Infrastructure as Code with
Terraform

The Well Done Infrastructure As Code Tool

Slides by Alberto Varela / @artberri

About me

Alberto Varela Sánchez

I am Full Stack Developer based in Bilbao. I am passionate about everything related to web development and I am currently working as a Software Developer at

plain concepts

@artberri | github.com/artberri | berriart.com

Agenda

- Introduction
- What is Terraform?
- Terraform Concepts
- Terraform VS …
- Continuous Integration/Delivery
- Workshop
- Q&A

1. Introduction

Like the principle that the same source code generates the same binary, an Infrastructure as Code (IaC) model generates the same environment every time it is applied.

Sam Guckenheimer (Product Owner, Azure Devops)

Infrastructure as Code

- More automation involves less human errors
- Supports collaboration between Ops and Dev
- Increases transparency
- Traceability
- Integrity
- Repeatability
- Your code is great documentation
- Agility

2. What is Terraform?

Terraform is used to create, manage, and update infrastructure resources such as virtual networks, VMs, security rules, containers, domains and more. Almost any infrastructure type can be represented as a resource in Terraform.

Terraform

What is It?

- A tool for… “Write, Plan, and Create Infrastructure as Code”
- Domain Specific Language (Hashicorp Configuration Language)
- Declarative
- Readable and writable
- Written in Go
- Free Software & Open Source
- Freemium (Premium options: GUI, VC hosting, support, …)
- Multiplatform

What is not?

- Not an abstraction layer for any cloud
- Not a Software provisioner

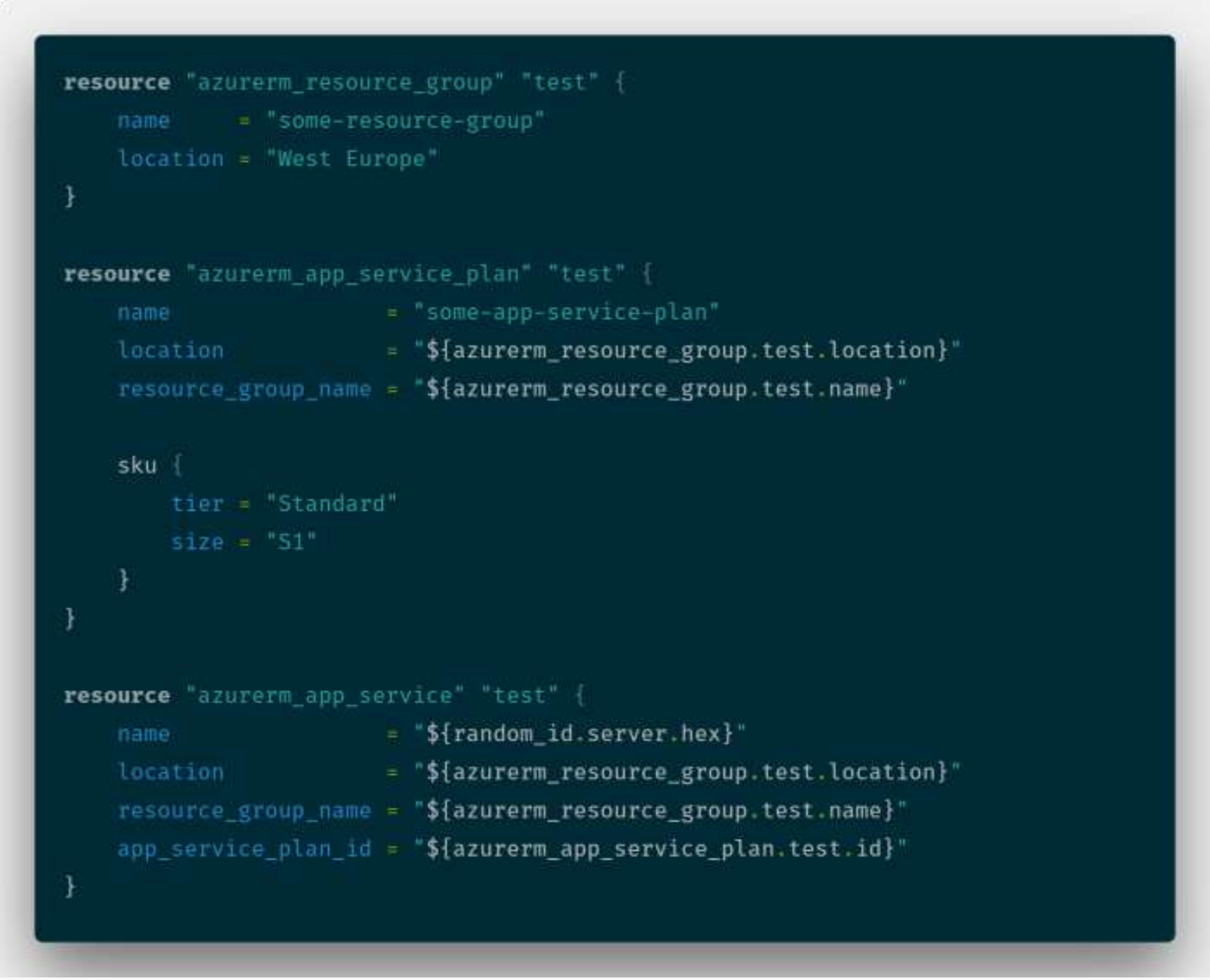
HashiCorp
Terraform

3. Terrafom Concepts

Basics

- Terraform syntax (or JSON)
- Terraform loads all the .tf files in a directory
- Mostly everything can be declared as a resource
- Changes will run parellelized

Resources



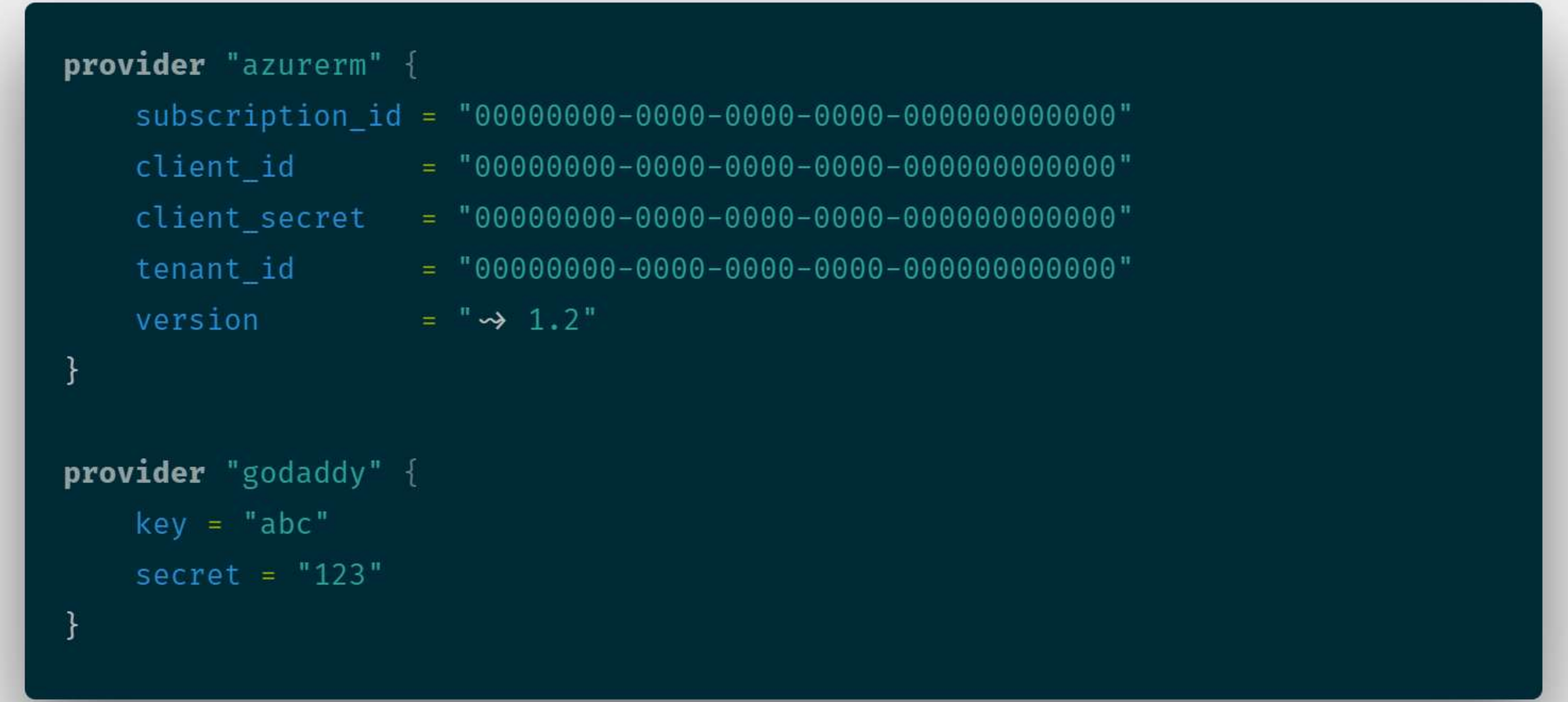
```
resource "azurerm_resource_group" "test" {
    name     = "some-resource-group"
    location = "West Europe"
}

resource "azurerm_app_service_plan" "test" {
    name                = "some-app-service-plan"
    location            = "${azurerm_resource_group.test.location}"
    resource_group_name = "${azurerm_resource_group.test.name}"

    sku {
        tier = "Standard"
        size = "S1"
    }
}

resource "azurerm_app_service" "test" {
    name                = "${random_id.server.hex}"
    location            = "${azurerm_resource_group.test.location}"
    resource_group_name = "${azurerm_resource_group.test.name}"
    app_service_plan_id = "${azurerm_app_service_plan.test.id}"
}
```

Providers

A provider is responsible for understanding API interactions and exposing resources.

Providers generally are an IaaS (e.g. AWS, GCP, Microsoft Azure, OpenStack), PaaS (e.g. Heroku), or SaaS services (e.g. Terraform Enterprise, DNSimple, CloudFlare).

Providers



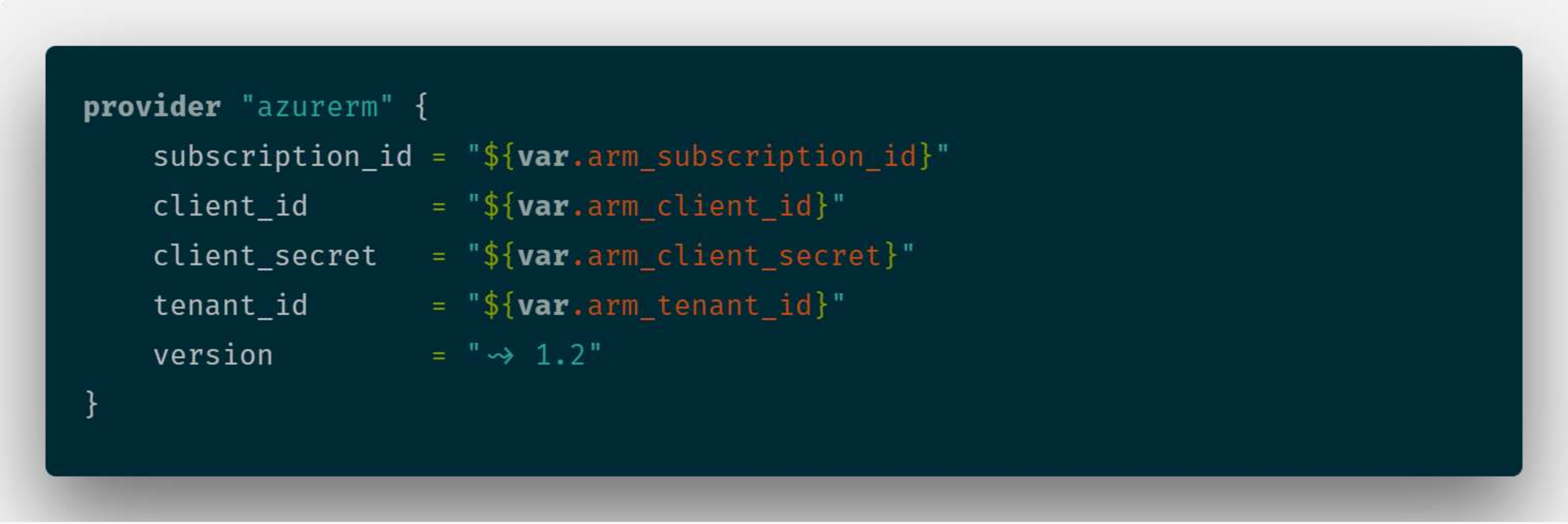
```
provider "azurerm" {
    subscription_id = "00000000-0000-0000-0000-000000000000"
    client_id       = "00000000-0000-0000-0000-000000000000"
    client_secret   = "00000000-0000-0000-0000-000000000000"
    tenant_id       = "00000000-0000-0000-0000-000000000000"
    version         = "~> 1.2"
}

provider "godaddy" {
    key = "abc"
    secret = "123"
}
```

Variables/Inputs

To become truly shareable and version controlled, it needs to parameterize the configurations.

We use variables for inputs that could change or to avoid hardcoding keys and secrets.

Variables/Inputs



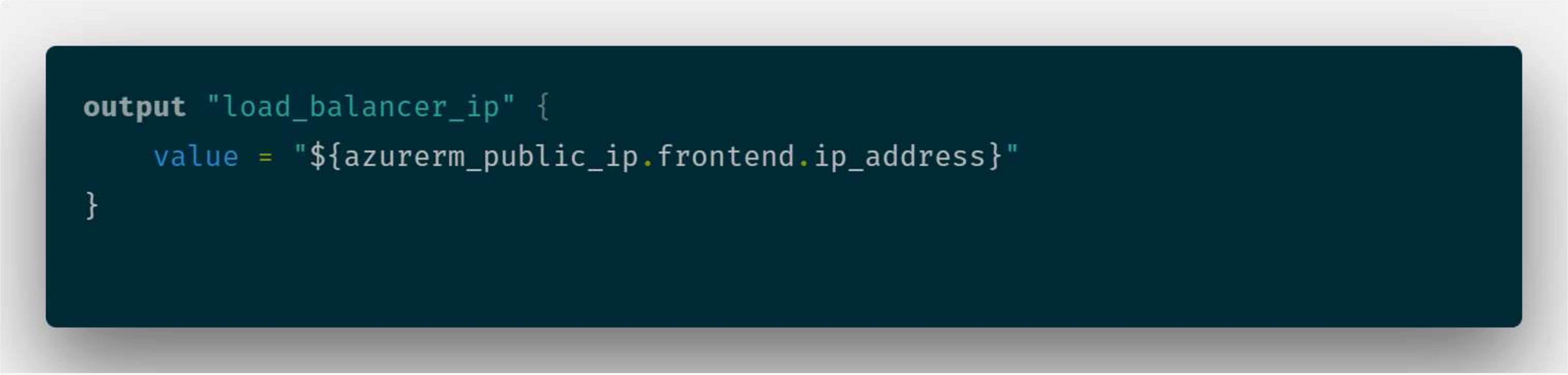
```
provider "azurerm" {
    subscription_id = "${var.arm_subscription_id}"
    client_id       = "${var.arm_client_id}"
    client_secret   = "${var.arm_client_secret}"
    tenant_id       = "${var.arm_tenant_id}"
    version         = "~> 1.2"
}
```

Outputs

A way to organize data to be easily queried and shown back to the Terraform user.

Terraform stores hundreds or thousands of attribute values for all your resources, but you may only be interested in a few values of importance, such as a load balancer IP, VPN address, etc.

Outputs

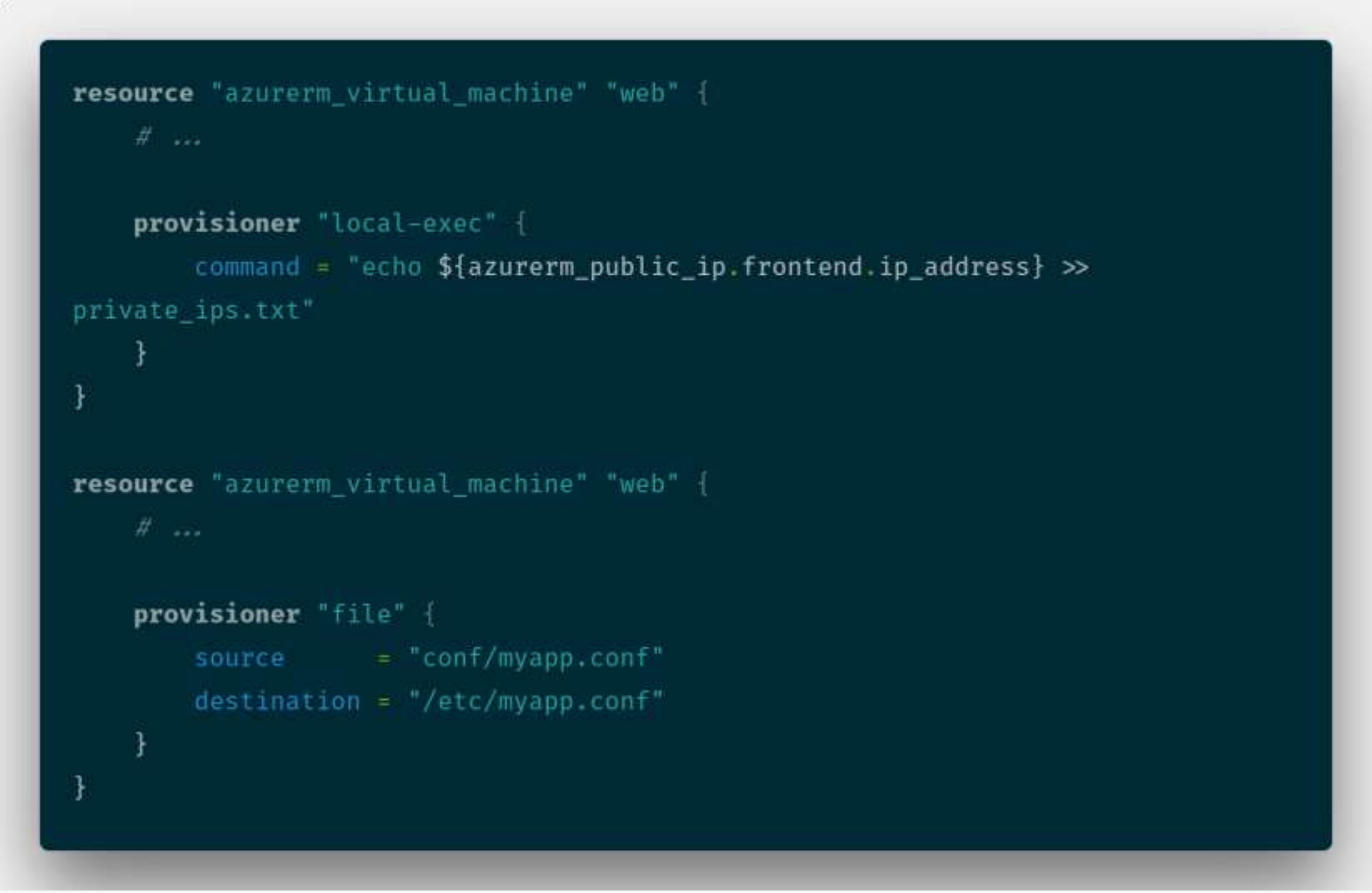


```
output "load_balancer_ip" {
    value = "${azurerm_public_ip.frontend.ip_address}"
}
```

Provisioners

Provisioners are used to execute scripts on a local or remote machine as part of resource creation or destruction. Provisioners can be used to bootstrap a resource, cleanup before destroy, run configuration management, etc.

Provisioners



```
resource "azurerm_virtual_machine" "web" {
    # ...

    provisioner "local-exec" {
        command = "echo ${azurerm_public_ip.frontend.ip_address} >>
private_ips.txt"
    }
}

resource "azurerm_virtual_machine" "web" {
    # ...

    provisioner "file" {
        source      = "conf/myapp.conf"
        destination = "/etc/myapp.conf"
    }
}
```

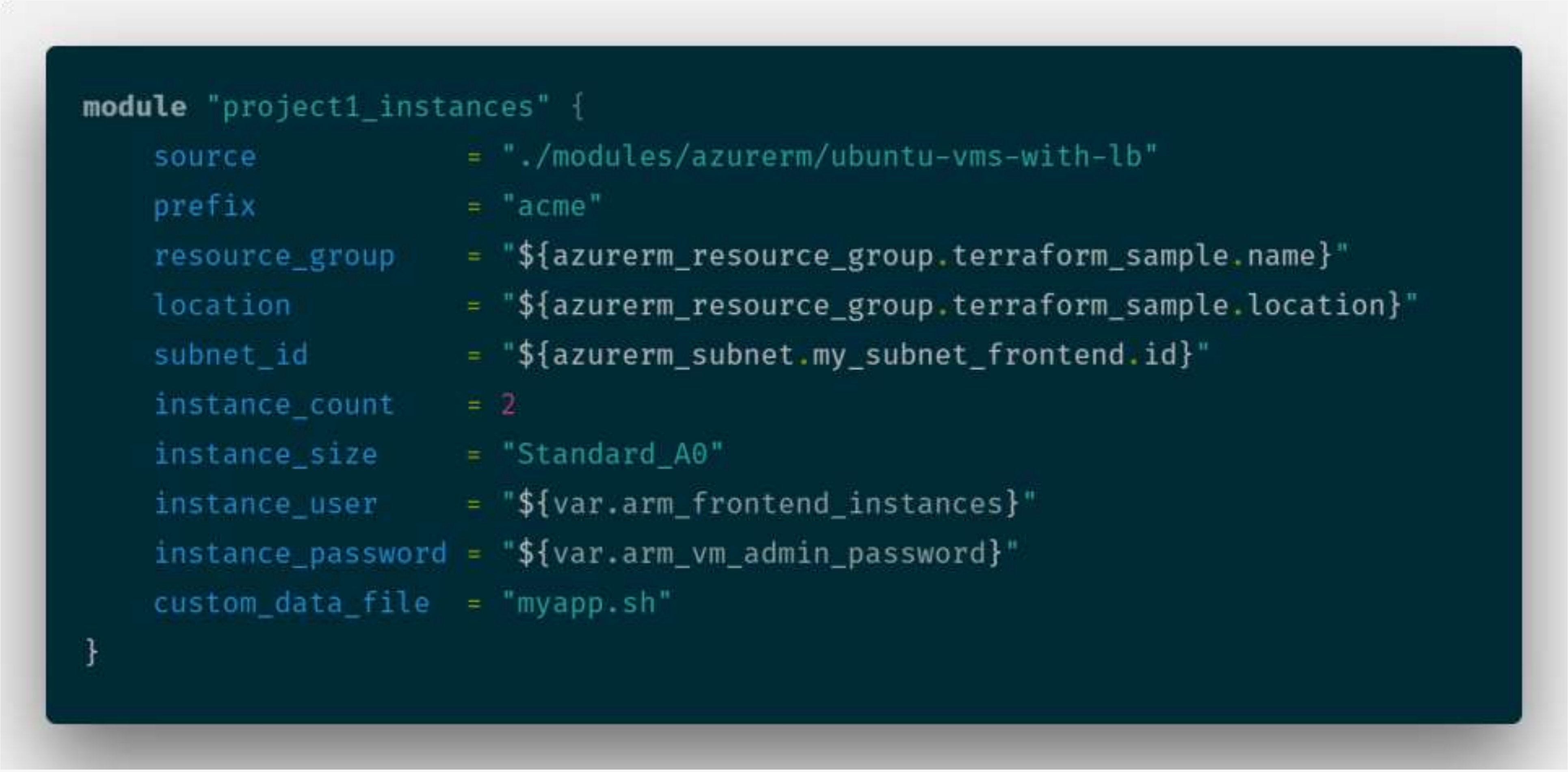
Modules

Modules in Terraform are self-contained packages of Terraform configurations that are managed as a group. Modules are used to create reusable components in Terraform as well as for basic code organization.

Create yours or use them from the Module Registry:

https://registry.terraform.io/

Modules

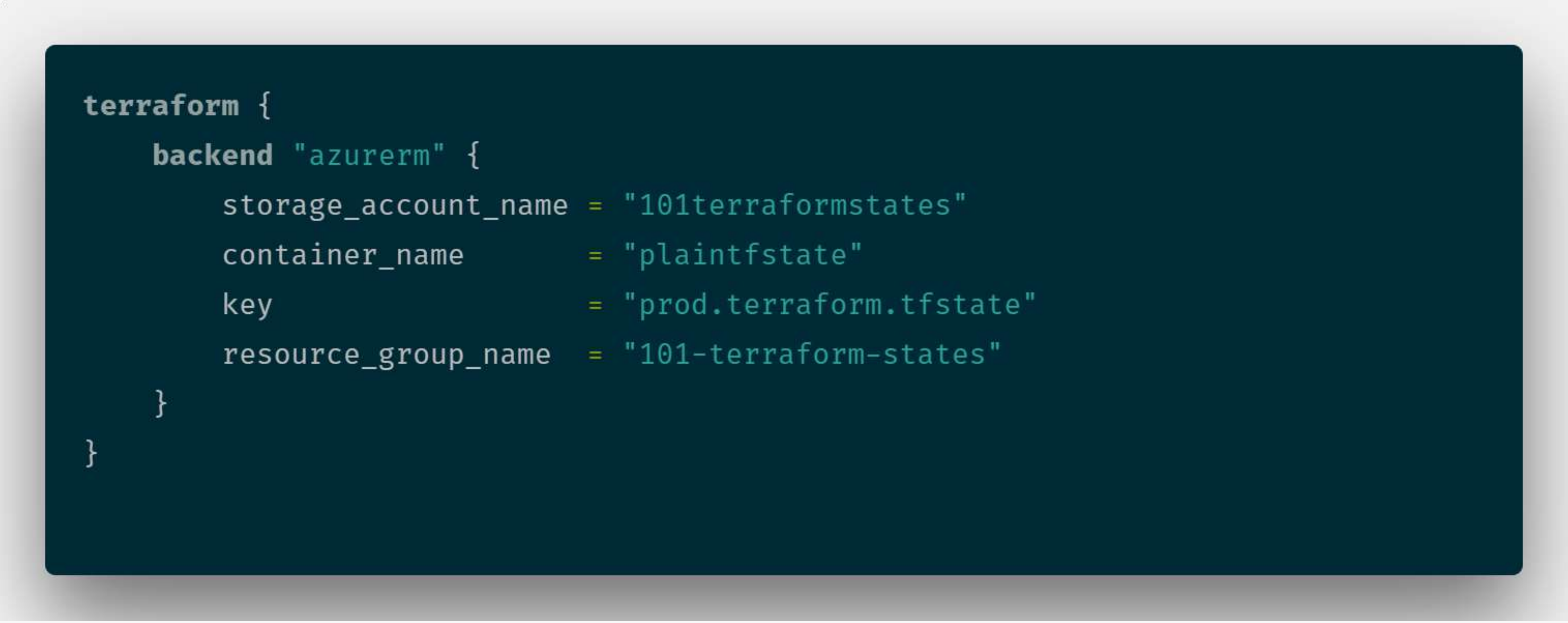


```
module "project1_instances" {
    source            = "./modules/azurerm/ubuntu-vms-with-lb"
    prefix            = "acme"
    resource_group    = "${azurerm_resource_group.terraform_sample.name}"
    location          = "${azurerm_resource_group.terraform_sample.location}"
    subnet_id         = "${azurerm_subnet.my_subnet_frontend.id}"
    instance_count    = 2
    instance_size     = "Standard_A0"
    instance_user     = "${var.arm_frontend_instances}"
    instance_password = "${var.arm_vm_admin_password}"
    custom_data_file  = "myapp.sh"
}
```

State/Backends

Terraform must store state about your managed infrastructure and configuration. This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures.

State/Backends



```
terraform {
    backend "azurerm" {
        storage_account_name = "101terraformstates"
        container_name       = "plaintfstate"
        key                  = "prod.terraform.tfstate"
        resource_group_name  = "101-terraform-states"
    }
}
```

Plugins

Terraform is built on a plugin-based architecture. All providers and provisioners that are used in Terraform configurations are plugins, even the core types such as AWS and Heroku. Users of Terraform are able to write new plugins in order to support new functionality in Terraform.

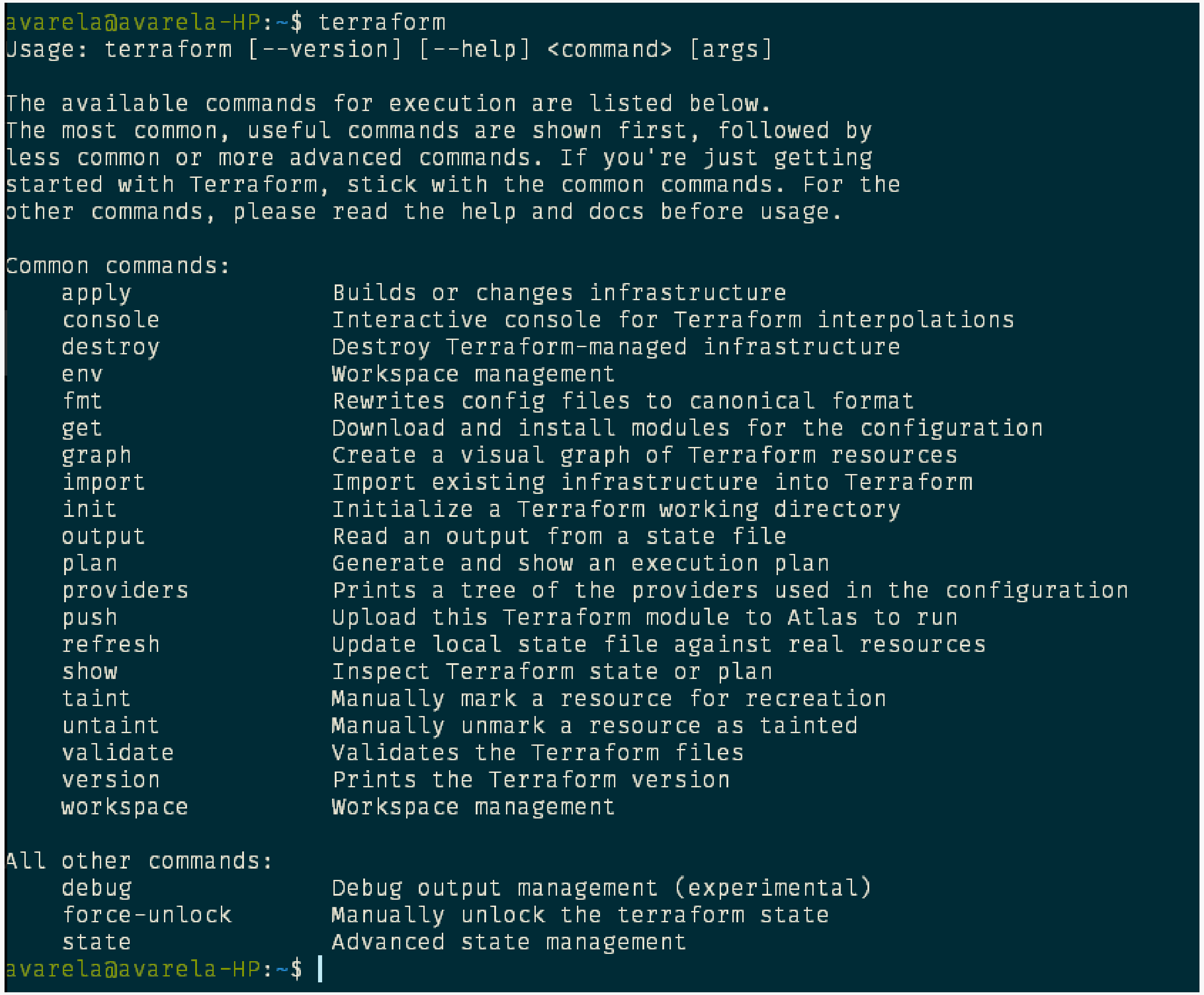
Plan

An execution plan describes which actions Terraform will take in order to change real infrastructure to match the written configuration.

CLI

More Important Commands

- init (≈download deps)
- plan (show execution plan)
- apply (execute changes)



```
avarela@avarela-HP:~$ terraform
Usage: terraform [--version] [--help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Workspace management
    fmt                Rewrites config files to canonical format
    get                Download and install modules for the configuration
    graph              Create a visual graph of Terraform resources
    import             Import existing infrastructure into Terraform
    init               Initialize a Terraform working directory
    output             Read an output from a state file
    plan               Generate and show an execution plan
    providers          Prints a tree of the providers used in the configuration
    push               Upload this Terraform module to Atlas to run
    refresh            Update local state file against real resources
    show               Inspect Terraform state or plan
    taint              Manually mark a resource for recreation
    untaint            Manually unmark a resource as tainted
    validate           Validates the Terraform files
    version            Prints the Terraform version
    workspace          Workspace management

All other commands:
    debug              Debug output management (experimental)
    force-unlock       Manually unlock the terraform state
    state              Advanced state management
avarela@avarela-HP:~$
```

4. Terraform VS ...

Terraform VS ARM/CLOUDFORMATION

TERRAFORM PROS

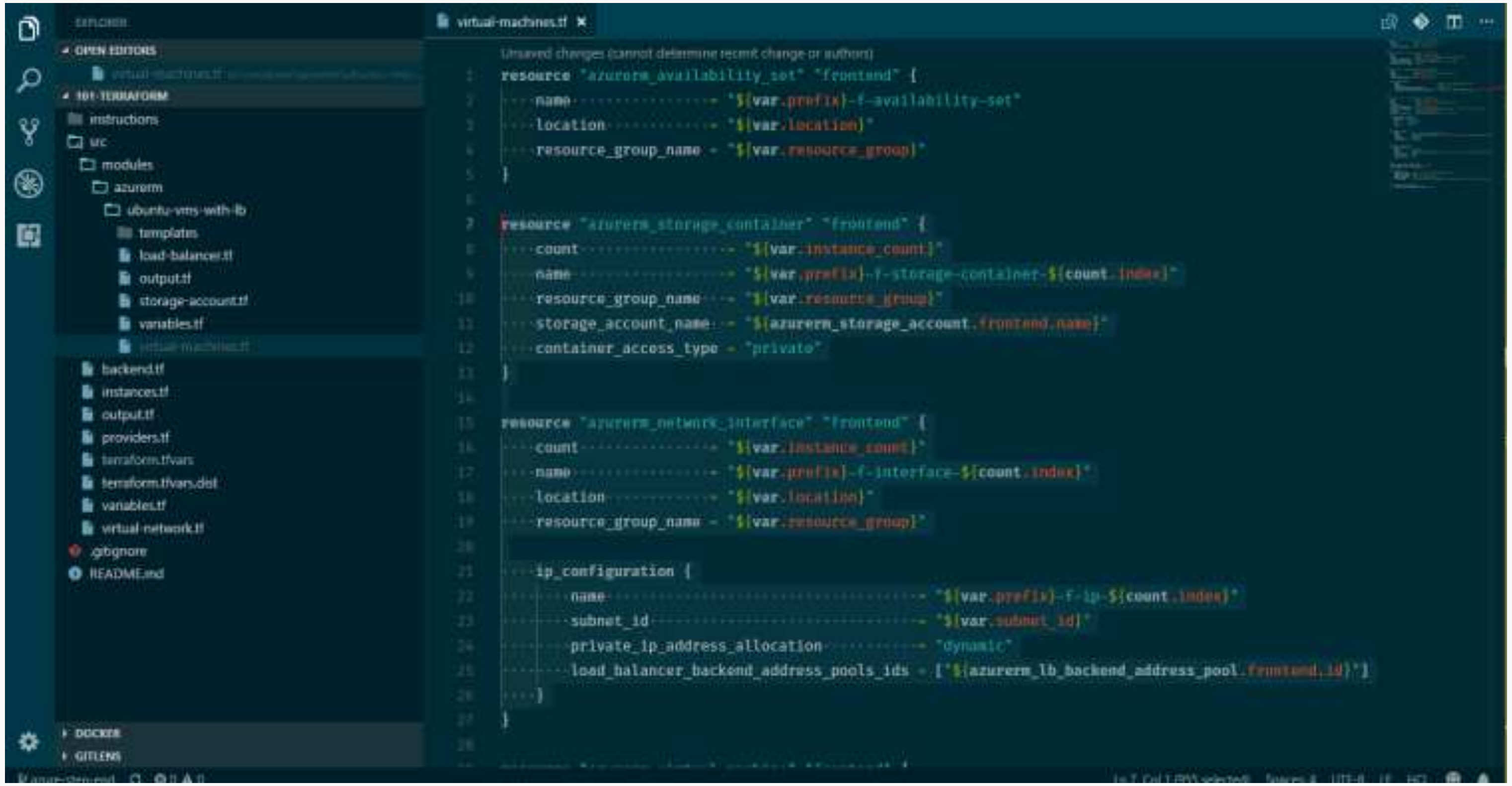
- More Readable & Writable
- More extensible
- Execution Plan
- Forget about dependsOn
- Multiple providers and pluggable

TERRAFORM CONS

- Not all resources are available but… You can use ARM templates inside Terraform code!

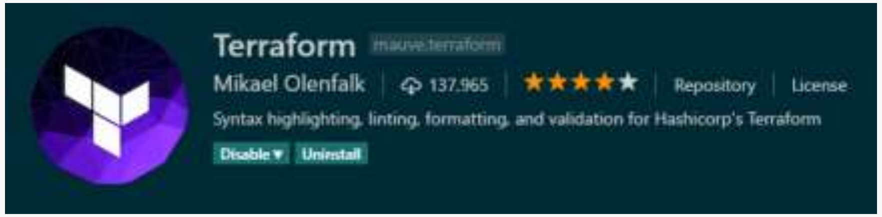
5. Continuous Integration/Delivery

Code



Unsaved changes (cannot determine recent change or authors)

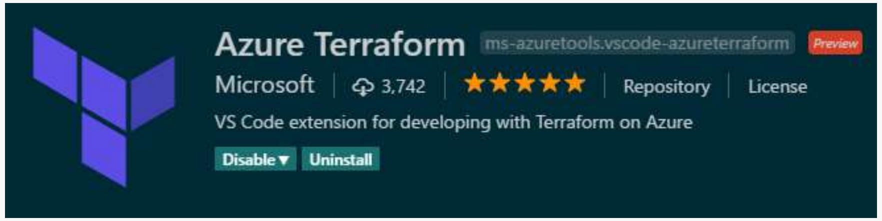
```
resource "azurerm_availability_set" "frontend" {
    name                = "${var.prefix}-f-availability-set"
    location            = "${var.location}"
    resource_group_name = "${var.resource_group}"
}

resource "azurerm_storage_container" "frontend" {
    count                 = "${var.instance_count}"
    name                  = "${var.prefix}-f-storage-container-${count.index}"
    resource_group_name   = "${var.resource_group}"
    storage_account_name  = "${azurerm_storage_account.frontend.name}"
    container_access_type = "private"
}

resource "azurerm_network_interface" "frontend" {
    count               = "${var.instance_count}"
    name                = "${var.prefix}-f-interface-${count.index}"
    location            = "${var.location}"
    resource_group_name = "${var.resource_group}"

    ip_configuration {
        name                                    = "${var.prefix}-f-ip-${count.index}"
        subnet_id                               = "${var.subnet_id}"
        private_ip_address_allocation           = "dynamic"
        load_balancer_backend_address_pools_ids = ["${azurerm_lb_backend_address_pool.frontend.id}"]
    }
}
```



Terraform mauve.terraform

Mikael Olenfalk | 137,965 | ★★★★★ | Repository | License

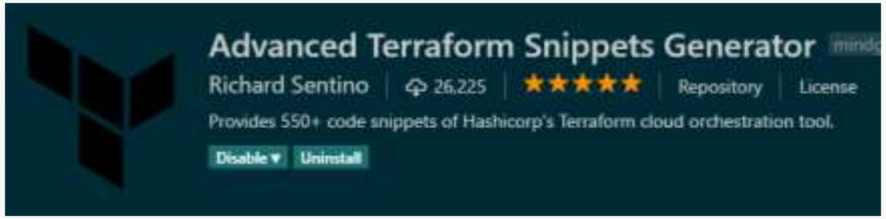
Syntax highlighting, linting, formatting, and validation for Hashicorp's Terraform



Azure Terraform ms-azuretools.vscode-azureterraform Preview

Microsoft | 3,742 | ★★★★★ | Repository | License

VS Code extension for developing with Terraform on Azure



Advanced Terraform Snippets Generator mindg

Richard Sentino | 26,225 | ★★★★★ | Repository | License

Provides 550+ code snippets of Hashicorp's Terraform cloud orchestration tool.

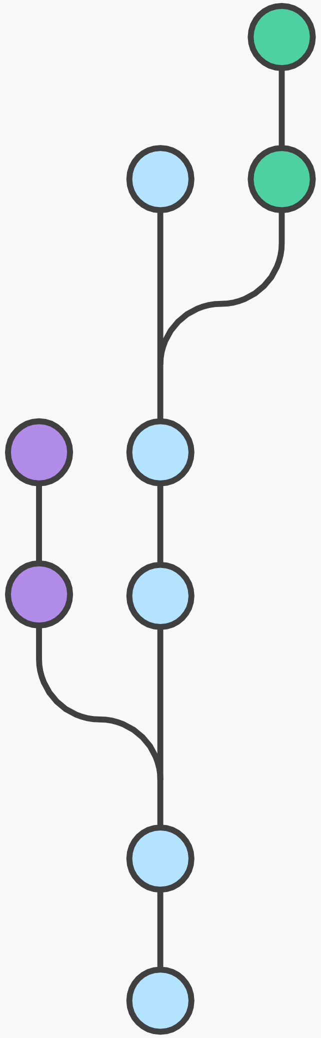
Structuring Repositories

Managing source code

- Only one repo for the whole organization resources
- One repo for main infrastructure + reusable modules
  and one extra repo per project

Managing environments

- Multiple Workspaces per Repo (Recommended)
- Branch per environment
- Directory per environment



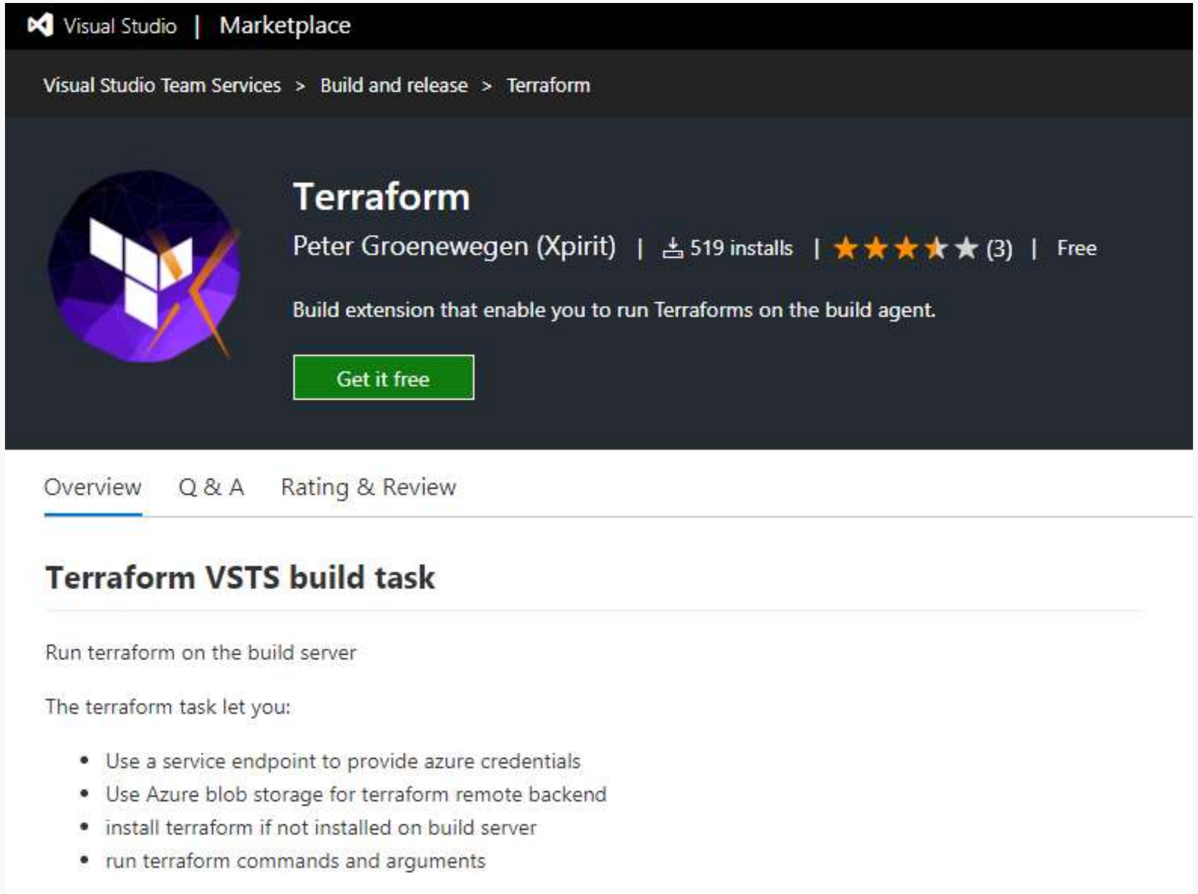
Automation

CI Tool

- Azure Devops/Jenkins/Gitlab or any other CI tool can be used

Flow

1. Merge Code
2. Init (≈download deps)
3. Plan (≈build)
4. Apply (≈release)



Visual Studio | Marketplace
Visual Studio Team Services > Build and release > Terraform

Terraform

Peter Groenewegen (Xpirit) | 519 installs | ★★★★★ (3) | Free

Build extension that enable you to run Terraforms on the build agent.

Get it free
Overview
Q & A
Rating & Review

Terraform VSTS build task

Run terraform on the build server

The terraform task let you:

- Use a service endpoint to provide azure credentials
- Use Azure blob storage for terraform remote backend
- install terraform if not installed on build server
- run terraform commands and arguments

Talk is cheap.
Show me the code!

Resources

- Terraform Azure Docs
  https://www.terraform.io/docs/providers/azurerm/
- Terraform Azure Workshop Repo
  https://github.com/artberri/101-terraform/
- Examples
  https://github.com/terraform-providers/terraform-provider-azurerm/tree/master/examples
- Module Registry:
  https://registry.terraform.io/

Q&A

Thank you!