Automation with Pierre Palatin

Pierre Palatin dives into different automation strategies, how to build confidence in your system, and why designing the UI may be your biggest challenge.

Viv: Hello, and welcome back to the SRE Prodcast. This is a limited series where we're discussing concepts from the SRE Book with experts here in Google. I'm Viv, and I'm your host today, along with MP—

MP: Hello.

Viv: —and we are here today with Pierre to talk about automation. Hi, Pierre.

Pierre: Hello.

Viv: You want to tell us a little bit about yourself and your work at Google?

Pierre: Yeah, of course. So I'm Pierre. I've been an SRE at Google for a long time now. I've been at Google SRE since the beginning [of SRE] for 14 years. I'm more on the software engineering side of SRE in general. I've been working on Gmail-related topics for the longest time as an SRE, where I learned a lot about how to run large production systems. But nowadays, I'm working on something called Prodspec and Annealing. So that's what a big chunk of Google production is using nowadays for doing continuous deployment— so, how to manage your production configuration at scale— because we have a lot of things which tend to get in conflict, so that tends to be a lot of work. Yeah. So that's what I do at Google. And outside of that, I'm a nerd. So usually, I do all computer-related stuff. I like graphic rendering, programming in general. And outside of that, when I'm

not talking about computer stuff, I'm probably talking about food. Usually.

Viv: I love it. Those are all great topics. So, automation. I guess the first question I have is: why should you automate? What are the main reasons to go in that direction?

Pierre: Why would someone want to automate? Because yeah, automating something is actually a bit of work. And it's not always what is needed. So from what I've seen, there are usually two main reasons, I guess. One is related to toil, and the other one is related to safety. So the toil-related is— so, toil as in, there is something that you need to do, you have the impression that you need to do it often, and it's frustrating to do. That's usually a fairly big creator. It might not take that much time in the end, but it's very frustrating. So there is this ratio of frustration versus time. And to produce that same excellent sign that you want to automate it, because for many people, it's more fun to automate it in the first place than to toil the thing. And that can end up with a better result, and you might have more time to do and to work on improving a part of your system. So toil is what I've seen as usually being the main motivation or the biggest factor for automation.

Safety will be probably the second one. I'm probably forgetting something. But safety is also commonly mentioned because we know that [we] as humans tend to do mistakes, and sometimes very dumb mistakes. And automation can avoid a world class of problems. It does replace it with another class of problems. But it does prevent a large chunk of issues. So having a computer systematically verifying that all those five properties before pushing something are verified is better than someone who is going to be tired this day; they had some family problem and is always going—they are going to screw it up or just because when we're too worn we screw up. So that's quite often the other reason why people want to automate— I've noticed at least.

Viv: I noticed you mentioned that, as you said, automation can take a lot of time. It can be consuming up front, so it is putting the work up there. So it does sound like there are a lot of good reasons to automate. Do you always move in that

direction first? Or when do you make that call?

Pierre: Yeah. And it's not always easy to determine that, right? So I have my software engineering side, which makes me always want to automate— there's a wonderful xkcd about that, about the cost of automation, that does not necessarily make it a good idea all the time. I guess there [are] *a priori* a few things to look into. I've mentioned the frustration of people having to do the work. I think that's a very important one because when people are prepared to work on interesting stuff, I mean, it's kind of a tautology, right? And so if you can automate it— automate something that is frustrating— that would be an excellent sign. It is pretty worth spending the time on that.

There is also a more project management side of it. So that's less of my domain, but in the end, well, if you're doing that in context of a company or an organization of some sort, there will be some trade-off about the time you need to spend up front. Do you have the time to do it now? Is that worth the time? Or are you just spending two quarters of work for saving a week of work? Maybe that's not worth it. So that's [the] project management kind of things. And from that, I could believe that automation is not necessarily different from many other projects. That's like looking at the trade-off both from a human, people perspective and from a technical perspective.

The other thing to take into account is the difficulty of doing it. Because there are some problems which are easy to automate, and even if the game is not that clear, if it takes you two hours, maybe just do it; and some others, which might save you a lot of time, but the end result is not clear. Is the end result going to be actually saving that much time? Or will it break down every time you use it? And that will be also be a criteria— I mean, if you want to automate something or not. That one, I've noticed that it's very, very hard to tell up front is the number of times that we saw that we were going to automate some things that's, "yeah, it's going to be great, we have these procedures that people follow up, and of course we can automate that," and then realize, yeah, it's breaking every single time because okay, it's actually difficult to measure this metric to know if the push is working. Or it's difficult to model that because it's changing all the time and so

on.

I think, yeah, and what to automatize also is probably a criteria. People care about automation— [that's] a generalization, so that goes as far as it goes as with any generalization— but there are things about your productions that you care more [about] than others, and in the sense that you will look more at the details. For example, [for] all of the services, I know there is usually at least like this one big, central server that's a really important part where all of the engineering bandwidth is going. And then there [are] all the supporting servers that people don't really care about. They are unnecessary. When they will be down, you will probably still have an outage, unfortunately. But they're less on top of people's minds. And I found out that things which are more on the sidelines is a prime spot for automation because people will not care about the details of how it happens. They just expect that it happens. And their automation will be great.

I guess it's a variation of the pet versus cattle metaphor: when you consider it as a pet, automation might help. [That's] definitely its place. But that might not be the best way to start. Why is the cattle inside? Yeah, you would probably want to automatize it because you don't care how it happens. You just want it to happen. And you probably have a lot of it, also. So yeah, that's where you will usually start for determining what needs to be automated.

Viv: So in terms of automating, what does it actually mean to automate a system?

Pierre: Yeah, what does it mean? And how much do we want to automate? The things in question and one reason is that it's one of my— a pet peeve of mine— is that about automation, when automation for me makes a lot of sense. Either one thing which makes far less sense to me is automation tools— like there is this one ecosystem of tools which seems odd: oh, yeah, those are automation. And those for me make far less sense. They have their space in industrial automation. But here I'm thinking more of automatizing your production, which is running on Google Cloud, on AWS, and so on, so this kind of large internet services, typically, production. And for me automating is the process of going from something that has been done manually to an extent— so this is a subtlety here, of course, but

from going from something manual to something which is part of your service. It's not something special, it's not necessarily a workflow— I think it might be?— but it's really part of your service.

So, do I have an example for that? Yeah, for example, for managing a mail system— a webmail system, let's say— and you need to deal where each mailbox is located. So you need to manage the load, you have like two [or] three regions, maybe, and you need to manage the load and initially you start with a small service. So from time to time, you have someone who looks into it and sees, yes, the cell over there is a bit overloaded; I should probably move more people there, or create new people. And you do that manually. Initially, it works because you don't have many users; that's great. And then you realize that it starts to get annoying— people do mistakes while you're doing it, and so on, so all the usual signs of yup, we should automate that.

And here you can have two approaches. One is okay, we have documentation about it. So I'm going to take the documentation, I'm going to use one of those automation tools, and I'm going to implement this process. Or you can change a bit of perspective on it and instead think, "Okay, now I have my system, and I want it to automatically manage the location of my users." There will be a bunch of criteria; the load is one of them.

And for me, the second approach is going to work a lot better. Like, taking a process which was made to human and transforming it into [an] automated step, but very similar, does not work well— I've never really understood why it does not work well, but in practice, it does not. Those processes never end up working well; I don't know why. I guess one of the reasons is that usually they are not running all the time. So things break down over time. The constraints of what is doable by human is very different from what the computer is good at doing. So that is a different constraint.

And so yeah, changing your perspective is that yeah, no, you're not building automation— you're building your service; you're building infrastructure. And here's the constraint: I want to keep the load of my users, my mail users, my mailbox properly balanced. How I will do that? Oh, I will probably have something

which continuously verifies the load's data through seven days' average, whatever, and go from there. Sometimes you might end up with the same result, continuous integration— so the process of creating your next version of your binary quite often looks very similar. But the perspective, I think, is important there.

I think that was the second part of your question. Yes. Oh, yeah. How much should be automated and what's the place of the humans and people in this automation? I guess, especially I think I mentioned something like, "automation is about going from something manual to something [that's] part of your infrastructure." I would like to say that, yeah, it means that everything is done by the computer, and then you don't have anything to do— that would be great. The reality of all the systems I have had to manage is that I've rarely seen a system that runs itself; in the end, there is always something to do. So I would imagine that the answer is not clear-cut about what the involvement of humans is.

I guess a different aspect to consider: so first, the involvement of humans, of people is probably going to have a lot of variation. There is a difference, because why are they involved? Is that— is someone involved because they have to fix the bug? Okay, we're still going to have [that]; I mean, we try to reduce them. But it only goes that well. And then there [is] the involvement, because it's part of the process— the typical case is approval, like there is an approval step, for example of moving your mailbox around. Maybe you want someone to validate the new layout of your production when it rebalances things. And yeah, you want a human in the process.

So yeah, for the first one for bugs, and so on, that's going to happen here, you want to reduce that. And I would say that is more of a sign of a failed automation or not. If people often need to fix those bugs, probably the automation is off the marks— something's missing. It's hard to understand a lot of the time, but once you start running into the problem, it might be easier to identify that. Yeah, it's really difficult to know whether a user move actually worked, because that's always a mystery box over there, which does not do what we think it does, and so that makes all of the automation not work.

The manual part— as in manual, as expected, and as part of the process— those are [the] typical ones— is wanting to have human validation. Yeah, this one I never know what to do with. Internally, in general, so at Google in general, our approach is that we should have the least amount of that possible for infrastructure management. There are many things at Google, and for infrastructure management, in general, that should be automated. And the reason, usually, is that we consider that decision that someone will take will not be as good as what the computer is going to do. And it is true to an extent. And we see that when it comes to evaluating the quality of all this.

So you have a new version of your binary, you push it through a canary, and it's running as a canary. And so before continuing to push the rest of production, you want to evaluate that. That's normal. And it used to be the case that the instruction in the playbook was saying, "yeah, go look at the graph. Good luck." And instead, we replaced that nowadays. We have [a] system which looks at those graphs automatically. And it's great because I don't like to do manual work. And we have pushback on that initially because the automation is going to miss things which looks of use to humans. Like oh, yes, that was obviously a spike. And yeah, because you did some kind of statistical analysis— I don't know anything about statistical analysis, but we do a lot of them, I know for that— and your statistical analysis misses this obvious spike; that sucks. But overall, for these kinds of things, we've noticed that yeah, it's quite accurate that humans are missing things all the time. The automated one might miss some things that someone would have got, but overall is going to be better.

There is still space, I think, for a bit of manual approval. And it depends on the impact of a wrong decision as impact and that includes as a mitigation. So for example, if you push a stateless front-end server— a bad stateless front-end server— by mistake, I mean, it's not great. You will lose traffic, definitely. We try to avoid that, obviously. But you can just roll it back to the previous one. And chances are that that's going to be fine.

Now, imagine another case where you're trying to update a database schema. So you have your database schema, you add new columns, you add a new type of data, or worse, you remove data from your database. Now the cost of— yeah,

exactly. Deleting data, that's always an interesting one with automation. And the cost of doing a mistake is much, much, *much* higher. Because what— you deleted the data?! Hopefully you have backup, [but] it's not a strategy. So you have backup; let's assume that you have backups. But even then, it's going to take time and so on, and all those variations.

I've seen outages, for example, where there was a small bug, and that generated corruption into the data. Well, that is a lot more work than just a whole bug to fix. And yes, those kinds of cases, in my opinion, might do with an extra layer of approval, like do sanity checks. Because, yes, the automation, the computer, will pick up a lot of things. And then we add the toil of the human [who] will pick another set of mistakes. And you will go with that. Yeah, and in general that's what we're doing: we remove most manual approval steps. I think we do have one for some database management and things like that.

MP: There's one case of automation that I'm a little curious about. I'm trying to figure out how to describe it well. I think it might actually fall in the category of kind of a roundabout manual approval when you have a free-floating parameter for your system, and you're allowing automation to basically manage that value for you, but you put limits on what that value could be. And occasionally a human has to go and adjust those values so that the automation can keep doing its thing; it kind of gets stuck on the edge of its limits. Because if you made the parameter space zero to infinity, your automation might just try to drive it to infinity, and then it causes other problems. Does that fall in the manual approval space? Or is that slightly different?

Pierre: That's a very interesting question. I know less about those systems where typically yeah, I've seen a lot of those types of automation for managing the shape and the size of your job— like the amount of memory you're using, the number of tasks [are] typical examples that comes to mind from what you describe. And yeah, I guess yeah, I guess it's a type of manual report. That's true. That's true. Because the reason I would add those limits in the first place is a sanity check is that yeah, I want us to adopt the thing automatically. I mean, exactly like you said, but yeah, I don't understand the systems that well. I know that your systems are awfully complex, so we cannot predict everything,

unfortunately. And I know that, yeah, if it starts using no memory, or more memory than a single machine has, for example, for the obvious extreme, I don't want that. And probably yeah, I don't understand how this system is going to work so I'm going to put limitations, and in a way that's very similar to those manual approvals, where it's to catch a different type of mistake that the computer can do, but a human can trivially block. Yeah.

Yeah, this kind of automation is also— you never know what to do with this kind of automated adaptation. Because the failure mode— it's very useful, because you start a new server, you want multiple copies of it, they might have very different load profiles, for example, so that just adds up automatically. I avoid the toil of having to tune all of them. That's great. But they have other effects, which is that they hide other problems. And so you're going to have, for example, your memory usage, which is going to slowly creep up. And then suddenly, you cannot start your new job, which is like the other kind of your production, like completely unrelated, but you cannot do it because it has been blocked by this small automation over there. It's a tough one.

What do we do for— and that goes away from the manual approval, but for this kind of metrics— when we do automated evaluation of a push? In your automated evaluation of a change, there are two approaches that we take. And it depends on whether you know what "normal" means. So, normal in the context of your monitoring, typically, is that— and there's one approach with all this statistical analysis, which is yeah, you don't know what normal is, but it's going to tell you if that changes significantly. It helps with problems creeping up a bit, to an extent. And then there is a one which is close to those limits that someone places explicitly where you say "Yeah, I think that, I don't know, I think the number of RPC errors I want to have; it should be around zero to one percent, for example, because I know my system." So that's how it is. And that can be also used to evaluate things. But that changes a lot of things on how to evaluate a result of the automation in the end.

Viv: Yeah, there's a lot of different, I guess, factors that play in. So I'm curious: when designing an automation system, does it vary between, you know, whether or not you think that it'll be mostly totally on its own, no human approvals

needed? Versus you think it will have more of that— I don't want to say collaboration, but if you are anticipating humans to have to do the sanity check for you?

Pierre: Yeah, I guess that's— I would typically start with yeah, I don't want to have people in the loop. And then when I'm forced to have them, I will add questions suited to people. And when I'm forced, I don't necessarily mean by someone most of the time, but forced because yeah, I just cannot get a signal, for example, like I cannot evaluate the outcome of that; it's just too complicated or too noisy. That would be a typical case.

I'm only half-joking about being forced because there is also a perception problem. So a bit further away from the purely technical, whatever that means, aspect of things, is the sign that— well, in the end there is still probably someone responsible for the system. Like, it will be an SRE on-call; we need to understand how things work, and we'd want to learn about all this automation work and build confidence on that. And that's a place where it can be actually useful to have those kind of human checkpoints in the middle, even if they are false or true, they are not really, but because they help build confidence, because oh, yeah, I've just adopted this new system over there, which permits me to be completely intent-based. It's magical. I'm fixing this place over there, and my production is going to be safely updated; it's great. And trust me, it's going to work. So as an SRE, typically you will be a bit skeptical of that. Because hope is not a strategy, as the saying goes, so you want to build confidence. It's a new system, you don't know it, and it's perfectly normal.

And I do think that's actually one of the cases where it can make sense. Yeah, ask for approval in the middle. That will make people— that just helps us. People are getting more confidence because they know that the system is not going to progress on its own because they are in the middle. They can take a look at it and look foolish after a few weeks, where they have to go click that. That's usually the reaction of most people: is that "Yeah, okay, can I have something which clicks the button automatically?" Glad you asked— I'm going to remove the button and everyone is happy. Or the other way around is that every time someone checks, they need to fix something, and then yeah, okay, maybe we need to go back to the

drawing board. So I guess that provides a— that can provide a very good feedback loop to improve your automation, also.

MP: I think the other one that I've seen in several different places is a dry run mode. So it's not even asking for an approval or going to attempt to do anything. It's just always telling you, "Well, if you'd let me, this is what I would be doing."

Pierre: Yeah, yeah, this is dry run mode. When it is possible, to do a dry run mode is indeed a great way to build up confidence. I like to know what is going to happen, and I've noticed that I'm not the only one, indeed. It's like yeah, okay, that's what's going to happen. And I know I can run it without touching production. Hopefully, the dry run is indeed a dry run. I'm not going to say I've never had a postmortem, where the dry run was not a dry run. But that's a different story, I guess. [Laughs]

Viv: Uh-oh.

Pierre: I might have taken down Gmail this way. But yeah, no, the dry run is a very useful tool. What I've noticed though, in those Prodspec and Annealing I was mentioning, so what we're using for continuous deployment these days in many parts of Google is that there are some things which are very difficult to have a dry run for. And we have this intent-based system, where we decide what we are going to push next, evaluate what we've pushed, and then decide, okay, no, we know that we can update this place, because it's for the user while they are still sleeping, I don't know, I'm making things up here, right ? It's a very dynamic system and the dry run— and the system fundamentally— is built on to the result of evaluating the push. Like, you modify something, you evaluate it, and during a dry run, that is tricky. Yeah. I mean, you can say, "Okay, I'm just going to assume that it passes because as a result, passing my joyride in my simulation", but only give—that gives relatively little information. I guess what I'm arriving at is [it's] sometimes harder to do a dry run than others. But when it's possible, yeah, but by all means, it's a very good tool.

There are variations of dry run, also. What [a] dry run means for one person can sometimes differ from another person. Because we tend to have multiple layers

of automation, I guess: one thing is going to tell something else to modify something else, there is maybe a cache in the middle, and some state-keeping over there, and then okay, what are you allowed to actually modify in your dry run? And what do you [not] want to touch? It's usually not too much of a problem. But from time to time, it can get interesting.

Viv: This is maybe going back a little bit, but I really like build confidence as kind of a tip or guideline for maybe what your system should be striving for. I am curious, do you have any other tips or guidelines on what your system should include? In particular, as you just mentioned, you know, if it's interacting with other automation systems. I guess for both cases. What are good things to consider?

Pierre: Yeah, I want to build confidence and improve— get people to use it, if you believe that's the thing to use, and so on. Yeah. So I guess I'm going to talk largely in the context of intent-based systems here. So there are often two types: big types of automation, imperative and intent-based. Imperative, that's your automation [where] you're going to describe the steps: first build the binary, then run in canary and verify that, then push to cluster A, then push to region B— things like that: very fixed.

And then there's intent based, which is, yeah, I want a new version of the binary over there, and the automation behind the scene… well, not too much behind the scene. But the automation knows that, oh, before I can push it everywhere, all at once, I need to test it somewhere. Oh, also, I'm not allowed to push it all at once, so I just need to pick a place to start it. So that's intent-based. And one thing, which is tricky with this type of automation is that it's quite often disconnected from what the users do. So the users do modifications somewhere, and actions are going to start in another place. While in an imperative style workflow, then quite often the person is going to say, "Yeah, push my config now," or "Push my new version now."

So in the intent-based, there is this disconnect. When there is a problem, the person in charge of the system will have no starting point. Like, ah, something is wrong. It has done automation again, which caused [a] problem. It's probably

right, to be fair, but this is very tricky— and that's very tricky to build confidence based on that. And here, that led me to believe that the UI of automation is absolutely fundamental. That's actually the most tricky part to a large extent of automation. And that's encouraging to help build confidence: how do you represent it? What do you show? Which kind of control do you give the user? Or do you need to engage the information overload, though I do have a preference for information overload versus not having the information? But it's tricky to balance. And that's actually one challenge that I'd say we are struggling with. No, it's not true, we have plenty of good solutions and so on, but that's a very tricky patch.

And I think it might be worse by— there are people like me who like writing code. And so I'm usually the first one to want to automate something. And UI might not be the first thing on my mind. I don't mind writing UI; actually, I do enjoy writing a bit of UI. But that's not the first thing I do— let's put it that way. And it's very easy to fall into the trap, I think. I've seen that I'm not the only one in that situation, I've noticed— it's very easy to fall into the trap where you think and concentrate on doing the automation. Because it looks like the difficult part, when the difficult part might be the actual UI, and the UI will be fundamental to help people debug, to build confidence, and so on.

I was mentioning industrial automation. So those big factories, like [a] chemical factory where you don't have a complicated process, but when the process goes wrong, you might kill everyone around— like [a] dangerous process, things like that. And this kind of automation, obviously there are different constraints; that's why it's approached differently. But this type of automation very often has a very strong component of how to show it, and you will see this very simple diagram. Maybe the process behind it will even be simplified, just to be able to have a clearer idea of what's happening. Obviously, when you're automating how to push your front-end server, the risk is very different, so the trade-offs are very different. But still, I do think the representation of automation is actually a challenge. And a good way to build confidence at the same time. So probably good places to investigate.

Viv: You mentioned intent versus imperative. And then also a while back, I know

you were talking about part of automating might be either choosing to implement what the user would typically do— I think you used a general existing process— or kind of shifting perspective. Do those map to each other? Like does one translate into imperative, the other into intent, or is that a separate result?

Pierre: Yeah, [I] think they will often map; they are at least very correlated. Not necessarily, but they will be often correlated. And I think there is space for both the imperative and the intent one. I mean, I've spent the last few years working on an intent-based system. So I'm clearly biased toward intent-based automation. But I do think there is clearly space for things to just work better as imperative. So those are a bunch of trade-offs. One of them that was mentioned is that intent might be harder to debug because it's harder to determine what the system is going to do. While with imperative, you might have even written your server steps, so it's very easy to understand. And that also ties back to the pet versus cattle— is that is a part of your system that you really care about, about the various steps you once tied control, like every week? Maybe you're going to adjust a bit of the steps. Imperative is probably going to be the way to go. For all the rest, that should probably be intent-based.

And from what I've seen, also, people tend to go more on the imperative side, because people like to— I mean, it's normal, me too— like to have control over other things, and the imperative workflow tends to give you this impression of control. But moving to intent adds also benefits because, well, firstly, it might automate more, so it's likely to break less often, to a large extent. And that will also encourage many rules across what will be different types of workflows. When you write a workflow, you do that from the perspective of what update you have to do. Like, I want to update my binary, so you're going to write that. But your system is likely more complicated than a single binary, because maybe first you have 10 binaries to run. And then you have the configuration of your database, you have your Pub/Sub config, or your load balancer. And then there is probably Kubernetes into the mix somewhere, which probably needs some configuration, and so on, and so on, and so on. When you take the perspective of an imperative workflow, you're going to miss all of that. So you're going to need to plan, but okay, how is that interaction? Is there another workflow? And people don't do that

in the end, because it's complicated.

With intent-based, you can have a better picture of what needs to happen. And then oh, yeah, I'm just going to stop there— the push of my new binary— because I just saw that there was a change in the database schema 10 minutes ago, and I should not modify the binary because that might be too dangerous to do it now. So it gives you more options to do that.

It's also: quite often, intent-based allows to shift—how to put that? Everything that people would have learned over time, on how to— I'm missing an expression here— all people have learned how a proper push should be done in your company, in your project, whatever. That allows you to encode that much more easily. In an intent-based system, compared to a workflow, while workflow tends to encourage a lot of customization, so when you want to avoid the customization, going on the intense intent side is quite helpful. And actually, I've seen that. I happen to have discussed recently with people from a few other companies building on intent-based systems and they were usually not working on intent-based automation for their own service. But because they are providing an automation system or production layer to other teams who're there— developers to multiple developer teams. And that was their way to actually encode those best practices. So it was a very good way of doing that, some things that workflow-based is quite often harder to do. It's not— as with everything computer stuff— it's never impossible, but it's a lot harder with imperative workflows.

MP: So looking over the episode notes, here, I see this phrase "delta-based production changes." And that's not a phrase that I've actually heard before. So I'm wondering what that actually means and kind of how that relates to this idea of intent-based versus imperative automation, and can that be used for either. Is it like an entirely different idea? I'm just not really sure how to parse that phrase.

Pierre: Yeah, I think that's also one of the things which is correlated, but not the same, in a sense. Okay, so what do I call delta-based changes? So, in the end of automation I've been talking about— we've been talking about— is about maintaining your infrastructure and maintaining your production. And that's

maintaining it in the face of changes. You want to change it because if you don't want to change it, there is nothing to automate, and we can all go on; it will be also boring and unlikely to live long. But we are here to deal with changes to your system. And whenever you introduce a change to your system, there are two big ways of doing it: either by data or by absolute value. So, a specific example: you want to change the number of tasks you're running for jobs— the number of replicating a Kubernetes deployment, for example. You can either send it to your system or indicate somewhere "oh, by the way I want plus 10. I want 10 extra tasks." Or you can carry around the change that you want: now you want to have 50 tasks because before you had 40 tasks. Now your target is "50 tasks"; it's not "plus 10." So it is what I did in the sense of workflow vs. intent because intent naturally goes to a world of using absolute value, while with workflow, it's very tempting to have those deltas.

It can get tricky also when you look at the details— that's where it gets messy because in the obstruction when you look really inside it, depending on how you look at it, it can differ. So what might start as a "plus 10" might be then converted later in your pipeline as an absolute value. And so, yeah, so the reason I was mentioning that I guess, it possibly applies— does it mostly apply to intent? No, actually, it probably applies to both. So the problem we've seen with delta-based changes—so where you carry the change you want, the delta you want to apply (log those "plus 10" tasks)— is that prediction never ends up in the state you want, somehow. That's also one of those big mysteries that production is always more complex than you expect. And so when you're just carrying around those plus 10, then you're going to have problems. For example, you might guarantee at least one semantics. Well, now you've added 20 tasks instead of 10 because something in the middle failed. Well, not good. While if you add the absolute value, you can retry as many times; it will be still 50 tasks in the end. And I've lost my train of thought— sorry about that.

Viv: I also didn't know what delta-based production changes [were]. I mean, I did know what it was, I guess, but I didn't hear that term.

Pierre: So those are the things that it's difficult to guarantee as a semantic of your changes, and the importance is great for automation because [with]

computer systems, you can usually guarantee at most once or at least once, and with delta it is difficult to manage. And the other one is bootstrapping. And quite often, you want to understand whether production is actually matching your intent of what you're supposed to have, or maybe you have a big outage. So you have to take down many things— you want to recreate it and you need this absolute value. You need this 50 tasks. And if you just add like delta applies, it might be very odd to reconstruct. We had some cases where those— which were particularly odd to work with— where those delta were themselves being transformed into the type of process... imagine that you're increasing an amount of quota, but this quota it's, let's imagine, I don't know, it's something like your database quota, but your database to run below it needs some compute resources. So when you ask my database to be twice as big, that means maybe I need now three new servers, a bit more quota on my Pub/Sub system, whatever.

But this translation varies over time. Because yeah. Today, doubling the size of your database, adding two extra servers on your database, might mean having only one new computer— like [an] actual machine. Maybe in six months, that will mean three machines, or maybe that will be optimized [and] it will work differently. This translation function between what you're asking in the first place and what you're getting in the end, that changes over time. And if you manage that with delta, well in one year, you will have no idea why you have the resources in the end. There were a bunch of requests. You can probably look at the history of their requests, but you're not going to have a good time [with] that. So dealing with absolute values when you can is great.

It still makes sense from a human perspective to have some places where you can just say, "yeah, I want 10 extra gigs of memory because that's how we think about the problem quite often", but as soon as possible that should probably be managed to absolute value when you can, and that will make your—I think I have a strong opinion on that one—that's going to make your automation probably work better.

MP: Yeah. It sounds to me like the major problem there is that with delta-based: you're adding an additional dependency back to the current state of production.

Pierre: Yeah.

Viv: Yeah. [Laughs]

Pierre: I really like how you put it. Yeah. You add the dependency on your current state, and you do know, we try to avoid that. A bunch of us think Prodspec, so Prodspec, you know, that's a representation of one configuration at Google. And one very strong principle we have is that we can rebuild this configuration without looking at production. It makes it easier for many small things— some larger than others— like rebuilding something which crashed and disappeared and was deleted by mistake, for example. Yeah, and when you do a delta, that's exactly that indeed— that you build on the current state, and your current state, it exists only in production, and that quickly has limits. Yeah. I guess there are some cases where it's still fine— all those parts of configurations that you never really change. Like you initially created your service in your cloud provider interface. You did that once for the lifetime of your service and maybe from time to time you need to change some things there. Not automating that is probably fine.

Viv: Yeah. Speaking of complicated things that potentially change, what about continuous enforcement? I guess in particular, too, I know way back we were talking about when to apply certain principles to, you know, intent versus imperative systems or workflow systems. So, I don't know. Do you have—

Pierre: Yeah, continuous enforcement, yes. That is also something I feel strongly about. Continuous enforcement. [Laughs] So the idea of continuous enforcement is that [you] assume that you're managing your production by having a configuration somewhere. Maybe it's in a Git repository, maybe it's in a database. You have your configuration somewhere and you have the state of production elsewhere, and you need to reconcile that. So typically, you update your configuration, and then you want production to be updated. And the idea of continuous enforcement is saying that the moment you modify your configuration, it's going to be automatically applied to production.

So one aspect I think is more of, how do I put that? It's more of a policy question. What do you want for your system? Do you want actually to have someone

having to trigger explicitly as a step or not? And I do believe there are— I don't have a good example in mind, but I think there are cases where it might make sense. But for me, that's a policy decision. That should not be guided by the limitation of your system. The technical limitation is that yeah, I'm deciding that I want someone to actually push that. And I guess that ties to the explicit human validation. Like having confirmation by someone reviewing it and that ties into that. Yeah. I do want this explicit step and okay, you're not going to have really continuous enforcement there. Or, what you're saying is that yeah, what I have is the configuration, for me, for all intended purposes, that is a state of production, so just make it happen—safely, hopefully, but make it happen. And that's where you really want continuous enforcement.

Now, what I've noticed is that production is always more complicated than what people think. I don't know why. And people are expecting their own production and they still get trapped. And me first, to be very clear: that's always tried by, oh yes, that was this thing. I never thought about this interaction. I never even thought about this domain of problems. So things are always more complicated than what we expect and what we end up finding with systems which do not use continuous enforcement, is that they are deltas—I should not use "delta." There are *differences* which creep up between the configuration and the state of production and those differences are going to creep up and then someone arrives; they did their own change of the configuration and they want to push it. And certainly, they try to update production and when they update production there is not only their change, but two weeks of changes by other people, which they probably have no clue about in general.

And that has led to many outages, actually. We have a certain number of postmortems, which are basically that yeah, that was not pushed when we pushed it, it was not the right time anymore to push it, or other variations of "they are off", and absolutely everything goes wrong. So by having continuous enforcement, you avoid this world class of problems. That is basically avoiding what caused the problem. Extra advantage: you're changing production closer to the moment where someone changed the configuration.

And this elapsed time between a change of configuration and a change of

production is important because the longer it is, the harder it is going to be to debug because the assumption might have changed. Like, oh yeah, I've changed—we configure our cache. Yes. But in the meantime, there was another server which was introduced, which was impacting the cache differently. That's one aspect. The other aspect is just altering the past. So maybe the person didn't change in the first place. So yeah, I'm on-call currently so I'm going to watch it, but then now that the other on-caller who is taking care of that when it gets actually pushed and they have no clue about it, or you might just have forgotten about it. I mean, I see that when I write code, is that I usually forget what I did the last week. And for production configurations, there is no reason that it is any different. So continuous enforcement is going to help you tremendously with that.

It is tricky and we are talking about getting confidence into a system, so it's usually a relatively difficult step because it's a change in mindset. Because you have to trust the machines to do the thing automatically. You don't have this push tool that you run, which can be difficult to change. And you have to build the confidence that it is going to do the right thing at the right moment— that you have encoded all of the constraints and then, oh, yeah so that actually pushed in the middle of the weekend, something which was very risky. Yeah. Why? Yeah, because someone submitted a change on Friday, and that started to change. And then, yeah, next week, you know, you're woken up on Sunday at 6 am. And those things need to be encoded. So that's where to set up a continuous deployment. I think it's very, very often with it— when very often where it fits our cases. Where it's not, I would guess that's when the automation does not do a good job at evaluating the results, for example. But more often than not, I think it's good to have. And that doesn't prevent you to have an explicit manual step of approval. If the policy— if there was a policy where that makes sense.

Oh, it also helps for something else actually as a continuous deployment— sorry, it also helps for something else as a continuous deployment. I don't think it's specific to Google, but I've seen it mostly at Google, is that for turn-up management, so that probably mostly applies to large services, where you quite often need to set up a new copy of your stack, because you're growing your

service regularly, or you're moving it around, and so on. So some of it probably tends to be Google-specific, but anyone with a large service will probably have to quite often turn up a new instance of your service. And I've seen countless attempts at automatizing that, like, and that always started with someone documenting the process, the newbie on the team being the one having to do the next turn-up. Sorry about the experience, but it allows you to learn about the system. But so that's how it goes.

MP: This sounds familiar.

Viv: It sounds like me! Okay. I did this very recently. [Laughs]

Pierre: [Laughs] Exactly. So yeah, it happens. Exactly. So I've seen many attempts of saying, okay, so I think actually if it were documented, we know exactly the steps that we need to have. Let's have instead this workflow, this script, this automation tool, which is going to take care of the turn-up. And when they can— when this approach is taken, I've never seen that working. I have never seen an extra example where the end result was actually better than doing it manually. And it was also quite often more frustrating because what was happening is that first there were many steps, which were manual anyway, because I was not doing automation, and there was not— the work was not put into the actual underlying automation. So it was still manual anyway. And more importantly, things were breaking over time, because those things, even if those turn-ups are more frequent than people expect, it's still like once, maybe once a month, once every two weeks, and so the assumptions behind the turn-up documentation changes, and there is always a step broken— like always.

And when the step is broken, when you did it manually, yeah, you know which commands you run, so you can debug it— it's not pleasant, but you can debug it. When it's your automation doing that, then you have no idea. You have this step in the middle with a crappy error like context, deadline exceeded, which failed. And it's impossible to debug. And I've never seen this approach work. What I've seen working for turn-ups was taking the problem completely sideways through continuous deployment, continuous enforcement, in the sense that here we're completely changing the premises... instead we are saying that we are modeling

production— that's what our production should look like at any point in time. And we continuously enforce that.

So, that's why I do turn-up almost for free in the first place. Because, well, it's part of the modeling, so to do turn-up, you just add your new cluster— how complicated can that be? And you get your new cluster turned up; it's great. And also, it means that when the architecture is changing of your service— which happens, again, more often than people think of, well, because of continuous enforcement— continuous deployment is always running, then we have to fix it immediately when they are changing. They have this new cache server? Well, we have to be part of the modeling that you're doing. And it's not, it's going to blow pins on one's face very, very quickly. So you have this quick feedback loop, which makes it easier to debug in the end. And that's the only approach I've seen working for managing those turn-ups in practice. And I guess that will apply for any type of automation that you do often, but not that often. And that provides us a solution for that.

Viv: Yeah, as mentioned, I did relatively recently— just because I'm relatively new to SRE— turn up a service, but I also recently turned down a service. Is that any different in terms of your recommendations there?

Pierre: Turn-downs. I don't know if I have a recommendation. I have very warning signs when it comes to turn-downs. I have not yet found a good way of managing intent-based turn-downs. So one is that turn-downs are doomed to be seen as a symmetric problem of turning up. And they are very different problems in practice. I do understand the symmetry of it— in my mind, I like the symmetry of systems, that's great— but it's a completely different problem. Because in turn-ups, when you screw it in the middle— the consequences, you read zero, because that's your new cluster, your new cluster, you forgot to turn up the front-end— well, no one is impacted, because you're not sending traffic there. Mostly. I mean, things get complicated, but to me the risk is much lower.

So turn-down, yeah. Any mistakes? You're going to have an outage. That's roughly the baseline. And it just might be worse than that. We were talking about deleting data. And there is one thing which keeps me up at night when it comes to

automation: deleting data. Because if I take out the front end, not great. I'm not going to be happy about that. I'm going to be stressed, but it's easy to fix. If I'm deleting a large chunk of data... that's a lot more complicated to recover. We have packets and so on, but it's very, very complicated to deal with. So first, there is a big risk aspect of turn-down, which is very hard to deal with.

So here are a few tricks. A common trick, for example, for turn-down of stateful data, of actual data, is that you make it inaccessible. So if you have an ACL system or sub-security system, you say, "Okay, I'm not deleting the data." But as the first step of my turn-down, I'm saying that, yeah, you cannot access it anymore. And then you realize that when everyone told you that, yes, I'm not using this data anymore, you find people who actually are still using the data. They have a small outage, but this small outage is when you remove the ACL and everyone's happy again. So that's one trick.

After that, if I want to deal with turn-down of intent-based, it's tricky. There is one pattern I think should be absolutely avoided, in most cases: turn-down—what I call turn-down by absence. So there is one thing which is very tempting to do when you have a relatively cleaning turn— relatively clean description of what your position should be. Something which is very tempting is to say, okay, but it means that anything that is in production, but not in my intent, in some scope, should be removed. Because that's an easy way than to manage your turn-down: you just remove it from your config, and it gets deleted. And that works. I will not use that for anything more beyond tests, let's say, or really not important production— or production which can be down for an hour or two without problem. Because in practice, when configuration gets wrong, that's probably one of the biggest failures usually with configurations: is that something gets generated in the middle, there is a bug in that. It's extremely common. And then suddenly, you have only half of your configuration which is generated. Oops. You don't have your production anymore. And there are all variations of that where the automation, something, will go slightly wrong, and suddenly, you don't have your production anymore.

The approach we've been taking instead is that we need an explicit signal. So it's that you model also what needs to be done— your job. You still describe it, but

you add a bit, which indicates that it needs to be done. Managing this information gets tricky. For me, that's still an area of, I was gonna say, research. That might be overselling it, but still something where we are iterating on how to manage that.

Another pattern I've noticed, which makes it harder, is how people think about turn-down. So what happens first, for large services especially, which is where my experience comes from, is that with Gmail, Blob Service, is that when you turn down something, it's not a one step process. It's many steps and it can take sometimes weeks. So imagine that you want to down a cluster, and you have a lot of user data there. So you will want to move this user data first. But while that is happening, which can take a lot of time, you still want to serve those users, obviously. So you have the signal of turn-downs that you're starting to turn down at one point, but you're still serving, so you still need to have your job running—you might still need to update them, even if that's taking a long time. If that's taking more than a day, you probably still want to update them.

But that's surely not how people approach it. Usually it's more, "Oh yeah, I want to get rid of this blank, so I'm first going to remove it from the configuration, and then I'm going to clean things up." And that's a bit obvious when said like that. But I've noticed that again and again. I've done it myself. So yeah, so it makes the primers more complicated because once the configuration has been removed or completely removed. And that you don't want to do turn-down by absence because it's dangerous. What do you do then? Which thing do you use? How do you even identify what needs to be turned down? So yeah, in Java, while navigating, trying to do a mix of adding extra safety measures, there is always an infinite string of defensive measures that we can add, and trying to provide various tools to avoid people shooting themselves in the foot, or else all things are [a] foot gun, I guess, in some cases.

That's another thing with automation— is that automation can be safer, and I think it's safer in many cases. But the failure mode is very different from humans. When someone is in control, it's more likely it's going to do random mistakes, like really random mistakes, but that will be usually more limited. The automation is not going to do random mistakes, because that's what it's been doing like millions of times before. However, when it goes wrong, the mistake is also

automated. And that can go much worse in that case. So that's a different kind of differences are needed there.

Viv: Thank you for shining some light. So I think I just have one more, hopefully— not like that, I'm sure we could do another episode on this— but kind of a quick question in this frame of mind for you. How close are we as SREs to automating ourselves out of our jobs? Which I know is something people like to riff on a lot?

Pierre: Yeah, that's the usual saying, especially behind automation, that okay, what's the goal of automation? Well, I'm lazy, I don't want to do— even if I'm lazy, actually, I don't want to have anything to do. I don't know, I always find this framing, I understand the framing. I also like to do other things than those toil-y aspects. But my perspective on it usually is that it's about complexity management quite often in the end, and I do believe that we're mostly—all of us are mostly working at a fixed amount of complexity. We can deal with a given amount of complexity, and yeah, that's it. And if we simplify something by automatizing it, this complexity is going to shift elsewhere. And this complexity might change a bit over time; for example, you're tired, yeah, you're not going to be able to cope with something stupid in your automation or something stupid in your code, and so on. Or you've just changed teams, you're full of energy, so okay, bring it on, you can understand every— you can try to understand everything, and that overall is complexity. So for me, it's not automatizing yourself necessarily out of a job. But it's wondering *what* to automatize so you can shift the complexity to a place that you actually care about having complexities there.

There is an obvious answer to that quite often, which is related to automatizing yourself out of a job. You will usually want to move toward being able to take enough features or more services with less people— that's quite often going to be in the direction to simplification, of course. But given that it's not a one-step process, thinking about where is the complexity and which complexity you're willing to trade off instead, is the framing I find interesting because your automation comes with its own complexity— you will have to manage your automation. And it can be more or less good.

And there are some places where the automation is going to be a lot more

complex to manage and some other— sometimes automation is a no brainer, so that's easy, but sometimes the automation has a cost to maintain it, like how to automatically evaluate your push. Is that worth doing all those fancy statistical systems? Someone has to understand it, and that's not going to be me. That's for sure. And what do we do with that? Do we want to remove the complexities that way? Okay. I'm going to catch up on my statistics, maybe? Yeah, in the end, we are dealing with complexity. And where do we shift it?

Complexity's also— I guess, it's also related to what people enjoy doing more or less because there are some types of work that one person is going to find extremely frustrating, while the next person [thinks], "yeah, it's a bit repetitive. But okay, I phase out, I'm taking care of that, I'm used to it, and I'm going to enjoy it." So this might even differ between people. That's where it goes into team management. But that's a different kind of form.

Viv: Well, thank you so much for coming to chat with us today about all of this. I definitely found it very interesting and informative. So yeah.

Pierre: Thanks a lot. Thanks a lot for hosting and for forwarding the good questions. I also enjoyed it a lot.

Viv: Great.

Pierre: I hope everyone else will also.

MP: And we want to make sure that our listeners know that just at the end of last year, you had your author on a whitepaper for Prodspec and Annealing, which are Google's current solutions to these problems.

Pierre: Exactly. Yeah.

MP: We will have a link to that whitepaper in our episode notes for folks that want to follow up with that.

Pierre: Yeah, it describes everything that we're doing nowadays for continuous deployment, to a large extent, to automation. It's very long, because I tend to give

a lot of details, but I hope people find some interesting bits in there.

Viv: We'll definitely link it here so folks can head on over. All right, I think that's all. So, thank you again and... we will be back with an episode next time on on-call, so see you all then.

MP: Thank you.