Infrastructure Build Automation

Asghar Ghori
Humber College Institute of Technology & Advanced Learning
Toronto, Ontario, Canada ©2023
www.humber.ca www.nixeducation.com

Roadmap for Part 1

- **Infrastructure Provisioning with Terraform**
  - Module 1: Introduction to Terraform
  - Module 2: Manage Single Infrastructure Resources
  - Module 3: Parametrize Configuration (Input Variables)
  - Module 4: Parametrize Configuration (Locals and Output Values)
  - Module 5: Set Lifecycle Rules and Dependencies
  - Module 6: Manage Multiple Infrastructure Resources
  - Module 7: Perform Guest OS Customization
  - Module 8: Switch to Remote Backend + Miscellaneous Stuff
  - Module 9: From Flat Architecture to Modular Architecture

Module 1

Introduction to Terraform

Infrastructure as Code (IaC)

- Process of **managing infrastructure using code** rather than manually configuring resources in a user interface
- Define, deploy, update, and destroy infrastructure
- Benefits:
  - Consistency across test, development, QA, and production states
  - Version controlled (can be stored in Git)
  - Repeatability (reusability) [Team collaboration]
  - Validation before deployment
  - No errors
  - Fast deployments
  - Detailed logging

IaC Categories

- **Ad-hoc scripts**
  - Step-by-step instructions
  - **Tools**: Bash, Ruby, Python, Go, etc.
- **Configuration management**
  - Designed to run on **existing** servers
  - **Tools**: Ansible, Puppet, Chef, and SaltStack
- **Provisioning**
  - Builds any infrastructure component: networking, VM, container, database instance, load balancer, firewall rule, etc.
  - **Tools**: Terraform, Azure Resource Manager (ARM), Bicep, AWS CloudFormation, Google Cloud Deployment Manager (CDM), OpenStack Heat, etc.

Popular IaC Tools

Tool	Type	Infrastructure	Language	Master	Agent	Source	Cloud	Launched
CloudFormation	Provisioning	Immutable	Declarative	No	No	Closed	AWS	2011
Heat	Provisioning	Immutable	Declarative	No	No	Open	All	2012
Terraform ✓	Provisioning	Immutable	Declarative	No	No	Open	All	2014
ARM	Provisioning	Immutable	Declarative	No	No	Closed	Azure	2014
CDM	Provisioning	Immutable	Declarative	No	No	Closed	GCP	2015
Puppet	Config Mgmt.	Mutable	Declarative	Yes	Yes	Open	All	2005
Chef	Config Mgmt.	Mutable	Procedural	Yes	Yes	Open	All	2009
SaltStack	Config Mgmt.	Mutable	Declarative	Yes	Yes	Open	All	2011
Ansible ✓	Config Mgmt.	Mutable	Procedural	No	No	Open	All	2012

Mutable objects **can** be modified or updated after they have been created. Changes and updates are made directly to the existing components.

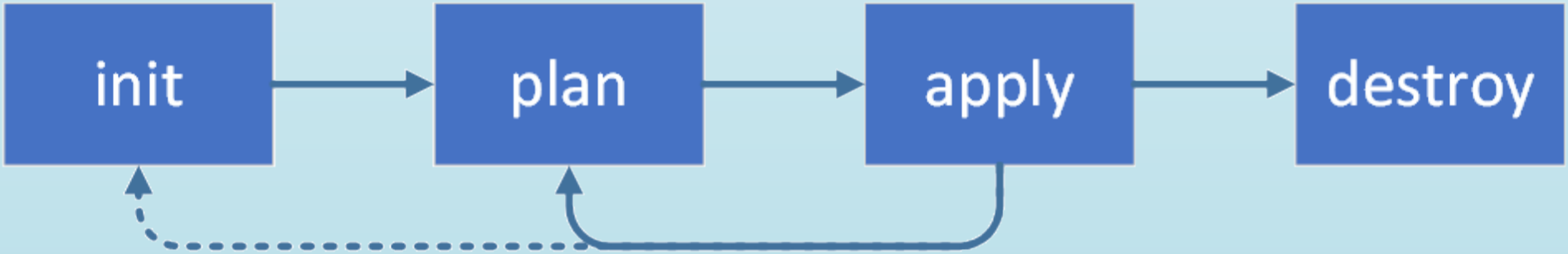
Immutable objects **cannot** be modified or updated after they have been created. New objects are created to reflect changes or updates (and to replace the old ones).

What is Terraform?

- IaC tool from HashiCorp
- Open source, immutable, declarative
- Code written in **HashiCorp Configuration Language** (HCL)
- Used to **build, change, and manage** infrastructure (VM, disk, security group, network interface, load balancer, etc.)
- You define your **desired end-state** infrastructure in a file
- Builds the **desired** infrastructure in the **specified** environment
- Leverages CSP's APIs and their authentication mechanisms
- Shows what will be created, changed, and deleted (**dry run**)

Terraform Workflow

- Scope, author, initialize, validate, plan, apply, and destroy



Terraform Community Help

Terraform Help on StackOverflow

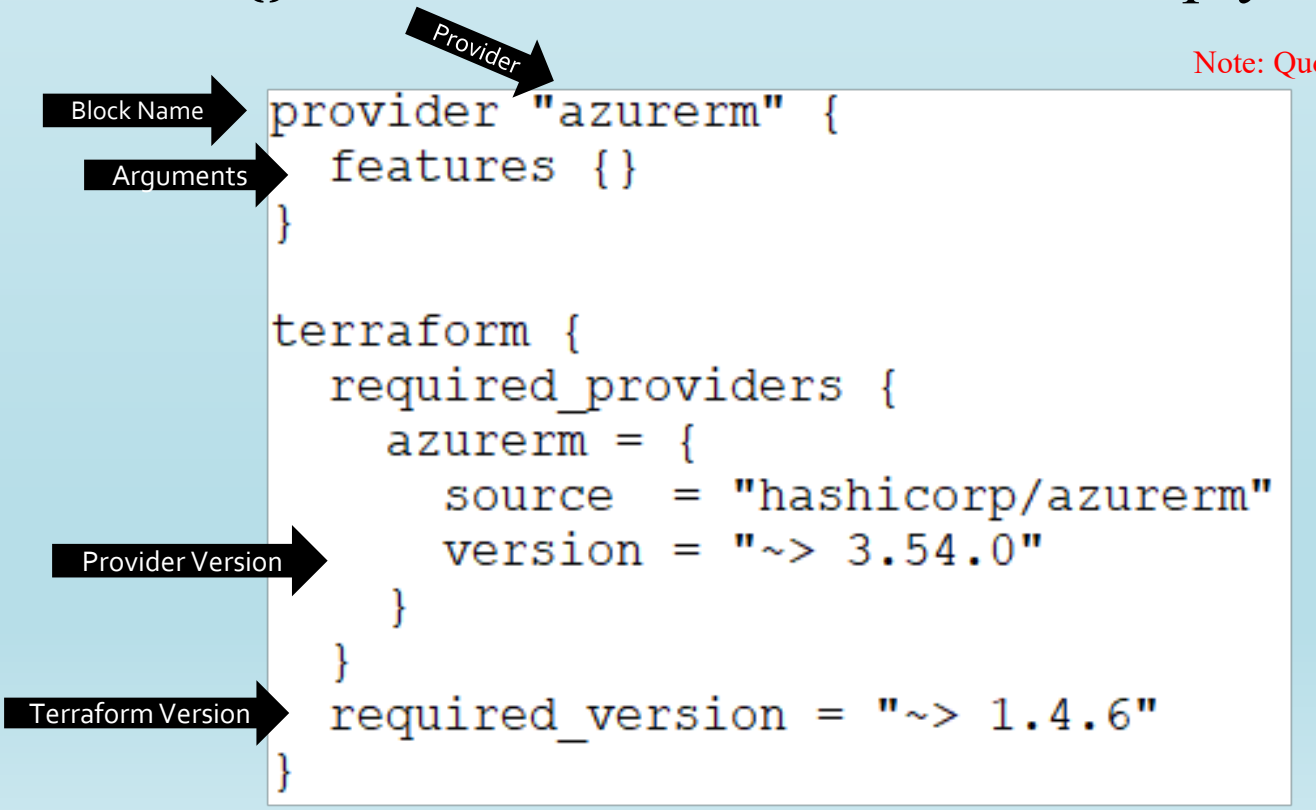
Terraform Issues on GitHub

Terminology

- **Terraform core**: The Terraform binary which serves as the root command
- **Plug-in**: The interface between Terraform core and provider to get a job done
- **Providers**:
  - Sets of APIs to interact with the target platform
  - Only supported Terraform plug-in at the moment
  - Examples: azurerm, aws, google, kubernetes, alicloud, oci, etc.
  - Three types:
    - **Official Providers**: Maintained by HashiCorp internally (azurerm, aws, google, and Kubernetes)
    - **Verified Providers**: Not maintained by HashiCorp (alicloud and oci)
    - **Community Providers**: Numerous
  - "terraform init" downloads them

Terminology: Provider Block

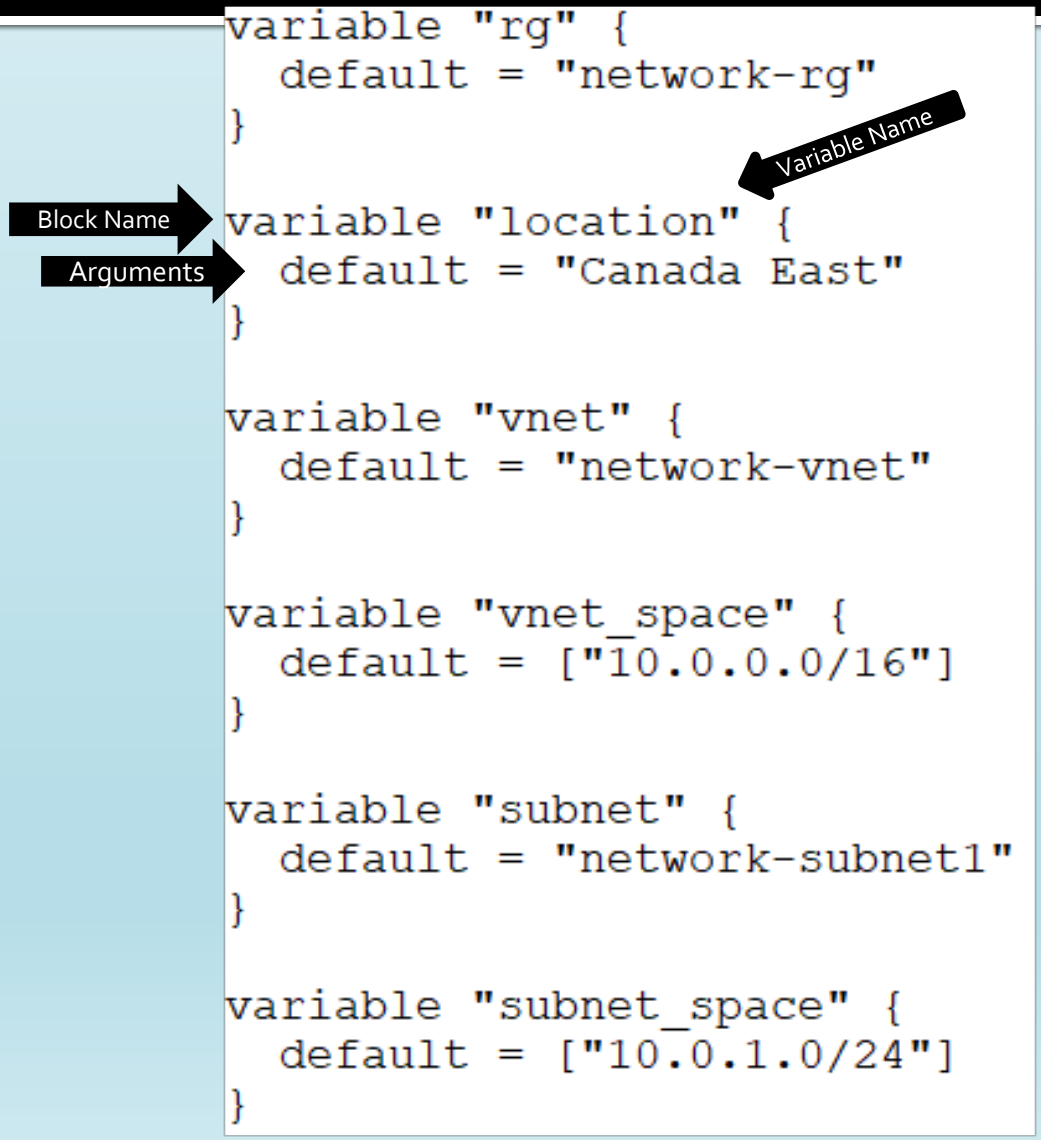
- Data to **describe** the provider
- Can be defined in **providers.tf** file
- **features {}** must be defined even if it is empty



Note: Quotation marks are optional

Terminology: Variable Block

- Key/value pairs to identify resource configuration
- Can be defined in **variables.tf** file



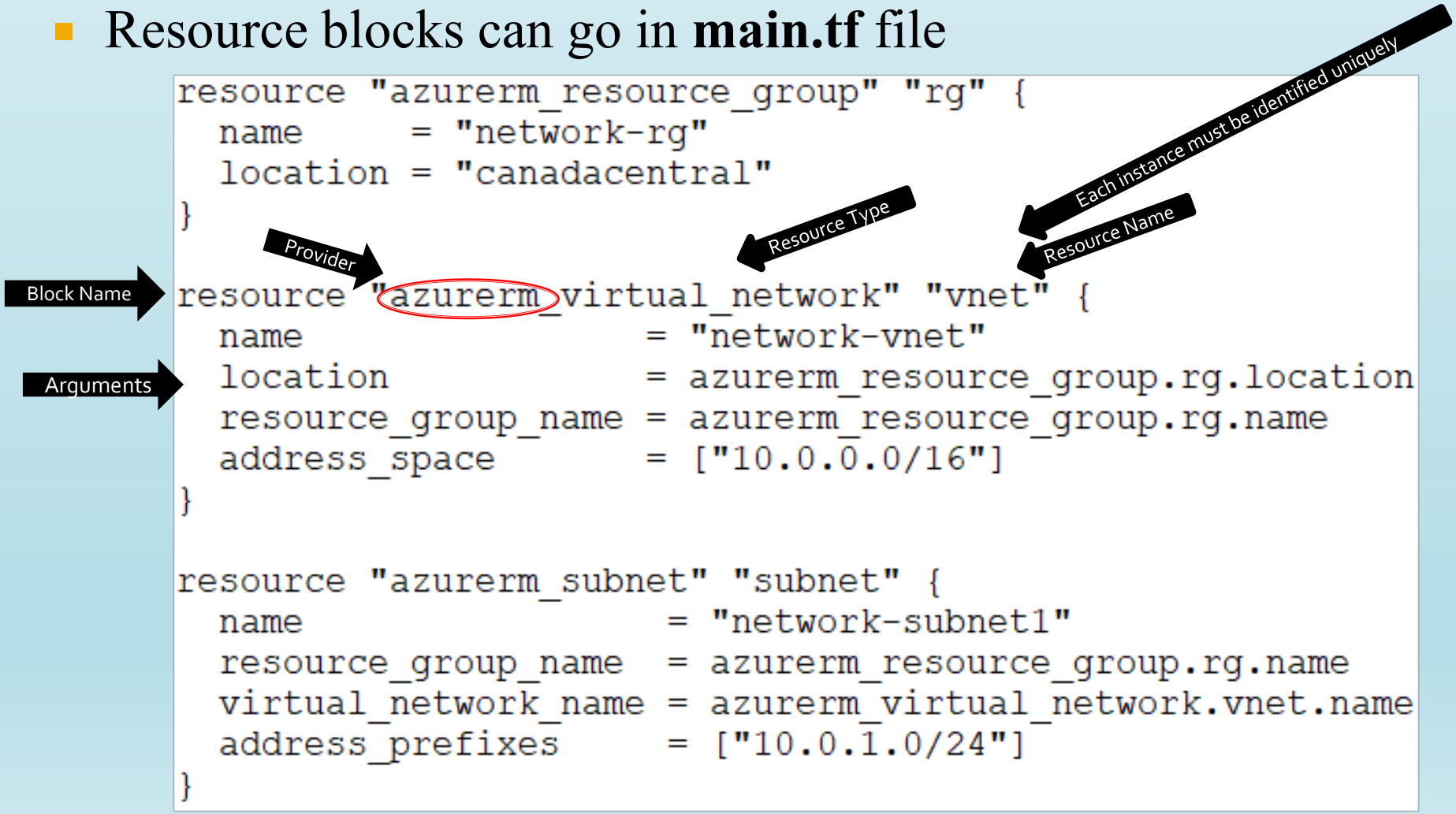

```
variable "rg" {
  default = "network-rg"
}

variable "location" {
  default = "Canada East"
}

variable "vnet" {
  default = "network-vnet"
}

variable "vnet_space" {
  default = ["10.0.0.0/16"]
}

variable "subnet" {
  default = "network-subnet1"
}

variable "subnet_space" {
  default = ["10.0.1.0/24"]
}
```

Terminology: Resource Block

- **Resource block**: key/value pairs to define a resource; must include all **required** configuration items
  - **Resource**: Describes a **single** object (RG, VM, data disk, extension, role, NIC, NSG, Public IP, recovery services vault)
  - **Resource type**: resource-specific (azurerm_resource_group)
  - **Resource name**: **Unique** name assigned to a resource type for Terraform internal use (“azurerm_resource_group” “vm1”)
  - **Resource dependency**: implicit or explicit

Terminology: Sample Resource Blocks

- Resource blocks can go in **main.tf** file



Suggested Terraform Filenames

- **providers.tf**: Stores provider information
- **variables.tf**: Parametrizes resource configuration
- **main.tf**: Resource definitions, dependencies, lifecycle rules, etc.
- Other suggested file naming:
  - network-vars.tf, vmwindows-vars.tf, vmlinux-vars.tf, etc.
  - network-main.tf, vmwindows-main.tf, vmlinux-main.tf, etc.

Note: Terraform files can be in json format as well. In that case, file extensions will be *.tf.json.

Tour of Terraform Website

Terraform Website Tour

Learning and Coding Guidelines

- There is **no perfect solution** to any problem. You can do the same thing using **different** techniques. Hence, there is **no solution** presented for the labs in this course.
- Best way to learn is to **try** things yourself
- **Do not panic** if something does not work: investigate, ask, try
- Do not be afraid to **look up** anything you want for additional information
- Free account can do **most** things, but **not** everything

- Write Terraform code with the **user** of the code in mind (not the author or the maintainer)
- **Simplify** as much as possible

Alerts

- Each lab is built upon the previous lab. You **must** complete a lab **before** moving on to the next one.

- Perform all the labs and complete them **on time** if you:
  - Do not want to be lagged behind
  - Do not want your marks deducted

Module 2

Manage Single Infrastructure Resources

Overview of TF Commands

- The root command is **terraform**
- Sub-commands:
  - **init**: Initializes the settings; pulls and installs required provider plug-ins under a hidden directory in the current directory; must be re-run each time a new provider is added.
  - **fmt**: Formats TF files to look neat and clean. Outputs the list of files it formats. Always a good idea to run it after TF files have been modified.
  - **validate**: Validates TF files for syntax errors, typos, inconsistencies, etc.
  - **plan**: Generates and shows the execution plan (dry run)

Overview of TF Commands contd.

- **apply**: Builds or changes infrastructure
  - (1) shows the execution plan (2) deploys the infrastructure
- **destroy**: Destroys Terraform-managed infrastructure
- **show**: Displays a detailed view of the current deployment state
- **state**: Lists the deployed resources
- **output**: Shows names, addresses, IDs, etc. of deployed resources
- **providers**: Prints the names and versions of the providers used
- **version**: Prints provider and terraform version information
- **taint/untaint**: Marks/unmarks a resource for recreation
- **import**: Imports existing infrastructure into Terraform (requires too much manual work)

Usage Help

- General usage and description for all subcommands:

  $ **terraform -help**

- Detailed help on individual subcommands:

  $ **terraform -help plan**

A Word on Resource Naming

- Naming should be meaningful and follow organizational rules
- Identify basic/key information about the resource
- Restricted to alphanumeric plus hyphen and underscore
- Also consistent across all Terraform resources, variables, and modules
- Suggestions for naming virtual machines:
  - <purpose>-<dept>-<app>-<env>-<OS>-vm001
    - Purpose (max 3): chs, tes, etc.
    - Dept (max 3): fin, cld, sal, mkt, eng, res, sci, etc.
    - App (max 3): sap, ora, mdw, web, etc.
    - Env (max 3): tst, lab, dev, qa, ppd, prd, etc.
    - OS (max 1): w, u, r, c, s, etc.
  - chs-fin-sap-prd-u-vm01
  - bear-res-dev-w-vm01
  - mdl3-eng-tst-r-vm01

A Windows virtual machine and hostname cannot be more than 15 characters in length

Lab02s1: Provision, Modify, and Destroy Basic Networking Resources

- See Lab02 Section 1

Module 3

Parametrize Configuration (Input Variables)

Input Variables

- Key/value pairs for resource customization (name, location, instance size, disk attributes, etc.)
- Avoid hard coding resource values by parametrizing them
- Declare, define, and consume variables
- A variable block has a **name** plus the following:
  - **Type**:
    - **Primitive**: string, numeric, and bool (simple value)
    - **Complex**: (a group of multiple values)
      - **Collection type**: list and map (**similar** values)
      - **Structural type**: object and tuple (**dissimilar** values)

Input Variables contd.

- **Default**: a literal value
- **Description**: general information about the variable
- **Sensitive**: **True** or **false** value to tell whether the variable's value is to be revealed

“String” Variable Example

- One variable block with one value

```
variable "rg" {
  default = "network-rg"
  type = string
description = "This resource group is to
}

variable "location" {
  default = "Canadacentral"
}

variable "vnet" {
  default = "network-vnet"
}

variable "vnet_space" {
  default = ["10.0.0.0/16"]
}

variable "subnet" {
  default = "network-subnet1"
}

variable "subnet_space" {
  default = ["10.0.1.0/24"]
}
```

```
resource "azurerm_resource_group" "rg" {
  name     = var.rg
  location = var.location
}

resource "azurerm_virtual_network" "vnet" {
  name                = var.vnet
  location            = var.location
  resource_group_name = var.rg
  address_space       = var.vnet_space
}

resource "azurerm_subnet" "subnet" {
  name                 = var.subnet
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = var.subnet_space
}
```

“List” Variable Example

- A sequence of values of the same type
- “string” value for all items in the list
- Use case: list of virtual machines, users, disk devices, regions, availability zones

```
variable "username" {
  type = list
  default = [“user1", "user2", "user3"]
}


username = var.username[0]
```

“Map” Variable Example

- A lookup table that matches keys to values
- Used to group **related** items for the same resource
- Use case: disk values, OS image information

```
variable "los_disk_attr" {
  type = map(string)
  default = {
    los_storage_account_type = "Premium_LRS"
    los_disk_size            = "32"
    los_disk_caching         = "ReadWrite"
  }
}
```

```
os_disk {
  [redacted]
  caching              = var.los_disk_attr["los_disk_caching"]
  storage_account_type = var.los_disk_attr["los_storage_account_type"]
  disk_size_gb         = var.los_disk_attr["los_disk_size"]
}
```

“Object” and “Tuple” Variable Examples

- **Dissimilar** values
- **Object:** A lookup table, matching a fixed set of keys to values of specified types
- **Tuple**: A fixed-length sequence of values of specified types
- Use case: different types of information

```
{
  name = "John"
  age  = 52
}
```

Object

```
["a", 15, true]
```

tuple

Lab02S2: Convert Previous Lab to Use Parametrized Values

- See Lab02 Section 2

Virtual Machine Resources

- Linux or Windows virtual machine
- Resources that are created and attached to a VM:
  - **Network Interface Card (NIC):** Required
  - **Private IP:** Automatically assigned from the subnet
  - **Public IP:** For external access (optional)
    - Dynamic or Static
  - **Disk for operating system:** Windows 128GB / Linux 32GB

Virtual Machine OS Image

- Publisher, offer, sku (stock keeping unit), and version

```
variable "linux_publisher" {
  default = "Canonical"
}
variable "linux_offer" {
  default = "UbuntuServer"
}
variable "linux_sku" {
  default = "19.04"
}
variable "linux_version" {
  default = "latest"
}
```

```
source_image_reference {
  publisher = var.linux_publisher
  offer     = var.linux_offer
  sku       = var.linux_sku
  version   = var.linux_version
}
```

- To generate and save the list of all available OS images:

  az vm image list --all > vmimages

Terraform "File" Function

- Define a variable block for ssh public key (Linux VM)

```
variable "pub_key" {
default = "~/.ssh/id_rsa.pub"
}
```

- Use the variable in Linux VM resource block

```
admin_ssh_key {
  public_key = file(var.pub_key)
  username   = var.linux_admin_user
}
```

- **Note:** Do NOT use a password with a Linux username; use an **ssh keypair** instead.

Virtual Machine Resource

```
resource "azurerm_linux_virtual_machine" "vmlinux" {
  name                = var.linux_name
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  network_interface_ids = [azurerm_network_interface.linux_nic.id]
  computer_name       = var.linux_name
  size                = var.vm_size
  admin_username      = var.linux_admin_user

  admin_ssh_key {
    username   = var.linux_admin_user
    public_key = file(var.pub_key)
  }

  os_disk {
    name                 = "${var.linux_name}-os-disk"
    caching              = var.los_disk_attr["los_disk_caching"]
    storage_account_type = var.los_disk_attr["los_storage_account_type"]
    disk_size_gb         = var.los_disk_attr["los_disk_size"]
  }

  source_image_reference {
    publisher = var.linux_publisher
    offer     = var.linux_offer
    sku       = var.linux_sku
    version   = var.linux_version
  }

}
```

NIC and Public IP Resources

```
resource "azurerm_network_interface" "linux_nic" {
  name = "${var.linux_name}-nic"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  ip_configuration {
    name                          = "${var.linux_name}-ipconfig"
    subnet_id                     = azurerm_subnet.subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.linux_pip.id
  }
}

resource "azurerm_public_ip" "linux_pip" {
  name                = "${var.linux_name}-pip"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  domain_name_label   = var.linux_name
  allocation_method   = "Dynamic"
}
```

Lab02S3: Deploy Virtual Machine

- See Lab02 Section 3

Module 4

Parametrize Configuration (Locals and Output Values)

Overview

- **Local values (Locals** in short**)**: use a value multiple times within a module
- **Output values**: return values (resource names, public/private IPs, FQDN, etc.)

Local Values

- A value that can be consumed multiple times within a module
- Defined in a **locals** block
- Maybe stored in variables file
- Used in situations where:
  - A single set of values is used in **several** places
  - One or more values in the set is **likely** to be changed in the future
- **Locals vs Input Variables**:
  - Locals take both **static** and **dynamic** values (var.xxxxxx)
  - Input variables take **static** values **only**
- A common example is to **group several tag values**

Local Values Example Use

```
locals {
  common_tags = {
    Name           = "Automation CCGC 5502"
    Project        = "Terraform"
    ContactEmail   = "first.last@humber.ca"
    ExpirationDate = "2025-12-31"
    Environment    = "Test"
  }
}
```

```
resource "azurerm_network_interface" "linux_nic" {
  name  = "${var.linux_name}-nic"

  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  tags                = local.common_tags
```

Lab03S1: Define and Consume Local Values

- See Lab03 Section 1

Output Values (Single)

- **Exposes** attributes (resource names, IPs, IDs, etc.) at the completion of deployment
- Recommended to be defined in a separate file
- An output block must have a name and a value to be exposed, and optionally a description, dependency, and whether you want it to be revealed (sensitive)
- Can be viewed using **terraform output** after **terraform apply** has been executed successfully

```
output "VirtualNetworkInfo" {
  value = azurerm_virtual_network.vnet.address_space
}

output "SubnetInfo" {
  value = azurerm_subnet.subnet.address_prefixes
}
```

Lab03S2: Define and Display Output Values

- See Lab03 Section 2

Module 5

Set Lifecycle Rules and Dependencies

Lifecycle Rules

- Special argument for controlled Terraform deployments
- Added as a nested block within a resource block
- The **lifecycle** meta-argument:
  - **create_before_destroy**: true or **false** (default)
  - **ignore_changes**: (applies to update operations only; accepts a list of attributes)
  - **prevent_destroy**: true or **false** (default)

```
lifecycle {
  prevent_destroy = true
  ignore_changes = [tags, ]
}
```

Dependencies

- Special argument for better control of Terraform resources
- Added as a nested block within a resource block
- The **depends_on** meta-argument:
  - By default, TF builds a graph of all resources, and parallelizes the creation and modification of any non-dependent resources
  - Explicit dependency is needed to handle hidden resource dependencies that Terraform cannot automatically deduce
  - Accepts a **list** of resource references

```
resource "azurerm_network_security_group" "nsg" {
  name                = var.nsg
  location            = var.location
  resource_group_name = azurerm_resource_group.rg.name
  depends_on          = [azurerm_resource_group.rg]
```

Lab03S3: Implement and Test Lifecycle Rules and Dependencies

- See Lab03 Section 3

Module 6

Manage Multiple Infrastructure Resources

Overview

- Special arguments to create multiple resources
- The **count** and **for_each** meta-arguments
- Each instance created is:
  - A distinct infrastructure resource
  - Created, updated, and destroyed separately
- **count** and **for_each** cannot co-exist in the same resource block

The count Meta-Argument

- If a resource block includes a **count** meta-argument whose value is a **whole number**, Terraform creates that many instances.
- Good to create **almost identical** resources

```
variable "linux_name" {
  default = "lab-db1-u-vm"
}

variable "nb_count" {
  default = "2"
}
```

Terraform **format** function

%1d = lab-db1-u-vm1
%2d = lab-db1-u-vm01
%3d = lab-db1-u-vm001
%4d = lab-db1-u-vm0001

```
resource "azurerm_linux_virtual_machine" "vmlinux" {
  count = var.nb_count
  name  = "${var.linux_name}${format("%1d", count.index + 1)}"
```

```
name  = "${var.linux_name}-nic-${format("%1d", count.index + 1)}"
```

Iteration over values with count

- Use the **element** function with count to refer to individual items within an array Terraform **element** Function
- Individual NIC to be assigned to each VM:

Terraform **splat** expression

```
network_interface_ids = [element(azurerm_network_interface.linux_nic[*].id, count.index + 1)]
```

Terraform **splat** Expression

- Individual Public IP to be created for each NIC:

```
public_ip_address_id = element(azurerm_public_ip.linux_pip[*].id, count.index + 1)
```

- For outputs:

```
output "Linux_hostnames" {
  value = [azurerm_linux_virtual_machine.vmlinux[*].name]
}
```

Azure Availability Set

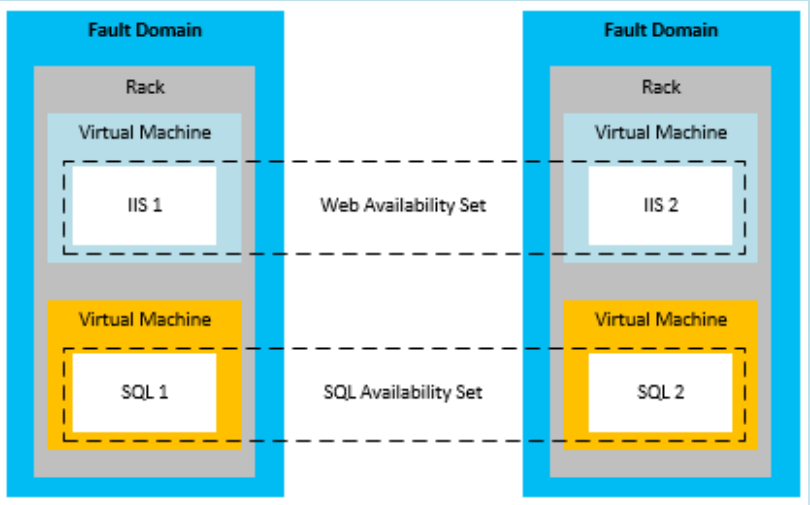
- Configure multiple VMs in an availability set for high-availability
- Each web/app/db tier should go in their own availability sets

Terraform **Availability Set** Resource

Fault Domain: A group of VMs that share **common physical infrastructure** (rack, servers, power source, network switch) and may **fail together** due to a common root cause (single point of failure).

Fault Domains

Update Domain: A group of VMs that can be upgraded or rebooted **together** during a planned maintenance (patching, upgrades, firmware) by Microsoft. During a planned maintenance, only one update domain is rebooted at a time.

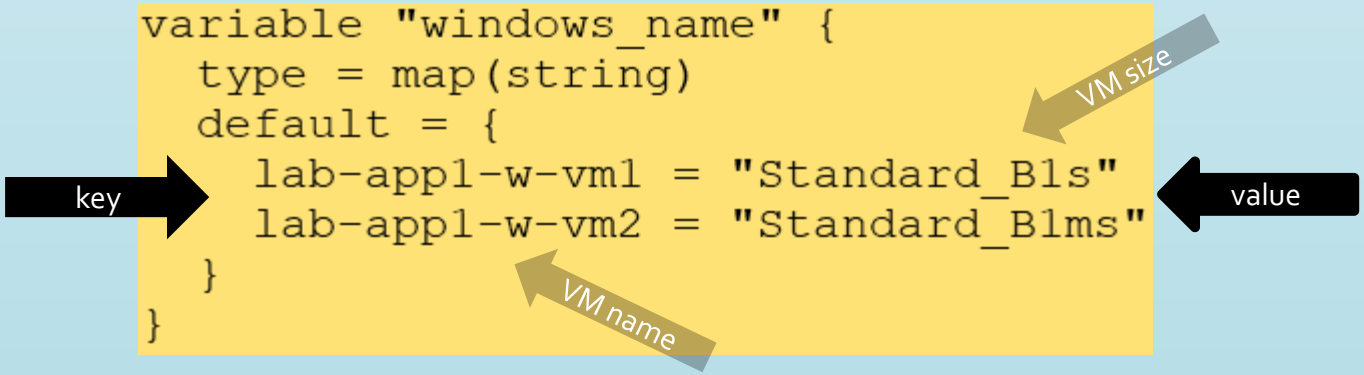


Lab04S1: Create Multiple VMs with Almost Identical Values

- See Lab04 Section 1

The for_each Meta-Argument

- If a resource block includes a **for_each** argument whose value is a **map** or a **set** of strings, Terraform creates one instance for each member of that map or set.
- Good to create resources with **some distinction**




```
resource "azurerm_windows_virtual_machine" "vmwindows" {
  name          = each.key
  for_each      = var.windows_name
```

```
name          = "${each.key}-nic"
```

Iteration over values with for_each

- Individual NIC to be assigned to each VM:

```
network_interface_ids = [azurerm_network_interface.windows_nic[each.key].id]
```

- Individual Public IP to be created for each NIC:

```
public_ip_address_id          = azurerm_public_ip.windows_pip[each.key].id
```

- For outputs:

```
output "Linux_hostnames" {
  value = values(azurerm_linux_virtual_machine.vmlinux)[*].name
}
```

Lab04S2: Create Multiple VMs with Distinct Values

- See Lab04 Section 2

Module 7

Perform Guest OS Customization

Overview

- **What is a Provisioner**: allows Terraform to run a command or playbook after VM creation or before VM removal
- Terraform can execute one or more commands inside a VM:
  - After launching (**creation-time** provisioners)
  - Before removing (**destroy-time** provisioners)
- **Creation-time (boot strapping, userdata)**: patching, storage config, user creation, file upload/edits, host firewall settings, system hardening, etc.
- **Destroy-time**: copy/archive data, app/db clean up, etc.

Provisioner Location

- A provisioner can be:
  - Nested within a VM resource block
  - An independent block (null_resource)
  - Nested within a VM module block
- Multiple provisioner blocks can be defined

Common Provisioners

- **file**: copies files/directories to provisioned machines
- **local-exec**: invokes a command on the Terraform machine to be executed on the provisioned VM
- **remote-exec**: invokes a command directly on the provisioned VM

- Note: Third-party provisioners are available for Chef, Puppet, and SaltStack

Note: HashiCorp does not recommend the use of provisioners, as their actions are not reflected in terraform plan or state

Connection Block

- Specifies how to connect to the guest OS:
  - **Linux**: SSH, username, private key, and IP/hostname
  - **Windows**: WinRM, username, password, and IP/hostname

Connection Block Options

- A connection block is defined inside a **provisioner** block

Connection Block (Single VM)

```
connection {
  type        = "ssh"
  user        = var.linux_admin_user
  private_key = file(var.priv_key)
  host        = azurerm_linux_virtual_machine.vmlinux.public_ip_address
  #host        = azurerm_public_ip.linux_pip.fqdn
}
```

Port 22

Single Linux VM

```
connection {
  type        = "winrm"
  user        = var.windows_admin_user
  password    = var.windows_admin_password
  host        = azurerm_windows_virtual_machine.vmwindows.public_ip_address
  #host        = azurerm_public_ip.windows_pip.fqdn
}
```

Port 5985

Single Windows VM

Connection Block (Multiple VMs)

Multiple Linux VMs with **count**

```
connection {
  type        = "ssh"
  user        = var.linux_admin_user
  private_key = file(var.priv_key)
  host        = element(azurerm_public_ip.linux_pip[*].fqdn, count.index + 1)
  #host       = element(azurerm_linux_virtual_machine.vmlinux[*].public_ip_address, count.index + 1)
}
```

Multiple Windows VMs with **count**

```
connection {
  type        = "winrm"
  user        = var.windows_admin_user
  password    = var.windows_admin_password
  host        = element(azurerm_windows_virtual_machine.vmwindows[*].public_ip_address, count.index + 1)
  #host        = element(azurerm_public_ip.windows_pip[*].fqdn, count.index + 1)
}
```

Multiple Linux VMs with **for_each**

```
connection {
  type        = "ssh"
  user        = var.linux_admin_user
  private_key = file(var.priv_key)
  host        = azurerm_linux_virtual_machine.vmlinux[each.key].public_ip_address
  #host        = azurerm_public_ip.linux_pip[each.key].fqdn
}
```

Multiple Windows VMs with **for_each**

```
connection {
  type        = "winrm"
  user        = var.windows_admin_user
  password    = var.windows_admin_password
  host        = azurerm_windows_virtual_machine.vmwindows[each.key].public_ip_address
  #host        = azurerm_public_ip.windows_pip[each.key].fqdn
}
```

Provisioner Examples

```
provisioner "file" {
  source = "some_local_file"
  destination = "remote_directory"

      Connection block goes here

}
```

Terraform VM

Target Linux or Windows VM

```
provisioner "local-exec" {
  command = "sleep 2; cat /etc/os-release"
}
```

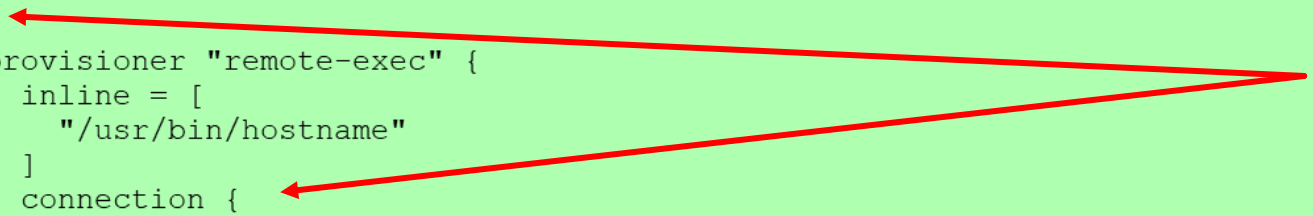
```
provisioner "remote-exec" {
  inline = [
    "/usr/bin/hostname"
  ]
```

One or more commands (Linux or Windows)

Special Resource Type: null-resource

- A special resource type to run provisioners that are **not directly** related to any other specific resource
- Connection block and provisioner can be defined inside a null_resource resource

```
resource "null_resource" "linux_provisioner" {
  count = var.nb_count
  depends_on = [
    azurerm_linux_virtual_machine.vmlinux
  ]

  provisioner "remote-exec" {
    inline = [
      "/usr/bin/hostname"
    ]
    connection {
      type        = "ssh"
      user        = var.linux_admin_user
      private_key = file(var.priv_key)
      host        = element(azurerm_public_ip.linux_pip[*].fqdn, count.index + 1)
      #host       = element(azurerm_linux_virtual_machine.vmlinux[*].public_ip_address, count.index + 1)
    }
  }
}
```



Provisioner Behaviour

- Default behaviour:
  - If a resource is created but creation-time provisioner fails, Terraform taints the resource for recreation
  - If a destroy-time provisioner fails, Terraform attempts to re-run it on the next destroy (when = destroy)
- Add **on_failure = continue** to override this behavior

Lab05s1: Employ Provisioners

- See Lab05 Section 1

Module 8

Switch to Remote Backend

+

Miscellaneous Stuff

Overview

- Backend keeps track of what was created (Terraform state)
- Referenced as the truthful source before changes or removals are applied
- Locked during Terraform **apply** to prevent inconsistencies
- Two types:
  - **Local** on the system (default):
    - Created on initialization (**terraform.tfstate**)
    - Information is stored in plain-text json format
  - **Remote** in Azure blob, AWS S3, or TF Cloud (recommended):
    - Safeguard sensitive information
    - Team collaboration
    - Allows remote operations (currently only via Terraform Cloud)

State Isolation

- Two ways:
  - Separate workspace for each environment (dev, test, prod, stage)
    - Two issues: (1) same backend with same authentication (2) invisible unless you run terraform workspace
  - Use different backends with different authentication for each environment
    - Store Terraform code in folder hierarchy. For instance:
      - dev/network/modules, dev/linux/modules

Lab05s2: Create Blob Container and Configure Remote Backend

- See Lab05 Section 2

```
terraform {
  backend "azurerm" {
    resource_group_name  = "tfstateRG"
    storage_account_name = "tfstate<nNumber>"
    container_name       = "tfstatefiles"
    key                  = "tfstate"
  }
}
```


"**key**" means **file** name

Miscellaneous Stuff

Built-In Functions

- List and explanation of several built-in functions are available at https://www.terraform.io/docs/configuration/functions.html
- User-defined functions are not supported

Data Sources

- Allow TF to use information defined outside of TF or by another separate TF configuration
- Queries for attributes (name, ID, location, etc.) of an **existing** resource for consumption by a resource being created
- The type and name for each data source block must be unique

data.<TYPE>.<NAME>.<ATTRIBUTE>

```
data azurerm_resource_group existing_rg {
  name = "LinuxRG"
}

resource "azurerm_linux_virtual_machine" "vmlinux" {
  name                = each.key
  for_each            = var.linux_name
  location            = data.azurerm_resource_group.existing_rg.location
  resource_group_name = data.azurerm_resource_group.existing_rg.name
  network_interface_ids = [azurerm_network_interface.linux_nic[each.key].id]
  availability_set_id = azurerm_availability_set.avset.id
  size                = each.value
```

Module 9

From Flat Architecture to Modular Architecture

Overview

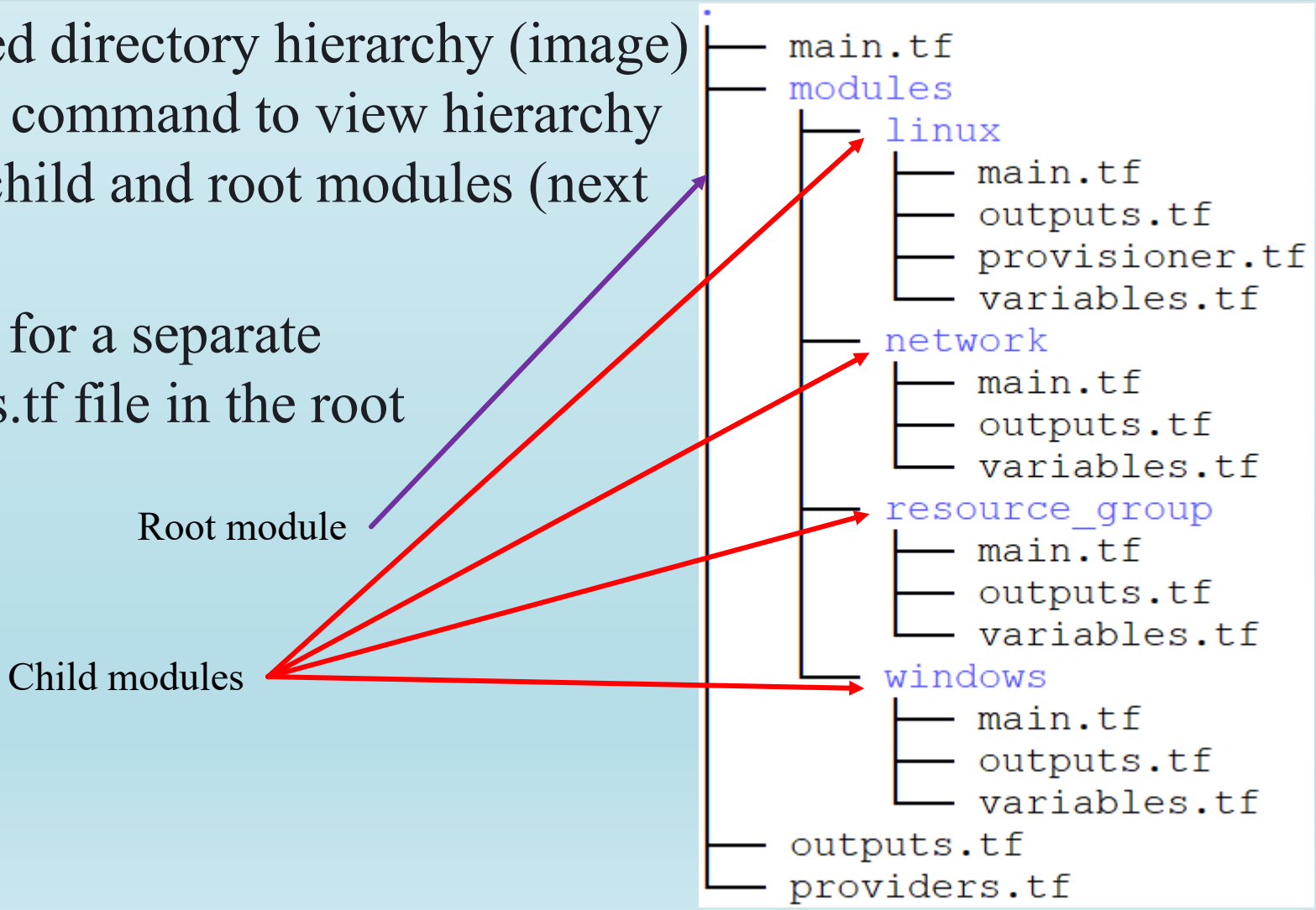
- A module is a **collection** of resources in a **single** directory
  - **root** and **child** modules
  - By default, Terraform only consumes files in the current directory
  - Root module can call child modules
  - Siblings cannot reference outputs from one another directly
  - **Benefits**: Better organization, code reusability, shareability with other teams, deployment consistency, and no duplication
  - Avoid 2+ levels of depth in module hierarchy for simplicity
  - Modules can be located:
    - Locally on the system
    - Remotely in git, Terraform Registry, HTTP locations, or TF Cloud (**terraform init** downloads the code for modules)

When NOT to Write Modules

- Do not write modules that are simple thin wrappers around single other resource types
- Avoid unnecessary complexity

Child and Root Module Relationship

- Suggested directory hierarchy (image)
- Use **tree** command to view hierarchy
- Update child and root modules (next slides)
- No need for a separate variables.tf file in the root module



Child Module Definitions

- Move **resource blocks** to **main.tf** (as is)
- Move **output blocks** to **outputs.tf** (as is)
- Move **provisioner blocks** to **provisioner.tf** (as is)
- Move **variable blocks** to **variables.tf**, and make the following changes:
  - **Declare variables** that you **DO** expect to change often (quantity, subnet/NSG, hostname, resource group, etc.)
  - **Define variables** that you **DO NOT** expect to change often (OS image, OS disk properties, username, public/private key, password, etc.)

Variable declaration: **without** a value

Variable definition: **with** a value

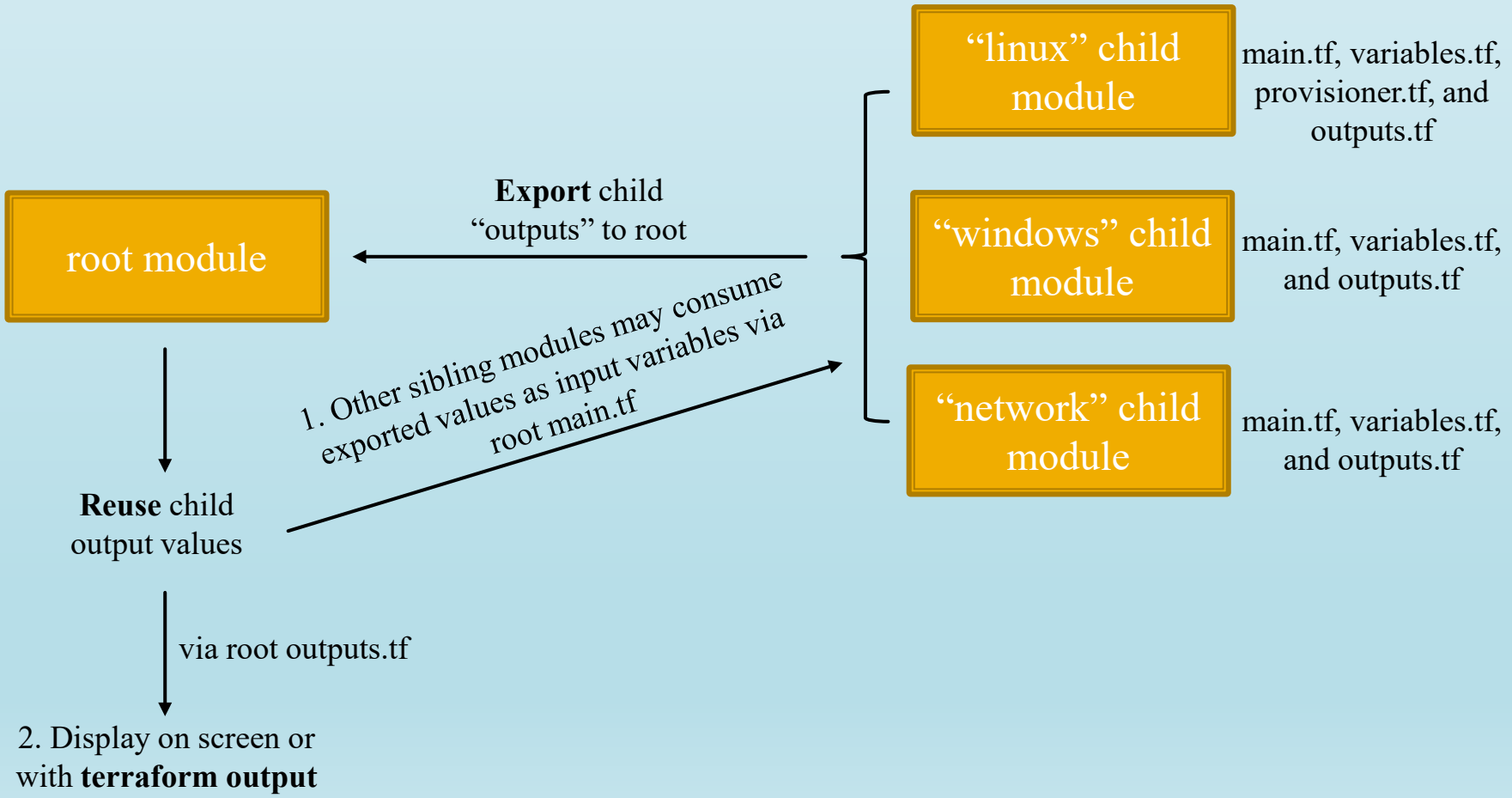
Root Module Definitions

- Update **main.tf**
- **Define** values that change **often**

Defined variables

```
module "network" {
  source              = "./modules/network"
  resource_group      = "NetworkRG"
  location            = "canadacentral"
  vnet                = "vnet-prd"
  vnet_space          = ["10.0.0.0/16"]
  subnet              = "subnet-prd"
  subnet_space        = ["10.0.0.0/24"]
  nsg                 = "nsg-prd"
}

module "linux" {
  source              = "./modules/linux"
  linux_name          = "terraform-u-vm"
  linux_avset         = "linux-avs"
  nb_count            = 2
  location            = "canadacentral"
  resource_group      = "LinuxRG"
  subnet_id           = module.network.subnet_ids
}

module "windows" {
  source              = "./modules/windows"
  windows_avset       = "windows-avs"
  windows_name        = {
    terraform-w-vm1   = "Standard_B1s"
    terraform-w-vm2   = "Standard_B1ms"
  }
  location            = "canadacentral"
  resource_group      = "WindowsRG"
  subnet_id           = module.network.subnet_ids

}
```

Output Values (child to root to child)



Sibling Communication

- Sibling modules exchange values via root module

```
output "rg_network" {
  value = azurerm_resource_group.rg_network
}
```

Output exposed to the root module by the resource_group module

```
module "network" {
  source          = "./modules/network"
  rg_network      = "Network-RG"
  resource_group  = module.resource_group.rg_network.name
  location        = module.resource_group.rg_network.location
  vnet            = "vnet1"
  vnet_space      = ["10.0.0.0/16"]
  subnet1         = "subnet1"
  subnet1_space   = ["10.0.1.0/24"]
  subnet2         = "subnet2"
  subnet2_space   = ["10.0.2.0/24"]
  nsg1            = "nsg1"
  nsg2            = "nsg2"
}
```

Exposed output value is passed to the network (sibling) module for consumption

Child to Root Output Values

- **outputs.tf** files in child and root modules

Definition in child module **(single value)**

```
output "rg_linux" {
  value = azurerm_resource_group.rg_linux
}
```

Definition in root module **(single value)**

```
output "rg_linux" {
  value = module.resource_group.rg_linux.name
}
```

Definition in child module **(multiple values)**

```
output "Linux_public_ip_addresses" {
  value = [azurerm_linux_virtual_machine.vmlinux[*].public_ip_address]
}
```

Definition in root module **(multiple values)**

```
output "Linux_public_ip_addresses" {
  value = module.linux.Linux_public_ip_addresses
}
```

Rules for Modular Configuration

Remember the rules for Terraform modular configuration

- For values that **DO** change often:
  - **Declare** variables in child variables.tf
  - **Define** them in root main.tf
  - **Consume** them in child main.tf
- For values that **DO NOT** change often:
  - **N/A**
  - **Define** variables in child variables.tf
  - **Consume** them in child main.tf

Important: Declared, defined, and consumed variable names **MUST** be identical

Tip: Start your journey with a single child module: update files, test config, troubleshoot errors, update files again, and retest the config until it is 'apply' successfully.

Lab06: Modularize Configuration

- See Lab06

More Azure Services
And
Assignment 1

Managed Disk

```
resource "azurerm_managed_disk" "data_disk" {
  for_each             = var.windows_name
  name                 = "${each.key}-data-disk"
  location             = var.location
  resource_group_name  = var.resource_group
  storage_account_type = var.data_disk_attr["data_disk_type"]
  create_option        = var.data_disk_attr["data_disk_create_option"]
  disk_size_gb         = var.data_disk_attr["data_disk_size"]
  depends_on           = [azurerm_windows_virtual_machine.vmwindows]
  tags = var.common_tags
}

resource "azurerm_virtual_machine_data_disk_attachment" "data_disk_attach" {
for_each           = var.windows_name
virtual_machine_id = azurerm_windows_virtual_machine.vmwindows[each.key].id
managed_disk_id    = azurerm_managed_disk.vmwindows[each.key].id
lun                = 0
caching            = var.data_disk_attr["data_disk_caching"]
depends_on         = [azurerm_managed_disk.data_disk]
}
```

Log Analytics Workspace

- **Log Analytics:**
  - A service that collects and analyzes performance and log data from cloud and on-prem resources (VMs, storage, OS, etc.)
  - Examples: (1) assess system updates and (2) troubleshoot OS incidents
- **Workspace:**
  - A storage location to collect, aggregate, and analyze data
  - Provides a geographic location, data isolation, and scope

```
resource "azurerm_log_analytics_workspace" "la_workspace" {
  name                = var.log_analytics_workspace["law_name"]
  location            = var.location
  resource_group_name = var.resource_group
  sku                 = var.log_analytics_workspace["log_sku"]
  retention_in_days   = var.log_analytics_workspace["retention"]

  depends_on = [
    azurerm_resource_group.resource_group,
  ]
  tags = local.common_tags
}
```

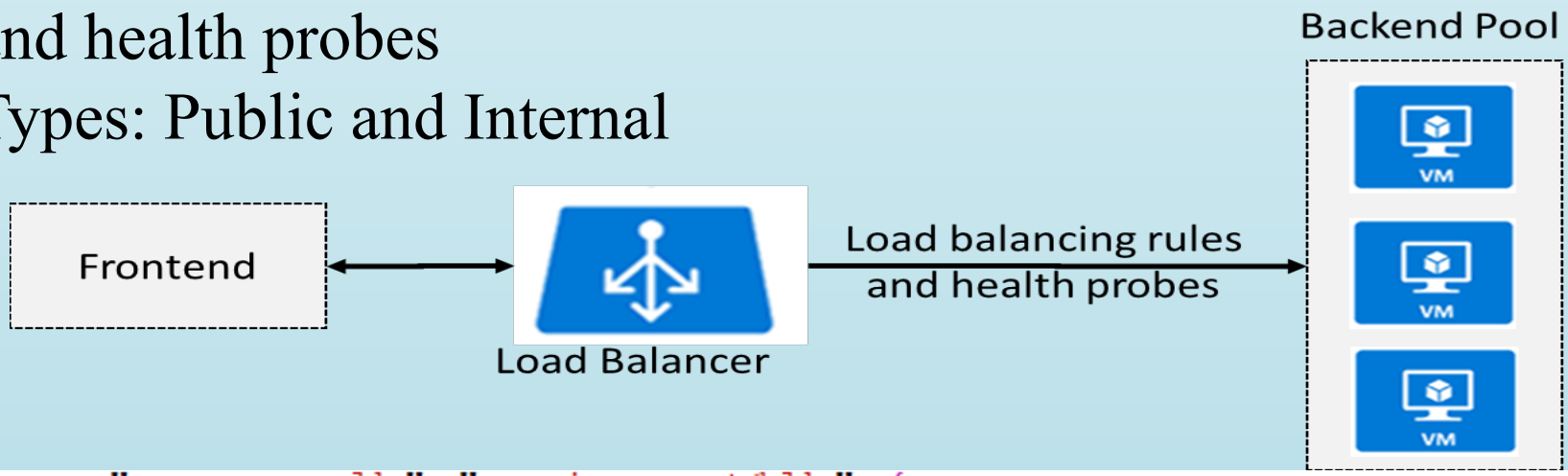
Recovery Services Vault

- Stores VM backups and replicated items
- Source and target regions should be different
- Used for disaster recovery purposes
- **Backup**: takes snapshots and stores them in RSV
- **Replication**: live copy from source to target

```
resource "azurerm_recovery_services_vault" "recovery_vault" {
  name                = var.recovery_services_vault["vault_name"]
  location            = var.location
  resource_group_name = var.resource_group
  sku                 = var.recovery_services_vault["vault_sku"]
  depends_on = [
    azurerm_resource_group.resource_group,
  ]
  tags = local.common_tags
}
```

Load Balancer

- **Distributes** inbound traffic to **backend** resources using rules and health probes
- Types: Public and Internal

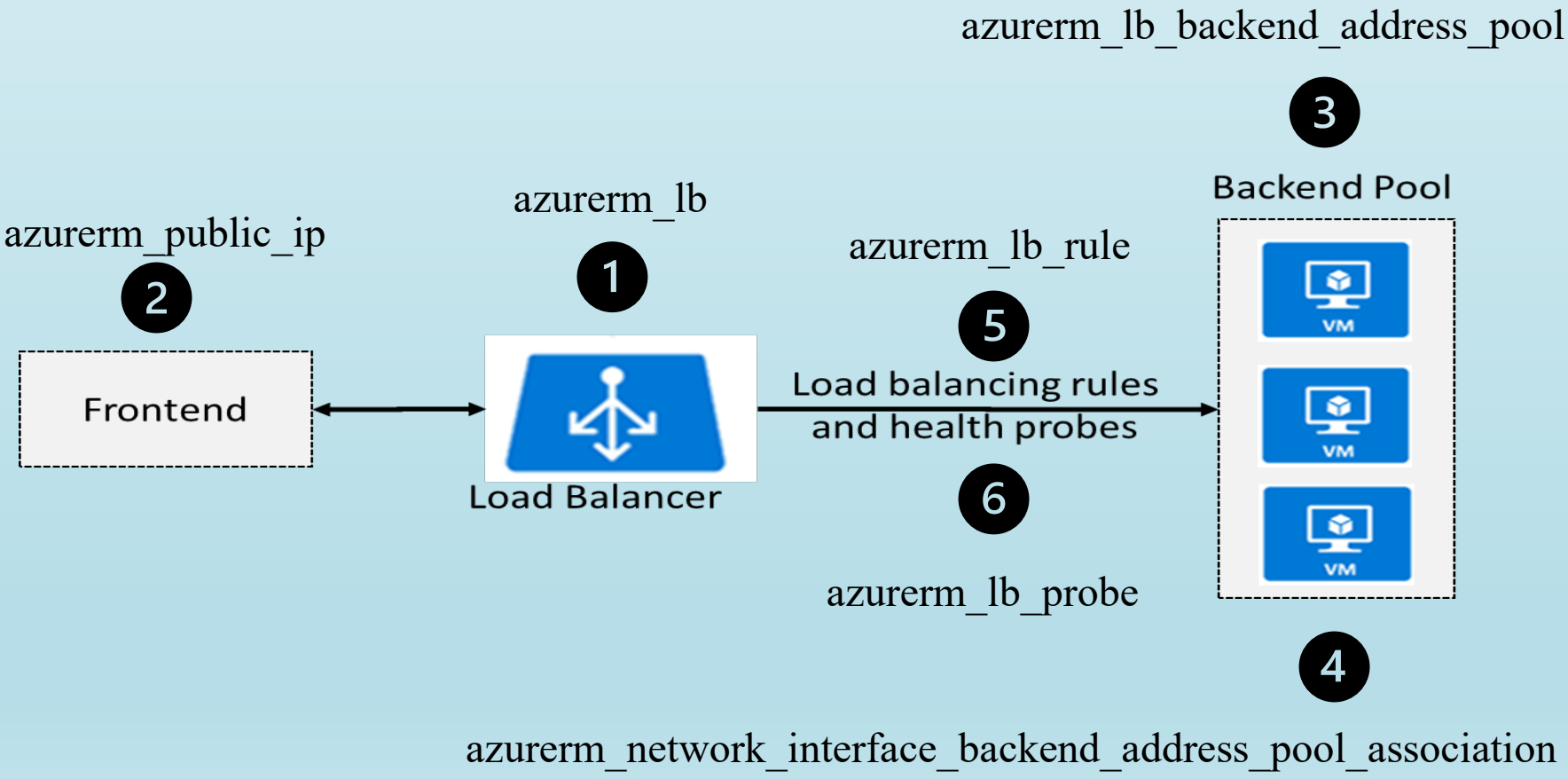



```
resource "azurerm_lb" "assignment11b" {
  name                = "assignment11b"
  location            = var.location
  resource_group_name = var.resource_group
  frontend_ip_configuration {
    name                 = "PublicIPAddress"
    public_ip_address_id = azurerm_public_ip.linux_pip[each.key].id
  }
  tags = var.common_tags
}
```

Load Balancer contd.

- You will need the following resources to build a load balancer:
  - (1) Load balancer (standard sku) using azurerm_lb
  - (2) Public IP (standard sku) as the frontend IP using azurerm_public_ip
  - Backend configuration:
    - (3) Address pool (azurerm_lb_backend_address_pool)
    - (4) Address pool association (azurerm_network_interface_backend_address_pool_association)
  - (5) Load balancing rule for the load balancer (azurerm_lb_rule)
  - (6) Health probes for the backend VMs (azurerm_lb_probe)

Load Balancer contd.



VM Extension

- Small **applications** installed on Linux and Windows VMs to provide configuration and automation tasks
- Examples: antimalware, log analytics, diagnostics, AAD login, custom script, network watcher, etc.
- To list available extensions: **az vm extension image list**

```
resource "azurerm_virtual_machine_extension" "LinuxDiagnostic" {
  for_each = var.linux_name
  name     = "LinuxDiagnostic"

  virtual_machine_id          = azurerm_linux_virtual_machine.vmlinux[each.key].id
  publisher                   = "Microsoft.Azure.Diagnostics"
  type                        = "LinuxDiagnostic"
  type_handler_version        = var.LinuxDiagVer
  auto_upgrade_minor_version = "true"
  depends_on = [
    azurerm_linux_virtual_machine.vmlinux,
    null_resource.whatever,
  ]
  tags = local.common_tags
}
```

Storage Account

- Azure storage account: blob, files, queue, and table
- **Account tier**: standard and premium
- **Account replication type (redundancy)**: LRS, ZRS, etc.
- We'll use **storage account type** StorageV2 (default) to store boot diagnostics data for VMs

```
resource "azurerm_storage_account" "storage_account" {
  name                     = var.storage_account
  resource_group_name      = azurerm_resource_group.resource_group.name
  location                 = azurerm_resource_group.resource_group.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
  }
```

VM Boot Diagnostic

- Endpoint for Azure storage account that can be used to store virtual machine boot diagnostics to enable access to **console** in Azure portal
- Defined within a virtual machine resource block
- Applies to **both** Linux and Windows virtual machines

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

Assignment 1

- See Assignment 1

Thank You