

Take Risks…

But Don’t Be Stupid!

Patrick Eaton, PhD
preaton@google.com

Take Risks…
But Don’t Be Stupid!

Patrick R. Eaton, PhD
patrick@stackdriver.com

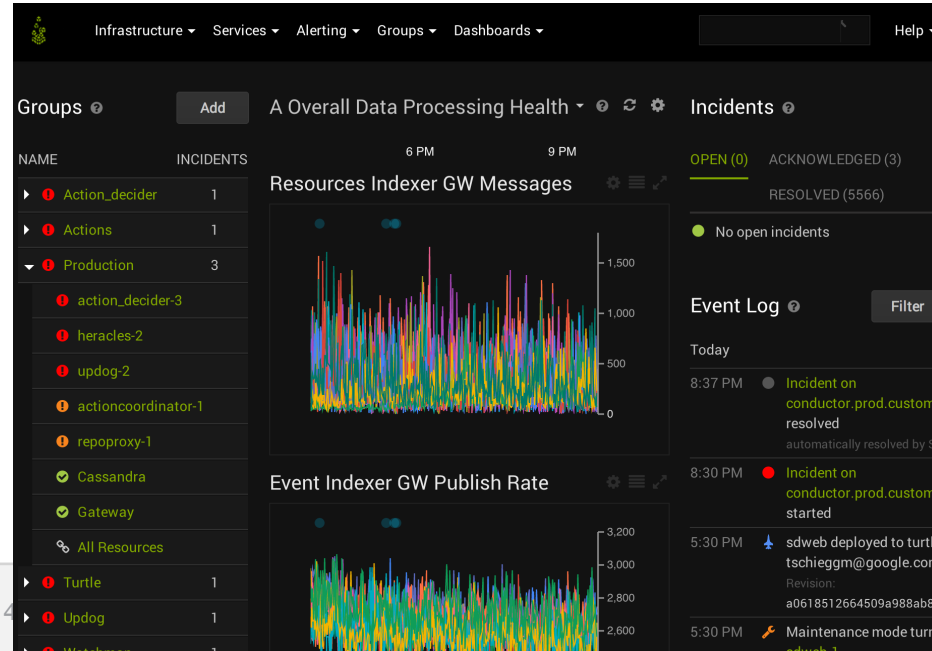


STACKDRIVER

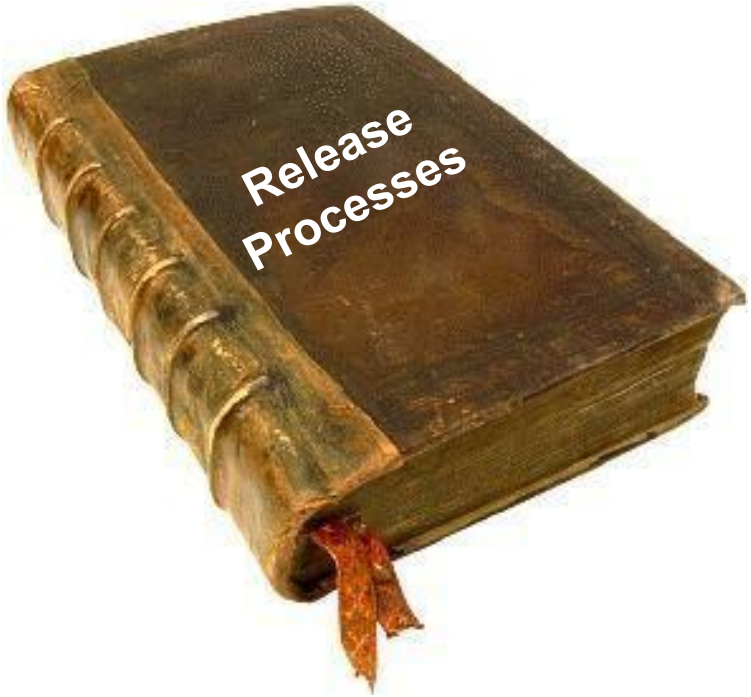
Stackdriver

- A hosted service providing intelligent monitoring to help SaaS companies innovate more by reducing the burden of day-to-day operations.
  - Cloud-native and cloud-aware
  - Designed for complex distributed applications

- Found August 2012 by Izzy Azeri and Dan Belcher
- Team of ~25, based in Boston
- Acquired by Google in May 2014



Some Software Cultures Avoid Risks

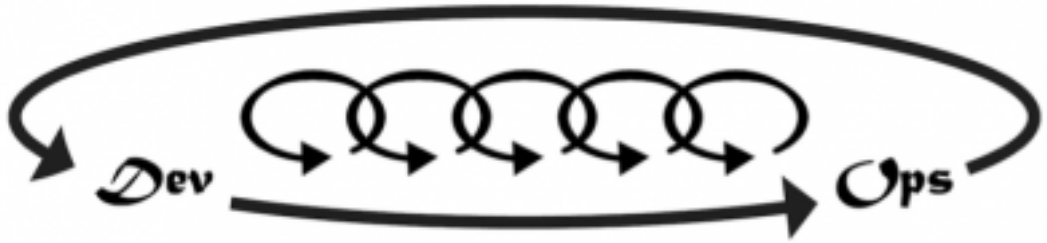


- Long release cycles
- Long QA cycles
- Lots of process
- High cost for mistakes

DevOps Movement Embraces Risk

Risk-taking is a foundational principle.
Kim, Behr, Spafford call it the “Third Way”.

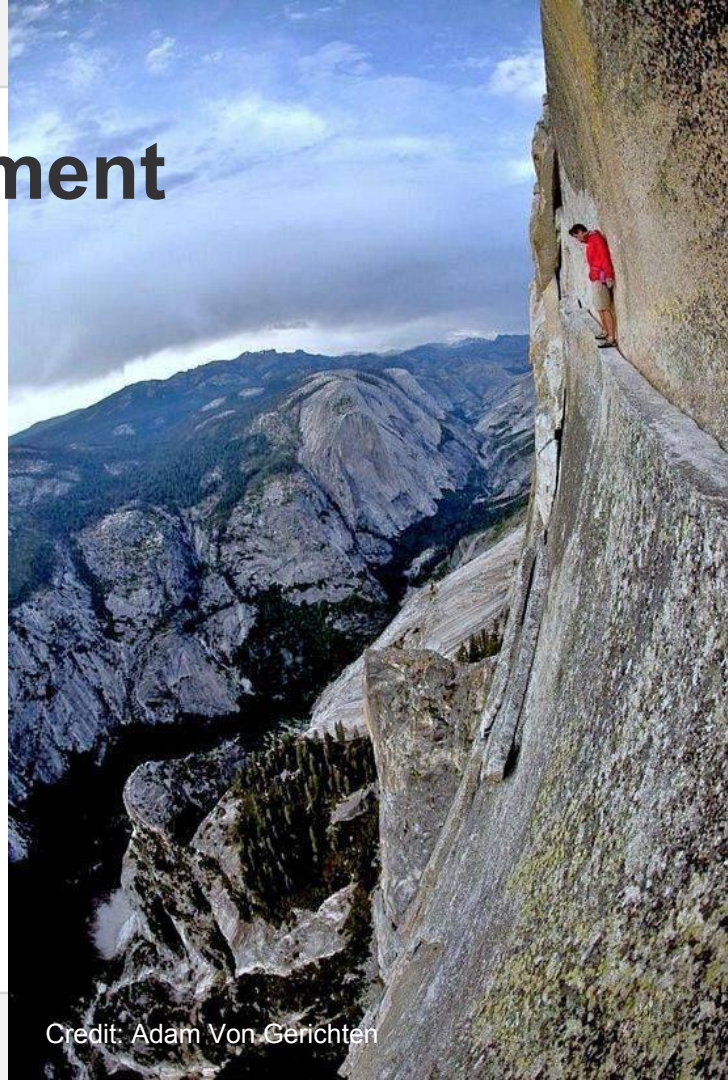
- Experiment; take risks and learn from failure.
- Use practice and repetition to achieve mastery.



source: itrevolution.com

Risk Taking Requires Judgement

- Balance risk and reward
- Take risks to push boundaries
- Retreat when you cross into the danger zone



Credit: Adam Von Gerichten

Google

Goals

A healthy view of risk-taking

How to design systems so that the impact of failures can be managed

Examples from Stackdriver of cost-conscious experimentation



source: kabuki00.pinger.pl

Are You Ready for Some Football?

Super Bowl XLVII - February 3, 2013

Baltimore Ravens vs. San Francisco 49ers

Won by Ravens 34-31





Are You Ready for Some Football?

Super Bo

Baltimore

Won by P





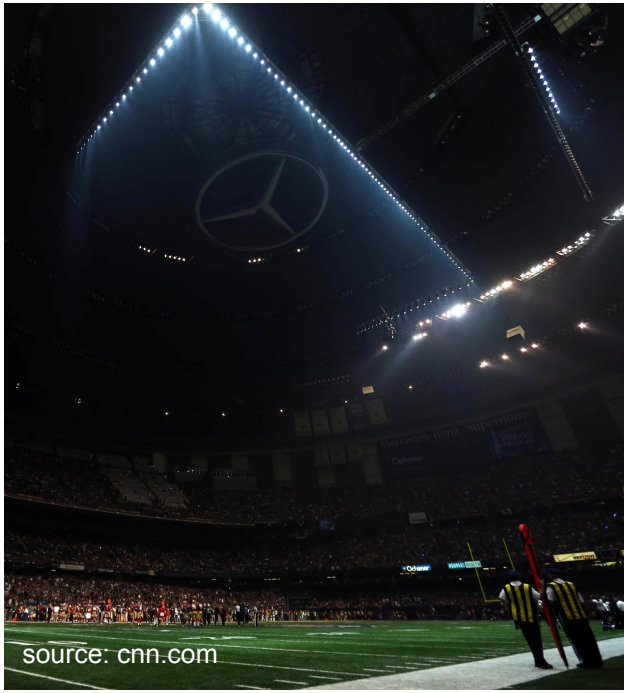
Strategies for Fault Mitigation

James Hamilton - Vice President and Distinguished Engineer on the Amazon Web Services

Blogged “The Power Failure Seen Around the World”

- http://bit.ly/1tbgBPy

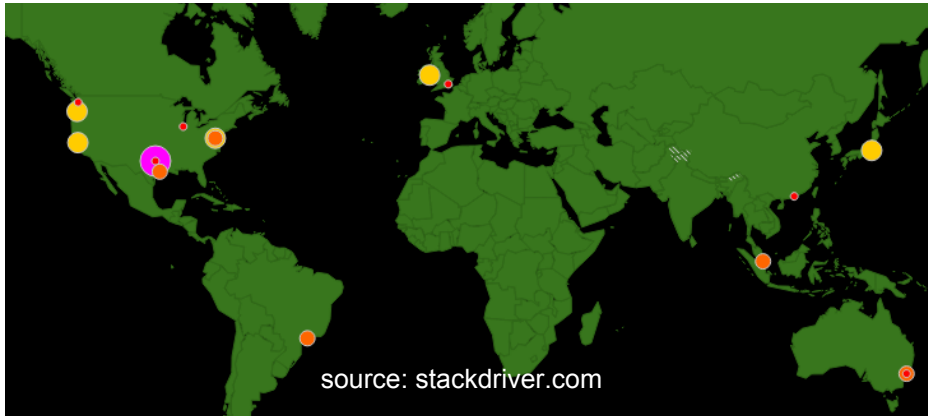
As when looking at any system faults, the tools we have to mitigate the impact are:
1) avoid the fault entirely,
2) protect against the fault with redundancy,
3) minimize the impact of the fault through small fault zones, and
4) minimize the impact through fast recovery.



Cloud Fault Domains

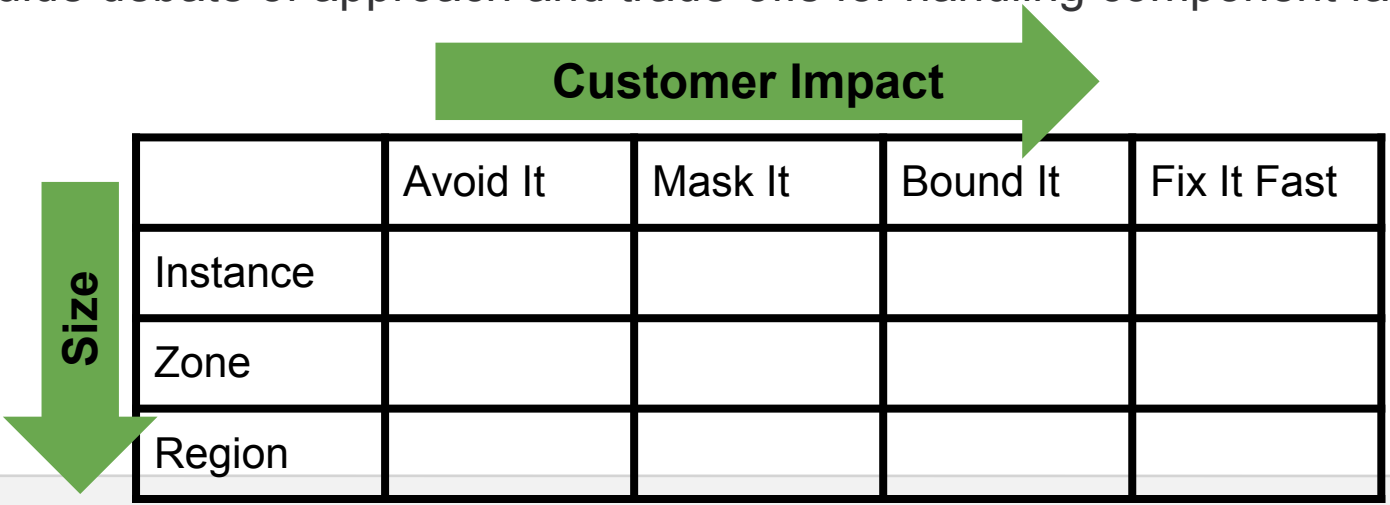
Fault Domain - group of resources that share a single point of failure. Resources in different fault domains **fail independently**.

Instance - A single virtual resource.
Zone - A sub-collection of resources in a region, typically a data center.
Region - A geographic area, often comprised of multiple data centers.
(**Provider** - Viable alternatives are emerging.)



“The Four Hamiltons”

- Framework for Fault Mitigation in the Cloud
  - High Scalability, http://bit.ly/1lP817l
- Cross Hamilton’s mitigation strategies with cloud fault domains.
- Guide debate of approach and trade-offs for handling component failures.



Avoid It!



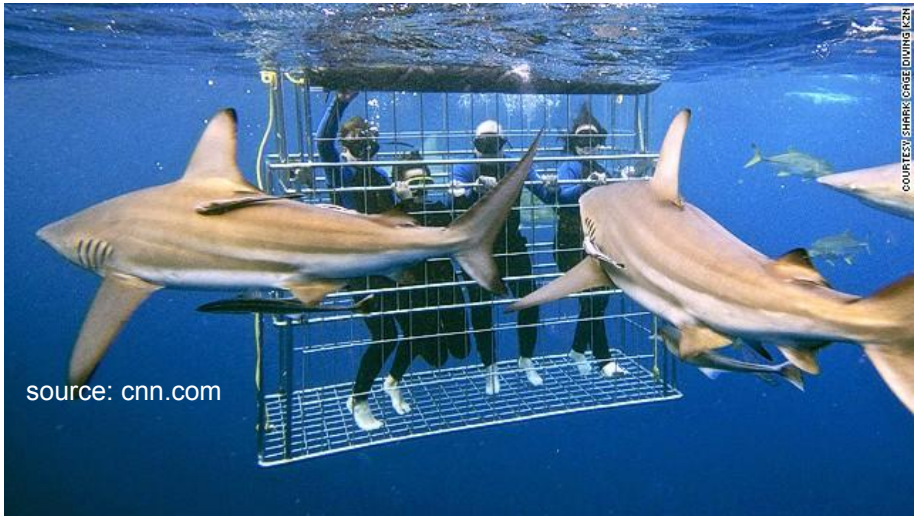
Formerly, “enterprise-grade” (expensive) hardware.
Now, solid architecture and good software engineering.

Techniques:

- Write good code. Test it thoroughly.
- Use high-quality software components (web servers, databases, etc.).
- Let someone else do it.
  - Use hosted or managed services that “do not fail”.
  - Our favorites include AWS RDS, AWS ELB, AWS SQS.

Bound It!

Minimize scope of the failure to reduce customer impact.



Techniques:

- Limit impact by sharding.
- Degrade gracefully.
  - Architect different subsystems/features to be independent.
  - Browse without search, download without upload, use cached results.

Mask It!

Use redundancy or replication to avoid customer impact.



Techniques:

- Use pools of peers/workers handling similar work.
- Master/slave, primary/secondary - with automatic failover.
- Clustering, quorums, gossip, peer-to-peer routing.

Fix It Fast!



Don’t rely on this strategy;
You are “doing it wrong!”

Techniques:

- Revert code.
- Provision and deploy new resources.
- Restore from replicas or back-ups.

Implement documented recovery procedures.

- Practice!!!

Switching Gears

A healthy view of risk-taking

The “Four Hamiltons” framework for designing robust architectures

Examples from Stackdriver of cost-conscious experimentation



About the Stackdriver Infrastructure



Key components:

- **Data collection** - querying cloud provider APIs
- **Ingest pipeline** - archiving/indexing billions of messages daily
- **Alerting subsystem** - evaluate user-defined policies
- **Batch processing** - aggregation and analysis
- **UI** - powerful graphing and visualization capabilities
- **Custom automation framework**

Technology:

- Django, Angular, Python, Cassandra, ElasticSearch, MySQL, Rabbit, Puppet
- Heavy use of hosted services: ELB, RDS, SQS, and SNS

Several hundred instances running in AWS.

~50 deployable units, pushing dozens of releases per day.

Stackdriver Ingest Pipeline

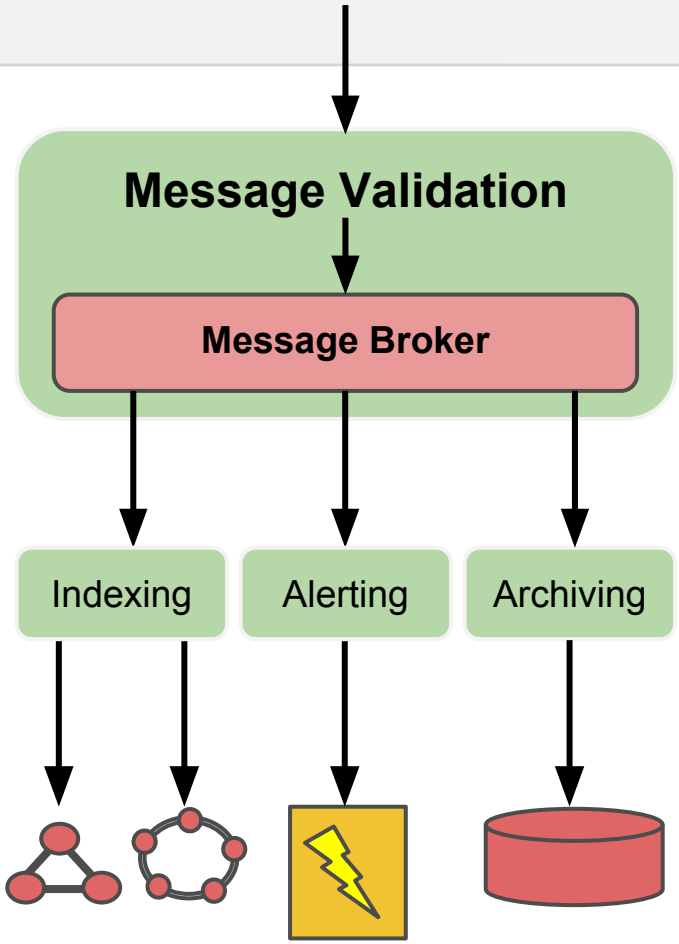
Purpose: Take data off the wire and get it where it needs to go.

Performed by set of cooperating components.

- Messaging with RabbitMQ
- Archive to S3
- Drive the custom alerting pipeline
- Index to Cassandra, ElasticSearch

Designed/built to tolerate instance failure.

- Strongly decoupled
- Multiple points for buffering



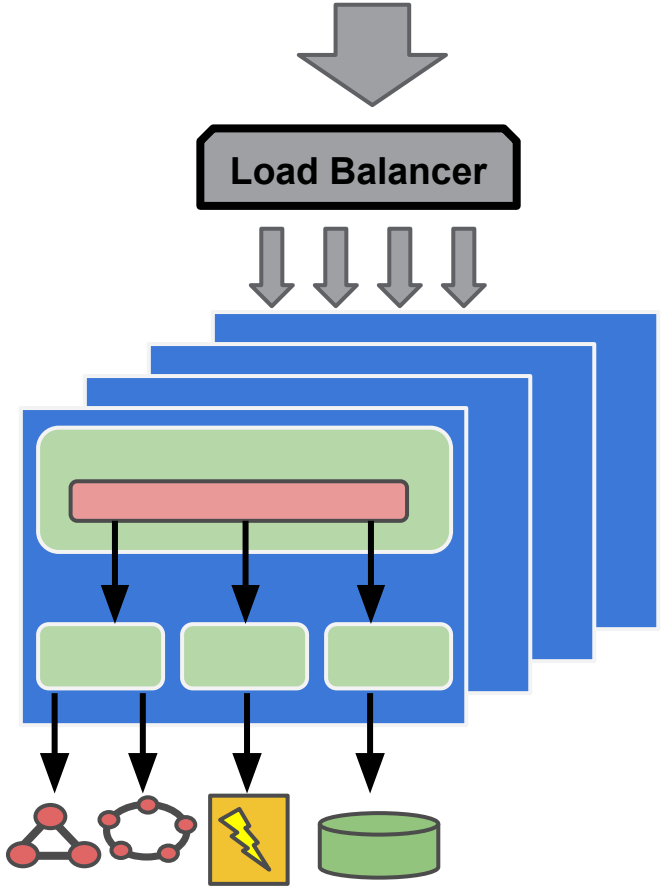
Scaling the Ingest Pipeline

A **cell** is...

- the set of components needed to process a single message,
- the unit of scaling,
- independent from other cells,
- composed of instances in a single zone (tolerates zone failures).

Much automation supports cell-based design.

Data sinks (C*, ES, S3) handle full load.



Innovate Ingest at Scale

Must continue to build, debug, fix, maintain, and enhance running pipeline.

“Big” data problem characterized by 3Vs

- variety, volume, velocity

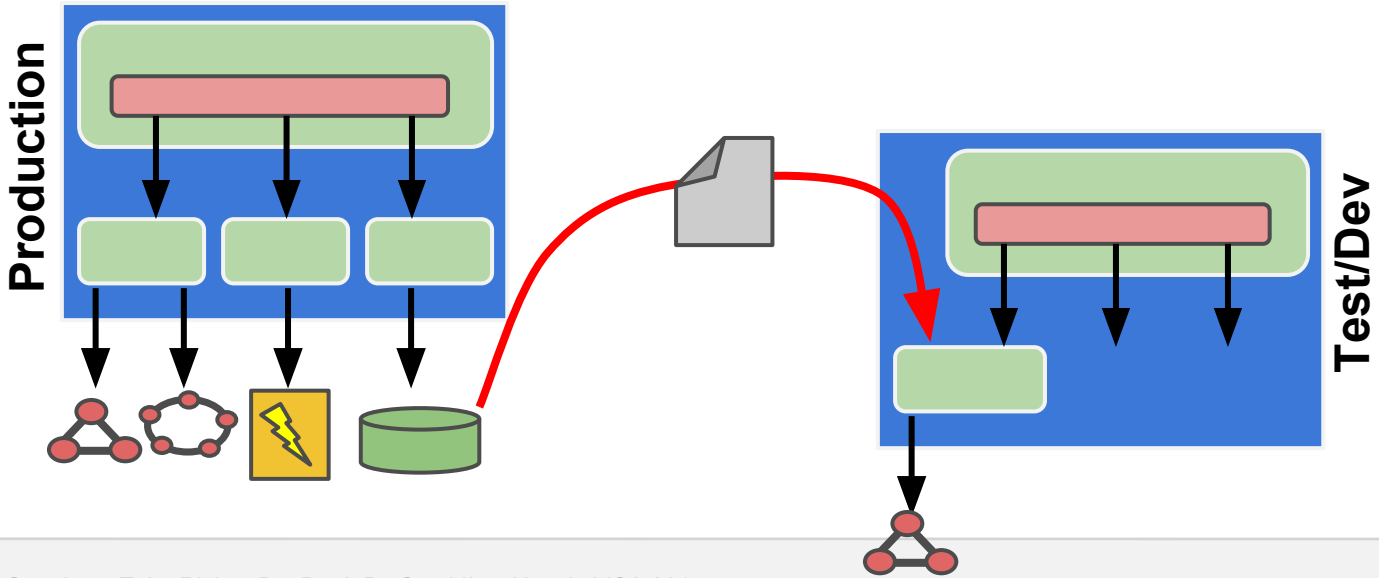
But resources are scarce.

- Money, time, dev resources, ops overhead.
- Cannot simply deploy one of everything in a test environment.


source: lovethesepics.com

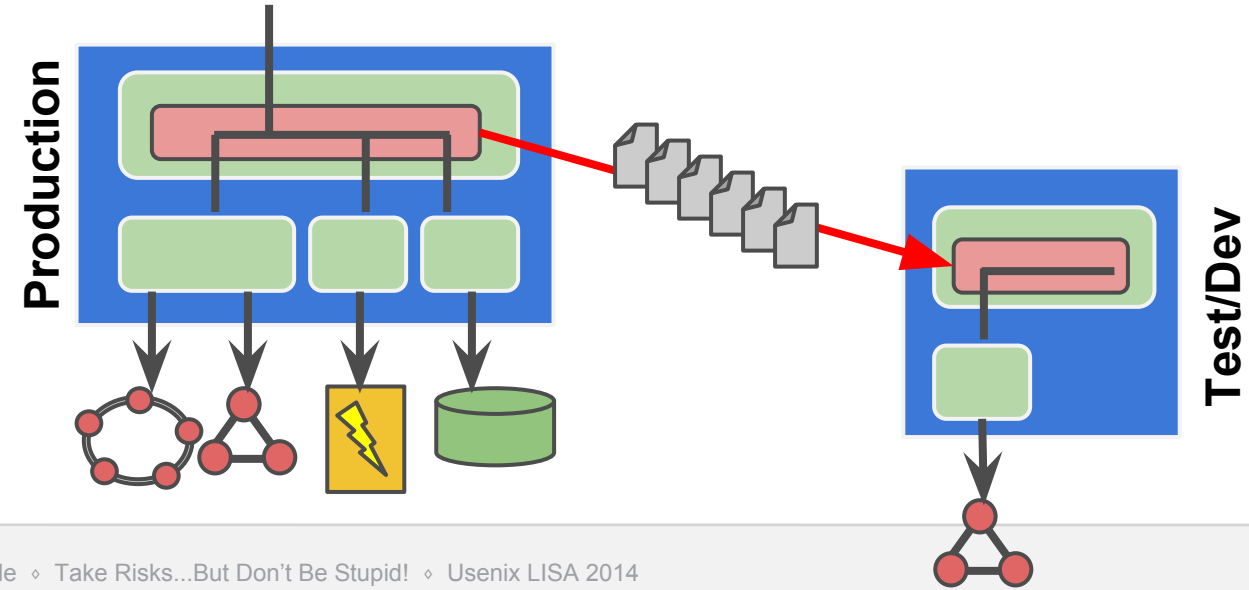
Pipeline Testing for Variety

- Expose test environment to full variety of data.
- Replay raw data stored in archive.



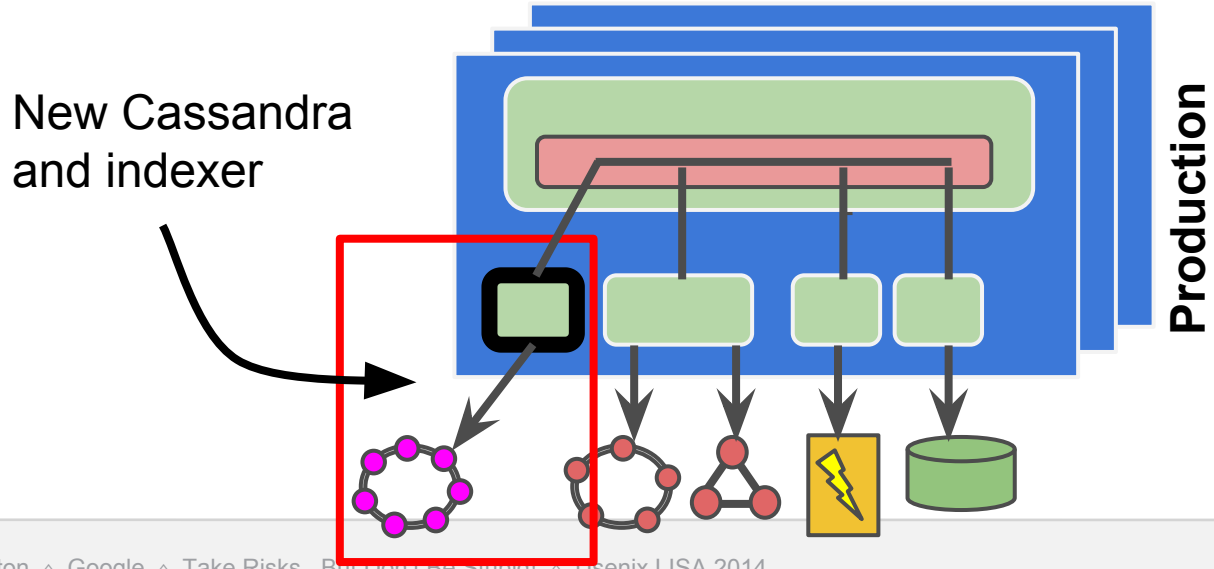
Pipeline Testing for Velocity

- Expose a single cell to the load of a cell at line speeds.
- Federate traffic from the message broker in one cell to cell.



Pipeline Testing for Volume

- Expose downstream components to full system load.
- Add another consumer of the message broker in each cell.



Challenges

- Access control
  - Components in test account have only read-only access to data
  - Cross-account IAM
- Manage access to relational data
  - Need to access config from prod
  - Copy any mutable config
- Automation



Conclusions

Risk-taking is an important strategy for innovation, but requires cultural support

Good system design is a safety net that helps protect you when experiments fail

Use production systems and data to perform high-fidelity tests at low cost





Take Risks…

But Don’t Be Stupid!

Patrick Eaton, PhD
preaton@google.com

Thank You!
Questions?

Patrick R. Eaton, PhD
patrick@stackdriver.com



STACKDRIVER