Apexon

WHITE PAPER

CHAOS ENGINEERING

APPROACHES, BEST PRACTICES & CASE STUDIES

apexon.com

“Chaos Engineering is the discipline of experimenting on a distributed system in order to induce artificial failures to build confidence in the system’s capability to withstand turbulent conditions in production.”

O’Reilly.com

Eight Fallacies of Modern-Day Distributed Computing

- The network is reliable
- Bandwidth is infinite
- Topology doesn’t change
- Transport cost is zero
- Latency is zero
- The network is secure
- There is one administrator
- The network is homogeneous

THE COST OF SYSTEM DOWNTIME

When IT infrastructure, networks, or applications unexpectedly fail or crash, it can have a significant impact on the business.

The actual cost varies greatly by business or organization, but just a few years ago, Gartner estimated the damage at ranges from a low of $140,000 to a high of $540,000 per hour. The impact can be seen in revenue loss and operational costs as well as customer dissatisfaction, lost productivity, poor brand image, and even derailed IT careers.

No matter how you measure it, IT downtime is costly. It’s also largely unavoidable due to the increasing complexity and interdependence of today’s distributed IT systems. The combination of cloud computing, microservices architectures, and bare-metal infrastructure create a lot of moving parts and potential points of failure, making those systems anything but predictable.

Distributed systems contain a lot of moving parts. Environmental behavior is beyond your control. The moment you launch a new software service, you are at the mercy of the environment it runs in, which is full of unknowns. Unpredictable events are bound to happen. Cascading failures often lie dormant for a long time, waiting for the trigger.

Build Testing vs. Chaos Engineering

Build Testing

A specific approach to testing known conditions.

Assertion: given specific conditions, a system will emit a specific output.

Tests are typically binary; determine whether a property is true or false.

Chaos Engineering

A practice for generating new information.

More exploratory in nature with unknown outcomes.

Tests effects of various conditions; generates more subjective information.

CHAOS ENGINEERING

Chaos Engineering is a new approach to software development and testing designed to eliminate some of that unpredictability by putting that complexity and interdependence to the test.

The idea is to perform controlled experiments in a distributed environment that help you build confidence in the system's ability to tolerate the inevitable failures. In other words, break your system on purpose to find out where the weaknesses are. That way, you can fix them before they break unexpectedly and hurt the business and your users.

As a result, you will better understand how your IT systems really behave when they fail. You can exercise contingency plans at scale to ensure those plans work as designed. Chaos Engineering also provides the ability to revert systems back to their original states without impacting users. It also saves a lot of time and money that would be spent responding to systems outages.

Any system is as strong as its weakest point. Chaos Engineering practices help identify weak points of the complex system pro-actively.

The purpose is not to cause problems or chaos. It is to reveal them before they cause disruption so you can ensure higher availability.

The more chaos experiments (tests) you do, the more knowledge you generate on system resilience. This helps minimalize downtime, thereby reducing SLA breaches and improving revenue outcomes.

At Apexon, we believe that a key element in Continuous Testing is monitoring and testing throughout the development, deployment and release cycles. Chaos Engineering integrated in DevOps value chains plays a vital role in achieving this.

TOOLING

There are a number of different tools available to support your Chaos Engineering efforts.

Which ones you use depends on the size of your environment and how automated you want the process to be. Below are just a few to be aware of.

CHAOS MONKEY

Tests IT infrastructure resilience.

LITMUS

Provides tools to orchestrate chaos on Kubernetes to help SREs find bugs and vulnerabilities in both staging and production.

CHAOS TOOLKIT

Enables experimentation at different levels: infrastructure, platform and application.

GREMLIN

Is a “failure-as-a-service” platform built to make the Internet more reliable. It turns failure into resilience by offering engineers a fully hosted solution to safely experiment on complex systems, in order to identify weaknesses before they impact customers and cause revenue loss.

TOXIPROXY

Simulates network conditions to support deterministic tampering with connections, with support for randomized chaos and customization. It can determine if an application has a single point.

And there are many more.

Apexon is currently powering the biggest Chaos Engineering community in India to help organizations build fault-tolerant and robust cloud-native applications to accelerate their digital initiatives.

Our approach is built on the principles of Continuous Testing and software test automation and is segmented by the Infrastructure, Network, and Application layers.

- INFRASTRUCTURE LAYER
- NETWORK LAYER
- APPLICATION LAYER

APPROACHES & BEST PRACTICES

Chaos Testing @ Infrastructure Layer

PURPOSE:

Anticipate production failures and mitigate them by simulating failure of virtual instances, availability zones, regions, etc

Primarily done in production or production-like environments

CHAOS ENGINEERING TOOLS FROM NETFLIX:

Chaos Monkey

Litmus

ToxiProxy

Swabbie (Formerly Janitor Monkey)

Conformity Monkey (Now part of Spinnaker)

OTHER OPTIONS:

Chaos Lambda (lower scale)

ALSO, CONSIDER A COMMERCIAL TOOL LIKE GREMLIN:

Simian Army is deprecated, and tools are being made part of Spinnaker

Chaos Monkey does not support deployments that are managed by anything other than Spinnaker

No abort or roll back function available

Limited support/coordination

No UI

Chaos Testing @ Network Layer

PURPOSE:

Ensure App doesn't have single points of failure; simulate network and system conditions supporting deterministic tampering with connections, but with support for randomized chaos and customization

Simulate network degradation/ intermittent connectivity; how applications behave in these conditions early during development

IDEAL FOR:

Mobile apps with offline functionality

SPA web apps that work without network connectivity

TOXIPROXY:

Latency (with optional jitter)

Complete service unavailability

Reduced bandwidth

Timeouts

Slow-to-close connections

Piecemeal information, with more optional delays

Chaos Testing @ Application Layer

PURPOSE:

Instill Chaos Engineering principles early in the development stage; build for resilience and stability

Developers & SDETs primarily lead this activity in this stage but consult and involve business/product owners for expected results. Ops can also be consulted or informed

USE CASES:

Dev Environment/Local Machine

Observe component/service under test behavior in the absence of a dependent service in another docker container.

Tools: Docker, KubeMonkey

Lower-Level Environments

Introduce chaos at container level: killing, stopping, and removing running containers.

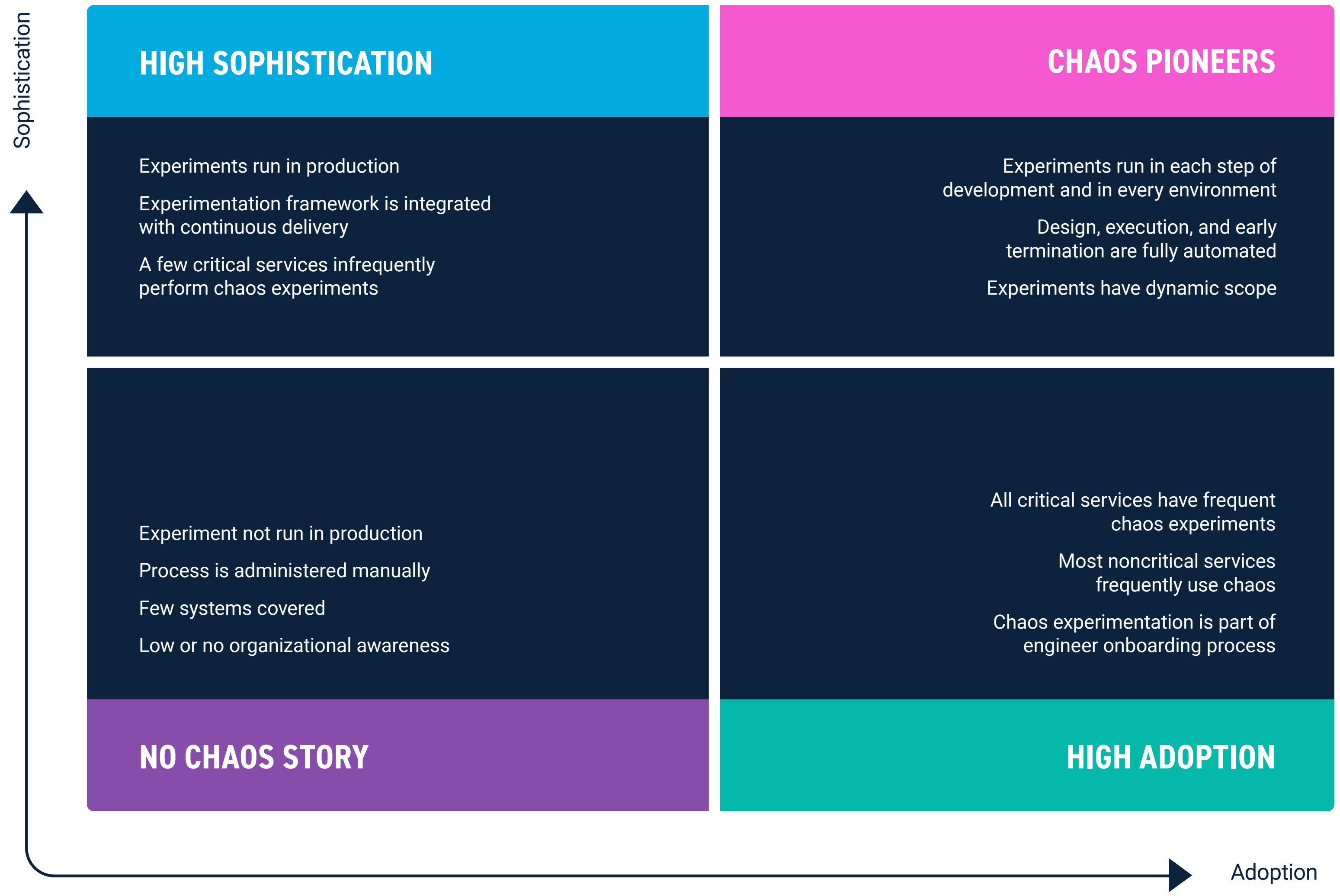
Tools: Pumba (similar to Chaos Monkey but works at container level)

Mimic service failures and latency between service calls

Tools: Service Mesh like Istio and Chaos Monkey for Spring Boot

MATURITY MODEL

The Maturity Model below provides a map for software delivery teams getting started with Chaos Engineering and evolving their use of it over time. It's a useful way to track your progress and compare yourself to other organizational adopters. Apexon uses this model when we work with clients to layout the most effective approach that will deliver the most productive results.



HIGH SOPHISTICATION

Experiments run in production

Experimentation framework is integrated with continuous delivery

A few critical services infrequently perform chaos experiments

CHAOS PIONEERS

Experiments run in each step of development and in every environment

Design, execution, and early termination are fully automated

Experiments have dynamic scope

NO CHAOS STORY

Experiment not run in production

Process is administered manually

Few systems covered

Low or no organizational awareness

HIGH ADOPTION

All critical services have frequent chaos experiments

Most noncritical services frequently use chaos

Chaos experimentation is part of engineer onboarding process

APPROACH & PROCESS

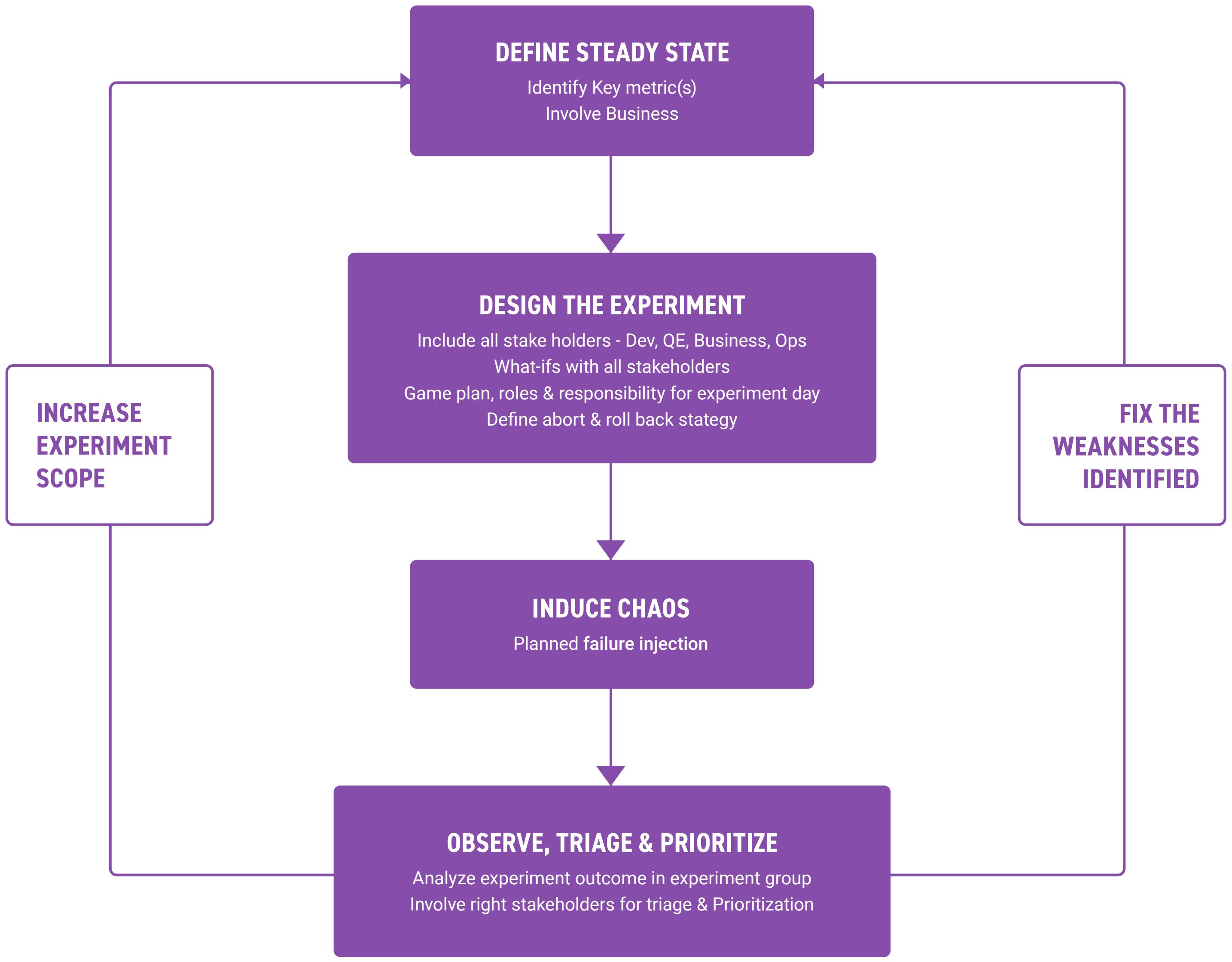
Apexon follows a disciplined process with several key steps that dictate how we design Chaos experiments. The degree to which we can adhere to these steps correlates directly with the confidence we can have in a distributed system at scale.

- Identify metrics and values to define steady state of system
- Hypothesize it will work well for control group and experimental group
- Introduce variables that reflect real world events like servers that crash, dependencies that fail, etc.
- Stimulate environment using introduced variables to disapprove set hypothesis
- Manage the blast radius by ensuring that the fallout from experiments are minimized and contained

LIFECYCLE MANAGEMENT

With those steps above as our roadmap, the workflow outlined below insures that critical Chaos experiment information is passed along at each stage and informing the next.

Experimental Lifecycle & Best Practices


DEFINE STEADY STATE
Identify Key metric(s)
Involve Business
DESIGN THE EXPERIMENT
Include all stake holders - Dev, QE, Business, Ops
What-ifs with all stakeholders
Game plan, roles & responsibility for experiment day
Define abort & roll back stategy
INDUCE CHAOS
Planned failure injection
OBSERVE, TRIAGE & PRIORITIZE
Analyze experiment outcome in experiment group
Involve right stakeholders for triage & Prioritization
INCREASE EXPERIMENT SCOPE
FIX THE WEAKNESSES IDENTIFIED

CASE STUDY ONE

LEADING TELECOMMUNICATIONS SERVICE PROVIDER

Apexon helped the customer in designing Microservices Platform based on popular container orchestration engine.

Being a telecommunications provider most critical aspect of their service is SLA.

EXPERIMENT	OUTCOMES
What if key components like ElasticSearch or Kafka or Redis are killed?	Platform components recovered within 2-4 mins. As they were stateful components, failure count wasn't stretched beyond the quorum.
What if multiple instances in different autoscaling groups are randomly shutdown?	AWS Autoscaling group replaced the killed instance with the new instance within 2-5 mins. Container orchestration platform started scheduling containers to this new instance.
What if there is a sudden resource exhaustion on underlying VMs?	We experimented CPU and Memory resource exhaustion and did notice performance degradation on those VMs. We also found that container orchestration platform stopped scheduling containers to those instances due to resource saturation.

CASE STUDY TWO

DATA INFORMATICS

Apexon helped this customer in designing and developing Pythom SDK.

SDK code was responsible for downloading or uploading Terabytes of data. It was critical for SDK to work even under inconsistent network conditions. Team proactively developed and verified the following experiments:

EXPERIMENTS		OUTCOMES
What if the latency increases exponentially, how will it affect SDK's upload/ download behavior?	◀ ▶	Increasing latency resulted in more time from uploading or downloading the blob.
What if there is vigorous fluctuation in network bandwidth?	◀ ▶	We experimented with decrease in bandwidth, which resulted in more time for uploading or downloading the blob.
What if there is timeout from the API backend, will SDK resume download/ upload from the stalled point?	◀ ▶	SDK continued upload or download from the stalled point for the blob whenever timeout was introduced.
What if the remote API backend crashes, will SDK retry and resume download / upload from the stalled point?	◀ ▶	SDK continued to retry with pre-defined attempts and resumed from the stalled point whenever API crash was stimulated.

Apexon

APEXON IS A PURE-PLAY DIGITAL ENGINEERING SERVICES FIRM FOCUSED ON HELPING COMPANIES ACCELERATE THEIR DIGITAL INITIATIVES FROM STRATEGY AND PLANNING THROUGH EXECUTION.

We leverage deep technical expertise, Agile methodologies and data-driven intelligence to modernize systems of engagement and simplify human/tech interaction.

We deliver custom solutions that meet customers' technology needs wherever they are in their digital lifecycle. Backed by Goldman Sachs and Everstone Capital, Apexon works with both large enterprises and emerging innovators — putting digital to work to enable new products and business models, engage with customers in new ways, and create sustainable competitive differentiation.

info@apexon.com www.apexon.com

FEELING SOCIAL?