Detecting Capacity Limits and Performance Bottlenecks Using Live Traffic

Susie Xia

Christopher Coleman

2018 QCon Beijing

Agenda

1	Introduction
2	Meet Redliner
3	Use Cases
4	Future Plans

LinkedIn Engagement & Growth

546M

Members

- 20M Companies
- 14M+ Open Jobs
- 29K+ Schools
- 11B+ Endorsements

20%

Sessions Growth (YoY)

- 5th straight quarter of this growth
- Record levels of engagement
- 60% (YoY) growth in viral actions, such as likes, comments, shares, and messages sent

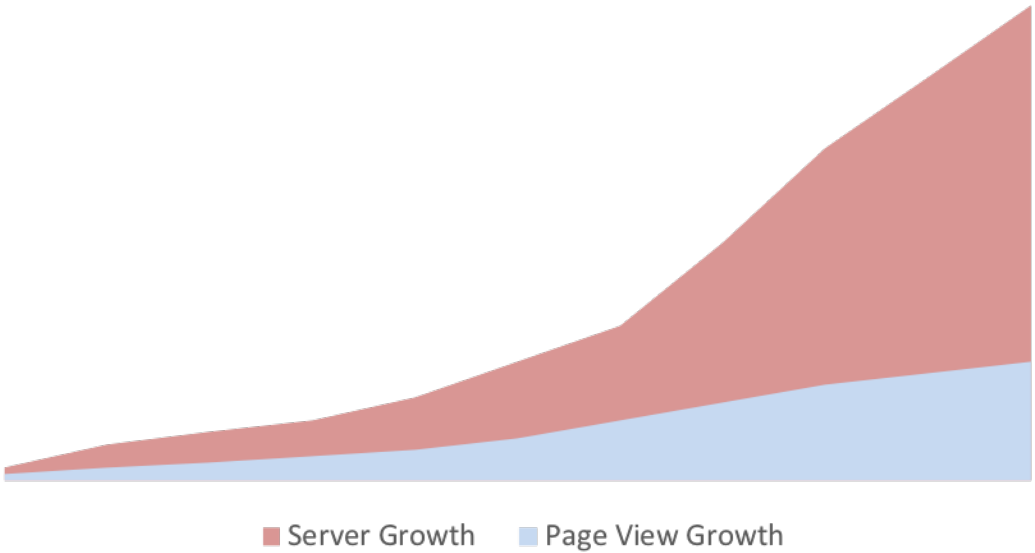
200+

Countries & Territories

- Available in 24 languages
- 70% members outside of US
- > 2+ new users join per second

Our Dilemma

WHY IS SERVER GROWTH OUTPACING PAGE VIEW GROWTH?



Over Provisioning

- Organic Growth
- Unexpected Events
- New Products & Features
- Emergency Uplifts

Motivations

- Resource Efficiently
- Capacity Plan Effortlessly
- Increase Throughput Reliably

Challenges

- External Interferences
- Evolving Product Landscapes
- Complex Downstream Dynamics

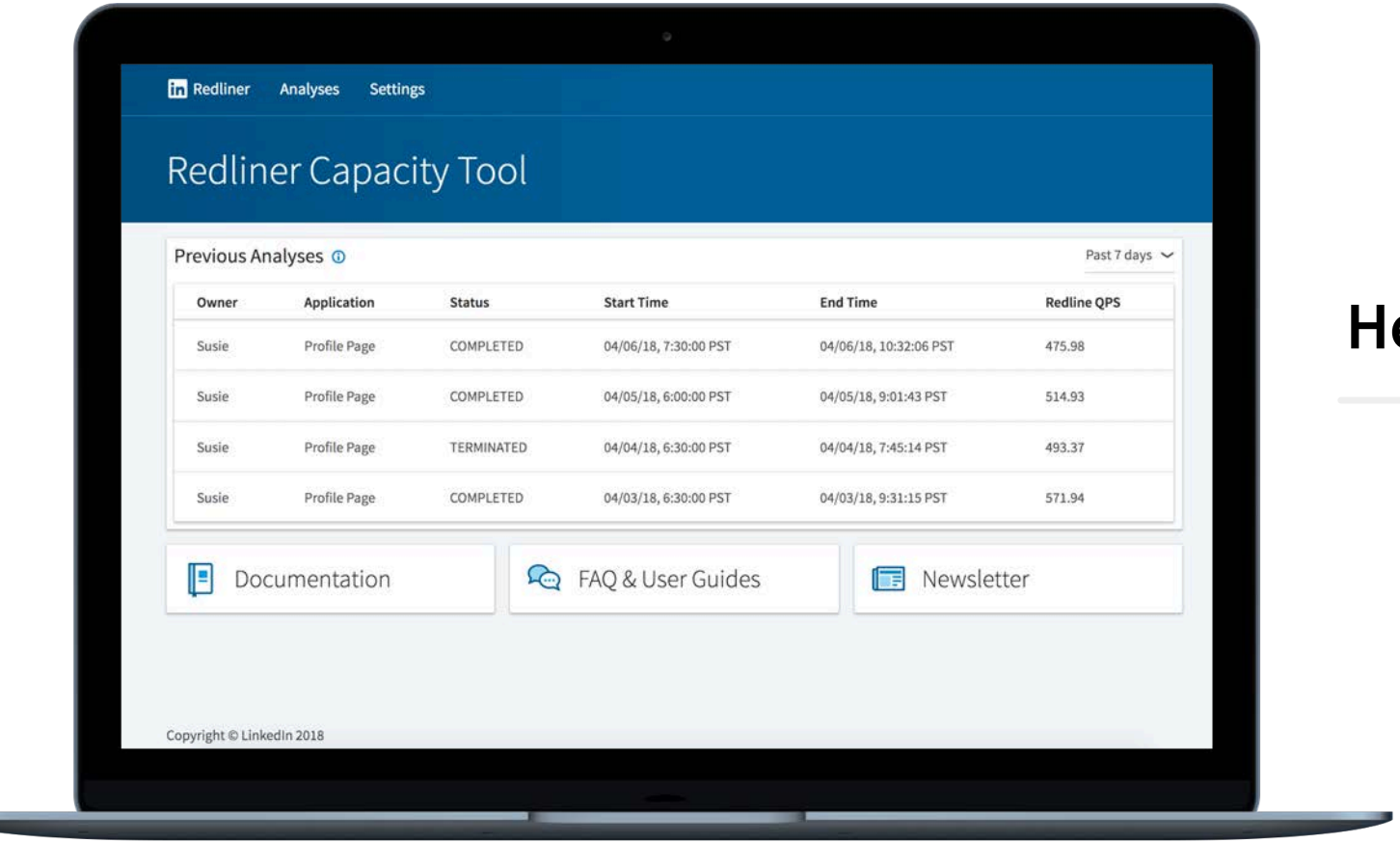
Load Testing Journey

Synthetic Load in Lab → Synthetic Load in Prod → Isolated Host → Record & Replay → Anything Else?

Learnings

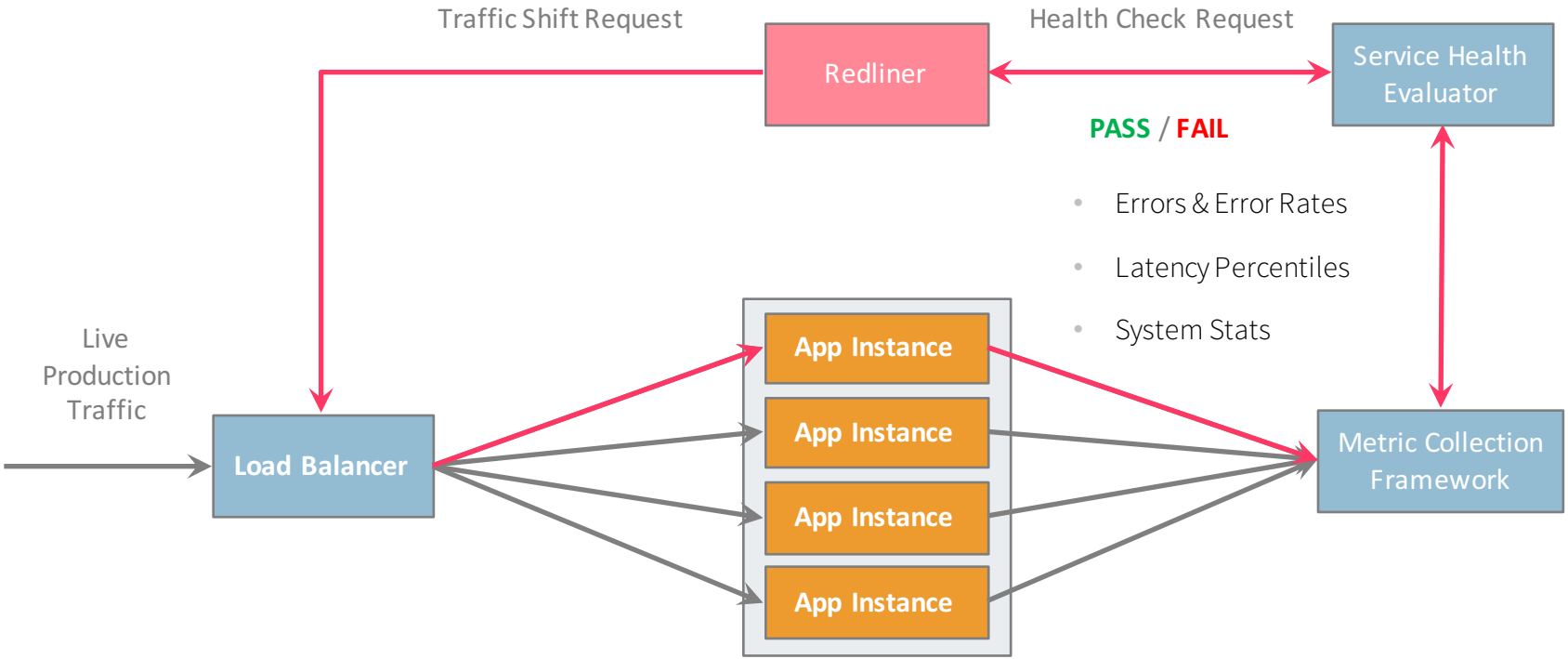
Goals

- Use Live Production Traffic
- Minimize Impact to Users
- Require Low Operational Overhead



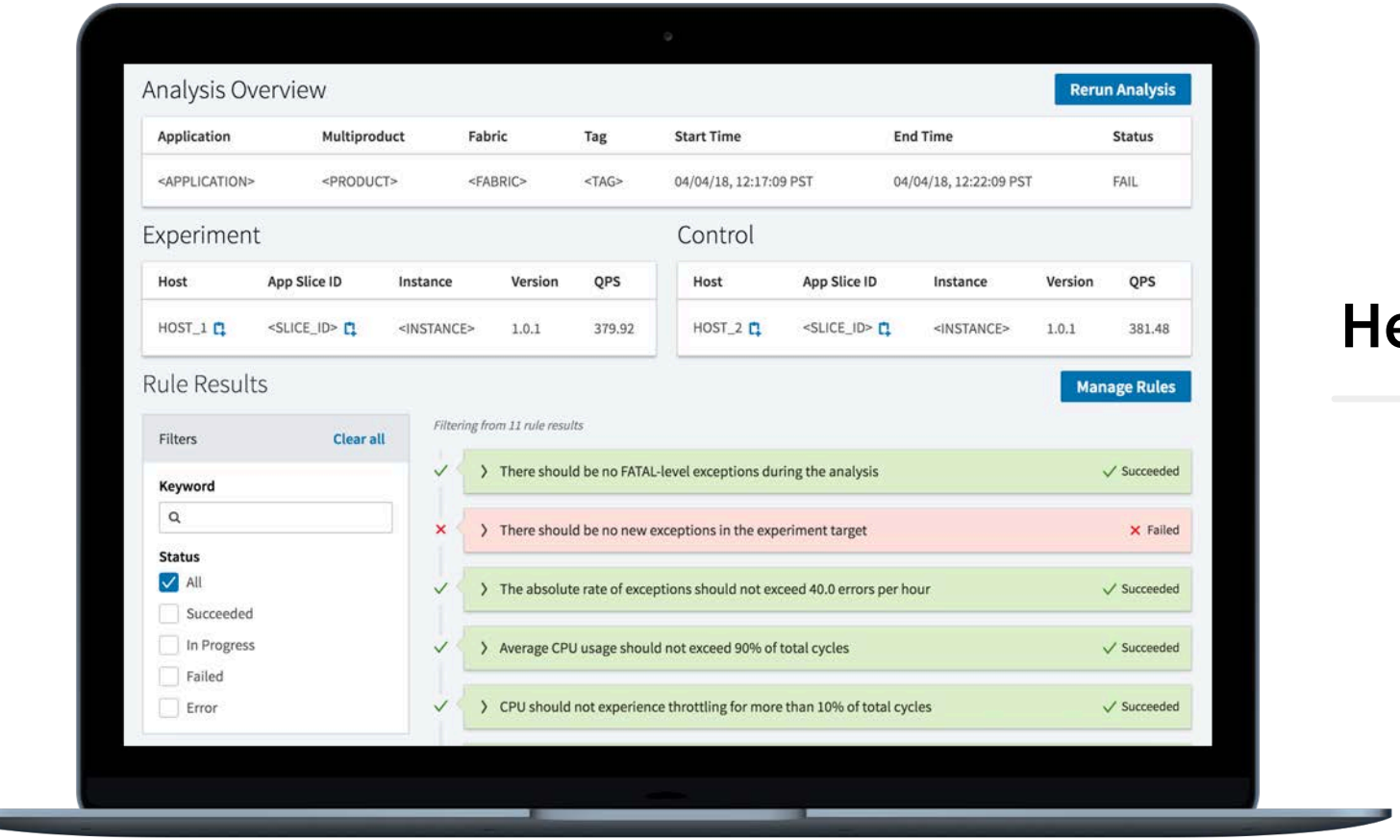
Hello, Redliner

Workflow


Traffic Shift Request
Redliner
Health Check Request
Service Health Evaluator
PASS / FAIL
Errors & Error Rates
Latency Percentiles
System Stats
Live Production Traffic
Load Balancer
App Instance
App Instance
App Instance
App Instance
Metric Collection Framework

Health Evaluations

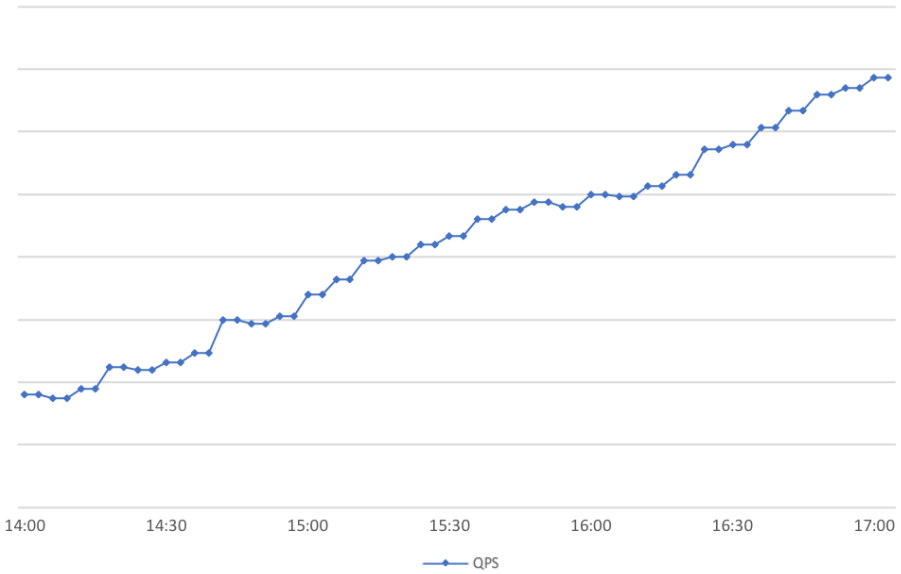
- Variety of health checks measured every set interval
- Evaluations at the host, cluster, and data center levels
- Incorporates signal from operational alerting system
- Performance comparisons between target and the cluster



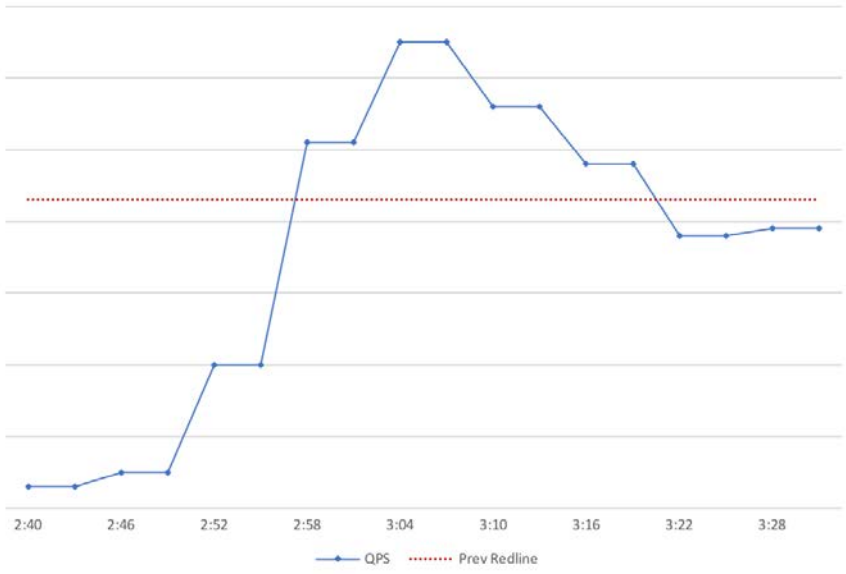
Health Checks

Dynamic Ramping

Slow, Steady Ramp



Fast, Aggressive Ramp

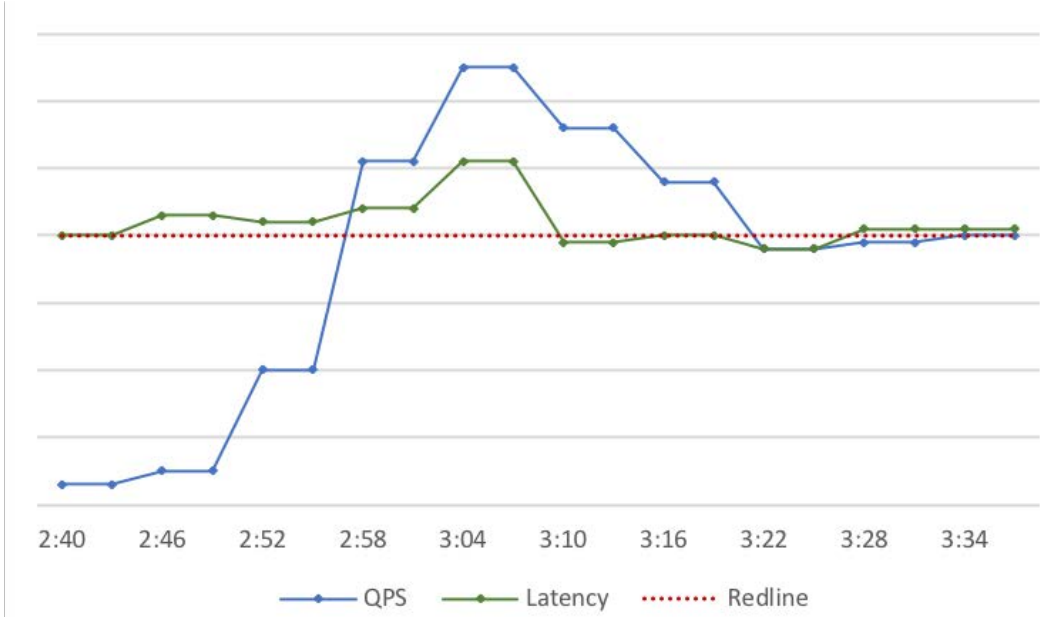


Complete Automation

- Manipulation of traffic between nodes in the cluster
- Determination of the node's and service's health
- Identification of potential bottlenecks under stress
- Remediation of any issues encountered during test

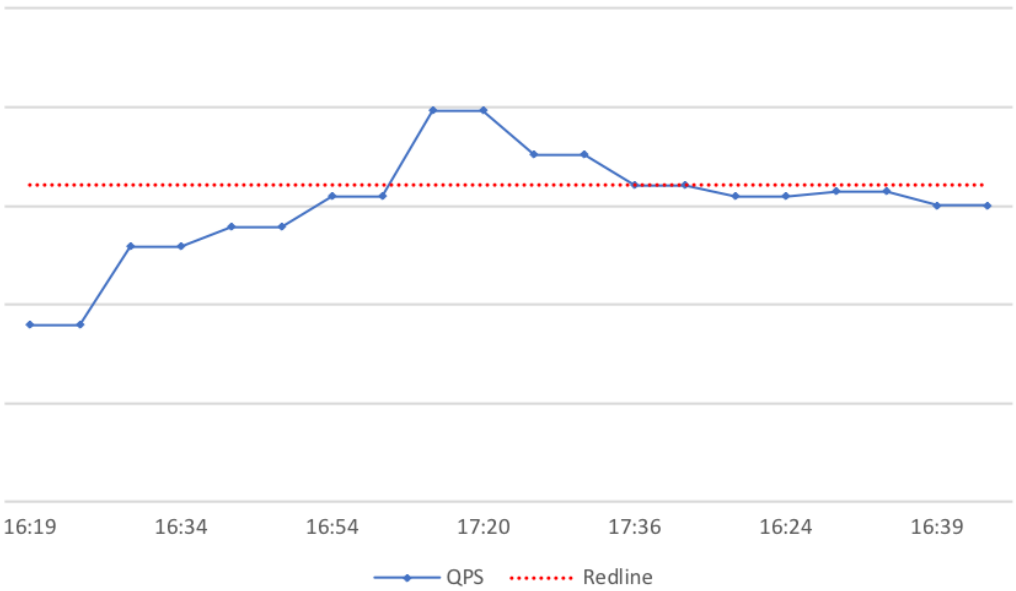
Use Cases

1. Find Single Instance Max Throughput



- Gradually stresses the service until it cannot safely handle any additional load
- Simplifies resource provisioning
- Provides starting point for tuning and optimizations

2. Improve Service Throughput



- Investigate health check failures from increased traffic
- Discover APIs “A”, “B”, “C” error rates jumped
- Caused API “D” latency to double
- Resolve issues one by one
- Repeat the Redliner test

Before Investigation

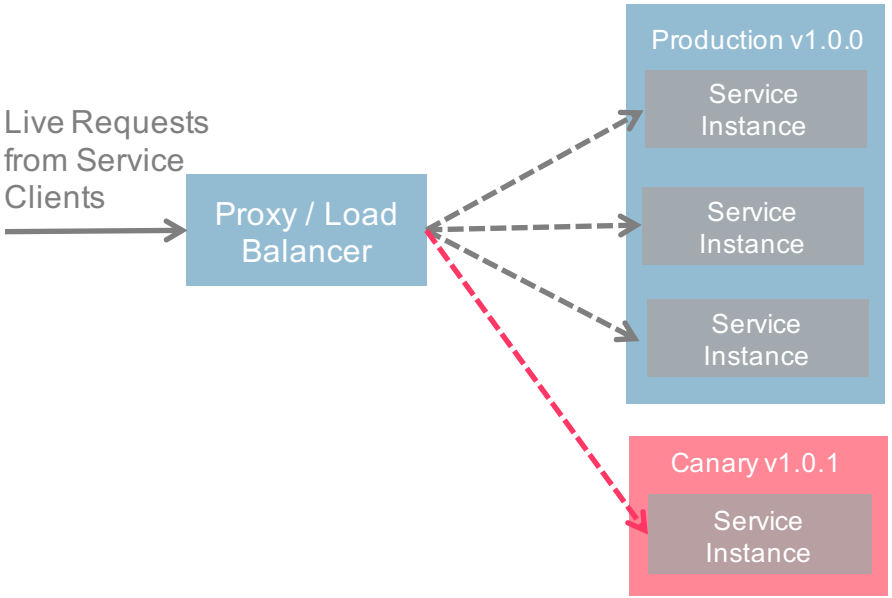
After Investigation

3. Detect and Diagnose Regressions

Test Id	Date	Version	Redline	Health Check Failures in Latency
Test 1	2017-11-19 09:01:11	v1.0.0	2536.33	• N/A
Test 2	2017-11-19 23:58:09	v1.0.1	534.19	• Endpoint A: Median latency exceeded 20% change in comparison to control target. • Endpoint B: Median latency exceeded 20% change in comparison to control target.

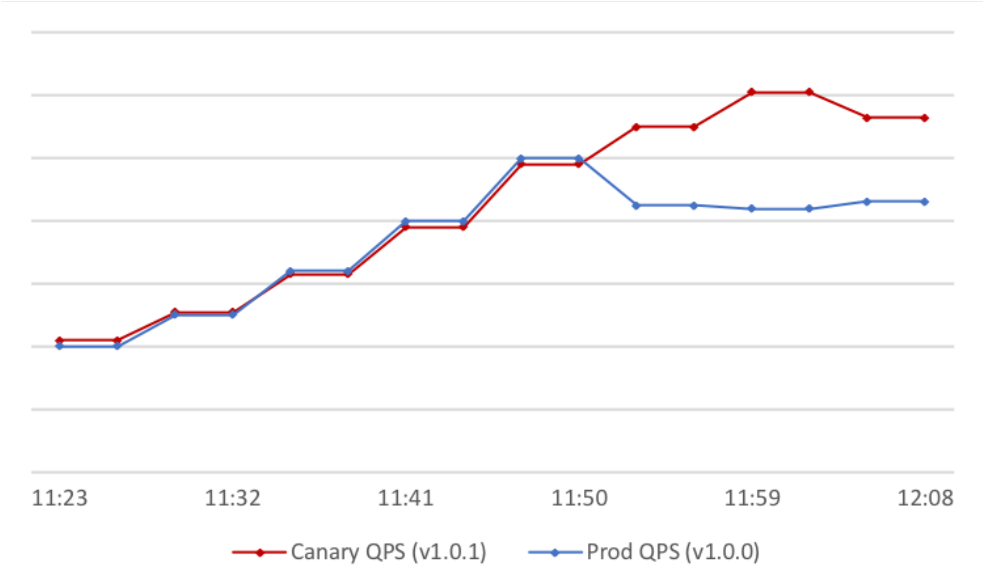
The Smiley Curve

4. A/B Load Testing



- Run Redliner test side-by-side on canary and production versions
- Code comparisons
- Configuration comparisons
- OS comparisons
- Security updates

A/B Load Test Example



- Same load on both canary and prod instances until one or both failed health check
- Prod instance hits health check failure *before* canary instance
- v1.0.1 on canary has better throughput – new version is encouraged to be deployed

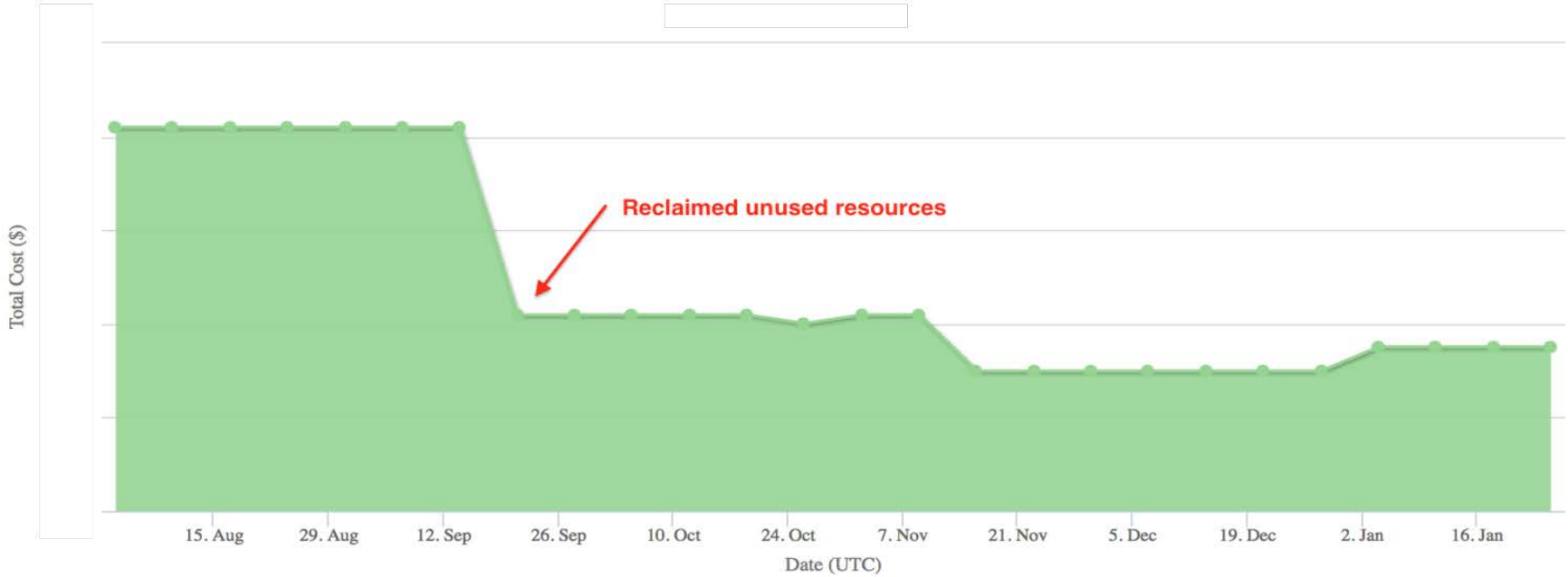
5. Identify Surplus Capacity

When $\boldsymbol{Redline\ QPS} < \boldsymbol{Total\ Service\ QPS}$, $Ideal\ \#\ of\ Instances = \frac{Total\ Service\ QPS}{Redline\ QPS} + Headroom$

When $\boldsymbol{Redline\ QPS} \geq \boldsymbol{Total\ Service\ QPS}$, $Ideal\ \#\ of\ Instances = 1 + Failover\ Headroom$

If $Total\ \#\ of\ Deployed\ Instances > Ideal\ \#\ of\ Instances$,
the service is ***over-provisioned***.

Server Cap Ex Trend for Service



Future Work

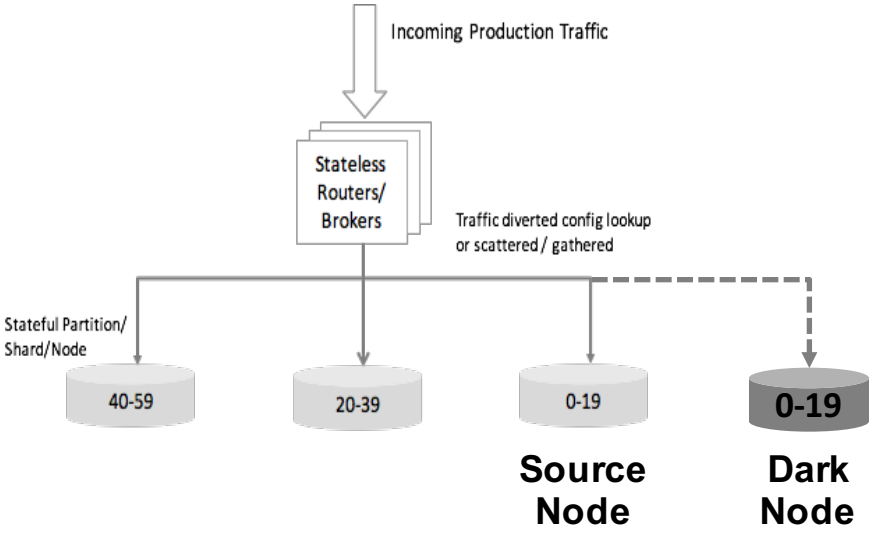
1. Dynamic Provisioning

- Auto Scaling – Scale predictably to handle natural changes in traffic throughout the day
- Efficient Host Packing – Create models for throughput based on resource allocations and deploy most efficient container size

2. Simulating Downstream Behavior

- Latency – Test against response times during peak traffic hours at any time in the day
- Errors & Failures – Test service behavior when downstream results are acting unreliably
- Connectivity – Test resiliency and recovery when dependencies are unavailable

3. Stateful Redlining



- **Source Node** – Storage node to test
- **Dark Node** – Exact replica of source node
- **Tee Traffic** – Copy the incoming live traffic to source node to dark node
- **Multiply Traffic** – Generate extra load on dark node based on incoming traffic

Key Takeaways

Reflection

- Don’t Be Afraid of Risk
- Prepare for the Surprises
- Build Performance Mindset

Don’t count servers.
Make servers count.

Thank you

https://engineering.linkedin.com/blog

chinajobs@linkedin.com

主办方 Geekbang. 极客邦科技 InfoQ

GMTC 2018

全球大前端技术大会

大前端的下一站

<<扫码了解更多详情>>