Resiliency Testing with Toxiproxy

Jake Pittis
Resiliency Testing with Toxiproxy

Jake Pittis
Reasoning about failure is hard.
Too many kinds of failures.
Large complex systems.
Constantly changing.
Our intuition is often wrong.
Incident

A natural failure in production.
A database writer goes down.
Incident

Root cause?

Ship fix!
Incident

Root cause?

Ship fix!

More Resilient
Gameday

Artificially exercising a known failure scenario in production.
Flash sales.
Gameday

↓

What broke?

↓

Ship fix!
Incident

Root cause?

Ship fix!

Gameday

What broke?

Ship fix!
<table>
<thead>
<tr>
<th></th>
<th>Authenticity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production Incident</td>
<td>100%</td>
</tr>
<tr>
<td>Gameday</td>
<td>Mostly</td>
</tr>
<tr>
<td></td>
<td>Authenticity</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>Production Incident</strong></td>
<td>100%</td>
</tr>
<tr>
<td><strong>Gameday</strong></td>
<td>Mostly</td>
</tr>
<tr>
<td></td>
<td>Authenticity</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Production Incident</td>
<td>100%</td>
</tr>
<tr>
<td>Gameday</td>
<td>Mostly</td>
</tr>
<tr>
<td></td>
<td>Authenticity</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>Production Incident</strong></td>
<td>100%</td>
</tr>
<tr>
<td><strong>Gameday</strong></td>
<td>Mostly</td>
</tr>
</tbody>
</table>
Resiliency is a product concern.
Automatically Prevent Regression
Automatically Prevent Regression

Accessible to All Developers
Automatically Prevent Regression

Accessible to All Developers

Lower Customer Impact
Automatically Prevent Regression

Accessible to All Developers

Lower Customer Impact

Maintain Authenticity
Chaos Engineering
Building Confidence in System Behavior through Experiments

Casey Rosenthal, Lorin Hochstein, Aaron Bluhowiak, Nora Jones & Ali Basiri

O’Reilly
Chaos Engineering

Running experiments in production to cause and fix unknown failure scenarios.
“Automate Experiments to Run Continuously”

Automatically Prevent Regression
“Minimize Blast Radius”

Lower Customer Impact
“Run Experiments in Production”

Maintain Authenticity
Toxiproxy
Inject failures via HTTP API.

Development and Test Environment

Automated Test Suite

Database

Application
Development and Test Environment

Latency of 200 ms.

Automated Test Suite

Database
Development and Test Environment

Blackhole data.

Automated Test Suite

Database
Development and Test Environment

Reject connections.

Database

Automated Test Suite

Application
Reactive Testing
Incident

Root cause?

Ship fix!

Does it work?

Regression?
A flash sale takes down redis while a deploy is going out.
A flash sale takes down redis while a deploy is going out.

Application boot relies on redis!?
A flash sale takes down redis while a deploy is going out.

Application boot relies on redis!?

Remove the dependency!
A flash sale takes down redis while a deploy is going out.

Application boot relies on redis!? 

Remove the dependency! 

Write a Toxiproxy test!
Toxiproxy[/.*/].down do
  assert application_can_boot
end
Accessible to All Developers ✅
Accessible to All Developers

✅

No Customer Impact

✅
Accessible to All Developers ✅

No Customer Impact ✅

Maintain Authenticity ✅
Accessible to All Developers

No Customer Impact

Maintain Authenticity

Automatically Prevent Regression
A few hundred Toxiproxy Tests
All you need is a thin client library.

Java, Node, Python, PHP or write your own!
I used it reactively just last week.
Proactive Testing
## Resiliency Matrix

<table>
<thead>
<tr>
<th>Services</th>
<th>Storefront</th>
<th>Checkout</th>
<th>Signups</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Redis</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Memcached</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
# Resiliency Matrix

<table>
<thead>
<tr>
<th>Services</th>
<th>Storefront</th>
<th>Checkout</th>
<th>Signups</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>Degraded</td>
<td>Down</td>
<td>Down</td>
</tr>
<tr>
<td>Redis</td>
<td>Up</td>
<td>Down</td>
<td>Up</td>
</tr>
<tr>
<td>Memcached</td>
<td>Degraded</td>
<td>Up</td>
<td>Up</td>
</tr>
</tbody>
</table>
## Resiliency Matrix

<table>
<thead>
<tr>
<th>Services</th>
<th>Storefront</th>
<th>Checkout</th>
<th>Signups</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>Degraded</td>
<td>Down</td>
<td>Down</td>
</tr>
<tr>
<td>Redis</td>
<td>Up</td>
<td>Down</td>
<td>Up</td>
</tr>
<tr>
<td>Memcached</td>
<td>Degraded</td>
<td>Up</td>
<td>Up</td>
</tr>
</tbody>
</table>
Toxiproxy['mysql_writer'].down do
  get '/'
  assert_response :ok
end
Incident

Root cause?

Ship fix and Toxiproxy test!

Gameday

What broke?

Ship fix and Toxiproxy test!

Create Resiliency Matrix

Test every intersection.
What’s next?
All our applications should have a resiliency matrices.

<table>
<thead>
<tr>
<th></th>
<th>Storefront</th>
<th>Checkout</th>
<th>Signups</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>Degraded</td>
<td>Down</td>
<td>Down</td>
</tr>
<tr>
<td>Redis</td>
<td>Up</td>
<td>Down</td>
<td>Up</td>
</tr>
<tr>
<td>Memcached</td>
<td>Degraded</td>
<td>Up</td>
<td>Up</td>
</tr>
</tbody>
</table>
Integrate Toxiproxy into all our applications by default.
Gameday everything we can’t write Toxiproxy tests for.
Automate the gamedays.
Gamedays

<table>
<thead>
<tr>
<th></th>
<th>Storefront</th>
<th>Checkout</th>
<th>Signups</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>Degraded</td>
<td>Down</td>
<td>Down</td>
</tr>
<tr>
<td>Redis</td>
<td>Up</td>
<td>Down</td>
<td>Up</td>
</tr>
<tr>
<td>Memcached</td>
<td>Degraded</td>
<td>Up</td>
<td>Up</td>
</tr>
</tbody>
</table>
Toxiproxy is open source.
(github.com/Shopify/toxiproxy)

Go read the readme for more information!

Thanks!