Using Terraform With Substrate

Document Level Classification

200

Introduction

This page covers how to write and run Terraform to manage resources in your Substrate account.

This page is intended to show you how to manually run terraform in your local development environment on your workstation. If you are running automated workloads or integrating with CI/CD, Terraform Enterprise is the **preferred method**. Reference the https://confluence-epicgames.atlassian.net/wiki/spaces/CE/pages/93488378.

We are in the process to migrate TFE Workload to **HCP Terraform**. For additional details please read the Data Migration Tracking page to get information about the advance and the HCP User guide to understand how to use the new SaaS platform.

If you are using Old Prod you may be more interested in using Atlantis. Atlantis is not compatible with Substrate.

What is Terraform?

Terraform is a command-line tool that manages cloud resources using a declarative configuration syntax. It works with AWS, Google Cloud, Azure, SignalFX, and many other providers.

Terraform is able to perform a diff between what you ask for and what is currently provisioned, and make appropriate, minimal changes to converge your resources into the state you want. Terraform tracks and modifies managed resources to account for drift and updates, identifies new resources and resources you want to discard, and shows you its plan before it takes action.

When you first start with AWS and other providers it's often easier to start in the web console. However, Terraform makes it possible to version your configuration in GitHub, undo and redo changes, deploy the same infrastructure to multiple environments, and more.

You can download Terraform and read the docs at terraform.io. We recommend starting with the most recent version of Terraform when you start a new project, since this is likely to support the latest features and bug fixes for AWS.

Running Terraform

Running Terraform from your laptop or workstation requires authentication to the AWS API. You can authenticate using `https://confluence-epicgames.atlassian.net/wiki/spaces/CE/pages/93487913`. For example:

```
# Note: Make sure aop is updated to the latest version.
#       Reference https://confluence-epicgames.atlassian.net/wiki/displ
# Authorize AWS SSO (typically run only the first time you set up AWS S
# Update local cache with AWS accounts and roles you have permission to
# There may be up to 30 minutes delay between access being approved and
# This command is an 'as needed' command to refresh the aws profiles av
aop aws-sso generate

# List available AWS accounts and roles
aws configure list-profiles

# Authenticate with AWS and retrieve credentials
aws sso login

# Set the AWS_PROFILE environment variable using the desired profile
export AWS_PROFILE=<AWS_PROFILE_NAME>

# Verify
AWS_PAGER='' aws sts get-caller-identity

# Run terraform plan
terraform plan
```

After writing your Terraform configuration, preview the changes Terraform will make by running `terraform plan`. When you verify that the changes are what you expect, run `terraform apply` to modify your AWS resources.

If you lose your local state file it will be difficult or impossible to restore the information it contains, and you will have to manually clean up or reconcile any resources you have created using Terraform. To protect against this, make sure to configure Remote State (below).

Remote State

Terraform keeps track of the resources it has created in the state file (typically called `terraform.tfstate`) on your computer. If you want to collaborate with your team, and to prevent losing the statefile if your laptop or harddrive has an accident, you'll want to use remote state.

Configuring Terraform

Configure your remote state by using the "s3" backend.

```
terraform {
  backend "s3" {
    bucket         = "188014929444-tfstate"
    region         = "us-east-1"
    key            = "vault-healthcheck/terraform.tfstate"
    dynamodb_table = "188014929444-tfstate"
  }
  ...
}
```

After configuring remote state, or if you just downloaded an existing project from GitHub, you will need to run `terraform init`.

```
# List available AWS accounts and roles
aws configure list-profiles

# Authenticate with AWS and retrieve credentials
aws sso login
```

```
# Set the AWS_PROFILE environment variable using the desired profile
export AWS_PROFILE=<AWS_PROFILE_NAME>

# Verify
AWS_PAGER='' aws sts get-caller-identity

# Run terraform init
terraform init
```

Preconfigured State Buckets

Each Substrate Account has an S3 bucket and Dynamo table provisioned in advance for Terraform state. They follow the pattern `<account-id>-tfstate`. See above for example.

You can find the account ID for your account using the AWS command line:

```
# List available AWS accounts and roles
aws configure list-profiles

# Authenticate with AWS and retrieve credentials
aws sso login

# Set the AWS_PROFILE environment variable using the desired profile
export AWS_PROFILE=<AWS_PROFILE_NAME>

# Verify
AWS_PAGER='' aws sts get-caller-identity

$ aws sts get-caller-identity
{
   "UserId": "...",
   "Account": "168314445178", ← current AWS account ID
   "Arn": "arn:aws:sts::168314445178:assumed-role/..."
}
```

Or by looking at `~/.aws/config`:

```
less ~/.aws/config
[default]
sso_start_url = https://epicgames.awsapps.com/start
sso_region = us-east-1

[profile fadd-dev-read-only] <-------------------------
sso_start_url = https://epicgames.awsapps.com/start
sso_region = us-east-1
sso_account_id = 384087105544 <-------------------------
sso_role_name = read-only
...
...
...
```

You can also find it via the AWS Console:

```
aop aws-sso browser
```

Recommended Workflow

There are a lot of different ways to use Terraform. In our experience, some of those approaches are easier than others. Here are some tips from Infra Platform's experiences with different workflows.

- **Do not** use the `terraform workspace` command. **Do** create different modules for different environments.
- **Do not** ask the user to specify `-var` or `-var-file` for plan and apply. **Do** create a different module with all the inputs it needs baked-in.
- **Strongly prefer** one Terraform module ↔ one AWS account. This is much easier to use with `aop`.
- **Do not** store secrets in Terraform state or input variables (these are stored in plain text). **Do** use the Vault provider for Terraform.

- **Do** add a `.aop` file to your project. See the aop docs for info and below for an example.

Here is an example of a multi-environment Terraform project that shares most of the config via the `common` module. The parts that differ between environments, such as the VPC ranges and DNS records are specified in environment-specific modules. The common pieces included the instances, load balancer, and input vars are defined in the `common` module.

This pattern allows you to share code but also gives each environment a very simple `aop sso login`, `terraform plan`, `terraform apply` workflow – no additional arguments, vars, or workspaces need to be specified.

```
├── README.md
└── terraform
    ├── common
    │   ├── instance.tf
    │   ├── loadbalancer.tf
    │   └── variables.tf
    ├── dev
    │   ├── .aop
    │   ├── dns.tf
    │   ├── main.tf
    │   ├── README.md
    │   └── vpc.tf
    ├── live
    │   ├── .aop
    │   ├── dns.tf
    │   ├── main.tf
    │   ├── README.md
    │   └── vpc.tf
    └── README.md
```

Handling Secrets

IAM user credentials and other secrets that are created by Terraform will end up in the state file, which is stored in S3. This is *not secure* and you should not create these types of credentials with Terraform. Instead, create the IAM user in Terraform and use `aws iam create-access-key` via the AWS CLI to create the credentials.

For other types of secrets like database passwords, create these in advance and store them in Vault. Then Terraform can reference them via the Vault Provider. See the https://confluence-epicgames.atlassian.net/wiki/spaces/CE/pages/93487474 documentation for more info about accessing Vault.

Note that you can mark variables as sensitive so they are not displayed in Terraform plan or output, but this *does not* protect them from appearing in the statefile in plain text. Use Vault for secrets.

Troubleshooting

No Credentials

If you encounter an error like this one:

```
Error: No valid credential sources found for AWS Provider.
    Please see https://terraform.io/docs/providers/aws/index.html for m
    providing credentials for the AWS Provider
```

This error indicates one of two things: Either you have not properly authenticated with aop, or Terraform doesn't know which profile to use. Credentials from `aop` expire after 1 hour, so you may need to refresh them (run `aop auth` to refresh your credentials). If you are certain you have valid credentials. Make sure `AWS_PROFILE` is set in your current shell so Terraform knows which profile to use. If you used `aop shell` this should be set for you.

Terraform may also be configured to use a particular profile. If the Terraform configs specify a particular profile, make sure you authenticate with that one using `aop`. Some Terraform configs may require access you don't have. In that case you may ask the Infra Platform team for help in #cloud-ops-support-ext on Slack.

Locked Environment

Currently, AWS credentials from `aop` expire every hour. Terraform locks the environment at the start of the run, and unlocks it when it completes. However, If your credentials expire during a Terraform run the run will fail and Terraform will get stuck. You'll see something like this:

```
Error releasing the state lock!

Error message: failed to retrieve lock info: ExpiredTokenException: The
```

And on subsequent attempts to run `terraform plan` or `terraform apply` you'll see:

```
Error: Error locking state: Error acquiring the state lock: Conditional
        status code: 400, request id: 2283NDOMBGVE9STBRHBQUSLFO7VV4KQNS
Lock Info:
  ID:        41e98cf6-ded7-87ab-9dbe-bab4c3bd1a12
  Path:      on-net-state/env:/use1a-egstore-backend/terraform/team-res
  Operation: OperationTypePlan
  Who:       cbednarski@stormworker.local
  Version:   0.12.9
  Created:   2019-10-15 22:04:11.501389 +0000 UTC
  Info:
```

You can resolve this by running`:`

```
$ terraform force-unlock <lock-id>
```

WARNING: Do not use this command if someone else is currently deploying to the same environment, or someone's changes to the state file will be lost and you will have orphaned resources in AWS.

Page Information:
Page ID: 81068270
Space: Cloud Developer Platform
Downloaded: 2025-07-12 04:07:16