海量 Node.js 的DevOps实践

王伟嘉 / Starkwang

@GMTC

Beijing 2019

Geekbang. 极客邦科技 InfoQ

极客邦科技
会议推荐2019

5月

QCon 北京
全球软件开发大会
大会：5月6–8日
培训：5月9–10日

QCon 广州
全球软件开发大会
培训：5月25–26日
大会：5月27–28日

6月

GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14–15日

GMTC 北京
全球大前端技术大会
大会：6月20–21日
培训：6月22–23日

7月

ArchSummit 深圳
全球架构师峰会
大会：7月12–13日
培训：7月14–15日

10月

QCon 上海
全球软件开发大会
大会：10月17–19日
培训：10月20–21日

11月

GMTC 深圳
全球大前端技术大会
大会：11月8–9日
培训：11月10–11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21–22日
培训：11月23–24日

12月

ArchSummit 北京
全球架构师峰会
大会：12月6–7日
培训：12月8–9日

Geekbang. 极客邦科技 | InfoQ 极客传媒

InfoQ官网 全新改版上线

促进软件开发领域知识与创新的传播

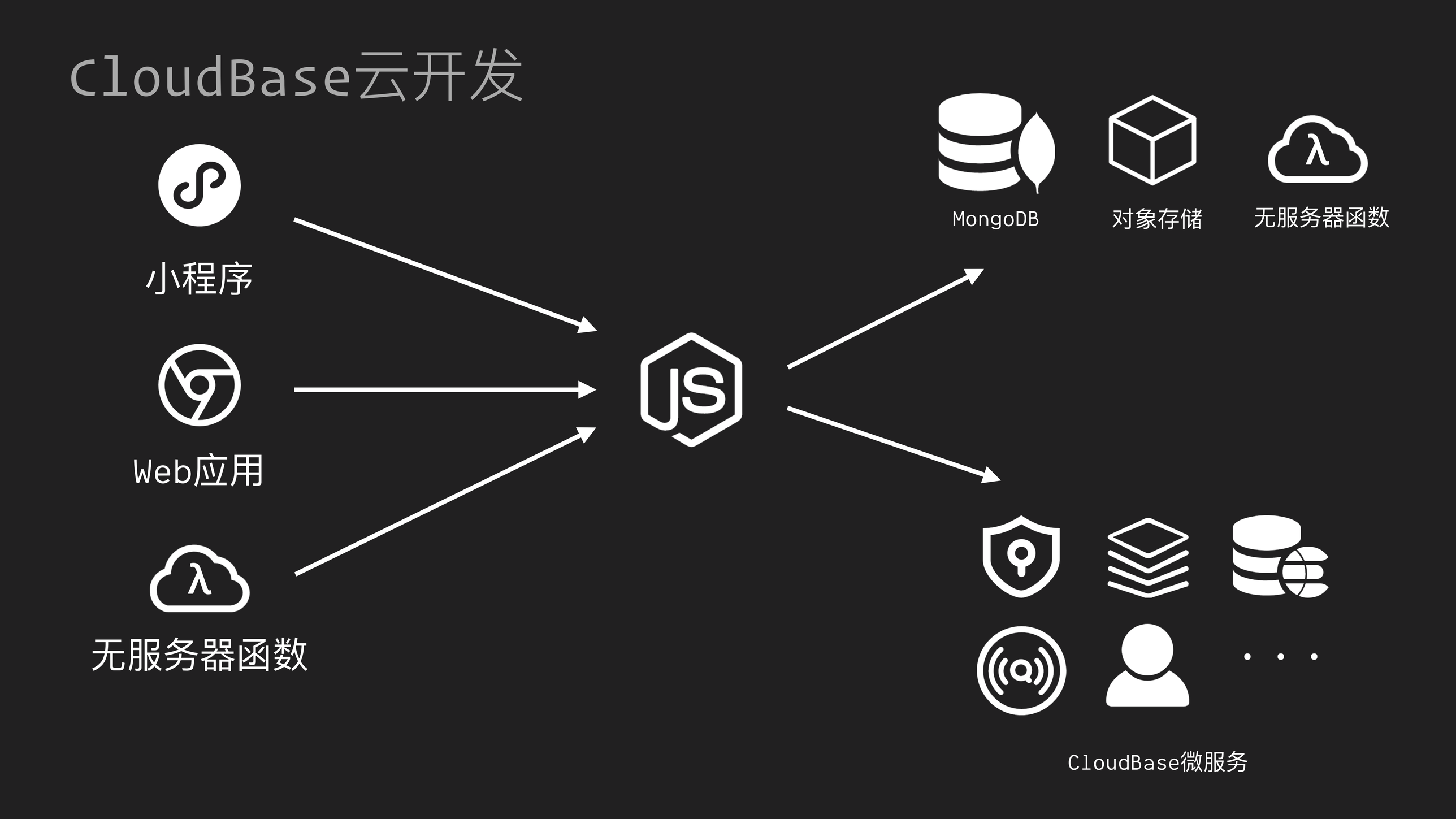
关注InfoQ网站
第一时间浏览原创IT新闻资讯

免费下载迷你书
阅读一线开发者的技术干货

```
{
    "name": "王伟嘉",
    "nickname": "Starkwang",
    "graduated_from": "Fudan University",
    "working_in": "Tencent Cloud",
    "tags": ["Node.js Core Collaborator"],
    "products": ["小程序·云开发", "CloudBase"]
}
```

Node.js at CloudBase

CloudBase云开发



快速入门DevOps

一种传统的分工

研发

技术选型

业务逻辑

海量服务

监控日志

QA

质量管理

测试用例

测试报告

版本发布

运维

服务器运维

实例扩容

服务调度

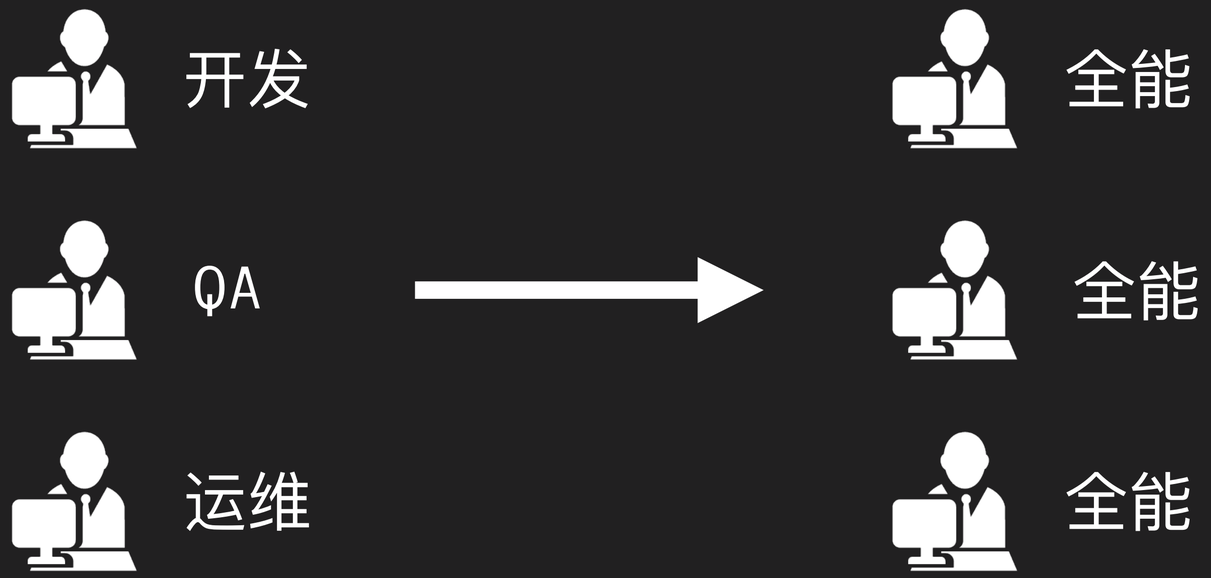
基础设施

? ? ?

? ? ?

时代已经变了！

技术的发展使每个人能承担更多的职责



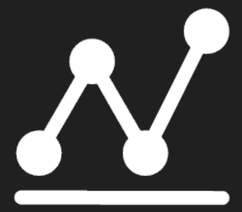
Development Operations

DevOps

基础设施

质量管理

持续集成



日志监控

基础设施篇

什么是基础设施?

Node.js服务

操作系统

运行时

数据库

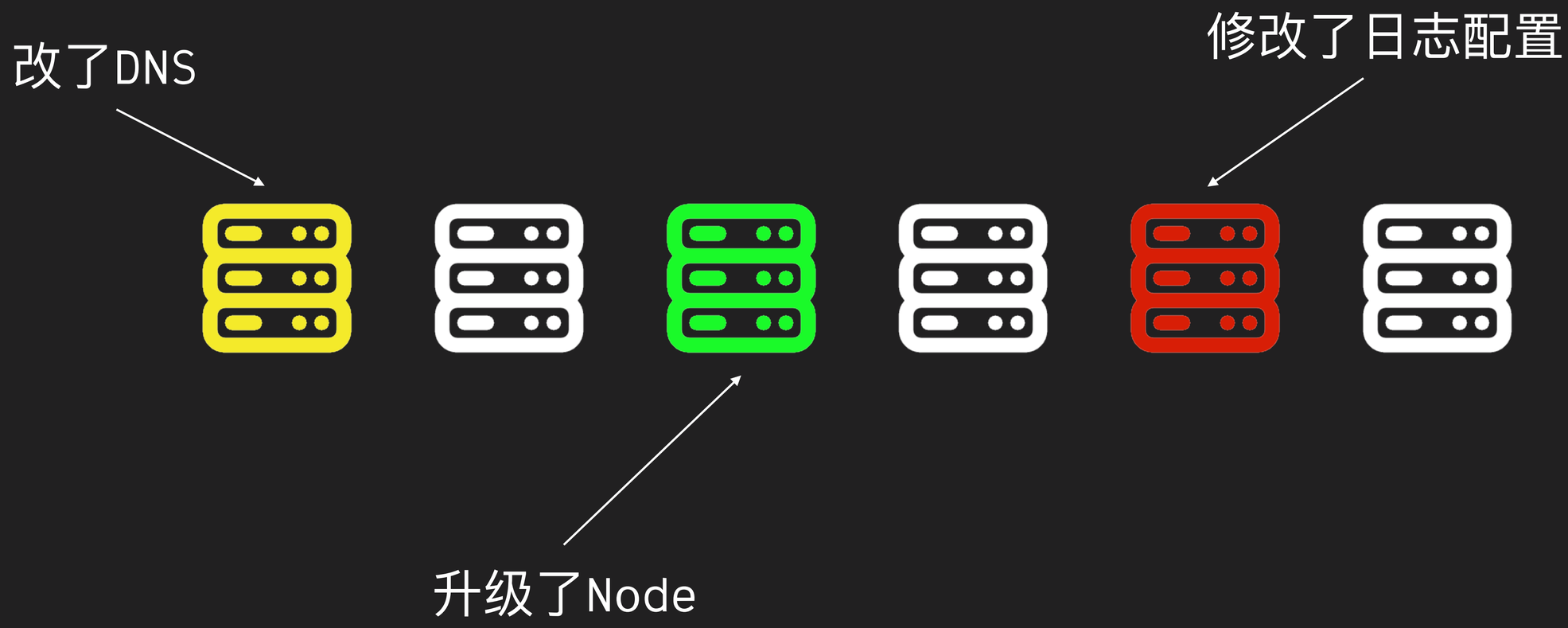
服务发现

负载均衡

.....

基础设施的挑战

集群漂移



基础设施的挑战

雪花服务器

手工配置的Nginx

第三方的SDK

SMS服务器

特殊的代理设置

？？？

基础设施的挑战

时间侵蚀

- 软件升级
- 磁盘被写满
- 进程死锁或崩溃
- 网络异常
- 硬件问题
- 台风把机房淹了
- ……

基础设施的挑战

Node.js的特殊性

没有可直接运行的二进制包

依赖环境运行时

node_modules黑洞

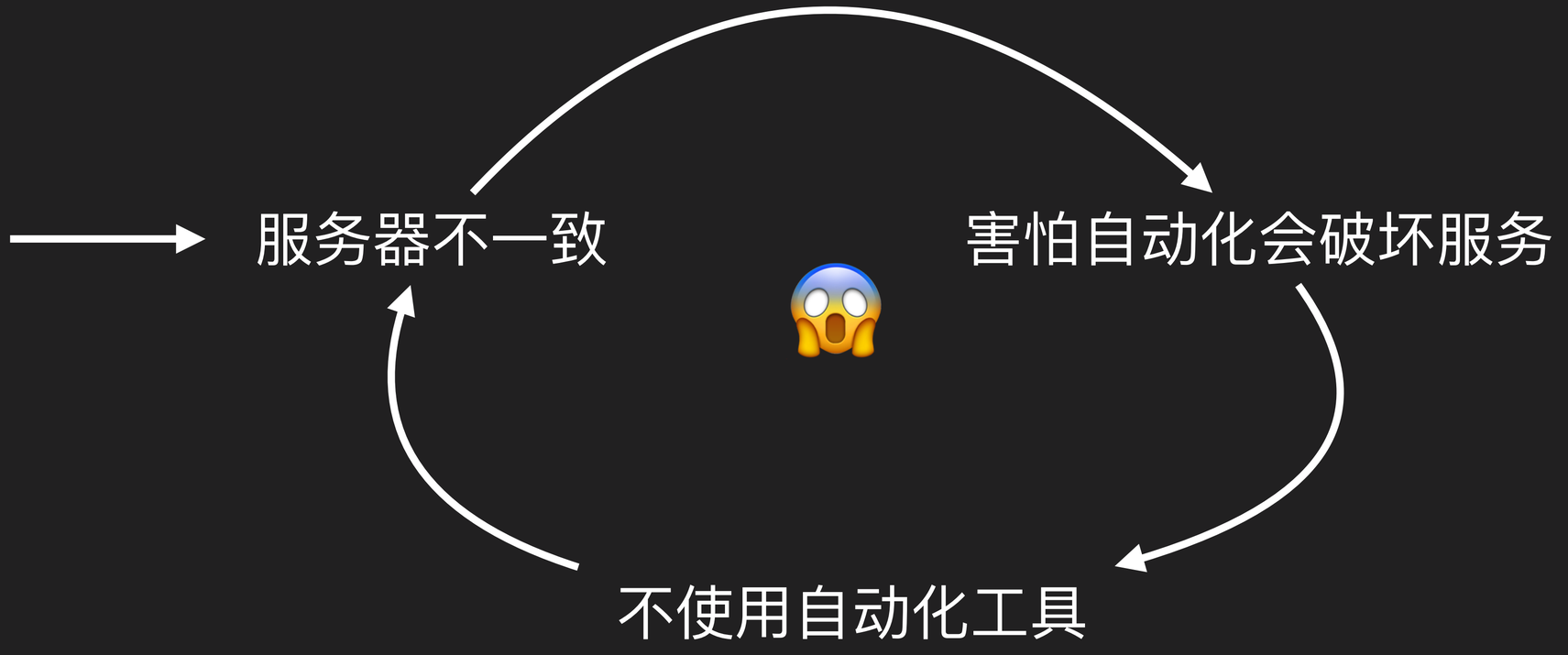
自动化恐惧症

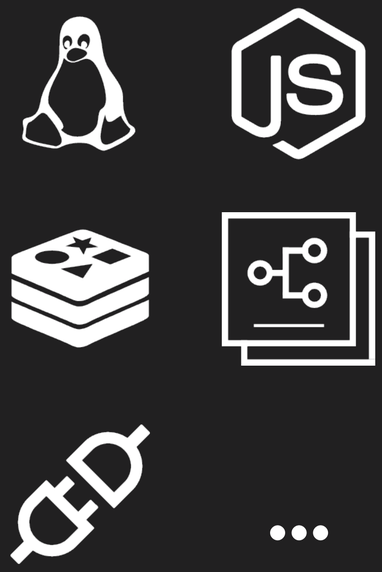
集群漂移

雪花服务器

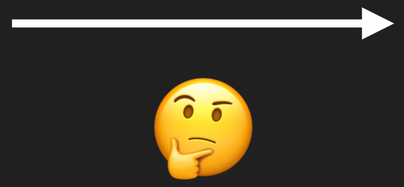
时间侵蚀

Node.js的特殊性





设施难以维护



```
FROM node:10
WORKDIR /usr/src/app
COPY package*.json ./
RUN export http_proxy=...
RUN npm install
...
```

代码容易维护

基础设施即代码

定义与原则

使用代码定义动态化的基础设施

Dockerfile

自动化脚本

配置文件

定义文件

VM、容器

智能网关

服务发现

任何可编程的基础设施

原则

- 低成本复制
- 用完可扔
- 严格一致
- 过程可重复
- 变更成本低

基础设施即代码的实践

迁移到动态基础设施

静态分配的VM → Docker、Kubernetes、FaaS

Nginx → API Gateway

hosts/DNS → 服务发现

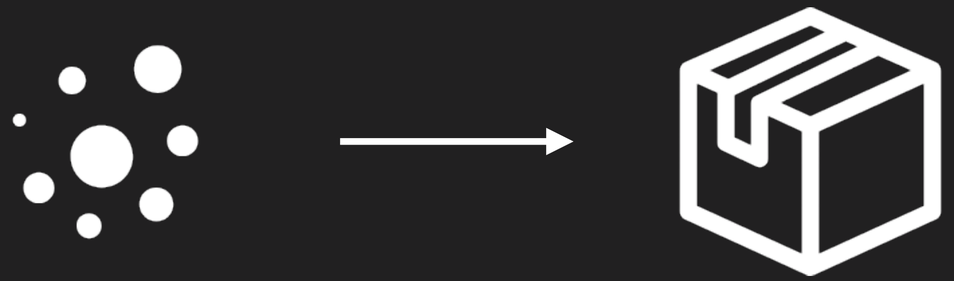
基础设施即代码的实践

代码构建服务

Dockerfile

```
FROM node:10

WORKDIR /usr/local/service/my-nodejs-app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 8080

CMD [ "npm", "start" ]
```

- 使用Node.js v10
- npm依赖
- 暴露8080端口
- 启动服务



基础设施即代码的实践

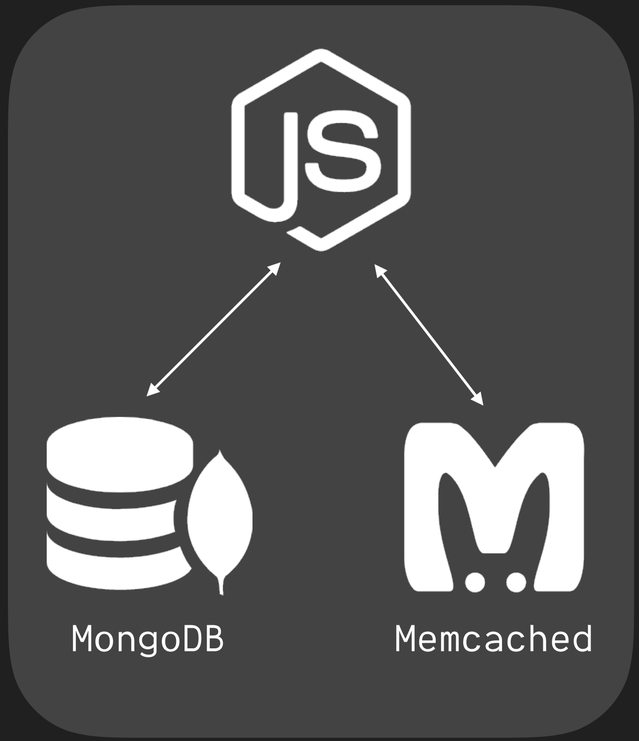
代码定义基础设施

docker-compose.yaml

```yaml
version: '3'
services:
  master:
    image: your-nodejs-service
    restart: always
    ports:
      - "0.0.0.0:8080:8080"
    volumes:
      - ./dev-ssh:/root/.ssh
    environment:
      NODE_ENV: production
  memcached:
    image: memcached
    restart: always
    ports:
      - "11211:11211"
  mongo:
    image: mongo
    restart: always
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: xxxx
      MONGO_INITDB_ROOT_PASSWORD: xxxx
      MONGO_INITDB_DATABASE: xxxx
```

主服务

依赖服务



基础设施即代码的实践

不可变服务器

修改现有的服务器

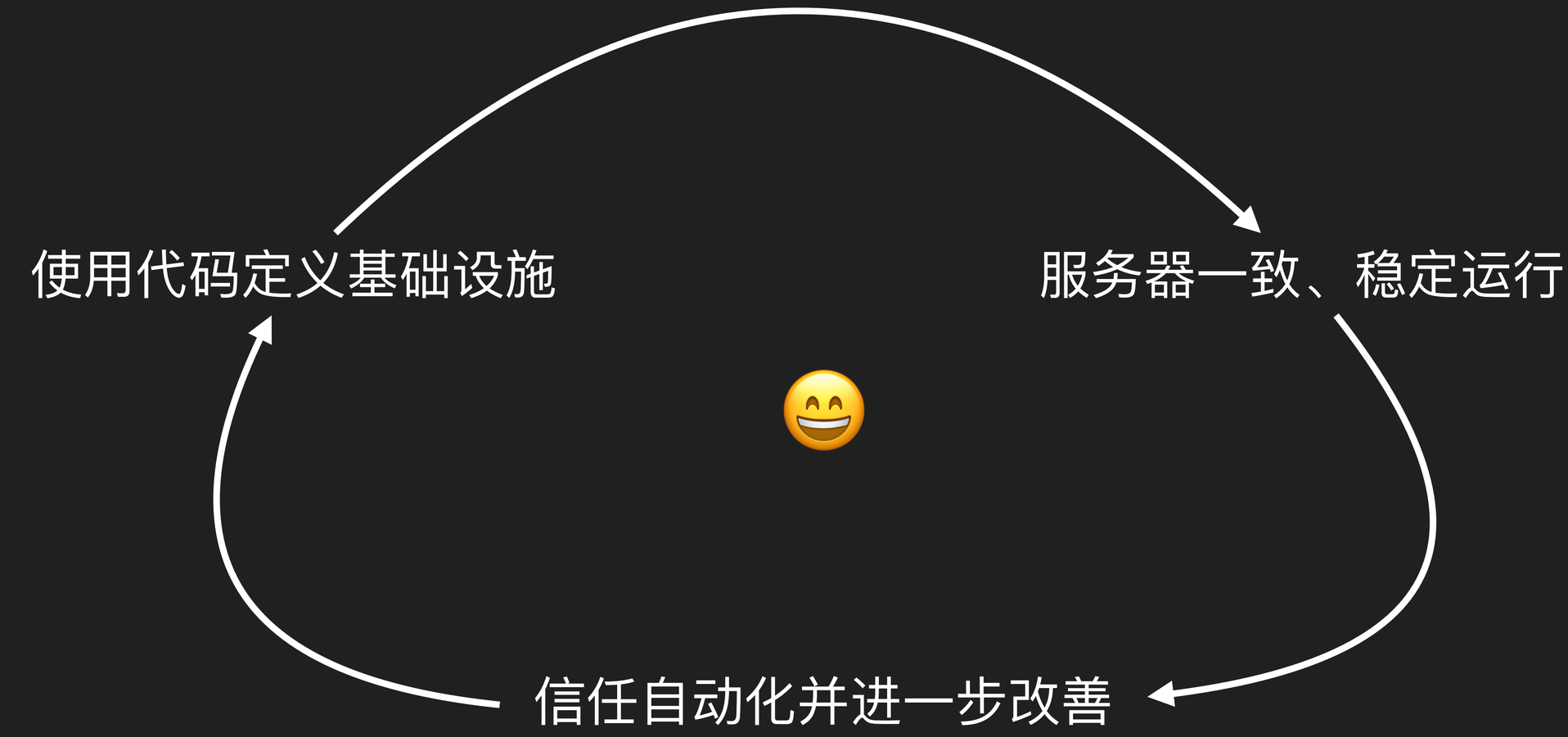
销毁，构建新的服务器

更高的一致性、可靠性

更简单、可预测

彻底解决传统模式的一些问题

打破恶性循环

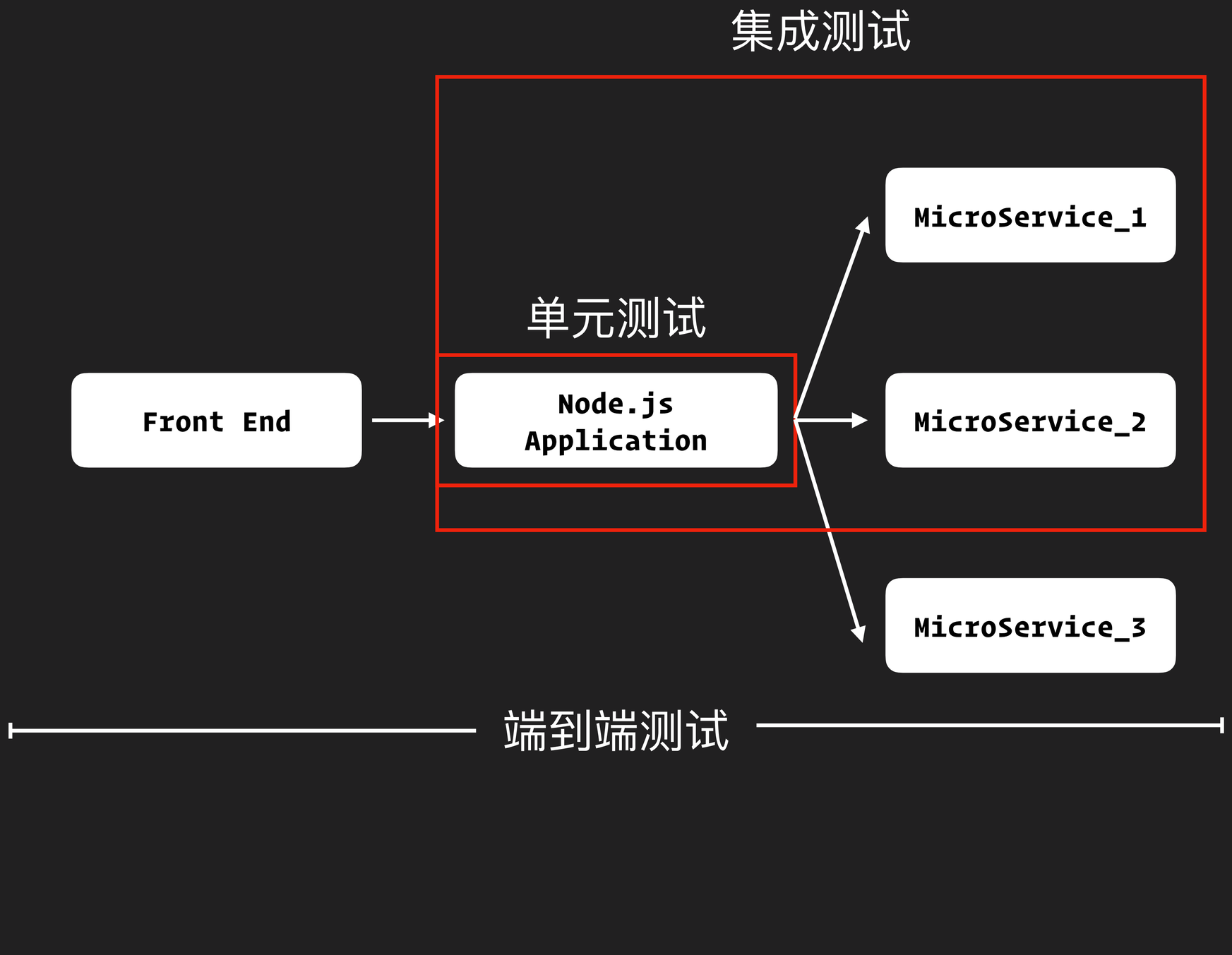


质量管理篇

为什么不写测试?

- 业务迭代超快，没必要写
- 需求都赶不过来，没时间写
- 代码架构设计有问题，不好写

“花在编写测试上的每一分钟时间，
都会在未来的维护成本上翻倍地赚回来。”


集成测试
单元测试
Front End
Node.js
Application
MicroService_1
MicroService_2
MicroService_3
端到端测试

单元测试

测试对象：类、函数、模块等最小测试单元

```
function sum(x, y) {
  return x + y
}
```

```
test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3)
})
```

```
class Foo {
  constructor(value) {
    this.value = value
  }
  read() {
    return this.value
  }
}
```

```
test('Foo#value() returns its real value', () => {
  const value = 42
  const foo = new Foo(value)
  expect(foo.read()).toBe(value)
})
```

单元测试

为什么你的单元测试写不起来?	解决方法
Dirty code太多，能测试的函数太少	尽量拆分出纯函数
模块间强耦合	设计模式是有用的!
调通就好了，没动力写单元测试	团队需要重视测试覆盖率

编写优质的单元测试

测试代码与业务代码低耦合

```
class Person {
    constructor(name) {
        this._name = name
    }
    getName() {
        return this._name
    }
}

const starkwang = new Person('starkwang')
```

```
assert.strictEqual(starkwang.getName(), 'starkwang')
```

```
assert.strictEqual(starkwang._name, 'starkwang')
```

编写优质的单元测试

测试可观测的行为

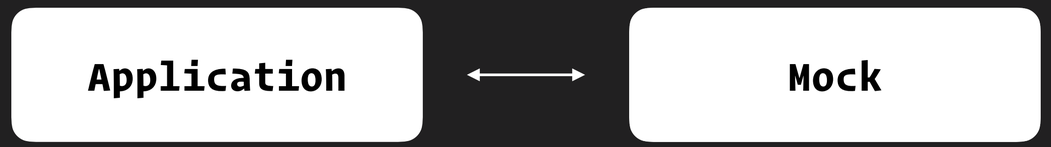
```
function foo(n) {
    return a(b(c(d(n))))
}
```

```
assert.strictEqual(foo(42), 42)
```

测试 foo 是否调用了a、b、c、d

集成测试

测试对象：子系统



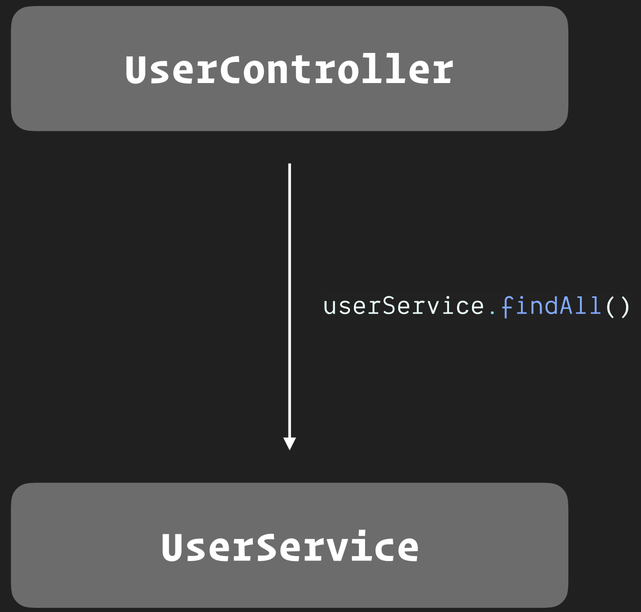

```
if (process.env.NODE_ENV === 'integration') {
  app.use(mockDatabaseService)
} else {
  app.use(databaseService)
}
```

```ts
import { Test } from '@nestjs/testing'
import { UserController } from './controller/user'
import { UserService } from './service/user'

describe('UserController', () => {
  let userController: UserController
  let userService: UserService

  beforeEach(async () => {
    const module = await Test.createTestingModule({
        controllers: [UserController],
        providers: [UserService],
      }).compile()

    userService = module.get<UserService>(UserService)
    userController = module.get<UserController>(UserController)
  });

  describe('findAll', () => {
    it('查找所有用户', async () => {
      const result = ['starkwang']
      jest.spyOn(userService, 'findAll').mockImplementation(() => result)
      expect(await userController.findAll()).toBe(result)
    })
  })
})
```

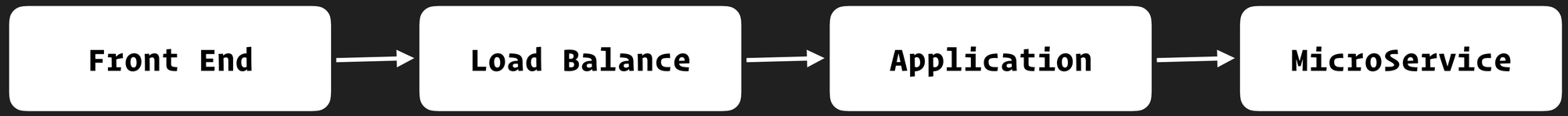
1. 创建测试module

2. mock行为

3. 断言结果

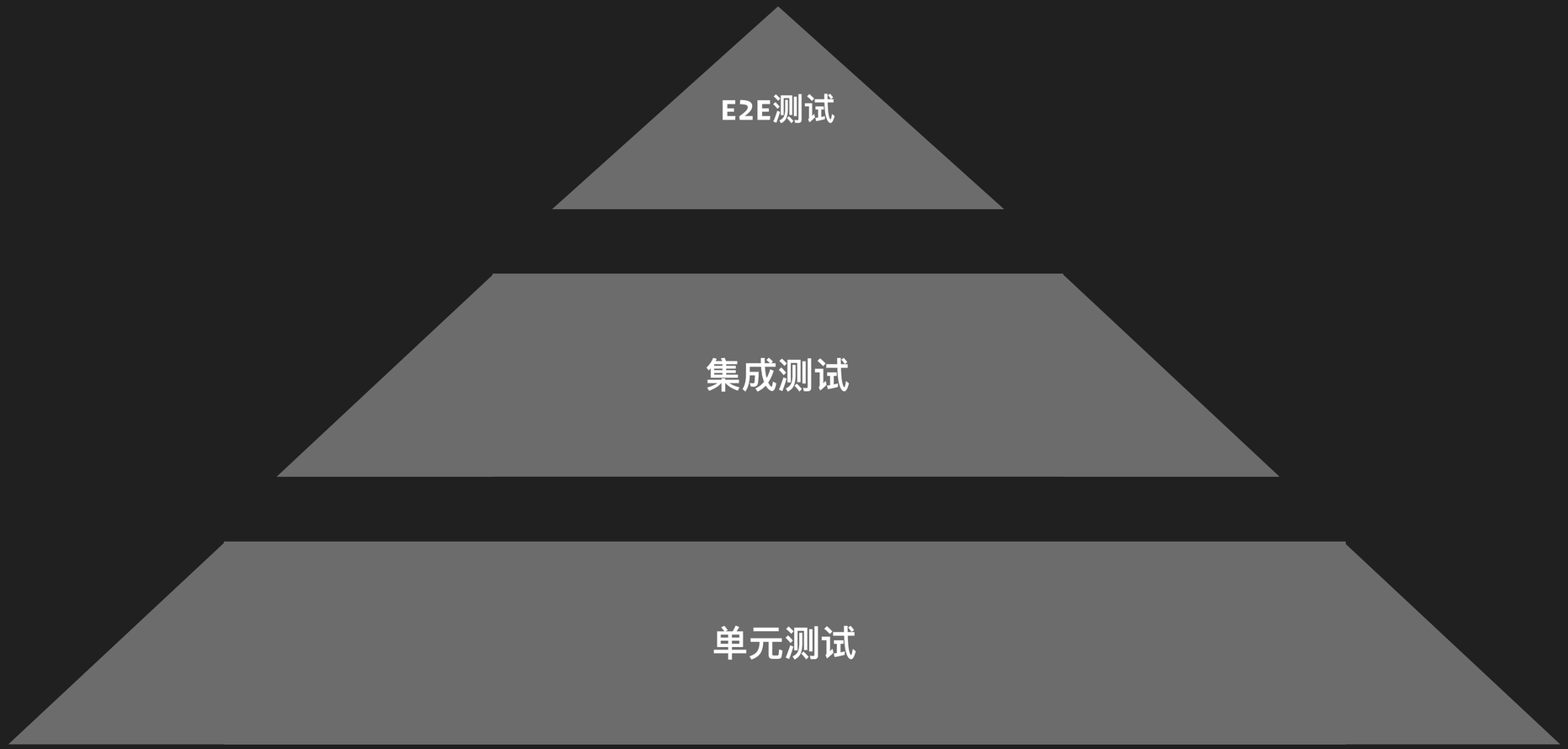


E2E测试

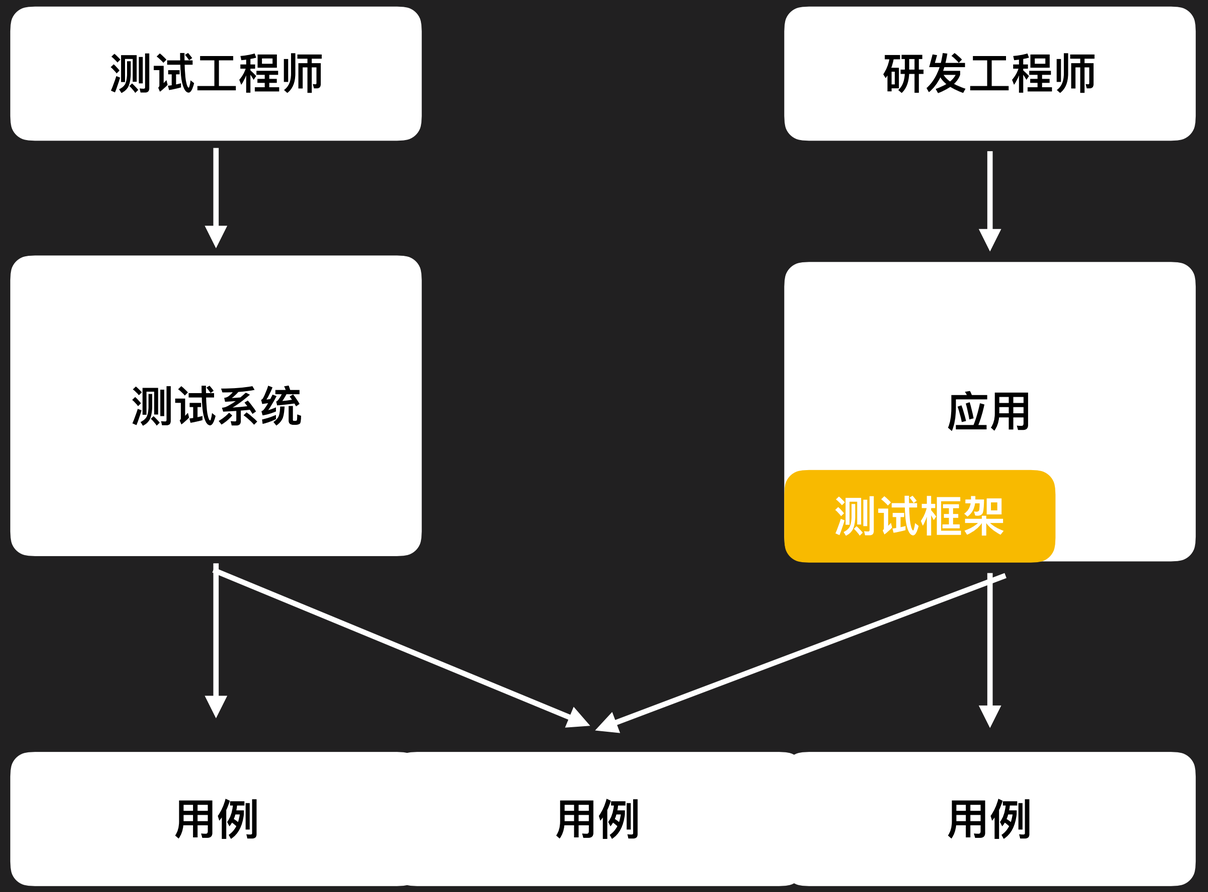


- 针对全链路的业务流程
- 始于前端，终于前端
- 用例需要覆盖全流程
- 可用于现网服务质量监控

测试金字塔



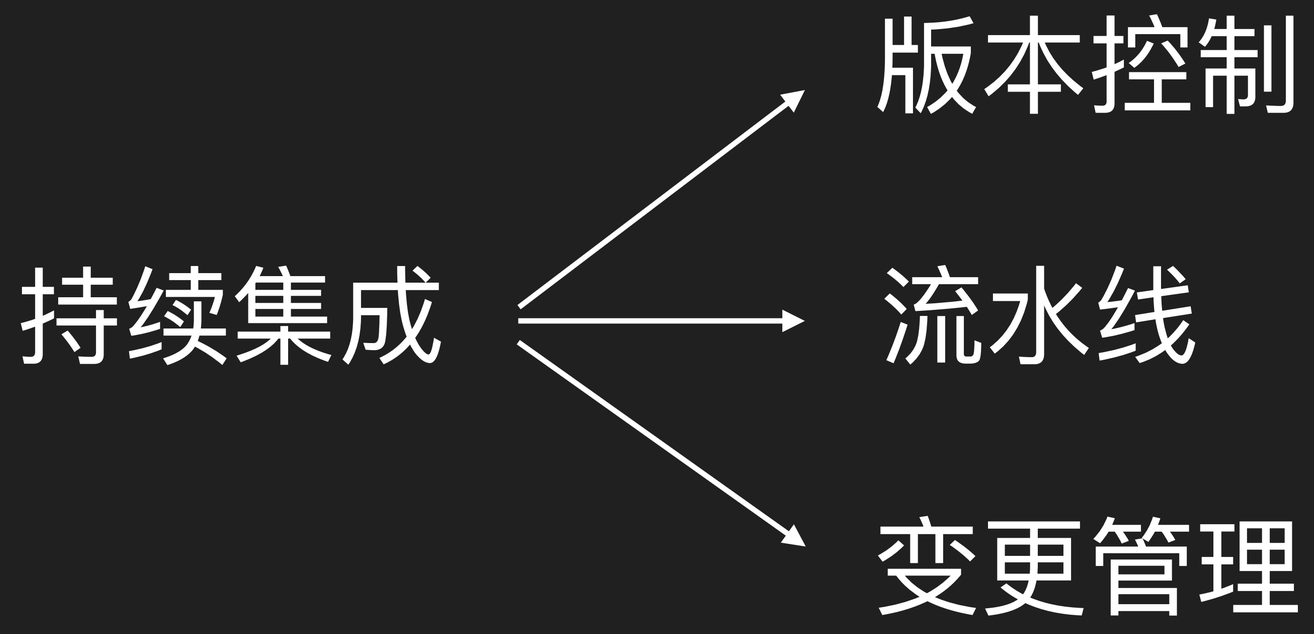
团队协作



持续集成篇

“如果做某件事很痛苦，那就频繁地去做它。”

- 分支合并
- Code Review
- 错误定位
- 进度管理
- 收集开发数据


持续集成
版本控制
流水线
变更管理

版本控制在管理什么？

- 代码
- 配置文件
- 测试代码
- CI、CD的定义
- 脚本
- 文档

- 编译产物（TS编译出的JS）
- 软件制品（zip、docker）
- 第三方依赖（node_modules）
- 日志、服务数据
- 秘钥、密码

VCS和外部系统的集成

WebHook

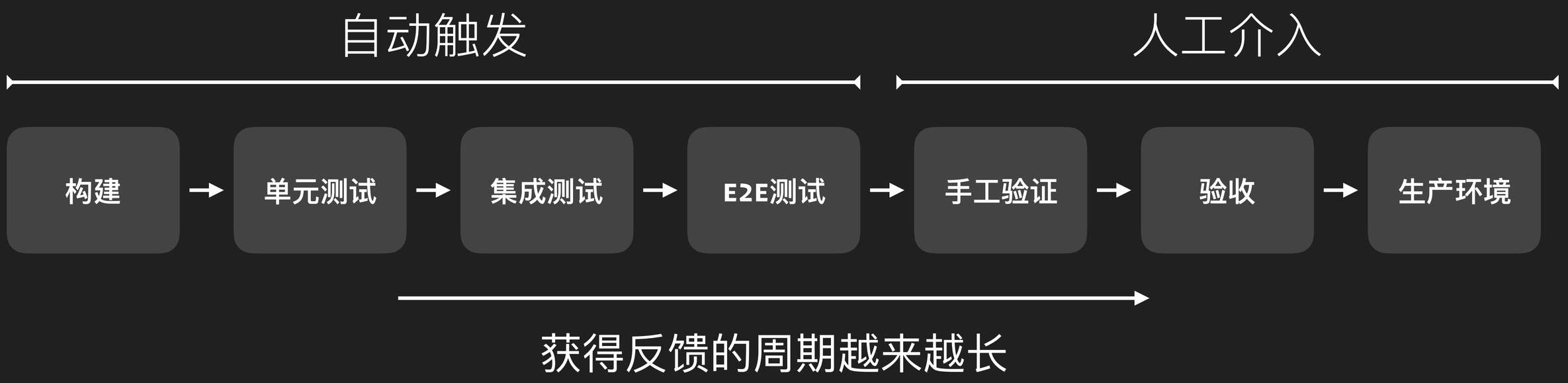
提交周知

自动化测试

覆盖率报告

发布系统

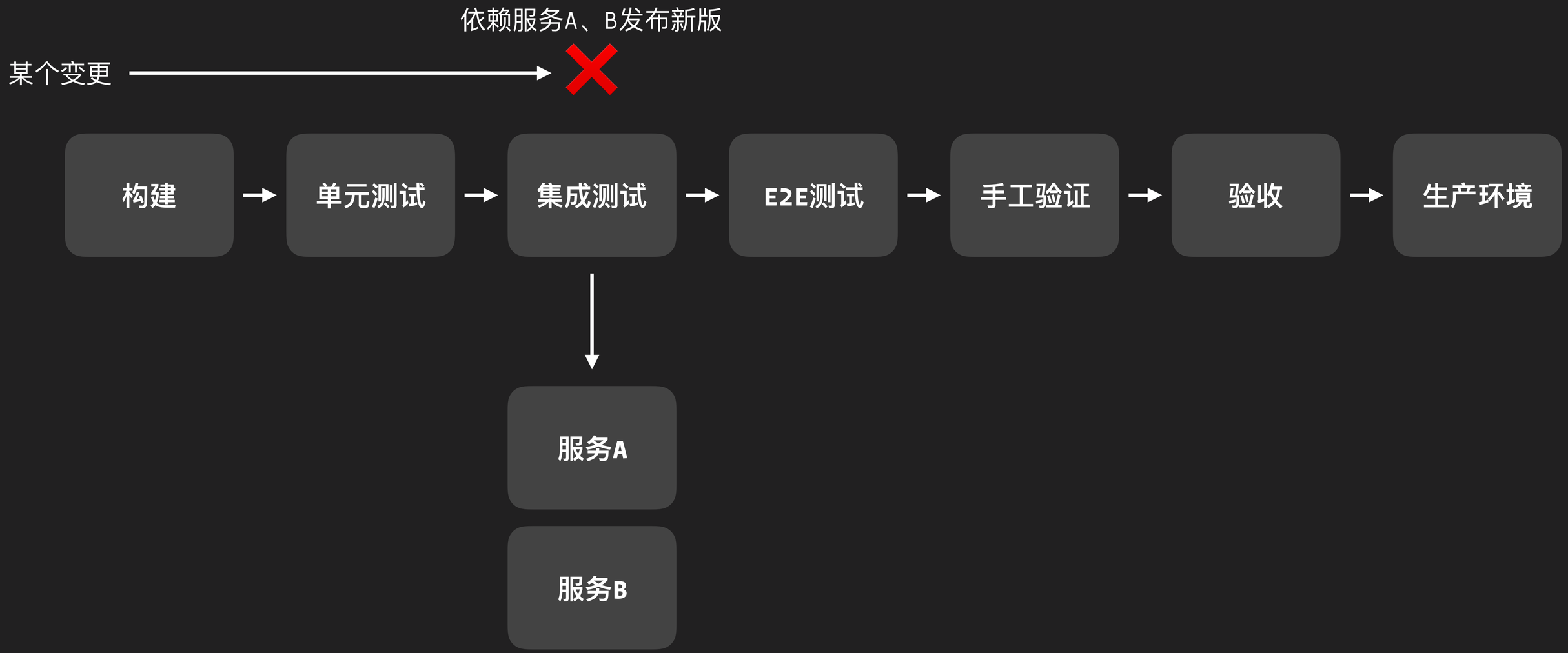
流水线设计



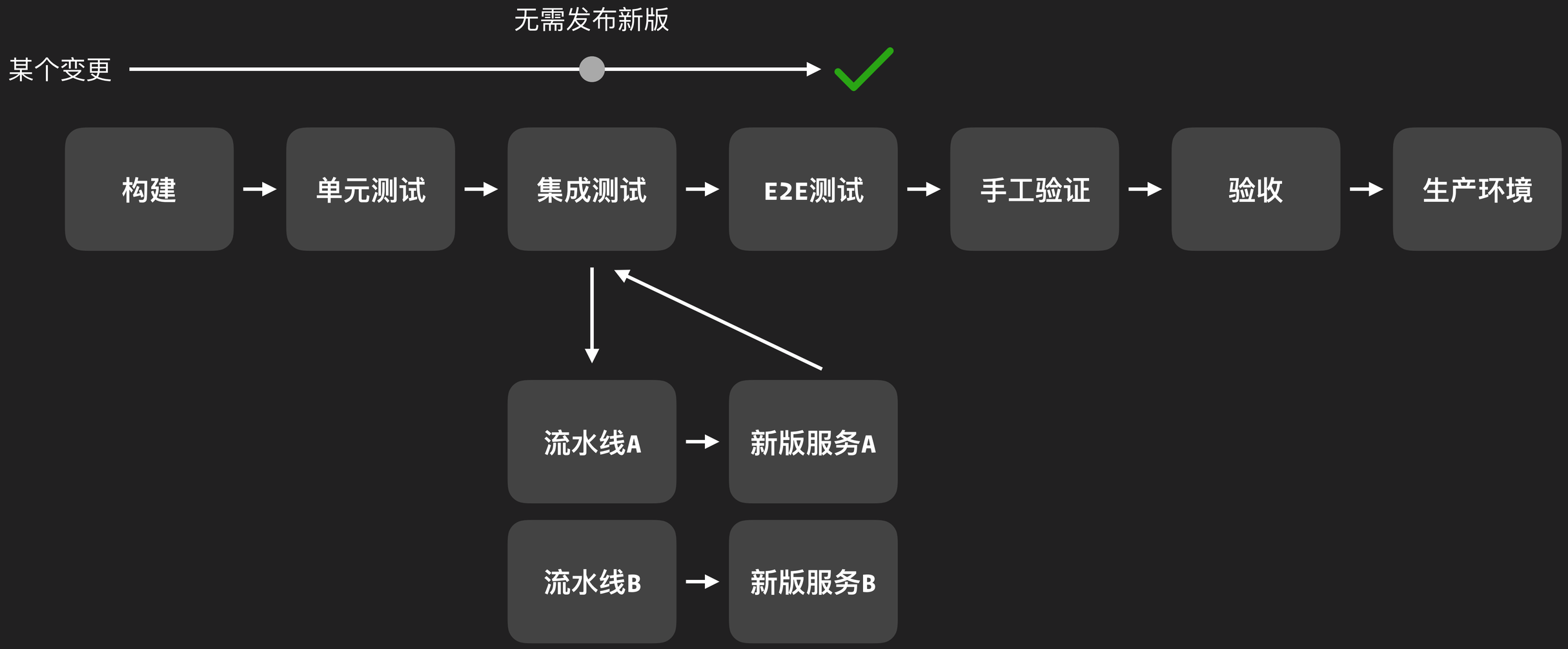
- 每个变更都能直接发布

- 每个变更都从流水线起点开始
- 主干流水线错误时立即修复

多流水线耦合

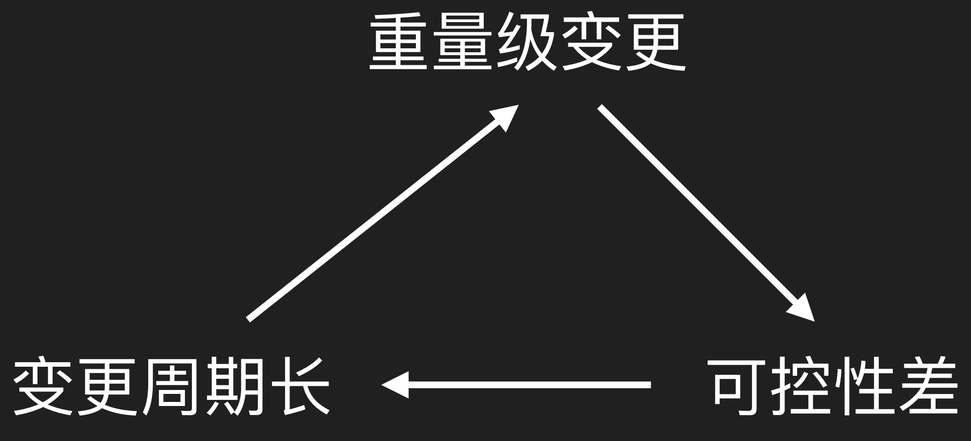


解耦流水线

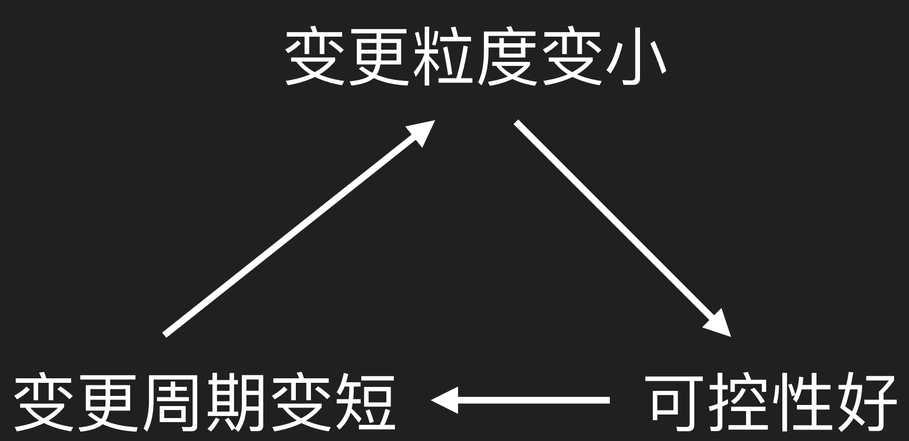

无需发布新版
某个变更
构建
单元测试
集成测试
E2E测试
手工验证
验收
生产环境
流水线A
新版服务A
流水线B
新版服务B

变更管理

重量级、间隔很长的变更



小粒度、频繁的变更



自动化的工作流

Git 提交

- Lint
- 检查提交格式

Merge Request

- 触发测试
- 覆盖率报告

构建

- 各种测试
- 生成发布日志
- 上传发布包

上线部署

- 上线知会
- 热切换

理想

开发

写代码

快速、自动地部署测试环境

Code Review

测试

写测试用例

单元测试、集成测试、E2E测试

测试及覆盖率报告

部署

自动化

轻量化

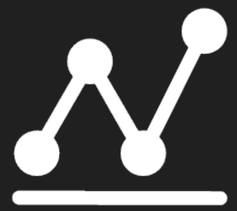
持续部署

上线

自动构建软件制品

流量无缝切换

自动化的监控告警



日志监控篇

When 什么时候需要打日志

What 日志需要什么信息

How 怎么打日志

Where 日志输出到哪

日志的分类

业务日志

调用量

PV / UV

用户操作

...

链路日志

后端微服务I/O

自身I/O

...

运维日志

CPU

内存

进程状态

网络吞吐

...

错误日志

启动

警告

错误

崩溃

...

日志内容

请求相关

请求ID

请求/响应时间

请求/响应内容

代码相关

文件名、行号

日志类别

错误栈

用户相关

Uin

UserAgent

Cookie

环境相关

服务器IP

进程PID

日志输出到哪里

本地文件

文件托管

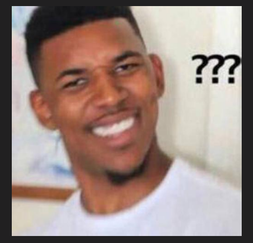
日志系统

ELK集群

前端同事：“你们这个服务性能很差啊，每次请求都要好几秒。”

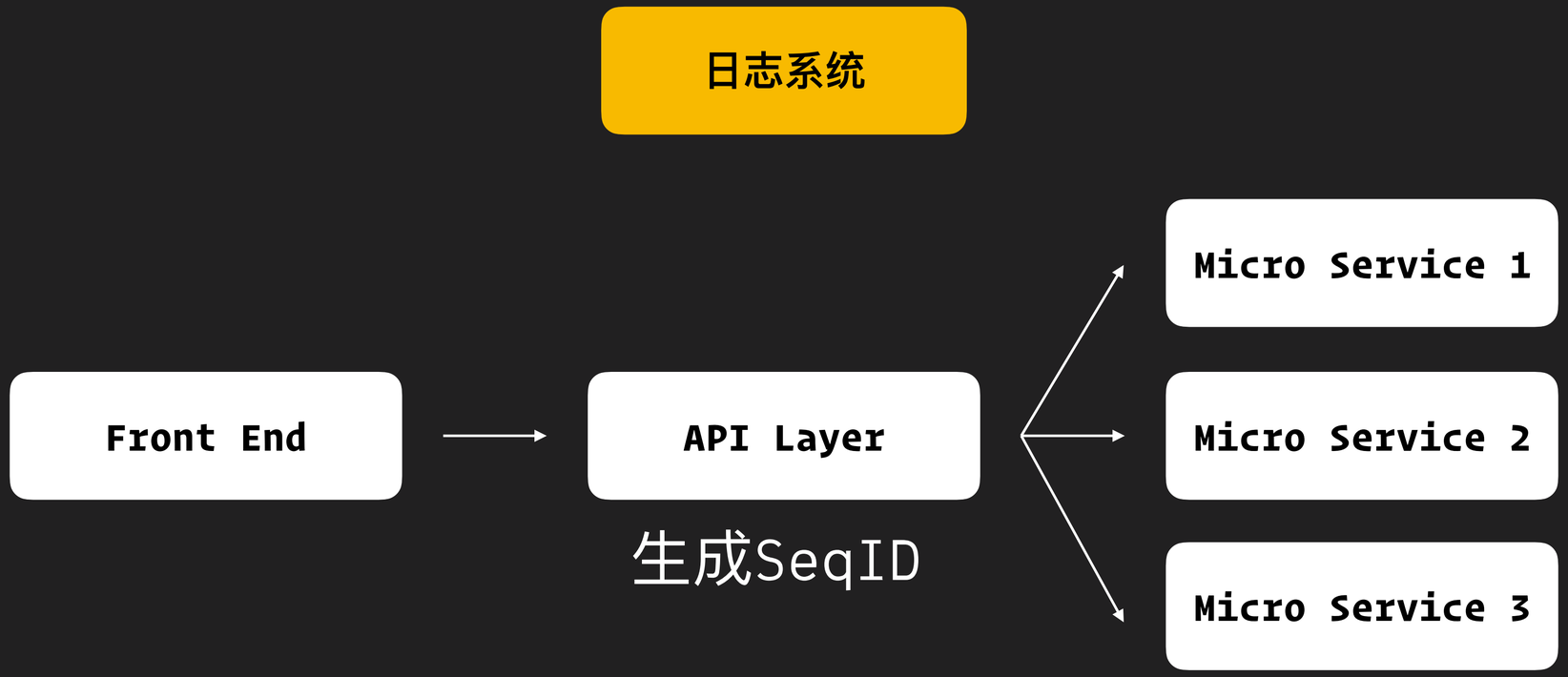
甩锅的DB：“我日志里平均耗时还不到10ms。”

甩锅的运维：“我这LB耗时都不到1ms。”

迷茫的你：“ 

”

全链路日志



SeqID有啥用：全链路性能分析、错误定位、...

SeqID放在哪：HTTP Header / Protobuf / …

哪里需要SeqID：错误日志、关键路径、...

日志上报@CloudBase

监控告警

对依赖服务的监控

返回码

耗时

对服务质量的监控

返回码、耗时

代码错误

关键路径失败

对业务的监控

重要用户的行为路径

业务数据波动

告警策略

以耗时监控为例

指标	告警阈值
均值	< 50ms
慢请求比例	< 1%
方差波动	< 5
均值24小时同比波动	< 100%
超过200ms的数量	< 1000（每10秒）
超过200ms的独立用户数	< 200（每10秒）

在2019-03-24 19:31:55 ~ 2019-03-24 19:32:55的1分钟之内，关键词“logType: mongoResponse AND mongoReturnCode: "InternalError"”在index tcb_baas_service_log-*中实际出现125次，大于其【事件数】告警阈值5。

MongoDB内部错误

在2019-03-25 09:45:55 ~ 2019-03-25 09:47:55的2分钟之内，使用关键词“code:* AND action:storage.*”搜索index tcb_baas_service_log-*的结果集中，字段“appId”的独立值总共有21次，大于其【字段统计】告警阈值20。

文件存储服务有问题

在2019-03-24 20:29:55 ~ 2019-03-24 20:30:55的1分钟之内，关键词“interfaceName:invokeFunction AND networkTimeCost:>300”在index tcb_baas_service_log-*中实际出现305次，大于其【事件数】告警阈值300。

Serverless函数延迟很大

总结

Summary

基础设施

- 面临的挑战
- 基础设施即代码

持续集成

- 版本控制
- 流水线
- 变更管理

质量管理

- 三大测试
- 测试金字塔
- 团队协作

日志监控

- 日志基础
- 全链路日志
- 监控上报
- 告警策略

Q / A

Github@starkwang

知乎@Starkwang