
1010
jdriven
Tech Radar
Fall 2024
10th edition
2024
FACTOR
LATEST
&
GREATEST

Opening Remarks

We are incredibly proud to present the 10th edition of the JDriven Tech Radar. This milestone marks a decade of technological progress, innovation, and an unwavering passion for the field of software engineering. Since the very first edition, we've seen countless technologies, tools, and trends come and go, each making its own impact on the way we build and deliver software.

Our Tech Radar is more than just a list of technical choices; it is a reflection of the deep expertise and dedication within JDriven. By evaluating and adopting new technologies, we strive not only to improve ourselves but also to make a lasting impact on the software world. We are proud that our Tech Radar has become a guide for many in the industry, helping to drive continuous innovation in the discipline of software engineering.

In this edition, we continue to focus on making conscious, future-oriented decisions that promote the quality and sustainability of software development. We look forward to continuing to innovate and push boundaries in the years ahead, together with our community.

What makes this anniversary edition extra special is that it is illustrated by the collective contributions of all our colleagues, celebrating 10 editions of JDriven Tech Radars.

Thank you to everyone who has made this possible, let's keep building an active and growing future for software engineering!

Erik Pronk
Managing Partner JDriven

Commit. Develop. Share.

Introduction

Our experienced specialists participate every day in the development of many different software projects all over the Netherlands, and are involved in worldwide professional communities. Each half year our engineers gather to discuss the latest emerging trends and developments in software engineering. We seek to capture these trends in a technology radar. Each edition of the radar shows the changes in these trends, compared to the previous edition. A shift can indicate that we see a technology is becoming more, or less, relevant for certain use cases. If a trend does not show up in later editions, it signifies that there are no relevant developments or experiences that cause us to shift our assessment. With this document we will discuss the shifts we have observed over the past half year. This analysis guides us in deciding which technologies we use and recommend.

Tech Radar

The idea to create a Tech Radar was initiated by ThoughtWorks. They periodically showcase their view on new trends and development. In addition, they advise everyone to define their own radar.

At JDriven we fully support that view. Building a Tech Radar is an instructive and valuable experience, where we mutually share our knowledge and create a common awareness around technology. We believe that specialists should be able to create and maintain their own toolset to perform their job to their best ability. When composing a radar, you facilitate a broad discussion on technology, so organisations can strike the right balance between the risks and rewards of innovation. We can help you to kick-start these discussions in your organisation. Let your teams innovate and inspire each other. Together they can draw up a set of technologies and techniques that can accelerate your business.

Overview

The radar consists of quadrants and rings, containing blips to indicate technologies of interest.

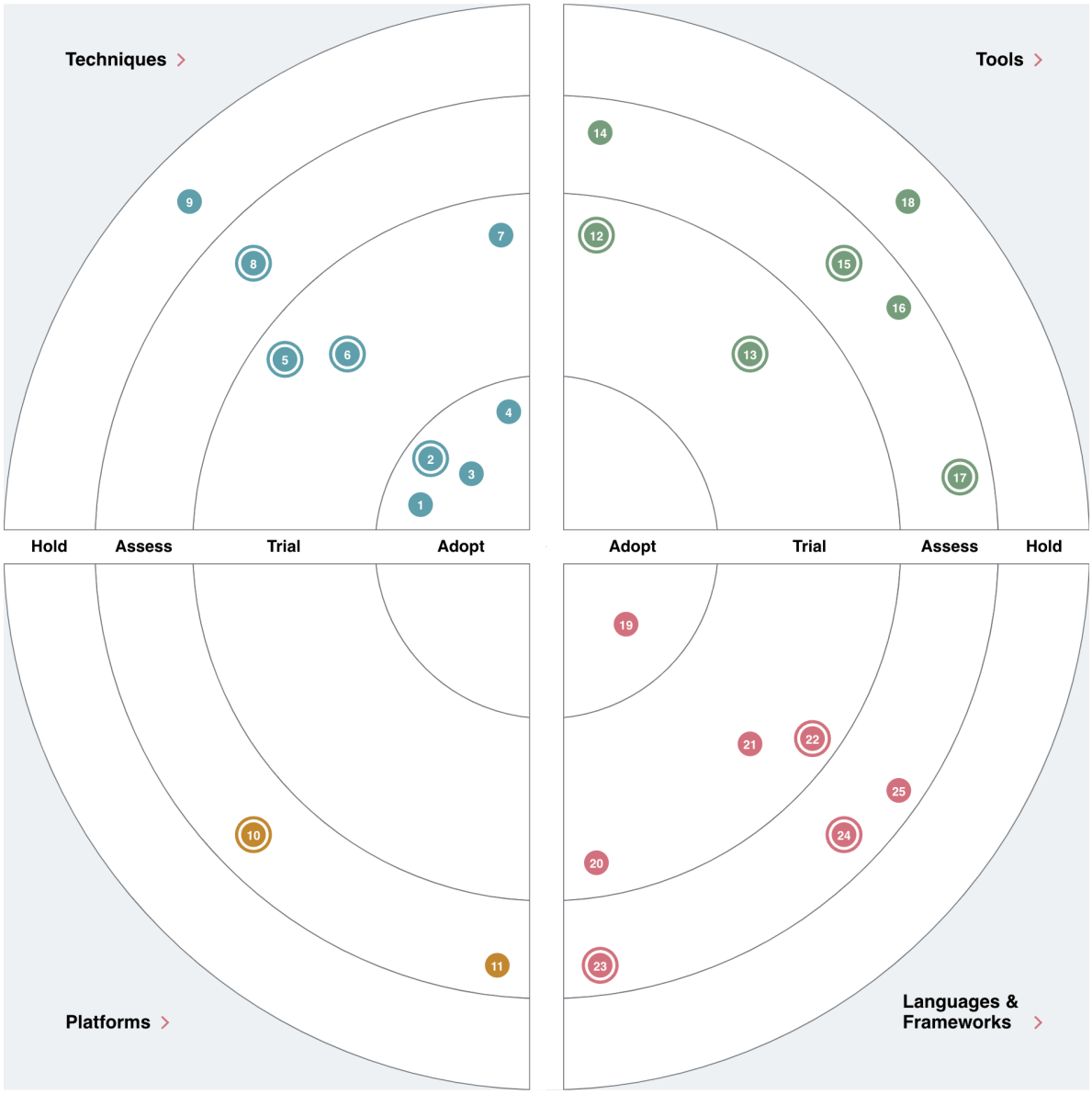
The quadrants subdivide the different subjects into categories:

- **Languages and frameworks** that support developers in their daily tasks.
- **Platforms** to deploy and execute programs.
- **Techniques** help developers create better software.
- **Tools** aid in the development and delivery process.

The rings in each quadrant indicate which phase of the adoption cycle we believe a technology is currently at:

- **Adopt**: We recommend to use this technology, wherever it fits the requirements.
- **Trial**: We advise to gain experience with this technology, wherever a project allows for a certain degree of risk.
- **Assess**: Interesting topic to learn more about and assess its future impact; yet too early to use in production.
- **Hold**: Do not deploy in a project not already using this technology.

In the subsequent sections we will delve into our views on recent developments in the field of software engineering more closely.

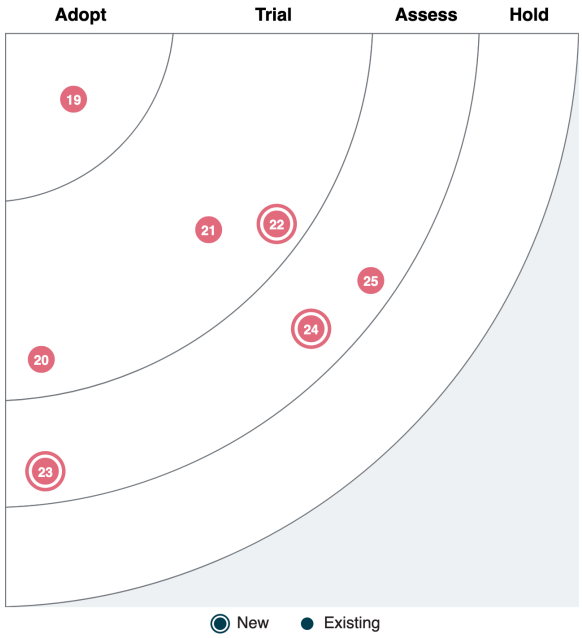


Commit. Develop. Share.

Tools	Languages & frameworks
ADOPT - **TRIAL** Bruno Devoxx Genie **ASSESS** Claude 3 CRaC GitHub Codespaces warp.dev **HOLD** OpenTofu	**ADOPT** Ktor **TRIAL** No frameworks Spring Modulith Timefold AI **ASSESS** Helidon Micronaut SourceGen Vector API **HOLD** -
Platforms	**Techniques**
ADOPT - **TRIAL** - **ASSESS** AsyncAPI OpenFGA **HOLD** -	**ADOPT** 1 TFB NFC CVE awareness GitOps Inner Source **TRIAL** 15-factor app eventmodeling Sustainable software engineering **ASSESS** Shape Up **HOLD** Distributed Monolith

Commit. Develop. Share.

Languages & frameworks



Adopt
19. Ktor

Trial
20. No frameworks
21. Spring Modulith
22. Timefold AI

Assess
23. Helidon
24. Micronaut SourceGen
25. Vector API

Ktor

ADOPT

Ktor is a Kotlin framework for quickly creating web applications with minimal effort. It is relatively lightweight, as it supports only the more common and contemporary functionalities required for web services, like routing, templating and serialization. By leveraging Kotlin's coroutines at its core, it makes it easy to build responsive web services that should scale relatively well. It also supports multiple HTTP engines out of the box.

The framework has existed for quite some time, and seems to get a decent amount of traction within the community. Besides that, we are noticing an increased usage of Ktor among our customers. Mainly because it offers a lightweight alternative for implementing microservices compared to e.g. a (typically heavier) Spring Boot application, especially when combined with an ORM library like Exposed.

Therefore, we have assigned Ktor to the *Adopt* ring.

No frameworks

TRIAL

Almost everyone uses frameworks for building applications. You can often build an application quickly and relatively easily with a framework, because a framework typically simplifies the following aspects (often using annotations and conventions):

- Loading configuration via properties
- Dependency Injection
- Built-in REST server
- Built-in JSON marshalling/unmarshalling
- Built-in REST client
- Built-in ORM
- Built-in transaction management
- Built-in log library

But frameworks also have disadvantages, often due to their size. You often notice this after some time, for example, when you want to use an additional library (dependency) that conflicts with one of the (transitive) libraries of the framework. Or when you upgrade the framework or such a library to a newer version. Suddenly, inexplicable errors can occur, such as complicated version conflicts of transitive dependencies, annotations no longer working, vague runtime errors causing tests or the application itself to fail to start, etc. It often takes a lot of effort to identify and resolve the cause because the framework is so large and complex and does a lot 'automagically' via annotations and/or conventions (which you don't all know). As a result, all the time saved by using a framework can turn out to be a short-term gain, but a long-term loss.

Why don't we just build an application without frameworks, and only with the lightweight libraries we need (cherry-picking)? This is very feasible with, for example, the following approach:

- Loading configuration via property files can be done with the standard Java `Properties` class. You don't need a library for that.
- Dependency Injection can also be done very well by yourself using constructor injection. You don't need IoC (Inversion of Control) and therefore no framework.
- A lightweight HTTP server like Jetty or Undertow is sufficient for exchanging text strings over HTTP. And with a library like Jackson, you can marshal/unmarshal JSON strings yourself.
- A simple HTTP client like OkHttp can be used to build a REST client in a similar way.
- Hibernate is also fine to import as a standalone library if you need an ORM.
- And Hibernate also provides transaction management via `EntityManager.getTransaction().begin()` and `EntityManager.getTransaction().commit()`.
- There are several log libraries available for standalone use, for example, SLF4J.

This approach requires a bit more code than with a framework where you get a lot for free via conventions and annotations. But not that much more. The big advantage, however, is that you bring in fewer transitive dependencies and thus less complexity. Also, the code is more explicit, with fewer 'automagic' annotations and conventions. This makes debugging easier. Ultimately, this setup is can be simpler than with a large cumbersome framework that contains a lot more than you actually use (YAGNI - You Aren't Gonna Need It). In other words, a good example of KISS (Keep It Simple Stupid). If a framework is not required, it can certainly be worthwhile to opt for a no-framework approach.

Spring Modulith

TRIAL

Domain driven design has helped developers understand that a software architecture can and must facilitate the evolution of software systems to follow changes in business requirements and insights. Using microservices is a way of isolating functional modules to that end, but it introduces challenges with regard to service orchestration and HTTP based communication. This has made developers reconsider whether this modularity can be achieved and enforced in monolithic applications.

Spring has been a framework that provides technical stereotypes such as @Controller, @Repository, etc. without regard for how components depend on each other. With Spring Modulith, it aims to be more opinionated on the domain-aligned structure of a monolithic Spring Boot application. In practice, it seems to mean that a properly set up, well-structured Spring Boot application using Spring Modulith will limit candidates for dependency injection based on package structure convention. It offers ways to deviate from the defaults by customizing module detection.

Spring Modulith has additional annotations available to mark packages as part of a chosen architectural pattern. Besides this, Spring Modulith also supplies a number of ArchUnit tests to verify the chosen architectural pattern. By default, it supports Layered, Onion and Hexagonal Architecture and has the ability to generate different kinds of documentation standards. Spring Modulith also provides a new way of integration testing modules independently, which will be a lot faster than the old @SpringBootTest because it only instantiates the classes of the module under test. Furthermore, they recommend using events to provide communication between the modules.

Recently Spring Modulith has been released officially, and we see Spring Modulith as a good way to add modulairity to a monolithic application.

Timefold AI

TRIAL

Timefold AI is an open-source constraint solver available for Java, Python, and Kotlin. It is designed to optimize complex planning problems such as scheduling, routing, and resource allocation. Using constraint satisfaction programming, a score can be assigned to a solution, and the problem can be optimized by utilizing one of the built-in search algorithms. Timefold AI builds on the foundation of OptaPlanner, a project that is no longer under active development, with improvements focused on performance, including an enterprise version that offers multithreading for enhanced efficiency.

At JDriven, we have observed recurring demand for constraint solvers in client projects and see Timefold AI as a promising alternative to custom-built solutions, especially for problems of moderate size. Hence, we place Timefold AI in the *Trial* ring, encouraging teams to experiment with it.

Helidon

ASSESS

Helidon is an open-source microservice framework, comparable to Spring Boot and Quarkus, and offers a lightweight alternative akin to Ktor and Vert.x. It supports both reactive and imperative programming styles, with two versions: Helidon SE (Standalone Engine), a lightweight, reactive framework, and Helidon MP (MicroProfile), which provides full compatibility with Jakarta EE MicroProfile. In version 4.0, Helidon introduced significant enhancements, including native support for virtual threads, improved gRPC integration and performance optimizations aimed at reducing resource consumption.

Helidon is backed by Oracle, which can be seen as both a strength and a potential limitation. On the one hand, Oracle's backing provides strong enterprise support and long-term stability. On the other

hand, some teams may have concerns over vendor lock-in or dependency on a large corporate entity for innovation.

Given the developments in version 4.0, we advise teams looking for alternative Java-based microservice frameworks to consider Helidon. Therefore, we place Helidon in the *Assess* ring.

Micronaut SourceGen

ASSESS

Micronaut SourceGen is a library that allows you to generate source code for Kotlin and Java. It is a fork from JavaPoet and has been updated to use newer Java constructs like records. Micronaut SourceGen provides an easy to use API to define the structure of the source code that needs to be generated. The library uses an annotation processor to generate the source code during the Java or Kotlin compilation.

Included with the library are annotations to generate builders and withers for records. This could replace the builder annotations provided by Lombok in a project.

The nice feature of the library is that it uses the default annotation processor of the compiler. Only new source files are created and it will not mangle existing class files. The API to create the source code is easy to use. The library is placed in *Assess* to try out and experiment with.

Vector API

ASSESS

The Vector API is an incubator API to predictably leverage SIMD (single instruction, multiple data) instructions on the Java platform. These instructions perform operations on multiple primitive data elements simultaneously, up to 16 elements on modern microarchitectures. This can lead to substantial performance benefits when the same mathematical operation(s) need to be applied to a large number of elements. Originally released as an incubator API in Java 16, it is planned to be merged into the Java mainline when project Valhalla becomes available.

Under some circumstances, the optimizing compiler in e.g. OpenJDK's HotSpot VM can auto-vectorize operations itself. Relying on this behaviour for performance tends to be precarious, though. Only if all implicit conditions are met, the optimizing compiler will perform the auto-vectorization. Furthermore, the Vector API leverages Intel's SMVL library to vectorize common mathematical functions.

We have put the Vector API in the *assess* category. Although it is scheduled to be released under its eighth incubator for the Java 23 release this fall, it is still not finalized. Its main goal is to alleviate performance bottlenecks on parts of the code that depend heavily on numerical calculations. Hence it should be used only after profiling indicates there is a performance issue, as code leveraging the Vector API will be more difficult to read than regular operations on primitive types. Also, further improvements in JIT technology might reduce the need for explicit annotations of vector types.

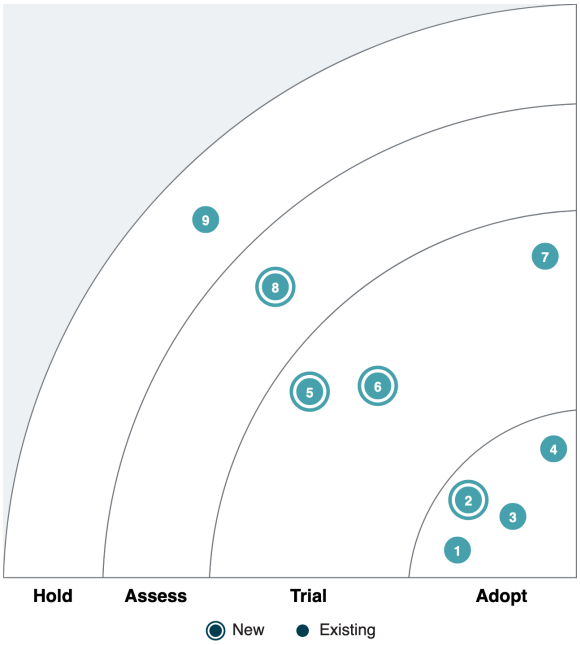
Techniques

Adopt
1. CVE awareness
2. GitOps
3. Inner Source
4. 1 TFB NFC

Trial
5. 15-factor app
6. eventmodeling
7. Sustainable software engineering

Assess
8. Shape Up

Hold
9. Distributed Monolith



CVE awareness

ADOPT

Within the world of software engineering, MITRE is mostly known for their database of Common Vulnerabilities and Exposures (CVE) that they maintain and make public. The MITRE organization has a history with the American department of defense. Many development teams by now outfit their CI/CD pipeline with automated tools for scanning dependencies to see if any CVEs have been found in those.

This is a good practice. Unfortunately, we've observed a lack of awareness among developers of what these tools do, and what findings mean. When notified of a CVE by a tool like OWASP DependencyCheck, it is still up to the developer to analyse the finding, to determine its impact and decide on possible mitigations.

There are roughly four ways this can go:

1. Regardless of whether the CVE notification is justified or not, there is a simple solution by updating a possibly vulnerable dependency to a newer safe version without any compatibility issues.
2. The CVE finding can be marked as a *false positive*, suppressing future notifications.
3. The CVE finding is correct, but an upgrade to a newer safe version breaks compatibility with existing software and requires a significant investment to apply.
4. The CVE finding is correct, but there is no newer safe version available of the vulnerable dependency, for example because it's no longer actively maintained.

In practice, most developers can be convinced to structurally apply option 1. Option 2 requires further investigation and should only be applied if the CVE indeed does not affect the software (often, a CVE pertains to a specific use of a library; not to all uses). Developers should resist the temptation of marking a CVE finding as a false positive just to stop nagging notifications. That temptation exists, because options 3 and 4 require a significantly higher investment. It may require reimbursing technical debt before upgrading, dropping a dependency altogether, or investing in forking or contributing to maintenance of the vulnerable dependency.

This is why JDriven recommends developers take the time to seriously understand CVEs, their impact, and possible mitigations. Be aware of technical debt, share knowledge, involve security experts, and get acquainted with dependencies and their maintainers, so we together we can keep our software safe.

GitOps

ADOPT

GitOps is a technique that tries to enhance devops automation with the help of best practices within software development like version control, compliance, collaboration and CI/CD.
Simply put, the technique exists of the usage of git and infrastructure as code, in collaboration with an CI/CD pipeline for everything. This will include **every** change of configuration of a running system

Git tracks every change in the Git history which creates an audit trail. Also, there is a central hub for collaboration through merge requests to implement changes in a compliant way.
Infrastructure as Code will take care of all infrastructure configuration via code. Combined with git you'll have a centralized place that describes your infrastructure.
Git and Infrastructure as Code combined with a CI/CD pipelne will enable automatic deployments of everything without the need of any human intervention.

Currently we see GitOps being used by many companies and it is considered both pleasant and self-evident. We therefore recommend embracing GitOps and we have included it in our technology radar in Adopt.

Inner Source

ADOPT

Inner source refers to applying open source principles within an organization. It is about adopting the collaborative, transparent, and decentralized approach commonly seen in open source projects and applying it to the organization's internal software development. It is an alternative to closed source, where repositories are typically available only to the responsible team.

In an environment where inner source is applied, teams within the company have the ability to openly access code repositories. This promotes collaboration, enhances code analysis, provides the opportunity for contributing code changes via Pull Requests across different projects. It promotes greater transparency, encourages knowledge sharing, and enables developers to work across teams and departments.

Within JDriven, as well as among the clients we work for, we see that these principles are increasingly embraced and that working according to the inner source methodology contributes to higher quality. Additionally, this broader perspective for developers within the organization provides more insights into other people's code, which promotes knowledge sharing.

For these reasons, at JDriven, we are placing inner source in *Adopt* and advocate these principles to our colleagues and clients.

1 TFB NFC

ADOPT

1, TFB, NFC is a work estimation technique that tries to solve the issues surrounding more established planning techniques, such as using story points. The idea is that, as tasks (*backlog items* in Scrum) become more complex, time estimates will become more uncertain, often to a point where these estimates hold little value. Too many of these kind of estimates will make sprint plannings unreliable, and hamper steady progress for the team.

Instead, 1, TFB, NFC takes an opposite view to story points, without going so far as to drop estimations entirely. Each story either is *1 point*, *too frighteningly big (TFB)*, or *no faintest clue (NFC)*. The idea is that only stories in the first category can be worked on, as the scope and solution are clear enough and sufficiently small. Stories that are TFB need to be split up, and NFC-type stories should be explored first, using a spike or a small prototype.

By applying this technique, the focus is on effective planning and refinement, rather than trying to quantify estimations by using bigger numbers. It helps to be familiar with the INVEST principle - where the E intentionally no longer stands for Estimable. A strong indication that a team is ready for this technique, is when there's little variation in the story points assigned to sprint backlog items that the team consider ready to be included into a sprint. By labeling such items as having a size of 1, planning and measuring velocity becomes a simple matter of counting the amount of sprint backlog items. Although we realise that such a classification might be too coarse in some cases, we think its strengths merit moving 1, TFB, NFC to the *Adopt* ring.

15-factor app

TRIAL

In 2011 Heroku published a set of principles for developing SaaS (Software as a Service) applications. This set is known as the 12-factor app. Aside from providing suggestions for the development of services and managing software dependencies, the factors focused heavily on building cloud agnostic applications.

The 15-factor app adds 3 principles to improve quality and maintainability of cloud native applications.

- **API-first:** Cloud native applications run in ecosystems with other services, managed services and facilities from the cloud environment. By focusing on well-defined APIs challenges in integration can be prevented.
- **Telemetry:** In the original 12-factor app "logging" was already a factor. Logging monitors the internal health of the applications. Telemetry is added to also watch the health of the application in its environment; monitoring resource usage, scaling, integration with other application and integration with the cloud environment.
- **Authentication and Authorization:** Cloud Native applications are easily distributed over datacenters around the globe or even different clouds. The reach of such apps is enormous and both the number as well as the different types of consumers increase the complexity regarding Security. Authentication and Authorization draws attention to these concerns during development.

We have put the 15-factor app in *Trial*. The 12-factor app already provides a good list of principles for developing Cloud Native applications, but the 15-factor app adds 3 extra factors which add a lot of value in modern Cloud Native applications.

Eventmodeling

TRIAL

Event Modeling is a method of describing systems using an example of how information has changed within them over time.

An event-centric approach to a business process and describing it in a clear manner helps business, architects and developers to have a deep understanding about the problem to be solved. Besides an event-centric approach, Event Modeling also takes the user experience into account. This combination results in a complete picture of user interaction with the system and how information flows through the system.

At JDriven we have seen that Event Modeling can be used alongside or instead of Big Picture Event Storming, to develop a common understanding of a business domain. It results in a more concrete picture for software implementations.

Sustainable software engineering

TRIAL

Green software is an emerging discipline at the intersection of climate science, software design, electricity markets, hardware, and data center design. Green software is carbon-efficient software, meaning it emits the least carbon possible. Only three activities reduce the carbon emissions of software; energy efficiency, carbon awareness, and hardware efficiency.

A software architecture that takes into consideration carbon efficiency is one where design and infrastructure choices have been made in order to minimize energy consumption and therefore carbon emissions. The measurement tooling and advice in this space is maturing, making it feasible for teams to consider carbon efficiency alongside other factors such as performance, scalability, financial cost and security.

The cloud now has a larger carbon footprint than the aviation sector. We believe that as software engineers and architects we also have a responsibility to reduce this footprint and improve the climate. By making conscious choices, we can also take these steps. In addition, efficient systems have an additional advantage: fewer resources means lower costs.

We are seeing increasing awareness within the industry on this topic, which has led to a shift of Sustainable Software Engineering from *Assess* to *Trial* in our new Tech Radar. This growing focus highlights the importance of green software, emphasizing that it is no longer just a theoretical consideration, but a practical necessity that is being increasingly adopted.

Shape Up

ASSESS

Shape Up describes a way of working to "shape" a solution so it fits within one iteration.

Iterative working is a common way of working within many projects, whereby each iteration consists of an estimated amount of work, usually expressed in Story Points. Shape Up turns this approach the other way around, by "shaping" a solution so it fits within one iteration. Shape Up describes a solution on a high level, without too many details, which must be complete and must have concrete boundaries. High level means a user experience design must be a sketch, complete means pitfalls are taken away and concrete boundaries means there's a clear starting and ending of the solution.

If the solution does not fit into one iteration, it needs to be shaped in such a way that it will fit in one iteration and still has value when delivered.

Distributed Monolith

HOLD

With the rise of cloud infrastructure and virtualization, developers and operations engineers have found a solution for the problems of their *Big Ball of Mud* monoliths. It allows them to cut up applications into microservices, which can offer several benefits over a monolith:

- With proper separation of functionality, every microservice serves a singular purpose, which lowers the *cognitive load*.
- Every part of an application can be individually changed and deployed, so that a functional change only affects a small part of the operational software.
- Infrastructure for parts of the application, that have more capricious performance requirements than other parts, can be scaled in isolation.
- Code repositories can be maintained by and transferred between teams.

However, microservices do have some downsides:

- Developers need to divide their attention between multiple moving parts in a delivery pipeline and software landscape.
- Interaction between parts is typically over HTTP, which presents additional demands for API management and security.

When adopting a microservices-architecture, the benefits won't automatically emerge. There is a risk, that the software succumbs to the downsides of both a *Big Ball of Mud* monolith and microservices, and becomes what is referred to as a *Distributed Monolith*.

There is a risk of ending up with a Distributed Monolith, when an organization decides to cut up an existing monolith into technical parts, for example because existing teams are divided based on technical expertise. A symptom of such a situation is that every functional change requires changes in parts managed by several teams. We call this anti-pattern *tight coupling*. A different situation, that we see occurring more and more often, is when architectural plans demand that every functionality should indiscriminately be built as a separate microservice. In such cases there is a risk, that even the smallest functional change requires a team to make changes in many small microservices that they maintain, which introduces extra work and an increased risk of making mistakes.

JDriven therefore advises companies to be careful when considering employing microservices. One should first and foremost strive for *low coupling, high cohesion*. In this context, the Single Responsibility Principle can come in handy: "Things that change for the same reasons, should be together". A software design method like *DDD*, which starts with finding distinctions in the functional domain, and then modelling that to *bounded contexts* and *context maps*, can further help to make the right cut. And if all above-mentioned possible benefits of microservices aren't desired, it's worth considering starting with building a *modular monolith* instead of microservices.

Platforms

Assess
10. AsyncAPI
11. OpenFGA

AsyncAPI

ASSESS

The AsyncAPI specification, akin to the OpenAPI specification for RESTful services, provides a standardized way to describe event-driven and message-driven APIs. This specification facilitates the design, documentation, and consumption of asynchronous APIs, enabling developers to define message formats, channels, and publish-subscribe patterns in a language-agnostic manner. As organizations increasingly adopt event-driven architectures, the Async API specification is a good tool for maintaining consistency and interoperability across distributed systems. The AsyncAPI provides a standard and we can generate documentation and code from it.

For projects that use event-driven APIs the AsyncAPI specification may be good fit and we recommend assessing it.

OpenFGA

ASSESS

OpenFGA is an authorization solution that offers a flexible approach to access control. OpenFGA supports standard role based access control (RBAC), but more interestingly also has the capability for fine-grained access control.

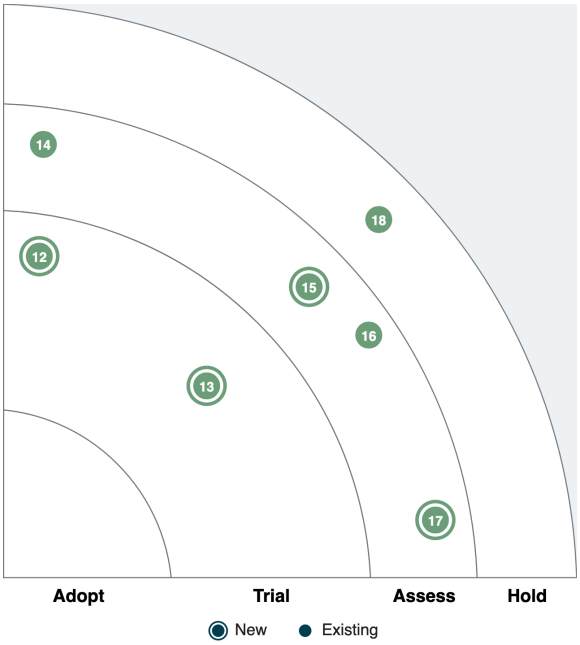
OpenFGA makes use of a relationship-based access control (ReBAC) model, allowing authorization definitions to be based on a user's relationship with a resource, not just their role. This enables fine-grained authorization, which offers a major advantage over RBAC which can become cumbersome for intricate scenarios. These relationship models can be deployed automatically and managed through code instead of manually. The modeling language that is used is expressive enough to handle a wide range of authorization needs while supposedly remaining understandable for non-programmers. However, OpenFGA's rule modeling language introduces a new concept for developers, which could result in a learning curve when switching to OpenFGA's approach.

Due to its backing by the Cloud Native Foundation it has good support and multiple SDKs for integration. While actively maintained, there might be less documentation or community support available due to it being a relatively new project compared to some established access control solutions.

OpenFGA was designed for performance and to efficiently handle authorization checks for large-scale applications with many users and resources. Implementing OpenFGA adds another layer to the application architecture, and it is unknown how its performance compares to traditional access control platforms. If the authorization needs are simple, the added complexity might outweigh the benefits.

The combination of a new authorization concept as well as a comprehensive platform warrants OpenFGA being in the *assess* ring.

Tools



Trial
12. Bruno
13. Devoxx Genie

Assess
14. Claude 3
15. CRaC
16. GitHub Codespaces
17. warp.dev

Hold
18. OpenTofu

Bruno

TRIAL

Bruno is an open-source API client similar to well-known alternatives like Postman, Insomnia and Hopscotch. According to Bruno's manifesto, API collections should be co-located within the source code repository, serving as a living set of examples on how to use the API, and with that turning it into "living documentation". Collections become first-class citizens, co-located with related information and easily version controlled.

Bruno's UI is comparable to Postman, offering a known environment if you are used to other API clients. It supports obviously various HTTP methods, request body formats, and response parsing. Bruno has plugins for Visual Studio Code, and support for other IDEs are in development. Key features include environment variables, collection variables, and the ability to run pre-request and post-request scripts.

From a developer's perspective, Bruno is a fresh alternative to existing commercial products, without needing a required account and storing and sharing collections in a cloud environment. By placing API collections together with source code, separate import/export processes are not needed as well. This automatically improves collaboration within development teams, and documentation evolves with the codebase.

While Bruno lacks some advanced features found in the more well-known API platforms, its focus on ease of use and integration with development workflows makes it an interesting choice. And because Bruno is open-source, community-driven improvements and customizations are also possible (let's hope the IntelliJ plugin is available at the moment you are reading this). We would definitely advise you to give Bruno a try, which is why we have added Bruno in Trial.

Devoxx Genie

TRIAL

Devoxx Genie is a fully Java-based large language model (LLM) Code Assistant plugin for IntelliJ IDEA, designed to integrate with local LLM-providers. Some well-known tools are already supported: Ollama, LMStudio, GPT4All, Llama.cpp and Exo. It can also connect to cloud-based LLMs (OpenAI, Anthropic, Mistral, Groq, Gemini, DeepInfra, DeepSeek, and OpenRouter). Some of the key features are a local chat history, a token cost calculator, web search and code highlighting. The flagship feature is the ability to add your whole project or package to the context of your prompt when asking questions to an LLM of your choice.

Running online LLMs usually involves sending data to servers managed by these LLM-providers. At the projects where we as JDriven are involved in, cloud-based AI coding assistants are hardly used because there is insufficient control over what happens to sensitive data. That is why we put Devoxx Genie in the *trial* ring. The ability to still leverage the power of coding assistants locally, within your IDE, is something many developers want. We believe that, while Devoxx Genie is still in development and has its quirks, it is a good substitute for cloud-based solutions.

Claude 3

ASSESS

Claude 3 is a foundational AI service from Anthropic, an offspring of OpenAI, with a strong focus on user safety. The main driver behind their model is Constitutional AI. This system tries to optimize helpfulness and minimize harmfulness in responses. Constitutional AI uses a set of rules to prevent harmful responses that are used to train a metamodel. The metamodel is then used to interfere with the training of the regular model, so that it learns not to generate content that is considered harmful. The metamodel does this by critiquing the regular model's output according to its constitution iteratively, until the outcome contains no more violations. To validate, the model is also checked against a large corpus of malicious input prompts.

We choose to put the use of Claude in the *assess* ring. Currently, the web UI cannot be used from inside the EU, making it difficult to easily experiment with the technology. Also, the choice of its constitutional rules, while overall sensible, still introduce a bias in the type of responses it generates. These responses, while universal, might not always align with specific cultural patterns. Furthermore, there is a fine line between harmlessness and evasiveness in responses. In Anthropic's research papers, this seemed to be handled reasonably well, but users are advised to draw their own conclusion.

CRaC

ASSESS

CRaC, Coordinated Restore at Checkpoint makes it possible to take a snapshot of a running JVM and store it to disk. This snapshot can then be used to start a JVM, resulting in massive startup performance gain. It offers an alternative to native compilation in terms of startup time but retains the flexibility of the JVM, which could be a middle ground for some more complex applications. The Spring Framework saw the potential and has fully integrated with CRaC. You do have to take into account that any value (secrets and such) seen by the JVM will be included in the snapshot.

We think it's a promising addition to the JVM which can provide quick gains to startup times for more complex microservices with big variation in load, and therefore we put CRaC in the *Assess* ring.

GitHub Codespaces

ASSESS

GitHub Codespaces is a Cloud Development Environment (CDE) that provides virtual machines for developers to work on. It is built on top of devcontainers, which uses a Docker image that contains the required development toolchain. Developers can use their own local IDE, or a web-based variant. It prevents reproducibility issues between developers because of diverging developer environments. Using a common toolchain, configuration and platform, onboarding new team members should become less time-consuming.

GitHub Codespaces has first-class support for Visual Studio Code, but support for the JetBrains IDEs is still lagging, and requires JetBrains Gateway.

We place GitHub Codespaces in the *assess* ring, as it augments, rather than replaces current work processes. That makes trying it out a low-risk exercise. We believe that standardizing development environments can aid in developer productivity. A minor drawback is that a stable internet connection is required for a satisfying developer experience.

As it is a GitHub service, it works best with code repositories already hosted on GitHub itself. There are also costs associated with using the virtual machine time, as well as storage costs for a codespace while it exists. Additionally, there might be difficulties with integrating on-premise services for e.g. testing purposes, requiring a VPN to do so.

Lastly, there are also similar offerings from other companies, that might provide a better match for certain use cases.

warp.dev

ASSESS

Warp is a terminal application that enhances the command-line interface experience for developers. The application provides nice command-line completion, an integrated command palette, and a block-based output system for improved result visualization and interaction. But the most interesting feature is the ability to use natural language to generate shell commands using artificial intelligence. Also error output from a shell command can be input for the AI to come up with a solution.

Warp provides a free plan with a limited number of AI calls per month. To have unlimited AI calls, you can upgrade to the paid plan. Warp also offers an option to pay for a team, with extra options to collaborate with other developers. An enterprise version is also available, where it is possible to use a private LLM.

The shell is a very powerful tool for developers, but it can be hard to remember all the commands and options. Warp makes the shell easier to use by providing a natural language interface. A developer doesn't have to leave the shell to get all the information they need, which is very powerful. As it is still in beta, we've added it to *Assess* in our Tech Radar.

OpenTofu

HOLD

HashiCorp has adjusted the source code license of its products, like Terraform, to a Business Source License. This has led to concern among Terraform users, who fear the ecosystem of tools will suffer from this. On August 23rd 2023 a fork of the Terraform source code was made, and dubbed OpenTofu. The goal is for this to continue existing as an open source project under control of the Linux Foundation.

However, closer inspection of HashiCorp's explanation of their license shows, that this change to a BSL only affects companies that offer commercial products that use HashiCorp products to compete with HashiCorp products. For the overwhelming majority of Terraform users, that is not the case. We therefore see no reason for them to switch to a Terraform alternative like OpenTofu, which is why we put it on *Hold*.

In April 2024, IBM announced that HashiCorp will continue under the umbrella of IBM by the end of the year. Although this may cause interest in OpenTofu to spike temporarily, we do not anticipate any hindrances for existing Terraform users as a consequence of this acquisition.

Commit.
Develop.
Share.