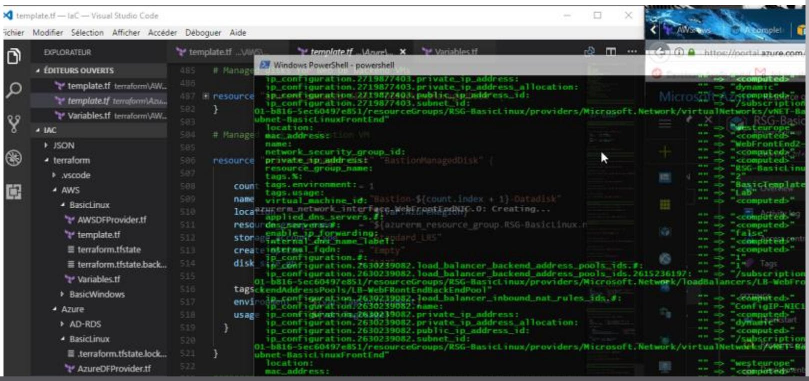


Teknews.cloud

Tips with NSG rules management in Azure through Terraform

David Frappart

26/02/2018

Contributor

Name	Title	Email
David Frappart	Cloud Architect	david@dfitcons.info

Version Control

Version	Date	State
1.0	2018/02/26	Final

Table of Contents

1 Introduction

In this article, we will come back to NSG and NSG rules creation options with Terraform and take a look on useful tips to know when desinging rules to be provisioned through Terraform.

2 Short review on NSG and NSG rules objects in Terraform

First, let's review shortly how to create NSG and associated rules with Terraform.

2.1 NSG Object

Terraform allows the creation of NSG with the associated resource azurerm_network_security_group. Below is a example of an NSG creation:

```
resource "azurerm_network_security_group" "Terra-NSG" {

    name                = "${var.NSGName}"
    location            = "${var.NSGLocation}"
    resource_group_name = "${var.RGName}"

  security_rule {
    name                       = "OK-HTTP-entrant"
    priority                   = 1000
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "80"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }

  security_rule {
    name                       = "OK-HTTPS-entrant"
    priority                   = 1100
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "443"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
  security_rule {
    name                       = "OK-SSH-entrant"
    priority                   = 1200
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "22"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }

    tags {
    Environment       = "${var.EnvironmentTag}"
    Location          = "${var.LocationTag}"
    ResourceType      = "${var.ResourceTypeTag}"
    SLA               = "${var.SLATag}"
    Owner             = "${var.OwnerTag}"
    ProvisioningDate  = "${var.ProvisioningDateTag}"
    }
}
```

In this example, the NSG rules are included in the NSG. However, while providing a simple use for the NSG and its associated rules, this model of coding the NSG is not very flexible, especially when considering the use of module. Indeed, having all the rules ti be added in the NSG object would imply to have only one type of NSG with always the same rules... Not very “Real life applicabe” to me.

2.2 Moving NSG rules outside of NSG

The answer to the previously explained issue comes with the Terraform NSG rule object. This resource allow to provision only the rule, independantly of the NSG. Evidently, an NSG must be specified so that Terraform (and Azure API) knows on wich NSG to associate the rule. The resource azurerm_network_security_rule is defined as below:

```
resource "azurerm_network_security_rule" "Terra-NSGRule" {


    name                            = "${var.NSGRuleName}"
    priority                        = "${var.NSGRulePriority}"
    direction                       = "${var.NSGRuleDirection}"
    access                          = "${var.NSGRuleAccess}"
    protocol                        = "${var.NSGRuleProtocol}"
    source_port_range             = "${var.NSGRuleSourcePortRange}"
    destination_port_range          = "${var.NSGRuleDestinationPortRanges}"
    source_address_prefixe          = "${var.NSGRuleSourceAddressPrefixes}"
    destination_address_prefixe     = "${var.NSGRuleDestinationAddressPrefixes}"
    resource_group_name             = "${var.RGName}"
    network_security_group_name     = "${var.NSGReference}"

}
```

As mentionned earlier, we do reference an NSG with the parameter network_security_group_name. This separation, while adding more object to be created for the required rules, has the advantage of being really more flexible. When used with module, It is also more DRY, since, instead of adding multiple time the code for the NSG rule, we just have to call the module for each rules required, while specifying the appropriate input, such as the NSG targeted for the rule but also the source and destination for the traffic.

3 NSG rules tips

After this review, let's look in the NSG rule object details.

3.1 Simple NSG rule

The first case that we will call here the simple NSG rule is to define a simple rule, with a unique source and destination for each rules, and a unique source and destination port range. This is typically the example shown earlier. Taking into consideration that the code is inside a module, we would have something like that to call the module and specify the appropriates inputs:

```
module "AllowSSHFromJumptoSampleSubnet" {

    #Module source
    source = "./Modules/05 NSGRule"

    #Module variable
    RGName = "${module.RG_SAmple.Name}"
    NSGReference = "${module.NSG_Sample.Name}"
    NSGRuleName = "AllowSSHRDPFromJumptoSampleSubnet"
    NSGRulePriority = 101
    NSGRuleDirection = "Inbound"
    NSGRuleAccess = "Allow"
    NSGRuleProtocol = "*"
    NSGRuleSourcePortRange = "*"
    NSGRuleDestinationPortRange = 22
    NSGRuleSourceAddressPrefixe = "${var.JumpServers}"
    NSGRuleDestinationAddressPrefixe = "${lookup(var.SubnetAddressRange, 0)}"
}
```

Ok, this is a simple rule and it does the job required.

In some case, we may need to define traffic on multiple source or target IP ranges. It could also be the same for the Network port.
With a simple rule, we do not have a possibility to specify more than one IP range with the exception of the service tags:

- Internet which define all address ranges outside of the VNet
- VirtualNetwork, which define all address ranges in the VNet
- AzureLoadBalancer which define all address ranges associated to the Azure Load Balancer service

Also, quite recently, were added service tag associated with the new capability of access to traffic manager, Azure Storage and Azure SQL:

- AzureTrafficManager
- Storage
- Sql

To be noted, there is a possibility to specify a region for Azure Storage and Azure SQL with the format Storage.<Region> and Sql.<Region>.
However, for disjointed IP ranges or port, there is no other choice than creating more rules. While this can easily be done, there is an overhead on the rules management and there is also the limitation on the number of NSG rules per subscription.

3.2 "Complex" NSG rule

The answer to those limitations is what I will call here the complex NSG rule. The fact is, the rule object in itself is not really more complex. Simply, Terraform allows to choose between a single range or multiple ranges by switching the parameters:

- source_address_prefix
- destination_address_prefix
- source_port_range
- destination_port_range

To the plural version :

- source_address_prefixes
- destination_address_prefixes
- source_port_ranges
- destination_port_ranges

The impact on this modification is that we solve the issue mentioned before and we can have in one rule more than one ports or port ranges and more than one IP ranges. So, in the case of On Premise subnets that need access to an Azure VNet or subnet, it becomes possible to specify the IP ranges in a variable and to have only one rule specifying the source IP range.
Thus, if the On Premise IP ranges list change, the rules update can benefit from the variables and there is no need to create more rules.
It also allows to avoid creating too much rules on the subscription quota. There is also in this matter a limitation to be aware of which is the number of IP Addresses or IP Addresses ranges allowed per rule. This number goes up to 2000 by default and 4000 on support request.

To illustrate the benefits, let's take an example with a requirement to access through RDP and SSH servers located in an Azure subnet from an On Premise list of IP ranges. We would define the port list and the Range list with variable as follow:

```
variable "AdminPortRange" {
    type        = "list"
    default     = ["22","3389"]
}
```

```
variable "OnPremSubnets" {


    type = "list"
    default = ["10.0.0.0/16",
    "172.16.19.0/20",
    "192.168.0.0/16",
    ]
}
```

Obviously the resource in the module would need to be modified accordingly, since it is not possible to have both the singular and the plural versions of each parameter at the same time:

```
resource "azurerm_network_security_rule" "Terra-NSGRule" {


    name                                = "${var.NSGRuleName}"
    priority                            = "${var.NSGRulePriority}"
    direction                           = "${var.NSGRuleDirection}"
    access                              = "${var.NSGRuleAccess}"
    protocol                            = "${var.NSGRuleProtocol}"
    source_port_range                 = "${var.NSGRuleSourcePortRange}"
    destination_port_ranges             = "${var.NSGRuleDestinationPortRanges}"
    source_address_prefixes             = "${var.NSGRuleSourceAddressPrefixes}"
    destination_address_prefixes        = "${var.NSGRuleDestinationAddressPrefixes}"
    resource_group_name                 = "${var.RGName}"
    network_security_group_name         = "${var.NSGReference}"

}
```

Another thing to take into consideration is the variable block in the module. The change from one to many ranges/ports implies changing the variable type from string to list. It is also impacted on the module call which becomes something like this:

```
module "AllowSSHRDPFromOnPrem" {

    #Module source
    source = "./Modules/05 NSGRule"

    #Module variable
    RGName = "${module.RG_Sample.Name}"
    NSGReference = "${module.NSG_Sample.Name}"
    NSGRuleName = " AllowSSHRDPFromOnPrem"
    NSGRulePriority = 201
    NSGRuleDirection = "Inbound"
    NSGRuleAccess = "Allow"
    NSGRuleProtocol = "*"
    #NSGRuleSourcePortRange = "*"
    NSGRuleDestinationPortRanges = ["${var.AdminPortRange}"]
    NSGRuleSourceAddressPrefixes = ["${var.OnPremSubnets}"]
    NSGRuleDestinationAddressPrefixes = ["${lookup(var.SubnetAddressRange, 0)}"]
}
```

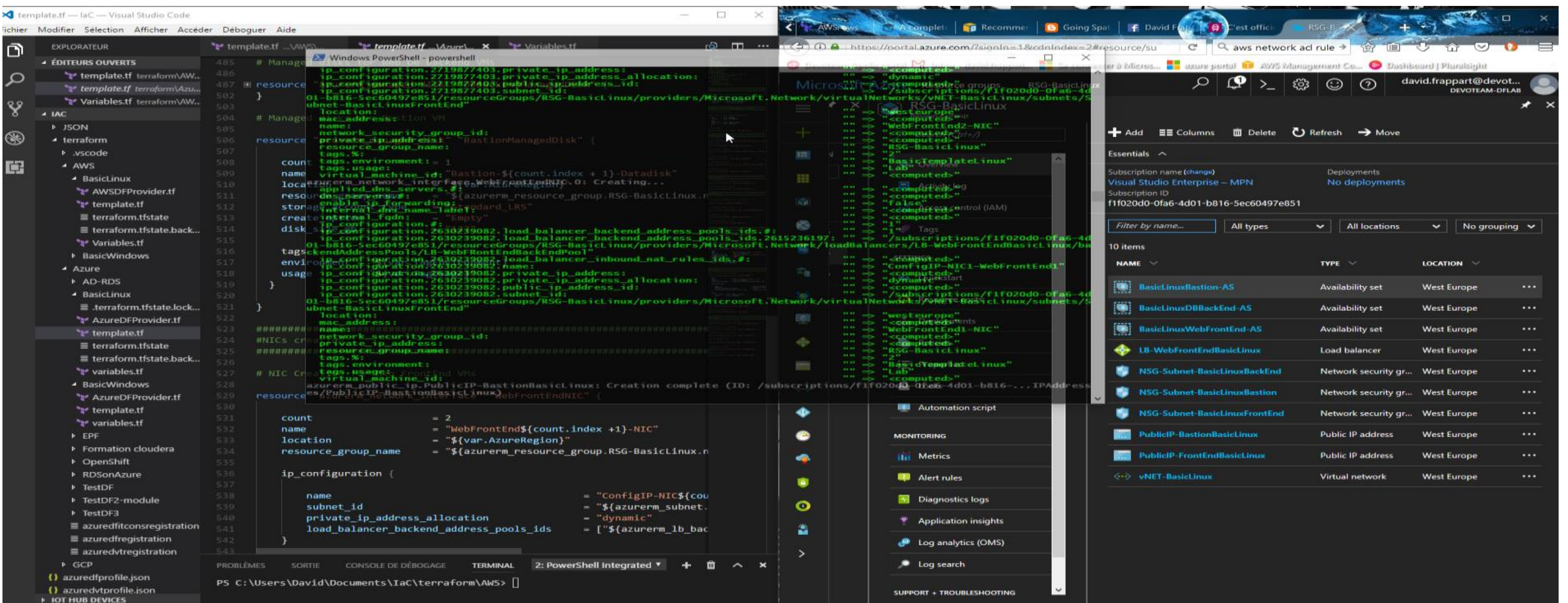
Cautious readers will notice that the NSGRuleSourePortRange remains singular but is also commented. There are 2 reasons behind this. First, the source port range is usually *Any*. But when we use the plural version of the parameters, we do not have access to service tags, and in API, *Any* is also a service tag, resulting in error when using it with the Complex Rule version. So we chose here to keep it on the singular version since the majority of the case requires Any as a source port range. Second, since it is always (or almost always) any, we can define it in the module with a default value so that the module call is simplified without the need to specify the source port range.
Lists variables syntax in Terraform is materialized with square brackets [ ] containing all strings of the list in quotes.

4 Conclusion - Considerations for NSG rules module design

So now that we have looked in the available possibilities for NSG modules comes the design choices. There are some serious limitations with the simple model but there are also the advantages of providing the access to Azure Services through services tags. On the other hand, the complex model, coupled with appropriate variables planification, provides some answers to the frequent need that comes from an IT service. My choice here would be to define use case for each model and to have both modules available. There is not so much overhead to have both modules in a module library and it would answer most of the potential requirement possible more easily.
Last, there is no GitHub repo with both modules available yet this time. However, the code detailed in the article should be helpful enough for the impatient ones.

Another Tech blog

My thoughts and experiences on Cloud related tek



```
    count                       = 2
    name                        = "WebFrontEnd${count.index +1}-NIC"
    location                    = "${var.AzureRegion}"
    resource_group_name         = "${azurerm_resource_group.RSG-BasicLinux.n

    ip_configuration {

        name                                    = "ConfigIP-NIC${cou
        subnet_id                               = "${azurerm_subnet.
        private_ip_address_allocation           = "dynamic"
        load_balancer_backend_address_pools_ids = ["${azurerm_lb_bac
    }
```

```
  ip_configuration.2719877403.private_ip_address:
  ip_configuration.2719877403.private_ip_address_allocation:
  ip_configuration.2719877403.public_ip_address_id:
  ip_configuration.2719877403.subnet_id:
01-b816-5ec60497e851/resourceGroups/RSG-BasicLinux/providers/Microsoft.Network/virtualNetworks/vNET-BasicLinux/subnets/S
ubnet-BasicLinuxFrontEnd"
  location:
  mac_address:
  name:
  network_security_group_id:
  private_ip_address:
  resource_group_name:
  tags.%:
  tags.environment:
  tags.usage:
  virtual_machine_id:
azurerm_network_interface.WebFrontEndNIC.0: Creating...
  applied_dns_servers.#:
  dns_servers.#:
  enable_ip_forwarding:
  internal_dns_name_label:
  internal_fqdn:
  ip_configuration.#:
  ip_configuration.2630239082.load_balancer_backend_address_pools_ids.#:
  ip_configuration.2630239082.load_balancer_backend_address_pools_ids.2615236197:
01-b816-5ec60497e851/resourceGroups/RSG-BasicLinux/providers/Microsoft.Network/loadBalancers/LB-WebFrontEndBasicLinux/ba
ckendAddressPools/LB-WebFRontEndBackEndPool"
  ip_configuration.2630239082.load_balancer_inbound_nat_rules_ids.#:
  ip_configuration.2630239082.name:
  ip_configuration.2630239082.private_ip_address:
  ip_configuration.2630239082.private_ip_address_allocation:
  ip_configuration.2630239082.public_ip_address_id:
  ip_configuration.2630239082.subnet_id:
01-b816-5ec60497e851/resourceGroups/RSG-BasicLinux/providers/Microsoft.Network/virtualNetworks/vNET-BasicLinux/subnets/S
ubnet-BasicLinuxFrontEnd"
  location:
  mac_address:
  name:
  network_security_group_id:
  private_ip_address:
  resource_group_name:
  tags.%:
  tags.environment:
  tags.usage:
  virtual_machine_id:
azurerm_public_ip.PublicIP-BastionBasicLinux: Creation complete (ID: /subscriptions/f1f020d0-0fa6-4d01-b816-...IPAddress
es/PublicIP-BastionBasicLinux)
```

```
PS C:\Users\David\Documents\IaC\terraform\AWS> 
```

Subscription name (change)
Visual Studio Enterprise – MPN
Subscription ID
f1f020d0-0fa6-4d01-b816-5ec60497e851

Deployments
No deployments

10 items

NAME	TYPE	LOCATION
BasicLinuxBastion-AS	Availability set	West Europe
BasicLinuxDBBackEnd-AS	Availability set	West Europe
BasicLinuxWebFrontEnd-AS	Availability set	West Europe
LB-WebFrontEndBasicLinux	Load balancer	West Europe
NSG-Subnet-BasicLinuxBackEnd	Network security gr...	West Europe
NSG-Subnet-BasicLinuxBastion	Network security gr...	West Europe
NSG-Subnet-BasicLinuxFrontEnd	Network security gr...	West Europe
PublicIP-BastionBasicLinux	Public IP address	West Europe
PublicIP-FrontEndBasicLinux	Public IP address	West Europe
vNET-BasicLinux	Virtual network	West Europe