1 Data-service

1.1 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.

[width=]images/maven-plugin.PNG

Figure 1: Maven Docker Plugin.

Om te connecteren met onze Cassandra cluster maken we gebruik van de Datastax 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.

1.2 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.

r0.5 [width=0.4]images/docker-logo.PNG

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.

1.3 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 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

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

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.

[width=0.5]images/ecs-script.PNG

Figure 2: 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 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 micro machines werkt verhoog je dit best. De netwerk capaciteiten hiervan zijn beperkt waardoor de limiet van 5 seconden overschreven kan worden.