Toggle VNet On and Off for Production and Development Environment

Problem Statement

When deploying resources on Azure in a secure environment, resources are usually created behind a Private Network (VNet), without public access and with private endpoints to consume resources. This is the recommended approach for pre-production or production environments.

Accessing protected resources from a local machine implies one of the following options:

- Use a VPN
- Use a **jump box**
    - With SSH activated (less secure)
    - With Bastion (recommended approach)

However, a developer may want to deploy a test environment (in a non-production subscription) for their tests during development phase, without the complexity of networking.

In addition, infrastructure code should not be duplicated: it has to be the same whether resources are deployed in a production like environment or in development environment.

Option

The idea is to offer, **via a single boolean variable**, the option to deploy resources behind a VNet or not using one infrastructure code base. Securing resources behind a VNet usually implies that public accesses are disabled and private endpoints are created. This is something to have in mind because, as a developer, public access must be activated in order to connect to this environment.

The deployment pipeline will set these resources behind a VNet and will secure them by removing public accesses. Developers will be able to run the same deployment script, specifying that resources will not be behind a VNet nor have public accesses disabled.

Let's consider the following use case: we want to deploy a VNet, a subnet, a storage account with no public access and a private endpoint for the table.

The *magic* variable that will help toggling security will be called `behind_vnet`, of type boolean.

Let's implement this use case using `Terraform`.

> The code below does not contain everything, the purpose is to show the pattern and not how to deploy these resources. For more information on Terraform, please refer to the official documentation.

There is no `if` *per se* in Terraform to define whether a specific resource should be deployed or not based on a variable value. However, we can use the `count` meta-argument. The strength of this meta-argument is if its value is `0`, the block is skipped.

Here is below the code snippets for this deployment:

- variables.tf

```
variable "behind_vnet" {
    type    = bool
}
```

- main.tf

```
resource "azurerm_virtual_network" "vnet" {
    count = var.behind_vnet ? 1 : 0

    name                = "MyVnet"
    address_space       = [x.x.x.x/16]
    resource_group_name = "MyResourceGroup"
    location            = "WestEurope"

    ...

    subnet {
        name              = "subnet_1"
        address_prefix    = "x.x.x.x/24"
    }
}

resource "azurerm_storage_account" "storage_account" {
    name                = "storage"
    resource_group_name = "MyResourceGroup"
    location            = "WestEurope"
    tags                = var.tags

    ...

    public_network_access_enabled = var.behind_vnet ? false
: true
}

resource "azurerm_private_endpoint"
"storage_account_table_private_endpoint" {
    count = var.behind_vnet ? 1 : 0

    name                = "pe-storage"
```

```
    subnet_id           =
azurerm_virtual_network.vnet[0].subnet[0].id

    ...

    private_service_connection {
        name                           = "psc-storage"
        private_connection_resource_id =
azurerm_storage_account.storage_account.id
        subresource_names              = [ "table" ]
        ...
    }

    private_dns_zone_group {
        name = "privateDnsZoneGroup"
        ...
    }
}
```

If we run

```
terraform apply -var behind_vnet=true
```

then all the resources above will be deployed, and it is what we want on a pre-production or production environment. The instruction `count = var.behind_vnet ? 1 : 0` will set `count` with the value `1`, therefore blocks will be executed.

However, if we run

```
terraform apply -var behind_vnet=false
```

the `azurerm_virtual_network` and `azurerm_private_endpoint` resources will be skipped (because `count` will be `0`). The resource `azurerm_storage_account` will be created, with minor differences in some

properties: for instance, here, `public_network_access_enabled` will be set to `true` (and this is the goal for a developer to be able to access resources created).

The same pattern can be applied over and over for the entire infrastructure code.

Conclusion

With this approach, the same infrastructure code base can be used to target a production like environment with secured resources behind a VNet with no public accesses and also a more permissive development environment.

However, there are a couple of trade-offs with this approach:

- if a resource has the `count` argument, it needs to be treated as a list, and not a single item. In the example above, if there is a need to reference the resource `azurerm_virtual_network` later in the code,

  ```
  azurerm_virtual_network.vnet.id
  ```

  will not work. The following must be used

  ```
  azurerm_virtual_network.vnet[0].id # First (and only) item
  of the collection
  ```

- The meta-argument `count` cannot be used with `for_each` for a whole block. That means that the use of loops to deploy multiple endpoints for instance will not work. Each private endpoints will need to be deployed individually.

Last update: August 22, 2024