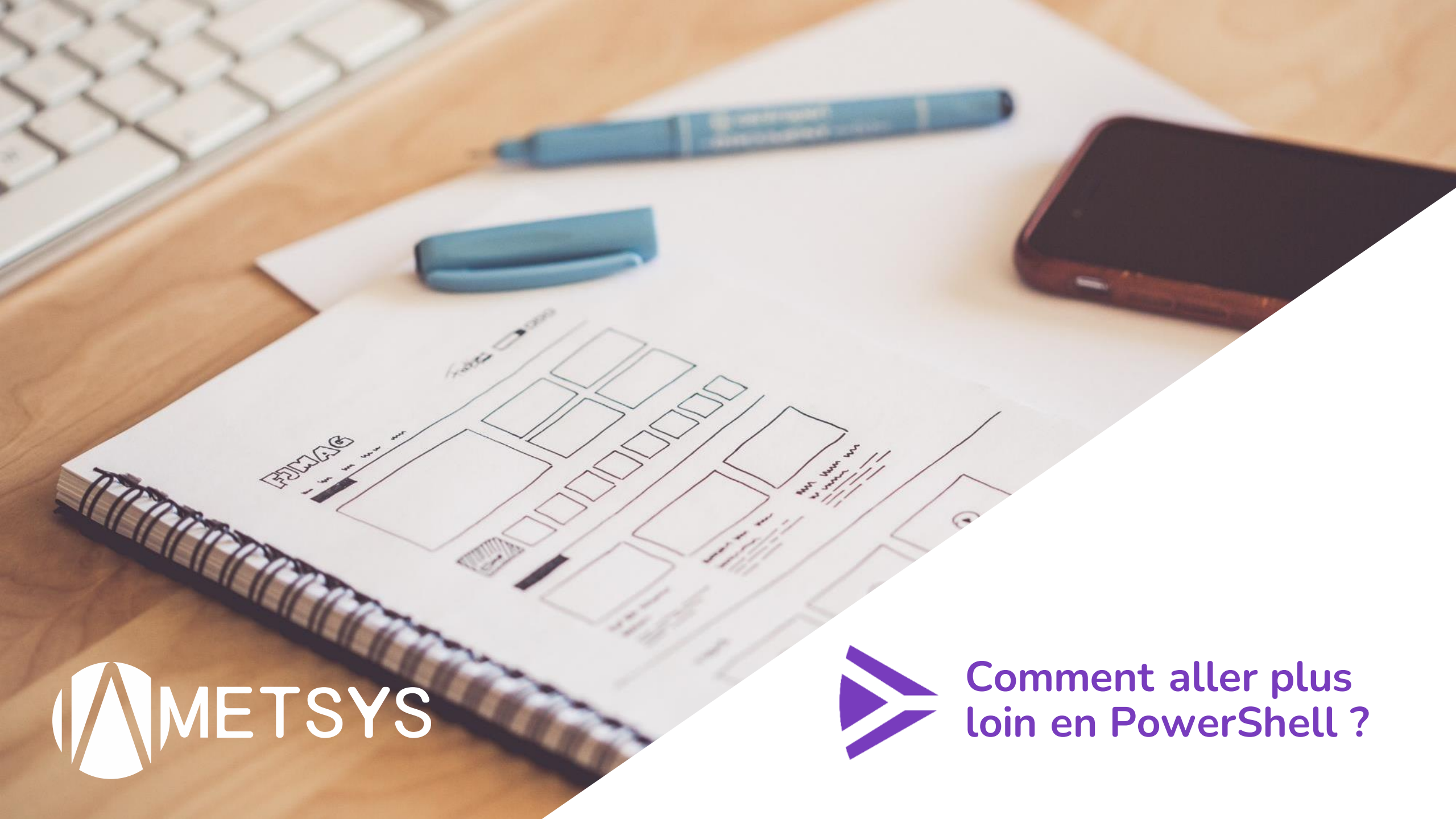
> SUP DE VINCI

Ingénierie des Systèmes d'Information

- Création en **1996**
- 4 Campus, structure juridique indépendante :
  - > Paris
  - > Rennes
  - > Bordeaux
  - > Nantes (2023)
- Plus de 25 ans d'expérience dans les **formations en alternance**
- Une **école à taille humaine** et un **suivi personnalisé**
- **2 titres certifiés** par l'État

SUP DE VINCI
INGÉNIERIE DES SYSTÈMES D'INFORMATION



METSYS

Comment aller plus loin en PowerShell ?

Présentation

Pierre LE QUERE
Lead Consultant
pierre.lequere@metsys.fr

Léo BOUARD
Lead Consultant
leo.bouard@metsys.fr

METSYS
Partageons notre savoir-faire

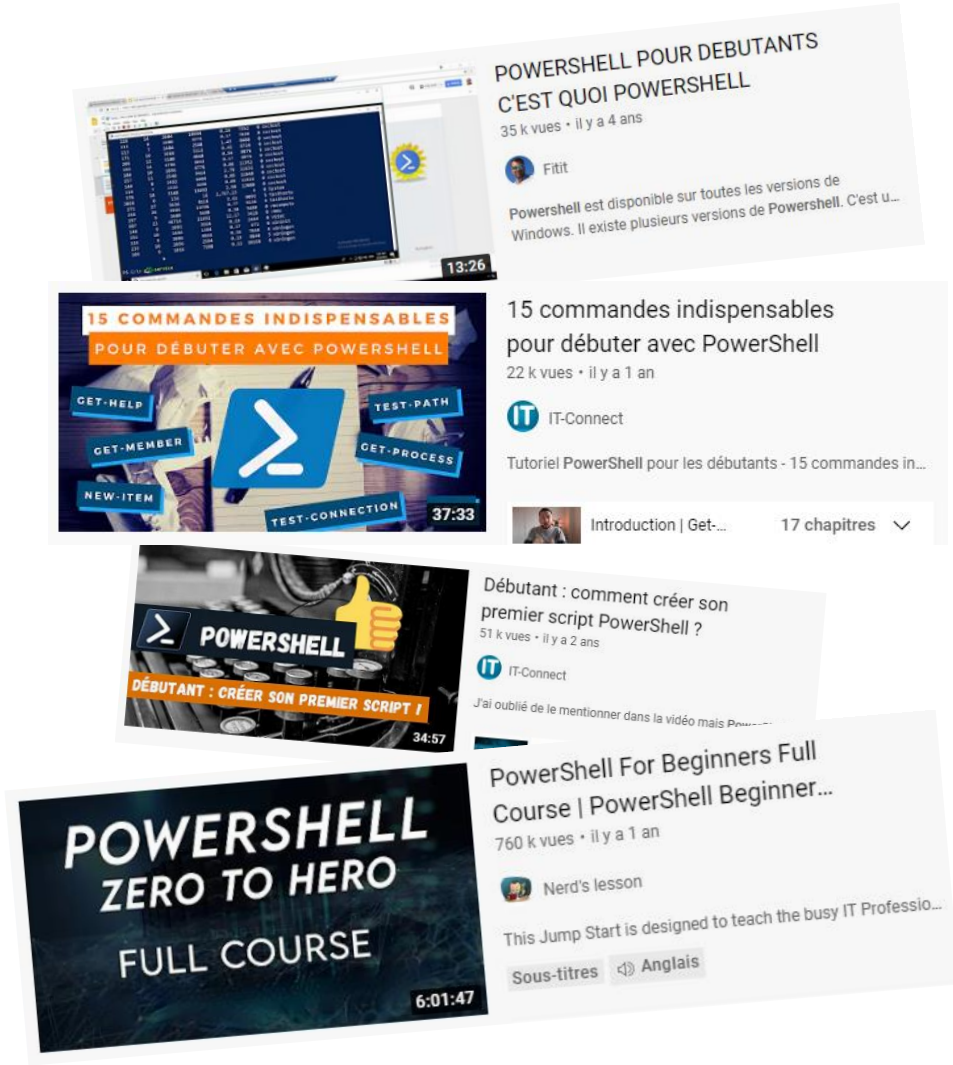
Rappel

PowerShell (anciennement Windows PowerShell) est un langage de développement créé par Microsoft, natif à Windows depuis Windows 7.

- Utilise les **Cmdlets**
- Langage de scripting orienté **objet**
- Langage d'administration système
- Variable avec le « $ » [String]$var
- Utilise les boucles standards (While, For, ForEach, etc..)
- **Fonctionnement en modules**

```
# ---

# Create "CONTOSO Groups" administrative unit
$body = @{
    displayName                 = "$entity Groups"
    description                 = "Cloud-only & on-premises groups from $e
    membershipType              = "Assigned"
}
Write-Host "Creating new Azure administrative unit: $($params.DisplayName)"
$groupAU = New-MgDirectoryAdministrativeUnit -BodyParameter $body

# Add permissions for "CONTOSO Administrators" group on "CONTOSO Groups" AU
"3aa71f9c-05c2-4121-a665-d145ebc0e43f" | ForEach-Object {
    $body = @{
        RoleId = $_
        RoleMemberIn
            Id = $ac
        }
    }
    Write-Host "Ad
    $null = New-M                                    -Administrati

}
```

Comment s'améliorer en PowerShell ?



https://www.phonandroid.com › powershell-4-scripts-a...

Powershell : 4 scripts amusants pour apprendre à s'en servir

19 juin 2018 — **Powershell** est un Invite de commandes, mais aussi un langage de programmation qui permet d'automatiser des tâches, de trier des quantités ...
Termes manquants : ~~s'améliorer~~ | Doit inclure : s'améliorer

https://www.it-connect.fr › powershell-pour-les-debutan...

Powershell pour les débutants (1ère partie) - IT-Connect

17 déc. 2014 — Première partie d'un cours pour débuter avec **PowerShell** et apprendre les bases de ce langage (cmdlet, pipeline, aide, chaine, etc.).

Get-Command : Informations de base sur les ... Get-Help : Aide de base (utiliser -full ou -ex...
Get-PSDrive : Informations sur les "lecteurs" ... Get-Module : Liste les "modules" actuellem...

Présentation du sujet · III. La structure des commandes · Les variables Powershell

https://blog.netwrix.fr › Opération IT

Tutoriel de Windows PowerShell Scripting pour débutants

13 janv. 2022 — Windows **PowerShell** est un moteur d'automatisation orienté objet ainsi qu'un langage de script. Doté d'un interpréteur de commandes ...

https://learn.microsoft.com › ... › Création de scripts

Prise en main de PowerShell - Microsoft Learn

il y a 7 jours — Fermez **PowerShell**. Relancez la console **PowerShell**, mais cette fois-ci, cliquez avec le bouton droit sur le raccourci Windows **PowerShell**, puis ...

https://learn.microsoft.com › ... › Création de scripts

Système d'aide - PowerShell - Microsoft Learn

il y a 7 jours — The Update-Help cmdlet downloads the most current Help files for Windows **PowerShell** modules, and installs them on your computer. For more ...

https://www.maformation.fr › actualites › comment-app...

Comment apprendre PowerShell - MaFormation

25 févr. 2022 — **PowerShell** permet à ses utilisateurs d'automatiser de nombreuses tâches et processus simples. Cette solution multiplateforme développée par ...

Sommaire

Les fonctionnalités avancées

Un pas de plus vers l'objet

REST et PowerShell

Un peu de sécurité

L'idée est d'avoir déjà des connaissances de base pour ce meet-up

Les mises à jour majeures de PowerShell

Windows PowerShell 1.0

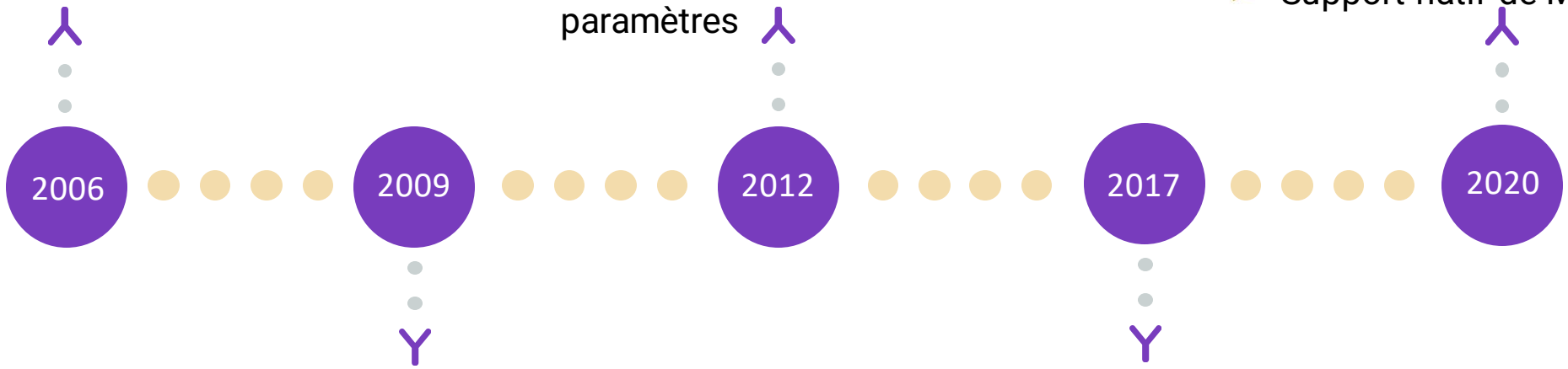
- Fonctionnalités initiales

Windows PowerShell 3.0

- Mise en tâche planifiée
- Auto-complétions (IntelliSense)
- Nouvelles commandes et paramètres

PowerShell 7

- Parallélisation avec ForEach-Object
- Opérateur ternaire
- Support natif de MarkDown



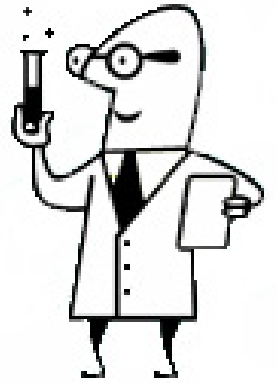
Windows PowerShell 2.0

- PowerShell remoting
- PSJob
- Création de modules

Windows PowerShell V5.1

- Transcription améliorée
- PowerShell Gallery & private repositories

Comment se tenir à jour ?



Sources officielles :

- https://devblogs.microsoft.com/scripting/
- https://devblogs.microsoft.com/powershell/
- https://learn.microsoft.com/en-us/powershell/scripting/whats-new/overview

En lisant du code PowerShell

- https://www.reddit.com/r/powershell
- https://github.com/search?&q=powershell

En suivant des blogs spécialisés :

- https://www.systemcenterdudes.com/ pour SCCM & Intune
- https://www.sharepointdiary.com/ pour SharePoint
- https://powershell-guru.com/ pour l'optimisation de performance
- https://adamtheautomator.com/tag/powershell/

METSYS

Les fonctionnalités avancées

Créer des fonctions avancées

```
Function <Nom_de_la_fonction> {
    [CmdletBinding()]
    Param (<liste_des_paramètres>)
    DynamicParam {...}
    Begin {...}
    Process {...}
    End {...}
}
```

- **[CmdletBinding()]**
  - Comportements globaux sur notre fonction
- **Param ()**
  - Déclarer des variables au sein d'une fonction
- **[Parameter()]**
  - Modifier la caractéristique d'une variable à l'intérieur du **Param()**
- **Begin {} Process {} End {}**
  - Permet de structurer son code dans sa fonction , pour éviter de répéter du code dans un pipeline

Explications

Param ()

```
Param (
        [Parameter(Mandatory=$true)]
        $Name,
        [Parameter(Mandatory=$true,
              ParameterSetName='jeu1')]
        $Online
    )
```

Voici des exemples pour se familiariser.

Begin {} Process {} End {}

```
Function <Nom_de_la_fonction> {
    [CmdletBinding()]
    Param ()
    Begin {...}
    Process {...}
    End {...}
}
```

Explications

[CmdletBinding()]

```
Function Test-ShouldProcess {
        [CmdletBinding(SupportsShouldProcess = $true )]
        Param()
        if ($PSCmdlet.ShouldProcess('Test','Appel de ShouldProcess'))
        {
            Write-Host "Execution de l'action car $true"
        }
        else
        {
            Write-Host "Simulation de l'action car $false"
        }
}
```

Voici des exemples pour se familiariser.

[Parameter()]

```
Param (
        [Parameter(Mandatory=$true,
            Position=0)]
        $Name
    )
```

Utiliser des modules

- **Module Binaire**
  - Peu utilisé
    - Les modules binaires sont un type de modules assez spécifiques. Ils contiennent du code compilé à partir d'un langage .NET.

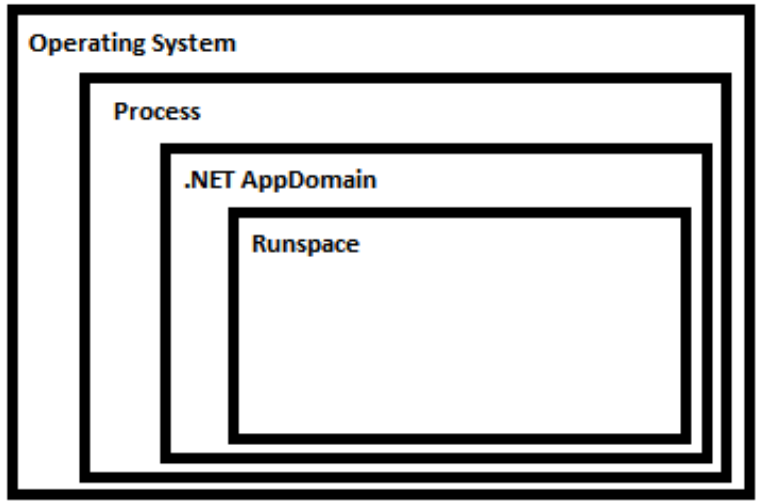
- **Module script**
  - Le plus utilisé
    - Module a créer avec un fichier PSM1.

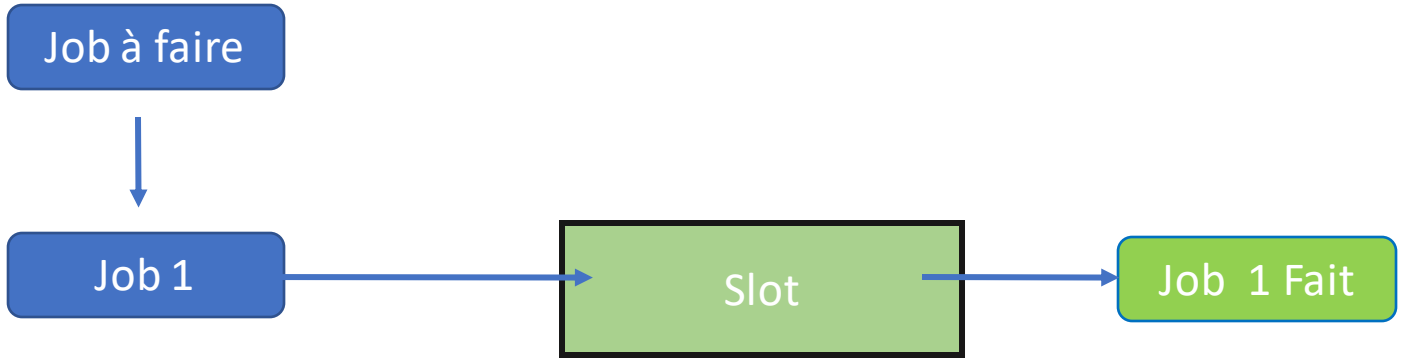
- **Module Dynamique**
  - Exemple : les modules Exchange On-Premises
  - Utilisé mais attention chargé en mémoire et existe le temps de la session PowerShell
    - Créer avec la commande New-module

Les runspaces

```powershell
$Runspace = [runspacefactory]::CreateRunspace()
$PowerShell = [powershell]::Create()
$PowerShell.Runspace = $Runspace
$Runspace.Open()
$PowerShell.AddScript({Start-Sleep 5})

1..10 | Foreach-Object {
    $Job = $PowerShell.BeginInvoke()
    while ($Job.IsCompleted -eq $false) {Start-Sleep -Milliseconds 100}
}
```

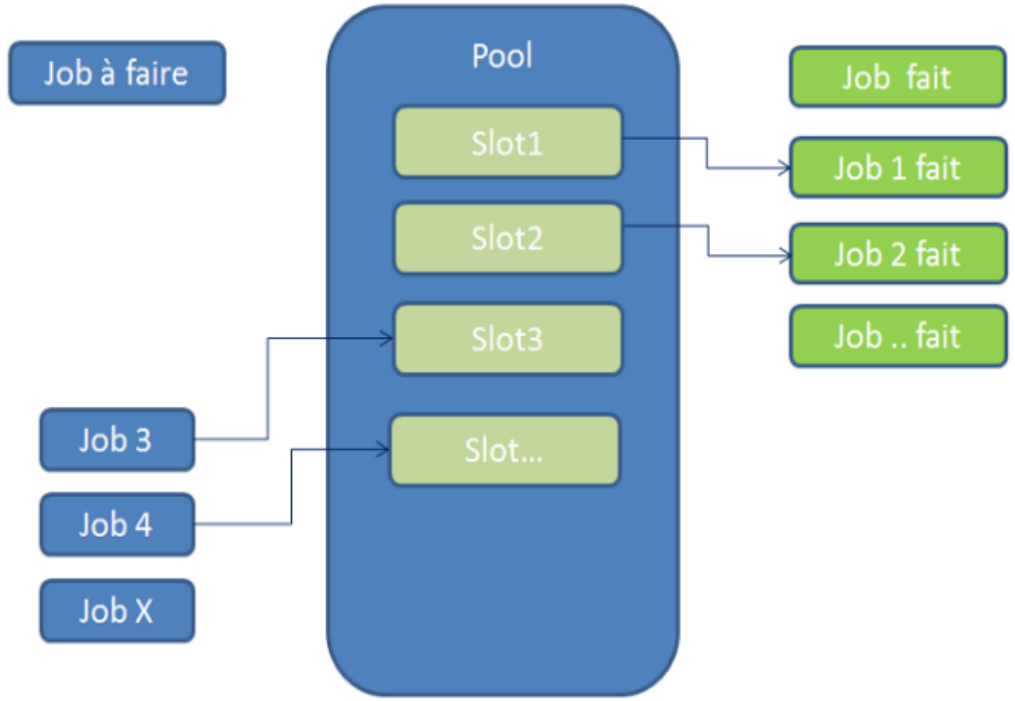
- **BeginInvoke()**
  - Objet asynchrone pour lancer le runspace
- **Job**
  - Tâche à effectuer
- **Slot**
  - Thread qui exécute le job





Plus de performance avec les pools de runspaces

```powershell
$RunspacePool = [runspacefactory]::CreateRunspacePool(1, 5)
$RunspacePool.Open()
$Jobs = @()

1..10 | Foreach-Object {
    $PowerShell = [powershell]::Create()
    $PowerShell.RunspacePool = $RunspacePool
    $PowerShell.AddScript({Start-Sleep 5})
    $Jobs += $PowerShell.BeginInvoke()
}
while ($Jobs.IsCompleted -contains $false) {Start-Sleep -Milliseconds 100}
```



Comparaison des performances

runspace1.ps1*

```powershell
Measure-command {

    $Runspace = [runspacefactory]::CreateRunspace()
    $PowerShell = [powershell]::Create()
    $PowerShell.Runspace = $Runspace
    $Runspace.Open()
    $PowerShell.AddScript({Start-Sleep 5})

    1..10 | Foreach-Object {
        $Job = $PowerShell.BeginInvoke()
        while ($Job.IsCompleted -eq $false) {Start-Sleep -Milliseconds 100}
    }

}


Measure-command {

    $RunspacePool = [runspacefactory]::CreateRunspacePool(1, 5)
    $RunspacePool.Open()
    $Jobs = @()

    1..10 | Foreach-Object {
        $PowerShell = [powershell]::Create()
        $PowerShell.RunspacePool = $RunspacePool
        $PowerShell.AddScript({Start-Sleep 5})
        $Jobs += $PowerShell.BeginInvoke()
    }
    while ($Jobs.IsCompleted -contains $false) {Start-Sleep -Milliseconds 100}

}
```

```
PS C:\windows\system32>
```

Version plus simple de la parallélisation

PowerShell v7.0 a inauguré une nouvelle méthode pour faire de la parallélisation :

ForEach-Object -Parallel

```
1..10 | ForEach-Object -Parallel { Start-Sleep 1 }
```

ThrottleLimit	Temps global	Modificateur temps
Sans parallélisation	10,19 sec	+1,9%
2	5,15 sec	-48,5%
5	2,12 sec	-78,8%
10	1,20 sec	-88,0%

Un peu de nuance

Aucune plus-value de la parallélisation sur le traitement d'opérations très rapides

```
# ForEach-Object
(Measure-Command {
    1..100000 | ForEach-Object {
        $_
    }
}).TotalMilliseconds
```

Nombre d'objets	Boucle utilisée	Temps de traitement
100 000	foreach ()	90 ms
100 000	ForEach-Object	400 ms
1 000	ForEach-Object -Parallel	900 ms
100 000	ForEach-Object -Parallel	+ 9 minutes

Pour aller plus loin : [Speeding Up the Pipeline - powershell.one]

METSYS

Un pas de plus vers l’objet

Mise à jour des connaissances - ARRAY

✓ Bonne pratique

```
$array = [System.Collections.Generic.List[int]]@()
1..10 | ForEach-Object {
    $array.Add($_)
}
```

```
$array = 1..10 | ForEach-Object { $_ }
```

✗ Mauvaise pratique

```
$array = @()
1..10 | ForEach-Object {
     $array += $_
}
```

Raisons :

- Via Generic.List :
  - Accès à de nouvelles méthodes
  - Possibilité de typer les objets membres
- Plus de performance (opérateur "+=" à éviter)
- Meilleure évolutivité

Mise à jour des connaissances - PSOBJECT

☑ Bonne pratique

```
[PSCustomObject]@{
    GivenName = "John"
    Surname = "Smith"
}
```

✕ Mauvaise pratique

```
New-Object -TypeName psobject -Property @{
    GivenName = "John"
    Surname = "Smith"
}
```

Raisons :

- Syntaxe plus simple
- Plus de performance (éviter le "New-Object" de manière générale)

<u>Attention :</u> pas compatible avec les versions antérieures à PowerShell 5

Mise à jour des connaissances - SPLATTING

Utilisation d'une HASHTABLE pour envoyer des paramètres à une commande

Permet une approche programmatique

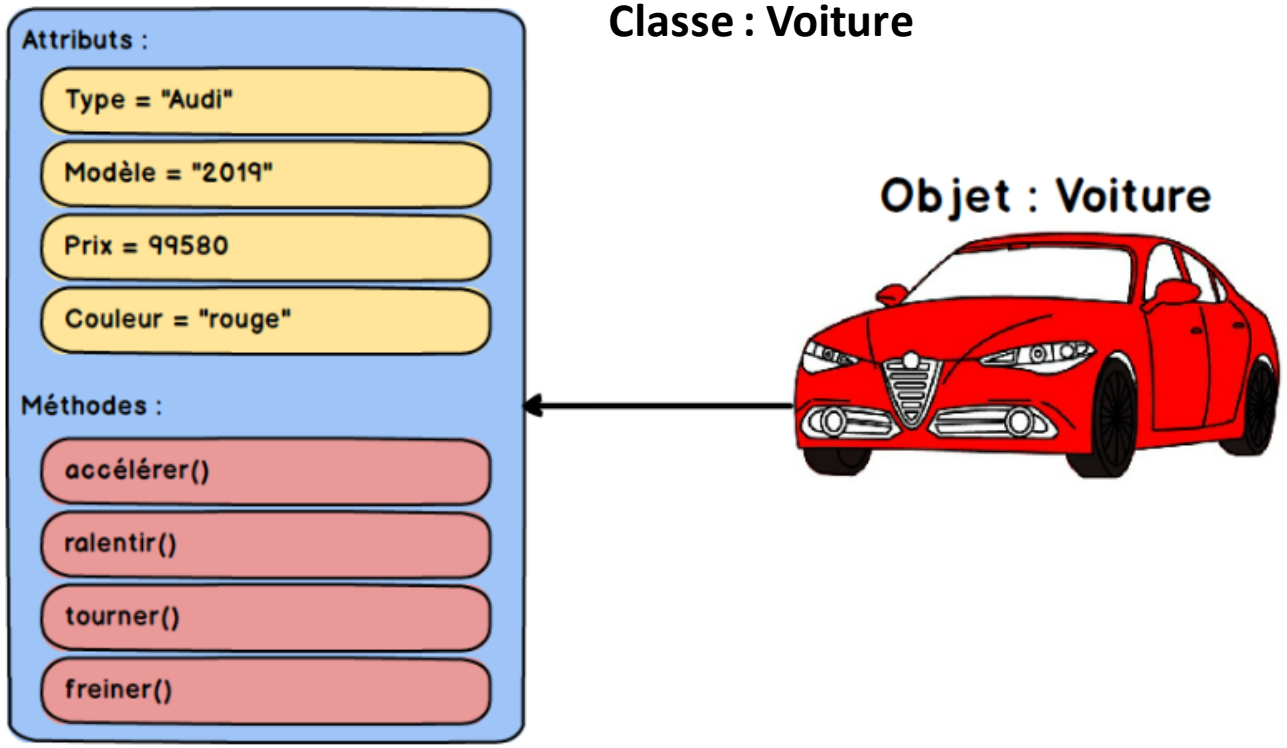
```powershell
$params = @{
    Body        = $body
    BodyAsHtml  = $true
    Encoding    = "UTF8"
    From        = "noreply@metsys.fr"
    SmtpServer  = "smtp.metsys.fr"
    Subject     = "Invitation au meet-up technique"
    To          = $to
    ErrorAction = "Stop"
}

if ($to -like "pierre.lequere*") { $params.Add("Cc","leo.bouard@metsys.fr") }

Send-MailMessage @params
```

```powershell
if ($to -like "pierre.lequere*") {
    Send-MailMessage -Body $body `
        -BodyAsHtml `
        -Encoding "UTF8" `
        -From "noreply@metsys.fr" `
        -SmtpServer "smtp.metsys.fr" `
        -Subject "Invitation au meet-up technique" `
        -To $to `
        -Cc "leo.bouard@metsys.fr" `
        -ErrorAction "Stop"
} else {
    Send-MailMessage -Body $body `
        -BodyAsHtml `
        -Encoding "UTF8" `
        -From "noreply@metsys.fr" `
        -SmtpServer "smtp.metsys.fr" `
        -Subject "Invitation au meet-up technique" `
        -To $to `
        -ErrorAction "Stop"
}
```

Classes et Objets

- **La classe**
  - Contient des variables et des fonctions permettant de créer des objets
- **L'objet**
  - C'est l'instance qui provient de la classe
- **La méthode**
  - Il s'agit des opérations applicables aux objets
- **La propriété**
  - Il s'agit des données représentant l'état de l'objet

Exemple :



Exemple d'utilisation en PowerShell

- New-Object (Ne plus utilisé)
- [PSCustomObject]@{}
- Class {}
- [Classe]::new()

1

```powershell
#Methode Script Toyota

$toyota = [PSCustomObject]@{
    Numberofwheels  = '4'
    NumberOfDoors = '5'
    Year = '10/10/2011'
    Marque = 'Toyota'
}
```

2

```powershell
#Methode Développeur
Class Toyota

{
    [int]$numberOfWheels

    [int]$numberOfDoors

    [datetime]$year

    [String]$marque

  Toyota() #Constructeur pour une toyota

  {
        [int]$this.numberOfWheels = "4"

        [int]$this.numberOfDoors = "5"

        [datetime]$this.year = "10/10/2011"

        [String]$this.marque = "Toyota"

    }
}

$Toyota = [Toyota]::new() # On instancie

$Toyota
```

```
Numberofwheels NumberOfDoors Year       Marque
-------------- ------------- ----       ------
4              5             10/10/2011 Toyota
```

Dans quel contexte utiliser des classes ?

- PowerShell permet d'écrire aussi à la manière des développeurs
- Adaptabilité des développeurs dans le langage de script
- Lisibilité des différents corps de métiers
- Moins besoin de monter en compétence

Polymorphisme

- Faire hériter des classes entre elles est très important pour éviter de répéter du code inutilement

```
Class Toyota
{
    [int]$numberOfWheels
    [int]$numberOfDoors
    [datetime]$year
    [String]$marque

  Toyota() #Constructeur pour une toyota
  {
        [int]$this.numberOfWheels = "4"
        [int]$this.numberOfDoors = "5"
        [datetime]$this.year = "10/10/2011"
        [String]$this.marque = "Toyota"

    }
}

$Toyota = [Toyota]::new() # On instancie
$Toyota
```

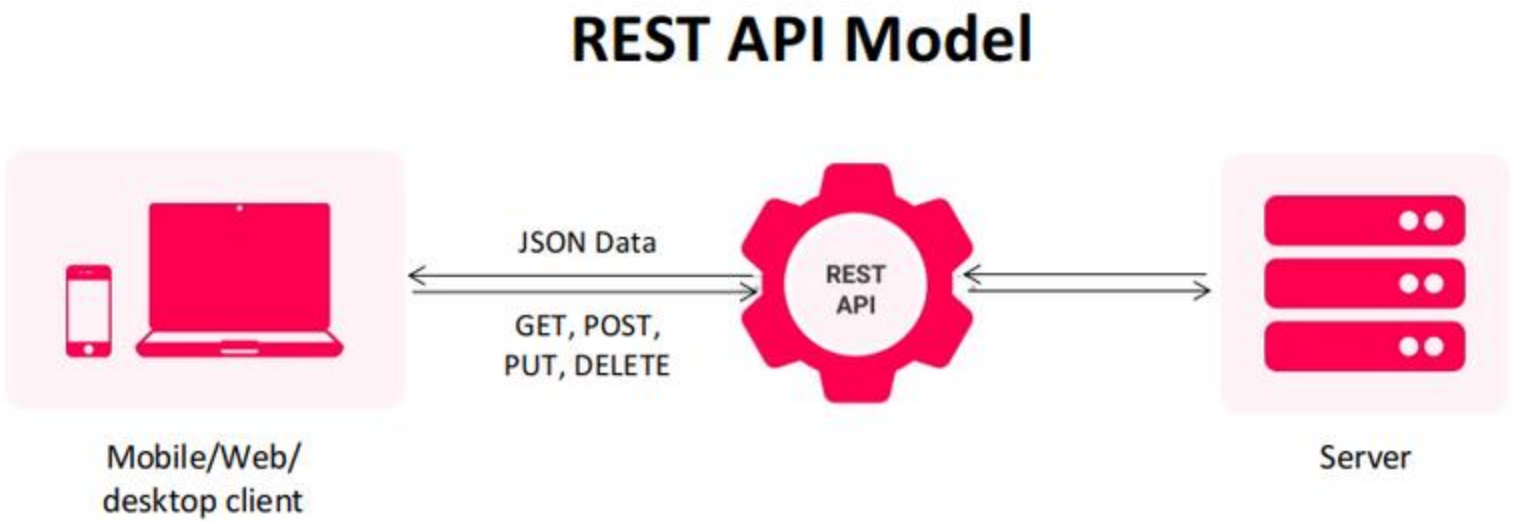
```
Class Lexus:Toyota
{
    [string]$moteur
    [int]$nbrAileron

    Lexus() #Constructeur
    {
        [String]$this.moteur = 'V12'
        [int]$this.nbrAileron = '1'

    }
}

$Lexus = [Lexus]::new() # On instancie
```

METSYS

REST et PowerShell

Le REST c'est quoi ?

- REST signifie ***Re****presentational* ***S****tate* ***T****ransfer*
- Ensemble de **normes**
- Lignes directrices **architecturales** qui structurent la façon de communiquer les données entre votre application et le reste du monde, ou entre différents composants de votre application.



Methodes HTTP pour des services REST

GET

- Récupérer une ressource

POST

- Créer une nouvelle ressource

PUT / PATCH

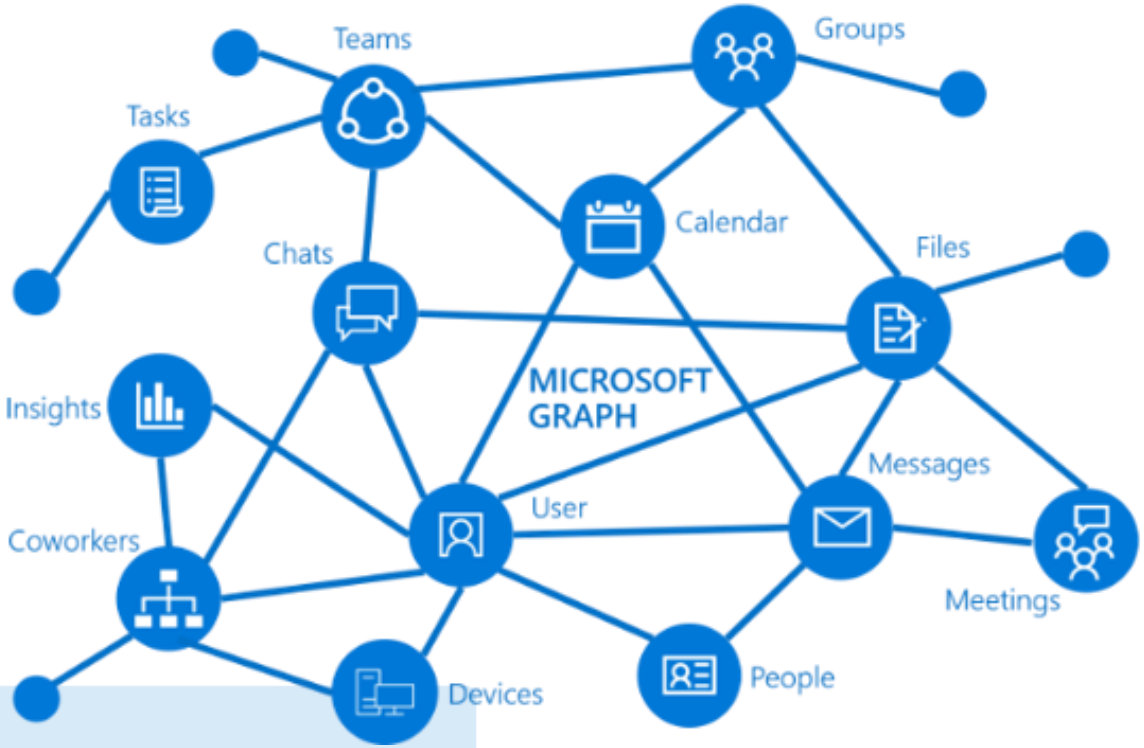
- Modifier ou ajouter une ressource

DELETE

- Supprimer une ressource

Microsoft Graph

- API officielle de Microsoft pour ses produits cloud
- Basé sur REST
- Envoi de JSON
- Modules PowerShell disponibles
- Permet de requêter :
  - Microsoft 365
  - SharePoint
  - Exchange Online
  - Microsoft Teams
  - Azure…

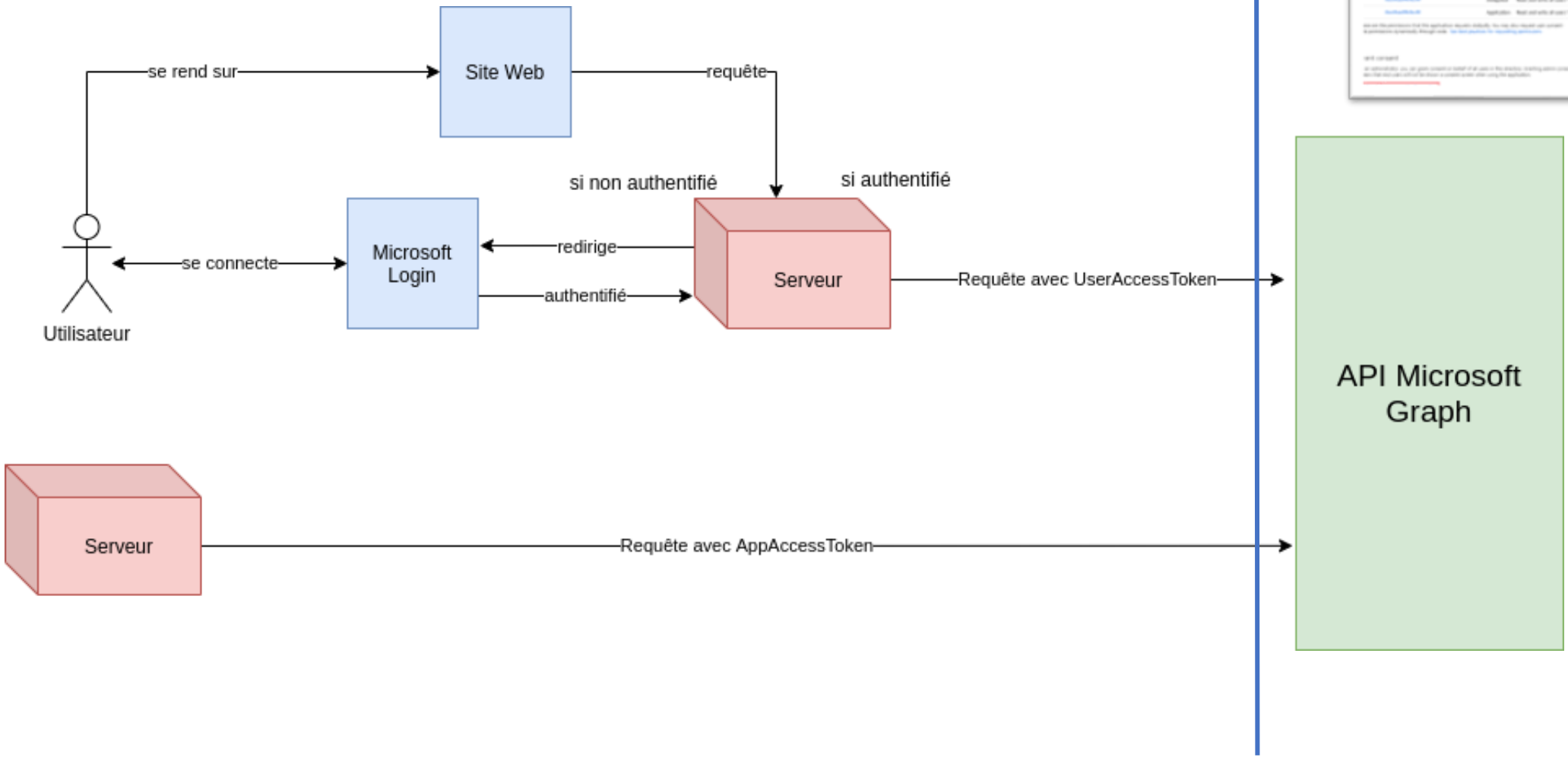


Important

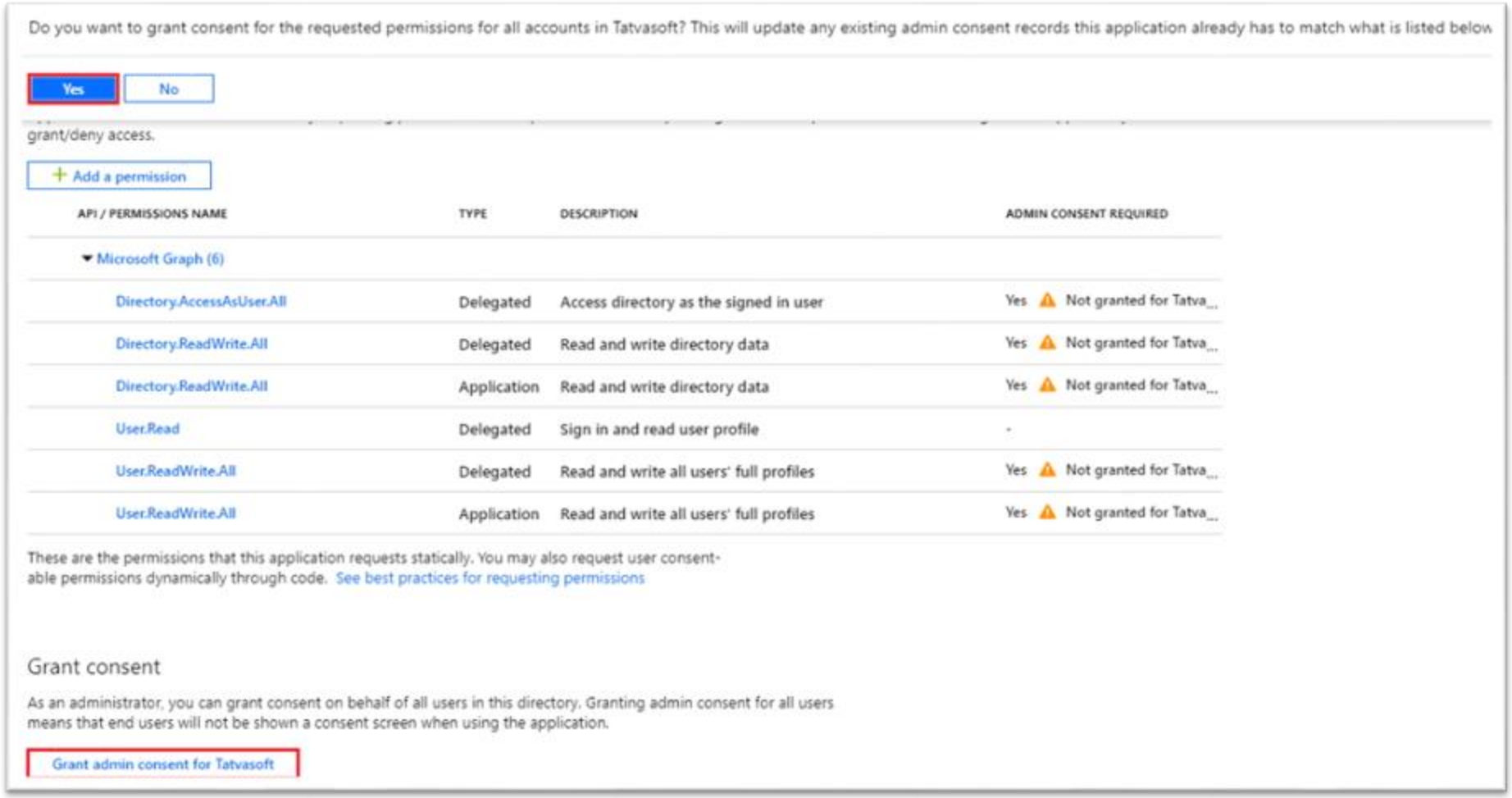
MSOnline is planned for deprecation. For more details on the deprecation plans, see the **deprecation update**. You can start trying **Microsoft Graph PowerShell** to interact with Azure AD as you would in MSOnline. In addition, Microsoft Graph PowerShell allows you access to all Microsoft Graph APIs and is available on PowerShell 7. For answers to frequent migration queries, see the **migration FAQ**.

Comprendre les permissions Microsoft Graph

Droit Graph



Comprendre les permissions Microsoft Graph



METSYS

Un peu de sécurité

Les bases pour une bonne cybersécurité

Disponibilité

- Cet objectif couvre la notion d'accès aux ressources informatiques sur les périodes prévues

Intégrité

- Garantir que les données servies n'ont pas été modifiées anormalement (intentionnellement ou non) .

Confidentialité

- Le pilier le plus évident, cherche à garantir que l'accès aux données n'est possible que si (et seulement si) les personnes sont bien autorisées et donc authentifiées.

Traçabilité

- Le dernier pilier cherche à garantir que les actions sur le SI sont bien journalisées et stockées de manière pérenne.

Défendre et attaquer en PowerShell

Signature des scripts

- Nécessite que tous les fichiers de scripts soient signés numériquement avant de pouvoir être exécutés.

Attaque sur KeePass

- KeePass est un gestionnaire de mot de passe. Ce type de logiciel permet de protéger les mots de passe de ses utilisateurs en les stockant dans un fichier de coffre-fort chiffré.

Chiffrement

- Le chiffrement est un procédé de cryptographie qui consiste à protéger des données qui sont alors incompréhensibles pour celui qui ne dispose pas de la clef du chiffrement.

Gestion des secrets dans PowerShell

Comment gérer les secrets dans les scripts ?

Plusieurs méthodes pour gérer les secrets & identifiants:

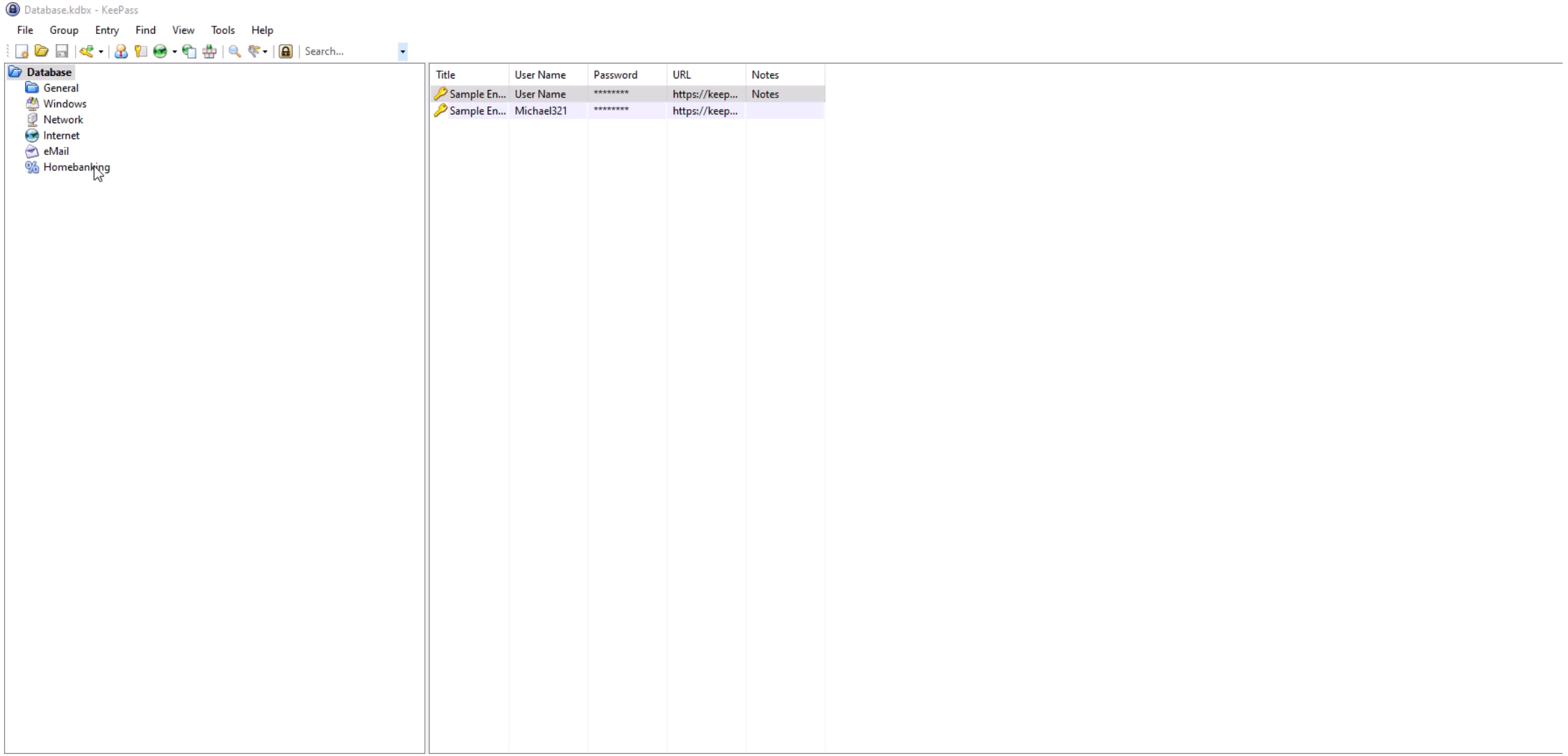
- Connexion par certificat quand c'est possible
- Avec des gestionnaires de mots de passe
  - KeePass et LockPass
  - Azure Key Vault
  - Module Microsoft.PowerShell.SecretManagement
- Via `ConvertFrom-SecureString` & `ConvertTo-SecureString`
- Via `Import-CliXml` & `Export-CliXml`

```
Get-Credential | Export-Clixml -Path ".\cred.xml"
$cred = Import-Clixml -Path ".\cred.xml"

Connect-PnPOnline -Credentials $cred
```

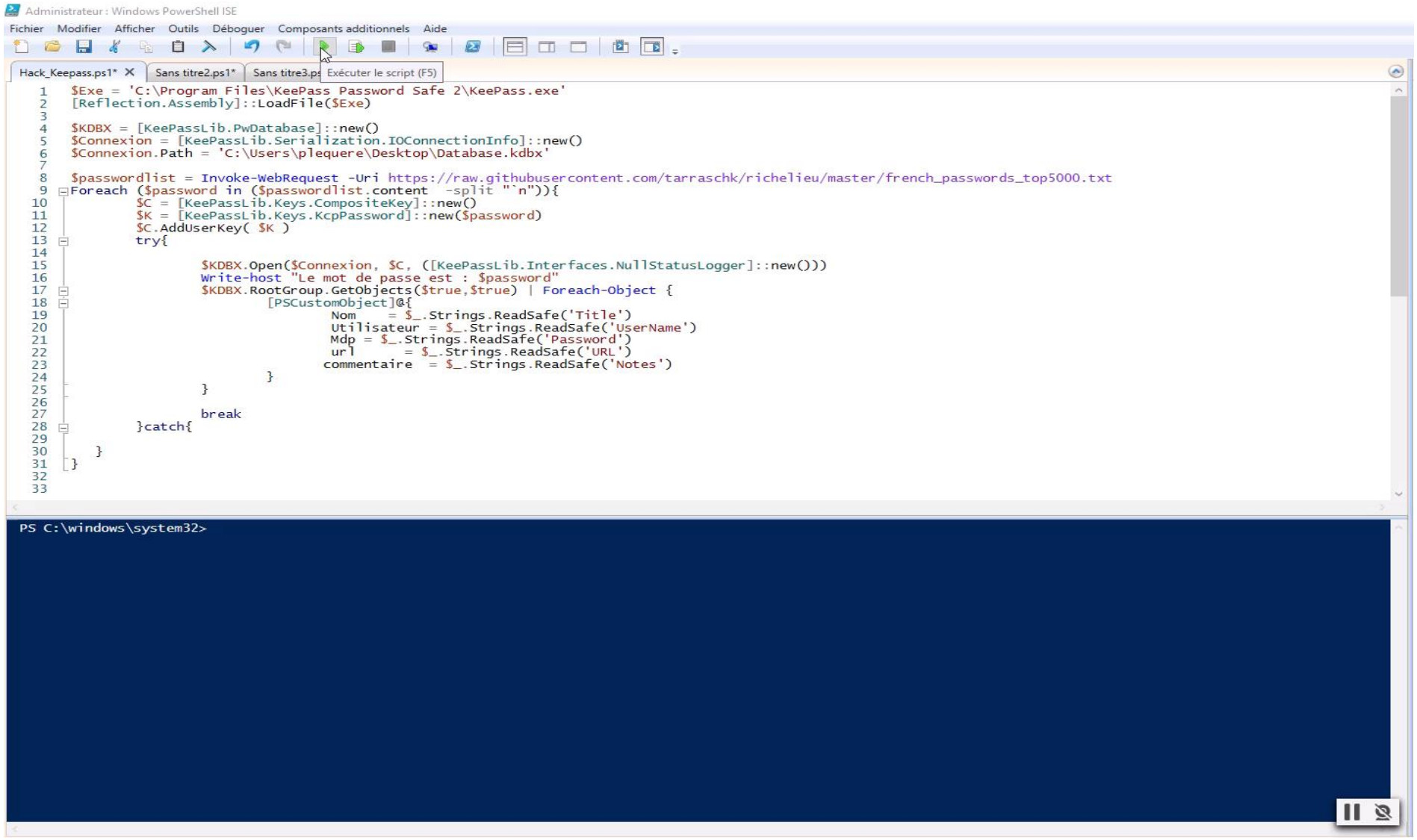
Gestion des secrets dans PowerShell

Démonstration

Création d'un KeePass



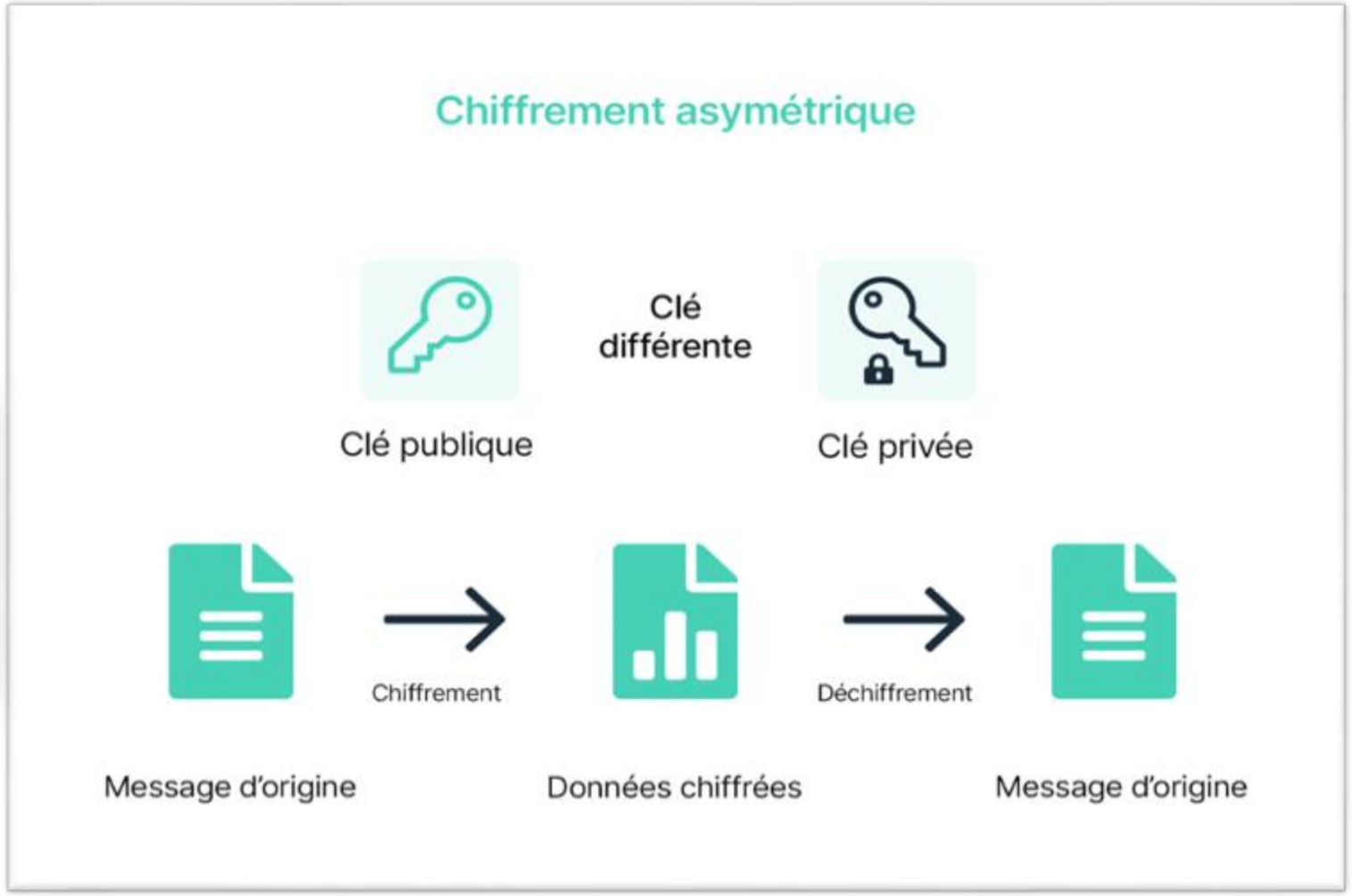
Brute Force sur KeePass



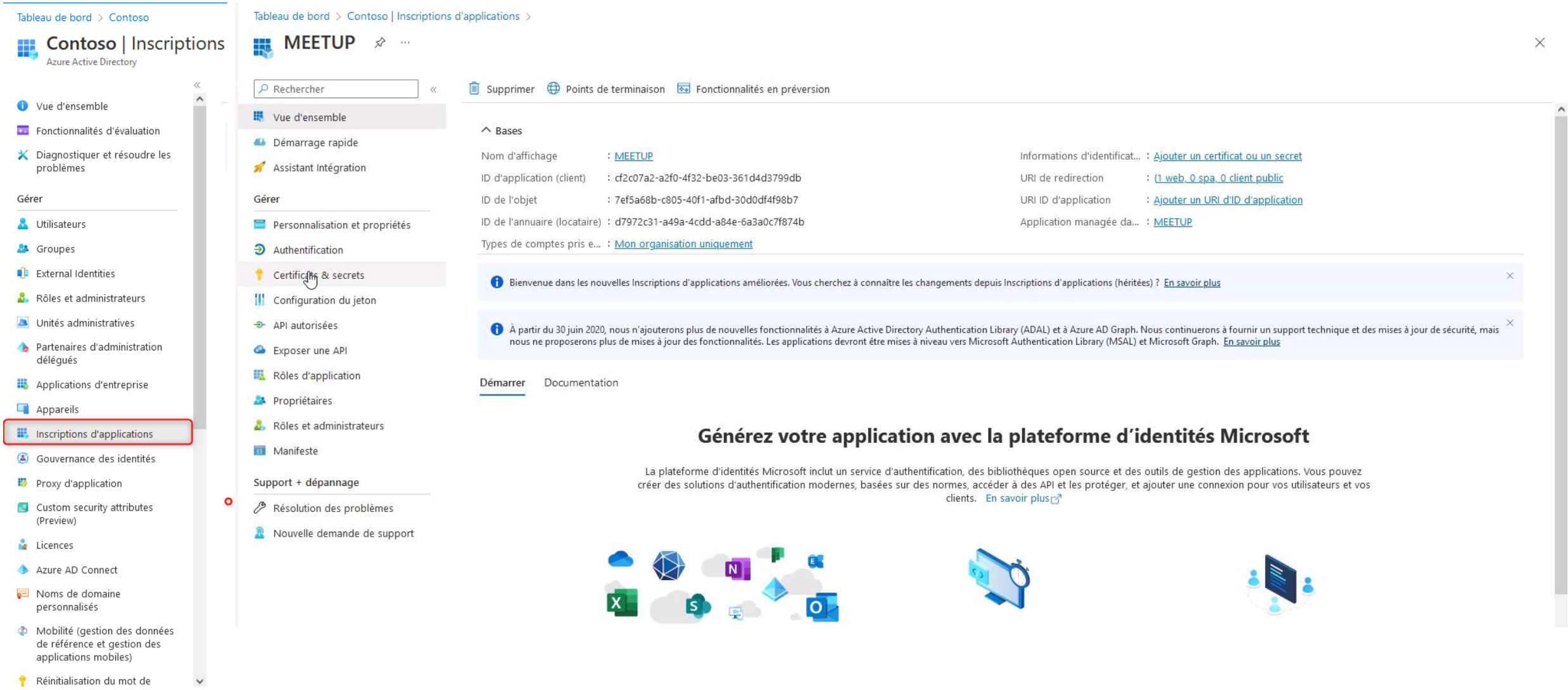
Signature des Scripts

- ADCS (PKI)
- Certificat Auto Signé
- GPO
- Set-ExecutionPolicy **All Signed**

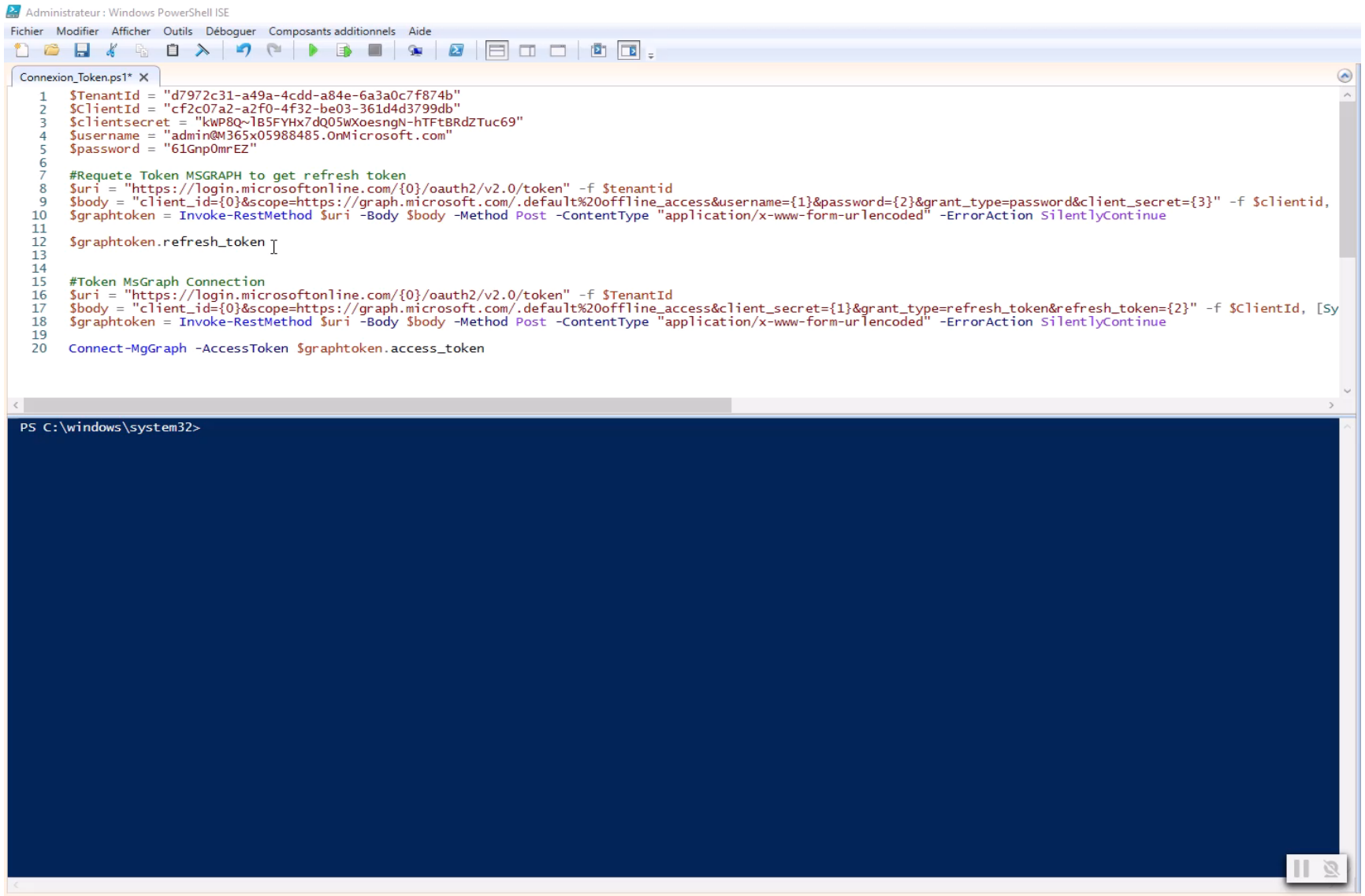
```
# SIG # Begin signature block
# MIIPFwYJKoZIhvcNAQcCoIIPCDCCDwQCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQUTFtiZi5ymuVxcjW4obHiRPBO
# s5Wgggo3MIIEmTCCA4GgAwIBAgIPFojwOSVeY45pFDkH5jMLMA0GCSqGSIb3DQEB
# BQUAMIGVMQswCQYDVQQGEwJVUzELMAkGA1UECBMCVVQxFzAVBgNVBAcTDlNhbHQg
# TGFrZSBDaXR5MR4wHAYDVQQKExVUaGUgVVNFUlRSVVNUIE5ldHdvcmsxITAfBgNV
# BAsTGGh0dHA6Ly93d3cudXNlcnRydXN0LmNvbTEdMBsGA1UEAxMUVVROLVVTRVJG
# aXJzdC1PYmplY3QwHhcNMTUxMjMxMDAwMDAwWhcNMTkwNzA5MTg0MDM2WjCBhDEL
# MAkGA1UEBhMCR0IxGzAZBgNVBAgTEkdyZWF0ZXIgTWFuY2hlc3RlcjEQMA4GA1UE
# BxMHU2FsZm9yZDEaMBgGA1UEChMRQ09NT0RPIENBIExpbWl0ZWQxKjAoBgNVBAMT
# IUNPTU9ETyBTSEEtMSBUaW1lIFN0YW1waW5nIFNpZ25lcjCCASIwDQYJKoZIhvcN
```

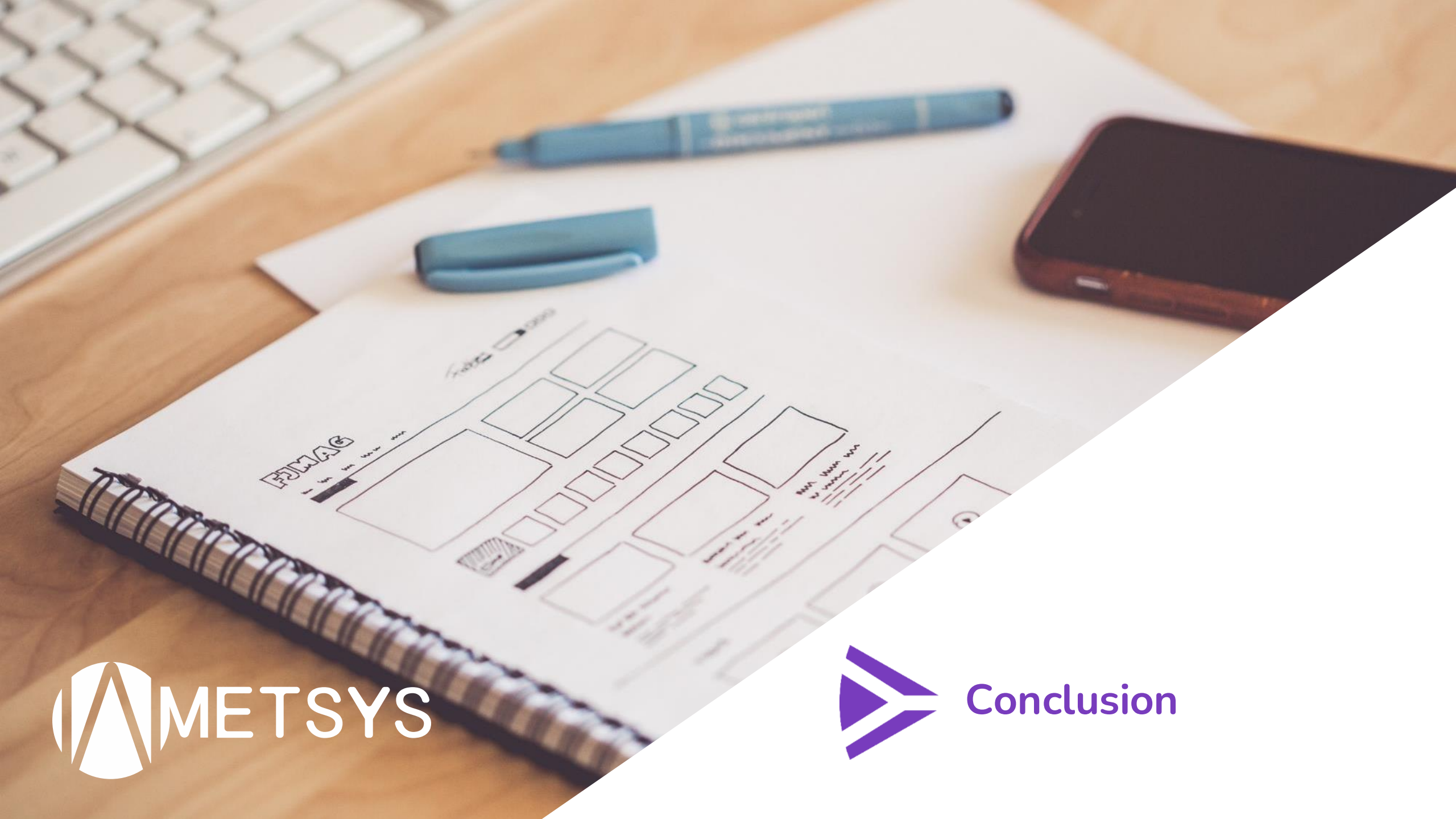


Sécuriser la connexion avec un Token



Sécuriser la connexion avec un Token





METSYS

Conclusion

METSYS

Des questions ?