GOTC

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

OPEN SOURCE , OPEN WORLD

「CNCF 云原生专场」

本期议题：云原生混沌工程平台 Chaos Mesh 原理与实践

周志强 2021年07月10日

关于我

GOTC

周志强

GitHub: STRRL
PingCAP R&D

Chaos Mesh 的核心开发成员, 拥有多年在云原生领域的开发经验. 开源爱好者, Chaos Mesh Committer.
目前专注于 Chaos Engineering, 致力于构建一站式的 Chaos Engineering Platform.

概览

1. 混沌工程与云原生应用
2. Chaos Mesh 设计与实现
3. 使用案例

混沌工程与云原生应用

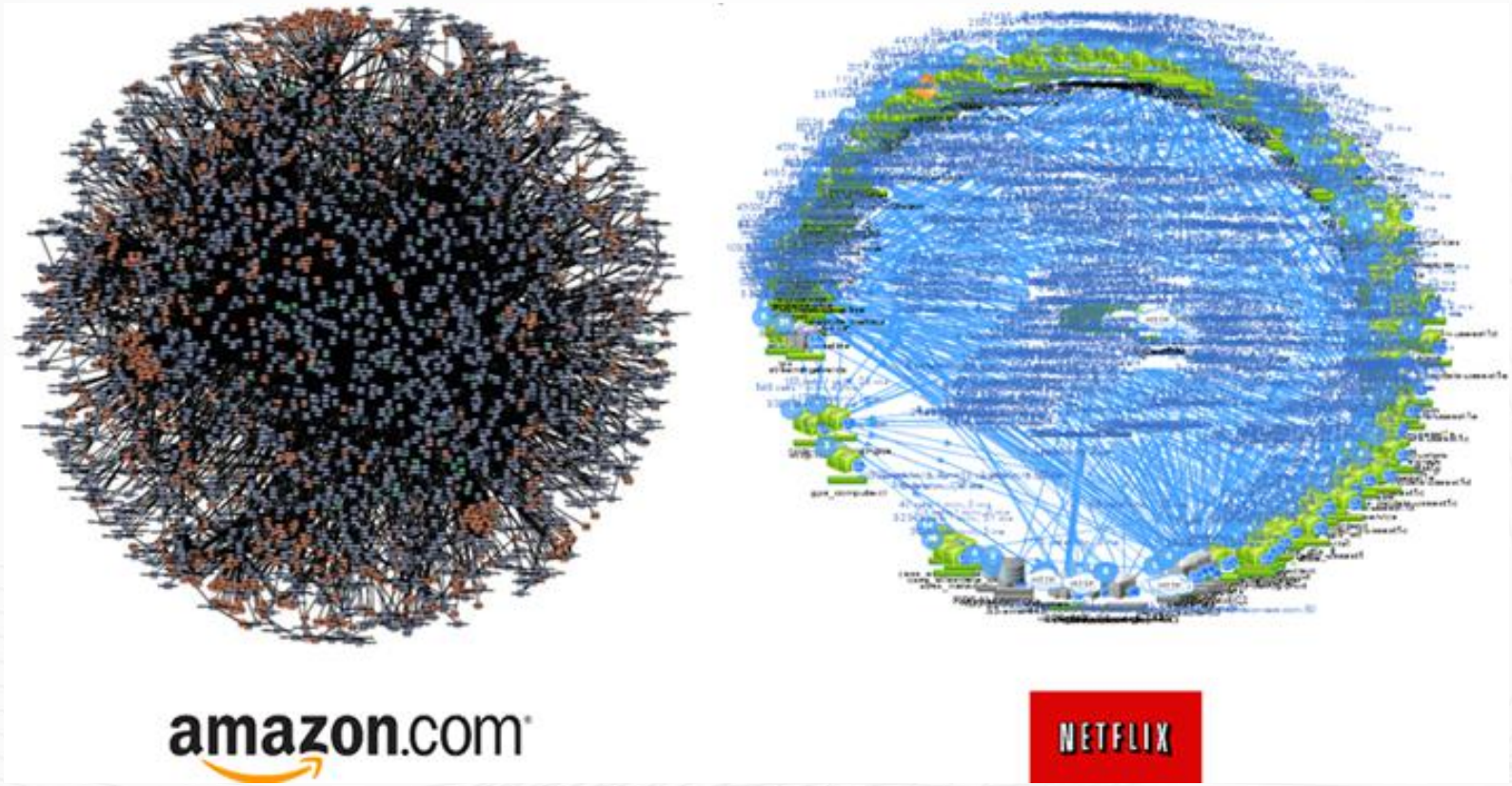
GOTC

故障随时随地都可能发生！

混沌工程与云原生应用

GOTC

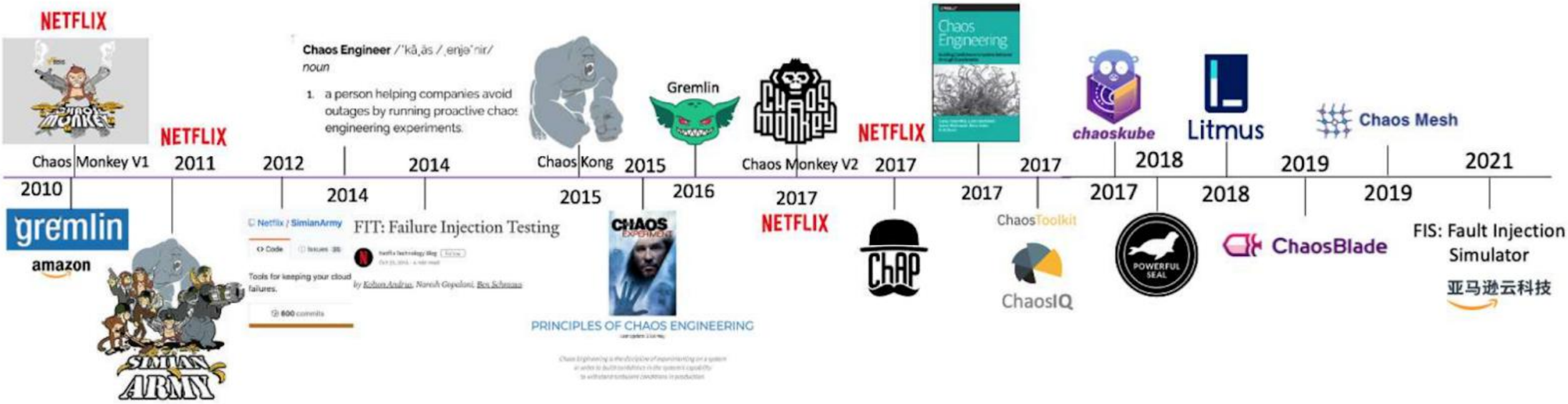
分布式系统 - 越来越复杂



混沌工程与云原生应用

GOTC

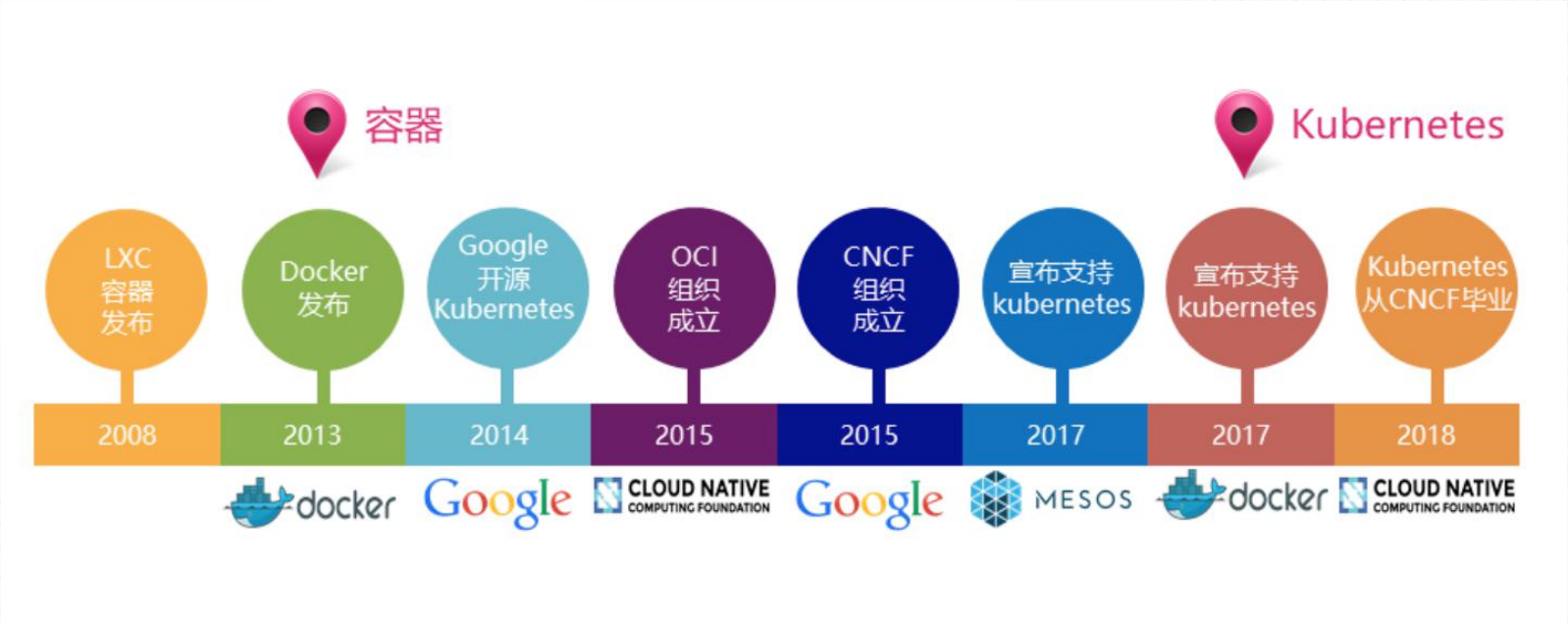
混沌工程的演进史



GOTC

云原生应用

- 容器化
- Kubernetes
- 弹性
  - 快速伸缩
  - 为失败设计
  - 优雅降级



混沌工程的目的

混沌工程是在分布式系统上进行实验的学科，目的是建立对系统抵御生产环境中失控条件的能力以及信心。

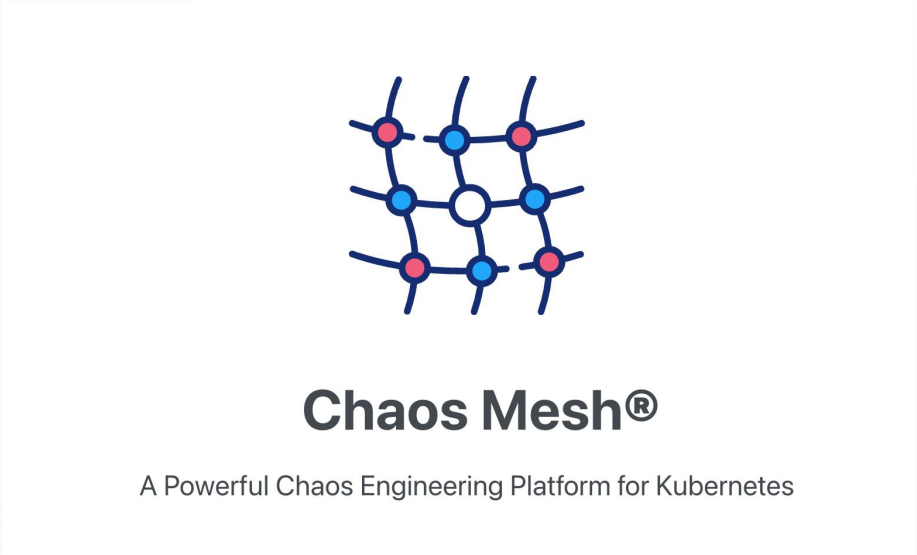
韧性 自愈

引用自混沌工程原则：https://principlesofchaos.org/zh/

Chaos Mesh 设计与实现

GOTC

Chaos Mesh 是什么



- 开源云原生混沌工程平台
- 混沌实验的注入与编排
- 易用，支持可视化操作

https://chaos-mesh.org/

https://github.com/chaos-mesh/chaos-mesh

GOTC

什么是云原生

- *云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用。云原生的代表技术包括**容器**、服务网格、微服务、**不可变基础设施**和**声明式 API**。*
- *这些技术能够构建容错性好、易于管理和便于观察的松耦合系统。结合可靠的自动化手段，云原生技术使工程师能够轻松地对系统作出频繁和可预测的重大变更。*

1. CNCF Charter https://github.com/cncf/foundation/blob/master/charter.md

Chaos Mesh 设计与实现

GOTC

云原生技术的优点

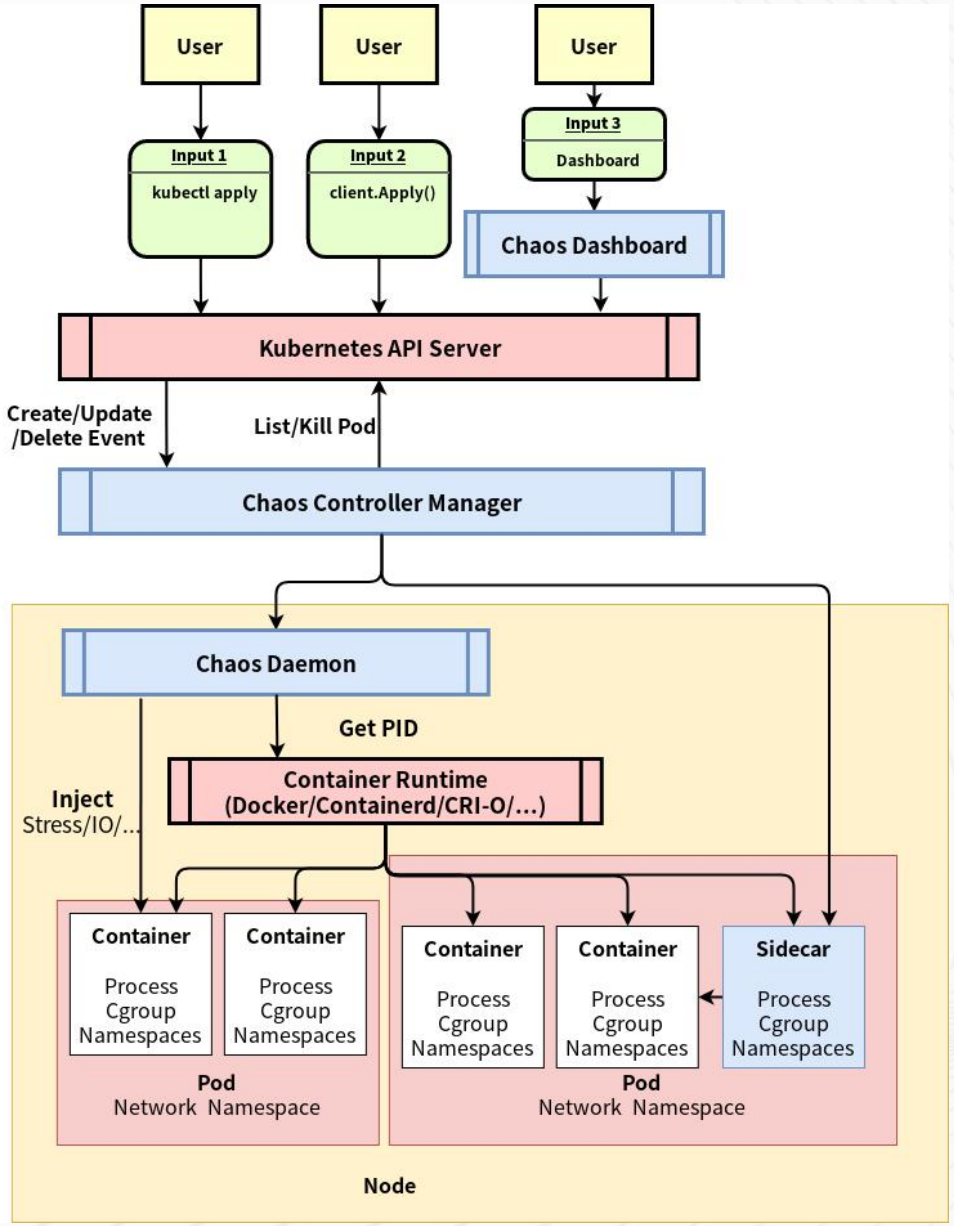
- 声明式 API
  - 明确地表示预期状态
  - 控制器可靠地实施预期状态
  - 方便地控制与观测
- 容器
  - 资源隔离，限制爆炸半径
  - 通过特权容器注入故障
- 不可变基础设施
  - 稳定地分发 Release
  - 便于试用，安装，部署

Chaos Mesh 设计与实现

GOTC

Chaos Mesh 整体架构

- 用户输入、观测
- 监听资源变化，进行注入/恢复
- 在具体节点上进行故障注入



声明故障

```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: Schedule
metadata:
 name: network-partition
spec:
 schedule: '@every 60s'
 type: NetworkChaos
 historyLimit: 5
 concurrencyPolicy: Forbid
 networkChaos:
   action: delay
   mode: one
   selector:
     namespaces:
     - default
     labelSelectors:
       app: backend
   target:
     mode: one
     selector:
       namespaces:
         - default
       labelSelectors:
         app: redis
   duration: 30s
   direction: to
   delay:
     latency: 10ms
     correlation: "100"
     jitter: 0ms
```

- 从 backend 和 redis 的副本中各随机挑选出一个 pod;

Chaos Mesh 设计与实现

GOTC

声明故障

```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: Schedule
metadata:
 name: network-partition
spec:
 schedule: '@every 60s'
 type: NetworkChaos
 historyLimit: 5
 concurrencyPolicy: Forbid
 networkChaos:
   action: delay
   mode: one
   selector:
     namespaces:
     - default
     labelSelectors:
       app: backend
   target:
     mode: one
     selector:
       namespaces:
         - default
       labelSelectors:
         app: redis
   duration: 30s
   direction: to
   delay:
     latency: 10ms
     correlation: "100"
     jitter: 0ms
```

- 从 backend 和 redis 的副本中各随机挑选出一个 pod;
- 在 backend 到 redis 方向的网络流量上注入网络延迟;

声明故障

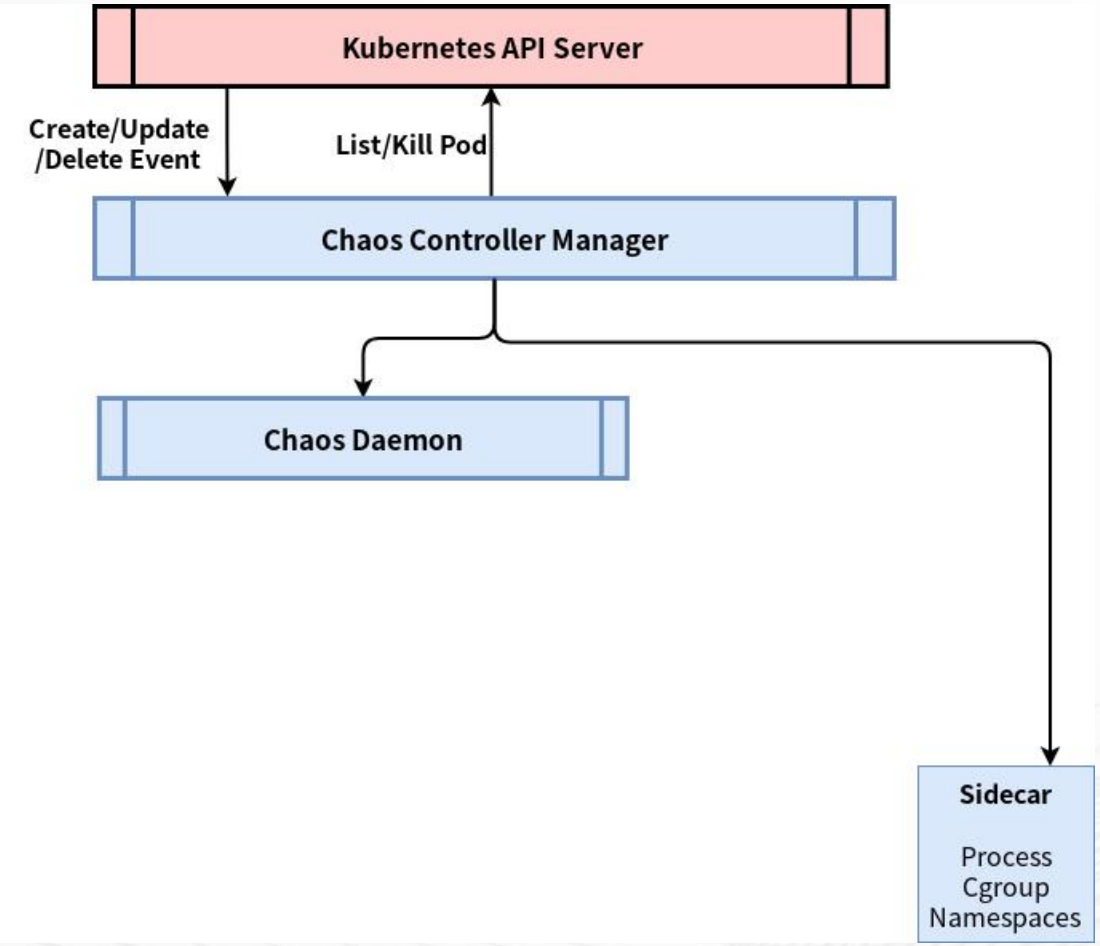
```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: Schedule
metadata:
 name: network-partition
spec:
 schedule: '@every 60s'
 type: NetworkChaos
 historyLimit: 5
 concurrencyPolicy: Forbid
 networkChaos:
   action: delay
   mode: one
   selector:
     namespaces:
     - default
     labelSelectors:
       app: backend
   target:
     mode: one
     selector:
       namespaces:
         - default
       labelSelectors:
         app: redis
   duration: 30s
   direction: to
   delay:
     latency: 10ms
     correlation: "100"
     jitter: 0ms
```

- 从 backend 和 redis 的副本中各随机挑选出一个 pod;
- 在 backend 到 redis 方向的网络流量上注入网络延迟;
- 持续 30 秒，每 60 秒执行一次.

Chaos Mesh 设计与实现

GOTC

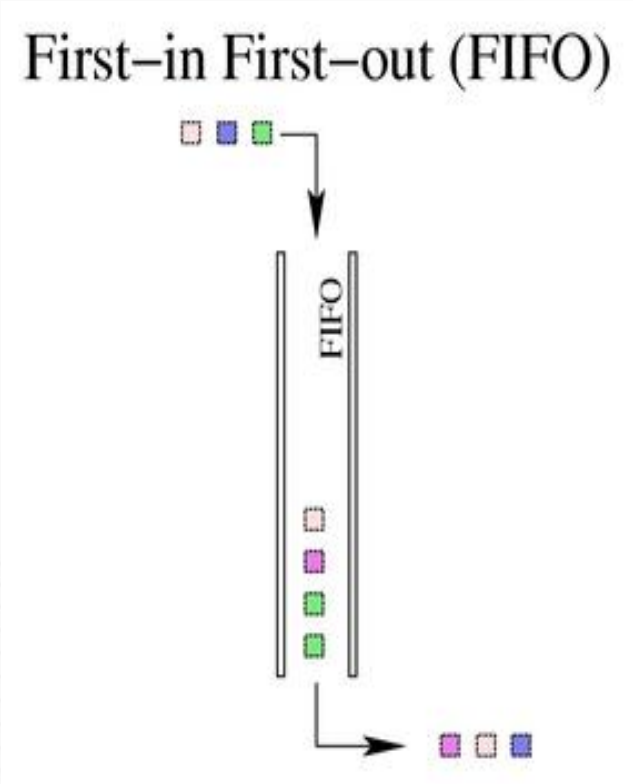
监听资源的变化



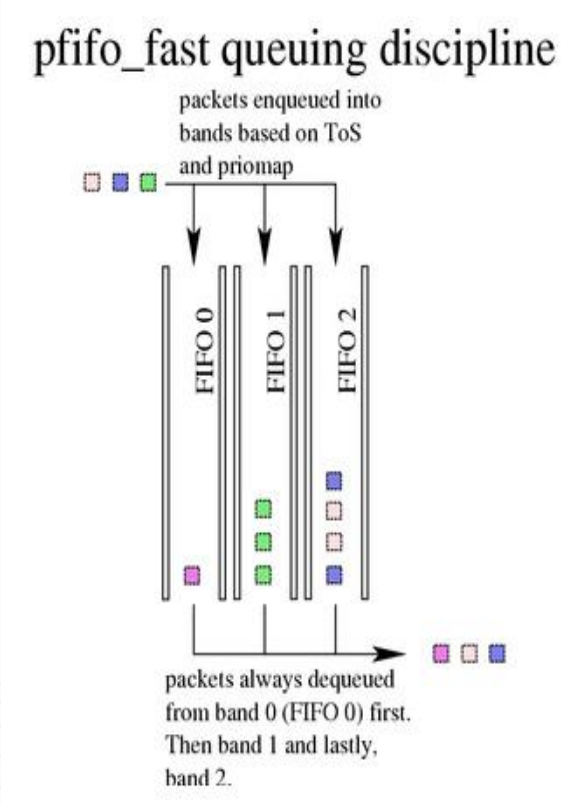
- 监听 PodChaos, NetworkChaos… 等资源的 创建/更新/删除
- 决定当前该 注入 / 恢复 / 等待
- （进行简单的注入，比如 PodKill）
- 向 Chaos Daemon / Sidecar 发送请求

GOTC

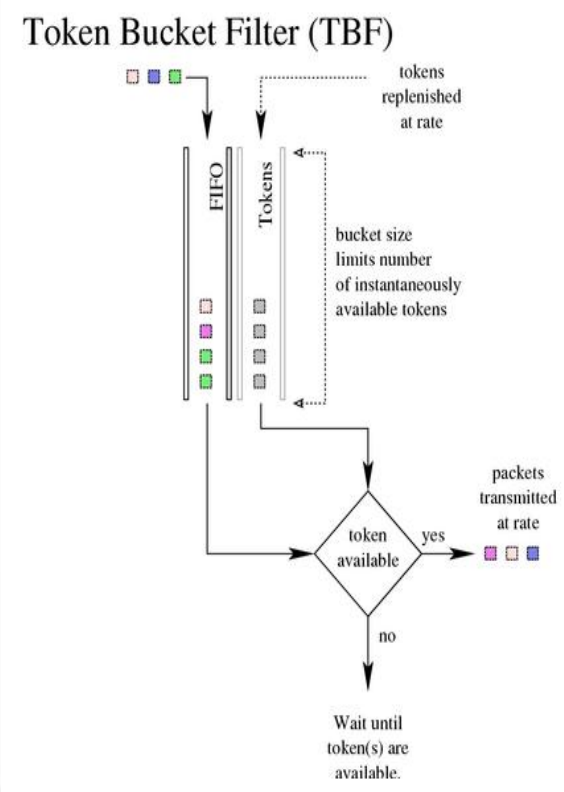
如何注入延迟?



FIFO



pfifo_fast / prio



tbf

Chaos Mesh 设计与实现

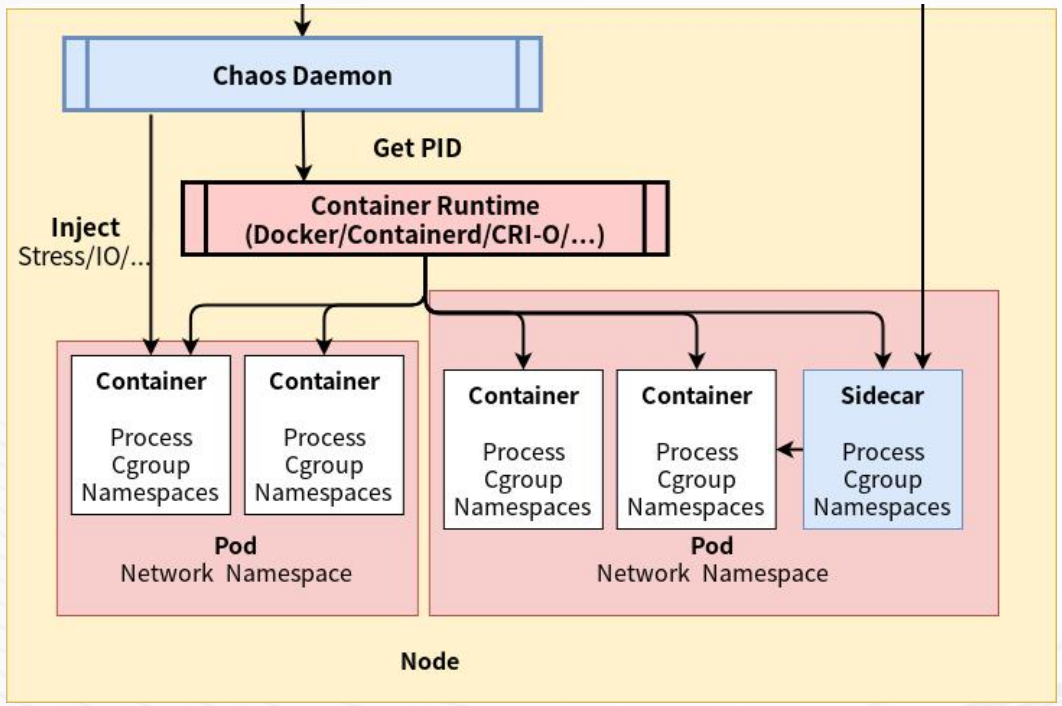
GOTC

Reconcile

Resource	Physical Implementation
ReplicaSet	Pod
Pod	Container
ConfigMap	key/value in etcd
...	...
NetworkChaos	ipset / iptables / tc qdisc / ...

Chaos Mesh 设计与实现

Chaos Daemon 注入



- Container 的实体：
  - 进程
  - Namespace　控制 } 隔离
    可见性
  - Cgroup　限制
    资源

- 注入的实质
  - 侵入 Namespace / Cgroup
  - 进行干扰、注入

Chaos Mesh 设计与实现

GOTC

侵入 namespace

- nsexec --net=/proc/xxx/ns/net iptables -A …..
- nsexec --net=/proc/xxx/ns/net tc qdisc add ….
- nsexec --pid=/proc/xxx/ns/pid stress-ng …
- nsexec --mnt=/proc/xxx/ns/mnt inject ...

Chaos Mesh 设计与实现

GOTC

方便的安装，部署

通过 helm 安装:

```
helm repo add chaos-mesh https://charts.chaos-mesh.org && \
helm install chaos-mesh chaos-mesh/chaos-mesh --namespace=chaos-testing
```

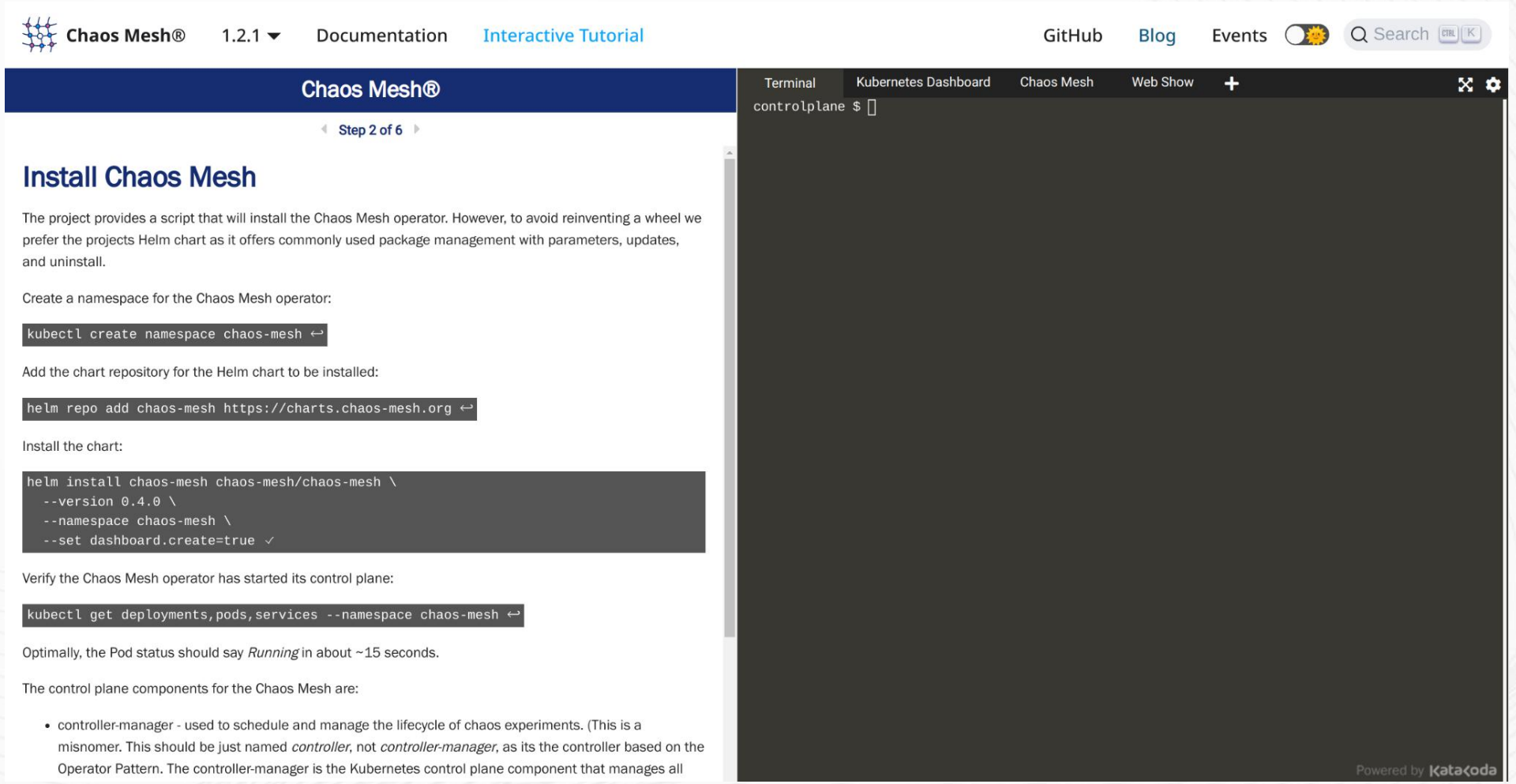
Chaos Mesh 设计与实现

GOTC

方便的安装，部署

在浏览器中试用:

https://chaos-mesh.org/interactive-tutorial



使用案例

GOTC

小鹏汽车

- 使用 **Chaos Mesh**
  - 测试监控告警系统
  - 模拟低网络延迟情况
- 覆盖监控测试系统 **100%**
- 通过使用 **Chaos Mesh** 减少路试成本 **90%**

使用案例

网易伏羲实验室

FUXI-Lab

- 测试组件(redis rabbitmq scheduler) 3+
- 目前发现的潜在问题 20+

broker node宕，引发connection Channel shutdown情形之一 	就目前出现的异常片段时机，初步判断是当前宕掉的broker node是客户端连接的一个endpoint	本地模拟混沌测试策略，300s随机kill一个pod，同时每隔5s，发布一条消息到broker队列		由于连接重连和consumer重启，目前无法清晰评估对客户端模块的影响
broker node宕，引发connection Channel shutdown情形之二 	同上，另外，上述判断主要根据节点状态以及实时日志的实际情况	本地模拟混沌测试策略，300s随机kill一个pod，同时每隔5s，发布一条消息到broker队列		出现概率较高
Error: {:aborted, {:no_exists, [:rabbit_vhost, [{{:vhost, :"$1", :_}, [], [:"$1"]}]}}} root@rabbitmq-qa-ha-0:/# rabbitmqctl list_vhosts Listing vhosts ... Error: {:aborted, {:no_exists, [:rabbit_vhost, [{{:vhost, :"$1", :_}, [], [:"$1"]}]}}}		600s随机kill一个pod	这种情况目前看是down掉的broker node还没起来或者上没有join到集群导致	
启动失败 		600s随机kill一个pod	这个问题，通过引入initContainer，对PV下的mnesia db进行清理操作，目前镜像yaml已更新，且运行后没有在遇到此类故障	

GOTC

THANKS

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE