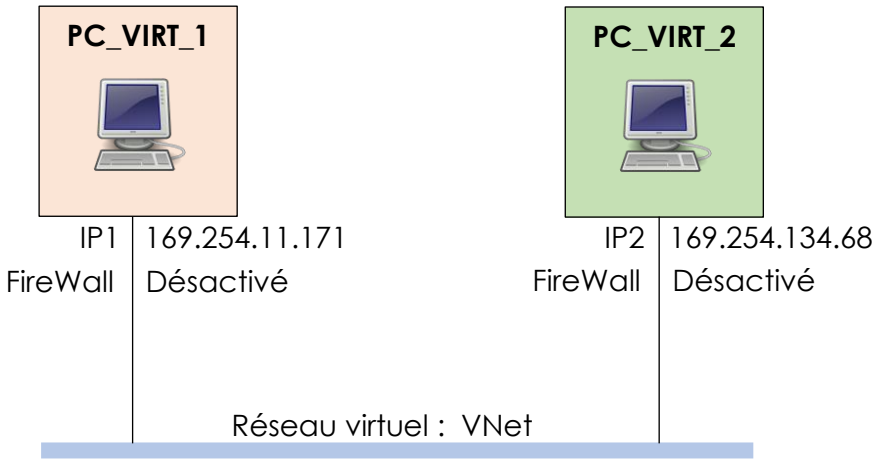
Mise en œuvre

L'utilisation de PowerShell afin de gérer une infrastructure demande à pouvoir exécuter des tâches sur des PC distants. Diverses **cmdlets** sont à notre disposition pour cet objectif et sont réunies sous le terme **Remoting**.

Partons de ce schéma de principe :



Remarques :

A) La configuration réseau de machines virtuelles sous **VirtualBox**, permet de choisir l'option **Réseau interne** et de lui assigner un nom. Dans le schéma ci-dessus, le nom choisi est **VNet**.

B) Pour permettre l'interopérabilité entre deux systèmes d'exploitation (Microsoft dans notre contexte), il faut permettre l'échange de données à distance. Sachant que la sécurité informatique est cruciale, notre manipulation n'a évidemment du sens qu'au sein d'une même entreprise sous la supervision de son service informatique.

Ici, par mesure de simplification, nous admettons de désactiver le **FireWall** de Windows.

C) Pour connaître l'adresse IP d'un PC (virtuel ou non), utilisez la commande :

```
PS> ipconfig /all
```

Cette commande affiche la configuration IP de chaque carte réseau du PC.

D) Pour tester la connexion IP entre 2 PC, utilisez la commande :

```
PS> ping Adresse_IP_PC_distant
```

Dans notre cas, de PC_Virt_1 à PC_Virt_2 :

```
PS> ping 169.254.134.68
```

WinRM

Windows Remote Management (WinRM) est l'implémentation Microsoft du protocole WS-Management, qui est un protocole SOAP (Simple Object Access Protocol) standard et compatible avec le pare-feu qui permet l'interopérabilité entre le matériel et les systèmes d'exploitation. Son activation est nécessaire pour réussir le **Remoting** avec PowerShell.

Ainsi, sur les deux PC virtuels, lancez la commande suivante, puis redémarrez-les :

```
PS> Enable-PSRemoting -SkipNetworkProfileCheck -Force
```

Autorisation d'exécution à distance

Pour terminer, il faut encore définir qu'un PC est autorisé à exécuter des instructions à distances. Dans notre cas, nous allons autoriser la réciprocité. Ainsi, sur chaque PC virtuel, lancez la commande :

```
PS> Set-Item WSMan:\localhost\Client\TrustedHosts *
```

Cas 1 : Ouvrir une session PowerShell vers un PC distant

```
# Voici la commande qui ouvre une session PowerShell sur PC_Virt_2 depuis PC_Virt_1.
# Une « pop-up » de connexion est ouverte et doit être complétée avec un login et un
# mot de passe valide par rapport au PC_Virt_2
PS> Enter-PSSession -ComputerName PC_Virt_2 -Credential Get-Credential

# Si tout se passe bien, le prompt change et nous confirme que l'on travaille sur le
# PC distant :
[PC_Virt_2]: PS>

# De là, tout est possible sur le PC distant. Exemple :
[PC_Virt_2]: PS> Get-ComputerInfo
```

Cas 2 : Créer une variable de session PowerShell vers un PC distant

```
# Voici la commande qui crée une session PowerShell sur PC_Virt_2 depuis PC_Virt_1
# et qui sauve la session dans une variable. Dans cet exemple, la variable est
# $session.
PS> $session = New-PSSession -ComputerName PC_Virt_2 -Credential Get-Credential

# Si tout se passe bien, le prompt ne change pas.
PS>

# De là, il est possible d'exécuter une commande à distance en recourant à la variable
# de session. Dans cet exemple, on crée le fichier example.log dans le dossier courant
# C:\Users\UserName\Documents\ du PC distant. Le fichier contiendra la date de
# l'instant de sa création.
PS> Invoke-Command -Session $session -ScriptBlock
    {New-Item -Name example.log -Path . -ItemType File -Value (Get-Date).ToString()}
```

Cas 3 : Exécuter un script Powershell sur un PC distant

```
# Reprendre le début du cas précédent afin de créer la variable $session.

# Admettons le script .\monScript.ps1 suivant

    $partInfo = Get-Partition
    Write-Output "DiskNumber `t| DriveLetter `t| Size [GB]"
    foreach ($part in $partInfo)
    {
        $sizeGB = [System.Math]::Round($part.Size / 1GB, 2)
        Write-Output "$($part.DiskNumber) `t`t  $($part.DriveLetter) `t`t  $sizeGB"
    }

# De là, il est possible d'exécuter ce script à distance en recourant à la variable
# de session.

PS> Invoke-Command -Session $session -FilePath .\monScript.ps1
```

Trucs

A) Vous souhaitez récupérer les données du PC distant sur le PC local

Alors reprenez le cas 2 et complétez la ligne de commande :

```
PS> $answer = Invoke-Command -Session $session -ScriptBlock
    {New-Item -Name example.log -Path . -ItemType File -Value (Get-Date).ToString()}

# De là, la variable locale $answer contiendra les données résultant du PC distant.
```

Ou reprenez le cas 3 et complétez la ligne de commande :

```
PS> Invoke-Command -Session $session -FilePath .\monScript.ps1 >> log.txt

# De là, le fichier local log.txt contiendra les données résultant du PC distant.
```

B) Vous souhaitez connaître l'adresse IP d'un PC à partir de son nom

Alors exécutez cette ligne de commande :

```
PS> [System.Net.Dns]::GetHostByName("PC_Virt_2").AddressList.IPAddressToString
```

C) Vous souhaitez connaître le nom d'un PC à partir de son adresse IP

Alors exécutez cette ligne de commande :

```
PS> [System.Net.Dns]::GetHostByAddress("169.254.134.68").HostName
```