An Architect's Guide to Site Reliability Engineering

Nathaniel T. Schutta
@ntschutta

Software development practices evolve.

Feature not a bug.

It is the agile thing to do!

We've gone from devs and ops separated by a large wall…

To DevOps all the things.

We’ve gone from monoliths to service oriented to microserivces.

And it isn’t all puppies and rainbows.

Shoot.

A new role is emerging - the site reliability engineer.

Why?

What does that mean to our teams?

What principles and practices should we adopt?

How do we work together?

What is SRE?

Important to understand the history.

Not a new concept but a catchy name!

Arguably goes back to the Apollo program.

Margaret Hamilton.

ROBERT MCMILLAN BUSINESS 10.13.15 7:00 AM

HER CODE GOT HUMANS ON THE MOON—AND INVENTED SOFTWARE ITSELF

Crashed a simulator by inadvertently running a prelaunch program.

That wipes out the navigation data.

Recalculating…

Hamilton wanted to add error-checking code to the Apollo system that would prevent this from messing up the systems. But that seemed excessive to her higher-ups. “Everyone said, ‘That would never happen,’” Hamilton remembers.
But it did. Right around Christmas 1968.

— ROBERT MCMILLAN

https://www.wired.com/2015/10/margaret-hamilton-nasa-apollo/

Luckily she did manage to update the documentation.

Allowed them to recover the data.

Doubt that would have turned into a Hollywood blockbuster…

Hope is not a strategy.

But it is what rebellions are built on.

Failures, uh find a way.

Traditionally, systems were run by sys admins.

AKA Prod Ops. Or something similar.

And that worked OK. For a while.

But look around your world today.

Service all the things!

Seems like everything is aaS these days…

Infrastructure. Container. Platform.
Software. Function. Pizza.

Pretty sure about that last one…

Architecture as a Service…

Nothing new really.

CORBA anyone?

Facilitate communication for disparate systems on diverse platforms…

Too soon?

EJB then.

Still have flashbacks?

Sorry. I’ve tried to forget most of it.

It didn’t stop there though did it?

Remember when SOA was the new hotness?

Everything had to be all SOA all the time!

There were books and talks and blogs (remember those?), it was great!

How about API first?

Popularized in certain quarters.

sitorsquat.com

Powered by Charmin enjoy the go

Gotta Go When You're on the Go?

Now you can take sit or squat wherever you go!
if you already have the app, be sure to update it to the new version on your phone.

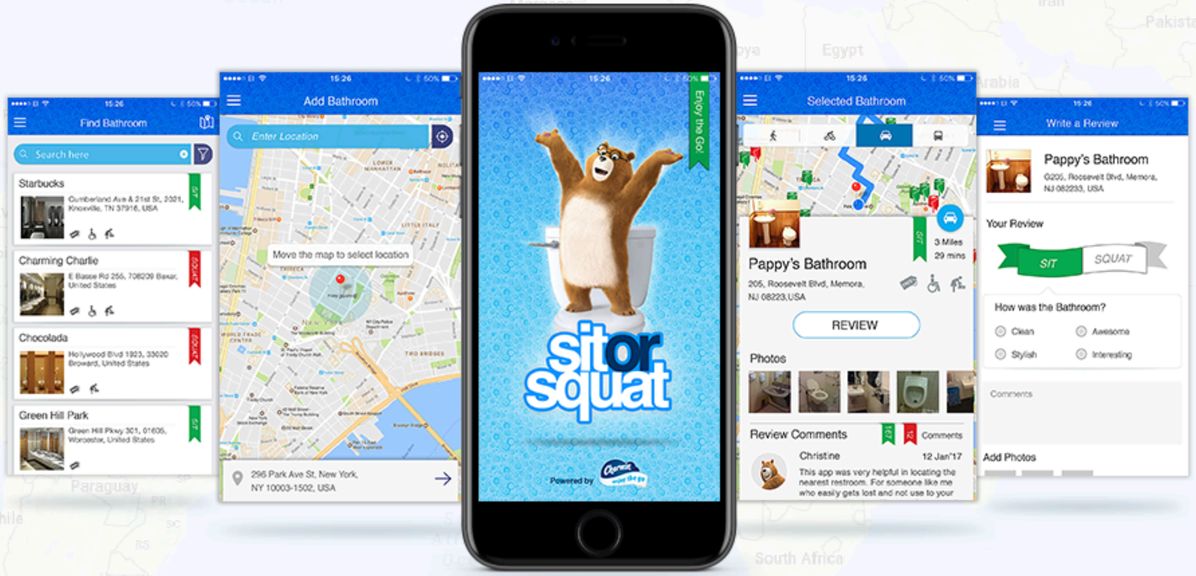
We have moved to an app-only version to give you a better experience when you gotta go. We hope you enjoy our new easy-to-go mobile app!

Download our improved SitOrSquat app that is available for FREE on: App Store | Google Play

Updates include new security upgrades and improved map functionality!
Check out more information on the SitOrSquat app by clicking here.

Find Charmin online! Charmin.com

Enjoy our Charmin Pooptastic iMessage stickers by clicking here.



Helpful?

How about this one?

darksky.net

Dark Sky API

DOCS FAQ LOG IN SIGN UP

Dark Sky API

The easiest, most advanced, weather API on the web.

TRY FOR FREE

Easy to Use

- Get setup in minutes
- Developer friendly
- Clear and concise documentation
- Pay-as-you-go pricing

Weather Conditions

- Forecasts and current conditions
- Global coverage
- Historical data
- Severe weather alerts

Advanced Data

- Cutting-edge forecast modeling techniques
- Minute-by-minute hyperlocal forecasts
- Backed by a wide range of data sources

LEARN MORE

Bet you use that one everyday.
Maybe without knowing it.

One of these perhaps?

developers.google.com

Google

Search for services, methods, and recent requests... Loading...

APIs Explorer

- Services
- All Versions
- Request History

Accelerated Mobile Pages (AMP) URL API	v1	This API contains a single method, batchGet. Call this method to retrieve the AMP URL (and equivalent AMP Cache URL) for given public URL(s).
Ad Exchange Buyer API	v1.4	Accesses your bidding-account information, submits creatives for validation, finds available direct deals, and retrieves performance reports.
Ad Exchange Buyer API II	v2beta1	Accesses the latest features for managing Ad Exchange accounts, Real-Time Bidding configurations and auction metrics, and Marketplace programmatic deals.
Ad Exchange Seller API	v2.0	Accesses the inventory of Ad Exchange seller users and generates reports.
Admin Reports API	reports_v1	Fetches reports for the administrators of G Suite customers about the usage, collaboration, security, and risk for their users.
AdSense Host API	v4.1	Limited Availability Generates performance reports, generates ad codes, and provides publisher management capabilities for AdSense Hosts.
AdSense Management API	v1.4	Accesses AdSense publishers' inventory and generates performance reports.
Android Device Provisioning Partner API	v1	Automates reseller integration into zero-touch enrollment by assigning devices to customers and creating device reports.
Android Management API	v1	The Android Management API provides remote enterprise management of Android devices and apps.
APIs Discovery Service	v1	Provides information about other Google APIs, such as what APIs are available, the resource, and method details for each API.
BigQuery API	v2	A data platform for customers to create, manage, share and query data.
BigQuery Data Transfer API	v1	Transfers data from partner SaaS applications to Google BigQuery on a scheduled, managed basis.
Blogger API	v3	Limited Availability API for access to the data within Blogger.
Books API	v1	Searches for books and manages your Google Books library.
Calendar API	v3	Manipulates events and other calendar data.
Cloud Monitoring API	v2beta2	Accesses Google Cloud Monitoring data.
Cloud Source Repositories API	v1	Access source code repositories hosted by Google.
Cloud Spanner API	v1	Cloud Spanner is a managed, mission-critical, globally consistent and scalable relational database service.
Cloud SQL Administration API	v1beta4	Creates and configures Cloud SQL instances, which provide fully-managed MySQL databases.
Cloud Storage JSON API	v1	Stores and retrieves potentially large, immutable data objects.
Cloud Tasks API	v2beta2	Manages the execution of large numbers of distributed requests. Cloud Tasks is in Alpha.
Cloud Tool Results API	v1beta3	Reads and publishes results from Firebase Test Lab.
Cloud User Accounts API	vm_alpha	Creates and manages users and groups for accessing Google Compute Engine virtual machines.

What caused this Cambrian explosion of APIs?

Technology changes.

Apple Inc.

iPhone X

Overview iOS Tech Specs Models and Pricing

Proliferation of computers in everyone’s pockets.

Commoditized hardware.

The Cloud!

Companies were altering their approach too.

Little company called Amazon made a major change.

Well, it had happened years earlier.

But a publicly shared rant detailed it.

https://plus.google.com/+RipRowan/posts/eVeouesvaVX

Steve Yegge - the Bezos mandate.

All data will be exposed through a public service interface.

These interfaces are *the*
communication method between teams.

No other form of communication is allowed. No direct reads, links etc.

No back doors.

All services must be designed to be public. No exceptions.

Don’t want to do this? You’re fired.

Unsurprisingly, things began to change.

And we learned some things.

Calls bounce between 20 services…
where was the problem?

Who do we page?

How do we monitor this web of services?

How do I even *find*
these other services?

Debugging gets harder…

We’ve seen this story continue today.

Can’t swing a dry erase marker without hitting someone talking about…

Microservices!

Bounded Context.

Domain-Driven Design.

Arguments over the definition of microservice…

Rewrite it in two weeks.

Miniservice. Picoservice.

What do we even mean by "application" today?!?

How about functions then?

ENGINEERING

Introducing Spring Cloud Function

ENGINEERING MARK FISHER JULY 05, 2017 8 COMMENTS

Spring Cloud Function is a new project with the following high-level goals:

- Promote the implementation of business logic via functions.
- Decouple the development lifecycle of business logic from any specific runtime target so that the same code can run as a web endpoint, a stream processor, or a task.
- Support a uniform programming model across serverless providers, as well as the ability to run standalone (locally or in a PaaS).
- Enable Spring Boot features (auto-configuration, dependency injection, metrics) on serverless providers.

Just as Spring has always promoted a plain old java object (POJO) based programming model, Spring Cloud Function promotes a programming model based on plain old functions. By that we mean the core interfaces defined in the `java.util.function` package: `Function`, `Consumer`, and `Supplier`.

Implementations of those types can be registered as beans either explicitly or implicitly via classpath scanning enabled by `@FunctionScan`. The parameter and/or return types can optionally use Reactor's `Flux`, which is a Reactive Streams `Publisher`. That enables interoperability with other Reactive Streams components, even those that are based on other implementations, such as RxJava 2, and it brings reactive features such as non-blocking IO and back-pressure to this processing model (for more information see Project Reactor). Whenever the parameter and/or return types are not `Flux`, Spring Cloud Function wraps them so that functions can interoperate via `Flux`. For simple item-at-a-time processing use-cases, you can keep it simple:

```
public class Greeter implements Function<String, String> {
  public String apply(String name) {
    return "Hello " + name;
  }
}
```

But if you need to implement functions that handle a dataset as the unit of processing, via windowing or reduce operations, you can use `Flux` types:

Have we found a golden hammer yet?

Bet that would be helpful during your next retro!

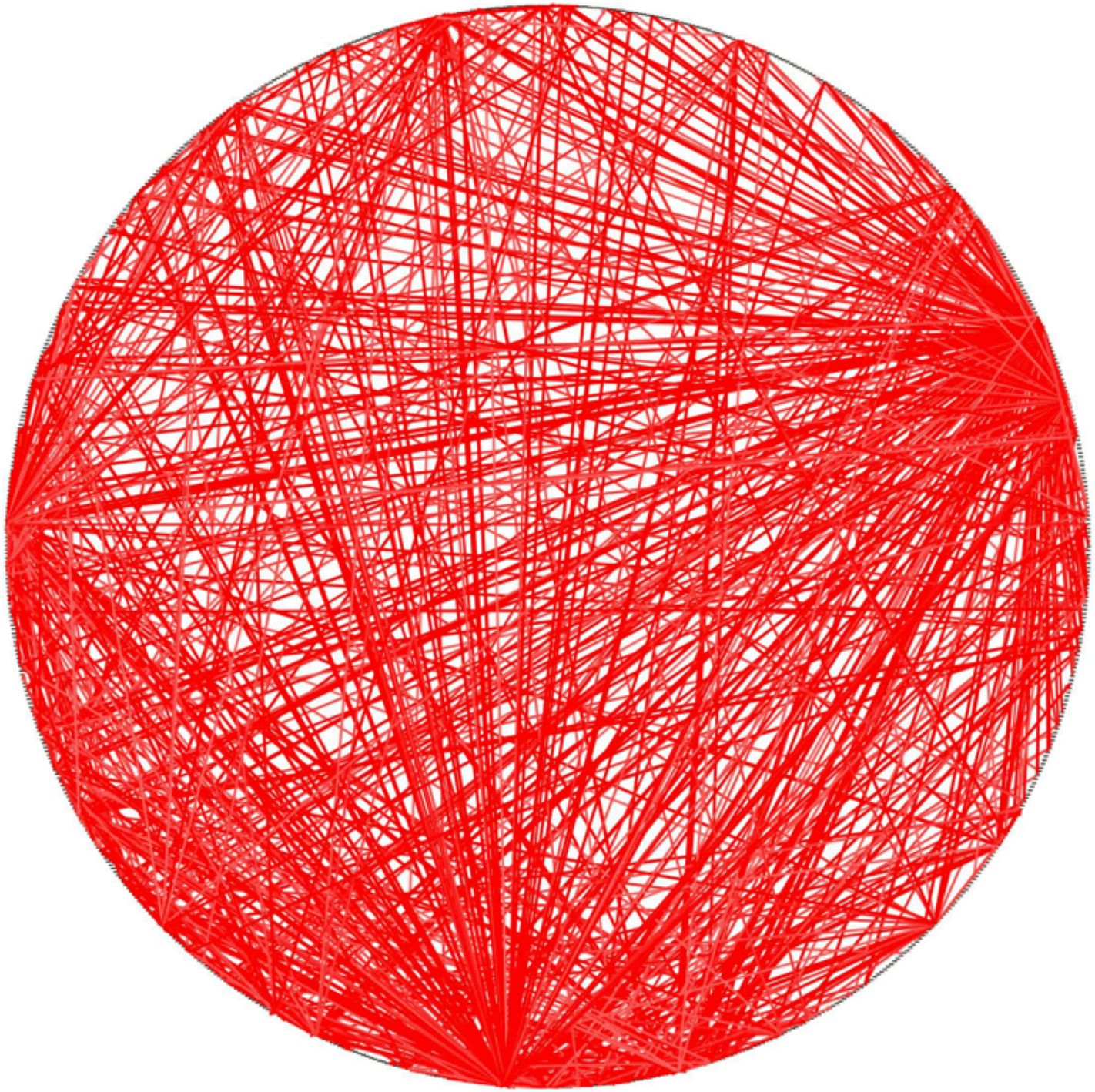
Turns out there are still engineering issues we have to overcome.

It isn’t all puppies and rainbows.

Sorry.

Turns out, those things Yegge mentions…are still things.

What would you say your microservice call pattern looks like?



http://evolutionaryarchitecture.com

The traditional sys admin approach doesn’t give us reliable services.

Inherent tension.

Conflicting incentives.

Developers want to release early, release often.

Always Be Changing.

But sys admins want stability.

It works. No one touch anything.

Thus trench warfare.

Doesn’t have to be this way!

We can all get along.

What if we took a different approach to operations?

“what happens when you ask a software engineer to design an operations team.”

https://landing.google.com/sre/book/chapters/introduction.html

Ultimately, this is just software engineering applied to operations.

Replace manual tasks with automation.

Focus on engineering.

Many SREs are software engineers.

Helps to understand UNIX internals or the networking stack.

Our operational approach has to evolve.

The “Review Board” meeting once a quarter won’t cut it.

How do we move fast safely?

Operations must be able to support a dynamic environment.

That is the core of what we mean by site reliability engineering.

How we create a stable, reliable environment for our services.

It doesn’t happen in spare cycles.

Make sure your SREs have time to do actual engineering work.

On call, tickets, manual tasks - shouldn’t eat up 100% of their day.

SREs need to focus on automating away “toil” aka manual work.

Isn’t this just DevOps?

Can argue it is a natural extension of the concept.

Think of SRE as a specific implementation of DevOps.

SRE Responsibilities.

What should my SRE team focus on?

Availability. Stability. Latency.
Performance. Monitoring.

Capacity planning. Emergency response.

Drive automation.

SREs help us establish our SLOs.

Embrace risk. Manage risk.

Risk is a continuum.

And a business decision.

What do our customers expect?

What do our competitors provide?

Cost.

YOU CAN BANK WITHOUT BORDERS.
Learn More and Contact an HSBC Tech Industry Specialist
HSBC Bank USA, N.A. Member FDIC.
HSBC

AdChoices

Next Story

Amazon Web Services

Popular Posts

Larry Ellison pokes AWS while unveiling intelligent database servic...

Nick D'Aloisio, who sold startup to Yahoo aged 17, has raised funding for a...

Zuckerberg asks forgiveness, but Facebook needs change

AWS fires back at Larry Ellison's claims, saying it's just Larry being Larry

Honest Company may be raising a down round

First impressions of the $49 Google Home

Amazon AWS S3 outage is breaking things for a lot of websites and apps

Posted Feb 28, 2017 by Darrell Etherington (@etherington)

Amazon's S3 web-based storage service is experiencing widespread issues, leading to service that's either partially or fully broken on websites, apps and devices upon which it relies. The AWS offering provides hosting for images for a lot of sites, and also hosts entire websites, and app backends including Nest.

The S3 outage is due to "high error rates with S3 in US-EAST-1," according to Amazon's AWS service health dashboard, which is where the company also says it's working on "remediating the issue," without initially revealing any further details.

SAMSUNG
Galaxy Note8
LOOK WHO'S FOUR!

AdChoices

Crunchbase

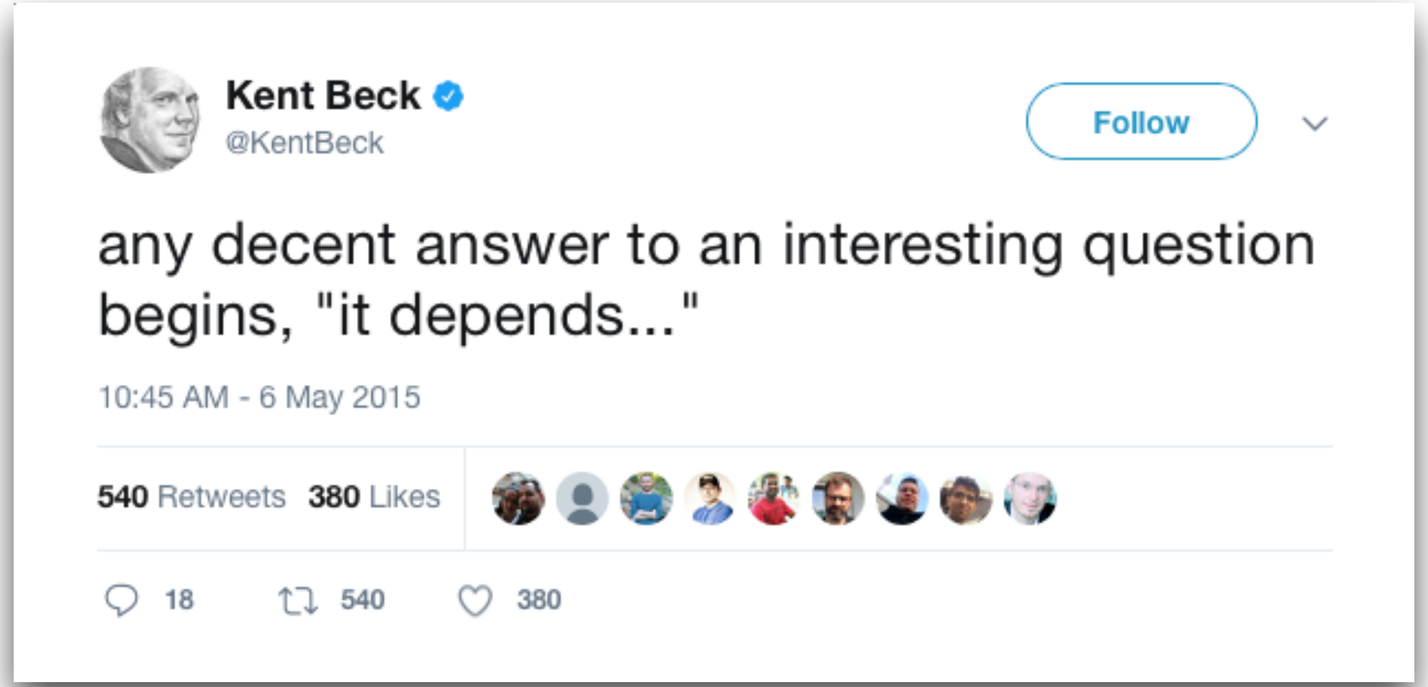
Amazon Web Services

FOUNDED
2006

OVERVIEW
Amazon Web Services provides information technology infrastructure services to businesses in the form of web services. Its products and solutions include cloud computing, compute, networking, storage and content delivery, databases, analytics, application services, deployment and management

UP NEXT

Should those sites/apps have had a redundant backup?



https://twitter.com/KentBeck/status/596007846887628801

How much would that have cost those sites/apps?

How much more revenue would that have driven for them?

It is a tradeoff.

Long term vs. short term thinking.

Heroic efforts can work for the short term.

But that isn’t sustainable.

In the long run it may be better to lower your SLOs for a short time…

To allow you to engineer a better long term solution.

Mean time to recovery.

Having a run book helps immensely.

…thinking through and recording the best practices ahead of time in a "playbook" produces roughly a 3x improvement in MTTR as compared to the strategy of "winging it."

— Benjamin Treynor Sloss
Site Reliability Engineering

We don't rise to the level of our expectations, we fall to the level of our training.

— Archilochus

Design/implement monitoring solutions.

Establish alerting.

Logging best practices.

Create dashboards.

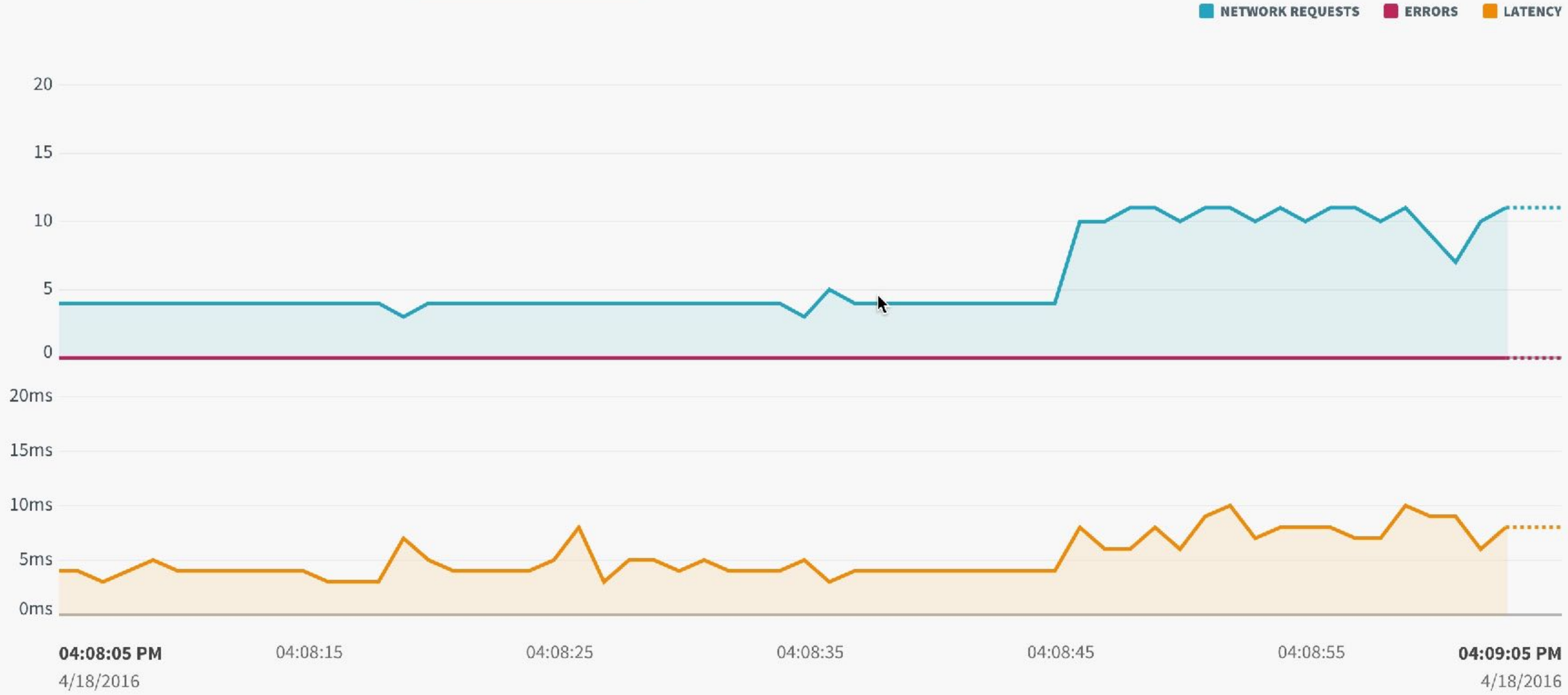
PCF Metrics | fortune-ui

https://metrics.run.pivotal.io/apps/ff713209-ad5d-4b6e-bfe1-0829d7f3ac68/network

fortune-ui

ORG: **platform-eng** SPACE: **nfjs-workshop** STATUS: **Running**

DASHBOARD | CONTAINER METRICS | NETWORK METRICS



metrics.run.pivotal.io

PCF Metrics

Search for Application...

Feedback NSCHUTTA@PIVOTAL.IO

spo-ring-todo ORG: app[0] SPACE: spo-ring-homework STATUS: Running

Oct 4, 9:16 am — Oct 5, 9:16 am (local)

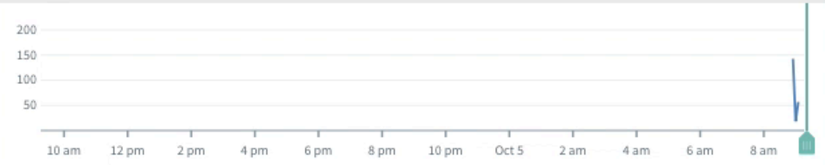
Sep 22 Sep 23 Sep 24 Sep 25 Sep 26 Sep 27 Sep 28 Sep 29 Sep 30 Oct 1 Oct 2 Oct 3 Oct 4 Oct 5

DASHBOARD

view instances

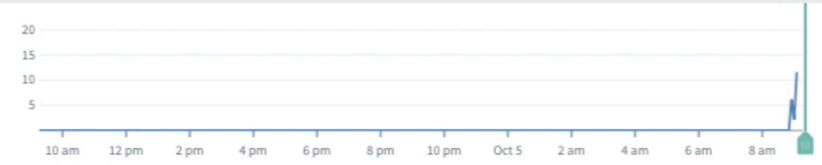
+ ADD CHART

cf.system.latency



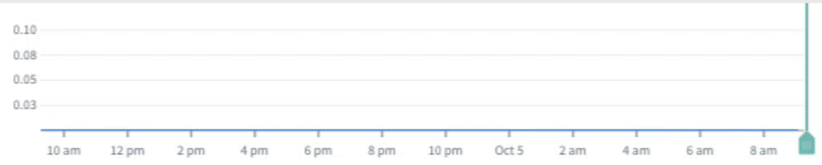
gauge | average across instances | milliseconds

cf.system.request-count



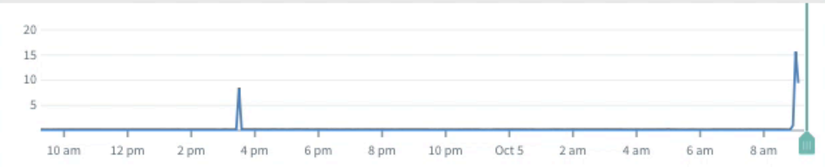
gauge | total across instances | per minute

cf.system.request-error-count



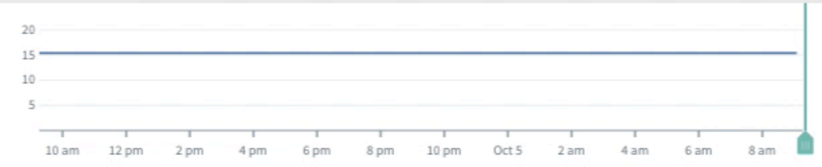
gauge | total across instances | per minute

cf.system.cpu



gauge | average across instances | percentage

cf.system.disk

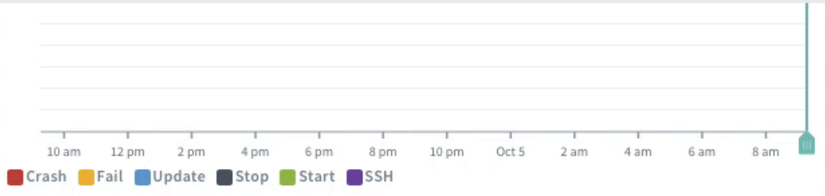


gauge | average across instances | percentage

cf.system.memory

gauge | average across instances | percentage

cf.system.events



LOGS

A Medium Corporation

mattklein123 Follow
Engineer @lyft
Oct 25 · 4 min read

Lyft's Envoy dashboards

Front/edge Envoy per-host row

I've given quite a few talks about observability in the age of the service mesh (most recent slides, unfortunately this talk series has not been recorded yet). Visibility into the inherently unstable network is one of the most important thing that Envoy provides and I'm asked repeatedly for the source of the dashboards that we use at Lyft. In the interest of "shipping" and getting something out there that can help folks, we are releasing a snapshot of our internal Envoy dashboards.

What we are releasing is unfortunately *not* going to be readily consumable. It is also not an OSS project that will be maintained in any way. The goal is to provide a snapshot of what Lyft does internally (what is on each dashboard, what stats do we look at, etc.). Our hope is having that as a reference will be useful in developing new dashboards for your organization.

107 1

Next story
Coworkers Will Become Custo...

Four Golden Signals.

https://landing.google.com/sre/book/chapters/monitoring-distributed-systems.html#xref_monitoring_golden-signals

Latency - how long does it take to service a request.

Traffic - level of demand on the system. Requests/second. I/O rate.

Errors - failed requests. Can be explicit, implicit or policy failure.

Saturation - how much of a constrained resource do we have left.

Important to consider the sampling frequency.

High resolution can be costly.

Aggregate data.

Some measures benefit from shorter intervals…others not so much.

Establish alerting thresholds.

Alert levels can be hard to get right.

Should be actionable.

Should require a human.

Should result in a sense of urgency.

Implies we cannot have more than a few pages a day - people burn out.

You can over alert and over monitor!

Eliminate toil.

Automate all the things.

Manual. Repetitive. Automatable.

One offs. Reactive. Grunt work.

Toil drives people out of SRE work.

Hurts morale. Sets a bad precedent.

People can’t do the same thing twice.

See golf.

We need consistency.

Deployment pipeline
has to be repeatable.

We cannot move fast safely unless SREs are freed from toil.

Postmortems.

We will make mistakes.

Outages will still happen.

Vital we learn from those experiences.

Do not blamestorm.

“Blameless postmortems.”

Goal is to prevent it
from happening again.

Document the incident.
What happened?

What was the root cause(s)?

What can we do to prevent this from happening in the future?

Be constructive, not sarcastic.

Consider a basic template.

Title/ID.

Authors.

Status.

Impact.

Root Causes.

Resolution.

Action Items.

Lessons Learned.

Timeline.

Whatever you think will help!

Lessons Learned

What went well

- Monitoring quickly alerted us to high rate (reaching ~100%) of HTTP 500s
- Rapidly distributed updated Shakespeare corpus to all clusters

What went wrong

- We're out of practice in responding to cascading failure
- We exceeded our availability error budget (by several orders of magnitude) due to the exceptional surge of traffic that essentially all resulted in failures

Where we got lucky[166]

- Mailing list of Shakespeare aficionados had a copy of new sonnet available
- Server logs had stack traces pointing to file descriptor exhaustion as cause for crash
- Query-of-death was resolved by pushing new index containing popular search term

Timeline[167]

2015-10-21 **(all times UTC)**

- 14:51 News reports that a new Shakespearean sonnet has been discovered in a Delorean's glove compartment
- 14:53 Traffic to Shakespeare search increases by 88x after post to **/r/shakespeare** points to Shakespeare search engine as place to find new sonnet (except we don't have the sonnet yet)
- 14:54 OUTAGE BEGINS — Search backends start melting down under load
- 14:55 docbrown receives pager storm, `ManyHttp500s` from all clusters

Can be difficult to create a postmortem culture.

Consider a postmortem of the month.

Book club.

Wheel of Misfortune.

Role play a disaster you faced before.

Ease into it.

Recognize people for their participation.

Senior management needs to encourage the behavior!

Perform retros on your postmortems!

Improve them!

We cannot learn anything without first not knowing something.

— Mark Manson

The Subtle Art of Not Giving a F*ck

All services are equal. Some services are more equal than others.

Defining our SLOs is a critical step towards production hardened services.

Availability is one of our most important objectives.

What is our availability goal?

SLAs vs. SLOs.

Terminology is often misused.

SLI - service level indicator.

A measure of some aspect of our system.

Latency, error rate, throughput…

Availability.

Percentage of time your service is available.

How much downtime can you tolerate?

99%: 7.20 hours a month,
14.4 minutes a day.

99.9%: 8.76 hours PER YEAR.
1.44 minutes a day.

99.99%: 4.38 minutes A MONTH.
8.66 seconds a day.

99.999%: 6.05 seconds A WEEK,
864.3 milliseconds a day.

By the way, the GKE's availability SLA is 99.5%. Just saying.

https://cloud.google.com/kubernetes-engine/sla

SLO - service level objective.

Our target value or a range of values.

Our SLO for availability is 99.9%.

Our average response time should be less than 100 milliseconds.

Can be very tricky to pick an SLO!
More on that in a minute.

We don’t always get to choose our SLOs.

Our users may have a thing or two to say about what they want!

We are also subject to the things we depend on.

If you depend on a service that only guarantees 99.9% availability…

You cannot guarantee 99.99%!

SLA - service level agreement.

Assign a consequence to missing/meeting an SLO.

Often contractual and involve some kind of financial penalty.

If there is no consequence, we're talking about an SLO.

SLAs are the realm of product and legal decisions.

Many people say SLA
when they mean SLO.

Pedantic much?

Ubiquitous language and all that…

SLOs are not a purely technical decision.

Our business and our customers will have an awful lot to say here.

Simple is good.

Infinity is not a goal.

More is not better.

SLOs should help you prioritize.

Give yourself some wiggle room - you can always tighten them up later.

Ensure there is some margin for error.

May have an “internal” SLO that is tighter than your advertised SLO.

Overachieving doesn’t help.

People will start to rely on your overachieving!

Introduce some latency or downtime to stay at your SLO.

Everyone wants 99.999%.

Everyone wants hot/hot.

Until they see the price tag.

If you have to ask…

Establish an error budget.

Establish your target availability.

Say 99.9%.

Your error budget is 0.01%.

Spend it however you want! Just don’t go over that limit.

Goal is not zero outages. Goal is to stay within the error budget.

Go ahead with those experiments.

Try different things.

Once we use up the budget though…

Have to slow our roll.

Aligns incentives.

Helps everyone understand the cost/benefit tradeoff.

Working with SREs.

Production Readiness Reviews.

Not a one time, up front thing.

Services should be reviewed and audited regularly.

Does not have to be high ceremony!

Get the team together - SREs, Devs, etc. Draw up the architecture.

Do we have a shared understanding of what the system does?

Do we have a shared understanding of our requirements?

As we talk through it, we will discover bottlenecks.

The Wombat service has a lower availability level than we need.

We will find interesting failure cases.

“When month end falls on the Harvest Moon.”

Review should result in a new architecture diagram or two.

And probably some new items on the backlog.

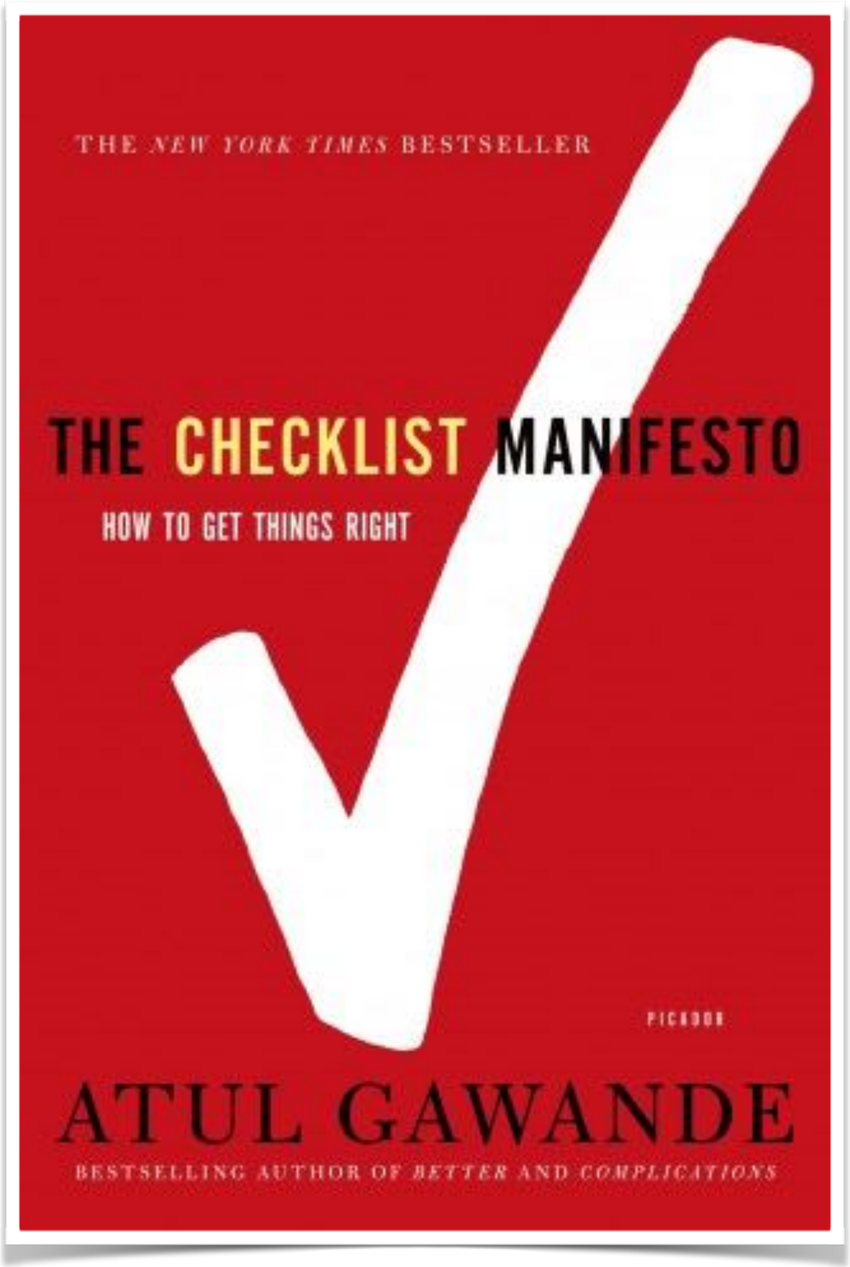
When we refer to an application or microservice as "production-ready," we confer a great deal of trust upon it: we trust it to behave reasonably, we trust it to perform reliably, we trust it to get the job done…

— Susan J. Fowler
Production-Ready Microservices

How do we know we can trust a service?

Consider having a checklist.

A checklist? Seriously?



http://atulgawande.com/book/the-checklist-manifesto/

You know who uses checklists?

Pilots. Surgeons.

Should be quantifiable and measurable.

“Fast” won’t cut it.

Stability.

Reliability.

Scalability.

Fault tolerance.

Performance.

Monitoring.

Documentation.

I know what some of you are thinking…

I don’t have time for all this.

We need to MOVE FAST.
And break things…

We're Agile. With a capital A.

How is your velocity when an outage brings your business to a halt?

Requires buy in from the grass roots level as well as management.

Perform an audit.

Go back to your checklist. Does the service meet our requirements?

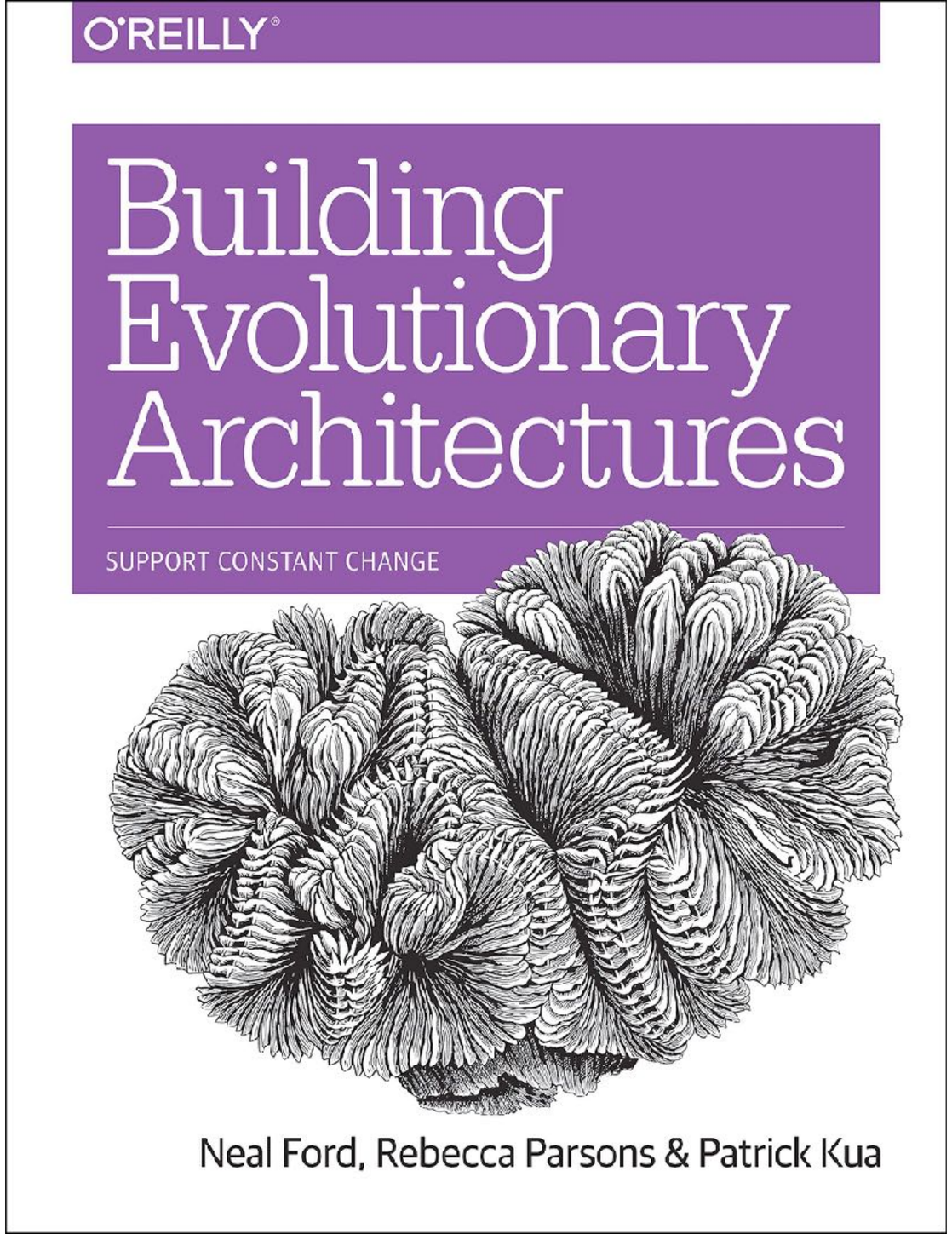
Probably results in some new things in our backlog!

Now we can create a production readiness roadmap.

What do we need to fix and when can/should we fix it.

Drive prioritization of the work.

A lot of this is manual. But some of it can be automated!



http://evolutionaryarchitecture.com

Fitness functions!

Basically, a set of tests we execute to validate our architecture.

How close does this particular design get us to our objectives?

Ideally, all automated. But we may need some manual verifications.

For example…

All service calls must respond within 100 ms.

Cyclomatic complexity shall not exceed X.

Hard failure of an application will spin up a new instance.

Reviews and audits should not be additional red tape.

Should not be overly bureaucratic.

Couple of hours…depending on the complexity of the system.

Architectural reviews.

Look for failure points.

Draw up the architecture.

What happens if *this* fails?

It can’t fail? Yeah it can - what happens if it does?

Think through how
our service could fail.

It is hard. We are really good at thinking through the happy path.

But we need to think about the road less traveled.

Test for these failure scenarios. Does our service respond appropriately?

Only one way to really know…

NETFLIX

SIMIAN ARMY

https://github.com/Netflix/SimianArmy

Chaos engineering.

http://principlesofchaos.org

Next steps.

Do you have an SRE team?

Do you wish you had an SRE team?

It can be built!

You probably have some engineers that can (and would!) do it.

Our applications are changing radically.

0
11
Dev
VOLUME

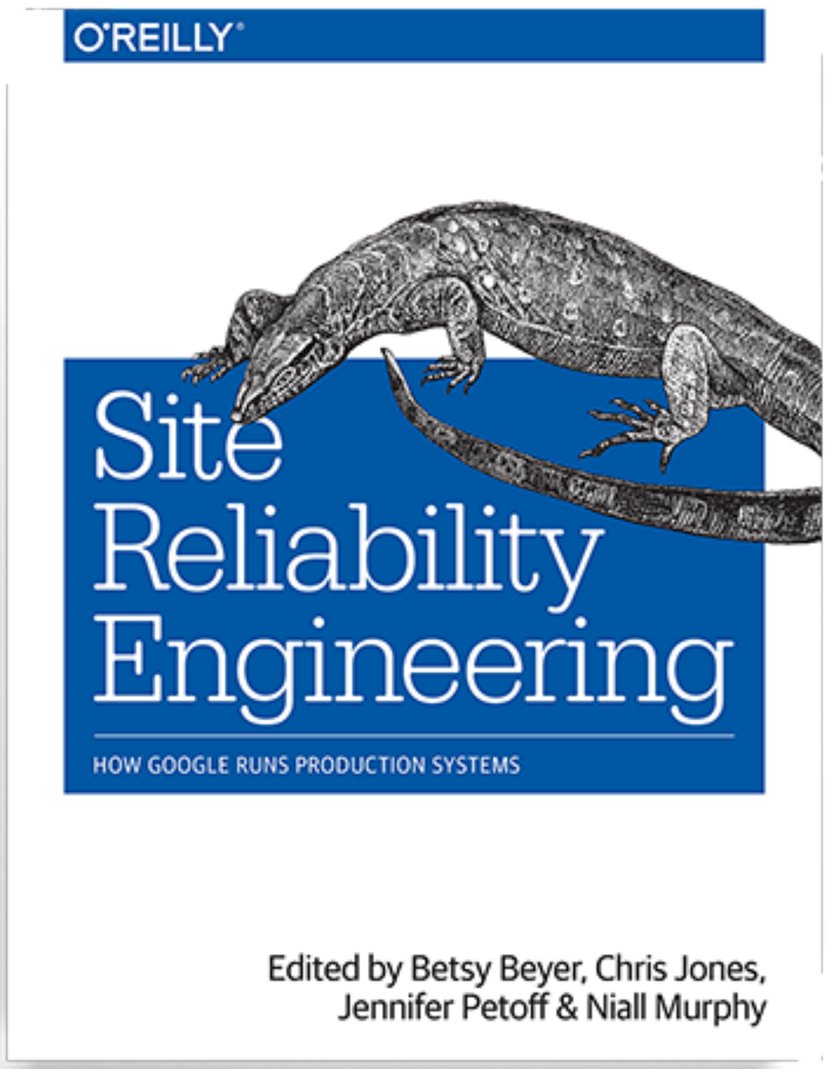
0
11
Ops
VOLUME

We all need to evolve to succeed.

We can’t move fast safely unless the environment enables it.

We have to work together.

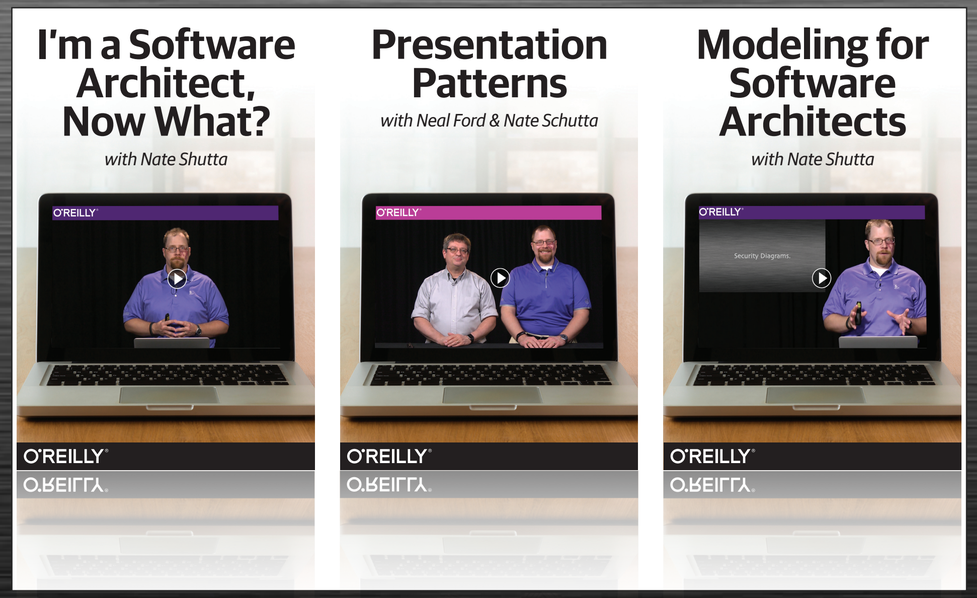
Want more?



https://landing.google.com/sre/book.html

Questions?

Thanks!



Nathaniel T. Schutta
@ntschutta