Applicaties bouwen met Azure AppFabric

Service Bus uit Azure AppFabric voorkomt problemen

Dennis van der Stelt

Tijdens de PDC van 2009, heeft Microsoft een nieuw platform gepresenteerd onder de naam 'AppFabric'. Momenteel biedt dit platform functionaliteit voor bi-directionele communicatie tussen applicaties en federated claims-based authenticatie. Beide zijn zeer belangrijk voor toepassingen die de migratie naar Windows Azure maken, maar ook voor on-premise applicaties. Dit artikel bespreekt de mogelijkheden en toepassingen van de Windows Azure AppFabric.

Microsoft maakt het met Windows Azure en haar integratie met het .NET Framework eenvoudig voor ontwikkelaars in te stappen in de wereld van cloud based computing. Daarover valt in dit .NET Magazine genoeg te lezen. Niet elke toepassing is echter geschikt voor cloud computing of mag simpelweg niet op die manier actief zijn vanwege wat voor oorzaak dan ook. Resultaat is dat er binnen je architectuur applicaties zowel in als buiten de cloud zullen opereren. Dit levert meteen bijzondere uitdagingen op met betrekking tot connectivity and security.

Stel je voor dat je een systeem hebt gebouwd welke moet communiceren met een applicatie of service in de cloud. De communicatie verloopt over de grenzen van het bedrijfsnetwerk. Dan kom je voor de uitdaging te staan of de corporate firewall die connecties wel doorlaat. De firewall is er ter bescherming voor aanvallen van buitenaf. Daarnaast kan Network Address Translation (NAT) voor een uitdaging zorgen, vanwege het feit dat je van buitenaf niet direct benaderbaar bent.

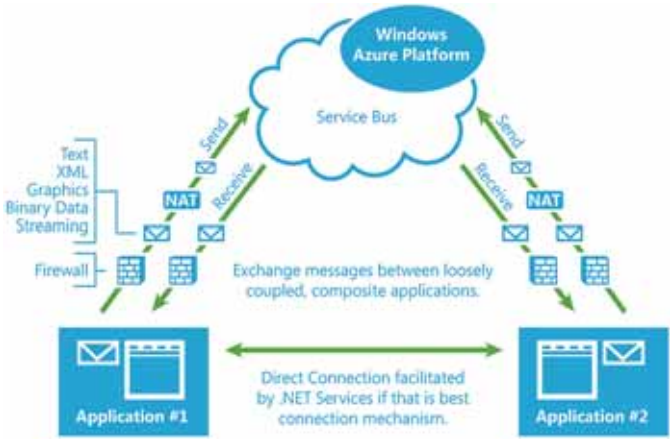
Een manier is om een poort te openen in de firewall. Veel thuisgebruikers maken daar gebruik van om met uPnP tijdelijk een directe toegang naar een desktop te realiseren, of permanent met port-forwarding. Dit levert echter risico's op ten opzichte van de beveiliging van het netwerk. Een andere manier is om een VPN op te zetten. Als het ware een soort tunnel waar anderen niet tussen kunnen komen. Configuratie is echter niet eenvoudig en flexibel en vaak is deze optie zelfs onmogelijk.

AppFabric Service Bus

De zojuist beschreven uitdagingen hoeven geen probleem meer te vormen als je gebruik maakt van de Service Bus uit de Windows Azure AppFabric. De Service Bus wordt door Microsoft gehost op hun eigen Windows Azure platform en faciliteert de verbinding tussen verschillende systemen. Hoewel de techniek erachter zeer complex is, is de werking ervan eigenlijk vrij eenvoudig. Als voorbeeld nemen we een service die toegang geeft tot externe partijen om gegevens op te halen.

1. Zodra de service gestart wordt, maakt deze zichzelf bekend bij de Service Bus door er een verbinding mee op te zetten.
2. De client wil communicatie opzetten met onze service, gebruikt echter het endpoint (url) van onze service in de ServiceBus.
3. De Service Bus dient nu als mediator tussen onze service en de client.
4. Elke vraag van de client applicatie naar de service, wordt nu afgehandeld door de Service Bus.

Je kunt de Service Bus zien als relay of mediator service in de cloud tussen je client en de service. Je kunt het vergelijken met een http proxy, die je instelt in je browser, waarna al het verkeer via deze proxy loopt. Het verschil met de Service Bus is echter, dat nu de service de verbinding initieel al heeft opgezet. Hierdoor ligt er al een verbinding die we kunnen gebruiken en waarover we zonder problemen terug kunnen communiceren. De Service Bus zal elk bericht forwarden naar de service over de verbinding die de service zelf heeft opgezet. Waar we normaal gesproken de verbinding enkel vanuit de kant van de client opzetten en tegen uitda-



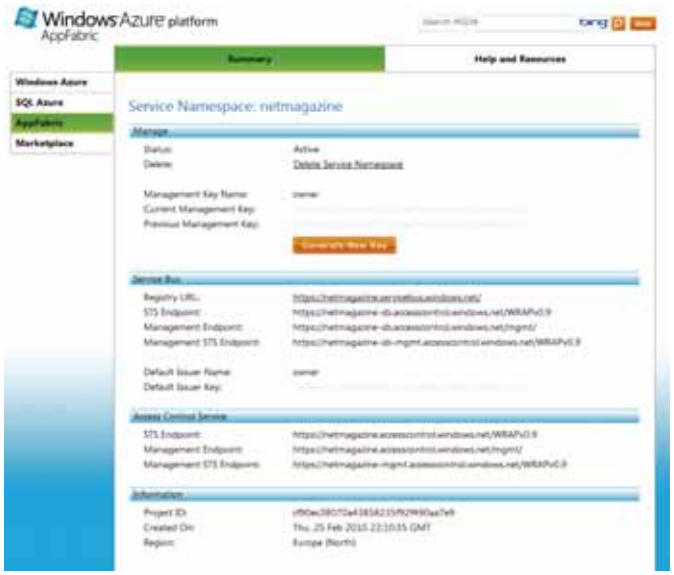
AFBEELDING 1: SERVICE BUS COMMUNICATIE KANALEN

gingen aanlopen zoals we die zojuist beschreven hebben.
Met de Service Bus is het mogelijk om:

- Services te vinden via een url van de Service Bus, ongeacht de locatie van de service.
- One-way messaging te gebruiken tussen zender en ontvanger met ondersteuning van uni-cast en multi-case datagram distribution.
- Full-duplex connecties op te zetten tussen zender en ontvanger met bi-directionele communicatie.
- Full-duplex connecties op te zetten op basis van peer-to-peer met 'Direct Connect'. Hierdoor is het mogelijk een directe verbinding op te zetten ongeacht of beide endpoints zich achter NAT bevinden.
- Publish/Subscribe te ondersteunen met meerdere publishers en subscribers.
- Ondersteuning te bieden voor HTTP en REST toegang voor niet .NET platformen.

Opzetten van de service

Voordat je daadwerkelijk begint met het bouwen van je systeem, dien je eerst in de Windows Azure portal een AppFabric project te creëren. Sinds kort kun je een introduction subscription nemen op Windows Azure, waardoor je een gelimiteerd aantal transacties, werkuren en storage tot je beschikking hebt. We krijgen hierbij twee Service Bus connecties, wat voor ons voldoende is om te testen. Heb je eenmaal een project aangemaakt, zul je een Service Namespace aan moeten maken.



AFBEELDING 2: HET AANMAKEN VAN EEN NAMESPACE.

In afbeelding 2 kun je zien dat ik de namespace https://netmagazine.servicebus.windows.net/ heb aangemaakt, waarvan alleen het netmagazine zelf te kiezen is. Dit is het unieke adres waarop straks de Service Bus zijn relay werk doet en berichten forward naar jouw service. Deze namespace, de management key name en de management key ('the secret'), dien je te gebruiken om de service aan te roepen.
Nu moeten we een contract samenstellen en omdat de Service Bus gebouwd is op WCF, is het ook eenvoudig om zelf WCF services te bouwen die van de Service Bus gebruik maken. Voor dit voorbeeld maken we een service die een emailadres valideert tegen een reguliere expressie. In codevoorbeeld 1 zie je hoe ons contract eruit ziet. In codevoorbeeld 2 staat de implementatie van onze service, waarin we de reguliere expressie voor het gemak even hebben weggelaten.

```
[ServiceContract(Name = "IEmailValidator", Namespace = "http://
schemas.tellus.com/emailcheck/2010/03/")]
public interface IEmailValidatorContract
{
  [OperationContract]
  bool ValidateEmail(string text);
}
```

CODEVOORBEELD 1

```
[ServiceBehavior(Name = "EmailValidatorService", Namespace =
"http://schemas.tellus.com/emailcheck/2010/03/")]
public class EmailValidatorService : IEmailValidatorContract
{
  public bool ValidateEmail(string emailAddress)
  {
    Console.WriteLine("Validating: {0}", emailAddress);

    string pattern = @"...";
    return Regex.IsMatch(emailAddress, pattern);

  }
}
```

CODEVOORBEELD 2

Tot op dit moment is er nog niets veranderd ten opzichte van standaard WCF services. In codevoorbeeld 3 zie je een heel eenvoudige en bijna standaard configuratie voor WCF services. De binding is echter nieuw, dit is de NetTcpRelayBinding. Deze binding heeft kennis van de Service Bus.

```
<system.serviceModel>
  <services>
    <service name="Microsoft.ServiceBus.Samples.EmailValidator-
    Service">
      <endpoint binding="netTcpRelayBinding" contract="Microsoft.
      ServiceBus.Samples.IEmailValidatorContract" />
    </service>
  </services>
</system.serviceModel>
```

CODEVOORBEELD 3

Nu kunnen we de service lokaal hosten in een console applicatie. Dit is nog niet mogelijk in IIS omdat de service dan niet automatisch gestart wordt. Dit gebeurt pas bij de eerste aanroep. Onze service moet echter initieel gestart worden en zichzelf bekend maken bij de Service Bus. Dublin zou hier een oplossing zijn, later kom ik hier nog kort op terug. Let bij je console applicatie wel op dat je niet het .NET Client Profile gebruikt, want die begrijpt niets van de Microsoft.ServiceBus assembly. Voeg deze assembly ook toe aan je project.
Ook het hosten van de service gebeurt grotendeels zoals je gewend bent. Dit kun je zien in codevoorbeeld 4. Breng de ServiceHost in de lucht met het juiste contract en het adres van je Service Bus namespace. Het adres wordt in dit voorbeeld in code opgemaakt op regel 3, met behulp van een speciale AppFabric methode. In regel 7 t/m 10 kun je zien dat we een endpoint behavior optuigen om onze credentials mee te geven, zodat de Service Bus ons accepteert. Deze voegen we in regel 14 t/m 16 toe aan elk endpoint dat al uit de applicatie configuratie is gehaald en geïnitieerd. Op de laatste regel openen we de host. Er staan nog twee regels code in die betrekking hebben op het transport channel en de discoverability van je service, waar we verder niet op in zullen gaan. Dit is een eenvoudig voorbeeld en alles dat nodig is om een service te hosten en de Service Bus als relay te gebruiken.

```
ServiceBusEnvironment.SystemConnectivity.Mode = ConnectivityMode.
AutoDetect;

Uri address = ServiceBusEnvironment.CreateServiceUri("sb", "net-
magazine", "EmailValidatorService");

ServiceHost host = new ServiceHost(typeof(EmailValidatorService),
address);

TransportClientEndpointBehavior sharedSecretServiceBusCredential =
new TransportClientEndpointBehavior();
sharedSecretServiceBusCredential.CredentialType = TransportClient
CredentialType.SharedSecret;
sharedSecretServiceBusCredential.Credentials.SharedSecret.Issuer
Name = "naam_van_issuer";
sharedSecretServiceBusCredential.Credentials.SharedSecret.Issuer-
Secret = "hele_lang_string_die_geheim_is";

IEndpointBehavior serviceRegistrySettings = new ServiceRegistry-
Settings(DiscoveryType.Public);

foreach (ServiceEndpoint endpoint in host.Description.Endpoints) {
  endpoint.Behaviors.Add(serviceRegistrySettings);
  endpoint.Behaviors.Add(sharedSecretServiceBusCredential);
}

host.Open();
```

CODEVOORBEELD 4

De client

Ook bij het bouwen van een client gebruiken we een console applicatie en doen we weinig anders meer dan wanneer we normaal met een WCF service communiceren. In codevoorbeeld 5 is te zien hoe we, via de Service Bus, met onze service communiceren.

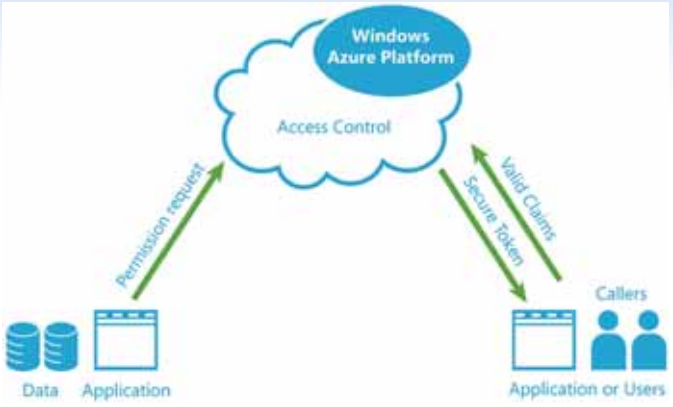
```
ChannelFactory<IEmailValidatorChannel> channelFactory =
  new ChannelFactory<IEmailValidatorChannel>("RelayEndpoint",
  new EndpointAddress(serviceUri));

channelFactory.Endpoint.Behaviors.Add(sharedSecretServiceBus-
Credential);

IEmailValidatorChannel channel = channelFactory.CreateChannel();
channel.Open();

bool result = channel.ValidateEmail("dvdstelt@gmail.com");
```

CODEVOORBEELD 5

Gemakshalve zijn het contract (de interface), het opmaken van het adres en het EndpointBehavior weggelaten aangezien dit hetzelfde is als bij de service. Hierna wordt een ChannelFactory gebruikt om een channel naar de Service Bus te creëren, waarna deze wordt geopend en er communicatie kan plaatsvinden. Zoals je ziet is er ook nu weinig tot geen verschil ten opzichte van normaal WCF gebruik.

AppFabric Access Control

We hebben gezien hoe we met behulp van de Service Bus kunnen communiceren over een complexe infrastructuur. Stel je voor dat er externe partijen zijn die van jouw systeem gebruik willen maken? We kunnen opnieuw onze eigen identity store opzetten en elke klant een account geven. Misschien zelfs meerdere accounts per klant met verschillende rollen. Een meer flexibele en schaalbare oplossing is claims based access control. Hiermee worden zogeheten claims gebruikt, waarmee jij als het ware kunt bewijzen dat je rechten hebt. AppFabric Access Control biedt de mogelijkheid om inkomende claims om te zetten naar nieuwe uitgaande claims die jouw systeem kan gebruiken.



AFBEELDING 3: APPFABRIC ACCES CONTROL.

In afbeelding 3 kun je zien hoe dit in zijn werk gaat. Linksonder zie je jouw systeem, rechtsonder zie je de derde partij. Jouw systeem (de applicatie of server) heeft een relatie met een specifieke Access Control namespace, zoals we die ook bij de Service Bus zagen. Zodra een derde partij jouw systeem wil benaderen, dient deze eerst een of meerdere valide claims neer te leggen bij de Access Control service. Dit kan simpelweg een gebruikersnaam en wachtwoord zijn. De Access Control service zal deze claims mappen naar claims die van toepassing zijn voor jouw systeem, waarna een security token wordt teruggegeven met daarin de claims. De derde partij stuurt deze token naar jouw systeem en is daarna geauthoriseerd en geauthenticeerd. Wat valide claims zijn en hoe je deze wilt mappen, dien je in te stellen via de AppFabric API of een tool, die bij de SDK wordt meegeleverd.

De ServiceBus en Access Control kregen tijdens de PDC09 de officiële platform naam AppFabric, maar beide services bestonden al eerder. Bij Access Control is het roer echter omgegooid en je kunt de huidige versie bijna als een nieuwe versie zien. Voorheen werd vooral de nadruk gelegd op de WS-* specificatie. Deze versie van Access Control is volledig gebouwd op REST. Dat houdt in dat het bruikbaar is op elk platform en elk systeem. De claims en security tokens worden in de header van een HTTP request meegegeven. Praktisch elke ontwikkelaar heeft ergens in zijn of haar carrière wel eens met System.Web.HttpRequest gewerkt en weet hoe dit in zijn werk gaat. In codevoorbeeld 6 kun je bijvoorbeeld zien hoe een security token in de header eruit ziet. Het is een name value collection waarvan elk een specifieke claim is.

```
role=Admin%2cUser&
customerName=Microsoft%20Corporation&
Issuer=https%3a%2f%2fnetmagazine.accesscontrol.windows.net
%2fWRAPv0.8&
Audience=http%3a%2f%2fnetmagazine%2fprintarticle&
ExpiresOn=1255912922&
HMACSHA256=yuVO%2fwc58%2ftYP36%2fDM1mS%2fHr0hswpsGTWwgfvAbpL64%3d
```

CODEVOORBEELD 6

Dat het eenvoudig is mag blijken uit het feit dat zelfs met puur JavaScript met behulp van Access Control authorisatie geregeld kan worden. Hiervoor zijn twee nieuwe open protocollen ontwikkeld, namelijk OAuth WRAP en SWT. Microsoft heeft geleerd van vorige protocollen en heeft deze samen met een groot aantal partners ontwikkeld, waaronder Yahoo en Google. SWT wordt gebruikt voor Federated Identities. Project Geneve, tegenwoordig Windows Identity Foundation, biedt wat Access Control ook biedt, maar dan binnen je eigen enterprise. Als jouw bedrijf of een

derde partij Federated security gebruikt, is het ook mogelijk met Access Control deze te mappen. De derde partij vraagt initieel bij haar Federated Identity Provider (bijvoorbeeld WIF) een token aan, stuurt deze token naar Access Control en krijgt een nieuwe token om naar jouw systeem te versturen.

Toekomst van Access Control

Zoals aangegeven is het op dit moment niet mogelijk om WS-* te gebruiken, maar dit is momenteel een van de hoogste prioriteiten binnen het Access Control team. Zowel WS-Trust als WS-Federation worden native ondersteund, waarmee hopelijk ook Windows CardSpace wat meer aandacht krijgt. Mijn hoop is dat het met Access Control eenvoudiger wordt om CardSpace te ondersteunen waardoor meer bedrijven hun websites met CardSpace uitbreiden.

Daarnaast wil Microsoft ondersteuning leveren voor andere Web Identity Providers, zoals Windows Live Id, Google, Yahoo, OpenID, etc. Op zich niet heel verwonderlijk, als je je voor kunt stellen dat het enige wat gedaan hoeft te worden is de gegevens die de Identity Providers bieden, te mappen naar een claim die jouw systeem aan kan. Theoretisch heel eenvoudig, de achterliggende implementatie voor Microsoft is waarschijnlijk wat complexer.

Twee smaken

Je kunt de AppFabric in twee varianten gebruiken. In dit artikel wordt de Azure variant besproken, die bestaat uit de voornoemde Service Bus en Access Control. Er is echter ook een Windows Server versie, die je lokaal op je ontwikkel-pc en uiteindelijk in je serverpark zult moeten installeren. Ook de servervariant bestaat momenteel uit twee componenten, te weten Project Velocity en Dublin. Beiden zijn nog in CTP status en zullen voor release waarschijnlijk nog hernoemd worden. Met Velocity kun je een gedistribueerd cache opzetten en deze laten repliceren over vele servers. Dublin gaat onder andere hosting van je WF en WCF services op zich nemen, inclusief vele additionele features die je normaal gesproken niet hebt of zelf dient te bouwen.

Hoewel het buiten de scope van dit artikel is, is het ook zeker interessant om aan deze variant van de AppFabric aandacht te besteden. Het is uiteindelijk ook niet ondenkbaar dat deze twee componenten uiteindelijk in enige vorm in Windows Azure en haar AppFabric terecht zullen komen. Net na de release van Windows Azure op de PDC 2008 was er een workflow component in Azure beschikbaar, maar deze is met het oog op het nieuw uit te komen Windows Workflow 4 verwijderd.

Als je meer informatie wilt vinden over de Azure AppFabric, kun je ook veel informatie vinden als je op internet zoekt m.b.v. de oude namen. Tijdens de bekendmaking van Windows Azure op de Professional Developers Conference van 2008 werd de naam .NET Services aangehouden. Nog verder terug werd de naam BizTalk Services gebruikt, hoewel te betwijfelen is of deze informatie nog relevant is voor de huidige AppFabric.

Conclusie

De Windows Azure AppFabric maakt het mogelijk om services en applicaties, zowel in de cloud als in eigen beheer, met elkaar op een veilige manier te laten communiceren. Met de Service Bus wordt het mogelijk over een complex infrastructuur te communiceren. Met de Access Control service wordt het eenvoudig gemaakt om claims te gebruiken voor authenticatie en authorisatie van gebruikers van jouw services of applicaties. Hiervoor is geen kennis van Windows Azure of de cloud zelf nodig. De Azure AppFabric zelf draait wel in de cloud. Waar het jouw taak is om systemen met elkaar te laten werken, verzorgt Microsoft betrouwbare en schaalbare services waar je jouw applicaties op kunt bouwen.

Dennis van der Stelt, is werkzaam bij Tellus. Daarvoor heeft hij gewerkt als software ontwikkelaar en trainer. Het blog van Dennis is te lezen op http://bloggingabout.net/blogs/dennis/. Voor vragen is hij bereikbaar via dennis@bloggingabout.net.