Chaos Validation Made Easy: Plug & Play with Resilience Probes

Neelanjan Manna

Software Engineer, OSS Maintainer

harness   Litmus

Sayan Mondal

Sr. Software Engineer, OSS Maintainer

harness   Litmus

What Causes Downtime

Application Failures

- Excessive Logging to debug
- Too many retries
- Service Timeout

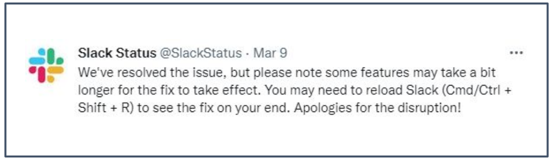
Infrastructure Failures

- Device failures
- Network failures
- Region not available

Operational Failures

- Capacity issues
- Incident management
- Monitoring dashboards not available

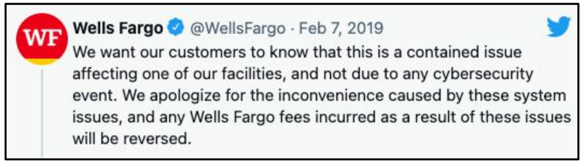
Reputational Impact


Slack Status @SlackStatus · Mar 9

We've resolved the issue, but please note some features may take a bit longer for the fix to take effect. You may need to reload Slack (Cmd/Ctrl + Shift + R) to see the fix on your end. Apologies for the disruption!

Slack's Outages

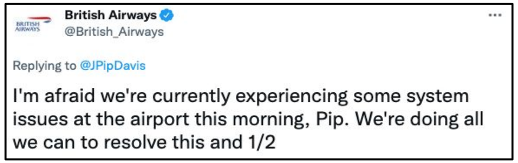
Financial Impact


Wells Fargo @WellsFargo · Feb 7, 2019

We want our customers to know that this is a contained issue affecting one of our facilities, and not due to any cybersecurity event. We apologize for the inconvenience caused by these system issues, and any Wells Fargo fees incurred as a result of these issues will be reversed.

Est. >$55M in losses to WF

Poor User Experience


British Airways @British_Airways

Replying to @JPipDavis

I'm afraid we're currently experiencing some system issues at the airport this morning, Pip. We're doing all we can to resolve this and 1/2

75,000+ passengers travel plans impacted

The problem with existing solutions

Failures impacting resiliency is inevitable

- Not proactively managed
- Downtimes maybe expensive

Failure Scenarios are Difficult to Implement

- Isn't implemented in a safe/controlled environment
- Isn't collaborative
- Not scalable

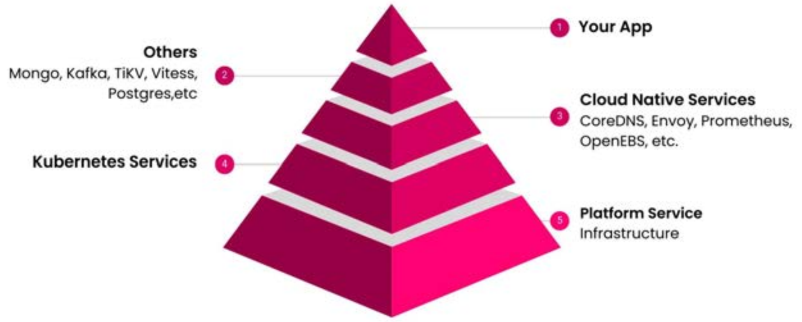
Failure Testing isn’t automated

- Believed to be just for Ops
- Difficult to manage chaos in CI/CD
- No monitoring of impact

The Cloud-Native problem

Proliferation of applications into micro services leads to a RELIABILITY challenge

In cloud native, your code depends on hundreds of other microservices and runs on many platforms. The potential of being subjected to a dependent component failure is huge.



Legacy DevOps

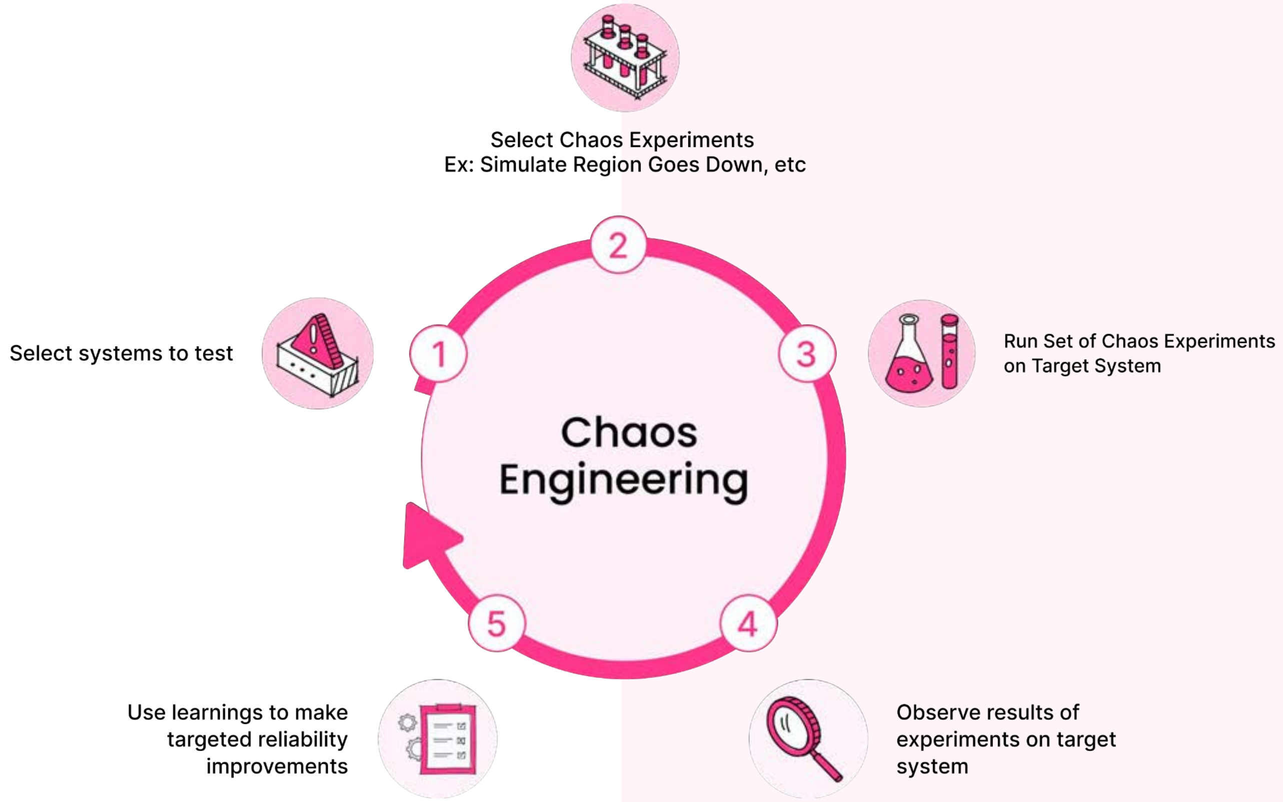
01 Build one application → Every Quarter → 02 Ship it. → 03 Run it.

Cloud native DevOps

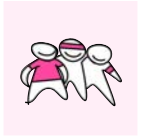
01 Build **10x** micro services → **Week** Every ~~Quarter~~ → 02 Ship them 10x faster. → 03 Run in 100x different environments

Too many fault scenarios. Significant increase in service down potential because of a failure of a dependent service

What is Chaos Engineering


Select Chaos Experiments
Ex: Simulate Region Goes Down, etc
2
Select systems to test
1
3
Run Set of Chaos Experiments on Target System
Chaos Engineering
5
4
Use learnings to make targeted reliability improvements
Observe results of experiments on target system

A Better Solution: Harness Chaos Engineering



Chaos engineering is *collaborative*

Collaborative chaos experiments in a centralized control plane

SREs + Developers

Experiments are in Git just like code

Robust *Experiments*

Public and private chaos hubs with ready to use experiments

Optimize initial investment

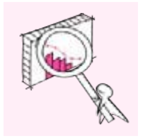
Reduce the inertia for starting chaos

Integrate into CI/CD systems

Rollout automated and controlled chaos experiments across prod/non-prod environments

Find weaknesses during build/test phase

Verifying at dev stage saves money



Enables *observability* for Chaos

Chaos metrics used to assess impact and manage SLOs/Errors

Measure the impact of inducing chaos

Build confidence by starting small

Harness Chaos Engineering

For Developers

Improve Resilience at CD with Chaos

Integrate with CI systems

Public and Private Chaos Hubs

Chaos Workflows

Chaos Center

Team Collaboration

Templatized Experiments

For SREs

Continuous Verification of SLOs

Chaos-assisted Observability

Drastically Improve Recovery Time

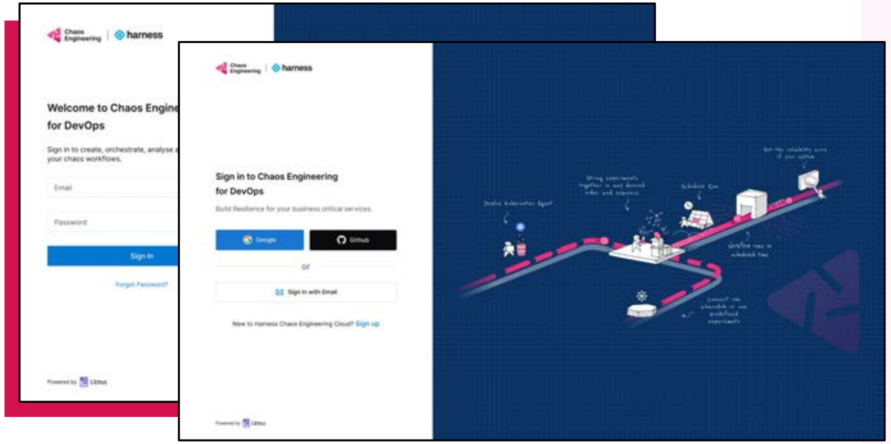
Reduce the Time to Triage Failures

Significantly Reduce Service Outages

Getting Started

01.

Get Started with SaaS or On-Premise



Welcome to Chaos Engine
for DevOps

Sign in to create, orchestrate, analyse a
your chaos workflows.

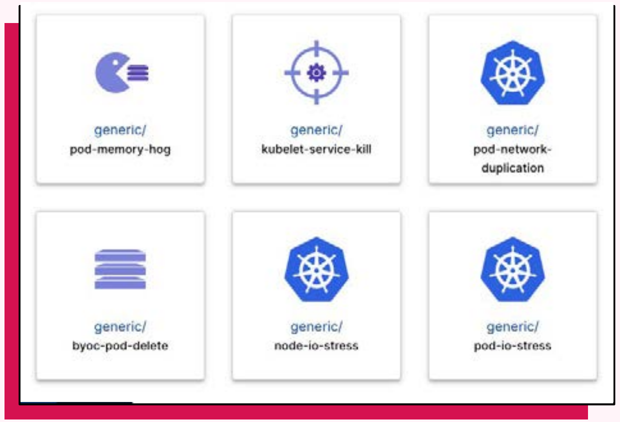
Sign in to Chaos Engineering
for DevOps

Build Resilience for your business critical services.

New to Harness Chaos Engineering Cloud? Sign up

02.

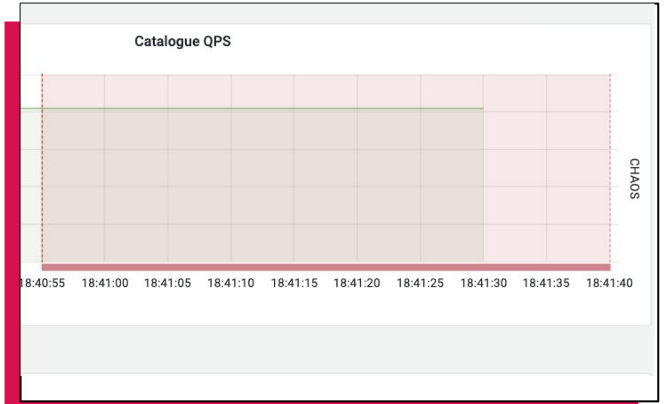
Pick an experiment, control your blast radius



Getting Started

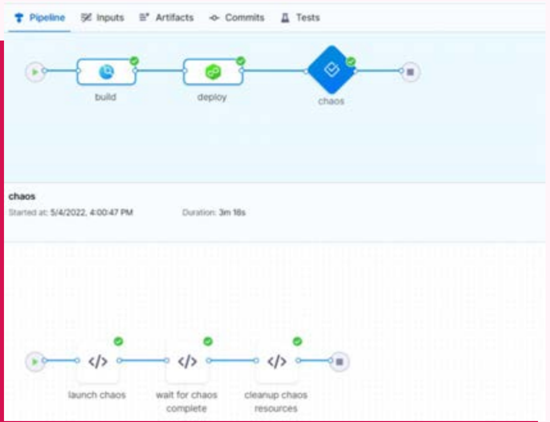
03.

Observe Impact



04.

Automate with CI/CD tooling



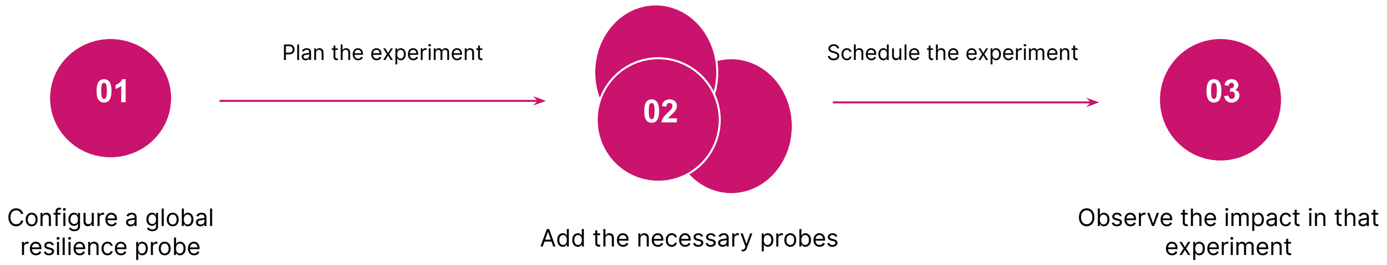
chaos
Started at: 5/4/2022, 4:00:47 PM
Duration: 3m 18s

What are Resilience Probes

Resilience Probes are reusable pluggable checks that could be used with any chaos experiment

Adhering to the 'Write once, Use anywhere' paradigm, this approach promotes the reuse of the same/new probe instead of creating a brand new one each time a chaos experiment is executed/edited.

How to use these probes?


01
Configure a global resilience probe
Plan the experiment
02
Add the necessary probes
Schedule the experiment
03
Observe the impact in that experiment

Types of probes

- HTTP Probe
- Command Probe
- Kubernetes Probe
- Prometheus Probe
- Datadog Probe
- Dynatrace Probe
- SLO Probe

- HTTP Probe
- Command Probe
- Datadog Probe
- Dynatrace Probe

Use cases

Query health/downstream URIs

Execute any user-desired health-check function

Perform CRUD operations against native & custom Kubernetes resources

Execute promql queries and match prometheus metrics for specific criteria

Let users validate the error budget for a given SLO

MODES

- SOT
- EOT
- OnChaos
- Continuous
- Edge

Hands on Demo

Thank You

Ask away any question!

Follow us on

/s_ayanide /NeelanjanManna

/s-ayanide /neelanjan00

harness.io