Script Migration SQL standard vers Web Partie 1 :

```
#Requires -RunAsAdministrator

<#
.SYNOPSIS
    Migration SQL Server 2017 Standard vers Web Edition
.DESCRIPTION
    Sauvegarde les bases, désinstalle SQL Standard, installe SQL Web, et restaure les bases
#>

# Configuration
$instanceName = "SQLSPI2017"
$productKey = ""
$backupPath = "D:\Usr\Data.Net\BaseSqlBackup"
$sqlSetupPath =
"\\sr9v-ftpspi\c$\Data\FtpSepteoPI\FtpPrive\SPIsuite\SQLServer2017Standard\SQL2017_St
andard\setup.exe"
$saPassword = ""
$tempSaPassword = "**************"  # Mot de passe temporaire complexe pour l'installation



# Création du dossier de sauvegarde
Write-Host "=== Étape 1: Création du dossier de sauvegarde ===" -ForegroundColor Cyan
if (!(Test-Path $backupPath)) {
    New-Item -ItemType Directory -Path $backupPath -Force | Out-Null
    Write-Host "Dossier créé: $backupPath" -ForegroundColor Green
} else {
    Write-Host "Dossier existant: $backupPath" -ForegroundColor Green
}

# Sauvegarde des bases de données
Write-Host "`n=== Étape 2: Sauvegarde des bases de données ===" -ForegroundColor
Cyan
try {
    # Import du module SQL Server
    Import-Module SQLPS -DisableNameChecking -ErrorAction Stop
    Write-Host "Module SQLPS importé" -ForegroundColor Green

    $serverInstance = ".\$instanceName"
    Write-Host "Connexion à: $serverInstance" -ForegroundColor Yellow

    $serverConnection = New-Object
Microsoft.SqlServer.Management.Common.ServerConnection
    $serverConnection.ServerInstance = $serverInstance
    $serverConnection.LoginSecure = $false
```

```
$serverConnection.Login = "SA"
$serverConnection.Password = $saPassword
$server = New-Object Microsoft.SqlServer.Management.Smo.Server($serverConnection)


#$server = New-Object Microsoft.SqlServer.Management.Smo.Server($serverInstance)

# Test de connexion
if ($server.Version -eq $null) {
    throw "Impossible de se connecter à l'instance SQL Server '$serverInstance'"
}
Write-Host "  -> Connexion réussie" -ForegroundColor Green
Write-Host "  -> Version SQL: $($server.Version)" -ForegroundColor Gray
Write-Host "  -> Edition: $($server.Edition)" -ForegroundColor Gray

# Liste des bases à sauvegarder (exclut les bases système)
$databasesToBackup = $server.Databases | Where-Object {
    $_.IsSystemObject -eq $false -and $_.Name -ne 'tempdb'
}

if ($databasesToBackup.Count -eq 0) {
    Write-Host "Aucune base utilisateur trouvée" -ForegroundColor Yellow
} else {
    Write-Host "`nBases trouvées: $($databasesToBackup.Count)" -ForegroundColor White
    $databasesToBackup | ForEach-Object { Write-Host "  - $($_.Name)" -ForegroundColor
Gray }
}

$successCount = 0
$errorCount = 0

foreach ($db in $databasesToBackup) {
    $dbName = $db.Name
    $backupFile = Join-Path $backupPath "$dbName.bak"

    Write-Host "`n[Base: $dbName]" -ForegroundColor Yellow
    Write-Host "  Taille: $([math]::Round($db.Size, 2)) MB" -ForegroundColor Gray
    Write-Host "  État: $($db.Status)" -ForegroundColor Gray
    Write-Host "  Fichier: $backupFile" -ForegroundColor Gray

    # Vérifier que la base est accessible
    if ($db.Status -ne 'Normal') {
        Write-Host "  -> IGNORÉE: Base en état '$($db.Status)'" -ForegroundColor Red
        $errorCount++
        continue
    }

    # Vérifier que le chemin de backup est accessible
```

```
        if (!(Test-Path (Split-Path $backupFile -Parent))) {
            Write-Host "  -> ERREUR: Chemin de sauvegarde inaccessible" -ForegroundColor
Red
            $errorCount++
            continue
        }

        try {
            $backup = New-Object Microsoft.SqlServer.Management.Smo.Backup
            $backup.Action =
[Microsoft.SqlServer.Management.Smo.BackupActionType]::Database
            $backup.Database = $dbName
            $backup.Devices.AddDevice($backupFile,
[Microsoft.SqlServer.Management.Smo.DeviceType]::File)
            $backup.Initialize = $true
            $backup.Checksum = $true
            $backup.ContinueAfterError = $false
            $backup.LogTruncation =
[Microsoft.SqlServer.Management.Smo.BackupTruncateLogType]::Truncate

            # Événement pour la progression
            $percentComplete = 0
            $backup.add_PercentComplete({
                param($sender, $e)
                if ($e.Percent -ne $percentComplete -and $e.Percent % 10 -eq 0) {
                    Write-Host "  -> Progression: $($e.Percent)%" -ForegroundColor Cyan
                    $script:percentComplete = $e.Percent
                }
            })

            Write-Host "  -> Démarrage de la sauvegarde..." -ForegroundColor Cyan
            $backup.SqlBackup($server)

            # Vérification du fichier créé
            if (Test-Path $backupFile) {
                $fileInfo = Get-Item $backupFile
                Write-Host "  -> SUCCÈS: $([math]::Round($fileInfo.Length / 1MB, 2)) MB"
-ForegroundColor Green
                $successCount++
            } else {
                Write-Host "  -> ERREUR: Fichier de sauvegarde non créé" -ForegroundColor Red
                $errorCount++
            }

        } catch {
            Write-Host "  -> ERREUR lors de la sauvegarde:" -ForegroundColor Red
            Write-Host "     $($_.Exception.Message)" -ForegroundColor Red
            if ($_.Exception.InnerException) {
```

```
            Write-Host "     Détails: $($_.Exception.InnerException.Message)"
-ForegroundColor Red
        }
        $errorCount++

        # Continuer avec les autres bases
        continue
    }
  }



  # Résumé de la sauvegarde
  Write-Host "`n=== Résumé de la sauvegarde ===" -ForegroundColor Cyan
  Write-Host "Réussies: $successCount" -ForegroundColor Green
  Write-Host "Échouées: $errorCount" -ForegroundColor $(if ($errorCount -gt 0) { "Red" }
else { "Gray" })

  if ($errorCount -gt 0) {
    $response = Read-Host "`nDes erreurs sont survenues. Continuer quand même?
(O/N)"
    if ($response -ne 'O' -and $response -ne 'o') {
      Write-Host "Migration annulée par l'utilisateur" -ForegroundColor Yellow
      exit 0
    }
  }

} catch {
  Write-Host "`nERREUR CRITIQUE lors de la sauvegarde: $_" -ForegroundColor Red
  Write-Host "Détails: $($_.Exception.Message)" -ForegroundColor Red
  if ($_.Exception.InnerException) {
    Write-Host "Cause: $($_.Exception.InnerException.Message)" -ForegroundColor Red
  }
  Write-Host "`nLa migration est interrompue." -ForegroundColor Red
  exit 1
}

# Arrêt des services SQL
Write-Host "`n=== Étape 3: Arrêt des services SQL ===" -ForegroundColor Cyan
$sqlServices = Get-Service | Where-Object { $_.DisplayName -like "*SQL*$instanceName*"
}

if ($sqlServices.Count -eq 0) {
  Write-Host "Aucun service SQL trouvé pour l'instance $instanceName" -ForegroundColor
Yellow
} else {
  foreach ($service in $sqlServices) {
    Write-Host "Arrêt du service: $($service.DisplayName)" -ForegroundColor Yellow
```

```
        try {
            Stop-Service -Name $service.Name -Force -ErrorAction Stop
            Write-Host "  -> Service arrêté" -ForegroundColor Green
        } catch {
            Write-Host "  -> ERREUR: $($_.Exception.Message)" -ForegroundColor Red
        }
    }
}

# Désinstallation SQL Server Standard
Write-Host "`n=== Étape 4: Désinstallation SQL Server 2017 Standard ==="
-ForegroundColor Cyan
Write-Host "Récupération du GUID de désinstallation..." -ForegroundColor Yellow

$uninstallKeys = @(
    "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*",
    "HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*"
)

$sqlProduct = Get-ItemProperty $uninstallKeys -ErrorAction SilentlyContinue |
    Where-Object { $_.DisplayName -like "*SQL Server 2017*" -and $_.DisplayName -like
"*Database Engine*" }

if ($sqlProduct) {
    Write-Host "Produit trouvé: $($sqlProduct.DisplayName)" -ForegroundColor White
    Write-Host "Désinstallation en cours... (Cela peut prendre plusieurs minutes)"
-ForegroundColor Yellow
  #  $uninstallString = $sqlProduct.UninstallString

    # Extraction du GUID si présent
   # if ($uninstallString -match '\{[A-F0-9-]+\}') {
   #     $guid = $matches[0]
   #     Write-Host "GUID: $guid" -ForegroundColor Gray
   #     Start-Process -FilePath "msiexec.exe" -ArgumentList "/x $guid /qn /norestart" -Wait
   #     Write-Host "  -> Désinstallation terminée" -ForegroundColor Green
   # } else {
        Write-Host "  -> Utilisation de la désinstallation alternative" -ForegroundColor Yellow
        # Alternative avec setup.exe
        $uninstallArgs = "/Action=Uninstall /FEATURES=SQL
/INSTANCENAME=$instanceName /Q"
        Start-Process -FilePath $sqlSetupPath -ArgumentList $uninstallArgs -Wait
        Write-Host "  -> Désinstallation terminée" -ForegroundColor Green
    #}
} else {
    Write-Host "Aucune installation SQL Server trouvée pour désinstallation"
-ForegroundColor Yellow
}
```