混沌工程与落地

chenshaowen.com 2023.10.17

主要内容

- 混沌产生 – 背景介绍
- 混沌工程 – 提供技术手段
- 混沌工程的实践 – 价值的哪里
- 后续计划 – 落地携手共建
- Q&A

背景 - 混沌学科的产生

一只南美洲亚马逊河流域热带雨林中的蝴蝶，偶尔扇动几下翅膀，可以在两周以后引起美国得克萨斯州的一场龙卷风。

最早是由美国气象学家爱德华·洛伦兹在20世纪60年代初期提出的。

1961年,洛伦兹在使用数学模型研究天气系统时,发现了一个非线性微分方程中的微小误差能导致计算结果的极大变化。这个发现展示了混沌状态的一个重要特征——对初值极敏感。

1975年,同为气象学家的James Yorke和Tien-Yien Li将这个敏感依赖于初始条件的概念命名为"蝴蝶效应”,并进行了进一步的研究,“蝴蝶效应”这一概念才真正为人们所认知和理解。

背景 - 混沌工程的产生

- 2008 年 Netflix 准备将其服务从物理机迁移到 AWS

在这个过程中，为了提高系统的稳定性，开发出工具杀 EC2 实例。
Netflix 在多年稳定性的实践历程中，总结了混沌工程的理念。

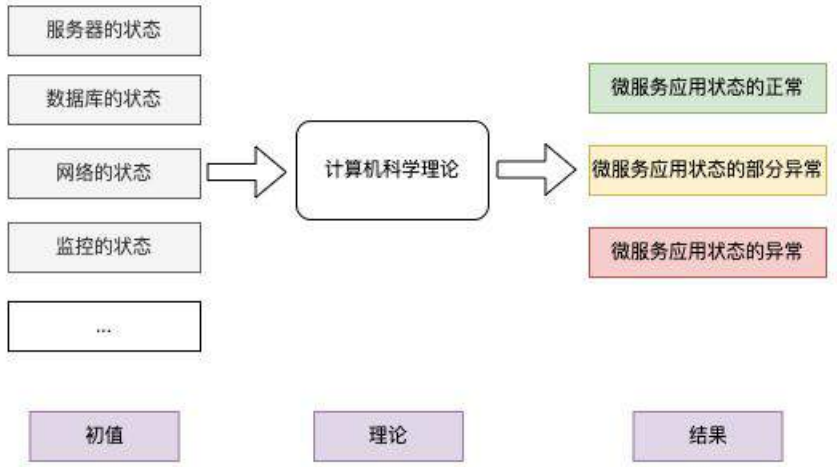
- Chaos Monkey

避免失败的最好办法就是经常失败。通过主动破坏自身环境，来发现系统的弱点。频繁的故障演练使开发团队能从问题中学习经验，从而对服务集群的稳定性有更高的重视。

如同一只野生、武装的猴子，放在数据中心，来造成随机的破坏。https://github.com/Netflix/chaosmonkey

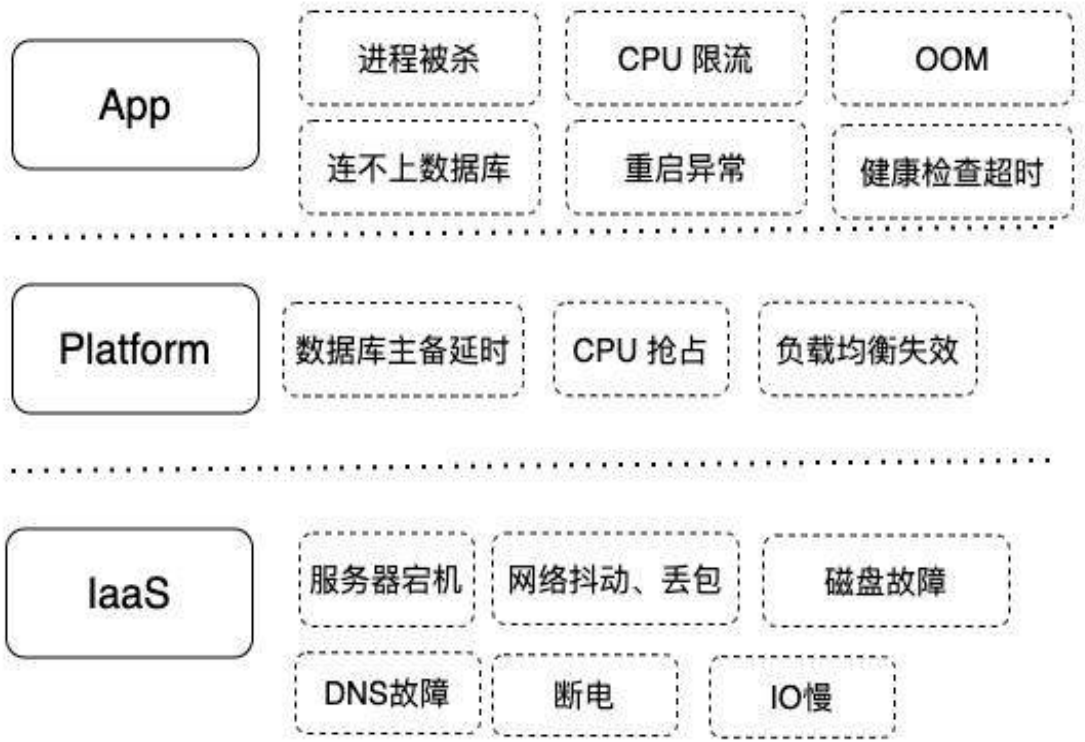
背景 - 微服务与混沌的关系

- 相同点
  1. 复杂非线性连接
  2. 敏感依赖性，一个微服务的变化会影响关联的其他微服务，与混沌敏感依赖初始条件类似
  3. 可能产生巨大的累计误差，蝴蝶效应类似
- 异同点
  1. 不确定性：微服务 < 典型混沌系统
  2. 微服务强调独立性，而混沌系统强调互相依赖



微服务应用
具不具备混沌的特征？

进行混沌工程实验的必要性



? 无处不在的故障

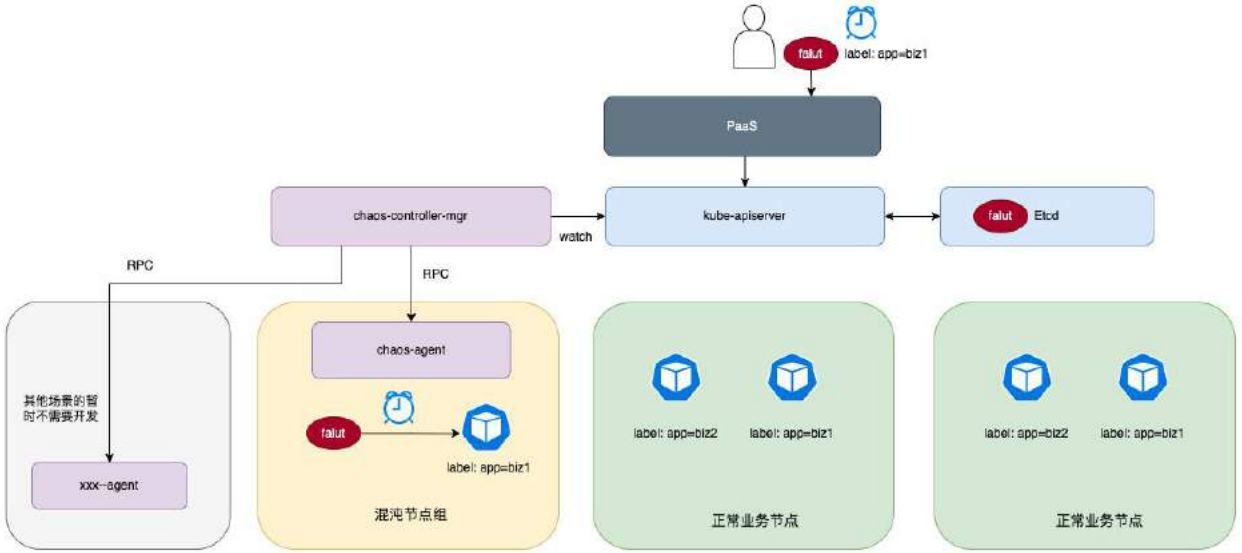
我们所处的基础设施环境、软件架构越来越复杂，故障已经不可避免

✓ 接受故障、拥抱故障、主动制造故障

被动等待 -> 主动出击

- 基础设施问题不便复现
  - *“我怀疑是 CPU 不够”*
  - *“我怀疑是网络有问题”*
  - *“我怀疑连不上数据库”*
  - *“我怀疑....”*
- 不用猜，你只需要一次混沌工程实验

混沌工程



- chaos-agent

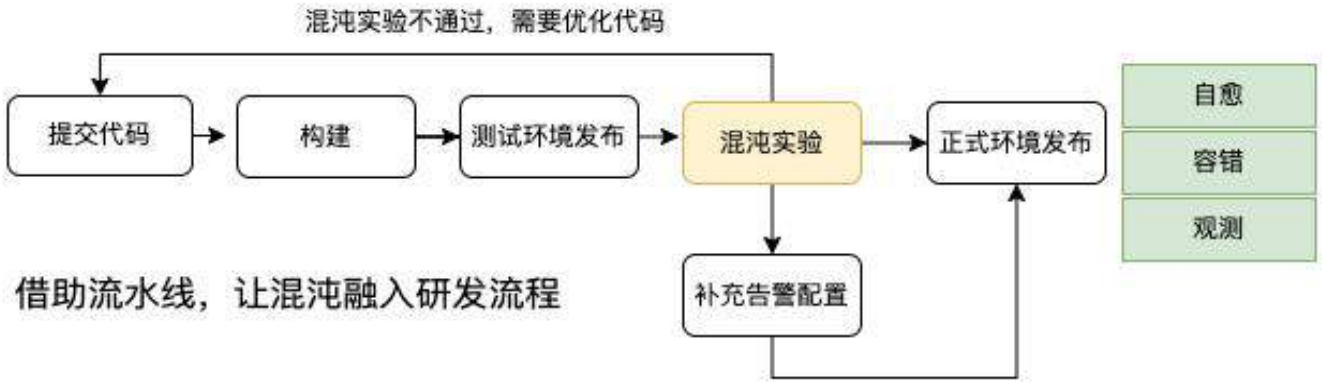
从 chaos-mesh 剥离的探针

- chaos-controller-mgr

自研的基于 K8s 的 Operator，控制两个对象 Network 和 Stress，提供三种故障注入能力

融入流水线，提供通用的编排能力

没有工具、产品的实施和约束，规范是很难落地的。



混沌工程 - 功能

- 目前部署的集群

  xxxx

  xxxxxx

- 已经集成

  【集群管理】- 【工作负载】- 【更多】- 【故障注入】

- 故障类型

  CPU 加压

  网络丢包

  网络断网

混沌工程 - 原理

CPU 加压

Pod 断网

```
ping 14.119.104.254
PING 14.119.104.254 (14.119.104.254) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
```

Pod 丢包

```
ping 14.119.104.254
PING 14.119.104.254 (14.119.104.254) 56(84) bytes of data.
64 bytes from 14.119.104.254: icmp_seq=11 ttl=48 time=29.5 ms
--- 14.119.104.254 ping statistics ---
12 packets transmitted, 5 received, 58.3333% packet loss, time 11144ms
rtt min/avg/max/mdev = 16.755/22.350/29.500/4.314 ms
```

步骤:

1. 提交故障注入请求

2. controller 处理请求，找到 Pod 所在的节点的 agent 调用 RPC

3. agent 找到容器所在的空间，在节点上写入故障

4. 定时时间到或暂停注入，清理故障

故障:

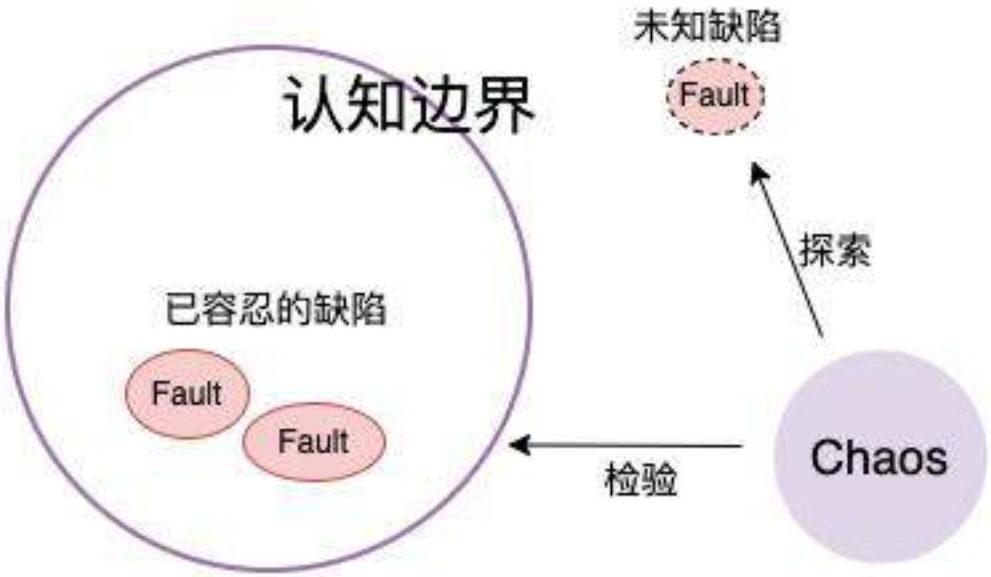
CPU - Stress 命令注入

网络 - Iptables、IpSet 命令劫持网络空间流量，通过 TC 控制

混沌工程实践 - 能带来的收益

- 揭露生产系统中未知的缺陷，提高系统的稳定性

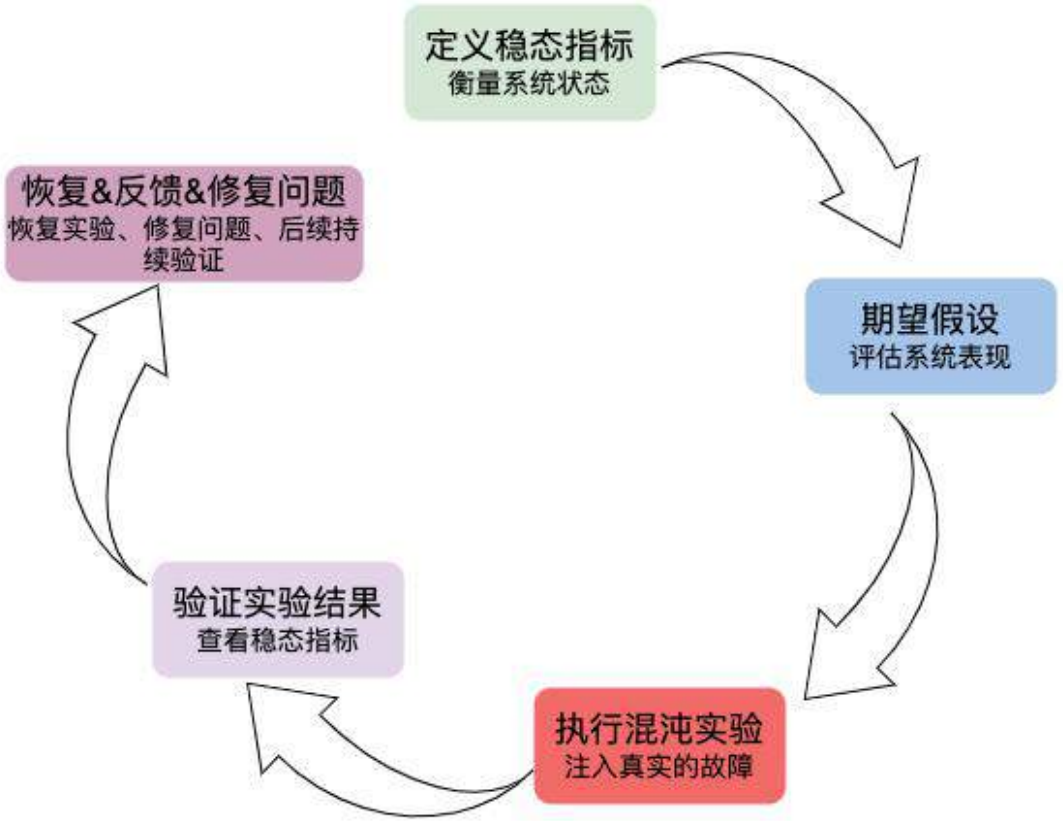
如果确定混沌实验会出现问题，那么该实验就没有任何意义



关键点:

- 对外 - 发现未知的缺陷，告警缺失、强依赖、非高可用等
- 对内 - 不是制造问题，而是有预期的检验

混沌工程实施的步骤



第一步，定义稳态指标

吞吐量、错误率、延时百分位等

第二步，期望假设

按照相关人员对系统的认知，评估表现

第三步，执行混沌实验

通过一定的手段，将故障注入系统

第四步，观察稳态指标

是否符合预期，记录数据

第五步，恢复实验，总结

恢复现场，修复发现的问题，继续实验

混沌工程实践 - 成熟度模型 CEMM

实验成熟度等级	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	并有工具支持交互式的比对实验组和控制组
故障注入场景	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	可以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

混沌工程实践 - 应用的韧性

- 冗余设计

  重要的中间件，两地三中心，允许其中至少一个断连

- 过载保护

  当请求激增时，能够采用限流、拒绝服务等措施保护整体不受损

- 服务降级能力

  在服务能力不够的情况下，能有有所取舍，保障重点服务不受影响

- 去中心化设计

  不允许出现中心化的节点

可以左摇右摆，但是依然能保持平衡

后续计划 - 混沌平台侧

- 建立目标标准

异常的感知能力要求

容忍一段时间一定的丢包 - 5% 30min

容忍一段时间的 CPU 压力 - 限流 30%，5min，预留弹性节点扩容

- 探索性实验

测试集群，随机常态化演练

- 提供一些应用侧改进的 Case 和通用手段

以便业务快速增加应用的韧性

9 月份随机测试总结

现象	推荐措施
应用断网了，各项指标正常	快速失败原则，该 crash 还是得 crash
应用重启、crash 了，无人关注	配置应用级别的 Metric 告警
错误日志告警缺失	查询日志时快捷创建关键字告警
应用 limit 设置太高	req 保护自己、limit 保护大家，合理设置，KAE 已经有推荐值
不用的应用及时清理	副本设置为 0

对测试环境不够重视，测试环境影响的是开发人员，正式环境影响的是外部用户，他们都很重要。

应该保持工作流的畅通：本地 -> 测试环境 -> (灰度环境) -> 正式环境。 态度与责任心是一种习惯，而不是只面向正式环境。

后续计划 - 业务侧

- 补齐测试环境的感知能力
  - 告警
  - 日志
  - 监控
- 韧性改造
  - 访问慢，服务慢、对象存储慢、中间件慢
  - 强依赖解耦

Q&A