# 配置中心，让微服务更『智能』

宋顺

携程框架架构研发部技术专家

携程框架架构研发部技术专家

2016年初加入携程，负责中间件产品的研发工作

开源配置中心Apollo主要作者

https://github.com/ctripcorp/apollo

毕业于复旦大学软件工程系，曾就职于大众点评，担任后台系统技术负责人

TABLE OF

# CONTENTS 大纲

1. 为什么需要配置中心?
2. 配置中心的一般模样
3. 如何让微服务更『智能』?
4. 配置中心的最佳实践

TABLE OF

# CONTENTS 大纲


1. 为什么需要配置中心?
2. 配置中心的一般模样
3. 如何让微服务更『智能』?
4. 配置中心的最佳实践

# 配置即『控制』

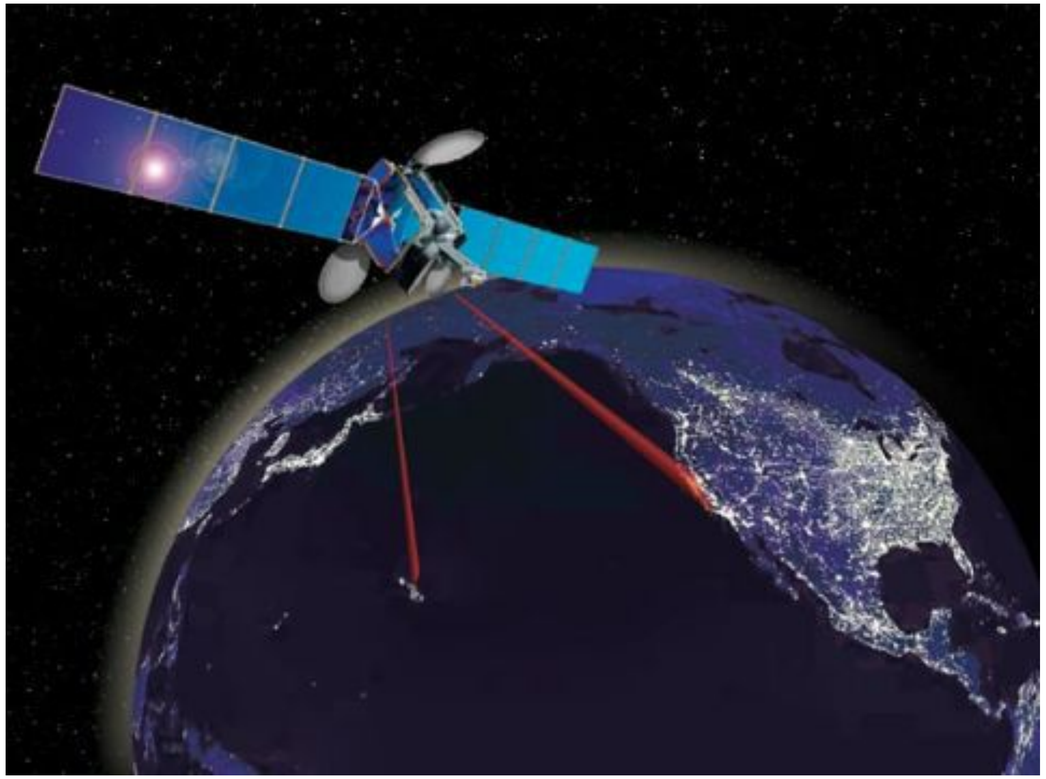
程序的发布其实和卫星的发射有一些相似之处

## 卫星发射升天后

- 处于自主驾驶状态，按照预设的轨道运行
- 间歇可收到地面的『控制』信号对运行姿态进行调整

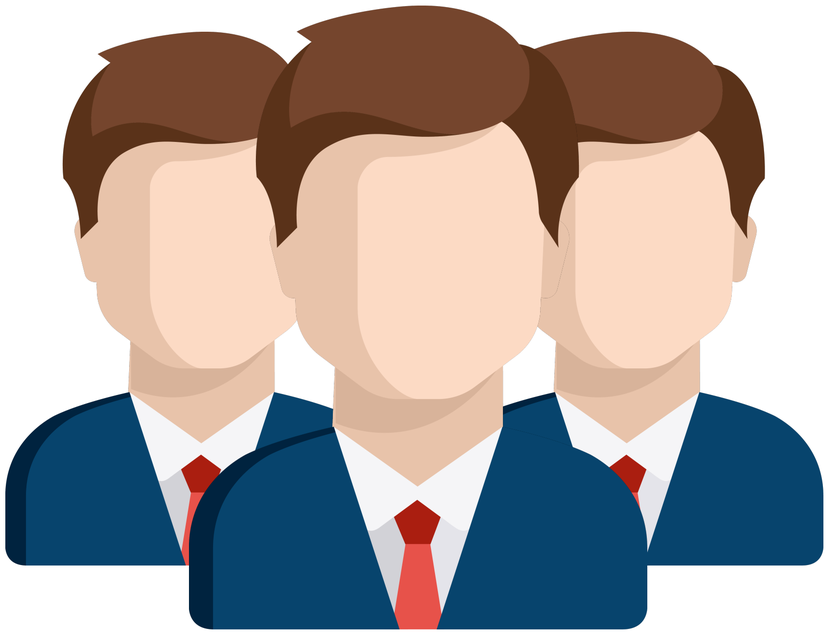
## 程序发布到生产环境后

- 按照预设的逻辑运行
- 通过调整配置参数来动态调整程序的行为
- 这些配置参数就代表着我们对程序的『控制』信号



图片来源：https://www.popularmechanics.com/space/a7194/how-it-works-nasas-experimental-laser-communication-system/

# 配置需要治理



- 权限控制、审计日志
- 灰度发布、配置回滚
- 不同环境、集群管理

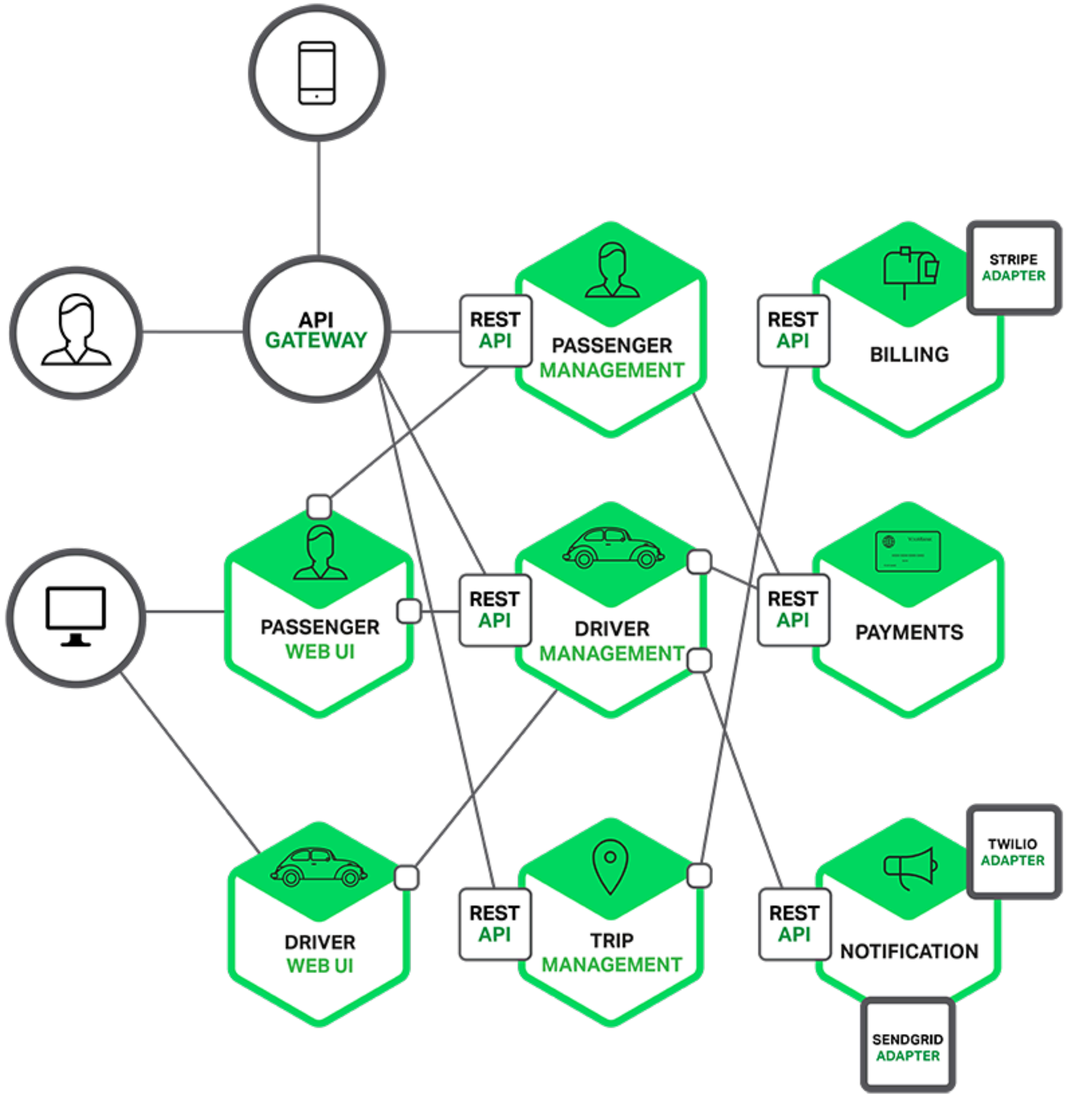
# 微服务的复杂性

## 单体应用时代

- 应用数量少
- 配置简单
- 运维可以登机器修改配置文件

## 微服务时代

- 应用数量多
- 配置数量也急剧增长
- 人工登机器修改不仅效率低，还容易出错




图片来源：https://www.nginx.com/blog/introduction-to-microservices/

需要一个统一的配置中心来管理微服务的配置！

TABLE OF

# CONTENTS 大纲

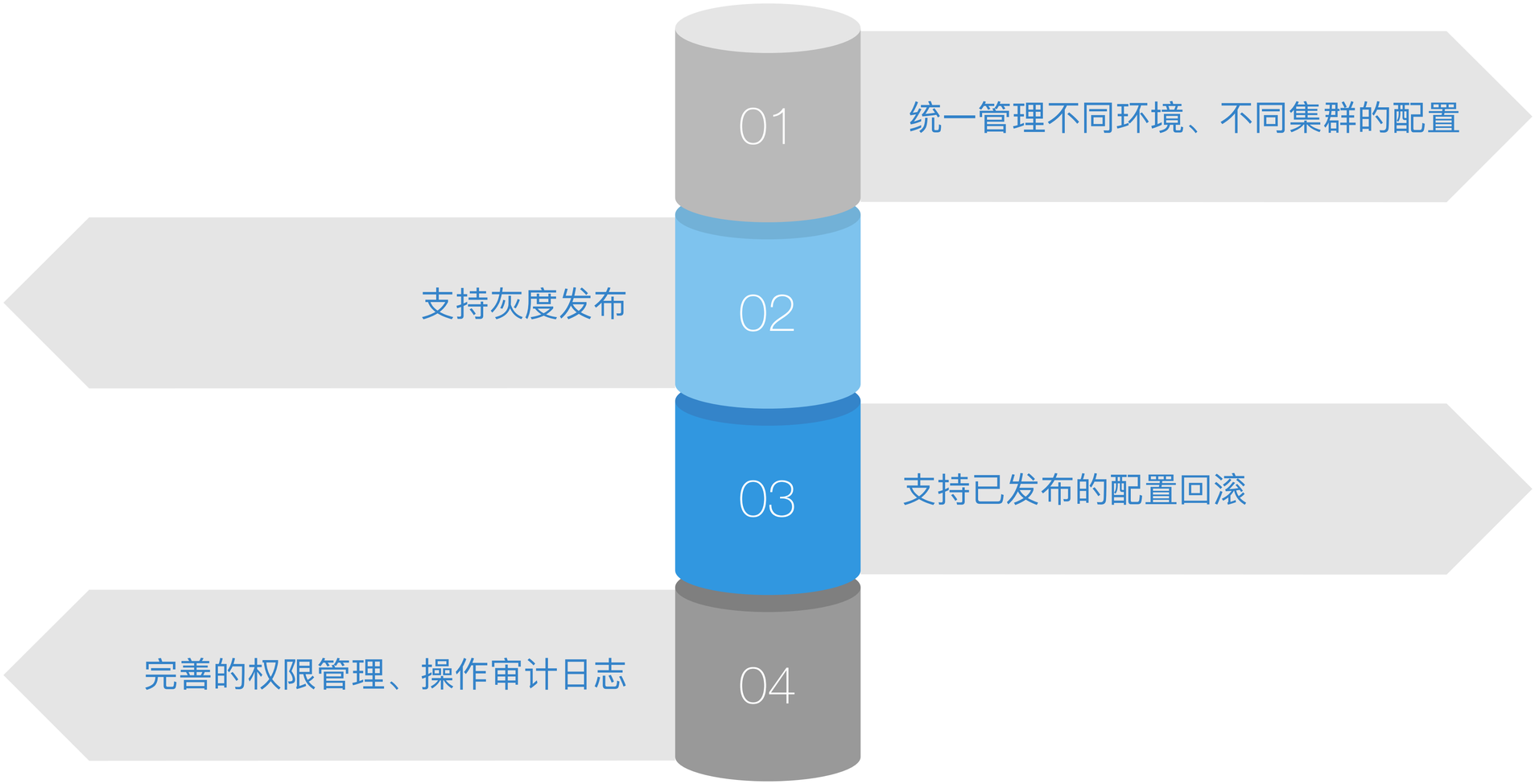

1. 为什么需要配置中心?
2. 配置中心的一般模样
3. 如何让微服务更『智能』?
4. 配置中心的最佳实践

TABLE OF

# CONTENTS 大纲

1. 为什么需要配置中心?
2. 配置中心的一般模样 (以开源配置中心Apollo为例)
3. 如何让微服务更『智能』?
4. 配置中心的最佳实践

# 治理能力


01
统一管理不同环境、不同集群的配置
02
支持灰度发布
03
支持已发布的配置回滚
04
完善的权限管理、操作审计日志

# Apollo 配置中心

搜索项目(AppId、项目名)　帮助　song_s

环境列表

- FAT
  - default 集群
  - dev 集群
- UAT
- PRO
  - default 集群
  - SHAOY 集群
  - SHAJQ 集群

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

表格　文本　更改历史　实例列表 1

过滤配置　同步配置　+新增配置

| 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 | |
| 已发布 | kibana.url | http://1.1.1.2:5600 | | song_s | 2016-11-25 20:57:27 | |
| 已发布 | elastic.document.type | biz1 | | song_s | 2017-01-11 19:14:06 | |
| 已发布 | elastic.cluster.name | es-cluster | | song_s | 2016-10-18 19:57:29 | |
| 已发布 | elastic.cluster | 2.2.2.2:9300,4.4.4.4:9300 | | zhanglea | 2016-12-08 14:19:43 | |
| 已发布 | page.size | 20 | | song_s | 2016-12-27 14:58:56 | |
| 已发布 | zookeeper.address | 10.1.12.2 | | song_s | 2016-10-19 11:33:50 | |

关联

FX.apollo properties

发布　回滚　发布历史　授权　灰度

表格　文本　更改历史　实例列表 1

同步配置

覆盖的配置　filter by key ...

| 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 | |

公共的配置 (AppId:100003173, Cluster:default)　filter by key ...

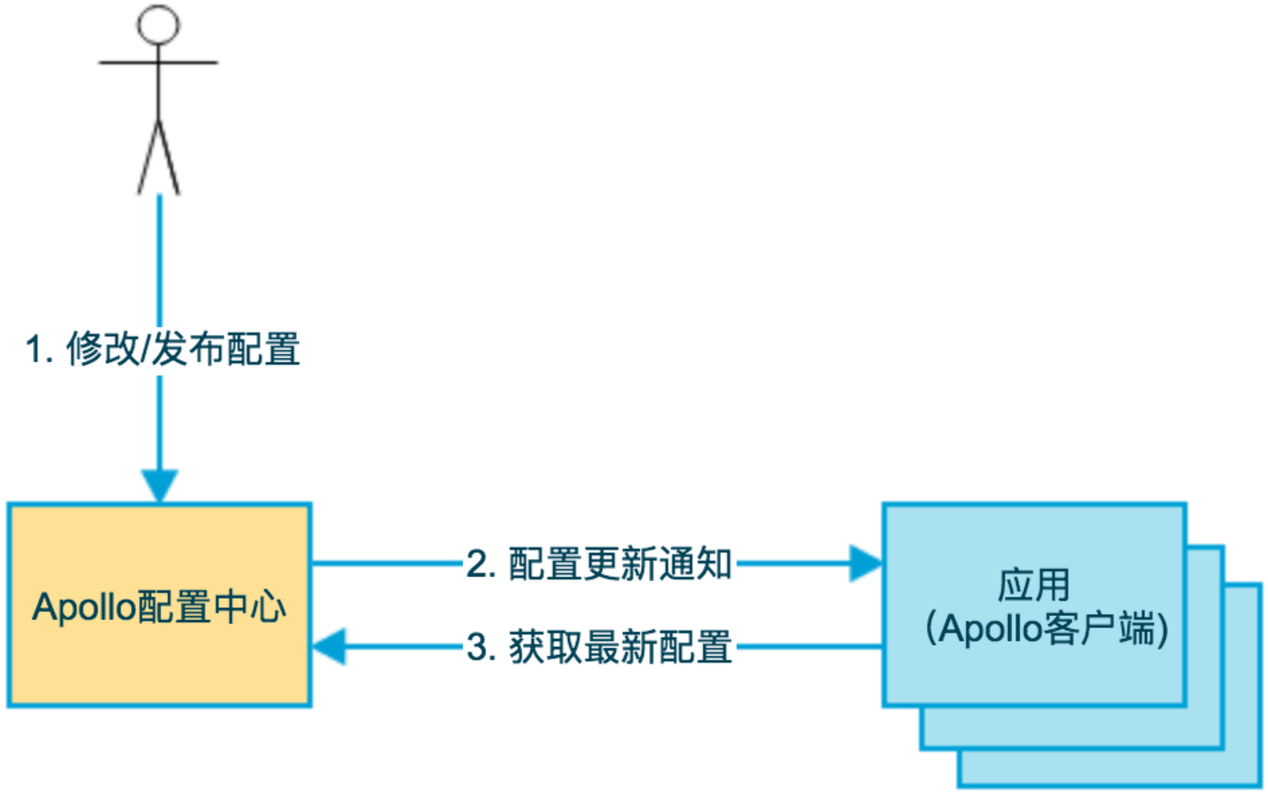
| Key | Value | 备注 | 最后修改人 | 最后修改时间 | 操作 |
|---|---|---|---|---|---|
| batch | 2000 | 样例项目会使用到，勿删。 | song_s | 2017-02-16 13:27:07 | |
| servers | 1.1.1.1,2.2.2.2 | 样例项目会使用到，勿删。 | song_s | 2016-10-12 14:03:34 | |

# 可用性

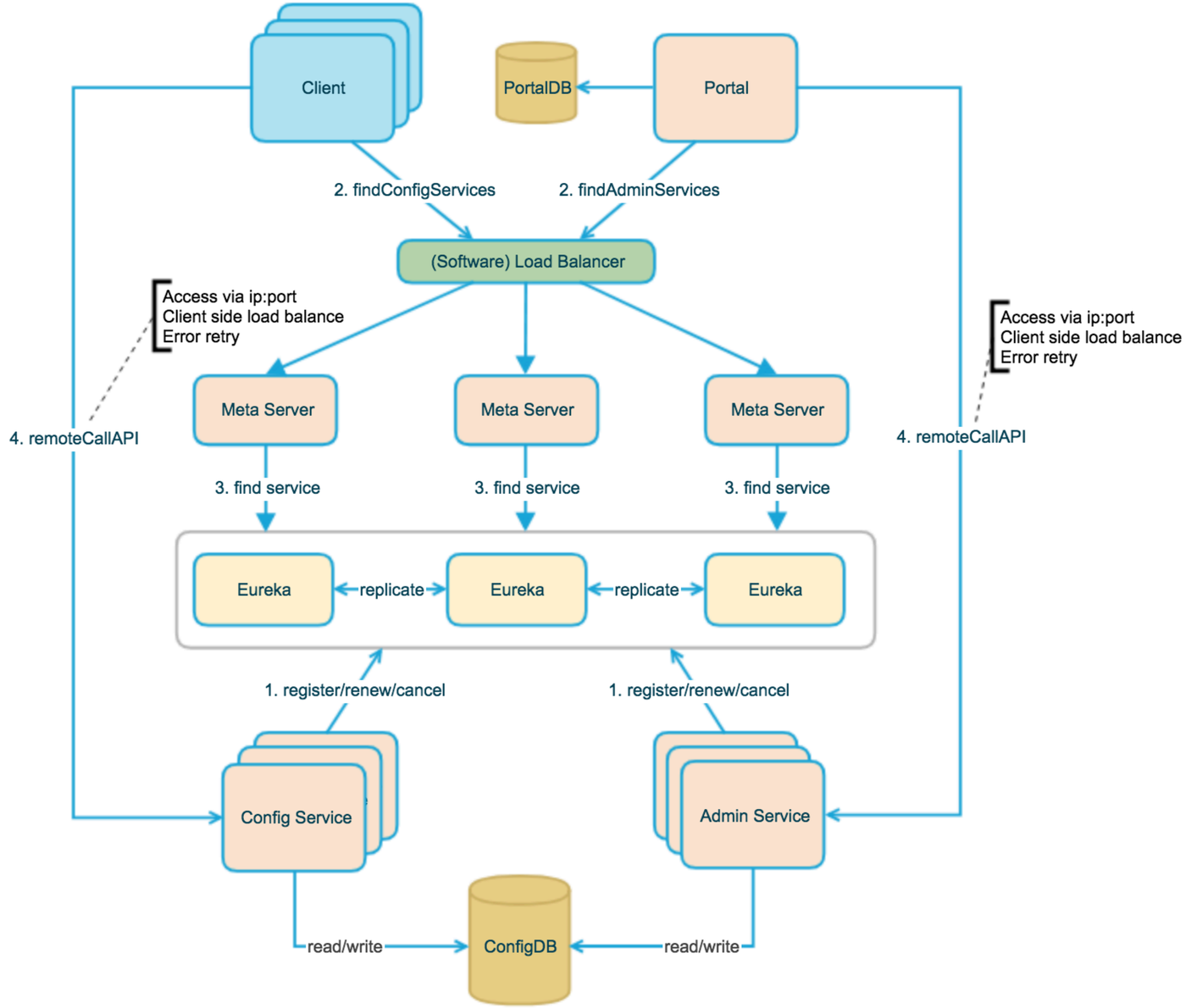
配置即『控制』

- 所以在一定程度上，配置中心已经成为了微服务的大脑
- 作为大脑，可用性要求显然是非常高的

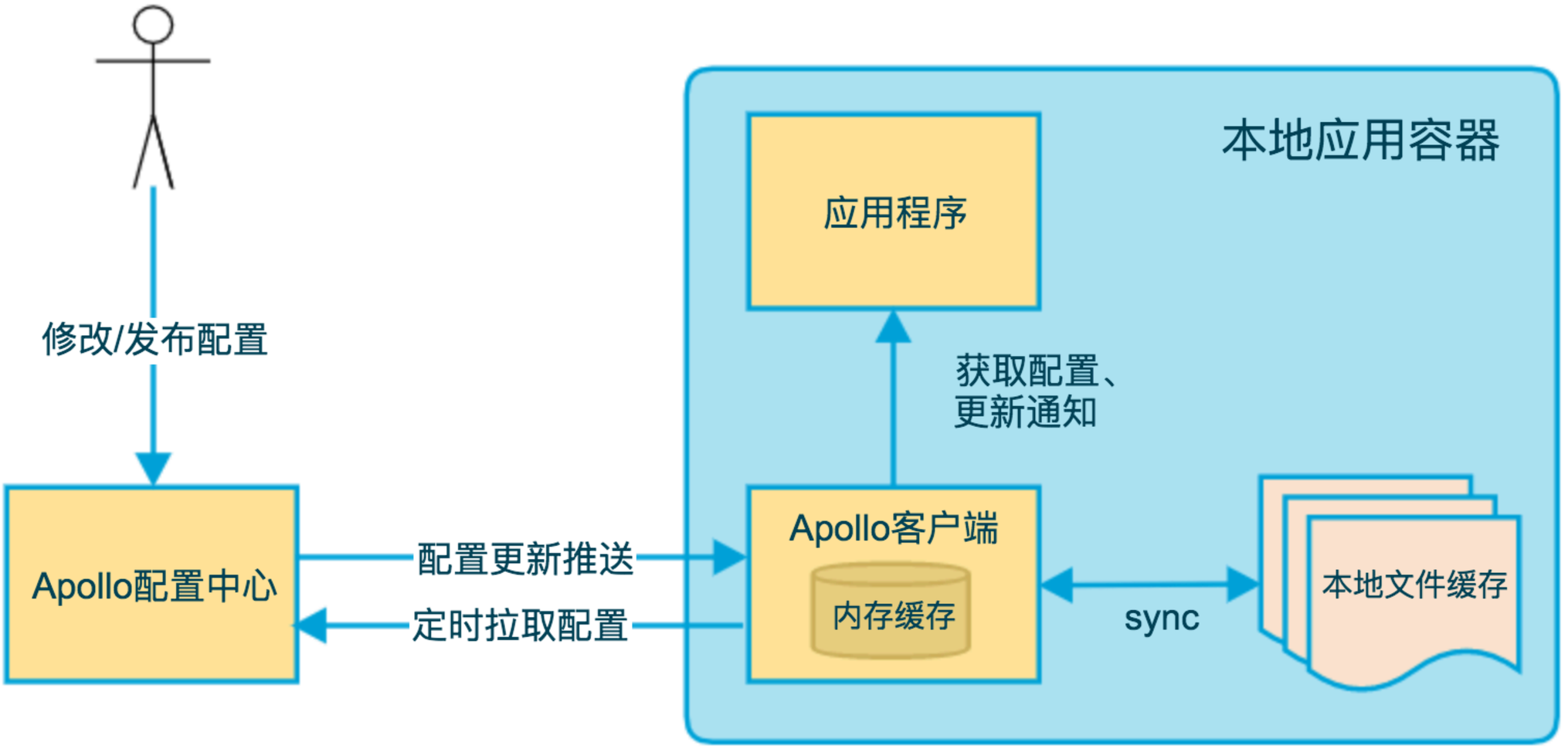
# Apollo at a glance



# 服务端高可用


Client
PortalDB
Portal
2. findConfigServices
2. findAdminServices
(Software) Load Balancer
Access via ip:port
Client side load balance
Error retry
4. remoteCallAPI
Meta Server
Meta Server
Meta Server
3. find service
3. find service
3. find service
Eureka
replicate
Eureka
replicate
Eureka
1. register/renew/cancel
1. register/renew/cancel
Config Service
Admin Service
read/write
ConfigDB
read/write
Access via ip:port
Client side load balance
Error retry
4. remoteCallAPI

# 客户端高可用


修改/发布配置
Apollo配置中心
配置更新推送
定时拉取配置
本地应用容器
应用程序
获取配置、
更新通知
Apollo客户端
内存缓存
sync
本地文件缓存

# 可用性场景举例

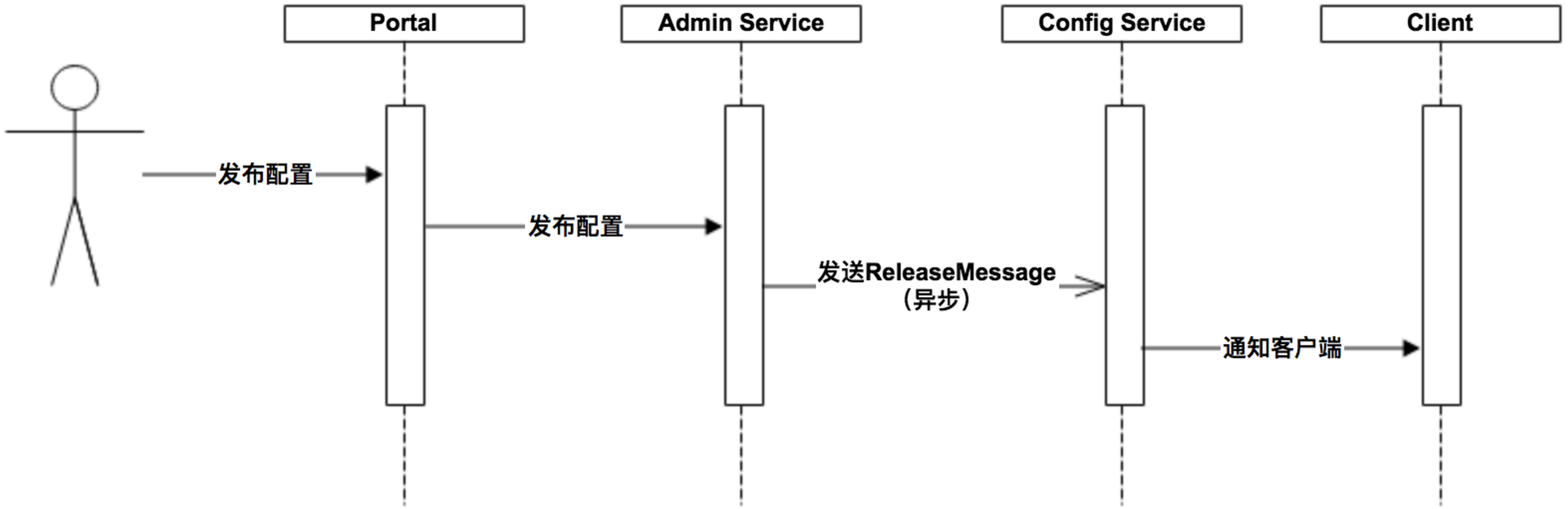
| 场景 | 影响 | 降级 | 原因 |
| --- | --- | --- | --- |
| 某台Config Service下线 | 无影响 | | Config Service无状态，客户端重连其它Config Service |
| 所有Config Service下线 | 客户端无法读取最新配置，Portal无影响 | 客户端重启时，可以读取本地缓存配置文件。如果是新扩容的机器，可以从其它机器上获取已缓存的配置文件 | |
| 某台Admin Service下线 | 无影响 | | Admin Service无状态，Portal重连其它Admin Service |
| 所有Admin Service下线 | 客户端无影响，Portal无法更新配置 | | |
| 某台Portal下线 | 无影响 | | Portal域名通过SLB绑定多台服务器，重试后指向可用的服务器 |
| 全部Portal下线 | 客户端无影响，Portal无法更新配置 | | |
| 某个数据中心下线 | 无影响 | | 多数据中心部署，数据完全同步，Meta Server/Portal域名通过SLB自动切换到其它存活的数据中心 |
| 数据库全部宕机 | 客户端无影响，Portal无法更新配置 | Config Service开启配置缓存后，对配置的读取不受数据库宕机影响 | |

# 实时性

配置即『控制』，所以我们希望我们的控制指令能迅速、准确地传达到应用程序

图片来源：http://toysoxo.com/toysoxo-shipping-delivery-informaton/

# 配置发布的过程


Portal
Admin Service
Config Service
Client
发布配置
发布配置
发送ReleaseMessage
（异步）
通知客户端

# 发送ReleaseMessage的实现方式

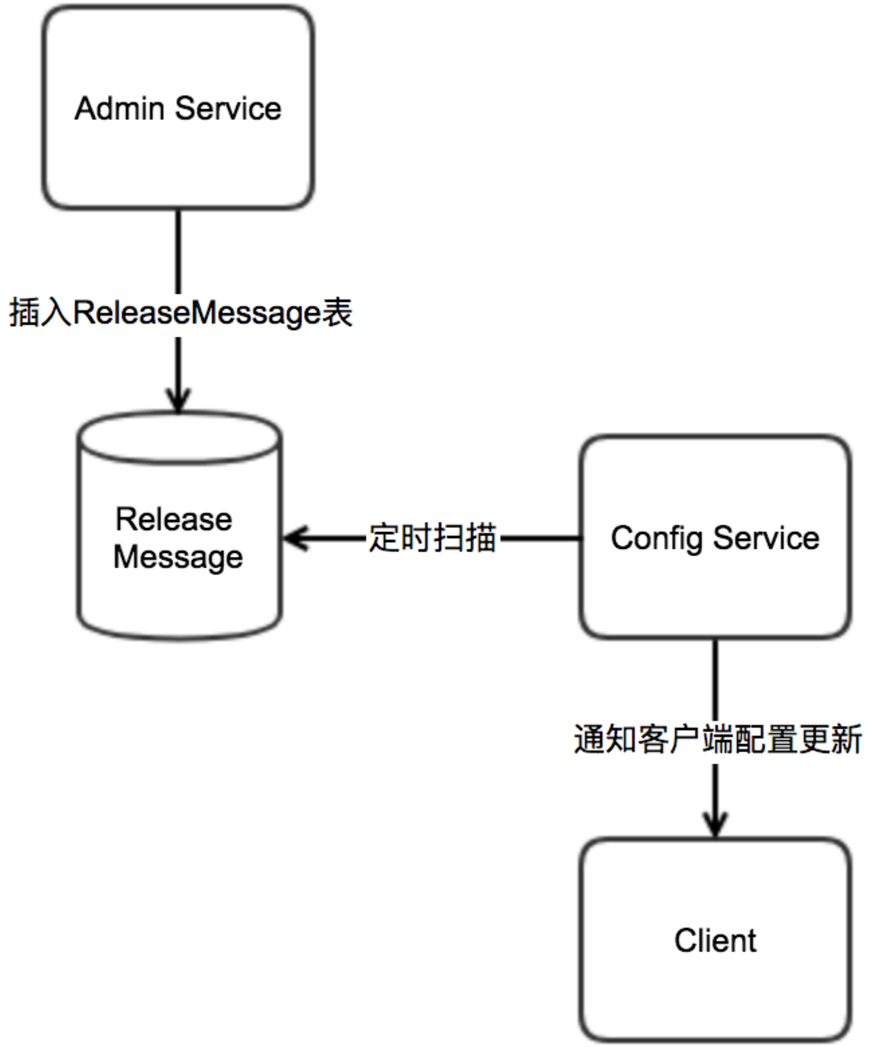


TABLE OF

# CONTENTS 大纲

1. 为什么需要配置中心?
2. 配置中心的一般模样
3. 如何让微服务更『智能』?
4. 配置中心的最佳实践

# 开关

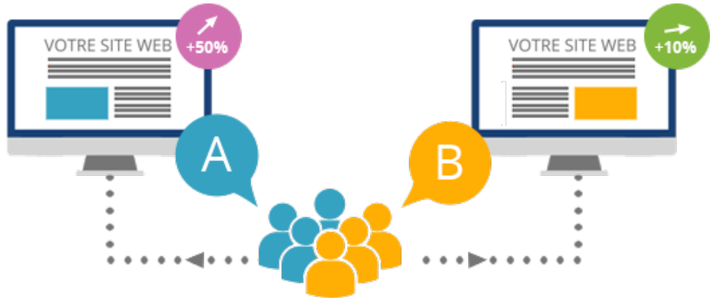
# 发布开关

- 发布开关一般用于发布过程中，比如：
  1. 有些新功能依赖于其它系统的新接口，而其它系统的发布周期未必和自己的系统一致，可以加个发布开关，默认把该功能关闭，等依赖系统上线后再打开。
  2. 有些新功能有较大风险，可以加个发布开关，上线后一旦有问题可以迅速关闭

- 需要注意的是，发布开关应该是短暂存在的（1-2周），一旦功能稳定后需要及时清除开关代码。

# 实验开关

- A/B测试
  - 针对特定用户应用新的推荐算法
  - 针对特定百分比的用户使用新的下单流程




- 功能验证
  - 有些重大功能已经对外宣称在某年某日发布
  - 可以事先发到生产环境，只对内部用户打开，测试没问题后按时对全部用户开放


- 实验开关应该也是短暂存在的，一旦实验结束了需要及时清除实验开关代码。

图片来源：https://www.search-factory.cn/ab-testing/, http://www.businesses.com.au/marketing/418276-top-4-reasons-to-hire-promo-staff-for-your-product-launch-

# 运维开关

- 运维开关通常用于提升系统稳定性，比如：
  1. 大促前可以把一些非关键功能关闭来提升系统容量
  2. 当系统出现问题时可以关闭非关键功能来保证核心功能正常工作

- 运维开关可能会长期存在，而且一般会涉及多个系统，所以需要提前规划。

# 服务治理

# 限流

正常的高速公路

图片来源：https://uspirg.org/sites/pirg/files/cpn/USN-041817-A3-REPORT/highway-boondoggles-3.html

# 限流

超出容量的高速公路

图片来源：https://abcnews.go.com/International/thousands-cars-stuck-beijing-traffic-jam-50-lane/story?id=34350370

# 限流

- 服务就像高速公路一样
  - 在正常情况下非常通畅
  - 不过一旦流量突增（比如大促、遭受DDOS攻击）时，如果没有做好限流，就会导致系统整个被冲垮，所有用户都无法访问。

- 所以我们需要限流机制来应对此类问题
  - 一般的做法是在网关或RPC框架层添加限流逻辑，结合配置中心的动态推送能力实现动态调整限流规则配置。

# 黑白名单

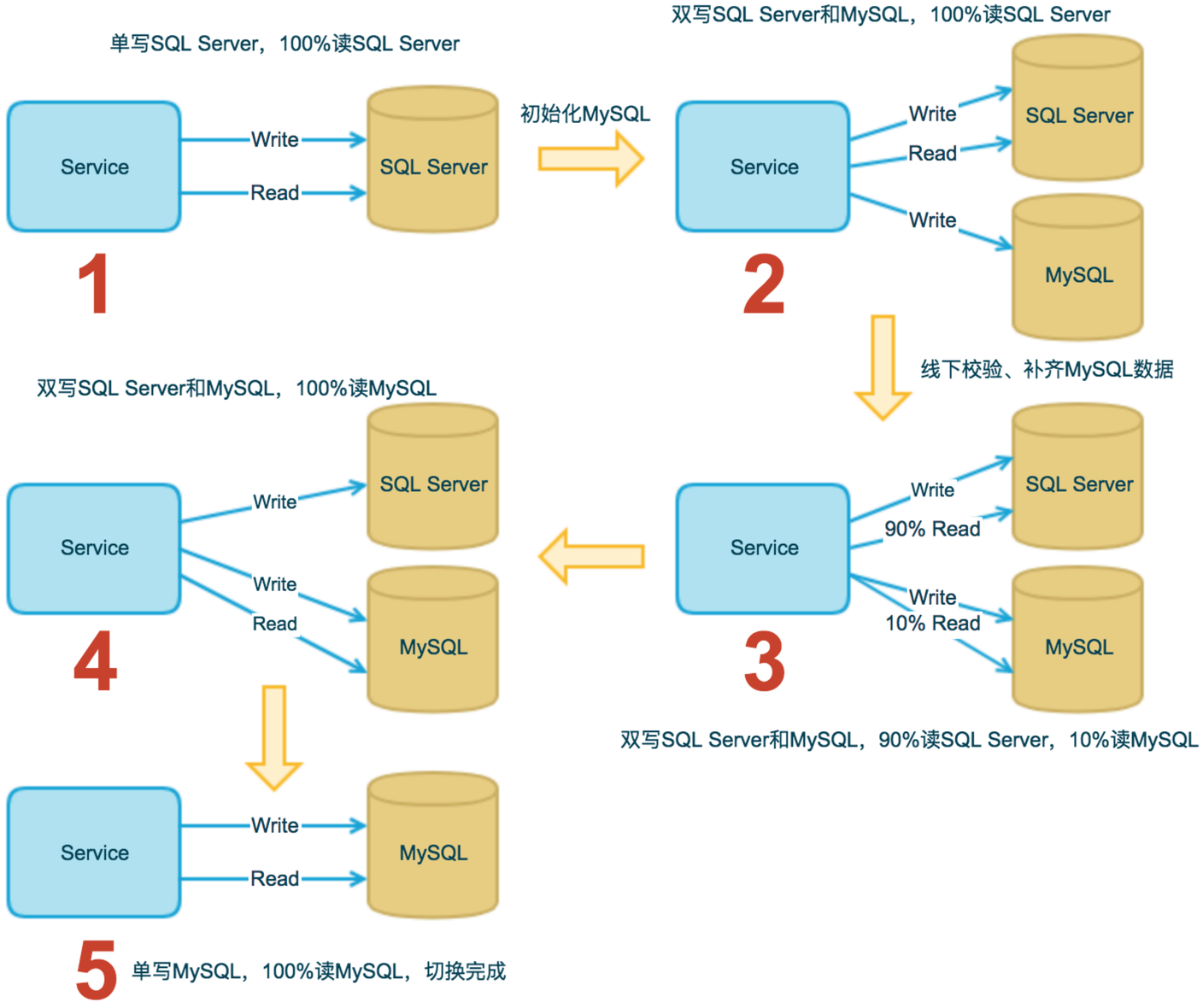
- 对于一些关键服务，哪怕是在内网环境中一般也会对调用方有所限制，比如：
  1. 有敏感信息的服务可以通过配置白名单来限制只有某些应用或IP才能调用
  2. 某个调用方代码有问题导致超大量调用，对服务稳定性产生了影响，可以通过配置黑名单来暂时屏蔽这个调用方或IP

- 所以我们需要黑白名单机制来应对此类问题
  - 一般的做法是在RPC框架层添加黑白名单逻辑，结合配置中心的动态推送能力来实现动态调整黑白名单配置。

# 数据库迁移

- 比如：原来使用的SQL Server，现在需要迁移到MySQL，这种情况就可以结合配置中心来实现平滑迁移

- 右图中的读写开关和比例配置都可以通过配置中心实现动态调整



# 动态日志级别

- 服务运行过程中，经常会遇到需要通过日志来排查定位问题的情况，然而这里却有个两难：
  1. 如果日志级别很高（如：ERROR），可能对排查问题也不会有太大帮助
  2. 如果日志级别很低（如：DEBUG），日常运行会带来非常大的日志量，造成系统性能下降

- 为了兼顾性能和排查问题，我们可以借助于日志组件和配置中心实现动态调整日志级别。

# 动态日志级别

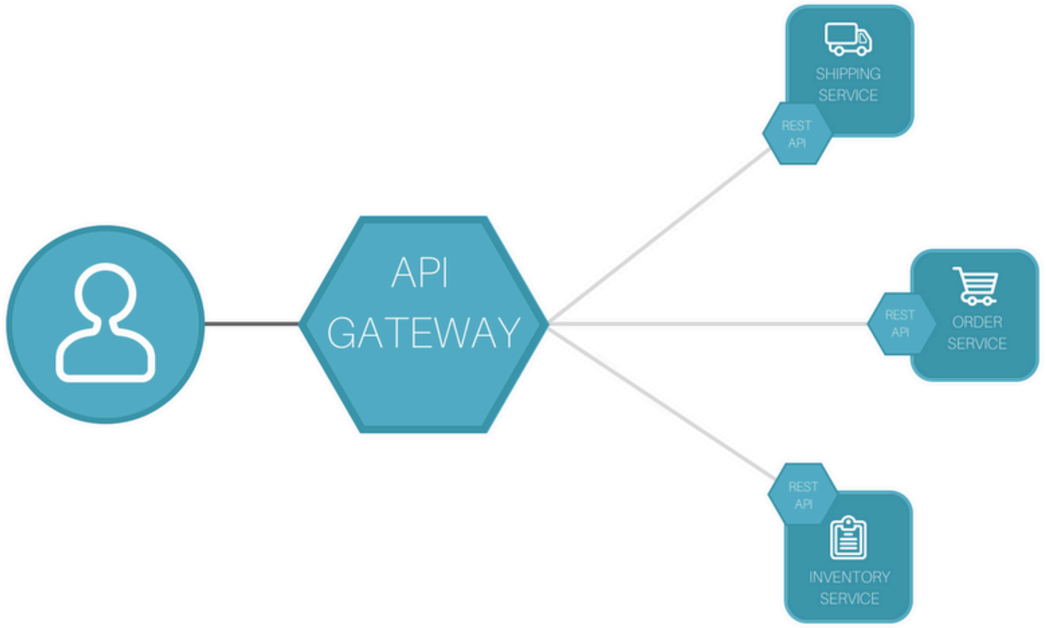
以Spring Boot和Apollo结合为例：

```
@ApolloConfigChangeListener
private void onChange(ConfigChangeEvent changeEvent) {
  refreshLoggingLevels(changeEvent.changedKeys());
}

private void refreshLoggingLevels(Set<String> changedKeys) {
  boolean loggingLevelChanged = false;
  for (String changedKey : changedKeys) {
    if (changedKey.startsWith("logging.level.")) {
      loggingLevelChanged = true;
      break;
    }
  }

  if (loggingLevelChanged) {
    // refresh logging levels
    this.applicationContext.publishEvent(new EnvironmentChangeEvent(changedKeys));
 }
}
```

详细样例代码可以参考：https://github.com/ctripcorp/apollo-use-cases/tree/master/spring-cloud-logger

# 动态网关路由

- 网关的核心功能之一就是路由转发
  - 其中的路由信息也是经常会需要变化的
  - 可以结合配置中心实现动态更新路由信息




图片来源：https://www.lunchbadger.com/vs-amazon-aws-api-gateway-express-pricing/

# 动态网关路由

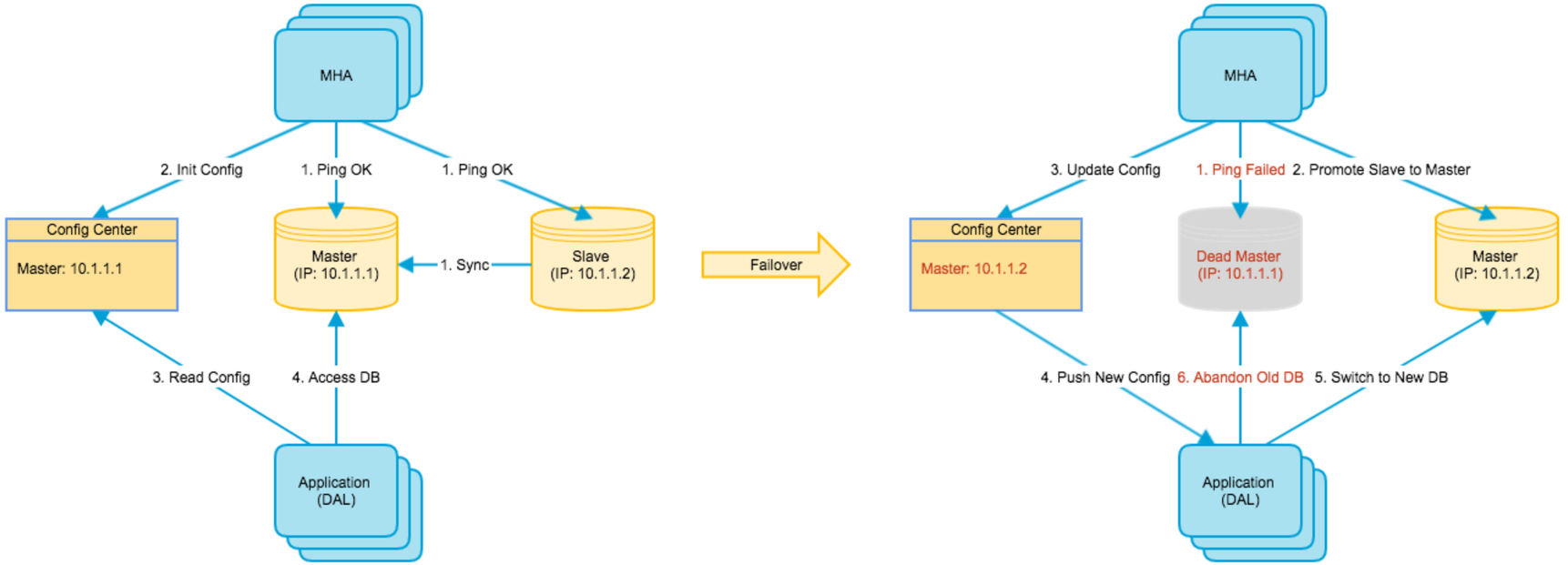
以Spring Cloud Zuul和Apollo结合为例：

```
@ApolloConfigChangeListener
public void onChange(ConfigChangeEvent changeEvent) {
  boolean zuulPropertiesChanged = false;
  for (String changedKey : changeEvent.changedKeys()) {
    if (changedKey.startsWith("zuul.")) {
      zuulPropertiesChanged = true;
      break;
    }
  }

  if (zuulPropertiesChanged) {
    refreshZuulProperties(changeEvent);
  }
}

private void refreshZuulProperties(ConfigChangeEvent changeEvent) {
  // rebind configuration beans, e.g. ZuulProperties
  this.applicationContext.publishEvent(new EnvironmentChangeEvent(changeEvent.changedKeys()));

  // refresh routes
  this.applicationContext.publishEvent(new RoutesRefreshedEvent(routeLocator));
}
```

详细样例代码可以参考：https://github.com/ctripcorp/apollo-use-cases/tree/master/spring-cloud-zuul

# 动态数据源

- 数据库是应用运行过程中的一个非常重要的资源，承担了非常重要的角色。
- 在运行过程中，我们会遇到各种不同的场景需要让应用程序切换数据库连接，比如：数据库维护、数据库宕机主从切换等。
- 切换过程如下图所示：



# 动态数据源

以Spring Boot和Apollo结合为例：

```
@Configuration
public class RefreshableDataSourceConfiguration {
  @Bean
  public DynamicDataSource dataSource(DataSourceManager dataSourceManager) {
    DataSource actualDataSource = dataSourceManager.createDataSource();
    return new DynamicDataSource(actualDataSource);
  }
}

public class DynamicDataSource implements DataSource {
  private final AtomicReference<DataSource> dataSourceAtomicReference;
  public DynamicDataSource(DataSource dataSource) {
    dataSourceAtomicReference = new AtomicReference<>(dataSource);
  }
  // set the new data source and return the previous one
  public DataSource setDataSource(DataSource newDataSource){
    return dataSourceAtomicReference.getAndSet(newDataSource);
  }
  @Override
  public Connection getConnection() throws SQLException {
    return dataSourceAtomicReference.get().getConnection();
  }
  ……
}
```

# 动态数据源

```
@ApolloConfigChangeListener
public void onChange(ConfigChangeEvent changeEvent) {
  boolean dataSourceConfigChanged = false;
  for (String changedKey : changeEvent.changedKeys()) {
    if (changedKey.startsWith("spring.datasource.")) {
      dataSourceConfigChanged = true;
      break;
    }
  }
  if (dataSourceConfigChanged) {
    refreshDataSource(changeEvent.changedKeys());
  }
}

private synchronized void refreshDataSource(Set<String> changedKeys) {
  try {
    // rebind configuration beans, e.g. DataSourceProperties
    this.applicationContext.publishEvent(new EnvironmentChangeEvent(changedKeys));
    DataSource newDataSource = dataSourceManager.createAndTestDataSource();
    DataSource oldDataSource = dynamicDataSource.setDataSource(newDataSource);
    asyncTerminate(oldDataSource);
  } catch (Throwable ex) {
    logger.error("Refreshing data source failed", ex);
  }
}
```

详细样例代码可以参考：https://github.com/ctripcorp/apollo-use-cases/tree/master/dynamic-datasource

TABLE OF

# CONTENTS 大纲

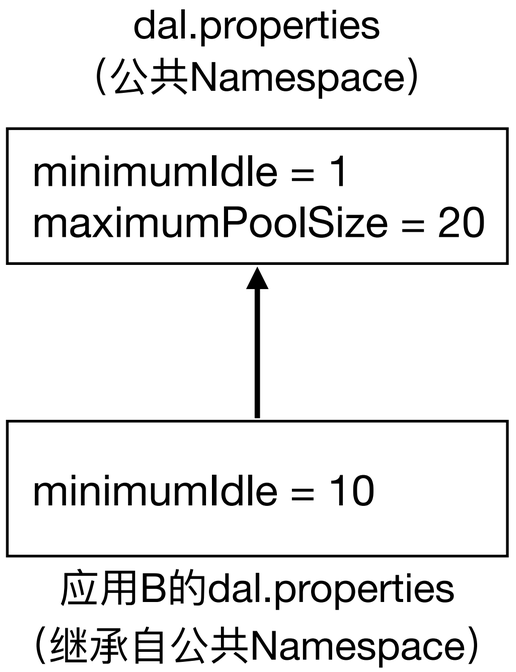
1. 为什么需要配置中心?
2. 配置中心的一般模样
3. 如何让微服务更『智能』?
4. 配置中心的最佳实践

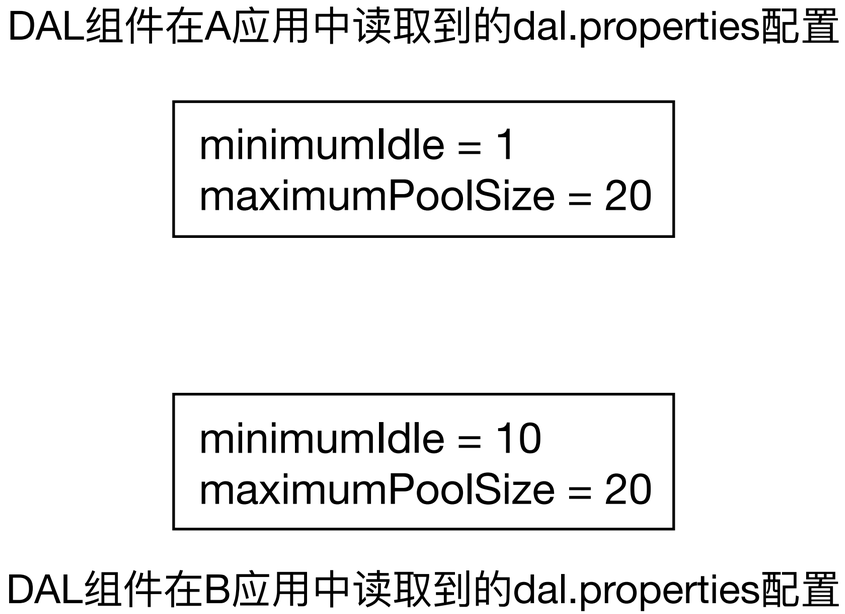
# 公共组件的配置

- 公共组件是指那些发布给其它应用使用的客户端代码，比如RPC客户端、DAL客户端等。
- 这类组件一般是由单独的团队（如中间件团队）开发、维护，但是运行时是在业务实际应用内的，所以本质上可以认为是应用的一部分。
- 这类组件的特殊之处在于大部分的应用都会直接使用中间件团队提供的默认值，少部分的应用需要根据自己的实际情况对默认值进行调整。

# 公共组件的配置

- 比如数据库连接池的最小空闲连接数量（minimumIdle），出于对数据库资源的保护，DBA要求将全公司默认的minimumIdle设为1，对大部分的应用可能都适用，不过有些核心/高流量应用可能觉得太小，需要设为10。



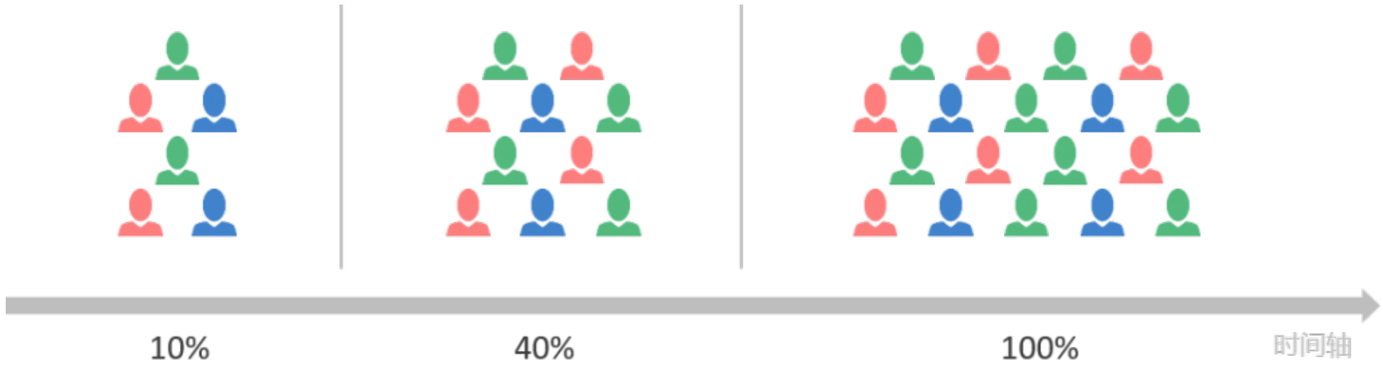




- 通过这种方式的好处是不管是中间件团队，还是应用开发，都可以灵活地动态调整公共组件的配置。

# 灰度发布

- 对于重要的配置一定要做灰度发布，先在一台或多台机器上生效后观察效果，如果没有问题再推给所有的机器。

- 对于公共组件的配置，建议先在一个或多个应用上生效后观察效果，没有问题再推给所有的应用。




图片来源：http://www.appadhoc.com/blog/one-page-canary-release-test/

# 发布审核

- 生产环境建议启用发布审核功能，简单而言就是如果某个人修改了配置，那么必须由另一个人审核后才可以发布

- 避免由于头脑不清醒、手一抖之类的造成生产事故。


图片来源：https://livinator.com/5-tips-to-get-easier-hoa-approval-for-your-home-renovations/

TABLE OF

# CONTENTS 大纲

1. 为什么需要配置中心?
2. 配置中心的一般模样
3. 如何让微服务更『智能』?
4. 配置中心的最佳实践

# THANKS!

SHANGHAI

关注『携程技术中心』微信公众号，了解更多技术干货！