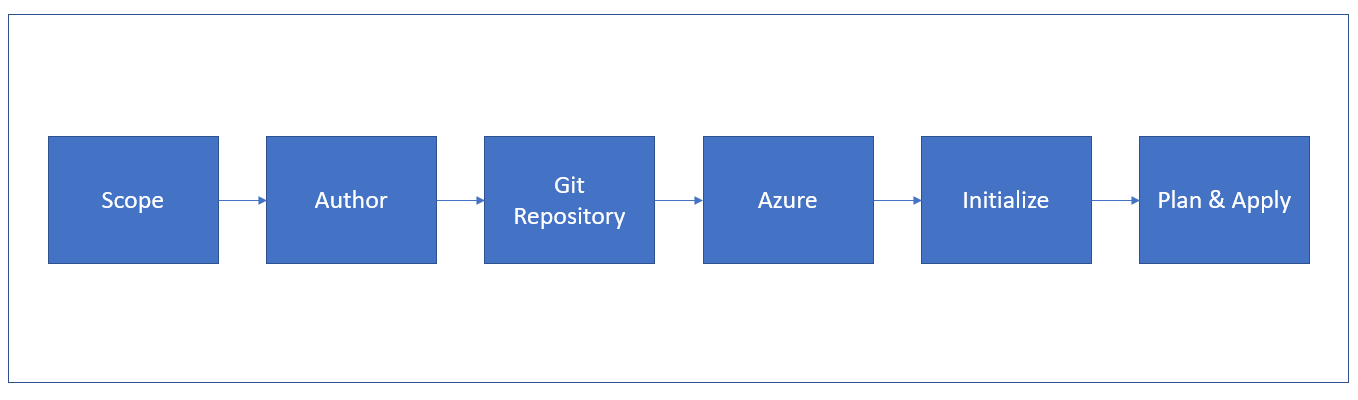
**Infrastructure as Code using Terraform**

As described in previous sections, Infrastructure as Code is the process of automating creation and management of cloud infrastructure for a particular project instead of manually configuring resources. A resource is any part of infrastructure such as a virtual machine, security group, network interface, etc. for a given environment.

**Terraform Workflow**



Workflow of terraform to build the cloud infrastructure will follow steps shown in block diagram above. The description of each step in the workflow is provided below,

**Scope:** In this step we identify the resources required for the project. In this particular project we required resources such as virtual machines, resource groups, network security groups, load balancer, etc.

**Author:** In this step we need to create configuration of identified resources using HashiCorp Configuration Language (HCL). Detailed overview of this process is provided in the ‘Building the Infrastructure with Terraform’ section of this report.

**Git Repository:** After creating the terraform configuration for the given project we need to add these files in the Git repository of our project.

**Azure:** We Need to link the project repository at GitHub with the azure account using which we are deploying our project. Detailed information about this process is mentioned in Deployment part of the report.

**Initialize**: To download the appropriate provider plug-ins for the project and initialize working directory with the terraform configuration files, we need to run terraform init command in the project directory containing terraform configuration files.

**Plan & Apply:** Execution of terraform plan command creates execution plant to verify infrastructure creation process. Also, terraform performs a refresh in order to determine the necessary actions to be performed to achieve the expected state of the infrastructure. This command helps us to know in advance that the infrastructure being created is matching with the expected infrastructure.

After successful verification of execution plan, we need to run terraform apply command to build real infrastructure with specified resources. It also creates a state file that stores existing state of infrastructure and compares configuration modification files with existing configuration of deployment environment.

In the coming sections will provide information about how we used terraform in our project.

**Prerequisites**

1. **Azure Subscription**

We need a valid azure subscription to deploy the project at cloud. In this project we are using Azure for Students subscription.

1. **Install the Azure CLI tool**

We need to install Azure CLI tool to create and manage Azure resources from PowerShell. In order to install Azure CLI tool we need to open PowerShell prompt as an administrator and run the command:

Invoke-WebRequest -Uri https://aka.ms/installazurecliwindows -OutFile .\AzureCLI.msi; Start-Process msiexec.exe -Wait -ArgumentList '/I AzureCLI.msi /quiet'; rm .\AzureCLI.msi

1. **Install Terraform**

In order to use terraform, install the terraform as a binary package (version 0.14.5) from official website of Terraform.

1. **Authenticate using the Azure CLI**

After successful installation of Azure CLI tool, next step is to link and authenticate Azure CLI tool with our Azure account. Terraform required authentication from Azure to create infrastructure. For this we need to log into azure account locally from the terminal using Azure CLI tool by running command:

az login

This will open your browser window and ask you to enter your Azure login credentials. After successful authentication, terminal will show your subscription information. More detailed about this step are described in Deployment part of this report.

**Building the Infrastructure with Terraform**

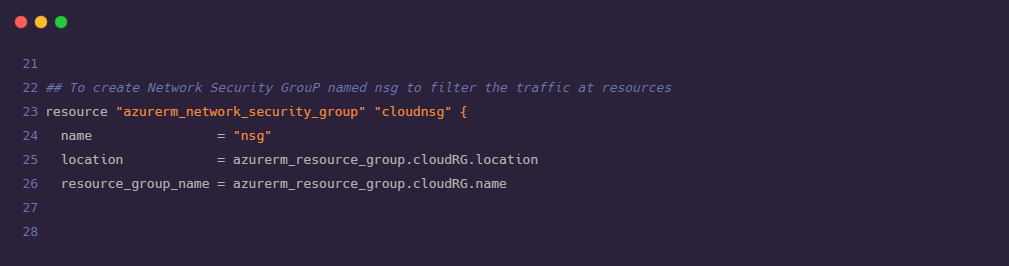
To run our project in azure, we need to create required infrastructure for the web application using terraform. We need to follow below steps to create the infrastructure,

We have written configuration for all the identified resources in main.tf file. We have defined some repeatedly used variables in terraform.tfvars and variables.tf files. Let us look at all resource step by step,

1. We need to define the service provider as azurem and its version that is 2.0. To create infrastructure, we need a resource group to hold all project related resources. We defined an Availability set which is logical grouping for isolation virtual machine resources from each other after deployment.



1. We need to create Network Security Group (NSG) to filter the traffic to and from resources in azure.



1. It is very important to define the rules to filter the traffic. For our project we have created four different rules to serve following purposes,

i) Allowing Ansible to connect with the Virtual Machines

ii) Allowing Visual Studio installed in our local machine to interact with azure to deploy web application

iii) Allowing web clients to connect to our web application

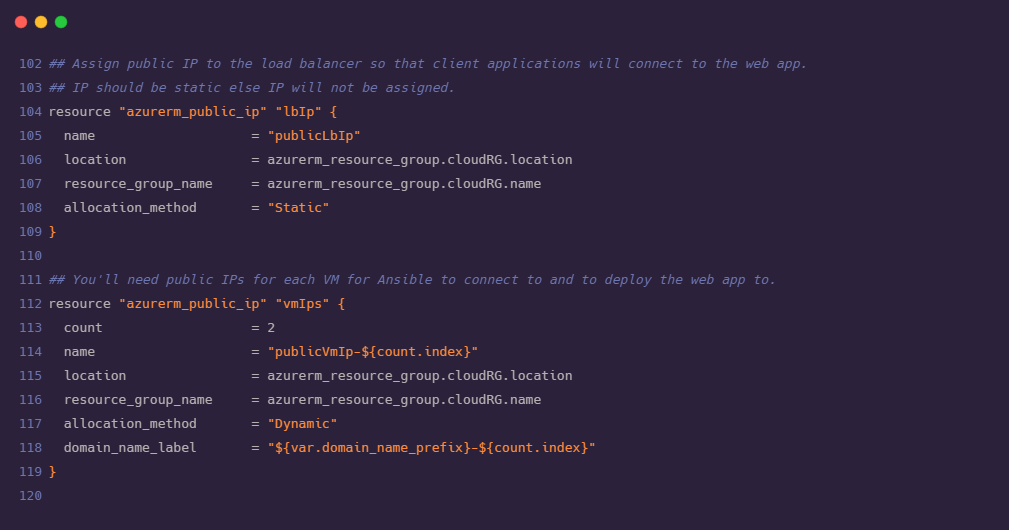
iv) Optional rule if we need Remote Desktop Protocol (RDP) for troubleshooting purpose



1. Now we need to define the infrastructure for the Virtual Network which is representation of our network in cloud. We also defined subnets under the Virtual Network.



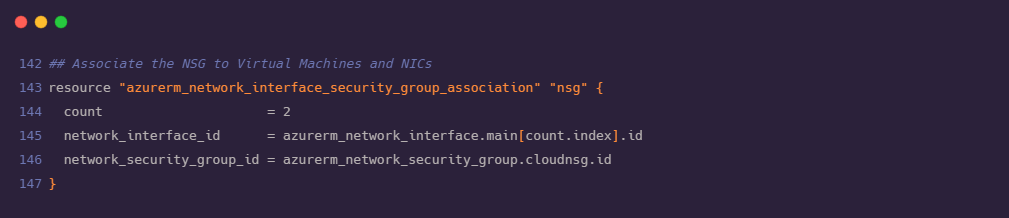
1. We need to assign static IP address to the load balancer in advance. Also, need to assign public IP address for each virtual machine that we are deploying. We require two virtual machines for this particular project.



1. As we have two virtual machines declared in fifth step. Now, it is important for every virtual machine to connect to the network. For this we need to create virtual Network Interface Cards (vNIC) for each virtual machine. vNIC severs as a gateway between Network and VMs. It is required to provide IP configuration to each of the Network Interface Card. Before creating vNIC it is mandatory to ensure that our subnet is up and running.



1. We need to apply the NSG rules that we have created in second and third steps to the VMs and NICs in our network.



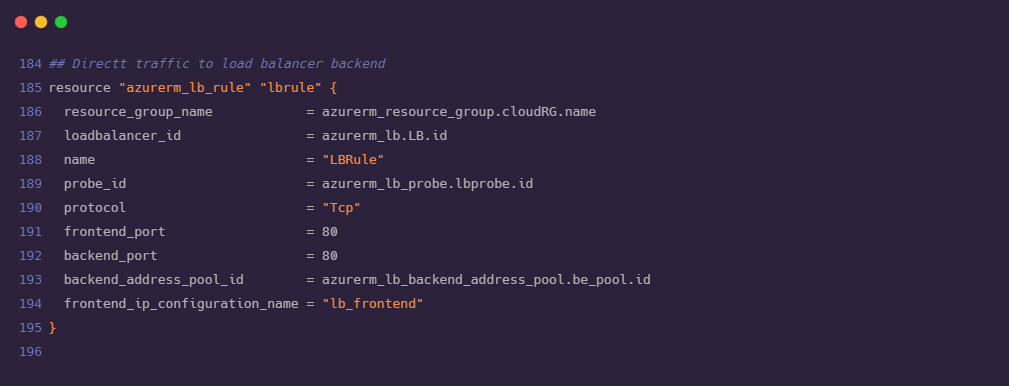
1. Now we are ready with the declaration for the creation of VMs. We need load balancer to rout the traffic to our VMs in such way that traffic would be diverted towards most healthy VM and avoid traffic to failed VM. As Load balancer is acting as an mediator between outside traffic and VMs, we need to configure Frontend and Backend of load balancer where backend part will consist of NICs and VMs.



1. As described in the previous step, load balancer diverts the traffic to the most healthy VM in the network. For this we need to configure the health probe for VMs this will periodically check on both VMs for an open port 80.



1. Using the health probe defined in above step we need to create a rule to divert the incoming traffic on port 80 to the healthy VM available in backend.



1. As an important step we need to associate VMs and vNICs created in earlier steps



1. We need to install custom script VM extension on each virtual machine. This will download the ConfigureRemotingForAnsible.ps1 script from GitHub. Execution of this script will open the Windows Remote Management (WinRM). Using WinRM Ansible connects with Azure Cloud Shell.



1. After declaring all the required resources in main.tf file, we are now ready with our terraform code for the deployment of our web application. Before proceeding to understand ansible configuration and deployment steps, we also need to look after terraform.tfvars and variables.tf file where we have stored values for frequently used variables in order to maintain consistency. We can modify the values of variables from these files without affecting the main configuration.



