Infrastructure as Code with Terraform

BECAUSE WE ARE DEVOPS

Who Am I?

Claudiu Șonel

DEVOPS CONSULTANT

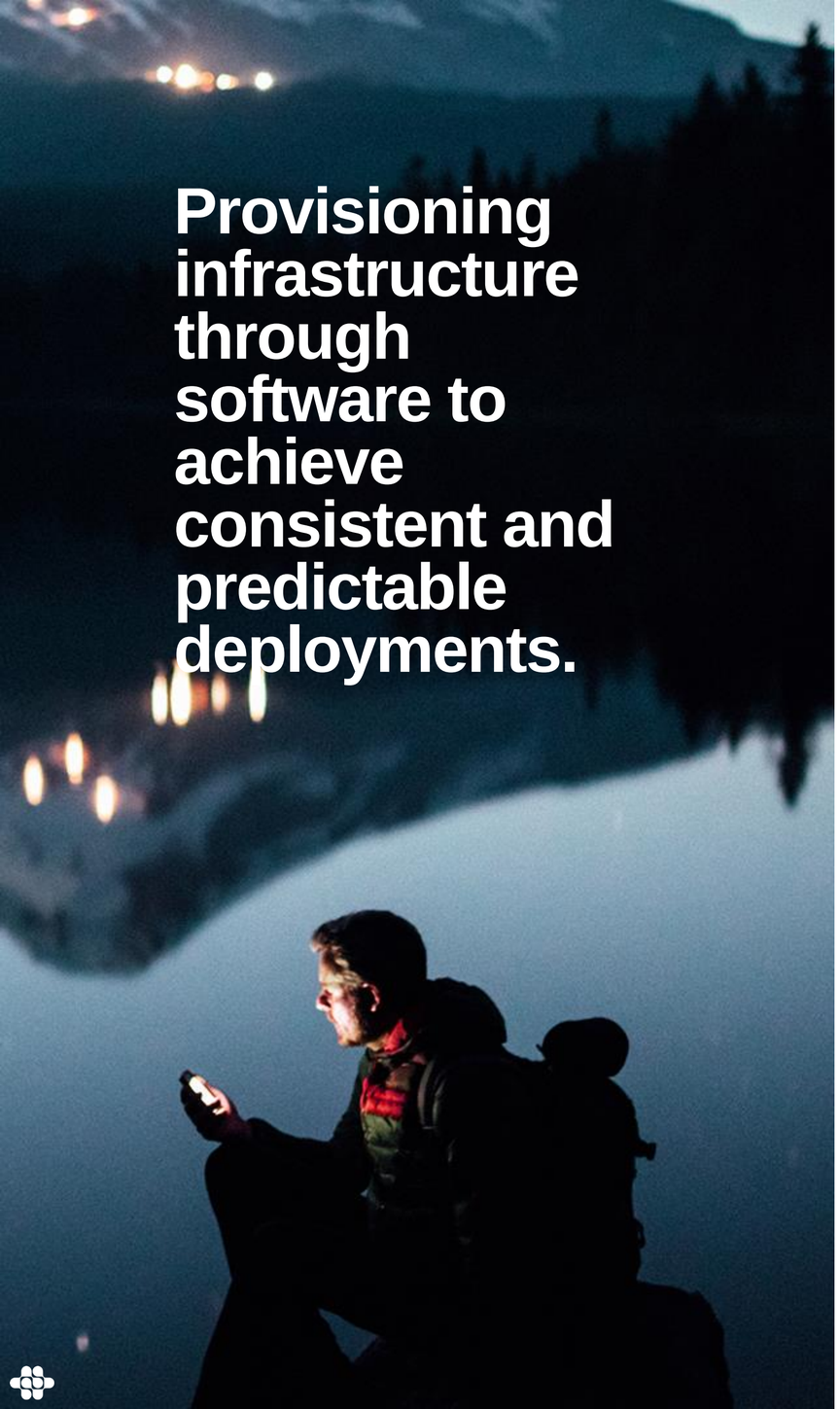
claudiu.sonel@endava.com

Agenda

1. INFRASTRUCTURE AS CODE AND ITS BENEFITS
2. TERRAFORM FUNDAMENTALS
3. INPUT VARIABLES AND OUTPUTS
4. TERRAFORM STATE
5. TERRAFORM MODULES
6. BUILT-IN FUNCTIONS AND DYNAMIC BLOCKS
7. TERRAFORM WORKSPACES
8. DEBUGGING TERRAFORM
9. Q&A
10. HOMEWORK

1

Infrastructure as Code



Infrastructure as Code

There are some core concepts in order to achieve that:

- Defined in code
- Stored in source control
- Declarative or imperative
- Idempotent and consistent
- Push or Pull

BENEFITS

- No more clicks
- Enables DevOps
- Automated deployment
- Consistent environments
- Reusable components
- Documented arhitecture
- Speed, cost and reduced risk

2

Terraform Fundamentals

Getting Started

Terraform is an *infrastructure automation tool*

The core of terraform is an *open-source* project maintained by HashiCorp. There are also paid version of terraform available, such as Terraform Cloud or Terraform Enterprise

Terraform is *vendor agnostic*

The core software of terraform is a *single binary compiled from Go*

Terraform uses a *declarative syntax*

The actual configuration files are written in HashiCorp Configuration Language (HCL) or JSON

What is Terraform?

Core Components

Executable

Configuration files

Provider plugins

State data

Installation

Steps:

1. Download the executable for your platform
2. Add to your PATH environment variable
3. Start using terraform!

You can find terraform also in package managers like apt, yum, homebrew or Chocolatey, and you can use it also as a docker container.

What is Terraform?

What is the Terraform Workflow?

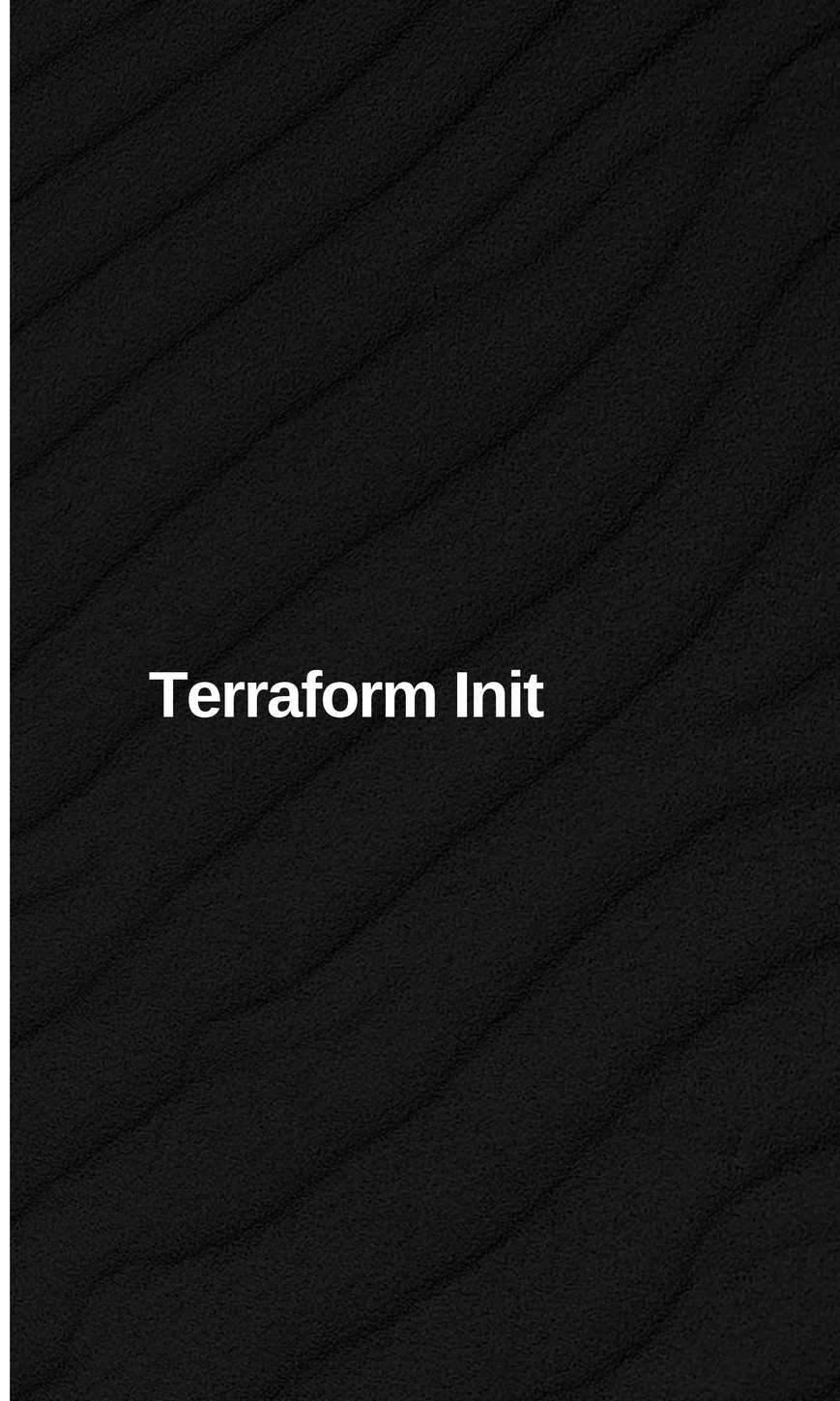
The Core Terraform Workflow

Initializing the Working Directory

```
terraform init
```

Initializes the working directory that contains your Terraform code

- Downloads ancillary components
- Sets up backend



Plan, Apply, and Destroy

Terraform Workflow: Write → Plan → Apply

1. Terraform Plan
   Reads the code and then creates and shows a plan of deployment

2. Terraform Apply
   Deploys the instructions and statements in the code, and updates the deployment state tracking mechanism file (state file)

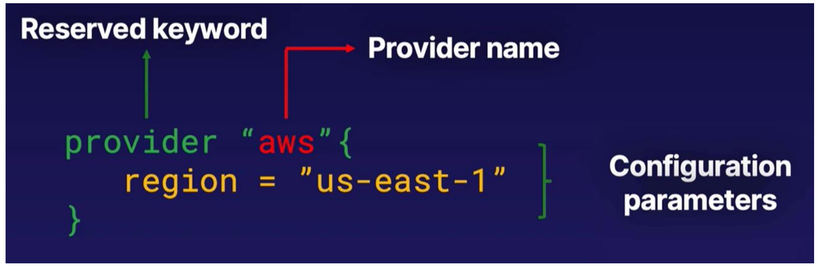
3. Terraform Destroy
   Looks at the recorded, stored state file created during deployment and destroys all the resources created by your code.
   CAUTION! It is a non-reversible command

Terraform key concepts

Terraform Object Types

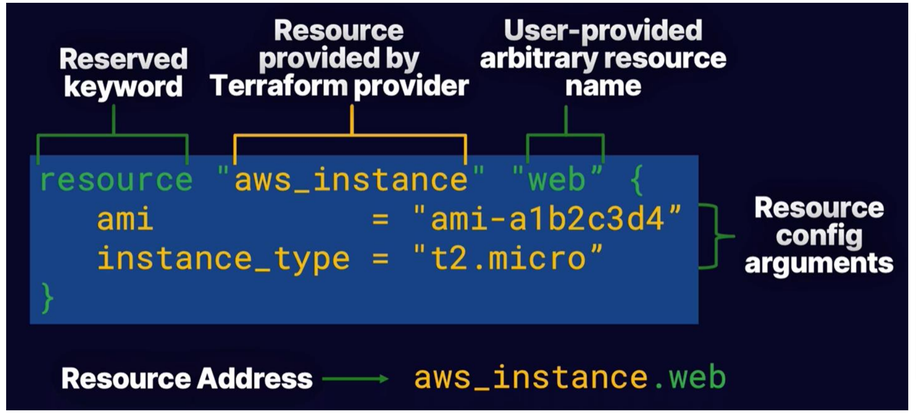
Providers

Provider block contains information about what provider you want to use.



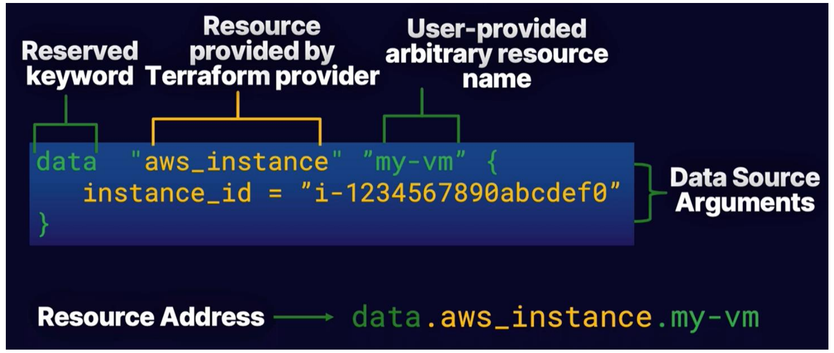
Resources

Resources are things you want to create in the target environment.



Data Sources

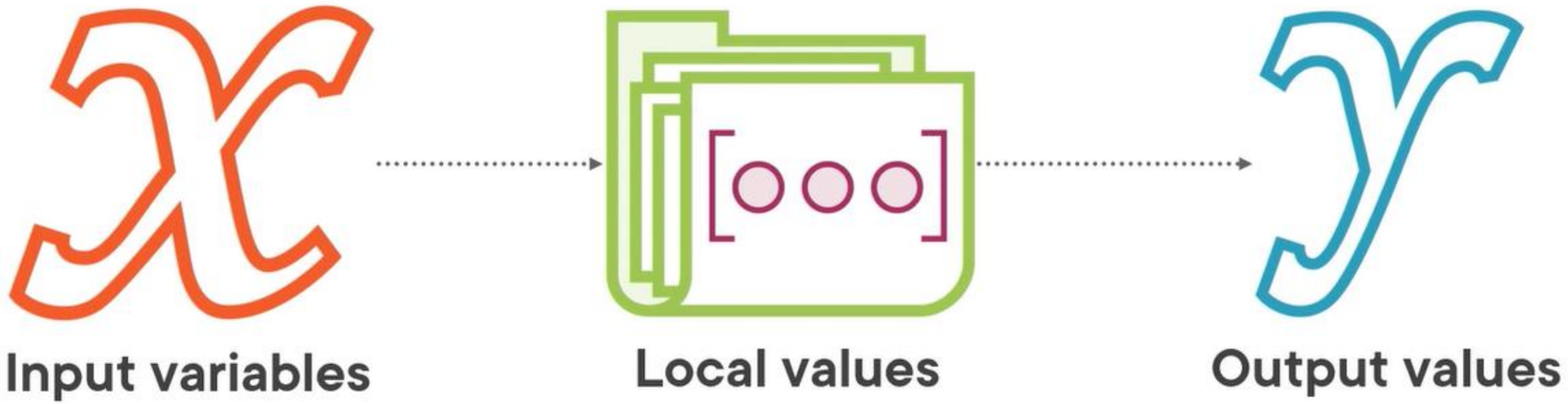
Data sources are a way to query information from a provider



3

Input Variables and Outputs

Variables and Outputs



Variable Syntax

Terraform Data Types

PRIMITIVE

String, number, boolean

COLLECTION

List, set, map

STRUCTURAL

Tuple, object

Collection Values Syntax

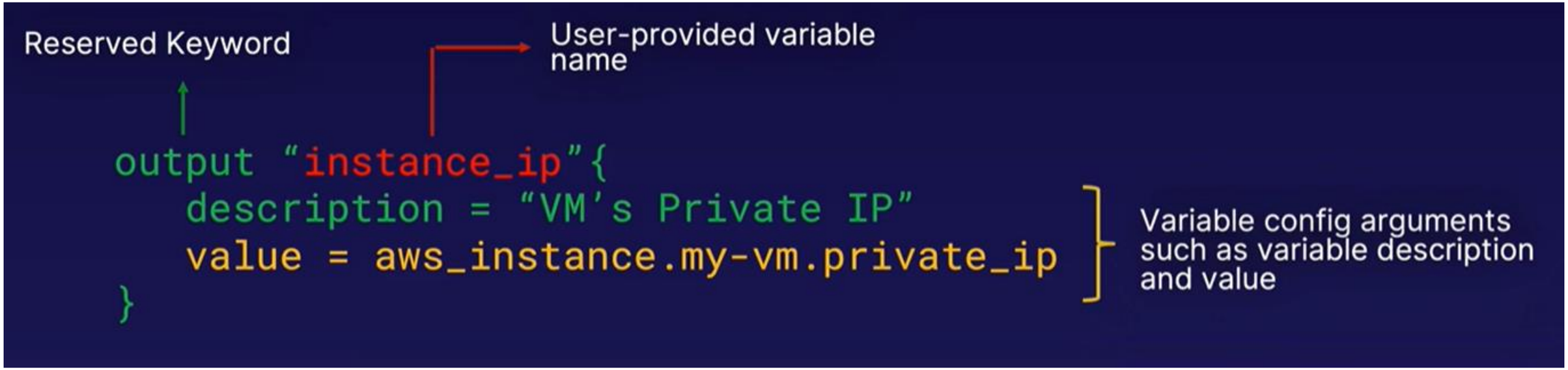
```
variable "aws_regions" {
  type = list(string)
  description = "Region to use for AWS resources"
  default = ["us-east-1", "us-east-2", "us-west-1", "us-west-2"]
}

Referencing:
        var.<name_label>[<element_number>]
        var.aws_regions[0]
```

```
variable "aws_instance_sizes" {
  type = map(string)
  description = "Instance sizes for AWS resources"
  default = {
    small  = "t2.micro"
    medium = "t2.small"
    large  = "t2.large"
  }
}

Referencing:
        var.<name_label>.<key_name>   or   var.<name_label>["key_name"]
        var.aws_instance_sizes.small  or   var.aws_instance_sizs["small"]
```

Local Values Syntax

```
locals {
  instance_prefix = "endava"
  common_tags = {
    company = "Endava"
    project = var.project
    billing_code = var.billing_code
  }
}

Referencing:
        local.<name_label>
        local.instance_prefix
        local.common_tags.company
```

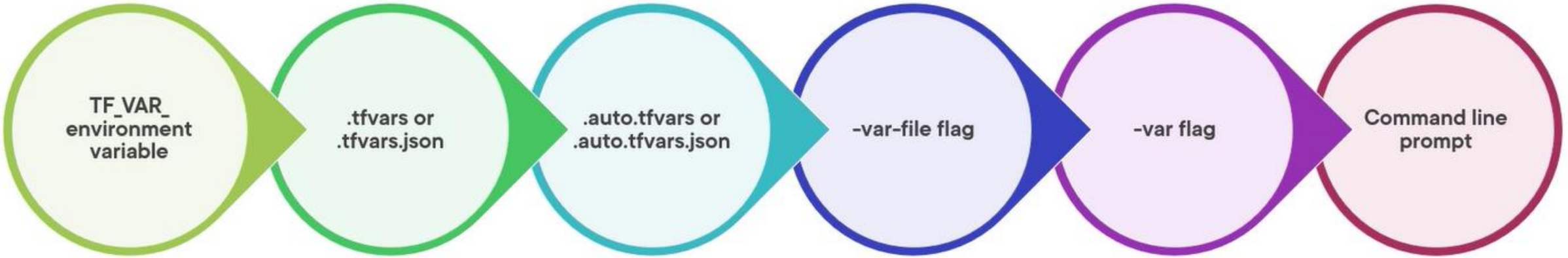
Outputs Syntax



VARIABLES

- Default value
- -var argument
- -var-file argument
- terraform.tfvars / terraform.tfvars.json
- .auto.tfvars / .auto.tfvars.json
- Environment variable TF_VAR_

Evaluation Precedence



4

Terraform State

STATE

- JSON format (Do not touch!)
- Resources mappings and metadata
- Location: local / remote (AWS S3, Google Cloud Storage, NFS, Terraform Cloud)
- Locking
- Workspaces

```
{
  "version": 4,
  "terraform_version": "0.12.26",
  "serial": 198,
  "lineage": "e4966818-25ec-a704-add7-ed6d536bb8e4",
  "outputs": {},
  "resources": []
}
```

State Commands

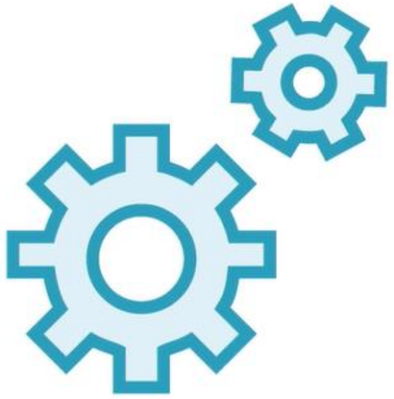
```
# List all state resources

terraform state list

# Show a specific resource

terraform state show ADDRESS

# Move an item in state

terraform state mv SOURCE DESTINATION

# Remove an item in state

terraform state rm ADDRESS
```

Terraform State Commands

5

Terraform Modules

Inputs



Resources

Data sources

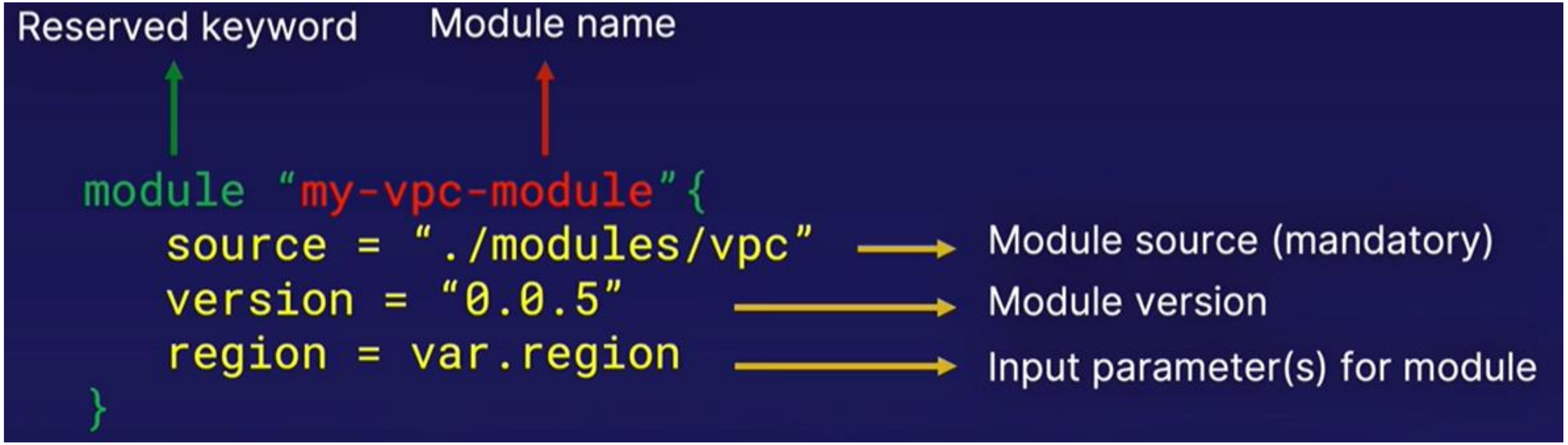
Outputs

What is Terraform Module?

MODULES

- Code reuse
- Remote or local source
- Versioning
- Terraform init
- Multiple instances

Modules Syntax



Module References

module.<name_label>.<output_name>

module.my-vpc-module.subnet_id

6

Built-in Functions and Dynamic Blocks

FUNCTIONS

- Terraform comes pre-packed with functions
- User-defined functions are not allowed
- General syntax: *function_name(arg1, arg2, …)*
- Test in terraform console
- https://www.terraform.io/language/functions

LOOPING

- count
- for_each
- dynamic blocks

COUNT

```
resource "aws_instance" "web_servers" {

  count = 3

  tags = {

    Name = "globo-web-${count.index}"

  }

}
```

Count References:

```
<resource_type>.<name_label>[element].<attribute>

aws_instance.web_server[0].name # Single instance

aws_instance.web_server[*].name # All instances
```

FOR_EACH

```
resource "aws_s3_bucket_object" "taco_toppings" {
  for_each = {
    cheese = "cheese.png"
    lettuce = "lettuce.png"
  }
  key    = each.value
  source = ".${each.value}"
  tags = {
    Name = each.key
  }
```

Count References:

```
<resource_type>.<name_label>[key].<attribute>

aws_s3_bucket_object.taco_toppings[“cheese”].id # Single instance

aws_s3_bucket_object.taco_toppings[*].id # All instances
```

DYNAMIC BLOCKS

- Dynamically constructs repeatable nested configuration blocks inside terraform resources
- Code look cleaner
- Dynamic blocks expect a complex variable type to iterate over
- It acts like a for loop and outputs a nested block for each element in your variable

Example

```
resource “aws_security_group” “my-sg” {
  name   = ‘my-aws-security-group’
  vpc_id = aws_vpc.my-vpc.id
  dynamic “ingress” {
    for_each = var.rules
    content {
      from_port   = ingress.value[”port”]
      to_port     = ingress.value[“port”]
      protocol    = ingress.value[“proto”]
      cidr_blocks = ingress.value[“cidrs”]
    }
  }
}
```

The config block you’re trying to replicate

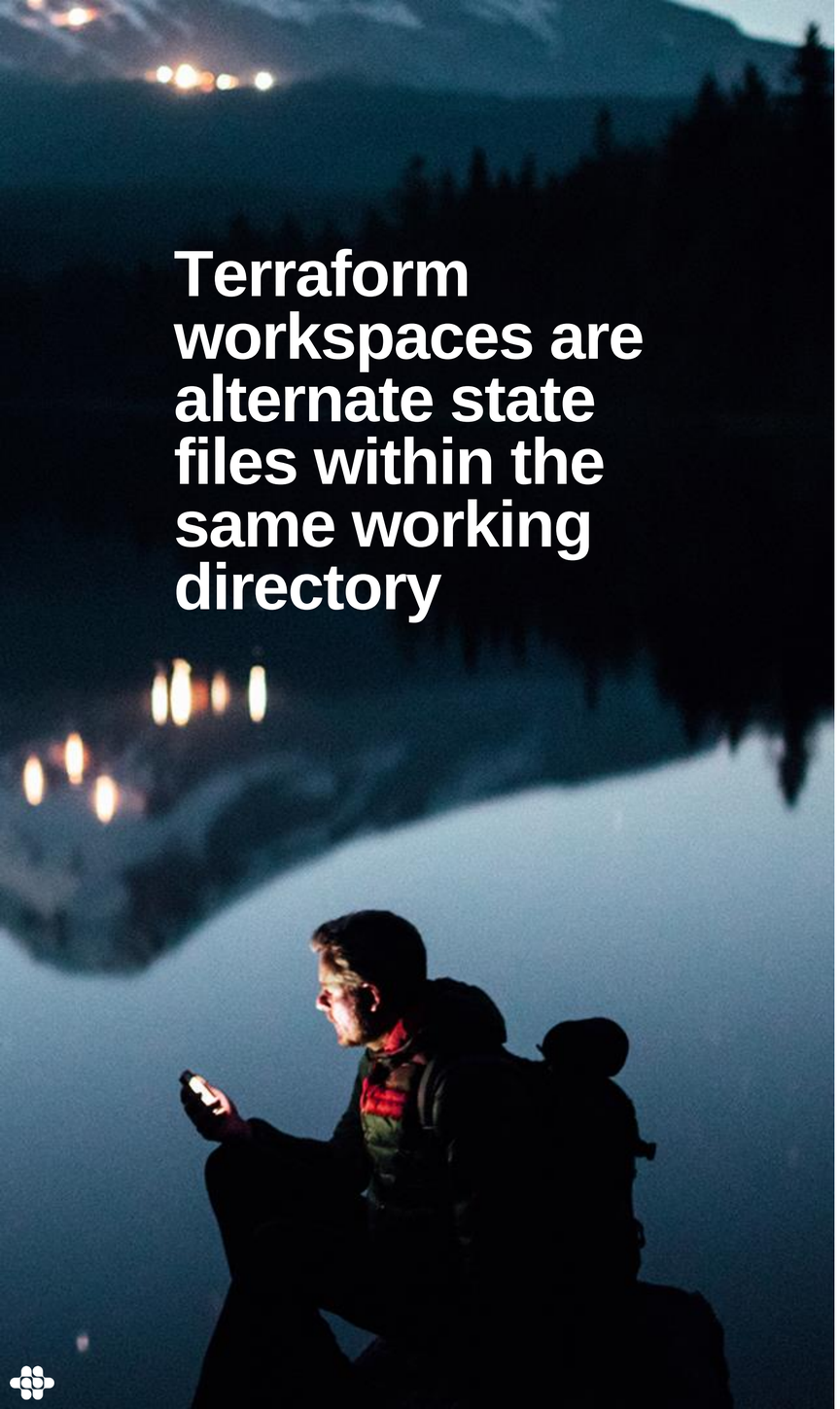
Complex variable to iterate over

The nested ”content” block defines the body of each generated block, using the variable you provided

```
variable “rules” {
  default = [
    {
      port        = 80
      proto       = “tcp”
      cidr_blocks = [“0.0.0.0/0”]
    },

    {
      port        = 22
      proto       = “tcp”
      cidr_blocks = [“1.2.3.4/32”]
    }
]
}
```

7

Terraform Workspaces



Terraform Workspaces (CLI)

Terraform starts with a single workspace called *default*. It cannot be deleted.

Access to a Workspace name is provided through the *${terraform.workspace}* variable

Locally, terraform states for non-default workspace will be stored in terraform.tfstate.d

Useful commands:

- `terraform workspace list`
- `terraform workspace new WORKSPACE`
- `terraform workspace select WORKSPACE`

8

Debugging Terraform

TF_LOG and TF_LOG_PATH

TF_LOG is an environment variable for enabling verbose logging in Terraform. By default, it will send logs to stderr

Can be set to the following levels: TRACE, DEBUG, INFO, WARN, ERROR

TRACE is the most verbose level of logging

To persist logged output, use the TF_LOG_PATH environment variable

Logging

Terraform taint

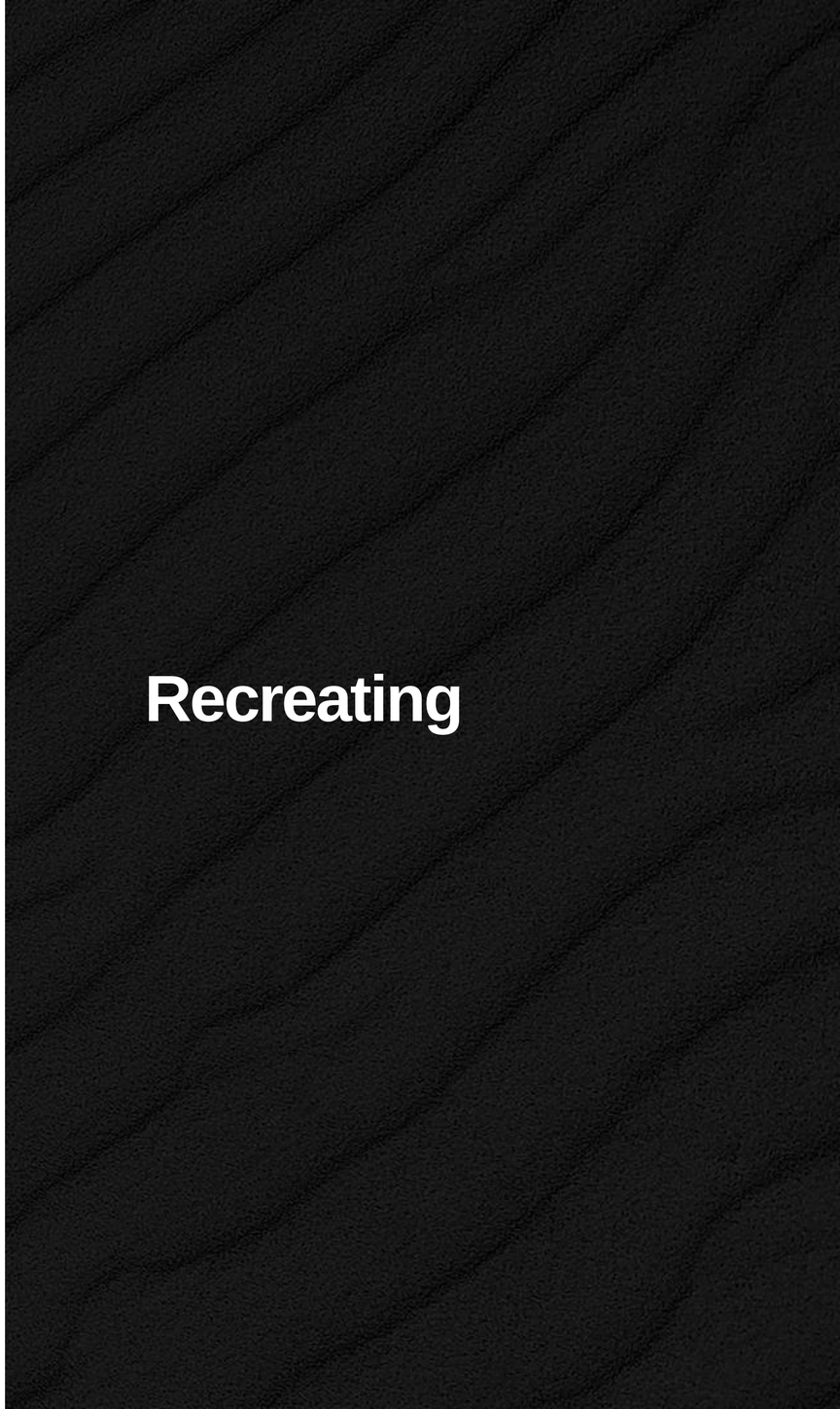
Taints a resource, forcing it to be destroyed and recreated

Modifies the state file, which causes the recreation workflow

Tainting a resource may cause other resources to be modified

Command syntax:

```
terraform taint ADDRESS
```



Terraform import

Maps existing resources to Terraform using and ID

ID is dependent on the underlying vendor. For example to import an AWS EC2 instance you'll need to provide its instance ID

Not import the same resource to multiple Terraform resources.

Command syntax:

```
terraform import ADDRESS ID
```

Importing

Terraform Configuration Block

A special configuration block for controlling Terraforms own behaviour

This block only allows constant values

It can be used for:

- Configuring backend for storing state files
- Specifying a required Terraform version
- Specifying a required terraform provider version and its requirements
- Enable and test Terraform experimental features
- Passing metadata to providers

```
terraform {
      required_version = ">=0.13.0"
      required_providers {
          aws = ">=3.0.0"
      }
}
```

Terraform Configuration Block

9

Q&A

10

Homework

Homework

https://lucid.app/documents/view/e6100ec2-90fd-4e6d-a588-f3d3622ab21a

Details:

For VPC you will use the official module from https://registry.terraform.io. VPC NAT gateway needs to be deployed in a singe AZ, with no HA.
For instances in private subnets, you need to create your own module that will create one autoscaling group with a desired capacity of 1 instance (t3.micro). The instance needs to have a nginx that reply with a simple page that contains the text "Foo" or "Bar". You will call that module twice in order to create the 2 autoscaling group described in the diagram.
Application Load Balancer can be a module or resource (your choice)Bastion will be a an EC2 instance that will have SSH open for a specific IP address (your home IP). It will help you to troubleshoot the instances that are in private subnets.

Region: eu-west-1
Availability Zones: eu-west-1a, eu-west-1b
CIDR: 10.0.0.0/16
Terraform version: 0.13.7

Optional:
Set ALB to redirect / to /foo, with a redirect 301 (not from nginx)
Terraform state to be stored in an S3 bucket
Your home IP address will be retrieved automatically and not hardcoded


endava