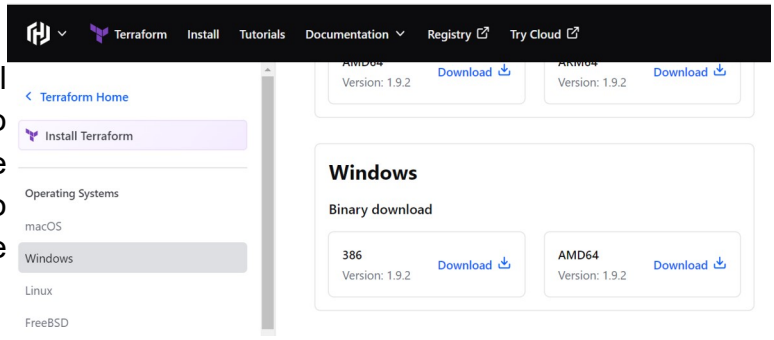
**Cos’è Terraform**

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 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.

## Installazione di Terraform

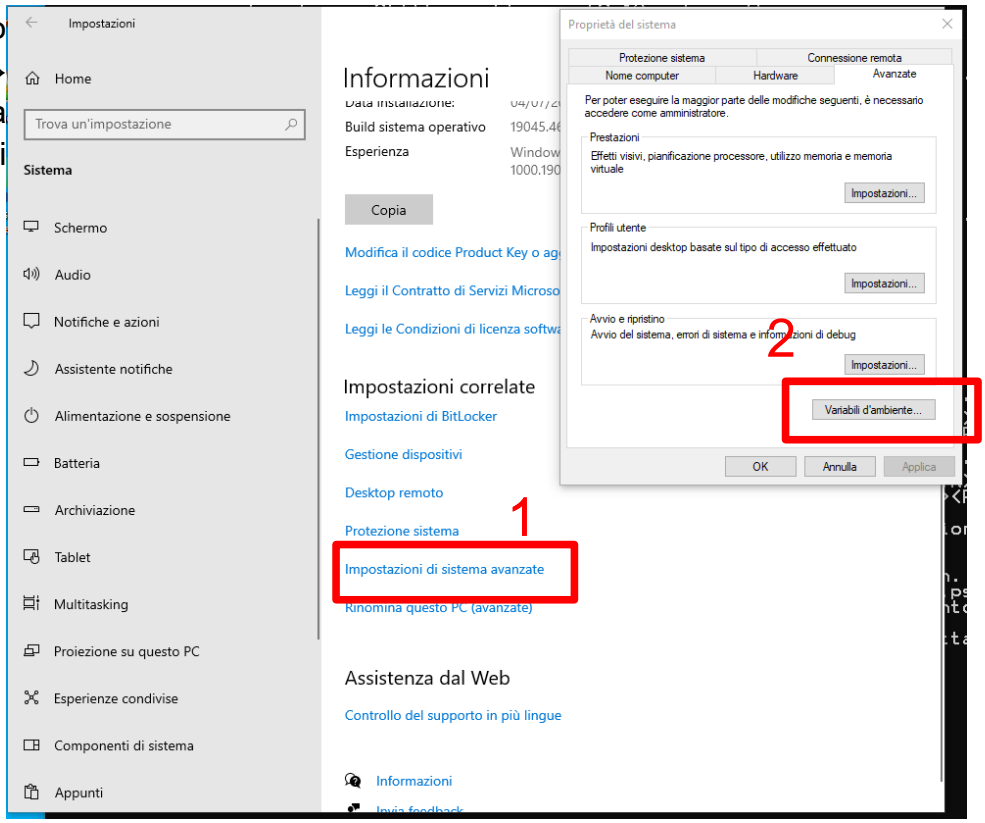



Scaricare il file d’installazione dal sito ufficiale per il sistema operativo dove state operando, estrarre il file zip ed eseguire il file .exe, in questo caso le operazioni saranne eseguite su Windows.

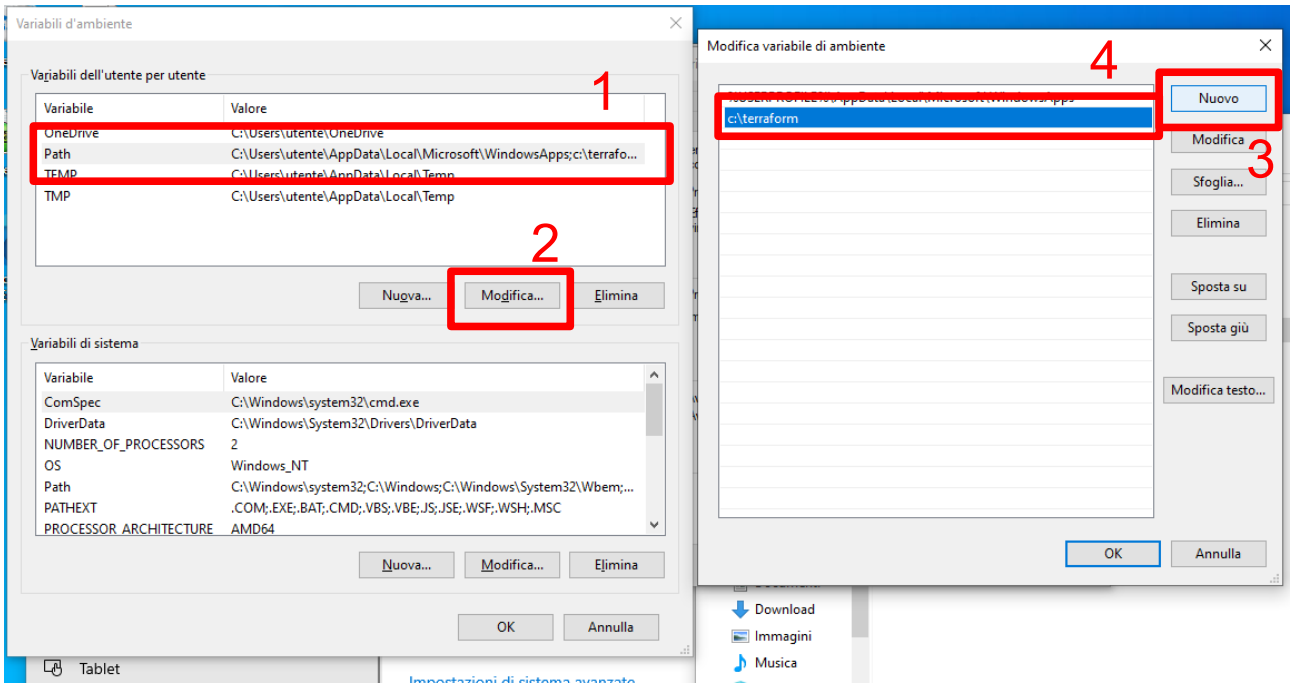
Sito download

## 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

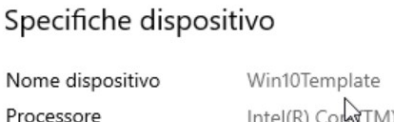
Apriamo il promt dei comandi e digitiamo `terraform --version` Se Terraform è stato installato correttamente, vedrai la versione installata visualizzata nel prompt dei comandi.

```
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
```

# Clonazine macchine windows

### Preparazione del template

Dopo aver creato una VM eseguiamo il primo avvio ed installiamo eventuali feature di nostro interesse. Nel nostro caso andremo ad aggiungere degli script poweshell da richiamare al momento della clonazione per installare servizi ed effettuare i settaggi di nostro interesse, il che ci permette di effettuare qualsiasi tipo di operazione senza limiti.




Per rendere possibile ciò diamo un nome di default alla macchina (Es. Win10Template) e impostiamo la scheda di rete in modalità dhcp, così da poterci collegare tramite protocollo WinRM alla macchina clonata per poi settarla (se la macchina non è impostata in DHCP l’assegnazione del nuovo IP potrebbe non andare a buon fine).

### 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
```

**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
```

**Script per la rinomina della macchina ed impostazione della scheda di rete**

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.* Gli output dei comandi verranno trasmessi sulla shell da cui eseguiremo 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'
```

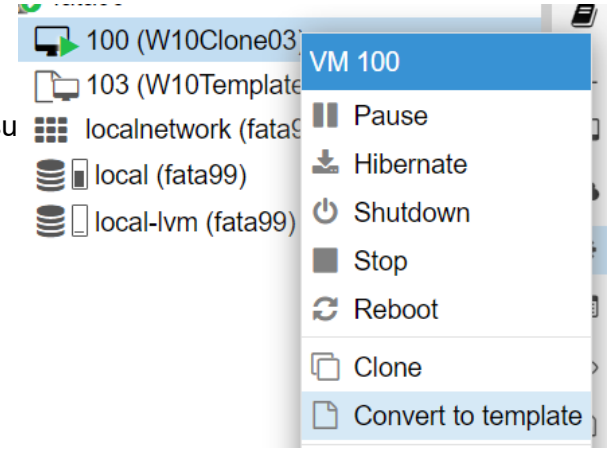
```
    -Ip '$IPAddress'
    -Gateway '$Gateway'
    -DNS1 '$DNS1'
    -DNS2 '$DNS2'"

# - Ricaviamo il nome dell'interfaccia
  $adapter = Get-NetAdapter | Where-Object {$_.Status -eq "Up"} | Select-Object -ExpandProperty Name
  Write-Output "Nome interfaccia rete: '$adapter'"
# - Impostamo 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
```

### Script per l'installazione di IIS

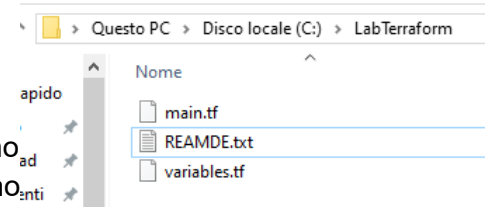
```
# - Installazione IIS
Enable-WindowsOptionalFeature -Online -FeatureName IIS-WebServerRole
# - Verifica installazione
Get-WindowsOptionalFeature -Online -FeatureName IIS-WebServerRole
```

Questi sono solo esempi, possiamo creare tutti gli script che vogliamo, l'importante è che vengano inseriti all'interno della macchina prima che questa venga convertita in template in quanto dopo questo termine per poter effettuare delle modifiche dovremmo ricreare un clone di questa.

Creiamo il template e siamo pronti a spostarci su terraform



# Programmazione script 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 omesso/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
}
```

## 3. Clonazione di template windows e esecuzione remota

### 3.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        = "W10Clone03"
  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         = "vmbr0"
  }
```

```
  provisioner "remote-exec" {
    connection {
      type         = "winrm"
      host         = "Win10Template"  # Use the VM's IP address or DNS
name
      user         = "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}"
    ]
  }
}
```

### 3.b. Provisioner `remote-exec`

- **Scopo**: Esegue uno script di configurazione su Windows dopo la creazione della VM.
- `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.

## 4. 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

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

- **Scopo**: Questa risorsa definisce un container LXC chiamato `my_container`, che verrà creato nel nodo Proxmox specificato.
- **Parametri**:

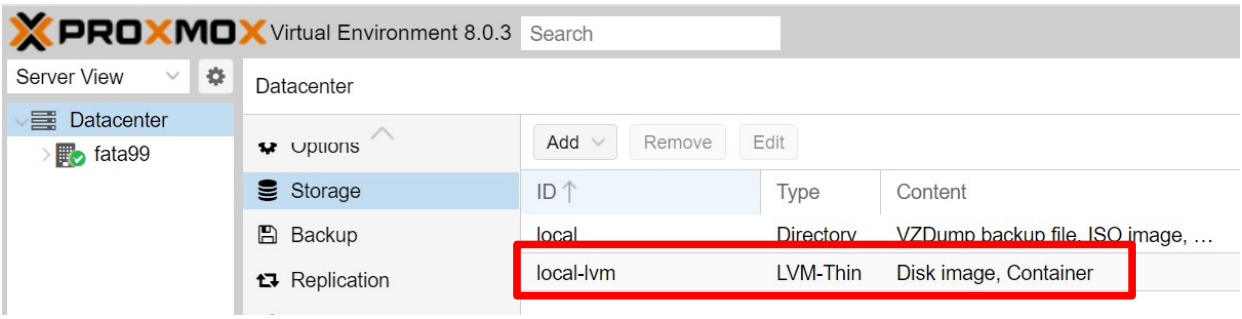
- **`target_node`:** Specifica il nodo Proxmox (in questo caso, fata99) su cui il container sarà creato.
- **`hostname`**: Imposta il nome host del container (`linux-basic`).
- **`ostemplate`**: Indica il template (precedentemente scaricati) da utilizzare per il container. In questo caso, si utilizza 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`.

### 4.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.

### 4.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.

### 4.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"
  ostemplate   = "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

}
```

## 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.

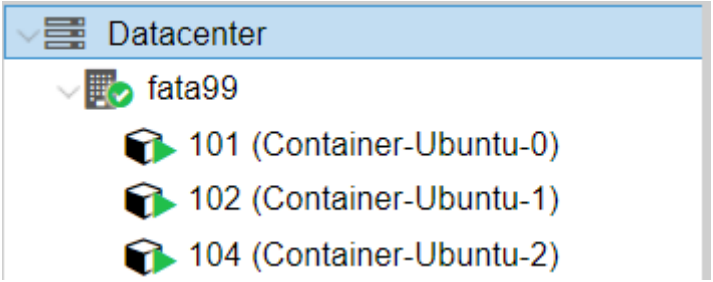
- `<TYPE>.<NAME>[<INDEX>]` o `module.<NAME>[<INDEX>]` (ad esempio, proxmox_lxc.my_container[0], ecc.) si riferisce a singole istanze

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}"
 ostemplate   = "local:vztmpl/ubuntu-23.04-standard_23.04-
1_amd64.tar.zst"
 password     = "Pa$$w0rd"
 unprivileged = true
 start        = true

 network {
  name   = "eth0"
  bridge = "vmbr0"
#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



## 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

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 otteremo 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:

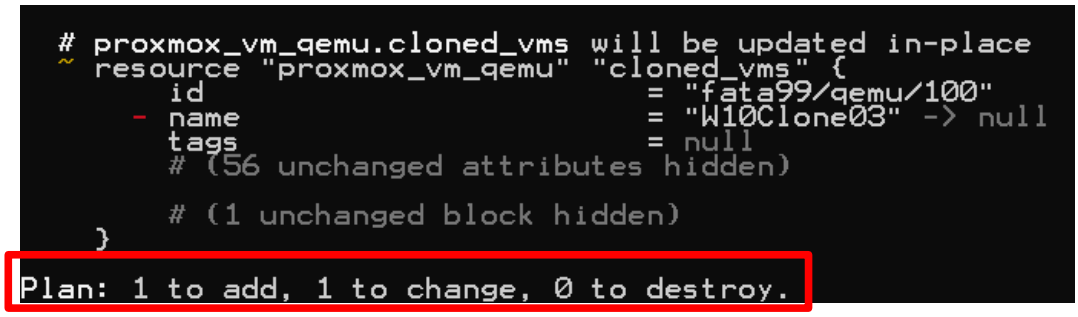
```
terraform validate
```

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

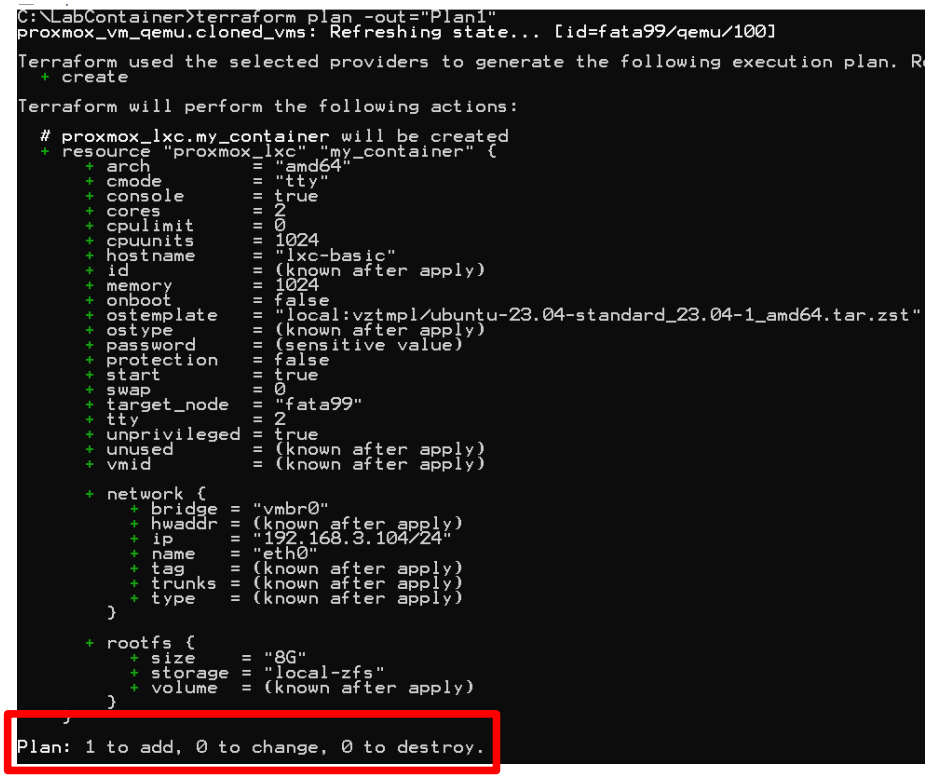
Prima di applicare le modifiche, è utile vedere quali cambiamenti verranno apportati all'infrastruttura. Utilizzare il comando plan per generare un piano di esecuzione, il valore del parametro -out rappresenta il nome arbitrario del piano, il parametro var permette di passare un parametro al main (il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

```
terraform plan -out="Plan1"

               -var="nome_variabile=valore"
```

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




In caso di esito positivo otteremo la seguente shermata



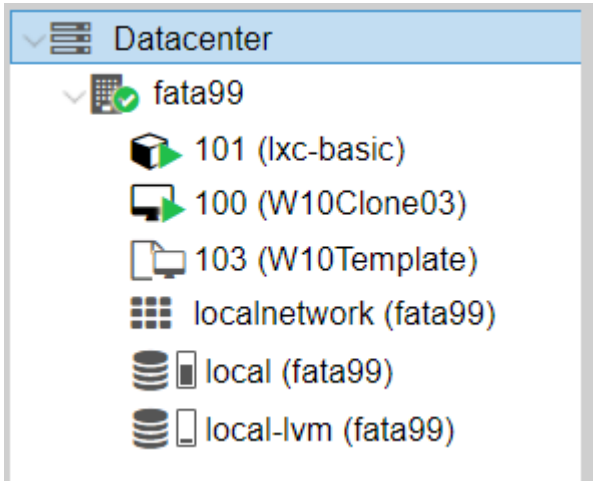

Se il piano generato è soddisfacente, è possibile procedere ad applicare le modifiche all'infrastruttura con il comando:

```
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
    hagroup              = 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/ubur
    ostype               = "ubuntu"
    password             = (sensitive value)
    protection           = false
    restore              = false
    searchdomain         = null
```




## Conclusione

E’ buona abitudine ripulire la cartella di lavoro durante la fase di debug dei file per evitare errori interni al sistema non dipendenti dal nostro script.

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.