EPISODE 1148

[INTRODUCTION]

[00:00:00] JM: Development environments are brittle and hard to manage. They lack the kind of fungibility afforded by infrastructure as code. Gitpod is a company that allows developers to describe development environments as code to make them easier to work with and enabling a more streamlined GitOps workflow.

Johannes Landgraf and Sven Efftinge are creators of Gitpod and they join the show to discuss the product and the motivation for building it.

[INTERVIEW]

[00:00:32] JM: Sven, welcome to the show.

[00:00:33] SE: Thanks for having me.

[00:00:35] JM: Simple question; what is a development environment?

[00:00:39] SE: That's a good question. It is not an IDE, at least not as how we define it. For us, a development environment is everything you need in order to code, debug, be creative software projects. So that includes all the tools you would need like compilers, SDKs. I call that usually the userland. Of course, a decent editor or IDE, also with all the plugins, extensions you would need for the languages you are using. Stuff like application server, databases, and of course a source code. But also, you need to download dependencies, generate code and so on. So I usually call that inflating the raw source code somehow until you're finally about to start coding. And so that is what I call dev environment.

[00:01:37] JM: And what is the dev environment's interaction with the IDE?

[00:01:41] SE: That depends on the IDE, but today's IDE is like VS Code being the most popular one. It embraces a terminal. So will use a lot of command line tools. But then also you are able to open editors, multiple ones, and you have really decent tool support. And of course integrations for instance, a SQL plugin to connect to your test database, stuff like that.

[00:02:07] JM: Tell me the problems with traditional dev environments.

[00:02:11] SE: The main problem with how we all work today is that when you start with a new project, you're using your machine, whatever that is, MacOS, or Windows, or Linux, and then you need to make sure you can somehow run this project. In order to do that, you typically go to some readme where it said like, "Yeah, in order to do that, you have to install MySQL and that version, and you need node, and that, and that, and that, and that." And that is the description for fairly standardized simple projects like a node package. It usually works somewhat, but for real work projects, this is kind of an onboarding task that takes two days easily. And then even after that, it's still not perfect. So you're running into issues like debugging still doesn't work or stuff like that. That's only the onboarding.

The problem really is that we create all these state on our machines for all the projects we are working on, so we are usually working on multiple projects. And then we need to massage and sync overtime as a project evolves. And in case we need to go back for instance to an old branch in order to fix something, we need to rollback all the changes on our environment. And this always results in configuration thrift. And we've seen that with deployment scenarios, right?

Today, what we have is continuous integration, continuous deployment, and we have infrastructure as code. So no one today any reasonably professional with a software project deploy your software using an FTP upload as we did like 20 years ago and then updating it through that and massaging the state and trying to keep it alive. But we would write everything down in code. So it's reusable and reproducible and versioned. That's also important. And we're doing that with basically everything, but not with our development environments. It's still like we work in the 90s.

[00:04:23] JM: Again, what are the advantages to putting a development environment in the cloud?

[00:04:28] ES: Putting it in a cloud is not the key point. The key point is I want to – I don't want to care. I don't want to care where it runs. I just want to have it and use it when I need it. And I want to have it in the right state and it should be quick. And once I'm done, I want to forget about it. We call it disposable dev environments.

Cloud is just a really convenient infrastructure where you can do that. You could possibly do that on your local machine with Docker, but then you're still limited to the local resources you have. And we've all seen like everyone who has used Docker locally knows you pretty early hit this kind of problem that your computer is getting warm and loud. So these are the advantages using the cloud. You can just scale any kind of compute you need. And complex software systems today need a lot of compute.

[00:05:27] JM: So if I load my development environment in GitPod, which is the company that you are building, what happens?

[00:05:36] ES: When you want to start a dev environment with Gitpod, you start off from some sort of context in gitlab.com or in GitHub or in BitBucket. A context is a branch commit, an issue or a merge request, and what we've built is a system that you'd go off from this point and Gitpod creates you a dev environment for that particular branch. So it checks out that branch for you and prepares the dev environment. So exactly what I described before, the goal is to script everything. That brings you into what I call a ready to go development environment. So you don't have to do anything. You don't have to wait for builds. You don't have to wait for downloading and dependencies and such things. You just click on a button on a merge request and in a couple of seconds you have everything there and can debug the application and try it out and search the code and so on.

So you can do that today with Gitpod. You go to GitLab or GitHub on any project and you prefix your URL with Gitpod.io – hash. But of course not every project has a proper configuration. So we would just open, clone the project. When you have a proper configuration, Gitpod would use the right Docker image. It would use – And it would also execute the tasks that are specific for a project like Maven install for Java projects, or NPM, or yarn, whatever you have there.

And the really cool thing is we do that even before you open the project. So Gitpod has this unique notion of what we call prebuilds. That means you install a webhook on your project. And whenever there is a change, we go through these processes and prepare a dev environment. And then this completed dev environment is ready. The dev environment is sitting there waiting for someone to use it. And then you get fresh copies of that whenever you go there. And you don't have to wait for anything. So you're really skipping the 20-minute builds and so on.

And only with that, you can really have these disposable dev environments, because if you don't have that, of course you would go back to reusing one dev environment, because you don't want to wait for the build every time, right? You don't want to wait for downloading dependencies. With that, you go back to what we are using locally, just moving that into the cloud now. But that's really not very beneficial, because you still have to take care of the state. You have to maintain the dev environment all the time. And you are also not executing the automation multiple times a day. So you are not sure it actually works. And it probably doesn't after sometime, because it's not executed.

[00:08:38] JM: What are the advantages of having this kind of disposable dev environment?

[00:08:43] ES: I think for me personally, the best thing, we've been using Gitpod to develop Gitpod for more than a year, and it somehow changes how you work. I have this peace of mind that I always can have a dev environment no matter where I am, what device I use. But also, I don't have to switch – Like switching between tasks, which is something I do a lot, because I usually work on one feature, which is taking me 1, 2, 3 days or even longer. And then colleagues ask for code reviews. And I can just do that in a new tab with a new dev environment. I do the code review. I don't have to get stash and fetch and reset the branch and rebuild again and then come back or so. I can just do that on the side. When I'm done, I close the tab, and that's it. So this is freeing me from a lot of syncing, managing state, fractioning I have there.

Also, exploring projects, app stream projects. Often, I use, for instance, some sort of library. And then I have the source code, but it's minified. I want to use the original one. And then I go to the GitHub project, open that in Gitpod and can work with that and see. I can even make sure like easy fixes, like when I go to upstream projects even also locally in your company. These projects that are shared utility projects or so, you're working on that every other month or so. Small fix, but that's super easy to do. You just do that. You don't have to remember how to set up a dev environment. I think the best thing I just have this peace of mind that I don't have to care about the state on my local machine.

[00:10:40] JM: So are there some difficult problems with trying to make that state management work in Gitpod?

[00:10:48] SE: You mean describing how to set up a dev environment such that it is in the read to code to state?

[00:10:56] JM: Yes.

[00:10:58] SE: It is relatively simple. We are trying to reuse existing tools as much as possible. We are big fans of very orthogonal designs. We don't want to reinvent wheels when there are good solutions for the problem we have. So we are reusing for instance Docker and Docker files. So that is kind of the basis. There you describe what kind of feature or tools, command line tools and so on, you need. And then we have a Gitpod YAML, which you put into the root of your project similar to a continuous integration configuration. And it's actually all in all pretty similar. You describe what is the basis, Docker file, Docker image, but then also you're saying, "Okay, what are the steps that need to be executed in order to create me a ready to code development environment?" And those are just shell commands. And you can have multiple shell commands that run in parallel. So you basically say I want to have a terminal in my IDE. And these are the things that should be run during prebuild. And this is the task that should run when I start a dev environment. There you have all the freedom of the batch basically.

[00:12:11] JM: Do you have any like recipes for people setting up like the same defaults for how people would set up an environment?

[00:12:22] SE: We have a couple of base Docker image that are very powerful and have lots of tools. So good defaults, which work for 80% of the projects. Further, we have documentation tutorials, and of course on community.gitpod.io, there are users sharing their configurations. You can always also search on GitHub and GitLab for the Gitpod YAML files. You'll see great examples. Yeah, for certain frameworks, there are certain problem that you can always solve in the same way obviously. Yeah, there are these patterns.

But again, the more important thing is that we try to make it really, really close to how you would do that locally if you script it. We don't want solutions to be very specific to Gitpod. If that is required, we think we have made a wrong decision the design of the product.

[00:13:27] JM: So you have this analogy of CI/CD for dev environments. Can you explain that analogy in more detail?

[00:13:34] SE: Sure. Yeah. We see this as if you are working on a modern software project with the [inaudible 00:13:43] sort of Dev Ops pipeline, continuous integration, you're trying to streamline the flow from creating a change until it's in production. And also making this lead time very short, right? And there are these bits in the workflow where you need a dev environment, and those are not automated. Everything else is automated. You push something and then there's a CI and then there's a testing phase and so on. When you need a dev environment, you go to the readme and you need to create a clone locally and so on.

And Gitpod integrates into this pipeline and automatically creates dev environments for the branches and even for the task. So the first thing is you start off from the main branch. You have a prebuild for the main branch obviously. So you start a dev environment. Create a change. Then you push it back. Then there's this CI build, but also in parallel, a prebuild for Gitpod is created for that branch. So a reviewer can start a dev environment and do a deep code review. Provide some feedback. Probably not changing anything. So this prebuild is reused by the original author also to apply the feedback on the code. And yeah, basically, that's how we envision it. If you are keen on automation and want to streamline your process and your workflows, then fits right in in a Dev Ops pipeline.

[00:15:21] JM: How does collaboration work within Gitpod?

[00:15:25] SE: Collaboration works in three ways. First, of course, just the normal way using Git and so on, right? That's not different whether you work locally. You can even mix that. Use

Gitpod only for some parts. And you communicate through Git and pull request and merge request and things like that.

The second is you can live share your workspace. So if you are working on something and you have a certain situation that you want someone to look at, you can invite, like open your workspace to your colleague, then your colleague can call-in, so to say, and sees the same workspace that you have and help you there.

And the most, the best tool we have for collaboration, at least this is my personal opinion about that, is the snapshot feature. When I earlier explained how prebuilds work, it is you're running through a certain task. And then at the end, we take a snapshot of the result. And the user can take snapshots at any point. So you can create a certain situation. Try to reproduce it back. And once you have that, you create a snapshot. You get a link, and then you paste it into the issue. And someone else can click on that issue at any time in the future and get a copy, a clone of that situation.

[00:16:54] JM: Let's go a little bit deeper on how Gitpod actually changes how development is done in a team.

[00:17:04] SE: Yeah. It depends on how deep. As I said, like we use Gitpod entirely for everything. I think no one in our team who would know now like can tell me how to set up what I need to set up locally in order to Gitpod locally, you would figure it out after sometime, but we really don't do that.

In other teams, it always starts with one user who likes Gitpod, the idea of Gitpod, and they convince their team just to add the configuration. That's the only thing it needs to really use the full power of Gitpod. And the rest of the team doesn't necessarily have to use Gitpod. They can still use their local things. And also Gitpod, for instance, understands these code configurations. That is shared. And then some people use it, and then other start using it only for smaller tasks, like code reviews on the side that's quite convenient. What we've seen is that overtime understand, "Okay, there's really no reason for me to work locally anymore." And then they keep using Gitpod entirely.

The thing is this works for people who are very used to VS Code, because currently that's the only IDE interface we have in Gitpod, and fans of other IDEs. I think most importantly the JetBrains tools, because [inaudible 00:18:40] still works quite well. Yeah, so we are looking forward what JetBrains comes up with in order to work with remote dev environments. And then we are happy to integrate that. So our product, Gitpod, is really not about any opinionated IDE choice. We just pick VS Code or because that's the most popular one and the one we like.

[00:19:04] JM: I'd love to know more about the engineering behind Gitpod. So when you actually are building it, how does Gitpod run in the cloud?

[00:19:14] SE: It's a Kubernetes application running in basically two Kubernetes clusters. It could run in more. The Gitpod.io thing runs in multiple regions. So we do that in order to make sure that you have a very good ping time to your container so that you get instant – A really good timing when you tie them the terminal, because there's turnaround and so on, and content is really important to be snappy. But when you use the Gitpod self-hosted version, you deploy that to one Kubernetes cluster.

And there is a bunch of parts that we call the meta pods that do kind of RD application. And then we have a certain component that is managing the workspaces. We call that the workspace manager component. And that can actually run anywhere, also on other remote Kubernetes clusters or so that we could load-off the workload to other clusters. And that is monitoring the workspaces at runtime.

In Gitpod.io, we have some additional things running that are not part of the self-hosted, because it's not necessary there. We need to take care of security. Also, we need to make sure no one is abusing Gitpod and using it for cryptocurrency mining, stuff like that, or yeah. We have seen all kinds of that things happening there. Yeah, but that's basically the architecture.

[00:20:53] JM: How are people using it for cryptocurrency mining?

[00:20:57] SE: They're just using our freemium compute. They're just throwing their stuff there and trying to find something. I guess they never find something. But they are heating up the CPUs, and we don't like that. It's a classic actually. So I'm actually quite close to other cloud IDE people. I know the code and the people, for instance, for a long time. And they always also had this issue. So whenever there is some free compute available somewhere, people abuse it for this stuff.

[00:21:29] JM: How does Gitpod help with debugging?

[00:21:32] SE: It doesn't do anything special. It just uses a standard debugging tool that you would use locally. So what runs in the browser is VS Code like IDE that fully support a debug adapter protocol that was invented by the VS Code team. So you can install all the debug adapters, which are available for basically every programming language that supports debugging, visual debugging. So you can stab through code just as you would assume or you would do in VS Code. And you don't need to do any kind of remote debugging as long as the process is run in your container. But of course, you can also from your container connect to other resources. So when we develop Gitpod, we are not running the cluster in the workspace itself. What we do is when you push a branch, we instantly create a preview environment of the changes, of the branch and push that to a certain namespace. And when I start a dev environment, I get a dev environment with all the compiled source code and all the configuration and compilers, and a kub control CLI that is configured with this preview application. So I'm working with a real Kubernetes cluster that is external. And I don't have to wait for it to come up. And I also don't need to run a dev environment in order to just try it out if I want to do a review that is more based on testing. I don't need a dev environment.

Yeah. So in that case, we need remote debugging obviously, or sometimes we use Telepresence also for the debugging for those who know this tool. But it's really just like you would use work locally. There's no difference.

[00:23:21] JM: Tell me about snapshotting of Gitpod.

[00:23:24] SE: The snapshot feature at the moment is it's based on – The architecture in Gitpod under the hood is that we have a special Docker registry that would keep your state as layers on the layer file system that is used by Docker. And when you take a snapshot, we just backup the current top layer. And when you start to snapshot, you get that as an image and you put a new layer on top. It's very close to commits actually in Git.

So snapshots keep the state, the file system state. And it also keeps the UI layer or it reopens the editor you have opened and all the layout changes you did. It would just be recreated for you. And then of course when you start it, it starts just like a standard dev environment. So it would run the task. You have configured in your Gitpod YAML you want have started when you first started dev environment. Yeah, that's basically how they work.

Going forward, we would love to be able to also snapshot processes. This is a big advance tech also in container land, and there are solutions. Yeah. But also there are challenges with backing up all the state, because it can be quite a lot states and so on. But that's for the future. Currently, we back everything up, but not the state of the processes.

[00:24:58] JM: Is Gitpod useful for the classic problem of somebody new joins your company and then you need to onboard and then you just set up their workspace? Is it useful for that kind of situation?

[00:25:12] SE: Yeah, absolutely. I mean, that's the obvious first thing, the onboarding problem. I'm just not focusing on that so much, because we've learned and experienced how cool it is to work with disposable dev environments continuously. Have this freedom of always automated dev environments. But when I talk to people and ask about their pain points that they have now, they for sure realize onboarding is a big pain point. And yeah, Gitpod eliminates that. So there's no two days of trying to get it to work. It's just works.

Interestingly, sometimes people say, "Well, I think it is important that my developers go through this pain and learn how to set up all the tools and so on." I never really understood even when I asked why that is important. It's somehow weird. But maybe it was more like in the past when developers also needed to install production system or so. I don't know.

[00:26:22] JM: There's a feature of Gitpod called parallel workspaces. Can you explain what a parallel workspace is?

[00:26:28] SE: That's just starting another workspace on the side that we try to explain. Because that's something you cannot do today, although developers have their workaround. So some developers keep having two Git clones for instance in two different folders in order to use one for the ongoing feature development. And another for doing code reviews and so on, because they hate this friction back and forth. And then you have at least dual workspaces. In Gitpod you have any number. Today I think there's a limit, but it's a very high limit. It doesn't make sense anyhow. You can just start another dev environment and when you need that, use it, close it, and it doesn't affect anything. You'd be working on it as dev environments.

[00:27:21] JM: What's your vision for the future with Gitpod?

[00:27:24] SE: I'm pretty sure this is going to be very normal, like the idea of having automated development environments. In a few years, it will be like today not having continuous integration, or 5 years ago not having version control. Yeah, it's going to happen. And the tech is there. I'm honestly looking forward to the launch of GitHub code spaces. They will help us a lot getting that across to the mainstream, also because this time the solutions are really good. In the past, these solutions were very ambitious, but really couldn't match the local dev environments. For instance, Cloud9 has been around, has done a lot of cool things, pioneered this area. But also, like I mentioned, Codenvy, Eclipse Che and so on. But now we have solved, for instance, this problem that the IDE you are running in the browser is now really more or less the same thing you are running locally, or many people run locally. And we have really good cloud technology. Yeah, there are no more so many reasons to do everything locally, because it's also powerful and you can solve all the problems and the benefits are so good.

So yeah, this is going to going to happen in a couple of years. And for Gitpod, yeah, we are in a position where we have created something in a good time and we love what we've been building. We are pretty sure also our prebuild stuff is going to be landing in other products that compete with us. Yeah, that's exciting. So I really want to just work on that and see how it solves the big problem in the development world.

[00:29:27] JM: Any other deep engineering problems that are worth exploring?

[00:29:32] SE: Yeah. So there is one deep engineering problem we are actually working on at the moment. And that is Gitpod uses containers. And with containers, there is this problem that you cannot allow people privileged access of pseudo and Docker and Docker and so on, because they could use system calls and breakout the container isolation. And then they are on the node where other containers run, and that is of course unacceptable. And because of that, we need to enforce limitations. And those limitations are – Yeah, they force people to build, like workaround in a configuration. The configurations could be much simpler.

For instance, a lot of Docker images that are like out in the wild, they are built with base [inaudible 00:30:31] basically. And so you cannot directly run them in Gitpod. But we want to allow everyone to just reuse stuff and make it as simple as possible. So we are looking deeply into this problem, which hasn't been – Like there are others looking into that deeply, like Google for instance with gVisor and there are others. And so this is really very, very big problem.

We think we can solve it for us. We cannot solve it generally for everyone, but we know what are the requirements and what is the context in Gitpod. And therefore we can make certain decisions. But for that, we really need to dive very, very deep and understand the Linux kernel and what system calls have – What possibilities and so on. We are working with our engineering team who are pretty good with this, but also with external consultants and experts in Kubernetes and container security and so on on this. That's the hardest, yeah, technical challenge we are working on at the moment.

[00:31:38] JM: How do you vet Gitpod for security?

[00:31:41] SE: Currently, we, for instance, don't allow any privilege or pseudo or root in the containers. Of course, all connections are HTTPS. Then all important information is stored, encrypted. I think – Yeah, this is mostly the common stuff. Of course, you're also generally always trying to be more beyond the safer side and trying to do things that way. You don't want to have this fire back at you. Yeah, but those are the things we are doing. And we are more working on being sure we can relax one or the other things in order to allow certain features to use. There is no reason not to use HTTPS for sure, but what I just said, the root access if really problematic.

[00:32:41] JM: Well, Sven, is there anything else like you'd like to explore around Gitpod?

[00:32:46] SE: Yes, I woke up to great news today that GitLab merged - a merge request we were creating together with them, that integrates a native Gitpod pattern on the GitLab project. So when you in a few days should land on gitlab.com and I think October 22nd is the next, or September 22nd is the next release for Gitlab self-manage. And I'm not sure if it's landing there or in October 22nd. But you no longer need our browser extension in order to have a Gitpod pattern on gitlab.com. It's going to be natively integrated now. And we are continuing working with making a really great integration in those directions so that you have really a seamless workflow when you use Gitlab with Gitlab CI and so on and you can just use Gitpod in order to start dev environments on your projects.

[00:33:44] JM: Cool. Well, thanks, Sven. Thanks for coming on the show. It's been real pleasure talking to you.

[00:33:47] SE: Thank you, Jeff. Have a great day. Bye-bye.

[END]