FSRAO

Terraform Structure Best Practices

By



Table of Contents

[Directory/Folder Structure 4](#_Toc152231350)

[Naming Methods Conventions 5](#_Toc152231351)

[Module Structure 6](#_Toc152231352)

[Basic File Structure 7](#_Toc152231353)

[“.tfvars” File 7](#_Toc152231354)

[Variable Usage 7](#_Toc152231355)

[Resource Based or Main File 7](#_Toc152231356)

[Outputs File 7](#_Toc152231357)

[Provider Usage 7](#_Toc152231358)

[Statefile Usage and Storage 8](#_Toc152231359)

Version History

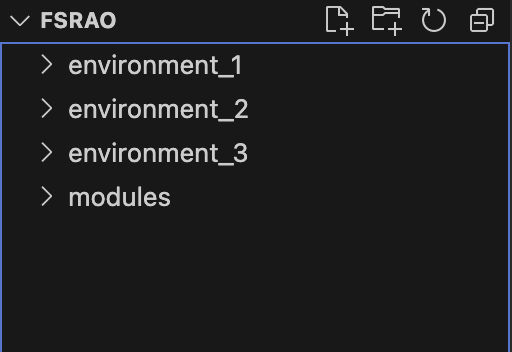
|  |  |  |  |
| --- | --- | --- | --- |
| Version Number | Date Of Change | Changed By | Comments |
| 1.0 | 30/11/2023 | Enes B. Kahraman | Created Document |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Directory/Folder Structure

Best practices dictate that your root directory should be split into folders, one for each environment, and one for your backend modules. Each environment should have its own Terraform code and work independently from one another.

**Isolation of Environments**: The root directory should be organized primarily by environment. This means having separate folders for each environment like “development”, “staging”, “production”, etc. This separation ensures that changes made in one environment do not inadvertently affect another, which is critical for avoiding configuration drift and managing state files securely and efficiently. This not only reduces the risk of accidental changes to the wrong environment but also simplifies the process of applying different security policies and access controls to each environment.

**Backend Modules Folder**: Include a dedicated folder for backend configurations and modules. These backend configurations are essential for defining how and where your Terraform state is stored and managed. Having a centralized location for backend configurations helps in maintaining consistency and reusability across different environments.



Ex: Environment names are purposely generic, should be more specific and meaningful.

# Naming Methods Conventions

**General Principles**

* **Clarity and Descriptiveness**: Names should clearly indicate the purpose of the resource or variable. Avoid ambiguous abbreviations.
* **Consistency**: Apply the same naming conventions throughout your Terraform codebase. Consistency aids in understanding and managing the code.
* **Use of Underscores**: Prefer underscores (\_) over hyphens (-) in naming. Terraform itself often follows this convention in its resources and built-in functions. For example, use app\_security\_group rather than app-security-group.
* **Environment Differentiation**: When managing multiple environments, include the environment in the naming convention to avoid confusion and potential conflicts. For example, “prod\_database\_instance” and “dev\_database\_instance”.

**Specific Guidelines**

* **Resource Naming**: Resource names should reflect the type of resource and its purpose. For example, “VM\_web\_server” for a VM acting as a web server.
* **Variables and Outputs**: For variables and outputs, use a clear, descriptive name that indicates its role. For example, “database\_password” is preferable over vague terms like “db\_pass”.
* **File Naming**: Name Terraform files with clear, descriptive names, related to their purpose or the resources they define. For example, variables.tf for variable definitions.
* **Modular Naming**: When creating modules, choose names that reflect the resource it uses. For example, “virtual\_network” for a module that uses the virtual network resource from your provider.
  + In modules, use variable interpolation to allow customizable naming when the module is used. For these variables, using the same, word for word variable name as the parameter itself allows for the best clarity and reusability results.

# 

Ex: “nic” and “rg1”is a generic name, and should be more specific and meaningful. Take note of the attribute and variable naming usage.

# Module Structure

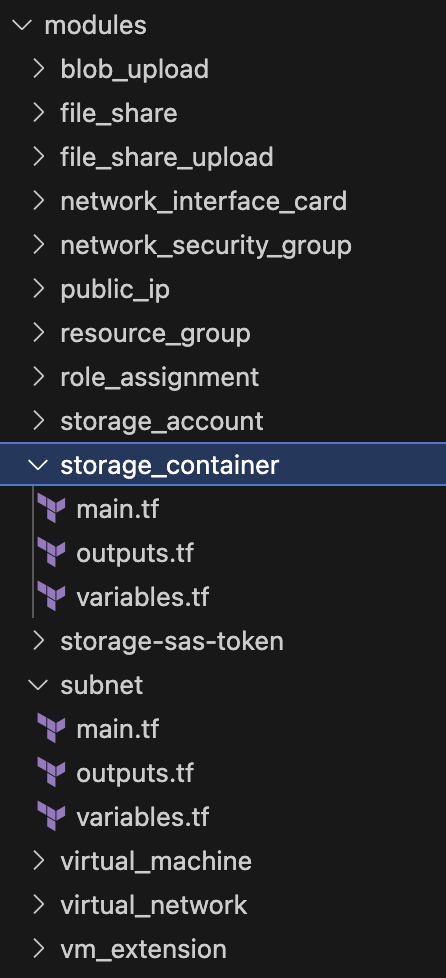
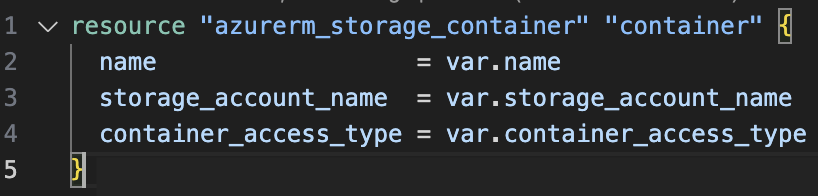
Module usage is a key component to Terraform literacy. Efficiently structuring modules in Terraform is crucial for creating reusable, maintainable, and scalable infrastructure as code. Modules in Terraform are containers for resources and should be thinned down into the smallest components possible for maximum flexibility. This is particularly emphasized in some resources listed in the registry on terraform such as NSGs and their rules, and route tables and their routes.

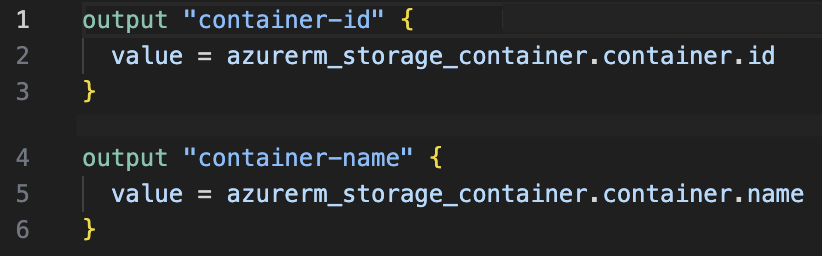
**Modularization**: Create resource-specific modules (e.g., virtual\_network, subnet, windows\_virtual\_machine), and name them accordingly for ease of readability.

**Reusability**: Design modules to use parameterization through input variables. Include outputs for necessary data sharing.

**Separate Directories**: Store each module in its own directory, under a “modules” folder which will be in the root directory.

**Standard Files**: Each module should contain 3 files. This includes “main.tf” for resources, “variables.tf” for inputs, “outputs.tf” for module outputs.







# Basic File Structure

The basic structure for any Terraform set up includes 5 types of files which are listed below:

## “.tfvars” File

A "tfvars" file is used to store environment-specific variables, with a separate file for each environment such as development, staging, or production. This allows for clear separation and precise control over the configuration applied to each environment. This file should contain all your hard code, this is where all your variables are set, maintained, and changed.

## Variable Usage

All the variables are defined in a single file, centralizing the definition of parameters that will be used throughout the Terraform code, thereby simplifying variable management and reference. Each environment should contain only one of these files. Outside of the environment, each module also contains its own variables file, and they are used for variable interpolation, and are to be used in your main/resource files within the environments.

## Resource Based or Main File

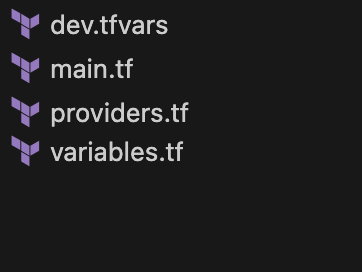
These files are organized around specific resource types or groups, enabling focused management of similar resources, such as network configurations or compute instances, for better organization and readability. This file(s) contains all your resource provisioning and should be used to call modules into use to form your infrastructure.

## Outputs File

Outputs files should only be found within modules, and this is where the attributes of the resources that would be referenced are stored. They define the output values that modules should expose for use by other parts of the Terraform configuration, facilitating modularity and reuse.

## Provider Usage

Best practices dictate that a single provider configuration should be used per environment, which dictates the infrastructure provider (like AWS, GCP, or Azure), and ensures that environment-specific settings and credentials are correctly applied. Including more than one provider increases the margin of error and manual input, and decreases readability, organization, and overall tidiness of code.



Ex: Files that should be found in each environment folder. . Provisioning could be kept in a main file or split into . resource groups.

# Statefile Usage and Storage

**Separate Statefiles Across Storage Accounts**: Use dedicated storage accounts for each environment's statefile to enforce strict isolation and security.

**Storage Account Versioning**: Implement versioning for storage accounts to safeguard statefiles with historical recovery options.

**Remote State Backends**: Utilize remote backends like Azure Blob Storage to store state files, providing a centralized and secure location for team access.

**State Locking**: Implement state locking to prevent conflicting operations on your state files.

**Encryption at Rest**: Ensure state files are encrypted at rest within the storage account, using the provided encryption features of the cloud storage service.

**Access Control**: Fine-tune access controls to state files to ensure only authorized personnel can read or modify the state, using IAM roles or policies.

**Statefile Segregation**: Keep state files for different Terraform projects or modules separate to avoid interference and improve clarity.

**Backup and Recovery**: Enable automatic backups for state files if the backend supports it, and have a recovery plan in place in case of data loss.

**Sensitive Data Handling**: Be cautious of any sensitive data that may be stored in the state file and leverage secrets management tools to handle such data.

**Monitoring and Alerting**: Set up monitoring for access and changes to the state files and configure alerts to notify of any unauthorized modifications.