Chapter 03

Objectives

- What is template
- What is provider
- What is module

First meet to `data source`

- Allow data to be **fetched or computed** for use elsewhere in Terraform configuration.
- Allow a Terraform refer information defined outside of Terraform, or defined by another separated Terraform configuration.

```
# Find the latest available AMI that is tagged with Compo
data "aws_ami" "web" {
  filter {
    name   = "state"
    values = ["available"]
  }

  filter {
    name   = "tag:Component"
    values = ["web"]
  }

  most_recent = true
}
```

Launch instance by selected AMI

```
resource "aws_instance" "web" {
  ami           = "${data.aws_ami.web.id}"
  instance_type = "t1.micro"
}
```

Template

Template

- Exposes `data sources` to custom template
- To generate strings for other Terraform `resources` or `outputs`

Use `template`

```
data "template_file" "foo" {

}
```

- use `template_file`
- template has 2 types: `file` and `inline`

Option 1 - inline template

```
data "template_file" "init" {
  template = "$${ip_address}:1234"

  vars {
    ip_address = "${aws_instance.foo.public_ip}"
  }
}
```

- `vars{}` block for input of template
- `$${consul_address}`, escaped, interpolations at runtime

Option 2 - template file

Given `nginx.tpl` to replace `ip address`:

```
server {
    listen       80;
    server_name  ${ip_address};
    ...
}
```

Configure to load template

```
data "template_file" "my_output" {
  template = "${file("${path.module}/nginx.tpl")}"

  vars {
    ip_address = "${aws_instance.foo.public_ip}"
  }
}
```

- load template file `nginx.tpl` and interpolate variable
- Useful trick `${path.module}`

Use rendered template

Template output is stored in `rendered`, usage it as below:

```
output "dump" {
    value = "${data.template_file.my_output.rendered}"
}
```

- `rendered` attributes
  - template output attributes

Practice #00 `aws/ch03/practices/300-template`

- Use `aws_instance` data source to get running instance
- Retrieve the `public_ip` of `aws_instance` and render template
- Practice 300-template

Provider

Provider

- Provider is used **to create, manage, and update** infrastructure resources
- Understanding **API interactions** and exposing resources
    - AWS, GCP, Azure, VMware vSphere ... etc

Other providers (doc)

- Github
- Gitlab
- RabbitMQ
- VMware vSphere
- Palo Alto Networks
- ... etc

Null Provider

- A unusual provider
- It does nothing
- When `triggers` argument changes, will cause the resource to be replaced

```
resource "null_resource" "cluster_setup" {
  triggers = {
    cluster_instance_ids = "${join(",", aws_instance.clus
  }
}
```

Triggered when:

- "10.0.0.1,10.0.0.2" -> "10.0.0.1,10.0.0.2, `10.0.0.3` "

Practice #01 `aws/ch03/practices/301-null-provider`

- Base on created instance at ch01
- Configure `null provider` trigger by instance ids and apply
- Scaling instnace and check triggers by null provider
- Practice 301-null-provider

Part 1

```
$ cd aws/ch03/practices/301-null-provider
$ terraform init
$ terraform apply (check result of `dump`)
```

Part 2 - scale out to 2 instance

```
$ cd aws/ch01/practices/102-remote-state-variables
$ vim main.tf (uncomment `count=2`)
$ terraform apply -var-file=./prod.tfvar
```

Part 3 - check trigger (dump result changed)

```
$ cd aws/ch03/practices/301-null-provider
$ terraform apply (check result of `dump`)
```

Module

Module

- Self-contained packages of Terraform configurations
- Manage configurations as a **group**
- Encapsulate their own resource
    - A resource in `A module` cannot directly depend on resources or attributes in `B module`
    - Export data through `outputs`
- Versioning and can be hosted

How to create a module?

Actually, you already created one.

Root Module

- In working directory, you've `terraform apply`
- These files are composed as a valid module

Use Module

Module configuration

```
module "foo" {
  source  = "devopsdays/module1"
  servers = 3
  version = "<=0.0.5"
}
```

- Only configure module name, `foo`
- `terraform init` trigger module installation

Module configuration (cont.)

- `source`
    - Terraform Registry
    - Github, Bitbucket, S3, Generic Git/Mercurial (doc)
    - Local path
- `version`
    - Only support on
        - Terraform Registry
        - Terraform Enterprise's private module registry
    - constraint like: `>=`, `<=`, `~>`, `>=1.0.0, <=2.0.0`
        - ~> 1.2.`0`: any version >= 1.2.0 and < 1.3.0, e.g. 1.2.`X`
        - ~> 1.`2`: any version >= 1.2.0 and < 2.0.0, e.g. 1.`X.Y`

Module has outputs

- You cannot acces resource in module directly
- All neccessory data exported through `outputs`

Reference output of module:

```
resource "example" "foo" {
  ami                    = "ami-20180915"
  instance_type          = "t1.micro"
  server_ip = "${module.module_a.public_ip}"
}
```

module: `vishwakarma`

- `vishwakarma/examples/eks_worker/main.tf`

```
module "network" {
  source           = "../../aws/network"
  aws_region       = "${var.aws_region}"
  bastion_key_name = "${var.key_pair_name}"
  key_pair = "devopsdays-workshop"
}
```

Advanced topics

Create you own Module

Creating Module

- Standard structure:

```
$ tree complete-module/
.
├── README.md
├── main.tf
├── variables.tf
├── outputs.tf
├── ...
├── modules/
    ├── nestedA/
    │   ├── README.md
    │   ├── variables.tf
    │   ├── main.tf
    │   ├── outputs.tf
    ├── nestedB/
```

- include `nestedA module`

```
module "network" {
        source = "modules/nestedA"
}
```

Creating Module (cont.)

- Module = folder with Terraform files
- Keep provider at root module
- More detail will be in ch04

Practice #02 `aws/ch03/practices/302-create-a-module`

- Create a module with a simple output
- Use the module and pass variable to a module
- Output the variable from the output of module
- Practice 302-create-a-module

Key takeawys

- template can genereate and compose formated string from other sources
- provider integrated with vendor API and interact with it
- module groups configurations and let you reuse pre-defined operations

Appendix

practice : `aws/ch03/practices/309-github-create-org-repo`

NOTE: Only works for github **account had organization**

- Create Github access token
  - https://github.com/settings/tokens/new
- Generate deployment key
  - `ssh-keygen -t rsa -f key`