# **Table of Contents**

[Table of Contents 1](#_Toc186709565)

[The Significance of the terraform.tfvars and output.tf 2](#_Toc186709566)

[**The Significance of the terraform.tfvars** 2](#_Toc186709567)

[**Using -var in Terraform** 4](#_Toc186709568)

[**Understanding the Precedence of Variable Values in Terraform** 4](#_Toc186709569)

[**Using -var-file in Terraform** 7](#_Toc186709570)

[**output.tf in Terraform** 8](#_Toc186709571)

[**Using terraform fmt to Format Terraform Code** 9](#_Toc186709572)

[**Using terraform validate to Validate Terraform Configuration** 10](#_Toc186709573)

# The Significance of the terraform.tfvars and output.tf

## **The Significance of the terraform.tfvars**

The terraform.tfvars file in Terraform is a key mechanism used to define the values of variables in a Terraform configuration. It is often used to simplify and centralize the management of input variables that your configuration depends on.

**Example: Using terraform.tfvars with a Key Name**

Assume you want to manage an AWS EC2 instance in Terraform, and you need to provide a key\_name for accessing the instance. You could define the key\_name variable in your Terraform configuration and set its value in terraform.tfvars.

In the **variables.tf** file, define the **key\_name** variable that you will later pass a value for in terraform.tfvars.

variable "key\_name" {

  type        = string

  description = "key for the EC2 instance"

}

A screenshot of a computer program

Description automatically generated

Provide a value for the key\_name variable in a **terraform.tfvars** file.

key\_name = "newkey"

A screenshot of a computer

Description automatically generated

In the main.tf file, you would reference the key\_name variable to assign it to the key\_name attribute of the aws\_instance resource.

A screenshot of a computer program

Description automatically generated

When you run terraform plan or terraform apply, Terraform will automatically use the value of key\_name defined in terraform.tfvars.

terraform plan



Terraform will use the value newkey for the key\_name when creating the EC2 instance.

A screen shot of a computer

Description automatically generated

## **Using -var in Terraform**

The -var option in Terraform allows you to specify a value for a variable directly from the command line. This is useful when you want to override default values or don't want to manage variable values in separate files. The -var flag is typically used for one-off variable assignments or when you want to provide specific values without modifying configuration files.

Syntax is

terraform plan -var "variable\_name=value"

terraform apply -var "variable\_name=value"

Example:

terraform apply -var "region=us-west-2" -var "instance\_type=t3.medium" -var "key\_name=my-key"

## **Understanding the Precedence of Variable Values in Terraform**

When you provide values for variables in multiple places in Terraform, the order of precedence determines which value is used. Terraform will use the variable values based on this specific order, and if a variable is defined in more than one place, Terraform will use the value from the location with the highest precedence.

The order of precedence is as follows:

1. **Command Line (-var)**
2. **terraform.tfvars**
3. **variables.tf (Variable Declaration)**

Example in Practice: Key name is defined in multiple places as shown in below.

variables.tf

A screenshot of a computer program

Description automatically generated

terraform.tfvars

A black screen with white text

Description automatically generated

main.tf

A computer screen shot of a code

Description automatically generated

If you run **terraform plan -var "key\_name=my-cli-key",** Terraform will use "my-cli-key" as the key name for the EC2 instance, ignoring both terraform.tfvars and the default value from variables.tf.

A screenshot of a computer

Description automatically generated

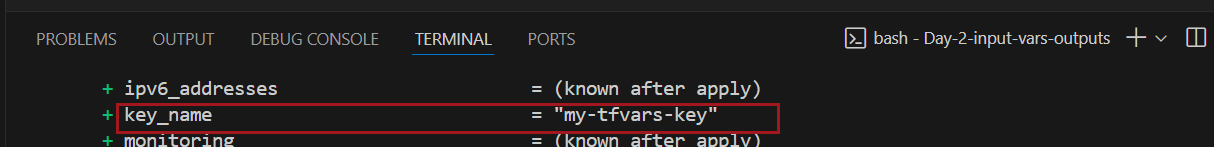
A screen shot of a computer program

Description automatically generated

If you run **terraform apply** (without specifying the -var flag), Terraform will first look for key\_name in terraform.tfvars. Since key\_name is defined there as "my-tfvars-key", it will use that value.

A black screen with yellow and green text

Description automatically generated



If you run terraform apply and neither -var nor terraform.tfvars is provided, Terraform will fall back to the default value defined in variables.tf, which is "default-aws-key".

Comment code in terraform.tfvars

A screenshot of a computer

Description automatically generated

Run terraform plan command

A black screen with yellow and green text

Description automatically generated

A screen shot of a computer

Description automatically generated

## **Using -var-file in Terraform**

The -var-file option in Terraform allows you to specify an external file that contains the variable definitions, rather than passing them individually on the command line or hardcoding them in your Terraform configuration. This option is useful for separating variable values from the Terraform configuration and for organizing values for different environments (e.g., development, production).

terraform plan -var-file="<filename>.tfvars"

First, create a **variables.tf** file where you define your variables. This is where you declare the variables you will be passing values to.

A screenshot of a computer program

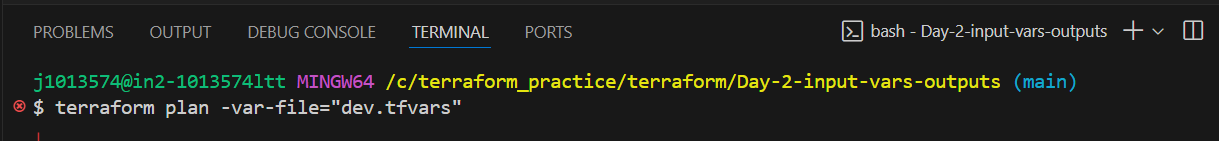
Description automatically generated

Next, create a .tfvars file (for example, dev.tfvars) that contains values for the variables.

A screenshot of a computer

Description automatically generated

Use the -var-file option to specify the .tfvars file during the terraform plan command. This will use the variable values defined in dev.tfvars.



This command will show the execution plan for your configuration, using the variable values defined in dev.tfvars.

A screenshot of a computer

Description automatically generated

## **output.tf in Terraform**

The output.tf file in Terraform is used to define output values that you want Terraform to display after the execution of terraform apply. Outputs are useful for sharing information between different Terraform modules or displaying key data like resource IDs, IP addresses, or any other important information that needs to be made available after the infrastructure is created or modified.

**Structure of output.tf:**

output "output\_name" {

value = <expression or resource attribute>

description = "<optional description of the output>"

sensitive = <optional boolean to hide sensitive information>

}

Let's consider a simple example where we define some outputs for an AWS EC2 instance.

**main.tf**:

A screenshot of a computer code

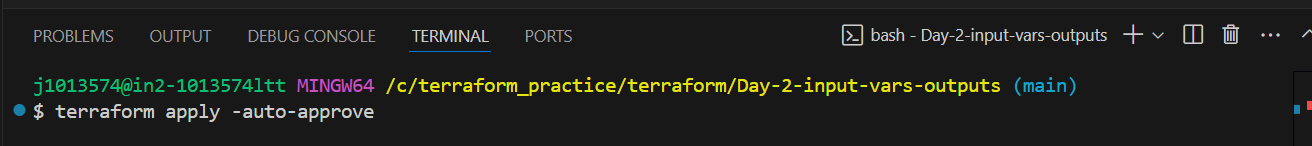
Description automatically generated

**output.tf**:

A screen shot of a computer

Description automatically generated

Run command terraform apply -auto-approve



Terraform will display the defined outputs.

A computer screen with white text

Description automatically generated

## **Using terraform fmt to Format Terraform Code**

The terraform fmt command automatically formats Terraform configuration files (.tf files) according to the official Terraform style guide. This ensures that your code is consistent, readable, and follows best practices for indentation, alignment, and structure.

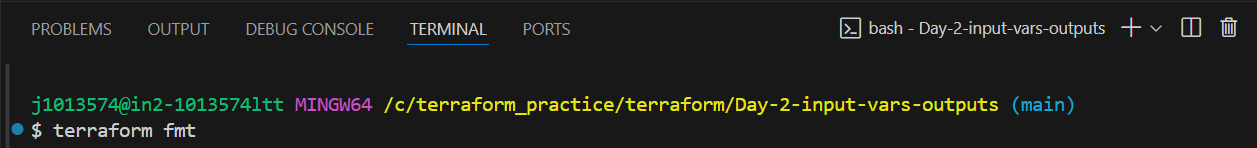
Let's consider you have a Terraform configuration file main.tf with inconsistent formatting:

The formatting is inconsistent: there's no indentation, and the = signs aren't aligned.

A screenshot of a computer

Description automatically generated

Now, run the terraform fmt command in the directory where the main.tf file is located.



After running the command, Terraform will reformat the file as follows:

A screenshot of a computer program

Description automatically generated

## **Using terraform validate to Validate Terraform Configuration**

The terraform validate command is used to check the syntax and configuration of your Terraform files to ensure they are correct before applying them. It helps identify common errors in the Terraform configuration such as missing or incorrect resource definitions, invalid references, or incorrect variable types. However, it does not interact with the actual cloud provider or create infrastructure—its sole purpose is to validate the configuration files.

Let's consider you have a Terraform configuration file main.tf with an invalid resource definition (missing a required argument):

In this example, the ami argument is missing for the aws\_instance resource. Terraform will flag this as an error during validation.

A screenshot of a computer program

Description automatically generated

Now, run the terraform validate command in the directory where main.tf is located:

A screen shot of a computer program

Description automatically generated