Structure Your Terraform Project

The following structure has been found to work well for Terraform projects. The idea is to put as much Terraform code as possible into modules and then have separate Terraform folders for each environment. There are two potential project structures:

This structure uses account/region for the environments. If you do not need to deploy to more than one region, that layer can be left out. This works best when there is only one environment in a region or when you want it to be clear where things are deployed:

```
project-root/
    accounts/
        sandbox/
            us-east-1/
                config.tf
                main.tf
                variables.tf
        testing/
            us-east-1/
                # config/main/variables
            us-west-2/
                # config/main/variables
        production/
            us-east-1/
                # config/main/variables
            us-west-2/
                # config/main/variables
    modules/
        my-service/
            main.tf
            outputs.tf
            variables.tf
```

This structure uses environment/ for the environments. This works best when there is more than one environment in a region.

```
project-root/
    environments/
        development/
            config.tf
            main.tf
            variables.tf
        qa/
            # config/main/variables
        integration-testing/
            # config/main/variables
        production/
            # config/main/variables
    modules/
        my-service/
            main.tf
            outputs.tf
            variables.tf
```

Module Reference Example

The environment's main.tf will reference any modules via relative path. The modules main.tf will reference any shared modules via GitLab major version branch. See the Module Versioning section in the Terraform Architecture document.

An example of an environment's main.tf linking to a project module and a shared module:

```
module "my-project-service" {
    source      = "../../modules/my-service"
    environment = var.environment
}

module "my-standalone-server" {
    source      = "git::ssh://git@gitlab.directsupply.cloud/terraform-modules/standalone-ec2.git?ref=
    environment = var.environment

    ...
}
```

Deployment

Builds that deploy projects like this will `cd` into the proper directory from which to run the `terraform apply`.

CI/CD guidelines

Here are some examples for how the platform teams recommend structuring Terraform projects and preventing unnecessary complexity.

Not Recommended

The example in this section shows a low level of cohesion. It separates Terraform configuration from the code and moving it into gitlab pipelines increases system coupling into gitlab.

While you will need to do this for secrets, please don't do it for the rest of your configuration. Separating the configuration from the code requires a developer to set a huge amount of variables and makes local Terraform debugging much more difficult because of that.

```
.plan-terraform: &plan-terraform
  image:
    name: registry.directsupply.cloud/docker-images/ds-builder
  script:
    - vault login -method=aws header_value=$VAULT_ADDR role=deployment-bastion-read-east > /dev/null
    - RABBIT_ADMIN_PASSWORD=`vault kv get -field=admin_password secrets/dsi/shared/rabbit/`
    - RABBIT_BUS_PASSWORD=`vault kv get -field=rabbit_bus_password secrets/cis/example/`
    - SQL_SERVER_PASSWORD=`vault kv get -field=database_password secrets/cis/example/`
    - cd $CI_PROJECT_DIR/terraform
    - terraform_0.12.10 init -backend-config="bucket=$BACKEND_BUCKET" -backend-config="key=$BACKEND_C
    - terraform_0.12.10 plan -var="image_version=$CI_PIPELINE_ID" -var="environment=$ENVIRONMENT" -va
      -var="lifespan=$LIFESPAN" -var="domain=$DOMAIN" -var="new_relic_app_name=$NEW_RELIC_APP_NAME" -
      -var="rabbit_admin_username=$RABBIT_ADMIN_USERNAME" -var="rabbit_admin_password=$RABBIT_ADMIN_F
      -var="rabbit_bus_password=$RABBIT_BUS_PASSWORD" -var="rabbit_bus_username=$RABBIT_BUS_USERNAME'
```

Recommended

When injecting secret variables into the plan or apply, please use `TF_VAR_` environment variables. These are automatically understood by Terraform. They also make it easier to run code locally and switch from bash to cmd or powershell. The below example is a rewrite of the yaml from above. The configuration was moved into the environment's main.tf.

```
.plan-terraform: &plan-terraform
  image:
    name: registry.directsupply.cloud/docker-images/ds-builder
  script:
    - vault login -method=aws header_value=$VAULT_ADDR role=deployment-bastion-read-east > /dev/null
    - export TF_VAR_rabbit_admin_password=`vault kv get -field=admin_password secrets/dsi/shared/rabb
    - export TF_VAR_rabbit_bus_password=`vault kv get -field=rabbit_bus_password secrets/cis/example/
    - export TF_VAR_image_version=$CI_PIPELINE_ID
    - cd $CI_PROJECT_DIR/terraform/accounts/$ACCOUNT
    - terraform_0.12.10 init
    - terraform_0.12.10 plan
```

```
module "api" {
  source = "../../modules"

  account                       = "sandbox"
  app_url                       = "myurl"
  consul_addr                   = "https://consul.service.consul"
  domain                        = "sandbox.directsupply-sandbox.cloud"
  environment                   = "development"
  environment_short             = "DEV"
  image_version                 = var.image_version
  new_relic_app_name            = "CIS - DEV - example"
  rabbit_admin_username         = var.rabbit_admin_username
  rabbit_bus_endpoint           = "rabbitmq://devtransit.directs.com/abc123"
  rabbit_bus_password           = var.rabbit_bus_password
  rabbit_bus_username           = "unityRabbitMQ_someApp_DEV"
  sql_server_connection_string = "Data Source=SQL-ABC-DEV-TRAN.directs.com;Initial Catalog=DB;Applica
  sql_server_username           = "AUser"
  vault_addr                    = "https://vault.us-east-1.management.directsupply-sandbox.cloud"
}
```