# Verder bouwen aan een reallife applicatie

## Applicaties in de wolken - Deel 2

**Maarten Balliauw**

In het vorige nummer van .NET magazine ben ik gestart met de bouw van een reallife applicatie voor het Windows Azure platform. De focus van dat artikel lag op de verschillende componenten waarmee een Windows Azure applicatie als TwitterMatic gebouwd kan worden. De verschillende opslagdiensten werden toegelicht, alsook de features die eind 2009 op de PDC werden voorgesteld aan het grote publiek. Nu ga ik dieper in op de ontwikkeling zelf: hoe worden de opslagdiensten precies gebruikt, waar moet ik op letten bij ontwikkeling voor Windows Azure? En vooral: hoe krijg ik mijn applicatie in de cloud?

Voor wie de vorige editie van .NET magazine gemist heeft: de applicatie die in deze reeks gebruikt wordt is TwitterMatic, een applicatie die gebruik maakt van de sociale netwerksite Twitter, waar je met een bericht van maximaal 140 tekens aan de wereld kunt vertellen waar je mee bezig bent op dat moment. TwitterMatic laat toe om zogenaamde 'tweets' te plannen en automatisch te posten op Twitter, zelfs als er niemand aanwezig is op kantoor. De beoogde architectuur is vrij eenvoudig: een ASP.NET MVC applicatie zal worden gehost in Windows Azure als web role, in een worker role op de achtergrond loopt een service die geplande tweets controleert en indien nodig ook het posten naar Twitter verzorgt. Alle gegevens worden bewaard in twee opslagsystemen: Windows Azure Table Storage, een erg eenvoudige en gelimiteerde database, en de Windows Azure Queue Service, een gelimiteerde manier om berichten in een wachtrij te plaatsen voor verwerking.

### ASP.NET MVC als front-end in een web role

De front-end van TwitterMatic is een ASP.NET MVC applicatie, die in een Windows Azure web role wordt gehost. De opzet is vrij eenvoudig: de front-end biedt de mogelijkheid om geplande tweets te beheren en doet dit met een aantal classes uit een gedeelde domain layer. Deze domain layer biedt ook zaken als validatie aan: een tweet moet bijvoorbeeld steeds een statusupdatebericht bevatten. Dit wordt afgedwongen door middel van 'DataAnnotations'. Dit is een concept dat in ASP.NET Dynamic Data naar voren is gekomen, ondertussen is opgepikt door de WCF RIA Services voor Silverlight en vanaf ASP.NET MVC versie 2 ook standaard ondersteund zal zijn. In de voorbeeldcode is echter gefocust op ASP.NET MVC 1, waardoor de DataAnnotations alleen met wat loodgieterij kunnen worden gebruikt. DataAnnotations zijn een krachtig instrument waarmee een domeinclass kan worden verrijkt met extra attributen die bijvoorbeeld validatie kunnen afdwingen. De TimedTweet class die wordt gebruikt in TwitterMatic ziet er als volgt uit:

```
public class TimedTweet : TableServiceEntity, IComparable
{
    public string Token { get; set; }
    public string TokenSecret { get; set; }

    [Required(ErrorMessage = "Twitter screen name is required.")]
    public string ScreenName { get; set; }

    [Required(ErrorMessage = "Message is required.")]
    [StringLength(140, ErrorMessage = "Message length must not
    exceed 140 characters.")]
    public string Status { get; set; }

    // ... meer properties ...
}
```

In dit voorbeeld komen twee types DataAnnotations terug: Required en StringLength. De eerste markeert een eigenschap als 'required': de eigenschap moet een waarde bevatten om de validatie te laten lukken. StringLength zorgt er dan weer voor dat een eigenschap geen string bevat die langer is dan het opgegeven aantal tekens. Optioneel kan bij een attribuut uit de DataAnnotations namespace nog een foutboodschap worden meegegeven. Ook aan lokalisatie is gedacht: de meeste attributen kunnen makkelijk worden gekoppeld aan een resource bestand waarin vertalingen van foutboodschappen zijn opgenomen.

### Table Storage

De TimedTweet class erft over van de class TableServiceEntity. Dit is een 'base-class' uit de Windows Azure SDK die het mogelijk maakt om een object op te slaan in Windows Azure Table Storage. Deze base class bevat drie eigenschappen (PartitionKey, RowKey en Timestamp) die nodig zijn voor table storage. De partition key en row key vormen samen de primary key voor een en-

titeit in table storage, de timestamp kan worden gebruikt voor concurrency. Partition key kan gebruikt worden om data logisch in te delen. Windows Azure zal rekening houden met deze key om de data te verdelen over verschillende nodes in verschillende datacentra en entiteiten met dezelfde partition key waar mogelijk op dezelfde locatie op te slaan.
De data layer in TwitterMatic bestaat uit een class TimedTweetRepository, die op een uniforme manier zaken als Create, Read, Update en Delete aanbiedt aan mijn applicatie en intern gebruik maakt van Windows Azure Table Storage. Op die manier kan later eventueel terug overgestapt worden naar SQL Server zonder al te veel code te wijzigen. De TimedTweetRepository class ziet er zo uit:

```
public class TimedTweetRepository : ITimedTweetRepository
{
    protected static readonly string TABLENAME = "TimedTweet";
    protected CloudTableClient cloudTableClient;

    protected TableServiceContext GetDataServiceContext()
    {
        TableServiceContext svc = cloudTableClient.GetDataService
        Context();
        svc.IgnoreMissingProperties = true;

        return svc;
    }

    public TimedTweetRepository(CloudTableClient client)
    {
        cloudTableClient = client;
    }

    public List<TimedTweet> RetrieveAll(string screenName)
    {
        TableServiceContext svc = this.GetDataServiceContext();

        List<TimedTweet> result =
                svc.CreateQuery<TimedTweet>(TABLENAME)
                   .Where(t => t.ScreenName == screenName).
                   ToList();

        foreach (var item in result)
        {
            svc.Detach(item);
        }
        return result;
    }

    // ... meer methods ...
}
```

De naam van de tabel in Table Storage wordt gespecificeerd in een constante TABLENAME, die door de hele TimedTweetRepository class terugkomt. Verder is er ook een CloudTableClient, die door de applicatie wordt aangeleverd en de authenticatie- en verbindingsgegevens naarTable Storage bevat. De GetDataServiceContext() method creëert een nieuwe client voor Table Storage wanneer nodig. Deze client is een speciale versie van de client die ook in ADO.NET Data Services (voorheen Astoria) terugkomt. In de RetrieveAll method wordt een query uitgevoerd op Table Storage die alle geplande tweets voor een bepaalde gebruiker teruggeeft.
De CloudTableClient wordt in de meeste controllers van de ASP.NET MVC applicatie aangeleverd aan de TimedTweetRepository. Het initializeren hiervan gebeurt op basis van een waarde in de ServiceConfiguration.csdef.

```
CloudStorageAccount cloudStorageAccount =
    CloudStorageAccount.FromConfigurationSetting("StorageConnection");

CloudTableClient tableClient =
    cloudStorageAccount.CreateCloudTableClient();

tableClient.CreateTableIfNotExist("TimedTweet");

TimedTweetRepository repository = new TimedTweetRepository(tableClient);
```

De connection string 'StorageConnection' wordt gebruikt voor het inlezen van accountgegevens. Daarna wordt een CloudTableClient opgezet die door de TimedTweetRepository kan worden gebruikt.

## Laat een ander het werk opknappen: worker role

De worker role krijgt binnen TwitterMatic de taak om geplande tweets te overlopen en indien nodig effectief naar Twitter te posten. In eerdere versies van Windows Azure kon per solution slechts één worker role worden gestart. In de huidige versie kunnen dat er meer zijn. TwitterMatic heeft daardoor historisch slechts één worker role terwijl er toch twee taken moeten worden uitgevoerd, losstaand van elkaar: het controleren op tweets die gepost moeten worden en deze in een wachtrij plaatsen, en het daadwerkelijk versturen van een tweet naar Twitter. In TwitterMatic zijn deze twee taken ondergebracht in een eigen thread om zo toch twee taken simultaan te kunnen uitvoeren binnen een worker role.
Een Worker Role bestaat uit een class die van de base class RoleEntryPoint uit de Windows Azure SDK overerft. Het implementeren van een Worker Role kan door één of meerdere van de volgende methoden te overriden:

- OnStart(), die wordt aangeroepen bij het starten van een worker role. Op dit moment kunnen bepaalde initializaties of checks die nodig zijn om de worker role succesvol te starten worden uitgevoerd.
- Run(), die kan worden vergeleken met de Main() method in een Windows applicatie. Deze wordt uitgevoerd als de worker role gestart is.
- OnStop(), die wordt uitgevoerd bij het stoppen van een worker role. Hier kunnen bijvoorbeeld databaseconnecties worden gesloten.
- RoleEnvironmentChanging(), een event handler die logica bevat die moet worden uitgevoerd wanneer een wijziging in de configuratie is gedaan: moet de worker role opnieuw opstarten met deze nieuwe configuratie? Mag de worker role blijven draaien?

De worker role voor TwitterMatic wordt binnen de Run() method geïmplementeerd:

```
public class WorkerRole : RoleEntryPoint
{
    protected IConfigurationProvider configuration;
    protected ITimedTweetRepository repository;

    protected Thread enqueueingThread;
    protected Thread publishingThread;

    public override void Run()
    {
        Trace.WriteLine("Started TwitterMatic worker process.",
        "Information");

        enqueueingThread = new Thread(new ThreadStart(Enqueue-
        Updates));
        publishingThread = new Thread(new ThreadStart(Publish-
        Updates));

        enqueueingThread.Start();
        publishingThread.Start();
```

```
        enqueueingThread.Join();
        publishingThread.Join();

        Trace.WriteLine("Stopped TwitterMatic worker process.",
        "Information");
    }

    // ... meer code ...
}
```

Binnen de Run() method wordt de enqueueingThread en de publishingThread gestart. Beide hebben hun eigen taak en werken met de TimedTweetRepository en de Windows Azure Queue Service die de tweets bevat die moeten gepost worden.


De EnqueueUpdates method ziet er uit als volgt:

```
protected void EnqueueUpdates()
{
    CloudStorageAccount cloudStorageAccount = CloudStorageAccount.
    FromConfigurationSetting("StorageConnection");
    CloudQueueClient queueClient = cloudStorageAccount.Create-
    CloudQueueClient();
    CloudQueue updateQueue = queueClient.GetQueueReference("update
    queue");
    updateQueue.CreateIfNotExist();

    while (true)
    {
        List<TimedTweet> dueTweets =
            Repository.RetrieveDue(DateTime.Now.ToUniversalTime());

        if (dueTweets.Count > 0)
        {
            foreach (var tweet in dueTweets)
            {
                if (tweet.SendStatus != "Pending delivery")
                {
                    updateQueue.AddMessage(new CloudQueueMessage
                    (tweet.RowKey));
                    tweet.SendStatus = "Pending delivery";
                    Repository.Update(tweet);
                }
            }
        }
        else
        {
           Trace.WriteLine("[Enqueue] No due tweets.", "Information");
        }
        Thread.Sleep(120000);
    }
}
```

In dit voorbeeld wordt gebruik gemaakt van de queue service: een CloudQueueClient wordt geinitializeerd op basis van de connection string die ook eerder al werd gebruikt. Deze CloudQueueClient biedt een CloudQueue object aan op basis van de naam van een queue, 'updatequeue'. Via dit object kan de communicatie met de queue plaatsvinden, zoals het opvragen van berichten in de queue en het markeren van een bericht als 'voltooid'.

Bovenstaand codevoorbeeld controleert in Table Storage of er tweets zijn die moeten worden verzonden. Iedere tweet die verzonden moet worden, wordt in de queue geplaatst door middel van de AddMessage method. Dit bericht zal dan gedurende maximaal zeven dagen in de queue blijven tot het wordt opgepakt door de PublishUpdates thread.

PublishUpdates maakt gebruik van meer methods van de queue:

- PeekMessage() controleert of de queue al dan niet berichten bevat
- GetMessage() haalt een bericht op en zorgt dat het een opgegeven tijd verborgen wordt. Na deze tijd verschijnt het bericht weer in de queue. Op die manier zal het bericht blijven bestaan in de wachtrij, mocht de de worker role crashen.
- DeleteMessage() verwijdert het bericht effectief van de queue. Als deze method niet wordt aangeroepen na verwerking van een bericht, zal het bericht terug in de queue verschijnen.

## Deployen naar Windows Azure

TwitterMatic is nu klaar om gedeployed te worden. Dit kan zowel vanuit Visual Studio als vanuit een msbuild taak die bijvoorbeeld op een build server draait. In Visual Studio volstaat het om rechts op de cloud service te klikken en te kiezen voor 'Publish...'. Visual Studio zal de solution dan compileren en er een package van maken. Dit is een groot bestand met daarin alle vereiste assemblies om de applicatie op Windows Azure te kunnen deployen. Als de applicatie gepackaged is, kan deze via http://windows.azure.com worden gepubliceerd.


Via deze interface kan TwitterMatic naar een staging en een productieomgeving gedeployed worden. De ServiceConfiguration.cscfg kan ook nog worden aangepast als de applicatie is gepubliceerd.

## Een eigen domeinnaam met Windows Azure

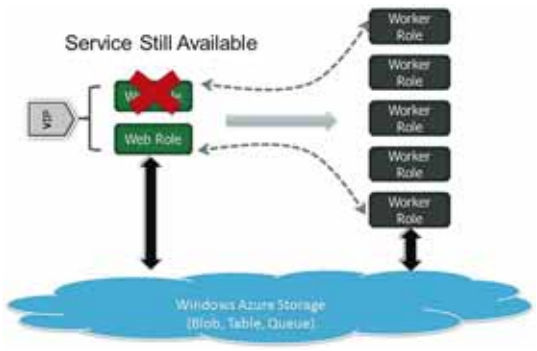
Standaard wordt een applicatie steeds gehost op het domein cloudapp.net. TwitterMatic is beschikbaar via http://twittermatic.cloudapp.net. Voor deze applicatie is echter ook de domeinnaam twittermatic.net geregistreerd. Deze kan worden gekoppeld met Windows Azure door in de DNS server van twittermatic.net een CNAME record aan te maken, een DNS verwijzing van twittermatic.net naar twittermatic.cloudapp.net. Windows Azure zal op basis van dit CNAME record de domeinnaam gebruiken om de applicatie te hosten. De DNS zone voor TwitterMatic ziet er als volgt uit:

```
twittermatic.net A 1 day  195.207.131.154
twittermatic.net MX 1 day 15 mail.twittermatic.net
twittermatic.net NS 1 day  ns1.eurobesthosting.eu
twittermatic.net NS 1 day  ns2.eurobesthosting.eu
www.twittermatic.net CNAME 1 day  twittermatic.cloudapp.net
```

Dit laatste record zorgt ervoor dat de domeinnaam wordt gekoppeld aan de applicatie die gehost is op Windows Azure.

## Queue vol? Meer workers!

Nu de applicatie gehost is op Windows Azure, wil ik nog enkele tips & tricks meegeven om de beschikbaarheid van de applicatie hoog te houden. Allereerst: zorg voor monitoring van de queue die gebruikt wordt voor het verzenden van tweets. Als deze steeds blijft vollopen is het aangeraden om een extra worker role op te starten, zodat meer berichten tegelijkertijd kunnen worden verwerkt.
Verder is het aan te raden om steeds meerdere instances van iedere role in een Windows Azure applicatie op te starten. Op die manier kan telkens één role probleemloos herstart worden, terwijl de applicatie beschikbaar blijft via de andere instance. Ook bij het upgraden van een applicatie is dit handig: Windows Azure verdeelt de aplicatie in (maximaal 10) zogenaamde upgrade domains en gaat per upgrade domain de instances stoppen, upgraden en weer online brengen. In het geval van twee instances betekent dat dus dat de applicatie ook tijdens een upgrade beschikbaar blijft op 1 instance.




## Conclusie

In dit artikel ben ik dieper ingegaan op de ontwikkeling van Twitter-Matic. Zelf ben ik erg enthousiast over cloud computing en het Windows Azure platform. Hopelijk zien we in de nabije toekomst meer en meer mensen gebruik maken van het Windows Azure platform en komen er nieuwe, innovatieve toepassingen die voor iedereen beschikbaar zijn.

### Links

http://windows.azure.com
http://www.twittermatic.net
http://twittermatic.codeplex.com
http://blog.maartenballiauw.be
http://twitter.com/maartenballiauw
http://www.microsoft.com/downloads/details.aspx?familyid=8D75D4F7-77A4-4ADF-BCE8-1B10608574BB&displaylang=en
http://code.msdn.microsoft.com/windowsazuresamples
http://www.microsoft.com/downloads/details.aspx?FamilyID=413E88F8-5966-4A83-B309-53B7B77EDF78&displaylang=en
http://linqtotwitter.codeplex.com
http://asp.net/mvc

**Maarten Balliauw,** is software engineer bij RealDolmen. Hij is Microsoft MVP.