Thema

Cloud computing is een hype in zo ongeveer elke IT-deelmarkt. Toch is het helemaal zo revolutionair niet; zaken als ASP en web services bestaan al een jaar of tien. Maar als softwarebouwers krijgen we meer en meer te maken met cloud-zaken, en ze kunnen écht ingrijpen in onze architectuur. Google AppEngine is een aardig voorbeeld van cloud-kracht, dat kan worden ingezet voor complete sites of voor bepaalde onderdelen. Wat zijn de sterke en zwakke punten van het model, en hoe beïnvloedt dat onze development?

Google AppEngine: kneedbare kracht

Ontwikkelen voor de cloud (1)

Erik de Ruijter
is IT-architect
bij ABM Amro bank.

Cloud computing is een verzamelnaam voor zo ongeveer alle logica die via het internet geboden wordt mede aan interne IT-gebruikers. Dus het kan een website-voorraadsysteem zijn, of een outsourced virusscan of backup, maar ook een complete reserve Citrix-server of 10 MIPS rekenkracht voor route-optimalisatie die over internet bijgeprikt wordt. En aan clouds zitten veel meer aspecten dan alleen die van ontwikkelaar. Denk maar aan beveiliging van al die 'externe' schakels, of het bereiken van de juiste uptime en performance; in dat laatste zijn clouds veelal 'service-orientatie in het kwadraat', en een SOA écht in de vingers krijgen was al aan zo weinig organisaties gegund....
Ook het aspect van contracten en nieuwe beheerderafspraken hoort niet primair op het developerbord thuis. We kijken in de komende nummers juist daarnaar: wat verandert er door cloud-bouwen vergeleken met standaard 3-tier applicaties?

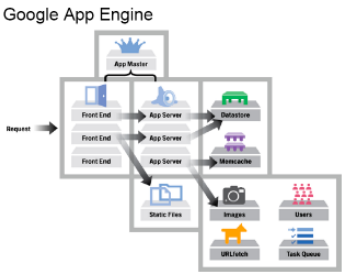
De markt kent drie technische niveaus die allemaal de naam 'cloud' kunnen dragen: het aanbieden van complete applicaties, van (runtime) platforms en van losse servers/services. Allen hebben ook populaire afkortingen gekregen:

- Applicatie-aanbod, oftewel SaaS (Software as a Service). Dit is niet fundamenteel anders dan de aloude ASP oftewel Application Service Provider: een outsource-partij kan een website bieden zowel aan onze klanten als medewerkers, en levert in ieder geval een deel van de businesslogica achter die site. Maar in bepaalde gevallen ook webservices, met bijvoorbeeld het weerbericht of kentekengegevens of routeplannerinfo. Bekende namen zijn Salesforce.com, Oracle On-Demand en Google Apps.
- Platform-aanbod, oftewel PaaS. De outsourcepartij biedt een hostingplatform voor onze zelf geassembleerde businesslogica, en ondersteunt dat met krachtige business services en bijbehorende ontwikkelomgeving. Bekende namen zijn Amazon EC2, Yahoo Merchant Solutions, Google AppEngine en Microsoft Azure.
- Losse servers of services. Er worden hiervoor twee namen gebruikt, IaaS (Infrastructure as a Service oftewel virtuele servers) en 'Internet services', maar in feite gaat het allemaal om IaaS. Want of nu de virtuele beschikbaarheid van een VM-guest-met-filesysteem geboden wordt of een SMTP bulkmailing service of Akamai cache, het is steeds een set 'bouwstenen' waar we zelf verantwoordelijk zijn voor alle businesslogica.

De eerste categorie, SaaS, is relatief 'saai'. In feite hebben we het over een gewone gehoste website met mogelijk wat business service-calls naar het interne datacentrum; en soms ook wat SOAP-services die als 'provider' geleverd worden. Vandaar dat we in deze serie primair kijken naar PaaS en IaaS vendoren. Overigens komen we de specifieke cloud connectivity zijde van SaaS ook bij PaaS tegen, dus daaraan besteden we de facto tóch ook aandacht. En de cloud-impact op het applicatieontwerp is niet onaanzienlijk. We krijgen bijvoorbeeld andere indelingen in het 3-tier model; aanpassingen

Java bleek voor Google onmisbaar om een breder publiek te bereiken.

op de IDE; specifieke aandacht voor latency; en natuurlijk inzet van hele specifieke cloud-API's in plaats van dingen zelf te bouwen of aan lokaal ingestelde libraries te verzamelen. Soms binnen de HTTP-envelop gewoon ook SOAP pratend, en soms heel anders ingerichte protocollen zoals JDBC/ODBC.



Figuur 1: AppEngine architectuur.

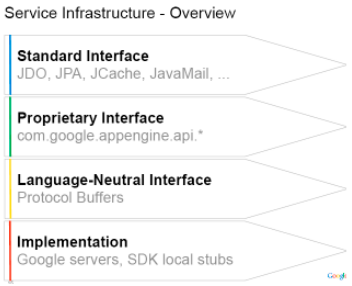
Google-aanbod en AppEngine positionering

Het bedrijf Google behoeft nauwelijks introductie. Het productaanbod wél, want door de verscheidenheid verliezen we regelmatig het overzicht. Grofweg kun je de aanbiedingen in drie groepen onderverdelen:

- Search. Niet alleen de overbekende website die zelfs het werkwoord 'googelen' aan de Van Dale deed toevoegen, maar ook aanverwante producten. Zoals AdWords, de advertentieverkoop gekoppeld aan de site maar ook inzetbaar als frame-onderdeel bij klantensites; dit is dé grootverdiener in de huidige commerciële Google-dienstverlening.
- Decentrale software. Dit zijn de Chrome browser en het Android smartphone-platform, en dit jaar komt daar voor netbooks het 'Chrome OS' bij.
- Applications. Dit betreft alle door Google aangeboden centraal draaiende logica die meer doet dan zoeken. Dit is de meest gemeleerde groep.

Allen hebben een gratis versie, en voor sommigen is het onzeker of ze ooit winstgevend kunnen worden zelfs met advertentieverkoop erbij; maar bijvoorbeeld Google Maps biedt ook SaaS webservices voor routeplanners, en Google Apps begint een aardige SaaS-aanbieder te worden die concurreert met lichtgewicht Office-suites en interne emailservers. En zelfs workflowprogrammering biedt, via een Cordys partnership. Grote namen in de Applications groep zijn Google Apps (inclusief GMail, Docs en Talk), Google Wave social networking/groupware, Maps/Earth/StreetView en Translate. Maar ook de minder direkt zakelijke produkten zoals Youtube, Picasa (foto's) en Orkut (communities) vallen onder Applications.

AppEngine is het PaaS-produkt dat bij de Google Applications-groep hoort. En de missie is om de klant, ons dus, dezelfde bouw- en hostingmogelijkheden te geven als Google intern zelf gebruikt. Dat kan omschreven worden als relatief simpele 3-tier applicaties die in hoge mate distribueerbaar zijn; mede om die reden wordt ook een erg simpele (en snelle) data tier gebruikt in plaats van een complexe SQL-struktuur.



Figuur 2: De AppEngine interface-lagen.

AppEngine basis

AppEngine is zoals gezegd voortgekomen uit interne Google-technologie. Natuurlijk zitten er in het interne gebruik ook engines zoals het zoekalgoritme die níet tot in detail voor derden beschikbaar zijn, maar het grootste deel van de basisomgeving waarmee succesvolle sites zoals Maps en Apps gebouwd zijn wordt nu geboden. Uitzonderingen zijn stukjes code die 'moeilijk beheersbare acties' plegen, zoals het efficiënt aanmaken van platte files wat Google Apps doet; AppEngine is op dit vlak nadrukkelijk gemuilkorfd, zodat de gehoste applicaties optimaal schaalbaar zijn. Maar voor vrijwel elke beperking is wel een oplossing te bedenken, al zal die soms door extra lagen slechter performen dan Apps zelf.

Google bood in eerste aanleg alleen Python als taal, ingedachtig de wortels van de eigen servers die in de scripttalenhoek liggen. Maar om een breder ontwikkelaarspubliek te bereiken, bleek Java toch onmisbaar, en er wordt nu ook een langzaam completer wordende Java-stack geboden. Nadrukkelijk gebaseerd op Standard Edition met wat extra's; Servlets en JSP's dus wel, EJB's en andere Enterprise (JEE) zaken niet. En door de opzet kunnen ook talen die via Java interpreteren gedraaid worden, onder andere JRuby en serverside JavaScript. We zullen verder de beschrijving vooral aan de hand van Java doen, maar de faciliteiten vanuit Python zijn gelijkwaardig.

Beide 'engines' worden gedraaid op Google-sites, in een vrij zwaar afgeschermde sandbox. Die zorgt ervoor dat applicaties een voorspelbaar gedrag vertonen in relatie tot de webload, en daarmee dus geen andere applicaties qua performance kunnen storen. Een aantal Java-opdrachten is daarom gemuilkorfd, onder andere het lanceren van subthreads en het genereren van bepaald netwerkverkeer. Google kent een url toe en zorgt dan zelf voor het deployen van onze eigen applicatie naar zoveel VM's als nodig is, in principe wereldwijd gespreid. Er wordt afgesproken op een bepaald verkeersvolume, en om dat te garanderen regelt Google de loadbalancing en het

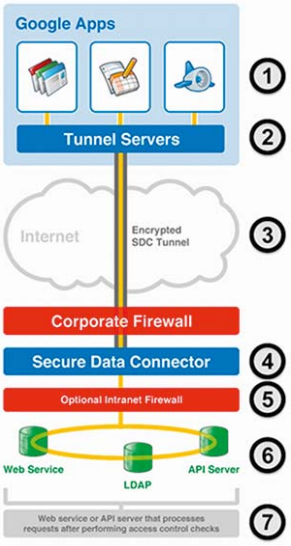
dynamisch op- en neerschalen al naar gelang het verkeer. Overigens zonder een formele Service Level Agreement (SLA) af te sluiten met onder andere uptime, op dit vlak is Google AppEngine minder professioneel dan een aantal cloud-concullega's.

Naast muilkorven van standaard Java wordt er ook 'gestuurd' en op punten uitgebreid. De geïmplementeerde API's bieden heel wat functionaliteit qua data-opslag en integreren met Google Apps. Maar ook voor beeldverwerking (Google Maps en Picasa!) wordt een uitgebreide set primitieven geboden. De VM delegeert uitvoering hiervan aan andere nodes in het Google-cluster, zodat de webperformance zelf beter voorspelbaar wordt. Ook kunnen we AJAX-applicaties maken middels Google Web Toolkit, maar die is niet AppEngine-specifiek; GWT is een clientside JavaScript-framework inclusief libraries dat voor veel webomgevingen inzetbaar is, en natuurlijk ook Google Docs van prik voorziet. En omdat standaard Java gebruikt wordt, zijn ook allerlei andere AJAX-varianten mogelijk, mits ze maar puur HTTP (dus ook SOAP) gebruiken voor communicatie tussen browser en server. Denk aan Adobe Flash/Flex en MS Silverlight.
Zoals we zien richt AppEngine zich in principe op complete applicaties; presentatielaag, businesslaag en datalaag. Daarmee is de positionering anders dan Amazon EC2 dat een meer 'granulaire' afname kent, en dan de Azure en IBM cloudservices; maar die twee zitten op de grens van IaaS en PaaS. De datalaag hoeft echter niet perse bij Google te liggen, maar kan gezien vanuit de engine ook 'extern' zijn. De formule is bedoeld voor data in ons eigen datacentrum. Met de 'Secure Data Connector' kunnen webservices benaderd worden alsof het dataopslag was; zelf moeten we in ons datacentrum dan nog heen en weer vertalen naar SQL, dus het is geen gemakkelijke route. Voor deze connector vooral, maar in principe voor elk applicatieproces, geldt dat de responstijd maximaal 30 seconden (!) mag zijn; ook hiermee beschermt Google de schaalbaarheid en performance van AppEngine-materiaal.
Voor het ontwikkelen van AppEngine-code geldt eenzelfde 'lichtgewicht' benadering als voor de keuzen qua geboden Java-functies. Gedeployed wordt met Standard Edition libraries, in .war-formaat. En die kun je lokaal aanmaken met de 'kale' AppEngine SDK of de wat meer rijke AppEngine Eclipse plugin. De SDK bevat ook mogelijkheden voor lokaal testen, dus onder andere valideren of de code in geïsoleerde vorm foutloos is kan zonder serverupload.

AppEngine speciale API's

Om nog wat scherper zicht te krijgen over hoe de Google Cloud het ontwikkelproces beïnvloedt kijken we wat nader naar twee specifieke groepen API's.
De eerste is de **dataopslag**. Al eerder was gemeld dat deze 'een erg simpele (en snelle) data tier gebruikt', en zeker geen volwaardige SQL-database. De nadruk bij het ontwerp van Google BigTable heeft gelegen op lees- en zoeksnelheid, niet perse op snelle updates. En de data-objecten kunnen vele veldtypen aan, inclusief XML-structuren, maar de database zelf kent geen relationele logica. Door deze keuze kan Google een stabiele en snelle replicatie aanbieden tussen alle AppEngine-nodes die een bepaalde site bedienen. Toch is de geboden functionaliteit niet mijlenver van SQL-niveau vandaan; transactionele updates en indexering zijn bijvoorbeeld wél weer te sturen, net als simpeler REST-persistentie. Ook kunnen relaties tussen objecten worden gedefinieerd, maar dat is dan primair om het optimale replicatiemodel van BigTable te kiezen; de API's voor queries zelf kennen geen gebruik van die relaties zoals dat met de SQL Join gebeurt. Opvallend is dat de ontwerpkeuzes van Google in lijn zijn met wat andere grote spelers doen. Microsoft bijvoorbeeld biedt weliswaar voor zware tools zoals Systems Center en Navision standaard SQL Server aan, en dat is ook een optie binnen MS Azure. Maar de lichtere en erg schaalbare Microsoft-produkten gebruiken geen SQL maar de 'Jet Engine', afgeleid van de MS-Access ISAM database; die vinden we onder andere onder de motorkap van Active Directory, MS Exchange en Sharepoint Server.
Een extra aanvulling op de opslag is MemCache. Dit is een 'distributed in-memory cache': dat betekent dat data hier beschikbaar zijn voor alle andere AppEngine nodes. Zaken die dus alleen tijdens een sessie persistent hoeven te zijn kunnen hierin bewaard worden, veel sneller dan in BigTable; zaken over sessies heen horen op disk. Doordat MemCache gebruikt wordt, is in AppEngine applicaties vrijwel geen behoefte aan server affiniteit, en dat scheelt een slok op de performance- en loadbalancer-borrel!
Voor 'platte bestanden' biedt AppEngine leesfuncties, maar geen schrijfopties. De bestanden dienen dan ook primair voor configuratiegegevens en informatie, en worden gedistribueerd tegelijk met de (te repliceren) applicatie. Er wordt gewerkt aan support voor het snel lezen van Megabyte-grote files, en voor een import hieruit naar de database toe.

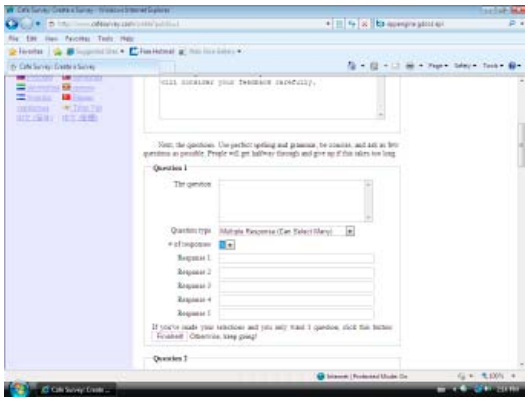
Dit is het officiële Google-verhaal. Het is echter 'niet de hele waarheid'. Juist door het open source-achtige ecosysteem rondom AppEngine ontstaan toevoegingen die de kloof met een SQL-database en een filesystem verder verkleinen. Jiql is zo'n aardige add-on, die toestaat om BigTable via JDBC te benaderen. Dat is vooral handig voor het porten van bestaande applicaties, of het verkleinen van de leercurve voor ontwikkelaars die deze API gewend zijn. Jiql voegt een stukje overhead toe, maar voor single-table opdrachten valt dat mee. Er wordt ook gewerkt aan Jiql-support voor 'join' queries, maar daarvoor is te voorspellen dat die erg traag gaat worden; de



Figuur 3: Secure Data Connector architectuur.

Door het open source karakter is de kloof tussen filesysteem en database kleiner geworden.

Kneedbare cloud-kracht zoals van AppEngine heeft toekomst.



Figuur 4.

joins moeten immers in de Jiql-laag uitgevoerd worden en niet door de database zelf. Een andere leuke add-on is GaeVFS, oftewel 'AppEngine Virtual File System'. Dit biedt een lees/schrijf filesysteem voor platte bestanden, dat onder water weer BigTable gebruikt en dat ook draait op Javaservers mét een standaard filesysteem. De positionering is net als Jiql: het is heel nuttig om bepaalde programma's en programmeerstijlen snel up and running te hebben, maar de keerzijde van de medaille is extra overhead vergeleken met meer direkte programmering.

Een tweede groep interessante API's is toegang tot **Google Docs**. Dit zijn de GDocs API's, die (als remote web service) deel zijn van Apps maar ook in AppEngine geboden worden. Momenteel zijn ze nodig om inkomende e-mail te kunnen verwerken, via een GMail API; maar die functie wil AppEngine volgens hun roadmap zelf gaan bieden, net zoals SMTP verzending nu al kan via de Javamail API. Een meer strategisch gebruik van GDocs samen met AppEngine is er bijvoorbeeld om vanuit een webapplicatie spreadsheet-kracht aan te roepen, of om een geformatteerd PDF document te laten aanmaken. Of simpelweg door de Google Charts API in te zetten, deze is met Docs verbonden en levert fraaie Excel-achtige grafieken. Dat zijn zaken die we bij concurrenten zoals Amazon of Yahoo niet snel zullen tegenkomen, en vormen een leuke kruisbestuiving binnen de Google-stal.

Architecture-gevolgen van AppEngine

Nu we de engine redelijk in breedte en diepte doorgescand hebben is het mogelijk een indruk te geven van de gevolgen voor developers. Die komen immers vanuit de hoek van dynamische webapplicaties die in het eigen datacentrum gehost zijn, voor Intranet en/of Internet; en cloud-werken eist gewoon aanpassingen.
Uitgaande van het basismodel 'alle drie lagen in de engine' zijn de aanpassingen niet eens zo groot. Als we echter bijvoorbeeld data via de Secure Data Connector weer uit onze eigen systemen halen, of SOAP calls gaan doen naar andere databronnen zoals beurskoersen, dan komt er een beduidend langere keten met ook echte 'cloud-latencies' waar het ontwerp strikt rekening mee moet houden.
De belangrijkste aanpassingen in ontwerp voor AppEngine zijn:

- Het data-opslagmodel. Geen platte files en geen SQL-data, vanwege de keuzen van Google BigTable die uiteindelijk voor onze eigen schaalbaarheidsbestwil zijn.
- Geen complexe logica per weboperatie. De processing mag maximaal 30 seconden duren, en moet binnen een enkele thread passen. (Batchverwerking mag in beperkte vorm, middels een cron-mechanisme.)
- Uitsluitend HTTP als protocol; operaties zoals SQL of e-mail moeten allemaal via API's van AppEngine, en soms 'ingepakt' worden in web services.
- Latency tussen de dynamische webapplicatie en de clients is beduidend beter dan bij interne hosting. Google zelf zal immers 'decentraal' hosten, op beperkte WAN-afstand van de client. Dat kan met centrale dynamische websites simpelweg niet, cacheproviders zoals Akamai kunnen alleen de 'statische' content versnellen. Zie hier nog een beduidend voordeel van het werken met een PaaS-cloud, dat overigens in principe ook met SaaS en ASP te bereiken is.

Google's kneedbaarheid

De vraag hoe het Google AppEngine-model onze development beïnvloedt is met het bovenstaande wel zo'n beetje beantwoord. En daarmee krijgen we ook een globaal beeld hoe dat bij andere PaaS cloud-providers zal gebeuren; hoewel die in zijn algemeenheid meer keuzen bieden om applicatielagen apart te gebruiken, onder andere Amazon met zijn opslag- en rekenservices. Maar dat aparte aanbod is eigenlijk meer IaaS dan PaaS en in de volgende afleveringen van deze serie gaan we dat stukje impact nader bekijken.
En dat geeft ook precies de positionering van AppEngine aan: Google wil niet perse op alle vlakken concurreren met Amazon of Microsoft, maar biedt vooral die technologie aan waar er synergie is met wat men toch al voor de andere Google-producten nodig had. Men is daarmee dus 'sterker' in bijvoorbeeld beeldbewerking en snelle datareplicatie, en 'zwakker' in relationele opslag. Dat zijn allemaal goed gefundeerde keuzen, en geven de zelfgekozen grenzen aan. Binnen die grenzen zijn nog ruimschoots mogelijkheden om met cloud-kracht webapplicaties te maken die qua respons en bouwgemak ver uitstijgen boven wat met 100% interne ontwikkeling haalbaar is - vergelijkbaar met bijvoorbeeld Google Maps of Youtube. Dus kneedbare Cloud-kracht zoals van AppEngine heeft toekomst, en gevolgen voor ons applicatie-ontwerp... «