云原生社区
Cloud Native Community

SHANGHAI

云原生社区Meetup第一期

UCAN

技术沙龙

主办方 云原生社区 Cloud Native Community

承办方 UCLOUD优刻得 云原生社区上海站 Cloud Native Community Shanghai

赞助商 tetrate 万科海外 Broadview

2020-11-28 上海站

杨可奥

PingCAP

Chaos Mesh核心开发者

当前 Chaos Mesh 的 maintainer。在混沌工程的实践和实现上拥有一定经验和见解。除了 Chaos Mesh 之外还维护有多个受欢迎的开源项目，如 pprof-rs。

云原生社区Meetup
第一期·上海站 × UCAN 技术沙龙

云原生社区 Cloud Native Community

Chaos Mesh
让应用与混沌在 Kubernetes 上共舞

演讲人：杨可奥 PingCAP

目录

混沌工程的动机

事故，任何时候都可能发生

云原生社区
Cloud Native Community

AWS

Summary of the AWS Service Event in the Sydney Region

We'd like to share more detail about the AWS service disruption that occurred this past weekend in the AWS Sydney Region. The service disruption primarily affected EC2 instances and their associated Elastic Block Store ("EBS") volumes running in a single Availability Zone.

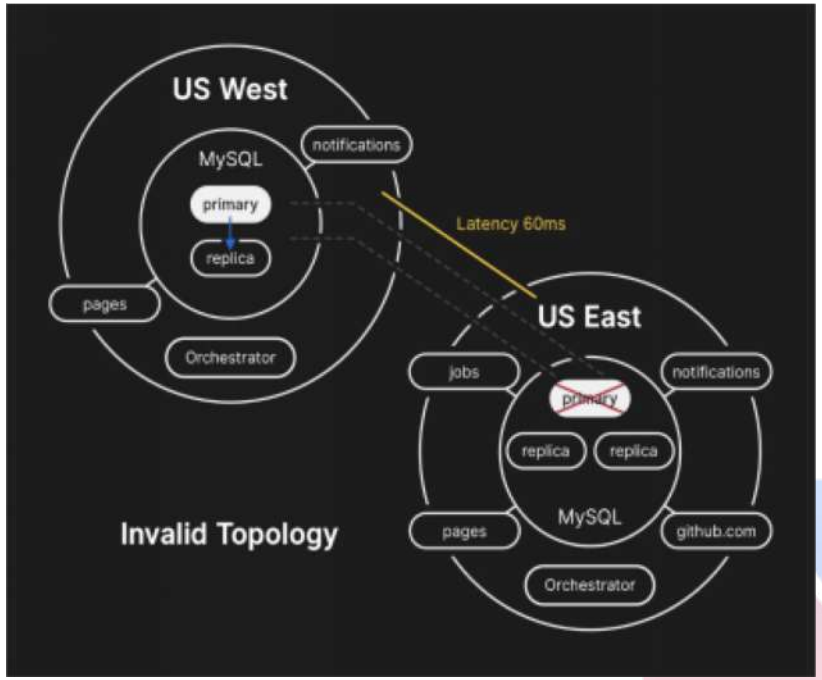
Loss of Power

At 10:25 PM PDT on June 4th, our utility provider suffered a loss of power at a regional substation as a result of severe weather in the area. This failure resulted in a total loss of utility power to multiple AWS facilities. In one of the facilities, our power redundancy didn't work as designed, and we lost power to a significant number of instances in that Availability Zone.

事故，任何时候都可能发生

Github

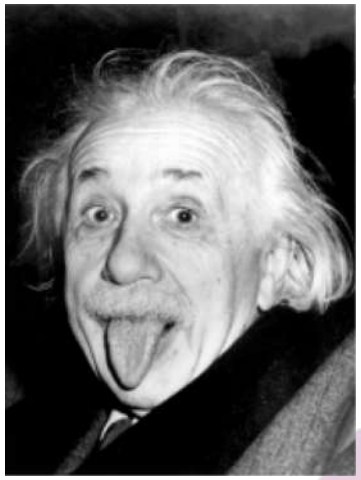
At 22:52 UTC on October 21, routine maintenance work to replace failing 100G optical equipment resulted in the loss of connectivity between our US East Coast network hub and our primary US East Coast data center. Connectivity between these locations was restored in 43 seconds, but this brief outage triggered a chain of events that led to 24 hours and 11 minutes of service degradation.



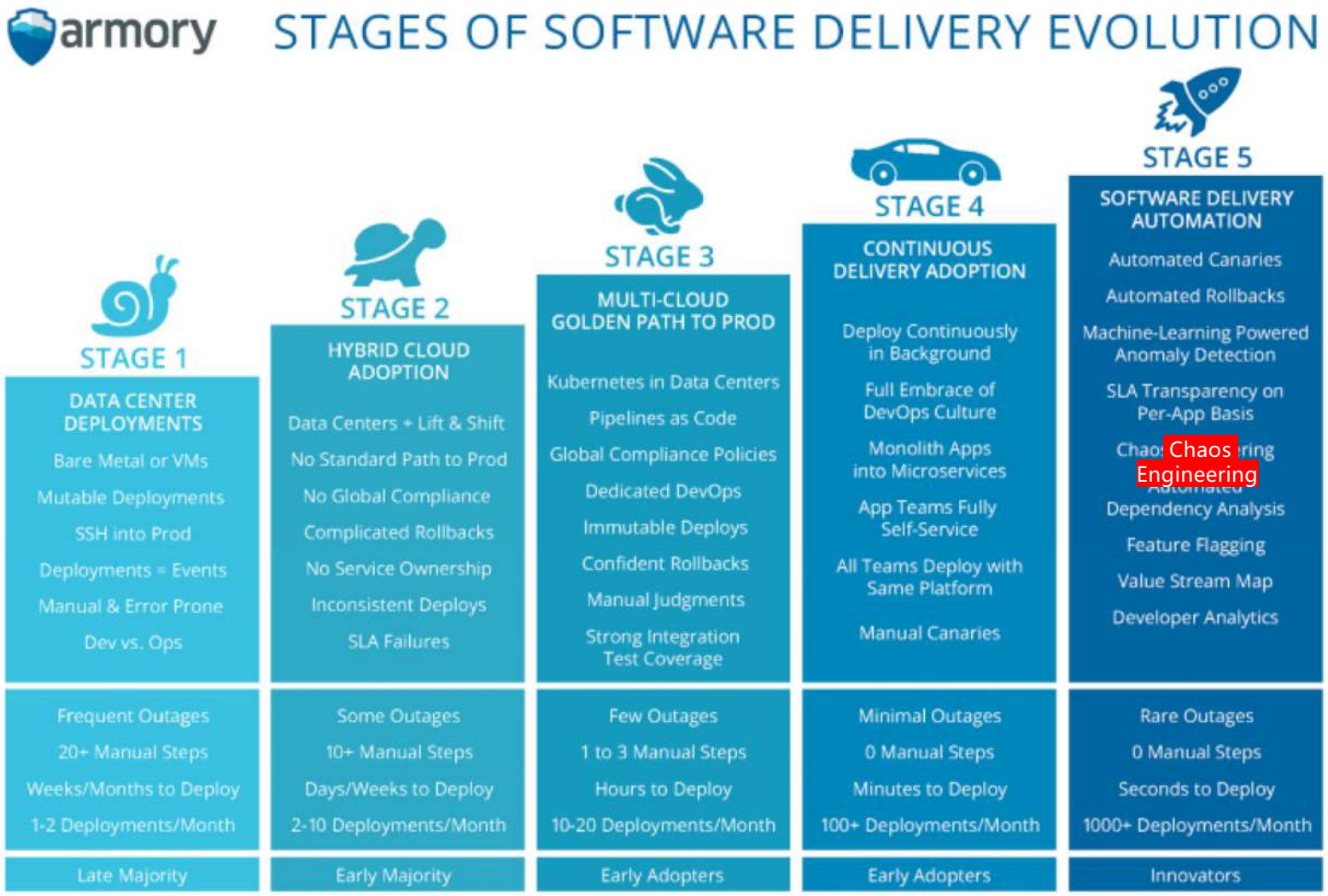
关于混沌，我们能知道很多

科学的研究方法 → 我们的软件没有混沌现象👍

- 明确目标，问题
- 作出假设
- 进行尝试和实验
- 观察现象
- 分析和总结



混沌工程正在受到重视


armory
STAGES OF SOFTWARE DELIVERY EVOLUTION
STAGE 1
DATA CENTER DEPLOYMENTS
Bare Metal or VMs
Mutable Deployments
SSH into Prod
Deployments = Events
Manual & Error Prone
Dev vs. Ops
Frequent Outages
20+ Manual Steps
Weeks/Months to Deploy
1-2 Deployments/Month
Late Majority
STAGE 2
HYBRID CLOUD ADOPTION
Data Centers + Lift & Shift
No Standard Path to Prod
No Global Compliance
Complicated Rollbacks
No Service Ownership
Inconsistent Deploys
SLA Failures
Some Outages
10+ Manual Steps
Days/Weeks to Deploy
2-10 Deployments/Month
Early Majority
STAGE 3
MULTI-CLOUD GOLDEN PATH TO PROD
Kubernetes in Data Centers
Pipelines as Code
Global Compliance Policies
Dedicated DevOps
Immutable Deploys
Confident Rollbacks
Manual Judgments
Strong Integration Test Coverage
Few Outages
1 to 3 Manual Steps
Hours to Deploy
10-20 Deployments/Month
Early Adopters
STAGE 4
CONTINUOUS DELIVERY ADOPTION
Deploy Continuously in Background
Full Embrace of DevOps Culture
Monolith Apps into Microservices
App Teams Fully Self-Service
All Teams Deploy with Same Platform
Manual Canaries
Minimal Outages
0 Manual Steps
Minutes to Deploy
100+ Deployments/Month
Early Adopters
STAGE 5
SOFTWARE DELIVERY AUTOMATION
Automated Canaries
Automated Rollbacks
Machine-Learning Powered Anomaly Detection
SLA Transparency on Per-App Basis
Chaos Engineering
Dependency Analysis
Feature Flagging
Value Stream Map
Developer Analytics
Rare Outages
0 Manual Steps
Seconds to Deploy
1000+ Deployments/Month
Innovators

混沌工程正在受到重视

云原生社区
Cloud Native Community

Observability and Analysis - Chaos Engineering (7)

Chaos Mesh 2,547
Cloud Native Computing Foundation (CNCF)

Chaos Toolkit 1,110
ChaosIQ

Chaosblade 3,119
Alibaba Cloud
MCap: $832.24B

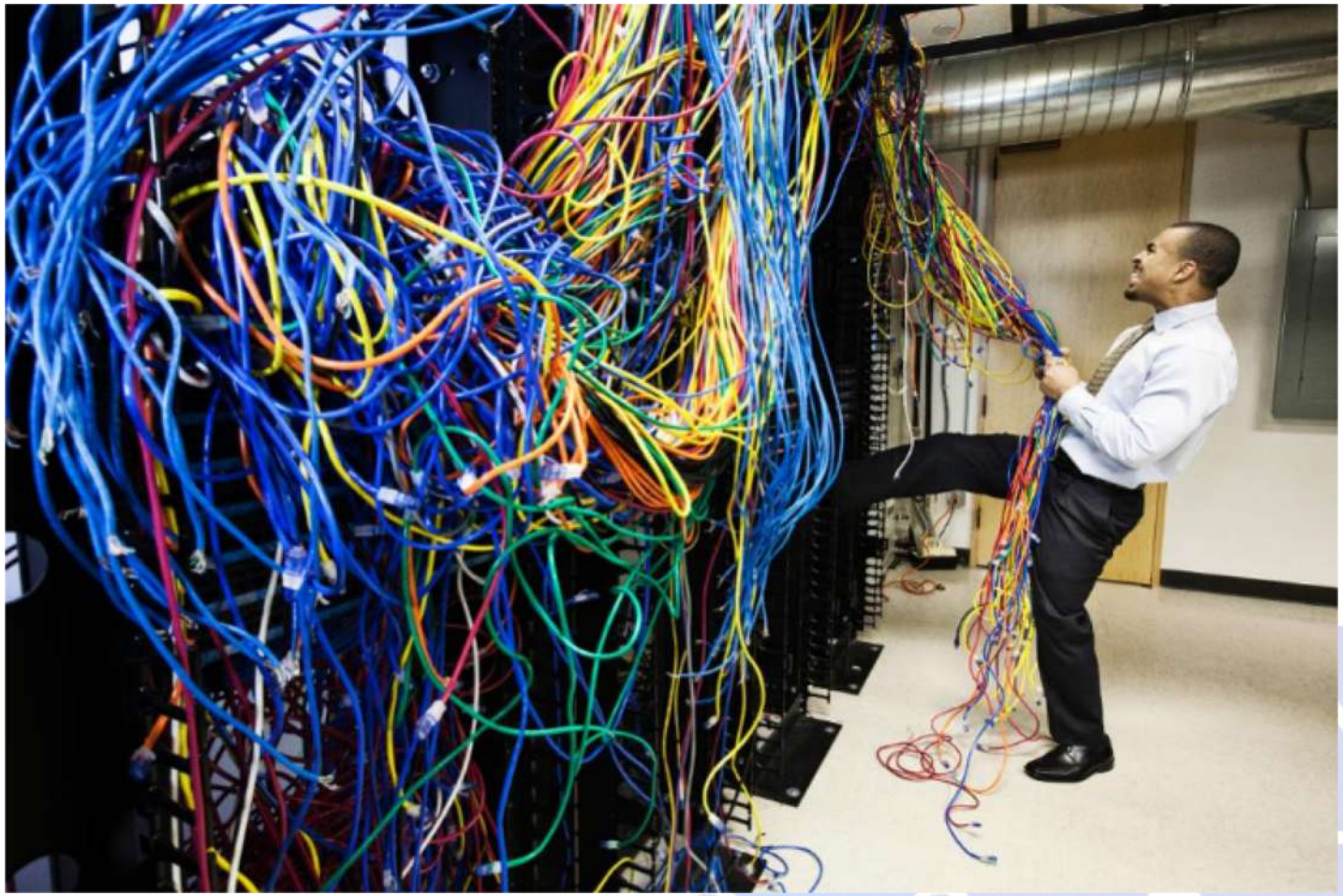
chaoskube 1,246
chaoskube

Gremlin Funding: $26.75M
Gremlin

Litmus 1,267
Cloud Native Computing Foundation (CNCF)

PowerfulSeal 1,380
Bloomberg

混沌实验？听上去很简单



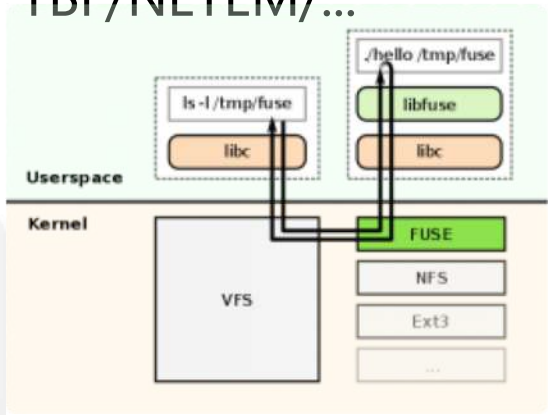
混沌实验？听上去很简单

1. 5 分钟入门混沌工程 —— 脚本随机杀进程
2. 10 分钟入门混沌工程 —— 脚本随机杀 Pod
3. …
4. 那网络故障呢？磁盘故障呢？恢复呢？如何控制作用范围？

这是一件复杂的事

IP Set

TBF/NETEM/...



这是一件困难的事

1. 天然的隔离性和安全性
2. Go 的线程模型与 namespace 机制难以融合
3. 要求运行时注入和恢复
4. 和内核打交道通常都是困难的😈

云原生社区
Cloud Native Community

Kubernetes 上的混沌工程方案 Chaos Mesh

Cloud Native

- 在 Kubernetes 上运行，被测对象也运行在 Kubernetes 上
- 测试的最小单元是 Pod 或 Container
- 使用 Helm 一键部署

```
chaos-mesh on impl-687 [!?] via v1.14.3
❯ helm install chaos-mesh helm/chaos-mesh --namespace=chaos-testing
NAME: chaos-mesh
LAST DEPLOYED: Thu Jul 16 14:21:19 2020
NAMESPACE: chaos-testing
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Make sure chaos-mesh components are running
   kubectl get pods --namespace chaos-testing -l app.kubernetes.io/instance=chaos-mesh
```

友善的接口

- 实验是作为 Kubernetes Custom Resource 管理的

```
chaos-mesh on ⚡ external-latency via 🐻 v1.14.3
❯ kubectl get Pods
NAME                        READY   STATUS    RESTARTS   AGE
ubuntu-6bd98f48c4-s6zdr     1/1     Running   2          4d22h
ubuntu2-bbccd55fd-mv6xm     1/1     Running   1          4d1h

chaos-mesh on ⚡ external-latency via 🐻 v1.14.3
❯ kubectl get StressChaos
NAME       AGE
burn-cpu   3d23h
```

```
podFailureChaos := &v1alpha1.PodChaos{
    ObjectMeta: metav1.ObjectMeta{
        Name:      "timer-failure",
        Namespace: ns,
    },
    Spec: v1alpha1.PodChaosSpec{
        Selector: v1alpha1.SelectorSpec{
            Namespaces: []string{
                ns,
            },
            LabelSelectors: map[string]string{
                "app": "timer",
            },
        },
        Action: v1alpha1.PodFailureAction,
        Mode:   v1alpha1.OnePodMode,
    },
}
err = cli.Create(ctx, podFailureChaos)
```

友善的接口

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=61 time=80.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=61 time=125 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=61 time=85.6 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=61 time=90.7 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=61 time=177 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=61 time=108 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=61 time=184 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=61 time=232 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=61 time=167 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=61 time=226 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=61 time=246 ms
64 bytes from 8.8.8.8: icmp_seq=13 ttl=61 time=79.9 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=61 time=78.6 ms
64 bytes from 8.8.8.8: icmp_seq=15 ttl=61 time=78.9 ms
64 bytes from 8.8.8.8: icmp_seq=16 ttl=61 time=79.7 ms
64 bytes from 8.8.8.8: icmp_seq=17 ttl=61 time=80.1 ms
64 bytes from 8.8.8.8: icmp_seq=18 ttl=61 time=225 ms
64 bytes from 8.8.8.8: icmp_seq=19 ttl=61 time=134 ms
64 bytes from 8.8.8.8: icmp_seq=20 ttl=61 time=150 ms
64 bytes from 8.8.8.8: icmp_seq=21 ttl=61 time=154 ms
64 bytes from 8.8.8.8: icmp_seq=22 ttl=61 time=130 ms
64 bytes from 8.8.8.8: icmp_seq=23 ttl=61 time=94.0 ms
64 bytes from 8.8.8.8: icmp_seq=24 ttl=61 time=229 ms
64 bytes from 8.8.8.8: icmp_seq=25 ttl=61 time=209 ms
64 bytes from 8.8.8.8: icmp_seq=26 ttl=61 time=167 ms
64 bytes from 8.8.8.8: icmp_seq=27 ttl=61 time=173 ms
64 bytes from 8.8.8.8: icmp_seq=28 ttl=61 time=82.3 ms
64 bytes from 8.8.8.8: icmp_seq=29 ttl=61 time=79.1 ms
64 bytes from 8.8.8.8: icmp_seq=30 ttl=61 time=81.1 ms
64 bytes from 8.8.8.8: icmp_seq=31 ttl=61 time=79.9 ms
64 bytes from 8.8.8.8: icmp_seq=32 ttl=61 time=79.2 ms
64 bytes from 8.8.8.8: icmp_seq=33 ttl=61 time=257 ms
64 bytes from 8.8.8.8: icmp_seq=34 ttl=61 time=223 ms
64 bytes from 8.8.8.8: icmp_seq=35 ttl=61 time=245 ms
64 bytes from 8.8.8.8: icmp_seq=36 ttl=61 time=203 ms
64 bytes from 8.8.8.8: icmp_seq=37 ttl=61 time=194 ms
64 bytes from 8.8.8.8: icmp_seq=38 ttl=61 time=85.4 ms
64 bytes from 8.8.8.8: icmp_seq=39 ttl=61 time=145 ms
```

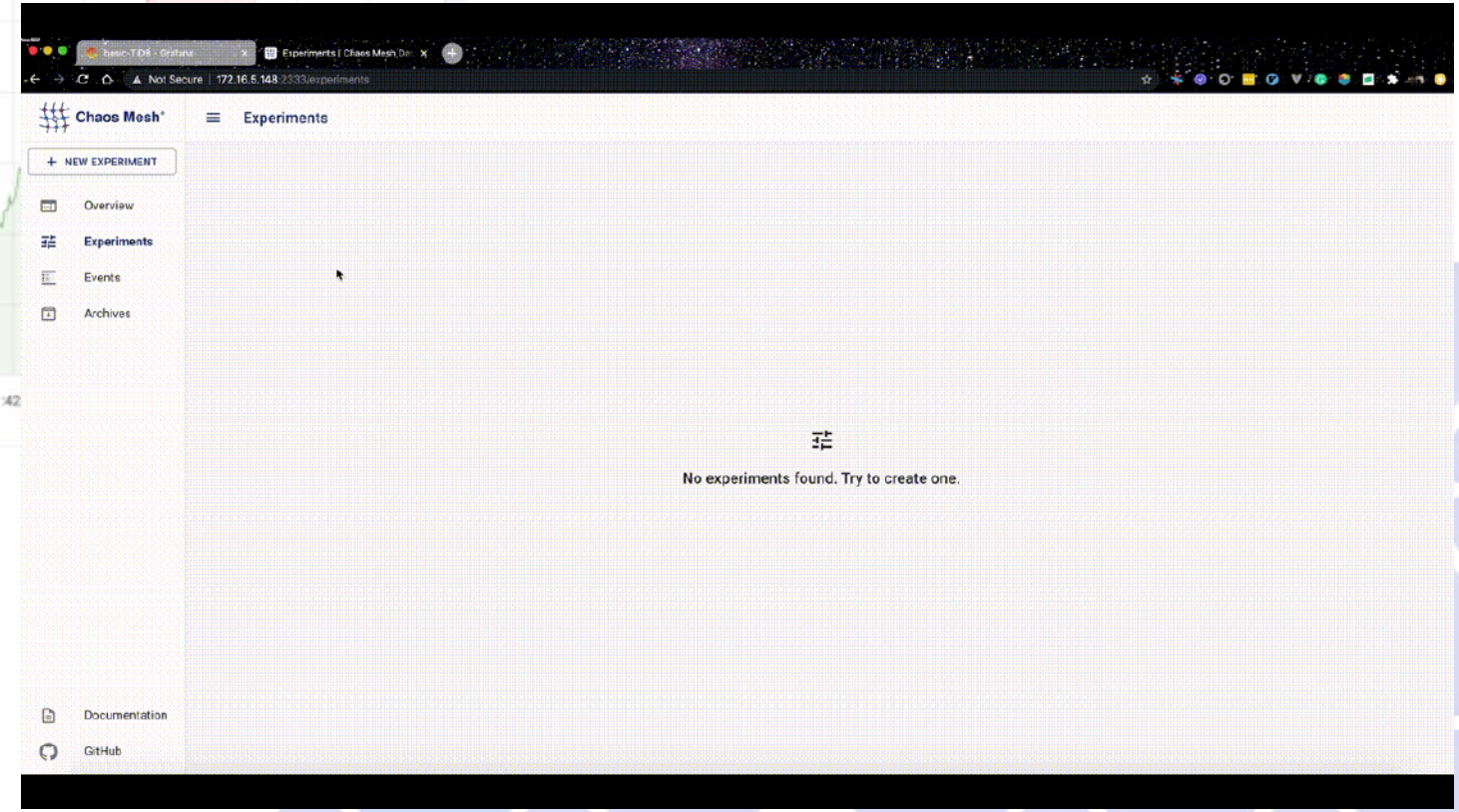
```
Project-2020/k8sTestCase/networkchaos
❯ cat network-delay-with-external-target-example.yaml
apiVersion: pingcap.com/v1alpha1
kind: NetworkChaos
metadata:
  name: network-delay-example
spec:
  action: delay
  mode: one
  selector:
    labelSelectors:
      "app": "ubuntu"
  delay:
    latency: "90ms"
    correlation: "25"
    jitter: "90ms"
  direction: to
  target:
    mode: one
    selector:
      labelSelectors:
        "app": "ubuntu2"
  externalTargets:
    - "8.8.0.0/16"
    - "www.bing.com"
  duration: "10s"
  scheduler:
    cron: "@every 15s"

Project-2020/k8sTestCase/networkchaos
❯
```

强大的工具箱

- PodChaos: kill / fail / ...
- NetworkChaos: delay / lose / dup / partition / ...
- IOChaos: latency / fault / ...
- TimeChaos: clock skew
- KernelChaos: kernel fault injection
- StressChaos: burn cpu and memory
- DNSChaos ....

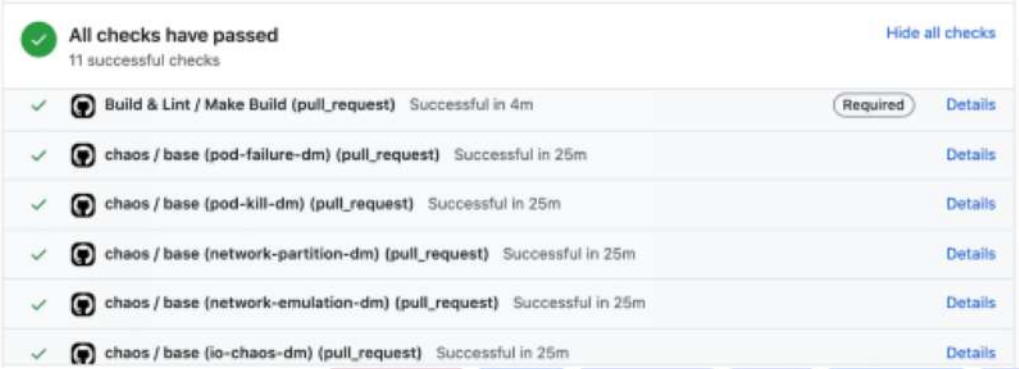
	chaos-mesh(v0.8.0)	chaosmonkey(v2.0.2)	chaosblade(v0.5.0)	chaoskube(v0.19.0)	litmus(v1.3.0)
Platform supported	K8s	VMs/ Container	JVM/Container/ K8s	K8s	K8s
CPU burn	✓	✗	✓	✗	✓
Mem burn	✓	✗	✓	✗	✓
container kill	✓	✓	✓	✗	✓
pod failure	✓	✗	✗	✗	✗
pod kill	✓	✗	✓	✓	✓
network partition	✓	✗	✗	✗	✗
network duplication	✓	✗	✓	✗	✗
network corrupt	✓	✗	✓	✗	✓
network loss	✓	✗	✓	✗	✓
network delay	✓	✗	✓	✗	✓
I/O delay	✓	✗	✓	✗	✗
I/O errno	✓	✗	✓	✗	✗
Disk fill	✓	✗	✓	✗	✓
Disk loss	✓	✗	✓	✗	✓
Time skew	✓	✗	✗	✗	✗
Kernel chaos	✓	✗	✗	✗	✗

Dashboard 和 Grafana 插件

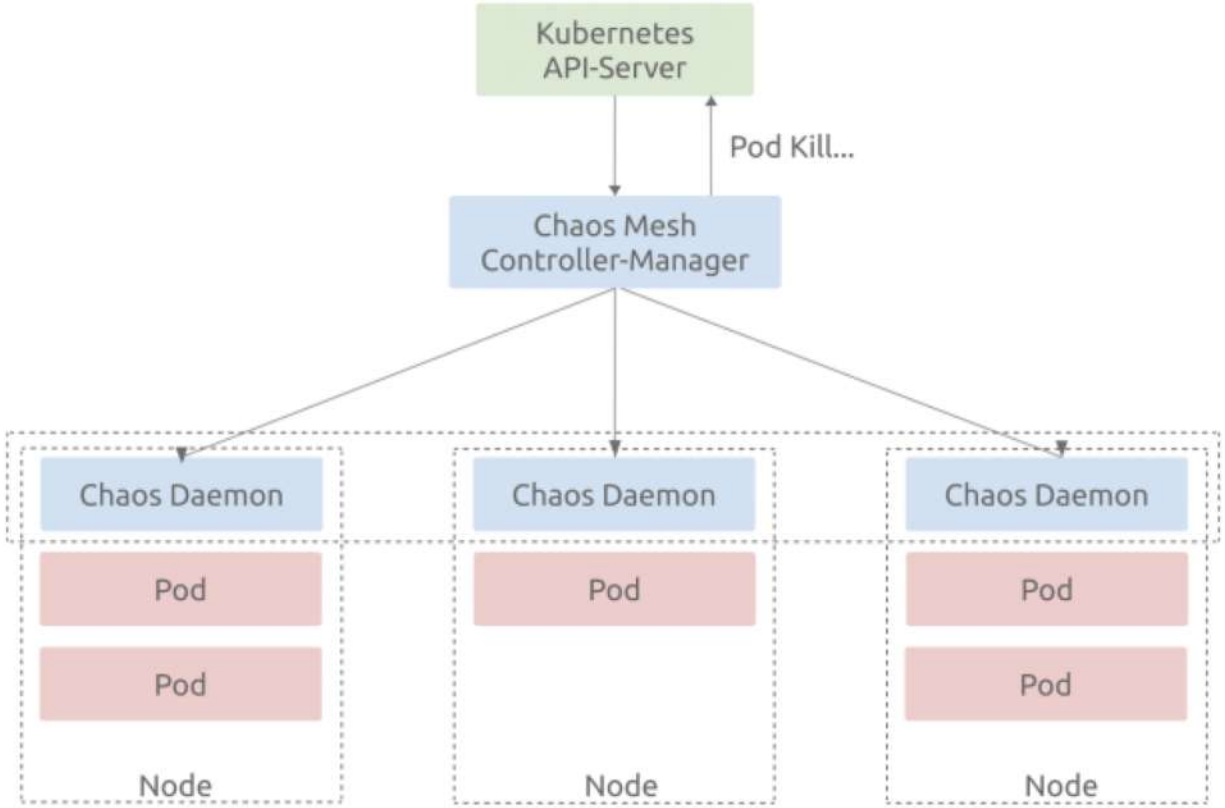


使用方案

- 在生产环境中使用 💣
  - 限制爆炸半径
- 在测试环境、测试集群中使用 🚀
- 在 CI 中使用
  - 使用预先定义的 Github Actions
  - 使用 Kubernetes Client 创建实验

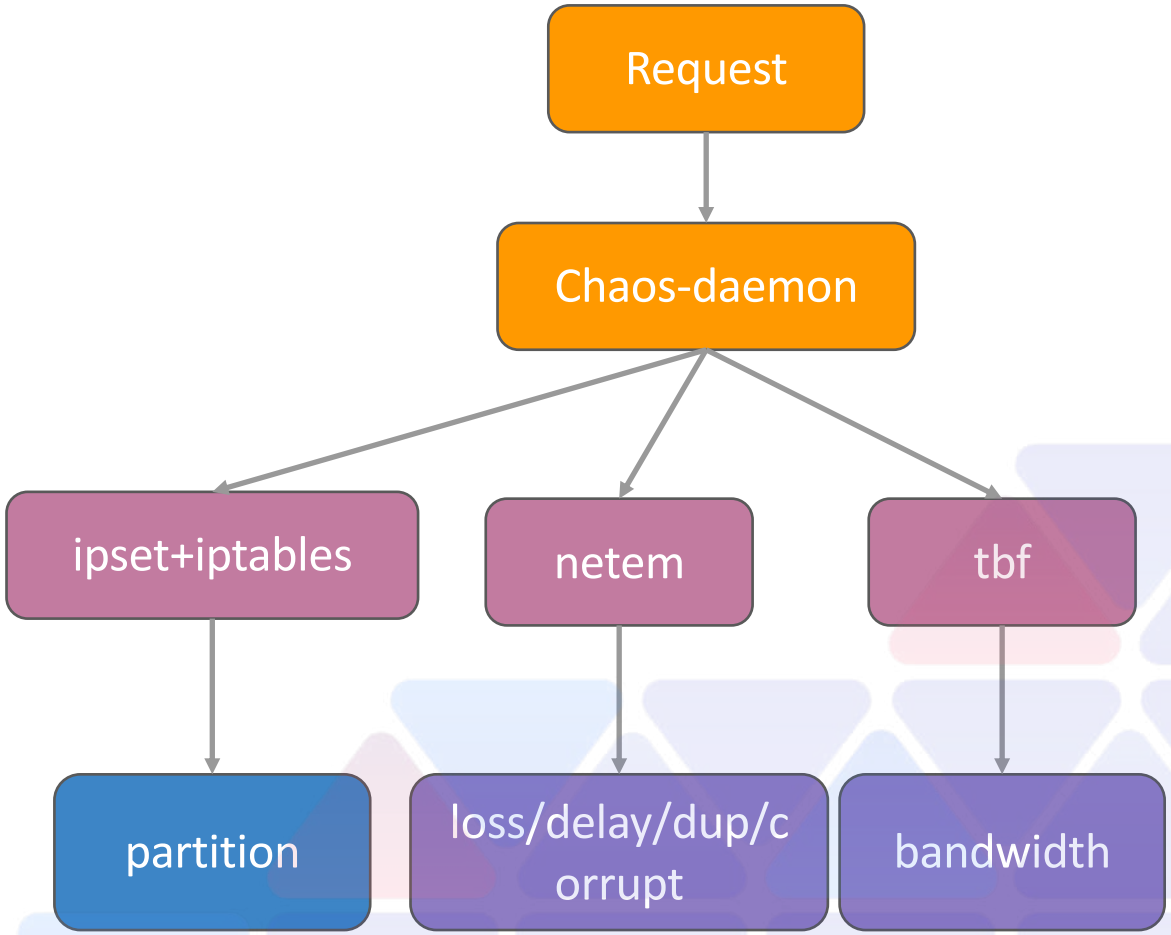


Chaos Mesh 的结构
以 NetworkChaos 为例


Kubernetes
API-Server
Pod Kill...
Chaos Mesh
Controller-Manager
Chaos Daemon
Chaos Daemon
Chaos Daemon
Pod
Pod
Pod
Pod
Pod
Pod
Node
Node
Node

NetworkChaos 实现方法

- Controller 向 chaos-daemon 发送请求
- [Pod network namespace] 设置 ipset 和 iptables
- [Pod network namespace] 设置 qdisc



如何进入目标 Pod 的 Network Namespace

云原生社区
Cloud Native Community

- ~~setns 系统调用~~
- **nsenter 命令 或在其他进程中 setns**
  - **开发、测试更加方便**
  - **使用起来更加简单**
- SideCar 共享 Network Namespace
  - 范围和权限更加可控

Chaos Mesh 使用案例

以 TiDB 为例

云原生社区
Cloud Native Community

- 假设
  - TiDB 使用 Raft 一致性算法构建副本，应当拥有容错的能力
  - 在杀掉一个节点之后，QPS 应当会下降
  - 一段时间之后，QPS会恢复正常
- 运行实验
  - 使得一个节点无法工作（Pod Failure）
- 观察和检验
  - QPS 下降之后却再也没有恢复到实验前的水平
  - 我们找到了一个 Bug ！

以 FUXI-Lab 为例

云原生社区
Cloud Native Community

FUXI-Lab

- Testing components(redis rabbitmq scheduler) **3+**
- Testing bugs **20**

问题描述	问题原因	测试方案	解决方案
官方术语Cluster Network Partition，或Split-Brain		600s随机kill一个pod	参照官方文档，关于"partition handling strategies"部分，涉及三种auto handling策略。这里考虑融入autoheal策略
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
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

Welcome to a “bug-free” world!

Without worrying about

云原生社区
Cloud Native Community

云原生社区Meetup

第一期·上海站

THANKS