Quickstart: Use Terraform to create a Linux VM

Article • 07/24/2023

This article was partially created with the help of AI. An author reviewed and revised the content as needed. **Read more**.

Applies to: ✓ Linux VMs

Article tested with the following Terraform and Terraform provider versions:

This article shows you how to create a complete Linux environment and supporting resources with Terraform. Those resources include a virtual network, subnet, public IP address, and more.

Terraform enables the definition, preview, and deployment of cloud infrastructure. Using Terraform, you create configuration files using HCL syntax. The HCL syntax allows you to specify the cloud provider - such as Azure - and the elements that make up your cloud infrastructure. After you create your configuration files, you create an *execution plan* that allows you to preview your infrastructure changes before they're deployed. Once you verify the changes, you apply the execution plan to deploy the infrastructure.

In this article, you learn how to:

- ✓ Create a random value for the Azure resource group name using random_pet.
- ✓ Create an Azure resource group using azurerm_resource_group.
- ✓ Create a virtual network (VNET) using azurerm_virtual_network.
- ✓ Create a subnet using azurerm_subnet.
- ✓ Create a public IP using azurerm_public_ip.
- ✓ Create a network security group using azurerm_network_security_group.
- ✓ Create a network interface using azurerm_network_interface.
- ✓ Create an association between the network security group and the network interface using azurerm_network_interface_security_group_association.
- ✓ Generate a random value for a unique storage account name using random_id.
- ✓ Create a storage account for boot diagnostics using azurerm_storage_account.
- ✓ Create a Linux VM using azurerm_linux_virtual_machine
- ✓ Create an AzAPI resource azapi_resource.
- ✓ Create an AzAPI resource to generate an SSH key pair using azapi_resource_action.

Prerequisites

- Install and configure Terraform

Implement the Terraform code

> **Note**
>
> The sample code for this article is located in the **Azure Terraform GitHub repo**. You can view the log file containing the **test results from current and previous versions of Terraform**.
>
> See more **articles and sample code showing how to use Terraform to manage Azure resources**

1. Create a directory in which to test the sample Terraform code and make it the current directory.

2. Create a file named `providers.tf` and insert the following code:

Terraform

```
terraform {
  required_version = ">=0.12"

  required_providers {
    azapi = {
      source  = "azure/azapi"
      version = "~>1.5"
    }
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~>2.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~>3.0"
    }
  }
}

provider "azurerm" {
  features {}
}
```

3. Create a file named `ssh.tf` and insert the following code:

Terraform

```
resource "random_pet" "ssh_key_name" {
  prefix    = "ssh"
  separator = ""
}

resource "azapi_resource_action" "ssh_public_key_gen" {
  type        = "Microsoft.Compute/sshPublicKeys@2022-11-01"
  resource_id = azapi_resource.ssh_public_key.id
  action      = "generateKeyPair"
  method      = "POST"

  response_export_values = ["publicKey", "privateKey"]
}

resource "azapi_resource" "ssh_public_key" {
  type      = "Microsoft.Compute/sshPublicKeys@2022-11-01"
  name      = random_pet.ssh_key_name.id
  location  = azurerm_resource_group.rg.location
  parent_id = azurerm_resource_group.rg.id
}

output "key_data" {
  value =
jsondecode(azapi_resource_action.ssh_public_key_gen.output).publicKey
}
```

4. Create a file named `main.tf` and insert the following code:

Terraform

```
resource "random_pet" "rg_name" {
  prefix = var.resource_group_name_prefix
}

resource "azurerm_resource_group" "rg" {
  location = var.resource_group_location
  name     = random_pet.rg_name.id
}

# Create virtual network
resource "azurerm_virtual_network" "my_terraform_network" {
  name                = "myVnet"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
}

# Create subnet
resource "azurerm_subnet" "my_terraform_subnet" {
  name                 = "mySubnet"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name =
azurerm_virtual_network.my_terraform_network.name
  address_prefixes     = ["10.0.1.0/24"]
}

# Create public IPs
resource "azurerm_public_ip" "my_terraform_public_ip" {
  name                = "myPublicIP"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  allocation_method   = "Dynamic"
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "my_terraform_nsg" {
  name                = "myNetworkSecurityGroup"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  security_rule {
    name                       = "SSH"
    priority                   = 1001
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "22"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
}

# Create network interface
resource "azurerm_network_interface" "my_terraform_nic" {
  name                = "myNIC"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  ip_configuration {
    name                          = "my_nic_configuration"
    subnet_id                     =
azurerm_subnet.my_terraform_subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          =
azurerm_public_ip.my_terraform_public_ip.id
  }
}

# Connect the security group to the network interface
resource "azurerm_network_interface_security_group_association" "exam-
ple" {
  network_interface_id      =
azurerm_network_interface.my_terraform_nic.id
  network_security_group_id =
azurerm_network_security_group.my_terraform_nsg.id
}

# Generate random text for a unique storage account name
resource "random_id" "random_id" {
  keepers = {
    # Generate a new ID only when a new resource group is defined
    resource_group = azurerm_resource_group.rg.name
  }

  byte_length = 8
}

# Create storage account for boot diagnostics
resource "azurerm_storage_account" "my_storage_account" {
  name                     = "diag${random_id.random_id.hex}"
  location                 = azurerm_resource_group.rg.location
  resource_group_name      = azurerm_resource_group.rg.name
  account_tier             = "Standard"
  account_replication_type = "LRS"
}

# Create virtual machine
resource "azurerm_linux_virtual_machine" "my_terraform_vm" {
  name                  = "myVM"
  location              = azurerm_resource_group.rg.location
  resource_group_name   = azurerm_resource_group.rg.name
  network_interface_ids =
[azurerm_network_interface.my_terraform_nic.id]
  size                  = "Standard_DS1_v2"

  os_disk {
    name                 = "myOsDisk"
    caching              = "ReadWrite"
    storage_account_type = "Premium_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-jammy"
    sku       = "22_04-lts-gen2"
    version   = "latest"
  }

  computer_name  = "hostname"
  admin_username = var.username

  admin_ssh_key {
    username   = var.username
    public_key =
jsondecode(azapi_resource_action.ssh_public_key_gen.output).publicKey
  }

  boot_diagnostics {
    storage_account_uri =
azurerm_storage_account.my_storage_account.primary_blob_endpoint
  }
}
```

5. Create a file named `variables.tf` and insert the following code:

Terraform

```
variable "resource_group_location" {
  type        = string
  default     = "eastus"
  description = "Location of the resource group."
}

variable "resource_group_name_prefix" {
  type        = string
  default     = "rg"
  description = "Prefix of the resource group name that's combined with
a random ID so name is unique in your Azure subscription."
}

variable "username" {
  type        = string
  description = "The username for the local account that will be cre-
ated on the new VM."
  default     = "azureadmin"
}
```

6. Create a file named `outputs.tf` and insert the following code:

Terraform

```
output "resource_group_name" {
  value = azurerm_resource_group.rg.name
}

output "public_ip_address" {
  value =
azurerm_linux_virtual_machine.my_terraform_vm.public_ip_address
}
```

Initialize Terraform

Run terraform init to initialize the Terraform deployment. This command downloads the Azure provider required to manage your Azure resources.

Console

```
terraform init -upgrade
```

Key points:

- The `-upgrade` parameter upgrades the necessary provider plugins to the newest version that complies with the configuration's version constraints.

Create a Terraform execution plan

Run terraform plan to create an execution plan.

Console

```
terraform plan -out main.tfplan
```

Key points:

- The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.
- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.
- To read more about persisting execution plans and security, see the security warning section .

Apply a Terraform execution plan

Run terraform apply to apply the execution plan to your cloud infrastructure.

Console

```
terraform apply main.tfplan
```

Key points:

- The example `terraform apply` command assumes you previously ran `terraform plan -out main.tfplan`.

- If you specified a different filename for the `-out` parameter, use that same filename in the call to `terraform apply`.
- If you didn't use the `-out` parameter, call `terraform apply` without any parameters.

Verify the results

Azure CLI

1. Get the Azure resource group name.

   Console

   ```
   resource_group_name=$(terraform output -raw resource_group_name)
   ```

2. Run az vm list with a JMESPath query to display the names of the virtual machines created in the resource group.

   Azure CLI

   ```
   az vm list \
     --resource-group $resource_group_name \
     --query "[].{\"VM Name\":name}" -o table
   ```

Clean up resources

When you no longer need the resources created via Terraform, do the following steps:

1. Run terraform plan and specify the `destroy` flag.

   Console

   ```
   terraform plan -destroy -out main.destroy.tfplan
   ```

   Key points:

   - The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.

- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.
- To read more about persisting execution plans and security, see the security warning section .

2. Run terraform apply to apply the execution plan.

Console

```
terraform apply main.destroy.tfplan
```

Troubleshoot Terraform on Azure

Troubleshoot common problems when using Terraform on Azure

Next steps

In this quickstart, you deployed a simple virtual machine using Terraform. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

Azure Linux virtual machine tutorials