

Onderwijsgroep Professionele Opleidingen
Handelswetenschappen en bedrijfskunde

Technische beslissingen die een organisatie kan nemen om RESTful API's performanter en stabiel te maken

Bachelorproef aangeboden door
Stanislas Melin
tot het behalen van de graad van bachelor
Toegepaste Informatica

Interne begeleider: Patrick Van den Bussche
Externe mentor: Jorn de Pril

Academiejahr 2022-2023

Voorwoord

Tegen het laatste jaar van mijn hogere studies in de Toegepaste Informatica was ik al redelijk veel met RESTful API's in aanraking gekomen. Ik heb tijdens mijn studies API's ontworpen, ontwikkeld en gebruikt en kan met vrije zekerheid zeggen dat ze niet snel weg zullen zijn. Het belang van RESTful API's voor hedendaagse softwareontwikkeling kan helemaal niet worden onderschat. Zij zijn essentieel voor het delen van gegevens tussen systemen en het bouwen van schaalbare en flexibele oplossingen. Het gaat dus zonder zeggen dat het optimaliseren van deze API's op het gebied van prestaties en stabiliteit van enorm belang is voor een organisatie die deze technologie gebruikt.

In deze bachelorproef zal ik een aantal technische beslissingen bespreken die organisaties kunnen nemen om de prestaties en stabiliteit van hun RESTful API's te verbeteren. Ik zal het tijdens deze paper hebben over caching, load balancing, API-gateways, coderingsmethoden, monitoring en andere architectuursbeslissingen die een impact kunnen hebben op de prestaties en stabiliteit van een RESTful API.

Ten eerste wil ik graag Jorn de Pril, mijn stagementor binnen ING, hartelijk bedanken voor de hulp tijdens mijn stage en het schrijven van deze bachelorproef. Ik kon altijd bij hem terecht als ik over eender wat twijfelde.

Vervolgens wil ik ook Enes Kullenovic, senior developer bij ING, bedanken voor alle waardevolle informatie dat hij zonder aarzelen met mij deelde. Ik heb gedurende de laatste drie maanden van hem zo veel geleerd.

Ten slotte wil ik ook mijn begeleiders en alle andere mensen die mij geholpen hebben tijdens het schrijven van deze opdracht. Zonder jullie steun en expertise was dit nooit mogelijk geweest.

Melin Stanislas

24/05/2023

Abstract

Table of Contents

VOORWOORD	2
ABSTRACT	3
LIJST MET GEBRUIKTE AFKORTINGEN	6
INLEIDING	7
DE STAGEOPDRACHT	8
EERST EEN BEETJE CONTEXT.....	8
ACTIEPLAN.....	8
DOELSTELLINGEN	9
<i>Voor ING.....</i>	9
<i>Voor mij.....</i>	9
STRUIKELBLOKKEN.....	10
<i>Grote organisatie</i>	10
<i>Bedrijfscultuur.....</i>	10
<i>Technische hindernissen.....</i>	10
<i>Traagheid</i>	10
<i>Documentatie.....</i>	10
DELIVERABLES	11
<i>Wat moet ik doen.....</i>	11
<i>Hoe wordt het programma gebruikt.....</i>	11
<i>Technologische keuzes</i>	12
<i>Architectuur van het project</i>	13
HOE HEB IK HET GEDAAN	14
<i>Files oppikken met Apache Camel.....</i>	14
<i>Meta-data uit de filename parsen</i>	15
<i>Files opslagen met CC-rest</i>	16
<i>Files opslagen met CCaaS.....</i>	17
<i>Files opslagen met SideCar.....</i>	19
<i>Kafka Idempotent Repository</i>	20
<i>Vulnerability scan met CheckMarx</i>	21
<i>De app deployen.....</i>	22
<i>De app monitoren</i>	22
CONCLUSIE	23
REFLECTIE.....	24
<i>Over de opdracht.....</i>	24
<i>Over de stageplaats</i>	24
<i>Over mijn doelstellingen</i>	25
VERDIEPEND ONDERWERP.....	26
ONDERWERP OMSCHRIJVING.....	26
ACTIEPLAN.....	26
STRUIKELBLOKKEN.....	27
<i>Technische complexiteit</i>	27
<i>Overweldigende hoeveelheid informatie</i>	27
<i>Taalvaardigheid</i>	27
<i>Tijdsdruk.....</i>	27
TECHNISCHE DIEPGANG	27
<i>Wat is een RESTful API</i>	27
<i>Wat bedoel ik met performance en stabiliteit</i>	28
<i>Bottom-up VS top-down.....</i>	28
<i>De REST API</i>	30
<i>Bottom-up approach beslissingen.....</i>	30
<i>Top-down approach beslissingen.....</i>	31

RESULTATEN	32
CONCLUSIE (S).....	32
NAWOORD.....	33
BRONVERMELDING	34
BIJLAGE.....	35
INTERVIEW DEVELOPERS (IK MOET DIE NOG EEN BEETJE OPKUISEN)	35

Lijst met gebruikte afkortingen

- REST – Representational State Transfer
- API – Application Programming Interface
- IT – Informatietechnologie
- DevOps – Development and Operations
- CRUD – Create, Read, Update, Delete
- POA – Power of Attorney
- CC-rest – Common Core – restservices
- CCaaS – Common Core as a Service
- TST – Test omgeving in La Rinascita

Inleiding

Ik zit nu in mijn derde jaar Bachelor in de Toegepaste Informatica te Odisee Hogeschool in Brussel. Tijdens het derde jaar is het verwacht van de student om een stage te lopen in een bedrijf met een IT-afdeling. Voor mijn stage heb ik, gedurende drie maanden, de pijn en vreugde van een team binnen de IT-afdeling van de bank ING gedeeld. Ik kwam in een "squad" terecht dat La Rinascita noemt. Dit internationaal team bestaat vooral uit ontwikkelaars en DevOps Engineers. Samen, werken zij aan een aantal RESTful API's dat door de hele bank gebruikt worden om het geheel van de documenten van de bank te archiveren en op te slaan.

Tijdens mijn stage is het mijn taak om één van de verouderde RESTful API's vanaf het begin opnieuw op te bouwen. Dit, aan de hand van nieuwere versies van libraries als Spring Boot, Apache Camel en Kafka en een nieuwere versie van Java. Het zal de bedoeling zijn voor het team om deze nieuwere manier van een kleine API te bouwen in de toekomst te gebruiken voor andere gelijkaardige projectjes.

Binnen ING is het een bekend feit dat de API's en services niet echt razendsnel zijn. Een RESTful API die niet goed presteert of onstabiel is, kan immers leiden tot slechte gebruikerservaringen en reputatieschade voor het bedrijf. Tijdens mijn stage kregen alle technische medewerkers een uitnodiging voor een conferentie rond het feit dat een zeer hoog aantal van de klanten van ING regelmatig ontevreden waren van de stabiliteit van hun services.

Daarom heb ik besloten om in deze bachelorproef te onderzoeken welke technische beslissingen een organisatie kan nemen om de prestaties en stabiliteit van hun RESTful API's te verbeteren. Er zijn natuurlijk veel verschillende beslissingen die genomen kunnen worden om dit te bereiken. In deze paper ga ik een aantal daarvan analyseren, bestuderen en uitproberen. Het onderzoek zal gebruik maken van een literatuurstudie, praktijkvoorbeelden en interviews met senior ontwikkelaars.

Eerst zal u in dit document een rapportering van de stageopdracht vinden. Daar zal ik eerst wat meer uitleg geven over de manier waarop ik mijn stageopdracht tot een goed einde gebracht heb. Daarbovenop zal ik wat meer ingaan op de doelstellingen en struikelblokken van de stageopdracht. Laatst, zal ik de resultaten bespreken en een woordje zelfreflectie geven over de stage.

Een keer dat u klaar bent met de epische verhalen te lezen over mijn stage aan een bureautje op de tweede verdieping van een *gebouw dat binnenkort verkocht zal zijn aan de Verenigde Staten*, komt het verdiepend onderwerp. In het navolgende stukje zal ik allereerst een inleiding geven op het onderwerp, waarbij ik de context en relevante achtergrondinformatie zal belichten. Daarna zal ik uitgebreid stilstaan bij de uitdagingen en hindernissen die ik heb ondervonden. Ten slotte zal ik inzoomen op de technische diepgang van het onderwerp en mijn bevindingen daarover presenteren.

De stageopdracht

Eerst een beetje context

Ik ben voor mijn stage bij ING in een team terechtgekomen dat "La Rinascita" noemt. Zij zijn verantwoordelijk voor het ontwikkelen en onderhouden van een paar applicaties die te maken hebben met Documentum. Documentum is een Enterprise Content Management platform dat door ING gebruikt wordt om alle documenten dat de bank genereert te archiveren en op te slaan. Met Documentum zelf werken is geen gemakkelijke of aangename zaak, daarom is het de taak van La Rinascita om het gebruik van dat platform te abstraheren in een schaalbare en flexibele REST API dat gemakkelijk geïntegreerd kan worden in andere producten van ING.

De hele bank komt dus in aanraking met producten die La Rinascita ontwikkelt. Het grootste project van ons team is Common Core as a Service (CCaaS), dit is een monolithische RESTful API dat een groot aantal endpoints aan verschillende klanten aanbiedt. Klanten kunnen aan de hand van CCaaS documenten opladen, downloaden, aanpassen en nog veel meer en ze kunnen ook allerlei CRUD-operaties uitvoeren op de meta-data van deze documenten.

Naast CCaaS ontwikkelt ons team ook andere applicaties die verantwoordelijk zijn voor het asynchroon opladen van batches van documenten naar Documentum. Deze applicaties noemt men de xQueues applicaties. Tijdens de stage zal het mijn taak zijn om een van deze verouderde xQueues applicaties, dat poa-xqueues noemt, vanaf nul opnieuw op te bouwen.

Actieplan

Om mijn stage bij ING tot een succesvol einde te brengen, zal ik gebruik maken van de Agile Scrum methodologie. Hierdoor kan ik de focus leggen op het continu leveren van verbeteringen van mijn product en kan ik snel inspelen op veranderingen en feedback.

Het eerste wat ik zal doen, is een product backlog opstellen. Dit zal ik samen met mijn stagebegeleider doen, waarbij we de prioriteiten van de verschillende taken en functionaliteiten van mijn opdracht bepalen. De oude versie van poa-xqueues werkt technisch gezien nog, deze is gewoon oud en gebruikt verouderde en (soms) onveilige versies van libraries. Daarom zal ik niet echt gelimiteerd zijn door de tijd, ik zal kunnen blijven werken aan het product totdat die aan alle eisen voldoet.

Daarna zal ik samen met het team de sprints plannen en verdelen we de taken uit de backlog. Ik zal de enige ontwikkelaar zijn die aan poa-xqueues werkt. Gedurende de sprints zullen we elke dag een korte stand-up meeting houden, waarbij ik het team op de hoogte houd van mijn voortgang en eventuele obstakels die ik ben tegengekomen. Op deze manier kan het team mij bijstaan en kunnen we snel reageren op problemen.

Tijdens de sprints zal ik ook nauw samenwerken met de developers en DevOps Engineers van La Rinascita. Op deze manier kan ik mijn kennis vergroten en kan ik sneller en beter inspelen op eventuele problemen. Ik zal feedback en suggesties van het team omarmen en hierop actie ondernemen.

Tot slot zal ik op regelmatige basis overleggen met mijn stagebegeleider om mijn voortgang en eventuele problemen te bespreken en om input te krijgen voor de verdere ontwikkeling van mijn opdracht. Op deze manier zorgen we ervoor dat ik op de juiste koers blijf en dat ik mijn doelen tijdig behaal.

Doelstellingen

Voor ING

La Rinasita heeft nood aan een nieuwe versie van de poa-xqueues applicatie omdat de huidige versie van het programma zwaar verouderd is. Het is over de tijd ongestructureerd geworden en gebruikt verouderde versies van libraries dat vandaag mogelijke vulnerabilities opbrengen.

Voor deze opdracht zijn er een paar vereisten. De app moet de Spring Boot framework gebruiken voor de structuur van het project, Apache Camel gebruiken voor de integratie van de app in de verschillende omgevingen en Kafka gebruiken om de logs te kunnen bekijken in Kibana. De app moet ook gebruik maken van de beste praktijken op het gebied van Java-ontwikkeling, -release en -testen. Het is ook de bedoeling om de app zo veilig als mogelijk te houden, daarvoor zullen speciale vulnerability analyse tools ter beschikking gesteld worden.

Voor mij

Als stagiair bij ING heb ik enkele persoonlijke doelstellingen voor deze stage. Ten eerste wil ik mijn technische vaardigheden op het gebied van softwareontwikkeling verbeteren. Ik wil leren werken met technologieën en tools die in de sector worden gebruikt.

Ten tweede wil ik graag leren hoe het is om in een groot bedrijf te werken en hoe de verschillende teams en afdelingen binnen ING samenwerken. Ik ben ervan overtuigd dat deze ervaring me zal helpen om mijn communicatie- en samenwerkingsvaardigheden te ontwikkelen, wat essentieel is in het leven.

Persoonlijk wil ik na mijn Bachelor graag verder studeren. Een masteropleiding in de Informatica lijkt mij erg aantrekkelijk. Maar ik onderschat zeker niet hoe waardevol het is om echte ervaring in de sector op te doen tijdens mijn stageperiode.

Tijdens het vak Thesis Preparation heb ik een paar persoonlijke doelstellingen gekozen waaraan ik graag zou willen werken tijdens de stage. Deze zijn:

- 1.4 IT-oplossingen bedenken en modelleren.
- 4.1 IT-oplossingen installeren.
- 4.2 IT-oplossingen configureren.
- 4.3 IT-oplossingen beveiligen en voorstellen doen om beveiligingsrichtlijnen en -procedures aan te passen.
- 11.1 Een opportuniteit omzetten in een project.
- 10.2 De complexiteit van een project in een internationale context kunnen begrijpen.
- 4.5 IT-oplossingen integreren en in productie brengen.
- 7.1 Een oplossing voor een (complex) IT-probleem kunnen ontwerpen.
- 11.1 Een opportuniteit omzetten in een project.

Struikelblokken

Een stage lopen in een groot bedrijf is zeker een interessante ervaring. Ik heb gedurende de laatste drie maanden veel geleerd over samenwerking in een internationale context. Toch, is het niet altijd een plezierige of gemakkelijke zaak. Tijdens de stage heb ik ook een paar problemen en moeilijkheden ontmoet. Hier zijn de struikelblokken die ik heb meegemaakt.

Grote organisatie

Als stagiair in een groot bedrijf zoals ING, kan je je overweldigd voelen door de grootte en complexiteit van de organisatie. Er zijn veel afdelingen en teams waar je mee moet samenwerken en het kan een uitdaging zijn om je weg te vinden en te begrijpen hoe alles samenhangt.

Bedrijfscultuur

Elk bedrijf heeft zijn eigen unieke bedrijfscultuur en het kan enige tijd duren om te wennen aan de manier waarop dingen worden gedaan. Het kan een uitdaging zijn om te begrijpen wat er van je verwacht wordt en hoe je je het best kunt aanpassen aan de bedrijfscultuur.

Technische hindernissen

Werken aan grote projecten met complexe technologieën kan uitdagend zijn, vooral als je nieuw bent in de organisatie. Het kan soms lang duren voordat je alle benodigde tools, systemen en processen onder de knie hebt en je kan tegen technische uitdagingen aanlopen.

Ik heb met allerlei kleine technische probleempjes gesukkeld op een quasi dagelijkse wijze. Soms werkt iets niet omdat een van jouw certificaten niet meer actueel is, soms omdat een van de producten dat door een ander team onderhouden wordt niet meer werkt. Die kleine hindernissen kunnen soms echt frustrerend zijn.

Traagheid

Men zou op eerste zicht kunnen denken dat alles in een groot bedrijf sneller gaat omdat er zo veel medewerkers zijn. Dat is niet het geval. Ik merkte onmiddellijk op dat de kleinste dingen binnen ING een paar dagen of weken kunnen nemen. De communicatie is niet altijd even vlot en het is soms een echte uitdaging om efficiënt aan een softwareoplossing te werken. Ik moest regelmatig wachten op een antwoord van iemand uit een ander team dan La Rinnascita of wachten tot een probleemticket opgelost werd, dit kon soms weken duren.

Documentatie

Ik weet niet zeker of dit een probleem is binnen elk groot bedrijf maar het gebrek aan documentatie van hun producten binnen ING was een van mijn grootste struikelblokken. Ik ben gewend aan duidelijke en complete documentaties van producten. Als ik een nieuwe tool of programmeertaal wil leren ga ik altijd eerst naar de documentatie. Daarom was ik een beetje verloren tijdens de eerste weken van mijn stage.

Als je iets wilt opzoeken over een bepaalde tool of product van ING is er een grote kans dat de documentatie nog niet bestaat of niet compleet is. Nieuwe features worden soms toegevoegd zonder de aanhangende documentatie en dat maakt het soms heel moeilijk om een stukje code, een foutmelding of een product te begrijpen.

Deliverables

Wat moet ik doen

Mijn taak binnen ING tijdens de stage was om een van de xQueues applicaties dat poa-xqueues noemt te herschrijven met een modernere aanpak. "Maar wat is poa-xqueues?" Hoor ik u vragen. De xQueues applicaties zijn een aantal producten binnen ING dat door La Rinascita ontwikkeld en onderhouden worden. Zij zijn verantwoordelijk voor het archiveren van batches documenten die door de bank gegenereerd worden op een platform dat Documentum noemt. POA staat voor "Power of Attorney", dat is een legaal document waarin iemand de bevoegdheid geeft aan een andere persoon om in zijn of haar plaats beslissingen te nemen of handelingen te verrichten. Poa-xqueues is dus een applicatie dat batches van deze poa-documenten naar Documentum stuurt voor archivering.

Hoe wordt het programma gebruikt

De poa-documenten worden door een gebruiker van poa-xqueues geplaatst in een zipfile met soms honderden andere poa-documenten. Deze zipfiles worden dan in een bepaalde directory op een netwerk-toegankelijke drive geplaatst door de gebruiker. Het is dan de bedoeling dat poa-xqueues de zipfiles oppikt, unzippt en de files begint te verwerken en op te slaan.

Een paar belangrijke gegevens die deel maken van de meta-data van een poa-document kunnen rechtstreeks in de filename van een poa-document gevonden worden. Poa documenten zijn altijd PDF's en moeten een bepaalde structuur in de filename respecteren. Voorbeeld van een geldige filename: "ACC_363032228671.01012014.051396001_005.pdf". Vanuit deze getallen kunnen wij meta-data extraheren. Een ander veld van de meta-data is het aantal poa-documenten dat in de zipfile gevonden zijn.

Een keer dat een zipfile of poa-document correct verwerkt is wordt deze verplaatst naar een "Success" folder, als er iets fout gaat wordt de error duidelijk gelogd en wordt de file verplaatst naar een "Failed" folder.

Technologische keuzes

Maven

Voor het bouwen en beheren van onze Java-projecten gebruiken we Maven. Maven is een krachtig build management tool dat helpt bij het bouwen, testen en distribueren van Java-applicaties. Het zorgt voor een gestructureerde manier om afhankelijkheden van externe libraries te beheren en helpt bij het beheren van verschillende configuraties voor ontwikkeling, testen en productie. Met behulp van Maven kunnen we eenvoudig nieuwe libraries aan onze projecten toevoegen en deze vervolgens automatisch laten integreren in ons build-proces.

Spring Boot Framework

Meeste producten binnen La Rinascita zijn Spring Boot applicaties. Het Spring Boot framework is een populair framework voor het ontwikkelen van Java-applicaties. Het is gebaseerd op het Spring framework en vereenvoudigt de configuratie en het opzetten van een Spring applicatie. Het biedt out-of-the-box functies zoals automatische configuratie, standaardmappen en -structuur voor het project, embedded servers, dependency management en meer.

Apache Camel

Apache Camel is een open-source integratie framework dat gebruikt wordt om verschillende systemen met elkaar te integreren. Het biedt een eenvoudige en flexibele manier om data uit verschillende bronnen op te pikken, te verwerken en naar verschillende bestemmingen te sturen. In het geval van de poa-xqueues applicatie wordt Apache Camel gebruikt om de zipfiles en Poa-documenten op te pikken van de netwerk-toegankelijke drive en ze te verwerken en op te slaan in Documentum. Apache Camel biedt verschillende componenten en mogelijkheden voor het integreren van systemen, zoals het werken met verschillende dataformaten, het routeren en filteren van berichten en het omgaan met fouten en uitzonderingen.

Apache Kafka

Apache Kafka wordt gebruikt om de communicatie tussen de verschillende instanties van poa-xqueues te verzorgen en ervoor te zorgen dat bestanden niet dubbel worden verwerkt. Wanneer een instantie van poa-xqueues een zipbestand oppikt om te verwerken, zal het een bericht sturen naar het Kafka-cluster om aan te geven dat het bestand in behandeling is. Andere instanties zullen dit bericht ontvangen en weten dat ze dit bestand niet meer moeten verwerken om duplicaten te voorkomen. Wanneer het bestand succesvol is verwerkt of wanneer er een fout optreedt, zal de instantie van poa-xqueues dit ook weer doorgeven via Kafka.

Logback

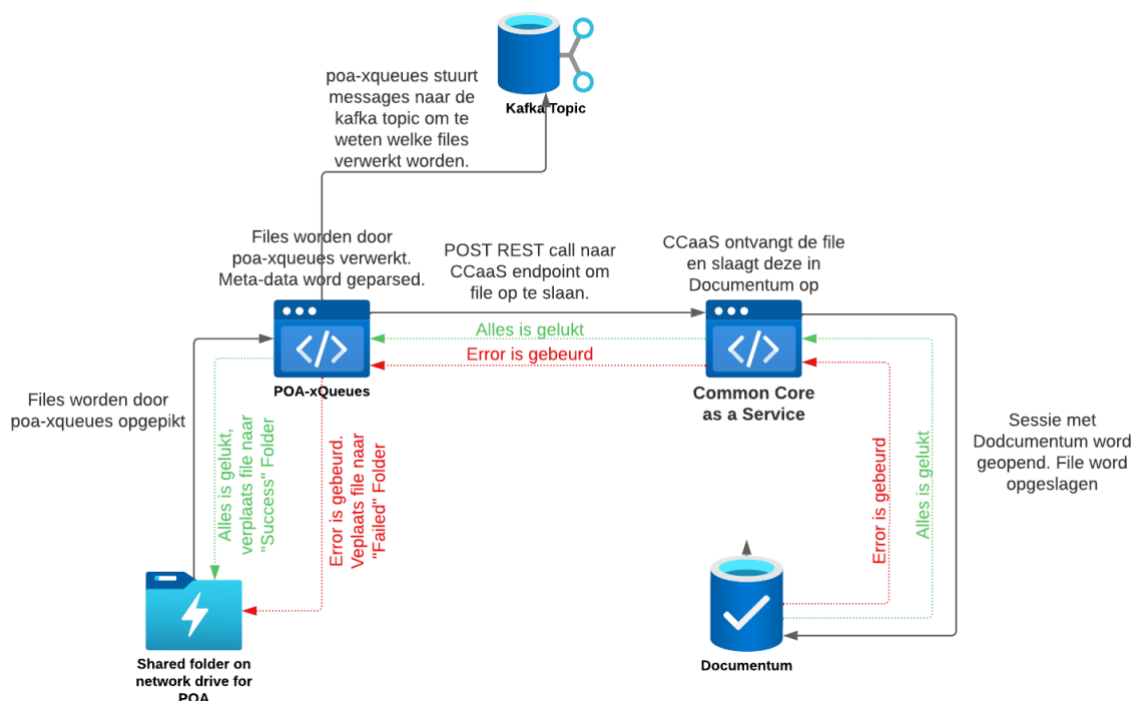
Logback is een logging framework voor Java, vergelijkbaar met bijvoorbeeld Log4j. Een logging framework zorgt ervoor dat er op gestructureerde wijze logberichten worden gegenereerd tijdens de uitvoering van een programma. Zo kunnen problemen en fouten geïdentificeerd en opgelost worden. De logs van de applicatie worden op verschillende manieren bijgehouden.

- Een **rolling-file-appender** is een type appender in Logback dat ervoor zorgt dat de logs automatisch geroteerd worden. Dit betekent dat oudere logs worden gearchiveerd en nieuwe logs worden bijgehouden. Dit is handig om te voorkomen dat de logbestanden te groot worden en om te zorgen dat de informatie nog steeds beschikbaar is voor latere analyse.
- Een **kafka-appender** is een appender in Logback die logberichten naar een Apache Kafka-cluster kan sturen. Dit kan handig zijn als je de logs van meerdere instanties van een applicatie wilt centraliseren en analyseren. Elastic is een platform voor logmanagement en -analyse, dat veel gebruikt wordt in combinatie met Kafka. Het kan logberichten van Kafka verwerken en visualiseren, zodat de informatie gemakkelijk te begrijpen en te analyseren is.

Deze technologische keuzes heb ik niet zelf gemaakt. Ik kreeg bij het begin van mijn stage een soort van voorbeeld-applicatie dat ik als referentie mocht gebruiken voor het ontwikkelen van poa-xqueues. Deze applicatie noemt pc-xqueues en heeft een paar functionaliteiten gemeen met poa-xqueues. Dat betekent niet dat ik alles kon “copy-pasten” en lichtjes aanpassen, de twee applicaties waren verschillend genoeg om dat niet mogelijk te maken maar ik kon pc-xqueues gebruiken als referentie van hoe ik de verschillende tools en frameworks kon gebruiken voor mijn applicatie.

Architectuur van het project

In de onderstaande afbeelding kunt u de architectuur van het project bekijken en hoe de verschillende componenten samenwerken om de files in Documentum op te slaan.



Figuur 1: Illustratie architectuur van poa-xqueues

Om dit te laten werken heb ik verschillende technologieën moeten leren gebruiken. Ik moest eerst begrijpen hoe files opgepikt kunnen worden door de applicatie. Dit doe ik aan de hand van Apache Camel.

Hoe heb ik het gedaan

Files oppikken met Apache Camel

Om nieuwe files op te pikken in een directory maak ik gebruik van Apache Camel. In Camel bestaat er een concept dat Routes noemt. Een route is een pad dat specificeert hoe de Apache Camel 'message routing engine' een bericht van de ene plaats naar de andere stuurt.

Een route is opgebouwd uit een of meerdere 'endpoints' die samen een 'message flow' definiëren. Een endpoint kan bijvoorbeeld een JMS-queue, een REST-service of een file directory zijn. In een Camel route definieer je welke endpoint(s) een bericht ontvangen en wat er met het bericht moet gebeuren (bijvoorbeeld: filteren, transformeren, splitsen, verrijken of combineren). Omdat men voor deze applicatie Camel boven op de Spring Boot framework gebruiken kan men Spring Componenten gebruiken in de Camel Routes.

Het definiëren van een Camel Route om de zipfiles te detecteren en op te pikken gebeurt zo:

```
@Override
public void configure() throws Exception {
    // If an error occurs in our route. It
    // gets picked up here, gets logged with
    // a full stacktrace and duration.
    onException(Exception.class)
        .to( uri: "file://" + poaProperties.getFailedDir())
        .setHeader( name: "elapsedTime", method(countElapsedTime))
        .log(LoggingLevel.ERROR, log, message: "FAILED UNZIPPING FILE: ${header.CamelFileName}")
        .log(LoggingLevel.ERROR, log, message: "ERROR STACKTRACE: ${exception.stacktrace}")
        .log(LoggingLevel.INFO, log, message: "DURATION: ${header.elapsedTime} SECONDS")
        .handled(true)
        .end();

    // Zip files get picked up in this route. They are detected using
    // this regex: "(?i)^.*\\.zip$". The file is then extracted and
    // the underlying pdf files are placed back in the input directory.
    // The zip file is then moved to the success directory.
    from( uri: "file://" + poaProperties.getInputDir() + getQueryParams()) RouteDefinition
        .messageHistory()
        .log(LoggingLevel.INFO, log, message: "INCOMING FILE: ${header.CamelFileName}")
        .routeId(ROUTE_ID)
        .routePolicy(throttlingExceptionRoutePolicy)
        .threads( poolSize: 10) ThreadsDefinition
        .log(LoggingLevel.INFO, log, message: "PROCESSING FILE: ${header.CamelFileName}")
        .bean(zipFileExtractor)
        .setHeader( name: "elapsedTime", method(countElapsedTime))
        .log(LoggingLevel.INFO, log, message: "SUCCESSFULLY UNZIPPED FILE: ${header.CamelFileName}")
        .log(LoggingLevel.INFO, log, message: "DURATION: ${header.elapsedTime}s")
        .to( uri: "file://" + poaProperties.getSuccessDir())
        .end();
}
```

Figuur 2: Code voor definiëren van Camel Route

Een beetje context over dit stukje code is zeker nodig. Een van de belangrijkste objecten in deze class is **poaProperties**. Dit is een class die ik definieerde dat dankzij de Spring Framework automatisch gevuld wordt met configuratie gegevens dat te vinden zijn in de config files van de applicatie. Zo kan ik in een file dat "application.properties" noemt de volgende lijn toevoegen "poa.inputDir=/path/to/input/directory" en deze property overall in mijn project gebruiken.

Een ander belangrijk object in deze code is de **zipFileExtractor**. Dit is een Spring Component dat de file van de Camel Route als parameter meekrijgt. Daarin staat de logica om de zipfile te unzippen en de onderliggende pdf-documenten terug in de input directory van het programma te plaatsen.

Het definiëren van een Spring Component dat door Camel gebruikt kan worden doe je zo:

```
@Slf4j
@Component
@RequiredArgsConstructor
public class ZipFileExtractor {
    // Injecting poaProperties object
    // 2 usages
    private final PoaProperties poaProperties;

    // The method with the @handler annotation is the one that is run
    // when an instance of this class is passed as argument in .bean(...)
    // in a Camel Route.
    // no usages  XT30JL
    @Handler
    public void unzipFile(@Body File file) throws IOException {
        String fileNameWithoutExt = file.getName().replace( target: ".zip", replacement: "");
        Path outputPath = Paths.get(poaProperties.getInputDir(), fileNameWithoutExt);

        log.info("Unzipping file: " + file.getName());
        try (ZipFile zipFile = new ZipFile(file)) {
            zipFile.extractAll(outputPath.toString());
            log.info("Files unzipped to " + outputPath);
        }

        log.info("Moving files to input dir");
        try(Stream<Path> stream = Files.list(outputPath)) {
```

Figuur 3: Spring Component Code

Mijn applicatie telt twee Camel Routes. Beide worden gebruikt om files op te pikken van de Input Directory en ze te verwerken. De eerste route (route-id=zipRoute) pikt zipfiles op en unzipt ze. De tweede route (route-id=docRoute) pikt pdf-documenten op, extraheert de meta-data van hun filename en stuurt ze via een POST call naar CCaaS om ze op Documentum te archiveren. Beide routes zullen hun files in de Success Directory plaatsen als geen probleem opkomt en in de Failed Directory plaatsen als een error ontmoet wordt.

Een keer dat ik Apache Camel onder de knie had moest ik leren hoe ik de meta-data vanuit de filename kon parsen.

Meta-data uit de filename parsen

In de docRoute worden de pdf-documenten opgepikt. Deze worden gedetecteerd aan de hand van een queryparameter in de definitie van de route. Deze parameter gebruikt een reguliere expressie om pdf-files te detecteren. Een keer dat de pdf-file opgepikt is wordt een Camel Header (een soort van variabele dat door de hele route gebruikt kan worden) gevuld met een meta-data object dat gegenereerd wordt door een andere Spring Component dat

PoaDocMetadataParser noemt. Dit object is een Java Record en maakt gebruik van Jackson om die gemakkelijk te kunnen meesturen in de POST call naar CCaaS in een JSON-string.

Om de metadata te parsen uit de filename wordt een redelijk complexe regex gebruikt die we nu samen gaan bekijken.

```
"(?:i)^(?<zipCount>\\d+)_ (?<filePrefix>\\w{3}) . (?<accountNumber>\\d{12}) . (?<fileDate>\\d{8}) . (?<boxNumber>\\d{9}) _ (?<sequence>\\d+)\\.\\.(PDF)$"
```

Figuur 4: Metadata Parser Regex

De verschillende delen van de regex doen het volgende:

- **(?i)** - Dit zegt dat de regex niet hoofdlettergevoelig is.
- **^** - Het begin van een string.
- **d+** - Een of meer opeenvolgende cijfers.
- **_** - Underscore
- **w{3}** - 3 opeenvolgende alfabetische karakters.
- **.** - Punt
- **d{12}** - 12 opeenvolgende cijfers.
- **.** - Punt
- **d{8}** - 8 opeenvolgende cijfers.
- **.** - Punt
- **d{9}** - 9 opeenvolgende cijfers.
- **_** - Underscore
- **d+** - Een of meer opeenvolgende cijfers.
- **.** - Punt
- **(PDF)** - De letters P, D en F in deze volgorde.
- **\$** - Het einde van een string.

Ik geef ook aan elke van deze stukjes een naam in mijn regex statement. Zo kan ik later in de code aan de hand van een Java Matcher Object elk stukje in een aparte variabele steken en valideren. Daarmee bouw ik een PoaDocMetadata object op.

Een keer dat ik de meta-data van het document kon parsen in mijn code was het eindelijk tijd om te leren hoe ik deze files nu naar Documentum ging sturen. Dat is gedurende mijn stage op drie verschillende manieren gebeurt.

Files opslagen met CC-rest

Zoals ik eerder heb vermeld, ontwikkelt het team waarbij ik stageloop een product genaamd CCaaS. Dit is de REST API die door een groot aantal producten van de bank wordt gebruikt om documenten te archiveren, aan te passen en te downloaden. Voor de introductie van CCaaS werd een andere REST API genaamd CC-rest gebruikt. CC-rest kan gezien worden als de voorloper van CCaaS, maar het was niet meer schaalbaar en performant genoeg, waardoor CCaaS werd ontwikkeld als vervanging.

Hoewel CC-rest nog steeds wordt gebruikt voor de meeste xQueues applicaties, is het niet langer de bedoeling om het te blijven gebruiken. Toch werd mij door een Senior Developer geadviseerd om CC-rest te gebruiken voor mijn xQueues applicatie om één simpele reden: het werkte gewoon. CCaaS bleek nog niet geschikt te zijn voor mijn specifieke use-case, zonder dat er een duidelijke verklaring voor was. Dit was echter iets wat ik later tijdens mijn stage zou onderzoeken en oplossen.

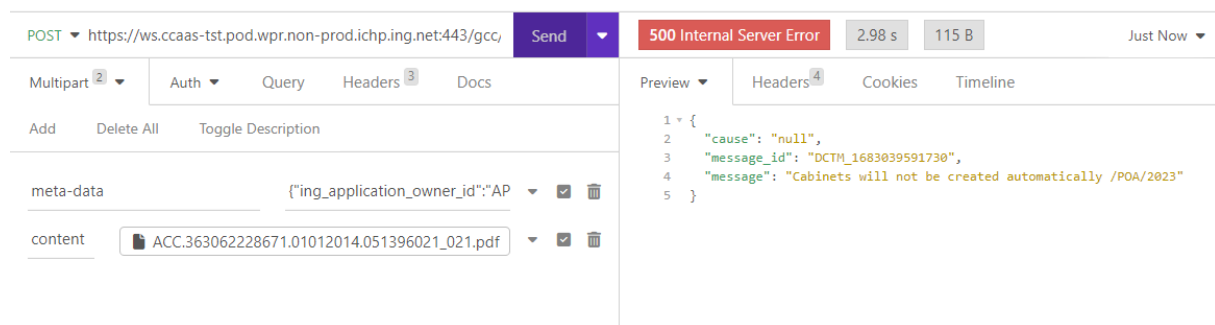
Het bleek vrij simpel te zijn om een request naar CC-rest te sturen om een poa-document op te laden. Ik maakte gebruik van de ingebouwde tool van het Spring Framework, genaamd Spring-Webflux. Dit geeft mij een WebClient-object waarmee ik allerlei requests kan sturen en data kan ontvangen. Dit WebClient-object is zeer aanpasbaar en kan zeer specifieke requests sturen met bijvoorbeeld headers, certificaten en truststores.

Het kostte me een paar keer proberen en falen om uiteindelijk een succesvolle request te sturen en iets nuttigs te doen met de ontvangen data. Ik gebruikte een speciale klasse die de "serializable" interface implementeerde om het resultaat van een request naar een Java Object te parsen. Later tijdens mijn stage zou ik dit vervangen door een Java Record - speciale objecten in Java 17 die sowieso serializable zijn.

Naar mijn mening had ik al goede voortgang geboekt en was ik klaar om me te richten op andere delen van de taak. Toen besprak ik het project met Jorn, mijn stagementor. Hij zei onmiddellijk: "Gebruikt poa-xqueues nog steeds CC-rest? Dat kan niet meer. Alle nieuwe xqueues-applicaties moeten CCaaS gebruiken." Er lag nog wat werk op me te wachten voor de rest van mijn stage. Het was tijd om te gaan debuggen waarom CCaaS niets wilde weten van die poa-documenten.

Files opslagen met CCaaS

Toen ik eerst hoorde dat ik CCaaS in plaats van CC-rest moest gebruiken om poa-documenten in Documentum op te slaan dacht ik dat het maar een kleine aanpassing ging zijn. Ik hoefde, volgens mij en volgens een andere developer van het team, enkel om de url van mijn request te vervangen met die van CCaaS. Dat werkte dus helemaal niet. Elke keer dat ik een poa-document probeerde op te laden kreeg ik een error. Deze error kreeg ik ook in Insomnia (een postman-achtige app dat jou toelaat om REST requests te sturen), hier leek het op:



Figuur 5: Error message van CCaaS

"Cabinets will not be created automatically /POA/2023"... Deze error heb ik een paar honderd keren gezien tijdens mijn stage in La Rinascenta. Ik heb echt van alles geprobeerd maar helaas wou het gewoon niet werken.

Ik heb de volgende weken aan andere tasks gewerkt. Ik implementeerde een feature in CCaaS dat de API toeliet om calls te maken naar haar eigen endpoints. Wat blijkbaar een zeer interessante feature was omdat het in onder twee weken in productie belandde. Dankzij deze feature kon het team nu hun eigen endpoints testen vooraleer veranderingen gemaakt werden in de productie-omgeving. Het gaf hun ook de mogelijkheid om de caches van bepaalde endpoints voor bepaalde types documenten op te warmen bij het opstarten van het programma, wat de API performanter maakt.

Ik heb in deze tijd ook een paar bugs in andere tools onderzocht. Ik nam ook een andere task op waar ik een oplossing moest vinden op een probleem dat het team had in een aantal van hun producten. De apps waren niet echt in staat om statische files van een externe locatie in te laden en te gebruiken. Het probleem was vooral flagrant met certificaten en trustores die elke keer in de JAR van een applicatie gepackaged waren. Het was moeilijk om de certificaten te updaten en vooral vervelend als de app een update kreeg waardoor de nieuwe certificaten niet gebruikt werden in de JAR. Het was zeker niet een dringend probleem maar ik was blij om ergens mee te kunnen helpen.

Ondertussen was ik mijn poa-xqueues probleem niet vergeten. Omdat ik een feature in de code van CCaaS had toegevoegd had ik al een veel beter idee van de structuur van dat programma. Ik had ook, om het programma te debuggen, duizenden lijnen code van CCaaS gelezen en vergeleken met de logs van de applicatie. Ik vond een paar dingen dat interessant leken te onderzoeken maar niks was het waard. Ik was het beu... Waarom wou het gewoon niet werken. Het werkte op CC-rest en CCaaS is op vele vlakken een simpele kopie van CC-rest. Ik had hulp nodig.

Ik vroeg een paar weken na het ontdekking van het probleem een beetje hulp aan de Senior Developer van ons team, Enes Kullenovic. Enes is een van de meest behulpzame en vriendelijke personen dat ik bij ING heb ontmoet. Ik heb altijd leuke babbeltjes met hem en vind hem een fascinerende medewerker. Hij weet alles wat er te weten is over CCaaS. Ik toonde hem het probleem en een paar dagen later stuurde hij een bericht via MS Teams terug. Hij had een mogelijke oplossing gevonden.

CCaaS gebruikt blijkbaar een heel raar systeem om met Documentum sessies open te houden en files door te sturen. Hij gaat eerst een normale sessie open maken met Documentum aan de hand van een Admin Superuser (van Documentum) en gaat een login request sturen voor een andere user, CCaaS opent dan een sessie met de andere user en hoeft geen authenticatie te doen omdat de login request nog actief is. Deze andere user noemt men hier de NPA-user en de bedoeling van deze rare methode is om het aantal passwords dat over het netwerk gestuurd worden te limiteren.

Deze NPA-users zijn dus de ingelogde users in Documentum en het zijn zij die alle veranderingen aan deze DB brengen. Zij maken nieuwe folders aan, lezen de documenten, maken aanpassingen in de meta-data, en zo voort. De NPA-user die voor de repository waar de poa-documenten worden opgeslagen gebruikt wordt heeft maar een gelimiteerd aantal rechten. Dat betekent dat het mogelijk is dat hij geen toestemming heeft om nieuwe poa-documenten op te slagen. Dat zou zeker de rare error die ik gedurende weken op mijn scherm kon zien verklaren.

Enes had gelijk. Ik probeerde deze foutieve npa-user te vervangen in de configuratie files van CCaaS (dat ik hier niet mag tonen omdat het een bank is en ik mag niet zomaar alles tonen), ik verplaatste die met een admin superuser en voilà, het werkte. Ik was echt blij om eindelijk een code zoo op mijn scherm te zien. Enkel Enes kon een oplossing op zo een raar probleem vinden, als Enes dit leest: "You're the best Enes!".

Ik moest nu gewoon de logica van poa-xqueues aanpassen om calls naar een zwaarbeveiligde API als CCaaS te sturen. Dat betekent een paar extra headers en certificaten dat met de request gestuurd moeten worden. Dat kan ik gemakkelijk doen met Webflux. Hier is een beetje code:

```
// This is the WebClient instance we will need to make
// http calls to CCaaS.
2 usages
private WebClient webClient;
// Properties object declared in be.ing.poa.xqueues.properties
// fields are automatically filled from src/main/resources/application.properties
// see: https://docs.spring.io/spring-boot/docs/2.7.8/reference/html/configuration-metadata.html
// see: https://projectlombok.org/
1 usage
private final PoaProperties poaProperties;
// This is the authorization service that generates our
// authorization information based on what is found in application.properties
2 usages
private final AuthorizationService authorizationService;

// This is executed after the constructor is called
no usages 1 XT30JL
@PostConstruct
private void postConstruct() throws DfException {
    // Creating sslContext to add certificates to request
    HttpClient httpClient = HttpClient.create().secure(sslSpec -> sslSpec.sslContext(authorizationService.getTwoWaySslContext()));
    ClientHttpConnector clientHttpConnector = new ReactorClientHttpConnector(httpClient);
    // Auth header is now obsolete
    // authHeader = authorizationService.generateHeaderAuth();
    webClient = WebClient.builder()
        .clientConnector(clientHttpConnector)
        .baseUrl(poaProperties.getDctmRestEndpoint())
        .build();
}
```

Figuur 6: Code Webflux ClientHttpConnector

De **AuthorizationService** is een class dat ik schreef dat alle nodige beveiligingsgegevens kan genereren voor mijn applicatie. In dit geval wordt deze service gebruikt om een SslContext te generen aan de hand van twee files, een certificaat en een key file. Ik hoef het niet eens te instantiëren omdat Spring Boot het voor mij doet bij de opstart van de applicatie dankzij automatische dependency-injection. En ik weet het, ik schrijf veel te veel commentaren als ik programmeer.

Nu dat ik calls kan doen met CCaaS voelt mijn project al veel meer klaar dan vroeger maar toch kwam er nog iets bij. xQueues applicaties maken normaal gezien gebruik van NOG een REST API om calls te maken naar CCaaS. Deze API noemt SideCar en zorgt eigenlijk voor dat hele beveiliging gedoe dat mij zoveel tijd nam om te implementeren. Tijd om opnieuw mijn applicatie aan te passen om requests te maken naar nog een andere service.

Files opslagen met SideCar

Omdat SideCar het meeste doet qua beveiliging om calls te doen met CCaaS was dit de gemakkelijkste manier te implementeren. Ik hoefde enkel de correcte URL in te stellen en een header toe te voegen aan de request. Al de rest dat ik eerder had toegevoegd aan de request mocht weg.

Hier is wat de output van het programma op lijkt als alles goed loopt:

```
n in 18.657 seconds (JVM running for 20.404)
2] b.i.p.x.r.DocRoute - INCOMING FILE: 1_ACC.363062228671.01012014.051396100_494.pdf
2] b.i.p.x.r.DocRoute - PROCESSING FILE: 1_ACC.363062228671.01012014.051396100_494.pdf
2] b.i.p.x.b.PoaDocMetadataParser - Parsing metadata from filename.
2] b.i.p.x.b.PoaDocMetadataParser - Finished parsing metadata.
2] b.i.p.x.b.PoaDocPoster - Creating Poa document.
2] b.i.p.x.b.PoaDocPoster - Poa document created with ID - 0000023gj85
2] b.i.p.x.r.DocRoute - SUCCESSFULLY SAVED FILE: 1_ACC.363062228671.01012014.051396100_494.pdf
2] b.i.p.x.r.DocRoute - DURATION: 2.504s
```

Figuur 7: Output van poa-xqueues

De files worden nu correct opgeslagen maar ik moet ook in gedachte houden dat meerdere instanties van de applicatie gaan draaien. Deze verschillende instanties gebruiken allemaal dezelfde shared directory om de poa-files op te pikken. Ik moet ervoor zorgen dat files niet dubbel worden verwerkt en dat er geen file twee keer wordt opgeslagen.

Kafka Idempotent Repository

Kafka is gebruikt om de bestandsverwerking tussen de verschillende instanties van de applicatie te coördineren. Aangezien er meerdere instanties van de applicatie op verschillende servers draaien, is er een gedeelde map waar de Poa-documenten en zip-bestanden worden geplaatst. Om te voorkomen dat dezelfde bestanden door meerdere instanties worden verwerkt, is een Kafka Idempotent Repository geïmplementeerd met de **InProgressRepository**-feature van Apache Camel.

De Idempotent Repository houdt bij welke bestanden momenteel worden verwerkt en zorgt ervoor dat elk bestand slechts één keer wordt verwerkt, zelfs als er meerdere instanties van de applicatie actief zijn. Wanneer een instantie van de applicatie een bestand ophaalt om te verwerken, wordt er een bericht naar Kafka gestuurd om aan te geven dat het bestand momenteel wordt verwerkt. Andere instanties van de applicatie luisteren naar deze Kafka-topics en slaan het verwerken van het bestand over als ze al zien dat het al wordt verwerkt.

Ik heb de Idempotent Repository niet zelf geïmplementeerd maar heb hem kunnen overnemen van een ander project van La Rinascita dat Kafka-on-Camel gebruikt. Ik hoef hem dan enkel te annoteren met een Camel Specifieke annotatie dat voor de automatische injectie van deze resource in andere Camel Services zorgt (@ManagedResource annotatie). Ik kan nu in de queryparameters van mijn Camel Route de "InProgressRepository" feature gebruiken. Dat doe ik zo:

```
1 usage  XT30/L
private String getQueryParams() {
    // .append("startingDirectoryMustExist=true&")
    StringBuilder queryParams = new StringBuilder("?")
    // .append("startingDirectoryMustExist=true&")
    .append("include=")
    .append(poaProperties.getZipRegex())
    .append("&")
    .append("directoryMustExist=true&")
    .append("autoCreate=false&")
    .append("delete=true&")
    .append("InProgressRepository=#kafkaInProgressRepository&")
    .append("readLock=changed");

    return queryParams.toString();
}
```

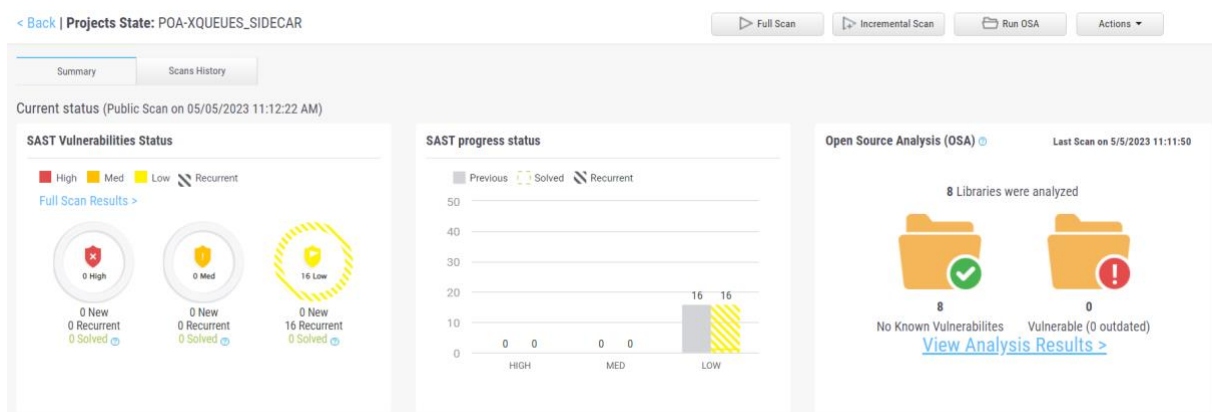
Figuur 8: Kafka InProgressRepository

U kunt zien dat ik niet eens een referentie naar een instantie van de Kafka Repository moet meegeven in de parameter. Dat komt omdat Camel op zijn eigen de resource injecteert dankzij de ManagedResource annotatie dat ik bovenaan de definitie van de Kafka Class heb geplaatst. Ik had nooit gedacht dat Camel zo goed zou werken.

Nu dat we de files op de correcte manier kunnen opslagen en niet dubbel verwerken is het tijd om het over te andere fases van de ontwikkeling te hebben. In een groot bedrijf als ING kan je niet zomaar code laten draaien op eender welke server. Er zijn een aantal veiligheidsmaatregelen dat zeer strikt gerespecteerd moeten zijn. Ik moest eerst een veiligheidsscan doen op het project.

Vulnerability scan met CheckMarx

Het ging niet super vlotjes om mijn CheckMarx account aan te maken maar niets gaat echt vlotjes binnen ING. Ik moest een ticket aanmaken bij een speciaal team dat zich met de accounts van ING-medewerkers bezighoudt. Een keer dat ik mijn account had en toegang had tot het platform moest ik leren hoe ik een CheckMarx scan op mijn project kon uitvoeren. Dat wordt gedaan aan de hand van een Azure Pipeline dat zich in de repository van mijn project moet bevinden. Deze pipeline doet bijna alles voor mij zij compileert het project, downloadt de dependencies, analyseert de code en genereert een summary van de scan op de CheckMarx platform. De pipelines zijn in yaml geschreven en draaien op de Azure cloud. Deze CheckMarx pipeline draait op mijn code elke keer dat ik de code op de Azure Repository push.



Figuur 9: Checkmarx summary pagina voor poa-xqueues

U kunt op de bovenstaande figuur zien dat mijn implementatie van poa-xqueues geen "medium" of "high" vulnerabilities heeft. Een paar "low" vulnerabilities zijn niet erg en gebeuren vooral omdat wij sommige verouderde versies van dependencies in het project gebruiken. Ik heb ze een per een bekeken en de overblijvende vulnerabilities krijg ik niet weg, ik ben verplicht om sommige versies van libraries te gebruiken omdat ik niet de allerlaatste versie van Java gebruik.

Nu dat ik een veiligheidsanalyse heb van mijn project en dat deze geen "high" of "low" vulnerabilities bevat, is het eindelijk tijd om het project op een echte server te laten draaien. De app deployen is geen gemakkelijke zaak, ik moet eerst de code compileren en packagen. Dat doe ik in mijn project aan de hand van de Maven Assembly Plugin. Deze laat mij toe om het compileren en packagen van mijn applicatie te configureren.

Om dat te doen heb ik enkel een file dat "assembly.xml" noemt in de root van mijn project toe te voegen. Hier kan de logica gedefinieerd worden om de verschillende files en directories, dat nodig zijn in het finaal product, in één tar.gz file te pakken.

Ik gebruik een aantal andere Maven Plugins dat mijn werk vergemakkelijken in het project. Deze moeten enkel toegevoegd worden in de "pom.xml" file (staat voor: project object model), dat is de file waar alle informatie over jouw Maven project geplaatst wordt. Dat kan bijvoorbeeld informatie zijn over de dependencies die je wilt toevoegen en welke versie daarvan, op welke manier de JAR gebouwd moet worden, of nog andere meta-data over het project (project- naam, versie, descriptie, enz..).

Ik heb nu een mooie tar.gz file met mijn JAR, alle config files, certificaten, trustores en scripts dat ik op een server kan deployen.

De app deployen

Er zijn voor de verschillende xQueues applicaties twee servers in de Test-omgeving (TST). Om een app te deployen op deze servers moet ik een aantal stappen volgen. Ik legde in de vorige sectie uit hoe men een tar.gz package kon laten generen door Maven. Deze package kan ik aan de hand van de ftp-server van ING uploaden en terug downloaden. Zo kan ik de package vanuit mijn computer uploaden, een ssh-sessie openen met een van de xqueues servers, de package daar downloaden, extraheren en runnen.

Een paar configuratie gegevens moeten aangepast worden een keer dat de app op de xQueues server geplaatst is. Ik moet bijvoorbeeld de input-directory van mijn programma aanpassen of de locatie van sommige certificaten en ssl-gerelateerde files specificeren in de context van deze server. Deze waarden moet ik aanpassen in de application.properties file dat in de config/ directory te vinden is. Spring applicaties zijn echt gemaakt om modulair te zijn met hun configuratie.

Ik heb ook een kleine bash script geschreven om de app op een eenvoudige manier te kunnen opstarten, afsluiten en herstarten. De script wordt dankzij de Maven Assembly Plugin ook mee gepackaged in de tar.gz file. Plaats de script in de root van de app en om die te gebruiken hoef je enkel de script te runnen met een van de volgende drie argumenten: "start", "stop" en "restart". De app wordt in een discrete terminal sessie gestart, dat betekent dat de gebruiker de logs van de applicatie niet te zien krijgt.

De app monitoren

Om de app te monitoren maakt ik gebruik van Logback, een logging framework dat sterk lijkt op Log4J en dat goede integratie met Spring Boot aanbiedt. Om die te configureren maak ik een "logback.xml" file in de resource folder van de applicatie. In deze file kan ik allerlei logging methodes definiëren maar de belangrijkste voor mijn project zijn de "rolling text file appender" en de "Kafka Appender".

Rolling text file appender

Deze appender zorgt ervoor dat er een nieuw logbestand wordt aangemaakt wanneer het huidige bestand een bepaalde grootte heeft bereikt. Op deze manier worden de logs netjes opgeslagen en blijven ze goed georganiseerd.

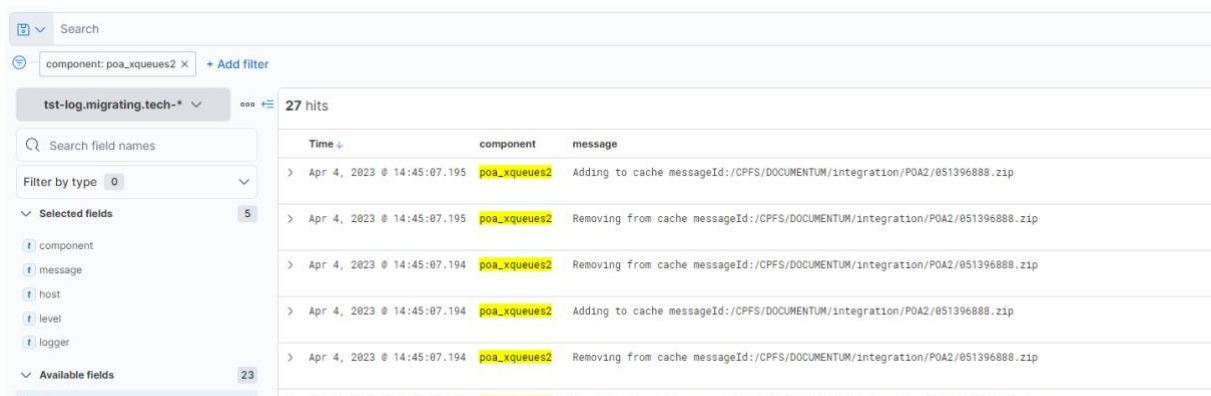
Het gebruik van een rolling text file appender biedt meerdere voordelen ten opzichte van het simpelweg schrijven van logs naar een enkel bestand. Zo is het bijvoorbeeld mogelijk om de logs op een efficiënte manier te bewaren voor langere tijd, zonder dat hierbij onnodig veel opslagruimte in beslag wordt genomen. Daarnaast is het op deze manier eenvoudiger om specifieke logbestanden terug te vinden en te raadplegen, bijvoorbeeld wanneer er een probleem is opgetreden dat moet worden opgelost.

Kortom, door gebruik te maken van een rolling text file appender zorg ik ervoor dat de logs van de poa-xqueues applicatie goed georganiseerd en efficiënt worden opgeslagen.

Kafka appender

Het gebruik van Kafka biedt daarbij enkele voordelen ten opzichte van andere loggingoplossingen. Ten eerste zorgt Kafka voor een betrouwbare en schaalbare manier om de logs te transporteren. Het gebruik van een streaming platform zoals Kafka zorgt ervoor dat de logs op een efficiënte manier verwerkt en gedistribueerd kunnen worden naar verschillende systemen en applicaties. Hierdoor is het mogelijk om grote hoeveelheden logs te verwerken en op te slaan, zonder dat dit ten koste gaat van de performance van de applicatie.

Een ander voordeel van Kafka is dat het de mogelijkheid biedt om de logs in real-time te analyseren en te visualiseren. Door de logs naar een extern platform zoals Elastic of Kibana te sturen, kunnen deze logs geanalyseerd worden op bijvoorbeeld fouten, performanceproblemen of ander incidenten. Op deze manier kan er gereageerd worden op mogelijke problemen in de applicatie en kan de performance van het product gemeten worden.



The screenshot shows the Elastic search interface. At the top, there is a search bar with the text 'Search'. Below it, a filter is applied: 'component: poa_xqueues2'. The interface shows 27 hits. On the left, there is a sidebar with 'Selected fields' (5) and 'Available fields' (23). The main table displays log entries with columns: Time, component, and message. The messages are related to adding and removing cache entries for a specific zip file.

Time	component	message
> Apr 4, 2023 @ 14:45:07.195	poa_xqueues2	Adding to cache messageId:/CPFS/DOCUMENTUM/integration/POA2/051396888.zip
> Apr 4, 2023 @ 14:45:07.195	poa_xqueues2	Removing from cache messageId:/CPFS/DOCUMENTUM/integration/POA2/051396888.zip
> Apr 4, 2023 @ 14:45:07.194	poa_xqueues2	Removing from cache messageId:/CPFS/DOCUMENTUM/integration/POA2/051396888.zip
> Apr 4, 2023 @ 14:45:07.194	poa_xqueues2	Adding to cache messageId:/CPFS/DOCUMENTUM/integration/POA2/051396888.zip
> Apr 4, 2023 @ 14:45:07.194	poa_xqueues2	Removing from cache messageId:/CPFS/DOCUMENTUM/integration/POA2/051396888.zip

Figuur 10: Poa-xqueues logs in Elastic

Conclusie

Poa-xqueues is nu van staat om de zipfiles met poa-documenten op te pikken en uit te pakken en de aantal poa-documenten in een zipfile op te tellen. Hij kan de individuele poa-documenten oppikken en de meta-data ervan uit de filename parsen. Door gebruik te maken van SideCar kan poa-xqueues de files op Documentum archiveren. Meerdere instanties van de app kunnen draaien zonder files dubbel te verwerken door gebruik te maken van een Kafka Idempotent Repository. De app is dankzij CheckMarx gescand voor vulnerabilities en voor verouderde en onbeveiligde versies van haar dependencies. Twee verschillende instanties van de app zijn gedeployd op de xqueues servers en men kan op Elastic dankzij een Logback Kafka-Appender de logs van de applicatie in real-time monitoren.

In deze stageopdracht was het mijn taak om poa-xqueues te herschrijven met een modernere aanpak en dat heb ik gedaan. Met deze nieuwe aanpak is poa-xqueues in staat om op een efficiëntere en betrouwbardere manier batches van poa-documenten te verwerken en te archiveren. Mijn code staat ook vol met commentaar en is goed gedocumenteerd wat waarschijnlijk zeer handig zal zijn voor de xQueues applicaties die in de toekomst ook herschreven zullen moeten zijn.

Reflectie

Over de opdracht

Het ontwikkelen van de poa-xqueues applicatie was een uitdagende taak waarbij ik moest werken met verschillende technologieën en tools. Ik ben in aanraking gekomen met zeer gevarieerde problemen en uitdagingen dat soms niets te maken hadden met programmeren. Dit is wat ik zo leerrijk aan de opdracht vond.

Het belangrijkste onderdeel van mijn stage was de samenwerking met mijn collega's. Hoewel ik in een groot bedrijf als ING werkte, was het soms moeilijk om dingen gedaan te krijgen. Er waren verschillende afdelingen en teams betrokken bij het project en het was soms een uitdaging om te weten waar ik hulp voor mijn zeer specifieke problemen kon vinden. Ik heb geleerd hoe belangrijk communicatie en samenwerking zijn in een dergelijke omgeving en hoe je hier het beste mee om kunt gaan.

Op een technisch vlak vond ik de opdracht wel zeer leuk en leerzaam om uit te werken. Ik had praktisch gezien nul ervaring met Java op het moment dat ik de stage startte, ik had wel een aantal lessen over Java en OOP gehad op school maar ik had nog nooit een groot project van A tot Z in deze taal uitgewerkt. Ik ben blij dat mijn eerste echte ervaring ermee poa-xqueues was. Maven was ook een zeer leuke tool om mee te werken. Het werkte goed en maakte mijn leven gemakkelijker op zo veel verschillende manieren.

Jorn noemt Apache Camel de "Swiss army knife for developers" en ik zou niet meer akkoord met hem kunnen zijn. De framework is super compleet en werkt ongelooflijk goed. Robuuste software als deze is een zeldzaam ding. Alle mogelijke edge-cases waren voorzien en ik moest nooit lang zoeken op de documentatie van de framework om de specifieke informatie dat ik zocht te vinden. Hetzelfde geldt voor Spring Boot, echt een plezier om mee te werken.

Kortom, ik heb veel geleerd over softwareontwikkeling en over de samenwerking in een groot bedrijf als ING. Ik leerde echt leuke mensen kennen en ben tevreden van het resultaat van mijn project. Jorn zei dat hij met plezier jaarlijks studenten van Odisee zou aannemen voor een stage, wat mij laat denken dat hij ook tevreden is met het werk dat ik gedurende de laatste drie maanden heb gepresteerd.

Over de stageplaats

Als stageplaats was ING een unieke en uitdagende omgeving om in te werken. ING is een van de grootste banken ter wereld en heeft een zeer complexe IT-infrastructuur. Het was daarom een enorme kans voor mij om te werken in een dergelijke omgeving en om te leren over de verschillende technologieën die worden gebruikt om zo'n systeem te ondersteunen.

Een van de dingen die mij opviel bij ING was de aandacht voor proces. ING werkt volgens strikte processen en regels om ervoor te zorgen dat alle producten en diensten van de hoogst mogelijke kwaliteit zijn. Daarmee bedoel ik niet dat hun IT-producten de beste zijn en ik denk dat eender wie de ING-app dagelijks gebruikt mij goed versta, toch worden de processen voor softwareontwikkeling en veiligheid strikt gevolgd. Dit kan soms een uitdaging zijn, omdat het werken volgens processen soms vertragend kan werken. Maar uiteindelijk zag ik in hoe belangrijk het is om processen te hebben en deze goed te volgen. Het zorgt ervoor dat er consistente kwaliteit wordt geleverd en dat de bank aan de regelgeving voldoet.

Bovendien werkt ING met een zeer diverse groep mensen, afkomstig uit verschillende culturen en landen. Dit heeft ervoor gezorgd dat ik veel heb geleerd over internationale

samenwerking. Het was fascinerend om te zien hoe iedereen samenkwam en samenwerkte aan projecten, ondanks de verschillende achtergronden en tijdzones.

Maar natuurlijk was het werken bij ING niet altijd gemakkelijk. Het is een zeer grote organisatie en het kan soms moeilijk zijn om door de lagen van bureaucratie en hiërarchie te navigeren om dingen gedaan te krijgen. Ook kan het soms moeilijk zijn om de juiste mensen te vinden om mee samen te werken, omdat er zoveel afdelingen en teams zijn binnen het bedrijf. Maar door te leren hoe de organisatie werkt en door goede relaties op te bouwen, was het mogelijk om deze obstakels te overwinnen.

Over het algemeen was het werken bij ING een zeer leerrijke ervaring en ik ben dankbaar voor de mogelijkheid die mij is geboden om daar stage te lopen. Ik heb veel geleerd over technologie, processen en het werken in een grote en diverse organisatie. Ik heb ook veel leuke mensen ontmoet en echte vriendschappen. Ik ben ervan overtuigd dat de ervaring en kennis die ik heb opgedaan bij ING mij in staat zullen stellen om een succesvolle carrière in de IT-industrie te beleven.

Over mijn doelstellingen

Eerder in dit document vermeldde ik de verschillende doelstellingen die ik gekozen had tijdens het vak Thesis Preparation. Dit waren de verschillende doelstellingen voor mijn stage:

- 1.4 IT-oplossingen bedenken en modelleren.
- 4.1 IT-oplossingen installeren.
- 4.2 IT-oplossingen configureren.
- 4.3 IT-oplossingen beveiligen en voorstellen doen om beveiligingsrichtlijnen en -procedures aan te passen.
- 11.1 Een opportuniteit omzetten in een project.
- 10.2 De complexiteit van een project in een internationale context kunnen begrijpen.
- 4.5 IT-oplossingen integreren en in productie brengen.
- 7.1 Een oplossing voor een (complex) IT-probleem kunnen ontwerpen.
- 11.1 Een opportuniteit omzetten in een project.

Ik kan met plezier zeggen dat ik een beetje van alles geproefd heb. Ik kreeg binnen La Rinascita enorm veel creatieve vrijheid. Ik heb het niet per se over de technologische keuzes van poa-xqueues maar ze lieten mij bijvoorbeeld andere aparte tasks opnemen als ik dacht dat ik een leuke oplossing ervoor kon bedenken. Het was echt leuk om zich altijd gesteund te voelen en te denken dat ik altijd hulp kon vragen aan mijn collega's.

Ik heb nu ook een veel beter idee van het werkveld en van de privésector van de informatica. Mijn droom is nog steeds om verder te studeren maar ik ben echt tevreden dat ik deze stage heb kunnen lopen en dat ik een idee heb van waar ik heen ga. Informatica is mijn passie en deze stage heeft deze passie enkel versterkt.

Verdiepend Onderwerp

Onderwerp omschrijving

In deze paper wordt onderzocht welke technische beslissingen een organisatie kan nemen om RESTful API's performanter en stabiel te maken. Wij gaan eerst samen nagaan wat RESTful API's exact zijn en wat performant en stabiel betekenen in deze context. Daarna, gaan wij bekijken welke beslissingen genomen kunnen worden om de performance en stabiliteit te verbeteren.

Ik heb dit onderwerp gekozen voor een aantal redenen. Ten eerste vind ik het een zeer interessante topic. Programmeren is een echte passie voor mij en ik ben de laatste jaren constant in aanraking gekomen met RESTful API's. Vervolgens heb ik voor dit onderwerp gekozen dankzij mijn stageplaats. Tijdens mijn stage bij ING, waar ik werkte aan de ontwikkeling van een applicatie met een RESTful API, merkte ik op dat er behoefte was aan meer kennis en informatie over hoe deze API's performanter en stabiel gemaakt konden worden. Ik besloot daarom om me te verdiepen in de technische beslissingen die genomen kunnen worden om deze doelen te bereiken.

Over het algemeen is het verbeteren van de prestaties en stabiliteit van RESTful API's een belangrijk aandachtspunt voor organisaties die afhankelijk zijn van deze API's om hun bedrijfsactiviteiten te ondersteunen. Door de juiste technische beslissingen te nemen en best practices te volgen, kunnen organisaties hun RESTful API's optimaliseren en beter inspelen op de behoeften van hun klanten.

Actieplan

Mijn actieplan voor dit onderzoek is als volgt. Ik heb bij ING een aantal zeer ingelichte senior developers leren kennen dat elke dag met REST api's omgaan. Zij weten alles wat er te weten is over deze programma's en ik ga ze daarom eerst interviewen. Vervolgens zal ik wat technische kennis moeten vergaderen door grondig onderzoek/literatuurstudie te doen. Ik zal daarna een REST api zelf schrijven met een paar endpoints en een databank. Op deze test api gaan wij onze bevindingen kunnen uittesten en meten. Alle code voor dit project kunt u terugvinden op GitHub (de link is in de bijlage).

De resultaten van onze bevindingen en tests gaan wij later compileren, vergelijken met de niet geoptimaliseerde api en evalueren op een aantal criteria. Deze criteria zijn:

- **Performance:** Hier gaan wij aan de hand van monitoring en metingtools kijken hoeveel requests per minuut de service aankan.
- **Stabiliteit:** Om dit te meten gaan wij meten welk percentage van de requests een niet 200 response code teruggeven.
- **Schaalbaarheid:** Hiervoor zullen wij meten hoe schaalbaar de oplossing is, daarmee bedoel ik: kan de API gemakkelijk opschalen om meer verkeer te verwerken.
- **Onderhoudbaarheid:** Hier zal ik uit persoonlijke ervaring bespreken hoe gemakkelijk het is om de API te onderhouden en updates uit te voeren.
- **Gebruiksvriendelijkheid:** Om dat te meten zal ik mijn persoonlijke feedback geven over hoe eenvoudig het was om de optimalisatie toe te passen aan een bestaand project.

Struikelblokken

Technische complexiteit

Aangezien dit onderwerp zich richt op technische beslissingen, kunnen er aspecten zijn die moeilijk te begrijpen of uit te leggen zijn. Hierbij is het van belang om de complexe informatie begrijpelijk te maken voor de lezers van de paper.

Overweldigende hoeveelheid informatie

Het onderwerp is zeer breed en er is veel informatie beschikbaar. Het kan moeilijk zijn om de informatie op een gestructureerde manier te presenteren en alleen de relevante informatie te selecteren.

Taalvaardigheid

Het schrijven van een paper vereist goede taal- en schrijfvaardigheden. Nederlands is niet mijn moedertaal en het kan soms lastig zijn om de informatie helder en duidelijk te verwoorden.

Tijdsdruk

Het schrijven van een paper neemt heel veel tijd in beslag en er zijn deadlines die gehaald moeten worden. Het is belangrijk om voldoende tijd in te plannen voor het schrijven van de paper en het uitvoeren van het onderzoek. Ik ben jammer genoeg iets te laat begonnen met het schrijven van deze paper waardoor ik veel minder ga kunnen slapen tijdens de komende maanden.

Technische diepgang

Wat is een RESTful API

Om dit project te starten wil ik graag eerst verduidelijken wat een RESTful API exact is. Ik twijfel er niet aan dat de meeste lezers van deze paper al een goed idee hebben van wat een REST API is maar toch is het een goed idee om iedereens geheugen te verfrissen.

Om RESTful API's goed te snappen is het volgens mij belangrijk om de uitleg tweedelig te splitsen. Het concept bestaat namelijk uit twee woorden, REST en API. Laten we eerst samen ondergaan wat een API werkelijk is.

Wat is een API

Een API (application programming interface) is een collectie van protocollen en definities voor het bouwen en integreren van een softwareoplossing. "Het wordt soms als een contract tussen een informatieaanbieder en een informatiegebruiker beschouwd (Red Hat, 2020)" Dit "contract" legt de regels vast voor welke informatie een gebruiker moet meegeven om de correcte informatie van de aanbieder terug te krijgen.

Denk bijvoorbeeld aan een boekbeheer service, de API-design van deze service zou kunnen specificeren dat een gebruiker een boekcategorie moet meegeven en dat de aanbieder een lijst van boeken dat tot deze categorie behoren terugstuurt.

Een API kan verschillende vormen aannemen, zoals een web-API, een bibliotheek-API, of een besturingssysteem-API. Een web-API maakt het mogelijk om communicatie tussen verschillende systemen over het internet te vergemakkelijken, terwijl een bibliotheek-API het mogelijk maakt om specifieke functies of procedures te gebruiken vanuit een

programmeertaal. Een besturingssysteem-API is een set van interfaces tussen een applicatie en het besturingssysteem. Het kan de applicatie bijvoorbeeld toestaan om toegang te krijgen tot de hardware van een apparaat. Een API kan dus op verschillende niveaus werken en bijdragen aan de interoperabiliteit van software, wat essentieel is voor moderne systemen die vaak uit verschillende componenten bestaan.

Wat is REST

REST staat voor "Representational State Transfer". Het werd oorspronkelijk beschreven door Roy Fielding in een paper die hij schreef, waar hij de principes uitlegde die het World Wide Web zo succesvol maakten. In de context van API's verwijst REST naar een set principes waaraan een API moet voldoen. Als een gebruiker een verzoek stuurt naar een REST API, wordt een representatie van de status van de resource teruggestuurd. Deze informatie wordt geleverd aan de hand van HTTP (Hypertext Transfer Protocol) en kan veel verschillende vormen aannemen. De meest voorkomende vorm is JSON omdat die gemakkelijk leesbaar is door mensen en computers.

REST is een client-server model waarin de server de staat van een resource representeert en de client instructies geeft om deze te veranderen. Het is een stateless communicatiemodel, wat betekent dat elke request van de client naar de server onafhankelijk is van alle voorgaande requests. Hierdoor kan de API eenvoudig geschaald worden zonder dat er rekening gehouden moet worden met sessies of context die overgedragen moeten worden tussen requests.

Wat bedoel ik met performance en stabiliteit

Als men over de performance en stabiliteit van een REST API spreekt, spreekt men over het aantal requests dat de API foutloos kan verwerken in een bepaalde tijdspan. In deze paper onderzoeken wij de mogelijke beslissingen dat een onderneming kan nemen om dat aantal requests te verhogen. Een performante REST API zorgt ervoor dat eindgebruikers snelle reactietijden ervaren en efficiënt kunnen werken met de software die erop gebouwd is.

Bottom-up VS top-down

In deze paper gaan wij twee verschillende soorten beslissingen bespreken. De eerste zijn de beslissingen die een onderneming kan nemen vooraleer de API geschreven wordt. Deze noemt de bottom-up approach en gaat meer om architecturale beslissingen. Een paar voorbeelden hiervan zijn:

- **Microservices versus monolithische architectuur:** Overweeg om over te stappen op een microservices architectuur, waarbij de applicatie in kleinere, onafhankelijke services wordt opgedeeld, in plaats van een grote monolithische architectuur.
- **Synchronous versus Asynchronous:** De juiste aanpak kiezen voor de communicatie tussen de componenten van de API. Synchronous communicatie zorgt voor betere controle en eenvoud, terwijl Asynchronous communicatie betere prestaties en schaalbaarheid biedt.
- **Beveiliging:** Een goede beveiliging van de API door middel van authenticatie en autorisatie, beperking van het dataverkeer, en gebruik van SSL/TLS voor communicatie.
- **Documentatie:** Goede documentatie van de API zodat ontwikkelaars deze gemakkelijk kunnen gebruiken en begrijpen.

De tweede soort zijn de beslissingen dat de onderneming kan implementeren een keer dat de applicatie al bestaat en gebruikt wordt. Dit noemt de top-down approach en zijn de meest frequent gebruikte oplossingen. Men gaat vaker een bestaand product proberen te verbeteren in plaats van een nieuw en beter product vanaf nul opnieuw te schrijven. Een paar voorbeelden van de top-down approach zijn:

- **Caching:** Dit kan op verschillende niveaus gebeuren, zoals op de server of op de client, en kan helpen om de verwerkingstijd van veelgevraagde gegevens te verminderen.
- **Load balancing:** Load balancing verdeelt de verzoeken van gebruikers over meerdere servers, zodat geen enkele server overbelast raakt en de prestaties behouden blijven.
- **API Gateway:** Een API Gateway fungeert als een tussenlaag tussen de API en de client en kan dienen als een controlepunt voor authenticatie, autorisatie en andere veiligheid gerelateerde zaken.
- **Monitoring en logging:** Door gebruik te maken van geautomatiseerde monitoring en logging tools kan een onderneming snel problemen detecteren en oplossen, waardoor de downtime en het verlies van gebruikers worden verminderd.
- **Rate limiting:** Rate limiting stelt een limiet aan het aantal verzoeken dat een gebruiker binnen een bepaalde periode kan doen, waardoor de API beschermd wordt tegen overbelasting en beveiligd wordt tegen DDoS-aanvallen.

Door een combinatie van bottom-up en top-down beslissingen te nemen kan een onderneming ervoor zorgen dat hun RESTful API's performanter en stabiel worden en tegemoetkomen aan de behoeften van hun gebruikers.

*** Ik ben tot hier geraakt... Vanaf volgende week ga ik veel meer tijd kunnen spenderen aan het schrijven van deze paper. ***

De REST API

Hier toon ik de niet geoptimaliseerde api die ik geschreven heb...

Bottom-up approach beslissingen

Hier leg ik de verschillende bottom-up approaches uit en hoe deze er uit zouden kunnen zien voor ons test projectje...

Microservices vs Monolithisch

...

Synchronous vs Asynchronous

...

Beveiliging

...

Design patterns & Best practices

...

Documentatie

Top-down approach beslissingen

Hier leg ik de verschillende top-down approaches uit en hoe deze er uit zien voor ons test projectje... De resultaten zullen ook gemeten worden voor elke methode

Caching

...

Encoding

...

Load-Balancing

...

API-Gateway

...

Monitoring & Logging

...

Resultaten

Hier ga ik de resultaten van de vorige metingen vergelijken met elkaar...

Conclusie (s)

...

Nawoord

Bronvermelding

What is a REST API? (2020, 8 mei). Red Hat. Geraadpleegd op 7 mei 2023,
van <https://www.redhat.com/en/topics/api/what-is-a-rest-api>

Bijlage

Interview Developers (ik moet die nog een beetje opkuisen)

Enes Kullenovic

What do you think of the state of our current API (performance & stability)?

I think there is a big gap between the monitoring tools and the implementation of performance tracking. The best practices for performance tracking are not met. The tools are there but are not used correctly. The system is very advanced but lacks in other places. This affects the quality of the product. We have many customers and they each have expectations of the performance of the product. The scale of this projects makes it very hard to track performance issues as it has many different aspects and as many customers. The step by step approach is sometimes a bottleneck to the tracking of performance. In Agile everything is constantly in motion which makes it hard to implement big changes. It's harder to track quality. A better understanding of this is needed of the product if one wants to track this. In our product we miss an integration environment: a place that is similar to PRD where one could make a clean stress test.

Are there any technical decisions that could have been made to make the product more performant or stable?

Having an integration environment would make it much easier to track real performance bottlenecks. When testing in the conventional method on smaller servers we have to deal with so much noise on the service that accurately measuring things become very hard.

An integration environment should be as close to production as possible. One introduces a separate system that is very similar to PRD which implements many of the integrations with other systems that will happen once in PRD. You always need to keep it in sync with PRD. This is the simplest way to monitor a big project.

Which technical decisions have been made inside ING that affected (negatively) the performance of our products?

At some point you need to make some decisions. We are prioritizing some customers that have the biggest issues with performance. Their impact is estimated as very important. We are trying to reach a top down approach. Once we have a reliable set of tools that can define the quality requirements of our systems we will have made a real leap forward.

Top to bottom tools include elastic search, tracing. In big projects like that

Some companies use Caching, Encoding, Monitoring, load balancing, API gateways. Which of these are implemented in our product?

A lot. This is a typical bottom up approach. Sometimes adding too much of these tools will also affect performance by bloating the dependency tree of your project. Unless you add new hardware to your machines trying to solve everything with more tools and code can sometimes make it slower. You need best practices in place in order to use the tools efficiently, this is where we seem to have a gap.

Caching is used in one of our slowest dependencies (in-memory caching). Its lack of documentation and implementation of best practices makes it very hard to use these efficiently in our product. One can always throw more hardware at the system but it just hides

the real issues under a blanket of hardware performance. The overall performance approach is not about that, it's about increasing the quality of your service at every level.

In our product we are using Spring Boot. In this framework everything is centered around beans. These beans are not used in a consistent way and each have heavy dependencies. These are adding to a lot of the mess that our project may already face. Our lack of implementation of best practices makes it very hard to efficiently fix real problems in the system. Since some of our problems come from XML beans, some from Java Classes, some from other types of implementations. It is all way too distributed and unorganized.

Microservice approach would make more sense but our unorganized way of coding makes it hard to switch over to any other architecture.

Tools to measure performance of an API: Java Inflight Recorder, Elastic Search, Kibana, APM,..

Measuring performance is not always super.

Enes recommended I take a look at MDN (Multimodal distribution).

Martin Navez

What do you think of the state of our current API (performance & stability)?

Some strong improvements can be brought to our product on performance. Our API uses many different external services that are often the bottleneck for the performance of our product. I personally think that it is kind of okay right now but I do believe that things could really benefit from a complete rewrite of our product. It is now a pretty old API and it has never been rewritten in its entirety.

If you could rewrite the product, what would you do differently?

The problem with our app right now I think is the messiness of it. Nothing is really clear and some things are really bloated and messy. This makes it very hard to understand certain pieces of code and makes it even harder to optimize it. I spend more time reading code to try to understand it than actually adding/modifying code. The real obvious solution would be to rewrite with clean code principles and modern conventions. The implementation of best practices would allow us to reflect on more impactful decisions that can be taken to improve stability.

Are there any technical decisions that could have been made to make the product more performant or stable?

One of the things we really regret not doing is real monitoring. If we could monitor our product from the inside for a duration of a whole year we could actually have a lot more data and insight on the different endpoints in our application. Our product was also partially rewritten at some point in the past but not entirely and it really suffers from the lack of consistency. I think the real problem is the lack of insight in the usage of our product.

I think that one of the other problems of our app right now is that we have one monolithic structure in the application. It is one big project with many many endpoints and to me it looks like it would be much much easier and performant if our endpoints were a little more distributed. Microservices always come to mind. They are easier to debug and maintain, are more performant and don't block the whole service whenever a problem comes up. Sadly, the app already exists and is used by many many different clients. This makes it very hard to get change started.

Which technical decisions have been made inside ING that affected (negatively) the performance of our products?

We generally never put much thought into the performance impact of our dependencies. It never mattered whether some dependency was the perfect choice as long as it worked. Our project uses Spring Boot for its RESTful side. Spring is pretty old and some of our dependencies are now decades old.

We also made the weird choice to use an external service called Documentum for the storage of our files. A majority of the problems we encounter in our product are Documentum related. It didn't really make sense to call on a buggy outside company for this trivial task. Many better solutions exist out there nowadays. Still, Documentum uses a lot of the same tools as us and comes with a neat and complete package which ING seems to really like.

Some companies use Caching, Encoding, Monitoring, load balancing, API gateways. Which of these are implemented in our product?

In our product we have recently re-implemented load balancing. I think we have tried implementing caching but I'm not sure we went all the way through with it.

It is planned that on the next sprint we will add more monitoring capabilities and polish the current monitoring in our app. That means cleaning up some logs. We also planned to implement APM which is a tool to measure performance for an API. Still, our product uses monitoring at multiple levels right now. The tools we use for this are Elastic Search, Kibana.

Darling Cathyrene Villas

What do you think of the state of our current API (performance & stability)?

I think that our product is not functioning as perfectly as we would want. We have enormous amounts of endpoints and that's a problem to maintain. If we had smaller services with smaller amounts of endpoints, our performance problems would be easier to solve.

We also make use of documentum which is one of the biggest bottlenecks for our service. If you look at some of our queries you notice that we always request for every single field in the db which is inefficient. Also caching some of these queries would help a lot.

Are there any technical decisions that could have been made to make the product more performant or stable?

Yes! The first thing I believe would be to limit what we request. We cater too many endpoints in our product. Our clients can do too much with too many endpoints which makes it messy, hard to maintain and slow.

Which technical decisions have been made inside ING that affected (negatively) the performance of our products?

I don't think we have made a lot of decisions for the sake of performance in our product. This is also why our product is a little messy. We never had performance in mind, we only added more and more things without worrying.

Some companies use Caching, Encoding, Monitoring, load balancing, API gateways. Which of these are implemented in our product?

We have not added caching to our product, which is a shame. It would drastically improve the performance of our product.

We have implemented load balancing but right now we have so much load that it barely improves anything. Still it's an important part of our API.

Some of our clients connect via an API Gateway (NginX) and that seems to be a good thing.

If you had to rewrite our API, what would you do differently?

Our monitoring is a little unorganized. It is sometimes hard to understand the logs that are picked up. Also the error messages are often not very useful and don't make much sense which makes it hard to think of possible solutions. In an ideal world our product would use a global class to handle all the errors with useful messages.

I would also add caching from the start. We know this would make it much faster but the way it's unorganized right now makes it hard to think of possible fixes to performance issues.

Sticking to design patterns would also make everything more clear and transparent for anyone to update the system on performance and stability.