Chaos Mesh®

Chaos Mesh 开源说

云原生混沌工程平台

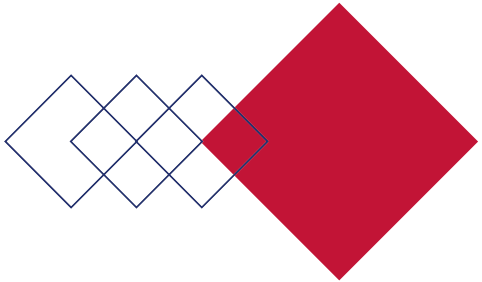
关于我们

姓名：杨可奥
GitHub 地址
：https://github.com/YangKeao/
公司、职位： PingCAP 研发

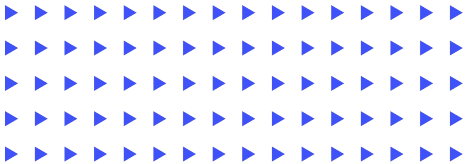
关于我们

姓名：周强
GitHub 地址
：https://github.com/zhouqiang-cl
公司、职位：PingCAP 工程效率负责人，ChaosMesh 负责人

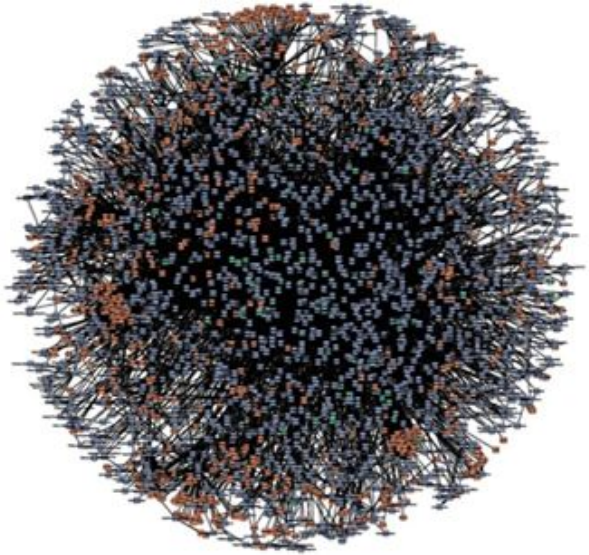
Chaos Mesh®



Part I - 混沌工程和 Chaos Mesh



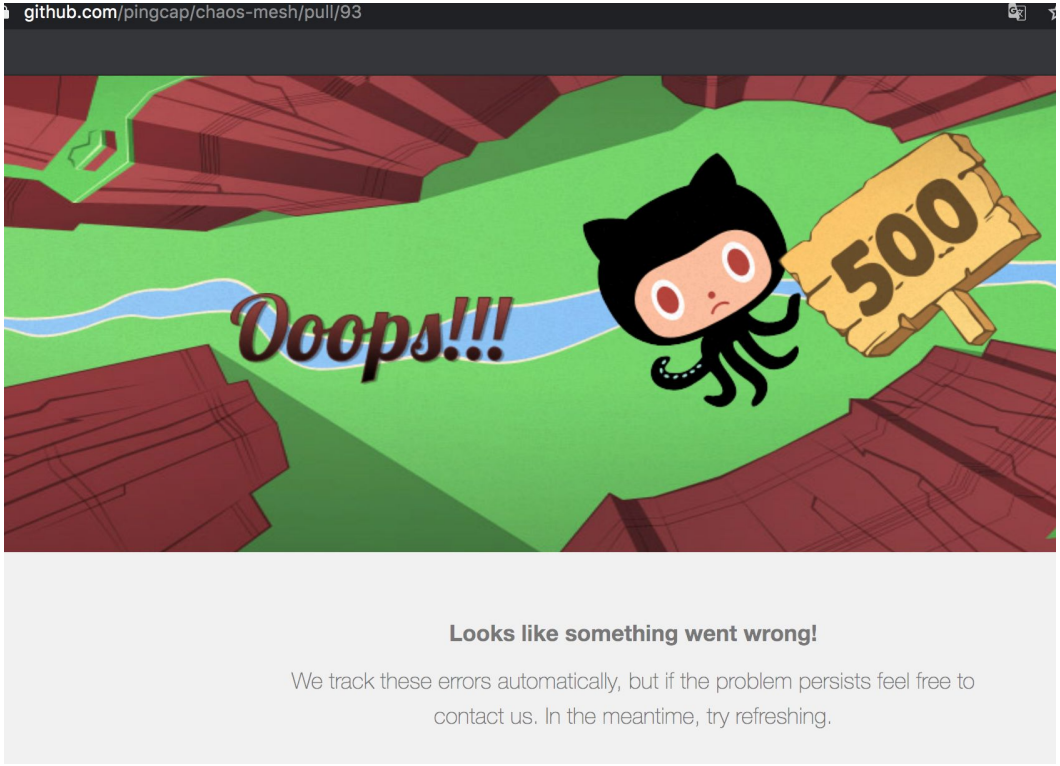
分布式系统越来越复杂



amazon.com

NETFLIX

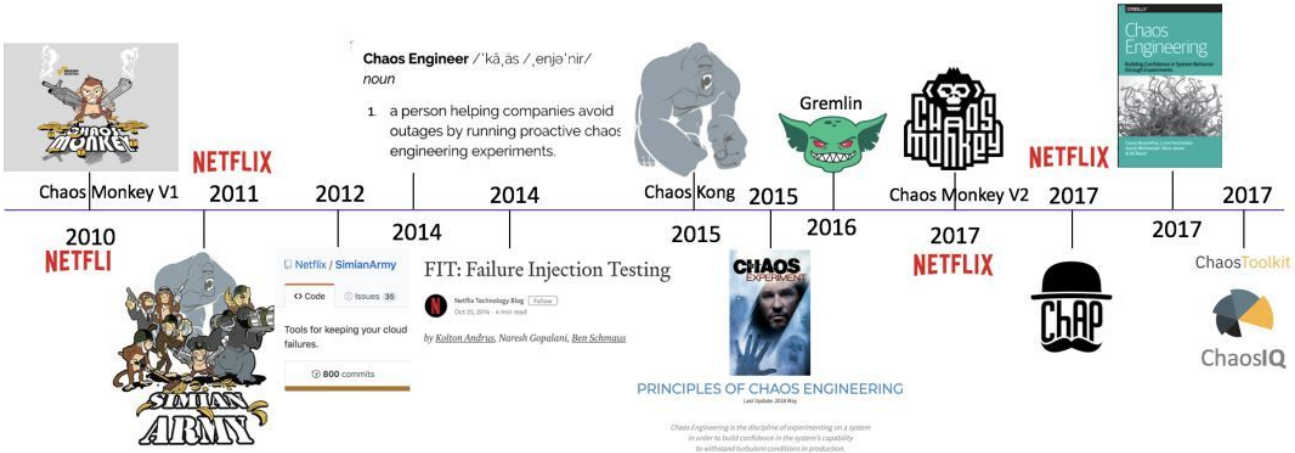
故障随时都有可能发生


github.com/pingcap/chaos-mesh/pull/93
Ooops!!!
500
Looks like something went wrong!
We track these errors automatically, but if the problem persists feel free to contact us. In the meantime, try refreshing.

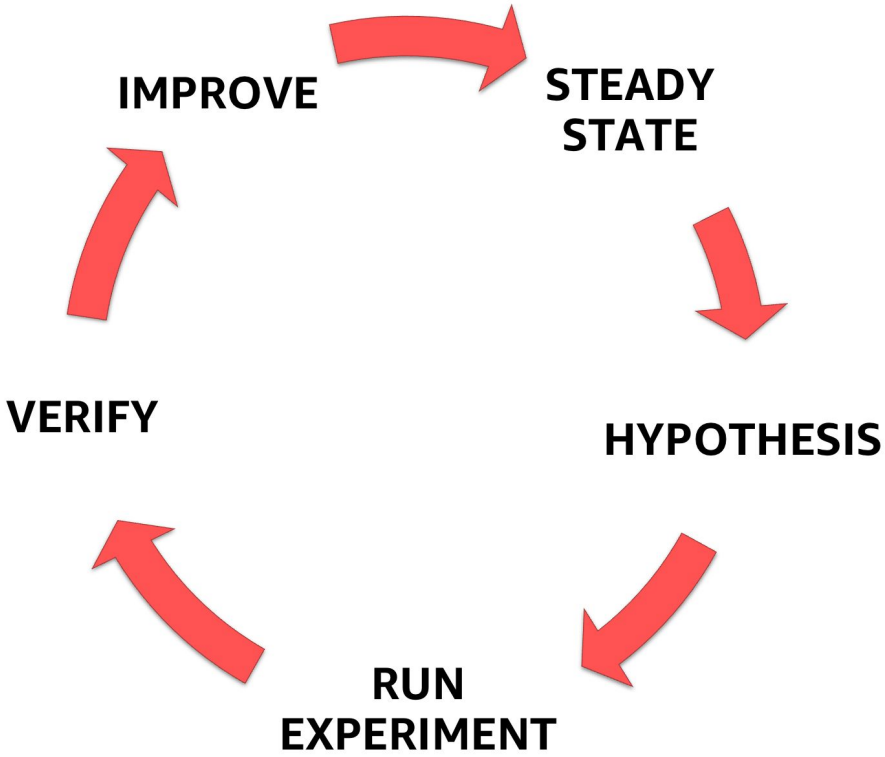
混沌工程定义

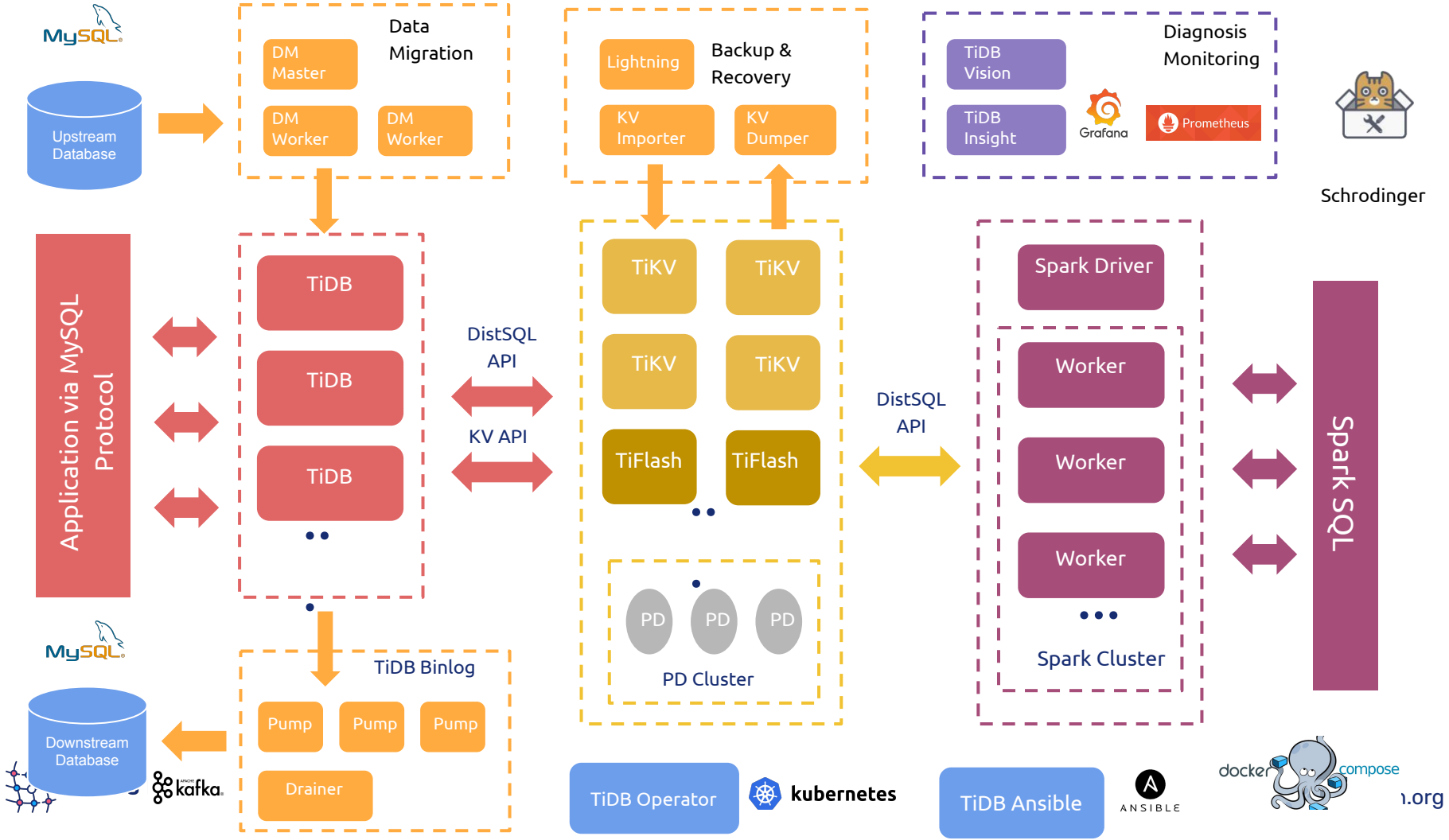
混沌工程是一门新兴的技术学科，他的初衷是通过实验性的方法，让人们建立对于复杂分布式系统在生产中抵御突发事件能力的信心。

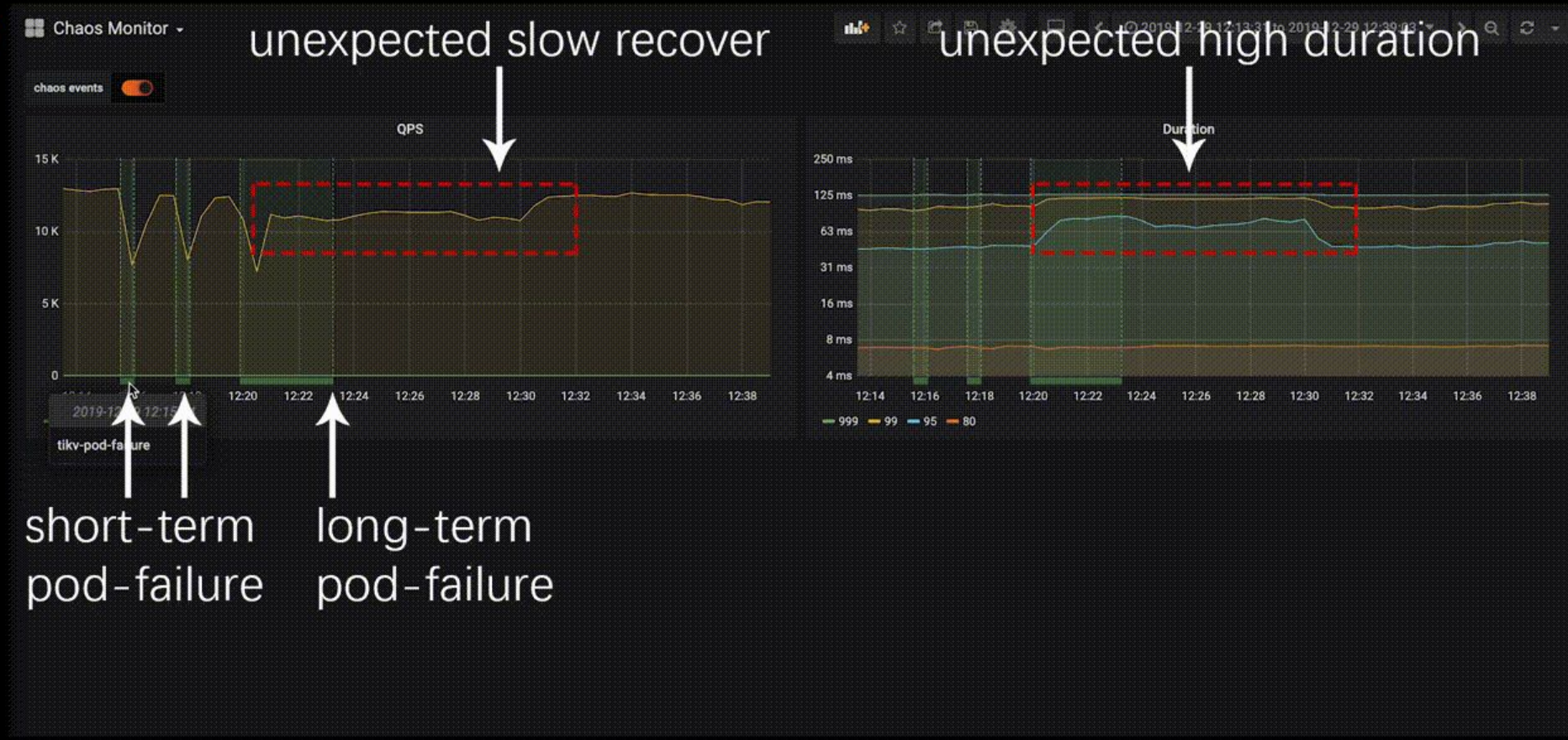
混沌工程的历史


Chaos Monkey V1
NETFLIX
2011
2012
2014
2010
NETFLI
2014
Chaos Engineer /ˈkā,äs /ˌenjəˈnir/
noun
1. a person helping companies avoid outages by running proactive chaos engineering experiments.
Chaos Kong
2015
2015
Gremlin
2016
Chaos Monkey V2
2017
NETFLIX
2017
NETFLIX
Chaos Engineering
2017
ChaosToolkit
ChaosIQ
Netflix / SimianArmy
Code
Issues 35
Tools for keeping your cloud failures.
800 commits
FIT: Failure Injection Testing
by Kolton Andrus, Naresh Gopalani, Ben Schmaus
CHAOS EXPERIMENT
PRINCIPLES OF CHAOS ENGINEERING
ChAP

混沌工程步骤

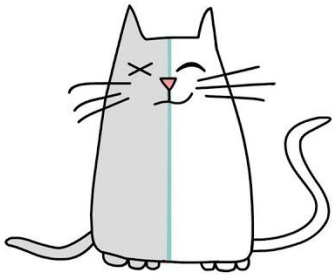



MySQL
Upstream Database
Data Migration
DM Master
DM Worker
DM Worker
Lightning
Backup & Recovery
KV Importer
KV Dumper
TiDB Vision
TiDB Insight
Diagnosis Monitoring
Grafana
Prometheus
Schrodinger
Application via MySQL Protocol
TiDB
TiDB
TiDB
DistSQL API
KV API
TiKV
TiKV
TiKV
TiKV
TiFlash
TiFlash
PD
PD
PD
PD Cluster
DistSQL API
Spark Driver
Worker
Worker
Worker
Spark Cluster
Spark SQL
MySQL
Downstream Database
kafka
TiDB Binlog
Pump
Pump
Pump
Drainer
TiDB Operator
kubernetes
TiDB Ansible
ANSIBLE
docker compose

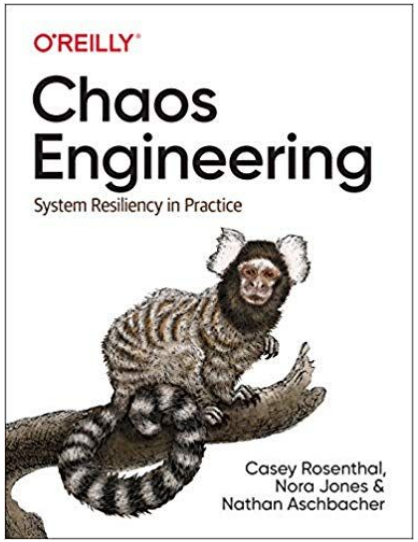


Here we have 2 short-time and a long-time pod-failures of TiKV pod

混沌工程在 PingCAP 的实践

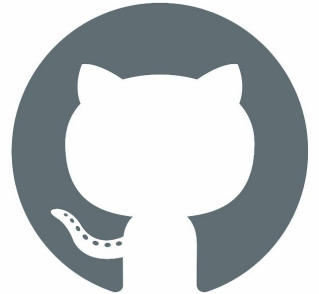


Schrodinger's Cat



Chaos Mesh®

Community



3,400+

Stars

86+

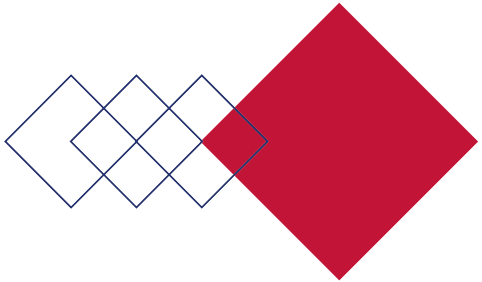
Contributors
and growing

Sandbox

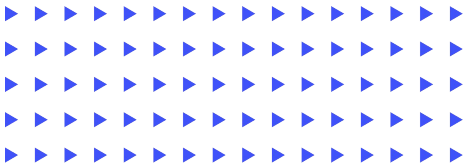
Cloud Native
Computing Foundation

PingCAP

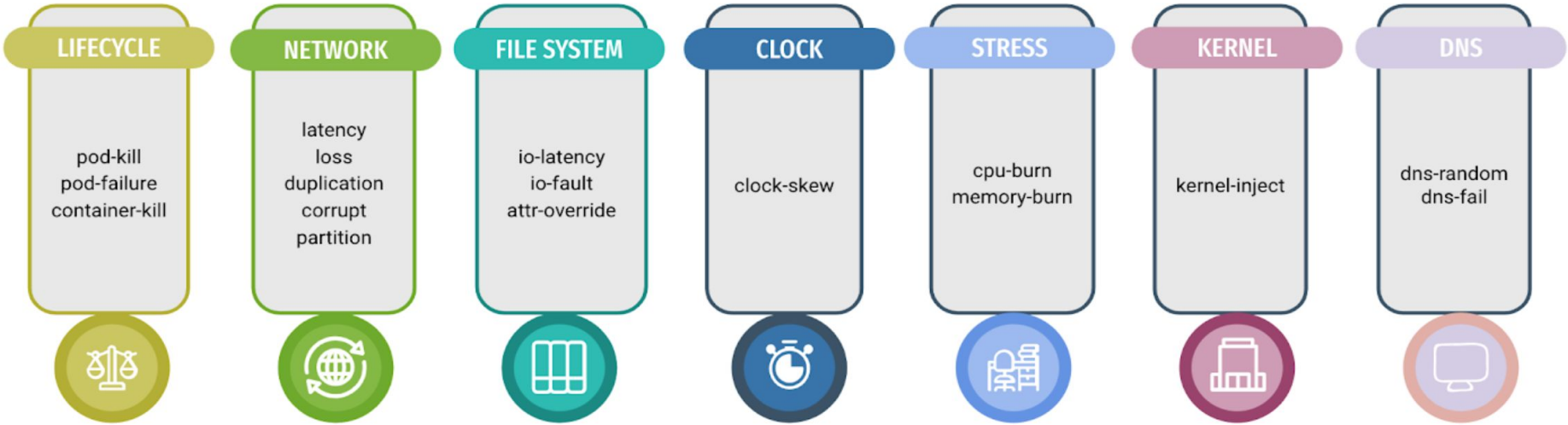
Chaos Mesh®



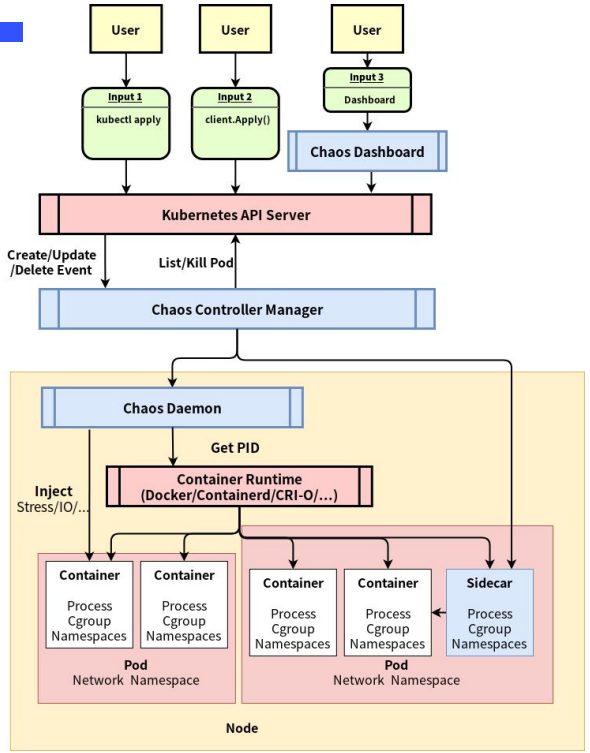
Part II - Chaos Mesh实现原理



Chaos Mesh 功能

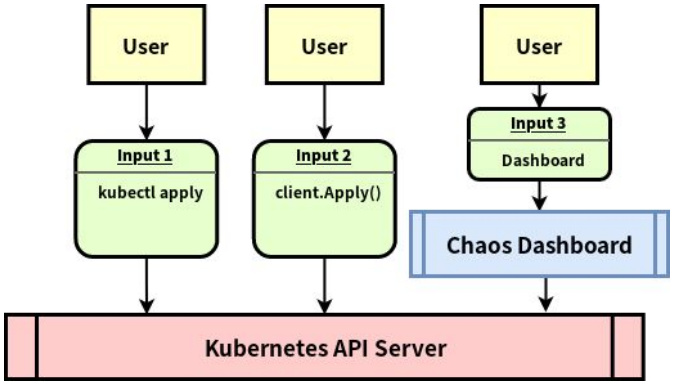


Chaos Mesh 整体架构



- 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

用户输入、观测



- 使用 kubectl 工具提交
- 使用 Kubernetes Client
- 使用 Chaos Dashboard

使用 kubectl 工具提交

```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: NetworkChaos
metadata:
  name: network
  namespace: chaos-testing
spec:
  action: partition
  mode: one
  selector:
    labelSelectors:
      "app.kubernetes.io/component": "tikv"
  direction: to
  target:
    selector:
      labelSelectors:
        "app.kubernetes.io/component": "tikv"
    mode: one
  duration: "10s"
  scheduler:
    cron: "@every 15s"
```

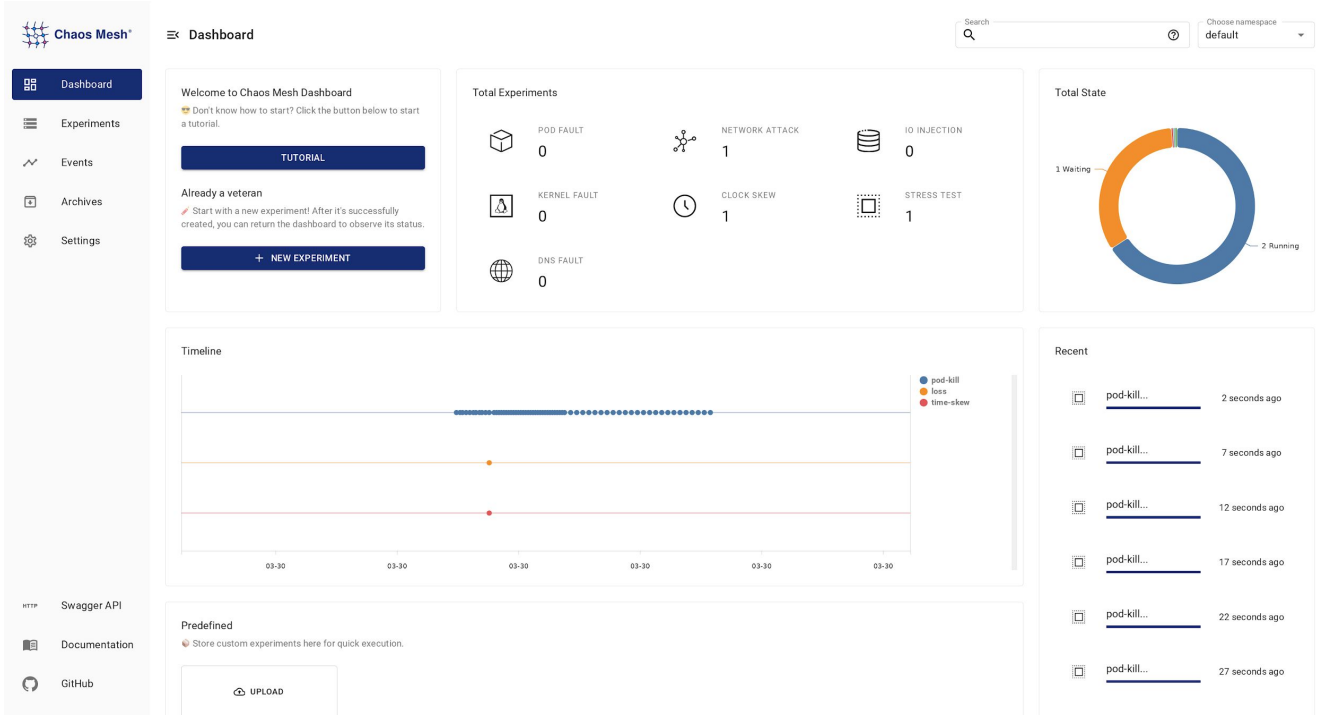
- kubectl apply -f ./network.yaml
- kubectl describe NetworkChaos network

使用 Kubernetes API/Client

```
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)
```

- 方便集成入测试流程
- 可编程地动态控制注入流程

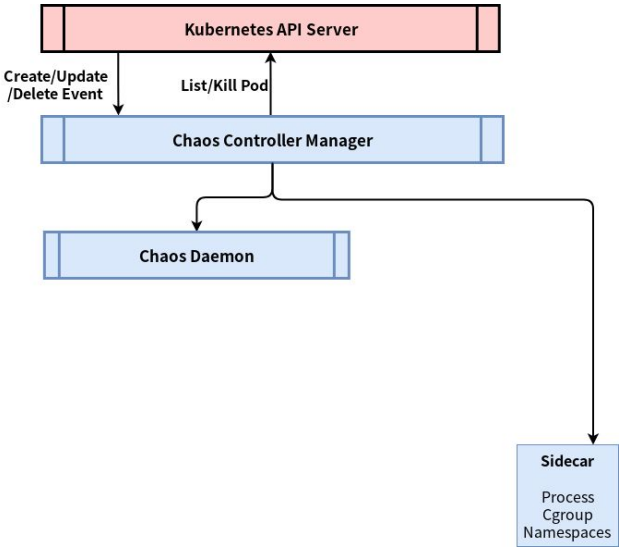
Chaos Dashboard



Chaos Dashboard

- 友好的用户界面
- 接入 RBAC 的权限管控
- 方便管理和观察已有的错误

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

Reconcile

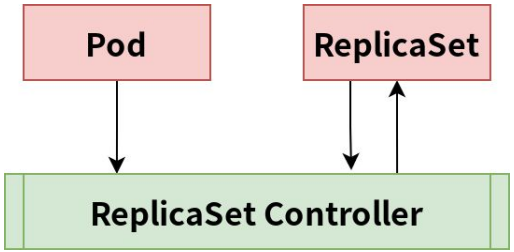
Resource		Physical Implementation
ReplicaSet		Pod
Pod		Container
ConfigMap		key/value in etcd
...		...
NetworkChaos		iptables rules / tc qdisc / ...

Reconcile 以 ReplicaSet 为例

```
Name:                hello-kubernetes-74c4d446d7
Namespace:           default
Controlled By:  Deployment/hello-kubernetes
Replicas:            0 current / 3 desired
Pods Status:         0 Running / 0 Waiting / 0 Succeeded / 0
Failed
Pod Template:
  Labels:  app=hello-kubernetes
           pod-template-hash=74c4d446d7
  Containers:
   hello-kubernetes:
           Image:    paulbouwer/hello-kubernetes:1.8
           Port:     8080/TCP
           Host Port:  0/TCP
           Limits:
           cpu:      50m
           Environment:  <none>
           Mounts:   <none>
  Volumes:           <none>
Events:              <none>
```

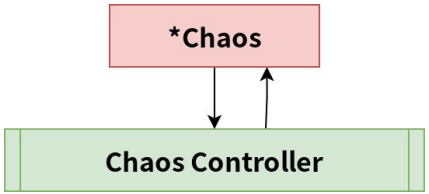
```
Name:                hello-kubernetes-74c4d446d7
Namespace:           default
Controlled By:  Deployment/hello-kubernetes
Replicas:            3 current / 3 desired
Pods Status:         3 Running / 0 Waiting / 0 Succeeded / 0
Failed
Pod Template:
  Labels:  app=hello-kubernetes
           pod-template-hash=74c4d446d7
  Containers:
   hello-kubernetes:
           Image:    paulbouwer/hello-kubernetes:1.8
           Port:     8080/TCP
           Host Port:  0/TCP
           Limits:
           cpu:      50m
           Environment:  <none>
           Mounts:   <none>
  Volumes:           <none>
Events:              <none>
```

Reconcile 以 ReplicaSet 为例



- Pod 和 ReplicaSet 的变化都会触发一次 Reconcile
- 更新 Pod 状态
- 比较 current replicas 和 desired replicas
- 如有必要，创建新的 Pod
- 更新 ReplicaSet 的状态

Reconcile 以 *Chaos 为例




```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: NetworkChaos
metadata:
  name: network
  namespace: chaos-testing
spec:
  action: partition
  mode: one
  selector:
    labelSelectors:
      "app.kubernetes.io/component": "tikv"
  direction: to
  target:
    selector:
      labelSelectors:
        "app.kubernetes.io/component": "tikv"
    mode: one
  duration: "10s"
  scheduler:
    cron: "@every 15s"
```

- 是不是已经删除了？
- 是不是注入过了？
- 需不需要 cronly 多次运行？
- 该如何注入 / 恢复？

Reconcile 以 *Chaos 为例

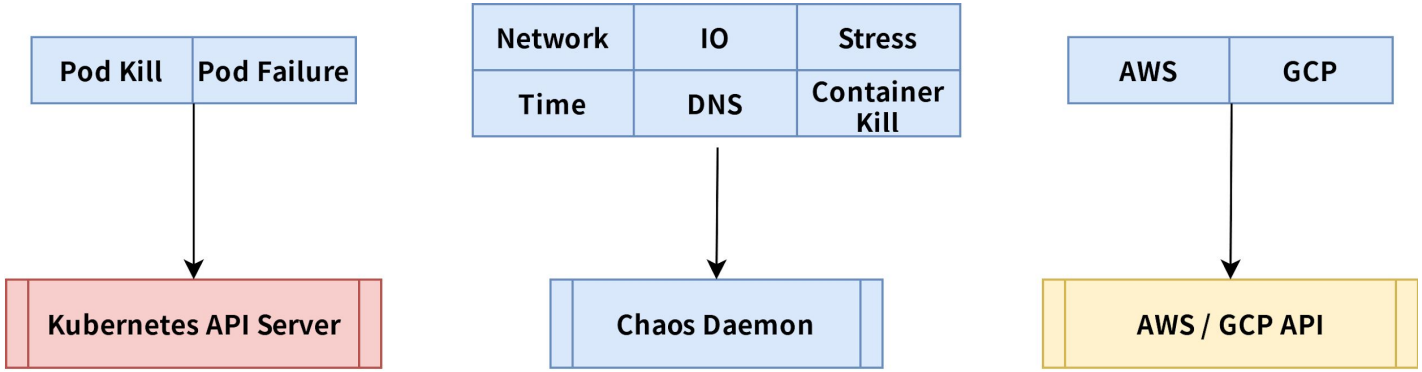
now: Wed Mar 31 02:39:38 PM CST 2021

```
duration: "10s"
scheduler:
  cron: "@every 15s"
```

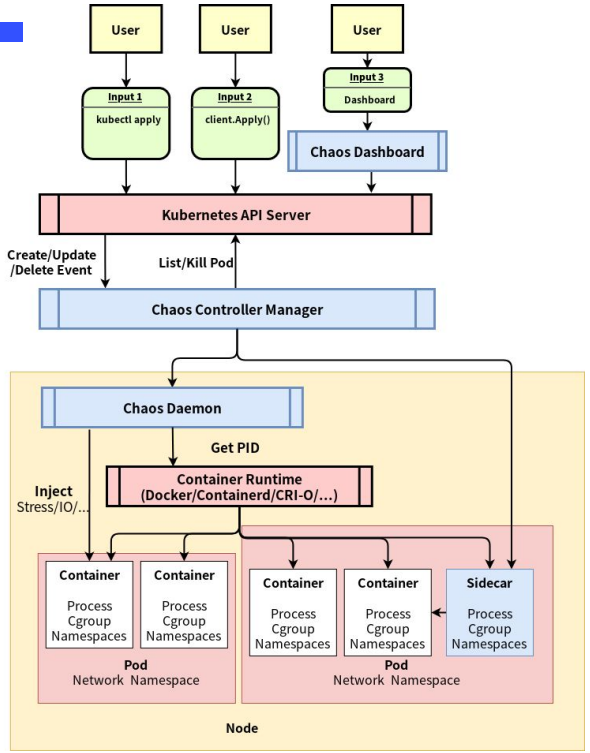
```
status:
  nextStart: "Wed Mar 31 02:39:37 PM CST 2021"
  nextRecover: "Wed Mar 31 02:50:37 PM CST 2021"
```

- 是不是已经删除了？
- 是不是注入过了？
- 需不需要 cronly 多次运行？

*Chaos 如何注入？

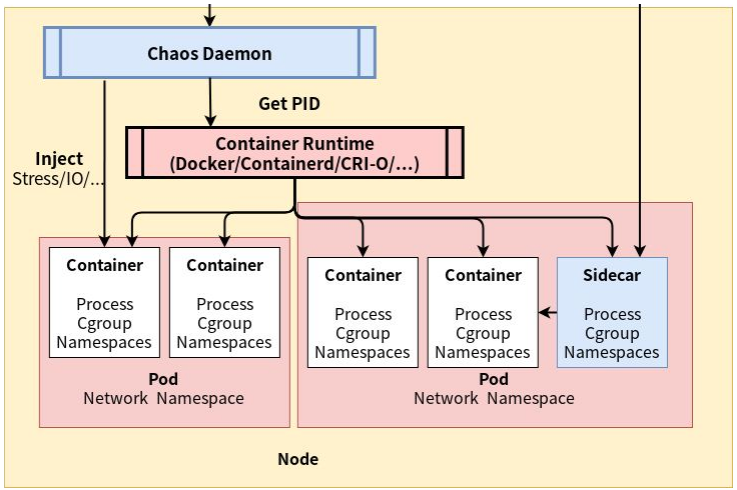

Pod Kill
Pod Failure
Kubernetes API Server
Network
IO
Stress
Time
DNS
Container Kill
Chaos Daemon
AWS
GCP
AWS / GCP API

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

Chaos Daemon 注入原理



- Container 的实体：
  - 进程
  - Namespace　控制可见性
  - Cgroup　限制资源

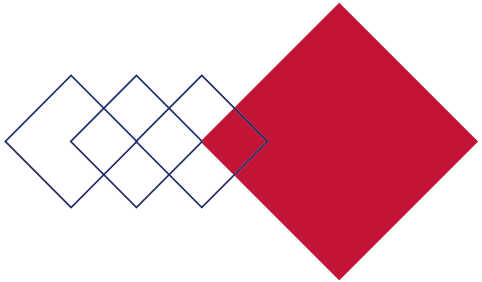
  } 隔离

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

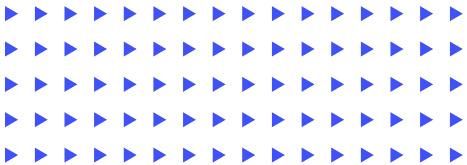
Cgroup 影响资源分配

- CPU 限制 CPU 资源分配
- Memory 限制 内存 资源分配
- Pid 限制 Pid 数量资源的分配
- ...

Chaos Mesh®



侵入 namespace



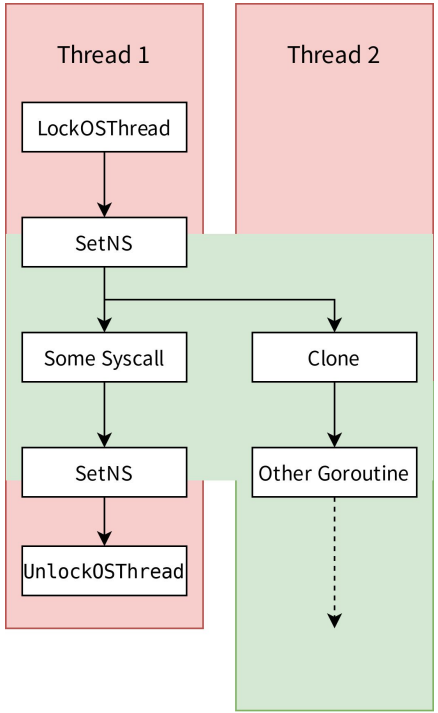
Namespace 影响查询过程

- mnt namespace　　影响 path resolution
- pid namespace　　影响 process lookup
- net namespace　　影响 network device lookup
- ...

Linux namespace and Go Don’t Mix

```
1: [pid  3361] openat(AT_FDCWD, "/proc/17526/ns/net", O_RDONLY) = 61
2: [pid  3361] getpid()                 = 3357
3: [pid  3361] gettid()                 = 3361
4: [pid  3361] openat(AT_FDCWD, "/proc/3357/task/3361/ns/net", O_RDONLY) = 62
5: [pid  3361] setns(61, CLONE_NEWNET)     = 0
<...>
6: [pid  3361] socket(AF_NETLINK, SOCK_RAW, NETLINK_ROUTE) = 63
7: [pid  3361] bind(63, {sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) = 0
8: [pid  3361] sendto(63, "\x20\x00\...", 32, 0, {sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) = 32
9: [pid  3361] getsockname(63, {sa_family=AF_NETLINK, pid=3357, groups=00000000}, [12]) = 0
10: [pid  3361] futex(0xc820504110, FUTEX_WAKE, 1 <unfinished ...>
11: [pid  3361] <... futex resumed> )       = 1
12: [pid  3361] futex(0xd82930, FUTEX_WAKE, 1) = 1
13: [pid  3361] futex(0xc820060110, FUTEX_WAIT, 0, NULL <unfinished ...>
14: [pid  3361] <... futex resumed> )       = 0
15: [pid  3361] recvfrom(63,  <unfinished ...>
16: [pid  3361] <... recvfrom resumed> "\x4c\x00\...", 4096, 0, {sa_family=AF_NETLINK, pid=0, groups=00000000}, [12]) = 236
<...>
17: [pid  3361] clone( <unfinished ...>
18: [pid  3361] <... clone resumed> child_stack=0x7f19efffee70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PAREN
T_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7f19effff9d0, tls=0x7f19effff700, child_tidptr=0x7f19effff9d0) = 3365
<...>
19: [pid  3361] setns(62, CLONE_NEWNET <unfinished ...>
20: [pid  3361] <... setns resumed> )       = 0
<...>
21: [pid  3365] sendto(65, "\x2c\x00\...", 44, 0, {sa_family=AF_NETLINK, pid=0, groups=00000000}, 12) = 44
22: [pid  3365] getsockname(65, {sa_family=AF_NETLINK, pid=3357, groups=00000000}, [12]) = 0
23: [pid  3365] recvfrom(65, "\x40\x00\...", 4096, 0, {sa_family=AF_NETLINK, pid=0, groups=00000000}, [12]) = 64
24: [pid  3365] close(65)                   = 0
25: [pid  3365] write(2, "Cannot find weave bridge: Link not found\n", 41
```

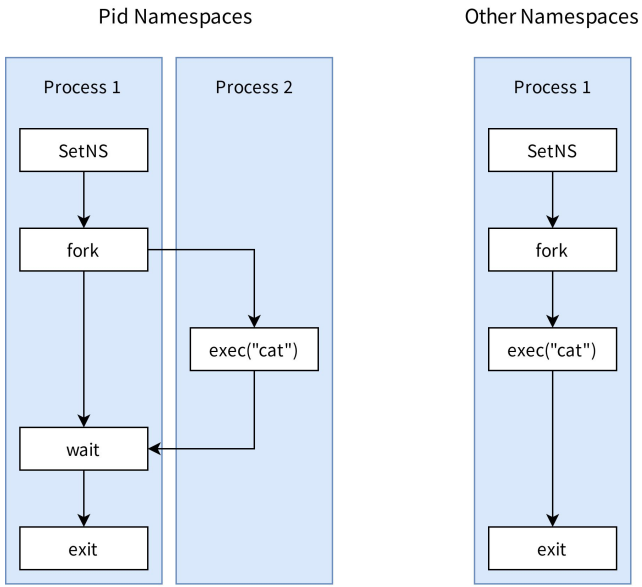
Linux namespace and Go Don’t Mix



Namespace 泄漏

会导致其他线程的 Namespace 难以追踪

nsenter



nsenter --pid /proc/xxx/ns/pid cat

nsenter 的问题

1. 信号处理较为随意，难以跟踪子进程

2. 难以应对 mnt namespace 的情形：

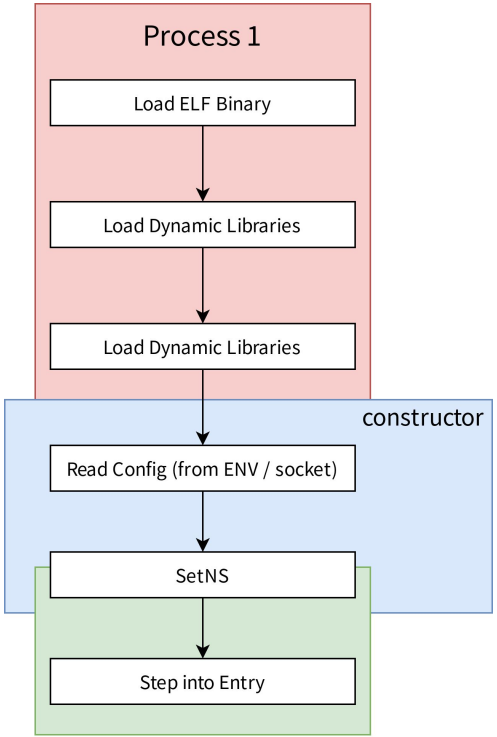
   nsenter --mnt /proc/xxx/ns/mnt cat

   nsenter: failed to execute cat: No such file or directory

   这一情形在 Distroless 容器逐渐流行的今天更加普遍了

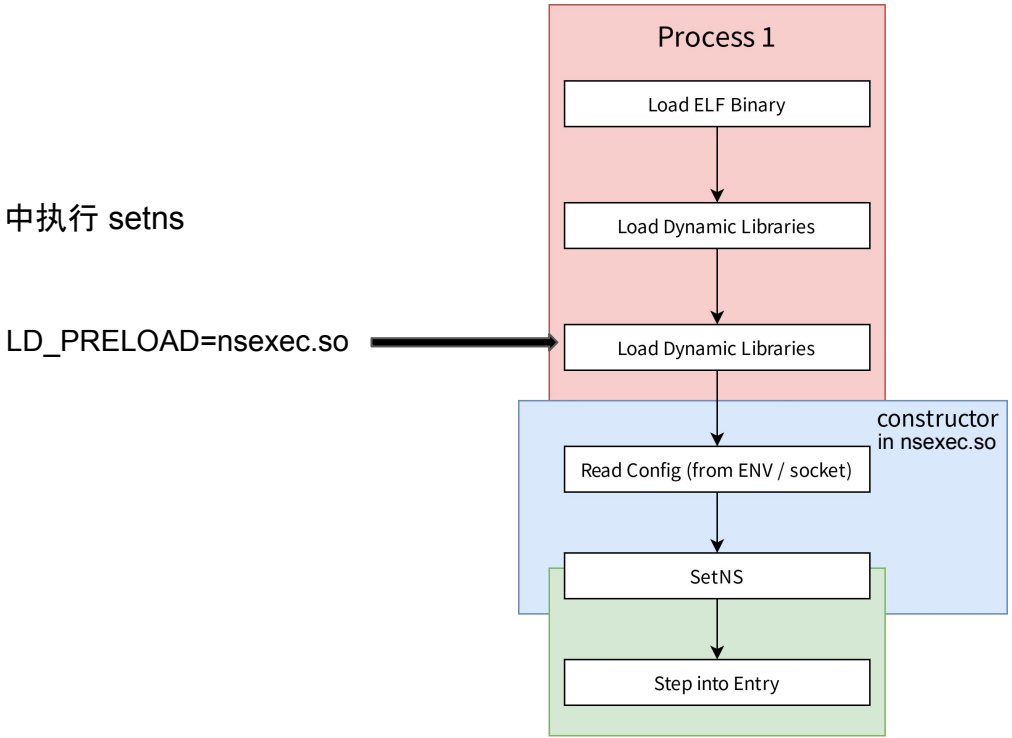
runc 方案

```
#cgo CFLAGS: -Wall
extern void nsexec();
void __attribute__((constructor)) init(void) {
        setns();
}
```



chaos-mesh/nsexec 方案

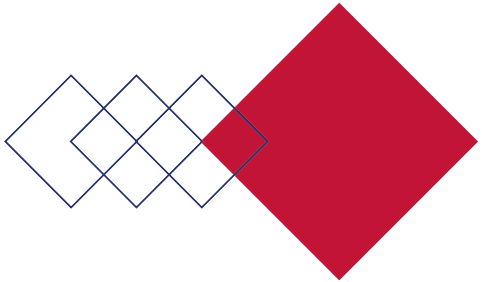
1. 设置 LD_PRELOAD
2. 在链接库的 constructor 中执行 setns



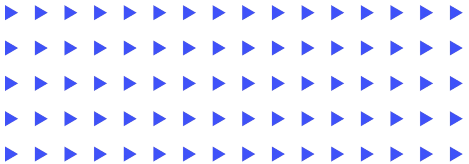
侵入 namespace

- 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 ...

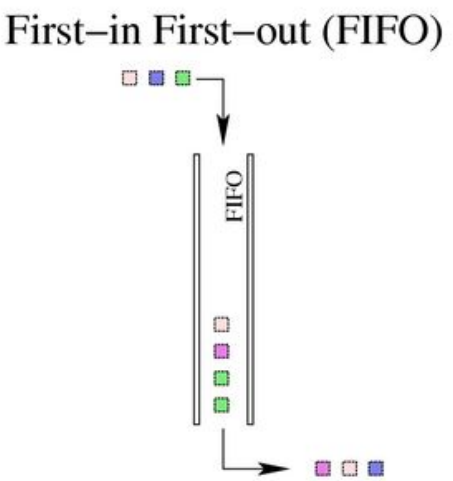
Chaos Mesh®



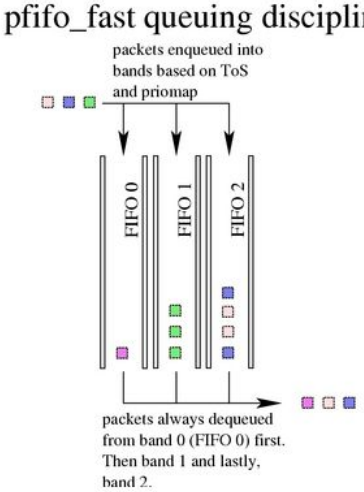
从流程中注入



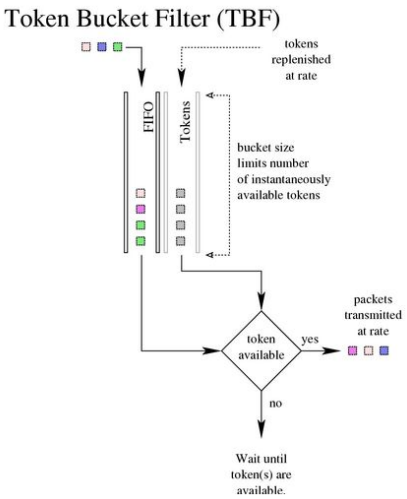
以（入方向）限流为例：tc qdisc



FIFO



pfifo_fast / prio



tbf

以时间偏移为例

- `clock_gettime` syscall (228 on x86_64)
- `clock_gettime` vDSO call
- Language/Runtime specific function call ⭐
  - Rust: `Instant::now()`
  - Go: `time.Now()`
  - C: `clock_gettime` in glibc
  - ...

language/runtime 专门的函数调用

- C: `clock_gettime` in glibc → `clock_gettime` vDSO call
- Rust: `Instant::now()` → `clock_gettime` in glibc → `clock_gettime` vDSO call
- Go: `time.Now()` → `clock_gettime` vDSO call
- ....

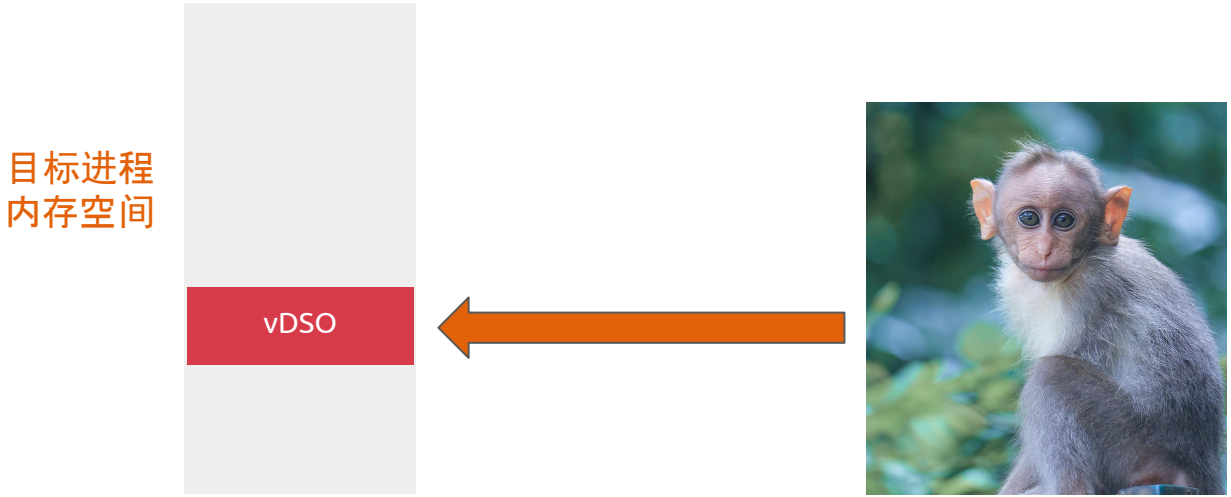
`clock_gettime` vDSO call 覆盖了大部分情况！

什么是 vDSO？

- 由 kernel 提供在用户态运行的函数的机制
  - clock_gettime
  - getcpu
  - gettimeofday
  - time
- 一段由 kernel 在程序启动时分配的内存区域
- 这部分内存的内容和动态链接库的 ELF 格式一致

如何完成注入？

1. `PTRACE_ATTACH` 容器中的每一个进程
2. `PTRACE_POKEDATA` 修改 vDSO 中时间相关函数的实现
3. `PTRACE_DETACH`



thix

浙江 杭州

扫一扫上面的二维码图案，加我微信

Join us !!!

Chaos Mesh 群组