**Creating a VM in Azure by using Terraform with for loops**

This post describes how to easily create multiple virtual machines on Azure® by using Hashicorp® Terraform® with the help of Visual Studio® Code (VS Code).

Overview

In this blog post, I focus on bulk VM creation on Azure. Not only this, I also emphasize the Infrastructure as Code (IaC) approach. I use VS Code throughout this blog, but you can use your favorite IDE.

Let's get started,

I generally went through a kind of requirement to create multiple VMs in Azure with some specific standard or typical nomenclature. Even when we build stuff on the fly, we have to keep an eagle eye on everything to sustain our standards.

And maintaining the standard is not at all easy, mainly because we have to keep the same name canons for VM name, OS disk, data disk, NIC name, etc.

To overcome this, IAC gives us a hold on every aspect, including what we are going to build in the backend on the real surface.

I have written some simple Terraform code to create a VM in Azure with similar names for every resource. I also use tags, which are the new way of segregating things. So, lets straight jump into the action and get our hands dirty with Terraform.

For instance, Azure VM creation requires a ResourceGroup (RG), VNET, SUBNET, NIC, and VM configuration. So, keeping all them in mind, let’s make all the files separately so that it is easy to understand.

First, we need to create a RG code block in Hashicorp configuration language (HCL).

I am going to create two RGs (one for VM and NIC and other for Networking stuff). I don’t recommend keeping everything together as a best practice. Before creating the RGs, let’s define a provider block for Terraform so it can understand that we are going to deploy resources in Azure. I keep this information for Terraform in an isolated file called *providers.tf:*

provider "azurerm" {

features {}

}

My resouceGroup code looks likes, *rg.tf:*

resource "azurerm\_resource\_group" "app" {

  name     = "UKS-PRD-RSG-BLG-APP"

  location = var.location

   tags = {

    Environment = "Production"

    Buildby     = "Gourav Kumar"

    Builddate   = "20200611"

    Bugetcode   = "Blog"

  }

}

resource "azurerm\_resource\_group" "net" {

  name     = "UKS-PRD-RSG-BLG-VNET"

  location = var.location

   tags = {

    Environment = "Production"

    Buildby     = "Gourav Kumar"

    Builddate   = "20200611"

    Bugetcode   = "Blog"

  }

}

Now, the second task is to create a VNET and a subnet for our VM. Here we go:

The VNET block is something like this, *vnet.tf:*

resource "azurerm\_virtual\_network" "vnet" {

  name                = "UKS-VNET-BLG-PRD"

  location            = var.location

  resource\_group\_name = azurerm\_resource\_group.net.name

  address\_space       = ["10.0.0.0/16"]

   tags = {

    Environment = "Production"

    Buildby     = "Gourav Kumar"

    Builddate   = "20200611"

    Bugetcode   = "Blog"

  }

}

And SUBNET is, *subnet.tf:*

resource "azurerm\_subnet" "subnet" {

  name                 = "UKS-VNET-BLG-PRD-APP"

  resource\_group\_name  = azurerm\_resource\_group.net.name

  virtual\_network\_name = azurerm\_virtual\_network.vnet.name

  address\_prefix       = "10.0.1.0/24"

}

We are almost done with networking stuff except for our VM's NIC. The next is NIC, but I am going to put this in VM's ResourceGroup.

Notice in resource\_group\_name that I referenced app RG because keeping NIC in Network RG does not suit me.

*Nic.tf:*

resource "azurerm\_network\_interface" "nic" {

  for\_each                      = toset(var.vm\_name)

  name                          = each.value

  location                      = var.location

  resource\_group\_name           = azurerm\_resource\_group.app.name

  ip\_configuration {

    name                          = "ipconfig1"

    subnet\_id                     = azurerm\_subnet.subnet.id

    private\_ip\_address\_allocation = "Dynamic"

  }

 tags = {

     Environment = "Production"

     Buildby     = "Gourav Kumar"

     Builddate   = "20200611"

     Bugetcode   = "Blog"

  }

}

Now we are done with all networking stuff, but most people do not consider a VM without diagnostic settings enabled to be a good build. And I don't want anyone to call me a "new builder" :P

Storage account code is simple, and we can call ourselves good Azure implementors.

The following code block for the Storage account is *storage\_account.tf:*

resource "azurerm\_storage\_account" "storage" {

  name                     = "diagblogvmnode"

  resource\_group\_name      = azurerm\_resource\_group.app.name

  location                 = var.location

  account\_tier             = "Standard"

  account\_replication\_type = "LRS"

   tags = {

     Environment = "Production"

     Buildby     = "Gourav Kumar"

     Builddate   = "20200611"

     Bugetcode   = "Blog"

  }

}

Now, we are all set for VM code, but remember, we have to use the same standards, such as VM name, NIC name, disk name, etc. And we have done this so far.

Final code for our VM, *vm.tf:*

resource "azurerm\_virtual\_machine" "vm" {

  for\_each               = toset(var.vm\_name)

  name                  = each.value

  location              = var.location

  resource\_group\_name   = azurerm\_resource\_group.app.name

  network\_interface\_ids = [azurerm\_network\_interface.nic[each.key].id]

  vm\_size               = "Standard\_D8s\_v3"

  delete\_os\_disk\_on\_termination    = true

  delete\_data\_disks\_on\_termination = true

  storage\_image\_reference {

    publisher = "OpenLogic"

    offer     = "CentOS"

    sku       = "7.7"

    version   = "latest"

  }

  storage\_os\_disk {

    name              = "${each.key}-osdisk"

    caching           = "ReadWrite"

    create\_option     = "FromImage"

    managed\_disk\_type = "Premium\_LRS"

    disk\_size\_gb      = "32"

  }

  storage\_data\_disk {

      name                      = "${each.key}-disk1"

      caching                   = "None"

      create\_option             = "Empty"

      disk\_size\_gb              = 512

      lun                       = 0

      managed\_disk\_type         = "Premium\_LRS"

  }

  os\_profile {

    computer\_name  = each.value

    admin\_username = "${each.key}-adm"

    admin\_password = "oOKgciM89u6kFq5beO3YPrFDjFLDjn7e"

  }

  os\_profile\_linux\_config {

      disable\_password\_authentication = false

  }

  boot\_diagnostics {

    enabled     = "true"

    storage\_uri = azurerm\_storage\_account.storage.primary\_blob\_endpoint

  }

 tags = {

     Environment = "Production"

     Buildby     = "Gourav Kumar"

     Builddate   = "20200611"

     Bugetcode   = "Blog"

}

}

And finally, I have put a few things in variables like all the VM names. You can put other things in variables, such as RG, VNET, and SUBNET name. (In case you are doing giant builds).

*Variable.tf:*

variable "vm\_name" {

  description = "Name of the VM."

  type        = list(string)

  default     = ["BlogVMNode1", "BlogVMNode2"]

}

variable "location" {

  description = "Region to build into"

  default     = "UK South"

}

My goal is to create two VMs by using a for loop. Thus, the vm\_name variable has two VM names (you can add multiples), and the second variable is location, where I want to deploy Azure resources.

Let me share how I keep the standard intact. I used VMName-nic for every nic, VMName-Osdisk, and -data1.  
All resources such as VM, NIC, DISK pop up with the same nomenclature, and, by doing this, we can easily understand what resources belong to which VM.

Let's get down to business, and learn how to deploy them in Azure.

It's time to boil the eggs.

Now, we need Azure CLI for this task. This is not a simple DevOps approach as we are neither going to run this through pipeline nor store the sate file somewhere other than our directory.

If you don’t have the az CLI installed, use the following link to install it:

<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>

First, we need to login into Azure:

az login

After login, we need to check how many subscriptions we have for this tenant:

az account list --output table

Now, set the right subscription for our deployment:

az account set --subscription "XXXXXX-XXXXXXX-XXXX-XXX"

Then, navigate to the directory where we have put our all terraform code file (I used temp for this example, but I don't recommend it):

Cd' C:\temp'

Initialize Terraform in this directory:

terraform init

Run terraform plan to check what we are going to build:

terraform plan

You can save the plan output for later use by adding the -out flag in the command.

Now the final step, terraform code execution and deployment:

terraform apply

It asks you to type yes. This is just a confirmation check, and you can use -auto-approve to automate approval. By doing this, it will not ask for manual confirmation:

terraform apply -auot-apply

Now you can check the VM that we have created:

az vm list -o table

Some secret and hacks

The for loop used in Terraform with the following syntax, creates and runs as many times as we have defined the variable (twice, in our case, since I have defined two VM names in the variable):

for\_each = toset(var.vm\_name)

The for loop in Terraform has a pair key and value, but both are generally similar.

The each Object:

In resource blocks, where you set for\_each, an additional each object is available in expressions so that you can modify the configuration of each instance. This object has two attributes:

each.key - The map key (or set member) corresponding to this instance.

each.value -The map value corresponding to this instance. (If a set was provided, this is the same as each.key.)

You can use the terraform fmt command to format the code.

Use this command to rewrite Terraform configuration files to conform to a canonical format and style. This command applies a subset of the Terraform language style conventions, along with other minor adjustments for readability.

A special Note :-

If you want to create windows VM then you have to change offer, publisher, version and configuration block in terraform code as follows, (Delete the config code for Linux, which is there in code )

os\_profile\_windows\_config {

provision\_vm\_agent = true

timezone = "UTC" # you can change this as per requirement

}

### Conclusion

Have fun using Terraform for Azure and maintain the standard with some simple code.

Thanks for reading, feedback is always welcome.  
You can reach me at [gouravin@outlook.com](mailto:gouravin@outlook.com)

Thanks,  
Gourav Kumar