JDriven Tech Radar Najaar 2022

Onze kijk op recente ontwikkelingen in het veld

6e 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.

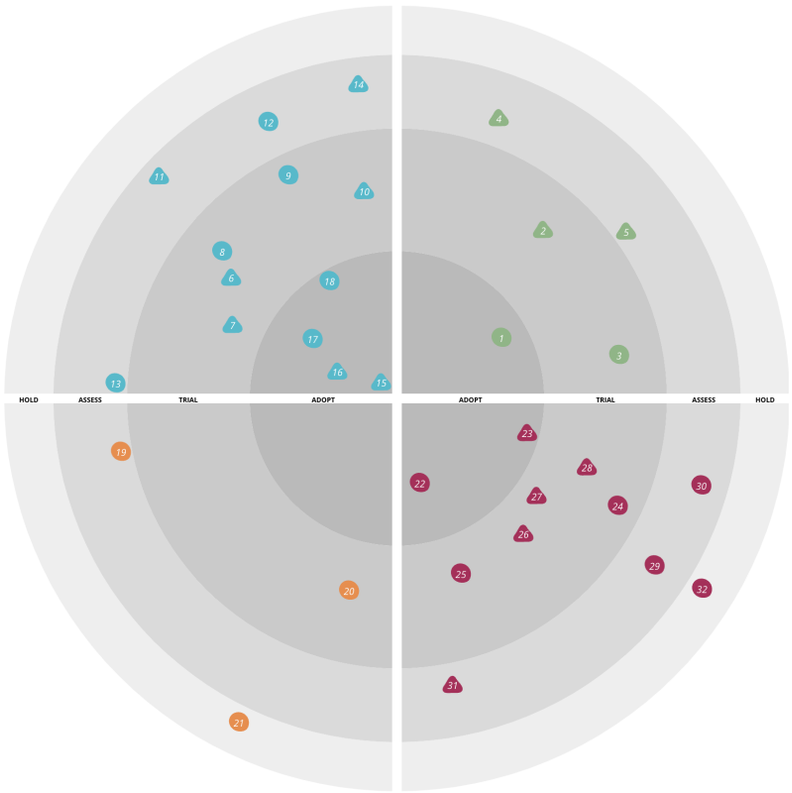
De kwadranten verdelen de verschillende onderwerpen in categorieën.

- **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.

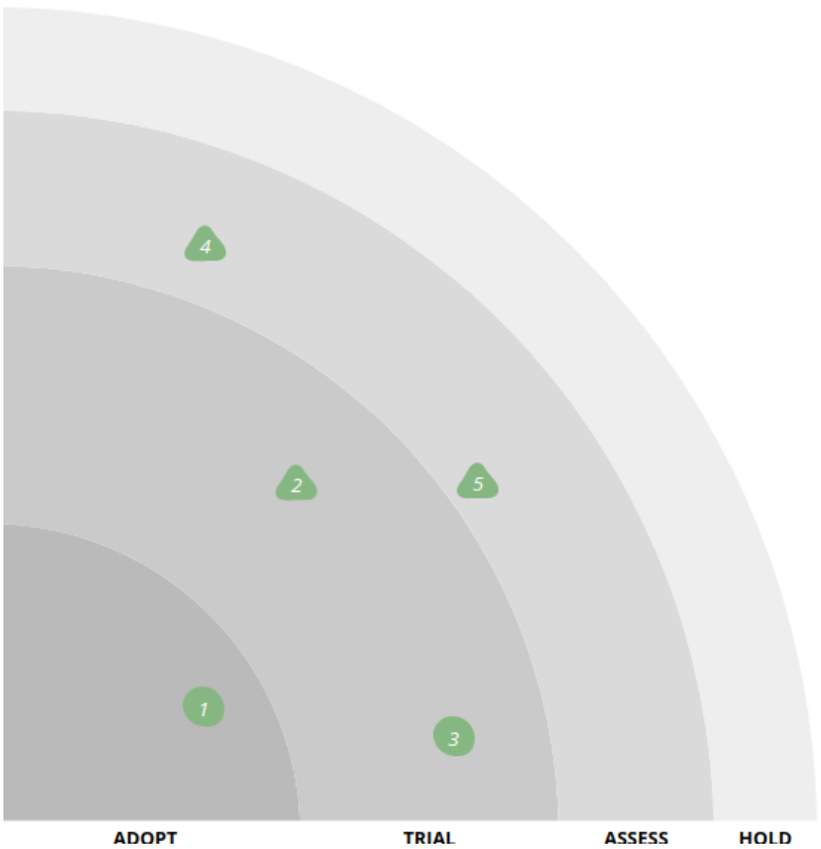
In de volgende secties zullen we onze kijk op de recente ontwikkelingen per kwadrant toelichten.



Tools	Talen & frameworks
ADOPT CNCF Trailmap Datafaker Renovate Spotless **TRIAL** Error Prone Support Gradle Kotlin DSL Kustomize OpenTelemetry Spring Boot Migrator **ASSESS** Diffblue Cover GitHub copilot Maven Daemon SigNoz	**ADOPT** Kotlin coroutines **TRIAL** Spring Modulith Unity3D **ASSESS** F# Virtual threads
Platforms	**Technieken**
ADOPT - **TRIAL** GraalVM **ASSESS** On-premise **HOLD** CircleCI	**ADOPT** Testing diamond Tinkering **TRIAL** Low-code Mob Programming SBOM WebAuthn Weighing developer experience **ASSESS** Data Oriented Programming Linked data Risk storming **HOLD** SAFe

Commit. Develop. Share.

Talen & frameworks



ADOPT

1. Kotlin co-routines

TRIAL

2. Spring Modulith
3. Unity3D

ASSESS

4. F#
5. Virtual threads

Kotlin coroutines

ADOPT

Kotlin Coroutines [https://kotlinlang.org/docs/coroutines-guide.html] is een first-class Kotlin library voor het schrijven van leesbare en onderhoudbare asynchrone, non-blocking code. De library is ontwikkeld door JetBrains en bouwt verder op de low-level *suspending function* APIs in Kotlin.

Wanneer je een applicatie schrijft in Kotlin waarbij je gebruikmaakt van asynchrone, non-blocking code, dan is Coroutines de de facto standaard. Het is stabiel, wordt goed ondersteund door IntelliJ en maakt het schrijven van multi-threaded code een stuk eenvoudiger en beter onderhoudbaar.

Spring Modulith

TRIAL

Spring Modulith [https://spring.io/projects/spring-modulith]

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 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.

Unity3D

TRIAL

Unity3D is een marktleider op het gebied van game engines. Unity is uitgebracht in 2005, initieel exclusief voor macOS, maar is al jaren ook beschikbaar op Linux en Windows. Over de jaren heen heeft Unity3D zich bewezen als een developer-vriendelijke game engine, met een eerlijk licentiemodel, en ontwikkeld volgens het "build once, deploy anywhere" principe. Nu wordt het gebruikt door miljoenen developers wereldwijd, die er jaarlijks duizenden games en applicaties mee ontwikkelen. Unity3D heeft zich afgelopen jaren ook naar andere sectoren vertakt, zoals de animatie-, auto- en de bouw- & constructiesectoren. Daarmee heeft Unity3D een complete ontwikkeloplossing voor interactieve 3D-toepassingen. Deze uitbreiding heeft voor veel nieuwe gebruikers en hernieuwde interesse in de engine gezorgd.

Het is ons opgevallen dat veel organisaties die met AR/VR of 3D-visualisaties van hun data experimenteren, voor Unity3D kiezen. De toepassing van Unity3D buiten de traditionele game-omgeving, en de toenemende noodzaak om 3D-data te visualiseren, heeft ons overgehaald om Unity3D in onze tech radar van dit voorjaar toe te voegen. Als je als organisatie aan het innoveren bent met 3D-data, of oplossingen zoekt voor om jouw AR/VR-concepten tot leven te brengen, dan is Unity3D zeker een goede keus.

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.

Laten we dit toelichten aan de hand van het volgende voorbeeld. De welbekende functie om de reeks van Fibonacci te berekenen kan in F# als volgt geschreven worden:

```
let fib n =
    let rec loop acc1 acc2 n =
        match n with
        | 0 -> acc1
        | 1 -> acc2
        | _ -> loop acc2 (acc1 + acc2) (n - 1)
    loop 0 1 n

printfn "%i" (fib 33) // output: 3524578
```

*De `fib` functie definieert een inner **rec**ursieve `loop` functie om de berekening te doen. Deze functie gebruikt weer pattern **match**ing om te bepalen of de `loop` functie nogmaals moet worden aangesproken. Als laatste gebruikt de compiler tail-call optimization om de **rec**ursieve `loop` functie om te zetten naar een for-loop. Met enkele regels code zijn al drie FP-begrippen aan bod gekomen!*

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.

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).

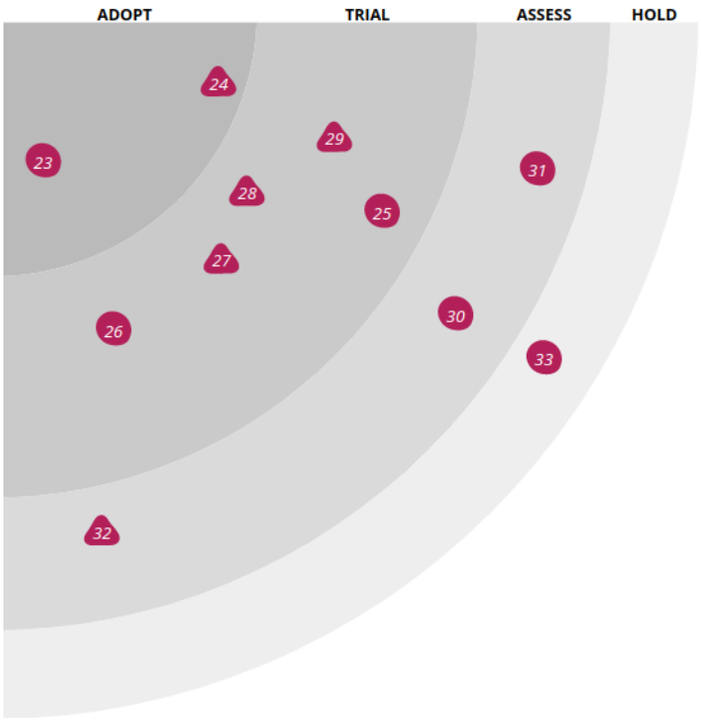
Dit concept om threads efficiënter te gebruiken door deze meerdere taken 'tegelijk' te laten uitvoeren is niet nieuw. Vroege versies van macOS en Windows maakten al gebruik van zogeheten cooperative (non-preemptive) multitasking, waarbij meerdere processen/taken op een beperkt aantal threads konden draaien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

23. Testing diamond
24. Tinkering

TRIAL

25. Lowcode
26. Mob Programming
27. SBOM
28. WebAuthn
29. Weighing developer experience

ASSESS

30. Data Oriented Programming
31. Linked data
32. Risk storming

HOLD

33. SAFe

Testing diamond

ADOPT

Om het testen van software naar een hoger niveau te tillen wordt vaak gekozen voor het groeperen van verschillende soorten testen. Een veelgebruikte onderverdeling is in de volgende drie categorieën:

- UI- en end-to-end testen
- Integratietesten
- Unittesten

In een traditionele software-omgeving ligt de focus op het testen van de units; daarna volgen de integratietesten en als laatste de UI-testen. Vaak wordt dat weergeven in een piramidevorm. De belangrijkste testen vormen de voet van de piramide, de andere testen vullen de tweede laag en de top. Het gebruik van de 'testpiramide' is een goede manier om monolieten door te testen.

In het huidige software-landschap is het gebruik van de testpiramide echter niet toereikend, doordat er vaak gebruikgemaakt wordt van meerdere microservices. De eerdergenoemde groeperingen zijn echter nog steeds relevant, maar een goede werking van de onderlinge verhouding tussen de services is van groter belang dan de afzonderlijke units. Wanneer je dit uittekent, verandert de piramide naar een andere vorm: de diamant. Vanuit dat oogpunt zijn de integratietesten nu het hart van je test suites, de andere testgroepen zijn secundair. De 'testdiamant' is daarom een uitstekende methode om complexe multi-service architecturen testbaar te maken.

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 door alleen het juiste te leren. Als we in een nieuw onderwerp duiken kan het best overweldigend zijn, daarom moeten we onszelf beperken in wat we willen leren. We kunnen hiervoor de "zones of proximal development" gebruiken. Deze bestaat uit drie zones: wat we al weten, wat we kunnen leren en wat we niet kunnen leren. Stel dat we een nieuw framework geschreven in Java/Kotlin willen leren. We kennen al Java, maar geen Kotlin, dus wat we al weten is Java. Dan kunnen we leren om het framework in Java te gebruiken. En wat we niet tegelijk kunnen leren is Kotlin. Op het moment dat we weten hoe het framework in Java werkt, kunnen we de volgende stap nemen, maar we moeten niet alles in één keer leren.

In onze industrie verschijnen en verdwijnen regelmatig frameworks, maar meestal zijn deze gebaseerd op algemene concepten. Het leren van deze concepten is belangrijk, omdat het dan makkelijker is om nieuwe frameworks te leren. Concepten veranderen niet zo vaak als technologieën. Voorbeeld van concepten zijn object-georiënteerd programmeren, domain driven design en design patterns.

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.

Low-code

TRIAL

Middels een low- of een no-code platform kunnen businessgebruikers applicaties ontwikkelen zonder te coderen. Het “programmeren” gaat met de hulp van user interfaces om processen en bronnen aan elkaar te modelleren. Bij een no-code platform wordt alles via de user interface gedaan. Bij een low-code platform is het wel mogelijk om code toe te voegen om het doel te bereiken.

Low-code en ook no-code platformen zitten in de lift en vormen een mogelijke oplossing voor de groeiende vraag naar ICT-oplossingen en het gebrek aan software engineers.

De voordelen van het gebruik van deze platformen zitten in tijd en beschikbaarheid van schaarse resources zoals programmeurs.

Het gebruik van deze platformen heeft ook nadelen. Zo zit je vast aan de betreffende vendor die het platform levert en is de functionaliteit die je kan realiseren beperkt door de grenzen van wat het platform kan leveren. Ook de integratie met externe systemen kan lastig zijn, zeker wanneer er niet over een standaardprotocol gecommuniceerd wordt. Bovendien moet je je er als bedrijf bewust van zijn dat wanneer er eenmaal voor een platform gekozen is, het lastig is om te migreren naar een andere oplossing.

Low-code en no-code platformen zijn geschikt om snel een MVP op te leveren. Belangrijk hierbij is dat de processen die geautomatiseerd worden relatief simpel zijn en weinig afhankelijkheden hebben. Wanneer je zelf aan de slag wilt met een low-code platform lees dan ook dit artikel [https://www.thoughtworks.com/insights/articles/making_case_low-code_platforms] van Thoughtworks.

Mob Programming

TRIAL

Mob Programming [https://en.wikipedia.org/wiki/Mob_programming] is een extremere vorm van pair programming waarbij het hele team tegelijk aan dezelfde feature werkt. Tijdens het mob programmen is er één driver achter het toetsenbord en de rest van het team geeft de driver instructies.

Hoewel dit op het eerste gezicht inefficiënt lijkt zijn wij van mening dat het ook enkele voordelen biedt:

- Kennisdeling; Doordat het hele team ermee bezig is, is de kennis goed verdeeld
- Training; Junioren krijgen veel ondersteuning op deze manier
- Geen review meer nodig
- Meerdere perspectieven

Hoewel het niet voor ieder team of probleem zal werken hebben wij er in de praktijk goede ervaringen mee.

SBOM

TRIAL

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].

WebAuthn

TRIAL

Web Authentication API (ook wel WebAuthn [https://webauthn.guide] genoed) 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.

De grote belofte van WebAuthn is authenticatie zonder een wachtwoord. Wachtwoorden zijn niet een veilige optie voor beveiliging. Als hackers ons wachtwoord stelen kunnen ze het zonder probleem gebruiken en zich voordoen als ons. 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. Verschillende besturingssystemen (iOS, Android, Windows, macOS) hebben al ondersteuning waardoor we bijvoorbeeld touch ID op MacBooks en

gezichtsherkenning en vingerafdrukherkenning op PC's kunnen gebruiken. Authenticatieproviders zoals Keycloak bieden ook standaard ondersteuning voor WebAuthn.

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.

Weighing developer experience

TRIAL

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 indivudu 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.

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.

Linked data

ASSESS

Linked Data is een techniek waarbij data aan elkaar gekoppeld en geïnterpreteerd kan worden zonder dat de data verplaatst hoeft te worden. Met Linked Data bouw je een gestructureerd netwerk op van je data, en dat kan dan gebruikt worden voor rijkere analyses, en om makkelijker je data, met context, te delen.

In academische kringen gaat de term 'Linked Data' al jaren rond. Alleen is er de afgelopen tijd een toegenomen interesse voor de techniek, en is Linked Data het academische perspectief ontstegen. De technologie ontwikkelt zich razendsnel door. Dat komt doordat er allerlei standaarden zijn gedefinieerd om basiszaken te beschrijven. Zulke standaarden beschrijven bijvoorbeeld hoe je een adres kan beschrijven en interpreteren.

Dankzij de inmiddels volwassen standaardisatie is Linked Data makkelijker toe te passen. Het Kadaster stelt bijvoorbeeld publieke Linked Data endpoints beschikbaar zodat je hun datasets aan elkaar kunt verbinden. Als je aan het onderzoeken bent hoe je je data kunt opslaan op een manier die makkelijk interpreteerbaar is voor machines, en die ook gekoppeld kan worden aan andere datasets, dan stellen wij voor om Linked Data uit te proberen.

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

TRIAL

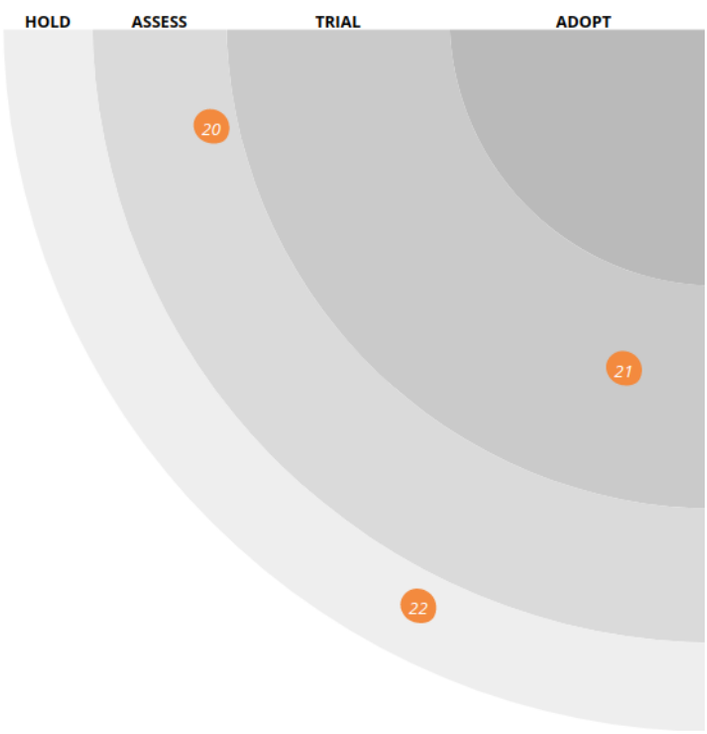
21. GraalVM

ASSESS

20. On-premise

HOLD

22. CircleCI



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.

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 (die waar 90% van de Cloud afnemers gebruik van maken) 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.

CircleCI

HOLD

CircleCI [https://circleci.com/] is een Continuous Integration en Continuous Deployment (CI/CD) cloud platform waarmee automatisch software gecompileerd, getest en gedeployed kan worden. CircleCI ondersteunt zeer complexe build pipelines op meerdere besturingssystemen (Linux, macOS en Windows) en hardware platforms (o.a. x86_64 en ARM).

Via integraties met onder andere GitHub, GitLab en Bitbucket kan automatisch bij elke commit een nieuwe build worden gestart. Builds draaien binnen Docker containers en pipelines worden geconfigureerd met een YAML file binnen de Git repository van een project.

Hoewel CircleCI een prima tool is, biedt het voor de meeste projecten weinig tot geen toegevoegde waarde boven tools als GitHub Actions, GitLab CI en Bitbucket Pipelines, terwijl er wel extra kosten aan verbonden zijn. Pas als een pipeline van een project baat heeft bij een hoge mate van parallellisatie of wanneer builds op verschillende (hardware) platforms nodig zijn, raden wij aan om naar CircleCI te kijken.

Tools

ADOPT

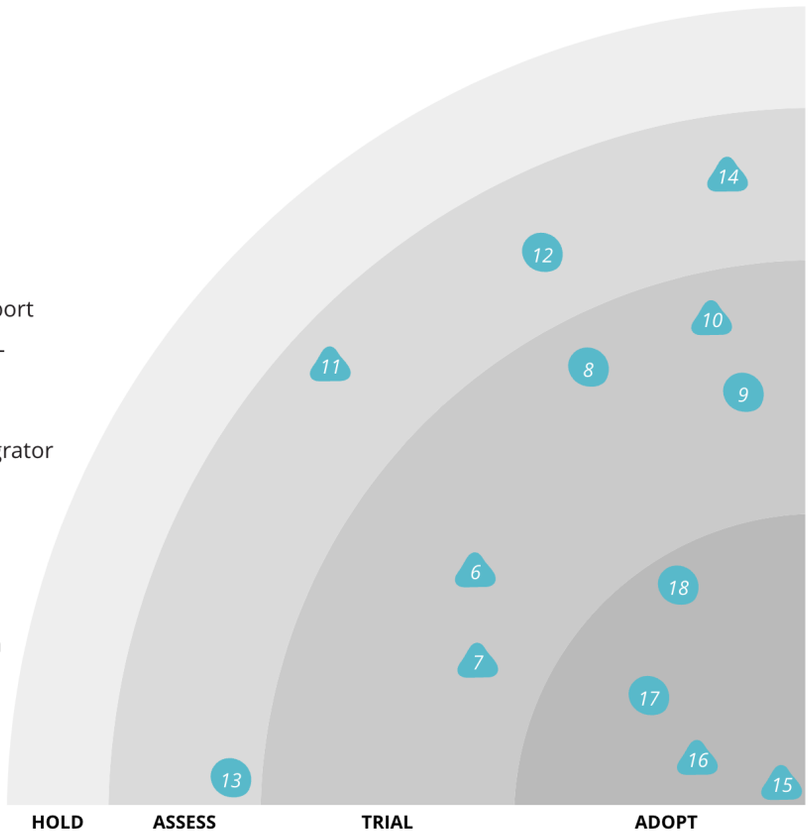
15. CNCF Trailmap
16. Datafaker
17. Renovate
18. Spotless

TRIAL

6. Error Prone Support
7. Gradle Kotlin DSL
8. Kustomize
9. OpenTelemetry
10. Spring Boot Migrator

ASSESS

11. Diffblue Cover
12. GitHub co-pilot
13. Maven Daemon
14. SigNoz



CNCF Trailmap

ADOPT

De Cloud Native Computing Foundation (CNCF) is opgericht in 2015 en houdt zich sindsdien bezig met container- en cloud-technologie. Onder andere Kubernetes en Helm vallen onder de opensource-projecten van de CNCF. Naast de vele projecten ontwikkelt de CNCF ook tools om projecten te helpen met Cloud Native development, biedt de organisatie cursussen en certificeringen aan en organiseert events.

De CNCF houdt een landschap [https://landscape.cncf.io/] bij van tools gerelateerd aan Cloud Native Development gesorteerd op onderwerp. Daarnaast heeft de CNCF de CNCF Trailmap [https://github.com/cncf/trailmap] ontwikkeld [https://github.com/cncf/trailmap/blob/master/CNCF_TrailMap_latest.pdf]. De Trailmap (eventueel aangevuld met het landschap) kan helpen bij het ontwikkelen van een Cloud Native Strategy.

De Trailmap geeft een goed overzicht van de route die leidt van een on-prem monoliet naar een Cloud Native applicatie. Zoals met iedere Trail gaat het niet zozeer om de bestemming, maar meer om de reis. De Trailmap is een handige tool om het gesprek over de Cloud strategy te voeren. Binnen teams kan de tool helpen om te bepalen waar een project staat, waar het zou moeten staan en hoe het project daar kan komen. Binnen de organisatie kan de Trailmap helpen om de juiste tools en technieken te selecteren om projecten optimaal te ondersteunen.

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 makkelijk 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:

1. Er zijn geen problemen dat de gegevens herleidbaar zijn naar persoonlijk identificeerbare gegevens. Alle gegevens zijn gegenereerd en niet gebaseerd op echte gegevens, dus het kan geen persoonlijke gegevens bevatten. Een alternatief zou kunnen zijn het gebruik van geanonimiseerde gegevens. Maar als het proces van anonimiseren niet goed is geïmplementeerd kunnen nog steeds gegevens herleidbaar zijn.
2. Het is heel makkelijk om een ongelimiteerde set van gegevens te maken. We moeten alleen aangeven hoeveel gegevens we willen en Datafaker zal de data genereren.
3. De gegenereerde gegevens kunnen locale-specifiek zijn. Datafaker maakt het mogelijk om gegevens te genereren voor een locale. Namen kunnen bijvoorbeeld worden gegenereerd aan de hand van een Nederlandse locale, maar ook voor een Arabische locale. Hierdoor is het mogelijk om gegevens te gebruiken voor het testen van onze code, waar we normaal gesproken niet snel zelf zouden achterkomen.

We kunnen Datafaker inzetten in onze unittesten als we willekeurige waarden willen gebruiken om onze productiecode te testen en waarbij de willekeurige waarden wel een betekenis hebben. Als we bijvoorbeeld een methode hebben die een argumentnaam heeft met type `String`, dan zouden we dat kunnen testen met een waarde van allemaal willekeurige karakters. Maar het heeft meer betekenis als we willekeurige namen kunnen gebruiken en daar gebruiken we Datafaker voor. Datafaker hoeft niet alleen voor unittesten gebruikt te worden, maar we kunnen het ook gebruiken om CSV, JSON, SQL, YAML en XML bestanden te genereren.

We kunnen Datafaker zelf ook uitbreiden met "custom providers". Deze providers zijn verantwoordelijk voor het genereren van willekeurige waarden vanuit een set van vaste waarden. Dus als we een provider missen die Datafaker nog niet heeft, dan kunnen we deze zelf schrijven en gebruiken.

Renovate

ADOPT

Renovate [https://docs.renovatebot.com/] is een opensource-tool voor het automatiseren van dependency updates, vergelijkbaar met Dependabot welke we al in het voorjaar van 2020 behandelden. Bovenop wat we al gezien hebben van Dependabot, biedt Renovate ons een eenvoudige flow voor het oppikken van nieuwe projecten, met minimale configuratie. Maar waar Renovate zich in onderscheidt, is de officiële ondersteuning voor meerdere platformen zoals Azure, BitBucket & GitLab. Dit maakt het een aantrekkelijke optie voor iedereen die geen GitHub gebruikt, nadat Dependabot door GitHub is opgekocht en geïntegreerd.

Wij willen Renovate in het bijzonder uitlichten voor iedereen die geen GitHub gebruikt, om dezelfde voordelen te genieten op andere platforms. Gegeven wat we het afgelopen jaar hebben gezien met kwetsbaarheden met hoge gevoeligheid is er genoeg reden om bij te blijven, en automatisering helpt ons daarbij.

Spotless

ADOPT

Spotless [https://github.com/diffplug/spotless] is een tool voor het controleren (en toepassen) van code formatting. Waar spotless zich in onderscheidt, is het feit dat het een build tool plugin is dat meerdere talen ondersteunt. Hierdoor is het makkelijk te integreren in projecten en pipelines.

Om deze reden zien wij Spotless als een goede oplossing om discussies over code style & formatting uit het team weg te nemen, zeker met de opkomst van meer diverse teams en ontwikkelomgevingen.

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

Gradle Kotlin DSL

TRIAL

Naast Maven is Gradle een build tool om applicaties en libraries te ontwikkelen. We gebruiken een Domain Specific Language (DSL) om onze build te beschrijven in een build script. Deze DSL is gebaseerd op de Java API van Gradle. We kunnen ervoor kiezen om de DSL te schrijven in Groovy of Kotlin. De code van het script wordt gecompileerd en dan uitgevoerd door Gradle. Traditioneel werd alleen Groovy ondersteund, maar we krijgen een hoop voordelen als we de Kotlin variant kiezen:

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 build scripts 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 build script schrijven. We krijgen voor Groovy build scripts 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 build scripts is nu ook makkelijker vanuit de IDE, want de IDE kan typed code al refactoren. Voor de IDE is het build script nu een Kotlin source, dus alles wat we normaal kunnen doen voor het refactoren van Kotlin-code kunnen we nu ook gebruiken voor

ons build script.

3. We kunnen gebruik maken van Kotlin-eigenschappen zoals delegated properties en null safety. Omdat het build script is geschreven in Kotlin kunnen we alles gebruiken wat Kotlin heeft te bieden. We kunnen bijvoorbeeld delegated properties gebruiken om een referentie te krijgen naar een object dat we ook later in het build script willen gebruiken.

De Kotlin DSL is al goed bruikbaar in projecten die Gradle gebruiken. De developer experience voor het schrijven van build scripts is beter, vooral als IntelliJ IDEA of Android Studio wordt gebruikt.

Kustomize

TRIAL

Kustomize [https://kustomize.io/] is een template engine voor Kubernetes configuraties. Het maakt het mogelijk om een eenduidig declaratief systeem te gebruiken om de volledige configuratie te specificeren voor meerdere deployment targets, zoals *test*, *staging* en *productie*. Kustomize doet dit door middel van een systeem van lagen van templates, de overlays. Elke overlay heeft de mogelijkheid de details van de laag erboven in te vullen of aan te passen. Het gebruik van Kustomize verenigt dit templatesysteem voor de gehele applicatiecontext, en wordt zowel voor in-house tools als voor off-the-shelf applicaties gebruikt. Bovendien is Kustomize volledig geïntegreerd in de standaard Kubernetes `kubectl` tool sinds versie 1.14. Dit maakt het gemakkelijk om Kustomize toe te passen in bestaande systemen voor continuous deployment. Door zijn aard als templatesysteem komt de syntax van Kustomize in hoge mate overeen met de reguliere Kubernetes configuratiebestanden.

Hoewel de templating syntax voor Kustomize af en toe gecompliceerd kan voorkomen, maakt de gelaagde architectuur het eenvoudig om een reguliere Kubernetes configuratie stap-voor-stap aan te passen aan Kustomize. Dit zorgt er ook voor dat het eenvoudiger is om overzicht te houden over de verschillen in configuratie van de deployments. Omdat de reguliere Kubernetes configuratie (met wat kleine aanpassingen) ook een correcte Kustomize configuratie is, zitten er weinig tot geen nadelen aan het uitproberen van Kustomize.

OpenTelemetry

TRIAL

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.

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.

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.

GitHub copilot

ASSESS

GitHub Copilot [https://copilot.github.com/] is een codegeneratie-assistent en plugin aangedreven door Codex, een nieuw AI-systeem van OpenAI. Door de AI getraind op code in publieke GitHub-repositories is GitHub Copilot beter in het genereren van context-gerelateerde code dan andere code assistants. Hierdoor kan het code of tekst genereren in elke context, zij het documentatie, commentaar, functies of code.

Github Copilot stuurt een deel van de data in de huidige file naar de server, waar deze door automatische processen wordt geanalyseerd en door de AI wordt gehaald voor suggesties. Alle stappen worden opgeslagen voor toekomstige voorspellingen. Aangeraden wordt om voorzichtig om te gaan met tool zoals deze.

Voorlopig is er geen license en worden gebruikers toegelaten op basis van een wachtlijst.

Maven Daemon

ASSESS

Maven is een populaire 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. Dit betekent dat bij volgende builds de JVM niet meer opgestart hoeft te worden, maar dat een al draaiende JVM gebruikt kan worden. Ook de Just-In-Time (JIT) optimalisaties worden bewaard en kunnen opnieuw gebruikt worden voor volgende build-executies. De classloaders van plugins worden gecachet, zodat ze maar één keer ingelezen hoeven te worden.
2. De standaardoptie bij het opstarten is dat de multithread-mode wordt gebruikt. Normaal gesproken kunnen we Maven opstarten met de optie `-T` of `--threads` om de build met parallelle threads te starten. Dit is een groot voordeel voor multimodule-projecten, want het betekent dat de modules parallel gebouwd kunnen worden. De totale buildtijd wordt nu bepaald door de module die de meeste tijd kost en niet meer de optelsom van de buildtijd van alle modules. Om de logging overzichtelijk te houden zorgt Maven Daemon ervoor om alle output van elke module bij te houden en de voortgang te tonen voor elke project op een eigen regel. Als er tijdens de build een fout optreedt kunnen we nog steeds wel de complete logging zien.
3. Maven Daemon is gebouwd met GraalVM en is een native executable die snel start.

Maven Daemon kan al gebruikt worden door individuele ontwikkelaars op hun projecten.

SigNoz

ASSESS

SigNoz [https://signoz.io/] is een open source APM (Application Performance Monitoring) voor metrics, traces en logs. SigNoz is een mogelijk alternatief voor DataDog, NewRelic en Prometheus in combinatie met Jaeger. Er zijn veel voordelen om metrics, traces en logs in een single pane te combineren. Zo biedt het overzicht en zijn achterliggende relaties bij problemen makkelijker te achterhalen. SigNoz is ook beschikbaar om zelf te hosten, dus ook geschikt voor systemen waarbij de data niet naar buiten mag. Omdat SigNoz de standaard OpenTelemetry ondersteunt, kan een breed scala aan talen worden gebruikt.

Commit.
Develop.
Share.