Applying the Serverless Mindset to Any Tech Stack

Serverless is a State of Mind

deliver:agile, May 1, 2019

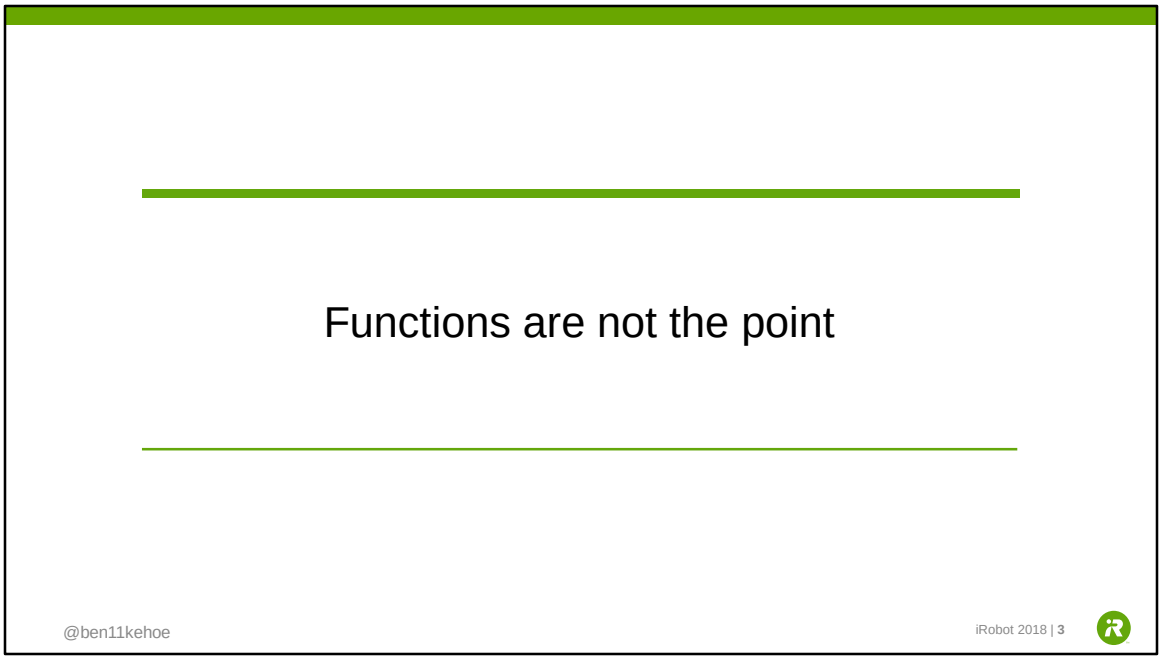
Ben Kehoe
Cloud Robotics Research Scientist
AWS Serverless Hero
@ben11kehoe

Serverless is a buzzword, perhaps more meaningless now even than “cloud”. What I want to convince you today is that at heart, the notion of serverless is not about technology, but about a mindset for choosing technology. And I want to convince you that you can take the serverless mindset and apply it to any situation, and that doing so will help you deliver more value to your customers. There’s a lot in common with agile here: maximizing the amount of work not done, a focus on delivering value, product folks and developers working together, because developers are not deep into technology far from customer value. But making this shift requires recontextualizing the role of engineers in the product delivery process. And when I say engineers, I mean anyone who is responsible for technology development and delivery—developers, operations folks, etc.

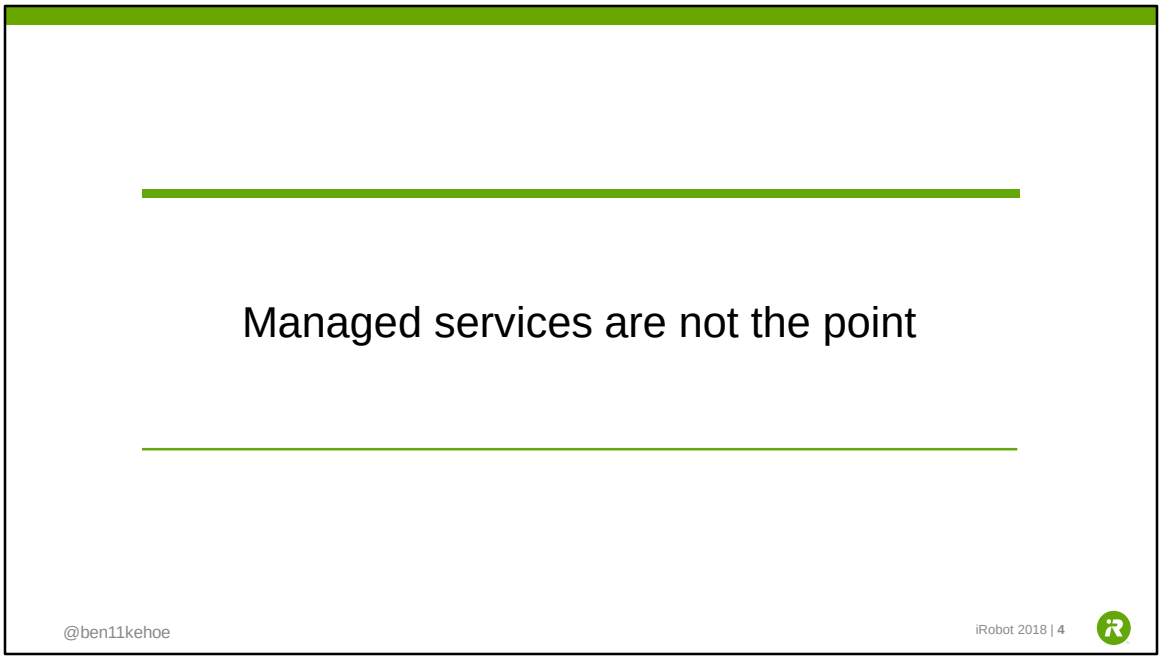
@ben11kehoe

iRobot 2018 | 2

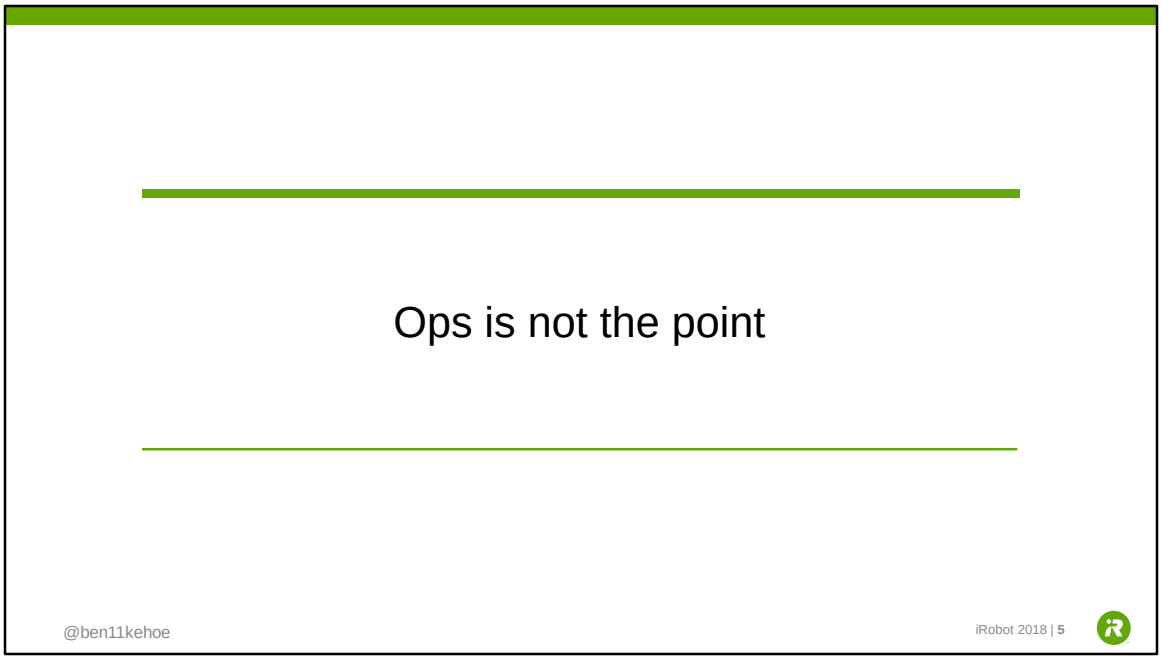
At iRobot, we're fully serverless in production. We sell millions of robots a year, and the entire Roomba line is now connected. So we have a high scale problem—and we don't have any VMs or containers in production. It's all AWS Lambda and about 30 other AWS services. This has been incredibly powerful; the amount of time we spend on operations is minimal, and doesn't really depend on traffic volume. All our business logic is in Functions as a Service.



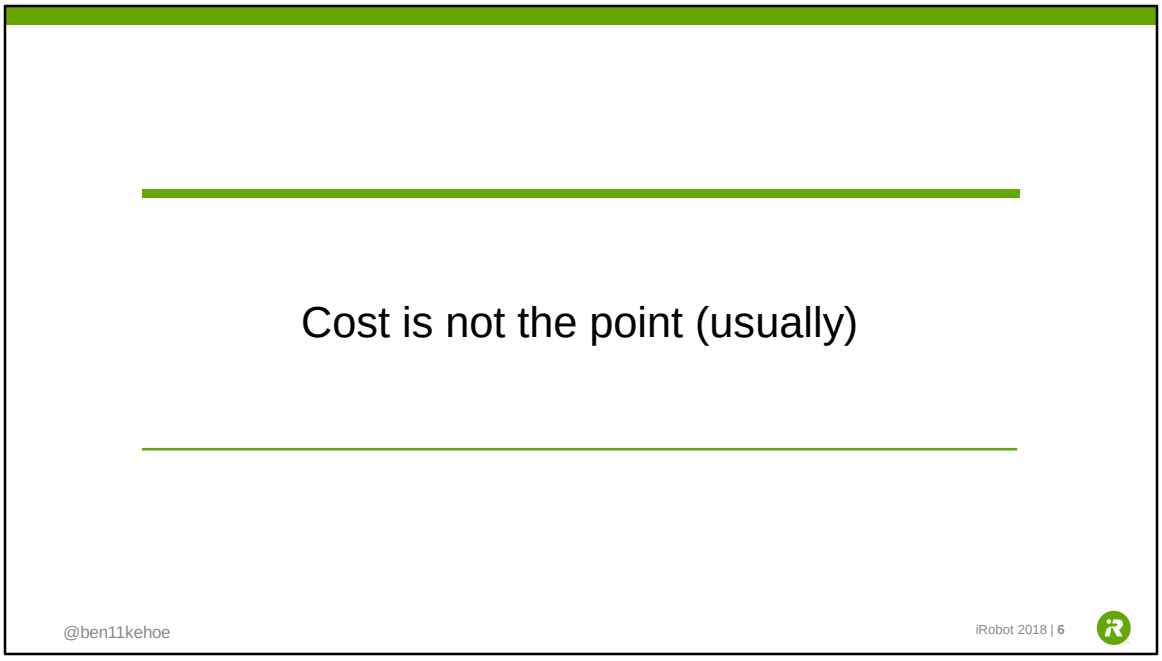
But functions are not the point of serverless. Don't get me wrong, functions are great. They let you scale transparently, you don't have to manage the runtime, they fit naturally with event-driven architectures. These are all fantastic, useful properties. But functions should end up being a small part of your overall solution. You should use functions as the glue, containing your business logic, between managed services that are providing the heavy lifting that forms the majority of your application.



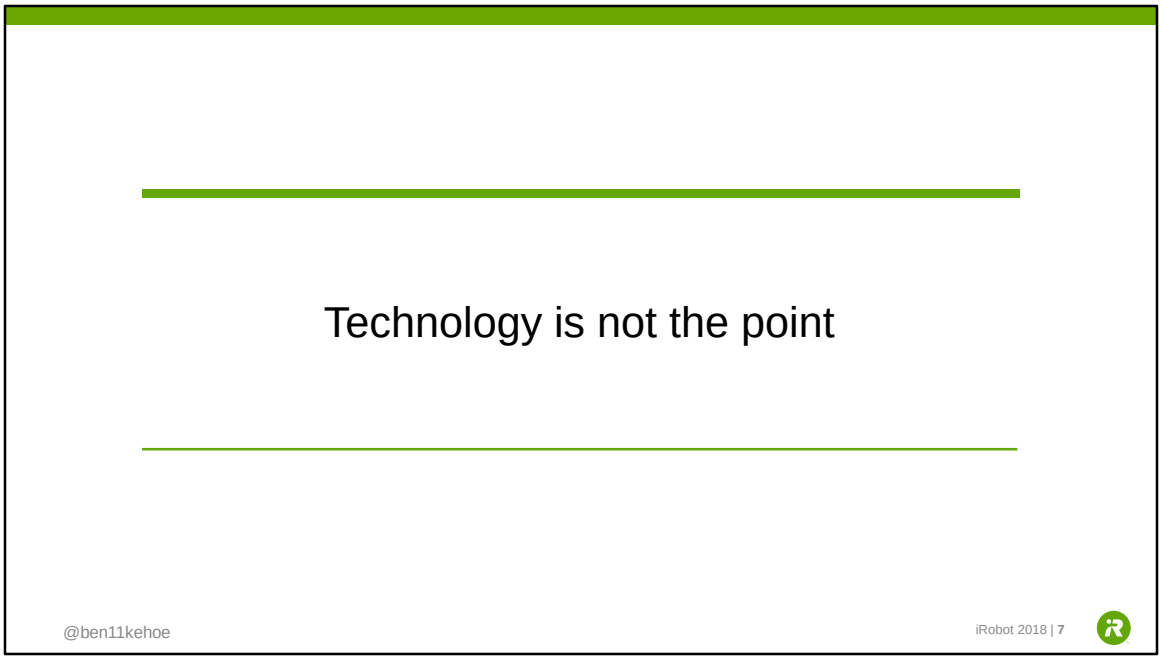
And managed services are great. But they, too, are not the point. We are fortunate to have such a wide range of managed services for so many different parts of our applications. Databases, identity and access management (so glad I don't have to own that myself!), analytics, machine learning, content delivery, message queues for all sorts of different patterns. Managed services provide the functionality you need with less hassle. You're not patching the servers they run on. You're not making sure the autoscaling is correctly providing the required throughput without a lot of idle capacity. It lowers your operational burden significantly…



But a lower operations burden is not the point. It's great to know that you can apply fewer operations resources to keep your applications healthy, and especially great that the resources you need scales mostly with the number of features you ship, not with traffic volume. On occasion you'll hear someone refer to serverless as "NoOps", but anybody actually doing serverless will tell you it's less ops, and different ops, new things like monitoring your provider. But it's still vastly reduced operations. This saves you money…



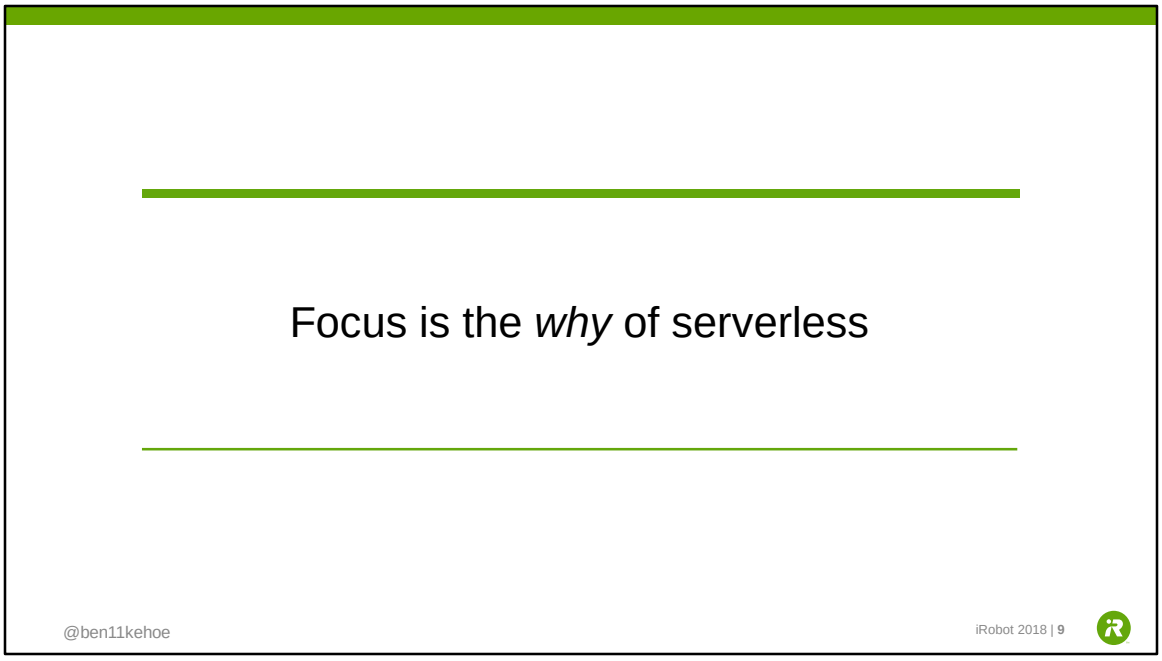
But your cloud bill is not the point. Well, sometimes all the business has asked you to do is reduce cost, and that's all you care about. And serverless will help you do that. But in general, your cloud bill is not the point. Your cloud bill is only one component of the total cost of your cloud applications. There's the operations salaries, first of all—and if you have fewer ops resources, that cost is lower. There's your development costs. But none of these are the point.



Because *technology is not the point*. The reason that we're doing this, any of this, is in service of some business goal. The customer value that your organization is trying to create *is the point*. Now, sometimes, what you're selling is literally technology. But even if your *product* is technology, that may not be the value—that may not be what you're selling. There's an old adage that people don't buy drills, they buy holes. When you need a hole in your wall, you don't care how fancy the drill is, you care how well it creates that hole you need. At iRobot, we don't sell robots. We don't even sell vacuums. We sell clean homes. Roomba gives you time back in your day to focus on the things that matter to you. So if technology isn't the point, what are we here for?

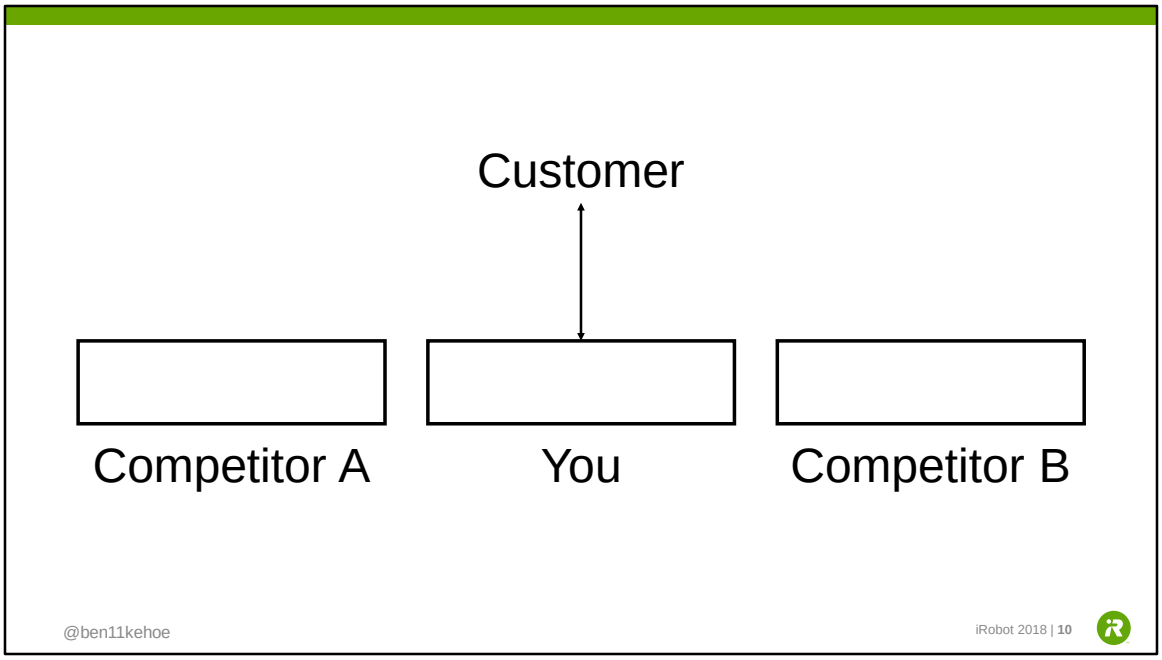
@ben11kehoe

The point is focus. Serverless is a way to focus on customer value. How do functions help you deliver value? They let you focus on writing business logic, not coding supporting infrastructure for your business logic. Managed services let you focus on writing your functions. Having less operations resources frees up people and money to be applied to creating new value for your customers. Observability gives you tools to address MTBF and MTTR, both of which are a measure of how often your customers aren't getting value. Spending less on the cloud means you can spend that money more directly in support of creating value.

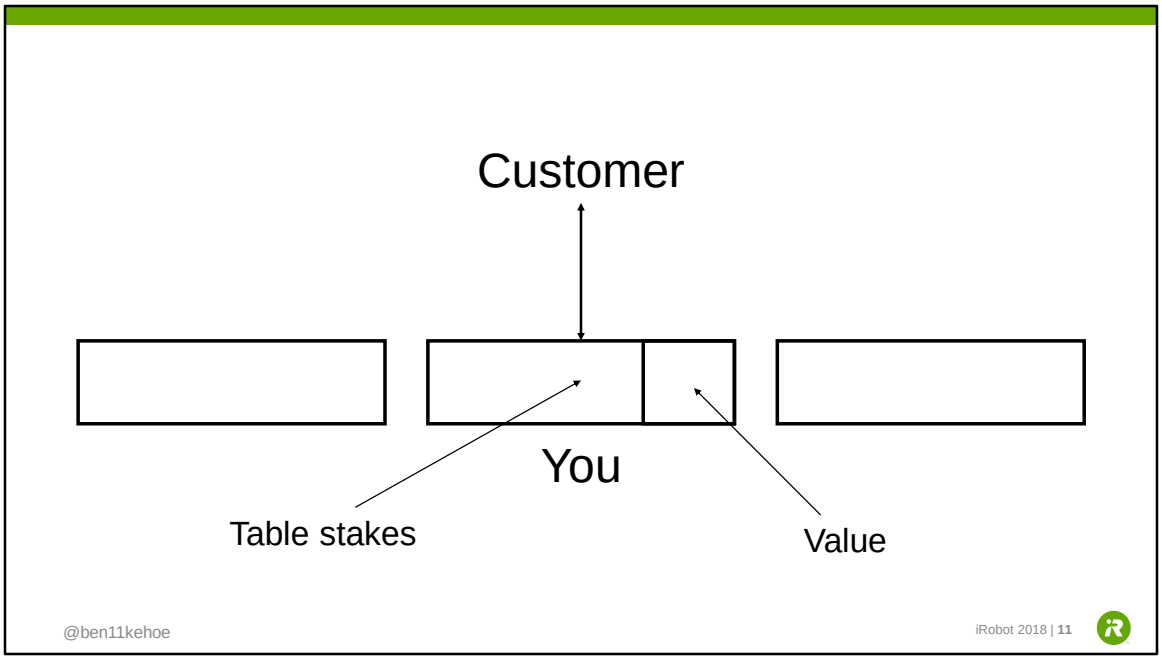


Focus is the why of serverless. You should be here because you want to focus on creating value, and at your company you are charged with applying technology to the creation of customer value. Going back to cost, Lyft's AWS bill, $100 million per year, has been in the news recently. Many people chimed in to say they could do it cheaper—they couldn't, but that's beside the point. Would Lyft's bill be lower if they switched to Lambda and managed services for everything they possibly could? Probably. But what would that do as they spent time rearchitecting? *They would lose focus*. The company is at a stage in its journey where growth is more important than cost control. Eventually, that might change. PUBLIC COs But right now, delivering value to their customers means executing with their current applications and processes. *They are making the serverless choice*.

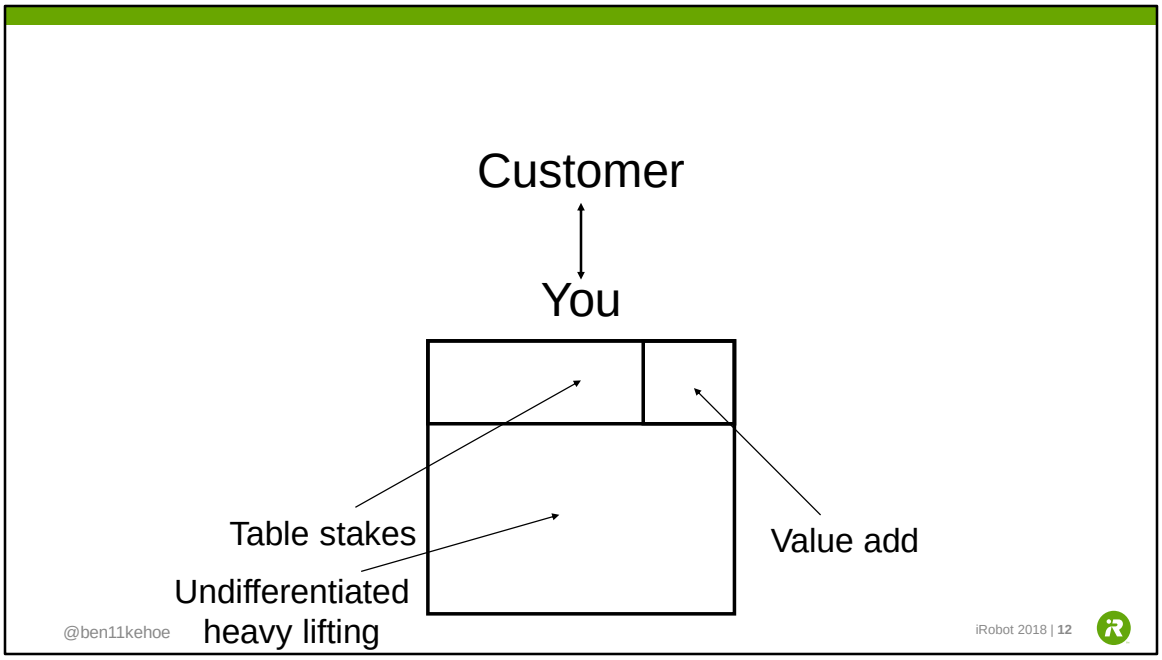
What I'm telling you is that serverless has never been about the technology we call serverless. So what *does* the technology that we call serverless have to do with it?



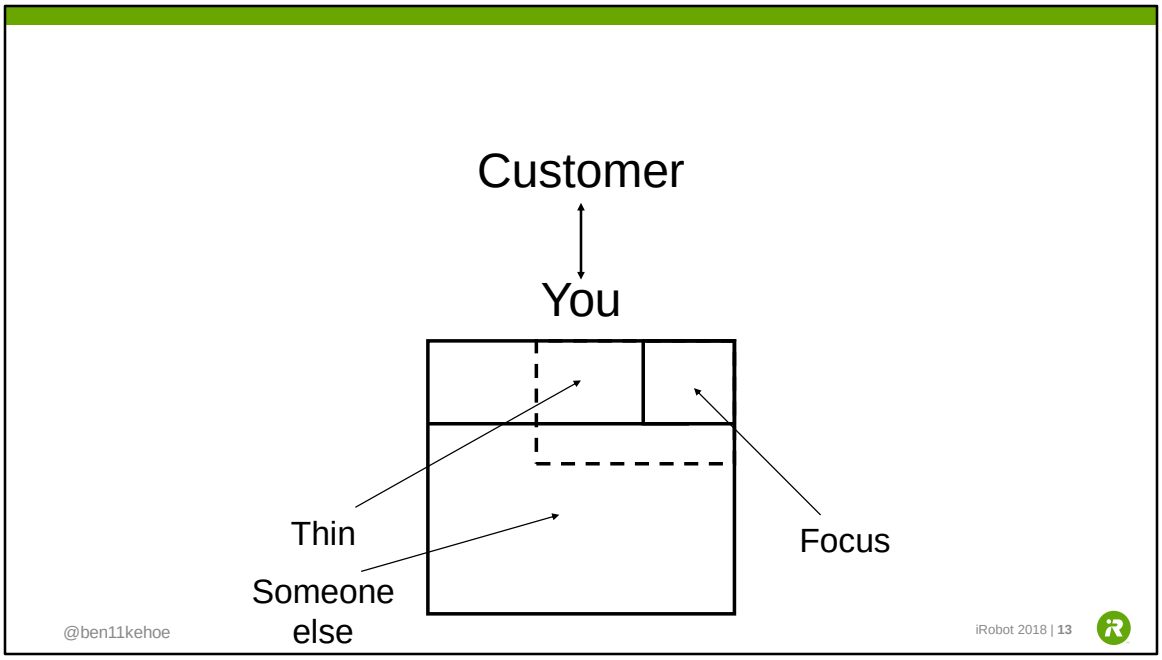
When you're delivering a product in a competitive marketplace, you're all selling boxes that are usually more or less the same shape. It's rare that you're in a situation where the options are radically different, or you are radically better than everybody else. So why would a customer choose you over your competitors?



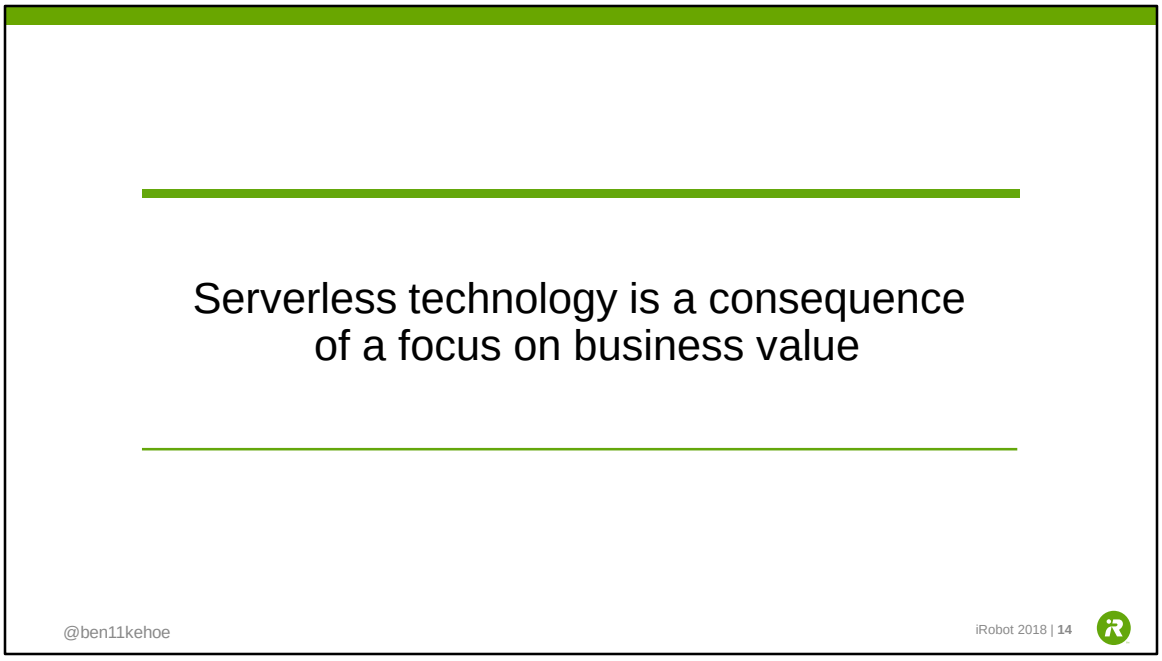
Unfortunately, most of your product is table stakes—what's needed just to play. Table stakes suck. Anything in this bucket can't win you points with your customers, but you can lose points. In a mobile app, you're going to have a settings page. If you do that wrong, people won't use your app. But if you do it 20% better than the next company, it doesn't really matter. Nobody's going to choose your app because it's got a fantastic settings page. And the worst part is, table stakes are really most of any given product. But somewhere in there is your core value. Your differentiator. What makes you different from everyone else (and identifying that is an important step). But all of this is only the product that the user sees.



What's worse is that it's only the tip of the iceberg. There's so much more required just to deliver that product in the first place. Technology-wise, you need source control, you need build infrastructure, you need web serving, you need licenses for 3rd party tech you might bring in. And never forget that you delivering technology is built on a host of corporate functions, like facilities and payroll. And the only thing that matters in all of this is that little chunk of value add up there in the corner. Everything else is, in some sense, a distraction from it.

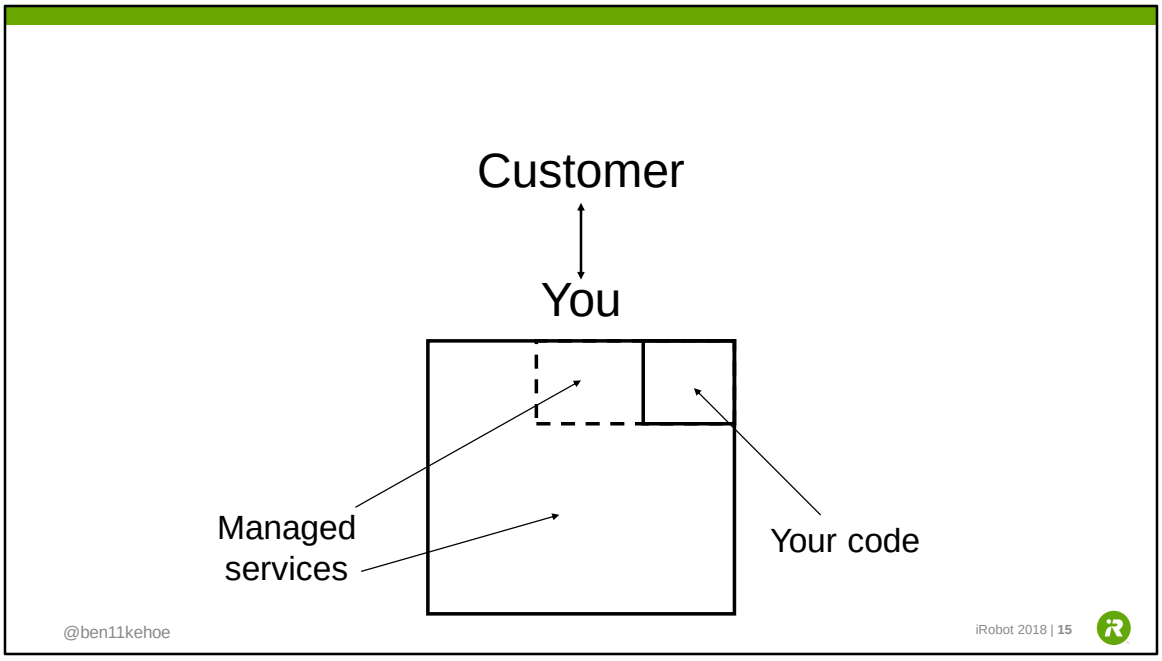


So what do you do? You get someone else to do it for you. This is already part of your organization. Breaking out smaller teams is about making sure the focus for a team is as tight as possible, so they can all be aligned, and rely on other teams for whatever is outside their focus area. Ok, you say, you buy what I've said so far, but what does this have to do with serverless technology?

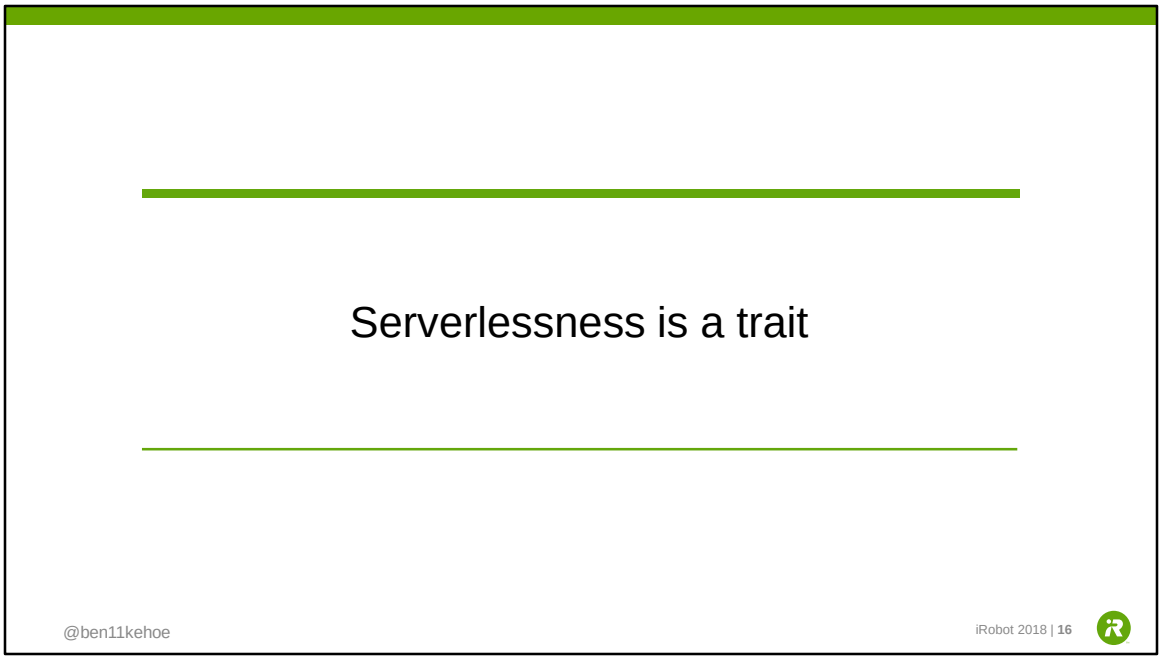


Technology is a consequence of how you're trying to deliver value. Dev and ops teams have traditionally been separated with the notion that they have different focuses. But we're seeing that trend changing. The traditional model put the focus on technology—dev tech vs ops tech. But we're seeing people realize that the focus should be on the value—the feature being delivered, including both how it's built and how it's run.

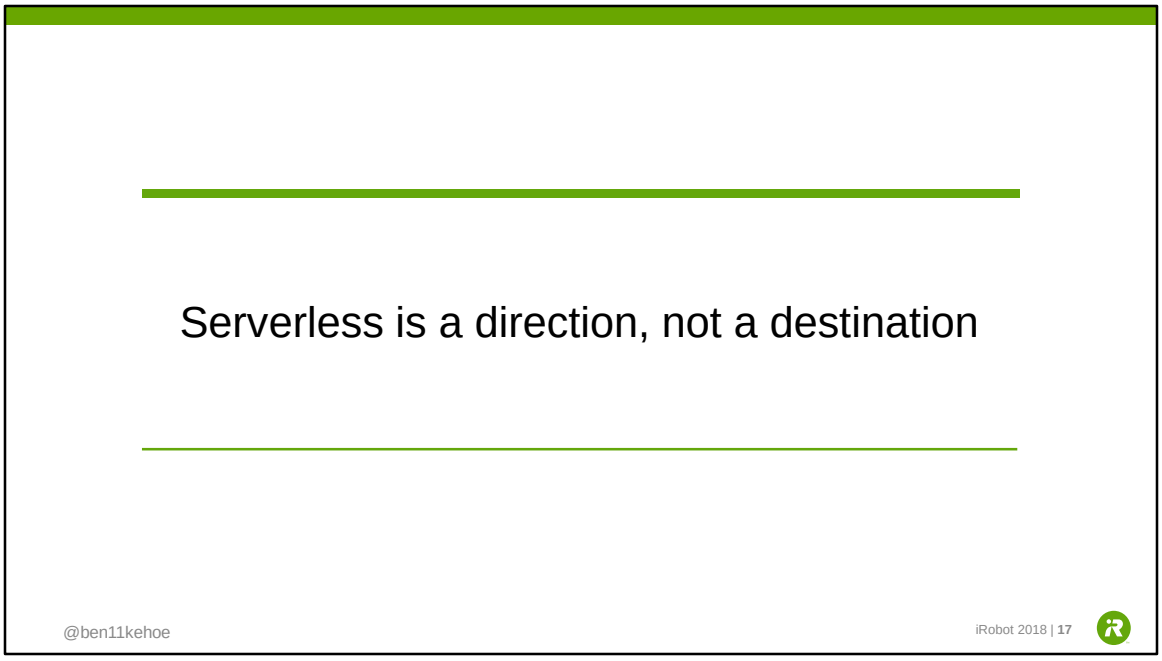
When we take this notion of focusing on customer value, and run it to its logical conclusion, we get serverless.



When you want to focus on delivering value, you want to write functions. When your function needs state, you want a database. To get it from someone else, you use DBaaS, and you choose between your options based on how well it lets you focus. And when you're choosing managed services, some of them may even be user-facing. If you can use social login instead of owning your own accounts, that's one less thing you have to manage, and one less piece of the user experience table stakes you need to own. Now, for everything you are outsourcing, you are still accountable. Your users don't care if their bad experience is caused by a third party you're using, it's still *your problem*. And you can own your outages to your users while accepting that you don't fully control your destiny there. This is an uncomfortable place to be, but it's worthwhile. As I mentioned earlier, you can't win points on these things, but you can lose points. This means that you need to know what "bad" looks like, so you have to have enough knowledge about the outsourced pieces of your product and your technology to know that you're delivering enough quality to your users. Note that deep expertise in a focus area, and broad but thin knowledge of adjacent areas is exactly analogous to the T-shaped skills concept, applied to organizations and teams.



And in this way, serverless is a trait. It's a trait of companies. A company is serverless if it decides that it shouldn't own technology that isn't core to delivering value to its customers. Few companies are really totally serverless. But within a company, you can still have parts that are serverless. If your team decides to focus only on the value it's delivering, and delegate anything outside that either to another team, or ideally outside, your team is going serverless. And you can't always choose to use an outside technology—that's fine, you can still make the best choice given the constraints. And with a big enough organization, it can cease to matter. When Amazon.com uses Lambda, that's fully serverless, even though it's on-prem in some sense. But what if it's just you? What if you've come here today, because you're excited about serverless, but you feel completely alone at your company? What if you're far removed from actual customer value—if your team is patching servers for a team that serves a team that creates user-facing content? I want to convince you that you can go serverless today, yourself, in any situation. This is where my talk turns into a motivational speech.

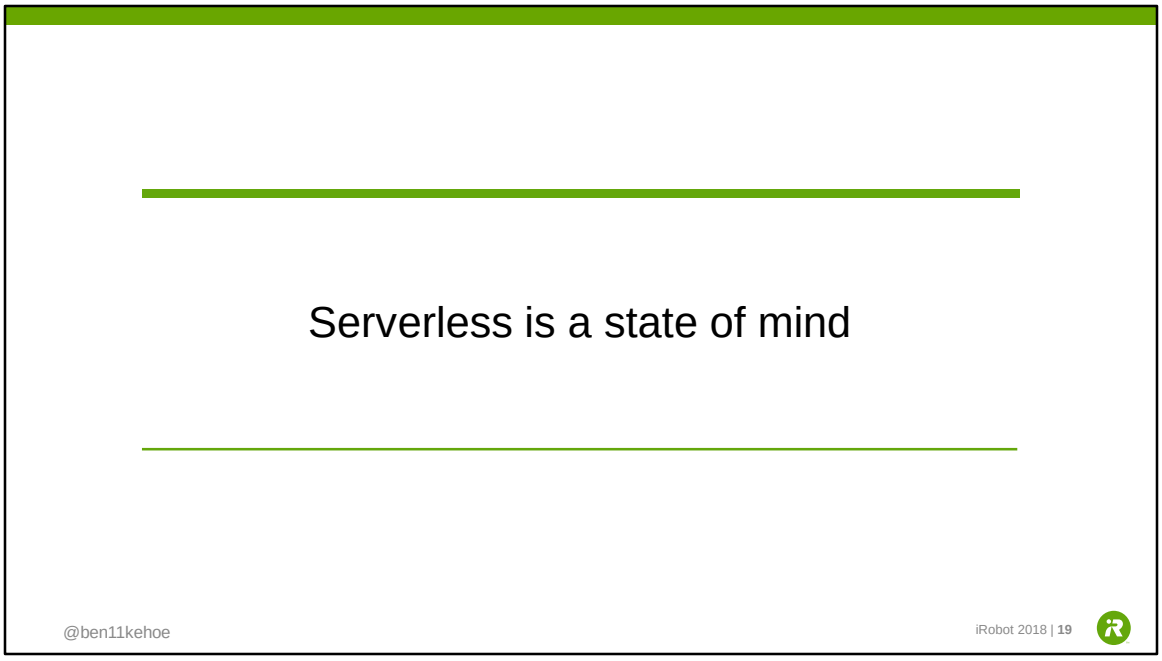


Because serverless is a direction, it's not a destination. I used to talk about serverless as a spectrum, because I knew there wasn't a bright line separating serverless technology from non-serverless technology. I mean, there almost never is a bright line separating any given grouping of anything, so I was pretty safe in that assumption. I talked about how something like Kinesis, where you need to manage shards, is serverless, but less serverless than SQS, where you don't. How you don't have to patch instances with RDS, but you do need to choose instance types and number. These technologies are all various shades of serverless. But recently I've come to realize a problem with portraying serverless as a spectrum is that it doesn't imply movement. Just because you're using something designated serverless of a sort doesn't mean you should feel comfortable that you've attained serverless, that it's acceptable to keep using that and think you've checked the serverless box.

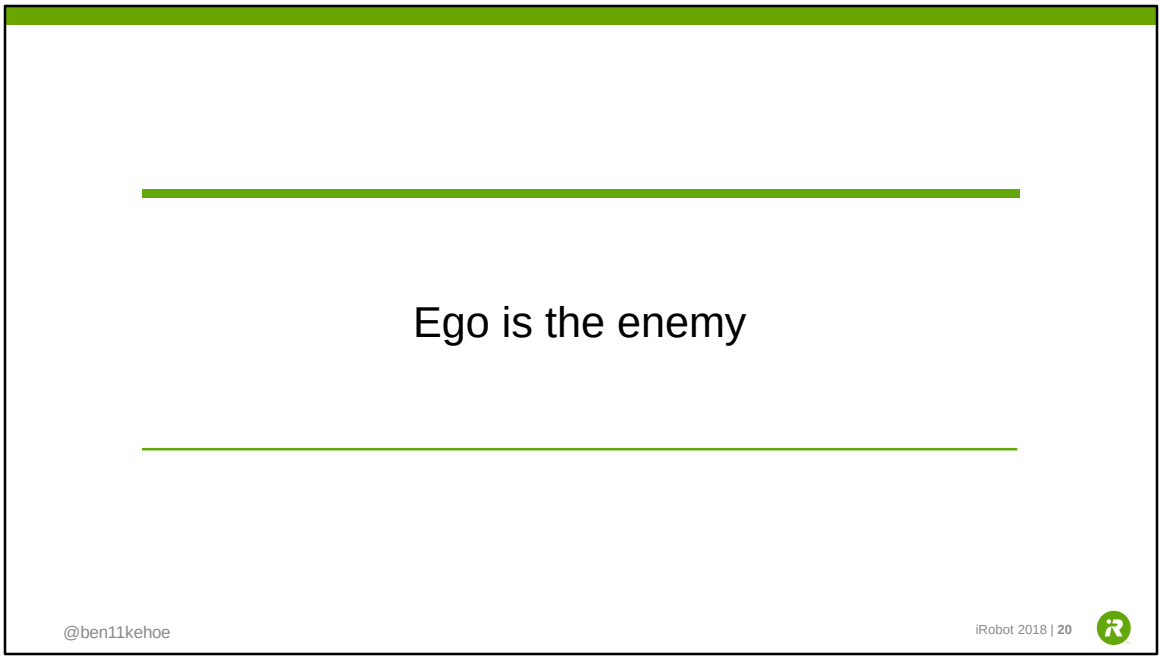
@ben11kehoe

iRobot 2018 | 18

I've started to think of serverless as a ladder. You're climbing to some nirvana where you get to deliver pure customer value with no overhead. But every rung on the ladder is a valid serverless step. If you move from on-prem to a public cloud, that's a rung on the ladder. If you move from VMs to containers, that's a rung on the ladder. If you move from no container orchestration, or custom orchestration, to Kubernetes, that's a rung on the ladder. If you move from long-lived servers to self-hosted FaaS, that's a rung on the ladder. But there's always a rung above you, and you should always keep climbing. One thing "ladder" doesn't convey is that it's not linear. Moving from VMs to containers to Kubernetes while staying on-prem are rungs on the ladder, but so is moving your VMs from on-prem to the cloud. There's often not a definitive "better" in these cases. I thought of the metaphor of many paths leading up a mountain, but one thing I like about the ladder is that it can be infinite. There isn't an end state. I love Lambda, but I am always looking for better ways of delivering code that keep me more focused on value.



So, like, serverless is a state of mind, man. It's about how you make decisions, not what your choices are. Every decision is made with constraints. But if you know the right direction, even when you can't move directly that way, you can take the choice that's most closely aligned, and then you're moving up another rung. So, how do you adopt this mindset? How do you make serverless choices?



To start, as Ryan Holiday says, ego is the enemy. A big part of truly embracing the serverless mindset is a change in how we operate as engineers. Depending on how we construct our identity—if we identify with the technology we work with or produce—this may be difficult. It also involves, to a large degree, an acceptance of a lack of control. Both of these changes require us to be humble.

Developers are part of something bigger than ourselves

@ben11kehoe

Because we're not the center of the universe. As developers and operators we've been heads-down because technology is hard. But as technology becomes easier to deliver, we have to look up and stop taking the rest of the organization for granted.

What are we trying to accomplish? We have a product feature hypothesis that we want to get in front of customers, so that it can be tested. That's what's important. Development stands between that hypothesis and customers. But development is bigger than developers. It's an on-going process involving more stakeholders than just developers. Product owners, designers, QA, operations, security. As developers, we feel at the center of it. But we're not.

Developers are not the point

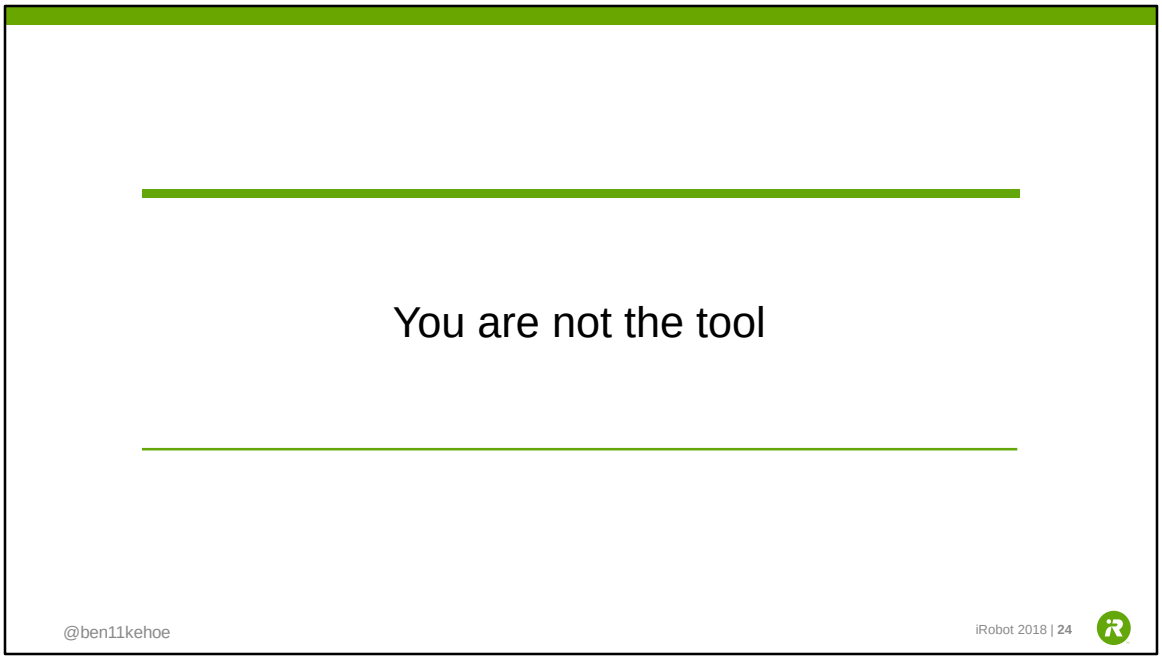
@ben11kehoe

iRobot 2018 | 23

Technology has been hard for a long time. It's taken clever people to create value through technology. So developers started to believe that cleverness was inherent and good. We've spent so long crafting Swiss watches that we've failed to recognize the advent of the quartz Casio, and impugn it as lacking in elegance. But what we want is to take that ability to be clever and apply it not to solving technology problems, but understanding and solving business problems. And when you have to code, you're solving technology problems.

In the watch analogy, we have to accept that our skills for gears and springs are less necessary, and move into electrical engineering.

As I said earlier, this is the hardest when we have constructed an identity around solving technology problems. But what we should view ourselves as is deliverers of technology-enabled value.



And this is important, because you are not the tool. The tool, any tool or technology you've been using has been an enabler for you. But it's time to either pass that tool to a team or company supporting you—in the case of serverless managed services, or to pass it to someone you support—in the case of build tooling, for example, by making it self-service. Take tool owner and user and split them, and become the one that provides value. But all the tools *are not you*.

Serverless development is a consequence of a focus on business value

@ben11kehoe

iRobot 2018 | 25

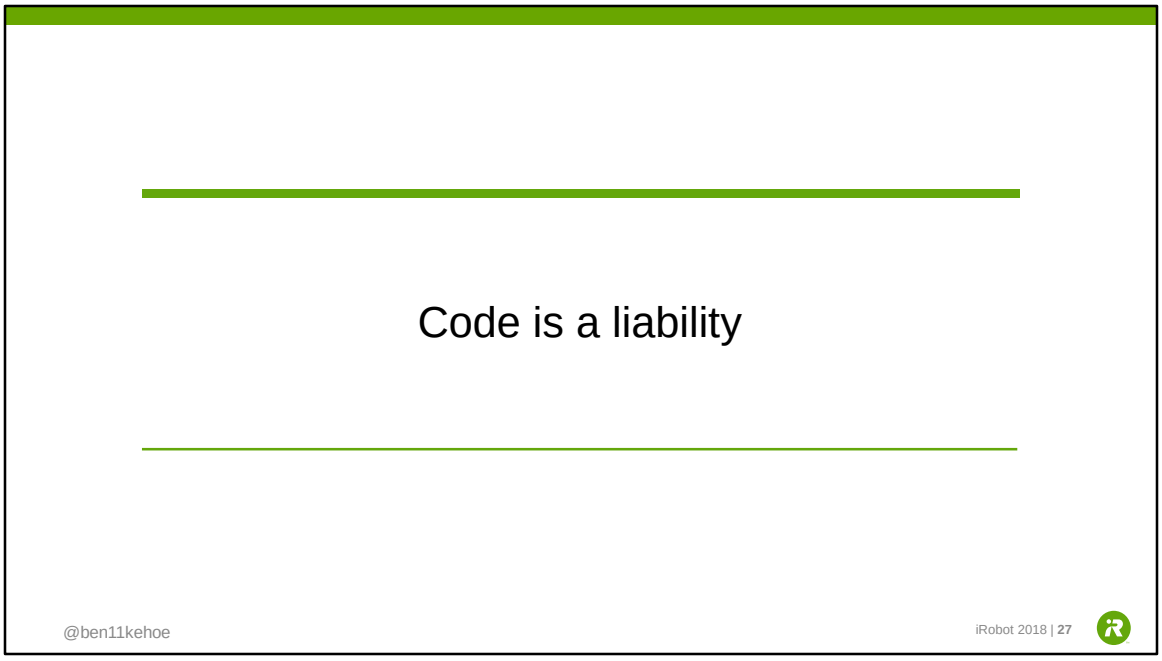
Because tool selection should not by driven by *people*, it should be driven by *value*.

Jumping in to serverless

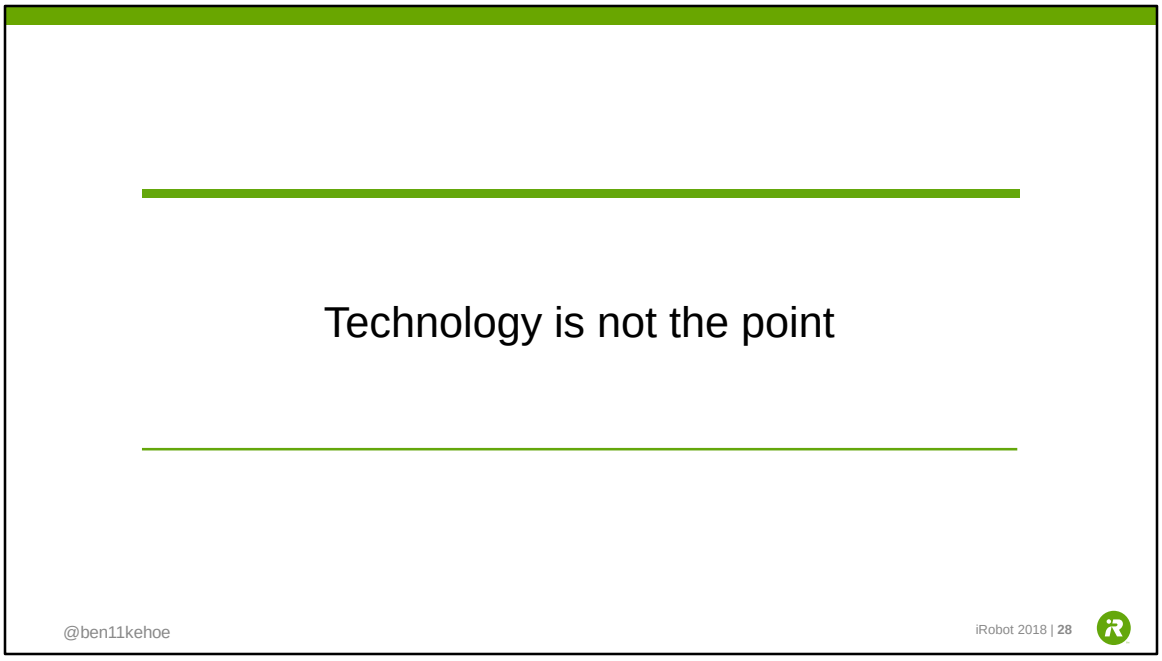
@ben11kehoe

iRobot 2018 | 26

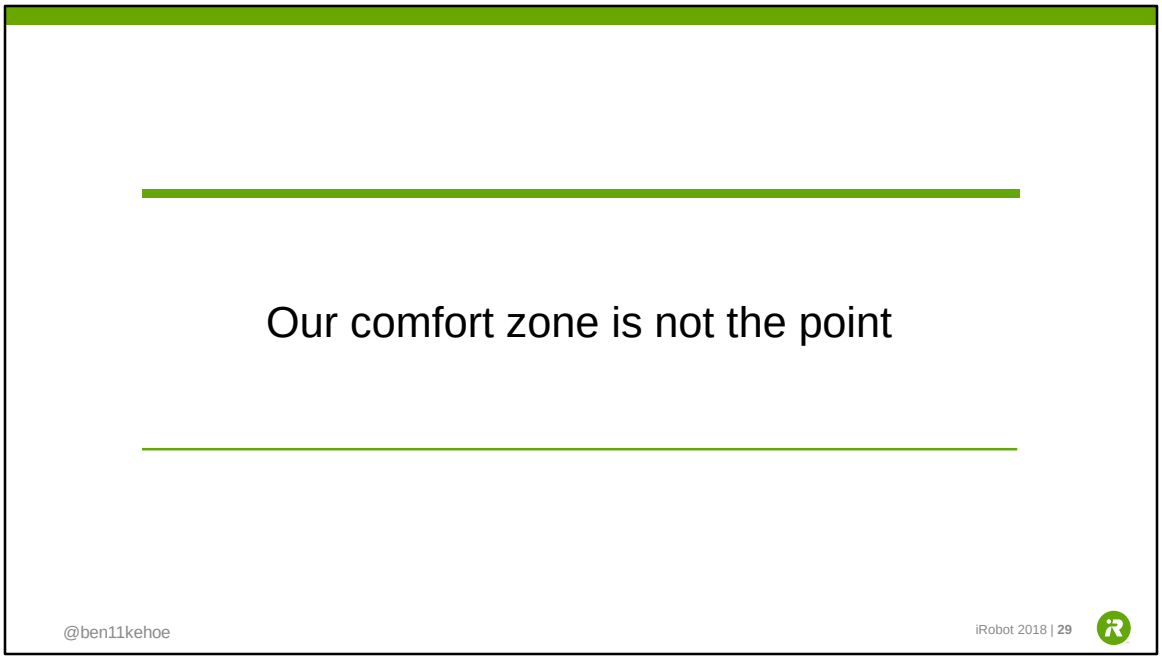
So you say, “serverless sounds great!” and jump in to set up a web app. The first thing you hit is, how do you handle web requests with Lambda? Well, you know Express, and there’s a framework to use Express in Lambda. Great, you just use that, and set up API Gateway as a passthrough. You set up DynamoDB, and where you’ve got simple CRUD operations, you write those in Lambda. Now you’ve got a proof of concept configured, but you want it in infrastructure-as-code. You find a framework that lets you write all your function code and infrastructure in the same JavaScript file, and it takes care of deploying everything. Great, right?



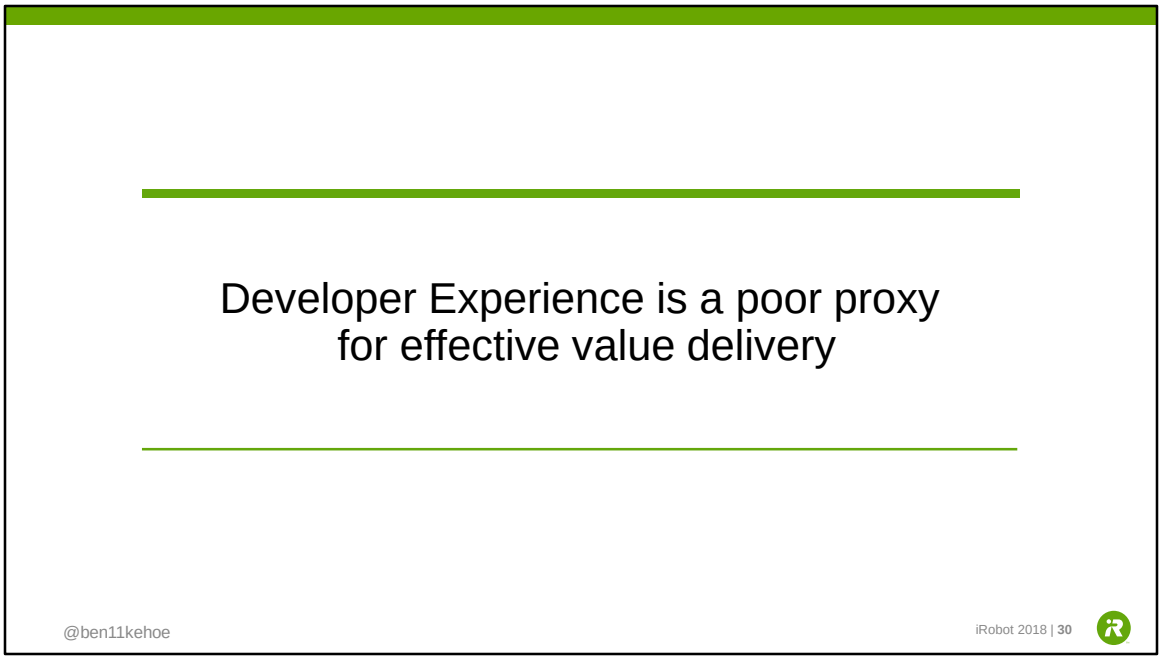
You've got way more code than you need in this. Code is not an asset, it's a liability. Just like the table stakes product features we talked about earlier, code can *at best* do exactly what you intend it to. Bugs subtract from this. You can only lose points through coding. The implementation can only be imperfect. The more code you own, the more opportunities from departing from your intended value there are.
First, API Gateway can handle a lot of the web request processing, for free; you've already paid for the request, so extract as much value out of the service as possible. Validate your requests there, that's code taken out of your function. Route API methods to separate functions, so you don't need routing logic in your functions. And if all you're doing in a function is an API call to DynamoDB, API Gateway can do that directly, and you've just eliminated a function entirely. And you can use CloudFormation to deploy your app.
But API Gateway models require you to learn JSON Schema, and service integrations require you to learn Velocity Templates. CloudFormation is in YAML. But you know JavaScript, isn't it better when you can just write code and it's done?



Remember that technology is not the point. And so here we should ask how can we best deliver value?



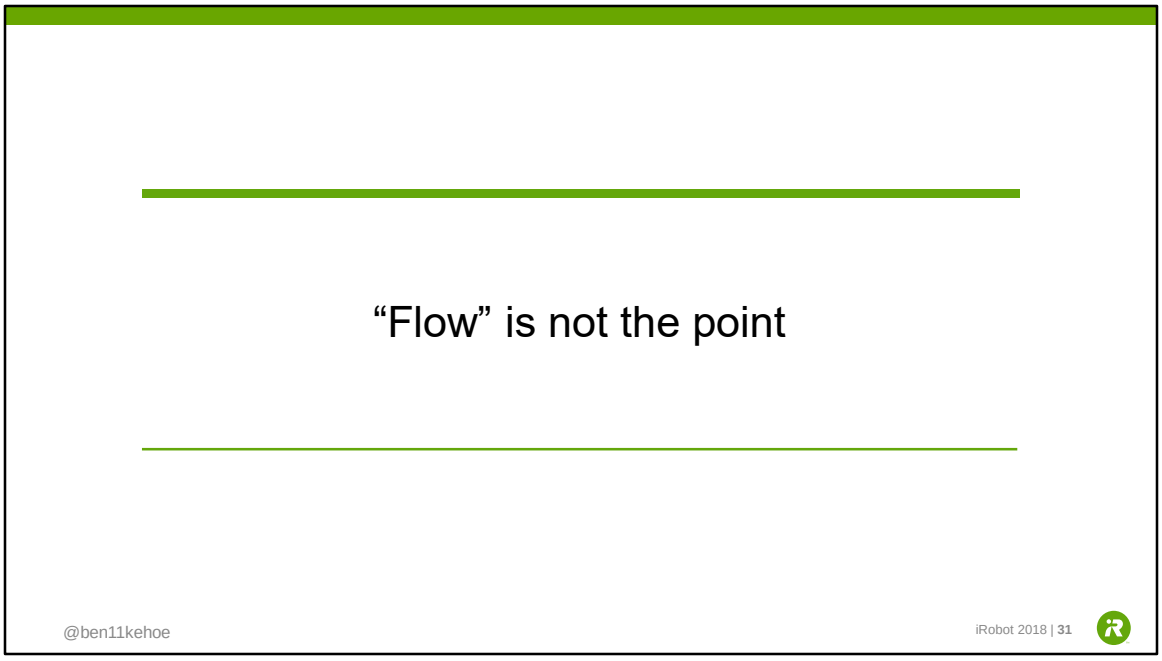
The answer may not be what's directly *easiest* or *most fun* for us. Just because something is harder for us does not mean it's not more effectively delivering value. Serverless development is not about what makes us the most comfortable—we have to stretch our boundaries to become more effective.



There's an idolatry of coding today. We've been told that "software is eating the world", and we've inaccurately translated that to "coding is eating the world". We've come to believe that developers are the only important people in an organization, and that our *sense* of productivity is the only thing that matters.

We've seen this before, when dev is separated from ops, what feels productive to developers to get something out the door and thrown over the wall to ops may not produce something easy to operate, which will suck resources into operations or even cause more maintenance work for developers. We can ease this with "you build it, you run it", but it's not possible to directly incentivize developers for everything that lowers TCO; we have to look for it.

We need to optimize development, and I want to convince you that what makes us as developers *feel* best is a bad guide for that optimization.



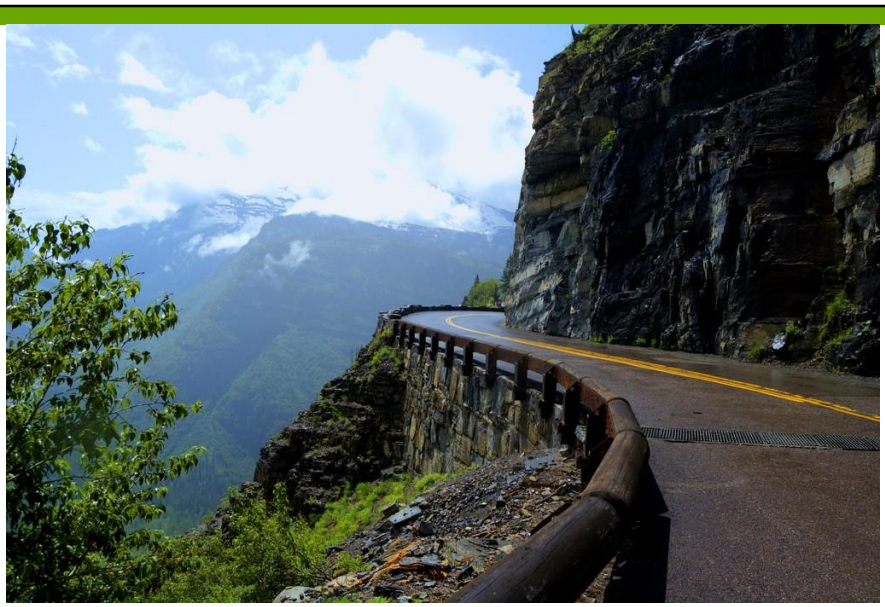
Flow, or deep work, or being in the zone, is a great feeling. Everyone loves that sense of productivity. But it’s a means to an end. Flow is useful, even required, to create complexity. And as I’ve said, technology has required complexity for a long time. And coding is really, really good at hitting that flow button in us. But flow is not the point. Say you’re about to leave work, and you look at traffic. On the highway, it’s stop-and-go. You can take side streets, which are free-flowing, but it will take you a bit longer to get home. Ignore what would happen if everybody chose the streets, etc. Just this one choice. How many people here would take the highway? How many would take the streets? I performed the gold standard for surveys, a Twitter poll, and 75% of the respondents said the streets. I’m a side-streets person myself! It feels so much better just to be moving. But now, what if you were a delivery driver? You want to fit more deliveries in a day, so you’re going to sit in that traffic. As developers, we need to prioritize actual delivery of value in the big picture over our narrowly-scoped feeling of productivity, because it may not *actually be productive* when you zoom out.

Introspection is key

@ben11kehoe

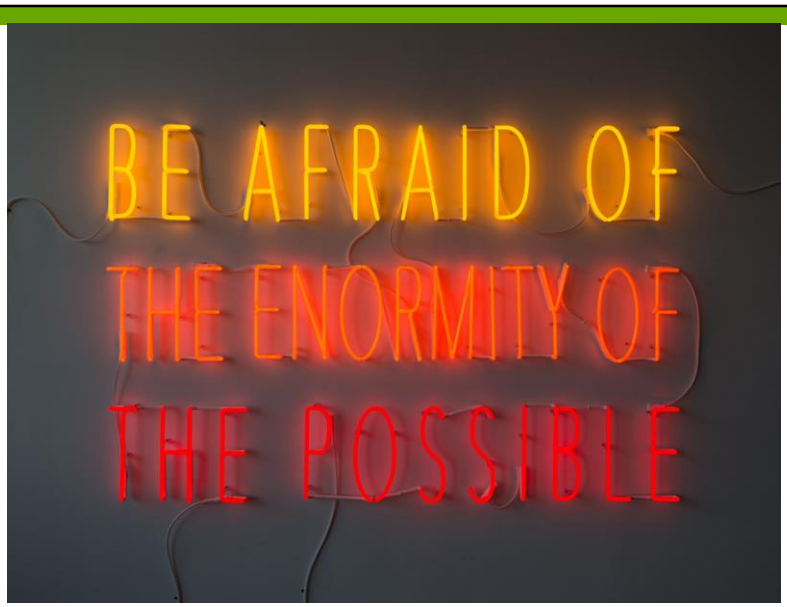
So being introspective is important as part of this process, because it helps you zoom out. Take time to step back and evaluate your methods. Where are you being inflexible? Is it because whatever you're committed to actually delivers value, or just because you know it well, or feel it is better in some other sense?
What biases, beliefs, and attachments have you built up that may be getting in the way of better serving the customer?

Serverless is about minimalism. Letting go. Removing distractions. Owning less technology. Marie Kondo is big now, and the same advice applies. Find the components of your stack that don't spark joy for your business.



@ben11kehoe

Part of this whole story is that constraints can be good. Removing options can help you focus. Obviously, not all constraints are good, but in general, the ability to *do anything* comes at the cost of it taking longer to *do one particular thing*. Guard rails may chafe, but you'll be faster than if you have constantly watch the edge.



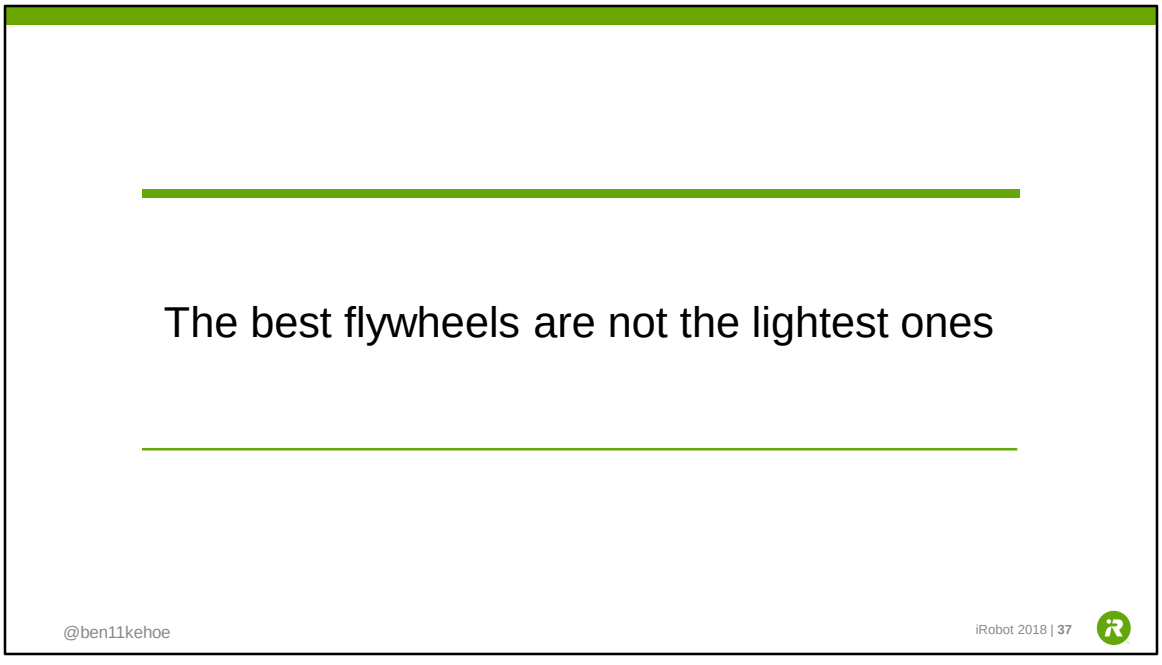
@ben11kehoe

I came across this photo, “Be afraid of the enormity of the possible” and I like it. Possibilities carry with them hidden complexity. For any technology, one of my primarily evaluation metrics is how much capability it has beyond the task I am applying it to. Because when there’s a lot of extra space, there’s unnecessary complexity that I have to both learn about and deal with. People tout Kubernetes as a single tool to accomplish every cloud need. And it can! But if everything is possible, anything is possible. For a given task, Kubernetes can go wrong because you haven’t accounted for the ways it acts for situations unrelated that task. On the other hand, when you’re look at serverless services, you may have to choose between a 80% solution from your main provider, or a 3rd party provider with a service that better fits your needs. But what are the operations needs for that new provider? What’s the auth like? Those are hidden complexities that you’ll pull in, and you’ll need to trade that off with feature differences.

Spent the day messing around with @AWSAmplify. Awesome tool. I've got GraphQL, auth, and file storage all set up in a Gatsby-powered web app. It's always annoying to configure and work with AWS, but the time saved is ridiculous.

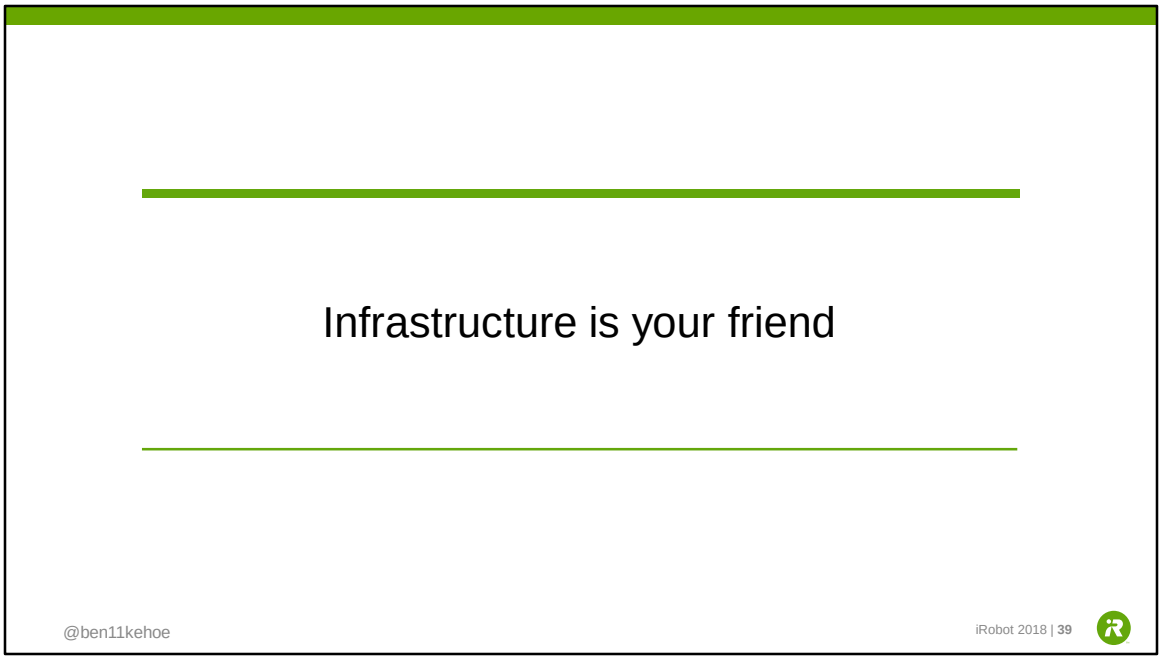
@andrew_j_mead

This tweet gets to the heart of the serverless mindset. How do you evaluate a technology?
Annoying. A new thing to learn. Using it may not hit that flow feeling in the same way. But it *saves you time*.



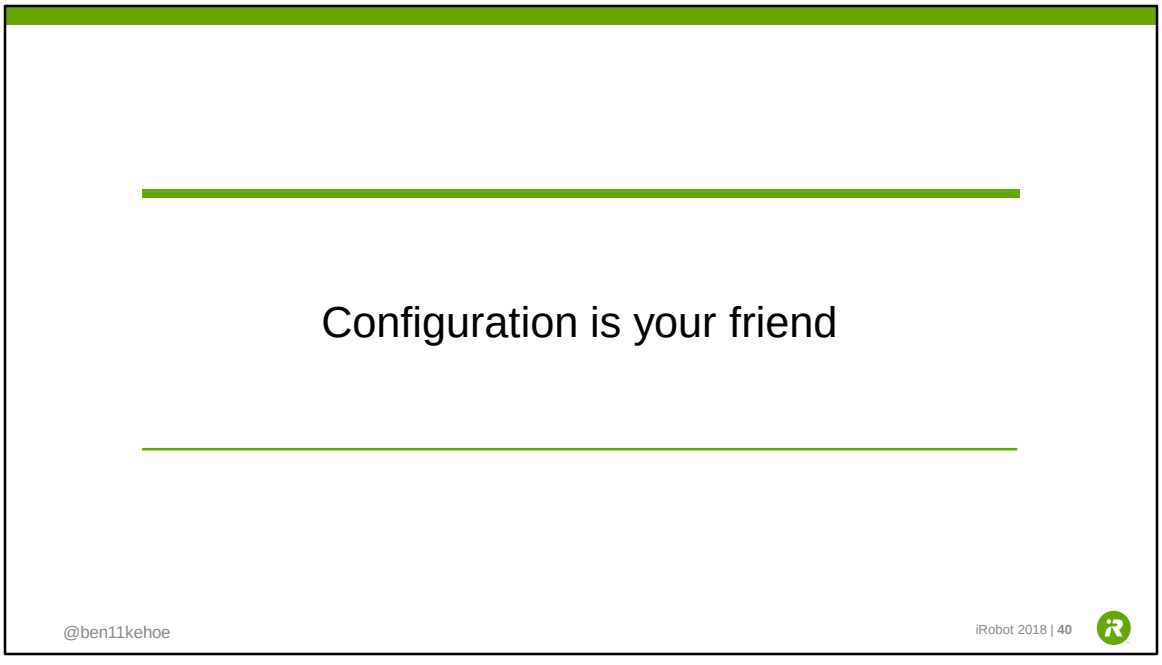
The best flywheels are not the lightest ones. They aren't the heaviest, for sure. But the point of a flywheel is momentum, and mass aids momentum. If you have to learn something new, something more, remember: just because it takes a little longer to start doesn't mean it's not going to be more effective in the long run. The "hello world" for a technology is almost never indicative of its overall effectiveness.

Days of programming can save you hours of configuration

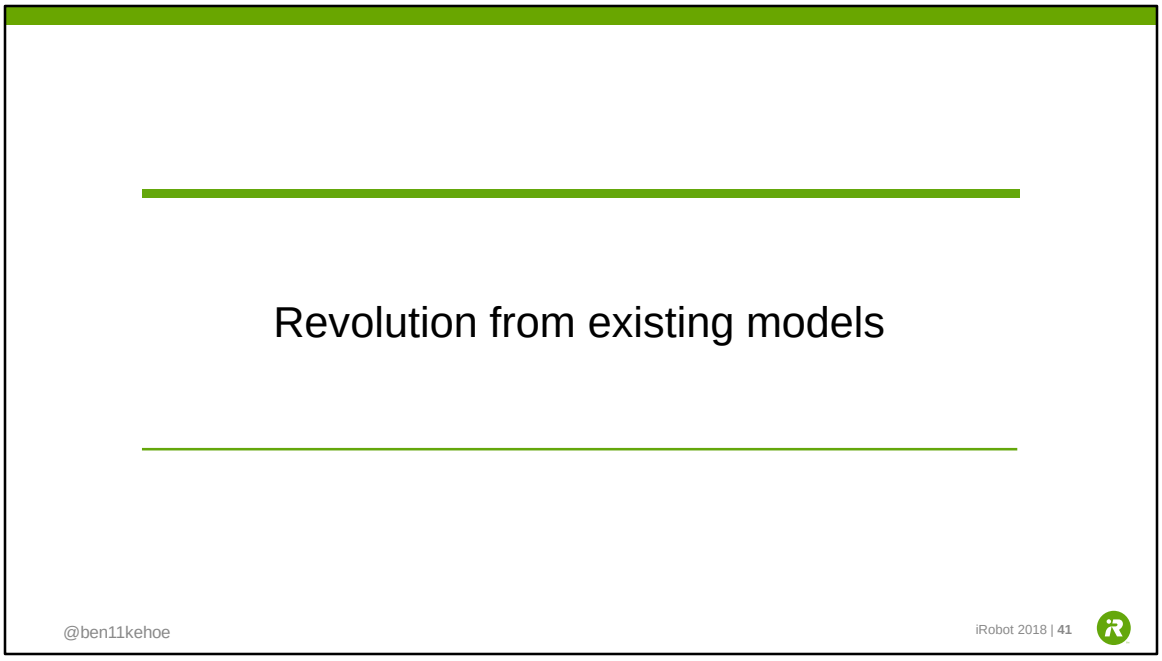


In serverless, we have a lot of infrastructure, because we have a lot of managed services. Instead of our application being defined by the execution graph inside our code, it's defined by the infrastructure graph, which connects the services and the few places where we need custom code. And that's great—less custom code means fewer places to screw things up.

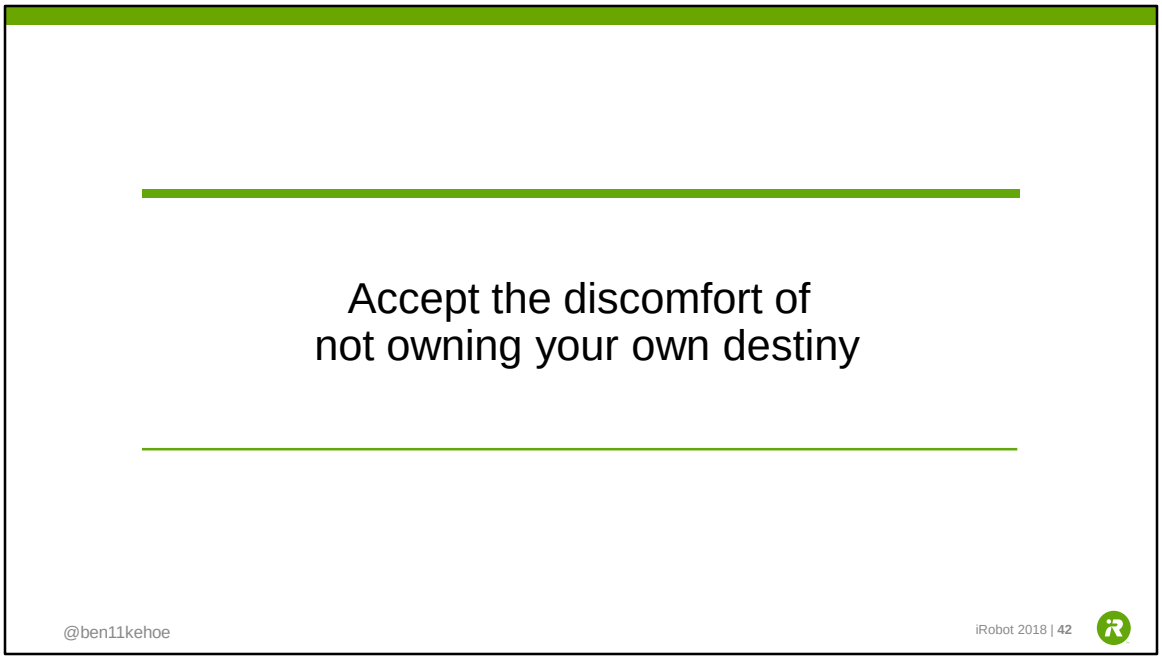
So then we get to the question of defining that infrastructure graph. And too often, we reach for coding—at worst, writing code that goes into a local tool responsible for configuring services directly. But the serverless way is to use infrastructure deployment by managed services, like CloudFormation on AWS. You don't worry about keeping different dev's clients in sync, and the statefulness is managed for you.



Here, I mean configuration as a non-Turing complete, declarative language. So we're still talking "infrastructure-as-code", but it's the nature of the code, it's not a program written in the imperative languages we use for other things, it's a document. I think developers look down on configuration as "not real programming". And because configuration tends to be domain-specific, we don't have the same advanced tools and IDEs to make creation of them as easy.



Once we're open to new things, serverless providers can enable better, more managed services by breaking from existing models. Google Cloud has gone all-in on containers and Kubernetes, but it means that while Google Cloud Functions takes zip files, any form of bringing code more complicated than that is always going to be a container image. AWS Lambda thought for a long time about how to bring more code and custom runtimes, and landed on layers, which allows the OS to stay fully managed, but code in the function can come from multiple, versioned sources, and the runtime can either be managed or custom, and all these pieces can come from and be maintained and updated by independent sources, without the function owner having to know and rebuild a container image or provide a pipeline that does.



When you're using a managed service, provider outages are stressful. There's nothing you can do to fix their problem. There is no getting around it, this will always feel awful. You'll think, "if I was running my own Kafka cluster instead of using Kinesis, I could find the issue and fix it". And that may be true, but you should remember two things: first and foremost, that would be a distraction from creating business value. But also, you would almost certainly be worse at running it. You'd have more and worse incidents. It's a service provider's purpose in life to be good at it. They have economies of scale you don't. Moving past the "I could always build it myself" attitude can be hard.

1. If the platform has it, use it
2. If the market has it, buy it
3. If you can reconsider requirements, do it
4. If you have to build it, own it

@shortjared

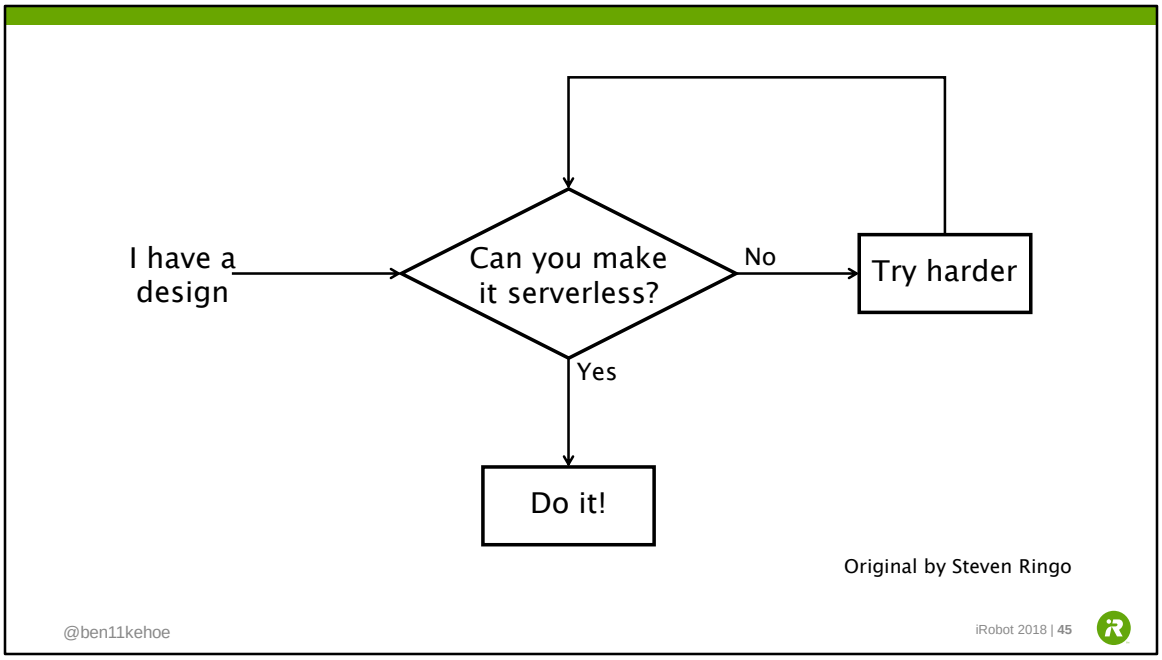
Jared Short recently provided a brilliant set of guidelines for choosing technology. In order, if you're on a cloud platform, stay within the ecosystem when possible. You're removing so many possibilities from the equation that way. But if you can't get what you need on the platform, buy it from somewhere else. If you can't buy exactly what you need, can you rethink what you're doing to fit what you *can* buy?

1. If the platform has it, use it
2. If the market has it, buy it
3. *If you can reconsider requirements, do it*
4. If you have to build it, own it

@shortjared

@ben11kehoe

This one is really important. It gets to the heart of time-to-market. You have something you think is valuable, and you want to ship it. But it's better to ship something *near* to that faster, than to build the exact thing—you don't know that it's the *right* thing yet, and not only will it take longer to ship, but your subsequent iterations on it will be slower, *and* maintenance of it will take resources that you could apply to shipping more things in the future. In agile, we accept that requirements can change, but this is about requirement change being a *dialogue* with product ownership, rather than unidirectional.

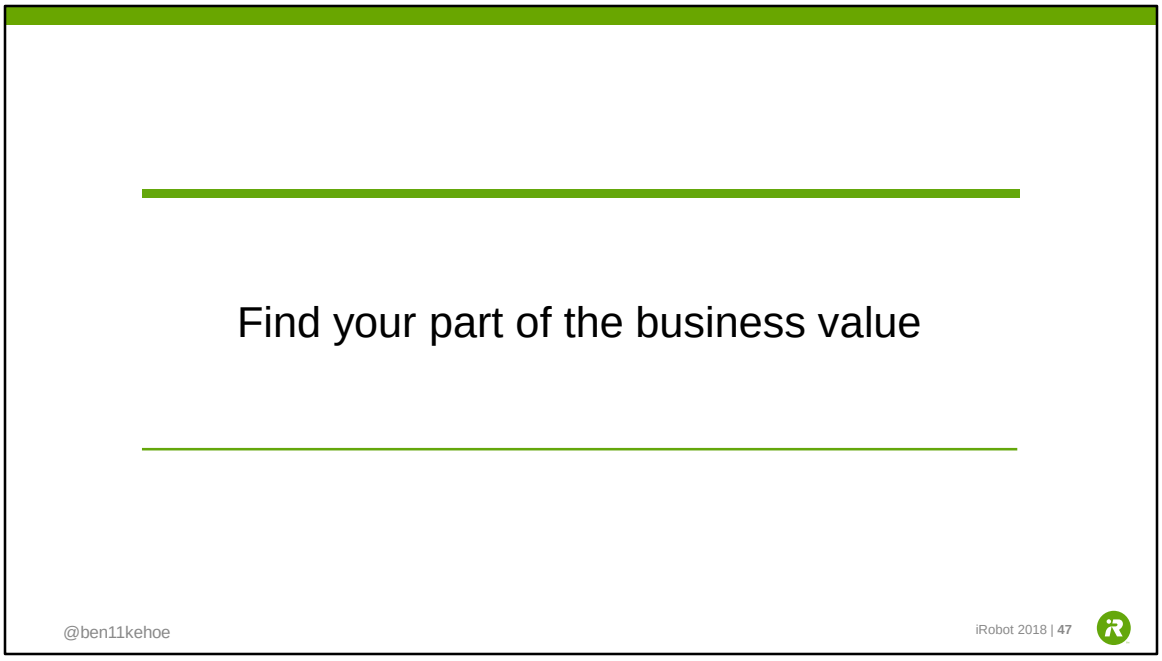


This diagram is funny, but in any situation, always look for tradeoffs where you can own less technology by making a tweak to the requirements

1. If the platform has it, use it
2. If the market has it, buy it
3. If you can reconsider requirements, do it
4. *If you have to build it, own it*

@shortjared

Finally, though, if you *have* to build it, *own it*. Find a way for it to be a differentiator. Now, this doesn't mean everything you've built already you should turn into a differentiator. Look at only the things you *can't have bought as a service in a perfect world*. Imagine what a completely serverless, greenfield implementation would look like, and find what needs to be built there.



So fundamentally, you want to find your part of the business value. What is your technology work in service of? Maybe you're far removed from user-facing product. You may only be contributing a small slice. But it's there, and you can find it, and focus on it. Start with the immediate value you're providing to others in the organization, and focus on that. And then start to trace the value chain. Make sure all your decisions are oriented around the value you're creating. Make serverless choices.

Hire the people who will automate themselves out of a job, then just keep giving them jobs.

Jessie Frazelle

And remember that you are not the tool. For any value that you're creating, automate that creation. If you manage build servers, find ways to make them self-service, so what you're delivering is not the builds per se, but the build tooling so teams can deliver the builds themselves. I love this quote, and you can turn it around; automate yourself out of a job, and keep demanding new jobs.

Summing up

- Serverless is about focus
- Technology is not the point
- The nature of development is changing
- Find your part of business value
- Make serverless choices

Thank you

iRobot