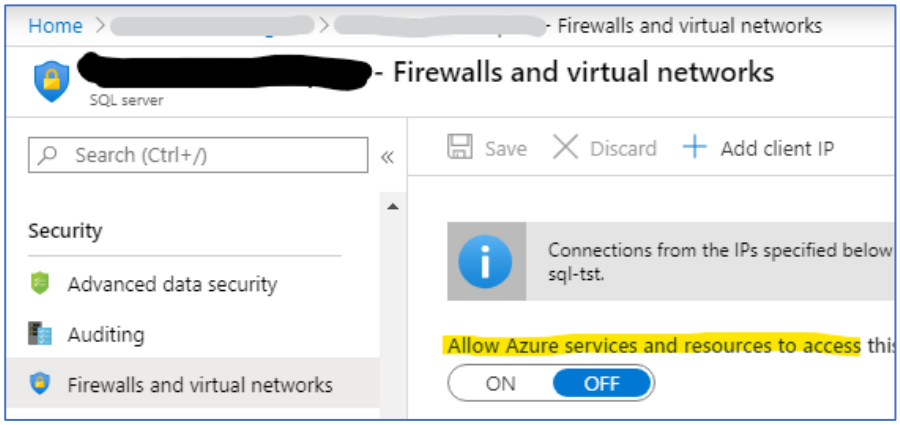
Automatisch instellen appservice IP nummers in Azure SQL firewall bij een release

Een Azure Resource Management en Azure DevOps puzzel voor webapp devs

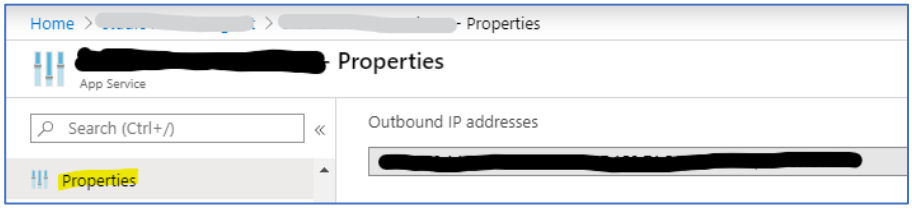
Het uitgangspunt van deze case is een Azure SQL database in een resource group. Er is één webapplicatie (binnen diezelfde resource group), die gebruik maakt van de database. Om de webapplicatie toegang te geven tot de database kan binnen Azure eenvoudigweg een switch worden omgezet naar “ON”:



Figuur 1 - Azure Portal: Geef Azure services toegang tot sql resource

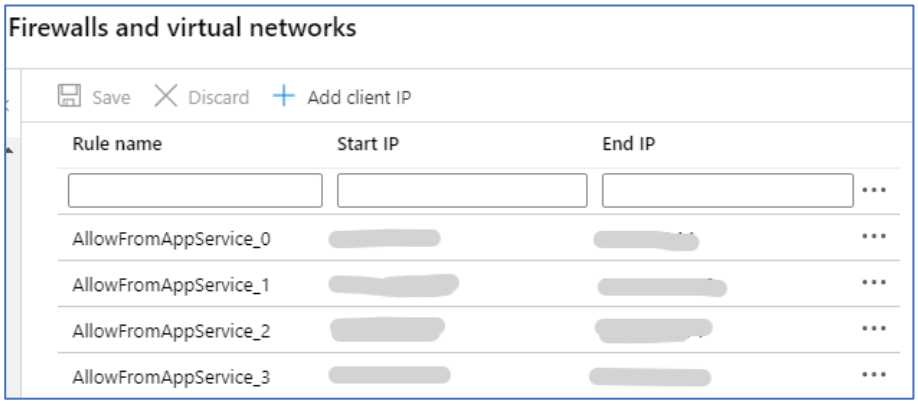
Deze optie geeft Azure services toegang tot SQL server (databases). Dit zijn niet alleen de Azure services binnen de resource group, of alleen de services binnen de Azure subscription, maar alle Azure services. Een ongewenste situatie. Graag willen we SQL server afschermen, zodat alleen de webapplicatie toegang heeft tot de SQL server (database).

Een appservice krijgt meerdere outbound IP adressen toegewezen.



Figuur 2 - Azure Portal: Outbound IP adressen van appservice

De outbound IP adressen kunnen worden toegevoegd in de firewall rules van de SQL server, maar deze IP adressen zouden weleens kunnen wijzigen.



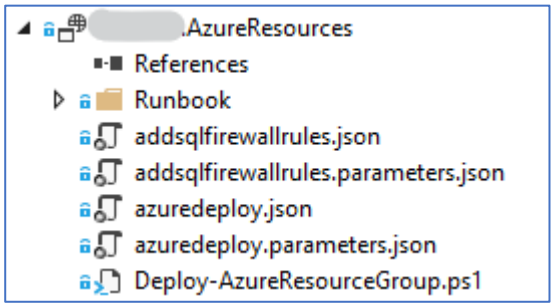
Figuur 3 - Azure Portal: SQL resource firewall rules

De Azure resources en de webapplicatie worden automatisch uitgerold door middel van ARM templates in combinatie met Azure DevOps. Na een uitrol zijn alle gegevens beschikbaar om de rules in te stellen. De wens is om ook de firewall rules automatisch in te stellen, door de gegevens van de appservice resource en SQL server resource tijdens de uitrol aan elkaar te koppelen.

De technische implementatie

In de Visual Studio solution van de webapplicatie is een Azure Resource Group project opgenomen met daarin twee ARM templates. Wanneer alle bestanden in source control zijn geplaatst wordt de oplossing met Azure DevOps gebuild en gereleased.

ARM templates



Figuur 4 - Visual Studio: Azure Resource Group project met ARM templates

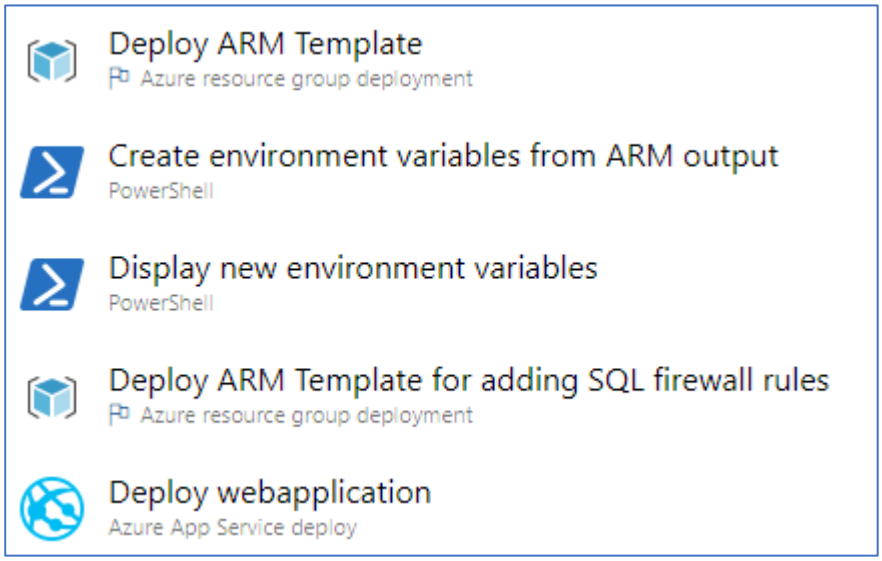
Het eerste ARM template (azuredeploy.json) rolt alle individuele Azure resources uit, waaronder de appservice en de SQL server instance. De outbound IP adressen van de appservice zijn opvraagbaar en worden, samen met de naam van de SQL resource, geëxporteerd via de outputs sectie van het template:

```
"outputs": {
   "outboundIpAddresses": {
     "type": "array",
     "value": "[split(reference(resourceId('Microsoft.Web/sites', variables('
appServiceName')), '2018-11-01').possibleOutboundIpAddresses, ',')]"
   },
   "sqlResourceName": {
     "type": "string",
     "value": "[variables('sqlResourceName')]"
   }
}
```

Het tweede ARM template (addsqlfirewallrules.json) krijgt als input parameter een array van IP adressen en de naam van de SQL server resource. Vervolgens wordt op iteratieve wijze (copy, count, index) de inhoud van de array toegevoegd aan de firewall rules.

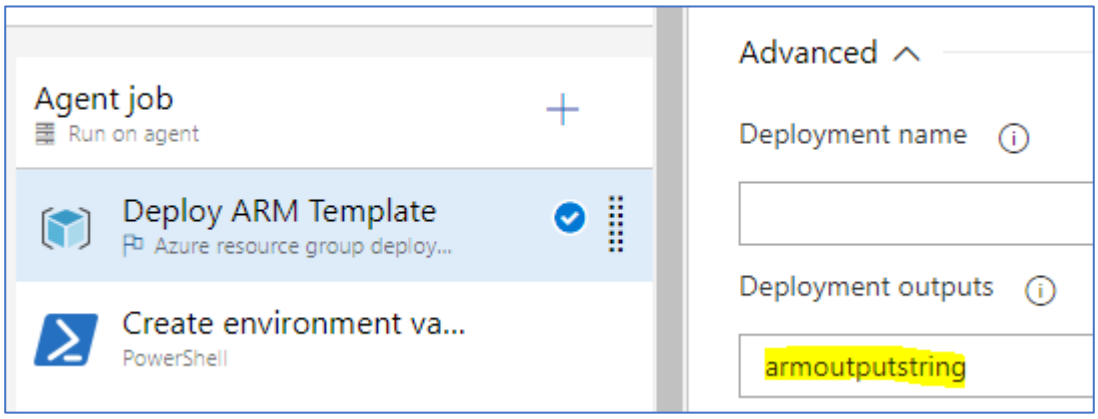
```
"parameters": {
    "outboundIpAddresses": {
      "type": "array",
      "metadata": {
        "description": "array of ip addresses"
      }
    },
    "sqlResourceName": {
      "type": "string",
      "metadata": {
        "description": "name of the sql resource"
      }
    }
}
"resources": [
  {
    "apiVersion": "2015-05-01-preview",
    "type": "Microsoft.Sql/servers/firewallRules",
    "comments": "Allow From Appservice",
    "name": "[concat(parameters('sqlResourceName'), '/', 'AllowFromAppService_', copy
Index())]",
    "location": "[resourceGroup().location]",
    "condition": "[greater(length(parameters('outboundIpAddresses')), 0)]",
    "properties": {
      "startIpAddress": "[parameters('outboundIpAddresses')[copyIndex()]]",
      "endIpAddress": "[parameters('outboundIpAddresses')[copyIndex()]]"
    },
    "copy": {
      "name": "firewallRuleCopy",
      "count": "[length(parameters('outboundIpAddresses'))]"
    }
  }
]
```

De volgende taken zijn gebruikt in de release om de appservice uit te rollen:



Figuur 5 - Azure DevOps: Release taken

Bij het uitrollen van het ARM template is een naam opgegeven voor de variabele waarin de outputs van het ARM template (als string) ingezet moet worden:

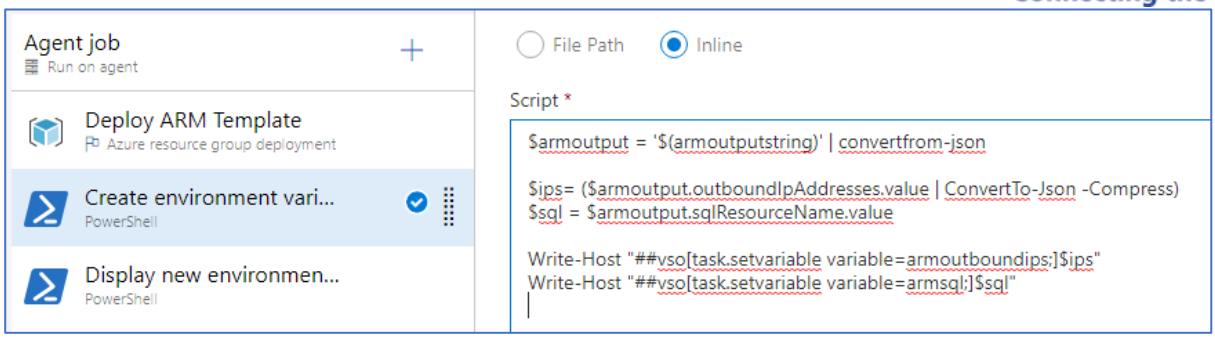


Figuur 6 - Azure DevOps: ARM template voor alle resources

Output van deze taak:

```
Resource group exists: true.
Creating deployment parameters.
The detected encoding for file 'D:\wrk2\r27\a\Package\Azure\azuredeploy.json' is 'utf-8'
Starting Deployment.
Deployment name is azuredeploy-20200123-103338-fea4
Updated output variable 'armoutputstring', which contains the outputs section of the current deployment object in string format.
Successfully deployed the template.
```

Voeg een PowerShell inline script toe om de variabele met ARM outputs om te vormen naar environment variabelen die in opeenvolgende taken kunnen worden gebruikt.



Figuur 7 - Azure DevOps: PowerShell script voor omvormen ARM output

Maak van de string weer een object:

```
$armoutput = '$(armoutputstring)' | convertfrom-json
```

Bewaar de waarden waarin je geïnteresseerd bent in een variabele. Gebruik `ConvertTo-Json -Compress`, zodat de variabele `$ips` een string array bevat:

```
$ips= ($armoutput.outboundIpAddresses.value | ConvertTo-Json -Compress)
$sql = $armoutput.sqlResourceName.value
```

Maak van deze variabele een environment variabele, zodat je deze in een volgende(!) taak kunt gebruiken. Je kunt de waarde niet in de huidige taak evalueren. Let op het gebruik en de plaatsing van de quotes:

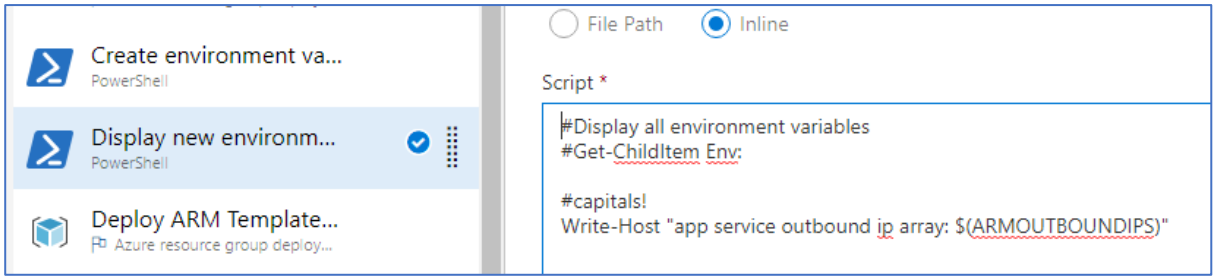
```
Write-Host "##vso[task.setvariable variable=armoutboundips;]$ips"
Write-Host "##vso[task.setvariable variable=armsql;]$sql"
```

Mocht je de inhoud van de lokale variabele willen bekijken:

```
Write-Host "app service outbound ip array: $ips"
```

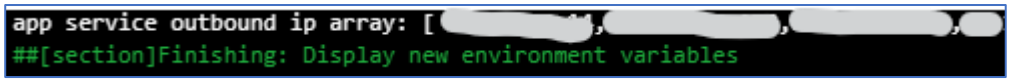
De output is dan:

De volgende taak is niet vereist voor de release. Het geeft feedback of de variabelen, die je zojuist gezet hebt, ook inderdaad bruikbaar zijn:



Figuur 8 - Azure DevOps: PowerShell controle taak

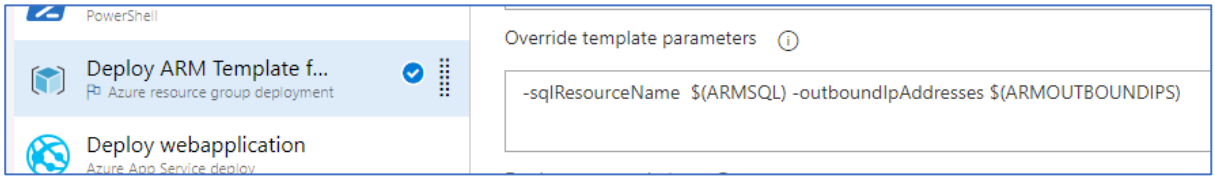
De output van deze taak, merk op dat de dubbele quotes in deze weergave verdwenen zijn, dit heeft verder geen effect.



Let op dat de nieuwe environment variabele alleen in hoofdletters beschikbaar is:

```
Write-Host "app service outbound ip array: $(ARMOUTBOUNDIPS)"
```

Nu kan de array met IP adressen worden doorgegeven als parameter aan het volgende ARM Template, dat de IP nummers toevoegt als rule in de SQL firewall.



Figuur 9 - Azure DevOps: ARM template voor firewall rules

```
-sqlResourceName  $(ARMSQL) -outboundIpAddresses $(ARMOUTBOUNDIPS)
```

Waarom deze implementatie?

Op zich is het een eenvoudige implementatie en binnen DevOps is het mogelijk op deze manier de twee acties achtereenvolgend uit te voeren. Maar omdat het één uitrol betreft, binnen één resource group, lijkt het logischer om alles enigszins bij elkaar te houden. Dit was initieel de intentie, maar onderweg richting de eindoplossing waren daar een aantal beren op de weg.

Leesbaarheid/onderhoudbaarheid

Wellicht is het mogelijk binnen het ARM template om de firewall rules en/of SQL resource afhankelijk te maken (dependson) van de appservice resource en vervolgens direct over de outbound IP adressen te itereren. Dit heb ik niet geprobeerd omdat ik niet dacht dat dit de leesbaarheid/onderhoudbaarheid van het ARM template ten goede zou komen, als dit al foutloos zou lukken. Het is namelijk niet mogelijk gedurende een ARM template uitrol de outbound IP adressen op te vangen in een variabele. Je hebt alleen de mogelijkheid om de resource properties uit te vragen.

Op alle plekken waar op dit moment in addsqlfirewallrules.json (zie boven) `parameters('outboundIpAddresses')` staat, zou dit vervangen moeten worden door:

```
split(reference(resourceId('Microsoft.Web/sites',
variables('webApplicationSettings').name), '2018-11-
01').properties.possibleOutboundIpAddresses, ',')
```

Linked Template (met link)

Na een ARM template uitrol kun je direct een tweede uitrol laten starten door een deployment resource te definiëren en deze te laten verwijzen naar een linked ARM template.

```
//nested deployment for adding sql firewall rules
{
  "apiVersion": "2018-05-01",
  "type": "Microsoft.Resources/deployments",
  "name": "linkedTemplateFirewallRules",
  "properties": {
    "mode": "Incremental",
    "templateLink": {
      "uri": "[variables('addSqlFirewallRulesARMTemplateUrl')]"
    },
    "parameters": {
      "sqlResourceName": {
        "value": "[variables('sqlResourceName')]"
      },
      "outboundIpAddresses": {
        "value": "[split(reference(resourceId('Microsoft.Web/sites', variables('webAp
plicationSettings').name), '2018-11-
01').properties.possibleOutboundIpAddresses, ',')]"
      }
    }
  },
  "dependsOn": [
    "[resourceId('Microsoft.Web/sites', variables('appServiceResourceName'))]",
    "[resourceId('Microsoft.Sql/servers', variables('sqlResourceName'))]"
  ]
}
```

Absolute of relatieve paden worden niet geaccepteerd als template link. Het moet persé een webadres zijn. In het ARM template kun je de url van het template vanuit de deployment context opvragen. De template link zou dan als volgt zou kunnen zijn:

```
"[uri(concat(deployment().properties.templateLink.uri, 'addsqlfirewallrules.json?'), '')]"
```

Dit werkt echter niet bij een deployment vanuit Visual Studio of vanuit Azure DevOps.

In dat geval zou het tweede ARM template in BlobStorage geplaatst kunnen worden.

```
"[concat('https://',variables('blobStorageResourceName'),'.blob.core.windows.net/',
'blobContainerName/addsqlfirewallrules.json')]"
```

Maar dat maakt het nodeloos ingewikkeld, ook om bijvoorbeeld toegang tot het bestand in te regelen (SAS token), aangezien we deze scripts en blobcontainer liever niet publiek toegankelijk willen maken.

Linked Template (inline)

Het tweede template kan ook integraal in het eerste worden gevoegd in plaats van een link naar een bestand te gebruiken.

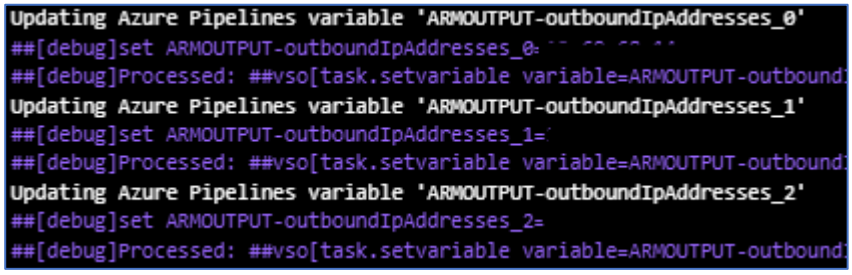
```
"properties": {
      "mode": "Incremental",

      "template": { [inhoud van addsqlfirewallrules.json op deze plek] }
```

Het is voor te stellen dat dit de leesbaarheid/onderhoudbaarheid van het ARM template ook niet ten goede komt. Er is wel een poging gewaagd. Maar door de onduidelijke foutmeldingen bij het releasen van het template hebben we deze poging gestaakt. Misschien is een extra stap nodig, maar daarmee wordt het wel overzichtelijker en beter te beheren.

ARMOutputs DevOps release taak

De ‘ARM Outputs’ taak wordt doorgaans gebruikt om de output parameters vanuit het ARM template beschikbaar te maken in de release taken. In dit geval was er een output parameter van het type Array. Ook daar kan deze taak mee omgaan. Echter, worden alle array entries omgezet tot losse variabelen.



Figuur 10 - Losse variabelen

In dit geval wilde ik de array als zodanig doorgeven aan het volgende ARM template. Gelukkig kun je tegenwoordig de outputs in de ‘Azure resource group deployment’ taak opvangen in een variabele en ze zelf met PowerShell uitvragen en omzetten tot environment variabelen.