EPISODE 578

[INTRODUCTION]

[0:00:00.3] JM: Super pedestrian is a robotic bicycle wheel that learns how you pedal and personalizes your bicycle ride. The engineering challenges of super pedestrian are at the intersection of robotics, software and real-time analytics. The first half of today's show is about super pedestrian. Goss Nuzzo-Jones and his colleague, Matt Cole, are engineers at Superpedestrian. The slides for their presentation are also in the show notes, if you don't catch any of the things that they say in the podcast episode.

The second half of today's show is about HubSpot, a massive business with lots of infrastructure challenges. Thomas Petr explained how HubSpot's engineering has matured and some of the scaling problems that they have tackled. Last month we had three Software Engineering Daily meet ups in New York, Boston and Los Angeles and at each of these meet ups, listeners from the SE Daily community got to meet each other and talk about software, what they're building and what they are excited about. I was happy to be in attendance at each of these, and I'm posting the talks that were given by our presenters. The audio quality is not perfect on these, but there are no ads and we're airing them on the weekend.

Thanks to HubSpot for hosting this meet up. They have beautiful offices, and if you're looking for a job or if you want to host a technology meet up in the Boston area, I strongly recommend checking out HubSpot. We would love to have you as part of our community. We have meet ups occasionally and you can be notified of these by signing up for our newsletter, and you can also come to softwaredaily.com and get involved with the discussions of episodes and software projects and you can check out our open source projects, the mobile apps and our website.

[INTERVIEW]

[0:01:52.4] JM: Special preparatory thanks to Zev, who has been an AV expert here. He's a software engineer by trade, but AV expert is moonlighting. I think this is everybody. So, I'm Jeff, and I think most people here have heard of the show or listened to it. I run Software Engineering

Daily, and I'm really happy that you all showed up for the meet up, because doing the podcast is kind of a solitary experience and it's really nice to get out and meet people in person.

Today, we've got a couple of awesome talks, and they're going to talk for about 30 minutes each, and after each talk we'll do 5, 10 minutes of questions, and this is Superpedestrian. This is Goss and Matt, correct? I will let them explain to you what Superpedestrian does and why it's awesome. So with that, Superpedestrian.

[0:02:42.3] MC: Thanks you, Jeff. Thanks for having us. We are here today to talk about working on the Copenhagen wheel at Superpedesetrian. The Copenhagen wheel, if you're not familiar, is an electric pedal assists bike wheel. So you'd replace your rear wheel with this Copenhagen wheel and when you pedal, it pushes. So, thank you, Jeff. This is Goss, Goss Nuzzo-Jones. He's the head of software at Superpedestrian. My name is Matt Cole, I'm the head of embedded software at Superpedestrian, and we're here to talk about the software in and behind the Copenhagen wheel.

Short intro on Superpedestrian. It was spun out at the MIT Sensible Lab in 2012. Our founder, Assaf Biderman was contracted by the City of Copenhagen to create a product that would enable more of the city to take longer commutes by bicycle. We're around 60 people and we're headquartered in Cambridgeport. So, first, we need to talk, I think, about why cities are important to the development of the Copenhagen wheel. Cities represent about 2% of the earth's habitable landmass, but they have two thirds of the energy consumption. So 66 or so percent, and that's just growing. They also produce more than 75% of the world's CO2 and CO2 equivalent emissions and produce 80% of the world's GDP.

Cities have been long known to be societies predominant engine of innovation, wealth creation, but they're also the main source of crime, pollution, disease, metrics from pattern production and personal income to electrical cable length are known to the square, to grow by the power law function with population size. So as we grow and grow, those shoot up. Cities also present massive economies of scale. So, with less infrastructure required per capita, if your resources required per person, so all that scales with population. In short, if we're able to address some of the key challenges to the city space, we can make a major impact on the entire planet.

Over time, we know that cities are getting more populated. In 2010, 50% of the world's population was in an urban area. It's projected to be 60% in 2030. More than two-thirds in 2050. Superpedestrian was created to help answer the questions surrounding how to move people through a dense urban area. With the growth in urban population, the demand for urban transport is increasing. By 2050 we're expected to have more than 2-1/2 times of the demand from today for urban passenger miles. Think about what would happen if LA had 2-1/2 times the traffic, or Mexico City, or Lagos, and any of the giant cities around the world, multiplied by that scale and you'll see what we're facing in 30 short years.

Here in the United States, the use of public transportation is on the rise. We've seen a sharp, steady increase over the last 15 or so years. Highway vehicle miles, however, are on a decline, but in Europe, per capita, all miles driven regardless of mode, have been in a steady decline since the early 90s. So how do we solve some of these problems? We think bikes are a wonderful solution. Commuting via bicycle is on a huge increase. You can see it's kind of an eye chart, but you can see that year-over-year we're seeing rapid increases on bike miles for commuters. It's becoming a great way to get from point to point in the city with great efficiency. It's faster. It costs less. You do better. You're more healthy, and biking is no longer limited to just recreational activities. We also have bike sharing programs proliferating around the United States, and that's adding to these miles with great scale.

[0:06:34.9] GNJ: However, we've run into a couple of problems, and by we I mean the greater population of earth, which is during the construction of these cities around cars, they grew a lot. So now you're city cores are very large, and this became evident in the city of Copenhagen and we got to take a look at some of the data that that city accumulated over the last 30, 40 years. Copenhagen was a very car-centric city in the 70s, and through a concerted effort of policy and cultural change, it's now — And more bikes than people, 55% of trips in the city center are by bike. But what they discovered is there is basically an upper limit to the amount of biking that occurs in the city.

So that peaked in the — I think, the early 90s, and they couldn't seem to — Any sort of public policy didn't seem to make a dent in it. What they discovered was there's a real barrier around distance. People don't want to spend too much time bike. The sweet spot is about 15 km of total commute time. Unsurprisingly, if there are hills in your commute, you don't want to do it.

So we thought of this is we should flatten those hills, we should make distance easier for you to overcome on a bike, an electrical bicycle. This is not a new idea. The first one was invented in the 1880s. It's been around in various incarnations. It's already in existing market. It's growing particularly in Europe and starting to in the U.S. But when we looked at this market, we saw a lot of opportunity, because as you can see they're not very attractive and they're very expensive. An entry-level electric bike, like most of those you see here is about $3,000, and they scale all the way up to about 10,000.

So what we decided is we'll make something that can retrofit your existing bike, the Copenhagen wheel. So this is a quick look at what they look like. Then as Matt said, they replace the rear wheel of your bike and they turn your bike into an electric bike. As you pedal, you get an assist from the motor, and that can amplify the torque that you're putting into the wheel by up to 10 times. They're quite a lot of fun to ride. That's an exploded view of all the different components in there.

Now we're going to switch gears and talk a little bit about the software involved in developing the product and keeping it running.

[0:09:07.3] MC: This is a software engineering podcast after all. So at Superpedestrian, there are four main areas of software that we develop. We have the embedded software, that's all the software in the wheel itself. We have the mobile apps that interface the human to the wheel. We've got a data backend that connects the mobile apps to our databases, and we have a whole suite of business software.

My job as the lead embedded engineer, I manage all of the software in and around the wheel itself. That includes the Bluetooth connection, integrated sensors. We have a torque sensor, voltage and current obviously, temperature sensors, motor and cassette, position sensors. The cassette being the gears on the outside of the rear wheel and an accelerometer, a 6-degree accelerometer. We also control the wheel output. We have a fairly intense algorithms that handle are output control. We can assist up to around greater than 450 watts. We can also resist for exercise mode. If you backpedal, you get e-braking, and in both of those modes you get energy recovery. So you can charge your battery by braking down a large hill. We also manage the firmware update process. We just went through a release cycle. So lots of people with Copenhagen wheels are now challenging that code, and we have embedded fault monitoring and remote diagnostics all enabled through the embedded software.

I'd like to take a brief look at the motor control. Not too many people here might be embedded engineers, but what we're doing is we take all the various disparate sensor inputs. We have a number of control models that run in series. So we apply all those inputs through the models. We limit based on physical constraints, temperature limits, speed limits, electrical limits, and then we feed out through the motor, the output torque. Then that feeds back as an input into the control loop, rinse, repeat. Can provide greater than 450 watts of peak power. On my own wheel I have seen a touch better than 500, occasionally. Backpedaling will get you electric braking, and we can harvest energy, as I said. We also have intelligent safety systems that monitor the state of the wheel, the health of the components while you're riding to keep the rider in a healthy and safe state.

To talk about those embedded diagnostics a little bit, we're constantly logging all the data we receive from the network of sensors that we have. When we see a fault trip, that is considered a safety hazard, the wheel turns off. You've got to have a human intervention. That's actually specified by the e-use regulations that you must have a human intervention when there is a safety related fault. We also log soft faults. We log hazards. We can see all of that through a web portal.

The wheel diagnostic data is communicated through the Bluetooth connection to our mobile apps. The mobile apps send that to our database, and it's stored for a period of time so that we can look and do field studies. We can do population studies and find out did our last firmware update brake a bunch of whales? Did we miss a constraint somewhere? Also, enable remote health checks and force a wheel to return its health and safety data.

[0:12:27.2] GNJ: So from that Bluetooth interface that Matt talked to or talked about, we built our mobile apps, and the mobile apps have a number of key functions. The first being authentication and identification. As you walk up to your Copenhagen wheel, it recognizes you and it won't turn on for anyone else except you or other people that you authorize. When you are riding, you can see on the right there's a little screen shot about different modes that you can put the wheel in and that changes how the wheel behaves and accelerates. So turbo is the most fun. It has the highest speed limit and goes fastest, accelerates the most. But there are other modes with more range or exercise challenges.

In addition to all that sensor data that we're gathering from the wheel, we capture sensor data from the phone. So in particular, we do a GPS trace on the phone while you ride. We match that up with the sensor data from the wheel and provide it back to the user in the form of a trip. Not unlike other GPS tracking apps out there, but with a little bit richer data.

We capture that data and manage that function, those functions in the app through our backhanded API. So it does some pretty basic transactional stuff, like log in and sign up. It also enables different levels of access to the wheels. So you as an owner of a wheel can do — Privileged to do some certain things such as update the firmware or adjust the speed limit, whereas if you loan it to your friend for the weekend, they can only ride it. There's other levels of access, such as maintenance or engineering.

We also enable those over the air updates to the wheel, managing different components on the server to basically form a compatibility matrix. With a long-running hardware product, certain components may change over time and we have to keep track of that to make sure we're tripping the right firmware down to those wheels.

We're storing very dense ride data when people are off on a trip, and that takes the form of those GPS traces as well as all of that sensor data. So a regular commuter who's going between 3 and 5 miles to work may generate 15 to 20 megabytes of uncompressed data a day. It's quite a bit. We also capture what we call event data, which is what Matt alluded to with the idea of hazards or faults. So that could be something as simple as the wheel is running a little hot. We're going to reduce assist to prevent damage to any component, and the user doesn't see that, but we capture it, and that could be a piece of information later if the wheel fails or it may just be informational, it's a hot day. So that ties in to some of our debug tools going forward.

This is a quick overview of the architecture that we have set up on the wheel API side. Because of the density of data that we're talking about, we have sort of a different scaling problem than it is perhaps typical. Our number of users is limited by the number of wheels that we sell and build. So we don't have an exponential user growth, but each user is generating a ton of data and it's expensive to process. We take advantage of the fact that we have these very powerful clients in the phones to summarize that data.

So the client summarize the data and provided to us for quick access in terms of a PostgreS database and then we store the full rich data as documents in S3 and process that asynchronously. Most notably, we take a lot of data and ingest a window of it and do an Elasticsearch cluster that is used by the engineering team.

Another way to look at some of these data, mostly looking at that summary data I alluded to is what we call our when support tool, and this is s designed for our support teams who are interfacing directly with the customer who might be at it. So they can find a wheel, see if it's been experiencing faults very quickly perhaps download some additional data, suggests some solutions or pass it on to an engineer to investigate further.

With all these data, we have some interesting ideas for what we could do going forward. One of the big things is detecting road conditions. Because we have a fairly precise accelerometer, we think that we can do pothole detection as well as rough road or potentially than slippery road detection. That kind of data exists for cars in a lot of different situations, but it's a very different ballgame if you're a cyclist, if a pothole means potentially an injury for you, whereas a car, it's an inconvenience.

The other thing that we're considering is bike service reminders. We think that we can detect certainly sort of recurring service, but also things like when your tire is getting a little low. We're starting to feed some of these data into health and fitness features. Obviously, the wheel is designed to make your commute easier, but on the off chance that you would like to use it to train, it can resist you at a variety of degrees of difficulty and you could use it as a trainer. Then sort of broadly rider behavior and training overtime, we think this could be really valuable to municipalities in terms of bike infrastructure planning.

Sort of in the more mundane from a purely software perspective, but obviously very important to us, the software team maintains our e-commerce and [inaudible 0:18:00.4] infrastructure.

Copenhagen wheels are sold configured to order. So you can customize one to your bike or you can buy a bike with it if you don't have one or you don't have one that you like.

We make the wheels here in Massachusetts and the software team also supports our manufacturing logistics infrastructure. We also support our marketing infrastructure, and that includes direct sales online, a whole network of bike shops around the U.S and in Europe and a variety of popup events both here in New England and elsewhere.

Currently, our software and embedded team is nine people. Both Matt and I find it a very fascinating place to work. We get a tremendous intersection of disciplines both from an engineering perspective as well as a business model and marketing challenge. The Copenhagen wheel is really fun. For those of you who live in the area, you should come down to Cambridgeport and try one. They're a great deal of fun to ride. Thank you all for being such a good audience. Do you have any questions?

[0:19:03.9] JM: Thanks, guys. Any questions? I'm just going to repeat the question real quick so it's on the recording. Differences in weather riding during the winter, does it get too brutal for the Superpedestrian, and what type of data are you seeing?

[0:19:18.8] GNJ: Generally, the performance of the wheel is similar in the winter versus the summer actually .Probably in the winter, you'll going to get — You're not going to overheat, which is nice, which is more often the limiting factor of motor performance. But it's great in the wind. Be careful on the snow, just in general. There are lower temperature limits for operation and charging. You're much more likely to hit the charging one. So if you store it outside, that can be a problem. But for operation, I don't know what the lower limit is, like negative 20 or something like that.

[0:19:53.5] MC: For operation, it's extremely low. The batteries are kind of a different animal. Lithium-ion batteries don't do very well in the cold. So we stopped charging at 0C. So in terms energy recovery, there's none of that, because the batteries themselves refused to take it below zero. They're happy to supply energy, but they won't charge.

We do see our ridership decline in the winter in Boston. I broke out my bike for the first time in about a month. I think we had a warm week in February, but I'm riding this week. I think, overall at Superpedestrian, I might be an outlier. I see a lot of people riding, real road warriors, but I think our ridership in the north, anywhere where it gets cold, you're limited by snow and ice on the ground, and the comfort of the biker.

[0:20:44.6] Q: Do you pick that up? [inaudible 0:20:46.4].

[0:20:49.1] MC: I haven't spent a lot of time looking. We would absolutely see that because we know your position and we know the wheel's rate of turn. So if I see that you're not moving, but the wheel is spinning, I know that you have hit some slippery ground.

[0:21:03.9] GNJ: But we don't do anything specific with it now. Yeah.

[0:21:06.5] Q: Just following with the analytics, do you see any differences like per regional, like West Coast, East Coast wear and tear on the bike, analytics of how people ride them, like so forth?

[0:21:18.6] GNJ: I don't think nothing particularly unexpected. Our West Coast are generally taking longer rides, but that's because the cities are bigger and more sprawling.

[0:21:29.1] Q: Discover a little bit on the embedded side. Do you have an RTOS? What are you using for communication? Is it FPGA-based?

[0:21:35.9] MC: We buy off-the-shelf parts. We have a TI motor control part that does most of our main — It's our main MCU on the control board. We run no RTOS. We're bare metal, so we program our own tasks and interrupts.

[0:21:50.7] Q: This might be a dumb question, but do you guys have an API or let people explore all your data so you can play around with it?

[0:21:58.3] GNJ: Not yet, but we have plans to.

[0:22:00.3] Q: So you say you are recording the software faults. What kind of faults are you recording?

[0:22:05.0] MC: A wide range of things. If you critically overheat your wheel, that's a fault. So there's a band where we start to limit the output, because you're getting too hot, but if it's a very hot day and you're using it all, there's nothing that we can do to prevent the thermal rise there. So limiting only has a certain effect. If you'll hit a top point, then that will be a fault where we won't operate anymore. You have to let the wheel cool down. We also have electrical faults. There are sensor failure faults where we're constantly looking for all those things.

[0:22:36.7] Q: Does that include the bike or just the wheel? I mean, the —

[0:22:41.2] MC: Our sensors are wholly limited to the wheel itself.

[0:22:45.5] Q: Is there antitheft system? Like if somebody, I guess, steal my bike and —

[0:22:51.3] GNJ: There is deaf deterrence. The wheel is a brick to anyone who doesn't have earphone or [inaudible 0:22:57.2].

[0:22:57.5] MC: It's about 20 pounds of rotating mass. Without the assist, it's a challenge to start moving.

[0:23:07.1] Q: Did you ever have any issues about security around the antitheft components too? Just people spoofing the user's other issues like that?

[0:23:15.0] GNJ: We haven't run into that, but we are currently in our product development process, and security features are something that we're actively considering for the next revision.

[0:23:24.9] Q: Hi. To the user, do you provide any data, like how many watt of energy you put in yesterday versus toady, so therefore how much fitter they're getting.

[0:23:34.7] GNJ: Yeah, we do some basic summary in the app and it's something that we're looking to continue to build out.

[0:23:39.7] JM: On the firmware side, do you do anything clever with charging the cells to maintain the length or the life of the battery?

[0:23:50.4] MC: Our BMS has self-management integrated into it. BMS, battery management system, if not everyone is familiar. It does wear leveling, obviously. That's a bar of entry for a large lithium-ion pack. We do sell balancing and state of charge estimation, and we do have a metric for battery health. That takes into account the number of full charge cycles that it's seen, the age, various temperature excursions. All these things affect the long-term health of lithium-ion batteries.

[0:24:23.6] Q: Do you also have some advisory services?

[0:24:27.2] GNJ: I'm not sure what you mean.

[0:24:27.9] Q: For example, let's say I own one and it's a hot day. It would not be advisable for me to take a long trip, right?

[0:24:37.7] GNJ: Yeah. So the app will let you know if something has gone wrong or you fix the certain class of hazards, like a temperature warning. You can still use it as bike wheel. So even if it reduces the assist, once you authenticated the wheel, it's still like a wheel. So even though you may not get the full effect, if it's a very hot day or it's been sitting out in the sun for a long time.

[0:25:04.3] JM: I'm going to ask one last questions, then we can take a break. Do you have plans for lateral product expansion or some kind of big vision for where — What are the opportunities for the company to expand it to?

[0:25:16.5] GNJ: Yeah. Like I mentioned, we are working on that on the next revision of the product. The sort of — From our perspective, one of the big drivers of that is disc brake support, which currently the Copenhagen wheel is limited to rim brakes. A lot of newer consumer bikes are disc brake only. That's the big physical.

[0:25:37.5] MC: I think if you ask the same question to our CEO, he'd have 30 minutes for you. I don't we feel comfortable spending those 30 minutes.

[0:25:45.7] JM: Fair enough.

[0:25:47.1] GNJ: As I mentioned, we're considering a lot of other potential feature additions and capability, including antitheft.

[0:25:54.3] JM: Awesome. Goss and Matt, thank you so much for showing up and giving the presentation.

[0:25:58.3] GNJ: Thank you.

[0:25:58.9] MC: Thank you.

[0:26:03.0] JM: Okay. We are going to get started in a moment with the second talk from Thomas Petr from HubSpot who works on infrastructure. Before we get there, I do want to thank HubSpot for sponsoring the venue of this meet up, and I know HubSpot is hiring. So anybody in the audience who's interested in looking for another job, whether you're in the physical meet up audience or on the podcast, you should definitely check out HubSpot. Great offices. I can vouch for that. The food is sponsored by Polsinelli, who is a law firm. If you have any legal questions around your technology, particularly in the zone of patents, I think, Polsinelli is a great place to go for your legal assistance.

With that, Thomas Petr is going to talk about infrastructure at HubSpot. Thomas, take it away.

[0:26:54.6] TP: Cool. I wanted to give you a real podcast experience. So I didn't make any slides. One laugh. That's good. Thank you, Zev. All right.

My name is Tom Petr, I'm an engineering lead here at HubSpot. Thank you all for coming. Thanks for having me. For those of you who don't know, HubSpot does more than host meet ups and keep the large orange sign industry afloat for sales and marketing platform to help businesses grow. We do that through this thing called inbound marketing. So, instead of you, you're trying to sell something, you're blasting people with like, "Hey! Buy my product. Buy my product. Why haven't you bought my product?" You generate interesting content that make people want out more about your product, and then they buy it.

We provide a suite of tools that make it easier. So we do website hosting. We do email. We do sophisticated analytics, social media. There's like 18 other things that I'm forgetting because I'm on stage right now. But it's really cool. The way we're able to build all this is not through heavy process. It's not through lots of deadlines. We focus on small autonomous teams. That means teams of two or three engineers, a product manager and a designer are all working together, owning a very specific slice of the product. So that means that we have a team, a team for email, a team for social media, blah-blah-blah.

The focus is really give that team a mountain to climb. Give that team the tools to climb the mountain, and the most important thing, make them excited to get to the top of the mountain. So you're probably thinking, "Okay. So you're telling me that we have a group of five people competing against like Hootsuite? We have a group of five people competing against MailChimp? That is insane." And it is kind of insane.

The secret weapon is our large infrastructure team. We have a team of 45, 50 people working on all the tools and services needed to deploy code quickly, safely, and it really lets people on the product team specialize and do what they do best, which is cool.

Let's talk about infrastructure. So we've been an AWS customer for 10 years now. Right now, we have 2,500 EC2 instances. We store many petabytes of data in S3. About a petabyte of data goes through our load balancers every month, and AWS has been a huge driver of growth for us. Being able to click a button to scale our infrastructure allows us to move up the stack and solve higher-level problems, build higher level automation. Just tackle other things.

Some of the things that we've done to make that easier is we built a tool called Rainmaker, which wraps around the AWS console. Sands down some rough edges. It fills in some potholes and it makes it really easy for our developers to provision a new machine or create an S3 bucket if necessary. We also run the HubSpot product on Apache Mesos with an open source scheduler that we created called Singularity, and the thing that's nice about that is developers go from a world where it's like, "Oh! I got to deploy my code to an actual server. That server might go down. That server might break," to "I just got to deploy my code and Singularity will take care of it."

Singularity is different from other Mesos schedulers, and that it doesn't try to be a generic compete utility, like marathon or things like that. It's really designed with developers in mind. So there's very specific workloads, like you can deploy a web service, you can deploy a worker. You can deploy a cron job, and it has a really rich UI so that you can see how your stuff is doing. You can tail logs. You can check resource usage, etc. It's really nice.

In that vein, people are deploying code a lot. That means we're generating a lot of builds. We used to use Jenkins for a really long time. We outgrew Jenkins just because of the number of repositories that we have in GitHub and the number of branches that we build, blah-blah-blah. So we ended up building our own CI tool called Blazar, and it's really neat. GitHub web hooks come in and it spits out Singularity tasks that handle all of our builds.

Finally, to tie everything together, we have this thing called Orion, which handles all of our deployment orchestration. It probably has a really nice rich web UI so that if a developer wants to deploy, they'll just go to Orion, type in the name or the thing they want, click deploy and they're done. It's very slick. It's very cool.

So talking a lot about — Deploying a lot, in case you're wondering, we do microservices. We kind of take it to the extreme. HubSpot has a lot of tools that make up HubSpot. HubSpot has a lot of teams. We have an absurd number of web services. What we call the HubSpot product is actually over 600 different individually deployable web services. In 2017 alone, our developers kicked off 2.5 million builds and deployed code 900,000 times, which is just like insane. It's insane because I've been in HubSpot for six years now. I really started working on the infrastructure stuff in earnest like four or five years ago, and when I started, it was literally just me working on it. It's pretty crazy to see where it's gone.

Going back to the deployment stuff, all these things really reflect our overall focus of not trying to build some flawless monolith with code that never fails. We really focus on incremental progress and minimizing meantime to recovery, because you're never going to always push perfect code every time.

Overall, we've been pretty happy with Amazon. Like I said, we've been customers for 10 years. It's just like the thing that we've used. Like any relationship, over time you find things that you don't really like about the other person. With Amazon, the biggest thing is just machines fail. There's that quote from the Amazon CTO, "Things fail all the time," and they're just kind of expected to deal with it yourself, and that was a pain I get, that we had to build more automation. We had to handle all these edge cases that in a perfect world we wouldn't have to worry about.

What I've described has been HubSpot for the past three years, but in 2017, everything changed. We had to open an EU data center, and that was really exciting, but also really scary, because we only ever ran in one region in AWS. So having to run multiple things in different places, it was just a new challenge. The cool thing was they gave us the opportunity to survey the landscape. A lot of things have changed in the 10 years since we originally chose to go with AWS. Particularly, we really admire Google. We use a ton of their open-source libraries. We're in the middle of migrating our data services to Kubernetes, which originated from them, and we are super interested to see what Google cloud platform could provide for us.

They had a lot of really nice compelling networking features. They had really cool hosted services, like BigQuery. Our machine learning team was super interested in GCP, because they had these big beefy machines with lots of GPUs that were pretty cheap. So my team started working on a proof of concept and we basically just want to answer the question; could be deploy just any random HubSpot app on to GCP and do it without anyone noticing a difference?

Our tools really ended up helping us here. So Rainmaker, that tool I mentioned before about provisioning machines. So we built that to abstract away Amazon, because we didn't want developers to have to think in terms of Amazon primitives, and what we ended up doing is just we looked and saw, “Okay. These are all the API calls we have to make to Amazon to create a machine.” Spin up the machine, set up networking, discs, etc. Let's just make an adapter for Google.

So what originally seemed like a really hard problem turned into just writing some code, having a drop-down on the form and — Bam! We had Google. That was really cool. Then once we had that, it was just a matter of setting up networking, setting up all the dependencies that we needed to run stuff. That meant Zookeeper, that meant MySQL, that meant Singularity. Once we had everything running there, we’re able to update Orion, add support for multiple regions. So a developer could just say, “I just want this in AWS in the US, or I just want this in GCP in the EU, or I want it to run in both.” We rolled that out. We got it working, and it was very cool.

It wasn’t all perfect. The two big worries that we had, one was that the way that we connected the networks together, we just use the standard — We took the AWS VPN service and the Google VPN service and just connected them together, and it worked, but we weren’t getting a throughput that we wanted. I think it went up about a gigabit a second and it took some time to ramp up to that. What we were looking for was closer to 10 gigabits a second.

One option was to rent some fibr, but that was way too expensive. The approach we ended up using was kind of rolling our own VPN. There's a software package called StrongSwan and we just set that up in AWS, GCP, published the IPs that they need to talk to and it all just worked. The other worry, which started as a selling point, and then became a huge worry, was this Google cloud platform thing called Live Migrations.

So I was talking about Amazon. I was complaining, “Machines in Amazon could disappear at a moment’s notice.” Sometimes they give you a heads up, but sometimes they don't. Instead, Google does this thing where if they have a planned outage or if they know the underlying hardware is going to go bad on the machine, they will take your VM, snapshot all the data and the RAM and the CPU, copy it over to a new piece of hardware. One that’s all copied, they’ll pause the execution of your VM, copy the rest and then start it up on the new machine. It’s very cool.

When we heard about it, we were just like, “Holy shit! This is really neat,” but if you're doing this on a machine with a lot of RAM or a lot of RAM churn, that pause that should be imperceptible could take up to a second, which is not that great if it happens to one of your underlying data stores. That can have ripple effects. That can really affect your service. We’ve voiced those concerns the Google. They kept saying like, “Yeah. It's going to be fine. Don’t worry about it,” but we didn't just take them at their words.

So what we ended up doing was we spun up an age-based cluster in GCP and we simulated the load that we usually have — Well, we didn’t simulate. We used the actual traffic in production, but sent it to Google, and we worked with the engineering team to trigger live migrations, and we ended up seeing no impact at all, which is really nice. So there is a happy ending there.

So things were generally looking good. We are feeling confident that we could actually run HubSpot on GCP. So what's next? Amazon, of course, was horrified. They want us to stay in AWS. Google wanted us to be their big success story, “They took HubSpot who is on AWS for 10 years and got them to fully migrate to Google,” but what we wanted to do ultimately was what was best for the customer. Customer, meaning our developers, but also actual HubSpot customers, and we couldn't decide the right thing to do without actually testing in the wild. So we ended up going with the kind of hybrid cloud and multi-provider approach. We settled on keeping AWS in the US and going with GCP in the EU.

So we started working on that. We originally did a scope down version of the EU data center. We didn't just clone HubSpot in the US and ran in the EU. We focused on data collection. What that means is — So if you’re interacting with HubSpot, if you fill out a form or click on a CTA link, open email, click a link in an email, or generate any kind of tracking event with HubSpot. If you're in the EU, our CDN will route you to our EU data center instead of the US.

It took, honestly — The hardest part of all this was the human aspect. It was, “Okay. Data collection. What services need to be running in the EU to support that? Okay. We got to get the tech leads of all of teams together and figure out, “Okay. What do you actually need to run to do this?” You’d be surprised how many skeletons are in the closet or how many things that we realized at the last minute. The best way that we ended up testing this was we got everything that we believed we needed running in the EU, and then for the span of probably a month and a half, every Wednesday during lunch — Lunch was provided. It was awesome. We would cut the network link between the US and the EU and then we start testing all these services to see are they still working? Do they work at all? Are they slow? What's going on? We come out with this laundry list of things. We didn't realize that a service was depending on some authentication service that lived in the US, or we didn't realize that the service was depending on some Amazon only data store.

So week by week we would cut the link, test things out, get our list of things to work on, work on it, come back next week, test it again. The lists will get shorter and shorter and shorter. It finally stopped being a list, which was awesome, and we declared success, and it was really cool. I guess the big take away is like we didn't go into this thinking, "Oh! We can depend on Amazon services because we'll be using a different cloud provider someday." We weren't trying to over optimize like that, but the abstractions that we made to make things easier for our developers ended up making it really easy for us to support multiple providers. That was really cool. So that is that.

Does anyone have any question?

[0:41:48.5] Q: Could you just walk through with your Mesos and Kubernetes, what you were doing as you were bringing that European data center up?

[0:41:54.4] TP: Yeah, totally. So Kubernetes, what we're using it right now is mostly date services. We use a ton of different data stores at HubSpot. We use MySQL, HBase, Elasticsearch, Memcache. Am I forgetting any? Reddis, Kafka? We're trying to kill Reddis. Reddis is going away.

[0:42:13.5] JM: Why are you trying to kill Reddis?

[0:42:14.9] TP: Just to cut down the number of things we own. [inaudible 0:42:16.9]. So the focus, and I guess to zoom out a little bit. I talked about Mesos. I talked about Singularity. It is like this awesome place to be. If you're developer writing code at HubSpot, you're deploying a singularity and life is good. If you're working on the infrastructure to HubSpot, your stuff doesn't run singularity, because singularity can't handle stateful services. It doesn't handle persistent storage or anything. So instead you're writing puppet code. Your writing [inaudible 0:42:47.9] plays, you're doing stuff like that, which is not necessarily bad. It's just not as nice.

So what we've been working on, we started early 2017, we're still working on it now, is migrating all of our data services on the Kubernetes, because Kubernetes has that nice support for persistent volumes and things like that. So we didn't end up using Kubernetes a ton for the EU just because the actual things that we are replicating in the EU, there weren't a ton of those things, but we do have a cluster there. We do run some things that — This is like very technical, but the trickiest thing that we have going on there is we run a lot of things in Docker, and we run around Docker registry so that we don't have to worry about the public one going down. That Docker registry is backed by S3, but we want the EU to be able to continue functioning if Amazon is down.

Okay. So we need a registry in the EU. So it's super easy to run one, but then what do we use for the data store? We could use GCS, but then we would need to keep all of our images and sync. So what we ended up doing was we built this. I guess it's like a proxy, and what you do is you point things that use S3 to this proxy, and what it does is first it checks is this file in GCS. If it is, return it immediately. If it's not, pull it in from S3, and while you're copying it over, save it in GCS. So we've got this registry in Kubernetes on the EU that's pulling the images from S3, storing in GCS so that we can use it later. One rankle was we're running that proxy in Docker in Kubernetes, we can host it in our own things. We ended up having to publish that one and have that be public, but kind of cool.

Does that answer your question?

[0:44:51.0] Q: So does the rook extension inside Kubernetes, will that make a difference for you? Because now being able to keep this file systems within Kubernetes.

[0:45:00.3] TP: That's a great question. We haven't really looked at any providers other than just the normal EBS and AWS and the PD one and GCP. Probably down the road we will be looking at GlusterFS or one of those things.

[0:45:15.8] JM: Are you using AWS Lambda for anything?

[0:45:18.7] TP: We're using it a little bit for things related to our content platform. I actually don't have a lot of insight into it other than we're using it.

[0:45:29.3] Q: Yeah, that was my same question. Do you think of anything on the serveless path other than deploying own Docker and have to scale your own stuff?

[0:45:38.6] TP: Serverless is something that we're aware of, but we haven't invested that much in. Honestly, I would be excited to use it just because I have tons of ideas for little web services that I want to write, and I don't end up doing it because of the initial cost of grabbing the boilerplate for a HubSpot blessed web service and updating it and deploying it and blah-blah-blah. If we were doing serverless now, it would be — So that I can just trigger little functions that I write.

[0:46:13.1] Q: You say you have about 600 service, and how do you do the service discovery and manage the service dependencies now and you hit the problem of a bunch of a dependency and then a long list of things that don't know who depends on who and how do you manage it now? Do you come up with a better strategy to deal with the service discovery and dependencies?

[0:46:38.4] TP: That's a great question. We have some very cool solutions, but I wouldn't say that we've totally solved it. Historically, HubSpot's kind of interesting and that all their services communicate over HTTP with JSON, but the way that they talk to each other is through well-known URLs. We would actually be hitting the same URLs that we publish to customers who want to use their API. We're kind of ouroborosing ourselves.

For example, I'm writing a web service. I want to talk to our email API to send an email. I would make a web request, the hubapi.com/email/blah-blah-blah, and for the longest time that web request would literally leave our data center, hit our ELB and then go back in. It's like, "What in the world?"

But it was the easiest thing to do at the time, but over time because we're, I guess, monoglot is the word. We're primarily a Java shop and we have a really strict control over the framework that we're using. So we were actually able to bake intelligence into our HTTP client so that when you make an HTTP request, it sees, "Oh! Hub API? That HubSpot." "Oh! /email? That's the email API." I know what IPs is the email API is. I'm going to direct you directly to the email API. It's pretty cool. The wrinkle is you need to know that the email API is that /email. That's the part we haven't solved yet.

[0:48:13.1] Q: It doesn't go through the internal balancer? Is it through some kind of static IP edges?

[0:48:17.4] TP: Yeah. We're not Kubernetes style, like service IPs yet. Every task in singularity has like a host port combination, and we — If this is a request coming in from the public internet, it will go through an ELB and then an nginx load balancer, but if it's a service to service request, wormhole, which is the library that handles all these, will do the load-balancing of the client side. It uses heuristics like sending it or per preferring staying inside the same availability zone, or if an upstream has been failing lately, avoid that upstream to do load-balancing that way.

So the question was if we were starting HubSpot today and we're spinning things up, and I wanted to support multiple regions. Would I want to go with AWS or Google?

It really depends on what we're trying to do. So we were using Amazon for a long time. We really focus — Because we're using it for so long, we took all the benefits for granted and we really just zeroed in on, "These are things we hate about Amazon," like X, Y, Z. We're all excited to try Google because they have all these nice features. I think because Google is such a recent player, they have less knobs and levers to turn. If you're starting from scratch on something's simple, that's probably the right — That's probably what you want. But if you're doing something big and complicated — We have found uses for almost every knob and lever in Amazon. Me, Tom Petr, with my Tom Petr brain, I would probably go with Amazon, but I don't think that you could make a bad call. Google has some really nice features, like superior networking, really cheap GPU instances.

Just being able to not think about instance classes — So in AWS — Don't get me started about reservations in AWS. So we run all different types of instance types in AWS, M4s, I3s etc., and in order to save the most amount of money, we got to buy reserves. As you may know, when you buy a reserve, it's cheaper than the normal cost, but you're paying for a year or three years no matter what. That's fine and you can do forecasting and make it work, but down the line you may realize that, "Oh! I am on —" My coworker is smiling because we have to deal with this. I am on C4s, but it turns out my workload is like very I/O intensive and like I should have gone with I3s, but I bought all these C4 reserves. What am I going to do?

There are some ways around that with convertible reservations and everything. But the nice thing about Google is you're not thinking about things in terms of classes. It's just give me this many CPU, this much memory and you're good to go. Pros and cons.

[0:51:27.6] Q: Do you guys use DynamoDB from Amazon?

[0:51:32.2] TP: No, we don't. We just run everything ourselves, MySQL —

[0:51:36.5] Q: You mostly just use EC2s and things like that. One problem I have hit before is you provision the capacity from AWS, but you can never quite hit that capacity. They always — like you can never, from a consumer point of view, if you provision 100, you always want to maximize one 100 at a time, right? But I found like most of the time you can hit like maybe 75 and then once in a while it's up to 120 and then back down. You're always not able to utilize maximum 100 all the time. I don't know if you guys see the same problem?

[0:52:12.7] TP: Yeah, we definitely had that problem, and running things on Mesos helped a lot. So we get big beefy M4 10XLs, M4 16XLs and we pack lots and lots of tax on to those. So we're able to have really dense, densely packed servers which helps there. Reserves help a lot, but like I said before, getting your reserves exactly right can be hard. You can under reserve and then pay extra money or you can over reserve and waste your money.

The approach that we're moving towards is using reserves but then for workloads that can handle it, blank spot instances to handle just the little cases where we're under reserved and maybe don't want to commit to a year for something. Yeah, it's hard. It would be nice if this is just solved.

[0:53:05.3] Q: Thank you. I'm really interested in understanding a little bit about your monitoring and like how you kind of handle that. What kind of monitoring services are you using to kind of manage all of these services?

[0:53:16.2] TP: Yeah, great question. Again, going back to the whole monoglot, the HubSpot framework for running things. When you make and new HubSpot service, you get a ton of things for free, and one of those things is monitoring. We have a tool called Rodan.

Okay, interesting side note. You can tell how old something is at HubSpot by what it's named after. We had a period of Godzilla monsters and then now we're in like a space theme and I don't know what the next one is going to be. Rodin, it's been around since — I want to say like 2013, and it's this really simple system that just ingests metrics and then it has a rules engine for alerting. There are open source equivalents now.

So when you make a service at HubSpot, it's automatically recording into Rodan for you, and when you go to Rodan you're going to see everything you need to know. If it's a web service, it'd be request per second, error rate, keep space, stuff like that, and we make it really easy to add extra instrumentation if you have other metrics that you want to share.

The other cool thing about Rodan is all the rules are not specific to your service. It's more your service is part of a family, and that family will have general rules, like fire off an alert if you're request per second hits zero or your error rate goes above whatever. So you get a lot of things for free, which is really nice.

[0:54:44.3] Q: Thanks for that answer. That was very technical. One more question. How do you scale your services in the system? Do you guys use any kind of queuing system?

[0:54:53.9] TP: Yeah. Scaling is complicated, because there's scaling individual services and then scaling the clusters themselves. My team is currently working on cluster scaling, but we're not there yet. Right now it's just — If we're running low on resources, we add more. In terms of individual services, we actually have a system called AutoScale that looks at different sources of data coming in to decide, “Should I scale up this worker or web service?”

So, for example, for web services, if your request for second go over a threshold or if your queue time starts climbing, we’ll know, “Okay. We should give you more instances.” The wrinkle there, which we actually found out the hard way recently, is there are so many variables there. What I mean by that is — Okay. So AutoScale scales up your service. It’s going to start using more memory. It’s going to start using more CPU. That we were expecting. One thing that we didn’t expect is database connections. When you scale up your service, you’re using more database connections and we actually have hit some issues recently where we max out connections on a DB because of that. So it's interesting.

[0:56:05.3] Q: How do you scale your database? [inaudible 0:56:08.7]. How do you scale? You just add more memory and CPU, go to larger instance for the database?

[0:56:15.8] JM: The question is how do you scale your database?

[0:56:18.5] TP: When we need to — So we run our — We got our database guy over here. So he’s going to keep me honest. So MySQL specifically, we run a master slave set up. Which is going to be a master and at least one slave. When we need to upsize a cluster, what we typically do is clone a slave, but have that clone be a larger instance size or clone multiple, because we need a full cluster. Then once we have those bigger machines up and replicating, we fail over the master to the machine. So now we have a master that is sized up, a slave that’s sized up and then the original ones. We just disperse those original ones.

[0:57:04.7] Q: Actually you get the maximize size with a two terabyte machine or something, and do you guys use some kind of sharding partitioning scheme?

[0:57:13.3] TP: We do sharding. We do use partitioning. One really exciting — It’s almost like I planted this question. So one really exciting thing that we’re working on in conjunction with our move to Kubernetes is transitioning all of our MySQL databases on to this project called Vitess, which came out. It's an open source project from YouTube, and you can almost think of it like a proxy for MySQL. So what it allows you to do is you can do query rewriting so that you can improve queries. You could do auto sharding. If your database is getting large, it will handle sharding it for you. It's Vitess, V-I-T-E-S-S.

[0:57:55.4] JM: Vitess, and you said that was something that came out of YouTube.

[0:57:59.3] TP: Yup. It's an open source product from YouTube. Very cool stuff.

[0:58:02.4] Q: Do you use the [inaudible 0:58:05.0] on RDS or Aura?

[0:58:07.4] JM: Are you using RDS or Aurora?

[0:58:09.5] TP: So we may have used RDS a long time ago, but I believe it was expensive and we had issues early on and we like that control of running this all ourselves. So we don't currently use RDS. We don't use Aurora either just because we have all of our things already squared away.

[0:58:30.5] JM: Why is HubSpot such a build over buy type of shop? You're a high level service. You would expect a high level service to be more of a buy over build kind of shop, right? But you guys, it sound like your — Condensed form of the question; why do you build over buy?

[0:58:50.8] TP: Yeah. I think it comes down to maximizing cost savings and just giving us all the knobs and levers that we need to scale. I want to say for all of our data stores, like we're using them at extreme scale and it's the kind of thing where if we were using a hosted system, it would just be too much money for us.

[0:59:13.1] Q: Thank you. Yeah, I'm really interest in the money part actually, billing. I could imagine there might be like a small army of people that you work with. I want to really know about the soft skill side of it, the people side of it. To actually make moves and to continue at the velocity that you guys are releasing new things. How do you do you work with like a billing department? Are you guys like really close to guys buttheads? I really want to know more just about — Yeah, just like the realities of like the costs of doing all these types of things. Do you think about it? Does it affect everything?

[0:59:47.1] TP: Oh! That's an awesome question. So the short answer is I want to say like five years ago, we really didn't think about it. Instances would just be running in EC2 and we wouldn't be aware of that. We just wouldn't be thinking about it, and it made a difference. Like we were spending a lot of money in Amazon and then we tightened the belt. This was an idea that was floated around that was kind of a bomber that they didn't do it. They wanted to actually have like bounties for shutting down machines just because they're so many lying around.

The combination of better reporting and, like I was saying with Mesos, just hacking more densely on bigger machines helped us cut costs down immensely. The thing that I like about HubSpot is the managers are very good shit umbrellas, like people don't — You're not going to do something because you're thinking like, "Oh! How much would that cost?" But everyone can kind of has a sense in their mind, like what good judgment is different service.

So we're not thinking about dollar amounts when we do things, but their finance and like managers in the organization. If they see something, they can reach and say like, "Hey, this actually kind of expensive. Can we do better?" You know what I mean? Instant managers.

[1:01:13.5] JM: Manage controls the budget.

All right. Well, I think we've elapsed all of our time for questions. Thank you everybody for asking awesome questions, and thanks Tom for a brave talk without slides. I think you actually did a pretty good job. I was pretty skeptical upfront, but I think you did all right.

[1:01:33.2] TP: Thank you everyone.

[1:01:34.9] JM: Yeah. Again, thanks everybody for coming out. Thank you HubSpot for hosting place. Thanks Polsinelli for the food. If you don't listen to podcasts, please check it out. If you do, I would love to hear your feedback, and I think we'll be hanging out for another 15, 20 minutes. So happy to talk to anybody and talk to other people, network.

[END]