



SUMMIT  
ONLINE

O P E 1 0

# Building resilient applications using chaos engineering on AWS

Adrian Hornsby

Principal Technical Evangelist  
Amazon Web Services







# AWS GameDay at Amazon

- A volunteer **firefighter**
- Created **GameDay** in **2006** to purposefully create regular major failures
- Founded **Chef**, the Velocity Web Performance & Operations Conference



Jesse Robbins, “Master of Disaster”



# Rise of the monkeys

“Simian Army to keep our cloud safe, secure, and highly available.”

- 2011 Netflix blog

## Set of scheduled agent:

- Shuts down services randomly
- Slows down performances
- Checks conformity
- Breaks an entire region
- Integrates with spinnaker (CI/CD)

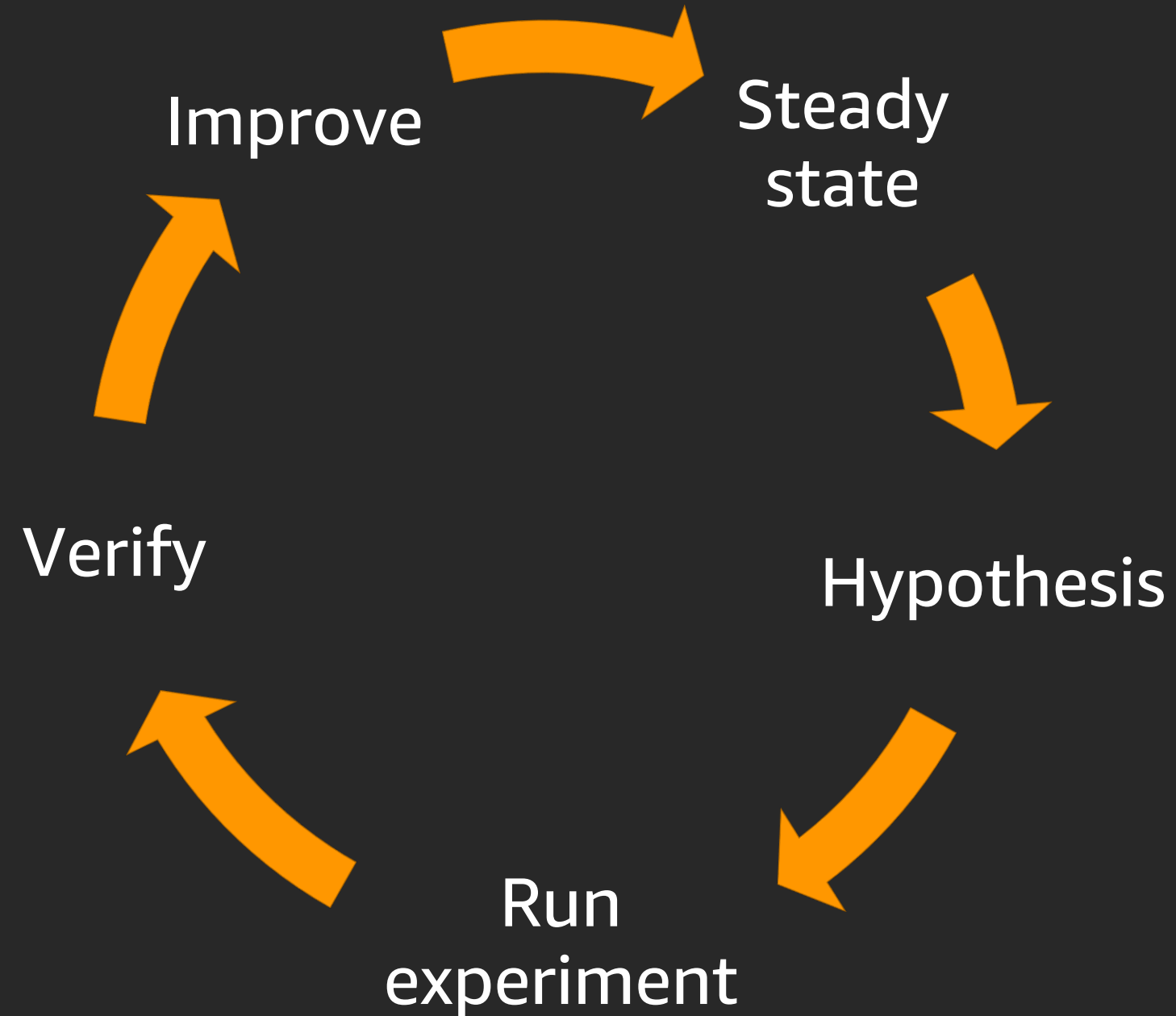


<https://github.com/Netflix/SimianArmy>

Chaos engineering is **NOT** about breaking things randomly **without a purpose**, chaos engineering is about breaking things in a **controlled environment** and through **well-planned experiments** in order to build confidence in your application to withstand turbulent conditions.

# Chaos engineering

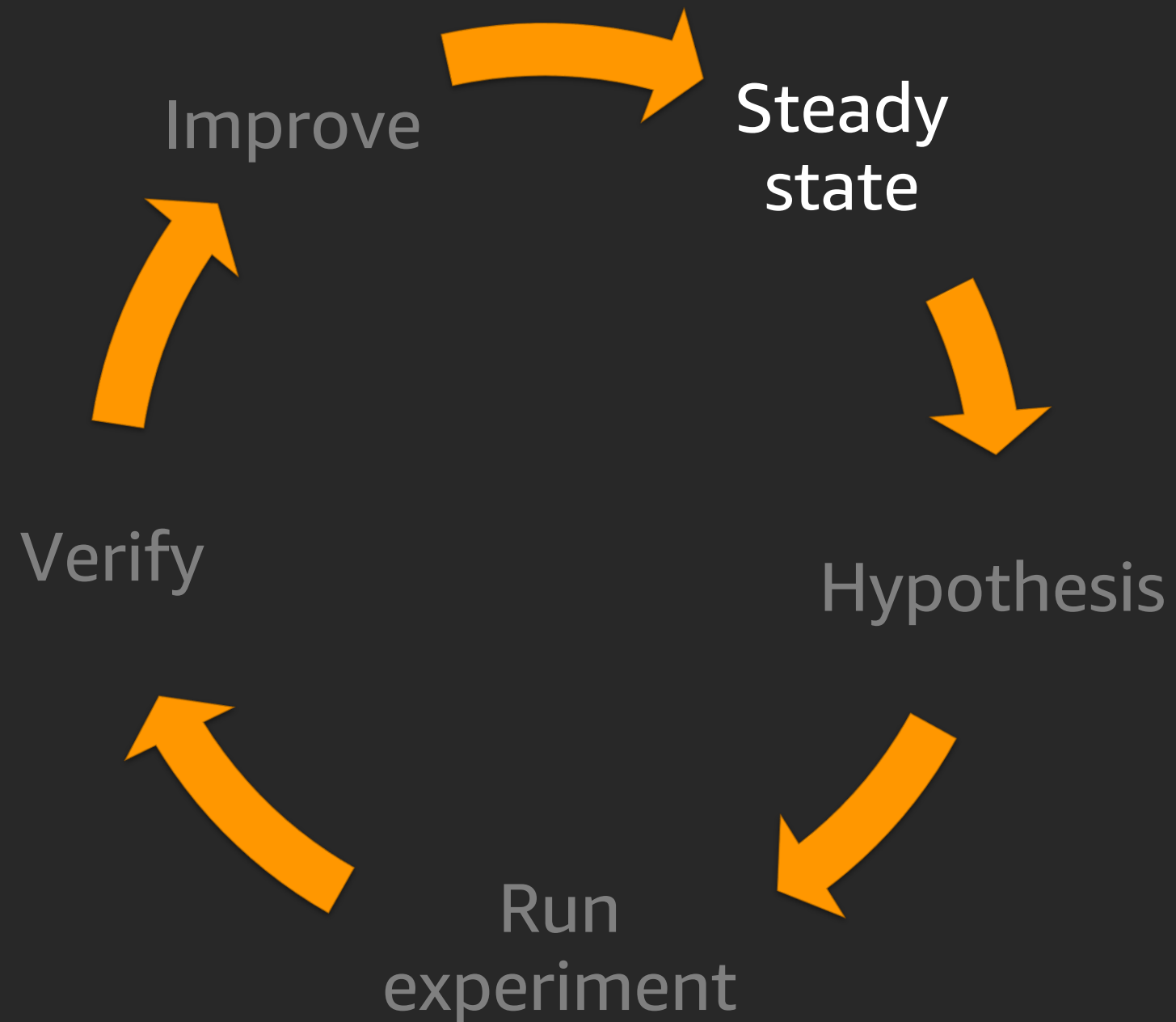
A scientific method



# Phases of chaos engineering

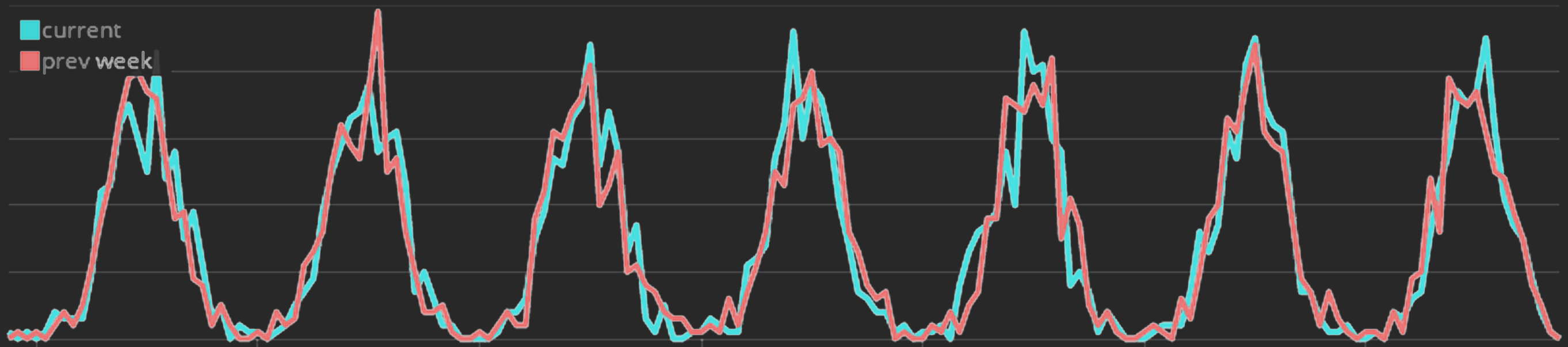


# Phases of chaos engineering



# What is steady state?

"Normal" behavior of your system



# What is steady state?

Business + ops metric



Orders per second



Netflix Technology Blog

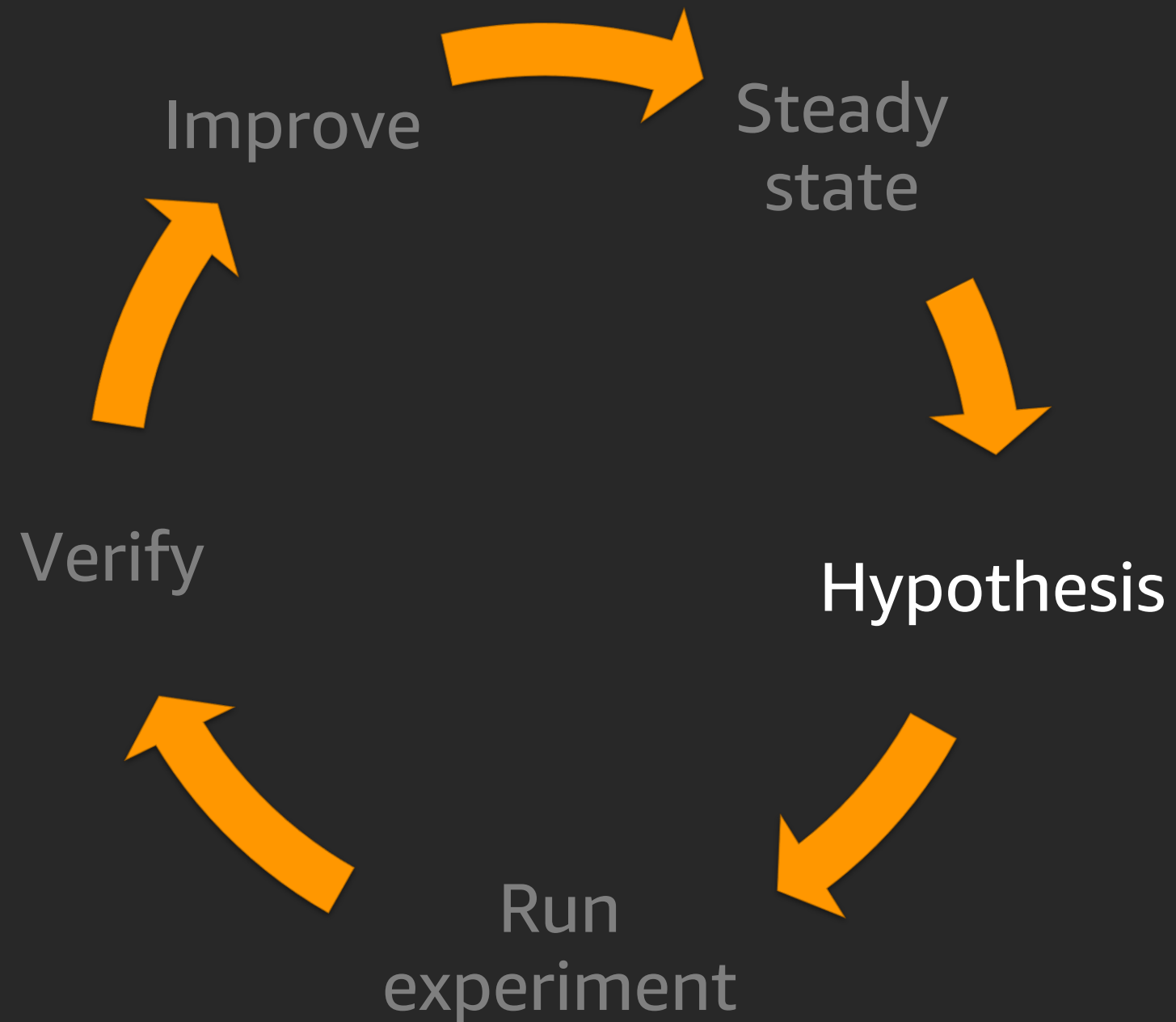
Learn more about how Netflix designs, builds, and operates our systems and engineering organizations

Feb 2, 2015 · 7 min read

## SPS: the Pulse of Netflix Streaming

<https://medium.com/netflix-techblog/>

# Phases of chaos engineering





# What if?

“What if this load balancer breaks?”

“What if Redis becomes slow?”

“What if a host on Cassandra goes away?”

“What if latency increases by 300ms?”

“What if the database stops?”

**Make it everyone's problem!**

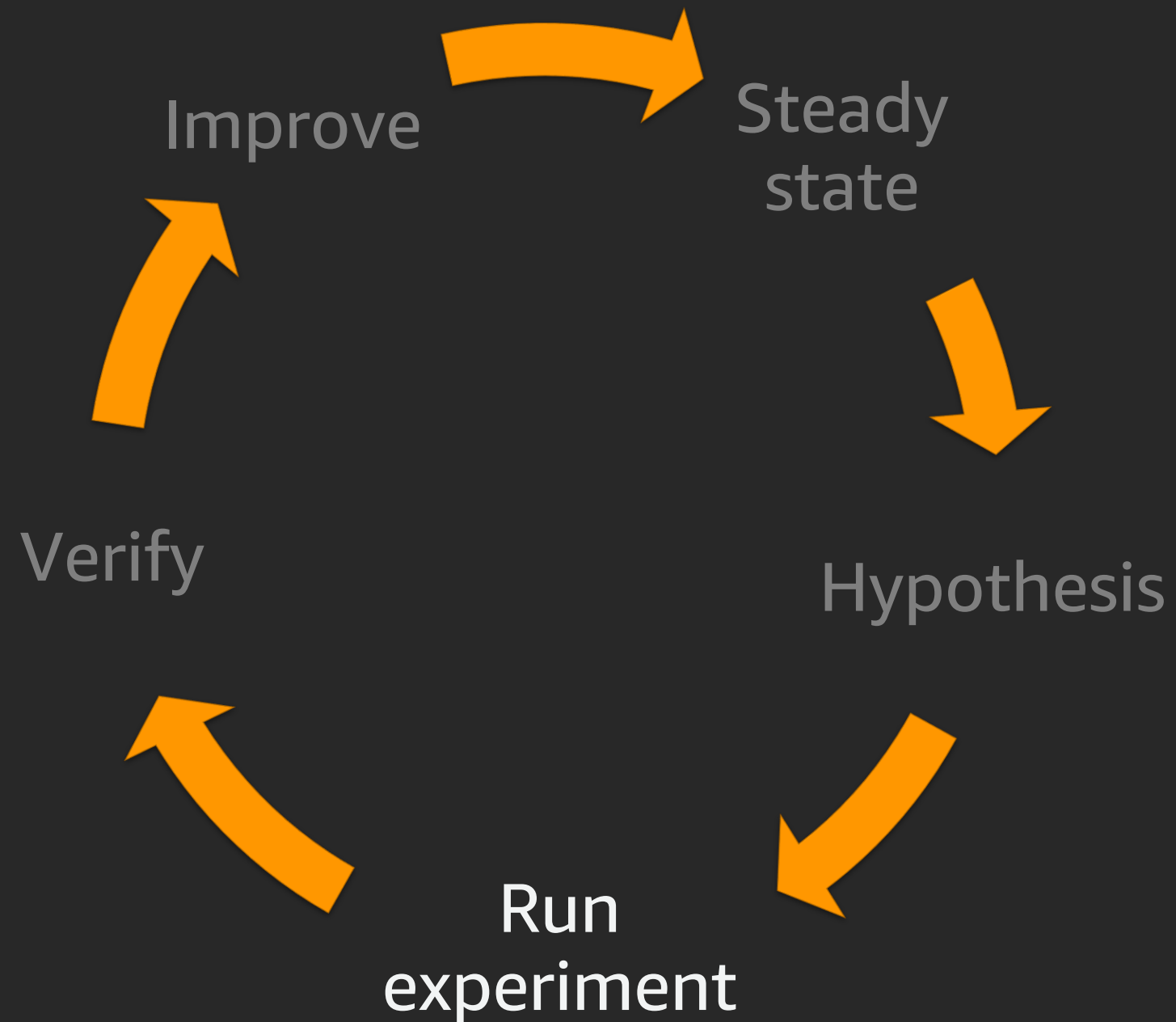
# Convergence



# Divergence



# Phases of chaos engineering



# Failure injection

## Start small and build confidence

- Application level (exceptions, errors, etc)
- Host level (services, processes, etc)
- Resource attacks (CPU, memory, IO, etc)
- Network attacks (dependencies, latency, packet loss, etc)
- AZ attack
- Region attack
- People attack

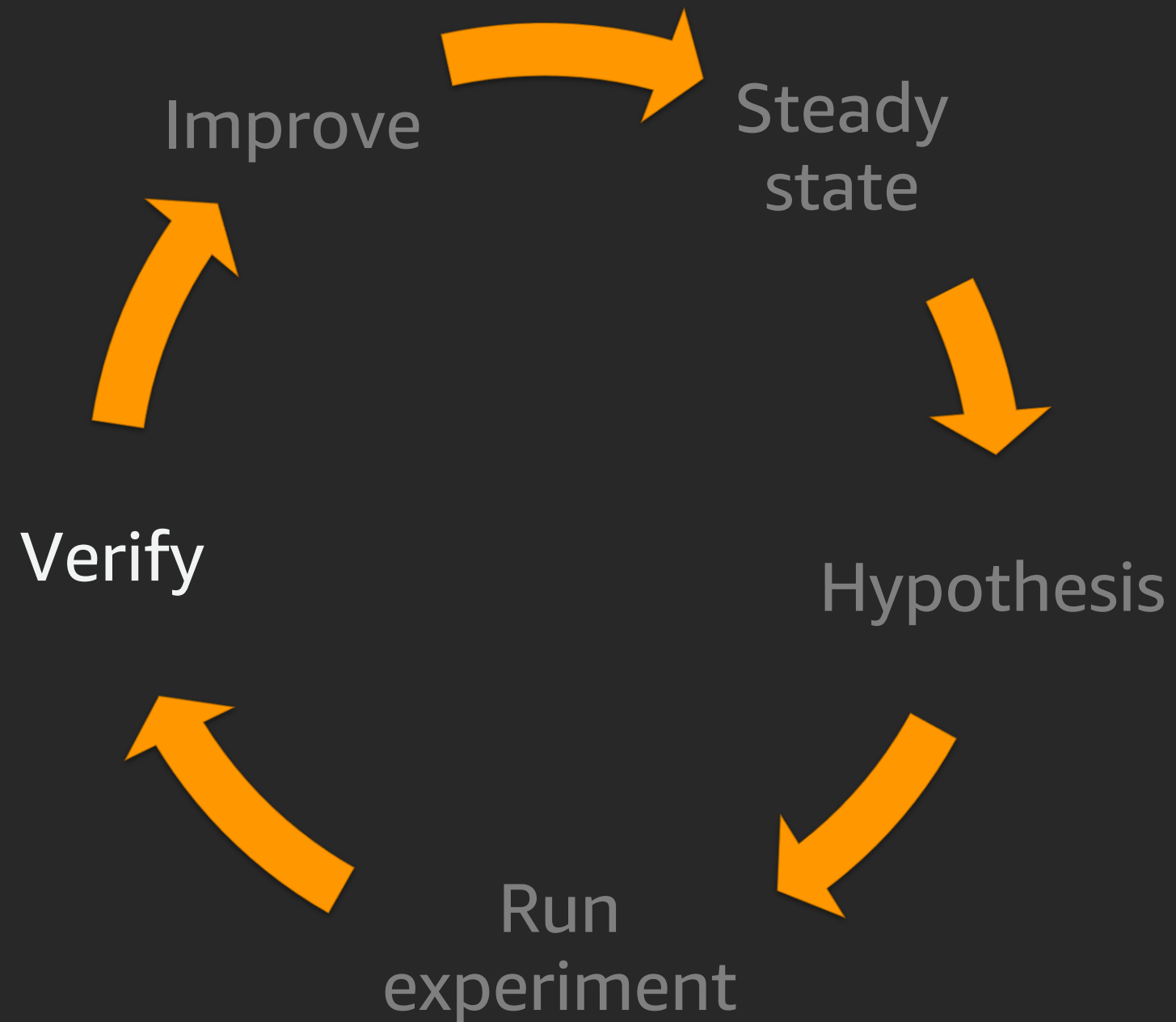


# Rules of thumbs

- Start very small
- As close as possible to production
- Minimise the blast radius.
- Have an emergency STOP!
  - Careful with state that can't be rolled back  
(corrupt or incorrect data)



# Phases of chaos engineering



# Quantifying the result of the experiment

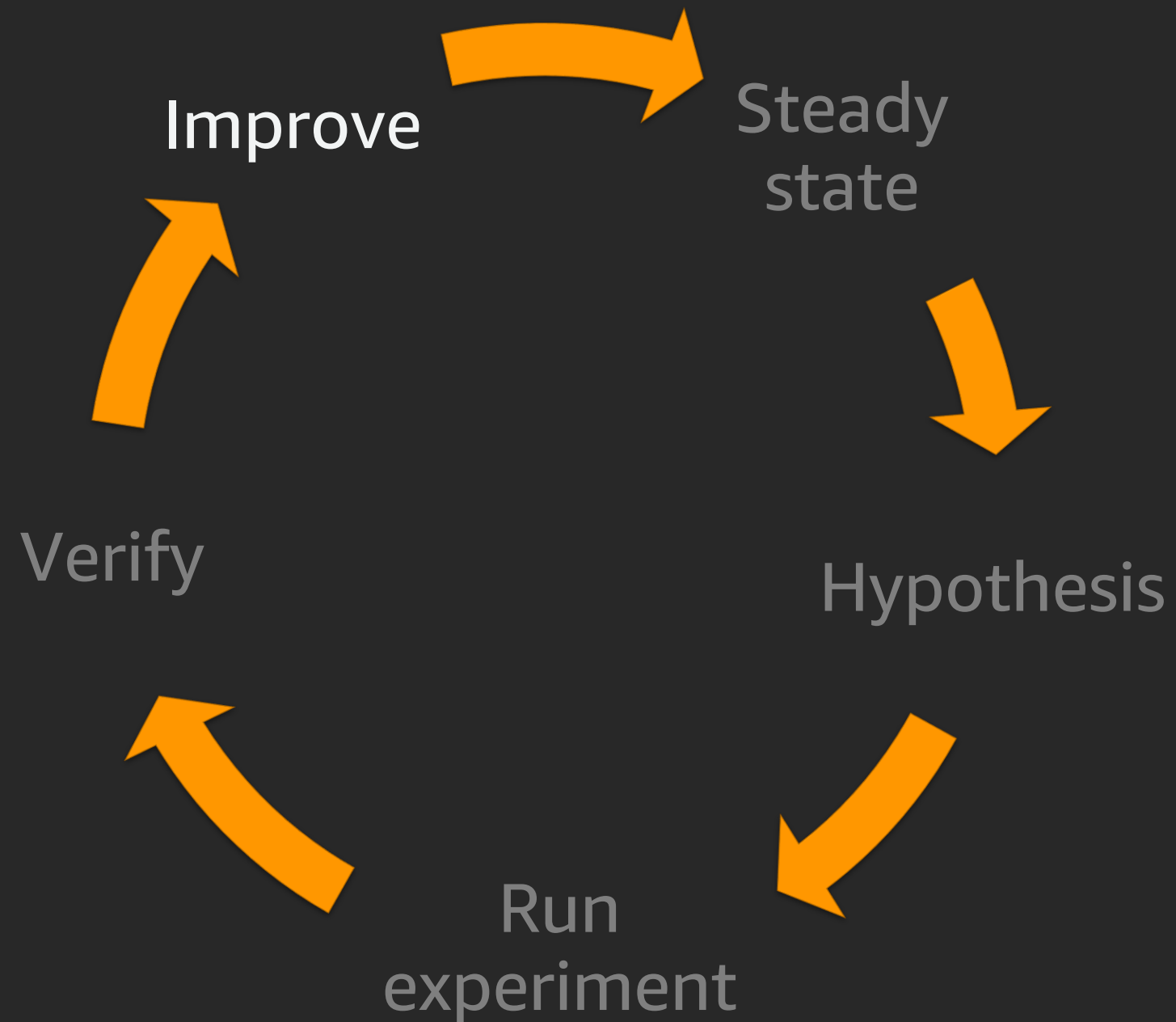
- Time to detect?
- Time for notification? And escalation?
- Time to public notification?
- Time for graceful degradation to kick-in?
- Time for self-healing to happen?
- Time to recovery – partial and full?
- Time to all-clear and stable?

# Postmortems – COE (Correction of Errors)

- What happened?
- How long did it take to detect the issue?
- Is there an existing backlog item that would've prevented the event?
- What was the impact on customers and your business?
- What were the contributing factors?
- What data do you have to support this?
- What lessons did you learn?
- What corrective actions are you taking?



# Phases of chaos engineering



Fix it!

# Tools and demos

Start simple and local!!

```
$ docker stop 94a214bbeebd
```

# DDoS yourself

```
$ wrk -t12 -c400 -d30s http://127.0.0.1/api/health
```

# Burn CPU with Stress(-ng)

<https://kernel.ubuntu.com/~cking/stress-ng/>

```
$ stress-ng --cpu 0--cpu-method matrixprod -t 60s
```



# Adding latency to the network

```
$ tc qdisc add dev eth0 root netem delay 300ms
```

# Blocks DNS resolution

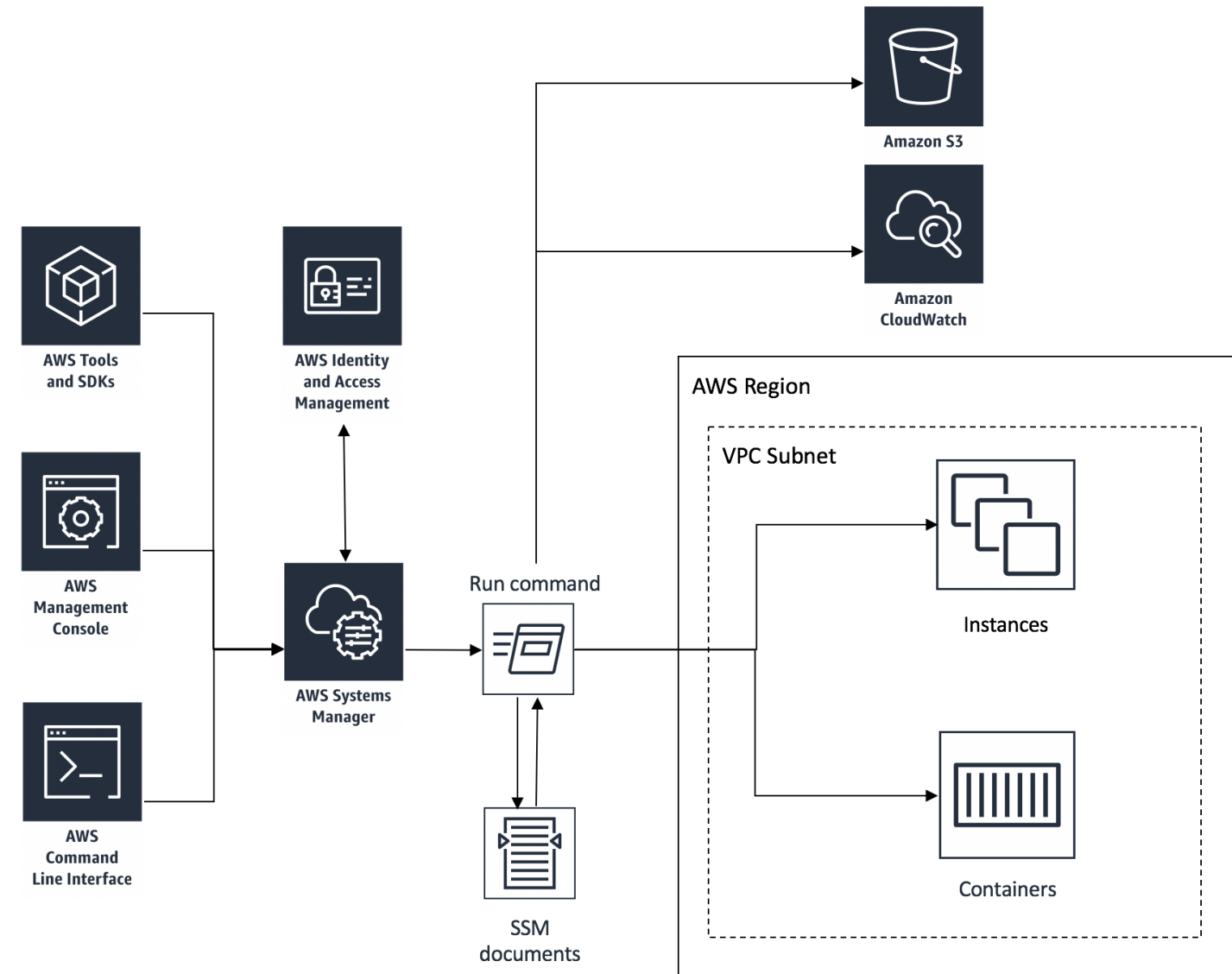
```
$ iptables -A INPUT -p tcp -m tcp --dport 53 -j DROP
```

# Other fun things to do

- Fill up disk
- Network packet loss (using traffic-shaping)
- Network packet corruption (using traffic-shaping)
- Kills random processes
- Detach (force) all EBS volumes
- Mess with /etc/hosts

# Injecting Chaos to Amazon EC2 using AWS System Manager

<https://medium.com/@adhorn/injecting-chaos-to-amazon-ec2-using-amazon-system-manager-ca95ee7878f5>



https://github.com/adhorn/chaos-ssm-documents

```
---
schemaVersion: '2.2'
description: Run a CPU stress on an instance
parameters:
  duration:
    type: String
    description: The duration - in seconds - of the attack. (Required)
    default: "60"
  cpu:
    type: String
    description: 'Specify the number of CPU stressors to use (default all)'
    default: "0"
mainSteps:
- action: aws:runShellScript
  name: ChaosCPUAttack
  inputs:
    runCommand:
      # https://www.mankier.com/1/stress-ng#Examples
      - stress-ng --cpu {{ cpu }} --cpu-method matrixprod -t {{ duration }}s
```

adhorn / chaos-ssm-documents

<> Code

Issues 0

Pull requests 0

Projects 0

Wiki

Security

Insights

Unwatch

Collection of AWS SSM Documents to perform Chaos Engineering experiments

chaos-engineering

chaos-monkey

chaos-testing

aws

aws-ec2

amazon-web-services

software-engine

15 commits

1 branch

0 releases

1 contributors

Branch: master

New pull request

Create new file

Upload file

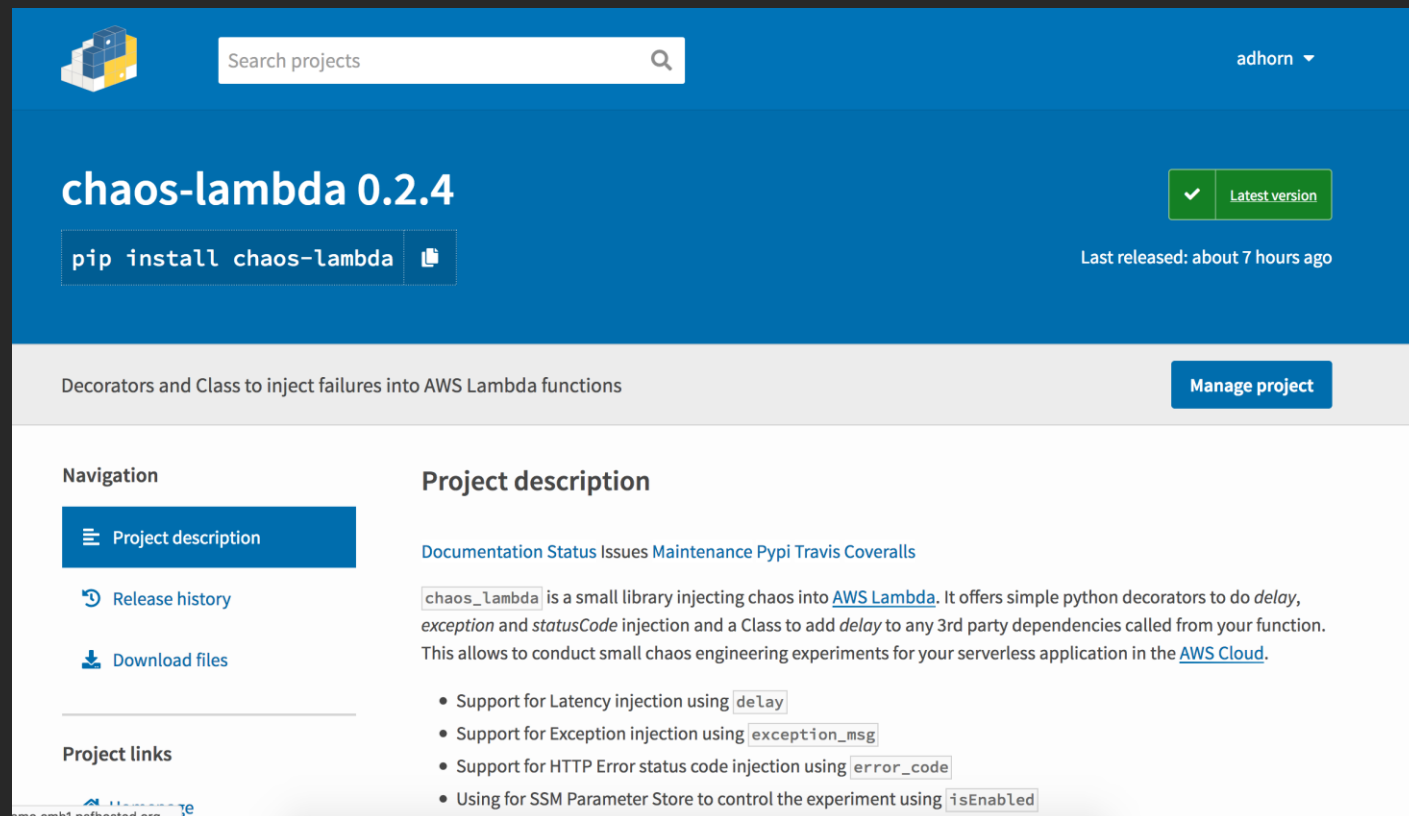
adhorn

improve readme

LICENCE	Adding Licence
README.rst	improve readme
blackhole-dns-stress.yml	Adding more attacks and fixing some typos
blackhole-dynamo-stress.yml	Adding dynamodb blackhole attack
blackhole-ec2-stress.yml	Adding EC2 blackhole attack
blackhole-s3-stress.yml	Adding S3 blackhole attack
blackhole-stress.yml	Adding more attacks and fixing some typos
cpu-stress.yml	Adding more attacks and fixing some typos
io-stress.yml	Adding more attacks and fixing some typos
latency-delta-stress.yml	Adding more attacks and fixing some typos
latency-stress.yml	Adding more attacks and fixing some typos
memory-stress.yml	Adding more attacks and fixing some typos
network-corruption-stress.yml	Adding more attacks and fixing some typos
network-loss-stress.yml	Adding more attacks and fixing some typos
upload-document.sh	adding tags

# Injecting chaos to AWS Lambda

```
$ pip install chaos-lambda
```



The screenshot shows the PyPI project page for **chaos-lambda 0.2.4**. The header is blue with a search bar and the user 'adhorn'. Below the header, the project name and version are displayed, along with a 'pip install chaos-lambda' button and a 'Latest version' badge. A description states: 'Decorators and Class to inject failures into AWS Lambda functions'. A 'Manage project' button is also present. The main content area is divided into 'Navigation' (Project description, Release history, Download files) and 'Project description'. The description includes links to Documentation, Status, Issues, Maintenance, Pypi, Travis, and Coveralls. It explains that `chaos_lambda` is a small library for injecting chaos into AWS Lambda, offering decorators for `delay`, `exception`, and `statusCode` injection, and a class for adding `delay` to 3rd party dependencies. It also mentions that this allows for small chaos engineering experiments in the AWS Cloud. A list of features is provided: Support for Latency injection using `delay`, Support for Exception injection using `exception_msg`, Support for HTTP Error status code injection using `error_code`, and Using for SSM Parameter Store to control the experiment using `isEnabled`.

```
import os
from chaos_lambda import inject_delay, inject_exception, inject_statuscode
```

```
# this should be set as a Lambda environment variable
os.environ['CHAOS_PARAM'] = 'chaoslamba.config'
```

```
@inject_exception
def handler_with_exception(event, context):
    return {
        'statusCode': 200,
        'body': 'Hello from Lambda!'
    }
```

```
@inject_exception(exception_type=TypeError, exception_msg='foobar')
def handler_with_exception_arg(event, context):
    return {
        'statusCode': 200,
        'body': 'Hello from Lambda!'
    }
```

```
@inject_exception(exception_type=ValueError)
def handler_with_exception_arg_2(event, context):
    return {
        'statusCode': 200,
        'body': 'Hello from Lambda!'
    }
```

```
@inject_statuscode
def handler_with_statuscode(event, context):
    return {
        'statusCode': 200,
        'body': 'Hello from Lambda!'
    }
```

```
@inject_statuscode(error_code=400)
def handler_with_statuscode_arg(event, context):
    return {
        'statusCode': 200,
        'body': 'Hello from Lambda!'
    }
```

```
@inject_delay
def handler_with_delay(event, context):
    return {
        'statusCode': 200,
        'body': 'Hello from Lambda!'
    }
```



https://github.com/adhorn/aws-lambda-chaos-injection

adhorn / aws-lambda-chaos-injection

Used by1

Watch5

★

<> Code

🔔 Issues4

🔗 Pull requests2

📁 Projects0

📖 Wiki

🛡 Security

📊 Insights

⚙ Settings

Chaos Injection library for AWS Lambda

serveless

chaos-engineering

chaos-monkey

aws

lambda-functions

amazon-web-services

sre

testing

Manage

🕒 77 commits

🌿 5 branches

📦 0 releases

👤 2 contributors

Branch: master

New pull request

Create new file

Upload files

Find File

adhorn

Merge pull request #15 from adhorn/refactortests

...

Latest commit

examples

pep8 love - part2

source

Allowing dynamic configuration of the decorator with arguments as def...

tests

fix test exception not called

.gitignore

cover to gitignore

.travis.yml

trying to fix travis.ci part 2

CODE\_OF\_CONDUCT.md

Adding Code of Conduct

LICENSE

Initial commit - FailureInjection Library

Makefile

Improve the Doc, setup

README.rst

minor documentation change

chaos\_lambda.py

version bump

conf.py

version bump

index.rst

refactoring - naming convention, making it more pythonic ;)

make.bat

Refactor to make it distribution ready

pytest.ini

trying to fix travis.ci

readthedocs.yml

fix the version of requirements needed to build the doc

requirements-dev.txt

trying to fix travis.ci

requirements.txt

fix requirements against ...

setup.cfg

trying to fix travis.ci

<https://github.com/adhorn/aws-chaos-scripts>

- Fail AZ Networking
- Fail ElastiCache
- Fail RDS
- Fail Instances

adhorn / aws-chaos-scripts

Unwatch

5

<> Code

🔔 Issues 0

🔗 Pull requests 0

🎬 Actions

📁 Projects 0

📖 Wiki

🛡 Security

📊 Insights

Collection of python scripts to run failure injection on AWS infrastructure

aws

chaos-engineering

chaos-monkey

software-engineering

amazon-web-services

sre

Manage topics

🕒 26 commits

🌿 1 branch

📦 0 packages

🏷 0 releases

👤 2 contributors

Branch: master

New pull request

Create new file

Upload files

Find file

adhorn Merge pull request #1 from setheliot/master

Latest commit

📁 scripts

fail\_az.py: add user configurable flag for limit ASG

📄 .gitignore

initial commit

📄 AUTHORS

initial commit

📄 LICENSE

initial commit

📄 README.md

README (fail\_az.py): add user configurable flag for limit ASG

📄 requirements.txt

Adding script to stop random instance

📄 setup.py

Adding support for elasticache redis failover.

📖 README.md

Disclaimer

⚠ USE AT YOUR OWN RISK ⚠

Using these scripts may create an unreasonable risk. If you choose to use the scripts provided here in activities, you do so at your own risk. None of the authors or contributors, or anyone else connected with in any way whatsoever, can be responsible for your use of the scripts contained in this repository. Use if you understand what the code does

Collection of python scripts to inject failure in the AWS Infrastructure

# Challenges of chaos engineering

# Big challenges to chaos engineering

## Mostly cultural

- No time or flexibility to simulate disasters
- Teams already spending all of its time fixing things
- Can be very political
- Might force deep conversations
- Deeply invested in a specific technical roadmap (micro-services) that chaos engineering tests show is not as resilient to failures as originally predicted

# Thank you!

Adrian Hornsby

<https://medium.com/@adhorn>

