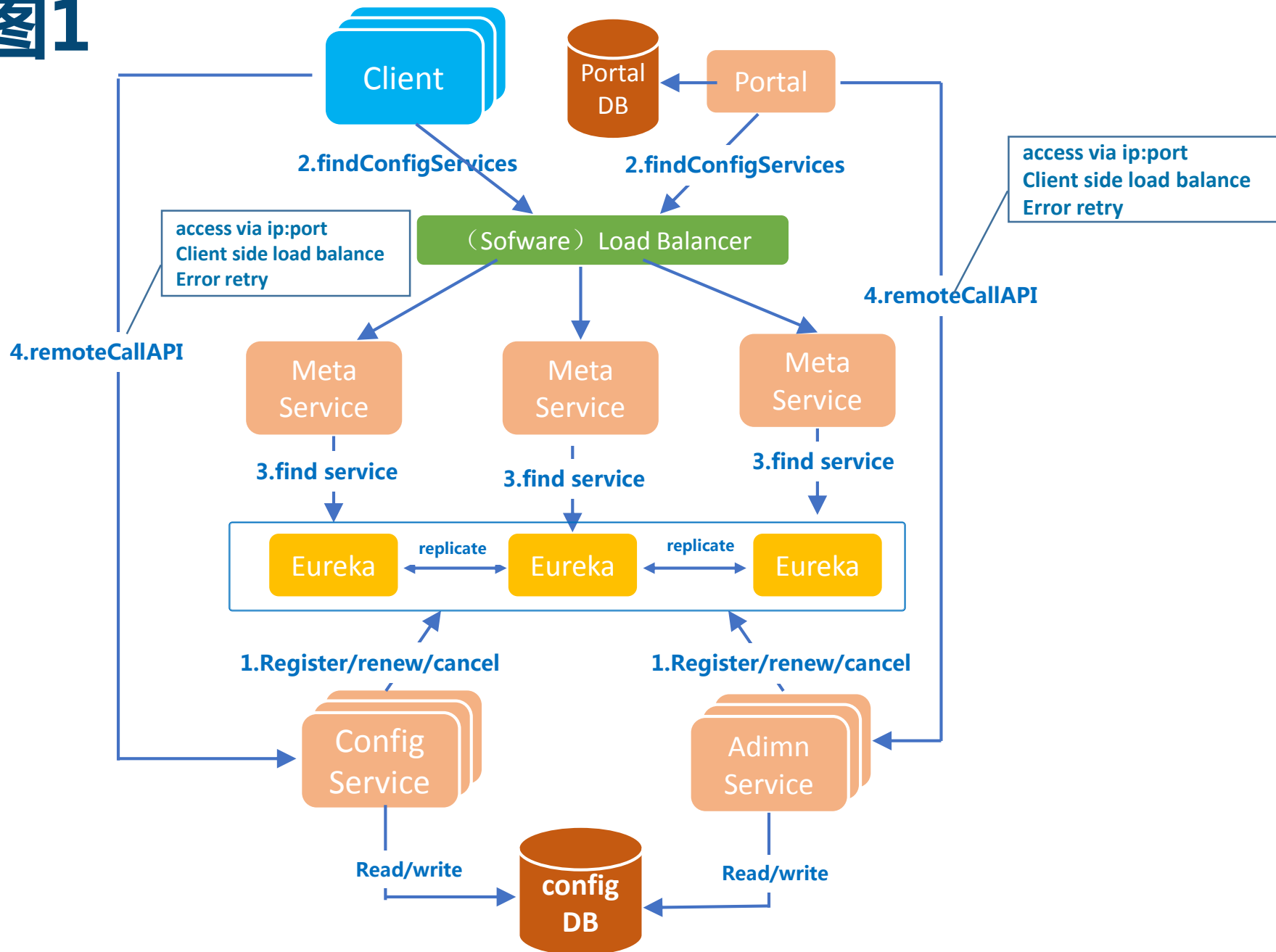
第

8

部分

Apollo架构设计之服务器端

架构视图1



模块介绍1

Config Service

- 配置获取接口
- 配置推送接口
- 服务Apollo客户端

Admin Service

- 配置管理接口
- 配置修改、发布接口
- 服务Portal

Meta Server

- Portal通过域名访问Meta Server获取Admin Service服务列表
- Client通过域名访问Meta Server获取Config Service服务列表
- 相当于一个Eureka Proxy
- 逻辑角色，和Config Service住在一起部署

模块介绍2

Eureka

- 服务注册和发现
- Config/Admin Service注册并报心跳
- 和Config Service住在一起部署

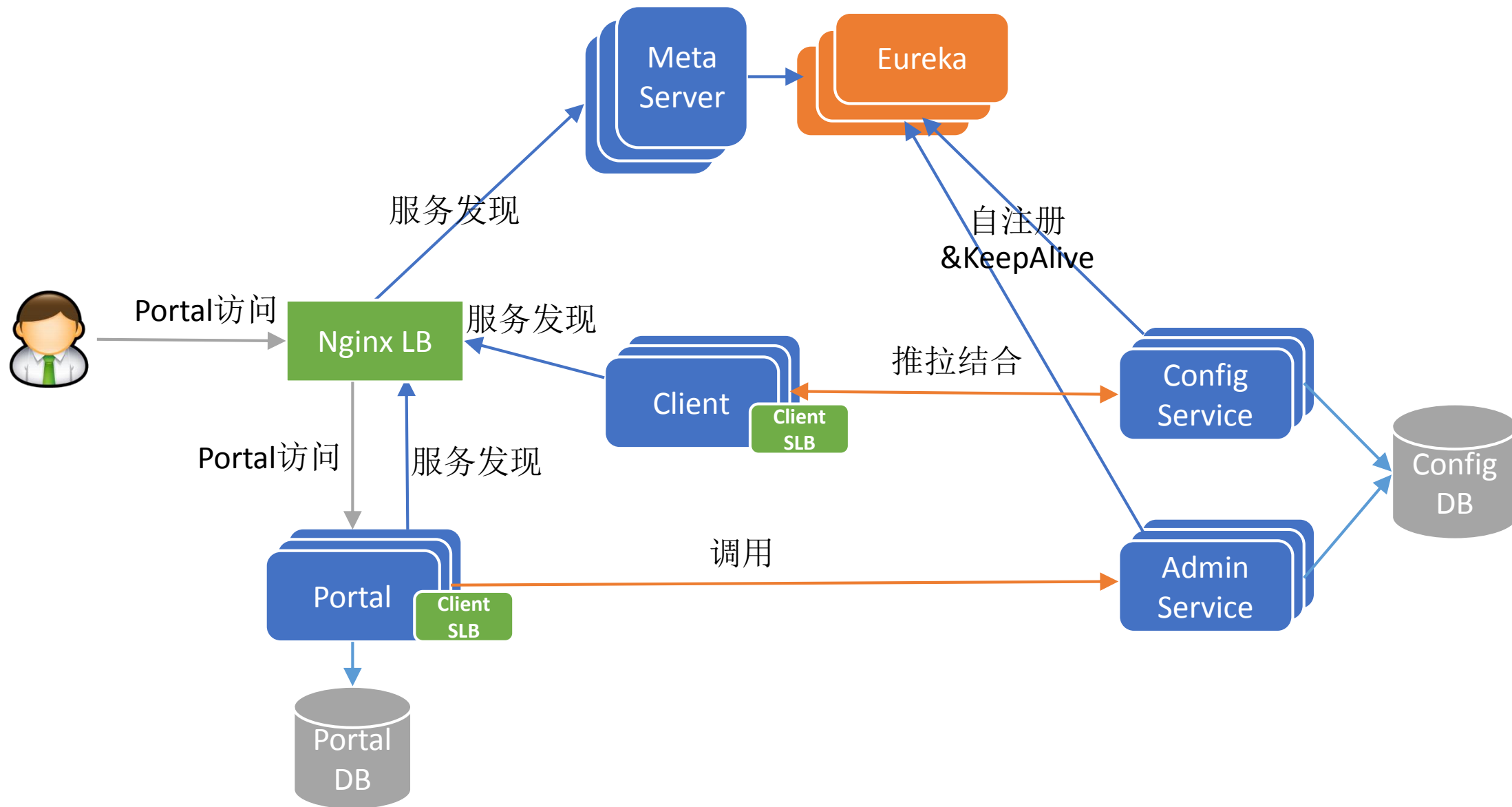
Portal

- 配置管理界面
- 通过Meta Server获取Admin Service服务列表
- 客户端软负载

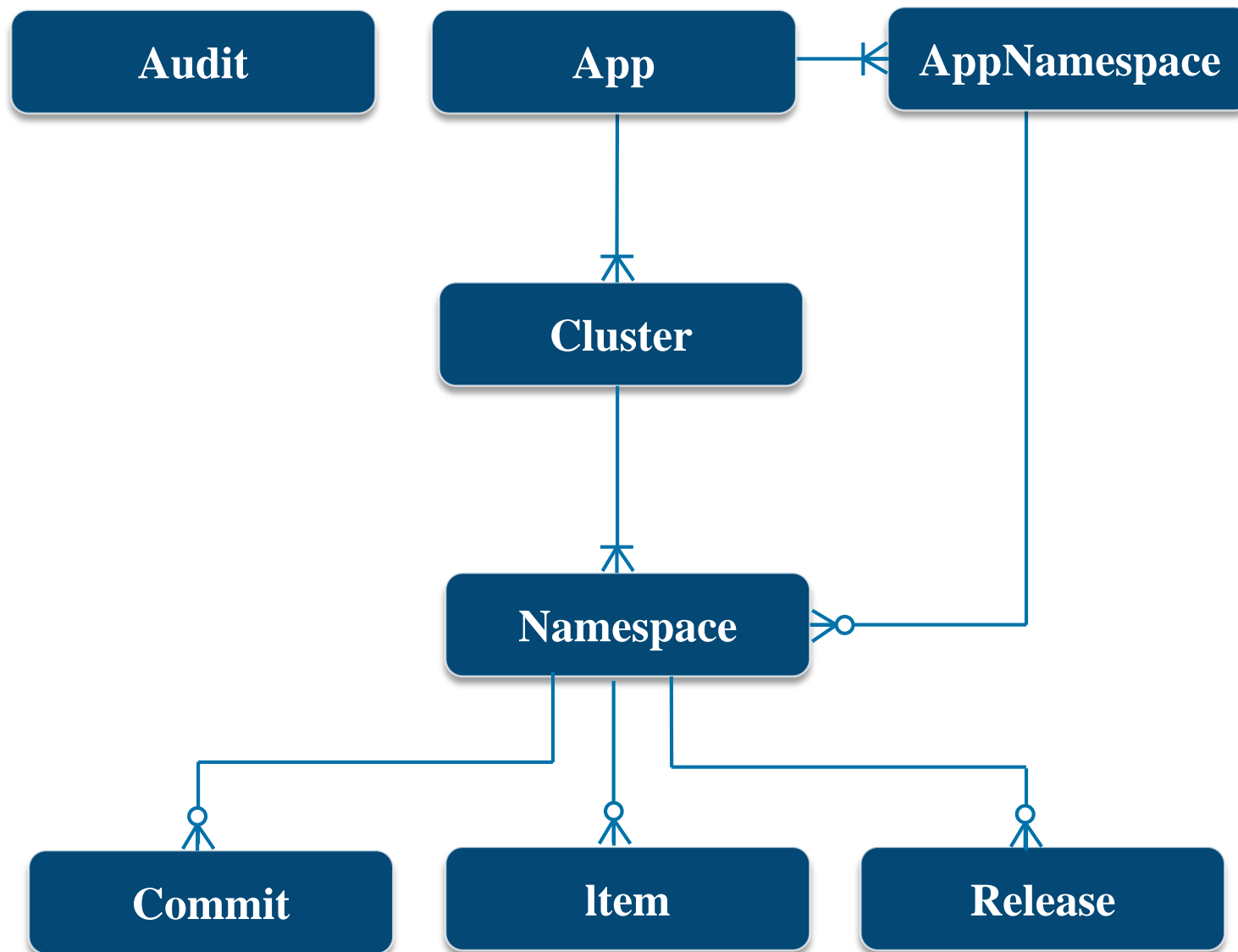
Client

- 应用获取配置，实时更新
- 通过Meta Server获取Config Service服务列表
- 客户端软负载

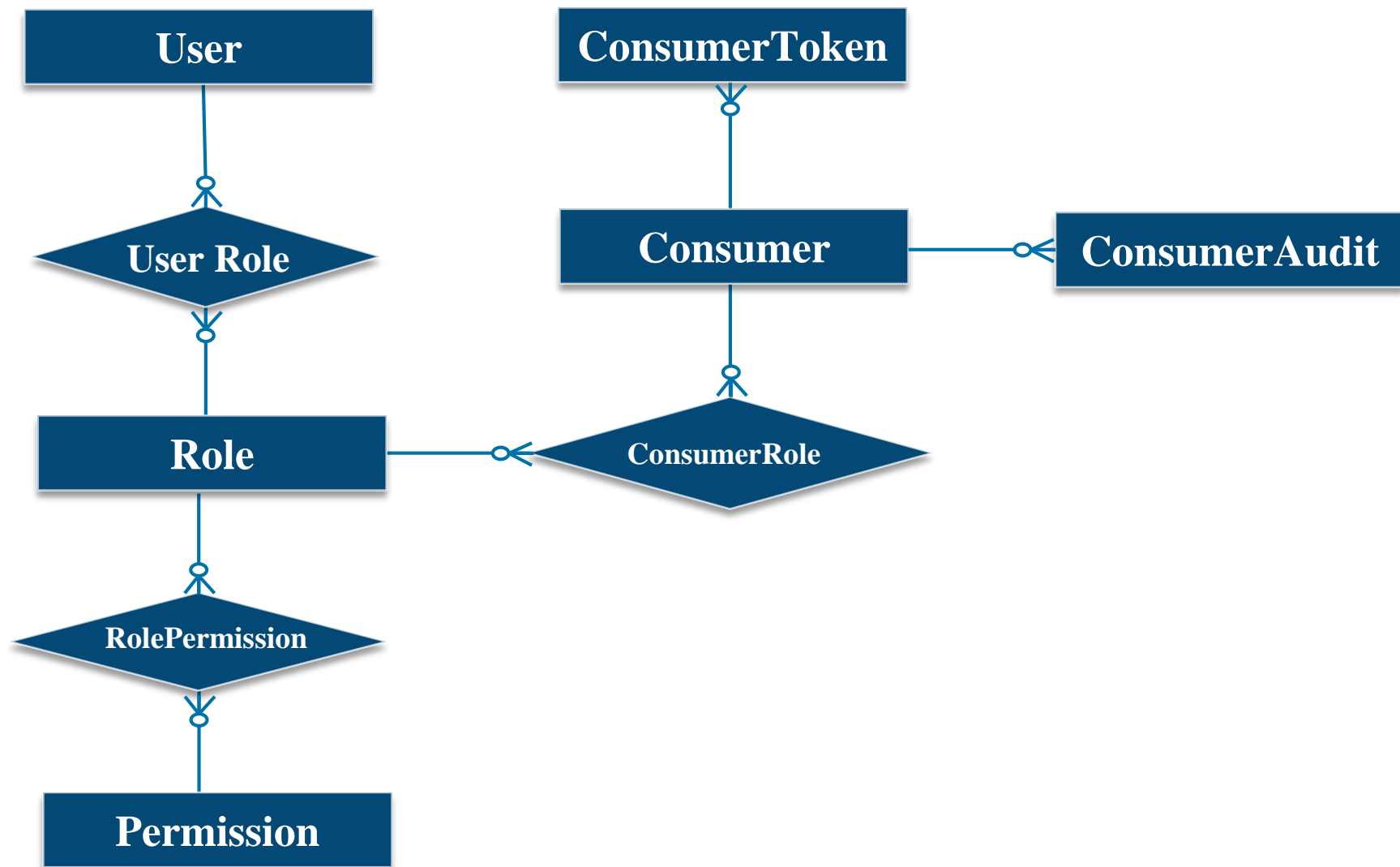
架构视图2



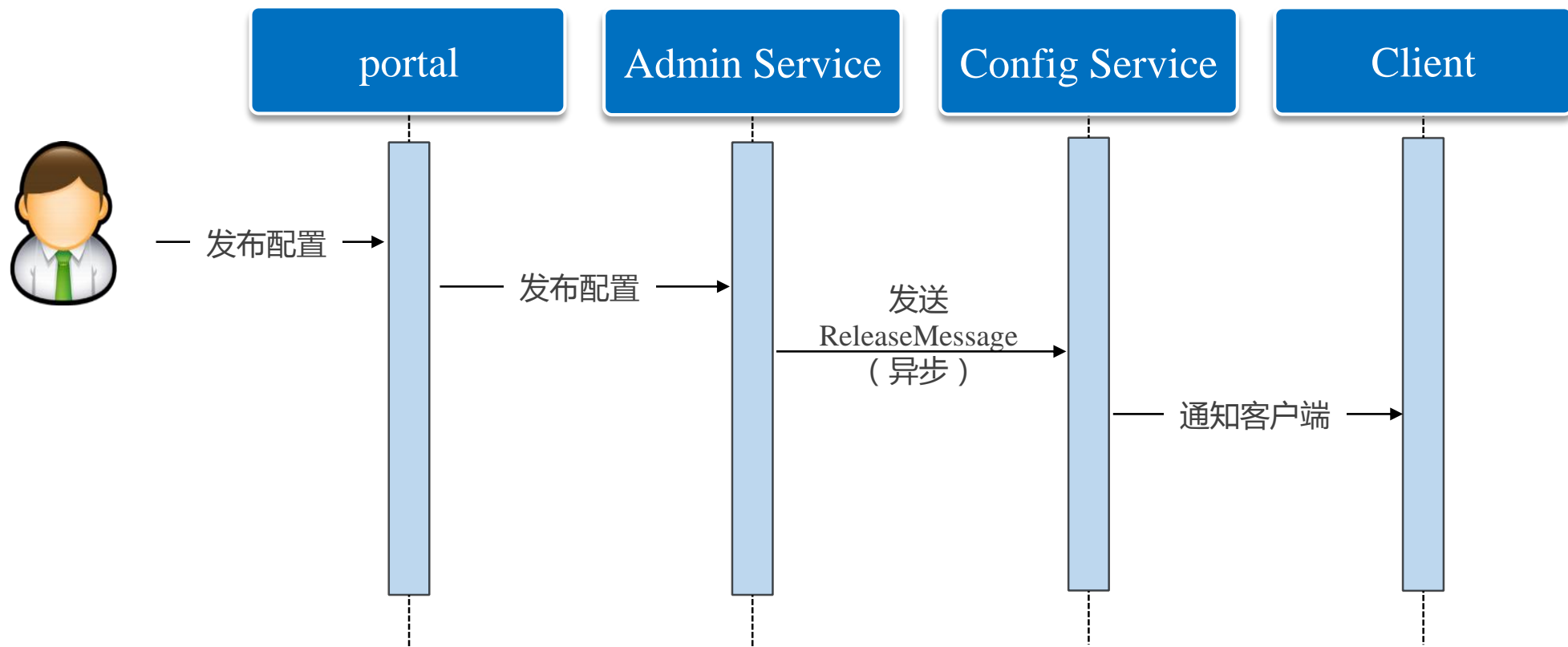
领域模型



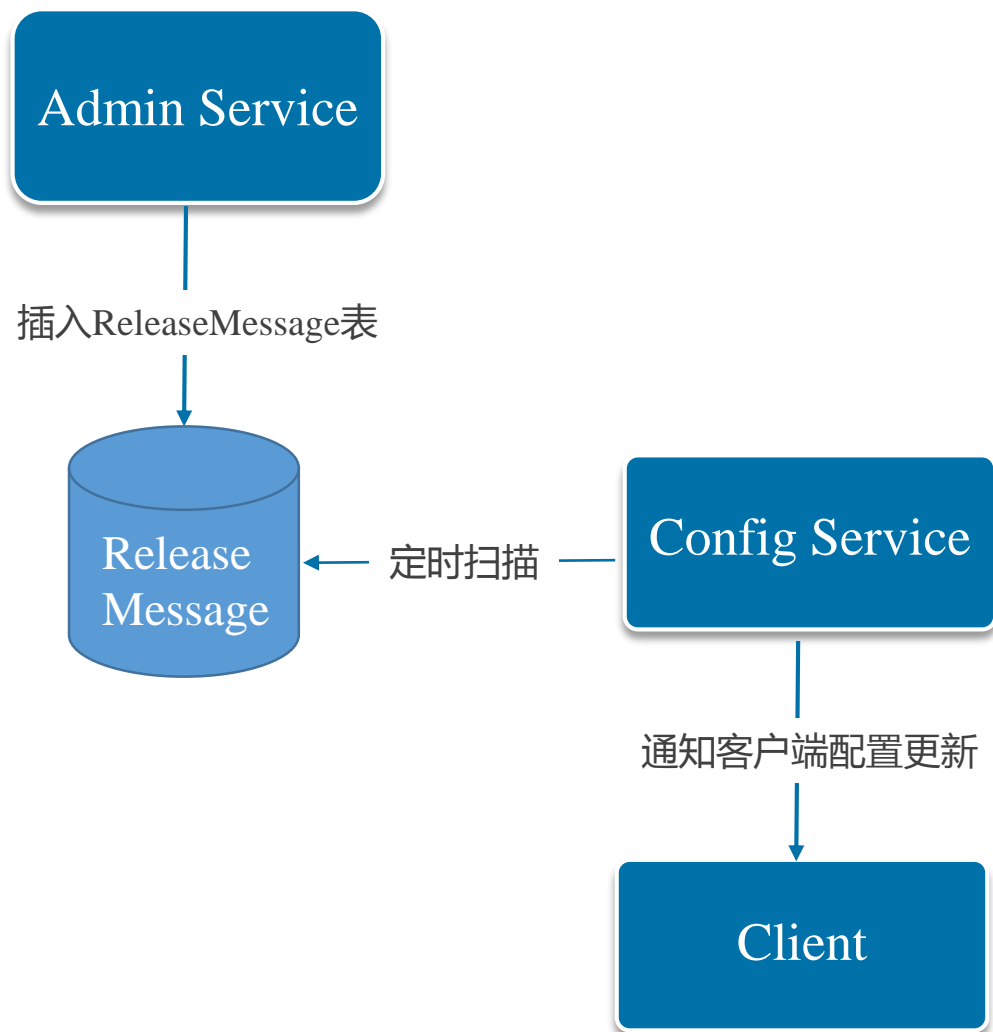
权限模型



实时推送设计



ReleaseMessage实现



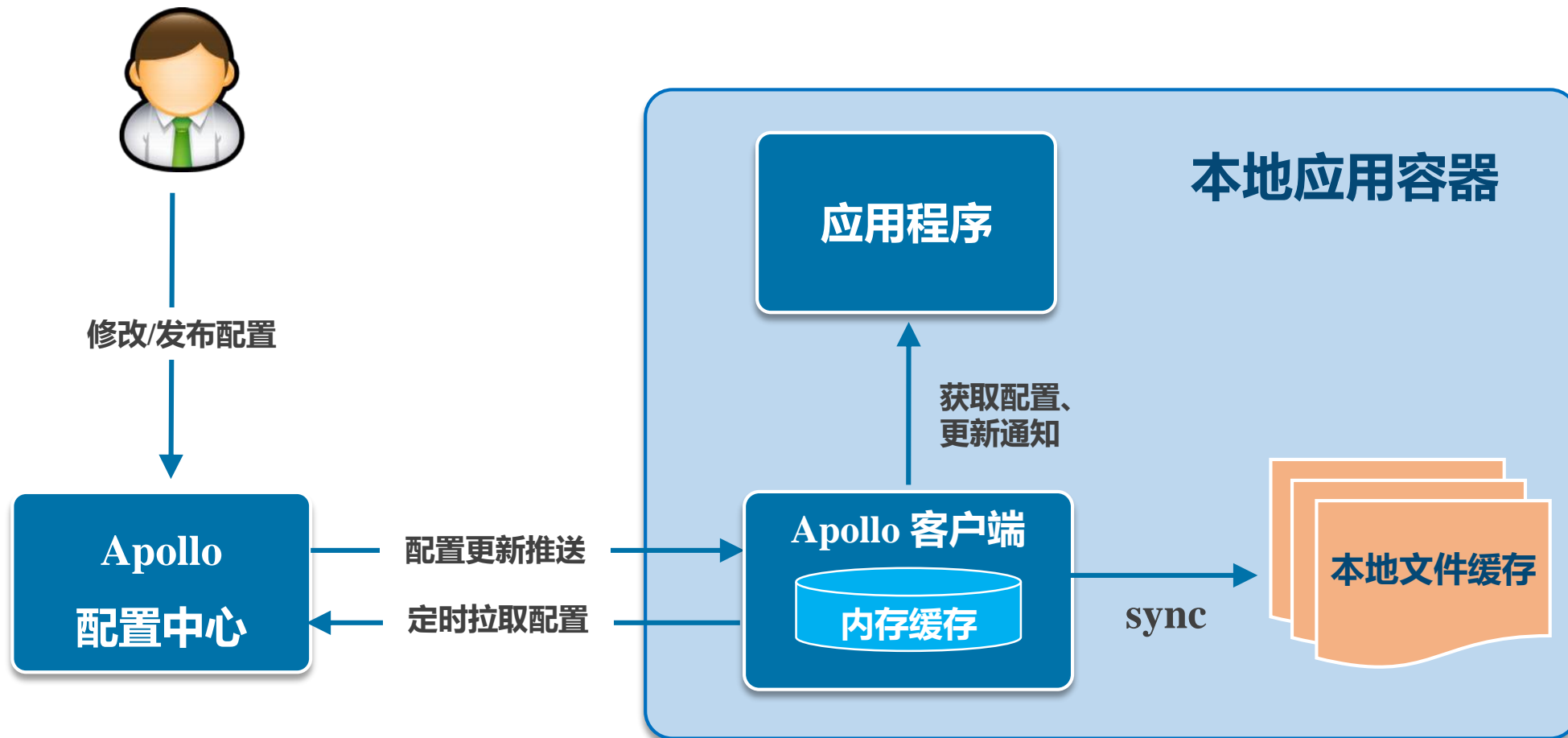
第

9

部分

Apollo架构设计之客户端

客户端架构



客户端实现总结

推拉结合

- 保持一个长连接，配置实时推送
- 定期拉配置(fallback)

配置缓存在内存

- 本地再缓存一份

应用程序

- 通过Apollo客户端获取最新配置
- 订阅配置更新通知

第

10

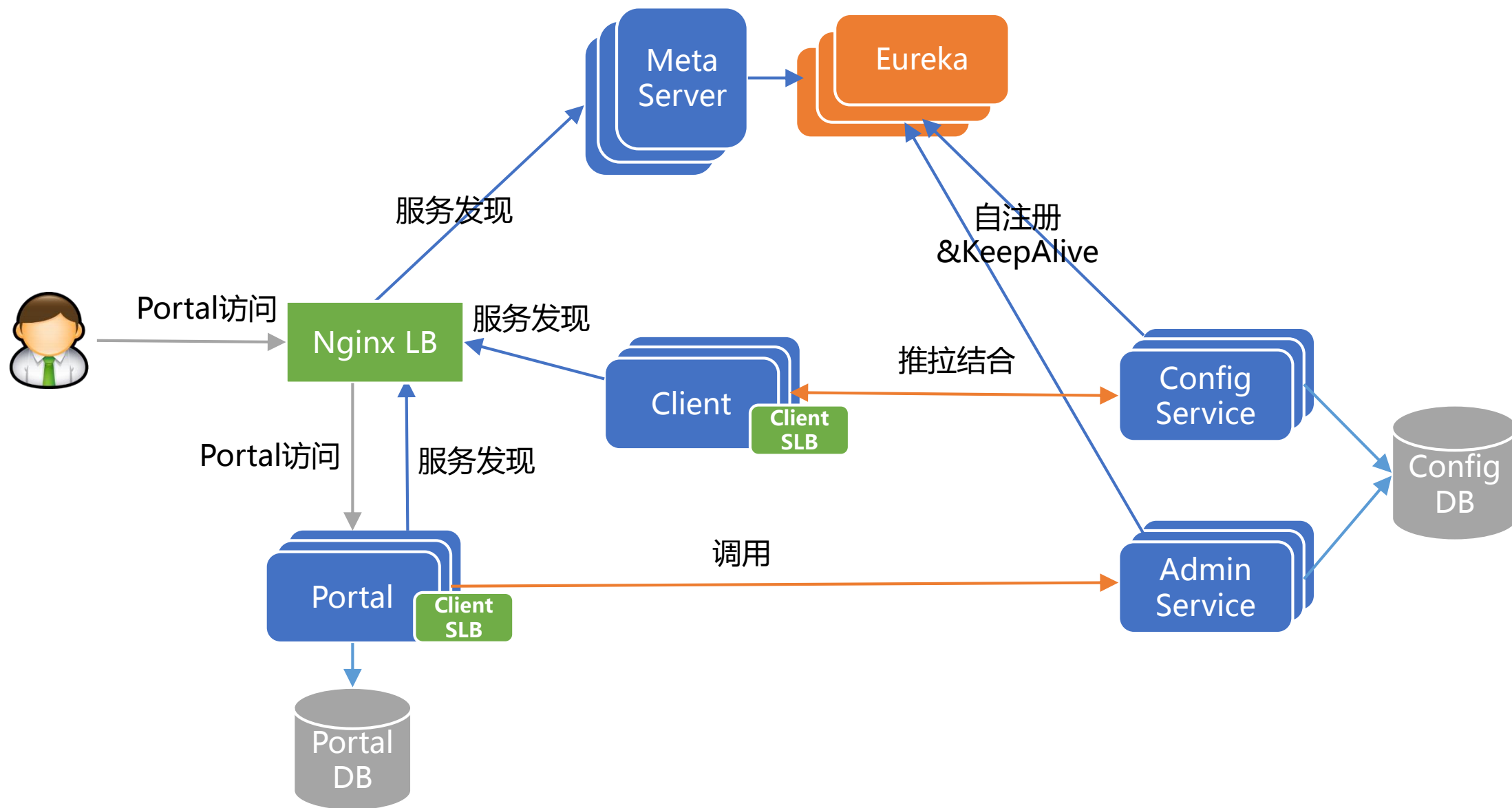
部分

Apollo架构设计之高可用和监控

Apollo HA高可用设计

场景	影响	降级	原因
某台ConfigService下线	无影响		ConfigService无状态部署，客户端重连重其它
所有ConfigService下线	客户端无法读取最新配置	客户端重启可获取本地缓存配置	
某台AdminService下线	无影响		AdminService无状态，Portal重连其它
所有AdminService下线	客户端无影响，Portal无法更新配置		
某台Portal下线	无影响		Portal无状态，通过域名重定向其它
全部Portal下线	客户端无影响，用户无法更新配置		
数据库宕机	客户端无影响，用户无法更新配置	ConfigService开启缓存后，对配置读取不受数据库宕机影响	

HA图例



Apollo监控

内置支持CAT

- <https://github.com/dianping/cat>
- 客户端+服务器端埋点
- 自动依赖扫描

定制扩展

- InfluxDB
- Prometheus



关键指标

- 接入应用数量
- 配置项数量
- 变更和发布数量
- 推送拉取次数
(success/failure)
- Config Service
 - 接口性能
 - GC
 - CPU

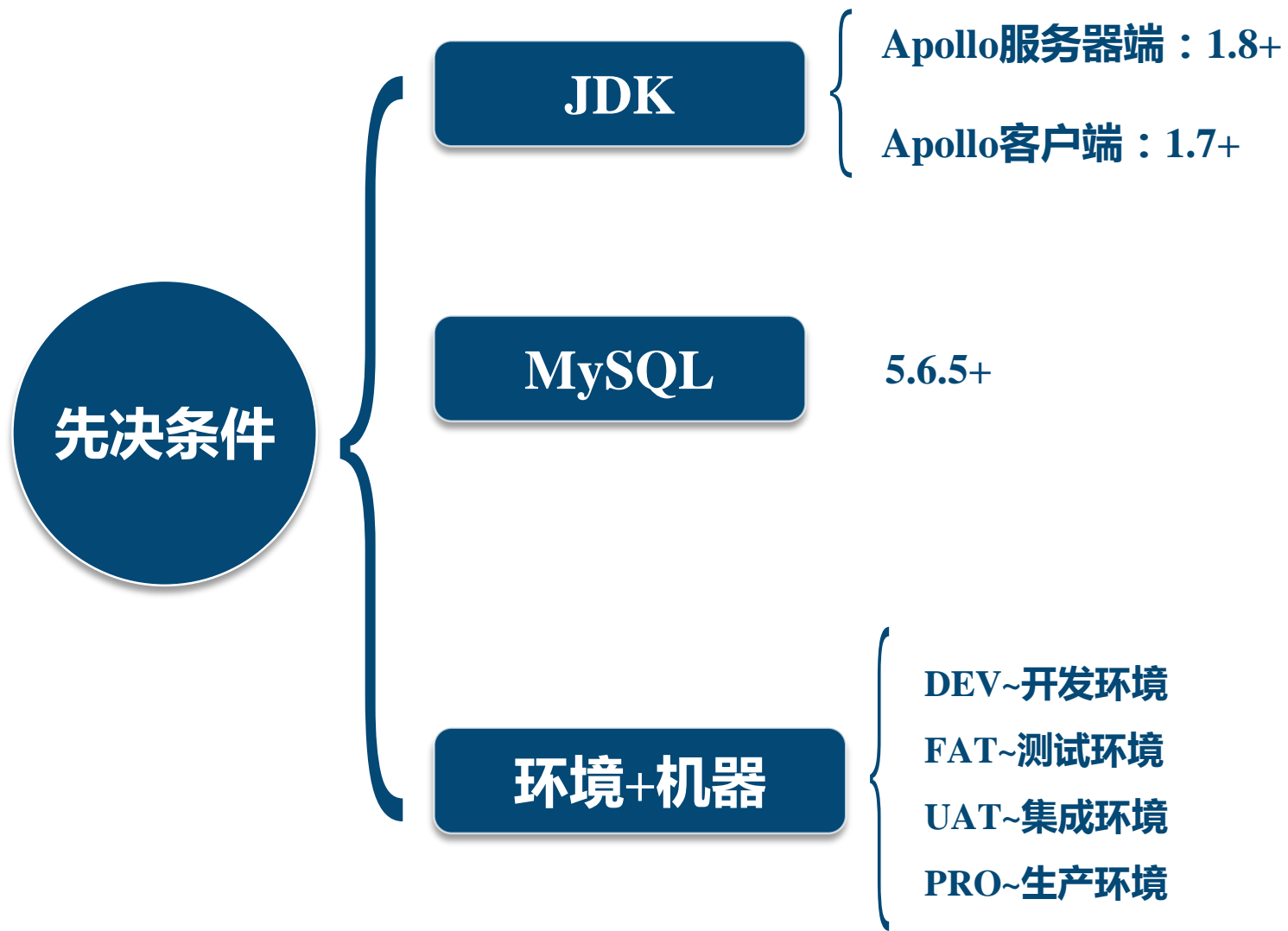
第

11

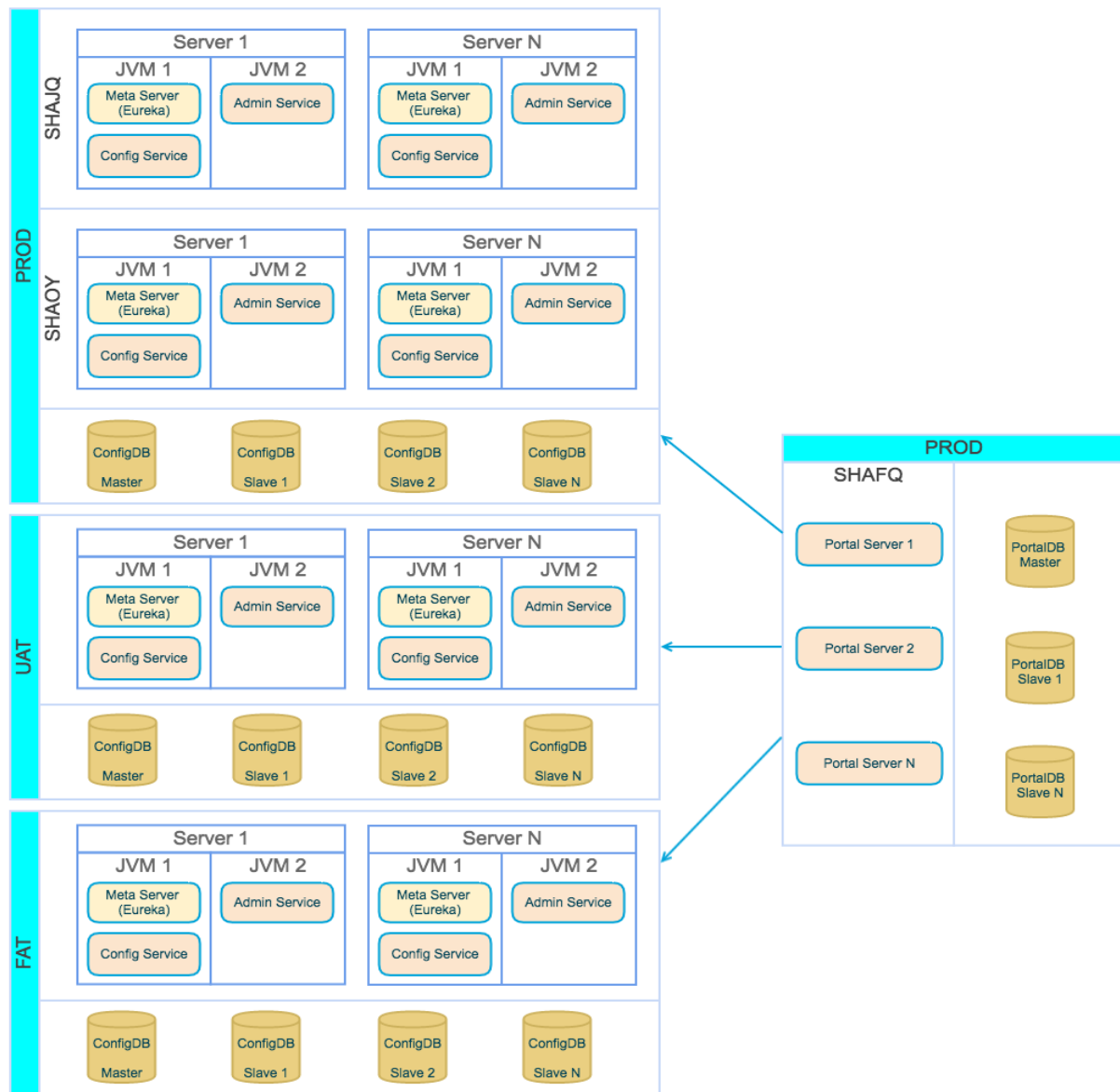
部分

Apollo分布式部署指南

先决条件

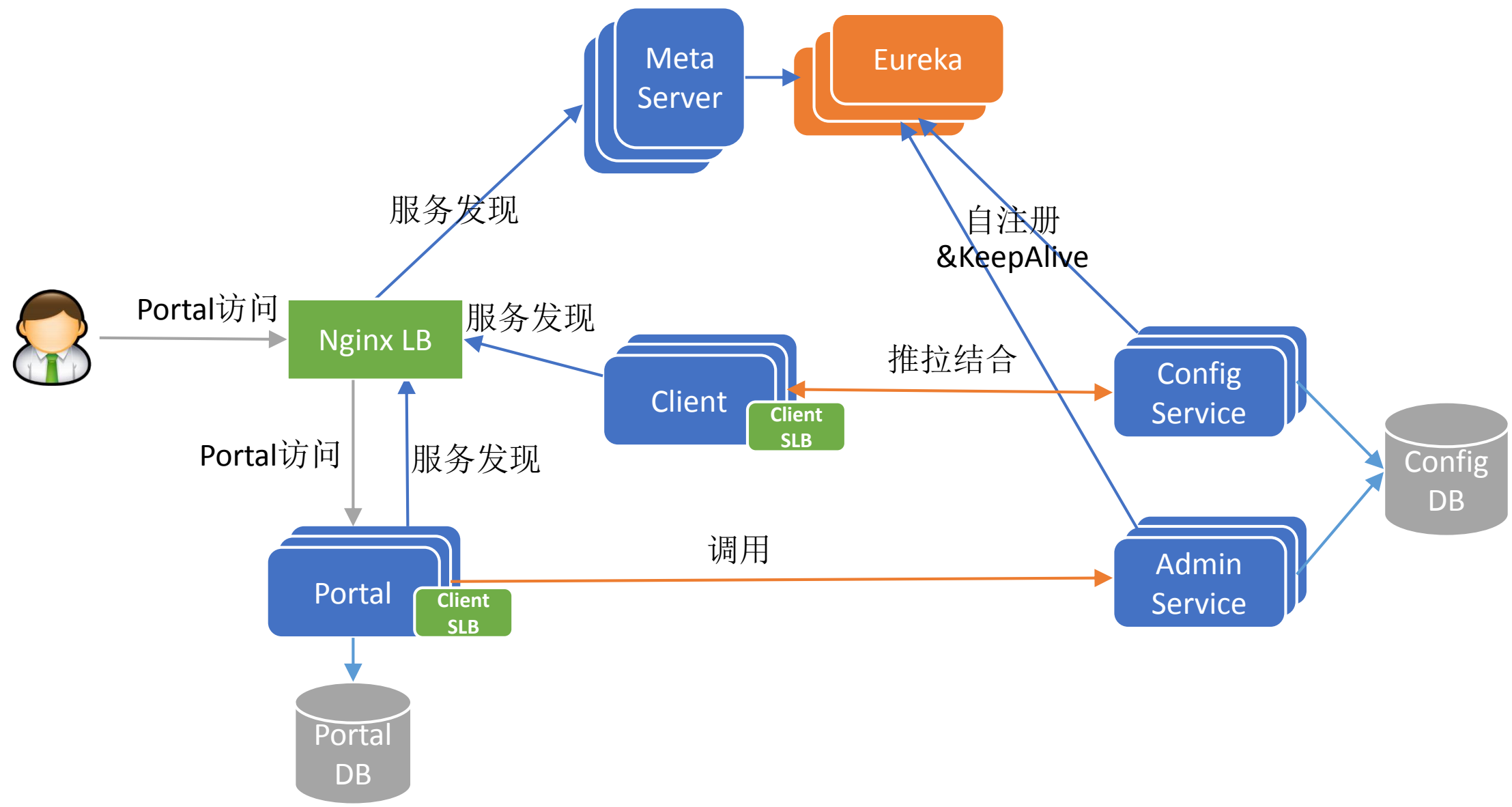


部署案例~ctrip



- Portal部署生产环境，管理 FAT/UAT/PRO
- MetaServer/ConfigService/Admin Service每个环境独立部署+独立DB
- 生产环境的 MetaServer/ConfigService/Admin Service双机房双活部署
- MetaServer和ConfigService住一个 JVM进程，AdminService住同机器另一个JVM进程

部署图例



关键配置和注意点

创建数据库

- ApolloPortalDB
- ApolloConfigDB

ApolloPortalDB调整

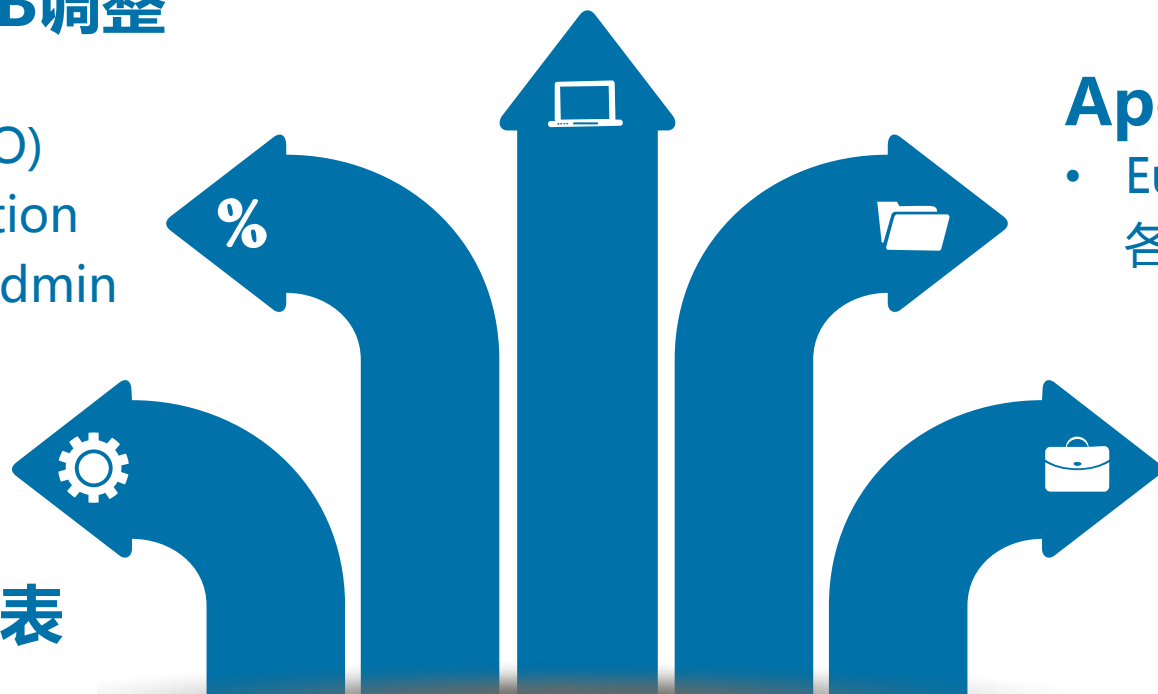
- 支持环境列表
(DEV,FAT,UAT,PRO)
- 部门列表organization
- 超级管理员superAdmin

ApolloConfigDB

- Eureka服务Url列表，
各环境不同

Portal打包各环境
MetaServer域名列表

注意多网卡问题



第

12

部分

Apollo Java客户端和多语言接入

环境要求

Java 1.7+

- Guava: 15.0+
- Apollo客户端默认会引用Guava 19，如果你的项目引入了其它版本，请确保版本号 ≥ 15.0

AppId

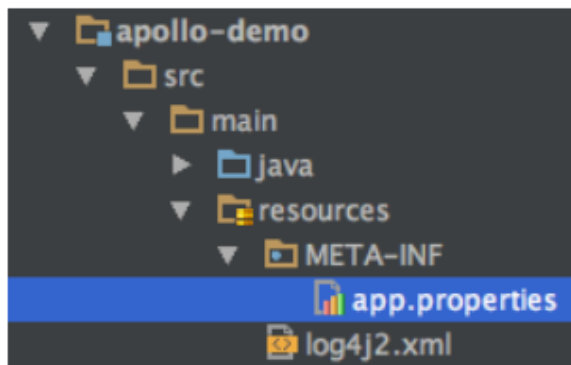
AppId是应用的身份信息，是从服务端获取配置的一个重要信息。

请确保classpath:/META-INF/app.properties文件存在，并且其中内容形如：

```
app.id=YOUR-APP-ID
```

推荐

文件位置参考如下：



v0.7.0版本后，Apollo也支持通过System Property传入app.id信息，如

```
-Dapp.id=YOUR-APP-ID
```

注：app.id是用来标识应用身份的唯一id，格式为string。

配置中心地址

客户端读取MetaServer地址方式

- **启动参数-Ddev_meta=http://someIp:8080**
- **apollo-core.jar中的apollo-env.properties (推荐)**
- **classpath中单独一份app-env.properties**
 - local.meta=http://localhost:8080
 - dev.meta=http://dev-apconfig.spring2go.com
 - fat.meta=http://fat-apconfig.spring2go.com
 - uat.meta=http://uat-apconfig.spring2go.com
 - pro.meta=http://apconfig.spring2go.com

运行环境设置Environment

客户端所在运行环境Env

- 启动参数-Denv=YOUR-ENV , 注意key小写
- OS环境变量ENV , 注意key大写
- 配置文件 (推荐)
 - Mac/Linux , 文件位置为opt/settings/server.properties
 - Windows , 文件位置为C:\opt\settings\server.properties
 - 格式env=DEV
 - 支持DEV/FAT/UAT/PRO
 - 本地开发模式env=Local

可选集群Cluster

一个环境中的一个app，对不同的集群可以有不同的配置

- 启动参数-Dapollo.cluster=app_cluster_v1, key全小写
- 通过配置文件（推荐）
 - Mac/Linux，文件位置为opt/settings/server.properties
 - Windows，文件位置为C:\opt\settings\server.properties
 - 可设置数据中心集群idc=xyz，注意key全小写

本地缓存路径

本地容灾降级，本地调试

- **Mac/Linux:** /opt/data/{appId}/config-cache
- **Windows:** C:\opt\data\{appId}\config-cache
- **注意应用需要有读写权限**
- **文件名** {appId}-{cluster}-{namespace}.properties

Apollo Client Jar依赖

参考[\[部署指南\]](#)打包到自己公司maven私服

- `<dependency>`
- `<groupId>com.ctrip.framework.apollo</groupId>`
- `<artifactId>apollo-client</artifactId>`
- `<version>0.9.1</version>`
- `</dependency>`

其它语言接入

01

[[.Net客户端使用指南](#)]原生支持

02

[[Go、Python、NodeJS](#)]第三方客户端

03

[[Apollo Http接口](#)]其它语言接入

第

13

部分

Apollo Client API实操 (Lab2)

第 14 部分

Apollo Client和Spring集成~XML方式 (Lab3)

第 15 部分

Apollo Client和Spring集成~代码方式 (Lab4)

第

16

部分

Apollo Client和Spring Boot集成 (Lab5)

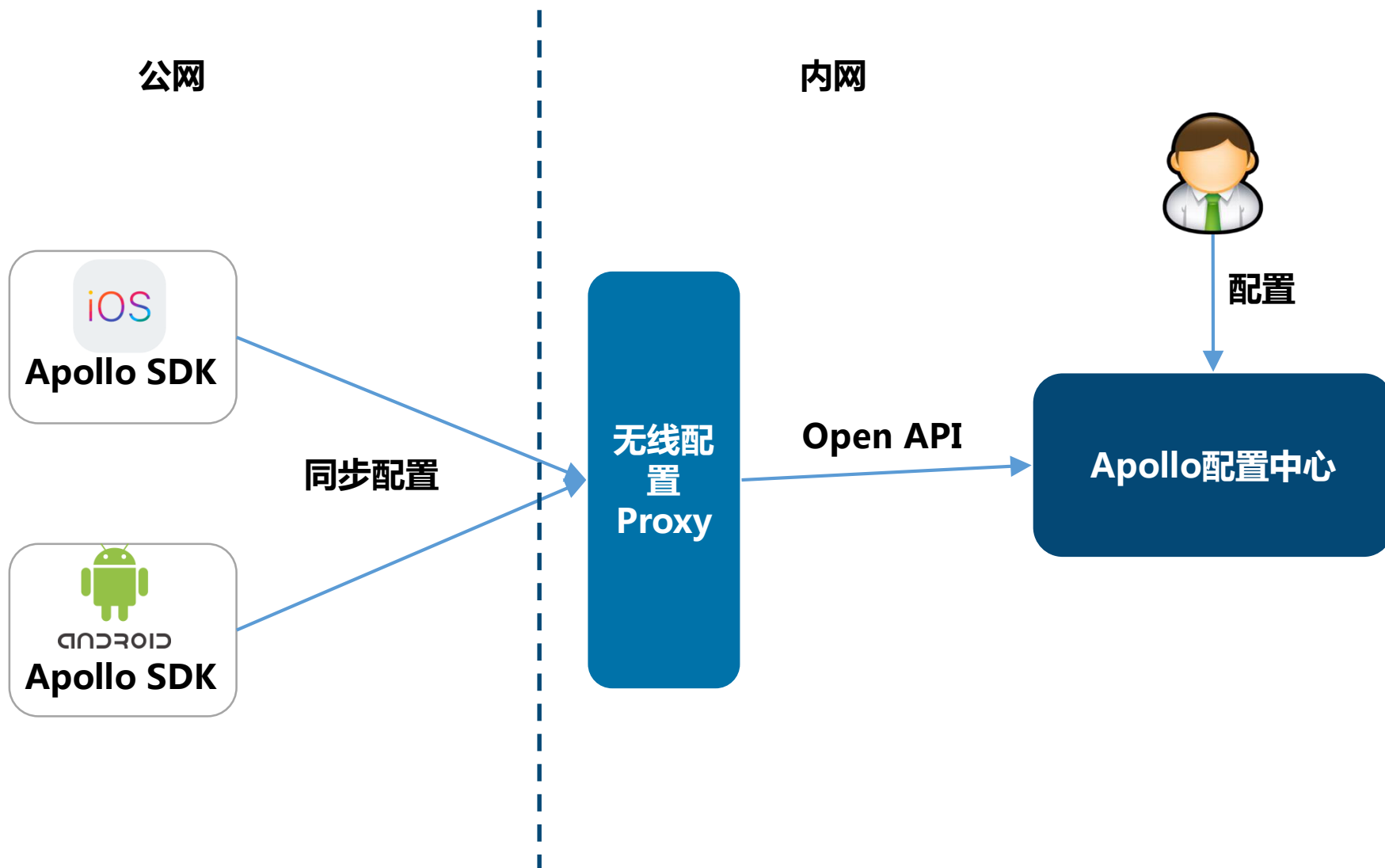
第

17

部分

Apollo开放平台接入实操（ Lab6 ）

场景~无线app接入apollo



第

18

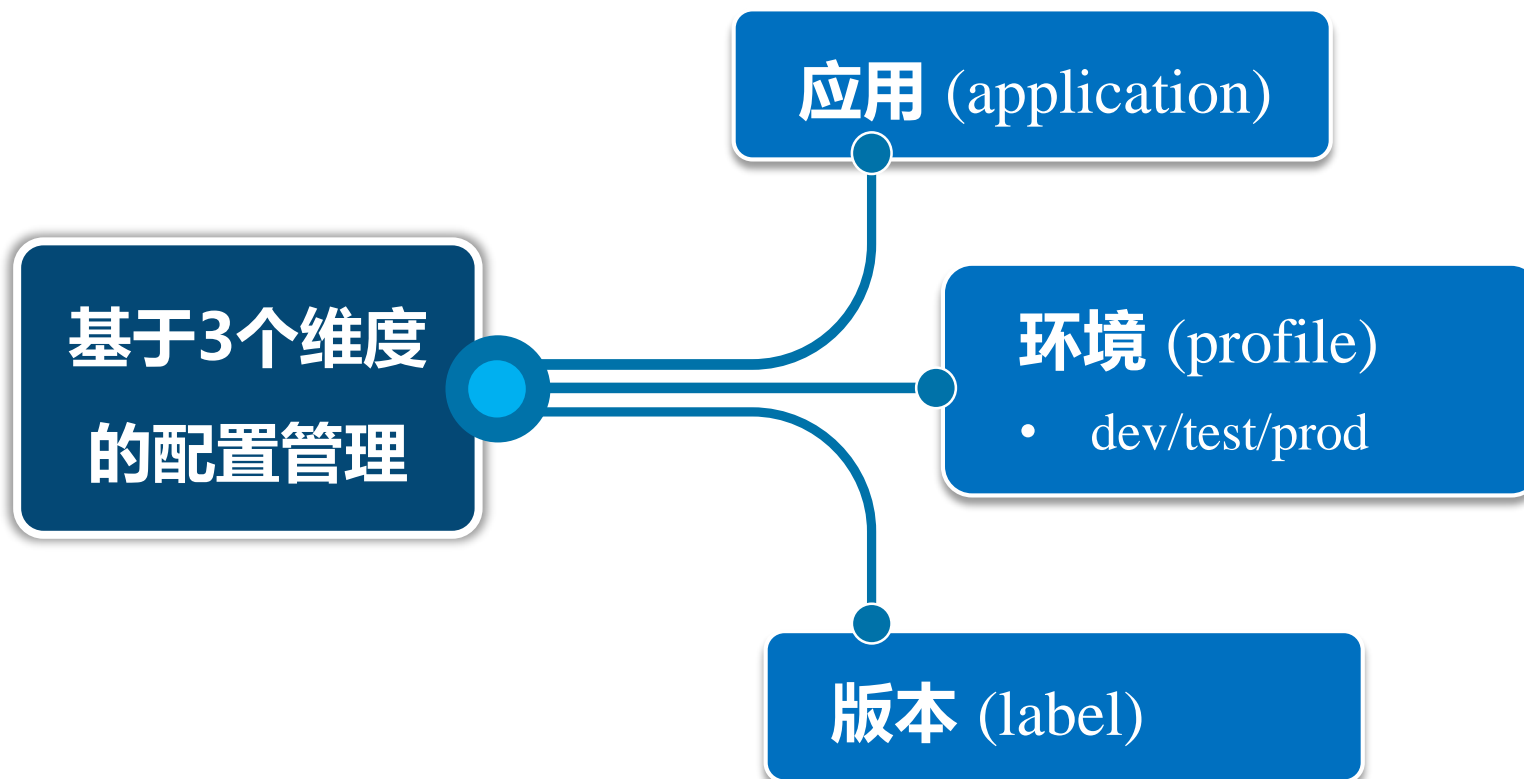
部分

Spring Cloud Config简介

Spring Cloud Config简介1



Spring Cloud技术栈自带的配置中心



Spring Cloud Config简介2

优势

配置存储支持Git

和Spring无缝集成

设计简单轻量

不足

动态配置能力弱

治理能力弱

不算严格企业级

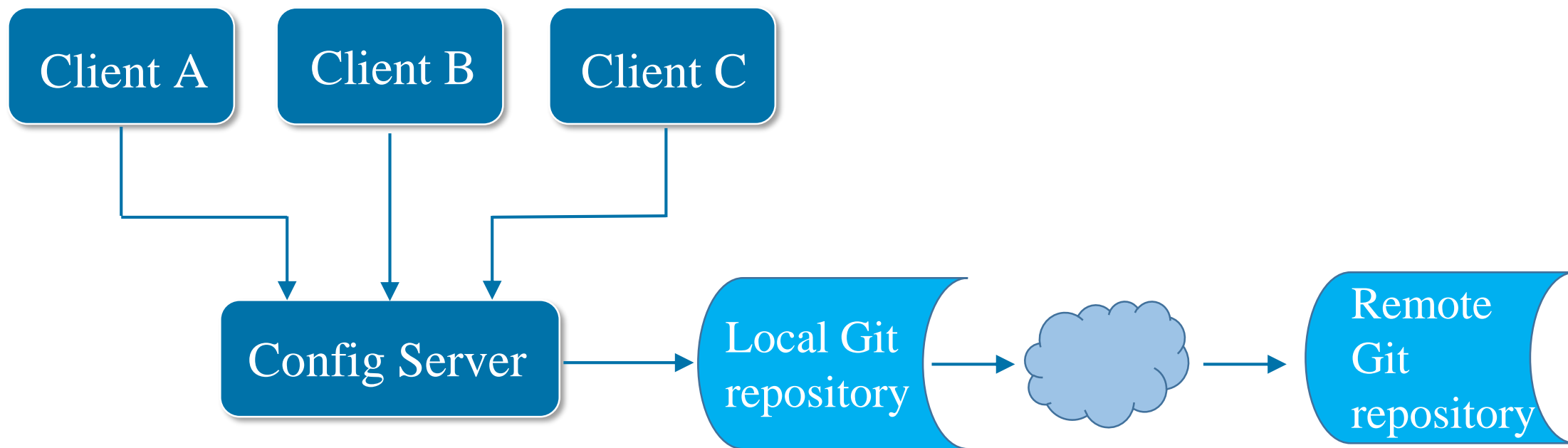
Config Server接口样例

接口URL:/{application}/{profile}/{label}

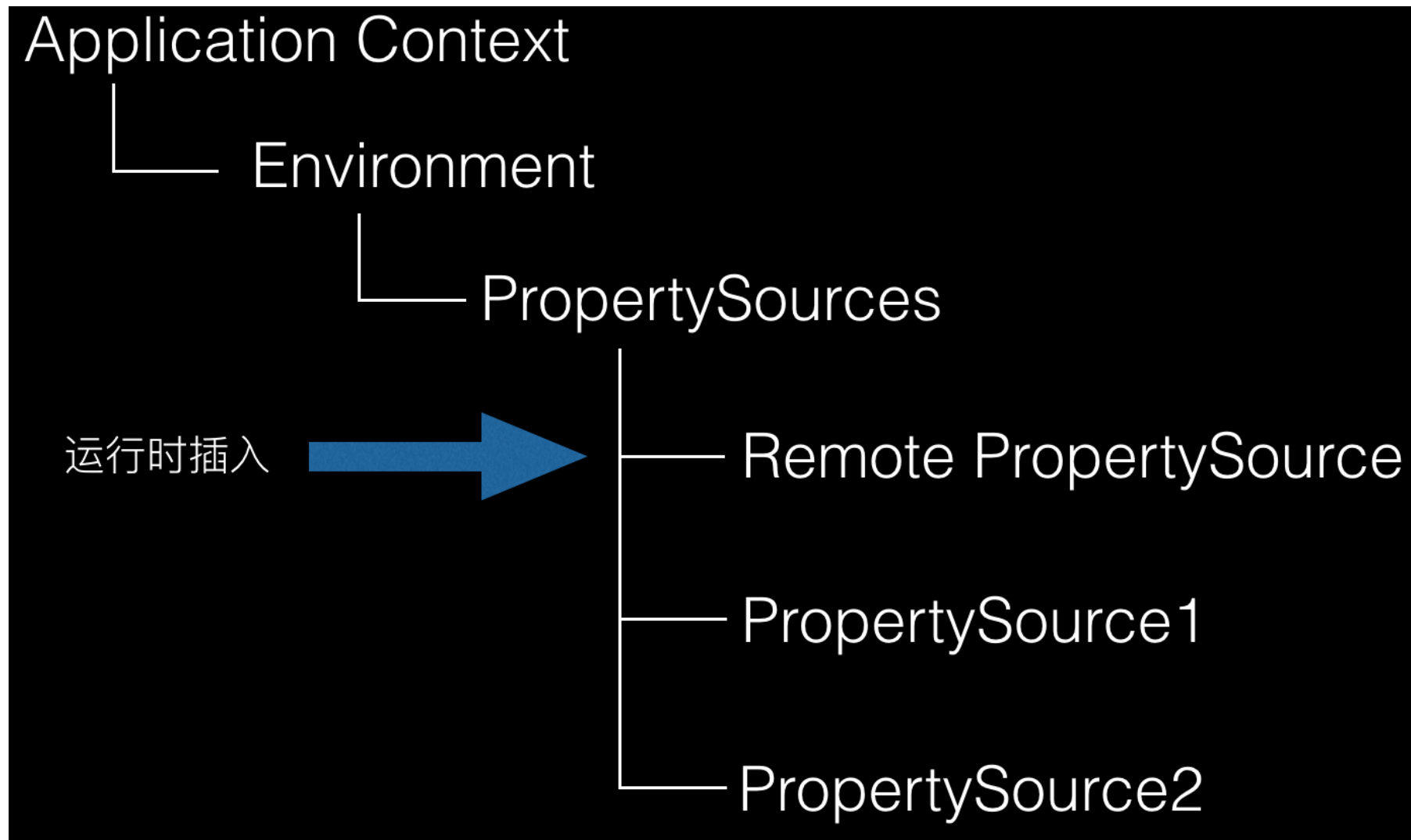
http://localhost:9000/appfoo/production/master

```
{
  "name": "appfoo",
  "profiles": ["production"],
  "label": "master",
  "version": "75cac1f5514358c4e3302c7ae07cd12db3dkkkb6",
  "propertySources": [{
    "name": "git@github.com:archcentric/config-repo.git/appfoo-production.properties",
    "source": {
      "foo": "Hello World "
    }
  }]
}
```

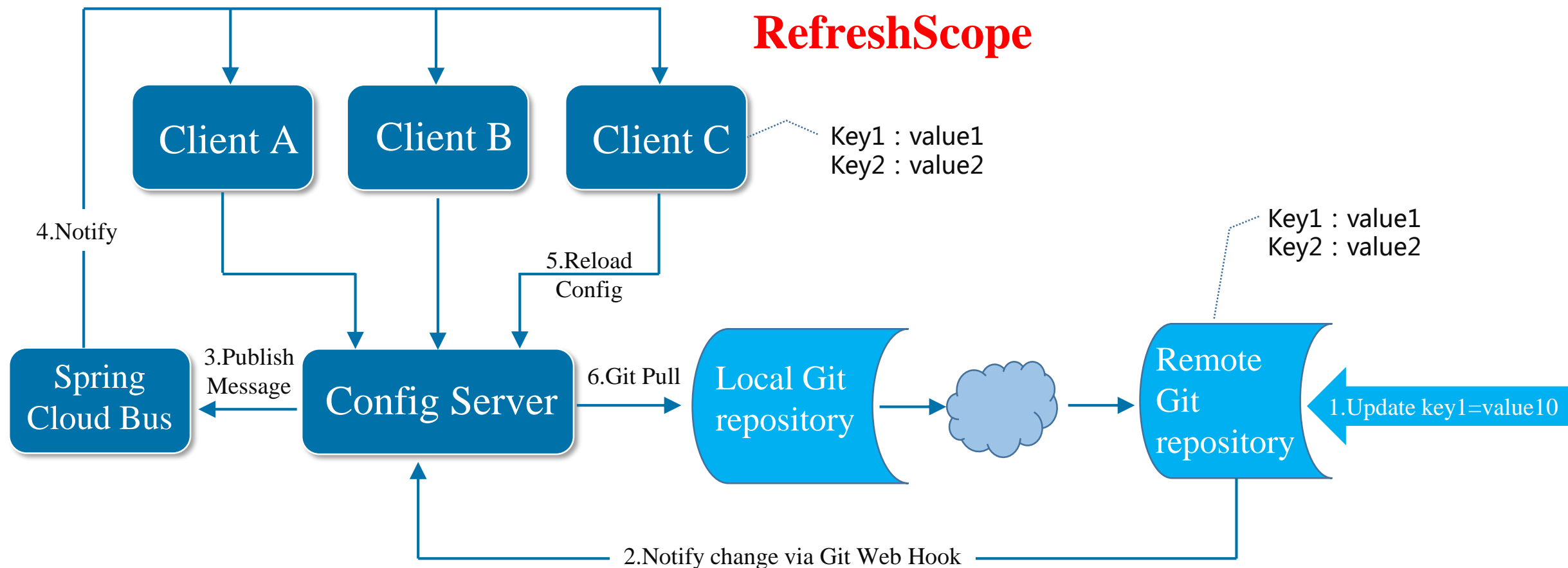
简化架构



Config Client实现细节



动态配置实现



第

19

部分

Apollo vs Spring Cloud Config

Apollo vs Spring Cloud Config

功能点	Apollo	Spring Cloud Config
配置界面	统一界面管理不同环境/集群配置	无，通过git操作
配置生效时间	实时	重启生效，或者Refresh，或git hook + MQ扩展
版本管理	界面上直接提供发布历史和回滚按钮	无，通过git操作
灰度发布	支持	不支持
授权/审计/审核	界面上直接操作，且支持修改和发布权限分离	需要通过git仓库设置，且不支持修改和发布权限分离
实例配置监控	可以方便看到当前哪些客户端在使用哪些配置	不支持
配置获取性能	快，通过数据库访问+缓存支持	较慢，需从git clone repo，然后本地文件读取
客户端支持	原生支持Java/.Net，提供API，支持Spring annotation	Spring应用+annotation支持

结论：Apollo是企业生产级配置中心，适用范围更广

第 20 部分

Apollo FAQ和开发常见问题

常见FAQ



Cluster是什么？

一个应用不同实例的分组，比如典型的多机房部署按数据中心分集群。



Namespace是什么？

一个应用下不同配置的分组，例如应用配置application，中间件配置framework，数据库配置database

常见FAQ

Q

多个应用想使用同一份配置，如何做到？

使用公有Namespace

Q

客户端访问配置是否有权限，是否支持配置加密？

Apollo Client获取配置没有做权限管控，配置加密需要应用层自己实现

应用开发常见问题

Q

本地开发没有问题，为什么线上会出问题

首先检查服务器上对应的data文件夹读写权限，此权限对应的是程序运行时所对应的角色；环境是否配对

为什么不能进行编辑发布操作

Q

编辑和发布权限需要管理员授权，即使是管理员自己也需要自己给自己授权

应用开发常见问题

 接入了apollo值不更新怎么办？

1. 通过Config每次获取最新值，
2. 监听配置变更事件，
3. Spring RefreshScope

一个应用可以创建多个版本的配置吗？ 

使用集群Cluster

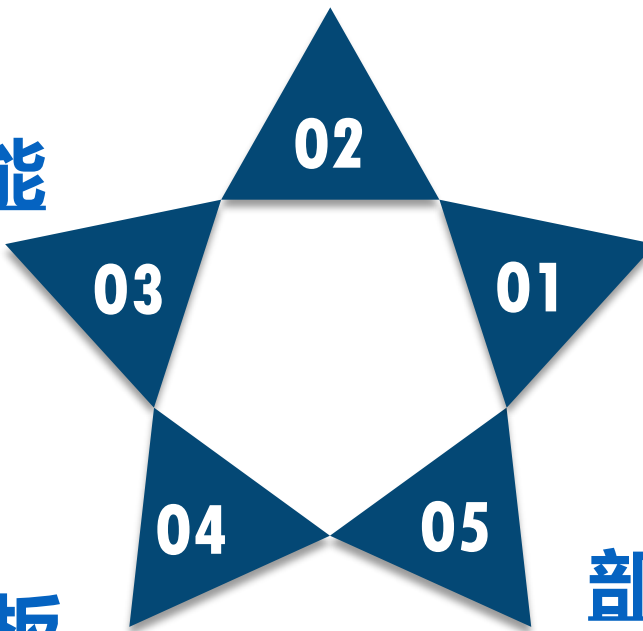
高级功能和定制参考

Apollo开发指南

Portal实现用户登录功能

- Spring Security
- 统一登录SSO/LDAP

邮件通知模板



灰度发布使用指南

部署常见问题

- 多网卡ip问题
- 独立Eureka集群
- Portal如何增加环境
- 如何删除应用、集群和Namespace

第

21

部分

参考资源和后续课程预览

参考文章

微服务来了，配置怎么办？

<http://p.primeton.com/articles/59f2c45e4be8e639a5002b84>

Feature Flag Driven Development

<https://blog.launchdarkly.com/feature-flag-driven-development/>

Feature flagging to mitigate risk in database migration

<https://blog.launchdarkly.com/feature-flagging-to-mitigate-risk-in-database-migration/>

Trunk based Development

<https://www.continuousdeliveryconsulting.com/blog/organisation-pattern-trunk-based-development/>

参考ppt

百倍速交付~谈Trunk-Based Development

<https://www.slideshare.net/bryan0817/trunkbased-development>

携程Apollo参考



Github站点

<https://github.com/ctripcorp/apollo>

Apollo源码解析(宇道源码)

<http://www.iocoder.cn/categories/Apollo/>

Spring Cloud Config

<http://nobodyiam.com/2016/04/02/dive-into-spring-cloud-config/>

**Dive into
Spring Cloud
Config**

**Spring
Cloud
Config官网**

<https://cloud.spring.io/spring-cloud-config/>

其它开源配置中心产品

<https://github.com/knightliao/disconf>

百度Disconf

Qihoo360 QConf

<https://github.com/Qihoo360/QConf>

<https://github.com/Netflix/archaius>

可以扩展对接Apollo

Netflix Archaius (客户端)

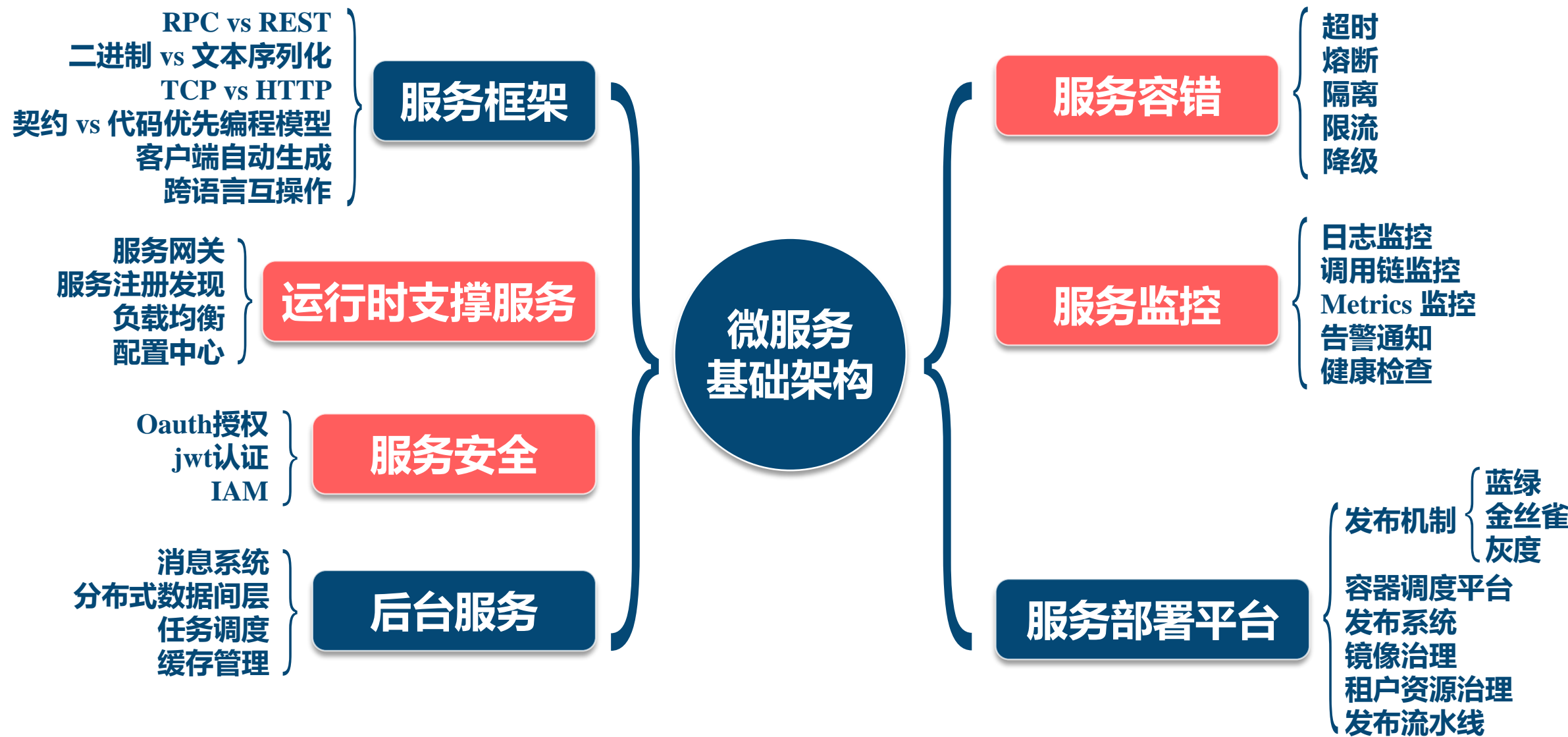
SaaS服务



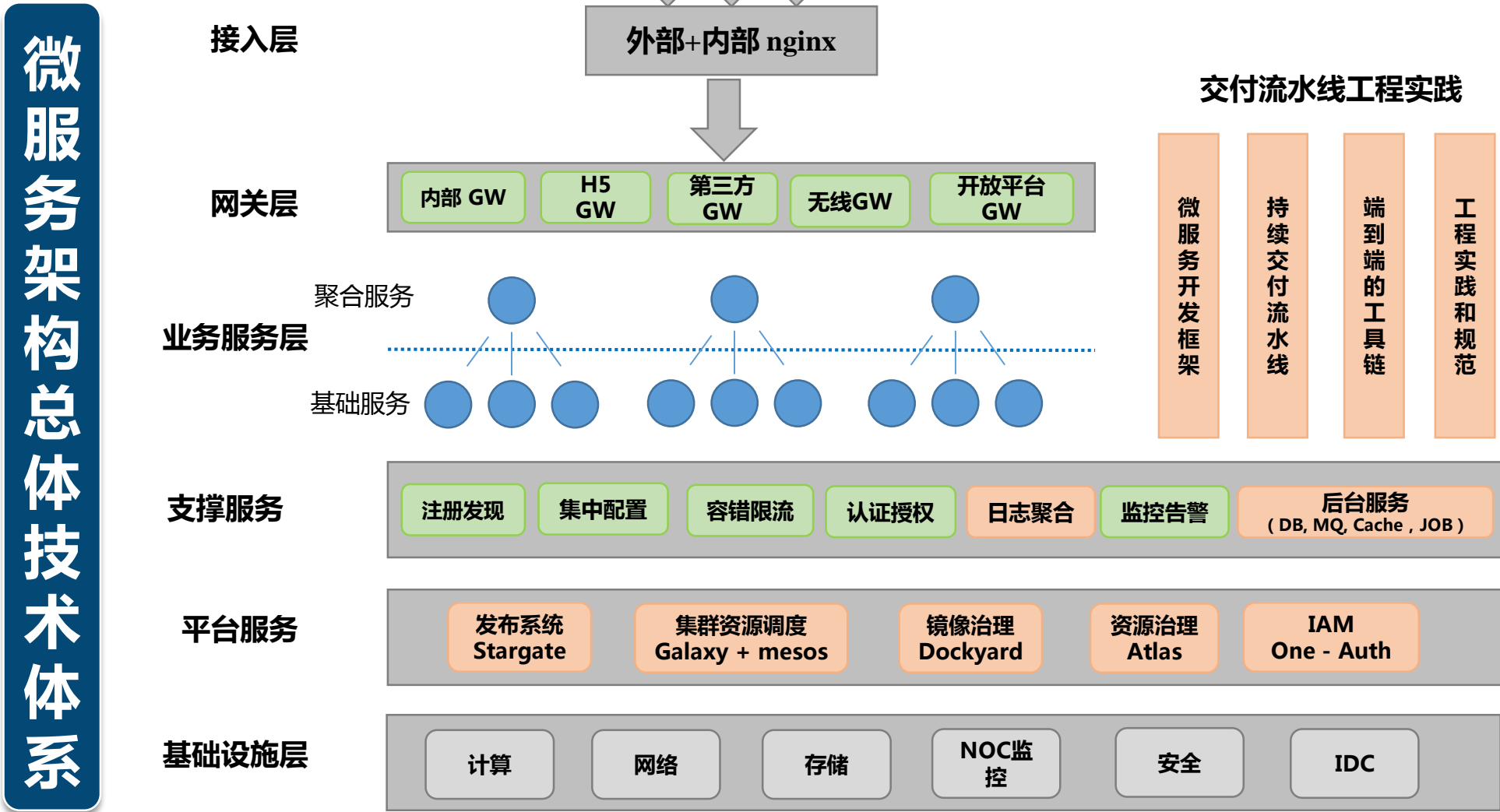
<https://launchdarkly.com/>

<https://featureflags.io/>

后续课程预览~2018课程模块



后续课程预览~技术体系



架构和技术栈预览

