

CHAOS ENGINEERING BOOTCAMP

Ana Medina
Chaos Engineer at Gremlin

SRECon EMEA - August 2018

CHAOS
ENGINEERING
BOOTCAMP

Ana Medina

Ana is currently working as a **Chaos Engineer** at Gremlin, helping companies avoid outages by running proactive chaos engineering experiments. She last worked at Uber where she was an engineer on the SRE and Infrastructure teams specifically focusing on chaos engineering and cloud computing.



@ana_m_medina

**CHAOS
ENGINEERING
BOOTCAMP**

Gremlin

Chaos Engineer /'kɑː, əs / ,enjə'nɪr/

noun

1. a person helping companies avoid outages by running proactive chaos engineering experiments.



gremlin.com

@gremlininc

**CHAOS
ENGINEERING
BOOTCAMP**

Ho Ming Li

Ho Ming Li is the **Lead Solutions Architect** at Gremlin. Prior to joining Gremlin, he worked at Amazon Web Services with many customers providing guidance around architectural and operational best practices. He takes a strategic approach to deliver holistic solutions, often diving into the intersection of people, process, business, and technology. His goal is to enable everyone to build more resilient software by means of Chaos Engineering practices.



@HoReal

**CHAOS
ENGINEERING
BOOTCAMP**

Join Slack



www.gremlin.com/slack

[#srecon18_europe](#)

**CHAOS
ENGINEERING
BOOTCAMP**

Agenda:

14:00 - 15:00



Foundation of Chaos Engineering



Breaking Things

15:00 - 15:30



Chaos Engineering Discussion

15:30 - 16:00



Break

16:00 - 16:30



Distributed Systems Chaos

16:30 - 16:45



Crafting your own experiment

16:45 - 17:00



Starting at your company

17:00 - 17:15



Advanced Chaos

17:15 - 17:30



Q&A

Part :

Foundations of Chaos Engineering

Chaos Engineering?

Thoughtful, planned experiments designed to reveal the weakness in our systems.

Like a vaccine, we inject harm
to build *immunity*.



What Chaos Engineering is

- Thoughtful chaos engineering experiments
- Controlled and planned chaos engineering experiments
- Preparing for unpredictable failure
- Preparing engineers for failure
- Preparing for GameDay
- A way to improve SLA
 - fortify systems
 - build and move fast
 - build confidence in systems
 - reveal weak points in your systems
 - build assurance that you can still serve your customers

What Chaos Engineering is not

- Random chaos engineering experiments
- Unsupervised chaos engineering experiments
- Unmonitored chaos engineering experiments
- Unexpected chaos engineering experiments
- Breaking production by accident
- Creating outages

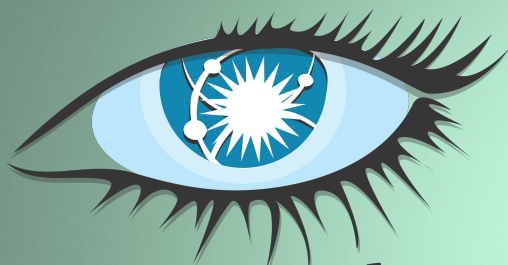
Why do Chaos Engineering

- Microservice Architecture is tricky
- Our systems are scaling fast
- Services will fail
- Dependencies on other companies will fail
- Prepares for real world scenarios
- Reduce amount of outages, reduce down time, lose less money

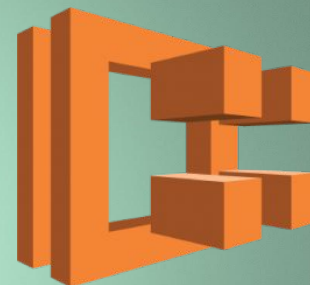
Inject Failure at any level

- Application
- API
- Caching
- Database
- Hardware
- Cloud Infrastructure / Bare metal

Top places to inject chaos



cassandra



Amazon ECS



kafka



elasticsearch

**CHAOS
ENGINEERING
BOOTCAMP**

Companies doing Chaos Engineering



What do you need before doing Chaos Engineering?

- Monitoring / Observability
- On Call / Incident Management
- Alerting and paging
- Clear instructions on how to roll back an experiment
- The cost of downtime per hour

Monitoring and Measuring

- System Metrics: CPU, Disk, I/O
- Availability
- Service specific KPIs
- Customer complaints

Some Monitoring / Observability tools to use



Part :

Breaking Things



Form a hypothesis.



Run an experiment.



Abort Conditions



Failure



Success



Find and fix issues.



Scale up and repeat.

Minimize the
blast radius.

Please get in groups of 3

Getting Access to your host:

While we wait, check out this survey:

bit.ly/ce-questions

```
1. bash

ssh into your hosts as root

$ ssh root@207.154.216.247

Password: chaosbootcamp
```

```
1. bash

Install kubernetes on all hosts

apt-get update && apt-get install -y apt-transport-https
curl -s
https://packages.cloud.google.com/apt/doc/apt-key.gp
g | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF
apt-get update
apt-get install -y kubelet kubeadm kubectl docker.io
```



```
1. bash

choose host that will be master

$ su -l kube
Password: chaosbootcamp

initialize kubernetes master node
$ sudo kubeadm init

start cluster on master
$ mkdir -p $HOME/.kube
$ sudo cp -i /etc/kubernetes/admin.conf
$HOME/.kube/config
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

```
1. bash

on the other two hosts, join master cluster as root
$ kubectl join --token 702ff6.bc7aacff7aacab17
174.138.15.158:6443 --discovery-token-ca-cert-hash
sha256:68bc22d2c631800fd358a6d7e3998e598deb29
80ee613b3c2f1da8978960c8ab

on master, verify nodes have joined master
$ sudo kubectl get nodes
```

**cool! you've gotten
Kubernetes set up,
lets setup the rest**

```
1. bash

on master setup a kubernetes add-on for networking
features and policy - Weave Net
$ curl -o weave.yaml
https://cloud.weave.works/k8s/v1.8/net.yaml
$ cat weave.yaml
$ kubectl apply -f weave.yaml

check the status of the containers, they should be
running
$ kubectl get pods --all-namespaces
```

```
1. bash

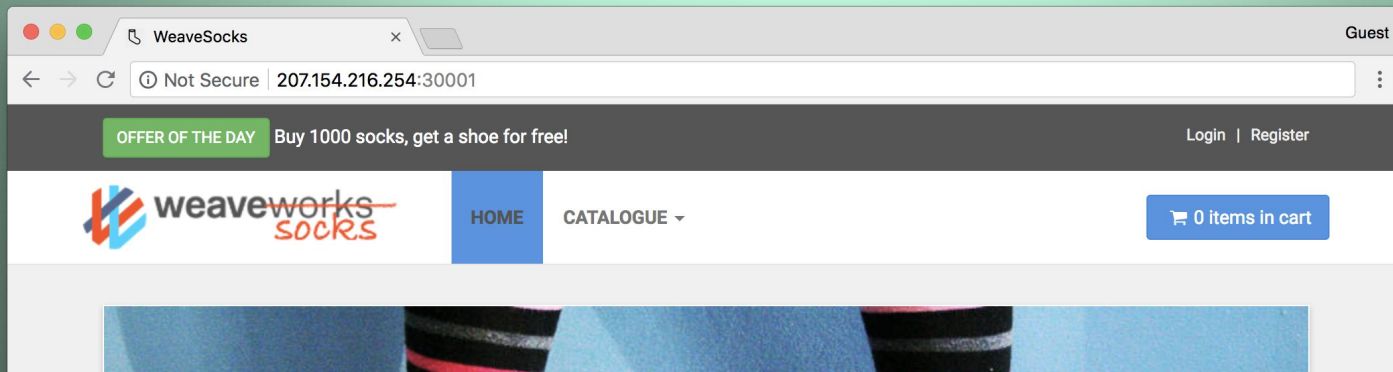
still on master, grab demo microservices sock shop
$ git clone
https://github.com/microservices-demo/microservices-
demo.git
$ cd microservices-demo/deploy/kubernetes/

create namespace
$ kubectl create namespace sock-shop

apply demo to the cluster
$ kubectl apply -f complete-demo.yaml

check if pods are running
$ kubectl get pods --namespace sock-shop
```

**hooray! things are running.
visit your ip address on port
30001**



**CHAOS
ENGINEERING
BOOTCAMP**

```
1. bash

On every host as root, lets get some monitoring in place
with Datadog
$ DD_API_KEY=faff9c88d8cdd357d76505f595f23797
bash -c "$(curl -L
https://raw.githubusercontent.com/DataDog/datadog-agent/master/cmd/agent/install_script.sh)"

on every host, let's grab the bootcamp files
$ su - chaos
$ git clone
https://github.com/tammybutow/chaosengineeringbootcamp
```

let the chaos begin
hello world, the chaos way

CHAOS
ENGINEERING
BOOTCAMP

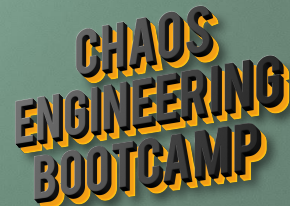

```
1. bash

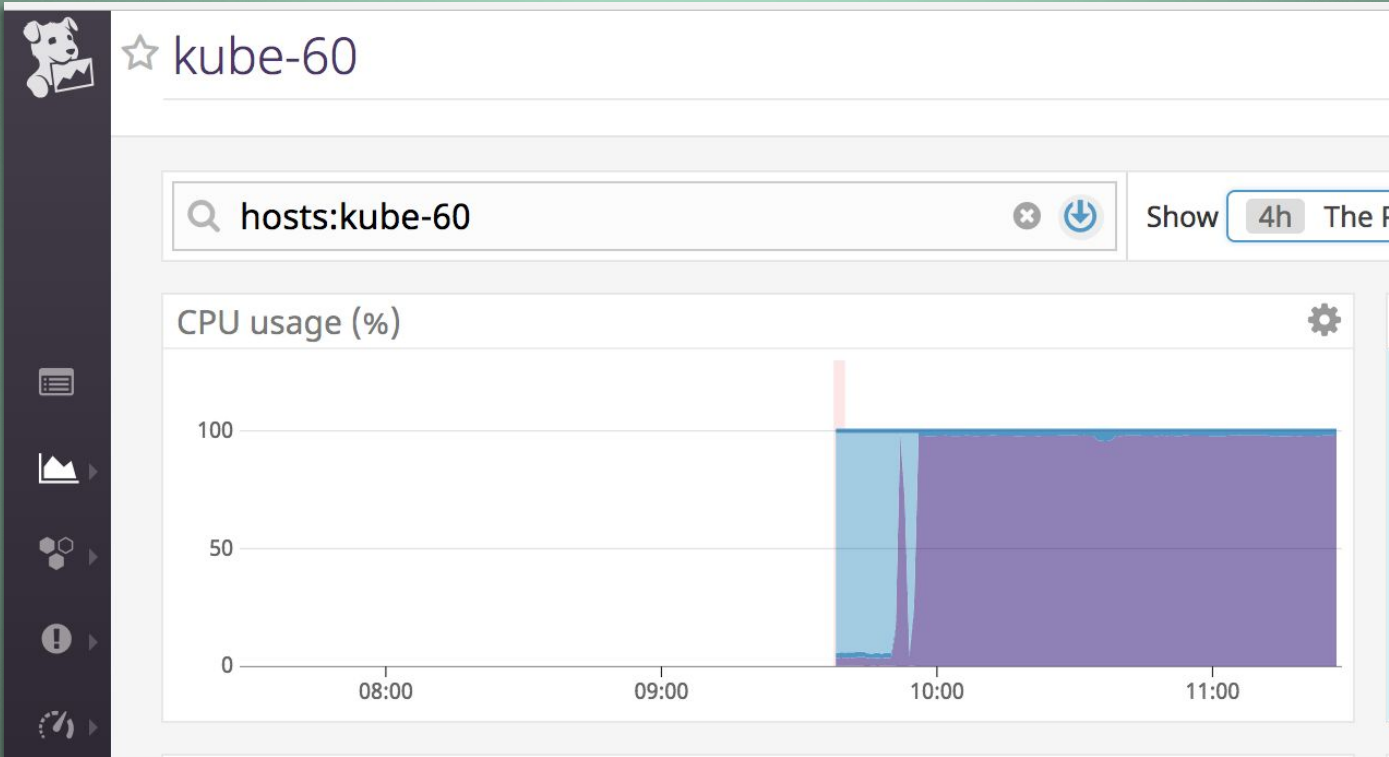
as chaos user on every host
$ cd chaosengineeringbootcamp
$ ls
$ chmod +x chaos_cpu.sh
$ ./chaos_cpu.sh

you can exit using control + c
let's see what we did
$ top
```

```
2.chaos@kube-60: ~/chaosengineeringbootcamp (ssh)
top - 09:26:17 up 1 day, 15:36, 1 user, load average: 32.70, 32.64, 32.54
Tasks: 225 total, 33 running, 192 sleeping, 0 stopped, 0 zombie
%Cpu(s): 99.0 us, 1.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 4046404 total, 298324 free, 705036 used, 3043044 buff/cache
KiB Swap: 0 total, 0 free, 0 used, 2963556 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM     TIME+ COMMAND
 27805 chaos    20   0 13932 3520 3084  R   6.6   0.1   0:24.07 openssl
 27878 chaos    20   0 13932 3500 3072  R   6.6   0.1   0:24.00 openssl
 27879 chaos    20   0 13932 3552 3124  R   6.6   0.1   0:24.49 openssl
 27793 chaos    20   0 13932 3504 3072  R   6.3   0.1   0:24.14 openssl
 27795 chaos    20   0 13932 3592 3160  R   6.3   0.1   0:24.16 openssl
 27799 chaos    20   0 13932 3568 3136  R   6.3   0.1   0:24.30 openssl
 27811 chaos    20   0 13932 3616 3184  R   6.3   0.1   0:24.14 openssl
 27813 chaos    20   0 13932 3460 3032  R   6.3   0.1   0:24.38 openssl
 27814 chaos    20   0 13932 3552 3124  R   6.3   0.1   0:24.07 openssl
 27883 chaos    20   0 13932 3600 3168  R   6.3   0.1   0:24.42 openssl
 27796 chaos    20   0 13932 3532 3104  R   5.9   0.1   0:24.35 openssl
 27797 chaos    20   0 13932 3512 3080  R   5.9   0.1   0:24.30 openssl
 27798 chaos    20   0 13932 3464 3032  R   5.9   0.1   0:24.71 openssl
 27800 chaos    20   0 13932 3504 3072  R   5.9   0.1   0:24.32 openssl
 27801 chaos    20   0 13932 3484 3052  R   5.9   0.1   0:24.44 openssl
 27802 chaos    20   0 13932 3480 3052  R   5.9   0.1   0:24.22 openssl
 27804 chaos    20   0 13932 3524 3084  R   5.9   0.1   0:24.06 openssl
 27808 chaos    20   0 13932 3552 3124  R   5.9   0.1   0:24.04 openssl
 27809 chaos    20   0 13932 3524 3092  R   5.9   0.1   0:24.04 openssl
 27877 chaos    20   0 13932 3536 3104  R   5.9   0.1   0:24.09 openssl
 27885 chaos    20   0 13932 3520 3084  R   5.9   0.1   0:23.84 openssl
 27803 chaos    20   0 13932 3556 3124  R   5.6   0.1   0:24.51 openssl
 27806 chaos    20   0 13932 3568 3136  R   5.6   0.1   0:24.32 openssl
 27807 chaos    20   0 13932 3524 3092  R   5.6   0.1   0:24.59 openssl
 27810 chaos    20   0 13932 3592 3160  R   5.6   0.1   0:24.30 openssl
 27815 chaos    20   0 13932 3536 3104  R   5.6   0.1   0:23.91 openssl
 27880 chaos    20   0 13932 3616 3184  R   5.6   0.1   0:24.57 openssl
 27881 chaos    20   0 13932 3532 3104  R   5.6   0.1   0:24.44 openssl
 27882 chaos    20   0 13932 3568 3136  R   5.6   0.1   0:24.23 openssl
 27794 chaos    20   0 13932 3484 3052  R   5.3   0.1   0:24.27 openssl
 27812 chaos    20   0 13932 3524 3092  R   5.3   0.1   0:24.34 openssl
 27884 chaos    20   0 13932 3520 3084  R   5.3   0.1   0:23.86 openssl
 25984 root      20   0 559292 98492 54292  S   1.7   2.4   4:14.49 kubelet
 26310 root      20   0 423732 326196 61644  S   1.7   8.1   3:59.80 kube-apiserver
 26302 root      20   0 152292 94120 49908  S   1.3   2.3   3:07.64 kube-controller
 26363 root      20  10 073g 66044 18360  S   1.0   1.6   2:00.12 etcd
 26370 root      20   0 49056 34352 23292  S   0.7   0.8   1:06.66 kube-scheduler
 24749 root      20   0 1276176 66608 33984  S   0.3   1.6   1:28.01 dockerd
 24758 root      20   0 434144 15344  9604  S   0.3   0.4   0:10.96 docker-containe
31323 chaos    20   0 40692 3908 3184  R   0.3   0.1   0:00.10 top
 1 root      20   0 37900 5864 3912  S   0.0   0.1   0:05.92 systemd
 2 root      20   0 0 0 0  S   0.0   0.0   0:00.00 kthreadd
 3 root      20   0 0 0 0  S   0.0   0.0   0:00.96 ksoftirqd/0
 5 root     0 20 0 0 0  S   0.0   0.0   0:00.00 kworker/0:0H
 7 root      20   0 0 0 0  S   0.0   0.0   0:05.84 rcu_sched
 8 root      20   0 0 0 0  S   0.0   0.0   0:00.00 rcu_bh
 9 root      rt  0 0 0 0  S   0.0   0.0   0:00.19 migration/0
10 root      rt  0 0 0 0  S   0.0   0.0   0:00.69 watchdog/0
```



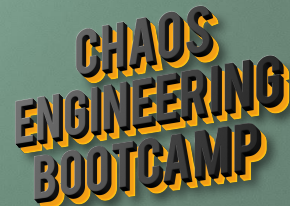


```
1. bash

lets stop the known chaos
$ pkill -u chaos
$ top
```

```
2.chaos@kube-60: ~/chaosengineeringbootcamp (ssh)
top - 09:34:24 up 1 day, 15:44, 1 user, load average: 5.29, 22.35, 28.84
Tasks: 161 total, 1 running, 160 sleeping, 0 stopped, 0 zombie
%Cpu(s): 4.3 us, 2.0 sy, 0.0 ni, 93.4 id, 0.2 wa, 0.0 hi, 0.2 si, 0.0 st
KiB Mem : 4046404 total, 327204 free, 679024 used, 3040176 buff/cache
KiB Swap: 0 total, 0 free, 0 used, 2990852 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
26310 root        20   0 423732 327108 61644 S   3.7   8.1   4:12.59 kube-apiserver
25984 root        20   0 559292 98492 54292 S   3.0   2.4   4:29.01 kubelet
26302 root        20   0 152292 94032 49908 S   2.7   2.3   3:18.18 kube-controller
26363 root        20   0 10.073g 68668 18360 S   2.0   1.7   2:06.73 atcd
 6001 dd-agent    20   0 912612 60124 32760 S   1.3   1.5   0:22.57 agent
26370 root        20   0 49056 34328 23292 S   1.0   0.8   1:10.51 kube-scheduler
24749 root        20   0 1276176 66608 33984 S   0.7   1.6   1:31.05 dockerd
  614 root        20   0 40732 6080 3640 S   0.3   0.2   0:09.90 systemd-journal
1354 syslog     20   0 256392 3600 2768 S   0.3   0.1   0:02.86 rsyslogd
2410 chaos     20   0 40520 3616 2984 R   0.3   0.1   0:00.21 top
26620 root        20   0 43824 30384 22208 S   0.3   0.8   0:22.10 kube-proxy
  1 root        20   0 37900 5864 3912 S   0.0   0.1   0:05.95 systemd
  2 root        20   0 0 0 0 S   0.0   0.0   0:00.00 kthreadd
  3 root        20   0 0 0 0 S   0.0   0.0   0:01.00 ksoftirqd/0
  5 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 kworker/0:0H
  7 root        20   0 0 0 0 S   0.0   0.0   0:06.10 rcu_sched
  8 root        20   0 0 0 0 S   0.0   0.0   0:00.00 rcu_bh
  9 root        rt   0 0 0 0 S   0.0   0.0   0:00.20 migration/0
10 root        rt   0 0 0 0 S   0.0   0.0   0:00.69 watchdog/0
11 root        rt   0 0 0 0 S   0.0   0.0   0:00.80 watchdog/1
12 root        rt   0 0 0 0 S   0.0   0.0   0:00.16 migration/1
13 root        20   0 0 0 0 S   0.0   0.0   0:01.21 ksoftirqd/1
15 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 kworker/1:0H
16 root        20   0 0 0 0 S   0.0   0.0   0:00.00 kdevtmpfs
17 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 netns
18 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 perf
19 root        20   0 0 0 0 S   0.0   0.0   0:00.05 khungtaskd
20 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 writeback
21 root        25   5 0 0 0 S   0.0   0.0   0:00.00 ksmd
22 root        39  19 0 0 0 S   0.0   0.0   0:00.03 khugepaged
23 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 crypto
24 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 kintegrityd
25 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 bioset
26 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 kblockd
27 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 ata_sff
28 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 md
29 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 devfreq_wq
34 root        20   0 0 0 0 S   0.0   0.0   0:00.00 kswapd0
35 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 vmstat
36 root        20   0 0 0 0 S   0.0   0.0   0:00.00 fsnotify_mark
37 root        20   0 0 0 0 S   0.0   0.0   0:00.00 ecryptfs-kthrea
53 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 kthrotld
54 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 acpi_thermal_pm
55 root        20   0 0 0 0 S   0.0   0.0   0:00.00 yballoon
56 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 bioset
57 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 bioset
58 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 bioset
59 root        0 -20 0 0 0 S   0.0   0.0   0:00.00 bioset
```



Part :

Chaos Engineering Discussion

4 volunteers needed for 2 teams

Every company should be doing chaos engineering.

- 2 minutes for brainstorming
- 2 minutes per speaker
- Winning team chosen via applause

**CHAOS
ENGINEERING
BOOTCAMP**

Agenda:

14:00 - 15:00



Foundation of Chaos Engineering



Breaking Things

15:00 - 15:30



Chaos Engineering Discussion

15:30 - 16:00



Break

16:00 - 16:30



Distributed Systems Chaos

16:30 - 16:45



Crafting your own experiment

16:45 - 17:00



Starting at your company

17:00 - 17:15



Advanced Chaos

17:15 - 17:30



Q&A

Part :

Distributed Systems Chaos

Chaos Monkey

Chaos Monkey is a resiliency tool that helps applications tolerate random instance failures.

github.com/Netflix/chaosmonkey



Simian Army

Tools for keeping your cloud operating in top form. Chaos Monkey is a resiliency tool that helps applications tolerate random instance failures.

github.com/Netflix/SimianArmy



Kube Monkey

An implementation of Netflix's Chaos Monkey for Kubernetes clusters

github.com/asobti/kube-monkey

Pumba

Chaos testing and network emulation tool for Docker.

github.com/alexei-led/pumba



Powerfulseal

A powerful testing tool for Kubernetes clusters.

github.com/bloomberg/powerfulseal

Litmus

Litmus is chaos engineering for stateful workloads on Kubernetes, hopefully without learning curves

github.com/openenbs/litmus

Gremlin

Failure as a Service.

Finds weaknesses in your system before they cause problems.

Run Gremlin Agents on Hosts or Containers. Schedule attacks using UI, API and CLI. Provides 11 attacks out of the box

gremlin.com



Why do Chaos Engineering?

prevent outages

W Northeast blackout of 2003 - 1 x Guest

Secure https://en.wikipedia.org/wiki/Northeast_blackout_of_2003

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA

The Free Encyclopedia

- Main page
- Contents
- Featured content
- Current events
- Random article
- Donate to Wikipedia
- Wikipedia store

Interaction

Help

- About Wikipedia
- Community portal
- Recent changes
- Contact page

Tools

- What links here
- Related changes
- Upload file
- Special pages
- Permanent link
- Page information
- Wikidata item
- Cite this page

Print/export

- Create a book
- Download as PDF
- Printable version

Northeast blackout of 2003

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help **improve it** or discuss these issues on the **talk page**. *(Learn how and when to remove these template messages)*

- This article **has an unclear citation style**. *(May 2013)*
- This article **needs additional citations for verification**. *(July 2008)*

The **Northeast blackout of 2003** was a widespread **power outage** throughout parts of the **Northeastern** and **Midwestern United States** and the Canadian province of **Ontario** on August 14, 2003, just after 4:10 p.m. EDT.^[1]

Some power was restored by 11 p.m. Most did not get their power back until two days later. In other areas, it took nearly a week or two for power to be restored.^[2] At the time, it was the world's second **most widespread blackout in history**, after the **1999 Southern Brazil blackout**.^{[3][4]} The outage, which was much more widespread than the **Northeast Blackout of 1965**, affected an estimated 10 million people in Ontario and 45 million people in eight U.S. states.

The blackout's primary cause was a **software bug** in the alarm system at the control room of **FirstEnergy Corporation**, an **Akron, Ohio**-based company, causing operators to remain unaware of the need to re-distribute load after overloaded transmission lines drooped into foliage. What should have been a manageable local blackout cascaded into collapse of the entire electric grid.

Contents [hide]

- Immediate impact
 - Unaffected regions
- Causes
 - Background



This image shows states and provinces that experienced **power outages**. Not all areas within these political boundaries were affected.



Northeast blackout of 2003

System operators were unaware of the malfunction. The failure deprived them of both audio and visual alerts for important changes in system state. Race condition triggered in the control software.

**Chas Engineering helps you test
monitoring tools, metrics,
dashboards, alerts, and thresholds.**

Learn about different outages:

github.com/danluu/post-mortems

**Injecting Chaos is a controlled way
will lead to engineers building
resilient systems.**

Outage post-mortem | Dropbo x Guest
Secure | <https://blogs.dropbox.com/tech/2014/01/outage-post-mortem/>



Topics ▾ Subscribe ▾ Dropbox blogs ▾ 🔍

Outage post-mortem

Akhil Gupta | January 12, 2014

0 0

On Friday evening our service went down during scheduled maintenance. The service was back up and running about three hours later, with core service fully restored by 4:40 PM PT on Sunday.

For the past couple of days, we've been working around the clock to restore full access as soon as possible. Though we've shared some brief updates along the way, we owe you a detailed explanation of what happened and



Some master-replica pairs were impacted which resulted in site going down

<https://blogs.dropbox.com/tech/2014/01/outage-post-mortem/>

The screenshot shows a web browser window with the following elements:

- Browser tab: "Scaling Uber with Matt Ranney x"
- Address bar: "Secure https://softwareengineeringdaily.com/2015/12/04/engineering-at-uber-with-matt-ranney/"
- Page header: "SOFTWARE ENGINEERING DAILY" with the tagline "The World Through the Lens of Software". A search bar with "Search..." and a magnifying glass icon is on the right.
- Navigation menu: "All Content", "Cloud Engineering", "Business and Philosophy", "Greatest Hits", "Hackers", "Data".
- Article title: "Scaling Uber with Matt Ranney" with social media icons for Facebook, Twitter, and LinkedIn.
- Author: "By Pranay" with a profile picture icon.
- Metadata: "Podcast | Friday, December 4 2015".
- Player: A video player with a play button, a progress bar at "00:00", and a volume icon.



Uber's Database Outage

- Master log replication to S3 failed
- Logs backed up on primary
- Alerts were fired and ignored
- Disk fills up on database primary
- Engineer deletes unarchived WAL files
- Error in config prevents promotion

Part :

Crafting your own experiment

Don't approach it with a random strategy, instead approach it like a scientific experiment, thoughtful and planned.

Crafting your own experiment

Brainstorming: What should we break?

1. Form a hypothesis - What could go wrong?
2. Plan your experiment
3. Minimize Blast Radius - Small experiments first
4. Run experiment
5. Observe results
 - if things broke -> go fix it.
 - If things went as planned, increase blast radius and go back to step #1

Part :

Starting at your company

**Are you confident about your
metrics and alerting?**

**Are you confident your customers
are getting as good as an experience
as they should be?**

**Are you losing money due to
downtime and broken features?**

Getting Started:

1. Identify top 5 critical systems
2. Choose system
3. Whiteboard the system
4. Determine what experiment you want to run: (resource, state, network)
5. Determine Blast Radius

GameDay

Chose a system / application

Allocate 2-4 hours

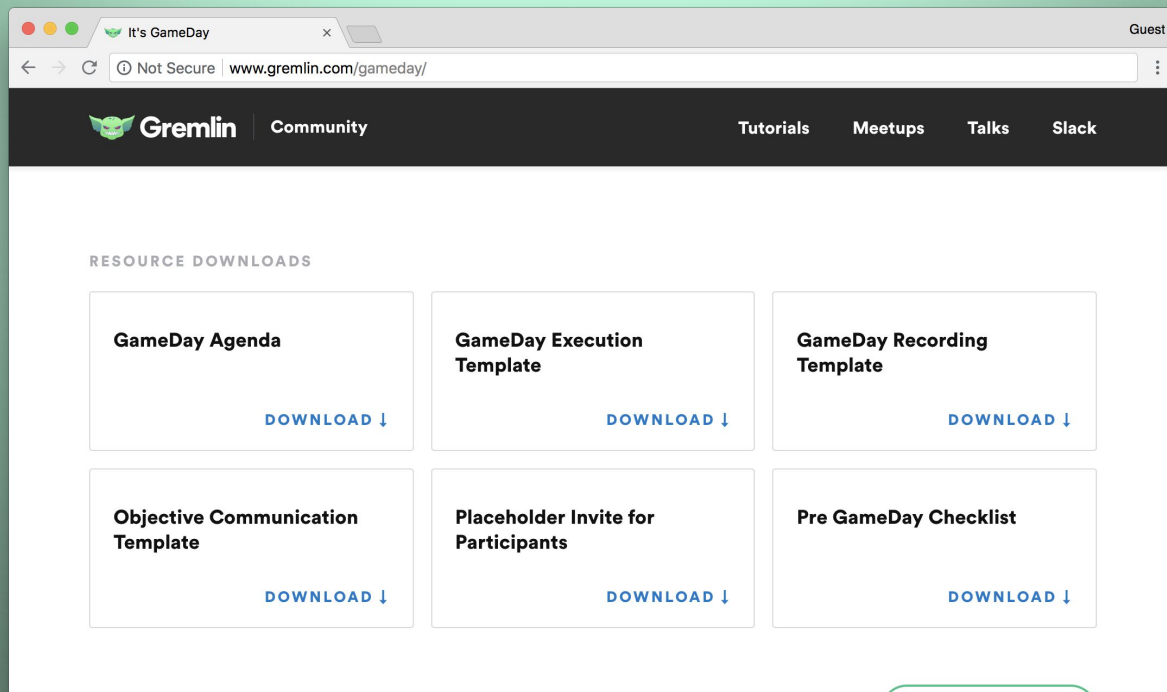
Room filled with the engineers that developed / support application

GameDay

“What could go wrong?”

“Do we know what will happen if
this breaks?”

gremlin.com/gameday



**CHAOS
ENGINEERING
BOOTCAMP**

Failure Fridays

Dogfooding.

We use Gremlin for Chaos Engineering Experiments

Dedicated time to practice Chaos Engineering to reveal weaknesses in our services



Part 7 :

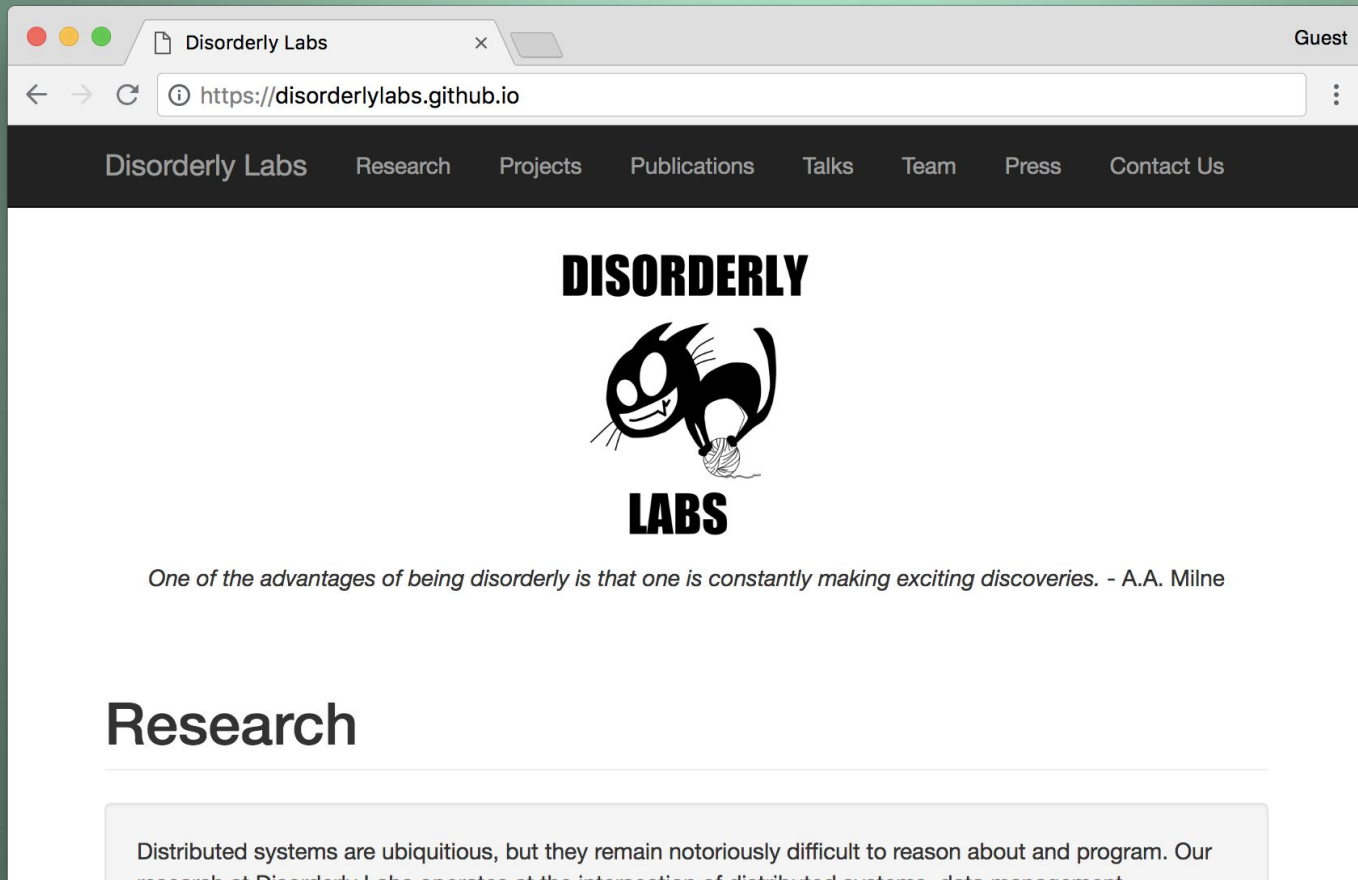
Advanced Chaos

Other Uses for Chaos Engineering

Preparing for IPO / Auditing

Training new engineers / on-call practicing

Testing alert thresholds



The screenshot shows a web browser window with the following elements:

- Browser Tab:** Disorderly Labs
- Address Bar:** <https://disorderlylabs.github.io>
- Navigation Menu:** Disorderly Labs, Research, Projects, Publications, Talks, Team, Press, Contact Us
- Logo:** The word "DISORDERLY" in a bold, black, sans-serif font above a stylized black and white illustration of a cat sitting on a ball of yarn. Below the illustration is the word "LABS" in the same bold, black, sans-serif font.
- Quote:** *One of the advantages of being disorderly is that one is constantly making exciting discoveries. - A.A. Milne*
- Section Header:**

Research
- Text:** Distributed systems are ubiquitous, but they remain notoriously difficult to reason about and program. Our research at Disorderly Labs operates at the intersection of distributed systems, data management,



The image shows a browser window displaying a PDF document. The browser's address bar shows the URL <https://people.ucsc.edu/~palvaro/socc16.pdf>. The PDF viewer interface includes a title bar with 'socc16.pdf', a page indicator '1 / 12', and navigation icons for back, forward, refresh, download, print, and bookmark. The main content area displays the title and authors of the paper.

Automating Failure Testing Research at Internet Scale

Peter Alvaro
UC Santa Cruz
palvaro@ucsc.edu

Kolton Andrus
Gremlin, Inc. (Formerly Netflix)
kolton@gremlininc.com

Chris Sanden Casey Rosenthal Ali Basiri
Lorin Hochstein
Netflix, Inc.
csanden,crosenthal,abasiri,lhochstein@netflix.com

Abstract

Large-scale distributed systems must be built to anticipate and mitigate a variety of hardware and software failures. In order to build confidence that fault-tolerant systems are correctly implemented, Netflix (and similar enterprises) regularly run *failure drills* in which faults are deliberately in-

the rule. In order to provide an “always on” experience to customers, the software used by Internet companies must be written to anticipate and work around a variety of error conditions, many of which are only present at large scale. It is difficult to ensure that such fault-tolerant code is adequately tested, because there are so many ways that a Internet-scale distributed system can fail.

Where to learn more?

[Principles of Chaos](#)

[Chaos Engineering Book](#)

[Awesome Chaos Engineering GitHub Repo](#)

[Chaos Engineering Slack](#) gremlin.com/slack

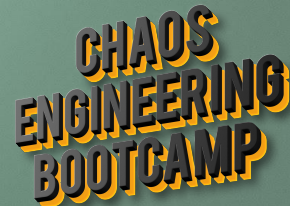
[Gremlin Community](#)

[Netflix - Chaos Kong](#)

[Chaos Engineering Meetups](#)

[Learn More about SEV](#)

[Learn More about GameDays](#)



Q&A

Any Questions?

THANKS!
we have stickers!



 [@ana_m_medina](https://twitter.com/ana_m_medina)

 [@HoReal](https://twitter.com/HoReal)

**CHAOS
ENGINEERING
BOOTCAMP**