## Introduction

In this Hands On Lab you will deploy a very simple Proof of Concept architecture using Terraform which includes:

* Management Linux Server accessible via SSH
* Two Web Servers in a VM Scale Set published via Azure Load Balancer
* The Web Servers will be built via an Ansible playbook called as part of the build process

The picture below gives a high-level view of the deployed architecture:



## Prerequisites

To complete this Lab you will need;

* A current Azure Subscription
* Owner rights on the subscription \*or\* be given a valid Service Principal to use for the Terraform deployment

### LAB Variables

Prior to starting the Lab complete the table below which details variables you will need to use throughout the lab. If you are in a guided Lab session your guide will help you identify some of these values.

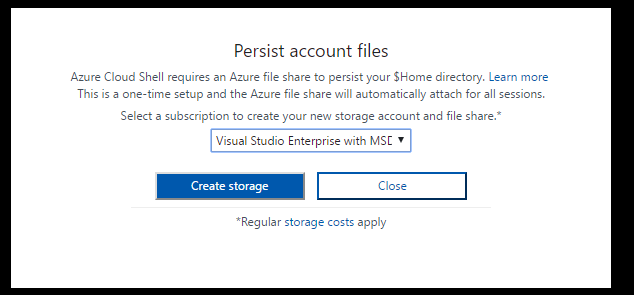
|  |  |  |
| --- | --- | --- |
| Item | Description | Value |
| LAB Subscription | The Azure subscription you will be using for this Lab |  |
| STUDENT ID | A unique number between 1 and 254 for use in this lab, should not be shared with anyone else. *Note no leading zeros, e.g., “01” should be “1”* |  |
| LAB Admin Password | The password you will use for administrator in this Lab | *Make it something memorable and avoid writing it down if possible. Should be a min of 12 charachters* |
| LAB Location | Which Azure Region you will use to deploy resources as part of this Lab |  |

### Create Storage Accounts

In this section we will be deploying the Storage Account which we will use to store our Terraform state files. Details on Azure Storage can be found at: <https://docs.microsoft.com/en-us/azure/storage/storage-introduction>

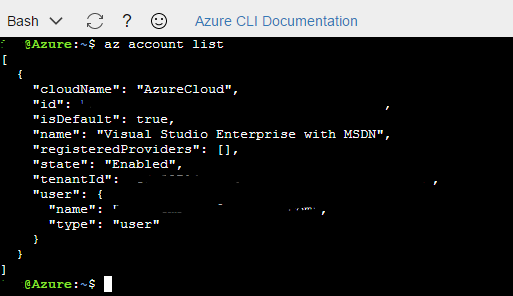
1. Open the *Cloud Shell* from the portal (<https://portal.azure.com>)
   1. The cloud shell will open in the bottom of the portal, if prompted click **Create storage**
   2. Details on how you can use the Azure Cloud Shell storage can be found here: <https://docs.microsoft.com/en-gb/azure/cloud-shell/persisting-shell-storage>





1. In case you have access to multiple Azure Subscriptions lets check we are going to be working on the correct subscription:
   1. Type

az account list



* 1. If you have just one account listed and it matches the one for this lab you’re all sorted, if not no worries simply run the following command (replacing with your subscription name)

az account set –-subscription <LAB Subscription>

1. Make a note of your *id* from the output of the above command – this is your subscription id
2. Next we will create a resource group to hold our storage account run the following command (replacing with your values for the Lab)

az group create -l <LAB Location> -n <STUDENT ID>-rgTFLabState

1. Now we will create the storage account itself by running the command below (using your own values). This command creates a new storage account within the resource group created above, in your chosen Lab region using Standard Locally Redundant Storage (LRS)

az storage account create --resource-group <STUDENT ID>-rgTFLabState --name <STUDENT ID>satflabstateinfra --sku Standard\_LRS --location <LAB Location>

1. We now need to make a note of the access key for the storage account we retrieve the access keys for the storage account with the following command (note how we use -g and -n as short hand for --resource-group and --name) – store these keys somewhere you can find them later (we want the **value** for key 1):

az storage account keys list -n <STUDENT ID>satflabstateinfra -g <STUDENT ID>-rgTFLabState

### Create Service Principal

In this section we will create the Service Principal that Terraform will use to deploy our Azure Resources. More information on Service Principals can be found here: <https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-application-objects>

1. Create the Azure Active Directory account and assign it to the built-in Contributor role – details of the default Role Based Access Control options are available at: <https://docs.microsoft.com/en-us/azure/active-directory/role-based-access-control-what-is>. In the *Cloud Shell* window type the following command:

az ad sp create-for-rbac --name <STUDENT ID>-terraform --role Contributor

1. Copy the output (JSON) of the command and store it somewhere safe for later on in the Lab

### Terraform

Now we have set up the storage to hold our Terraform state file and the Service Principal for Terraform to use to manage our Infrastructure we can start playing with Terraform itself.

1. Download Terraform from <https://www.terraform.io/downloads.html>
2. Download the Terraform files for this Lab from: <https://github.com/Olivier-Subramanian/AzureTerraformAnsibleLab>
3. Open a Powershell or Terminal window and navigate to your home / personal area
4. Create a folder called *tflab*
   1. E.g. on Windows:

Cd Documents

Mkdir tflab

* 1. For Mac:

Cd ~/

Mkdir tflab

1. Copy (and if required extract) the downloaded terraform file to the new location and the contents of master.zip to the new location created above
2. From the terminal window *cd* into your tflab directory
3. Depending on your environment you should be able to execute terraform by either entering:
   1. terraform
   2. .\terraform
   3. ./terraform (Mac)
4. Open your preferred text editor, e.g., Notepad or Visual Studio Code
5. Open *c:\tflab\vars.tf*
6. As discussed in the Service Principal section, Terraform needs to know which subscription and username / password to use in order to manage your Azure infrastructure. In our Lab these details are stored in the *vars.tf* file we are now editing. Populate the *subscription\_id*, *client\_id*, *client\_secret*, *tenant\_id, admin\_password* and *StudentId* with your values.
   1. The *client­\_id* is the *appId* from when you created the Service Principal earlier
   2. The *client\_secret* is the *password* from the Service Principal you created
   3. *tenant\_id* was also output in the Service Principal creation
   4. Your *subscription\_id* you stored earlier when creating the storage account
   5. Your *StudentId* is hopefully well known to you by now!
   6. *admin\_password* is your Lab password
7. Save the vars.tf file
8. Update the main.tf file to reference your storage account name and resource\_group\_name
   1. Storage\_account\_name is your subscription\_id
   2. Resource\_group\_name is the resource group you specified earlier.
9. Save the main.tf file
10. At your terminal window run the command below to set up your Terraform environment with the state file stored in Azure. You will need to replace <ACCESS KEY> with the value you noted down earlier (Key 1 for <STUDENTID>satflabstateinfra)

.\terraform init -backend=true -backend-config=”access\_key=<ACCESS KEY>”

Make sure that you select no at the prompt.

1. We need to ensure Terraform has access to the Terraform modules we are using for this deployment - from the command line enter the command below:

.\terraform.exe get (./terraform get)

1. Next we get Terraform to compare the existing Azure estate to what the Terraform file requires:

.\terraform.exe plan

1. Finally we ask Terraform to deploy our requested infrastructure

.\terraform.exe apply

This process will take some time to complete so lets take a moment to review the Portal and make things a bit easier to find going forward.

1. On the menu on the left of the screen click on *More services >* at the bottom of the list
2. This opens a list of all the services that are currently available in the Portal.
3. Scroll up and down the list and you will notice that some services have stars against them – these items appear in the menu bar by default.
4. While the deployment runs ensure the following are in your menu:
   1. Resource Groups
   2. Virtual machines
   3. Storage accounts
   4. Virtual networks
   5. Virtual Machine Scale Sets
   6. Load Balancers

Now that you have the portal lined up to help you find things lets have a quick peek inside the Terraform files to see what’s going on here.

Inside the \tflab folder there is a file called *main.tf* this is the file which the *Terraform apply* command is currently working through, at the top of the file we have the backend definition:

terraform {

backend "azure" {

storage\_account\_name = "1satflabstateinfra"

container\_name = "dev-tfstate"

key = "dev.terraform.tfstate"

}

}

This instructs Terraform to store the state of the Azure infrastructure it is managing in an Azure storage account, when working with CI/CD pipelines or in a team environment centralised state is a useful capability and Azure is a supported backend. More information on Terraform backends can be found at: <https://www.terraform.io/docs/backends/index.html>

The next section configures the Subscription and user that Terraform will be using to work against Azure:

# Configure the Microsoft Azure Provider

provider "azurerm" {

subscription\_id = "${var.subscription\_id}"

client\_id = "${var.client\_id}"

client\_secret = "${var.client\_secret}"

tenant\_id = "${var.tenant\_id}"

}

In this example we are pulling the details from the *vars.tf* file you created. As this has sensitive information you may want to instead generate these details from Environment variables, or dynamic scripts, for example in a Jenkins job using the Azure CLI to request the *client\_secret* from Azure Keyvault and either storing it in an Environment variable or supplying as a command line argument, or even just writing out a temporary vars file.

So at this point we have configured Terraform to store its state file in an Azure Storage Account and provided it with the information it needs in order to work against Azure. Our simple example makes use of 3 modules:

* Network, which sets up the network for Lab and manages it within a resource group
* Singlevm, which builds a single virtual machine and homes it within a resource group
* Vmscaleset, which builds out a Virtual Machine Scale Set, again homed within its own resource group

If you continue reading the *main.tf* file you can see we call each of these modules and provide some variables for them to use in building out the infrastructure.

### Ansible

OK so where is the Ansible part of this Lab? If you now open up *main.tf* within the */modules/vmscaleset* folder you will see the resource definitions to support a Web Server scaleset.

Within this file near line 100 is a Custom Script Extension, this downloads a shell file from GitHub and then executes it:

"fileUris": ["https://raw.githubusercontent.com/ianalderman/HOLAzureTerraformAnsible/master/scripts/provsvr.sh"],

"commandToExecute": "bash provsvr.sh"

This shell script installs Ansible & GIT on the webserver instance before calling an Ansible Playbook via the command below:

ansible-pull -d /etc/ansible/init -U <https://github.com/Olivier-Subramanian/holansiblepull.git>

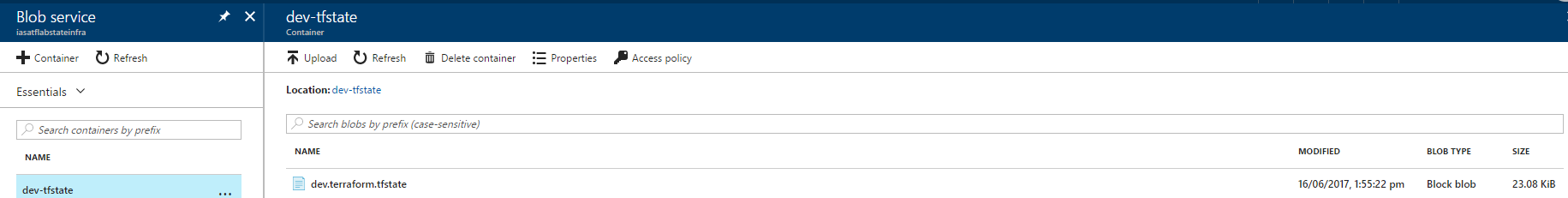
This playbook which can be seen here: <https://raw.githubusercontent.com/Olivier-Subramanian/holansiblepull/master/local.yml>, installs Apache and configures it listen on port 8081 **(not port 80).**

Once the Terraform apply has finished you should be able to see the default website via browsing to the IP address of your scale set. You can find this by searching for your load balancer (it will start with your student id!).

**N.B. In my testing the website was actually up a few minutes before the deployment finished, this is likely a delay waiting for the script extension to report completion on both nodes.**

How is it that you can access the test site on port 80 (HTTP), when it is published on port 8081? If you have another look at the vmscaleset module you will see we define a load balancing rule which maps the internet facing port to the internal one.

## Find your state file

OK so the deployment has now finished, if you use the portal to browse your storage account you should see something like below: 

This is the result of the *backend* command we ran earlier which ensures that the Terraform managed state is stored in a central location.

## Tidy Up

Congratulations on completing the Lab! One very important last step – delete the resource groups you have created in this Lab to ensure you do not run up an unexpected bill!

You can run

.\terraform.exe destroy

To clear up all the infra you deployed with Terraform, you will need to use the portal or CLI to tidy up the resource group for the storage account and the Service Principal.

## A note on Resource groups in this Lab

You will have noticed during this lab we made use of a number of different resource groups. The advantage of doing so is to enable resources that share a common lifecycle to be grouped and administered together, you could also apply differing security permissions to each group through the use of the RBAC (Role Based Access Control) groups. For example you could use the Resource groups in this Lab and RBAC to ensure only the Network Team could make any changes to the network, whilst each development team had rights to their servers.

An impact of using multiple resource groups is that as a deployment is targeted at a resource group you need more deployments. If we were to build this Lab out using an ARM template we would need 3 – one for the Network, one for the management box and one for the web servers. If we used a single resource group we could deploy everything in one pass.

Operationally there are benefits to the this segregation – project team A can deploy to their resource groups without risk to another team, where you have built out DR or A/B deployments separating them in to separate resource groups also prevents you taking out both in one go. What is the best route for you will depend on your unique Business requirements – but something to think about!

Further information on resource groups can be found at: <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview>