VMware Cloud on AWS

VMware Cloud on AWS Terraform deployment – Phase 2

Gilles Chekroun
July 6, 2022

This blog post is a continuation of the phase 1 blog post for Using Terraform with multiple providers to deploy and configure VMware Cloud on AWS. We will pass parameters from one phase to the other. As noted in the phase 1, all source files are available for download here.

Output file from phase 1

In our phase 1, we created an output.tf file with all the parameters we will need in subsequent phase including the NSX-T proxy.

Below is what the `output.tf` file looks like:

```
output "sddc_subnet"                {value = module.VPCs.Subnet10-Att_vpc_id}
output "proxy_url"                  {value = module.SDDC.proxy_url}
output "vc_url"                     {value = module.SDDC.vc_url}
output "vc_public_IP"               {value = module.SDDC.vc_public_IP}
output "cloud_username"             {value = module.SDDC.cloud_username}
output "cloud_password"             {
  sensitive = true
  value = module.SDDC.cloud_password
}
output "Windows_IP"                 {value = module.EC2s.Windows_IP }
output "nsxt_private_IP"            {value = module.SDDC.nsxt_private_IP}
output "nsxt_cloudadmin"            {value = module.SDDC.nsxt_cloudadmin}
output "nsxt_cloudadmin_password"   {
  sensitive = true
  value = module.SDDC.nsxt_cloudadmin_password
}
```

All output parameters in this file come from different modules. For example, we can see how the proxy_url output parameter is sourced from the end of the SDDC module:

```
output "proxy_url"                  {value =
trimsuffix(trimprefix(vmc_sddc.Terraform_SDDC1.nsxt_reverse_proxy_url, "https://"), "/sks-
nsxt-manager")}
```

Using Terraform functions "`trimsuffix`" and "`trimprefix`" we can remove the string https:// and "`/sks-nsxt-manager`" from the `nsxt_reverse_proxy_url` output and get the `host` value needed for the NXST Provider.

After the `terraform apply` command in Phase 1 the output will be:

```
Outputs:

Windows_IP = "35.86.xx.xx"
cloud_password = <sensitive>
cloud_username = "cloudadmin@vmc.local"
nsxt_cloudadmin = "cloud_admin"
nsxt_cloudadmin_password = <sensitive>
nsxt_private_IP = "10.10.xx.xx"
proxy_url = "nsx-52-38-xx-xx.rp.vmwarevmc.com/vmc/reverse-proxy/api/orgs/7421a286-f7bf-4f34-
8567-xxxx/sddcs/9e0a0411-6cbb-4145-b4d5-xxxx"
sddc_subnet = "subnet-074b4d8bd1xxxx"
vc_public_IP = "52.38.xx.xx"
vc_url = "vcenter.sddc-52-38-xx-xx.vmwarevmc.com"
```

State files

After we execute the apply command in Phase 1, Terraform will generate a state file for it named `phase1.tfstate`. We can read this file and grab output parameters in our `deploy.sh` script and set our environment with the following three parameters needed for the NSX-T Terraform provider:

- host

- nsxt_username
- nsxt_password

```
export TF_VAR_host=$(terraform output -state=./phase1.tfstate proxy_url |
sed 's/\"//g')
export TF_VAR_nsxt_username=$(terraform output -state=./phase1.tfstate
nsxt_cloudadmin)
export TF_VAR_nsxt_password=$(terraform output -state=./phase1.tfstate
nsxt_cloudadmin_password)
```

Phase 2 – NSX-T provider

main.tf

Step 1: Set backend for phase 2 state file and read phase 1 state file.

```
terraform {
  backend "local" {
    path = "../../phase2.tfstate"
  }
}
# Import the state from phase 1 and read the outputs
data "terraform_remote_state" "phase1" {
  backend = "local"
  config = {
    path    = "../../phase1.tfstate"
  }
}
```

Step 2: Set NSX-T provider

```
provider "nsxt" {
  host                    = var.host
  username                = var.nsxt_username
  password                = var.nsxt_password
  vmc_auth_mode           = "Bearer"
  vmc_token               = var.vmc_token
  allow_unverified_ssl    = true
  enforcement_point       = "vmc-enforcementpoint"
}
```

variables.tf

The `VMC_subnets` variable will hold the details for NSX-T subnets we will create.

```
variable "vmc_token"  {}
variable "host" {}
variable "nsxt_username" {}
variable "nsxt_password" {}
```

```
/*================
Subnets IP ranges
=================*/
variable "VMC_subnets" {
  default = {

    Subnet12            = "12.12.12.0/24"
    Subnet12gw          = "12.12.12.1/24"
    Subnet12dhcp        = "12.12.12.100-12.12.12.200"

    Subnet13            = "13.13.13.0/24"
    Subnet13gw          = "13.13.13.1/24"
    Subnet13dhcp        = "13.13.13.100-13.13.13.200"

  }
}
```

In the `main.tf` file the NSX module will look like:

```
module "NSX" {
  source = "../NSX"

  Subnet12              = var.VMC_subnets["Subnet12"]
  Subnet12gw            = var.VMC_subnets["Subnet12gw"]
  Subnet12dhcp          = var.VMC_subnets["Subnet12dhcp"]
  Subnet13              = var.VMC_subnets["Subnet13"]
  Subnet13gw            = var.VMC_subnets["Subnet13gw"]
  Subnet13dhcp          = var.VMC_subnets["Subnet13dhcp"]
  Home_Gilles           = var.Home_Gilles
}
```

Note that the `Home_Gilles` variable is for holding my home IP address for a secure vCenter access.

NSX Module

In this module we are going to create the networking and security elements needed in the SDDC. This includes:

- MGW firewall rules
- CGW firewall rules
- NSX segments (12 and 13)
- Compute groups (12 and 13)
- Management group for Home IPs
- Security groups based on NSX tags for Blue VMs and Red VMs
- DFW rules to block ping from Blue VMs to Red VMs

MGW FW rules

Since the VMC Networking environment is pre-build at SDDC creation, we need to use the predefined gateway resource:

```
resource "nsxt_policy_predefined_gateway_policy" "mgw" {
```

For additional protection, we will not allow deletion of this resource using:

```
lifecycle { prevent_destroy = true }
```

The FW rule order in the code is the FW order in the User Interface. As an example, here is the vCenter access rule for my Home IP:

```
rule {
  action = "ALLOW"
  destination_groups     = ["/infra/domains/mgw/groups/VCENTER"]
  destinations_excluded = false
  direction              = "IN_OUT"
  disabled               = false
  display_name           = "vCenter Inbound"
  ip_version             = "IPV4_IPV6"
  logged                 = false
  profiles               = []
  scope                  = ["/infra/labels/mgw"]
  services               = [
    "/infra/services/HTTPS",
    "/infra/services/ICMP-ALL",
    "/infra/services/SSO"
  ]
  source_groups          = [nsxt_policy_group.Home_Gilles.path]
  sources_excluded       = false
}
```

Default rules will need to be added to the code otherwise they will be removed at the first `terraform apply` execution:

```
on first terraform apply
  rule {
    action = "ALLOW"
    destination_groups     = []
    destinations_excluded = false
    direction              = "IN_OUT"
    disabled               = false
    display_name           = "ESXi Outbound"
    ip_version             = "IPV4_IPV6"
    logged                 = false
    profiles               = []
    scope                  = ["/infra/labels/mgw"]
    services               = []
    source_groups          = ["/infra/domains/mgw/groups/ESXI"]
    sources_excluded       = false
  }
  rule {
    action = "ALLOW"
    destination_groups     = []
    destinations_excluded = false
    direction              = "IN_OUT"
    disabled               = false
    display_name           = "vCenter Outbound"
    ip_version             = "IPV4_IPV6"
    logged                 = false
    profiles               = []
    scope                  = ["/infra/labels/mgw"]
    services               = []
    source_groups          = ["/infra/domains/mgw/groups/VCENTER"]
    sources_excluded = false
  }
}
```

NSX Segments

VMware Cloud on AWS NSX segments are created using the `nsxt_policy_fixed_segment` resource. DHCP can be coded as well:

```
resource "nsxt_policy_fixed_segment" "seg12" {
  display_name        = "seg12"
  description         = "Terraform provisioned Segment"
  connectivity_path   = "/infra/tier-1s/cgw"
  transport_zone_path = data.nsxt_policy_transport_zone.TZ.path
  subnet {
    cidr              = var.Subnet12gw
    dhcp_ranges       = [var.Subnet12dhcp]
  }
}
```

Compute and Management groups

The snippet below shows Compute group configuration based on IP address. Note the `cgw` domain:

```
resource "nsxt_policy_group" "group12" {
  display_name = "tf-group12"
  description  = "Terraform provisioned Group"
  domain       = "cgw"
  criteria {
    ipaddress_expression {
      ip_addresses = [var.Subnet12]
    }
  }
}
```

Similarly, the snippet below shows Management group configuration. Note the `mgw` domain:

```
resource "nsxt_policy_group" "Home_Gilles" {
  display_name = "Home_Gilles"
  description  = "Terraform provisioned Group"
  domain       = "mgw"
  criteria {
    ipaddress_expression {
      ip_addresses = var.Home_Gilles
    }
  }
}
```

NSX Security groups and NSX Tags

Next we will create a compute group for Blue VMs:

```
resource "nsxt_policy_group" "Blue_VMs" {
  display_name = "Blue_VMs"
  description = "Terraform provisioned Group"
  domain       = "cgw"
  criteria {
    condition {
      key = "Tag"
      member_type = "VirtualMachine"
      operator = "EQUALS"
      value = "Blue|NSX_tag"
    }
  }
}
```

DFW Firewall rules

In this example we have 2 compute groups created above for Blue VMs and Red VMs. We want to restrict PING between the 2 groups. To do that we can create a security policy named `Colors` with 2 rules:

- Drop PING from Blue_VNs to Red_VMs groups
- Drop PING from Red_VMs to Blue_VMs groups

```
resource "nsxt_policy_security_policy" "Colors" {
  display_name = "Colors"
  description = "Terraform provisioned Security Policy"
  category = "Application"
  domain = "cgw"
  locked = false
  stateful = true
  tcp_strict = false

  rule {
    display_name = "Blue2Red"
    source_groups = [
      nsxt_policy_group.Blue_VMs.path]
    destination_groups = [
      nsxt_policy_group.Red_VMs.path]
    action = "DROP"
    services = ["/infra/services/ICMP-ALL"]
    logged = true
  }
  rule {
    display_name = "Red2Blue"
    source_groups = [
      nsxt_policy_group.Red_VMs.path]
    destination_groups = [
      nsxt_policy_group.Blue_VMs.path]
    action = "DROP"
    services = ["/infra/services/ICMP-ALL"]
    logged = true
  }
}
```

The `outputs` for this module will be the NSX segment names needed for Phase 3 – vSphere deployment of Virtual Machines.

```
output "segment12_name"     {value =
nsxt_policy_fixed_segment.seg12.display_name}
output "segment13_name"     {value =
nsxt_policy_fixed_segment.seg13.display_name}
```

Phase 2 deployment

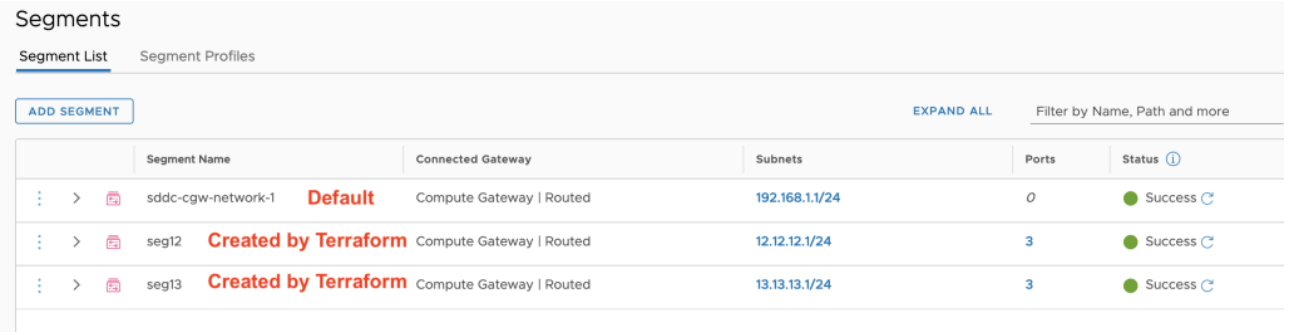
After the phase 2 `terraform apply` in our `deploy.sh` script, the output will give us:

```
Outputs:

segment12_name = "seg12"
segment13_name = "seg13"
subnet12 = "12.12.12.0/24"
subnet13 = "13.13.13.0/24"

Press enter to continue (^C to stop)...
```

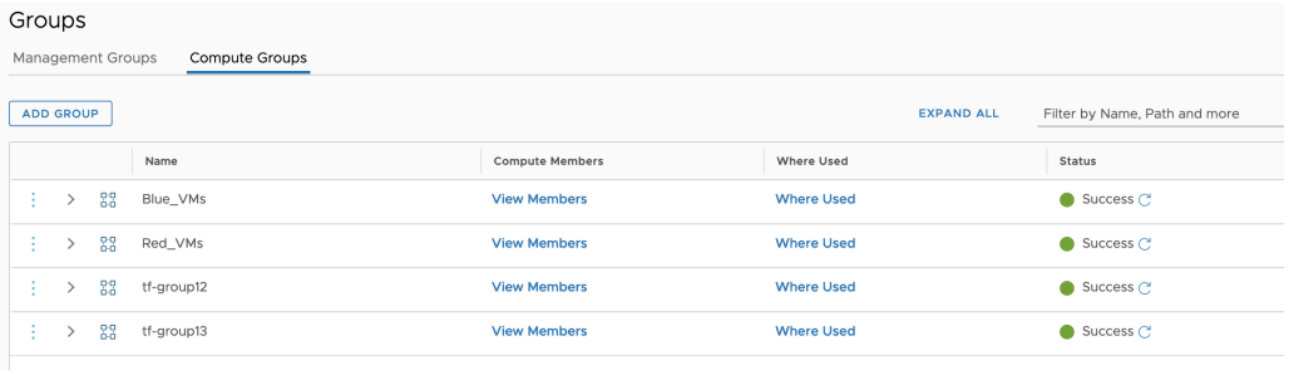
VMware Cloud Console

Finally, we can review the created network configuration in the VMC Console UI.

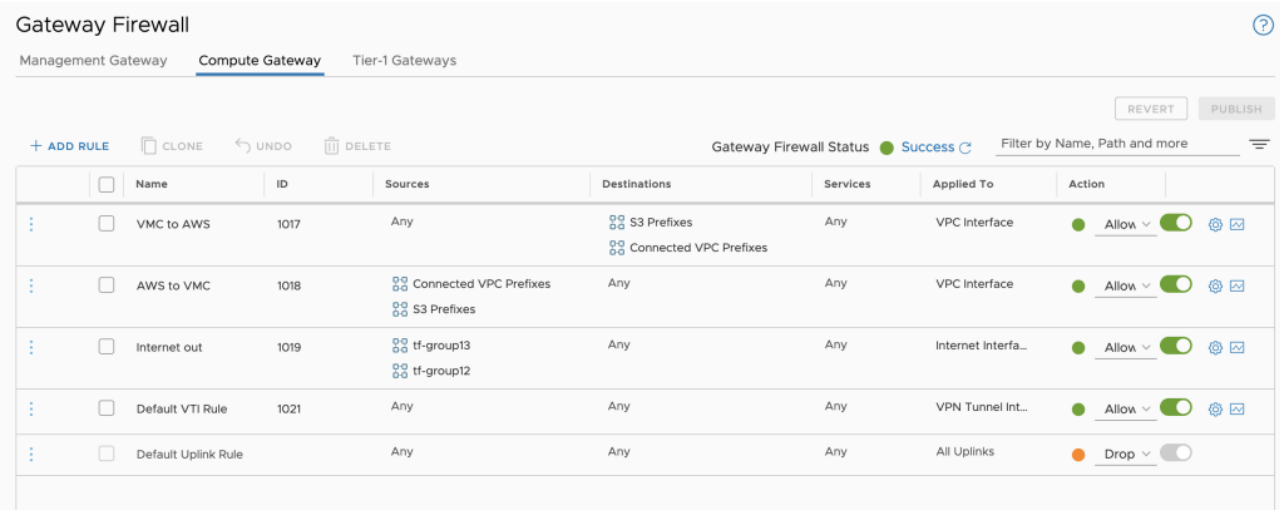
Created Segments



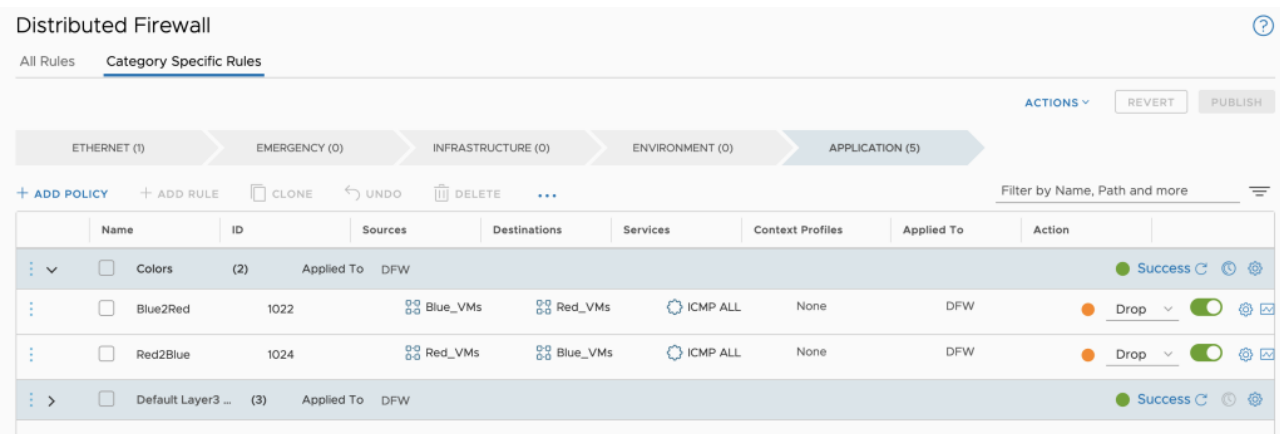
Created Groups



Created CGW FW rules



Created DFW rules



In our next and final blog post (phase 3), we will deploy an S3 Content Library and 6 VMs (3 blue and 3 red)

Stay tuned!

Gilles Chekroun

Gilles Chekroun is Lead VMware Cloud on AWS Solutions Architect in the European team. He joined VMware in 2015 after spending 20 years at Cisco in the Data Centre Network...

Related Articles

VMware Cloud on AWS

VMware Cloud on AWS Free Trial is available now! Start your hybrid cloud journey today!

Nancy Cheng
July 11, 2023

VMware Cloud on AWS

VMware Lab Platform with VMware Cloud on AWS- Delivering on-demand labs anytime and anywhere

noeldnerb
July 6, 2023

VMware Cloud on AWS

Introducing the New and Improved Networking Experience for VMware Cloud Console

Nancy Cheng
June 9, 2023

VMware Cloud on AWS

Now Live – vCenter Federation for VMware Cloud on AWS

Aakash Chandhoke
June 8, 2023

VMware Cloud on AWS

Custom Managed Elastic DRS Policy: Improving Cost-Efficiency and Performance in VMware Cloud on AWS

Amitha Shetty
May 22, 2023

vmware®

Company

About Us

Executive Leadership

News & Stories

Investor Relations

Customer Stories

Diversity, Equity & Inclusion

Environment, Social & Governance