Fail Better: Radical Ideas from the Practice of Cloud Computing

Tom Limoncelli
Stack Overflow
ACM Highlights

- Learning Center tools for professional development: [http://learning.acm.org](http://learning.acm.org)
  - 4,500+ trusted technical books and videos by O’Reilly, Morgan Kaufmann, etc.
  - 1,300+ courses, virtual labs, test preps, live mentoring for software professionals covering programming, data management, cybersecurity, networking, project management, more
  - Training toward top vendor certifications (CEH, Cisco, CISSP, CompTIA, ITIL, PMI, etc.)
  - Learning Webinars from thought leaders and top practitioner
  - Podcast interviews with innovators, entrepreneurs, and award winners

- Popular publications:
  - Flagship *Communications of the ACM (CACM)* magazine: [http://cacm.acm.org/](http://cacm.acm.org/)
  - *ACM Queue* magazine for practitioners: [http://queue.acm.org/](http://queue.acm.org/)

- ACM Digital Library, the world’s most comprehensive database of computing literature: [http://dl.acm.org](http://dl.acm.org).

- International conferences that draw leading experts on a broad spectrum of computing topics: [http://www.acm.org/conferences](http://www.acm.org/conferences).

- Prestigious awards, including the ACM A.M. Turing and Infosys: [http://awards.acm.org](http://awards.acm.org)

- And much more... [http://www.acm.org](http://www.acm.org).
Radical Ideas from
The Practice of Cloud System Administration

Tom Limoncelli, SRE
Stack Exchange, Inc
New York City

the-cloud-book.com
@YesThatTom

www.informit.com/TPOSA
Discount code TPOSA35
Who is Tom Limoncelli?

Sysadmin since 1988

Worked at Google, AT&T/Bell Labs and many more.

SRE at Stack Exchange, Inc (NYC)
http://careers.stackoverflow.com

Blog: EverythingSysadmin.com

Twitter: @YesThatTom
THE PRACTICE OF CLOUD SYSTEM ADMINISTRATION
DESIGNING AND OPERATING LARGE DISTRIBUTED SYSTEMS

THOMAS A. LIMONCELLI • STRATA R. CHALUP • CHRISTINA J. HOGAN
<table>
<thead>
<tr>
<th>Part I</th>
<th>Design: Building It</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>Designing in a Distributed World</td>
<td>9</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Designing for Operations</td>
<td>31</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Selecting a Service Platform</td>
<td>51</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Application Architectures</td>
<td>69</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Design Patterns for Scaling</td>
<td>95</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Design Patterns for Resiliency</td>
<td>119</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part II</th>
<th>Operations: Running It</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 7</td>
<td>Operations in a Distributed World</td>
<td>147</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>DevOps Culture</td>
<td>171</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Service Delivery: The Build Phase</td>
<td>195</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>Service Delivery: The Deployment Phase</td>
<td>211</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>Upgrading Live Services</td>
<td>225</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>Automation</td>
<td>243</td>
</tr>
<tr>
<td>Chapter 13</td>
<td>Design Documents</td>
<td>275</td>
</tr>
<tr>
<td>Chapter 14</td>
<td>OnCall</td>
<td>285</td>
</tr>
<tr>
<td>Chapter 15</td>
<td>Disaster Preparedness</td>
<td>307</td>
</tr>
<tr>
<td>Chapter 16</td>
<td>Monitoring Fundamentals</td>
<td>331</td>
</tr>
<tr>
<td>Chapter 17</td>
<td>Monitoring Architecture and Practice</td>
<td>331</td>
</tr>
<tr>
<td>Chapter 18</td>
<td>Capacity Planning</td>
<td></td>
</tr>
<tr>
<td>Chapter 19</td>
<td>Creating KPIs</td>
<td></td>
</tr>
<tr>
<td>Chapter 20</td>
<td>Operational Excellence</td>
<td></td>
</tr>
<tr>
<td>Epilogue</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Appendix A    | Assessments                 |
Appendix B    | The Origins and Future of Distributed Computing and Clouds |
Appendix C    | Scaling Terminology and Concepts |
Appendix D    | Templates and Examples      |
Appendix E    | Recommended Reading         |
The Cloud
The Cloud
The Cloud
The Cloud!!!!!!
The Cloud!!1!!
We <heart> The Cloud
The cloud solves all problems.
Cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud cloud.
Distributed Computing
Distributed Computing

- Divide work among many machines
- Coordinated central or decentralized
- Examples:
  - Genomics: 100s machines working on a dataset
  - Web Service: 10 machines each taking 1/10th of the web traffic for StackExchange.com
  - Storage: xx,000 machines holding all of Gmail's messages
Distributed computing can do more “work” than the largest single computer.

More storage.
More computing power.
More memory.
More throughput.
Mo’ computers, Mo’ problems

Thousands of Users

• Bigger risks
• Failures more visible
• Automation mandatory
• Cost containment becomes critical
Mo’ computers, Mo’ problems

Thousands of Users
• Bigger risks
• Failures more visible
• Automation mandatory
• Cost containment becomes critical

In response: Radical ideas on
• Reducing risk / Improve safety
• Reliability becomes a competitive differentiator
• New automation paradigms
• Cost and economics
Make peace with failure

Parts are imperfect
Networks are imperfect
Systems are imperfect
Code is imperfect
People are imperfect
Learn how to FAIL BETTER
Buy the best, most reliable computer in the world. It is still going to fail.

If it doesn’t, you’ll still need to take it down for maintenance.
3 ways to fail better

1. Use cheaper, less reliable, hardware.
2. If a process/procedure is risky, do it a lot.
3. Don’t punish people for outages.
Fail Better Part 1 of 3:

Use cheaper, less reliable, hardware.
• Loss-damage waiver
• Liability
• Personal accident insurance
• Personal effects coverage
• Loss-damage waiver
• Liability
• Personal accident insurance
• Personal effects coverage
- Loss-damage waiver
- Liability
- Personal accident insurance
- Personal effects coverage
- Loss-damage waiver
- Liability
- Personal accident insurance
- Personal effects coverage
- Loss-damage waiver
- Liability
- Personal accident insurance
- Personal effects coverage
High-End Server
High-End Server

RAID
High-End Server

RAID

Dual PS

UPS
High-End Server

RAID

Dual PS

UPS

Gold Maintenance
Load Balancer

Code Changes to Coordinate and Distribute Work
Load Balancer — Load Balancer

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

Code Changes to Coordinate and Distribute Work
Load Balancer

Code Changes to Coordinate and Distribute Work

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance

High-End Server
RAID
Dual PS
UPS
Gold Maintenance
Reliability through software

• Resiliency through software:
  • Costs to develop. Free to deploy.

• Resiliency through hardware:
  • Costs every time you buy a new machine.
Write code so that the system is distributed.

Best hardware.

Double-spending
Write code so that the system is distributed.

Best hardware.

Double-spending
These techniques work for large grids of machines…

…and every-day systems too.
Big resiliency is cheaper

Load Balancer

50% overhead

90% 90% 90% 90% 90%

10% overhead
The right amount of resiliency is good. Too much is a waste.

Aim for an SLA target so you know when to stop.
Load balancing & redundancy is just one way to achieve resiliency.
The cheapest way to buy terabytes of RAM.
Fail Better Part 1 of 3:

Use cheaper, less reliable, hardware.
Fail Better Part 2 of 3:

If a process/procedure is risky, do it a lot.
Risky behavior vs. Risky procedures
Risky Behaviors are inherently risky

- Smoking
- Shooting yourself in the foot
- Blindfolded chainsaw juggling
Risky behavior is risky.
Risky Processes can be improved through practice

- Software Upgrades
- Database Failovers
- Network Trunk Failovers
- Hardware Hot Swaps
StackExchange.com has a “DR” site in Oregon.

StackExchange.com runs on SQL Server with “AlwaysOn” Availability Groups plus...

Redis, HAproxy, ISC BIND, CloudFlare, IIS, and many home-grown applications
Process was risky

- Took 10+ hours
- Required “hands on” by 3 teams.
- Found 30+ “improvements needed”
- Certain people were S.P.O.F.
Drill Results

<table>
<thead>
<tr>
<th>Bugs Filed</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>10</td>
</tr>
</tbody>
</table>
Drill Results

- Bugs Filed: 30 → 20
- Hours: 10 → 5
Drill Results

- Bugs Filed: 30, 20, 12
- Hours: 10, 5, 2
Why?

- Each drill “surfaces” areas of improvement.
- Each member of the team gains experience and builds confidence.
- “Smaller Batches” are better
Software Upgrades

• Traditional
  • Months of planning
  • Incompatibility issues
  • Very expensive
  • Very visible mistakes
  • By the time we’re done, time to start over again.

• Distributed Computing
  • High frequency (many times a day or week)
  • Fully automated
  • Easy to fix failures
  • Cheap… encourages experiments
“Big Bang” releases are inherently risky.
Small batches are better

**Fewer changes each batch:**
- If there are bugs, easier to identify source

**Reduced lead time:**
- It is easier to debug code written recently.

**Environment has changed less:**
- Fewer “external changes” to break on

**Happier, more motivated, employees:**
- Instant gratification for all involved
Risk is *inversely proportional* to how recently a process has been used.
Netflix “Chaos Monkey”

- Randomly reboots machines.
- Keeps Netflix “on its toes”.
- Part of the Simian Army:
  - Chaos Monkey (hosts)
  - Chaos Kong (data centers)
  - Latency Monkey (adds random performance delays)
Fail Better Part 2 of 3:

If a process/procedure is risky, do it a lot.
Fail Better Part 3 of 3:

Don’t punish people for outages.
There will always be outages.
There will always be outages.
Getting angry about outages is equivalent to expecting them to never happen... which is irrational.
Out-dated attitudes about outages

- Expect perfection: 100% uptime
- Punish exceptions:
  - fire someone to “prove we’re serious”
- Results:
  - People hide problems
  - People stop communicating
  - Discourages transparency
  - Small problems get ignored, turn into big problems
New thinking on outages

- Set uptime goals: 99.9% +/- 0.05
- Anticipate outages:
  - Strategic resiliency techniques, oncall system
  - Drills to keep in practice, improve process
- Results:
  - Encourages transparency, communication
  - Small problems addressed, fewer big problems
  - Over-all uptime improved
There are only Contributing Factors

John Allspaw
http://www.kitchensoap.com/2012/02/10/each-necessary-but-only-jointly-sufficient/
After the outage, publish a postmortem document

- People involved write a “blameless postmortem”
  - Identifies what happened, how, what can be done to prevent similar problems in the future.
  - Published widely internally and externally.
- Instead of blame, people take responsibility:
  - Responsibility for implementing long-term fixes.
  - Responsibility for educating other teams how to learn from this.
Summary:
On Aug 25, 2014 there was an outage of all web properties (Core and Careers) from 3:27pm to 3:32pm NYC-time (approx 7 minutes). The cause was an incorrect change to security settings. The solution was to revert the change via Puppet. Measures being implemented to prevent this problem in the future are listed below.

<table>
<thead>
<tr>
<th>Outage Type</th>
<th>Sites Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outage Timeframe</td>
<td>2014-08-25 19:24, about 7m of downtime</td>
</tr>
<tr>
<td>Worst-Case Outage window</td>
<td>7 minutes</td>
</tr>
<tr>
<td>Assets affected</td>
<td>All</td>
</tr>
<tr>
<td>Summary of causes</td>
<td>Bad change to firewall rules.</td>
</tr>
<tr>
<td>Recommendations</td>
<td>Need to refactor firewall rules to be more easy to understand and update; Need to develop better testing methods for firewall rulesets.</td>
</tr>
</tbody>
</table>

Background Information
The intended change: SRE was attempting to update the firewall rules to permit internal openid calls to work directly rather than going out to the internet and back in.
Outage Schedule of Events

2014-08-25 19:01  da2d38d6a Change pushed to Git
2014-08-25 19:26  Puppet runs on ny-lb05 (pushed bad change / outage BEGINS)
2014-08-25 19:27  Pagerduty and Pingdom page oncall (Tom)
2014-08-25 19:27  @David asked "Who broke everything but chat?" on SRE-team
2014-08-25 19:27  da2d38d6a1 Revert pushed to Git
2014-08-25 19:30  Puppet runs on ny-lb06 (pushed revert)
2014-08-25 19:32  Puppet runs on ny-lb05 (pushes revert) (outage RESOLVED)
Things that went Right

- Use of version control with Puppet means we are able to revert bad changes quickly.
- Everyone worked together to find and fix the problem.

Processes Needing Improvement

- Firewall rules should be refactored to be easier to understand and update.
- Firewall rules need a better testing method.

Immediate to do

- Improve comments in iptables files (there are wrong and misleading comments)
  (Done: b55e654d9f)

Long term to do

- Move LB firewalls to the new structure being developed.
- Establish better testing methodology for firewall changes.
I dunno about anybody else, but I really like getting these post-mortem reports. Not only is it **nice to know what happened**, but it’s also great to see **how you guys handled it** in the moment and **how you plan to prevent these events** going forward. Really neato. Thanks for the great work :)

—–Anna
Fail Better Part 3 of 3:

Don’t punish people for outages.
Take-homes

• “cloud computing” = “distributed computing”

1. Use cheaper, less reliable, hardware
   • Create reliability through software (when possible)
   • Pay only for the reliability you need

2. If a process/procedure is risky, do it a lot
   • Practice makes perfect
   • “Small Batches” improves quality and morale

3. Don’t punish people for outages
   • Focus on accountability and take responsibility
Home Life
AWKWARD MOMENT
Radical Ideas from
The Practice of Cloud
System Administration

Tom Limoncelli, SRE
StackExchange.com
the-cloud-book.com
@YesThatTom
Very Reasonable

Radical Ideas from

The Practice of Cloud System Administration

Tom Limoncelli, SRE
StackExchange.com
the-cloud-book.com
@YesThatTom
If you liked this talk…

…there’s more like it in
http://the-cloud-book.com

Save 35%
www.informit.com/TPOSA
Discount code TPOSA35
Q&A
ACM: The Learning Continues…

• Questions about this webcast? learning@acm.org

• ACM Learning Webinars (on-demand archive): http://learning.acm.org/webinar

• ACM Learning Center: http://learning.acm.org

• ACM SIGMIS: http://sigmis.org/

• ACM Queue: http://queue.acm.org/