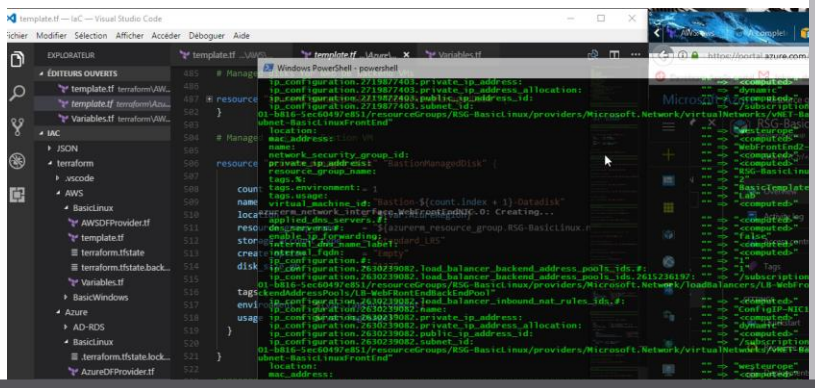


Teknews.cloud

Exploring Terraform features for Infrastructure as Code optimizations

David Frappart

01/12/2017

Contributor

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

Version Control

Version	Date	State
1.0	2017/12/01	Final

Table of Contents

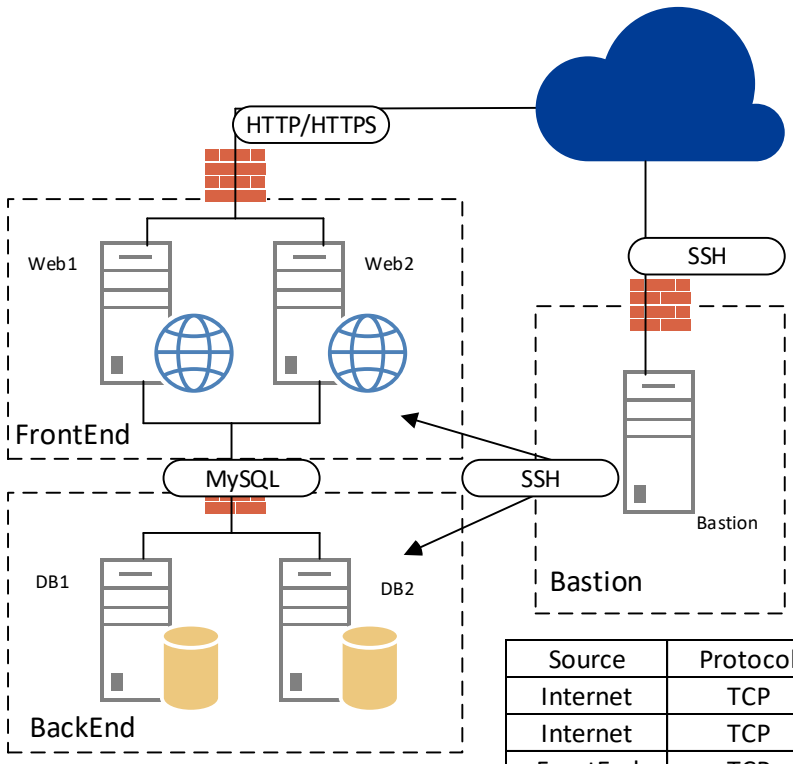
1 Optimizing a Terraform infrastructure as code template

We explored in a previous article how to deploy a sample architecture with Terraform. While we could demonstrate the power of Terraform to deliver standardized environment through the coding of the infrastructure in Terraform, we limited ourselves with a basic template, monolithic. Meaning this template included in a main file all the Azure resources declaration and provisioning configuration.
In this article, we propose to go beyond the simple monolithic template and use some Terraform built-in functions and the module concept to DRY the code.

2 Target Architecture

The sample architecture is identical to the previous article, consisting of Front-End Web Servers, behind a load balancer, Back-End Database server and a bastion server accessible through SSH.

2.1 Architecture Schema



Source	Protocol	Port	Target
Internet	TCP	80/443	FrontEnd
Internet	TCP	22	Bastion
FrontEnd	TCP	3306	BackEnd
Bastion	TCP	22	FrontEnd
Bastion	TCP	22	BackEnd

2.2 Related Azure Building Blocks

Since the architecture is similar to the previous one, we keep the same Azure building blocks:

- A single resource group
- A vNet decomposed in 3 subnets, Front-End, Back-End and Bastion
- Network Security group associated to the Subnet
- An internet facing Azure Load Balancer
- 1 public IP linked to the Load balancer
- 1 public IP linked to the Bastion Server
- Azure VMs for the Front-End Web
- 2 Aure VMs for the Back-End DB
- 1 Azure VMs for the Bastion Server
- Additional Managed disks for the Data disks of each VMs
- VMs Extension Agent for Custom Script deployment (more on this subject in the code analysis)
- Azure Network Watcher Agent (because Network Watcher Agent is nice and we will look into it in a later blog article)

3 Terraform advanced features

3.1 Variable usage in the template

Available type of variable in Terraform are:

- String
- List
- Map

Until now, we used mainly string variables. With this kind of variable, we can avoid hardcoding of value by the use of string for single value variable.
String variable are declared as a variable block as follow:

```
variable "RGName" {

    type    = "string"
    default = "RG-001"
}
```

And are used in the template by using the interpolation syntax as in the following example:

```
resource "azurerm_resource_group" "RG-BasicLinux" {

    name        = "${var.RGName}"
    location    = ""
}
```

Maps variables are lookup table from string keys to string values. We display an example of map declaration below:

```
variable "VMSize" {

  type = "map"
  default = {
      "0" = "Standard_F1S"
      "1" = "Standard_F2s"
      "2" = "Standard_F4S"
      "3" = "Standard_F8S"

  }
}
```

This variable is then used as follow in a template:

```
module "VMs_FEWEB" {

    #module source

    source = "./Modules/14 LinuxVMWithCount"

    #Module variables

    VMCount                     = "3"
    VMName                      = "WEB-FE"
    VMLocation                  = "${var.AzureRegion}"
    VMRG                        = "${module.ResourceGroup.Name}"
    VMNICid                     = ["${module.NICs_FEWEB.LBIds}"]
    VMSize                      = "${lookup(var.VMSize, 0)}"


}
```

As displayed, we read the value of the map VMSize with the lookup() function. We will go back to this function in the next chapter.

Last, the list variable is an ordered sequence of strings, indexed and starting from 0. AS an example, we can declare the following list variable:

```
variable "VMSize" {

  type = "list"
```

```
  default = ["Standard_F1S","Standard_F2s","Standard_F4S","Standard_F8S"]
}
```

However, we will not use list variable in the template.

3.2 Terraform Module

Module are useful in case of multiple similar resources used in a template. In the previous template, when creating VMs, we simply copy-paste code chunk and modified the associated parameters for each resource. However, this is ot a very efficient way of coding. Typically, it is not optimum in a DRY (Don't Repeat Yourself) sense. There is a way to optimize the code in this sense with Terraform modules.
A simple description would be to consider Terraform module as the equivalent of functions in other development languages. Following this logic, when creating a VM, instead of writing multiple time the same code, we use a module in which the resource is created. The module is called each time a type of resource is created. As in other language, the module takes input and can optionnaly return output. Finally, the code for the resource creation is only created once and called as a module as many times as needed.
On the other hand, the more complex a resource is, and the more parameters it requires, the more input may have to be used to call the module. Or another way would be to create more module for variations of the resource with a different set of fixed parameters.
So regarding input, we have two approach, either with multiple input and a few module reusable, or with less input but a wider range of module for the same resource.
Now regarding the output, as for other development languages, it is completely optional. However, in the case of an infrastructure template, on some occasions, outputing some values aboutn one resource may be useful for other resource implementation. For example, an Azure vNet requires a Resource Group reference. Let's demonstrate this use case. In the following code, we call a module used for a Resource Group Creation:

```
module "ResourceGroup" {

    #Module Location
    source = "./Modules/01 ResourceGroup"

    #Module variable
    RGName                  = "${var.RGName}"
    RGLocation              = "${var.AzureRegion}"
    EnvironmentTag          = "${var.EnvironmentTag}"
    EnvironmentUsageTag     = "${var.EnvironmentUsageTag}"
}
```

Now let's have a look at the code contained in the module:

```
variable "RGName" {
  type    = "string"
  default = "DefaultRG"
}


variable "RGLocation" {
  type    = "string"
  default = "Westeurope"
}


variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
}

variable "EnvironmentUsageTag" {
  type    = "string"
  default = "Poc usage only"
}

#Creating a Resource Group
resource "azurerm_resource_group" "Terra-RG" {


    name        = "${var.RGName}"
    location    = "${var.RGLocation}"

    tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
    }

}
```

In this code, we can see 2 sections. The first one is a list of variables declaration. The second is the Azure Resource Group declaration, refering to the variables declared earlier.
Now if we compare the azurerm_resouce_group resource parameters requirements with the module variables in the module call, we can see without surprise that they are the same.
A few more information on the module. We call a module with a reference to its path as in

```
source = "./Modules/01 ResourceGroup"
```

We differentiate the differente resource created with the module with the name of the object created by the module as in the following:

```
module "ResourceGroup" {...

}
```

At this point, we are aboslutely unable to use any information from the module. The reason being that no output were declared in the module. A useful attribute to export would be something identifying the Resource Group as a unique resource in Azure. This attribute is the Azure id resource and can be exported with the output command. As a matter of fact, not only properties which are attributed after creation can be exported. It is also possible to export other values specified at the creation, like the Resource Group Name parameter. When exporting values, we code the following:

```
output "Name" {

  value = "${azurerm_resource_group.Terra-RG.name}"
}

output "Location" {

  value = "${azurerm_resource_group.Terra-RG.location}"
}

output "Id" {

  value = "${azurerm_resource_group.Terra-RG.id}"
}
```

After adding this code in the module, it now export the Id, Location and Name of the corresponding created resource. However, since it was created inside a module called ResourceGroup, when using one of the value exported, we will use the following syntax:

```
RGName                              = "${module.ResourceGroup.Name}"
```

Calling a module is similar to a resource. When calling the resource we use the following:

```
azurerm_resource_group.<resource_name>.<resource_property>
```

A module can in a syntax point of view be seen as a resource also. So to call the module exported value, we would use the following:

```
module.<module_name>.<module_exported_value>
```

An important point to be taken into account is the impact of using module on Terraform capability to managed dependance.
In a monolitic template, Terraform, on the plan step and then on the apply step, is able to build its own map of the resources dependencies. As an example, Terraform will know that it must have the resource group before deploying resource in the resource group.
In a template using module, this capability is somehow broken. To ensure the dependence management, a module relying on another resource created in the same template should take as input the resource created itself, through a module, instead of a simple string value provided through a variable.
Let's have a look at the vNet module:

```
module "SampleArchi_vNet" {

    #Module location
    #source = "./Modules/02 vNet"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//02
vNet/"

    #Module variable
    vNetName                = "SampleArchi_vNet"
    RGName                  = "${module.ResourceGroup.Name}"
    vNetLocation            = "${var.AzureRegion}"
    vNetAddressSpace        = "${var.vNetIPRange}"
    EnvironmentTag          = "${var.EnvironmentTag}"
    EnvironmentUsageTag     = "${var.EnvironmentUsageTag}"


}
```

We can see in this module the use of `"${module.ResourceGroup.Name}"` instead of the variable call used in the Resource Group module `name          = "${var.RGName}"`.
This way we ensure that Terraform understands that it need the same resource group created in the template to provision the vNet. Failure to do so would generate an error on the apply step, Terraform being too fast for Azure provider to create the Resource Group.
As suggested by the code, the use of Terraform interpolation to call a module output value is only possible if such output is defined in the module, as we already illustrate.

3.3 Used Terraform built-in function

3.3.1 The lookup(map,key,[default]) function

This function performs a dynamic lookup into a map variable. For example, we have declared the map variable Vmsize which can have different value, depending of the index associated. The lookup function

allows us to get the desired value of the map and to use it in the template. It is possible to define a default value, in case the key is not found. We defined our variable so that all the possible VM sizes would be associated to an index, so we did not define a default value.

3.3.2 The element(list,index) function

This function returns a specific value of a list at the index specified in input. We use this function when managing dependencies in module with count. For example, when we create a resource with count that needs another resource also created with a count, we can associate the appropriate value of the list to the parameter in the resource. Let's say that we create Data Disks with a count value of ${var.count}, meaning we create ${var.count} Data Disks.

```
resource "azurerm_managed_disk" "TerraManagedDiskwithcount" {


    count                   = "${var.Manageddiskcount}"
    name                    = "${var.ManageddiskName}${count.index+1}"
    location                = "${var.ManagedDiskLocation}"
    resource_group_name     = "${var.RGName}"
    storage_account_type    = "${var.StorageAccountType}"
    create_option           = "${var.CreateOption}"
    disk_size_gb            = "${var.DiskSizeInGB}"

    tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
  }

}
```

Following the creation of the DataDisks, we will create VMs, also with a count value of ${var.count}. To associate in the loop the corresponding DataDisk to the VM, we will use the element() function as follow.

```
resource "azurerm_virtual_machine" "TerraVMwithCount" {

    count                   = "${var.VMCount}"
    name                    = "${var.VMName}${count.index+1}"
    location                = "${var.VMLocation}"
    resource_group_name     = "${var.VMRG}"
    network_interface_ids   = ["${element(var.VMNICid, count.index)}"]
    vm_size                 = "${var.VMSize}"
    availability_set_id     = "${var.ASID}"
```

```
    boot_diagnostics {…
}

    storage_image_reference {…
    }

    storage_os_disk {…
    }

    storage_data_disk {

        name              = "${element(var.DataDiskName,count.index)}"
        managed_disk_id   = "${element(var.DataDiskId,count.index)}"
        create_option     = "Attach"
        lun               = 0
        disk_size_gb      = "${element(var.DataDiskSize,count.index)}"

    }
```

The code displays the use of the same function for the NIC association. However, since the NIC parameter takes a list, it is a little more complex and not a good example for now.

3.4 Other Terraform built-in capabilities used

Terraform can handle condition. The condition is usually applied on a parameter value, attributing one value or another depending of the condition. The syntaxe used for Terraform condition is as follow:

```
CONDITION ? TRUEVAL : FALSEVAL
```

In the template we will use the typical application of Terraform conditional, using a Boolean as a condition on a count value. If the Boolean is true, the count value will be the value assigned in input. If the Boolean is false, the count value will be 0, allowing the creation of a type of resource depending on the condition. The use case is for NIC creation on which a specific parameter is added if the NIC is in a load balancer back-end pool. Either it is load balanced, and thus the parameter is present in the configuration, or it is not load balanced and thus the parameter is absent. In the code we work around this configuration difference by using a Boolean variable true of false. If the Boolean is true, the NIC is I a load balancer and then it has a count value of the corresponding count variable required, if not the count is 0 and the resource is not created. WE then use another resource declaration with this time a count value to 0 if the Boolean is true and if it is false a count value to the corresponding count desired. The code is displayed below:

```
resource "azurerm_network_interface" "TerraNICnopipwithcountNotLoadBalanced" {
```

```
    count                   = "${var.IsLoadBalanced ? 0 : var.NICcount}"
    name                    = "${var.NICName}${count.index+1}"
    location                = "${var.NICLocation}"
    resource_group_name     = "${var.RGName}"

    ip_configuration {

        name                                        = "ConfigIP-
NIC${var.NICName}${count.index+1}"
        subnet_id                                   = "${var.SubnetId}"
        private_ip_address_allocation               = "dynamic"

            }

    tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
    }


}
```

In this sample, we use a resource with a clear name, TerraNICnopipwithcountNotLoadBalanced. The condition is applied on the count parameter. If the Boolean IsLoadBalancer is true, the resource, corresponding as the name implies to a NIC that is not specifically configured in a back-end pool, is assigned a count value equal to 0. The resource is then not created. If the Boolean is false, the resource is created and applied a count value equal to the variable NICcount.

In the following sample code, if the Boolean IsLoadBalanced is equal to true, the resource TerraNICnopipwithcountLoadBalanced has a count value equal to the value of the variable NICcount. If the Boolean is equal to false, the count value is equal to 0 and the resource is not created.

```
resource "azurerm_network_interface" "TerraNICnopipwithcountLoadBalanced" {

    count                   = "${var.IsLoadBalanced ? var.NICcount : 0}"
    name                    = "${var.NICName}${count.index+1}"
    location                = "${var.NICLocation}"
    resource_group_name     = "${var.RGName}"

    ip_configuration {

        name                                        = "ConfigIP-
NIC${var.NICName}${count.index+1}"
        subnet_id                                   = "${var.SubnetId}"
        private_ip_address_allocation               = "dynamic"
```

```
        load_balancer_backend_address_pools_ids     =
["${element(var.LBBackEndPoolid,count.index)}"]

            }

    tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
    }


}
```

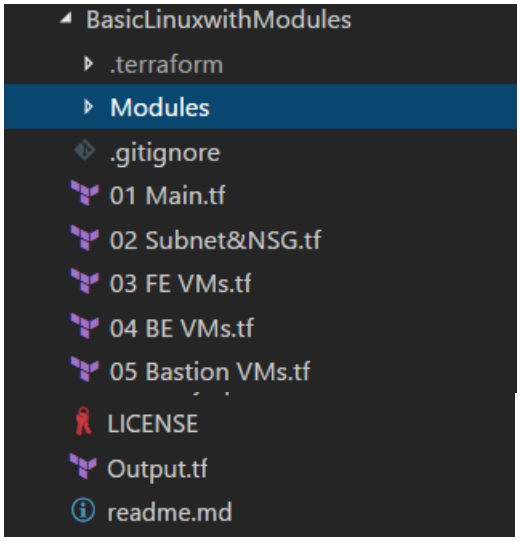
With this use of the conditional, we can use the same module for the NICs creation if those NICs are either loadbalanced or not.

4 Template exploration

In this part, we are going to explore the template structure.

4.1 Template files organization

While in the previous template, we wrote all the resource declaration in a single file, it can become easily very heavy to perform all the resource declaration in this way. Terraform, when the plan or apply command is launched, will take all the .tf files in the folder and compute the Terraform state, without any regard for the fact that configuration may be in different files. Thus, solely to ease the reading of the template, we have split the template in different files:



As displayed, we have a first file called main, which contains the main resources of the template, the Resource Group and the Network and storage resources. We also include in this file the provider configuration. In a second file, we declare all the subnets and NSG associated to the subnets. The logic here is to declare NSG resource only if they are relative to subnets. If we did have specific NSG applied only to VMs, we would declare the NSG in the associated .tf file.
After are the files for each VMs groups, respectively the Front-End VMs, the Back-End VMs and the Bastion VM. We also have a separate file for the output which contains value that we may need for further steps in the deployment pipeline.

4.2 The main file

AS discussed earlier, the main file contains the provider connections parameters and the primary resources declaration. No surprise for the provider configuration, we use variables declared in a separate file to avoid exposing the sensitive information.

```
# Configure the Microsoft Azure Provider with Azure provider variable defined in
AzureDFProvider.tf

provider "azurerm" {

    subscription_id = "${var.AzureSubscriptionID}"
    client_id       = "${var.AzureClientID}"
    client_secret   = "${var.AzureClientSecret}"
    tenant_id       = "${var.AzureTenantID}"
}
```

For the Resource Group creation, we call on a module. This module is declared with a name that must be unique in the template. It called the module content which is a .tf file located elsewhere. In our case, it is located on Github.
Other than the location of the module file, we have to provide with value for each of the parameters that are required for the resource group creation. This example is still quite simple; we have the following code for the module call:

```
module "ResourceGroup" {

    #Module Location
    #source = "./Modules/01 ResourceGroup"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//01
ResourceGroup/"

    #Module variable
    RGName                  = "${var.RGName}"
```

```
    RGLocation              = "${var.AzureRegion}"
    EnvironmentTag          = "${var.EnvironmentTag}"
    EnvironmentUsageTag     = "${var.EnvironmentUsageTag}"
}
```

A resource group requires in Terraform syntax a value for its name, the Azure region, called the location and optionnaly the Tags that we added in the module code.
Regading the location of the module file, we can see in comment the local path. We added this after pushing the module file to a Github repository.
The module file is coded as below:

```
#################################################################
#This module allow the creation of a RG
#################################################################


#Variable declaration for Module



variable "RGName" {
  type    = "string"
  default = "DefaultRG"
}


variable "RGLocation" {
  type    = "string"
  default = "Westeurope"
}


variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
}

variable "EnvironmentUsageTag" {
  type    = "string"
  default = "Poc usage only"
}

#Creating a Resource Group
resource "azurerm_resource_group" "Terra-RG" {
```

```
    name        = "${var.RGName}"
    location    = "${var.RGLocation}"

    tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
    }

}

#Output for the RG module

output "Name" {

  value = "${azurerm_resource_group.Terra-RG.name}"
}

output "Location" {

  value = "${azurerm_resource_group.Terra-RG.location}"
}

output "Id" {

  value = "${azurerm_resource_group.Terra-RG.id}"
}
```

We can identify 3 sections in the code. The first is the variable declaration, which translate all the variable that are specified when calling the module. The second is the resource declaration itself. This is completely similar to a classical Resource Group declaration in Terraform. Last is the output section which allow us to export values outside of the module and to use those values either in other modules or in the template outputs. The exported values are documented in the Terraform documentation. Note thatis is also possible to export parameters declared in input from the module. While this might seem strange to do so instead of using the input value itself, it is useful to ensure that following object are interpreted as depending of the resource created in the module.
In our case, as one can expect, nearly all Azure resource created will use as a parameter a value containing the Resource Group name. While in a monolithic template it would be possible to use the Resource Group name as a global variable, doing so would somehow break Terraform intelligence and be followed on the apply step, Terraform being unable to find the Resource Group object.

4.3 The Subnet and NSG file

In this file are created the subnet and their associated NSG. Meaning the NSG that will be applied to the NICs located in those subnet, if no other NSG are applied directly on the NICs. As reminded in the Architecture description section, we will have for this template the following:

- 1 Front-End subnet
- 1 Back-End subnet
- 1 bastion subnet
- 1 Network Security Group associated to each subnet

For this, we will use 2 differents modules, one for the subnet and another for the Network Security Group. Since the Subnet object requires a NSG id if a NSG is effectively associated at his level, we declare first the call for the NSG. Again, it is solely for a human logic point of view. Terraform could create the object even if the declaration happened after.

4.3.1 NSG module

The module for the NSG on the Front-End subnet is called as follow:

```
module "NSG_FE_Subnet" {

    #Module location
    #source = "./Modules/07 NSG"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//07 NSG/"

    #Module variable
    NSGName                 = "NSG_${lookup(var.SubnetName, 0)}"
    RGName                  = "${module.ResourceGroup.Name}"
    NSGLocation             = "${var.AzureRegion}"
    EnvironmentTag          = "${var.EnvironmentTag}"
    EnvironmentUsageTag     = "${var.EnvironmentUsageTag}"


}
```

It is interesting to take note of the value for the Resource Group, here called RGName in relation to the declared variable in the module file. As discussed earlier, it uses the output data from the Resource Group module and thus we can see the value ${module.ResourceGroup.name}. Terraform interpolation is used here and we should see this kind of syntax each time a value exported from amodule is used in another module call.
The code for the module is displayed below:

```
#################################################################
#This module allow the creation of a Netsork Security Group
#################################################################
```

```
#Variable declaration for Module

variable "NSGName" {
  type    = "string"
  default = "DefaultNSG"
}



variable "RGName" {
  type    = "string"
  default = "DefaultRSG"
}

variable "NSGLocation" {
  type    = "string"
  default = "Westeurope"
}

variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
}

variable "EnvironmentUsageTag" {
  type    = "string"
  default = "Poc usage only"
}

#Creation of the NSG
resource "azurerm_network_security_group" "Terra-NSG" {

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


    tags {
        environment = "${var.EnvironmentTag}"
        usage       = "${var.EnvironmentUsageTag}"
    }
}


#Output for the NSG module
```

```
output "Name" {

  value = "${azurerm_network_security_group.Terra-NSG.name}"
}

output "Id" {

  value ="${azurerm_network_security_group.Terra-NSG.id}"
}
```

Again 3 sections, one for the variables, one for the resource creation and one for the output. This kind of module structure is ideal if one is looking for simplicity in the module use. Each time one type of resources is created in a module. An interesting fact to be noted is that a Terraform object is not necessarily an object in Azure. It may be encapsulated in an Azure object. In those case, the module structure may be required more complexity with more than one kind of resource declared. More on that in the VMs files section.
For the output, we export the name and the id. The name is mainly used for the global output while we will see that the id is required for the subnet creation. To distinguish the value exported fffrom the module from the one exported from the Terraform object, it can help to have a different syntax type. We can see here that a capital letter was used for the output Name and Id. This point an important aspect: Terraform is case sensitive. A variable declared with capital and associated in the module call without the proprer syntax will result in an error while executing the plan step.

4.3.2 Subnet module

For this module, let's start with the module code:

```
#Creation fo the subnet

resource "azurerm_subnet" "TerraSubnet" {


    name                        = "${var.SubnetName}"
    resource_group_name         = "${var.RGName}"
    virtual_network_name        = "${var.vNetName}"
    address_prefix              = "${var.Subnetaddressprefix}"
    network_security_group_id   = "${var.NSGid}"


}
```

Only the section with the resource creation is displayed here. WE have the following parameters:
Name

- Resource_group_name

- Virtual_network_name
- Address_prefix
- Network_security_group_id

Which are associated with the variables:

- SubnetName
- RGName
- vNetName
- Subnetaddressprefix
- NSGid

Let's now have a look at the module call:

```
#FE-PRD_Subnet

module "FE_Subnet" {

    #Module location
    #source = "./Modules/06 Subnet"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//06
Subnet"

    #Module variable
    SubnetName                  = "${lookup(var.SubnetName, 0)}"
    RGName                      = "${module.ResourceGroup.Name}"
    vNetName                    = "${module.SampleArchi_vNet.Name}"
    Subnetaddressprefix         = "${lookup(var.SubnetAddressRange, 0)}"
    NSGid                       = "${module.NSG_FE_Subnet.Id}"
    EnvironmentTag              = "${var.EnvironmentTag}"
    EnvironmentUsageTag         = "${var.EnvironmentUsageTag}"

}
```

The code shows each variable assigned with a value. Here enter the lookup function which call upon the map variables. For example, the subnetaddressprefix variable use the lookup function to call the map variable SubnetAddressRange and the first value of the map, as displayed here:

```
variable "SubnetAddressRange" {
#Note: Subnet must be in range included in the vNET Range

    default = {
      "0" = "10.0.0.0/24"
      "1" = "10.0.1.0/24"
      "2" = "10.0.2.0/24"

    }
```

```
}
```

By reading this we can see that the ip range for the Front-End subnet is 10.0.0.0/24.
WE can also take note of the use of 2 module values output for the RGName variable, the vNetName variable and the NSGid variable. All others values here are taken from the template variable file.

4.4 The VMs files

In this section, we will look at the creation of the Front-End VMs, which includes the most Azure Resource, due to the postion behind an Azure Load Balancer.

In the FE VMs files, we create the following:

- The NSG rules associated to the NSG on which the VMs' NICs depends on
- The Public IP Address associated to the Internet Facing Load Balancer
- The Load balancer itself
- The VMs' NICs
- The VMs' Managed Data disks
- The VMs

4.4.1 NSG rules

For the NSG rules, we make use of a module which is coded in a way that allows us to use it for any rule that we may need. By going this way, we have a very generic module that we can reuse many time. The counterpart of this generic module is that many parameters need to be specified in the module code. This mean that, in the module, the variables declaration section is pretty big and that the call itself may have more fixed values than other module. As much as possible, we make use pf the variable file or others modules outputs to rely on Terraform interpolation rather than hardcoded value. However, we will see that we still have to specify some parameters, such as the rule direction, or the ports. The module call is displayed below for the rule allowing incoming http traffic:

```
module "AllowHTTPFromInternetFEIn" {

    #Module source
    #source = "./Modules/08 NSGRule"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//08
NSGRule"

    #Module variable
    RGName = "${module.ResourceGroup.Name}"
    NSGReference = "${module.NSG_FE_Subnet.Name}"
    NSGRuleName = "AllowHTTPFromInternetFEIn"
    NSGRulePriority = 101
```

```
    NSGRuleDirection = "Inbound"
    NSGRuleAccess = "Allow"
    NSGRuleProtocol = "Tcp"
    NSGRuleSourcePortRange = "*"
    NSGRuleDestinationPortRange = 80
    NSGRuleSourceAddressPrefix = "Internet"
    NSGRuleDestinationAddressPrefix = "${lookup(var.SubnetAddressRange, 0)}"
}
```

As explained before, we have some reference to the variable file, as in the NSGDestinationAddressPrefix parameter, or module output, as in the RGName and the NSGReference parameters.
Another way to approach the NSG rules creation would be to have multiple NSG rule module defining each time a kind of allowed traffic. In this case, we could limit ourself in the call to the reference to other module and to the NSGRulePriority which could be included in a map variable in the variables file.
The code for the module itself is displayed below:

```
#################################################################
#This module allow the creation of a Netsork Security Group Rule
#################################################################

#Variable declaration for Module


# The NSG rule requires a RG location in which the NSG for which the rule is
created is located
variable "RGName" {
  type    = "string"
  default = "DefaultRSG"
}

#The NSG rule requires a reference to a NSG
variable "NSGReference" {
  type    = "string"

}

#The NSG Rule Name, a string value allowing to identify the rule after deployment
variable "NSGRuleName" {
  type    = "string"
  default = "DefaultNSGRule"
}


#The NSG rule priority is an integer value defining the priority in which the rule
is applyed in the NSG
```

```
variable "NSGRulePriority" {
  type    = "string"

}

#The NSG rule direction define if the rule is for ingress or egress trafic. Allowed
value are inbound and outbound
variable "NSGRuleDirection" {
  type    = "string"

}

#The NSG Rule Access value,  a string value defining if the rule allow or block the
specified traffic. Accepted value are Allow or Block
variable "NSGRuleAccess" {
  type    = "string"
  default = "Allow"
}

#The NSG rule protocol define which type of trafic to allow/block. It accept the
string tcp, udp, icmp or *
variable "NSGRuleProtocol" {
  type    = "string"

}

#The NSG rule source port range define the port(s) from which the trafic origing is
allowed/blocked
variable "NSGRuleSourcePortRange" {
  type    = "string"

}

#The NSG rule destination port range define the port(s) on which the trafic
destination is allowed/blocked
variable "NSGRuleDestinationPortRange" {
  type    = "string"

}

#The NSG rule address preifx defines the source address(es) from whichthe trafic
origin is allowed/blocked
variable "NSGRuleSourceAddressPrefix" {
  type    = "string"
```

```
}

#The NSG rule address preifx defines the source address(es) from whichthe trafic
origin is allowed/blocked
variable "NSGRuleDestinationAddressPrefix" {
  type    = "string"

}
 # creation of the rule

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        = "${var.NSGRuleDestinationPortRange}"
    source_address_prefix         = "${var.NSGRuleSourceAddressPrefix}"
    destination_address_prefix    = "${var.NSGRuleDestinationAddressPrefix}"
    resource_group_name           = "${var.RGName}"
    network_security_group_name   = "${var.NSGReference}"

}


# Module output


output "Name" {

  value = "${azurerm_network_security_rule.Terra-NSGRule.name}"
}

output "Id" {

  value ="${azurerm_network_security_rule.Terra-NSGRule.id}"
}
```

The output for this module are mainly for potential global output rather than a usefulness in another module call. In any case, the outputs still exist and are thus exploitable if the need arise.

4.4.2 Public IP Creation

After the NSG rules creation, the next required resource is the public IP for the load balancer. This module quite simple, so is the module call:

```
#Azure Load Balancer public IP Creation

module "LBWebPublicIP" {

    #Module source
    #source = "./Modules/10 PublicIP"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//10
PublicIP"


    #Module variables
    PublicIPCount           = "1"
    PublicIPName            = "lbwebpip"
    PublicIPLocation        = "${var.AzureRegion}"
    RGName                  = "${module.ResourceGroup.Name}"
    EnvironmentTag          = "${var.EnvironmentTag}"
    EnvironmentUsageTag     = "${var.EnvironmentUsageTag}"


}
```

The module code itself is displayed below:

```
########################################################################
# This module allow the creation of a Public IP Address
########################################################################

#Module variables

#Public IP Count

variable "PublicIPCount" {
  type  = "string"
  default = "1"

}
#The Public IP Name

variable "PublicIPName" {
  type    = "string"
```

```
}

#The Public IP Location

variable "PublicIPLocation" {
  type    = "string"

}

#The RG in which the Public IP resides

variable "RGName" {
  type    = "string"

}

variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
}

variable "EnvironmentUsageTag" {
  type    = "string"
  default = "Poc usage only"
}

# Creating Public IP

resource "random_string" "PublicIPfqdnprefix" {



    length = 5
    special = false
    upper = false
    number = false
}

resource "azurerm_public_ip" "TerraPublicIP" {


    count                        = "${var.PublicIPCount}"
    name                         = "${var.PublicIPName}"
    location                     = "${var.PublicIPLocation}"
```

```
    resource_group_name             = "${var.RGName}"
    public_ip_address_allocation    = "static"
    domain_name_label               =
"${random_string.PublicIPfqdnprefix.result}${var.PublicIPName}"

    tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
    }

}

#Module output

output "Names" {

  value = ["${azurerm_public_ip.TerraPublicIP.*.name}"]
}

output "Ids" {

  value = ["${azurerm_public_ip.TerraPublicIP.*.id}"]
}

output "IPAddresses" {

  value = ["${azurerm_public_ip.TerraPublicIP.*.ip_address}"]
}

output "fqdns" {

  value = ["${azurerm_public_ip.TerraPublicIP.*.fqdn}"]
}
```

To be noted, the use of the count parameter with a default value of 1. In this case, we can use the same module for either a single IP creation of for multiple Public IP creation. This choice has an impact on the output format which have to be in the list form. The other point to be noted is the use of the random provider which result's is added to the domain_name_label parameter, equivalent to the fqdn of the public IP. The default fqdn use Microsoft DNS domain and requires unicity over all the Azure infrastructure in a chosen region.

4.4.3 The load balancer

The load balancer is used to allow access to the Front-End servers on the TCP 80 port. It is composed in Terraform semantic of a load balancer resource, a back-end address pool, a health probe, load balancer rule(s) and potentially NAT rules. However we do not use NAT rule in the template or in the module. The module is coded to declare all the preceding resource in 1 module, thus simplifying the module call. While it is still possible to use a more granular approach by splitting each of the resource in different modules, it was not the choice here. The code for the module is displayed below:

```
#########################################################################
# This module creates an external load balancer
#########################################################################

#Module variables

variable "LBCount" {
    type    = "string"
    default = "1"
}

variable "ExtLBName" {
    type = "string"

}

variable "AzureRegion" {
    type = "string"

}

variable "RGName" {
    type = "string"

}

variable "FEConfigName" {
    type = "string"

}

variable "PublicIPId" {
    type = "list"

}
```

```
variable "LBBackEndPoolName" {
    type = "string"
}

variable "LBProbeName" {
    type = "string"
}

variable "LBProbePort" {
    type = "string"
}

variable "FERuleName" {
    type = "string"
}

variable "FERuleProtocol" {
    type = "string"
}

variable "FERuleFEPort" {
    type = "string"
}

variable "FERuleBEPort" {
    type = "string"

}

variable "TagEnvironment" {
    type = "string"
}

variable "TagUsage" {
    type = "string"
}
# Creating Azure Load Balancer for front end http / https


resource "azurerm_lb" "TerraExtLB" {

    count                       = "${var.LBCount}"
    name                        = "${var.ExtLBName}${count.index+1}"
    location                    = "${var.AzureRegion}"
```

```
    resource_group_name         = "${var.RGName}"

    frontend_ip_configuration {

        name                    = "${var.FEConfigName}"
        public_ip_address_id    = "${element(var.PublicIPId,count.index)}"
    }

    tags {
    environment = "${var.TagEnvironment}"
    usage       = "${var.TagUsage}"
    }
}


# Creating Back-End Address Pool

resource "azurerm_lb_backend_address_pool" "TerraLBBackEndPool" {

    count               = "${var.LBCount}"
    name                = "${var.LBBackEndPoolName}${count.index+1}"
    resource_group_name = "${var.RGName}"
    loadbalancer_id     = "${element(azurerm_lb.TerraExtLB.*.id,count.index)}"

}


# Creating Health Probe

resource "azurerm_lb_probe" "TerraLBProbe" {

    count               = "${var.LBCount}"
    name                = "${var.LBProbeName}"
    resource_group_name = "${var.RGName}"
    loadbalancer_id     = "${element(azurerm_lb.TerraExtLB.*.id,count.index)}"
    port                = "${var.LBProbePort}"


}

# Creating Load Balancer rules

resource "azurerm_lb_rule" "TerraLBFrondEndrule" {

    count                          = "${var.LBCount}"
```

```
    name                            = "${var.FERuleName}"
    resource_group_name             = "${var.RGName}"
    loadbalancer_id                 =
"${element(azurerm_lb.TerraExtLB.*.id,count.index)}"
    protocol                        = "${var.FERuleProtocol}"
    probe_id                        =
"${element(azurerm_lb_probe.TerraLBProbe.*.id,count.index)}"
    frontend_port                   = "${var.FERuleFEPort}"
    frontend_ip_configuration_name  = "${var.FEConfigName}"
    backend_port                    = "${var.FERuleBEPort}"
    backend_address_pool_id         =
"${element(azurerm_lb_backend_address_pool.TerraLBBackEndPool.*.id,count.index)}"


}


output "LBBackendPoolIds" {

    value = ["${azurerm_lb_backend_address_pool.TerraLBBackEndPool.*.id}"]
}
```

As one can see, we used the count parameter which allows us to create more than one load balancer with the same module. In this template it is not used. We may rewrite this module at a later time since it is not clear if this functionality is very useful for the load balancer use case. We make use of the element() function to call reference of azure load balancer related resource at different step of the code. The module call is displayed below:

```
module "LBWebFE" {

    #Module source
    #source = "./Modules/15 External LB"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//15
External LB"


    #Module variables
    LBCount             = "1"
    ExtLBName           = "LBWebFE"
    AzureRegion         = "${var.AzureRegion}"
    RGName              = "${module.ResourceGroup.Name}"
    FEConfigName        = "LBWebFEConfig"
    PublicIPId          = ["${module.LBWebPublicIP.Ids}"]
    LBBackEndPoolName   = "LBWebFE_BEPool"
    LBProbeName         = "LBWebFE_Probe"
```

```
    LBProbePort         = "80"
    FERuleName          = "LBWebFEHTTPRule"
    FERuleProtocol      = "tcp"
    FERuleFEPort        = "80"
    FERuleBEPort        = "80"
    TagEnvironment      = "${var.EnvironmentTag}"
    TagUsage            = "${var.EnvironmentUsageTag}"


}
```

4.4.4 The availability set

The availability set is another example of a quite simple module. This comes without surprise since the resource is quite simple to create. For ease of reference in other modules, we did not use the count parameter in this module. The code for the module is displayed below:

```
#################################################################
#This module allow the creation of an availability set for VMs
#################################################################

#Variable declaration for Module

#The AS name
variable "ASName" {
  type    = "string"

}

#The RG in which the AS is attached to
variable "RGName" {
  type    = "string"

}

#The location in which the AS is attached to
variable "ASLocation" {
  type    = "string"

}
```

```
#Tag value to help identify the resource.
#Required tag are EnvironmentTAg defining the type of
#environment and
#environment Tag usage specifying the use case of the environment

variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
}

variable "EnvironmentUsageTag" {
  type    = "string"
  default = "Poc usage only"
}



# Availability Set Creation

resource "azurerm_availability_set" "Terra-AS" {

    name                    = "${var.ASName}"
    location                = "${var.ASLocation}"
    managed                 = "true"
    resource_group_name     = "${var.RGName}"
    tags {
        environment = "${var.EnvironmentTag}"
        usage       = "${var.EnvironmentUsageTag}"
  }
}

#Output


output "Name" {

  value = "${azurerm_availability_set.Terra-AS.name}"
}

output "Id" {

  value = "${azurerm_availability_set.Terra-AS.id}"
}
```

As we discussed in a previous article, the availability set is not used in Terraform as the reference for the backend address pool. Which is also why it is not created before the load balancer. The reference to the back end pool is at the VMs' NICs level and we will have a look at how we addressed it in the NICs section. The module call is displayed below. In this case, only the Availabiity Set name is specified while all the others parameters make use of the variable files and other module output.

```
#Availability set creation


module "AS_FEWEB" {

    #Module source

    #source = "./Modules/13 AvailabilitySet"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//13
AvailabilitySet"


    #Module variables
    ASName                  = "AS_FEWEB"
    RGName                  = "${module.ResourceGroup.Name}"
    ASLocation              = "${var.AzureRegion}"
    EnvironmentTag          = "${var.EnvironmentTag}"
    EnvironmentUsageTag     = "${var.EnvironmentUsageTag}"

}
```

4.4.5 The VM's NICs

We made the choice for the NICs creation to use a module with the count parameter to alow the cration of groups of NICs. Also, since it may be behind a load balancer, we made use of the conditional capabilities of Terraform as explained in 3.4 to build only one module with the choice for the resource created depending on a Boolean called isloadbalanced. Either we have a load balanced NIC(s) and thus we apply a value to the count parameter inside the module to the count parameter in input, or it is not load balanced, and the boolean has a value of false and the resource count is set to 0. We have another resource corresponding to the NIC(s) not load balanced, using the same Boolean. If the Boolean is equal to true, this time the value for the count parameter is set to 0 and thus we do not create the resource. Only the resource for the load balanced NIC(s) is created. If in contrary, we have a valueof the Boolean set to false, we do apply the value to the count parameter with the input for the nic count.
The module call is displayed below, followed by the module code:

```
#NIC Creation
```

```
module "NICs_FEWEB" {

    #module source

    #source = "./Modules/09 NICWithoutPIPWithCount"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//09
NICWithoutPIPWithCount"


    #Module variables

    NICcount            = "3"
    NICName             = "NIC_FEWEB"
    NICLocation         = "${var.AzureRegion}"
    RGName              = "${module.ResourceGroup.Name}"
    SubnetId            = "${module.FE_Subnet.Id}"
    IsLoadBalanced      = "1"
    LBBackEndPoolid     = ["${module.LBWebFE.LBBackendPoolIds}"]
    EnvironmentTag      = "${var.EnvironmentTag}"
    EnvironmentUsageTag = "${var.EnvironmentUsageTag}"


}
```

```
#####################################################################
#This module allow the creation of VMs NICs
#####################################################################

#Variables for NIC creation

#The count value
variable "NICcount" {
  type    = "string"

}

#The NIC name
variable "NICName" {
  type    = "string"

}


#The NIC location
```

```
variable "NICLocation" {
  type    = "string"

}

#The resource Group in which the NIC are attached to
variable "RGName" {
  type    = "string"

}

#The subnet reference
variable "SubnetId" {
  type    = "string"

}

#Behind a lb
variable "IsLoadBalanced" {
  type    = "string"
  default = "0"

}

#BackendPool id

variable "LBBackEndPoolid" {
  type = "list"
  default  = ["defaultPool1","defaultPool1"]



}

variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
}

variable "EnvironmentUsageTag" {
  type    = "string"
  default = "Poc usage only"
}
```

```
# NIC Creation

resource "azurerm_network_interface" "TerraNICnopipwithcountLoadBalanced" {

    count                   = "${var.IsLoadBalanced ? var.NICcount : 0}"
    name                    = "${var.NICName}${count.index+1}"
    location                = "${var.NICLocation}"
    resource_group_name     = "${var.RGName}"

    ip_configuration {

        name                                        = "ConfigIP-
NIC${var.NICName}${count.index+1}"
        subnet_id                                   = "${var.SubnetId}"
        private_ip_address_allocation               = "dynamic"
        load_balancer_backend_address_pools_ids     =
["${element(var.LBBackEndPoolid,count.index)}"]

            }

    tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
    }


}

resource "azurerm_network_interface" "TerraNICnopipwithcountNotLoadBalanced" {

    count                   = "${var.IsLoadBalanced ? 0 : var.NICcount}"
    name                    = "${var.NICName}${count.index+1}"
    location                = "${var.NICLocation}"
    resource_group_name     = "${var.RGName}"

    ip_configuration {

        name                                        = "ConfigIP-
NIC${var.NICName}${count.index+1}"
        subnet_id                                   = "${var.SubnetId}"
        private_ip_address_allocation               = "dynamic"

            }
```

```
    tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
    }


}

output "LBNames" {

  value =
["${azurerm_network_interface.TerraNICnopipwithcountLoadBalanced.*.name}"]
}

output "LBIds" {

  value = ["${azurerm_network_interface.TerraNICnopipwithcountLoadBalanced.*.id}"]
}

output "LBPrivateIPs" {

  value =
["${azurerm_network_interface.TerraNICnopipwithcountLoadBalanced.*.private_ip_addre
ss}"]
}

output "Names" {

  value =
["${azurerm_network_interface.TerraNICnopipwithcountLoadBalanced.*.name}"]
}

output "Ids" {

  value =
["${azurerm_network_interface.TerraNICnopipwithcountNotLoadBalanced.*.id}"]
}

output "PrivateIPs" {

  value =
["${azurerm_network_interface.TerraNICnopipwithcountNotLoadBalanced.*.private_ip_ad
dress}"]
}
```

The outputs are critical here since we use the exported value(s) in the VMs module.

4.4.6 The managed disk(s)

The managed disks are created with a module in which the ocunt parameter value is provided as an input. As for the NICs module, we specify the count value with the corresponding need Front-End servers. We make use of variables and module output for the most of the input. The CreateOption is defined here as Empty. This parameter is worthy of note since it requires a re-creation of the resource when the template is applied more than once. Prudence is so required in case Datas are indeed stored on the disks.

```
#Datadisk creation

module "DataDisks_FEWEB" {

    #Module source

    #source = "./Modules/06 ManagedDiskswithcount"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//06
ManagedDiskswithcount"


    #Module variables

    Manageddiskcount    = "3"
    ManageddiskName     = "DataDisk_FEWEB"
    RGName              = "${module.ResourceGroup.Name}"
    ManagedDiskLocation = "${var.AzureRegion}"
    StorageAccountType  = "${lookup(var.Manageddiskstoragetier, 0)}"
    CreateOption        = "Empty"
    DiskSizeInGB        = "63"
    EnvironmentTag      = "${var.EnvironmentTag}"
    EnvironmentUsageTag = "${var.EnvironmentUsageTag}"


}
```

```
#####################################################################
#This module allow the creation of a Managed disk with count option
#####################################################################

#Variable declaration for Module
```

```
#The count value
variable "Manageddiskcount" {
  type    = "string"

}


#The Managed Disk name
variable "ManageddiskName" {
  type    = "string"

}

#The RG in which the MD is attached to
variable "RGName" {
  type    = "string"

}

#The location in which the MD is attached to
variable "ManagedDiskLocation" {
  type    = "string"

}

#The underlying Storage account type. Value accepted are Standard_LRS and
Premium_LRS
variable "StorageAccountType" {
  type    = "string"

}
#The create option. Value accepted
#Import - Import a VHD file in to the managed disk (VHD specified with source_uri).
#Empty - Create an empty managed disk.
#Copy - Copy an existing managed disk or snapshot (specified with
source_resource_id).

variable "CreateOption" {
  type    = "string"

}

# Specifies the size of the managed disk to create in gigabytes.
```

```
#If create_option is Copy, then the value must be equal
#to or greater than the source's size.
#Take also into account the pricing related to the size:
#129 GB equal 512 Provisioned so prefer
#corresponding value to storage tiers desired

variable "DiskSizeInGB" {
  type    = "string"

}


variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
}

variable "EnvironmentUsageTag" {
  type    = "string"
  default = "Poc usage only"
}




#ManagedDisk creation

resource "azurerm_managed_disk" "TerraManagedDiskwithcount" {


    count                = "${var.Manageddiskcount}"
    name                 = "${var.ManageddiskName}${count.index+1}"
    location             = "${var.ManagedDiskLocation}"
    resource_group_name  = "${var.RGName}"
    storage_account_type = "${var.StorageAccountType}"
    create_option        = "${var.CreateOption}"
    disk_size_gb         = "${var.DiskSizeInGB}"

    tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
  }

}
```

```
#Output

output "Names" {

  value = ["${azurerm_managed_disk.TerraManagedDiskwithcount.*.name}"]
}

output "Ids" {

  value = ["${azurerm_managed_disk.TerraManagedDiskwithcount.*.id}"]
}

output "Sizes" {

  value = ["${azurerm_managed_disk.TerraManagedDiskwithcount.*.disk_size_gb}"]
}
```

4.4.7 VMs resource creation

For this resource creation, we use a module which refers to the NICs module and also the managed disks module:

```
#VM creation

module "VMs_FEWEB" {

    #module source

    #source = "./Modules/14 LinuxVMWithCount"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//14
LinuxVMWithCount"


    #Module variables

    VMCount                     = "3"
    VMName                      = "WEB-FE"
    VMLocation                  = "${var.AzureRegion}"
    VMRG                        = "${module.ResourceGroup.Name}"
    VMNICid                     = ["${module.NICs_FEWEB.LBIds}"]
    VMSize                      = "${lookup(var.VMSize, 0)}"
```

```
    ASID                        = "${module.AS_FEWEB.Id}"
    VMStorageTier               = "${lookup(var.Manageddiskstoragetier, 0)}"
    VMAdminName                 = "${var.VMAdminName}"
    VMAdminPassword             = "${var.VMAdminPassword}"
    DataDiskId                  = ["${module.DataDisks_FEWEB.Ids}"]
    DataDiskName                = ["${module.DataDisks_FEWEB.Names}"]
    DataDiskSize                = ["${module.DataDisks_FEWEB.Sizes}"]
    VMPublisherName             = "${lookup(var.PublisherName, 4)}"
    VMOffer                     = "${lookup(var.Offer, 4)}"
    VMsku                       = "${lookup(var.sku, 4)}"
    DiagnosticDiskURI           = "${module.DiagStorageAccount.PrimaryBlobEP}"
    BootConfigScriptFileName    = "installapache.sh"
    PublicSSHKey                = "${var.AzurePublicSSHKey}"
    EnvironmentTag              = "${var.EnvironmentTag}"
    EnvironmentUsageTag         = "${var.EnvironmentUsageTag}"

}
```

The BootConfigScriptFileName is not used at this point. It point to a local file which is not used in this configuration as a bootstrap script. To make use of a script file we would need to create an additional module for the custom script extension, with a dedicated reference to the script file that is needed. Since it is not very flexible in terms of re-use, we looked first at a bootstrap capability. In the full template, we created a custom script extension in a module to install Ansible. We will then use Ansible at a later step to deploy the required middleware from the Bastion server. For Windows VMs, the BootConfigScriptFileName allows to associate a specific file and to copy it onto the VM's disk. Specifying in a custom script the renaming of the file cpoied could ease the basic middleware config and thus allow us the use of a generic module for the VMs creation.

```
##############################################################################################
#This module allow the creation of n Linux VM with 1 NIC
##############################################################################################

#Variable declaration for Module

#The VM count
variable "VMCount" {
  type    = "string"

}

#The VM name
variable "VMName" {
  type    = "string"

}
```

```
#The VM location
variable "VMLocation" {
  type    = "string"

}

#The RG in which the VMs are located
variable "VMRG" {
  type    = "string"

}

#The NIC to associate to the VM
variable "VMNICid" {
  type    = "list"

}

#The VM size
variable "VMSize" {
  type    = "string"
  default = "Standard_F1"

}


#The Availability set reference

variable "ASID" {
  type    = "string"

}

#The Managed Disk Storage tier

variable "VMStorageTier" {
  type    = "string"
  default = "Premium_LRS"

}

#The VM Admin Name
```

```
variable "VMAdminName" {
  type    = "string"
  default = "VMAdmin"

}

#The VM Admin Password

variable "VMAdminPassword" {
  type    = "string"

}

# Managed Data Disk reference

variable "DataDiskId" {
  type    = "list"


}

# Managed Data Disk Name

variable "DataDiskName" {
  type    = "list"


}

# Managed Data Disk size

variable "DataDiskSize" {
  type    = "list"

}

# VM images info
#get appropriate image info with the following command
#Get-AzureRMVMImagePublisher -location WestEurope
#Get-AzureRMVMImageOffer -location WestEurope -PublisherName <PublisherName>
#Get-AzureRmVMImageSku -Location westeurope -Offer <OfferName> -PublisherName
<PublisherName>

variable "VMPublisherName" {
```

```
  type    = "string"

}


variable "VMOffer" {
  type    = "string"

}

variable "VMsku" {
  type    = "string"

}

#The boot diagnostic storage uri

variable "DiagnosticDiskURI" {
  type    = "string"

}

#The boot config file name

variable "BootConfigScriptFileName" {

  type    = "string"

}


variable "PublicSSHKey" {

  type = "string"
}

#Tag info

variable "EnvironmentTag" {
  type    = "string"
  default = "Poc"
}

variable "EnvironmentUsageTag" {
  type    = "string"
```

```
  default = "Poc usage only"
}



#VM Creation


resource "azurerm_virtual_machine" "TerraVMwithCount" {

    count                   = "${var.VMCount}"
    name                    = "${var.VMName}${count.index+1}"
    location                = "${var.VMLocation}"
    resource_group_name     = "${var.VMRG}"
    network_interface_ids   = ["${element(var.VMNICid, count.index)}"]
    vm_size                 = "${var.VMSize}"
    availability_set_id     = "${var.ASID}"

    boot_diagnostics {

      enabled = "true"
      storage_uri = "${var.DiagnosticDiskURI}"

    }


    storage_image_reference {
        #get appropriate image info with the following command
        #Get-AzureRmVMImageSku -Location westeurope -Offer windowsserver -
PublisherName microsoftwindowsserver
        publisher   = "${var.VMPublisherName}"
        offer       = "${var.VMOffer}"
        sku         = "${var.VMsku}"
        version     = "latest"

    }

    storage_os_disk {

        name                = "${var.VMName}${count.index+1}-OSDisk"
        caching             = "ReadWrite"
        create_option       = "FromImage"
        managed_disk_type   = "${var.VMStorageTier}"

    }
```

```
    storage_data_disk {

        name                = "${element(var.DataDiskName,count.index)}"
        managed_disk_id     = "${element(var.DataDiskId,count.index)}"
        create_option       = "Attach"
        lun                 = 0
        disk_size_gb        = "${element(var.DataDiskSize,count.index)}"


    }

    os_profile {

        computer_name   = "${var.VMName}"
        admin_username  = "${var.VMAdminName}"
        admin_password  = "${var.VMAdminPassword}"
        custom_data     = "${file("${var.BootConfigScriptFileName}")}"

    }

    os_profile_linux_config {

    disable_password_authentication = true

    ssh_keys {
      path     = "/home/${var.VMAdminName}/.ssh/authorized_keys"
      key_data = "${var.PublicSSHKey}"
    }

    }
    tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
    }


}
```

4.4.8 Custom VMs' agents

As we just discussed, specific VM agent can be used to add functionalities to the VM or for provisioning purpose. The front-end VMs are configured with the Network watcher agent. We will look into the Network Watcher functionalities in a later article. For now, let's have a look at how this agent is added to the VM.
We use a dedicated module, which takes in input the Front-End VMs name as a list, and the variable AzureRegion, the output of the resource group module and Tags defined in the variable file. The code for calling the module is as follow:

```
#Network Watcher Agent

module "NetworkWatcherAgentForFEWeb" {

    #Module Location
    #source = "./Modules/20 LinuxNetworkWatcherAgent"
    source = "github.com/dfrappart/Terra-AZBasiclinuxWithModules//Modules//20
LinuxNetworkWatcherAgent"


    #Module variables
    AgentCount              = "3"
    AgentName               = "NetworkWatcherAgentForFEWeb"
    AgentLocation           = "${var.AzureRegion}"
    AgentRG                 = "${module.ResourceGroup.Name}"
    VMName                  = ["${module.VMs_FEWEB.Name}"]
    EnvironmentTag          = "${var.EnvironmentTag}"
    EnvironmentUsageTag     = "${var.EnvironmentUsageTag}"
}
```

Required parameters for the VM's extension are the publisher, the type and the type handler version. To get the information on the agent, we use a PowerShell command as follow:

```
Get-AzureRmVMExtensionImageType -Location westeurope -PublisherName
microsoft.azure.networkwatcher


PublisherName : microsoft.azure.networkwatcher
Type          : NetworkWatcherAgentLinux
Id            : /Subscriptions/f1f020d0-0fa6-4d01-b816-
5ec60497e851/Providers/Microsoft.Compute/Locations/westeurope/Publishers/microsoft.
azure.networkwatcher/ArtifactTypes/VMExtension/Types/NetworkWatcherAgentLinux
Location      : westeurope
Name          :
RequestId     : e4e5ee24-2ae6-44d3-8e6c-428547661564
StatusCode    : OK
```

```
PublisherName : microsoft.azure.networkwatcher
Type          : NetworkWatcherAgentWindows
Id            : /Subscriptions/f1f020d0-0fa6-4d01-b816-
5ec60497e851/Providers/Microsoft.Compute/Locations/westeurope/Publishers/microsoft.
azure.networkwatcher/ArtifactTypes/VMExtension/Types/NetworkWatcherAgentWindows
Location      : westeurope
Name          :
RequestId     : e4e5ee24-2ae6-44d3-8e6c-428547661564
StatusCode    : OK


Get-AzureRmVMExtensionImage -Location westeurope -PublisherName
microsoft.azure.networkwatcher -Type networkwatcheragentwindows


Id               : /Subscriptions/f1f020d0-0fa6-4d01-b816-
5ec60497e851/Providers/Microsoft.Compute/Locations/westeurope/Publishers/microsof

t.azure.networkwatcher/ArtifactTypes/VMExtension/Types/networkwatcheragentwindows/V
ersions/1.4.104.0
Location         : westeurope
PublisherName    : microsoft.azure.networkwatcher
Type             : networkwatcheragentwindows
Version          : 1.4.104.0
FilterExpression :

Id               : /Subscriptions/f1f020d0-0fa6-4d01-b816-
5ec60497e851/Providers/Microsoft.Compute/Locations/westeurope/Publishers/microsof

t.azure.networkwatcher/ArtifactTypes/VMExtension/Types/networkwatcheragentwindows/V
ersions/1.4.20.0
Location         : westeurope
PublisherName    : microsoft.azure.networkwatcher
Type             : networkwatcheragentwindows
Version          : 1.4.20.0
FilterExpression :

Id               : /Subscriptions/f1f020d0-0fa6-4d01-b816-
5ec60497e851/Providers/Microsoft.Compute/Locations/westeurope/Publishers/microsof

t.azure.networkwatcher/ArtifactTypes/VMExtension/Types/networkwatcheragentwindows/V
ersions/1.4.306.5
Location         : westeurope
PublisherName    : microsoft.azure.networkwatcher
```

```
Type              : networkwatcheragentwindows
Version           : 1.4.306.5
FilterExpression :
```

The resource creation in the module is displayed below:

```
##############################################################################################
#This module allows the creation of NetworkWatcher Agent
##############################################################################################

#Variable declaration for Module


variable "AgentCount" {
  type    = "string"

}

variable "AgentName" {
  type    = "string"

}

variable "AgentLocation" {
  type    = "string"

}

variable "AgentRG" {
  type    = "string"

}

variable "VMName" {
  type    = "list"

}

variable "EnvironmentTag" {
  type    = "string"
```

```
}

variable "EnvironmentUsageTag" {
  type    = "string"

}

resource "azurerm_virtual_machine_extension" "Terra-NEtworkWatcherLinuxAgent" {


  count                = "${var.AgentCount}"
  name                 = "${var.AgentName}${count.index+1}"
  location             = "${var.AgentLocation}"
  resource_group_name  = "${var.AgentRG}"
  virtual_machine_name = "${element(var.VMName,count.index)}"
  publisher            = "Microsoft.Azure.NetworkWatcher"
  type                 = "NetworkWatcherAgentLinux"
  type_handler_version = "1.4"

      settings = <<SETTINGS
        {


        "commandToExecute": ""
        }
SETTINGS

  tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
  }
}
```

4.5 The output file

Apart from using the output for modules calls, we can also use outputs globally for the template. The outputs are saved in the Terraform state. Terraform display the output after the apply is completed and allow also the outputs to be called with the Terraform CLI command Tarraform output. Those values can be used for other steps of configuration if needed, with any tool capable to get the information in the file. The output file is displayed here as an example:

```
#########################################################
# This file defines which value are sent to output
#########################################################

#########################################################
# Resource group info Output

output "ResourceGroupName" {

    value = "${module.ResourceGroup.Name}"
}

output "ResourceGroupId" {

    value = "${module.ResourceGroup.Id}"
}

#########################################################
# vNet info Output

output "vNetName" {

    value = "${module.SampleArchi_vNet.Name}"
}

output "vNetId" {

    value = "${module.SampleArchi_vNet.Id}"
}

output "vNetAddressSpace" {

    value = "${module.SampleArchi_vNet.AddressSpace}"
}

#########################################################
# Log&Diag Storage account Info

output "DiagStorageAccountName" {

    value = "${module.DiagStorageAccount.Name}"
}

output "DiagStorageAccountID" {
```

```
    value = "${module.DiagStorageAccount.Id}"
}

output "DiagStorageAccountPrimaryBlobEP" {

    value = "${module.DiagStorageAccount.PrimaryBlobEP}"
}

output "DiagStorageAccountPrimaryQueueEP" {

    value = "${module.DiagStorageAccount.PrimaryQueueEP}"
}

output "DiagStorageAccountPrimaryTableEP" {

    value = "${module.DiagStorageAccount.PrimaryTableEP}"
}

output "DiagStorageAccountPrimaryFileEP" {

    value = "${module.DiagStorageAccount.PrimaryFileEP}"
}

output "DiagStorageAccountPrimaryAccessKey" {

    value = "${module.DiagStorageAccount.PrimaryAccessKey}"
}

output "DiagStorageAccountSecondaryAccessKey" {

    value = "${module.DiagStorageAccount.SecondaryAccessKey}"
}

#########################################################
# Files Storage account Info

output "FilesExchangeStorageAccountName" {

    value = "${module.FilesExchangeStorageAccount.Name}"
}

output "FilesExchangeStorageAccountID" {
```

```
    value = "${module.FilesExchangeStorageAccount.Id}"
}

output "FilesExchangeStorageAccountPrimaryBlobEP" {

    value = "${module.FilesExchangeStorageAccount.PrimaryBlobEP}"
}

output "FilesExchangeStorageAccountPrimaryQueueEP" {

    value = "${module.FilesExchangeStorageAccount.PrimaryQueueEP}"
}

output "FilesExchangeStorageAccountPrimaryTableEP" {

    value = "${module.FilesExchangeStorageAccount.PrimaryTableEP}"
}

output "FilesExchangeStorageAccountPrimaryFileEP" {

    value = "${module.FilesExchangeStorageAccount.PrimaryFileEP}"
}

output "FilesExchangeStorageAccountPrimaryAccessKey" {

    value = "${module.FilesExchangeStorageAccount.PrimaryAccessKey}"
}

output "FilesExchangeStorageAccountSecondaryAccessKey" {

    value = "${module.FilesExchangeStorageAccount.SecondaryAccessKey}"
}



##########################################################
# Subnet info Output
##########################################################

##########################################################
#FE_Subnet

output "FE_Subnet" {
```

```
    value = "${module.FE_Subnet.Name}"
}

output "FE_SubnetId" {

    value = "${module.FE_Subnet.Id}"
}

output "FE_SubnetAddressPrefix" {

    value = "${module.FE_Subnet.AddressPrefix}"
}


########################################################
#BE_Subnet

output "BE_SubnetName" {

    value = "${module.BE_Subnet.Name}"
}

output "BE_SubnetId" {

    value = "${module.BE_Subnet.Id}"
}

output "BE_SubnetAddressPrefix" {

    value = "${module.BE_Subnet.AddressPrefix}"
}
########################################################
#Bastion_Subnet

output "Bastion_SubnetName" {

    value = "${module.Bastion_Subnet.Name}"
}

output "Bastion_SubnetId" {

    value = "${module.Bastion_Subnet.Id}"
}
```

```
output "Bastion_SubnetAddressPrefix" {

    value = "${module.Bastion_Subnet.AddressPrefix}"
}

########################################################
#Bastion Output

output "Bastionfqdn" {

    value = ["${module.BastionPublicIP.fqdns}"]
}

output "BastionpublicIPAddress" {

    value = ["${module.BastionPublicIP.IPAddresses}"]
}

########################################################
#Azure Web LB Output

output "LBWebPublicIPfqdn" {

    value = ["${module.LBWebPublicIP.fqdns}"]
}

output "LBWebPublicIPpublicIPAddress" {

    value = ["${module.LBWebPublicIP.IPAddresses}"]
}
```

The output from the template is displayed below:

```
Apply complete! Resources: 61 added, 0 changed, 0 destroyed.

Outputs:

BE_SubnetAddressPrefix = 10.0.1.0/24
BE_SubnetId = /subscriptions/f1f020d0-0fa6-4d01-b816-5ec60497e851/resourceGroups/RG-
001/providers/Microsoft.Network/virtualNetworks/SampleArchi_vNet/subnets/BE_Subnet
BE_SubnetName = BE_Subnet
Bastion_SubnetAddressPrefix = 10.0.2.0/24
Bastion_SubnetId = /subscriptions/f1f020d0-0fa6-4d01-b816-5ec60497e851/resourceGroups/RG-
001/providers/Microsoft.Network/virtualNetworks/SampleArchi_vNet/subnets/Bastion_Subnet
Bastion_SubnetName = Bastion_Subnet
```

```
Bastionfqdn = [
   fqbqnbastionpip.westeurope.cloudapp.azure.com
]
BastionpublicIPAddress = [
   52.166.95.57
]
DiagStorageAccountID = /subscriptions/f1f020d0-0fa6-4d01-b816-5ec60497e851/resourceGroups/rg-
001/providers/Microsoft.Storage/storageAccounts/taluxdiaglogstorage
DiagStorageAccountName = taluxdiaglogstorage
DiagStorageAccountPrimaryAccessKey =
pYZiW73y75CJ4wuG1ZA5I7nuQKhJ7xFuV2+mPkS/Ao9Kotb81lKzThI3+606nzZt9Bxav8wwqrwQ+ilEENcGk
g==DiagStorageAccountPrimaryBlobEP = https://taluxdiaglogstorage.blob.core.windows.net/
DiagStorageAccountPrimaryFileEP = https://taluxdiaglogstorage.file.core.windows.net/
DiagStorageAccountPrimaryQueueEP = https://taluxdiaglogstorage.queue.core.windows.net/
DiagStorageAccountPrimaryTableEP = https://taluxdiaglogstorage.table.core.windows.net/
DiagStorageAccountSecondaryAccessKey =
RxTye0MOxaDBxrR6Wdq6nd05o4vwfOtGsIAXN8tyjXuvFbNQ9U/t9LFgN8j/GfdS/OvQINajrhO6LuUdNTRHK
A==
FE_Subnet = FE_SubnetFE_SubnetAddressPrefix = 10.0.0.0/24
FE_SubnetId = /subscriptions/f1f020d0-0fa6-4d01-b816-5ec60497e851/resourceGroups/RG-
001/providers/Microsoft.Network/virtualNetworks/SampleArchi_vNet/subnets/FE_Subnet
FilesExchangeStorageAccountID = /subscriptions/f1f020d0-0fa6-4d01-b816-
5ec60497e851/resourceGroups/rg-001/providers/Microsoft.Storage/storageAccounts/oalglfilestorage
FilesExchangeStorageAccountName = oalglfilestorageFilesExchangeStorageAccountPrimaryAccessKey =
GRhusCIedyu4ebI5e8OMNXkD1ID/slApvuzkP6SFxoN7NxRksnRfnr2/1kESvVfWuY6d/R5pN5NDBXZedbhJiQ
==
FilesExchangeStorageAccountPrimaryBlobEP =
https://oalglfilestorage.blob.core.windows.net/FilesExchangeStorageAccountPrimaryFileEP =
https://oalglfilestorage.file.core.windows.net/
FilesExchangeStorageAccountPrimaryQueueEP = https://oalglfilestorage.queue.core.windows.net/
FilesExchangeStorageAccountPrimaryTableEP =
https://oalglfilestorage.table.core.windows.net/FilesExchangeStorageAccountSecondaryAccessKey =
hR/xR0HUVW1HmzP9qoNceDMm0ZJDSnl3/3enqaUYFtCQDnMni6MBLjiPINmOYF+WxM9gUY8RRyCy+uRL3
3TeuQ==
LBWebPublicIPfqdn = [
   psjaplbwebpip.westeurope.cloudapp.azure.com
]
LBWebPublicIPpublicIPAddress = [
   52.166.92.30]
ResourceGroupId = /subscriptions/f1f020d0-0fa6-4d01-b816-5ec60497e851/resourceGroups/RG-001
ResourceGroupName = RG-001
vNetAddressSpace = [
   10.0.0.0/20
]
```

```
vNetId = /subscriptions/f1f020d0-0fa6-4d01-b816-5ec60497e851/resourceGroups/RG-
001/providers/Microsoft.Network/virtualNetworks/SampleArchi_vNet
vNetName = SampleArchi_vNet
```

5 Conclusion

In this article, we transformed the basic Terraform template to use the module functionality. The template code is available on Github here. In a later article, we will have a look at the Network Watcher capabilities with the VM agent and also with other Azure Object Network related.

Another Tech blog

My thoughts and experiences on Cloud related tek