F5 HashiCorp Workshop Documentation

F5 Networks, Inc.

2020 年 03 月 06 日

F5 2020 Read The Docs Guide

F5 HashiCorp Workshop

F5 Networks, Inc.

Contents:

1 初めての Terraform

1.1 Goal:

ここでは OSS 版の Terraform を利用して AWS 上に一つインスタンスを作り、それぞれのコンポーネントや用語について説明をします。

1.2 Prerequisites:

- インターネットに接続可能な PC
- 管理者権限（Terraform をインストールするため）

1.3 Terraform Installation:

1. Terraform がインストールされていない場合は [こちら](https://www.terraform.io/downloads.html) よりダウンロードをしてください。
2. ダウンロードしたら unzip して実行権限を付与し、パスを通します。下記は macOS の手順です。

```
$ unzip terraform*.zip
$ chmod + x terraform
$ mv terraform /usr/local/bin
$ terraform -version
Terraform v0.12.21
```

注釈: Windows の場合はこちらのリンクをご参照ください。 https://dev.classmethod.jp/tool/try-terraform-on-windows/

---

3. 次に任意の作業用ディレクトリを作ります。

```
$ mkdir -p tf-workspace/hello-tf
$ cd  tf-workspace/hello-tf
```

1.4 Terraform Configuration Files:

1. 早速このフォルダに Terraform のコンフィグファイルを作ってみます。コンフィグファイルは 'HashiCorp Configuration Language'というフレームワークを使って記述していきます。

   'main.tf'と 'vaiables.tf'という二つのファイルを作ってみます。'main.tf'はその名の通り Terraform のメインのファイルで、このファイルに記述されている内容が Terraform で実行されます。'variables.tf'は変数を定義するファイルです。各変数にはデフォルト値や型などを指定できます。

```
$ cat <<EOF > main.tf
terraform {
    required_version = "~> 0.12"
}

provider "aws" {
    access_key = var.access_key
    secret_key = var.secret_key
    token      = var.session_token
    region     = var.region
}

resource "aws_instance" "hello-tf-instance" {
ami = var.ami
count = var.hello_tf_instance_count
instance_type = var.hello_tf_instance_type

tags = {
    Name = var.instance_name
}
}

EOF
```

2. 次に 'variables.tf'ファイルを作ります。

```
$ cat << EOF > variables.tf
variable "access_key" {}
variable "secret_key" {}
variable "session_token" {}
variable "region" {}
variable "ami" {}
```

(次のページに続く)

(前のページからの続き)

```
variable "instance_name" {}
variable "hello_tf_instance_count" {
    default = 1
}
variable "hello_tf_instance_type" {
    default = "t2.micro"
}
EOF
```

1.5 Terraform Commands:

1. 二つのファイルができたらそのディレクトリ上で Terraform の初期化処理を行います。‘init‘処理ではステートファイルの保存先などのバックエンドの設定や必要ばプラグインのインストールを実施します。

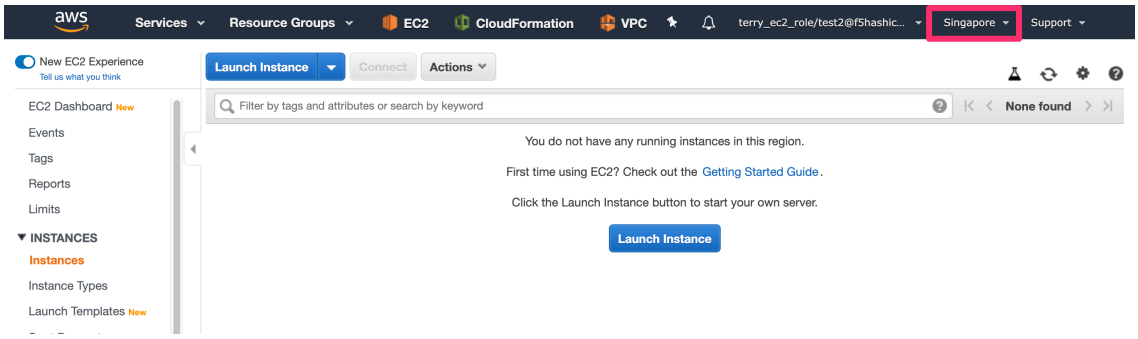
```
$ terraform init
```

2. ここでは AWS のプラグインがインストールされるはずです。

```
$  ls -R .terraform/plugins
.terraform/plugins:
linux_amd64

.terraform/plugins/linux_amd64:
lock.json                          terraform-provider-aws_v2.51.0_x4
```

3. 次に ‘plan‘と ‘apply‘を実施してインスタンスを作ってみましょう。AWS のコンソールまたは aws cli でインスタンスの状況確認しておいてください。

```
$ aws ec2 describe-instances --query "Reservations[*].Instances[*].
↪{InstanceId:InstanceId,State:State,Name:Tags[?Key=='Name']|[0].Value}"
[]
```



4. ‘plan‘は Terraform によるプロビジョニングの実行プランを計画します。実際の環境やステートファイルとの差分を検出し、どのリソースにどのような変更を行うかを確認することができます。‘apply‘はプランに基

づいたプロビジョニングの実施をするためのコマンドです。

また、実行前に変数に値をセットする必要があります。方法としては

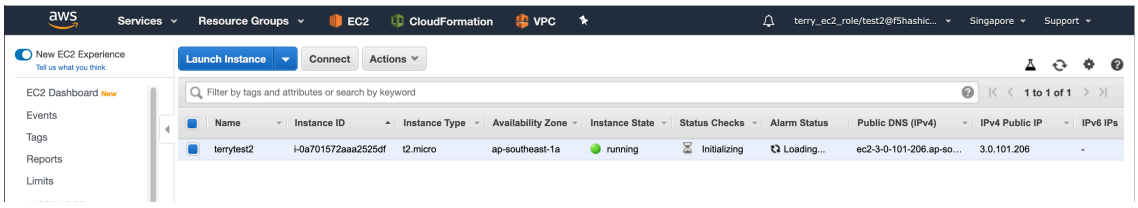
- ‘tfvars‘というファイルの中で定義する
- ‘terraform apply -vars=***‘という形で CLI の引数で定義する
- ‘TF_VAR_***‘という環境変数で定義する
- Plan 中に対話式で入力して定義する

がありますが、今回は環境変数でセットします。

```
$ export TF_VAR_access_key=************
$ export TF_VAR_secret_key=************
$ export TF_VAR_session_token=*********
$ export TF_VAR_instance_name=<enteryourname>
$ export TF_VAR_region=ap-southeast-1
$ export TF_VAR_ami=ami-07ce5f60a39f1790e
$ terraform plan
$ terraform apply
```

5. Apply が終了すると AWS のインスタンスが一つ作られていることがわかるでしょう。AWS のコンソールまたは aws cli でインスタンスの状況確認してください。

```
$ aws ec2 describe-instances --query "Reservations[*].Instances[*].
↪{InstanceId:InstanceId,State:State,Name:Tags[?Key=='Name']|[0].Value}"
[
    {
        "InstanceId": "i-00918d5c9466da418",
        "State": {
            "Code": 48,
            "Name": "running"
        },
        "Name": "xxx"
    }
]
```



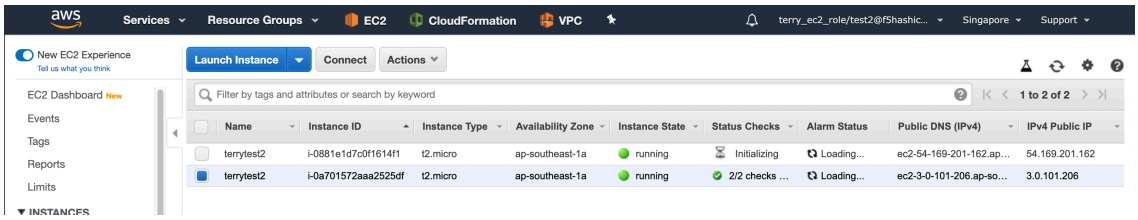
1.6 Terraform Modification:

1. 次にインスタンスの数を増やしてみます。‘hello_tf_instance_count‘の値を上書きして再度実行します。

```
$ export TF_VAR_hello_tf_instance_count=2
$ terraform plan
$ terraform apply -auto-approve
```

注釈: ちなみに今回は ‘-auto-approve‘というパラメータを使って途中の実行確認を省略しています。AWS のインスタンスが二つに増えています。Terraform は環境に差分が生じた際は Plan で差分を検出し、差分のみ実施するため既存のリソースには何の影響も及ぼしません。

```
$ aws ec2 describe-instances --query "Reservations[*].Instances[*].
↪{InstanceId:InstanceId,State:State,Name:Tags[?Key=='Name']|[0].Value}"
[
    {
        "InstanceId": "i-00918d5c9466da418",
        "State": {
            "Code": 48,
            "Name": "running"
        },
        "Name": "xxx"
    },
    {
        "InstanceId": "i-0b0aea4b4ab27ef4b",
        "State": {
            "Code": 16,
            "Name": "running"
        },
        "Name": "xxx"
    }
]
```



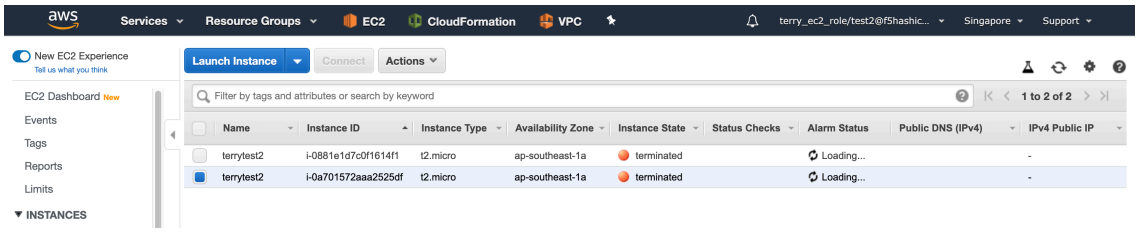
1.7 Destroy Environment:

1. 次に ‘destroy‘で環境をリセットします。

```
$ terraform destroy
```

2. 実行ししばらくすると EC2 インスタンスが ‘terminated‘の状態になってることがわかるはずです。

```
$ aws ec2 describe-instances --query "Reservations[*].Instances[*].
↪{InstanceId:InstanceId,State:State,Name:Tags[?Key=='Name']|[0].Value}"
[
    {
        "InstanceId": "i-00918d5c9466da418",
        "State": {
            "Code": 48,
            "Name": "terminated"
        },
        "Name": "xxx"
    },
    {
        "InstanceId": "i-0b0aea4b4ab27ef4b",
        "State": {
            "Code": 16,
            "Name": "terminated"
        },
        "Name": "xxx"
    }
]
```



1.8 Enterprise 版の価値:

Apply が実行されると ‘terraform.tfstate‘というファイルが生成されます。このファイルは現在のインフラの状態を Json 形式で保持しているものですが、次の Plan のタイミングの差分の検出などで扱われ非常に重要です。例えばチームで作業をする際などはこのステートの共有方法をどうやって運用するかなどの考慮が必要になります。

また、このファイルには各リソースの ID のみならずデータベースや AWS 環境のシークレットなど様々な機密性の高いデータが含まれておりステートファイルをセキュアに保つことも運用上重要です。

以降の章ではステートファイルのみならず、OSS 版では Terraform を安全に利用するために考慮する必要がある様々な運用上の課題に対して Enterprise 版がどのような機能を提供しているかを一つずつ試してみます。

参考リンク

- State
- Backends
- init
- plan
- apply
- AWS Provider

2

Terraform Cloud による Remote state 管理

さて、最初の Workshop では Local 環境で Terraform を実行し、State ファイルも Local 環境に作成されました。

State ファイルは非常に重要なファイルで様々な情報がつまっています。

- Provisioning された Resource の識別情報
- API キーやパスワードなどの Secret
- など

Terraform で継続的に Provisioning を行なうためには State ファイルの管理が必至です。Terraform はデフォルトの挙動として、実行されて得た State ファイルを Local 環境に保存します。ただ、Local 環境で State ファイルを管理するにはいくつかの問題があります。

- 個人の Local 環境だけに存在すると、チームでの作業が出来ない- 例えば A さんのローカルマシン上にだけ State ファイルがある場合、A さん以外の人はその環境にたいして、それ以上の Provisioning が出来ない。
- 誤って削除してしまうと元に戻せない（よって全てのインフラ情報が損失してしまう）- 既存の環境を State ファイルに取り込む import というコマンドもありますが、非常に手間と時間がかかります。
- State ファイルは常に最後の Terraform 実行の情報だけが記載されるので、過去のインフラ状態のトラッキングが出来ない- トラッキングのために、Terraform の実行毎に State ファイルを共有スペース（ファイルサーバーや S3 など）や VCS などに保存するやり方もありますが、手間がかかります。

そこで、Terraform OSS のユーザーはこれらの問題を回避するために様々な仕組みをカスタムしてきました。ただ、これらのカスタマイズは各ユーザー側の開発・メンテナンスなどを必要とし、その管理のために本来の仕事とは別の時間を費やしてしまいます。

Terraform Cloud 及び Terraform Enterprise にはこれらの問題を解決すべく、様々な Team collaboration 及び Governance の機能を予め用意してあります。これからの Workshop では、これら機能を紹介していきます。

2.1 事前準備

1. この Workshop を行なうには Terraform Cloud のアカウントが必要です。こちらからサインアップをしてください。（すでにアカウントをお持ちの方はスキップしてください。）

<https://app.terraform.io/signup/account>

2.2 Remote State 管理機能

Terraform cloud には Remote State 管理機能があります。ちなみに、**この機能は誰でも無料で利用できます**。

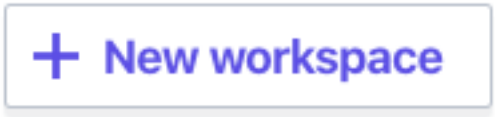
ここでは、Remote State 管理機能を使うエクササイズを行います。

2.3 Workspace の設定

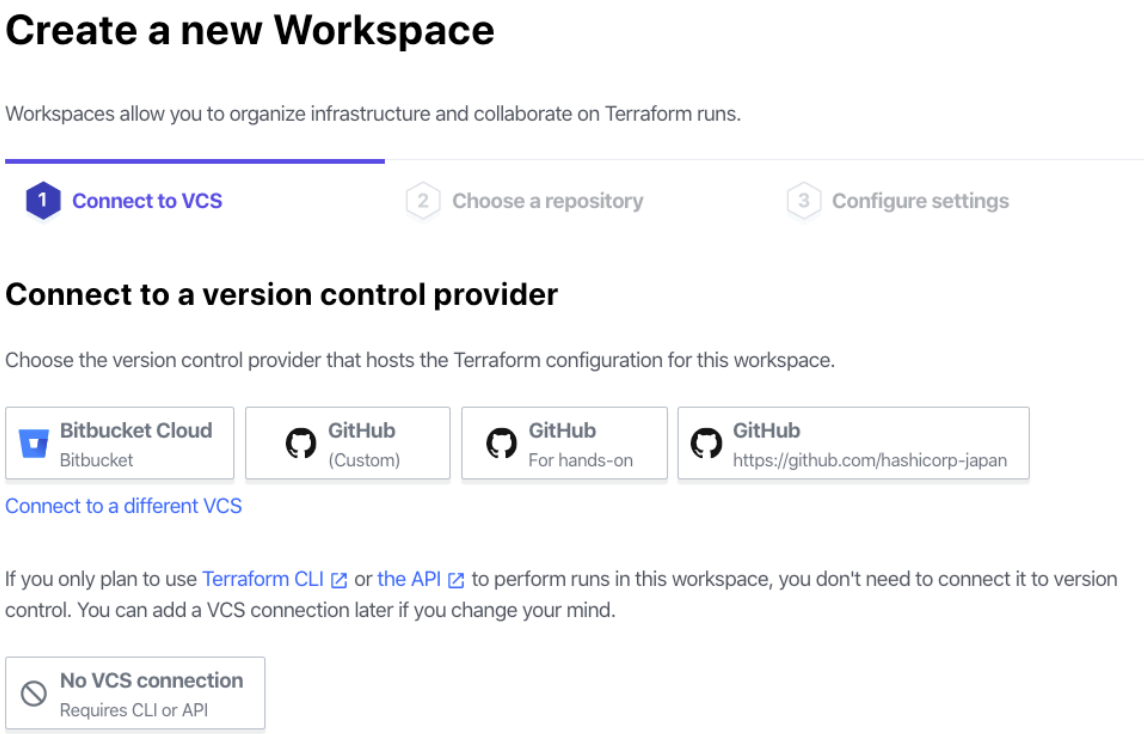
1. Terraform Cloud にログインし、新規 Workspace を作成します。ワークスペース名は任意で構いません。

   注釈: 1 つの Organization 内では全ての Workspace 名が一意である必要がありますので、複数のユーザーで作業する場合、Workspace 名がユニークになるようにしてください。

   Workspace は以下のボタンより作成できます。

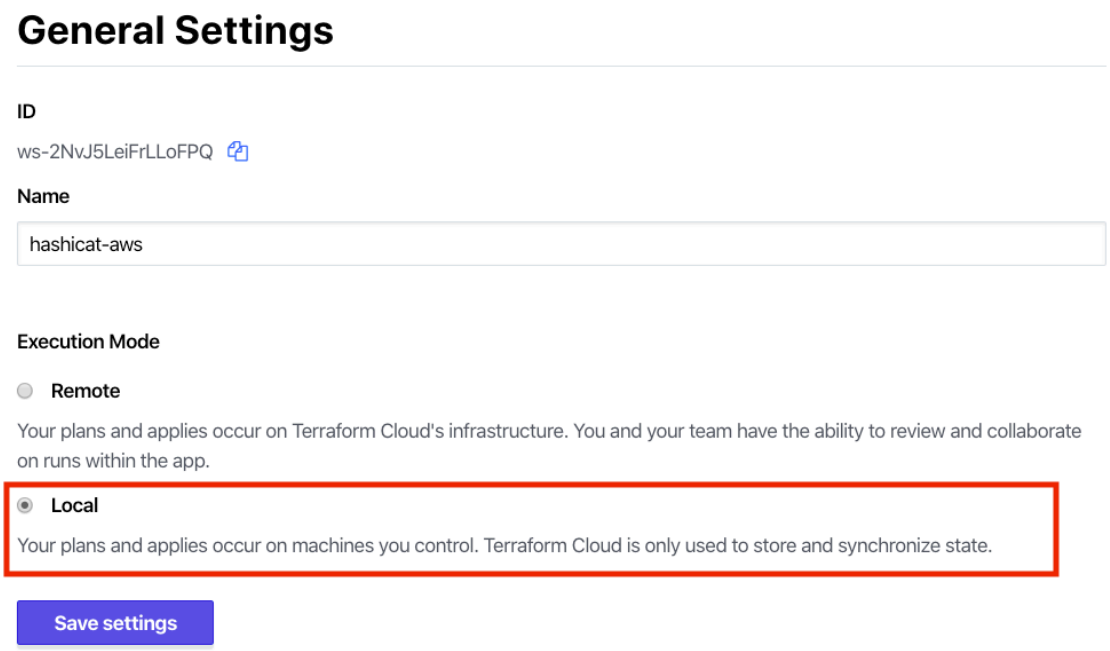
   

2. 以下の画面で、**No VCS Connection** を選択してください。



Workspace 名にには重複しない任意の名前をつけてください。以下、このページでは、ここで指定した名前を **YOURWORKSPACE** という置き換え表示で表します。

3. つぎに、Workspace の **Setting** > **General** > にナビゲートし、Execution mode を **Local** に設定して保存してください。

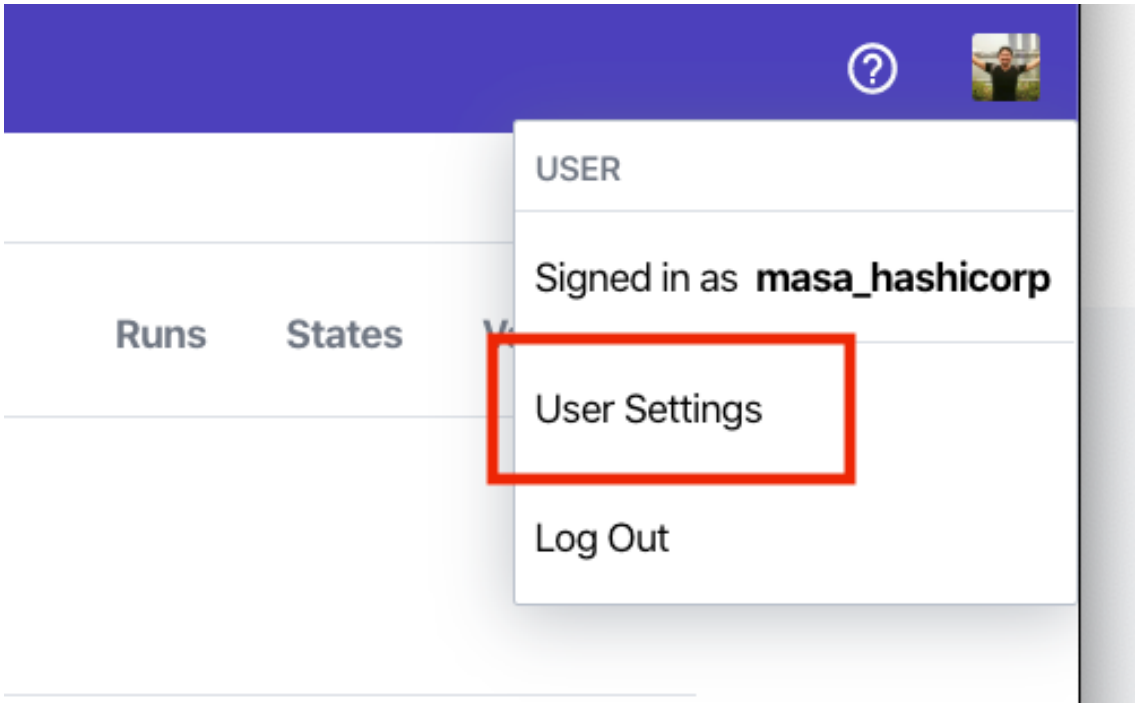


Execution mode を **Local** に設定すると、Terraform の実行は Local 環境で行いますが、作成される State ファイルは Terraform Cloud に保存されます。

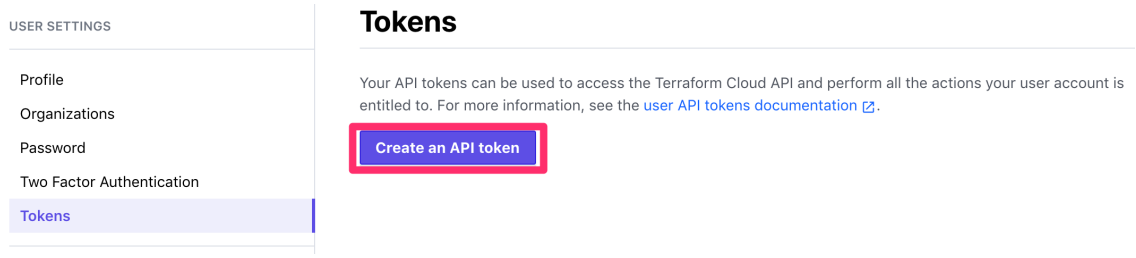
2.4 User Token の作成

さて、次に Local の Terraform 環境から Terraform Cloud にアクセスするために、User token を作成します。この User token はローカル環境や別のシステム（CI/CD パイプラインや外部ツールなど）から Terraform Cloud API を叩く際に必要となります。

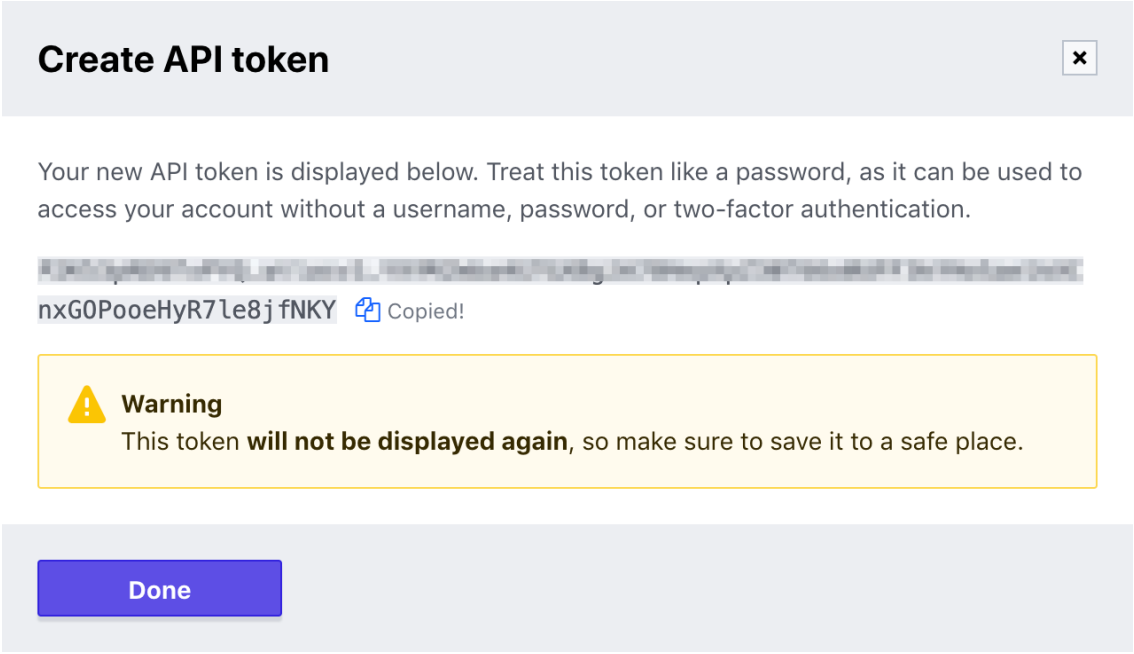
1. 右上の自分のアイコンをクリックして **User Settings** を選択します。



2. そこから、**Tokens** メニューから **Create an API Token** ボタンで User Token を作成します。Description にはこの Token についての?明を追加できます。



3. 作成された Token はこの画面でしか表示されないので、必ずコピーもしくは Download しておいてください。



4. 次に、ここで作成された Token を Local 環境の `~/.terraformrc` に書き込みます。

注釈: Windows の場合、%APPDATA%terraform.rc となります。

```
root@workstation:~# cat ~/.terraformrc
credentials "app.terraform.io" {
    token = "TdobpJ0do60AZw.atlasv1.
↪LK7nXDhzqJNy7zqIkwm0WaMPPuz4vEL5RU7aDTZ1vQQf16vjfEwyOrzDdw4KQejeGnM"
}
```

これで Local 環境から Terraform Cloud の API にアクセスする準備が整いました。

2.5 Remote Backend の設定

1. つぎに Terraform に Remote Backend を使用するコードを追加します。以下のコードを `remote_backend.tf` という名前で作成してください。 **YOURORGANIZATION** は使用している Organization の値に、 **YOURWORKSPACE** は使用している Workspace に置き換えてください。

```
terraform {
backend "remote" {
    hostname = "app.terraform.io"
    organization = "YOURORGANIZATION"
    workspaces {
    name = "YOURWORKSPACE"
```

(次のページに続く)

(前のページからの続き)

```
    }
}
}
```

2. ここまでの準備が出来ましたら、Terraform を実行します。以下のコマンドを実行してください。

```
terraform init
```

3. ここで、もし直前の Workshop で作成された State ファイルが存在していると以下のように、「既存 State ファイルを Remote Backend にコピーするか？」と尋ねられます。 **Yes** と入力して下さい。

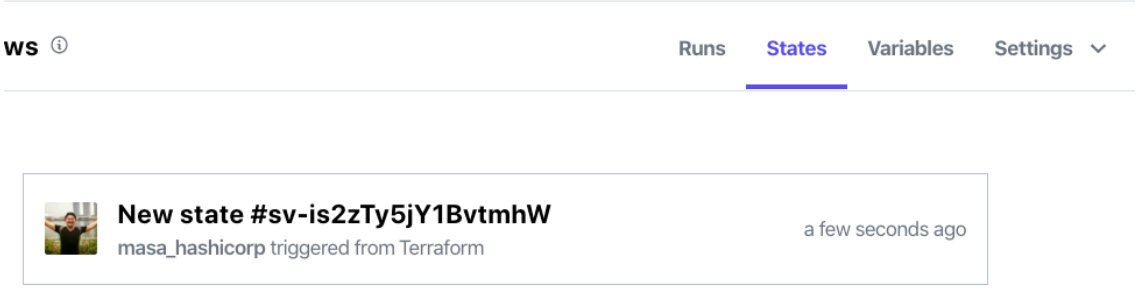
```
root@workstation:~/hashicat-aws# terraform init

Initializing the backend...
Do you want to copy existing state to the new backend?
Pre-existing state was found while migrating the previous "local" backend to␣
↪the
newly configured "remote" backend. No existing state was found in the newly
configured "remote" backend. Do you want to copy this state to the new "remote
↪"
backend? Enter "yes" to copy and "no" to start with an empty state.

Enter a value: yes


Successfully configured the backend "remote"! Terraform will automatically
use this backend unless the backend configuration changes.
```

4. それでは `apply` してみましょう。

```
terraform apply
```

この `apply` では Local の State ファイルではなく、Terraform cloud 上の State ファイルを使用します。よって、もう Local の State ファイルは必要ないので削除しても構いません。

5. この段階で、Terraform cloud の Workspace を確認すると、State ファイルが作成されているはずです。



2.6 まとめ

これで Remote Backend の設定は完了です。ここでのエクササイズでは、個人個人で Workspace を作りましたが、これをチームで共有することで State ファイルの共有が実現できます。

ただ、State ファイルの共有が実現できたとしてもまだまだチーム利用としては足りない機能が多々あります。それらを次からの Workshop で見ていきたいと思います。

<https://github.com/hashicorp-japan/terraform-workshop/tree/master/contents>

F5 Networks, Inc. | f5.com

US Headquarters: 401 Elliott Ave W, Seattle, WA 98119 | 888-882-4447 // Americas: info@f5.com // Asia-Pacific: apacinfo@f5.com // Europe/Middle East/Africa: emeainfo@f5.com // Japan: f5j-info@f5.com