EPISODE 386

[INTRODUCTION]

[0:00:00.4] JM: Continuous delivery is a model for deploying small frequent changes to an application. In a continuous delivery workflow, code changes that are pushed we repository set off a build process that spins up a new version of the application. Testing is performed against the new build before advancing it to production, merging it with the existing code base. Many continuous delivery products are getting built today because it's a wide open space, much like cloud providers or monitoring tools. There’s just a lot of players. There are subjective product and engineering decisions to be made depending on the audience for your product. Heroku Flow is a continuous delivery platform built on top of Heroku, which is a platform as a service.

Andy Appleton is an engineer at Heroku and he joins the show to describe how Heroku Flow was built. Two years of work went into the project from initial conception to launch, and it's a good story. Full disclosure, Heroku is a sponsor of Software Engineering Daily. I also use the product a ton on my own. I used to Heroku Flow and I use Heroku and did I really enjoy them. I use them for the products that I'm invested quite heavily in, so I'm happy to have Heroku as a sponsor of the show.

[SPONSOR MESSAGE]

[0:01:27.3] JM: Spring is a season of growth and change. Have you been thinking you’d be happier at a new job? If you’re dreaming about a new job and have been waiting for the right time to make a move, go to hire.com/sedaily today.

Hired makes finding work enjoyable. Hired uses an algorithmic job-matching tool in combination with a talent advocate who will walk you through the process of finding a better job. Maybe you want more flexible hours, or more money, or remote work. Maybe you work at Zillow, or Squarespace, or Postmates, or some of the other top technology companies that are desperately looking for engineers on Hired. You and your skills are in high demand. You listen to a software engineering podcast in your spare time, so you’re clearly passionate about technology.

Check out hired.com/sedaily to get a special offer for Software Engineering Daily listeners. A $600 signing bonus from Hired when you find that great job that gives you the respect and the salary that you deserve as a talented engineer. I love Hired because it puts you in charge.

Go to hired.com/sedaily, and thanks to Hired for being a continued long-running sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:02:57.1] JM: Andy Appleton is an engineer at Heroku. Andy, welcome to Software Engineering Daily.

[0:03:01.1] AA: Hey, thanks for having me.

[0:03:02.4] JM: Today we're going to talk about continuous delivery, and specifically continues delivery on top of a platform as a service. There are a ton of continuous delivery products and I've done shows about this, numerous shows, but my opinion is that it's a really big space and that it's worth giving a lot of surface area too. Why use continuous delivery such a big space and why are there so many products for it?

[0:03:29.9] AA: I guess the first things is that it's a phrase that means a different thing to pretty much everybody you ask the question of. I suppose you could define it loosely as just a way to deploy your software more quickly or easily. I think because it means so much to so many different people, you get tons of different products that kind of implement the particular flavor of deployment process that the individual or company or whoever it is that's building service kind of thinks of as being the most optimum.

[0:04:08.7] JM: Given that subjectivity around what continues delivery is, give ma description for what a modern continuous integration or continuous delivery workflow looks like from your point of view.

[0:04:21.1] AA: Okay. Sure. I think it starts with having an application which has test suite that you trust so you know that you can run your tests and have some degree of certainty that the application is functional. Once you have that, I think you can start doing more interesting things like cook your deployment process into your source code control repository, so doing something like a poll request gets merged into the master branch in my rep and it automatically gets deployed to staging, for example.

[0:04:56.7] JM: You mentioned testing there. Does continuous delivery absolutely require that a team write a lot of tests?

[0:05:06.0] AA: I would not say it requires that a team write a lot of tests. I think, for me at least, the thing that it does require though is that you have some automated method of knowing that your app is in an okay state. I think, generally, that means tests. That could be that you have a bunch of unit level tests that you trust. It could be that you just have high-level black box smoke test kind of tests. I think, generally, yeah. It means tested in some form or another, just some way to give you the confidence that I made a change and it does what I think it does, but it also didn't break other stuff elsewhere.

[0:05:49.4] JM: Right. Yeah. There's a couple of apps that I'm working on right now and I am typically just deploying to a staging environment or some sort of testing environments, some sort of sandbox and doing manual testing and it's not perfect, it's not flashy. Eventually, as the product gets more evolved, I would love to have some more automation around it. In the meantime I'm actually kind of a proponent of manual testing in the early days of a product if you're just trying to move fast and make sure that the core functionality, because in the early days of a product, you got pretty narrow functionality. I don't think there's anything wrong with that and it allows product development to move more smoothly.

I actually don't think that manual testing process in the early days is orthogonal to a continuous delivery process. I think you can have a "continuous process" in the early days where you ship it. It builds in a new staging or testing environment of some kind and you do some manual testing and then if it works you promote it to prod.

[0:06:54.0] AA: Yeah. I would totally agree with that. That's kind of what I had in mind when I said it just has to be some way of confirming that your app is working. If it's a small enough app that you can do manual testing on it, then I think that's still kind of testing — You can still call that testing. You're exercising the application. It's just that you're doing it by hand. I think that's completely legitimate.

[0:07:19.7] JM: What are some of the big areas of confusion around continuous delivery?

[0:07:25.5] AA: I think that probably, for me at least, that comes back to different people's different interpretations of what it means. You could think of it in a very narrow sense of just, "I have some automated way to rebuild and redeploy my code." You could extend it slightly to be my tests run automatically when I push my code. Then if they pass, then a deployment happens and you can kind of keep spreading that out and out to — I don't know. My code builds successfully and before I can deploy, I have to make some API call to some third-party service. I think the challenges come when you layer more and more stuff into your pipeline. You have more space for things to go wrong as well. If you're making a third-party API call and that third-party API gets down, now you're not deploying your code, for example.

[0:08:22.0] JM: Now, we've given some coverage to what continuous delivery is and your perspective on it, and I'd like to get into the process of building a continuous delivery platform. You work at Heroku. You're on the developer experienced team and you were part of this process of architecting out exactly what the Heroku continuous delivery experience would look like. For a developer experience team, what is that role consist of?

[0:08:56.2] AA: Okay. Yeah. It's a funny one actually. It's an odd name for a team at Heroku because I think you could make the argument that the entire Heroku product is a developer experience product. Our team is not responsible for all of Heroku. We focus our efforts on deployment workflow, I think would be the simplest way to put it. Yeah, we've built up what we consider to be a really good way of managing your teams code delivery workflow. I could go through kind of the steps we went through if that's interesting to you, kind of the blocks we built it from and then that they come together to make a whole process that I think works really nicely.

[0:09:43.1] JM: Yeah, sure. I'd love to hear that.

[0:09:44.9] AA: Sure. Okay. The first feature that we built was automatic deployments link to GitHub. You make a push to a particular branch that you specify ahead of time and that triggers a new build and a new deployment From there, we lay it on the ability to have those deployments be blocked on CI. You make a push. It waits for CI to pass and then we deploy.

The next thing that we built was a feature that we call review apps, which is kind of ephemeral staging environment linked to each poll request which you open on GitHub. You open a new PR and we, Heroku, the platform, already has the knowledge of what your app requires to run, so we're able to spin up a brand new instance of your app, connect it to all the databases and wherever else you need that runs and attach it to your poll request. Now you've got a place that you can go and manually click around and it's good for checking that nothing broke. It's also really good for sharing a change with — I don't know, someone non-technical who doesn't want to pulling down a branch and viewing it locally on their machine.

From there, we built out kind of more formal dev staging production pipeline for deploying your code. Now, you can kind of imagine that you group apps together as development staging and production. You can have as many of any of those as you want. Typically, we would have many development apps or review apps as we call them, then one or two staging apps and then one production app, say.

What that lets you do then is you test your code in a review app. You're happy with it, so you merge it to master. We hook it together with our continuous deployment at that stage so you can have your master branch continuously deploying to your staging app. Then when you're happy with everything in staging, you can then take that built artifact that you've already built once for staging and deploy it straight to production. It means that you save the build time a second time around.

Then once we've got all that, that's a couple of years work for our team. Once we had all that, the kind of missing piece of the jigsaw was a CI service, a way to run your tests automatically. Up until then, we were hooking into GitHub's commit statuses API, which means that it works with any CI service. That's actually still how it works today, although we implement a CI service of our own as well. You have your kind of test apps which live just for the length of a test run. You have review apps which live for the length of poll request. You have a stage app which lives indefinitely, and you have your production app.

[0:12:45.8] JM: Okay. You mentioned those review apps, and that's literally — that's what I was talking about earlier and I use Heroku for a couple of projects of significant size. Again, that's how we do testing. We don't have automated tests yet. It's a really, really nice experience, just like you ship code and then it spins up a review app and then you can instantly go into — You can basically share your environment, the new environment of code that you've made with anybody on your team and it's a very pleasurable experience.

I think you said that this was a multi-year plan that you arched out at the beginning. How long did it take? Are you saying multi-gear in terms of the entire team's developers hours, or was this like in absolute terms. This took multiple years to write.

[0:13:44.5] AA: It took multiple years to write. The one thing I would — About that you just said that's not quite right is we didn't map this out ahead of time. The way that our team likes to work is we picked one feature that we could deliver in a fairly short amount of time, which was the automatic deployment piece at the beginning. Then we think to ourselves what feels like it's missing now. Where is the pain point? Then we build review apps. Then once we've got review apps, then you start saying, "Well, there's no formal definition of staging or production. Then we start to think about pipelines and it sort of rolls on one thing to the next to the next. There's no feeling of like we know what it's going to look like at the end so much as kind of a closer process, like a more of a feedback loop that helps us decide what to build next.

[SPONSOR MESSAGE]

[0:14:44.6] JM: At Software Engineering Daily, we need to keep our metrics reliable. If a botnet started listening to all of our episodes and we had nothing to stop it, our statistics would be corrupted. We would have no way to know whether a listen came from a bot, or from a real user. That's why we use Encapsula to stop attackers and improve performance.

When a listener makes a request to play an episode of Software Engineering Daily, Encapsula checks that request before it reaches our servers and filters bot traffic preventing it from ever reaching us. Botnets and DDoS are not just a threat to podcasts. They can impact your application too. Encapsula can protect your API servers and your microservices from responding to unwanted requests.

To try Encapsula for yourself, got to encapsula.com/sedaily and get a month of Encapsula for free. Encapsula's API gives you control over the security and performance of your application. Whether you have a complex microservices architecture, or a WordPress site, like Software Engineering Daily.

Encapsula has a global network of over 30 data centers that optimize routing and cacher content. The same network of data centers that is filtering your content for attackers is operating as a CDN and speeding up your application.

To try Encapsula today, go to encapsula.com/sedaily and check it out. Thanks again Encapsula.

[INTERVIEW CONTINUED]

[0:16:28.4] JM: In terms of that feedback loop, how are you interacting with people who are consuming those products, the developers who are using those products and who have complaints or who have suggestions? How do you work on that feedback loop and get that feedback turned back into the next iteration of the product you're working on?

[0:16:50.8] AA: Yeah. That's a really important process for us. The first thing which is probably quite specific to a company like Heroku, but we're really lucky that we build a product that is essentially aimed to people just like ourselves. We like to think of our releases and our feedback as kind of being tiered. The first thing we can do is build something and just use it internally in our team for a week or two and it would really rough at this stage and probably have a few bugs.

We're a user of these features so we provide our own feedback. Then once we're fairly happy with it, we can start sharing it more broadly amongst engineers in Heroku. There are also uses of the product and good people to provide feedback. Once we get beyond internal users, we tend to maintain our beta lists. A good example of this is when we built our CI service. Once we had something that we felt worked well enough and where we're comfortable sharing it with the public, we put our short Google form asking you to kind of, "Are you interested? Please sign up here."

From there we were able to start adding people into a beta program and the kind of lose rule of thumb with the beta program was we would add some small number of people, some number in the tens maybe and list their feedback, fire an email address that the whole team monitors and the whole team responds to. Once we feel like we've stopped hearing new bits of feedback, once we feel like we've kind of exhausted that group of people in terms of what they want from the thing, then we open it more broadly, maybe to hundreds.

The lose rule of thumb is ask for feedback from the smallest group that you can still get useful feedback from. When you stop getting useful feedback, then widen the group and widen the group. That's quite a good way to know we find anyway as well when you sort of have a large closed beta group that's not giving you anymore useful feedback or is kind of diminishing returns, then you're like, "Well, we must be ready to launch this publicly now because no one is asking for more stuff." Obviously, you kind of used your judgment in that as well, but that's the process that we like to follow.

[0:19:17.4] JM: Heroku is an opinionated developer platform. You and I were discussing that in our conversation before the show and I figure there must be a tradeoff between having an iteration cycle where you're soliciting feedback from the users and the maintenance of that opinionated platform. What are some ways in which Heroku is opinionated, and does that ever conflict with the ability to serve the short term requests of users?

[0:19:56.0] AA: Yeah. That's a good question. We trust our own judgment is the first thing to say on that. I would say before we solicit any feedback from anybody we, in our minds, have a good idea of what we think this thing might end up looking like. That's kind of our starting point, is we say, "We think this might probably end up looking a bit like this." Then we get feedback, that kind of lets you cause correct.

Now, a lot of the time you'll be able to do something where you hear a piece of feedback from one user where they ask for some very specific feature and then you hear feedback from some other user where they ask for another very specific feature. As you gather more and more of those bits of feedback but don't yet act on them, you kind of just sort of let them percolate and you start to see as more and more people ask for different things, you see a kind of general way that you could solve the problem for everybody. Once that starts kind of working its way out, that tends to be a really good way to figure out how your original vision for the product might have been slightly off. You can kind of work out that, "Oh, wait. There's this large chunk of functionality that we maybe didn't foresee," or something along those lines, and then you can use that to correct the direction that you're heading in.

[0:21:25.8] JM: One of the modes of opinion that I interpret from Heroku is to be almost a no-ops platform. Nobody can be no-ops, but I haven't had many issues on Heroku where I've had to do anything more complicated than restart my Dynos. There haven't been any severe bugs I've had to work through. Maybe I just got lucky. Is that one of the goals to sort of limit the amount of operational duties that a developer has to participate in for their application maintenance?

[0:22:06.7] AA: Yeah, for sure. I think there's kind of a maybe 80% case of applications which have very similar requirements and kind of operate in the same way. A big goal is to serve those 80% kind of very closely so that — It's like you say it, as close to being no operational burden on the team is possible. Then when you get customers who have much more specific requirements to try and build the Heroku platform in such a way that there's these scape patches or ways to dropdown a level and let them do the thing that they want to do. Now, that's not always the kind of — We don't always meet that immediately and it's sort of plays back into what I was saying about customer feedback. You get one customer who would really, really like to do this one thing with the kind of compute power on a Heroku Dyno, for example, that's not really made easy by Heroku.

Then you hear from another customer who wants to do something different but kind of related. Again, you can sort of aggregate that feedback and then find the right kind of hole to punch in the abstraction so that the people who need it can use it and the people who don't, don't have to care that it exists at all.

[0:23:38.1] JM: I was reading about the process of designing the continuous integration, continuous delivery workflow for Heroku. It sounded like you took a close look at the other tools that were on the market, particularly Jenkins. Jenkins has been around for a longtime. Arguably, I think at the first continuous integration tool ever. What did you learn from Jenkins?

[0:24:07.9] AA: Yeah. Certainly if it wasn't the first continuous integration tool ever, it's certainly the first on that I ever used. I think it will probably ring true for a lot of people. Jenkins is an interesting one, because it's very very broad and flexible. I think you could probably set a Jenkins server up to do most of — I said earlier, everybody has a different interpretation of continuous delivery. You could probably fulfill that with Jenkins 99% of the time, whatever your interpretation is, I would say. It has some — so and I count that as a positive. For the product that it is, that's awesome, because that's I think why people want to use Jenkins is so that they can tune their workflow with it, so that they can self-host as well.

I think what we learned from Jenkins is we learned — I think it's probably fair to say we learned what we liked and we learned what we didn't like and we kind of figured out what we thought fit with a Heroku-centric workflow. I think if you're going to be using Jenkins and you deploy your app to Heroku, you sort of cup down on the large functionality that it offers to a much smaller subset. It kind of helped us to kind of pair-down to the things that we felt would be important to customers who are already in the Heroku ecosystem.

[0:25:33.6] JM: Yeah. I think of the tradeoff being a little less flexibility and a little — There's a little less flexibility on Heroku, but a lot less configuration kind of burden. I've worked in Jenkins before and had to do a lot of kind of configuration — But I don't know. Maybe I'm not giving enough credit. I think it's like you said. It's just like very configurable, but especially if you're optimizing for those 80% of applications that people want to run on a low operations platform as a service, that's just a different set of requirements for a continuous integration than this blue ocean continuous integration tool, like Jenkins.

Let's talk a little bit about the process of developing something no top of a platform as a service and then we'll get into some of the engineering discussions. When you're building a feature on top of a platform as a service, so you already have a preexisting platform, what are the APIs that are available to you? Because there are people who have come before you and they've already build some stuff around — For example, the API for spinning up a Dyno. I'm sure on the backend, it's pretty well-defined for you. What are the things that you were able to leverage as you were arching out the continuous delivery construction and what were the things that you had to write from scratch?

[0:27:01.8] AA: Yeah. That's a good question too. This was a huge leg-up for our team where we've got four engineers on the team who built this CI service and the reasons we're able to do it with four engineers is because Heroku already exists and we're building on top of Heroku. Heroku is a platform which it already knows how to provision a container of a certain size. It already knows how to build your application in a way that it can run, it knows how to attach databases and any other kind of external resource that you might need. That's, I imagine, at a different CI company, that's going to be a huge portion of your work day-to-day and we as a team got that for free. That was an incredible place to start. That's kind of also why we figured it was a good product to build because it's getting to the point where it's snapping pieces together. That kind of makes it sound a little more trivial than it was, but that's the kind of idea.

Yeah. We can orchestrate service essentially. The other thing that Heroku, for the first however long of its life as a product was really focused on was this sort of apparently infinite amount of compute available. If you're willing to pay for more and more and more Dynos then they're available to you. Behind the scenes, that takes a lot of engineering work to make sure that we kind of maintain enough slack to be able to service what people are going to be asking, but at the level that we started from, we were able to more or less safely just assume there's an API and it gives us compute power, which is an amazing leg-out.

[0:28:56.1] JM: I've done a few shows with Heroku before. We talked about this issue of having enough resources in the pool that you can pull from and the regulation of that pool is pretty complicated, but once you get it, I don't know if you want to call it a solved problem. From your point of view, it's not really your problem. You're just calling some API.

[0:29:20.8] AA: Right. Yeah. The API has its own rate limiting in place which kind of lets you as the consumer of the API just kind of go nuts with it and trust that the service will tell you if you're overdoing it. Yeah, we sort of — As an engineering organization, we sort of have this line where you either work on top of the API or you work beneath the API. Yeah, if you're a team working on top, you get to just leverage all of the hard work of the team working on the orchestration and runtime layer which is — Yeah. It's amazing, to be honest.

[0:30:02.1] JM: I think this brings up an interesting point in terms of product development. I'm not sure what you'd be able to say about this, but I feel like Heroku wanted to. You could open up lower level APIs and make it easier for people to kind of like go under the hood if they wanted to and maybe have a lower opinion framework that would basically be accessing the same kind of resources that people would have on these other cloud providers, but maybe kind of an incremental opening up of lower level stuff.

I feel like that's not really where the priorities lie for Heroku. It said the priorities are more build up this higher level marketplace of services and plugins that are really easy to use. If you want to plug-in a database, it's like a few clicks and you don't have to do much setup.

I just interviewed somebody from Salesforce Einstein and Salesforce, I know, owns Heroku, but there's some easy kind of like few click integrations with Einstein, but it's kind of interesting because if you're building a platform as a service, you really could go in a lot of directions these days especially because the market is just growing so fast and there's customers for all kinds of cloud products. Could you comment on just the product philosophy behind Heroku and how when you have so many different directions you could go in, how do you select what are the right types of products to build?

[0:31:40.6] AA: I can talk about this from sort of my tight perspective and I can also give some thoughts on the company as a whole. The reason I say that is just that it is not all my responsibility and so I kind of don't know what everyone else is thinking. I think it kind of feels like the process of running a web application, or web server before Heroku came along was quite involved and in difficult and you have to know how to configure nginx or Apache or whatever and tons of stuff like that. Then Heroku comes along and kind of smoothed the layer over the top of that which makes it really simple and easy to do.

I think a good way of thinking about where product development will go for Heroku is, to kind of take another more recent example, we've got a Kafka as a service offering now. That kind of grew out of the same observation which was people are starting to build these kind of evented applications which require something like Kafka in the background, and Kafka is an amazing piece of software but it's hard to set up and run a cluster of Kafka instances.

From there — Well, people are using this thing. We believe it's really good software. There's no kind of one click way to get this running right now, and so that's a good place for Heroku to be able to come along and help out. I could see other stuff like that where it's kind of — Salesforce Einstein you mentioned is sort of in the same spirit, I think, where it kind of just comes along and takes the thing that people are doing and sort of smooth out and makes it easy.

[SPONSOR MESSAGE]

[0:33:43.9] JM: Your enterprise wants to adopt containers but you aren't sure how. CoreOS will help you along your journey to a containerized architecture. CoreOS are the container experts trusted by Salesforce, eBay, Ticketmaster and other world-class organizations. Go to softwareengineeringdaily.com/coreos to find our top five episodes about containers and Kubernetes as well as a whitepaper about migrating an enterprise to Kubernetes with CoreOS.

They've hosted, attended and spoken at many shows about containers and Kubernetes because those technologies are the future of the web. That's why CoreOS built Tectonic, an enterprise-ready Kubernetes platform. At softwareengineeringdaily.com/coreos you can learn about how containers can make your organization run more efficiently.

Thanks to CoreOS for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:34:52.2] JM: Right. Recommendation system. It's the kind of thing where, okay, a million people have written a recommendation system from scratch. Why is there not a cloud service that makes it easier to do that? I think that's kind of the MO Einstein.

Getting back to the continuous integration, continuous delivery process that people are taking advantage of on Heroku. Heroku Flow is this set of things that you've talked about, that you've built. There's the review apps, the continuous integration process. Just the other features that you built along the way. I have seen firsthand how this works for a small team. How does it scale up to larger teams where there's more services, there's more interactions, there's more opportunities for, I guess, failures. Has the response and the usage been different between small teams and larger teams?

[0:36:02.1] AA: I could talk about some of our internal usage of this stuff which for certain applications that we run, spreads out to a lot of engineers. For example, our dashboard app, dashboard.heroku.com, that's an ember application which probably has in the region of — I don't know, 30 or 40 people who contribute to it in a more or less frequent way. For an application like that, it scales fantastically. The only thing you tend to see is that there are lots more review apps open at a given time because there's a lot more poll request open at a given time. We still have one team, kind of a core group of engineers responsible for merging PRs and pushing that code out through the deployment pipeline. Yeah, it works fantastically for that.

When I think about some of our bigger customers who make heavy use of Heroku, it tends to get to be more of a question of engineering team structure than it does application structure. I think you often tend to see this thing where the architecture of the applications can mirror the structure of the teams which maintain them, and I think some people have opinions about whether that's good or bad. I'm not really passing judgment either way. That's what happens in Heroku as well to a degree.

What that tends to mean is that you can still sort of treat each application, each team that manages a bunch of applications as a smaller group. I feel like there's kind of an optimum number of engineers to be working on any bit of code application before you get diminishing returns and you might want to separate out their work anyways.

So I don't know, personally, I think I wouldn't want to build a team of more than kind of 7 or 8 people all contributing to the same codebase. The way we kind of cut the line is one pipeline which is the kind of thing that — It's like the overall container for your review apps and your staging environments and your CI apps. One pipeline is it's a one-to-one mapping with a GitHub repository. It's like one codebase, one application, one pipeline. When you get to that, you kind of — Naturally, your team is kind of split down into a group of — I don't know, like one to 10 people, say.

Then Heroku has organizations and teams and more structures on top of that that you can use to divide your larger group of engineers up in smaller groups of engineers with their own sets of permissions and what have you.

[0:38:57.4] JM: Right. The product development process — I guess we didn't go into it too much and I should return to that. Between the mapping out of what the Heroku CDCI stuff was going to look like. I read this blog post a day that the developer experience team spent just entirely in a café talking about all of the different things that you wanted to build, just getting the vision together. I'm wondering how the interaction between different teams at Heroku went. Was there interaction between user experience people and then backend engineering? I guess what was the sequence of communication workflows? Because I imagine engineering at a platform as a service or, basically, any cloud provider, engineering is probably — It's kind of different than typical SaaS company or just like a web app. It's kind of a different beast. I'm wondering about the product development process and how you interact with different teams.

[0:40:19.4] AA: Yeah. That's a good observation. Yeah, we have engineers who work in an extremely low level who know Linux back to front and know how to orchestrate containers, and then we have engineers who work at much much higher level who are awesome at creating frontend experiences. We also have every level along the spectrum in between and everyone is responsible for that bit and everyone has to keep their thing running well.

The way that our team is structure is it's fairly generalist. We have only one person who I'd say is just out and out a backend engineer. We have a couple of people who work across the stack, frontend to backend. We have a designer and frontend engineer as well.

For start, our team is setup that we can do everything we need to do within the team to a large degree. Now, for something like CI, there's also a lot of help that we need from other teams. One big example of that would be we need to know how to build your app and run your tests on a language-by-language basis. Personally, I'm very comfortable with Ruby. I know how to do that stuff in Ruby, but I know next to nothing about Python, for example. Ruby and Python are both language that Heroku supports, and so they're both languages that CI needs to support.

Luckily, we have a languages team where we have someone who's awesome at Python. We have someone who knows the JVM back to front and someone who knows Go really well. We needed to get the help of each language maintainer to say, "First of all, this is how you build the thing." Also, we want an experience where if you're following the idioms of your language, then maybe you don't do any setup at all.

For example, with Ruby, it's probably a good starting point to say, "Run, bundle, install with the test gem group, and then run rake test." For most Ruby apps, that will execute your tests. We can kind of codify that into CI and we can do the same for different languages as well.

To get back to your question about kind of how do we manage that, the first thing that we did once our team had kind of spent that day getting excited about all the possibilities and like you kind of go nuts and start discussing, "Oh, wouldn't it be cool if this and that, and whatever," and then you realize you've described four years' worth of work for your small team. Then you sort of stop pairing back to, "Well, how could we build a proof of concept in —" I think we gave ourselves two weeks.

What we did there was said, "This is a thing that's never going to ship to our customers. As we're building it, it's a throw away thing but it's supposed to represent — It should work. You should be able to run tests on it. You should be able to see the results in the Heroku dashboard review app as that happens was how we did it. Then we got to show it to the company and use that as the thing to kind of get buy-in or get more people excited about what we're planning to do. It's easy to just describe what you're going to do, but I think it has a very low impact.

If you can show something working, then people start kind of getting excited and thinking — The languages team who I spoke around already, they saw that demo and then each individual language maintenance is going, "Oh! Wow! With language, if I could just do this and that, then the experience will be great for whatever, Python users." The other thing it did was teams like the runtime team who manage Dyno allocation or the department of data who manage Postgres and Redis and Kafka and so on, it got them thinking about what impact we might have on them operationally.

That kind of opens up these communication channels where a team can say, “Hey, if you do this in the way that you implemented it in your demo and it gets really popular, this will cause us problems in this way and that way.” Then we can design our systems that we mitigate those problems.

Yeah, the short answer to the question is we built a proof of concept and Heroku have a monthly demo day where every team shows off what they’ve been working on, so we demoed it there. Everybody in the company saw it and was impressed by the vision and they started thinking about the details and how they could chip in or how it might cause problems, and there’s a list of things that we can look to address in a real design.

[0:45:12.7] JM: When you’re building a platform as a service that’s also a marketplace of products that people plug into, there is the question of integrations. You have to provide the right API surface for other things to integrate with, but you need to keep it narrow enough so that you don’t have people breaking the workflow offering you have. Can you talk about building integrations for Heroku CI, Heroku Flow?

[0:45:45.8] AA: Yes. The first thing is that just like with the question of computation or compute power allocation where the problem was mostly solved already first by another team in Heroku, we’ve already got the Heroku add-ons marketplace and that’s already full of providers who offer — We have a few of our own, like Postgres and Redis, but, say — I don’t know. You want MySQL, we have add-on providers who provide MySQL or MongoDB or whatever, something that we don’t offer first party.

We were able to build on top of that and say, “Your test require Mongo, for example,” we, as Heroku, already know how to provision a MongoDB instance for your applications. We’ll take that same process and do it for CI as well.

An issue that this can cause — And we heard about it first from our own internal add-on, the people who maintain our own internal add-ons is if you’re provisioning and de-provisioning add-ons very frequently, that's a different usage pattern to the kind of thing that Heroku is typically expecting which is you might provision Postgres once for your application and then expect it to live indefinitely. With CI, you're going to provision Postgres and then kill it again in two minutes and then provision another one and kill it and again and again and again.

We heard pretty early from our own internal add-on teams and then also from add-on partners, external add-ons, that they would know, if this is for CI, I'm going to provision it one way. If it's for production use, I'm going to provision it in another way. That was something that we were able to work with our ecosystem team internally to add just to kind of small change the API which lets add-on providers know why they're being provisioned.

Now, if they need to, they can say, "You get one type of thing for production. You get something else if it's a review app and you could potentially get something else again if it's a CI app. I don't mean something else as in a completely different product, but maybe it's from a slack pool instead of being spun up fresh. That can be helpful as well, because provisioning of something that you're spinning up fresh can take minutes, which is no good if you're waiting for your 30-second suite to run. It's nice in that way as well and that people can provide a different experience if they need to.

[0:48:32.7] JM: As you've been building this, I imagine that the goal post has been continually moving down the field and you're thinking of more things to build on top of it. What are the priorities for what features you want to continue to build in the future?

[0:48:49.2] AA: I think where we're at at the moment with CI is we launched it publicly a few months back and you can kind of think of that as an extension of gradual widening of the group of users. Now it's available to all people and that's a whole load more feedback that you can solicit.

We're still listening to feedback. We're still receiving feature requests and we're sort of collating that together and kind of using that to inform where we think it should go next. In terms of concrete plans, I think we're still kind of processing that and trying to figure out what we think is missing. There are some things which we explicitly exclude as goals for Heroku CI. An example of that would be we don't particularly think we're a good place to test your library code. If you maintain an open source node package, for example, some other CI provider might be a better fit.

We think that the real sweet spot for Heroku CI is you have an application which itself runs on Heroku. Now, Heroku CI sounds like a really good idea because you basically have the closest production to test environment parity that you could have with running on Heroku just like your application is.

[0:50:26.8] JM: Fascinating. To close off, another thing you and I were discussing in our conversation before the show was the way that — At least the way that I use Heroku is just kind of for a singular application. Right now I've only deployed monoliths to it, but I know that people do deploy services, microservices to Heroku and there's kind of a shift in development going on towards more microservices, towards more using APIs as a service and I'm wondering how you think that — I guess you would call it the serverless stuff and I'm wondering how you frame this in terms of wider scope or product development for the future on Heroku.

[0:51:21.3] AA: Okay. Yeah. I think the starting point for the way we think about that is that's the way that our engineering team builds applications. Heroku is made up of probably hundreds of individual services which do their jobs and communicate with each other. As much as possible, those applications are deployed on Heroku. There's the odd exception where you're doing something really low level with orchestrating things that can't run on Heroku, but most of our services run on Heroku as well.

We're writing applications in that fashion that you described and we're doing it on Heroku. We're filling a whole set of pain points around that stuff. I would say what we see at the moment is kind of small chunks of that problem being taken and addressed, and I think it's probably — In order to say — Heroku for a monolithic application is super easy, Heroku for a set of microservices. You're starting to get towards the boundaries — You sort of have to do a bit more of the leg work yourself.

At the moment, we're sort of taking small chunks of that pain and we're able to kind of make this thing slightly easier, make that thing slightly easier. An example would be DNS discovery. A way of letting one applications Dynos know that another applications Dynos live such and such DNS address — I'm sorry. IP address. What we don't have, and this is kind of a thing that I have been turning over in my head and don't really have a great answer to at the moment and I can imagine it being kind of a slow burn of is there kind of a unification of those ideas. A way to say, "My application is composed of — My customer facing application is composed of all these services," and be able to describe how they all interact together and have them all be able to find each other and tools for working that way. Yeah, it's a big problem to solve and I think lots of people in lots of different companies are thinking about it in lots of different ways, not unlike continuous delivery and deployment actually. A very early stage of kind of wondering what solution to that huge set of requirements might look like.

[0:54:05.8] JM: One thing is for sure, is that it's a much more enjoyable time to be a developer than the days of, like you said, configuring nginx and whatever else. I never had to learn how to do that stuff thankfully. Honestly, I use Heroku all the time. I really love it. It solves so many of my problems and just continued success to your team. I will continue to use it, I'm sure.

[0:54:37.9] AA: Awesome. Like I say, we're in the same boat inside Heroku. We use it all the time for our applications. Yeah, when there's pain points, we feel them too.

[0:54:48.9] JM: Okay, Andy. It's great talking to you and thanks again for coming on. Thank you to Heroku for being a sponsor of the show. I couldn't be happier to have them as a sponsor because I use it for, like I said, my own software development. Thanks again. Good talking to you.

[0:55:05.7] AA: Cool. Thank you.

[SPONSOR MESSAGE]

[0:55:09.7] JM: VividCortex is the best way to improve your database performance, efficiency, and uptime. It's a cloud-hosted monitoring platform that eliminates your most critical visibility gap, providing insights at 1-second granularity into production database workload and query performance. It measures the execution and resource consumption of every statement and transaction, so you can proactively fix future database issues before they impact customers.

To learn more, visitvividcortex.com/sedaily and find out why companies like Github, DigitalOcean, and Yelp all use VividCortex to see deeper into their database performance. Learn more atvividcortex.com/sedaily, and get started today with VividCortex.

[END]