Windows Azure AppFabric Caching

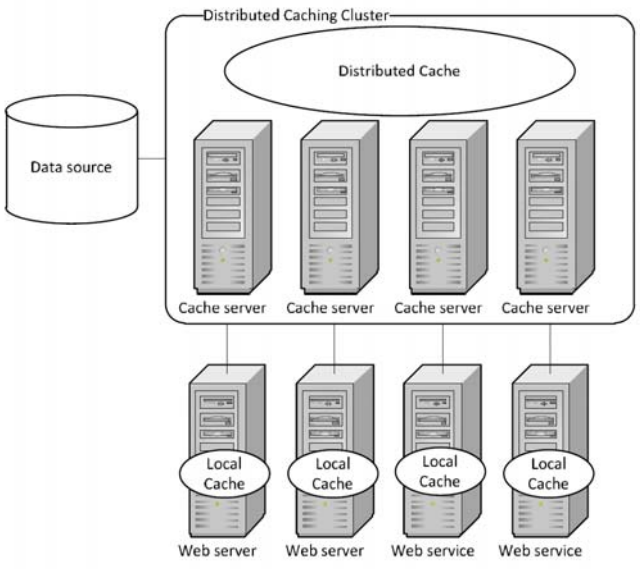
Hoe werkt het en wat zijn de voordelen?

Jonas Butt

Windows Azure AppFabric biedt service bus en access control diensten. Later dit jaar komen daar enkele diensten bij, waaronder een dienst voor distributed caching: Windows Azure AppFabric Caching. In het kort is dit de cloudvariant van de caching oplossing die je kent van Windows Server AppFabric. Cloudapplicaties kunnen hier gebruik van maken om hun prestaties te verbeteren en schaalbaarheid te vergroten.

De motivatie voor het toepassen van caching, en specifiek in-memory caching, is simpel en duidelijk: Caching is een manier om relatief trage en beperkt te schalen resources zoals een database of bestandssysteem te ontzien, met als doel de performance en schaalbaarheid van je applicatie te vergroten. Caching brengt additionele complexiteit met zich mee en dient dan ook pas te worden overwogen zodra hier aanleiding voor is.

Een voorbeeld hiervan is dat uit performancetesten blijkt dat de database de verwachte load niet aan kan. Een distributed cache biedt dan, in tegenstelling tot het schalen van bijvoorbeeld een database, een eenvoudige en goedkope manier om je applicatie te schalen. In de basale vorm van in-memory caching vindt het cachen van gegevens plaats in het geheugen van het proces waarbinnen een applicatie draait. Dit wordt local caching genoemd en het is de snelst mogelijke manier van caching, omdat de gegevens zich hierbij dicht bij de applicatielogica en de eindgebruiker bevinden.



FIGUUR 1: EEN EENVOUDIGE WEERGAVE VAN EEN DISTRIBUTED CACHE.

Bij .NET webapplicaties betekent dit dat geheugen van IIS processen gebruikt wordt voor het opslaan van objecten. Het probleem met local caching ontstaat zodra een webapplicatie wordt verspreid over meerdere servers en dus over meerdere processen. Iedere instantie van de applicatie gaat dan alle gegevens zelf ophalen uit het bronsysteem en lokaal cachen. Door dit individueel cachen kan consistentie in de gegevens in de cache niet eenvoudig worden gewaarborgd. Bovendien wordt het bronsysteem hiermee onnodig vaak geraadpleegd en is het geheugen dat lokaal beschikbaar is voor de cache beperkt. Er ontbreekt een manier om gegevens in de cache te delen en een consistente en coherente weergave van de gedeelde gegevens voor alle instanties van je applicatie. Precies dit probleem wordt opgelost met een distributed cache.

Een distributed cache kan worden gezien als de beheerder van cache objecten die verspreid zijn over het geheugen van een verzameling servers, dat een cache cluster wordt genoemd. De distributed cache verdeelt de gegevens over de servers volgens een bepaalde topologie, zoals gepartitioneerd of gerepliceerd. Bij de gepartitioneerde topologie wordt een enkel object op één locatie opgeslagen, wat gunstig is voor het geheugengebruik. Bij de gerepliceerde topologie worden alle gegevens op alle servers opgeslagen waarmee de beschikbaarheid van de gegevens wordt gegarandeerd, met als nadeel het aanvullende geheugengebruik. Een combinatie van gepartitioneerd en gerepliceerd is ook mogelijk.

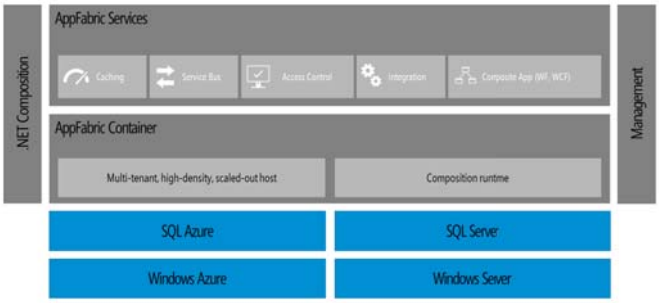
Een voordeel is dat doordat de distributed cache een cluster van servers vormt deze, in tegenstelling tot bijvoorbeeld een database, eenvoudig geschaald kan worden door meer servers toe te voegen. En doordat gegevens redundant worden opgeslagen kan het uitvallen van een node worden opgevangen en de beschikbaarheid van gegevens worden gewaarborgd. Binnen een cloudomgeving zoals Windows Azure is dit van groot belang, omdat de roles

daarin onverwacht kunnen herstarten of uitvallen. Een distributed cache wordt vrijwel altijd gecombineerd met de mogelijkheid om ook lokaal per webrole te cachen. Dit is immers de snelste vorm van caching, maar het houdt wel in dat de distributed cache de mogelijkheid moet bieden om de consistentie tussen de gegevens in het cache cluster en de gegevens in de local caches te garanderen. Dit kan op drie manieren worden gedaan:

- Per Request.
  Bij iedere aanvraag van een object wordt eerst gecontroleerd of de versie in de local cache gelijk is aan de versie in de distributed cache. Indien ze niet gelijk zijn, dan wordt eerst de local cache bijgewerkt. Figuur 12 toont hier een voorbeeld van. Deze controles hebben vanwege het additionele netwerkverkeer impact op de performance van de distributed cache.
- Notification Based.
  Hierbij pollen de local caches periodiek om te controleren of er nieuwere gegevens in de distributed cache aanwezig zijn en synchroniseren deze indien nodig. Ook notifications op basis van polling hebben impact op de performance, maar minder dan bij per request.
- Timeout Based.
  Hierbij krijgen de objecten in de local caches een levensduur door middel van een absolute of sliding expiration timeout. De performance hiervan is het beste ten opzichte van de andere twee manieren, maar je accepteert hierbij dat gegevens gedurende de timeout mogelijk niet recent zijn.

Windows Azure AppFabric Caching

Het ligt voor de hand om op een cloudplatform, waar dynamische schaalbaarheid van groot belang is, te beschikken over een distributed cache. Windows Azure AppFabric Caching biedt deze mogelijkheid op het Windows Azure platform.



FIGUUR 2: APPFABRIC OVERZICHT.

Net als andere Azure AppFabric onderdelen wordt Windows Azure AppFabric Caching aangeboden als dienst. Deze is met enig configuratiewerk te activeren met als grote voordeel dat het opzetten en beheren van een cache cluster volledig uit handen wordt genomen door Windows Azure. Een andere voordeel is dat een distributed cache in de cloud kosten kan besparen, doordat door caching toe te passen het gebruik van andere resources zoals storage transacties zal afnemen. Qua features is Windows Azure AppFabric Caching momenteel een subset van de on-premises variant Windows Server AppFabric Caching. Zo biedt het:

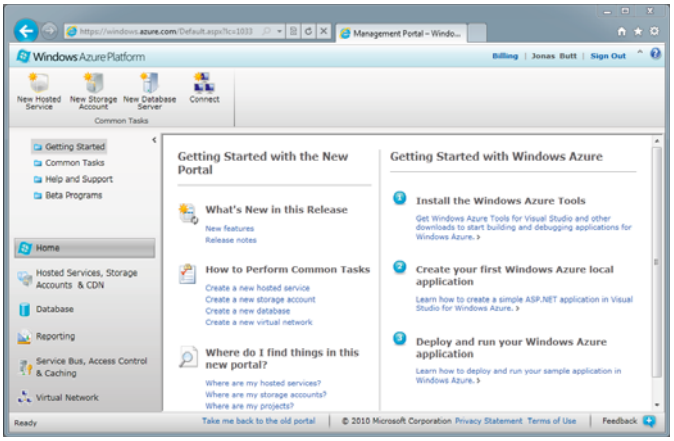
- Een API om .NET objecten te cachen
- Providers voor ASP.NET session state en output caching
- Beveiliging middels Access Control Service
- Geen limiet op grootte van objecten in de cache
- Een local cache
- Eenvoudige provisioning via een portal

Nog niet alle features uit Windows Server AppFabric Caching zijn beschikbaar, maar Microsoft streeft ernaar de features die zinvol binnen de cloud uit de on-premises variant beschikbaar te maken binnen Windows Azure.

Aan de slag

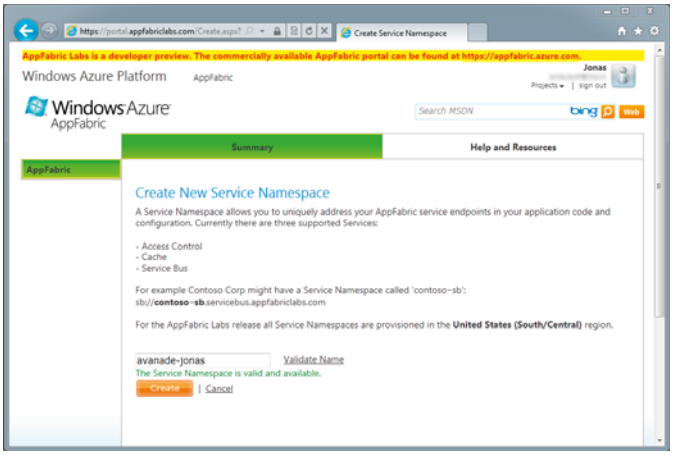
Een inherent voordeel van de diensten die Windows Azure AppFabric aanbiedt, zo ook van Caching, is dat het afnemen ervan weinig inspanning vereist. Voordat je begint dien je Visual Studio 2010 en de Windows Azure Tools for Microsoft Visual Studio 2010 geïnstalleerd te hebben. En natuurlijk heb je toegang tot Windows Azure nodig, die je als ontwikkelaar via Microsoft op diverse voordelige manieren kunt krijgen zoals bijvoorbeeld bij een MSDN subscription.

Vanuit de recentelijk vernieuwde Windows Azure portal vinden we onder de tab Service Bus, Access Control & Caching, de link naar de AppFabric portal. Om AppFabric Caching in te stellen maak je nu nog gebruik van de oude AppFabric portal. Sommige mogelijkheden zoals Caching zijn op dit moment alleen beschikbaar als Community Technology Preview (CTP) via een Labs versie op portal.appfabriclabs.com.



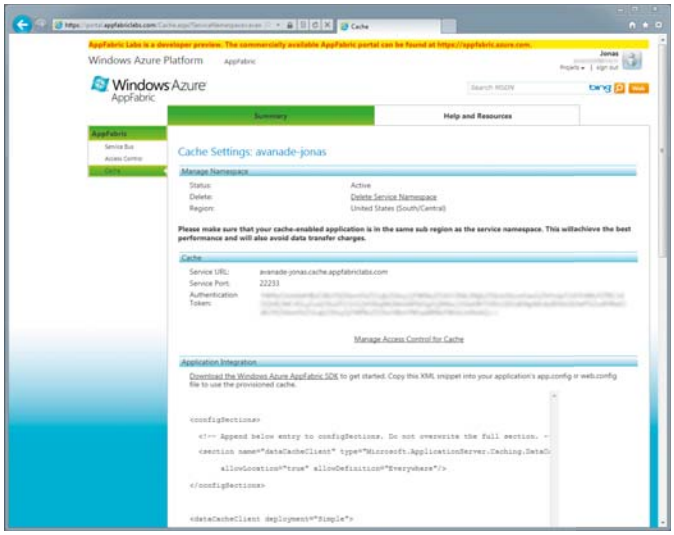
FIGUUR 3: DE VERNIEUWDE WINDOWS AZURE MANAGEMENT PORTAL.

Hier maak je eerst een project aan met een namespace die het adres van de endpoints bepaalt waarop de AppFabric diensten aangeroepen worden. Nadat de namespace geactiveerd is, kunnen



FIGUUR 4: EEN NAMESPACE VOOR DE APPFABRIC SERVICES AANMAKEN.

de individuele diensten worden beheerd. Op de beheerpagina voor de AppFabric Caching dienst vind je de service URL, service port en het authentication token. Rechten tot de caching service kunnen worden beheerd via de link naar Access Control. Ook vind je op de beheerpagina de link naar de Windows Azure AppFabric SDK en de configuratie die benodigd is om gebruik te maken van AppFabric Caching. De SDK bevat de assemblies die nodig zijn om binnen een Azure project de caching service uit te proberen. Hiermee is de caching dienst ingesteld en kunnen we de dienst in onze cloudapplicatie gebruiken.



FIGUUR 5: DE BEHEERPAGINA VAN DE APPFABRIC CACHING SERVICE.

Webapplicaties kunnen op twee manieren gebruik maken van de caching service zonder codewijziging. Door de configuratie van de beheerpagina in de web.config van je webapplicatie te plakken kan je eenvoudig zowel ASP.NET session state als ASP.NET output caching laten werken met de AppFabric Caching service. De configuratie bevat de service URL, service port en authentication token en de registratie van de session state store provider en de output cache provider.

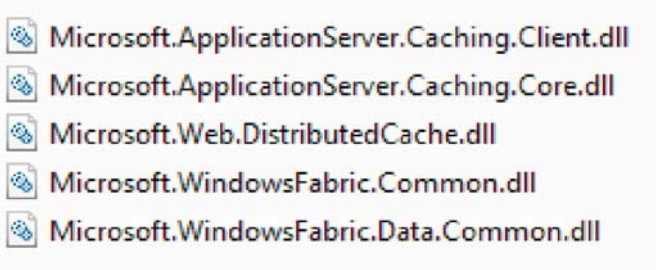
```
<configSections>
    <section name="dataCacheClient"
           type="Microsoft.ApplicationServer.Caching.DataCache
           ClientSection, Microsoft.ApplicationServer.Caching.Core"
                allowLocation="true" allowDefinition="Everywhere"/>
</configSections>

<dataCacheClient deployment="Simple">
    <hosts>
     <host name="avanade-jonas.cache.appfabriclabs.com"
       cachePort="22233" />
    </hosts>

    <securityProperties mode="Message">
        <messageSecurity
            authorizationInfo="[AcsToken]">
        </messageSecurity>
    </securityProperties>
</dataCacheClient>
```

FIGUUR 6: CACHE CLIENT CONFIGURATIE .

```
<system.web>

    <compilation debug="true" targetFramework="4.0" />

    <sessionState mode="Custom" customProvider="AppFabricCacheSess
ionStoreProvider">
        <providers>
            <add name="AppFabricCacheSessionStoreProvider"
                    type="Microsoft.Web.DistributedCache.Distri
butedCacheSessionStateStoreProvider, Microsoft.Web.Distributed
Cache"
                    cacheName="default" useBlobMode="false" />
        </providers>
    </sessionState>

    <caching>
        <outputCache defaultProvider="DistributedCache">
            <providers>
                <add name="DistributedCache"
                  type="Microsoft.Web.DistributedCache.
                   DistributedCacheOutputCacheProvider,
                   Microsoft.Web.DistributedCache"
                        cacheName="default" />
            </providers>
        </outputCache>
    </caching>

</system.web>
```

FIGUUR 7: CONFIGURATIE VAN EEN CUSTOM SESSION STATE STORE EN OUTPUT CACHE PROVIDERS.

Het enige dat nu dient te gebeuren is het toevoegen van references naar de benodigde assemblies. Deze zijn te vinden in de map waar de Azure AppFabric SDK geïnstalleerd is.



FIGUUR 8: WINDOWS AZURE APPFABRIC CACHING ASSEMBLIES.

Naast de providers biedt AppFabric Caching een API om de cache direct aan te spreken. Door het beperkte aantal klassen leer je er snel mee werken. Het startpunt is de DataCacheFactory die verantwoordelijk is voor het maken van een DataCache instantie op basis van configuratie. Deze configuratie kan uit de app.config of web.config komen, maar kan ook geheel in code worden gedefinieerd. Dit betekent dat je tijdens de uitvoering van je applicatie bijvoorbeeld de local cache aan of uit kan zetten. Aan de DataCacheFactory kan je een DataCache opvragen op basis van een naam, maar omdat named caches in de CTP niet ondersteund zijn, kan je alleen de default cache opvragen met de methode GetDefaultCache.

```
DataCacheFactoryConfiguration dataCacheFactoryConfiguration
= new DataCacheFactoryConfiguration();

dataCacheFactoryConfiguration.LocalCacheProperties =
    new DataCacheLocalCacheProperties(
        objectCount: 1000,
        defaultTimeout: TimeSpan.FromMinutes(10),
        invalidationPolicy: DataCacheLocalCacheInvalidationPolicy.
        TimeoutBased);

DataCacheFactory dataCacheFactory = new DataCacheFactory
(dataCacheFactoryConfiguration);
```

FIGUUR 10: HET INSCHAKELEN VAN DE LOCAL CACHE IN CODE.

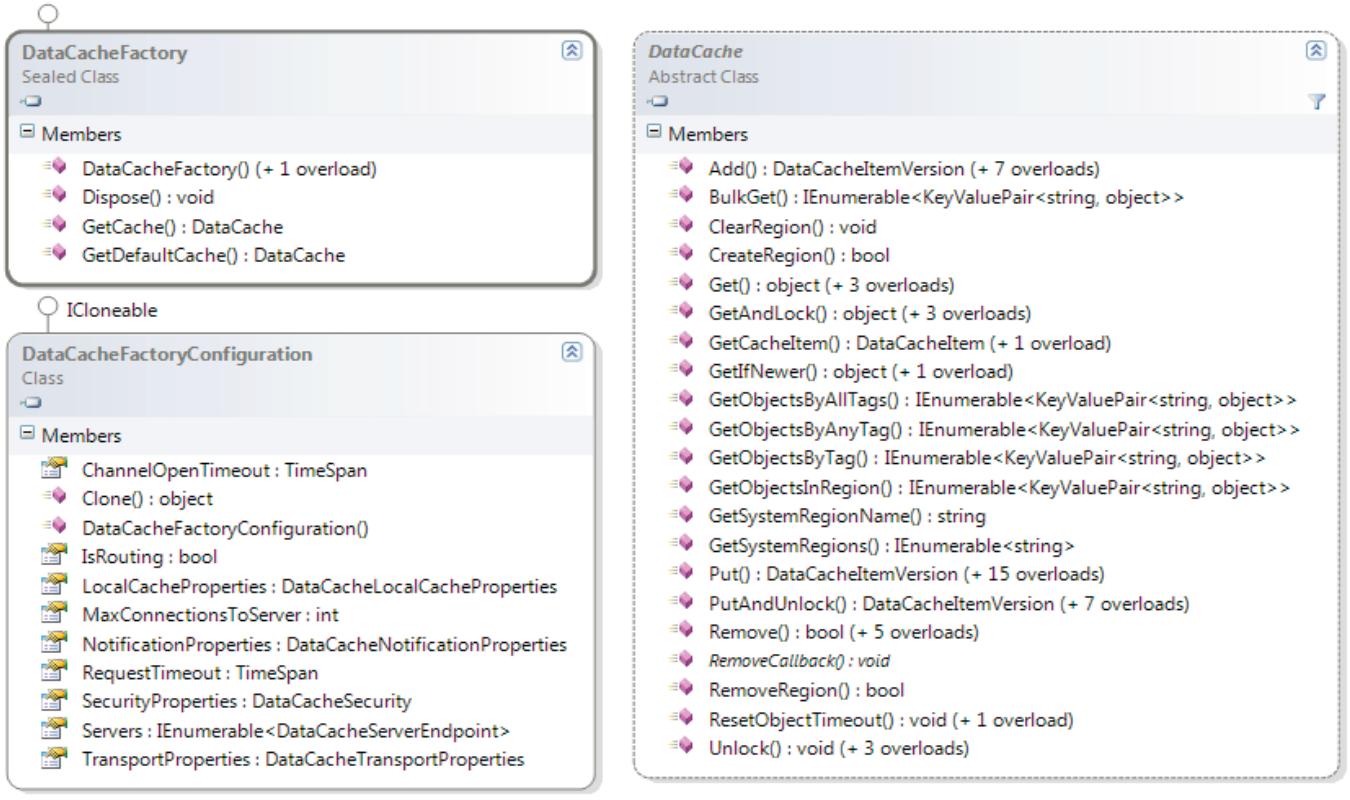
Met de DataCache kan je vervolgens eenvoudige tot geavanceerde caching scenario's uitwerken, omdat de API van deze klasse uitgebreid is. Het gebruik van de cache gebeurt volgens het zogenoemde cache-aside usage pattern, wat inhoudt dat je applicatie zelf de logica moet bevatten voor het controleren of gevraagde gegevens zich al in cache bevinden. Zijn de gegevens aanwezig in de cache, dan worden deze opgehaald en gebruikt. Staan de gegevens nog niet in de cache, dan worden ze opgehaald uit het bronsysteem en worden ze in de cache geplaatst. Figuur 11 toont een eenvoudige uitwerking hiervan. Hier wordt een SyndicationFeed object op basis van zijn key uit de cache gehaald via de indexer. Vervolgens wordt er gecontroleerd of dit object null is wat betekent dat de gegevens nog niet in de cache staan. Indien het opgehaalde SyndicationFeed object null is, dan wordt de feed opgehaald en wordt het resultaat hiervan in de cache geplaatst voorzien van een expiration timeout.

```
public class RssFeedsRepositoy
{
 private static readonly TimeSpan RssFeedChacheExpirationTimeOut
 = TimeSpan.FromMinutes(5);
 private const string RssFeedChacheKeyFormat = "RssFeed_{0}";

 private DataCache cache;

 public RssFeedsRepository()
 {
    DataCacheFactory dataCacheFactory = new DataCacheFactory();
    cache = dataCacheFactory.GetDefaultCache();
 }
 public IEnumerable<SyndicationItem> GetRssFeedItems (string
 rssFeedUrl)
 {
    string rssFeedCacheKey = String.Format (RssFeedCacheKeyFormat,
    rssFeedUrl);

    SyndicationFeed rssFeed = cache[rssFeedCacheKey] as
    SyndicationFeed;

    if (rssFeed == null)

    {
       XmlReader xmlreader = XmlReader.Create(rssFeedUrl);
       rssFeed = SyndicationFeed.Load(xmlreader)

       cache.Add(rssFeedCacheKey, rssFeed, RssFeedCacheExpiration-
       Timeout);
    }
    return rssFeed.Items;
    }
}
```

FIGUUR 11: EEN VOORBEELD VAN HET CACHEN EN OPHALEN VAN EEN OBJECT VOLGENS HET CACHE-ASIDE USAGE PATTERN.

Verder ondersteunt de DataCache het werken met regions, tags, locks en versies. Met regions kan je objecten in de cache groeperen om de performance te bevorderen. Tags en versies zijn nuttig voor het zoeken naar items in de cache. Locks kunnen ingezet worden om concurrency af te handelen. Daarnaast kan je notification callbacks instellen en in één keer meerdere objecten uit de cache opvragen. Een interessante methode van de DataCache is GetIfNewer. Wanneer je de local cache ingeschakeld hebt, kan je met deze methode ervoor zorgen je toch altijd de meeste recente versie van een gecached object ophaalt. Indien een nieuwere versie in de distributed cache bestaat, wordt deze opgehaald en in de local cache opgeslagen. Indien de versies gelijk zijn, dan wordt het object uit de local cache gebruikt. Hiermee combineer je de performancevoordelen van een local cache met het consistentievoordeel van de distributed cache.

```
/// <summary>
/// Ensures that the newest version of a cached object is returned
/// from either the local cache or the distrbuted cache.
/// </summary>
```



FIGUUR 9: DE DRIE BELANGRIJKSTE KLASSEN OM DE APPFABRIC CACHING SERVICE TE GEBRUIKEN.

```
public TCachedObjectType GetNewest<TCachedObjectType>(string key)
 : where TCachedObjectType : class
{
    DataCacheItemVersion version;

    // Gets cached object from local cache if it exists.

/ Otherwise gets cached object from distributed cache and adds it
 to local cache.
    object cachedObject = cache.Get(key, out version);

    // Gets cached object from distributed cached if it is newer
      than given version.
    // If newer it adds it to local cache.
    object possiblyNewerCachedObject = cache.GetIfNewer(key,
   ref version);

    if (possiblyNewerCachedObject != null)
    {
        // Cached object from distributed cache is newer than
           cached object from local cache.
        cachedObject = possiblyNewerCachedObject;
    }

    return cachedObject as TCachedObjectType;
}
```

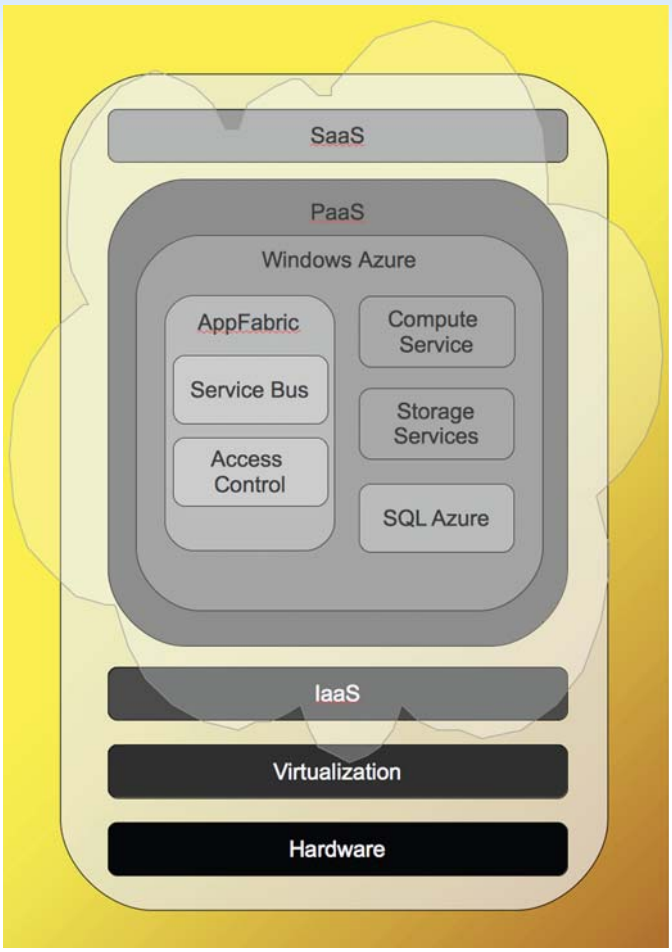
FIGUUR 12: DE LOCAL CACHE GEBRUIKEN EN CONTROLEREN OP DE NIEUWSTE VERSIE .

Alternatieven

Voor de ontwikkeling van on-premises .NET applicaties zijn er diverse mogelijkheden. Windows Server AppFabric Caching is al enige tijd beschikbaar als gratis toevoeging voor Windows Server 2008 en IIS 7. De codebase hiervan is gebruikt voor het ontwikkelen van Windows Azure AppFabric Caching en dat heeft als voordeel dat applicaties die al gebruik maken van de API van Windows Server AppFabric Caching zonder wijziging in de caching logica kunnen worden overgezet naar Windows Azure. NCache en memcached zijn twee andere distributed caching oplossingen voor .NET applicaties. NCache is een commercieel product en memcached is open source software. Laatstgenoemde, in combinatie met de open source beheertool CloudCache, was tot voor kort de enige oplossing voor distributed caching binnen Windows Azure. Het grote nadeel van memcached is dat je zelf verantwoordelijk bent voor het opzetten en beheren van een cache cluster. Daar bovenop komen nog de kosten voor het hosten hiervan binnen Windows Azure worker roles en het feit dat objecten in de cache beperkt zijn tot een grootte van 1MB. Tenslotte is de ASP.NET Cache ook binnen Windows Azure beschikbaar, maar omdat deze alleen local caching ondersteunt biedt het niet de voordelen van een distributed cache.

Tenslotte

Caching is een belangrijk onderdeel in de architectuur van moderne webapplicaties. Door in-memory caching toe te passen kan je een grote mate van schaalbaarheid van je applicatie bewerkstelligen, omdat in-memory caching sneller toegang biedt tot veelgebruikte data en eenvoudiger te schalen is dan bijvoorbeeld een database. Schaalbare webapplicaties worden in de praktijk vrijwel altijd gehost op een server farm met daarin meerdere web front ends. Door het gebruik van meerdere web front ends ontstaat de noodzaak voor één consistente toegang tot de cachinglaag. Dit wordt met een distributed cache bereikt en Windows Azure AppFabric Caching biedt deze mogelijkheid binnen Windows Azure. Het belangrijkste voordeel van deze dienst is dat het opzetten en beheren van een cache cluster volledig uit handen wordt genomen, waardoor je je als ontwikkelaar volledig kan richten op het optimaliseren van je applicatie door middel van caching. Momenteel is deze dienst nog in CTP vorm. Het is dus nog even wachten tot dat Microsoft belangrijke features zoals notifications heeft geïmplementeerd, het prijsmodel heeft bepaald en de dienst in productie heeft gezet. Naar verwachting komt Windows Azure AppFabric Caching in de eerste helft van 2011 beschikbaar. Tot die tijd kan je de dienst gratis uitproberen via de Windows Azure AppFabric Labs Portal en met behulp van de SDK.



DE PLAATS VAN DE APPFFABRIC IN DE CLOUD.

Links

- Windows Azure AppFabric Labs Portal
  https://portal.appfabriclabs.com
- Windows Azure AppFabric SDK V2.0 CTP
  http://www.microsoft.com/downloads/en/details.aspx?FamilyID=d89640fc-c552-446e-aead-b1e0d940f31b
- Windows Azure SDK
  http://www.microsoft.com/downloads/en/details.aspx?FamilyID=7a1089b6-4050-4307-86c4-9dadaa5ed018

Jonas Butt, (jonas.butt@avanade.com) is solution developer bij Avanade (www.avanade.com), een samenwerkingsverband tussen Microsoft en Accenture.