The serverless revolution

Obligatory Introduction

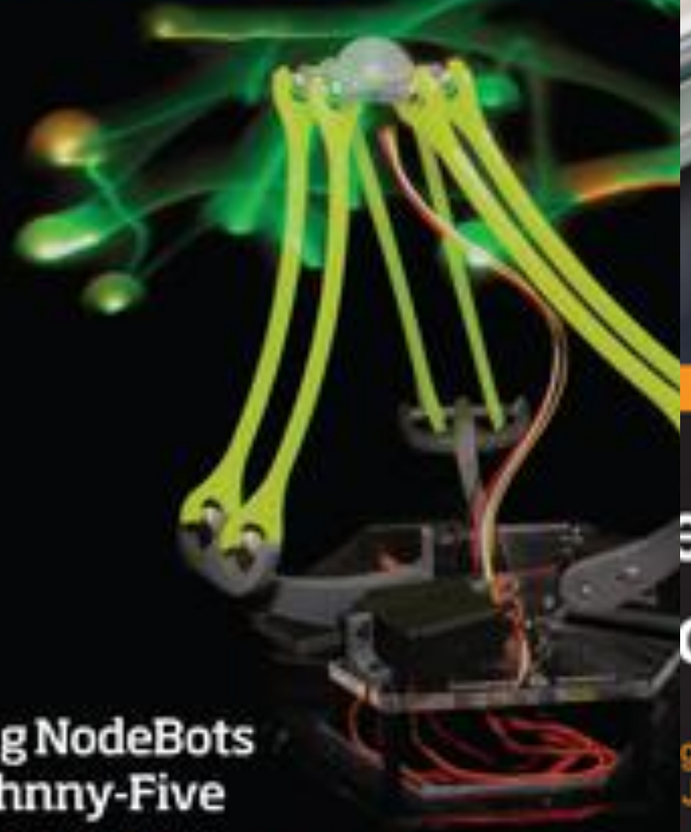
→ Mx. Kas Perch (They/Their/Them)

→ EE Student at Arizona State University

→ Looking for DevRel work!

Robots/ Nodebots Author

Make: JavaScript Robotics



Building NodeBots with Johnny-Five

Rick Waldron & Backstop Media

With Donovan Buck, Bryan Hughes, Pawel Szymczykowski, Raquel Velez, Kassandra Perch, Susan Hinton, Julian David Duque, Andrew Fisher, David Resseguie, Jonathan Beri, Emily Rose, Anna Gerber, Sara Gorecki & Lyza Danger Gardner

Community Experience Distille

earning JavaScript obotics

gn, build, and program your own remarkable robots JavaScript and open source hardware

ssandra Perch

[PACKT] open so

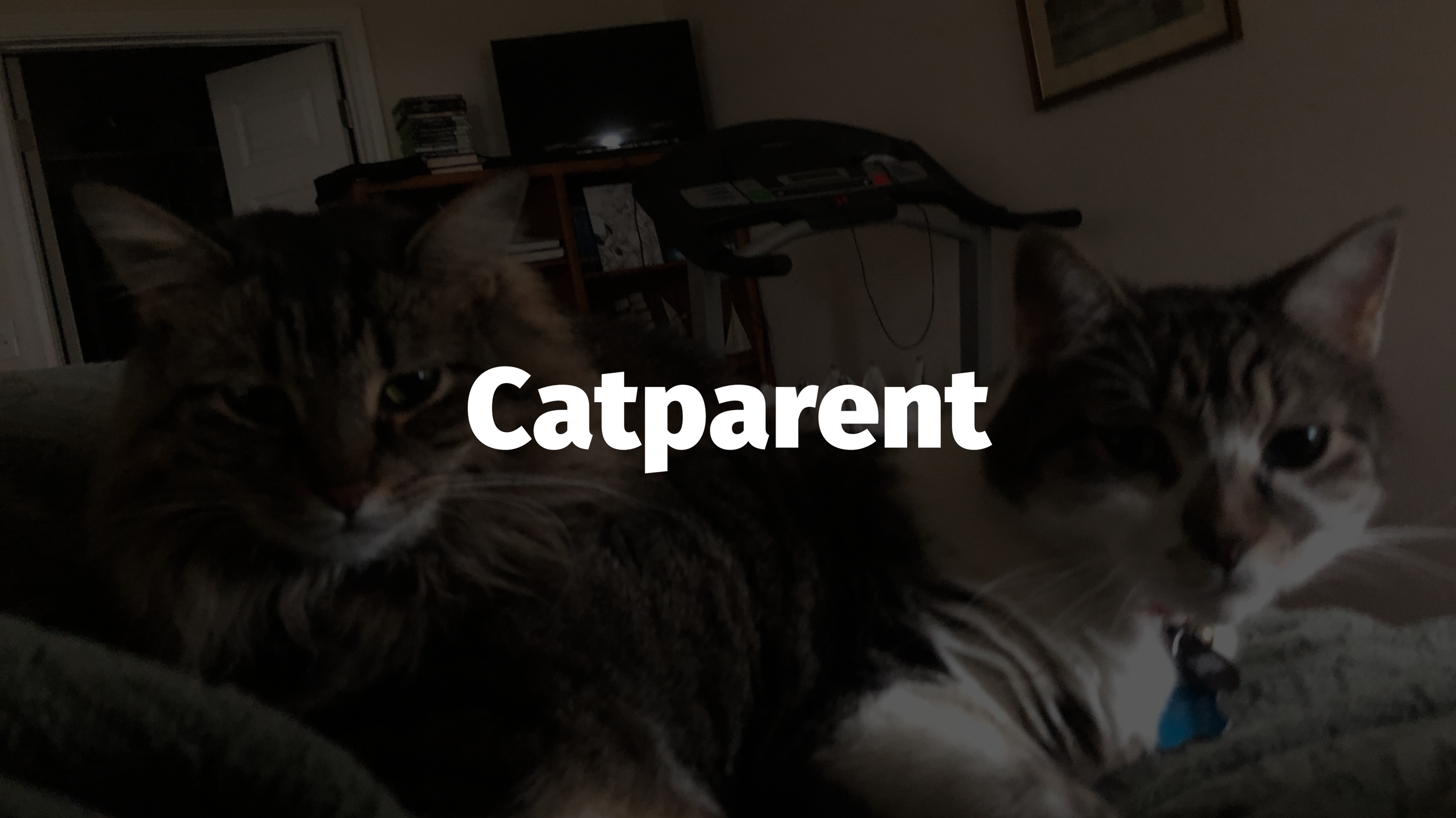
ands-On Robotics with avaScript

robotic projects using Johnny-Five and control hardware JavaScript and Raspberry Pi

ssandra Perch

Packt

www.packt.co


Catparent

Right, right, serverless

"We just spent 10 years convincing people that JS belongs on servers, and now they want to take servers away!"

→ Some Rando


WELP

Okay, But No

Serverless doesn't mean the end of servers, just like cloud computing didn't mean the end of DevOps (and Serverless doesn't mean the end of ops either, chill out)

Also called Functions As A Service (FaaS), I like that name a little better.

...but it's still not a great name.

What should we call it then?

I have **no idea**, so I'll just use Serverless/FaaS until someone comes up with something better, or forever, whichever.

What Serverless Is (Optimistic)

The latest abstraction-- another layer added on in order to allow developers to deploy code with even less need to control infrastructure

What Serverless Is (Pessimistic)

The latest abstraction -- so, wait, we now not only lose control over how our code is run, but we now rely on a third-party to determine our language runtime and environment. But it's easier for devs, so yay?

What Serverless Is (Real Talk)

The latest abstraction -- we gain usability for developers, we create some visibility issues, overall it's the way we've been moving since ever.

Because at the end of the day, developers love abstractions (don't @ me on this one).

What the abstraction is

→ With EC2, we abstracted out servers

→ With Docker, we abstracted out entire environments

→ With Serverless, we're abstracting out the runtime environment

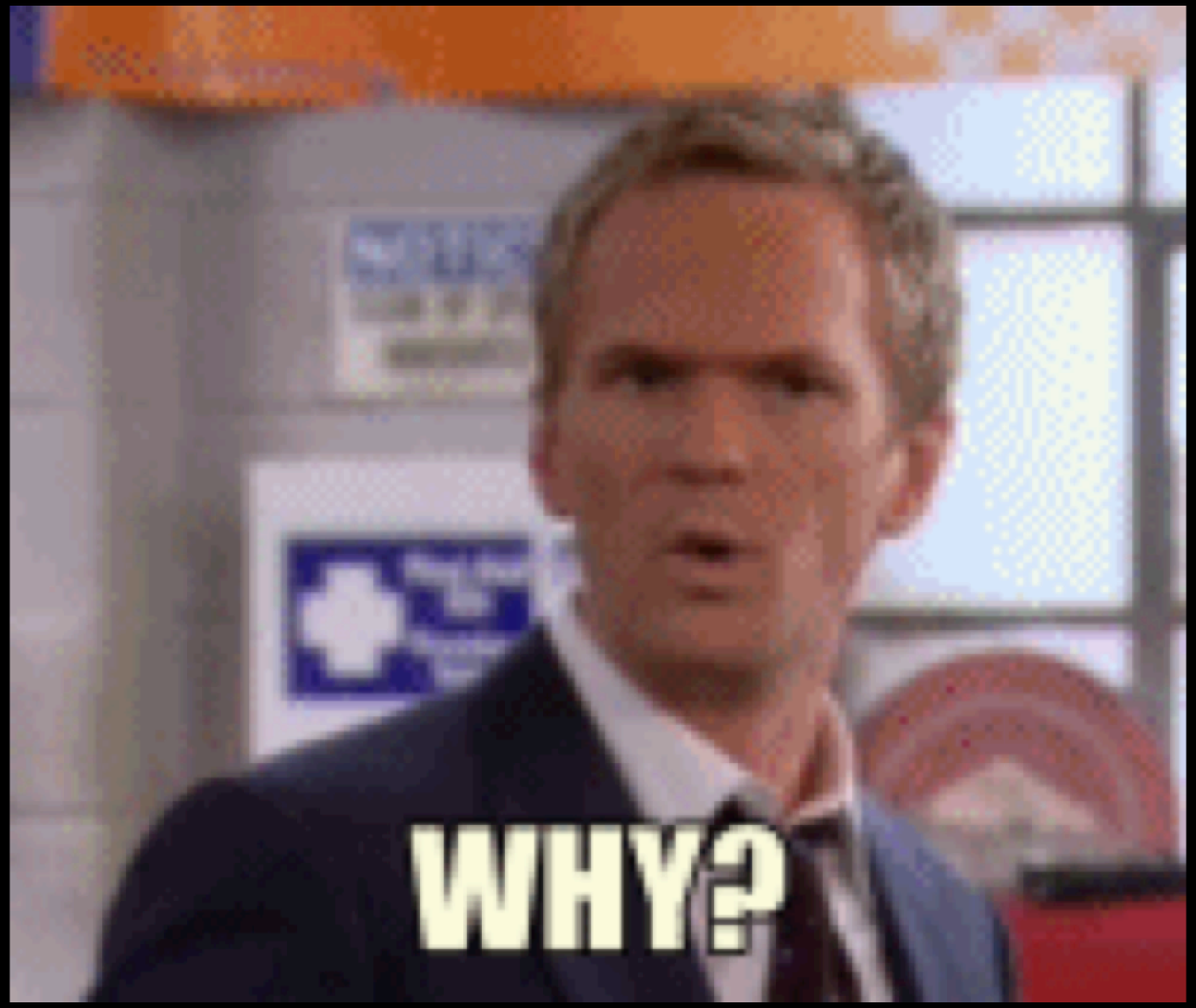
Huh?

What?

→ With EC2, you don't have to manage a physical server

→ With Docker, you automate your environment

→ With Serverless, you don't even have to manage your runtime environment


WHY?

Why is this a good thing?

→ Billed for the time your function runs, and that's all

→ Your write, deploy, and run code (but like for realsies)

→ Scales pretty darn well in spiky situations

Because we have more important problems to be solving


io.js

Demo Time

http://nodebotani.st/set?color=[css color]


What
Serverless is
NOT

Serverless vill not solve all of your problems

→ Serverless vill solve some problems

→ It will make other problems worse

Thanks, Kas, real helpful.

So what problems does serverless solve?

Serverless solves the problem of 'I need to run this code, but it only has traffic spikes at intervals of [a time period greater than 30 minutes] and it only takes [less than five minutes], and I don't mind a few seconds of latency every now and then.

Wait. A few seconds every now and then?

Cold starts!

Cold starts are when your serverless provider needs to spin up your runtime environment so it can run the function *before* running the function, leading to additional latency *for that first invocation.*

How long is a cold start?

It depends on your provider and language -- interpreted languages take a little less time to cold start than say, a Java .jar file, and your provider's method of constructing your runtime matters.

Why do cold starts exist?

This is why you only pay for the time your function is running-- the provider doesn't run a server ready for your code 24/7, it spins up runtimes as needed.

How do I avoid cold starts?

If your **main** question is this, maybe you should think about why you want to use serverless in the first place. (Don't @ me unless you have genuine questions about architecture)

Summary of when Serverless should work for you

→ If your code needs to run periodically, and you don't need to run a server 24/7 to host this code

→ If your functions don't take minutes to run (or need a long-running persistent connection)

→ If your traffic is more sporadic than constant

Servers.lol

(I have stickers!)

But what's the catch?

→ Observability and Metrics are **usually** a nightmare

→ Developer Tooling from **most** providers is kind of a nightmare

Why are Observability and Metrics so difficult?

→ The farther you abstract out, the harder it is to see in.

→ Your serverless provider (in most cases) determines what you can (or cannot) see.

What about developer tooling?

→ Some third-party tools are cropping up

→ Some providers are stepping up to the plate (props, Azure functions)

AHHH Did I miss serverless? How do I start?

→ serverless.com (serverless framework, handles most major providers, great tutorials)

Thank You!

→ Mx. Kas Perch (They/Them/Their)

→ Need a Dev Advocate? I have cards!

→ Want a sticker? I have servers.lol stickers