【Terraform入門コース】

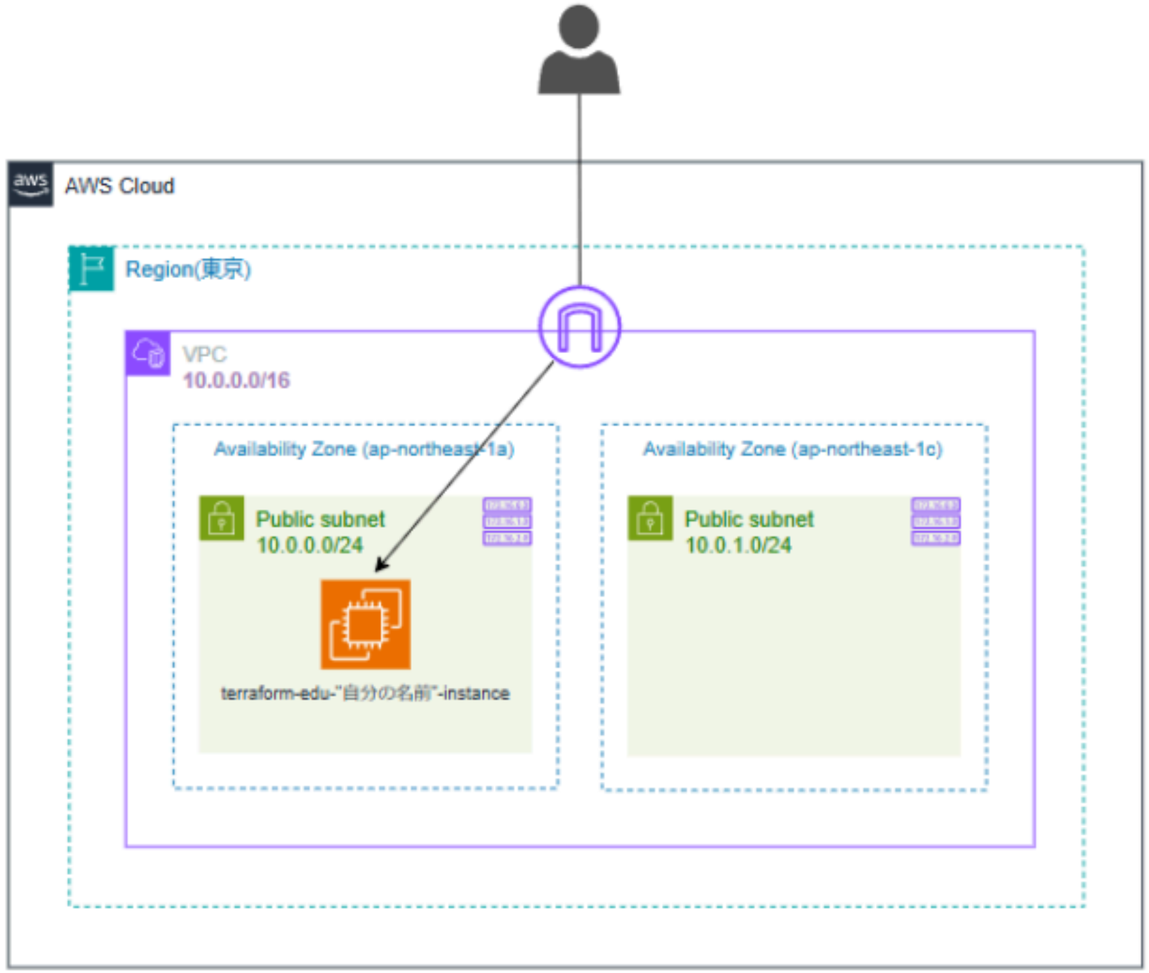
基本操作解説・実践

by @kitasan_chi

本研修の対象者と目標

- **実施時間**
  Terraform 入門研修 (1時間 ~ 2時間) ※内休憩10分
- **対象者**
  インフラ構築の最低限の基礎知識がある方（VPC,EC2等）
  IaC に興味があり、Terraform を初めて触る方
- **目標**
  Terraform の基本的な概念を理解する
  Terraform を使ってAWS上に簡単なリソースを
  作成・変更・削除できるようになる

今回作成する構成図



よくある構成

★VPC - 10.0.0/16

★パブリックサブネット×2

1a - 10.0.0/24 , 1c - 10.0.1/24

★IGW,RT,SG

★EC2

インスタンスタイプ：t3.micro

EBS：4GB

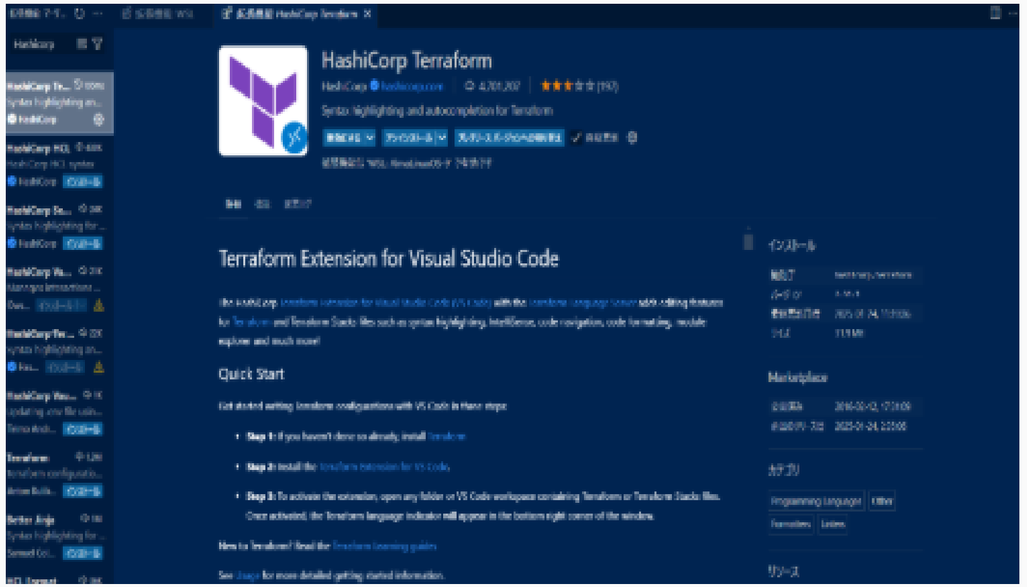
タイムテーブル

時間	内容	資料リンク
15分くらい	IaCとTerraformの概要 (座学)	https://www.canva.com/design/DAGjRbyjsfg/af5cwpHJQReqzM1zir8viQ/edit?utm_content=DAGjRbyjsfg&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton
それ以降の時間(21時まで)	基本操作解説・実践 (座学+ハンズオン)	https://www.canva.com/design/DAGjSNDhW90/JT9x1Lg4xGTrjm_lG6wbew/edit?utm_content=DAGjSNDhW90&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

前提条件

1. VSCode が入っていること
2. 拡張機能の「Remote - SSH」と「HashiCorp Terraform」が入っていること

※導入済みの方は飛ばしてOK



導入資料はこちら

事前準備

- 以下からSSHキーをダウンロードをお願いします（社内用）
  https://drive.google.com/drive/u/0/folders/hogehoge

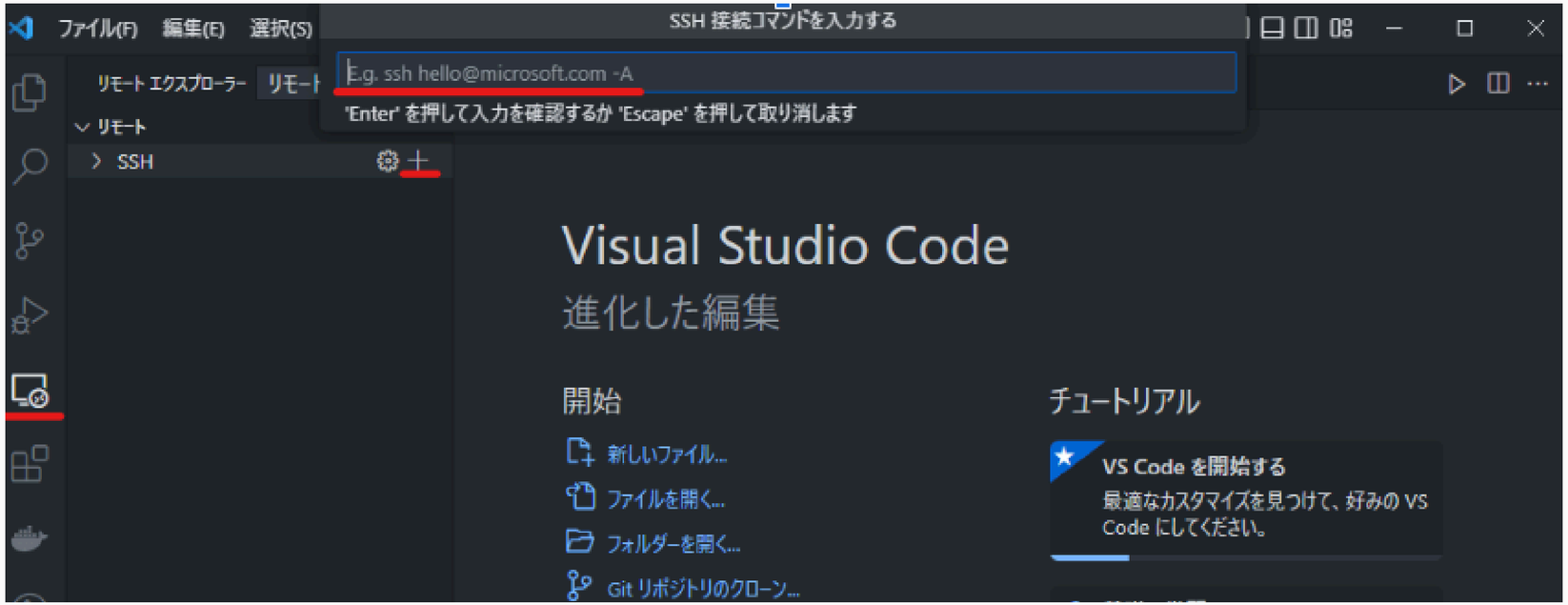
事前準備

- VSCodeの左タブでパソコンマークをクリック > バーの＋ >入力欄で以下コマンドで接続

```
$ ssh -i "C:/Users/自分ユーザ名/Downloads/terraform-control-server.pem" ec2-user@54.92.123.162 -p 22
```

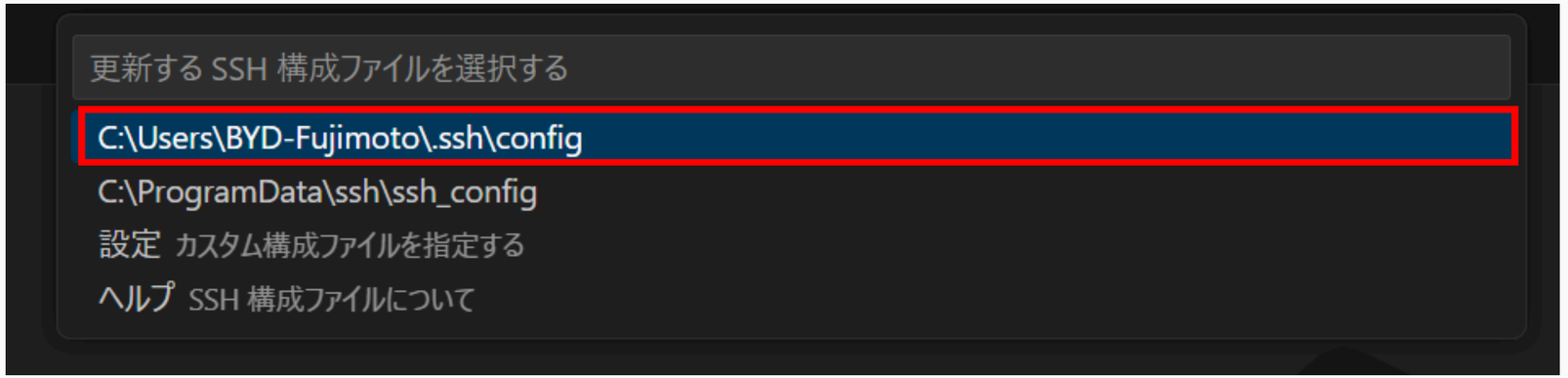
例：

$ ssh -i "C:/Users/BYD-Kitada/Downloads/terraform-control-server.pem" ec2-user@54.92.123.162 -p 22

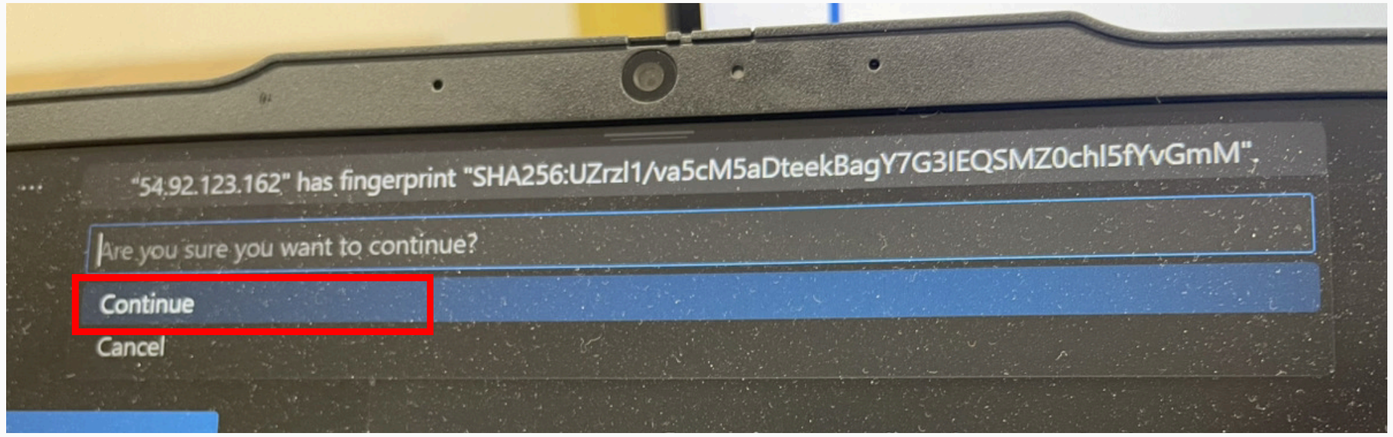


事前準備

- Enterを押したらSSH接続構成ファイル(C:\Users\ユーザ名\.ssh\config)を選択

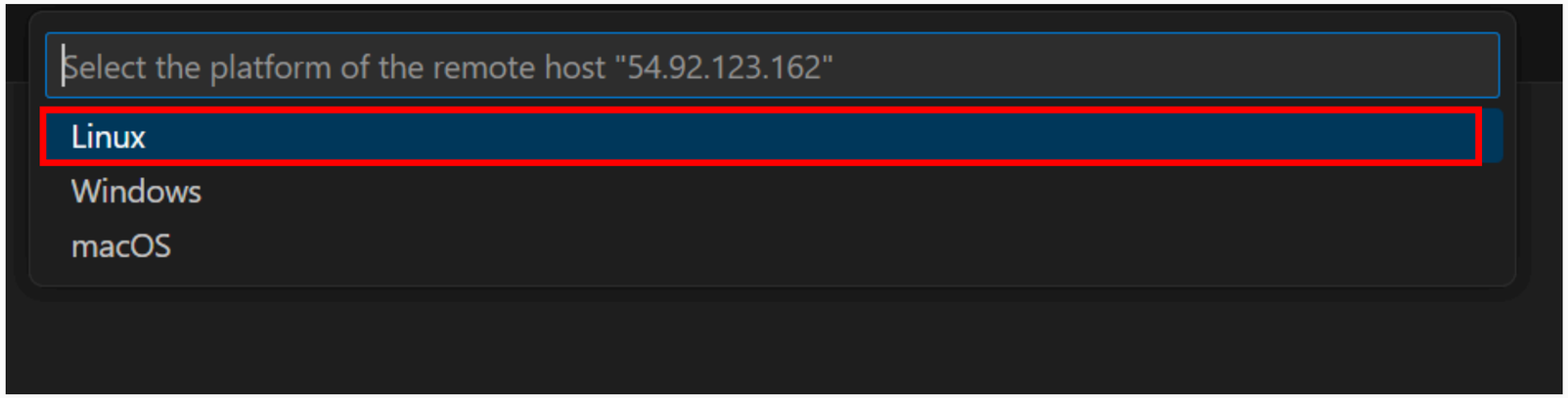


- 「Are you sure you wait to continue?」と出るので Continue を選択



事前準備

- Linux を選択



- 右上のこのボタンを教えてターミナルを出す

事前準備

- ターミナルで以下コマンドでtfenvのパスを通す ※rootにはならないでください。

```
$ source .bash_profile
```

- 使用するTerraformバージョンの指定
  ※インストール済み

```
$ tfenv use 1.11.3
```

Tips　tfenv とは・・・？

Terraformのバージョン管理ツール

複数の環境下でTerraformを使っている場合に、それぞれ使っているバージョンが違う場合バージョン管理を行う為のツールです。

```
[ec2-user@terraform-control-server ~]$ tfenv use 1.11.3
Switching default version to v1.11.3
Default version (when not overridden by .terraform-version or TFENV_TERRAFORM_VERSION) is now: 1.11.3
```

事前準備

- 自分のディレクトリを作成して移動

```
$ mkdir <自分の苗字>
$ cd <自分の苗字>
```

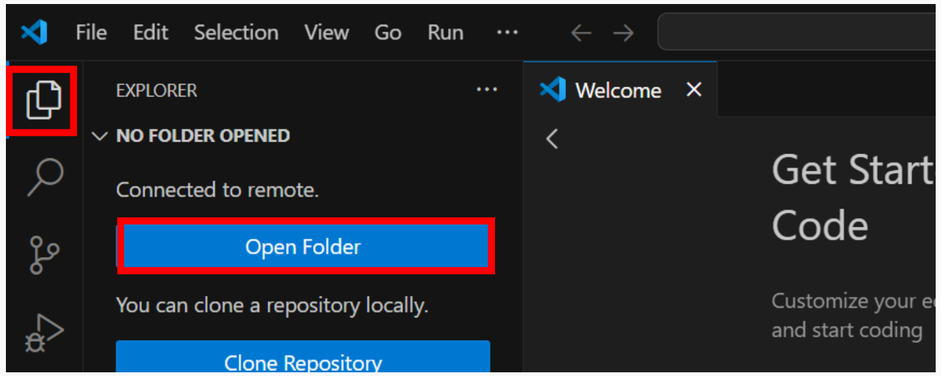
- github からクローンして移動

```
$ git clone https://github.com/shinkitada/education-terrafom.git
$ cd education-terrafom
$ mv terraform.tfvars.example terraform.tfvars
```

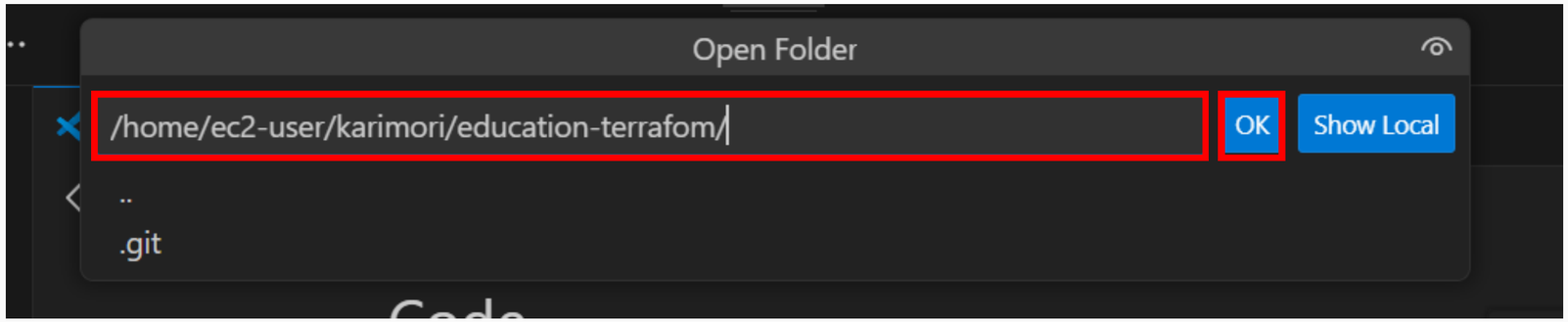
Githubリポジトリ：https://github.com/shinkitada/education-terrafom.git

事前準備

- VSCodeの左タブでパソコンマークをクリック > 「Open Folder」 をクリック

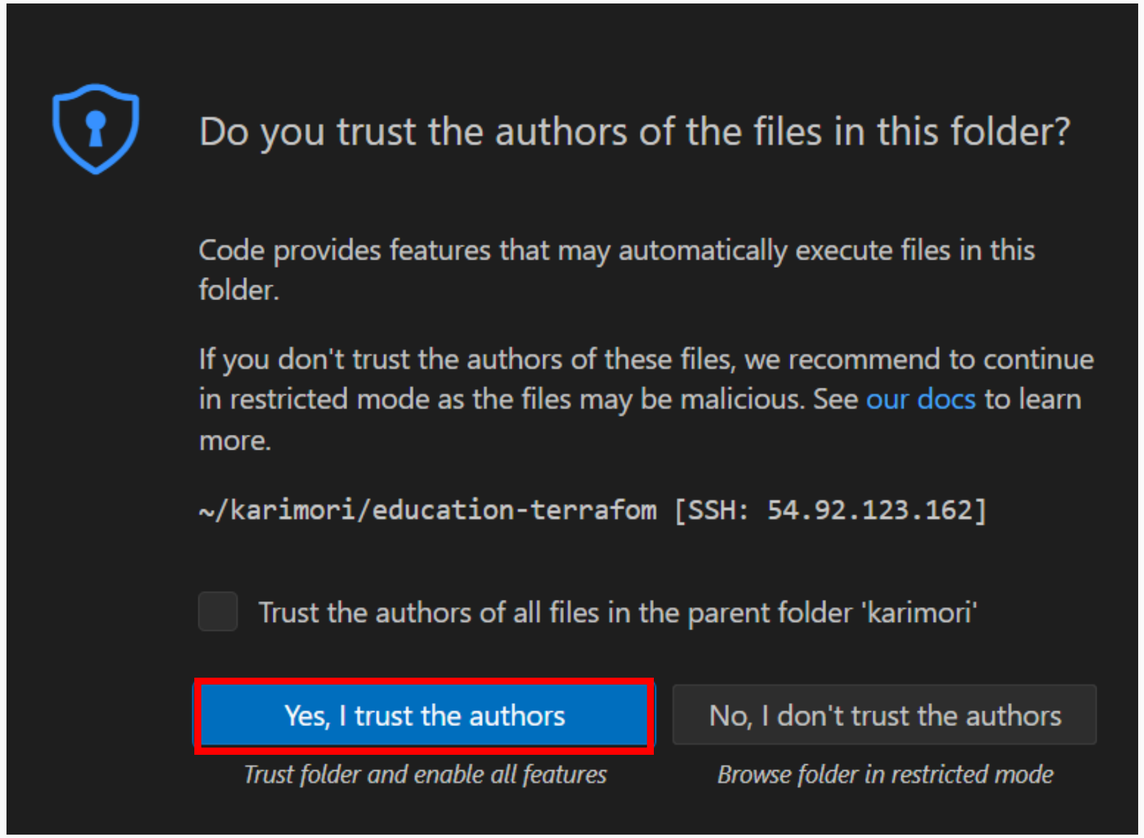


- /home/ec2-user/ユーザ名/education-terraform/ を選択して > OK をクリック



事前準備

- 「Do you trust the authors of the files in this folder?」と表示されるので
  Yes, I trust the authors をクリックする

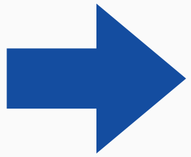


事前準備

- terraform.tfvarsを開いて「name」を自分の苗字に変更して、「Ctrl＋S」で保存する

terraform.tfvars

```
#変数一覧
region      = "ap-northeast-1"
name        = "hogehoge"
environment = "prd"
```



terraform.tfvars

```
#変数一覧
region      = "ap-northeast-1"
name        = "kitada"
environment = "prd"
```

Terraformコマンドについて

一連の流れ

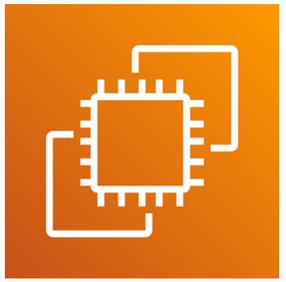
作成

init
初期化

fmt / validate
整形 / 構文を検証

plan
計画 (dry-run)

apply
実行



リソース作成

削除

plan -destroy
削除前確認 (dry-run)

destroy
削除実行

リソース削除

リソース作成

- initでTerraformの初期化、必要なプロバイダー情報やプラグインをインストール

```
$ terraform init
```

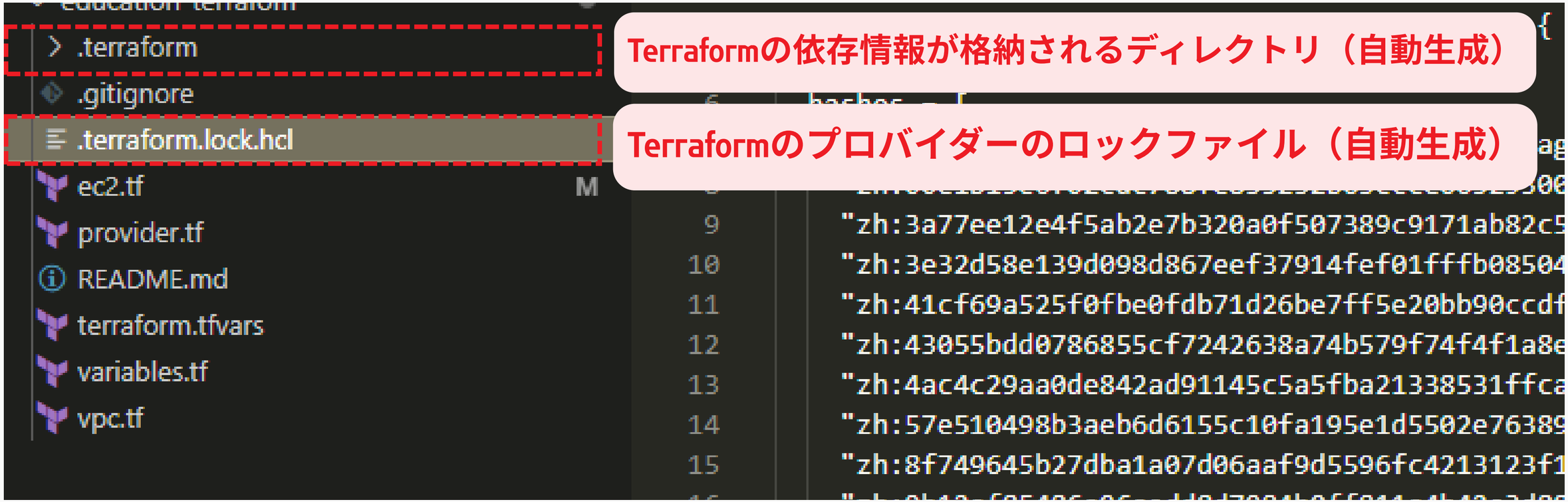
```
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

- fmt / validate でフォーマットの整形と構文を検証

```
$ terraform fmt
$ terraform validate
```

リソース作成

- initを実行すると以下lockファイルと.terraformディレクトリが自動作成されています



リソース作成

- planを実行

```
$ terraform plan
```

```
      + id                   = (known after apply)
    }

  # tls_private_key.keygen will be created
  + resource "tls_private_key" "keygen" {
      + algorithm                     = "RSA"
      + ecdsa_curve                   = "P224"
      + id                            = (known after apply)
      + private_key_openssh           = (sensitive value)
      + private_key_pem               = (sensitive value)
      + private_key_pem_pkcs8         = (sensitive value)
      + public_key_fingerprint_md5    = (known after apply)
      + public_key_fingerprint_sha256 = (known after apply)
      + public_key_openssh            = (known after apply)
      + public_key_pem                = (known after apply)
      + rsa_bits                      = 4096
    }

Plan: 14 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
```

リソース作成

- applyを実行

```
$ terraform apply
```

```
Plan: 14 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

アクションの実行確認が出てくるので「yes」と入力してEnter

リソース作成

```
aws_subnet.my_subnet_1c: Still creating... [10s elapsed]
aws_subnet.my_subnet_1a: Creation complete after 11s [id=subnet-0560fd0f1a615ab00]
aws_subnet.my_subnet_1c: Creation complete after 11s [id=subnet-0899e043a8c366307]
aws_route_table_association.my_route_public_1a: Creating...
aws_route_table_association.my_route_public_1c: Creating...
aws_instance.my_instance: Creating...
aws_route_table_association.my_route_public_1a: Creation complete after 0s [id=rtbassoc-002ed80488604479d]
aws_route_table_association.my_route_public_1c: Creation complete after 0s [id=rtbassoc-0355730e76728d69d]
aws_instance.my_instance: Still creating... [10s elapsed]
aws_instance.my_instance: Creation complete after 12s [id=i-036dbd6866d8712d5]

Apply complete! Resources: 14 added, 0 changed, 0 destroyed.
```

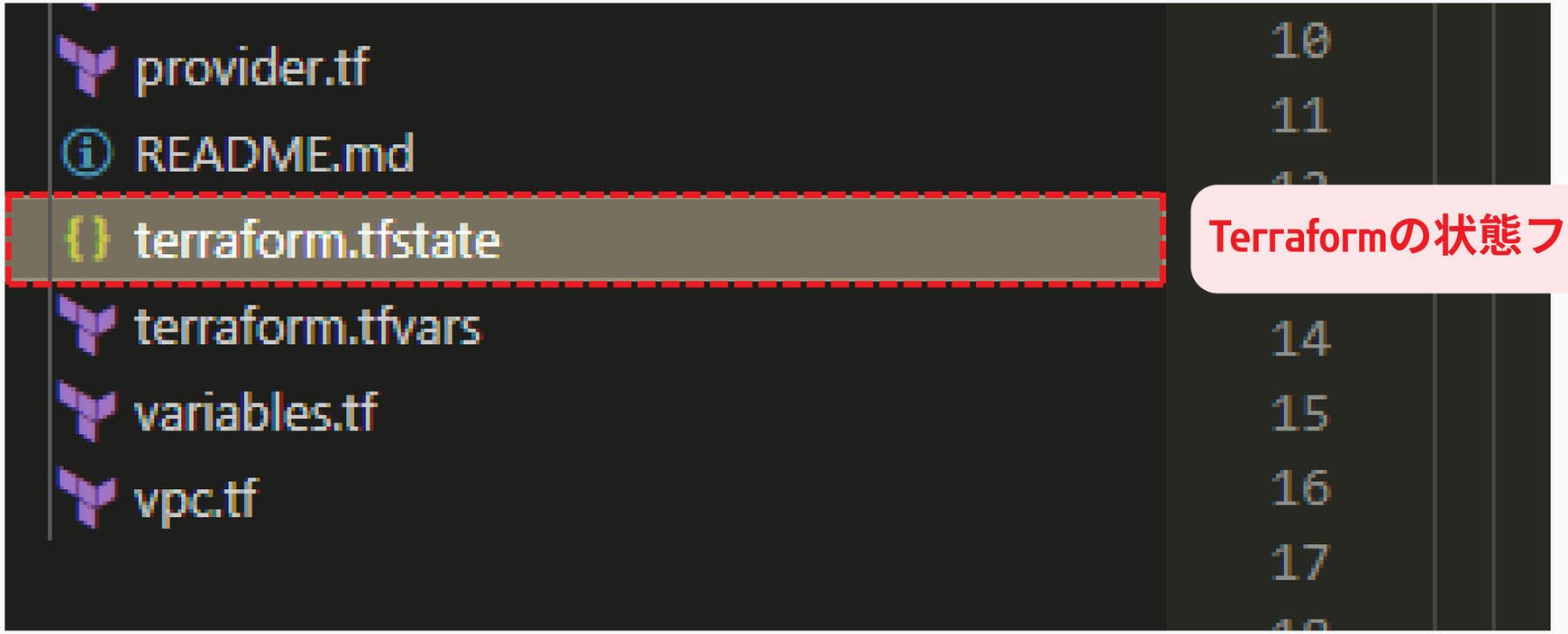
「Apply complete! 」と出ればOK

terraform-edu-fujimoto-instance	i-0c31d83416230632b	実行中	t3.micro

AWSコンソール上から確認すると起動していることが確認できます

リソース作成

- apply を実行すると「terraform.tfstate」が自動作成されています



tfstate ファイルとは

- tfstate には実行で作成されたリソース内容が全て書き込まれています。

つまり、以降 Terraform は tfstate ファイルを参照し
リソースの差分確認（追加リソースの作成や削除）を行います

リソース確認

- では、作成したリソース一覧を確認してみましょう。

```
$ terraform state list
```

```
[ec2-user@terraform-control-server education-terrafom]$ terraform state list
data.http.my_ip
aws_instance.my_instance
aws_internet_gateway.my_igw
aws_key_pair.key_pair
aws_route.my_route
aws_route_table.my_public_rt
aws_route_table_association.my_route_public_1a
aws_route_table_association.my_route_public_1c
aws_security_group.allow_ssh
aws_subnet.my_subnet_1a
aws_subnet.my_subnet_1c
aws_vpc.my-vpc
local_file.private_key_pem
local_file.public_key_pem
tls_private_key.keygen
```

先ほど作成されたリソースの一覧

リソース確認

- 作成したインスタンスのパブリックIPを確認

```
$ terraform state show aws_instance.my_instance | grep public_ip
```

```
t2-user@terraform-control-server education-terrafom]$ terraform state show aws_instance.my_instance | grep publi
  associate_public_ip_address          = true
  public_ip                            = "35.77.104.204"
```

SSH接続を試すので

ここのIPアドレスをメモしておくこと

作成したリソースへ接続

- 秘密鍵が .key ディレクトリ配下に作成されているのでSSH接続してみましょう

```
$ cd .key
$ ls -la
```

```
[ec2-user@terraform-control-server .key]$ ls -la
total 12
drwxr-xr-x. 2 ec2-user ec2-user   60 Apr  7 04:41 .
drwxr-xr-x. 5 ec2-user ec2-user 4096 Apr  7 04:41 ..
-rw-------. 1 ec2-user ec2-user 3247 Apr  7 04:41 kitada-key.id_rsa
-rw-------. 1 ec2-user ec2-user  800 Apr  7 04:41 kitada-key.id_rsa.pub
```

自分の名前の秘密鍵、公開鍵が出来ていればOK

```
$ ssh -i "自分の名前-key.id_rsa" ec2-user@作成したインスタンスのパブリックIPアドレス
```

例：

$ ssh -i "kitada-key.id_rsa" ec2-user@35.77.104.204

作成したリソースへ接続

```
-rw-------. 1 ec2-user ec2-user 3247 Apr  7 04:41 kitada-key.id_rsa
-rw-------. 1 ec2-user ec2-user  800 Apr  7 04:41 kitada-key.id_rsa.pub
[ec2-user@terraform-control-server .key]$ ssh -i "kitada-key.id_rsa" ec2-user@35.77.104.204
The authenticity of host '35.77.104.204 (35.77.104.204)' can't be established.
ED25519 key fingerprint is SHA256:CnEaABg1Y8SiAAqeZzR5C5VaX2rrn5o+WHqGQ4KHzv8.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '35.77.104.204' (ED25519) to the list of known hosts.
```

接続確認が出てくるので「yes」と入力してEnter

```
Warning: Permanently added '35.77.104.204' (ED25519) to the list of known hosts.
[ec2-user@ip-10-0-0-207 ~]$
```

接続できればOKです

リソースの削除

- exit か Ctrl + d で作成したEC2から抜ける

```
$ exit
```

```
[ec2-user@ip-10-0-0-207 ~]$ exit
logout
Connection to 35.77.104.204 closed.
[ec2-user@terraform-control-server .key]$
```

controlサーバーに戻ればOKです

- education-terraformディレクトリに戻る

```
$ cd ..
```

リソースの削除

- 削除前のドライランを実行

```
$ terraform plan -destroy
```

```
            SizlE80UJYp8PLvoSEzeoavjaYgU5/nAzwXFSk1bFQ+nRnvFTMDyJY1jYJQm0Gwx
            ig68mX7vNSxwDY+1lioAmDLGuR+ZbwjQnIH3L+rolZZAvGhTWb+96FazSuy9Vt3/
            f/O7vaCtdbpAQjHbV0OblFbWV8sBbbEkdSrr6moMc3+YcVSVz+GeyyQmSh1vuKTH
            0hgIbTKek9TFOcwJ6rl7LXppE9XI9dSN2k96EtthbV+B5SS6dcCxeVtytGF1CtH8
            a817mfFVOv+a2p29WpDrlE1gDJvI0XHtSOYEz0zdeUpa3sPAvzzgZJbFwKoA8BzW
            divknOrl8HT0FDX9tO5mIRkWmeWNVKCeDrUbMxTPUsQhTGZDAapz6S8XGZ+1cLqZ
            1n6x+U5Yuic06fiFcaLs/YM0kAVbrFn1xxFnQwslrbqu6vSxpC/oBxkURAqUI0BM
            luBmrUBfdNnLNpVruBn5qXu3cOIhueOJnpCtumR/5XGAhdS4GsD/qSeJvfZI+PPa
            UK/25p/p4A8mYQ9AjoUryfMeQg0U2hDCzQi3sHFmpVBLTMJmsIK2G/ykxCW2UqlN
            gKRaaoniZXGQz2JDY8bkAvVszNmxS5zi2dm4K7QftdOcEShyuHk2YYK3yI8rAJgK
            +A1LwMl38DVblUNvLXMvG1sCAwEAAQ==
            -----END PUBLIC KEY-----
        EOT -> null
      - rsa_bits                      = 4096 -> null
    }

Plan: 0 to add, 0 to change, 14 to destroy.

---

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
[ec2-user@terraform-control-server education-terrafom]$
```

リソースの削除

- リソースの削除を行う

```
$ terraform destroy
```

```
Plan: 0 to add, 0 to change, 14 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

アクションの実行確認が出てくるので「yes」と入力してEnter

リソースの削除

```
aws_key_pair.key_pair: Destroying... [id=kitada-key]
aws_subnet.my_subnet_1a: Destroying... [id=subnet-0560fd0f1a615ab00]
aws_key_pair.key_pair: Destruction complete after 0s
tls_private_key.keygen: Destroying... [id=20408652f94411b8da2137fbf552a11aab297fdb]
tls_private_key.keygen: Destruction complete after 0s
aws_subnet.my_subnet_1a: Destruction complete after 0s
aws_security_group.allow_ssh: Destruction complete after 1s
aws_vpc.my-vpc: Destroying... [id=vpc-0e5931c2daf849d3a]
aws_vpc.my-vpc: Destruction complete after 0s

Destroy complete! Resources: 14 destroyed.
```

「Destroy complete ! Resources: ~ destroyed! 」

と出れば削除完了です

コード解説

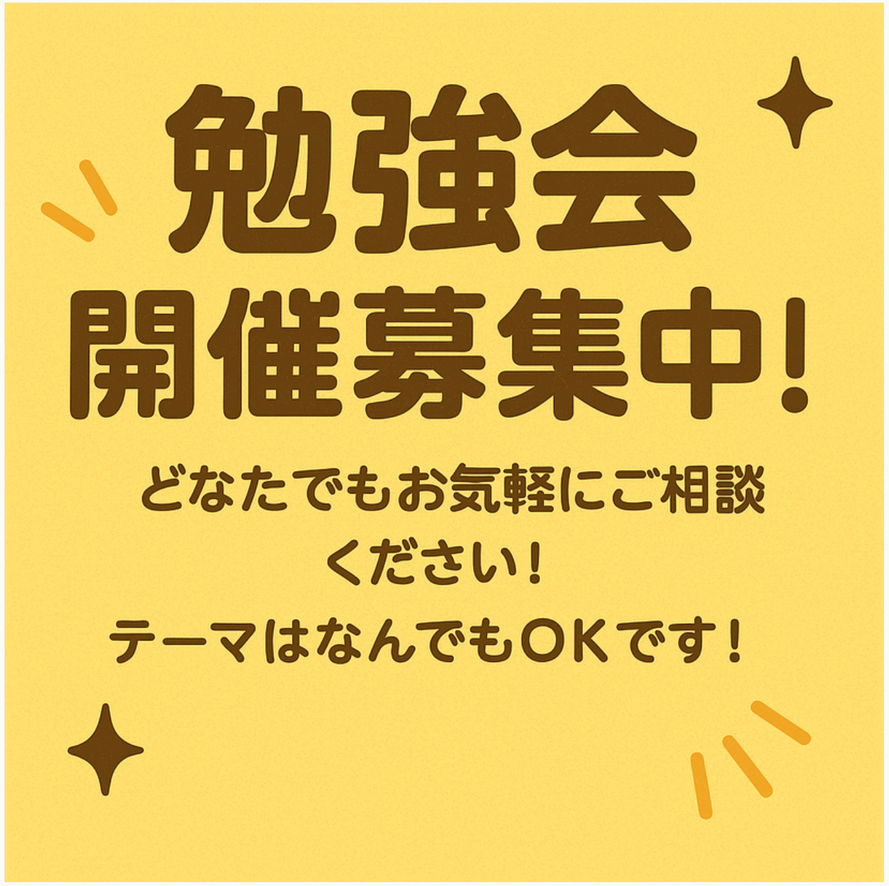
- ではコードの解説をしていきます。
  解説は以下の記事で行っているのでこのページに飛んで説明を行います。

  リンク： https://qiita.com/kitasan_chi/items/caed57bf5d4af00095d0

最後に（告知）

告知その１
【もくもく会、始めました】

告知その２
【勉強会開催募集中です】



最後に

本日はご参加ありがとうございました！

また次回もお待ちしております〜