To create a reusable Terraform module for managing common tags, you can define a module that includes a set of default tags and allows adding custom tags for individual resources. Here's how to structure such a module:

1. Create the Module Directory Structure

```
plaintext
Copy code
tags-module/
  ├── main.tf
  ├── variables.tf
  ├── outputs.tf
```

2. Define Common Tags in `variables.tf`

Define the variables for common tags and additional custom tags.

```
hcl
Copy code
variable "common_tags" {
  description = "A map of common tags applied to all resources"
  type        = map(string)
  default     = {
    Environment = "production"
    ManagedBy   = "terraform"
  }
}

variable "additional_tags" {
  description = "A map of additional tags to be merged with common
tags"
  type        = map(string)
  default     = {}
}
```

3. Combine Tags in `main.tf`

Merge the common tags with the additional tags using the `merge` function.

```
hcl
Copy code
output "combined_tags" {
  value = merge(var.common_tags, var.additional_tags)
}
```

This output can then be used in the parent module to apply the combined tags to resources.

4. Output the Combined Tags in `outputs.tf`

Expose the combined tags for use.

```
hcl
Copy code
output "tags" {
  description = "Merged tags to be used for resources"
  value       = merge(var.common_tags, var.additional_tags)
}
```

5. Use the Module in a Parent Module

Here's an example of how to use this module in a parent module.

```
hcl
Copy code
module "tags" {
  source          = "./tags-module"
  common_tags     = {
    Environment = "staging"
    ManagedBy   = "terraform"
  }
  additional_tags = {
    Project = "MyProject"
```

```
    Owner   = "team@example.com"
  }
}

resource "aws_s3_bucket" "example" {
  bucket = "example-bucket"

  tags = module.tags.tags
}
```

6. Example Execution

- The `common_tags` in the module are overridden if provided in the parent module.
- The `additional_tags` are merged with `common_tags`.
- The resulting tags for the S3 bucket in this example would be:

```
hcl
Copy code
tags = {
  Environment = "staging"
  ManagedBy   = "terraform"
  Project     = "MyProject"
  Owner       = "team@example.com"
}
```

This modular approach ensures flexibility and consistency while managing resource tags across different Terraform configurations.

Creating a Terraform module with a set of common variables is a great way to standardize configurations and improve maintainability. Here's how you can create a module with common variables:

1. Module Directory Structure

```
plaintext
```

```
Copy code
common-variables/
  ├── main.tf
  ├── variables.tf
  ├── outputs.tf
```

2. Define Common Variables in `variables.tf`

List variables that are frequently used across different Terraform configurations.

```
hcl
Copy code
variable "region" {
  description = "The AWS region where resources will be deployed"
  type        = string
  default     = "us-east-1"
}

variable "environment" {
  description = "The environment for the resources (e.g., dev,
staging, production)"
  type        = string
  default     = "dev"
}

variable "project_name" {
  description = "The name of the project"
  type        = string
  default     = "my_project"
}

variable "tags" {
  description = "A map of tags to apply to resources"
  type        = map(string)
  default     = {
    ManagedBy = "terraform"
  }
```

```
}
```

3. Expose Outputs in `outputs.tf`

Expose the variables as outputs so they can be reused in the parent module.

```
hcl
Copy code
output "region" {
  value = var.region
}

output "environment" {
  value = var.environment
}

output "project_name" {
  value = var.project_name
}

output "tags" {
  value = var.tags
}
```

4. Use the Module in a Parent Configuration

You can now reference the common variables module in a parent Terraform configuration.

```
hcl
Copy code
module "common_variables" {
  source      = "./common-variables"
  environment = "production"
  project_name = "example_project"
  tags = {
    Owner   = "team@example.com"
```

```
    Purpose = "example"
  }
}

resource "aws_s3_bucket" "example" {
  bucket = "${module.common_variables.project_name}-bucket"

  tags = module.common_variables.tags
}
```

5. Customize and Extend as Needed

- Add more variables for frequently used attributes like `vpc_id`, `subnet_ids`, or `availability_zones`.
- Allow overrides in the parent configuration for flexibility.

Benefits of This Approach

- **Standardization:** Common variables ensure uniformity across resources.
- **Reusability:** The module can be shared across different projects.
- **Flexibility:** Variables can be overridden or extended as needed.

This approach simplifies your Terraform configurations while keeping them organized and scalable.