



## Indice generale

Cos'è Terraform.....	2
Obiettivo del Test.....	2
Ambiente in esame.....	2
Obiettivo.....	2
Installazione di Terraform.....	3
Aggiungere Terraform al PATH di sistema.....	3
Verificare l'installazione.....	4
Preparazione del template.....	4
Configurazione Winrm – Macchina Terraform.....	5
Configurazione Winrm – Macchina Template.....	5
Creazione degli script.....	6
Script .vbs.....	6
Script .bat.....	6
Script .ps1.....	7
Modifica registro di sistema.....	7
Utilizzo di Terraform.....	9
Costruzione del file main.tf – Analisi del codice.....	9
Clonazione di template windows.....	10
Creazione container Linux.....	11
Esecuzione remota.....	13
Esecuzione Locale.....	13
Dipendenze delle Risorse.....	14
Uso combinato di depends_on e local/remote_exec:.....	14
Creazione di più macchine utilizzando un solo blocco resource.....	15
Import di una risorsa.....	16
Descrizione:.....	16
Costruzione del file variables.tf – Analisi del codice.....	20
Applicazione delle modifiche di Terraform.....	21
Inizializzazione ed applicaione.....	21
Eliminazione di una risorsa.....	23
Passaggi:.....	23
Conclusione.....	24
Script Terraform utilizzato nel laboratorio.....	25
Main.tf.....	25
variable.tf.....	26
Script per la configurazione della macchine windows.....	27

# Cos'è Terraform

Terraform è uno strumento di Infrastructure as Code (IaC) che consente di automatizzare la gestione dell'infrastruttura IT mediante file di configurazione dichiarativi. Questo approccio facilita la creazione, modifica e gestione delle risorse IT in modo efficiente e ripetibile.

In questo report, utilizzeremo Terraform su una macchina Windows per clonare macchine e creare container virtuali in Proxmox, una piattaforma di virtualizzazione open-source. Terraform automatizzerà il provisioning delle VM a partire da un template esistente, semplificando e velocizzando il processo di creazione delle risorse.

Vedremo come configurare Terraform su Windows e integrarlo con Proxmox per ottimizzare la clonazione delle macchine virtuali.

Terraform non è da intendersi come un linguaggio di programmazione, tramite il codice non diamo dei comandi, ma descriviamo un'infrastruttura. Le istruzioni in terraform non vengono eseguite sequenzialmente, ma in parallelo (salvo che non venga specificato esplicitamente).

## Obiettivo del Test

### Ambiente in esame

#### 1. Proxmox Virtual Environment 8.0.3:

Questo software di virtualizzazione è stato utilizzato per creare e gestire le macchine virtuali.

Impostazioni di rete	
IP	192.168.3.99
Subnet mask	255.255.255.0
GateWay	192.168.3.7

#### 2. OPNsense 24.7:

È stata implementata una macchina virtuale utilizzando OPNsense come router per collegare la rete interna (192.168.x.y) ad una sottorete realizzata ad hoc per "appoggiare" temporaneamente le macchine (10.0.0.x). OPNsense ha fornito la funzionalità di DHCP server, consentendo la gestione automatica degli indirizzi IP per le macchine virtuali e i dispositivi connessi alla rete.

Impostazioni di rete	
IP – LAN interface	192.168.3.59
IP – WAN interface	10.0.0.59
Subnet mask	255.255.255.0

Impostazioni server DHCP	
IP range	
10.0.0.100	10.0.0.110
GateWay	10.0.0.59

#### 3. Windows 10 Template:

Utilizzando una macchina template di Windows 10, verranno generate n macchine virtuali. Questo approccio permette di ridurre significativamente il tempo necessario per la creazione di nuove istanze, poiché il template contiene già le impostazioni di base e le applicazioni necessarie

Impostazioni di rete	DHCP
----------------------	------

#### 4. Windows 10 Client Fisico con Terraform:

Infine, è stato utilizzato un client fisico con Windows 10, per effettuare le operazioni di Terraform.

Impostazioni di rete	
IP	192.168.3.120
Subnet mask	255.255.255.0
GateWay	192.168.3.7
Trace route per 10.0.0.0	192.168.3.59

#### Obiettivo

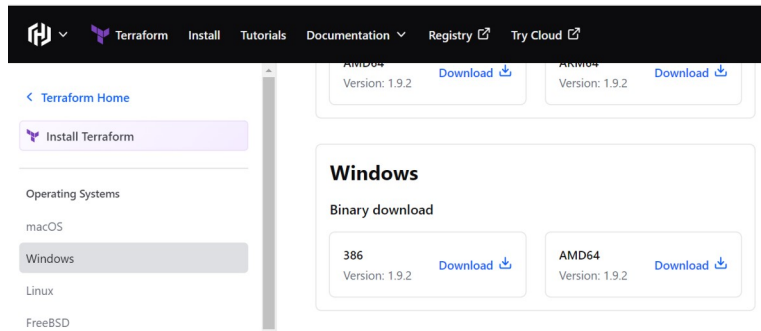
L'obiettivo principale di questo test è generare un numero definito di macchine Windows a partire da un template preconfigurato e completare la loro configurazione, inclusi le impostazioni di rete e il nome della macchina. Questo processo si propone di ottimizzare l'efficienza del provisioning delle macchine virtuali e garantire una configurazione coerente e standardizzata.

Dopo che OPNsense assegna gli indirizzi di rete, Terraform si collega a ciascuna macchina generata. Utilizzando uno script PowerShell precedentemente caricato sulla macchina template, Terraform avvia la configurazione finale delle macchine. Questo script riceve i dati necessari, come le impostazioni di rete e i nomi delle macchine, permettendo una personalizzazione rapida e automatizzata.

## Installazione di Terraform

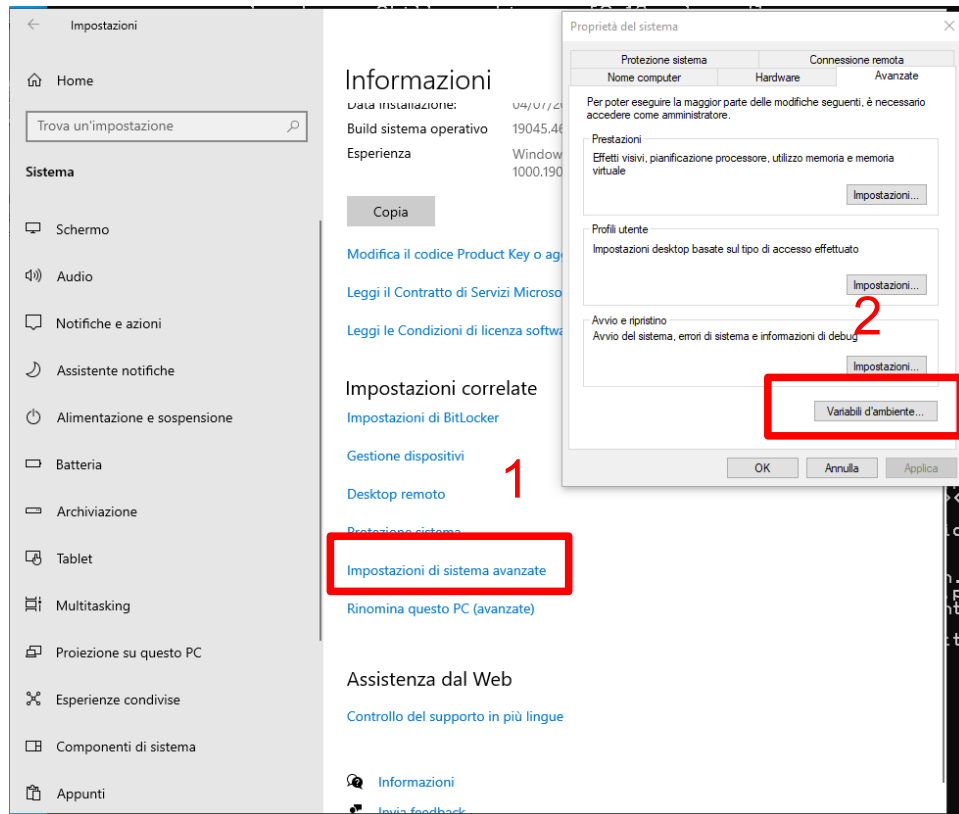
Scaricare il file d'installazione dal sito ufficiale per il sistema operativo dove state operando ed estrarre il file zip, in questo caso le operazioni saranno eseguite su Windows.

[Sito download](#)

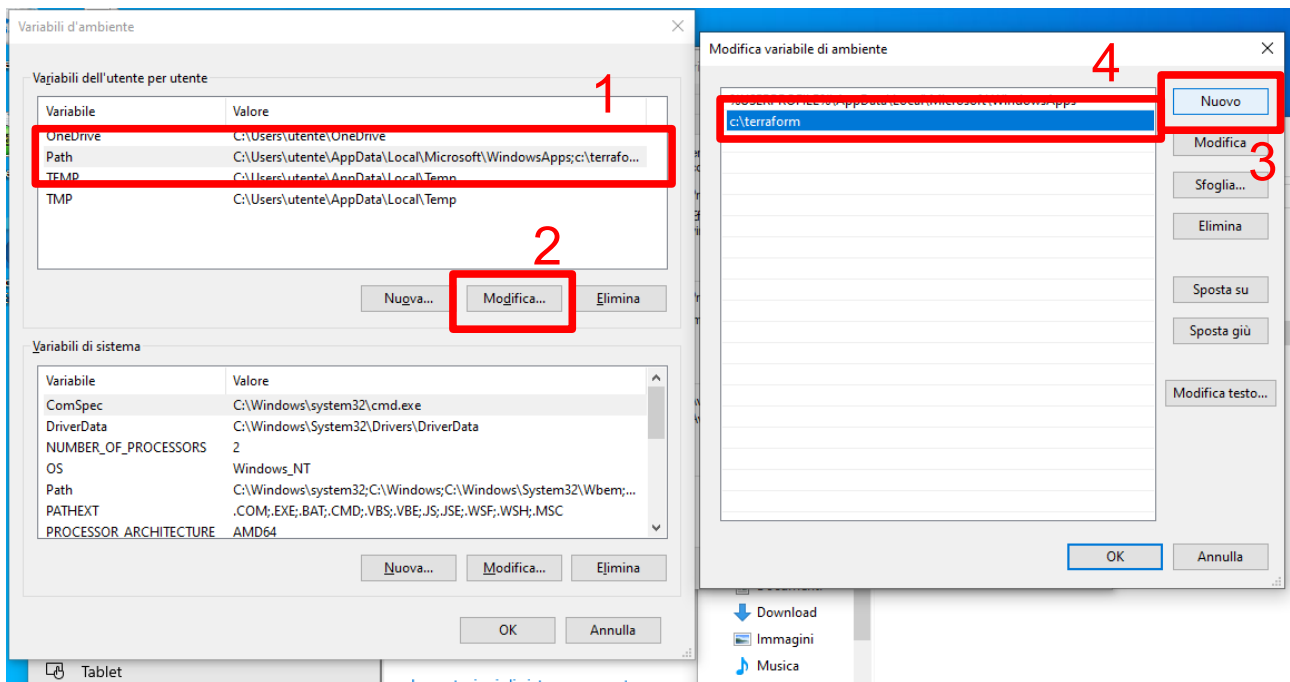


## Aggiungere Terraform al PATH di sistema

Per utilizzare Terraform da qualsiasi posizione nel prompt dei comandi, bisogna aggiungere il percorso della cartella contenente `terraform.exe` al PATH di sistema. Per fare ciò fare tasto destro su Start → Sistema → Impostazioni di sistema avanzate → Variabili d'ambiente



Nella schermata che si apre selezionare la voce Path → Modifica → Nuovo → Path di terraform



## Verificare l'installazione

Apriamo il prompt dei comandi e digitiamo `terraform --version`. Se Terraform è stato installato correttamente, vedrai la versione installata visualizzata nel prompt dei comandi.

```
C:\Users\utente>terraform --version
Terraform v1.9.1
on windows_386

Your version of Terraform is out of date! The latest version
is 1.9.2. You can update by downloading from https://www.terraform.io/downloads.html
```

## Preparazione del template

Dopo aver creato una VM eseguiamo il primo avvio ed installiamo eventuali feature di nostro interesse.

Dopo la clonazione, si verifica un problema poiché la macchina appena creata risulta identica al template, con lo stesso nome, indirizzo IP e indirizzo MAC. Sebbene Proxmox fornisca un nuovo indirizzo MAC alla macchina clonata, è necessario forzare il riavvio della scheda di rete per applicarlo. Per realizzare ciò, si utilizza un piccolo file batch (.bat) che viene eseguito all'avvio di Windows per riavviare la scheda di rete. Inoltre, si impiega uno script PowerShell per ripristinare il servizio WinRM, che potrebbe perdere alcune impostazioni a causa del riavvio della scheda di rete. Questi script vengono eseguiti all'avvio della macchina, richiamati da un file VBS che si aggiunge al registro di sistema.

L'idea è quella di impostare le macchine in modalità DHCP, in modo che il server DHCP assegni loro un indirizzo IP. Per evitare di intasare la rete principale e semplificare l'individuazione delle macchine, si utilizzerà una rete d'appoggio 10.0.0.0/24, sulla quale il server DHCP fornirà un numero limitato di indirizzi IP. In questo modo, non sarà necessario tenere traccia delle macchine già configurate.

Successivamente, si eseguirà da remoto uno script PowerShell che rinominerà le macchine e imposterà l'indirizzo di rete definitivo per ciascuna di esse.

### **Configurazione Winrm – Macchina Terraform**

Otteniamo le informazioni sui profili di connessione di rete attualmente in uso sul sistema e cambiamolo impostando la categoria di rete su "Privata" :

```
Get-NetConnectionProfile  
Set-NetConnectionProfile -Name "Rete 2" -NetworkCategory Private
```

Avviamo il servizio WinRM

```
sc start winrm
```

Consentire l'esecuzione di script

```
Set-ExecutionPolicy RemoteSigned -Force
```

Configuriamo il servizio Windows Remote Management (WinRM) applicando le impostazioni predefinite

```
winrm quickconfig -q
```

Similmente alla configurazione del server, configuriamo il client WinRM per consentire la comunicazione non criptata ed abilitiamo l'autenticazione di base per il client WinRM.

```
Set-Item -Path WSMan:\localhost\Client\AllowUnencrypted -value $true  
Set-Item -Path WSMan:\localhost\Client\Auth\Basic -value $true
```

Aggiungiamo il server all'elenco degli host attendibili per il client WinRM sostituendo "nome\_del\_server" con il nome della macchina server

```
Set-Item -Path WSMan:\localhost\Client\TrustedHosts -value "win10Template"
```

Ora è tutto pronto e possiamo eseguire comandi sul computer remoto, utilizziamo il seguente comando inserendo il nome del computer remoto ed il nome utente. All'interno del parametro -ScriptBlock Inseriamo il comando da eseguire sul computer remoto.

```
Invoke-Command -ComputerName "nome_server" -ScriptBlock { Get-Process } -Credential  
"nome_utente" -Authentication Basic
```

### **Configurazione Winrm – Macchina Template**

Otteniamo le informazioni sui profili di connessione di rete attualmente in uso sul sistema e cambiamolo impostando la categoria di rete su "Privata" :

```
Get-NetConnectionProfile  
Set-NetConnectionProfile -Name "Rete 2" -NetworkCategory Private
```

Avviamo il servizio WinRM

```
sc start winrm
```

Consentire l'esecuzione di script

```
Set-ExecutionPolicy RemoteSigned -Force
```

Configuriamo il servizio Windows Remote Management (WinRM) applicando le impostazioni predefinite

```
winrm quickconfig -q
```

Abilitiamo la comunicazione non criptata per le connessioni WinRM.

```
Set-Item -Path WSMan:\localhost\Service\AllowUnencrypted -Value $true
```

Abilita l'autenticazione di base per le connessioni WinRM, che invia credenziali in testo chiaro.

```
Set-Item -Path WSMan:\localhost\Service\Auth\Basic -Value $true
```

## Creazione degli script

### Script .vbs

Nella cartella C:\Windows creiamo il file CambiaMacAd.vbs che conterrà il seguente codice

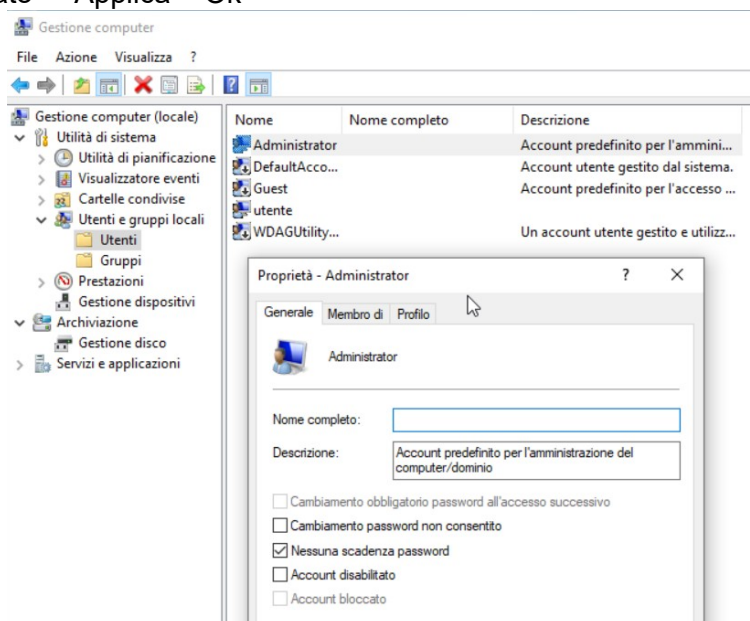
```
Set WshShell = CreateObject("WScript.Shell")
WshShell.Run "cmd /c ""C:\Windows\CambiaMac.bat""", 0, True
WshShell.Run "powershell -Command ""Start-Process powershell -ArgumentList '-ExecutionPolicy Bypass -File C:\Windows\AttivaWinrm.ps1' -Verb RunAs""", 0, True
```

### Script .bat

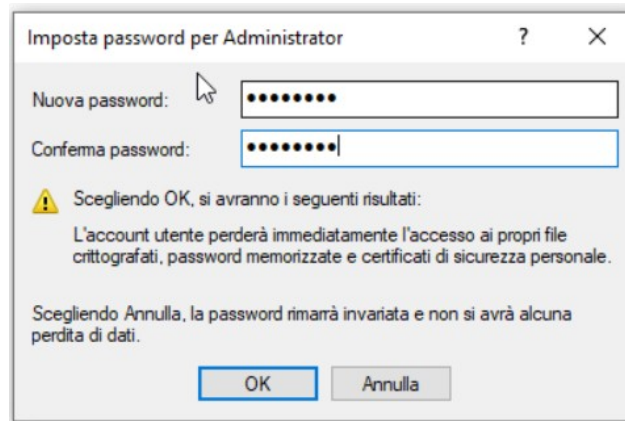
Nella cartella C:\Windows creiamo il file CambiaMac.bat che conterrà il seguente codice

```
netsh interface set interface name="Ethernet 2" admin=disabled
netsh interface set interface name="Ethernet 2" admin=enabled
```

2. Abilitiamo l'utente Administrator: Tasto destro su Start > Gestione Computer > Utenti e gruppi locali > Utenti > Tasto destro su Administrator > Proprietà > Rimuoviamo la spunta dalla casella "Account disabilitato" > Applica > Ok



3. Impostiamo la password dell'utente: Tasto Dx > Impostazine Password > Inseriamo la nuova password



4. Eseguiamo manualmente un primo avvio del file con l'utente Administrator

```
c:\Windows>runas /user:Administrator "CambiaMac.bat"
Immettere la password per Administrator:
Tentativo di avvio di CambiaMac.bat come utente "WIN10TEMPLATE\Administrator" ...
```

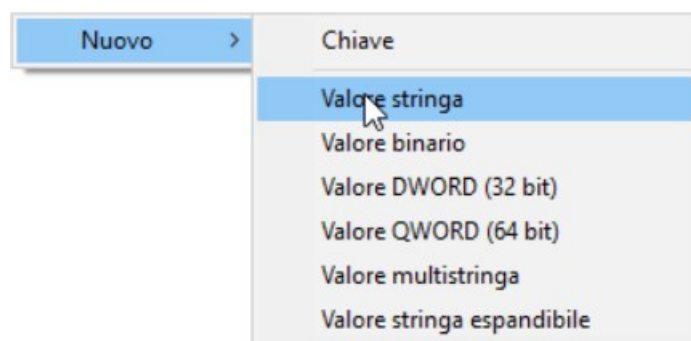
### **Script .ps1**

Nella cartella C:Windows creiamo il file AttivaWinrm.ps1 che conterrà il seguente codice

```
Set-Item -Path WSMan:\localhost\Service\AllowUnencrypted -Value $true
Set-Item -Path WSMan:\localhost\Auth\Basic-Value $true
```

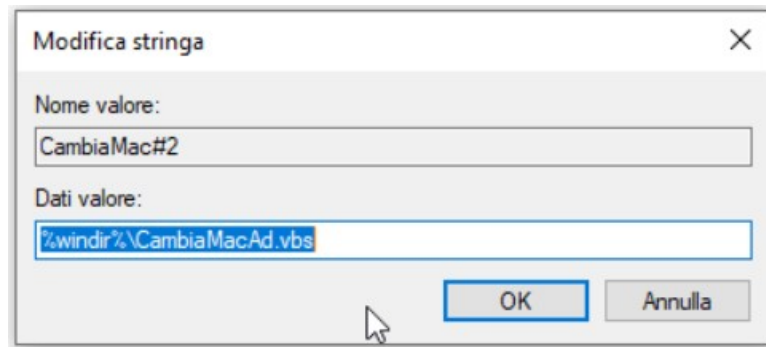
### **Modifica registro di sistema**

6. Inseriamo il file nel registro di sistema così che venga eseguito all'avvio. Apriamo Regedit e apriamo la chiave HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run  
Run tasto dx > Nuovo > Valore stringa

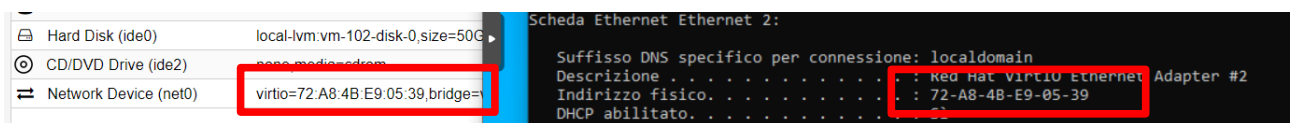




Diamo un nome alla stringa ed inseriamo il path del link



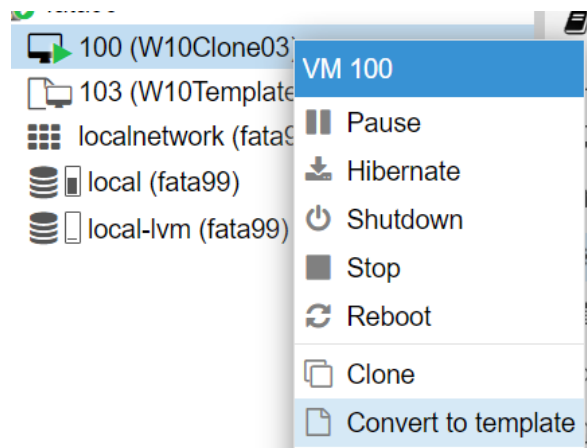
7. Per testare il corretto funzionamento cambiamo il MAC della scheda di rete dall'interfaccia di proxmox e riavviamo la macchina



Ed effettuiamo un test di connessione Winrm con il comando:

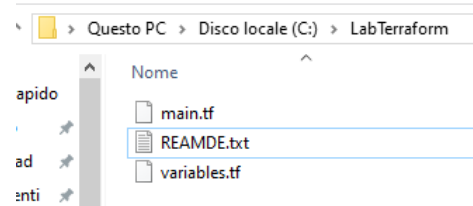
```
Invoke-Command -ComputerName "indirizzo_macchina" -ScriptBlock { Get-Process } -  
Credential "utente" -Authentication Basic
```

8. Creiamo il template e siamo pronti a spostarci su Terraform



# Utilizzo di Terraform

Creiamo una cartella dedicata al progetto ed al suo interno due file `main.tf` e `variables.tf` i quali conterranno rispettivamente il codice e le variabili.



## **IMPORTANTE**

*Questi file descrivono e definiscono **l'intera infrastruttura** all'interno del nostro provider, questo significa che al suo interno dovranno essere descritti contemporaneamente **tutti** gli elementi presenti su di esso.*

*Se un elemento viene omissso/aggiunto/modificato questo verrà eliminato/creato/alterato alla successiva applicazione dei piani (vedremo in seguito)*

## Costruzione del file `main.tf` – Analisi del codice

### 1. Blocco 'terraform'

Scopo: Questo blocco definisce le dipendenze del provider Terraform necessario per interagire con l'infrastruttura Proxmox.

**required\_providers:** Specifica il provider proxmox con il suo source e la versione. In questo caso, il provider è fornito da telmate e la versione specificata è 2.9.11. Questo garantisce che Terraform utilizzi una versione compatibile del provider per evitare problemi di compatibilità.

```
terraform {
  required_providers {
    proxmox = {
      source = "telmate/proxmox"
      version = "2.9.11"
    }
  }
}
```

### 2. Blocco 'provider'

Scopo: Configura il provider Proxmox per permettere a Terraform di comunicare con l'API di Proxmox.

**N.B.** Ogni provider ha la sua configurazione, assicurati di inserire quella relativa al tuo caso

Parametri:

- `pm_api_url`: URL dell'API di Proxmox, fornito tramite una variabile (`var.pm_api_url`).
- `pm_user`: Nome utente per l'autenticazione.

- `pm_password`: Password per l'autenticazione.
- `pm_tls_insecure`: Impostato su `true` per consentire connessioni TLS non sicure.

```
provider "proxmox" {
  pm_api_url      = var.pm_api_url
  pm_user         = var.pm_user
  pm_password     = var.pm_password
  pm_tls_insecure = true
}
```

## Clonazione di template windows

### a. Resource `"proxmox_vm_qemu" "cloned_vms"`

**Scopo:** Questa risorsa definisce una macchina virtuale QEMU clonata da un template esistente in Proxmox.

- **Parametri:**

- **name:** Nome della VM, costruito dinamicamente utilizzando una variabile (`var.cont`).
- **target\_node:** Specifica il nodo Proxmox su cui verrà creata la VM (in questo caso, `fata99`).
- **clone:** Nome del template da cui clonare la VM (`W10Template`).
- **os\_type:** Specifica il tipo di sistema operativo, in questo caso `win10`.
- **cores e sockets:** Configurano la CPU della VM.
- **memory:** Quantità di RAM allocata per la VM (2048 MB).
- **scsihw:** Specifica il tipo di hardware SCSI da utilizzare.
- **bootdisk:** Specifica il disco di avvio della VM.

### Sezione di Rete

- **network:** Configura la rete della VM, specificando il modello di rete (`virtio`) e il bridge di rete (`vmbr0`).

```
resource "proxmox_vm_qemu" "cloned_vms" {

  name          = "W10Clone"
  target_node   = "fata99"
  clone         = "W10Template"
  os_type       = "win10"
  cores         = 2
  sockets       = 1
  cpu           = "host"
  memory        = 2048
  scsihw        = "virtio-scsi-pci"
  bootdisk      = "scsi0"

  network {
    model = "virtio"
  }
}
```

```

        bridge          = "vmbro0"
    }
}

```

## Creazione container Linux

Proxmox offre una vasta gamma di template per diverse distribuzioni Linux. È possibile scaricare questi template direttamente dall'interfaccia web di Proxmox

### a. Resource `"proxmox_lxc" "my_container"`

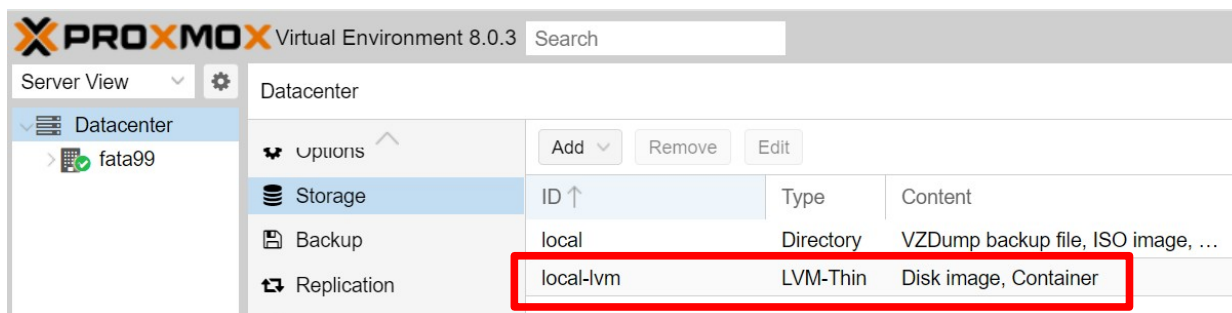
- **Scopo:** Questa risorsa definisce un container LXC chiamato `my_container`, che verrà creato nel nodo Proxmox specificato.
- **Parametri:**
  - **target\_node:** Specifica il nodo Proxmox (in questo caso, `fata99`) su cui il container sarà creato.
  - **hostname:** Imposta il nome host del container (`linux-basic`).
  - **ostemplate:** Indica il template (precedentemente scaricati) da utilizzare per il container. In questo caso, si utilizza un'immagine di Ubuntu 23.04 memorizzata nel pool di storage `local`.
  - **password:** Imposta la password per l'utente `root` del container (`BasicLXCContainer`).
  - **unprivileged:** Indica se il container deve essere eseguito in modalità non privilegiata. Impostato su `true`, questo aumenta la sicurezza del container.
  - **start:** Specifica se il container deve essere avviato automaticamente dopo la creazione. Impostato su `true`.

### b. Sezione `'rootfs'`

**Scopo:** Definisce il filesystem radice (root filesystem) del container

**Parametri:**

- **storage:** Specifica il pool di storage da utilizzare per il filesystem radice. In questo caso, si utilizza `local-lvm`, che è un pool di storage basato su LVM (Logical Volume Manager). Nel caso di proxmox per ricavare tale informazione dobbiamo recarci nella sezione Storage del Datacenter e verificare in quale di questi vengono salvati i container.



- **size:** Imposta la dimensione del filesystem radice a 8 GB.

### c. Sezione `'network'`

**Scopo:** Configura la rete del container.

**Parametri:**

- **name:** Specifica il nome dell'interfaccia di rete all'interno del container (`eth0`).
- **bridge:** Indica il bridge di rete a cui il container sarà connesso. In questo caso, si utilizza `vmbr0`, che è un bridge di rete configurato su Proxmox.
- **ip:** Assegna un indirizzo IP al container. Questo valore è fornito tramite una variabile (`var.ip_addresses_linux`), che consente di gestire dinamicamente gli indirizzi IP.

### d. Configurazione risorse

**Scopo:** Configura le risorse hardware allocate al container.

**Parametri:**

- **memory:** Imposta la quantità di RAM allocata per il container a 1024 MB (1 GB).
- **cores:** Specifica il numero di core della CPU da assegnare al container, in questo caso 2.

```
resource "proxmox_lxc" "my_container" {
  target_node = "fata99"
  hostname    = "lxc-basic"
  otemplate   = "local:vztmpl/ubuntu-23.04-standard_23.04-
1_amd64.tar.zst"
  password    = "BasicLXCContainer"
  unprivileged = true
  start       = true

  // Terraform will crash without rootfs defined
  rootfs {
    storage = "local-lvm"
    size    = "8G"
  }

  network {
    name     = "eth0"
    bridge   = "vmbr0"
    ip       = var.ip_addresses_linux
  }

  memory = 1024
  cores  = 2
}
```

## Esecuzione remota

### Provisioner `remote-exec`

- **Scopo:** Esegue uno script di configurazione su Windows dopo la creazione della VM, questa porzione di codice può essere inserita all'interno di un blocco `resource` dopo aver definito le caratteristiche della macchina.
- **connection:** Configura la connessione a utilizzare il protocollo WinRM per eseguire comandi remoti.
  - **Parametri:** Include dettagli come l'host (indirizzo IP o nome DNS della VM), l'utente, la password, la porta e altre impostazioni di sicurezza.
- **inline:** Contiene il comando PowerShell da eseguire, che chiama uno script di configurazione (`post-create-config.ps1`) passando variabili per il nuovo hostname, indirizzo IP, gateway e server DNS.

```
provisioner "remote-exec" {
  connection {
    type      = "winrm"
    host      = "Win10Template" # Use the VM's IP address or DNS
    name      = "utente" # Ensure this user has permissions to
    run the script
    password   = "Pa$$w0rd"
    port       = 5985 # Default WinRM port for HTTP
    use_ntlm   = true
    https      = false # Set to `true` if using HTTPS for WinRM
    insecure   = true # Set to `false` if using a valid SSL
    certificate
    timeout    = "3m"
  }

  inline = [
    "powershell -ExecutionPolicy Bypass -File C:\\script\\post-
    create-config.ps1 -NewHostname ${var.hostnames} -IPAddress $
    {var.ip_addresses_windows} -Gateway ${var.gateway} -DNS1 ${var.dns1} -
    DNS2 ${var.dns2}"
  ]
}
```

## Esecuzione Locale

### Provisioner `local-exec`

- **Scopo:** Esegue comandi o script sulla macchina locale dove Terraform è in esecuzione. È utile per attività di post-elaborazione o configurazioni che devono essere eseguite una volta che Terraform ha completato la creazione delle risorse, come la generazione di file di configurazione o l'invio di notifiche.

### Parametri:

- **command:** Specifica il comando o lo script da eseguire sulla macchina locale. Può essere un comando semplice o un percorso a uno script eseguibile.
- **script:** È possibile utilizzare uno script locale invece di specificare un comando direttamente. Fornisce maggiore flessibilità per eseguire script più complessi.

```
provisioner "local-exec" {  
    command = "echo 'Le risorse sono state create!'"  
}  
  
provisioner "local-exec" {  
    script = "path/to/script.sh"  
}
```

Per l'esempio pratico procedere dopo la sezione "Dipendenze delle Risorse"

## Dipendenze delle Risorse

### Parametro `depends_on`

- **Scopo:** Assicura che Terraform rispetti un ordine specifico nella creazione, aggiornamento o eliminazione delle risorse. Utilizzato per garantire che una risorsa venga creata o aggiornata solo dopo che altre risorse siano state completate. Questo è particolarmente utile quando le risorse hanno dipendenze implicite o quando si ha bisogno di garantire un ordine di esecuzione preciso.

### Parametri:

- **`depends_on`:** Una lista di riferimenti ad altre risorse che devono essere completate prima che la risorsa corrente venga creata o modificata.

```
resource "nome_risorsa" "esempio" {  
    # configurazioni della risorsa  
    depends_on = [resource.nome_risorsa.altra_risorsa]  
}
```

## Uso combinato di `depends_on` e `local/remote_exec`:

- **`null_resource con depends_on`:** Può essere usato per eseguire azioni solo dopo che altre risorse sono state completate, ad esempio eseguendo uno script o una configurazione che richiede che tutte le risorse precedenti siano state create.

```
resource "null_resource" "post_setup" {  
    depends_on = [aws_instance.example_instance]  
    provisioner "local-exec" {  
        command = "echo 'Le istanze sono state create!'"  
    }  
}
```

## Comportamenti Chiave:

- **Creazione e Aggiornamento:** Terraform rispetta le dipendenze definite da `depends_on` durante la creazione e l'aggiornamento delle risorse, garantendo che tutte le risorse necessarie siano pronte prima di procedere.
- **Eliminazione:** Durante la fase di eliminazione, Terraform elimina le risorse nell'ordine inverso rispetto a `depends_on`, assicurando che le risorse dipendenti vengano rimosse solo dopo che le risorse da cui dipendono sono state eliminate.

## Creazione di più macchine utilizzando un solo blocco resource

Il meta-argomento `'count'` in Terraform permette di creare più istanze di una risorsa o modulo, specificando un numero intero che rappresenta il numero di istanze da generare. Ogni istanza avrà un oggetto di infrastruttura distinto, che verrà creato, aggiornato o distrutto separatamente quando viene applicata la configurazione.

Quando `'count'` è impostato, è disponibile un oggetto aggiuntivo `count` nelle espressioni, in modo da poter modificare la configurazione di ogni istanza. Questo oggetto ha un solo attributo:

- `count.index` - Il numero di indice distinto (a partire da 0) corrispondente a questa istanza

Terraform distingue tra il blocco stesso e le multiple istanze di risorsa.

- `<TYPE>.<NAME>` o `module.<NAME>` (ad esempio, `proxmox_lxc.my_container`) si riferisce al blocco di risorse.
- `<TYPE>.<NAME>[<INDEX>]` o `module.<NAME>[<INDEX>]` (ad esempio, `proxmox_lxc.my_container[0]`, ecc.) si riferisce a singole istanze

Prendendo in esame il caso precedente modifichiamo lo script per creare 3 container, modifichiamo il codice come segue:

- Aggiungiamo il `count` all'inizio della risorsa;
- Modifichiamo il parametro `hostname` così che concateni il nome al numero incrementale del `count`;
- Modifichiamo la variabile `ip_addresses` rendendola una lista di stringhe da cui attingeremo per prendere gli indirizzi delle macchine.

```
resource "proxmox_lxc" "my_container" {
  count      = 3
  target_node = "fata99"
  #Concateno nome con numero
  hostname   = "Container-Ubuntu-${count.index}"
  otemplate  = "local:vztmpl/ubuntu-23.04-standard_23.04-1_amd64.tar.zst"
  password   = "Pa$w0rd"
  unprivileged = true
  start      = true
}
```



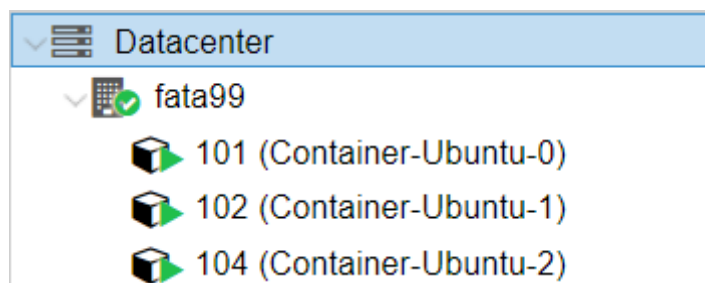
```

network {
  name    = "eth0"
  bridge  = "vbr0"
  #Accedo alla lista degli indirizzi
  ip      = "${var.ip_addresses_linux[count.index]}/24"
}
... }

#Dichiaro una lista di indirizzi
variable "ip_addresses_linux" {
  type = list(string)
  default = ["192.168.3.100", "192.168.3.101", "192.168.3.102"]
}

```

Di seguito il risultato su proxmox



## Import di una risorsa

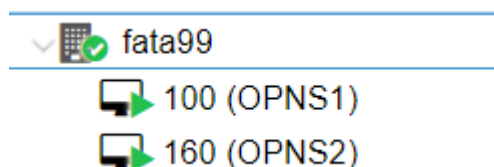
Nelle operazioni quotidiane può capitare di trovarsi a dover lavorare con risorse preesistenti o generate manualmente. In questo caso per far sì che Terraform possa gestire anche queste risorse c'è la necessità di integrarle nel flusso di lavoro, possiamo utilizzare il seguente comando:

```
terraform import proxmox_vm_qemu.nome_risorsa pve/qemu/ID
```

Descrizione:

- **terraform import:** Comando di Terraform usato per importare risorse esistenti nel tuo stato di Terraform.
- **proxmox\_vm\_qemu.opnsense\_1:** Identificatore della risorsa in Terraform che rappresenta la VM QEMU su Proxmox. `proxmox_vm_qemu` è il tipo di risorsa, e `nome_risorsa` è il nome della risorsa.
- **pve/qemu/100:** Identificatore specifico della risorsa nella piattaforma Proxmox. `pve` indica il cluster Proxmox, `qemu` indica che si tratta di una VM QEMU, e `100` è l'ID della VM su Proxmox.

Prendiamo in esame la seguente situazione dove abbiamo creato manualmente due VM OpnSense.



Prima di lanciare il comando aggiungiamo il blocco resource che andrà a gestire la risorsa che stiamo importando

```
resource "proxmox_vm_qemu" "opnsense_1" {  
  target_node = "fata99"  
}
```

Lanciamo il comando, se l'operazione va a buon fine otterremo la seguente schermata:

```
c:\terraform\LabContainer>terraform import proxmox_vm_qemu.opnsense_1 pve/qemu/100  
proxmox_vm_qemu.opnsense_1: Importing from ID "pve/qemu/100"...  
proxmox_vm_qemu.opnsense_1: Import prepared!  
  Prepared proxmox_vm_qemu for import  
proxmox_vm_qemu.opnsense_1: Refreshing state... [id=pve/qemu/100]  
  
Import successful!  
  
The resources that were imported are shown above. These resources are now in  
your Terraform state and will henceforth be managed by Terraform.
```

Con il comando `terraform show` possiamo vedere le caratteristiche della macchina importata

```
c:\terraform\LabContainer>terraform show  
# proxmox_vm_qemu.opnsense_1:  
resource "proxmox_vm_qemu" "opnsense_1" {  
  agent          = 0  
  args           = null  
  balloon        = 0  
  bios           = "seabios"  
  boot           = "order=scsi0;ide2;net0"  
  bootdisk       = null  
  bridge         = null  
  cicustom       = null  
  cipassword     = (sensitive value)  
  ciuser         = null  
  cores          = 1  
  cpu            = "x86-64-v2-AES"  
  define_connection_info = true  
  desc           = null  
  disk_gb        = 0  
  force_create   = false  
  full_clone     = false  
  hgroup         = null  
  hastate        = null  
  hotplug        = "network,disk,usb"  
  id             = "fata99/qemu/100"  
  ipconfig0      = null  
  ipconfig1      = null
```

Partendo da queste informazioni andiamo a trascriverle nel blocco resource;

```
resource "proxmox_vm_qemu" "opnsense_1" {  
  target_node = "fata99"
```

```

name          = "OPNS1"
cores         = 1
sockets       = 1
memory        = 2048
cpu           = "x86-64-v2-AES"
onboot        = true
scsihw        = "virtio-scsi-single"
full_clone    = false # Impostazione che richiede la sostituzione
boot          = "order=scsi0;ide2;net0" #c
automatic_reboot = true

disk {
  size      = "10G"
  type      = "scsi"
  storage   = "local-lvm"
  iothread  = 1
  format    = "raw"
  volume    = "local-lvm:vm-100-disk-0"
}

network {
  model     = "virtio"
  bridge    = "vmbr0"
  firewall  = true
  macaddr   = "12:57:2C:C2:9F:F5"
}

network {
  model     = "virtio"
  bridge    = "vmbr0"
  firewall  = true
  macaddr   = "82:86:B9:EE:C1:51"
}

}

```

In questa fase potrebbe esser necessario effettuare diverse prove per far sì che il blocco rispecchi a pieno le caratteristiche della macchina, basterà leggere il report del comando `plan` e modificare i valori all'interno del main con il settaggio attuale della macchina che terraformi riporta **alla sinistra della** freccia.

```

# proxmox_vm_qemu.opsense_1 will be updated in-place
~ resource "proxmox_vm_qemu" "opsense_1" {
  + additional_wait      = 0
  + automatic_reboot     = true
  ~ boot                 = "order=scsi0;ide2;net0" -> "order=scsi0;net0"
  + clone_wait          = 0

```

Main.tf scritto inizialmente

```
boot = "order=scsi0;net0"
```

Main.tf modificato

```
boot = "order=scsi0;ide2;net0"
```

A questo punto il Plan ci dirà che non ci sono macchine da distruggere e apporterà delle sue modifiche interne, per questo test creeremo due container per verificare l'effettiva applicazione del piano.

```
# proxmox_vm_qemu.opnsense_1 will be updated in-place
~ resource "proxmox_vm_qemu" "opnsense_1" {
  + additional_wait      = 0
  + automatic_reboot     = true
  + clone_wait          = 0
  + guest_agent_ready_timeout = 100
  id                    = "fata99/qemu/100"
  name                  = "OPNS1"
  + oncreate             = true
  + preprovision         = true
  tags                  = null
  # (48 unchanged attributes hidden)

  - timeouts {}

  # (3 unchanged blocks hidden)
}
```

Plan: 2 to add, 1 to change, 0 to destroy.

Applicando le modifiche verifichiamo che il nostro OpnSense è rimasto inalterato (ha mantenuto le impostazioni) e sono stati correttamente creati i due container.

The screenshot shows the Proxmox VE web interface. On the left, a tree view under 'Datacenter' shows a cluster named 'fata99' containing several VMs: 101 (Container2), 102 (Container1), 100 (OPNS1), 160 (OPNS2), 103 (W10Template), localnetwork (fata99), local (fata99), and local-lvm (fata99). VM 100 (OPNS1) is selected. The middle pane shows the 'Console' tab for VM 100. The right pane shows the console output, which displays network configuration for LAN (vtnet0) and WAN (vtnet1) interfaces, followed by a list of tasks: 0) Logout, 1) Assign interfaces, 2) Set interface IP address, 3) Reset the root password, and 4) Reset to factory defaults.

## Costruzione del file variables.tf – Analisi del codice

La sintassi per creare una variabile è la seguente

```
variable "nome_variabile" {  
    type          = tipo_variabile  
    default       = valore_predefinito  
    description = "Descrizione della variabile"  
}
```

Il file utilizzato nel nostro caso è il seguente

```
variable "pm_user" {  
    description = "proxmox user"  
    default     = "root@pam"  
}
```

```
variable "pm_password" {  
    description = "Proxmox password"  
    default     = "Pa$$w0rd"  
}
```

```
variable "pm_api_url" {  
    description = "Proxmox API URL"  
    default     = "https://192.168.3.99:8006/api2/json"  
}
```

```
variable "new_vm_name" {  
    type          = list(string)  
    default       = ["W10Pro-1", "W10Pro-2"]  
}
```

```
#variabili per connessioni winrm  
variable "admin_password" {  
    default     = "Pa$$w0rd"  
}
```

```
### N.B Windows e linux utilizzano formati diversi per quanto riguarda  
IP  
variable "ip_addresses_windows" {  
    default = "192.168.3.102"  
}
```

```
variable "ip_addresses_linux" {  
    default = "192.168.3.103/24"  
}
```

```
variable "gateway" {  
    default = "192.168.3.2"  
}
```

```

variable "dns1" {
    default = "8.8.8.8"
}
variable "dns2" {
    default = "192.168.3.2"
}

#variabile che passo come parametro al plan
variable "cont" {
    default = "0"
}

```

## Applicazione delle modifiche di Terraform

### Inizializzazione ed applicaione

Apriamo il cmd e spostiamoci sulla directory dove si trovano i nostri file. Inizializziamo il progetto Terraform. Questo comando scarica i provider e prepara l'ambiente di lavoro.

```
terraform init
```

E' possibile utilizzare il parametro chdir prima del comando `init` per operare da una directory differente a quella di lavoro. Questo parametro si può utilizzare con tutti i comandi di Terraform

```
-chdir=C:\Percorso\Directory\Lavoro
```

In caso di esito positivo otterremo la seguente shermata

```

C:\LabContainer>terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of telmate/proxmox from the dependency lock file
- Using previously-installed telmate/proxmox v2.9.11

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.

```

Dopo l'inizializzazione, è consigliabile ma non necessario controllare la configurazione per assicurarsi che non ci siano errori di sintassi o di configurazione. Utilizzare il comando:

```
terraform validate
```

```

C:\LabContainer>terraform validate
Success! The configuration is valid.

```

Prima di applicare le modifiche, è utile vedere quali cambiamenti verranno apportati all'infrastruttura. Utilizzare il comando `plan` per generare un piano di esecuzione, il valore del parametro `-out` rappresenta il nome arbitrario del piano, il parametro `var` permette di passare un parametro al main (il nome della variabile deve essere uguale al nome utilizzato in `variables.tf`).

```
terraform plan -out="Plan1"

-var="nome_variabile=valore"
```

E' **importante** eseguire **questo passaggio** per verificare che vengano effettuate solo le modifiche di nostro interesse, infatti può capitare che per un errore di sintassi Terraform modifichi altre installazioni già presenti nell'infrastruttura che non siano previste.

```
# proxmox_vm_qemu.cloned_vms will be updated in-place
~ resource "proxmox_vm_qemu" "cloned_vms" {
  id          = "fata99/qemu/100"
  - name      = "W10Clone03" -> null
  tags       = null
  # (56 unchanged attributes hidden)

  # (1 unchanged block hidden)
}

Plan: 1 to add, 1 to change, 0 to destroy.
```

In caso di esito positivo otterremo la seguente schermata

```
C:\LabContainer>terraform plan -out="Plan1"
proxmox_vm_qemu.cloned_vms: Refreshing state... [id=fata99/qemu/100]
Terraform used the selected providers to generate the following execution plan. Resources marked with + are to be created, ~ are to be updated, and - are to be destroyed.

Terraform will perform the following actions:

# proxmox_lxc.my_container will be created
+ resource "proxmox_lxc" "my_container" {
+   arch          = "amd64"
+   cmode         = "tty"
+   console       = true
+   cores         = 2
+   cpulimit      = 0
+   cpunits       = 1024
+   hostname      = "lxc-basic"
+   id            = (known after apply)
+   memory        = 1024
+   onboot        = false
+   ostemplate     = "local:vztmpl/ubuntu-23.04-standard_23.04-1_amd64.tar.zst"
+   ostype        = (known after apply)
+   password      = (sensitive value)
+   protection    = false
+   start         = true
+   swap          = 0
+   target_node   = "fata99"
+   tty           = 2
+   unprivileged  = true
+   unused        = (known after apply)
+   vmid          = (known after apply)

+   network {
+     bridge = "vbr0"
+     hwaddr = (known after apply)
+     ip     = "192.168.3.104/24"
+     name   = "eth0"
+     tag    = (known after apply)
+     trunks = (known after apply)
+     type   = (known after apply)
+   }

+   rootfs {
+     size      = "8G"
+     storage   = "local-zfs"
+     volume    = (known after apply)
+   }
+ }

Plan: 1 to add, 0 to change, 0 to destroy.
```

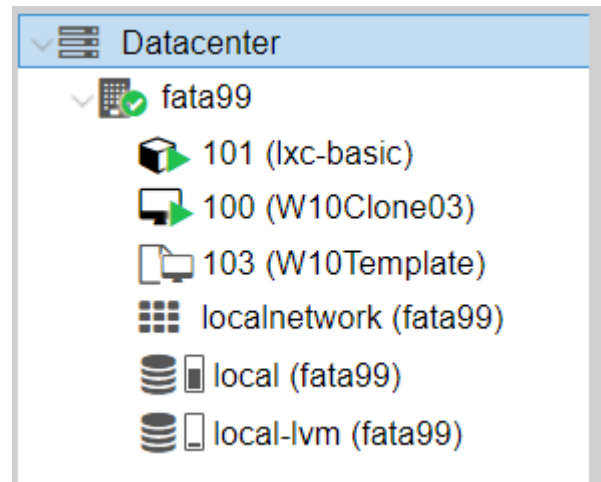
Se il piano generato è soddisfacente, è possibile procedere ad applicare le modifiche all'infrastruttura con il comando:

```
terraform apply "Plan1"
```

Dopo che Terraform ha applicato le modifiche, è buona prassi verificare che le risorse siano state create o aggiornate come previsto. Puoi farlo accedendo alla console del provider o utilizzando i comandi Terraform per elencare le risorse.

```
terraform show
```

```
C:\LabContainer>terraform show
# proxmox_lxc.my_container:
resource "proxmox_lxc" "my_container" {
  arch          = "amd64"
  bwlimit       = 0
  cmode         = "tty"
  console       = true
  cores         = 2
  cpulimit      = 0
  cpuunits      = 1024
  description   = null
  force         = false
  hagrout       = null
  hastate       = null
  hookscript    = null
  hostname      = "lxc-basic"
  id            = "fata99/lxc/101"
  ignore_unpack_errors = false
  lock          = null
  memory        = 1024
  nameserver    = null
  onboot        = false
  ostemplate    = "local:vztmpl/ubuntu"
  ostype        = "Ubuntu"
  password      = (sensitive value)
  protection    = false
  restore       = false
  searchdomain  = null
}
```



## Eliminazione di una risorsa

Per eliminare una risorsa specifica in Terraform, possiamo utilizzare il comando `terraform destroy` insieme all'opzione `-target` per specificare la risorsa che desideriamo rimuovere.

```
terraform destroy -target=<resource_type>.<resource_name>
```

### Passaggi:

1. **Verifica le Risorse:** Assicurati di conoscere il tipo e il nome della risorsa da eliminare. Possiamo utilizzare `terraform state list` per elencare tutte le risorse gestite da Terraform.
2. **Esegui il Comando:** Esegui il comando `terraform destroy -target` specificando la risorsa da eliminare.
3. **Conferma:** Terraform chiederà conferma prima di procedere con l'eliminazione della risorsa specificata.



```

C:\terraform\LabWinTemp_RemShell>terraform destroy -target=proxmox_vm_qemu.cloned_vms[0]
proxmox_vm_qemu.cloned_vms[0]: Refreshing state... [id=fata99/qemu/104]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with
the following symbols:
- destroy

Terraform will perform the following actions:

# null_resource.run_once0 will be destroyed
- resource "null_resource" "run_once0" {
  - id = "153860156" -> null
}

# null_resource.run_once1 will be destroyed
- resource "null_resource" "run_once1" {
  - id = "198597540" -> null
}

# proxmox_vm_qemu.cloned_vms[0] will be destroyed
- resource "proxmox_vm_qemu" "cloned_vms" {
  - additional_wait = 0 -> null
  - agent          = 0 -> null
  - automatic_reboot = true -> null
  - balloon        = 0 -> null
}

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

null_resource.run_once0: Destroying... [id=153860156]
null_resource.run_once1: Destroying... [id=198597540]
null_resource.run_once0: Destruction complete after 0s
null_resource.run_once1: Destruction complete after 0s
proxmox_vm_qemu.cloned_vms[0]: Destroying... [id=fata99/qemu/104]
proxmox_vm_qemu.cloned_vms[0]: Destruction complete after 4s

Warning: Applied changes may be incomplete

The plan was created with the -target option in effect, so some changes requested in the configuration may have
been ignored and the output values may not be fully updated. Run the following command to verify that no other
changes are pending:
    terraform plan

Note that the -target option is not suitable for routine use, and is provided only for exceptional situations such
as recovering from errors or mistakes, or when Terraform specifically suggests to use it as part of an error
message.












Destroy complete! Resources: 3 destroyed.

```

## Conclusione

E' buona abitudine ripulire la cartella di lavoro durante la fase di debug dei file in quanto terraform tiene traccia degli stati delle macchine all'interno dell'infrastruttura, ed in fase di test questo potrebbe causare dei comportamenti anomali o inaspettati.

Seguendo questi passaggi, sarai in grado di applicare le modifiche alla tua infrastruttura utilizzando Terraform in modo efficace e sicuro. La pianificazione e la verifica delle modifiche sono passaggi cruciali per garantire che l'infrastruttura venga gestita correttamente e senza errori.

 .terraform	24/07/2024 16:30	Cartella di file	
 .terraform.lock.hcl	24/07/2024 16:20	File HCL	2 KB
 main.tf	24/07/2024 16:56	File TF	2 KB
 Plan1	24/07/2024 16:57	File	7 KB
 README.txt	24/07/2024 16:18	Documento di testo	1 KB
 terraform.tfstate			
 terraform.tfstate.backup			
 variables.tf			
 main.tf	24/07/2024 16:56	File TF	2 KB
 README.txt	24/07/2024 16:18	Documento di testo	1 KB
 variables.tf	24/07/2024 16:19	File TF	2 KB

## Script Terraform utilizzato nel laboratorio

Con i seguenti script Terraform genera 2 cloni del Template Windows, dopo di che la risorsa "[configurazione\\_post\\_creazione](#)" genererà 10 risorse, una per ogni indirizzo nel range del server DHCP, che tenteranno la connessione verso le macchine create per poi avviare lo script powershell che trovate al capitolo successivo. Alcune connessioni andranno a buon fine (n su 10), mentre le altre andranno in timeout. Le risorse di tipo "[null\\_resource](#)" rimangono nella cartella Terraform sulla macchina locale, non occupano spazio sul virtualizzatore e sono gestite autonomamente da Terraform.

### Main.tf

```
terraform {
  required_providers {
    proxmox = {
      source = "telmate/proxmox"
      version = "2.9.11"
    }
  }
}

provider "proxmox" {
  pm_api_url      = var.pm_api_url
  pm_user         = var.pm_user
  pm_password     = var.pm_password
  pm_tls_insecure = true
}

resource "proxmox_vm_qemu" "cloned_vms" {
  count          = 2 # Numero di cloni

  name          = "W10Clone0${count.index}"
  target_node   = "fata99"
  clone         = "W10Template"
  os_type       = "win10"
  cores         = 2
  sockets       = 1
  cpu           = "host"
  memory        = 2048
  scsihw        = "virtio-scsi-pci"
  bootdisk      = "scsi0"

  network {
    model      = "virtio"
    bridge     = "vmbnet0"
  }
}

# Dummy null_resource che dipende dalla creazione di tutte le macchine
resource "null_resource" "configurazione_post_creazione" {
  # Dipendenza
  depends_on = [proxmox_vm_qemu.cloned_vms]
```

```

count          = 10 #Numero di indirizzi nel nostro range DHCP 100-110

provisioner "remote-exec" {
  connection {
    type      = "winrm"
    host      = "10.0.0.10${count.index}" # Compongo IP con il numero del
contatore
    user      = "utente"
    password  = "Pa$$w0rd"
    port      = 5985
    use_ntlm  = true
    https     = false
    insecure  = true
    timeout   = "3m"
  }

  # Richiamo lo script sulla macchina remota
  inline = [
    "powershell -ExecutionPolicy Bypass -File C:\\script\\post-create-
config.ps1 -NewHostname ${var.hostnames}${count.index} -IPAddress $
${var.ip_addresses}${count.index} -Gateway ${var.gateway} -DNS1 ${var.dns1} -
DNS2 ${var.dns2}"
  ]
}
}

```

## variable.tf

```

variable "pm_user" {
  description = "proxmox user"
  default     = "root@pam"
}

variable "pm_password" {
  description = "Proxmox password"
  default     = "Pa$$w0rd"
}

variable "pm_api_url" {
  description = "Proxmox API URL"
  default     = "https://192.168.3.99:8006/api2/json"
}

###variabili post creazione
variable "hostnames" {
  default = "Win10Macchina"
}

variable "ip_addresses" {
  default = "10.0.0.5"
}

variable "gateway" {

```

```

    default = "10.0.0.59"
  }
  variable "dns1" {
    default = "8.8.8.8"
  }
  variable "dns2" {
    default = "8.8.4.4"
  }

```

## Esecuzione

L'applicazione del piano genererà 12 risorse: 2 macchine, 10 "null\_resource"

```
Plan: 12 to add, 0 to change, 0 to destroy.
```

Avvio della clonazione delle macchine

```

Saved the plan to: Plan1

To perform exactly these actions, run the following command to apply:
  terraform apply "Plan1"

C:\terraform\Prova>terraform apply "Plan1"
null_resource.run_once[0]: Destroying... [id=758505211]
null_resource.run_once[5]: Destroying... [id=577481358]
null_resource.run_once[3]: Destroying... [id=1621770011]
null_resource.run_once[8]: Destroying... [id=7013778801]

```

Terminata la creazione cominciano i tentativi di connessione

```

null_resource.run_once[2]: Creating...
null_resource.run_once[0]: Creating...
null_resource.run_once[2]: Provisioning with 'remote-exec'...
null_resource.run_once[2] (remote-exec): Connecting to remote host via WinRM...
null_resource.run_once[2] (remote-exec): Host: 10.0.0.102
null_resource.run_once[2] (remote-exec): Port: 5985
null_resource.run_once[2] (remote-exec): User: utente
null_resource.run_once[2] (remote-exec): Password: true
null_resource.run_once[2] (remote-exec): HTTPS: false
null_resource.run_once[2] (remote-exec): Insecure: true
null_resource.run_once[2] (remote-exec): NTLM: true
null_resource.run_once[2] (remote-exec): CACert: false

```

Connessione stabilita

```

null_resource.run_once[4] (remote-exec): Connected!
null_resource.run_once[3] (remote-exec): Connected!
#< CLIXML
<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04"><Obj S="progress" RefId="0"><TN RefId="0"><T>System.Management.Automation.PSCustomObject</T><T>System.Object</T></TN><MS><I64 N="SourceId">1</I64><PR N="Record"><AV>Preparazione dei moduli per il primo utilizzo.</AV><AI>0</AI><Nil /><PI>-1</PI><PC>-1</PC><T>Completed</T><SR>-1</SR><SD> </SD></PR></MS></Obj></Objs>#< CLIXML
<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04"><Obj S="progress" RefId="0"><TN RefId="0"><T>System.Management.Automation.PSCustomObject</T><T>System.Object</T></TN><MS><I64 N="SourceId">1</I64><PR N="Record"><AV>Preparazione dei moduli per il primo utilizzo.</AV><AI>0</AI><Nil /><PI>-1</PI><PC>-1</PC><T>Completed</T><SR>-1</SR><SD> </SD></PR></MS></Obj></Objs>#< CLIXML

```

Avvio degli script

```

null_resource.run_once[3] (remote-exec): C:\Users\utente>powershell -ExecutionPolicy Bypass -File C:\script\post-create-config.ps1 -NewHostname Win10Macchina3 -IPAddress 10.0.0.53 -Gateway 10.0.0.59 -DNS1 8.8.8.8 -DNS2 8.8.4.4

null_resource.run_once[4] (remote-exec): C:\Users\utente>powershell -ExecutionPolicy Bypass -File C:\script\post-create-config.ps1 -NewHostname Win10Macchina4 -IPAddress 10.0.0.54 -Gateway 10.0.0.59 -DNS1 8.8.8.8 -DNS2 8.8.4.4

```

Le connessioni verso indirizzi non assegnati falliscono

```
Error: remote-exec provisioner error

with null_resource.run_once0[1],
on main.tf line 51, in resource "null_resource" "run_once0":
51:   provisioner "remote-exec" {

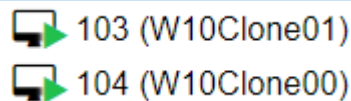
timeout - last error: unknown error Post "http://10.0.0.101:5985/wsman": dial tcp 10.0.0.101:5985: connectex:
Impossibile stabilire la connessione. Risposta non corretta della parte connessa dopo l'intervallo di tempo oppure
mancata risposta dall'host collegato.

Error: remote-exec provisioner error

with null_resource.run_once0[4],
on main.tf line 51, in resource "null_resource" "run_once0":
51:   provisioner "remote-exec" {

error executing "C:/Temp/terraform_131979861.cmd": unknown error Post "http://10.0.0.104:5985/wsman": net/http:
timeout awaiting response headers
```

Le macchine sono state generate



E lo script correttamente eseguito

```
C:\Users\utente>ipconfig /all

Configurazione IP di Windows

Nome host . . . . . : Win10Macchina4
Suffisso DNS primario . . . . . :
Tipo nodo . . . . . : Ibrido
Routing IP abilitato. . . . . : No
Proxy WINS abilitato . . . . . : No

Scheda Ethernet Ethernet 2:

Suffisso DNS specifico per connessione:
Descrizione . . . . . : Red Hat VirtIO Ethernet Adapter #2
Indirizzo fisico. . . . . : DA-A9-AA-41-42-22
DHCP abilitato. . . . . : No
Configurazione automatica abilitata : Sì
Indirizzo IPv6 locale rispetto al collegamento . . . : fe80::253c:5a64:4b01:386%10(Preferenziale)
Indirizzo IPv4. . . . . : 10.0.0.54(Preferenziale)
Subnet mask . . . . . : 255.255.255.0
Gateway predefinito . . . . . : 10.0.0.59
IAID DHCPv6 . . . . . : 25:789312
DUID Client DHCPv6. . . . . : 00-01-00-01-2E-28-38-59-02-00-00-00-00-01
Server DNS . . . . . : 8.8.8.8
                        8.8.4.4
NetBIOS su TCP/IP . . . . . : Attivato
```

## Script per la configurazione della macchina windows

All'interno del disco C abbiamo creato la directory script.

Questo script riceve come parametro il nuovo nome della macchina, IP, Gateway, DNS1 e 2.

N.B. Se le macchine possono vedersi a vicenda gli output dei comandi verranno trasmessi sulla shell da cui eseguiamo terraform, il che torna utile per il monitoraggio delle operazioni ed il debug.

# - Leggiamo i parametri

```

param(
    [string]$NewHostname,
    [string]$IPAddress,
    [string]$Gateway,
    [string]$DNS1,
    [string]$DNS2
)
function ConRete() {
    Write-Output "I parametri passati sono:
        -Nome Host '$NewHostname'
        -Ip '$IPAddress'
        -Gateway '$Gateway'
        -DNS1 '$DNS1'
        -DNS2 '$DNS2'"

    # - Ricaviamo il nome dell'interfaccia
    $adapter = Get-NetAdapter | Where-Object {$_.Status -eq "Up"} | Select-Object -
ExpandProperty Name
    Write-Output "Nome interfaccia rete: '$adapter'"
    # - Impostiamo il nuovo indirizzo IP e Gateway
    New-NetIPAddress -InterfaceAlias $adapter -IPAddress $IPAddress -PrefixLength 24 -
DefaultGateway $Gateway
    # - DNS
    Set-DnsClientServerAddress -InterfaceAlias $adapter -ServerAddresses $DNS1, $DNS2
    Write-Output "La nuova configurazione di rete è: "
    Get-NetIPConfiguration

    # - Rinomina del Pc e riavvio per applicare le modifiche
    Rename-Computer -NewName $NewHostname -Restart -Force
}
ConRete

```