AWS云上混沌工程实践之对照实验设计和实施

黄帅

AWS 资深云架构咨询顾问

极客时间 | 企业服务

这是一封正经“福利”贴

想做团队的领跑者 需要迈过这些“槛”

成长型企业，易忽视人才体系化培养 企业转型加快，团队能力又跟不上	VS	从基础到进阶，超100+一线实战 技术专家带你系统化学习成长
团队成员技能水平不一， 难以一“敌”百人需求	VS	解决从小白到资深技术人所遇到 80%的问题
寻求外部培训，奈何价更高且 集中式学习	VS	多样、灵活的学习方式，包括 音频、图文 和视频
学习效果难以统计，产生不良循环	VS	获取员工学习报告，查看学习 进度，形成闭环

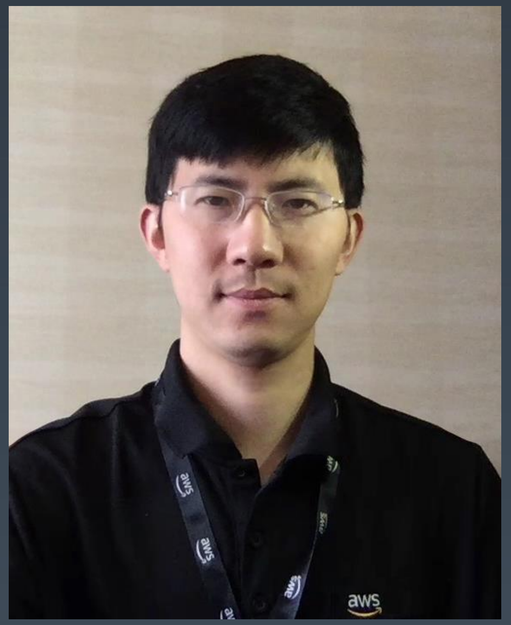
课程顾问「橘子」

回复「QCon」
免费获取
学习解决方案

 # 极客时间企业账号 # 解决技术人成长路上的学习问题

自我介绍

- 黄帅 [Henry Huang] - 云架构咨询顾问
- 来自亚马逊 AWS 专业服务团队
- 负责企业级客户的云架构设计和优化
- 近来专注于混沌工程实验设计和落地实施



@henrysher

目录

混沌工程发展历程

天花

- 天花是一种由天花病毒引起之人类传染病
- 天花主要透过空气传播
- 最早的天花在公元前3世纪埃及木乃伊中发现（已死去逾三千年的法老拉美西斯五世）
- 十八世纪的欧洲，估计每年有40万人死亡

接种与疫苗

最早出现的天花预防法为接种，成功接种的人可建立持久的免疫力，因患上天花而死的机会亦会降低；若失败，接种者会染上天花，并可能将之散播。

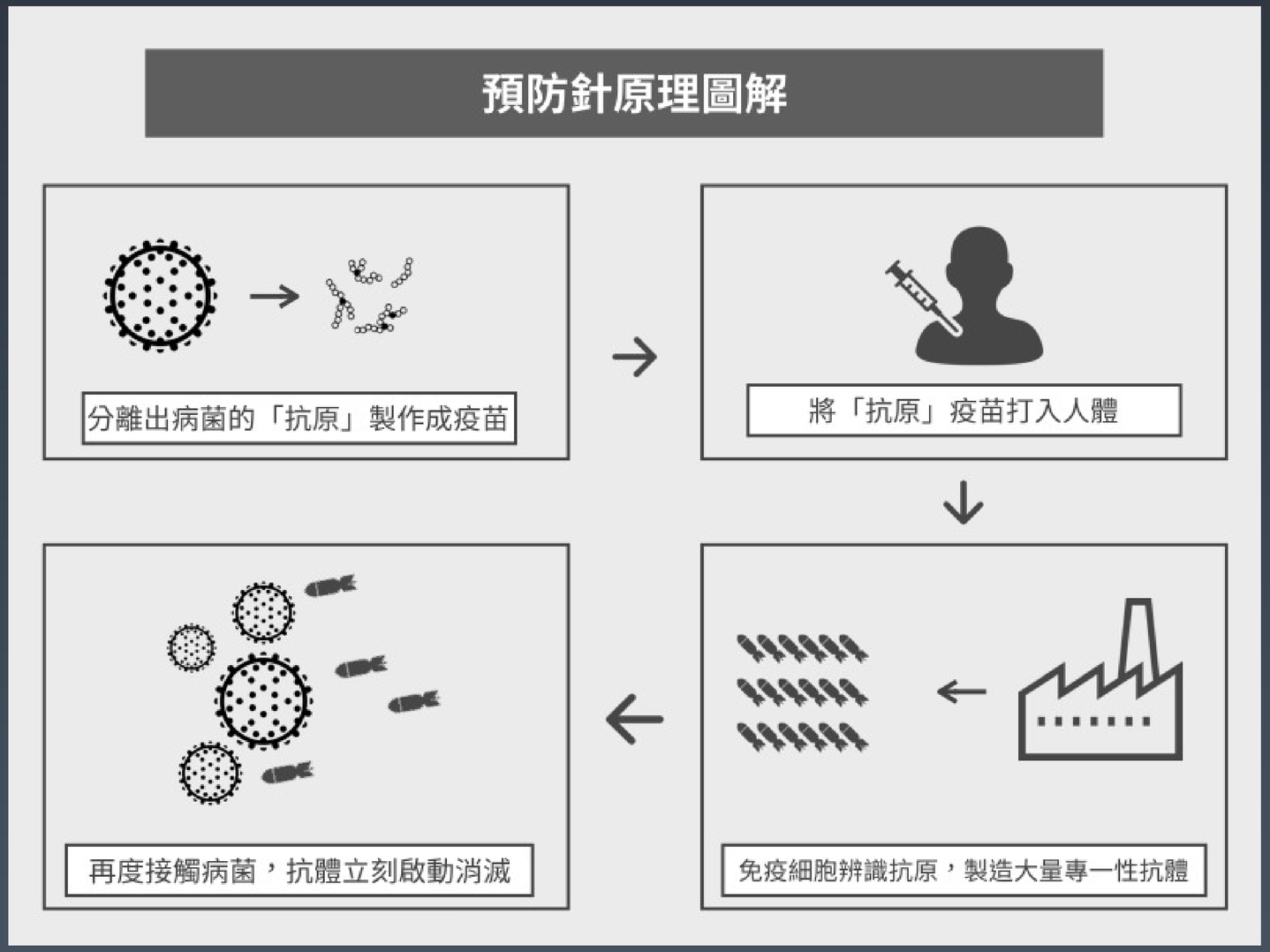
居住在英格兰郊区的爱德华·詹纳医生发现了牛痘（一种对人类较为温和的痘病毒）能用以预防天花。

世界卫生组织于1980年正式宣布扑灭天花，使之成为首个于世上绝迹的人类传染病。

https://en.wikipedia.org/wiki/Edward_Jenner

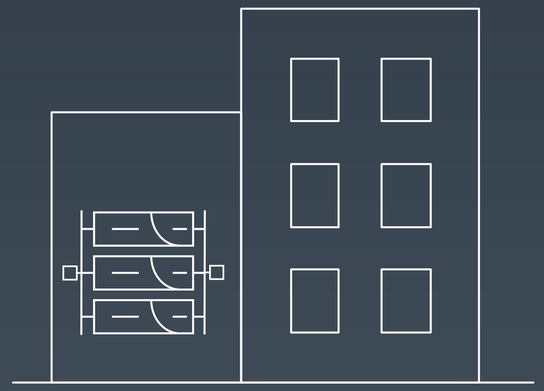
疫苗与混沌工程

- 受控实验，准备故障注入方式（疫苗）
- 将故障（抗原）注入（人体）系统
- 了解（人体）系统抵御故障（抗原）的行为（产生抗体）
- 真实故障（病菌）发生时，已建立抵御故障（病菌）的能力和信心（消灭）



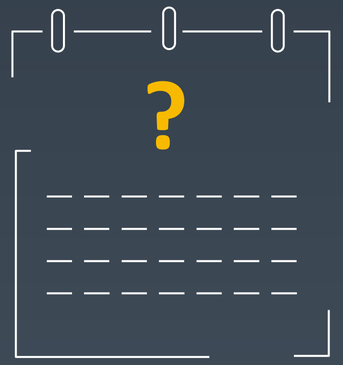

过去
当前
未来

过去 当前 未来

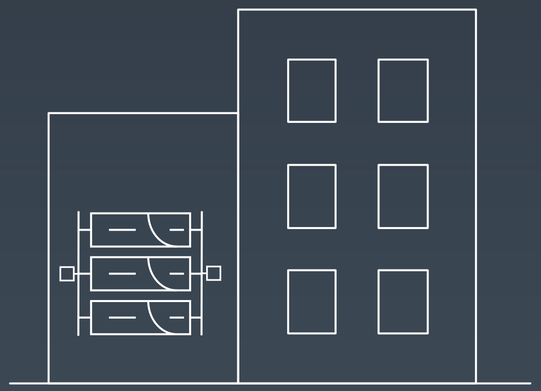


灾备技术

混沌工程



韧性系统



灾备技术

RPO

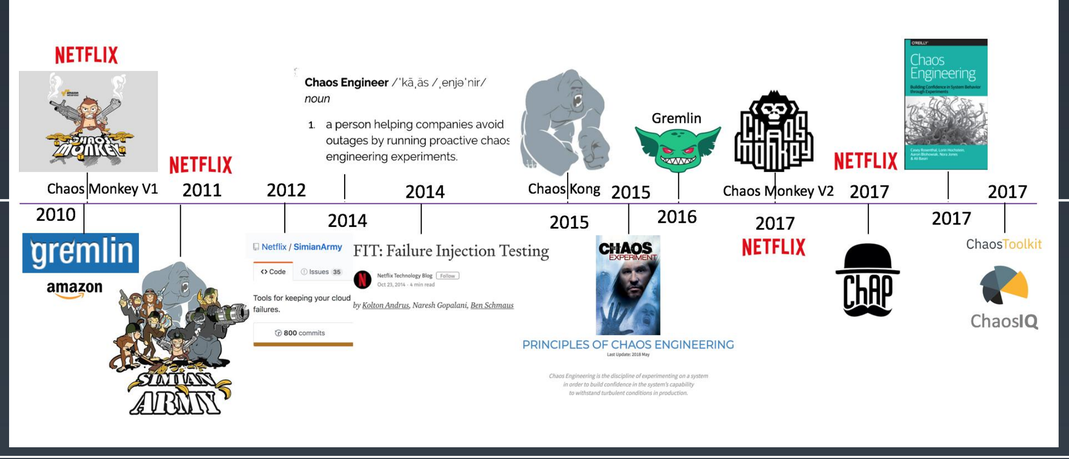
Recovery Point Objective

即恢复时间目标，主要指当发生灾难或紧急事件时，业务系统所能容忍的停止服务的最长时间，也就是从灾难发生到业务系统恢复服务功能所需要的最短时间周期。

RTO

Recovery Time Objective

即数据恢复点目标，主要指当发生灾难或紧急事件时，业务系统所能容忍的数据丢失量。



混沌工程

- 2010年 Kolton Andrus 在 Amazon 内部开发了可用性测试工具 Gremlin
- 2010年 Netflix 内部开发了 AWS 云上随机终止 EC2 实例的混沌实验工具 Chaos Monkey
- 2011年 Netflix 释出了其猴子军团工具集 Simian Army
- 2012年 Netflix 向社区开源由 Java 构建 Simian Army, 其中包括 Chaos Monkey V1 版本
- 2014年 Netflix 开始正式公开招聘 Chaos Engineer
- 2014年 Netflix 提出了故障注入测试 FIT, 利用微服务架构的特性，控制混沌实验的爆炸半径
- 2015年 Netflix 释出 Chaos Kong, 模拟AWS区域中断的场景
- 2015年 Netflix 和社区正式提出混沌工程的指导思想 – Principles of Chaos Engineering
- 2016年 Kolton Andrus 创立了 Gremlin, 正式将混沌实验工具商用化
- 2017年 Netflix 开源 Chaos Monkey 由 Golang 重构的 V2 版本，集成 CD 工具 Spinnaker
- 2017年 Netflix 释出 ChAP 混沌实验自动平台, 可视为应用故障注入测试 FIT 的加强版
- 2017年 由 Netflix 前混沌工程师撰写的新书 “混沌工程” 在网上出版
- 2017年 Russell Miles 创立了 ChaosIQ 公司, 并开源了 chaostoolkit 混沌实验框架

Hi, Tyler! | 0 Credits Available | Help |

audible an amazon company

Home Library Wish List Browse Listener Page Blog

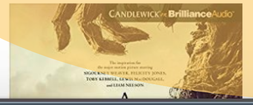
Search for a great book

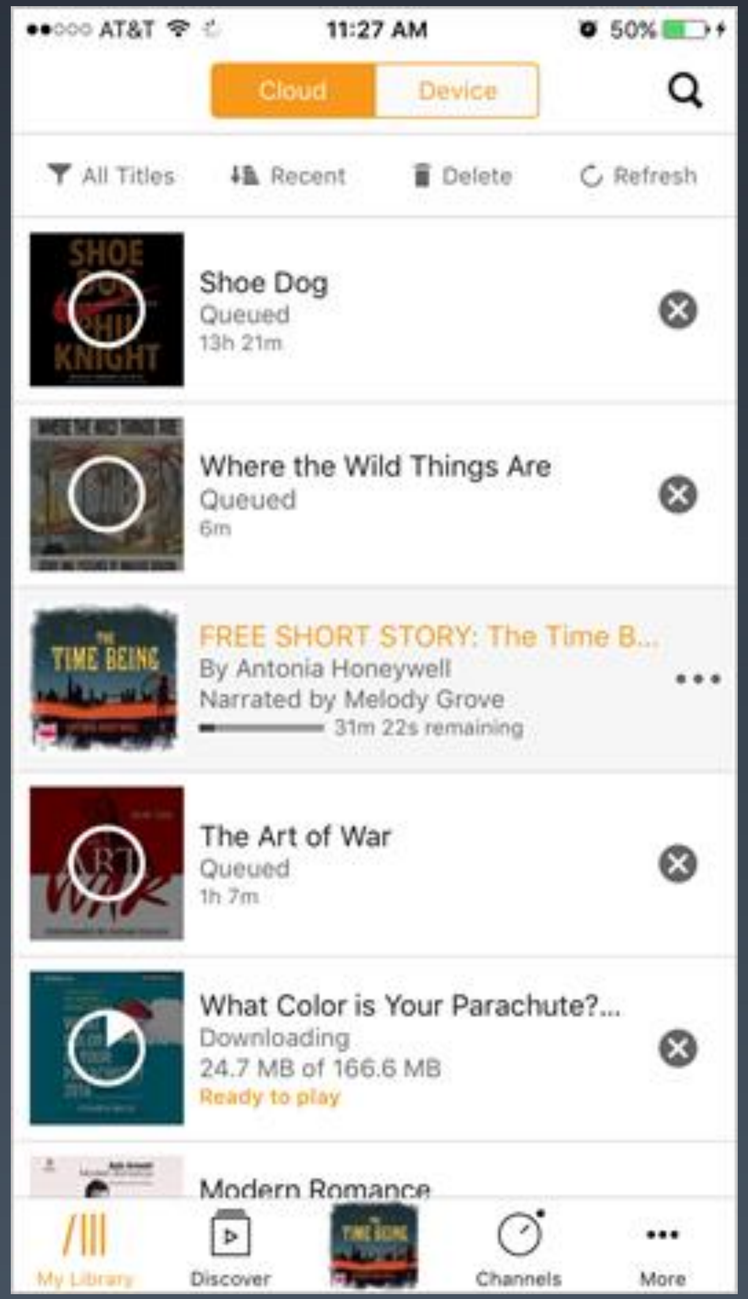
Advanced Search

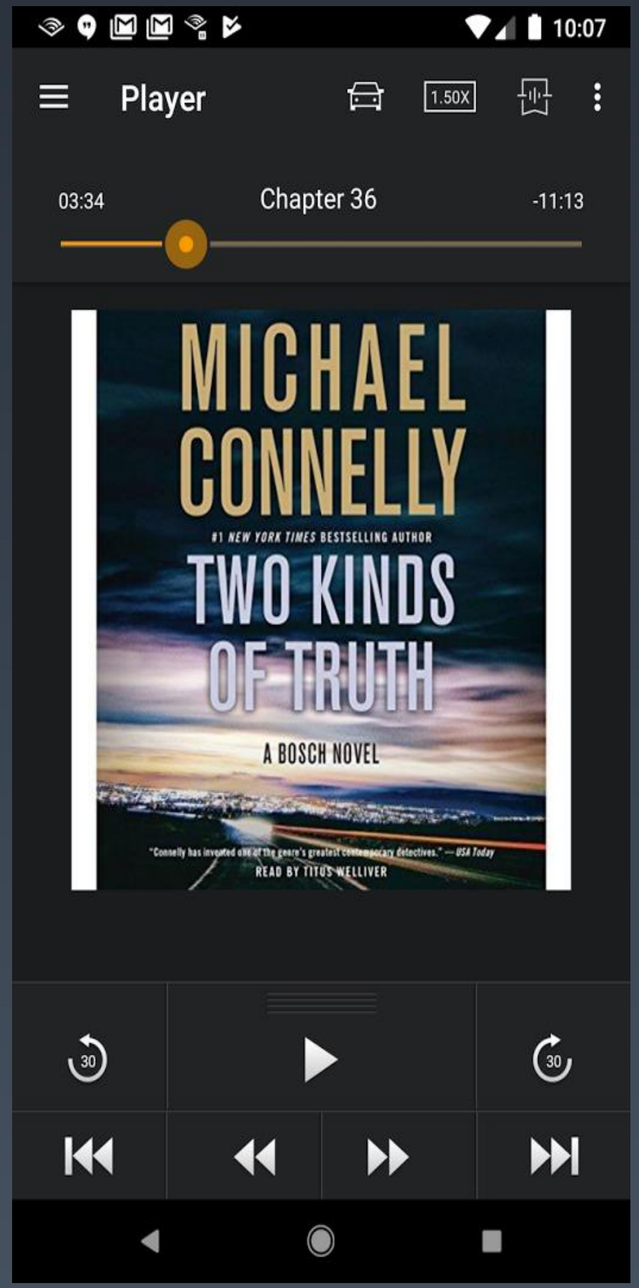
AUTHOR'S PICKS

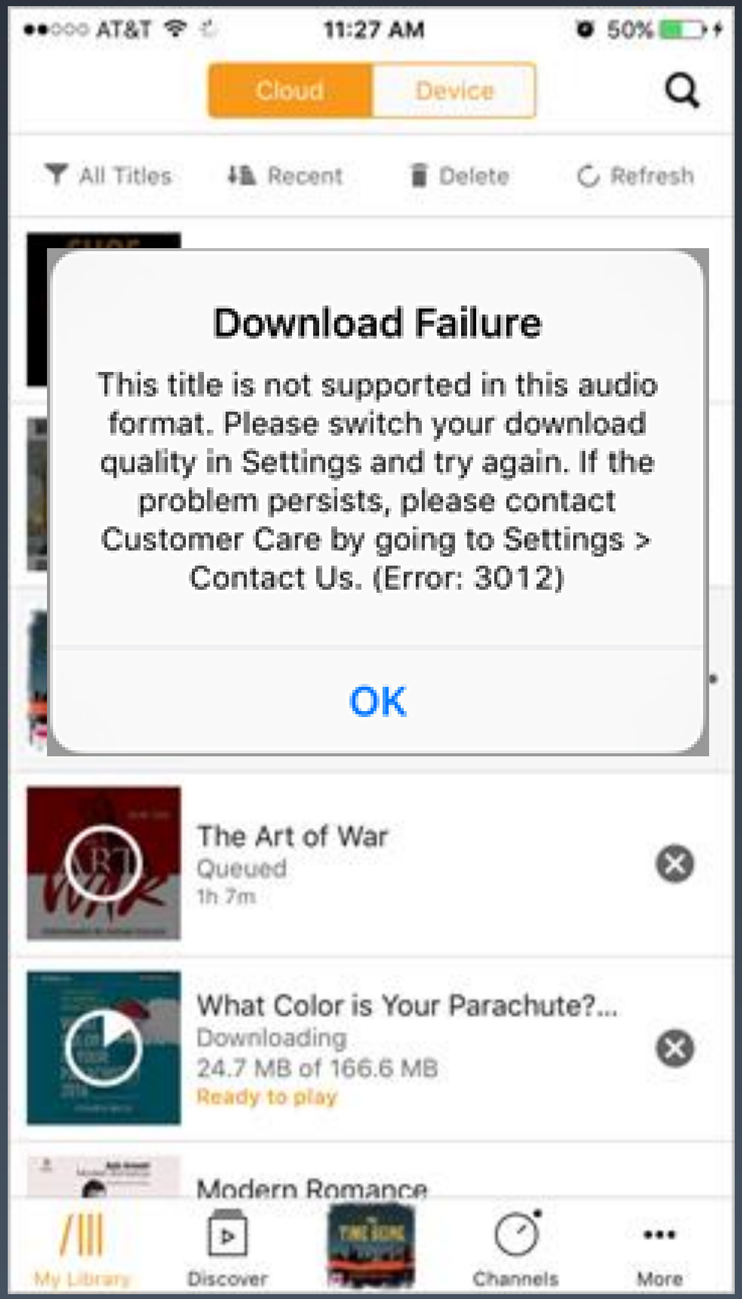
Award-winning children's and young adult author Julie Berry shares the audiobooks that have moved her the most.

Julie Berry; author of *Wishes and Wellingtons*



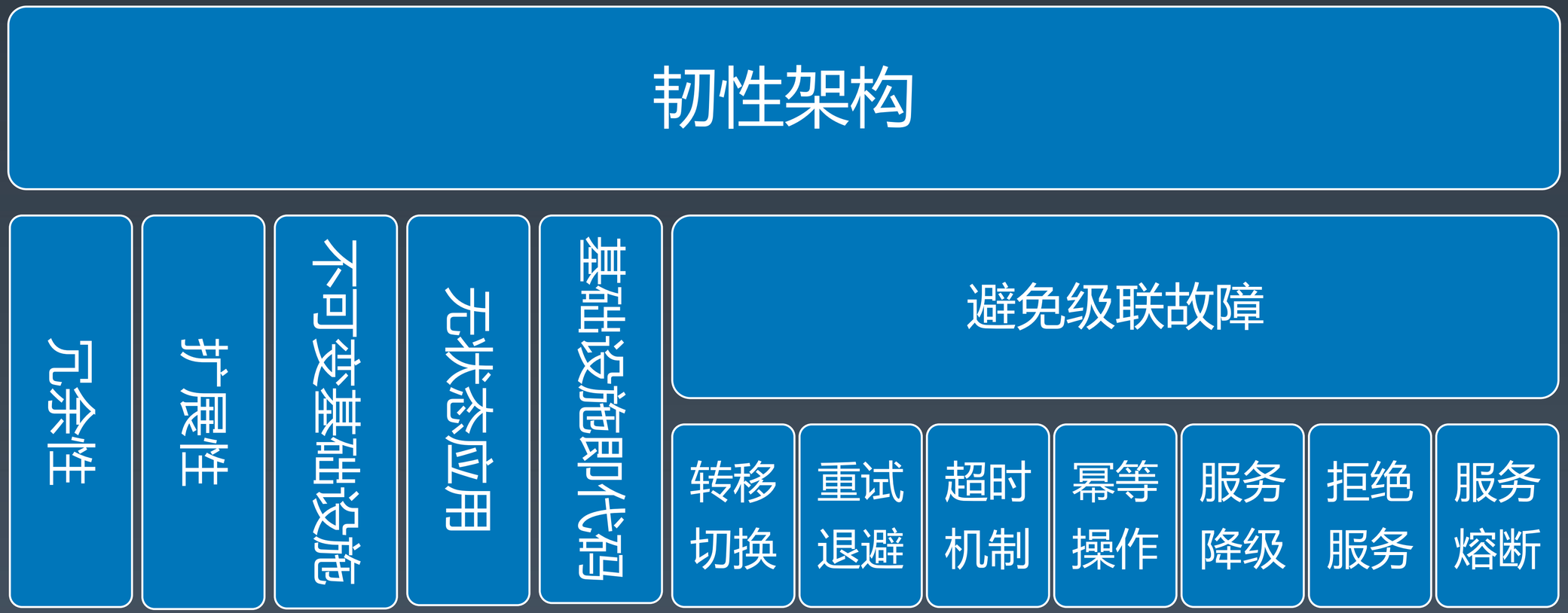

AT&T
11:27 AM
50%
Cloud
Device
All Titles
Recent
Delete
Refresh
Shoe Dog
Queued
13h 21m
Where the Wild Things Are
Queued
6m
FREE SHORT STORY: The Time B...
By Antonia Honeywell
Narrated by Melody Grove
31m 22s remaining
The Art of War
Queued
1h 7m
What Color is Your Parachute?...
Downloading
24.7 MB of 166.6 MB
Ready to play
Modern Romance
My Library
Discover
Channels
More

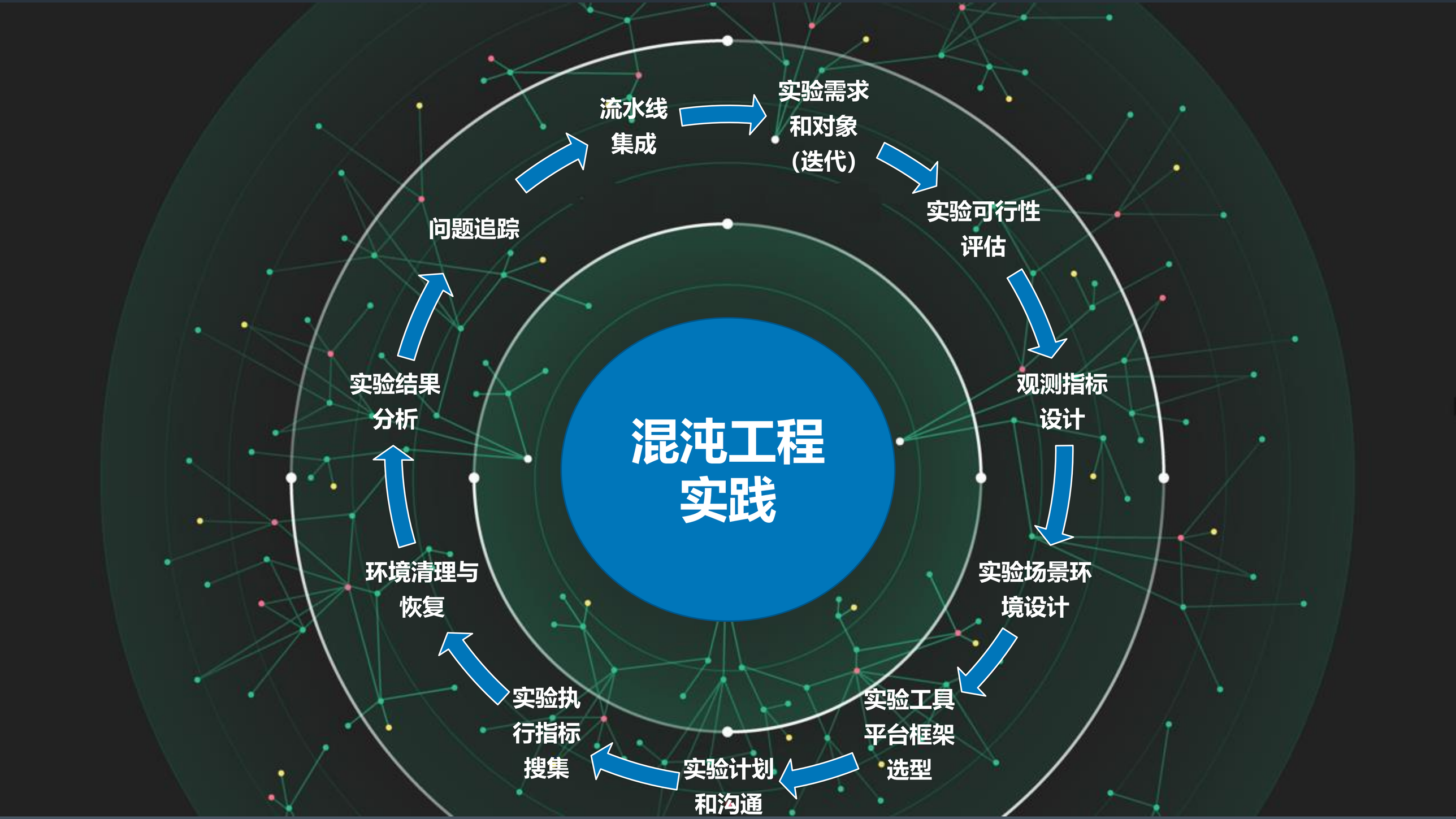

10:07
Player
1.50X
03:34
Chapter 36
-11:13
MICHAEL CONNELLY
#1 NEW YORK TIMES BESTSELLING AUTHOR
TWO KINDS OF TRUTH
A BOSCH NOVEL
READ BY TITUS WELLIVER
30
30


AT&T
11:27 AM
50%
Cloud
Device
All Titles
Recent
Delete
Refresh
Download Failure
This title is not supported in this audio format. Please switch your download quality in Settings and try again. If the problem persists, please contact Customer Care by going to Settings > Contact Us. (Error: 3012)
OK
The Art of War
Queued
1h 7m
What Color is Your Parachute?...
Downloading
24.7 MB of 166.6 MB
Ready to play
Modern Romance
My Library
Discover
Channels
More

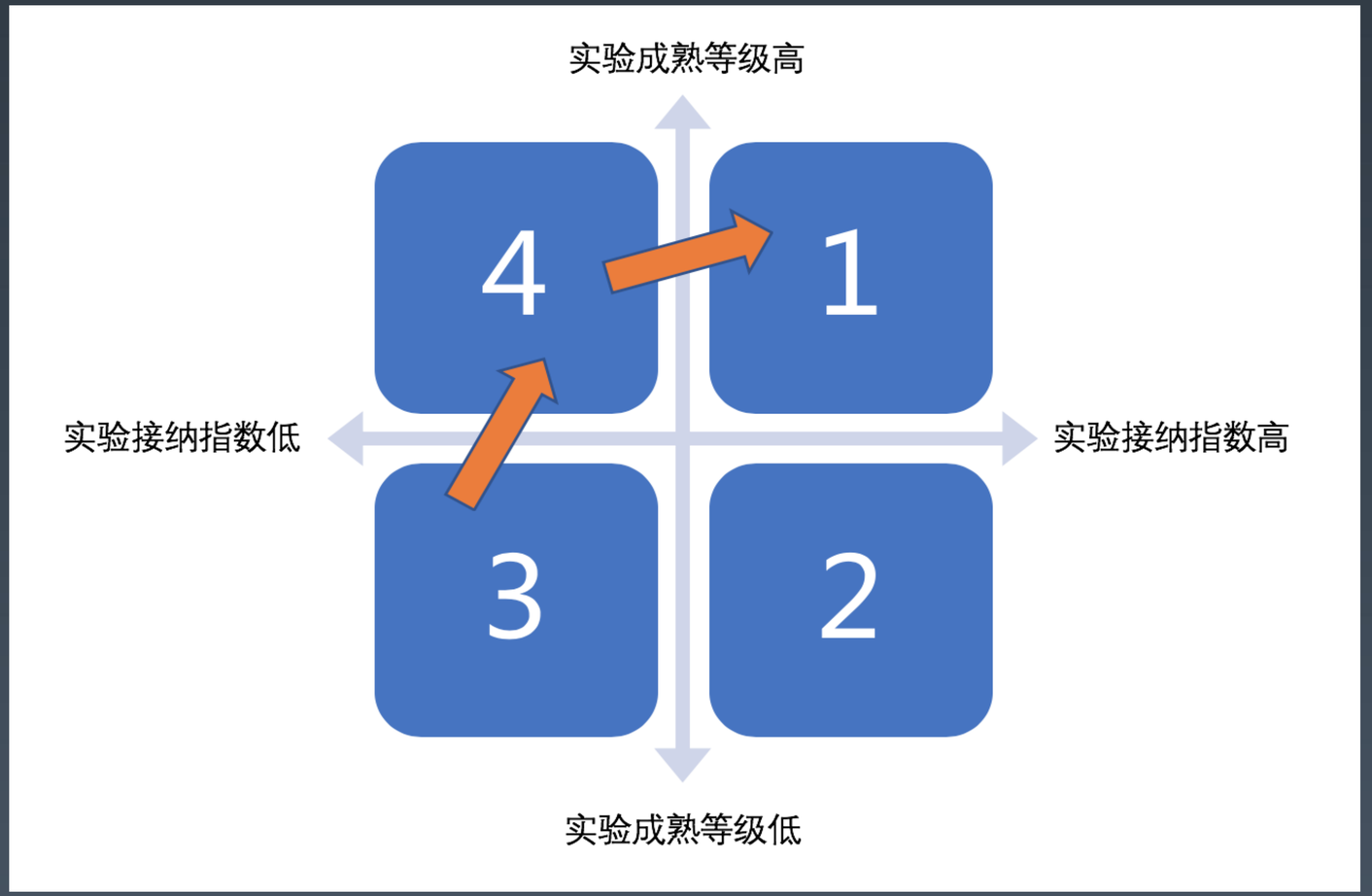
混沌工程实验的设计方法

混沌工程实验目标


韧性架构
冗余性
扩展性
不可变基础设施
无状态应用
基础设施即代码
避免级联故障
转移切换
重试退避
超时机制
幂等操作
服务降级
拒绝服务
服务熔断


混沌工程
实践
实验需求
和对象
（迭代）
实验可行性
评估
观测指标
设计
实验场景环
境设计
实验工具
平台框架
选型
实验计划
和沟通
实验执
行指标
搜集
环境清理与
恢复
实验结果
分析
问题追踪
流水线
集成

混沌工程可行性评估模型



混沌工程实验成熟度等级

成熟度等级	1级	2级	3级	4级	5级
架构抵御故障的能力	无抵御故障的能力	一定的冗余性	冗余且可扩展	已使用可避免级联故障的技术	已实现韧性架构
实验指标设计	无系统指标监控	实验结果只反映系统状态指标	实验结果反映应用的健康状况指标	实验结果反映聚合的业务指标	可在实验组和控制组之间比较业务指标的差异
实验环境选择	只敢在开发和测试环境中运行实验	可在预生产环境中运行实验	未在生产环境中，用复制的生产流量来运行实验	在生产环境中运行实验	包括生产在内的任意环境都可以运行实验
实验自动化能力	全人工流程	利用工具进行半自动运行实验	自助式创建实验，自动运行实验，但需要手动监控和停止实验	自动结果分析，自动终止实验	全自动的设计、执行和终止实验
实验工具使用	无实验工具	采用实验工具	使用实验框架	实验框架和持续发布工具集成	工具支持交互式的比对实验组和控制组
故障注入场景爆炸半径范围	只对实验对象注入一些简单事件，如突发高CPU高内存等等	可对实验对象进行一些较复杂的故障注入，如EC2实例终止、可用区故障等等	对实验对象注入较高级的事件，如网络延迟	对实验组引入如服务级别的影响和组合式的故障事件	可注入如对系统的不同使用模式、返回结果和状态的更改等类型的事件
环境恢复能力	无法恢复正常环境	可手动恢复环境	可半自动恢复环境	部分可自动恢复环境	韧性架构自动恢复
实验结果整理	没有生成的实验结果，需要人工整理判断	可通过实验工具的到实验结果，需要人工整理、分析和解读	可通过实验工具持续收集实验结果，但需要人工分析和解读	可通过实验工具持续收集实验结果和报告，并完成简单的故障原因分析	实验结果可预测收入损失、容量规划、区分出不同服务实际的关键程度

混沌工程实验接纳指数

接纳指数	描述
1级	公司重点项目不会进行混沌工程实验； 只覆盖了少量的系统； 公司内部基本上对混沌工程实验了解甚少； 极少数工程师尝试且偶尔进行混沌工程实验。
2级	混沌工程实验获得正式授权和批准； 由工程师兼职进行混沌工程实验； 公司内部有多个项目有兴趣参与混沌工程实验； 极少数重要系统会不定期进行混沌工程实验。
3级	成立了专门的混沌工程团队； 事件响应已经集成在混沌工程实验框架中以创建对应的回归实验； 大多数核心系统都会定期进行混沌工程实验； 偶尔以Game Day的形式，对实验中发现的故障进行复盘验证。
4级	公司所有核心系统都会经常进行混沌工程实验； 大多数非核心系统也都会经常进行混沌工程实验； 混沌工程实验是工程师日常工作的一部分； 所有系统默认都要参与混沌工程实验，不参与需要特殊说明。

具体可行性评估问卷

回答可行性评估问题表（基于实验成熟度等级的8个方面）

- 架构抵御故障的能力
- 实验指标设计
- 实验环境选择
- 实验自动化能力
- 实验工具使用
- 故障注入场景（爆炸半径范围）
- 环境恢复能力
- 实验结果整理

观测指标的设计

- 指标类型

  业务性指标：价值最大，探测难度最大 – Netflix SPS 业务指标

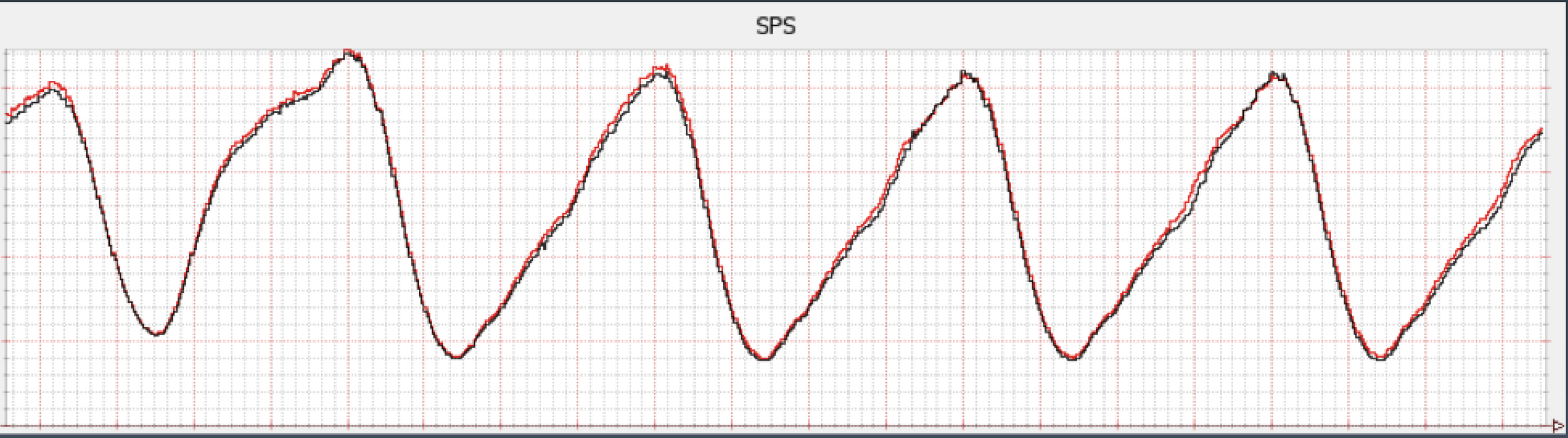
  应用层指标：健康状况

  基础设施指标：较易获取，只反映基础设施的运行状况
- 指标对照：

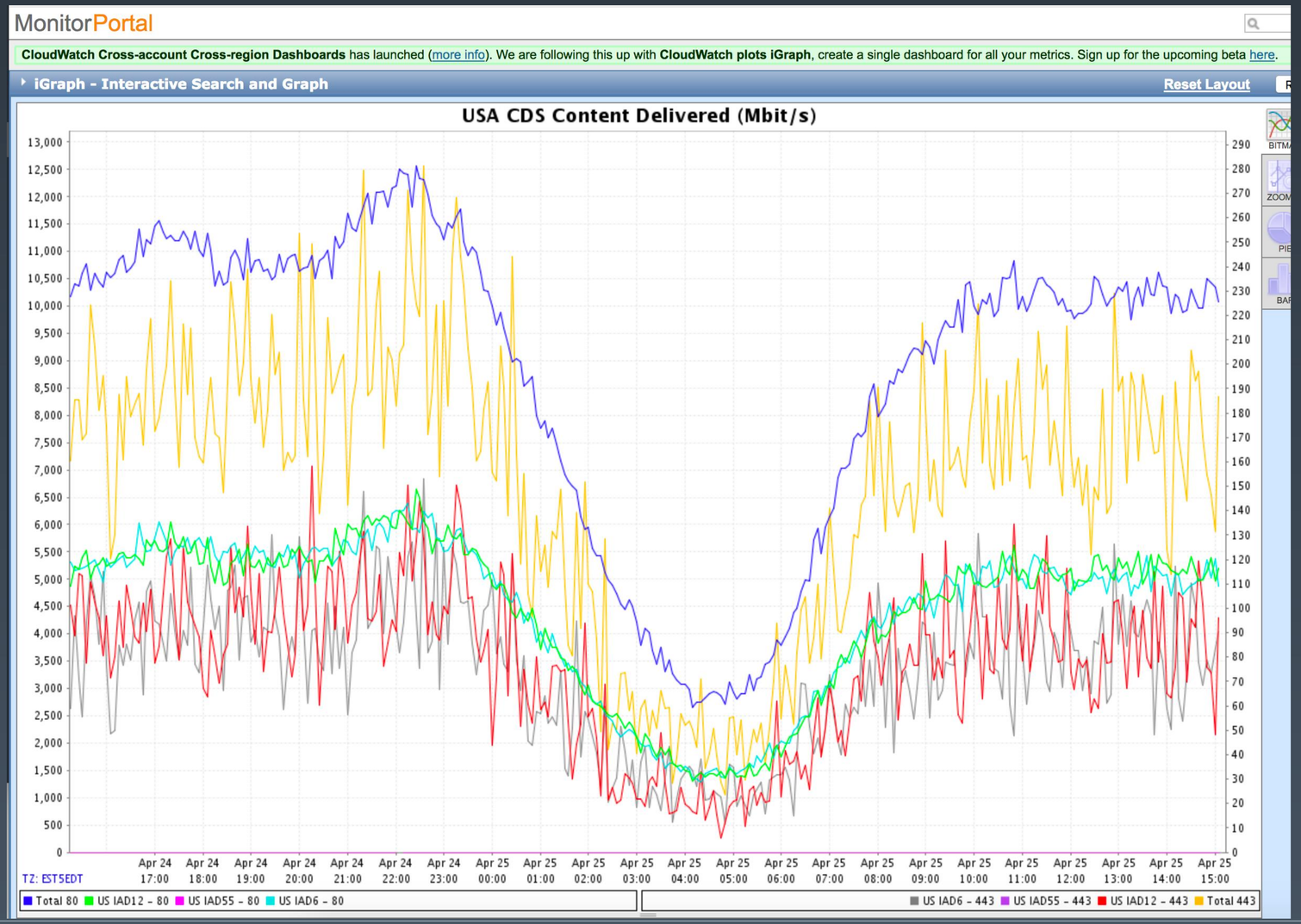
  应定义指标的稳定状态进行对照

  如无法准确定义稳定状态，则使用多个指标的阈值联合进行对照
- 指标观测和记录系统

SPS: Netflix关键业务指标

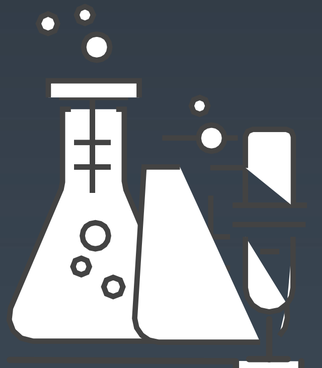


Audible 关键业务指标



故障注入场景设计

故障注入场景	具体描述
依赖型故障	AWS托管服务异常：ELB / S3 / DynamoDB / …
主机型故障	EC2实例异常终止，EC2实例异常关闭，EBS磁盘卷异常卸载，RDS数据库实例故障切换，ElastiCache实例故障切换，…
操作系统内故障	CPU、内存、磁盘空间、IOPS 占满或突发过高占用，大文件，只读系统，系统重启，熵耗散，…
网络故障	网络延迟，网络丢包，DNS解析故障，DNS缓存毒化，VIP移动，网络黑洞，…
服务层故障	不正常关闭连接，进程被杀死，暂停 / 启用进程，内核奔溃，…
请求拦截型故障	异常请求，请求处理延迟，…



故障注入对象

高

中

低

无服务器架构

微服务架构

调用链可追踪的单体架构

单体架构

实验工具的选择

工具名称	最新版本	项目维护状态	主要构建语言	涉及场景	特定依赖
ChaosMonkey	**2.0.2**	**2016年后不再功能开发**	**Go**	**终止 EC2 实例**	**依赖于Spinnaker**
SimianArmy	2.5.3	已退役	Java	终止EC2实例 关闭EC2实例 阻断网络流量 卸载磁盘卷 CPU/IO/磁盘空间突发过高 杀进程 路由失效 DNS失效 网络丢包 网络延迟 DynamoDB故障 …	无特定依赖

实验工具的选择

工具名称	最新版本	项目维护状态	主要构建语言	涉及场景	特定依赖
orchestrator	3.0.14	活跃	Go	纯MySQL集群故障场景	无特定依赖
kube-monkey	0.3.0	只发布了一个正式版本	Go	杀K8S Pods	针对K8S集群
chaostoolkit	1.1.0	活跃	Python	实验框架，可集成多个IaaS或PaaS平台，可使用多个故障注入工具定制场景，可与多个监控平台合作观测和记录指标信息	通过插件形式支持多个IaaS/PaaS，包括AWS/Azure/Google/K8S…
ChaoSlingr	无正式版本	项目停滞成只读状态	Python	安全注入试验框架，目前只支持端口变更场景	依赖于AWS Lambda
PowerfulSeal	2.0.1	活跃	Python	杀K8S Pods 杀容器 杀虚拟机	支持OpenStack/AWS/本地机器
drax	0.4.0	停滞	Go	杀Mesos任务	支持DC/OS

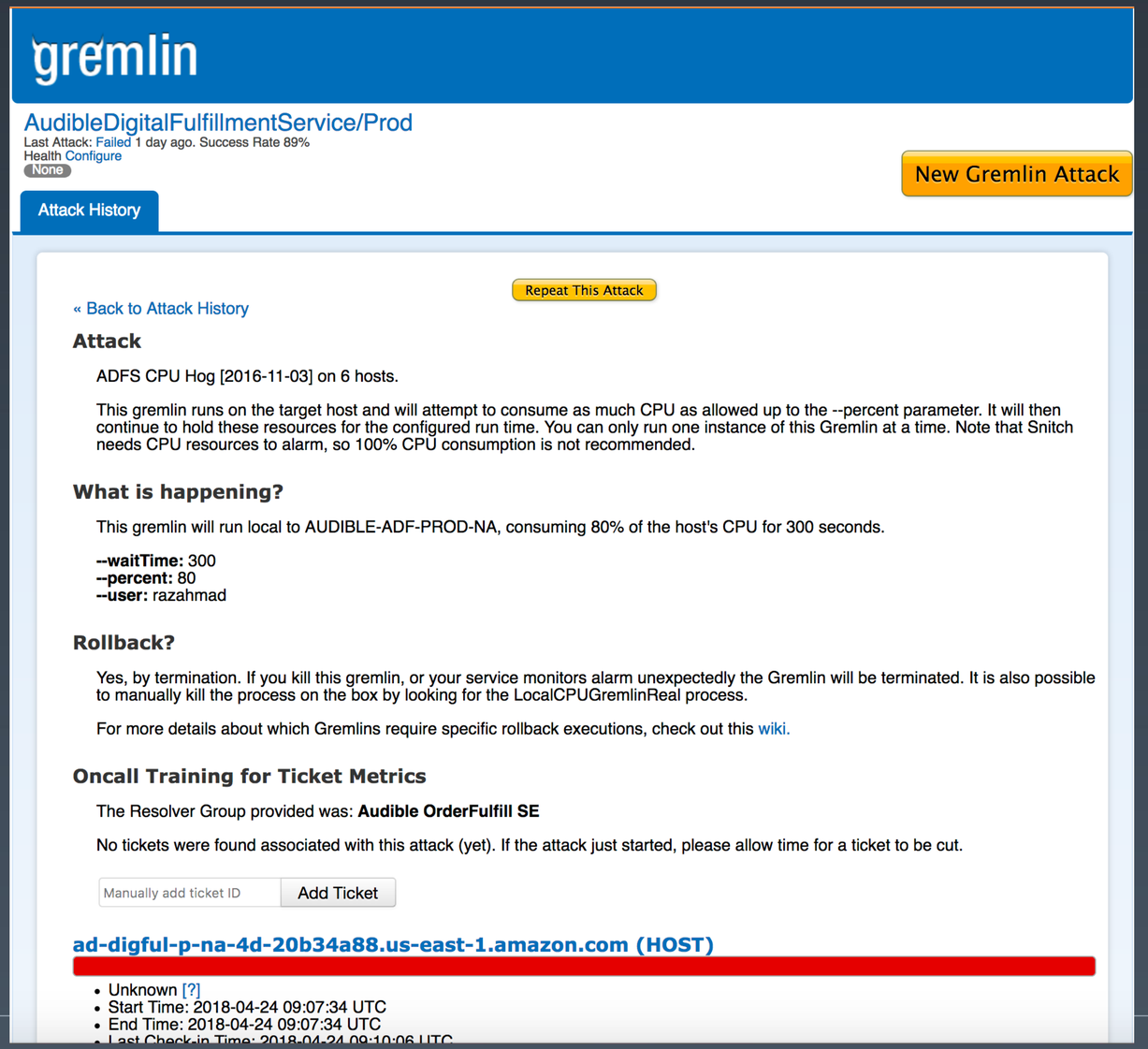
实验工具的选择

工具名称	最新版本	项目维护状态	主要构建语言	涉及场景	特定依赖
pod-reaper	2.3.0	停滞	Go	杀K8S Pods	针对K8S集群
muxy	0.0.6	停滞	Go	网络代理，模拟网络故障	无特定要求
toxiproxy	2.1.4	活跃	Go	网络代理，模拟网络故障	无特定要求
Pumba	0.6.3	活跃	Go	杀停删容器 暂停容器内进程 网络延迟 网络丢包 网络带宽限流	依赖于Docker
blockade	0.4.0	停滞	Python	杀容器 网络终端 网络延迟 网络丢包 网络分区	依赖于Docker
chaos-lambda	0.3.0	停滞	Python	杀自动伸缩组的EC2实例	依赖于Lambda

实验工具的选择

工具名称	最新版本	项目维护状态	主要构建语言	涉及场景	特定依赖
namazu	0.2.1	停滞	Go	文件系统故障 网络故障 Java功能调用故障	针对类Zookeeper分布式系统
chaos-monkey-spring-boot	2.0.2	活跃	Java	应用延迟 异常处理 内存过高	依赖于Spring Boot框架
byte-monkey	1.0.0	停滞	Java	异常处理 应用延迟	依赖于Java
GomJabbar	无正式版本发布	停滞	Java	优雅 / 粗暴关闭实例 无害远程执行命令 流量控制	依赖于RunDeck或Ansible
turbulence	0.8	停滞	Go	杀虚拟机 CPU/RAM/IO过高 网络分区 网络丢包 网络延迟	依赖于BOSH
chaosblade	0.0.2	活跃	Go	实验框架，目前支持JVM	无特定依赖

Gremlin自动化平台



Attack History | Attack!

Step 1 - Select a Gremlin

WARNING: Gremlins that add latency or packet loss can cause your hosts to reboot if they are running old Kernel versions. To fix this issue, you will have to repurpose your hosts.

Dependency Gremlins

Latent Dependency Gremlin

This Gremlin will add latency to incoming packets from any vips, IPs, or hosts in the list provided.

Select

Packet Loss Dependency Gremlin

This Gremlin will drop a configurable percentage of packets coming from or going to any vips, IPs, or hosts in the list provided.

Select

Hostclass Gremlins

Latent Hostclass Gremlin

This Gremlin will add latency to incoming packets from any hosts in the list of hostclasses provided. A common use of this Gremlin is to test how your service responds to increased latency from SABLE clusters.

Select

Packet Loss Hostclass Gremlin

This Gremlin will drop a configurable percentage of packets coming from or going to any hosts in the list of hostclasses provided. A common use of this Gremlin is to test how your service responds to SABLE being down.

Select

Network Gremlins

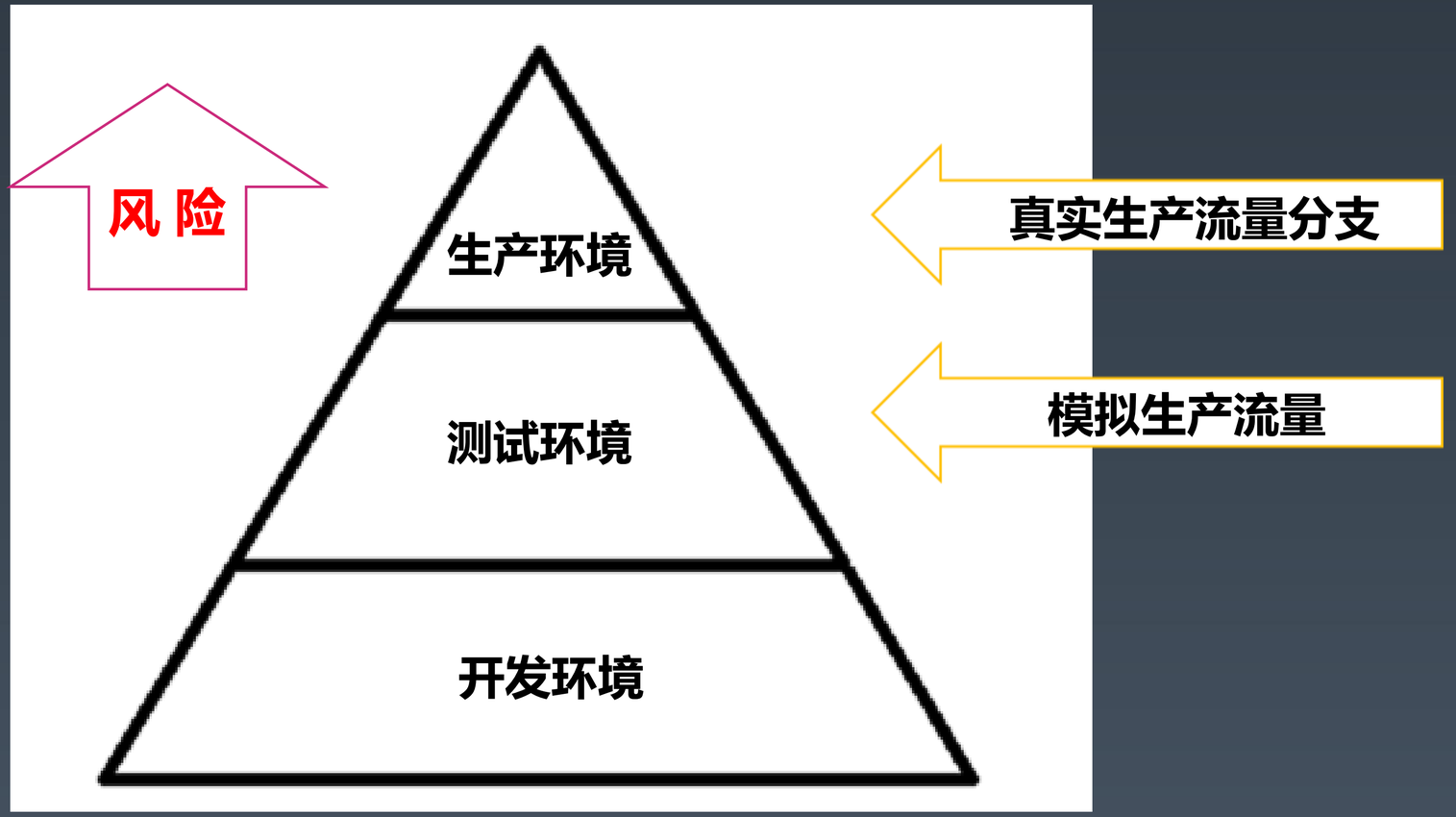
Latency Gremlin

This gremlin runs on the target hosts and introduces a set amount of latency into every incoming packet for the configured run time.

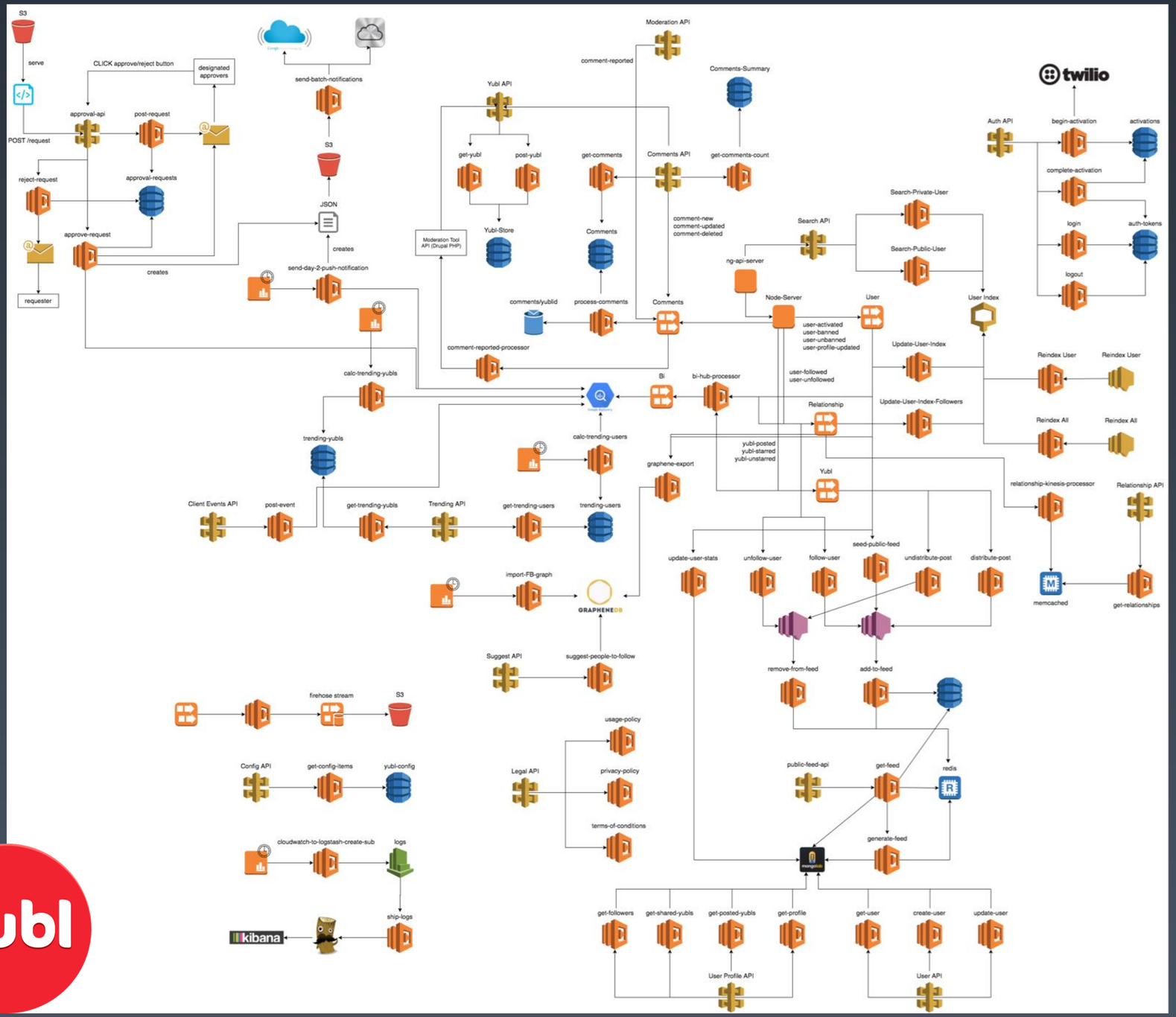
WARNING: If your hostclass has 'reboot host action' turned on with external monitoring, this gremlin can cause your hosts to reboot! Check your external monitoring configuration here. If your hosts still reboot, you may need to repurpose them due to a kernel bug.

Select

实验环境的设计和准备



混沌工程实验示例


S3
serve
CLICK approve/reject button
designated approvers
POST /request
approval-api
post-request
reject-request
approval-requests
approve-request
requester
creates
send-batch-notifications
S3
JSON
creates
send-day-2-push-notification
Moderation API
comment-reported
Comments-Summary
Yubl API
get-yubl
post-yubl
Yubl-Store
Moderation Tool API (Drupal PHP)
get-comments
Comments API
get-comments-count
Comments
comment-new
comment-updated
comment-deleted
comments/yublid
process-comments
Comments
comment-reported-processor
Search API
Search-Private-User
Search-Public-User
ng-api-server
Node-Server
User
User Index
user-activated
user-banned
user-unbanned
user-profile-updated
user-followed
user-unfollowed
Update-User-Index
Update-User-Index-Followers
Reindex User
Reindex User
Reindex All
Reindex All
twilio
Auth API
begin-activation
activations
complete-activation
login
auth-tokens
logout
calc-trending-yubls
trending-yubls
Bi
bi-hub-processor
Relationship
calc-trending-users
graphene-export
yubl-posted
yubl-starred
yubl-unstarred
Yubl
relationship-kinesis-processor
Relationship API
memcached
get-relationships
Client Events API
post-event
get-trending-yubls
Trending API
get-trending-users
trending-users
seed-public-feed
update-user-stats
unfollow-user
follow-user
undistribute-post
distribute-post
import-FB-graph
GRAPHENEDB
Suggest API
suggest-people-to-follow
remove-from-feed
add-to-feed
firehose stream
S3
usage-policy
Legal API
privacy-policy
terms-of-conditions
Config API
get-config-items
yubl-config
public-feed-api
get-feed
redis
generate-feed
cloudwatch-to-logstash-create-sub
logs
ship-logs
kibana
get-followers
get-shared-yubls
get-posted-yubls
get-profile
get-user
create-user
update-user
User Profile API
User API

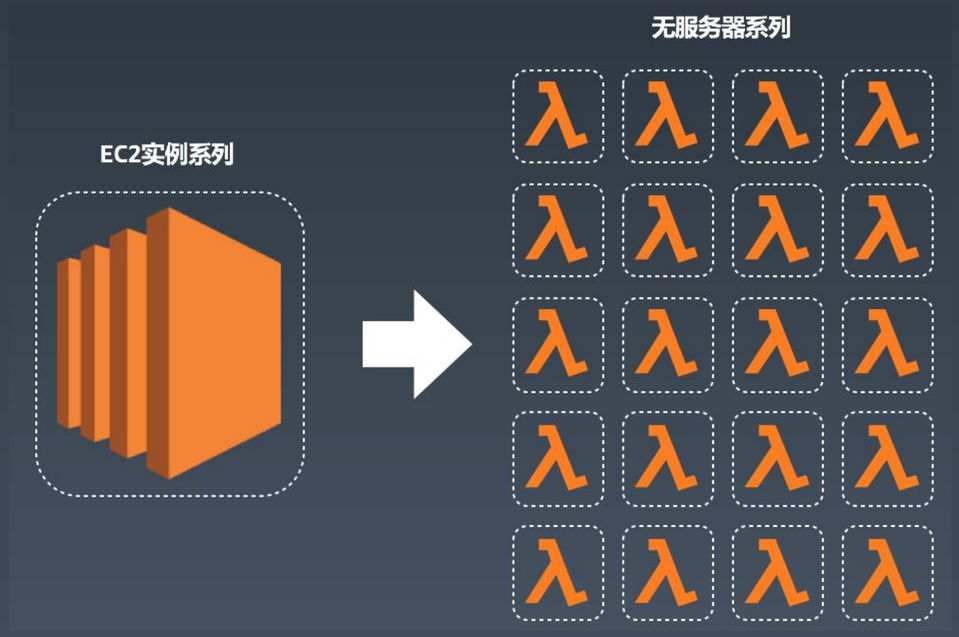
无服务架构的挑战

- 无服务器架构的函数数量庞大
- 每个函数都要正确配置和安全控制
- 无服务器架构引入大量的托管中间件服务

  每种托管的中间件服务都有自己特有的故障模式

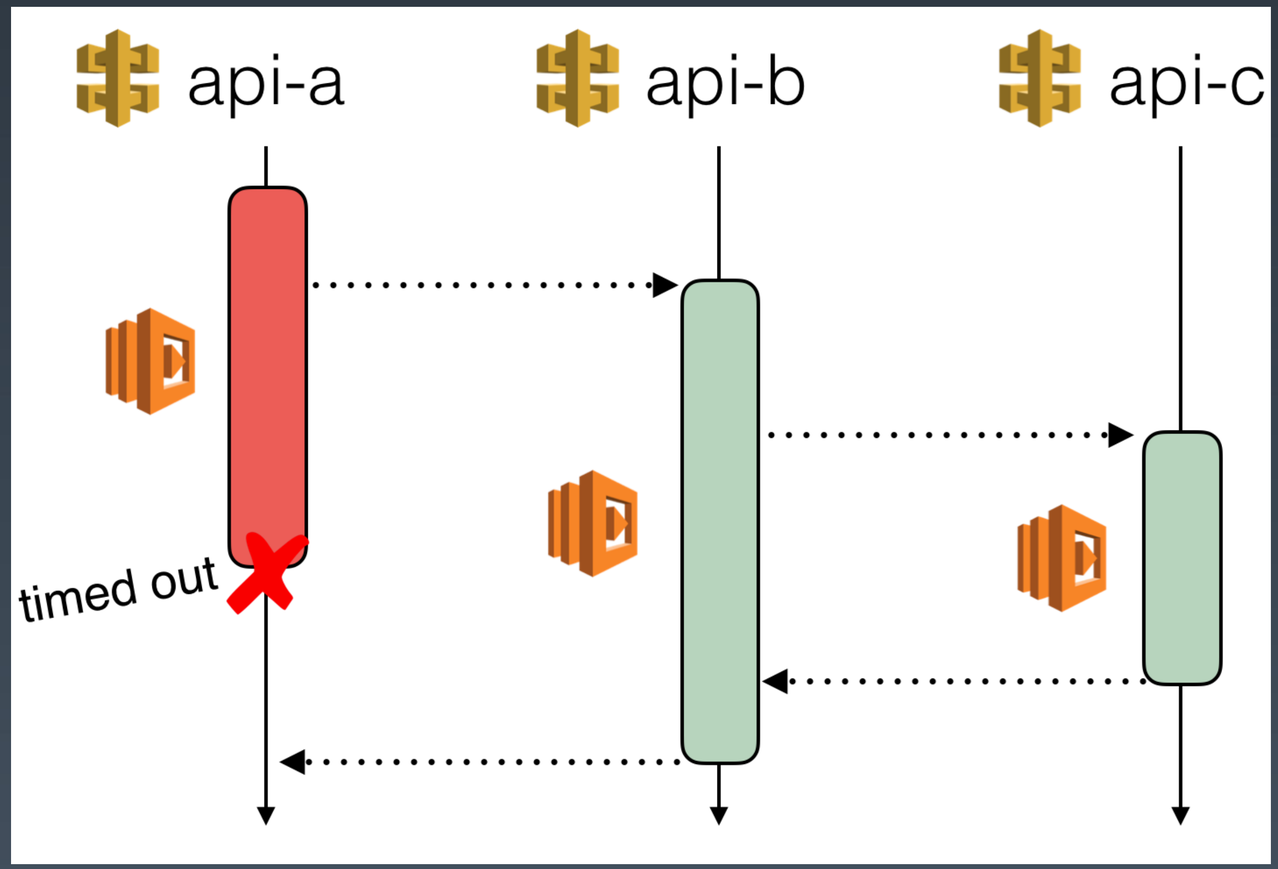
  托管特性，用户无法管控

  故障发生时，用户可以做的事非常有限



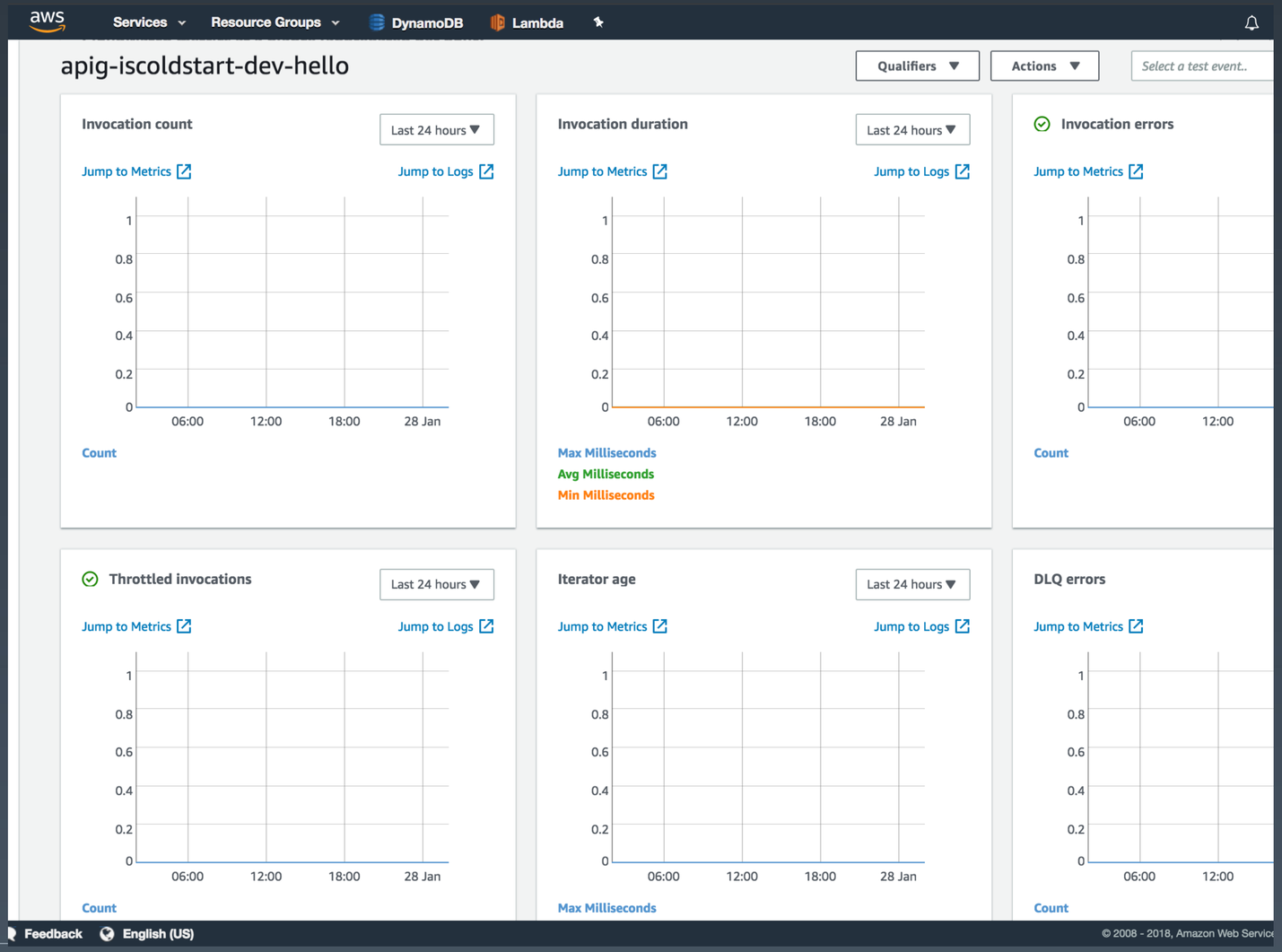
选定故障场景

不合理的超时设置

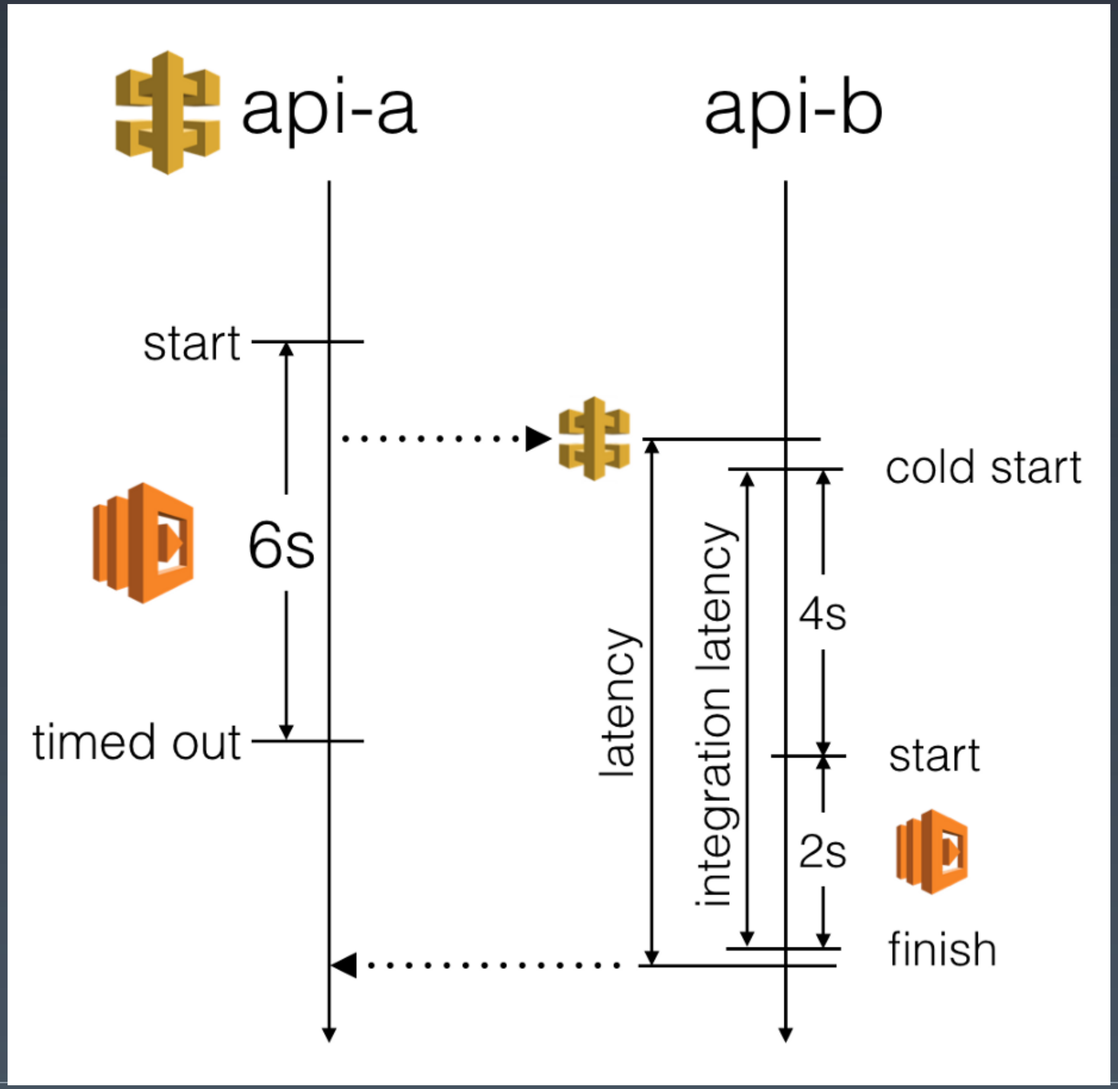


观测指标

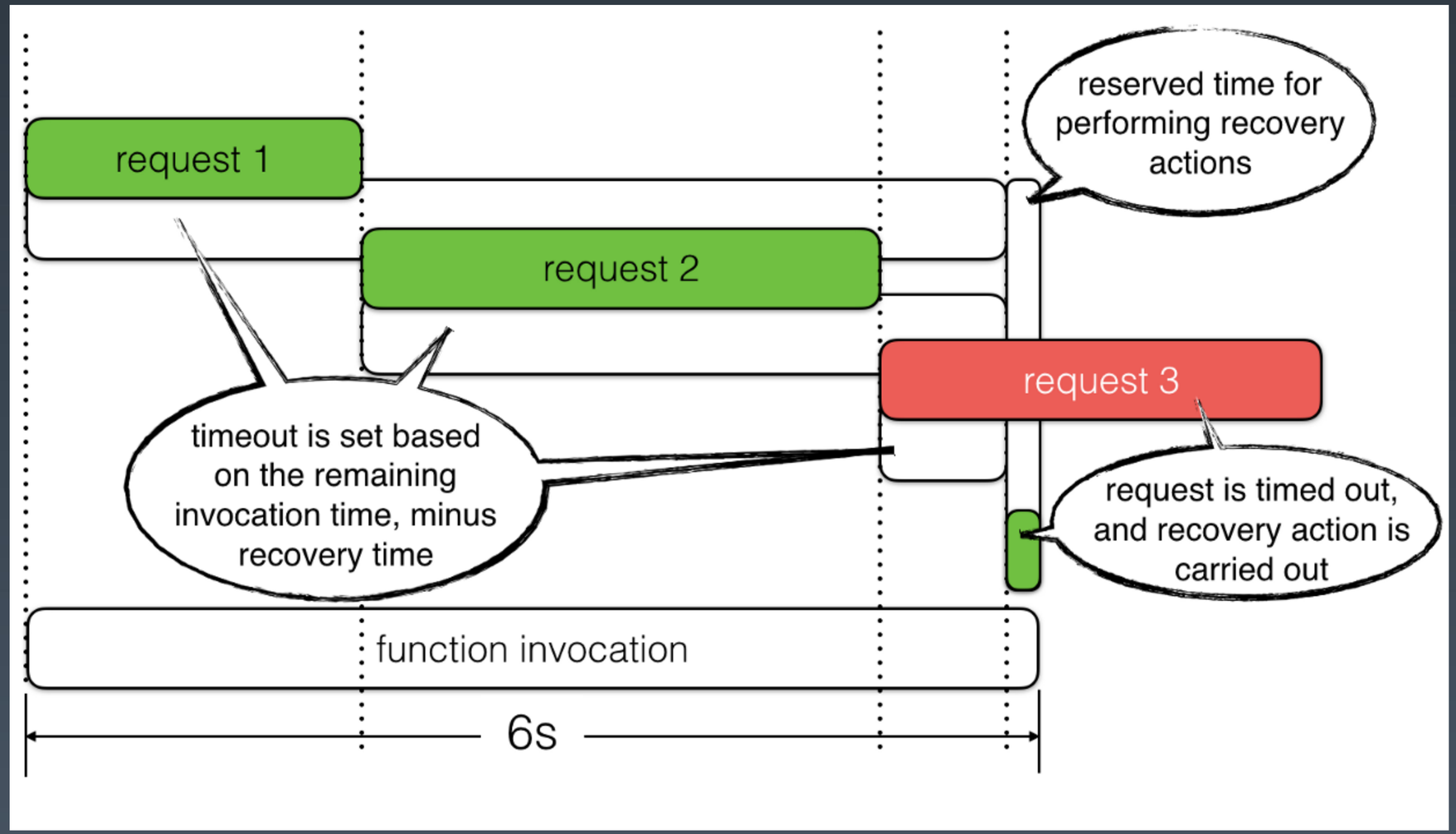
- 响应的延迟时间
- 差错统计
- 完成请求响应的百分比



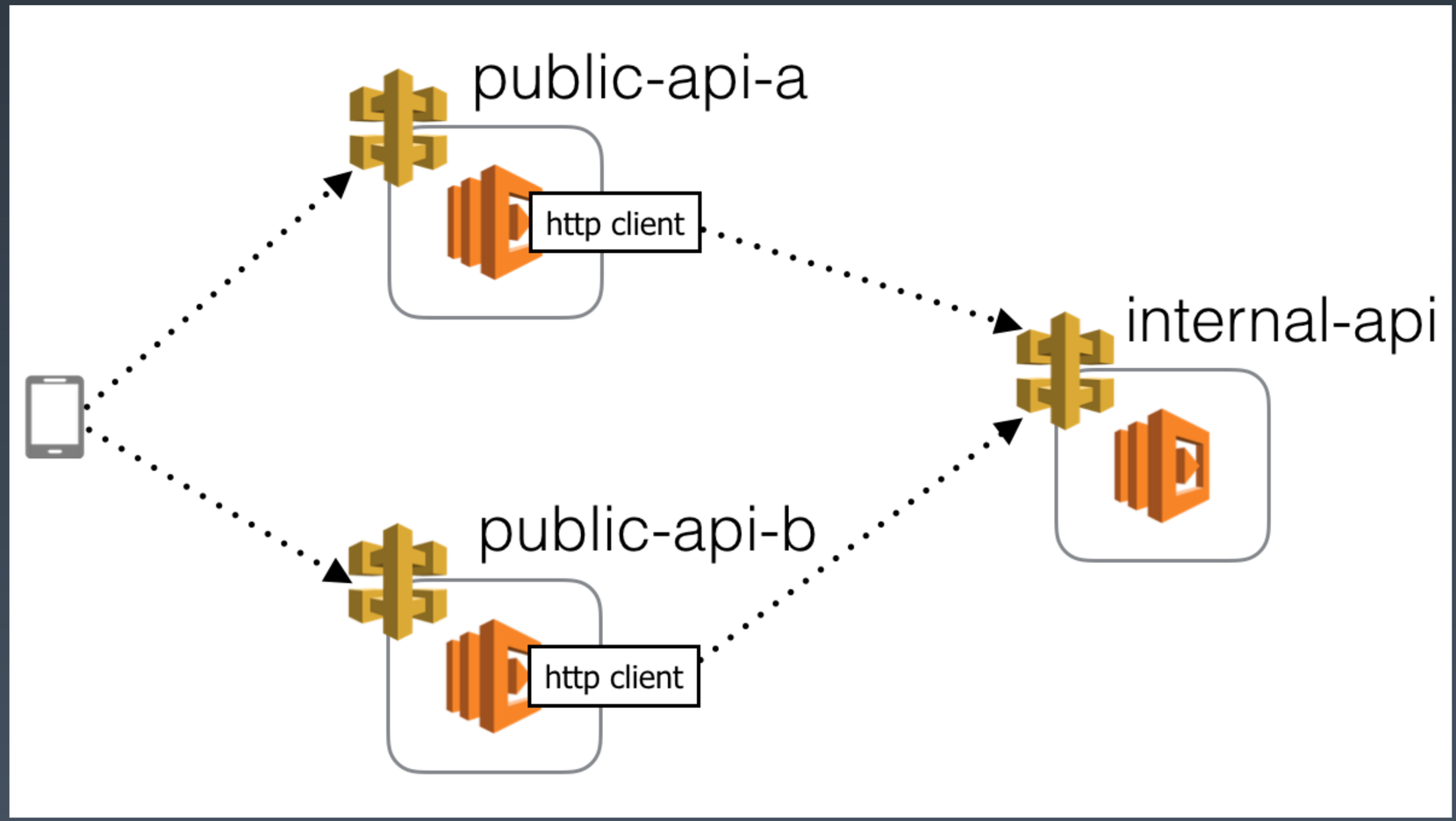
场景: 超时设置


api-a
api-b
start
6s
timed out
cold start
latency
integration latency
4s
start
2s
finish

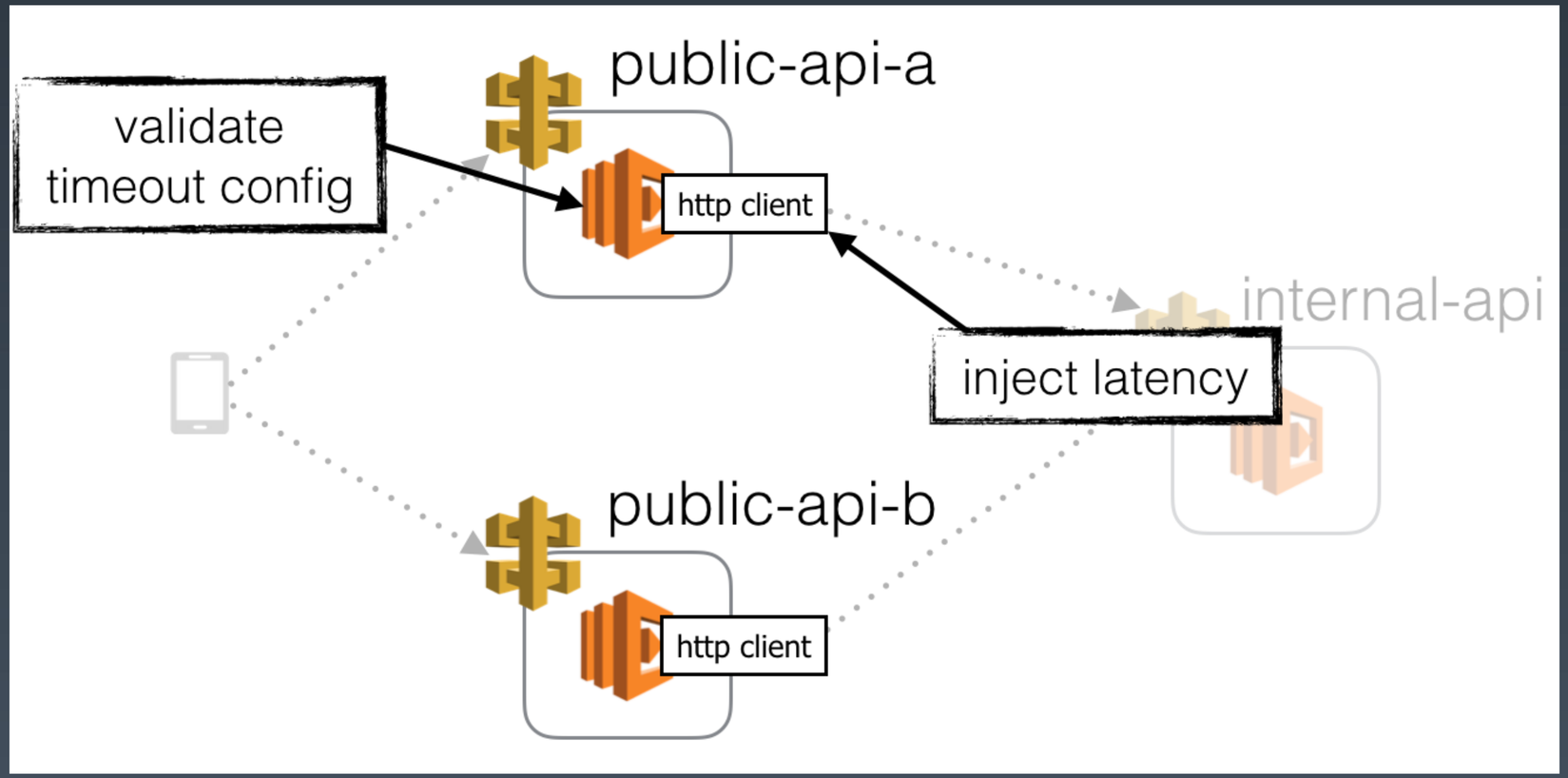
超时设置方案


request 1
request 2
request 3
reserved time for performing recovery actions
timeout is set based on the remaining invocation time, minus recovery time
request is timed out, and recovery action is carried out
function invocation
6s

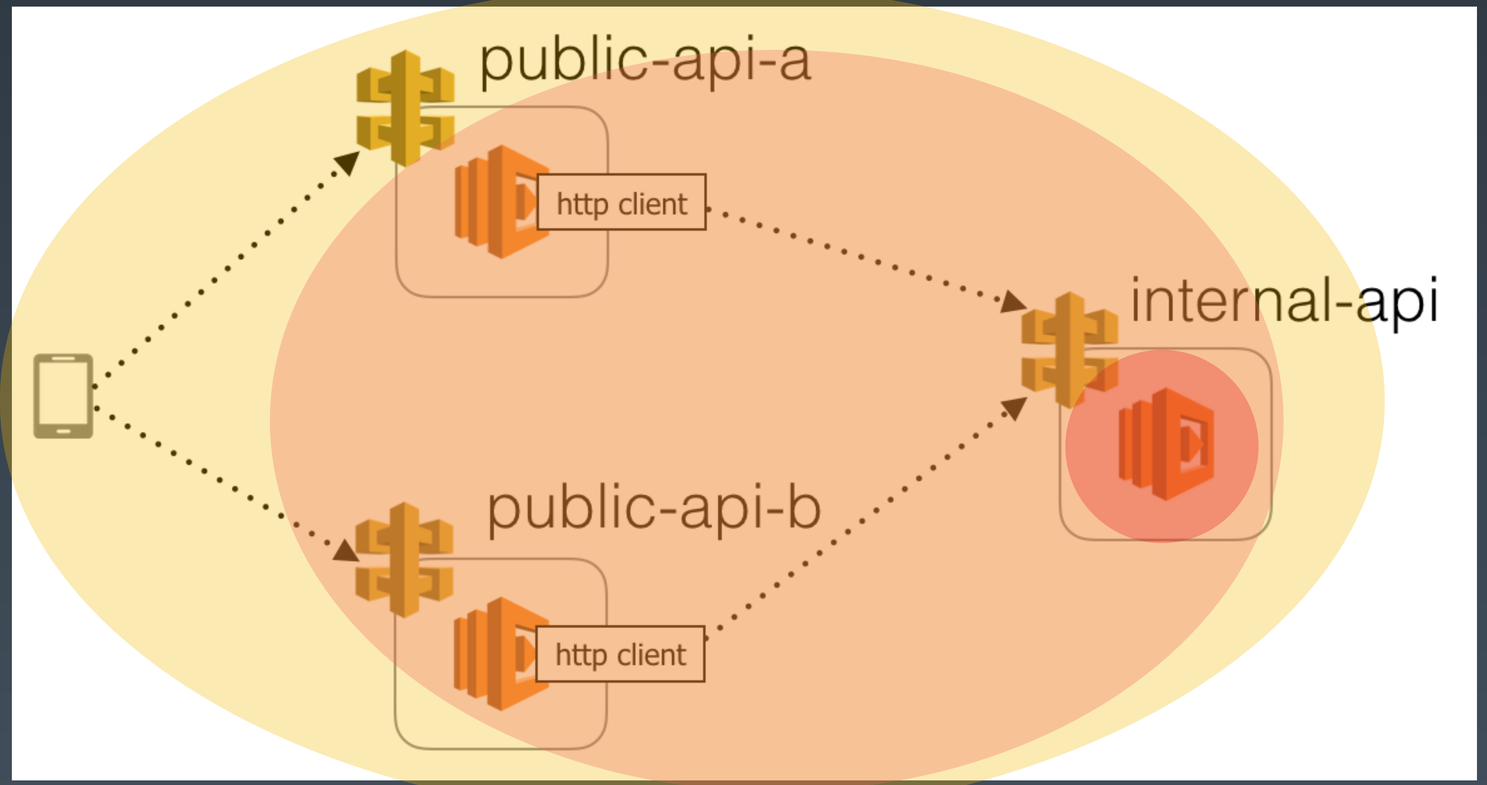
实验环境


public-api-a
http client
internal-api
public-api-b
http client

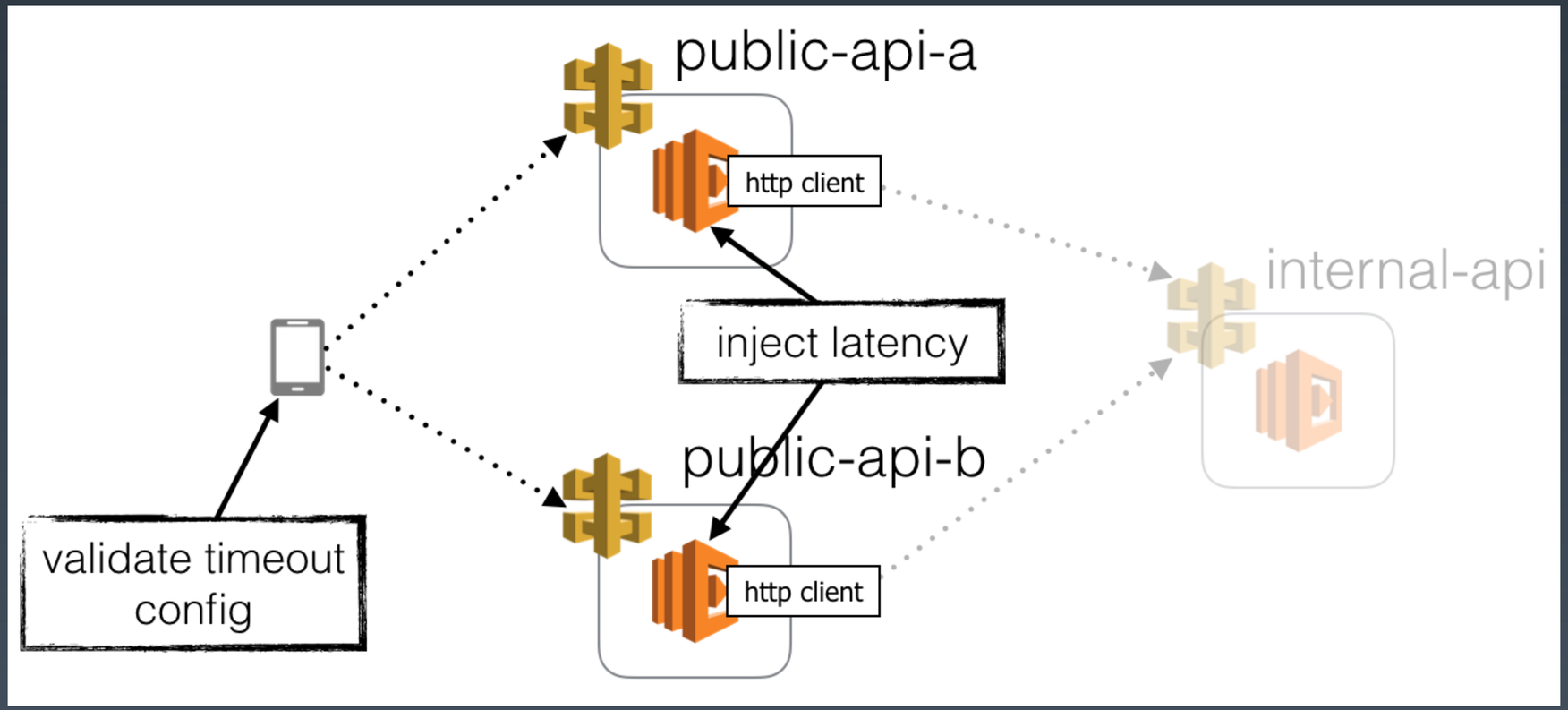
爆炸半径分析


public-api-a
validate
timeout config
http client
internal-api
inject latency
public-api-b
http client

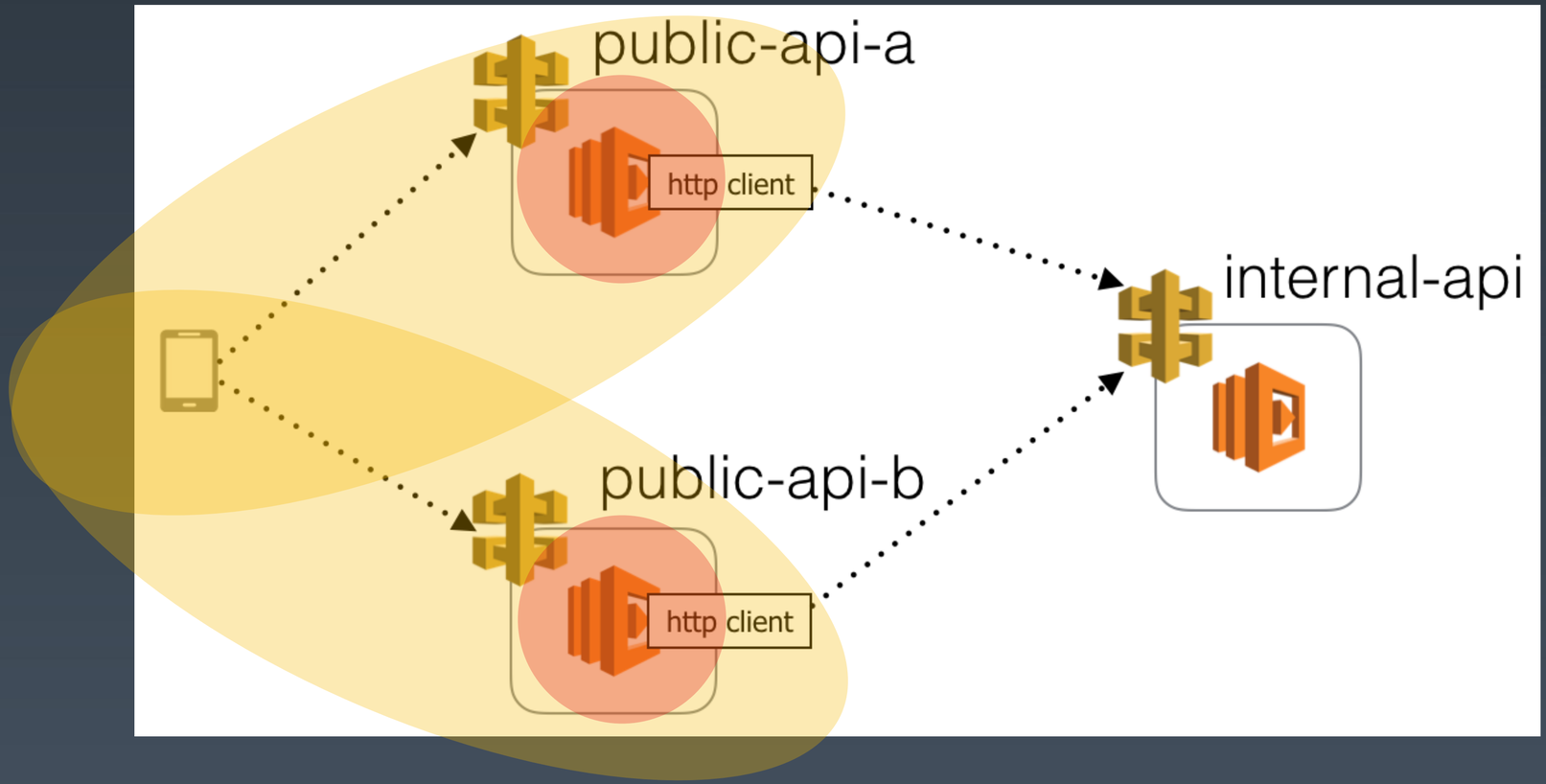
爆炸半径分析


public-api-a
http client
internal-api
public-api-b
http client

爆炸半径分析


public-api-a
http client
internal-api
inject latency
public-api-b
validate timeout
config
http client

爆炸半径分析


public-api-a
http client
internal-api
public-api-b
http client

注入细节

```
const co      = require('co');
const Promise = require('bluebird');

let injectLatency = co.wrap(function* (config) {
  if (config.isEnabled === true && Math.random() < config.probability) {
    let delayRange = config.maxDelay - config.minDelay;
    let delay = Math.floor(config.minDelay + Math.random() * delayRange);

    console.log(`injecting [${delay}ms] latency to HTTP request...`);
    yield Promise.delay(delay);
  }
});
```

注入细节

```
const co      = require('co');
const Promise = require('bluebird');

let injectLatency = co.wrap(function* (config) {
  if (config.isEnabled === true && Math.random() < config.probability) {
    let delayRange = config.maxDelay - config.minDelay;
    let delay = Math.floor(config.minDelay + Math.random() * delayRange);

    console.log(`injecting [${delay}ms] latency to HTTP request...`);
    yield Promise.delay(delay);
  }
});
```

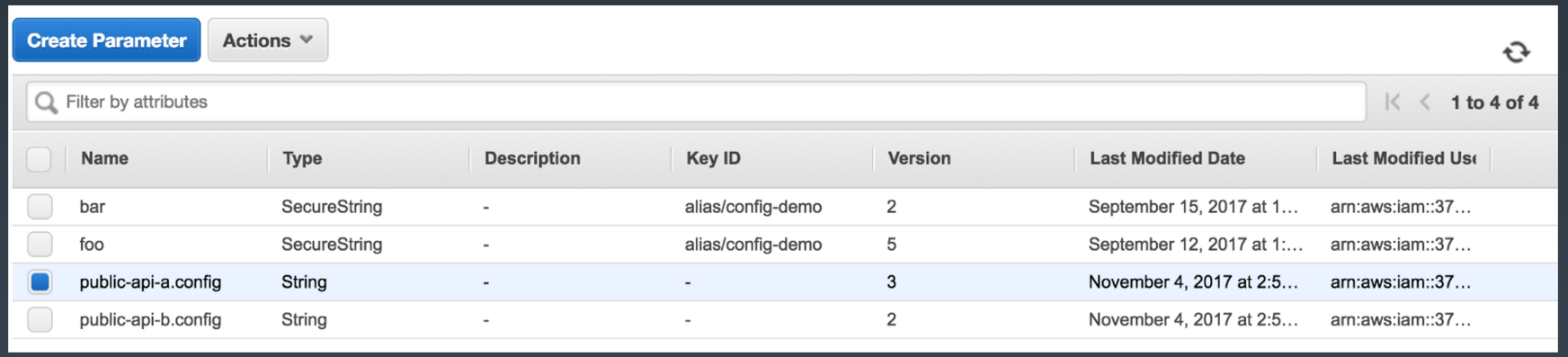
注入细节

```
const co      = require('co');
const Promise = require('bluebird');

let injectLatency = co.wrap(function* (config) {
  if (config.isEnabled === true && Math.random() < config.probability) {
    let delayRange = config.maxDelay - config.minDelay;
    let delay = Math.floor(config.minDelay + Math.random() * delayRange);

    console.log(`injecting [${delay}ms] latency to HTTP request...`);
    yield Promise.delay(delay);
  }
});
```

注入细节

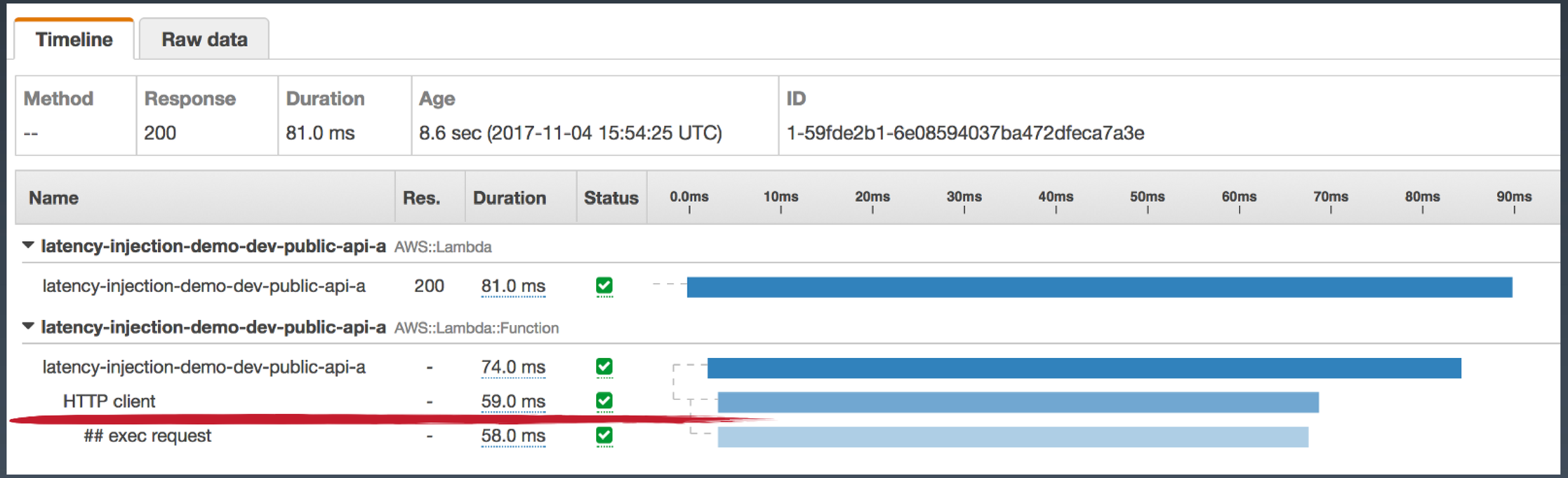
```
let Req = function* (options) {
  let request = getRequest(options);
  let fullResponse = options.resolveWithFullResponse === true;

  let latencyInjectionConfig = options.latencyInjectionConfig;
  yield injectLatency(latencyInjectionConfig);  // <- this is the import bit

  try {
    let resp = yield exec(request);
    return fullResponse ? resp : resp.body;
  } catch (e) {
    if (e.response && e.response.error) {
      throw e.response.error;
    }

    throw e;
  }
};
```

注入配置

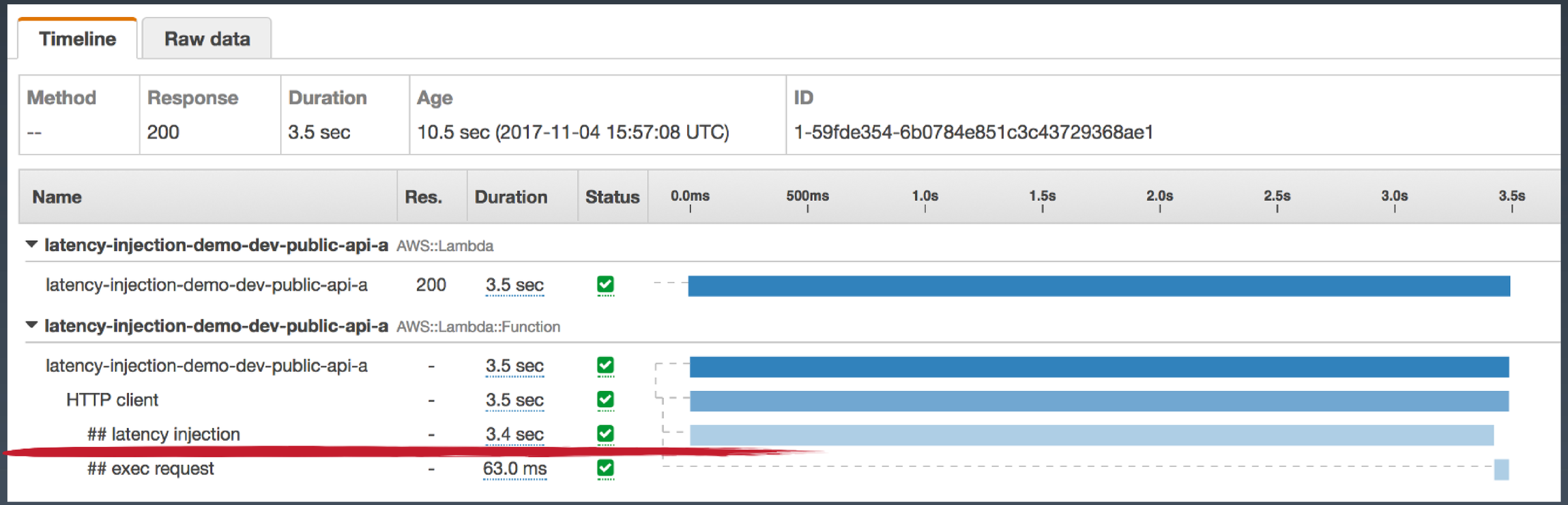



```
{
  "internalApi": "https://xx.amazonaws.com/dev/internal",
  "chaosConfig": {
    "httpClientLatencyInjectionConfig": {
      "isEnabled": true,
      "probability": 0.5,
      "minDelay": 100,
      "maxDelay": 5000
    }
  }
}
```

未注入时


Timeline
Raw data
Method
--
Response
200
Duration
81.0 ms
Age
8.6 sec (2017-11-04 15:54:25 UTC)
ID
1-59fde2b1-6e08594037ba472dfeca7a3e
Name
Res.
Duration
Status
0.0ms
10ms
20ms
30ms
40ms
50ms
60ms
70ms
80ms
90ms
latency-injection-demo-dev-public-api-a AWS::Lambda
latency-injection-demo-dev-public-api-a 200 81.0 ms
latency-injection-demo-dev-public-api-a AWS::Lambda::Function
latency-injection-demo-dev-public-api-a - 74.0 ms
HTTP client - 59.0 ms
exec request - 58.0 ms

注入后


Timeline
Raw data
Method
--
Response
200
Duration
3.5 sec
Age
10.5 sec (2017-11-04 15:57:08 UTC)
ID
1-59fde354-6b0784e851c3c43729368ae1
Name
Res.
Duration
Status
0.0ms
500ms
1.0s
1.5s
2.0s
2.5s
3.0s
3.5s
latency-injection-demo-dev-public-api-a AWS::Lambda
latency-injection-demo-dev-public-api-a 200 3.5 sec
latency-injection-demo-dev-public-api-a AWS::Lambda::Function
latency-injection-demo-dev-public-api-a - 3.5 sec
HTTP client - 3.5 sec
latency injection - 3.4 sec
exec request - 63.0 ms

工具代码

artilleryio / **chaos-lambda**

Watch 11 | Star 227 | Fork 17

Code | Issues 11 | Pull requests 2 | Projects 0 | Wiki | Insights

Serverless chaos monkey for AWS (runs on AWS Lambda)

reliability-engineering | aws | fault-tolerance | chaos-monkey

62 commits | **1** branch | **6** releases | **3** contributors | MPL-2.0

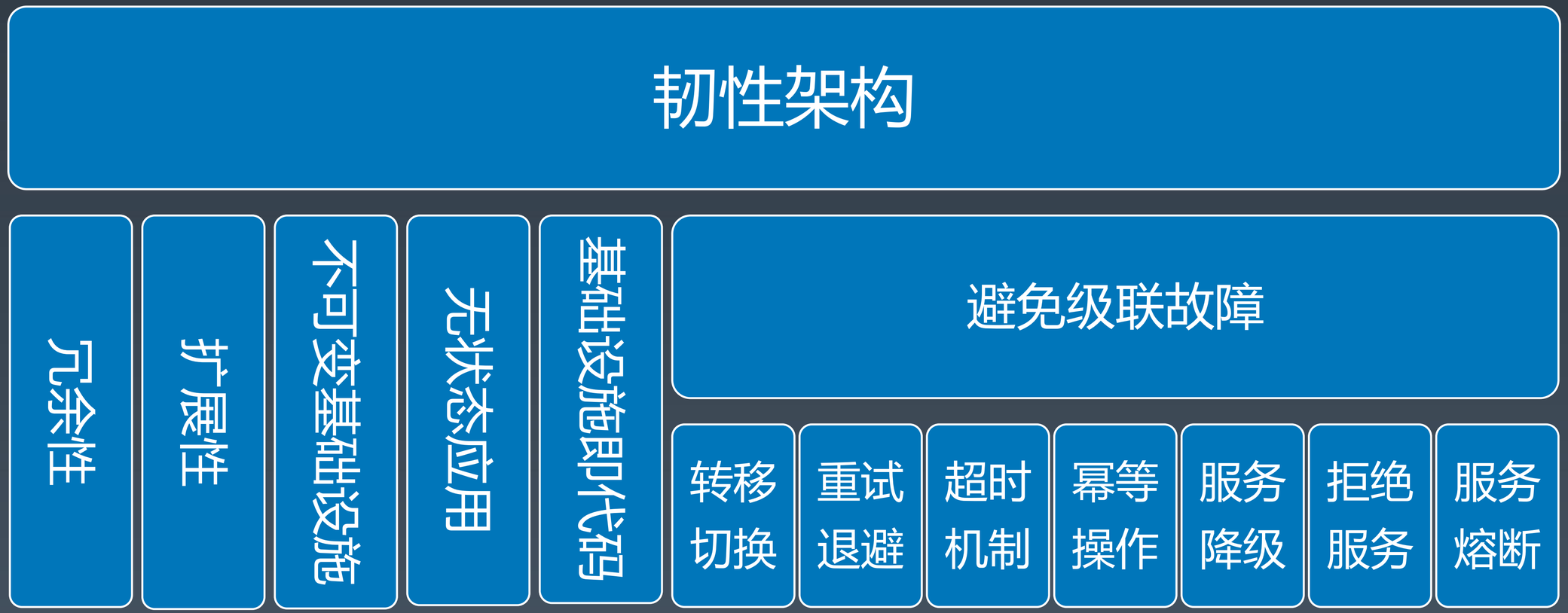
Branch: **master** | New pull request | Create new file | Upload files | Find File | Clone or download

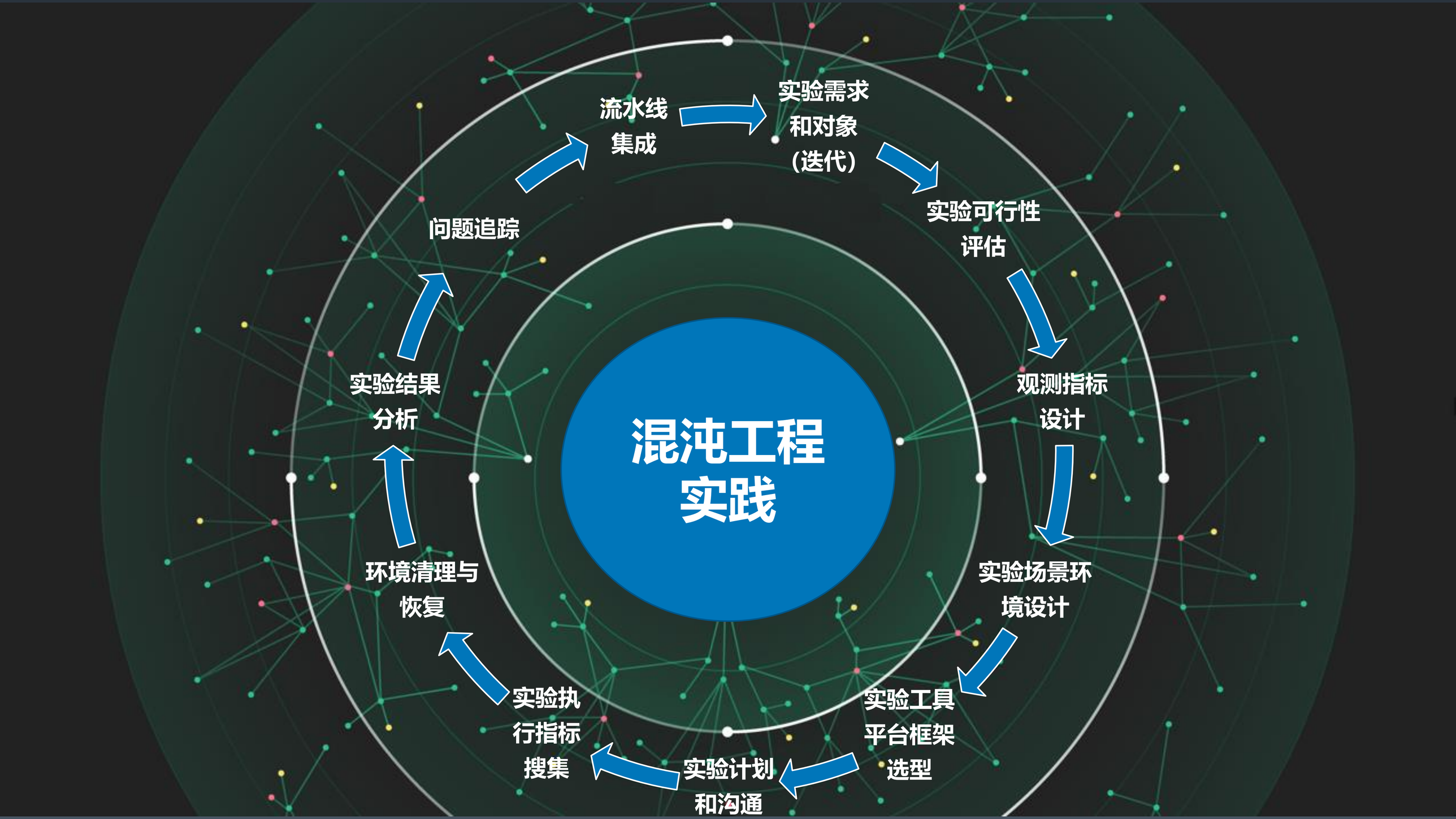
hassy Merge pull request #21 from davinerd/master — Latest commit 24d23ef on Mar 26, 2018

bin	Rename files for the new name	2 years ago
lambda	fixed conflicts	a year ago
lib/commands	fix: Resolve module path correctly	2 years ago
.gitignore	Updating .gitignore	3 years ago
CONTRIBUTING.md	Rename Chaos Lambda to Chaos Llama	2 years ago
Chaosfile.json	fixed conflicts	a year ago
LICENSE.txt	All basics in place	3 years ago
README.md	doc: Update link to website in README	2 years ago
package.json	2.0.2	2 years ago

总结与展望

混沌工程实验目标


韧性架构
冗余性
扩展性
不可变基础设施
无状态应用
基础设施即代码
避免级联故障
转移切换
重试退避
超时机制
幂等操作
服务降级
拒绝服务
服务熔断


混沌工程
实践
实验需求
和对象
（迭代）
实验可行性
评估
观测指标
设计
实验场景环
境设计
实验工具
平台框架
选型
实验计划
和沟通
实验执
行指标
搜集
环境清理与
恢复
实验结果
分析
问题追踪
流水线
集成

GTLC
GLOBAL
TECH LEADERSHIP
CONFERENCE
全球技术领导力峰会

Geekbang. 极客邦科技 | TGO 鲲鹏會

500+高端科技领导者与你一起探讨
技术、管理与商业那些事儿

2019年6月14–15日 | 上海圣诺亚皇冠假日酒店

扫码了解更多信息

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

混沌工程实验最佳实践

成熟度等级	5级
架构抵御故障的能力	实现韧性架构
实验指标设计	可在实验组和控制组之间比较业务指标的差异
实验环境选择	包括生产在内的任意环境都可以运行实验
实验自动化能力	全自动的设计、执行和终止实验
实验工具使用	工具支持交互式的比对实验组和控制组
故障注入场景 爆炸半径范围	可注入如对系统的不同使用模式、返回结果和状态的更改等类型的事件
环境恢复能力	韧性架构自动恢复
实验结果整理	实验结果可预测收入损失、容量规划、区分出不同服务实际的关键程度