JDriven Tech Radar Voorjaar 2024

Onze kijk op recente ontwikkelingen in het veld

9e editie

Commit to know.

Develop to grow.

Share to show.

jdriven

Commit. Develop. Share.

Introductie

Onze ervaren specialisten werken dagelijks mee aan tal van softwareprojecten in Nederland en zijn betrokken in wereldwijde community's. Halfjaarlijks komen wij vanuit JDriven bij elkaar om te bespreken wat wij aan nieuwe trends en ontwikkelingen zien. Deze trends proberen wij te vangen in een technologieradar. Elke editie van de radar laat verschuivingen zien t.o.v. een vorige editie. Een verschuiving kan betekenen dat wij een technologie interessanter zien worden waar van toepassing, of juist minder geschikt ongeacht de toepassing. Indien een trend niet meer voorkomt in een latere editie, dan zijn er geen nieuwe ontwikkelingen en/of ervaringen die ons eerdere beeld zouden hebben bijgesteld. In dit document willen we toelichten welke verschuivingen we de afgelopen periode hebben waargenomen, om op basis daarvan weer richting te geven aan wat wij inzetten en aanraden.

Tech Radar

Het idee voor het opstellen van een Tech Radar komt voort vanuit Thoughtworks. Zij dragen al langer periodiek met een radar hun visie uit op nieuwe trends en ontwikkelingen. Bovendien raden zij iedereen aan een eigen radar op te stellen [https://www.thoughtworks.com/radar/byor].

Bij JDriven onderschrijven we dat. Het opstellen van een Radar is een leerzame en waardevolle ervaring waarin onderling kennis wordt gedeeld en een technisch bewustzijn wordt gecreëerd. Wij geloven erin dat specialisten zelf in staat moeten zijn om het gereedschap voor hun werkzaamheden samen te stellen. Met het opstellen van een radar faciliteer je discussies over technologie, om als organisatie de juiste balans te vinden in wat voor risico's en voordelen innovatie kan opleveren. Wij kunnen je helpen dit op te starten binnen je organisatie. Laat je teams elkaar inspireren tot innovatie en gezamenlijk komen tot een set aan technologieën en technieken die de ontwikkeling in je bedrijf versnellen.

Indeling

Een radar bestaat uit kwadranten en ringen, met daarbinnen blips om interessante technologieën en technieken aan te duiden.

De kwadranten verdelen de verschillende onderwerpen in categorieën.

- **Talen & frameworks** die je ondersteunen bij de ontwikkeling van software
- **Platformen** waarop je software kunt uitvoeren
- **Technieken** die je helpen betere software te maken
- **Tools** ter ondersteuning van je ontwikkel- en delivery-proces

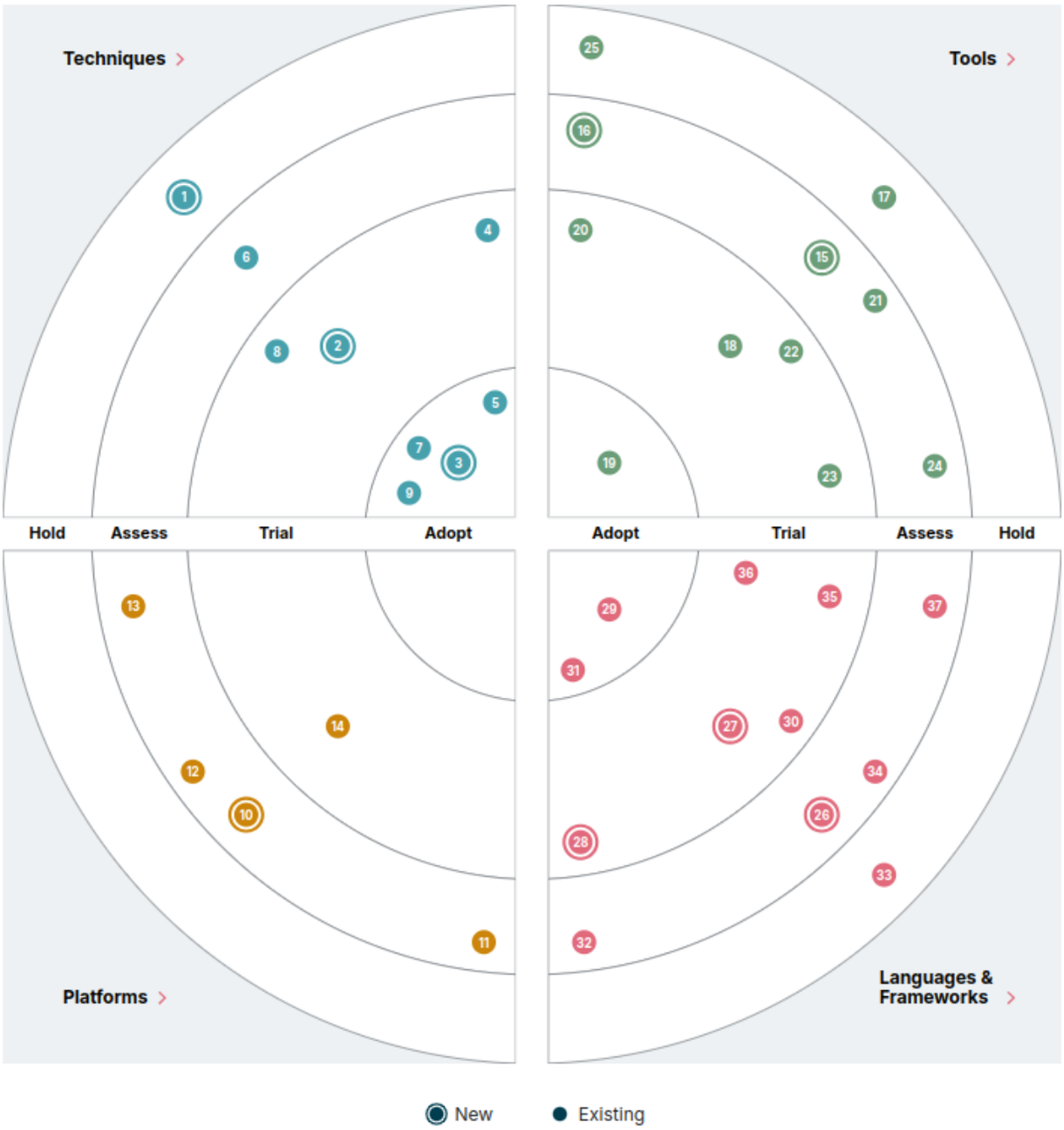
De ringen in elk kwadrant geven aan in welk stadium van adoptie wij denken dat die technologie zich bevindt.

- **Adopt** → Wij raden sterk aan deze technologie te gebruiken, waar van toepassing.
- **Trial** → Interessant om alvast ervaring mee op te doen (in een project dat het risico kan dragen).
- **Assess** → Goed om beter te begrijpen en toekomstige impact in te schatten, maar nog niet om toe te passen.
- **Hold** → Niet (meer) gebruiken.

In de volgende secties zullen we onze kijk op de recente ontwikkelingen per kwadrant toelichten.

Commit. Develop. Share.

© 2024 JDriven

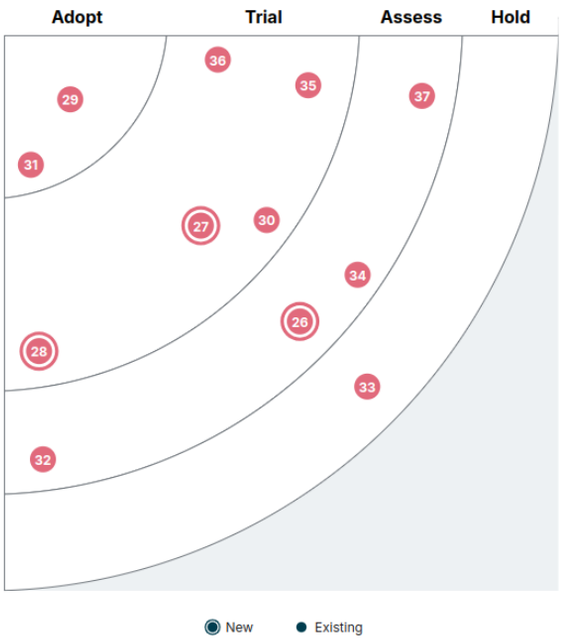


Commit. Develop. Share.

Tools	Talen & frameworks
ADOPT Vector Databases **TRIAL** ChatGPT jlink Maven Build Cache Extension Qodana **ASSESS** Claude 3 GitHub Codespaces Konsist Thunderdome Dev **HOLD** Diffblue Cover JobRunr	**ADOPT** Quarkus Structured concurrency **TRIAL** Backstage HTMX ktor Langchain4j No frameworks **ASSESS** Jakarta 11 Jetbrains compose Terraform CDK Vector API **HOLD** OpenTofu
Platforms	**Technieken**
ADOPT - **TRIAL** Cloud events **ASSESS** Azure container apps eBPF Moderne platform OpenFGA **HOLD** -	**ADOPT** Automatic merging of dependency updates CVE awareness Dev Ex Prompt Engineering **TRIAL** 1 TFB NFC Inner Source Unit testing voor alerting **ASSESS** Sustainable software engineering **HOLD** Distributed Monolith

Commit. Develop. Share.

Talen & frameworks



Assess

- 26. Vector API
- 32. Jakarta 11
- 34. Terraform CDK
- 37. Jetbrains compose

Trial

- 27. No frameworks
- 28. ktor
- 30. Langchain4j
- 35. HTMX
- 36. Backstage

Adopt

- 29. Structured concurrency
- 31. Quarkus

Hold

- 33. OpenTofu

Quarkus

ADOPT

Quarkus [https://quarkus.io/] is een fijnwerkend framework voor developers die Context en Dependency Injection (CDI) en MicroProfile gewend zijn. Volgens hun eigen site is het: "A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM, crafted from the best of breed Java libraries and standards." Het is vergelijkbaar met andere frameworks zoals Spring Boot, en is met name geschikt om serverapplicaties te bouwen.

Quarkus is ontwikkeld om het leven van developers gemakkelijk te maken. Er is weinig configuratie nodig om ermee te kunnen werken. Het framework biedt ondersteuning om eenvoudig in development-, test- of productiemodus te draaien. Quarkus gaat efficiënt om met geheugen. Met behulp van extensies is de functionaliteit eenvoudig uit te breiden. Het is een volwassen product geworden (inmiddels is versie 3 al uit). Quarkus biedt standaard ondersteuning voor het bouwen van Docker containers, zodat je die in Kubernetes of OpenShift kunt draaien.

Het gebruik van Smallrye Mutiny als Reactive Streams library maakt het mogelijk om imperatieve stijl naast reactive stijl te laten bestaan. Dit stelt developers in staat om laagdrempelig en stapsgewijs een applicatie volledig non-blocking te maken.

Testen wordt ook uitgebreid ondersteund. Met de `@QuarkusTest` annotatie kun je gemakkelijk integratietests opnemen in de build pipeline, waarbij ook test containers worden ondersteund. Het is ook mogelijk om jouw applicatie lokaal in developer mode te starten. Met de tool Skaffold [https://skaffold.dev/] (geen standaard onderdeel van Quarkus) kun je daarnaast snel een nieuwe deployment testen in een lokaal draaiend Kubernetes-cluster (bv. minikube of Rancher Desktop).

Het is zeer geschikt voor HTTP RESTful web development en integreert goed met gangbare message

Commit. Develop. Share.

protocols, authentication protocols, datastores en platforms. Dankzij de rijkelijk aangeklede handleidingen is het bovendien relatief eenvoudig om Quarkus naar behoefte uit te breiden.

Het maken van native executables is al lang niet meer een onderscheidende factor voor Quarkus, maar door de brede ondersteuning en uitgebreide documentatie heeft Quarkus ons bewezen een serieuze speler te zijn in het aanbod van JVM CDI Frameworks. In het afgelopen jaar werd Quarkus bij diverse klanten succesvol ingezet (o.a. financiële sector). Hiermee heeft het zich als stabiel framework bewezen, waarmee Quarkus in deze editie de *Adopt* status heeft verkregen.

Structured concurrency

ADOPT

Structured concurrency is een programmeerparadigma dat een gestructureerde aanpak biedt voor het parallel uitvoeren van taken. Het introduceert een aantal tools en abstracties om parallel uitgevoerde code te vereenvoudigen, met als doel de leesbaarheid, onderhoudbaarheid en het debuggen van code te verbeteren. In tegenstelling tot traditioneel thread management, zorgt structured concurrency ervoor dat alle gelijktijdige taken expliciet worden gescopet en beheerd, waardoor het eenvoudiger wordt om te redeneren over hun levenscyclus. Dit paradigma helpt ontwikkelaars om veelvoorkomende valkuilen van verkeerd thread management zoals resourcelekken en onvoorspelbaar gedrag te voorkomen, door een meer georganiseerde en deterministische manier te bieden om met concurrency om te gaan.

Wij raden aan om, waar mogelijk, gebruik te maken van structured concurrency bij het ontwikkelen van applicaties. In bijvoorbeeld Kotlin raden wij aan om gebruik te maken van Kotlin Coroutines bij code die baat heeft bij parallele uitvoering. In Java 21 is structured concurrency nog in preview. Toch raden we aan om deze nieuwe functie alvast uit te proberen en er vertrouwd mee te raken, zodat het kan worden gebruikt zodra het gereleaset wordt.

Backstage

TRIAL

Backstage [https://backstage.io/] is een open source developer portal. Het is door Spotify gemaakt en gedoneerd aan de Cloud Native Computing Foundation (CNCF). Het is in 2020 gepubliceerd en is dus een relatief nieuw platform.

Het doel van Backstage is het bieden van een centrale plek voor software, tooling en documentatie. Het biedt daarbij de mogelijkheid om templates te schrijven, waarmee snel een nieuw project is op te starten.

Backstage heeft veel integraties en plugins voor verschillende tools om dit doel te realiseren. De sterke punten van Backstage komen het beste naar voren als het bij grote projecten wordt gebruikt.

Voorbeelden van grote bedrijven die Backstage gebruiken:

- LinkedIn
- Vodafone
- Lego
- Siemens

Backstage staat in *Trial*, want het nut van een developer portal is niet altijd duidelijk en kan verschillen per project. De cultuur en omvang van een bedrijf zijn belangrijke factoren.

Commit. Develop. Share.

HTMX

TRIAL

htmx [https://htmx.org/] is een JavaScript-library gericht op het bouwen van een hypermedia-driven application [https://hypermedia.systems/hypermedia-systems/] (HDA). HDA en htmx zijn een reactie op de zware Single Page Application JavaScript-frameworks als React en Angular, en leggen de aandacht terug op de kracht van HTTP, HTML, hyperlinks en REST, samengevat als *hypermedia architecture*. De opkomst van SPA's kan verklaard worden door het feit dat het moeilijk was om in web 1.0 vlotte UI-overgangen en interactiviteit te bouwen zonder trage en merkbare page reloads. HTML biedt nog altijd slechts 2 HTTP methods (GET en POST) en 2 HTML-elementen (anchor en form) voor interactiviteit met server content.

De oplossing, die SPA-frameworks voor dit gebrek aan HTML-interactiviteit hebben geïntroduceerd, heeft echter wat nadelen: ze verandert RESTful backend-API's in data-API's die JSON serveren, maakt JavaScript leidend om HTML te tonen, en introduceert een complexe frontend-architectuur leunend op vele JavaScript-libraries voor schijnbaar eenvoudige zaken zoals URL's, web content indexering (SEO), de back button, caching, progressive enhancement en loose coupling. htmx gebruikt JavaScript om de bovengenoemde tekortkomingen van HTML te adresseren, om daarna uit de weg te gaan en websites weer hypermedia-gedreven te kunnen maken, en daarmee de tekortkomingen van SPA's te vermijden.

We zien een toename van interesse in htmx en HDA onder full-stack developers, en zelfs onder frontend developers die hun carrière begonnen zijn met het toepassen van SPA-frameworks en de genoemde problemen herkennen. JDriven adviseert om bij het opzetten van een frontend-architectuur te kijken of een SPA-architectuur wel de beste oplossing is, of dat HDA en htmx een goede fit zouden kunnen zijn. Let daarbij op de toepasbaarheid van tools in het frontend-ecosysteem zoals design systems, Tailwind CSS en Playwright. Omdat htmx in feite een terugkeer naar serverside rendering is, is het ook belangrijk om de ontwikkelingen op het gebied van Edge Computing, die ook spelen op het gebied van SPA's, mee te nemen in de overweging.

ktor

TRIAL

Ktor [https://ktor.io/] is een framework voor Kotlin om snel webapplicaties mee te bouwen. Het is een relatief lichtgewicht framework. Ktor ondersteunt standaard alleen de meer moderne en veelgebruikte functionaliteiten voor webservices, zoals routing, templating en serialisatie. Door het gebruik van de Coroutines bibliotheek van Kotlin is het gemakkelijk om services te maken die onder belasting goed blijven functioneren. Ook is er standaard ondersteuning voor meerdere HTTP-servers.

We hebben er voor gekozen *Ktor* in de trial ring te plaatsen. Het framework bestaat al een geruime tijd (negen jaar), en heeft inmiddels aardig wat tractie binnen de gemeenschap opgebouwd. De ondersteuning voor externe afhankelijkheden en opties is echter beperkt in vergelijking met meer gevestigde frameworks als Spring. Ontwikkelaars moeten daarom meer code schrijven om deze afhankelijkheden binnen hun project te gebruiken.

Als de benodigdheden voor een service grotendeels door Ktor vervuld kunnen worden, is het een interessante optie om in korte tijd een webservice op te zetten.

Commit. Develop. Share.

Langchain4j

TRIAL

Langchain4j [https://github.com/langchain4j/langchain4j] is een framework om eenvoudig applicaties te ontwikkelen met AI large language models (LLM).

Deze LLM's kunnen verscheidene taken met betrekking tot natuurlijke taalverwerking uitvoeren, zoals vertalingen maken en content samenvatten. Ze kunnen ook generatieve taken als contentcreatie uitvoeren en zijn ook extreem nuttig om vragen te beantwoorden.

Met dit framework kunnen LLM-providers eenvoudig worden uitgewisseld omdat er een abstractielaag tussen zit. Implementaties voor OpenAI GPT-4, Azure OpenAI en Google Vertex AI zijn beschikbaar, en ook lokale modellen kunnen gebruikt worden met Ollama [https://ollama.ai/]. Het is een tegenhanger van de Langchain [https://www.langchain.com/] library voor Python en Javascript. De Java-implementatie is volop in ontwikkeling en er komen steeds nieuwe functies bij.

Indien je een Java-applicatie wilt uitbreiden met AI-functies, dan is het bekijken van dit framework de moeite waard. Er kan dan ook eenvoudig gewisseld worden tussen en getest worden met verschillende modellen.

Wij bij JDriven geloven dat de abstractielaag van dit framework leidt tot flexibelere software zonder "vendor lock-in", wat een voordeel is, gezien het feit dat alle features en modellen nog volop in beweging zijn. Als er bijvoorbeeld wegens privacy of betere resultaten van model gewisseld moet worden, is dit eenvoudig te realiseren.

No frameworks

TRIAL

Vrijwel iedereen gebruikt frameworks voor het bouwen van applicaties. Je kan er vaak snel en relatief eenvoudig een applicatie mee bouwen, omdat een framework typisch de volgende zaken eenvoudig maakt (vaak mbv. annotaties en conventies):

- configuratie via properties laden
- Dependency Injection
- ingebouwde REST server
- ingeboude JSON marshalling/unmarshalling
- ingebouwde REST client
- ingebouwde ORM
- ingebouwd transactiemanagement
- ingebouwde log library

Maar frameworks hebben, veelal door hun omvang, ook nadelen. Dat merk je vaak pas na enige tijd, als je bijvoorbeeld een extra library (dependency) wilt gebruiken, en deze conflicteert met een van de (transitieve) libraries van het framework, of wanneer je het framework of zo'n library upgradet naar een nieuwere versie. Plotseling kunnen er onbegrijpelijke fouten optreden zoals ingewikkelde versieconflicten van transitieve dependencies, niet meer werkende annotaties, vage runtime errors waardoor bijvoorbeeld de tests of de applicatie zelf niet meer opstarten, etc. Het kost vaak veel moeite om de oorzaak te achterhalen en op te lossen, omdat het framework zo groot en complex is en veel 'automagisch' via annotaties en/of conventies (die je niet allemaal weet) doet. Niet zelden lijkt alle tijdwinst door het gebruik van een framework daardoor in één klap verloren.

Waarom bouwen we een applicatie niet gewoon zonder frameworks, en met alleen de lichtgewicht libraries die we nodig hebben (cherry-picking)? Dat is heel goed mogelijk met bijvoorbeeld de

Commit. Develop. Share.

volgende aanpak:

- Configuratie laden via properties files kan gewoon met de standaard Java `Properties` class. Daar heb je geen library voor nodig.
- Dependecy Injection kun je ook heel goed zelf doen middels constructor injection. Daar heb je geen IoC (Inversion of Control) en dus ook geen framework voor nodig.
- Een lichtgewicht HTTP-server zoals Jetty [https://eclipse.dev/jetty/] of Undertow [https://undertow.io/] is voldoende om tekst strings te kunnen uitwisselen over HTTP. En met een library als Jackson [https://github.com/FasterXML/jackson] kun je zelf JSON strings marshallen/unmarshallen.
- Een eenvoudige HTTP-client zoals OkHttp [https://square.github.io/okhttp/] kan gebruikt worden om op soortgelijke manier een REST client te bouwen.
- Hibernate [https://hibernate.org/] is ook prima als losse library te importeren als je een ORM nodig hebt.
- En Hibernate biedt ook transactiemanagement via `EntityManager.getTransaction().begin()` en `EntityManager.getTransaction().commit()`.
- Er zijn meerdere log libraries beschikbaar om los te gebruiken, bijvoorbeeld SLF4J [https://www.slf4j.org/].

Deze aanpak vergt iets meer code dan bij een framework waar je het één en ander cadeau krijgt. Een groot voordeel is echter dat je veel minder transitieve dependencies en daarmee complexiteit binnenhaalt. Tevens is de code meer expliciet, met minder 'automagische' annotaties en conventies. Dat maakt het debuggen een stuk eenvoudiger. Uiteindelijk is deze opzet daarmee eigenlijk veel eenvoudiger dan met een groot log framework waar van alles in zit en waarvan je lang niet alles gebruikt (YAGNI - You Aren't Gonna Need It). Ofwel een goed voorbeeld van KISS (Keep It Simple Stupid). Als een framework niet vereist is, kan het dus de moeite waard zijn om te kiezen voor een no-framework opzet.

Jakarta 11

ASSESS

Jakarta 11 [https://jakarta.ee/specifications/platform/11/] is de nieuwste (aankomende) versie van Jakarta EE. Deze nieuwste versie heeft een aantal interessante features zoals Virtual Threads en Jakarta Data (vergelijkbaar met Spring Data). Het ondersteunt alleen Java 21 en het verwijdert een aantal oude specificaties. Dit alles zorgt ervoor dat dit een interessante versie van Jakarta EE wordt wat het volgens ons de moeite waard maakt om te beoordelen.

Jetbrains compose

ASSESS

JetBrains zet met Compose Multiplatform in op het ontwikkelen van businessfunctionaliteit in één codebase, met een op diverse platforms toegespitste UI. Het combineert de krachten van Jetpack Compose en Kotlin Multiplatform om ontwikkeling voor servers, Android, iOS, web (via Kotlin/Wasm), en desktop (Linux, macOS, Windows) vanuit een enkele codebase mogelijk te maken.

Voor developers die reeds bekend zijn met de wereld van Kotlin en/of Android development, is Compose Multiplatform veelbelovend. Het biedt de mogelijkheid om business logic en UI-code bruikbaar te maken voor alle genoemde platforms. De vraag blijft altijd, of de effectiviteit van het werken met één codebase opweegt tegen de extra lagen van abstractie, die nodig zijn om te komen tot een grote gemene deler in de code. Ook is het zo, dat de iOS-functionaliteit nog in alpha-fase is, en Kotlin/Wasm voor web nog experimenteel. Maar het biedt een interessante optie voor organisaties die willen investeren in business logic, een homogene developer skillset en een uniforme UI over meerdere platforms.

© 2024 JDriven

Terraform CDK

ASSESS

Terraform maakt gebruik van HashiCorp Configuration Language (HCL) om op een cloudagnostische wijze de infrastructuur van een applicatie te beheren. HCL is een declaratieve taal om cloudresources te beschrijven. Echter betekent het gebruik hiervan dat developers een nieuwe taal moeten leren om met deze tooling overweg te kunnen. Door gebruik te maken van CDK for Terraform (CDKTF) kan men nu ook aan de slag met Java, C#, Python, TypeScript of Go voor het beheren van infrastructuur.

CDKTF verdient een plekje op de Tech Radar omdat Terraform onderhand een bewezen technologie is, waar deze toevoeging een aantal voordelen biedt ten opzichte van de conventionele aanpak. Zo krijgt men code completion en IDE-support cadeau, integreert de infrastructuurdefinitie beter met bestaande tooling zoals codeformatters en linters, en wordt het laagdrempeliger voor developers om Terraform te gaan gebruiken in hun projecten.

Op 21 augustus 2022 heeft HashiCorp de tooling als 'production ready' aangemerkt, met de kanttekening dat de tooling nieuw is en daarom nog niet volledig volwassen. Daarnaast is de documentatie omtrent bepaalde onderwerpen benedenmaats, en introduceert de CDK een aantal nieuwe concepten en dus wat extra complexiteit. Al met al een interessante ontwikkeling om in de gaten te houden, zeker als Terraform al gebruikt of overwogen wordt.

Vector API

ASSESS

De Vector API is een incubator API om voorspelbaar gebruik te maken van SIMD (single instruction, multiple data) instructies binnen het Java platform. Deze instructies opereren op meerdere data-primitieven tegelijk, tot wel 16 elementen op moderne processoren. Hierdoor kan een aanzienlijke snelheidswinst worden behaald, wanneer dezelfde mathematische operatie moet worden uitgevoerd op een groot aantal data-elementen. De Vector API is onder Java 16 als incubator vrijgegeven. Wanneer project Valhalla gereed is, zal de Vector API in de Java mainline geïntegreerd worden.

Onder bepaalde omstandigheden kan de HotSpot VM in OpenJDK zelfstandig SIMD instructies gebruiken voor bepaalde operaties. Dit gedrag kan echter onvoorspelbaar zijn, waardoor hier niet goed op te vertrouwen is. Alleen als aan alle (impliciete) randvoorwaarden voldaan is, zal de JIT compiler de auto-vectorisatie uitvoeren. Een ander voordeel van de Vector API is, dat deze gebruikt maakt van Intel's SVML bibliotheek, die SIMD implementaties van veelgebruikte wiskundige functies bevat.

Wij hebben ervoor gekozen om de Vector API in *assess* te zetten. Hoewel dit de achtste incubator zal worden voor de Java 23 release deze herfst, is de API nog steeds niet stabiel verklaard. Het hoofddoel van de API is het oplossen van problemen met performance voor gedeeltes van de code die een sterk numerieke focus hebben. Omdat het gebruik van de Vector API de leesbaarheid van de code niet ten goede komt, is het belangrijk dat deze alleen gebruikt wordt om specifieke performance problemen mee aan te pakken. Tot slot kunnen toekomstige verbeteringen in de JIT ertoe leiden dat de compiler beter in staat zal zijn om auto-vectorisatie toe te passen, waardoor het nut van het gebruik van de Vector API afneemt.

© 2024 JDriven

OpenTofu

HOLD

HashiCorp heeft de sourcecode-licentie van zijn producten, zoals Terraform, aangepast naar een Business Source License [https://www.hashicorp.com/bsl]. Dat heeft onder gebruikers van Terraform geleid tot zorgen, dat het omringende ecosysteem van tools daaronder gaat lijden. Op 23 augustus 2023 is er een fork gemaakt van Terraform, genaamd OpenTofu. Het doel is om dit als opensourceproject te laten voortbestaan onder beheer van de Linux Foundation.

Nadere inspectie van de uitleg [https://www.hashicorp.com/license-faq] van HashiCorp over deze licentievoorwaarden wijst echter uit, dat de wijziging naar een BSL alleen bedrijven treft, die commerciële producten maken die concurreren met HashiCorp's producten, door gebruik te maken van HashiCorp's eigen producten. Dat is voor de grootste groep gebruikers van Terraform niet van toepassing. Daarom zien wij voor hen geen reden om uit te wijken naar een alternatief voor Terraform zoals OpenTofu, wat we daarom op *Hold* plaatsen.

Eind april 2024 kondigde IBM aan [https://www.prnewswire.com/news-releases/ibm-to-acquire-hashicorp-inc-creating-a-comprehensive-end-to-end-hybrid-cloud-platform-302126646.html] dat HashiCorp tegen het eind van 2024 als onderdeel van IBM verder zal gaan. Alhoewel dit tot hernieuwde belangstelling voor OpenTofu zal kunnen leiden, zien we geen reden om aan te nemen dat deze overname bestaande gebruikers van Terraform in de weg zal zitten.

© 2024 JDriven

Technieken

Hold

1. Distributed Monolith

Trial

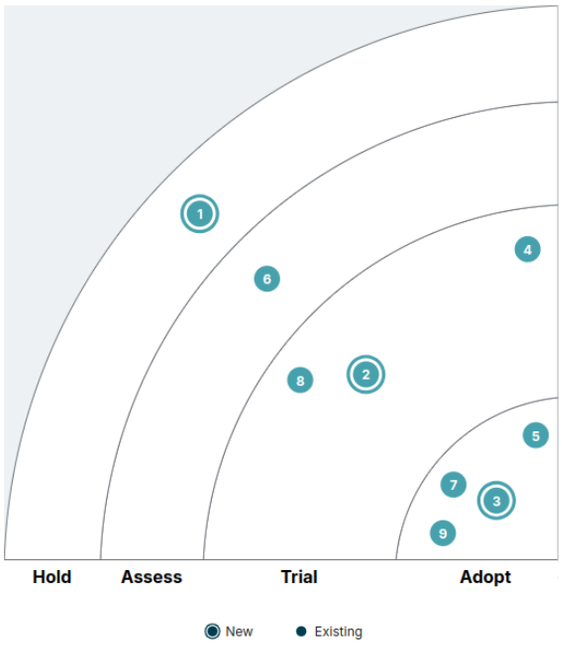
2. 1 TFB NFC
4. Inner Source
8. Unit testing voor alerting

Adopt

3. CVE awareness
5. Dev Ex
7. Automatic merging of dependency updates
9. Prompt Engineering

Assess

6. Sustainable software engineering



Automatic merging of dependency updates

ADOPT

Het automatisch mergen van dependency updates is vandaag de dag steeds belangrijker aan het worden. Tegenwoordig komt het steeds vaker voor dat kwetsbaarheden ontdekt worden die meekomen met dependencies in projecten. Wij vinden dat het automatisch updaten en mergen van de desbetreffende dependencies de juiste manier is, zodat code zo up-to-date mogelijk blijft.

Enkele tools die het automatisch mergen van de dependency updates mogelijk maken zijn onder andere Renovate [https://github.com/renovatebot/renovate] en Dependabot [https://github.com/dependabot/dependabot-core]. Wat deze in essentie doen is het continu scannen van de dependencies in het project en hier merge requests voor aanmaken indien er updates beschikbaar zijn. Ontwikkelaars kunnen deze merge requests handmatig of automatisch mergen.

JDriven kiest ervoor om deze techniek op te nemen in de Tech Radar en in *Adopt* te plaatsen, omdat het teams efficiënt laat werken door het bijhouden van dependency versies (deels) te automatiseren, wat leidt tot kwalitatieve en veilige software.

Commit. Develop. Share.

CVE awareness

ADOPT

Binnen de wereld van software engineering is MITRE [https://www.mitre.org/focus-areas/cybersecurity] vooral bekend van de database met Common Vulnerabilities and Exposures (CVE) [https://www.cve.org/] die het bijhoudt en gratis openbaar maakt. De organisatie MITRE heeft in diverse hoedanigheden een rol gespeeld bij het Amerikaanse ministerie van defensie. Veel development teams richten inmiddels hun CI/CD pipeline in om met automatische tools gebruikte dependencies te analyseren, en deze te vergelijken met kwetsbaarheden die als CVE bekend zijn.

Dit is een goed gebruik. Helaas moeten we constateren dat er onder developers een gebrek aan bewustzijn is over wat deze tools doen, en wat meldingen betekenen. Wanneer een CVE gemeld wordt door een tool als OWASP DependencyCheck [https://owasp.org/www-project-dependency-check/], is het nog altijd aan een developer om deze melding te analyseren, en te bepalen wat de impact en benodigde mitigatie is.

Er zijn grofweg vier soorten vervolgstappen:

1. Ongeacht of de CVE-melding terecht is of niet, is er een simpele oplossing door een mogelijk kwetsbare dependency te upgraden naar een versie die de CVE niet bevat, zonder enige compatibiliteitsproblemen.
2. De CVE-melding mag als *false positive* worden aangemerkt en onderdrukt worden in de scanning tool.
3. De CVE-melding is terecht, maar een nieuwere versie van de dependency in kwestie heeft compatibiliteitsproblemen met de bestaande software.
4. De CVE-melding is terecht, maar er is geen simpele oplossing in de vorm van een dependency upgrade, bijvoorbeeld omdat die dependency niet actief onderhouden wordt.

In de praktijk zijn de meeste developers te motiveren om structureel optie 1 toe te passen. Optie 2 zou alleen moeten volgen uit onderzoek dat uitwijst dat de CVE inderdaad niet van toepassing is (bijv. omdat deze alleen geldt bij specifiek gebruik van de dependency). Developers moeten de verleiding weerstaan, om voor optie 2 te kiezen om van CVE-meldingen af te komen die ze niet direct eenvoudig kunnen oplossen. Die verleiding bestaat, doordat opties 3 en 4 een significant grotere investering van tijd en moeite vereisen. Mogelijk moet technical debt eerst worden ingelost, moet er overgestapt worden op een alternatief voor de dependency, of moet er gekeken worden naar een fork, of naar actieve contributie aan een oplossing voor de CVE in de dependency.

JDriven raadt developers daarom aan om gemelde CVE's serieus te nemen, en de tijd te nemen om ze te doorgronden. Door technical debt bij te houden, kennis te delen, security experts te betrekken, en actief betrokken te zijn bij (interne en externe) software dependencies, houden we onze software veilig.

Dev Ex

ADOPT

DevEx, afgeleid van Developer Experience, is een cruciaal facet van softwareontwikkeling dat de algemene ervaring van ontwikkelaars tijdens het werken aan een project of in een specifieke ontwikkelingsomgeving omvat. Het is gericht op de bruikbaarheid van ontwikkelingstools, efficiënte workflows, duidelijke documentatie, en de tevredenheid en productiviteit van ontwikkelaars.

Binnen de context van softwareontwikkeling is DevEx de sleutel tot het optimaliseren van processen en het bevorderen van een positieve werkomgeving voor ontwikkelaars. Door te zorgen voor intuïtieve tools, gestroomlijnde workflows en effectieve ondersteuning, draagt het bij aan het versnellen van ontwikkelingscycli en het leveren van hoogwaardige softwareproducten. Het is de brug die ontwikkelaars verbindt met de mogelijkheden van technologie, en het stimuleert innovatie in de dynamische wereld van softwareontwikkeling.

© 2024 JDriven

Prompt Engineering

ADOPT

Prompt engineering is een discipline die draait om het vormgeven van nauwkeurige, effectieve aanwijzingen voor door AI aangedreven modellen, zoals GPT-X (Generative Pre-trained Transformer). Deze aanwijzingen sturen de reacties van het AI-model in de richting van het gewenste resultaat. Het is een proces dat experimenteren, iteratie, verificatie en afstemming omvat. Prompt engineering biedt verschillende voordelen die het potentieel hebben om software-engineering te verbeteren:

- Versnelde Ontwikkeling
- Verbeterde Creativiteit
- Consistentie en Betrouwbaarheid
- Leren en Groeien

Prompt engineering stimuleert een cultuur van continu leren en groeien. Ontwikkelaars kunnen modelreacties analyseren, hun aanwijzingen aanpassen en hiervan leren. Het verstrekken van deze inzichten aan de makers van het AI-model kan resulteren in geoptimaliseerde modellen die betere en robuustere resultaten opleveren. Hoewel het gebruik van AI-modellen snelle resultaten kan opleveren, vereist het behalen van hoogwaardige resultaten een investering van tijd, moeite en het verwerven van nieuwe vaardigheden.

1 TFB NFC

TRIAL

1, TFB, NFC [https://estimation.lunarlogic.io#description] is een techniek om planningen binnen het ontwikkelproces te verbeteren. Naarmate taken (*backlog items* in Scrum) complexer worden, nemen de onzekerheden in tijdsplanning eveneens toe, tot een punt waar schattingen weinig tot geen betekenis meer hebben. Een sprintplanning met teveel van dergelijke taken wordt onbetrouwbaar. Een stabiele vooruitgang voor een project wordt hierdoor moeilijk te realiseren.

1, TFB, NFC neemt een diametraal andere houding aan in vergelijking met techieken als story points, zonder helemaal van schattingen af te zien. Elke taak of story krijgt één van de drie labels: *1 point*, *too frighteningly big (TFB)* of *no faintest clue (NFC)*. Enkel taken die 1 story point hebben gekregen, zijn beknopt genoeg. Ontwikkelaars kunnen deze taken direct oppakken en realiseren. Het meten van velocity is dan simpelweg een kwestie van het tellen van het aantal afgeronde taken aan het einde van de sprint. Taken met het label TFB zijn te groot om aan te werken, en dienen in refinement sessies te worden opgesplitst. Taken met het NFC label bevatten nog te veel onduidelijkheden om op te splitsen of in te schatten. Om beter inzicht in deze onduidelijkheden te krijgen, kan het team een spike of een klein prototype bouwen om meer zicht op het probleem te krijgen.

Wij hebben er voor gekozen om 1, TFB, NFC in de *trial* ring te plaatsen. Het maakt het meten van velocity makkelijk, omdat alle taken dezelfde waarde hebben. Teams die veel tijd kwijt zijn aan refinements, of regelmatig sprintdoelen missen, kunnen deze techniek gebruiken om meer realistische doelen te stellen.

Wij realiseren ons dat deze eenvoudige classificatie in sommige gevallen te grof is en het planningsproces kan hinderen. Wij moedigen teams daarom aan om het idee verder te verkennen, en te kijken waar het concept gebruikt kan worden om hun routines te verbeteren.

© 2024 JDriven

Inner Source

TRIAL

Inner source verwijst naar het toepassen van open source principes binnen een organisatie. Het gaat erom de samenwerkende, transparante en gedecentraliseerde aanpak die vaak wordt gezien in open source projecten toe te passen op de interne softwareontwikkeling van de organisatie. Het is een alternatief op closed source, waarbij normaliter repositories enkel beschikbaar zijn voor het verantwoordelijke team.

In een omgeving waar inner source wordt toegepast, zijn teams in staat binnen het bedrijf openlijk code repositories in te zien. Dit bevordert het samenwerken, verbetert de codeanalyse, en geeft de mogelijkheid voor het bijdragen van codewijzigingen via Pull Requests over verschillende projecten heen. Het bevordert een grotere transparantie, stimuleert kennisdeling en stelt ontwikkelaars in staat om over teams en afdelingen heen te werken.

Dit model kan de kwaliteit van software verbeteren, de ontwikkeling versnellen en een cultuur van samenwerking en innovatie binnen de organisatie bevorderen door gebruik te maken van de gezamenlijke expertise en inspanningen.

JDriven staat voor commit, develop, share met de opensourceprincipes in gedachten. Daarom vinden wij dat organisaties moeten overwegen de broncode open te stellen voor interne ontwikkelaars.

Unit testing voor alerting

TRIAL

Binnen het domein van software-observability en -monitoring is het opzetten van robuuste alertmechanismen cruciaal. De onvoorspelbaarheid van events vereist nauwkeurige en responsieve alerts, vaak gebouwd op complexe regels. Echter, de validatie van deze regels voltrekt zich vaak pas in productie wanneer er onverwachts echte situaties voordoen.

De techniek 'unit testing voor alering' is ontworpen om de precisie van de regels en alerts te verhogen door proactieve evaluatie en verfijning. Deze aanpak stelt teams in staat om regels grondig te definiëren en te valideren voordat ze live getriggerd worden door incidenten. Dit verhoogt het vertrouwen in de regels en alerts aanzienlijk.

Het primaire doel heeft een dubbele functie: het verminderen van valse alarmen en ervoor zorgen dat echte problemen snel en nauwkeurig worden benadrukt. Door verschillende scenario's en omstandigheden te simuleren, kunnen teams beoordelen hoe goed hun rules presteren en deze verfijnen voor optimale responsiviteit.

Belangrijke tools zoals Prometheus bieden speciale ondersteuning voor het testen van rules. Door deze preventief te valideren, hebben we een aanzienlijke afname van valse positieven. Hierdoor is een snellere en nauwkeurigere reactie op echte problemen mogelijk.

JDriven is van mening dat teams en organisaties die met rules en alerts werken, zouden moeten beginnen met het toevoegen van unit tests voor alerts.

Sustainable software engineering

ASSESS

Groene software is een opkomende discipline op het snijvlak van klimaatwetenschap, softwareontwerp, elektriciteitsmarkten, hardware en datacenterontwerp. Groene software is CO_2 -efficiënte software, wat betekent dat het zo min mogelijk CO_2 uitstoot. Slechts drie activiteiten verminderen de CO_2-uitstoot van software; energie-efficiëntie, koolstofbewustzijn en hardware-efficiëntie.

Een softwarearchitectuur die rekening houdt met koolstofefficiëntie is er één waarin ontwerp- en infrastructuurkeuzes zijn gemaakt om het energieverbruik en dus de koolstofuitstoot te minimaliseren. De meetinstrumenten en het advies op dit gebied worden steeds volwassener, waardoor het voor teams haalbaar wordt om CO_2-efficiëntie te overwegen naast andere factoren zoals prestaties, schaalbaarheid, financiële kosten en veiligheid.

De cloud heeft nu een grotere ecologische voetafdruk dan de luchtvaartsector. Wij vinden dat we als software engineers en architecten ook een verantwoordelijkheid hebben in het verminderen van deze voetafdruk en het verbeteren van het klimaat. Door hierin bewuste keuzes te maken, kunnen we die stappen ook zetten. Daarnaast hebben efficiënte systemen een bijkomend voordeel: minder resources zorgt voor minder kosten.

© 2024 JDriven

Distributed Monolith

HOLD

Developers en operations engineers hebben met de opkomst van cloud infrastructuur en virtualisatie een oplossing gevonden voor de problemen van voor hun *Big Ball of Mud* monolieten. Het biedt de mogelijkheid voor het opknippen van applicaties in microservices, wat meerdere voordelen kan bieden boven een monoliet:

- Bij goed gescheiden functionaliteit kan een microservice een eenduidig doel dienen, wat de *cognitive load* verlaagt.
- Ieder deel van de applicatie is los aan te passen en uit te rollen, zodat een functionele wijziging slechts een klein deel van je draaiende software raakt.
- De infrastructuur voor applicatie-onderdelen, die meer grillige performance-eisen hebben, is doelgericht op en af te schalen.
- Code repositories zijn onderhoudbaar door en overdraagbaar tussen meerdere teams.

Microservices hebben echter ook een keerzijde:

- Developers moeten hun aandacht verdelen over meerdere bewegende delen in een delivery pipeline en softwarelandschap.
- Interactie tussen de delen vindt plaats over HTTP, wat extra eisen stelt aan API management en security.

Bij een overstap op een microservices-architectuur is het niet vanzelfsprekend, dat de voordelen tot uiting komen. Het risico bestaat, dat de software dan verwordt tot een zogenaamde *Distributed Monolith*, en lijdt onder de nadelen van zowel de *Big Ball of Mud* monoliet als microservices.

Het risico om hierop uit te komen bestaat, wanneer een organisatie besluit een bestaande monoliet op te splitsen in technische componenten, bijvoorbeeld omdat bestaande teams zijn opgedeeld in technische expertise. Een symptoom van zo'n situatie is dat iedere functionele wijziging de technische componenten van meerdere teams raakt. Dit noemen we wel *tight coupling*. Een andere situatie, die we vaak zien voorkomen, is een architectuur-besluit dat ieder functioneel te onderscheiden onderdeel sowieso z'n eigen microservice moet hebben. In dergelijke gevallen kan het zijn, dat een team voor iedere functionele wijziging meerdere kleinere onderdelen moet aanpassen, wat veel extra werk en kans op fouten met zich meebrengt.

JDriven adviseert organisaties om deze redenen om voorzichtig te zijn met het inzetten van microservices, en allereerst te streven naar *low coupling, high cohesion*. In deze context kan het Single Responsibility Principle [https://en.wikipedia.org/wiki/Single_responsibility_principle] goed van pas kan komen: "Wat wijzigt om dezelfde redenen, zou samen moeten zijn". Een software-ontwerpmethode als *DDD*, die begint bij het onderscheiden van functionaliteiten en het modelleren van *bounded contexts* en *context maps*, kan verder helpen om de juiste knip te leggen. Als niet alle bovengenoemde mogelijke voordelen van microservices verlangd zijn, is het het overwegen waard om dit eerst te implementeren in een *modulaire monoliet* in plaats van microservices.

Commit. Develop. Share.

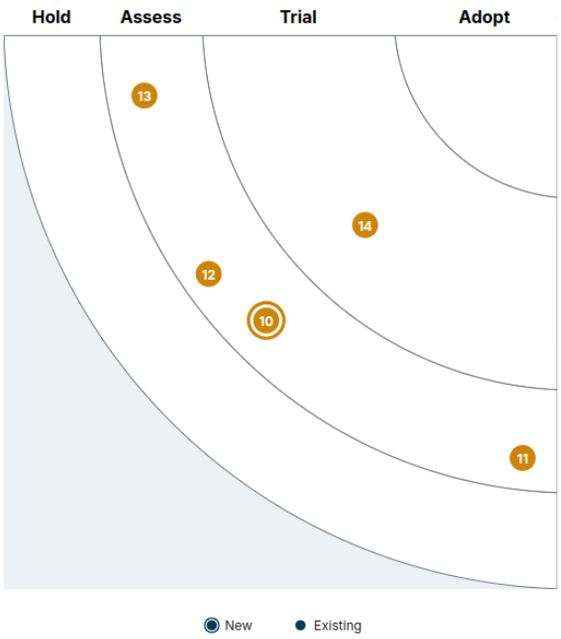
Platforms

Assess

10. OpenFGA
11. Moderne platform
12. eBPF
13. Azure container apps

Trial

14. Cloud events



Cloud events

TRIAL

Cloud Events [https://cloudevents.io/] is een open standaard die is ontwikkeld om de compatibiliteit en samenwerking tussen event-gedreven systemen in de cloud te verbeteren. Het biedt een uniforme manier om events te specificeren, ongeacht de verschillende cloudplatformen, services of systemen die betrokken zijn. Deze standaardisatie stelt organisaties in staat om events te beschrijven in een formaat dat door verschillende cloudproviders kan worden begrepen en verwerkt, waardoor de complexiteit van event-verwerking tussen systemen wordt verminderd.

Cloud Events zijn gebaseerd op een set van metadata en standaardformaten voor events beschreven in JSON of YAML. De metadata van Cloud Events biedt een consistente en gestructureerde manier om informatie over events vast te leggen. Dit omvat details zoals de identificatie van het event, de tijd waarop het plaatsvond en de bron waar het event vandaan komt. Daarnaast biedt Cloud Events de mogelijkheid om zelf optionele metadatavelden naast de standaard metadatavelden vast te leggen.

Cloud Events biedt flexibiliteit en draagbaarheid door een gemeenschappelijke taal te bieden voor het beschrijven van gebeurtenissen, ongeacht de onderliggende technologieën, protocollen en infrastructuur. De voordelen van het implementeren van Cloud Events zijn onder meer een verbeterde interoperabiliteit, verhoogde flexibiliteit en vereenvoudigde integratie tussen verschillende cloudservices en applicaties. Dit stelt ontwikkelaars in staat om gebeurtenisgestuurde architecturen te bouwen die gemakkelijk schaalbaar, flexibel en onderling uitwisselbaar zijn in verschillende cloudomgevingen.

JDriven zet Cloud Events op *Trial* omdat wij denken dat het het onderzoeken waard is voor met name grote enterprise-omgevingen, die baat hebben bij een bredere standaard.

Commit. Develop. Share.

Azure container apps

ASSESS

Azure Container Apps is een nieuwe service op Microsoft Azure die het inzetten en schalen van containerized applicaties in een serverless omgeving vereenvoudigt. Het is gepositioneerd tussen Azure App Service, dat vooral gericht is op eenvoudige webapplicaties, en Azure Kubernetes Service, wat een beheerd Kubernetes-platform is.

Container Apps is gebouwd op Kubernetes, KEDA en Dapr. Hierdoor maakt het event-driven applicatiearchitecturen mogelijk door ondersteuning van geavanceerde autoscaling (inclusief scale-to-zero en traffic splitting), ondersteuning voor langlopende processen en achtergrondtaken. Er is geen directe toegang tot de Kubernetes API's beschikbaar.

Wij denken dat Azure Container Apps een goed alternatief is voor teams die Azure App Service ontgroeid zijn, maar niet hun eigen Kubernetes-cluster willen of hoeven te beheren. Daarom plaatsen wij Azure Container Apps in *Assess*.

eBPF

ASSESS

eBPF (extended Berkeley Packet Filter) is een techniek om programma's in een sandbox binnen de Linux-kernel te draaien, door middel van een Just-in-Time compiler. Dit stelt gebruikers in staat om programma's te schrijven die systeem- en netwerkgedrag observeren en filteren, of extra veiligheidsmaatregelen toepassen, zonder restricties op systeemtoegang. De JIT-compiler zorgt ervoor dat de programma's niet buiten de geprogrammeerde context kunnen opereren. Bovendien verifieert de compiler bepaalde eigenschappen voor veiligheid en "liveness" voordat een eBPF-programma wordt uitgevoerd. Traditioneel wordt dergelijke aangepaste kernelfunctionaliteit verkregen door het schrijven van custom patches voor de Linux-kernel, of het schrijven van kernelmodules. Het nadeel van deze methoden is dat dit erg bewerkelijk is en de stabiliteit van het gehele besturingssyteem in gevaar kan brengen. Bovendien is de in-kernel API van Linux niet stabiel, waardoor dergelijke aanpassingen vaak voor een beperkt aantal kernelversies werken. Daardoor is frequent onderhoud en een grote kennis van de kernel nodig.

We hebben er dit najaar voor gekozen deze techniek op *Assess* te zetten. eBPF lost een beperkte klasse van problemen op, met name voor organisaties die een grote hoeveelheid infrastructuur beheren of speciale eisen aan veiligheid stellen. Ook vereist het schrijven van dergelijke programma's veel specialistische kennis. Een gebruiker heeft daarnaast specifieke rechten op een doelsysteem nodig, en een Linux-kernel waarbij de JIT-functionaliteit beschikbaar is gemaakt. Deze functionaliteit is daardoor meestal alleen beschikbaar voor organisaties die hun eigen systemen beheren; publieke cloudproviders zullen deze functionaliteit veelal uitgeschakeld hebben voor gebruikers om hun eigen infrastructuur te beschermen.

© 2024 JDriven

Moderne platform

ASSESS

Het Moderne Platform is een SaaS-oplossing gebouwd bovenop de open-source software OpenRewrite. Het biedt geautomatiseerde code-refactoringstools waarmee ontwikkelaars met een eenvoudige klik of CLI-opdracht snel en moeiteloos complete codebases kunnen migreren. Dit wordt gedaan met behulp van meegeleverde recepten die bijvoorbeeld Spring Boot-apps kunnen upgraden van versie 2 naar 3.

In onze Tech Radar hebben we eerder OpenRewrite genoemd. Een SaaS-platform dat de sterke punten van OpenRewrite combineert met tools om de ervaring van ontwikkelaars rond het gebruik van OpenRewrite te verbeteren, is iets waar we positief tegenover staan. Concurrenten zoals Snyk, Dependabot en Sonar waarschuwen alleen voor softwarekwetsbaarheden en bevindingen. Wat het Moderne Platform voor ons vooral interessant maakt, is dat het deze problemen ook onmiddellijk oplost, waardoor ontwikkelaarstijd vrijkomt.

Vanwege onze beperkte ervaring met het Moderne Platform zijn we nog niet zeker of het geschikt is voor missiekritieke toepassingen. Wanneer de CLI niet wordt gebruikt, maakt een Moderne-agent verbinding met in-house code- en artefactopslagplaatsen, wat voor sommige organisaties een obstakel kan zijn. Bovendien hangt de effectiviteit van het platform af van het aantal recepten en de taalondersteuning die ze hebben, iets waar ze momenteel hard aan werken om uit te breiden.

OpenFGA

ASSESS

OpenFGA [https://openfga.dev/] is een autorisatie-oplossing die een flexibele benadering van toegangscontrole biedt. OpenFGA ondersteunt standaard role-gebaseerd access control (RBAC), maar biedt interessant genoeg ook de mogelijkheid voor fine-grained toegangscontrole.

OpenFGA maakt gebruik van een relatie-gebaseerd access control (ReBAC), waarmee het mogelijk is om machtigingen te definiëren op basis van de relatie van een gebruiker met een resource, niet alleen op basis van hun rol. Dit maakt een fine-grained autorisatie mogelijk, wat een groot voordeel is ten opzichte van RBAC, dat omslachtig kan worden voor ingewikkelde scenario's. Deze relatiemodellen kunnen automatisch worden geïmplementeerd en via code worden beheerd in plaats van handmatig. De gebruikte modelleringstaal is expressief genoeg om een breed scala aan autorisatiebehoeften aan te kunnen, terwijl het nog begrijpelijk blijft voor niet-programmeurs. De regelmodelleringstaal van OpenFGA introduceert echter een nieuw concept voor ontwikkelaars, wat kan leiden tot een leercurve bij het overschakelen naar de aanpak van OpenFGA.

Dankzij de steun van de Cloud Native Foundation heeft het goede ondersteuning en meerdere SDK's voor integratie. Hoewel het actief wordt onderhouden, is er mogelijk minder documentatie of community support beschikbaar omdat het een relatief nieuw project is in vergelijking met sommige gevestigde access control oplossingen.

OpenFGA is ontworpen voor prestaties en om efficiënt autorisatie controles uit te voeren voor grootschalige applicaties met veel gebruikers en resources. Het implementeren van OpenFGA voegt een extra laag toe aan de applicatiearchitectuur en het is niet bekend hoe de prestaties zich verhouden tot traditionele platforms voor toegangscontrole. Als de autorisatiebehoeften eenvoudig zijn, weegt de extra complexiteit mogelijk niet op tegen de voordelen.

De combinatie van een nieuw autorisatieconcept en een uitgebreid platform rechtvaardigt een plek in de assess ring voor OpenFGA.

Tools

Assess

- 15. GitHub Codespaces
- 16. Claude 3
- 21. Thunderdome Dev
- 24. Konsist

Hold

- 17. Diffblue Cover
- 25. JobRunr

Trial

- 18. ChatGPT
- 20. Qodana
- 22. jlink
- 23. Maven Build Cache Extension

Adopt

- 19. Vector Databases

Vector Databases

ADOPT

Vector databases zijn gespecialiseerde databases die efficiënt hoog-dimensionale vectoren opslaan, indexeren en ophalen. Deze vectoren worden gebruikt voor semantisch zoeken naar tekst, afbeeldingen en meer, waardoor toepassingen zoals ChatGPT en GitHub Copilot kunnen functioneren.

Bij JDriven zijn we van mening dat bedrijven zouden moeten overwegen om vector databases te gebruiken wanneer ze de mogelijkheden van semantisch zoeken verkennen of zich begeven in het AI-domein.

Hoewel er talloze leveranciers van vector databases zijn, zijn de meeste, op basis van onze ervaring, vergelijkbaar, zonder groot onderscheidend kenmerk. Daarom hebben we ervoor gekozen om elk van deze databases niet afzonderlijk op te nemen in onze Tech Radar. De meest populaire databases zijn Pinecone, Chroma, pgVector en Weaviate. Bovendien hebben bedrijven zoals Elasticsearch, Redis en Neo4j vectorfunctionaliteiten geïntegreerd in hun gereedschapskist.

Commit. Develop. Share.

ChatGPT

TRIAL

Het bekende taalmodel dat in staat is om complexe gesprekken te voeren met gebruikers, is intussen meer dan een jaar oud. ChatGPT kan ontwikkelaars helpen bij het oplossen van problemen, verbeteren van code, detecteren van bugs, uitvoeren van statische codeanalyse en meer.

Echter kunnen we niet blindelings vertrouwen op ChatGPT. Er zijn een aantal redenen om zorgvuldig om te gaan met de resultaten. Allereerst zal de tool code met vertrouwen presenteren, zelfs als het fouten bevat, niet geoptimaliseerd is of constructies gebruikt waarvoor betere alternatieven beschikbaar zijn. De gratis versie heeft gegenereerde code opgeleverd met problemen zoals delen door nul, n+1 selectieproblemen en refactors die de kwaliteit van de code verminderden. De gegenereerde code door de betaalde versie is van hogere kwaliteit, maar nog steeds niet vlekkeloos.

Naast functionele problemen kan de tool een beveiligings- of privacyrisico vormen. Chatgeschiedenis wordt gelogd en aan het model gevoed. Daarom is het belangrijk om alleen code bloot te stellen die van generieke aard is en geen bedrijfsspecifieke details of gevoelige informatie bevat. Er is geen garantie dat gegevens veilig zijn.

Al met al is ChatGPT een nuttige en capabele tool, vooral voor het leren van een nieuwe taal, concept of framework. Gebruik het om aanwijzingen te krijgen voor het oplossen van problemen, maar vertrouw niet blindelings op de oplossingen zonder precies te weten wat de gegenereerde code doet. Gebruik het voor refactoring, maar houd er rekening mee dat het mogelijk geen passende patronen of benaderingen toepast.

jlink

TRIAL

jlink is een tool om een op maat gemaakte Java Runtime Environment (JRE) te bouwen. Deze tool doet dit door het strippen van componenten uit de JRE die de doelapplicatie niet nodig heeft. Te denken valt onder andere aan overbodige modules, documentatie en losse tools voor de Java-omgeving. Een op maat gemaakte JRE biedt twee voordelen: het leidt tot bruikbare artefacten (zoals Docker images) van een kleiner formaat, en vermindert de hoeveelheid potentiële ingangen voor aanvallen. Een kleiner formaat images leidt tot kortere uitroltijden, met name binnen cloud omgevingen, en mogelijk ook tot lagere kosten bij het beheren en hosten van images en omgevingen. Door onnodige modules niet toe te voegen, kunnen kwaadwillende partijen geen gebruik maken van mogelijke veiligheidslekken in deze modules. Dergelijke lekken kunnen in catastrofale gevallen leiden tot privilege escalation en niet-geautoriseerde datatoegang.

We hebben jlink onder *Trial* ingedeeld, omdat wij duidelijke voordelen zien bij het gebruik hiervan. Zoals met praktisch alle tooling zijn er echter ook een aantal keerzijden. De tool om te bepalen welke modules nodig zijn voor een gegeven applicatie, jdeps, geeft niet altijd een eenduidige rapportage van welke modules nodig zijn. Vaak moet met de hand nog een aantal modules aan de lijst worden toegevoegd. Een serieuzer probleem is dat afwezigheid van benodigde modules kan leiden tot cryptische foutmeldingen en veranderd gedrag van de applicatie. Dit zorgt ervoor, dat een op maat gemaakte JRE een klein risico kan vormen, wat het voor sommige toepassingen ongeschikt maakt.

© 2024 JDriven

Maven Build Cache Extension

TRIAL

Maven Build Cache [https://maven.apache.org/extensions/maven-build-cache-extension/] is een extensie voor Maven (beschikbaar vanaf versie 3.9) waarmee je je Maven builds efficiënter kunt maken door slim gebruik te maken van caching (vergelijkbaar met de Gradle Build Cache).

De build cache kan op verschillende momenten in het build-proces de outputs, zoals gegenereerde code of gecompileerde classes, in een cache plaatsen. Als op een later moment de build nogmaals wordt uitgevoerd met (deels) ongewijzigde inputs, kunnen de eerder gecachete outputs worden hergebruikt. Hierdoor kunnen delen van de build, zoals het compileren van code en het uitvoeren van tests, worden overgeslagen, waardoor de totale buildtijd afneemt. Bijkomend voordeel is dat build caches ook gedeeld kunnen worden door gebruik te maken van een remote cache.

Door de Maven Build Cache extension te introduceren is het in veel gevallen mogelijk om de buildtijd van je Maven-projecten significant te verminderen. Doordat dit een vrij nieuwe feature van Maven is, bestaat de mogelijkheid dat nog niet elke third-party Maven-plugin hier goed mee samenwerkt. Daarnaast is elk Maven-project anders, dus zal het enig uitzoekwerk kosten om de build cache op de juiste manier te laten werken. Een verkeerd geconfigureerde build cache kan leiden tot instabiele, niet-reproduceerbare builds. Om deze redenen plaatsen wij het in *Trial*.

Qodana

TRIAL

Qodana is een Static Code Analysis tool van JetBrains. Het wordt meegeleverd met de IntelliJ IDE, maar moet apart geactiveerd worden. Los van IntelliJ-integratie kent het veel CI/CD- en IDE-integraties en kan het o.a. Java, Kotlin, JavaScript en TypeScript codebases scannen. Het biedt een Qodana Cloud dashboard voor het in kaart brengen van de codekwaliteit over meerdere projecten.

Na een paar jaar als preview beschikbaar te zijn geweest, is Qodana officieel uitgekomen in 2023. Het kent een gratis community-model, maar voor JavaScript/TypeScript- en Spring-ondersteuning ben je aangewezen op een betaald model. Dat betaalde model is echter ook wat het interessant maakt t.o.v. SonarQube, want in Qodana betaal je niet voor de hoeveelheid lines of code (LoC), maar voor het aantal actieve committers. Bovendien zit de CI/CD-integratie bij Qodana in de gratis variant. Dat betekent dat, afhankelijk van de aard van je project en hoe het onderhouden wordt, Qodana aanzienlijk goedkoper zou kunnen uitvallen. Het is de moeite waard om die vergelijking te maken en de tool te proberen.

Claude 3

ASSESS

Claude 3 [https://en.wikipedia.org/wiki/Claude_(language_model)] is een foundational AI service van Anthrophic, een afsplitsing van OpenAI, met een sterke focus op gebruiksveiligheid. Het belangrijkste concept achter hun model is Constitutional AI. Dit systeem tracht behulpzaamheid te optimaliseren en schadelijkheid te minimaliseren in de uitkomsten. Constitutional AI gebruikt een set met regels om misbruik van de AI technologie te voorkomen. Deze set wordt gebruikt om een metamodel mee te trainen. Dit metamodel wordt vervolgens gebruikt om het trainingsproces van het reguliere model te begeleiden, zodat het model leert om geen schadelijke inhoud te genereren. Dit werkt door het metamodel iteratief kritiek en suggesties voor verbetering te laten genereren, zodat de uiteindelijke uitkomst de "grondwetten" niet meer overtreedt. Om de resultaten te valideren, wordt Claude getest tegen een grote set van bewust kwaadaardige prompts.

Wij kiezen ervoor om Claude in de *assess* ring te plaatsen. Op dit moment kan de web UI niet vanuit de EU gebruikt worden, waardoor het lastig is om er gemakkelijk mee te experimenteren. Ook kunnen de

gekozen regels, ondanks dat deze over het algemeen als verstandig worden beschouwd, nog steeds voor een bevooroordeelde (biased) uitkomst zorgen. Dergelijke uitkomsten zullen ook niet altijd stroken met bepaalde culturele patronen. Bovendien is er een dunne scheidslijn tussen het geven van onschadelijke en van ontwijkende antwoorden. In het door Anthropic gepubliceerde onderzoek [https://www.anthropic.com/research] lijkt dit redelijk onder controle, alhoewel gebruikers wordt aangeraden zelf na te gaan in hoeverre deze technologie toepasbaar is voor hun (specifieke) doeleinden.

GitHub Codespaces

ASSESS

GitHub Codespaces [https://github.com/features/codespaces] is een Cloud Development Environment (CDE) die ontwikkelaars voorziet van virtual machines om op te ontwikkelen. Het maakt gebruik van devcontainers [https://containers.dev/], zodat ontwikkelaars hun eigen lokaal draaiende IDE kunnen gebruiken. Dit is een op Docker gebaseerde technologie, en gebruikt Docker images waar de benodigde toolchain op geïnstalleerd is. Hiermee worden problemen met reproduceerbaarheid voorkomen, omdat alle ontwikkelaars dezelfde omgeving gebruiken. Omdat alle ontwikkelaars dezelfde toolchain, configuratie en platform gebruiken, is het onboarden van nieuwe teamleden eenvoudig.

GitHub Codespaces heeft uitstekende support voor Visual Studio Code via een extensie. Ondersteuning voor de JetBrains IDE's ligt op dit moment nog wat achter. Bovendien moeten gebruikers JetBrains Gateway [https://www.jetbrains.com/remote-development/gateway/] gebruiken.

We hebben ervoor gekozen om GitHub Codespaces in de *assess* ring te plaatsen, omdat het naast bestaande workflows kan worden gebruikt, maar deze niet vervangt. Daarom is het uitproberen ervan redelijk risicoloos. We geloven dat een gestandaardiseerde ontwikkelomgeving de productiviteit van ontwikkelaars kan verhogen. Wel kan een slechte internetverbinding mogelijk voor een minder vloeiende ervaring zorgen.

Aangezien GitHub Codespaces een dienst vanuit GitHub is, is het gebruiksgemak het grootst wanneer de gebruikte code repository daar reeds gehost is. Bovendien zijn er kosten verbonden aan de tijd dat een virtual machine in gebruik is. Ook voor opslag van een codespace moet betaald worden zolang deze niet verwijderd wordt. Het integreren met on-premise diensten, bijvoorbeeld voor testen, kan uitsluitend via een VPN worden gedaan.

Ten slotte kan het raadzaam zijn om ook te kijken naar oplossingen van andere partijen voor het hosten van developeromgevingen.

Konsist

ASSESS

Konsist [https://docs.konsist.lemonappdev.com/getting-started/readme] is een *static code-analyzer* library voor Kotlin die helpt codeconventies te garanderen en gedefinieerde architecturele grenzen kan controleren. Controles worden uitgevoerd in unittests en kunnen als zodanig worden uitgevoerd tijdens pull requests om de architectuur van een project te beschermen. Konsist ondersteunt Kotlin-projecten, waaronder Android, Spring en Kotlin Multiplatform, terwijl ArchUnit, een vergelijkbare tool, alleen (JVM) bytecode-analyse ondersteunt.

Wij bij JDriven geloven in codekwaliteit en een helder gedefinieerde software-architectuur. Konsist is een zeer nuttig hulpmiddel in onze gereedschapskist om codeconventies en architecturale grenzen te formaliseren en te verifiëren. Omdat Konsist nog in de kinderschoenen staat, houden we de ontwikkeling ervan in de gaten voordat we de adoptie ervan volledig aanraden.

Thunderdome Dev

ASSESS

Thunderdome.dev [https://thunderdome.dev/] is een opensource-samenwerkingstool. Een onderscheidende feature is als planning poker-tool. Het kan ook worden gebruikt bij andere processen, zoals retrospectives, team checkins, en feature/story mapping. Het proces van planning poker werkt goed in de context van persoonlijke aanwezigheid van teamleden. In een hybride werkomgeving mist die context, waardoor het ontwikkelteam het proces op een andere manier moet vormgeven. Thunderdome.dev zorgt bij planning poker voor een basisstructuur waarmee dit makkelijker wordt.

Aan de ene kant is het een erg toegankelijke tool, aan de andere kant is de toepassing sterk afhankelijk van de werkwijze en behoeftes van het ontwikkelteam. We plaatsen Thunderdome.dev daarom in *Assess*. Houd bij een assessment rekening met alle features van deze tool. Met name voor de mogelijkheid als aanvulling op, of vervanging van een gevestigde tool als Jira.

Diffblue Cover

HOLD

Diffblue Cover [https://www.diffblue.com/products/], een geautomatiseerde tool voor het genereren van unittesten voor Java code, toont sterke punten in het begrijpen van de te testen code en het creëren van geschikte mocks. Echter, het vermogen om complexe mockdata te genereren is beperkt, en de benadering van het genereren van testen komt mogelijk niet overeen met bestaande projectstijlen.

Hoewel Diffblue Cover nuttig kan zijn voor het genereren van boilerplatecode en het opzetten van eerste testen, zal het handmatige testen niet volledig vervangen. Tevens is de compatibiliteit beperkt tot specifieke versies van Spring, Java en JUnit, en het heeft moeite met methoden zonder parameters of klassen zonder constructors. Verder kan Diffblue Cover niet worden uitgevoerd op een heel project of zelfs op pakketten, wat de toepasbaarheid ervan beperkt.

Daarom wordt Diffblue Cover in het kwadrant *Hold* geplaatst van deze editie van onze Tech Radar.

JobRunr

HOLD

JobRunr [https://www.jobrunr.io/] is een Java-library voor het draaien en aansturen van achtergrondprocessen. Met JobRunr kun je jobs plannen, batches uitvoeren, of workflows maken met behulp van Java Lambda-functies, met ingebouwde functies voor het opnieuw proberen van jobs die gefaald zijn. JobRunr komt met een webinterface waarmee je de jobs in de gaten kunt houden, jobs kunt aansturen, en het kan je inzicht geven wanneer er fouten zijn opgetreden. Alhoewel de gratis versie erg interessant is en het overwegen waard om te gebruiken, hebben we JobRunr op *Hold* geplaatst vanwege de pro versie. JobRunr Pro is namelijk niet opensource en er zijn maar een aantal mensen die eraan werken, wat voor wat risico zorgt dat sommige bedrijven liever vermijden. Dus ons advies is om hier zorgvuldig een keuze te maken.

Commit. Develop. Share.

Commit.
Develop.
Share.