Tencent Infrastructure Automation for Terraform Combination with DevOps

Product Documentation

Contents

Combination with DevOps Code Management

Last updated：2023-03-07 10:35:48

Concepts

GitOps

GitOps is a method of continuous delivery proposed by Weaveworks. Its core idea is having a Git version repository that stores declarative infrastructure and applications of application systems. With Git at the center of the delivery workflow, you can submit pull requests and use Git to accelerate and simplify the deployment and Ops of Terraform applications. With such a simple tool like Git, you can focus more on feature creation rather than Ops tasks such as application system installation, configuration, and migration.

Terraform root module

The root configuration (root module) is the working directory running Terraform CLI.

The criteria of a root module:

- **Minimize the number of resources in each root module**

Avoid configuring too many resources in a root directory and storing too many resources in a directory or state. Every time Terraform is run, all the resources in the specified root directories will be refreshed. If a state contains too many resources, the execution may be slow. In general, a state should contain up to 100 resources (up to a dozen of resources at best).

- **Use different directories for different applications**

To manage applications and projects separately, put their resources in their own Terraform directories. A service can be a certain application or general service, for example, a shared network. You should nest all the Terraform code of a certain service in a directory (including sub-directories).

Project Structure

Split the Terraform configuration of the service into two top-level directories: `modules` **directory** containing the actual service configuration; `environments` **directory** containing the root configuration of each environment.

- The `modules` directory contains abstracted and reusable modules.
- The `environments` directory isolates different environments, which allows for using providers for different cloud vendors and configuring different accounts for multi-cloud management (the `prod` directory isolates businesses by `workspace`).

Below is the recommended project structure. For more information, see the code repositories of CODING and GitHub.

```
.
├── README.md
├── environments
│ ├── dev
│ │ ├── main.tf
│ │ └── provider.tf
│ └── prod
│ ├── cicd
│ │ └── main.tf
│ ├── local.tf
│ ├── main.tf
│ ├── provider.tf
│ └── qta
│ └── main.tf
└── modules
├── network
│ ├── main.tf
│ ├── outputs.tf
│ ├── provider.tf
│ └── variables.tf
├── security_group
│ ├── main.tf
│ ├── outputs.tf
│ ├── provider.tf
│ └── variables.tf
└── tke
├── main.tf
├── outputs.tf
├── provider.tf
└── variables.tf
```

Code Reuse

To reuse code, we recommend you use Terraform modules. You can use custom or provided modules, such as Tencent Cloud modules. The `Modules` directory of the project encapsulates module templates. Resources are categorized, and resources of the same category are managed in the same directory. Target resources are managed in a template so as to connect target and dependent resources, for example, the TKE template and its dependent template of `Network` resources.

Building a Secure Terraform Update Process

Secure infrastructure relies on a stable and secure Terraform update process.

Planning before execution

Always generate a plan first for Terraform executions and save it in the output file. Execute the plan after gaining the approval of the infrastructure owner. Even when you design the prototype for a change locally, you should generate a plan and view the resources to be added, modified, and terminated in an application.

Implementing an automated workflow

Execute Terraform via an automated tool to ensure execution context consistency and avoid human errors.

Avoiding importing existing resources

- Avoid importing existing resources (through `terraform import`), as this may make it hard to understand the source and configuration of a manually created resource. Instead, create and delete resources in Terraform.
- In cases where deleting old resources would create significant toil, use the `terraform import` command with explicit approval. Resources imported to Terraform can be managed only by Terraform.

Avoiding modifying the Terraform state

If you use Terraform to manage the infrastructure and manually update resource attributes in the console, the Terraform execution plan may turn out unexpected. To modify the Terraform state information, run the `terraform state` command.

Versioning

Similar to other code forms, infrastructure code can be stored in the versioning system to retain records and allow for easy rollback.

Environment Isolation

The following isolation methods are supported:

Isolation by directory

Group resources by directory and configure root modules in sub-directories for isolation (as adopted in this document).

Note: In `environments`, the `dev` and `prod` directories use different root modules to isolate the infrastructure configurations of different environments.

Isolation by branch

Use different branches in different environments and initialize Terraform in the root module of the corresponding branch.

Note: Similar to isolation by directory, isolation by branch adopts root modules to configure certain environments in different branches and deploys modification by merging changes in different branches.

Isolation by workspace

Create resources based on different `workspace` configurations in the same root module.

Note: In the `prod` directory, `cicd` and `qta` isolate the configurations of different businesses by `workspace`.

Continuous Integration and Deployment Terraform Application in GitHub

Last updated：2023-04-20 14:56:21

This document describes how to implement automated deployment with Terraform and GitHub Actions.

Prerequisites

1. Register at GitHub.
2. Register at Tencent Cloud.
3. Get the credentials. Create and copy `SecretId` and `SecretKey` on the Manage API Key page.

Creating a Project

Create a code repository on GitHub. Below is the directory structure:

```
.
├── README.md
├── environments
│   ├── dev
│   │   ├── main.tf
│   │   └── provider.tf
│   └── prod
│   ├── cicd
│   │   └── main.tf
│   ├── local.tf
│   ├── main.tf
│   ├── provider.tf
│   └── qta
│   └── main.tf
└── modules
├── network
│   ├── main.tf
│   ├── outputs.tf
│   ├── provider.tf
│   └── variables.tf
├── security_group
│   ├── main.tf
```

```
│ ├── outputs.tf
│ ├── provider.tf
│ └── variables.tf
└── tke
├── main.tf
├── outputs.tf
├── provider.tf
└── variables.tf
```

Directory structure description:

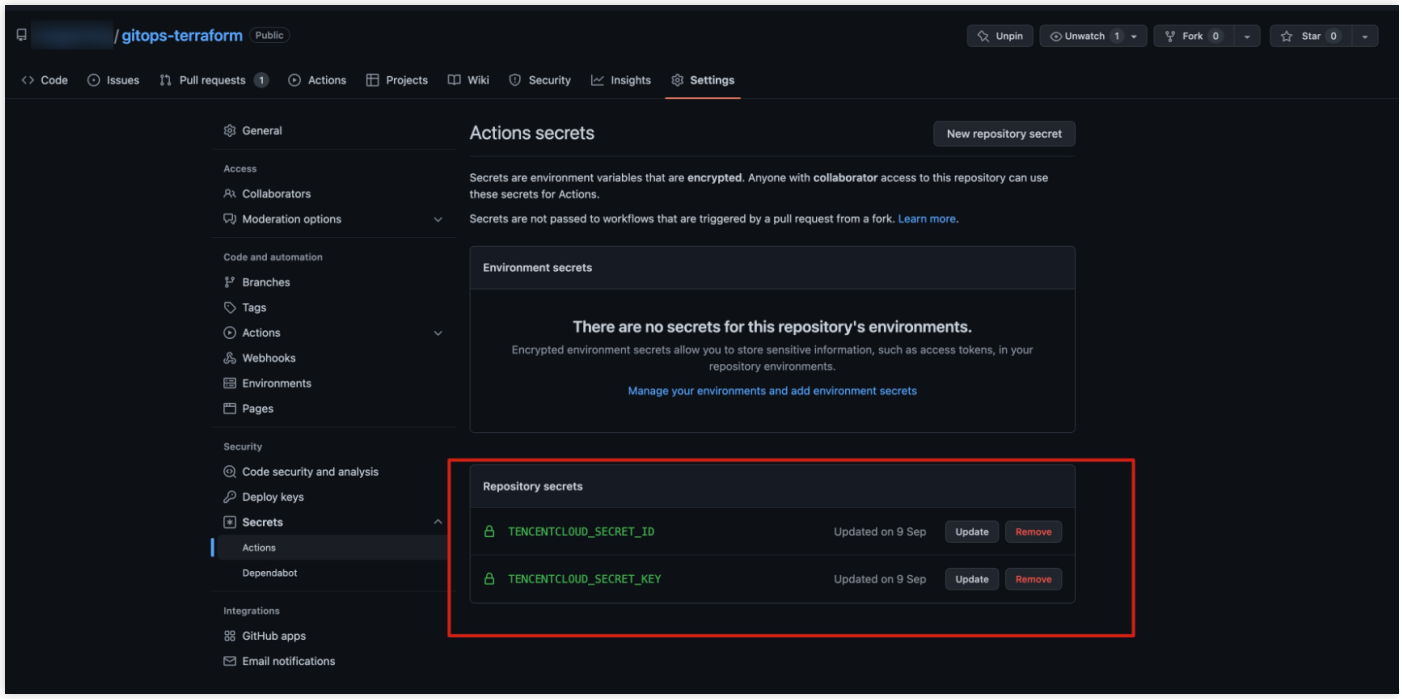
1. The project structure mainly consists of `environments` and `modules` directories.
2. The `environments` directory isolates `dev` and `prod` environments and sets different configurations for different environments. Each environment directory is an independent root module.

   Note：
   - `dev` demonstrates how to create a VPC.
   - `prod` demonstrates how to isolate businesses through `workspace`. VPCs are created in the `cicd` directory, and container clusters are created in the `qta` directory.
3. `modules` encapsulates resource information for reuse. Here, it contains demo modules of the VPC, security group, and TKE.
4. For the complete code, see gitops-terraform.

Workflow Configuration

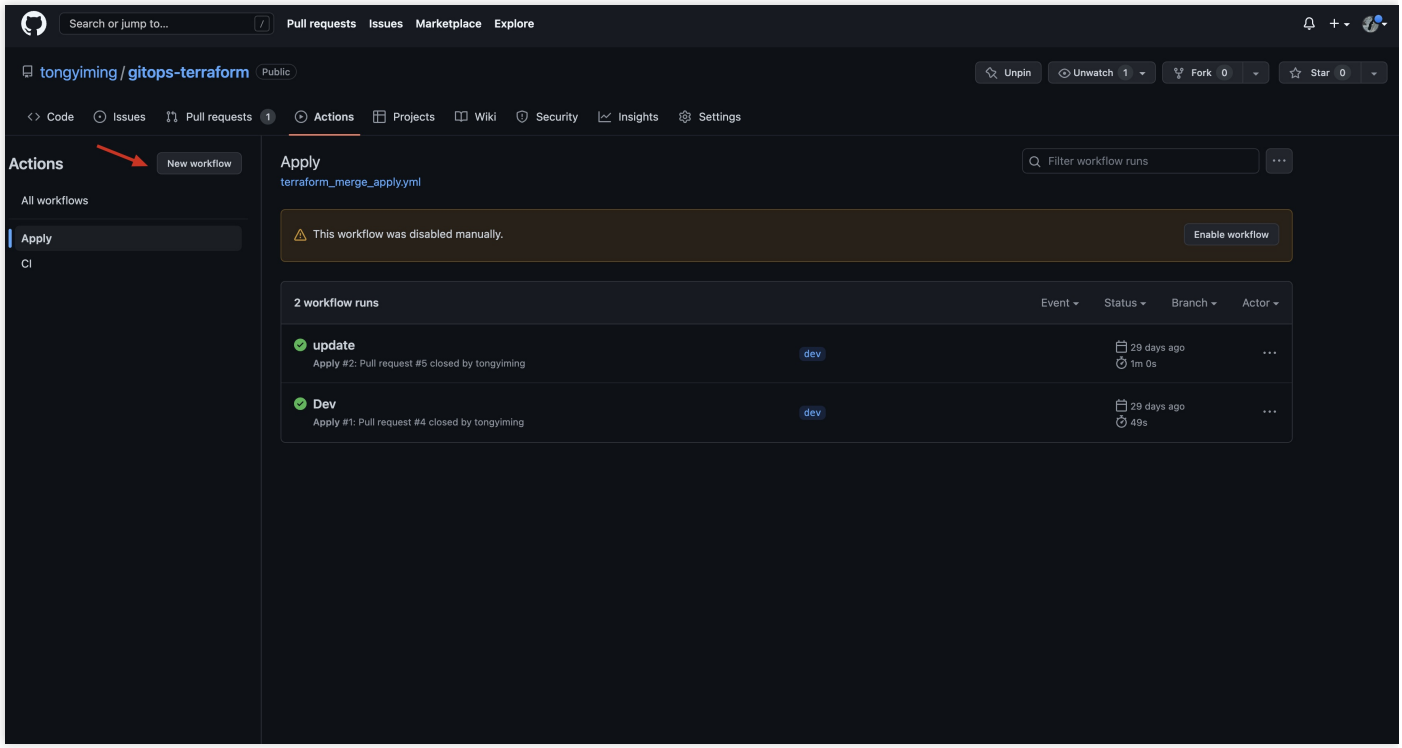
1. To avoid security issues due to AK/SK leakage, you need to set environment variables in `https://github.com/${USER}/${PROJECT}/settings/secrets/actions`. Replace the secrets with

the copied `SecretId` and `SecretKey`.



2. Configure the workflow through GitHub Actions.

Click **New workflow** next to **Actions**, or create a workflow by adding a YAML file in the `.github/workflows/` directory. For more information on the workflow configuration, see Related Operations.



Check workflow

1. Terraform should not have too many root module resources. Similarly, avoid reading all resources during a check, which should be triggered in a fine-grained manner.

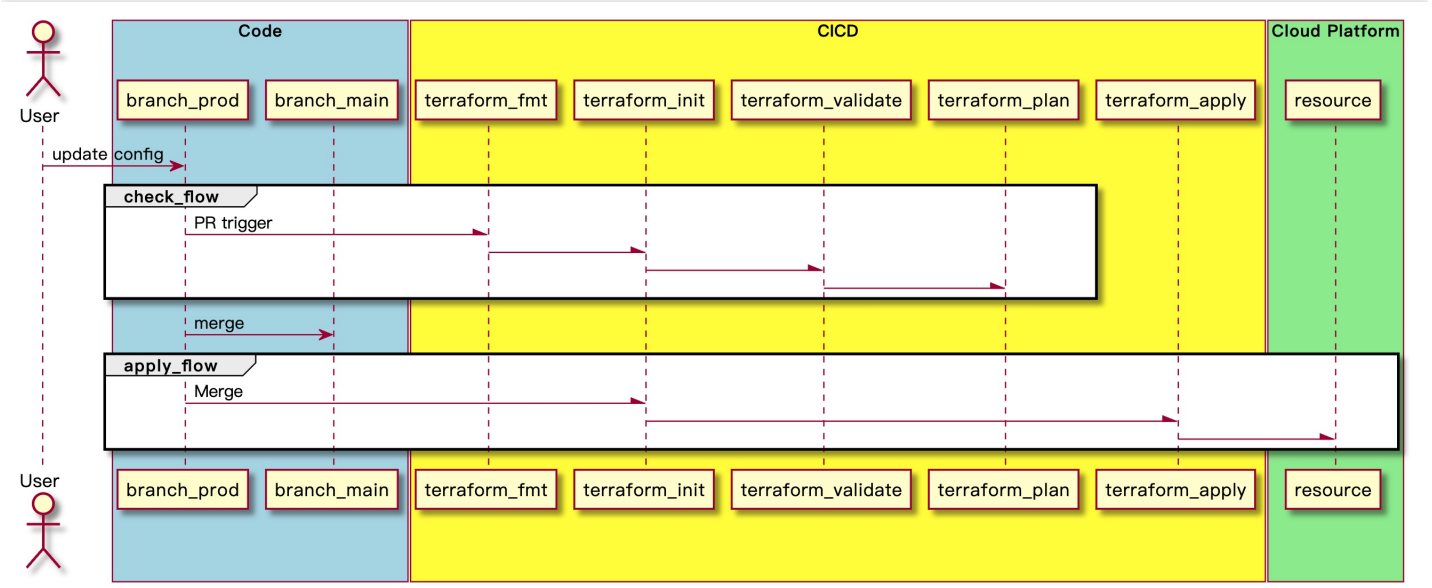
2. In this document, environments are distinguished by branch. For example, if the configuration in the `dev` branch needs to be updated, the update will be triggered only in `dev`. After the configuration update, submit the pull request and merge the code into `main` (main branch). In this way, the system does not need to scan all the sub-directories of `environments` for update checks, reducing unnecessary state sync operations.

3. The workflow mainly runs `terraform fmt`, `terraform init`, `terraform validate`, and `terraform plan` to check the code and display the build plan, so as to determine whether to execute the deployment.

Deployment workflow

1. If all operations in the check workflow are successful and the output of `terraform plan` is as expected, you can perform the merge operation.

2. After the merge is completed, trigger the deployment operation (i.e., `terraform apply`) as shown below:



Related Operations

After the environment directory specified by `environment` is entered, the system will check whether a sub-directory exists, and if so, the system will isolate different business environments (such as `qta` and `ci`) through `workspace`; if not, the specified directory is equivalent to a common root module.

Creating a check workflow

Download Terraform and verify the Terraform code as follows:

```
# This is a basic workflow to help you get started with Actions

name: CI

# Controls when the workflow will run
on:
pull_request:


# A workflow run is made up of one or more jobs that can run sequentially or in p
arallel
jobs:
# This workflow contains a single job called "build"
build:
# The type of runner that the job will run on
runs-on: ubuntu-latest
env:
TENCENTCLOUD_SECRET_KEY: ${{ secrets.TENCENTCLOUD_SECRET_KEY }}
TENCENTCLOUD_SECRET_ID: ${{ secrets.TENCENTCLOUD_SECRET_ID }}

# Steps represent a sequence of tasks that will be executed as part of the job
steps:
- uses: actions/checkout@v3
- uses: hashicorp/setup-terraform@v2
with:
terraform_wrapper: false

- name: check env
run: |
if [ ! -d "environments/$GITHUB_HEAD_REF" ]; then
echo "************************SKIPPING************************************"
echo "Branch '$GITHUB_HEAD_REF' does not represent an oficial environment."
echo "********************************************************************"
exit 1
fi

- name: terraform fmt
id: fmt
```

```
run: terraform fmt -recursive -check

- name: terraform init
id: init
working-directory: environments/${{ github.head_ref }}
run: terraform init

- name: terraform validate
id: validate
working-directory: environments/${{ github.head_ref }}
run: terraform validate

- name: terraform plan
id: plan
if: github.event_name == 'pull_request'
working-directory: environments/${{ github.head_ref }}
run: |
plan_info=""
dir_count=`ls -l | grep "^d" | wc -l`
if [ $dir_count -gt 0 ]; then
for dir in ./*/
do
env=${dir%*/}
env=${env#*/}
echo ""
echo "========> Terraform Plan <========"
echo "At environment: ${{ github.head_ref }}"
echo "At workspace: ${env}"
echo "================================="
terraform workspace select ${env} || terraform workspace new ${env}
plan_info="$plan_info\n$(terraform plan -no-color)"
done
else
plan_info="$(terraform plan -no-color)"
fi
plan_info="${plan_info//'%'/'%25'}"
plan_info="${plan_info//$'\n'/'%0A'}"
plan_info="${plan_info//$'\r'/'%0D'}"
echo "::set-output name=plan_info::$plan_info"
continue-on-error: true

- uses: actions/github-script@v6
if: github.event_name == 'pull_request'
with:
script: |
const output = `#### Terraform Format and Style \`${{ steps.fmt.outcome }}\`
#### Terraform Initialization \`${{ steps.init.outcome }}\`
```

```
#### Terraform Validation \`${{ steps.validate.outcome }}\`
#### Terraform Plan \`${{ steps.plan.outcome }}\`
<details><summary>Show Plan</summary>
\`\`\`\n
${{ steps.plan.outputs.plan_info }}
\`\`\`
</details>
*Pushed by: @${{ github.actor }}, Action: \`${{ github.event_name }}\`*`;
github.rest.issues.createComment({
issue_number: context.issue.number,
owner: context.repo.owner,
repo: context.repo.repo,
body: output
})
```

Creating a deployment workflow

```
name: Apply

on:
pull_request:
types:
- closed
branches:
- main


jobs:
build:
if: github.event.pull_request.merged == true
runs-on: ubuntu-latest
env:
TENCENTCLOUD_SECRET_KEY: ${{ secrets.TENCENTCLOUD_SECRET_KEY }}
TENCENTCLOUD_SECRET_ID: ${{ secrets.TENCENTCLOUD_SECRET_ID }}

steps:
- uses: actions/checkout@v3
- uses: hashicorp/setup-terraform@v2

- name: terraform init
id: init
working-directory: environments/${{ github.head_ref }}
run: terraform init

- name: terraform apply
working-directory: environments/${{ github.head_ref }}
```

```
run: |
dir_count=`ls -l | grep "^d" | wc -l`
if [ $dir_count -gt 0 ]; then
for dir in ./*/
do
env=${dir%*/}
env=${env#*/}
echo ""
echo "========> Terraform Apply <========"
echo "At environment: ${{ github.head_ref }}"
echo "At workspace: ${env}"
echo "================================="

terraform workspace select ${env} || terraform workspace new ${env}
terraform apply -auto-approve
done
else
terraform apply -auto-approve
fi
```