GOPS 2021
Shenzhen

# GOPS 2021 全球运维大会 深圳站

-XOPS 风向标

中国 · 深圳

指导单位：DCA 云计算开源产业联盟 RPA产业推进方阵

主办单位：高效运维社区 GreatOPS Community 开放运维联盟 OOPSA Open OPS Alliance RPA时代

时间：2021年5月21日-22日

# ChaosOps 探索与落地实践

黄帅　　亚马逊资深技术专家

# 黄帅

## 亚马逊 资深技术专家

在软件研发领域有十多年架构设计、分布式系统运维以及团队管理经验，近年来对混沌工程企业实践有深入的研究，力主推动亚马逊云科技全球混沌工程服务 AWS Fault Injection Simulator (FIS) 的落地，于2021年3月成功发布。

# 目录
CONTENTS

# 云原生的新挑战

三个生产事件的启示

1.0

# 生产事件一：服务状态仪表盘悖论

GOPS 2021
shenzhen

服务状态仪表盘悖论

(Service Status Paradox)

提供服务健康状态

健康状态更新失败

架构特别简单

存在整整十年之久

却未有人发现

这样的悖论还有吗？

**Service Status**

| | DCO-001 | DCO-002 | DCO-003 | DCO-004 | DCO-005 | DCO-006 | DCO-007 | DCO-008 |
|---|---|---|---|---|---|---|---|---|
| Core services | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy |
| Ninja | Healthy | Degraded | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy |
| Athos | Healthy | Healthy | Disrupted | Healthy | Healthy | Healthy | Healthy | Healthy |
| Metis | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy |
| Demeter | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy | Disrupted |
| Leto | Informational | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy |
| Other services | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy | Healthy |

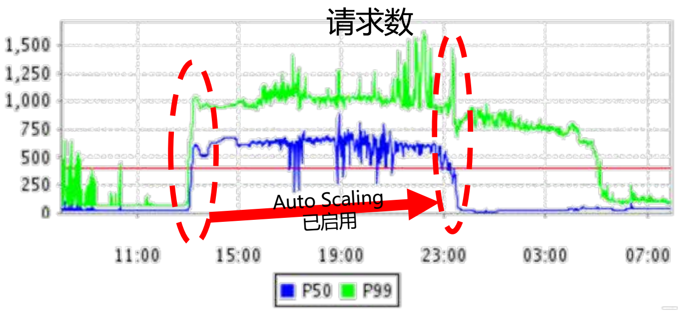
Healthy　Degraded　Disrupted　Informational

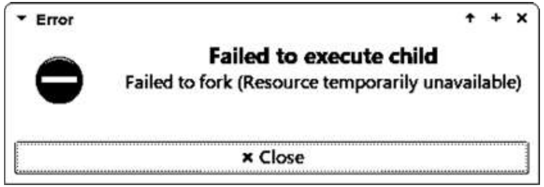
## 故障类型：依赖失效

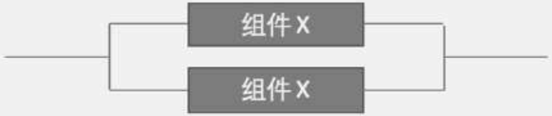
# 生产事件二：冗余性悖论

冗余性悖论

请问：冗余性是否一定能提高可用性？




$$A=1-(1-Ax)^2$$

| 组件 | 可用性(A) | 宕机时长(按年计) |
| --- | --- | --- |
| 一个组件 X 提供过服务 | 99% (2个9) | 3天15个小时 |
| 两个组件 X 并行提供服务 | 99.99% (4个9) | 52分钟 |
| 三个组件 X 并行提供服务 | 99.9999% (6个9) | 31秒 |

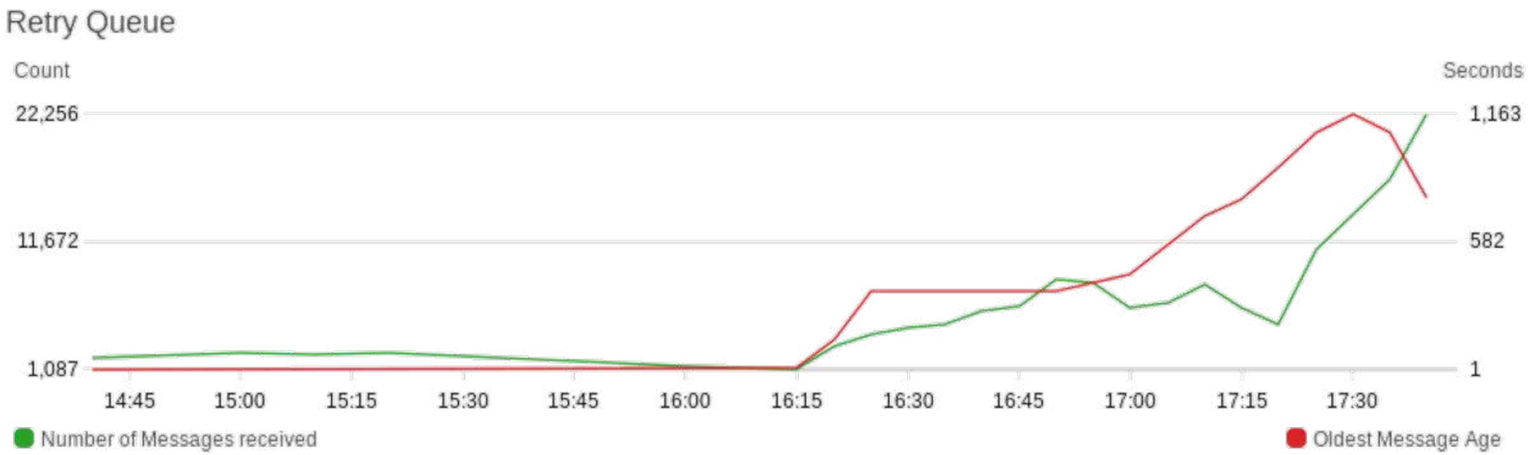
如果故障高度相关，则冗余不会增加可用性，还会产生雪崩效应。

为什么？

故障类型：资源耗尽

# 生产事件三：告警系统瘫痪

所有的功能发布被暂停




<u>故障类型：重试风暴</u>

# 云原生的新挑战

## 1.1

故障的宿命论

# 故障来自哪里？

软件错误
Software Error

主观不可避免

软件缺陷
Software Defect

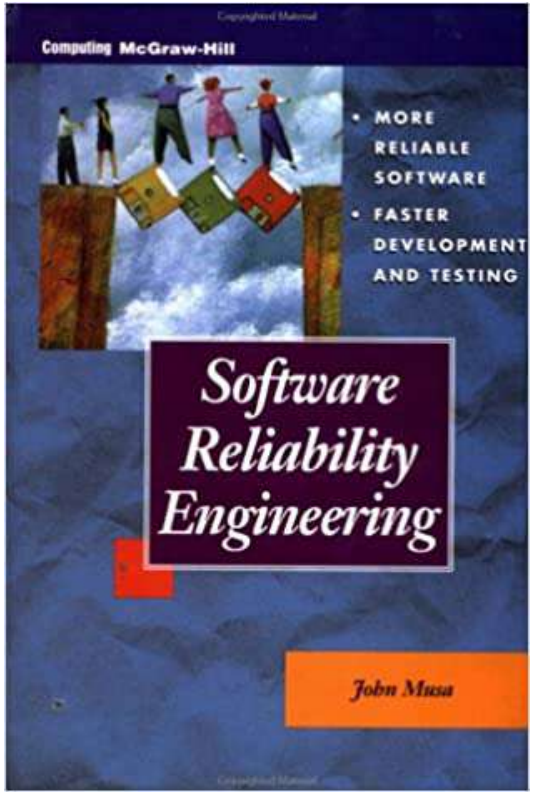
潜伏特性

软件故障
Software Fault

缺陷激活的表象

软件失效
Software Failure

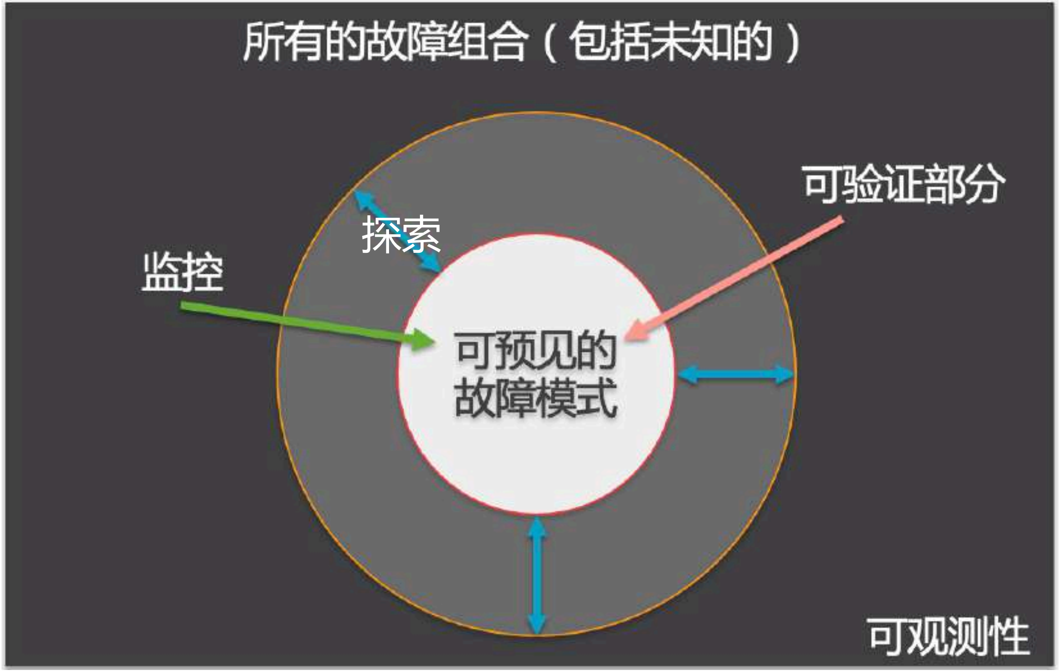
行为结果




1988

GOPS 2021 shenzhen

# 故障的宿命论




一切皆因人类设计的系统变得愈发复杂，超出了人类认知范围。

# 常见的故障类型

GOPS 2021 shenzhen

- 负载尖峰和突发过载
- 邻近的故障转移
- 崩溃查询
- 重试风暴
- 资源耗尽
- 资源限制

- CPU问题
- 内存问题
- 线程枯竭
- 发布和变更
- 启动时间过长
- 依赖失效

小故障引起海啸，这和云原生有什么关系？

# 云原生的新挑战

面临的问题

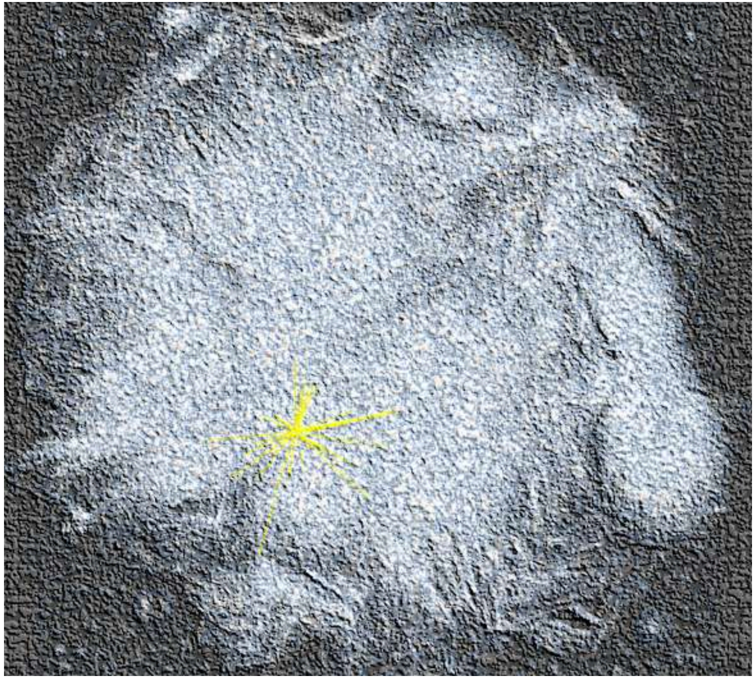
1.2

# 云原生时代的新挑战

GOPS 2021
shenzhen

分布式架构的复杂性

1. 预生产无法和生产保持完全一致
2. 测试通过，一上生产就出问题
3. 集成测试的复杂性
4. 回归测试的不确定性
5. 端到端测试的难度
6. 故障注入测试的重视



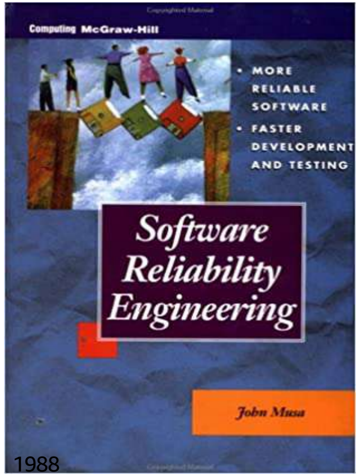
2021

1. 级联故障模式的探索
2. 可观测性的复杂度
3. 配置错误的危险性
4. 功能标记爆炸问题
5. 生产调试的风险
6. 人为错误的避免
7. 应急响应的肌肉记忆

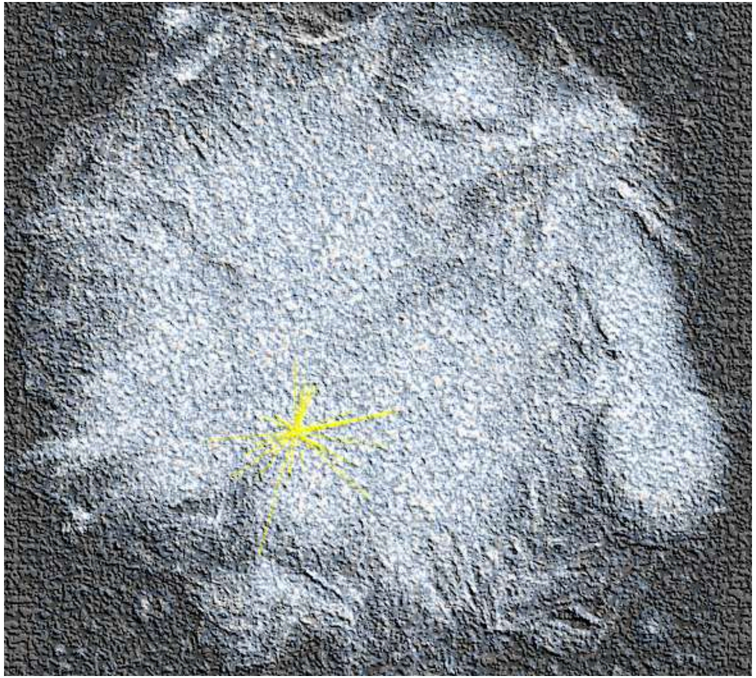
# 可靠性/韧性工程的焕然一新



软件可靠性工程

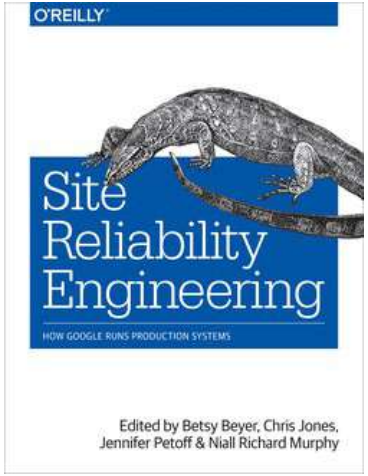



分布式架构的复杂性



2021

站点可靠性工程



# 韧性之道的演进

GameDay文化

2.0

# GameDay文化

Jesse Robbins, “Master of Disaster”


他是一名志愿消防员。

2001：亚马逊唯一的 Master of Disaster 头衔

2004：在亚马逊发明了 GameDay

2007：著名 Web 性能和运维大会（Velocity）的创始人

2008：Chef 的创始人

2013：PagerDuty 的早期投资人

2019：Orion Labs 的创始人和 CEO

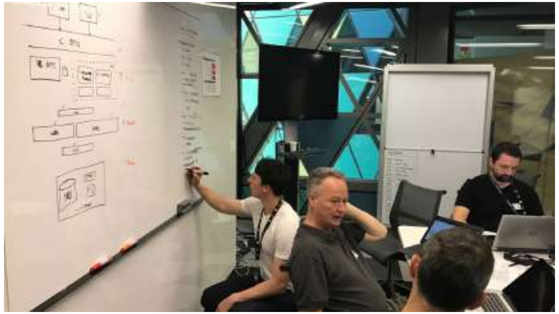
“游戏日计划”是一个基于工程师团队的**交互式**和**开放式**的学习与训练。旨在测试系统中**模拟**各种事件响应的**流程**，比如故障注入、被侵入、性能扩展要求等等。目的是**训练**工程师团队的**响应能力**以及建立如何应对的“**肌肉记忆**”。

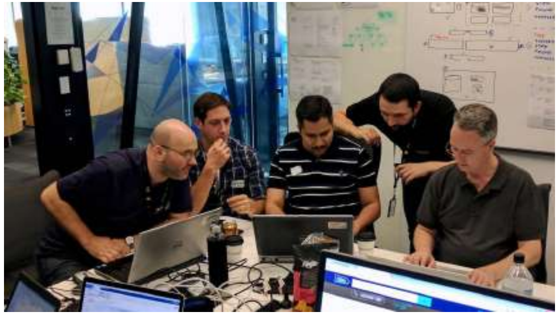
幸运大转盘

FAQs

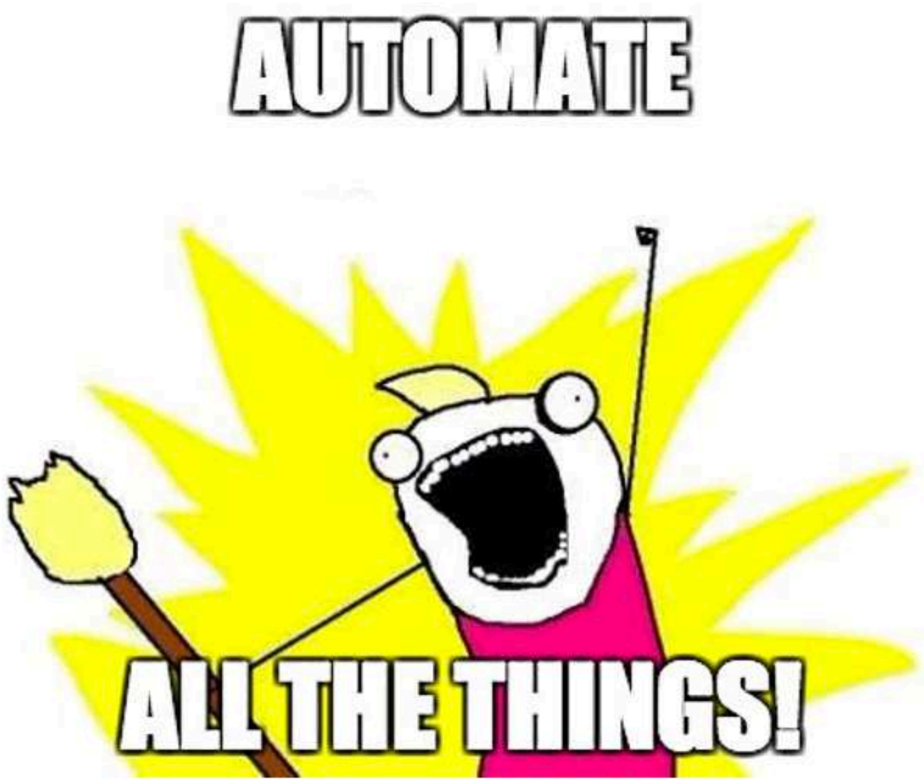
For 2020, please see 2020 GameDay FAQ
For 2019, please see 2019 GameDay FAQ
For 2018, please see 2018 GameDay FAQ
For 2017, please see 2017 GameDay FAQ
For 2016, please see 2016 GameDay FAQ
For 2015, please see 2015 GameDay FAQ
For 2014, please see 2014 GameDay FAQ
For 2013, please see 2013 GameDay FAQ
For 2012, please see 2012 GameDay FAQ
For 2011, please see 2011 GameDay FAQ
For 2010, please see 2010 GameDay FAQ
For 2009, please see 2009 GameDay FAQ
For 2008, please see 2008 GameDay FAQ
For 2007, please see 2007 GameDay FAQ
For 2006, please see 2006 GameDay FAQ
For 2005, please see 2005 GameDay FAQ
For 2004, please see 2004 GameDay FAQ

# GameDay文化





# GameDay文化



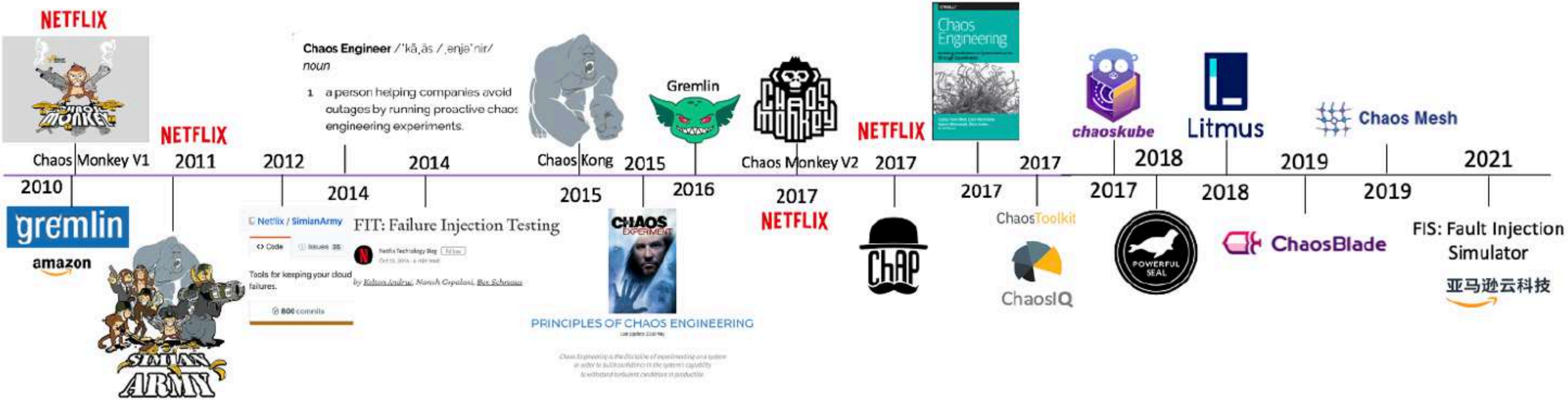
# 韧性之道的演进

## 混沌工程的诞生

# 2.1

# 混沌工程的十年演进史

这是最好的时代，海内外齐头并进，不分伯仲！



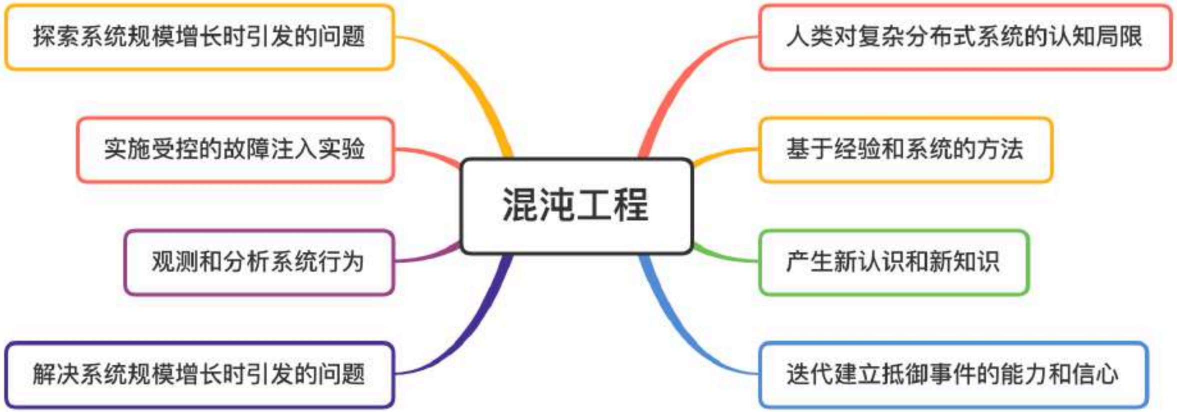

引用自亚马逊官方博客 https://amazonaws-china.com/cn/blogs/china/aws-chaos-engineering-start/

# 混沌工程的目标：韧性架构和自愈能力

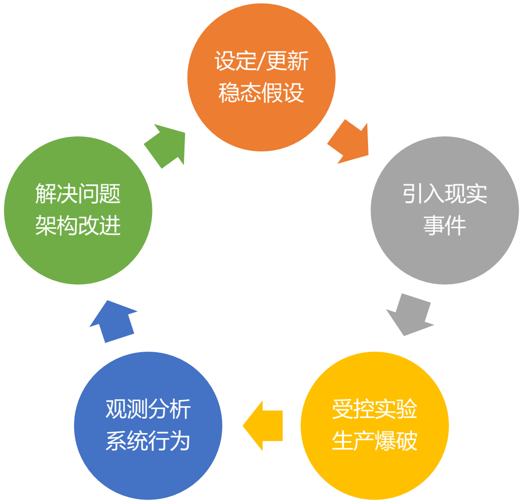
## 韧性架构

- 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

# 混沌工程的内容




混沌工程的定位




混沌工程原理

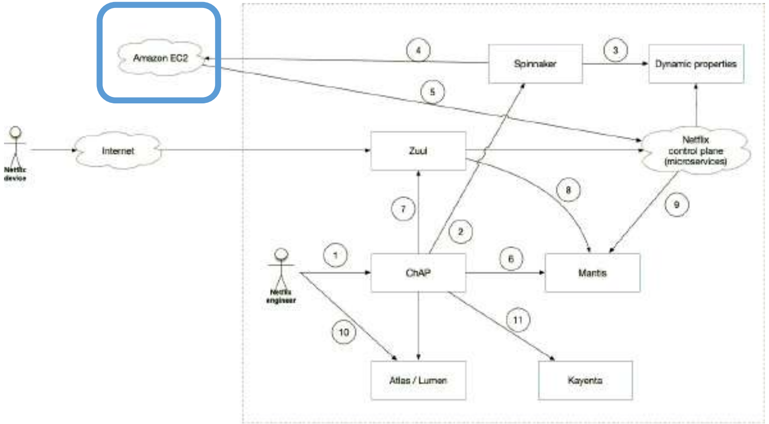
# 韧性之道的演进

## 2.2

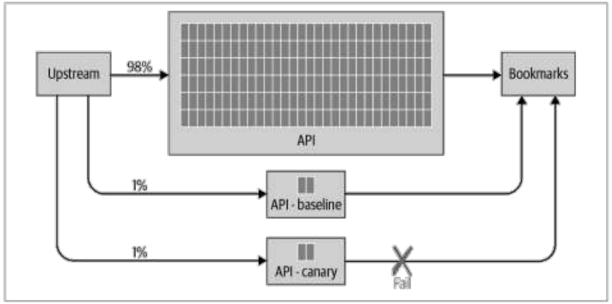
一个最著名的实践案例

# Netflix 混沌工程生产实践 - ChAP

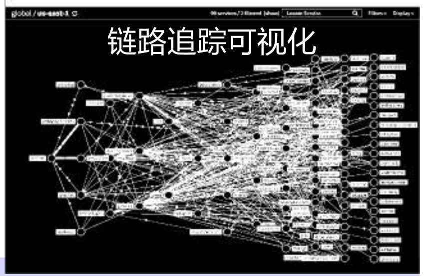
## ChAP工作原理



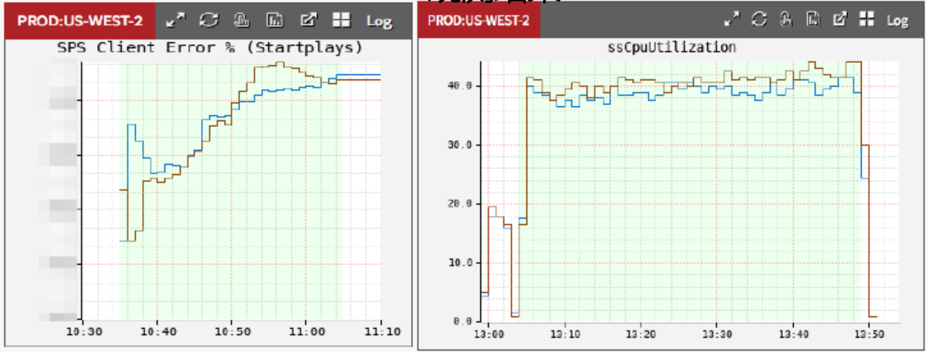

## ChAP故障注入模式




## ChAP控制台


## ChAP观测结果



# 混沌工程技术的演进总结（**精华**）

| 技术内容 | 2014年之前 | 2017年之后 |
| --- | --- | --- |
| 故障场景类型 | 基础设施的单一故障 | 多层次组合故障 |
| 故障场景编排 | 预先指定的故障 | 故障优先级排序和自动挑选 |
| 故障注入模式 | 随机注入 | 对照实验观测 |
| 目标工作环境 | 非生产环境 | 生产灰度环境 |
| 监控/观测能力 | 监控与日志系统 | 含链路追踪的可观测性体系 |
| 结果/成果分析 | 阈值或人工判定 | 自动灰度稳态分析 |
| 安全管控方法 | 人工评估 | 最小爆炸半径和一键关停等 |
| 具体呈现形态 | 命令行工具 | 自动化平台 |

# ChaosOps呼之欲出

## 应对云原生的挑战

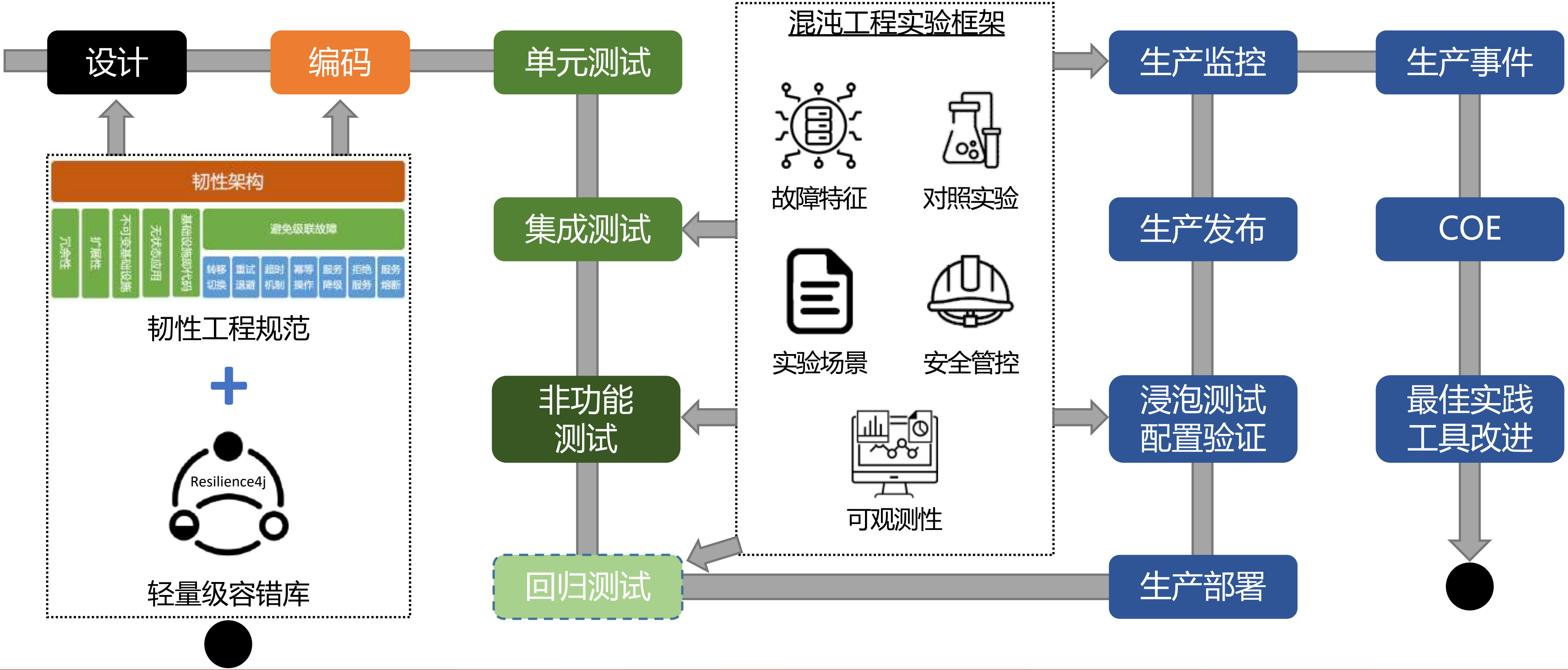
3.0

# ChaosOps又是一个新造的名词？

- 新名词的诞生，往往是因为没有现存的名词，可以突出某项工作的价值。
- 毋庸置疑，ChaosOps自然是要突出混沌工程在整个DevOps实践的价值。

**ChaosOps**是在既有的**DevOps实践**中，通过**创新的混沌工程方法和技术**，应对**云原生时代可靠性工程的新挑战**。

# 混沌工程融入系统开发测试、部署和发布流程



# ChaosOps呼之欲出

核心原理

3.1

故障特征

对照实验

实验场景

安全管控

可观测性

# ChaosOps呼之欲出

## 3.2

核心原理

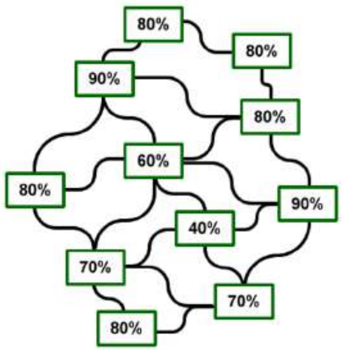
故障特征　对照实验　实验场景　安全管控　可观测性

# 常见故障特征



- 负载尖峰和突发过载 负载测试
- 邻近的故障转移 容灾演练
- 崩溃查询 功能测试
- 重试风暴 故障注入
- 资源耗尽 故障注入
- 资源限制 故障注入

- CPU问题 故障注入
- 内存问题 故障注入
- 线程枯竭 故障注入
- 发布和变更 回归测试
- 启动时间过长 故障注入
- 依赖失效 故障注入



Network running normally

# 常见故障注入点



| 故障注入点 | 具体描述 |
|---|---|
| 依赖型故障 | AWS 托管服务异常：ELB / S3 / DynamoDB / Lambda ... |
| 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，容器异常终止，容器异常关闭，... |
| 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，熵耗尽，... |
| 网络故障 | 网络延迟，网络丢包，DNS 解析故障，DNS 缓存毒化，VIP 转移，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，内核崩溃，... |
| 请求拦截型故障 | 异常请求，请求处理延迟，... |

# ChaosOps呼之欲出

## 核心原理

3.3

故障特征

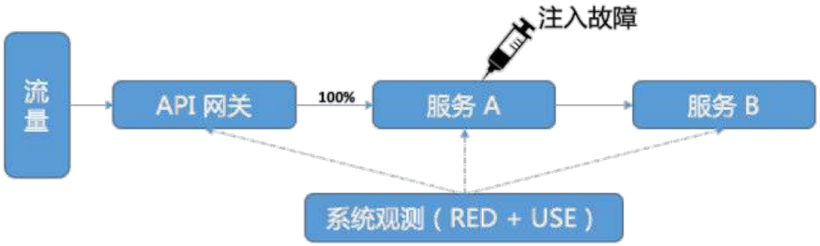
对照实验

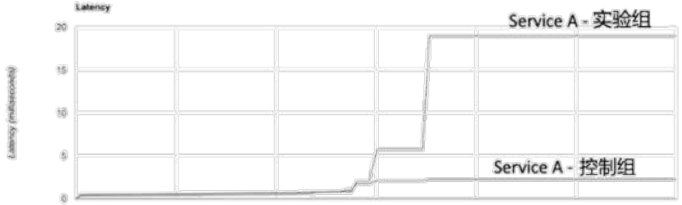
实验场景

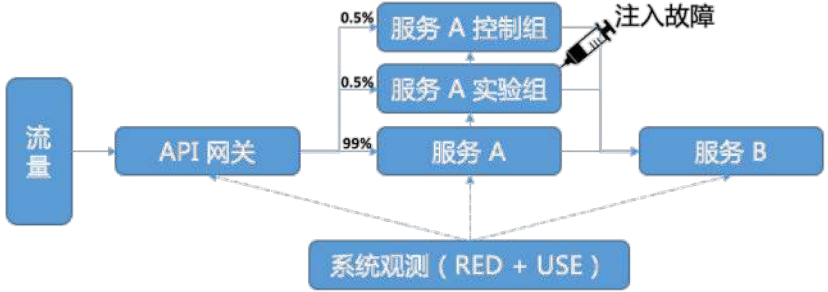
安全管控

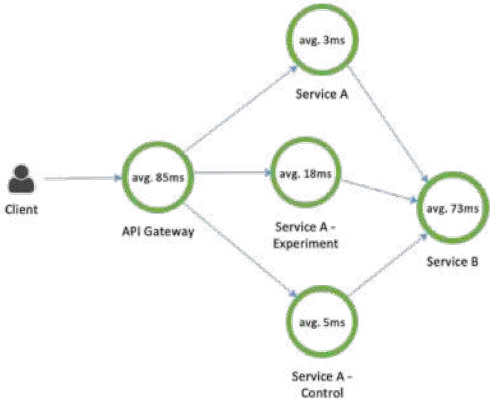
可观测性

# 对照实验原理



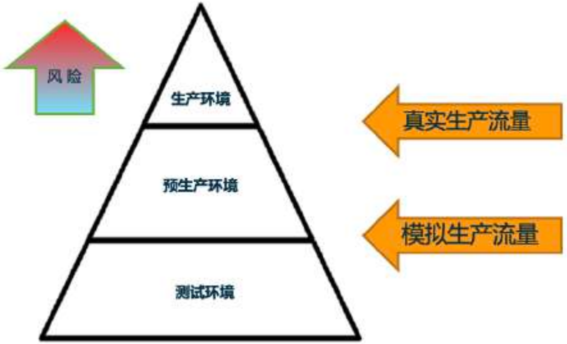



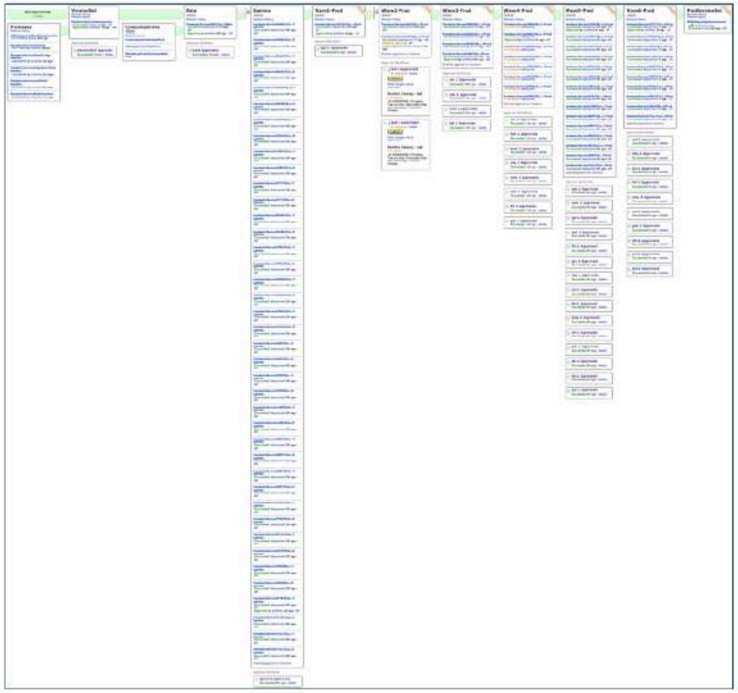

**用很小比例可能受到影响的用户进行实验**
**借以提升整个系统的可靠性**

# 示例：生产级对照实验

1. 生产的灰度环境
2. 很小比例的生产流量
3. 对比故障注入效果
4. 非生产，则使用流量回放







# ChaosOps呼之欲出

## 核心原理

# 3.4

故障特征

对照实验

实验场景

安全管控

可观测性

# 实验场景完整示例

实验名称

实验描述

实验权限

暂停条件

实验对象

故障注入点

```
{
    "tags": {
        "Name": "StopAndRestartRandomeInstance"
    },
    "description": "Stop and Restart One Random Instance",
    "roleArn": "arn:aws:iam::0123456789:role/MyFISExperimentRole",
    "stopConditions": [
        {
            "source": "aws:cloudwatch:alarm",
            "value": " "arn:aws:cloudwatch:us-east-1:0123456789:alarm:No_Traffic"
        }
    ],
    "targets": {
        "myInstance": {
            "resourceTags": {
                "Env": "test"
            },
            "resourceType": "aws:ec2:instance",
            "selectionMode": "PERCENT(25)"
        }
    },
    "actions": {
        "StopInstances": {
            "actionId": "aws:ec2:stop-instances",
            "description": "stop the instances",
            "parameters": {
                "startInstancesAtEnd": "true",
                "duration": "PT2M",
            },
            "targets": {
                "Instances": "myInstance"
            }
        }
    }
}
```

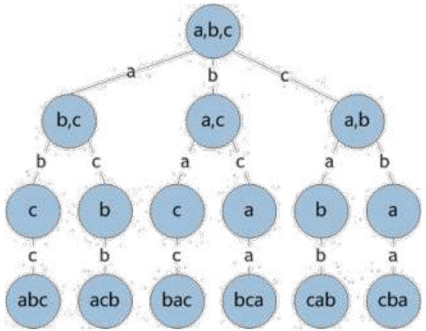
# 标准化实验场景库


定制化实验场景

- 历史故障重现
- 历史故障类比引申
- 探索性实验研究

- https://status.azure.com/zh-cn/status/history/
- https://status.cloud.google.com/summary
- http://aws.amazon.com/premiumsupport/technology/pes/

## 实验场景爆炸问题

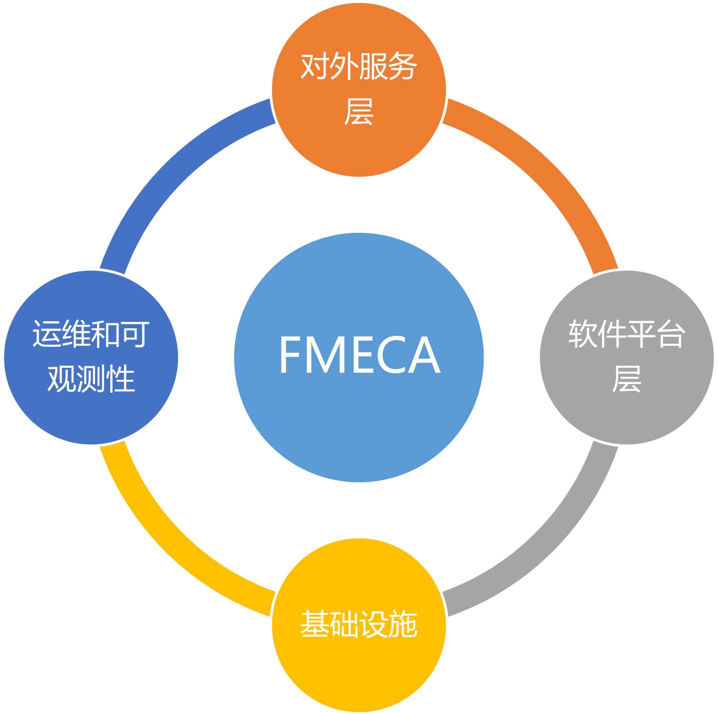



某系统是由相互依赖的10种组件构成，按组件失效的全排列组合计算，总共会产生 10! = 3,628,800 种实验场景。

**实验场景的人力、时间和资源开销**

- 实验场景的持续维护
- 实验场景的最小爆炸半径
- 实验场景的执行模式
- 实验场景的有效性探索和验证
- 实验场景发现问题时的根因分析

# 失效模式影响的关键因素分析法（FMECA）




## History of the FMEA

- 1940s - First developed by the US military in 1949 to determine the effect of system and equipment failures
- 1960s - Adopted and refined by NASA (used in the Apollo Space program)
- 1970s – Ford Motor Co. introduces FMEA after the Pinto affair. Soon adopted across automotive industry
- Today – FMEA used in both manufacturing and service industries

# 以对外服务层为例

| | | | |
|---|---|---|---|
| System | Cloud Service Availability | FMEA Number | 1 |
| Subsystem | Application Layer | Prepared By | Adrian Cockcroft |
| Component | Code created by the application dev team | FMEA Date | 12/6/2018 |
| Design Lead | | Revision Date | 4/26/2021 |
| Core Team | | Page | 1 of 1 |

## Potential Failure Modes and Effects

| Item / Function | Potential Failure Mode(s) | Potential Effect(s) of Failure | Sev | Potential Cause(s)/ Mechanism(s) of Failure | Prob | Current Design Controls | Det | RPN | Recommended Action(s) | Responsibility & Target Completion Date | Action Results: Actions Taken | New Sev | New Occ | New Det | New RPN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Authentication | Client can't authenticate | Can't connect application | 5 | Certificate timeout, version mismatch, account not setup, credential changed | 3 | Log and alert on authentication failures | 3 | 45 | | | | | | | 0 |
| | Slow or unreliable authentication | Slow start for application | 4 | Auth service overloaded, high error and retry rate | 3 | Log and alert on high authentication latency and errors | 4 | 48 | | | | | | | |
| | | | | | | | | 0 | | | | | | | 0 |
| Client Request to API Endpoint | Service unknown, address un-resolvable | Delay while discovery or DNS times out, slow fallback response | 6 | DNS configuration error, denial of service attack, or provider failure | 1 | Customer eventually complains via call center | 10 | 60 | Dual redundant DNS, fallback to local cache, hardcoded IP addresses. Endpoint monitoring and alerts | | | | | | 0 |
| | Service unreachable, request undeliverable | Fast fail, no response | 4 | Network route down or no service instances running | 1 | Autoscaler maintains a number of healthy instances | 1 | 4 | Endpoint monitoring and alerts | | | | | | 0 |
| | Service reachable, request undeliverable | Connect timeout, slow fail, no response | 4 | Service frozen/not accepting connection | 1 | Retry request on different instance. Healthcheck failure instances removed. Log and alert. | 2 | 8 | | | | | | | 0 |
| | Request delivered, no response - stall | Application request timeout, slow fail, no response | 4 | Broken service code, overloaded CPU or slow dependencies | 1 | Retry request on different instance. Healthcheck failure instances removed. Log and alert. | 2 | 8 | | | | | | | 0 |
| | Response undeliverable | Application request timeout, slow fail, no response | 4 | Network return route failure, dropped packets | 1 | Retry request on different instance. Healthcheck failure instances removed. Log and alert. | 2 | 8 | | | | | | | 0 |
| | Response received in time but empty or unintelligible | Fast fail, no response | 3 | Version mismatch or exception in service code | 2 | Retry request on different instance. Healthcheck failure instances removed. Log and alert. | 2 | 12 | | | | | | | 0 |
| | Request delivered, response delayed beyond spec | Degraded response arrives too late, slow fallback response | 6 | Service overloaded or GC hit, dependent services responding slowly | 2 | Retry request on different instance. Healthcheck failure instances removed. Log and alert. | 2 | 24 | | | | | | | 0 |
| | Request delivered, degraded response delivered in time | Degraded timely response | 2 | Service overloaded or GC hit, dependent services responding slowly | 2 | Log and alert on high service latency and errors | 2 | 8 | | | | | | | 0 |

RPN=Sev*Prob*Det → Priority

| Effect | SEVERITY of Effect | Ranking |
|---|---|---|
| Hazardous without warning | Very high severity ranking when a potential failure mode affects safe system operation without warning | 10 |
| Hazardous with warning | Very high severity ranking when a potential failure mode affects safe system operation with warning | 9 |
| Very High | System inoperable with destructive failure without compromising safety | 8 |
| High | System inoperable with equipment damage | 7 |
| Moderate | System inoperable with minor damage | 6 |
| Low | System inoperable without damage | 5 |
| Very Low | System operable with significant degradation of performance | 4 |
| Minor | System operable with some degradation of performance | 3 |
| Very Minor | System operable with minimal interference | 2 |
| None | No effect | 1 |

| PROBABILITY of Failure | Failure Prob | Ranking |
|---|---|---|
| Very High: Failure is almost inevitable | >1 in 2 | 10 |
| | 1 in 3 | 9 |
| High: Repeated failures | 1 in 8 | 8 |
| | 1 in 20 | 7 |
| Moderate: Occasional failures | 1 in 80 | 6 |
| | 1 in 400 | 5 |
| | 1 in 2,000 | 4 |
| Low: Relatively few failures | 1 in 15,000 | 3 |
| | 1 in 150,000 | 2 |
| Remote: Failure is unlikely | <1 in 1,500,000 | 1 |

| Detection | Likelihood of DETECTION by Design Control | Ranking |
|---|---|---|
| Absolute Uncertainty | Design control cannot detect potential cause/mechanism and subsequent failure mode | 10 |
| Very Remote | Very remote chance the design control will detect potential cause/mechanism and subsequent failure mode | 9 |
| Remote | Remote chance the design control will detect potential cause/mechanism and subsequent failure mode | 8 |
| Very Low | Very low chance the design control will detect potential cause/mechanism and subsequent failure mode | 7 |
| Low | Low chance the design control will detect potential cause/mechanism and subsequent failure mode | 6 |
| Moderate | Moderate chance the design control will detect potential cause/mechanism and subsequent failure mode | 5 |
| Moderately High | Moderately High chance the design control will detect potential cause/mechanism and subsequent failure mode | 4 |
| High | High chance the design control will detect potential cause/mechanism and subsequent failure mode | 3 |
| Very High | Very high chance the design control will detect potential cause/mechanism and subsequent failure mode | 2 |
| Almost Certain | Design control will detect potential cause/mechanism and subsequent failure mode | 1 |

# ChaosOps呼之欲出

## 核心原理

# 3.5

故障特征

对照实验

实验场景

安全管控

可观测性

# 爆破半径：不能让混沌实验加速生产事件

如何控制爆炸半径

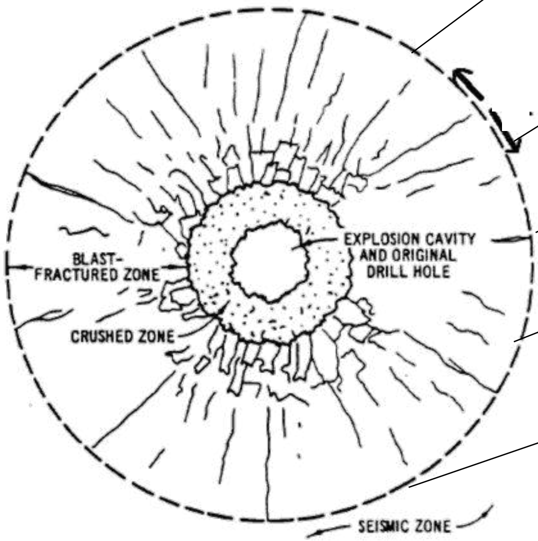
一键关停指标

流量精细控制

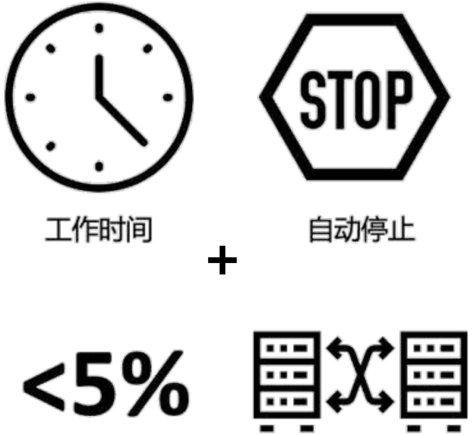
隔离爆破对象

软硬依赖识别

资源限制管理




精细的爆破对象隔离、流量控制和软硬依赖治理，
可有效避免引入更多噪音，影响爆破实验的根因分析




工作时间 + 自动停止

生产流量 故障转移

生产级爆炸半径控制的最佳实践

对爆炸半径的控制是一个迭代学习的过程

# ChaosOps呼之欲出

## 核心原理

# 3.6

故障特征

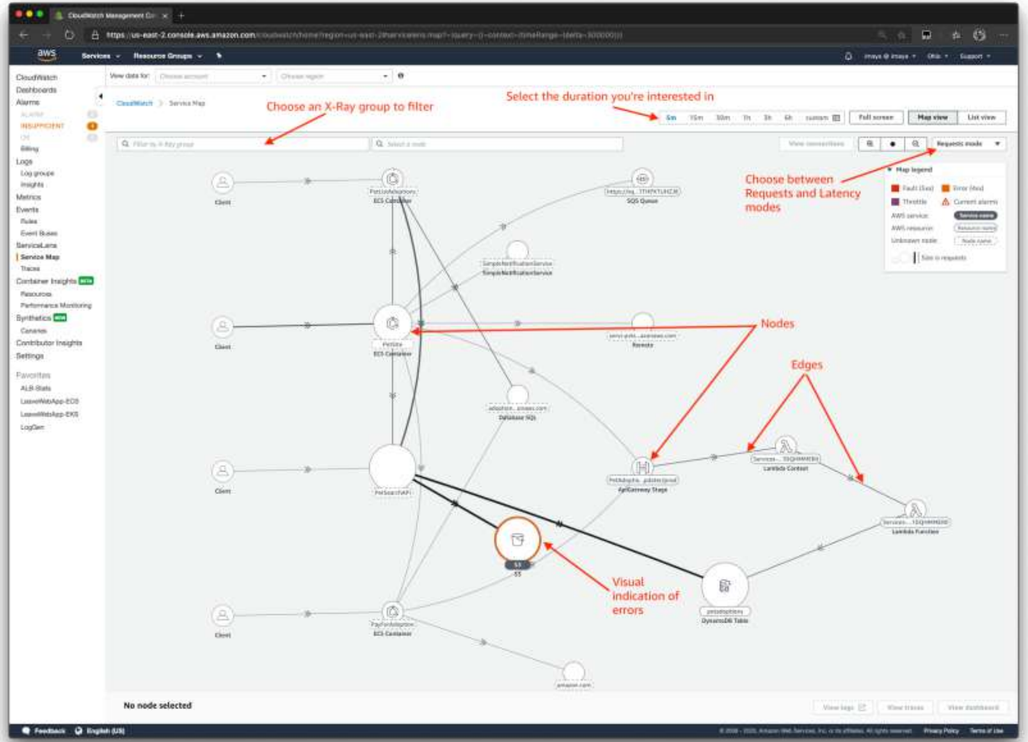
对照实验

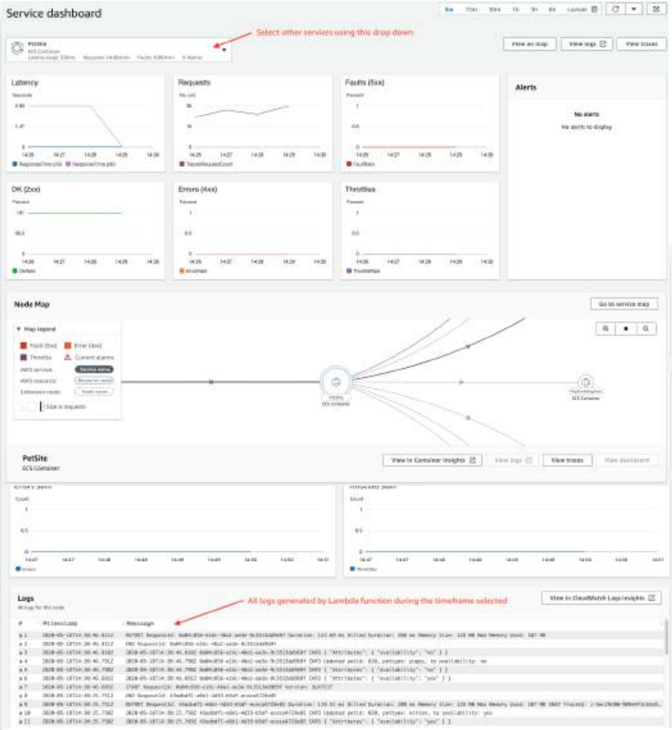
实验场景

安全管控

可观测性

# 系统观测示例 – 服务透视

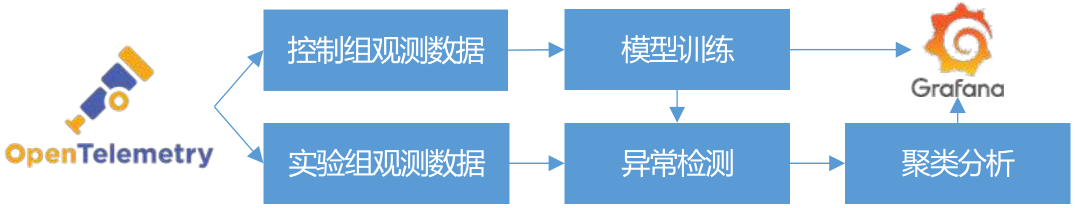

Choose an X-Ray group to filter
Select the duration you're interested in
Choose between Requests and Latency modes
Nodes
Edges
Visual indication of errors
No node selected


Service dashboard
Select other services using this drop down
Latency
Requests
Faults (5xx)
Alerts
OK (2xx)
Errors (4xx)
Throttles
Node Map
PetSite
Logs
All logs generated by Lambda function during the timeframe selected

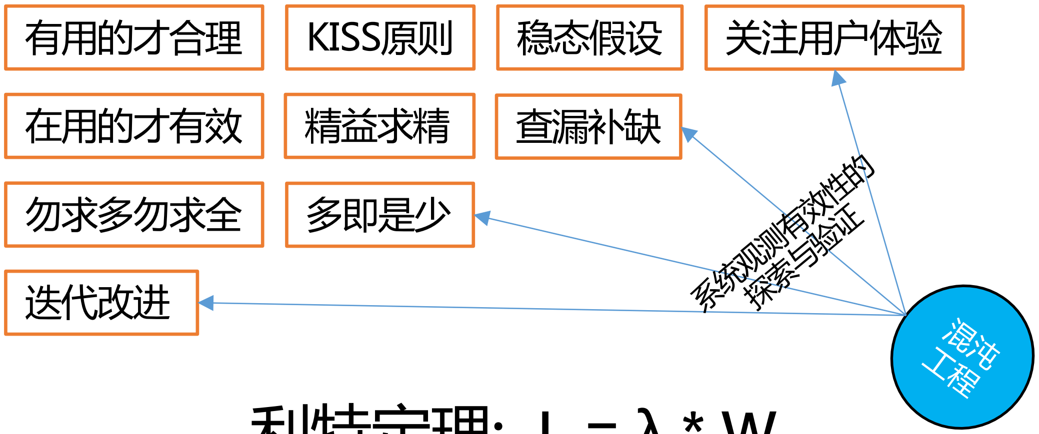
# 系统观测是混沌实验的关键步骤







# 混沌实验可以提升系统观测的有效性




利特定理： L = λ * W

| | |
|---|---|
| The RED Method - Weaveworks | Metrics |
| Four Golden Signals - Google | |
| The USE Method - Brendan Gregg | |

# 下一步

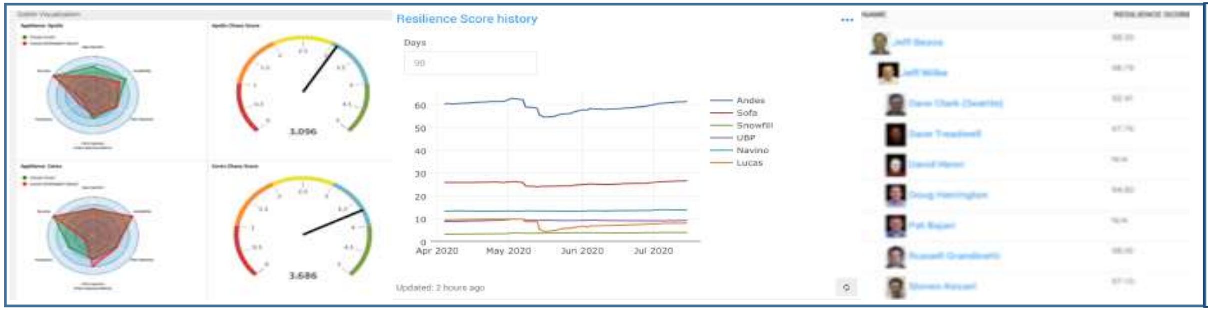
效果与推广

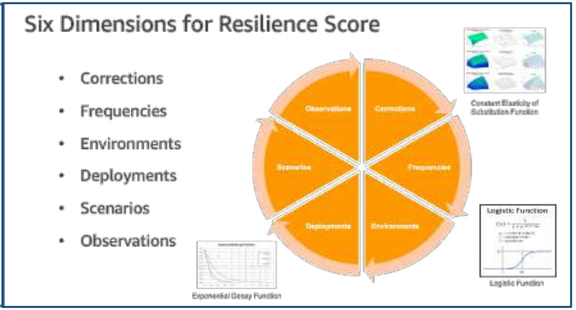
4.0

# 效果与推广

- 韧性（Resilience）是指软件通过适度降级和快速恢复而在遇到故障时保持可用性的**能力**。
- 只能通过在**遇到故障情况时**分析应用程序的行为来衡量软件的韧性。
- 混沌工程实验用于**验证**是否已使用预防故障的**最佳实践**以及软件行为是否已达到韧性目标。
- **韧性分数**是一种报告机制，用于**衡量**服务对故障的韧性。

推荐“混沌工程实践”公众号



GOPS 2021
shenzhen

# Thanks

高效运维社区
开放运维联盟

荣誉出品