EPISODE 1533

[INTRODUCTION]

[00:00:00] ANNOUNCER: This episode is hosted by Jordi Mon Companys. Check out the show notes to follow him on Twitter.

[INTERVIEW]

[00:00:07] JMC: Hi, Sam. Welcome to Software Engineering Daily.

[00:00:10] SA: Hi, Jordi. Thanks for having me.

[00:00:12] JMC: My pleasure. And we're here to talk about Dagger, the project and product that Sam and others have recently founded and launched. In general, software supply chain has become an incredibly complex problem, both the trusted supply chain and the entrusted one, right? The one that companies control. But also, what goes beyond what any company can control? And any company can control is basically the tip of the iceberg.

What I mean by trusted components of the software supply chain are just packages, workflows, processors, checks, tests, whatever that are internally compulsory within an organization. But also, dependencies, libraries, outbound calls, analysis and all the modern software, all the modern processes and steps that modern software relies upon. And those mostly, especially in the open source world or companies that rely on open source software, sit outside of the controlled space of any company no matter how big this company is.

For that, you have two options. Be or become Google and have a massive monolith the size of the Pyramids of Giza in which you have all the code, all the dependencies and everything contained, right? But most of us are just not Google in a way, right? We tend to rely on others. And open source projects are maintained and owned by other people. And that's the reality for most of us. Does Dagger help with this huge complex landscape to start with?

[00:01:56] SA: Yeah. First of all, I'll describe in a few words what is Dagger. Dagger is a programmable Ci/CD engine that runs your pipeline in containers. As you describe, the pipeline can be a series of step that can build your application, test your code, trigger or deploy. And that pipeline usually is described in most of the CI system as a YAML file, which you usually call that a workflow in most CI systems. Or it's actually a pipeline.

And all of those steps actually are pieces of code that needs to be more or less complex. Usually small companies start with a very simple deployment script and they end up with something fairly large and complex. And the problem there is that, on one side, you have part of this complexity and the graph of dependencies that we call at Dagger. I can explain in more details what it means and why it matters.

But, yeah, on one side you have some very large complexity in a language like YAML. On the other side you have a certain number of batch scripts and custom code that you need to run and maintain. Usually, this code is not very – run efficiently. At least the parallelization caching, all of that needs to be taken to account. Security with secrets.

And also, it's usually not portable. You probably have heard about people having gigantic Jenkins configuration that they need to migrate over at some point. It never happens. But usually when it happens, everything has to be done from scratch. Because the code is not reusable. The code that exists actually is usually deeply tied to the infrastructure. This infrastructure, again, can be Jenkins running on various cloud providers, **[inaudible 00:03:56]**, etc. Or even GitHub actions that's taking off also. But you end up with the same problem. Everything is kind of all glued together to the underlying infrastructure.

And so, Dagger aims to solve that problem in a couple of ways. First of all, we think that your CI/CD pipeline needs to be programmable with the same kind of tools that you use when you write your application. That's why we released yesterday an SDK for Go. So, now you can actually use a Go library, a Go SDK to run to program your pipelines using Dagger.

Once you implemented your pipeline, every single step of the pipeline runs inside a container. But you don't have to manage that complexity yourself. Dagger simplifies this for you. And then it means you can also run it everywhere and move it from an infrastructure to another. Dagger

manages secrets, for instance. It can parallelize your steps when – They can be parallelized through this notion of DAG again. And they can also be cached. The important thing is, if you manage a large code base and you just change a very simple line of code, you don't want to re-execute the whole pipeline again. And Dagger actually simplifies that as well through a caching mechanism that I can go into more details.

[00:05:29] JMC: Okay. Yeah, let's actually dive into the caching mechanism. Does it cache dependencies and, therefore, save time when calling those every time one after the other? Or is it like incremental builds, in the case of CI, that you just said, yeah, you change the line of code. The code base that you're trying to build is 10,000 lines of code. So, therefore, 99,999 do not need to be built because you only change the one, right?

[00:06:00] SA: Yeah.

[00:06:01] JMC: Are those like caching pre-build and caching during the build? Are those things that Dagger **[inaudible 00:06:11]**?

[00:06:11] SA: It's managed not at the code level, but at the – I would say it's not exactly quite correct. But at the directory level. If that makes sense? Basically, the directory is seen as an input. It can be any kind of input. It can be a remote Git repository. It can be another container image. It can be any kind of binary input.

And if this input didn't change, basically we know that we don't have to rerun the same operation on it. As an example, let's say you build a Go application. And from this Go application you produce a binary. You will include this binary into a container. And the result of this container will have an addressable SHA-256 with a long digest.

We know for sure that if the code didn't change, at the end, the very end, you don't need a new SHA-256. And you don't actually do need to rerun the whole content again. I mean, the whole build process.

And so, well, imagine if you have to check this yourself and want to manage that condition. Well, with Dagger, you don't have to manage any of that. Right now, it works. Actually, underneath, we

are using a project called BuildKit. BuildKit is the build engine of Docker. It's a project that we worked on. Actually, I didn't share much of my background. But before that, before Dagger, before I co-founded Dagger with Solomon Hykes, Andrea Luzzardi, we were, the three of us, at Docker for many years. Actually, Solomon was the founder. And we worked on those projects before. We actually built them from scratch.

We understand pretty well how they work. And we thought that it was very important to leverage them into Dagger. Most people use BuildKit today, but they don't know about it. It's actually the build engine. Every time you do a Docker build, you actually use BuildKit and use this caching mechanism.

[00:08:20] JMC: Is Dagger going to abstract Dagger users from BuildKit, too, like Docker does?

[00:08:25] SA: Right. Exactly. Yeah. Exactly. Docker does it today by using Docker files. But BuildKit is much more powerful than just running Docker files. And so, when you use Dagger, you can reuse your Docker file. BuildKit actually understands Docker file pretty well. I mean, the support is obviously very, very solid. But you can actually use the primitives of BuildKit without Docker files. And this is what Dagger gives you access to through the Go SDK.

[00:09:01] JMC: Let's actually jump back, because you mentioned Solomon. You name you mentioned the other co-founder of Dagger. His name was? Apologies. I didn't get it. The third co-founder of Dagger?

[00:09:10] SA: Oh, Andrea Luzzardi?

[00:09:11] JMC: Okay.

[00:09:11] SA: Andrea, actually, a few words about him. Because we we don't often talk about Andrea. Because Solomon is now pretty famous in the space for good reasons. And Andrea was actually the one who created Swarm at Docker. And he also was the engineer in my team that had the very first commit on Docker back in the days when Docker was still a prototype and one of the many projects we're working on. Andrea had his, basically, name on the very first line of code at Docker.

[00:09:47] JMC: Nice. Actually, this connects very well with what I was going to ask you. Dagger is a bit in that same phase right now, right? Docker was when Andrea made the same commit. But you guys are learning from your – You guys are applying the learnings from Docker, right? Tell me about not necessarily the time at Docker, but actually with what idea did you guys leave Docker? And what motivated you to found eventually Dagger? What was the problem? How did you research that? And how did you come about this experiment right now?

[00:10:23] SA: Yeah, it's actually an interesting story, because I don't think it started the usual way for a startup. And what I mean by that is we actually left Docker at different times. I left in mid-2018. And to be honest, I wasn't sure what to do next. I just wanted to take some time off and get some mental space to think about it. And Solomon was the same. He left a bit before me. And Andrea a bit after basically. But we were all the same thinking about life. Thinking about what's next.

And what we ended up realizing was that we wanted to work together again. That was really the starting point of the company. It was really the team. And it's really not usual, because most startups start with an idea and then form a team around it. That what I see a lot with other startups. And so, yeah, since it was the opposite, we ended up, the three of us together, thinking about, "Okay, what should we do?" And so –

[00:11:27] JMC: Baseline, the starting point was that you guys wanted to work together.

[00:11:31] SA: Yeah.

[00:11:32] JMC: Whether it was building a farm.

[00:11:34] SA: Pretty much.

[00:11:34] JMC: Yeah. Or whatever, right? That came second.

[00:11:38] SA: Yeah. Well, thankfully, we didn't build a farm. And so, what we started to do in the very early days was to interview people. We started to ask for meetings with any kind of

companies, small, medium, large. Any kind of people that would be to talk to us also. Because we didn't have anything to sell, anything to share other than our experience.

And along the way, we started to zoom into different categories. Usually, you can split this industry in four main categories. There is the code, the build and test. Then you have the deployment. That's kind of, I mean, strongly linked to the build and test as well. And then you have the run. How you run those assets after you manage the deployment logic and everything?

And so, for sure, we knew that we were not going to do something for an IDE or something that can help you every day with your development. And something at the end that would manage the runtime of your containers that I think cloud providers are actually doing a good work right now in that space. And so, we identify a problem really in the middle with the build, test and deploy.

And this is where, after talking to a lot of people, at some point we were hearing the same thing over and over again. And that thing was, well, we started simple with a PaaS like Heroku or something very simple to use. At some point, it was too small for us, or too expensive, or not flexible enough. We started to adopt managed services. Our sales problem is then you – the experience that you have with something like Heroku, you lose it when you move to a managed service.

And so, this is when the problems start because this is when you actually need an internal platform, an internal PaaS. And this is when you usually open your IDE, name your files, deploy it at the stage. Write this file or build. Run this file from your CI. It's one of the steps in the YAML workflow that I described. And you think that the problem is solved.

The problem is that this file and this scripting thing evolve and never end actually. And as the company gets larger, you usually end up with a very large deployment dashboard. You have a platform team that's getting bigger and bigger. Usually, proportional to the rest of the engineering team. And their goal is to – we usually talk about developer productivity. And, really, usually the problem is like how can a developer be abstracted from the complexity of the infrastructure and ship their feature really fast?

And so, this is usually when they have to build this team is actually pure software engineering work. We call that platform engineering. But it's actually a shame that they don't have access to the same tool than other software engineers. That's when you end up with gigantic files, YAML files and a bunch of JSON transformation, building, schema validation on top. Trying to make it easy for devs. But it's not easy at all.

And seeing all of those companies doing these two themselves over and over again in a way that's not reusable. Because usually those platform teams don't – when they they actually create something pretty good that works, their code is not reusable because they take a lot of shortcuts for the company that they work for. And even when they leave the company and they gain this experience solving the problem once, they usually start from scratch again.

And so, we saw this problem with so many companies out there that at some point were convinced that everyone has this problem. And this is when we thought, "Okay, there is there is really a product missing there." A product that can allows you to program your pipelines, first of all. Makes them portable. The infrastructure, I think, is great. Most CI systems are already performant today. But it's a problem that the CI infrastructure actually wraps everything else, right? The management of pipelines. The way you will do your steps. The way you will manage your secrets. All of that.

Yeah. And actually, I didn't share about what coming next. But, obviously, the reusability of code and the extension system is very important. More SDKs to support more programming languages, etc. I can dig into whatever you're interested about.

[00:16:36] JMC: Okay. Thanks for the background. That was perfect actually. And I see that same problem everywhere in my day-to-day job. I mean, it's kind of clear. The solution is not clear. The problem is. Exactly.

Let's start from the beginning, making it programmable. When I think about something as code, and therefore programmable, I usually tend to – my mind goes to YAML, JSON and any database programming language in a way, if I'm allowed to say that. And therefore, I'm thinking of declarative.

But I did hear in your answer to the first question that you just released a Go SDK uh. What balance between imperative and declarative are you striking at Dagger? And, yeah, does that balance out in the programmable phase for –

[00:17:34] SA: Yeah. This question about declarative versus imperative, I mean, comes up a lot, especially here at KubeCon with Kubernetes. We tend to take shortcuts there and think that everything has to be declarative. And that's it. I think it's not that simple. Dagger actually forces you to be declarative. But declarative means usually certain limitations. We try to actually have you not suffer from those limitations in some ways.

What I mean by that is when you use the Go SDK, it doesn't feel like a declarative API. That's very important. And certain steps honestly don't need to be declarative on a deployment pipeline.

On the infrastructure side, because I hear a lot about **[inaudible 00:18:31]** even Kubernetes in some ways. It's very important for those products and projects to be declarative. Because when you think about infrastructure provisioning, you really want to define and describe and then state. If I need this port on this machine, or that number of memory, or that number of pods, well, I don't want to say add pod, right? I prefer to say I want five pods, right? And so, this is where you need to be declarative. Or you don't want to say I need more memory, etc.

When it comes to deployment pipeline, well, you have – this is when I need to explain a bit about what is a DAG.

[00:19:20] JMC: Okay. Let's move on to that.

[00:19:21] SA: Yeah, deployment pipeline is basically a graph of dependencies. So, you have a series of steps. You can call that the nodes in a graph. And those nodes depend on each other or not depends. Basically, you can have some – I'll take an example to make sure it's clear. Let's say I'm building a code that I pulled from a Git repository. And I need a container at the end. Let's say that actually I will build two containers outside of this repository.

I have the code. Let's say it's a monorepo with a frontend and a backhand. Let's say both will end up being containers, containerized. There, for instance, the first node of the graph will be pulling the Git repo. That will be the first thing. If I have all those steps in my graph, that defines how to build a container. We know that those steps won't execute until the code is available, until the code has been built, until maybe the test has been run. And then until I have access maybe to the Docker file and the assets needed for building the container, and until all these steps actually resolve, then I can build the container.

And so, this is where you realize that this is a graph of dependencies. And some of those steps can run in parallel. For instance, those two containers, maybe they don't depend on each other. Actually, sometimes it's not true. But let's say, in that case, they don't depend on each other. They can be built in parallel.

The Dagger engine actually takes care of all of that. Figuring out what to run first? In what order? What can be parallelized? What can be taken from the cache instead of being rerun? Because let's say that I'm re-running it again and the code didn't change. I don't need to actually rerun everything again. In the CI/CD space, usually we use that term DAG a lot. It's not actually going to CI/CD only, because you have a lot of DAGs. For instance, Git.

[00:21:26] JMC: Yes. Yes. I think we mentioned it before. Not in this conversation, but in another one.

[00:21:30] SA: Yeah. And so, it's a generic. It's a generic graph algorithm. And so, it's very –

[00:21:37] JMC: We should mention that it stands for direct acyclic graph, right?

[00:21:40] SA: Right. Exactly. Directed acyclic graph.

[00:21:44] JMC: Directed.

[00:21:45] SA: Directed, yeah. Because there is a direction. And it enforces the algorithm and forces that you avoid cycles. That's very important that when it validates the engine, validates the graph, we detect if there is a cycle or not in the dependencies. And make sure that it's

directed in such a way that it goes from the top all the way to the bottom and all the steps are resolved. That's how a DAG works, at least for the Dagger engine.

And I don't remember the initial question why I was explaining Dag.

[00:22:19] JMC: Well, we came actually from the right balance between declarative and imperative in the Go SDK. I guess, is the first step in Dagger defining your pipeline in a programmable way? Does it require you to create this DAG the first thing? Or not necessarily?

[00:22:37] SA: Right. And so, well, first of all, you don't want to have the DAG in mind when you write, you program your pipeline. You just want to say, "Hey, this is how I built my code. This is how I fetch the source code. This is how I want to build my container. This is how I want to run my test." Because this graph can be gigantic for large applications. You can build hundreds of containers out of a monorepo or something like that. And, actually, not mentioning running the test and what happens if one of the tests fails? Etc.

This graph ideally wants to keep that in mind while you program your pipeline. But the DAG at the end needs to be declarative. It's very important that, at some point in the process, you have the whole representation of what's going to happen. And so, this is what Dagger does basically.

So, you have to go a little bit about how it works. We have the Go SDK in the front. Basically, we generate – You can use it like any Go library. Use it for pulling images, running commands inside the container, pushing containers, adding secrets, referencing secrets, taking some code from a local directory, taking some other information from the host, like environment variables, etc. You have a lot of primitives that allow you to do pretty much whatever you want.

And then, at some point in that code, you actually need to execute something. For instance, you usually call this at the end. You can actually run multiple execs. but I will keep that for later, because Dagger can actually allow you to pretty simply manage multiple DAGs without even caring about it. But let's –

[00:24:28] JMC: Okay. Yeah, let's keep it simple for this.

[00:24:29] SA: Yeah, let's keep this simple. And so, the idea is that, at some point, you will run your pipeline. And this is when, basically, we have the whole representation of the DAG.

A little bit of details about what's going on underneath, if people are interested, we actually use GraphQL, which is very not common for that use case. A lot of conversation, a lot of data that happened within the team about that. But we strongly believe that it was the right choice to make this DAG declarative. And it will actually save us a lot of time down the road for releasing more SDKs, having extensions written for many different languages. But, yeah, I'm getting into the kind of the weeds of the internals of the project.

[00:25:25] JMC: That connects with the portability, right? We can talk about it later, right?

[00:25:29] SA: Right. And also, for your question about being declarative, what's good is that at some point you have a declarative gateway somehow. It's not a gateway. It's an API. But my point is that it's your gateway towards running containers. And GraphQL will make sure that it's really declarative. And so, that's what I mean by being declarative without having to push that complexity to the developer writing the pipeline. That's very important to us.

[00:25:58] JMC: I see. You've described how Dagger is programmable, right? You just described it in this example. You mentioned that those pipelines described now in code are portable, right? How does anyone else, let's say, that I'm a developer in your team, would be able to use that same pipeline you just designed the DAG off and made it portable? How does that work?

[00:26:21] SA: Right now, I will describe what you have now with what we announced yesterday and what will be possible tomorrow. What we announced yesterday is a Go SDK. What you would do right now is you would share your code exactly as you would with any other Go code. Let's say you have a Git repo and showing the code of your app with someone.

Right now, actually, Dagger doesn't ship a CLI. The way you use it is you compile this code. You run it. That's it. So, like any other Go application. And then the Dagger is fully embedded. And basically, what the Dagger engine does will be fully portable. It will run exactly the same way everywhere you will do it.

And by the way, it was important for us while releasing the Go SDK to use the native tools that you use when you do Go. For instance, you install the Go SDK with a Go mode. I mean, single module. Simple as that.

[00:27:20] JMC: Why as that decision important for you guys?

[00:27:22] SA: It was important because – well, something I didn't cover is that we also introduced a few months ago a version of Dagger that supports Q, which is, yeah, a config language that's getting some traction. And what we realized was, although Q is very powerful, very promising, the development tools, like the integrations with IDEs with the language server and all of that, are not there yet. And generating APIs references from code, that kind of stuff. And so, the point is having all the tools that you usually use when you write any kind of application, it's very important. Because otherwise, you have to build all of that yourself. We actually built a language for ourselves for Q. At some point, we've realized that while bringing support for all the languages, it's better to leverage what those languages already offer.

For instance, like the compilation of your Go code in VS Code for instance, or other IDEs, is trivial, right? We take that as a given today. We think that, "Oh, yeah, for sure. Auto compilation." But it's actually super complex. And remove this from a developer, you will see that it's actually pretty critical for the developer productivity. Yeah, that's why it's very important to us to fully integrate into the developer environment. And this is what we'll do also for other languages down the road.

[00:28:53] JMC: Exactly. The Go SDK, I presume – tell me more about it. I mean, the strategy. But it's a starting point, right? What's coming next? Yeah, what's coming next?

[00:29:05] SA: Yeah. I mean, there are more things available on the repository, on the open source repository. I didn't mention it. But all of that is open source today. Right now, we are working on Python SDK, on Node SDK that support TypeScript and JavaScript.

And then we have some plans. Some people ask for PHP, for Java, I heard. So far, what I can share is that Python and Node are getting close to be ready. Yeah, and probably announcing tech previews by the end of the year. That's what we're working on right now.

Yeah. The thing is people should not wait to use it. And the point is even though **[inaudible 00:29:57]** previews, and behind it's using BuildKit. GraphQL actually turns into LLB instructions, which is what BuildKit is using behind. It's the language that BuildKit implements.

And the point is, although those SDKs are new, behind it, you are actually turning all of this into BuildKit instructions very simply. It's as solid as a Docker file. If you trust using Docker file for production, you should give Dagger a try. And obviously, the reason we don't promote this as production-ready is because the API might change. We still need a lot of feedback from users, which is why actually it's important for people to give it a try and give us feedback. And so, yeah, it will keep evolving and getting –

[00:30:51] JMC: Let's actually plug it. Where can users find all the requirements? All the things to download and try themselves? Where can they go?

[00:30:59] SA: Yeah. The simple thing is to go to dagger.io. And then, there you will find all the links you need to try it. You can go also directly to the docs, docs.dagger.io, and you will find all the steps to start and give it a try. You can also see a link to our GitHub repo. Everything is available.

We use also Discord a lot. There is a pretty active community of DevOps and platform engineers on our Discord. I encourage people to join us. We actually run community calls every Thursday. I believe there is one tomorrow. It's a 10 AM Pacific time. Yeah, we may want to change that time at some point for including more people on different time zones. Yeah, right now, 10 AM.

There are also recordings of all the calls available. And people usually join. It usually lasts an hour, an hour and a half. And people join us. Question, share their use case about Dagger. Share even like what they need in order to use Dagger. What they're missing? That kind of stuff. It's really interesting how the project is getting designed within the community. Even our launch

was done all in the open yesterday. And that was fun to see from people from the outside. Something we learned from Docker back in the days, where the community is really, really, really important to get a project to a major state really quickly. Yeah.

[00:32:44] JMC: Let's say I'm a Python engineer, a Python developer. The Python SDK is already ready. You, a Go developer. And the Go SDK that was announced yesterday was ready. Why is that important? I mean, I'm developing, I don't know, machine learning, notebooks, whatever. You have other goals. Your team has other goals. But you and I work from the same application in essence. With Dagger, would that be a problem? Or what's in it in Dagger that makes this actually a good scenario for –

[00:33:20] SA: Yeah. Well, first of all, it will work like any code again. I would ask you with the question, if you write some Python, notebooks or whatever for your team **[inaudible 00:33:32]**, how do you share your code today, right? You're probably using a Git repo somewhere.

And so, well, the idea is you move from a world where your CI/CD pipeline maybe on the CI system managed by a specific team. And even if you are a part of the team managing the pipeline, since it's completely integrated in the CI infrastructure, question is how will you and your team maintain and keep this pipeline evolving? Well, you're likely to have to open – If you're lucky and you use GitHub action, you may have to open a pull request on that Git repo, changing few things on the workflow YAML. It's YAML, first of all.

And so, you'll usually want to test these locally because you have no way to do so. So, you will open a pull request. Wait for GitHub Runner to kick in. You will wait patiently through the logs drinking coffee. And at some point, you'll see that, "Oh, I made a mistake. It's actually not working." You wait for the logs. Look at the error and start again.

That's usually how you will maintain your pipeline for your CI system. And again, in the context of GitHub action, in particular with Jenkins, can be entirely different depending on how you use it.

Imagine if this is just code, like any kind of code, and you can actually run it locally first. So, you change this code. You open a pull request on something that you already tried and run. And it's

actually real code. It's not like a set of instructions that you're not exactly sure how they will run until they hit the CI infrastructure. The concept is very different. But I would say very different from the existing way to run CIs, but not very different from the way people work today with other applications and code.

[00:35:36] JMC: I find it funny that the claim of GitOps was operations by pull request, right? So, bringing a best practice from the dev world into ops. This, in a way, is the opposite, is bringing a best practice of the infra world, of the ops world, into the – I mean, you can apply it to everywhere. But this is like improving the dev side of things in a way.

[00:36:00] SA: Yeah, yeah. And it's also not saying that you will skip the pull request and the validation step. But it's also very important that the main difference there is that when you will submit your pull request exactly like when you do with code, you will know it will have a good idea of what happens, right?

[00:36:21] JMC: It's going to be binary, right?

[00:36:23] SA: Yeah. Imagine if software engineers today, we will work like all platform engineers and every time they would open a pull request they would be like, "Oh, shit. I don't know if this code actually is going to work, because I cannot try it. I need to see how it runs on the CI system." And you just open a PR. And actually, the whole team gets a notification about your code failing. Thankfully, we have a solution for that. And, yeah.

[00:36:51] JMC: Okay. Where do you think then – I mean, you just mentioned that the community is vital in the development of the product. But I was to push you a bit into sort of like to be opinionated, what do you think Dagger is going like in the next few months? Where do you see things moving towards?

[00:37:13] SA: Yeah. Well, I think, Dagger – We're going to release more SDKs. We're going to get those APIs stable so people can actually rely on them for very complex and production application. At some point, I mentioned we were going to release an extension mechanism so Dagger can actually extend this GraphQL API with extensions written in any language. It starts to be a bit meta. But the way it will work is that, as soon as you have a pipeline that you like and

you put a lot of work into making it generic, because you don't want to do these same mistakes again of deploying writing a complex CI/CD pipeline and having to do that again during your next job, for instance. Well, you'll have the ability to publish this as an extension that you can share with other people.

Let's say you have the perfect way to build a mono repo and many containers the right way because you know exactly what to do, and you've done it many times. You have a specific way to inject your secrets at build time or whatever. Something very complex that took you some time. Where, at some point, you may want to create an extension from it. Or, also, it can be for other companies who have services. For instance, let's say you work at Vercel and you would like all the Dagger users to have access to Vercel in a very simple way in their pipeline. Well, you could actually distribute your own Vercel extension. Not pointing any names. Netlify, DigitalOcean, like any kind of infrastructure provider, right?

[00:38:59] JMC: In fact, didn't you? Didn't you guys already published one of these? Or not? **[inaudible 00:39:06]** something?

[00:39:07] SA: Yes. The thing is – so the Q version, that's table, has an extension systems with a lot of extensions that we published in the past. Now, with this new core, basically we actually ported the Q support into its own SDK. And we decided not to launch the extension mechanism yet, because we want to see what people do with the – As soon as you manage an extension, if at some point we have to change the API because some users tell us it's not the right thing, it will break extensions, we want to do things step-by-step to make sure it works. But technically, yes, there is already the extension support in the core.

Yeah, and we expect more extensions, more adoption, more SDKs. We actually – well, a quick hint just to share something interesting with people who actually listen to this point. We're also going to – we are working on a cloud service that will provide a web UI. Basically, it allows you to instrument this GraphQL API directly from the web. We're working on that now.

And we actually might release it in tech preview before the end of the year. But that basically – Yeah, we're going to change a lot the way platform engineers work. That what you should take

from this and follow the launch, the several launches that we will announce in the next few weeks.

[00:40:44] JMC: In fact, I mean, with that, we can wrap it up. Because that was fantastic. I actually had a couple more questions. But before I move on to the last question. Where can people follow you specifically and Dagger? Can you tell us where? I presume Twitter must be a relevant means of announcements?

[00:41:07] SA: Yeah, I think Twitter is – For people using Twitter, I think it's a good way. We are @dagger_io on Twitter. I am Sam_Alba. I don't tweet a lot, to be honest.

[00:41:20] JMC: Solomon does, right?

[00:41:21] SA: Solomon does, yeah. I actually read a lot of his tweets.

[00:41:26] JMC: Yeah, he's a really interesting –

[00:41:28] SA: Yeah, of course, every day. And he's solomonstre on –

[00:41:34] JMC: Solomonstre?

[00:41:35] SA: Solomonstre with the French spelling. Monstre, Solomonstre, basically. And so, yeah, every time. All the tweets are being – all the important announcements are usually relayed on Twitter by Solomon, myself, the rest of the team. And obviously, the dagger_io account. And I would say, also Discord.

[00:41:55] JMC: The community.

[00:41:56] SA: Yeah, the community on Discord. We share everything we do on Discord. Probably more than on Twitter actually. And people can ask instant help, or questions about anything, or something missing in the docs, or anything. Yeah, Discord is probably the best way. But Twitter works, too.

[00:42:16] JMC: I like the fact that your product development strategy, your process looks a bit like a DAG. You're not moving on to the next step until one is finished, right? Once you've validated that the APIs look like – the community likes them and so forth, then you will provide the next thing, which is very cautious. And I like it. And it's very bottoms-up-driven.

Last thing, just about the company. Again, what does the product look like in the future? You raised the money not long ago, right?

[00:42:47] SA: Right. Yeah. Yeah, in March, we closed our series A. Now we have funding for quite some time, I would say. The team is about – I think we are 21 people exactly. Yeah.

[00:43:00] JMC: And it's a distributed company, correct?

[00:43:01] SA: Yes, it's fully distributed. We have people in India, Portugal, France, UK, North America, South America. Yeah, it's fully distributed or remote. Yeah, and we plan to grow some more. If at some point people are interested to work on the goal after trying Dagger or they think that they would enjoy working on this product every day, they should reach out.

Actually, they can send me an email directly, sam@dagger.io. And we also have hiring@dagger.io. There is also a jobs page on the website. I think it's dagger.io/careers. Yeah, just to make sure I spell it right.

[00:43:52] JMC: Well, Sam, unless you want to cover something else that we've missed, I'm happy to conclude the interview here. Did we miss anything?

[00:44:02] SA: I don't think so. Except if at some point people have questions, too many questions about – if they all have the same questions, yeah, it means we probably missed something.

[00:44:12] JMC: If the Software Engineer Daily has also – the community has a lot of questions, I'm happy to bring you in another time, or Solomon, or Andrea, or whatever.

[00:44:19] SA: Happy to join.

[00:44:20] SA: And deep dive into some of these specifics. But otherwise, good luck. And I'll see you around.

[00:44:28] SA: Great. Thanks for the opportunity.

[00:44:30] JMC: My pleasure.

[END]