AWS re:Invent

STP208

How to build a company founded on engineering principles

Brian Scanlan

Principal Systems Engineer

Intercom

Agenda

Why principles are useful

Intercom’s engineering principles

Intercom’s engineering principles in practice

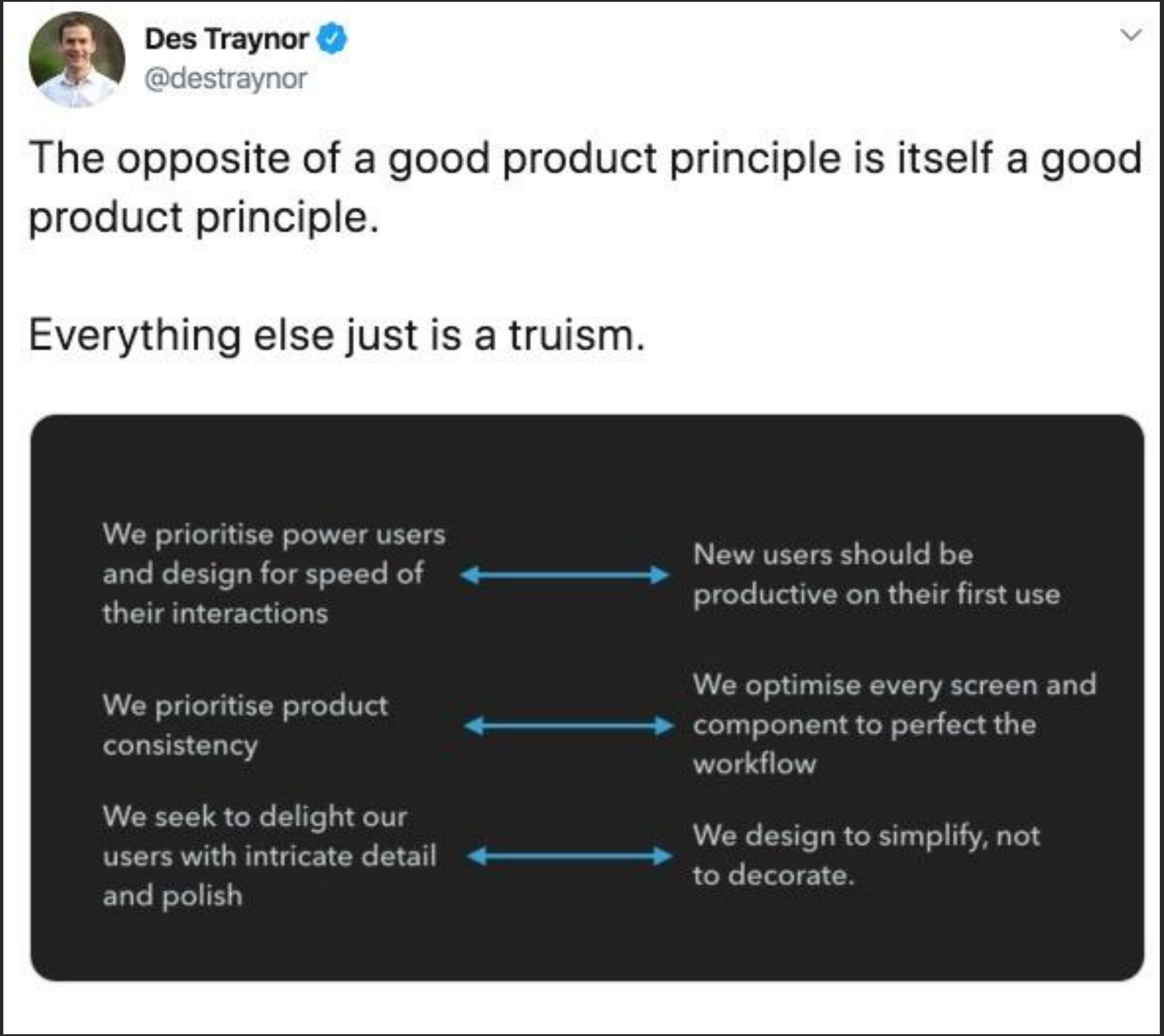
Why principles are useful

A good set of principles allows an organization to work off a common mental model

A good set of principles allow an organisation to work off a common mental model

Bad principles?

“We don’t ship bugs”



Des Traynor
@destraynor

The opposite of a good product principle is itself a good product principle.

Everything else just is a truism.

We prioritise power users and design for speed of their interactions ⟷	New users should be productive on their first use
We prioritise product consistency ⟷	We optimise every screen and component to perfect the workflow
We seek to delight our users with intricate detail and polish ⟷	We design to simplify, not to decorate.

Good principles!

“We build on AWS”

Look at what works in practice.

Look at what you truly believe in.

Refresh them regularly

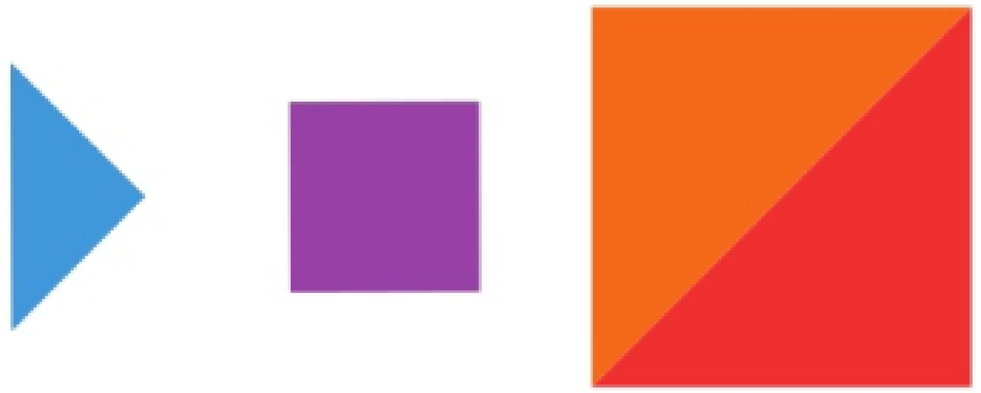
Avoid truisms and consider the opposite opinion.

Intercom's engineering principles

INTERCOM

Opinionated by default, flexible under the hood
Design principle number two
We optimize our designs to feel simple and opinionated by default, but progressively reveal power and flexibility.
Start with the problem
R&D Principle number one
Start by deeply understanding the problem we're solving. Continually evolve this understanding, and persistently return to it to ensure you haven't veered off course.
Ship to learn
The sooner you ship, the sooner you get
on your assumptions and your solution,
can learn quickly if you're having impact
is the beginning more than the end.
Engineering principle number three
Large changes are hard to under
harder to debug. We deliver com
changes in a series of small, contr
to understand steps.
at you ship
what matters
Our delivere
pride and
R&D Principle number two
Shape the solution
d with
sitivity
pride
perso
odular systems over bespoke
re-use, evolve and merge
hing new.

Ship to learn



R&D Principle number three

The sooner you ship, the sooner you get feedback on your assumptions and your solution, so you can learn quickly if you're having impact. Shipping is the beginning more than the end.

Build in small steps

Engineering principle number three

Large changes are hard to understand, and harder to debug. We deliver complex changes in a series of small, controlled, easy to understand steps.

Be technically conservative

Engineering principle number two

We like familiar solutions with boring technologies. We reuse the same patterns in different solutions as much as possible.

Keep it simple

Engineering principle number four

[redacted]

Complexity is the enemy of our ability to move quickly. We will trade off performance, financial cost, and perfect abstraction in order to keep a solution simple.

Build with positivity and pride

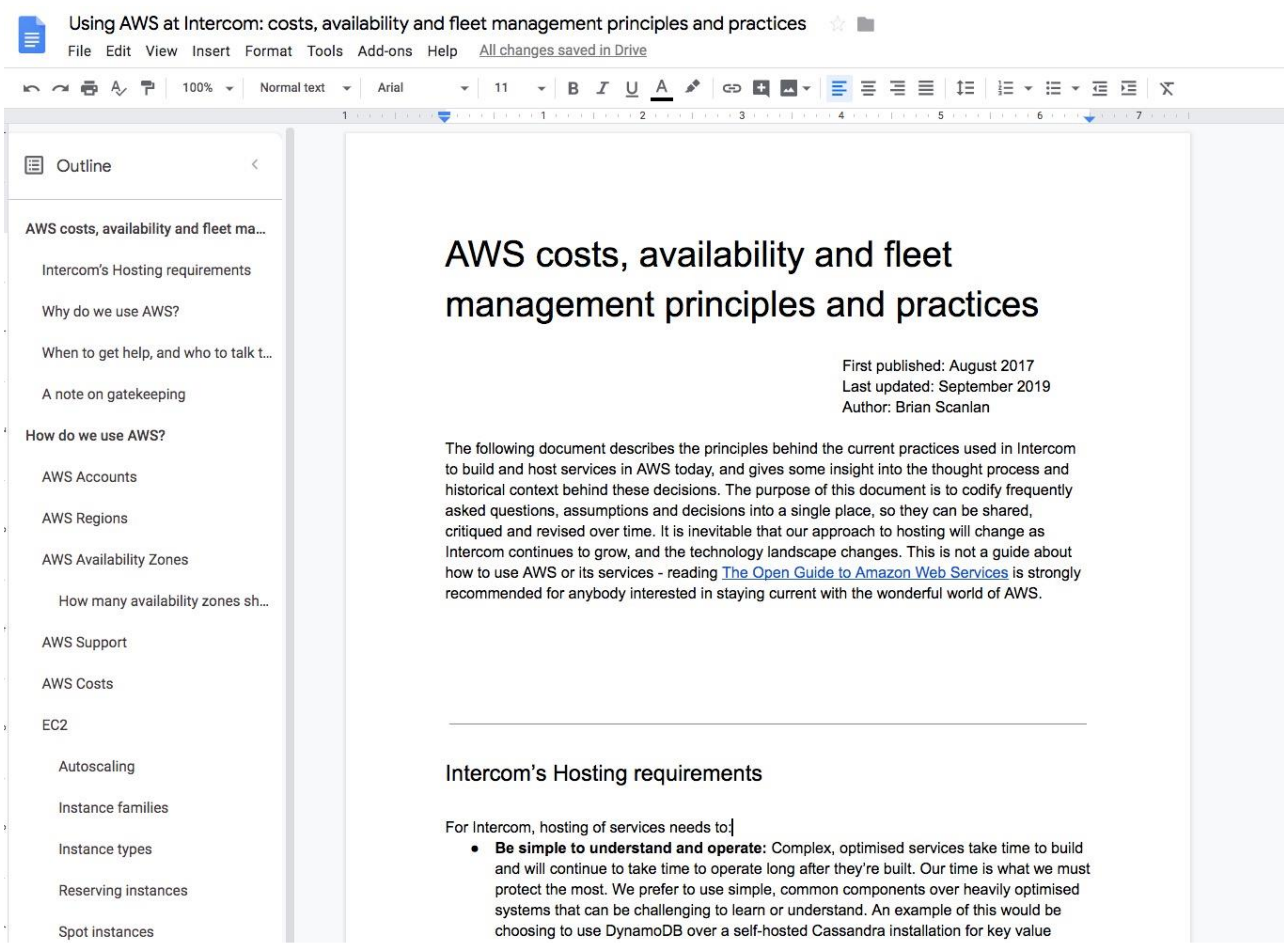
Engineering principle number five

Great things are only built by high functioning groups of humans. We are optimistic, positive, and assume good intent. We are eager to teach and learn.

Intercom’s engineering principles in practice

Example #1

AWS costs


Using AWS at Intercom: costs, availability and fleet management principles and practices
File
Edit
View
Insert
Format
Tools
Add-ons
Help
All changes saved in Drive
100%
Normal text
Arial
11

Outline

AWS costs, availability and fleet management principles and practices

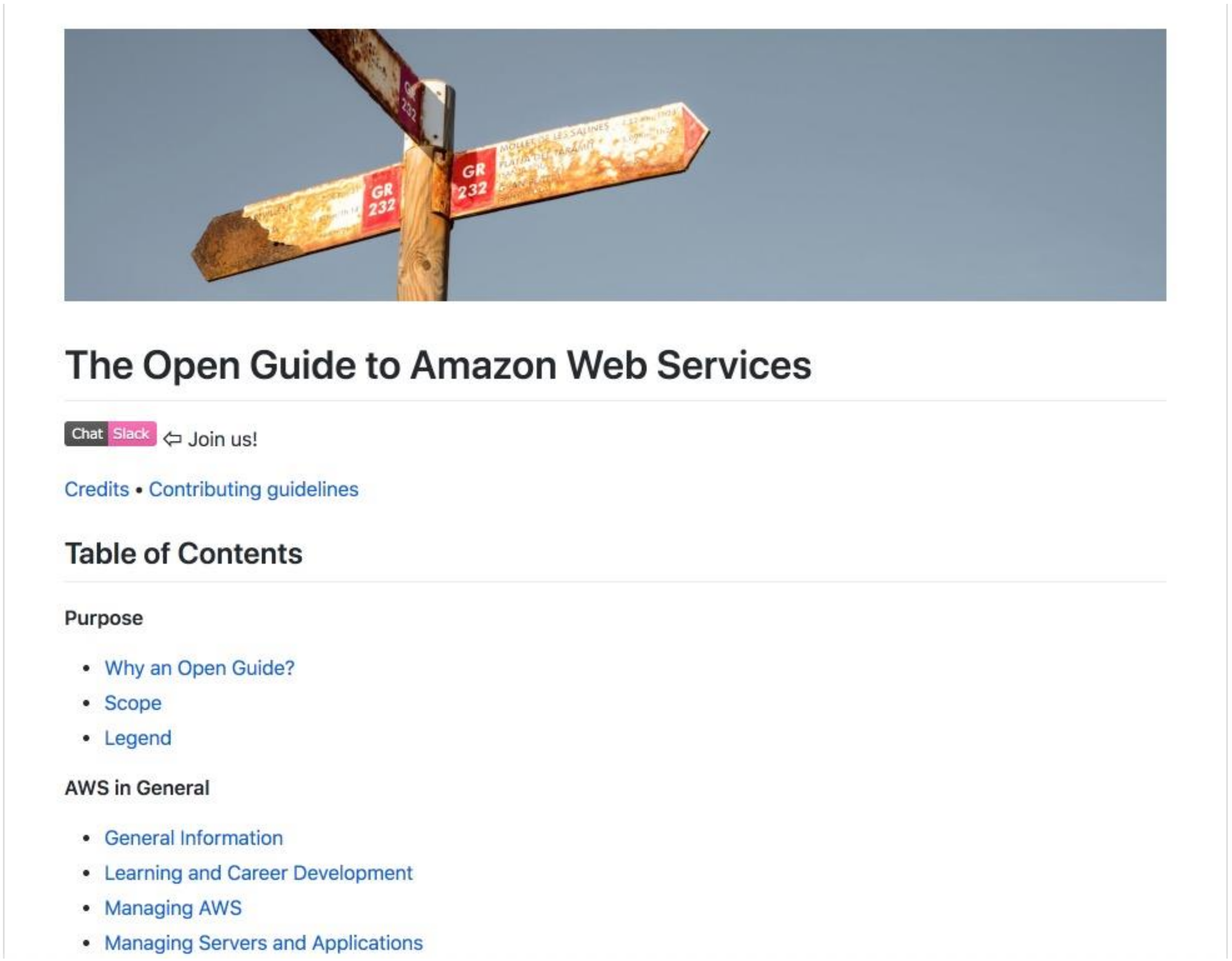
First published: August 2017
Last updated: September 2019
Author: Brian Scanlan

The following document describes the principles behind the current practices used in Intercom to build and host services in AWS today, and gives some insight into the thought process and historical context behind these decisions. The purpose of this document is to codify frequently asked questions, assumptions and decisions into a single place, so they can be shared, critiqued and revised over time. It is inevitable that our approach to hosting will change as Intercom continues to grow, and the technology landscape changes. This is not a guide about how to use AWS or its services - reading The Open Guide to Amazon Web Services is strongly recommended for anybody interested in staying current with the wonderful world of AWS.

Intercom's Hosting requirements

For Intercom, hosting of services needs to:

- **Be simple to understand and operate:** Complex, optimised services take time to build and will continue to take time to operate long after they're built. Our time is what we must protect the most. We prefer to use simple, common components over heavily optimised systems that can be challenging to learn or understand. An example of this would be choosing to use DynamoDB over a self-hosted Cassandra installation for key value



The Open Guide to Amazon Web Services

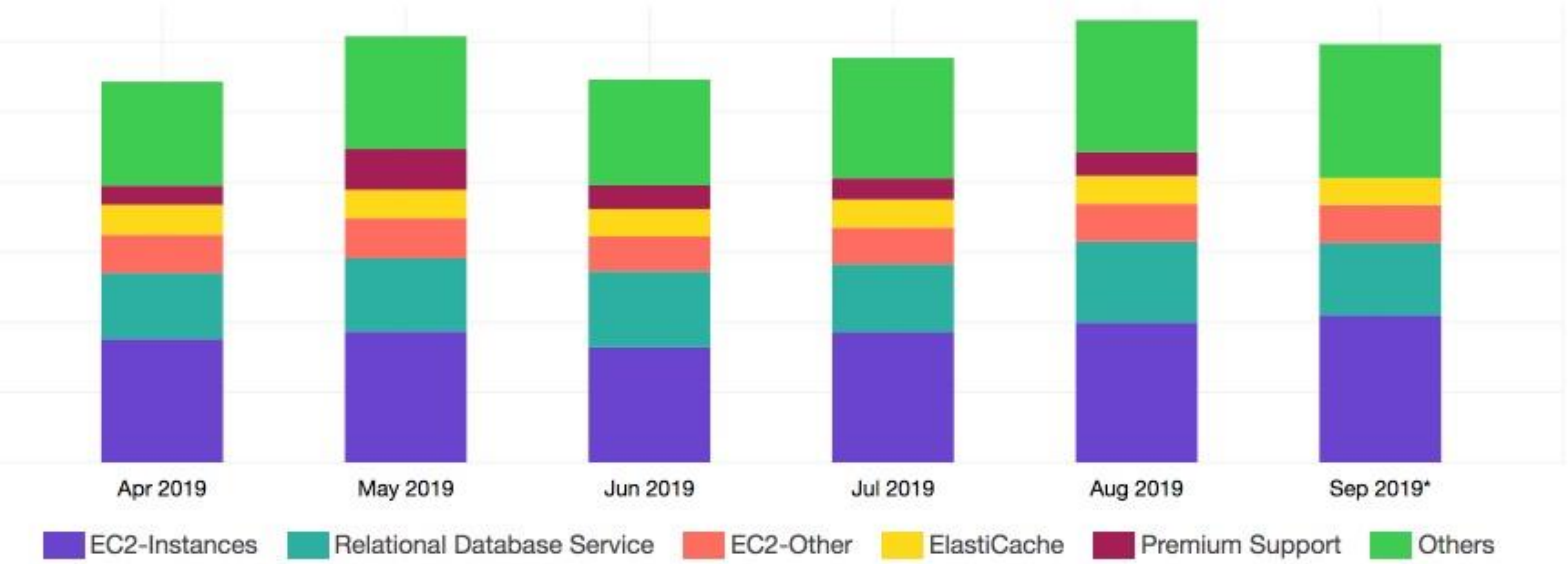
Chat Slack ⇦ Join us!

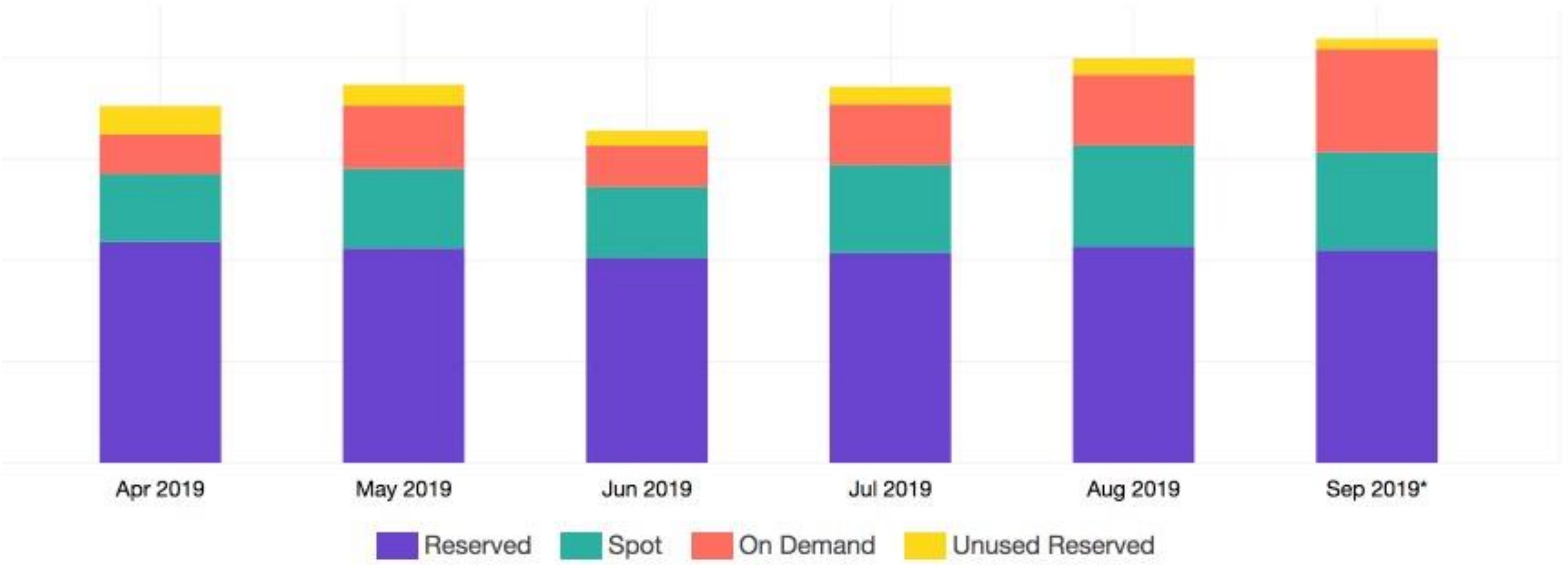
Credits • Contributing guidelines

Table of Contents

“Our approach to managing AWS costs is **REACTIVE** and prioritizes taking action against the highest contributors to our costs **as observed in production**”

“The complexity of implementing multi-cloud makes this a decision we don’t even want to contemplate”


Apr 2019
May 2019
Jun 2019
Jul 2019
Aug 2019
Sep 2019*
EC2-Instances
Relational Database Service
EC2-Other
ElastiCache
Premium Support
Others


Apr 2019
May 2019
Jun 2019
Jul 2019
Aug 2019
Sep 2019*
Reserved
Spot
On Demand
Unused Reserved

Things we do:

Tag resources

Use Cost Explorer to visualize trends

Things we do:

Work with product team to understand usage

Use a small number of modern instance families

Things we do:

Use auto scaling support for multiple instance types and purchase options

Costs are important, but…

We ship to learn.

We’re technically conservative.

Example #2

Monolith

“Rebuilding your monolith from scratch using Go microservices”

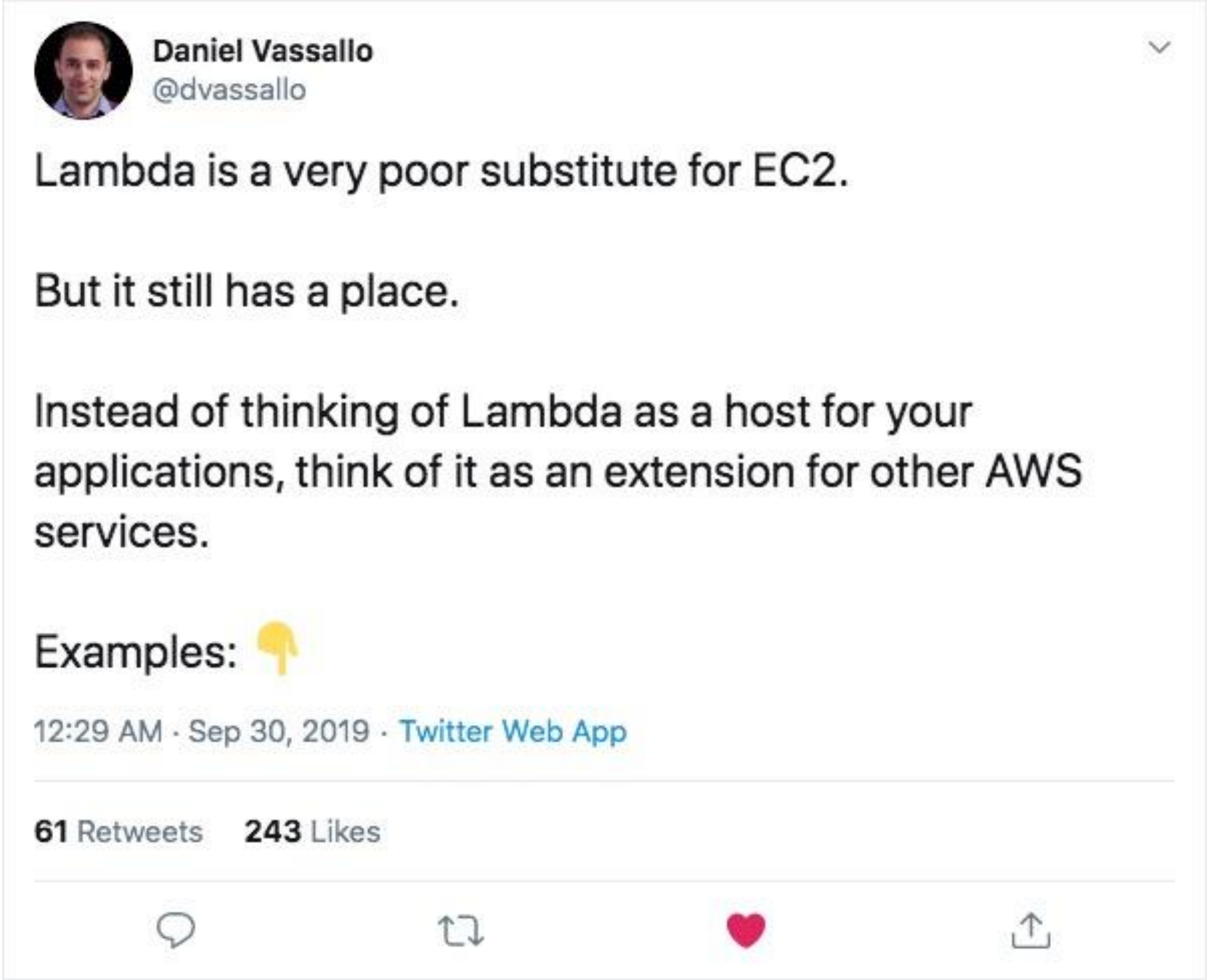
“Our monolith was poorly tested and deployed once every 6 months, and boy did you not want to be in the office that week (or month)”

“Our monolith kept slowing us down, so we had to break it apart!”

Majestic monolith

Majestic monolith running on EC2 instances

Supercharging our monolith with serverless



Daniel Vassallo
@dvassallo

Lambda is a very poor substitute for EC2.

But it still has a place.

Instead of thinking of Lambda as a host for your applications, think of it as an extension for other AWS services.

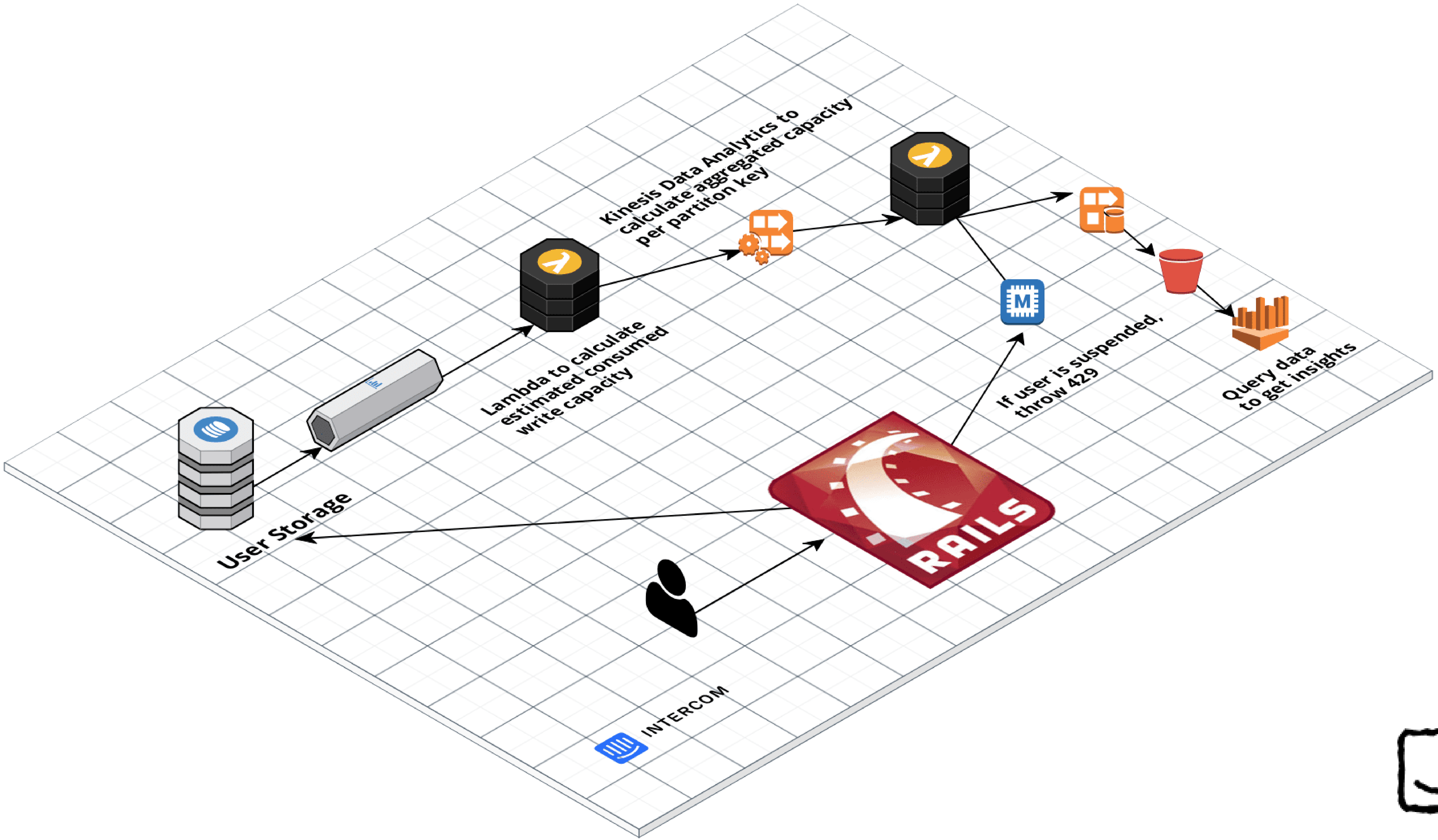
Examples: 👇

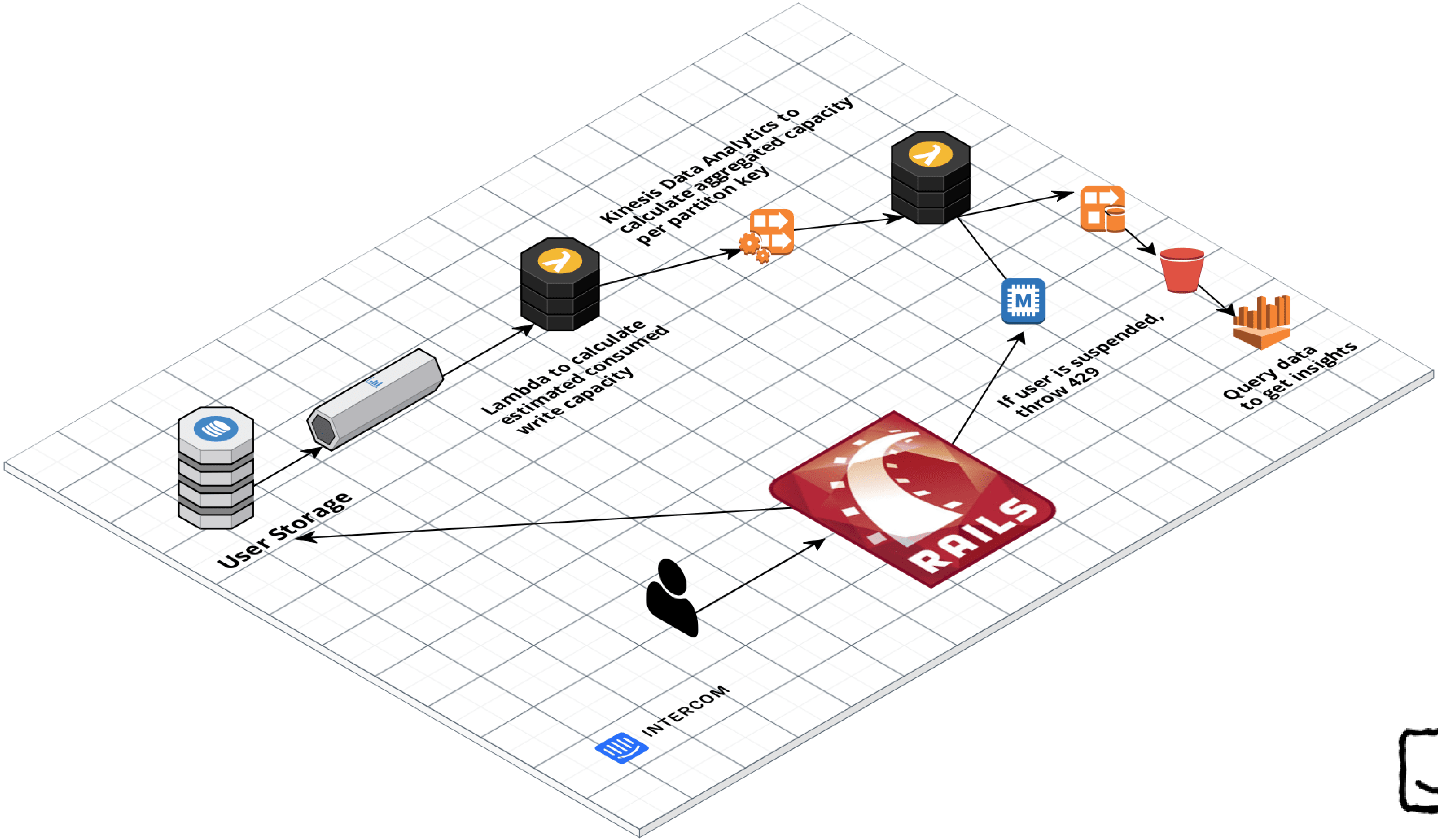
12:29 AM · Sep 30, 2019 · Twitter Web App

61 Retweets 243 Likes

Example #3

Replacing MongoDB with Amazon DynamoDB


Kinesis Data Analytics to calculate aggregated capacity per partiton key
Lambda to calculate estimated consumed write capacity
If user is suspended, throw 429
Query data to get insights
User Storage
RAILS
INTERCOM


Kinesis Data Analytics to calculate aggregated capacity per partiton key
Lambda to calculate estimated consumed write capacity
If user is suspended, throw 429
Query data to get insights
User Storage
RAILS
INTERCOM

Thank you!

Brian Scanlan

@brian_scanlan

Please complete the session survey in the mobile app.