



AWS Well-Architected

# Testing Resiliency of EC2, RDS, and S3

Rodney Lester

Reliability Lead, AWS Well-Architected

# Agenda

Deploying the Test Infrastructure

AWS Well-Architected Reliability Pillar

Scenario Walk Through

Lab Time

Component and Availability Zone Failure Simulation

Summary of Tests and Best Practices

# Deploying the Test Infrastructure

---

# Lab prerequisites

- Use your **own AWS Account** to build out the labs today
- You must have an AWS role with **administrative privilege**
- It is **your responsibility** to delete any AWS resources after today to **prevent ongoing costs!**

# Go to the Lab Guide

- Navigate to the lab guide

- <http://bit.ly/WARel300>

- 1. Deploy the Infrastructure

- We can start testing when the web servers are available
- This will take about 20 minutes until you can perform the tests in step 3

- 2. Configure Execution Environment

- After section 1, while your service is deploying, you can start section 2
- If you do not finish this section don't worry, you will have time later



# AWS Well-Architected Reliability Pillar

---

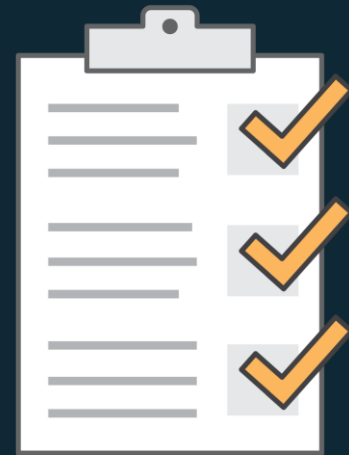
# What is it?

“The reliability pillar encompasses the ability of a system to recover from infrastructure or service disruptions, dynamically acquire computing resources to meet demand, and mitigate disruptions such as misconfigurations or transient network issues.”

– AWS Well Architected Framework Whitepaper

## Design Principles

- Test recovery procedures
- Automatically recover from failures
- Scale horizontally to increase aggregate system availability
- Stop guessing capacity
- Manage change using automation



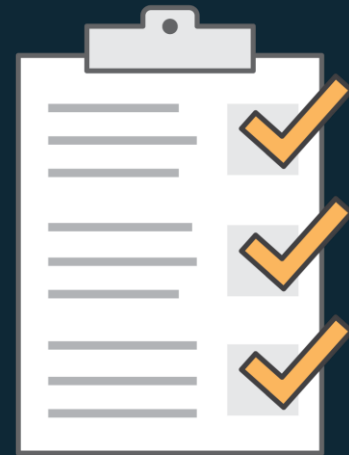
# What is it?

“The reliability pillar encompasses the ability of a system to recover from infrastructure or service disruptions, dynamically acquire computing resources to meet demand, and mitigate disruptions such as misconfigurations or transient network issues.”

– AWS Well Architected Framework Whitepaper

## Design Principles

- **Test recovery procedures**
- **Automatically recover from failures**
- Scale horizontally to increase aggregate system availability
- Stop guessing capacity
- Manage change using automation





# Poll

Have you ever experienced an availability loss in your system(s) due to infrastructure issues?

- a. Yes
- b. No
- c. Not sure

# Failure management

“Everything fails, all the time.” – Werner Vogel, Amazon CTO



We are going to work on withstanding COMPONENT failures and planning for RECOVERY



Mid-2000s: Amazon, Jesse Robbins starts GameDay

- Test, train & prepare Amazon systems, software, & people to respond to disaster
- Increase retail website resiliency by injecting failures into critical systems

2010: Netflix introduced Chaos Engineering – Simian Army

- e.g. Chaos Monkey, Chaos Gorilla and Chaos Kong.



“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.”

*<http://principlesofchaos.org>*

# Automated recovery

Manual recovery will take much longer



Use AWS services to automate

- Amazon Auto Scaling
- Amazon Relational Database Service (Amazon RDS)
- Amazon Route 53
- Amazon Simple Storage Service (Amazon S3)

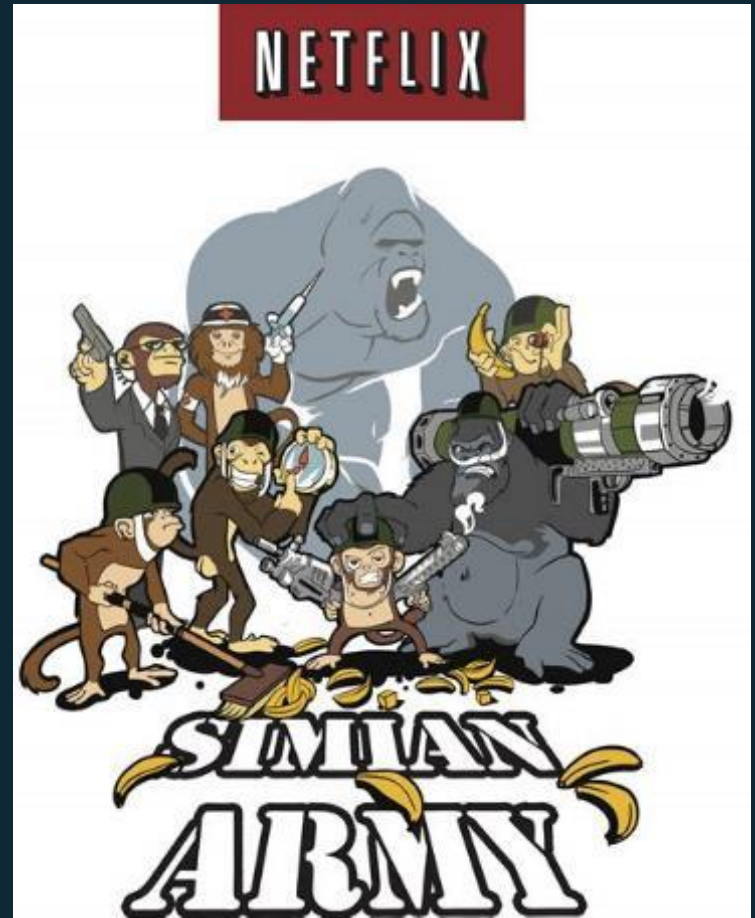


# Simian Army

Set of scheduled agents:

- shuts down services randomly
- slows down performances
- checks conformity
- breaks an entire region
- Integrates with spinnaker (CI/CD)

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



# The Chaos Toolkit

- Simplifying Adoption of Chaos Engineering
- An Open API to Chaos Engineering
- Open source extensions for
  - Infrastructure/Platform Fault Injections
  - Application Fault Injections
  - Observability
- Integrates easily into CI/CD pipelines

<https://github.com/chaostoolkit>

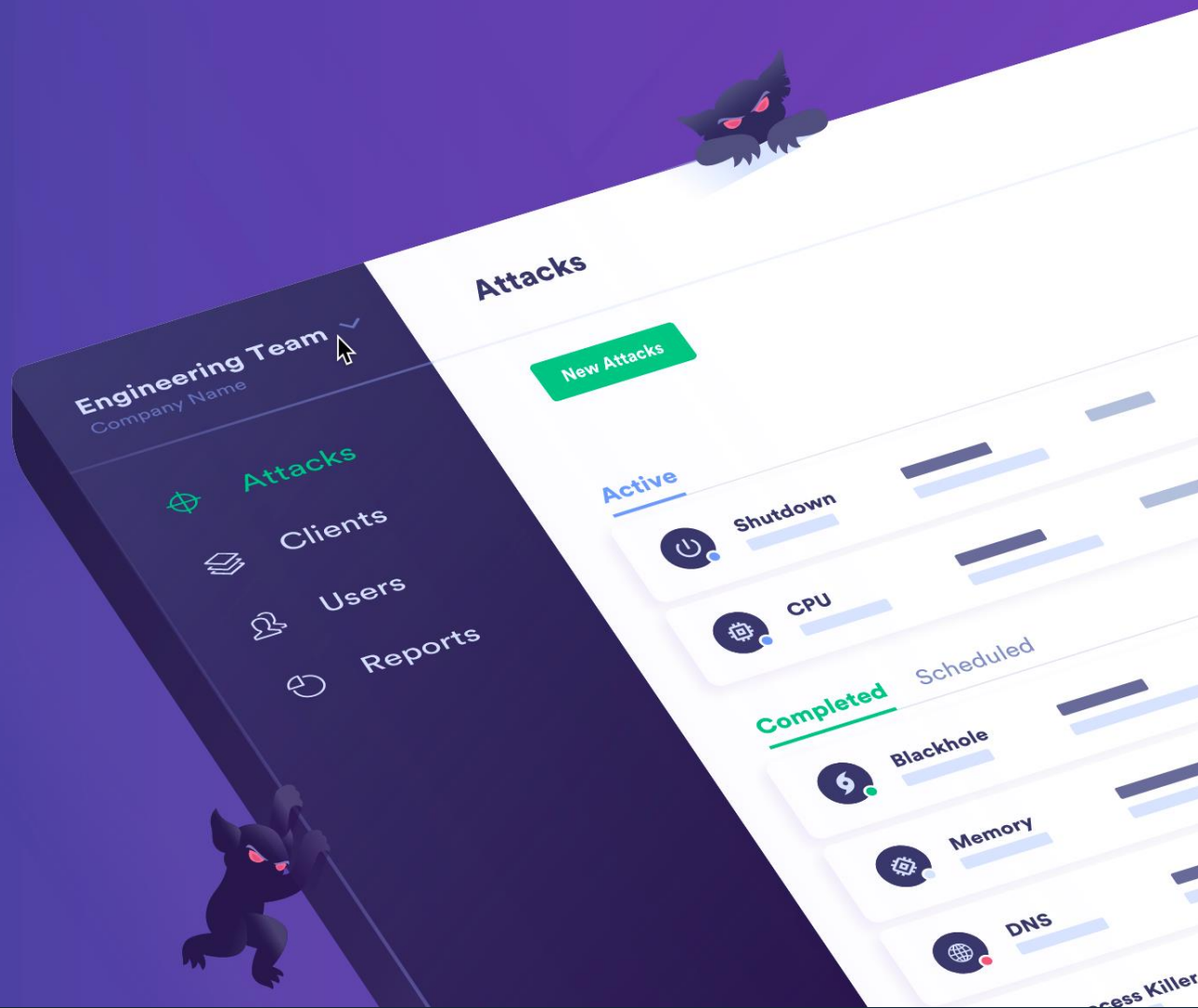
```
$ pip install -U chaostoolkit-aws
```





Gremlin

<https://www.gremlin.com/>



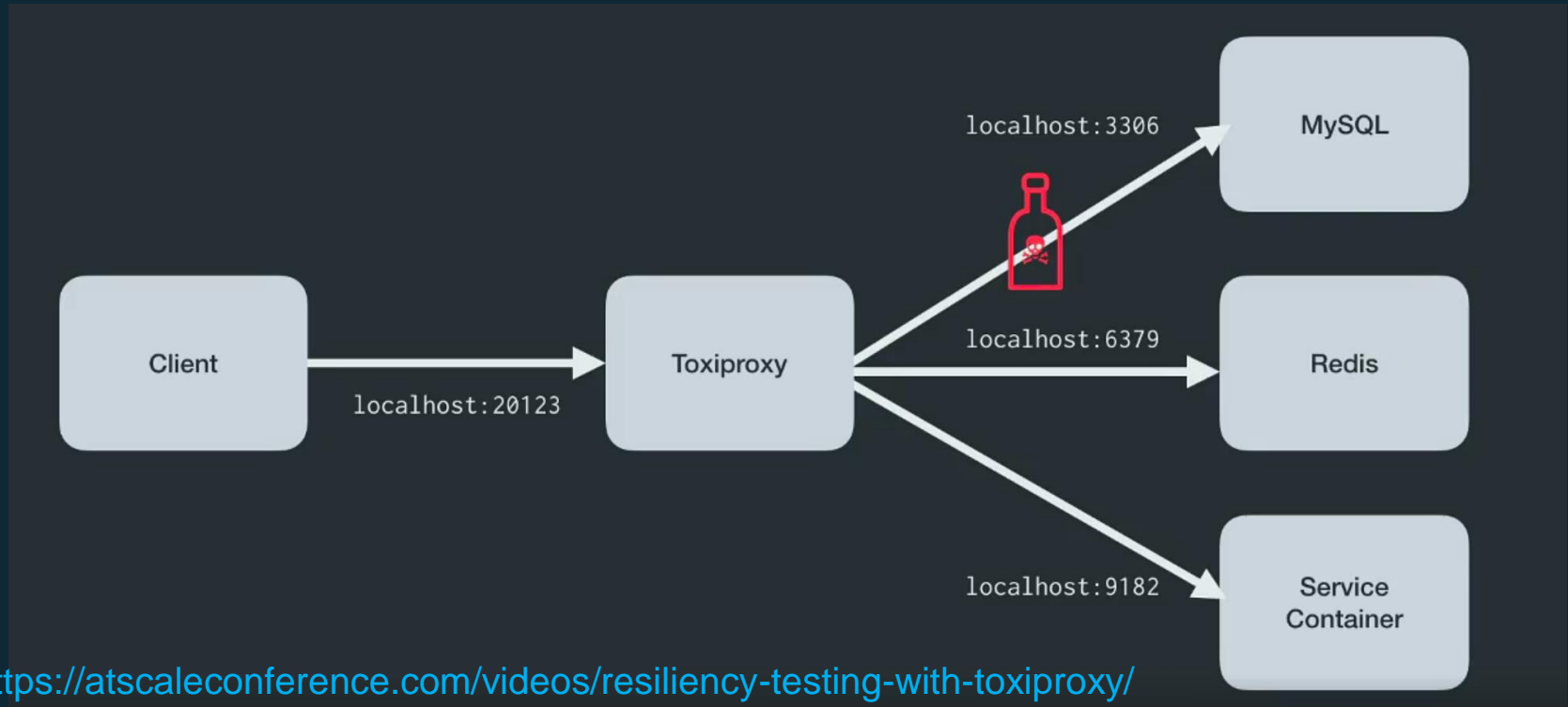
# ToxiProxy

- HTTP API
- Build for Automated testing in mind
- Not for production environment
- Fast
- Toxics for:
  - Timeouts, latency, connections and bandwidth limitation, etc..
- CLI
- Stable and well tested (used for 3 years at Shopify)





# Resiliency testing with Toxiproxy

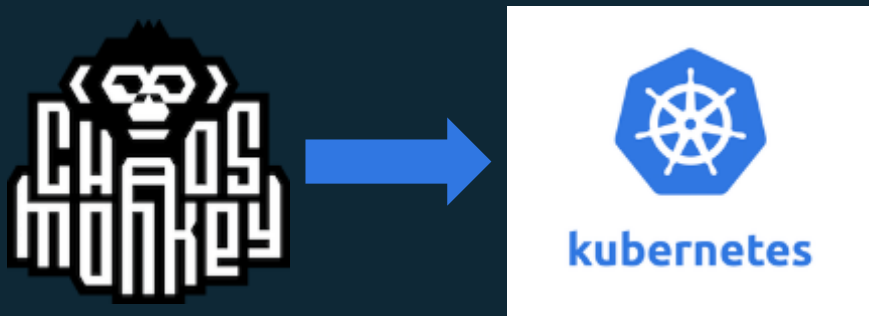


<https://atscaleconference.com/videos/resiliency-testing-with-toxiproxy/>

# Kube-monkey

## An implementation of Netflix's Chaos Monkey for Kubernetes cluster

- It randomly deletes Kubernetes (k8s) pods in the cluster encouraging and validating the development of failure-resilient services.



<https://github.com/asobti/kube-monkey>

# Pumba

Kill Container command

Network Emulation command

<https://github.com/alexei-led/pumba/>

## Pumba: Chaos testing tool for Docker

chat on gitter

Build Success

go report A+

codecov 48%

3.6MB

9 layers

version master-72...

commit not given

### Logo



PUMBA

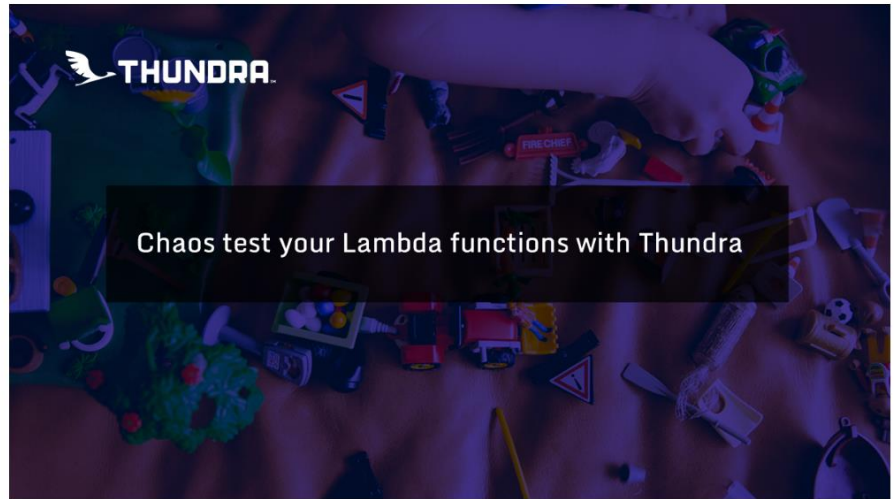
### Demo

```
gaia-mbp:pumba alexei$ docker run -it --rm --name tryme alpine sh -c "apk add --update iproute2 && ping www.example.com"
fetch http://dl-cdn.alpinelinux.org/alpine/v3.4/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.4/community/x86_64/APKINDEX.tar.gz
(1/4) Installing libmnl (1.0.3-r1)
(2/4) Installing libnftnl-libs (1.0.5-r0)
(3/4) Installing iptables (1.6.0-r0)
(4/4) Installing iproute2 (4.4.0-r1)
Executing iproute2-4.4.0-r1.post-install
Executing busybox-1.24.2-r9.trigger
OK: 8 MiB in 15 packages
PING www.example.com (93.184.216.34): 56 data bytes
64 bytes from 93.184.216.34: seq=0 ttl=127 time=253.156 ms
64 bytes from 93.184.216.34: seq=1 ttl=127 time=144.636 ms
64 bytes from 93.184.216.34: seq=2 ttl=127 time=246.227 ms
64 bytes from 93.184.216.34: seq=3 ttl=127 time=190.263 ms
64 bytes from 93.184.216.34: seq=4 ttl=127 time=150.392 ms
64 bytes from 93.184.216.34: seq=5 ttl=127 time=144.794 ms
64 bytes from 93.184.216.34: seq=6 ttl=127 time=144.233 ms
```

# Thundra



AWS Lambda



Failures are inevitable. Just as we need to test our application to find bugs in our business logic before they affect our users. We need to test our application against infrastructure failures. And we need to do it before they happen in production, and cause irreparable damages.

The discipline of **Chaos Engineering** shows us how to use controlled experiments to uncover these weaknesses.

# Poll

What are you doing now for Resilience Testing (Chaos Testing)?

- a. Using a framework like Gremlin or Chaos Monkey
- b. Using our home-grown scripts and programs
- c. Other
- d. None of the above

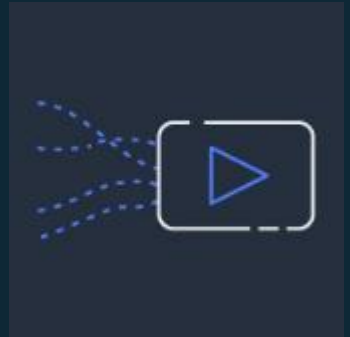
# Why are we here?

Just use Netflix OSS, Chaos Toolkit, Gremlin, etc?

- Chaos Monkey part of Spinnaker CI/CD?
- Share SSH keys to inject failure?
- Install external agent?
- Pay a license?
- In production?
- Can I do AZ failure?

Hard to understand how to start.

*Can I do this?*



# Yes, you can do this!

You can write code to perform these scenarios in any language

Bash shell, Python, Java, PowerShell, C#



You can write your own AZ Failure simulation



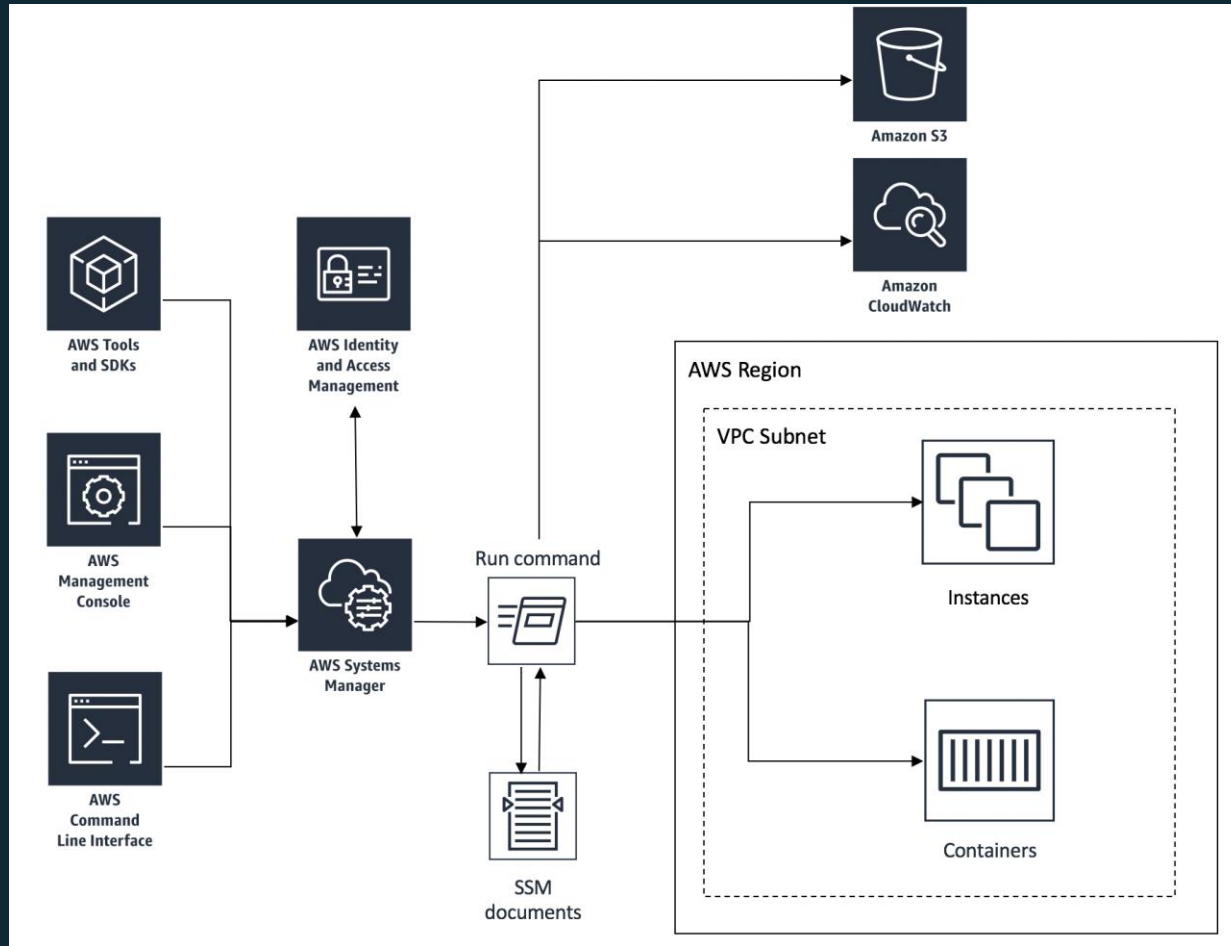
You can simulate your own regional failure

# Using AWS System Manager (SSM)

...to inject Chaos into Amazon EC2



AWS Systems Manager

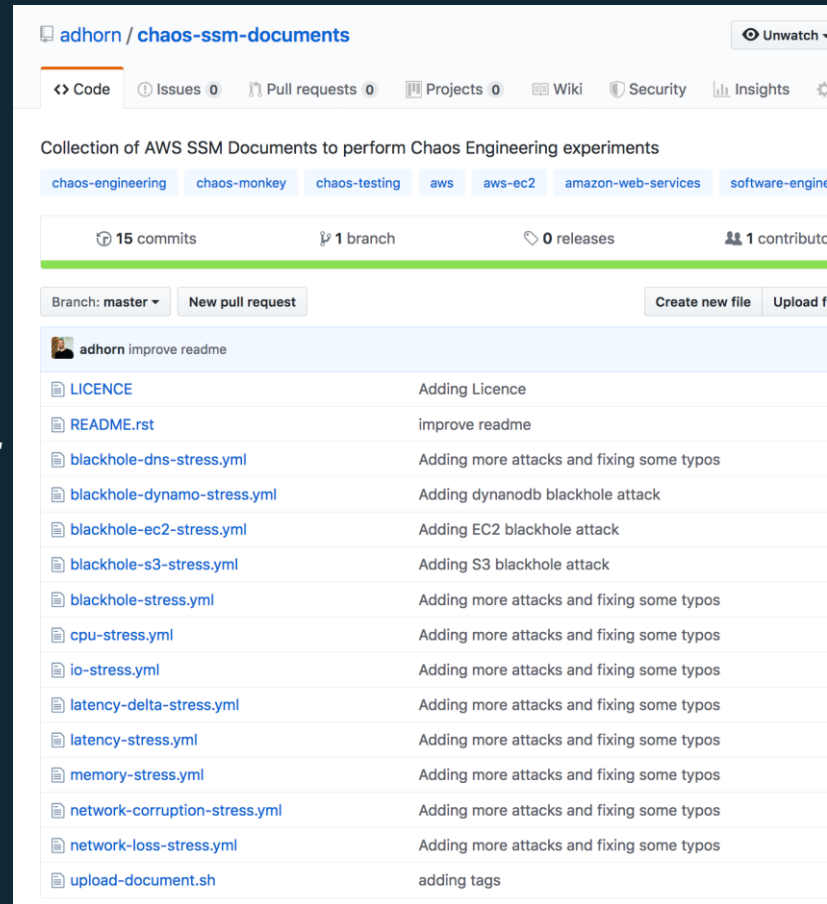




# SSM documents

```
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 }}
```

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



The screenshot shows the GitHub repository page for 'adhorn / chaos-ssm-documents'. The repository is described as a 'Collection of AWS SSM Documents to perform Chaos Engineering experiments'. It has 15 commits, 1 branch, 0 releases, and 1 contributor. The repository is on the 'master' branch. A table lists the files in the repository:

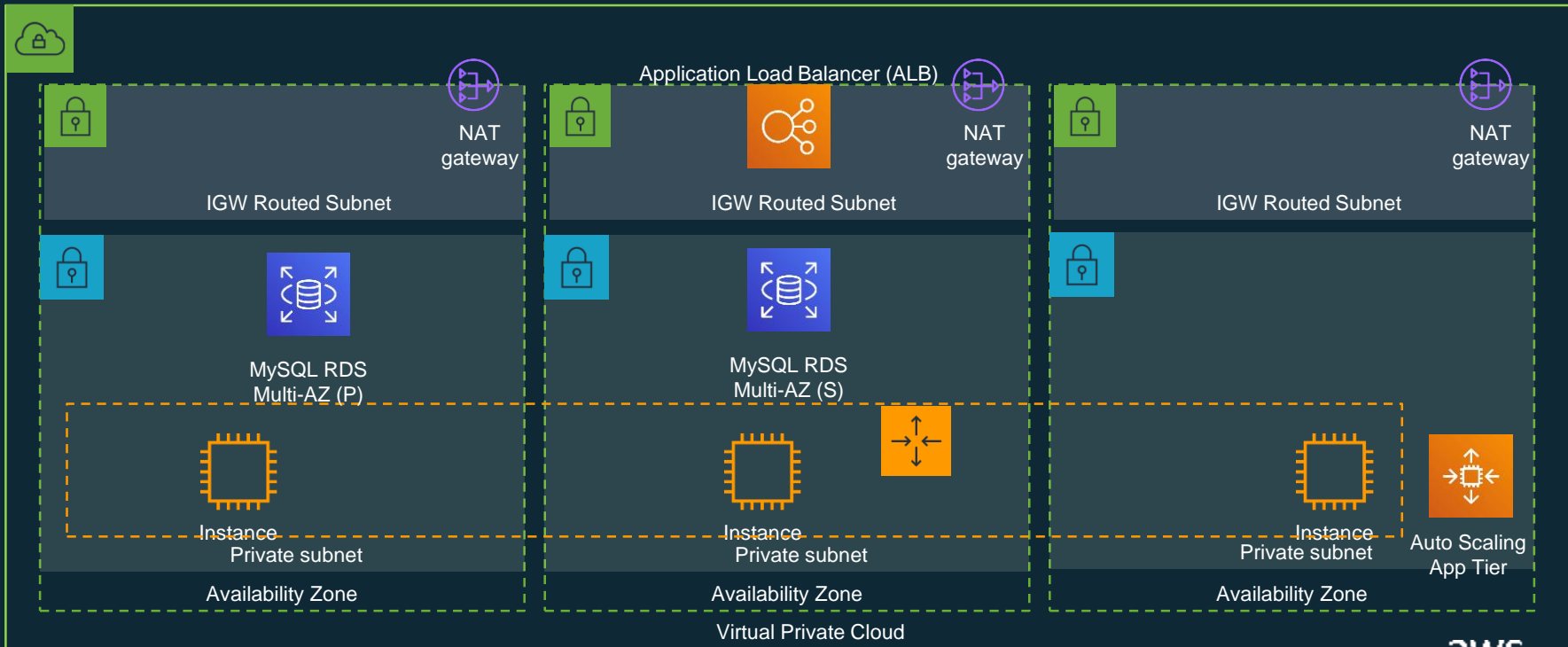
File Name	Description
LICENCE	Adding Licence
README.rst	improve readme
blackhole-dns-stress.yml	Adding more attacks and fixing some typos
blackhole-dynamo-stress.yml	Adding dynamodb blackhole attack
blackhole-ec2-stress.yml	Adding EC2 blackhole attack
blackhole-s3-stress.yml	Adding S3 blackhole attack
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



# Scenario Walkthrough

---

# Scenario architecture for testing



# Lab Time:

## Component and Availability Zone Failure Simulation

---

## Step 3. “Test Resiliency Using Failure Injection”

### 1. Deploy the Infrastructure

- When state machine has completed “WaitForWebApp” state...
- Then you can start testing!

WaitForWebApp

### 2. Configure Execution Environment

Complete Step 2 now if not done already

### 3. Test Resiliency Using Failure Injection

Decide which language you are comfortable using

- Bash
- PowerShell
- Python
- Java
- C#

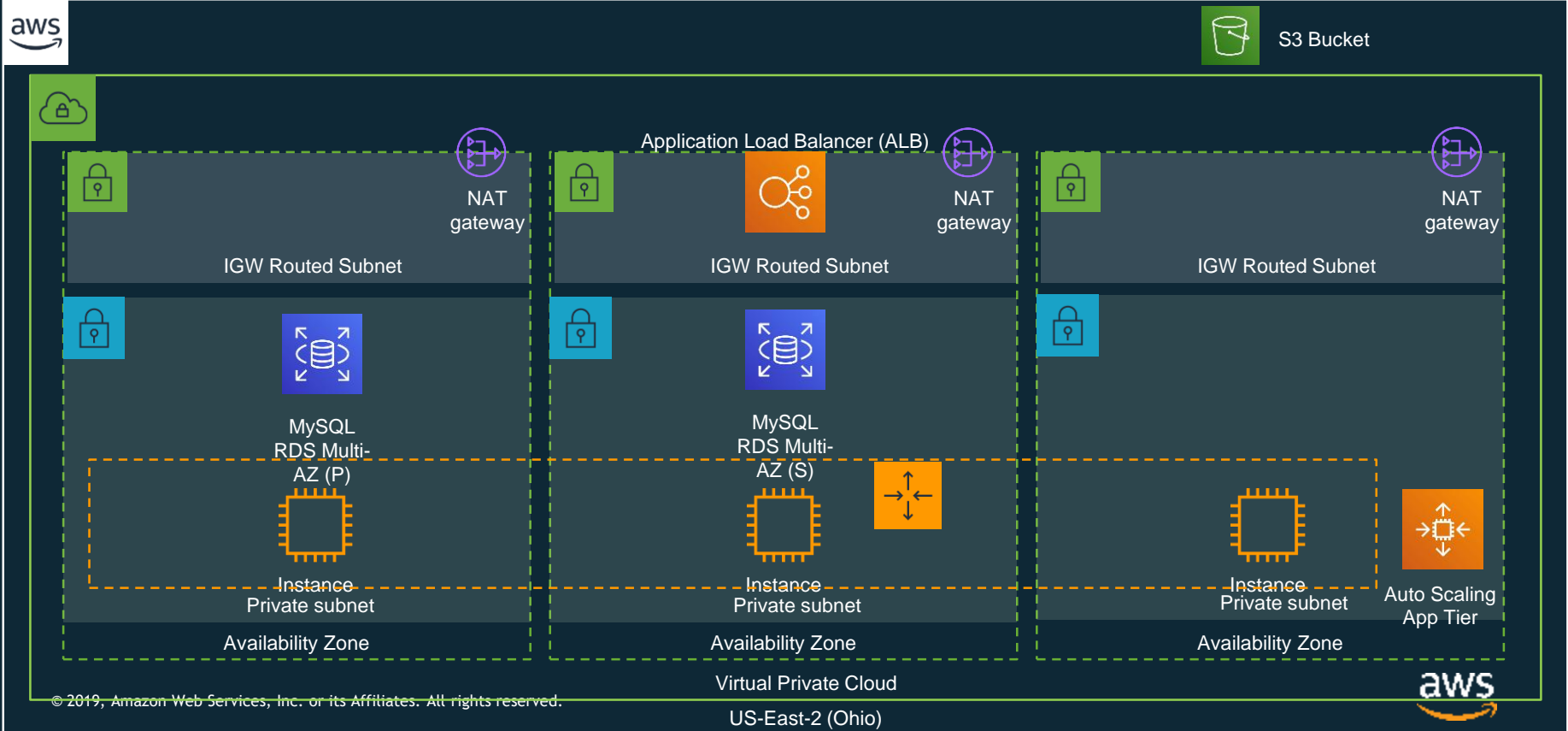
Ask for help if you need it



# Summary of Tests

---

# Simulated failures



# Poll

Are you confident that your system(s) meet your resiliency requirements?

- a. Yes
- b. No
- c. Not sure



# Best Practices

---

# What did you learn?

No fear – You can write these tests

The simulation may not be obvious

- Think about the effects before implementation
- Revise based on results
- Writing this code is not difficult

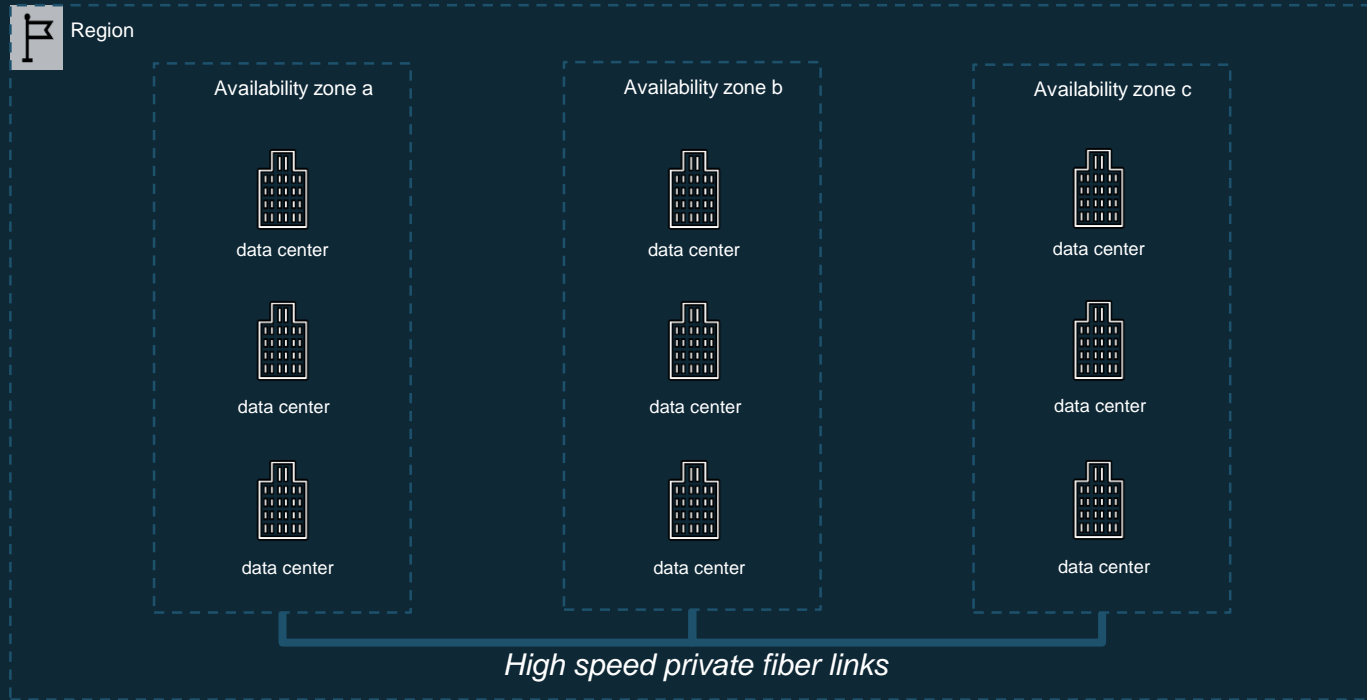
???



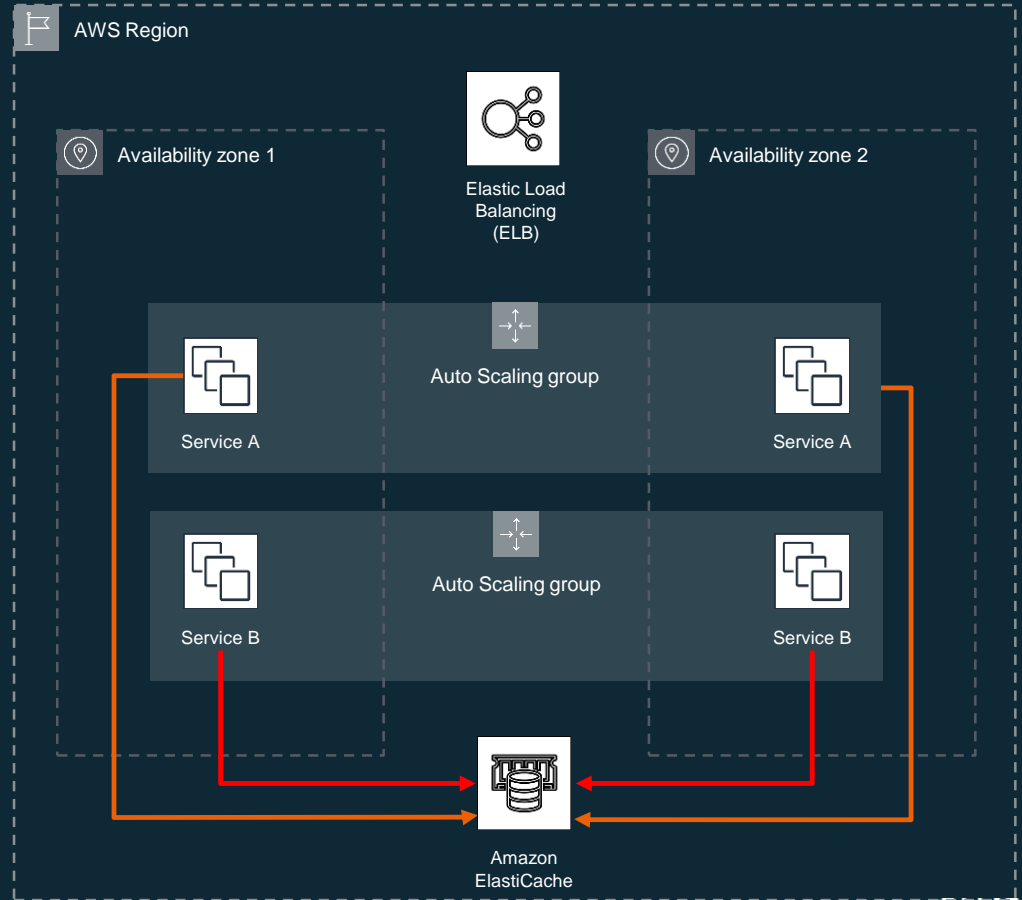
On very large implementations, you'll need to use orchestration to have things happen almost simultaneously



# AWS Region and availability zones



# Stateless Services



# Things to watch out for

You will find problems and resolve them, but real failure might not look like the simulation

- Expect this to be an ongoing effort
- Add these tests to your pipeline/acceptance testing

Some failure modes are destructive

- Automated deployment will help bring the environment back

Failover is easier than failback

- Transactions will be in flight
- Flapping can be worse than an outage

Game day exercises are essential

- Practice, practice, practice

# This code is available for download

Lab guide has download link

<http://bit.ly/WARel300>



awslabs/aws-well-archite...  
529 Stars · 191 Forks

It is licensed under the Apache License, Version 2.0 or MIT No Attribution License

- <https://aws.amazon.com/apache2.0>

# Thank you!

<https://aws.amazon.com/architecture/well-architected/>