The Open Source Way

Episode 17: SapMachine – The OpenJDK for All Things SAP

Transcript

Karsten: Welcome to the Open Source Way. This is our podcast series, SAP's podcast series about the difference that open source can be, and in each episode, we'll talk with experts about open source and about why they do it the Open Source Way. I'm your host Karsten Hohage, and in this episode, I'm going to talk to Christoph Langer and Thomas Stüfe about SapMachine or SAP Machine; I don't know, they will tell us how to pronounce that. Hi Christoph, hi Thomas. Nice to have you here.

Christoph: Hi, hi Karsten.

Thomas: Hi Karsten, nice to be here.

Karsten: Okay, great. Let's hear who these guys are. Thomas joined SAP in 2000 to, originally, work in the AS/400 porting group. He joined the VM team in 2004 and there worked on the VM container, SAP JVM, and now SapMachine. Thomas is a developer with focus on VM runtime, mostly the aspects of supportability, memory footprint, and generally, as he puts it, platform stuff. Christoph also used to work for the AS/400 porting group, but as an IBM employee, working in a joint team with SAP colleagues back then. He switched to SAP and the VM team in 2014, starting there as a support engineer but doing more and more development stuff over time. Christoph is now a development manager and takes care of the open JDK updates. So, you both started at SAP in the AS/400 porting group. Is that also where you met? And maybe you'll also quickly explain what AS/400 even is, because under that name, at least, it's been out of the race for a bit. Right?

Christoph: When I joined SAP, I think Thomas had already been there for, I don't know, a few years, and I remember taking over some code from him. So, he invented some fancy way to get the SAP installer running for AS/400, so leveraging Windows Machine, and I took over this code. I think the thing is still known as AS/400, but it has now been rebranded several times, so people know it as AS/400, the operating system, or i5/OS, IBM i, or whatever.

Thomas: Yeah, the AS/400 was really a fascinating machine, or it still is, it still exists, and Christoph mentioned the rebranding, that certainly didn't help. I think the last iteration, they call it IBM i; let that sink in for a minute. It's like a single letter, single lowercase "i". That's genius. And, also, it happens to be like a pronoun in the English language. So, try googling that! And the fact that everyone still knows that thing as AS/400, twenty years after they rebranded it, that means the original brand name is still intact, so nobody knows what the new name is. But yes, AS/400 is an extremely interesting machine. Just imagine, you had a black box with a completely proprietary kernel, a very strange operating system, proprietary hardware, and you had an inbuilt baked into the hardware almost, so, like a database. And it also came with a database. The idea is that you put this into a basement and switch it on, install your AS/400 custom written business software, and then you forget about it for a decade because it just works, and it is extremely reliable

Karsten: Okay. The part that I seem to remember about it was exactly that deeply built in database. But that was back in the days. That's not actually what we're here to talk about. So, Christoph, how do you say it, "SapMachine" or "SAP Machine"?

Christoph: I think I tend to use "SapMachine", but some people say, "SAP Machine". It sounds a bit like "submarine" because it was invented as kind of a submarine type of project. And then we called it "SapMachine". Yeah. So, what is SapMachine? Most of the people who are listening to this are into development, IT, and so on. So, you know, Java is a programming language that runs on a virtual machine, not directly natively, is not compiled for operating systems and architecture. JDK provides this runtime environment, and the most known one is the OpenJDK as the reference implementation. We call it upstream implementation, and SapMachine is what we then deliver for SAP, our own build of what we support for the customers, with some small add-ons, and so on, as we need it.

Karsten: So, would that be comparable to, say, I mean, where OpenJDK is a community project and SapMachine is basically a distribution – so would it be comparable to RedHat and Linux?

Christoph: Yeah, sure, so you can imagine this like the Linux Gardener and like SuSe, or, whatever, RedHat: those vendors, they take the kernel sources and package their own distribution of the operating system and with OpenJDK and SapMachine the relation is similar. So, the OpenJDK is really the upstream code to which lots of people also contribute. So, Oracle, I mean, maybe we'll get to that later, is the main maintainer of this, but others are contributing. And then lots of vendors take this code and build it and provide their distributions, and SapMachine is one of them.

Karsten: I think I understand, although I always must pay close attention when you say something about upstream, downstream. I originally studied Geology, and, of course, in the oil industry, upstream, downstream mean different things; but from oil back to Java. Thomas, how did it get to JDK being an open project at all, to SAP's involvement, etc., especially with Java being of all things Oracle these days?

Thomas: Well, that's a long story, and in fact, our involvement with JVM, with Java Virtual Machines, started way before we started to be interested in open source. Java is 25 years this year, so, it was invented in 1996 – not invented, I think they started working on it in 1991 or so, and Java 1.0 was released in 1996 and came, of course, with its own Java Virtual Machine. And the interesting thing is, the OpenJDK nowadays has an unbroken chain of history, starting at that first version. It's the same code base and, also, similar to distributions, or maybe to BSD, for instance; BSD also has a similarly long history. You have a lot of distributions nowadays floating around, which are descendants, direct descendants from that one version. And there are, of course, different TDMs in the market, always have been, which clean-room implementations. And so, that's interesting to know. Not everyone, not every JVM nowadays is descended from that one version. For instance, IBM J9 or so, I think, at least in the start was a completely different implementation. One interesting thing is that the OpenJDK code base is massive. Yesterday, I measured, and it's 12 million lines of code. So let that sink in a bit; half of this is maybe test code.

Thomas: So, but this still leaves you with six or seven million lines of code. So, it's a large beast. And when you compare this, for instance, with the Linux kernel, the Linux kernel is, I think, 20 something, maybe 21, 22, 23 million lines of code, as of release five. So, we are like half a Linux kernel, complexity wise. JVM has a core which is a library written in C++. It's a so-called hot spot. And that alone is like one million lines of code. The Sun engineers, and later the Oracle engineers and the whole community managed to keep all that stuff from bit rocking. So, I have seen a lot of large-scale software projects that have existed for a long time become very inflexible, very unmaintainable messes, and they managed to keep it backward compatible, mostly. You can still run Java 1.0 programs on Java 18, and it's still very actively maintained, I mean, Python, for instance, they had this break between 2 and 3, so Python is interesting because it's almost the same age. And so, they had this one break between 2 and 3 where they broke compatibility. So, I think this shows that there's very good stewardship in this program, in this JVM.

Karsten: Although I have the impression that Python – when was that break in compatibility, speaking in years?

Thomas: I actually don't know when Python 3 came up. I'm not a Python expert.

Karsten: Because for Python, I saw a sudden rise in popularity, just early in the 21st century. And in the very late 90s, early 2000s, Python was all over. Was that maybe caused by exactly that break?

Christoph: No, the break was later, I think.

Thomas: But that's a good question. And actually, all this time almost, Java was declared a dead language over and over again. It's far from it. I think the Java language led the indices of popular languages I think up to last year, and it was passed by Python but it's still up there. So, it's one of the top five most popular languages. In ’96 they brought out Java 1.0, and in 2006, 2007, they open sourced it. And they put it under GPL v2, the code base, this original code base. And I wondered why; I think there are a number of reasons. One thing I think is that they wanted to increase the adoption of Java, because it was competing, especially at that time, with .NET languages. That's what I think. Maybe there were other reasons. And this actually worked because adoption of Java increased a lot. And I think Microsoft followed suit with open sourcing .NET, but much later – I think they started to open source it in 2014. I think the compiler project was called Rosetta. So, they now do the same thing, but Java had a large head start at the time. And I also think a lot of the Java ecosystem was open source, and so they were kind of under pressure to open source the code base.

Christoph: The good thing was that they did it before Oracle bought them, right?

Thomas: Oh, yes, that's a big thing.

Karsten: But so, in the end, the reason for open sourcing was that if they hadn't done that, they would have been surpassed by those who did open source, right?

Thomas: Maybe they would have suffered the same fate as Adobe Flash. Maybe not, because Adobe Flash was bad for a whole range of other reasons. But it certainly helped. It also increased cooperation beyond the SUN level, because suddenly you had a lot of companies taking part in the development, and this is certainly helpful. And also, I think a lot of SUN developers do a lot of open source. And maybe this also helped with developer retention because I think they started losing people in that time. But all of that is just in my head. Maybe it's all completely different.

Karsten: So, how did SAP get into it then?

Thomas: In 2002, 2003, SAP became really interested in having an alternative to ABAP. So, ABAP was well established, as you know. I mean, it's a very well-done ecosystem, but it was very closed, and it felt old, and you had a lot of developers demanding a different language, and Java was very popular at the time. We were thinking about ways to do that, and one of the first ideas was to bring Java into the ABAP server as a first-class language. And that was extremely complicated because the ABAP server is also a strange beast, similar to this one. But in so far as it, for instance, does its own context switching. You have this idea of reprocesses, and a reprocess is tied to a CPU, and so you do your own task switching by working on user data which are mapped into memory. And so, when you're done, you, for instance, do database access and you switch over to a different work process, we call it rollin/rollout, and that means that all the user data could be mapped to different processes in the lifetime. And ABAP didn't have a problem with that. I mean, that was not a problem. But doing this with the whole JVM data was very difficult. It was pretty much impossible with the hotspot – by the way, we call JVM the "hotspot VM" to distinguish it from other VMs. So, the solution SAP came up with was the so-called VM container. We took a different VM, also from SUN, called the CVM, the compact JVM. Completely different implementations than the normal VM; it was written in C, and I think the target audience was embedded folks. So, it was very small. It was completely written in C, it was very tight and had a very small footprint, which was, of course, nice, and we thought we would be able to do this. I mean, we could map the whole data, the text segment, data segment, everything and all the dynamic data of this tiny JVM into shared memory and use it for different processes.

Karsten: And that CVM was what you brought into the ABAP server?

Thomas: Exactly. That was the VM container, and Harald Kuck was chef at that time, and he was very enthusiastic about this project. We were successful with this in a way because we got it to work on all the platforms, I think even including the AS/400. And so, this was voted out. And if I remember correctly, the problem was that people started using this thing – I mean, it's more suited to a model of stored procedure-like things, you know, so when you do very limited things with Java, and they started using it as a general purpose VM with lots of threads. And we, for instance, had to emulate that with so-called green threads.

Karsten: By the way, as you're just taking a break in the story there, I have to be a smartass for one second here. Harald Kuck, although his last name is Kuck, was not a cook, but he was in charge. You just said he was "chef" ...

Thomas: Oh.

Karsten: ...and for all native speakers, a "chef" is a cook. Harald Kuck is also called "the cook", but his last name is Kuck, K-U-C-K, and he was not a cook.

Thomas: Hmm. Most definitely.

Karsten: Sorry, go ahead.

Thomas: So, we had the problem that the ABAP server is single-threaded and then does its own context switching. And we had the problem that we couldn't just use native threads. So, we used something called green threads which is an interesting technology where you basically do your own contact switching, too. And interestingly enough, this one continues to get revived as an idea. Nowadays, we have this in the OpenJDK, we have this project Loom which is a research project aimed to drastically increase the number of threads a JVM can hold, because native threads are kind of expensive, and by using this technique, you can get this a lot lower, but there are large, huge technical challenges. The next step was that we shifted our focus to a different JVM because we saw that we had a lot of problems.

Christoph: There was also the J2EE infrastructure you want to leverage, right? So, what is integrated in the ABAP server, I mean, it should kind of fit into this programming model where you could write your procedures in Java, not only ABAP, but then there was also this J2EE technology out at the time. And I think SAP bought some, some small company, InQMy, which had developed a J2EE server and tried to integrate this together with ABAP, like next to each other. The double stack or some solutions should be based on J2EE, the J2EE server. And I think this is when the hotspot based VM came into play.

Thomas: That's why.

Karsten: Thanks. I wasn't sure, but I was thinking whether, at that time, like around 2003, 2004, we were already doing double stack with a true full stack Java server for the SAP portal and things...

Thomas: Right, yes. So, we had this J2EE server, and we needed a JVM. SAP is special in that we support a lot of platforms, or we did support a lot of platforms. So, the thing was that whatever the customer has sitting in his data center, we would run on it. So, if the administrators like Windows, we did Windows, if they were like a Unix shop, then all stuff ran on Unix. And this is diminished nowadays, especially with cloud and so on. But it was a big thing at that time, and therefore we needed a JVM on all platforms and needed it to behave the same, and also support cycles. So, we have always had the problem of support cycles organizing the stack below us.

Christoph: When the J2EE stack was rolled out, I mean, we were on all those platforms and had different VMs, so, for each vendor, like IBM brought their own J9 on their platforms, and then there was HP-UX, and so on. So, platforms which were not provided by the SUN implementation of JDK. And that, yeah, meant trouble in support because, you know, one VM goes to another platform and the thing behaves completely differently. And so, we thought that it would be good if there was one implementation for all the platforms.

Karsten: Yeah, those were exactly the times I was referring to before, when single engines or single solutions at SAP also dropped this supporting of 24 platforms or how many it was back from the 90s.

Thomas: Exactly. And so, the result of those efforts was the so-called SAP JVM, and that became the backbone of our J2EE server. It was able to run on all platforms SAP supported, and this SAP JVM was not based on the CVM, like the container, but it was based on the real, the standard SUN VM. And so, we had licensed both JVMs, and so, SAP started working and later the SUN JVM – they open sourced the code base. So, by that time, in 2007, we already had several years of experience with this code base, but on a proprietary level; the code base is almost identical, I think at least today it is. So, that's interesting to know because Oracle sells the same JDK with different licenses. And so, the Oracle JDK is not the OpenJDK. I think the differences between the OpenJDK, and the Oracle JDK are probably quite minor. So, we started making a long, long list of enhancements and fixes with the JVM. So, we did it because SAP has many problems others don't have, and we were able to focus on that, we improved performance. We definitely improved the developers' quality of life. We had a lot of quality-of-life improvements, and our developers were very happy, and we had a lot of monitoring, a lot of supportabilities. We developed our very own profile with an associated backend in the JVM. And the profiler was technologically very, very good. Unfortunately, we never open sourced it, and it probably won't happen.

Karsten: So, when did we arrive at today's state with the upstream/downstream relationship between OpenJDK and SAP's distribution?

Christoph: I think that was in 2007, when the OpenJDK project was announced. So, at that time, we were completely heavily working close source, based on the source code we got from the Oracle license. And I think the safety quality of the OpenJDK project also started to evolve from then. So, I mean, in the beginning they more or less dropped the sources, you could see them. I mean, there was no GitHub yet. You could not just create pull requests. It was difficult to get things in and so on. And also, I mean, we were focused on our own closed-source world, and then only some colleagues who were the frontrunners here started trying to look at the open-source codebase and to get into a connection with the SUN, or Oracle at the time to establish all the community processes.

Karsten: That was 2007, you said. Right?

Christoph: That's the time I have in my head.

Christoph: It was definitely before Oracle bought SUN, and that was in 2010, so I think it's not too wrong.

Karsten: And was it already called SapMachine at the time?

Christoph: No. At that time, we had SAP JVM, or, I mean, we still have SAP JVM, that is our SAP internal distribution. So, with that one we also had some restriction that we could not distribute as free software, not even as a standalone Java Virtual Machine implementation which could compete with the Oracle JDK. So, we have a field of use restrictions for this SAP JVM. That means we only run it with SAP software.

Karsten: So, basically, we have three different flavors if we want to look at the whole picture. We have the OpenJDK, we have the SAP JVM if it's totally proprietary SAP purposes, and we have SapMachine, our distribution of OpenJDK, right? Okay. And then, one thing that also confused me a little bit is, Thomas, you were talking about these times as if you had been there all along. Was that part of being AS/400 porting group or have you both been with that all along? Or what's your personal involvement in all that history?

Thomas: I joined the VM container team in 2004, and I think the story is interesting because Harald Kuck came to us, and he was proud, and he did the presentation. The thing was, everything you built, you added to the R/3 kernel, and it had to work on all platforms. Also, you couldn't do any adding of technologies which introduced platform restrictions. So, he presented the VM container and then he asked us: "You guys, you guys can port this. so, you don't create any problems for me." And we were all excited. And I remember I was sitting in that presentation, in Harald's presentation, and I really thought that I liked it, that I wanted to work there. And at that time, I was frustrated with the AS/400, because as a system, it was very interesting. But as a system it's as closed as it can get. It's the complete opposite of open source, and you couldn't look at anything. And so, I started using Linux a lot, and in Linux you can look at everything, you can look at the system libraries, you can look at the kernel and it's not even that hard. And I liked it.

Karsten: So, you basically had this awakening event or phase from going from the closed stuff to the open stuff? Kind of sounds like it.

Thomas: It took a while, honestly, it took a while. You could argue that I started to do a lot of work on Windows, and this is also closed source, but it's a tiny bit more open because at least you find a lot of information on the internet. There are a lot of people, so you can kind of peek into technology, you know, but with AS/400 there's one guy, Frank Saltis, who does in-depth articles about the AS/400 architecture and it's clustered with centimeters of patents. I also signed an NDA. I probably can't talk about any of that.

Karsten: Okay, then let's not talk about it.

Christoph: But I think for you, I mean, it was the journey from SAP's VM history, let's say. So, I mean, I stayed with the AS/400 team a bit longer, so, I made all those things, the incoming J2EE server, the VM container, then the J2EE server from the AS/400, so, in the beginning, we had an IBM-proprietary, AS/400-proprietary JVM. Then there was a switch to IBM J9, when IBM was rolling out the J9 technologies for our platforms. So, we switched technology under the SAP J2EE engine. And then there was the SAP JVM thing. So, then we did another migration project to migrate to the SAP JVM, and that was all at the time when open source and OpenJDK weren't there yet. So, I mean, I joined the SapMachine team in 2014, I mean, it was an SAP JVM team, so we had not yet founded SapMachine. So, SapMachine 10, so Java 10, was the first release we did.

Karsten: But I guess the work on the SAP Java activities was not reserved for AS/400 people, right? It kind of sounds like a topic where people came in from all directions, probably. Is that right?

Christoph: I think it's just coincidence that the two old AS/400 guys from our team are here.

Thomas: We actually do have four people from IBM, we had IBM platform porting background, Christoph and I are AS/400 people, and we have two people from this mainframe background. So, I thought that porting on the AS/400 gives you a perspective because you start realizing how similar today's operating systems actually are. And I think it makes you think about system design a lot. And maybe, maybe this is just really very conducive to JVM design.

Christoph: We also have compiler experts that we hired from university in our team, because, I mean, one of the tasks when we are doing SAP JVM on all these platforms – I mean, the JIT compiler is the thing which accelerates the Java code. And so, we had to have something for all those platforms where SUN didn't have an implementation yet.

Karsten: So, in this whole game of "There is an OpenJDK, there is a SapMachine, and so on", I assume SAP has also become a substantial contributor to the OpenJDK.

Christoph: Everything we do, we try to do in the OpenJDK first and see if it's good for all. And this made us quite a large contributor. OpenJDK is still dominated by Oracle. They have the biggest engineering task force there. They make probably about 80 percent of all the changes there. But there's a statistic which they publish every half year when a new Java release comes. And so, in this statistic, which counts the numbers of commits, we are always at about the third position. So, there's also RedHat. They are very engaged with the developers doing upstream changes. But then it's mostly us, some other folks like Amazon and other people. The project was now switched to GitHub. Maybe that's a reason for that. There are also more independent contributors, not from the two, three, four or five companies which heavily engage in the JDK technology. But here and there, there are contributors from some libraries. And they make contributions. And this also adds up to the amount of what we did.

Karsten: Are there any famous or noteworthy contributions by SAP that one could name here?

Christoph: We have a few that we should name here. So, I mean, in the beginning, it was our ports, as I said, where we did PowerPC, AIX port and IBM system Z. So, this is mostly then bringing the JIT compiler to work there. So, we contributed this as a JEP 175, then also something which was well received, which was maybe not a huge amount of code, but I mean, it's quite nice, it helps for the null pointer exception. So, maybe that's something for the developers of Java. When something goes wrong, you'll often get null pointer exception. Some references zero and there's no object. And yeah, you would not get a place in the codeline, or whatever, which would help. And so, we added that very early for SAP JVM, and we then brought this into OpenJDK. And then there was one large thing which Thomas did. I mean, there was the elastic meta space JEP 387 where especially Thomas put a lot of work in. I mean, they are also from experience and SAP workloads. So meta space is a special memory segment of the JVM, not where Java objects are stored, but some metadata. And nobody really took care of it, and it was complex. And so, yeah, Thomas dug into that and contributed to a revamping of it.

Karsten: Okay, so we have the famous developer of elastic metaspace here in the podcast today. No, I have no idea, of course, of how known or unknown this is out there. Let's turn away from technology, otherwise we'll dig deeper and deeper, and I'll understand less and less. Thomas, what's really SAP's business case for doing SapMachine? Does it pay off? Why do we do it open source?

Thomas: I think there are several answers to this, and depending also on whom you ask, and these are actually two questions. The first question is why do we even bother doing a JVM in-house at SAP? That's one question. And the other thing is: Why FOSS? I mean, also, why do we contribute upstream? And the first question we also kind of answered when diving into history: one thing is support. I mean, SAP sells a lot of business, and we may not be the cheapest on the market, but what we always promise our customers is support, and also a very long support. So, if they buy our stuff, then we help them protect that investment. And you have to guarantee that somehow. And so, of course, you could just download it from the internet and slip it on a CD or on a shipment and call it a day, but that doesn't really work. And so, it's better if you own or at least know the stack below you very well. So, and below the Java VM, below a Java program, you have the Java VM, and then you have the operating system and libraries and so on. And for us it's of course not feasible to own all that stuff. But JVM is somewhat special because when you have support worthy problems, they usually show up as cracks in the JVM, they manifest as problems on the JVM, regardless of whether you have a Java program error or whether their operating system is not doing what it's supposed to do, or even hardware problems, and JVM problems themselves, of course. So, it usually makes sense to have people who know the stuff and who can dive into this instead of relegating this to a vendor. So, that's one thing. And the other thing is, of course, the already mentioned SAP specific enhancement. So first, of course, very obviously the platforms, no one was doing this for us. So, we had to do this, and we wanted to have monitoring profiling, supportability, and so on. So, that's why we do it, and I think it makes a lot of sense. And if you do this, I'm actually convinced that at the moment you start to use an infrastructure heavily and you rely on earning money with that, it pays to keep developers in-house who develop this stuff if you can afford it, and if you can keep the talent.

And why FOSS? I think long term it's just a lot cheaper because with the SAP JVM, we were downstream from a very large codebase. I mean, 12 million lines of code, and we are a very small team, and we did a lot of stuff. So, you are downstream, and you start developing a lot of stuff. And the Delta to upstream grows. You have a lot of patches accumulated. And what we saw was that with the SAP JVM, the technical debt grew and grew, and the effort to merge upstream and downstream was extreme. Let's say you change something in a vital part of the JVM, like the garbage collector. And when you are done, it's fine. But at some point, you need to decide what you do now, you know, and so, you keep this downstream patch forever. And that downstream patch will cause costs forever because maybe someone changes the garbage collector upstream, too, so, you always need to make sure that your stuff still works, that it doesn't break upstream, and that upstream doesn't break your stuff. I think we ended up spending a large amount of all of our productivity only on these problems. That's my personal opinion. And so, therefore, when the OpenJDK came out, we started contributing changes upstream, like larger changes. I mean, that's one of the reasons why we started to contribute platform ports. By doing this, we made sure that people upstream will somewhat garden and maintain this stuff, and at least keep it in mind when they change things. And we didn't have a SapMachine at that time, but we had the SAP JVM, and the changes we contributed to the OpenJDK trickled down to us via the proprietary SUN JVM, and then back to us. So, this reduced our maintenance effort tremendously, I think. And so, for me, I think doing the SapMachine is almost like an afterthought. I mean, it's very good for a lot of reasons, but the most important thing is contributing upstream and reducing your maintenance cost. You can also cooperate with the community. If you do things downstream and keep it downstream, you can talk to no one. I mean, the code base is large, and everyone is an expert in some fields. If you contribute upstream, you suddenly have a much larger pool of people you can talk to.

Karsten: I would like to know now, Christoph: if one wanted to work for the benefit of SapMachine or wanted to contribute to the OpenJDK, should one get involved with the OpenJDK only, or would you recommend going specifically into something for a SapMachine directly? Where would you start?

Christoph: Yeah, generally, you should go for the OpenJDK. I mean, if you come and figure out a small bug or so that you can see in the source code or whatever, then the right place to address it, obviously, is the OpenJDK. And, I mean, once it's fixed there, we consume it in SapMachine more or less automatically. What we could do if somebody inside SAP or also from outside used the OpenJDK communication channels, we could try foster it, to help those people to get to, you know – I mean, you have to know a bit about the process. I mean, today, okay, it's a GitHub. I mean, it's quite easy, and many people know how to handle GitHub and open pull requests. So, then you sign a contributor’s agreement, but then you're free to discuss and push your change once it's accepted. So yeah, then that's really the way you should go. And for SapMachine we do this open source thing. We want to do everything in the OpenJDK. We only have very few devs for SapMachine when there's really something which is maybe not appropriate for the OpenJDK general purpose. But if we want to do something special for our special, let's say, supportability concerns, we have a little deviation here and there, but I think most things can be addressed in the OpenJDK.

Karsten: Okay. And I guess everyone will find the OpenJDK on GitHub easily. And apart from that, we probably also have some helpful links coming with the podcast publication now that we've addressed that. What would you like our listeners to take away as key points from this podcast?

Thomas: I'll keep it short. One thing: OpenJDK is a very good community for starting open source; they are very friendly, very professional, very welcoming to newcomers. And technical wise, it can be very difficult to get into. But I always liked that because – I don't know if I compared it with the kernel before, and we know that the kernel can be a lot more abrasive. You know, you need a thick hide and OpenJDK is nice. And the other thing is, JVM is a really interesting technology. If you want to learn modern optimization techniques, it's a good place to start. And it's also awesome because if you do something in the JDK and the OpenJDK, and it's good, then that literally gets used on a billion devices all over the world. So, this is a nice thought.

Karsten: Wow, okay. Anything to add to that? Anything more than being used by a billion users?

Christoph: That's a big motivation! I think the JDK technology, I find it very interesting, and then really to work on this open-source project which a lot of workloads all over the world are using. I mean, it's a good feeling if you make a change and everybody starts adopting that. Yeah. So that's definitely a takeaway, and also the community. I can agree with Thomas. I find it quite nice in the OpenJDK. So, the people there, I mean, sometimes maybe there are projects for which it's easier to go and drive by and do a contribution via GitHub. But I mean, if you take it seriously and if you have a thing, you will always find people who respond, they will try to sponsor you. So, I mean, I can just agree with what Thomas said.

Karsten: So, basically, if you're not afraid to be reviewed by thousands, and if you want to be used by billions, right?

Christoph: So, I mean it’s not thousands; I mean, if there are complex things, just find one or two reviewers...

Karsten: Okay. Or if you are not afraid to review the complex things, then also join. Thanks. Let's just keep it at that for now. Thank you very much, Christoph, thank you very much, Thomas, for being our guests today. Let's say bye together. Bye-bye. And thank you all for listening to the Open Source Way. If you enjoyed this episode, please share it, and don't miss the next one. We publish every last Wednesday of the month. Thank you again, Thomas and Christoph, and bye, bye for today.