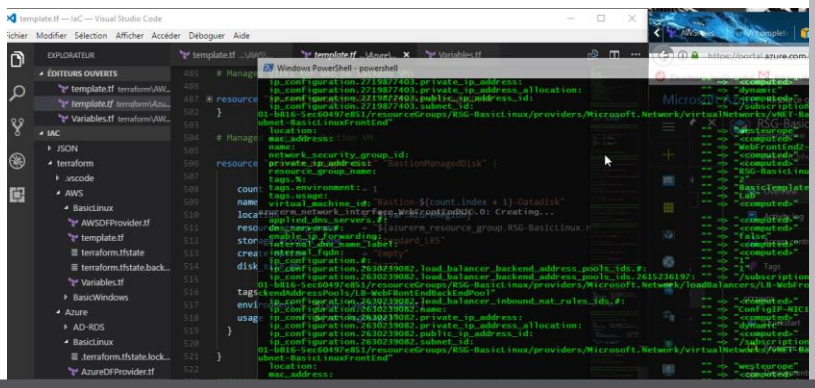

Teknews.cloud

Using ARM JSON Template with Terraform - A use case with Azure Subnet Service Endpoint

David Frappart

21/01/2018

Contributor

Name	Title	Email
David Frappart	Cloud Architect	david@teknews.cloud

Version Control

Version	Date	State
1.0	2018/07/16	Final

Table of Contents

1 Introduction

For those who read my previous article, you may have guessed that I'm quite fond of Terraform for Infrastructure as Code. IT's not that I dislike completely the native tools such as ARM JSON, bu I found that it was faster to get on board people with IaC Through Terraform.
That being said, since Terraform is developed by a 3rd party company (aka Hashicorp), it may happen that the targeted resource just does not have an equivalent in terraform.
In this article, we will look for workaround of such case.

2 When Terraform provider is not going fast enough

2.1 Concepts – the azurerm_template_deployment resource

As I said in introduction, with the crazy fast pace of new stuff coming in Azure, it is obviously tedious work to keep up to date for Terraform developpers and those working on the Azurerm provider. That being said they do make an incredible work as we are not years away from new stuff in Azure but rather just a few months.
A workaround, when the resources to deploy are not available is to use the resource azurerm_template_deployment.
This resource allows us to include a JSON template as a long string. It requires only a few parameters which are the name of the template, the resource group in which the template is to be deployed and the template body.

2.2 Syntaxe

The resource, visually speaking, come in the following format:

```
resource "azurerm_template_deployment" "Template-Example" {
  name                = "terraexampletemplate"
  resource_group_name = "${module.ResourceGroupInfra.Name}"

  template_body = <<DEPLOY
{
…
}
DEPLOY


  parameters {
```

```
    "param1"              = "${var.param1}"
    "param2"              = "${module.ModuleNAme.Param2}"
  }

  deployment_mode = "Incremental"
}
```

As discussed earlier. Terraform expect the long string in a format as displayed below:

```
template_body = <<DEPLOY
{
}
DEPLOY
```

Between the two balises DEPLOY, we just add the JSON Template for the resource we want to deploy.

One thing however that is not so nice with this long string is the lack of interpolation capability which is one of the strong point of Terraform. Fortunately, we can use interpolation for the template parameters by adding after the JSON string the parameters section in the following format:

```
  parameters {
    "param1"          = "${var.param1}"
    "param2"          = "${module.ModuleNAme.Param2}"
  }
```

The parameters sections accept terraform variable or module output. The only prerequisite being that the parameters name (in the example, param1 and param2) exist in the JSON string.

Lastely, the parameter deployment_mode allow us to specify what can of deployment is expected. Usually, as stated in Terraform documentation, we use the value Incremental.

2.3 A simple example

As an example, we will try to deploy a subnet with a few endpoint service activated in a Vnet deployed through Terraform. Now, the endpoint are available directly in Terraform provider so it is not necessarily a good example regarding the main use case. However, it is simple enough so that we can easily write the JSON part without too much diiffuculty for non ARM JSON specialist such as myself.
To de so, we need to deploy first with terraform a Vnet in a resource group, an dnothing else. Thus the simple example...

3 The ARM JSON for Services Endpoints

In a previous article, I already mentioned the JSON string for the endpoints services in the Azure subnet. However, I was displaying the resource through the deployment of a VNet through JSON. While it works, it also erases terraform created subnet, which could pose some problem. Thus, this time, i display below a template to deploy on an existing VNet:

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "String"
        },
        "existingVNetName": {
            "type": "String"
        },
        "subnetName": {
            "type": "String"
        },
        "subnetAddressPrefix": {
            "type": "String"
        },
        "nSGID": {
            "type": "String"
        }
    },
    "resources": [
        {
            "type": "Microsoft.Network/virtualNetworks/subnets",
            "name": "[concat(parameters('existingVNETName'), '/',
parameters('subnetName'))]",
            "apiVersion": "2017-10-01",
            "properties": {
                "addressPrefix": "[parameters('subnetAddressPrefix')]",
                "networkSecurityGroup": {
                    "id": "[parameters('nSGID')]",
                    "location": "[parameters('location')]"
                },
                "serviceEndpoints": [
                    {
                        "service": "Microsoft.AzureCosmosDB"
```

```
                    },
                    {
                        "service": "Microsoft.KeyVault"
                    },
                    {
                        "service": "Microsoft.Sql"
                    },
                    {
                        "service": "Microsoft.Storage"
                    },
                    {
                        "service": "Microsoft.ServiceBus"
                    },
                    {
                        "service": "Microsoft.EventHub"
                    }
                ]
            }

        }
    ]
}
```

4 Hiding the JSON File from the .tf file

Now then, we discussed previously how to call the JSON with the terraform resource azurerm_template_deployment. However, since the JSON is displayed as a big block inside the terraform config, it makes the whole config difficult to read. One way to avoid this is to use the template capability of terraform and to reference the file through interpolation. For this, we need the template provider and the associated data source template_file. The template data source refers to the JSON file and can then be interpolated in place of the long JSON string:

```
#########################################################################
# This file activate Endpoint for CosmosDB
#########################################################################

data "template_file" "templateSubnetwEP" {
  template = "${file("./Templates/templateEPSubnet.json")}"
}

resource "azurerm_template_deployment" "Template-VNetEndpoint" {
  name                = "terraVNettemplate"
```

```
  resource_group_name = "${module.ResourceGroupInfra.Name}"

  template_body = "${data.template_file.templateSubnetwEP.rendered}"

  parameters {
    "location"            = "${module.SampleArchi_vNet.RGLocation}"
    "ExistingVNetName"    = "${module.SampleArchi_vNet.Name}"
    "subnetName"          = "CreatedfromJSON_Subnet"
    "subnetAddressPrefix" = "10.0.3.0/24"
    "nSGID"               = "${module.NSG_FE_Subnet.Id}"
  }

  deployment_mode = "Incremental"
}
```

5 Conclusion

In this article we looked for workaround for the times when Terraform is just not enough. A word of caution though, Terraform has no control on resources deployed in the JSON, meaning we can't find those in the state. It means that the idempotence is not really full with this method. Also, error may happen when running the destroy command and particular caution should be taken when deploying resource like that. As an example, I tried to deploy subnet both with terraform and with JSON (through Terraform) with the VNet resource in JSON, and the config was not consistent. Subnet were overwritten between 2 deployments. That's because the subnet are nested resources in the VNet, so if a VNet is deployed in JSON and Terraform resource, it becomes unstable. That point is taken care with the example described here (with the resource in JSON a particular Subnet on the already existing VNet) but since there is a blind point, it can become tricky.
Anyway, I will probably still deploy with template for resource in preview or not already available, but test it thoroughly, just in case.

Another Tech blog

My thoughts and experiences on Cloud related tek