Best practices voor schaalbaarheid in Azure

Applicaties zijn niet zonder meer schaalbaar in de cloud

Kurt Claeys

Het bouwen van applicaties op het Azure platform laat toe dat deze zeer schaalbaar kunnen zijn. Het Azure platform ondersteunt een aantal mogelijkheden en technieken om applicaties te gaan schalen, al naar gelang het gebruik ervan. Dit zowel qua processing-capaciteit (= het aantal actieve processors, dat de applicatie ter beschikking heeft) als qua opslag (= het volume van data in een database of file storage).

Azure maakt niet zomaar automatisch van alle bestaande applicaties schaalbare applicaties. Het ondersteunt schaalbaarheid, maar de applicatie moet hiervoor open staan. Het is zo dat de applicatie qua architectuur een aantal patronen moet toepassen om deze schaalbaarheid effectief te kunnen bereiken. In dit artikel bespreken we enkele van deze patronen en zien we hoe deze praktisch in code kan worden geïmplementeerd.
Schalen wil zeggen dat door toevoeging van meer rekenkracht en opslagcapaciteit een applicatie op een nette manier de eventuele pieken in het gebruik ervan kan opvangen. Dit geldt voor zowel rekenkracht als opslagcapaciteit. We kennen in het algemeen twee vormen van schaalbaarheid : we spreken van scale-out als men meerdere instanties van dezelfde krachten (CPU, storage) kan activeren en de gebruikersrequests over al deze instanties kunnen worden verdeeld. We spreken van scale-up als men aan een bestaande hardware-eenheid meer kracht (een snellere CPU, meer diskspace) toekent, waarbij de load niet wordt gedistribueerd, maar alleen door de opgewaardeerde machine wordt verwerkt.
Azure compute gebruikt het scale-out principe. Je kunt eenvoudig extra instanties, die identiek zijn aan de eerste, activeren. Een loadbalancer in het Azure datacenter zal alle requests over deze instanties verdelen. De load balancer doet dit op basis van het round-robin principe : hier dient elke virtuele machine (= instantie) evenveel werk te doen omdat de load balancer de requests gewoon distribueert over het aantal instanties. Het spreekt dat de load balancer perfect op de hoogte is van het aantal instanties waarover deze de load kan verdelen. Bij het activeren van extra instanties zal de load balancer hiervan op de hoogte worden gebracht.
Het bouwen van schaalbare applicaties komt dus neer op het implementeren van proces en opslag design patterns, die het platform mogelijk maken om dezelfde prestatie te blijven bereiken bij zowel een hoge piek als bij normaal gebruik. Bij Azure spreekt men over elastische schaalbaarheid : de extra rekenkracht en opslagcapaciteit kan eenvoudig ook worden teruggedraaid. Dit brengt een uitgavenbesparing met zich mee. Het weghalen van capaciteit is niet realiseerbaar in het scale-up principe. Deze elastische schaalbaarheid is de belangrijkste meerwaarde van het Azure platform.
Om schaalbaarheid te bereiken zijn er twee werkwijzen :

- Zorgen dat er meer processingkracht en opslagcapaciteit klaar staan zodat je applicatie ze - indien nodig - kan aanwenden.
- Eerder pro-actief een deel van de nodige processingkracht verplaatsen naar plaatsen buiten de applicatie zodat ze niet de performance ervan kunnen beïnvloeden.

Schaalbaarheid creëren in een applicatie draait om het gebruiken van een combinatie van deze twee opties. Het gaat om het toepassen van een aantal technieken, zeg maar design patterns. Uiteraard is schaalbaarheid alleen effectief uitvoerbaar als het platform dit ook ondersteunt, als het platform deze patronen kan herkennen en kan aanwenden om hierop te gaan schalen. Dit is nu net waar Azure in uitblinkt: als de solution architect deze patronen opzet kan Azure zeer goed in beide richtingen schalen.
De volgende technieken zijn daarbij van belang:

- Scale-out naar meerdere instanties op basis van performance metrics.
- Partitionering van data.
- Het opzetten van een asynchrone architectuur op basis van queues.
- Gebruik van het content delivery network.
- Gebruik van federated identity.
- Caching van bepaalde stukken data.

In dit artikel bespreek ik de eerste twee, in een volgende artikel komen de andere aan bod.

Scale-out naar meerdere instanties

In Azure bevinden applicaties zich in virtuele machines die achter een loadbalancer aanwezig zijn en de requests gaan uitvoeren. Een applicatie kan zich dus in meerdere virtuele machines gaan bevinden. De keuze van het aantal virtuele machines ligt vast in de serviceconfiguratie (zie de figuur).

EEN VOORBEELD VAN EEN SERVICECONFIGURATIE.

Om te gaan schalen moet je dus deze configuratiefile aanpassen. Dit kan handmatig via de Azure management portal (voor actieve deployments), via Visual Studio (bij een nieuwe deployment) en via de Management API. Deze API is een REST API die remote management mogelijk maakt van de op Azure aanwezige applicaties. Na aanpassing van deze configuratie zal de fabric controller in het datacenter het nodige doen: indien je in de nieuwe configuratie meer instanties vraagt dan in de huidige zal de fabric controller een extra virtuele machine gaan opstarten die dan ook het werk achter de loadbalancer uitvoert. De loadbalancer zal te weten komen dat er extra kracht is en die nieuwe virtuele machine meenemen in het lijstje van machines waarover hij het werk zal verdelen. Mocht je in de nieuwe configuratie minder instanties opgeven als in de huidige versie worden de nodige machines stilgelegd. Dit is elastic scalability!

Vraag is nu uiteraard : wanneer ga je deze configuratie aanpassen. Je zou dit handmatig kunnen doen of door een scheduling mechanisme op bepaalde tijdstippen. Maar het is vaak de bedoeling om dit te doen op basis van de benodigde prestaties. Azure ondersteunt het werken met een Diagnostics API waarbij men kan opgeven welke diagnostics counters moeten worden gemonitord. Deze counters zijn de counters die we kennen van een klassieke windows server omgeving. Denk aan het gebruikspercentage van de processor, de vrije ruimte op de schijf in de virtuale machines, het aantal open IP connecties, enz.
De data van deze counters zal door Azure naar een table op een Azure storage account automatisch worden gekopieerd, waarbij de frequentie van deze kopie via code kan worden bepaald.
Het bepalen van deze diagnostics configuratie zie je in onderstaand codefragment.

```
DiagnosticMonitorConfiguration diagnosticMonitorConfiguration
= DiagnosticMonitor.GetDefaultInitialConfiguration();
diagnosticMonitorConfiguration.PerformanceCounters.DataSources.
Add(
    new PerformanceCounterConfiguration()
    {
        CounterSpecifier = @"\Processor(_Total)\% Processor Time",
        SampleRate = new TimeSpan(0, 0, 10)
    });
diagnosticMonitorConfiguration.PerformanceCounters.DataSources.
Add(
    new PerformanceCounterConfiguration()
    {
        CounterSpecifier = @"\LogicalDisk(_Total)\Free Megabytes",
        SampleRate = new TimeSpan(0, 0, 10)
    });
diagnosticMonitorConfiguration.PerformanceCounters.
ScheduledTransferPeriod = TimeSpan.FromMinutes(1);
DiagnosticMonitor.Start("DiagnosticsConnectionString",
diagnosticMonitorConfiguration);
```

CONFIGURATIE VAN DE DIAGNOSTICSMONITOR

Hierbij specificeren we dat we interesse hebben in de twee vernoemde counters, dat we hiervan elke 10 seconden de waarde willen bepalen en dat we deze metrics data elke minuut naar de storage account willen transfereren. Interessant om weten is dat we deze code ofwel in de applicatie in de startup method kunnen plaatsen ofwel kunnen uitvoeren vanuit een andere applicatie. Het is dus mogelijk om applicaties die reeds actief zijn en opgestart zijn zonder deze configuratie toch remote te gaan beïnvloeden om alsnog de counters te activeren en de data ervan te laten doorsturen naar storage. Dit laat een scenario toe waarbij je slechts tijdelijk enkele counters wil zien zonder dat je daarvoor de applicatie moet herstarten.
Volgende stap is het interpreteren van de countergegevens. Een manier zou kunnen zijn : het gemiddelde maken van de CPU utilization over een bepaalde periode. Als je dan bijvoorbeeld vaststelt dat gedurende geruime tijd deze utilization boven de 80% zit, dan is het waarschijnlijk tijd om een extra instantie te activeren.
De countergegevens tonen je alle details die je nodig hebt om deze beslissing te nemen : de structuur van data (zie figuur hieronder) bevat het tijdstip, de naam van de role, de instantie, de naam van de counter en zijn waarde.

Timestamp	Role	RoleInstance	CounterName	CounterValue
2010-12-07 22:18:34.427	WebRole1	WebRole1_IN_0	\Processor(_Total)\% Processor Time	0,935368
2010-12-07 22:18:34.427	WebRole1	WebRole1_IN_0	\LogicalDisk(_Total)\Free Megabytes	232197
2010-12-07 22:18:34.427	WebRole1	WebRole1_IN_0	\TCPv4\Connections Active	3
2010-12-07 22:18:34.427	WebRole1	WebRole1_IN_0	\Processor(_Total)\% Processor Time	1,093117
2010-12-07 22:18:34.427	WebRole1	WebRole1_IN_0	\LogicalDisk(_Total)\Free Megabytes	232197
2010-12-07 22:18:34.427	WebRole1	WebRole1_IN_0	\TCPv4\Connections Active	3
2010-12-07 22:18:34.427	WebRole1	WebRole1_IN_0	\Processor(_Total)\% Processor Time	1,405619
2010-12-07 22:18:34.427	WebRole1	WebRole1_IN_0	\LogicalDisk(_Total)\Free Megabytes	232197
2010-12-07 22:18:34.427	WebRole1	WebRole1_IN_0	\TCPv4\Connections Active	3
2010-12-07 22:18:34.060	WebRole1	WebRole1_IN_1	\Processor(_Total)\% Processor Time	0,936866
2010-12-07 22:18:34.063	WebRole1	WebRole1_IN_1	\LogicalDisk(_Total)\Free Megabytes	232197
2010-12-07 22:18:34.063	WebRole1	WebRole1_IN_1	\TCPv4\Connections Active	3
2010-12-07 22:18:34.063	WebRole1	WebRole1_IN_1	\Processor(_Total)\% Processor Time	1,56187
2010-12-07 22:18:34.063	WebRole1	WebRole1_IN_1	\LogicalDisk(_Total)\Free Megabytes	232197
2010-12-07 22:18:34.063	WebRole1	WebRole1_IN_1	\TCPv4\Connections Active	3
2010-12-07 22:18:34.063	WebRole1	WebRole1_IN_1	\Processor(_Total)\% Processor Time	1,249368
2010-12-07 22:18:34.063	WebRole1	WebRole1_IN_1	\LogicalDisk(_Total)\Free Megabytes	232197
2010-12-07 22:18:34.063	WebRole1	WebRole1_IN_1	\TCPv4\Connections Active	3
2010-12-07 22:20:07.613	WebRole1	WebRole1_IN_0	\Processor(_Total)\% Processor Time	0
2010-12-07 22:20:07.613	WebRole1	WebRole1_IN_0	\LogicalDisk(_Total)\Free Megabytes	232197
2010-12-07 22:20:07.613	WebRole1	WebRole1_IN_0	\TCPv4\Connections Active	3
2010-12-07 22:20:07.613	WebRole1	WebRole1_IN_0	\Processor(_Total)\% Processor Time	0
2010-12-07 22:20:07.613	WebRole1	WebRole1_IN_0	\LogicalDisk(_Total)\Free Megabytes	232197

DE GEGEVENS AFKOMSTIG VAN DE DIAGNOSTICSMONITOR.

Daar deze countergegevens zich op een storage account in de cloud bevinden is het aan te raden om deze naar een lokale SQL server database te downloaden. Op een SQL server heb je immers heel wat meer query mogelijkheden dan op de table storage. Ook is het kostenefficiënter om het interpreteren van de data lokaal te doen. Het downloaden van de data kan gebeuren door het aanspreken van de storageclient library.
Na het interpreteren van de metrics kun je dus beslissen om extra instanties te activeren of het aantal instanties te verminderen. Dit kan door het aanpassen van de configuratiefile wat door middel van de management API kan gebeuren. Deze management API is een REST style API maar er is een vriendelijkere .NET library hiervoor te vinden op code.msdn.microsoft.com/windowsazure-samples
Het volgende codefragment toont hoe dit in zijn werk gaat. Je plaatst de nieuwe configuratie met het aangepaste aantal instanties in een string en geeft die samen met de naam van je service en een aanduiding of het om de productie- of staging slot gaat aan de ChangeConfigurationBySlot methode.

```
newConfig =
 @"<?xml version=""1.0""?>
    <ServiceConfiguration serviceName=""WebRole1""
  xmlns=""http://schemas.microsoft.com/ServiceHosting/2008/10/
```

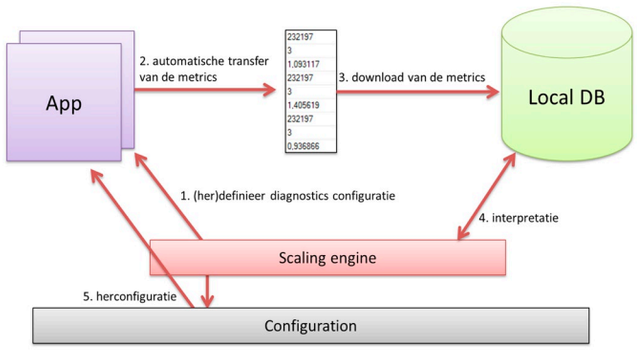
```
ServiceConfiguration""       osFamily=""1"" osVersion=""*"">
     <Role name=""WebRole1"">
       <Instances count=""4"" />
     </Role>
    </ServiceConfiguration>";

ChangeConfigurationBySlot(channel, subscriptionID, serviceName,
deployment Slot, newConfig);
```

AANPASSEN VAN DE CONFIGURATIE.

Scaling engine

Om de performance te monitoren en de nodige scaling uit te voeren zou je een scalingengine kunnen bouwen die deze besproken technieken toepast. De flow van deze engine zou er als volgt kunnen uitzien.

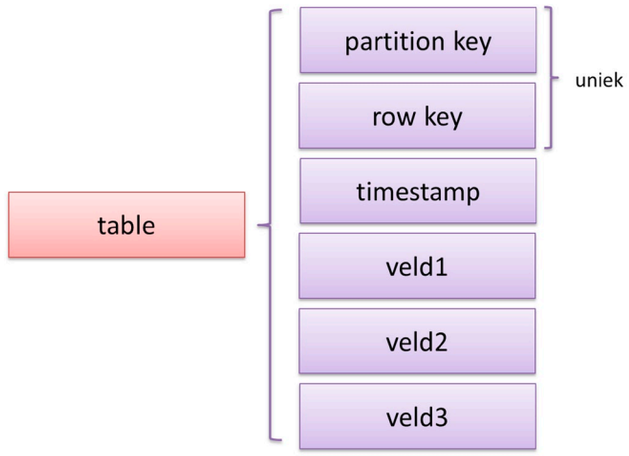


DE FLOW VAN EEN SCALING ENGINE.

1. De engine zal de diagnostics configuratie van applicaties gaan instellen en aanpassen.
2. Azure doet automatisch een transfer van de metrics naar storage.
3. De metrics worden gedownload naar een lokale database.
4. De engine zal deze metrics interpreteren en een beslissing nemen.
5. De configuratie zal aangepast worden.

Partitionering van data

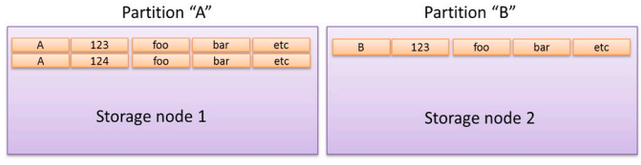
Om grote hoeveelheden data snel te kunnen benaderen kun je de data partitioneren. Hierdoor kan de gebruikte opslagtechnologie (Azure Storage in dit geval) een optimalisatie doen naar het bewaren van de data om een hoge verwerkingssnelheid te blijven behouden bij een groei van het aantal records. Partitionering in Azure storage doe je aan de hand van de inhoud van de partitionkey. Bij elke recordstructuur in Azure storage is er steeds een partitionkey en een rowkey aanwezig. De combinatie van deze twee keys vormt de unieke sleutel van het datarecord.



DE STRUCTUUR VAN EEN TABLE IN AZURE STORAGE.

Azure zal de records met een bepaalde waarde in de partitionkey eventueel gaan afzonderen van records met een andere inhoud in het partitionkey veld. Dit kan zelfs zo ver gaan dat deze verschillende subsets van data die je door het toepassen van partitionering creëert op verschillende storage nodes in het datacenter zullen worden geplaatst. Zo kunnen queries zeer snel worden uitgevoerd op voorwaarde dat je de partitionkey kent en gebruikt in de query. Azure merkt ook op welke partities er meest aangesproken worden vanuit de applicatie. Dit zijn de 'hot' partities. Azure zal zelf deze 'hot' partities qua opslag zodanig organiseren dat er steeds een goede performance op deze data behouden blijft.

Partition key	Row key	Data		
A	123	foo	bar	etc
A	124	foo	bar	etc
B	123	foo	bar	etc



PARTITIONERING VAN DATA.

Tot zover de eerste twee 'Patterns for scalability'. Naast deze zijn er tal van andere technieken die door ze toe te passen je applicaties schaalbaar kunnen maken. Deze hou ik voor een volgend artikel dat je op mijn blog (http://blogs.msdn.com/b/kclaeys) zal kunnen lezen.

Kurt Claeys, werkt als Technology Solution Professional rond Windows Azure voor Microsoft in het incubation team en heeft een achtergrond als .NET solution architect.