JDriven Tech Radar voorjaar 2021

Onze kijk op recente ontwikkelingen in het veld

Commit to know.

Develop to grow.

Share to show.

jdriven

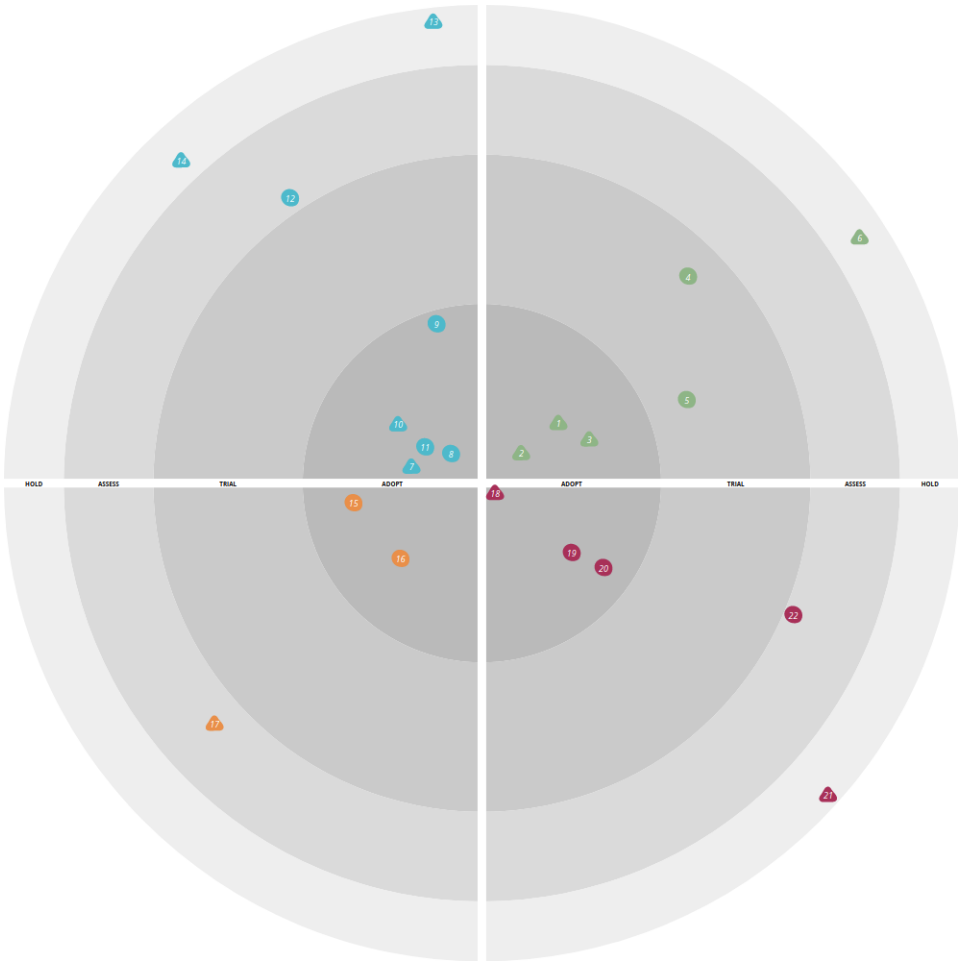
Introductie

Onze ervaren specialisten werken dagelijks mee aan tal van softwareprojecten in Nederland en zijn betrokken in wereldwijde communities. 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. Tevens raden zij iedereen aan een eigen radar op te stellen.

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.



Commit. Develop. Share.

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.

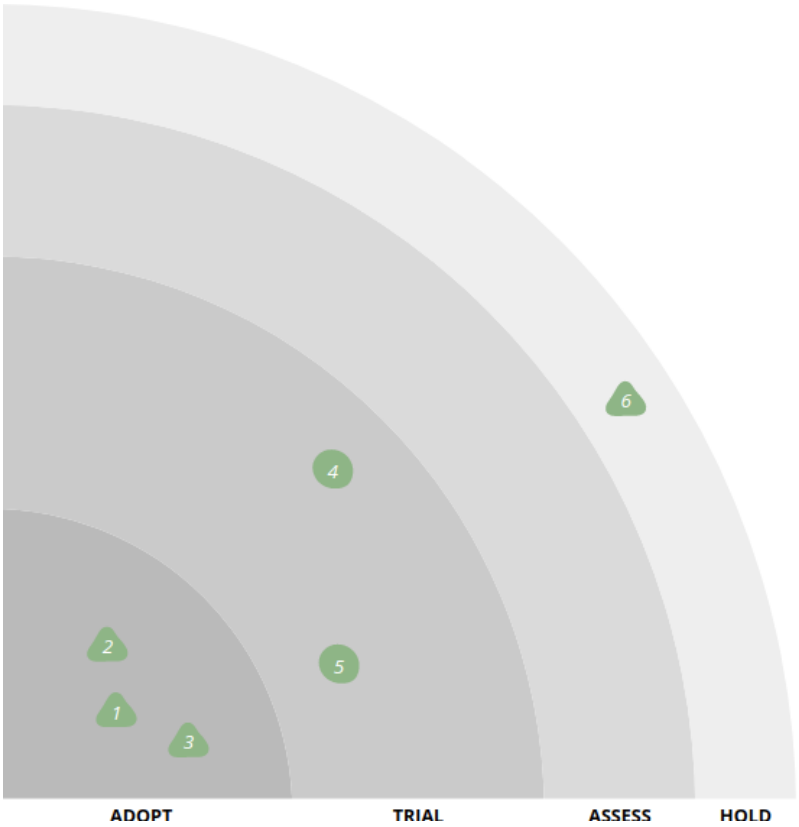
- **Talen & frameworks** die je ondersteunen bij de ontwikkeling van software
- **Platformen** waarop je software kunt uitvoeren
- **Technieken** die je helpen betere software te maken
- **Tools** ter ondersteuning van je ontwikkel- en delivery-proces

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.

Talen & frameworks



ADOPT

1. Apache Ignite
2. Java 16 (altijd latest)
3. Kotlin + Arrow

TRIAL

4. Micronaut
5. Quarkus

HOLD

6. Java + Vavr

Apache Ignite

Adopt

Apache Ignite is een gedistribueerde database voor high-performance computing met de snelheid van in-memory operaties. Daarbij behoort Apache Ignite tot een van de snelste JVM-gebaseerde caching oplossingen. Apache Ignite is een veelzijdig product. Het kan worden ingezet als een standalone partition tolerante in-memory key-value data store. Hierbij kan Ignite zo worden ingesteld dat de data wordt gepersisteerd op disk of in een externe database. Ook kan het worden ingezet als een in-memory SQL database tussen de applicatie en een omvangrijke relationele database, zodat data sneller beschikbaar kan worden gemaakt voor de applicatie. Tevens is het mogelijk om Apache Ignite te gebruiken als een horizontaal (tot op zekere hoogte lineair) computational grid. Aanvullend biedt Apache Ignite atomaire functies om concurrency en eventually consistent vraagstukken op te lossen. Tot slot is het ook mogelijk om Apache Ignite in te richten als een serverless platform voor uitvoering van functies.

Ofschoon de toepassing van Apache Ignite veelzijdig is, geldt wel dat een rechtmatige toepassing doorgaans geavanceerde en specialistische cases betreft. Kennis van gedistribueerde systemen, het CAP-theorema, stale data en concurrency is een vereiste.

Java 16, en andere niet-LTS releases

Adopt

Met de nieuwe releasecadans van Java komt er elk half jaar een nieuwe release uit, en elke drie jaar een Long-Term Support release. LTS releases krijgen zeker vijf jaar ondersteuning vanuit Oracle, waar niet-LTS releases ondersteund worden tot de volgende release. Niet-LTS releases kun je dan ook zien als kleine incrementele verbeteringen, en LTS releases als een bundeling van alle incrementele verbeteringen sinds de vorige LTS.

Wij zien dat verschillende van onze klanten lang vasthouden aan oudere releases van Java. Java 8 uit 2014(!) is daarbij geen uitzondering en sommigen ronden pas net de migratie naar Java 11 af, bijna drie jaar na de release. Ook adoptie van zeker de eerdere niet-LTS versies bleef lange tijd achter, en voor die eerste niet-LTS releases waren er slechts kleine voordelen ten opzichte van de LTS.
Met de komst van Java 16 zien we echter steeds meer features die developers op Java 11 en daarvoor nog altijd moeten missen: Switch expressions, Text blocks, Pattern matching, Records, en ook vele runtime-verbeteringen. Daarnaast wordt Java 16 ondersteund tot aan het uitkomen van LTS Java 17, dus zelfs risico-averse organisaties kunnen alvast updaten met een duidelijk pad naar de volgende LTS.

Wij trekken het echter breder en raden aan om altijd op de laatste release van Java te werken. Wij zijn van mening dat het actueel houden en patchen van uw technologieën onderdeel hoort te zijn van het normale bedrijfsproces. Bij blijven in Java versies dwingt je namelijk ook tot goede hygiëne in andere opzichten. Een voorbeeld daarvan is dat dan ook de laatste versies kunnen worden gebruikt van build tools zoals Gradle en Maven en van frameworks zoals Spring Boot 2.5. Deze incrementele stappen worden aanzienlijk makkelijker als het routine begint te worden om met regelmaat base images, build pipelines en development tools bij te werken. Bijkomend voordeel is dat wanneer het vervallen van support of een nieuwe CVE je dwingt tot updaten, de stappen een stuk sneller te nemen zijn.

Kotlin + Arrow

Adopt

Arrow is een functional programming (FP) library voor Kotlin. Het biedt verschillende erg nuttige toevoegingen aan de taal in de vorm van datastructures. Optional en Either zijn hier een voorbeeld van met tevens enkele krachtige manieren om deze te gebruiken zoals de Either.fx (equivalent van de scala for-yield). Tevens voegt dit een boel nuttige extensies toe aan de Iterable class. Doordat Arrow gebruik kan maken van de Kotlin extension functions, integreert het erg goed met de basistaal en voelt het meer als een uitbreiding in plaats van een losstaande library.

Kotlin ondersteunt van nature (voornamelijk door betere defaults) al iets meer de FP programmeerstijl. Echter mist het nog een aantal datastructuren en goede manieren om hiermee om te gaan. Arrow is een volwassen library dat dit probleem op een goede manier oplost en zich erg goed weet te integreren met de taal. Hiermee is Arrow een uitstekende keuze om meer FP mogelijkheden toe te voegen aan kotlin.

Micronaut

Trial

Micronaut is een modern JVM full-stack framework voor het bouwen van eenvoudig te testen microservices en serverless applicaties.
Het IT-landschap is in de afgelopen jaren sterk veranderd. Met de komst van Docker, Kubernetes, Serverless architecture en Microservices zijn eisen als snelle opstarttijden en geheugengebruiksoptimalisatie belangrijke eisen geworden. In deze nieuwe wereld is Java achterop geraakt als platform, en dat is goed te begrijpen nu technieken als Spring, JavaEE en Jakarta meer dan 10 jaar oud zijn en gebouwd voor een tijd dat applicaties nog als monolieten werden gebouwd.

Micronaut biedt een product dat past in het nieuwe IT-landschap door reflection technieken die traditioneel in Spring gebruikt worden (IoC) te vervangen door een compile-time generation (AOT). Hierdoor is zowel een snelle opstarttijd als sterke reductie in geheugengebruik te maken die, in combinatie met een uitgeklede JVM door gebruik van bijvoorbeeld GraalVM, een applicatie opleveren die geschikt is voor een microservice-architectuur.

Feature technisch biedt Micronaut ook meer boilerplate beperkende middelen, zoals annotations voor API-client definities om ook aan de datasource kant even makkelijk als aan de webserver kant implementaties te schrijven. Dit soort extra features maken het zeer geschikt voor het snel implementeren van web-to-api of api-to-api microservices.

> *Langzame adoptie Micronaut*
> *Adoptie van Micronaut wordt gehinderd door de overweldigend grote hoeveelheid applicaties die al met een andere techniek (bijv. Spring) zijn gebouwd, inclusief het argument dat er meer developers zijn die al ervaring hebben met een van de bestaande frameworks en daardoor snel inzetbaar zijn op een project. Hoewel er een zekere vorm van waarheid in dit vooroordeel zit, is de learning curve voor Micronaut zeer laag en is binnen een week een simpele api-to-api microservice te maken door iemand die nog nooit eerder met Micronaut heeft gewerkt. Toch leidt dit vooroordeel ertoe dat ook wij nog maar weinig ervaring hebben opgedaan met Micronaut waardoor wij het in trial schalen voor de tech radar.*

Quarkus

Trial

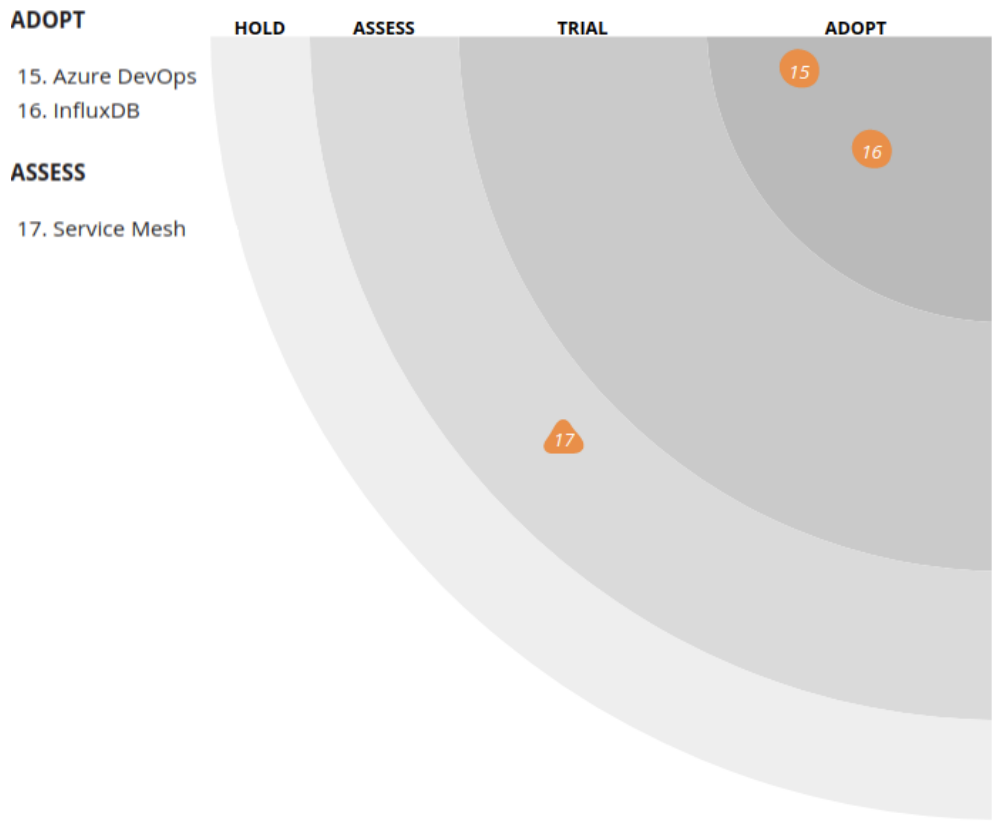
Quarkus is een Kubernetes Native Java framework voor de ontwikkeling van microservices. Quarkus biedt ondersteuning aan de Micro Profile standaarden, native packaging met GraalVM en integratie met een cloud platform of lokaal bijvoorbeeld met Testcontainers. Dit maakt Quarkus vergelijkbaar met Micronaut en een interessant en uitgebreider (full stack) alternatief voor bijvoorbeeld Spring Boot. Quarkus is een relatief jong (2018) product met een best-of-breed filosofie. In 2019 is het product opgenomen in de stack van RedHat en heeft veel potentie.

Java + Vavr

Hold

Vavr, voorheen JavaSlang, is een library om functionele elementen en collecties te gebruiken in Java. Vavr is net als Arrow voor Kotlin een library om FP elementen toe te voegen aan de taal. Vavr biedt hier onder andere enkele datastructuren als een Either en true immutable collections. Hoewel dit mooie verbeteringen zijn voor de taal, geeft het wel problemen zoals name clashes (vavr heeft bijv. een eigen List). Ook is niet altijd alles goed te integreren met standaard Java omdat de taal zich niet erg goed leent voor FP, hierdoor krijg je eigenlijk meer Vavr code dan Java. Hierbij is ons advies: voor een meer FP manier van werken, is het beter om naar de nieuwste Java versie te upgraden of om i.p.v. Java een taal als Kotlin + Arrow te gebruiken.

Platforms



Azure DevOps

Adopt

Azure DevOps biedt op dit moment wellicht de meest complete SaaS-oplossing voor de ondersteuning van software development van product backlog tot deployment. We zien dat grote organisaties dit omarmen, mede vanwege de rijke ondersteuning voor on-premise deployments en het fijnmazige autorisatiemodel. Ook is het mogelijk om eigen dashboards toe te voegen en voor de meeste zaken is er ook een API beschikbaar.

Zowel op het gebied van repositories als op het gebied van CI/CD is het een stuk uitgebreider dan de Atlassian stack of GitLab/GitHub. Het gebruik van deze features leidt uiteraard wel tot een lock-in met het product, zeker indien de pipelines middels de UI worden gebouwd en onderhouden.

Daarom zien wij het vooral goed werken voor grote organisaties die diverse software naar diverse platformen moet kunnen uitleveren. Voor wat kleinere organisaties of volledig container-based delivery pipelines zijn een groot deel van de features echter onnodig.

InfluxDB

Adopt

InfluxDB is momenteel één van de populairste time series databases. Influx Data (leverancier) biedt de OSS open source variant (enkelvoudige standalone instantie) en een betaalde enterprise variant (gedistribueerd, schaalbaar). InfluxDB onderscheidt zich van andere time series databases in de benodigde opslag van data en de snelheid waarmee time series kunnen worden opgevraagd. Het product bevat alle benodigde ondersteuning om het systeem onderhoudsvrij in te regelen. Daarnaast biedt het basisondersteuning t.a.v. authenticatie- en autorisatievraagstukken. Voor integratie met andere systemen biedt InfluxDB standaard een (wat rudimentaire) HTTP(S) API en een HTTPS subscriptie-mogelijkheid. Influx Data biedt aanvullend Telegraf met tal van plugins geschreven door de community en libraries in verschillende programmeertalen voor een directe koppeling met InfluxDB.
Onlangs is versie 2.0 vrijgegeven met volledige ondersteuning van Flux naast de standaard query language. Flux maakt het nog makkelijker om de data uit een InfluxDB te combineren met andere (JDBC) databronnen. In al zijn eenvoud is InfluxDB een prima keuze als time series database. Deze eenvoud maakt wel dat er voor integratie mogelijk extra inrichting nodig is. Dat zou dan ook de overweging zijn in de keuze omtrent een time series database, InfluxDB met bijvoorbeeld één of meer Telegraf's, of een alternatief dat reeds onderdeel is van of beter kan integreren met uw landschap.

Service Meshes

Assess

Met container-orkestratiesystemen (Kubernetes, Openshift, Swarm) werd het maken en beheren van services op basis van containers eenvoudig. Analoog maken Service Meshes (Istio, Linkerd, Anthos, App Mesh) de complexiteit van grotere service landscapes inzichtelijk en daarmee eenvoudiger te overzien. Ze maken de inrichting en het beheer eenvoudiger, maar niet eenvoudig.

De taken van een Service Mesh bestaan uit het managen van alle communicatie-aspecten tussen services. Mooi voorbeeld is het routen van 5% van traffic naar een nieuwere versie van de service (Canary testing). Een volwassen implementatie van een mesh gaan nog veel verder, denk hierbij aan load balancing, policy creation, metrics en, heel belangrijk, service-to-service authentication.

Ondanks de voordelen die zo'n platform kan bieden, vinden wij dat het ook aardig wat extra onderdelen en concepten toevoegt aan de daadwerkelijke deployment. Dit vraagt dus ook een stuk extra kennis van development teams en misschien zelfs een extra platformteam om ook daadwerkelijk de vruchten van een Service Mesh te kunnen plukken.

Een Service Mesh is dus zeker kostbaar; bij keuze voor een Service Mesh worden wel veel zaken ineens geregeld die anders per team verschillend en doorgaans herhaaldelijk handmatig gedaan worden. Vanuit het oogpunt van compliancy is het centraal afdwingen van policies zeer interessant en misschien zelfs een doorslaggevende factor.

Alhoewel Service Meshes meer kennis vragen van teams, geven ze na de inrichting de teams er wel een veel sterkere focus op het realiseren van features voor terug.

Kortom, voor de meeste service landscapes zijn Service Meshes waarschijnlijk nog te zwaar, maar zware compliance-eisen, hoge deployment rate en usability-aspecten zijn drijfveren om deze technologie goed te bekijken.

Service Meshes zijn momenteel in een volwassen vorm te gebruiken op Kubernetes en daarvan afgeleide platforms zoals Openshift en Google Kubernetes Engine. Daarnaast zijn er ook oplossingen die een stap hoger werken, zoals AWS App Mesh of Vamp.io, waarbij het niet uitmaakt waar de service precies op draait. De trend is dat voor container-orkestratiesystemen alle grote partijen hiervoor Istio in ieder geval als basis gebruiken of daarnaartoe bewegen.

Technieken

ADOPT

18. Github flow
19. Infra as code
20. Static Application Security Testing

ASSESS

22. Zero Trust

HOLD

21. Gitflow

GitHub flow

Adopt

GitHub flow is een lichte op branches gebaseerde source control-werkwijze voor teams en projecten die regelmatig deployments doen. GitHub flow is een simpele git workflow die teams kan helpen om de Continuous Delivery-methodiek te implementeren. GitHub flow kenmerkt zich door slechts één enkele `main` branch, meestal de `master` genoemd. De workflow schrijft voor dat vanuit deze branch het team `feature` branches kan aanmaken en zo geïsoleerd aan een nieuwe functionaliteit kan werken. Wanneer deze functionaliteit gereed is, kan de `feature` branch worden gemerged in de `master` branch waardoor er een nieuwe deployable zal ontstaan.

Infrastructure as code

Adopt

Infrastructure as code is een techniek die operations engineers in staat stelt in code te beschrijven hoe infrastructuur (servers, containers, netwerktoegang, access control lists, storage, etc) eruit hoort te zien. Dit stelt een organisatie in staat om deze zaken reproduceerbaar te hebben, en wijzigingen eenvoudiger, betrouwbaarder en traceerbaar door te voeren - de voordelen van CI/CD die gelden voor applicatiecode, gelden ook voor infrastructure as code. Met de brede adoptie van cloud computing en containers achten wij het uitzonderlijk en onwenselijk als deze zaken nog met ad hoc scripts of zelfs handmatig worden uitgevoerd. Denk bijvoorbeeld aan snowflake servers als bekend voorbeeld van dat laatste.
Infrastructure as code stelt organisaties in staat om gecontroleerde wijzigingen van infrastructuur beter te begeleiden en te beleggen bij teams. Dit helpt de traditionele brug te bouwen tussen operations enerzijds - die stabiliteit en voorspelbaarheid in hun SLA's behoeven - en anderzijds development - waar agility en continuous delivery best practices zijn. Dit maakt dat het instrumenteel is geweest in de opkomst van de DevOps mindset, zoals het mantra "you build it, you run it" gevat beschrijft. Wij zien dit als de standaard binnen software delivery teams. Dat betekent overigens niet dat iedere developer hiermee een operations engineer is geworden; slechts dat applicatiecode en infrastructuurcode binnen hetzelfde team onderhouden kunnen worden.

Commit. Develop. Share.

Er zijn hierbij tools die cloud provider-agnostisch zijn, en tools die zich juist op een specifieke cloud of orchestration platform toespitsen. Aanwezige kennis, vendor deals en langetermijn-plannen kunnen helpen bepalen wat de voorkeur geniet; in beide categorieën zijn tegenwoordig sterke alternatieven.

Static Application Security Testing

Adopt

Application security testing kun je dynamisch (op live features in draaiende software) en statisch (op de code en libraries gebruikt om de software te bouwen) uitvoeren. Idealiter doe je beide. In het kader van het mantra 'fail fast is fail cheap' is Static Application Security Testing (SAST) iets dat je vroeg in je delivery pipeline kan zetten. Op deze manier zijn potentiële security risico's bekend en kan worden voorkomen dat ze in je live applicatie terechtkomen, in overeenstemming met het shift left paradigma.

We zien dat er steeds meer keuze uit SAST tooling beschikbaar komt. Inmiddels is er een keuze voor ieder budget en voor iedere set aan requirements, en daarmee zijn wij van mening dat er geen excuus meer is om het vroegtijdig voorkomen van vulnerabilities middels een SAST oplossing niet te doen. Daarmee wordt het ook belangrijker voor teams en organisaties om duidelijk te hebben welke behoeftes een rol spelen in die keuze. Denk daarbij niet alleen aan de kosten van de aanschaf en support van de tool, maar ook aan ondersteuning voor de gebruikte programmeertalen, libraries, containers, artifact stores en CI/CD tools. Wellicht heb je voor sommige zaken al tools en zoek je een aanvulling; misschien geef je de voorkeur aan een all-in-one suite.

Ook de kwaliteit en bruikbaarheid van analyseresultaten die de SAST tool genereert zijn belangrijk; veel false positives en cryptisch beschreven bevindingen kunnen een behoorlijke aanslag vormen op de tijd van developers die de resultaten moeten duiden en verwerken. Er worden regelmatig nieuwe kwetsbaarheden gevonden en bijv. in de Common Vulnerabilities and Exposures database bijgehouden, en het is zaak dat een SAST tool up to date blijft met die bevindingen. Sommige tools kijken uitsluitend naar een subset aan bekende kwetsbaarheden; andere zijn overcompleet en hanteren bijv. een additionele eigen lijst.

Zero Trust

Assess

Zero trust is een methodiek die is voortgebouwd op het gegeven dat vertrouwen stricter te definiëren is als menselijk gevoel. In het Zero Trust model wordt de kern van de belangrijkste (digitale) assets in kaart gebracht en de wegen waarop deze zijn te benaderen. Door deze wegen volgens een zestal regels te voorzien van alleen de noodzakelijke toegang per weg, is een hogere mate van veiligheid te realiseren in vergelijking met bestaande modellen, zoals de perimeter gebaseerde methodieken (trusted domain, DMZ, etc.) en de distrust methodieken (role based access, etc.).

Git-flow

Hold

Git-flow werd in de begindagen van Git geopperd als branching model. Sindsdien is het veelvuldig toegepast en ook tegenwoordig zie je het nog steeds terug in de praktijk.
De wereld is echter behoorlijk veranderd sinds het uitkomen van Git. Tegenwoordig is het veel normaler om Continuous Delivery toe te passen, in plaats van expliciet versioned releases die langere tijd parallel blijven bestaan.
Zoals inmiddels ook in het voorwoord van de introductie van de techniek is verwoord, raden wij aan kritisch te kijken of Git-flow nog wel past bij de dagelijkse praktijk van het project. GitHub flow is bijvoorbeeld een lichtgewicht alternatief en we zien zelfs teams trunk based development adopteren, in combinatie met pair programming.

Tools

ADOPT

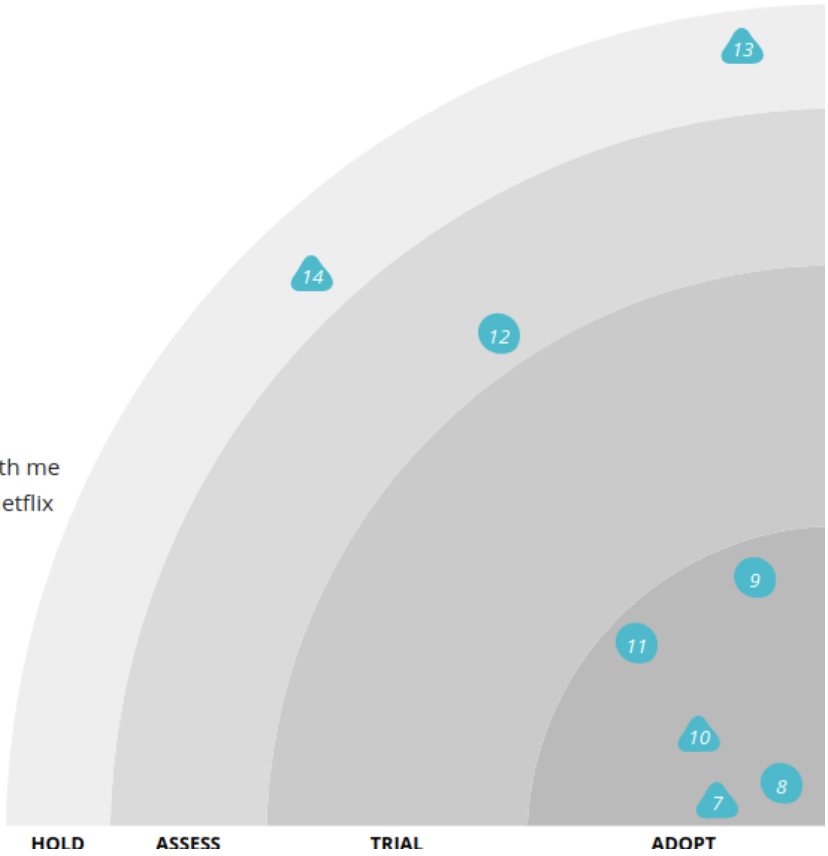
7. ArchUnit
8. AWS CDK
9. Localstack
10. Resilience4J
11. Testcontainers

ASSESS

12. JOOQ

HOLD

13. IntellIJ Code with me
14. Spring Cloud Netflix



ArchUnit

Adopt

ArchUnit is een eenvoudige en uitbreidbare test library om te controleren of Java code voldoet aan architectuur regels, op basis van standaard JUnit tests. ArchUnit gaat na of de afhankelijkheden tussen packages, classes, applicatielagen en dwarsdoorsneden voldoen aan zelf geformuleerde architectuurregels. Met het toepassen van ArchUnit borg je de kwaliteit van de software architectuur binnen een service voor nu en naar de toekomst, zodat een big-ball-of-mud voorkomen kan worden.

AWS CDK

Adopt

AWS CDK stelt je in staat om door middel van onder andere Java (of een andere programmeertaal) AWS resources aan te roepen. Op deze manier wordt infrastructure as code een stuk beter leesbaar en makkelijker te gebruiken voor developers. AWS CDK genereert aan de hand van de geschreven code een CloudFormation script dat gebruikt wordt om de gewenste infrastructuur in AWS op te zetten. Door middel van AWS CDK kan infrastructuurcode gemakkelijk door de developers geschreven en onderhouden worden. Het is transparant wat er gebeurt en de code is bij de applicatie in de betreffende source code repository te bewaren. Zo kan er ook versiebeheer worden toegepast en kan de infrastructuur worden meegenomen in de delivery flow van de betreffende applicatie.
AWS CDK biedt ondersteuning voor Java, JavaScript, TypeScript, Python en C#. Er wordt gewerkt aan de ondersteuning van meer programmeertalen. Op dit moment is het met AWS CDK mogelijk om CloudFormation scripts te genereren, maar er wordt actief gewerkt aan uitbreiding door middel van plugins voor andere platform managers, waaronder Terraform.
Wij zien grote voordelen in het gebruik van AWS CDK door het gemak dat het developers oplevert bij

het intuïtief begrijpen en schrijven van infrastructuurcode. Ook zien we voordelen in het plaatsen van de infrastructuurcode bij de betreffende applicatie en het toevoegen aan versiebeheer. We raden dus zeker aan om AWS CDK te gaan gebruiken voor projecten in AWS-omgevingen.

LocalStack

Adopt

LocalStack stelt je in staat lokaal te ontwikkelen alsof je praat met de AWS Cloud, zonder dat je daarbij verbinding hoeft te hebben met Amazon. LocalStack biedt lokaal dezelfde APIs en functionaliteit aan als de echte instanties, zonder direct dezelfde kosten of benodigde connectiviteit. Het is gemakkelijk te draaien door een Docker container op te starten waarbij je de gewenste AWS services direct kan aanspreken door middel van bijvoorbeeld een SDK of CLI tool.

In de ThoughtWorks Radar staat beschreven dat zij cloud sandboxes prefereren boven lokale simulatie van clouddiensten. Echter zijn cloud sandboxes niet altijd wenselijk omdat dit een afhankelijkheid creëert met het beschikbaar zijn van de clouddiensten, complexiteit t.a.v. beveiliging en inrichting van de infrastructuur, en eventuele kosten die dit allemaal met zich meebrengen. Verder zien wij er met betrekking tot LocalStack juist voordeel in dat het de AWS API's simuleert en je daardoor het DevOps-principe "Build once, Deploy many" kan hanteren.

Alhoewel cloud sandboxes zeker hun voordelen hebben is het niet voor iedereen een optie. Daarentegen zien wij wel juist de voordelen van lokaal je code draaien tegen (gesimuleerde) clouddiensten zoals LocalStack dat doet, en daarbij de feedback cycle van je oplossing zo kort mogelijk te houden (shift left). Wij hebben LocalStack in Adopt gezet vanwege onze positieve ervaringen in onze projecten en we zouden er in de toekomst weer voor kiezen.

Resilience4j

Adopt

Resilience4j is een lichtgewicht fault tolerance library voor Java. Onder fault tolerance horen de volgende enterprise patterns: Circuit Breaker, Rate limiter, Retry en Bulkhead.

Resilience4j is een lichtgewicht fault tolerance library geïnspireerd op Netflix Hystrix, maar ontworpen voor Java 8+ en functioneel programmeren. Het wordt lichtgewicht genoemd, omdat de library alleen Vavr gebruikt en verder geen andere externe library-afhankelijkheden heeft. Netflix Hystrix heeft daarentegen een compilatie-afhankelijkheid van Archaius. Deze bevat veel meer externe afhankelijkheden, zoals Guava en Apache Commons-configuratie.

Er is een duidelijk overzicht van het verschil tussen Resilience4j en Hystrix in te zien op de website van Resilience4j. Resilience4j is simpel, duidelijk en lichtgewicht. Het is voor developmentteams goed toe te passen als alternatief op de grote Netflix Hystrix. Het Spring project Spring Cloud Hystrix is inmiddels ook deprecated en wordt niet meer onderhouden.

Testcontainers

Adopt

Testcontainers is een Java library om tijdens JUnit tests lichtgewicht, wegwerpinstanties op te tuigen van veelgebruikte databases, browser engines, of wat dan ook in een Docker image is beschreven. Met Testcontainers vervalt voor een deel de noodzaak tot een lokale inrichting van een integratieopstelling om vast te stellen of je wijzigingen kunnen werken. Er zijn verschillende modules beschikbaar voor de meest gangbare databases, maar ook messagingplatformen als Kafka, RabbitMQ, Pulsar, of cloudplatformen als Localstack voor AWS of GCloud voor GCP.

Eerder hadden wij Testcontainers nog in Trial laten staan wegens sporadische problemen met

stabiliteit, geheugengebruik of moeilijkheden met het inrichten van een opstelling. De laatste tijd zien wij echter dat deze eerdere problemen zijn weggevallen, waardoor enkel de voordelen overblijven. Tests geven dezelfde resultaten ongeacht op welke machine ze draaien, en het onboarden van nieuwe developers gaat vele malen sneller. Wij raden dan ook aan te toetsen of Testcontainers een toevoeging kan zijn voor een project, om daarmee snellere en stabielere testen op te zetten.

jOOQ

Assess

jOOQ is een lichtgewicht object-relational mapper dat het 'active record' pattern implementeert. Het is een alternatief voor zware ORM frameworks als Hibernate. In tegenstelling tot o.a. JPA heeft jOOQ geen eigen tekstuele DSL (JPQL) maar biedt het een fluent API waarmee queries kunnen worden geschreven. Doordat de API dicht bij SQL blijft geeft het veel controle, voorkomt het type-mapping problemen en kunnen zeer complexe queries worden geschreven. Tevens biedt jOOQ een codegenerator om objecten te genereren vanuit een databaseschema.

IntelliJ Code With Me

Hold

Met het huidige thuiswerkbeleid is er nu meer behoefte aan manieren om op afstand samen te werken zoals remote pair programming. Een tijd terug is de plugin Code With Me geïntegreerd met IntelliJ zelf en is pair programming nog makkelijker gemaakt. Code With Me probeert met een tussenliggende JetBrains server je IDE te delen door anderen een client te laten downloaden, die het dan mogelijk maakt om zelf code aan te passen of gewoon mee te kijken met andermans code.

In eerste instantie is dit dus een heel mooie oplossing, echter zitten er hier risico's aan verbonden. Zo maak je dus standaard gebruik van de JetBrains servers voor alle in- en uitgaande netwerk rondom het pair programming. Het is dan ook goed om stil te staan bij het feit dat je gevoelige bedrijfsinformatie en bijbehorend intellectueel eigendom deelt via een server van een derde partij. Verder is er tegenwoordig de mogelijkheid om Code With Me te gebruiken met eigen on-premise servers, maar dit moet zelf geconfigureerd worden.

Vanwege de huidige situatie en vraag naar wegen om samen te werken willen wij de veiligheidsrisico's benadrukken van dergelijke tools als Code With Me. Om die reden hebben wij Code With Me op Hold gezet, omdat we andere developers aanraden om kritisch te kijken naar dit soort tools om veiligheidsproblemen te voorkomen.

Spring Cloud Netflix (Hystrix, Zuul, Eureka, Feign)

Hold

Al meer dan twee jaar terug werd aangekondigd dat Spring Cloud Netflix in onderhoudsmodus gaat. Dit is al in navolging op de eerdere aankondigingen t.a.v. Hystrix en Ribbon. Het Spring project Spring Cloud Hystrix is inmiddels deprecated en wordt niet meer onderhouden. In deze radar wordt overigens een interessant alternatief behandeld: Resilience4j.

Commit.
Develop.
Share.