EPISODE 1366

[INTRODUCTION]

[00:00:00] KP: One of the most painful parts of getting started on a new development team is getting one's environment set up, whether it's because of undocumented steps, an overly complex setup, or simply the challenges of understanding how all the pieces fit together. Getting a dev environment up often feels like a chore to be suffered through in order to do what I want, contribute to the code base. Gitpod seeks to solve these and other common challenges. With Gitpod, you can spin up fresh automated dev environments in the cloud in seconds.

In this episode, I interview Sven Efftinge, CEO of Gitpod.

[INTERVIEW]

[00:00:45] KP: Sven, welcome to Software Engineering Daily.

[00:00:48] SE: Hi, Kyle. Thanks for having me.

[00:00:50] KP: Before we get into our main discussion, I was wondering if you could tell me a little bit about your journey as a software engineer.

[00:00:57] SE: Yeah, sure. So, I mean, I started out as a musician. And when I kind of got closer to my 30s I realized I couldn't make a lot of money with that or actually a living. So I studied computer science. And immediately, I think in the first year already, I got a job and then built tools. Like I started with Java and Java EE stuff and just thought, “Hey, this is all so bad. It needs to be improved.” And so I jumped into code generation stuff, but also like domain specific languages, compiler constructions. I built open source tools around programming language. I built a java alternative called Extend, which is a functional statically-typed programming language. And I think spent a couple of years as a manager in a consulting firm and then five years or so ago I created a startup, founded a startup with two friends where we built development tools for international clients like General Electrics,

or Ericsson. So they asked us to build specific tools for their engineers. That company still exists. It's called TypeFox. And in that company we also built the Gitpod thing eventually, and so which is now its own company. And yeah, I mean, I think I've been around in engineering for 15 years now or so.

[00:02:34] KP: And what is Gitpod?

[00:02:36] SE: Gitpod provides dev environment for engineers and engineering teams. A dev environment is basically what you need in order to start coding. That is an IDE obviously, but also like the repositories, the source code, but also all the tools you need in order to compile the source code, run the source code, like application servers, database, things like that, your SDKs, your libraries, your builds tools and so on. And what Gitpod does is it allows you to write down in code how such a dev environment should be structured and then it provisions that for you in the cloud.

And the cool thing is you can version that configuration in code. So it's dev environments as code. And then you get fresh dev environments for all the tasks you do on a daily basis. Like you want to start fixing a bug, you just spin up a fresh death environment based on the current default branch commit. Use it and then you close the deaf environment. It's kind of disposable. You don't have to go back and configure. And I, for instance, never do git fetch or something like that anymore.

And yeah, so that's basically GItpod. It's a new way of bringing your development environments into the continuous developer pipeline. If you think about it, all the tools today are somehow in this pipeline, in the cloud. You have the GitLab, GitHubs, but also your CI/CD tools and so on. Only when it comes to development, people still go local and configure their local machines and massage them so they work with all the projects. And Gitpod basically moves this stuff into the cloud, streamlines. It allows you to share configuration in your team and basically be ready to code on all your projects at any time.

[00:04:39] KP: So Gitpod is going to deliver me an environment in the cloud. That has some similarities in my mind to Terraform. How do you compare and contrast to a tool like that?

[00:04:49] SE: Well, Terraform is having infrastructure as code more or less, right? And you can create new clusters or things like that as you will. The dev environment, it is similar, but you would not do that with Terraform. What a dev environment really is is kind of this interactive shell and then the tools on top you need, like an IDE. With Gitpod, for instance, you get a web IDE, which is VS Code. And you can just work with a browser, with an iPad, whatnot. But you can also install a local companion app and then you have port forwarding to your local machine so you can really use local host, URLs, although you are running in a kind of remote workspace. And then you can also use stuff like the VS Code remote support. So you use your local VS Code app, the standard one, and connect to the remote workspaces, but also use JetBrains and so on. So in some sense it is similar to Terraform, but very, very specialized on development environments.

[00:06:02] KP: So how do I specify my environment?

[00:06:05] SE: At the core, you put in a .gitpod YAML into your repository. And in there you describe first what kind of container image you want to use. So you can point to an existing container image or you point to a local Docker file, like local in that repository. You can just put a .gitpod Docker file, which is what most people do, next to the Gitpod YAML. And there you describe what tools you want to have in terms of the Docker file.

The next thing is you describe what are the things that should run when the repository is cloned? So the dockerfile stuff is kind of the basis. That's your operating system you put in all the tools you need. But then we clone the repository and then you need to basically inflate the repository somehow. Inflating means running the build script, downloading dependencies, running code generators, compilers, whatnot, so that it is in a state that you can start coding.

The cool thing about this INI task is that Gitpod has the notion of prebuilds. And so these tasks run asynchronously. They run before you want to have a Dev environment. Gitpod registers on your GitHub or GitLab repository. It registers a web hook. When someone pushes a branch, Gitpod will start building a workspace for this branch, for this commit. So we clone the repository and run these scripts. We run NPM install or maven install, whatever you have.

And so this shaves off already all this time for you. At the end of this time a snapshot is taken. And then when a user comes by, a developer comes by and wants to start coding, this is all done. You end up in a completely inflated dev environment that has all the dependencies there. All the code is there. So you can really just start coding.

And then there are some more stuff in the Gitpod YAML that you can put in like what tasks should be executed when you really start an interactive workspace. So you probably want to have something like a watcher process or you want to start your dev mode, things like that. You can define what extensions all users should have installed in their VS Code instance. We also have user configuration, but that's not what you put into the Git repository obviously. So the Gitpod YAML owns only the stuff that is shared in the team. Port mappings, what kind of ports should we make public, for instance? Or do you want to get notifications when this port gets served? Stuff like that.

[00:08:59] KP: And how do I manage my environment variables in this scenario?

[00:09:04] SE: You have different ways of doing that. There is a user level, environment variable feature where like in Gitpod there is a dashboard where you see all the user belongings, the settings and so on. And you can also have and set up environment variables that get injected into your workspaces. You can also say, “I want to have this environment variable only for this kind of project.” So you can pattern match the projects as well. And then also of course you can have team level environment variables through Docker as usual.

[00:09:43] KP: Are you seeing any best practices or ways in which teams are collaborating with the project?

[00:09:50] SE: Yeah, the typical integration is through a webhook or a GitHub app where you connect Gitpod to your repository. And then Gitpod prebuilds all the branches. So on every branch, people can just start a workspace and work on that. But then also when you do – For instance, you create a pull request or a merge request in GitLab, then Gitpod will push a comment with a link to the prebuild blocks where you can see the progress there

and also of course a link to start a workspace based on that PR. Because a really cool use case, of course, is a deeper pull request, a code review.

And so people just start a dev environment for a code review while they have in another tab a running device environment for their current ongoing feature development. And that's a very common thing how you use this in a team basically. Other things, of course, you can take snapshots of your workspaces at any point in time. So if for instance you're able to identify an issue and you have a certain scenario where you can trigger that and you just want to share that with someone, but you need to share that asynchronously, you take a snapshot of your workspace, which generates a link. And then you can share that link with anyone. And people who click on that will get a fresh copy of that workspace. So they are not landing in your workspace. And the other is of course live collaboration. So you can invite people to your workspace and then they can also see the running thing.

[00:11:37] KP: So there's a phrase I think no development lead, or maybe I shouldn't speak for everyone, but a phrase I don't like to hear. And it's, “Well, it ran on my machine just fine.” If Gitpod makes it run in the cloud, have you solved that problem?

[00:11:51] SE: Yeah. I mean, it's not because it runs in the cloud, but it runs based on automation, right? That's the thing. It either runs or it doesn't. And yeah, I mean, usually. There are of course very often cases where it sometimes doesn't run nicely. The thing is, with Gitpod, this automation is applied continuously. As I said, like we run this really on every push, we run the automation.

And so if that is kind of fragile or sometimes doesn't work or it's just not perfect for everyone, the whole team understands that and there is a bigger impact and people really want to get this sorted out. And yeah, so that is definitely a point we make. Like you don't have this runs on my machine scenarios for sure. They are all the same machines. They are all set up the same way. And if it's a flakiness issue, you will see it more often than not, because it's running this automation continuously.

[00:13:03] KP: Do Gitpod users have to worry about something like the cold start problem as they set up the environment? So the first time I build my environment, it's understood,

maybe a lot of packages have to install and this sort of thing. If I do that locally, I know if I'm working in node there's a node modules directory still sitting there for me. If I'm spinning these environments up and down in the cloud, might I have to wait for all those things every morning when I start up my routine?

[00:13:30] SE: Oh, yeah. That's indeed solved by the prebuilds, right? This is running asynchronously. And if we hadn't solved that issue with something like prebuilb, of course it would not be practical to start fresh dev environments for every task because you would have to wait for that. But if you have to reuse a workspace per project, then going into the cloud is not really super helpful because – or automation in general is not super helpful because you would run this automation only once when you onboard on a project and then never again. So you run this once and then you have this state full mass that you have to massage over time and sync manually. And then of course configuration drift happens.

And so it was super important for us to get this continuously applied automation. And then of course with no compromises so that people should not wait. Like time is really important. So we introduce this notion of prebuilbs. And so that's all said. It wounds up your machines basically.

[00:14:43] KP: And can you give an example of what a prebuild is? What's maybe a popular one?

[00:14:48] SE: Yeah. I mean, just for example, you have a React project. When you check out a project and you clone it freshly, what you need to do is – And given you have all the tools you need like NPM and so on. It's all installed. Then the minimal thing you do is NPM install. And that, depending on the size of the project, runs in a few seconds, or in a minute, or something like that. Then maybe you have also GRPC things in there, right? You want to run the code generator as well. And then maybe it is not only a React pro, but there is –I don't know, Rust backend thing. And you need to run the Rust compiler and it also pulls down dependencies and so on.

And so you make sure this is all done already when you enter a new workspace. You tell Gitpod, “Hey, this, you can do already when someone pushes a change. You run through this.” And so that's what a prebuild is. You don't have to wait for this stuff. It's already there.

[00:15:54] KP: So the use case for Gitpod for managing my application code makes perfect sense to me. Is there part of the story for the databases and persistent storage that my application might want to connect to?

[00:16:07] SE: Yeah, you mean a long living state in test databases for instance?

[00:16:13] KP: Yeah. I mean, I guess, on one side you could say that's out of scope for Gitpod. That the team should set up a staging database and a dev database and just put in the credentials. I’m curious if there's anything beyond that or any best practices to apply.

[00:16:28] SE: Yeah. I mean, the best practice here is to not have long living state, but seed a database freshly. But of course it's not always practical. So if you want to have a database that is long-living, a test database that you use. You have, like, of course, the state somehow, somewhere, not in Gitpod. The point is can I how can I connect to it? And so that's what I previously mentioned is the local companion app. And the local companion app allows you to connect to ports in the workspace, but it also allows to do the connection in the other direction so you can provide services in the workspace that are reachable from your local machine, for instance. So basically, what you need to do is port forwarding mapping and you can automate this as well. So when you start a workspace and you are on your machine and you have access to a test database, you can configure it so that it connects to that. This is still kind of relatively like the local companion app is in preview mode. So we are still working on that. We definitely know and this is something we want to solve because that's just life. That's how developers work today. There might be a best practice, but we have to also deal with reality and existing environments. And so that's the thing we are solving.

And product-wise, in Gitpod, we have some sort of basic values and approaches. And one is that we try to be very orthogonal, which means we want to allow everyone to bring their environment and their tools and make them work within Gitpod as is. So we really try hard

to not impose any kind of strange – In Gitpod you need to solve these completely different things. So we are trying to get everything. All the tools you like locally, they should run and Gidpod. And they do for the most part, I think. And so the biggest issue here is really the port mapping, the networking. Lots of tools are built with local hosts in mind and lots of existing environments assume you are in the network of your local machine. And so we are solving that with bridging the network by doing port forwarding.

[00:19:07] KP: So if an individual or a team wants to adopt Gitpod, what's the onboarding process?

[00:19:12] SE: Basically you go to Gidpod and create a new project. You import your repository into Gitpod. Like what you basically do is you install the Gitpod app in GitHub, or um install a webhook on GitLab. There is a flow that that walks you through this. And then you can configure your repository and then you can invite members and you are all set up. That's it, like it's really mostly that. In GitLab we have native integration so that you already have a Gitpod button on the various contexts, like you see that already. For GitHub, you should install the browser extension that we have that would add a button to the various pages.

But yeah, it's not hard. Like you set up your project, you work a bit on the configuration, and then you invite your colleagues.

[00:20:18] KP: So Gitpod is as a service then? Is that how you look at it?

[00:20:23] SE: Yeah, it's a SaaS. Primarily a SaaS service, but it's also open source. And we also allow or offer self-hosting. So you can also install your Gitpod installation on your own Kubernetes clusters if you want. But I think the easiest to get started is just using Gitpod IO, the SaaS service.

[00:20:46] KP: Yeah. And when I sign up for that, so it's going to create my dev environment for me. Do I bring my own cloud? Or is that something you're provisioning as well for me?

[00:20:56] SE: Yeah. No. That is of course provision for you as well. You don't bring your own cloud. It's really like if you want to try it out now, you just go to your GitHub repository and prefix the URL with Gitpod IO hash. That's the easiest thing to get a workspace of your source code. It's really, really easy. But if you really want to adopt the value, then you should import the project and make sure you have the configuration, because then you get the automation in it, like you have a tailored Docker file. You can describe what a prebuild should do and so on.

[00:21:35] KP: So Gitpod is a fairly broad generic in a good way tool. Any developer could perhaps pick this up and use it for their project. I'm curious though, have you seen any early patterns and early adoption? Are there particular industries or styles of coders that have really gravitated towards Gitpod?

[00:21:53] SE: I can talk about that from two ends. Like individuals, I think, it's mostly – Actually, interestingly, engineers who are a bit more experienced and know a lot about automation. They probably also have pretty well shaped CI pipelines and want to just streamline the efficiency also of their teams and so on. And so we have these individuals. And then we also have companies reaching out. And those are mostly larger who have a dedicated kind of platform team where the whole team is just about developer experience, developer productivity. And they also reach out. And that's often financial institutions, surprisingly. Like you would think they are usually slow. And at least like in adopting these things, they pretty much well understand the pain points here. But it's also a case that these companies often have already something in place which doesn't work so nice like a Citrix solution or something like that. And of course going with something that runs in your browser more easily and is also less expensive and so on is of course very nice. But mostly, I think people who are concerned about developer experience and developer velocity, I think that's the people we see who adopt that at the moment.

[00:23:34] KP: From a security perspective it seems there's some possible advantages here as well. If I were a large enterprise company, the type that really wants to carefully control what developers have access to, managing everything in the cloud in a controlled way through Gitpod seems to bring some advantages. Do you see that as part of a more enterprise story?

[00:23:56] SE: I mean, it is a bit more – Shifting security left is important for also small teams and so on. So lots of people talk about that. But yeah, large the company, the more they have compliance things and so on ongoing. And then it is of course super helpful to move software engineering into the cloud. Because what happens currently is you have this nice pipeline, but then there is this black box where people just grab the code, copy it on some machine that is somewhere in the world and then there is this copy laying around and you don't know what this machine. You have to protect the whole machine, which is much more expensive and complicated than just access through HTTPS through a browser. Yeah, there is a really good, helpful lever there. And that's definitely also very, very interesting. And we are looking at that and it is important for us. But that said, we are – Our primary focus is really getting developers to have just a better time coding. Like that's the primary thing. We are really trying to build a product that developers love here. And then of course it needs to be secure, for sure.

[00:25:26] KP: Well, Gitpod is going to be useful to me whether I'm working remotely or on site. But of course a lot of us are working much more remotely these days. Does that impact the way Gitpod is deployed and used do you think?

[00:25:40] SE: It's very comfortable that you don't have to think too much about on what machine you are when you want to start coding on your project. So with Gitpod you always get the same beefy machine in the cloud no matter you are working from your Chromebook, or small laptop, or you are in the office and have have a beefy machine. So I think that's for sure very useful and flexible in that sense.

[00:26:14] KP: One of the reasons I carry around a laptop with a lot of memory and stuff in it is because I develop on that laptop. If I become a Gitpod power user, I could definitely see myself migrating towards more of a thin client kind of solution. I'm curious where you stand on that. What's your development environment like?

[00:26:32] SE: Personally I still use pretty powerful machines just because I have them, or I don't know. I like them. I work on a Macbook Pro. But I also have a Mac mini on which I work. I mean, with the Apple Silicon, it's also quite powerful actually. But I would not need

to. It's more like I think the interface is more important, having a proper keyboard and a proper touch pad and so on. Definitely, like I see lots of people who like Gitpod because they can now use smaller lighter machines. Especially when you're traveling a lot, it's just nice to not needing to carry around a big, heavy machine with you. So yeah, I mean, we see all this. But for me personally, like I don't do any local development for I think the last two years anymore. I don't think I even have Git installed on my computers.

[00:27:32] KP: Oh wow!

[00:27:33] SE: I do everything in Gitpod. Like we developed Gitpod for, I think, two and a half years now. And all other things are always in Gidpod.

[00:27:44] KP: Well, can you comment on the way your development process? Maybe your efficiency has improved in that time.

[00:27:52] SE: Yeah. I mean, two years ago, we didn't really have like KPIs around developer velocity or cycle time and so on. So it's a little hard to tell. But it's just like I don't have this uncomfortable worries that when I come back from vacation I need to update my dev environment, I need to catch up. Did any tools change and so on? Or I just want to work on something and then I get blocked because suddenly something doesn't work anymore. I mean, this happens with Gitpod as well, of course, at some. Point but then it's very visible because the whole team sees it and immediately takes action. And so it's not like every developer needs to figure that out on their own and so on over time. So it's much more aligned. And then just you know going out on GitHub and GitLab and you see some code. I can just spin it up in Gitpod and code around on it. That definitely has an impact on doing small changes. Like you are working on something and then there are an upstream library, there's a small issue. Usually, like there is so much to do in order to get a PR on that. You want to write a test and so on and you need to clone it and read the readme and set up everything. You don't do that. But if you have a GitHub setup for that, you just go there, click it, work on it and create a PR briefly on the side.

It's really no issue. And I think that's important also for Gitpod. It's cool if you are a power developer and you work on many projects. But it's also really cool if you are kind of part of

this group that supports developers in some way, like QA people, product people and so on that need a dev environment from time to time, because they are – Currently, with what the mainstream uses, they are kind of left behind. They have this intimidating hurdle that is setting up a dev environment. They just don't do it. And then they don't go as deep as they could. And that helps a lot.

[00:30:19] KP: Makes sense. Well, I see a strong use case for a development manager. Or as you were describing it, a bigger company where people just work on the platform, to hand a new team member an idealized environment. It's got everything pre-configured. Maybe there're some special tools the team tends to use or plug-ins they like. Those can all be pre-installed. Although at the same time I know some developers love to hyper-customize their environments and maybe they want to do the spacing or the colors and fonts in a certain way that's not agreeable for the whole team. Is there a story for me to individualize my environment?

[00:30:57] SE: Yeah, that's a question I um often hear actually. I think that for some reason, or of course it is important for a developer. You are spending hours every day in this environment and you want to adjust it to your personal taste and how you would be most efficient and so on. So you can do that in Gipod. Like you have this centralized configuration, which is Gitbase, Gitpod YAML, I mentioned that. Aand then you can have user space setting, like environment variables. You can have user level extensions, but also settings in your IDE and so on. And then we are also working on getting you cloned a .files repo so where you can also run and configure personalized kind of command line things. So that's basically what it is. You can think of it as you get the team experience and then you can put your user layer on top more or less.

[00:32:06] KP: Makes sense.

[00:32:07] SE: So, yeah, definitely important.

[00:32:09] KP: Can we go back to the web IDE? Remind listeners – We'd started out talking about when I spin up my environment, what tool set am I entered in and what configuration options do I have?

[00:32:23] SE: So the default uh mode is that you open up in a VS Code running in the browser. So that is the thing you – it's also open source. It's this open VS Code server that we released two weeks ago, and it is really stock VS Code. It's kind of the most upstream standard VS Code from what the Microsoft use code team is working on. And all we did there is just adding the possibility to run it in this browser remote server approach, which is the same thing that GitHub Codespaces does and Gitpod. And it's basically this protocol that we support there. And so you can run that as you want. So that's the default option.

And the other is you can go with your local VS Code or with – We are working on a JetBrains IDE integration, which is probably coming out later this year, early next year. So if you are a Java developer, probably you like IntelliJ a lot, and then you can just use that.

[00:33:36] KP: And would I then be using IntelliJ in the browser as well? Or what's the vision there?

[00:33:42] SE: I know. You would use the desktop version of it. So the desktop – JetBrains is building a remote development mode where the headless version of their IDE is running remotely. And then there is a protocol that communicates to a thin client running locally. So it feels like it is just the normal IDE, but the heavy stuff runs on the remote machine. Like the heavy stuff means indexing all the files and doing code completion, calculation and all that language specific stuff. That is running remotely. And then the stuff that needs to be visualized to the user, that's kind of done locally and the communication happens through a protocol. And Gitpod just integrates with this mode. And VS Code also has this mode already. And Gitpod also integrates with that. Like when you start your dev environment in Gitpod in a browser, you can, just with a single command, open your local VS Code on that same workspace.

And the difference is not big actually. It's the same VS Code more or less running there, but in the browser, there are certain key bindings actually. I think that's the main pain point here. There are certain key bindings that you cannot override in a browser. Very prominent is Command-W that is used to close tabs. So in VS Code you would like to close the active editor in the browser. That closes your browser tab, which is not nice, right? So there are

these smaller caveats. And that's the reason why people sometimes want to go local. I mean, I don't go local because I got used to that over the years to these programmings. But yeah, like, the browser IDEs, especially with VS Code, there is not much – Yeah, there's really no compromises anymore. Like it's just working really well.

[00:35:55] KP: Yeah. That Command-W is a very unfortunate collision, I think, across the two modes.

[00:36:01] SE: Yeah, that's true.

[00:36:04] KP: So I've never looked too under the hood at my development tools. So in my mind, at least, I think of something like JetBrains or IntelliJ as being you know a very different beast from VS Code. Maybe they're more similar than I think. But do you face any challenges in supporting these two seemingly orthogonal projects?

[00:36:26] SE: So, no, mostly because we built Gidpod very ide agnostic. Like what both tools do basically, they run in this container or connect to this container that is running in the cloud. And in that container we also have a main process running and a CLI. We built extensions for both tools, and they communicate with this process like through also messaging. And yeah, there is – Of course, VS Code is using Typescript as a programming language. And so we need to build it Typescript. And JetBrains use Java. So we need to write that in Java. The tools themselves in Gitpod are mostly written in Go Lang. So we have in that team that is taking care of this, we have already three programming languages that we need to use there. But, I mean, that's not a super big issue, I think.

And we try to keep the extensions, like this stuff, the sugar we add to the IDEs relatively thin and lean. We want people to, in the end, just get the experience they are used to. It's not like we need to – We think it's a good idea to put Gitpod in there all the time and surface that and rebrand or something like that. We don't do that. It's not important. It's not helping developers actually.

[00:37:56] KP: Should I be at all worried about latency? Let's assume I have a decent network connection, but it's still local versus cloud. Should a developer have a concern about that?

[00:38:06] SE: I mean, you should try it out and see for yourself. But there are certain things where you do a keystroke and you want to get a reaction immediately. And that is, of course, typing in the editor. But that is done entirely client-side no matter you use VS Code or JetBrains. So there's a document buffer that runs on the client side. So there's no difference. The terminal is different. There, you have this turnaround cycle with remote. So you send the key and it renders it and then it sends it back. VS Code has a mode where it renders locally. And then when it gets the message back from the server side, it tries to match it. And in 95% of the cases, it's just good.

So with terminal, sometimes you might see an effect. If your Internet connection is really bad, then it feels a little sluggish. You can turn on this mode and then you don't feel it at all. Content assist I think is with the thing that then comes to mind as well. That is something that usually happens on the server side. And so if you have a long ping time, it’s the most important thing here, or like cycle round trip time. Then, yeah, you might see it. But, really, compared to the usual calculation time that that content assist has in many scenarios, it's negligible. You don't really see it. It's 50 milliseconds or something like when it's bad. What's the round trip adds? So I don't see it. I don't feel it. But I’m also living in Europe and we have a Europe cluster. So we run clusters around the world and we are working on having them very close to you. Currently we'd only have America and Europe. So we have one in North America and one in Europe. And depending on where you are, you get connected to one or the other. But we are working on having smaller, more regional clusters. And then you really don't have any important latency anymore.

[00:40:23] KP: And broadly speaking, how does pricing work?

[00:40:28] SE: At the moment the pricing is you get a very generous premium tier where you can use 50 hours per month. That means running a workspace that is active for one hour is one workspace hour. And you get 50 of that. So you can spend quite some time in Gitpod before you reach that end. You can use that on public and private repositories. So

there are no limitations there. And then you can subscribe to user-based plans where you have more hours or even unlimited hours on a monthly basis. That's the current approach. We are currently thinking a bit about changing that to a usage-based pricing, but that's something for the future.

[00:41:19] KP: And are there any aspects of the roadmap you're comfortable sharing?

[00:41:23] SE: Yeah. I mean, we are working on – Like the container isolation is really important. So one big difference between Gitpod and GitHub Codespaces, GgitHub Codespaces is solving a very similar problem. So they have built out a really nice product in GitHub which is very similar to Gitpod. What they do is they run this in VMs. And VMs means you have more isolation, right? It's just very well isolated and you know that. But it also means the cloud density is not – Like most containers are VMs idling all the time. So it's super expansive and not efficient. Like if all developers on the planet would run their stuff on those VMs, we need a lot of machines and use a lot of energy for that.

And so we are using containers, which is more like a bit – It's a bit harder to get isolation right there. Put it like that. And so we already reached a few very important milestones. So we have an isolation, and that's something you should talk about with Chris also. There are really cool isolation layers so that we can allow root in the containers. You can do Docker in our containers and so on. And you cannot break out of the containers anymore. So that's a very unique feature of Gidpod actually. And we are doubling down on this stuff, making all these better so you can really run anything even also VMs in Gitpod containers. Other things are of course the better integration with the JetBrain that I already mentioned, the local companion app. Easier integration with existing environments in your network. We are building out the prebuild feature so that you also have incremental prebuilds working nicely, things like that. So there's lots of stuff in Gitpod.

And, of course, also integration with other platforms. Like you could also spin up dev environments from Jira. And currently we have only GitLab Bitbucket and GitHub. And yeah, making it easier to install the self-hosted version.

[00:43:45] KP: Well, Sven, is there anything you think we should have touched on that we didn't get to?

[00:43:50] SE: Well, probably, but nothing comes to mind.

[00:43:55] KP: Maybe to wrap up, can you remind listeners how they look into getting started with Gitpod?

[00:44:00] SE: Yeah, happy to do that. The easiest thing is to go to gitpod.io and, yeah, just get started from there. The alternative is go to your GitHub repository and prefix the URL with gitpod.io# and you get, in a few seconds, a full workspace with clone.repository and you can just start working on that. And then, of course, I mentioned that already. Gitpod is open source. So join the community. There is Gitpod IO/chat, which where we can chat. But also on GitHub, it's Gitpod-io/gitpod. Everything is open source. Just have a look there.

[00:44:42] KP: Well, Sven, thanks so much for taking the time to come on Software Engineering Daily.

[00:44:47] SE: Thank you, Kyle.

[END]