Barre de navigation

Sommaire

Contexte

Cette page a pour but de décrire la mise en place d'une configuration minimale nécessaire pour un hôte supervisé par le pack **windows-by-WinRM__shinken.**

Les configurations suivantes sont destinées à des postes Windows configurés en "groupe de travail" (*Workgroup*).
La configuration pour un poste d'un "domaine" (*Active Directory*) peut différer légèrement. Plus de détails dans la section Procédure de configuration.

Versions Windows supportées

Voici la liste des versions Windows que la sonde peut superviser à distance :

- Windows 11
- Windows 10
- Windows Server 2025
- Windows Server 2022
- Windows Server 2019
- Windows Server 2016
- Windows Server 2012 R2

Procédure de configuration

La supervision d'un hôte par un **Poller Shinken** se fait par requête WinRM.

- Sur le **Poller shinken**, la sonde SNMP installé est responsable d'envoyer les requêtes et de traiter les réponses.
- Sur les hôtes à superviser, le service windows WinRM qui est responsable de répondre aux requêtes.

Le service Windows **WinRM** sur l'hôte supervisée permet l'exécution de commande à distance, et notamment la récupération d'informations sur la machine mise à disposition par **WMI** (Windows Management Instrumentations).

Alors, il est nécessaire de configurer le service **WinRM**, un utilisateur qui exécutera des commandes via **WinRM** et accèdera aux informations nécessaires aux checks.

Les commandes ci-dessous doivent être exécutées par un administrateur local de la machine (*sauf mention contraire*).

Quelques recommandations pour la configuration par domaine (*Active Directory*) :

La configuration pour un ordinateur membre d'un domaine est similaire.
Il est possible de configurer ses machines depuis son contrôleur de domaine.

Il est conseillé de configurer par GPO le démarrage de WinRM et ses accès d'authentification, ainsi que la langue de l'utilisateur de supervision et ses groupes de supervisions locaux associés. Les droits d'accès au service W32Time sont aussi disponibles par GPO.
Ensuite, il est possible avec un script de démarrage à utilisation unique de configurer les permissions *WinRM configSDDL* et les permissions WMI *Root/CIMv2* et *Root/StandardCimV2*.

Configuration de WinRM

Le service **WinRM** est installé par défaut sur les systèmes d'exploitations à partir de Windows 7 / Windows Server 2008 R2, mais celui-ci n'est pas nécessairement démarré et configuré.

Voici la commande à exécuter via PowerShell pour vérifier si le service est bien démarré :

```
Get-Service WinRM
```

Exemple :

```
C:\Users\Administrateur> Get-Service WinRM

Status   Name               DisplayName
------   ----               -----------
Running  WinRM              Gestion à distance de Windows (Gest...
```

Configuration minimale

WinRM dispose d'une commande de configuration intégrée qui permet entre autres de :

- Démarrer le service
- Activer le démarrage automatique de celui-ci
- Mettre en place l'écouteur (HTTP ou HTTPS) pour recevoir les requêtes des sondes Shinken
- Configurer le Firewall Windows pour autoriser les accès vers l'écouteur

Pour effectuer les actions de configuration, il suffit d'exécuter la commande suivante :

```
winrm quickconfig
```

Exemple :

```
C:\Users\Administrateur> winrm quickconfig

WinRM n'est pas configuré pour la gestion à distance de cet ordinateur.
Les modifications suivantes doivent être effectuées :

Créez un écouteur WinRM sur HTTP://* pour accepter les demandes de la gestion des services Web sur toutes
les adresses IP de cet ordinateur.

Activez l'exception de pare-feu WinRM.
Configurez LocalAccountTokenFilterPolicy pour attribuer des droits d'administration à distance à des
utilisateurs locaux.

Effectuer ces modifications [y/n] ? y

WinRM a été mis à jour pour la gestion à distance.

Écouteur WinRM créé sur HTTP://* pour accepter les demandes de la gestion des services Web sur toutes les
adresses IP de cet ordinateur.

Exception de pare-feu WinRM activée.
LocalAccountTokenFilterPolicy configuré pour attribuer des droits d'administration à distance à des
utilisateurs locaux.
```

Authentification

Selon le choix du mode d'authentification, il faut configurer l'authentification à WinRM.

Plus de détails sur l'authentification ici (NEW_PAGE - SPAC-27 - Modèles d'hôtes du pack windows-by-WinRM__shinken).

Par défaut, le service WinRM est configuré pour autoriser l'authentification "**Negotiate**" et désactive par défaut "**Basic**".

"**Negotiate**" est le protocole de négociation nécessaire pour communiquer via **ntlm.**

ntlm

Vous pouvez activer l'authentification **ntlm** en configurant "**Negotiate**" avec la commande suivante :

```
winrm set winrm/config/service/auth '@{Negotiate="true"}'
```

basic

> L'utilisation du protocole « Basic » n'est pas recommandée lorsque des modes d'authentification plus sécurisés sont disponibles sur le système.

```
winrm set winrm/config/service/auth '@{Basic="true"}'
```

L'authentification **basic** n'ajoute pas d'encryption, alors il est necessaire de configurer l'hôte pour accepter les communications non-chiffrées :

```
winrm set winrm/config/service '@{AllowUnencrypted="true"}'
```

Configuration de l'utilisateur

À l'image des commandes WMI (*voir la page Modèle windows-by-WMI__ntlmv2*), les commandes WinRM requièrent une authentification au préalable afin de récupérer les informations de supervision sur l'hôte windows. (*-u "$_HOSTDOMAINUSER$" -p "$_HOSTDOMAINPASSWORD$"*)

L'utilisation du compte administrateur du poste Windows permet d'obtenir toutes les informations du système, car celui-ci possède par défaut tous les droits d'accès (*WMI, DCOM, WinRM, etc.*).

Cependant, pour des raisons de sécurité, il n'est pas conseillé de l'utiliser pour effectuer la supervision. Il faut alors créer un utilisateur qui exécutera les commandes demandées par les sondes, avec uniquement les droits nécessaires.

Qu'il s'agisse d'un compte local ou d'un compte AD, voici la procédure pour créer un nouvel utilisateur dédié au monitoring avec les droits suffisant pour collecter les informations nécessaires au fonctionnement des sondes fournies par Shinken.

Création de l'utilisateur

La création est possible directement sur le poste local Windows, ou sur l'Active Directory.

Par interface

Cela s'effectue dans la console de "**Gestion de l'ordinateur**" (*compmgmt.msc*) puis dans **Utilisateurs et groupes locaux > Utilisateurs,** clic droit et **Nouvel utilisateur...**

? Unknown Attachment

Par ligne de commande

Dans un PowerShell :

```
net user "shinken" "mon_motdepasse" /ADD
```

> Remplacer ici "mon_motdepasse".

Configuration de la langue

Pour assurer l'interprétation des commandes Windows par la sonde, il est nécessaire de configurer la langue du nouvel utilisateur.

Par interface

Installer la langue (depuis le compte administrateur)

Il est nécessaire d'installer la langue "**English (United-States)**" depuis le compte administrateur pour les windows suivants :

- windows server 2019
- windows server 2022

Sinon vous pouvez sauter cette étape et installer la langue pour le nouvel utilisateur de supervision créé.

La première étape est de télécharger la langue "**English (United-States)**" depuis le compte de l'administrateur.

Lancer les paramètres Windows puis accéder à la catégorie "**Heure et Langue**".

? Unknown Attachment

Ensuite, accéder à la sous-catégorie "**Langue**" puis sélectionner "**Ajouter une langue**"

? Unknown Attachment

Dans la nouvelle fenêtre, choisir "**English (United-States)**", et installer la langue.

? Unknown Attachment

? Unknown Attachment

La langue sera désormais listée. Cliquer dessus et sélectionner "**Option**", puis s'assurer que le **module linguistique** est installé.

? Unknown Attachment ? Unknown Attachment

La langue "**English (United-States)**" est désormais disponible sur la machine et il est nécessaire de l'ajouter depuis le compte de supervision.

Configurer la langue (depuis le compte de supervision)

Il est nécessaire de se connecter au nouveau compte créer pour changer la langue de l'utilisateur.

Répéter les mêmes étapes que pour l'installation de la langue par l'administrateur.

Lors de l'installation, sélectionner "**Installer le module linguistique et définir comme ma langue d'affichage windows**".

Une fois installé, s'assurer que :

- La langue "**English (United-States)**" en première position de la liste des langues préférée.
- La langue "**English (United-States)**" dans la section "**Langue d'affichage de Windows**".

? Unknown Attachment

? Unknown Attachment

Par ligne de commande

Après avoir ouvert un PowerShell en mode administrateur :

La commande ci-dessous va installer le pack de langage nécessaire à la sonde.

> Cette commande n'est disponible que sur Windows 11 et Windows Server 2025.
> Il vous faudra sinon télécharger la langue manuellement.

Cette opération peut prendre plusieurs minutes.

```
Install-Language -Language en-US
```

Une fois le pack de langue téléchargé, il est possible de l'appliquer au nouvel utilisateur crée :

Il faut remplacer le nom d'utilisateur et le mot de passe ligne 3 et 4.

```
& {
$LangCode = "en-US"
$UserName = "shinken"
$Password = "my_password"

# La langue est elle disponible ?
$languages = Get-WinUserLanguageList
if ($languages.LanguageTag -notcontains $LangCode) {
    Write-Host "Langue $LangCode non présente, ajout..." -ForegroundColor Gray
    $languages.Add($LangCode)
    Set-WinUserLanguageList $languages -Force
    Write-Host "Langue $LangCode ajoutée." -ForegroundColor Green
} else {
    Write-Host "Langue $LangCode déjà disponible." -ForegroundColor Green
}
Write-Host ""

$Computer = $env:COMPUTERNAME
$SID = (Get-WmiObject Win32_UserAccount -Filter "Name='$UserName' AND Domain='$Computer'").SID
if (-not $SID) { throw "Impossible de récupérer le SID pour $UserName" }

# Recupération du profil
$UserProfile = Get-CimInstance Win32_UserProfile | Where-Object SID -eq $SID

# Le profil peut ne pas exister si l'utilisateur a été créé mais jamais été connecté.
# Tentative de connection à l'utilisateur ...
if (-not $UserProfile) {
        $SecurePassword = ConvertTo-SecureString $Password -AsPlainText -Force
        $Credential = New-Object System.Management.Automation.PSCredential ($UserName, $SecurePassword)
    try {
        Start-Process -FilePath "whoami.exe" -Credential $Credential -WindowStyle Hidden -Wait
    } catch {
        Write-Host " Impossible de lancer un logon test pour $UserName ($_)" -ForegroundColor Red
    }

    # Récupération à nouveau du profil
    $UserProfile = Get-CimInstance Win32_UserProfile | Where-Object SID -eq $SID
}

if (-not $UserProfile) { throw "Profil introuvable pour $UserName" }

$HivePath = Join-Path $UserProfile.LocalPath "NTUSER.DAT"
if (-not (Test-Path $HivePath)) { throw "Profil non initialisé (NTUSER.DAT introuvable) : $HivePath" }

$regRelative = "Control Panel\Desktop"
$valueName = "PreferredUILanguages"
[string[]]$LangArray = @($LangCode)

$IsLoaded = Test-Path "Registry::HKEY_USERS\$SID"
if ($IsLoaded) {
    $fullPath = "Registry::HKEY_USERS\$SID\$regRelative"
```

```
    if (-not (Test-Path $fullPath)) { New-Item -Path $fullPath -Force | Out-Null }
    if (-not (Get-ItemProperty -Path $fullPath -Name $valueName -ErrorAction SilentlyContinue)) {
        New-ItemProperty -Path $fullPath -Name $valueName -Value $LangArray -PropertyType MultiString -Force
| Out-Null
    } else {
        Set-ItemProperty -Path $fullPath -Name $valueName -Value $LangArray
    }
} else {
    Write-Host "Utilisateur non connecté. Chargement de la ruche..." -ForegroundColor Gray
    reg load "HKU\TempHive" $HivePath | Out-Null
    try {
        $tempPath = "Registry::HKEY_USERS\TempHive\$regRelative"
        if (-not (Test-Path $tempPath)) { New-Item -Path $tempPath -Force | Out-Null }
        if (-not (Get-ItemProperty -Path $tempPath -Name $valueName -ErrorAction SilentlyContinue)) {
            New-ItemProperty -Path $tempPath -Name $valueName -Value $LangArray -PropertyType MultiString -
Force | Out-Null
        } else {
            Set-ItemProperty -Path $tempPath -Name $valueName -Value $LangArray
        }
    } finally {
        reg unload "HKU\TempHive" | Out-Null
        Write-Host "Ruche déchargée." -ForegroundColor Gray
    }
}

Write-Host "Langue préférée utilisateur définie sur $LangCode (REG_MULTI_SZ)." -ForegroundColor Green
}
```

Ensuite, il est nécessaire de **se déconnecter puis se reconnecter** au nouvel utilisateur afin de correctement appliquer le changement de langue.

Une fois l'opération réalisée, il est possible de se reconnecter au compte administrateur Windows et de poursuivre la configuration.

Configuration des groupes

Par interface

Une fois le nouvel utilisateur créé sur le poste client ou sur le domaine, et sa langue configurée, il faut ajouter l'utilisateur dans les groupes suivants : **U tilisateurs de gestion à distance** et **Utilisateurs de l'Analyseur de performances** sur le poste Windows concerné.

Cela s'effectue dans la console de "**Gestion de l'ordinateur**" (*compmgmt.msc*) puis dans **Utilisateurs et groupes locaux > Groupes,** clic droit sur le groupe puis **Propriétés**. Une nouvelle fenêtre s'ouvre, cliquer sur **Ajouter…** :

Sur certaines distributions inférieures à Windows 2012, le groupe "**Utilisateurs de gestion à distance**" et sa fonction n'existent pas.

? Unknown Attachment

Par ligne de commande

Dans un PowerShell :

- Si la langue de l'utilisateur administrateur est anglais :

```
net localgroup "Remote Management Users" "shinken" /ADD
net localgroup "Performance Monitor Users" "shinken" /ADD
```

- Si la langue de l'utilisateur administrateur est français :

```
net localgroup "Utilisateurs de gestion à distance" "shinken" /ADD
net localgroup "Utilisateurs de l'Analyseur de performances" "shinken" /ADD
```

Il est possible que la commande "net localgroup "Utilisateurs de l'Analyseur de performances" "shinken" /ADD" ne fonctionne pas lors d'un copier-coller.

Il y a un bug avec le terminal PowerShell sur le copier-coller qui ne retranscrira pas le symbole apostrophe ' lorsque l'action "coller" est déclenché avec un clic droit dans le terminal.
Il faut utiliser le raccourci "CTRL+V" affin de coller correctement la commande
Sinon, il est possible de générer le caractère spécial apostrophe ' avec le raccourci "ALT + 0146". (*Différent de l'apostrophe de la touche 4*).

Example du bug du copier-coller dans un terminal PowerShell Administrateur où l'apostrophe n'est pas retranscrit :

? Unknown Attachment

.

Configuration des permissions

Permissions WinRM pour l'utilisateur

Il est nécessaire d'ajouter les droits de lecture et d'exécution aux commandes WinRM au nouvel utilisateur :

Par interface

Dans une console PowerShell, exécuter la commande suivante :

```
winrm configSDDL default
```

Une nouvelle fenêtre s'ouvre. Dans celle-ci, ajouter le nouvel utilisateur et cliquer sur **Ajouter...** et attribuer les droits :

- Lecture(Get,Enumerate,Subscribe)
- Exécution(Invoke)

En cochant les cases correspondantes dans le tableau des droits situé en dessous de la liste des utilisateurs (*cliquer sur l'utilisateur au préalable*).

? Unknown Attachment

Par ligne de commande

Dans un PowerShell :

```
$user = "shinken"

$GENERIC_READ = 0x80000000
$GENERIC_EXECUTE = 0x20000000

$user_sid = (New-Object -TypeName System.Security.Principal.NTAccount -ArgumentList $user).Translate([System.
Security.Principal.SecurityIdentifier])

# get the existing SDDL of the WinRM listener
$sddl = (Get-Item -Path WSMan:\localhost\Service\RootSDDL).Value

# convert the SDDL string to a SecurityDescriptor object
$sd = New-Object -TypeName System.Security.AccessControl.CommonSecurityDescriptor -ArgumentList $false,
$false, $sddl

# apply a new DACL to the SecurityDescriptor object
$sd.DiscretionaryAcl.AddAccess(
    [System.Security.AccessControl.AccessControlType]::Allow,
    $user_sid,
    ($GENERIC_READ -bor $GENERIC_EXECUTE),
    [System.Security.AccessControl.InheritanceFlags]::None,
    [System.Security.AccessControl.PropagationFlags]::None
)

# get the SDDL string from the changed SecurityDescriptor object
$new_sddl = $sd.GetSddlForm([System.Security.AccessControl.AccessControlSections]::All)

# apply the new SDDL to the WinRM listener
Set-Item -Path WSMan:\localhost\Service\RootSDDL -Value $new_sddl -Force
```

Autorisation aux objets CIM

Par interface

La sonde va demander les informations systèmes via les objets CIM, il est nécessaire d'ajouter les droits à l'utilisateur.

> Cette section n'est pas nécessaire si les machines supervisées sont configurées avec un Active Directory.

Il faut en premier temps lancer la fenêtre de contrôle WMI avec la commande :

```
wmimgmt.msc
```

Une fois lancé, clic droit sur **Contrôle WMI (local)** puis sélectionner **Propriétés**.

? Unknown Attachment

Accéder à la section Sécurité, puis dans l'arborescence ci-dessous, sélectionner **Root > CIMV2** puis cliquer sur le bouton **Sécurité** situé en bas à droite.

? Unknown Attachment

Enfin, ajouter l'utilisateur afin de lui appliquer de nouveaux droits, puis cocher **Activer le compte** et **Appel à distance autorisé**.

? Unknown Attachment

Ensuite, répétez l'opération pour StandardCimV2

? Unknown Attachment

Par ligne de commande

Dans un PowerShell. Cette commande va écrire un script PowerShell qu'il permet d'ajouter les droits nécessaires à la lecture des objets CIM.

```
$script_name = ".\Set-WmiNamespaceSecurity.ps1"

@'
# Set-WmiNamespaceSecurity.ps1
# Example: Set-WmiNamespaceSecurity root/cimv2 add steve Enable,RemoteAccess

Param ( [parameter(Mandatory=$true,Position=0)][string] $namespace,
    [parameter(Mandatory=$true,Position=1)][string] $operation,
    [parameter(Mandatory=$true,Position=2)][string] $account,
    [parameter(Position=3)][string[]] $permissions = $null,
    [bool] $allowInherit = $false,
    [bool] $deny = $false,
    [string] $computerName = ".",
    [System.Management.Automation.PSCredential] $credential = $null)

Process {
    $ErrorActionPreference = "Stop"
    Function Get-AccessMaskFromPermission($permissions) {
        $WBEM_ENABLE = 1
        $WBEM_METHOD_EXECUTE = 2
        $WBEM_FULL_WRITE_REP = 4
        $WBEM_PARTIAL_WRITE_REP = 8
        $WBEM_WRITE_PROVIDER = 0x10
        $WBEM_REMOTE_ACCESS = 0x20
        $WBEM_RIGHT_SUBSCRIBE = 0x40
        $WBEM_RIGHT_PUBLISH = 0x80
        $READ_CONTROL = 0x20000
        $WRITE_DAC = 0x40000

        $WBEM_RIGHTS_FLAGS = $WBEM_ENABLE,$WBEM_METHOD_EXECUTE,$WBEM_FULL_WRITE_REP,`

            $WBEM_PARTIAL_WRITE_REP,$WBEM_WRITE_PROVIDER,$WBEM_REMOTE_ACCESS,`

            $READ_CONTROL,$WRITE_DAC

        $WBEM_RIGHTS_STRINGS = "Enable","MethodExecute","FullWrite","PartialWrite",`
            "ProviderWrite","RemoteAccess","ReadSecurity","WriteSecurity"

        $permissionTable = @{}
        for ($i = 0; $i -lt $WBEM_RIGHTS_FLAGS.Length; $i++) {
            $permissionTable.Add($WBEM_RIGHTS_STRINGS[$i].ToLower(), $WBEM_RIGHTS_FLAGS[$i])
        }

        $accessMask = 0
        foreach ($permission in $permissions) {
            if (-not $permissionTable.ContainsKey($permission.ToLower())) {
                throw "Unknown permission: $permission`nValid permissions: $($permissionTable.Keys)"
            }
            $accessMask += $permissionTable[$permission.ToLower()]
        }
        $accessMask
    }

    if ($PSBoundParameters.ContainsKey("Credential")) {
        $remoteparams = @{ComputerName=$computer;Credential=$credential}
    } else {
        $remoteparams = @{}
    }

    $invokeparams = @{Namespace=$namespace;Path="__systemsecurity=@"} + $remoteParams
    $output = Invoke-WmiMethod @invokeparams -Name GetSecurityDescriptor
    if ($output.ReturnValue -ne 0) {
        throw "GetSecurityDescriptor failed: $($output.ReturnValue)"
    }

    $acl = $output.Descriptor
    $OBJECT_INHERIT_ACE_FLAG = 0x1
```

```
$CONTAINER_INHERIT_ACE_FLAG = 0x2
$computerName = (Get-WmiObject @remoteparams Win32_ComputerSystem).Name

if ($account.Contains('\')) {
    $domainaccount = $account.Split('\')
    $domain = $domainaccount[0]
    if (($domain -eq ".") -or ($domain -eq "BUILTIN")) {
        $domain = $computerName
    }
    $accountname = $domainaccount[1]
} elseif ($account.Contains('@')) {
    $domainaccount = $account.Split('@')
    $domain = $domainaccount[1].Split('.')[0]
    $accountname = $domainaccount[0]
} else {
    $domain = $computerName
    $accountname = $account
}

$getparams = @{Class="Win32_Account";Filter="Domain='$domain' and Name='$accountname'"} + $remoteParams
$win32account = Get-WmiObject @getparams
if ($win32account -eq $null) {
    throw "Account was not found: $account"
}

switch ($operation) {
    "add" {
        if ($permissions -eq $null) {
            throw "-Permissions must be specified for an add operation"
        }

        $accessMask = Get-AccessMaskFromPermission($permissions)
        $ace = (New-Object System.Management.ManagementClass("win32_Ace")).CreateInstance()
        $ace.AccessMask = $accessMask

        if ($allowInherit) {
            $ace.AceFlags = $OBJECT_INHERIT_ACE_FLAG + $CONTAINER_INHERIT_ACE_FLAG
        } else {
            $ace.AceFlags = 0
        }

        $trustee = (New-Object System.Management.ManagementClass("win32_Trustee")).CreateInstance()
        $trustee.SidString = $win32account.Sid
        $ace.Trustee = $trustee

        $ACCESS_ALLOWED_ACE_TYPE = 0x0
        $ACCESS_DENIED_ACE_TYPE = 0x1

        if ($deny) {
            $ace.AceType = $ACCESS_DENIED_ACE_TYPE
        } else {
            $ace.AceType = $ACCESS_ALLOWED_ACE_TYPE
        }
        $acl.DACL += $ace.psobject.immediateBaseObject
    }
    "delete" {
        if ($permissions -ne $null) {
           throw "Permissions cannot be specified for a delete operation"
        }

        [System.Management.ManagementBaseObject[]]$newDACL = @()
        foreach ($ace in $acl.DACL) {
            if ($ace.Trustee.SidString -ne $win32account.Sid) {
                $newDACL += $ace.psobject.immediateBaseObject
            }
        }
        $acl.DACL = $newDACL.psobject.immediateBaseObject
    }
    default {
        throw "Unknown operation: $operation`nAllowed operations: add delete"
    }
```

```
    }

    $setparams = @{Name="SetSecurityDescriptor";ArgumentList=$acl.psobject.immediateBaseObject} +
$invokeParams
    $output = Invoke-WmiMethod @setparams
    if ($output.ReturnValue -ne 0) {
        throw "SetSecurityDescriptor failed: $($output.ReturnValue)"
    }

}


'@ | Set-Content $script_name
```

Ensuite, executer le script deux fois avec les paramètres suivants :

```
$user = "shinken"
.\Set-WmiNamespaceSecurity.ps1 "root\cimv2"  add $user Enable,RemoteAccess
.\Set-WmiNamespaceSecurity.ps1 "root\standardcimv2" add $user Enable,RemoteAccess
```

Il est possible par la suite de supprimer le script :

```
rm .\Set-WmiNamespaceSecurity.ps1
```

Autorisation au service W32Time

Afin de récupérer les informations de temps, il est nécessaire au nouvel utilisateur d'avoir certains accès en lecture au service de temps de windows W32Time.

Par ligne de commande

Dans un PowerShell :

```
$user = "shinken"

$sid = (New-Object System.Security.Principal.NTAccount($user)).Translate([System.Security.Principal.
SecurityIdentifier]).Value
$currentSddl = & sc.exe sdshow w32time
# Ajoute les droits suivants à l'utilisateur
# CC – SERVICE_QUERY_CONFIG
# LC – SERVICE_QUERY_STATUS
# LO – SERVICE_INTERROGATE
$newAce = "(A;;CCLCLO;;;${sid})"
$newSddl = $currentSddl + $newAce

sc.exe sdset w32time "$newSddl"

Restart-Service w32time
```

Appliquer les permissions

Les nouvelles permissions attribuées au nouvel utilisateur seront appliquées lorsque de nouveaux tokens de connections seront générés. Il est possible de :

- Attendre quelques minutes que les anciens tokens expirent et que de nouveaux, ayant de nouvelles permissions, soient générés
- Redémarrer la machine

Configuration par script

Voici un script permettant de configurer rapidement par ligne de commandes un poste Windows.
Il reprend l'entièreté des commandes ci-dessus.

Le script va :

- Installer la langue Anglaise (Etats-Unis) uniquement sur Windows 11 ou Windows Server 2025. Il est necessaire d'installer la langue par interface autrement.
- Créer un utilisateur, et l'ajouter dans les groupes nécessaires.
- Configurer la langue du nouvel utilisateur de supervision
- Configurer WinRM pour l'authentification Basic ou Negotiate, puis HTTP.
- Configurer l'utilisateur de supervision pour executer des commandes WinRM
- Configurer l'utilisateur de supervision pour récupérer les informations WMI/CIM root\cimv2
- Configurer l'utilisateur de supervision pour récupérer les informations WMI/CIM root\standardcimv2
- Configurer l'utilisateur de supervision pour récupérer les informations de W32Time, le service NTP de Windows.

> Le script necessite les droits administrateurs pour l'execution

Le script est conçu pour être copier-coller dans un terminal powershell. Il est aussi possible de l'enregistrer dans un fichier (.ps1) avant de l'exécuter.

```
& {

param(
    [Parameter(Mandatory = $true)][string]$UserName,
    [Parameter(Mandatory = $true)][string]$Password,
    [string]$Basic,
    [string]$Negotiate,
    [string]$AllowUnencrypted
)

$LangCode = "en-US"
$Steps = New-Object System.Collections.ArrayList

function Parse-Bool($value) {
    try {
        return [System.Convert]::ToBoolean($value.ToString().Trim())
    } catch {
        throw "Paramètre booléen invalide: '$value'"
    }
}

if (-not $Basic) {
    $Basic = Read-Host "Basic (true or false)"
}
if (-not $Negotiate) {
    $Negotiate = Read-Host "Negotiate (true or false)"
}
if (-not $AllowUnencrypted) {
    $AllowUnencrypted = Read-Host "AllowUnencrypted (true or false)"
}

$ParsedBasic = Parse-Bool $Basic
$ParsedNegotiate = Parse-Bool $Negotiate
$ParsedAllowUnencrypted = Parse-Bool $AllowUnencrypted

function Add-StepResult {
    param([string]$Name,[bool]$Ok,[string]$Msg="")
    $icon = if ($Ok) {"OK"} else {"KO"}
    $status = if ($Ok) {"Success"} else {"Failed"}
    [void]$Steps.Add([PSCustomObject]@{ Step=$Name; Status=$status; Message="$icon $Msg" })
}

function Execute-Step {
    param([string]$Name,[scriptblock]$Action)
    try {
        & $Action
        Add-StepResult -Name $Name -Ok $true -Msg "Étape réussie"
    } catch {
        Add-StepResult -Name $Name -Ok $false -Msg ("Erreur : " + $_.Exception.Message)
```

```
        throw "Étape $Name échouée"
    }
}

function Install-LanguageIfNeeded {
    param([string]$LangCode)
    Write-Host "[1/8] Installation de la langue $LangCode et configuration" -ForegroundColor Yellow
    try {
        Install-Language -Language $LangCode -ExcludeFeatures
    }
    catch {
        Write-Warning "Impossible d'installer la langue $LangCode. Veuillez l'installer par interface
graphique depuis un compte admin"
    }

    $languages = Get-WinUserLanguageList
    if ($languages.LanguageTag -notcontains $LangCode) {
        Write-Host "Langue $LangCode non présente, ajout..." -ForegroundColor Gray
        $languages.Add($LangCode)
        Set-WinUserLanguageList $languages -Force
        Write-Host "Langue $LangCode ajoutée." -ForegroundColor Green
    } else {
        Write-Host "Langue $LangCode déjà disponible." -ForegroundColor Green
    }
    Write-Host ""
}

function Configure-WinRM {
    Write-Host "[2/8] Configuration rapide de WinRM" -ForegroundColor Yellow
    winrm quickconfig -q
    if ($LASTEXITCODE -ne 0) { throw "WinRM quickconfig a échoué ($LASTEXITCODE)" }
    Write-Host "WinRM configuré." -ForegroundColor Green
    Write-Host ""
}

function Add-UserToExistingGroup {
    param(
        [string]$UserName,
        [string]$PrimaryGroup,
        [string]$FallbackGroup
    )

    $groupUsed = $null

    foreach ($grp in @($PrimaryGroup,$FallbackGroup)) {
        $output = net localgroup "$grp" 2>$null
        if ($LASTEXITCODE -eq 0) {
            $groupUsed = $grp
            break
        }
    }

    if (-not $groupUsed) {
        throw "Aucun des groupes '$PrimaryGroup' ou '$FallbackGroup' n'existe."
    }

    if ($output -match "^\s*$UserName\s*$") {
        Write-Host "Utilisateur $UserName déjà dans le groupe $groupUsed" -ForegroundColor Gray
    } else {
        & net localgroup "$groupUsed" $UserName /ADD | Out-Null
        if ($LASTEXITCODE -ne 0) {
            throw "Échec ajout utilisateur $UserName dans le groupe $groupUsed ($LASTEXITCODE)"
        } else {
            Write-Host "Utilisateur $UserName ajouté au groupe $groupUsed" -ForegroundColor Green
        }
    }
}

function Create-User {
    param([string]$UserName, [string]$Password)
```

```
    Write-Host "[3/8] Création de l'utilisateur $UserName" -ForegroundColor Yellow

    $exists = net user $UserName 2>$null
    if ($LASTEXITCODE -ne 0) {
        & net user $UserName $Password /ADD | Out-Null
        if ($LASTEXITCODE -ne 0) {
            throw "Échec création utilisateur via 'net user' ($LASTEXITCODE)"
        }
    } else {
        Write-Host "Utilisateur déjà existant : $UserName" -ForegroundColor Gray
    }

    Add-UserToExistingGroup -UserName $UserName -PrimaryGroup "Remote Management Users" -FallbackGroup
"Utilisateurs de gestion à distance"
    Add-UserToExistingGroup -UserName $UserName -PrimaryGroup "Performance Monitor Users" -FallbackGroup
"Utilisateurs de l’Analyseur de performances"

    $SecurePassword = ConvertTo-SecureString $Password -AsPlainText -Force
    $Credential = New-Object System.Management.Automation.PSCredential ($UserName, $SecurePassword)
    try {
        Start-Process -FilePath "whoami.exe" -Credential $Credential -WindowStyle Hidden -Wait
    } catch {
        Write-Host " Impossible de lancer un logon test pour $UserName ($_)" -ForegroundColor Red
    }

    Write-Host "Utilisateur $UserName créé/validé et groupes assignés." -ForegroundColor Green
}


function Configure-UserLanguage {
    param(
        [string]$UserName,
        [string]$Password,
        [string]$LangCode
    )

    Write-Host "[4/8] Configuration de la langue pour l'utilisateur $UserName" -ForegroundColor Yellow

    $Computer = $env:COMPUTERNAME
    $SID = (Get-WmiObject Win32_UserAccount -Filter "Name='$UserName' AND Domain='$Computer'").SID
    if (-not $SID) { throw "Impossible de récupérer le SID pour $UserName" }

    $UserProfile = Get-CimInstance Win32_UserProfile | Where-Object SID -eq $SID
    if (-not $UserProfile) { throw "Profil introuvable pour $UserName" }

    $HivePath = Join-Path $UserProfile.LocalPath "NTUSER.DAT"
    if (-not (Test-Path $HivePath)) { throw "Profil non initialisé (NTUSER.DAT introuvable) : $HivePath" }

    $regRelative = "Control Panel\Desktop"
    $valueName = "PreferredUILanguages"
    [string[]]$LangArray = @($LangCode)

    $IsLoaded = Test-Path "Registry::HKEY_USERS\$SID"
    if ($IsLoaded) {
        $fullPath = "Registry::HKEY_USERS\$SID\$regRelative"
        if (-not (Test-Path $fullPath)) { New-Item -Path $fullPath -Force | Out-Null }
        if (-not (Get-ItemProperty -Path $fullPath -Name $valueName -ErrorAction SilentlyContinue)) {
            New-ItemProperty -Path $fullPath -Name $valueName -Value $LangArray -PropertyType MultiString -
Force | Out-Null
        } else {
            Set-ItemProperty -Path $fullPath -Name $valueName -Value $LangArray
        }
    } else {
        Write-Host "Utilisateur non connecté. Chargement de la ruche..." -ForegroundColor Gray
        reg load "HKU\TempHive" $HivePath | Out-Null
        try {
            $tempPath = "Registry::HKEY_USERS\TempHive\$regRelative"
            if (-not (Test-Path $tempPath)) { New-Item -Path $tempPath -Force | Out-Null }
            if (-not (Get-ItemProperty -Path $tempPath -Name $valueName -ErrorAction SilentlyContinue)) {
                New-ItemProperty -Path $tempPath -Name $valueName -Value $LangArray -PropertyType
MultiString -Force | Out-Null
```

```
            } else {
                Set-ItemProperty -Path $tempPath -Name $valueName -Value $LangArray
            }
        } finally {
            reg unload "HKU\TempHive" | Out-Null
            Write-Host "Ruche déchargée." -ForegroundColor Gray
        }
    }

    Write-Host "Langue préférée utilisateur définie sur $LangCode (REG_MULTI_SZ)." -ForegroundColor Green
}

function Configure-WinRMAuth {
    param([bool]$Basic, [bool]$Negotiate, [bool]$AllowUnencrypted)

    Write-Host "[5/8] Configuration de l'authentification WinRM" -ForegroundColor Yellow

    if (!$Basic -and !$Negotiate) {
        throw "Aucune méthode d'authentification n'est paramétrée. Veuillez choisir Basic ou Negotiate."
    }

    if ($Basic -and !$AllowUnencrypted) {
        throw "L'authentification Basic nécessite d'authoriser la connection non chiffré (AllowUnencrypted).
Le support HTTPS n'est pas disponible pour le moment."
    }

    if (!$Basic -and $Negotiate -and $AllowUnencrypted) {
        throw "Negotiate ne nécessite pas une connection chiffré (AllowUnencrypted). Il est déconseillé de
l'activer."
    }

    if ($Basic) {
        winrm set winrm/config/service/auth '@{Basic="true"}'
    }

    if ($Negotiate) {
        winrm set winrm/config/service/auth '@{Negotiate="true"}'
    }

    if ($AllowUnencrypted) {
        winrm set winrm/config/service '@{AllowUnencrypted="true"}'
    }

    Write-Host "Configuration d'authentification définie." -ForegroundColor Green
    Write-Host ""
}

function Configure-WinRMSDDL {
    param([string]$UserName)
    Write-Host "[6/8] Attribution des droits WinRM (SDDL)" -ForegroundColor Yellow

    $GENERIC_READ = 0x80000000
    $GENERIC_EXECUTE = 0x20000000
    $user_sid = (New-Object System.Security.Principal.NTAccount $UserName).Translate([System.Security.
Principal.SecurityIdentifier])

    $sddl = (Get-Item -Path WSMan:\localhost\Service\RootSDDL).Value
    $sd = New-Object System.Security.AccessControl.CommonSecurityDescriptor $false, $false, $sddl
    $sd.DiscretionaryAcl.AddAccess(
        [System.Security.AccessControl.AccessControlType]::Allow,
        $user_sid,
        ($GENERIC_READ -bor $GENERIC_EXECUTE),
        [System.Security.AccessControl.InheritanceFlags]::None,
        [System.Security.AccessControl.PropagationFlags]::None
    )
    $new_sddl = $sd.GetSddlForm([System.Security.AccessControl.AccessControlSections]::All)
    Set-Item -Path WSMan:\localhost\Service\RootSDDL -Value $new_sddl -Force
    Write-Host "Droits SDDL appliqués." -ForegroundColor Green
    Write-Host ""
}
```

```
Function Configure-CIMSDDL {
    param(
        [parameter(Mandatory=$true,Position=0)][string] $namespace,
        [parameter(Mandatory=$true,Position=1)][string] $operation,
        [parameter(Mandatory=$true,Position=2)][string] $account,
        [parameter(Position=3)][string[]] $permissions = $null,
        [bool] $allowInherit = $false,
        [bool] $deny = $false,
        [string] $computerName = ".",
        [System.Management.Automation.PSCredential] $credential = $null
    )

    Write-Host "[7/8] Attribution des droits WMI/CIM" -ForegroundColor Yellow
    $ErrorActionPreference = "Stop"

    Function Get-AccessMaskFromPermission($permissions) {
        $WBEM_ENABLE = 1
        $WBEM_METHOD_EXECUTE = 2
        $WBEM_FULL_WRITE_REP = 4
        $WBEM_PARTIAL_WRITE_REP = 8
        $WBEM_WRITE_PROVIDER = 0x10
        $WBEM_REMOTE_ACCESS = 0x20
        $READ_CONTROL = 0x20000
        $WRITE_DAC = 0x40000

        $WBEM_RIGHTS_FLAGS = $WBEM_ENABLE,$WBEM_METHOD_EXECUTE,$WBEM_FULL_WRITE_REP,
            $WBEM_PARTIAL_WRITE_REP,$WBEM_WRITE_PROVIDER,$WBEM_REMOTE_ACCESS,
            $READ_CONTROL,$WRITE_DAC

        $WBEM_RIGHTS_STRINGS = "Enable","MethodExecute","FullWrite","PartialWrite",
            "ProviderWrite","RemoteAccess","ReadSecurity","WriteSecurity"

        $permissionTable = @{}
        for ($i = 0; $i -lt $WBEM_RIGHTS_FLAGS.Length; $i++) {
            $permissionTable.Add($WBEM_RIGHTS_STRINGS[$i].ToLower(), $WBEM_RIGHTS_FLAGS[$i])
        }

        $accessMask = 0
        foreach ($permission in $permissions) {
            if (-not $permissionTable.ContainsKey($permission.ToLower())) {
                throw "Permission inconnue: $permission`nPermissions authorisées: $($permissionTable.Keys)"
            }
            $accessMask += $permissionTable[$permission.ToLower()]
        }
        $accessMask
    }

    if ($PSBoundParameters.ContainsKey("Credential")) {
        $remoteparams = @{ComputerName=$computerName;Credential=$credential}
    } else {
        $remoteparams = @{}
    }

    $invokeparams = @{Namespace=$namespace;Path="__systemsecurity=@"} + $remoteParams
    $output = Invoke-WmiMethod @invokeparams -Name GetSecurityDescriptor
    if ($output.ReturnValue -ne 0) {
        throw "GetSecurityDescriptor a échoué: $($output.ReturnValue)"
    }

    $acl = $output.Descriptor
    $OBJECT_INHERIT_ACE_FLAG = 0x1
    $CONTAINER_INHERIT_ACE_FLAG = 0x2
    $computerName = (Get-WmiObject @remoteparams Win32_ComputerSystem).Name

    if ($account.Contains('\')) {
        $domainaccount = $account.Split('\')
        $domain = $domainaccount[0]
        if (($domain -eq ".") -or ($domain -eq "BUILTIN")) {
            $domain = $computerName
        }
        $accountname = $domainaccount[1]
```

```
    } elseif ($account.Contains('@')) {
        $domainaccount = $account.Split('@')
        $domain = $domainaccount[1].Split('.')[0]
        $accountname = $domainaccount[0]
    } else {
        $domain = $computerName
        $accountname = $account
    }

    $getparams = @{Class="Win32_Account";Filter="Domain='$domain' and Name='$accountname'"} + $remoteParams
    $win32account = Get-WmiObject @getparams
    if ($null -eq $win32account) {
        throw "Compte utilisateur inconnu: $account"
    }

    switch ($operation) {
        "add" {
            if ($null -eq $permissions) {
                throw "Les permissions doivent être spécifié."
            }

            $accessMask = Get-AccessMaskFromPermission($permissions)
            $ace = (New-Object System.Management.ManagementClass("win32_Ace")).CreateInstance()
            $ace.AccessMask = $accessMask
            $ace.AceFlags = if ($allowInherit) { $OBJECT_INHERIT_ACE_FLAG + $CONTAINER_INHERIT_ACE_FLAG }
else { 0 }

            $trustee = (New-Object System.Management.ManagementClass("win32_Trustee")).CreateInstance()
            $trustee.SidString = $win32account.Sid
            $ace.Trustee = $trustee
            $ace.AceType = if ($deny) { 0x1 } else { 0x0 }

            $acl.DACL += $ace.psobject.immediateBaseObject
        }
        default {
            throw "Opération inconnue: $operation`nOpérations authorisés : add"
        }
    }

    $setparams = @{Name="SetSecurityDescriptor";ArgumentList=$acl.psobject.immediateBaseObject} +
$invokeParams
    $output = Invoke-WmiMethod @setparams
    if ($output.ReturnValue -ne 0) {
        throw "SetSecurityDescriptor a échoué: $($output.ReturnValue)"
    }
}

function Configure-W32TimeSDDL {
    param([string]$UserName)
    Write-Host "[8/8] Attribution des droits sur W32Time" -ForegroundColor Yellow

    $sid = (New-Object System.Security.Principal.NTAccount($UserName)).Translate([System.Security.Principal.
SecurityIdentifier]).Value
    $currentSddl = & sc.exe sdshow w32time
    $newAce = "(A;;CCLCLO;;;${sid})"
    $newSddl = $currentSddl + $newAce
    sc.exe sdset w32time "$newSddl"
    Restart-Service w32time

    Write-Host "Droits W32Time appliqués pour $UserName." -ForegroundColor Green
    Write-Host ""
}

function Run-All-Steps {
    $LangCode = "en-US"

    try {
        Execute-Step "Langue en-US" { Install-LanguageIfNeeded -LangCode $LangCode }
        Execute-Step "WinRM quickconfig" { Configure-WinRM }
        Execute-Step "Création utilisateur + Ajout aux groupes" { Create-User -UserName $UserName -Password
$Password }
```

```
        Execute-Step "Langue UI du profil" { Configure-UserLanguage -UserName $UserName -LangCode $LangCode }
        Execute-Step "WinRM Auth" { Configure-WinRMAuth $ParsedBasic $ParsedNegotiate
$ParsedAllowUnencrypted }
        Execute-Step "WinRM RootSDDL" { Configure-WinRMSDDL -UserName $UserName }
        Execute-Step "WMI/CIM root\cimv2 (Enable+RemoteAccess)" { Configure-CIMSDDL "root/cimv2" add
$UserName Enable,RemoteAccess }
        Execute-Step "WMI/CIM root\standardcimv2 (Enable+RemoteAccess)" { Configure-CIMSDDL "root
/standardcimv2" add $UserName Enable,RemoteAccess }
        Execute-Step "W32Time SDDL" { Configure-W32TimeSDDL -UserName $UserName }
    } catch {
        Write-Host "$($_.Exception.Message)" -ForegroundColor Red
    }
}

function Configure {
    $Steps = New-Object System.Collections.ArrayList

    Write-Host "============================================================" -ForegroundColor DarkGray
    Write-Host "CONFIGURE PACK [windows-by-WinRM] for supervised host" -ForegroundColor Cyan
    Write-Host "============================================================" -ForegroundColor DarkGray
    Write-Host ""

    Run-All-Steps
    Write-Host ""
    Write-Host ("" * 70) -ForegroundColor DarkGray
    Write-Host "RÉSUMÉ DES ÉTAPES" -ForegroundColor Yellow
    Write-Host ("" * 70) -ForegroundColor DarkGray

    foreach ($s in $Steps) {
        if ($s.Status -eq "Success") {
            Write-Host ("[OK] {0} : {1}" -f $s.Step, $s.Message) -ForegroundColor Green
        } else {
            Write-Host ("[KO] {0} : {1}" -f $s.Step, $s.Message) -ForegroundColor Red
        }
    }

    Write-Host ("" * 70) -ForegroundColor DarkGray
}

Configure

}
```