



# **Chaos and feature experimentation for safer, faster deployments with AWS**



**Adrian Cockcroft**


VP Cloud Architecture Strategy  
AWS

@adriancockcroft 



**Yaniv Bossem**

Solutions Architect  
AWS

@yanivbossem 



**Adrian Cockcroft**

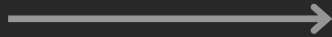
VP Cloud Architecture Strategy

AWS

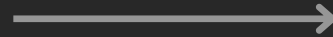
@adriancockcroft 

# Pathway for innovation

Speed



Scale



Strategic



Time to  
value



Distributed  
optimized  
capacity



Critical workloads  
datacenter  
replacement

You don't add innovation to an organization  
**You get out of its way!**

What is the fundamental  
metric for **innovation**?

# Time to value



Do some work



How long?



Value to a customer

# Time to value



Do some work



Months?



Value to a  
customer



# Time to value



Do some work



Days?



Value to a customer

# Time to value



Do some work



Minutes?



Value to a customer

There is no economy of scale in software

**Smaller changes are better**

# Lots of small changes



Automated  
continuous  
delivery pipeline



Tagging  
feature flags,  
A/B tests



Rapid  
cheap  
builds

Lots of small changes

HOURS

to

SECONDS

Move from **Java Monolith** to **Go Microservices**

SLOW  
build

BIG  
build

FAST  
build

SMALL  
build

Change one  
small thing  
at a time

Easier to



tell if it breaks

Easier to



roll back to  
previous version

Easier to



measure time  
to value

# De-couple

**New code** from **new feature**



Incrementally change  
system with many  
small safe updates

Turn on features for  
testing and when it  
works—for everyone



Less risk

Faster problem detection

Faster repair

Less work in progress

Less time merging changes

Happier developers

Faster flow



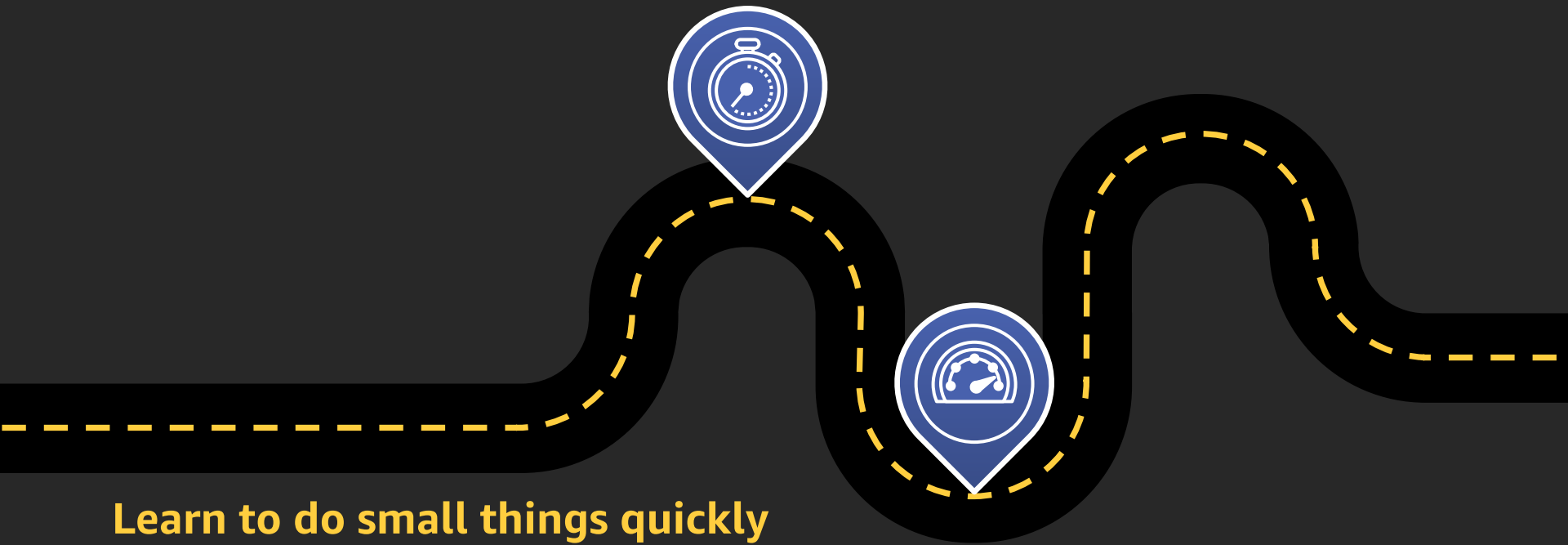
# How do we get there?

## Measure time to value everywhere

Automate collection and reporting of commit to deploy



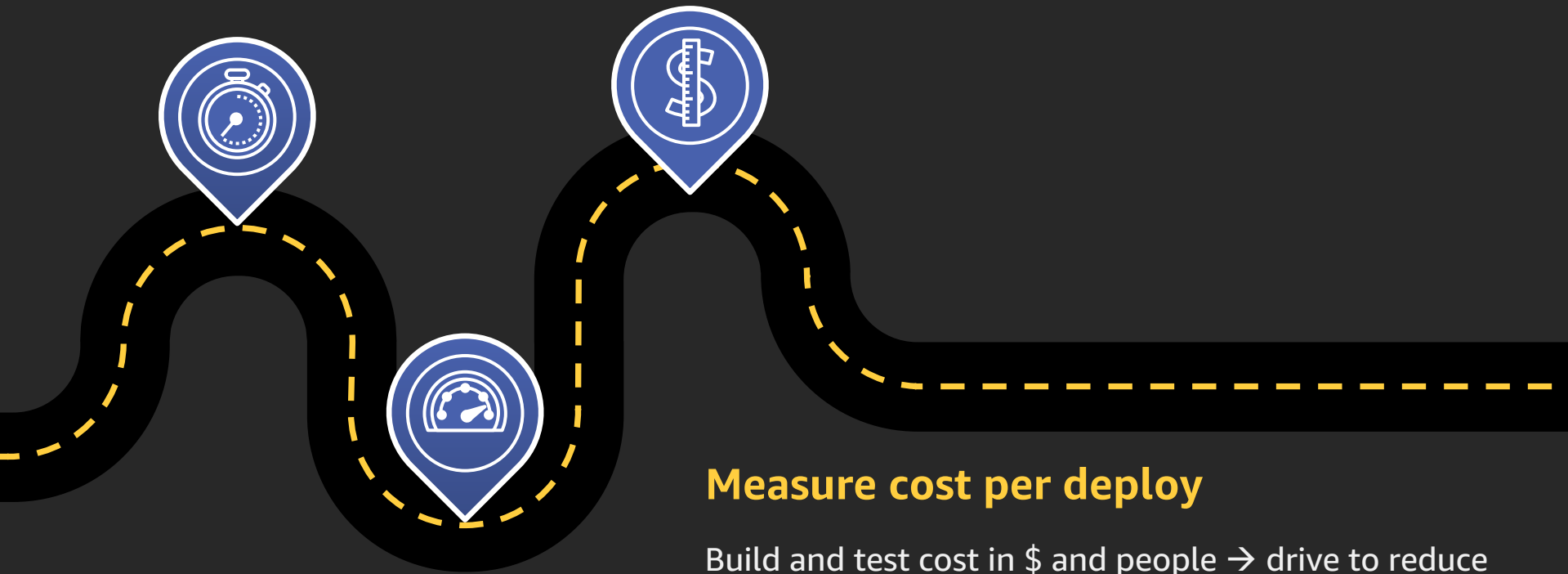
# How do we get there?



## **Learn to do small things quickly**

Don't get bogged down speeding up everything  
create a fast path for simple and safe changes

# How do we get there?



## Measure cost per deploy

Build and test cost in \$ and people → drive to reduce

Number of tickets filed per deploy → drive to one

Number of meetings per deploy → drive to zero

THE **LEAN** SERIES

ERIC RIES, SERIES EDITOR

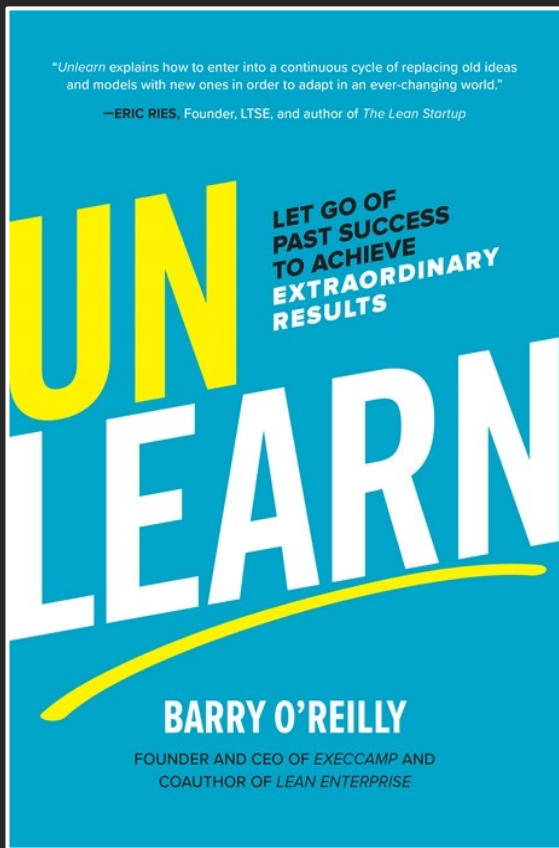
Jez Humble, Joanne Molesky & Barry O'Reilly

# LEAN ENTERPRISE

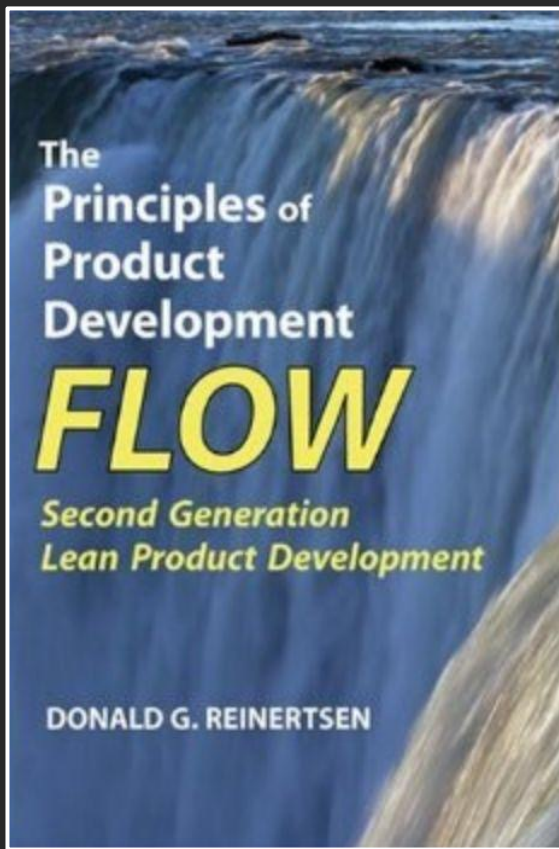
How High Performance  
Organizations  
Innovate at Scale

O'REILLY®

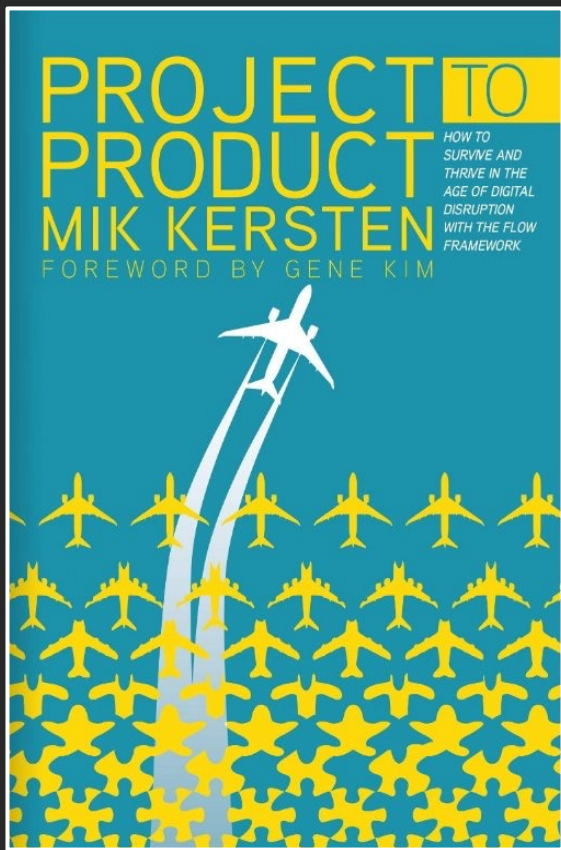
Hypothesis-  
driven  
**development**



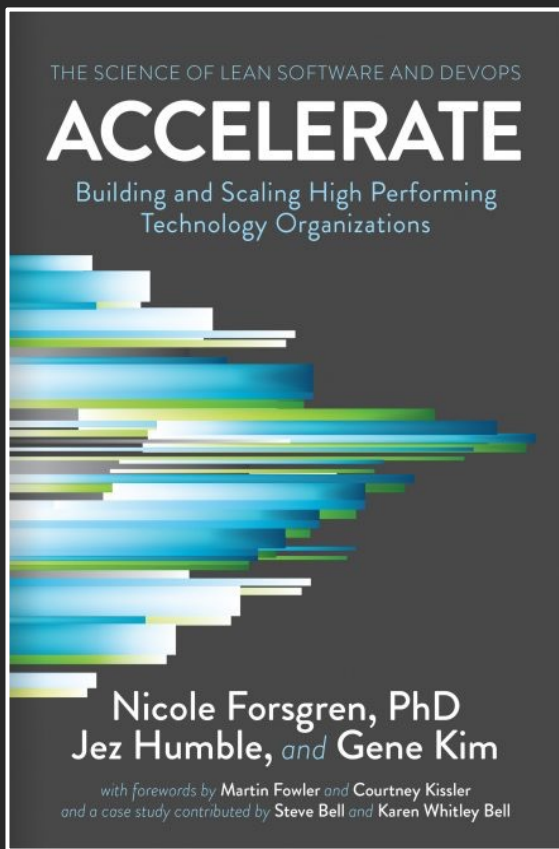
**Break away**  
from your old  
ways of working



Theoretical  
basis for using  
consistently  
**small changes**



Get rid of 90%  
of your project  
managers as  
you move to  
**continuous  
improvement**



Survey data  
showing that  
**low latency**  
time-to-value  
works





# Time to value



Learn to do simple things  
quickly to unblock innovation



Avoid complex  
one-size-fits-all processes



# Time to value



The best IT architecture today is:

Minimalist, messy and inconsistent

Provides guard rails for security, scalability and availability

**Designed to evolve rapidly**  
and explore new technologies

Supports low latency continuous delivery

# Pathway for innovation

Speed



Scale



Strategic



Time to  
value



Distributed  
optimized  
capacity



Critical workloads  
datacenter  
replacement



# Distributed optimized capacity

Highly scaled

Distributed for availability

Cost optimized high utilization

Cloud-native architecture



## Cloud-native principles

Pay as you go, afterwards

Self service—no waiting

Globally distributed by default

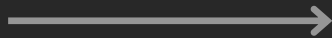
Cross-zone/region availability models

High utilization—turn idle resources off

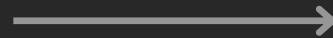
Immutable code deployments

# Pathway for innovation

Speed



Scale



Strategic



Time to  
value



Distributed  
optimized  
capacity

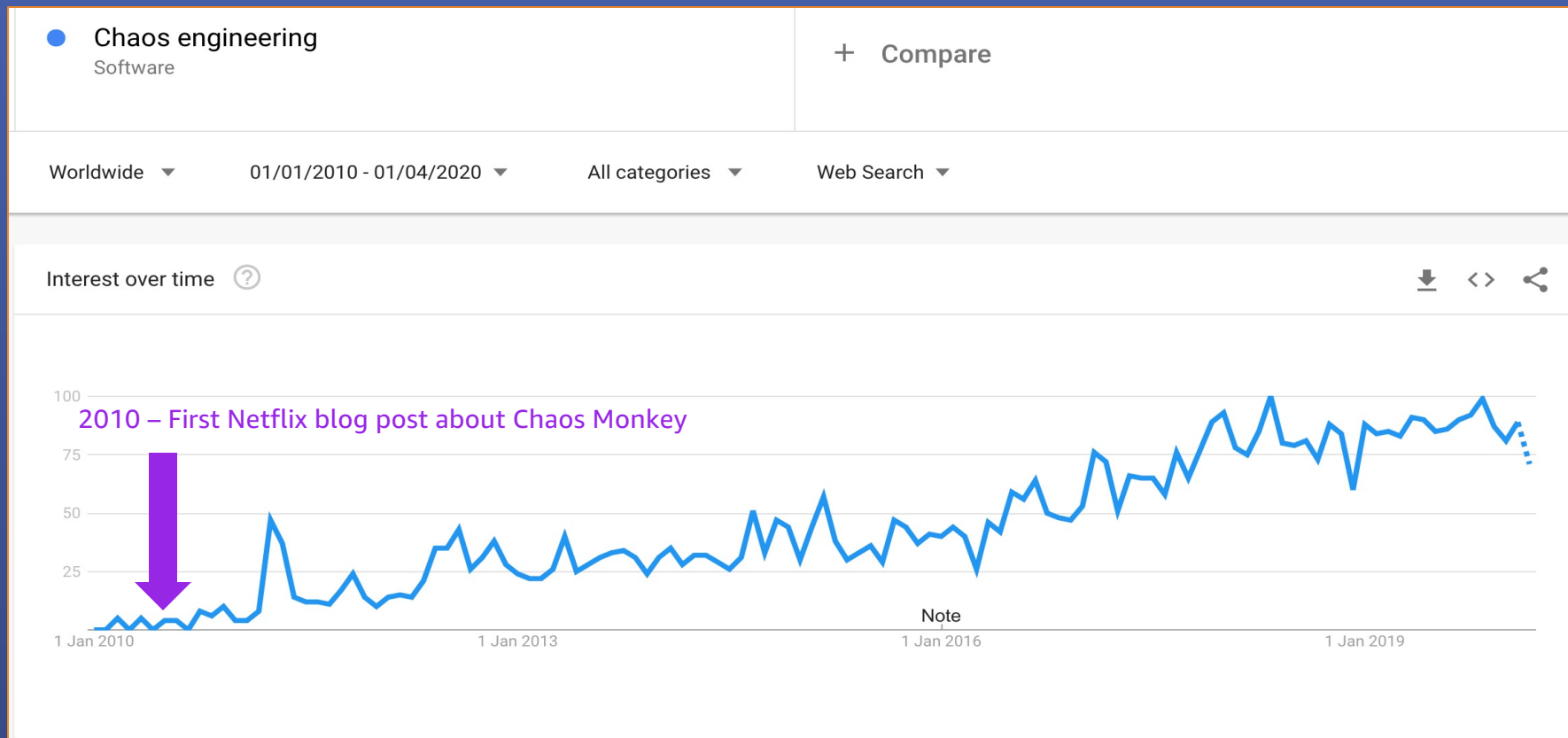


Critical workloads  
datacenter  
replacement

# Chaos engineering definition

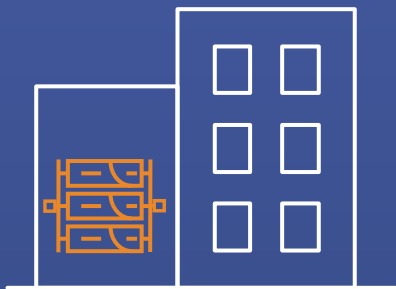
Experiment to ensure that the impact  
of failures is mitigated

# Trends – chaos engineering – 10 years on ...





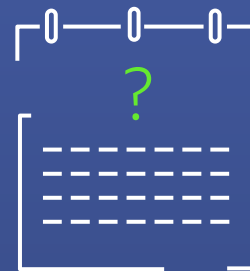
# Resilience



Disaster  
recovery



Chaos  
engineering

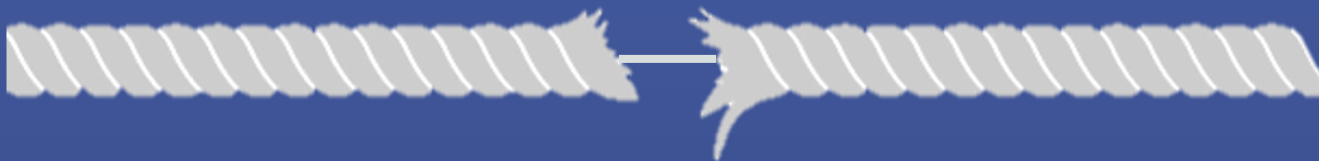


Continuous  
resilience

We build redundancy into systems so that if something fails, we can fail over to an alternative

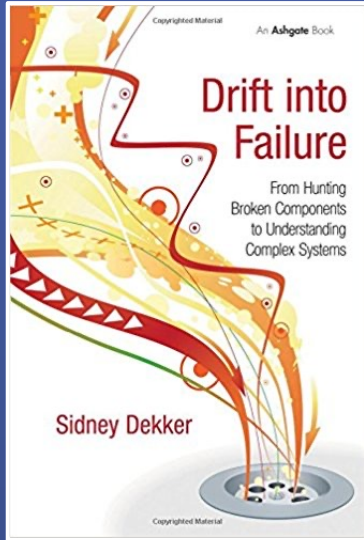
However, our ability to fail over is complex and hard to test, so often the **whole system falls over**

How can we do better?



The last strand that breaks is not the cause of a failure!

Build resilient systems like a rope, not a chain, but make sure you know how much margin you have and how “frayed” your system is



# Drift into Failure

**Sidney Dekker**

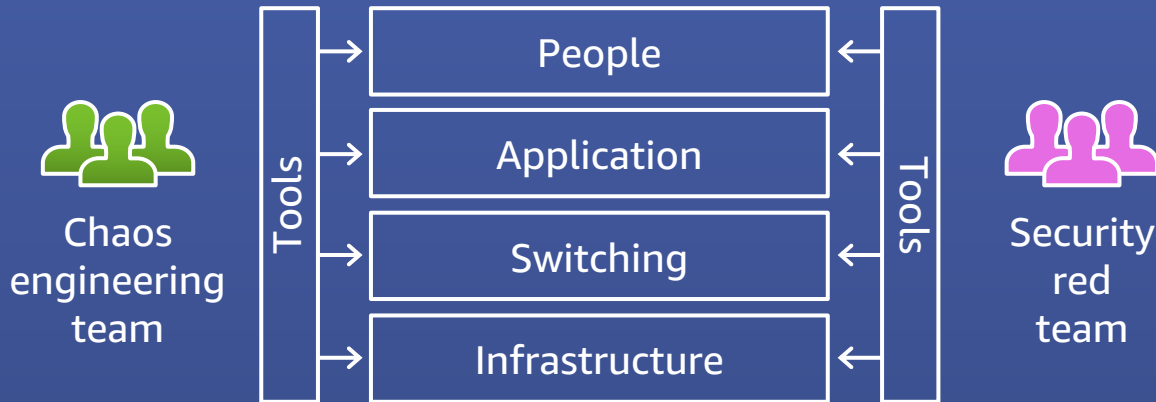
Everyone can locally optimize for the right outcome at every step, and you may still get a catastrophic failure as a result . . .

We need to capture and learn from near misses, test and measure the safety margins, before things go wrong

# Chaos architecture

Four layers, two teams, an attitude

**Experiment to ensure that the effect of failures is mitigated**



New! **AWS Fault Injection Simulator**

**Run experiments to ensure that both availability and performance impact of failures are mitigated**

**“You can’t legislate  
against failure; focus on  
fast detection and response.”**

—Chris Pinkham

Observability

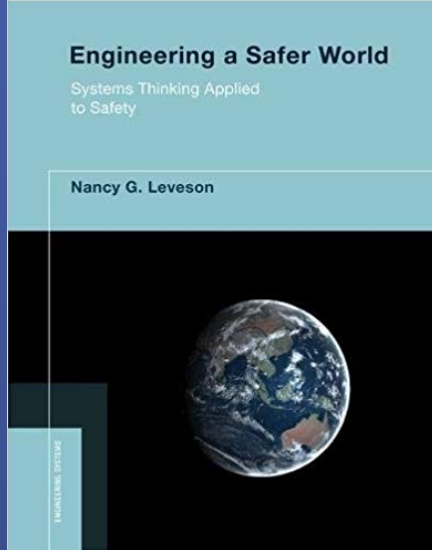


A diagram illustrating the relationship between Observability and Control in the context of the quote. Two white arrows point upwards from the words 'Observability' and 'Control' to the words 'fast detection' and 'response' respectively in the quote above. 'Observability' is colored yellow and 'Control' is colored purple, matching the colors of 'fast detection' and 'response' in the quote.

Control

Let's see what we can learn  
from experts who have been  
working on controlling safety  
critical systems for decades





# Engineering a Safer World

Systems Thinking Applied to Safety – 2012

Professor Nancy G. Leveson – MIT

STPA – Systems Theoretic Process Analysis

STAMP – Systems Theoretic Accident Model & Processes

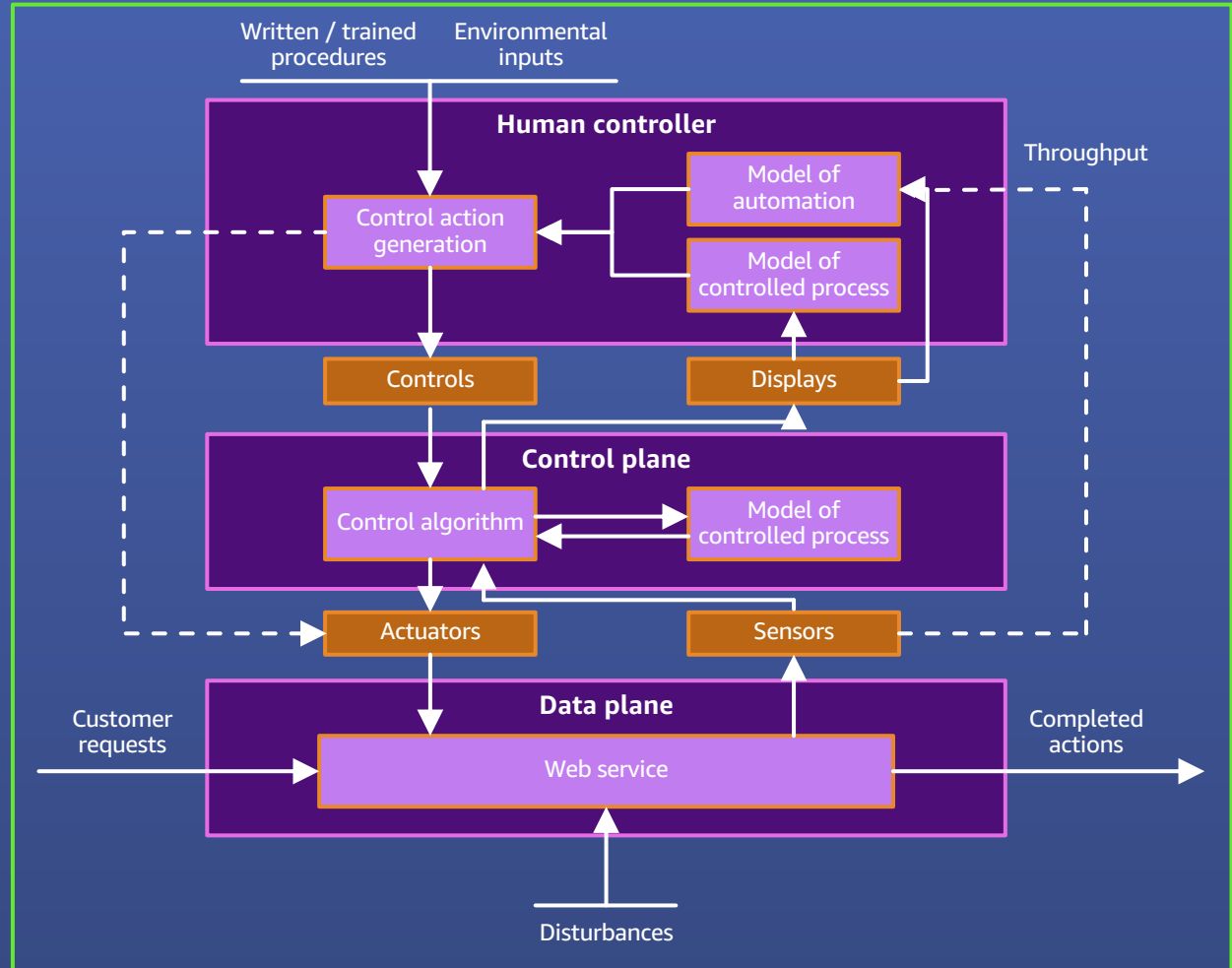
<http://psas.scripts.mit.edu> for handbook and talks



# Observability and control

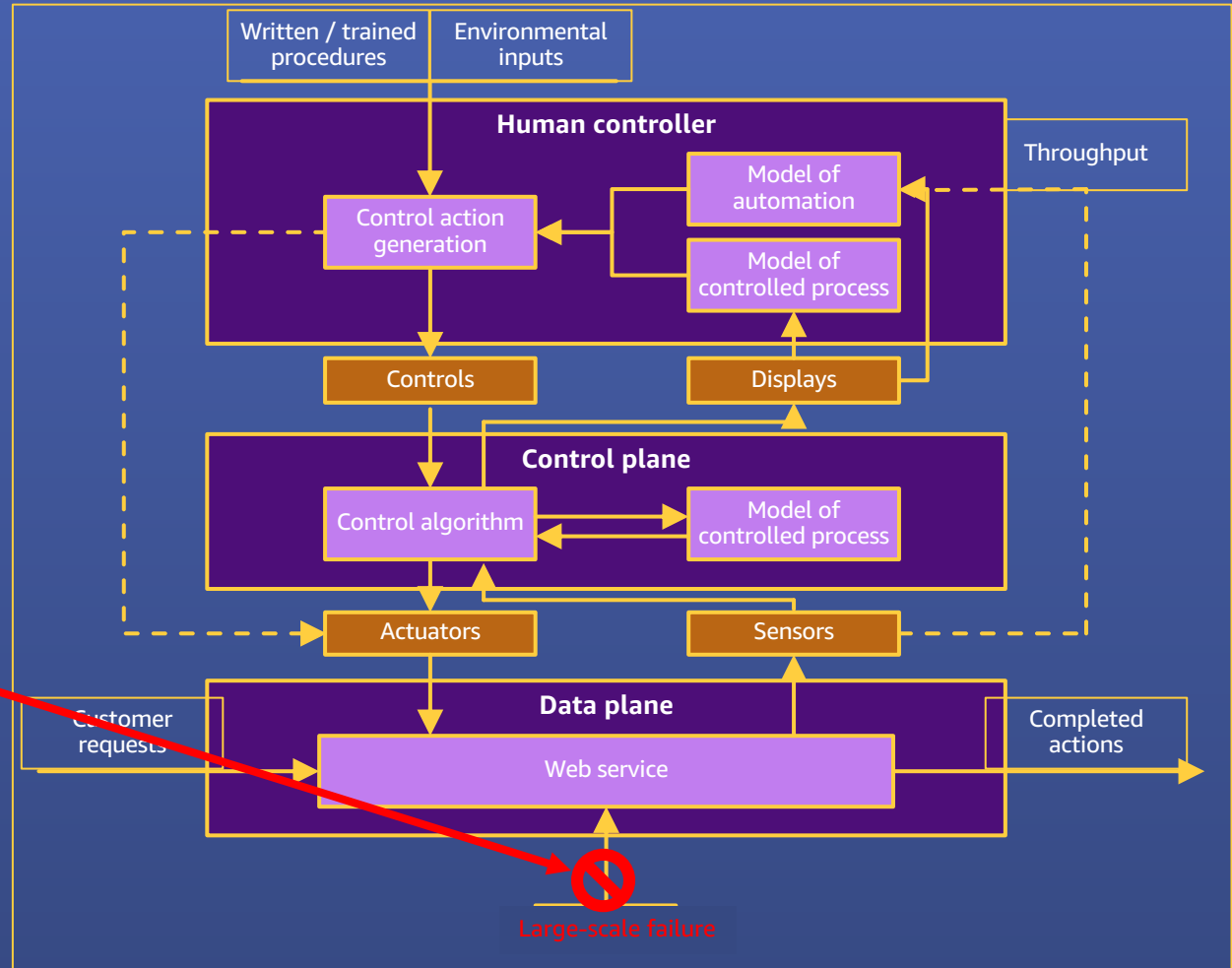
## STPA model control diagram

Understand hazards that could disrupt successful application processing





What happens if there is a big enough disturbance to break the web service?



# Large-scale failures

## **Application is “out of control,” many possibilities**

- Automation has failed

- Network partition, no route connecting application to customers

- Application crashed or corrupted, not easily restartable

- Anything else you didn't think of

## **Trigger failover to alternative system**

# Scenario

## AWS AVAILABILITY ZONES

### Symmetry and assertions

- Services and data are consistent across three zones
- Zone failure modes are independent
- Application should work normally with any zone offline
- Routing service manages failover



# Scenario

## AWS AVAILABILITY ZONES

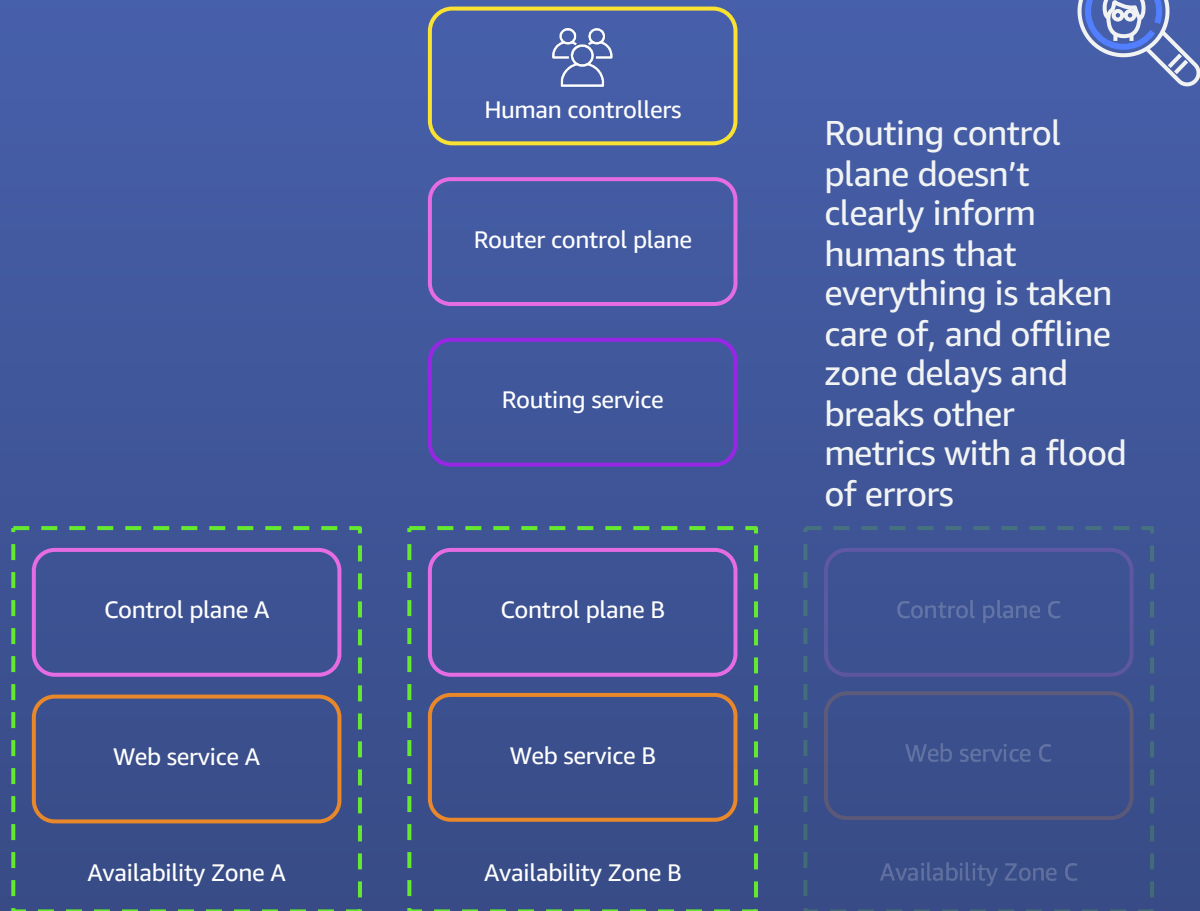
- Router control plane detects offline AZ, stops routing traffic to it, and retries requests on the online AZs
- Automated response; what could go wrong?



# STPA hazards

## SENSOR METRICS CHECKLIST

- Missing updates
- Zeroed
- Overflowed
- Corrupted
- Out of order
- Updates too rapid
- Updates infrequent
- Updates delayed
- Coordination problems
- Degradation over time



# STPA hazards

## MODEL PROBLEMS CHECKLIST

- Model mismatch
- Missing inputs
- Missing updates
- Updates too rapid
- Updates infrequent
- Updates delayed
- Coordination problems
- Degradation over time

Confused human controllers disagree among themselves about whether they need to do something or not, with floods of errors, displays that lag reality by several minutes, and out-of-date runbooks



Human controllers

Router control plane

Routing service

Control plane A

Web service A

Availability Zone A

Control plane B

Web service B

Availability Zone B

Control plane C

Web service C

Availability Zone C





# STPA hazards

## HUMAN CONTROL ACTION CHECKLIST

- Not provided
- Unsafe action
- Safe but too early
- Safe but too late
- Wrong sequence
- Stopped too soon
- Applied too long
- Conflicts
- Coordination problems
- Degradation over time

Human controllers should not need to do anything! However, they are confused and working separately, trying to fix different problems – some of their tools don't get used often, and are broken or misconfigured to do the wrong thing



Human controllers

Router control plane

Routing service

Control plane A

Web service A

Availability Zone A

Control plane B

Web service B

Availability Zone B

Control plane C

Web service C

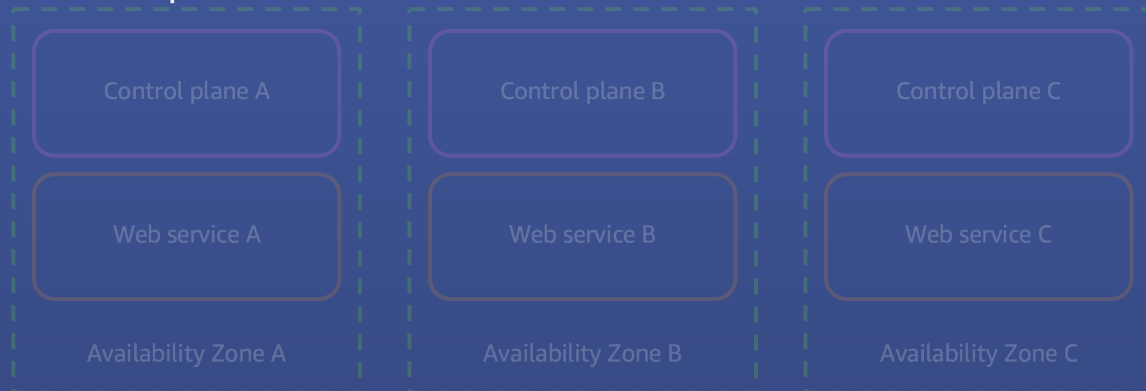
Availability Zone C



# Most likely result

**Instead of failing over, system falls over**

In-rush of extra traffic from failed zone, and extra work from a cross zone request retry storm causes zones A and B to struggle and triggers a complete failure of the application – meanwhile, the routing service also has a retry storm and is impacted



Failing over without  
falling over

# How to fail over without falling over

## Alert correlation

Floods of alerts need to be reduced to actionable insights (new: Amazon DevOps Guru)

Observability system needs to cope with floods without failing

Run regular chaos engineering experiments (new: AWS Fault Injection Simulator)

## Retry storms – prevent work amplification

Reduce retries to zero except at subsystem entry and exit points

Reduce timeouts to drop orphaned requests

Route calls within the same zone

## Symmetries

High level of automation, consistent configuration as code  
Consistent instance types, services, versions, zones, and Regions  
Patterns and guard rails for resilience

## Principles

If it can be the same, make it look and act identically  
If it's different, make that clearly visible  
Test your assumptions continuously

# What about multi-Region?



Get AZ failover solid  
before attempting  
multi-Region



Use STPA to  
analyze multi-Region-  
specific hazards



Follow  
Well-Architected  
guide patterns

# Best wishes for safer, faster deployments

Speed



Scale



Strategic



Time to  
value




Distributed  
optimized  
capacity



Critical workloads  
datacenter  
replacement



**Yaniv Bossem**  
Solutions Architect  
AWS  
@yanivbossem 

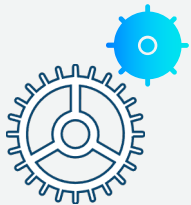


**“We needed to build systems  
that embrace failure as a  
natural occurrence.”**

Werner Vogels  
CTO  
AWS



# AWS Well-Architected Framework



Operational  
excellence



Security



Reliability



Performance  
efficiency



Cost  
optimization

# Testing resiliency

## Resiliency

“The ability of a system to recover from infrastructure or service disruptions...”

### Design principles for reliability

- Automatically recover from failure
- Test recovery procedures



<http://bit.ly/reliability-pillar>



# Fault isolation to protect your workload

## AWS Well-Architected Framework

[Add a link to your architectural design](#)

### REL 10. How do you use fault isolation to protect your workload? [Info](#)

Fault isolated boundaries limit the effect of a failure within a workload to a limited number of components. Components outside of the boundary are unaffected by the failure. Using multiple fault isolated boundaries, you can limit the impact on your workload.

☐ Question does not apply to this workload [Info](#)

Select from the following

☒ Deploy the workload to multiple locations [Info](#)

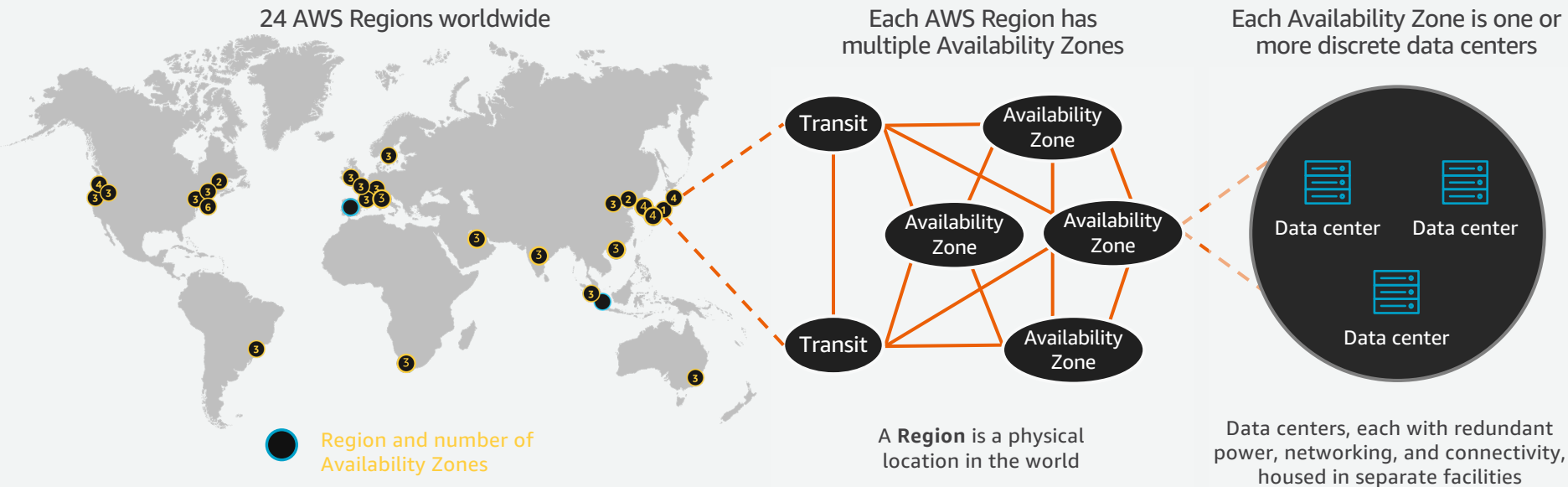
☐ Automate recovery for components constrained to a single location [Info](#)



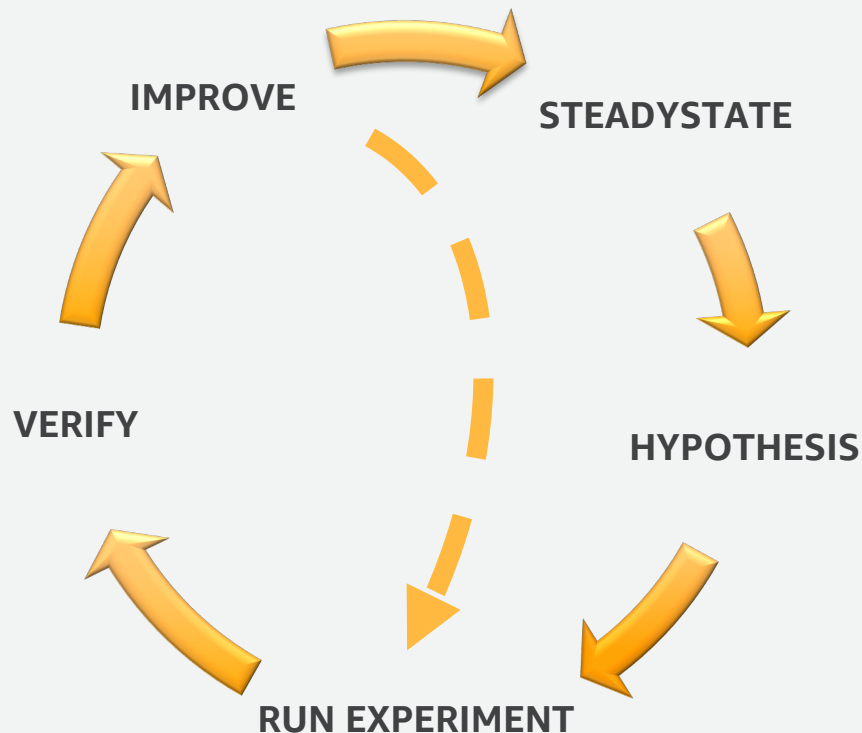
<https://bit.ly/use-fault-isolation>

# AWS Regions and Availability Zones

AWS Regions are physical locations around the world where we cluster data centers



# Chaos engineering: A scientific method



# Chaos experiment

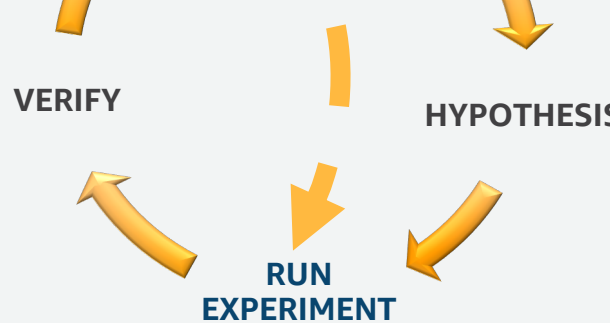
Inject **events** that simulate

**Hardware failures**, like servers dying

**Software failures**, like malformed responses

Nonfailure events, like spikes in traffic or **scaling** events

Any event capable of disrupting steady state



# Tools for the job

Plenty of open-source and commercial tools

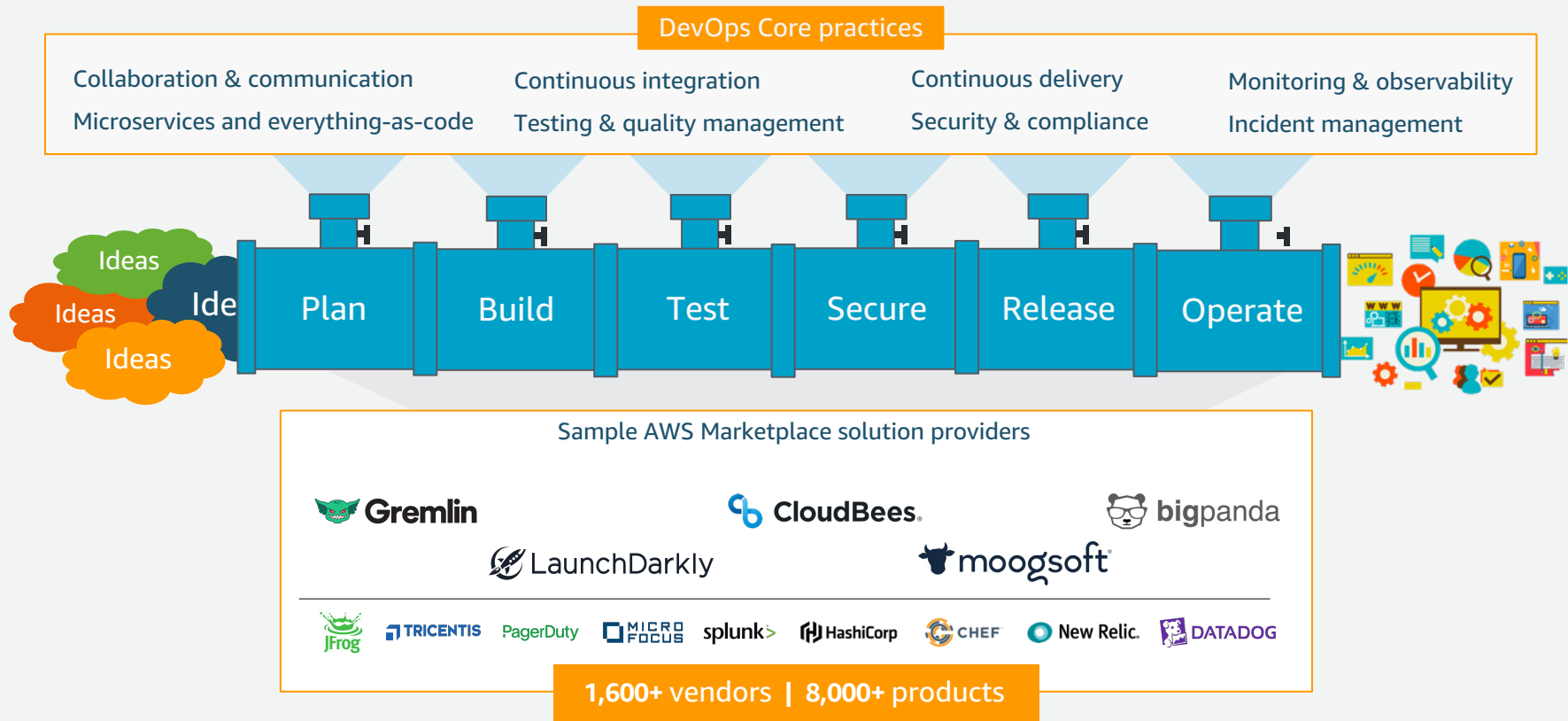
Just use these?

Yes! Absolutely. Please do.

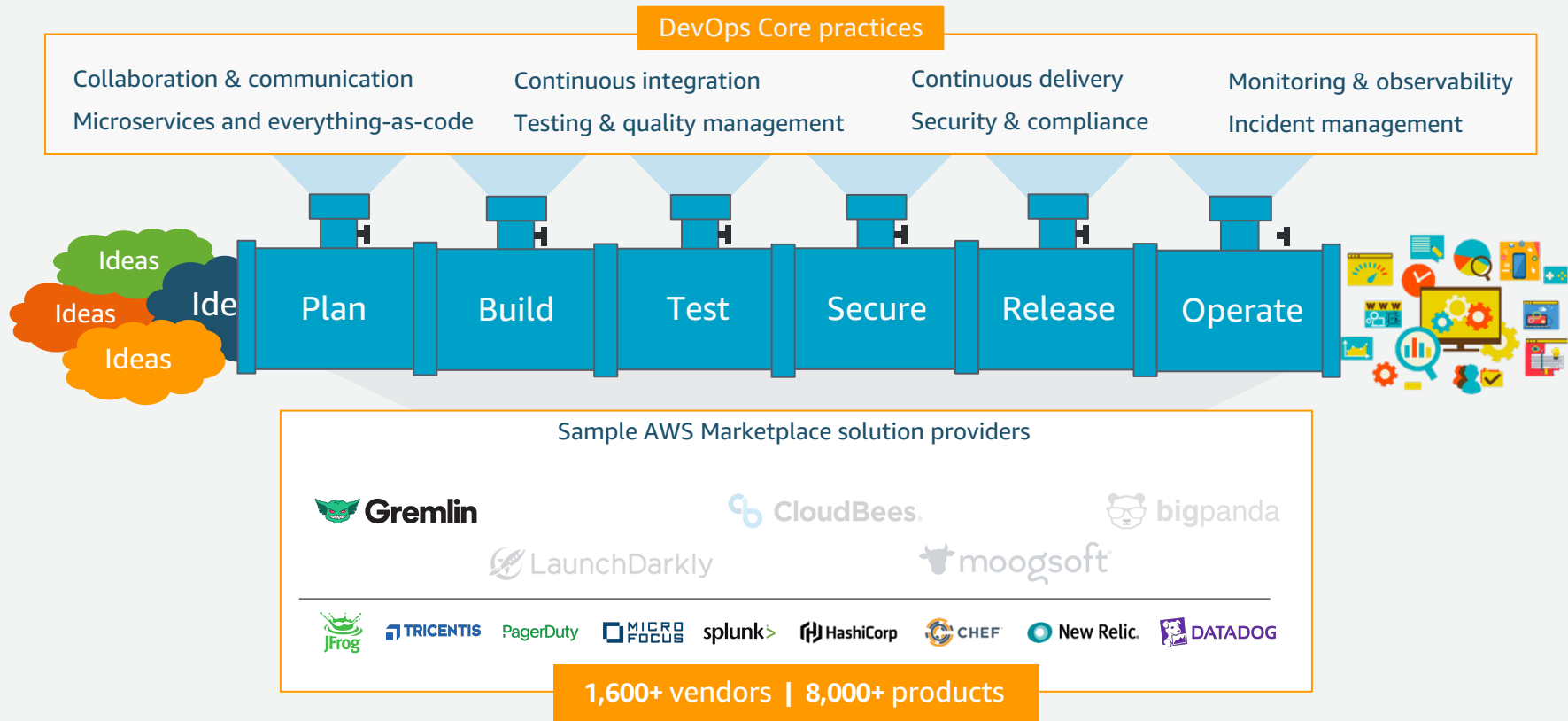




# AWS Marketplace: Destination for third-party solutions to use with AWS



# AWS Marketplace: Destination for third-party solutions to use with AWS



# Gremlin: Chaos Engineering Platform

## Attacks

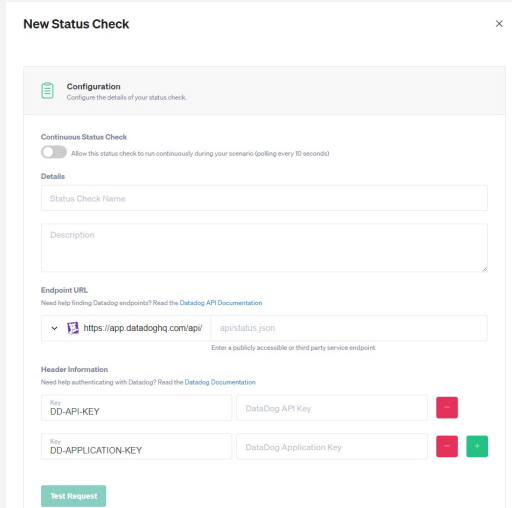
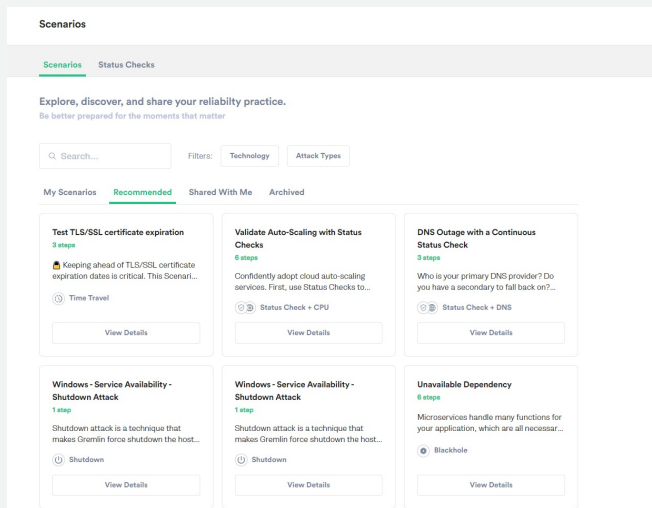
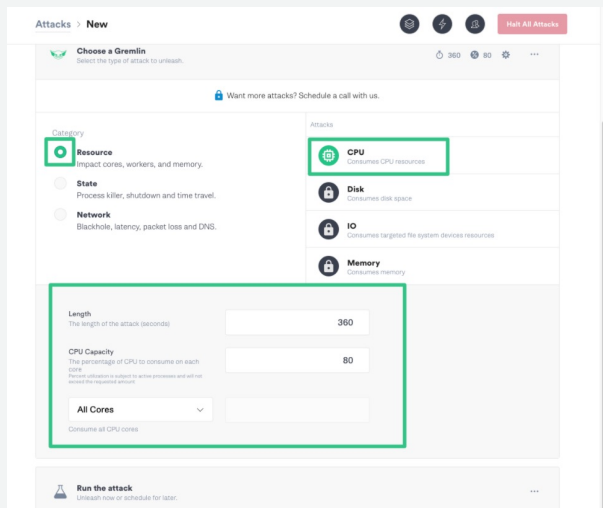
Use Gremlin's comprehensive set of 12 Chaos Engineering experiments to find weaknesses in your systems and reduce incidents.

## Guidance

Reduce your time to resolution by testing your monitoring and alerting, and preparing your teams with real-world failure scenarios.

## Safety

Continuous Status Checks let you safely monitor your systems during an attack or Scenario, and halt an experiment if your systems enter an undesirable state.



# JP Morgan Chase increases core resiliency

with Chaos Engineering using Gremlin

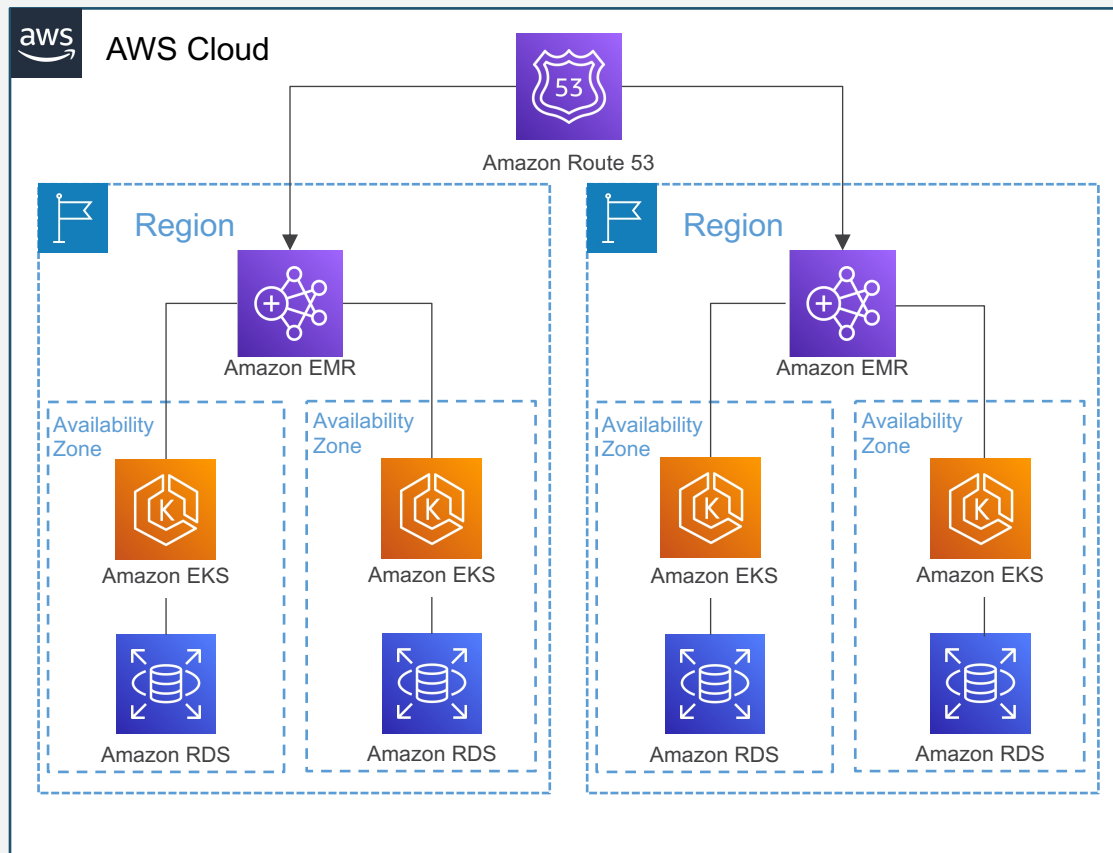
**Gremlin** | JPMORGAN CHASE & CO.

**Sufficient failover capacity**

**RTO/RPO achieved**

**Automated resiliency**

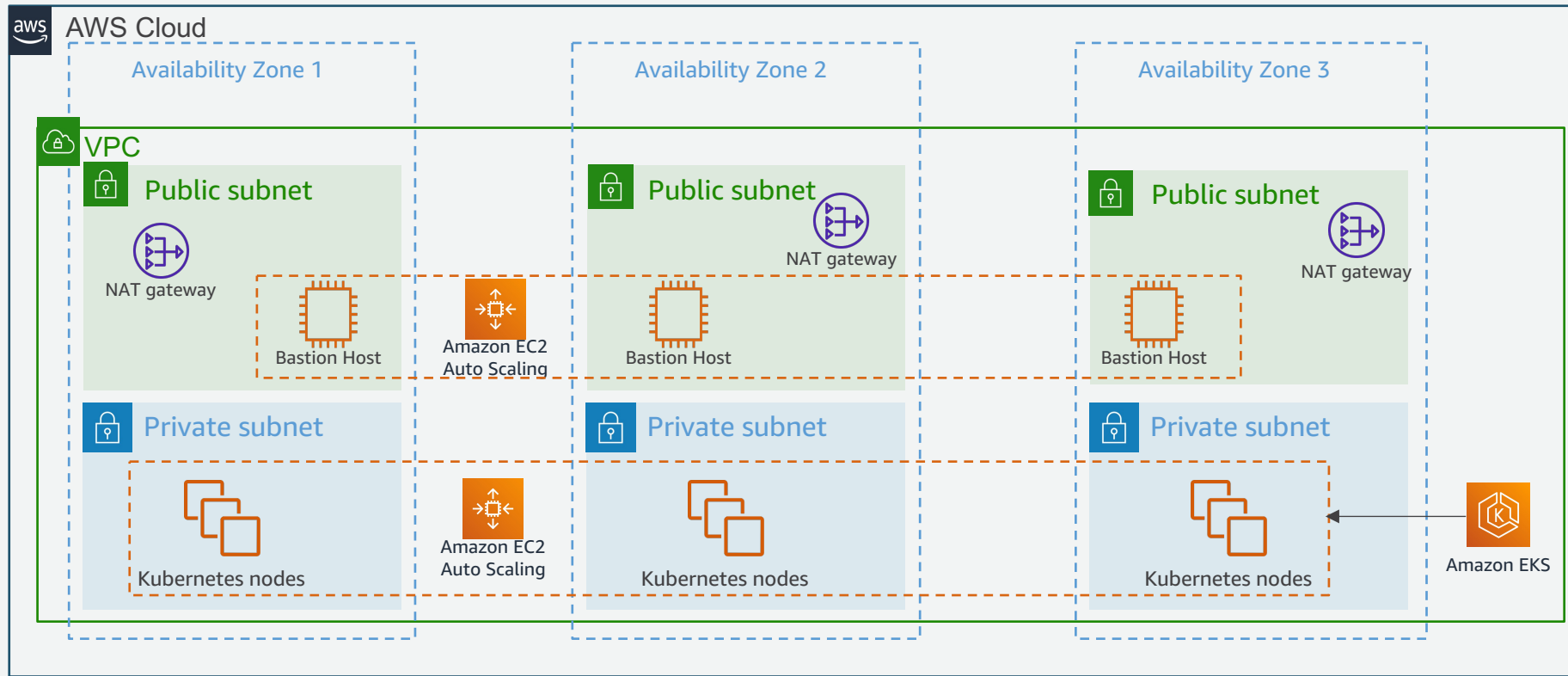
**Service degradation  
detection**



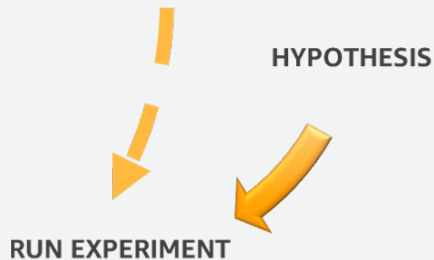
# Example : CPU increase experiment

<https://gremlin.awsworkshop.io/>

# Our architecture for the example



# Hypothesis



## Hypothesis 1

If our EC2 instances CPU increases, then availability will not be impacted

## Hypothesis 2

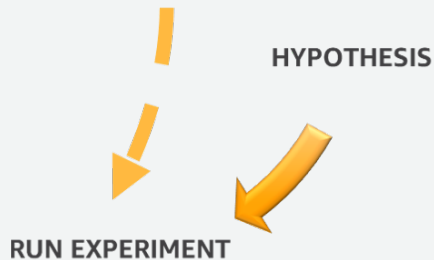
If an entire Availability Zone dies, then availability will not be impacted

Load will be adequately served by remaining two EC2 instances

Elastic Load Balancing will detect unhealthy instance  
and reroute requests

Autoscaling will create more instances

# Hypothesis



## Hypothesis 1

If our EC2 instances CPU increases, then availability will not be impacted

Load will be adequately served by remaining two EC2 instances

Elastic Load Balancing will detect unhealthy instance  
and reroute requests

Autoscaling will create more instances



APPLICATION NAME					
Sock Shop					
EXPERIMENT DETAILS		ABORT CONDITIONS		HYPOTHESIS	
<p>GREMLIN ATTACK</p> <p><b>Resource - CPU</b></p>		<ul style="list-style-type: none"> <li>- HTTP 500</li> <li>- Instance Count lowers</li> <li>- Data Loss</li> </ul>		X	NO IMPACT
<p>SCOPE OF FAILURE</p> <p>3/3 hosts   80% CPU Increase</p>					DEGRADED PERFORMANCE
<p>DURATION OF FAILURE</p> <p>360 seconds</p>					SERVICE OUTAGE
					IMPROVED PERFORMANCE
RESULTS			CONCLUSIONS		
<p>What did you see?</p>				EXPECTED	
				DETECTED	
				HANDLED	
				AUTOMATE	
				DIG DEEPER	



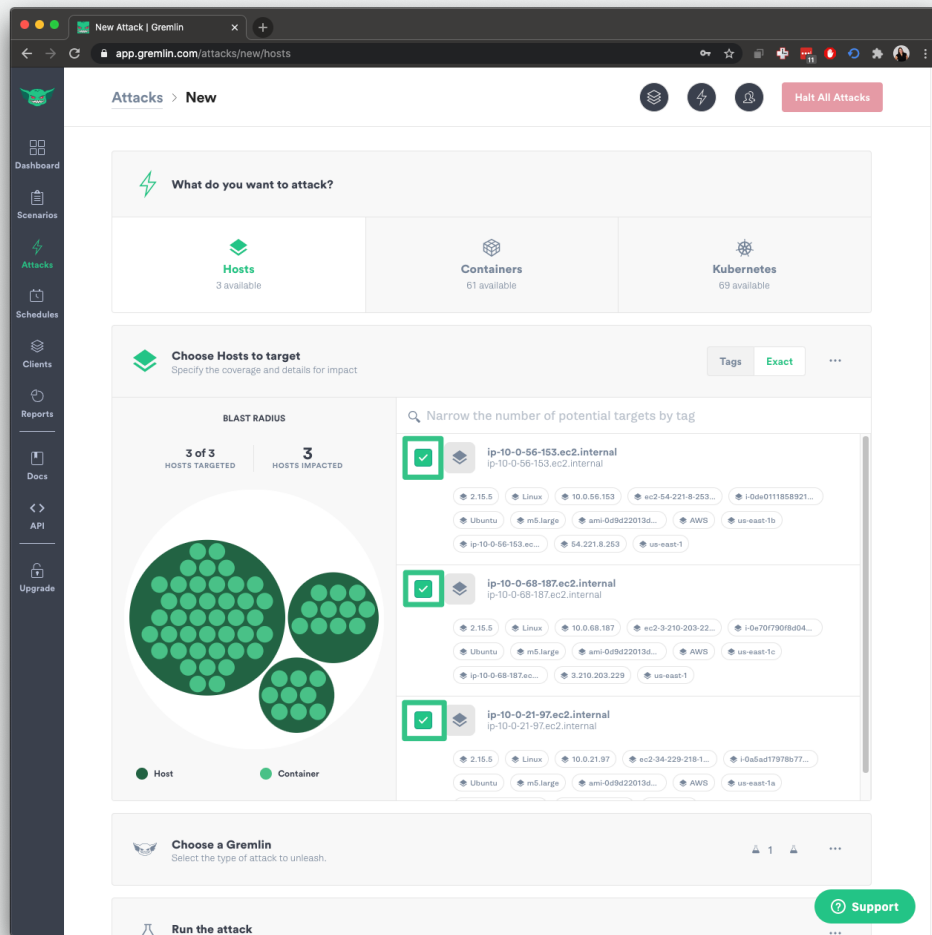
**Chaos Engineering: Bootcamp**

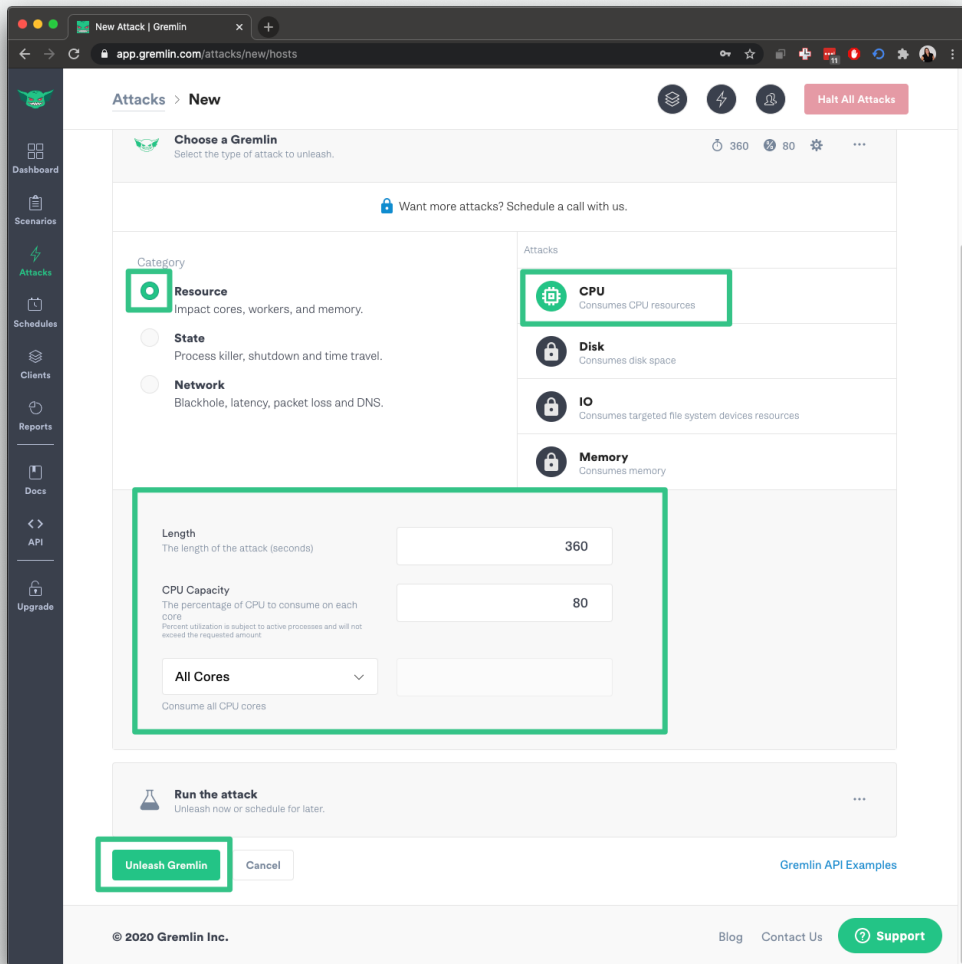
JOIN US ON [GREMLIN.COM/SLACK](https://gremlin.com/slack)

The screenshot shows the Gremlin web interface. The left sidebar contains navigation links: Dashboard, Scenarios, Attacks (highlighted with a green box), Schedules, Clients, Reports, Docs, API, and Upgrade. The main content area is titled 'Attacks' and has tabs for 'Infrastructure' (selected) and 'Scenarios'. A 'New Attack' button is highlighted with a green box. Below it, a 'Completed' section contains a search bar and a table of attacks.

Name	End Date	Length	Targets	Impact
<b>Blackhole</b> Successful	7 days ago 8/27/2020 8:43 am	5 minutes	ip-10-0-86-151.ec2.internal Docker	

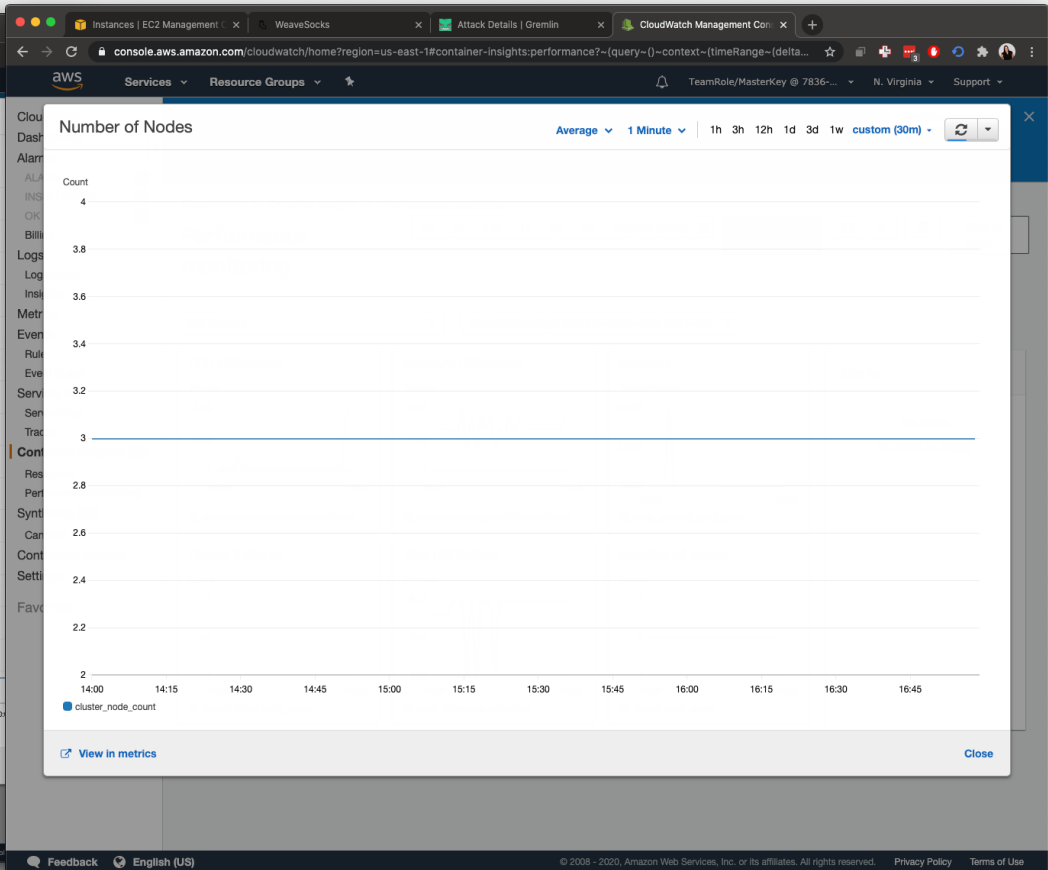
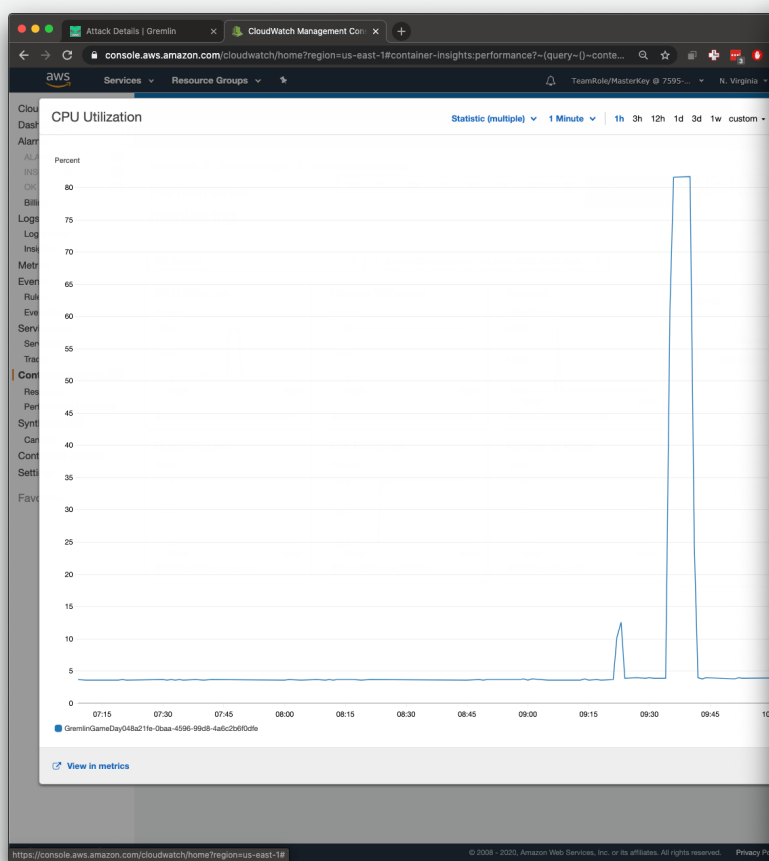
© 2020 Gremlin Inc. | [Blog](#) [Contact Us](#) [Support](#)



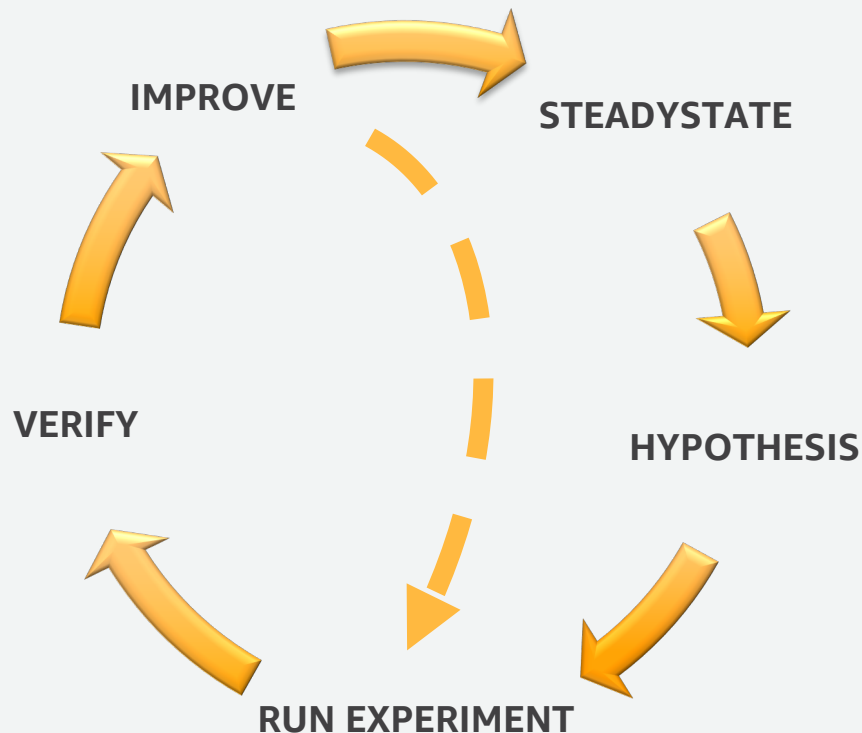


# One minute later...





# Chaos engineering: A scientific method



# Action items

- Setup auto scaling
- Setting up alarms
- Setting auto scaling up setup
- Setting auto scaling down setup



# Setup Auto scaling

The screenshot displays the AWS Management Console interface for configuring Auto Scaling Groups. The left sidebar shows the navigation menu with categories like INSTANCES, IMAGES, ELASTIC BLOCK STORE, NETWORK & SECURITY, LOAD BALANCING, and AUTO SCALING. The main content area is titled 'Auto Scaling groups (1/1)' and features a table listing the existing Auto Scaling group. The group 'mod-f679f4081e7 mod-f679f4081e7d405c | Version 1' is listed with 3 instances and a desired capacity of 3. Below the table, the 'Automatic scaling' tab is selected, showing 'Scaling policies (0)' and 'Scheduled actions (0)'. The 'Add policy' button is highlighted, indicating the next step in the setup process.

Name	Launch template/configuration	Instances	Status	Desired capacity
mod-f679f4081e7	mod-f679f4081e7d405c   Version 1	3	-	3

Scaling policies (0)

No scaling policies are currently specified

Add policy

Scheduled actions (0)

No scheduled actions are currently specified

Create scheduled action

# CloudWatch alarms

Step 3  
Add name and description

Step 4  
Preview and create

Warning  
This alarm will trigger when the blue line goes above the red line for 1 datapoints within 1 minute.

Percent

50

40

30

20

10

00:00 01:00 02:00

node\_cpu\_utilization

Namespace  
ContainerInsights

Metric name  
node\_cpu\_utilization

InstanceId  
i-0c95f27e695615909

NodeName  
ip-10-0-68-42.ec2.internal

ClusterName  
GremlinGameDay58615c90-3c98-4691-be36-2dc1

Instance name  
mod-f679f4081e7d405c-ng

Statistic  
Average

Period  
1 minute

Conditions

Threshold type

☒ Static  
Use a value as a threshold

☐ Anomaly detection  
Use a band as a threshold

Whenever node\_cpu\_utilization is...

Define the alarm condition.

☒ Greater  
> threshold

☐ Greater/Equal  
≥ threshold

☐ Lower/Equal  
≤ threshold

☐ Lower  
< threshold

than...  
Define the threshold value.  
50  
Must be a number

Additional configuration

Cancel Next


# Scale up

EC2 > Auto Scaling groups > mod-f679f4081e7d405c-NodeGroup-G5MEM2PRVPUR

## Create scaling policy

Policy type  
Simple scaling

Scaling policy name  
ScaleUp

CloudWatch alarm  
Choose an alarm that can scale capacity whenever:  
Scale-Up-Alarm 

[Create a CloudWatch alarm](#)

breaches the alarm threshold: node\_cpu\_utilization > 50 for 1 consecutive periods of 60 seconds for the metric dimensions:  
InstanceId = i-0c95f27e695615909  
NodeName = lp-10-0-68-42.ec2.Internal  
ClusterName = GremlinGameDay58615c90-3c98-4691-be36-2dc9a2867422

Take the action  
Add 1 capacity units

And then wait  
600 seconds before allowing another scaling activity

Cancel **Create**

# Scale down

The screenshot shows the AWS Management Console interface for creating a scaling policy. The breadcrumb trail indicates the path: EC2 > Auto Scaling groups > mod-f679f4081e7d405c-NodeGroup-G5MEM2PRVPUR. The form is titled 'Create scaling policy' and contains the following sections:

- Policy type:** A dropdown menu set to 'Simple scaling'.
- Scaling policy name:** A text input field containing 'ScaleDown'.
- CloudWatch alarm:**
  - A dropdown menu set to 'Scale-Down-Alarm' with a refresh icon.
  - A link: 'Create a CloudWatch alarm'.
  - Text: 'breaches the alarm threshold: node\_cpu\_utilization =< 30 for 1 consecutive periods of 60 seconds for the metric dimensions:'.
  - InstanceID = i-06db15c65fc70d96c
  - NodeName = ip-10-0-44-29.ec2.internal
  - ClusterName = GremlinGameDay58615c90-3c98-4691-be36-2dc9a2867422
- Take the action:**
  - A dropdown menu set to 'Set to'.
  - A text input field containing '3'.
  - A dropdown menu set to 'capacity units'.
- And then wait:**
  - A text input field containing '600'.
  - Text: 'seconds before allowing another scaling activity'.

At the bottom right of the form are 'Cancel' and 'Create' buttons. The footer of the console shows 'Feedback', 'English (US)', and copyright information for 2008-2020.

# Validate

New Attack | Gremlin

app.gremlin.com/attacks/new/hosts

Attacks > New

**Choose a Gremlin**  
Select the type of attack to unleash.

Want more attacks? Schedule a call with us.

**Category**

- Resource**  
Impact cores, workers, and memory.
- State  
Process killer, shutdown and time travel.
- Network  
Blackhole, latency, packet loss and DNS.

**Attacks**

- CPU**  
Consumes CPU resources
- Disk  
Consumes disk space
- IO  
Consumes targeted file system devices resources
- Memory  
Consumes memory

**Length**  
The length of the attack (seconds)

360

**CPU Capacity**  
The percentage of CPU to consume on each core  
Percent utilization is subject to active processes and will not exceed the requested amount

80

**All Cores**  
Consume all CPU cores

**Run the attack**  
Unleash now or schedule for later.

**Unleash Gremlin** Cancel

Gremlin API Examples

© 2020 Gremlin Inc. Blog Contact Us Support

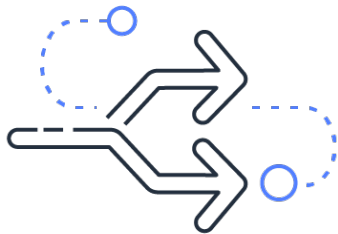
**When things do break, the most common causes were bad code pushes and dependency issues**

<https://www.gremlin.com/state-of-chaos-engineering>

**Things break! But we can minimize the  
impact and blast radius**

# Feature flags

- Allows A/B testing, beta groups etc
- Kill Switches and circuit breakers
- Config changes are versioned as code and deployed as code
- Throttle requests



Leading-edge feature management tools:





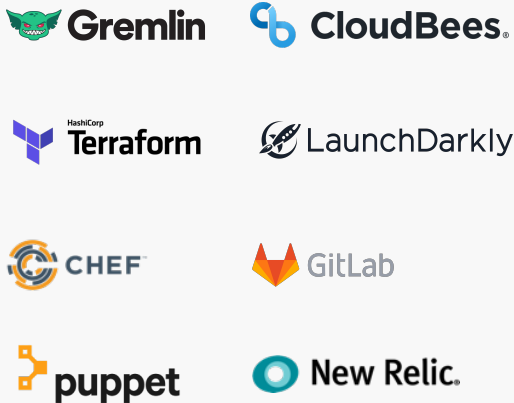
# Monitor while you rollout

# AWS Marketplace can help you get started

## Find



A breadth  
of DevOps solutions:



## Buy



Through flexible  
pricing options:

Free trial  
Pay-as-you-go  
Hourly | Monthly | Annual  
| Multi-Year  
Bring Your Own License (BYOL)  
Seller Private Offers  
Channel Partner Private Offers

## Deploy

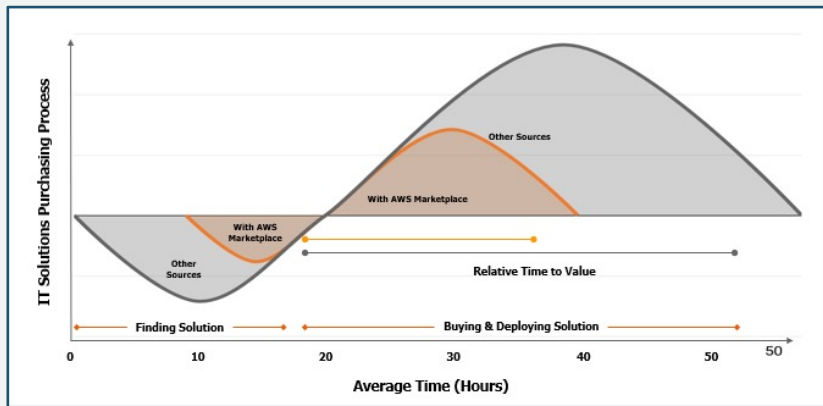


With multiple  
deployment options:

AWS Control Tower  
AWS Service Catalog  
AWS CloudFormation  
(Infrastructure as Code)  
Software as a Service (SaaS)  
Amazon Machine Image (AMI)  
Amazon Elastic Container Service  
(ECS)  
Amazon Elastic Kubernetes Service  
(EKS)

# Why AWS Marketplace?

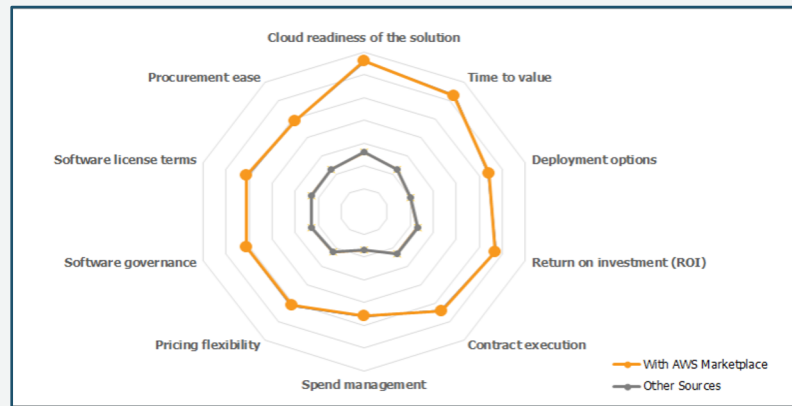
## Find, buy, and deploy solutions quicker



IT decision-makers (ITDMS) cut their time in half using AWS Marketplace compared to other sources.

\*Amazon Web Services (AWS) Marketplace surveyed 500 IT decision-makers (ITDMs) and influencers across the U.S. to understand software usage, purchasing, consumption models, and compared savings.

## Make more satisfying purchases



ITDMS feel 2.4x better about purchasing using AWS Marketplace compared to other sources.

# Webinar summary



Create a path to innovation for your organization



Validate your system's resiliency using Chaos engineering



Safely rollout code and test in production with feature management




Easily experiment with software using AWS Marketplace



**Adrian Cockcroft**  
VP Cloud Architecture Strategy  
AWS  
@adriancockcroft 



**Yaniv Bossem**  
Solutions Architect  
AWS  
@yanivbossem 

**Thank you!**