Adobe Experience Manager

Mike Heymans

2017

Inhoudstafel

1 Voorwoord

In het kader van mijn opleiding Toegepaste Informatica aan de Hogeschool Gent heb ik onderzoek gedaan naar het Adobe Experience Manager platform, beter gekend als AEM. Het resultaat dat nu voor u ligt is er n van drie maanden hard werk waarin heel wat gevloek heeft plaatsgevonden. In nazicht van dit onderzoek ben ik tevreden over het resultaat maar vooral voldaan met de verworven kennis. Al heeft deze kennis gezorgd voor nieuwe vraagtekens die in de toekomst mijn aandacht zullen opslorpen.

Dit project is uitgevoerd in opdracht van AS Adventure, met mijn projectleider Steven Rymenans als begeleider.

Deze proef was niet mogelijk geweest zonder de inzet van AS Adventure, die zo vriendelijk waren om een licentie beschikbaar te stellen alsook mij te begeleiden in mijn ontdekkingsreis binnen de wereld van AEM. Ik wil ook mijn collegas bedanken voor hun uitmuntende begeleiding en eindeloos geduld.

Ik bedank ook mijn begeleiders voor hun samenwerking en ondersteuning tijdens dit traject.

Veel plezier met het lezen van deze scriptie.

2 Reden van onderzoek

Adobe Experience Manager is het gelicentieerde contentmanagementsysteem (CMS) van Adobe. Een CMS-applicatie maakt het mogelijk voor mensen zonder kennis van html of css om webpaginas aan te maken of te wijzigen. Het is ideaal voor websites die vaak de inhoud van hun site wijzigen of snel willen inspelen op actuele gebeurtenissen. Dit komt doordat designers zelf de paginas kunnen editeren in plaats van eerst content uit te denken en de realisatie over te laten aan een front-end developer.

Toen AS Adventure besloot om zijn huidige website, en die van zijn zusterbedrijven, niet langer uit te besteden aan een extern bedrijf maar deze intern te gaan beheren en ontwikkelen, is er besloten om deze via het AEM-platform op te zetten. Dit zou het bedrijf in staat stellen om zijn content volledig door het content team te laten beheren, zonder tussenkomst van webdesigners. Denk maar aan de homepage die van achtergrondafbeelding wijzigt of een gepersonaliseerde pagina voor een merk, allemaal mogelijk dankzij AEM.

Dat dit allemaal out-of-the-box mogelijk is, is te mooi om waar te zijn. Voor het content team aan de slag kan moeten er componenten gebouwd, deze worden gebouwd door... ontwikkelaars. Eigenlijk komt het erop neer dat componenten de bakstenen vormen om een website op te bouwen en het content team zijn de metsers. Als er nood is aan een baksteen met een nieuwe feature moet hiervoor een aanvraag gedaan worden bij de steenbakkers, gespeeld door de ontwikkelaars. Deze stenen bevatten niet enkel HTML en CSS maar ook Java (en diverse frameworks) zijn nodig om dit nieuw type steen te ontwikkelen.

Buiten de content die door het marketingteam wordt voorzien is er ook data die vanuit een databank moet komen. Hierbij hebben we het over records die niet manueel worden aangepast maar met duizenden tegelijk of data die niet enkel betrekking heeft op de site maar ook tijdens andere processen een rol spelen. Een voorbeeld hiervan zijn de prijzen, tijdens de solden is het niet reel om elke prijs handmatig aan te passen op de paginas. In het onwaarschijnlijke geval dat het handmatig aanpassen van een specifieke prijs nodig is, zou deze data enkel op de pagina gewijzigd zijn en niet in het ERP. Het is dus logischer om deze wijziging in het ERP te doen en deze te laten doorkomen op de pagina. Een tweede voorbeeld is een 1+1 gratis actie op bepaalde producten, deze willen we ook via het ERP kunnen instellen en laten doorkomen op de site alsook het weghalen hiervan.

Toen we aan dit project begonnen zijn, hebben we enkele manieren gevonden om deze wijzigingen weer te geven op de site en toen onze eerst twee sites, Juttu en Yaya, live stonden waren we tevreden over het resultaat. Maar deze shops zijn relatief nieuw en beperkt in aanbod, toen de vernieuwde AS Adventure site live ging staken er enkele problemen de kop op. Aangezien deze problemen in productie voorkomen was wachten niet echt een optie en hebben we deze

opgelost, niet zozeer met de beste maar wel de snelste oplossing.

Deze proef heeft als nut het herzien van wat we hebben opgebouwd, hoe we het AEM-platform aangepast hebben om aan onze behoeften te voldoen en of dit ook effectief de beste manier is om de vereiste functionaliteit te bekomen. Door het in kaart brengen van onze methodologien hopen we een dieper inzicht te verwerven in het platform op zich alsook een leidraad te voorzien voor ontwikkelaars die een eerste keer kennis maken met AEM of zij die reeds werken met AEM en geconfronteerd worden met diezelfde problemen waarvoor wij een oplossing zoeken. Zoals het gezegde luidt: het is goed te leren van je fouten maar beter om te leren uit die van een ander.

3 De database

Om onze AEM-applicatie te bouwen hebben we nood aan een database waaruit we onze informatie kunnen halen. De manier waarop we deze informatie halen zal afhangen van model tot model, sommige data zal naar AEM worden gestuurd wanneer deze gewijzigd is, andere zal worden opgehaald wanneer deze vereist is. We zorgen om deze reden dat beide gedachtegangen mogelijk zijn met elk model.

3.1 Apache Cassandra

Apache Cassandra is een opensource, NoSql (Not only Sql) database die de dag van vandaag door vele internationale bedrijven in gebruik is genomen. Netflix, Instagram en het CERN hebben in Cassandra een performante, stabiele databank gevonden, die de beschikbaarheid van hun data garandeert.

Deze garantie wordt geboden door de structuur waarmee Cassandra data beheert. Een databank bestaat uit een collectie van noden die een cluster vormen. Het beheer gebeurt decentraal, elke node heeft dezelfde rol, dit maakt dat wanneer een node offline is, de andere ongehinderd blijven werken. Om het volle potentieel van Cassandra te benutten, is het aangeraden om de noden niet enkel over verschillende servers maar ook over verschillende datacenters te verspreiden. Bij deze setup mag er een datacenter offline gaan, de gebruikers zullen hier geen weet van hebben. En wanneer het datacenter terug online is, zal de data gedeeld worden met de herstelde noden. Dit is ook positief voor de schaalbaarheid. Wanneer een node wordt toegevoegd, haalt deze zijn configuratie op bij een bestaande node en sluit zich naadloos aan in de cluster.

Bij het aanmaken van een cluster kan men een replicatie factor meegeven. Deze geeft aan op hoeveel verschillende noden een record moet bewaard worden. Bij een factor 3 zal elk record op 3 noden beschikbaar zijn. Des te groter de factor, des te groter de performantie en stabiliteit maar ook de duplicatie aan data vergroot mee. Bij het kiezen van de replicatiefactor moet er een afweging gemaakt worden tussen niveau van beschikbaarheid en het extra geheugen dat de replicatie vereist.

Een groot verschil met een relationele databank is dat het linken van verscheidene tabellen niet mogelijk is. In plaats daarvan werkt Cassandra met collec-

ties die rechtstreeks in een kolom worden opgeslagen. Dit wordt mogelijk door het aanmaken van UDTs (User Defined Types). Wanneer een lijst van UDTs (of primitieve types) wordt opgeslagen, wordt deze omgevormd naar een json-achtige structuur. Doordat alle data in n rij zitten, hoeven er geen joins gedaan te worden over meerdere tabellen wat de performantie aanzienlijk verhoogt.

Er zijn ook enkele nadelen verbonden aan het kiezen voor Cassandra als database, n van deze nadelen is dat het, out-of-the-box, onmogelijk is om tabellen te querien zoals je met een relationele databank zou doen. Dit komt doordat Cassandra werkt met key-value paren om de data op te slagen en dus enkel aan de hand van een key (of deel van een key) naar een record gezocht kan worden. Een bijgevolg hiervan is dat aggregatie functies enkel op partitie niveau kunnen worden uitgevoerd.

Aangezien Cassandra een non-relationele databank is, is een ander nadeel de afwezigheid van transacties op databank niveau. Enkel op rijniveau kan men gebruik maken van lightweight transacties door een conditie toe te voegen voor een schrijf operatie. Hiervoor zou men bv. een version id kunnen toevoegen aan een tabel zodat wanneer men een schrijf operatie wil doen, er gekeken kan worden of de versies nog matchen. Indien dit niet het geval is wordt de operatie niet uitgevoerd en de gebruiker hiervan op de hoogte gebracht.

3.2 Een Cassandra cluster opzetten

Nu we onze databank gekozen hebben, is het tijd dat we deze opzetten. De stappen die nodig zijn, beginnend van niets, om een cluster op te zetten worden in dit segment besproken. We zullen eerst de machines aanmaken, vervolgens Cassandra installeren en als laatste stap onze nodes met elkaar in contact brengen. Voor dit project heb ik gekozen om met 3 nodes te werken, dit om toch enige performantie te hebben zonder het prijzig te maken.

3.2.1 De machines

Als eerste maken we de 3 machines aan waarop we Cassandra zullen draaien. Via de AWS-console navigeren we naar EC2 -¿ Instances waar we de Launch instance kiezen. Het eerste scherm laat ons kiezen welk besturingssysteem we willen gebruiken, hier kiezen we Amazon Linux AMI. In stap 2 bepalen we hoe krachtig onze machines zullen zijn. Het is perfect mogelijk om met de t2.micro (deze is gratis te gebruiken bij een Free Tier) maar met slecht 1 CPU en 1GB RAM heeft deze niet zo veel te bieden. Hier heb ik voor de t2.large geopteerd, deze zal beter aansluiten bij onze behoeften. Stap 3 hoeven we enkel het aantal instances dat we willen aanmaken wijzigen naar 3, de rest laten we ongewijzigd. Stap 4 en 5 slagen we over, deze zijn niet van toepassing. In Stap 6 maken we

een nieuwe security group aan die de poorten nodig voor het gebruik van onze databank openzet. Voor onderstaande poorten maken we een Custom TCP Rule en maken we ze compleet publiek (in productie omgevingen is dit ten strengste af te raden.) door de toegestane IP-adressen op 0.0.0.0/0 te zetten. Figuur 1 toont ons hoe we dit bekomen.

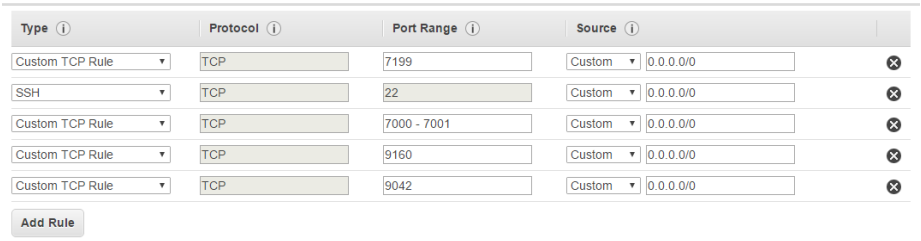


Figure 1: security settings.

Indien deze stap afgehandeld is, krijgen we een overzicht te zien. Als alles naar wens is klikken we op Launch waardoor er een prompt opengaat. Vanuit deze prompt kunnen we een pem-file aanmaken en downloaden, het is zeer belangrijk om deze op een veilige plaats op te slaan, deze nodig is om via ssh een connectie naar onze machines te maken. Bij het aanmaken van nieuwe machines kunnen we dezelfde pem-file hergebruiken zodat we n key hebben voor al onze instances. We slagen het bestand op onder aem-ec2.pem.

3.2.2 Cassandra als service

Nu we onze machines hebben is het tijd om hiervan Cassandra nodes te maken. Via het Instance scherm kunnen we achterhalen wat de publieke IP-adressen zijn van deze machines (laten we ervan uitgaan dat deze 1.0.0.1,1.0.0.2 en 1.0.0.3 zijn.). Volgende stap moet op elke machine identiek herhaalt worden buiten de nodige aanpassingen aan de cassandra.yaml. Laten we eerst op onze machines inloggen via het commando:

```
$ ssh  -i  aem-ec2.pem ec2-user@1.0.0.1
```

Nu zijn we ingelogd als ec2-user op onze machine. Om Cassandra 3.x te kunnen draaien hebben we Java 1.8 nodig, via het commando java -version kunnen we dit controleren. Indien de versie lager ligt, zijn we verplicht om Java 1.8 te installeren. Een mogelijke manier om dit te doen is als volgt:

We navigeren naar onze jvm folder

```
$ cd /usr/lib/jvm
```

Vervolgens downloaden we de java 1.8 tar file

```
$ sudo wget --no-cookies --no-check-certificate --
    header "Cookie: gpw_e24=http://www.oracle.com/;
    oraclelicense=accept-securebackup-cookie" http://
    download.oracle.com/otn-pub/java/jdk/8u121-b13/
    e9e7ea248e2c4826b92b3f075a80e441/jdk-8u121-linux-
    x64.tar.gz
```

En pakken we deze uit

```
$ sudo tar xzf jdk-8u121-linux-x64.tar.gz
```

Na het uitpakken kunnen we onze jdk registreren als java optie

```
$ sudo alternatives --install /usr/bin/java java /usr/
    lib/jvm/jdk1.8.0_121/bin/java 2
```

Als we nu het volgende commando runnen zien we twee java mogelijke java engines, diegene dat al genstalleerd was en onze nieuwe. Hier kiezen we om onze nieuwe als default te gebruiken.

```
$ sudo alternatives --config java
```

```
$ Enter to keep the current selection[+], or type
    selection number: 2
```

Als alles correct verlopen is zien we nu 1.8 staan wanneer we opnieuw het commando java -version uitvoeren.

Nu volgt de installatie van Cassandra zelf, hierbij is de eerste stap het toevoegen van de datastax.repo zodat we via het yum commando Cassandra kunnen installeren. Als dit gebeurt is kunnen we via yum zowel Cassandra als Nodetool installeren.

Maak het bestand datastax.repo aan.

```
$ sudo touch /etc/yum.repos.d/datastax.repo
```

Maak het bestand datastax.repo aan.

```
$ sudo touch /etc/yum.repos.d/datastax.repo
```

Vul deze met de nodige data

```
$ sudo touch /etc/yum.repos.d/datastax.repo
```

En zorg dat deze conform is aan Figuur 2

Nu kunnen we Cassandra en Nodetool installeren.

```
$ sudo yum install dsc30
```

```
[datastax]
name = DataStax Repo for Apache Cassandra
baseurl = http://rpm.datastax.com/community
enabled = 1
gpgcheck = 0
```

Figure 2: datastax.repo

```
$ sudo yum install cassandra30-tools
```

Als we nu Cassandra zouden opstarten op de 3 machines, hebben we 3 clusters met elks 1 node gemaakt. Vanzelfsprekend is dit niet het gezochte resultaat en willen we 1 cluster met 3 nodes, om dit mogelijk te maken moeten we de aanpassingen maken, getoond in Figuur 3 (locatie: /etc/cassandra/conf/cassandra.yaml).

```
cluster_name: 'Aem-cluster'

seed_provider:
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
          - seeds: "1.0.0.1,1.0.0.2"

listen_address:
# Het publieke IP-adres van de machine
broadcast_address: 1.0.0.1
# Dit stelt poort 9042 open aan alle thirth party applicaties waaronder
# DevCenter en Java-applicaties
rpc_address: 0.0.0.0
# Default gelijk aan het rpc_address waardoor
# we deze manueel op het publieke IP-adres
broadcast_rpc_address: 1.0.0.1
```

Figure 3: cassandra.yaml

Nu dat onze yamls correct staan kunnen we op onze machines cassandra starten met het commando sudo service cassandra start. Na enkele minuten zien we dan dat de 3 nodes elkaar gevonden hebben via het commando nodetool status.

3.2.3 DevCenter

Nu we onze databank hebben kunnen we onze nodige modellen toevoegen, het gebruikte script kan je terugvinden in de bijlagen. Voor het uitvoeren van deze scripts kan gebruikt maken van DevCenter, een open-source tool waarmee een

connectie kan gemaakt worden met een cluster. Deze manier van werken is aangenamer dan telkens te moeten sshen naar een node om daar via het cqlsh commando onze databank te kunnen querien. Onze drie tabellen die we gaan gebruiken zijn Product, Category en Price. Deze drie vertonen elk een andere graad van dynamiek, gepast voor het onderzoek dat we zullen verrichten. We voorzien telkens ook een log tabel voor deze drie zodat we aan de hand van een datum, die fungeert als ondergrends, de gewijzigde rijen kunnen opvragen. Eenmaal onze databank operationeel is kunnen we beginnen aan de service die deze zal aanspreken.

4 Data-service

4.1 De Service

Om onze databank aan te spreken gaan we een rest service voorzien waar we enkele simplistische methodes toekennen die het mogelijk maken om: de ids van de gewijzigde data periodiek op te halen aan de hand van een datum die fungeert als ondergrens en een endpoint waar we n element kunnen ophalen aan de hand van een id. Dit maakt het mogelijk om periodiek de gewijzigde producten, winkels en prijzen uit onze databank op te halen en naar AEM te versturen. Of deze manier van werken haalbaar is voor al onze modellen gaan we later uitmaken.

We gebruiken hiervoor een Java Spring-boot service omdat deze makkelijk in opzet zijn, gezien het een rest service is kan hier perfect voor een andere programmeertaal gekozen worden. Onze configuratie zullen we doen via een application.yml bestand, deze wordt automatisch door de laatste versie van Spring-boot ondersteund. We voorzien ook de docker-maven-plugin om het onszelf iets makkelijker te maken. We configureren deze zoals Figuur 4. De configuratie die we meegeven is redelijk transparant, imageName duidt aan hoe we onze docker images gaan noemen, een samenstelling van de artifact naam en versie zorgt ervoor dat elke versie van onze service een unieke imageName krijgt. Moest er dan ooit de nood zijn om een vorige versie terug te zetten, kan dit in een handomdraai. De dockerDirectory duidt waar we onze image willen aanmaken, de properties onder resources vertelt Docker wat er in de image moet worden opgenomen. Als Docker correct genstalleerd is, kunnen we het commando 'mvn clean package docker:build' uitvoeren om onze image te bouwen.

```
<plugin>
    <groupId>com.spotify</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    <version>0.4.11</version>
    <configuration>
        <imageName>${project.artifactId}:${project.version}</imageName>
        <dockerDirectory>src/main/docker</dockerDirectory>
        <resources>
            <resource>
                <targetPath>/</targetPath>
                <directory>${project.build.directory}</directory>
                <include>${project.build.finalName}.jar</include>
            </resource>
        </resources>
    </configuration>
</plugin>
```

Figure 4: Maven Docker Plugin.

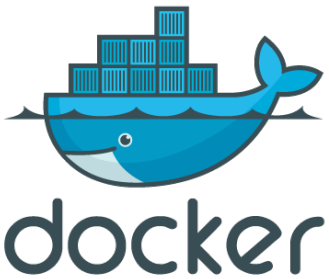
Om te connecteren met onze Cassandra cluster maken we gebruik van de Datas-

tax java-driver. Datastax is een bedrijf dat een commercile versie van Apache Cassandra aanbiedt alsook support voorziet. Hun Java-driver is open-source en dus gratis te gebruiken, in mijn ervaring biedt deze ook een betere ondersteuning voor het gebruik van UDTs tegenover de JPA-implementatie voor Cassandra. Om deze driver te gebruiken moeten we onze klassen die overeenstemmen met een tabel in onze databank annoteren met @Table. De velden in deze klassen krijgen nog de annotatie @Column en @PartitionKey als het gaat om een primary key kolom. De types van onze databank voorzien we ook klassen voor die we annoteren met @Field. De velden hier annoteren we met @Field. Eenmaal we onze tabellen hebben vertaald naar overeenstemmende klassen volstaat het om een sessie met onze cluster aan te maken, bij het aanmaken hiervan volstaat het om het IP-adres van n node te voorzien, de driver zal dan via discovery de locatie van de overige nodes vinden en zo fouttolerantie te waarborgen. Het is toch aan te raden om meerdere IP-adressen mee te geven zodat wanneer er een node niet beschikbaar is tijdens het aangaan van de sessie, deze kan terugvallen op een andere node.

Verder voorzien we drie rest controllers: n voor elk hoofdmodel dat we hebben voorzien. Voor onze AEM-applicatie te kunnen voeden moeten we minstens twee endpoints per controller moeten voorzien, n voor het ophalen van de gewijzigde IDs per model en n voor het effectief ophalen van de data.

4.2 Docker

De volgende stap is ons process is een manier zoeken om onze service in de cloud te draaien. Het is aanvaardbaar om tijdens de ontwikkeling onze service lokaal te draaien maar als we een volwaardig platform willen creeeren moet ook dit onderdeel remote draaien.



Docker is een open-source project dat dit process voor ons zal versimpelen. Vroeger moest er heel wat tijd (en bijgevolg ook budget) gespendeerd worden aan het werkende krijgen van services op verschillende machines. Dit komt omdat niet elke machine hetzelfde geconfigureerd is alles even goed ondersteund. Docker lost dit probleem op door maar n vereiste te hebben, dat de machine de Docker service heeft draaien. Oorspronkelijk ondersteunde enkel Linux systemen deze service native maar ondertussen zijn ook Windows en Mac mee op de kar gesprongen. Om deze lokaal te trainen, voor testing doeleinden, kan men naar de docker website gaat en de gewenste

versie te downloaden. De tutorials die je daar kan vinden leggen perfect uit hoe je ermee aan de slag kan. Natuurlijk gaan we, na een korte uitlichting, onze Docker remote gaan draaien.

Wie Docker zegt, zegt containers. Om onafhankelijk van de host een applicatie te kunnen draaien, steekt Docker deze, en al het nodige (environment, tools, libraries,settings,...), in een container. Docker bundelt al het nodige samen en maakt er een exporteerbare image van. Deze image kunnen we dan eender waar draaien zonder ons zorgen te moeten maken om infrastructurelen verschillen. Dit wil zeggen dat wanneer een image succesvol in een trainingsomgeving draait, deze zonder vrees overgezet kan worden naar een productie omgevening.

Een ander voordeel van het containersysteem is dat men elke container, tijdens het opstarten, specifieke parameters kan meegeven met betrekking tot de resources die deze ter beschikking krijgt. Als men enkele zwaardere services heeft, kan met de container hiervan meer RAM toekennen dan anderen om aan de behoefte te voldoen. Het is perfect mogelijk om meerdere containers op n machine of verdeelt over meerdere machines te draaien wat de schaalbaarheid en beschikbaarheid ten goede komt. Zelfs een release hoeft geen downtime meer te betekenen, de containers kunnen n voor n vervangen worden met een nieuwere versie.

Buiten de configuratie met betrekking tot de resources kan men ook omgevingsvariabelen meegeven (hoe dit mogelijk is zien we dadelijk). Dit heeft als voordeel dat we dezelfde container kunnen gebruiken voor onze trainings-en productieomgeving door ander variabelen, zoals de databaselocatie en credentials, urls van andere services, enz., mee te geven tijdens het starten van onze containers.

4.3 AWS en Docker

Nu wordt het tijd dat we onze services in de cloud gaan draaien. Hiervoor zijn enkele stappen nodig beginnend bij de installatie van de nodige software op onze ontwikkelings machine. De eerste installatie is die van de AWS Command Line Interface (of kortweg AWS CLI), gelukkig voor ons heeft Amazon hier een uitstekende handleiding[1] voor. Het is belangrijk niet te vergeten om na de installatie deze ook te configureren[2].

Het tweede wat we nodig zullen hebben is Docker op onze ontwikkelings machine, niet omdat we onze containers hier gaan draaien maar omdat we deze hier gaan bouwen. Voor diegene zonder ervaring met Docker raad ik deze manier aan om een beter begrip te krijgen van hoe dit in zijn werk gaat. De ervaren lezer mag natuurlijk zijn images op zijn gekozen manier bouwen. Om Docker te installeren

[1] http://docs.aws.amazon.com/cli/latest/userguide/installing.html
[2] http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html

kan de Linux-gebruiker zijn shell gebruiker, Mac en Windows hebben minder geluk en zullen een installer[3] moeten gebruiken.

Eenmaal onze ontwikkelings machine klaar is keren we terug naar AWS om een ECS-cluster (EC2 Container Service) op te zetten, hierin gaan we onze containers laten draaien. Een cluster kunnen we zien als een logische groepering van machines, wanneer we n dezelfde image meermaals deployen zullen de benodigde containers automatisch verdeelt worden over de machines in onze cluster.

Voor we aan onze cluster beginnen maken we snel 2 IAM roolen aan, n rol voor onze machines en n rol voor de service die we op onze cluster zullen starten. De rol voor de machines geven we een logische naam: ecs-instance-role en geven we de permissie AmazonEC2ContainerServiceforEC2Role Geeft de machine schrijfrechten op CloudWatch en ECS, leesrechten op ECR. De rol voor de service noemen we ecs-service-role en geven we de de permissie AmazonEC2ContainerServiceRole (schrijf-en leesrechten op EC2 en ELB).

Een ECS cluster opzetten is geen werk, even navigeren naar scherm, kiezen om een nieuwe te maken, we geven deze een naam(bv. tst-ecs-cluster) en voor nu selecteren we de optie 'Create an empty cluster'. Als we nu een cluster aanmaken dan zien we deze verschijnen onder cluster verschijnen maar zonder instances in, deze toevoegen is een volgende stap.

Vervolgens maken we een Launch configuration aan, dit zal beschrijven hoe onze machines opgestart moeten worden. De configuratie heeft als voordeel dat we dit maar eenmaal moeten doen en dat alle machines binnen onze cluster met de zelfde configuratie opstarten. DIt is grotendeels gelijk aan het maken van een machine, let wel op dat je als AMI[4] de Amazon ECS-optimized[5] kiest. Dit zal onze machines automatisch van Docker voorzien en deze ook starten wanneer de machine boot. Ken ook de ecs-instance-rol toe aan onze machines bij de stap Configure details. In deze stap voegen we ook een script toe onder User data (Advanced Details). Hier schrijven we het script als gezien in Figuur 5. Dit zal ervoor zorgen dat de machines die met deze configuratie starten aan onze cluster worden toegevoegd.

```
#!/bin/bash
echo ECS_CLUSTER=tst-ecs-cluster > /etc/ecs/ecs.config
```

Figure 5: ECS bootup script.

Nu we een Launch configuration hebben kunnen we deze gebruiken om effectief machines aan te maken. Dit doen we door een Auto Scaling Group aan te maken via de EC2 module. Tijdens het opzetten van deze groep kiezen we de Launch

[3] https://docs.docker.com/engine/installation/
[4] Amazon Machine Image
[5] te vinden onder AWS Marketplace

configuration die we juist hebben aangemaakt. We geven de groep een naam en hoeveel machines we willen bij de opstart ervan alsook een subnet, dit subnet kan gebruikt worden om strengere security filters in te stellen door enkel machines binnen dit subnet met elkaar te laten communiceren. Men zou dan via routing kunnen werken om een bepaald deel van de endpoints publiek toegankelijk te maken. In dit project gaan we hier niet verder op in. Onder Scaling Policies kunnen we beschrijven onder welke voorwaarden er meer machines moeten worden bijgezet, bijvoorbeeld als het CPU verbruik van onze cluster gedurende 5 minuten boven de 80% is. Omgedraaid kunnen we ook instellen wanneer er machines verwijdert mogen worden buiten de piekuren. Als alles goed verlopen is zien we na creatie een aantal machines, gelijk aan het gekozen aantal, opstarten onder ons EC2 Instances scherm. Deze machines zouden, dankzij het ecs-script, ook te zien zijn als bij onze ECS-cluster gaan kijken onder Instances.

Nu rest ons enkel nog het toevoegen van een loadbalancer en we zijn klaar om onze Docker containers te bouwen en te deployen naar onze nieuwe cluster. Een loadbalancer fungeert als gateway naar onze machines, inkomende requests op onze loadbalancer worden verdeelt en doorgestuurd naar de machines die de balancer als gezond beschouwt. Om de gezondheid van een machine te controleren doet deze periodiek een request naar de machines, zolang de machines met een status 200 antwoorden beschouwt de balancer ze als gezond. Hoe frequent en hoe vaak een antwoord positief of negatief moet zijn om de status van de machine te veranderen kan ingesteld worden. Ook de snelheid waarmee de machine moet antwoorden kan geconfigureerd worden, standaard is dit 5 seconden maar als je met gratis micro machines werkt verhoog je dit best. De netwerk capaciteiten van deze machines zijn beperkt waardoor de limiet van 5 seconden overschreven kan worden waardoor onze loadbalancer deze als ongezond zal beschouwen.

5 Architectuur van AEM

In dit hoofdstuk gaan we de bouwstenen van AEM bekijken, de frameworks die worden gebruikt om de geboden functionaliteit mogelijk te maken en de interfaces die we ter beschikking krijgen om onze applicatie te beheren. De reden dat we dit doen is omdat deze ingebakken zijn in AEM, wie met AEM aan de slag gaat, zal ongetwijfeld met deze technologien te maken krijgen. Het is aan te raden eerst deze sectie te lezen om een beter begrip te krijgen wat we doen en waarom dit doen tijdens de ontwikkeling van componenten en controllers. We beginnen bij de open source, en dus gratis, bestanddelen en bekijken vervolgens waarvoor we betalen.

5.1 OSGi

OSGi (Open Services Gateway initiative) is een framework dat ons in staat stelt om Java applicaties uit verschillende componenten op te bouwen. Deze componenten worden bundels genoemd die onafhankelijk van elkaar genstalleerd, verwijdert en vervangen kunnen worden in een OSGi container. Een bundel bestaat uit de jars en resources nodig voor de interne werking van deze bundel. Tijdens het bundelen kan er gespecificeerd worden welke packages zichtbaar zijn voor de andere bundels, als we niets configureren zal geen enkele package publiek beschikbaar zijn. Dit is in contrast met de normale werking van jars waar elke jar op het classpath aan alle klassen kan. Wanneer we een bundel installeren (of verwijderen) hoeven we de container niet stop te zetten. Dit geeft dat er geen down time is tijdens het updaten van onze applicatie.

Omdat de bundels onafhankelijk van elkaar genstalleerd worden, kan een bundel niet rechtstreeks rekenen op klassen voorzien door een andere bundel. Indien bundels toch afhankelijk zijn van elkaar worden er interfaces voorzien die geregistreerd worden in een service laag. Stel dat bundel A een externe StoreService gebruikt om een winkel te kunnen ophalen, zolang deze interface geregistreerd is in de service laag kan onze bundel starten zonder probleem. Wanneer bundel B start met een implementatie van StoreService (bv. StoreServiceImpl), kunnen we deze registreren in de service laag. Bundel A luistert naar veranderingen in de service laag en zodra StoreServiceImpl beschikbaar is, zal deze in gebruik genomen worden. In het geval dat bundel B verwijdert word, zal deze ongeregistreerd worden en zal bundel A hiervan op de hoogte zijn.

5.2 Apache Felix

Apache Felix is de open source OSGi-container van Apache en wordt gebruikt door AEM voor het installeren van de bundels die onze componenten bevatten.

In deze container zullen onze bundels draaien en met elkaar communiceren om samen de applicatie van zijn functionaliteit te voorzien.

5.3 JCR en Apache Jackrabbit

De Java Content Repository API (JCR) is een api die gebruikt kan worden om data op te slaan in een boomstructuur. Deze boom bestaat uit twee zaken: nodes en properties, nodes vormen de toppen van onze boom waardoor we kunnen navigeren, de top van onze bomen heet de rootnode. De properties zijn de blaadjes van onze toppen die effectieve data bevatten zoals een stuk tekst of getallen. We kunnen door onze nodes navigeren aan de hand van de methodes die de api beschikbaar stelt, we kunnen nagaan of onze node kinderen heeft en hoe deze heten alsook onze ouder bekijken. De data structuur van onze nodes kunnen we definiren aan de hand van een type. Dit type kan gebruikt worden om bepaalde velden af te dwingen alsook restricties omtrent de toegestane velden op te leggen. Dit type kan ook indiceren dat alles is toegestaan.

In functie van een CMS applicatie kunnen we onze boom beschouwen als de data op onze pagina's. Onze rootnode is onze homepage en diens kinderen vormen onze navigatie, wanneer we naar de pagina merken navigeren bewegen we ons vanaf onze rootnode naar diens kind 'merken'. De kinderen van de node 'merken' stellen dan elk een effectief merk voor. Omdat de data ongestructureerd is hoeft niet elk merk dezelfde properties te bevatten, tijdens het generen van de pagina kunnen we zaken tonen, of juist niet, aan de hand van de aanwezigheid van bepaalde properties. Dit is een voorbeeld om aan te tonen hoe je met JCR een website kunt opbouwen.

Apache Jackrabbit is Apaches implementatie van JCR en is gentegreerd in Apache Sling.

5.4 Apache Sling

Sling is een framework van Apache dat de gebruiker in staat stelt om REST calls te doen via http. Het ondersteund verscheidene bestand types waaronder json, xml en html. Sling is het framework dat AEM gebruikt om aan de hand van een url van een request de juiste node te vinden en diens data terug te geven in de correcte vorm. Via een extensie mee te geven kunnen we aan Sling duidelijk maken hoe we onze data willen, bv. door .json achteraan een url toe te voegen, weet Sling dat we de data als json terug willen.

Sling is reeds geintegreerd met zowel Apaches Jackrabbit alsook Felix waardoor het de eigenschappen van beiden heeft. Dit wil niet zeggen dat JCR de enige manier is om met Sling data op te slaan, indien gewenst kan er ook met andere

databanken zoals SQL of MongoDB gewerkt worden.

Wanneer een Sling bundel genstalleerd word, registreert het een ResourceProvider-Factory bij de Service laag van onze OSGi container.

5.5 CRX

Nu we de open source onderdelen van AEM hebben gezien is het tijd om de gelicentieerde delen bespreken. Het eerste dat we bespreken is Adobe CRX wat, simpelweg gezegd, Adobes implementatie is van Apache Jackrabbit en Slin. Buiten alle features die deze frameworks bieden heeft Adobe bijkomende functionaliteit toegevoegd, anders zou het maar raar zijn om hiervoor te betalen. We bekijken kort diegene die we gebruiken tijdens

Zoals we reeds gezien hebben is de datastructuur van JCR een boom waarin we kunnen navigeren. Een feature die CRX voorziet is een UI waarmee we kunnen navigeren door onze nodes, de CRXDE. Met deze interface kunnen we onze nodes bekijken zoals we een filesystem zouden gebruiken. We kunnen nodes expanderen en hun kinderen bekijken om ook deze te expanderen en zo dieper in onze boom af te dalen, of we kunnen de properties van de node zelf bekijken. De kracht van deze UI is dat we snel een gestructureerd overzicht van de nodes die onze website vormgeven.

De package manager is een ander belangrijk onderdeel van de CRX en geeft ons de optie om onze OSGi bundels via een interface te managen. Met deze interface kunnen we bundels aan onze applicatie toevoegen of bestaande bundels verwijderen. Buiten beheren welke bundels genstalleerd zijn kunnen we deze ook starten en stoppen vanuit de package manager. Dit versimpelt de manier waarop we onze applicatie beheren en is dus van grote toegevoegde waarden voor de ontwikkelaars.

5.6 subsec-aem

Alles wat we tot hiertoe gezien hebben heeft als functie het voorzien van componenten die bepaalde functionaliteit bezitten. Eenmaal we de componenten ontwikkeld hebben en via onze bundels genstalleerd zijn, is het tijd voor de designers aan de slag te gaan. De belangrijkste feature van AEM is de mogelijkheid om content te voorzien, gebruik makend van onze componenten. We kunnen deze slepen op templates, configureren en voorzien van inhoud. Een designer kan zo een volledige pagina, inclusief de navigatie hiernaar toe, opbouwen zonder een enkele html tag of javascript regel te schrijven. Wanneer een pagina af is, kan men deze ook publiceren via AEM zodat het resultaat door de wereld gezien kan worden.

De DAM (Digital Asset Manager) is een opslag plaats waar we onze verschillende media kwijt kunnen zoals foto's of video's. Eenmaal opgeslagen in de DAM kunnen we deze gebruiken op onze website door middel van referentie, later zien we hoe dit praktisch in zijn werk gaat.