



Networld



HashiCorp

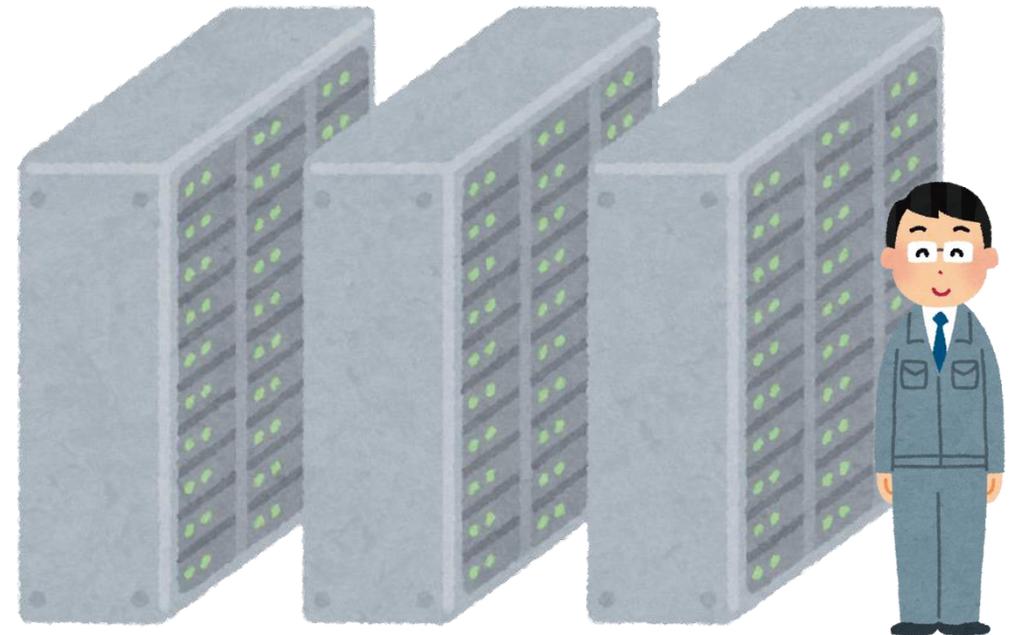
HashiCorp

Terraform Enterprise概要資料

ネットワークド HashiCorpチーム

もくじ

1. Terraform による Infrastructure as Code とは？
2. Terraform のデモ (動画)
3. Terraform Enterprise とは？
4. Terraform Enterprise のデモ (動画)
5. まとめ





Terraform による Infrastructure as Code とは？

Infrastructure as Code (Iac) とは？

インフラ

CPUやメモリ、ディスクといったリソース、
あるいは仮想マシンやアプライアンスそのもの



を

コードで 表現すること



Terraform で Infrastructure as Code を実現する



再利用可能な
ソースコードとして
インフラの構成を
定義する

多様なインフラを
ワークフローを
変えることなく
導入する

Azure 上の HashiCorp Terraform

使い慣れた自動化ツールを使用して、コードとしてのインフラストラクチャを簡単に管理する

ドキュメントを見つける >

Azure を初めてご利用になりますか? 無料で始める >

概要 シナリオ お客様事例 開始する ブログ ドキュメント >

インフラストラクチャ管理の簡略化

仮想マシン (VM)、ネットワーク、コンテナなどのインフラストラクチャ リソースの作成、管理、更新に利用できる富言的な構成ファイルで、コードとしてのインフラストラクチャを定義します。Terraform 構成言語を使用すると、ワークフロー全体でリソース管理を簡単に自動化できます。

Azure で Terraform を使用する理由

- 事前統合**
Terraform は [Azure Cloud Shell](#) に組み込まれており、お持ちのサブスクリプションに対して認証されているため、統合済みで、準備が整っています。Visual Studio Code の [Azure Terraform 拡張機能](#) を使用して Azure 内でモジュールの構築とテストを行うことにより、Terraform コマンドのサポート、グラフを使ったリソースの視覚化、Visual Studio Code 内の Azure Cloud Shell の直接の統合が実現します。
- コミュニティ主導**
Microsoft と HashiCorp のエンジニアリング チームは、今後も Terraform コミュニティと協力して Azure Terraform [プロバイダーとモジュール](#) の開発を進めていきます。機能のリクエスト、問題の報告、開発への協力をご希望の方は、[GitHub リポジトリ](#) にご参加ください。
- エンタープライズ対応**
[Terraform Enterprise](#) を使用して業務を効率化し、あらゆるインフラストラクチャをより安全かつ効率的にプロビジョニングします。インフラストラクチャの展開を 1 つのワークフローに一元化し、あらゆる環境のプロビジョニング、管理、監査を行います。

出典 : Azure 上の HashiCorp Terraform
<https://azure.microsoft.com/ja-jp/solutions/devops/terraform/>

Terraformで超サクッとループでリソースを用意する方法

2019年08月22日 6 コメント 19 いいね

こんにちは、朝7時に起きるのがしんどくなってきた、もこ@札幌オフィスです。

ネットワークを構築する際、Variableにいい感じに必要な情報を代入してループで一気にresourceをつくってみたので、Terraformを公開します。

尚、Terraformのバージョンはv0.12.6、aws providerはv2.24.0を利用しています

やってみた

このようなVPC情報とサブネット情報が含まれた変数があったとします。

```
1 variable "production-vpc" {
2   default = {
3     name = "production-vpc"
4     cidr = "10.0.0.0/16"
5     subnets = {
6       subnet1 = {
```

出典 : Terraformで超サクッとループで
リソースを用意する方法
<https://dev.classmethod.jp/cloud/aws/terraform-network-variable/>

Terraform による IaC で開発者が自らインフラを構築できる

導入前



ソフトウェア
開発者



インフラ要件
(作業指示書)



インフラ SE が
作業を実施



開発者が
要求した
リソース

導入後



ソフトウェア
開発者



Terraform の
コード (HCL)



Terraform が
プロビジョニング

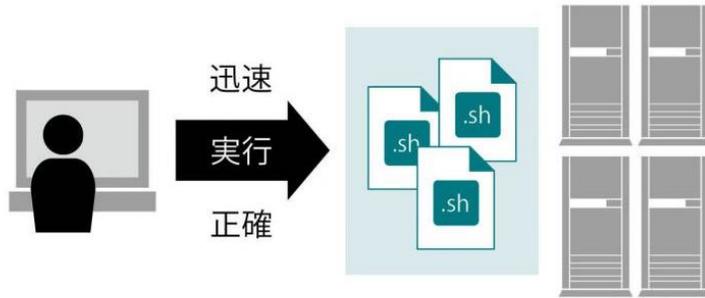


開発者が
要求した
リソース

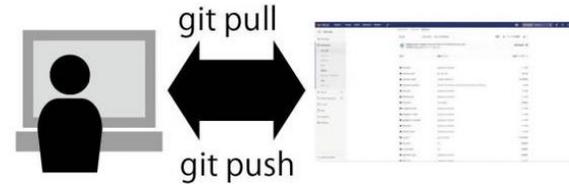
Infrastructure as Code のメリットとは

図表3 インフラ構築・運用をコードで記述することのメリット

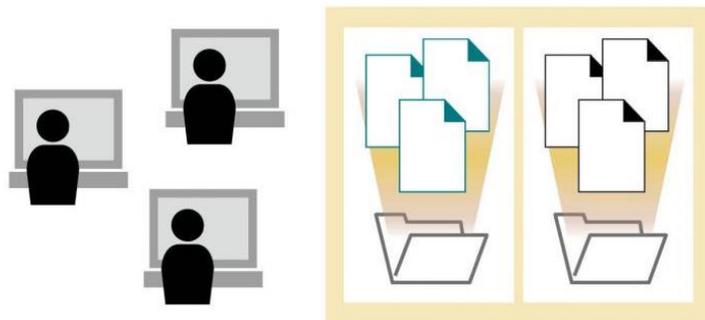
専用のUI (とくにGUI) を通さずに
インフラの変更を自動化できる



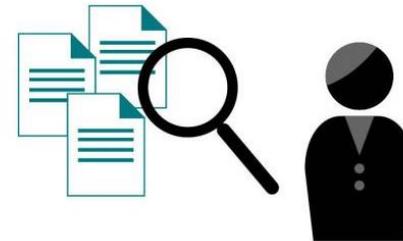
ソフトウェア開発と同じ手法で
バージョン管理を行える



一度記述したコードは
共有・再利用が可能である



コード化された手順を第三者が検証できる



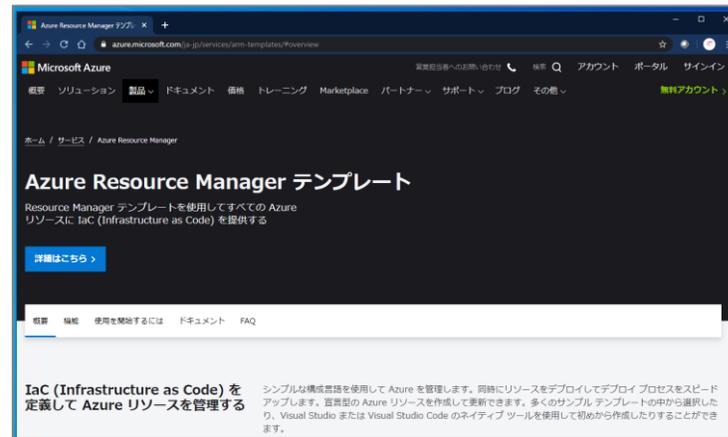
出典 : [iMagazine] Infrastructure as Codeの留意点とメリット サーバー更改プロジェクトへの適用で得られた知見・実感
<https://www.imagazine.co.jp/infrastructure-as-codeの留意点とメリット%E3%80%80~サーバー更改プロ/>

各クラウドやプロダクト専用の IaC ツールはある

AWS なら CloudFormation など



Azure なら ARM テンプレートなど

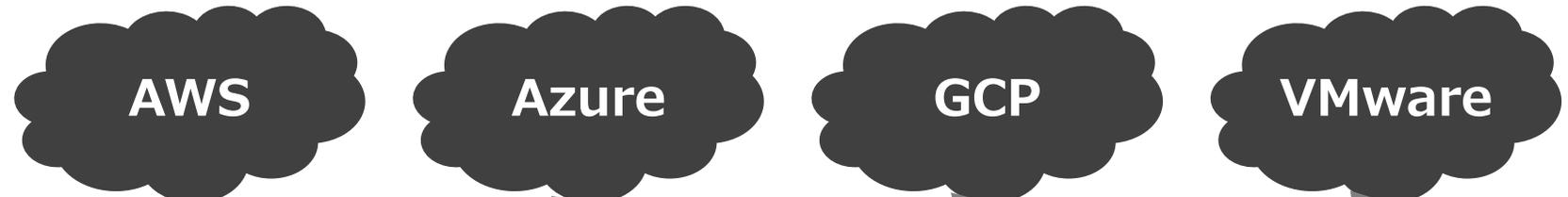


VMware なら PowerCLI など



Ansible や Puppet などは
もう少し OS に寄ってるイメージ...

Terraform : ひとつのコード体系 (HCL) で さまざまな環境に対応



A screenshot of a GitHub web interface showing a Terraform configuration file. The browser address bar indicates the repository is 'terraform-nutanix-sample'. The code is written in HCL and includes sections for 'Define Variables' and 'Provider'. The 'Define Variables' section lists variables such as 'prov_username', 'prov_password', 'prov_endpoint', 'prov_vname_prefix', 'prov_num', 'prov_subnet_cidr', 'prov_diskimage_cidr', 'prov_vcdm', 'prov_sock', and 'prov_men'. The 'Provider' section defines the 'nutanix' provider with configuration options like 'username', 'password', 'endpoint', 'insecure', and 'port'.

Terraform があると...

- 異なるクラウド/オンプレミスでも単一のコードで対応可能
- 環境変更の前にレビューがしやすい
- コードに基づいた自動作業だから作業漏れが発生しない
- 現在の状態を State ファイルで確認できる
- 環境の変更履歴が追える
- 作成したコードを再利用できる

Terraform だけでこれだけの環境のプロビジョニングが可能

ACME	Cloudflare	GitHub	Local	OVH	Spotinst
Akamai	CloudScale.ch	GitLab	Logentries	Packet	StatusCake
Alibaba Cloud	CloudStack	Google Cloud Platform	LogicMonitor	PagerDuty	TelefonicaOpenCloud
Archive	Cobbler	Grafana	Mailgun	Palo Alto Networks	Template
Arukas	Consul	Gridscale	MongoDB Atlas	PostgreSQL	TencentCloud
Avi Vantage	Datadog	Hedvig	MySQL	PowerDNS	Terraform
Aviatrix	DigitalOcean	Helm	Naver Cloud	ProfitBricks	Terraform Cloud
AWS	DNS	Heroku	Netlify	Pureport	TLS
Azure	DNSimple	Hetzner Cloud	New Relic	RabbitMQ	Triton
Azure Active Directory	DNSMadeEasy	HTTP	Nomad	Rancher	UCloud
Azure Stack	Docker	HuaweiCloud	NS1	Rancher2	UltraDNS
A10 Networks	Dome9	HuaweiCloudStack	Null	Random	Vault
Bitbucket	Dyn	Icinga2	Nutanix	RightScale	Venafi
Brightbox	Exoscale	Ignition	1&1	Rundeck	VMware NSX-T
CenturyLinkCloud	External	InfluxDB	OpenStack	RunScope	VMware vCloud Director
Chef	F5 BIG-IP	JDCloud	OpenTelekomCloud	Scaleway	VMware vRA7
CherryServers	Fastly	Kubernetes	OpsGenie	Selectel	VMware vSphere
Circonus	FlexibleEngine	LaunchDarkly	Oracle Cloud Infrastructure	SignalFx	Vultr
Cisco ASA	FortiOS	Librato	Oracle Cloud Platform	Skytap	Yandex
Cisco ACI	Genymotion	Linode	Oracle Public Cloud	SoftLayer	

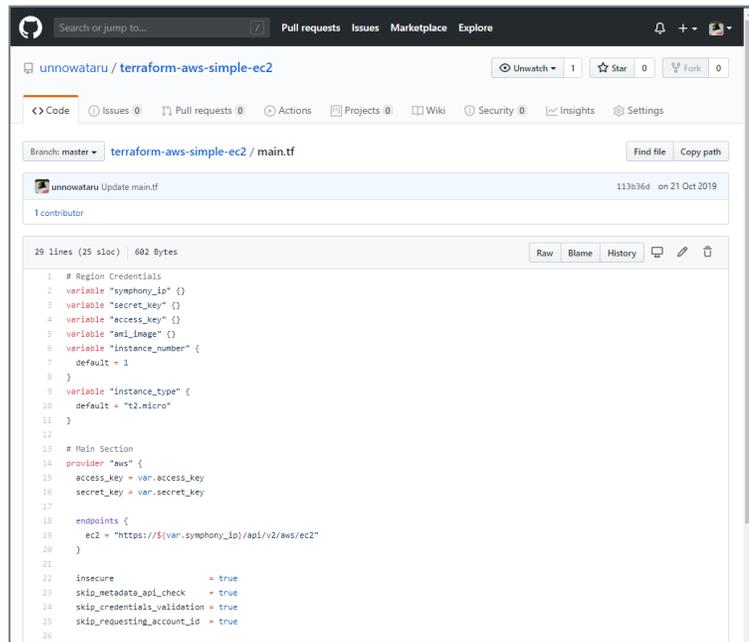
出典 : HashiCorp Terraform Providers

<https://www.terraform.io/docs/providers/index.html>

コードで 開発者 (Developer) が必要なリソースを割り当て

Terraform と **AWS** の組み合わせ例

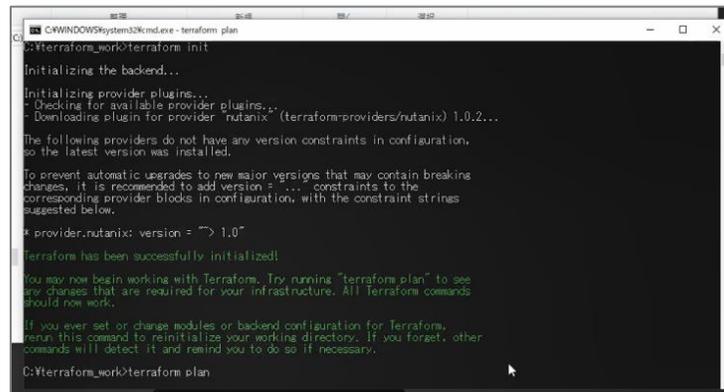
コードの作成



```
1 # Region Credentials
2 variable "symphony_ip" {}
3 variable "secret_key" {}
4 variable "access_key" {}
5 variable "ami_image" {}
6 variable "instance_number" {
7   default = 1
8 }
9 variable "instance_type" {
10  default = "t2.micro"
11 }
12
13 # Main Section
14 provider "aws" {
15   access_key = var.access_key
16   secret_key = var.secret_key
17 }
18 endpoints {
19   ec2 = "https://s${var.symphony_ip}/api/v2/aws/ec2"
20 }
21
22 insecure           = true
23 skip_metadata_api_check = true
24 skip_credentials_validation = true
25 skip_requesting_account_id = true
```

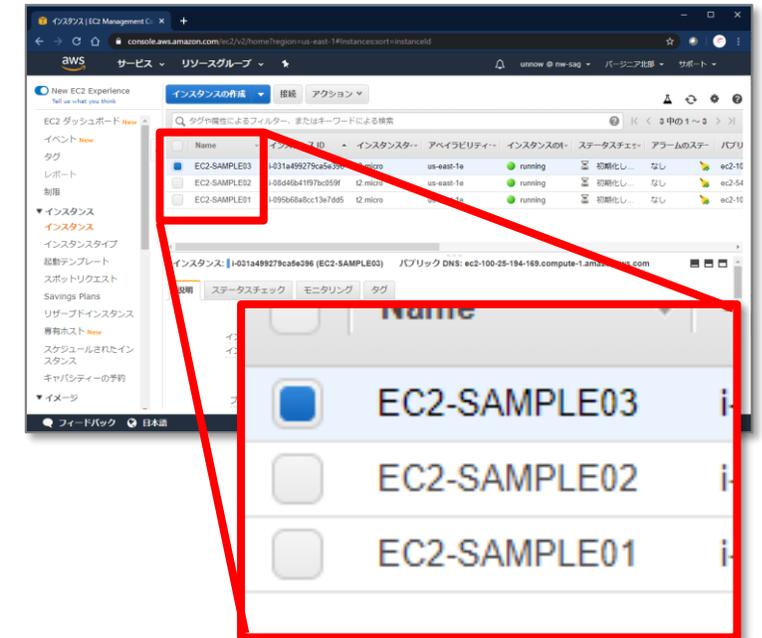
コードの実行

- Terraform INIT
- Terraform PLAN
- Terraform APPLY
- Terraform DESTROY



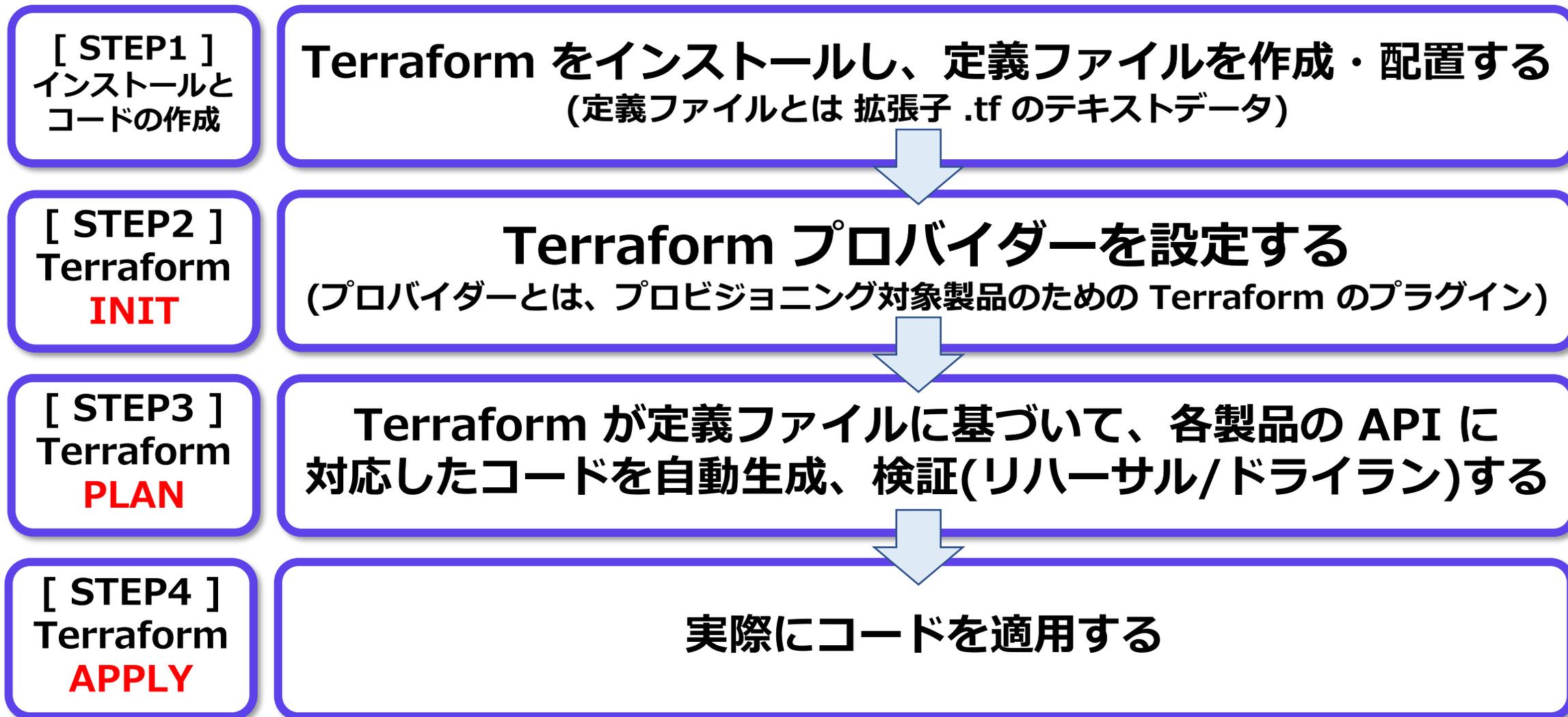
```
C:\WINDOWS\system32\cmd.exe - terraform plan
C:\terraform_work>terraform init
[Initializing the backend...]
[Initializing provider plugins...]
- Checking for available provider plugins...
- Downloading plugin for provider "nutanix" (terraform-providers/nutanix) 1.0.2...
The following providers do not have any version constraints in configuration,
so the latest version was installed.
To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = ... constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.
# provider.nutanix: version = "> 1.0"
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.
C:\terraform_work>terraform plan
```

仮想マシンの**作成**・**削除**



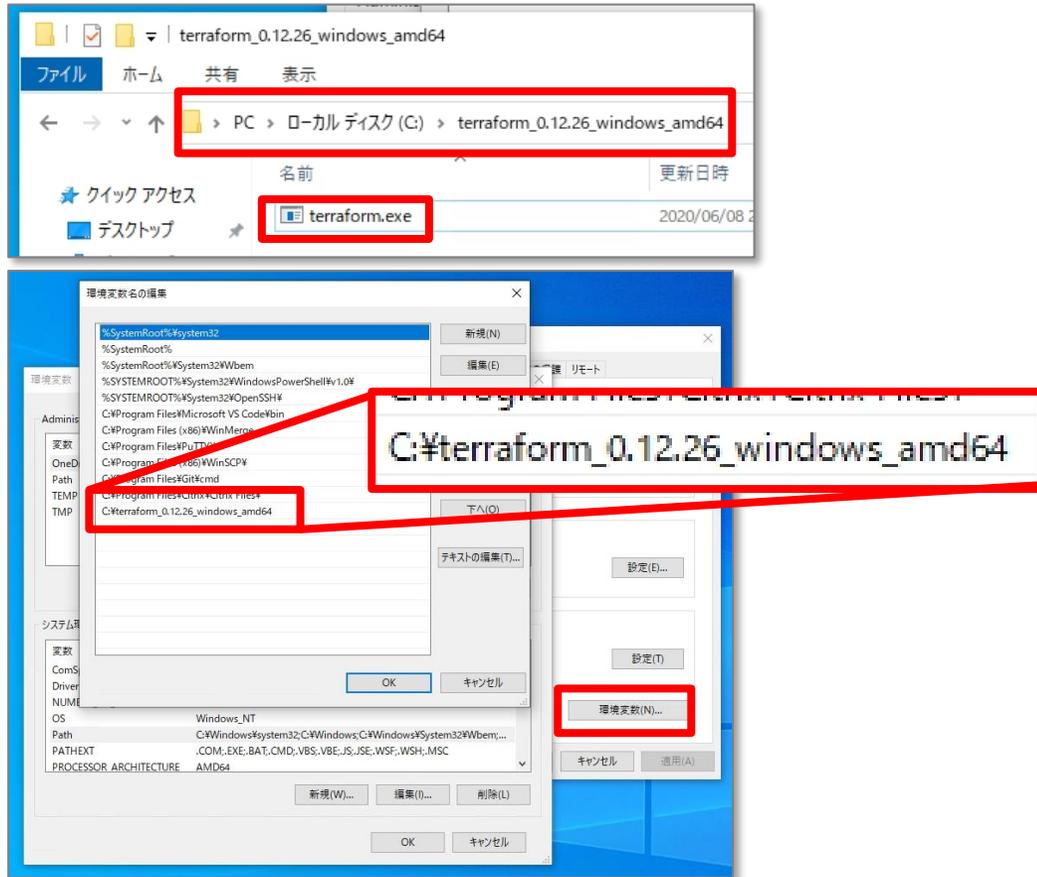
<https://github.com/unnowataru/terraform-aws-simple-ec2>

Terraform を実行するためのステップ

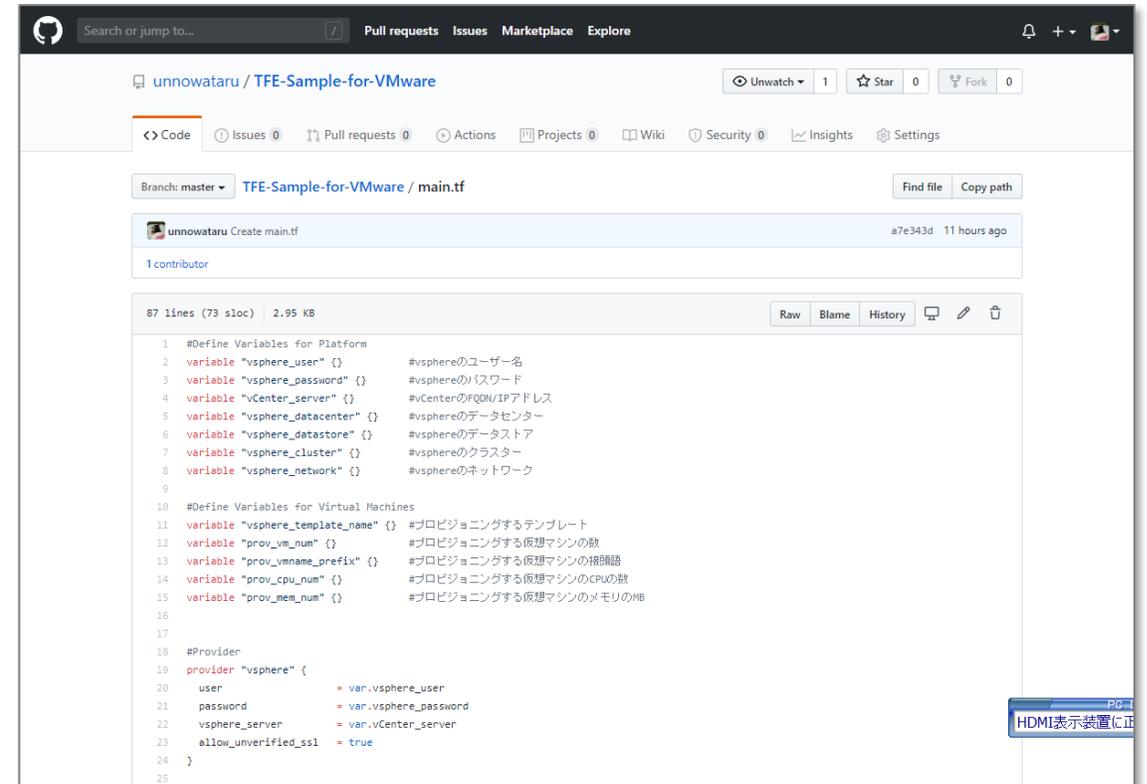


[STEP1] インストールとコードの作成

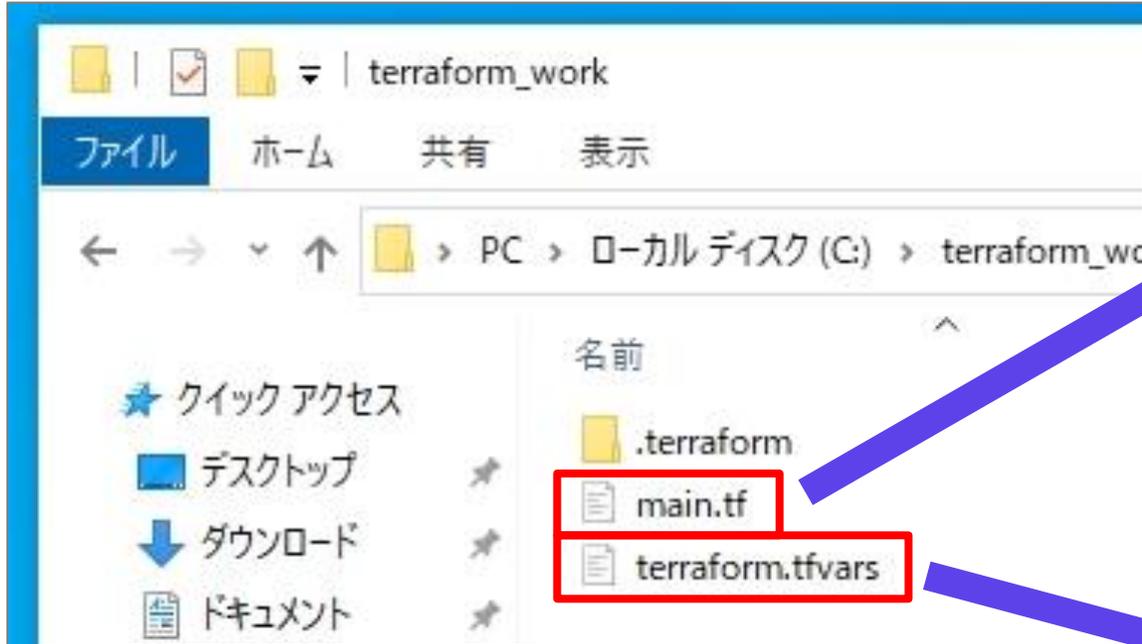
インストール



コードの作成



Terraform のコード : vSphere 向けサンプル



[main.tf]
Terraformのメインコード

[terraform.tfvars]
変数のみを格納しているファイル

 いますぐダウンロードできます!!

[https://github.com/unnowataru/
TFE-Sample-for-VMware](https://github.com/unnowataru/TFE-Sample-for-VMware)

Terraform のコード解説 : main.tf

```
1 #Define Variables for Platform
2 variable "vsphere_user" {} #vsphereのユーザー名
3 variable "vsphere_password" {} #vsphereのパスワード
4 variable "vCenter_server" {} #vCenterのFQDN/IPアドレス
5 variable "vsphere_datacenter" {} #vsphereのデータセンター
6 variable "vsphere_datastore" {} #vsphereのデータストア
7 variable "vsphere_cluster" {} #vsphereのクラスター
8 variable "vsphere_network" {} #vsphereのネットワーク
9
10 #Define Variables for Virtual Machines
11 variable "vsphere_template_name" {} #プロビジョニングするテンプレート
12 variable "prov_vm_num" {} #プロビジョニングする仮想マシンの数
13 variable "prov_vmname_prefix" {} #プロビジョニングする仮想マシンの接頭語
14 variable "prov_cpu_num" {} #プロビジョニングする仮想マシンのCPUの数
15 variable "prov_mem_num" {} #プロビジョニングする仮想マシンのメモリのMB
16
17 #Provider
18 provider "vsphere" {
19   user = var.vsphere_user
20   password = var.vsphere_password
21   vsphere_server = var.vcenter_server
22   allow_unverified_ssl = true
23 }
24
25 #Data
26 data "vsphere_datacenter" "dc" {
27   name = var.vsphere_datacenter
28 }
29
30 data "vsphere_datastore" "datastore" {
31   name = var.vsphere_datastore
32   datacenter_id = data.vsphere_datacenter.dc.id
33 }
34
35 data "vsphere_compute_cluster" "cluster" {
36   name = var.vsphere_cluster
37   datacenter_id = data.vsphere_datacenter.dc.id
38 }
39
40 data "vsphere_network" "network" {
41   name = var.vsphere_network
42   datacenter_id = data.vsphere_datacenter.dc.id
43 }
44
45 data "vsphere_virtual_machine" "template" {
46   name = var.vsphere_template_name
47   datacenter_id = data.vsphere_datacenter.dc.id
48 }
49
50 #Resource
51 resource "vsphere_virtual_machine" "vm" {
52   count = var.prov_vm_num
53   name = "${var.prov_vmname_prefix}${format("%03d", count.index+1)}"
54   resource_pool_id = data.vsphere_compute_cluster.cluster.resource_pool_id
55   datastore_id = data.vsphere_datastore.datastore.id
56 }
57
58 #Resource for VM Specs
59 num_cpus = var.prov_cpu_num
60 memory = var.prov_mem_num
61 guest_id = data.vsphere_virtual_machine.template.guest_id
62 scsi_type = data.vsphere_virtual_machine.template.scsi_type
63 network_interface {
64   network_id = data.vsphere_network.network.id
65   adapter_type = "vmxnet3"
66 }
67
68 #Resource for Disks
69 disk {
70   label = "disk1"
71   size = data.vsphere_virtual_machine.template.disks.0.size
72   eagerly_scrub = data.vsphere_virtual_machine.template.disks.0.eagerly_scrub
73   thinly_provisioned = data.vsphere_virtual_machine.template.disks.0.thin_provisioned
74 }
75
76 clone {
77   template_uuid = data.vsphere_virtual_machine.template.id
78   customize {
79     windows_options {
80       computer_name = "${var.prov_vmname_prefix}${format("%03d", count.index+1)}"
81       network_interface {} #テンプレートと同一のNIC枚数
82     }
83   }
84 }
85
86 }
```

variable : 変数を定義するセクション

ユーザー名やvCenterのFQDN、プロビジョニングしたい仮想マシンの名前、スペックなどを変数として定義している。

```
1 #Define Variables for Platform
2 variable "vsphere_user" {} #vsphereのユーザー名
3 variable "vsphere_password" {} #vsphereのパスワード
4 variable "vCenter_server" {} #vCenterのFQDN/IPアドレス
5 variable "vsphere_datacenter" {} #vsphereのデータセンター
6 variable "vsphere_datastore" {} #vsphereのデータストア
7 variable "vsphere_cluster" {} #vsphereのクラスター
8 variable "vsphere_network" {} #vsphereのネットワーク
9
10 #Define Variables for Virtual Machines
11 variable "vsphere_template_name" {} #プロビジョニングするテンプレート
12 variable "prov_vm_num" {} #プロビジョニングする仮想マシンの数
13 variable "prov_vmname_prefix" {} #プロビジョニングする仮想マシンの接頭語
14 variable "prov_cpu_num" {} #プロビジョニングする仮想マシンのCPUの数
15 variable "prov_mem_num" {} #プロビジョニングする仮想マシンのメモリのMB
```

Terraform のコード解説 : main.tf

```
87 lines (73 sloc) | 2.95 KB
1 #Define Variables for Platform
2 variable "vsphere_user" {} #vsphereのユーザー名
3 variable "vsphere_password" {} #vsphereのパスワード
4 variable "vCenter_server" {} #vCenterのFQDN/IPアドレス
5 variable "vsphere_datacenter" {} #vsphereのデータセンター
6 variable "vsphere_datastore" {} #vsphereのデータストア
7 variable "vsphere_cluster" {} #vsphereのクラスター
8 variable "vsphere_network" {} #vsphereのネットワーク
9
10 #Define Variables for Virtual Machines
11 variable "vsphere_template_name" {} #プロビジョニングするテンプレート
12 variable "prov_vm_num" {} #プロビジョニングする仮想マシンの数
13 variable "prov_vmname_prefix" {} #プロビジョニングする仮想マシンの接頭辞
14 variable "prov_cpu_num" {} #プロビジョニングする仮想マシンのCPUの数
15 variable "prov_mem_num" {} #プロビジョニングする仮想マシンのメモリ
16
17
18 #Provider
19 provider "vsphere" {
20     user = var.vsphere_user
21     password = var.vsphere_password
22     vsphere_server = var.vCenter_server
23     allow_unverified_ssl = true
24 }
25
26 #Data
27 data "vsphere_datacenter" "dc" {
28     name = var.vsphere_datacenter
29 }
30
31 data "vsphere_datastore" "datastore" {
32     name = var.vsphere_datastore
33     datacenter_id = data.vsphere_datacenter.dc.id
34 }
35
36 data "vsphere_compute_cluster" "cluster" {
37     name = var.vsphere_cluster
38     datacenter_id = data.vsphere_datacenter.dc.id
39 }
40
41 data "vsphere_network" "network" {
42     name = var.vsphere_network
43     datacenter_id = data.vsphere_datacenter.dc.id
44 }
45
46 data "vsphere_virtual_machine" "template" {
47     name = var.vsphere_template_name
48     datacenter_id = data.vsphere_datacenter.dc.id
49 }
50
51 #Resource
52 resource "vsphere_virtual_machine" "vm" {
53     count = var.prov_vm_num
54     name = "${var.prov_vmname_prefix}${format("%03d", count.index+1)}"
55     resource_pool_id = data.vsphere_compute_cluster.cluster.resource_pool_id
56     datastore_id = data.vsphere_datastore.datastore.id
57 }
58 #Resource for VM Specs
59 num_cpus = var.prov_cpu_num
60 memory = var.prov_mem_num
61 guest_id = data.vsphere_virtual_machine.template.guest_id
62 scsi_type = data.vsphere_virtual_machine.template.scsi_type
63
64 network_interface {
65     network_id = data.vsphere_network.network.id
66     adapter_type = "vmxnet3"
67 }
68
69 #Resource for Disks
70 disk {
71     label = "disk1"
72     size = data.vsphere_virtual_machine.template.disks.0.size
73     eagerly_scrub = data.vsphere_virtual_machine.template.disks.0.eagerly_scrub
74     thin_provisioned = data.vsphere_virtual_machine.template.disks.0.thin_provisioned
75 }
76
77
78 clone {
79     template_uuid = data.vsphere_virtual_machine.template.id
80     customize {
81         windows_options {
82             computer_name = "${var.prov_vmname_prefix}${format("%03d", count.index+1)}"
83         }
84         network_interface {} #テンプレートと同一のNIC(複数)
85     }
86 }
87 }
```

provider : インフラを定義するセクション

ここでは vsphere としているが、AWS や Azure などプロビジョニング対象となるインフラをここで定義する。

ちなみに "vsphere" の場合、ちょっとカッコつけようと思って "vSphere" とか書いたりすると terraform plan 実行時に怒られることがあるので注意。

これで2時間ぐらいハマりました(;▽;)

```
18 #Provider
19 provider "vsphere" {
20     user = var.vsphere_user
21     password = var.vsphere_password
22     vsphere_server = var.vCenter_server
23     allow_unverified_ssl = true
24 }
```

Terraform のコード解説 : main.tf

```
87 lines (73 sloc) | 2.95 KB
1 #Define Variables for Platform
2 variable "vsphere_user" {} #vsphereのユーザー名
3 variable "vsphere_password" {} #vsphereのパスワード
4 variable "vcenter_server" {} #vcenterのFQDN/IPアドレス
5 variable "vsphere_datacenter" {} #vsphereのデータセンター
6 variable "vsphere_datastore" {} #vsphereのデータストア
7 variable "vsphere_cluster" {} #vsphereのクラスター
8 variable "vsphere_network" {} #vsphereのネットワーク
9
10 #Define Variables for Virtual Machines
11 variable "vsphere_template_name" {} #プロビジョニングするテンプレート
12 variable "prov_vm_num" {} #プロビジョニングする仮想マシンの数
13 variable "prov_vmname_prefix" {} #プロビジョニングする仮想マシンの接頭辞
14 variable "prov_cpu_num" {} #プロビジョニングする仮想マシンのCPUの数
15 variable "prov_mem_num" {} #プロビジョニングする仮想マシンのメモリの数
16
17
18 #Provider
19 provider "vsphere" {
20     user = var.vsphere_user
21     password = var.vsphere_password
22     vcenter_server = var.vcenter_server
23     allow_unverified_ssl = true
24 }
25
26 #Data
27 data "vsphere_datacenter" "dc" {
28     name = var.vsphere_datacenter
29 }
30
31 data "vsphere_datastore" "datastore" {
32     name = var.vsphere_datastore
33     datacenter_id = data.vsphere_datacenter.dc.id
34 }
35
36 data "vsphere_compute_cluster" "cluster" {
37     name = var.vsphere_cluster
38     datacenter_id = data.vsphere_datacenter.dc.id
39 }
40
41 data "vsphere_network" "network" {
42     name = var.vsphere_network
43     datacenter_id = data.vsphere_datacenter.dc.id
44 }
45
46 data "vsphere_virtual_machine" "template" {
47     name = var.vsphere_template_name
48     datacenter_id = data.vsphere_datacenter.dc.id
49 }
50
51 #Resource
52 resource "vsphere_virtual_machine" "vm" {
53     count = var.prov_vm_num
54     name = "${var.prov_vmname_prefix}${format("%03d", count.index+1)}"
55     resource_pool_id = data.vsphere_compute_cluster.cluster.resource_pool_id
56     datastore_id = data.vsphere_datastore.datastore.id
57 }
58 #Resource for VM Specs
59 num_cpus = var.prov_cpu_num
60 memory = var.prov_mem_num
61 guest_id = data.vsphere_virtual_machine.template.guest_id
62 scsi_type = data.vsphere_virtual_machine.template.scsi_type
63
64 network_interface {
65     network_id = data.vsphere_network.network.id
66     adapter_type = "vmxnet3"
67 }
68
69 #Resource for Disks
70 disk {
71     label = "disk1"
72     size = data.vsphere_virtual_machine.template.disks.0.size
73     eagerly_scrub = data.vsphere_virtual_machine.template.disks.0.eagerly_scrub
74     thin_provisioned = data.vsphere_virtual_machine.template.disks.0.thin_provisioned
75 }
76
77 clone {
78     template_uuid = data.vsphere_virtual_machine.template.id
79     customize {
80         windows_options {
81             computer_name = "${var.prov_vmname_prefix}${format("%03d", count.index+1)}"
82             network_interface {} #テンプレートと同一のNIC(複数)
83         }
84     }
85 }
86
87 }
```

data : 既存のリソースから情報を取り込む

変数で指定したオブジェクトから情報を取り込み、プロビジョニングに利用できるようにする。

この例では、指定した仮想マシンテンプレートの情報を取り込み、OSタイプや仮想ハードウェアの情報が Terraform 側で利用できるようになっている。

```
data "vsphere_virtual_machine" "template" {
    name = var.vsphere_template_name
    datacenter_id = data.vsphere_datacenter.dc.id
}
```

Terraform のコード解説 : main.tf

```
87 lines (73 sloc) | 2.95 KB
1 #Define Variables for Platform
2 variable "vsphere_user" {} #vsphereのユーザー名
3 variable "vsphere_password" {} #vsphereのパスワード
4 variable "vcenter_server" {} #vcenterのFQDN/IPアドレス
5 variable "vsphere_datacenter" {} #vsphereのデータセンター
6 variable "vsphere_datastore" {} #vsphereのデータストア
7 variable "vsphere_cluster" {} #vsphereのクラスター
8 variable "vsphere_network" {} #vsphereのネットワーク
9
10 #Define Variables for Virtual Machines
11 variable "vsphere_template_name" {} #プロビジョニングするテンプレート
12 variable "prov_vm_num" {} #プロビジョニングする仮想マシンの数
13 variable "prov_vmname_prefix" {} #プロビジョニングする仮想マシンの接頭辞
14 variable "prov_cpu_num" {} #プロビジョニングする仮想マシンのCPUの数
15 variable "prov_mem_num" {} #プロビジョニングする仮想マシンのメモリのMB
16
17
18 #Provider
19 provider "vsphere" {
20   user = var.vsphere_user
21   password = var.vsphere_password
22   vcenter_server = var.vcenter_server
23   allow_unverified_ssl = true
24 }
25
26 #Data
27 data "vsphere_datacenter" "dc" {
28   name = var.vsphere_datacenter
29 }
30
31 data "vsphere_datastore" "datastore" {
32   name = var.vsphere_datastore
33   datacenter_id = data.vsphere_datacenter.dc.id
34 }
35
36 data "vsphere_compute_cluster" "cluster" {
37   name = var.vsphere_cluster
38   datacenter_id = data.vsphere_datacenter.dc.id
39 }
40
41 data "vsphere_network" "network" {
42   name = var.vsphere_network
43   datacenter_id = data.vsphere_datacenter.dc.id
44 }
45
46 data "vsphere_virtual_machine" "template" {
47   name = var.vsphere_template_name
48   datacenter_id = data.vsphere_datacenter.dc.id
49 }
50
51 #Resource
52 resource "vsphere_virtual_machine" "vm" {
53   count = var.prov_vm_num
54   name = "${var.prov_vmname_prefix}${format("%03d",count.index+1)}"
55   resource_pool_id = data.vsphere_compute_cluster.cluster.resource_pool_id
56   datastore_id = data.vsphere_datastore.datastore.id
57 }
58 #Resource for VM Specs
59 num_cpus = var.prov_cpu_num
60 memory = var.prov_mem_num
61 guest_id = data.vsphere_virtual_machine.template.guest_id
62 scsi_type = data.vsphere_virtual_machine.template.scsi_type
63
64 network_interface {
65   network_id = data.vsphere_network.network.id
66   adapter_type = "vmxnet3"
67 }
68
69 #Resource for Disks
70 disk {
71   label = "disk1"
72   size = data.vsphere_virtual_machine.template.disks.0.size
73   eagerly_scrub = data.vsphere_virtual_machine.template.disks.0.eagerly_scrub
74   thin_provisioned = data.vsphere_virtual_machine.template.disks.0.thin_provisioned
75 }
76
77 clone {
78   template_uuid = data.vsphere_virtual_machine.template.id
79   customize {
80     windows_options {
81       computer_name = "${var.prov_vmname_prefix}${format("%03d",count.index+1)}"
82     }
83     network_interface {} #テンプレートと同一のNIC(複数)
84   }
85 }
86
87 }
```

resource : 実際に作成するリソースを定義

変数で指定した情報や data で取り込んだ情報を利用して、対象となるインフラにリソースをプロビジョニング(作成)する。

当然のことながら、HCLはプログラミング言語なので、演算子などもきちんと使える。

ここでは変数 "count" に応じて仮想マシンの名前を1から指定した数字まで、前ゼロ3桁でプロビジョニングするような仕組みになっている。

#Resource

```
resource "vsphere_virtual_machine" "vm" {
  count = var.prov_vm_num
  name = "${var.prov_vmname_prefix}${format("%03d",count.index+1)}"
  resource_pool_id = data.vsphere_compute_cluster.cluster.resource_pool_id
  datastore_id = data.vsphere_datastore.datastore.id
}
```

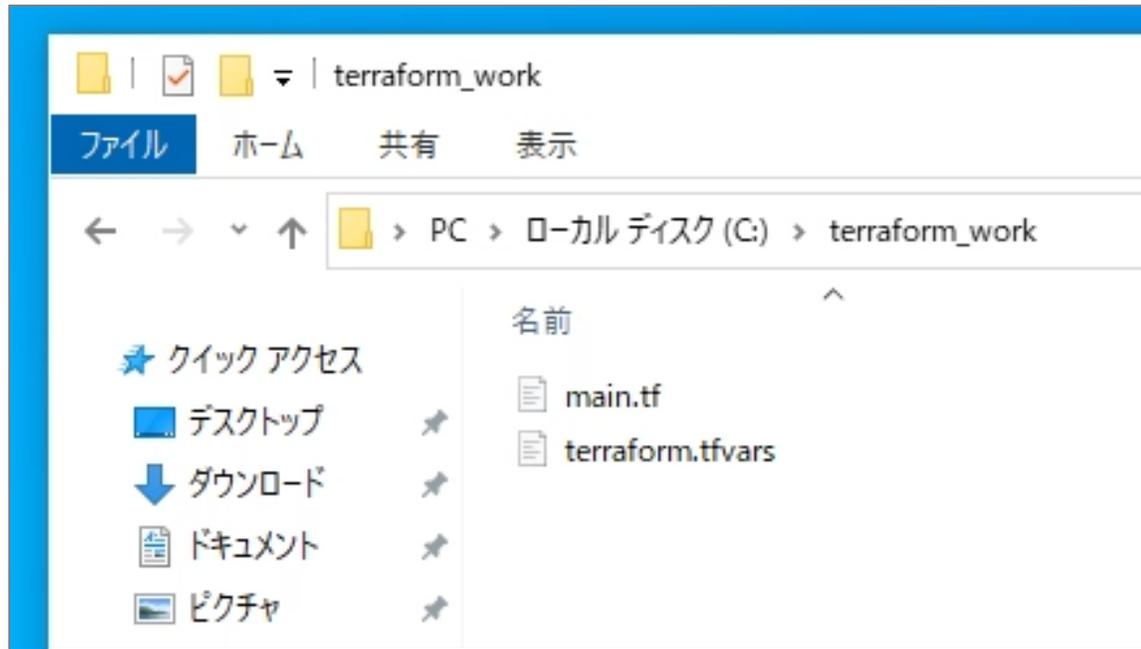
Terraform のコード解説 : terraform.tfvars

パスワードやアクセスキーのお漏らしには注意しましょう!!

```
1  #Define Variables for Platform
2  vsphere_user      = "Administrator@vsphere.local" #vsphereのユーザー名
3  vsphere_password  = "Netw0rld123!"                #vsphereのパスワード
4  vCenter_server    = "vc01.vsphere.local"          #vCenterのFQDN/IPアドレス
5  vsphere_datacenter = "Datacenter"                 #vsphereのデータセンター
6  vsphere_datastore = "datastore1"                  #vsphereのデータストア
7  vsphere_cluster   = "Cluster"                     #vsphereのクラスター
8  vsphere_network   = "VM Network"                  #vsphereのネットワーク
9
10 #Define Variables for Virtual Machines
11 vsphere_template_name = "WIN2019"                  #プロビジョニングするテンプレート
12 prov_vm_num           = 1                          #プロビジョニングする仮想マシンの数
13 prov_vmname_prefix    = "TFVM"                    #プロビジョニングする仮想マシンの接頭語
14 prov_cpu_num          = 2                          #プロビジョニングする仮想マシンのCPUの数
15 prov_mem_num          = 4096                       #プロビジョニングする仮想マシンのメモリのMB
```

[STEP2] Terraform INIT

フォルダーにコードを配置



コマンドラインで Terraform INIT

```
管理: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19041.264]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>cd c:\terraform_work

c:\terraform_work>set HTTP_PROXY=http://172.16.24.250:8080

c:\terraform_work>set NO_PROXY=vc01.vsphere.local

c:\terraform_work>terraform init
Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "vsphere" (hashicorp/vsphere) 1.18.3...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "... constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.vsphere: version = "" > 1.18"

Terraform has been successfully initialized!

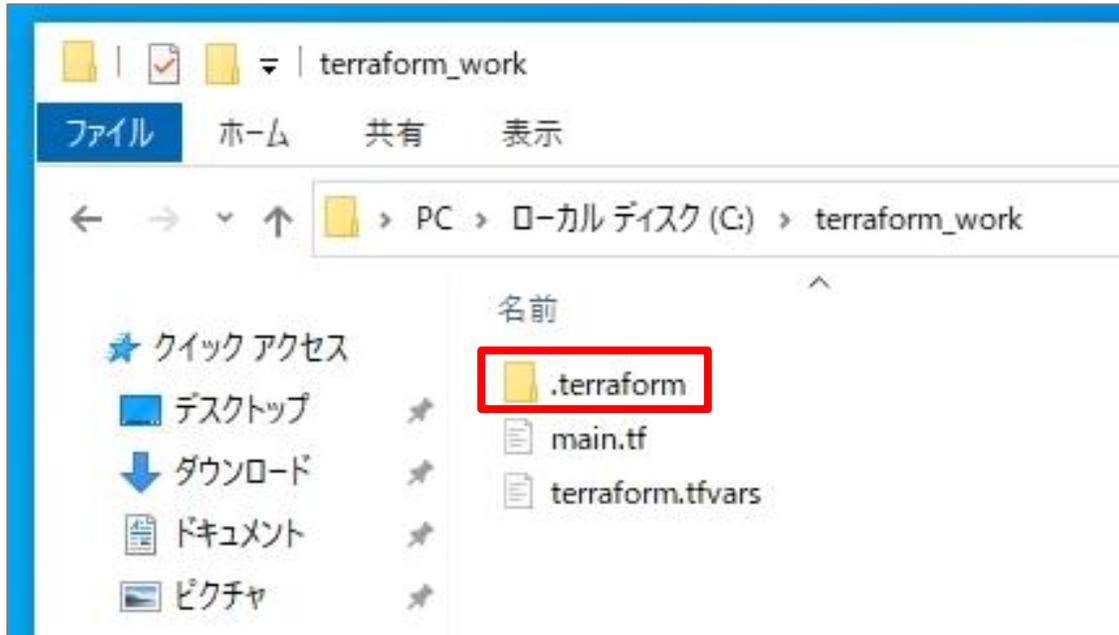
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

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.

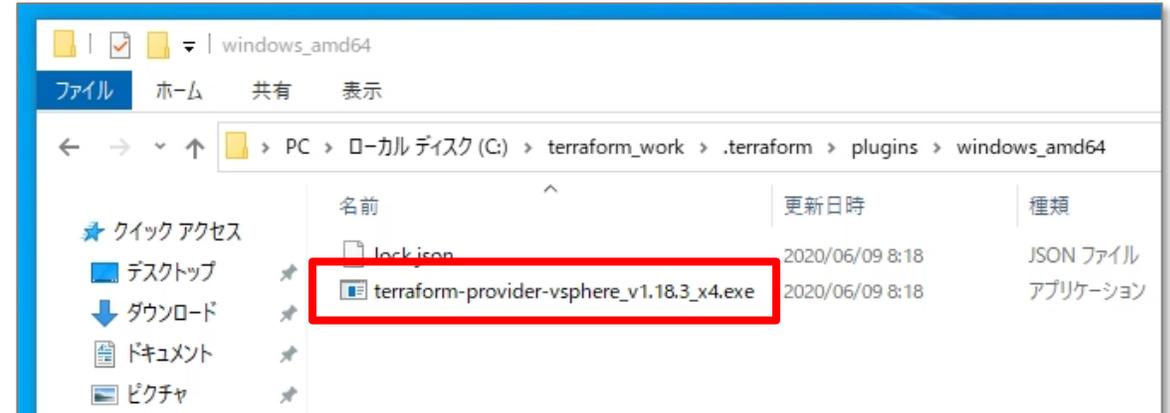
c:\terraform_work>
```

[STEP2] Terraform INIT

フォルダーにプロバイダーが
ダウンロードされる

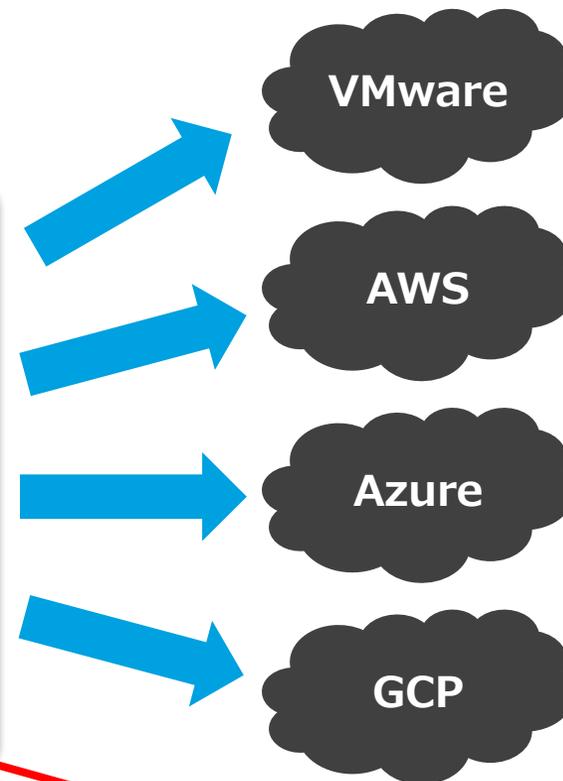


プロバイダーの実体が
配置されていることがわかる



Terraform のプロバイダーとは？

プロバイダーの役割



【ザックリ理解!!】
インフラの特性や固有の依存関係なども整理して、各サービスやプロダクトのAPIへ翻訳・変換してくれる

[STEP3] Terraform PLAN

Terraform PLAN でコードの検証を実施する

```
管理: C:\Windows\system32\cmd.exe
c:\terraform_work> terraform plan
Refreshing Terraform state in memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

data.vsphere_datacenter.dc: Refreshing state...
data.vsphere_datastore.datastore: Refreshing state...
data.vsphere_compute_cluster.cluster: Refreshing state...
data.vsphere_network.network: Refreshing state...
data.vsphere_virtual_machine.template: Refreshing state...

-----

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# vsphere_virtual_machine.vm[0] will be created
+ resource "vsphere_virtual_machine" "vm" {
  boot_retry_delay = 10000
  change_version   = (known after apply)
  cpu_limit        = -1
  cpu_share_count  = (known after apply)
  cpu_share_level  = normal
  datastore_id     = "datastore-34"
  default_ip_address = (known after apply)
  ept_rvi_mode     = automatic
  firmware         = bios
  force_power_off  = true
  guest_id         = "windows9Server64Guest"
  guest_ip_addresses = (known after apply)
  hardware_version = (known after apply)
  host_system_id   = (known after apply)
  hv_mode          = hvAuto
  id               = (known after apply)
}
```

コードに不備があるとエラーが表示される

```
管理: C:\Windows\system32\cmd.exe
c:\terraform_work> terraform plan
var.vsphere_template_name
Enter a value: aa

Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

data.vsphere_datacenter.dc: Refreshing state...
data.vsphere_datastore.datastore: Refreshing state...
data.vsphere_compute_cluster.cluster: Refreshing state...
data.vsphere_virtual_machine.template: Refreshing state...
data.vsphere_network.network: Refreshing state...

Error: error fetching virtual machine: vm 'aa' not found

on main.tf line 46, in data "vsphere_virtual_machine" "template":
46: data "vsphere_virtual_machine" "template" {

c:\terraform_work>
```

[STEP4] Terraform APPLY

Terraform APPLY でコードをインフラに適用する

```
管理: C:\Windows\system32\cmd.exe - terraform apply
c:\terraform_wor>terraform apply
data.vsphere_datacenter.dc: Refreshing state...
data.vsphere_virtual_machine.template: Refreshing state...
data.vsphere_compute_cluster.cluster: Refreshing state...
data.vsphere_datastore.datastore: Refreshing state...
data.vsphere_network.network: Refreshing state...

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# vsphere_virtual_machine.vm[0] will be created
+ resource vsphere_virtual_machine vm {
  + boot_retry_delay      = 10000
  + change_version        = (known after apply)
  + cpu_limit              = -1
  + cpu_share_count       = (known after apply)
  + cpu_share_level       = "normal"
  + datastore_id           = "datastore-34"
  + default_ip_address    = (known after apply)
  + ept_rvi_mode           = "automatic"
  + firmware               = "bios"
  + force_power_off       = true
  + guest_id               = "windows9Server64Guest"
  + guest_ip_addresses    = (known after apply)
  + hardware_version       = (known after apply)
  + host_system_id         = (known after apply)
  + hv_mode                = "hvAuto"
  + id                     = (known after apply)
  + imported               = (known after apply)
  + latency_sensitivity    = "normal"
  + memory                 = 4096
  + memory_limit           = -1
  + memory_share_count     = (known after apply)
  + memory_share_level     = "normal"
  + migrate_wait_timeout   = 30
}
```

プロビジョニングされるリソースを確認し、
問題がなければ“YES”で適用を開始する

```
管理: C:\Windows\system32\cmd.exe - terraform apply
+ uuid          = (known after apply)
+ write_through = false
}

+ network_interface {
  + adapter_type      = "vmxnet3"
  + bandwidth_limit   = -1
  + bandwidth_reservation = 0
  + bandwidth_share_count = (known after apply)
  + bandwidth_share_level = "normal"
  + device_address     = (known after apply)
  + key                 = (known after apply)
  + mac_address        = (known after apply)
  + network_id         = "network-30"
}

Plan: 1 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_
```



Terraform のデモ (動画)

デモのお品書き

**Terraform (OSS版) をダウンロードして、
Windows 10 に初期設定する**

GitHub からコードをコピーしてくる

**vSphere 6.7u1 の環境で テンプレートを使って
Windows Server を 1台 プロビジョニングする**
(仮想マシンの名前は TFVM001 とする)

デモ : OSS版 Terraform と vSphere の組み合わせ

- <https://www.youtube.com/watch?v=8EhO1ocX2zw>

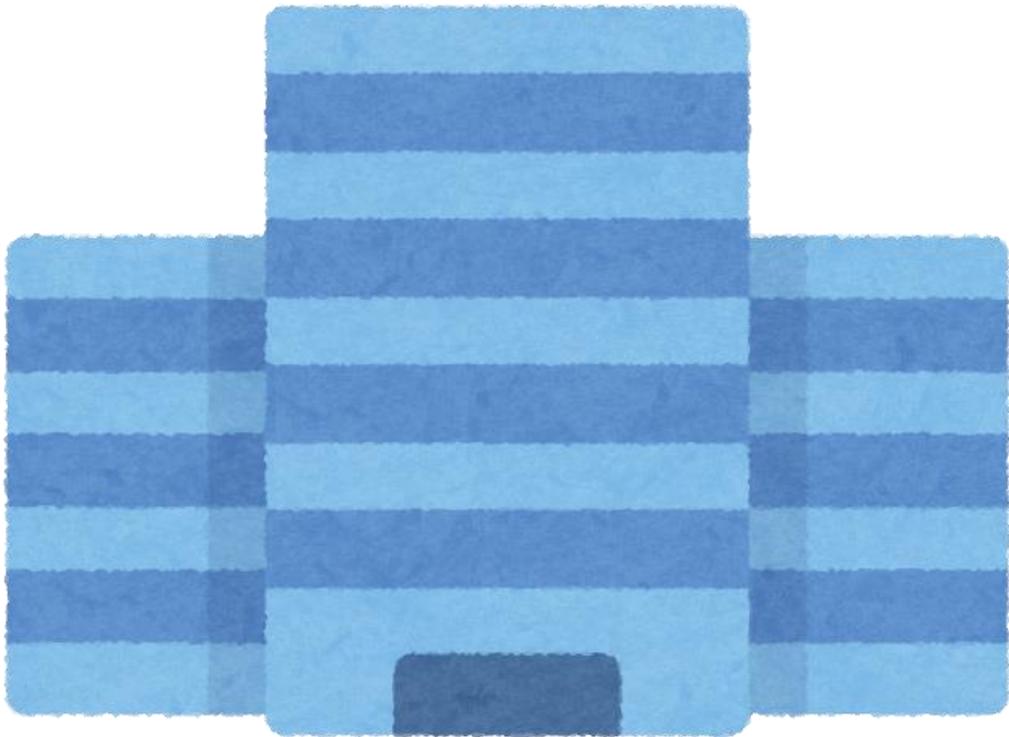




Terraform Enterprise とは？

Terraform Enterprise とは？

有償版の Terraform のひとつです!!



こんなお客様へ!!

Terraform を導入したいけど…

- サポートがほしい
- チームで取り組みたい
(属人性をできる限りなくしたい)
- 管理・監査が必要
(ヒヤリハットを防ぎたい)
- オンプレミスでも使いたい

VMware も無償版があるけど、一般的には本番では使われない

Google search results for "vmware hypervisor 無償" (vmware hypervisor free).

約 98 件 (0.45 秒)

www.vmware.com > products > vsphere-hypervisor > **無償の vSphere Hypervisor | VMware |**
サーバを仮想化するベアメタルハイパーバイザーで IT で、時間とコストを削減できます。無償の vSphere Hypervisor、およびサポートにおいて業界標準を確立している世界最小かつ堅牢なアーキテクチャで

blogs.vmware.com > 2014/05 > vspherehypervisor > **やってみよう！初めての無償版ESXi -**
2014/05/26 - というご質問をよく受けるのですが、ESXi 仮想サーバ構築のベースを提供する製品、vSphereはESXi製品です。まずは無償のVMware vSphere Hypervisor

www.vmware.com > JP-get-esxi > **VMware vSphere Hypervisor の無償ダウンロード**
無償の VMware vSphere Hypervisor を使用して仮想化アプリケーションを統合することで、ハードウェア、電力、削減します。
このページに 2 回アクセスしています。前回のアクセス

vmware®
クラウド ソリューション 製品 サポート & サービス ダウンロード パートナー 企業情報

製品 > vSphere Hypervisor

vSphere Hypervisor

お問い合わせ >
今すぐダウンロード

概要 スポットライト 導入方法 リソース

少ないハードウェアで多くを実現する vSphere Hypervisor

アプリケーションを統合できます。サーバを仮想化するベアメタルハイパーバイザーで IT インフラストラクチャを管理することで、時間とコストを削減できます。無償の vSphere Hypervisor は、信頼性、パフォーマンス、およびサポートにおいて業界標準を確立している世界最小かつ堅牢なアーキテクチャである VMware vSphere ESXi を基盤としています。



VMware vSphere Hypervisor の概要 (2:16)
今日の IT コストの節約に役立つ、本番環境対応の使いやす...

vSphere の無償評価
vSphere 6.7 は新たなレベルの仮想化を実現します。より深い...
今すぐ確認 >

その前に...

Terraform Cloud ってのもある

The screenshot shows a news article on the website atmarkit.co.jp. The article title is "HashiCorpが「Terraform Cloud」のフルリリースを発表" (HashiCorp announces full release of Terraform Cloud). The text describes the release of Terraform Cloud as the first step in 2019, highlighting the addition of Remote State Management and expansion capabilities. It mentions the release date as September 10, 2019. The article includes social media sharing buttons for print, Twitter (10 shares), Facebook, and a bookmark icon. At the bottom, there is a large HashiCorp Terraform logo.

The screenshot shows the login page for Terraform Cloud at app.terraform.io/session. The page features the HashiCorp Terraform Cloud logo at the top. Below the logo, it says "Sign in to Terraform Cloud". There are two input fields: "Username or email" and "Password". A "Sign in" button is located below the password field. There is a link for "Forgot password?". At the bottom, there is a link to "Create your free account" and a note to "View Terraform Offerings to find out which one is right for you." The footer contains the copyright notice: "© 2020 HashiCorp, Inc. Support Terms Privacy Security".

その前に...

Terraform Cloud ってのもある

 Terraform Cloud Get started with our Cloud Infrastructure Automation as a Service	Free Collaborate on infrastructure as code for Terraform configurations.	Team & Governance Collaborate on infrastructure as code, manage users, and enforce provisioning policies.	Business Everything organizations need to use Terraform at scale.
Cloud Pricing	\$0 up to 5 users	Starting at \$20 user/month	Contact Sales

<https://www.hashicorp.com/products/terraform/pricing/>

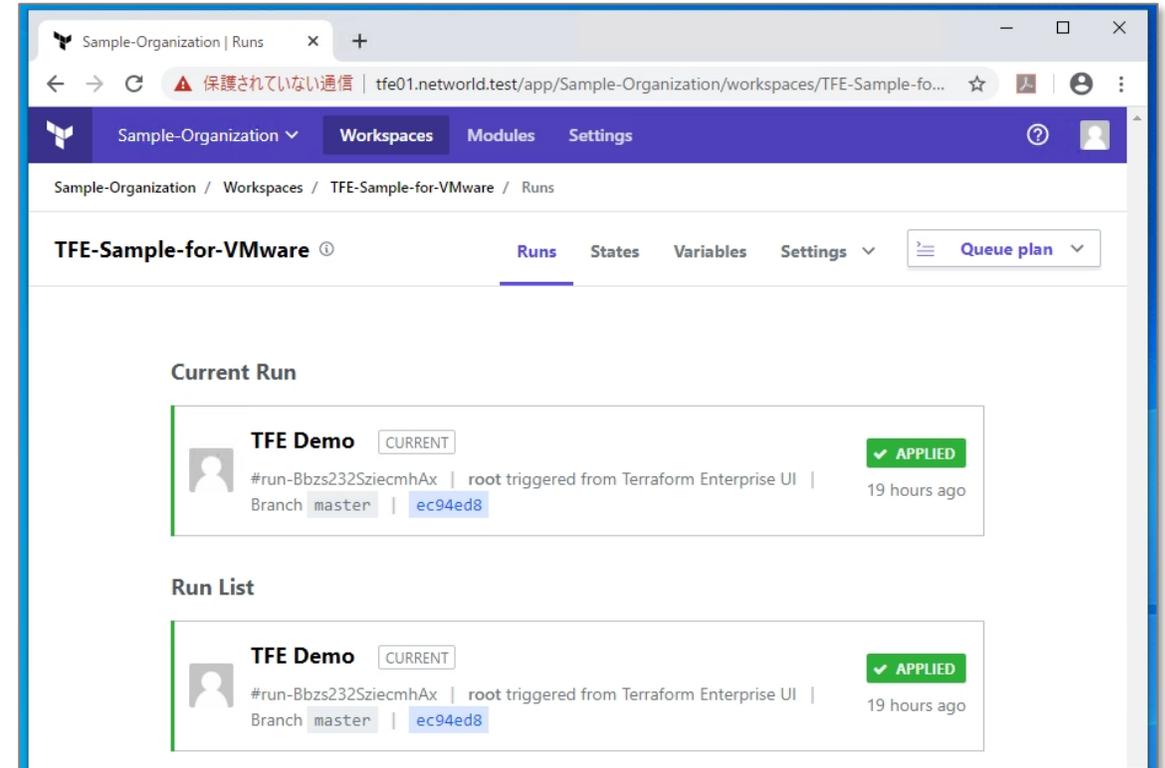
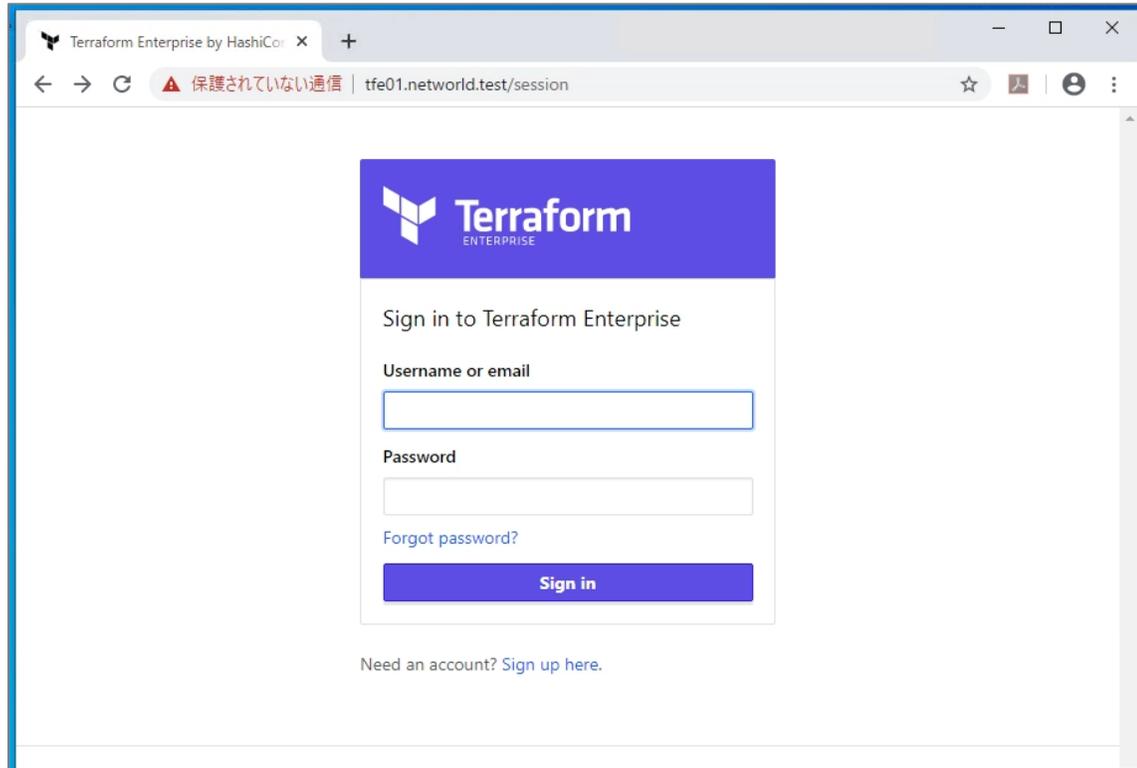
【これが難しい!!】 無償・有償 と オンプレミス・クラウド

	オンプレミス	クラウド
無償	Terraform (OSS) <ul style="list-style-type: none">• CUI のみ• ローカルの端末で実行• オンプレでもパブリックでもOK	Terraform Cloud (Free) <ul style="list-style-type: none">• GUI あり• HashiCorp による SaaS• オンプレミス利用NG
有償	Terraform Enterprise <ul style="list-style-type: none">• GUI あり• お客様自身によるホスティング• オンプレでもパブリックでもOK• チーム管理&ポリシー機能あり• エンタープライズ向け機能あり (SAMLや監査ログなど)	Terraform Cloud Business <ul style="list-style-type: none">• GUI あり• HashiCorp による SaaS• オンプレでもパブリックでもOK• チーム管理&ポリシー機能あり• エンタープライズ向け機能あり (SAMLや監査ログなど)

この資料はコレについての説明です!!



Terraform Enterprise (TFE) の GUI



OSS と TFE の機能の違い (弊社製品紹介ページより)

カテゴリ	機能	OSS	Enterprise
Infrastructure as Code	HCL による Infrastructure as Code (HashiCorp Configuration Language)	✓	✓
	ワークスペース	✓	✓
	変数の設定	✓	✓
	実行 (PlanとApply)	✓	✓
	依存関係の可視化	✓	✓
	プロバイダー	✓	✓
	モジュール	✓	✓
	Public Module Registry	✓	✓

OSS と TFE の機能の違い (弊社製品紹介ページより)

カテゴリ	機能	OSS	Enterprise
コラボレーション	インフラ状態をリモートに保存	—	✓
	VCS連携	—	✓
	ワークスペース管理	—	✓
	セキュアな変数管理	—	✓
	リモート実行	—	✓
	Private Module Registry	—	✓
	Full API Coverage	—	✓
	チーム管理	—	✓
	SAMLによるSSO	—	✓

OSS と TFE の機能の違い (弊社製品紹介ページより)

カテゴリ	機能	OSS	Enterprise
ガバナンスとポリシー	Sentinel によるポリシー管理	—	✓
	費用予測	—	✓
	監査ログ	—	✓
インフラのセルフサービス化	Configuration Designer	—	✓
	ServiceNow	—	✓
運用	高可用性クラスタアーキテクチャ	—	✓
	パフォーマンススケーリング	—	✓
	オンプレミスネットワーク接続	—	✓
	オンプレミス構築	—	✓
サポート	英語(メーカー)サポート	—	✓
	日本語サポート ※オプション	—	✓

TFE注目機能：VCS と連携したワークスペース

VCS とは
Version Control System のこと
(例えば、GitHubとかGitLabとか…)

Sample-Organization | New Workspace

Sample-Organization / Workspaces

Create a new Workspace

Workspaces allow you to organize infrastructure and collaborate on Terraform runs.

- 1 Connect to VCS
- 2 Choose a repository
- 3 Configure settings

Connect to a version control provider

Choose the version control provider that hosts the Terraform configuration for this workspace.

GitHub GitLab Bitbucket Azure DevOps

Use an existing VCS connection

If you only plan to use Terraform CLI or the API to perform runs in this workspace, you don't need to connect it to version control. You can add a VCS connection later if you change your mind.

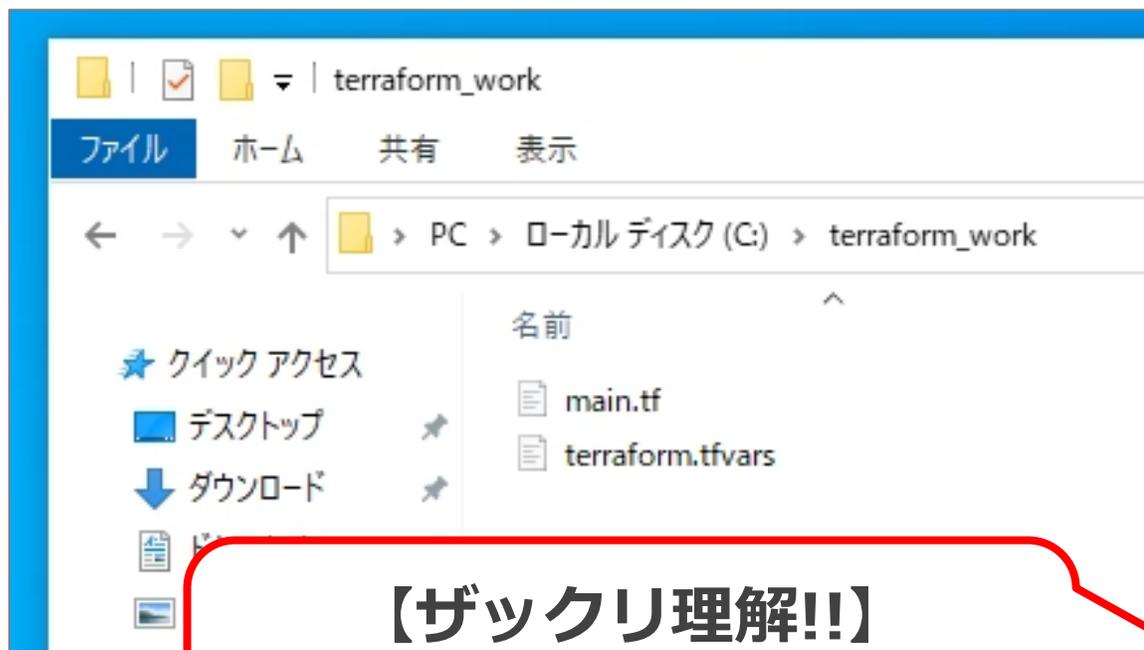
No VCS connection
Requires CLI or API

【ザックリ理解!!】

ひとつのワークスペースは
ひとつの VCS リポジトリに紐付く

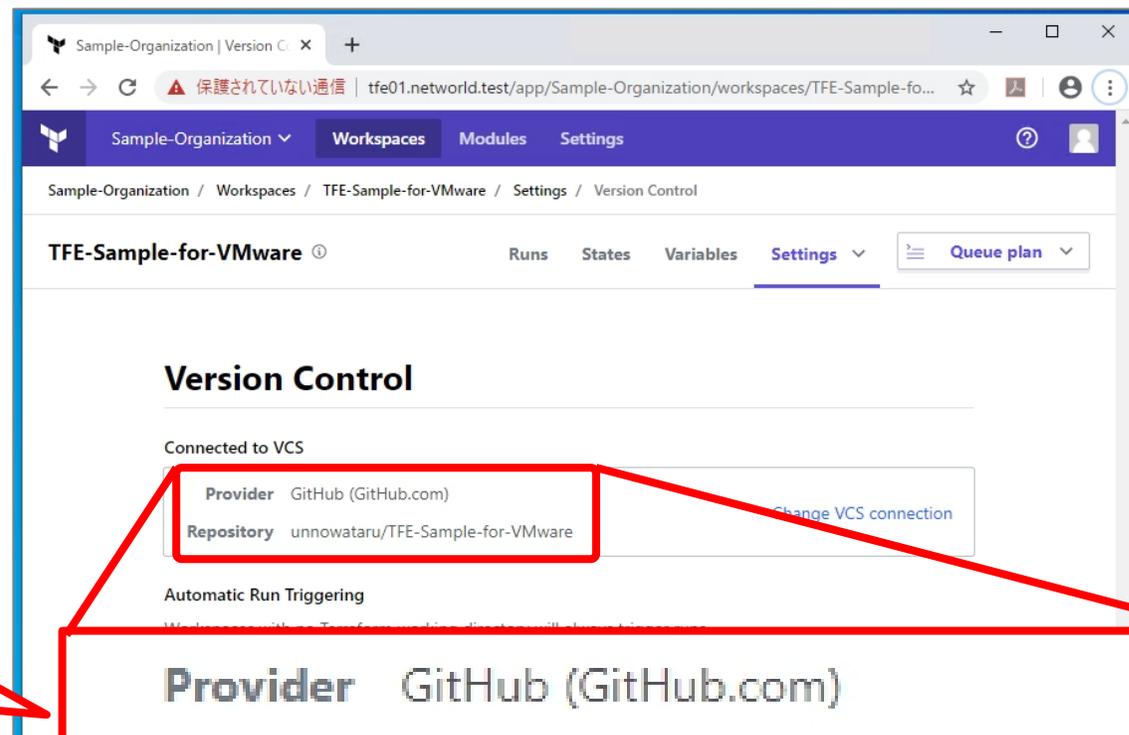
TFE注目機能：VCS と連携したワークスペース

ローカルにコードを
配置する代わりに...



【ザックリ理解!!】
ここからコードを取得して、
Terraform のサーバーに格納する

ワークスペースに VCS の
リポジトリを紐付ける



Provider GitHub (GitHub.com)
Repository unnowataru/TFE-Sample-for-VMware

TFE注目機能 : Secure Variables

Sample-Organization | Variables

Variables

These variables are used for all plans and applies in this workspace. Workspaces using Terraform 0.10.0 or later can also load default values from any *.auto.tfvars files in the configuration.

Sensitive variables are hidden from view in the UI and API, and can't be edited. (To change a sensitive variable, delete and replace it.) Sensitive variables can still appear in Terraform logs if your configuration is designed to output them.

When setting many variables at once, the Terraform Enterprise Provider or the variables API can often save time.

Terraform Variables

These Terraform variables are set up in your configuration. To set a value for a variable, click its HCL definition.

Key	Value	...
vsphere_user	Administrator@vsphere.local	...
vsphere_password	Sensitive - write only	...
vCenter_server	vc01.vsphere.local	...
vsphere_datacenter	Datacenter	...
vsphere_datastore	datastore1	...

【ザックリ理解!!】
GUIに入力したパスワードはもう二度と読み取ることはできない!!

vsphere_password

SENSITIVE

Sensitive - write only

vsphere_password

Sensitive - write only

SENSITIVE

vCenter_server

vc01.vsphere.local

vsphere_datacenter

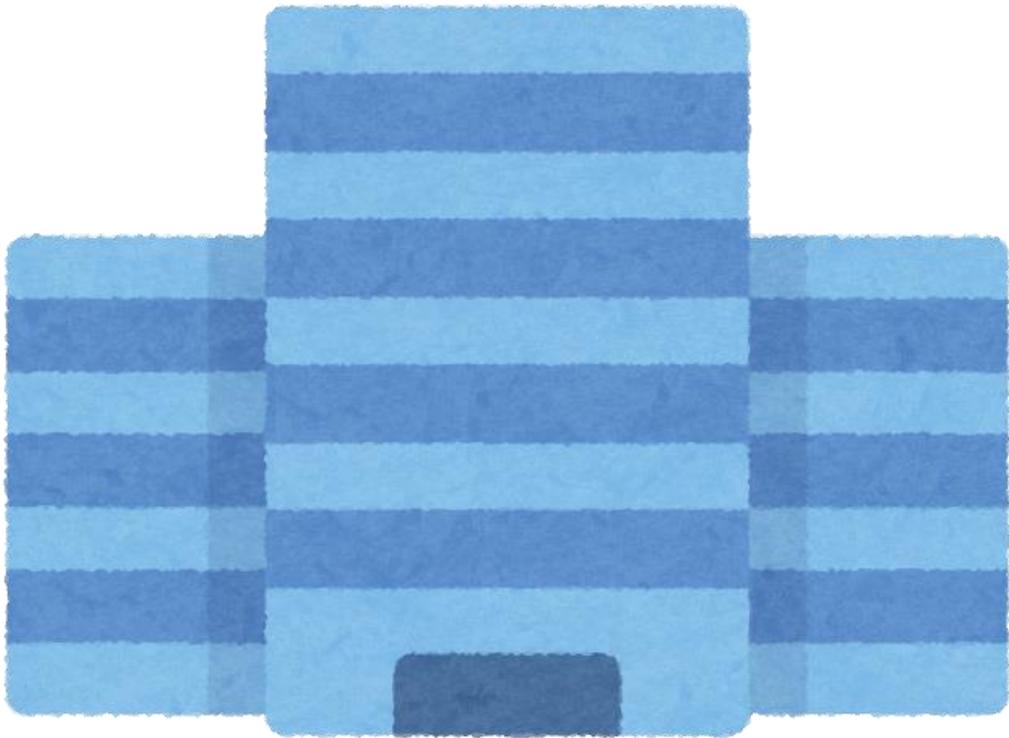
Datacenter

vsphere_datastore

datastore1

TFE注目機能：Sentinel とは？

HashiCorp 製品のひとつで Terraform に統合されています



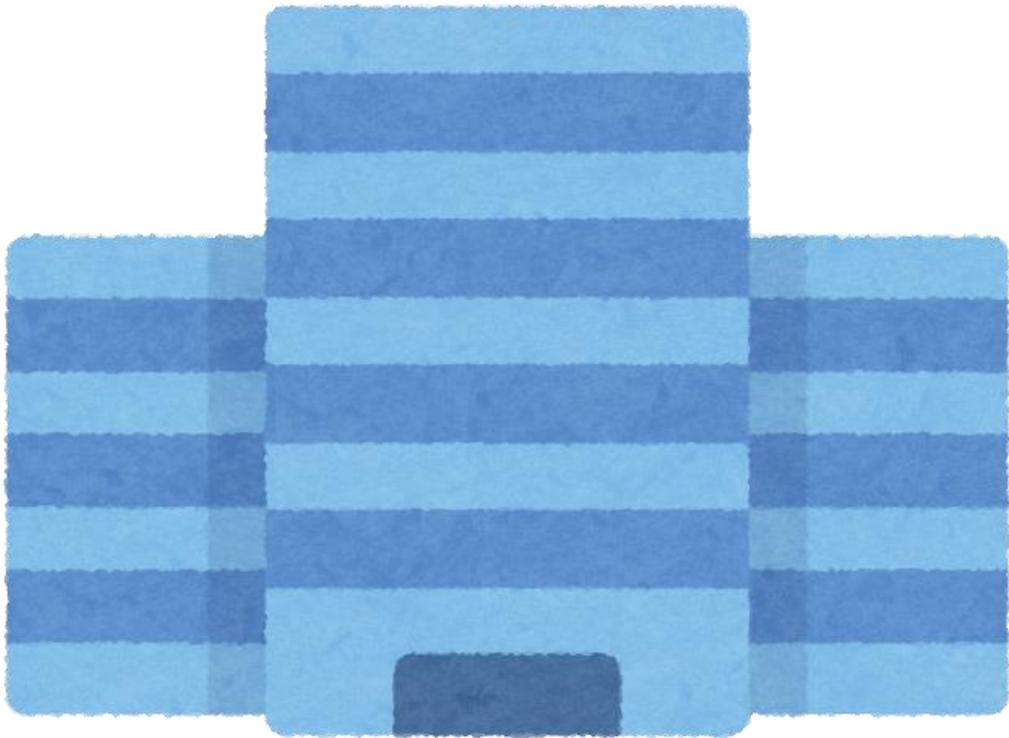
こんなお客様へ!!

Terraform を導入したいけど...

- サポートがほしい
- チームで取り組みたい
(属人性をできる限りなくしたい)
- 管理・監査が必要
(ヒヤリハットを防ぎたい)

TFE注目機能：Sentinel とは？

HashiCorp 製品のひとつで Terraform に統合されています



こんなお客様へ!!

Terraform を導入したいけど...

- サポート
 - チーム (属)
 - 管理・監査が必要 (ヒヤリハットを防ぎたい)
- 【ザックリ理解!!】**
特にこの部分に有効で、
いろいろできちゃう Terraform で
事故が起きないように**ポリシー設定**できる

TFE注目機能 : Sentinel

Policy as Code : ポリシーをコード化し、プロビジョニングするときに自動的にルールが適用されているか確認。

Version Control : ポリシー作成やバージョン管理、コードのレビューなどを既存の VCS ベースの手法で実現。

Policy Sets : 複数のポリシーをグループ化し、複数の異なる環境に対し、独自のルールの設定が可能。

Testing : Sentinel Simulator により、カンタンにポリシーのテストや開発が可能。

Sentinel にはどんなポリシーがあるのか？

VMware 向けのサンプルだと...

VMware

- Require Storage DRS to be enabled
- Restrict virtual disk size
- Restrict VM CPU count and memory
- Enforce NFS 4.1 and Kerberos

Azure 向けのサンプルだと...

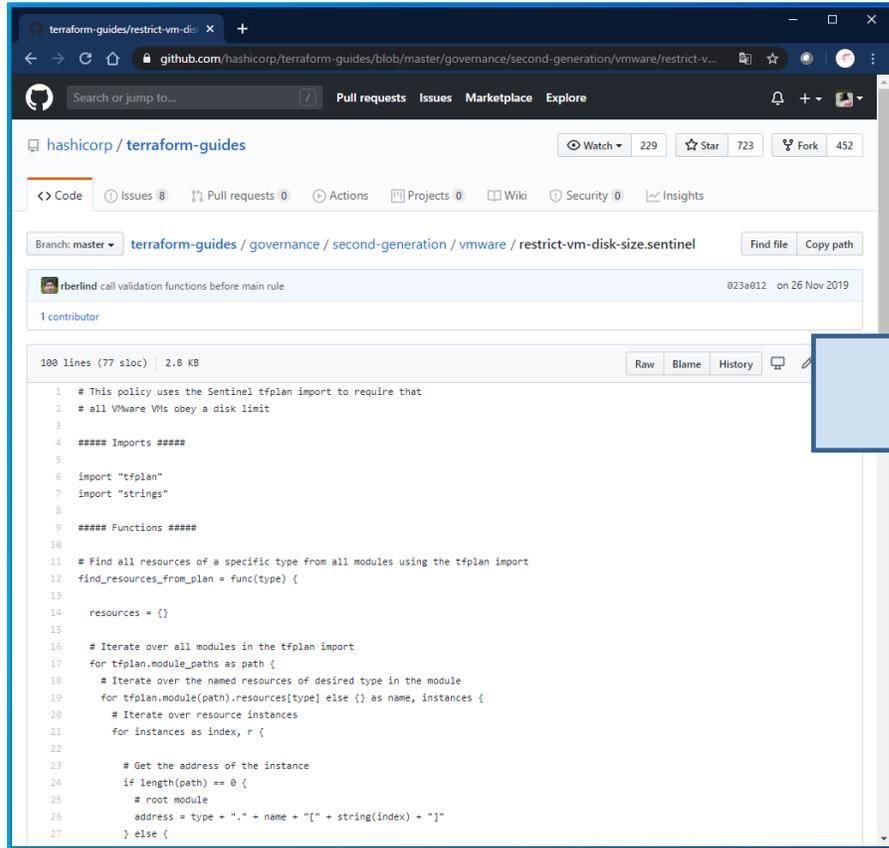
Microsoft Azure

- Restrict VM images
- Restrict the size of Azure VMs
- Enforce limits on AKS clusters

<https://www.terraform.io/docs/cloud/sentinel/examples.html>

Sentinelはどうやって設定するのか？

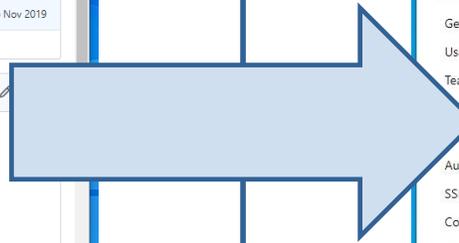
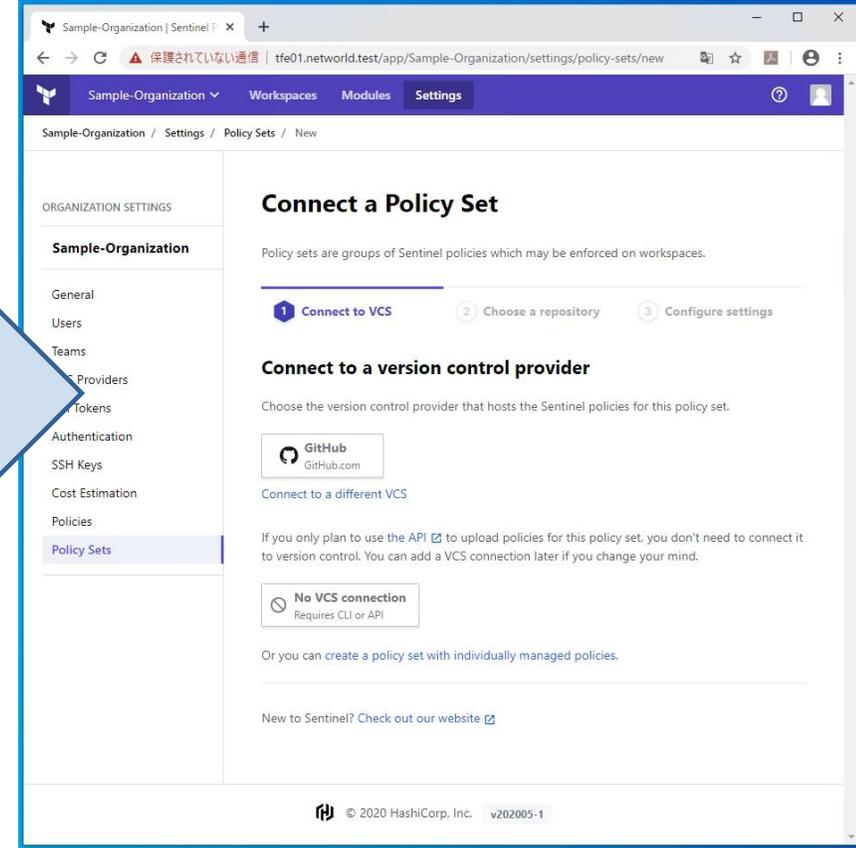
VCS にコードを作成して…



A screenshot of a GitHub repository page for the file `restrict-vm-disk-size.sentinel`. The file content is as follows:

```
1 # This policy uses the Sentinel tfplan import to require that
2 # all VMware VMs obey a disk limit
3
4 ##### Imports #####
5
6 import "tfplan"
7 import "strings"
8
9 ##### Functions #####
10
11 # Find all resources of a specific type from all modules using the tfplan import
12 find_resources_from_plan = func(type) {
13
14     resources = {}
15
16     # Iterate over all modules in the tfplan import
17     for tfplan.module_paths as path {
18         # Iterate over the named resources of desired type in the module
19         for tfplan.module(path).resources[type] else {} as name, instances {
20             # Iterate over resource instances
21             for instances as index, r {
22
23                 # Get the address of the instance
24                 if length(path) == 0 {
25                     # root module
26                     address = type + "." + name + "[" + string(index) + "]"
27                 } else {
```

リポジトリからポリシーを設定



<https://github.com/hashicorp/terraform-guides/blob/master/governance/second-generation/vmware/restrict-vm-disk-size.sentinel>

TFE注目機能：オンプレミスでの構築・利用

curl <https://install.terraform.io/pfte/stable> | sudo bash



```
192.168.100.21 - root@tfe01:~ VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
[root@tfe01 ~]# curl https://install.terraform.io/pfte/stable | sudo bash
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %         Dload  Upload  Total  Spent    Left  Speed
100 135k 100 135k  0     0 44321      0  0:00:03  0:00:03 --:--:-- 44315
Determining local address
The installer will use network interface 'ens192' (with IP address '192.168.100.21')
Determining service address
The installer was unable to automatically detect the service IP address of this machine.
Please enter the address or leave blank for unspecified.
Service IP address: 192.168.100.21
Does this machine require a proxy to access the Internet? (y/N) y
Enter desired HTTP proxy address: http://172.16.24.2:8080
The installer will use the proxy at 'http://172.16.24.2:8080'
Installing docker version 19.03.8 from https://get.docker.com
# Executing docker install script, commit: b50c66738e923476ebc137275446e106860ae08c
+ sh -c 'yum install -y -q yum-utils'
```

【ザックリ理解!!】

最小構成なら このコマンド一発で
ほとんどインストールできる

TFE注目機能：オンプレミスでの構築・利用

初期セットアップと管理コンソールはこんな感じ

【ザックリ理解!!】
<https://<TFEのサーバー>:8800> で
管理コンソールに接続できるぞ

TFE注目機能：オンプレミスでの構築・利用

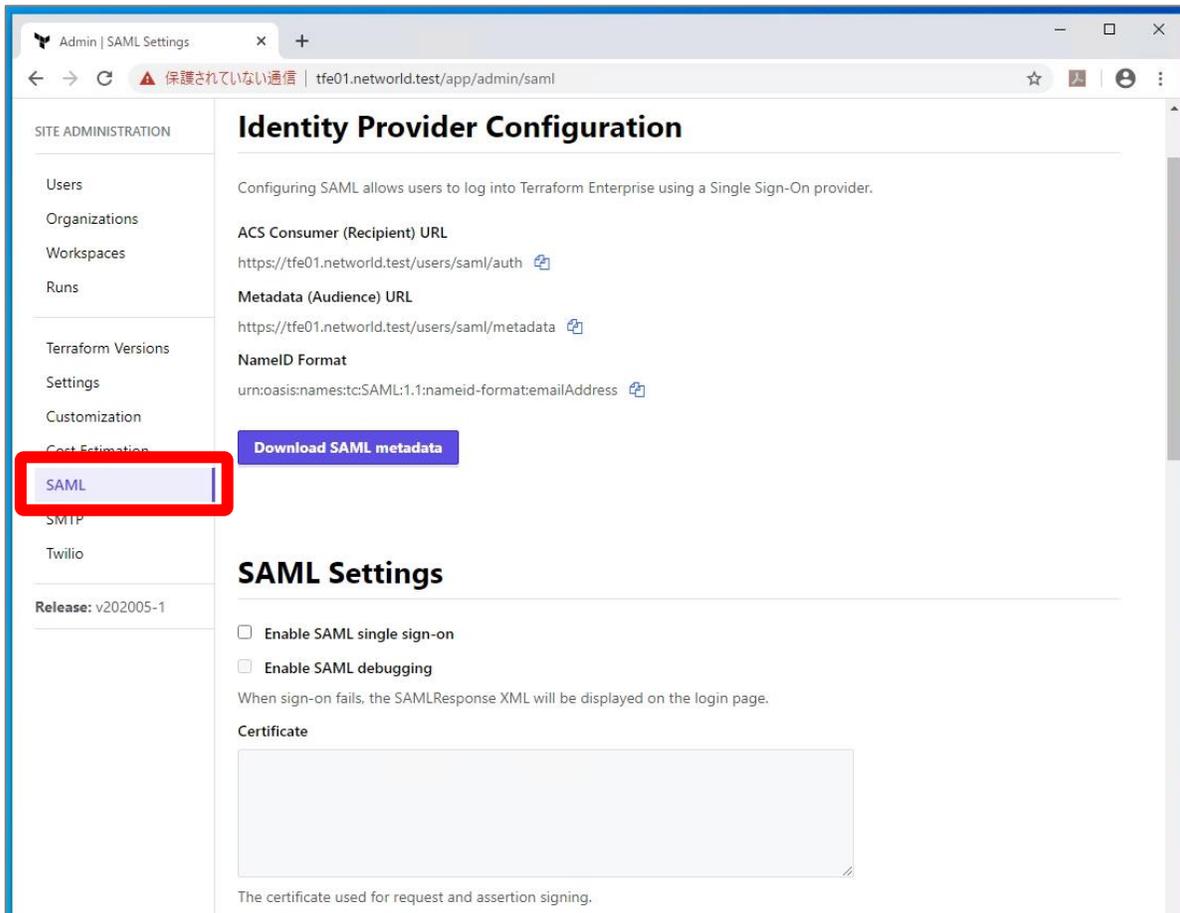
“docker ps” で Terraform Enterprise のプロセスを確認

```
192.168.100.21 - root@tfe01:~ VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
[root@tfe01 ~]# docker ps
```

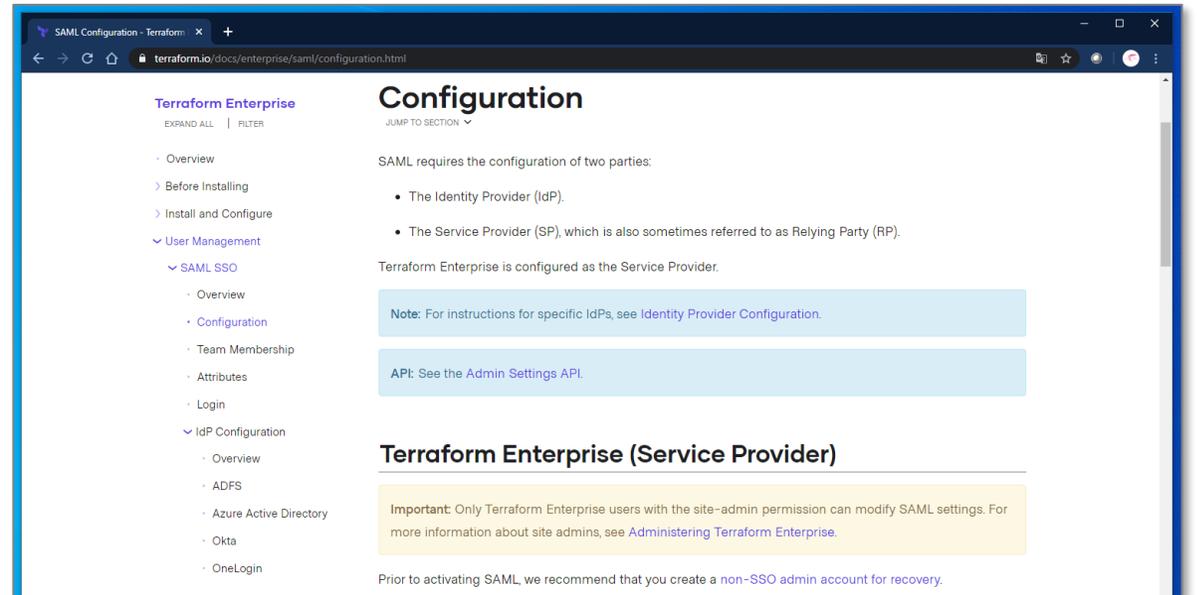
CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
d410a913addf	192.168.100.21:9874/hashicorp-atlas:CIRC-74535-514208127d	ptfe_sidekiq	"/usr/bin/init.sh ba..."	22 hours ago	Up 22 hours	
c8148c7636ca	192.168.100.21:9874/hashicorp-tf-registry:CIRC-3974-delbe025	ptfe_registry_api	"setup-ssl /usr/bin/..."	22 hours ago	Up 22 hours	
e48d910459c0	192.168.100.21:9874/hashicorp-atlas:CIRC-74535-514208127d	ptfe_atlas	"/usr/bin/init.sh ba..."	22 hours ago	Up 22 hours	0.0.0.0:9292->9292/tcp
7f8c9e41e8d6	192.168.100.21:9874/hashicorp-tf-registry:CIRC-3974-delbe025	ptfe_registry_worker	"setup-ssl /usr/bin/..."	22 hours ago	Up 22 hours	
f6de88ad3a2d	192.168.100.21:9874/hashicorp-terraform-build-worker:CIRC-500-7d346cb	goofy_shamir	"start-tbw /terrafor..."	22 hours ago	Up 22 hours	
a2ae1c34ca0c	192.168.100.21:9874/hashicorp-archivist:CIRC-128-9398793	ptfe_archivist	"/usr/bin/wait-for-t..."	22 hours ago	Up 22 hours	0.0.0.0:7675->7675/tcp
7bc4ca81678e	192.168.100.21:9874/hashicorp-terraform-build-manager:CIRC-50-431cd32	ptfe_build_manager	"/usr/bin/tbm-start"	22 hours ago	Up 22 hours	
0238974b9f89	192.168.100.21:9874/hashicorp-ptfe-vault:CIRC-53-b485c55	ptfe_vault	"vault-start"	22 hours ago	Up 22 hours	0.0.0.0:8200->8200/tcp
aa740c0c826c	192.168.100.21:9874/hashicorp-ptfe-postgres:86ba725	ptfe_postgres	"docker-entrypoint.s..."	22 hours ago	Up 22 hours	0.0.0.0:5432->5432/tcp
5ca156539474	192.168.100.21:9874/hashicorp-ptfe-rabbitmq:df253ff	ptfe_rabbitmq	"/start.sh rabbitmq..."	22 hours ago	Up 22 hours	0.0.0.0:5672->5672/tcp, 0.0.0.0:32791->4369/tcp, 0.0.0.0:32790->5671/tcp, 0.0.0.0:32789->25672/tcp

TFE注目機能：SAML 認証

Single Sign-On 連携が可能



The screenshot shows the 'Admin | SAML Settings' page in a web browser. The left sidebar contains a navigation menu with 'SAML' highlighted in a red box. The main content area is titled 'Identity Provider Configuration' and includes fields for 'ACS Consumer (Recipient) URL', 'Metadata (Audience) URL', and 'NameID Format'. A 'Download SAML metadata' button is visible. Below this is the 'SAML Settings' section with checkboxes for 'Enable SAML single sign-on' and 'Enable SAML debugging'. A 'Certificate' field is also present.

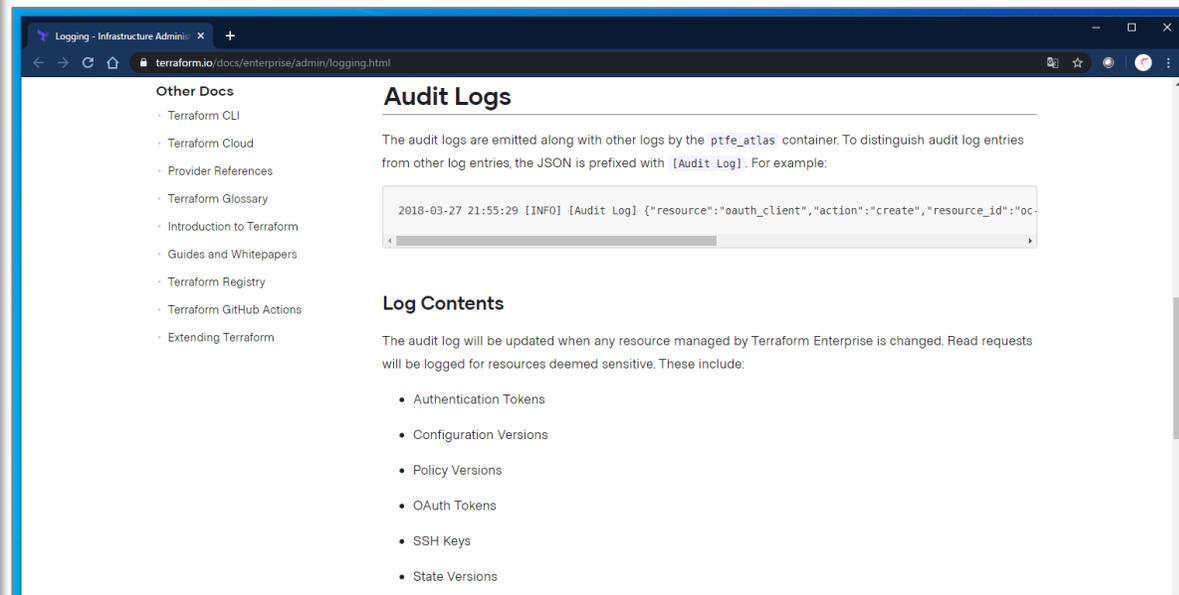
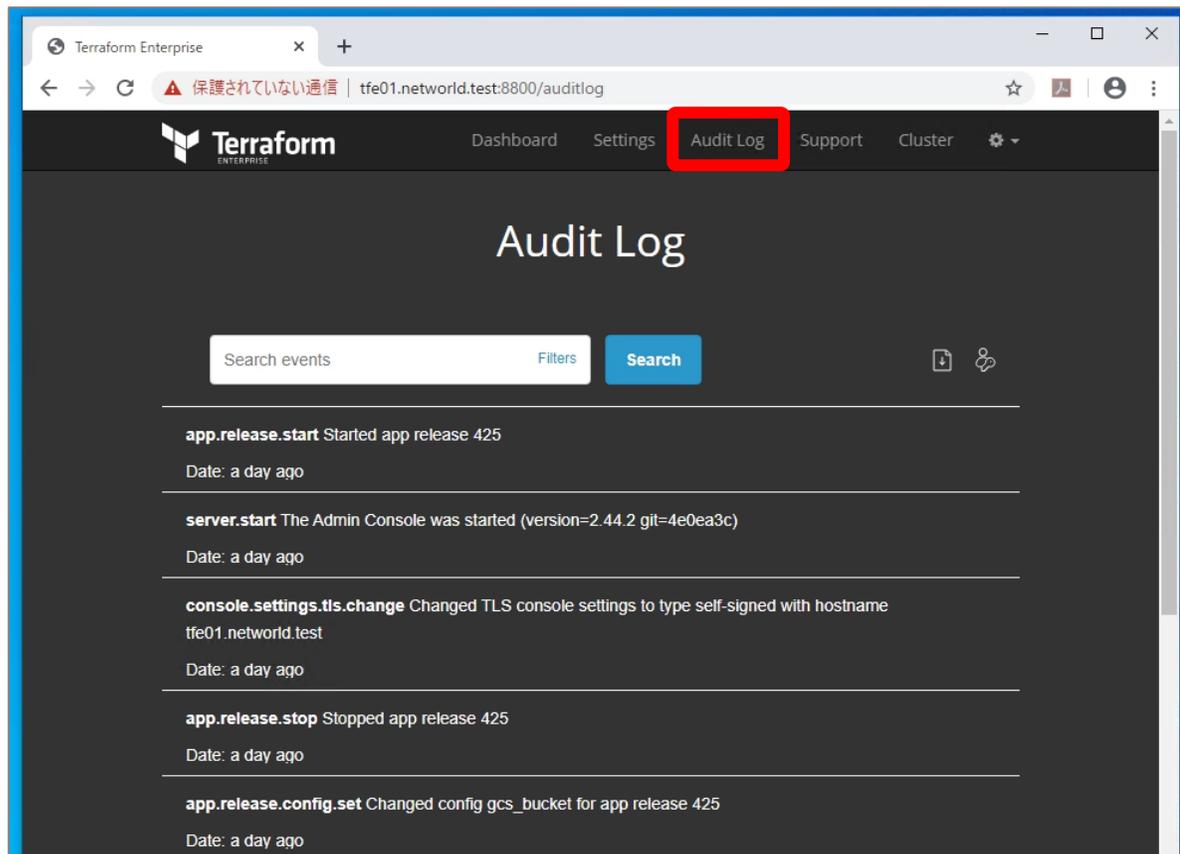


The screenshot shows the 'SAML Configuration - Terraform' documentation page. The left sidebar lists navigation options, with 'SAML SSO' expanded to show 'Configuration'. The main content area is titled 'Configuration' and explains that SAML requires the configuration of two parties: the Identity Provider (IdP) and the Service Provider (SP). It notes that Terraform Enterprise is configured as the Service Provider. There are two callout boxes: a blue one with a note about specific IdPs and a yellow one with an important note about site-admin permissions. The page also includes a section for 'Terraform Enterprise (Service Provider)' with an important note and a recommendation to create a non-SSO admin account for recovery.

<https://www.terraform.io/docs/enterprise/saml/configuration.html>

TFE注目機能：監査ログ

「Terraform で いつどこで誰が何をしたのか」をロギング

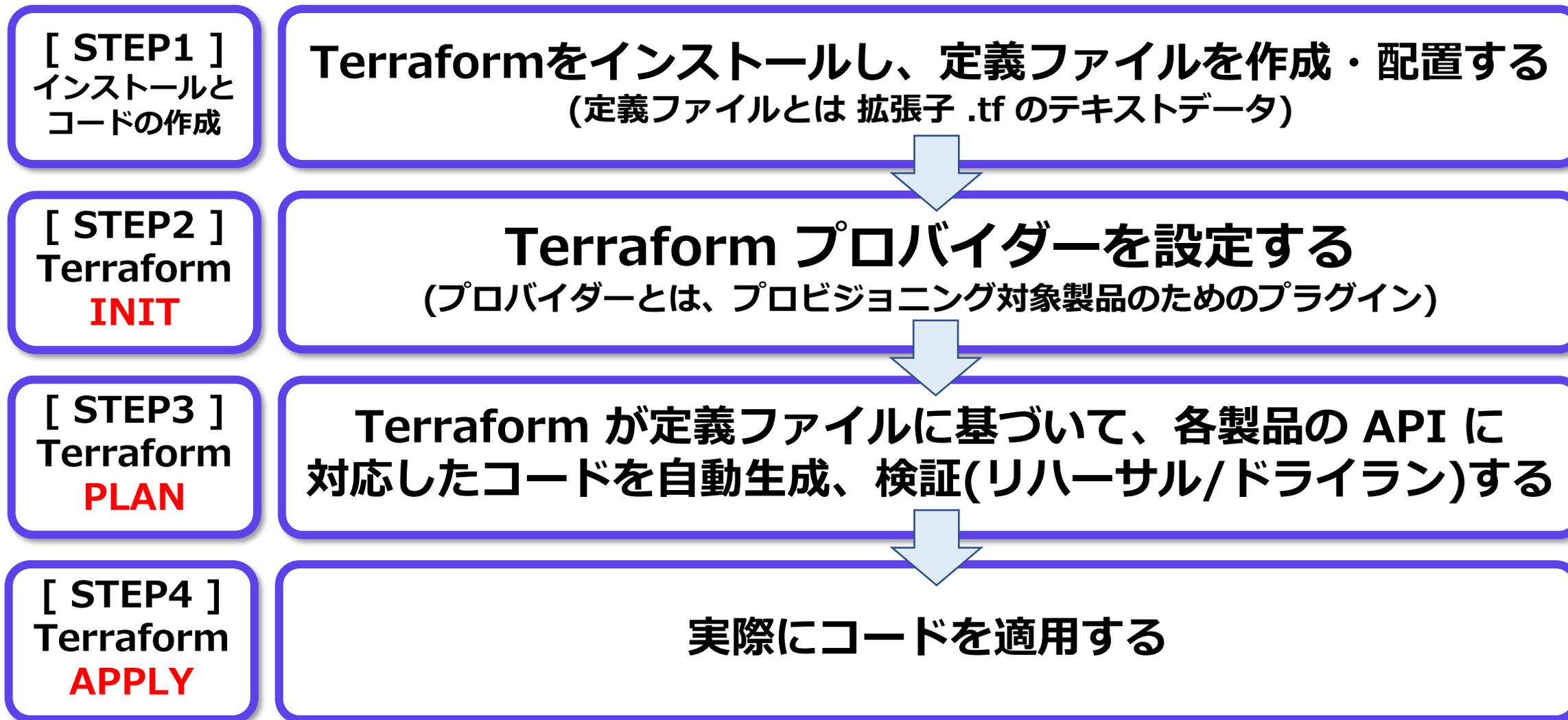


<https://www.terraform.io/docs/enterprise/admin/logging.html>

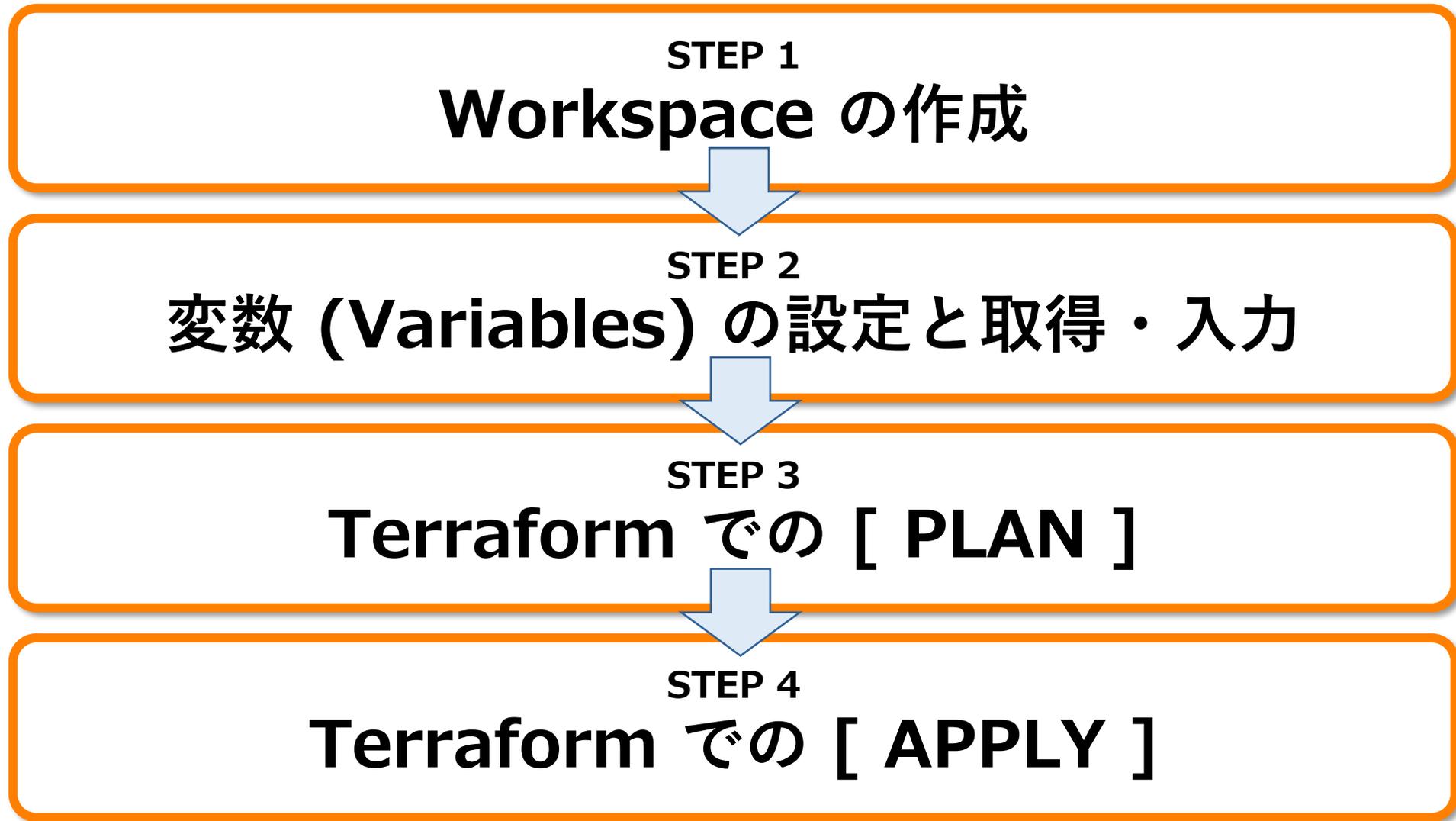


Terraform Enterprise のデモ (動画)

おさらい : Terraform を実行するためのステップ



では、Terraform Enterprise ではどうなるのか？



デモのお品書き

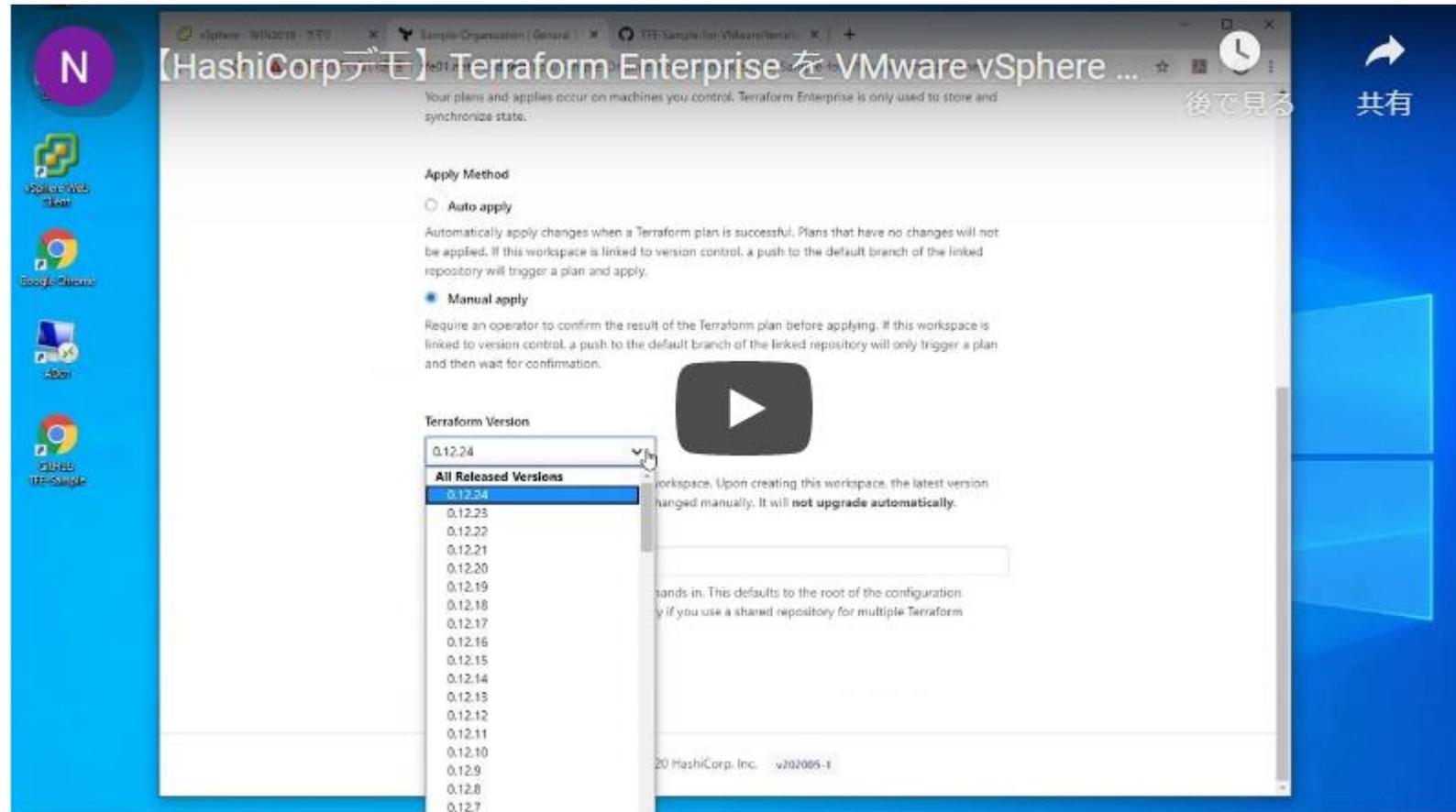
**Terraform Enterprise にログインして
GitHub と連携させたワークスペースを作成する**

Terraform (OSS版) と完全に同じコードを使う

**vSphere 6.7u1 の環境で テンプレートを使って
Windows Server を 1台 プロビジョニングする**
(仮想マシンの名前は TFEVM001 とする)

デモ : Terraform Enterprise と vSphere の組み合わせ

- <https://www.youtube.com/watch?v=C8k285e4TUs>





3スライドで まとめ

まとめ：Infrastructure as Code (Iac) とは

インフラ

CPUやメモリ、ディスクといったリソース、
あるいは仮想マシンやアプライアンスそのもの

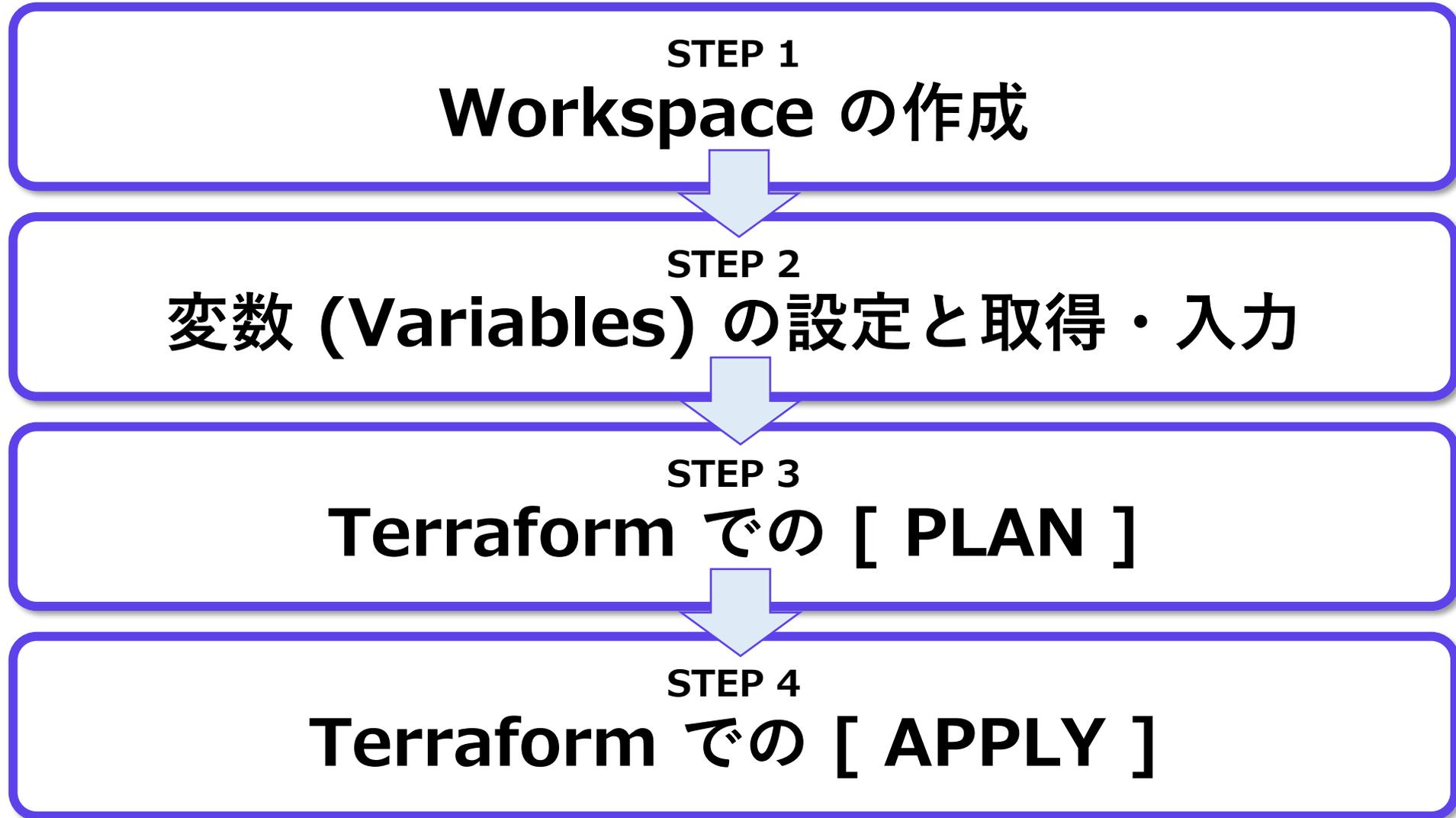


を

コードで 表現すること

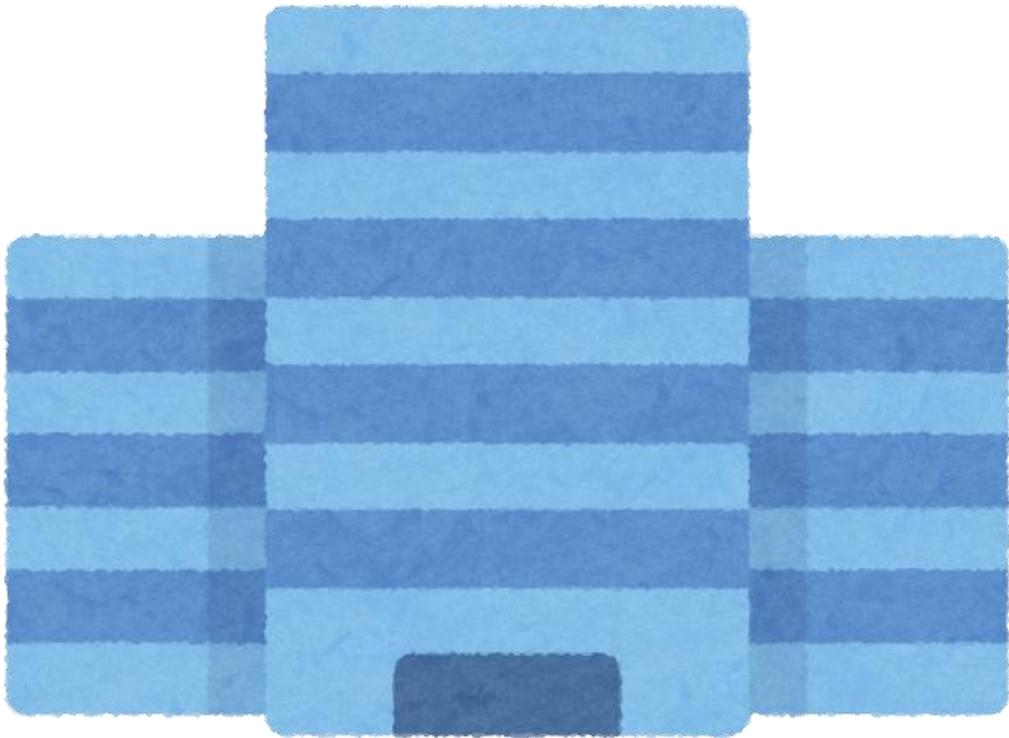


まとめ : Terraform Enterprise 実行ステップ



まとめ：Terraform Enterprise とは

安心・安全 ハイブリッドクラウドで
に使うなら **Terraform Enterprise**



こんなお客様へ!!

- サポートがほしい
- チームで取り組みたい
(属人性をできる限りなくしたい)
- 管理・監査が必要
(ヒヤリハットを防ぎたい)
- オンプレミスでも使いたい



Networld