SQL Data Services (SDS)

'DE MOEITE WAARD OM MEE TE EXPERIMENTEREN'

Steef-Jan Wiggers

SQL Data Services (SDS) is een nieuwe dienst van Microsoft die past binnen de mantra van 'Software as a Service', maar anders wordt genoemd: 'Data as a Service'. Het maakt deel uit van het Windows Azure Platform. SDS ondersteunt allerlei functionaliteit, die terug te vinden is in SQL Server 2008. Maar nu is deze te vinden op het internet waardoor ontwikkelaars kunnen beschikken over data voor mashups op een verantwoorde en veilige manier. Daarbij spelen beschikbaarheid en functionaliteit een belangrijke rol. In dit artikel laten we zien hoe ontwikkelaars deze dienst in de toekomst kunnen gebruiken.

Microsoft introduceerde in maart vorig jaar de online dataopslag. Dave Campbell had de leiding over de ontwikkeling van deze dienst die past binnen het plaatje van andere ontwikkelingen zoals hosted CRM, BizTalk-services, online infrastructuurservices voor SharePoint en Exchange2007. Al deze diensten worden gepositioneerd onder de noemer 'Software + Services', een visie afkomstig van Ray Ozzie. Het aanbieden van Microsoft-producten, die zich on-premise, binnen de organisatie, bevinden en worden beheerd off-premise, via het internet. Het geheel van deze diensten wordt nu door middel van het Windows Azure Service Platform aangeboden of heeft dit platform als uitgangspunt.

Microsoft wil zich met deze dienst richten op het Midden- en Kleinbedrijf (MKB) voor kostenbesparing en op ontwikkelaars van dataintensieve applicaties zoals mashups waarbij geen infrastructuur nodig is, althans investering daarvoor wordt overbodig. Tenslotte wil men zich ook richten op de enterprise klanten voor eenvoudige archivering en toegang tot publieke data, zoals brochures, catalogussen, artikelen en whitepapers, zodat opslag op media als tape of (SATA) schijf overbodig wordt.

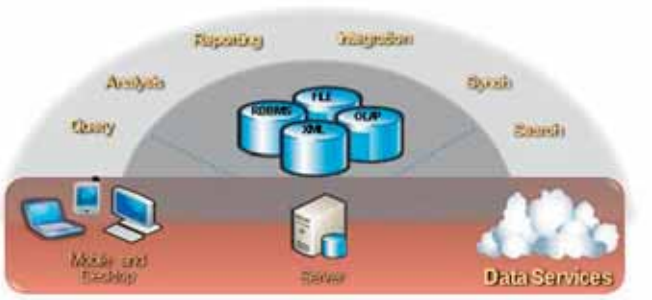
SDS dataopslag

Het concept van dataopslag via het internet is niet nieuw. Amazon webservices SimpleDB en GoogleBase bieden al dergelijke diensten aan. Microsoft zal echter met SDS meer elementen van SQL Server toevoegen, waardoor het meer zal brengen dan alleen dataopslag. Concurrentie voor deze dienst zal in de toekomst tussen de verschillende leveranciers toenemen. Mogelijk zullen er meerdere aanbieders komen. MySQL van Sun biedt een dergelijk model van dienstverlening bijvoorbeeld nog niet.
De voordelen die SDS biedt, kunnen worden onderverdeeld in drie categorieën: applicatieflexibiliteit, schaalbaarheid en Server Level Argreement (SLA). Vanuit applicatieflexibiliteit gezien kun je eenvoudig gebruik maken van standaardprotocollen zoals REST en SOAP om data via SDS te onttrekken, wat bijvoorbeeld voor mashups zeer handig is. Vanuit schaalbaarheid kun je je voorstellen dat SDS als service zeer schaalbaar is en op basis van gewenste capaciteit kan worden gebruikt en in de vorm van on-demand kan worden ingezet. De dienst maakt gebruik van robuuste Microsoft-technologie, is in hoge mate beschikbaar en betrouwbaar, waardoor je stevige SLA kunt realiseren.
SDS kan worden toegevoegd aan het rijtje diensten dat Microsoft SQL Server inmiddels biedt. Met deze nieuwe service heeft de klant de keuze data op te slaan binnen haar organisatie met SQL Server en/of daarbuiten op het net. In Figuur 1 is dit schematisch weergegeven.

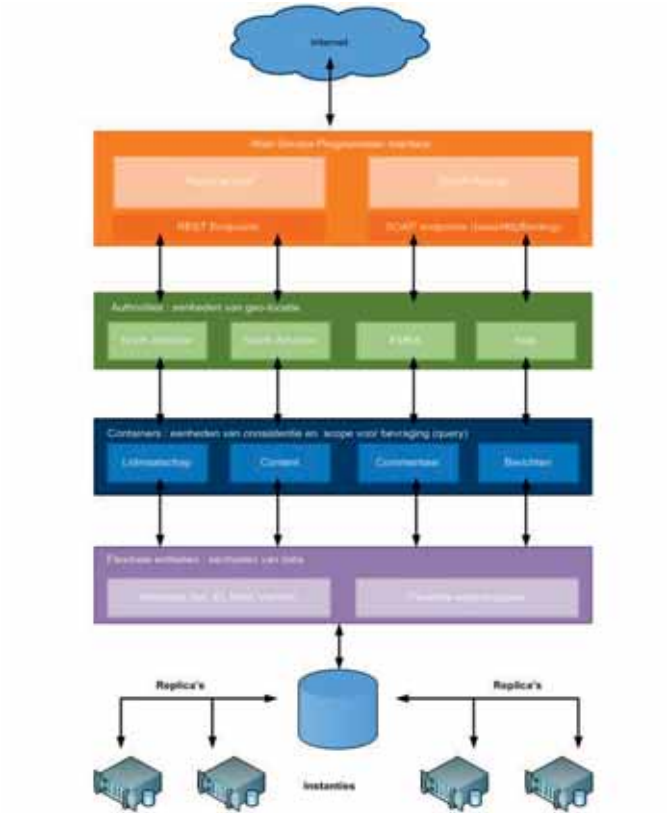
SDS Architectuur

Voordat we ingaan op de toepassing van SDS, kijken we eerst naar de architectuur van deze technologie. Zoals in figuur 2 is te zien, bestaat deze uit een aantal lagen. Als eerste de clientkant, die gebruik maakt van de SDS Service. Daarnaast is er een SDS runtime omgeving die bestaat uit een logische laag, opgedeeld in Authority, Container en Entity. Tenslotte is er de laag die verantwoordelijk is voor de opslag. Tussen de SDS runtime omgeving en clientkant kan gecommuniceerd worden met behulp van het SOAP- en REST-protocol.

Het datamodel en de programmeerinterface komen verderop in het artikel aan bod. De onderste laag is verantwoordelijk voor de



FIGUUR 1. MICROSOFT SQL SERVER ALS DIENST BINNEN EN BUITEN DE ORGANISATIE



FIGUUR 2. SDS ARCHITECTUUR

opslag van data. Instanties in het bovenstaande figuur zijn aangepaste geclusterde SQL Server 2008 instanties die draaien op Windows 2008 Servers.

Data model

SDS voorziet in een op entiteiten gebaseerd datamodel, dat drie belangrijke componenten bevat: Authority, Container en Entity. Vaak wordt ook wel van het ACE-model gesproken. Figuur 3 geeft de basiscompenenten van SDS weer.

Het is een implementatie van het entity-attribute-value (EAV) datamodel, waarbij het aanmaken van entiteiten eenvoudig is. Het model is flexibel, schemaloos en kan indien nodig ad hoc worden aangepast. Het EAV-model wordt meestal gebruikt om in een verzameling van entiteiten te kunnen voorzien, die veel eigenschappen nodig heeft om zichzelf te kunnen duiden, terwijl een individuele entiteit er maar een klein aantal behoeft. Een voorbeeld hiervan is een verzameling van patiënten, waarbij een arts per bezoek van een enkele patiënt een aantal observaties noteert.

Authorities

SDS Authority kun je beschouwen als een database in de relationele databasewereld. Wanneer je een authority aanmaakt genereert SDS een DNS-naam voor je, zodat je toegang krijgt tot je authority. Als je een authority aanmaakt, die je bijvoorbeeld SDS-dotnetmagazine noemt, dan kun je deze met behulp van de volgende URI benaderen:

sdsdotnetmagazine.data.beta.msds.com

Een Authority wordt ook gezien als een eenheid van geolocatie, wat inhoudt dat de aangemaakte DNS-naam overeenkomstig is met een datacenter van Microsoft waar jouw data wordt gehost.

Stel dat je twee authorities aanmaakt: **americasdsdotnetmagazine.data.beta.msds.com** en **europesdsdotnetmagazine.data.beta.msds.com** dan is elke geassocieerd met het datacenter voor haar gebruikers. In figuur 2 is dit ook duidelijk te zien.

Containers

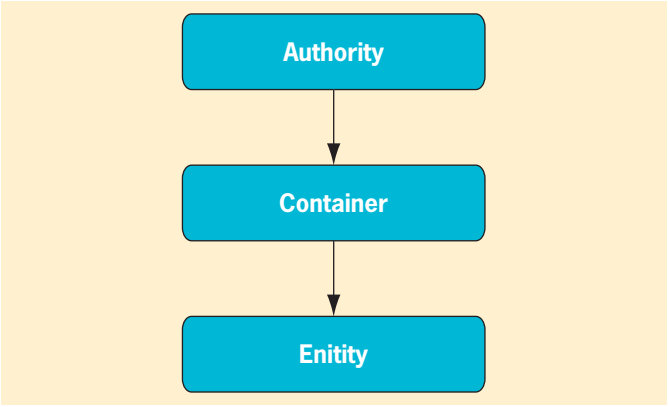
Authority bevat op haar beurt een verzameling van containers (zie figuur 2). Een container kun je vergelijken met een tabel in een relationele database. Het verschil tussen een container en een tabel is dat in SQL Server database je een schema toevoegt aan een tabel om alle rijen homogeen te maken. Dit geldt niet voor een container, omdat deze geen schema nodig heeft. Binnen een container zijn dus mogelijk verschillende soorten entiteiten op te slaan. In feite bevat een container een verzameling van entiteiten. Een voorbeeld om dat te verduidelijken. Binnen een relationele database als SQL Server kun je een medewekerstabel aanmaken en medewerkersgegevens opslaan als records. Al deze records zijn van hetzelfde type. In SDS maak je een medewerkerscontainer aan en sla je medewerkersentiteiten op. Deze entiteiten kunnen verschillende eigenschappen hebben, maar alle medewerkers zijn entiteiten. De container vormt in feite een omgeving waar entiteiten, je data consistent kunnen opslaan en waar bevragingen (query) op je data (entiteiten) kunnen plaatsvinden.

Entity

Tot slot heb je een entiteit die vergeleken kan worden met een rij (record) in een tabel van een relationele database. Een entiteit is simpelweg een verzameling van eigenschappen (property bag) van naam/waarde paren (name/value pairs). Deze paren worden onderverdeeld in twee categorieën namelijk: onderscheidbare systeemeigenschappen (distinguished systemproperties) en flexibele eigenschappen (flexible properties). De onderscheidbare systeemeigenschappen zijn gelijk voor elke entiteit:

- ID
- Kind
- Version

Het unieke van elke entiteit wordt weergeven door de 'ID' en zal binnen een container waar deze zich bevind dan ook uniek moeten zijn. Een entiteit met een gelijke 'ID' kan wel in meerdere containers voorkomen. 'Kind' wordt als eigenschap gebruikt voor het uit elkaar houden van gelijkwaardige entiteiten. Aangezien er geen schema aan een entiteit gekoppeld is, kunnen entiteiten met eenzelfde 'Kind'-eigenschap verschillen in structuur. Tot slot wordt 'Version' gebruikt om de huidige versie van een entiteit weer te geven. Deze eigenschap wordt bij operatie op de entiteit aangepast.



FIGUUR 3. BASIS COMPONENTEN VAN SDS

	Defenitie	Doel	Voorbeeld aan de hand van autobeurs
Authority	Verzameling van containers	Kunnen organiseren van containers voor veiligheid, co-locatie en betaling van gebruik	Amsterdam
Container	Verzameling van entiteiten	Kunnen organiseren van entiteiten voor context en bevragingen	Verkopen van auto's Tonen van auto's
Entiteit	Verzameling van eigenschappen	Eenheid van opslag	Verkopen van een bepaald auto weergeven als type, naam, prijs

FIGUUR 4. SCHEMATISCHE WEERGAVE VAN RELATIE TUSSEN AUTHORITY, CONTAINER EN ENTITY

Wat betreft de flexibele eigenschappen kan de ontwikkelaar deze bepalen op de data, die hij wil opslaan. Deze eigenschappen ondersteunen simple types als: string, decimal, bool, datetime, en binary.

Figuur 4 geeft nog schematisch de relatie tussen authorities, containers en entities weer.

Bij dit schema moet nog wel de opmerking worden gemaakt dat het 'provisioning model' van SDS, waar authority, container en entity deel van uitmaken, eraan ten grondslag ligt. Daar is een bepaalde hiërarchie aangegeven, die er als volgt uit ziet:

Klant{SDS account(1..N){Authority(1..N){Container(0..N){Entity(0..N).

Een klant kan middels één of meerdere accounts, één of meerdere authorities aanmaken, die op haar beurt nul of meerdere containers bevatten met nul of meer entiteiten. Elke account is gebonden aan een facturatiemechanisme gericht op het gebruik van het aantal authorities (vandaar de opmerking voor doel van authority betaling van gebruik). Aangezien SDS nog in de beta-versie verkeert is dit model momenteel nog niet van toepassing.

Progammeer Interface

SDS is – zo stelt Microsoft – een web 2.0 vriendelijke dataservice vanwege het feit dat het open standaard formaten ondersteunt zoals SOAP- en REST-interfaces. Hier kunnen ontwikkelaars dus met elke gewenste ontwikkeltaal gebruikmaken van de dataservices. Nu zullen enkele C#-voorbeelden worden geschetst voor SOAP en REST.

SOAP

Microsoft SDS levert een zogenaamd manipulatiemodel waarmee je operaties op authorites, containers en entiteiten kunt loslaten. Binnen SDS kun je de volgende operaties uitvoeren, die via SOAP en REST webinterfaces worden ondersteund:

- Aanmaken van autoriteiten
- Aanmaken en verwijderen van containers
- Aanmaken, veranderen en verwijderen van entiteiten.

Voor het aanmaken van een autoriteit kun je dus SOAP of REST gebruiken. Wanneer je met behulp van SOAP SDS wilt gebruiken, dan kan dit via HTTP of HTTPS (HTTP over SSL). Het laatste is minder snel dan het eerste, maar biedt wel de voordelen van beveiliging van data. SDS maakt standaard gebruik van HTTP en bij het aanleggen van een referentie naar de service zal standaard een app.config worden aangemaakt zoals in het onderstaande voorbeeld:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="SitkaSoapEndpoint" />
          ...
          <security mode="TransportCredentialOnly">
            <transport clientCredentialType="Basic" proxyCredential-
            Type="None"
                     realm="" />
            <message clientCredentialType="UserName" algorithmSuite=
            "Default" />
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://data.beta.msds.com/soap/v1"
      binding="basicHttpBinding"
         bindingConfiguration="SitkaSoapEndpoint" contract=
         "SitkaSSLClient.ISitkaSoapService"
         name="SitkaSoapEndpoint" />
    </client>
  </system.serviceModel>
</configuration>
```

Bij bovengaand voorbeeld van de app.config zie je naam Sitka staan, wat de codenaam is voor SDS beta. Door aanpassing van de volgende regels kan van SSL gebruik gemaakt worden. Keuze voor HTTP of HTTPS is afhankelijk van de behoefte van de dataconsumerende applicatie.

```
<security mode="TransportCredentialOnly">
<endpoint address="http://data.beta.msds.com/soap/v1"
        binding="basicHttpBinding" …
```

Zoals eerder beschreven kunnen er één of meerdere authorities worden aangemaakt binnen SDS en daarin kunnen één of meerdere containers worden opgeslagen als houder van entiteiten. Om een authority aan te maken, moet je eerst een lege scope creëren. De manier om objecten te adresseren binnen SOAP, is vergelijkbaar met URI binnen REST. Daar wordt namelijk het scope object binnen SDS gebruikt. De onderstaande code geeft weer hoe een authority wordt aangemaakt:

```
using (SitkaSoapServiceClient proxy = new SitkaSoapService
Client())
{
proxy.ClientCredentials.UserName.UserName = "username";
proxy.ClientCredentials.UserName.Password = "password";

// To target the "service" level you use the "empty" scope.
Scope myServiceScope= new Scope();
// Create authority
Authority auth = new Authority();
auth.Id = "myAuthorityName";
proxy.Create(myServiceScope, auth);
}
```

Bij het aanmaken van een authority moet je een gebruikersnaam en wachtwoord opgeven voor de service. Deze credentials heb je als een account aangemaakt bij SDS. Na het creëren van het scope-object en het opgeven van de credentials kun je de authority aanmaken. Als je de authority hebt aangemaakt, kun je vervolgens een container aanmaken. De volgende code geeft weer hoe dit moet.

```
// Identify scope. To create a container authority must be in
scope.
myAuthorityScope.AuthorityId = auth.Id;

// Create a container
Container c1 = new Container();
c1.Id = "myContainerName";
proxy.Create(myServiceScope, c1);
```

Bij het aanmaken van de container dien je een scopeobject aan te maken en de AuthorityID toe te kennen. Dit is de AuthorityID, die eerder is gecreëerd bij het aanmaken van de authority. Er is nu een containerobject aangemaakt en scope aangepast om naar de authority object te laten wijzen. Daarnaast is ook verwijzing naar container in de scope gecreëerd. Na het aanmaken van de authority en container wil je deze natuurlijk vullen met entiteiten.

```
// Now create a new entity
Entity e1 = new Entity();
e1.Id = "SomeId";
e1.Kind = "myEntityKind";
e1.Properties = new Dictionary<string, object>();
…
// Create
proxy.Create(myServiceScope, e1);
```

Samengevat is er nu een authority aangemaakt met een container, waar zich een entiteit in bevind.

REST

Met behulp van het REST-protocol zijn de dezelfde operaties uit te voeren als met SOAP. De werking is echter anders. Elke REST-authority heeft zijn eigen DNS-naam: **https://sdsdotnet-magazine.data.beta.msds.com/v1**. SSL, HTTPS is verplicht voor het bewaren van user ID, wachtwoord en databeveiliging voor het REST-protocol. Voor het aanmaken van een authority dien je eerst een XML-document te maken waarin de identifier van de auhtority zich bevindt, die geheel lower-case dient te zijn.

```
<s:Authority xmlns:s='http://schemas.microsoft.com/sitka/2008/03/'>
   <s:Id>NewAuthorityId</s:Id>
</s:Authority>
```

Met behulp van een HTTP-request naar de SDS service met de POST-methode kun je het XML-document versturen. Deze service weergegeven in onderstaande code zal een response sturen met een HTTP-status, waarin duidelijk is of de operatie is gelukt of niet.

```
using System;
using System.Text;
using System.Net;
using System.IO;

namespace CreateAuthorityUsingREST
{
   class Program
   {
      // Provide your values for these members variables
      private const string userName = "YourUserName";
      private const string password = "YourPassword";
      // Authority Id below must be unique in the service.
      private const string sampleAuthorityId = "YourAuthorityId";

      private const string XmlContentType = "application/xml";
      private const string HttpPostMethod = "POST";

      static void Main(string[] args)
      {
         string ServiceUri = "https://data.beta.msds.com/v1/";
         string authorityUri = CreateAuthority(ServiceUri);
      }

      private static string CreateAuthority(string serviceUri)
      {
         // Template to create authority request body. The only
         thing that
         // is variable is the authority id.
         const string AuthorityTemplate =
                  @"<s:Authority xmlns:s='http://schemas.microsoft.
                  com/sitka/2008/03/'>
                  <s:Id>{0}</s:Id>
                 </s:Authority>";

         if (String.IsNullOrEmpty(serviceUri))
         {
            throw new ArgumentOutOfRangeException("ServiceUri");
         }

         string authorityUri = null;
         try
         {
            // Data going over to the server
            string requestPayload = string.Format(Authority-
            Template, sampleAuthorityId);

            HttpWebRequest request = (HttpWebRequest)HttpWeb-
            Request.Create(serviceUri);
            // Security requirement - need to pass userid,
            password
            request.Credentials = new NetworkCredential(userName,
            password);

            // POST=create; PUT=update; DELETE=delete;
            GET=retrieve
            request.Method = HttpPostMethod;
            UTF8Encoding encoding = new UTF8Encoding();
            request.ContentLength = encoding.GetByteCount
            (requestPayload);
            request.ContentType = XmlContentType;

            // Write the request data over the wire (1st container).
            using (Stream reqStm = request.GetRequestStream())
            {
               reqStm.Write(encoding.GetBytes(requestPayload),
               0, encoding.GetByteCount(requestPayload));
            }

            // Get the response and read it in to a string.
            HttpWebResponse response = (HttpWebResponse)request.
            GetResponse();
            if (response.StatusCode != HttpStatusCode.Created)
            {
                  Console.WriteLine("Failed to create authority");
            }
            // Container created successfully. Get the container URI.
            authorityUri = response.Headers[HttpResponseHeader.
            Location];
         }
         catch (WebException ex)
         {
            using (HttpWebResponse response = ex.Response as
            HttpWebResponse)
            {
                  if (response != null)
                  {
                  string errorMsg = ReadResponse(response);
                  Console.WriteLine(string.Format("Error: {0}",
                  errorMsg));
                  Console.WriteLine("Unexpected status code
                  returned: {0} ", response.StatusCode);
                  }
            }
         }

         return authorityUri;
      }
   }
}
```

Om uiteindelijk te controleren of de authority is aangemaakt, kun je gebruik maken van de volgende URI: **https://<authority-id>.data.beta.msds.com/v1/**. Aanmaken van een container gaat op een vergelijkbare manier waarbij eveneens een XML-document aangeboden wordt met behulp van een HTTP-request naar de SDS service via de POST-methode. Het XML-document ziet er dan als volgt uit:

```
<s:Container xmlns:s='http://schemas.microsoft.com/sitka/2008/03/'>
   <s:Id>NewContainerId</s:Id>
</s:Container>
```

De container identifier dient uniek te zijn zoals ook aangegeven is in het SOAP-voorbeeld. De code voor het aanmaken is vergelijkbaar met het aanmaken van de authority en controleren gaat door middel van de volgende URI: **https://<authority- id>.data.beta.msds.com/v1/<container-id>**. Als je bijvoorbeeld een container hebt aangemaakt met container identifier 'autorai' dan zou je met de URI **https://<authority-id>.data.beta.msds.com/v1/AutoRai_REST** de volgende XML-response kunnen verwachten:

```
<s:Container xmlns:s="http://schemas.microsoft.com/sitka/2008/03/"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:x="http://www.w3.org/2001/XMLSchema">
  <s:Id>AutoRAI _ REST</s:Id>
  <s:Version>1</s:Version>
</s:Container>
```

Tot slot geldt voor het aanmaken van entiteiten eveneens dat je een XML-document aanbiedt aan de SDS Service via de POST-methode. En wederom met een URI zoals deze **https://<authority-id>.data.beta.msds.com/v1/<container-id>/<entity-id>** kun je controleren of de entiteit succesvol is aangemaakt.

```
<Auto xmlns:s="http://schemas.microsoft.com/sitka/2008/03/"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:x="http://www.w3.org/2001/XMLSchema">
   <s:Id>AutoREST</s:Id>
   <s:Version>1</s:Version>
   <merk xsi:type="x:string">Mercedes Benz</merk>
   <type xsi:type="x:string">MLK 220 CDI</type>
   <prijs xsi:type="x:string">35000 euro</prijs>
</Auto>
```

Querying SQL Server Data Services

Door het aanmaken en uiteindelijk vullen van containers met entiteiten wil je de data uiteindelijk een keer opvragen. In feite is nu het moment aangebroken dat je met de data aan de gang wilt en deze wilt opvragen, aanpassen of verwijderen. Microsoft SDS ondersteunt een op tekst gebaseerde querytaal, welk het LINQ pattern voor C# volgt. De taal is zodanig ontworpen dat het een eenvoudige bevraging met behulp van filters ondersteunt. Deze kunnen plaatsvinden op enkele authorities of containers waarbij je rekening dient te houden met het volgende:

- Een authority kan bevraagd worden met containers, die voldoen aan een gespecificeerde conditie. De scope is daarbij beperkt tot een enkele authority. Bevragingen over verscheidene authorities is niet mogelijk.
- Een container kan bevraagd worden waarbij de entiteiten aan een gespecificeerde condities voldoen. De scope is ook hier beperkt tot een enkele container. Bevragen over meerdere containers is niet mogelijk.

Momenteel wordt bij bevragen van entiteiten de volledige entiteit geretourneerd. Daarnaast zijn de selectiecondities beperkt tot eenvoudige vergelijkingsoperators zoals: <,>,<=,=>,!=,== en logische operators zoals: &&, ||, !. De syntax voor een bevraging (query) is als volgt:
from e in entities [**where condition**] **select e**

De bovenstaande query zal door middel van een interactie over een aantal entiteiten heen gaan en entiteiten retourneren die voldoen aan de gespecificeerde conditie. Als we het voorbeeld van de auto nog eens nemen, kunnen we een volgende query schrijven:

```
from e in entities
where e["Name"] == "Mercedes Benz" &&
      e["Type"] == "MLK 220 CDI"
select e
```

In de bovenstaande query gericht op een bepaalde container worden alle auto entiteiten bevraagd waarbij de naam van de auto Mercedes Benz is en type MLK 220 CDI.

Query via SOAP

Via de SOAP webinterface is het mogelijk bevragingen te doen aan zowel de entiteiten als de authority en container. Daarbij kun je de GET-methode gebruiken of de Query-methode. De GET-methode kun je het beste toepassen als de identifier bekend is van je authority, container of entiteit. De volgende code geeft een weerspiegeling van het bevragen van een authority. Je maakt hier eveneens weer gebruik van een scope gericht op de authority die je wilt bevragen.

```
Scope myAuthorityScope = new Scope();
myAuthorityScope.AuthorityId = authorityId;
Authority auth = (Authority)proxy.Get(myAuthorityScope);
```

De Query-methode gebruik je op het moment dat het ID van je authority, container of entiteit niet bekend is. Daarbij dien je de scope dusdanig te specificeren op de query, die je wilt uitvoeren. De code hiervoor staat hieronder, waarbij een bepaalde container bevraagd dient te worden om daar entiteiten uit te verkrijgen op basis van een bepaalde conditie. Als voorbeeld nemen we wederom de auto, waarbij we alle type auto's van een bepaald merk willen ophalen.

```
Scope myContainerScope = new Scope();
myContainerScope.AuthorityId = authorityId;
myContainerScope.ContainerId = carContainerId;
string sampleQuery = @"from e in entities
            where e[""Name""] == ""Mercedes Benz""
            select e";
IEnumerable<Entity> cars = proxy.Query(myContainerScope,
                           sampleQuery);
```

Query via REST

Naast het SOAP-protocol kun je ook via het REST-protocol bevragingen doen aan authority, container of entiteit. Daarbij heb je eveneens twee methodes zoals bij SOAP-interface: de GET en Query-methode. De query kan met behulp van een browser door daar simpel weg de URI in te tikken. Daarbij moet wel rekening gehouden worden met karakters, die een speciale betekenis hebben zoals ? of ;. Deze dienen te worden vervangen door %3F en %3B en een spatie wordt %20. Een logische operator als && wordt %26%26. Voor een query zoals **from e in entities where e["Name"] == "Mercedes Benz" && e["Type"] == "MLK 220 CDI" select e** binnen een bepaalde container wordt de URI voor binnen de adresbalk van de browser als volgt: **https://<authority_id>.data.beta.msds.com/v1/<container_id>?q='from%20e%20in%20entities%20where%20e["Name"]%20==%20"Mercedes%20Benz"%20**

%26%26%20e["Type"]%20==%20"MLK%20220%20CDI"%20select%20e.
Voor de GET-methode geldt dat als je URI gebruikt binnen REST je met de volgende URI een authority bevraagt: **https://<authority-id>.data.beta.msds.com/v1/**. Deze zal als resultaat een XML-document van de authority teruggeven. Indien je alle entiteiten in een bepaalde container wilt ophalen met de volgende URI: **http://sdsdotnetmagazine.data.beta.msds.com/v1/cars?q="** kan het volgende resultaat verwacht worden:

```
<s:EntitySet
    xmlns:s="http://schemas.microsoft.com/sitka/2008/03/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:x="http://www.w3.org/2001/XMLSchema">
  <carKind>
    <s:Id>5fdc7c86-b3de-4f6f-a424-3e9c4d9215ad _ MySampleBook</s:Id>
    <s:Version>1</s:Version>
    <Name xsi:type="x:string">Mercedes Benz</Name>
    <Type xsi:type="x:string">MLK 220 CDI</Type>
    <Price xsi:type="x:string">35000</Price>
  </carKind>
<carKind>
...
</carKind>
...
</s:EntitySet>
```

Er zijn meerdere voorbeelden te bedenken voor het bevragen van gevegens uit een authority, één of meerdere containers en/of entiteiten. Zoals bijvoorbeeld het opvragen van grote hoeveelheden data uit entiteiten, waarbij je van het zogenaamde paging principe gebruik kunt maken. Dit houdt in dat een grote hoeveelheid van data niet in één keer wordt opvraagt of wordt getoond aan een gebruiker. Op basis van een entiteit identifier wordt een bepaalde hoeveelheid van data opgehaald en/of getoond. Onderstaande code is daar een illustratie van, waarbij auto's van 5000 euro of duurder worden opgevraagd met een beperking tot 200 resultaten.

```
using (SitkaSoapServiceClient proxy = new SitkaSoapService
Client())
{
   Scope myContainerScope = new Scope() {
   AuthorityId = authorityCarId,
   ContainerId = containerCarId};

   string lastId = "";
   string query = @"from e in entities
        where e[""Price""] => ""5000"" &&
    e.Id > ""{0}""
        select e";
   while (true)
   {
      int count = 0;
      foreach (Entity entity in proxy.Query(
                           myContainerScope,
                           String.Format(query, lastId)))
      {
         // Process entities
         lastId = entity.Id;
         count++;
      }
      if (count < 200)
           break; // last page if less than 200 results
   }
}
```

Berperkingen

SDS is nog in de Beta-versie en kent zijn beperkingen. Daar dien je rekening mee te houden. Als je een authority wilt aanmaken, zal de identifier geheel lower-case moeten zijn. Verwijderen van authority is in de beta nog niet mogelijk. De grootte van de container is gelimiteerd tot 2 Gb en van de entiteit grootte tot 2 Mb. Wat betreft het uitvoeren van queries zijn er deze beperkingen:

- JOIN's worden niet ondersteund;
- Queries over meedere containers zijn nog niet mogelijk;
- Vergelijkings- en logische operatoren zijn beperkt;
- Indexering over maximaal 256 Kb.

Het kan zeker geen kwaad de documentatie rond SDS goed te besturen en codevoorbeelden te bekijken en guidelines te lezen.

Toekomst

In de toekomst zal meer functionaliteit aan SDS worden toegevoegd. Zoals in dit artikel naar voren is gekomen, is SDS nog in Beta-versie en maakt het onderdeel uit van het Windows Azure Service Platform CTP. Voor ontwikkelaars zal er ondersteuning zijn of gaan komen voor andere ontwikkeltalen zoals Visual basic, Java en Ruby of zelfs voor Microsoft Office Access. Voor SDS zal meer ondersteuning komen voor protocollen zoals AtomPub en JSON.

Deze nieuwe service van Microsoft is zeker de moeite waard om kennis van te nemen en/of om een experiment mee aan te gaan om het nader te onderzoeken. De concurrentie zal zeker niet stil zitten nu Microsoft deze service in de toekomst betaald gaat inzetten. Daarom kun je verwachten dat deze dienst er zeker zal gaan komen.

Links

Test-Drive SQL Server Data Services - Roger Jenning VS Magazine July 2008;
SQL Server Data Services (SSDS) Primer - http://msdn.microsoft/en- us/library/cc512417.aspx;
Microsoft SQL Server Data Services - http://www.microsoft.com/sql/dataservices/default.mspx;
SSDS blog - http://blogs.msdn.com/ssds/default.aspx;
Introduction to Microsoft SQL Server Data Services - http://oakleafblog.blogspot.com/2008/07/teched-2008-it- pro-session- dat251.html;
Dave Campbell: SQL Server Data Services and the Future of Data in the cloud - Channel 9 - http://channel9.msdn.com/shows/Going+Deep/Dave- Campbell-SQL-Server-Data-Service-and- the-Future-of-Data- in- the- Cloud/;
SQL Server Data Services: Good news, bad news for partners - Barbara Darrow - http://searchitchannel.techtarget.com/news/article/0,289142,sid96_gci305490,00.html;
Microsoft Extends SQL Server To The Web With Data Services - J.Nicholas Hoover InformationWeek - http://www.informationweek.com/news/internet/showArticle.jhtml?articleID=206901860;
Cloud database vendors: What, us worry about Microsoft? SQL Server Data Services not inspiring panic in the competition - Eric Lai - ComputerWorld - http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9067979;
Microsoft adds database to list of online services - John Fontana - Networkworld - http://www.networkworld.com/news/2008/030508- microsoft-database-online-services.html
Develop Robust and Scalable Apps with SQL Server Data Services - David Robinson - http://msdn.microsoft.com/en- us/magazine/cc700349.aspx;
SSDS Examples - Mike Admundsen http://amundsen.com/examples/ssds/.

Steef-Jan Wiggers is Information Architect bij Inter Access B.V. Hij is te bereiken via steef.jan.wiggers@interaccess.nl en heeft een blog over o.a. BizTalk (http://www.soa- thoughts.blogspot.com/)