Chaos Engineering & Resiliency Kickstart

Develop a passion for learning.

EMERGING TECHNOLOGY

What is Chaos Engineering?

Chaos engineering has ushered in the next generation of cloud-native applications.

Despite this, the answer to the preceding question can difficult for many individuals and organizations. Chaos Engineering is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production.

WHAT IS CHAOS ENGINEERING?

Chaos Engineering is the discipline of **experimenting** on a **distributed system** in order to **build confidence** in the system's **capability to withstand** turbulent conditions in production.

WHAT IS CHAOS ENGINEERING MORE SIMPLY?

Experiment to ensure that the impact of failure is mitigated

Failures

What went wrong?

What kind of thing failed

Impacts

What are the effects of the failure?

What mitigation mechanisms are in place?

You can only be as strong as your
weakest link

How can we try to think of everything that might fail?

HISTORY LESSON

Commercial solutions leveraging chaos engineering are beginning to appear. Many more are likely on the horizon. Creating chaos sounds messy, but people have been creating controlled chaos for years with Disaster Recovery and High Availability disciplines.

HISTORY LESSON

Chaos engineering

2004	Amazon—Jesse Robbins. Master of disaster
2010	Netflix—Greg Orzell. @chaosimia - First implementation of Chaos Monkey to enforce use of auto-scaled stateless services
2012	NetflixOSS open sources simian army
2016	Gremlin Inc founded
2017	Netflix chaos eng book. Chaos toolkit open source project
2018	Chaos concepts getting adopted widely, and this conference!

LOGISTICS

Class Hours:

- Start time is X:XXam
- End time is X:XXpm
- Class times may vary slightly for specific classes
- Breaks mid-morning and afternoon (15 minutes)

Lunch:

- Lunch is 11:45am to 1pm
- Yes, 1 hour and 15 minutes
- Extra time for email, phone calls, or simply a walk.

Telecommunication:

- Turn off or set electronic devices to vibrate
- Reading or attending to devices can be distracting to other students
- Try to delay until breaks or after class

Miscellaneous

- Courseware
- Bathroom
- Fire drills

DAY 1

Designing and Applying Your First Chaos Experiments

- What is Chaos Engineering and Why do I need it?
- Building Confidence in Complicated and Complex Systems
- Understanding and Working with the Emergent and Novel with Cynefin, Chaos and Microservices
- Working with Experiments and Chaos in Practice
- Why do Chaos in Production?
- Establishing Prerequisites to Chaos Engineering
- Architecting and Designing for Chaos
- Building a Hypothesis for a Chaos Engineering experiment
- Chaos and Testing
- Failure Injection Testing in Context
- Minimizing the Impact of Experiments - Understanding your Experiment's Blast Radius, Scope & Opt-Out
- Idempotency and Responsibility with Chaos Engineering Experiments
- Chaos and Logging
- Designing and Implementing Steady State monitoring for your System
- Defining a Chaos Engineering Experiment
- Defining Steady State Probes
- Defining Experimental Methods
- Defining Continuous State Probes
- Defining Close State Probes
- Working with Other People's Systems
- Running an Experiment Manually.
- Establish a Kubernetes system that can be subjected to Chaos Experiments
- Automating your first Experiment using the Chaos Toolkit

DAY 2

Operationalizing Chaos in the Enterprise

- How to focus on the Kubernetes Platform to build Chaos Engineering Experiments
- Brainstorming Platform-level Chaos Engineering Experiments
- Establishing Kubernetes Platform Steady State Probes
- Building Platform Experimental Methods and Continuous State Probes
- Establishing Platform Close State Probes
- Running Platform Experiments Manually and through Automation
- How to focus on the Pod resiliency to build Chaos Engineering Experiments
- Brainstorming Application-level Chaos Engineering Experiments
- Establishing Application Steady State Probes
- Building Application-specific Experimental Methods and Continuous State Probes
- Establishing Application Close State Probes
- Running Application Experiments Manually and through Automation
- Building Custom Experimental Steps
- Defining Cross-Level Hypotheses and Experiments
- Building and Executing Cross-Level Chaos Engineering Experiments
- How to establish a Chaos Engineering Practice
- Progressing towards greater Chaos Maturity

DAY 3

Building Reliable, Resilient and Observable Pipelines

- Learning, Resilience and Reliability
- Software Defined Delivery Goals
- One thing at once.
- Deliver with Confidence with Green/Blue and Canary
- Having a “Big Red Button”
- Defining Observability
- Why a system needs Observability
- Who needs Observability?
- The Three Pillars of Observability
- The Four Golden Signals of Monitoring
- Working with Spring Extension for Chaos
- How Monitoring and Observability Work Together
- The Extents of Observability
- Common Observability Measures and Practices
- Working with Chaos to programmatically circuit break
- How to Successfully Apply Coding for Failure to a Production System
- Applying Observability to Your Own System
- Identifying API interaction for resiliency under load
- Horizontal Pod Autoscaler
- Driving scale with custom metrics
- Effective Pre-Mortem Learning: Chaos Engineering

meet the instructor

George Niece

Cloud, DevOps, IoT Consultant with a Linux sysadmin background. Focused on cloud-native application modernization

Twitter
@georgeniece

LinkedIn
Linkedin.com/in/GeorgeNiece

mail
George.Niece@DigitalTransformationStrategies.net

Expertise

- Cloud
- AppDev
- IoT
- Automation
- CI/CD
- Microservices
- Agile

HISTORY LESSON

Netflix began moving out of on-premise datacenters into the cloud in 2008, where they began practicing some form of resiliency testing in production. Over time this become known as Chaos Engineering. Chaos Monkey started the ball rolling, gaining notoriety for turning off services in the production environment. Chaos Kong transferred those benefits from the small scale to the very large.

STARTING OFF RIGHT

Now we'll do a short GTKY

Get To Know You

Name

Title or Role

What you do?

Why are you here?

Something with chaos

WHY DO WE NEED CHAOS ENGINEERING?

Chaos Engineering is the foundation of a new application paradigm in resiliency and reliability. Rather than inject bugs into our applications, we test them in their natural state injecting faults.

WHY DOES THE BUSINESS NEED CHAOS ENGINEERING?

All of these companies suffered catastrophic downtime in that last year that cost millions of dollars in lost revenue and significant reputational damage

BUILDING CONFIDENCE IN COMPLICATED AND COMPLEX SYSTEMS

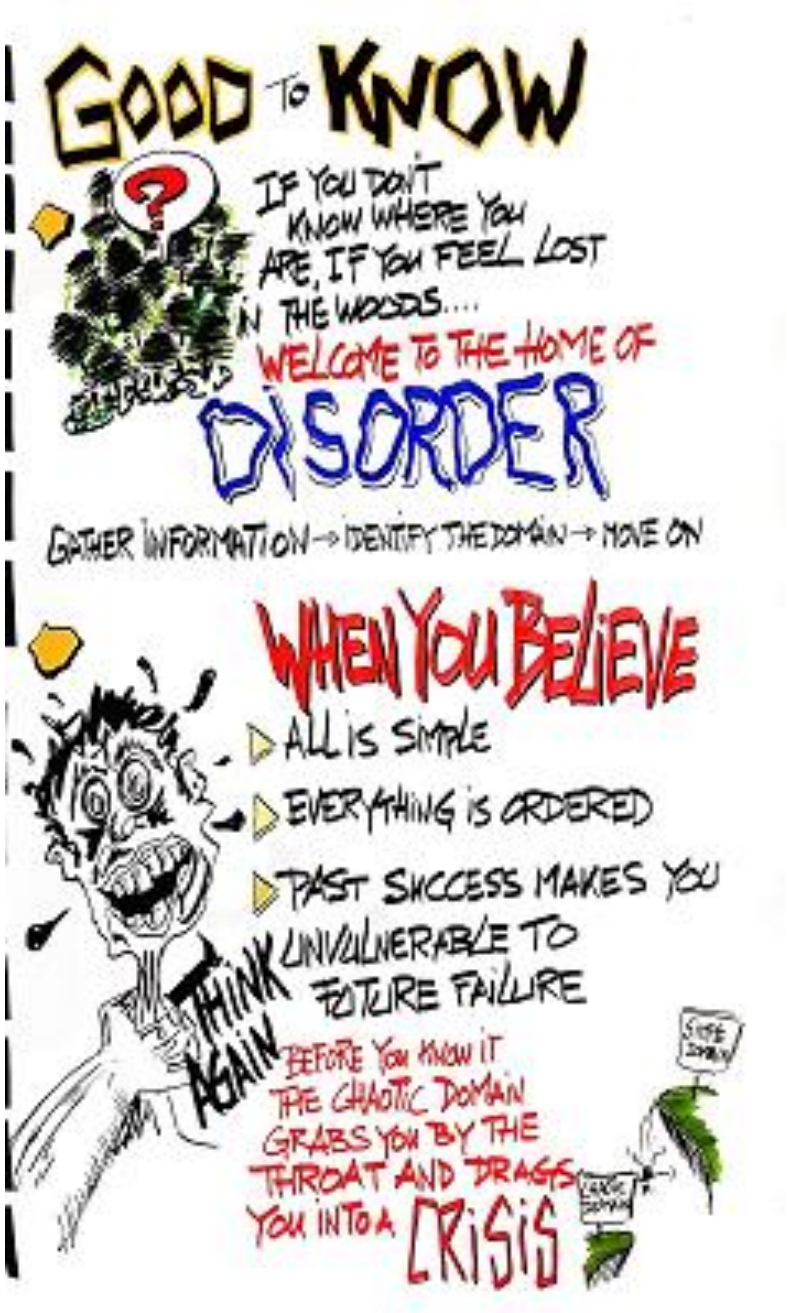
When you create a microservice-based application, you need to deal with complexity. Of course, a single microservice is simple to deal with, but dozens or hundreds of types and thousands of instances of microservices is a complex problem. It isn't just about building your microservice architecture—you also need high availability, addressability, resiliency, health, and diagnostics if you intend to have a stable and cohesive system.

Synoptic Illegibility

If you can't write down exactly what **really** happens, (can't write a synopsis) you can't automate an ad-hoc and messy system.

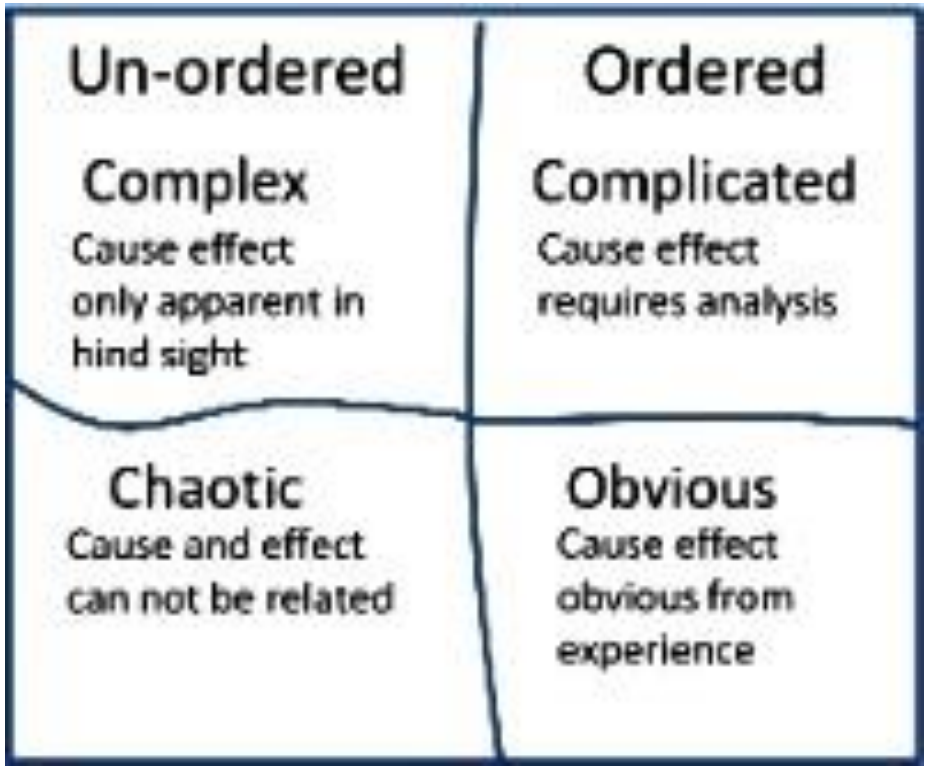
WHAT IS CYNEFIN?

kuh-NEV-in is the Welsh word for habitat. Developed in the 2000's by IBM as a way to put perspective on complex systems



UNDERSTANDING AND WORKING WITH THE EMERGENT AND NOVEL WITH CYNEFIN, CHAOS AND MICROSERVICES

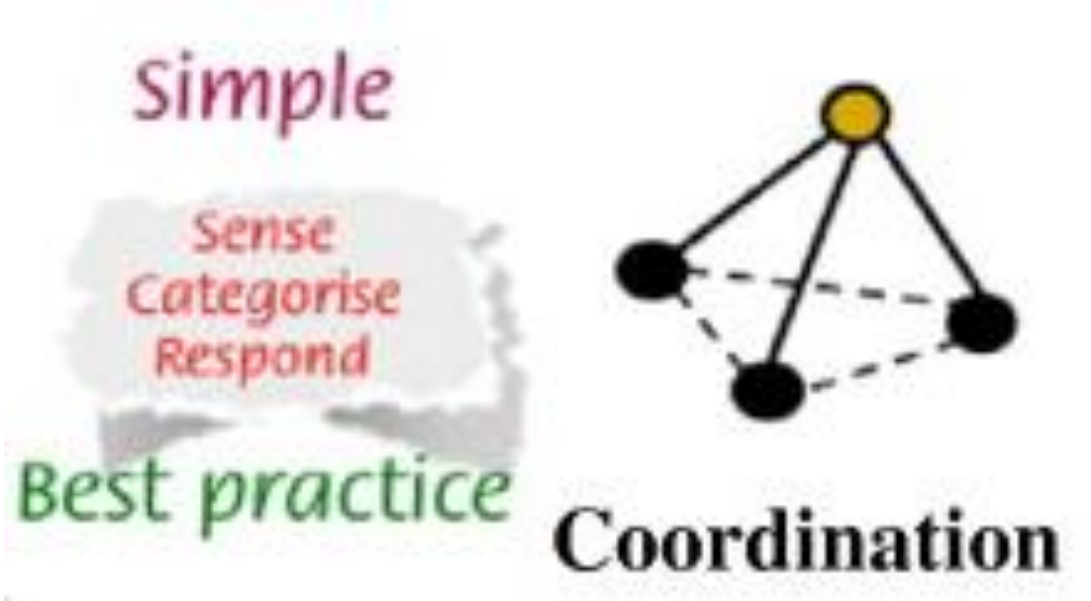
We need to acknowledge that systems are not always stable and the state of the system may change over time. A system may be stable one day and we believe we know how it works. However we may find that its performance is degrading overtime or it may simply break.



ORDERED AND UNORDERED CHAOS

In the ordered, we may have to involve an expert (say a mechanic in the case of a car) to analyze what is going on. In the case of the un-ordered, where a car has broken down and we have been thrown into chaos, we may need to involve a rescue service to recover the vehicle and take it to a garage in order for a mechanic to undertake the analysis.

DOMAINS OF CHAOS – SIMPLE OR OBVIOUS

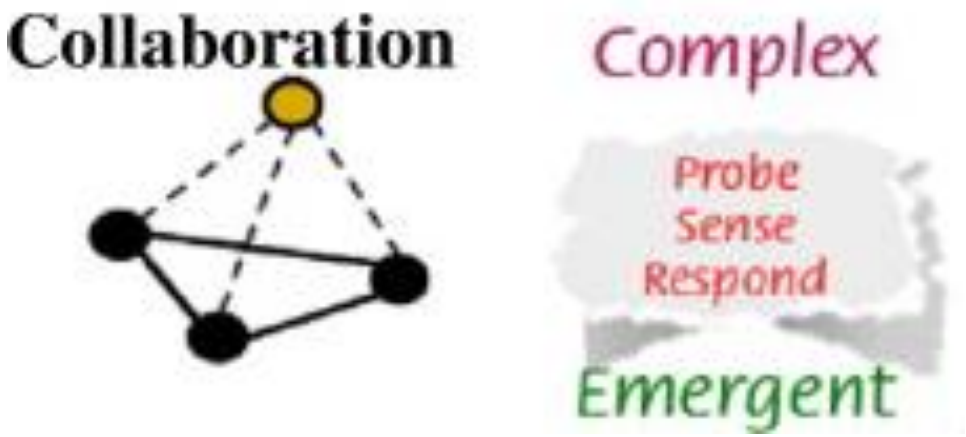


Obvious (known knowns) – here we know what we are doing and have seen it a thousand times before, so we sense, categorize and respond (S-C-R). Here we expect to see best practices employed.

DOMAINS OF CHAOS - COMPLICATED

Complicated (known unknowns) – we don't know what is going on but know that we can analyse what has happened and work it out, so we are sensing, analyzing and responding (S-A-R). This is the domain of good practice.

DOMAINS OF CHAOS – COMPLEX



Complex (unknown unknowns) - we cannot determine what will cause a particular outcome but we can run some experiments to see if they move us in the right direction so we probe, sense and respond (P-S-R). There is no right or wrong answer, so we may want to run a series of experiments or in fact run a number in parallel.

DOMAINS OF CHAOS - DISORDER

Disorder (not determined) – these are the items that we are yet to determine in which domain they belong.

WORKING WITH EXPERIMENTS AND CHAOS IN PRACTICE

Chaos Engineering is a disciplined approach to identifying failures before they become outages. By proactively testing how a system responds under stress, you can identify and fix failures before they become all hands meetings and newsworthy.

Chaos Engineering lets you compare what you think will happen to what actually happens in your systems. You literally experiment (break things on purpose) to learn how to build more resilient systems.

WHAT ARE CHAOS EXPERIMENTS?

- Experiments are documented tasks that reveal system vulnerability when performed, with the chaos engineering goal of teaching us something new.
- Usually one of two outcomes
  - Success – The system is confirmed not to be vulnerable to the experiment performed. Celebrate.
  - Success – The system is confirmed to be vulnerable to the experiment performed. Celebration may be appropriate here, as well, after all you just identified a vulnerability. After the celebration, fix the issue and run the experiment to validate.

WHAT ARE CHAOS PROBES?

- Points of failure. Centralized systems are easier to manage but have a single point of failure.
- Fault / Tolerance. Decentralized are resilient because of the redundancy of effort.
- Scalability. Centralized - limited scalability. Distributed - more scalable. Decentralized - infinite scalability.
- Development. Centralized is simple and technically easier to develop. Decentralized requires more considerations and more technically complex.
- Security. Centralized is less secure with a single point of attack. Decentralized, especially emerging technology like chaos engineering and AI/ML, become more secure as they scale.

WHY DO CHAOS IN PRODUCTION?

Systems behave differently depending on environment and traffic patterns. Since the behavior of utilization can change at any time, sampling real traffic is the only way to reliably capture the request path. To guarantee both authenticity of the way in which the system is exercised and relevance to the current deployed system, Chaos strongly prefers to experiment directly on production traffic.

POP QUIZ: CHAOS ENGINEERING

What is the biggest roadblock to chaos in most organizations?

2 MINUTE

POP QUIZ: CHAOS ENGINEERING

What is the biggest roadblock to chaos in most organizations?

CULTURE

- No time for experiments
- Teams already too busy fixing things
- Politics
- Deeply invested in product technical roadmap (micro-services) that chaos engineering tests show are not as resilient as predicted
- Could start broad and deep conversations

2 MINUTE

Lab 1- Setting up the Chaos Toolkit

Lab 1- Setting up the Chaos Toolkit

The Chaos Toolkit is a free, open source project that enables you to create and apply Chaos Experiments to discover, and eventually improve and address, weaknesses across your system's infrastructure, platform and application levels.

This tutorial takes you through the setup of the Chaos Toolkit's **chaos** command. It also teaches you how Chaos Toolkit Extensions, that select which layers and technologies can be targeted by your experiments, can be installed into your environment.

https://katacoda.com/chaostoolkit/courses/01-chaostoolkit-getting-started/chaostoolkit-install

Chaos Engineering

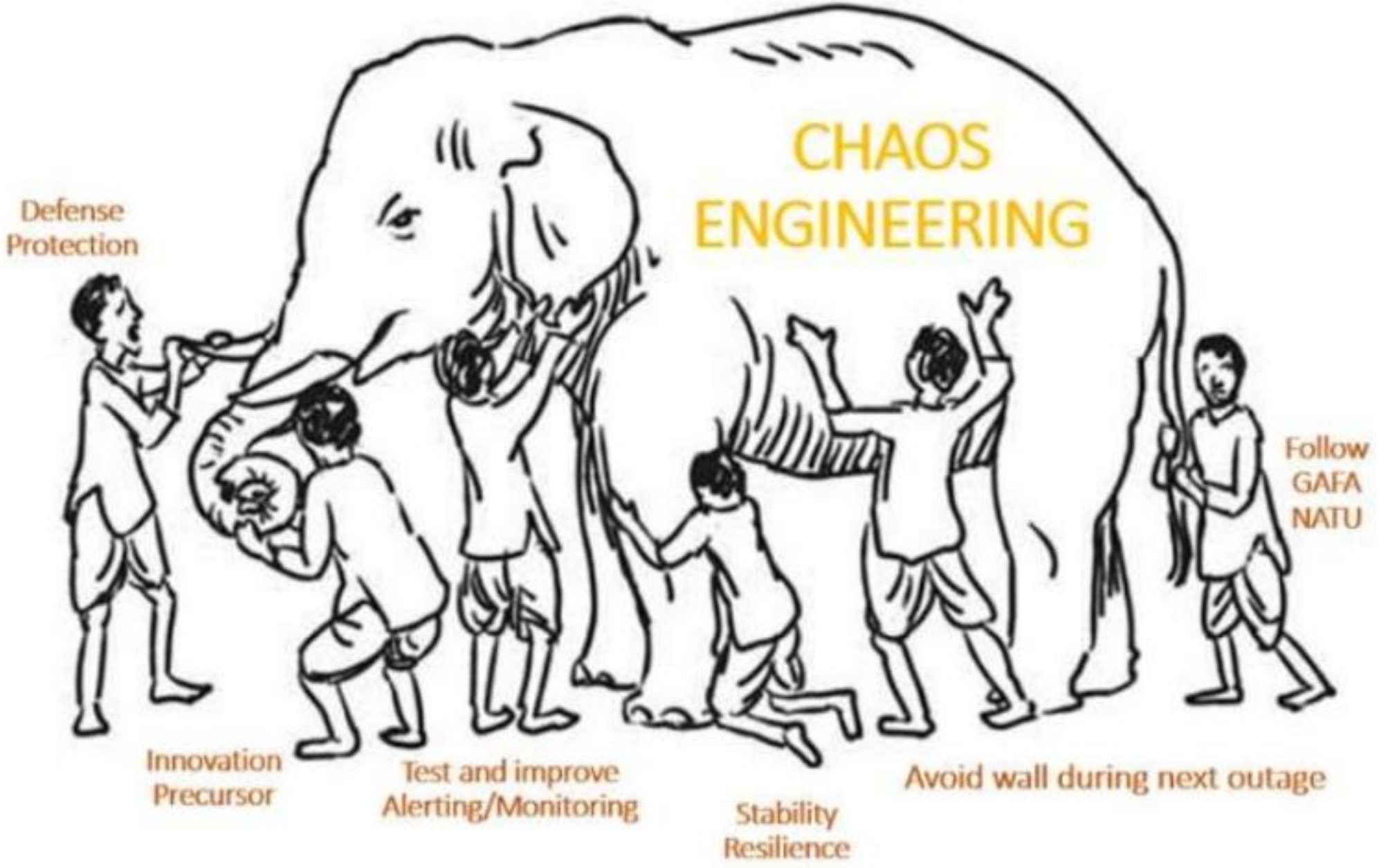
Concepts

ESTABLISHING PREREQUISITES TO CHAOS ENGINEERING

Imagine someone says in a meeting, “We will break everything in production, and it will be fun”.

STEADY STATE HYPOTHESIS

"a model that characterizes the steady state of the system based on expected values of the business metrics."



“Gee, Chaos Engineering sounds really cool, but our software and our organization are both completely different from Born Internet Companies like Netflix and so this stuff just wouldn’t apply to us.”

ARCHITECTING AND DESIGNING FOR CHAOS

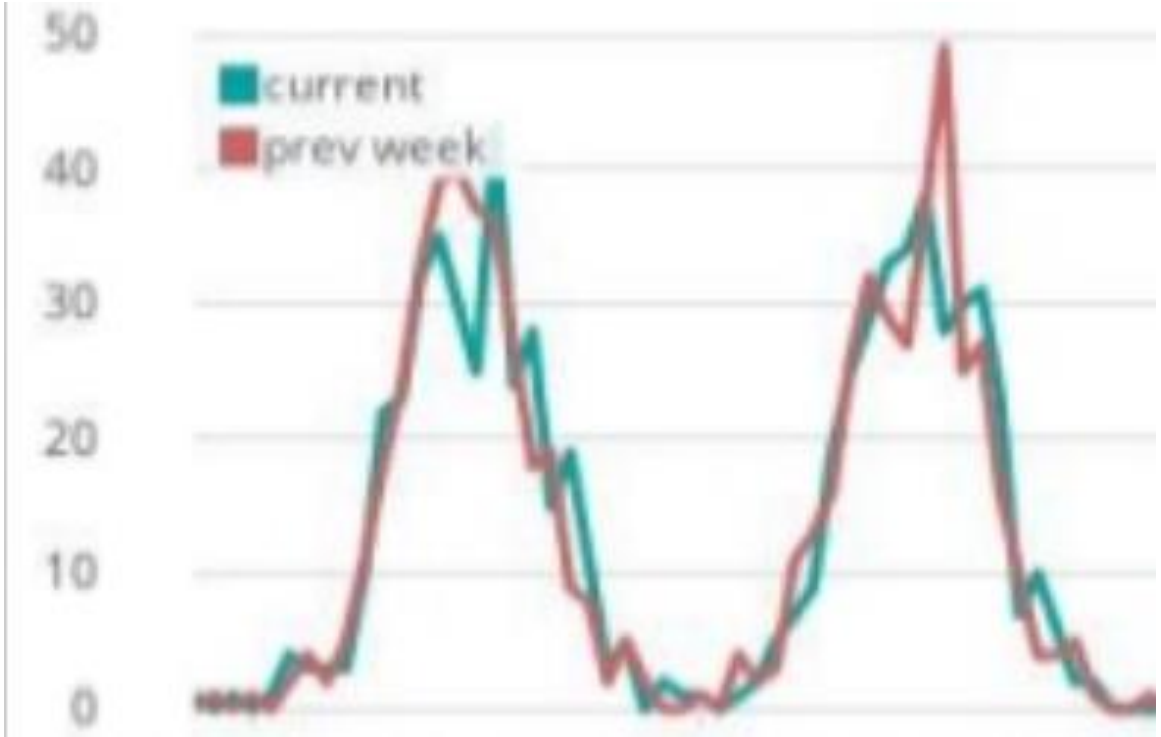
Opportunities for chaos experiments are boundless and may vary based on the architecture of your distributed system and your organization's core business value.

CHAOS ENGINEERING CONCEPT: PRINCIPALS

To specifically address the uncertainty of distributed systems at scale, Chaos Engineering can be thought of as the facilitation of experiments to uncover systemic weaknesses. These experiments typically follow four steps.

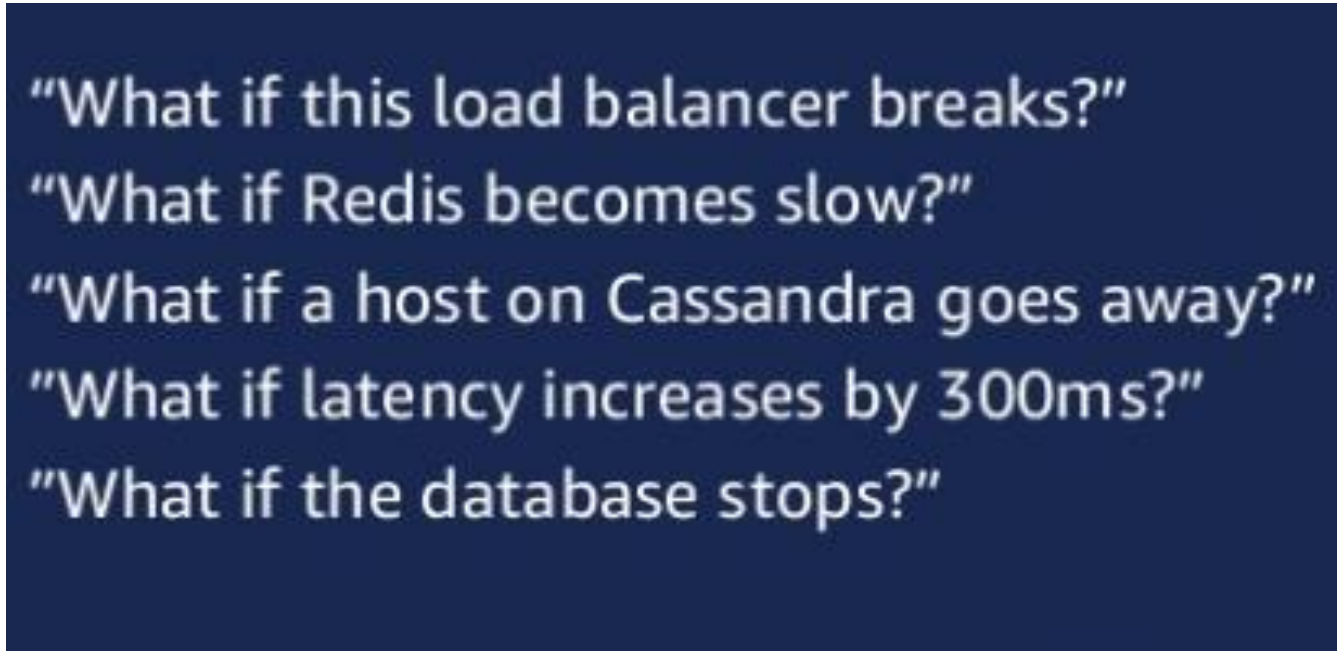
The harder it is to disrupt the steady state, the more confidence we have in the behavior of the system. If a weakness is uncovered, we now have a target for improvement before that behavior manifests in the system at large.

BASIC PRINCIPALS – PREFERRED STATE



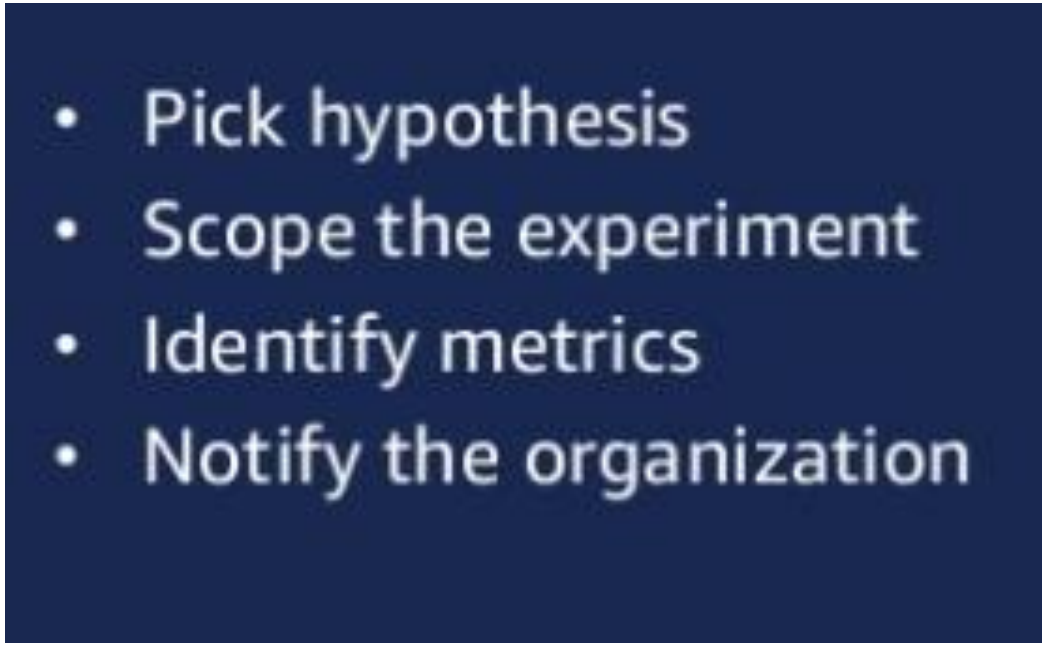
1. Start by defining ‘steady state’ as some measurable output of a system that indicates normal behavior.

BASIC PRINCIPALS – HYPOTHESIS



2. Hypothesize that this steady state will continue in both the control group and the experimental group.

BASIC PRINCIPALS – DESIGN

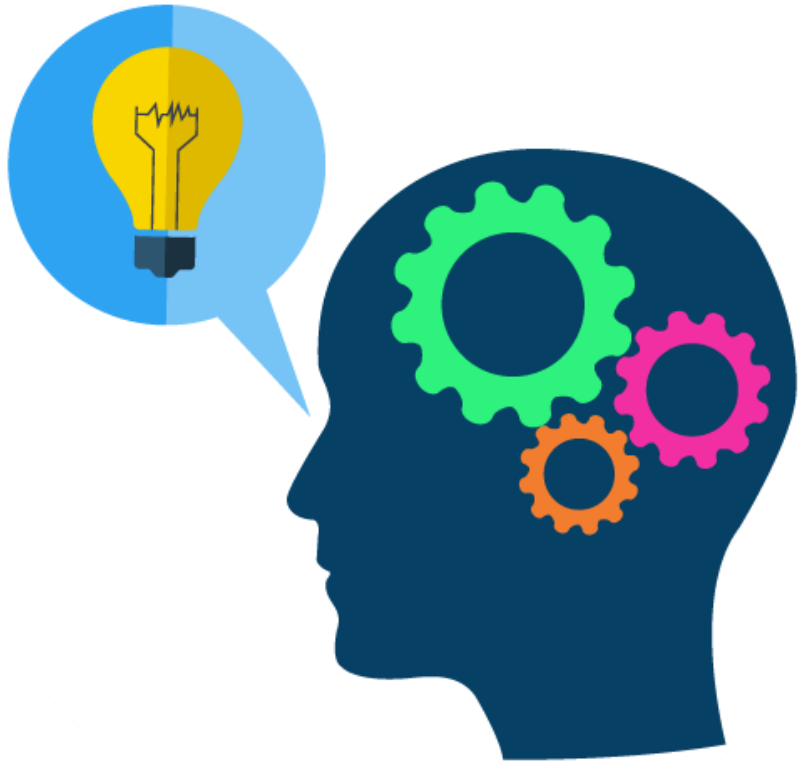


3. Decide and design what the experiment will entail, who needs to know, and the blast radius of the changes.

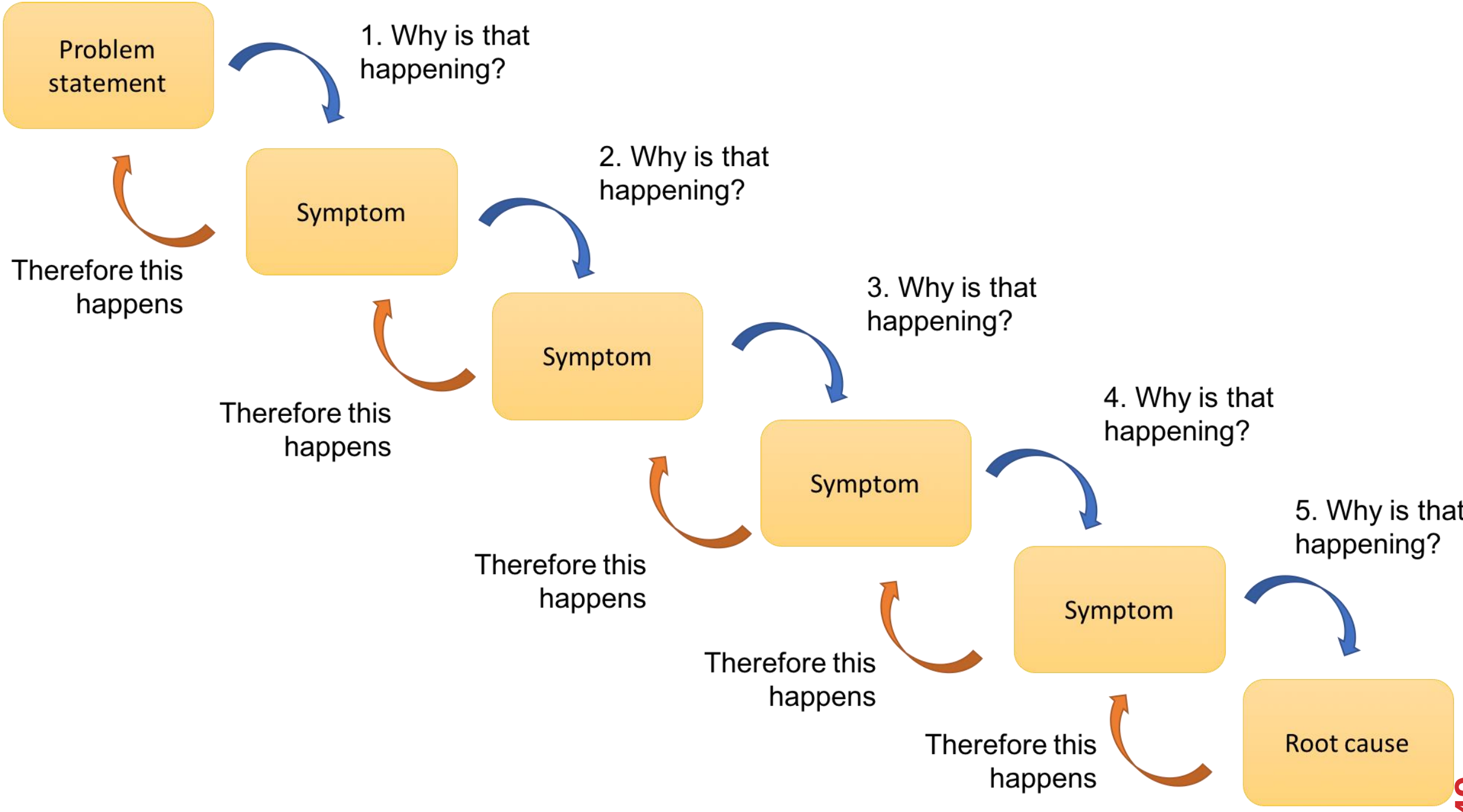
BASIC PRINCIPALS – BREAK STUFF

4. Introduce variables that reflect real world events like servers that crash, hard drives that malfunction, network connections that are severed, etc.

BASIC PRINCIPALS – LEARN



5. Try to disprove the hypothesis by looking for a difference in steady state between the control group and the experimental group.



Example – Problem: "THE SERVER CRASHES TOO OFTEN"

Why?

Because a new API was pushed to that server

Why?

Because we just launched a new feature that uses

Why?

Because we had an engineer that was new and didn't know how to use that API properly

Why?

Because that engineer was never trained

Why?

Because their manager didn't believe in training

BASIC PRINCIPALS – FIX

6. Correct the issue and rerun the experiment to validate that the learning was correct.

POP QUIZ: CHAOS ENGINEERING

Is Chaos Engineering about causing Chaos?

2 MINUTE

POP QUIZ: CHAOS ENGINEERING

Is Chaos Engineering about causing Chaos?

Of course not. The whole point is to stop chaos at unpredictable or understaffed moments in time. Break things intentionally during people's normal work hours rather than on the weekend or during a major holiday.

2 MINUTE

CHAOS CONCEPTS: BLAST RADIUS

The blast radius maps our **scope of intention**. We want to be as **precise** with this as possible, to **break only what we intend to break** and control the variables as much as possible. When first starting out it is critical to keep the blast radius as **small as possible**, and to **abort experiments** as soon as something unexpected is affected

- Application
- Caching
- Backing Service
- Server
- Cluster
- Rack
- Region
- Killswitch

CHAOS CONCEPTS: AVAILABILITY

Chaos engineering is specifically about keeping systems available during attacks, mistakes, and failures

POP QUIZ: CHAOS ENGINEERING

Do we create experiments for things we know will break?

2 MINUTE

POP QUIZ: CHAOS ENGINEERING

2 MINUTE

Do we create experiments for things we know will break?

No. When we know we have a single point of failure we don't need to test that killing that SPOF is bad. Instead create experiments in areas that we think could be problematic like DDOS, latency, server redundance, self-healing and failover.

PLANNING YOUR FIRST CHAOS EXPERIMENT

1. Ask each team member to ask themselves, *"What could go wrong?“.*
2. Each person thinks about the services and environments, and reviews potential weaknesses and expected outcomes.
3. Document your answers to the question to inform priorities about which scenarios are more likely (or more frightening) and should be tested first.
4. Sit down as a team, let each person express their thoughts, and whiteboard the service(s), dependencies (both internal and external), and data stores.
5. Grok a team picture of *"What could go wrong?"*.

Note: When in doubt, injecting a failure or a delay into each of your dependencies is a great place to start.

Lab 2- Our First Chaos Experiment

Lab 2- Our First Chaos Experiment

In this scenario you will learn how to write a simple Chaos Toolkit experiment and then use the chaos run command to execute your experiment.

https://katacoda.com/chaostoolkit/courses/01-chaostoolkit-getting-started/simple-experiment

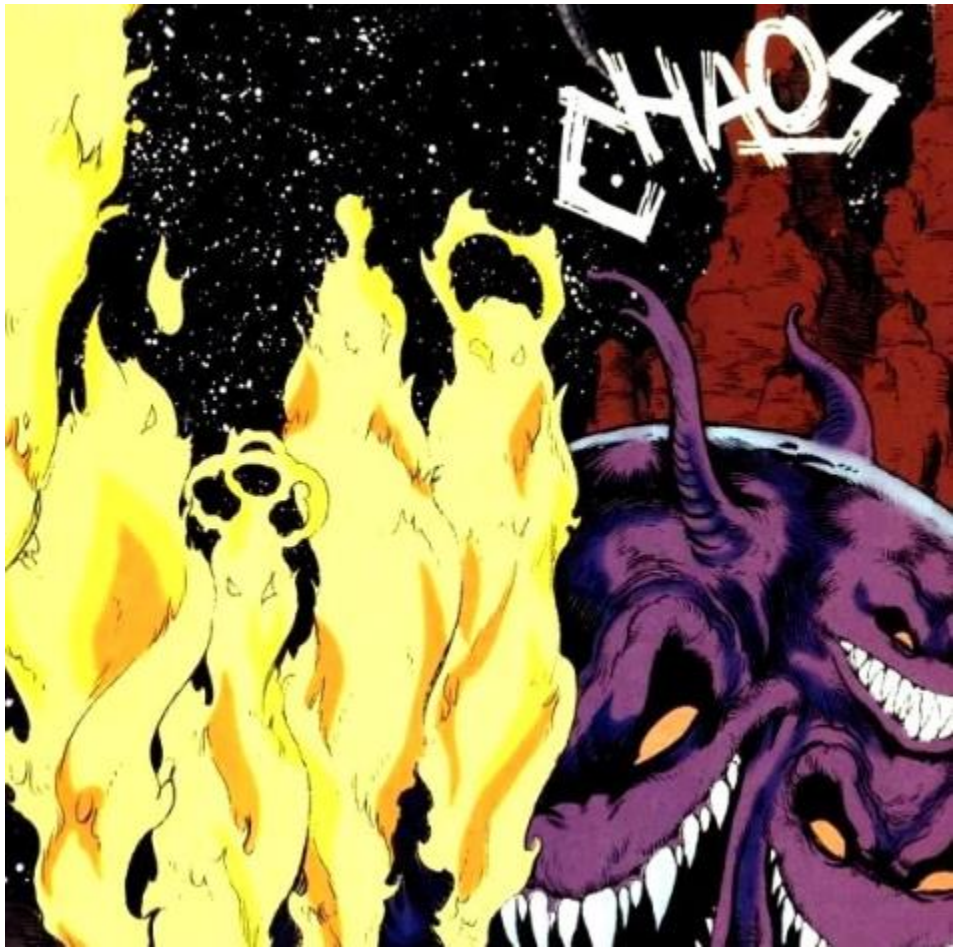
COMMON CHAOS EXPERIMENTS – RESOURCE EXHAUSTION

Resources on computers are finite. A machine/VM/container will inevitably hit a resource limit at some point, and the application will be forced to handle the lack of a resource. Commonly, this is CPU, Memory, or I/O.

COMMON CHAOS EXPERIMENTS – NETWORK AVAILABILITY

Network dependencies are a fact of life in a distributed system, and as distributed systems are growing in adoption AND complexity, chaos engineering becomes an optimal way to test for potential failures on the path to increasing resilience.

COMMON CHAOS EXPERIMENTS – BACKING SATURATION



There are a variety of ways that an application may overwhelm a data store (poor queries, lack of indices, bad sharding, upstream caching decisions, etc), but all of them result in what appears to be an unresponsive data layer.

COMMON CHAOS EXPERIMENTS – DNS FAILURE

It's easy to forget the critical role that DNS plays in keeping our systems running. Many companies experienced customer-facing issues when a real-world DNS failure occurred in October 2016. Because a failure like this is relatively rare, getting a recovery plan together is challenging.

COMMON CHAOS EXPERIMENTS: REDEFINING FAILURE

What can fail informs us on what we need to build experiments, probes and monitoring around for the organization.

A fairy tale...

Once upon a time, in theory, if everything works perfectly, we have a plan to survive the disasters we thought of in advance.

Taxonomy Infrastructure failures

Device failures
Disk, power supply, cabling, circuit board, firmware

CPU failures
Cache corruption, logic bugs

Datacenter failures
Power, connectivity, cooling, fire, flood, wind, quake

Internet failures
DNS, ISP, internet routes

Taxonomy Infrastructure failures

Device failures
Disk, power supply, cabling, circuit board, firmware

CPU failures
Cache corruption, logic bugs

Datacenter failures
Power, connectivity, cooling, fire, flood, wind, quake

Internet failures
DNS, ISP, internet routes

Taxonomy Software stack failures

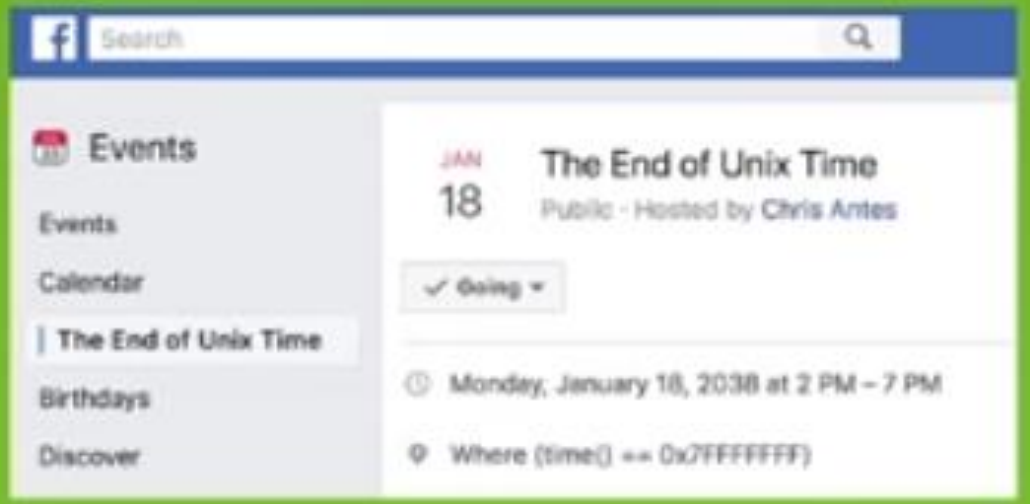
Time bombs

Counter wrap round, memory leak

Date bombs

Leap year, leap second, epoch

End of Unix Time



Taxonomy Software stack failures

Expiration
Certificates timing out

Revocation
License or account shut down by supplier

Exploit
Security failures e.g. Heartbleed

Taxonomy Software stack failures

Language bugs
Compiler, interpreter

Runtime bugs
JVM, Docker, Linux, Hypervisor

Protocol problems
Latency dependent or poor error recovery

Taxonomy Application failures

Time bombs (in application code)
Counter wrap round, memory leak

Date bombs (in application code)
Leap year, leap second, epoch, Y2K

Content bombs
Data dependent failures

Taxonomy Application failures

Configuration
Wrong config or bad syntax

Versioning
Incompatible mixes

Taxonomy Application failures

Cascading failures
Error handling bugs

Cascading overload
Excessive logging, lock contention, hysteresis

Retry Storms
Too many retries, work amplification, bad timeout strategy

Taxonomy Operations failures

Poor capacity planning

Inadequate incident management

Failure to initiate incident

Unable to access monitoring dashboards

Insufficient observability of systems

Incorrect corrective actions

CHAOS ENGINEERING – DEFINED AGAIN

In today's interconnected, internet-based world, no one is safe from system failure. The only way to minimize the impact from failure to your customers, employees, partners, your reputation, and your bottom line, is to proactively address it upfront. Chaos engineering is the optimal way to do this.

CENTRALIZED VS DISTRIBUTED CHAOS

- Points of failure. Centralized systems are easier to manage but can have single points of failure (SPOF).
- Fault / Tolerance. Decentralized are resilient because of the redundancy of effort.
- Scalability. Centralized - limited scalability. Distributed - more scalable. Decentralized - infinite scalability.
- Development. Centralized is simple and technically easier to develop. Decentralized requires more considerations and more technically complex.
- Security. Centralized may be less secure with a single point of attack (SPOA) for DDOS and similar vectors. Decentralized, especially emerging technology like chaos engineering and AI/ML, become more secure as they scale, but may need security scrutiny at the edge with 3rd party vendors, external devices (IoT) and Edge Computing.

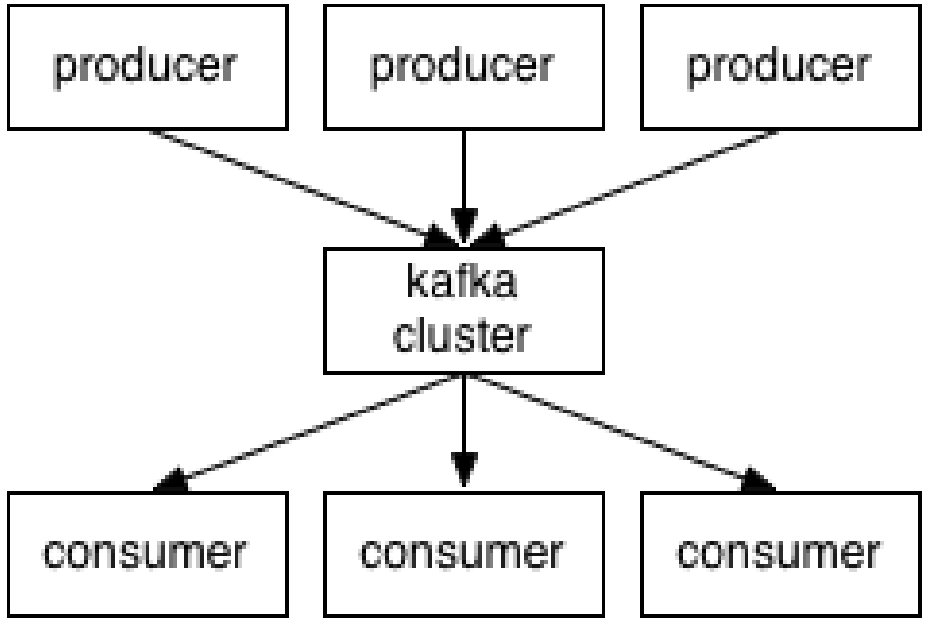
CHAOS ENGINEERING CONSENSUS

Speaking of consensus…

In terms of SRE architecture, there is no central authority making chaos decisions as a governing body. Different schools of thought drift to open source vs. developed tools. When starting out in the Chaos Engineering world there is value in having paid expertise from CE Coaches or CE Tool Vendors (i.e. Gremlin)

DEFINING A CHAOS ENGINEERING EXPERIMENT

Application name: Kafka Consumer/Producer Application
Real World Scenario / Question: What happens when we experience packet loss between our application and the message queue?
The Hypothesis: much like the the latency test, we expect queues to fill, potentially losing some data due to failure to complete a transaction
Monitoring Tools: Humans / native Linux tools / Datadog
The Experiment: Network Chaos; 10% packet loss; Scope: Single Producer Node <-> Message Queue, TCP Port 6667; Duration: 10 minutes
Abort conditions: Data loss; Excessive 500 errors for users or consuming services; Compound latency beyond 1000ms in any consuming service.
The Results: Run the test and record your results, as they will be unique to your application and environment



Chaos Toolkit

Key Concepts

ELEMENTS OF THE CHAOS TOOLKIT

```
   "version": "1.0.0",
    "title": "System is resilient to provider's
failures",
    "description": "Can our consumer survive gracefully
a provider's failure?",
    "tags": [
        "service",
        "kubernetes",
        "spring"
    ],
    "steady-state-hypothesis": {
        "title": "Services are all available and
healthy",
        "probes": [
            {
                "type": "probe",
                "name": "all-services-are-healthy",
                "tolerance": true,
                "provider": {
                    "type": "python",
                    "module": "chaosk8s.probes",
                    "func": "all_microservices_healthy"
```

The key concepts of the Chaos Toolkit are `Experiments`, `Steady State Hypothesis` and the experiment's `Method`. The Method contains a combination of `Probes` and `Actions`.

```
    "method": [
        {
            "type": "action",
            "name": "stop-provider-service",
            "provider": {
                "type": "python",
                "module": "chaosk8s.actions",
                "func": "kill_microservice",
                "arguments": {
                    "name": "my-provider-service"
                }
            },
            "pauses": {
                "after": 10 } },
        {  "ref": "all-services-are-healthy“
},
        {  "type": "probe",
            "name": "consumer-service-must-still-
respond",
            "provider": {
                "type": "http",
                "url":
"http://192.168.42.58:31018/invokeConsumedService"
            }
        }
    ],
    "rollbacks": []
```

The key concepts of the Chaos Toolkit are `Experiments`, `Steady State Hypothesis` and the experiment's `Method`. The Method contains a combination of `Probes` and `Actions`.

DEFINING STEADY STATE PROBES

```
{
    "type": "probe",
    "name": "the-sunset-service-must-be-running",
    "tolerance": true,
    "provider": {
        "type": "python",
        "module": "os.path",
        "func": "exists",
        "arguments": {
            "path": "sunset.pid"
        }
    }
}
```

Here we define a Probe element, which we've added to our steady state Probes list that checks for a file on the filesystem.

WHAT ARE PROBES?

A Probe is an element that collects system information, such as checking the health status of a node. A probe is a way of observing a particular set of conditions in the system that is undergoing experimentation.

DEFINING STEADY STATE PROBES

```
{
    "type": "probe",
    "name": "we-can-request-sunset",
    "tolerance": 200,
    "provider": {
        "type": "http",
        "timeout": 3,
        "verify_tls": false,
        "url": "https://localhost:8443/city/Paris"
    }
}
```

Here we define a Probe element, which we've added to our steady state Probes list above, that executes a URL request.

DEFINING CLOSE STATE PROBES

```
{
    "type": "probe",
    "name": "read-tls-cert-expiry-date",
    "provider": {
        "type": "process",
        "path": "openssl",
        "arguments": "x509 -enddate -noout -in cert.pem"
    }
},
```

Here we define a Probe element, which we've added to our close state Probes list.

DEFINING EXPERIMENTAL METHODS

The **Method describes** the sequence of **Probe** and **Action** elements to apply. The Method is declared under method property at the top-level of the experiment.

The method MUST have at least one element but this can be either a Probe or an Action.

The elements MUST be applied in the order they are declared.

An experiment's activities are contained within its Method block.

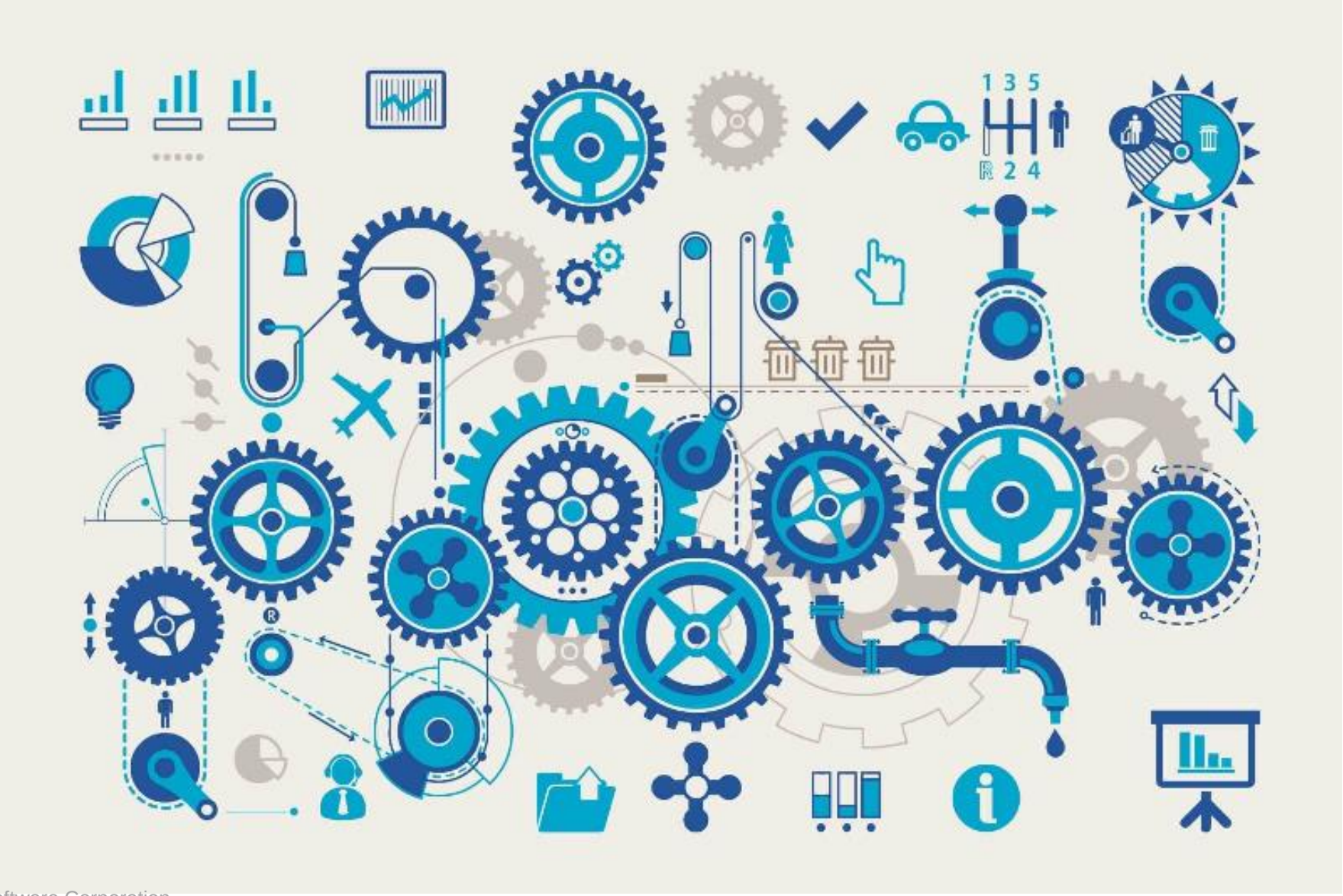
MINIMIZING THE IMPACT OF EXPERIMENTS

An essential principle of chaos engineering is minimizing the blast radius - the potential impact of the experiment you're running. Instead of testing on every host in our production environment, we can start with a single host in our test environment. This allows us to grow our confidence in the system and our understanding in lockstep with scale and the potential risk.

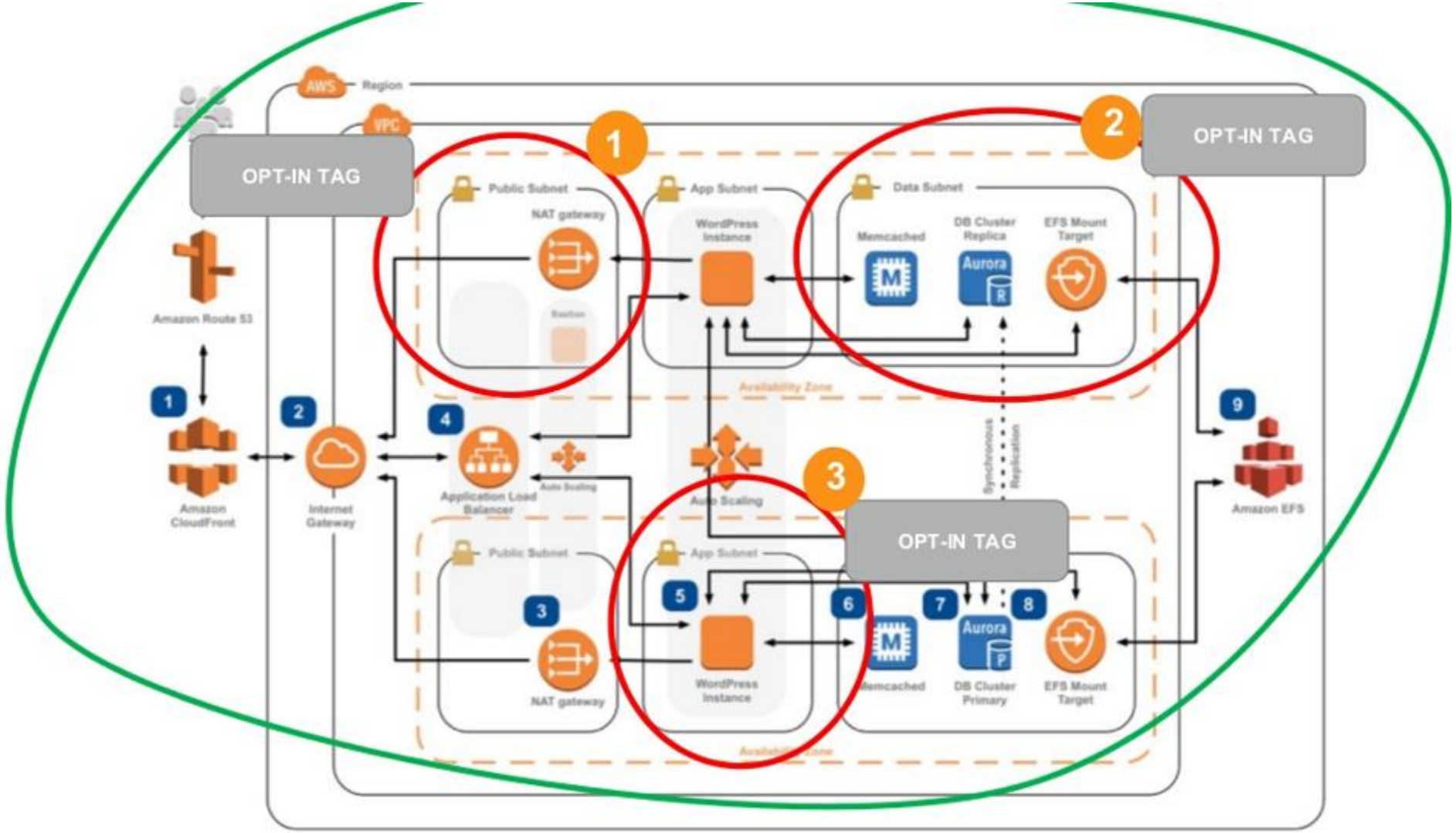
MINIMIZING THE IMPACT OF EXPERIMENTS

Everyone involved or impacted needs to be aware of ongoing experiments: talk to stakeholders, set up dashboards, send out email or chat notifications. Visibility is a crucial aspect of Chaos Engineering, and even more so once you have CE automation always at work in the background.

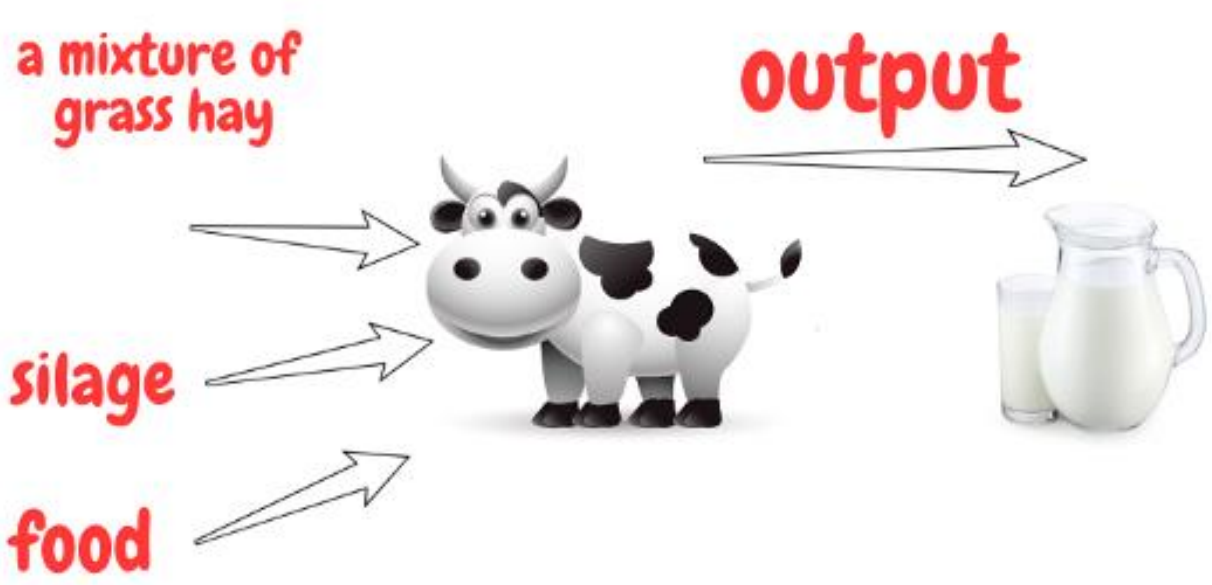
UNDERSTANDING YOUR EXPERIMENT'S BLAST RADIUS, SCOPE & OPT-OUT



UNDERSTANDING YOUR EXPERIMENT'S BLAST RADIUS, SCOPE & OPT-OUT



IDEMPOTENCY WITH CHAOS ENGINEERING EXPERIMENTS



Idempotence is not only for CE experiments but is a key component of the antifragile systems including microservices architectures.

POP QUIZ: EXPERIMENTS

Do some nodes voluntarily not receive compensation?

5 MINUTES

POP QUIZ: PROBES

Do some nodes voluntarily not receive compensation?

5 MINUTES

POP QUIZ: HYPOTHESIS

Do some nodes voluntarily not receive compensation?

5 MINUTES

Lab 3- Discover Experimentation

Lab 3- Discover Experimentation

While you can write your experiments from scratch, it is the mission of the Chaos Toolkit to make it as easy and quick as possible to **create your own experiments** and so the toolkit also comes with the chaos discover and chaos init commands.

Rather than starting from a blank slate, the **chaos discover** and **chaos init** commands can help you to explore your current target environment to see what can be acted or probed upon, and then to generate great starting points for your own chaos engineering experiments.

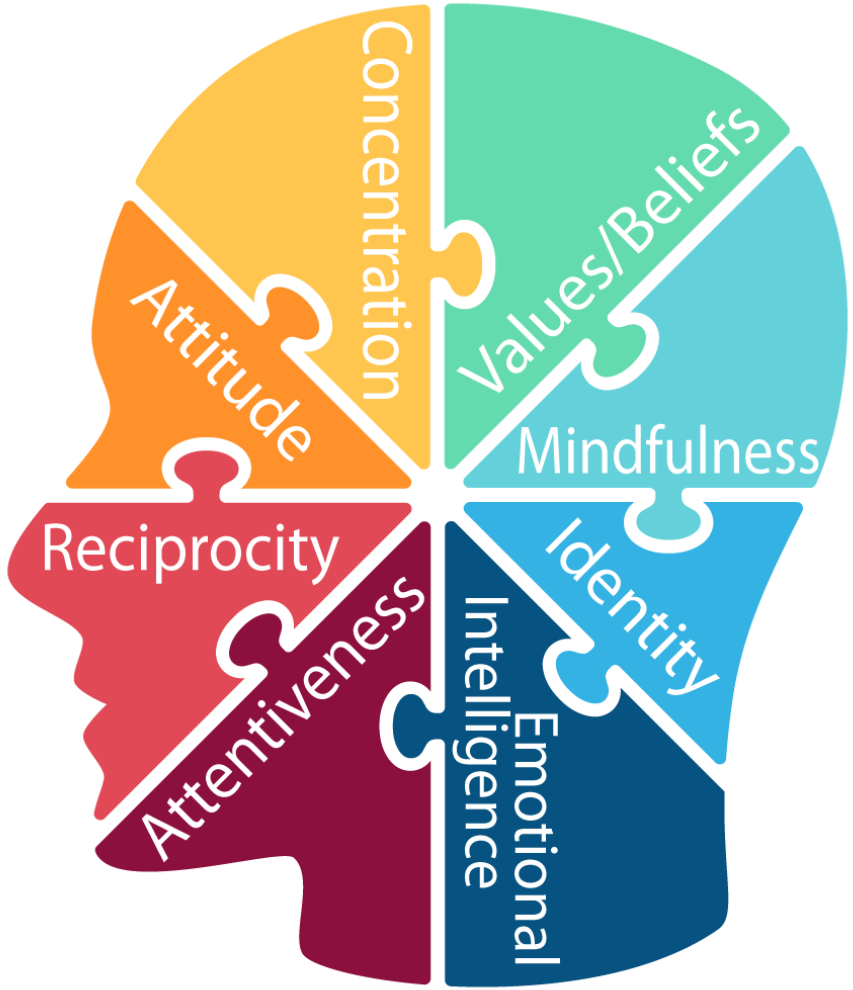
https://katacoda.com/chaostoolkit/courses/01-chaostoolkit-getting-started/discovery-and-init

Operationalizing Chaos

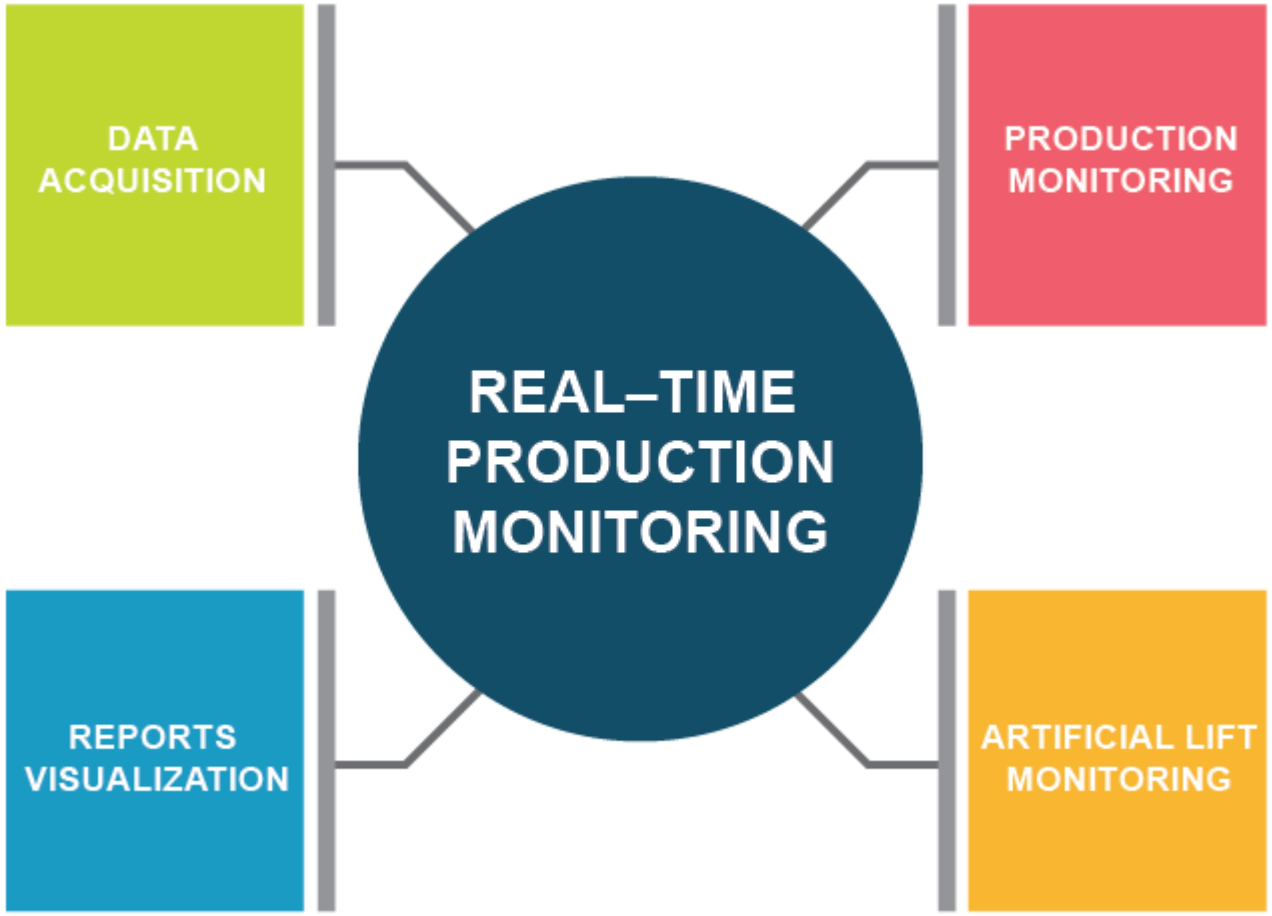
In the Enterprise

RESPONSIBILITY WITH CHAOS ENGINEERING

Chaos Engineers are not creating havoc, rather they are following a scientific approach to experiments that identify issues that could cause impacts. It is the responsibility of the engineering team to discover and verify failure domains to ensure that product requirements are met.



DESIGNING AND IMPLEMENTING STEADY STATE MONITORING FOR YOUR SYSTEM

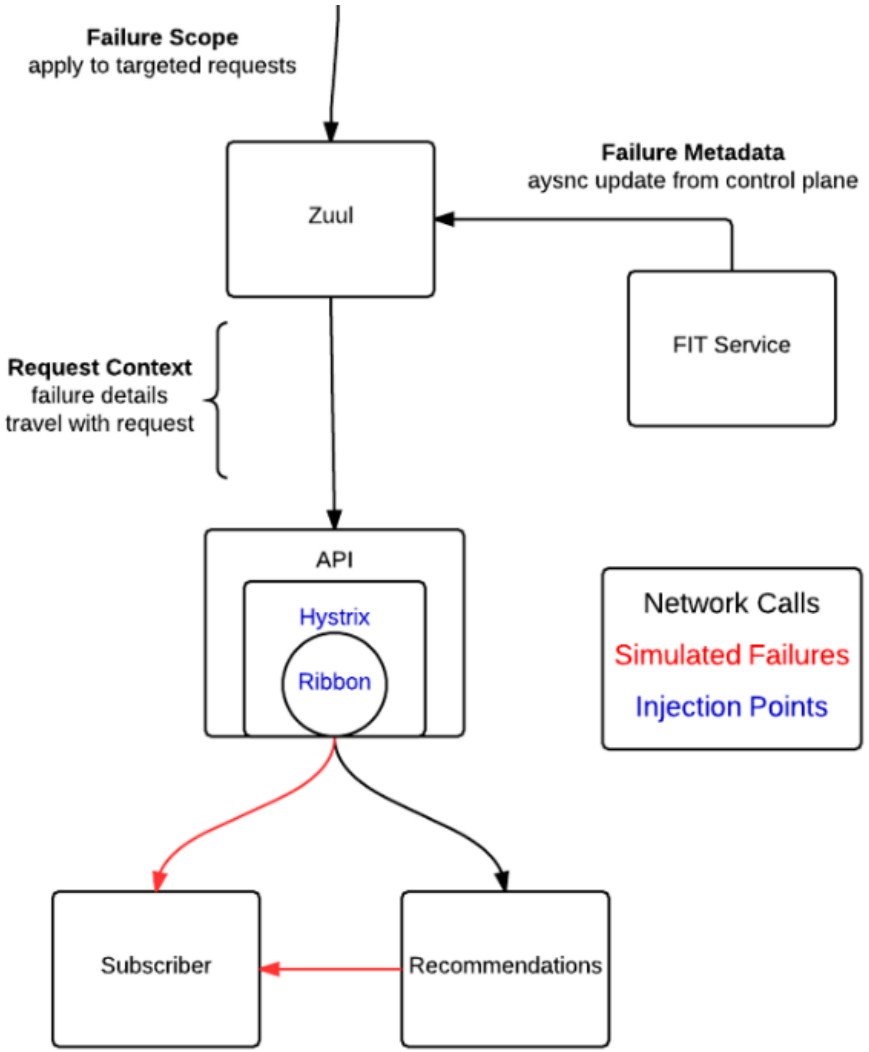


An essential element of Chaos Engineering is a **monitoring** system that you can use to **determine the current state** of your system. Without **visibility** into your system's behavior, you won't be able to draw **conclusions** from your **experiments**

SECURITY CHAOS ENGINEERING

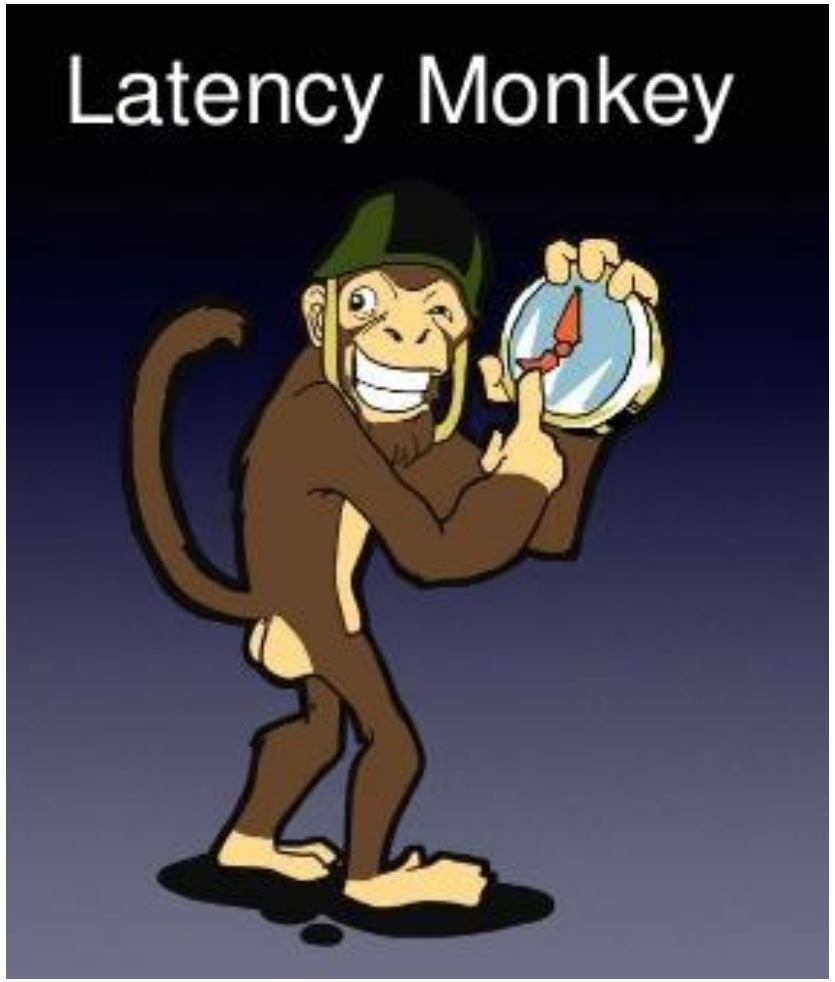
Security Chaos Engineering is the discipline of instrumentation, identification, and remediation of failure within security controls through proactive experimentation to build confidence in the system's ability to defend against malicious conditions in production.

FAILURE INJECTION TESTING IN CONTEXT



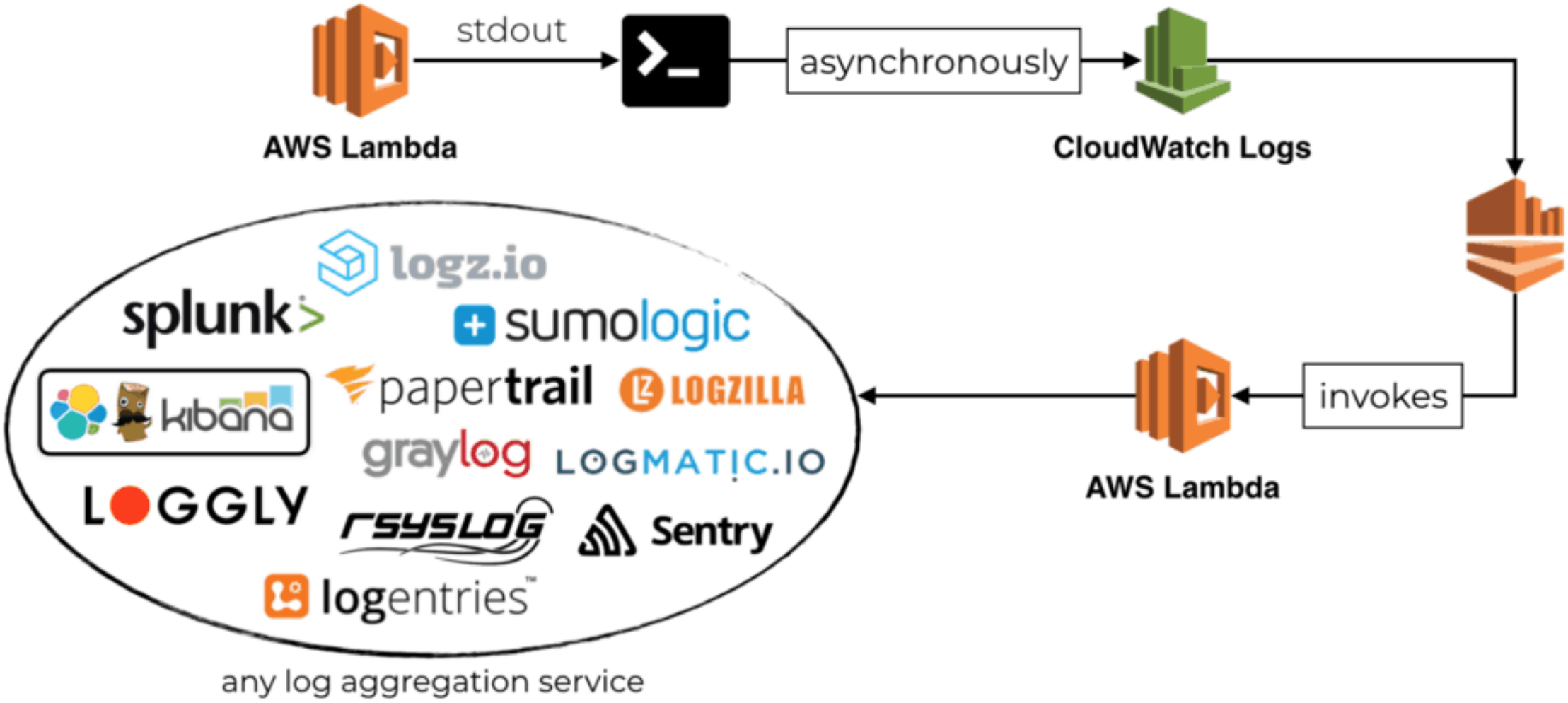
FIT allows an engineer to add a failure scenarios to application, network, routing or data flow. As those scenarios propagate through the system, injection points between components or microservices will check for the failure scenario and take some action based on the scenario.

FAILURE INJECTION TESTING IN ACTION

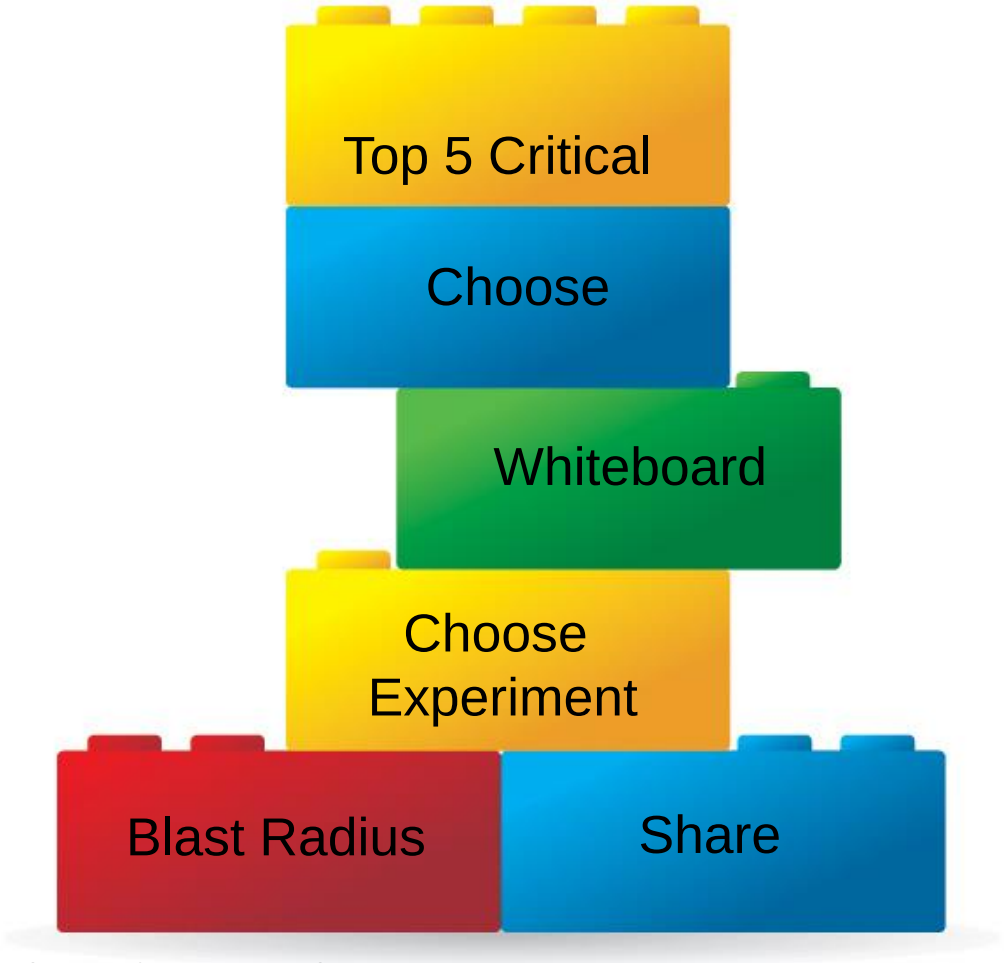


Latency monkey adds a delay and/or failure on the server side of a request for a given service. This provides us good insight into how calling applications behave when their dependency slows down—threads pile up, the network becomes congested, etc.

CHAOS LOGGING



BRAINSTORMING PLATFORM-LEVEL CHAOS ENGINEERING EXPERIMENTS

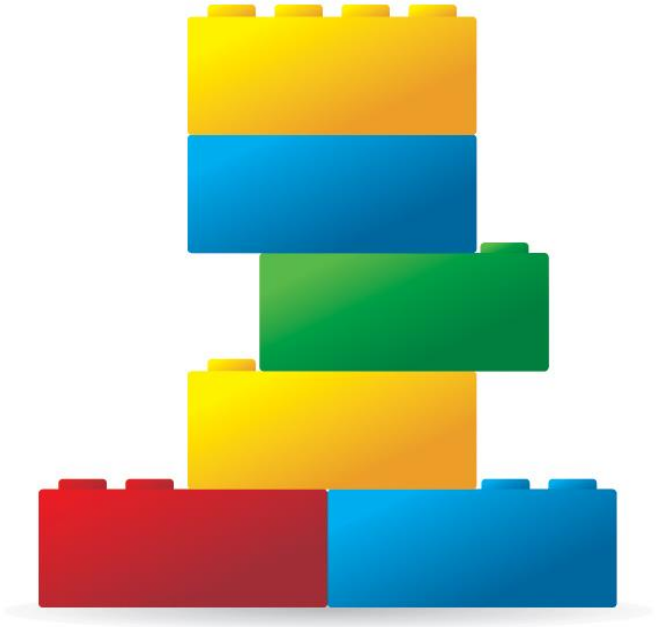


Every company, team or line of business has different applications, networking, applications, resources, people and effective blast radius.

IF YOU NEED INSPIRATION FOR PLATFORM-LEVEL CHAOS ENGINEERING EXPERIMENTS

https://github.com/danluu/post-mortems

Everyone makes mistakes including some that you may have heard of as inspirational building blocks.



DEFINING PLATFORM STEADY STATE PROBES

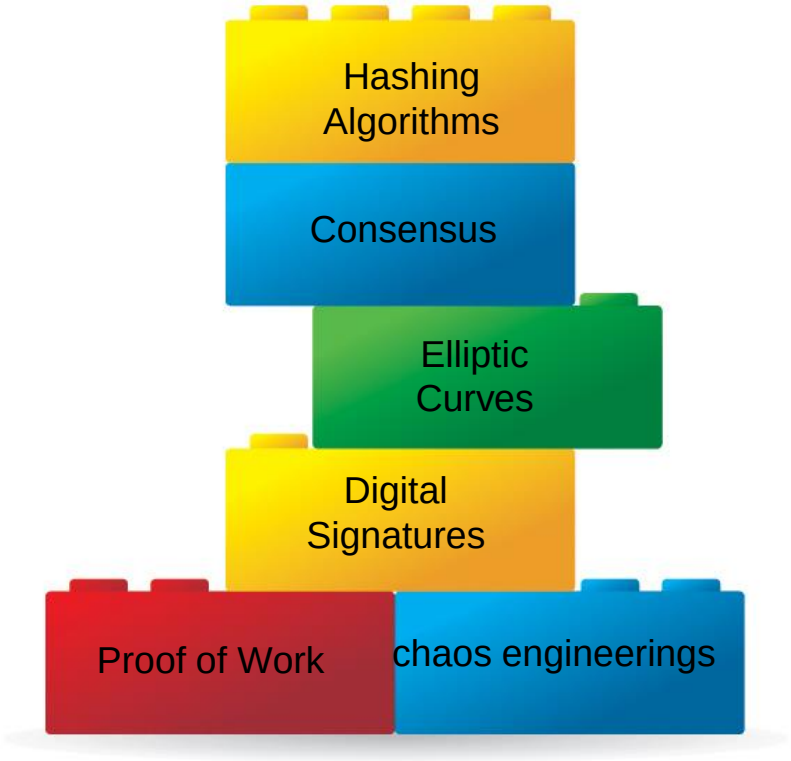
```
"steady-state-hypothesis": {
  "title": "Service should have nodes.",
  "probes": [
    {
      "type": "probe",
      "name": "nodes_found",
      "tolerance": true,
      "provider": {
        "type": "python",
        "module": "chaosk8s.node.probes",
        "func": "get_nodes",
        "arguments": {
          "label_selector": "eks-gremlin-chaos"
        }
      }
    }
  ]
```

A Probe is an element that collects system information, such as checking the health status of a node. Here we define a Probe element, which we've added to our steady state Probes list above, that calls the get_nodes function and retrieves the list of nodes for the specified label-selector.

CHAOS ENGINEER

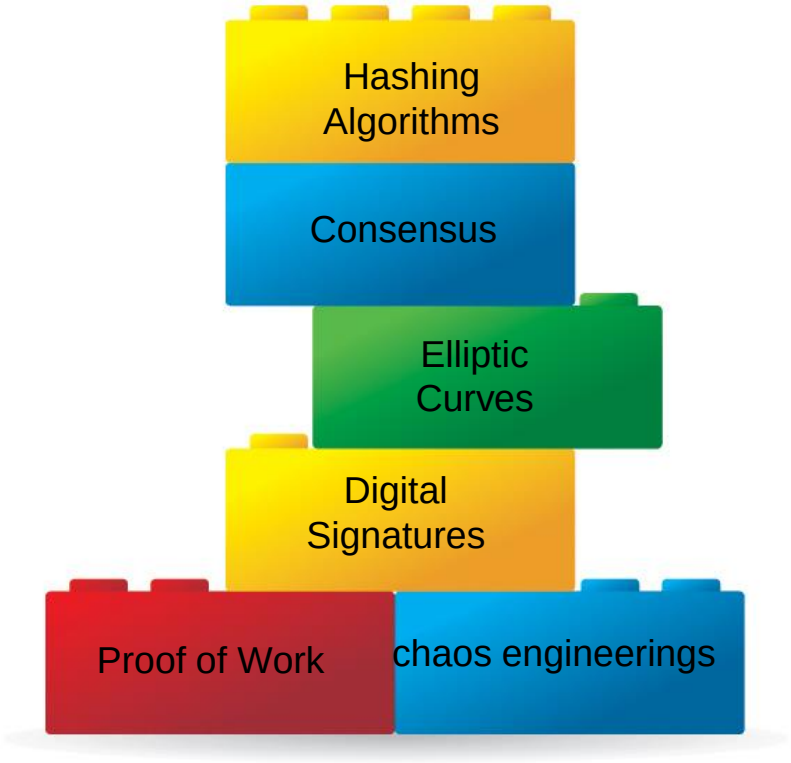
A person helping companies avoid outages by running proactive chaos engineering experiments

ESTABLISHING PLATFORM CLOSE STATE PROBES



There are a variety of concepts, or building blocks, that are required to implement a chaos engineering. This module covers those topics, such as hashing algorithms, digital signatures, and more.

RUNNING PLATFORM EXPERIMENTS MANUALLY AND THROUGH AUTOMATION



There are a variety of concepts, or building blocks, that are required to implement a chaos engineering. This module covers those topics, such as hashing algorithms, digital signatures, and more.

Chaos Tooling

Variety for Every Platform

PLATFORM TESTING TOOLS – KUBE MONKEY

An implementation of Netflix's Chaos Monkey for Kubernetes clusters

github.com/asobti/kube-monkey

PLATFORM TESTING TOOLS – PUMBA

Chaos testing and network emulation tool for Docker.

github.com/alexei-led/pumba

PLATFORM TESTING TOOLS – POWERFULSEAL

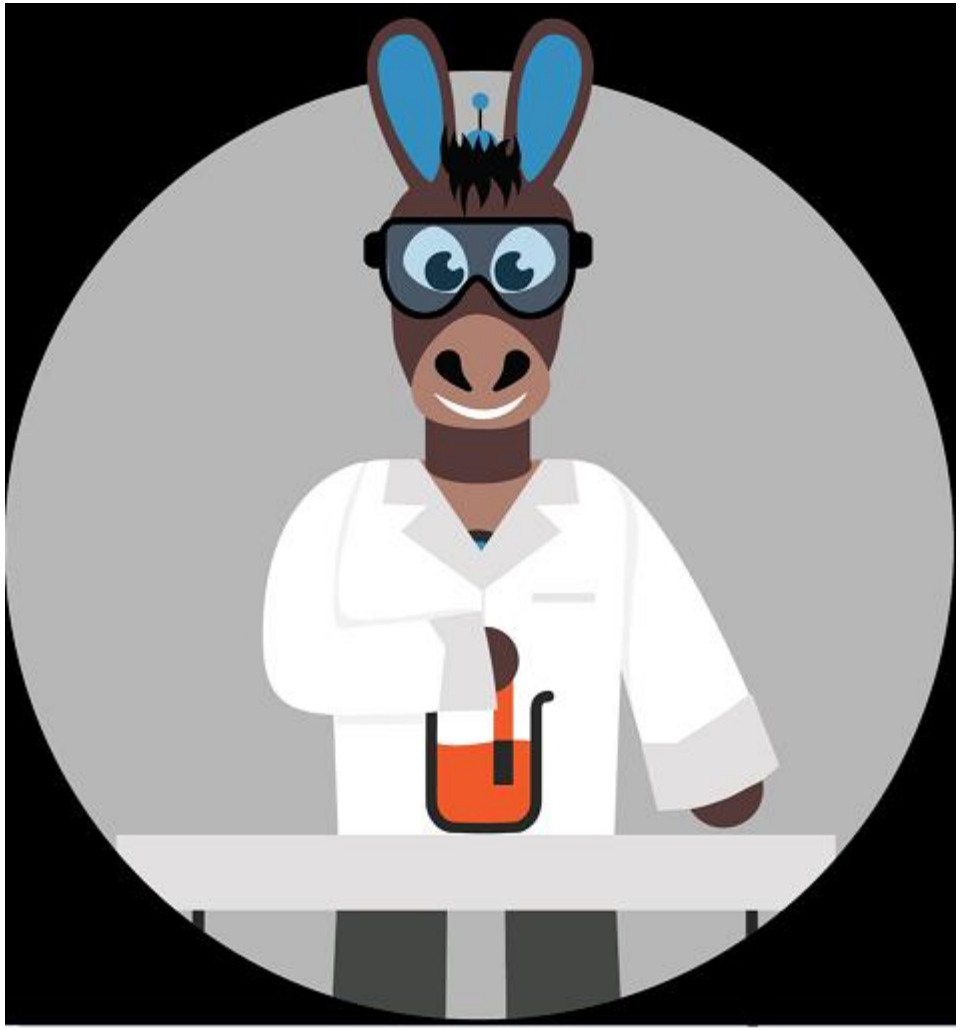
A powerful testing tool for OpenStack Kubernetes clusters

github.com/bloomberg/powerfulseal

PLATFORM TESTING TOOLS – LITMUS

Chaos engineering tool for stateful workloads on Kubernetes, hopefully without learning curves.

github.com/openebs/litmus



PLATFORM TESTING TOOLS – CHAOS LAMBDA

EC2 instances are volatile and can be recycled at any time without warning. Amazon recommends running them under Auto Scaling Groups to ensure overall service availability.

Chaos Lambda increases the rate at which these failures occur during business hours, helping teams to build services that handle them gracefully.

github.com/openebs/litmus

PLATFORM TESTING TOOLS – CHAOS LEMUR

This project is a self-hostable application to randomly destroy virtual machines in a BOSH-managed environment Open Stack, PCF and VMWare environments, as an aid to resilience testing of high-availability systems.

github.com/strepsirrhini-army/chaos-lemur

Operationalizing Chaos

In the Enterprise

POP QUIZ: SOME STUFF

What is the … ?

5 MINUTES

POD RESILIENCY

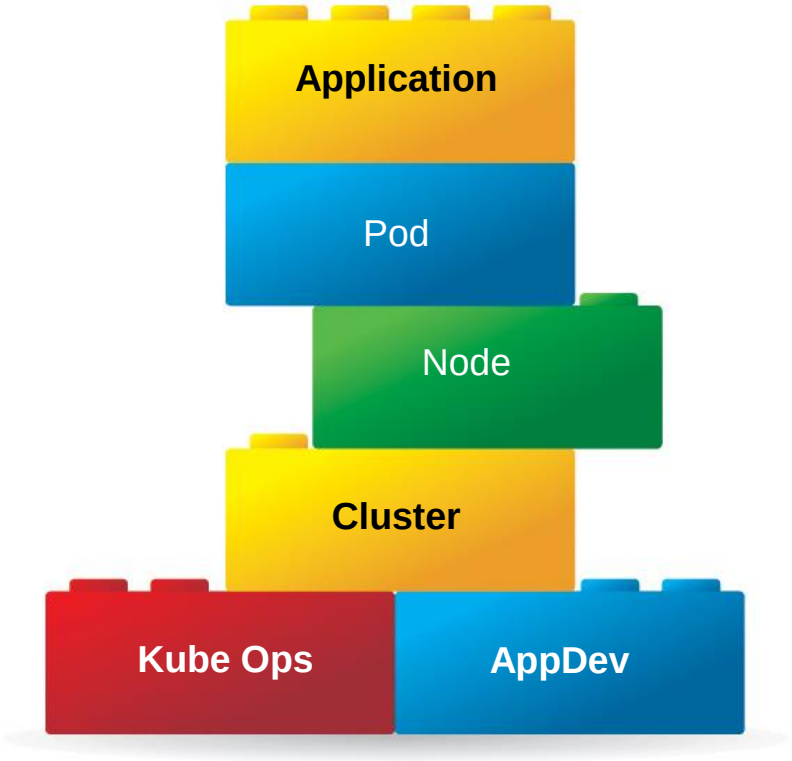
Kubernetes Pod Chaos Monkey is a very simple implementation of Netflix's Chaos Monkey for kubernetes clusters. It randomly kills kubernetes pods in the cluster to encourage and validate the development of failure-resilient services.

In a simple experiment, running an app using only one pod, we postulate that the result of using KPCM and pod crashing will be quite severe. Our hypothesis is that our application will become (at least) temporarily unavailable.

These types of experiments along with tools that inject failures or latency can cause serious issues. Ensure you understand your blast radius and never execute this type of experiment in production without ensuring a safety zone, prior to dialing up to higher %.

HOW TO FOCUS ON THE POD RESILIENCY TO BUILD CHAOS ENGINEERING EXPERIMENTS

In many system contexts there is a dividing line between those who are responsible for managing a Kubernetes cluster (including all the realcosts associated with these resources), let's call those the Kube Ops, and those trying to deploy and manage applications upon the cluster, the AppDev.



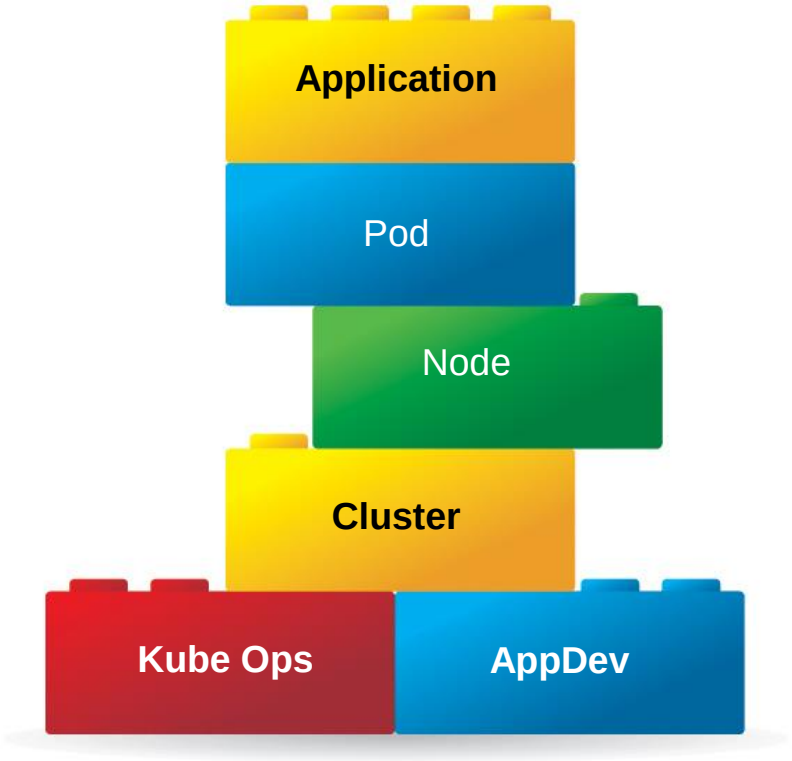
POD RESILIENCY AND THE APPDEV TEAM

Ensuring their application is healthy.

Ensuring they have enough redundancy and capacity across pods.

Ensuring they claim the persistent storage they need for their application to maintain its state.

Ensuring they can go home at EOD.



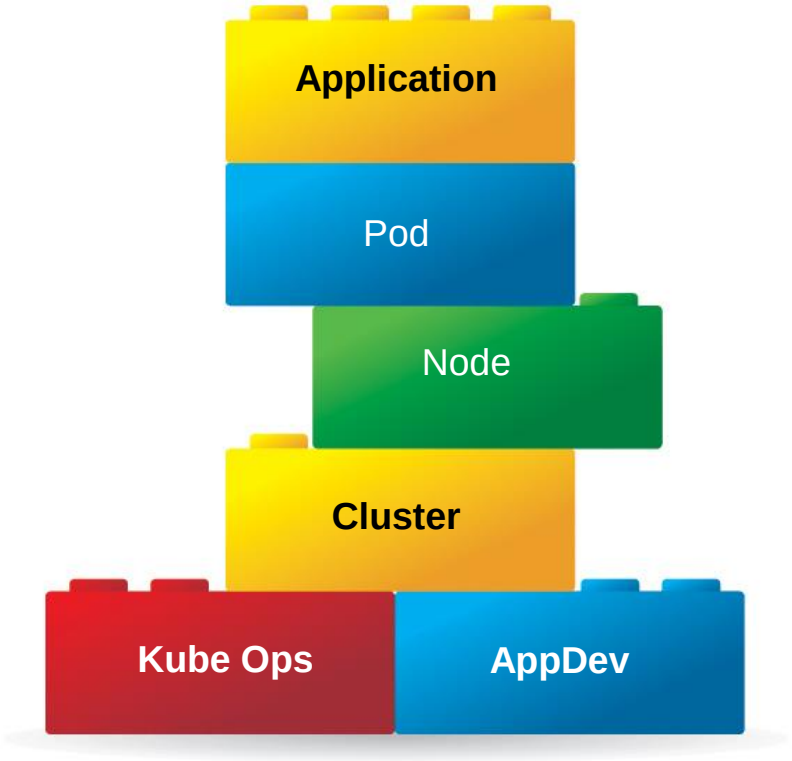
POD RESILIENCY AND THE KUBEOPS TEAM

Managing Nodes.

Managing Persistent Disks.

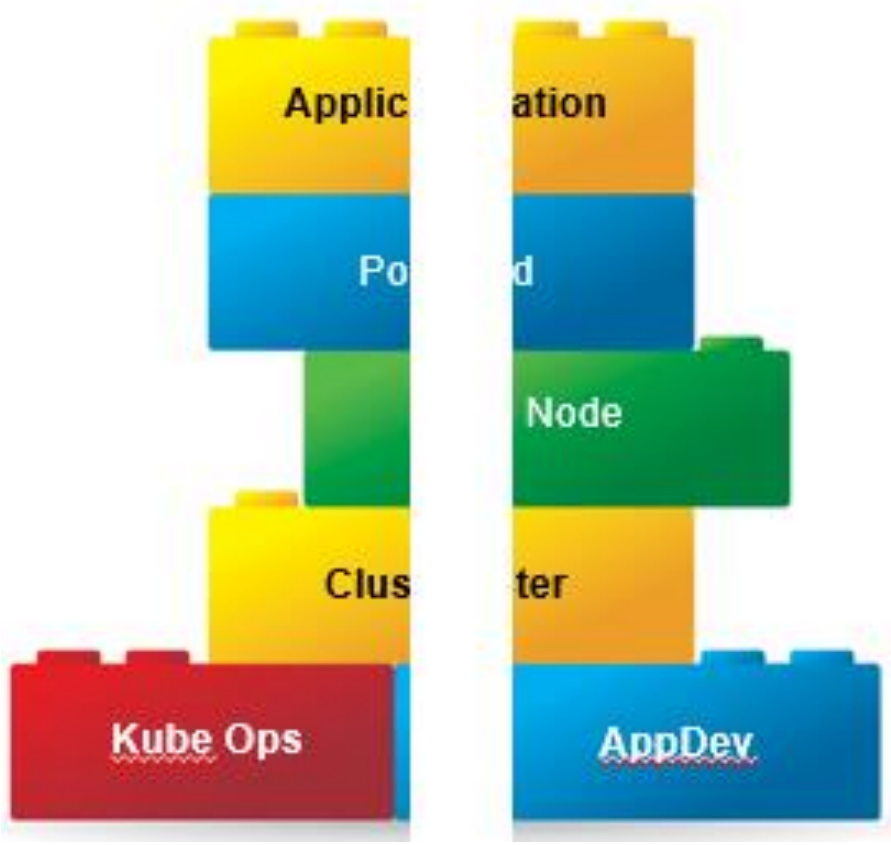
Keeping an eye on the costs for our Cloud Provider.

Ensuring the cluster is not the problem, so they don't get paged over the weekend.



POD RESILIENCY: A TALE OF TWO TEAMS - 1

A common example is a KubeOps Admin removing a node from the cluster, for maintenance, that leaves an application in a state where it cannot maintain a required number of pods. Through no fault of their own the KubeOps team and the AppDev team end up in conflict and a possible system outage.



POD RESILIENCY: A TALE OF TWO TEAMS - 2

As Chaos Engineers, when we suspect a system weakness we execute an experiment.

This weakness is at the people and process level.

The solution lies in how the two roles operate, coordinate and compromise during their daily work.

Lab 4- Reporting and Visualization

Lab 4- Reporting and Visualization

Running an experiment is helpful for immediate learning, but usually you'll want to **share the outcomes of your experiments** so your whole team, and maybe even your company at large, can learn from your experiments too. After all, **organizational learning is a big goal of Chaos Engineering!**

The Chaos Toolkit comes with a **chaos report** command for just this purpose. It turns your experiments into reports in various formats so everyone can learn from your chaos efforts.

https://katacoda.com/chaostoolkit/courses/01-chaostoolkit-getting-started/report

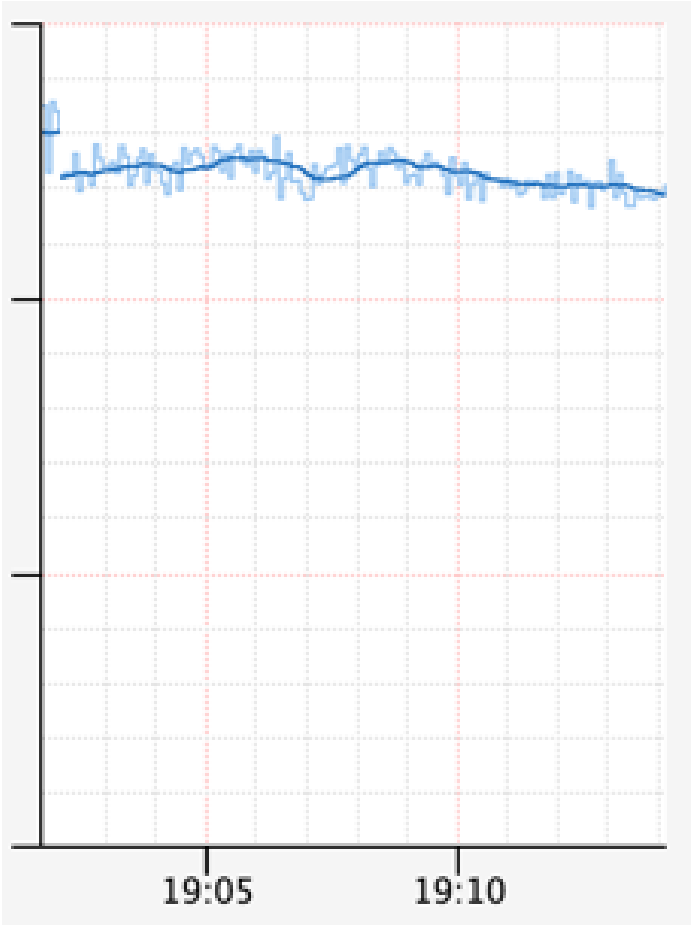
LAB NOTES

Application Chaos

In the Enterprise

FINDING THE NEEDLE IN THE HAYSTACK

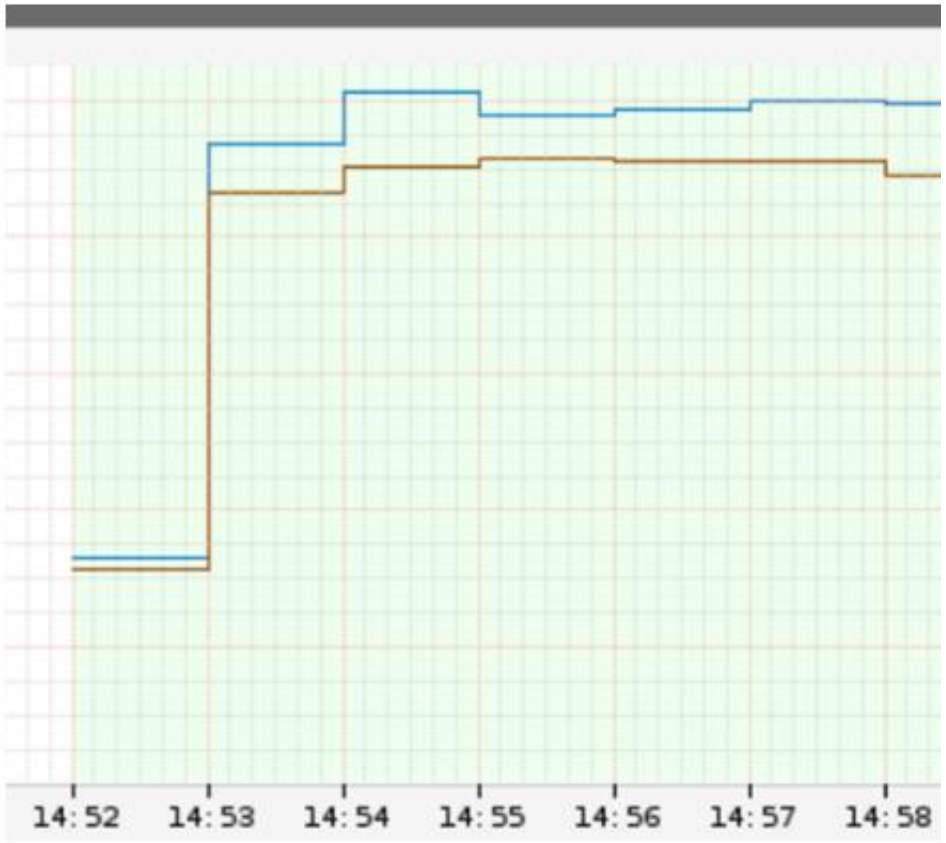
With production wide experiments, the impact affects metrics for the whole system. Statistics for the experimental population are mixed with the remaining population. The experimental population size (and the effect size) has to be large in order to be detectable in the natural noise of the system.



BRAINSTORMING APPLICATION-LEVEL CHAOS ENGINEERING EXPERIMENTS

Can you determine when the experiment ran? Did it have an impact greater than the noise of the system? In order to create bigger differences that are verifiable by humans and machines, Chaos Engineers end up running larger and longer experiments, risking unnecessary disruptions for our customers.

ESTABLISHING APPLICATION STEADY STATE PROBES



To limit this blast radius, in Netflix ChAP they take a small subset of traffic and distribute it evenly between a control and an experimental cluster. This makes it much easier for humans and computers to see when the experiment and control populations' behaviors diverge.

APPLICATION-LEVEL CHAOS ENGINEERING EXPERIMENTS

When we run Chaos Engineering experiments, we are interested in the behavior of the entire overall system. The **code** is an important part of the system, but there's a **lot more** to our system than just code. In particular, **state** and **input** and **other people's systems** lead to all sorts of system behaviors that are difficult to foresee.

CHAOS TOOLKIT SPRING EXTENSION

To enable chaos for Spring we have to add configuration

```
{
    "name": "enable_chaosmonkey",
    "provider": {
        "arguments": {
            "base_url": "http://localhost:8080/actuator"
        },
        "func": "enable_chaosmonkey",
        "module": "chaosspring.actions",
        "type": "python"
    },
    "type": "action"
}
```

CHAOS TOOLKIT SPRING EXTENSION

To enable chaos for Spring we have to add configuration

```
{
    "name": "enable_chaosmonkey",
    "provider": {
        "arguments": {
            "base_url": "http://localhost:8080/actuator"
        },
        "func": "enable_chaosmonkey",
        "module": "chaosspring.actions",
        "type": "python"
    },
    "type": "action"
}
```

CHAOS TOOLKIT SPRING EXTENSION

To enable chaos for Spring we can add that to our codebase through the Maven dependency in our *pom.xml*

```
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>chaos-monkey-spring-boot</artifactId>
    <version>2.0.0</version>
</dependency>
```

CHAOS SPRING APPLICATION PROPERTIES & YAML

application.properties:

```
spring.profiles.active=chaos-monkey
chaos.monkey.enabled=true
```

application.yml:

```yaml
spring:
  profiles:
    active: chaos-monkey
chaos:
  monkey:
    enabled: true
```

CHAOS SPRING ENABLEMENT FOR SPRING BOOT APP

By starting the application with *chaos-monkey* spring profile **we don't have to stop and start the application** if we want to **enable** or **disable it** while our app is running:

```
java -jar your-app.jar --spring.profiles.active=chaos-monkey
```

Another useful property is **management.endpoint.chaosmonkey.enabled**. Setting this property to **true** will enable the management endpoint for our Chaos Monkey:

```
http://localhost:8080/chaosmonkey
```

HOW DOES THIS STUFF WORK - WATCHER

Chaos Monkey consists of **Watchers**. A Watcher is a Spring Boot component. It makes use of **Spring AOP** to see when a public method is executed in classes annotated with the following Spring **annotations**:

- Component
- Controller
- RestController
- Service
- Repository

HOW DOES THIS STUFF WORK - ASSAULTS

Chaos Monkey consists of **Assaults**. Based on the configuration in our app properties file, our public methods will be either assaulted or not, by one of the following:

Latency Assault – This type of attack adds latency to our calls. This way our application responds slower and we can monitor how it behaves when for example the database responds slower.

Exception Assault – This tests how well our application can handle exceptions. Based on configuration it will throw a random Runtime Exception once enabled.

AppKiller Assault – not surprisingly this causes the app to die randomly

HOW DOES THIS STUFF WORK - ASSAULTS

By default, Watcher is only enabled for our services. This means that our assaults will be executed only for public methods in our classes annotated with ***@Service***.

We can easily change that default behavior by configuring properties:

chaos.monkey.watcher.controller=false
chaos.monkey.watcher.restController=false
chaos.monkey.watcher.service=true
chaos.monkey.watcher.repository=false
chaos.monkey.watcher.component=false

Note: Once the application started, we cannot dynamically change the watcher using the Chaos Monkey for Spring Boot management port that we mentioned earlier.

HOW DOES THIS STUFF WORK - ASSAULTS

By default, Watcher is only enabled for our services. This means that our assaults will be executed only for public methods in our classes annotated with ***@Service***.

We can easily change that default behavior by configuring properties:

chaos.monkey.watcher.controller=false
chaos.monkey.watcher.restController=false
chaos.monkey.watcher.service=true
chaos.monkey.watcher.repository=false
chaos.monkey.watcher.component=false

Note: Once the application started, we cannot dynamically change the watcher using the Chaos Monkey for Spring Boot management port that we mentioned earlier.

MANAGEMENT CONSOLE ENABLEMENT- LATENCY

```
curl -X POST http://localhost:8080/chaosmonkey/assaults \
-H 'Content-Type: application/json' \
-d \
'{
    "latencyRangeStart": 2000,
    "latencyRangeEnd": 5000,
    "latencyActive": true,
    "exceptionsActive": false,
    "killApplicationActive": false
}'
```

MANAGEMENT CONSOLE ENABLEMENT- EXCEPTION

```
curl -X POST http://localhost:8080/chaosmonkey/assaults \
-H 'Content-Type: application/json' \
-d \
'{
    "latencyActive": false,
    "exceptionsActive": true,
    "killApplicationActive": false
}'
```

MANAGEMENT CONSOLE ENABLEMENT- KILL APP

```
curl -X POST http://localhost:8080/chaosmonkey/assaults \
-H 'Content-Type: application/json' \
-d \
'{
    "latencyActive": false,
    "exceptionsActive": false,
    "killApplicationActive": true
}'
```

MORE COMPLETE APPLICATION CONFIGURATION

```
1   spring.application.name=ChaosDemo
2   server.port=8090
3
4   spring.profiles.active=chaos-monkey
5   chaos.monkey.enabled=false
6
7   management.endpoint.chaosmonkey.enabled=true
8
9   management.endpoints.web.exposure.include=*
10
11  chaos.monkey.assaults.latencyActive=false
12  chaos.monkey.assaults.killApplicationActive=false
13  chaos.monkey.watcher.restController=true
14
15  chaos.monkey.assaults.level=1
16
```

The default configuration for the service shows that we've turned on the watchers for incoming requests (in addition to the default for @Service invocations) in the provider application's configuration (Line 13)

EXPERIMENT EXAMPLE 1: WHAT IF A DEPENDENT SERVICE DIES? HYPOTHESIS

```
"steady-state-hypothesis": {
        "probes": [
            {
                "name": "consumer-service-must-still-respond",
                "provider": {
                    "type": "http",
                    "url": "http://localhost:8080/invokeConsumedService"
                },
                "tolerance": 200,
                "type": "probe"
            }
        ],
        "title": "System is healthy"
    },

```

EXPERIMENT EXAMPLE 1: WHAT IF A DEPENDENT SERVICE DIES? ACTION

```
"method": [
        {
            "name": "enable_chaosmonkey",
            "provider": {
                "arguments": {
                    "base_url": "http://localhost:8090/actuator"
                },
                "func": "enable_chaosmonkey",
                "module": "chaosspring.actions",
                "type": "python"
            },
            "type": "action"
        },

```

EXPERIMENT EXAMPLE 1: WHAT IF A DEPENDENT SERVICE DIES? ASSAULT

```
{   "name": "configure_assaults",
    "provider": {
        "arguments": {
            "base_url": "http://localhost:8090/actuator",
            "assaults_configuration": {
                "level": 1,
                "latencyRangeStart": 2000,
                "latencyRangeEnd": 5000,
                "latencyActive": false,
                "exceptionsActive": false,
                "killApplicationActive": true,
                "restartApplicationActive": false
            }
        },
        "func": "change_assaults_configuration",
        "module": "chaosspring.actions",
        "type": "python"
    },
    "type": "action"
},
```

EXPERIMENT EXAMPLE 1: WHAT IF A DEPENDENT SERVICE DIES? PROBE

```
{
    "name": "trigger-kill",
    "provider": {
        "type": "http",
        "url": "http://localhost:8080/"
    },
    "type": "probe"
}
```

EXPERIMENT EXAMPLE 1: WHAT IF A DEPENDENT SERVICE DIES? ROLLBACKS

```
"rollbacks": [
    {
        "type": "action",
        "name": "uncordon_node",
        "provider": {
            "type": "python",
            "module": "chaosk8s.node.actions",
            "func": "uncordon_node",
            "arguments": {
                "name": "gke-disruption-demo-default-pool-9fa7a856-jrvm"
            }
        }
    }
]
```

POP QUIZ: PARENTS AND CHILDREN

Can a chaos engineering block, have more than one parent?

Can a chaos engineering have more than one child?

10 MINUTES

Lab 5- Spring Boot Chaos Toolkit

Lab 5- Spring Boot Chaos Toolkit

This tutorial will build a simple microservice-based system using Kubernetes and then show how you can explore that system using Chaos Engineering to discover and overcome a weakness in the form of a cascading failure.

All of the sample code for this tutorial can be found here.

https://katacoda.com/chaostoolkit/courses/02-chaos-engineering-experiments-with-chaostoolkit/service-down-not-visible-to-users

Complex Chaos

Connecting the Dots

```
"method": [
        {
            "type": "action",
            "name": "stop-provider-service",
            "provider": {
                "type": "python",
                "module": "chaosk8s.actions",
                "func": "kill_microservice",
                "arguments": {
                    "name": "my-provider-service"
                }           },
            "pauses": {
                "after": 10
            }       },
        {
            "type": "probe",
            "name": "consumer-service-must-still-respond",
            "provider": {
                "type": "http",
                "url":
"http://192.168.42.58:31018/invokeConsumedService"
            }       }    ],
```

BUILDING CUSTOM EXPERIMENTAL STEPS

`method` stanza defines our experiment, but you'll periodically include **`probes`** here which provide additional insight. These may later be promoted to the **`steady-state-hypothesis`**.

CHAOS SWISS ARMY TOOLKIT

You may have noticed that there are multiple ways to do similar experiments in chaos engineering.

Multiple tools & different patterns and that doesn't even consider the custom environments, applications, people, process, culture, and tools that make up any medium to large organization.

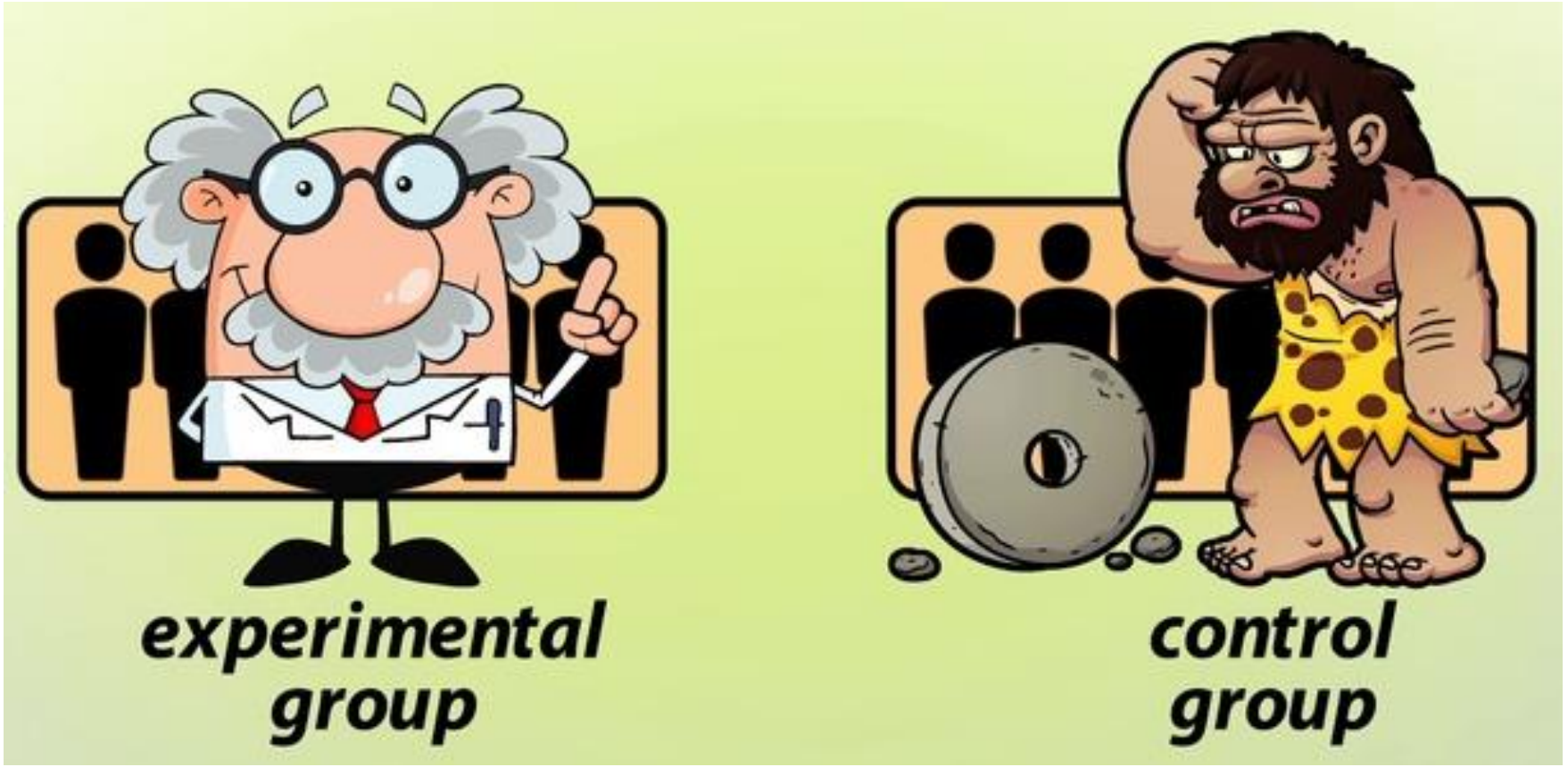
CHAOS ENGINEERING PATTERNS

In previous slides, in this section, we outlined killing a service with the Chaos Toolkit extension for Kubernetes, however, we outlined the same thing with the Spring Extension for Chaos earlier in the class.

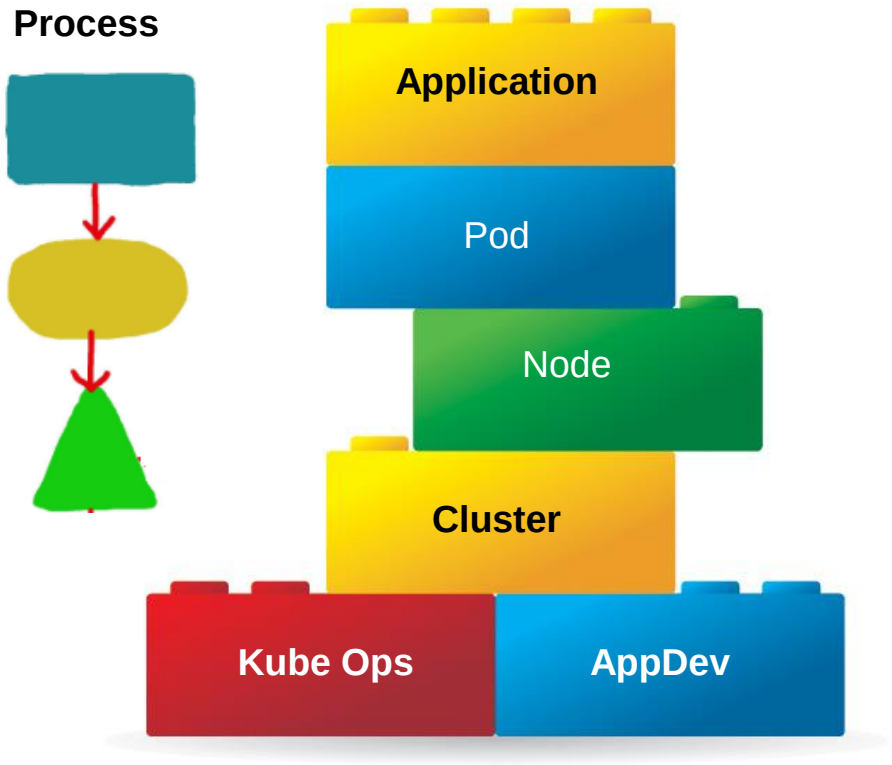
CHAOS ENGINEERING EVOLUTION

Chaos Engineering requires Continuous Learning and not everyone is going to be excited about jumping on that hamster wheel.

DEFINING CROSS-LEVEL HYPOTHESES AND EXPERIMENTS



BUILDING AND EXECUTING CROSS-LEVEL CHAOS ENGINEERING EXPERIMENTS



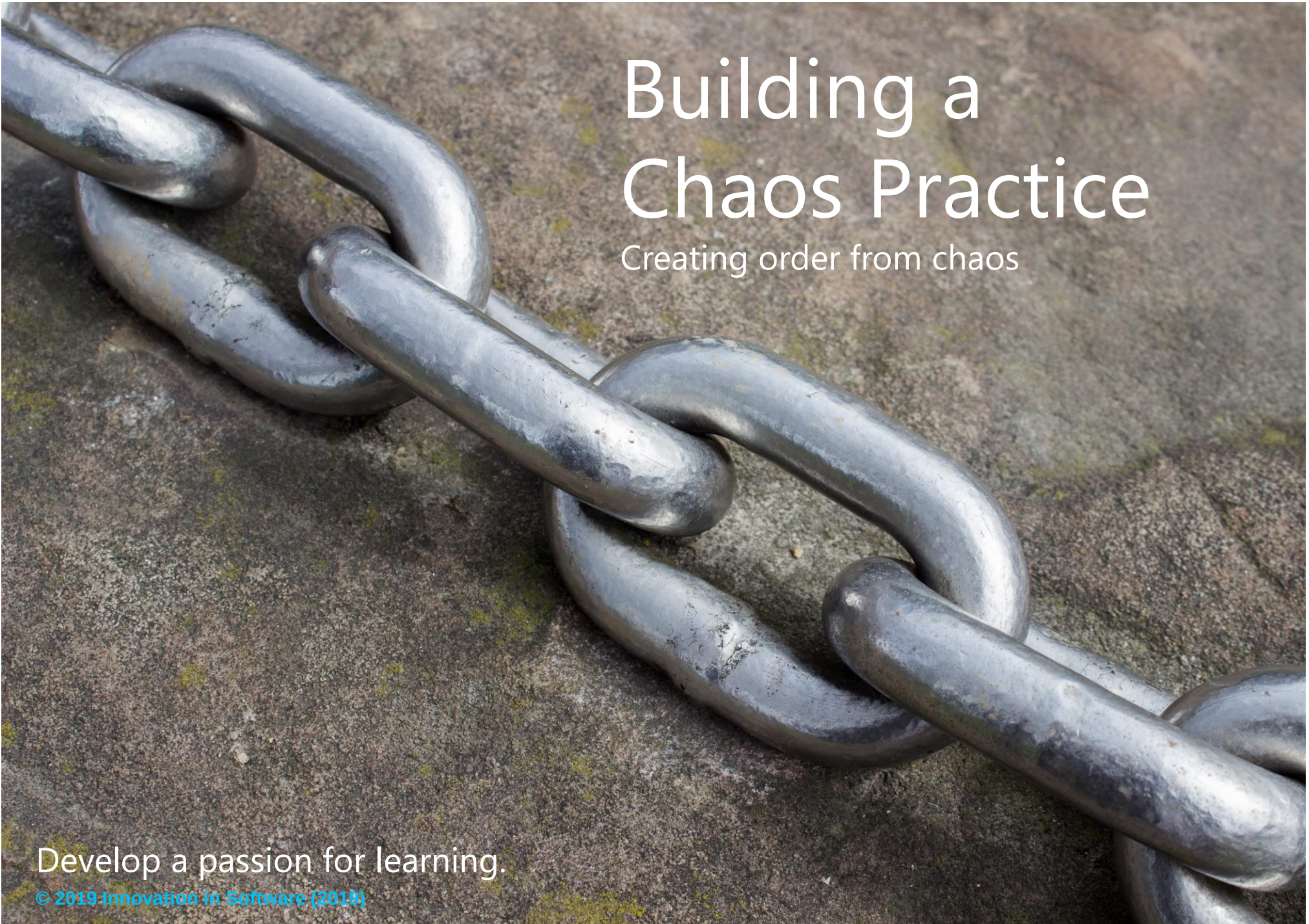
Coordination is required across teams and Chaos Engineering is by definition an extension of DevOps. Like DevOps, CE requires cultural change in many organizations to be fully adopted, especially in the notion of running experiments in production.

POP QUIZ: PARENTS AND CHILDREN

Can a chaos engineering block, have more than one parent?

Can a chaos engineering have more than one child?

10 MINUTES



Building a Chaos Practice

Creating order from chaos

Develop a passion for learning.

CHAOS ENGINEERING LEADERSHIP - PERSISTENCE

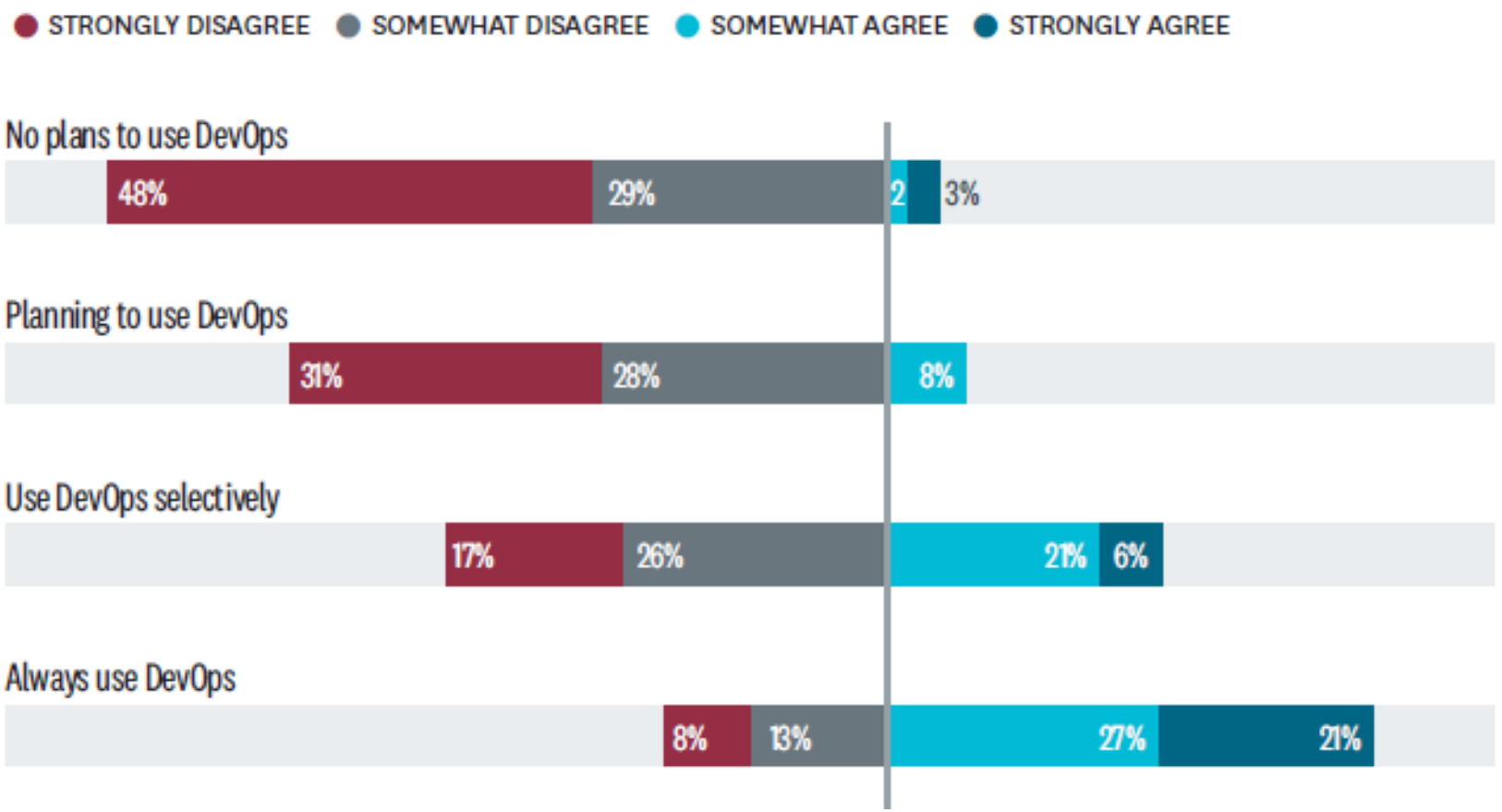
Effective leaders create strong alignment among SREs and let them figure out the best way to tackle problems in their own domains.

In the case of impacts, we need to create strong alignment to build components that are resilient to unexpected impact and work coherently end-to-end.

DEVOPS CHAOS THEORY

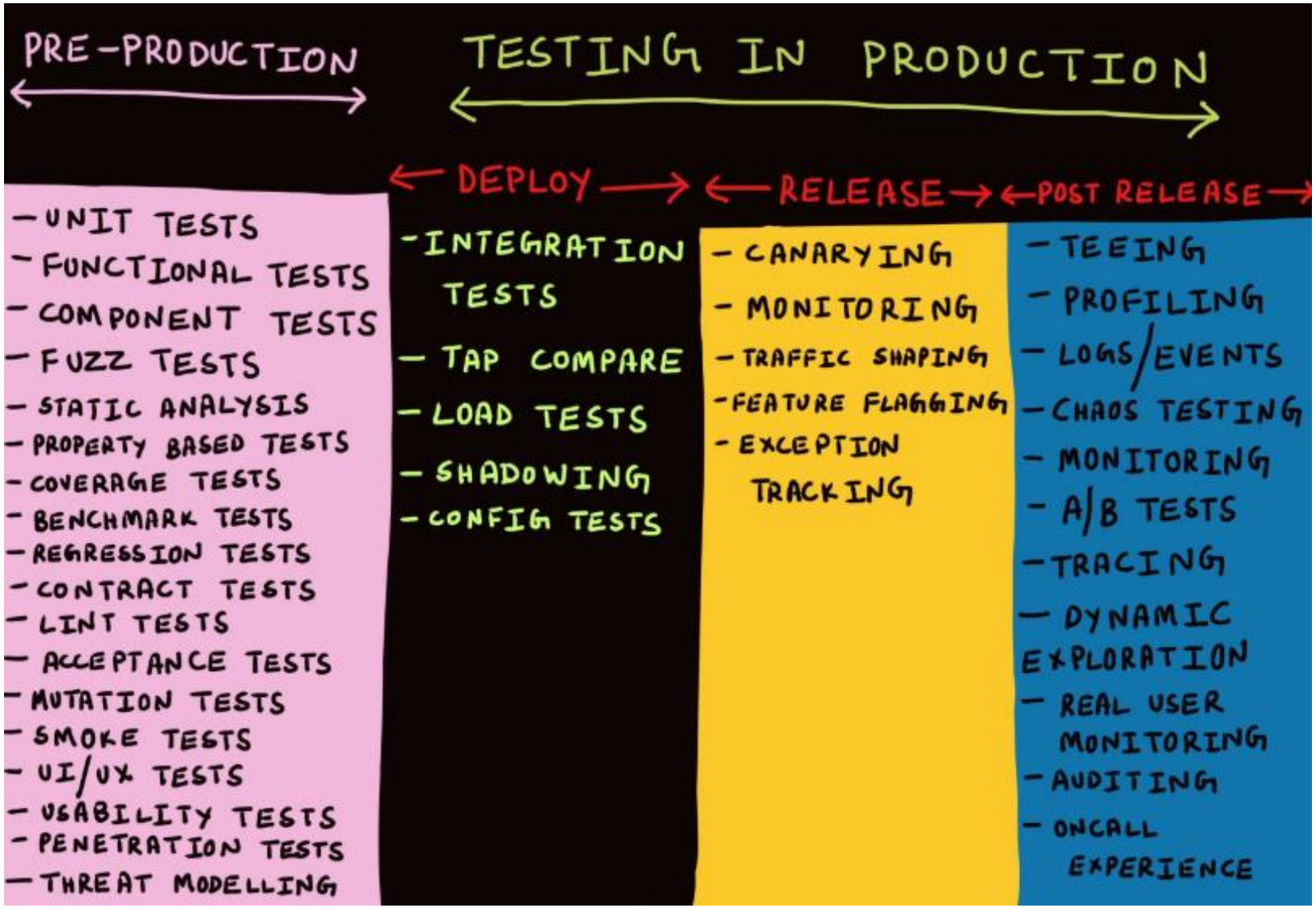
$$C = K * V * S^2$$

OUR BUSINESS LEADERS HAVE A GOOD UNDERSTANDING OF DEVOPS METHODS AND TERMINOLOGY?



Harvard Business Review, September 2018

CHAOS ENGINEERING IN CONTEXT



CHAOS ENGINEERING & APPDEV

Sample experiments for a generic three tier application deployed in two data centers.

This list is not comprehensive and depends on the application tooling support and maturity, but it is a good starting point.

- Terminate random virtual server in a region.
- Subject entire fleet to high CPU/Memory within one deployment location.
- Increase latency in one or more servers.
- Increase message latency
- Block access to a storage system.
- Failover a database to its secondary.
- Random killing of critical processes and/or services.

PROGRESSING TOWARDS GREATER CHAOS MATURITY

Chaos Maturity Model (CMM) gives us a way to map out the state of a chaos program within an organization.

The two metrics in the CMM are sophistication and adoption. Without sophistication, the experiments are dangerous, unreliable, and potentially invalid. Without adoption, the tooling will have no impact.

CHAOS COMPETENCE

The journey of a thousand miles begins with one step.

Lao Tzu

CHAOS MATURITY - INITIATE

- Experiments are not run in production.
- The process is administered manually.
- Results reflect system metrics, not business metrics.
- "What is this chaos engineering nonsense?"
- Simple events are applied to the experimental group, like "turn it off."
- "We don't test in production"
- What is this chaos engineering nonsense?"

CHAOS MATURITY - DEFINE

- Experiments are run with production-like traffic (shadowing, replay, etc.).
- Self-service setup, automatic execution, manual monitoring and termination of experiments.
- Results reflect aggregated business metrics.
- Expanded events like network latency are applied to experimental group.
- Results are manually curated and aggregated.
- Experiments are statically defined.
- Tooling supports historical comparison of experiment and control.

CHAOS MATURITY - MEASURE

- Experiments run in production.
- Setup, automatic result analysis, and manual termination are automated.
- Experimentation framework is integrated with continuous delivery.
- Business metrics are compared between experiment and control groups.
- Events like service-layer impacts and combination failures are applied to experimental group.
- Results are tracked over time.
- Tooling supports interactive comparison of experiment and control.

CHAOS MATURITY - OPTIMIZE

- Experiments run in each step of development and in every environment.
- Design, execution, and early termination are fully automated.
- Framework is integrated with A/B and other experimental systems to minimize noise.
- Events include things like changing usage patterns and response or state mutation.
- Experiments have dynamic scope and impact to find key inflection points.
- Revenue loss can be projected from experimental results.
- Capacity forecasting can be performed from experimental analysis.
- Experimental results differentiate service criticality.

POP QUIZ: BLOCK HASH

Is there is a practical reason that the hash of the current block is not contained in the header?

5 MINUTES

Lab 6- Chaos with Chaos IQ

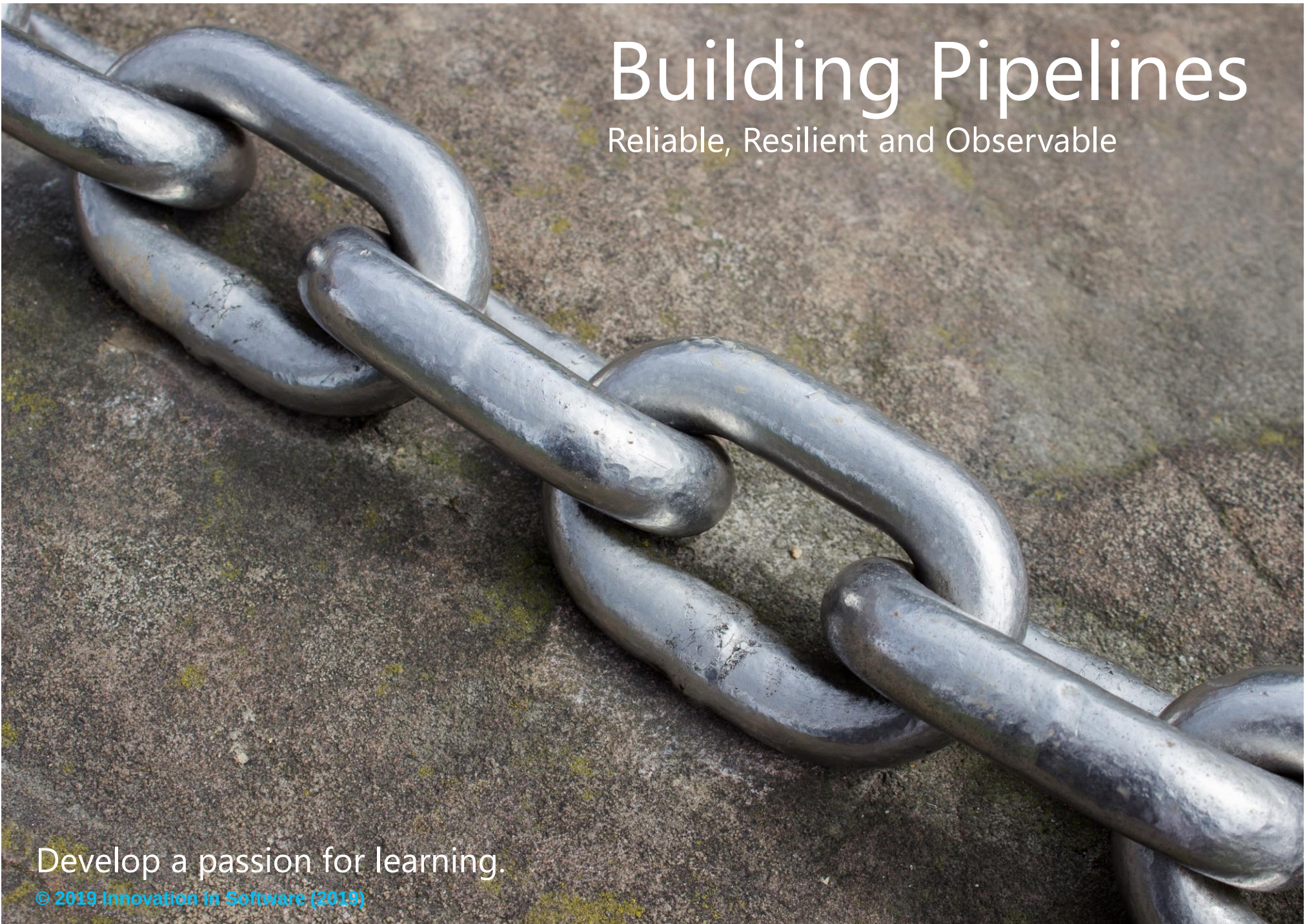
Lab 6- Chaos with Chaos IQ

ChaosIQ offer a number of **commercial tools** and services to **support successful chaos** engineering.

In this tutorial you can explore the various features they offer that extend and are integrated with by the Chaos Toolkit including:

Using ChaosIQ to bootstrap useful experiments even faster using the toolkit's **chaos discover** command.

https://katacoda.com/chaostoolkit/courses/03-commercial-integrations/chaosiq

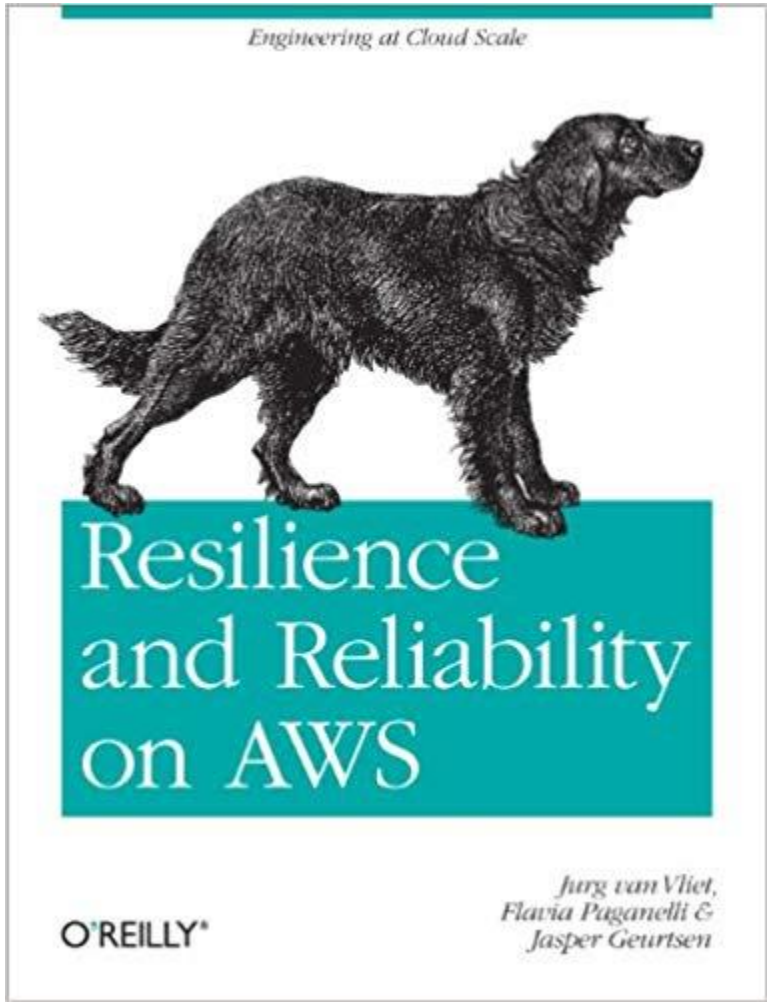


Building Pipelines

Reliable, Resilient and Observable

Develop a passion for learning.

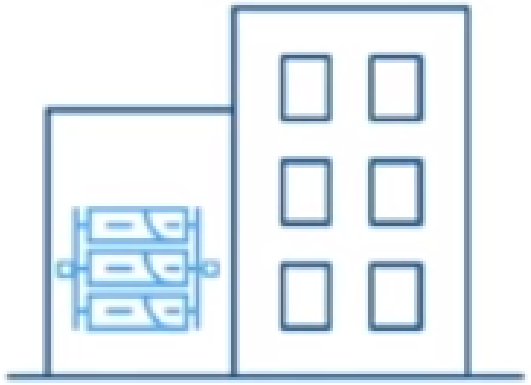
LEARNING, RESILIENCE AND RELIABILITY



To practice resiliency you have to consider that systems are constantly changing that means we're always chasing a moving target. To follow that means we have to enable knowledge sharing and continuous learning in our teams.

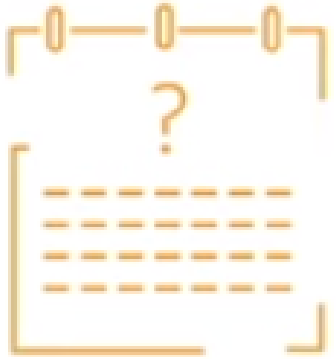
Resilience

Past — Present — Future



Disaster recovery

Chaos engineering

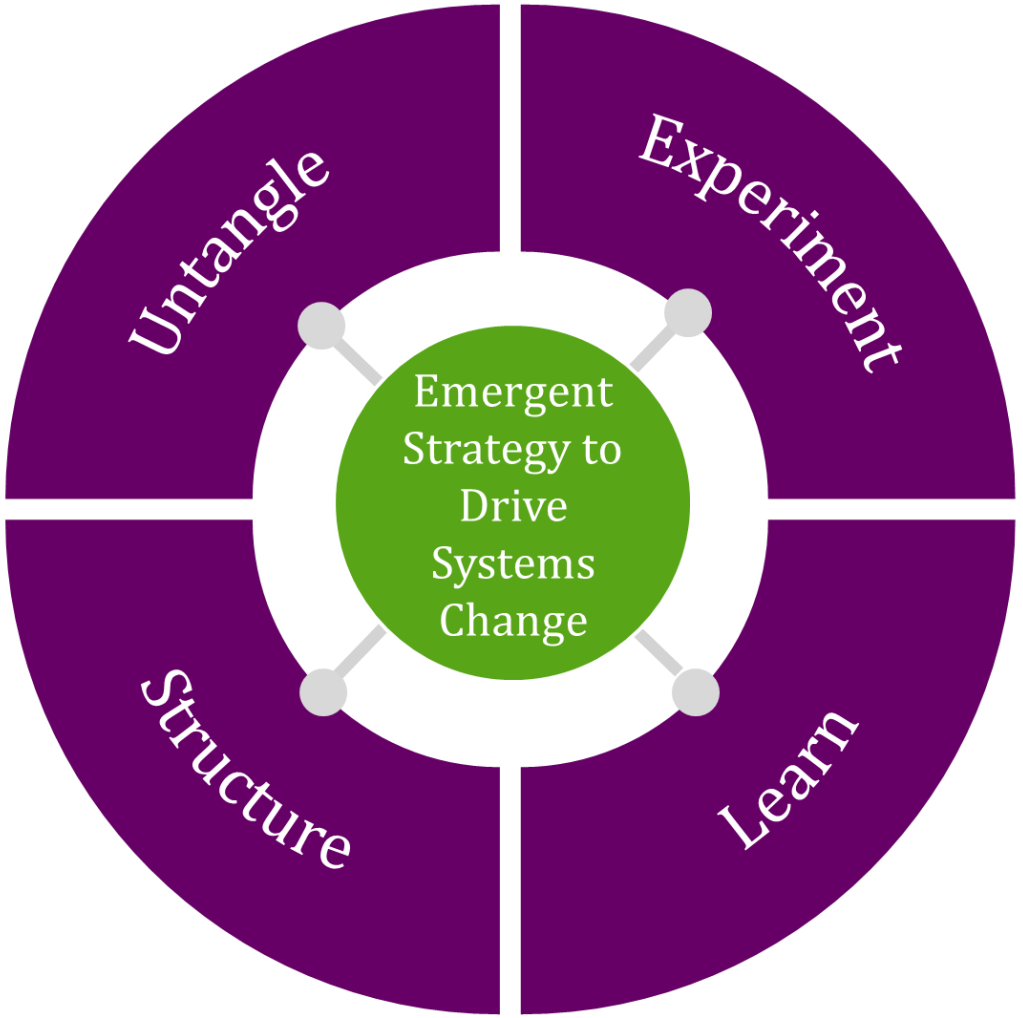


Resilient critical systems

ANTIFRAGILE SOFTWARE

Creating great software that embraces change to build the right thing, right, requires you to apply techniques that help you discover what is the best route towards the goal of useful software, and to create structures in your software that do not unnecessarily block your routes to that goal by being able to adapt to the ubiquitous and accelerating pace of change.

EMERGENT SYSTEMS



SOFTWARE DEFINED DELIVERY GOALS

We recognize that delivering useful software shapes our world. We recognize that code is the best way to specify precise action. We recognize that code is only useful when we deliver it.

Delivery is not a detail, it is our job. Now is the time to apply our core skills to engineer our delivery. We divide our work between ourselves and computers: humans for decisions, and automation for tasks.

DOING ONE THING AT ONCE - MINDFULNESS

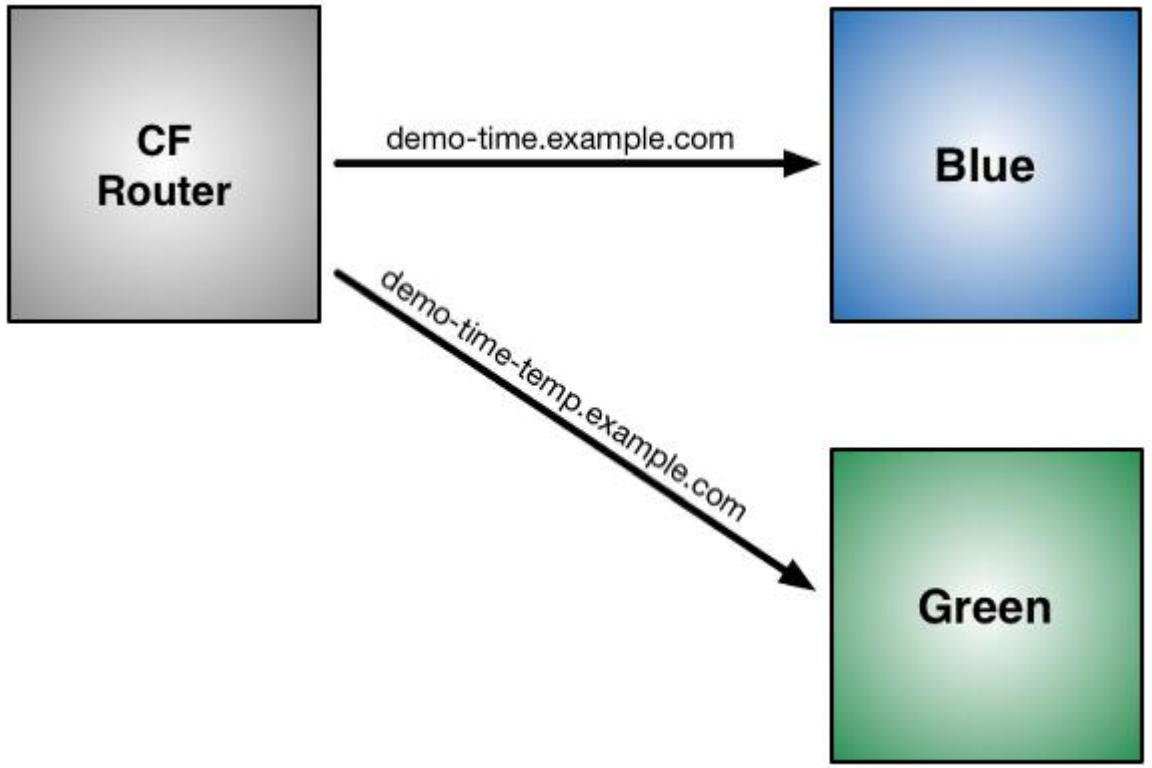
DELIVER WITH CONFIDENCE

Remove distracting behaviors in your software delivery, such as moving too fast, unnecessary movements, process fillers, and lack of value stream Secure-SDLC delivery.

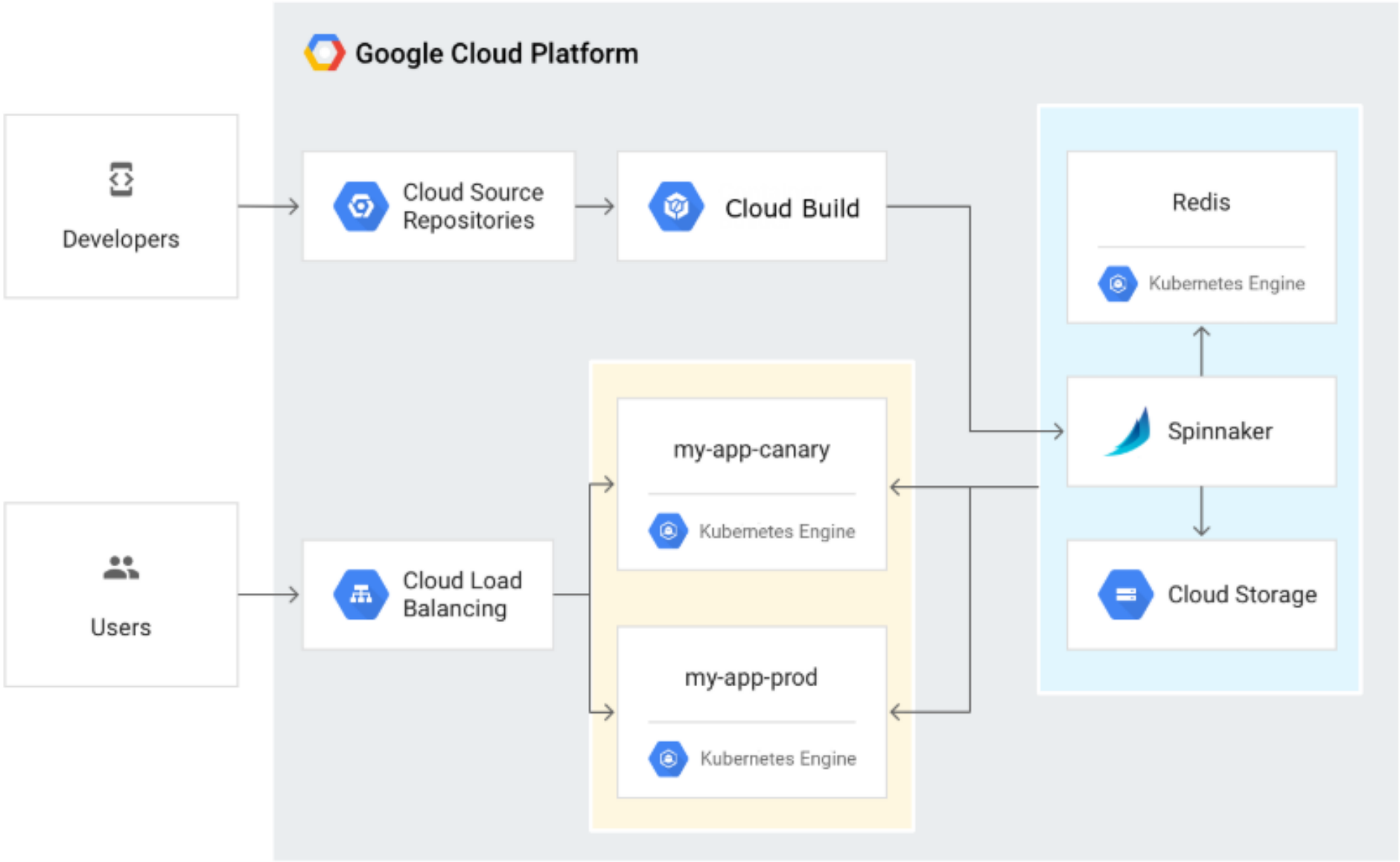
Reduce distractions in your DevOps focus.

Discover and fine-tune the style of Delivery based on your skills and capabilities.

GREEN/BLUE



CANARY



HAVING A BIG RED BUTTON

A key consideration in performing any chaos-engineering experimentation is the necessity for strong safety mechanisms.

Patrick Higgins

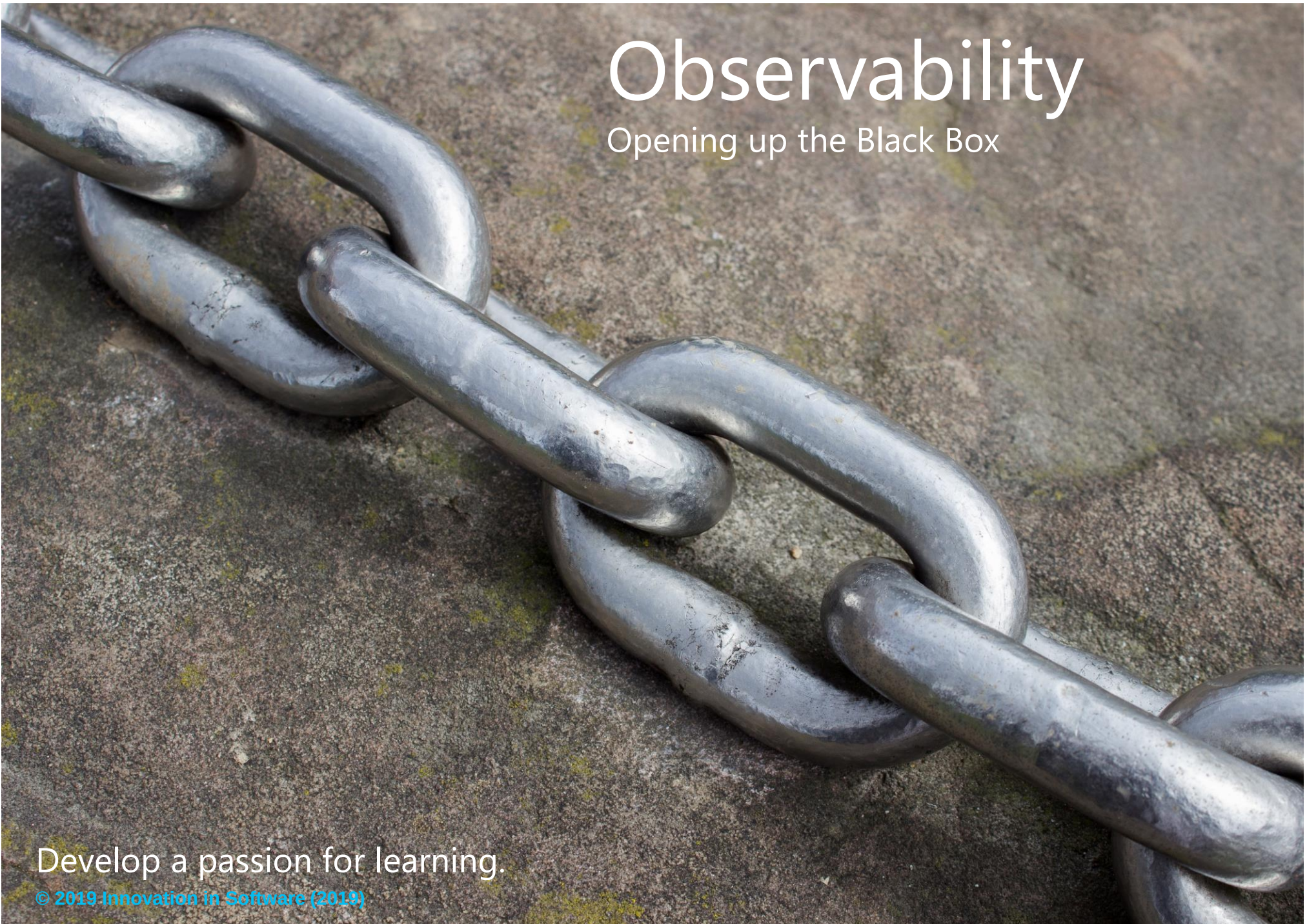
Software Engineer at Gremlin

POP QUIZ: BLOCK HASH

Is there is a practical reason that the hash of the current block is not contained in the header?

5 MINUTES

Lab 7- Brainstorming experiments

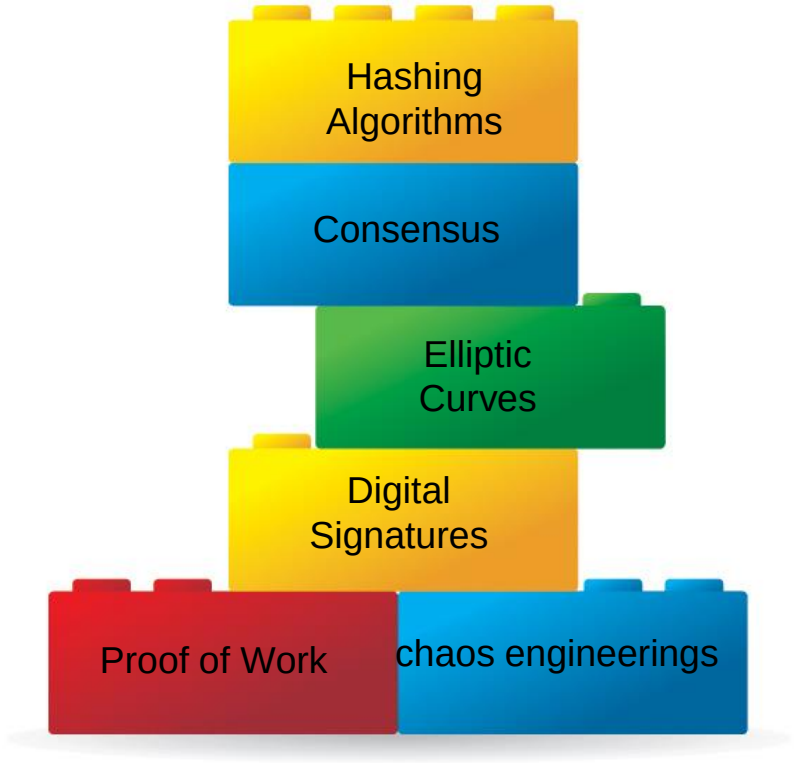


Observability

Opening up the Black Box

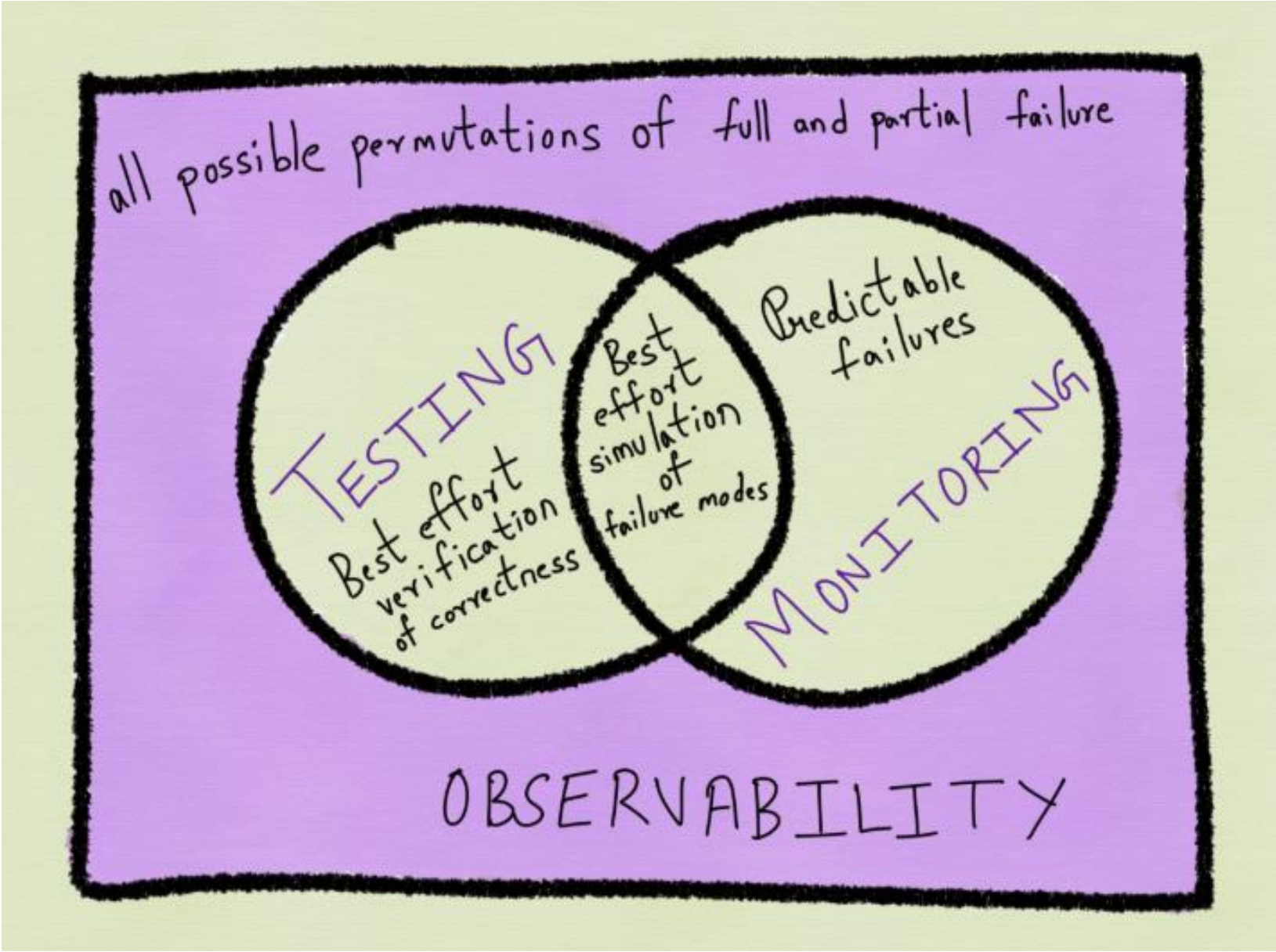
Develop a passion for learning.

DEFINING OBSERVABILITY



There are a variety of concepts, or building blocks, that are required to implement a chaos engineering. This module covers those topics, such as hashing algorithms, digital signatures, and more.

DEFINING OBSERVABILITY



WHY A SYSTEM NEEDS OBSERVABILITY

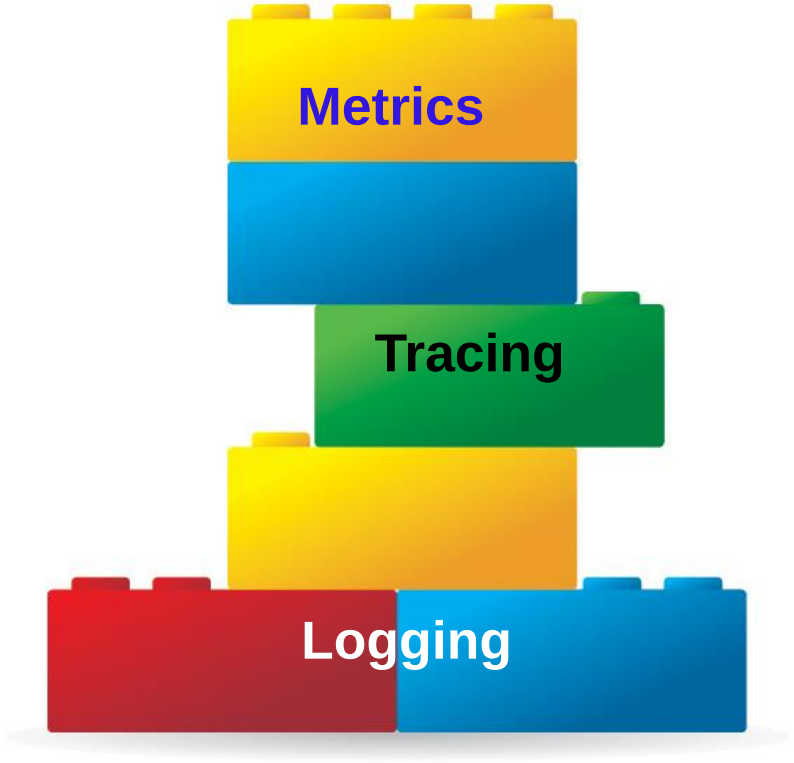
Observability falls into 5 categories

- Infrastructure Monitoring
- Log Analysis
- Distributed Tracing
- Application Performance Monitoring (APM)
- Real User Monitoring

WHO NEEDS OBSERVABILITY?

Aggregate visibility into our systems is **paramount** to gaining **understanding** of those systems and how they change over time. They are the **canary in the coal mine**. Observability is a combination of people, process and tools.

THE THREE PILLARS OF OBSERVABILITY



You need a plan to collect data from many sides and that can be interpreted by all of your dev team.

This way comprehensive troubleshooting, and solutions can be performed and created from developer and operations teams.

THE FOUR GOLDEN SIGNALS OF MONITORING

Latency

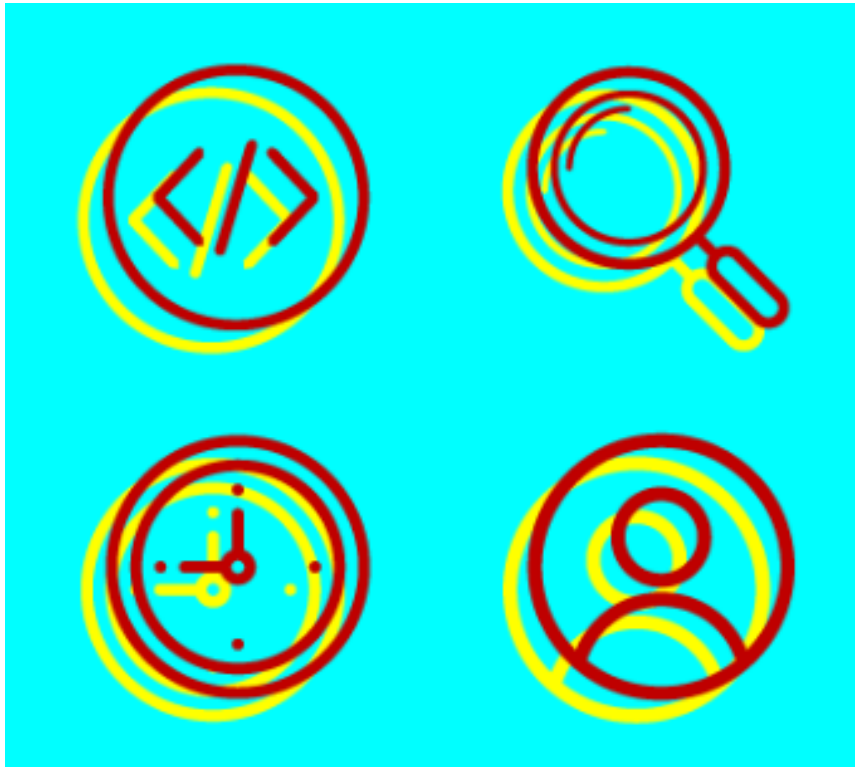
Traffic

Errors

Saturation

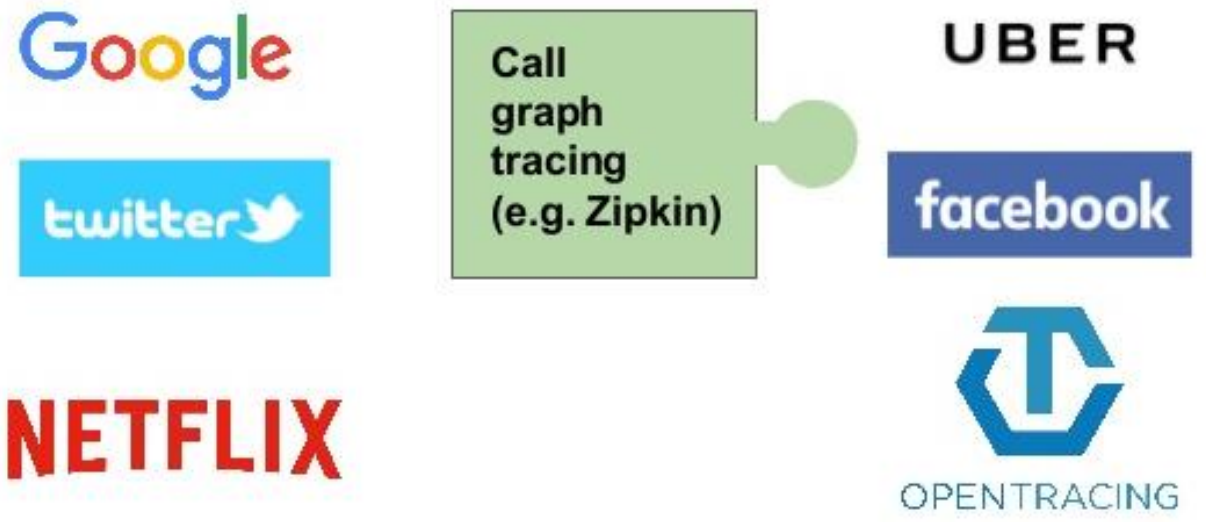
Google's site reliability engineers (SREs) defined four key metrics to monitor.

THE EXTENTS OF OBSERVABILITY



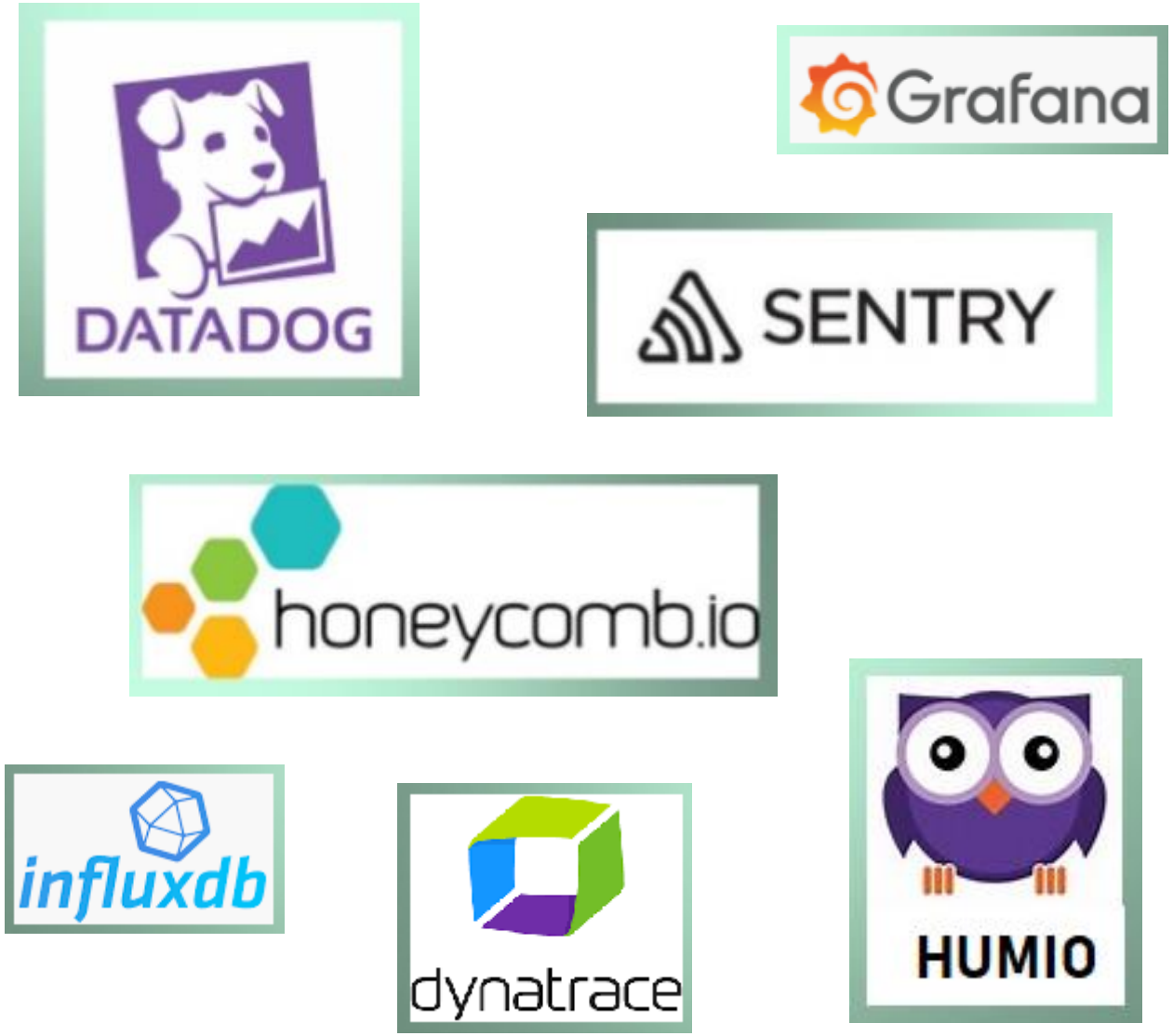
Different kinds of applications, tools, capabilities, and processes let you explore data to different extents.

APPLYING OBSERVABILITY TO YOUR OWN SYSTEM



What the heck is going on?

OBSERVABILITY TOOLS



There are many tools that can be used for observability, many start as open source projects that convert to tiered pay models or enterprise supported subscriptions.

POP QUIZ: BLOCK HASH

Is there is a practical reason that the hash of the current block is not contained in the header?

5 MINUTES

Lab 8- Chaos with Gremlin

Lab 8- Chaos with Gremlin

This tutorial will use Gremlin to experiment on a microservice-based system using Kubernetes and then show how you can explore that system using the Gremlin Chaos Engineering tooling to discover and overcome a weakness in the form of a cascading failure.

https://katacoda.com/chaostoolkit/courses/02-chaos-engineering-experiments-with-chaostoolkit/service-down-not-visible-to-users

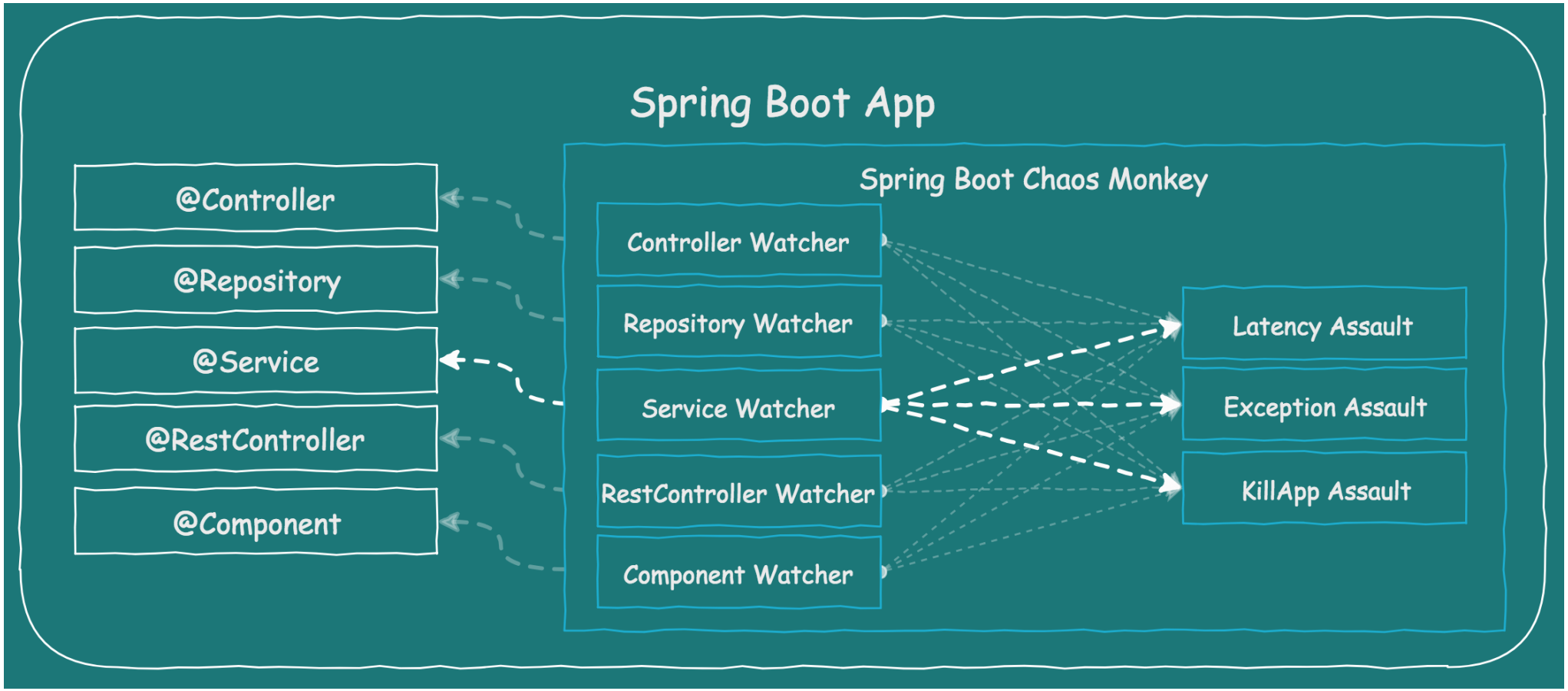
Chaos AppDev

Develop a passion for learning.

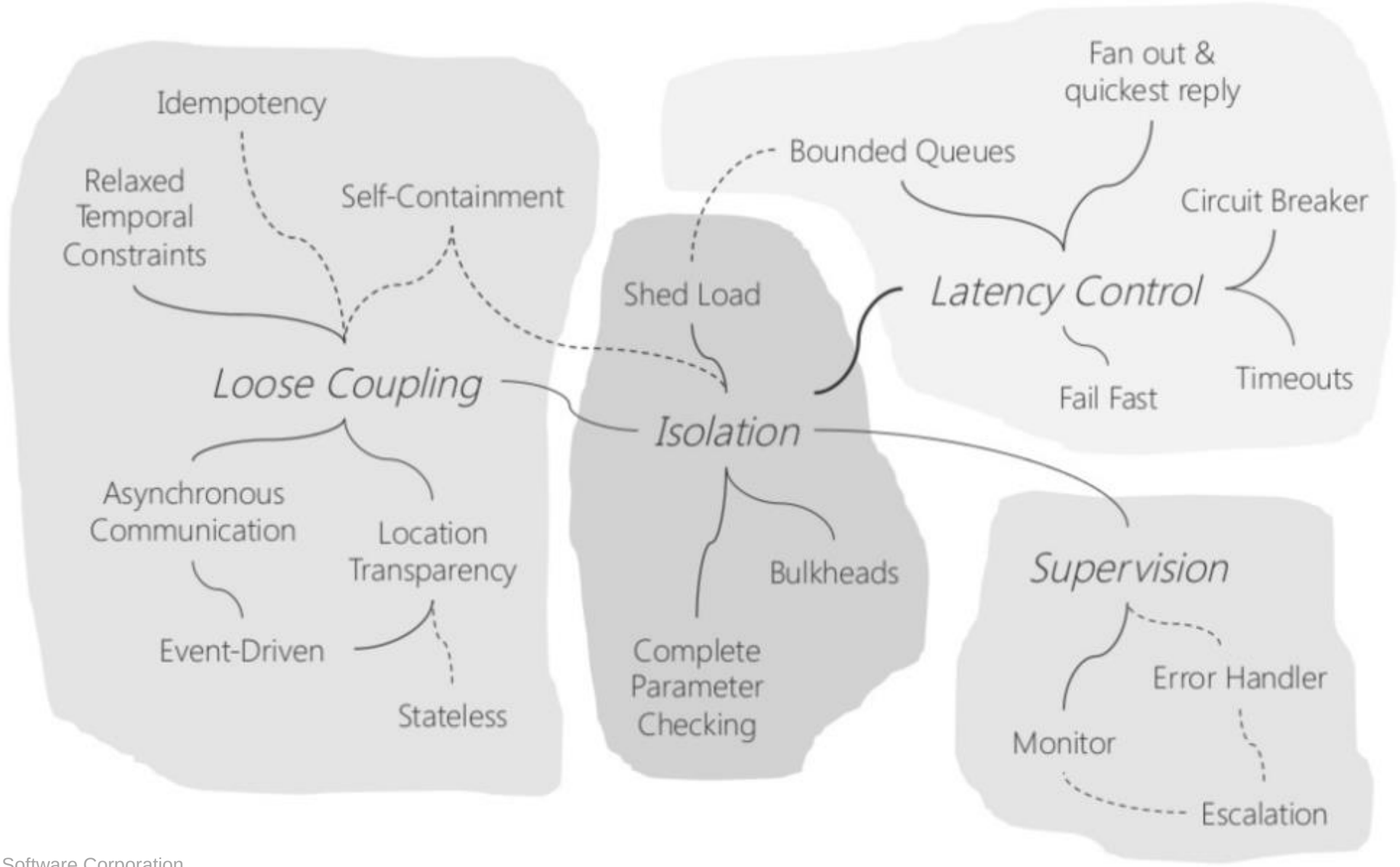
HOW TO SUCCESSFULLY APPLY CODING FOR FAILURE TO A PRODUCTION SYSTEM

> *A complex system that works is invariably found to have evolved from a simple system that worked.*
>
> Gall's Law

WORKING WITH SPRING EXTENSION FOR CHAOS



PATTERNS OF RESILIENCE



IDENTIFYING API INTERACTION FOR RESILIENCY UNDER LOAD

DEVELOPERS

will access your assets through your API to build **Mobile Apps** and **Web Apps** based on the data and software you share.

THE API

provides universal access to whatever assets you choose to share. Developers can **"plug in"** their apps and data.

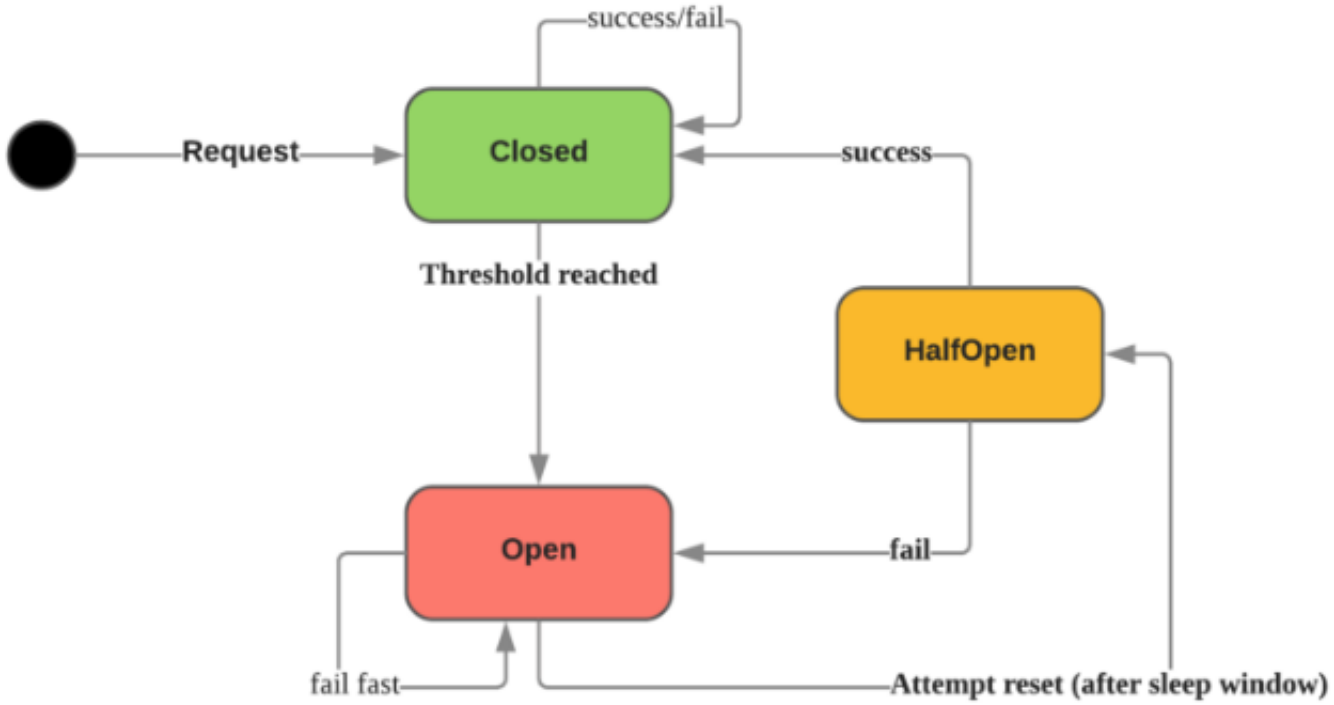
ASSETS

Your data and software (and brand) become more valuable by being leveraged by partners, developers, and third-party services.

END USERS

have access to apps that provide richer experiences by leveraging the data and services of other apps.

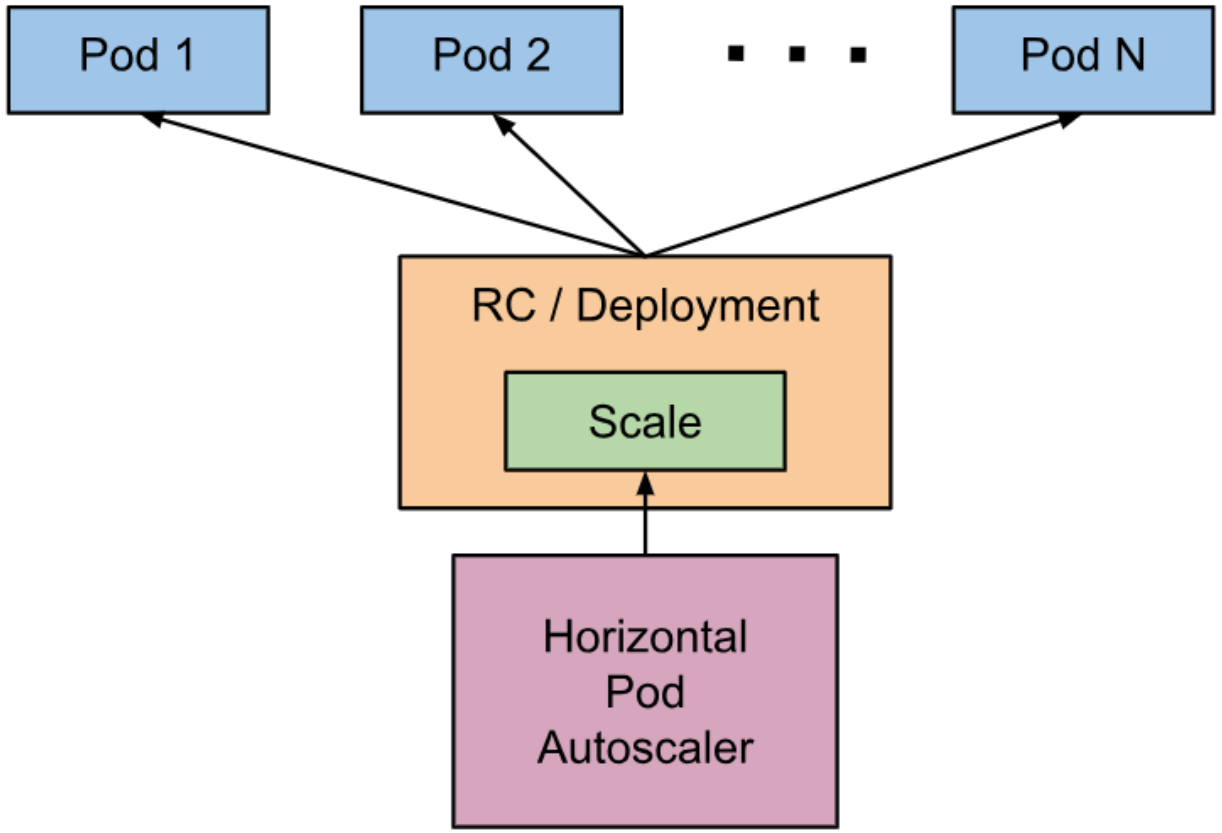
WORKING WITH CHAOS TO PROGRAMMATICALLY CIRCUIT BREAK



A natural complement to microservices and distributed architectures, circuit breakers exist to prevent cascading failures in systems.

Cascading failures not only directly impact the customer, they consume increasing resources across the system.

HORIZONTAL POD AUTOSCALER



This scaling is based on observed CPU utilization or the custom metrics API

DRIVING SCALE WITH CUSTOM METRICS

```
metrics:
 - type: Resource
   resource:
     name: cpu
     target:
       type: AverageUtilization
       averageUtilization: 50
 - type: Pods
   pods:
     metric:
       name: packets-per-second
     targetAverageValue: 1k
 - type: Object
   object:
     metric:
       name: requests-per-second
     describedObject:
       apiVersion: extensions/v1beta1
       kind: Ingress
       name: main-route
     target:
       kind: Value
       value: 10k
```

With the noted configuration in your hpa.yaml file, HorizontalPodAutoscaler would attempt to ensure that each pod was consuming roughly 50% of its requested CPU, serving 1000 packets per second, and that all pods behind the main-route Ingress were serving a total of 10000 requests per second.

EFFECTIVE PRE-MORTEM LEARNING: GAME DAY

An exercise where we place our systems: technology + people + processes, under stress in order to learn and improve resilience

POP QUIZ: TRANSACTION VERIFICATION

What is the purpose of this rule:

For each input, if the referenced output exists in any other transaction in the pool, the transaction must be rejected.

5 MINUTES

Chaos Engineering Architecture

Develop a passion for learning.

Chaos Engineering Team

People

Application

Switching

Infrastructure

Infrastructure and services

No single point of failure

Switching and interconnecting

Data replication

Traffic routing

Avoiding issues

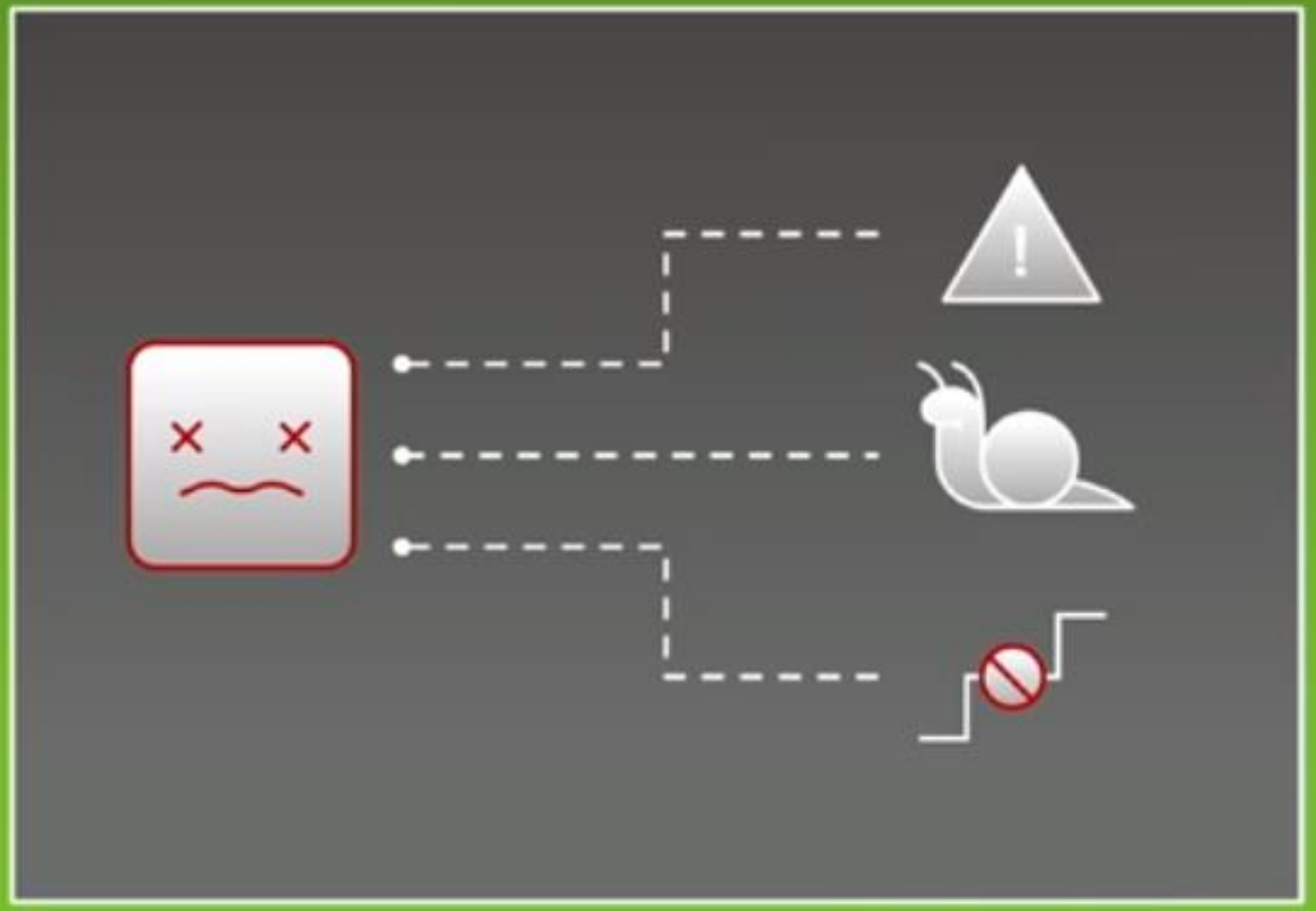
Anti-entropy recovery

Application failures

Error returns

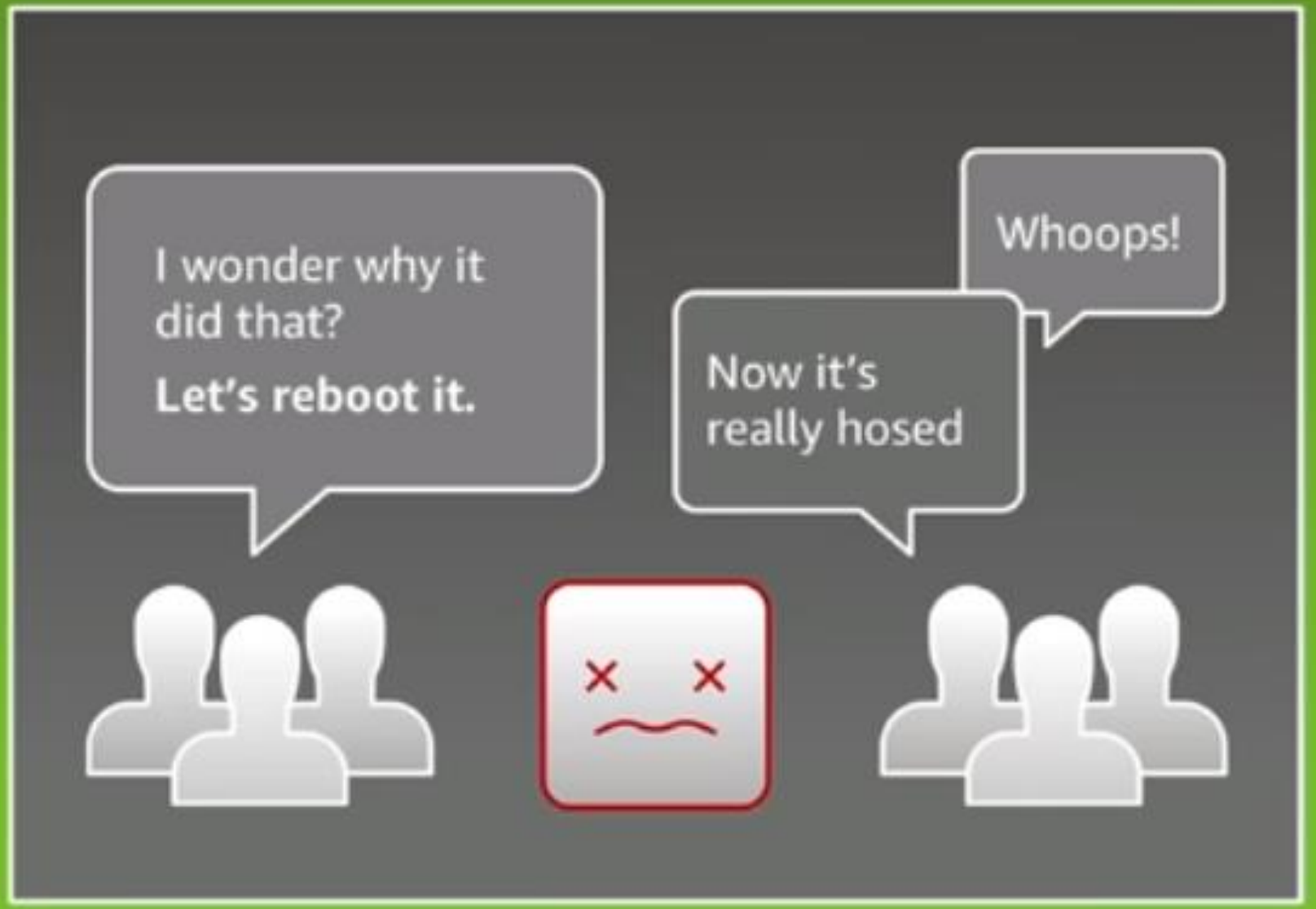
Slow response

Network partition



People

Unexpected application behavior often causes people to intervene and make the situation worse



Chaos Engineering Best Practices

Develop a passion for learning.

START CHAOS IN LOWER ENVIRONMENTS

Yes, you must eventually test in your production system, but it makes sense to start in test, staging or development environments and work your way up. Once you've hit 100% of your application in your lower environment, you reset to the smallest bit possible in production, and take it from there.

KILL -9 CHAOS_EXPERIMENT

Make sure you have a way to **stop all CE experiments immediately**, on the spot, and **return** all systems to their **steady state**.

MINIMIZE THE BLAST RADIUS

Start with the smallest chaos experiment you can perform that will teach you something about your system. See what happens. Then **increase the scope as you learn** and as your confidence grows.

AUTOMATE WITH A PIPELINE

Once you understand a particular kind of failure and you've tested it, you want to **automate** the testing of it in a **continuous deployment** (CD) pipeline so you maintain that competence.

DON'T EXPERIMENT GUARANTEED FAILURES

Never conduct a chaos **experiment** in any environment that you already know will cause **severe damage**, affect customers, or damage your CE reputation.

UNDERSTAND THE EVOLUTION OF CHAOS

Chaos Engineering, Security Chaos Engineering, Fault Injection Testing, Resiliency Engineering, Site Reliability Engineering were almost all **failures initially** (depending on who you ask, of course). Only **time and experience** make an **emerging pattern really useful**, in most organizations.

WHICH BRINGS US TO OUR NEXT TO LAST BEST PRACTICE!

DON'T BE A CHAOS MONKEY

Chaos Monkey was Netflix's famous—or **infamous**—tool that randomly rebooted servers. Unfortunately, today many people believe that chaos engineering means **randomly breaking things**. The idea behind chaos engineering is to **perform thoughtful, planned, and scientific experiments** instead of simply random failure testing.

Common language, terminology, and definitions help mitigate **communication failure** between people working on resiliency.

I'm proposing some terminology, try to use common definitions rather than making up your own!

Reference

Develop a passion for learning.

BOOK: CHAOS ENGINEERING

https://www.oreilly.com/library/view/chaos-engineering/9781491988459/

Written by folks from Netflix who have migrated to many other organizations over the last 5 years.

Best of all this book is completely free.

BOOK: ANTIFRAGILE SOFTWARE

https://leanpub.com/antifragilesoftware

Written by Russ Miles with Sylvain Hellegoarch, and Grant Tarrant-Fisher.

These are the folks who have driven a boatload of the progress in Chaos Engineering and the Chaos Toolkit in the last two years.

SITE: CHAOS TOOLKIT

https://chaostoolkit.org/

The Chaos Toolkit aims to be the simplest and easiest way to explore building, and automating, your own Chaos Engineering Experiments.

SLACK: CHAOS ENGINEERING WORKSPACE

https://chaosengineering.slack.com

Slack workspace managed by some of the names in this space including Tammy Butow, Ana Medina, Russ Miles, Sylvain Hellegoarch among others.

Great place to network, learn and share Chaos experiences.

GITUB: AWESOME CHAOS ENGINEERING

https://github.com/dastergon/awesome-chaos-engineering

A repository of a boatload of knowledge for Chaos Engineering.

Beware that some of this material is pretty dated and goes back to the days before people really were trying to do Chaos Engineering without creating chaos

SITE: MEDIUM.COM

https://medium.com/search?q=chaos%20engineering

Medium.com is an emerging website with comprehensive technical content for developers. The content is detailed and with plenty of examples. Best online resource for chaos engineering content.