EPISODE 19

[INTRODUCTION]

[0:00:00.5] JM: On January 31st, 2017 GitLab experienced a major outage of their online repository hosting service. The primary database server experienced data loss due to a combination of malicious spam attacks and engineering mistakes that occurred while trying to respond to this spam attacks.

GitLab responded to the event transparently. The company setup a post mortem describing the event in detail. In subsequent posts, GitLab expressed sympathy for the employee who made engineering mistakes that led to the deletion of data. The employee was not judged or disciplined for an error that was quite understandable. The response from the developer community was very positive.

Engineers know that building cloud services is hard and engineering is as much about avoiding errors as it is about appropriately responding to the mistakes that inevitably happen. GitLab is a developer platform that combines repository hosting with several other features such as issue tracking and code review and CD.

Today's guest is Pablo Carranza who works on infrastructure at GitLab and in this episode, he walks us through GitLab's product, the engineering stack and the post mortem of the outage. We also discuss working in Amazon and the importance of post mortems and culture which I found quite an enjoyable conversation because I encountered this things at Amazon myself.

It was great conversation, I enjoyed talking to Pablo.

[SPONSOR MESSAGE]

[0:01:44.5] JM: For more than 30 years, DNS has been one of the fundamental protocols of the internet. Yet, despite its accepted importance, it has never quite gotten the due that it deserves. Today's dynamic applications, hybrid clouds and volatile internet, demand that you rethink the strategic value and importance of your DNS choices.

Oracle Dyn provides DNS that is as dynamic and intelligent as your applications. Dyn DNS gets your users to the right cloud service, the right CDN, or the right datacenter using intelligent response to steer traffic based on business policies as well as real time internet conditions, like the security and the performance of the network path.

Dyn maps all internet pathways every 24 seconds via more than 500 million traceroutes. This is the equivalent of seven light years of distance, or 1.7 billion times around the circumference of the earth. With over 10 years of experience supporting the likes of Netflix, Twitter, Zappos, Etsy, and Salesforce, Dyn can scale to meet the demand of the largest web applications.

Get started with a free 30-day trial for your application by going to dyn.com/sedaily. After the free trial, Dyn's developer plans start at just $7 a month for world-class DNS. Rethink DNS, go to dyn.com/sedaily to learn more and get your free trial of Dyn DNS.

[INTERVIEW]

[0:03:40.0] JM: Pablo Corranza is an infrastructure lead at GitLab. Pablo, welcome to software engineering daily.

[0:03:44.4] PC: Hi, thanks.

[0:03:45.2] JM: GitLab is a developer platform, it combines a Git repository with a bunch of other features, describe the GitLab product?

[0:03:53.6] PC: GitLab is basically a Git repository management with Wiki's with Asia tracking, with CICD pipeline, monitoring pages and specifically the whole development, infrastructure you're going to need that you can install on premises.

[0:04:12.2] JM: This was a challenge that other people had worked on prior to GitLab. What was the unsolved set of problems in the repository management, the wiki'ing space that GitLab has been working on?

[0:04:29.7] PC: Honestly, it was started in 2011 by Dimitri. I don't know exactly what he was looking for but in general, he didn't find something that was useful enough for him that he can run on premises. Back at that time, there were a couple of solutions but these solutions were each one, you had to build your own right? You had to build your own repo management, you need to build your own easy tracker, you had to basically get all these pieces that were separated and join them all together to get something that was covering all the needs.

GitLab offers the whole package, you just install one package and you have it all.

[0:05:15.8] JM: Why do people want to run their git hosting and their CICD stuff et cetera on premises?

[0:05:22.0] PC: It depends on the company right? I mean, you have companies that cannot use the cloud for example, you have companies that want to have everything in house right? Also, you have the issue of what happens if your service goes down, you cannot actually perform any work right? There are many reasons.

[0:05:44.1] JM: Right. Yeah, if I'm just like a rogue developer and I like to throw everything in the cloud, can I still use GitLab?

[0:05:52.7] PC: Yeah, there's one more thing I wanted to say and it's that…

[0:05:55.4] JM: Sure.

[0:05:56.0] PC: Git is also additional system right? The idea of Git is that you can have multiple repos all around the world right? Each client can have its own repo. Concentrating everything in a single place, it's kind of an anti-pattern for that.

[0:06:12.6] JM: Right, Yeah. What is the typical strategy that a distributed company that is using GitLab with a bunch of repos in different places? What is their strategy for repository management?

[0:06:26.9] PC: You can have a single instance if you want, that's going to be fine, you can have it locally, you can also use mirroring if you want to have multiple places and there's another product we're working on which is GEO which allows you to have a closer copies of your repo, of your whole infrastructure actually.

[0:06:46.7] JM: Okay, let's first talk about if I have a GitLab deployment and my entire company is entirely in one place, what is the architecture for the GitLab application that I'm deploying on premises?

[0:07:00.5] PC: Okay, it's kind of simple, it's just a rails app which is going to be using a database namely Pause Swiss Reddit for running psychic and then a local file system right? That's quite simple. There are a couple of components like for example, workhorse that's ready to go, there are a couple of things that are a little bit specialized right now but in general, a single installation of GitLab is quite simple.

[0:07:30.3] JM: I sere, if I have a single deployment, is there any relationship between my on premise deployment and the GitLab product, the GitLab infrastructure that's in the cloud, like under the GitLab company?

[0:07:45.4] PC: The GitLab installation, the GitLab.com, the website, that's built using the same package we ship to our customers. It's an instance of GitLab E, with the caveat that we have a larger infrastructure, we have multiple instances running different parts of the system and we are interconnecting them. In general, that's pretty much it. The main difference is probably the scale and the solutions we need to use to reach that scale.

[0:08:14.4] JM: Can you talk more about what is under the GitLab, your own — the company's infrastructure, the company's deployment of GitLab because it does have a lot of scale but if everybody's hosting their stuff on premise, in their own deployments of GitLab, what exactly is being stored in your own GitLab deployment?

[0:08:36.6] PC: Our GitLab deployment in the end is like any other GitLab installation right? What we're storing there is basically git repos and we're also installing all the meta data right?

For example, issues, all the comments on your issues, et cetera right? All this data that you're using for your daily work, right?

For example, if you push your co, you're going to create a merit request, you're going to have comments in it, you're going to run your CICD pipeline, you can have your own personal runner for running that but you can store all the information, all the meta data from this work and your repo in GitLab.com.

[0:09:16.0] JM: Even though I have my repository on premise, if I'm like a medical company and I have to host my own installation of GitLab, the GitLab.com can host the metadata like the issue tracking, is that what you're saying?

[0:09:32.9] PC: What I'm saying is that, GitLab, the package, it's a full application. You can have your meta data, you can have your issues, you can have your merit request, you can have your code, you can have everything on premises and you're completely detached from GitLab.com.

[0:09:48.8] JM: Right.

[0:09:49.3] PC: GitLab.com is another instance.

[0:09:52.0] JM: Okay, all right. So when you're talking about all the data that's being stored in the GitLab.com and the GitLab instance. You're just referring to meta data about GitLab itself, the GitLab product, the data that the company is managing.

[0:10:07.6] PC: Let's say you have a repo right? Let's say you create a project, you have this remote repo from the git perspective right? Where you can push and you can pull. Classical kit operations, you code something, you commit, you push. You push to GitLab.com all right?

Now, you create an issue because you want to have something done, you create that in GitLab.com, you can have the whole development cycle in GitLab.com right? Now let's say that for some reason you want to run it yourself, you install the package and you have your own personal GitLab.com, it's completely detached.

[0:10:44.0] JM: I see, I can choose whether to sync it with GitLab.com if I want or is there a way to sync it to run everything on my computer, my own local computer with all my issues and stuff and then sync it later on with the GitLab cloud?

[0:10:59.7] PC: You can import from a different instance, there's mirroring you can do that is going to affect only the repos.

[0:11:11.1] JM: Let's talk about the GitLab infrastructure itself because if you are — you've got a product that can host, could basically host anybody just like a remote get repository management system but it can also be hosted on premise but I guess we should talk about the larger cloud instance, the cloud deployment of GitLab itself.

What's the infrastructure or what are the unique scalability challenges of building a service that's going to manage lots and lots of user repositories?

[0:11:48.7] PC: The main challenge for us has been for quite some time, storage unsurprisingly enough. The thing is that we're storing a lot of data lately in between repos, I think we have something like 76 terabytes of git repos and we have something like another 50 terabytes of regular files like uploads, logs, et cetera.

Git particularly is extremely expensive both from the CPU perspective and the IOPS perspective and also there's the problem of we just keep getting more data which means that we need to scale horizontally. That's one of the main challenges we have. The second challenge we had was with the database itself. Again, same issue, we're just growing a lot.

[0:12:36.2] JM: What are some of the architectural philosophies of GitLab that you employ as the product is growing?

[0:12:41.8] PC: GitLab started being a rails application which is basically that it was a monolith and the problem with that is that at some point, it's great for building a product but the issue is that at some point it just doesn't scale anymore. The philosophies we're using now is that we're trying to have a data driven approach to understand exactly what the problem is and what we

need to do to fix it which means that we do a lot of data gathering and then we do a lot of analysis to understand what the bottle neck is right?

For example, on the storage case, we try to solve that first with a little bit of a brute force attack using the history of the file system. It just didn't work, we analyzed the data, we decided to go a different path which was to extract git and build basically a git as a service per se.

That's probably called git led because of the way it behaves. This is pretty much the same, what I mean here is that you need to adopt, you need to learn, you need to try things and you need to be around soon and then you need to iterate.

As you iterate, you're going to be learning more and as you learn, you're going to be adjusting your path.

[SPONSOR MESSAGE]

[0:13:57.5] JM: Your application sits on layers of dynamic infrastructure and supporting services. Datadog brings you visibility into every part of your infrastructure, plus, APM for monitoring your application's performance. Dashboarding, collaboration tools, and alerts let you develop your own workflow for observability and incident response.

Datadog integrates seamlessly with all of your apps and systems; from Slack, to Amazon web services, so you can get visibility in minutes. Go to softwareengineeringdaily.com/datadog to get started with Datadog and get a free t-shirt. With full observability, distributed tracing, and customizable visualizations, Datadog is loved and trusted by thousands of enterprises including Salesforce, PagerDuty, and Zendesk.

If you haven't tried Datadog at your company or on your side project, go to softwareengineeringdaily.com/datadog to support Software Engineering Daily and get a free t-shirt. Our deepest thanks to Datadog for being a new sponsor of Software Engineering Daily, it is only with the help of sponsors like you that this show is successful. Thanks again.

[INTERVIEW CONTINUED]

[0:15:21.8] JM: I know a number of shops where when their rails application begins to creak, they move to J Ruby so they start to be on java, is that a strategy you guys have considered?

[0:15:36.8] PC: Not really, we do use a set of native gems which doesn't allow us to start doing J Ruby. Also, it's not so much about the language, I don't think that Ruby is to blame here. I think the issue here is that when you're building a product, you are trying to deliver the most value to the customer, the sooner the better.

That's going to push you to do things in a way that you're not considering scales soon. As you grow, you start seeing where the bottle necks are. The bottle necks are not about the language itself, the bottle necks are architectural issues. Not by moving to J Ruby, not by running a faster card, you're going to be running faster in the same road.

[0:16:23.4] JM: there is of course the classic premature optimization criticism people will make where if you say, if we build this way, we'll be able to scale to thousands and thousands of users and then you'll very quickly be shot down and say that's a premature optimization.

Do you think that holds true for the GitLab product or do you sometimes look around the architecture and say I wish that people would have thought more long term about this.

[0:16:54.0] PC: It's tricky. The way you can remove the judgment of premature optimization is by having data. When you have data, the conversation changes a lot and I think that that's the path we're following now, we're gathering data, we're trying to understand exactly what the bottle neck is.

Because multiple times, the problem with premature optimization is that you will say, I think that the problem is this, therefore if we do this or that, we're going to be running much faster. That's an assumption. Now, what data do you have to actually prove that assumption?

That's when the game changes, if you have an assumption and you have an experiment, you go and run your experiment, you gather the data and then you can actually make an informed decision. With that data, you know that reality actually is this way. Therefore you can start

adjusting and you can start planning in a way that is going to be easier and it's going to be actually be much more effective, much more cost effective right?

From the I wish people would have thought about X. Again, I think that by the time the product was being built, the goal was to ship a product and I think that we succeeded there.

[0:18:09.5] JM: You and I have both worked at Amazon, we could talk about that more later but this brings to mind the story of the S3 team. I don't know if you know the story but when the Amazon S3 team was presenting to Bases and say okay, here is our product and Bases said that if this works, if it has any amount of success, the product will fall over and therefore it will not be a success and that will not work because its core infrastructure.

The team went back to the drawing board, they spent like another year or six months or something, reconfiguring it. I always found that to be an interesting story because a lot of people will just say, yeah, let's ship it and they wouldn't prematurely optimize but it's sort of an example where it seems like bases might have been right. You actually do want to prematurely optimize for S3 because you can't lose trust on a product like that.

[0:19:05.4] PC: Also, you have to think that the case of Bases is that they were already in a large scale. Bases were seeing the growth, they were having, he was actually challenging the this was not premature optimization, this is required, this is a high requirement.

[0:19:22.9] JM: Do they have EC2 already at that point or are you talking about the Amazon.com marketplace.

[0:19:27.0] PC: The amazon.com marketplace in general. The growth of it was insane. I mean, the numbers they're running is insane.

[0:19:34.2] JM: Yeah, I don't remember, was S3, were they originally thinking like this is a product that we're just going to dog food internally first and then deploy?

[0:19:43.4] PC: I don't know about that.

[0:19:44.8] JM: Okay, all right. Anyway. GitLab is mostly hosted on Azure and I've talked to a lot of people about AWS, I've talked to a lot of people about GCP. I'm going to Microsoft Build next week and I'm eager to learn a lot about Azure because as I understand, the cloud rankings, AWS has the most market share, Azure has the second most and Google has the third most and the fact that I haven't done any recording on Azure despite the fact that it's the second biggest cloud provider.

I think removes some of my credibility as a software engineering reporter. What's been your experience with Azure?

[0:20:23.7] PC: It's been challenging. Basically because we started being a small company and Azure I think they weren't really ready to deal with startups and they're learning through the process, we're learning with them too. Also, I do see that they are going through a process in which they are following the steps of AWS a little bit.

For example, not so long ago, they are building this new environment which is called ARM, Age Resource Management. They are making us jump over the fence to this new resource management infrastructure which is something that OBS did years ago right?

You can see that they're following kind of the same steps. Also, I do think that if they have the second largest share, that's going to be because they're the main people who run windows basically but we run Linux. Which sometimes ends up being that the infrastructure is not really friendly to Linux sometimes.

It's improving a lot and they have really great support people from the Linux side but you can see that it's a learning process, you can see that we're working together, we're improving the lines of communication day to day to actually get into a better shape because it was really challenging for us.

[0:21:45.4] JM: I don't really understand when you're talking about they're forcing you to make this migration, what does that mean and at what layer of the stack are you having to make some changes?

[0:21:55.2] PC: No, it's not about the layer of the stack. From my perspective it's just a hypervisor and that's it. I understand that the changes are particularly internal to their network as in they had a lot of learning in the classic environment and with that, they're building a new environment which is probably much more cost effective but I don't have the data to understand exactly why they need this change to happen.

What is happening is that they are deprecating all the function from the classic environment and that's basically making this move to the new environment.

[0:22:29.7] JM: Does that lead to some outages?

[0:22:32.9] PC: Well, that leads to us having to boot machines on the other side. On the particular case of we using NFS as a storage system, yes, that led us to take down time.

[0:22:44.4] JM: I see. Speaking of that down time management process. Talk about the monitoring stack, because you're an infrastructure, you spend a lot of time looking at the monitoring and deployment process, describe the monitoring pipeline.

[0:22:57.6] PC: Yeah. We use Prometheus extensively, we actually have a Prometheus team and we ship it as part of the product because we really believe in that monitoring stack. In general, the way it works is that we have a couple of servers, we have two servers for HA and we have another server which is public.

These servers are pulling data from the different host, from exporters. This exporters provide the metrics for different things, system on metrics, pause specific, you name it. This data exists in this Prometheus servers which you can query which is highly valuable for us and we also have a graphani instance which is showing a set of dashboards.

One thing we did because of the open nature of GitLab is that we have a private one which is HA and that's the one we use internally and we're basically modifying it all the time. We have a public one which is a replicated version of the private one.

[0:23:58.5] JM: What is new about Prometheus because I did a couple of shows about it and people talk about how important Prometheus is as a monitoring tool, explain what Prometheus does differently?

[0:24:11.9] PC: You come from Amazon so you know the monitoring tools they have in right? In the outside world so to speak, when I join the company, we had a nagios installation. The problem with an nagios installations or the big issue for me with those kind of systems is that their host based. They're great for monitoring hosts individually.

When you start using something like Prometheus, you're going to start aggregating a lot of metrics. You can query those metrics later and you can particularly start grouping them by service. You can start seeing how the oldest, this hosts are behaving at the same time and you can start correlating different data.

Particularly, the really useful thing is that you can do that after the fact right? You don't need to set new checks to see what is going on there, you already have a lot of data and you can go and play with the data afterwards. You can do discovery of how the systems are behaving after whatever happened right?

That is extremely powerful. It's extremely powerful because let's say you have an outage or you have a degradation of the system and you have no clue what's going on, then you can go and start quarrying the information that you already have available and then you can discover things that are going to explain all the metrics you're seeing.

[0:25:36.0] JM: How does a deployment work at GitLab?

[0:25:38.6] PC: Okay, a deployment in our infrastructure, one thing I didn't mention is that we have something like 40 hosts in the front end, deployment right now, we use the same package we ship, we use GitLab E, the package and the way it works is that we deploy, we basically perform a pseudo app get install of the package in a specific host that is not getting any traffic, we run the abrasions there and then we perform a rolling install through the rest of the hosts.

That's in a nutshell how deployment works. It can be improved because the package is large and it takes time to un-package but in general, we're doing the same thing our customers are doing, the reason behind it is that we ship a package, this package has to work for our customers.

[0:26:26.8] JM: I want to get into this incident that happened back in January where there was a major production event but — I think it's a great case study in how to handle an incident like this and how to make — basically, how to build a productive team around responding to this kinds of events. I guess, before we dive into that. Talk a little bit about the process of on call and what the responsibilities of the team are in keeping the up time of GitLab and monitoring it for issues.

How does on call rotation work and what are the responsibilities of somebody who is on call?

[0:27:11.2] PC: So, the on call we perform is 12 hours, it's a one week shift 12 hours per day because you need to sleep and the duties while on call is first of all that you will be the first person who is going to reply to an alert, whatever that alert is. You also need to investigate whenever something, whatever something happens to the system, you have to be using the monitoring tools to see the health of the system and you need to leave a same trace of what happened if anything happened.

For that we have the run books for example which are completely public. The way alert work is that whenever an alert is triggered, it will point to the dashboard in monitoring, it will also point to the ram book that you need to execute.

It's your duty as an on call person to handle the problem if it is a problem of course, update the documentation if it's needed and in case you don't know how to handle, escalate.

[0:28:14.4] JM: What is that term run book mean?

[0:28:16.8] PC: Run books are a project, it's an open source project where it started when I joined the company, it was a way of reverse engineering the tribal knowledge. Because I was just joining the company, I didn't know how to do a lot of things and what I did was I started writing down every action that was being performed, it was a way of generating documentation.

Then I started pushing for you start as a way of handling the outages and we also added alerts. Both the alerts and the run books live in the same project because they go hand in hand.

[0:28:58.0] JM: If I understand how a run book works correctly, let's say you have an event where our log servers are running out of disk space and you look up that issue in the run book and say okay, if the log servers are running out of disk space, first, check the number of other hosts that you might be able to divert traffic to and then look at how much space those hosts have.

Then consider redirecting some traffic to them. It's basically like a list of steps, it's like a recipe for responding to an adverse event. Is that correct?

[0:29:34.0] PC: Yeah, it's correct. The basic structure is a pre check where you're going to say you get an alert or something, you will have a way of checking that you're doing the right thing. A set of checks to perform and then a possible resolution and then a post check and in some cases even a rollback plan. The idea is that — under stress — you don't want to think.

[0:29:57.9] JM: Right and I think don't run books also service a decent path to automating some of your infrastructure in the future because you might have a run book that describes something, just because somebody has not written the right scripts that can automate that response.

[0:30:13.6] PC: Yeah, that's another plus of this is that you are actually codifying your reaction to a problem which means that you can afterwards automate it. Once you have a run book that is solid and is not going to be changing, you can actually have it automated perfectly fine.

[0:30:31.0] JM: Back in January, there was this incident where a database administrator at GitLab deleted a production database and this was a result of a series of events. It started with spammers hammering the GitLab database by creating little snippets. Explain what was going on?

[0:30:50.5] PC: Yeah, multiple things were happening. One of the first thing that happened that we discovered afterwards was that we were having a lot of spammers who were hammering the

API to create these snippets. These snippets are used as public snippets to basically gain pay trunk. The issue was that in the middle of this mess, one of our own employees was marked as a spammer. So it was marked to be deleted.

These added a lot of load to the database because it's not so easy to delete someone who's in the guts of the database. There's a lot of issues, there's a lot of things. So this process was failing and was generating the topples at the same time, we have the spammers who were creating new snippets when we were in the middle of blocking this IP, removing the snippets, etcetera that added a lot of load to the database. There was a lot of load being generated because of all these situation.

So what happened was that the secondary database lost the replication. The reason why that happens, what had happened was that we were running post-rescue L and we had it set up to follow replication using the wall segments which is the right ahead log and the primary database was writing wall segments way too fast for the secondary database to follow it and at some point, the primary just dropped old wall segments and the secondary was not able to keep up and just dropped replication completely.

We were in the effort of recovering replication and that's exactly when this confusion happened and we basically deleted the database that was the whole thing. One thing to mention here is that these databases, this is only about the GitLab.com the instance, going back to the beginning of the conversation none of the all premises customer had any form of data loss. This was only about GitLab.com and about mirror data. What we loss there was some issues and merch request. No repo was affected whatsoever.

[0:33:10.5] JM: Can you describe more detail of the atmosphere in which the accidental deletion occurred? Because there was a number of things going on and also this team — there was a great write up about this. I will put it in the show notes but this person who was anonymized as team member one, they were somebody who was tired. It was at the end of the day and it was just a really unfortunate intersection of timing and these spammers. It was just really bad luck but describe why was this able to occur? Do you think this was a preventable intersection of events?

[0:33:55.2] PC: I think it is preventable. In all honestly of course hindsight is 20-20 right? The person was tired and also because we were dealing with spammers and it was a little bit of all hands on deck because there was a lot of noise going on and the atmosphere was just, "What can we do?" it was ongoing. It was an ongoing thing that we were all working in and nothing else than that. It was preventable from the perspective of there was no safety measurements that prevented this from happening.

This is something that is easy to say afterwards but in general, this could have been avoided with a good measurement, with a good automation. Just with that it would have been enough. The thing is that…

[0:34:54.2] JM: What kind of automation?

[0:34:55.0] PC: Sorry?

[0:34:56.0] JM: Like what kind of automation?

[0:34:58.1] PC: Well if you only have a simply really solid script that is going to help you recover the replication which we had but it was not good enough, only with that you don't need to perform any money election. The problem with the RAM books and I think you know this probably from Amazon is that human error is a common thing. It happens, we're tired, we make mistakes.

[0:35:22.5] JM: Can you describe a little more in detail what caused team member one to accidentally delete the database and then how did they realized that, "Oops I made a mistake" and then what did they do after they realized they made a mistake?

[0:35:36.6] PC: So this person was executing, trying to recover replication and was SSH into both databases. It was just a confusion.

[0:35:46.9] JM: Right and so what did they do after they realized that they had deleted it? What was the path to recovering from this event?

[0:35:57.9] PC: Okay, so what happened afterwards was first this person commented in Slack that I made a huge mistake and was tired and the rest of the team, what happened immediately was, “Oh okay so what can we do?” and all the team started helping immediately to forge a plan to move forward. That’s when we started testing, we started basically trying to recover from backups. They weren’t working and we all started basically trying to find the way to move forward.

[0:36:29.7] JM: Yeah, I know that it sounds like the lack of blame, nobody blaming team member one was really a valuable reflection of the GitLab culture.

[0:36:42.7] PC: Yes, blame is useless. If you focus on blame what you’re going to be doing is you are going to be stopping at that point. You are not going to be focusing on solving the problem. What you need to focus on at that point is in solving the problem.

[0:36:58.4] JM: Now some customers did loose metadata to this but all of the customers are developers. Were the developers who were affected were they sympathetic?

[0:37:09.7] PC: Yep, I think that was an interesting thing about the openness. When we started trying to find a way to moving forward, we basically got together in a boardroom in a media conference. We started the documents where we rolled the timeline of events to understand exactly what happened and what can we do and we proposed a lot of ideas in there right? Immediately we opened this document for anyone to see and we tweeted this is what happened.

So the service is down because we made a mistake, you can find more about it in these document. So people started jumping in and we firstly performed a series of debugging of exactly what the path is forward and by the moment that we had a path forward, the team member one basically he was too tired, he stepped out so another team member took the lead on moving forward and when we find a way of moving forward, all we had to do was basically wait to get to the situation where we can restore the data.

We were pulling the data from a different data center and that was going to take a while so at that point, I think it was Sid who said that we could basically open this and start streaming it. We

said, “Yeah sure” it’s one of the things about GitLab culture is that it is open by default so it doesn’t make a difference to us to work completely in the open.

[0:38:45.8] JM: How far does GitLab take that openness as a default? Because I love when companies are extremely open but a company like Apple for example probably could not be extremely open because a lot of the value of their product is the surprise of, “Oh we suddenly released this new augmented reality set of glasses” and you’re blown away by it. Do you think that different companies can be different degrees of open?

[0:39:15.9] PC: I think so. GitLab particularly, we developed an open core product where you can read all the code. We have open issue trackers you can go and read what we are doing. Of course there is a point where things become private as in, I don’t know, you were talking about a specific person. If we’re going to be compromising security of course that is not going to be in the open and in general everything is open by default. There has to be a good reason to make it closed.

I think it’s because of the nature of the company. Different companies are going to be working differently and that’s perfectly fine.

[SPONSOR MESSAGE]

[0:40:03.8] JM: Artificial intelligence is dramatically evolving the way that our world works, and to make AI easier and faster, we need new kinds of hardware and software, which is why Intel acquired Nervana Systems and its platform for deep learning.

Intel Nervana is hiring engineers to help develop a full stack for AI from chip design to software frameworks. Go to softwareengineeringdaily.com/intel to apply for an opening on the team. To learn more about the company, check out the interviews that I’ve conducted with its engineers. Those are also available at softwareengineeringdaily.com/intel. Come build the future with Intel Nervana. Go to softwareengineeringdaily.com/intel to apply now.

[INTERVIEW CONTINUED]

[0:40:54.1] JM: After this event, GitLab did a postmortem and I remember having postmortems at Amazon. They were really important because after an event like this, everybody has some adrenalin, their eyes are wide and I think people are particularly engaged. So it's actually a really good time to pull strategic elements out of an event and be able to integrate them into your culture for later on. Talk about the importance of postmortems and how the postmortem from this event, from the production database outage, how did that affect GitLab going forward?

[0:41:41.4] PC: So we used postmortem, what you are talking about is the COE process in AWS in Amazon which is the correction of error. I think it's a great name by the way because what it tries to do is tries to understand exactly the root cause why this happened and you use that as an exercise to understand what took you there. The postmortem is basically the same idea is you write the timeline, you go through the events and you try to understand how did things happen.

You have this idea of the five why's in AWS which is that you have to ask why five times, I think it's a Toyota theme. The idea behind it is that you shouldn't be stopped being on the first thing because there is a reason why that thing happened and there is a reason why the previous thing happened etcetera, etcetera right? So when you finally reached the root cause, then you can start talking about okay, how do we make this not ever happen again.

That's part of the continues improvement quota. This is a way of fixing things that they don't happen again. If you have the same problem two times then you are not trying hard enough honestly. It's the same idea and this is something that I started using with the RAM books as a way of understanding why things were happening. It was a way of reverse engineering the problems we were having and protection to make them explicit and be able to work with them.

So the goal with that was to get the RAM books outdated and change the problems, right? With the incident postmortem, we already had a timeline. We had a couple of ideas but we wanted to perform a postmortem that made sense and that explained the whole thing, all the problems, all the issues we were having and make really explicit what we were doing about it because you're going to have problems. The thing is what you do with the problems afterwards.

[0:43:50.9] JM: Let's talk more about culture. We first talk a little bit about GitLab and then we'll talk about Amazon. GitLab is an entirely remote company right? Or is there some office?

[0:44:03.0] PC: Are you — you've kind of hop it.

[0:44:04.5] JM: I asked is the GitLab culture entirely remote or is there some centralization?

[0:44:11.1] PC: GitLab is 100% remote. It's remote only. I think there is some issue in San Francisco used for I don't exactly know what but there is no offices. This is actually interesting because we all play by the same rules.

[0:44:28.6] JM: What do you mean?

[0:44:29.9] PC: What I mean is that when you have some people remote and I have seen this when working at Amazon that you had a team that was split between Dublin and Seattle for example and you can see that there was a core of the team somewhere and then there was the satellite team. The satellite team wasn't in all the conversations. They were missing data. In GitLab that doesn't happen because the core is distributed.

So we are forced to work synchronously. We're forced to work with issues. We're forced to use Slack but particularly we're forced to share data in a written form in a way that is accessible for everyone in the team which flattens the structure. You don't have a group of two, three or five people who has much more high bandwidth communication. All the people plays by the same rules.

[0:45:26.9] JM: Yeah, what are the costs of that? I am running Software Engineering Daily which I work with one other person full-time and that's remote and then I'm building this other company, Ad for Price, which is about five or six people and we are entirely remote. We are distributed across the world. We're in different time zones and Slack is our centralization point and I love it. I love the flattening and the standardization but personally, I do miss some of the super high bandwidth in person interactions and that stuff doesn't seem replaceable. So what are the costs to not having some of that in person interaction?

[0:46:10.1] PC: So you will have this form of interaction. You can actually have a handout with this person and have a conversation. What matters is that whatever you talk about there doesn't exists until you write it down. In general, you will be, I mean we're humans, we do talk. That's the way we are but the issue or the big issue is that when you're only having the conversation with a single person and all the rest of the people is out.

The cost per s, maybe there is this chance of if you are too far away in time zones, time zones are hard and sometimes you will need to talk to someone who is far away which means you will need to stretch your working hours to be able to talk to this person which means you will need to reallocate your working hours.

[0:46:58.6] JM: How is your experience at GitLab different than Amazon. You were at Amazon about two and a half years, were you in Seattle?

[0:47:06.4] PC: I was in Dublin. I was working in Dublin and in the US not working for two and a half years or almost three.

[0:47:13.6] JM: Right, how was that and how was it different in GitLab?

[0:47:16.9] PC: Amazon is a huge company. Amazon is like a small country. Everything is moving all the time, you're going to have emails. You are going to have your 600 emails every morning that you need to learn how to skim through or 10 minutes of fire, house of communication every morning but in generally Amazon locally is like a classic company from the perspective of you are going to be working in an office and you will be sharing these sync time with the people.

You do have to travel a lot. I travel to Seattle a couple of times and I think that the main difference with GitLab, the first difference was the size of it and in Amazon, pretty much all the problems are — I am being a little bit naïve here but all the problems solved are solved from the perspective of you already have as tree, you already have a lot of infrastructure. You want performance deployed, you have Apollo, you want to do this or that, you already want to have a lot of tools that are going to speed you up once you get to learn them.

[0:48:16.4] JM: Once you learn them which takes like a year.

[0:48:21.1] PC: I think I mentioned once that the onboarding process that I did at Amazon was like six months. That's when you finally understand all the moving bits because you have to learn while the things are moving and so you have to get into the rhythm of learning. GitLab is a much smaller company. At this point when I joined it was something like 50 people now it's a 150. It's growing a lot but it is far away from the numbers from Amazon.

Which means that there are a lot of things that are not solved yet which means that you do need to work in a lot of different places, in a lot of different — you need to own a much larger piece than what you are going to be owning at Amazon. At Amazon you are going to be owning a specific problem of a specific team. Here you have to own the whole infrastructure for example.

[0:49:16.3] JM: What team did you work on at Amazon can you say?

[0:49:19.7] PC: I was in networking and network automation.

[0:49:22.7] JM: Network automation, yeah the thing that was cool about Amazon was just that the energy that I liked was that the energy level was always so high. I felt like the entire place was pulsing was energy and I think that it is really interesting that they were able to build an organization where that energy really scaled because I think a lot of companies when they get big they start to lose that energy.

[0:49:47.2] PC: Yeah, that's true. Amazon was vibrant all the time and it looks like it is always trying to find the next thing and doesn't really know exactly what the next thing is. You have large issues to solve in the networking space for example. In networking automation, there is a lot of work to do there yet and given the size of the network which is really, really large the problems are hard. What you were saying at the beginning of Bases saying that I take this to the white board because it is not going to work.

It's going to fail off the first time, the first time you want to run it networking is kind of the same thing. Even if you are building a new product, you have to think about the scale of things at the beginning which forces you to think in a specific way. The thing about GitLab is that it's also

vibrant. It's also fast paced. I do find a little bit of the Amazon craziness in GitLab and I enjoy it. One thing I think you mentioned once was that the people that were at Amazon, they can go back to Amazon because you've already match the culture.

You think that way and that culture is actually going to go with you that way of working, that way of thinking. Ownership for example is the thing that you are going to have very present because it is one of the leadership principles.

[0:51:13.1] JM: What do you think about the difference between Amazon's culture and the culture of other large companies, I am thinking like Facebook or Google where they have all these perks and they really make you feel comfortable as an employee whereas Amazon, you never really get comfortable but it seems like the people that come out of Amazon really value their time there. So in that way, it's almost like going to MIT or something, just getting brutalized with how difficult it is but what do you think of the costs and benefits of the luxurious culture versus the work you to the bone culture?

[0:51:50.1] PC: I think it's different people wants different things. The perk of Amazon is that you can innovate all you want. So one of the things that will happen at Amazon is that if you build something that actually makes sense to someone you're going to be owning it and you may have a new product from that. I don't know so much all the other cultures because I wasn't there working but one of the things we were saying when we were inside Amazon is that we don't actually need those perks. We want to solve the problem, we want to have these really difficult things to deal with.

[0:52:26.1] JM: Right, that's the perk, that's the biggest perk.

[0:52:29.4] PC: That's the biggest perk, right. You want to have a nice couch? Fine but that's not what we're here for.

[0:52:38.8] JM: Amazon did have some nice couches.

[0:52:40.6] PC: Yeah that's true. It depends on the office though.

[0:52:44.4] JM: Yeah, do you have any other interesting memories from Amazon? I don't know I always like reminiscing with people who have worked at Amazon because I only worked there eight months and I wouldn't consider my experience there a success but I learned so much that I really just value the company, I treasure the company and I love talking to people about their experience there but do you have any parting memories about Amazon?

[0:53:11.0] PC: I think at Amazon I learned the value of data. I learned the value of having this scientific approach as scientific as we can get in software engineering but we used data a lot to drive behavior for example in networking and that was extremely successful from the perspective of understanding exactly what's happening and having the ability to adjust our direction. That was something I brought with me and that we're doing at GitLab pretty much every day. You know you can have a lot of assumptions but if you measure, you have data and that changes the game completely.

[0:53:54.3] JM: How do you see the GitLab product evolving into the future and for people who have not tried GitLab, what would you say to them to encourage them to try it out?

[0:54:03.5] PC: So to try it out, if you want to have your data on promises just install the package. It's really simple and you're going to have a full blown development system that includes even CICD. It's really full, it's the whole development OS so to speak. It's great. How do I see it evolving? I see that the main challenges we are having right now are all in the scale realm and that's going to help us. That is going to force us to change a lot of the things we are doing.

That's going to help us to drive change in a way that we can cope with all these scaling issues and I think that a lot of really interesting problems are going to come from there and a lot of interesting solutions because it is not about only solving the scaling problem, it is about shipping it to the customer in a way that just works.

[0:55:00.5] JM: Pablo Corranza thanks for coming on Software Engineering Daily. It's been great talking to you.

[0:55:04.0] PC: Thank you.

[END OF INTERVIEW]

[0:55:05.6] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning from each other. Check it out at symphono.com/sedaily.

Thanks again to Symphono for being a sponsor of Software Engineering Daily for almost a year now. Your continued support allows us to deliver this content to the listeners on a regular basis.

[END]