
jdriven
Tech Radar
Najaar 2024
10th edition
2024

Voorwoord

Met veel trots presenteren we de 10e editie van de JDriven Tech Radar. Deze mijlpaal markeert 10 edities van technologische vooruitgang, innovatie en een passie voor ons vakgebied van software engineering. Sinds de allereerste editie hebben we talloze technologieën, tools en trends de revue zien passeren, elk met hun eigen impact op de manier waarop we software bouwen en opleveren.

Onze Tech Radar is meer dan een lijst van technische keuzes; het is een weerspiegeling van expertise en toewijding van de JDriven-collega's. Door het evalueren en adopteren van nieuwe technologieën is het onze missie niet alleen onszelf te verbeteren, maar ook een blijvende impact te maken op de software engineering community. We zijn trots dat onze Tech Radar hier een belangrijke rol in speelt.

Met deze jubileumeditie blijven we ons richten op het maken van bewuste, toekomstgerichte keuzes die de kwaliteit en duurzaamheid van softwareontwikkeling verbeteren. We kijken ernaar uit om ook in de komende jaren, samen met de community, te blijven innoveren en onze grenzen te verleggen.

Wat deze jubileumeditie extra bijzonder maakt, is dat deze geïllustreerd is door de creativiteit van al onze collega's, ter ere van 10 edities van JDriven Tech Radars.

Bedankt aan een ieder die dit mogelijk heeft gemaakt, laten we samen verder bouwen aan een actieve en groeiende community voor software engineering!

Erik Pronk
Managing Partner JDriven

Commit. Develop. Share.

© 2024 JDriven

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.

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

Indeling

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

De kwadranten verdelen de verschillende onderwerpen in categorieën.

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

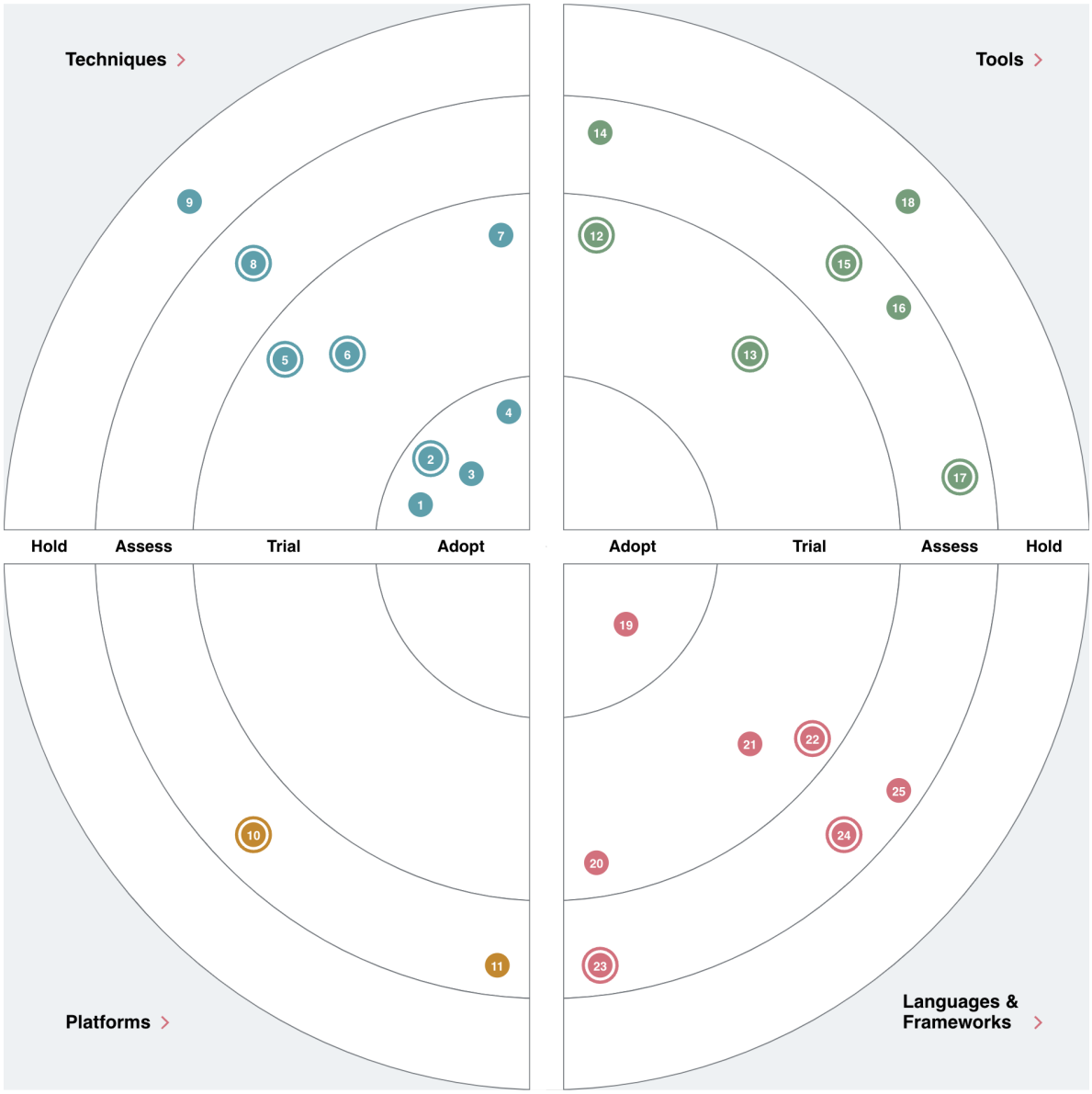
De ringen in elk kwadrant geven aan in welk stadium van adoptie wij denken dat die technologie zich bevindt.

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

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

Commit. Develop. Share.

© 2024 JDriven

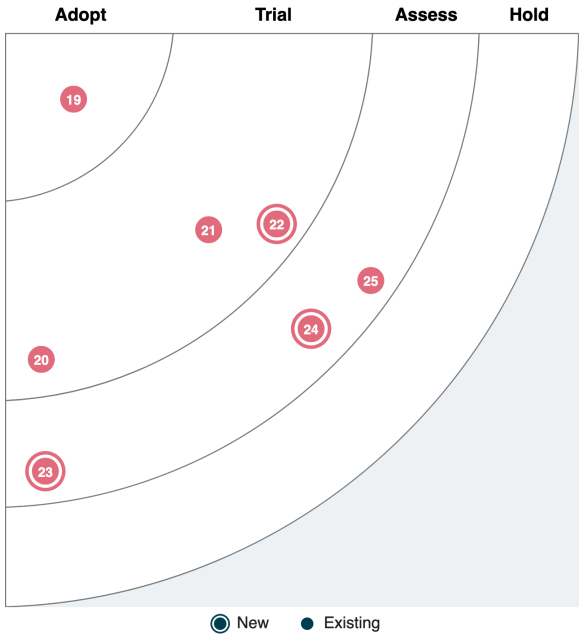


Commit. Develop. Share.

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

Commit. Develop. Share.

Talen & frameworks



Adopt
19. Ktor

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

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

Ktor

ADOPT

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

Het framework bestaat al een geruime tijd, en heeft inmiddels aardig wat tractie binnen de gemeenschap opgebouwd. Daarnaast zien we het gebruik van Ktor bij onze klanten toenemen. Het biedt namelijk een lichtgewicht alternatief voor microservices ten opzichte van bijvoorbeeld een (typisch iets zwaardere) Spring Boot-applicatie, zeker in combinatie met een ORM-library als Exposed.

We hebben er om deze redenen voor gekozen *Ktor* in de *Adopt* ring te plaatsen.

No frameworks

TRIAL

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

- configuratie via properties laden
- Dependency Injection
- ingebouwde REST-server
- ingebouwde JSON marshalling/unmarshalling
- ingebouwde REST-client
- ingebouwde ORM

Commit. Develop. Share.

- ingebouwd transactiemanagement
- ingebouwde log library

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

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

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

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

Commit. Develop. Share.

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 extra benodigde infrastructuur voor onderlinge 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 modulaire 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.

Spring Modulith biedt een aantal annotaties om aan te duiden welke packages onderdeel zijn van een gekozen architectuurpatroon. Naast de annotaties biedt het ook een aantal ArchUnit testen, om vervolgens de gekozen architectuur af te dwingen. Standaard ondersteunde architectuurpatronen zijn: Layer Architecture, Onion Architecture en Hexagonal Architecture en kan vervolgens verschillende soorten documentatiestandaarden genereren. 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 events voor communicatie tussen de modules.

Recent heeft de officiële release van Spring Modulith plaatsgevonden en zien wij Spring Modulith als een goed framework om modulariteit mee aan te brengen in monolithische applicaties.

Timefold AI

TRIAL

Timefold AI is een open-source constraint solver beschikbaar voor Java, Python en Kotlin. Het is ontworpen om complexe planningsproblemen zoals roosteren, routering en resource-allocatie te optimaliseren. Met behulp van constraint satisfaction programming kan een score aan een oplossing worden toegewezen, en kan het probleem worden geoptimaliseerd door een van de ingebouwde zoekalgoritmes te gebruiken. Timefold AI is een fork van OptaPlanner, een project dat niet langer actief wordt ontwikkeld, met verbeteringen die zich richten op prestaties, waaronder een enterprise-versie die multithreading biedt voor verhoogde efficiëntie.

Bij JDriven zien we een toenemende vraag naar constraint solvers bij onze klanten en beschouwen we Timefold AI als een veelbelovend alternatief voor maatwerkoplossingen, vooral bij problemen van beperkte omvang. Daarom plaatsen we Timefold AI in de *Trial* ring en moedigen we teams aan om het uit te proberen.

Helidon

ASSESS

Helidon is een open-source microservices framework, vergelijkbaar met Spring Boot en Quarkus, en biedt een lichtgewicht alternatief zoals Ktor en Vert.x. Het ondersteunt zowel reactieve als imperatieve programmeerstijlen en is beschikbaar in twee versies: Helidon SE (Standalone Engine), een lichtgewicht, reactief framework, en Helidon MP (MicroProfile), dat volledige compatibiliteit biedt met Jakarta EE MicroProfile. In versie 4.0 heeft Helidon belangrijke verbeteringen doorgevoerd, waaronder

Commit. Develop. Share.

native ondersteuning voor virtuele threads, verbeterde gRPC-integratie, en performance optimalisaties die gericht zijn op het verminderen van het resourceverbruik.

Helidon wordt ondersteund door Oracle, wat zowel een kracht als een mogelijke beperking kan zijn. Aan de ene kant zorgt Oracle's ondersteuning voor sterke enterprise-support en langdurige stabiliteit. Aan de andere kant kunnen sommige teams zich zorgen maken over vendor lock-in of afhankelijkheid van een grote corporate partij voor innovatie.

Gezien de ontwikkelingen in versie 4.0 adviseren wij teams, die op zoek zijn naar alternatieve op Java gebaseerde microservice frameworks, om Helidon te overwegen. Daarom plaatsen we Helidon in de *Assess* ring.

Micronaut SourceGen

ASSESS

Micronaut SourceGen is een library waarmee je broncode kunt genereren voor Kotlin en Java. Het is een fork van JavaPoet en is bijgewerkt om nieuwere Java-constructies zoals records te gebruiken. Micronaut SourceGen biedt een gebruiksvriendelijke API om de structuur van de te genereren broncode te definiëren. De library gebruikt een annotatieverwerker om de broncode te genereren tijdens de Java- of Kotlin-compilatie.

Er zijn annotaties om builders en withers voor records te genereren in de library. Dit zou de builder-annotaties van Lombok in een project kunnen vervangen.

Het beste kenmerk van de bibliotheek is dat het integreert met de standaard annotation processor van de compiler. Er worden alleen nieuwe bronbestanden gemaakt en het zal niet knoeien met bestaande classes. De API om de broncode te maken is eenvoudig te gebruiken. De library is in *Assess* geplaatst om uit te proberen en mee te experimenteren.

Vector API

ASSESS

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

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

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

Commit. Develop. Share.

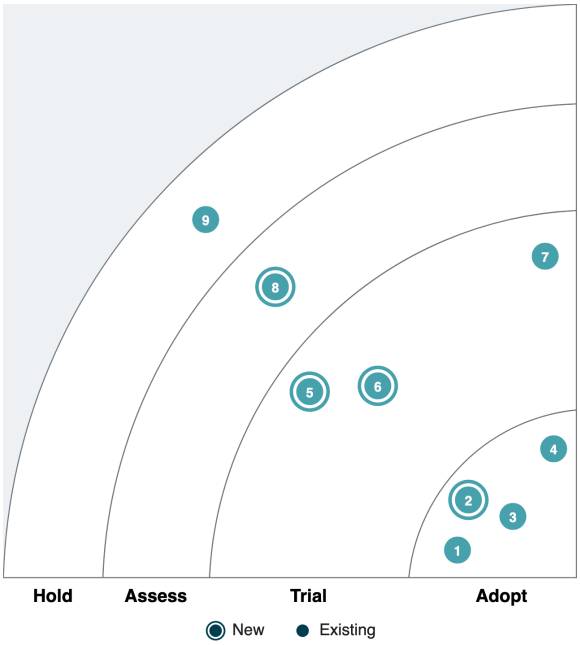
Technieken

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

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

Assess
8. Shape Up

Hold
9. Distributed Monolith



1 TFB NFC

ADOPT

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

1, TFB, NFC neemt een diametraal andere houding aan in vergelijking met technieken als story points, zonder helemaal van schattingen af te zien. Elke taak of story krijgt één van de drie labels: *1 point*, *too frighteningly big (TFB)* of *no faintest clue (NFC)*. Enkel taken die 1 story point hebben gekregen, zijn beknopt en duidelijk genoeg voor een ontwikkelaar om op te pakken. Taken met het label TFB zijn te groot om aan te werken, en dienen in refinement sessies te worden opgesplitst. Taken met het NFC label bevatten nog te veel onduidelijkheden om op te splitsen of in te schatten. Om beter inzicht in deze onduidelijkheden te krijgen, kan het team een spike of een klein prototype bouwen om meer zicht op het probleem te krijgen.

Het gebruik van deze techniek zorgt ervoor, dat de nadruk meer komt te liggen op effectieve planning en refinement, dan op pogingen om schattingen middels grotere getallen exacter te kwantificeren dan mogelijk. Het beheersen van het INVEST principe - waarbij de E bewust niet langer staat voor Estimable - helpt hierbij. Een goede indicatie dat een team zo ver is, is wanneer er weinig variatie meer is in het aantal story points dat toegekend is aan sprint backlog items die het team als klein genoeg beschouwt om mee te nemen in een sprint. Door dergelijke taken een grootte van 1 te geven, is plannen en meten van velocity simpelweg een kwestie van het tellen van backlog items in de sprint. Alhoewel we ons realiseren dat deze eenvoudige classificatie in sommige gevallen te grofmazig zal zijn, beschouwen we het als een voldoende sterke techniek om 1, TFB, NFC naar de *Adopt* ring te verplaatsen.

Commit. Develop. Share.

CVE awareness

ADOPT

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

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

Er zijn grofweg vier soorten vervolgstappen:

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

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

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

GitOps

ADOPT

GitOps is een techniek om je devops automation te versterken door best practices binnen software development zoals versie beheer, samenwerking, CI/CD, en compliance toe te passen.
De techniek bestaat simpel gezegd uit het gebruik van git en infrastructure as code, in samenwerking met een CI/CD pipeline voor alles.
Dat geldt dus ook voor **elke** configuratie aanpassing in een draaiend system.

Met Git wordt elke wijziging bijgehouden in de git history waardoor er een audit trail ontstaat en is er een centraal punt waar men kan samenwerken door middel van merge requests om wijzigingen door te voeren.
Infrastructure as Code zorgt dat alle infrastructuur configuratie in code beschreven staat, die centraal bijgehouden wordt in een git repository
Door een CI/CD pipeline op de git repository aan te sluiten is het mogelijk om de infrastructuur automatisch te laten deployen zonder menselijke interventie.

Commit. Develop. Share.

We zien dat veel bedrijven het al vanzelfsprekend vinden om GitOps te gebruiken. Wij adviseren dan ook om GitOps te omarmen en hebben het daarom in Adopt gezet voor onze technologieradar.

Inner Source

ADOPT

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

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

Binnen JDriven, maar ook bij onze klanten zien we dat deze principes steeds meer omarmd worden en dat het werken volgens de inner source methodiek bijdraagt aan een hogere kwaliteit. Daarnaast zorgt deze bredere blik voor ontwikkelaars binnen de organisatie voor meer inzichten in andermans code waardoor kennisdeling bevorderd wordt.

Om deze redenen plaatsen wij als JDriven inner source in *Adopt* en dragen deze principes uit naar onze collega's en klanten.

15-factor app

TRIAL

In 2011 publiceerde Heroku een set van principes voor het ontwikkelen van SaaS-applicaties (Software as a Service). Deze set aan principes staat bekend als de 12-factor app. Naast een aantal principes voor het ontwikkelen van de services en het managen van de dependencies lag de focus vooral op het cloud-agnostisch ontwikkelen van de applicatie zodat deze eenvoudig in diverse omgevingen kan draaien.

De 15-factor app voegt 3 principes toe die de kwaliteit en onderhoudbaarheid van Cloud Native applicaties kan verbeteren.

- **API-first:** Cloud Native applicaties draaien in een ecosysteem met andere services, managed services en faciliteiten die cloudomgevingen aanbieden. Door de focus op goede APIs te leggen kunnen uitdagingen in integratie worden voorkomen.
- **Telemetry:** In de originele 12-factor app was "logging" al een factor. Logging geeft inzicht in de interne staat van de applicatie. Telemetry is toegevoegd om de aandacht ook te leggen op het resourceverbruik, scaling, integratie met andere applicaties en de integratie met de cloud.
- **Authentication en Authorization:** Cloud Native applicaties kunnen makkelijk gedistribueerd worden over diverse datacenters of zelfs diverse clouds. Het bereik van dergelijke apps is enorm en zowel het aantal als de verschillende typen consumers vergroten de complexiteit op het gebied van security. Authentication en Authorization legt de focus op security bij het ontwikkelen van Cloud Native applicaties.

We hebben de 15-factor app in *Trial* gezet. De 12-factor app geeft al een set aan goede handvatten voor het ontwikkelen van Cloud Native Applications, maar de 15-factor app geeft 3 extra factoren die niet mogen ontbreken in moderne Cloud Native applicaties.

Commit. Develop. Share.

Eventmodeling

TRIAL

Event Modeling is een methode waarbij het beschrijven van gebeurtenissen in een business proces centraal staat. Deze gebeurtenissen leiden weer tot het veranderen van informatie.

Het centraal stellen van een business proces en deze op een eenduidige manier beschrijven, helpt zowel de business als de architecten en ontwikkelaars om een probleem en oplossing te doorgronden. Bij Event Modeling wordt ook de gebruikerservaring meegenomen. Zo ontstaat een compleet beeld van interacties van gebruikers met het systeem en hoe de informatie door het systeem vloeit.

Bij JDriven zien we dat Event Modeling een goede aanvulling op of vervanging van Big Picture Event Storming kan zijn, om vanuit een gedeeld begrip van het businessdomein sneller tot concrete implementaties ervan te komen.

Sustainable software engineering

TRIAL

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

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

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

We zien binnen de branche steeds meer bewustwording op dit gebied, wat heeft geleid tot een verschuiving van Sustainable Software Engineering van het *Assess* naar *Trial* in onze nieuwe Tech Radar. Deze groeiende aandacht onderstreept het belang van groene software en het feit dat het niet langer alleen een theoretische overweging is, maar een praktische noodzaak die steeds breder wordt toegepast.

Shape Up

ASSESS

Shape Up beschrijft een werkwijze waarbij een bepaalde oplossing dusdanig "bijgeschaafd" is, dat het in een iteratie past.

Bij veel projecten staat iteratief werken centraal, waarbij wordt gekeken hoeveel werk in een iteratie past, bijvoorbeeld door middel van het inschatten van Story Points. Shape Up keert dit om, door de oplossing constant bij te schaven, zodat het in een iteratie past. Centraal staat dat de oplossing voor een probleem weinig details bevat, compleet is en duidelijke kaders heeft. Weinig details betekent dat bijvoorbeeld een gebruikerservaring uit een grove schets bestaat. Compleet betekent dat valkuilen geïdentificeerd en weggenomen zijn. Duidelijk kaders betekent dat het duidelijk is, wat het beginpunt

Commit. Develop. Share.

en het eindpunt van de oplossing is.

Blijkt dat de oplossing niet in een iteratie past, dan kan moet de oplossing dusdanig bijgeschaafd worden, zodat het nog steeds de beoogde waarde oplevert én in de iteratie past.

Distributed Monolith

HOLD

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

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

Microservices hebben echter ook een keerzijde:

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

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

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

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

Commit. Develop. Share.

Platforms

Assess
10. AsyncAPI
11. OpenFGA

AsyncAPI

ASSESS

De AsyncAPI-specificatie, vergelijkbaar met de OpenAPI-specificatie voor RESTful-diensten, biedt een gestandaardiseerde manier om event-driven en message-driven API's te beschrijven. Deze specificatie vergemakkelijkt het ontwerp, de documentatie en het gebruik van asynchrone API's, waardoor ontwikkelaars formaten van berichten, kanalen en publish-subscribe patronen kunnen definiëren op een taal-onafhankelijke manier. Naarmate organisaties steeds vaker event-driven architecturen adopteren, ontwikkelt de AsyncAPI-specificatie zich tot een goed hulpmiddel voor het handhaven van consistentie en interoperabiliteit tussen gedistribueerde systemen. De AsyncAPI biedt een standaard en we kunnen die gebruiken om documentatie en code te genereren.

Voor projecten, die event-driven API's gebruiken, bevelen we aan AsyncAPI zeker eens te overwegen.

Commit. Develop. Share.

OpenFGA

ASSESS

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

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

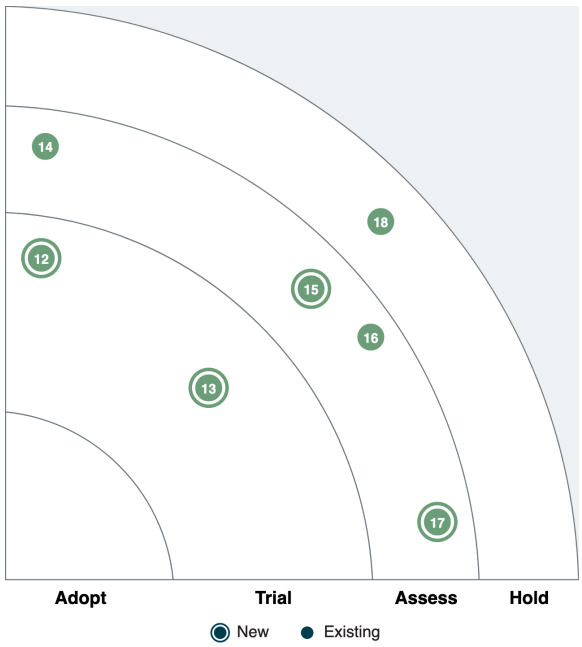
Dankzij de steun van de Cloud Native Foundation heeft het goede ondersteuning en meerdere SDK's voor integratie. Hoewel het actief wordt onderhouden, is er mogelijk minder documentatie of community support beschikbaar omdat het een relatief nieuw project is in vergelijking met sommige gevestigde access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

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

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

Commit. Develop. Share.

Tools



Trial
12. Bruno
13. Devoxx Genie

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

Hold
18. OpenTofu

Bruno

TRIAL

Bruno is een open-source API-client vergelijkbaar met bekende alternatieven zoals Postman, Insomnia en Hopscotch. Volgens Bruno's manifesto moeten API-collecties worden opgeslagen in de broncode repository, zodat ze als een levende set voorbeelden dienen voor het gebruik van de API, waarmee het meteen "levende documentatie" wordt. Collecties worden volwaardige 'burgers', die samen met de bijbehorende informatie worden bewaard en eenvoudig met versiebeheer kunnen worden beheerd.

Bruno's UI is vergelijkbaar met die van Postman, wat een bekende omgeving biedt als je gewend bent aan andere API-clients. Het ondersteunt uiteraard verschillende HTTP-methoden, request body-formaten en response parsing. Bruno heeft plugins voor Visual Studio Code, en ondersteuning voor andere IDE's zijn in ontwikkeling. Belangrijke functies zijn onder meer environment-variabelen, collectie-variabelen en de mogelijkheid om pre-request en post-request scripts uit te voeren.

Vanuit het oogpunt van een ontwikkelaar is Bruno een fris alternatief voor bestaande commerciële producten, zonder dat je meteen verplicht een account nodig hebt en collecties in een cloud omgeving op moet slaan en delen. Door API-collecties naast de broncode te plaatsen, zijn import/export-processen ook niet meer nodig. Hierdoor verbetert automatisch de samenwerking binnen de ontwikkelteams, en de API-documentatie groeit samen met de codebase.

Hoewel Bruno enkele geavanceerde functies mist die in de bekendere API-platforms te vinden zijn, maakt de focus op gebruiksgemak en integratie met ontwikkelworkflows het een interessante keuze. En omdat Bruno open-source is, zijn verbeteringen en aanpassingen vanuit de community mogelijk (laten we hopen dat de IntelliJ plugin beschikbaar is op het moment dat je dit leest). We zouden het zeker adviseren om Bruno uit te proberen, en hebben Bruno daarom in Trial gezet.

Devoxx Genie

TRIAL

Devoxx Genie is een volledig op Java gebaseerd large language model (LLM) Code Assistant-plugin

Commit. Develop. Share.

voor IntelliJ IDEA, ontworpen om te integreren met lokale LLM-providers. Een aantal bekende tools wordt al ondersteund: Ollama, LMStudio, GPT4All, Llama.cpp en Exo. Het kan ook verbinden met cloud-gebaseerde LLM's (OpenAI, Anthropic, Mistral, Groq, Gemini, DeepInfra, DeepSeek en OpenRouter). Enkele van de belangrijkste functies zijn een lokale chathistorie, een tokenkostencalculator, webzoekopdrachten en code-highlighting. De belangrijkste functie is de mogelijkheid om je hele project of pakket toe te voegen aan de context van je prompt wanneer je vragen stelt aan een LLM naar keuze.

Door online LLM's te gebruiken wordt er normaal gesproken data naar servers gestuurd die onder beheer vallen van deze LLM-providers. Bij de projecten waar we als JDriven betrokken bij zijn, wordt nauwelijks gebruik gemaakt van cloud-gebaseerde AI-coding assistants omdat er onvoldoende controle is over wat er met gevoelige data gebeurt. Daarom hebben we Devoxx Genie in de *trial* ring gezet. De mogelijkheid om nog steeds de kracht van coding assistants lokaal, binnen je IDE, te benutten is iets wat veel ontwikkelaars willen. We geloven dat, hoewel Devoxx Genie nog in ontwikkeling is en zijn eigenaardigheden heeft, het een goed alternatief is voor cloud-gebaseerde oplossingen.

Claude 3

ASSESS

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

Wij kiezen ervoor om Claude in de *assess* ring te plaatsen. Op dit moment kan de web-UI niet vanuit de EU gebruikt worden, waardoor het lastig is om er gemakkelijk mee te experimenteren. Ook kunnen de gekozen regels, ondanks dat deze over het algemeen als verstandig worden beschouwd, nog steeds voor een bevooroordeelde (biased) uitkomst zorgen. Dergelijke uitkomsten zullen ook niet altijd stroken met bepaalde culturele patronen. Bovendien is er een dunne scheidslijn tussen het geven van onschadelijke en van ontwijkende antwoorden. In het door Anthropic gepubliceerde onderzoek lijkt dit redelijk onder controle, alhoewel gebruikers wordt aangeraden zelf na te gaan in hoeverre deze technologie toepasbaar is voor hun (specifieke) doeleinden.

CRaC

ASSESS

CRaC, Coordinated Restore at Checkpoint maakt het mogelijk om een snapshot te maken van een draaiende JVM en deze op te slaan op de disk. Vervolgens kun je een JVM laten starten op basis van deze snapshot, wat de opstarttijd flink verkort. Het biedt qua opstarttijd een alternatief voor native compilatie met behoud van de flexibiliteit van de JVM, wat voor complexere applicaties een gulden middenweg kan betekenen. Het Spring Framework heeft hier onder andere flink op ingezet en heeft CRaC volledig geïntegreerd. Er moet echter wel rekening mee gehouden worden dat alle waarden die gezien zijn door de JVM (denk vooral aan secrets) in deze snapshot terecht komen.

Wij vinden het een veelbelovende JVM-toevoeging die een quick win kan leveren voor complexere microservices die veel variatie in load zien, en daarom plaatsen wij CRaC in *Assess*

Commit. Develop. Share.

GitHub Codespaces

ASSESS

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

GitHub Codespaces heeft uitstekende support voor Visual Studio Code via een extensie. Ondersteuning voor de JetBrains IDE's ligt op dit moment nog wat achter. Bovendien moeten gebruikers JetBrains Gateway gebruiken.

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

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

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

warp.dev

ASSESS

Warp is een terminal-applicatie die de ervaring van de command-line interface verbetert voor ontwikkelaars. De applicatie biedt mooie command-line completion, een geïntegreerde command-palette en een block-based outputsysteem voor verbeterde resultatenvisualisatie en interactie. Maar de meest interessante feature is de mogelijkheid om met natuurlijke taal shell-commando's te genereren met behulp van AI. Ook foutuitvoer van een shell-commando kan worden ingevoerd en wordt AI gebruikt om met een oplossing te komen.

Warp biedt een gratis plan met een beperkt aantal AI-aanroepen per maand. Om onbeperkte AI-aanroepen te hebben, kun je upgraden naar het betaalde plan. Er is ook een optie om voor een team te betalen en dan kan het team gebruik maken van extra opties om samen te werken via de shell. Een enterprise versie is ook beschikbaar, waarbij het bijvoorbeeld mogelijk is om een eigen LLM te gebruiken.

De shell is een krachtige tool voor ontwikkelaar, maar het kan moeilijk zijn om alle command-line opties te onthouden. Warp maakt het gebruik van de shell makkelijker door een natuurlijketaal-interface te gebruiken. Als ontwikkelaar hoef je de shell niet verlaten om alle informatie die je nodig hebt te krijgen en dat is zeer krachtig. Aangezien het nog in bèta is, hebben we het toegevoegd aan *Assess* in onze Tech Radar.

© 2024 JDriven

OpenTofu

HOLD

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

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

Eind april 2024 kondigde IBM aan dat HashiCorp tegen het eind van 2024 als onderdeel van IBM verder zal gaan. Alhoewel dit tot hernieuwde belangstelling voor OpenTofu zal kunnen leiden, zien we geen reden om aan te nemen dat deze overname bestaande gebruikers van Terraform in de weg zal zitten.

Commit. Develop. Share.

Commit.
Develop.
Share.