Infrastructure as Code

Infrastructure-as-Code

2018 State of DevOps recommends IAC as one of the capabilities that are statistically shown to improve software delivery and operational performance.

Respondents using infrastructure as code are 1.8 times more likely to be in the elite performance group.

Benefits

Code Management	Commit, version, trace and collaborate, just like source code
Declarative	Specify the desired state of infrastructure, not updates
Auditable	Diff infrastructure between desired state and current state
Portable	Build reusable modules across an organization

Declarative Infrastructure

Declarative (statement)

YAML

“I should have five servers”

VS

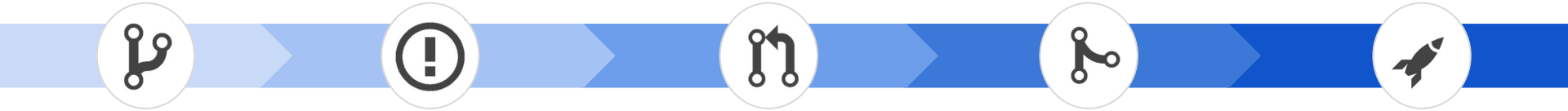
Imperative (command)

“Give me five servers”

Automation Pipeline

The toolkit encourage customers to collaborate on infrastructure through GitOps.

- **Collaborate** in source control
- **Collaborate** in source control
- Ensure **consistency**
- Enforce **policies** proactively



Developer submits **Pull Request**

CI runs **Validation**

Administrator reviews for **Policy Compliance**

Administrator merges the **New Config**

CD updates **Deployed Infrastructure**

Tencent Cloud

Design Principles

Immutable infrastructure

Release pipelines

Integration tests

Rollbacks

Incorporate monitoring signals with deployment

Manual judgements

Terraform

Hashicorp Terraform

Open Core	Open Source core developed by Hashicorp, with enterprise version available
Provisioning	Focus on provisioning infrastructure, not configuring it (cf. Chef/Puppet)
Multi-cloud	Support for dozens of different providers via pluggable design, notably AWS, GCP, VMWare, DNS, Kubernetes, and **Tencent**
Shareability	Resources can be shared using modules, and independent providers can be developed in Go

Key Concepts

01

Resource Configuration

Declarative configuration of the infrastructure setup using Hashicorp Configuration Language (HCL)

02

State

Terraform's view of existing infrastructure and desired outcome

03

Provider

Implements API calls to create/update/delete resources

The Language (HCL)

- **configurable language**, not a programming language
- limited **set of primitives**
  - variables
  - resources / data
  - outputs
  - modules
  - locals
  - backends / providers
- no traditional statements or control loops
- logic expressed through **assignments**, **count**, and **interpolation functions**

```
variable "region" {
  default = ""
}

locals {
  header = "vpc"
}

resource "tencentcloud_vpc" "items" {
  for_each = { for s in var.vpcs : join("_", [var.region,
var.management.cost_center,
  var.management.environment, s.name]) => s }
name = join("_", [local.header, var.region,
var.management.cost_center,
  var.management.environment, each.value.name])

 ...
}

output "vpcs" {
  value = { for k, v in tencentcloud_vpc.items : k => v }
}
```

Tencent Cloud

Providers

- think of **providers as “plugins”**
- providers **expose specific APIs as Terraform resources**, and manage their interactions
- default providers **can be used implicitly**, by using their resources
- Use **explicit declaration to pin** a provider to a specific version, or set required or default attributes
- at initialization Terraform automatically detects and downloads default providers

```
provider "tencentcloud" {
  secret_id  = var.secret_id
  secret_key = var.secret_key
  version = ">=1.58.0"  # will be deprecated
}

provider “hcloud” {
  token = trimspace(file(~/.hetzner-token”))
}

provider “google” {
  version = “~> 2.2.0” # will be deprecated
}

provider “google-beta” {
  version = “~> 2.2.0”  # will be deprecated
}
```

State and backends

- Terraform saves the **state of resources** it manages **in a state file**
- state is used to **manage infrastructure after creation**, it's the moving baseline to **converge existing to desired state**
- state is **configured with the backend primitive**, and backend config variables
- **remote state** (vs default local state) allows concurrent usage and provides **locking**
- **state access is sensitive**, since it contains all resource attributes (e.g. SA keys if created)
- state can be manipulated from the console

```
terraform {
  backend "cos" {
    bucket = "jliao-1253831162"
    prefix = "terraform/state"
  }
}
```

Variables and modules

- variables are essential to
  - **parameterize values** shared between resources (e.g., region, zone, etc.)
  - **provide variable input** to root or dependent modules
  - module outputs can become inputs to other modules
- Terrform allows you to specify values for variables
  - in *TF_VAR_xxx* environment variables
  - in a *terrform.tfvars[.json]* file
  - in **.auto.tfvars[.json]* files
  - with *-var* or *-var-file* options

```
variable "secret_key" {
  description = "Tencent Cloud login key"
  type        = string
  default     = ""
}

provider "tencentcloud" {
  secret_key = var.secret_key
}

# in terrform.tfvars
secret_key = "w4ze4"

# use environment variable
export TENCENTCLOUD_SECRET_KEY="w4ze4”

# from the console command line
terraform apply -var secret_key = "w4ze4"
```

Tencent Cloud

File structure

- **no file hierarchy**, flat namespace of .tf files inside a folder
- files can contain **any mix of primitives** (variables, resources, outputs, locals, etc.)
- **the root module** consists of the .tf files in your working directory when you run *terraform plan* or t*erraform apply*
- **any folder contains Terraform files is a module**, and can be referenced from other modules (not an import, one module “instance” for each declaration)

```
-- environments/
  -- dev/
    -- backend.tf
    -- main.tf
    -- provider.tf
  -- qa/
    -- backend.tf
    -- main.tf
    -- provider.tf
  -- prod/
    -- backend.tf
    -- main.tf
    -- provider.tf
```

Dependency tree

- **Terraform builds a dependency graph** from the Terrafom configurations, and walks this graph to generate plans and refresh state
- interpolation in attributes is resolved at run time and primitives are **connected in a dependency tree**
- **Terraform walks the dependency tree in order** when creating or changing resources
- **explicit dependencies** from the *depends_on* parameter are used to create edges between resources

```
# VPC creation
resource "tencentcloud_vpc" "jliao-vpc" {
  name        = "vpc-jliao-tf"
  ...
  tags = var.resource_tag
}

#Subnet creation
resource "tencentcloud_subnet" "subnet1" {
  availability_zone = var.availability_zone
  name              = "public-subnet"
  vpc_id            = tencentcloud_vpc.jliao-vpc.id
  cidr_block        = "10.2.0.0/24"
   ...

  tags = var.resource_tag
}
```

Tencent Cloud

Dependency tree [modules]

- use **module**.<module_name>.<attribute> to refer to a dependency from one module to another module attribute

Sample code coming soon

Tencent Provider

- **Jointly maintained** by Tencent Cloud TF team and Hashicorp Terraform team
- **Open Source** - Pull Requests welcome

```
terraform {
  required_version = "> 0.14.06"
  required_providers {
    tencentcloud = {
      source  = "tencentcloudstack/tencentcloud"
      version = ">=1.58.0"
    }
  }
}

provider "tencentcloud" {
  secret_id  = var.secret_id
  secret_key = var.secret_key
  region     = var.region
}
```

Tencent Cloud

Terraform Key Commands

terraform refresh	**Refresh the state of the Terraform-managed resources from TC**
terraform init	**Initialize Terraform by downloading modules and providers**
terraform plan	**Refresh state (terraform refresh) and show planned changes**
terraform apply	**Apply planned changes to the live TC environment**
terraform fmt	**Reformat your configuration in the standard style**
terraform state	**Advanced Terraform state management**

Terraform Principles

Small Root Configs

Instead of using one massive root config, separate logical components into separate “deployments”

Use modules instead of config reuse

Don’t reuse the exact same config for different environments, instead use modules for reuse

Parameterize Intelligently

Only parameterize values which actually need to be, emphasis should be on ease of understanding

Use built-in functions

Terraform has lots of built-in functions, know them and use them

Plan first

Always run plan and review the output before running apply

Use terraform fmt

Automatically maintains consistent formatting for you

Terraform tips

- Always try to use backend, such as Tencent COS bucket
- Use plan output file for review and debug
- Set up log directory and log files