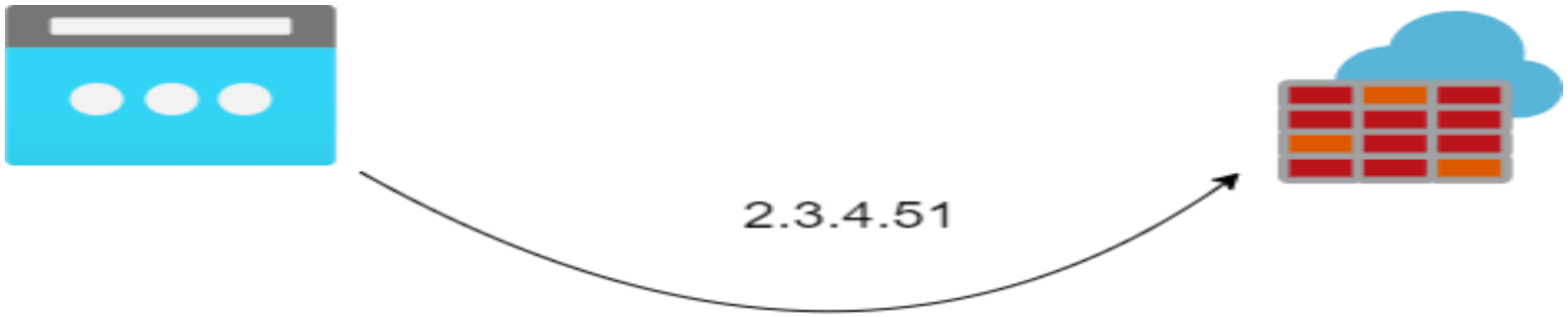
Output Values

Understanding Output Values

- Terraform has capability to output the attribute of a resource with the output values.
- **EXAMPLE:**
- We want Public IP as an output when we create public IP.
- An outputted attributes can't only be used for the reference but it can also act as a input to other resource being created via terraform.



Terraform Variables

Understanding the Challenge.

Repeated static values in the code can create more work in the future.

```
resource "azurerm_resource_group" "example" {
  name     = "output_demo"
  location = "West Europe"
}

resource "azurerm_public_ip" "example" {
  name                = "acceptanceTestPublicIp1"
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
  allocation_method   = "Static"

  tags = {
    environment = "Production"
  }
}
```

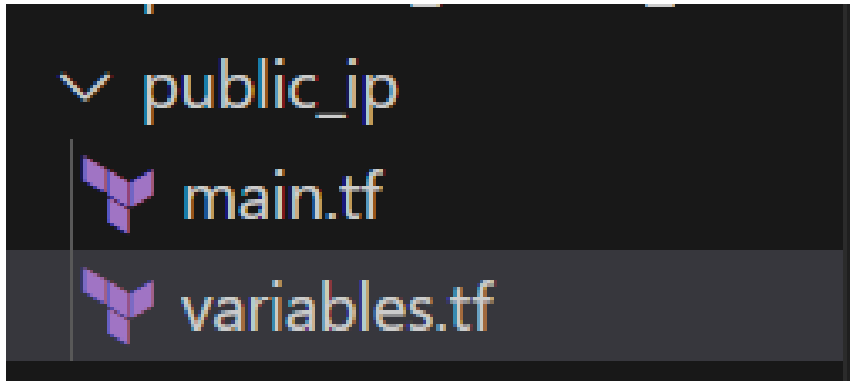
Better Approach

- A better solution would be to define repeated static value using variable.

```
resource "azurerm_resource_group" "example" {
  name     = var.rg_name
  location = var.location
}

resource "azurerm_public_ip" "example" {
  name                = var.ip_name
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
  allocation_method   = "Static"
}

variable "rg_name" {
    description = "the rg name"
    default =  "output_demo"
}
variable "location" {
    description = "the location where all the resources wil be created"
    default = "eastus"
}

variable "ip_name" {
    description = "public ip name"
    default= "vmip"
}
```

Benefits of Variables

- Update important values in one central place instead of searching and replacing them throughout your code, saving time and potential mistakes.



- No need to touch the core Terraform configuration file. This can avoid human mistakes while editing.

Join us in our Adventure

https://www.linkedin.com/in/akash-kumar-480b3858/

https://www.instagram.com/akash_sinha08/

Variable Assignment Approaches

Variable Assignment Approaches

Variables in Terraform can be assigned values in multiple ways:

- Variable Defaults
- Input from User
- Command-Line Flags
- From a File
- Environment Variables

```
resource "azurerm_resource_group" "example" {
  name     = var.rg_name
  location = var.location
}

resource "azurerm_public_ip" "example" {
  name                = var.ip_name
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
  allocation_method   = "Static"
}

variable "rg_name" {
    description = "the rg name"
    default =  "output_demo"
}
variable "location" {
    description = "the location where all the resources wil be created"
    default = "eastus"
}

variable "ip_name" {
    description = "public ip name"
    default= "vmip"
}
```

Join us in our Adventure

https://www.linkedin.com/in/akash-kumar-480b3858/

https://www.instagram.com/akash_sinha08/

Variable Definitions File (TFVars)

Understanding the Challenge.

- Managing variables in production environment is one of the very important aspect to keep code clean and reusable.

- HashiCorp recommends creating a separate file with name of terraform.tfvars to define all variable value in a project.

1.Main.tf Terraform Configuration File.

2. variables.tf file that defines all the variables.

3. providers.tf file that defines all the providers

4. terraform.tfvars file that defines value to all the variables

```
terraform {
  required_version = "~> 1.1

  required_providers {
    azurerm = { ···
    }
    random = { ···
    }
    helm = { ···
    }
  }
  backend "azurerm" { ···
  }

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

```
resource "azurerm_resource_group" "gp_rg" {
  name     = var.rg_name
  location = var.location
  tags     = local.common_tags
}


resource "azurerm_user_assigned_identity" "gp_user_mi" {
  location            = azurerm_resource_group.gp_rg.location
  name                = var.mi_name
  resource_group_name = azurerm_resource_group.gp_rg.name
  tags                = local.common_tags
}
```

prod > terraform.tfvars > ...

```
# Resource Group settings
rg_name         = "prod-aks-use-RG"
location        = "eastus"
mi_name         = "prod-aks-use-MI"
apim_subnet_name = "prod-aks-use-apim-SNET"
```

```
variable "rg_name" {
  description = "Name of the Azure res
  type        = string
}

variable "location" {
  description = "Azure region location
  type        = string
}

variable "mi_name" {
  description = "Name of the managed i
  type        = string
}
```

Selecting tfvars File

Organizations can have wide set of environments: Dev, Stage, Prod

If you have multiple variable definitions file (*.tfvars) file, you can manually define the file to use during command line.

```
infrastructure-as-code\aks\prod>terraform plan -var-file="prod.tfvars"
```

If file name is terraform.tfvars → Terraform will automatically load values from it. If file name is different like

prod.tfvars → You have to explicitly define the file during plan / apply operation.

Join us in our Adventure

https://www.linkedin.com/in/akash-kumar-480b3858/

https://www.instagram.com/akash_sinha08/

Variable Precedence

Declaring Variable Values

When variables are declared in your configuration, they can be set in a number of ways:

1. Variable Defaults

2. Variable Definition File (*.tfvars)

3. Environment Variables

4. Setting Variables in the Command Line.

Variable Definition Precedence

Terraform loads variables in the following order, with later sources taking precedence over earlier ones:

1. Environment variables

2. The terraform.tfvars file, if present.

3. The terraform.tfvars.json file, if present.

4. Any *.auto.tfvars or *.auto.tfvars.json files, processed in lexical order of their filenames.

5. Any -var and -var-file options on the command line

Join us in our Adventure

https://www.linkedin.com/in/akash-kumar-480b3858/

https://www.instagram.com/akash_sinha08/