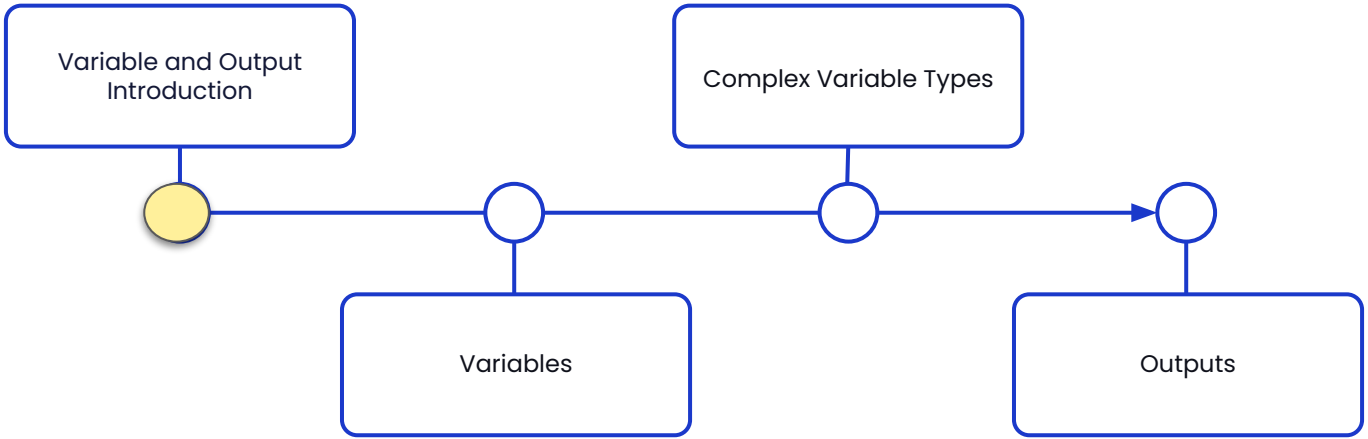
Terraform


Variable and Output Introduction
Complex Variable Types
Variables
Outputs

Variables and Outputs in Terraform

Variables and outputs are fundamental components in Terraform that enhance configuration management.

They enable developers to create dynamic, reusable, and modular infrastructure as code by allowing parameters to be defined and data to be exported.

Understanding their significance is crucial for building efficient and scalable Terraform configurations.

Why Variables and Outputs Matter

Variables enhance configuration flexibility, allowing users to adjust parameters without altering the code, which promotes reusability.

Outputs enable users to extract and share important data from the infrastructure configurations, facilitating better integration between modules.

Together, variables and outputs streamline the management of infrastructure as code, making it easier to maintain, update, and collaborate on projects.

What are Variables?

Definition of Variables

Variables in Terraform are named entities that hold values, allowing users to customize configurations without altering the underlying code.

Purpose of Variables

Their primary purpose is to provide flexibility and reusability in Terraform configurations, enabling parameterization and making it easier to manage different environments.

Benefits of Using Variables

Using variables enhances modularity, reduces duplication of code, and facilitates easier updates and maintenance of infrastructure as code.

What are Outputs?

Definition of Outputs

Outputs in Terraform are a way to declare and export specific values from your configuration, making them available for use after deployment.

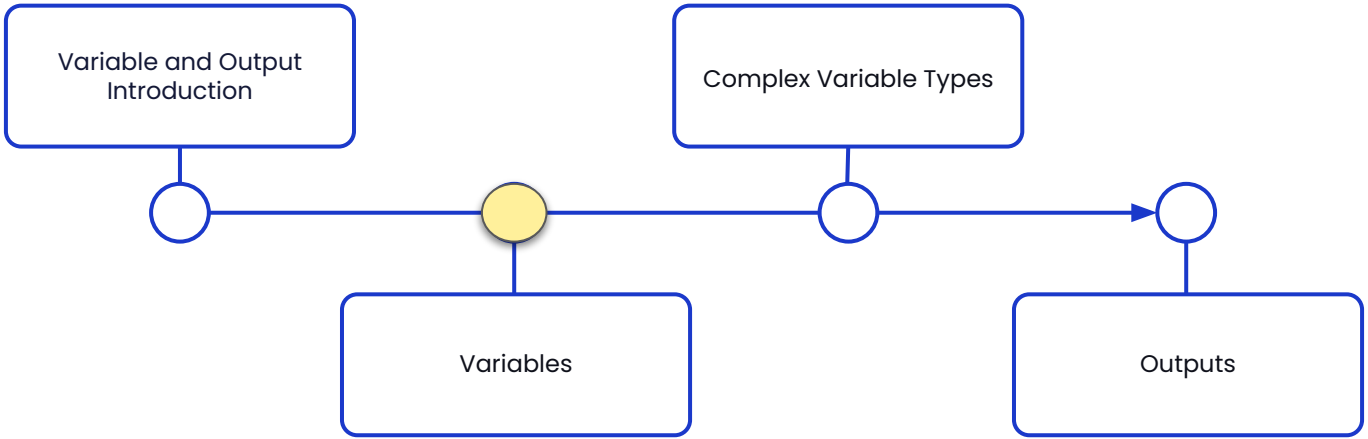
Purpose of Outputs

Outputs help retrieve useful information about your infrastructure, such as IP addresses, resource IDs, and other attributes necessary for further configurations.

Use Cases for Outputs

Common use cases include passing data to modules, sharing information between configurations, and providing essential details to users or other systems.

Terraform


Variable and Output Introduction
Complex Variable Types
Variables
Outputs

Example of Defining a Variable

Basic Variable Definition

```
variable "region" {
    description = "The AWS region to deploy resources in"
    default     = "us-west-2"
}
```

Variable Types in Terraform

String variables hold text values, enabling the configuration of names, descriptions, or any textual data.

Number variables are used for numeric values, allowing you to specify quantities like instance counts or sizes.

Boolean variables represent true/false values, useful for enabling or disabling features in configurations.

Complex types include lists and maps, which allow for organized collections of values, providing flexibility in configurations.

Example of Variable Validation

- Terraform allows validation of variable inputs using the `validation` block within the variable definition.
- For example, to ensure a variable 'instance_count' is a positive integer, use: `validation { condition = var.instance_count > 0 error_message = "Must be a positive integer." }`
- Validation rules can include checks for types, ranges, and specific patterns, enhancing input integrity and reducing errors.

Validating Variable Input

Sensitive Variables

Importance of Sensitive Variables

- Sensitive variables prevent accidental exposure of confidential data.
- Useful for managing secrets like API keys and passwords.
- Enhances security by keeping sensitive data hidden from logs.

Using the Sensitive Attribute

- Add the `sensitive = true` attribute to a variable definition.
- Outputs marked as sensitive will not display in Terraform CLI output.
- This attribute helps comply with security best practices.

Ways to Assign Variable Values

Variable values can be assigned directly in the Terraform configuration files using the variable block.

Values can be set via command-line interface (CLI) using the -var flag for temporary overrides.

Environment variables prefixed with TF_VAR_ can be used to set variable values automatically recognized by Terraform.

Variables can also be defined in .tfvars files, allowing for organized and centralized management of variable values.

Assigning Values via Command Line

- Variables can be set directly in the command line using the -var flag. For example, to set the region variable, you would use: `**terraform apply -var 'region=us-east-1'**`.
- Multiple variables can be assigned simultaneously by repeating the -var flag, like so: `**terraform apply -var 'region=us-east-1' -var 'instance_type=t2.micro'**`.
- This method is useful for temporary overrides, allowing quick changes without modifying the configuration files.

Setting Variables with Environment Variables

In Terraform, environment variables can be used to set variable values dynamically by prefixing the variable name with TF_VAR_. For example, if you have a variable named 'region', you would set the environment variable as 'TF_VAR_region'. This approach is particularly useful in CI/CD pipelines or when you want to avoid hardcoding sensitive information in configuration files. Terraform automatically recognizes these environment variables and assigns their values to the corresponding variables during execution.

Variable Precedence Order

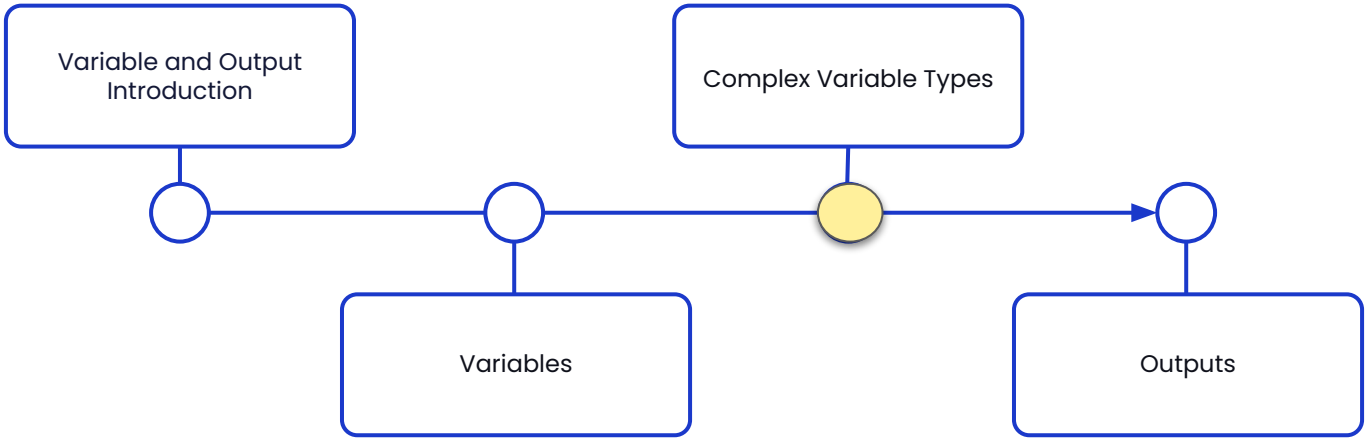
1. Terraform reads variables in a specific order to determine their values, prioritizing more specific sources over default values.

2. CLI variables set using the -var flag take the highest precedence, allowing for temporary overrides.

3. Environment variables prefixed with TF_VAR_ are recognized next, providing flexibility for CI/CD pipelines.

4. Lastly, .tfvars files and default values are considered, ensuring a fallback if no other value is provided.

Terraform


Variable and Output Introduction
Complex Variable Types
Variables
Outputs

Introduction to Complex Variable Types

Lists allow you to store multiple values in a single variable, providing flexibility when dealing with collections of items.

Maps consist of key-value pairs, enabling you to group related data together, which enhances data organization in configurations.

Objects facilitate structured data representation with custom attributes, making it easier to manage complex configurations with multiple parameters.

Using Lists and Maps

Understanding Lists in Terraform

- Lists are ordered collections of items, allowing duplicates.
- Example: variable my_list = ["apple", "banana", "orange"] defines a list of fruits.
- Access elements by their index, e.g., my_list[0] returns "apple".

Utilizing Maps in Terraform

- Maps are key-value pairs, where each key must be unique.
- Example: variable my_map = {"name" = "example", "type" = "module"} defines a map.
- Access values using their keys, e.g., my_map["name"] returns "example".

Nested Variables

Defining and Using Nested Variables

- Nested variables in Terraform allow for the creation of complex data structures, enhancing the organization of configurations.
- Example of a nested list: `subnets = [ ["subnet-1a", "subnet-1b"], ["subnet-2a", "subnet-2b"] ]` allows for specifying multiple subnets in a structured way.
- Example of a nested map: `instance_types = { "web" = { "instance_type" = "t2.micro", "count" = 2 }, "db" = { "instance_type" = "t2.medium", "count" = 1 } }` organizes instance types with associated properties in a readable format.

Example of a Complex Variable Configuration

Real-World Example of Complex Variable Usage

- In this example, we define a variable named 'subnets' that is a map containing multiple AWS subnet configurations.
- Each key in the map represents a subnet name, and the value is an object with attributes like 'cidr_block' and 'availability_zone'.
- This allows for a structured and scalable way to manage multiple subnets in a single variable, simplifying references in resource definitions.

Validating Complex Variables

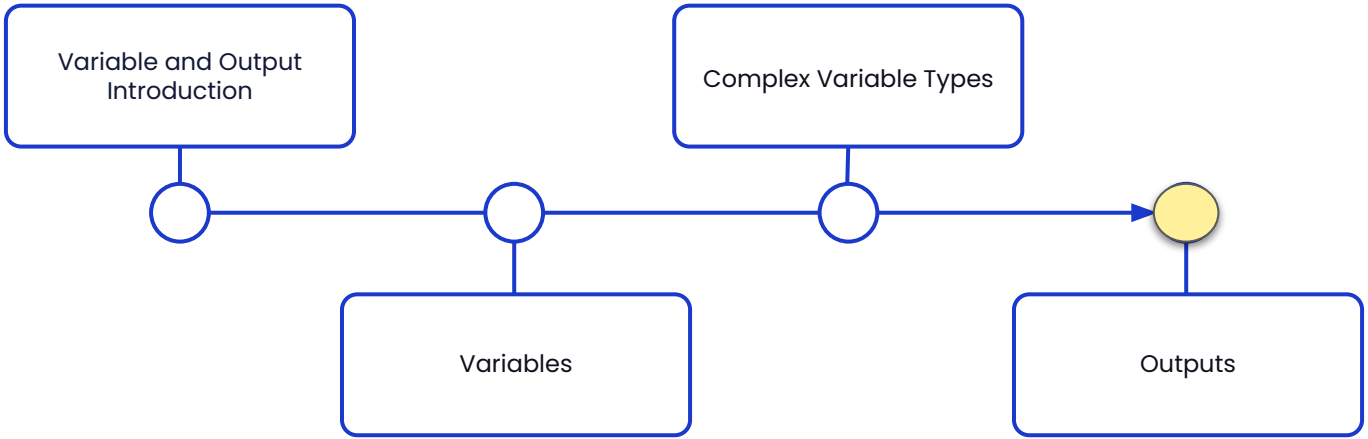
Use of Validation Rules

- Validation rules ensure complex variable inputs meet specified criteria.
- For example, using regex to validate a string format for resource names.
- Type constraints can limit the accepted data types for complex variables.

Examples of Validation

- Example: Validate a list of subnet CIDRs with a specific format: `cidrsubnet()` function checks each entry.
- Example: Use `length()` function to enforce a minimum number of elements in a list variable.
- Example: Create a custom validation rule for object variables to ensure required keys are present.

Terraform


Variable and Output Introduction
Complex Variable Types
Variables
Outputs

Introduction to Outputs

Outputs in Terraform are used to extract and display information from your infrastructure configuration after it has been applied. They enable users to reference specific attributes of resources, such as IP addresses or URLs, making this data accessible for other modules or external applications. Outputs play a crucial role in providing visibility into the created resources, facilitating easier integration and management of complex infrastructures.

Basic Syntax of Outputs

Defining Outputs in Terraform

- An output block starts with the keyword 'output'.
- The output name follows the keyword, which should be a valid identifier.
- The value to be exported is defined using the 'value' argument, specifying what data to output.

Example of Outputting Resource Attributes

- Using the output block in Terraform allows you to export resource attributes after deployment. For example, to output the public IP address of an AWS EC2 instance, you can define: `output "instance_ip" { value = aws_instance.example.public_ip }`.
- This output statement captures the public IP of the EC2 instance named 'example' and makes it accessible after the Terraform apply command is executed.
- Outputs are particularly useful for sharing important resource information with other configurations or modules, enhancing the interconnectivity of your infrastructure.

Outputting Resource Attributes

Using Outputs in Modules

In Terraform, outputs are defined within a module using the `output` block, allowing you to specify what information should be made available after module execution. To define an output, you use the syntax `**output "output_name" { value = <value_expression> }**`. This makes the defined value accessible to other modules or the root module. To retrieve these outputs in the parent module, you can reference them using the syntax `**<module_name>.<output_name>**`, enabling you to create inter-module dependencies and share important data between different parts of your infrastructure.

Sensitive Outputs

Importance of Sensitive Outputs

- Sensitive outputs protect confidential data like passwords and API keys.
- Masking outputs prevents accidental exposure in logs or console.
- Essential for maintaining security in production environments.

How to Use Sensitive Attribute

- Add the 'sensitive' argument in the output block definition.
- Example: output "db_password" { value = var.password sensitive = true }
- Terraform will not display sensitive outputs in the CLI or state files.

Passing Variables to Modules

In Terraform, variables can be passed to modules by defining input variables within the module and providing values in the module's caller configuration. This allows for greater modularity and reusability, as different configurations can use the same module with varying parameters.

When calling a module, use the syntax `module.<MODULE_NAME>.<VARIABLE_NAME> = <VALUE>` to assign values to the module's input variables. This makes it easy to customize the behavior of a module based on the specific requirements of different environments or projects.

Digital Career Institute

Next up

Terraform | Week 2

DAY 4:
Multi-Env Management

LEARN FOR **A NEW LIFE.**

DCI