TAMMY BUTOW

CAUSING CHAOS IN PROD SINCE 2009

@TAMMYBUTOW

ENGINEERING MANAGER, DROPBOX
CASEY ROSENTHAL

ASSISTING & ANSWERING YOUR CHAOS QUESTIONS

@CASEYROSENTHAL

ENGINEERING MANAGER, NETFLIX
THE CHAOS BOOTCAMP

+ LAYING THE FOUNDATION (9:00 - 10:30)
+ MORNING BREAK (10:30 - 11:00)
+ CHAOS TOOLS (11:00 - 11:30)
+ ADVANCED TOPICS + Q & A (11:30 - 12:30)
THANKS TO

- DROPBOX
- NETFLIX
- DIGITALOCEAN
- DATADOG
- GOOGLE
- AMAZON
- NATIONAL AUSTRALIA BANK
PART I: LAYING THE FOUNDATION
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.
CHAOS ENGINEERING CAN BE THOUGHT OF AS THE FACILITATION OF EXPERIMENTS TO UNCOVER SYSTEMIC WEAKNESSES.
PRINCIPLES OF CHAOS ENGINEERING

1. DEFINE STEADY STATE
2. HYPOTHESIZE STEADY STATE WILL CONTINUE
3. INTRODUCE VARIABLES THAT REFLECT REAL WORLD EVENTS
4. TRY TO DISPROVE THE HYPOTHESIS
WHY DO DISTRIBUTED SYSTEMS NEED CHAOS?

DISTRIBUTED SYSTEMS HAVE NUMEROUS SYSTEM PARTS.

HARDWARE AND FIRMWARE FAILURES ARE COMMON. OUR SYSTEMS AND COMPANIES SCALE RAPIDLY

HOW DO YOU BUILD A RESILIENT SYSTEM WHILE YOU SCALE?
WE USE CHAOS!
You can inject chaos at any layer to increase system resilience and system knowledge.
WHO USES CHAOS ENGINEERING?

1. NETFLIX
2. DROPBOX
3. GOOGLE
4. NATIONAL AUSTRALIA BANK
5. JET
WHAT ARE COMMON EXCUSES TO NOT USE CHAOS ENGINEERING?

NO EXCUSES.
GET READY FOR CHAOS.
HANDS-ON TUTORIAL (LET’S JUMP IN!)

NOW IT IS TIME TO CREATE CHAOS. WE WILL ALL BE DOING A HANDS-ON ACTIVITY WHERE WE INJECT FAILURE.
EVERYONE HAS A DIGITALOCEAN SERVER, USERNAME AND PASSWORD.

1. LOGIN WITH TERMINAL
2. VISIT YOUR IP IN YOUR BROWSER
YOU MUST BE MEASURING METRICS AND REPORTING ON THEM TO IMPROVE YOUR SYSTEM RESILIENCE.
CHAOS WITHOUT MONITORING IS FUTILE
THE LACK OF PROPER MONITORING IS NOT USUALLY THE SOLE CAUSE OF A PROBLEM, BUT IT IS OFTEN A SERIOUS CONTRIBUTING FACTOR. AN EXAMPLE IS THE NORTHEAST BLACKOUT OF 2003.

COMMON ISSUES INCLUDE:
+ HAVING THE WRONG TEAM DEBUG
+ NOT ESCALATING
+ NOT HAVING A BACKUP ON-CALL
Northeast blackout of 2003

From Wikipedia, the free encyclopedia

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

Some power was restored by 11 p.m. Many others did not get their power back until two days later. In more remote areas it took nearly a week to restore power.[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 a control room of the FirstEnergy Corporation, located in Ohio. A lack of alarm left operators unaware of the need to re-distribute power after overloaded transmission lines hit unplowed foliage, which triggered a race condition in the control software. What would have been a manageable local blackout cascaded into massive widespread distress on the electric grid.

### Contents

[show]

### Immediate impact

A LACK OF ALARMS LEFT OPERATORS UNAWARE OF THE NEED TO RE-DISTRIBUTE POWER AFTER OVERLOADED TRANSMISSION LINES HIT UNPRUNED FOLIAGE.

THIS TRIGGERED A RACE CONDITION IN THE CONTROL SOFTWARE.
WHAT SHOULD YOU MEASURE

1. AVAILABILITY — 500s
2. SERVICE SPECIFIC KPIs
3. SYSTEM METRICS: CPU, IO, DISK
4. CUSTOMER COMPLAINTS
CASE STUDY: KUBERNETES SOCK SHOP

1. UNDERSTAND SYSTEM
2. DETERMINE SLAs/SLOs/KPIs
3. SETUP MONITORING
4. INJECT CHAOS
5. MEASURE RESULTS
6. LEARN
7. INCREASE SYSTEM RESILIENCE
LUCKY YOU. YOUR MONITORING IS ALREADY UP.

1. DATADOG IS UP AND READY
2. THE AGENT IS ALREADY REPORTING METRICS FOR YOU!
<table>
<thead>
<tr>
<th>Hostname</th>
<th>Status</th>
<th>CPU</th>
<th>IOWait</th>
<th>Load 15</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>kube-26</td>
<td>UP</td>
<td>20%</td>
<td>0.2%</td>
<td>0.34</td>
<td>ntp</td>
</tr>
<tr>
<td>kube-154</td>
<td>UP</td>
<td>15%</td>
<td>0.5%</td>
<td>0.34</td>
<td>system</td>
</tr>
<tr>
<td>kube-15</td>
<td>UP</td>
<td>15%</td>
<td>0.3%</td>
<td>0.14</td>
<td>system</td>
</tr>
<tr>
<td>kube-182</td>
<td>UP</td>
<td>14%</td>
<td>0.2%</td>
<td>0.17</td>
<td>ntp</td>
</tr>
<tr>
<td>kube-189</td>
<td>UP</td>
<td>14%</td>
<td>0%</td>
<td>0.17</td>
<td>system</td>
</tr>
<tr>
<td>kube-185</td>
<td>UP</td>
<td>14%</td>
<td>0.2%</td>
<td>0.22</td>
<td>ntp</td>
</tr>
<tr>
<td>kube-111</td>
<td>UP</td>
<td>13%</td>
<td>0%</td>
<td>0.22</td>
<td>system</td>
</tr>
<tr>
<td>kube-100</td>
<td>UP</td>
<td>13%</td>
<td>0%</td>
<td>0.19</td>
<td>ntp</td>
</tr>
<tr>
<td>kube-21</td>
<td>UP</td>
<td>13%</td>
<td>0.3%</td>
<td>0.12</td>
<td>ntp</td>
</tr>
<tr>
<td>kube-105</td>
<td>UP</td>
<td>13%</td>
<td>0.2%</td>
<td>0.19</td>
<td>system</td>
</tr>
<tr>
<td>kube-126</td>
<td>UP</td>
<td>12%</td>
<td>0%</td>
<td>0.29</td>
<td>system</td>
</tr>
<tr>
<td>kube-25</td>
<td>UP</td>
<td>12%</td>
<td>0.1%</td>
<td>0.23</td>
<td>system</td>
</tr>
<tr>
<td>kube-183</td>
<td>UP</td>
<td>12%</td>
<td>0.2%</td>
<td>0.19</td>
<td>system</td>
</tr>
<tr>
<td>kube-77</td>
<td>UP</td>
<td>12%</td>
<td>0%</td>
<td>0.18</td>
<td>system</td>
</tr>
<tr>
<td>kube-179</td>
<td>UP</td>
<td>12%</td>
<td>0.3%</td>
<td>0.2</td>
<td>system</td>
</tr>
<tr>
<td>kube-135</td>
<td>UP</td>
<td>12%</td>
<td>0.1%</td>
<td>0.23</td>
<td>system</td>
</tr>
<tr>
<td>kube-141</td>
<td>UP</td>
<td>12%</td>
<td>0.2%</td>
<td>0.31</td>
<td>ntp</td>
</tr>
<tr>
<td>kube-163</td>
<td>UP</td>
<td>12%</td>
<td>0.2%</td>
<td>0.18</td>
<td>system</td>
</tr>
<tr>
<td>kube-191</td>
<td>UP</td>
<td>12%</td>
<td>0.2%</td>
<td>0.21</td>
<td>system</td>
</tr>
<tr>
<td>kube-166</td>
<td>UP</td>
<td>12%</td>
<td>0.2%</td>
<td>0.14</td>
<td>system</td>
</tr>
<tr>
<td>kube-160</td>
<td>UP</td>
<td>12%</td>
<td>0.4%</td>
<td>0.18</td>
<td>system</td>
</tr>
</tbody>
</table>
CHAOS TYPES

KNOWN

UNKNOWN

KNOWN

UNKNOWN
LET’S INJECT KNOWN CHAOS

1. CHOOSE A SIMIAN ARMY SCRIPT

$cd ~/SimianArmy/src/main/resources/scripts
1. CHOOSE A SIMIAN ARMY SCRIPT

```bash
cd ~/SimianArmy/src/main/resources/scripts
chaos@kube-tammy:~/SimianArmy/src/main/resources/scripts$ ls
burncpu.sh faildynamodb.sh filldisk.sh networklatency.sh
burnio.sh failec2.sh killprocesses.sh networkloss.sh
faildns.sh fails3.sh networkcorruption.sh nullroute.sh
```
$vim burncpu.sh

#!/bin/bash
# Script for BurnCpu Chaos Monkey

cat << EOF > /tmp/infiniteburn.sh
#!/bin/bash
while true;
  do openssl speed;
done
EOF

# 32 parallel 100% CPU tasks should hit even the biggest EC2 instances
for i in {1..32}
do	nohup /bin/bash /tmp/infiniteburn.sh &
done
LET’S INJECT
KNOWN CHAOS

```bash
chaos@kube-tammy:~/SimianArmy/src/main/resources/scripts$ ls
burncpu.sh faildynamodb.sh filldisk.sh networklatency.sh
burnio.sh failec2.sh killprocesses.sh networkloss.sh
faildns.sh fails3.sh networkcorruption.sh nullroute.sh
chaos@kube-tammy:~/SimianArmy/src/main/resources/scripts$ chmod +x burncpu.sh
chaos@kube-tammy:~/SimianArmy/src/main/resources/scripts$ ./burncpu.sh
nohup: nohup: nohup: appending output to 'nohup.out'
nohup: nohup: nohup: appending output to 'nohup.out'
nohup: nohup: nohup: appending output to 'nohup.out'
nodup: nohup: nohup: appending output to 'nohup.out'
```
<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIWT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>10887</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3516</td>
<td>3600</td>
<td>R</td>
<td>6.2</td>
<td>0.1</td>
<td>0:07.10</td>
<td>openssl</td>
</tr>
<tr>
<td>18107</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3472</td>
<td>3836</td>
<td>R</td>
<td>6.2</td>
<td>0.1</td>
<td>0:07.13</td>
<td>openssl</td>
</tr>
<tr>
<td>18125</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3592</td>
<td>3160</td>
<td>R</td>
<td>6.2</td>
<td>0.1</td>
<td>0:07.10</td>
<td>openssl</td>
</tr>
<tr>
<td>18093</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3336</td>
<td>2980</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.23</td>
<td>openssl</td>
</tr>
<tr>
<td>18094</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3464</td>
<td>3032</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.05</td>
<td>openssl</td>
</tr>
<tr>
<td>18102</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3500</td>
<td>3664</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.10</td>
<td>openssl</td>
</tr>
<tr>
<td>18103</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3526</td>
<td>3884</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.30</td>
<td>openssl</td>
</tr>
<tr>
<td>18108</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3472</td>
<td>2992</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.08</td>
<td>openssl</td>
</tr>
<tr>
<td>18109</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3464</td>
<td>3028</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.14</td>
<td>openssl</td>
</tr>
<tr>
<td>18110</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3492</td>
<td>3856</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.21</td>
<td>openssl</td>
</tr>
<tr>
<td>18111</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3468</td>
<td>3012</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:06.99</td>
<td>openssl</td>
</tr>
<tr>
<td>18112</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3526</td>
<td>3884</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.69</td>
<td>openssl</td>
</tr>
<tr>
<td>18117</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3596</td>
<td>3160</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:06.97</td>
<td>openssl</td>
</tr>
<tr>
<td>18118</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3592</td>
<td>3156</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.01</td>
<td>openssl</td>
</tr>
<tr>
<td>18121</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3584</td>
<td>3156</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.06</td>
<td>openssl</td>
</tr>
<tr>
<td>18122</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3512</td>
<td>3676</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.08</td>
<td>openssl</td>
</tr>
<tr>
<td>18124</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3444</td>
<td>3000</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.05</td>
<td>openssl</td>
</tr>
<tr>
<td>18126</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3512</td>
<td>3876</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.13</td>
<td>openssl</td>
</tr>
<tr>
<td>18127</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3532</td>
<td>3096</td>
<td>R</td>
<td>5.9</td>
<td>0.1</td>
<td>0:07.05</td>
<td>openssl</td>
</tr>
<tr>
<td>18086</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3368</td>
<td>2844</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.17</td>
<td>openssl</td>
</tr>
<tr>
<td>18089</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3248</td>
<td>2820</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.21</td>
<td>openssl</td>
</tr>
<tr>
<td>18091</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3426</td>
<td>2984</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.19</td>
<td>openssl</td>
</tr>
<tr>
<td>18099</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3464</td>
<td>3028</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.06</td>
<td>openssl</td>
</tr>
<tr>
<td>18106</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3554</td>
<td>3888</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.11</td>
<td>openssl</td>
</tr>
<tr>
<td>18113</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3492</td>
<td>3856</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:06.96</td>
<td>openssl</td>
</tr>
<tr>
<td>18114</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3592</td>
<td>3160</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.18</td>
<td>openssl</td>
</tr>
<tr>
<td>18115</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3516</td>
<td>3000</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.10</td>
<td>openssl</td>
</tr>
<tr>
<td>18116</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3564</td>
<td>3132</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.08</td>
<td>openssl</td>
</tr>
<tr>
<td>18119</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3516</td>
<td>3000</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.09</td>
<td>openssl</td>
</tr>
<tr>
<td>18120</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3524</td>
<td>3884</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.12</td>
<td>openssl</td>
</tr>
<tr>
<td>18123</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3484</td>
<td>3048</td>
<td>R</td>
<td>5.6</td>
<td>0.1</td>
<td>0:07.04</td>
<td>openssl</td>
</tr>
<tr>
<td>18090</td>
<td>chaos</td>
<td>20</td>
<td>0</td>
<td>13936</td>
<td>3348</td>
<td>2916</td>
<td>R</td>
<td>5.2</td>
<td>0.1</td>
<td>0:07.18</td>
<td>openssl</td>
</tr>
<tr>
<td>8583</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>462516</td>
<td>88536</td>
<td>42860</td>
<td>S</td>
<td>4.9</td>
<td>7.2</td>
<td>0:56.51</td>
<td>kubelet</td>
</tr>
<tr>
<td>8781</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>1046590</td>
<td>78756</td>
<td>41700</td>
<td>S</td>
<td>5.3</td>
<td>1.7</td>
<td>0:35.48</td>
<td>kube-controller</td>
</tr>
<tr>
<td>3470</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>1436400</td>
<td>65916</td>
<td>29088</td>
<td>R</td>
<td>1.8</td>
<td>2.8</td>
<td>0:19.78</td>
<td>dockerd</td>
</tr>
<tr>
<td>8975</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>178744</td>
<td>138264</td>
<td>46524</td>
<td>S</td>
<td>1.8</td>
<td>3.4</td>
<td>0:22.28</td>
<td>kube-apiserver</td>
</tr>
<tr>
<td>8835</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>18.045g</td>
<td>48892</td>
<td>16656</td>
<td>S</td>
<td>0.7</td>
<td>1.0</td>
<td>0:11.76</td>
<td>etcd</td>
</tr>
<tr>
<td>13340</td>
<td>10881</td>
<td>20</td>
<td>0</td>
<td>1649396</td>
<td>274304</td>
<td>9816</td>
<td>S</td>
<td>0.7</td>
<td>6.8</td>
<td>1:02.37</td>
<td>java</td>
</tr>
<tr>
<td>13370</td>
<td>099</td>
<td>20</td>
<td>0</td>
<td>956386</td>
<td>58624</td>
<td>27712</td>
<td>S</td>
<td>0.7</td>
<td>1.4</td>
<td>0:93.93</td>
<td>mongod</td>
</tr>
</tbody>
</table>
CHAOS IN DATADOG
LET’S STOP THE KNOWN CHAOS

1. KILL WHAT I RAN AS CHAOS USER

`pkill -u chaos`
NO MORE CHAOS
WHAT KIND OF CHAOS DO WE INJECT AT DROPBOX?

1. WE KILL MYSQL PRIMARY
2. WE KILL MYSQL REPLICA
3. WE KILL THE MYSQL PROXY
HOW DO WE MAKE MYSQL RESILIENT TO KILLS?

WE USE SEMI SYNC, GROUP REPLICATION AND WE CREATED A TOOL CALLED AUTO REPLACE TO DO CLONES AND PROMOTIONS.
CHAOS CREATES RESILIENCE
INJECT CHAOS IN YOUR SYSTEM
LET'S INJECT KNOWN CHAOS

chaos@kube-tammy:~/SimianArmy/src/main/resources/scripts$ ls
burncpu.sh faildynamodb.sh filldisk.sh networklatency.sh
burnio.sh failec2.sh killprocesses.sh networkloss.sh
faildns.sh fails3.sh networkcorruption.sh nullroute.sh
chaos@kube-tammy:~/SimianArmy/src/main/resources/scripts$ chmod +x burncpu.sh
chaos@kube-tammy:~/SimianArmy/src/main/resources/scripts$ ./burncpu.sh
nohup: nohup: nohup: appending output to 'nohup.out'
nohup: nohup: nohup: appending output to 'nohup.out'
nohup: nohup: nohup: appending output to 'nohup.out'
nohup: nohup: nohup: appending output to 'nohup.out'
nohup: nohup: nohup: appending output to 'nohup.out'
appending output to 'nohup.out'
WHAT TYPES OF CHAOS DID YOU INJECT?
WHAT WAS YOUR HYPOTHESIS?
30 MIN MORNING TEA BREAK
10:30 — 11:00
THANKS TO GOOGLE!
PART II: CHAOS TOOLS
WHAT TYPES OF CHAOS DID YOU INJECT?
WHAT WAS YOUR HYPOTHESIS?
SOME CHAOS CASE STUDIES.....
LET’S GO BACK IN TIME TO LOOK AT WORST OUTAGE STORIES WHICH THEN LED TO THE INTRODUCTION OF CHAOS ENGINEERING.
CHAOS @ DROPBOX

DROPBOX’S WORST OUTAGE EVER
SOME MASTER-REPLICA PAIRS WERE IMPACTED WHICH RESULTED IN THE SITE GOING DOWN.

NOW WE HAVE
CHAOS @ DROPBOX

1. CHAOS DAYS
2. RACK SHUTDOWN
3. SERVICE DRTs
QUICK THOUGHTS…..

+ SO MANY WORST OUTAGE STORIES ARE THE DATABASE.
+ I LEAD DATABASES AT DROPBOX & WE DO CHAOS.
+ FEAR WILL NOT HELP YOU SURVIVE “THE WORST OUTAGE”.
+ DO YOU TEST YOUR ALERTS & MONITORING? WE DO.
+ HOW VALUABLE IS A POSTMORTEM IF YOU DON’T HAVE ACTION ITEMS AND DO THEM? NOT VERY.
CHAOS @ UBER

UBER’S **WORST OUTAGE EVER:**
1. MASTER LOG REPLICATION TO S3 FAILED
2. LOGS BACKED UP ON PRIMARY
3. ALERTS FIRE TO ENGINEER BUT THEY ARE IGNORED
4. DISK FILLS UP ON DATABASE PRIMARY
5. ENGINEER DELETES UNARCHIVED WAL FILES
6. ERROR IN CONFIG PREVENTS PROMOTION

— Matt Ranney, UBER, YOW 2015
Scaling Uber with Matt Ranney

by Pranay | December 4, 2015 | in Cloud Engineering, Greatest Hits, Podcast | 0

Podcast: Play in new window | Download

“If you can make a system that can survive this random failure testing, then you will more or likely survive whatever other chaotic conditions exist.”

Uber is a transportation and logistics company that manages many aspects of its ride-sharing services through mobile apps and distributed technology. Uber faces unique challenges in rapidly scaling its services internationally, and at one point increased its developer headcount from 200 to over 1000 in less than a year.

Matt Ranney is the Chief Systems Architect at Uber and was previously a founder and CTO of Voxer. At QCon San Francisco, he gave a talk called Scaling Uber.
CHAOS @ UBER

+ UBER BUILT UDESTROY TO SIMULATE FAILURES.
+ DIDN’T USE NETFLIX SIMIAN ARMY AS IT WAS AWS-CENTRIC.
+ ENGINEERS AT UBER DON’T LIKE FAILURE TESTING (ESP. DATABASES)
  .....THIS IS DUE TO THEIR WORST OUTAGE EVER:

— Matt Ranney, UBER, YOW 2015
CHAOS @ NETFLIX

SIMIAN ARMY CONSISTS OF SERVICES (MONKEYS) IN THE CLOUD FOR GENERATING VARIOUS KINDS OF FAILURES, DETECTING ABNORMAL CONDITIONS, AND TESTING THE ABILITY TO SURVIVE THEM. THE GOAL IS THE KEEP THE CLOUD SAFE, SECURE AND HIGHLY AVAILABLE.

+ CHAOS MONKEY
+ JANITOR MONKEY
+ CONFORMITY MONKEY
CHAOS @ GITLAB

GITLAB’S WORST OUTAGE EVER... KEEPS REPEATING

1. ACCIDENTAL REMOVAL OF DATA FROM PRIMARY DATABASE
2. DATABASE OUTAGE DUE TO PROJECT_AUTHORIZATIONS HAVING TOO MUCH BLOAT
3. CI DISTRIBUTED HEAVY POLLING AND EXCESSIVE ROW LOCKING FOR SECONDS TAKES GITLAB.COM DOWN
4. SCARY DATABASE SPIKES

https://about.gitlab.com/2017/02/10/postmortem-of-database-outage-of-january-31/
CHAOS @ GITLAB

GITLAB ARE NOT YET DOING CHAOS ENGINEERING. SHOULD BE FOR SURE.
CHAOS @ GOOGLE

GOOGLE RUN DRTs AND HAVE BEEN FOR MANY YEARS
Meet Kripa Krishnan, Google's queen of chaos

If aliens were to invade earth, or an earthquake lopped California into the ocean, Google wants you to know that it will still be there for you.

That’s because a group of 10 Google engineers go around purposely breaking things at the company, under the most dire circumstances they can imagine, just to see what happens.

Through this process, the team makes sure that Google can keep itself running no matter what happens.

http://www.businessinsider.com/profile-of-google-disaster-recovery-testing-boss-kripa-krishnan-2016-8
“RESILIENCE HAS TO BE DESIGNED. HAS TO BE TESTED. IT’S NOT SOMETHING THAT HAPPENS AROUND A TABLE AS A SLEW OF EXCEPTIONAL ENGINEERS ARCHITECT THE PERFECT SYSTEM. PERFECTION COMES THROUGH REPEATEDLY TRYING TO BREAK THE SYSTEM”

— VICTOR KLANG, TYPESAFE
CHAOS @ BUILDKITE

DECIDED TO REDUCE DATABASE CAPACITY IN AWS. RESULTED IN AN OUTAGE AT 3:21AM. PAGERDUTY WAS MISCONFIGURED AND PHONES WERE ON SILENT.

NOBODY WOKE UP DURING THE 4 HOUR OUTAGE.....
OH NOES!
CHAOS @ STRIPE

“A DATABASE INDEX OPERATION RESULTED IN 90 MINUTES OF INCREASINGLY DEGRADED AVAILABILITY FOR THE STRIPE API AND DASHBOARD. IN AGGREGATE, ABOUT TWO THIRDS OF ALL API OPERATIONS FAILED DURING THIS WINDOW.”

https://support.stripe.com/questions/outage-postmortem-2015-10-08-utc
INTRODUCING CHAOS IN A CONTROLLED WAY WILL RESULT IN ENGINEERS BUILDING INCREASINGLY RESILIENT SYSTEMS.

HAVE I CONVINCED YOU?
OUTAGES HAPPEN.

THERE ARE MANY MORE YOU CAN READ ABOUT HERE:

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

YOU SET IT UP AS A CRON JOB THAT CALLS CHAOS MONKEY ONCE A WEEKDAY TO CREATE A SCHEDULE OF TERMINATIONS.

HAS BEEN AROUND FOR MANY YEARS! USED AT BANKS, E-COMMERCE STORES, TECH COMPANIES + MORE
Chaos Monkey is a service that randomly terminates VM instances and containers—these frequent failures promote the creation of resilient services. Chaos Monkey 2.0 is tightly integrated with Spinnaker: it relies on the Spinnaker APIs to terminate instances, retrieves deployment information from Spinnaker, and is configured using the Spinnaker UI.

Here, I'll walk you through setting up and running Chaos Monkey on Google Compute Engine (GCE).
Chaos Monkey is responsible for randomly terminating instances in production to ensure that engineers implement their services to be resilient to instance failures.

See how to deploy for instructions on how to get up and running with Chaos Monkey.

Once you’re up and running, see configuring behavior via Spinnaker for how users can customize the behavior of Chaos Monkey for their apps.

https://netflix.github.io/chaosmonkey/
Spinnaker is an open source, multi-cloud continuous delivery platform for releasing software changes with high velocity and confidence.

https://www.spinnaker.io/
DEMO TIME

SPINNAKER + CHAOS MONKEY
Deployment Steps

Step 1. Prepare an AWS Account

1. If you don’t already have an AWS account, create one at http://aws.amazon.com by following the on-screen instructions.

2. Use the region selector in the navigation bar to choose the AWS Region where you want to deploy Spinnaker on AWS.

3. Create a key pair in your preferred region.

4. Open the IAM console at https://console.aws.amazon.com/iam/, create an IAM role called BaseIAMRole, and choose Amazon EC2 as the role type. EC2 instances launched with Spinnaker will be associated with this role.

5. If necessary, request a service limit increase for the Amazon EC2 m4.xlarge instance type. You might need to do this if you already have an existing deployment that uses this instance type, and you think you might exceed the default limit with this reference deployment.

Create stack

Select Template

Specify Details
Options
Review

Select the template that describes the stack that you want to create. A stack is a group of related resources that you manage as a single unit.

Design a template
Use AWS CloudFormation Designer to create or modify an existing template. Learn more.

Choose a template
A template is a JSON/YAML-formatted text file that describes your stack’s resources and their properties. Learn more.

- Select a sample template
- Upload a template to Amazon S3
  - Choose File
  - No file chosen
- Specify an Amazon S3 template URL

Create stack

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. Learn more.

Stack name: SpinnakerVelocity

Parameters

- **KeyName**: chaoskeypair
- **Password**: **********
- **SpinnakerPrivateSubnetCIDR**: 10.100.11.0/24
- **SpinnakerPublicSubnetCIDR**: 10.100.10.0/24
- **SpinnakerVPCCIDR**: 10.100.0.0/16
- **SSHLocation**: 0.0.0.0/0

The IP address range that can be used to SSH to the Bastion host. Note: a value of 0.0.0.0/0 will allow access from ANY ip address

The following resource(s) require capabilities: [AWS::IAM::Role]

This template contains Identity and Access Management (IAM) resources that might provide entities access to make changes to your AWS account. Check that you want to create each of these resources and that they have the minimum required permissions. Learn more.

I acknowledge that AWS CloudFormation might create IAM resources.
LEO

Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-119-generic x86_64)

* Documentation: https://help.ubuntu.com/

System information as of Tue Jun 20 01:24:44 UTC 2017

System load: 0.24 Processes: 432
Usage of /: 39.6% of 9.71GB Users logged in: 0
Memory usage: 38% IP address for eth0: 10.100.10.58
Swap usage: 0%

Graph this data and manage this system at:
https://landscape.canonical.com/

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

9 packages can be updated.
2 updates are security updates.

Last login: Tue Jun 20 01:24:45 2017 from ip-10-100-10-172.us-west-2.compute.internal
ubuntu@ip-10-100-10-58:~$
New Application

- Name: bootcamp
- Owner Email: tammy@dropbox.com
- Repo Type: Select Repo Type
- Chaos Monkey: Enabled
- Description: Enter a description
- AWS Settings: Prefer AMI Block Device Mappings
- Instance Health: Consider only cloud provider health when executing tasks
- Instance Port: 
- Pipeline Behavior: Enable restarting running pipelines

* Required

[Cancel] [Create]
Cloud Launcher

New Spinnaker deployment

Deployment name
spinnaker-1

Zone
us-central1-f

Machine type
4 vCPUs 26 GB memory Customize
Upgrade your account to create instances with up to 32 cores

More

Networking
Network name
default

Subnetwork name
default

Jenkins
Show Jenkins options

Spinnaker overview
Solution provided by Google Click To Deploy

Spinnaker will be deployed on a single Compute Engine instance. By default, this instance is an n1-highmem-4 (4 vCPUs, 26 GB RAM).

Software

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Debian 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>Spinnaker (OSS)</td>
</tr>
</tbody>
</table>

Terms of Service

The software or service you are about to use is not a Google product. By deploying the software or accessing the service you are agreeing to comply with the Google Cloud Launcher terms of service and the terms of any third party software licenses related to the software or service. Please review these licenses carefully for details about any obligations you may have related to the software or services. To the limited extent an open source software license related to the software or service expressly supersedes the Google Cloud Launcher Terms of Service, that open source software license governs your use of that software or service.

Google is providing this software or service "as-is" and will not perform any ongoing maintenance. Ongoing upgrades and maintenance are your responsibility.
Chaos Monkey is a service that randomly terminates VM instances and containers—these frequent failures promote the creation of resilient services. Chaos Monkey 2.0 is tightly integrated with Spinnaker: it relies on the Spinnaker APIs to terminate instances, retrieves deployment information.

https://blog.spinnaker.io/running-chaos-monkey-on-spinnaker-google-compute-engine-gce-155dc52f20ef
https://blog.spinnaker.io/running-chaos-monkey-on-spinnaker-google-compute-engine-gce-155dc52f20ef
CHAOS KONG TAKES DOWN AN ENTIRE AWS REGION. NETFLIX CREATED IT BECAUSE AWS HAD NOT YET BUILT THE ABILITY TO TEST THIS.

AWS REGION OUTAGES DO HAPPEN!
CHAOS FOR KUBERNETES

ASOBTI, AN ENGINEER @ BOX CREATED
https://github.com/asobti/kube-monkey

IT RANDOMLY DELETES KUBERNETES PODS IN THE CLUSTER ENCOURAGING AND VALIDATING THE DEPLOYMENT OF FAILURE-RESILIENT SYSTEMS.
A SUITE OF TOOLS FOR KEEPING YOUR CLOUD OPERATING IN TOP FORM. CHAOS MONKEY IS THE FIRST MEMBER. OTHER SIMIANS INCLUDE JANITOR MONKEY & CONFORMITY MONKEY.

https://github.com/Netflix/SimianArmy
GREMLIN INC

GREMLIN PROVIDES “FAILURE AS A SERVICE”. IT FINDS WEAKNESSES IN YOUR SYSTEM BEFORE THEY END UP IN THE NEWS.

LIKE A VACCINATION, THEY SAFELY INJECT HARM INTO YOUR SYSTEM TO BUILD IMMUNITY TO FAILURE.

https://gremlininc.com/
MONITORING + ALERTING

- DATADOG
- GRAFANA
- PAGERDUTY
PART III: ADVANCED TOPICS AND Q&A
CHAOS ENGINEERING FOR DATABASES

• GOOD TO USE:
  • MYSQL
  • ORCHESTRATOR
  • GROUP REPLICATION
  • SEMI SYNC

https://github.com/github/orchestrator
Authored by Shlomi Noach at GitHub. Previously at Booking.com and Outbrain

https://github.com/github/orchestrator
CHAOS ENGINEERING WITH GO

THINK ABOUT WHAT FAILURE YOU CAN INJECT AND THEN CATCH.

WE DO THIS WITH MAGIC POCKET AT DROPBOX.
GO CLIENT TO THE CHAOS MONKEY REST API

THIS PROJECT WAS STARTED FOR THE PURPOSE OF CONTROLLED FAILURE INJECTION DURING GAME DAYS.

https://github.com/mlafeldt/chaosmonkey

go get -u github.com/mlafeldt/chaosmonkey/lib
VIZCERAL

A TOOL FOR “INTUITION ENGINEERING” TO HELP YOU VISUALIZE YOUR NETWORK AND TRAFFIC.

CREATED BY NETFLIX.

https://github.com/Netflix/vizceral
VIZCERAL
BY @JRSQUARED

SPLIT FAILOVER! cooooOOOO ahhhhhhhh
@jrsquared (I said whoa & almost dropped phone)
An open source load testing tool.

Define user behaviour with Python code, and swarm your system with millions of simultaneous users.

http://locust.io/
Chaos Lemur

This project is a self-hostable application to randomly destroy virtual machines in a BOSH-managed environment, as an aid to resilience testing of high-availability systems. Its main features are:

- Triggers on a user-defined schedule, selecting 0 or more VMs to destroy at random during each run.
- Manual triggering of unscheduled destroys.
- Per-deployment and per-job probabilities for destruction of member VMs.
- Optional blacklisting of deployments and jobs to protect their members from destruction.
- Runs against different types of IaaS (e.g. AWS, vSphere) using a small infrastructure API.
- Optionally records activities to DataDog.

Although Chaos Lemur recognizes deployments and jobs, it is not possible to select an entire deployment or job for destruction. Entire deployments and jobs will be destroyed over time by chance, given sufficient runs.

https://github.com/strepsirrhini-army/chaos-lemur
Monkeys in Lab Coats: Applying Failure Testing Research @Netflix

Summary
The authors present their experience in collaboration between industry and academia, describing how a "big idea"—lineage-driven fault injection—evolved from a theoretical model into an automated failure testing system that leverages Netflix's state-of-the-art fault injection and tracing infrastructures.

Bio
Kollon Andrue is the founder of Gremlin Inc. He is passionate about building resilient systems, primarily as it lets him break things for fun and profit. Peter Alvaro is an Assistant Professor of Computer Science at the University of California Santa Cruz. He is the creator of the Dedalus language and co-creator of the Bloom language.
DISORDERLY LABS

One of the advantages of being disorderly is that one is constantly making exciting discoveries. - A.A. Milne

Research

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, programming languages and formal verification. We build languages, models and tools to help tame the fundamental complexity of distributed programming.
Lineage-driven Fault Injection

Peter Alvaro
UC Berkeley
palvaro@cs.berkeley.edu

Joshua Rosen
UC Berkeley
rosenville@gmail.com

Joseph M. Hellerstein
UC Berkeley
hellerstein@cs.berkeley.edu

ABSTRACT
Failure is always an option in large-scale data management systems, and it is practically a certainty. Fault-tolerant protocols and components are notoriously difficult to implement and debug. We propose a novel approach for discovering bugs in fault-tolerant data management systems: lineage-driven fault injection. A lineage-driven fault injector runs backwards from correct system outcomes to determine whether failures in the execution could have prevented the outcome. This approach identifies bugs that exist for a particular configuration. Molly identifies bugs rapidly, in many cases using an order of magnitude fewer executions than random fault injection. Otherwise, Molly certifies that the code is bug-free for that configuration.

enriching new system architectures with well-understood fault-tolerance mechanisms and henceforth assuming that failures will not affect system outcomes. Unfortunately, fault-tolerance is a global property of entire systems, and guarantees about the behavior of individual components do not necessarily hold under composition. It is difficult to design and reason about the fault-tolerance of indivi dual components, and often equally difficult to assemble a fault- tolerant system even when given fault-tolerant components, as witnessed by recent data management system failures [16, 57] and bugs [36, 49]. Top-down testing approaches—which perturb and observe the behavior of complex systems—are an attractive alternative to verification of individual components. Fault injection [11, 36, 36, 44, 50] is the dominant top-down approach in the software engineering and dependability communities. With minimal programmer in vestment, fault injection can quickly identify shallow bugs caused by a small number of independent faults. Unfortunately, fault injection is poorly suited to discovering rare counterexamples involving complex interactions of multiple failures and types of failure, and these bugs are the hardest to find.

https://people.ucsc.edu/~palvaro/molly.pdf
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  Case
Rosenthal  Ali Basiri
Lorin Hochstein
Netflix, Inc.
csanden,csrosenthal,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 injected in their production system. The combinatorial space of failure scenarios is too large to explore exhaustively. Existing failure testing approaches either randomly explore the space of potential failures randomly or exploit the "hunches" of domain experts to guide the search. Random strategies waste resources testing "uninteresting" faults, while programmer-guided approaches are only as good as human 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.

Chaos Engineering [10], or “experimenting on a distributed system in order to build confidence in the system’s capability to withstand turbulent conditions in production,” has emerged as a discipline to tackle resilience of these large-scale distributed systems [28, 35]. Engineers create frameworks that automate failure injection, usually on small scale test tenants.

WHERE CAN YOU LEARN MORE?

https://chaos-slack.herokuapp.com/
JOIN THE CHAOS COMMUNITY

Chaos Community

Home of the Chaos Engineering community of practice.
See the Principles of Chaos Engineering for more information on Chaos Engineering.

Chaos Community Day

Chaos Community Day is an invite-only event to discuss chaos engineering among practitioners.

- 2016
- 2015

Discussion forums

- Chaos Community Google Group
- Chaos Engineering LinkedIn Group

http://chaos.community/
YOUR TOOL HERE!

LOOK FORWARD TO SEEING YOU AT CHAOS COMMUNITY DAYS AND HEARING FROM YOU IN THE SLACK COMMUNITY AND ON THE MAILING LISTS.
CHAOS ENGINEERING BOOTCAMP

TAMMY & CASEY
VELOCITY SAN JOSE 2017
THANKS FOR ATTENDING THE: CHAOS ENGINEERING BOOTCAMP

VELOCITY SAN JOSE 2017