IMPORT WITH VARIABLES

```
#main.tf

provider "aws" {
  region = var.aws_region
}

resource "aws_instance" "tf_example_import" {
  for_each      = var.ec2_ids
  ami           = var.ami_id
  instance_type = var.instance_type

  tags = {
    Name = "TF-example-import-${each.key}"
    role = "terraform"
  }
}

# Single loop for imports
import {
  for_each = var.ec2_ids
  to       = aws_instance.tf_example_import[each.key]
  id       = each.value
}
```

IMPORT WITH VARIABLES - 2

```
# variables.tf
variable "aws_region" {
  default = "us-west-1"
}

variable "ami_id" {
  default = "ami-0945610b37068d87a"
}

variable "instance_type" {
  default = "t2.micro"
}

variable "ec2_ids" {
  description = "Map of logical names to existing EC2 instance IDs"
  type        = map(string)
  default = {
    one   = "i-0aaa111bbb222ccc0"
    two   = "i-0ddd333eee444fff1"
    three = "i-0ggg555hhh666iii2“
    four  = "i-0ddd333eee444fff1"
    five = "i-0ggg555hhh666iii2"
  }
}
```

EXTENDING AWS

- Side-by-side provider: Ship your own provider (e.g., example/awsx) with resources aws resources. Use both hashicorp/aws and your provider in the same config and pass values between them via references.
- Modules, not providers: Wrap AWS resources in modules. This is the usual way to add behavior/guardrails on top of the official provider.
- Bridge/shim provider: Build a provider that talks to your system and AWS (via SDK), exposing new resource types (myaws_*) that orchestrate across both.

BLOCK ORDER

- terraform block (backend, required_providers, required_version)

Processed at init. Cannot use variables or resources.

- variable

Values fixed before graph build. Can’t reference resources.

- locals

Evaluated after variables; can reference var.*/local.*, not resources.

- provider

Configured before planning; can reference vars/locals/env. Child modules inherit unless overridden or re-mapped via providers {} on a module block.

BLOCK ORDER - 2

- module

Inputs can reference vars/locals/resources from the calling module in ways that form valid dependencies.

- data

Runs during plan (read-only). May depend on vars/locals and, if referenced, on resources

- resource

Planned, then created/updated/destroyed during apply in dependency order.

- output

Evaluated after apply; can reference anything in final state; supports sensitive

PROVIDER - CONFIGURE

```
// Provider Configure
func (p *MyProvider) Configure(
          ctx context.Context,
          req provider.ConfigureRequest,
          resp *provider.ConfigureResponse,
) {
          var config ProviderModel
          diags := req.Config.Get(ctx, &config)
          resp.Diagnostics.Append(diags...)
          if resp.Diagnostics.HasError() {
                    return
          }
          client := NewAPIClient(config.ApiUrl.ValueString(), config.ApiToken.ValueString())
          // Pass to resources and data sources
          resp.DataSourceData = client
          resp.ResourceData = client
}
```

RESOURCE - CONFIGURE

```
// Resource struct
type WidgetResource struct {
    client *APIClient
}

// Resource Configure
func (r *WidgetResource) Configure(
    ctx context.Context,
    req resource.ConfigureRequest,
    resp *resource.ConfigureResponse,
) {
    if req.ProviderData == nil {
        return
    }
    r.client = req.ProviderData.(*APIClient)
}
```

SURVEY - TYPES

Survey types

- Text (single line input)
- Textarea (multi-line input)
- Password (hidden input, stored securely)
- Multiple Choice (Single Select)
- Multiple Choice (Multiple Select)
- Integer / Float (numeric input with optional range)

Best practices

- Use multiple-choice when possible (limits user error).
- Keep questions short and clear.
- Map answers to expected playbook variables.
- Validate input ranges when using free-text or numeric fields.

TEMPLATES (.J2)

```
Hello {{ user_name }}, welcome to {{ app_name }}!
```

```
---
- name: Render welcome file
  hosts: localhost
  gather_facts: false
  vars:
    user_name: "Donis"
    app_name: "Ansible Demo"
  tasks:
    - name: Create /tmp/welcome.txt from template
      ansible.builtin.template:
        src: templates/welcome.txt.j2
        dest: /tmp/welcome.txt
```

AWS_EC2

```
plugin: aws_ec2
regions: -
        us-west-1
filters:
        instance-state-name: running
keyed_groups: -
        key: tags['role’]
        prefix: tag_role
hostnames:
        - ip-address
```

ANSIBLE / TERRAFORM

Using an Ansible provider in Terraform is debated because the tools serve different purposes. Terraform manages infrastructure state, while Ansible is better at configuration. Combining them can feel unnatural since Terraform wants a fixed end state, while Ansible often just runs tasks.

ANSIBLE / TERRAFORM - 2

The provider has been called incomplete and can lead to state drift—Terraform doesn’t always know what Ansible changed. This makes results less predictable and adds maintenance risk in complex setups.

Supporters like the convenience of handling provisioning and configuration in one place. With the provider, a single Terraform plan can build servers and then configure them, reducing the need for extra scripts or pipelines.

Most teams, however, keep Terraform and Ansible separate but connected: Terraform first, then Ansible through CI/CD. The provider exists, but many view it as immature and recommend it only for simple use cases.

TERRAFORM – ANSIBLE PROVIDER

```
terraform {
  required_providers {
    ansible = {
      source  = "ansible/ansible"
      version = ">= 1.2.0"
    }
  }
}
```

TERRAFORM – ANSIBLE INVENTORY

```
resource "ansible_host" "web" {
  name    = "203.0.113.10"           # instance IP/DNS
  groups = ["web"]
  variables = {
    ansible_user                 = "ec2-user"
    ansible_ssh_private_key_file = "~/.ssh/id_rsa"
  }
}
```

TERRAFORM – ANSIBLE PLAYBOOK

```
# Run an Ansible playbook against that host
resource "ansible_playbook" "configure_web" {
  playbook   = "${path.module}/site.yml"  # your Ansible playbook
  name       = ansible_host.web.name
  depends_on = [ansible_host.web]
}
```

LIMITED FACTS

```
---
- name: Collect only the distro name and version
  hosts: all
  gather_facts: false
  tasks:
    - name: Get just distro facts
      ansible.builtin.setup:
        filter:
          - ansible_distribution
          - ansible_distribution_version

    - name: Show them
      ansible.builtin.debug:
        msg: "{{ ansible_facts.ansible_distribution }} {{ ansible_facts.ansible_distribution_version }}"
```