What is variable.tf and terraform.tfvars?

**Terraform variable.tf** is a file where you can define [variables](https://developer.hashicorp.com/terraform/language/values/variables) for your Terraform configuration. This file can contain the variable definitions as well as the optional default value for the variable. Here is an example of variable.tf which has -

1. Two variables with no default value - instance\_type, github\_repo
2. One variable with default value - location
3. *# variable.tf*
4. *# No default value*
5. variable "instance\_type" {
6. type = string
7. description = "EC2 Instance Type"
8. }
9. *# No default value*
10. variable "tag" {
11. type = string
12. description = "The tag for the EC2 instance"
13. }
14. *# default value for the variable location*
15. variable "location" {
16. type = string
17. description = "The project region"
18. default = "ap-south-1"
19. }

*TERRAFORM*

Terraform variables.tf

**Terraform.tfvars** is a file where you actually ***assign a values to the variables***. I am just gonna use the previous variable.tf and assign the values to the variables -

*# terraform-dev.tfvars*

instance\_type = "t2.micro"

tag = "EC2 Instance for DEV"

location = " ap-south-1"

*TERRAFORM*

Here are the benefits of using terraform.tfvars -

1. You can have multiple terraform.tfvars based on your project setup for example -
   * **DEV** - terraform-dev.tfvars
   * **QA** - terraform-qa.tfvars
   * **PROD** -terraform-prod.tfvars

Terraform tfvars files for DEV, QA, PROD environment

Table of Content

1. [How to create variable.tf?](https://jhooq.com/terraform-variable-and-tfvars-file/#1-how-to-create-variabletf)
2. [How to create terraform.tfvars?](https://jhooq.com/terraform-variable-and-tfvars-file/#2-how-to-create-terraformtfvars)
3. [How to create multiple .tfvars files for different environments?](https://jhooq.com/terraform-variable-and-tfvars-file/#3-how-to-create-multiple-tfvars-files-for-different-environments)
4. [How do you pass a variable(.tfvars) to the command line to Terraform using --var-file?](https://jhooq.com/terraform-variable-and-tfvars-file/#4-how-do-you-pass-a-variabletfvars-to-the-command-line-to-terraform-using---var-file)
5. [Best practices for using variable.tf and terraform.tfvars](https://jhooq.com/terraform-variable-and-tfvars-file/#5-best-practices-for-using-variabletf-and-terraformtfvars)

Pre-requisite

Before we start working with Terraform variables, here are the pre-requisites -

1. You must install terraform *(*[*click here on how to install terraform*](https://jhooq.com/install-terrafrom/)*)*
2. You must have either [AWS](https://aws.amazon.com/) or [Google Cloud account](https://cloud.google.com/) *(Click to here for*[*AWS*](http://localhost:1313/terraform-ec2-instance-setup/)*and*[*Google Cloud terraform setup tutorial*](https://jhooq.com/how-to-setup-virtual-machine-on-google-cloud-platform/)*)*

1. How to create variable.tf?

Let's take a very basic example to understand the concept of variable.tf in terraform. In this example, we are going to set up an **EC2 Instance** on **AWS.**.

For setting up an **EC2 Instance** we will need the following information -

1. Region - location
2. Instance Type - instance\_type
3. Tags - tag

1.1 Let's create variable.tf file for region, instance type, and tags -

Here is the code for variable.tf -

*# variable.tf*

*# No default value*

variable "instance\_type" {

type = string

description = "EC2 Instance Type"

}

*# No default value*

variable "tag" {

type = string

description = "The tag for the EC2 instance"

}

*# default value for the variable location*

variable "location" {

type = string

description = "The project region"

default = " ap-south-1"

}

*TERRAFORM*

1.2 Create main.tf for provisioning EC2 instance -

Here is the code for main.tf file

provider "aws" {

region = var.location

access\_key = "<INSERT\_YOU\_ACCESS\_KEY>"

secret\_key = "<INSERT\_YOU\_SECRET\_KEY>"

}

resource "aws\_instance" "ec2\_example" {

ami = "ami-0cca134ec43cf708f"

instance\_type = var.instance\_type

tags = {

Name = var.tag

}

}

*BASH*

2. How to create terraform.tfvars?

After creating the variables.tf in [Step-1](https://jhooq.com/terraform-variable-and-tfvars-file/#1-types-of-terraform-variables), let's create .tfvars and in that file, we are going to assign values to the variable -

1. location - "ap-south-1"
2. instance\_type - "t2.micro"
3. tag- "EC2 Instnace for DEV"

*# terraform.tfvars*

instance\_type = "t2.micro"

tag = "EC2 Instnace for DEV"

location = " ap-south-1"

*TERRAFORM*

3. How to create multiple .tfvars files for different environments?

There can be a situation where you need to create multiple tfvars files based on the environment like DEV, QA, PRODUCTION.

So in such scenario, you can create one tfvars file for each environment -

1. terraform-dev.tfvars
2. terraform-qa.tfvars
3. terraform-prod.tfvars

Terraform tfvars files for DEV, QA, PROD environment

Here is the content of terraform-dev.tfvars, terraform-qa.tfvars, terraform-prod.tfvars based on different environment types -

**DEV**

*# terraform-dev.tfvars*

instance\_type = "t2.micro"

tag = "EC2 Instnace for DEV"

location = " ap-south-1"

*TERRAFORM*

**QA**

*# terraform-qa.tfvars*

instance\_type = "t2.micro"

tag = "EC2 Instnace for QA"

location = " ap-south-1"

*TERRAFORM*

**PROD**

*# terraform-prod.tfvars*

instance\_type = "t2.micro"

tag = "EC2 Instnace for PROD"

location = " ap-south-1"

*TERRAFORM*

4. How do you pass a variable(.tfvars) to the command line to Terraform using --var-file?

Referencing the same example from [Step-2](https://jhooq.com/terraform-variable-and-tfvars-file/#2-terraform-tfvars-file---terraformtfvars) we can pass the variables terraform-dev.tfvars, terraform-qa.tfvars, terraform-prod.tfvars based on the environment we are working.

Here is how I am going to run the terraform init, terraform plan and terraform apply command in **DEV, QA, and PROD** environment.

**DEV** - Keep in mind that you have to supply the correct .tfvars file based on the environment you are working on.

*# 1. terraform init for DEV*

terraform init --var-file="terraform-dev.tfvars"

*# 2. terraform plan for DEV*

terraform plan --var-file="terraform-dev.tfvars"

*# 3. terraform apply for DEV*

terraform apply --var-file="terraform-dev.tfvars"

*BASH*

**QA** - Keep in mind that you have to supply the correct .tfvars file based on the environment you are working on.

*# 1. terraform init for QA*

terraform init --var-file="terraform-qa.tfvars"

*# 2. terraform plan for QA*

terraform plan --var-file="terraform-qa.tfvars"

*# 3. terraform apply for QA*

terraform apply --var-file="terraform-qa.tfvars"

*BASH*

**PROD** - Keep in mind that you have to supply the correct .tfvars file based on the environment you are working on.

*# 1. terraform init for PROD*

terraform init --var-file="terraform-prod.tfvars"

*# 2. terraform plan for PROD*

terraform plan --var-file="terraform-prod.tfvars"

*# 3. terraform apply for PROD*

terraform apply --var-file="terraform-prod.tfvars"

*BASH*

5. Best practices for using variable.tf and terraform.tfvars

1. **Separate reusable variables into a separate tfvars file:** Having a separate .tfvars file for all reusable variables provides clarity, readability, and maintainability. Below you will find a screenshot of my project where i have 3 different tfvars for DEV, QA, PROD environment -

Terraform tfvars files for DEV, QA, PROD environment

1. **Lockdown read/write access:** Ensure that only the appropriate people can modify the variable values, whether through IAM policies or other security controls.
2. **Use default values:** Whenever possible, set default values for variables that can be overridden later. This simplifies the process of managing variable values and reduces risk. Here is an example of how to set default values -
3. *# default value assigned for variable `location`*
4. variable "location" {
5. type = string
6. description = "The project region"
7. default = " ap-south-1"
8. }

*TERRAFORM*

1. **Reuse variable names:** Reusing variable names between variable.tf and terraform.tfvars helps reduce confusion and ensures that all variable values referenced in variable.tf can be overridden in terraform.tfvars.
2. **Organize variable values:** Organize, group, and document the variables to provide context and clarity. This will reduce the amount of time needed to understand what values are being used for