JDriven Tech Radar Voorjaar 2023

Onze kijk op recente ontwikkelingen in het veld

7e 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 u helpen dit op te starten binnen uw organisatie. Laat uw teams elkaar inspireren tot innovatie en gezamenlijk komen tot een set aan technologieën en technieken die de ontwikkeling in uw 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.

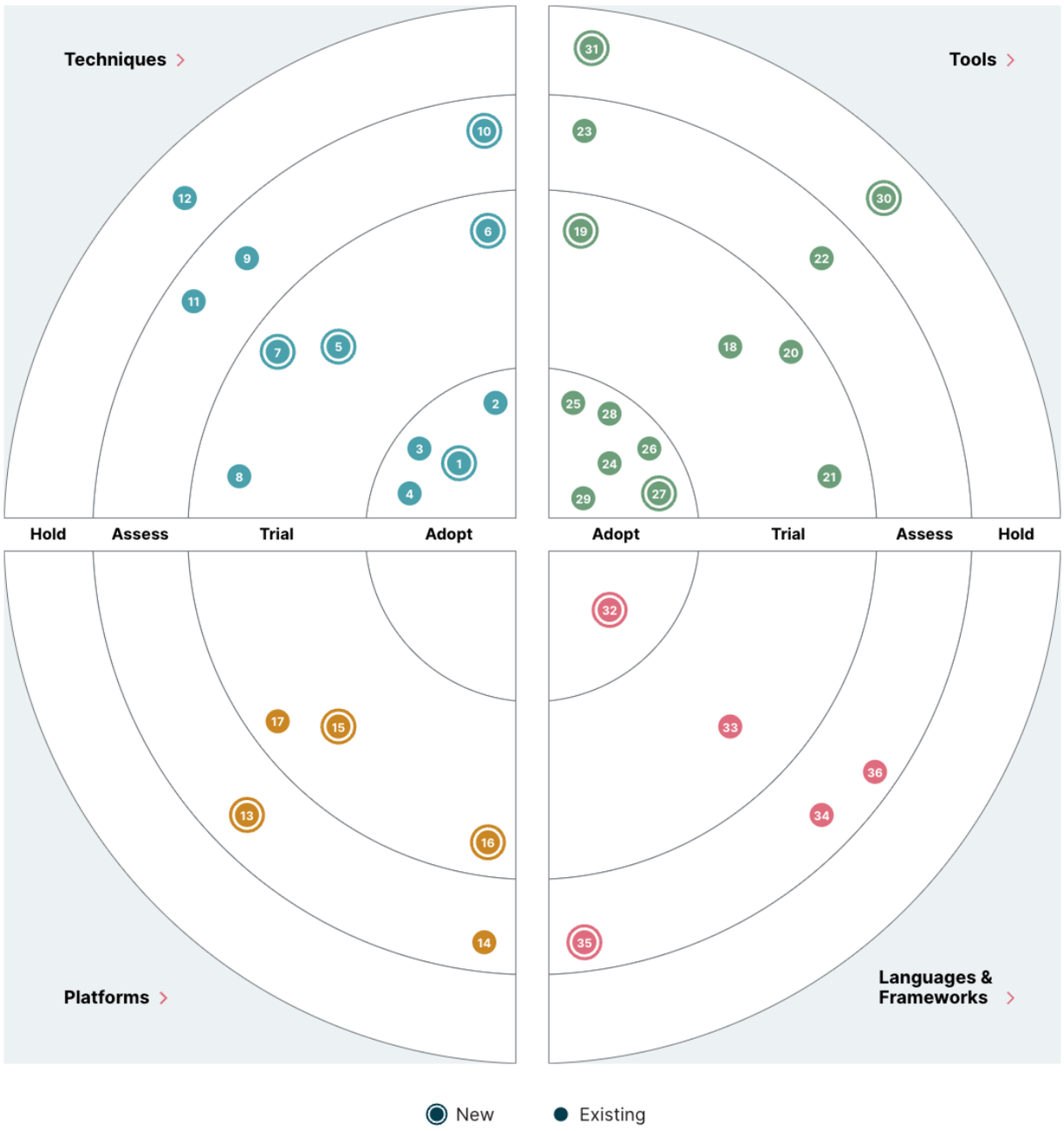
- **Talen & frameworks** die je ondersteunen bij de ontwikkeling van software
- **Platformen** waarop je software kunt uitvoeren
- **Technieken** die je helpen betere software te maken
- **Tools** ter ondersteuning van je ontwikkel- en delivery-proces

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.

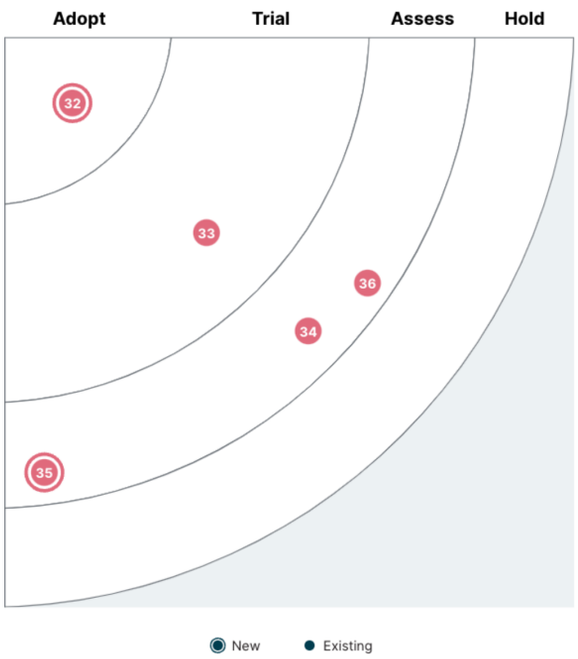


Commit. Develop. Share.

Tools	Talen & frameworks
ADOPT CNCF Roadmap Datafaker Gradle Kotlin DSL k9s Maven Daemon OpenTelemetry **TRIAL** Error Prone Support Sigrid Spring Boot Migrator Testkube **ASSESS** Diffblue Cover SigNoz **HOLD** Docker 4 Desktop Lombok	**ADOPT** Azure Bicep **TRIAL** Spring Modulith **ASSESS** F# Service Weaver Virtual threads **HOLD** -
Platforms	**Technieken**
ADOPT - **TRIAL** Apache Pulsar Edge Computing GraalVM **ASSESS** Dapr On-premise **HOLD** -	**ADOPT** DDD SBOM Tinkering Weighing developer experience **TRIAL** AI code assist Developer Productivity Engineering Team Topologies WebAuthn **ASSESS** Data Oriented Programming GPU Programming Risk storming **HOLD** SAFe

Commit. Develop. Share.

Talen & frameworks



Adopt
32. Azure Bicep

Trial
33. Spring Modulith

Assess
34. F#
35. Service Weaver
36. Virtual threads

Azure Bicep

ADOPT

Azure Bicep [https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview?tabs=bicep] is een domeinspecifieke taal (DSL) waarmee ontwikkelaars infrastructuren als code kunnen schrijven voor Azure-resources. Deze technologie vereenvoudigt het proces van het maken en implementeren van Azure-resources door ontwikkelaars in staat te stellen declaratieve code te schrijven die de gewenste staat van hun infrastructuur definieert. Azure Bicep is een open-sourceproject ontwikkeld door Microsoft en is gebouwd op de bewezen Azure Resource Manager (ARM) infrastructuur.

Ontwikkelaars gebruiken Azure Bicep om Azure-resources zoals virtuele machines, opslagaccounts en netwerkinterfaces te beheren. Door Bicep te gebruiken, kunnen ontwikkelaars herbruikbare code-modules maken die kunnen worden gebruikt in verschillende omgevingen, waardoor de tijd en moeite die nodig zijn voor infrastructuurimplementatie worden verminderd. Azure Bicep maakt ook infrastructuurautomatisering mogelijk, waardoor het gemakkelijker wordt om Azure-resources op schaal te beheren en te onderhouden.

Wij geloven dat als u ervoor kiest om te werken met Infrastructure as Code, Azure Bicep de standaardkeuze is als u werkt met Azure. De adoptie ervan stelt organisaties in staat hun infrastructuur-implementatiesnelheid te verbeteren, de betrouwbaarheid te vergroten en het risico op menselijke fouten te verminderen. Bovendien zorgt de integratie met Azure Resource Manager ervoor dat Azure Bicep volledig compatibel is met bestaande ARM-templates, waardoor het gemakkelijk is aan te nemen voor organisaties die al Azure gebruiken. Ook is er een Azure Bicep Visual Studio-extensie dat ontwikkelaars een gestroomlijnde ervaring voor het schrijven van infrastructuren voor Azure-resources biedt. Met deze extensie kunnen ontwikkelaars profiteren van IntelliSense en syntax highlighting, waardoor het gemakkelijker wordt om Bicep-code te schrijven, te onderhouden en te testen. Daarom raden wij aan dat bedrijven Azure Bicep gaan gebruiken om hun Azure-infrastructuurimplementatieproces te stroomlijnen en hun algehele infrastructuurbeheer te verbeteren.

Spring Modulith

TRIAL

Domain Driven Design heeft developers doen inzien dat softwarearchitectuur kan en moet helpen met het faciliteren van de evolutie van softwaresystemen bij wijzigende inzichten in business requirements. Microservices kunnen daarbij ingezet worden om functionele units te isoleren, maar dat introduceert uitdagingen op het gebied van service-orkestratie en HTTP-communicatie. Dit heeft developers doen heroverwegen of diezelfde modulariteit is aan te brengen en af te dwingen in monolithische applicaties.

Spring biedt als framework technische stereotypen zoals @Controller, @Repository, etc, zonder een mening op te leggen over hoe componenten die er gebruik van maken van elkaar afhankelijk zijn. Met Spring Modulith [https://spring.io/projects/spring-modulith] beoogt het juist wel een domain-aligned structuur aan te moedigen in een Spring Boot applicatie. In de praktijk lijkt dit te betekenen dat een goed opgezette Spring Boot & Modulith applicatie kandidaten voor dependency injection aan banden legt op basis van package structure conventies. Daarbij biedt het wel weer manieren om moduledetectie aan te passen t.o.v. die conventies.

Het borduurt voort op ArchUnit in hoe het afhankelijkheden test en beperkt, maar het is onduidelijk of het daarbij ook ingrijpt op het runtime bootstrap-proces van een applicatie, of dat het slechts hulp biedt bij de integratietests en - nog een goede feature - C4 architectuurdocumentatie in AsciiDoc genereert. Spring Modulith biedt een nieuwe manier voor het integratietesten van individuele modules, die naar verwachting een stuk sneller zal uitpakken dan het gebruik van @SpringBootTest, omdat het alleen de modules instantieert die voor de test relevant zijn.

Tevens stuurt Spring Modulith aan op het gebruik van event-driven pubsub voor loose coupling. Dit zou niet de enige aan te bevelen manier van loose coupling moeten zijn, en wederom is het niet duidelijk of dit een systeem is dat moet helpen bij integratietests, of dat het ook at runtime in een applicatie kan worden ingezet.

Met dit in het achterhoofd is het het onderzoeken waard in hoeverre Spring Modulith kan helpen met modulariteit in monolithische applicaties.

F#

ASSESS

Java krijgt steeds meer mogelijkheden om functioneel te programmeren (FP). Lambdas, records en sealed classes zijn de laatste jaren aan de taal toegevoegd. Pattern matching, een techniek voor het herkennen van een specifiek patroon in data, is op dit moment in ontwikkeling. Door ook te kijken naar volledig functionele talen, wordt het gemakkelijker om de FP-concepten erachter beter te begrijpen.

Een taal die hier uitermate geschikt voor is, is F# [https://fsharp.org/], een taal die draait op de .NET runtime. F# is een strongly typed programmeertaal, waar zowel functionele, imperatieve en objectgeoriënteerde programmeermethoden gemakkelijk gecombineerd kunnen worden. Doordat de syntax erg compact en eenvoudig is, kan men snel aan de slag met de features die de taal biedt. FP-begrippen als pure (higher order) functions, immutability, currying, pattern matching, recursiviteit, algebraïsche datatypes, tail-call optimization, lazy evaluation, tuples en types zijn basiselementen binnen de taal. Omdat al deze opties in de taal zelf opgenomen zijn, kun je sneller begrijpen wat de achterliggende concepten zijn.

De toepasbaarheid van non-JVM talen is binnen de JVM-community wellicht klein. Het zal echter gemakkelijker worden om nieuwe features in Java zoals volledige pattern matching (JEP 433 [https://openjdk.org/jeps/433]), efficiënter maken van tail-calls (Project Loom [http://cr.openjdk.java.net/~rpressler/loom/Loom-Proposal.html]), lazy static fields (JEP draft [https://openjdk.org/jeps/8209964]) en Value Objects (JEP draft [https://openjdk.org/jeps/8277163]) te gaan gebruiken. Om die reden bevelen wij F# aan als taal om functioneel programmeren te leren kennen.

Service Weaver

ASSESS

Service Weaver [https://opensource.googleblog.com/2023/03/introducing-service-weaver-framework-for-writing-distributed-applications.html] is een framework voor het ontwikkelen van gedistribueerde applicaties. Het is ontworpen om ontwikkelaars te helpen bij het bouwen van complexe systemen die gebruikmaken van een gedistribueerde architectuur.

Service Weaver omvat een reeks abstracties en patronen die de definitie, communicatie en samenwerking van services mogelijk maken. Door een gedecentraliseerde aanpak te hanteren, kunnen services autonoom werken en communiceren via asynchrone berichten.

Service Weaver is een nieuw Framework dat veel potentieel biedt bij de ontwikkeling van gedistribueerde applicaties. Daarom is Service Weaver geplaatst in het *Assess* kwadrant van deze editie van onze Tech Radar.

Virtual threads

ASSESS

Threads zijn kostbare, maar ook dure resources. Dit principe zie je tegenwoordig weer vaak terugkomen. Reactive systemen werken bijvoorbeeld zoveel mogelijk non-blocking, en Javascript kan single-threaded draaien, maar toch meerdere dingen tegelijk doen. De 'threads' die in deze gevallen de taken uitvoeren, mappen in dat geval niet 1-op-1 op de OS-threads onder de motorkap. Daarom wordt in dit geval vaak gesproken over Virtual Threads, of Fibers ('vezels', kleinere threads).

Dit concept om threads efficiënter te gebruiken door deze meerdere taken 'tegelijk' te laten uitvoeren is niet nieuw. Vroege versies van macOS en Windows maakten al gebruik van zogeheten cooperative (non-preemptive) multitasking, waarbij meerdere processen/taken op een beperkt aantal threads konden draaien door elkaar periodiek de ruimte te geven.

De afgelopen jaren zijn er meerdere reactive libraries uitgekomen (zoals RxJs, RxJava, Project Reactor), die dit principe weer een nieuw leven hebben ingeblazen. Daar bovenop hebben de grote framework-ontwikkelaars deze libraries omarmd en allerlei 'reactive' functionaliteiten toegevoegd (Reactive Spring, Quarkus, enz).

Ook Kotlin heeft met de Coroutines library een bijna native manier ontwikkeld om meerdere taken concurrent op een beperkt aantal threads uit te kunnen voeren. Bijna native, omdat deze library, net als de andere reactive Java-libraries, in essentie nog gewoon op een 'traditionele' JVM draaien die met normale OS-threads werkt.

Met Project Loom (JEP 425, beschikbaar in Java 19 als Preview feature) wordt het concept van Virtual threads in de JVM geïmplementeerd. Het zal de komende tijd interessant zijn om te zien hoe de ontwikkelaars van de eerdergenoemde talen en frameworks hierop zullen inspelen. Naar onze verwachting zullen zij vooral gebaat zijn met deze ontwikkeling. Native ondersteuning voor Virtual Threads kan vooral op het gebied van performance een positieve bijdrage leveren, aangezien de ondersteuning hiervoor nu niet meer bovenop de JVM hoeft te worden gebouwd, maar aangeboden wordt vanuit de JVM zelf, dus op een lager abstractieniveau. Deze positieve ontwikkeling kan een reden zijn om alvast de mogelijkheden te bekijken op het vlak van Virtual Threads, Coroutines, en reactive programming.

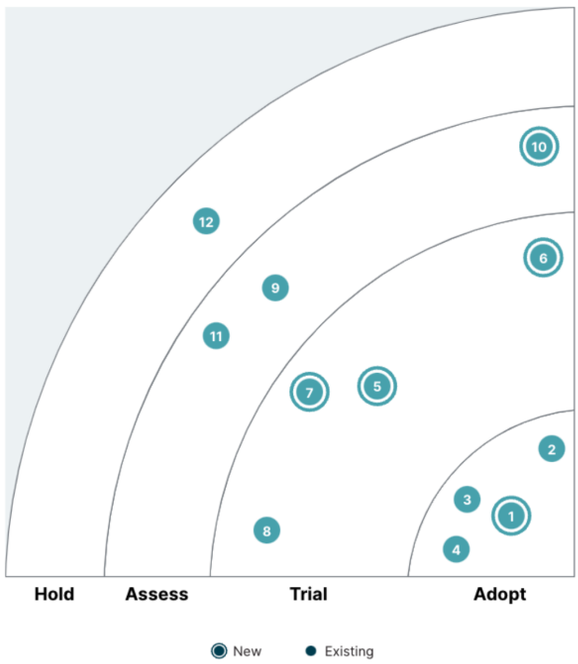
Technieken

Adopt
1. DDD
2. SBOM
3. Tinkering
4. Weighing developer experience

Trial
5. AI code assist
6. Developer Productivity Engineering
7. Team Topologies
8. WebAuth

Assess
9. Data Oriented Programming
10. GPU Programming
11. Risk storming

Hold
12. SAFe



DDD

ADOPT

Domain Driven Design (DDD) is een techniek die onderverdeeld kan worden in strategisch en tactisch ontwerp. Het eerste biedt manieren om een probleemdomein te benaderen en bounded contexts en hun onderlinge afhankelijkheden te identificeren, alsmede een ubiquitous language. Het tweede biedt manieren en building blocks om deze bounded contexts de modelleren en implementeren.

Alhoewel DDD als een concept reeds 20 jaar oud is, zien we dat het recentelijk weer in de schijnwerpers staat. Tools voor CQRS, Event Sourcing en message driven architectures bestaan al en blijven doorontwikkeld worden, en technieken zoals event storming en domain storytelling worden vaker gebruikt. Maar belangrijker nog, is dat architecten en developers ondervinden dat de overstap van monolieten naar microservices hun problemen soms heeft verergerd; in het bijzonder wanneer bounded contexts en de afhankelijkheden daartussen onvoldoende zijn gemodelleerd en begrepen. Zij grijpen nu naar DDD voor hulp.

SBOM

ADOPT

Een Software Bill of Materials (SBOM) is een definitie van softwareafhankelijkheden vastgelegd op één centrale plek. In deze SBOM staan alle afhankelijkheden van derden met bijbehorende exacte versienummers gespecificeerd. Het helpt teams om snel inzicht te verschaffen van alle afhankelijkheden. Daarnaast geeft het de mogelijkheid om snel de versies van deze afhankelijkheden bij te werken.

Bij grote kwetsbaarheden is het zaak acuut en adequaat te handelen. Je wilt snel in kaart hebben of je eigen software kwetsbaar is. Een SBOM maakt dat mogelijk.

Thoughtworks heeft de SBOM op de technology radar staan onder SBOM [https://www.thoughtworks.com/radar/techniques?blipid=202110076].

Tinkering

ADOPT

Tinkering is experimenteren met ideeën/frameworks in een omgeving om de volledige eigenschappen te begrijpen. Leren door middel van tinkering is gebaseerd op een presentatie door Tom Cools [https://tomcools.be/talks/]. In de IT-sector blijven we leren omdat sommige technologieën verdwijnen en nieuwe technologieën verschijnen. Tinkering maakt het mogelijk om gestructureerd te leren, kennis te vergroten en vaardigheden aan te scherpen. Het belangrijkste aspect is om alleen het juiste te leren door middel van "zones of proximal development": wat we al weten, wat we kunnen leren en wat we niet kunnen leren. Het delen van wat we hebben geleerd is een goede gewoonte. Als we willen uitleggen wat we hebben geleerd aan anderen moeten we een goed begrip hebben over een onderwerp. Het dwingt ons om alvast na te denken over vragen die mensen kunnen hebben, en daarop de antwoorden te hebben.

Tinkering is erg bruikbaar om met focus iets nieuws te leren, zodat we meer kennis krijgen en onze vaardigheden verbeteren en plaatsen we daarom in *Adopt*.

Weighing developer experience

ADOPT

Laten we beginnen met een definitie van developer experience en wat hiermee bedoeld wordt. Developer experience, ook wel DX genoemd, beschrijft de perceptie en het gevoel dat ontwikkelaars hebben bij een taal, een tool of een techniek. Hoe beter de ontwikkelaar in staat is om met behulp van de taal, tool of techniek zijn werk te doen, hoe beter zijn gevoel hierbij is. Hierbij let een ontwikkelaar over het algemeen op drie elementen: gebruiksvriendelijkheid, gebruiksgemak en betrouwbaarheid. Gebruiksvriendelijkheid gaat over hoe eenvoudig de taal, tool of techniek is in het gebruik. Gebruiksgemak gaat over hoe makkelijk de functionaliteiten te vinden zijn. Het derde punt gaat over hoeveel vertrouwen je hebt in het gebruik van de taal, tool of techniek.

Waarom is DX van belang?

Wanneer de DX van een taal, tool of techniek hoog is, zal het betreffende individu of het team snel geneigd zijn hier nogmaals voor te kiezen. Het is dus van groot belang voor de ontwikkelaars van een taal, tool of techniek de juiste aandacht en focus te hebben op DX.

Een hoge DX begint met het centraal stellen van de ontwikkelaar en je kunnen verplaatsen in de eindgebruiker.

Om deze reden hebben wij Weighing developer experience in het *Adopt* quadrant van deze editie van onze Tech Radar geplaatst.

AI code assist

TRIAL

Een AI-code assistent is een tool die gebruik maakt van machine learning-algoritmen en statistische modellen om code te voorspellen. Het helpt ontwikkelaars bij het schrijven van code door suggesties te doen voor het voltooien van codefragmenten, het corrigeren van fouten en het geven van aanbevelingen voor optimalisatie. Tevens kan er gebruik gemaakt worden van prompt-based engineering, waar met natuurlijke taal code wordt gegeneerd.

Als ontwikkelaar kun je een AI-code assistent gebruiken door deze te integreren als plugin in je IDE (Integrated Development Environment). Hierdoor krijg je tijdens het typen van de code direct suggesties en correcties aangeboden, waardoor je productiever en efficiënter kunt werken. Bovendien kan een AI-code assistent je helpen om nieuwe talen en frameworks sneller te leren, doordat het

suggesties doet op basis van best practices en veelgebruikte patronen.

Op dit moment bevinden de AI-code assistenten zich nog in de kinderschoenen. Zo hebben AI-code assistenten weinig kennis van specifiek domeinen, genereren ze mogelijk foute code en kan het eigendomsrecht ook een probleem zijn. Wij geloven echter, dat ondanks deze reële bezwaren, binnen vijf jaar de AI-code assists mainstream zullen worden in software ontwikkeling. Tools als Tabnine [https://www.tabnine.com], GitHub Copilot [https://github.com/features/copilot/], Machinet [https://www.machinet.net] en IntelliCode [https://visualstudio.microsoft.com/services/intellicode/] strijden om de aandacht. De vraag is dan ook niet of je dit als ontwikkelaar gaat gebruiken, maar wanneer. Wij raden daarom aan om dergelijke tooling nu te onderzoeken, zodat wanneer AI-code assistenten volwassen zijn, het meteen voor je productiecode gebruikt kan worden.

Developer Productivity Engineering

TRIAL

Developer Productivity Engineering (DPE) [https://gradle.com/developer-productivity-engineering/] is een tak van software engineering die zich richt op het verbeteren van de efficiëntie en productiviteit van ontwikkelaars op basis van meetbare data. DPE richt zich op het ontwerpen en bouwen van tools, processen en infrastructuren die de workflow van ontwikkelaars kunnen stroomlijnen en hun ontwikkelingscyclus kunnen versnellen. DPE omvat een breed scala aan activiteiten, zoals bijvoorbeeld het ontwerpen en implementeren van geautomatiseerde testsystemen en het optimaliseren van de build- en deploy-processen om de feedback loop te verkorten. Doordat het optimaliseren gedaan wordt op basis van meetbare data kan dit team bewijzen hoeveel tijd door hun activiteiten wordt bespaard.

Gradle is dit op het moment in de markt aan het brengen en vooral voor grotere bedrijven is dit een interessante trend om te volgen. Doordat het op meetbare resultaten is gebaseerd en het op verschillende niveaus kan worden geïmplementeerd, vinden we het geschikt om het op *Adopt* te zetten.

Team Topologies

TRIAL

Team Topologies [https://teamtopologies.com/key-concepts] is een praktisch model voor organisatorisch ontwerp en team interactie, gebaseerd op vier fundamentele team types en drie interactie patronen. Het is een model dat teams ziet als de basis van delivery, en dat team structuren en communicatielijnen laat groeien met technologische en organisatorische volwassenheid. Team Topologies biedt een duidelijke manier voor teams om met elkaar samen te werken, teneinde de resulterende software architectuur helderder en beter onderhoudbaar te maken. Daarbij interpreteert het problemen tussen teams als waardevolle signalen die kunnen helpen bij zelfsturende organisatie.

Team Topologies benadrukt het optimaliseren van team interactie, en vult daarmee het strategisch ontwerp van Domain Driven Design aan bij het in kaart brengen van interactie tussen software systemen. In combinatie met het bewustzijn van Conway's Law en team cognitive load, helpt dit bij het produceren van architecturen die onderhouden kunnen worden door een organisatie.

WebAuthn

TRIAL

Web Authentication API (ook wel WebAuthn [https://webauthn.guide] genoemd) is een specificatie geschreven door W3C en FIDO in samenwerking met Google, Mozilla, Microsoft, Yubico en andere bedrijven. De API maakt het mogelijk om gebruikers te registreren en authenticeren met behulp van public key cryptografie in plaats van een wachtwoord. WebAuthn gebruikt public key cryptografie waarbij de public key wordt bewaard op de server waarvoor we ons willen authenticeren. De private key wordt beveiligd opgeslagen op onze laptop of telefoon. Om de private key weer te ontsleutelen moeten we biometrische eigenschappen gebruiken zoals een vingerafdruk of een gezichtsscan. Dit betekent dat authenticatie nu gedaan wordt met iets dat we zijn (vingerafdruk of gezichtsscan) en iets dat we hebben (telefoon of computer). Elke grote webbrowser heeft support voor WebAuthn, waardoor het nu al bruikbaar is als authenticatie-optie voor gebruikers.

Een nadeel is dat iedere website nog wel een andere manier heeft om de public key te kunnen registeren voor de gebruiker. Hierdoor is er voor de gebruiker niet een standaardmanier om dit te kunnen doen. Het zou mooi zijn als in de toekomst dit meer gestroomlijnd is.

Niets houdt ons tegen WebAuthn voor authenticatie te gebruiken in onze webapplicatie. We kunnen WebAuthn gebruiken als 2-factor authenticatie in het authenticatieproces of meteen daadwerkelijk inloggen zonder wachtwoord aan te bieden en is daarom onderdeel van het *Trial* kwadrant.

Data Oriented Programming

ASSESS

Data oriented programming is een paradigma dat zich focust op data en datatransformaties. Het data oriented programming paradigma is niet iets nieuws, het wordt al jaren toegepast in talen zoals C en Clojure. Belangrijk is dat het geen alternatief is op object-georiënteerd programmeren maar een toevoeging hierop, ze zijn namelijk erg goed in combinatie te gebruiken.

Binnen data oriented programming paradigma is data de kern, maar wat wordt hier nu mee bedoeld? Als je abstract kijkt naar de meeste applicaties dan bestaan ze uit een keten van datatransformaties. Binnen data oriented talen wordt hierop ingezet door voornamelijk datastructuren als de `HashMap` in te zetten om data in op te slaan, omdat deze makkelijk te transformeren is vanwege de uitgebreide Map API.

Java krijgt steeds meer features om Data Oriented Programming [https://www.infoq.com/articles/data-oriented-programming-java/] te ondersteunen. Hierbij worden nog steeds types in plaats van `HashMap`s gebruikt, maar er worden wel constructies toegevoegd om meer concepten van data oriented programming toe te passen. Om die reden zijn records, sealed classes en pattern matching aan de taal toegevoegd. Deze constructies geven de mogelijkheid om data te scheiden van functionaliteit, wat een groot onderdeel van data oriented programming is.

GPU Programming

ASSESS

General purpose programming on graphics processing units (GPU) is een techniek waarmee generieke programma's op een GPU kunnen worden uitgevoerd. Deze techniek wordt veelal gebruikt voor toepassingen die veel verwerkingskracht nodig hebben, zoals machine learning, data-analyse en complexe wetenschappelijke en financiële simulaties. Oorspronkelijk zijn GPU's ontworpen voor het verwerken en renderen van computer graphics, maar GPU's bieden grote snelheidsvoordelen voor alle type toepassingen die een grote mate van parallelle verwerking toestaan.

We hebben GPU-programmering geselecteerd voor deze editie van onze Tech radar, omdat het snel populairder wordt binnen verschillende sectoren. De hogere verwerkingssnelheid kan taken, waarvoor eerder niet de tijd beschikbaar was om op het resultaat te wachten, binnen bereik brengen. De specifieke architectuur van deze processoren maakt het echter ingewikkeld om software te ontwikkelen. De veelgebruikte GPGPU programmeertalen CUDA en OpenCL zijn bovendien geïnspireerd op C en redelijk low-level. Om effectieve code te schrijven veronderstellen deze talen een gedegen kennis van het GPGPU platform. Hierdoor is het nog steeds te vroeg voor universele adoptie binnen de industrie.

Desondanks geloven we dat, zelfs voor Java-ontwikkelaars, GPU-programmeren blijft groeien in belang. Een mooi voorbeeld is onder andere de snelle opkomst van Large Language Models (LLM's), zoals ChatGPT. Deze zijn sterk afhankelijk van GPU's om een gebruiker binnen een acceptabele tijd van een antwoord te voorzien.

Tot slot is GPU-programmeren een krachtige technologie die de manier waarop we gegevens verwerken en analyseren transformeert.

Naarmate GPU's blijven evolueren en toegankelijker worden, kunnen we in de toekomst nog meer wijdverspreide adoptie van deze technologie verwachten.

Voor nu raden we bedrijven aan om te beoordelen of er taken en/of services zijn die kunnen profiteren van de GPU en hiermee nieuwe diensten te kunnen aanbieden.

Risk storming

ASSESS

Bij het implementeren van oplossingen voor business requirements kan het gebeuren dat developers structureel onderwerpen, architectuurvraagstukken, of code mijden. Het kan komen door technical debt, maar vaker is het een kwestie van onzekerheid; een soort fear driven development. Het is belangrijk om zulke situaties te herkennen en erkennen, en vervolgens een plan te maken om het te adresseren.

Een naam voor deze bezigheid die in zwang lijkt te raken is Risk Storming. Hierbij doe je op diverse niveaus van je softwaresystemen - van een enkele Java class binnen een applicatie tot het totale landschap van interacterende services - een analyse van de 'hot spots' om tot een overzicht te komen van waar developers een risico ervaren. Denk aan plaatsen waar de business requirements onbekend zijn en slechts uit de oplossing afgeleid kunnen worden, of een technische oplossing die zelf onvoldoende begrepen wordt. Maar ook als er een zorg bestaat over een onderbelicht aspect als security, compliance, onderhoudbaarheid of uitbreidbaarheid. Het is dus breder, of nader gedefinieerd, dan technical debt. En het kan in zowel greenfield- als brownfield-projecten voorkomen.

Eenmaal geïdentificeerd is het zaak om het risico van het niet wegnemen van de zorgen rondom deze hot spots te kwantificeren, en het wegnemen in te plannen in lijn met werk aan de value streams waar een team zich om bekommert. Dat deze risicoanalyse op diverse detailniveaus van een softwareoplossing kan worden gedaan, sluit aan bij het C4-model van Simon Brown. Simon Brown heeft zelf dan ook een aanpak voor risk storming geïnitieerd, die hij toepasselijk op https://riskstorming.com/ beschrijft, en die de moeite waard lijkt om als concrete invulling van bovengenoemde activiteiten uit te proberen.

SAFe

HOLD

Scaled Agile Framework (SAFe) is een samenstelling van organisatiestructuren en processtructuren met als doel om lean en agile toepassingen schaalbaar te maken. Schaalbaarheid van deze principes betekent prioriteit geven aan het verbeteren van samenwerking tussen alle groepen in de organisatie. SAFe is effectief in het creëren van bewustwording over best practices binnen grote organisaties. Uit onze ervaring blijkt dat SAFe vaak uitmondt in het uitrollen van talrijke nieuwe processen die de effectiviteit van ontwikkelteams in de weg staan.

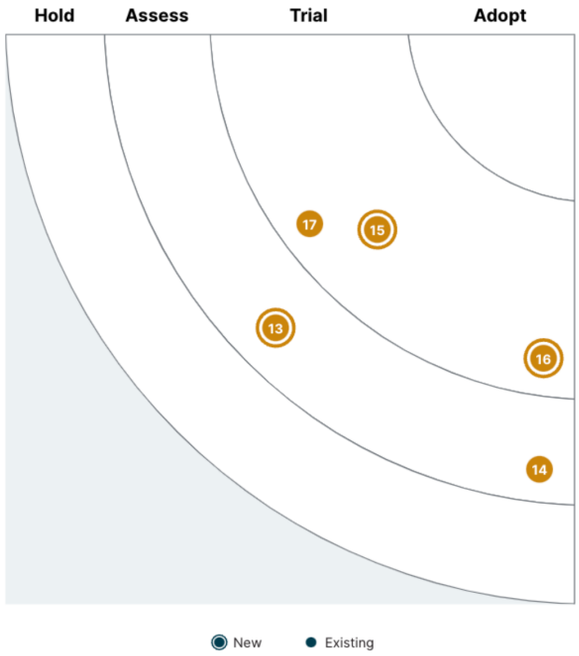
Een voorbeeld van een nieuw proces is Program Increment (PI) planning-evenementen. Een planning-evenement heeft als doel om samenwerking en communicatie tussen teams aan te moedigen. Er wordt daardoor een verwachting gecreëerd dat een planning-evenement nodig is om effectiviteit te bewaken, terwijl flexibele, spontane communicatie tussen teams hetzelfde kan bereiken.

We zien dat bedrijven de belofte van SAFe van betere Lean en Agile processen omarmen. Doordat SAFe als totaaloplossing wordt gezien, is het makkelijk om te vergeten dat SAFe grote veranderingen vereist, ook op bestuurlijk niveau. Er wordt dan eerst gestuurd op het introduceren van nieuwe SAFe processen, voordat duidelijk is of de hele organisatie er klaar voor is. Dit leidt tot een situatie waar er gebrek is aan team buy-in op de veranderingen. Onderliggende organisatorische en bedrijfsculturele problemen worden meestal niet opgelost.

Platforms

Assess
13. Dapr
14. On-premise

Trial
15. Apache Pulsar
16. Edge Computing
17. GraalVM



Apache Pulsar

TRIAL

Apache Pulsar [https://pulsar.apache.org/] is een open-source gedistribueerd pub-sub messaging systeem dat een hoge mate van schaalbaarheid, betrouwbaarheid en prestaties biedt. Vergelijkbaar met Apache Kafka en RabbitMQ biedt Pulsar een manier voor applicaties om met elkaar te communiceren door berichten uit te wisselen via een broker.

Pulsar heeft echter enkele verschillen in vergelijking met Kafka en RabbitMQ. Een van de belangrijkste voordelen van Pulsar is de mogelijkheid om zowel een queue- als pub-sub model te ondersteunen in één systeem, wat flexibiliteit biedt voor verschillende typen berichtenverkeer. Pulsar biedt ook geavanceerde functies zoals dynamische partitionering, die zorgt voor een betere load balancing en resource gebruik in gedistribueerde omgevingen.

In termen van schaalbaarheid is de architectuur van Pulsar ontworpen om miljoenen berichten per seconde te verwerken met lage latentie en hoge doorvoer, waardoor het een goede keuze is voor real-time streaming toepassingen. Pulsar heeft ook een ingebouwd gelaagd opslagsysteem waarmee gegevens efficiënt kunnen worden bewaard en opgehaald.

Hoewel Apache Kafka en RabbitMQ hun eigen sterke punten en gebruiksmogelijkheden hebben, is Apache Pulsar een aantrekkelijke optie voor bedrijven die op zoek zijn naar een zeer schaalbaar, duurzaam en flexibel berichtensysteem dat zowel queue- als pub-sub berichtensystemen aankan.

Bij JDriven zien we dat steeds meer klanten experimenteren met Apache Pulsar. Gezien bovengenoemde voordelen hebben wij Pulsar op *Trial* gezet. Voor nieuwe landschappen kan Apache Pulsar een interessant alteratief voor Kafka of RabbitMQ zijn. Wanneer er geen problemen zijn op bestaande platforms zie wij echter nog geen redenen om deze te migreren naar Pulsar.

Edge Computing

TRIAL

Edge computing brengt compute en data storage dichter bij de eindgebruiker, die daardoor snellere berekeningen en responsetijden mag verwachten. Dit is vergelijkbaar met hoe content delivery networks de performance van statische websites verbeteren. Recentelijk hebben diverse cloud providers edge functions aan hun edge nodes toegevoegd. Edge functions introduceren de optie om bepaalde functionaliteit van de client te verplaatsen naar deze edge nodes.

Vendors zoals Netlify, AWS en Vercel zijn hun portfolio rondom edge functions en bijbehorende middleware aan het uitbreiden. Vanuit de frontend bezien is er nu een plek dichter bij de client, die een deel van de server side rendering kan verzorgen. Dit heeft het voordeel van snelheid, maar het introduceert ook het beveiligingsrisico van meer contactpunten in je architectuur, en kan een toename van complexiteit van je architectuur betekenen. Dientengevolge zien we potentie en risico's, wat het rechtvaardigt om met deze nieuwe oplossingen te experimenteren op het punt tussen Single Page Application (SPA) frameworks en de backend.

GraalVM

TRIAL

GraalVM [https://www.graalvm.org/] is een "high-performance" JDK, ontworpen om de uitvoering van toepassingen geschreven in Java en andere JVM-talen te versnellen, terwijl het ook runtimes biedt voor JavaScript, Python, en een aantal andere populaire talen via polyglot-technologieën. Om native executables te maken bevat het de ahead-of-time (AOT) compiler.

Dit is nu onderdeel van OpenJDK [https://www.graalvm.org/2022/openjdk-announcement/]. Belangrijk is wel te weten dat de polyglot-technologieën geen onderdeel zijn van OpenJDK, maar waarschijnlijk wel zijn te gebruiken. Inmiddels is support voor native executables beschikbaar voor alle grote frameworks. Hierdoor is de drempel om GraalVM te gebruiken een stuk lager geworden, vooral met gebruik van third party libraries met de Shared Metadata [https://medium.com/graalvm/enhancing-3rd-party-library-support-in-graalvm-native-image-with-shared-metadata-9eeae1651da4].

In trial om native executables te gebruiken met de kanttekening dat het niet voor elk project past qua toepassing. Het is een afweging tussen aan de ene kant snellere opstarttijden en minder geheugengebruik, maar aan de andere kant een lagere throughput.

Dapr

ASSESS

Dapr [https://dapr.io/] (Distributed Application Runtime) is een door Microsoft ontwikkeld platform dat de connectiviteit van microservices vereenvoudigt. Het biedt een abstractielaag over technieken die vaak in elke microservice moeten worden geïmplementeerd. Denk hierbij aan pub/sub messaging, state management, secrets management, configuratie, tracing en logging. Dapr draait in een sidecar naast de microservice. De microservice communiceert vervolgens met de sidecar via een SDK welke beschikbaar is voor o.a. Java, .NET, Python, Javascript en Rust. Wij hebben Dapr opgemomen in het *Trial* quadrant van deze editie van de Tech Radar omdat het de ontwikkeling van een microservicelandschap sterk vereenvoudigt. Een nadeel dat wij zien is dat het waarschijnlijk lastig weer uit een applicatie te krijgen is.

On-premise

ASSESS

Heroverweging van cloud versus On-premise [https://world.hey.com/dhh/why-we-re-leaving-the-cloud-654b47e0].

Het is ruim 15 jaar geleden dat we voor het eerst kennis maakten met de Cloud. Met grote beloftes en gelikte marketingpraat werden zowel managers als technici overgehaald om over te stappen. Hoe kon je de beloofde kostenbesparingen negeren? Of de reductie in werk en gemak van gebruik ten opzichte van eigen servers?

In deze begindagen wist nog niemand wat deze, voor het merendeel onbewezen, nieuwe techniek echt zou gaan brengen en hebben we veel verhuisd naar een Cloud-infrastructuur. Nu, ruim 15 jaar later, hebben we een goed idee van wat de krachten zijn van de Cloud, maar hebben we relatief weinig stilgestaan bij de zwaktes.

Hoge kosten

Hoewel een groot deel van de hoge kosten die bedrijven ervaren veroorzaakt worden door een slechte inrichting, zijn ook goede, optimaal ingerichte Cloud-infrastructuren kostbaar. Dit is vooral bij inrichtingen waar geen van de krachtige punten van de Cloud worden ingezet. Hier kan het weg migreren van de Cloud kostenbesparing opleveren, die snel kan oplopen en in enkele gevallen meer dan de helft aan kosten kan schelen. Globaal hebben de volgende scenario's vooral baat bij de Cloud:

- Kleine applicaties die maar spaarzaam worden gebruikt. Dit omdat de kosten van de Cloud schalen op een manier die een on-premise oplossing niet kan. Zeker wanneer er wordt geoptimaliseerd naar een serverless implementatie.
- Applicaties met sterk wisselend gebruik. De toegang tot een bijna eindeloze "stand-by" infrastructuur die een Cloud provider kan bieden zorgt voor goede performance en voorkomt downtime die niet kosteneffectief geëvenaard kan worden in een eigen datacenter.
- Applicaties die een hoge bereikbaarheid nodig hebben. De grote reservecapaciteit van een Cloud provider met standaard ingeregelde failovers die zelfs land- en continent-overschrijdend kunnen zijn is hierin ongeëvenaard.

Daarentegen heeft on-premise ook globaal gezien zijn voordeel scenario's:

- Applicaties met zeer vast gebruik. Als een applicatie vrijwel altijd dezelfde hoeveelheid resources nodig heeft en nauwelijks fluctuatie vertoont, ligt ook de noodzakelijke infrastructuur-hoeveelheid vast. Net als in de echte wereld is dan huren duurder dan kopen.

Hoge complexiteit

Bij de introductie van de Cloud is een van de grote verkooppunten een veel eenvoudiger gebruik (ten opzichte van on-premise) geweest. Nu was dat in de begintijd misschien wel waar, maar is met de toevoeging van vele nieuwe tussenliggende services en de groei van bestaande services naar volwaardige volwassen services de complexiteit zo ver toegenomen, dat het minimaal net zo veel expertise is gaan eisen als van de (oude) on-premise infrastructuur.

Betrouwbaarheid Cloud-aanbieder

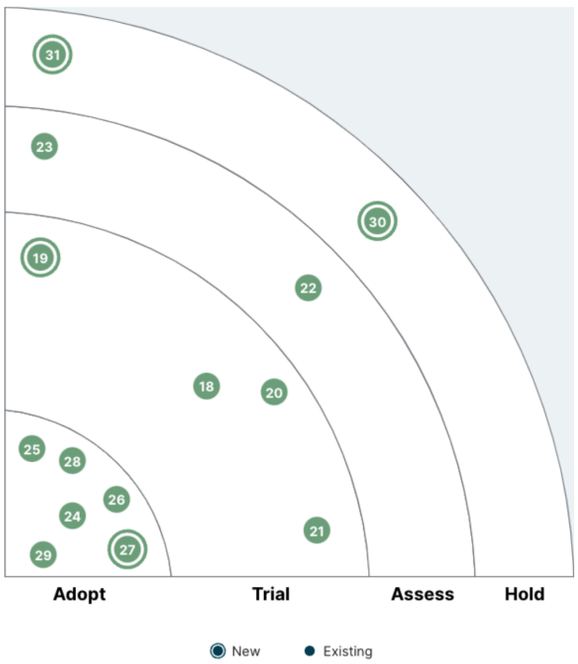
De markt voor Cloud-infrastructuuraanbieders is klein. In korte tijd na de introductie lag AWS zo ver voor op de concurrentie dat het grootste deel van de markt gebruik maakt van deze diensten. Concurrentie heeft dit inmiddels bijgebeend, maar de aanbieders van Cloud-infrastructuren zijn letterlijk op één hand te tellen. Dit heeft een nieuwe, internetbrede, single-point-of-failure geïntroduceerd waardoor een storing bij één Cloud-provider ervoor kan zorgen dat een groot deel van het internet onbereikbaar is geworden.

On-premise

Het is belangrijk om af te stappen van de automatische gedachte om nieuwe projecten onmiddellijk in de Cloud te zetten, en weer serieuze kosten- en technische overwegingen te nemen. Zeker omdat, net als de ontwikkelingen in de Cloud, de ontwikkelingen van on-premise applicaties niet stil heeft gestaan. Platformen als OpenShift en Docker geven eenzelfde niveau van gemak bij het inrichten buiten de Cloud.

Elk project heeft er baat bij om, naast de vele overwegingen voor de inzet van de beste ontwikkeltechnieken, ook dezelfde hoeveelheid overweging te geven aan de infrastructuur waarop deze wordt uitgerold.

Tools



Trial
18. Error Prone Support
19. Sigrid
20. Spring Boot Migrator
21. Testkube

Assess
22. Diffblue Cover
23. SigNoz

Adopt
24. CNCF Trailmap
25. Datafaker
26. Gradle Kotlin DSL
27. k9s
28. Maven Daemon
29. OpenTelemetry

Hold
30. Docker 4 Desktop
31. Lombok

CNCF Trailmap

ADOPT

De CNCF Trailmap [https://github.com/cncf/trailmap] beschrijft de route die leidt van een on-premise monoliet naar een Cloud Native applicatie.

De trailmap kan gebruikt worden om het gesprek te voeren over de Cloud strategy. De trailmap geeft inzicht in het startpunt en de gewenste plaats op de route naar een Cloud Native applicatie. De trailmap kan worden aangevuld met het landschap [https://landscape.cncf.io/] dat een actueel beeld geeft van tools en technieken. Beide tools worden onderhouden door de Cloud Native Computing Foundation (CNCF). De CNCF is opgericht in 2015 en houdt zich sindsdien bezig met container- en cloud-technologie.

De trailmap is een laagdrempelige manier om te controleren waar een organisatie staat en te bespreken waar een organisatie zou willen staan. Hierdoor kan de trailmap op diverse niveaus in de organisatie gebruikt worden, van development teams tot (ICT) management. Samen met het landscape biedt de tool handvatten om de organisatie naar de juiste plek te bewegen. Om deze redenen hebben wij hebben de CNCF trailmap in het adopt quadrant van deze editie van onze Tech Radar geplaatst.

Datafaker

ADOPT

Datafaker [https://www.datafaker.net] is een Java/Kotlin library om testgegevens te genereren die eruit zien als echte gegevens. Met Datafaker kunnen we namen, adressen, telefoonnummers, creditcardnummers, medische gegevens, maar ook "lichtere" gegevens zoals de namen van karakters uit Star Wars of uitspraken uit de Back to the Future film genereren. Het gebruik van Datafaker heeft een aantal voordelen, zoals data die niet te herleiden is naar persoonlijk identificeerbare gegevens, het maken van een ongelimiteerde set van gegevens en gegenereerde gegevens kunnen specifiek voor een land en taal worden gemaakt. Het is mogelijk om Datafaker zelf uit te breiden met nieuwe data.

We plaatsen Datafaker in *Adopt* omdat het een volwassen library is die een toegevoegde waarde heeft voor de kwaliteit testen.

Gradle Kotlin DSL

ADOPT

Gradle is na Maven de meest gebruikte build tool om applicaties en libraries te ontwikkelen op de JVM. Gradle onderscheidt zich van Maven doordat het sneller en vooral flexibeler is. Waar de build in Maven beschreven wordt met een XML-document gebruikt Gradle een Domain Specific Language (DSL). Traditioneel werd alleen Groovy ondersteund, maar sinds een aantal jaar is er ook een Kotlin variant van de DSL beschikbaar. Vanaf Gradle 8.2 zal de Kotlin DSL zelfs de standaard zijn voor nieuwe projecten.

Deze Kotlin variant biedt een aantal voordelen:

1. Om te beginnen is Kotlin een statisch getypeerde taal, terwijl Groovy dynamisch getypeerd is. Hierdoor krijgen we type-safe model accessors als we onze buildscripts schrijven in Kotlin. Dit betekent dat we code completion krijgen in onze IDE (voor nu ondersteunen IntelliJ IDEA en Android Studio dit) als we ons buildscript schrijven. We krijgen voor Groovy buildscripts soms ook code completion in IntelliJ IDEA, maar dat is alleen voor bekende APIs en bijvoorbeeld niet voor third-party plugins. Met Kotlin kan de IDE traditionele code completion bieden gebaseerd op classes en is het niet afhankelijk van specifieke features. Hierdoor werkt code completion ook voor third-party plugins.
2. Herstructuren, refactoring, van buildscripts is nu ook makkelijker vanuit de IDE, want de IDE kan typed code al refactoren. Voor de IDE is het buildscript nu een Kotlin source, dus alles wat we normaal kunnen doen voor het refactoren van Kotlin-code kunnen we nu ook gebruiken voor ons buildscript.
3. We kunnen gebruik maken van Kotlin-eigenschappen zoals delegated properties en null safety. Omdat het buildscript is geschreven in Kotlin kunnen we alles gebruiken wat Kotlin te bieden heeft. We kunnen bijvoorbeeld delegated properties gebruiken om een referentie te krijgen naar een object dat we ook later in het buildscript willen gebruiken.

De stabiliteit van de Kotlin DSL is in de afgelopen jaren dermate verbeterd dat wij de *blip* in deze editie van onze Tech Radar naar *Adopt* hebben verplaatst. Voor bijna alle projecten zal de Kotlin DSL meerwaarde bieden boven de oude Groovy DSL. Het feit dat Gradle de Kotlin DSL nu ook als standaard gebruikt onderschrijft dit. Alleen voor bestaande zeer complexe buildconfiguraties wordt nu nog afgeraden om te migreren.

k9s

ADOPT

k9s [https://k9scli.io/] is een product om Kubernetes clusters te beheren via de command line. Het geeft je de mogelijkheid om zowel je cluster te beheren als real-time inzicht te krijgen in de status van het cluster. Dit kan door middel van het inzien van logbestanden, het wijzigen van deployments en ook verbinding maken met de pods. k9s is een gratis alternatief voor Lens Desktop [https://k8slens.dev/desktop.html] dat inmiddels een commercieel product is geworden, waarvoor de overgangsperiode in januari 2023 is afgelopen. Gezien k9s dezelfde functionaliteit biedt als Lens Desktop, plaatsen wij het op *Adopt*.

Maven Daemon

ADOPT

Maven is een build-tool voor het ontwikkelen van Java-applicaties. We kunnen Maven Daemon [https://github.com/apache/maven-mvnd] gebruiken om onze builds sneller te maken. Maven Daemon heeft verschillende eigenschappen die het mogelijk maken om de build te versnellen:

1. Bij de eerste keer opstarten wordt er een achtergrondproces gestart die de JVM draaiend houdt.
2. De standaardoptie bij het opstarten is dat de multithread-mode wordt gebruikt.
3. Maven Daemon is gebouwd met GraalVM en is een native executable die snel start.

Maven Daemon kan al gebruikt worden door op projecten en is daarom geplaatst in het *Adopt* kwadrant.

OpenTelemetry

ADOPT

OpenTelemetry [https://opentelemetry.io/] beoogt een eenvoudige, universele, vendor-neutrale, loosely coupled oplossing te zijn voor logging, metrics & tracing. Gesteund door de Cloud Native Compute Foundation (CNCF) en de grote spelers in de observability-wereld, heeft dit goede kans op termijn onze projecten te raken. Er zijn tools, API's en SDK's beschikbaar in verschillende talen voor het instrumenteren van applicaties, verzamelen van meetdata en om deze metrics, logs en traces te exporteren. Daarmee wordt het mogelijk de performance en het gedrag van applicaties te analyseren in een backend naar keuze.

Wij zien vooral de mogelijkheden om middels deze taal- en vendor-onafhankelijke standaard eindelijk één oplossing te hebben die herhaaldelijk is toe te passen, ongeacht de lokale situatie. Voor Java is er al een indrukwekkende set aan integraties met bestaande frameworks, waaronder een Java agent om daar snel mee te kunnen starten.

Error Prone Support

TRIAL

Error Prone Support [https://error-prone.picnic.tech/] is een extensie vanuit Picnic voor Google's Error Prone statische analysetool. Het doel is de codekwaliteit binnen een project te verbeteren, onder andere door het controleren van onderhoudbaarheid en consistentie, en het signaleren van veelvoorkomende valkuilen. Error Prone is een plugin voor de Java compiler. Dit maakt snelle feedback mogelijk om veelvoorkomende fouten op te sporen en te repareren. Refaster breidt de functionaliteit verder uit en vervangt suboptimale codepatronen door een voorgeschreven optimale implementatie. Error Prone support heeft bovendien ondersteuning voor Bug Patterns en Refaster regels voor AssertJ, Guava, Streams en andere veelgebruikte bibliotheken. Vooral de Refaster regels zijn welkom, aangezien er al een tijd een issue [https://github.com/google/error-prone/issues/649] open staat waarin Google verzocht wordt hun Refaster templates vrij te geven. De toevoegingen van Picnic reflecteren de technologiekeuzes van Picnic, met duidelijke ideeën hoe oude code-patronen omgezet kunnen worden naar nieuwe patronen. In de toekomst verwacht Error Prone Support prestaties te verbeteren en om het makkelijker te maken Refaster templates te testen.

We waarderen het als bedrijven als Picnic een bijdrage leveren aan Open Source software. Hun toevoegingen van bugchecks en Refaster templates maken het voor iedereen met dezelfde technologie-stack mogelijk om hun code te verbeteren. We raden gebruikers aan om Error Prone Support te proberen op projecten met een laag risico. Om deze reden is Error Prone Support in het *Trial* quadrant van deze editie van onze Tech Radar geplaatst. Zo kunnen ontwikkelaars ervaring

opdoen met de voordelen en risico's, voordat Error Prone Support voor grotere projecten wordt gebruikt.

Sigrid

TRIAL

Sigrid [https://www.softwareimprovementgroup.com/solutions/sigrid-software-assurance-platform/] is een code quality benchmarking tool, gericht op het verbeteren van de kwaliteit van softwarecode door middel van statische codeanalyse. Het bevat benchmarks op basis van best practices en patterns in code, opgebouwd vanuit onder andere open source projecten. Het beoordeelt hoe gescande code hiertegen afsteekt. Sigrid geeft ontwikkelaars en ook management inzicht in de technische kwaliteit van de code. Het richt zich daarbij wat sterker op het management, maar levert waardevolle inzichten voor zowel de ontwikkelaars als de stakeholders. Analyse gebeurt na het aanmaken van een pull-request in de repository. Het kent daarbij geen early warning systeem of een plug-in voor een IDE. Volgens Sigrid zal een plug-in ook niet worden ontwikkeld, er wordt aangeraden om Sigrid naast bijvoorbeeld Sonarlint en Sonarqube te gebruiken. Het wordt bij diverse klanten van JDriven ingezet waarbij we samen met de klant onderzoeken wat de voor- en nadelen zijn. Om deze reden voegen we Sigrid toe aan het *Trial* quadrant van deze editie van onze Tech Radar.

Spring Boot Migrator

TRIAL

Spring Boot Migrator [https://github.com/spring-projects-experimental/spring-boot-migrator] (SBM) is bedoeld voor ontwikkelaars om te helpen bij het migreren van applicaties naar Spring Boot, het upgraden van bestaande Spring Boot-applicaties of het migreren van een applicatie om Spring Boot-features te gebruiken.

Het is een experimenteel Spring-project gemaakt door de ontwikkelaars van het Spring Framework zelf. Met de release van Spring Boot 3 eind 2022 zou de Spring Boot Migrator uitermate nuttig kunnen zijn voor applicaties. Het project gebruikt (deels) OpenRewrite [https://docs.openrewrite.org] recipes voor het migreren en bijwerken van de code.

Het project is in ontwikkeling en heeft momenteel nog enkele beperkingen. Bekijk het GitHub-project om de werkelijk ondersteunde build-tools te zien (op dit moment alleen Maven).

Op dit moment zoekt het project early-adopters. Het wordt ondersteund door de Spring-ontwikkelaars zelf en kan daarom een mooie trial zijn voor ontwikkelingsteams om zo ook feedback richting het project te rapporteren.

Testkube

TRIAL

Testkube [https://testkube.io] is een framework om testen uit te voeren en te coördineren op Kubernetes. Testkube definieert testen als Kubernetes Custom Resources (CRD) en biedt ondersteuning voor tools als Maven en Gradle. Zo is het mogelijk dat integratie testen worden uitgevoerd als er bijvoorbeeld een nieuwe versie van een microservice wordt gedeployd op Kubernetes. Door middel van een dashboard, command-line tools en/of configuratie bestanden is het mogelijk om testen te beheren. Testkube is onderdeel van het Cloud Native Interactive Landscape (CNCF). Het is een product dat het uitvoeren van testen anders doet dan in de Continuous Integration (CI) pipeline en we plaatsen het daarom in *Trial* zodat ervaring kan worden opgedaan met deze andere manier van testen uitvoeren.

Diffblue Cover

ASSESS

Diffblue Cover [https://www.diffblue.com/products/] is in staat regressietest-suites aan te maken, die meedraaien in de continuous integration pipeline. Deze regressietests bieden bescherming tegen ongewenste effecten van code-aanpassingen, zodat deze sneller en eerder in het ontwikkelproces worden gedetecteerd. Diffblue Cover's AI-engine kan snel een suite aan tests afleiden uit bestaande code, die een afspiegeling zijn van de huidige functionaliteit. Door het produceren van grote aantallen unit-integratietests zit de waarde niet in individuele tests, maar in het vermogen van alle tests samen om regressies op te sporen. De gegenereerde tests dekken een grote verscheidenheid aan scenario's af, waaronder uitzonderingsgevallen en raamwerk-code die anders gemist zou zijn (bewust of onbewust) door ontwikkelaars.

Wij geloven dat tools als Diffblue Cover de potentie hebben het werk van een ontwikkelaar aan te vullen. Door het repetitieve handwerk weg te nemen bij het testen van varianten, kan de ontwikkelaar zich richten op de interessante gevallen. Het vermogen om regressies op te sporen kan van onschatbare waarde zijn bij het werken met anderzijds slecht geteste code bases, of wanneer de noodzaak bestaat snel en continu naar productie uit te rollen. Vergeleken met andere AI-code-completion tools spreekt het ons aan dat Diffblue Cover lokaal draait. Maar daar staat tegenover dat de plugin of CLI-tool 5 GB in gebruik neemt, en redelijk aan de prijs is per developer. Om deze redenen plaatsen wij Diffblue Cover in het *Assess* quadrant van deze editie van onze Tech Radar.

SigNoz

ASSESS

SigNoz [https://signoz.io/] is een open source APM (Application Performance Monitoring) platform voor cloud-native applicaties, dat gebruikers in staat stelt om te monitoren, troubleshooten en prestatieproblemen op te lossen in real-time. Het biedt een end-to-end oplossing voor distributed tracing, log management en visualisatie van metrics. Door de overzichtelijke weergave zijn achterliggende relaties bij problemen makkelijker te achterhalen. Daarnaast ondersteunt SigNoz OpenTelemetry waardoor een breed scala aan talen gebruikt kan worden.

SigNoz is een mogelijk alternatief voor DataDog, NewRelic en Prometheus in combinatie met Jaeger en is om deze reden in het *Assess* quadrant van deze editie van onze Tech Radar geplaatst.

Docker 4 Desktop

HOLD

Docker for desktop [https://www.docker.com/products/docker-desktop/] is een product om makkelijk met containers te werken op desktop computer of laptop en wordt vooral gebruikt voor Microsoft Windows en MacOS systemen. De licentie voorwaarden voor Docker for desktop zijn in 2022 gewijzigd, waardoor bedrijven mogelijk geld moeten betalen om Docker for desktop te gebruiken. In 2023 is de overgangsperiode van één jaar verstreken en gaan de licentie voorwaarden nu echt gelden. Daarom plaatsen we Docker for desktop nu op *Hold*. Dit betekent dat voor de inzet van Docker for desktop eerst goed moet worden nagedacht en misschien andere tools een beter alternatief zijn.

Lombok

HOLD

Lombok [https://projectlombok.org/] is een codegenerator om boilerplate code zoals getters, setters en constructors in Java code te laten genereren. De code wordt hoofdzakelijk vanuit annotaties gegenereerd. De wildgroei aan annotaties, de benodigde kennis van de Lombok implementatie en de opkomst van nieuwere Java language features, zoals records, maakt dat JDriven terughoudend is bij het inzetten van Lombok voor nieuwe projecten.

Er zijn veel Lombok-annotaties, waarvan niet elke developer zich bewust is van de consequenties. Lombok genereert code waarvan een developer moet weten dat deze er is, maar het feitelijk niet kan zien omdat het in de codegeneratie-stap onzichtbaar aan de byte code wordt toegevoegd. Als developer zijn we tien keer zo vaak code aan het lezen als aan het schrijven. Dat maakt juist dat we het lezen moeten vergemakkelijken om het zo makkelijker schrijfbaar te maken. In het boek "Clean Code" van Robert C. Martin is dit ook al besproken.

Bedenk bij het gebruik van bijvoorbeeld @Getter, @Setter, @NoArgsConstructor en @Builder of het daadwerkelijk bijdraagt aan beter leesbare code, of dat het juist belemmerend is bij het lezen van de code. Het gebruik van meerdere Lombok-annotaties kan overlappende functionaliteit hebben, waardoor de code overbodige annotaties bevat die de leesbaarheid niet verhogen. Soms is het volledig uitschrijven van de Java code in een POJO (Plain Old Java Object) zonder deze Lombok-annotaties beter om de leesbaarheid te verhogen.

Wij als JDriven adviseren om simpele voorbeelden die vaak met Lombok zijn uitgeschreven niet klakkeloos in projecten als voorbeeld te gebruiken, maar goed na te denken over de toepassing van Lombok-annotaties. Om deze reden plaatsten wij Lombok in het *Hold* kwadrant van deze editie van onze Tech Radar.

Commit. Develop. Share.

Commit.
Develop.
Share.