

*jax*magazine

The digital magazine for enterprise developers

TECHNOLOGY NIGHTMARES

Horror stories for the brave

Does technical debt still matter in DevSecOps?

Plus some dos and don'ts for developers

A beginner's guide to Java programming nightmares

These are the error codes that haunt us

Don't use Jenkins Pipelines for CD

4 reasons why Jenkins Pipelines are brittle



Programming mistakes to avoid

New year, new JAX Magazine issue. If you're looking for something light after overindulging on Christmas goodies, this is not it. The new issue is not for the faint-hearted as it presents a list of technology nightmares that you should avoid and learn from.

First, let's debunk a popular myth: All developers make mistakes, no matter how experienced they are. There are always new mistakes to make but once you come to terms with this flaw that we all share, you'll be alright. If your New Year's resolution is to help others, you should write down your mistakes and share them with the community so that others can learn from them. This is exactly what our authors did so that you can enjoy a collection of horror stories about technical debt, automation traps, blockchain nightmares, data tsunamis, DevOps ordeals, container tragedies and more.

Mike Bursell explains why technical debt is not such a bad thing in the era of DevSecOps and goes into the reasons why technical debt still exists, how it can be useful and explores some basic dos and don'ts for developers. Steve Burton reveals why Jenkins Pipelines can be surprisingly brittle and how developers can get around this hurdle while Oren Eini focuses on

the automation trap and how not to fall into it. Just because a technology is popular doesn't mean it's suitable for every situation; if you're here for the nightmare scenarios that can arise from jumping on the latest bandwagon, you won't be disappointed. Speaking of popularity, as containers are gaining momentum, we need to understand that scanning a container in build time is an essential part of security hygiene. Failing to do so will result in ... Read Shiri Ivtsan's article to find out.

This JAX Magazine issue also contains ten professional red flags to watch out for, a beginner's guide to Java programming nightmares, courtesy of Georgi Minkov, and the scariest story of them all: What happens when you're on call and you discover you need to go bug hunting. Don't worry, though, Irit Shwarchberg will teach you how to mitigate those catastrophic failures and save yourself from a midnight bug hunt.

We've got more 'juice' to fuel this nightmare series so if your worst nightmare is not mentioned above, fear not; open the magazine and get ready to be spooked (and preferably learn from other developers' mistakes).

Gabriela Motroc, Editor

Index

Does technical debt still matter in DevSecOps?	4	Scan your container in build time	22
Mike Bursell		Shiri Ivtsan	
How wrong technology can cause nightmares	6	Does serverless spell the death of containers?	24
Richard Gall		Darren Royle	
Falling into the automation honey trap	8	10 professional red flags to watch out for	26
Oren Eini		Kostis Kapelonis	
A Blockchain nightmare	10	Beginner's guide: Java programming nightmares	30
Glenn Mariën		Georgi Minkov	
Banks drown in a data tsunami: Solutions	11	Learning from DevOps nightmares	33
Karen Krivaa		Brian Dawson	
The tangible costs of bad data	14	To catch a bug	35
Christophe Thibault		Irit Shwarchberg	
Best and worst Java 11 experiences	16	How to get (and stay) into the tech industry	36
Our Java Experts		Profile: Sherry List	
Don't use Jenkins Pipelines for CD	19		
Steve Burton			



HOT

Meet Corretto, Amazon's free distribution of OpenJDK

Amazon has entered the Java race with its latest offering: Corretto, a free distribution of OpenJDK that comes with long-term support. Not to mention that the tech giant has big plans (bigger than becoming Oracle's nemesis!) that include becoming the default OpenJDK on Amazon Linux 2 this year.

SuperGloo makes working with service meshes a breeze

If service meshes are something you'd like to explore but you find the experience too challenging, SuperGloo is here to make things easier for you. The goal of this open-source project is to manage and orchestrate meshes at scale. In short, it promises to simplify the installation, management, and operation of your service mesh.

ThoughtWorks Technology Radar: Istio and Knative are worth exploring

ThoughtWorks' latest Technology Radar, released in mid-November 2018, once again took it upon itself to assess the major trends in software development. There are a lot of hidden gems but our personal highlight is that service meshes are worth exploring. What's more, WebAssembly seems to be a strong language contestant and Visual Studio Code is a choice that won't give you headaches.

"Java risks setbacks if it becomes controlled by one or two mega vendors"

The news that tech giant IBM is acquiring Red Hat made waves throughout the industry in late 2018. In a conversation with Murray Rode, CEO of TIBCO about the impact of the acquisition and what the Java market has to lose if things go south, we learned that "Java risks setbacks if it becomes controlled by one or two mega vendors. It's quite difficult to make the argument that two [JCP] seats controlled by the same entity will benefit the Java ecosystem, consumers of Java technology, and the Java community at large."

Poll results: No love for React Native

Two months ago, we launched a poll to find out which technologies will dominate this year, which technologies will fall behind and what's going to stay static. As it turns out, respondents think that this year's biggest loser will be React Native, followed by Go. If you'd like to know what to focus on in 2019, the answer is easy: containers and serverless.



NOT

Some dos and don'ts for developers

Does technical debt still matter in DevSecOps?

Despite all the advantages of DevSecOps, challenges like technical debt still remain. In this article, Mike Bursell explains why this isn't such a bad thing. After all, identifying a problem is half the battle. He also goes into the reasons why technical debt still exists, how it can be useful, and explores some basic do's and don'ts for developers.

by Mike Bursell

Before we go any further, I should be clear about something: I'm a big fan of technical debt. If you're still reading after that revelation, I'm going to assume that you're either on the same page as me or so horrified by the idea that you want to learn more, so let me explain a little. First of all: a guilty secret. I love technical debt. My reasons aren't exactly altruistic, but they do play into part of what it means for me to be a security person. Technical debt will always keep me in a job, will allow me to poke my nose into interesting projects, and, well, there's lots of it. There's a better reason than those, however, which is that once technical debt is acknowledged, we're at least part-way to solving it.

Because, to be clear, if your project has technical debt (and here's a clue: they all do), and it's hidden, then it's useless. It's worse than that; it actually weighs on the project, as you can never easily know what you're missing, but could have addressed. If it's visible, however, things can change, and you can improve matters – if not now, then at least in the future. The reason for this comes down to the definition of technical debt. For me, technical debt comes in two forms:

1. Stuff that we didn't do, but which we should have done, and
2. Decisions we made that could have been better.

If we know what we didn't do, then we can do it later, when appropriate; if we know we made suboptimal decisions, then we can change them. We'll come back to that “when appropriate” below; it's important.

Technical debt and DevSecOps

We should also explicitly be talking about DevSecOps, because security is so often a hidden subject of technical debt. Let's define DevSecOps: my preferred understanding of DevSecOps is the putting of security at the center of the practice, processes, and delivery of DevOps through automation, tooling, and cultural change.

Here are some examples of security-related technical debt that I've seen over the years:

- Neglecting to put authentication on currently non-public APIs
- Lumping capabilities together
- Hard-coding roles to initial expectations without thought to future requirements
- Compiling cipher suite selections into the application

All of these could appear in projects using classic development methodologies or more agile ones like DevSecOps. In fact, I would argue that in many cases, projects employing classic development methodologies are actually in a better position to remedy technical debt than more agile or DevOps projects. This is because there is generally a well-defined product management role that can collect customer requirements between releases, rank them, and decide which to prioritize for the “next release”. In the DevOps world, it can be easy for difficult features – or difficult decisions – to fall to the bottom of the “board”. Unless you consolidate requests, a focus on per-sprint features can obscure debt. These problems are compounded if you have swallowed the simplistic line that “with DevSecOps, security is everybody's responsi-

bility”, and therefore have no security experts embedded in the team.

As with more traditional methodologies, it remains the case with DevOps that security is complex and often requires deep knowledge. Technical debt can accrue easily particularly if you are asking people who are not experts to make decisions that they are not well-equipped to make. To take an example from above: how should someone with little knowledge of security best practices know when it’s appropriate to compile cipher suites into an application and when it isn’t? Technical debt is a security issue because:

- Security concerns are often non-functional and are therefore less likely to be tracked.
- Deep knowledge of security is scarce as there are too few owners, architects, and designers who understand the impact of security.
- Security is often left till last, so it’s easy for it to drop off the bottom of the TODO list.

So, let’s delve a little deeper into why technical debt can be an asset to a project, and more specifically, to an open source project. As noted above, when technical debt is acknowledged, informed decisions can be made to remedy that debt or to leave it as is for the time being. This is the point of the “when appropriate” comment above: sometimes, and however unpopular a view this may be, security needs to come second to usability, or a particular timeline. Or maybe your team doesn’t have the requisite expertise or knowledge about your customers’ plans to make an informed decision on design or implementation at this point. This is fine, but you should record the decision and why it was made, because then you can revisit it in the future. What’s more, if this decision is recorded and the project is open source, it may be that somebody else who does have the resources, expertise, or knowledge of possible deployments will remedy the debt that you cannot currently manage.

A case study

It’s time for a case study of how technical debt can aid a project. Let’s consider that encryption was not included on a particular interface. What steps can we take to remedy this? The first is to document it, because then it can become known. There are at least four places in which we should document it:

1. In project documentation, e.g. “We ran out of time, existing reqs don’t include certificate management.”
2. In product documentation, e.g. “This API is designed to be used in a protected environment, and should not be exposed on the public Internet.”
3. In source code, e.g. in-line comments:


```
// 2018-05-02 (mbursell@redhat.com) Planned to use TLS 1.2
// (check for newer version) probably won't need client
// authentication. Don't use ECC cipher suites.
// Certificate management and revocation currently not specced.
```
4. In unit tests, e.g. a test that tests to see whether data is transmitted over the interface in cleartext, though it’s expected to fail.

DevSecOps means putting security at the center of the practice and processes.

How do these help? Well, the project documentation allows clearer requirements to be created, the product documentation allows customers to make informed decisions about how to deploy the application safely, and the source code documentation allows for simpler future implementation. The last – the unit test – may seem strange (who wants a test that’s known to fail in their project?). However, if you really want to ensure that a feature or piece of functionality is implemented, there’s nothing like a red light on a project dashboard to keep you honest. So, now we have documented known and useful technical debt.

Basically, what we’re trying to do is avoid blame (particularly when we’re the ones in danger of being the recipients). Good documentation should benefit multiple stakeholders, whether they are architects, designers, engineers, testers, operations, product managers, project owners, sales and marketing, support or customers.

What to do and what to avoid

Finally, what can you do to encourage a culture where technical debt is exposed and documented? Here are some “dos” and some “don’ts”.

Do:

- Encourage honesty.
- Document everything.
- Record “I suddenly thought ...” moments.
- Reserve time in every meeting for discussion of technical debt.
- Consider a project “debt recorder” role – which can rotate through the team.

Don’t:

- Allow a culture of blame, even after the fact.
- Assume that it’s better to hide technical debt from the customer.
- Put off creating unit tests just because a component hasn’t been implemented yet.



Mike Bursell joined Red Hat in August 2016, following previous roles at Intel and Citrix working on security, virtualization and networking. After training in software engineering, he specialized in distributed systems and security, and has worked in architecture and technical strategy for the past few years. Mike has been closely involved in the telco market and ETSI NFV since mid-2013, with contributions in the phase I SEC, INF and SWA groups, and is rapporteur for three security work items. He served for two years as the Vice-Chairman for the ETSI NFV Security Working Group, and has also been involved in the OPNFV project. He regularly speaks at industry events in Europe, North America and APAC. Mike has an MA from the University of Cambridge and an MBA from the Open University.

When the tech doesn't fit

How wrong technology can cause nightmares

Just because a technology is popular doesn't mean it's suitable for every situation. Whenever you get all excited about the latest new thing, it's important to rationally evaluate if you need to adopt it. In this article, Richard Gall goes over the nightmare scenarios that can arise from jumping on the latest bandwagon and how your organization can avoid them.

by Richard Gall

In the Wallace and Gromit animation "The Wrong Trousers", plasticine inventor Wallace creates a pair of mechanical trousers that allows the wearer to walk on surfaces that would otherwise be contrary to the laws of physics. Up walls, along ceilings – Wallace's trousers were a remarkable technological feat for a man with an already impressive track record of inventions.

The results, however, turn out to be pretty nightmarish. When Feathers McGraw (a criminal that also happens to be a penguin) steals the trousers for his own nefarious ends, everything begins to fall apart.

The Aardman animation is actually a great example of how the wrong technology can cause some pretty nightmarish scenarios. Of course, Wallace's trousers weren't intrinsically bad or disruptive. In the right situation, being able to walk on the ceiling could have been pretty useful. In the wrong hands, and deployed in the wrong way, technology can have pretty disastrous consequences.

Its impact can be felt in a number of ways:

- It makes life harder for people in the workplace – both engineers and non-technical people.
- It can impact customers and users that want – or need – to buy products or use information.
- It could open up new issues around privacy and safety – if information is lost or exposed.
- It can cost big, big money.

There are, broadly speaking, three ways in which the technology can cause significant problems:

- out of date technology
- poor migration planning
- moving to unnecessary solutions

Let's start by looking at what happens when you fail to keep up ...

Nightmare scenario 1: Out of date technology

The first is failing to keep up to date with changes and opportunities. This could be anything from failing to update legacy systems to pure complacency.

A great example of this is the continued dominance of Excel inside many organizations. Now, don't get me wrong, Excel is a perfectly fine tool for working with data – up to a point. But when you begin relying on Excel for managing huge datasets on products or customers, you are going to run into problems. At a basic level, it's a huge productivity nightmare – no one wants to upgrade their RAM just to open a spreadsheet. But it also could have repercussions for both security and governance.

Bob Katz, a former IT consultant, told me about a particularly bad experience he had doing work for a public mining company. "I was originally brought in as an Excel 'expert' to fix the earlier implementation but it was clear me that Excel was the wrong tool for the organization. Excel files were being distributed worldwide without version control, no reconciliation to the accounting systems, Excel errors etc.," he told me over email. Why was the company even using Excel? Katz explained that it was because "Excel is a 'common language' within the Finance organization, both at corporate and the operating units in Brazil." So, essentially, it's a question of comfort – and, to a certain extent, some ignorance about other potential options.

For Katz's client, the whole episode really was a nightmare. They spent "6 figures" on an "unworkable solution that had to be scrapped," only to later implement a SaaS solution that was, by Katz's account, a success.

"We were able to sync and reconcile with the company's Brazil operations to convert from Brazil GAAP to Canadian GAAP and eventually US GAAP and IFRS reporting standards, all within the same application. We were also able to report and analyze operational metrics as well. We were also able to put together multiple forecast and Budget versions variations of gold price, operational changes, headcount, capital planning etc. and compare to previous budget/forecast versions."

There are plenty of similar stories out there. Alan Santillan, who experienced struggles with Excel while working for a U.S.

healthcare organization, said, “Using Excel as an alternative to dedicated business software not only limits productivity, but also increases employee frustration and quality of life.”

Both stories are small scale stories about technological complacency leading to organizational nightmares. However, there are plenty of very public examples where complacency has been lethal. Just look at Blockbuster. This was a company completely unprepared for the impact of technology, only to be outfoxed and outmaneuvered by Netflix [1], today one of the most popular entertainment brands on the planet.

Of course, the Blockbuster scenario was much more complex than a frustrating inability to see beyond Excel spreadsheets, but nevertheless comes from the same kind of culture that refuses to be curious about technology. Ironically, this conservatism can’t stop change – it will cause significant internal pain or it will completely put you out of business.

Nightmare scenario 2: Moving to the wrong software

If getting stuck on the wrong software – or forgetting to even think about it – can cause problems, so can moving to the wrong software. This sort of scenario comes out of a completely different organizational culture – one where people are forced to make decisions ‘just because’ or because changing things is better than remaining stagnant. It’s often an attitude that develops out of a fear of the previous scenario. Rather than being complacent and conservative, curiosity and experimentation come to be valued for their own sake.

Many people have been in this sort of situation. Today, this is perhaps even more common than being hampered by conservatism. Many people have stories of CEOs reading about big data or microservices (if they’re a little more technically aware) and immediately slapping down requests on the desk of their CTO.

It’s not hard to find examples of this. Perhaps the best example was the NHS IT project, NPfIT, which was billed as the largest public IT project ever. The idea behind it was to centralize patient information, and in doing so making all aspects of the NHS much more efficient. Ultimately, it should have improved the patient experience. The frictions between medical institutions would have all but disappeared with this system. A noble pursuit, perhaps. Ultimately, the project failed simply because it was not implemented properly. According to a Guardian article from 2013 [2], this cost an eye-watering \$10 billion.

There were a whole host of reasons that the implementation failed, but the crux of the matter was a total lack of alignment at every point. From government to contractors all the way down to the doctors and patients it was meant to serve, the solution (which gradually developed into a complex Frankenstein solution with a lack of leadership and vision) was ultimately nothing more than a dream around which people were retrofitting solutions.

True, the challenge here was as much cultural and strategic as it was technical, but it’s important to remember that the strategy was ultimately defined by a flawed understanding of the software at the center of the project. There was a complete lack of understanding of what planning and support would be needed in order to make it work.

Often, that is precisely the nightmare. It’s not the hyped software that’s the problem; it’s the fact that the software becomes

shorthand for great solution! Causing everyone to ignore the various facets needed to adapt it to their own needs and goals.

The NPfIT is a great example of technological solution bloating like the decision-making infrastructure around it, but it’s also worth looking at something much more up to date. I’m thinking here of microservices, but you would probably find something similar in the case of other trends like artificial intelligence and blockchain (don’t get me started on the trouble blockchain can cause).

The problem with something like microservices is that they are actually incredibly useful. They are appealing to people who find themselves facing development challenges and tricky pieces of legacy IT. But they’re not incredibly useful to everyone.

It’s easy to see, however, how the decision to make the architectural change comes about. Say your current website is slow and your developers hate it. Making changes feels glacial – while it is making money and you seem to be growing in the right direction, you feel hampered – maybe like you’re sitting in nightmare scenario 1.

You read about the operational efficiency that microservices seem to offer; you find that all the Important People in the industry are talking about them. The whole concept just makes sense. Before you know it, you’ve sent your engineering team (which has now doubled in size because you need the resources) down a microservice rabbit hole, where just about every aspect of your business has been re-architected into a microservice that is plugged into your site. After 18 months of pain, it becomes clear that the whole project was pointless and a colossal waste of money.

You’ve landed yourself in a true technical nightmare and you’re not sure why. You should have taken Martin Fowler’s advice: “Don’t even consider microservices unless you have a system that’s too complex to manage as a monolith.”

Avoid nightmares: Be technologically mindful

In talking about all these nightmares, it’s easy to feel like you’re going to run into trouble whatever you do. To a certain extent, that’s probably true; it’s inevitable if you’re working with software. However, there is a middle path you can take [3] to avoid the sorts of situations like the ones mentioned above. This is a path that is always mindful and alive to the opportunities new technologies might offer, but is also attentive to business goals as well as the needs of people. If you can do this, you can avoid stagnation and also avoid diving down a rabbit hole from which you can’t escape.



Richard Gall is Communications Manager at Packt, utilizing his background in writing and publishing to help Packt in their mission to help the world put software to work in new ways. Packt, one of the biggest tech publishers in the world, deliver effective learning and information services to IT professionals.

References

- [1] <http://bit.ly/2B9li6X>
- [2] <https://www.theguardian.com/society/2013/sep/18/nhs-records-system-10bn>
- [3] <http://bit.ly/2zWkwjt>



© John Williams RUS/Shutterstock.com

What causes the automation trap?

Falling into the automation honey trap

Automating routine, boring tasks sounds great. Automation promises to rid developers of scutwork and let them focus on the meaty details. However, as Oren Eini explains, this is a snare that causes more problems than it solves.

by Oren Eini

Developers encounter numerous problems throughout their day. For every complex problem, there is a simple obvious solution that – while tempting – is completely wrong. For junior employees, it will seem even more obvious. However, it is only when they dive into the details that they realize the long way really covers less distance.

Every developer who has been in the business for a year or two encounters the issue of needing to solve the same problem again ... and again ... and again. The quintessential example is the admin section of a website. That part is usually basic crud and has very little logic or interesting stuff going on. In fact, the vast majority of business software in general can be seen

as displaying data from a database and storing it back again. This simple task can take up the majority of a developer's day. To shorten this time-consuming process, it can be tempting to look for an easy way out by having the computer do the leg work. Today, many are looking to artificial intelligence (AI) and machine learning (ML) to solve their problems.

... but it's a honey trap.

What causes the automation trap?

Developers, by their very nature, want to automate things, including many of the development lifecycle. Such automated solutions range from the Computer Aided Software Engineering (CASE) tools of the 1980s to the Microsoft Light Switch a few years back to the AI and ML frenzy of today. In between

Are AI and ML ready for prime time? Even though both can do amazing things, any efforts to use them to completely automate development processes are bound to fail.

each frenzy, there have been hundreds of attempts to automate developers out of a job. From my experience, this tactic has always resulted in a hefty cleanup bill and a big pile of technical debt.

I get it. It's tempting to try to pawn off boring and repeatable tasks to a machine versus an actual person. It can be done, and there are many situations where it is appropriate, however, the trick is that a lot of these mundane tasks are not fully ready for cookie cutter assembly line mode. With every application, there is going to be subtle differences in requirements from what the tools provide.

Instead of writing boring sequential code, users are faced with having to master the automation tool. And instead of tailoring everything to their specific needs, they have to use their customization hooks. These kinds of solutions do not play well with traditional software development practices; a great example is trying to build a business workflow inside Salesforce and then not having access to source control or unit testing.

While they can automate most of it, what's left leaves such a hole that the rest of your project seems to fall right into it.

A great example of this type of limited automation is SharePoint. It is widely deployed and promises to simplify day-to-day tasks, but as soon as something needs to be completed "out of the box," costs skyrocket. Although the tool can do 90 % of the required tasks, the hurdles associated with the last mile system typically outweigh the costs.

Is it RAD or is it BAD?

This is not to say that simplifying anything is a bad idea. It's all about balancing and understanding how much automation is too much.

Visual Basic, Ruby on Rails, and now Node.js are all platforms for rapid application development (RAD) that allow developers to quickly produce workable applications with minimum fuss. Essentially, these tools are geared toward developers: debugging, source control, continuous integration and unit testing are in the box.

Business application development (BAD) platforms, however, rarely address these concerns. In Microsoft LightSwitch, for example, a lot of the data is stored in XML files. When adding a modification to the application with BAD, a lot of stuff changes, making routine tasks such as code reviews absurdly difficult.

The historical track record of attempting to simplify development has typically fallen short. The first attempt to build a CASE tool goes back to 1968, with ISDOS. From ISDOS to LightSwitch to SharePoint and now AI, we have had a half century of attempts.

Will AI and ML change the rules?

Are AI and ML ready for prime time? Even though both can do amazing things (and are only getting better over time), any efforts to use them to completely automate development processes are bound to fail.

As any developer will say, a computer will do what you asked it to do, not what you meant for it to do. Add to the mix the usual complexities of software projects, software teams will still need to do a gut check that cannot be manufactured in a bot.

Even if AI gets to the point of human-equivalent intelligence, that would still not be sufficient. For developers, there is good news: You don't need to worry about being made redundant by a server in the cloud somewhere. The overall task of development isn't something that can be fully automated. The bad news? Large parts of it can be. Developers need to keep an eye on what is going on in the market and be able to bet on what technologies are going to assist (but not hinder) their productivity.

The best way to do that is to consider how such technologies and tools work in the overall workflow of development. For example, tools that don't play well with source control or code reviews are going to introduce a lot more friction than they save. A component that is used to handle logins, on the other hand, is a black box that is very well defined and don't really need to be tinkered with. It will save a lot of headaches.

When evaluating an automation tool, look at features such as source control, code review, debuggability, testability, ease of deployment and overall automation capabilities. Not having really good answers for these features means that the tool in question probably isn't meant for developers and it's better to avoid that particular automation honey trap.



Oren Eini, CEO and founder of Hibernating Rhinos [1], has more than 20 years of experience in the development world with a strong focus on the Microsoft and .NET ecosystem. Recognized as one of Microsoft's Most Valuable Professionals since 2007, Oren is also the author of "Inside RavenDB". He frequently speaks at industry conferences such as DevTeach, JA00, QCon, Oredev, NDC, Yow! and Progressive.NET. An avid blogger, you can also find him under his pseudonym as Ayende Rahien at <https://ayende.com/blog/>.

References

- [1] <http://www.hibernatingrhinos.com/>

Living in a blockchain world

A Blockchain nightmare

Blockchain's substantial successes shouldn't obscure the upcoming challenges it will face. Glenn Mariën goes over some of these issues that keep him up at night, from public buy-in to diversifying the blockchain developer pipeline.

by Glenn Mariën

I first encountered Bitcoin in the spring of 2009, when a friend sent me a link to Satoshi's whitepaper and suggested I take a look. Soon enough, I was up and running, zooming coins from wallet to wallet. I quickly realized that blockchain held the potential to transform the world and that Satoshi's dream was much more akin to an inevitable reality. From that, I was only steps away developing applications off the Bitcoin API and beginning a career on this newest electronic frontier. I'd found the breakthrough technology of our time. Through all the ups and downs of the past decade, I've never regretted building my career around the blockchain. It's hard to believe that the heady days of 2009 were only a decade back. It's often said that one month in crypto feels like a full year in another industry. It's true: Crypto has packed a full century of growth, maturation, and upheaval into just a few short years.

For all blockchain's successes – and here I could present you a long list of market caps, a full roster of institutional investors, and libraries full of press clippings – several obstacles remain in place. The first and most familiar problem is educating the public about the technology, its societal ramifications, uses, and even limitations. The distributed ledger is hardly intuitive, particularly if you're not familiar with traditional ledgers, and so more than once I've had to explain the basic principles of double-entry bookkeeping to lay audiences! But it has a second, less visible problem that keeps me up at night; the challenge of finding developers ready to meet the unique challenges of blockchain development.

A few years ago – or a few decades ago if you're operating in "blockchain time" – everyone was looking for Java and C++ coders. With the advent of Ethereum, tools like Hyperledger and Solidity became essential and blockchain programmers the world over began cramming new languages. The ledger is immutable; however, the tools we use to create it are anything but. While some of these changes have built a better blockchain, the constant adjustments create damaging uncertainty. Young coders find themselves faced with a dilemma: If they devote themselves to the mastery of one language, they run the risk it will become obsolete before they find a job and/or if they spend the same time learning the rudiments of several languages, their skills might prove too shallow for success.

We've known for decades (and here I mean real decades, not blockchain-time decades) that the United States needs better

STEM education. I was fortunate enough to receive a computer from my parents at the age of 15, setting me on my path towards an education in programming. The Asus A7N8X-E Deluxe motherboard on my computer pinged and beeped when I turned the machine on. As I waited for the CPU to finish booting, I reflected on if only I could control those chimes, I could make my computer play a song. I taught myself Visual Basic, wrote a motherboard melody, and never looked back. But would I have found my calling if I hadn't received that computer? Would I have been inspired to code if it had booted silent? Questions like these make me grateful for the work being done by programs such as Girls who Code [1], Kode with Klossy [2], and Tynker [3] that introduce teenagers – and girls in particular – to the power of coding. Coder Kids' [4] decision to run a blockchain course for the very youngest programmers is indicative of this. Should potential developers make it to college without code, there's a chance the universities will convert them. Top universities [5] now teach blockchain and hundreds of more courses will appear in semesters to come.

Blockchain's tenth birthday offers us a chance to reflect on the challenges to overcome and the setbacks faced on the amazing gains and the maddening losses. At times, the road to adoption has run uphill, but as we pause and look around, it's clear that the steepest part of the climb is behind us. Mainstream adoption for blockchain won't happen in the next year, but it will certainly grow by its end. By the time the children and teens now Tynkering or Koding with Klossy grow up and enter the workforce, we could, at last, be living in a blockchain world. What will you do to make that happen?



Glenn Mariën is the Founder and CTO of Metal. An early adopter of crypto, he previously developed backend systems for Bitcoin and other cryptocurrencies.

References

- [1] <https://girlswhocode.com/>
- [2] <https://www.kodewithklossy.com/>
- [3] <https://www.tynker.com/>
- [4] <https://coder-kids.com/courses/>
- [5] <https://uk.pcmag.com/why-axis/117172/cryptocurrency-education-is-on-the-rise-at-colleges-and-univ>



How to stay ahead of the data tsunami

Banks drown in a data tsunami: Solutions

In order to stay ahead of the data tsunami they need to deal with nowadays, banks need a system architecture that supports ingesting, processing and analyzing huge amounts of data, or else they risk drowning in the flood of big data.

By Karen Krivaa

Banks are buckling from the strain of processing huge amounts of data. Online banking apps that were designed to provide 24/7 access to accounts are crashing, creating customer frustration while tarnishing banks' reputation. Personalized services that are needed to compete with fast moving Fintech companies are dependent on fast data analytics and are being adopted too slowly. Additionally, regulations are getting stricter, and fraud and risk are growing. To stay ahead of the data tsunami, banks need a system architecture that supports ingesting, processing and analyzing huge amounts of data, or else they risk drowning in the flood of big data.

Internet banking disasters

Banks across the globe are experiencing technical difficulties, robbing customers of access to their internet banking and phone apps.

Here are three examples from earlier this year. Toronto based TD Bank [1] customers had difficulty accessing their accounts online for over a week due to processing delays, after a previous outage of banking apps just a few months before. When TSB migrated its customers [2] from Lloyds Bank to its new owner, Spanish bank Sabadell last April, close to two million customers were locked out of their online banking accounts, and one was even mistakenly credited with £13 000. And in September, Barclays bank's [3] mobile and telephone services crashed, leaving furious customers

locked out of their accounts for several hours due to a ‘technical glitch’.

And even when users aren’t blocked all together, they often suffer from slow response times. When screens take too long to load, or the system pauses too long after a click, customers give up. Eighty-seven percent [4] of banking and finance leaders reported abandoning an app due to poor performance, according to a recent survey.

In addition to maintaining digital banking services that are up and running effectively, banks are under pressure to launch new services to stay competitive. This requires fast access to customer data, third party data and real time analytics. Many banks offer cash for clicks, including online mortgages and car loans that require ingesting customer account data and credit ratings and other third-party information to calculate the risk, price accordingly and receive the necessary approvals.

Communications also are becoming automated to improve the quality of customer service. Bots have been implemented to provide quicker responses to customer inquiries on the phone, by email and text and even using social media. Both American Express and Wells Fargo have already introduced bots to communicate with their customers using Facebook Messenger. All these applications require quick access to customer data, and many are based on machine learning and need to feed in data from past interactions to constantly improve the quality of their responses.

Simpler architectures for lower risk and increased efficiency

One of the challenges in processing data is that the speed of inputting data is quite often much faster than processing. This problem becomes even more pronounced in the context of Big Data, where the volume of data and the requirement for new insights keeps growing.

The difficulty with the traditional multi-tier architecture is that processing, data management, analytics and presentation are in different layers, which can create delays or latency when processing the data.

One possibility is to speed up processing by running all processing, data management, analytics and presentation on a unified platform – in memory, but this can be expensive and may be limited by the capacity of data required. By integrating intelligently with data lakes, which store multi-petabytes of data, access to historical data is simplified and accelerated via smart in-memory indexing. This powers faster and smarter machine learning insights – enriching the data in the unified fast layer with historical data from a single application.

Another level of efficiency can be achieved by embedding machine learning analytics with the data. With this architecture data aggregations can be created before analysis occurs, and if an analysis is cut off mid-stream the calculations can resume where they left off, greatly accelerating the analytic process to reach sub-second latencies. This type of architecture can result in a significant reduction in processing time, speeding up large batch analysis, reporting and other critical business operations from hours to minutes, from minutes to seconds or from seconds to sub-seconds.

The simplified architecture not only enables banks to ingest more data faster, but also reduces total cost of ownership and data movement complexity by radically minimizing the number of ‘moving parts.’

ROI of fast data

Even though the price of RAM is decreasing, the flood of big data can force banks to explore more cost-effective ways to utilize in-memory computing.

To reduce the total cost of ownership, hot data selection allows RAM to be reserved for the data that is the most critical while saving other data to a multi-tiered more cost-effective data storage.

When customizing the hot data selection, important queries and data can be predefined and prioritized to ensure fast and predictable results according to business goals. If there are no clear priorities defined the most popular queries can automatically be flagged as hot data. For example, a common query such as “what is my current balance” can have immediate results.

This hierarchy of data allows for quicker loading and processing while enabling higher availability and stability.

Reader takeaways

Providing flatter system architectures where the data is closer to the analytics and closer to the business logic will speed up transactions for applications including online and mobile digital banking apps, bots that communicate with customers, personalized banking services, intraday risk analysis, fraud analysis and instant loans that need to analyze internal data and information from third parties.

The banks that will come out ahead despite fierce Fintech competition, will be those that can avoid disastrous computer crashes, have a robust flexible data information backbone to launch and maintain new services, while providing fast response times for their existing apps to keep their customers connected and happy.



Karen Krivaa's value to GigaSpaces comes from a solid high-tech foundation and over 20 years of marketing experience in product and portfolio solutions marketing. With her dynamic energy, executive vision and belief in “team”, Karen executes data-driven strategies while strengthening the GigaSpaces brand. She is responsible for driving strategic marketing planning and execution, awareness and customer acquisition efforts across all channels and branding. Prior to joining GigaSpaces, Karen held leadership positions in companies including RADVISION, Alvarion, NICE and Applied Materials.

References

- [1] <https://www.bizjournals.com/philadelphia/news/2018/03/08/td-bank-online-outage-march-braca-gartner.html>
- [2] <https://www.computerweekly.com/blog/Fintech-makes-the-world-go-around/TSB-IT-system-chaos-as-bank-cuts-rope-from-Lloyds-systems>
- [3] <https://www.telegraph.co.uk/business/2018/09/20/barclays-internet-banking-goes-offline-angering-customers/>
- [4] <https://www.cutimes.com/2017/10/19/mobile-apps-failing-many-fis-due-to-poor-performan/?slreturn=20181021080905>

**VERY
EARLY BIRD**

Register by
Feb 21 and
save **£ 200!**



May 14 – 17, 2019 | London

Expo: May 15 – 16, 2019

**The Conference for
Continuous Delivery,
Microservices, Docker & Clouds**



Agile &
Company
Culture



Cloud
Platforms



Container
Technologies



Continuous
Delivery &
Automation

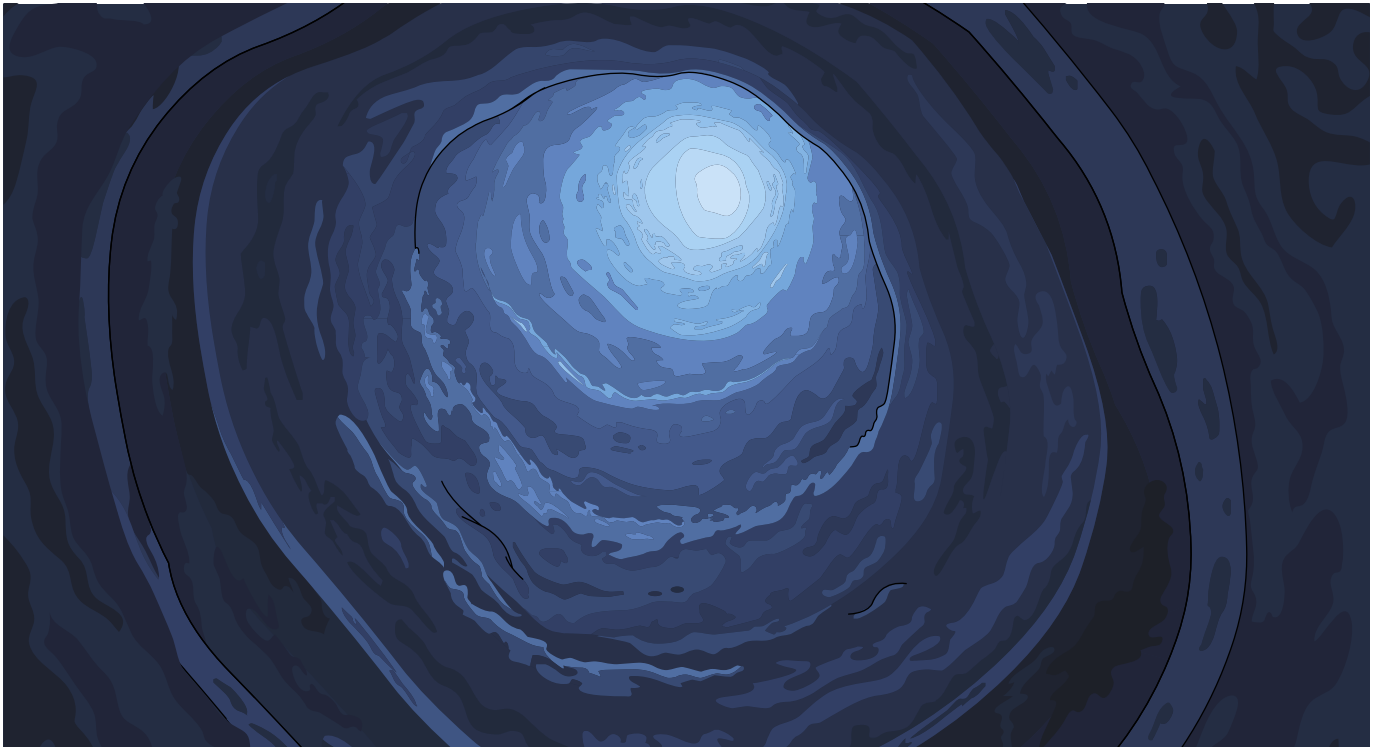


Microservices



Core Java &
Programming

devops.jaxlondon.com



© Pushkin/Shutterstock.com

Big data can mean big business

The tangible costs of bad data

Digital advertising relies on good insights from users. But how can a mobile marketing strategy be effective with 'bad' data? Christophe Thibault explains how developers can harness the power of big data in their digital marketing strategies without running into the nightmares of broken data.

by Christophe Thibault

For any organization looking to succeed on mobile today, harnessing the power of data can improve customer acquisition and retention by enabling personalization, ultimately offering a better customer experience. The effectiveness of a data-based strategy is achieved through high quality, first-party data, which maps the entire mobile user journey and fuels purpose-built AI technology to deliver business results. But what happens when the data is 'bad' or questionable? This could lead to an absolute marketing nightmare and could consequently advise some horrific business decisions based on inaccurate or circumstantial facts. Data is considered by some as 'the new oil'. However, much like relying on

low-quality oil, it can lead to more problems than you may have bargained for if used incorrectly.

The tangible costs of bad data

Bad data provides an incomplete view of the mobile user journey, as organizations only see what happens inside their own apps, sites, or ecosystem. In many cases, the increasingly complex and siloed range of tech solutions on offer today has often led to poorly integrated data, insights, and activation. Aside from this lack of quality data, consumer unease towards the lack of transparency around how their data is both collected and used makes it impossible for marketers to run effective campaigns without fully taking into account users' privacy.

Data fuels the algorithms that led to great AI, making it a powerful tool to build brand loyalty and meet users' needs. However, if the data isn't analyzed in depth, it could lead to companies spending money on campaigns that don't appeal to their consumers.

Data fuels the algorithms that lead to great artificial intelligence, making it a powerful tool to build brand loyalty and meet users' needs. However, if the data isn't analyzed in depth, it could lead to companies spending large sums of their budget on marketing campaigns that just don't appeal to their consumers – or simply target the wrong people. In fact, IBM estimate that mismanaged bad data costs the U.S. economy alone \$3 trillion per year [1].

A thorough analysis of credible data can uncover consumer patterns and reveal issues that consumers might be facing that might otherwise be ignored. These issues can include things like cart abandonment, product defects, language used for marketing or the time that campaigns are posted, all of which ultimately affect conversion rates negatively. It's crucial for brands to be able to understand their needs and cater to them accordingly. This is especially true in a continuously evolving marketing landscape, where consumers are used to on-demand and personalized services.

Most would think that it's all about quantity over quality, as the term 'big data' implies. However, if you're targeting certain users because they visited a website or an app on an ad hoc basis, you won't achieve optimal results as you only see a snippet of their journey and not the full picture. For example, if you happen to be discussing the latest Tesla with a friend and look it up on the official website, does that necessarily mean you are in the market for a new Model S? Analyzing the rest of your mobile journey might well reveal otherwise.

Consistency and visibility over the complete user journey is key here. To achieve that, you'll need transparent, credible, and consented first-party data. This is the only way to access the end-to-end, fully integrated insights that deliver the highest returns from digital investments. But on mobile, brands only see the part of the consumer journey that happens in their own apps, mobile sites, and ecosystem. There is no way for them to see the full user journey, with 95 % of a mobile users' journey remaining a mystery. Even the giant walled gardens of Facebook and Google only see a third of a user's time on mobile. This means that offering a meaningful and personalized mobile experience is reliant on educated guess work at best.

The same goes for the data your CRM receives. As you only see your own first-party data, you don't know the mobile habits of consumers across the remaining 4 hours and 59 minutes a day that they aren't on your app or site. If you are using third-party data to fill some of these gaps, chances are you're not collecting them from the most reliable sources.

Avoid the nightmares with Mobile Journey Marketing

So, just how can your business wake up from this 'data nightmare'? The solution lies within Mobile Journey Marketing (MJM), a new discipline of marketing that enables organizations to understand the entire mobile user journey, and to market across it. MJM is founded on three basic principles – effectiveness, simplicity, and transparency.

Effectiveness is achieved through high-quality, first-party data, which maps the entire mobile user journey and fuels purpose-built AI technology to deliver business results. Simplicity is attained by breaking down existing silos in which current tech platforms operate and integrating technology in one place by using purpose-built AI to remove unnecessary and tedious human labor. Transparency is brought about through explicit and unambiguous user choice management; from opting-in to exercising their right to be forgotten.

Don't let bad data break your business – GDPR is also a great opportunity to deal with this problem once and for all, especially when it comes to marketing. Explicitly asking users for consent will ensure that your marketing team will be able to provide more sophisticated, targeted messages and engender consumer trust by delivering what they actually want to see. With targeted marketing based on consented user data, there's a chance to truly move the needle on the idea of adverts becoming more like personalized recommendations and less like annoyances to be suffered through. In that sense, this is not only an opportunity for businesses to deal with poor data, but also a responsibility to finally change consumer perceptions towards advertising.



Christophe Thibault is the Chief Algorithms Officer at Ogury [2].

References

- [1] <https://hbr.org/2016/09/bad-data-costs-the-u-s-3-trillion-per-year>
- [2] <https://www.ogury.com/>



© Pablo Prat/Shutterstock.com

Manual on JDK 11

Best and worst Java 11 experiences

We invited eight Java experts to share their best and worst experiences with Java 11, the hacks they discovered so far, their tips and tricks and more. By the time we're done with all the parts of this series, it should look like a manual on Java 11.

Before Java 11 was released, we talked to eight Java influencers about the migration to the latest version and whether it's a "worthy" migration milestone even if it doesn't have "killer feature to drive adoption," as Martin Thompson opined in the interview series. If you haven't migrated yet, we thought you'd like to know the pros and cons of this new version.

We invited eight Java experts to share their best and worst experiences with Java 11. Since this series is meant to be a manual on Java 11, our interviewees will also talk about their first impression of Java 11, the tips and tricks, the cleverest hacks and inevitably, if they really care about Java 11 –more than they cared about Java 8, at least. Our interviewees will share the following:

- First impression of the new release
- Tips and trips for "navigating" JDK 11
- Cleverest hack discovered so far
- How much they care about JDK 11

JAX Magazine: What are your best and worst experiences with Java 11?

Mala Gupta: Using HTTP Client to work with non-blocking asynchronous code and using annotations with local variables for lambda parameters were one of the best experience using JDK11. Experimenting with different combination of configuration values with Epsilon, a no-op GC, was super exciting. It was also challenging to replicate the values.

“There are some possible inconveniences for those users depending on the deprecated bits in Java 11, including the Nashorn JavaScript engine, the CORBA modules, the Pack200 Tools and API and so on.”

New Java learners (in one of my workshop), were confused with launching single source code programs without compilation. The launch didn't work if they compiled the code by mistake or moved to multiple source code files.

Josh Long: This release has very little for me as a programmer – that is, very little has changed for me in my day to day experience writing code. Sure, being able to prefix lambda parameters with `var` just like I can already redundantly prefix them with their type is kind of nice. But, I've not yet needed it. I think the biggest change, as a programmer, is the new standardized reactive HTTP library [1]. But, of course, there are already good reactive HTTP libraries in the ecosystem so this shouldn't be seen as the main reason to upgrade.

I also like that this is an LTS release, to remove the last roadblocks for those customers hesitant to move beyond Java 8.

Those are nice features, to be sure, but not candy – nothing that'll make my code markedly easier to read and nothing that fills a gap I can't otherwise fill. Java 11 has a few nice goodies from the security perspective. I think the really quick turnaround integration of TLS 1.3 is amazing. TLS 1.3 was finalized in March of 2018!

The new support for ChaCha20 and Poly1305 cryptographic algorithms is a welcome addition. Good job, Java team!

I don't really see any downsides for me, in my enterprise Java-centric world, but there are some possible inconveniences for those users depending on the deprecated bits in this release including the Nashorn JavaScript engine, the CORBA mod-

ules, the Pack200 Tools and API, and so on. I also like that this is an LTS release, to remove the last roadblocks for those customers hesitant to move beyond Java 8.

Lukas Eder: After the initial trouble adding JAXB as an optional dependency for JDK 9 and 10 support, things have gotten even hairier now that JAXB has been removed from the JDK in JDK 11. While it isn't too difficult to get dependencies right, it does cause our jOOQ users some extra trouble to configure when setting up jOOQ for the first time.

Luckily (for jOOQ), we're not the only library that causes these problems to their users, so it doesn't look like it's our fault. I'm still working on the correct Maven dependency declaration for jOOQ that works on all of JDK 6 – 11.

Don't get me wrong though. I really think JAXB shouldn't be part of the JDK and its removal is a good thing (just like CORBA's removal).

Matthew Gillard: Best experience: Better startup performance than Java 9 and 10.

Worst experience: Startup is still comparable to Java 8.

Marcus Biel: First port of call is certainly the OpenJDK project page, where the 17 JEPs of the Java 11 release are listed. At first glance, Java 11 seems to offer mainly changes “under the hood” and no real developer features. However, this isn't the case. The String API enhancements (repeat, isBlank, strip, lines) [2] for example, I think are pretty cool. For me, it's often the little things that make our lives as developers easier. Up until now, we've had to access an external library like commons-lang for such simple things like “isBlank” – great that we can save that in the future!

Last, but not least, I think it's always good when obsolete features are thrown overboard – like the Java EE and CORBA modules or Java Web Start. I think it's always good when obsolete features are thrown overboard.

Trisha Gee: The best thing is that with the 6-month release cadence, each release is small. This means:

- a) much easier to see what's in it and
- b) smaller impact when upgrading.

Simon Ritter: I don't have a worst experience, using JDK 11 has been very positive so far. Admittedly, I don't use any of the modules removed from the core libraries (the java.se.ee aggregator module [3] and its constituents), so my applications have just worked without any changes.

Manual on Java 11

Now that we've revealed the pros and cons of Java 11, let's focus on the other important facets of this release. In the next parts, we will talk about the experts' first impression of Java 11, the tips and tricks, the cleverest hacks and inevitably, if they really care about Java 11 – more than they cared about Java 8, at least.

Read the rest of the Manual on Java 11 series on JAXenter.com:

1. Does Java 11 tick all the right boxes? [4]
2. Cleverest hacks to simplify your Java 11 navigation [5]
3. One small step for the new Java release train, one giant leap for Java 8 users [6]

The best experience for me would be using the launch single-file source-code programs (JEP 330). The shebang support finally makes it really easy to write a simple application and not have to go through several steps to get it to execute. I also really like the small change to the Predicate interface that adds the `not()` static method. This aids code readability in a number of situations.

Tal Weiss: I think one of the best new features is the new standardized HTTP Client API, that's been incubating since JDK 9. It's undergone many changes in its implementation since JDK 9, through JDK 10, and it's finally been standardized in JDK 11. My experience with the new API is great. It feels well written and efficient, and it's good to finally have a standard HTTP client API in the runtime.

tl;dr

• Pros

- + using HTTP Client to work with non-blocking asynchronous code
- + using annotations with local variables for lambda parameters
- + the really quick turnaround integration of TLS 1.3
- + the new support for ChaCha20 and Poly1305 cryptographic algorithms
- + better startup performance than Java 9 and 10
- + String API enhancements (repeat, isBlank, strip, lines) are pretty cool

- + obsolete features are thrown overboard
- + smaller impact when upgrading
- + using the launch single-file source-code programs
- + the small change to the Predicate interface that adds the `not()` static method

• Cons

- launching single source code programs without compilation is confusing
- has very little for programmers
- possible inconveniences for those users depending on the deprecated bits in this release
- JAXB has been removed from the JDK in JDK 11. It causes jOOQ users some extra trouble to configure when setting up jOOQ for the first time
- startup is still comparable to Java 8

References

- [1] <http://openjdk.java.net/jeps/321>
- [2] <https://4comprehension.com/java-11-string-api-updates/>
- [3] <https://docs.oracle.com/javase/9/docs/api/java.se.ee-summary.html>
- [4] <https://jaxenter.com/manual-java-11-first-impression-part-2-150153.html>
- [5] <https://jaxenter.com/manual-java-11-series-part-3-150308.html>
- [6] <https://jaxenter.com/manual-java-11-series-part-4-150512.html>

Our Java Experts



Marcus Biel is a speaker, author, JCP member, and Java Clean Code Evangelist

@MarcusBiel



Lukas Eder is the founder and head of R&D at Data Geekery GmbH, the company behind jOOQ and a Java Champion.

@lukaseder



Trisha Gee is a Developer Advocate at JetBrains, a key member of the London Java Community and a Java Champion.

@trisha_gee



Matthew Gillard works for Oracle in Bristol (UK) on Open-Source, Java and Serverless/Cloud infrastructure.

@MaximumGilliard



Mala Gupta is the founder of eJavaGuru.com. She is a Java Champion, book author, speaker, and co-leader of DelhiJUG

@eMalaGupta



Josh Long is the Spring Developer Advocate at Pivotal. He is the author of 5 books and 3 best-selling video trainings. He is also a Java Champion.

@starbuxman



Simon Ritter is the Deputy CTO of Azul Systems.

@speakjava



Tal Weiss is the CTO and co-founder of OverOps.

@weisstal

Four reasons why Jenkins Pipelines are brittle

Don't use Jenkins Pipelines for CD

Jenkins may seem all-purpose, but it's really not. Just because you think you can use Jenkins Pipelines for continuous delivery doesn't mean you should. Steve Burton explains why Jenkins Pipelines can be surprisingly brittle and how developers can get around this hurdle.

by Steve Burton

It's amazing how many companies have tried to execute continuous delivery using nothing but Jenkins Pipelines, and as a result are in a world of pain. Just last week I was on a call with one of our customers about what life was like trying to use Jenkins Pipeline for a 20+ microservices application.

The customer said, "It was painful – actually, it was a lot of pain."

I've heard another customer describe Jenkins as "DevOps duct tape."

Jenkins documentation defines Jenkins Pipeline as a suite of plugins that support implementing and integrating continuous delivery pipelines into Jenkins. And this leads us nicely into the first reason why Jenkins Pipelines are brittle.

1. Jenkins Pipeline relies on (lots of) plugins

Now, before you slap me with a wet fish, allow me to explain – because I love a good SDK just like the next developer. Extensibility is a good thing in software. BUT it can also be a bad thing when extensibility isn't managed or structured properly.

For example, let's imagine I have a simple Docker application and want to build a simple deployment pipeline using Jenkins. I go to the Jenkins plugin site [1] and search for "Docker" plugins and get figure 1.

It's 26 different plugins related to Docker, sorted by relevance. In fact, someone was even kind enough to name their plugin "Yet Another Docker," which is about as helpful as a poke in the eye with a stick.

Maybe I'm being overly dramatic here, so let's just click on the first plugin called "Docker" and see what's what (fig. 2).

Unfortunately, at a second glance we hit the second reason on why Jenkins Pipelines are so brittle; I spot eight plugin dependencies with another seven optional dependencies. You can see where I'm going with this.

In a world where apps and services change more frequently than Tesla's stock price, it seems logical that plugin dependencies could be problematic as technology stacks and plugins naturally evolve. The last thing you want is a broken deployment pipeline because the pipeline itself is broken vs. the actual software artifact or build that's being tested.

Simply put, which Jenkins Pipeline plugin should one use? And what happens when one needs to upgrade plugins and maintain version dependencies?

Do I need a plugin for DockerHub, Docker, Kubernetes, and every other technology stack or tool my app interacts

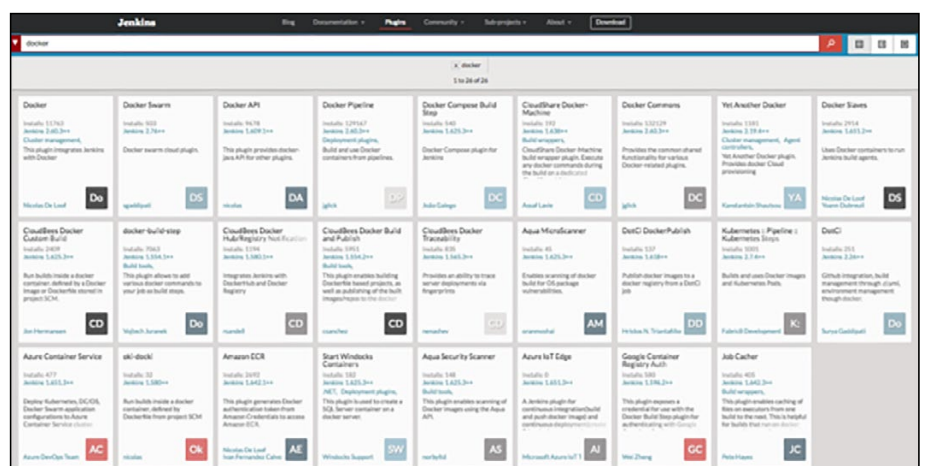


Figure 1: Docker plugins on the Jenkins website

Jenkins is unquestionably top dog for continuous integration use cases. But continuous delivery is way more than the building and testing of code into artifacts. CD is about taking those artifacts into production where they will delight the customer.

with? How many plugin dependencies do you think this list of plugins would have? Over 100 isn't unrealistic.

2. Jenkins Pipeline relies on jobs/scripts

Let's put aside plugins for a second, and focus on the core concepts or entities of a deployment pipeline:

- Build or Artifact (e.g. containers, AMI, Function)
- Application/Service
- Environment
- Variables & Secrets
- Deployment Workflow
- Stages
- Steps
- Trigger
- Approval
- Verification or Health Check
- Rollback
- Release Strategy (Blue/Green, Canary, etc.)
- Tests & Tools (test, security, monitoring, etc.)
- User Groups & Users (RBAC)

How many of the above actually exist today as first-class citizens in Jenkins Pipeline? More importantly, how many of them require you to write Jenkins Jobs or Shell Scripts to manage or perform the above tasks? – Answer: nearly all of them.

A Jenkins Pipeline is another way of saying: “I hardcoded my deployment pipeline with scripts.” Hardcoding things like environments, variables, secrets, dependencies and release strategies would have worked back in 2008 when all you had

was an apache web server, a few instances of tomcat/weblogic/websphere, and a chunky Oracle database. However, it doesn't work in 2018 with public cloud, containers, and microservices. You're going to spend more time maintaining your deployment pipelines than actually deploying new versions of your app.

In fact, it's not uncommon for customers to have a team of DevOps engineers solely focused on maintaining deployment pipelines as applications, technology stacks, and tools change every week. Instead of DevOps teams adding innovation to deployment pipelines, they spend their time fixing or dealing with pipelines that break. This is not good.

The more you script the core components of your deployment pipeline, the more you maintain those scripts as applications and technology changes.

You want your deployment pipelines to be dynamic in the sense that they can automatically adapt or change based on the meta-data that exists in your cloud provider or DevOps tool ecosystem.

In addition, the majority of Jenkins Pipelines I've seen still prompt the user for manual inputs like “Please provide the Major and Minor build version you would like to deploy.” Why can't this information be dynamically polled from the build or artifact repository?

Shell scripts by nature will make your deployment pipelines brittle. The more complex your deployment pipeline, the more brittle your pipeline will become.

3. Debugging a Jenkins Pipeline is a PITA

How about we ignore those pesky plugins and sexy scripts for the time being? Let's imagine we've hard-coded the best deployment pipeline using Jenkins Pipeline for our application that has 20 microservices, with each microservice having a dev, QA, staging, and production environment.

Quick question: Do we need one Jenkins Pipeline or 20 pipelines for our application? And can those pipelines be executed in parallel?

Anyway, let's imagine we push our big red “Build Now” button to kick off our deployment pipeline and get the visual shown in figure 3.

The above screenshot doesn't look too bad at first glance. We can see our Jenkins Pipeline has 10 steps, and they're all green. But notice the long list of “Shell Script” executions, and specifically the lack of context those windows have. What exactly are those shell scripts doing and telling the user who is watching the deployment?

What if those scripts fail? Are we able to understand in detail what happened? Or are we simply just seeing a console confirming that a shell script executed?

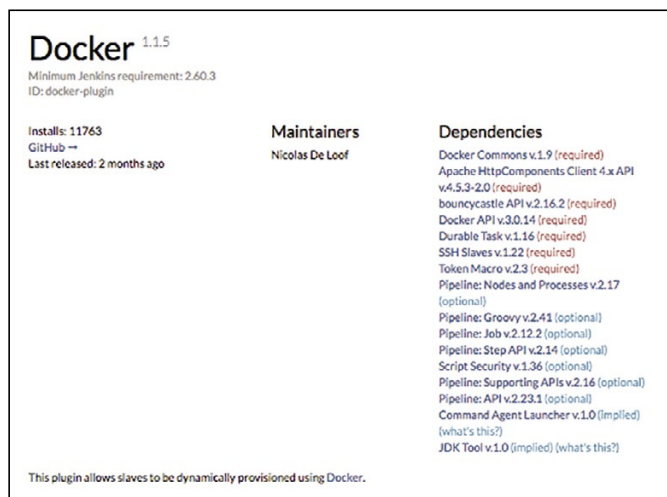


Figure 2: The Docker plugin

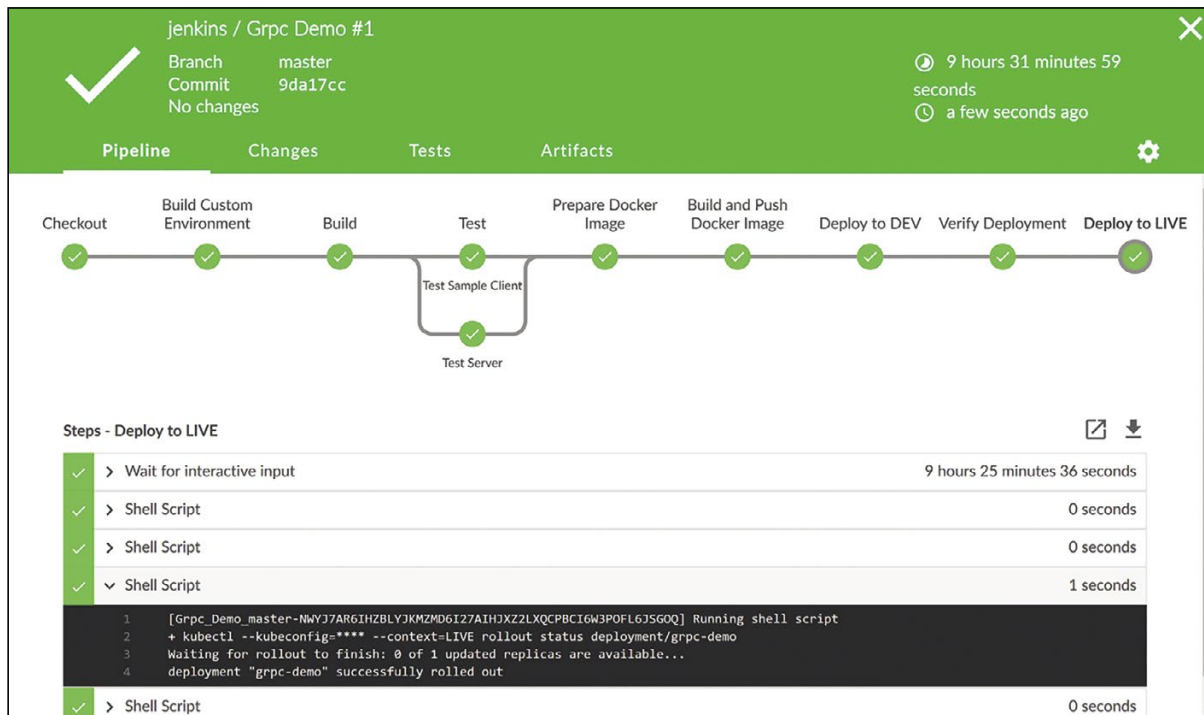


Figure 3: A Jenkins Pipeline example

Again, not a massive problem if our application is relatively simple with a few microservices/environments. But this could be a world of pain if we have to observe 20 different pipelines, each with hundreds of shell script outputs.

Understanding the real status and health of a Jenkins Pipeline (and your application) isn't an easy task. One customer told me, "Jenkins Pipeline really lacks any sort of global context, dependency model, or high-level view of a deployment pipeline in a microservices world."

4. Deployment verification and rollback

Simply put, continuous delivery is not just about deployment. Everyone does continuous delivery for a reason, normally to increase the velocity of your innovation and business so customers spend more wonga [2].

Not knowing the business impact of a deployment in production [3] is a big deal. Not having a rollback strategy is also a big deal.

I'm not talking about running a Jenkins Job to run a simple load test or unit test. I'm talking about observing how customers are impacted by a production deployment and managing that risk in real time. At my company, we call this Continuous Verification [4] and Smart Rollback.

To achieve this in Jenkins Pipeline you have to write a Job or shell script. Why can't Jenkins Pipeline just integrate with your monitoring ecosystem and automatically tell you whether your deployment was successful? It's possible. Just remember that knowing the success of a deployment isn't a deployment step or job completing. Rather, success is a deployment having a positive impact on your business.

Conversely, a deployment having a negative impact on your business makes your deployment pipelines brittle. It leads you into a false sense of security that everything with your deployment or application is OK – when actually it's not.

CI != CD

Look, I admit it: Jenkins is unquestionably top dog for continuous integration (CI) use cases. But continuous delivery is way more than the building and testing of code into artifacts. Continuous delivery is about taking those artifacts into production where they will delight the customer.

Remember, Jenkins Pipeline is just a suite of plugins that allow you to build your own CD platform. This means that you're actually the one coding and maintaining your CD process as your applications and services evolve.

Today, many real continuous delivery solutions exist that integrate with and complement Jenkins for CI. Harness [5] is one, but if you search you'll find others. Try them! Don't be that customer or team that thinks building your own CD platform is doable with DevOps duct tape.

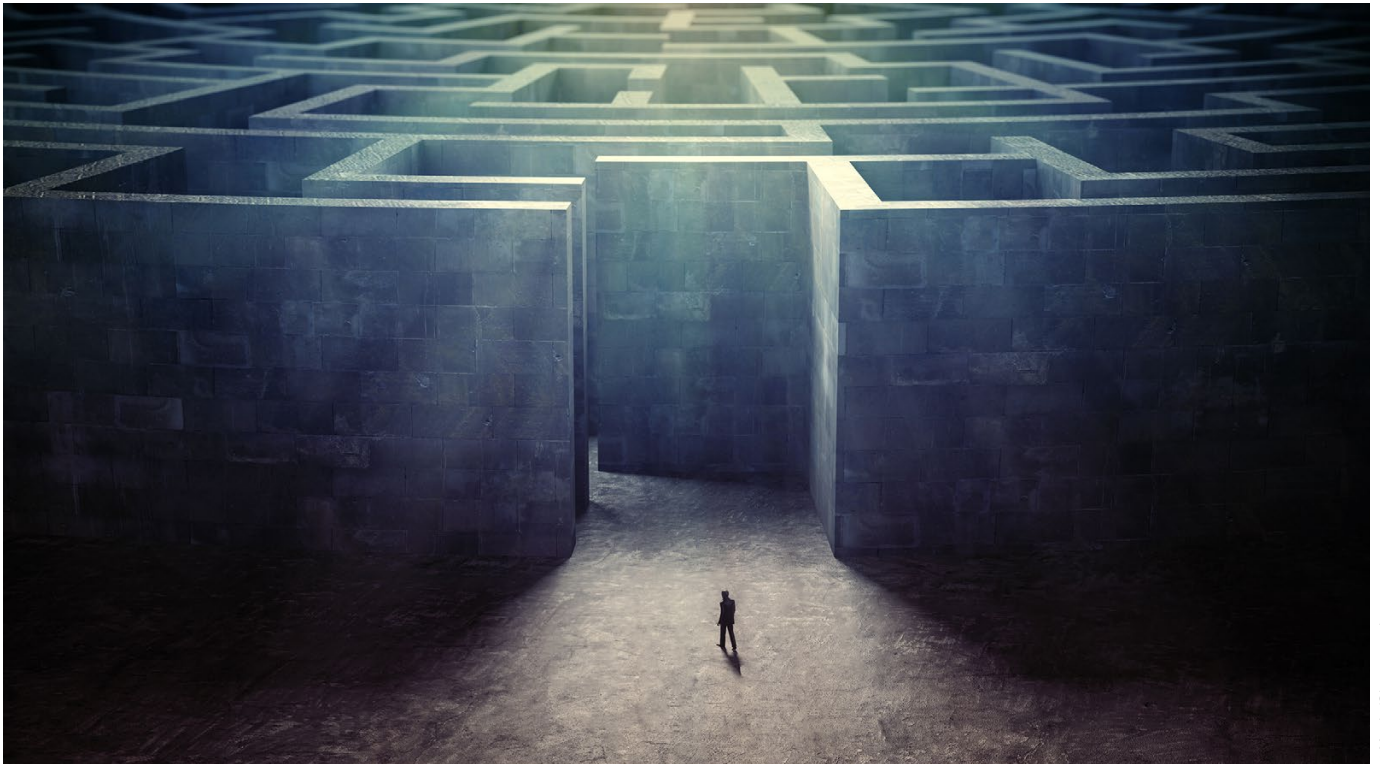


Steve Burton is a DevOps Evangelist at Harness. Steve has held VP Marketing positions at Moogsoft and Glassdoor, and also ran Product Marketing at AppDynamics where he helped disrupt and transform the application performance management (APM) market. Stephen has also held senior product management and pre-sales positions at Symantec and VERITAS software. Stephen's career started at Sapient where he was a Java developer working on large scale enterprise J2EE implementations. During his educational years he also had several internships at Kewill Systems and Oracle Corporation. Steve holds a Bachelor's degree in Computer Science from Lancaster University. He enjoys playing golf in his spare time.

@BurtonSays

References

- [1] <https://plugins.jenkins.io/>
- [2] <https://www.urbandictionary.com/define.php?term=Wonga>
- [3] <https://harness.io/2018/01/production-deployments-actually-succeed-fail/>
- [4] <https://harness.io/harness-continuous-delivery/secret-sauce/continuous-verification/>
- [5] <https://harness.io/>



© Mopie/Shutterstock.com

Security hygiene

Scan your container in build time

As containers have grown in popularity, so too has the understanding that we need to find solutions to keeping the code within them secure from vulnerabilities. In this article, Shiri Ivtsan gives some suggestions on how you can achieve that.

by Shiri Ivtsan

Developers are increasingly turning to containers to make building, testing, and running applications across a variety of environments faster and more efficient. As containers have grown in popularity, so too has the understanding that we need to find solutions to keeping the code within them secure from vulnerabilities. [1]

According to Forrester's latest survey [2] on the state of container usage, security is cited as the biggest barrier to container adoption by organizations. However, if developers hope to keep pace with the industry, containers are a must have, making finding solutions for keeping them secure a top priority.

One of the most significant challenges that development teams face when it comes to security is overcoming the lack of visibility of the code within.

Lacking the full picture – missing dependencies

Developers love working with containers because they are lightweight units known for their efficiency and speed of movement. They answer to modern day agile work environments and the ever-shortening times to market. The downside of their compact size, often measuring only tens of megabytes, is the loss of some of the most important bits that tell us the story behind our applications.

To achieve their lightness, containers mandate a strict cut-off point for all application dependencies. When dependencies

The answer to open source security concerns is early detection. If you've gotten to production and still haven't checked your container for vulnerabilities - you've gone too far.

are cut off from the components that go into the container, they cannot be retracted and the ability to track them to their source component is lost forever. Think about it like a plant being uprooted. Good luck guessing which flower came from which set of roots.

These dependencies are important for security concerns, particularly as they pertain to identifying components with known vulnerabilities which are the primary threat to open source usage.

The issue with open source dependencies

Open source components, built on a chain of dependencies and linkages with open source libraries, are interconnected to one another in “invisible” ways only accessible with the help of Software Composition Analysis tools (SCA) that automatically and continuously track these connections.

Developers rely on dependency management tools like Maven (Java), Bower (JavaScript), or Bundler (Ruby) that pull in third-party dependencies automatically, further increasing the invisible net of open source dependencies that are likely to go unnoticed.

Unlike in non-containerized environments where all dependencies, direct and transitive, remain attached to one another by default, in containerized environments transient dependencies are chopped off to conserve space.

When vulnerabilities are detected in the dependencies that were cut off, there is no possibility to trace them back to the component – now containerized – which they affect. The result is a contaminated container, with a known vulnerability in it, with no way to locate the vulnerability in the containerized environment. Without the ability to identify the libraries in those dependencies, known vulnerabilities which would otherwise be flagged may make their way into the containerized application.

If you think a vulnerable container is headache enough, think again

As opposed to VMs that operate as isolated units fully equipped with their own operating systems, containers are

still essentially applications running on the same OS and therefore are affected by each other.

This connectivity, as Forrester's 2017 report cites, raises the stakes for users that have multiple containers running as a vulnerable container with as little as a single exploited kernel may affect all containers related to it. Security teams should be cognizant of how the lack of visibility into one container can have a domino effect on the security of all other containers in its orbit.

Vulnerability management in the build stage

The answer to open source security concerns is early detection. Checking for open source vulnerabilities in all stages of the SDLC and picking up on them during the build stage allows for timely remediation and eliminates costly post-deployment tear and replace ops.

This approach is doubly true for container usage particularly because of the structure of containers and the security limitations that inform containerized spaces. In this environment, it is about taking inventory of the open source components in your container in build time, pre-packaging, pre-bundling, and of course pre-cutting off of dependencies.

If you've gotten to production and still haven't checked your container for vulnerabilities – you've gone too far.



Shiri Ivtsan is a Product Manager at WhiteSource. Shiri is an Experienced Cloud Solutions Architect and Product Manager who holds B.Sc. in Industrial Engineering and Management. Prior to joining WhiteSource, Shiri worked for various companies where she held roles in R&D, as Solutions Architect, R&D Team Leader & Product Manager.

References

- [1] <https://www.whitesourcesoftware.com/open-source-vulnerability-management-report/>
- [2] http://i.dell.com/sites/doccontent/business/solutions/whitepapers/en/Documents/Containers_Real_Adoption_2017_Dell EMC_Forrester_Paper.pdf



© Esteban De Armas/Shutterstock.com

Containers are just another tool of the cloud-native tool belt

Does serverless spell the death of containers?

Technology moves at the speed of light; as new innovations become popular, others fall out of favor. Has the rise of serverless technology sounded the death knell for containers? Darren Royle explains why containers may not only survive this existential challenge but thrive.

by Darren Royle

To misquote Mark Twain, the rumors of the death of containers have been greatly exaggerated. Serverless technologies may well usurp other technological functions, but I would argue that this will only happen on a case-by-case basis. So, rather than the horse and cart being replaced by the automobile, a more accurate analogy would be the advent of air travel complementing the shipping industry.

It is also clear that three big hosting platforms – AWS, Google Cloud and Microsoft Azure – are not attempting to replace containers with serverless technologies. Instead, they

are augmenting them with other services that containers can make use of.

I would encourage people to consider containers as another tool of the cloud-native tool belt – perhaps even as the final piece of the puzzle – as opposed to a function destined for the figurative computing scrapheap. Below are some examples of the so called serverless future. They are themselves incredibly useful, but not indicative of a future without containers.

The serverless one-liner

Serverless essentially falls into a shared idea of host-provided services. These services are now becoming more granular,

connectable and abstract and, as such, can be used to provide a whole products architecture without the need of a server. Hence the term, ‘serverless’.

There are considerable functions to mention here – and there are a lot more out there – but these three are particularly fundamental in the building of any web based application.

Where do we keep all these cat gifs?

File hosting, a contained function which has been replaced by object storage, is a common example of the move to ‘serverless’. If you need to send static assets like images, then having your own server that serves these up to your users cannot even begin to compete in terms of efficiency and cost with that of object storage. The cost of serving a single static asset on Amazon’s S3 is infinitesimally small – half a cent per 1000 requests – not to mention the sheer scale that Amazon are able to offer to even the most pedestrian user.

When you add to that the benefit of costing based on number of assets served (rather than the uptime of a static file host), you can see why it would be ineffective to attempt to replicate this on a standard server setup. It’s cheap, reliable, requires zero maintenance, and is easy to set up.

And the ‘hilarious’ comments that go with them?

Another example is database. Although this is not a completely solved problem with every hosting supplier, it is getting there rapidly.

We have already seen things like Aurora from Amazon and Cloud SQL and Datastore from Google. These are services that replace your own managed database cluster (whether SQL or not) but still provide all of the same functionality you would see with a native Postgres or Mongo database. The joy being that you don’t need to deal with scale, replication, backup, or patching. It’s all taken care of for you by your host provider, all you need to do is plugin your credit card and tap into it.

As with object storage, many of these services provide a pricing plan that is tailored towards use and capacity, meaning costs scale alongside your businesses success.

Now how do I turn them into a meme?

Finally, we have ‘cloud functions’, which some would point to as the serverless technology to sound the death knell for containers. Services like ‘Lambda’ from Amazon or Azure ‘Functions’ are still seen in their infancy, but provide the concept of “processing on demand”.

This can be a powerful addition to the toolkit of developers and system designers, effectively allowing custom code to be triggered on a given hook. Take the example of a photo processor. Here, a photo will come in and your processor will scan the photo and highlight any faces it can find. In a non-serverless environment, you would have a server that accepted a HTTP request, most likely a POST request with the photo attached to the body of the request.

The server would then run your code that would find the faces, save the results to a database, and store the file somewhere (probably object storage). This can easily be replicated as a cloud function, which can be specifically triggered on

events like HTTP requests. Again, these are managed and scaled for you by the hosting provider, and have the benefit of costs based on usage (many platforms allow 10k functions to be run per month for free).

For startups, this can provide bulletproof architecture very simply and takes no additional time to produce the code. The cloud function APIs that I have used mimic things like the ‘Express’ framework in JavaScript, providing the same request and response API your developers are familiar with. I have also used it to mimic Go’s inbuilt HTTP framework, the Context objects of which developers also recognize. In many cases, migrating is a job of copy and pasting code from your existing HTTP server into the file structure the cloud provider prefers and writing a manifest that specifies how each function is triggered.

The less in serverless

This holy trinity of ‘storage’, ‘state’ (database) and ‘processor’ (cloud functions) can be the only things many web services require. This means that even relatively complex services can have a fully scalable, reliable, and incredibly low maintenance architecture that is costed dependent on success rather than projected capacity requirements.

Contain yourself

So where does this leave containers – are they now surplus to requirements?

While it is true that there are many applications that can use the trinity outlined above and never need to touch containers, this is not the case with all applications. Platforms like ours at Diffblue have processing requirements that can’t be fully defined as ‘cloud functions’. Our main product takes a supplied code base and will ‘automagically’ (sic) generate unit tests for it. The majority of our platform’s requirements, there are like-for-like components that we can and do utilize from our cloud provider.

There are also processes that we need to do that just don’t fit. For example, we need to compile Java test cases and run them. This isn’t possible within ‘cloud functions’ and this is where containers excel.

Containers give us the ability to completely specify and maintain the environment that our code runs in. It’s not simply enough to run the code, we need to know on the host OS that we have access to a specific version of Java, or Maven, or even Git, not to mention running custom compiled C++ applications. Cloud functions just can’t offer the same power right now.

I have seen the big three hosting providers quickly integrate Kubernetes, the new lingua franca for how containers should run and interact with one another. Containers are another aspect that augments the rest of the ‘serverless’ service rather than replaces them.



Darren Royle is the Software Development Team Lead at Diffblue, the Oxford based ‘AI for Code’ startup. Prior to joining Diffblue, Darren served as CTO of Oxford Learning Solutions Ltd.



Some (not so) subtle signs you need a new job

10 professional red flags to watch out for

Complaining about your job is common, but it can be hard sometimes to judge what you should leave a job over. Kostis Kapelonis explains ten of the most common red flags that developers need to watch for and when they should start sending out CVs ASAP.

by Kostis Kapelonis

Passing the interview process and landing a new job is always an overwhelming experience. This is especially true if you are also a recent graduate and this is your first job. You are filled with equal amounts of excitement and anxiety for what lies ahead. Is this company finally the dream job?

If you are a seasoned veteran and have already changed companies a few times, you should hopefully also know what to address in the interview process in order to avoid “bad” companies. Information Technology is slowly becoming an integral part for all companies, not just software ones. So, it is important to understand that the level of software maturity and how a company treats the software development process differs a lot between each organization.

Unfortunately, this also means that several people are working for companies with less than ideal conditions without even realizing that this is not how things are supposed to be. It is very easy to stay for several years in a company – becoming another cog – if you are not aware that your daily routine would be much better in another organization.

For this purpose, in this article, we will see a list of 10 red flags that should tell you whether your current company treats software development in a serious manner or if you should be preparing your CV for your next position.

Red flag 1: You get hired and there is no workstation/laptop/desk for you

Now don’t get me wrong. If you arrive in the morning and they tell you that your desk will be ready in the afternoon,

The level of software maturity differs a lot between each organization.

everything is fine. If you arrive on Monday and the company has a full training program for the whole week until your workstation arrives on Friday, this is also fine.

But arriving in your department and realizing that you have no equipment to work for more than a day is a red flag. First of all, it shows that the company is unorganized in general. Ordering a new workstation is not rocket science. In a big company, the amount of spare equipment should at least prevent such cases by providing a temporary solution. If the company cannot meet the deadline of delivery for a single workstation, can it meet the deadline for a complex software project?

Secondly, it shows neglect from the people involved in your hiring process. The HR manager, your department manager, and even your new team leader should all be concerned with what you should be doing in your first month. If your inability to work does not concern them, it means that your position is not a critical one and was only created for some unrelated reason (e.g. budget allocation).

In summary, waiting for more than 1-2 days for your workstation to arrive should only happen with an important excuse. Fun fact: I know a person that waited for two months until getting a workstation. In the meantime he was just reading books for the technologies used in the project. He no longer works for that company.

Red flag 2: Developers are isolated from everybody else (even their managers)

Remember that the hiring interview is a two-way evaluation process. The company evaluates you and you evaluate them back. During the interview, always ask to see the working space for the developers. This will tell you a lot about the company. It's a really big red flag if a company refuses to show you where developers work until you get hired. If this happens, then just say thank you and walk out of the door.

When you do get to see the other developers, notice the mood on their faces. Are they sad? Exhausted? Demotivated? These are signs of your future if you work there. However, an important thing to also notice is the location of managers according to their respective team. Ideally, each manager should be in the same space as the teams they are managing.

Working in a team means that everybody is communicating with one another. Effective communication is one of the pillars of software development. Communication between developers themselves but also between developers and managers is something that should happen easily and spontaneously. If the managers are in a completely different building than their developers, it means that the company not only hinders communication between the teams, but also treats developers as sweatshop workers that should stay in their rooms (usually in the basement) producing software like working on a factory floor.

Multidisciplinary teams are a huge plus for all companies that understand that software is a collaborative process. Your team should be a mix of developers, test engineers, designers, and managers. All of them should be collocated in the same space. The only exception to this rule is if the company works in a remote or distributed manner. But, for the run-of-the-mill typical corporate organization, looking at the allocation of people between buildings is an important factor to understand how easy communication is between members.

Red flag 3: Daily stand-up meetings take more than 15 minutes

Without getting into details about the Agile process, it is at least important to understand the purpose of the daily stand-up meeting. The objectives of this meeting are well defined. Each person on the team has three questions they should answer:

1. What work was finished the previous day for the sprint?
2. What work will be done today for the sprint?
3. Is there a blocker issue that prevents work?

That's it! These are the only topics that should be mentioned. The stand-up meeting is not a discussion, it is not a stakeholder status update, it is not an analysis meeting, and it is certainly not a brainstorming session. Yet, a lot of companies misuse it and let stand-up meetings drag on for one or even two hours instead of the original 15 minute deadline.

This is a red flag for multiple reasons. Firstly, it wastes everybody's time. Discussing topics that are only relevant to some members of the team is demotivating for the rest of the people. It is ok if extra discussions happen after the stand-up, but only if they contain the subset of the team that is actually involved or needed.

Secondly, it shows that the company does not understand what Agile is. If the company misinterprets a 15 minute stand-up as a full hour meeting, it will probably misinterpret other parts of the software lifecycle as well such as time to deployment, code quality, commit feedback loop, etc.

Lastly, it shows a lack of effective management in general. If the stand-up meeting has become a status update for stakeholders, it means that the existing "official" process for actually getting a status update is not enough. Therefore, people are misusing the stand-up as a substitute.

Red flag 4: Your new team is indifferent towards you

Typically, getting a new person in a team should be a positive experience. Ideally, the team needed a new member, a position was announced, and the interview process finished with your arrival on the scene! Maybe you have a skill that the team needed. Maybe you will be trained on something the team needed. Maybe a new project started and the team size needs to be adjusted.

Your new team should welcome you and make it easy for you to blend in. A mentor should be assigned to you that will be responsible for your onboarding process. Your team is also your primary source of information in the starting months.

Having multiple managers passing you tasks shows that the company treats software developers as factory workers who can quickly move from one machine to another just by shouting over.

Now, it goes without saying that getting a negative reaction from your team is a huge red flag. If your new team is actively aggressive towards you then something is really wrong with the company. However, a big negative sign is also indifference towards you. If the team is apathetic towards you, there are two possible reasons why.

First, the company may have a bad working environment and thus a huge personnel turnover. People are coming and going all the time in your department. Maybe new developers are getting hired, see the mess in the code or the software process, and instantly search for another company. In that case, your team knows that you will leave once you realize the situation. So, they don't feel like they should invest in you. Alternatively, they are the ones that are already applying to leave, so they don't feel they should make any connection with you if they are going to change companies soon.

The second reason might be that simply your team does not care. After so many long hours, missed deadlines, frantic firelights, and time-consuming meetings they simply do not have the energy to welcome you and connect with you.

Both reasons are equally bad. Your team's reaction towards you says a lot of things about the time you will stay in the company.

Red flag 5: Team bonding comes from working long hours

You join your new team and, after some time, you realize that people know intimate details about one another. Everybody knows everything about every other team member and topics that are discussed within the team are the same ones that you would discuss with very close friends.

It is important to understand quickly where this attitude comes from. Of course there are some very rare cases where a team is composed of like-minded people who enjoy working together and are also meeting outside of work or in company team bonding events. In most cases, however, a good colleague is not always a good friend. While it is perfectly normal to make friends from some work colleagues, it is highly unlikely that you would like to discuss your personal matters with your whole team.

Unfortunately a more common occurrence is having teams that are working really long hours to meet deadlines. Working 12-14 hours every day and even weekends with another person will force you to bond with that person since the job starts to become your full life. People are especially more likely to bond with one another under periods of heavy pressure and stress, as there is the common feeling of having to endure the same problems together.

This is a red flag because, while on the surface you see a well bonded team, the big problem is the long hours the company imposes on its people. Ironically this bonding is also one

of the main reasons that keep people in companies like this, as they "enjoy" their team and think it is the "best team ever".

Don't fall into this trap. Creating bonds out of long hours is a problem and you should not let your "good team" blind you.

Red flag 6: Being given tasks by many managers

Task management is one of the most basic problems a company must solve. There are many software suites for this issue, so it should be easy to understand how tasks are created, who assigns them, and who owns them. One of the classic questions that YOU should ask in the interview is who assigns tasks to your team.

In a well-organized company you get tasks from a single person – your manager. Each task has also an expected time frame, importance, and difficulty. It is okay if another task needs priority before the one you are currently assigned to. You should suspend it and resume working on it later.

A big problem is the unclear ownership of tasks. A typical bad day is looks something like this:

- In the morning, you get a task from your manager and start working on it.
- At noon, another team is having issues with a deadline. Another manager asks for your help. You switch tasks.
- Something breaks in production and a third manager asks you to fix the issue right away. You switch tasks.

Then, the next day, the first two managers are sad to realize that you didn't work on what they thought you should work on.

Problems like this are a big red flag for the organization of the company. It is okay if the tasks you work on come from multiple people, but they should also be aware of one another. Ideally, only a single person should be your manager. Having multiple managers passing you tasks shows that the company treats software developers as factory workers who can quickly move from one machine to another just by shouting over.

Red flag 7: Being asked all the time about your current task

Going back to the subject of task management, I already mentioned that there are a lot of software suites designed exclusively for this reason. Any internal or external project stakeholder should be able to get at a glance the current status of a project. The task/issue system should be the single source of truth for the status of the project. Reports and status updates should be easily exportable from the system itself without the need for manual intervention.

Of course, this assumes that the company has a good issue system in place and people know how to use it. Unfortunately

Getting a counter-offer shows that the company does not pay attention to its people.

ly, this is not always the case. A very common problem for developers is getting continuous interruptions in regards to what they are working now and what they are going to work on next. This is a red flag for several reasons.

Getting interruptions all the time is a bad way to work as a developer. Getting into the zone requires uninterrupted periods of time that allow you to think and write code. It is very difficult to get work done if most of your day is spent on micromanagement discussions.

The more important problem is that this micromanagement shows an unorganized company. Either the task management system is ineffective or the company does not know how to use it. Lack of knowledge on task management almost always maps to lack of project management in general. Missed deadlines, wrong estimates and unrealistic expectations go hand-in-hand with micromanagement.

An even bigger red flag is being required to attend special meetings that were created specifically for “getting a status” update on the project. Wasting people’s time with something that is already offered by the system shows a company that doesn’t understand how software gets developed.

Red flag 8: Not getting access to production environments or new tools

A professional in a field is somebody who has access to the best tools of the trade and also knows how to use them. The same is true for developers. New tools and technologies come out all the time. In most cases, they solve problems that needed manual work before their appearance.

Your company should have an official process for getting access to new tools, updating existing tools, and generally keeping up with the developments in the specific business area. Developers should have the freedom to evaluate new technologies and create prototypes for interesting tools.

Unfortunately, a lot of companies still think with the “if it works, don’t fix it” mentality. Legacy projects are announced as frozen and tool upgrading becomes a lengthy process with various layers of manual approvals. If you realize that you spend more time every day battling with the company process than actually getting work done, the battle is already lost.

In a similar topic, developers should also get access to production systems and all the systems they need access to. It is okay to have extra security checks and failsafe facilities in place, but the company should trust its developers that they can get the job done without having to jump through extra hoops.

Red flag 9: Limited internet access and no admin account on workstation

Another way companies do not trust developers is when they limit internet access and workstation privileges. Working

with the best tools possible is a prerequisite for any developer. Any developer should be able to choose their favorite terminals, editors, IDEs and so on. A company that locks down workstations essentially does not trust their developers to have the knowledge they are supposed to have.

Another red flag is limited internet access. Disabling all sites for social media might seem like a good idea initially, but it also cuts developers from learning about new tools and technologies. Content filtering solutions are never perfect and they can often be problematic in what content they allow through. Treating developers with mistrust and assuming that they will want to slack off all day is a red flag as it shows how a company treats people in general and not just in their browsing habits.

Fun fact: There are several companies that instead of cutting access to specific sites do the exact opposite. All sites are banned by default and you need to request access for a specific site that you need. Sad, but true.

Red flag 10: Getting a counter-offer when you say you are leaving

Maybe you learned everything you could learn. Maybe you got bored. Maybe you had long hours. Maybe you found several of the other red flags mentioned here to be true. At some point, it is perfectly normal to leave your current company.

In some cases, you might get a counter-offer. Your company is giving you a raise to stay. There are many articles online that try to propose a solution and whether you should accept the counter-offer or not. I am actually amazed at why this topic is so controversial. The answer is very simple. You should reject the counter-offer and move on. Getting a counter offer is a huge red flag in the first place.

First of all, it means that the company knows it is underpaying you. If they are ready to offer you a raise for the same role, why didn’t they do it earlier?

Secondly, it doesn’t solve the problem of your original decision to leave. The reasons that forced you to leave will still be there if you accept the counter-offer. If the original reason was about money, then trying to get a raise by trying to leave is the worst way possible to achieve it.

Getting a counter-offer shows that the company does not pay attention to its most important capital, its people. This is also true if you learn that one of your colleagues was presented with a counter-offer, it doesn’t have to be you to understand the mentality of the company.

In conclusion

There you have it! We hope that this article gave you some ideas on what to ask in your next interview, how to understand if your company values you, and how to talk to friends that are trapped in horrible companies because of “the good team”.



Kostis Kapelonis is a developer advocate at Codefresh, a continuous delivery platform built for Kubernetes and containers. Formerly a Software Engineer, Kostis has years of experience containerizing applications, building CI/CD pipelines, and developing Java applications. He lives in Greece and loves roller skating.

These are the error codes that haunt us

Beginner's guide: Java programming nightmares

New developers make beginner's mistakes. How can we support our newbie brethren so they don't make a mess of our code? Georgi Minkov explores some of the more common Java mistakes and their solutions with a tour of the horrors of his own early code.

by Georgi Minkov

In our everyday job, we tackle an enormous amount of problems and challenges, from “How do I secure data?”, “What kind of type is this?” to “What are those concepts supposed to be?” This article is not pointing to a specific programming issue like how an encryption works or avoiding SQL injections. Instead of focusing on one topic or one nightmare, I will go back to the basics and try to find the roots of our horrors.

You're about to find out more about some of the programming horrors I faced when I was just starting to code. The goal of the article would be to help fellow developers improve their working process and find solutions for some of their ongoing problems. The following examples would be for Java; however, feel free to apply them to the language which concerns you most.

Returning null as base value

From my experience working in a bespoke software development company [1], there is a fierce discussion on how base scenarios of our program need to be handled. In this basic scenario, we define what value a function needs to return when there are no suitable results. In the next snippet, we have to return workers which name contains string “sup”, but what happens when no names satisfy this condition? One approach is to return null as a result, but that hides some risks. One of the biggest would be that the person working on the assignment after you on a different task may neglect that fact and try to use your result in a different way, which will cause *NullPointerException* (Listing 1).

To prevent that you can return empty List or whatever your collection is. Additionally, your coworkers would be quite happy because they would be able to use this List without triggering the exception.

Of course in some companies returning null as base value is standard. If this is true in your case, head down to the next part.

Checking for nulls

Here's the situation – you call a service or some other part of your project and expect it to return a result of a specific type. Save this result in a local variable and then use it. Using this variable before checking if there is really a result in it can cause a problem like *NullPointerException* or a logical malfunction. To prevent that kind of problem, it's advisable to check whether the result is null and/or empty (Listing 2).

Not documenting your code

Most of us hate documentation or dealing with it in normal daily routine, but in programming, it can save lives. I'm not exaggerating; I'm sure you've been in a situation where you needed to use a certain method or class but had no idea what

Listing 1

```
List<Person> workers = manufactory.getWorkersByCondition(); // returning null
for (Person worker : workers) { // NullPointerException occurs
    // some logic
}
```

Listing 2

```
List<Item> shoppingCart = session.getShoppingCart();
if (shoppingCart != null) { // you can add && !shoppingCart.isEmpty()
    // operate over cart
} else {
    // if needed specific operations
}
```

it does and how you could operate with it. In the best case scenario, the code is written so well that it explains itself. However, this is rarely the case.

There are different types of documentation and different tools which could help like Swagger [2] or Doxygen [3] for Spring Boot project documentation. Here are a couple examples of where a line of comments could help both you and the developers coding after you:

- When you work on a project for a long time and return to an old feature, a couple lines of comments can direct you in why and how certain things are configured.
- When a third person uses your API or some method, they can determine whether this functionality is really needed and how it works with your documentation.

There are different best practices on when and how to write comments, but in my personal opinion most important ones are:

- When there is something unusual in your code, like *if* without *else* or complicated logic, you can put a comment why you did things the way you did.
- When you create some specific tool or service, you can use some documentation tools or documentation standards for your language to briefly describe important parts and any complex peculiarities.

Next time when you spend hours of coding ask yourself – “Did I forget something?”

Exception handling

Often, it is enticing to neglect exceptions handling, but this would be a big threat. First, you never know where the ex-

Listing 3

```
try {
    client = clientFactory.getClientBy(username, password, serverURI);
} catch (CustomException ex) {
    logger.error(ex.getMessage(), ex); // this line can save us from
                                     painful debugging
}
```

Listing 4

```
try (Scanner scanner = new Scanner(new File("./description.txt"))) {
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
    // ..more logic
} catch (FileNotFoundException ex) {
    // some safe logic
} catch (CustomException ex) {
    // another safe logic
}
```

ception may occur. Second, you just broke project structure. The best practices for beginner and experienced developers are to handle the exceptions, knowing when to throw and catch them.

Hiding an exception is another way of handling them that can be a real nightmare. From personal experience, you can lose hours of debugging only to find out that something catches an exception but does nothing with it (Listing 3).

Exceptions are thrown on purpose, so you need to take care of them. If needed, you should rethrow them, log information, or make safe logic for handling. Make sure you address the issue causing particular exceptions. If you do not handle it on purpose, write a comment and document it so other developers are up to date with the code.

Trying to do everything by yourself

If you work with Java, you know that there are an enormous number of libraries that can be used by developers. When you are facing a problem or thinking on how to implement specific functionality, it is important to research the available libraries that are applicable to your case. Other developers have spent years on perfecting the libraries. What’s more, they’re free! For example, if you need a logging tool use Log4j [4]; if you need network libraries, Netty [5] could be a great choice. Some of these open source libraries have become standards in their field.

The next time when you need to do some XML parsing or log information, it’s important to ask “Am I the first one who needs this, or has someone else already implemented a solution?”

Freeing up resources

This is especially targeted to newbies in the business. When you open a network connection or connection to a file it’s important to close it after finishing our job. Otherwise, we block resources that can be used for something else.

Here’s an example of a developer’s poor memory: when an exception is thrown during operations over those resources. Yes, implementations like *FileInputStream* (as files were mentioned) have a finalizer that invoke *close()* and frees up resource. However, this will be called on garbage collection cycle (and we cannot predict when this will happen). There is some beautiful piece of craft introduced in Java 7 try-with-resources (Listing 4).

This handles any objects that had implemented *Auto-Closure* Interface and closes the resources that we used without needing to explicitly call *close()*.

Memory management

Next, we will briefly look over memory leaks and expensive allocations. Java’s automatic memory management has been a big relief to developers, who no longer need to free the memory manually. However, it’s also important to understand what happens under the hood of our applications and see how the memory is actually used. As long as our program creates references to an unneeded object, it will not be deleted; that could be labeled as a memory leak.

Do you know what expensive allocations or garbage allocations are? In short, it is when a program is creating con-

tinuously short living objects. This approach is dangerous because it will cause performance slowdown of the application, continuously creating and deleting unneeded objects. In the next snippet, we can see a simple example in Listing 5.

String is immutable here. So, to fix this Java snippet, we will create new instance with every iteration with *StringBuilder* (Listing 6).

Concurrent modification

Sooner or later, you will face *ConcurrentModificationException* and start searching for the killer. This situation occurs when you try to modify a collection while iterating it without tools from iterator object (Listing 7).

This can happen in a one-threaded application, now imagine having a multithreaded one. One thread is iterating and looking over a list with values while a second thread is removing elements from the list. This can be pretty tricky and we need to be careful when working with it. For that, we have things like synchronization locks and implementation for collection specified for concurrency.

It's important to keep in mind that the phase before the actual coding is just as important. If done wrong, you could be left to deal with some real nightmares. I will share some of my experiences and the solutions I have found below.

Not planning your solutions

When first starting my programming job, one of the biggest nightmares I faced was when I started implementing imme-

diately right after receiving the specifications from the client. This led to six weeks of great effort only to realize that I was developing something that was never required by the client.

My advice is to take the time to sit down and fully understand the requirements from a client or your PM/PO. Ask as many questions as possible to clear all uncertainties. Get in line with the business goals of the project. You should think of your solution as part of a bigger picture, so you could find the right questions. Try to only code useful pieces without wasting any time. This should guarantee positive feedback.

Not following coding standards

Another nightmare I got myself into was not following company standards and coding without considering my colleagues. This led to huge misunderstandings and a lot of tension within the project. In order to avoid getting mixed up in a situation like this, I would highly recommend reading your company's best development practices and making sure your code is compliant.

This will help when your code is reviewed by clients, managers, and fellow developers. It might actually help you in getting additional knowledge and improving your programming skills. In addition, it is incredibly important to follow the coding convention for the programming language you will be using.

Not testing your code

Another thing I managed to get myself into was spending an entire sprint fixing bugs from my last release after underestimating testing. No matter how many lines of code you've actually managed to create, you absolutely have to test how they integrate with other parts of your project. This could be easily done by doing a simple research on testing styles and then choosing what fits you best. You could go with unit testing, system testing, integration, or any of the other testing styles.

All in all

Bugs and technological misconceptions are beautiful mysteries that we create. We hold our own destinies in our own hands; whether we haunt the endless circles of development hell or Sherlock Holmes our way out of these technological nightmares depends on how well we can follow our own coding processes.



Georgi Minkov is a Java and Oracle Developer at Dreamix, a custom software development company. He has experience with Java and Oracle technologies. He indulges in experimenting with different technologies, is a teaching assistant at Sofia University, invests in learning AI and hardware trends, and loves photography and knowledge sharing.



@Dreamix_Ltd

References

- [1] <https://dreamix.eu>
- [2] <https://swagger.io>
- [3] <http://www.doxygen.org>
- [4] <https://logging.apache.org/log4j/2.x/>
- [5] <https://netty.io>

Listing 5

```
String answerToEverything = "";
for (int index = 0; index < 100000; ++index) {
    answerToEverything = answerToEverything + "!42!";
}
```

Listing 6

```
StringBuilder answerOfEverything = new StringBuilder();
for (int index = 0; index < 100000; ++index) {
    answerOfEverything.append("!42!");
}
System.out.print("Answer of everything and beyond " +
    answerOfEverything.toString());
```

Listing 7

```
List<String> operationSystems = new ArrayList<>(Arrays.asList("Fedora", "Arch",
    "Debian", "Manjaro"));

for (String system : operationSystems) {
    if (system.contains("ar")) {
        operationSystems.remove(system);
    }
}
```




Five most common DevOps failures

Learning from DevOps nightmares

Nobody likes a failed implementation. However, learning from your mistakes is what keeps a mistake from turning into a true DevOps nightmare. Brian Dawson of CloudBees explains five different ways developers can solve break free from their dreadful errors and return to the DevOps implementation of their dreams.

By Brian Dawson

Applications should not crash. Systems should not go down. In the world of DevOps, failure isn't exactly encouraged. While we have seen every kind of iteration positioning failure as a positive thing (failing forwards, failing better, failing often, you name it), the fact of the matter is that 'failure' isn't particularly beneficial or conducive to achieving our goals.

Yet, DevOps failures are an unfortunate reality for many of us and they are often inescapable. Perhaps, then, it may be useful to redefine what failure is and what failure is not. Failure is being unable to meet your objectives; it is a lack of performance. Failure is not making and then learning from your mistakes, which will allow you to succeed later on.

Speed is as critical to software development as it is recovery from failure, so the rate at which you can leverage the insights gained from making mistakes is critical to your success. With this philosophy as the background, here are some of the most common DevOps failures, and how to learn from them quickly.

1. Falling out of step

DevOps means having to deliver at the speed of business, which can be particularly challenging in today's fast-paced, technological world. As such, many teams run process in parallel during continuous integration and continuous delivery, allowing them to accelerate their automated testing and shorten their feedback cycles.

According to Laura Frank Tacho, Director of Engineering at CloudBees, “While this may be a good idea in theory, it can sometimes cause significant misconfiguration. Allowing the code to deploy at the same time tests are running consequently defeats the purpose of pre-deployment automated testing.”

As such, it is vital to remember that code must pass all appropriate checks and balances before being deployed to the customer. While DevOps principles assert that engineers should be ready to deploy at any given moment, deployment should be managed and deliberate. Continuous delivery pipelines must be constructed in such a manner that parallel testing serves as a gate to production, not an automatic route to deployment. Rather, deployment should only occur at the final stage of the pipeline after all tests, validations, checks, and approvals have passed.

2. Getting locked out

It’s important to have the right layers of protections in place at every stage of the development process. Without validation of configuration, for instance, engineers can lock themselves out and be forced into the tedious process of manually logging in to each machine to reconcile misconfigurations.

“To preempt these types of situations, engineers should validate all changes early – and often – to ensure not only the quality of the application, but the environments and even the CD pipelines itself,” explained Carlos Sanchez, Principal Software Engineer at CloudBees. Applying cohesive credentials and permissions management to DevOps and productions systems will also help engineers focus on problem prevention, rather than detection.

3. Organizational obstacles

Not all problems DevOps engineers face are technical; sometimes it can be a case of their organization’s culture and structure. When multiple teams and departments are concentrated in various silos, it becomes difficult to treat systems and software developments as a whole, particularly pursuing continuous improvement. For instance, when attempting to facilitate cross-departmental collaboration, answers from the opposing teams can be “that’s not our department” or “they won’t speak with us.” Even in more collaborative companies the time required for cross-functional hand-off and context switching is a significant obstacle to achieving velocity.

Organizing teams around product or software functionality and including all stakeholders in the software development process will allow DevOps leaders to improve the synergy amongst stakeholders. It should also prevent the ‘us versus them’ mentality that can sometimes plague traditional software development. Viktor Farcic, Senior Consultant at CloudBees, agreed, pointing out that, “Culture must not be overlooked as a factor in DevOps. While process, practices, tools, and technology are undoubtedly core pillars of the practice, culture is equally important. It is surprisingly one of the more difficult aspects to change.”

4. Unsuitable KPIs

The pace of business is only becoming faster as new technologies improve operational processes and transform business

models for the digital age. Shareholders are applying intense pressure on organizations to keep up with this rapid pace. As a result, organizations are finding themselves having to report multiple key metrics to measure their performance across all areas.

In light of this, Juni Mukherjee, Product Marketer at CloudBees, explained that these metrics aren’t always the most suitable. “To effectively prove performance, DevOps leaders must first establish a set of principles that will help them understand what, and how, meaningful metrics can be obtained; for example, considering lead times as time to production, not to completion.”

5. To scale or not to scale

When working with open source software, it is vital to have expertly trained developers who can ensure that the software is still suitable for their organization’s particular use case. It’s also important to make sure it takes full advantage of the innovations the open source community has to offer.

However, in order to select appropriate open source software, Sacha Labourey, CEO at CloudBees, explained that DevOps teams should first “take a long-term view of their organization’s needs and business priorities, and discuss how the solution will need to scale over the next few years. Doing so will allow organizations to understand the prep work needed to scale usage.”

Teams should also determine if the solution aligns with their existing tool chain. As organizations continue to scale, they will be hard-pressed to find a tool that can do it all. As a result, integrations along the way are inevitable. Ensuring that these integrations are stable is critical for their success. Understanding how the solution will fit and work with existing tools is an important factor in scaling for the future, and, ultimately, the organization’s growth.

In conclusion

To successfully practice DevOps, you must build seamless connections linking people, processes, tools, and goals. However, this entails hard work. There will be mistakes and communications along the way. These issues, as challenging as they may be, are solvable if DevOps processes are treated as an iterative process of learning and improvement. By using DevOps correctly, developers can turn short-term failures into long-term success.



Brian Dawson is currently a DevOps evangelist and practitioner at CloudBees [1], where he helps the open source community and CloudBees customers in the implementation of agile, continuous integration, continuous delivery and DevOps practices. Before CloudBees, Brian spent over 22 years as a software professional in multiple domains including QA, engineering and management. Most recently he led an agile transformation consulting practice helping organizations small and large implement CI, CD and DevOps.

References

- [1] <https://www.cloudbees.com/>

Avoiding your own 3 a.m. disasters

To catch a bug

Being on call is a fact of life for some developers. In this article, Irit Shwarchberg explains that although bugs are a fact of life, there are ways to mitigate those catastrophic failures and save yourself from a midnight bug hunt.

by Irit Shwarchberg

The phone rang. It was 3 a.m. – not entirely unexpected, but still unwelcome. It was my turn on call. No, not at the hospital or at a crisis center, but at the telecom company where I was working. I was in the middle of a weeklong rotation as the go-to person for any bugs detected in the system, day or night. And just my luck, it was during the night part of “day or night” that an error with the data from the source had been detected, causing a complete system fail.

The bug was mine to find and I got to work looking for it. I began by searching our Cognos reporting system and Informatica; first opening the report in Cognos trying to see if there was an incorrect calculation anywhere. I went through all of the data items within the report and everything looked fine – nothing had been touched for the last few months. Then I went to dig inside Informatica. I started to open various maps, looking for a specific one that related to the bug. I began by opening the final map in the data flow, which inserts data into the table that the report relies on.

When I didn’t find the problem there, I went on to check the other maps, having to find and check all of the source tables. Making the task even more difficult was that with so many maps, it was nearly impossible to tell what had already been checked and what still needed to be looked at. Although I combed the system from top to bottom to the best of my ability, the bug was nowhere to be found. Even worse, the bug was affecting a system for employee timesheets and salaries, so my coworkers’ livelihoods were entirely in my hands. Failing to find the bug on my own, I pulled in some of my colleagues to help, but they had no luck either. Neither did the Business Intelligence team.

The problem, at least on the surface, was missing data. But we triple checked all of the fields and couldn’t find any errors. We went through the data flow over and over again trying to find a connection that was made in the wrong way or a definition that was incorrect. After four days, I was nowhere and still had no idea where that missing data was. As the end of the month approached with salaries on the line, we were racing against the clock. We had no option but to enter all of the information manually.

I began opening each row of data to map the information transformation into the system. If the map was small and had

only a source-to-table with a few rows, it was like winning the jackpot. But most maps have complex data flows, and understanding each data transformation is not only very difficult, it can also take days. Yet luckily, through our manual check, I eventually found the culprit. There was a field whose function was to count mistakes in the data – to show how many errors existed. Yet whoever had created the field had only accounted for a single digit. And with more than nine errors in the data, the system had simply failed.

Our crisis had been averted and our salaries were paid in time. Our hunt for the bug left us with a very valuable lesson about data: understanding how our data is mapped cannot be understated. In any database, there is both the data itself and the field that it lives within. But when you look across an organization with multiple databases, ETL tools, and multiple owners of those databases – not to mention multiple employees entering the data – consistency is rarely achieved. In one database, gender may be entered as male/female, and in another as M/F; dates can be entered month first or day first, with a two-digit or four-digit year; and factors such as middle names can throw the entire system for a loop. And, as in the case of our 3 a.m. system failure, the people who build the data flows and processes are rarely the same people to use them. As a result, they’re often unaware of the limitations of the fields they create and the disasters that may cause.

How do you avoid your own 3 a.m. disaster? At the end of the day (and in the middle of the night), it comes down to knowing your data. These system fails are unavoidable; bugs will ultimately happen. But understanding how your data is mapped, what fields exist, what limitations are placed on each of them, and how they’re interconnected across databases will not only give organizations better control, it will also give them more accuracy, allow different teams to work with multiple data sets, and help avoid situations of missing data, such as the one I experienced. For anyone looking to avoid manually searching through each field, automating that process is key.



Currently heading Customer Success at Octopai, **Irit Shwarchberg** is a BI expert with more than ten years of experience leading complex data projects in IT and Telecom, both on the development side and the analysis side.



Sherry List, front-end lead developer at Nordea Bank

How to get (and stay) into the tech industry

Women are underrepresented in the tech sector – myth or reality? In 2017, we launched a diversity series aimed at bringing the most inspirational and powerful women in the tech scene to your attention. Today, we'd like you to meet Sherry List, front-end lead developer at Nordea bank.

A research study by The National Center for Women and Information Technology [1] showed that “gender diversity has specific benefits in technology settings,” which could explain why tech companies have started to invest in initiatives that aim to boost the number of female applicants, recruit them in a more effective way, retain them for longer, and give them the opportunity to advance. But is it enough?

What got you interested in technology?

My parents say I was always the one who disassembled any electronic devices and tried hard to assemble them back, even though that it was often a failure! They also noticed that I was very interested to see what my uncle used to do with the computer. So, they decided to send me to a computer course for kids. I think that was the beginning :)

When I was around 13, I was 100 % sure that I wanted to become a Software Engineer. Therefore I decided to go to a technical high school. After finishing high school, I was lucky to have an older cousin who is an entrepreneur. He offered me a position at his company as a web designer. It was the best thing that happened to me. At his company, I managed to work with really amazing developers and engineers who really shaped my mentality to become a self-learner.

Instead of teaching me from A-Z, they gave me assignments and resources where I could surf and learn how to find my answers. After working there for a couple of years, I started my education at university as a software engineer. I worked on a side project with a friend of mine, which later became the first E-Learning solution for enterprise companies in Iran. I'm still proud of it! I would say compared to


many around me, I knew what I want to be, and I just followed my dream.

I always consider myself to be very lucky to have all these amazing and supportive people around me. My parents and my brother have always supported me in what I wanted to do and totally respect my choices. My husband always encourages me whenever I get cold feet to do something that I consider to be totally outside of my comfort zone or when I feel like I'm not good enough to do it. He keeps up with the crazy schedule I build for myself with organizing meetups, conferences, workshops and also traveling and giving talks at different events ... I really appreciate his patience.

My friends are also so amazing; I receive constant support and encouragements from Chris Norring [2], Kenneth Christiansen [3], Ana Cidre [4] and Lars Knudsen [5]. These people are always there to help me.

Portrait

For the past 15 years, Sherry has worked with a variety of web technologies and is currently focused on Angular. She lives in beautiful Copenhagen, where she works as a front-end lead developer at Nordea bank. Apart from her everyday job, she is a co-organizer of the ngVikings conference, as well as some meetups Meetups groups, such as ngCopenhagen and GDG Copenhagen. She loves animals and supports various non-profit animal protection organizations.

 @sherrylst

“At the moment, tech is mostly dominated by males; there are not many role models for females. Even though there is a lot of focus in bringing women in tech, many don’t think they can become successful in this field.”

As for role models, hopefully, I have many of them. My favorites are Tracy Lee [6], Aysegül Yönet [7], Carmen Popoviciu [8], Shmuela Jacobs [9] and Simona Cotin [10]. These ladies are not only great developers, fantastic speakers and amazing human beings, but are also empowering people around them.

All of us (Women in Tech) have met at least one person in our lives who couldn’t see us shine. I met a few ... I mostly either ignore them but one time I became so angry that all of a sudden, I just quit my job!

A day in Sherry’s life

I work as a front-end lead developer at Nordea bank in the beautiful city of Copenhagen. Next to my daily job, I organize ngCopenhagen [11] and GDG Copenhagen [12] meetups. I’m also one of the organizers of ngVikings conference [13], which is the only travelling Angular conference in the world. Recently, I also started to give talks at conferences mostly with a very good friend of mine, Ana Cidre. Ah, I forgot to mention that I am also Women Techmaker Lead of Copenhagen [14].

So my day starts with checking the news and my emails while drinking my morning coffee. Then I walk to the beach with my dog, Lance. Depending on the weather, we normally spend 30 minutes together there. Then I take public transport to my office and start my work. We normally have a stand-up at 9:30 and a coffee date with my colleagues at 11:30. During the day, apart from working on my tasks, I chat a bit with Ana, we mostly exchange articles or plan for our talks or events. I finish my day at the office around 5:00 pm and take public transportation back home.

I normally have at least one meeting, either related to my community works (organizing meetups or conferences) or a hangout call with Ana about our talks.

I am super proud that I have been able to push myself out of my comfort zone and build communities and organize conferences and also give talks at international conferences. I see it as a big step in my career.

Why aren’t there more women in tech?

As you know, it’s not an easy question to answer. In my opinion, it’s a result of various issues. First of all, at the moment, tech is mostly dominated by males. Therefore, there are not many role models for females. So, even though there is a lot of focus in bringing women in tech, many of them switch jobs, since they don’t think they can become successful in this field.

On the other hand, parents are also encouraging their daughters to choose other career paths, for example, economics, medicine, etc. At school, teachers mostly encourage guys to take more computer courses. As a result of all these, females don’t think that they do belong to the tech industry, so they don’t bother even to try to join this industry.

But I want to be optimistic and say that hopefully, it will all change soon.

For sure we still have a long journey ahead of us and not all of the statistics look promising ... But yet, there is still good news out there. Bigger companies have realized that ‘Diversity and Inclusion’ is not just a nice feature to have, they should adopt it. There are a lot of efforts out there to close all these gaps. Meanwhile, women around the world are fighting for this transformation. Many statistics show that 60 % of global graduates are female, which I consider to be a good sign.

Women in STEM

Most consumer purchasing decisions are made by women! It would be great if they could take part in building those products. Diversity has a direct impact on the products that companies build; if they make it more inclusive for everyone, they will gain more customers.

On the other hand, team cultures at companies will change, we will hear fewer sexist jokes at work, and so fewer women will switch jobs to join other industries which have more women.

Obstacles

Well, I have lived and worked in different countries. I believe women have different challenges and obstacles in each part of the world. But there are some common ones. For example, the stereotype of being a developer. I even remember someone asked one of my fellow female speakers at a speaker dinner: why are you here, which of these guys is your partner? Because of course it’s not expected to be a female and give a technical talk at a conference!

The other common challenge is that most of the time when you are in a meeting (of course as the only female member of the team), people don’t expect you to have any (technical) opinion or they don’t take it in consideration! Of course it’s different depending on the companies and countries. But it has happened to me a lot! As a woman in tech, you should develop a very strong personality :)

Tips and tricks

1. Always believe in yourself and your abilities and never let anyone discourage you. I read this phrase somewhere and I love it: 'Be like a compiler and ignore comments.' Of course, you need to always take constructive comments into consideration. But believe me, ignore the rest.
2. Don't underestimate the power of networking. Join your local meetups and conferences and once you are there, talk to people. There are loads of nice people out there and they can help you in many different ways or even one day you may apply for a job and see that the person who is sitting there is the same person you talked to a few days ago! (It happened to me!)
3. Read and study and try to share what you learned with the others. Because once you share your knowledge, people will ask you questions and if you don't know the answer, you will try to find it. That way, you expand your knowledge.

References

- [1] https://www.ncwit.org/sites/default/files/resources/impactgenderdiversitytechbusinessperformance_print.pdf
- [2] https://twitter.com/chris_noring
- [3] <https://twitter.com/kennethrohde>
- [4] https://twitter.com/AnaCidre_
- [5] <https://twitter.com/denladeside>
- [6] <https://twitter.com/ladyleet>
- [7] <https://twitter.com/AysSomething>
- [8] <https://twitter.com/CarmenPopoviciu>
- [9] <https://twitter.com/ShmuelaJ>
- [10] https://twitter.com/simona_cotin
- [11] https://www.meetup.com/de-DE/ngCopenhagen/?chapter_analytics_code=UA-68620812-1
- [12] <https://www.meetup.com/de-DE/GDG-Copenhagen/>
- [13] <https://ngvikings.org>
- [14] <https://www.womentechmakers.com/directory/shahrzad-aziminia>

Imprint

Publisher
Software & Support Media GmbH

Editorial Office Address
Software & Support Media
Schwedlerstraße 8
60314 Frankfurt, Germany
www.jaxenter.com

Editor in Chief: Sebastian Meyen
Editors: Jane Elizabeth, Eirini Papadopoulou, Gabriela Motroc, Hartmut Schlosser
Authors: Markus Biel, Mike Bursell, Steve Burton, Brian Dawson, Lukas Eder, Oren Eini, Richard Gall, Trisha Gee, Matthew Gillard, Mala Gupta, Josh Long, Shiri Ivtsan, Kostis Kapelonis, Karen Krivaa, Sherry List, Glenn Mariën, Georgi Minkov, Simon Ritter, Darren Royle, Irit Shwarchberg, Tal Weiss, Christophe Thibault

Copy Editors: Jonas Bergmeister, Jasmin Höhl, Frauke Pesch
Creative Director: Jens Mainz

Layout: Dominique Kalbassi

Sales Clerk:
Anika Stock
+49 (0) 69 630089-22
anika.stock@sandsmedia.com

Entire contents copyright © 2018 Software & Support Media GmbH. All rights reserved. No part of this publication may be reproduced, redistributed, posted online, or reused by any means in any form, including print, electronic, photocopy, internal network, Web or any other method, without prior written permission of Software & Support Media GmbH.

The views expressed are solely those of the authors and do not reflect the views or position of their firm, any of their clients, or Publisher. Regarding the information, Publisher disclaims all warranties as to the accuracy, completeness, or adequacy of any information, and is not responsible for any errors, omissions, inadequacies, misuse, or the consequences of using any information provided by Publisher. Rights of disposal of rewarded articles belong to Publisher. All mentioned trademarks and service marks are copyrighted by their respective owners.