Special Edition 2 / 2025
#JAVAPRO

CORE JAVA • FRAMEWORKS & APIs • ARCHITECTURE • CLOUD • AI

JAVAPRO

THE FREE MAGAZINE FOR THE JAVA COMMUNITY

30 YEARS OF JAVA

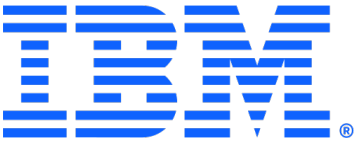
SPECIAL EDITION

PART 2

With the kind support of our partners.

JAVAPRO PARTNER NETWORK

Platinum Sponsors



Gold Sponsors


Silver Sponsors




Bronze Sponsors





JAVAPRO

Publisher:
JAVAPRO
Im Gewerbepark 29
92637 Erbendorf
Germany

E-Mail: info@javapro.io
Website: http://www.javapro.io

Editor in Chief:
Markus Kett (V.i.S.d.P.)

Editorial:
info@javapro.io

Design, Layout & Print:
Impuls Mediengruppe GmbH
Im Gewerbepark 29
92681 Erbendorf
Germany

30 Years of Java – Part 2 of an Ongoing Success Story

Thirty years of Java is more than just an anniversary. It's the story of a technology that was not only built to last, but continuously shaped the way we develop software. Java has empowered generations of developers, supported countless applications, and transformed entire industries. This second part of our special anniversary issue revisits the enduring strength, flexibility, and innovation of one of the most influential ecosystems in modern computing.

Evolution, Not Revolution

Java has rarely been disruptive in the traditional sense. Instead of abrupt breaks, it has evolved with purpose. From the original "write once, run anywhere" vision to generics, lambdas, modules, and most recently virtual threads and structured concurrency - Java has consistently advanced, without losing its foundation.
Its greatest strength lies in this consistency. Java integrates new paradigms without discarding existing ones. It protects investments while expanding possibilities. This unique balance between innovation and stability is what has kept Java at the forefront for three decades - and it's what will keep it there going forward.

A Rich and Expanding Ecosystem

Java today is far more than a language. It is a global ecosystem, enriched by a vast range of libraries, frameworks, tools, runtimes, IDEs, and standards. It thrives equally in legacy enterprise systems and cutting-edge cloud-native architectures. Whether on servers, in containers, in stream processing pipelines, or in AI-assisted developer tools - Java remains relevant and adaptable.

Especially exciting is Java's convergence with artificial intelligence. Development workflows are changing. Smart assistants, automated

tests, semantic code analysis, and AI-powered migration tooling are no longer visions of the future - they are part of everyday development for a growing number of teams.

Facing Challenges, Shaping the Future

Of course, Java is not without challenges: increasing complexity, rising security demands, new compliance rules, ecosystem fragmentation, and the emergence of competing languages. Yet, Java has always excelled when underestimated. It responds not with rigidity, but with engagement. Open-source initiatives such as OpenJDK, Jakarta EE, GraalVM, and others continue to push the platform forward - openly, transparently, and sustainably.

The true source of Java's innovation is not only in the technology itself, but in its community - global, diverse, and collaborative. This is what has carried Java for the past 30 years, and what will continue to shape its future.

The Next Chapter Has Already Begun

This second anniversary issue is not a retrospective - it's a look ahead. It highlights that Java isn't at the end of its story, but entering a bold new phase. The problems we solve are more complex, the tools more intelligent, the responsibilities greater. But the core remains: clean, robust, maintainable software - created by people, supported by community, written in a language that continues to prove itself.
With that, we welcome you to the second part of our birthday celebration. Welcome to the next chapter in the Java story.

Markus Kett
Editor in Chief
JAVAPRO

https://linkedin.com/in/markuskett
https://twitter.com/MarkusKett

#JAVAPRO #JCON

Cinema, Code, Community: JCON EUROPE 2025 Raises the Bar for Java Events

From hands-on workshops to live coding on the big screen to the new mentoring revolution – JCON EUROPE 2025 in Cologne was a celebration for Java enthusiasts from around the globe.

A Global Gathering of the Java Community

From May 12–15, 2025, Cologne became the epicentre of the international Java community. For the tenth edition of JCON EUROPE, developers from more than 60 countries across five continents gathered at the Cinedom to celebrate 30 years of Java. The atmosphere? Euphoric, collaborative – with plenty of good humour.

"30 years of Java. 10 years of JCON. The spirit of celebration was everywhere," one attendee wrote on social media. Another summed it up: "JCON feels more like a reunion of old friends."

Honoring JCON: 10 Years of Innovation Recognized by Oracle

A standout moment of the conference: Oracle honored the JCON team for ten years of continuous commitment to the Java community. The award was personally presented by Sharat Chander (Java Community Lead at Oracle) to Markus Kett (Founder of JCON) and Richard Fichtner (Co-

Organizer of JCON). This recognition highlights JCON's central role in the global Java community – not just as a conference, but as a platform for innovation, exchange, and learning.

Workshops: An In-Depth Introduction to Key Topics

On Monday, the conference kicked off with a full day of hands-on workshops. In small groups, participants engaged in intensive coding, testing, and discussions. Topics ranged from Testing (Christian Stein, Marc Philipp), Java Performance and Caching (Florian Habermann, Christian Kuemmel), AI-powered application development (Marta Tolosa, Sydney Nurse, Sven Ruppert), to efficient deployment and startup optimization (Mark Stoodley). Parallel to this, the full-day Java Luminaries Summit brought together leading Java experts for in-depth exchange.

The workshops provided an excellent opportunity to explore cutting-edge technologies and work directly with thought leaders in the field. Attendees particularly appreciated the mix of theory, practical exercises, and room for individual questions.

Many used the workshop day as the perfect launch into the conference week, reporting "valuable aha moments" and "immediately actionable learnings" for their teams.

AI & GenAI: Java's Next Frontier

AI has firmly arrived in the Java community – and the potential is enormous. Especially in the enterprise space, where Java is a leading force, modern AI technologies now meet vast, mature datasets – a treasure trove waiting to be unlocked.

It's no surprise that Generative AI was one of the hottest topics at JCON 2025. Numerous high-profile talks and sessions showcased how GenAI can seamlessly integrate into Java ecosystems: from LangChain4J to vector embedding models to AI-driven application development and business process optimization.
The message was clear: Java and GenAI are a powerful combination – unlocking enormous potential for the next generation of enterprise applications.

Conference with Cinema Glamor

Instead of sterile conference rooms, live coding sessions lit up the big cinema screens. The idea of staging tech talks with Slido Q&A in a cinema setting is now a well-established JCON tradition – and it continues to captivate.

Each day of JCON EUROPE 2025 featured an inspiring keynote that reflected the diversity and future focus of the Java world.

On Tuesday, Markus Kett opened the conference with "Java's Ignored Potential," demonstrating how Java-native in-memory data processing can drastically improve the performance of data-intensive applications – a topic that resonated widely with attendees. On Wednesday, Sharat Chander, the face of the global Java community, took the stage with "Happy Birthday, Java!" – an emotional and informative look back at three decades of Java's legacy, from powering the Mars Rover to the Olympic Games. The final keynote on Thursday was presented by Mark Stoodley (Chief Architect Java at IBM) and Markus Kett with "Rethinking Microservice Persistence" – a revolutionary new approach to database architectures: microservice-based data systems moving away from database monoliths, offering up to 80% savings in compute, energy, CO2 emissions, and cloud costs.

Beyond the keynotes, the program delivered impressive technical depth, dynamic live coding, and a highly engaged community. Cay Horstmann offered deep insights into Virtual Threads and Project Valhalla, Alina Yurenko guided attendees through the world of GraalVM. At the same time, François Martin, Lize Raes, and Andres Almiray brought topics such as Testing, Security, and Generative AI with Java to life. The lineup was rounded out by sessions from community leaders like Brian Vermeer, Sandra Ahlgrimm, Simon Martinelli, Ana-Maria Mihalceanu, and many others – transforming the cinema experience into a true Java festival.

For those unable to attend in person: all sessions and keynotes are available on the official JAVAPRO YouTube channel.

Mentoring Hub: Deep Conversations Beyond the Stage

Another standout feature was the new Mentoring Hub format: here, experienced developers connected with newcomers and early-career professionals. In small groups, participants engaged in coaching, discussions, and idea-sharing – a format that moved beyond the traditional conference model. Rather than passively listening, attendees engaged in direct dialogue with seasoned Java experts.

Whether in sessions with Bruno Souza, discussions about "Next Steps for Developers," or exploring what it means to become a "mature developer," the feedback was overwhelmingly positive. "It added a kind of depth and connection that went beyond the usual conference experience," one participant remarked.

A Statement in Rint: JAVAPRO Returns to Paper

Another highlight: the return of JAVAPRO in print. The special edition "30 YEARS OF JAVA" was completely snapped up within just two hours – not a single copy remained at the booth. The revival of JAVAPRO as a print magazine was seen not only as a surprise but as a strong signal: technical journalism for developers remains as relevant as ever.

For those who missed out, the articles will be published gradually online at javapro.io.

The limited special edition is also being distributed at selected conferences and features articles on Core Java, GenAI, Microservices, Architecture, Frameworks & APIs, Testing, Security, Cloud, retrospectives on 30 years of Java, and future topics such as "GenAI with Java" and new JVM languages.

Community Spirit & Late-Night Talks

Beyond the technical program, JCON 2025 also shone on the human side. The "Hallway Track," spontaneous networking between sessions, and 1:1 speaker meetings were all buzzing with activity. Many attendees described the event as a "conference family" rather than just a professional gathering.

A social highlight was the VIP dinner on Wednesday evening – attended by members of the Java community, speakers, sponsors, and the organizing team. In a relaxed atmosphere, conversations were had, toasts raised, and visions for Java's future shared. The dinner reinforced the event's spirit of connection – and provided space for personal exchanges on equal footing.

And of course, there was celebration too: 30 years of Java – in great style. The community toasted three decades of innovation and collaboration – complete with a specially designed anniversary cake, shared and enjoyed together at the VIP dinner. A sweet moment that strengthened the sense of community even further.

Alongside expert talks and in-depth discussions, small playful details added to the fun: the colorful ribbons were especially popular. Those proudly wearing a rainbow of badge extensions like "JVM," "Star Trek," and "Maven" (or similar) demonstrated humor, broad technical knowledge – or simply that they'd had a great week.

Every year, the new JCON T-shirts are eagerly anticipated. This year's special 10th anniversary edition was highly sought after and proudly worn by many attendees during the conference. For some, these shirts have long become collector's items — and anyone who secured a limited edition now owns a piece of JCON history.

JCON Goes to the USA

But that wasn't all. Officially announced on stage in Cologne, and already sparking excitement: with the first U.S. edition of JCON @ IBM TechXchange from October 6–9, 2025 in Orlando, Florida, the successful JCON format will cross the Atlantic for the first time. Another milestone in its evolution – and a clear sign that the community spirit knows no borders.

An Anniversary that Connects – and Leaves Us Wanting More

Ten years of JCON, 30 years of Java, four inspiring days. JCON EUROPE 2025 was a resounding success – both organizationally and in terms of

content. It vividly demonstrated just how modern, diverse, and forward-looking the Java community is. A huge thank you to all participants, volunteers, and the entire organizing team, and especially to our fantastic speakers, partners, exhibitors, and sponsors who made JCON EUROPE 2025 possible.

The combination of innovation, practical relevance, mentoring, and genuine community spirit makes JCON far more than just a developer conference – it's a true home for Java enthusiasts. If you missed it, you definitely missed out.

Good news: the recap video is online, and the session videos are now being released step by step. Many talks are already available to stream — an excellent opportunity to revisit the highlights or discover sessions you may have missed.

Save The Date:

The next JCON EUROPE will take place from April 20–23, 2026, once again at the Multiplexkino Cinedom in Cologne!

#JAVAPRO #FRAMEWORKS #API

Actionable Intelligence from Running JVMs

Author:

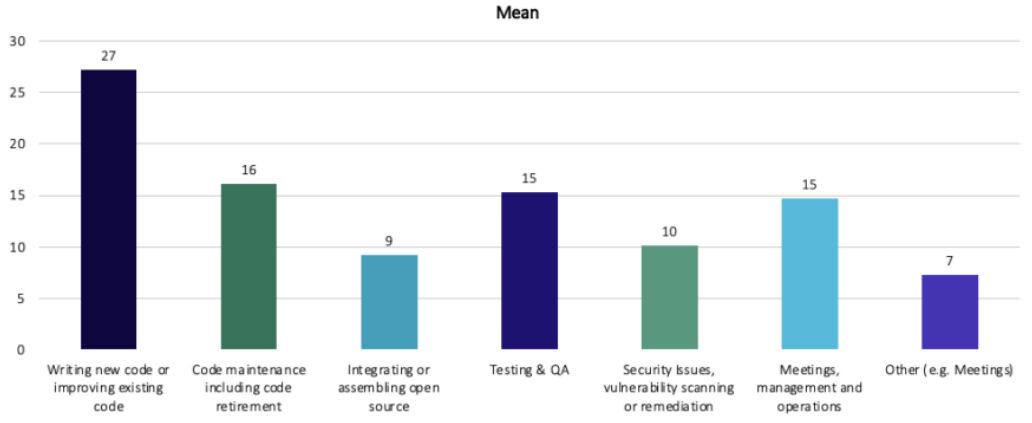
Robert Statsinger is a Solution Architect with a strong background in Application Security and Observability. He possesses over 30 years of software experience as a researcher, developer, architect, and trainer. His additional technology grounding includes Applications Performance Management, Enterprise Applications Integration, Artificial Intelligence and Cognitive Modeling, and Embedded Systems. Robert holds a Masters Degree in Computer Science from the University of Southern California.

Your JVMs are Pretty Smart - You Should Listen to Them

People don't watch or listen to their JVMs, which is a shame because they have a lot to teach.

Background: Developer Productivity

Censuswide recently performed a survey of developers and asked them to divide up how they spend their time in aggregate. The mean results appear below:



Columns 2, 4, and 5 are interesting for our discussion here and they tell us that the surveyed developers spend over 40% of their time on average on code maintenance, testing & quality assurance, and vulnerability management activities. This article discusses how deriving runtime intelligence from your JVMs can help you shrink columns 2 and 5 (and also take column 4 down a notch) so that you can spend more time in column 1. :)

What Can Your Running Production JVMs Tell You?

One very important thing that your production JVMs can tell you about is: all of the code they run. For our purposes here, what's important about that is that you can use this to tell you about all of the code that does not run. Let's say your development team started out as a small shop with informal coding standards and a get-it-done-right-now-before-our-funding-runs-out mentality. Many software development organizations start out like this - and then never change their practices. Fast forward 5 or 10 years and you might find yourself living in a world with:

- bloated global libraries full of platform code that need to get included with every build
- forgotten knowledge of whether older features are ever used anymore
- uncertainty about how much of that code even runs anymore

Even if this doesn't match your circumstances, chances are that over that 10 years you've needed to update the version of Java you're using (maybe more than once), or update a major framework on which your applications depend (looking at you, Spring Boot…), or completely

refactored things to move to a microservices architecture, or added new features that deprecated old ones. Any or all of these things are likely to result in some code getting left for dead.

Note that we are not just discussing unreachable code. Unreachable code can't run because there is no control flow path to it from the rest of the program. This is only a subset of the problem - the rest of the problem is made up of perfectly reachable code that could run, but never does - that is, public classes and methods that simply aren't being used. These two issues combine to produce a bunch of code that is unused and dead - it just lies there, adding technical debt.

Unused and Dead Code: The Silent Killer of Developer Productivity

A recent evaluation that took place in a large Java software shop resulted in the following:

- Nearly 50% of the code in a set of tested applications was simply never used
- As many as 200,000 specific maintenance activities per year was taking place against this unused code
- The actionable intelligence provided could save over 25 FTEs annually, or over 55000 hours of wall clock time.

Of course, your mileage may vary but even if these numbers are cut in half, what software development organization doesn't want to shrink column 2 in the developer productivity graphic by 27,000 hours of time every year?

There are a lot of ways to try to find unused and dead code, such as IDEs, static analysis tools, code coverage tools, and even manual inspection (Eeeek!). Some of these approaches tend to be incomplete (i.e. finding unreachable code but not unused reachable code) or provide false readings (i.e. code that would never actually run in production is exercised by testing tools). A code coverage tool such as JaCoCo is a good option in development/test environments but it tends to generate results based on specific test cycles. JaCoCo might be telling you about a bunch of

tests for code that you dont actually need if that code doesn't ever run in production. What you need to know about is code that never runs in production - that is what will enable you to speed up your pipelines and remove both unnecessary tests and unnecessary code.

Inventorying All of your Running Code

The undisputable truth about the code that actually runs in production comes from the things doing the running: your production JVMs themselves.

So, what if you did this:

1. In a production environment, when a Java program starts, perform a quick initial scan of all of the code that could run - you can do this by examining the classpath after all of the class loaders have run and including any extra jars that the program might use.
2. Record all of the code that runs, and when it last ran. Of course you'll want to observe for a while to make sure all of the production behavior of your application has a chance to be invoked, but considering the minimal effort involved, why not just sit back, let your apps run, and use the results as a verification - or refutation - of the results you get from the tools mentioned above?
3. Perform some simple arithmetic:

(All of the Production Code that Could Run) - (All of the Production Code that Does Run) = (All of the Production Code that Never Runs)

Once you've identified chunks of code that may not ever run, what's next? The first thing you could do is mark individual methods or classes as **@Deprecated**. That will start immediate labor savings for you, and if you're using Java 9 or later you could add the since and **forRemoval** attributes as well. You then have a process in place for timing the actual removal of that code. Ultimately, the benefit here will be reclaiming software development capacity by reducing technical debt.

Detecting All of your Running Vulnerable Code

A side effect of the above is that seeing all of the code that actually runs

means seeing which vulnerable code actually runs (it's a simple matter to compare the code that has run to a curated Java CVE knowledgebase). That means you can give a huge assist to software composition analysis

It's almost cliche to talk about the friction between security and development and the issues that can result but it's a measurable effect, and increasing the efficiency of Java CVE triage will increase the efficiency of your SDLC, and shrink the size of column 5 in the developer productivity graphic.

This use case also applies to code that you didn't write. Your outsourced or vendor-developed apps might well find themselves the targets of attack, so understanding their intrinsic attack surface based on running code allows you to have substantive, productive conversations with your vendors.

Conclusion

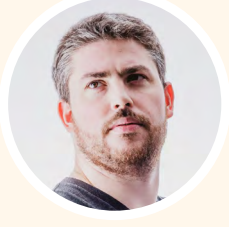
Many software development organizations suffer from the problem of dead code which accumulates over time as codebases grow and age. Anti-patterns such as boat anchors and kitchen sink libraries contribute to this problem. In addition, many software development teams encounter friction with security teams - their mutual goals ought to be aligned towards securing the business while maximizing developer speed and productivity. The „single source of truth" for what code you actually need - and what code actually makes you vulnerable - is your running production JVMs. They provide a treasure trove of useful information that can help you reclaim engineering capacity, reduce technical debt, and shrink your Java attack surfaces while aligning your application security and software development teams and practices.

#JAVAPRO #AI

AI-Powered Form Wizards: Chat, Click, Done

Author:

Loïc Magnette is a seasoned software developer with a strong background in consulting. Currently a senior developer at Oniryx, he specializes in Java and Angular, delivering innovative solutions and sharing his knowledge as a speaker. As a co-organizer of the Belgian Java User Group (BeJUG), he fosters connections within the developer community. Outside of tech, Loïc's passion for wildlife inspires his work and creativity.



Forms are everywhere—tax declarations, job applications, or even signing up for a new service. Although some forms are simple, many include ambiguous fields, complicated logic, or subpar design. This may frustrate users and make them more likely to make mistakes. Completing paperwork shouldn't be like solving a puzzle

Traditional forms, with their rigid structures and often confusing layouts, present a significant hurdle for users. Our objective was to dismantle this static paradigm and replace it with a dynamic, conversational interface. Instead of forcing users to navigate a pre-defined maze of fields, we envisioned an interactive experience where an AI assistant adapts in real-time. This approach fundamentally shifts the burden of data entry

and validation. A user chatting with an AI can dramatically reduce errors and streamline the overall process. Imagine a conversation, not a questionnaire, where the AI guides you through each step.

Through this article we'll take as an example a website that allows you to adopt puppies.

Back to Basics

Before we try to replace forms with a chatbot, we need to take a minute to just quickly review a few elements. What tools, library, framework we're using but also a few key AI concepts.

Tooling

Java has significantly improved its ability to integrate and communicate with AI over the last two years. These days, we have tools like Spring AI and LangChain4j that offer strong integration. LangChain4j will be used in this article. Because Quarkus offers a fantastic integration with LangChain4j, we will also use it as a result. Quarkus is well known for its cloud-ready, lightweight solution. However, the developer experience is a significant benefit in this instance. It's the ideal framework for working with AI, where you need to make frequent adjustments to your prompt. It offers hot reload while in development mode. There's no need to reload; you can simply make changes to your code and see the outcome right away.

AI Concepts

Before we go further, we're just going to take the time to recap some key concepts when using AI. Those concepts will be useful to build our chat-oriented approach, but if you are already familiar with AI and LangChain4j, you can directly jump to the next step.

Prompt and AI Service:

LangChain4j provides a rich API to interact with LLMs, allowing you to configure the integration. On top of that, the LangChain4j

extension of Quarkus makes everything even easier with enterprise-grade configuration. You, of course, need to first decide which model you want to use and add the corresponding extension. Then you just need to configure a couple of properties, such as the API key.

Then you can define your first AI-powered service. To do so, you only need an interface annotated @RegisterAIService. Finally, define a method and use the @SystemMessage and @UserMessage to provide your prompt to interact with the LLM.

```
@RegisterAiService
public interface Bot {

@SystemMessage(„““ You are an AI named Pawtrick answering
questions about puppy adoption. Your response must be
polite, use the same language as the question, and be
relevant to the question.
When you don‘t know, respond that you don‘t know the
answer and the bank will contact the customer directly.
„““)
    String chat(@UserMessage String question);
}
```

The @SystemMessage annotation is the first message delivered to the LLM. It gives the scope and preliminary instructions. It outlines the function of the AI service in the exchange.

The primary instructions sent to the LLM are defined in the @UserMessage annotation. Usually, it includes requests along with the structure of the anticipated response.

```
@RegisterAiService
public interface Bot {
    @SystemMessage(„““
            You are an AI named Pawtrick answering
questions about puppy adoption.
Your response must be polite, use the same language as the
question, and be relevant to the question.
```

```
When you don‘t know, respond that you don‘t know the
answer and the bank will contact the customer directly.
„““)
@UserMessage(„““You should try to answer the user
questions about puppy adoption.{question}„““)
String chat(String question);
}
```

Example of AI service with @SystemMessage and @UserMessage

Memory:

An LLM is stateless by definition, which means it will completely forget everything from one exchange to the next. We must give it a means of recalling the previous exchange if we hope to engage in meaningful dialogue with it. We refer to that as memory.

Using Quarkus, the chat memory is on by default. If you want different memory for each user, then you need to add a parameter to your AI Service method and annotate it with @MemoryId.

```
@RegisterAiService
public interface Bot {

    @SystemMessage(„““
            You are an AI named Pawtrick answering
questions about puppy adoption.
   Your response must be polite, use the same language as
the question, and be relevant to the question.
   When you don‘t know, respond that you don‘t know the
answer and the bank will contact the customer directly.
   „““)
@UserMessage(„““ You should try to answer the user
questions about puppy adoption.
{question}„““)
 String chat(@MemoryId long id, String question);
}
```

Example of AI service using @MemoryId

RAG:

RAG, often referred to as Retrieval Augmented Generation, is a technique for giving your model personalized knowledge, basically, information that it most likely lacks, to enable it to deliver insightful responses in your situation. To accomplish this, documents must be ingested, stored in a vector database, and made retrievable. Quarkus Easy RAG makes it simple to set up. Of course, you can - and probably should - go beyond this implementation. You can find more information on more complex RAG approaches at:

- https://glaforge.dev/talks/2024/10/14/advanced-rag-techniques/
- https://docs.langchain4j.dev/tutorials/rag/#advanced-rag.

Tools/function Calling:

Certain LLMs have the ability to invoke functions from your code. This gives you the chance to provide them with a wide range of capabilities. This is referred to as a tool or function calling. This provides you with the chance to further expand your capabilities. Basically, everything you can program, you can give them access to. For instance, you give access to your database or call a web service. Naturally, you must exercise caution because doing so could give the LLM permission to do risky actions. We don't want our LLM to be able to erase our database, for instance. Using Quarkus and LangChain4j makes it very simple.

```
@ApplicationScoped
public class PuppiesService {

    @Tool(„Get all the available puppies“)
    public List<Puppy>  getPuppies() {
        return Puppy.listAll();
    }
```

Example of tool definition

```
@RegisterAiService(
        tools = PuppiesService.class
)
public interface Bot {
```

```
    @SystemMessage(„““
            You are an AI named Pawtrick answering
questions about puppy adoption.
            Your response must be polite, use the same
language as the question, and be relevant to the question.
            When you don‘t know, respond that you don‘t
know the answer and the bank will contact the customer
directly.
            „““)
    @UserMessage(„““
            You should try to answer the user questions
about puppy adoption.
            {question}
            „““)
    String chat(@MemoryId long id, String question);
}
```

Example of AI service where a tool is provided

Ready?

Now that we have everything organized, we can attempt to use a chatbot to replace those forms.

How to Get Started

How do we go from messages from a user to fill in an object, validate the data, and give feedback to the user?

Saying something like „fill the object, validate the data, and write feedback for the user" was a naïve way to go about this. Using a large model, like the most recent OpenAI model, may yield some results, but this is not a given.

Rather, we ought to tackle the issue as we typically do when programming a complicated feature. In essence, break the issue down into smaller issues before assembling everything.

Fill in the Object

Most likely, the first step is to attempt to gather whatever information the user submits and to fill out the form or object.

Structured Output

To fill in the form, you are probably trying to fill in some POJO with the information. LLMs are capable of easily outputting data as JSON, but you need to give them a format. Then, with the help of LangChain4j, you can automatically parse it to a POJO.

Quarkus provides a placeholder ({response_schema}) that can be included in your prompt. It will dynamically be replaced by the defined schema of the method's return object, making the whole process easy. If you don't include it in your system and user messages, Quarkus automatically appends it after your prompt to enforce the format

```
@RegisterAiService
public interface FormHelper {

    @SystemMessage(„ You‘re an helpful bot that should fill
an object based on the user message“)
    @UserMessage(„““
            Fill the the provided object based on the
information given by the user.
            You should only update the field for which you
have information.
            A field that is null must be filled by the user.

            {userMessage}
            {adoptionForm}
            „““)
    AdoptionForm fillAdoptionForm(@MemoryId long id, String
userMessage, AdoptionForm adoptionForm);
}
```

You can use the @Description annotation to give a description of each field in your POJO to aid in the data mapping even further.

```
@Entity
public class AdoptionForm extends PanacheEntity {

    @NotNull
    @NotEmpty
    @Description(„the firstname of the person willing to
adopt“)
    public String firstName;
```

Usage of @Description to provide a description to a field

One key issue you might encounter, depending on the model you're using is that not all the LLMs are the same when it comes to structured output. The LLM may not follow your instructions even with Quarkus' assistance; in this scenario, you may need to improve your prompt. Because of this, I have found that using few-shot prompting, which involves displaying both positive and negative output, is a very successful strategy.

Memory

We should not only use the current user message and the current form state, but also the last few messages using the memory. If you are wondering, imagine the following exchange:

Bot: "When would you like to go pick up your puppy?"

User: "I would probably come the 4th of July."

Bot: "Sorry, but we're closed on this date. Could you provide another date?"

User: "Oh, of course, let's say the 6 then."

The LLM would not be able to understand that when the user says the 6, he means the 6th of July if we don't use the memory.

Validation

You should now verify the content of your POJO after enriching it. One crucial factor to consider is that the validation's output should be readable by an LLM, which should then be able to provide feedback to the user.

In my situation, I decided to verify the information in my POJO using bean validation. Writing validation and producing a uniform, structured validation output was a simple solution. If you wish to be even more helpful, you can even include a detailed error message.

```
@Entity
public class AdoptionForm extends PanacheEntity {

    @NotNull
    @NotEmpty
    @Description(„the firstname of the person willing to
adopt“)
    public String firstName;
    @NotNull
    @NotEmpty
    public String lastName;
    @NotNull
    @NotEmpty
    @Email
    public String email;
    @NotNull
    @NotEmpty
    @Pattern(regexp = „^\\+(?:[0-9] ?){6,14}[0-9]$“)
    public String phone;
```

Object with bean validation example

```
@Inject
Validator validator;

public Set<ConstraintViolation<AdoptionForm>>
validateForm(AdoptionForm adoptionForm) {
        return validator.validate(adoptionForm);
}
```

Using validator to find the error in the form

Based on this output, you can ask the LLM to provide some feedback to the user on what data is missing or invalid. You can even enhance this feedback with your own knowledge using RAG.

```
@SystemMessage(„““
            You‘re an helpful and polite bot who try to
help user fill a form.
            Your response must be polite, use the same
language as the question.
            „““)
@UserMessage(„““
            You are to assist the user with fixing
validation issues in the adoption form for a puppy.
            Address only one issue at a time.
            Respond directly to the user‘s queries or
comments.

            ---
            Validation Issues: {validationIssues}
            User Message: {userMessage}
            „““)
String helpSolveIssues(@MemoryId long id,
Set<ConstraintViolation<AdoptionForm>> validationIssues,
String userMessage);
```

Example of AI service generating feedback for the user based on the validation errors

Orchestration

All you need to do now is put everything together so you can help a user fill out the appropriate information. You can achieve this in several ways.

Probably the easiest method is to use some basic code to orchestrate everything and imperatively chain all the stages. You have a lot of control, but you must handle every situation by hand, and if your form is big and complex, it can get complicated.

```
 @Inject
 FormHelper formHelper;
public ChatMessage<AdoptionForm>
```

```
helpAdoptAPuppy(ChatMessage<AdoptionForm> chatMessage) {
        var userMessage = chatMessage.message();
        var filledAdoptionForm = formHelper.
fillAdoptionForm(userId, userMessage, chatMessage.form());
        var validationIssues =
validateForm(filledAdoptionForm);
        if (validationIssues.isEmpty()) {
            AdoptionForm.persist(filledAdoptionForm);
            var completionMessage = formHelper.
confirmValidForm(userId, userMessage);
            return new ChatMessage<>(completionMessage,
filledAdoptionForm);
        }
        var guidanceMessage = formHelper.
helpSolveIssues(userId, validationIssues, userMessage);
        return new ChatMessage<>(guidanceMessage,
filledAdoptionForm);
}
```

Example of simple orchestration using imperative code

To handle every scenario and give the user a more granular experience, you might also employ a workflow or rule engine. You can have a great deal of flexibility with this option.

The final choice is a more agentic approach; you could just design a new AI service and specify the fundamental steps (fill the POJO, validate the data, and provide the user with feedback). Here, you give the LLM the tools - the filling and validation mechanisms - and let it handle the rest.

And What About RAG?

We haven't really discussed RAG up to this point, but it may be helpful when filling out forms. You may request certain information, but the user is unsure of where to look for it. By incorporating RAG into your chat, the LLM may now give the user useful information to fill in specific information. Allowing the user to not only fill in the form but also give them some confidence in what they are filling in.

If you're using the quarkus-langchain4j-easy-rag extension, integrating the RAG is effortless. By default, Quarkus generates, discovers, and provides the retrieval augmentor to your AI service. If you don't use this extension, the process is quite easy. You need to define a retrieval augmentor like the one shown below.

```
@ApplicationScoped
public class RetrievalAugmentorExample implements
Supplier<RetrievalAugmentor> {
   private final RetrievalAugmentor augmentor;
    RetrievalAugmentorExample(PgVectorEmbeddingStore
store, EmbeddingModel model) {
        EmbeddingStoreContentRetriever contentRetriever =
EmbeddingStoreContentRetriever.builder()
                .embeddingModel(model)
                .embeddingStore(store)
                .maxResults(3)
                .build();
        augmentor = DefaultRetrievalAugmentor
                .builder()
                .contentRetriever(contentRetriever)
                .build();
    }
    @Override
    public RetrievalAugmentor get() {
        return augmentor;
    }
}
```

Then provide it to your AI service so it can use it when exchanging with the LLM.

```
@RegisterAiService(
        tools = PuppiesService.class,
        retrievalAugmentor = RetrievalAugmentorExample.
class
)
public interface Bot {
```

You could even go one step further and provide users access to a chat feature throughout your app, which would make it easier for them to navigate your website. We've all been in the position where we can't find the form or page we're looking for on a website. How helpful would it be to receive guidance from an assistant?

Going even Further?

Having an assistant who can help me navigate the process of filling out a form has already made the user experience much better. However, is there any way we could make a user's job even easier?

We could, of course, why don't we allow the user to upload files? The user experience might be improved even further. We could allow the user to provide us information in the form of a word, PDF, image, or even audio that we might employ. To do so, we need a multimodal model. After that, we have two choices.

- The content of the attachment can be extracted in the proper format by calling an LLM in the case of sound and images, or by using a library like Apache Tika for documents. After which, provide the user message together with the content for the remaining steps.
- Another option is to just provide the LLM the file containing our call and let it take care of it.

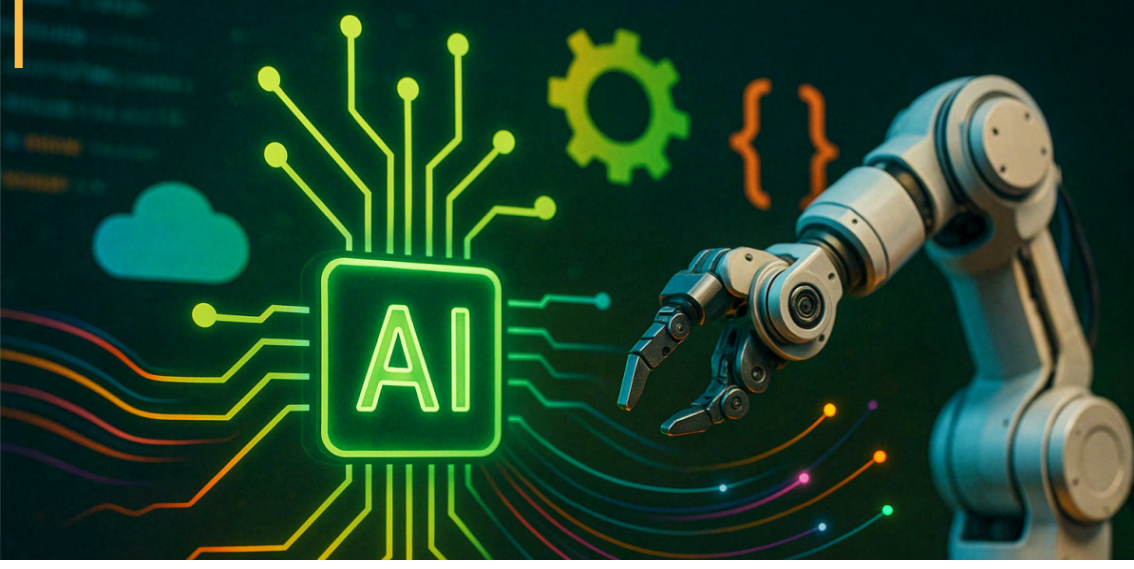
By doing so, we can automatically fill the form without the user having to do anything.

When using this approach, we still need to be careful about the potential risk of using an unknown file. We also need to pay attention to the context window size of our model, since the user could provide us with a very large file, and it would overflow the size of the context window.

Wrap Up

Transforming traditional forms into a conversational experience revolutionizes user interfaces. Using AI tools like LangChain4j and Quarkus, developers can build dynamic assistants that simplify data

entry. These assistants guide users step-by-step and validate input in real time. They can also provide more information on how to fill a form and where to find the information by harnessing the strength of RAG. Multimodal capabilities further enhance user confidence and ease by allowing them to let an LLM find the right information simply. This approach makes the user experience richer and helps improve the data quality by providing 24/7 support to the user. The AI-powered form represents a shift towards more intuitive digital experiences. It paves the way for advanced AI integration in everyday applications.



#JAVAPRO #AI

AI Tooling for Software Development: The Useless, the Useful, and the Future

Author:

Lize Raes is Product Manager and Developer Advocate at Naboo.ai, where she helps to build the developer productivity toolbox of the future. As collaborator at LangChain4j, she loves inspiring developers to apply AI in real-world applications. Committed to applying technology to societal challenges, Lize has embraced roles such as cochlear implant researcher at Ghent University, bioinformatics engineer for drug development software, and advisor to the Belgian government during COVID-19. In her free time, you will find her behind the piano or in her woodworking atelier.

JCON 2025 SPEAKER

AI-powered tools for software development can speed up the development process, improve code quality, and enhance team collaboration. The best tools for you and your team depend largely on your specific use cases and workflows. This article provides an overview of today's most popular AI development tools, outlining their ideal use cases and how to use them to improve efficiency without increasing code churn or introducing bugs.

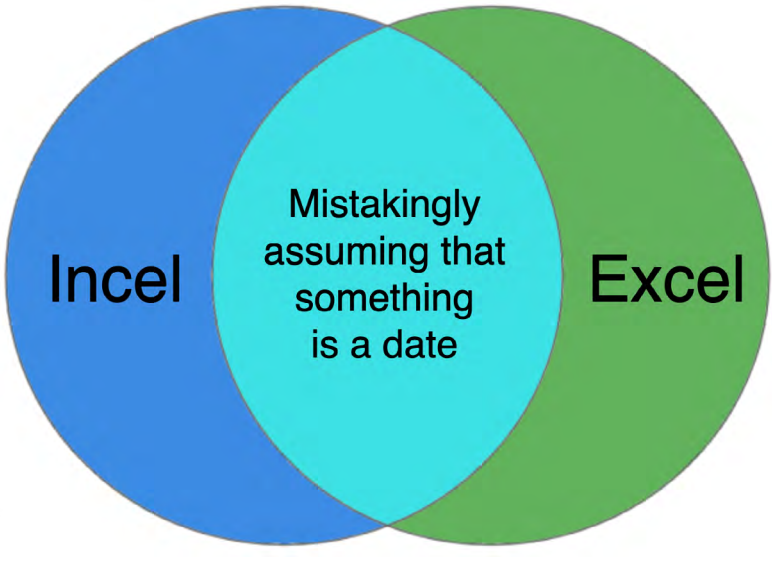
Do I Need to be Proficient in AI Tools as a Software Professional?

Yes and no.

First, like any development tool, AI-powered tools require some effort and experience to use them effectively. This is no different from mastering an IDE and its built-in features. Knowing how to use AI-assisted tools is gradually becoming a standard expectation for certain development jobs.

However, the AI tooling landscape is evolving at an incredibly fast pace, with some tools receiving substantial updates as frequently as every week. Many tools are still maturing, and major paradigm shifts continue to happen. For example, code completion is being replaced by AI agents that not only suggest code but also run tests and adapt dynamically. AI tools are expanding to assist in more stages of the development workflow. It's reasonable to wait and see which tools will become essential before fully committing to any specific one.

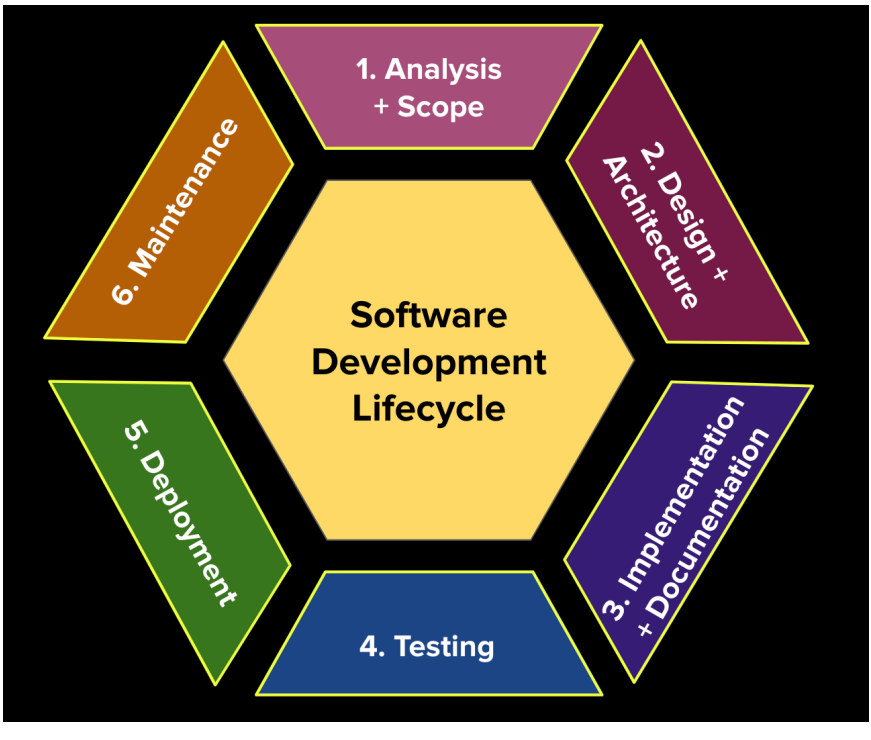
Another observation is that some companies still rely on pen and



AI tooling adoption will follow a similar pattern, with companies evolving at different speeds. Many Java-based organizations operate in regulated industries, handle sensitive data, or treat their code as intellectual property. For them, sending code, tickets, or technical specifications to a commercial AI provider is simply not an option. The most popular AI tools today don't offer on-prem support, so if you aren't an AI tooling expert yet, you are still safe… for now.

Which AI Tool Fits My Use Case?

When we think about AI tooling for software development, code assistants are usually the first to come to mind. But how much time do we actually spend writing code? Surprisingly little. AWS estimates that developers write code for about an hour per day, while GitLab suggests it accounts for only 25% of our work time. The rest is spent on tasks like testing, understanding the codebase, searching for information, attending meetings, managing releases, and, hopefully, a bit of innovation at the coffee machine. To maximize productivity, we need to look beyond code assistants and consider AI tools for the entire software development lifecycle (SDLC).



Where you spend most of your time depends on your role. There is no „typical" software developer. Some of us navigate decade-old codebases to modernize services, others rapidly build MVPs to stay ahead in the startup race, and some focus on optimizing build speeds and deployment processes. Your daily responsibilities will determine which AI tools provide the biggest productivity boost, and how you'll use them.

The first half of the article focuses on AI tools other than code assistants. This is where some of the biggest time savings lie, even if we don't always realize how many tedious tasks and bottlenecks are now solvable with AI. The second half covers code assistants in depth, featuring a

detailed comparison table and direct insights from the maintainers and creators of some of these tools.

Non Coding Assistants

Here's how AI can save time and reduce frustration across different parts of your workflow other than coding.

For when your team loses time due to changing customer requirements

"Developers will never be out of a job because customers and product managers don't know what they want."

It sounds funny and true, but LLMs are surprisingly effective at helping people refine their own ideas. AI can save significant time in defining scope and functionality by:

- Translating customer requests into formal specifications
- Clarifying edge cases and ambiguities upfront
- Limiting scope by generating frontend mocks that expose customers to a concrete representation of their idea, reducing last-minute surprises

By avoiding costly rework and misinterpretations, AI can prevent frustration on both sides. These capabilities are achievable with raw LLMs, or you can use specialized tools like ScopeMaster.

For when You Hate Creating and Managing Tickets

LLMs can automate task and resource planning by:

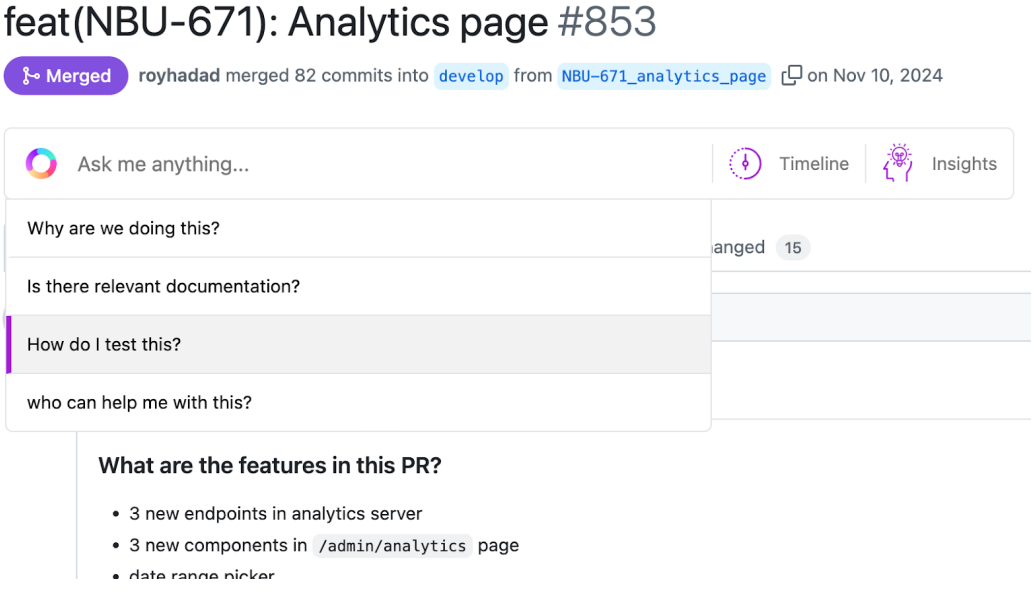
- Generating tickets based on tech specs + team expertise and availability
- Updating tickets dynamically when related actions occur (e.g., after

a PR merge)

You can use tools for this, like Atlassian Rovo, or set up a custom solution by prompting an LLM to generate Jira-API-compatible tickets based on specs and team member profiles.

For When You Can't Remember what a Ticket was About, or Where to Look

Naboo.ai is a smart context-retrieval tool that integrates with GitHub, GitLab, Jira, Confluence, Slack, and more. It lets you ask questions about your project and retrieves relevant discussions, tickets, PRs, and meeting notes without you needing to switch contexts.



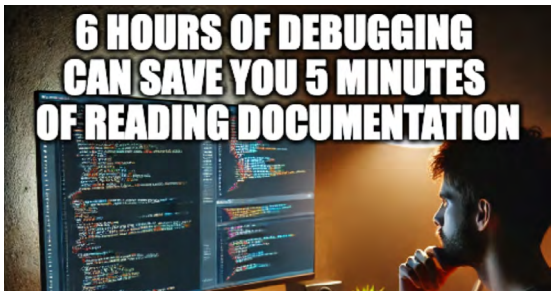
Considering that knowledge workers spend an average of 20% of their time searching for information (McKinsey, 2012), tools like this can significantly reduce wasted time and prevent bugs caused by missed updates. Naboo also helps with onboarding, diving into new projects, or picking up your tasks after a holiday.

For When You Can't Keep Up with the Latest Technologies and Architectures

Many software architects love having an „AI intern" with encyclopedic knowledge of frameworks and tech stacks. LLMs and coding models have been trained on vast amounts of code and can present a solid selection of options with pros and cons. However, in other ways they still behave like a clueless intern level, so always apply common sense when presented with AI-generated suggestions. Since LLMs are also trained

on some latest trends, they might over-recommend certain frameworks.

A well-crafted prompt can help filter responses. For example, I will always add "For the frontend, use HTML + Javascript only, avoid frameworks where possible."

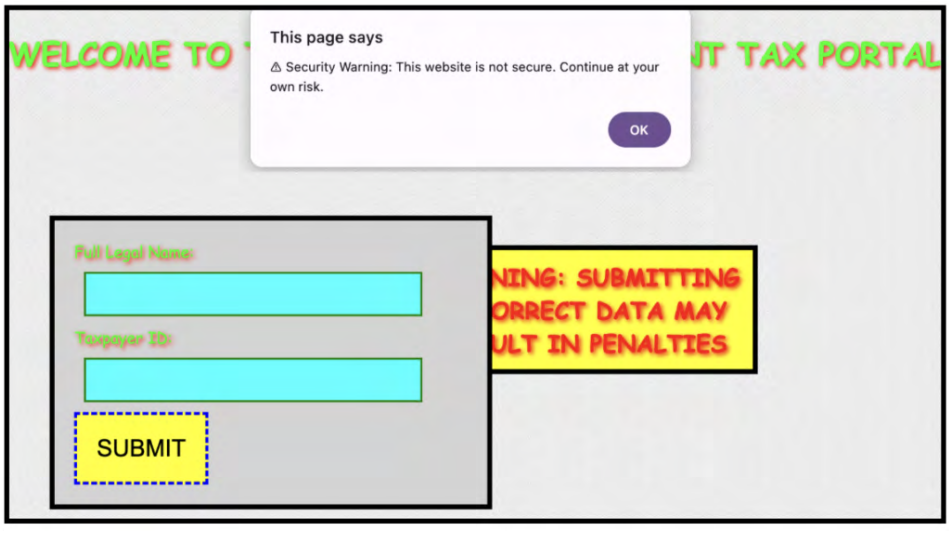


For Managing GitHub Issues and PRs

- CodeRabbit: GitHub plugin for PR summarization and reviews. Sometimes suggests improvements you can accept instantly. Free for open-source projects.
- Dosubot: GitHub plugin for triaging and answering issues. Free for public repos.

For When You Struggle with Keeping Your Documentation in Sync

Swimm.ai focuses on documentation and code explainability. It has different tools to ensure your documentation is accurate and up-to-date, including an auto-update mode that syncs documentation with code changes.



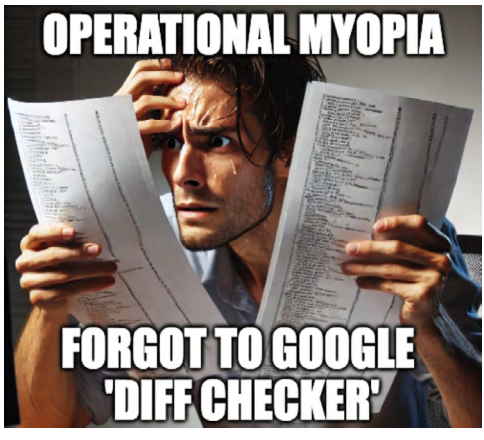
AI Models can be surprisingly helpful for design and UX decisions. Being helpful for the average human is exactly what they were trained for, so when it comes to user-friendly design suggestions and understandable guidelines, they shine. Many modern models (ChatGPT, Claude, Gemini, ...) support direct image inputs for design reviews, and specialized tools like Uizard and Galileo AI can generate UI designs and prototypes.

For Incident Management, Log Analysis, and Other Tailored Use Cases

Take a close look at your team's workflows:

- What tasks waste the most time?
- What tasks do people dread?

Chances are, an AI (or even a non-AI) tool already exists to help. Often, simply recognizing inefficiencies and searching for a solution is enough to make life easier.



Many tasks are automatable with relatively little effort. Good candidates include:

- **Incident management** (automated bug analysis and task dispatching)
- **Logfile anomaly detection**
- **User feedback analysis** (embedding vectors can help categorize common issues)
- **Email and Slack summaries**

It's easier than ever to build AI-powered tools on top of existing LLMs. If you're using Java, LangChain4j and SpringAI are great starting points.

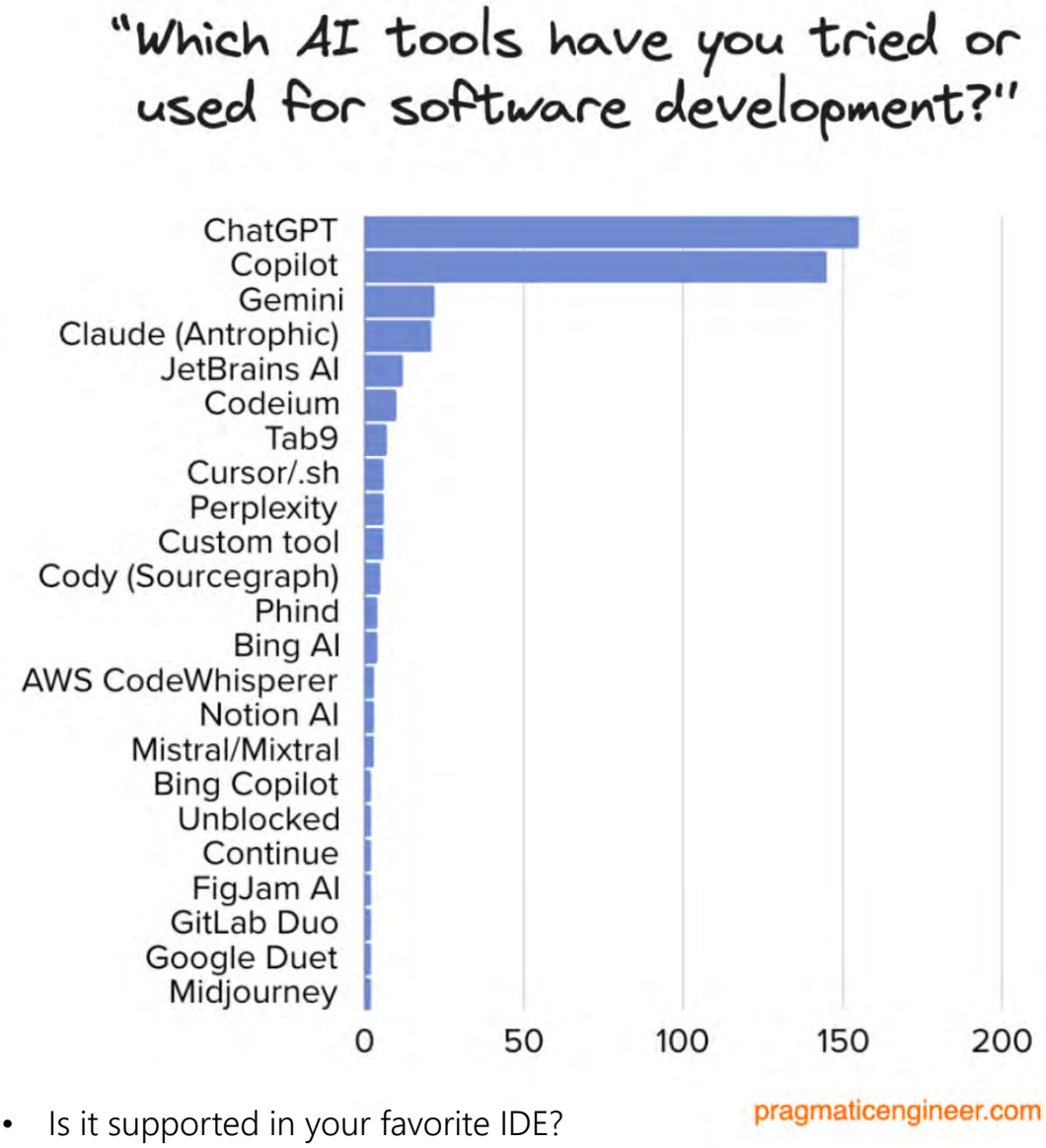
Coding Assistants

With options like GitHub Copilot, Cursor, JUNIE, Replit, and many more, it's easy to feel like you're falling behind if you haven't tried them yet. But let's ease that FOMO a bit.

A small-scale study conducted in Spring 2024 by Gergely Orosz for his Pragmatic Engineer newsletter (do subscribe!) surveyed 211 developers of various profiles.

If you've only experimented with ChatGPT and GitHub Copilot, you're far from alone. In the meantime, those tools have made significant progress and new players are rapidly gaining traction.

Which coding tool is right for your workflow depends on many factors:



- Is it supported in your favorite IDE?
- Do you need an on-prem solution for your employer?
- Do you want a proactive tool or rather perfect control over every line added?
- Do you want code completion or higher level brainstorming on architectural choices?
- Do you want to enforce your team's coding style or use the latest java features?

Without further ado, here's a huge comparison table (split in two for readability) to help you choose the right AI assistant for your team and workflow.

Tool	Code Completion	Chat	Smart Apply	Context Retrieval	Output Not Copyrighted Guarantee	Supported IDEs
Cursor	✅	✅	✅	unknown (works great) / agent mode	–	VSCode clone
Github Copilot	✅	✅	–	unknown (not great)	✅ (opt-in)	supports almost any IDE
DevoxxGenie	–	✅	–	manual select / full codebase / RAG (opt.), AST GraphRAG (upcoming)	–	IntelliJ and all JetBrains IDEs, VSCode
JUNIE	–	✅	edits files	custom / agent mode	–	IntelliJ, WebStorm, PyCharm (more underway)
JetBrains AI Assistant	✅	✅	✅ (error prone)	custom	–	IntelliJ and most JetBrains IDEs
Qodo (formerly Codium)	✅ (and fast)	✅	changes in new file (show diff)	manual select / agent mode	–	All JetBrains IDEs, VSCode, Visual Studio
Codeium's Windsurf AI	✅	✅	edits files	RAG / agent mode	✅ (trained on permissive license code)	custom IDE

Tool	Underlying Model	On Prem Option	Respects Code Flavor	Pricing	Agent Mode	Controls Tools	Nice To Haves	Watch Out
Cursor	Claude, GPT, Gemini, DeepSeek, custom model	–	✅ rules file	free tier, 20$/month	✅	terminal, tests, compiler, pluggable MCP servers	exceptionally good at propagating changes to all necessary files, intuitive 'Apply Diff' for each modified file	VSCode clone (separate IDE), no IntelliJ support
Github Copilot	Claude, GPT, Gemini	–	–	free tier, 10$/month		–	one of the first ones around and by far the most used	hallucinates methods (can't look over file boundaries)
DevoxxGenie	huge choice incl. local models	✅ local models	✅ prompting / guidelines	free, OSS (own key or local model)	– (may come)	–	great copy mode (files AND file structure) for use in Claude etc., huge model choice, free, OSS	no code completion
JUNIE	Claude-3.7	–	✅ guidelines.md	in preview	✅	terminal, tests, compiler	runs entire test suite each time (slower but quality guarantee), can later integrate deterministic IDE functionality	not on Windows, still in preview
JetBrains AI Assistant	custom model	only in full line code compl.	–	full line compl. free, rest 100$/yr	–	–	focus on supporting existing big customers	
Qodo (formerly Codium)	Claude, GPT, Gemini, DeepSeek	✅ (on request)	✅ best-practices.md	free for individuals	✅	terminal, tests, compiler, pluggable MCP servers	great attention to quality and guided processes, airtight on-prem option, extra PR tooling	
Codeium's Windsurf AI	custom model	✅ (on request, own hardware)	✅ fine-tuning on repo	free tier, 15$/month	✅	terminal, tests, compiler, pluggable MCP servers	trained on permissive license repos only (no copyright issues)	custom IDE
Replit	custom OSS model	–	–	20$/month	✅	built-in execution environment	builds project for your review, works well for standard code (user validation, payment module, ...)	supports limited languages, no Java! fails on non-standard use cases
Devin	OpenAI o1	–	✅ build up 'tribal knowledge'	500$/month (no seat limit)	✅	built-in execution environment, slack	can find and correct bugs, and execute tasks	slack-based workflow, ask and wait (a lot)
Moddy	plug your own model	✅ local models	✅ own recipes	free (own key or local model)	✅	apply OpenRewrite recipes	tailored to apply OpenRewrite tried-and-tested recipes (migration, exploration) to huge codebases	not a general coding assistant

Important Notes:

- This table reflects the state of AI coding assistants as of mid-March 2025. Given how fast these tools evolve, some details may be outdated by the time you read this. For the most up-to-date version or to contribute corrections, check out aitoolcomparator.com.
- Each tool is under active development, so missing features today may become available soon.
- While I've done my best to compile accurate information, there may be minor errors, especially considering the many versions and pricing plans. This table is meant as a high-level reference to help developers quickly identify tools that match their needs.

General Tips for Using AI Code Assistants

Store your prompts: If you've spent time crafting a clear prompt that works well, don't lose it! Many tools allow you to store custom shortcuts for frequently used prompts.

Use multiple tools: Each AI assistant has strengths and weaknesses. For example:

- Use Claude for brainstorming and leverage DevoxxGenie's convenient copy function.
- Generate well-structured tests with Qodo, which offers precise control over test cases.

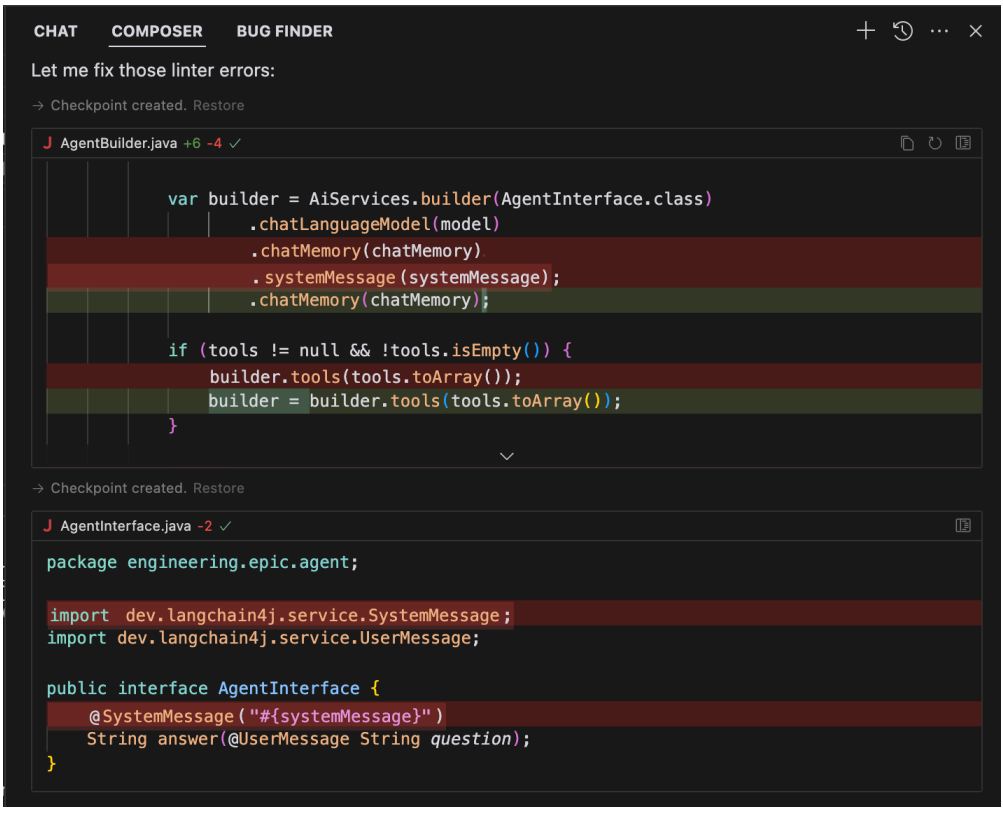
Go beyond autocomplete: Many developers assume AI assistants equal autocomplete, but that's not the only way to use them. Depending on your workflow, autocompletion might be more hassle than help. Take time to test different tools and ways of operating them, to find what actually improves your productivity

Use Cases Where Different Coding Assistants Excel

My own tests on small-scale repositories, along with feedback from fellow developers, provide only part of the picture. To get a broader perspective, I reached out to the maintainers of Qodo, JUNIE, DevoxxGenie, and Moddy to understand where their tools truly shine. Some of their insights are reflected in the comparison table above, while the rest are detailed here.

For Writing Fast Proof-of-Concepts (PoCs) and Small Features

For PoCs and small projects, Cursor can significantly speed up tremendously (that is, after you gained some experience with how to best go about it).



To get the best results when creating a new project, take it step by step and test along the way. Explicitly state requirements, even if they seem obvious. For example:
„When clicking the stop button, I want the state to reset entirely, as if the page was just opened." Without clear instructions, Cursor might introduce unnecessary custom logic to this stop button.

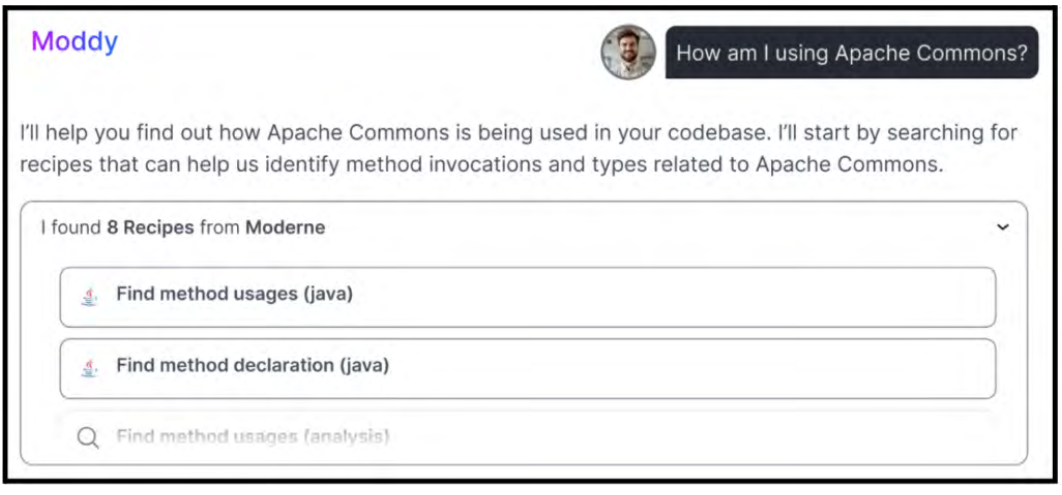
Use agent mode ('Composer') for tasks requiring the terminal. If you're feeling adventurous, activate YOLO mode to let it run terminal commands without your approval. Coding will feel like playing a video game… though at your own risk!

For Upgrading Large Legacy Repositories

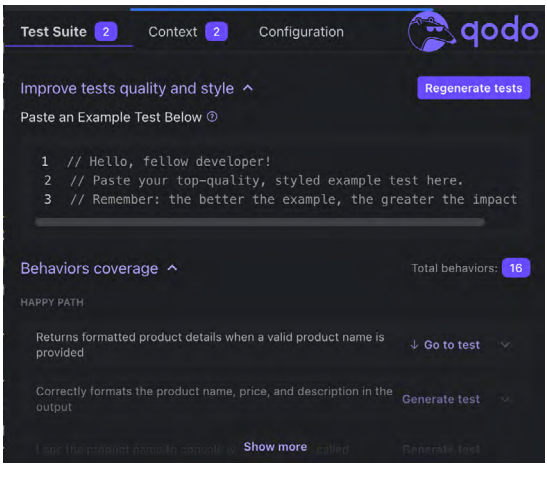
Moddy, the AI assistant from Moderne, leverages thousands of OpenRewrite code refactoring recipes, combining:

- LLM strengths in natural language processing with
- Proven refactoring techniques for structured, safe modifications

This makes Moddy particularly well-suited for tasks like large-scale legacy code modernization



For Controlled Changes and High Code Quality in Bigger Codebases



Qodo aims at bigger companies and code quality. Both its normal mode and its agent mode are structured in a way to support typical developer workflows. For example, when generating tests, Qodo lists happy paths, edge cases, and more, allowing developers to select which tests to implement.

This way of combining AI with known workflows, allows for more control. On top of this, changes are proposed in an entirely separate file, so we can cherry pick what we need. Working with Qodo feels less like trying to steer an unguided projectile than working with many of the other coding assistants.

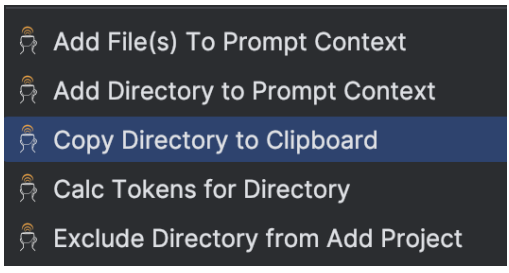
Extra goodies:

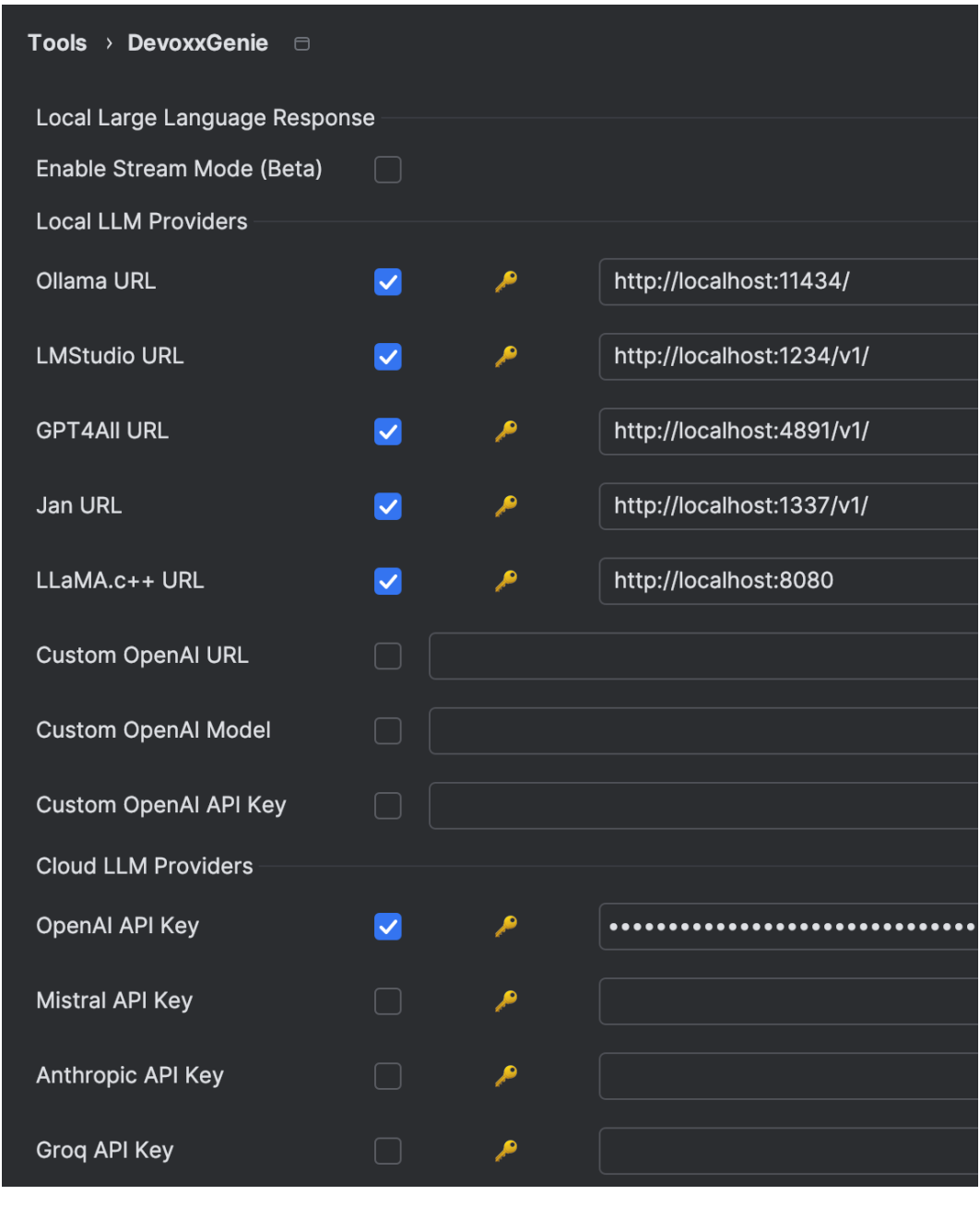
- they have PR-related functionality to make collaboration easier
- they have zero-retention agreements with their model providers so they offer single tenant SaaS, and can even offer airtight on-prem for companies on request

For When You Want to Work with Local Models

DevoxxGenie allows you to plug in virtually any model, including those

running locally on your own machine. This means you can have a fully functional coding assistant even while on a plane or in areas with low connectivity.





You can limit token usage by including only the necessary files (e.g., only the package you're working on, or just .java files). On top of that, DevoxxGenie allows you to calculate the request cost upfront before sending it. Best of all, DevoxxGenie is free (use your own key for commercial models) and open source.

For When You Want to Combine Deterministic Tools (refactoring, getters/setters generation, etc.) with the Power of AI

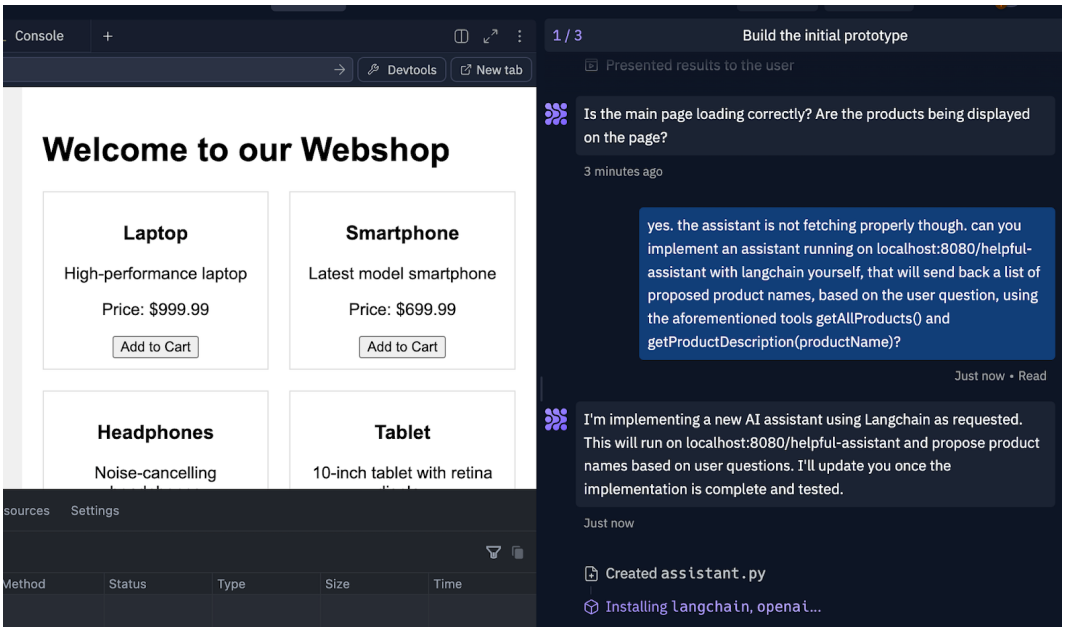
Though still in preview and evolving rapidly, JUNIE is expected to integrate IntelliJ's built-in trusted tools with AI-powered assistance. This combination is something I'm personally looking forward to, as it could combine the best of both worlds (trusted, deterministic tooling combined with AI's flexibility) and result in more reliable and high-quality code changes.

For Small Apps Consisting of Very Standard Blocks

Replit and Devin are both well-suited for quickly building small applications from scratch, as long as they consist of standard components like authentication and payment handling. These tools generate the code for you and provide a running instance to test or review. However, once you try to go beyond common patterns and try to implement something more customized, they tend to struggle.

A few specifics: Replit currently does not support Java but does work with Python, JavaScript, C++, and some related languages. Devin, on the other hand, is structured as a Slack-based workflow with a web browser interface. It can generate repositories, write code from scratch, fix bugs, and even test its own outputs.

The main downside of both tools is that they offer very limited intervention during the coding process. You're locked into an ask → wait → test workflow, without the ability to use your own IDE. This setup might work well when they consistently produce excellent results that you can trust, but currently we are not there yet.



Replit building an instance and asking for feedback

What Not to Expect from Coding Assistants

Gergely Orosz summarized these takeaways from his research last year:

- **The good:** When AI tools work well, they're a massive help in completing projects, increasing test coverage, and making experimentation easier.
- **The bad:** Poor output, hallucinations, and devs over-trusting these tools, top the list of complaints.

So, where do coding assistants struggle? When should you not rely on them?

- Making incremental changes on top of changes: At some point, the tool loses track of context and starts generating inconsistent results.
- Handling large-scale modifications: Most AI assistants struggle with major refactorings (except for Moddy, which is built specifically for that purpose).
- Navigating spaghetti code: When changing one line risks breaking three interconnected services, AI assistants are prone to creating more problems than they solve. JUNIE does mitigate this by running the full test suite for every proposed change, but it may keep iterating indefinitely just as other tools when facing such degree of complexity.

- Working with the latest libraries and language features: If they weren't included in the model's training data, the model won't know about them.

- Performing deterministic tasks: AI tools are, by nature, non-deterministic. For things like refactoring, debugging, renaming variables, detecting code smells, and linters, traditional tools are faster, more accurate, and cheaper.
- Security: While AI coding tools generally do a better job than the average developer when it comes to security, they are far from foolproof. It's always worth running a dedicated security scan with a tool like Snyk to catch vulnerabilities.

What Does the Future Hold?

I hope this article has given you insights into how AI can support your development workflow or helped you narrow down which tools to experiment with. Already today, AI can help us be more productive, reducing time spent on repetitive tasks and allowing more focus on higher-level problem-solving. That said, these tools are evolving at an incredible pace, so if none of them seem quite right for you yet, it might be worth checking again in half a year. Maybe by then, we'll have one tool to rule them all.

So Where Are AI Coding Tools Headed?

Right now, AI coding assistants feel a lot like clumsy interns with vast

encyclopedic knowledge: extremely useful and frustratingly limited. But the underlying models and the tools built on top of them are improving rapidly.

This year, the biggest focus is on agent mode: finding the right balance between proactiveness and control, integrating capabilities like terminal output monitoring and git access to create powerful and reliable coding agents.

Next year, I predict the focus will shift toward:

- Integrating deterministic tooling with AI assistants, for example, combining AI suggestions with refactoring engines and upgrade recipes for more structured, predictable changes.
- Expanding AI coding assistants to cover more of the development workflow: a code assistant might evolve into a full-fledged PR creator, reviewer, or even a system that translates formal specifications directly into working implementations. (see also: spec-driven development, spearheaded by Tessl.io)
- Taking over the inner development loop (write - run - test - repeat): AI will not just write and autocomplete code but also run, test, and iterate based on observed results.
- Helping with deployment and predicting next steps, especially in recurring processes where AI can anticipate and automate routine tasks.

Or maybe, we'll see an entirely unexpected breakthrough that changes everything. At this point, nothing would surprise me.

Will They Take Over Our Work?

One thing is certain: the way we work will change dramatically over the next five years. AI tools are already enabling some teams to move faster than ever before, despite their current limitations. And with the way trends are evolving, the transformation is only accelerating.

Here's what we're seeing:

1. Models are improving at an exponential rate, they are getting faster, smaller, and cheaper even beyond Moore's Law. Every month, a new model takes the top spot in benchmarks, whether it's a major commercial release or a compact open-source model like DeepSeek, Phi3, or Gemma3 that can run entirely on-prem.
2. Conceptual leaps are happening rapidly. We've gone from simple next-word prediction to predicting entire code blocks, and now, to autonomous agents that can navigate a codebase and execute terminal commands.

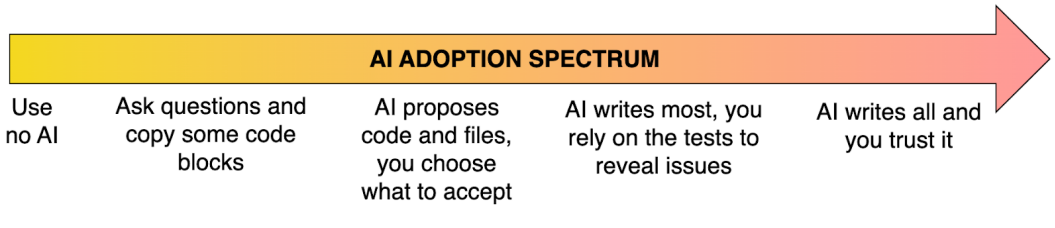
So yes, AI will take over part of our tasks at many companies in the coming years.

What Will Our Roles Look Like Then?

The biggest bottlenecks will no longer be writing the code itself. Instead, developers will shift focus to higher-level challenges, such as software architecture, UX design and writing extensible code that AI assistants can build upon.

Everything will move faster and more innovation will happen. Large enterprises with massive legacy codebases will have better tooling to modernize their stacks, while startups will accelerate even more aggressively. We're already seeing the rise of billion-dollar one-person companies, made possible by AI.

Depending on our company and our personality we will operate somewhere along this spectrum:



And a controversial prediction that might well turn out to be true:

	1980	1990	2000	2010	2020	2030
1st	Pascal	C	C	Java	Python	English
2nd	Fortran	Ada	Java	Javascript	Javascript	Mandarin
3rd	BASIC	Pascal	Javascript	PHP	Java	Hindi

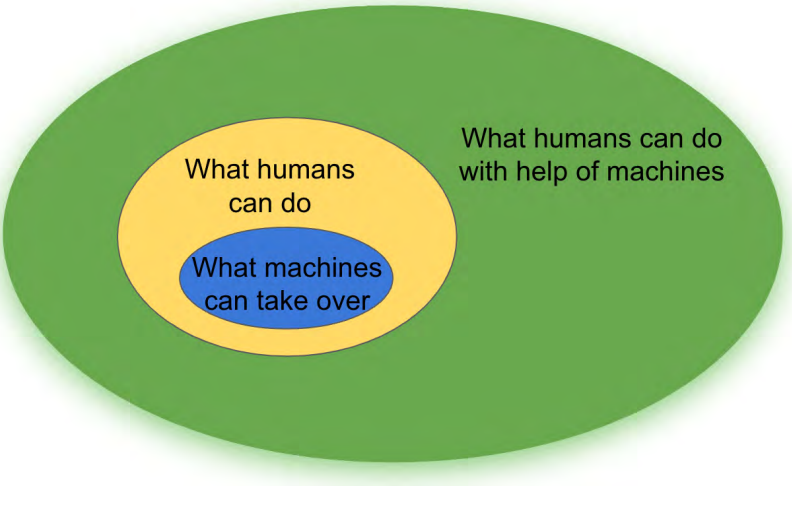
Most popular programming languages by decade

So, Will Developers Be Replaced?

History is full of revolutions where machines replaced human labor. In 1900, 40% of the U.S. workforce were farmers. Today, thanks to mechanization and progress, only about 1% of the population works in agriculture.

The invention of the computer wiped out entire job categories such as typists and switchboard operators. Yet, at the same time, computers created entirely new professions. Today, there are over 27 million software developers worldwide, a job that would not exist without them.

Machines eliminate some jobs but create new ones. This image illustrates what's going on (what humans can do with the help of machines - the green zone - is ever expanding).



So will we need as many pure software developers as we do now? Maybe, maybe not. But we will certainly need developers who know how to collaborate with AI. So, if you haven't already, start leveling up your act with your AI sidekick.

One thing's certain in this fast-moving world: going into security will

guarantee you a job. AI-powered criminals have driven Amazon's attack rate up 7.5x, and with LLMs tapping directly into emails, shopping carts, and databases via plug-and-play MCP servers, security is the next gold rush. You're welcome!

#JAVAPRO #AI

Move Fast, Break Laws: AI, Open Source and Devs (Part 1)

Author:

Steve Poole is an experienced JVM and Java Developer, Developer Advocate, DevOps Leader, and Security Champion with expertise in software supply chain security, AI, public speaking, education, and writing. An open-source contributor (Apache, Eclipse, OpenJDK) and developer relations expert. Regular presenter at international conferences on technical topics. Formerly with IBM and RedHat, with extensive experience from operating systems to JVMs to AI. Sci-fi lover, robot builder, and occasional mad scientist. Working with Java since its early days.

The software development landscape is rapidly changing, with legislation emerging as a key driver of industry trends. As our reliance on software and AI grows, so does our vulnerability to cybercrime, which is now a multi-trillion-dollar problem. This has caught the attention of regulators worldwide.

This article series explains the various regulatory efforts in play and summarises actions that developers and executives should consider as they prepare for 2025, the year of software legislation.

Part 1 (this article) covers the background, what a software supply chain is and thoughts on AI and open source.

Part 2 will explore how governments are working to create legislation and what the current status is.

Part 3 offers both a Software supply chain and an AI governance & compliance checklists for developers and executives to consider

Part 4 will discuss cybersecurity and incident reporting requirements, examines geopolitical compliance and liability management, and wraps up the series.

There's a lot to take in. I hope you're sitting comfortably.

Accountability Cannot be Outsourced.

I am not a lawyer. This document is a technical view of the legislation and regulations being developed or repurposed. It's imperative to get your own legal assessment when deciding if these elements apply to your situation. Having said that, some aspects are shared. The primary one is accountability. There's no dodging your responsibilities. That means wherever you are in the software supply chain, you have responsibilities to those consuming your software and those using it. Regulations collectively require organisations to assess, monitor, and manage third-party risks, and you'll have to prove that you did the right thing at the right time.

Blaming others without proper due diligence and safeguards is not a valid defence!

What is a Software Supply Chain?

As a developer, you may assume that a software supply chain is a fancy term for dependency management. However, a software supply chain refers to developing, delivering, and maintaining software, from code creation to deployment. It includes source code, dependencies, CI/CD pipelines, build tools, package repositories, cloud services, and infrastructure.

For engineering leaders, these are the elements to consider when ensuring security, compliance, and efficiency in software building and

deployment. For executives, software supply chains include all the aspects considered in risk management, regulatory compliance, and business continuity.

When discussing software supply chains, we're not just discussing the elements you control. Your software supply chains include all the software supply chains for all the software that is a part of yours, from dependencies to tools to platforms.

Why are Software Supply Chains so important?

When reviewing the ways that the bad actors exploit software, many of the attacks come from vulnerabilities. That's weaknesses, bugs and even misguided features. Another cyberattack weapon: malware can often be found embedded in legitimate downloads and libraries. Then, there are times when the bad actors create a fake component and trick others into using it. The software supply chain is the common thread to all these, even to some of the more sledgehammer-style attempts such as Denial of Service attacks.

From poor software engineering skills that result in code without the required security posture to CI/CD systems that are insecure and are subverted to include malware. In most cases, the root of these attacks can be traced back to some element of behaviour or lack of skill by the organisation producing the software.

The bad actors exploit these behaviours and insufficient skill sets in many ways, sometimes in incredibly devious and elaborate forms but usually in simple forms.

Why Now?

The bar for the bad guys is low. Developer community defences are relatively nonexistent, and their attention to the problem is still limited. No wonder cybercrime makes more money than the illicit drug trade. (Cybercrime costs the world around 9 Trillion Dollars annually and is growing rapidly.)

As the final straw in a long line of headline exploits, the Log4Shell

exploit demonstrated that the software industry could not police itself sufficiently. Software is far too valuable as an enabler of the modern world to be left undefended. The result is a worldwide effort to create practical and effective incentives to address the problems. From a government point of view, software supply chains (and hence developers and development organisations) are the foundational element behind the tsunami of attacks we see, and therefore, why they are the target for much regulatory activity.

What's the AI angle?

AI is still software and can be exploited in its own way. AI in the software supply chain (as models, training data, supporting libraries, etc.) is still vulnerable to the same attacks that traditional software is susceptible to. AI is also weak to new attack styles - poisoned models, for example- and is used to make other attacks more effective. Related to this is that although the AI technology emerging today might change the world tomorrow, the bad guys already use it to compromise software supply chains.

Regulatory reviews on the use of AI are resulting in specific legislation. However, once you look past the shiny AI exterior, the interior still consists of software, packages, data, build, test, deployment processes, tools, etc.

Whether you use AI as a tool in your software development process or as a competitive advantage in your production process, you and your organisation will likely have to consider both AI-specific and software supply chain legislation.

Open Source Implications

For decades, open-source software (OSS) has been synonymous with innovation. It is often driven by communities with minimal regulatory intervention or consideration. Even so,, Licensing, copyright, IP rights, and other issues can be viewed as a burden for those just trying to help others through their open-source efforts.

Unfortunately, the landscape is undergoing a dramatic transformation. Critical infrastructure and major industries' reliance on OSS and high-

profile supply chain attacks like SolarWinds and Log4Shell have exposed the vulnerabilities and governance gaps in many open-source projects.

The rise of AI models trained on open datasets and utilising open-source AI libraries has added to the complexity, value and potential risk many see in consuming open-source technologies.

Everyone acknowledges that open-source projects, in whatever form, are a critical component of the software stacks we rely on. It's estimated that 90% of modern applications are open-source.

Governments certainly don't want to curtail open-source development, but it's seen as a significant enabler for bad actors. Therefore, governments are beginning to demand greater traceability, provenance, and accountability from free and open-source software.

Since open source is increasingly perceived as a critical component of the regulated software supply chain rather than a mere free-for-all domain, the aim is to find a balance that has the least effect on the creators while still closing the doors to the bad guys.

The Bottom Line?

The bottom line is that open-source projects will inevitably gain new burdens, and some maintainers will undoubtedly decide to retire their projects. The hope is that the primary burden can be placed on those who consume the open-source software rather than the contributors themselves.

The proposed changes can be seen as unfavourable, as a big brother play to enforce arbitrary rules on the development community, but that's not entirely fair. It's certainly a set of big sticks, but we're seeing a realisation that, although invisible, software is as important as any other national infrastructure component. Infrastructure that needs to be high-quality and robust. Many of the elements of these regulations are about indirectly improving the software engineering skills of developers and their organisations to achieve the necessary standards.
How this plays out is a big TBD, but there is no doubt that this is real and already happening. Those who can reinvent their software creation

approach and keep the bureaucracy to a minimum will have a real competitive advantage

Next Time

Read **part 2** to understand more about how governments create regulations and what the list looks like. Global efforts to stem the tide of cyberattacks and manage AI usage are giving the software industry a relative tsunami of technical and business challenges to evaluate.

Code the Future at JCON USA 2025 @ IBM TechXchange!

OCT
06-09

JCON USA 2025
at IBM TechXchange

Join the ultimate Java event, October 6-9 in Orlando, Florida!

- Learn from top experts
- Dive into hands-on workshops
- Network with developers from around the world

Mark Stoodley
Chief Architect for IBM Java

Edward Burns
Principal Architect for Java on Azure

Register Now and Save 30%

#JAVAPRO #AI

Move Fast, Break Laws: AI, Open Source and Devs (Part 2)

Author:

Steve Poole is an experienced JVM and Java Developer, Developer Advocate, DevOps Leader, and Security Champion with expertise in software supply chain security, AI, public speaking, education, and writing. An open-source contributor (Apache, Eclipse, OpenJDK) and developer relations expert. Regular presenter at international conferences on technical topics. Formerly with IBM and RedHat, with extensive experience from operating systems to JVMs to AI. Sci-fi lover, robot builder, and occasional mad scientist. Working with Java since its early days.

JCON 2025 SPEAKER

Part 2 (this article) will explore how governments are working to create legislation and what the current status is.

Part 3 offers both a Software supply chain and an AI governance & compliance checklists for developers and executives to consider

Part 4 will discuss cybersecurity and incident reporting requirements, examines geopolitical compliance and liability management, and wraps up the series.

There's a lot to take in. I hope you're sitting comfortably.

Accountability Cannot be Outsourced.

I am not a lawyer. This document is a technical view of the legislation and regulations being developed or repurposed. It's imperative to get your own legal assessment when deciding if these elements apply to your situation. Having said that, some aspects are shared. The primary one is accountability. There's no dodging your responsibilities. That means wherever you are in the software supply chain, you have responsibilities to those consuming your software and those using it. Regulations collectively require organisations to assess, monitor, and manage third-party risks, and you'll have to prove that you did the right thing at the right time.

Blaming others without proper due diligence and safeguards is not a valid defence!

Behind the Scenes of Regulatory Development

Governments trying to create regulations in any field need experts to help develop an appropriate and helpful approach. Although there is minimal trust in the software industry to self-police, there is broad recognition that the expertise to do so is within the software industry.

Government bodies, existing standards groups, industry groups, and others collaborate to create standards, frameworks, and regulations.

At the highest level, there are several involved in this discussion.

- United Nations
- International Organization for Standardization (ISO)
- World Trade Organization (WTO)
- Financial Action Task Force (FATF)
- International Telecommunication Union (ITU)

The critical takeaway is the global nature of the responses. While laws and regulations in one country may differ regarding a particular element,

common standards and approaches are at the root of the legislation.

Currently (and crudely), the US focuses on cybersecurity, the EU on AI control, and China on data privacy. Other countries have their initiatives, but the general pattern is clear.

There is an informal distribution of work by world governments. Different ones focus on building standards for different elements of the efforts underway. One country may formalise the standard by converting some or all of it into a law, while another may just include the standard itself as the requirement to comply with.

The Brussels Effect

This term https://en.wikipedia.org/wiki/Brussels_effect refers to the impact large economic groups have on others. In this case, it relates to the EU's standards having a de facto effect on companies outside the EU. There are other variants of the term, but the net is that regardless of your country, software use in your organisation is likely to require compiling with the sum of the legislation being developed. As multinational organisations need to comply with all the laws in all the countries they do business in, they naturally distil a union of these laws and regulations and ultimately require their suppliers to follow suit.

The "Brussels Effect" obviously works outside the EU, so the net takeaway is that we will all have to deal with the sum of all the regulations from all the major economic blocks. It's simply a matter of time.

The Laws, Regulations and Other Instruments in Play

At some point in a topic like this, there just have to be lists of government controls. Take time to read through the list to learn how organisations worldwide are approaching these challenges. It's worth noting that few of the individuals involved are software engineers. Most involved see ‚software' as a scary, magical, and now uncontrolled element. The general concept applied to software is that it is like any other manufacturing component and can be dealt with similarly.

There are people at all levels who understand software, but they are in

the minority and struggle to be effective. As this disconnect is explored and corrected, many court cases will inevitably occur. The devil is in the details, and there are many details.

Hence, I advise focusing on the common elements and taking all reasonable steps to create a strong, robust software supply chain and software engineering culture. See later for the checklist

Regulation and Compliance for AI

European Union	The **EU AI Act (Regulation (EU) 2024/1689)** is the first comprehensive legal framework for AI, taking a risk-based approach.
	Risk Categories: AI systems are classified as unacceptable risk (banned), high risk (regulated), and limited or minimal risk.
	Obligations for High-Risk AI: Providers must implement trustworthiness and safety measures, including rigorous risk management, testing, and data quality controls.
	Conformity Assessment: High-risk AI requires an AI quality management system and compliance audits.
	Liability Considerations: The pending AI Liability Directive may extend software liability for AI-related harm.

More Information: *https://digital-strategy.ec.europa.eu/en/policies/european-ai-act*

United States	**AI Governance:** Multiple different guidelines, regulations and bills.
	NIST AI Risk Management Framework: Voluntary but widely influential guidelines for AI risk mitigation.
	Proposed Federal Legislation: Bills such as the Algorithmic Accountability Act aim to introduce impact assessments for AI
	Regulatory Oversight: The FTC and the **AI Bill of Rights** promote fairness and transparency.
	State-Level Initiatives: NYC Local Law 144 mandates bias audits for automated hiring tools.

More Information: *https://www.nist.gov/itl/ai-risk-management-framework*

United Kingdom	**Ethical and Sector-Specific AI Oversight**
	No comprehensive AI Act, but sector-specific regulations apply.
	Guidance-based approach with oversight from regulatory bodies.
	Algorithmic Transparency Standards encourage disclosure for AI systems in public sector applications.

More Information: *https://www.gov.uk/government/publications/ai-regulation-a-pro-innovation-approach*

China	**Strict AI Regulation**
	Algorithmic Recommendation Rules (2022) mandate government registration for AI algorithms.
	Generative AI Regulations (2023) require compliance with state-approved values.
	Transparency & Security Controls: Mandatory **content moderation, bias mitigation, and human oversight.**

More Information: *http://www.cac.gov.cn/2023-07-13/c_1694165100702412.html*

Regulation and Compliance for AI

European Union	**Cyber Resilience Act & NIS2 Directive**
	Cyber Resilience Act (CRA): Mandates secure-by-design principles, prohibits products with known vulnerabilities, and enforces post-release patching.
	NIS2 Directive: Extends security requirements to more organizations, mandates **supply chain security audits.**

More Information: *https://digital-strategy.ec.europa.eu/en/policies/cyber-resilience-act*

United States	Executive Orders and Standards
	Executive Order 14028 (2021) mandates secure development practices and **Software Bill of Materials (SBOMs) f**or government procurement.
	NIST Secure Software Development Framework (SSDF) outlines best practices for source code integrity and vulnerability management.
	Cybersecurity Maturity Model Certification (CMMC) requires defense contractors to meet security benchmarks.

More Information: *https://www.nist.gov/itl/executive-order-14028*

United Kingdom	Product and Infrastructure Security
	Product Security and Telecommunications Infrastructure (PSTI) Act (2022) bans default passwords, requires vulnerability disclosure policies.
	NIS Regulations Update expands supply chain security oversight to new sectors.

More Information: *https://www.ncsc.gov.uk/news/product-security-act*

China	Cybersecurity Law and Supply Chain Controls
	Multi-Level Protection Scheme (MLPS) mandates security testing and state-approved infrastructure for critical applications.
	Cybersecurity Review Process requires foreign software vendors to pass national security audits.

More Information: *http://en.mps.gov.cn/n2254314/index.html*

Next Time

Read **part 3** to understand the sorts of checklists and evaluations that developers and their executives have to consider around software supply chain matters and AI governance & compliance

Part 3 is already available online and will also appear in the upcoming PDF issue.

#JAVAPRO #DATABASE #ANALYTICS

A Tale of Two Runtimes: Setting Up Your Local Java Development with Flink

Author:

Passionate about creating functional and efficient software, Alexandros Charos was introduced to the world of software engineering in his teenage years and has never stopped learning since. Currently, as a Software Development Manager at OPAP, Greece's leading lottery and Sportsbook operator, Alex has over 15 years of experience in software engineering, specializing in distributed systems. He has led numerous successful projects worldwide, demonstrating expertise in designing and implementing scalable solutions. In his free time, he enjoys playing football, reading literature, and running.

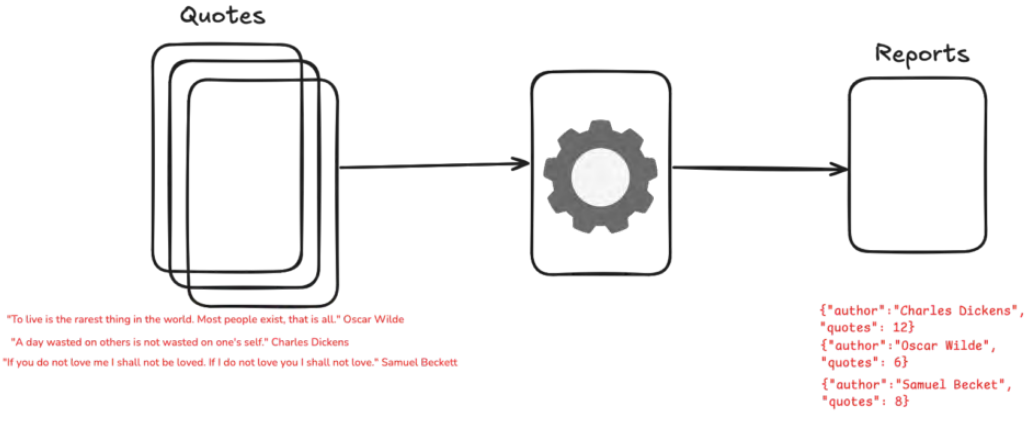
It was the best of builds, it was the worst of builds. One ran effortlessly in the IDE, the other stubbornly broke at runtime. Welcome to the tale of two runtimes: your local development setup and the Flink cluster where your job is ultimately meant to live.

In this post, we'll walk through how to set up your Flink Java project so you can develop, test, and run it locally with ease. Just like Charles Dickens's works, if you're just getting started with Apache Flink, the setup

can feel a bit intimidating...
Flink is a powerful framework for building scalable, fault-tolerant, and real-time stream processing applications. We will go through together how to run and test your jobs locally. Whether you're debugging from your IDE, spinning up a local cluster, or writing your first test cases, this post walks through the nuts and bolts of setting up a smooth development workflow with Flink.

Our Use Case

We'll be working with a rather simple use case in our example, just for the sake of having something to work with. Let's imagine that we're consuming a stream of quotes by various authors, and we wish to keep track of each author's quotes during the past day. We want to have this count updated every minute and, after each day, to have it reset again



The quotes are served from a Kafka cluster, and we have to send the updates to another one.

This is a typical Flink use case, so we have decided to implement it with Flink. So let's dive right in!

Setup

Let's take a quick look at how you would go about setting up your local project.

Maven Archetype

There's a handy archetype that Maven provides to get you started, so we'll use this one:

```
mvn archetype:generate \
-DarchetypeGroupId=org.apache.flink \
-DarchetypeArtifactId=flink-quickstart-java \
-DarchetypeVersion=1.20.1 \
-DgroupId=gr.charos.literature \
-DartifactId=quotesjob \
-Dversion=0.1 \
-Dpackage=gr.charos.literature \
-DinteractiveMode=fals
```

One of the main advantages of using the Maven archetype is that it comes pre-configured with the shade goal in its packaging phase, which builds a Flink job artifact file (JAR) that can be deployed in a Flink cluster.

The archetype is also kind enough to create a main class that acts as the entry point for our job.

```
package gr.charos.literature;
//... Omittting imports

public class DataStreamJob {

  public static void main(String[] args) throws Exception
{
    // Sets up the execution environment, which is the
main entry point
    // to building Flink applications.
    final StreamExecutionEnvironment env =
      StreamExecutionEnvironment.getExecutionEnvironment();

    /*
     * Here, you can start creating your execution plan
for Flink.
     *
     * Start with getting some data from the environment,
like
     *   env.fromSequence(1, 10);
     *
     * then, transform the resulting DataStream<Long>
using operations
```

```
     * like
     *   .filter()
     *   .flatMap()
     *   .window()
     *   .process()
     *
     * and many more.
     * Have a look at the programming guide:
     *
     * https://nightlies.apache.org/flink/flink-docs-stable/
     *
     */

    // Execute program, beginning computation.
    env.execute(„Flink Java API Skeleton“);
  }
}
```

You might be tempted to rename it to something of your choice. Just make sure that you also do so in the pom.xml file!

```
<execution>
  <phase>package</phase>
  <goals>
    <goal>shade</goal>
  </goals>
  <configuration>
    <createDependencyReducedPom>false</
createDependencyReducedPom>
    <artifactSet>
      <excludes>
        <exclude>org.apache.flink:flink-shaded-force-
shading</exclude>
        <exclude>com.google.code.findbugs:jsr305</exclude>
        <exclude>org.slf4j:*</exclude>
        <exclude>org.apache.logging.log4j:*</exclude>
      </excludes>
    </artifactSet>
    <filters>
      <filter>
```

```
      <!-- Do not copy the signatures in the META-INF
folder.
Otherwise, this might cause SecurityExceptions when using
the JAR. -->

        <artifact>*:*</artifact>
        <excludes>
      <exclude>META-INF/*.SF</exclude>
      <exclude>META-INF/*.DSA</exclude>
      <exclude>META-INF/*.RSA</exclude>
        </excludes>
       </filter>
    </filters>
    <transformers>
      <transformer implementation="org.apache.maven.
plugins.shade.resource.ServicesResourceTransformer"/>
      <transformer implementation="org.apache.maven.
plugins.shade.resource.ManifestResourceTransformer">
<strong>        <mainClass>gr.charos.literature.
DataStreamJob</mainClass></strong>
      </transformer>
    </transformers>
  </configuration>
</execution>
```

Flink & Java Versions

A thing we need to keep in mind is Flink's support for Java versions. It is documented here.

Kafka Cluster(s)

It's always useful to set up a local Kafka broker so as to be able to play with your implementation, along with some sort of UI where you may view the broker's state and perhaps manipulate it. The docker-compose file below works just fine for that purpose.

Heads up, this docker-compose.yml file has an image (init-kafka) whose purpose is to simply create the topics which we'll be using.

```
version: „2.2“
name: quotes-cluster
services:
  quotes_broker:
    image: apache/kafka:latest
    hostname: quotes_broker
    container_name: quotes_broker
    expose:
      - ‚9092‘
      - ‚9093‘
    ports:
      - ‚9092:9092‘
      - ‚9093:9093‘
    networks:
      - quotes-cluster-network
    environment:
      KAFKA_NODE_ID: 1
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
‚CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT,PLAINTEXT_
HOST:PLAINTEXT‘
      KAFKA_ADVERTISED_LISTENERS:
‚PLAINTEXT://127.0.0.1:9093,PLAINTEXT_HOST://host.docker.
internal:9092‘
      KAFKA_PROCESS_ROLES: ‚broker,controller‘
      KAFKA_CONTROLLER_QUORUM_VOTERS: ‚1@quotes_
broker:29093‘
      KAFKA_LISTENERS: ‚CONTROLLER://:29093,PLAINTEXT_
HOST://:9092,PLAINTEXT://:9093‘
      KAFKA_INTER_BROKER_LISTENER_NAME: ‚PLAINTEXT‘
      KAFKA_CONTROLLER_LISTENER_NAMES: ‚CONTROLLER‘
      CLUSTER_ID: ‚test-cluster-id‘
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_LOG_DIRS: ‚/tmp/kraft-combined-logs‘
    healthcheck:
      test: [ „CMD-SHELL“, „kafka-topics.sh --bootstrap-
server kafka:9092 --list“ ]
```

```
      interval: 5s
      timeout: 10s
      retries: 5

  quote-broker-ui:
    image: tchiotludo/akhq
    container_name: quote-broker-ui
    ports:
      - „8082:8080“  # AKHQ Web UI
    depends_on:
      - quotes_broker
    networks:
      - quotes-cluster-network
    environment:
      AKHQ_CONFIGURATION: |
        akhq:
          connections:
            source-cluster:
              properties:
                bootstrap.servers: „quotes_broker:9092“


  init-kafka:
    image: confluentinc/cp-kafka:6.1.1
    depends_on:
      - quotes_broker
    networks:
      - quotes-cluster-network
    entrypoint: [ ,/bin/sh‘, ,-c‘ ]
    command: |
      ”
      # blocks until kafka is reachable
      kafka-topics --bootstrap-server quotes_broker:9092
--list
      echo -e ,Creating kafka topics‘
      kafka-topics --bootstrap-server quotes_broker:9092
--create --if-not-exists --topic authored-quotes
--replication-factor 1 --partitions 1
      kafka-topics --bootstrap-server quotes_broker:9092
```

```
--create --if-not-exists --topic authored-quote-counts
--replication-factor 1 --partitions 1

      echo -e ,Successfully created the following topics:‘
      kafka-topics --bootstrap-server quotes_broker:9092
--list
      ”

networks:
  quotes-cluster-network:
    driver: bridge
```

Our Implementation

Pretty much outside the scope of this article, I’ll just broadly describe the implementation here so as to have a general idea of how we implemented this solution. As mentioned earlier, this is a very simple Flink use case.

Source Stream

Our source is a Kafka cluster where messages come serialized as JSON in the format below:

```
{
  „author“:“Charles Dickens“,
  „quote“:“A day wasted on others is not wasted on one‘s
self.“
}
```

We’ll be serializing this in a “Quote” record and using the JsonDeserializationSchema class for deserializing from the source topic.

```
Properties kafkaProps = new Properties();

Config config = getConfig(args);

kafkaProps
  .setProperty(„bootstrap.servers“,config.
sourceBootstrapServers());
```

```
kafkaProps
  .setProperty(„group.id“, config.sourceGroupId());

JsonDeserializationSchema<Quote> jsonFormat =
  new JsonDeserializationSchema<>(Quote.class);

FlinkKafkaConsumer<Quote> kafkaConsumer =
  new FlinkKafkaConsumer<>(config.sourceTopic(),jsonFormat,
kafkaProps);

DataStream<Quote> textStream = env.
addSource(kafkaConsumer);
```

In order to do this, we need to pull in a couple of dependencies (one of the Kafka connector, and one for Flink's JSON support):

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-json</artifactId>
  <version>${flink.version}</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-connector-kafka</artifactId>
  <version>${flink.connector.kafka.version}</version>
</dependency>
```

The Process Function

We wish to keep track of each author's quote count of the past day, so we'll opt for a simple implementation where we'll key our source by the author's name and then keep their quotes in our state.

We'll be windowing by processing time, with 1-minute windows since this is how often we wish to update our counts.

```
DataStream<AuthorQuotesCount> authorQuotes =
    textStream
      .keyBy(Quote::author)
      .window(
        TumblingProcessingTimeWindows.of(Duration.
ofSeconds(5)))
      .process(new QuoteCountFunction());
```

Our QuoteCountFunction implementation class will update the state with the observed quotes within that window and push the latest state

```
public class QuoteCountFunction
  extends ProcessWindowFunction<Quote, AuthorQuotesCount,
String, TimeWindow> {
  private transient ValueState<AuthorQuotes> currentState;

  private final StateTtlConfig ttlConfig =
    StateTtlConfig
      .newBuilder(Duration.ofDays(1)) // Keep last day
      .setUpdateType(StateTtlConfig.UpdateType.
OnReadAndWrite)
      .setStateVisibility(StateTtlConfig.StateVisibility.
NeverReturnExpired)
      .cleanupInRocksdbCompactFilter(1000)
    .build();

  @Override
  public void open(OpenContext openContext) {
    ValueStateDescriptor<AuthorQuotes> mState =
      new ValueStateDescriptor<>(„state“, AuthorQuotes.
class);

    mState.enableTimeToLive(ttlConfig);

    currentState = getRuntimeContext().getState(mState);
  }
@Override
public void process(String key,
                    Context context,
                    Iterable<Quote> elements,
```

```
                          Collector<AuthorQuotesCount> out)
throws Exception {
  AuthorQuotes  current = currentState.value();

  if (current == null) {
    current = new AuthorQuotes(key);
  }
  for (Quote element : elements) {
    current.getQuotes().add(element.quote());
  }
  out.collect(new AuthorQuotesCount(key,current.
getQuotes().size()));
  currentState.update(current);
  }
}
```

Our QuoteCountFunction implementation class will update the state with the observed quotes within that window and push the latest state

```
public class AuthorQuotes {
    private final String author;
    private List<String> quotes;
    public AuthorQuotes(String author) {
        this.author = author;
    }

    public String getAuthor() {
        return author;
    }


    public List<String> getQuotes() {
        if (quotes == null) {
            quotes = new ArrayList<>();
```

Sink

Eventually, we'll publish to our sink something along the lines of the following recordfor the author in question.

```
public record AuthorQuotesCount(String author, Integer
quotesCount) {}
```

...and we'll serialize with JSON!

```
Properties destKafkaProps = new Properties();

destKafkaProps
  .setProperty(„bootstrap.servers“,config.
destinationBootstrapServers());

FlinkKafkaProducer<AuthorQuotesCount> kafkaProducer =
  new FlinkKafkaProducer<>(
    config.destinationTopic(), jsonSerialization,
destKafkaProps);

authorQuotes.addSink(kafkaProducer);
```

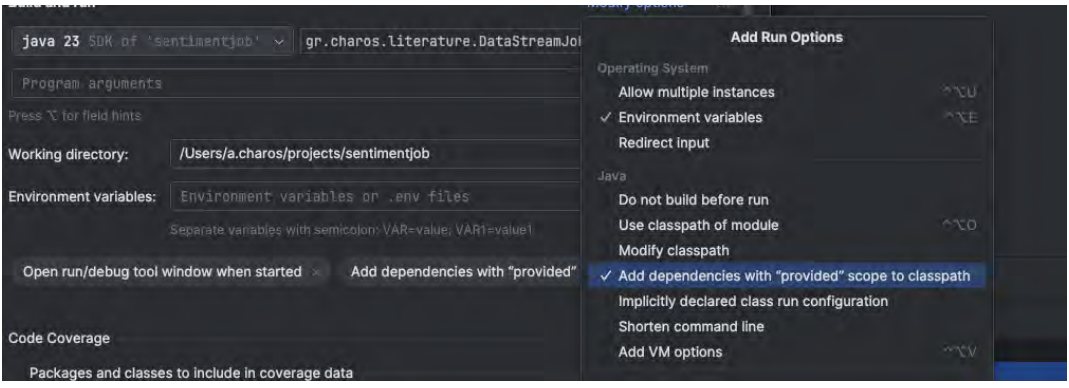
Running on our IDE

One of the best things about working with Flink is how easy it is to start it up directly from your IDE and debug any initial teething issues that are bound to come up, especially if we're only getting started with it.

The first time I tried to run my main method from my IDE. I got the following error.

```
Error: Unable to initialize main class gr.charos.
literature.DataStreamJob
Caused by: java.lang.NoClassDefFoundError: org/apache/
flink/api/common/serialization/DeserializationSchema
```

This happened because our project declares the Flink libraries with the provided scope. Of course, this is fair play since our job is supposed to run on a flink cluster where exactly those libraries will be on the runtime. To run and debug our job from the IDE, we'll need to include the provided-scoped libraries in the classpath.



This makes writing our first iteration amazingly simple and allows us to start iterating fast on our job!

Local Flink Cluster

Running outside our IDE helps (to a point) reduce the "works on my PC" cases and also helps when someone simply wants to spin up our job but does not want to worry with the actual codebase. There are also some subtle differences in terms of the execution environment that Flink launches when it runs locally.

```
final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
```

When running locally, you will be provided a LocalStreamEnvironment here, as per Flink documentation:

The LocalEnvironment is a handle to local execution for Flink programs. Use it to run a program within a local JVM - standalone or embedded in other programs. The local environment is instantiated via the method ExecutionEnvironment.createLocalEnvironment(). By default, it will use as many local threads for execution as your machine has CPU cores (hardware contexts). You can alternatively specify the desired parallelism. The local environment can be configured to log to the console using enableLogging()/disableLogging().

There are two ways to set up your local Flink cluster. Depending on various factors, you may choose to use one or the other. I'm going to leave both approaches here and pick and choose those that best suit your setup.

In my opinion, the Flink binary will be helpful whichever way you decide,

so it makes sense to have the binary also set up your cluster.

Option 1: Flink binary

You can download Flink from the official website (https://flink.apache.org/downloads/) or install it using SDKMAN. Please note that SDKman may be slightly behind the latest released version.

Option 2: Docker-Compose

There's a thorough guide on setting up in the official Flink documentation (https://nightlies.apache.org/flink/flink-docs-master/docs/deployment/resource-providers/standalone/docker/#flink-with-docker-compose). Going with session mode, where you can submit (via the web ui or the CLI) jobs, seems like the way to go

```
version: „2.2“
services:
  jobmanager:
    image: flink:latest
    ports:
      - „8081:8081“
    command: jobmanager
    environment:
      - |
        FLINK_PROPERTIES=
        jobmanager.rpc.address: jobmanager

  taskmanager:
    image: flink:latest
    depends_on:
      - jobmanager
    command: taskmanager
    scale: 1
    environment:
      - |
        FLINK_PROPERTIES=
        jobmanager.rpc.address: jobmanager
        taskmanager.numberOfTaskSlots: 2
```

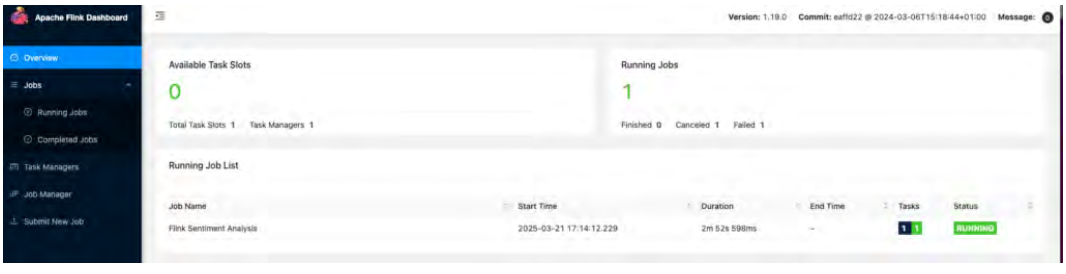
Deploying your Job

You can then run your job once you build it:

$ mvn clean package

$ flink run target/quotesjob-0.1.jar

Navigating to http://localhost:8081 will give you an overview of your



Unit (...and Integration, I suppose...) Testing

Ahhh.. Without a doubt, my favourite part!

Cue here the standard lecture that we are obliged to give (or receive) in terms of what a Unit actually is and what the extent of a unit test ought to be before it actually becomes an integration test...

We should definitely keep our business code as isolated as possible and test those closures irrespective of Flink. However, when we want to test functions which use internal state and timers, ie, interact more with Flink's runtime, things can get a bit tricky. For such cases, Flink provides us with test harnesses.

I think Flink's training repository is helpful in any case. Still, it's worth reviewing the exercises and tests, if only to see the smoothness with which they are implemented and gain some inspiration for setting up your testing codebase.

Harness

Let's say we've written a ProcessWindowFunction that keeps state or works with timers. How do we test it without spinning up an entire Flink job? That's

where Flink's test harnesses come in. You can read more about them here:

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-test-utils</artifactId>
  <version>${flink.version}</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-test-utils-junit</artifactId>
  <version>${flink.version}</version>
  <scope>test</scope>
</dependency>
```

There is a huge variety of examples for harness tests in Flink's GitHub

Harness Setup

I think the biggest pain in using harness testing is setting the harness up. There is some plumbing required to get it going, but once you do, it is really powerful and allows you to simulate various conditions (processing time, for example) and allows for thorough testing.

It is a KeyedInputStream we'll be testing, with one input, so we'll be using the respective class that Harness provides us.

```
private KeyedOneInputStreamOperatorTestHarness<String,
Quote, AuthorQuotesCount> testHarness;
```

This harness will be instantiated with a window operator and keying information (key extractor and key type info.

Setting up the WindowOperator instance is what troubled me most when attempting to use the harnesses for the first time, mainly due to the many arguments that we need to pass to it...

TL;DR: What the WindowOperator Needs

- Type of window (e.g., TumblingEventTimeWindows.of(...))
- Key selector function
- Key serializer
- State descriptor
- The wrapped ProcessWindowFunction (via
- InternalIterableProcessWindowFunction)

```
ListStateDescriptor<Quote> stateDesc =
  new ListStateDescriptor<>(
    „window-contents“,
    STRING_INT_TUPLE.createSerializer(
      new ExecutionConfig()));
WindowOperator <String,  // Key Type
                Quote,   // IN type
                Iterable<Quote>, //IN type of Iterables
that will be passed
                AuthorQuotesCount, // OUT type
                TimeWindow // Type of window
                > windowOperator =
  new WindowOperator<>(
    TumblingEventTimeWindows.of(Duration.ofMillis(100)),
    new TimeWindow.Serializer(), // Time window serializer
    Quote::author, // Our Key function
    BasicTypeInfo.STRING_TYPE_INFO.createSerializer(
      new ExecutionConfig()), // Key serializer (String in
our case)
    stateDesc, // See above
    new InternalIterableProcessWindowFunction<>(
      new QuoteCountFunction()), // wrapping our process
function
    ProcessingTimeTrigger.create(), // Processing time
trigger
    0, //    Will not work
    null //  with lateness today..!
);
```

We can now start writing unit tests to ensure our function works as expected.

```
@Test
public void testProcessCount() throws Exception {
  // manipulate processing time
  testHarness.setProcessingTime(0);
  // push elements and their timestamp
  testHarness.processElement(
    new StreamRecord<>(
      new Quote(„Orwell“,“Freedom is the right to tell
people what they do not want to hear.“),
      10));
  testHarness.processElement(
    new StreamRecord<>(
      new Quote(„Huxley“,“After silence, that which comes
nearest to expressing the inexpressible is music.“),
      20));
  testHarness.processElement(
    new StreamRecord<>(
      new Quote(„Orwell“,“Happiness can exist only in
acceptance.“),
      50));
  testHarness.processElement(
    new StreamRecord<>(
      new Quote(„Dickens“,“There are dark shadows on the
earth, but its lights are stronger in the contrast.“),
      100));
  testHarness.processElement(
    new StreamRecord<>(
      new Quote(„Steinbeck“,“Power does not corrupt. Fear
corrupts... perhaps the fear of a loss of power.“),
      100));
  // first window comple, start of second window.
  testHarness.setProcessingTime(100);
  assertEquals(2, testHarness.getRecordOutput().size());
  long orwellRecordsCount =
    testHarness.getRecordOutput().stream()
      .filter(p->p.getValue().author().equals(„Orwell“)).
count();
```

```
          long huxleyRecordsCount =
    testHarness.getRecordOutput().stream()
      .filter(p->p.getValue().author().equals(„Huxley“)).
count();
         long dickensRecordsCount =
    testHarness.getRecordOutput().stream()
      .filter(p->p.getValue().author().equals(„Dickens“)).
count();
          long steinbeckRecordsCount =
    testHarness.getRecordOutput().stream()
      .filter(p->p.getValue().author().equals(„Steinbeck“)).
count();

  assertEquals(1, orwellRecordsCount);
  assertEquals(1, huxleyRecordsCount);
  assertEquals(0, dickensRecordsCount);
  assertEquals(0, steinbeckRecordsCount);
  int orwellQuotes = testHarness.getRecordOutput().
stream()
    .filter(
      p->p.getValue().author().equals(„Orwell“))
    .findFirst().get().getValue().quotesCount();
          int huxleyQuotes = testHarness.getRecordOutput().
stream()
    .filter(
      p->p.getValue().author().equals(„Huxley“))
    .findFirst().get().getValue().quotesCount();
  assertEquals(2, orwellQuotes);
  assertEquals(1, huxleyQuotes);
}
```

Using Flink's MiniCluster

Another approach to Unit (or Integration test if we're being pedantic) is to use Flink's own MiniCluster. This essentially allows you to work directly with the API that you use when defining the job, making it much more straightforward to set up.

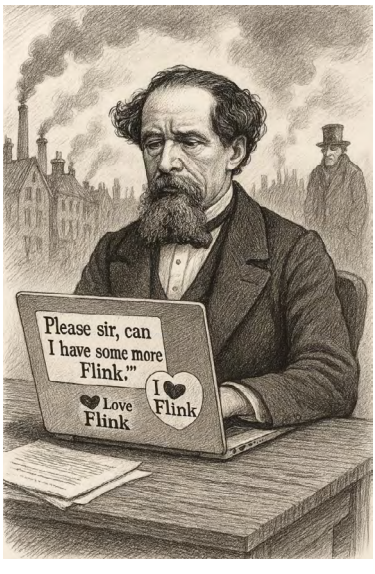
This approach is especially helpful if you use Event-Timed windows.

Honestly, the official and other online documentation is relatively weak on how to set it up to work with processing time, which is where test harnesses really shine.

When to Pick Which Approach in Unit Testing?

- Use test harnesses for low-level control, especially with processing time or detailed operator testing.
- Use MiniCluster for broader integration tests or event-time testing,

Summing Up



IDE setup? ✅

Local cluster? ✅

Docker Compose? ✅

Unit & Integration tests? ✅

We're all set!

We've gone through the core steps of setting up a local development environment for Flink: running and debugging directly from the IDE, spinning up a local or Docker-based cluster, and writing both unit and integration tests using Flink's test harnesses and MiniCluster.

This setup should give you a solid foundation for developing and testing your Flink jobs with confidence. There's a lot more to explore from here, but you now have all the essentials to start experimenting, iterating, and building your first streaming applications.

Happy coding!

#JAVAPRO #PROJECTMANAGEMENT

The Framework Illusion: Let's Fix Your Value Delivery

Author:

Marin Niehues is Chief Strategy Officer at PLOIN GmbH and advises companies on how to combine strategy and working methods in a meaningful way. He shows how to achieve real success with a clear focus and solid structures.

https://www.linkedin.com/in/marin-niehues/

We all love our Frameworks. From Agile to DevOps to SAFe, organizations are constantly looking for the "one single solution" that will solve their challenges once and for all. While the impulses behind this search are understandable—who doesn't want a proven, repeatable solution to complex problems?—the reality is that frameworks, by themselves, do not inherently deliver results.

Instead, frameworks merely provide a structure that can help teams and organizations do their actual work more effectively. They can outline rituals, suggest pathways, and offer conceptual scaffolding, they do literally just "frame work". But ultimately, they are only as impactful as the market-facing results provided with them.

In this article, we'll explore why frameworks are more accurately viewed as tools rather than turnkey solutions, and how to address the deeper fundamentals to focus on the real goal: building an effective value delivery

system that enables teams to ship great products to their customers.

Part 1: The Appeal of Frameworks

1. Simplicity in a Complex World

In a world full uncertainty where nothing is a given, Frameworks promise to bring clarity to chaos by offering a set of "best practices" or clearly defined steps that, in theory, can be applied to any team or project. When someone is grappling with rapid market changes, tight deadlines, budget constraints, and cross-functional misalignment, it's tempting to believe that adopting a particular framework can solve those deep-seated issues instantly.

2. Replicability and Benchmarking

Business leaders love frameworks because they seem to provide replicability. You see a successful company using a certain set of practices (see Toyota or Spotify), read a glowing case study, and assume that copying that framework will yield the same results. Benchmarking against industry leaders becomes more straightforward when their processes are neatly packaged. This leads organizations to assume, often mistakenly, that if they adhere to the same set of processes, they will share in the success.

3. The Illusion of Control

Frameworks also offer a sense of control. By standardizing how teams plan, execute, measure progress, and even communicate, frameworks can create the comforting appearance that the organization is "on the right track." Standardization can be beneficial, but it can also conceal underlying systemic problems—if managers only focus on following a process, they might overlook critical signals of dysfunction or undervalue important creative or situational adaptations that don't fit neatly into the framework's prescribed steps.

Part 2: Why Frameworks Are Not Results

1. Outcomes Versus Outputs

One of the most common misunderstandings comes from mixing up the frameworks with the actual results that a company wants to achieve. A framework might result in a standardized set of "outputs"—for example, defined sprints, stand-up meetings, backlogs, and so forth—but these outputs are not the true end goals. Real outcomes revolve around delivering value: higher-quality products, faster time to market, better customer satisfaction, and healthier profit margins. It doesn't matter if a backlog is perfect or if a stand-up meeting is executed flawlessly. What matters is whether the organisation reliably delivers valuable outcomes that customers recognise and appreciate.

2. Cultural Foundations

Culture is often the make-or-break factor in whether a framework can help an organization or remains just a set of processes. A framework by itself doesn't dictate whether people trust each other, feel empowered to experiment, or are encouraged to give honest feedback. Without a culture that promotes continuous learning and shared ownership of goals, the best framework in the world can turn into a series of mundane checkboxes.

- Psychological Safety: Teams must feel safe to fail fast, experiment, and innovate..
- Shared Purpose: There has to be a clear, well-communicated vision or goal that the framework is helping the team move towards.
- Leadership Commitment: Leaders must model the values of adaptability, transparency, and accountability.

3. Frameworks Are Tools, Not Transformations

A transformation involves changing the organization's DNA—its policies, structure, and attitudes toward risk, collaboration, and learning. A tool like Scrum or Kanban is part of the organizational toolkit. Tools facilitate

tasks, but they don't accomplish them on their own. It's like having a hammer: you can have the best hammer in the world, but if your strategy for building a house is flawed, or if you lack the skills, materials, and teamwork necessary to use it effectively, you won't end up with a sturdy home - or any house at all.right track." Standardization can be beneficial, but it can also conceal underlying systemic problems—if managers only focus on following a process, they might overlook critical signals of dysfunction or undervalue important creative or situational adaptations that don't fit neatly into the framework's prescribed steps.

4. The Limits of Standardization

Although frameworks allow for a certain level of standardisation, this can also make people complacent or inflexible, unless it is managed carefully. When the environment changes—and it always does—teams that are overly committed to "doing the framework correctly" may resist necessary adaptations, missing opportunities to optimize or reinvent their processes in the face of new challenges.

Part 3: The Real Work Happens Outside the Framework

1. Technical Excellence

Many organizations assume that adopting a framework like Scrum or Kanban will magically boost the quality of their technical delivery. While frameworks can provide cadence, opportunities for retrospection, and clearer communication channels, they don't inherently improve code quality, architecture, or integration processes. Real technical excellence involves:

- Robust Engineering Practices: Continuous integration, continuous delivery, automated testing, and well-structured code reviews.
- Scalable Architecture: Designing systems with flexibility, reusability, and maintainability in mind.
- DevOps Mindset: Breaking down silos between development and operations to ensure faster, more reliable releases.

No sprint plan or Kanban board can compensate for the resulting technical debt or slow delivery times.

2. Organizational Design

True agility and value delivery can be severely hampered by poor organizational structures. Layered hierarchies, siloed departments, and lengthy approval chains can crush the speed and flexibility that any framework attempts to achieve. Consider:

- Cross-Functional Teams: Are teams composed of all the skill sets needed to deliver an increment of value from end to end?
- Ownership and Accountability: Is it clear who owns which parts of the product or service delivery, and do they have the autonomy to make decisions?
- Lean Governance: Are decision-making pathways streamlined, or are they bogged down by bureaucratic overhead?

3. Continuous Improvement Mindset

Frameworks like Scrum encourage retrospectives. But a retrospective is only as valuable as the willingness to learn from and act on the insights uncovered. Establishing a continuous improvement culture requires:

- Humility: An acceptance that even the best team can (and should) always do better.
- Systemic Feedback Loops: Mechanisms that allow users, customers, and internal stakeholders to provide rapid and meaningful feedback.
- Data-Driven Decisions: Leveraging metrics like cycle time, lead time, escape defects, and customer satisfaction to understand where to improve.
- Empowerment for Change: Providing teams with the authority and resources to enact changes without excessive gatekeeping or external approval.

4. Leadership's Role

Finally, it's crucial to recognize how leadership shapes the environment that either supports or sabotages a framework's effectiveness. Leaders set the tone for:

- Prioritization: Ensuring that the organization stays focused on high-impact goals rather than chasing every new opportunity or crisis.
- Culture of Accountability: Balancing empowerment with clear accountability so that initiatives don't stall in the absence of decisive ownership.
- Resource Allocation: Investing adequately in the right tools, training, and staffing levels to enable success.
- Psychological Safety & Trust: Encouraging truth-telling and open communication, especially when the news isn't good.

Part 4: Practical Strategies to Build True Value Delivery

Now that we've established that frameworks are not the end goal, let's discuss practical strategies to build robust value delivery, both technically and organizationally. These strategies can integrate well with any framework, but they also stand on their own merit.

1. Define Value Through Your Customer's Lens

Before jumping into frameworks or processes, align on what "value" truly means to your customer. Value isn't what you assume—it's what your customer defines it to be.

What problem is the customer trying to solve with your product or service?

The key is to deeply understand their goals, challenges, and priorities. Once you have this clarity, articulate "value" in clear, measurable terms.

2. Adopt Incremental Delivery

Whether or not you embrace a formal Agile or Lean framework, the principle of delivering in small increments remains powerful. Incremental delivery ensures:

- Faster Feedback: Reduces the risk of building something no one wants or needs, because feedback loops are shorter.
- Improved Quality: Bugs and design flaws are easier to catch earlier.
- Adaptability: The organization can pivot quickly as new insights or market conditions emerge.

3. Build Cross-Functional Collaboration

Create teams that bring together diverse skill sets - engineering, design, QA, product management, marketing, etc. This cross-functional approach eliminates hand-offs between separate departments, which often become bottlenecks. When everyone needed to deliver an increment of value is on the same team, you foster collective accountability and shorten the feedback loops that slow down value creation.

4. Foster a Culture of Experimentation

Value delivery isn't just about implementing known features correctly; it's also about discovering what truly resonates with users and customers. A culture of experimentation can take many forms:

- A/B Testing: Try out different versions of a feature or design to see which performs better in real-world usage.
- User-Centric Metrics: Track user behavior to pinpoint where friction occurs and to measure the impact of new features.
- Hypothesis-Driven Development: Frame each new initiative as a hypothesis about how it will impact your goals, and define clear success metrics before you start building.

This iterative, data-informed approach aligns perfectly with many frameworks but can also stand alone as a core organizational practice.

5. Establish Clear and Meaningful Metrics

Frameworks often come with a set of prescribed metrics or ceremonies, but they may not always reflect what matters most to your unique business. Consider metrics that directly map to business value, such as:

- Cycle Time: How quickly can an idea go from concept to production release?
- Lead Time: From the moment a customer raises a request or opportunity is identified, how long until it's delivered?
- Production Incidents or Bug Rates: How stable is your product? How often do issues escape into production, and how severe are they?
- Product Adoption & Engagement: If you're working on digital products, is adoption increasing? Are users continuously engaged, or do they drop off?

Your metrics should form a balanced scorecard that reflects technical health, customer satisfaction, and business outcomes.

6. Flatten Organizational Structures Where Possible

Rigid hierarchies create delays and misunderstandings. While complete decentralization might not be feasible or beneficial in some contexts, aim for a structure that empowers decision-making at the team level. This can involve:

- Empowered Product Owners: In a product-centric organization, product owners or managers should have the autonomy and resources to shape product roadmaps based on stakeholder and user feedback.
- Elimination of Unnecessary Approval Layers: Trim down the sign-offs required for budgeting, small-scale experiments, or routine releases.
- Collaboration Over Command-and-Control: Encourage managers to be in a position to guide teams to solve their own problems rather than dictating solutions.

7. Scaling Wisely

As your organization grows, the complexity of delivering value can increase exponentially. Many frameworks have "scaled" versions, like SAFe for Agile, but these can quickly become bureaucratic if adopted as rigid templates. Instead, focus on principles:

- Autonomy: Maintain as much local decision-making as possible.
- Visibility: Ensure teams can easily share their learnings, metrics, and blockers. Use lightweight governance models—like communities of practice—to spread best practices rather than top-down mandates.
- Consistency vs. Flexibility: Standardize where it makes sense (e.g., on engineering best practices, code review protocols), but remain flexible in areas that benefit from experimentation and local adaptation.

Part 5: Frameworks as Enablers - Not Solutions

It's important to note that frameworks are not the enemy. They can be extremely useful when applied with the correct mindset and adapted to the organization's specific context. A well-implemented framework can provide:

- A Shared Language: Everyone talks about the same things in the same way, reducing miscommunication.
- Disciplined Cadence: Regular planning and review cycles ensure work doesn't drift and that stakeholders stay informed.
- Structured Introspection: Retrospectives can be incredibly powerful if people truly engage in self-reflection and commit to actionable improvements.

The key is to recognize that the framework is a tool—a framing device that provides a useful starting point for continuously refining your organization's processes. It does not, in and of itself, fix cultural, technical, or structural issues.

Part 6: Avoiding the Pitfalls of Framework-Centrism

Given the allure of frameworks, many organizations fall into common traps:

1. Silver Bullet Syndrome: Adopting a new framework to solve a deep-seated organizational problem is akin to putting a new coat of paint on a house with a crumbling foundation. It may look nicer for a short while, but the underlying structural issues remain and eventually resurface.

2. Framework Flip-Flopping: Some organizations jump from Scrum to Kanban to SAFe—and sometimes back again—each time hoping the new approach will finally crack the code to successful delivery. This pattern often indicates an avoidance of the real, underlying challenges, whether they be cultural resistance, outdated technology stacks, or dysfunctional organizational designs.

3. Cargo-Cult Implementation: Cargo culting involves mimicking the visible actions of successful companies—like holding daily stand-ups or calling something a "sprint"—without understanding the reasoning behind it. The result is a veneer of process without the substance that actually drives value.

4. Over-Measurement or Wrong-Measurement: Organizations can become obsessed with velocity charts, burn-downs, or other framework-specific metrics. While these can be useful within context, losing sight of customer-centric outcomes or business impact in favor of internal metrics can derail the ultimate purpose of delivering real-world value.

Part 7: Bringing It All Together

Building a healthy, sustainable value delivery ecosystem requires a multi-faceted approach—one that acknowledges the complex interplay of culture, technology, and organizational design. Here's a concise summary of how to bring these elements together:

- Start with Clarity of Purpose: Align every level of the organization on what "value" means to their customers and how you'll measure it.

- Optimize Organizational Structures: Design teams to be cross-functional, minimize bureaucracy, and ensure decision-making is as close to the work as possible.
- Empower People & Culture: Leadership must champion an environment of trust, continuous learning, and shared accountability.
- Leverage Frameworks as Tools: Use frameworks as tools for structure and discipline, but don't expect them to actually solve any problem right out of the box. Adapt, evolve, and tailor them to your unique context.
- Pursue Continuous Improvement: Measure what matters, retrospect often, and act on lessons learned. Make small, incremental changes that continuously move the needle on your most important metrics.

Frameworks can assist, but the true drivers of success—teamwork, strategic clarity, technical excellence, and learning cultures—must be cultivated from within.

Conclusion

Culture, technical proficiency, organizational design, leadership commitment, and continuous improvement lie at the heart of sustained success. Frameworks can be helpful guides, offering structure, vocabulary, and cadence. But ultimately, they only "frame" the real work that must be done.

So, the next time your organization considers rolling out another new framework, pause and ask the more critical questions: Are we aligned on what we're trying to achieve? Do we have the necessary technical excellence and organizational structure to truly support rapid, iterative delivery? Are our leaders modeling the behaviors required for a culture of continuous learning? And perhaps most importantly, are we measuring—and improving—the things that truly matter to our customers and stakeholders?

ENTERPRISE

MicroStream

OPEN-SOURCE

ECLIPSE FOUNDATION

ECLIPSE DATA GRID

ENTERPRISE

Next Generation Caching & In-Memory Searching.

1000x Faster than a Database.
Runs Between Your Application & Database.

- Caching
- In-Memory Searching
- Indexing
- Native Java Object Model
- Any Logic in Java
- ACID Persistence
- Distributed
- Open-Source
- SaaS

github.com/eclipse-datagrid

Eclipse Data Grid is an in-memory data processing layer to boost any sluggish database applications and relieve the database. Eclipse Data Grid is much more than a common cache. It enables ultra-fast in-memory data processing: caching, indexing, searching, and any complex data operations - up to 1000x faster than databases. Unlike traditional caching solutions, Eclipse Data Grid is a native Java layer using the native Java object model. This allows you to work with native Java objects, complex Java object graphs, and any Java types, and to integrate any Java libraries and implement any complex logic in your in-memory data layer with Java. Eclipse Data Grid is distributed, highly horizontally scalable and completely Open-Source. It's built for everyone who needs more than just a cache. Move your complex and performance-critical data and data operations to Eclipse Data Grid to dramatically reduce database workloads and save costs, boost your application and your business.

Contact Us:
www.microstream.one • hello@microstream.one
LinkedIn: linkedin.com/company/microstream

Book a Call:
www.microstream-intro.youcanbook.me

MicroStream

#JAVAPRO #PERFORMANCE

JVM Iceberg – Modern Performance Edition

Author:

Head of Java/Kotlin Engineering at VirtusLab, Artur Skowronski has been in the industry for ten years. During this time, he has had the opportunity to work in various roles, such as Software Engineer, Tech Lead, Architect, and even Technical Product Manager. This diverse experience enables him to approach problems from a holistic perspective.

He still loves to get his hands dirty - for more than two years, he has been publishing weekly reviews of events in the JVM world - https://jvm-weekly.com/

The „Iceberg" meme is an internet phenomenon that humorously, and sometimes unsettlingly, illustrates levels of knowledge or initiation into a given topic – from simple, widely known facts at the tip of the iceberg to the dark, esoteric depths comprehensible only to the most battle-hardened veterans. Picture an iceberg floating on water: what's visible on the surface is just the beginning, while the real magic (or nightmare) lurks beneath, in increasingly inaccessible layers.

Personally, I love it. So I decided to create Java ones. I've already published one covering JVM as a whole, but this time I decided to focus on a particular topic - performance! I hope you will like it!

Level 1: The Tip

Project Loom (Virtual Threads)

TLDR: Project Loom is Project Loom - you know it all, you love it all

Project Loom is a new concurrency model in Java (introduced in JDK 19/21) offering lightweight virtual threads managed by the JVM. Virtual threads are significantly cheaper than traditional system threads – thousands can be launched with minimal overhead, as they consume minimal resources and are suspended/resumed by the runtime (e.g., during I/O) instead of blocking system threads; this improves scalability and throughput in highly parallel applications while maintaining the simple, thread-based programming model.

JDK Flight Recorder (JFR)

TLDR: JVM profiling with minimal overhead. A must-have for diagnosing bottlenecks. Like a black box for your Java app (minus the crash)

JDK Flight Recorder (JFR) is the JVM's built-in profiler and event recorder—think of it as a super lightweight flight data recorder for your Java application. It tracks all the good stuff: CPU usage, memory allocations, thread activity—you name it—with minimal performance overhead. Seriously, it's so efficient you can (and should) run it in production.

It's been baked into the JDK since version 11, and you can kick it off at startup or even attach it mid-flight. Perfect for continuous monitoring without dragging down your app.

And the best part? JFR helps you catch bottlenecks, memory leaks, and other sneaky performance gremlins before they start tanking your uptime. Quietly powerful, just like we like our tooling.

Java Mission Control (JMC)

TLDR: Graphical analysis of JFR data. Perfect for your morning coffee while tracking memory leaks.

Java Mission Control (JMC) is your go-to visual toolkit for making sense of what Java Flight Recorder (JFR) captures while your app's doing its

thing. CPU spikes? Thread pileups? Suspicious memory allocations? JMC puts it all on a pretty timeline so you don't have to piece it together from log spaghetti.

It's fast, it's surprisingly user-friendly, and yes — even if you're not a JVM tuning ninja, you can spot memory leaks or GC overloads without needing a PhD in diagnostic tooling. Just fire it up and start connecting the dots.

Because honestly, if you're already collecting JFR data, not using JMC is like owning a telescope and only using it to look at clouds.

Foreign Function & Memory API (Panama)

TLDR: JNI without JNI. Java gets closer to C performance without losing comfort.

The Foreign Function & Memory API—aka Project Panama—is Java's shiny new way of calling native code and poking around in off-heap memory without diving into the dark depths of JNI. It's all about letting you talk to C (or C++) libraries directly, but with clean, modern Java APIs that won't make your eyes bleed.

No more boilerplate glue code or wrestling with native headers. Panama makes it way easier (and faster) to integrate with performance-critical native libs—think things like image processing, numerical computing, or custom hardware interfaces.

For anyone building apps that need to crunch serious data or do things the JVM wasn't exactly born for, Panama opens up some exciting new doors—and it holds the door open for you too.

GraalVM

TLDR: Turbo for JVM with a JIT compiler written in Java. Aggressive optimization, plus polyglot runtime support.

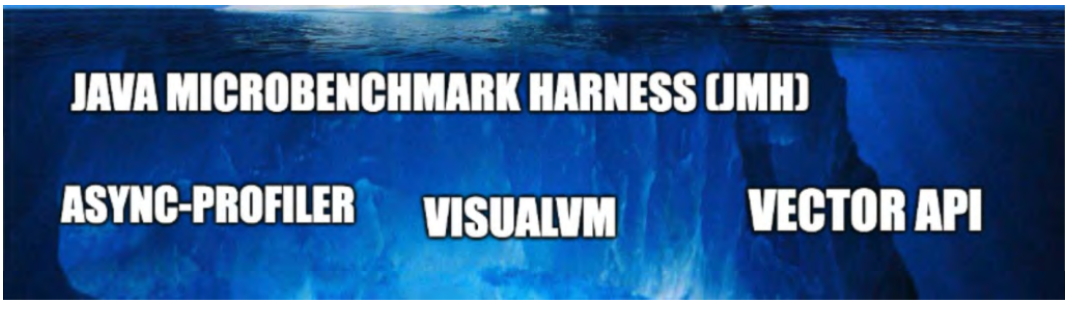
GraalVM is like HotSpot's cooler, more experimental cousin. It's built on OpenJDK but swaps in the Graal compiler—a next-gen JIT written in Java

itself—for smarter, leaner, and more aggressive optimizations. Think: fewer CPU cycles wasted, snappier performance, and an all-around tighter runtime.

But wait, there's more—it's not just about squeezing more out of Java. GraalVM is also a full-blown polyglot platform, meaning you can run Java, JavaScript, Python, Ruby, R, and more in a single VM. Yes, really.

In practice? It lets you build flexible apps where components written in different languages play nice together in one runtime. Faster builds, cleaner code, and the freedom to pick the right tool for the job—all without leaving the GraalVM playground.

Level 2: Just Below the Surface



Vector API

TLDR: SIMD in pure Java. Numerical computing and multimedia in JVM have never been faster

The Vector API—introduced via JEP 338 and still cooking nicely in preview—is Java's ticket to the SIMD party. Instead of processing data one sad element at a time, this API lets you crunch multiple values in parallel using Single Instruction, Multiple Data magic. Think IntVector, FloatVector, DoubleVector—entire arrays of data, sliced and diced in a single CPU operation.

This isn't just cool for the sake of it—it's a big win for number-heavy tasks like image and signal processing, scientific computing, or anything that screams "data crunch me harder." Add, multiply, compare—all way faster than your regular loops.

Even better? It's built with portability in mind. The API figures out what SIMD instructions your CPU supports and uses them behind the scenes,

so your code stays clean and runs fast on multiple architectures. Write once, vectorize everywhere.

VisualVM

TLDR: Classic monitoring tool. Intuitive UI, effective performance, and evergreen status among developers.

VisualVM has been around forever—and there's a reason it's still in toolbox of many devs. This classic GUI-based tool lets you peek inside your running Java apps in real time: CPU and memory usage, thread activity, object allocations, heap dumps, stack traces—you name it.

Sure, it might not have the shiny new branding of some newer tools, but when you need to track down a rogue memory leak or figure out why your app's suddenly cooking CPUs like breakfast, VisualVM gets the job done. Fast.

It's lightweight, easy to use, and perfect for both local dev and staging environments. Sometimes, you don't need fancy—you just need something that works. VisualVM is exactly that.

Java Microbenchmark Harness (JMH)

TLDR: The standard benchmarking tool. Helps avoid pitfalls with JVM and JIT that can skew results.

JMH (Java Microbenchmark Harness) is the official go-to tool from the OpenJDK crew when you want to actually measure how fast your Java code runs. It's built specifically for benchmarking on the JVM, which means it handles all the tricky stuff—JIT warm-ups, dead-code elimination, and those sneaky measurement traps that make your stopwatch lies look believable. In short: if you want numbers you can trust, you use JMH.

Whether you're optimizing a hot loop or validating that a refactor didn't nuke performance, JMH lets you write clean, focused microbenchmarks that produce reliable data, not vibes.

Async-Profiler

TLDR: Native sampling profiler with flame graphs. A master of low overhead and accurate diagnostics.

Forget the old-school safepoint-skewed profilers — async-profiler is a modern, native-level (yep, written in C++) sampling profiler that knows how to keep it real. It skips the safepoint bias that messes with your stack traces and gives you an honest look at what your app's actually doing.

It covers CPU, memory allocations, I/O, and even lock contention, with crazy low overhead, which means you can run it in production without sweating bullets. And when you're done? Boom—flame graphs. One glance and you'll know exactly where the bottlenecks are hiding.

If you're serious about performance tuning on the JVM, async-profiler is basically your X-ray vision.

Level 3: Deeper Level

YourKit and JProfiler

TLDR: Commercial profiling powerhouses, convenient for both local and production environments. Comfort for the price of a license? Why not!

If async-profiler is your lean command-line ninja, YourKit and JProfiler are the luxury sedans of the Java profiling world—fully loaded, smooth UI, and packed with features. Both are commercial tools, but they earn their keep with powerful diagnostics: deep CPU and memory profiling, heap analysis, leak detection, and top-notch thread monitoring.

They shine especially in big, complex projects where you need that extra level of insight, and can afford the tooling. Bonus points for IDE integrations, support for remote sessions, and features tailored for

different stages of development and ops.

Not everyone needs them, but when you do, they're rock solid.

Ahead-of-Time Compilation (AOT)

TLDR: Fast start at the expense of peak performance. Ideal for microservices and serverless scenarios.

Ahead-of-Time (AOT) compilation lets you skip the whole JIT warm-up ritual and go straight to a native binary. Instead of compiling code during runtime, you compile it before—resulting in lightning-fast startup and reduced memory usage. The trade-off? You might lose a bit of peak performance, but for many workloads, that's totally worth it.

The poster child here is GraalVM Native Image. Apps built this way can launch in milliseconds, which makes it a perfect fit for microservices, serverless functions, or anything where "cold start" sounds like a curse word.

But! That's not the only game in town — Project Leyden is Oracle's long-term plan to bring similar startup + footprint improvements within the OpenJDK, aiming for more standardized and JVM-integrated solutions without needing a separate toolchain.

Z Garbage Collector (ZGC)

TLDR: The garbage collector of the future? Pauses under 10 ms, maximum concurrency, and since JDK 21, even with generations.

ZGC (Z Garbage Collector) is HotSpot's answer to the age-old question: „Can I GC without killing my app's vibe?" Spoiler: yes. Designed for massive heaps and ultra-low pause times (we're talking ~10 ms or less), ZGC does most of its work concurrently — so your app keeps humming while GC quietly tidies up in the background.

And starting with JDK 21, ZGC got even better with generational support. That means it now reclaims memory more efficiently without sacrificing its signature low-latency magic.

In real life? ZGC is what you reach for when you're building data-heavy, user-hungry systems that need to stay snappy under pressure. Whether you're scaling up or handling real-time workloads, ZGC helps keep things smooth, fast, and garbage-free(ish).

AWS Lambda SnapStart

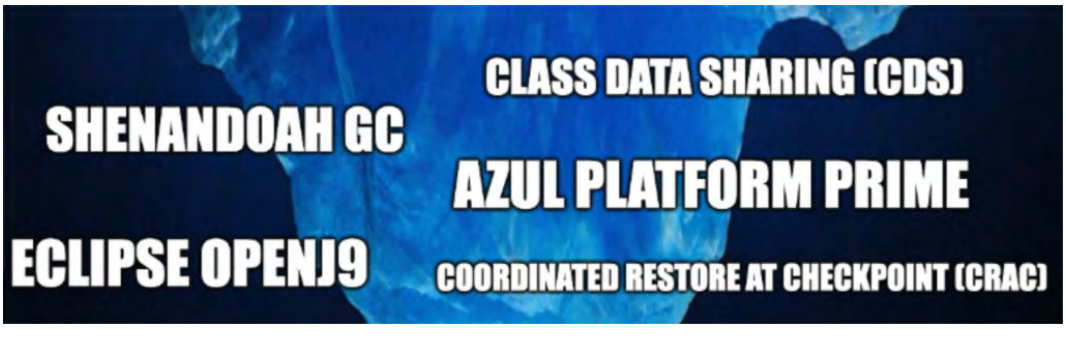
TLDR: Cold start up to 10 times faster with no additional costs. The magic of AWS!

AWS Lambda SnapStart is Amazon's magic trick for making Java functions start way faster. Instead of going through the whole cold start dance every time, SnapStart takes a snapshot of your function after it's fully warmed up—class loading, dependency wiring, the whole shebang.

Once you publish a new version, Lambda does that initialization once, saves the memory state, and next time your function runs… boom it just resumes from that snapshot. No repeated setup, no waiting around. You get up to 10x faster startup, with no extra cost.

In practice? SnapStart makes Java way more viable in serverless scenarios, where every millisecond counts. Suddenly, Java's not the "slow starter" in the room—it's sprinting off the blocks with the best of them.

Level 4: Even deeper



Shenandoah GC

TLDR: Red Hat made a GC for fans of ultra-short pauses. Worth trying on large heaps.

Designed by the fine folks at Red Hat, Shenandoah GC is all about keeping pause times tiny—even when your heap isn't. It does most of its garbage collection work concurrently with the application, which means

your 2GB dev box and your 200GB production beast get the same short pauses. No kidding.

The secret sauce? Concurrent memory compaction and region-based collection that keeps latency low and throughput solid.

In real-world terms: Shenandoah shines in systems where responsiveness is non-negotiable—financial platforms, real-time dashboards, anything that panics when the GC hiccups. It's one of those tools that just works quietly in the background… and your users never know how close they came to a full-GC freeze.

Class Data Sharing (CDS)

TLDR: Faster start, lower memory usage, ideal for microservices in containers.

Class Data Sharing (CDS) is one of those JVM features that quietly pulls serious weight. Instead of reloading and recompiling the same classes every time your app starts, CDS lets you create a shared class archive—basically a pre-baked bundle of commonly used classes.

The JVM can then load these directly from disk into shared memory, skipping all the warm-up overhead. Result? Faster startup and lower memory usage—especially handy in containerized or microservices setups where you're spinning up JVMs like there's no tomorrow.

In practice, CDS helps your apps boot quicker, run leaner, and scale better. Less RAM, less CPU, and less time wasted on things you've already loaded a hundred times.

Eclipse OpenJ9

TLDR: JVM focused on fast start and minimal memory footprint. Cloud-native JVM at its best!

Eclipse OpenJ9 is an alternative JVM implementation, originating from IBM J9, designed with a focus on fast startup and small memory footprint. OpenJ9 offers unique features such as shared class memory between

runs and JIT Server mode, which offloads the JIT compilation cost from the application process.

In practice, OpenJ9 excels in cloud environments where resources are limited, and fast application startup is critical. With lower memory usage and quicker startup, OpenJ9 allows for more efficient resource usage and better scalability of applications.

Coordinated Restore at Checkpoint (CRaC)

TLDR: Checkpoint and restore JVM. Applications start instantly with full warm-up.

Coordinated Restore at Checkpoint (CRaC) is one of the coolest (pun fully intended) things happening in OpenJDK right now. It introduces an API for creating snapshots of a running Java application—including its JIT-warmed code, populated caches, and full runtime state—and then restoring from that snapshot like nothing ever happened.

The result? Your app wakes up from its cryo-nap with zero warm-up and hits full performance instantly. No cold starts, no ramp-up, just go. A bit like reading Save States in the emulator.

In practice, CRaC is a game-changer for use cases where startup time and latency matter—like autoscaling microservices, serverless workloads, or anything that doesn't have time to wait for the JVM to "get ready." Java, but instant on.

Azul Platform Prime

TLDR: JVM on steroids with pause-free GC C4 and Falcon JIT based on LLVM. Real-time and transactions? Absolutely.

Azul Platform Prime is like the high-performance luxury edition of the JVM. Under the hood, it packs the C4 garbage collector (pause-free, even under pressure), the Falcon JIT compiler (LLVM-based and laser-focused on peak performance), and ReadyNow!, which nukes warm-up time so your app can hit full speed right out of the gate.

In practice, Azul Prime is built for the kind of workloads where latency kills and throughput pays — think financial systems, trading platforms, or real-time apps that can't afford a hiccup. It's not your everyday JVM — but if you need max performance with zero compromise, this one earns its "Prime" label.

Level 5: The Bottom

Falcon JIT

TLDR: LLVM meets JVM. A compiler with sharp optimization edge.

We already gave Falcon JIT a nod earlier, but let's be honest—it deserves its own spotlight. Developed by Azul, Falcon is a just-in-time compiler that swaps out the classic HotSpot C2 for something far spicier: LLVM. That's right—the same backend used to power compilers in C, Rust, Swift, and more is now optimizing your Java code.

The result? Lean, mean, machine code that's more aggressively optimized than what C2 typically offers. In real-world terms: better throughput, faster execution, and lower resource usage—all without changing your application code.

If your workload is CPU-hungry, latency-sensitive, or just plain performance-obsessed, Falcon is like strapping a turbocharger to your JIT.

OpenLiberty InstantOn

TLDR: Running Java containers in milliseconds. IBM dusted off CRIU, and Java got a turbo boost.

Open Liberty InstantOn, brought to you by IBM, is all about skipping the slow boot sequence and getting straight to business. It uses CRIU

(Checkpoint/Restore In Userspace) to take a snapshot of your Java app's fully-initialized state—then brings it back to life almost instantly.

InstantOn is a big win in cloud-native and serverless environments where startup time can make or break scalability. Apps scale faster, respond sooner, and your platform stops feeling like it's running in molasses.

Thread-Local Allocation Buffers (TLAB)

TLDR: Faster object allocation in multithreaded applications. Each thread gets its own piece of Eden!

Thread-Local Allocation Buffers (TLAB) is a mechanism in the JVM that allocates each thread its own memory area in the young generation of the heap (Eden). This allows threads to allocate objects without needing synchronization, speeding up the allocation process.

TLAB increases the performance of multithreaded applications by reducing synchronization costs during object allocation. As a result, applications can scale better on multi-core systems.

Epsilon GC

TLDR: A GC that doesn't collect garbage. Sounds strange? Perfect for performance testing!

Epsilon GC is the "do nothing" garbage collector in OpenJDK—literally. It doesn't reclaim memory. At all. It just allocates until the heap is full and then… well, crashes.

Why would anyone want that? Because it's perfect for performance testing. With Epsilon, there's zero GC overhead, so you can benchmark your app without any collector skewing the results. It's also handy for short-lived processes where memory management is irrelevant.

It's not meant for production (unless you really like living on the edge), but for benchmarking, tuning, or chaos experiments — Epsilon keeps things simple, brutal… and honest.

Arthas

TLDR: Arthas – live JVM diagnostics. A must-have in production when debugging without modifying code.

Arthas is an open-source diagnostic tool from Alibaba's middleware team that lets you troubleshoot live Java applications—no code changes, no restarts, no redeploys. By enhancing bytecode on the fly, it gives you powerful tools for real-time monitoring: think class decompilation, method tracing, thread analysis, and resource inspection—all from a slick interactive CLI.

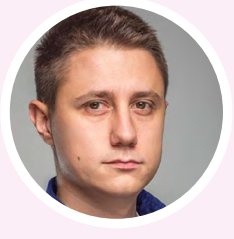
In practice, Arthas is a go-to for DevOps teams and developers working on high-availability systems. It helps you catch memory leaks, deadlocks, or rogue SQL calls before they snowball—without taking your app offline.

Just like his namesake in Warcraft III, Arthas isn't afraid to step into the battlefield mid-fight and take control (sorry, I'm nerd, I couldn't resist this one).

#JAVAPRO #ARCHITECTURE #MICROSERVICES

Dynamic consistency boundaries

Author:



Milan Savic is Software Engineer on a mission to bridge the gap between (usually unnecessarily) complex theories in software and reality. Tools he uses to achieve this goal involve writing code and blogs and speaking at conferences. He is an independent contractor working for AxonIQ where he implements Axon Framework and Server.

In software, Consistency may refer to many things - Code Consistency, Data Consistency, User Interface Consistency, etc. In this article, I am focusing on Data Consistency. Simply put, Data Consistency ensures that data remains accurate and reliable across different parts of a system. More formally, Data Consistency is one of the Database Transaction properties (ACID1) defined as:

> ***„Consistency (or correctness) ensures that a transaction can only bring the database from one consistent state to another, preserving database invariants."***
>
> *- Wikipedia*

The invariants may be enforced in various ways depending on the database system. For example, in Relational Databases, there are many ways to define consistency: constraints (primary keys, foreign

keys, uniqueness, etc.), ACID properties of the transaction (Atomicity, Consistency, Isolation, Durability), referential integrity, and many more. The common consensus (so far) of achieving consistency in Event Stores is that event streams define the atom of consistency. Consistency across event streams is not possible. Inside the event stream, which has append-only characteristics, the consistency is ensured by preventing multiple events from being appended at the same position (index). Event streams are commonly associated with a concept borrowed from the DDD (Domain Driven Design) - an **Aggregate.**

> ***„An aggregate is a cluster of associated objects that we treat as a unit for the purpose of data changes."***
>
> *- Domain-Driven Design: Tackling Complexity in the Heart of Software by Eric Evans*

Dynamic Consistency Boundaries (DCB) redefine the granularity level of consistency for Event Stores, moving from event streams (aggregates) to individual events. You could still say that DCB deals with dynamically defined event streams. The original application of DCB was for Event Stores; however, it might be broadened to any messaging system with an append-only nature.

Messaging

A broad definition of messaging would be two or more participants in communication exchanging information via various means. For us, the interesting type of messaging would be a (semi)durable append-only log with pub/sub characteristics. In other words, multiple producers produce messages to the log, and self-paced consumers read from the log.

Each message in the log has its position (a.k.a. index), uniquely identifying it. This property helps consumers with the deduplication process. The consumer takes the message with its index from the log, processes it, and remembers the index. This process can be restarted at any time, and upon restarting, it will resume reading from the log at the remembered index. Also, suppose the message gets re-delivered to the consumer for various reasons (usually associated with the Distributed Systems fallacies). In that case, the consumer can easily deduce whether it has already processed the message (if the index of the received message is less than or equal to the last remembered index, the consumer recognizes that the message

has already been processed).

Message Streams

A large message log may be divided into multiple message streams for various reasons, such as domain semantics, security (to prevent mixing messages from different domains in a single stream), scalability, etc.

Usually, the message stream is backed up by a physical concept, such as an array of files storing messages. The consistency across physical streams is not guaranteed - the client cannot write to two message streams atomically (what if writing to one stream fails? what if a rollback fails? etc.). The consistency guarantee of this type of messaging system ensures that a new message can be appended to the stream only if its index follows sequentially after the last one - without overwriting or gaps.

Once we decide which (types of) messages belong to which stream, changing this ownership comes with a price. While splitting one stream into multiple smaller ones is not a complex operation (the order of messages from the previous stream is preserved in the following streams), the merging operation is not so trivial. When merging multiple streams, we must decide on the order of messages. We could use timestamps of the messages to determine the order; however, in distributed messaging systems, this is impossible (since we cannot rely on multiple servers being aligned time-wise). Usually, a manual intervention in cooperation with the business is required. This makes message streams somewhat rigid when it comes to refactoring them.

Event Sourcing

Event sourcing, often misunderstood, is „just" a way of persistence. Instead of persisting the current state of our system, we persist it as a series of ordered events. When we need to figure out the system's current state, in order to make a decision, we „replay" historical events. Although this sounds trivial and not a big deal, employing it correctly requires quite a mind shift.

Event Stores are the most suitable database technology for durably storing events. Essentially, an Event Store is a Messaging System with

an append-only log and pub/sub (a.k.a append/read) characteristics, but with a twist. The twist is that it stores messages (events, in this case) forever. Everything we talked about Messaging Systems and Message Streams applies here as well.

Some upfront design process is required before we start implementing an event-sourced system. This process can be Event Modeling3, Event Storming4, or something else found useful to identify which events are essential for the system. Usually, as one of the steps in the design process, we determine consistency boundaries. We typically try to fit events in aggregates as our consistency boundary. Translating this to Event Store terms means mapping aggregates to event streams in one-to-one relationship. But wait. We said that streams are not convenient when changing the ownership of events. This obstacle implies that once we establish consistency boundaries (aggregates), modifying them becomes difficult after the system is deployed to production. Issues don't stop there. Sometimes, there are business rules that cannot fit inside a single stream. This means that to maintain consistency across streams, we must rely on alternative techniques instead of aggregates, which provide eventual consistency rather than immediate consistency (process managers are one example). Also, specific use cases require that a single event belong to multiple streams. An event may affect multiple decisions spread across numerous streams (aggregates). By definition of an event stream, this is impossible - a single event belongs to one stream only!

Does this mean streams are a poor choice for an Event Store? Not necessarily. The issue lies in the granularity of consistency, which isn't well-adjusted. Mapping aggregates to streams lacks flexibility. But what if we rethink the concept of a stream? Rather than aligning with an aggregate, a stream is better suited to a broader scope. Revisiting Eric Evans' book on DDD (Domain-Driven Design: Tackling Complexity in the Heart of Software), the Bounded Context seems like a more fitting analogy. Within a Bounded Context, we store events related to a specific sub-domain. The challenge then becomes maintaining consistency boundaries within the Bounded Context. This is where DCB comes into play, adjusting the granularity level.

Dynamic Consistency Boundaries

DCB provides a way of guaranteeing consistency during an append to the Event Store. The Event Store client reads (only) the necessary events to rehydrate the state and make a decision. The client gets events and a consistency marker as a result of reading. This consistency marker is used to maintain consistency in the Event Store. When the client decides to append new events, it'll form a transaction consisting of events to be appended, the criteria used to filter events during read, and the consistency marker. The Event Store uses the criteria to check whether the consistency marker has changed in the meantime (after reading and before appending). If the marker hasn't changed, the transaction can be successfully appended; otherwise, it will be rejected. In relational databases, this mechanism is called optimistic locking.

Although simple, the DCB mechanism might be too much to grasp without an example. Let's build a simple in-memory Event Store that supports DCB and use a simple domain to exemplify it.

Before diving into the Event Store details, let's get familiar with the terminology.

term	definition
global sequence	Each event in the Event Store is associated with the global sequence number which determines its position in the globally ordered Event Store log. The global sequence of the first event is 0.
head	The global sequence of the next event to be appended to the Event Store.
tag	Specifies the event in more detail. It is just a key-value pair, e.g. tag{key="email", value="student@university.com"}. Event Store must store tags together with events and provide a search based on them. Usually, an Event Store indexes events based on tags for faster retrieval.
criterion	Building part of the criteria. It is composed of tags. Between them, an AND operator is applied - for an event to meet the criterion, all its tags must match the criterion tags.
criteria	Filters out events from the Event Store. It is composed of criterions. An OR operator is applied between them - for an event to meet the criteria, at least one criterion should be satisfied.

In our simplistic version of the Event Store API, we support only two operations:

1. *read* - reads a finite stream of events from the Event Store based on provided criteria.
2. *append* - appends events at the end of the Event Store log. It accepts the consistency condition as the parameter used to check the consistency of this append.

```
public interface EventStore {

    MarkedEvents read(Criteria criteria);

    void append(List<Event> events, ConsistencyCondition
condition);
}

public record MarkedEvents(long consistencyMarker,
                           Stream<SequencedEvent> events)
{ }
```

Since our implementation is entirely in memory, we will use a SortedMap to store our events. The key of this map is the global sequence of the event and the value is the event with its corresponding tags. We want to support concurrent access to the Event Store, hence, we will use the ConcurrentSkipListMap implementation. However, dealing with concurrency in this article will make the code unnecessarily polluted; hence, I'll skip it to show the essence.

```
SortedMap<Long, Event> events = new
ConcurrentSkipListMap<>();
```

read operation provides MarkedEvents – all events matching the given criteria. These events are marked with Event Store's consistency marker. Here, the Event Store's head is used as a consistency marker. Later, while appending events, the consistency marker is used to skip the events the client uses to make the decision. This little trick will help us narrow the search space to find conflicts.

```
public MarkedEvents read(Criteria criteria) {
    var consistencyMarker = head();
```

```
    var sourced = events.values()
                        .stream()
                        .filter(event -> criteria.
matches(event.tags()));
    return new MarkedEvents(consistencyMarker, sourced);
}
```

During the append each event is described in more detail with tags that associate this event with specific concepts from the Domain. append accepts a transaction (list of events) to be appended and the ConsistencyCondition denoting consistency requirements for the append. The ConsistencyCondition is composed of a consistency marker and criteria.

The consistency marker tells the Event Store to search for events that match the given criteria after its position. If no events match the criteria after the consistency marker, the consistency condition is fulfilled, and the transaction is accepted; otherwise, it's not.

```
public void append(List<Event> events,
                   ConsistencyCondition condition) {
    if (!validate(condition)) {
        throw new InvalidConsistencyConditionException();
    }

    events.forEach(e -> this.events.put(head(), e));
}

private boolean validate(ConsistencyCondition condition) {
    return events.tailMap(condition.consistencyMarker())
                 .values()
                 .stream()
                 .noneMatch(e -> condition.matches(e.
tags()));
}
```

The Client

The architecture of the client application is beyond the scope of this article. Therefore, we will introduce only minimal abstractions to

illustrate how DCB functions in practice. Our client application consists of multiple request handlers, each responsible for constructing criteria based on the incoming request. These handlers derive their state from a sequence of events retrieved from the Event Store according to the defined criteria and then process the request. Upon handling a request, the handler generates a list of events to be appended to the Event Store. The interface defining a request handler is shown below.

```
public interface RequestHandler<R, S> {

    Criteria criteria(R request);

    S initialState();

    S source(Object event, S state);

    List<Event> handle(R request, S state);
}
```

The component responsible for dispatching requests to the appropriate handler is known as the Request Dispatcher. Its role is to identify the most suitable handler for a given request and forward it accordingly. Before doing so, it must construct the handler's state based on relevant events. To retrieve these events, the dispatcher first obtains the criteria from the handler based on the request. It then uses this criteria to fetch events from the Event Store. As a result, the dispatcher acquires both the events and the consistency marker, which will later be used for appending.

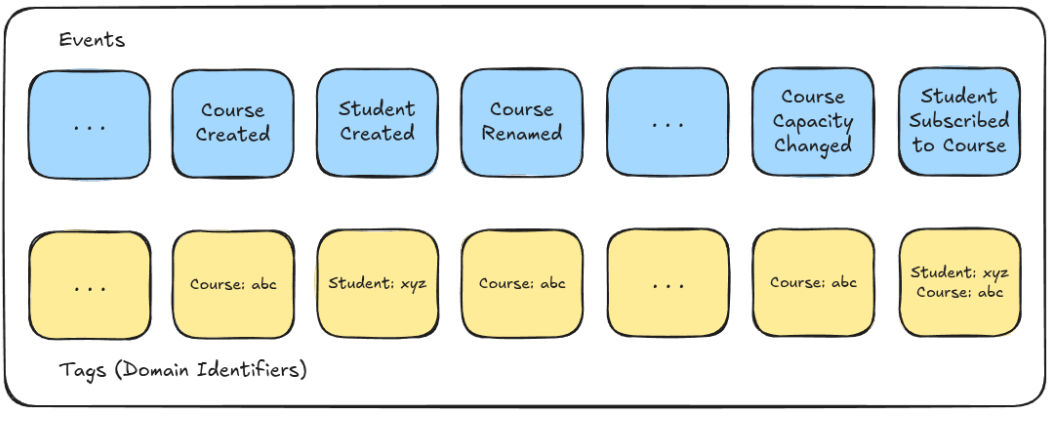
```
public void dispatch(Object request) {
    var handler = findHandler(request);
    var state = handler.initialState();
    var criteria = handler.criteria(request);
    var result = eventStore.read(criteria);
    var consistencyMarker = result.consistencyMarker();
    var condition = consistencyCondition(consistencyMarker,
criteria);
    result.events()
          .reduce(state, (current, event) ->
                  handler.source(event.payload(),
current))
          .map(sourcedModel ->
```

```
                handler.handle(request, sourcedModel))
            .flatMap(events ->
                    eventStore.append(events, condition));
}
```

We derive the handler's state from the events fetched from the Event Store. Once the state is entirely sourced, it is safe to pass the request to the handler. The handler will return a list of events to be appended to the Event Store. The dispatcher creates a consistency condition using the consistency marker obtained from the Event Store and the same criteria used to read the events. The dispatcher appends the events to the Event Store with this consistency condition. Meanwhile, new events that match our criteria may have been appended to the Event Store. If this happens, our transaction will be rejected; otherwise, it will succeed.

How to Use Criteria for Filtering?

Using an example is the best way to understand how criteria-based querying works. In this case, we have a Student that can subscribe to a Course. There are events for student and course creation, renaming the course, course capacity change, and an event that a student subscribes to the course. The event stream is depicted in the image below.



All these events are tagged with specific Domain Identifiers. Certain events, like a student subscribed to a course, are tagged with two tags - student and the course. This is because the event belongs to the student and course Domain concepts.

To handle the request to subscribe the student to the course, we need to source our model based on the events we are interested in. Those

events depict whether the course capacity has changed and whether the student has subscribed to a course. Let's see how to form a criteria for this use-case:

```
anyOf(allOf(tag(„eventType“, „CourseCapacityChanged“),
            tag(„courseId“, „abc“)),
      allOf(tag(„eventType“, „StudentSubscribedToCourse“),
            tag(„courseId“, „abc“)),
      allOf(tag(„eventType“, „StudentSubscribedToCourse“),
            tag(„studentId“, „xyz“)))
```

Another interesting scenario would be to check whether the student subscribed to a course.

```
anyOf(allOf(tag(„eventType“, „StudentSubscribedToCourse“),
            tag(„studentId“, „xyz“),
            tag(„courseId“, „abc“)))
```

Conclusion

DCB introduces a different view on the consistency in event-sourced systems. Further, DCB can be applied to any Messaging System with append-only log and pub/sub nature. Let's sum up how DCB changes the current state of event-sourced systems.

- Reduces the number of events needed to rehydrate the system's current state by finely filtering events based on the criteria.
- Removes the necessity of other techniques to provide consistency in the single event stream. Now, we can dynamically define the boundary of consistency by pulling the events we need to make the decision.
- Reduces the append contention. Finely defined criteria for request handlers have less chance of conflicting since they source only the necessary events.
- If applied without caution, the consistency boundary of particular request handlers might be too large, causing frequent conflicts. This concern is not new to event-sourced systems; it also exists with aggregates. However, with DCB, refactoring consistency boundaries is much more flexible since our event stream stays intact.

- Removes the burden of correctly completing the initial design since refactoring the consistency boundaries is not as tricky as with aggregates.

All code samples are backed by the GitHub repository: https://github.com/m1l4n54v1c/event-store. Note that the code presented in this article is simplified to show the essence of the DCB. Also, the code in the repository understands CQRS and deals with Command Handlers, not Request Handlers. Introducing CQRS concepts here is definitely out of the scope and doesn't add to the understanding of the DCB.

1. In computer science, ACID (atomicity, consistency, isolation, durability) is a set of properties of database transactions intended to guarantee data validity despite errors, power failures, and other mishaps. (wikipedia.org) ↩️
2. An Event Store is a type of database designed specifically for storing events in an append-only fashion. ↩️
3. Event Modeling is a method of describing systems using an example of how information has changed within them over time. (eventmodeling.org) ↩️
4. EventStorming is a flexible workshop format for collaborative exploration of complex business domains. (eventstorming.com)

#JAVAPRO #OPENSOURCE

Java at Eclipse: Honoring the Legacy, Securing the Future of Open Source Innovation

Author:

As the Community Manager for Eclipse Adoptium, Carmen Delgado brings experience in project, operations, and financial management across various industries to help Adoptium working group members achieve their goals and objectives. Her background includes successful terms in healthcare, pharma, fintech, and tech startups. Additionally, She actively contributes to Step4ward, a mentoring program in Spain, demonstrating her commitment to fostering diversity and inclusion in the tech world.

Co Authors

Tanja Obradovic
Senior Manager Java Programs

Thomas Froment
Program Manager

Java changed software development and reshaped open source innovation forever, but its future was uncertain for a while. Would it remain an innovative force, or would closed governance and slow progress relegate it to history? Enter the Eclipse Foundation, **where the future of open source Java was reimagined.**

Java's journey famously began in the mid-1990s, revolutionising software development with its "write once, run anywhere" promise. Originally,

Java, the language, was called „Oak," a name chosen by its creator, James Gosling, after an oak tree outside his office. Oak trees are known for their deep roots and longevity, much like Java, which has remained a cornerstone of software development for decades. While the name was later changed due to trademark issues, its original symbolism reflects Java's foundational role in modern computing: a language built to be powerful, adaptable, and enduring. Open-sourced by Oracle in 2007, it has become the backbone of enterprise applications, cloud computing, and mobile development.

Hard Times for Java

However, as the technological landscape evolved, so too did the challenges faced by developers and enterprises. What once served as a robust and reliable foundation for enterprise applications began to show signs of strain under the weight of new demands. Among the most significant shifts was the rise of cloud-centric development, which transformed how applications were built, deployed, and maintained from the early 2000s onwards. Traditional enterprise Java frameworks, particularly Java EE, struggled to keep pace with these changes.

One of the core issues hindering Java EE's progress was its governance model. The framework was largely controlled by a single entity, limiting external contributions and slowing the pace of innovation, frustrating developers and organisations that relied on it for mission-critical applications. This stagnation created an urgent need for a more dynamic, adaptable approach to enterprise Java development.

At the same time, organisations required stable, well-tested, and freely available Java runtimes. While the OpenJDK ecosystem thrived, providing a strong foundation for Java development, it lacked a coordinated effort to deliver trusted, vendor-neutral binaries. Organisations sought reliability and consistency in their Java environments, but the absence of a unified approach made it difficult to standardise implementations across different infrastructures. This gap in the ecosystem underscored the necessity for a collaborative, community-driven solution that could provide both innovation and stability.

JakartaEE: A New Beginning for Enterprise Java

In 2017, Oracle contributed **Java EE to the Eclipse Foundation**, where it was rebranded as Jakarta EE, marking a turning point. This transition ensured enterprise Java was open source and developed **collaboratively and vendor-neutrally.**

The **Jakarta EE Working Group** was formed to bring together industry leaders to define a **modern, cloud-native future for enterprise Java.** Through ongoing updates and community-driven development, Jakarta EE reinforced Java's position in enterprise software.

The first major milestone in this journey was **Jakarta EE 8**, which was released under the governance of the Eclipse Foundation. This version provided full compatibility with **Java EE 8**, guaranteeing a seamless transition for enterprises while laying the groundwork for future innovation.

Next came **Jakarta EE 9**, a significant step in modernising the platform. This release focused on **simplifying and streamlining** the platform by removing outdated technologies and transitioning to the jakarta.* namespace. These changes established a more flexible foundation for future enhancements while making Jakarta EE more adaptable to evolving industry needs.

With **Jakarta EE 10 and beyond,** the platform fully embraces **cloud native architectures, microservices, and rapid innovation cycles,** ensuring enterprise Java remains competitive in a modern development landscape. These advancements enable developers to build more scalable, lightweight, and resilient applications tailored to contemporary cloud environments.

Once constrained by slow, proprietary governance, enterprise Java found new life under Eclipse, evolving into a cloud native powerhouse.

A Thriving Family of Java Technologies and Tools

Eclipse Temurin as Flagship Runtime

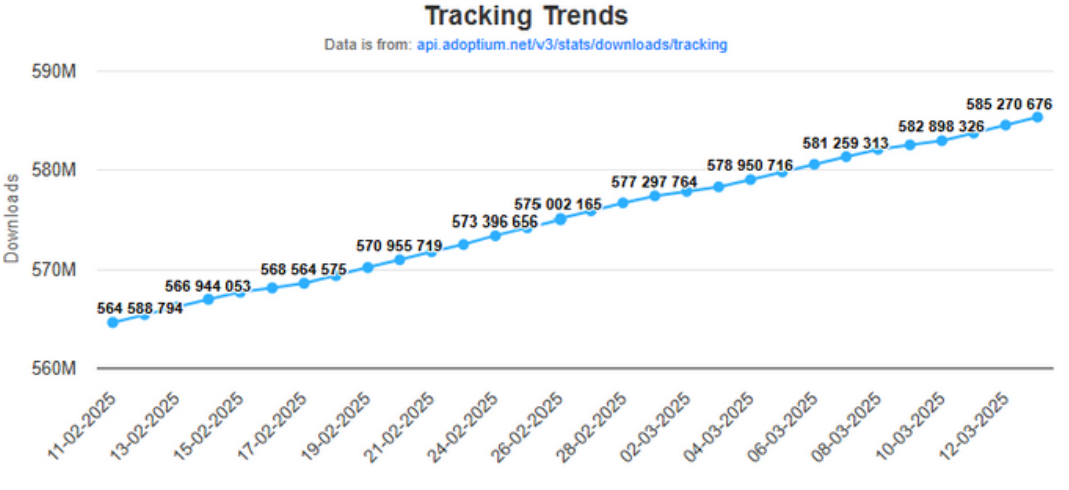
Java runtimes lacked a trusted, vendor-neutral home – until Temurin under the Adoptium brand emerged, ensuring enterprises had reliable,

well-performed, freely available Java distributions.
Originally **AdoptOpenJDK,** the project became **Eclipse Temurin under Adoptium Working Group** in 2020, with Temurin as its flagship project.

One of the core strengths of Eclipse Temurin lies in its commitment to quality. It provides T**CK-certified and AQAvit-verified binaries**, ensuring that every release meets the highest industry standards. The **Technology Compatibility Kit (TCK)** guarantees that Temurin remains fully compatible with the Java specification, while **AQAvit (Adoptium Quality Assurance) testing** subjects the binaries to extensive validation, including functional correctness, security checks, and performance benchmarks. This meticulous quality control process ensures that Temurin users can rely on stable, predictable, and secure Java runtimes.

Accessibility and cross-platform support have also been a priority for Temurin. With **support for 58 builds in a combination of versions and platforms**, the project ensures that Java developers can deploy applications across a vast range of environments, from traditional server infrastructure to modern cloud-native ecosystems. This broad compatibility makes Temurin one of the most versatile OpenJDK distributions available today.

The project's impact on the industry is evident through its **over 500 million cumulative downloads** – a testament to its widespread adoption by enterprises, cloud providers, and individual developers alike. This remarkable milestone underscores the trust that the Java community places in Temurin as a reliable, production-grade runtime.



To further strengthen its commitment to quality, the **Adoptium Working Group** has continued expanding **AQAvit**, enhancing its ability to detect regressions, ensure performance stability, and provide long-term reliability for Java applications. These continuous improvements help maintain a consistent and predictable user experience, making Temurin a preferred choice for organisations that prioritise performance and dependability.

Additionally, the introduction of the **Adoptium Marketplace** has made it easier for enterprises to find and adopt t**rusted Java binaries**. By offering a curated selection of certified builds, the marketplace provides organisations with confidence in their Java runtime choices, fostering greater security and standardisation within the Java ecosystem.

Through these initiatives, Eclipse Temurin has solidified its position as a premier open-source Java runtime, ensuring that Java remains a powerful and reliable platform for modern software development.

Eclipse IDE: Adapting to Modern Java Development Needs Through a Major Renovation

A developer's best friend for decades, the Eclipse IDE is now embracing the future with modern tooling, cloud integration, and renewed innovation.

For over two decades, the **Eclipse IDE** has been a trusted companion for Java developers. Since 2024, Eclipse IDE has been undergoing a significant technical renovation, one that will shape its future. This effort, which is the focus of 2025, follows key performance improvements made throughout last year. These changes have already made Eclipse more responsive and efficient, better aligned with modern development workflows.

But the Eclipse IDE is more than just an IDE – it is a Rich Client Platform (RCP), an extensible foundation that supports a wide range of desktop-based applications and products. The platform remains a crucial tool in various sectors where reliability and open-source vendor neutrality are essential.

Looking ahead, the **Eclipse IDE Working Group** is focusing on

creating a fully modern, platform-agnostic version of the IDE, with a renovated user experience. This transformation aims to simplify long-term maintenance and ensure a sustainable technology foundation. As part of this, the user experience is being completely redesigned to meet the needs of developers today.

This project, named **„Initiative 31,"** represents a key moment for the Eclipse IDE. It's also an opportunity for Java developers to get involved and contribute to the only truly open source IDE, helping shape its future.

Another key aspect of Eclipse's modernisation is the integration of AI-driven tools, such as the highly awaited and now available Copilot plugin for the Eclipse IDE.

Cloud Dev Tools: Adapting to Kubernetes and Microservices

The **Eclipse Foundation** leads several initiatives in **Cloud Development Tools**, focusing on **web-based** and **cloud-native** development environments. These tools enable developers to work efficiently from anywhere, leveraging **containerized** and **scalable architectures** like Kubernetes and Microprofile.

Key projects include:

Eclipse Theia is a flexible and extensible cloud and desktop IDE that is similar to VS Code. While both are hosted under the Eclipse umbrella, Theia is a completely independent project from the Eclipse IDE,. It serves a different purpose, focusing primarily on cloud-native development and providing a lightweight, customisable environment for web-based applications. In contrast, Eclipse IDE is a more traditional, feature-rich IDE, designed primarily for desktop development with extensive support for Java and other languages, as well as a robust platform for building complex, enterprise-level applications. **Theia AI**, integrated within Theia, offers a framework that simplifies the integration of AI-powered features into development tools. It provides reusable components for managing interactions with large language models (LLMs) and customising the user interface to handle data and prompts. Given the rapid evolution

of LLMs, with various models available through cloud, on-premises, or local solutions, Theia AI allows users to choose their preferred hosting solution and LLM, accommodating different needs.

Eclipse Che – A **Kubernetes-native** development workspace platform that supports collaborative, containerised, and secure development environments.

Langium – A **TypeScript-based** framework for developing **domain-specific languages (DSLs)**, providing deep integration with **VS Code** and cloud-based IDEs.

Additionally, the **MicroProfile** initiative plays a critical role in the advancement of cloud-native Java applications. As an open standard for building microservices, MicroProfile leverages Jakarta EE foundations to provide a lightweight and efficient framework for enterprise-grade applications. By focusing on portability, interoperability, and optimised performance, MicroProfile enables developers to build scalable microservices architectures that seamlessly integrate with modern cloud platforms.

As businesses continue to embrace cloud technologies, the Eclipse Foundation's efforts ensure that Java remains a powerful and adaptable language for modern software solutions.

Breaking Vendor Lock-In with Open VSX

Not to be overlooked, the Foundation also hosts and manages **Open VSX,** the only open, transparent, vendor-neutral registry for VS Code extensions, offering an alternative to proprietary marketplaces and enabling unrestricted access and sharing of extensions. As mentioned in the release announcement, these extensions cover a wide range of functionalities, from code formatting and linting to language support, debugging, and version control. What sets Open VSX apart is its commitment to openness and inclusivity. Anyone can contribute to the platform, and its extensions can be used in any compatible IDE, making it a true reflection of the diverse needs and preferences of the developer community.

A Thriving Community Moving Java Forward

Beyond technology, this is about a thriving community that keeps Java moving forward. Numerous unsung heroes – developers, maintainers, and contributors – made these advancements possible. With 644 commit authors for Jakarta EE since 2017, 339 for Adoptium since 2020, and 3,199 for the Eclipse IDE, these numbers stand as a testament to the strength of this vibrant ecosystem.

Thousands of contributors, industry leaders, and Java enthusiasts rallied behind Jakarta EE, proving that open collaboration drives real progress. The same spirit fuels Adoptium, where millions trust Temurin as their go-to Java runtime.

Major Milestones

The Eclipse Foundation has played a pivotal role in advancing open-source Java, fostering innovation, and strengthening collaboration within the industry. One of the most significant transformations has been Jakarta EE's evolution into a leading cloud-native Java platform. This shift has enabled enterprises to build scalable, resilient applications that meet the demands of modern cloud environments. Alongside Jakarta EE, the success of Temurin, an open-source Java runtime, has been marked by a major milestone, surpassing 500 million downloads while maintaining robust multi-platform support.

The modernisation of Eclipse IDE has also been a critical focus, ensuring that it remains relevant and effective for developers in the coming decade. With continuous improvements and adaptability to new technologies, Eclipse IDE remains a cornerstone of Java development. Meanwhile, Java itself continues to expand its influence in emerging fields such as Generative AI, security, and performance optimisation, proving its adaptability and enduring significance.

The growth of cloud-native Java development has been further reinforced by initiatives such as MicroProfile and Eclipse Che, which provide developers with powerful tools for building modern applications. These projects, along with increased collaboration between industry leaders, are driving sustainable innovation in the open source Java ecosystem.

To maintain this momentum, the Eclipse Foundation remains committed to strengthening governance and fostering collaboration across its projects. It actively supports the expansion of developer adoption and innovation through community engagement while facilitating the growth of new ecosystems centred around Java, AI, and cloud-native architectures. By championing these efforts, the foundation ensures that open source Java remains at the forefront of technological advancement for years to come

Ensuring Java's Future

Like the mighty oak the original programming language was named after, Java's future depends on strong roots and continuous growth, never ceasing to branch out and sprout new life. Through Jakarta EE, Adoptium, and modern tools, Java is thriving – but its true strength lies in the diverse community that nurtures it. By staying engaged and contributing, we ensure Java remains open, innovative, and resilient for generations to come. Just as an oak stands firm through the storms of time, Java, with the Eclipse Foundation's leadership, will continue to evolve and flourish. The best days are still ahead – shaped by those who build with it.

How You Can Contribute to Java's Future Success

Here is what you can do to get involved:

- Join an **Eclipse Working Group** (Jakarta EE, Adoptium, MicroProfile, Eclipse IDE, etc.)
- Advocate for **vendor-neutral, open-source Java** within your company, be a public adopter, and check on Temurin and Cloud Dev Tools adopters page.
- Participate in **community discussions, events, and projects.**

Help improve Java's ecosystem by **contributing code, documentation, or sponsorship.**

#JAVAPRO #SECURITY

How to Containerize a Java Application Securely

Author:

Mohammad-Ali A'râbi is Software Engineer, Docker Captain, Author of „Docker and Kubernetes Security", Gamer, especially a huge fan of Fallout: New Vegas and Mortal Kombat series.

JCON 2025 SPEAKER

TL;DR

- **Containerization** or **Dockerization** is the process of packaging an application and its dependencies into a Docker image.
- It's beneficial to containerize your Java application as it provides a consistent environment for development, testing, and deployment.
- Java is a compiled language, so your Docker image will only contain the compiled Java bytecode and the Java runtime environment.
- **Supply-chain security** is the process of checking the dependencies in your application for vulnerabilities.

- **SBOM** (Software Bill of Materials) is a list of all the dependencies in your application.
- It's a good practice to generate an SBOM when building your Docker image and push it to a registry for later reference

Technical Requirements

In this article, we will use the following tools:

- Git: To push the code to a git repository.
- Docker Desktop: To build and run the Docker image.

Most of the Docker commands are available on Docker Engine as well, but we will use some are only available on Docker Desktop, e.g. docker init. You could dodge that particular command by copying the Dockerfile and other generated files from the GitHub repository. I also assume that you use a Unix-like shell (e.g., bash) to run the commands. So, the commands are compatible with Linux, macOS, and Windows Subsystem for Linux (WSL).

Hello World Java Application

For this article, we will use a toy Spring Boot application. Let's create a template Spring Boot application using the Spring Initializr. Visit start.spring.io and create a new project with the following settings:

- Project: Maven
- Language: Java
- Spring Boot: 3.4.4
- Packaging: Jar
- Java: 24

I also named added the following metadata, but you can choose your own:

- Group: io.dockersecurity
- Artifact: hello
- Name: hello
- Description: Demo project for Docker Security showcase
- Package name: io.dockersecurity.hello

Click on the „Generate" button to download the project. Then unzip the project and go to the root:

```
unzip hello.zip
cd hello
```

Let's initialize a git repository here and commit the code:

```
git init
git add .
git commit -m „Initial commit"
```

Now, let's push the code to a remote repository on GitHub, to use the CI/CD pipeline later. To do it, you should create a new repository on GitHub and copy the URL of the repository:

```
git remote add origin <your-repo-url>
git push -u origin master
```

My repo URL was git@github.com:DockerSecurity-io/hello.git, so you can access the code here: github.com/DockerSecurity-io/hello.

Let's Compile

To run the project locally, we have two options:

- Run the project on the host machine, in which case you need to have Java and Maven installed.
- Run the project in a Docker container, in which case you only need Docker installed.

I'll go for the latter, because I don't have Java 24 and Maven installed on my machine.

```
docker init
```

The interactive wizard will detect that you have Java project. Press Enter to accept the „Java" option, accept the default for source directory and Java version, and enter the port manually:

```
? What application platform does your project use? Java
? What's the relative directory (with a leading .) for
your app? ./src
? What version of Java do you want to use? 24
? What port does your server listen on? 8080
```

The following files are generated:

- Dockerfile: The Dockerfile to build the image.
- compose.yaml: The Docker Compose file to run the image.
- .dockerignore: The .dockerignore file to exclude files from the build context.
- README.Docker.md: The README file with instructions on how to build and run the image.

Let's take a look into the Dockerfile:

```
# syntax=docker/dockerfile:1
################################################################
###################

FROM eclipse-temurin:24-jdk-jammy as deps

WORKDIR /build

COPY --chmod=0755 mvnw mvnw
COPY .mvn/ .mvn/

RUN --mount=type=bind,source=pom.xml,target=pom.xml \
    --mount=type=cache,target=/root/.m2 ./mvnw
dependency:go-offline -DskipTests

################################################################
###################
```

```
FROM deps as package

WORKDIR /build

COPY ./src src/
RUN --mount=type=bind,source=pom.xml,target=pom.xml \
    --mount=type=cache,target=/root/.m2 \
    ./mvnw package -DskipTests && \
    mv target/$(./mvnw help:evaluate -Dexpression=project.
artifactId -q -DforceStdout)-$(./mvnw help:evaluate
-Dexpression=project.version -q -DforceStdout).jar target/
app.jar

################################################################
###################

FROM package as extract

WORKDIR /build

RUN java -Djarmode=layertools -jar target/app.jar extract
--destination target/extracted

################################################################
###################

FROM eclipse-temurin:24-jre-jammy AS final

ARG UID=10001
RUN adduser \
    --disabled-password \
    --gecos „“ \
    --home „/nonexistent“ \
    --shell „/sbin/nologin“ \
    --no-create-home \
    --uid „${UID}“ \
    appuser
USER appuser
COPY --from=extract build/target/extracted/dependencies/
```

```
./
COPY --from=extract build/target/extracted/spring-boot-
loader/ ./
COPY --from=extract build/target/extracted/snapshot-
dependencies/ ./
COPY --from=extract build/target/extracted/application/ ./

EXPOSE 8080

ENTRYPOINT [ „java“, „org.springframework.boot.loader.
launch.JarLauncher“ ]
```

The following base images are used:

- eclipse-temurin:24-jdk-jammy: The Java Development Kit (JDK) image to compile the Java code, based on Ubuntu 22.04 (Jammy Jellyfish).
- eclipse-temurin:24-jre-jammy: The Java Runtime Environment (JRE) image to run the compiled Java code, based on Ubuntu 22.04 (Jammy Jellyfish).

At the time of writing this article, Java 24 was recently released, so the Eclipse Temurin images are not available yet. To address that, we will use the following images instead:

- sapmachine:24-jdk-ubuntu-noble: The JDK image based on Ubuntu 24.04 (Noble Numbat).
- sapmachine:24-jre-ubuntu-noble: The JRE image based on Ubuntu 24.04 (Noble Numbat).

These images are provided by SAP, the German vendor of ERP systems, who also provides a free and open-source distribution of the OpenJDK. You can find their Java images on Docker Hub: hub.docker.com/_/sapmachine.

After replacing the base images in the Dockerfile, you can execute the following command to build and run the image:

```
docker compose up
```

This command will use the Docker Compose configuration that looks like this:

```
services:
  server:
    build:
      context: .
    ports:
      - 8080:8080
```

The server service will build the image using the Dockerfile in the current directory and expose the port 8080 on the host machine.

The application starts successfully, but also directly stops with exit code 0. This means that the application is running, but there is no endpoint to access it. Let's add a simple endpoint to the application.

Note. Don't forget to commit everything and push before proceeding.

Add a Controller

Let's add a simple controller to the application. Create a new file src/main/java/io/dockersecurity/hello/HelloController.java with the following content:

```
package io.dockersecurity.hello;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.
RestController;

@RestController
public class HelloController {

    @GetMapping(„/“)
    public String hello() {
        return „Hello, Docker Security!“;
    }
}
```

Also, add the following block to your pom.xml to include the Spring Web dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Now, let's build the project and run it again:

```
docker compose up --build
```

The application should start successfully and you can access it at localhost:8080. Let's check the endpoint:

```
curl http://localhost:8080
```

It should say „Hello, Docker Security!". Voilà!

Extract the SBOM

SBOM, or Software Bill of Materials, is a list of all the dependencies in your application. It's a good practice to generate an SBOM when building your Docker image and push it to a registry for later reference.

The image that is built during the docker compose up command is tagged automatically as hello-server. Let's create an SBOM for it with the following command:

```
docker sbom hello-server
```

This command lists all the dependencies and shows them in the terminal in a human-readable format. To create a proper SBOM file, let's use SPDX, a standard for SBOMs:

```
docker sbom hello-server --format spdx > sbom.json
```

This command creates an SBOM in the SPDX (Software Package Data Exchange) format and saves it to the sbom.json file.

It would be great to push this file to a registry for later reference. To do that, we can use attestations!

Attestations

When building a Docker image, you can add attestations to it. Attestations

are metadata that can be used to verify the integrity of the image. The attestation can be uploaded into the registry along with the image, and later used to verify the image.

To add an SBOM attestation to the image, we can use the following command:

```
docker buildx build --tag <namespace>/<image>:<version> \
--attest type=sbom --push .
```

This command builds the image and adds an SBOM attestation to it. The image is then pushed to the registry. I used the following tag and pushed my image: aerabi/spring-hello:latest.

Note. To use this command, you should turn on containerd image store on your Docker Desktop. You can do it in the Docker Desktop settings.

To verify the SBOM attestation, you can use the following command:

```
docker buildx imagetools inspect
<namespace>/<image>:<version>
```

You can check this command with the image I pushed:

```
docker buildx imagetools inspect aerabi/spring-
hello:latest \
```

As Java is a compiled language, we used different images for building and running the application. This is called „multi-stage build". A vulnerability in the build image could also affect the runtime image, so it's important to check dependencies in all stages of the build process.

To let BuildKit check the dependencies in all the stages, you can add the following ARG command under each FROM command in the Dockerfile:

```
FROM sapmachine:24-jdk-ubuntu-noble as deps
ARG BUILDKIT_SBOM_SCAN_STAGE=true

FROM deps as package
ARG BUILDKIT_SBOM_SCAN_STAGE=true

FROM package as extract
ARG BUILDKIT_SBOM_SCAN_STAGE=true
```

Build the image again with SBOM attestations and push it to the registry:

```
docker buildx build --tag <namespace>/<image>:<version>
--sbom=true --push .
--format „{{ json .SBOM.SPDX }}“
--format „{{ json .SBOM.SPDX }}“
```

Now, let's check the SBOM attestation for the image:

```
docker buildx imagetools inspect
<namespace>/<image>:<version> \
--format „{{ json .SBOM.SPDX }}“ >> sbom.multi.json
```

You can compare the SBOM with the one we stored initially to see if there are any differences. The one extracted from the multi-stage build should be more comprehensive.

Docker Scout

Docker Scout is a tool that can be used to check for vulnerabilities in Docker images. It is shipped together with Docker Desktop. Let's check the vulnerabilities in the image we built:

```
docker scout cves <namespace>/<image>:<version>
```

For my image, it shows the following output:

```
SBOM obtained from attestation, 578 packages found
✓ Provenance obtained from attestation
✓ No vulnerable package detected
```

This means that the image is free from vulnerabilities. Great job! Also, the SBOM attestation is obtained from the image, so no further scan was needed to get the list of dependencies.

Conclusion

In this article, we learned how to containerize a Java application securely. We used a Spring Boot application as an example and built a Docker image for it. We also generated an SBOM for the image and pushed it to a registry with an attestation. Finally, we checked for vulnerabilities in the image using Docker Scout.

It's also important to note that the build process should be automated and integrated into the CI/CD pipeline. The GitHub repository for the project is available at github.com/DockerSecurity-io/hello and contains a GitHub Actions workflow for building and pushing the image to the registry.

I hope this article was helpful and you learned something new.

#JAVAPRO #FRAMEWORK #API

Next Generation Caching & In-Memory Searching

Author:

Markus Kett and his teams have been working on IDE tools for Java and database development, as well as on various open-source projects for 20 years. Markus is CEO and co-founder of MicroStream, the company behind the Eclipse open-source projects EclipseStore, Eclipse Serializer, and RapidClipse IDE. He is also the editor-in-chief for the free JAVAPRO magazine in Germany and the founder and co-organizer of the Java community conference series JCON. He is an independent editor for several magazines and a speaker at numerous international developer conferences, user

When Traditional Databases Reach Their Limits

Enterprise applications frequently face performance bottlenecks that stem from the underlying persistence layer. Despite decades of database evolution, many core applications still struggle with latency, throughput, and architectural complexity. These issues are particularly evident in systems that depend on complex join operations across multiple tables, need to process unstructured data, or must execute cross-database queries spanning disparate systems to fulfill analytical workflows.

In such environments, it is common to see a combination of various technologies being deployed in tandem: a traditional relational database, a distributed cache, a NoSQL database, and sometimes a dedicated search server such as Elasticsearch. While each of these systems addresses a specific technical challenge, their co-existence leads to architectural fragmentation, high infrastructure costs, duplicated data models, and increased development and maintenance efforts. Moreover, the overall latency and resource usage often remain suboptimal due to inherent I/O-bound limitations and serialization overheads between systems.

Eclipse Data Grid addresses these challenges through a Java-native in-memory data layer that is positioned between applications and their underlying databases. Its primary function is to offload complex data processing from the database tier and execute it within memory, thereby significantly improving performance while reducing infrastructure load and database licensing costs. Acting as a general-purpose, distributed in-memory data grid, Eclipse Data Grid enables low-latency, high-throughput data access and processing capabilities using plain Java.

A General-Purpose In-Memory Data Grid for Java

Eclipse Data Grid is a distributed, Java-based in-memory data processing platform that supports a broad range of use cases, from traditional caching scenarios to advanced in-memory computation. Unlike conventional caches, which are generally limited to key-value lookups and basic TTL-based eviction strategies, Eclipse Data Grid provides developers with a programmable data layer. This allows for the execution of custom business logic and complex data operations directly within memory, using the full expressive power of the Java programming language.

Support for Common Caching Use Cases

At its foundation, Eclipse Data Grid supports standard caching requirements. It can function as a distributed key-value cache, conforming to familiar APIs such as JCache (JSR 107), enabling straightforward adoption for applications that already rely on caching abstractions. For these use cases, it offers robust performance, high availability, and horizontal scalability across JVM nodes.

Next-Generation In-Memory Data Processing

Beyond traditional caching, Eclipse Data Grid introduces a novel approach to in-memory computing by empowering developers to build full-featured, Java-native applications directly on top of the grid. Developers can work with the native Java object model, avoiding the need for object-relational mapping, JSON serialization, or schema translations. Complex object graphs, including circular references, polymorphic structures, and nested collections, are supported without compromise.

With support for Java Streams, including parallel streams, developers can implement expressive queries, aggregation pipelines, and graph traversals that operate directly on in-memory data structures. The Lucene integration enables fulltext search. Any other Java libraries can also be seamlessly integrated for advanced search capabilities. These features make Eclipse Data Grid an ideal platform for processing large data volumes, performing real-time analytics, and executing business-specific algorithms without the overhead of moving data between systems.

A key differentiator is the off-heap bitmap indexing engine, which enables sub-millisecond search across billions of Java objects. These indexes operate independently of the JVM heap, providing both performance and memory efficiency.

In this architecture, Java assumes the role that stored procedures or proprietary scripting languages play in traditional databases. Developers can implement any business logic, algorithm, or transformation directly in Java, using familiar paradigms and tooling.

ACID-Compliant Persistence with EclipseStore

Persistence is handled by EclipseStore, a companion Eclipse Foundation project that provides ACID-compliant, object-graph-oriented persistence. Unlike traditional caching systems, which often rely on naive snapshot-based persistence, EclipseStore provides transactional consistency, journaling, and delta-based storage. This ensures that only modified objects are persisted, reducing write amplification and enabling fine-grained rollback and recovery capabilities.

By integrating EclipseStore, Eclipse Data Grid offers consistent, durable storage semantics across JVM nodes, and a convenient schema migration concept without object-flattening procedures. The system supports lazy loading of object graphs, fine-grained locking on object graph in memory, and optimized data formats tailored to the Java runtime.

This architectural model also addresses one of the major limitations of traditional distributed caches: their dependence on high volumes of RAM. In common caching solutions, all cached data must reside entirely in RAM to ensure low-latency access. If an application requires 128 GB of cached data, the infrastructure must provision at least 128 GB of free memory - often more due to meta data overhead and replication requirements. Furthermore, to avoid data loss during node failures, traditional caches employ sharding with replication, which further multiplies RAM needs. For instance, a replication factor of two would double the required RAM to 256 GB.

Eclipse Data Grid overcomes this limitation through its native integration with EclipseStore and the use of GigaMap, a high-performance, Java-native structure with built-in lazy loading. Data is automatically persisted and can be loaded into memory on demand. This means that only frequently accessed (hot) data needs to be kept in RAM, while the rest can remain on disk or in a BLOB storage like S3. Consequently, applications can work with datasets far larger than available memory - for example, managing 128 GB of data with nodes provisioned with just 16 GB of RAM each. Since EclipseStore avoids costly ORM mapping and operates on native object serialization, access to persisted data remains highly performant.

Distinguishing Eclipse Data Grid from Traditional Caching Solutions

Traditional distributed caches are often limited by their reliance on key-value semantics and their inability to process complex object graphs in memory. These systems typically store data in a serialized form, often as JSON or binary blobs, which breaks object references and requires developers to manage serialization and deserialization manually. This not only introduces performance penalties but also limits the expressiveness and type safety of in-memory operations.

In contrast, Eclipse Data Grid retains the full structure of Java object graphs in memory. Object integrity is preserved, reference cycles are supported, and collections behave as expected. This native representation allows for rich querying and data manipulation without leaving the Java type system.

Moreover, the use of Java Streams and third-party libraries enables more sophisticated querying and indexing mechanisms than traditional caches offer. While distributed caches might support rudimentary SQL-like query languages or map-reduce APIs, they generally fall short in supporting real-world business logic that demands flexibility, integration, and developer productivity.

By moving business logic directly into the in-memory layer, Eclipse Data Grid reduces reliance on heavyweight database engines, minimizes data movement, and eliminates the impedance mismatch between the object-oriented application and the underlying storage.

Architecture and Cluster Design

Eclipse Data Grid follows a writer-reader cluster model. At the core is a single writer node responsible for executing all write operations. This includes the execution of EclipseStore store methods, which persist object changes to a durable storage medium. Write operations are strictly ACID-compliant and isolated from read operations, which are delegated to one or more reader nodes.

The writer node acts as the authoritative source of truth. Once a store operation is executed, the updated object or object graph is serialized and published to a Kafka stream. Reader nodes consume this stream and merge the updates into their own local in-memory object graphs. This design ensures that readers operate on an eventually consistent view of the data, while maintaining full consistency on the writer node and local consistency on each reader.

All data access operations, including search, computation, and transformation, are performed on reader nodes. These nodes can be horizontally scaled to serve read-heavy workloads and are stateless with respect to write operations.

External applications and services interact with the data grid via RESTful APIs. Each node exposes a set of REST endpoints that can be generated from templates, allowing developers to implement complex operations in the in-memory layer and expose them without requiring custom client libraries. This approach promotes interoperability, enabling applications written in Python, JavaScript, Go, or other languages to leverage the grid without integrating a Java-based SDK.

The infrastructure can be deployed on-premises, in any cloud environment, or as a managed service. The only requirement for deployment is a Kubernetes cluster, which can be provisioned using Helm charts. MicroStream also provides a SaaS offering for evaluation, development, and testing, with support for elastic scaling and automated configuration.

Getting Started with Eclipse Data Grid

Adopting Eclipse Data Grid requires familiarity with core Java features for in-memory data processing. Developers should be proficient in designing domain models, working with Java Streams, and utilizing modern concurrency features such as virtual threads where applicable.

The first step is understanding EclipseStore and its persistence model. Developers should learn how to configure object graph storage, perform CRUD operations, implement indexing strategies, and utilize lazy loading and locking APIs. Once this foundation is established, the next step is to understand the architecture of the Eclipse Data Grid cluster, including the roles of writer and reader nodes, Kafka integration, and REST endpoint generation.

A strong understanding of the Java language, its standard library, and available open-source tools for data processing and search will significantly improve the effectiveness of in-memory applications built on this platform.

Target Use Cases and Application Domains

Eclipse Data Grid is designed to address a broad set of performance- and complexity-related challenges in enterprise applications. It is particularly well-suited for systems experiencing performance issues

due to ORM overheads, complex SQL joins, or latency introduced by multiple distributed components. Applications built with Hibernate or JPA that suffer from N+1 query problems, excessive serialization, or lack of control over fetch behavior can benefit significantly from the grid's native object graph handling.

In scenarios where NoSQL databases are used but schema flexibility and consistency guarantees are still needed, Eclipse Data Grid offers a compelling alternative by enabling flexible object modeling with strong transactional semantics.

The platform also excels in use cases involving data analytics, real-time reporting, session management, and micro-batch or event-driven processing. Its ability to execute custom logic in memory makes it particularly valuable for pre-processing data in machine learning pipelines, simulating complex workflows, or performing fine-grained filtering on incoming data streams.

Finally, from a cost and maintenance perspective, Eclipse Data Grid simplifies application architecture by reducing the number of external systems required. By consolidating caching, persistence, and computation into a unified, Java-native layer, it eliminates the need for separate caches, search engines, and data transformation services, thereby lowering both operational complexity and licensing costs.

Conclusion

Eclipse Data Grid represents a new generation of in-memory data processing platforms tailored for Java developers. By enabling complex, application-specific logic to run entirely within memory and by supporting ACID-compliant persistence of full Java object graphs, it bridges the gap between the performance requirements of modern applications and the limitations of traditional database technologies. Its architecture supports scalable, distributed deployments, facilitates integration through REST APIs, and empowers developers to leverage the Java ecosystem to its fullest. For organizations facing challenges in data performance, architectural sprawl, or database cost management, Eclipse Data Grid offers a robust and future-proof solution.

For your IT projects you don't need a know-it-all.

You need a {BUDDY}

Richard Fichter
CEO @ XDEV

Java Champions

Outdated software? Rising maintenance costs? Security risks? We help you to make Java applications fit for the future - with a clear concept and at eye level. In addition to modern tools, we offer premium support for your **Java modernization.**

- **Not a know-it-all - a buddy:** We support your team with pragmatic methods without playing the wise guy.
- **Modernization with strategy:** agile methods & proof of concept for a secure update.
- **Robust solutions:** We rely on proven technologies and practical concepts - without unnecessary complexity.

Let us move your Java project forward together!

BEST {BUDDYS} IN CODE

trusted by

ATU · EMPORIO ARMANI · CONRAD · Fraport · Red Bull · Allianz

Arrange a free discovery call here!

XDEV