**Architecture & flux de sauvegarde – Google Apps Script (CLASP) ↔ GitHub  
Windows 11 Famille**

*Version imprimable • Généré le 24/08/2025 à 01:23*

Ce document décrit l’architecture recommandée pour sauvegarder vos projets Google Apps Script (via CLASP) dans un dépôt GitHub, avec un script PowerShell d’orchestration, un dossier de logs et une routine quotidienne sécurisée.

**I. Arborescence recommandée**

/ (racine du dépôt GitHub)

├─ 01\_Moteur/ # Projet Apps Script A (CLASP)

│ ├─ .clasp.json # Lie ce dossier au Script ID du projet GAS A

│ ├─ appsscript.json # Manifest Apps Script

│ ├─ MoteurV2.js

│ ├─ InterfaceV2.js

│ └─ ...

├─ 03\_BaseDeDonnées/ # Projet Apps Script B (CLASP)

│ ├─ .clasp.json # Script ID du projet GAS B

│ ├─ appsscript.json

│ └─ Code.js

├─ Tools/

│ └─ backup\_gas.ps1 # Script PowerShell d’orchestration (sauvegardes)

├─ Logs/

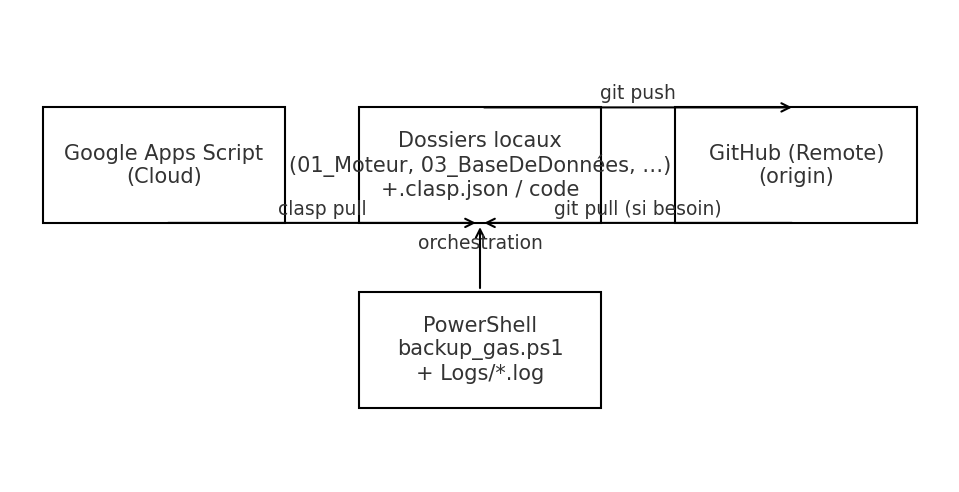
│ └─ backup\_YYYYMMDD\_HHMMSS.log # Journal d’un run (un fichier par exécution)

├─ .gitignore # Exclusions Git (node\_modules/, \*.log, etc.)

├─ README.md # Guide d’utilisation (où sont les projets, comment lancer le backup)

└─ package.json (optionnel) # Scripts npm utiles (ex: \"backup\": \"pwsh .\\Tools\\backup\_gas.ps1\")

**II. Schéma de flux (vue d’ensemble)**



• Le script PowerShell coordonne les opérations : pour chaque projet, il exécute d’abord un ‘clasp pull’ (synchronisation depuis l’éditeur Apps Script), puis un cycle Git (add/commit/push) vers GitHub. Les sorties sont capturées dans un fichier de log horodaté.

**III. Gabarit PowerShell – backup\_gas.ps1 (prêt à copier-coller)**

À enregistrer dans Tools\backup\_gas.ps1 ; lancez-le depuis la racine du repo :

pwsh .\Tools\backup\_gas.ps1

Contenu suggéré :

# Tools/backup\_gas.ps1

param(

[string[]]$Projects = @("01\_Moteur", "03\_BaseDeDonnées"), # Dossiers des projets CLASP

[string]$RepoRoot = "$PSScriptRoot\.." # Racine du dépôt (un cran au-dessus de Tools)

)

# 1) Journal horodaté

$ts = (Get-Date).ToString("yyyyMMdd\_HHmmss")

$logsDir = Join-Path $RepoRoot "Logs"

New-Item -ItemType Directory -Force -Path $logsDir | Out-Null

$logFile = Join-Path $logsDir ("backup\_{0}.log" -f $ts)

Start-Transcript -Path $logFile -Append | Out-Null

function Invoke-Step {

param([string]$Cmd, [string]$WorkDir)

Push-Location $WorkDir

try {

Write-Host "→ $Cmd"

iex $Cmd

if ($LASTEXITCODE -ne $null -and $LASTEXITCODE -ne 0) {

throw "Commande en échec ($LASTEXITCODE): $Cmd"

}

} finally {

Pop-Location

}

}

function Has-Changes {

param([string]$WorkDir)

Push-Location $WorkDir

try {

$out = git status --porcelain

return -not [string]::IsNullOrWhiteSpace($out)

} finally {

Pop-Location

}

}

try {

Set-Location $RepoRoot

foreach ($p in $Projects) {

Write-Host "=== Projet: $p ==="

$projPath = Join-Path $RepoRoot $p

# 2) Synchroniser depuis Apps Script

Invoke-Step -Cmd "clasp status" -WorkDir $projPath

Invoke-Step -Cmd "clasp pull" -WorkDir $projPath

# 3) Versionner côté Git

if (Has-Changes -WorkDir $projPath) {

Invoke-Step -Cmd "git add ." -WorkDir $projPath

Invoke-Step -Cmd "git commit -m `"Backup GAS -> GitHub ($ts) [$p]`"" -WorkDir $projPath

Invoke-Step -Cmd "git push" -WorkDir $projPath

} else {

Write-Host "Aucun changement à valider pour $p"

}

}

# 4) Rotation des logs > 30 jours (optionnel)

Get-ChildItem $logsDir -Filter "backup\_\*.log" |

Where-Object { $\_.LastWriteTime -lt (Get-Date).AddDays(-30) } |

Remove-Item -Force -ErrorAction SilentlyContinue

} catch {

Write-Error $\_

} finally {

Stop-Transcript | Out-Null

}

**IV. Paramétrages essentiels (.clasp.json, .gitignore)**

**A. .clasp.json (par projet)**

{

"scriptId": "1AbcDEF…XYZ",

"rootDir": "src",

"fileExtension": "gs"

}

• `scriptId` : identifiant du projet Apps Script (depuis l’URL de l’éditeur).

• `rootDir` : dossier local où résident les fichiers du projet (optionnel).

• `fileExtension` : extension locale (.gs ou .js).

**B. .clasp.ignore (côté push vers Apps Script)**

# Fichiers à ignorer lors de 'clasp push'

node\_modules/

\*.map

README.md

**C. .gitignore (côté Git/GitHub)**

# Ignorer les modules et les logs

node\_modules/

Logs/\*.log

# Mac/Windows temporaires

.DS\_Store

Thumbs.db

# CLASP (si vous jugez sensible de le versionner)

#.clasp.json

**V. Check‑list de mise en place**

• Installer Git for Windows, GitHub Desktop, VS Code, Node.js (LTS) et CLASP (npm i -g @google/clasp).

• Se connecter à GitHub Desktop et cloner ou publier le dépôt.

• Dans chaque dossier projet (01\_Moteur, 03\_BaseDeDonnées…), configurer .clasp.json ; tester 'clasp login' puis 'clasp pull'.

• Créer Tools/backup\_gas.ps1 avec le gabarit ci‑dessus ; tester en local (pwsh .\Tools\backup\_gas.ps1).

• Vérifier la création du log dans Logs/ et la présence de commits/push sur GitHub.

• Configurer .gitignore /.clasp.ignore selon vos politiques (secrets, cache…).

**VI. Planification (Planificateur de tâches Windows)**

• Ouvrir « Planificateur de tâches » → Créer une tâche.

• Déclencheur : quotidien (ex. 08:00).

• Action : Démarrer un programme → Programme/script : pwsh.exe (ou powershell.exe).

• Arguments : -File "C:\Chemin\vers\repo\Tools\backup\_gas.ps1"

• Démarrer dans : C:\Chemin\vers\repo

• Cocher « Exécuter avec les autorisations maximales » si besoin (accès réseau).

**VII. Dépannage (erreurs classiques)**

|  |  |
| --- | --- |
| Symptôme | Piste de résolution |
| clasp : commande introuvable | Vérifier Node.js/NPM et que le dossier NPM global est dans PATH. Réinstaller : npm i -g @google/clasp |
| clasp login ne s’ouvre pas | Essayer un autre navigateur par défaut ; lancer 'clasp login --no-localhost'. |
| git push demande des identifiants | Configurer SSH ou PAT (Token) avec Git Credential Manager. |
| Permission Apps Script (web app) | Déployer une version et vérifier « Who has access » dans l’interface Google. |
| Conflits Git | Faire git pull/rebase, résoudre les conflits dans VS Code, commit puis push. |

**VIII. Variantes & extensions**

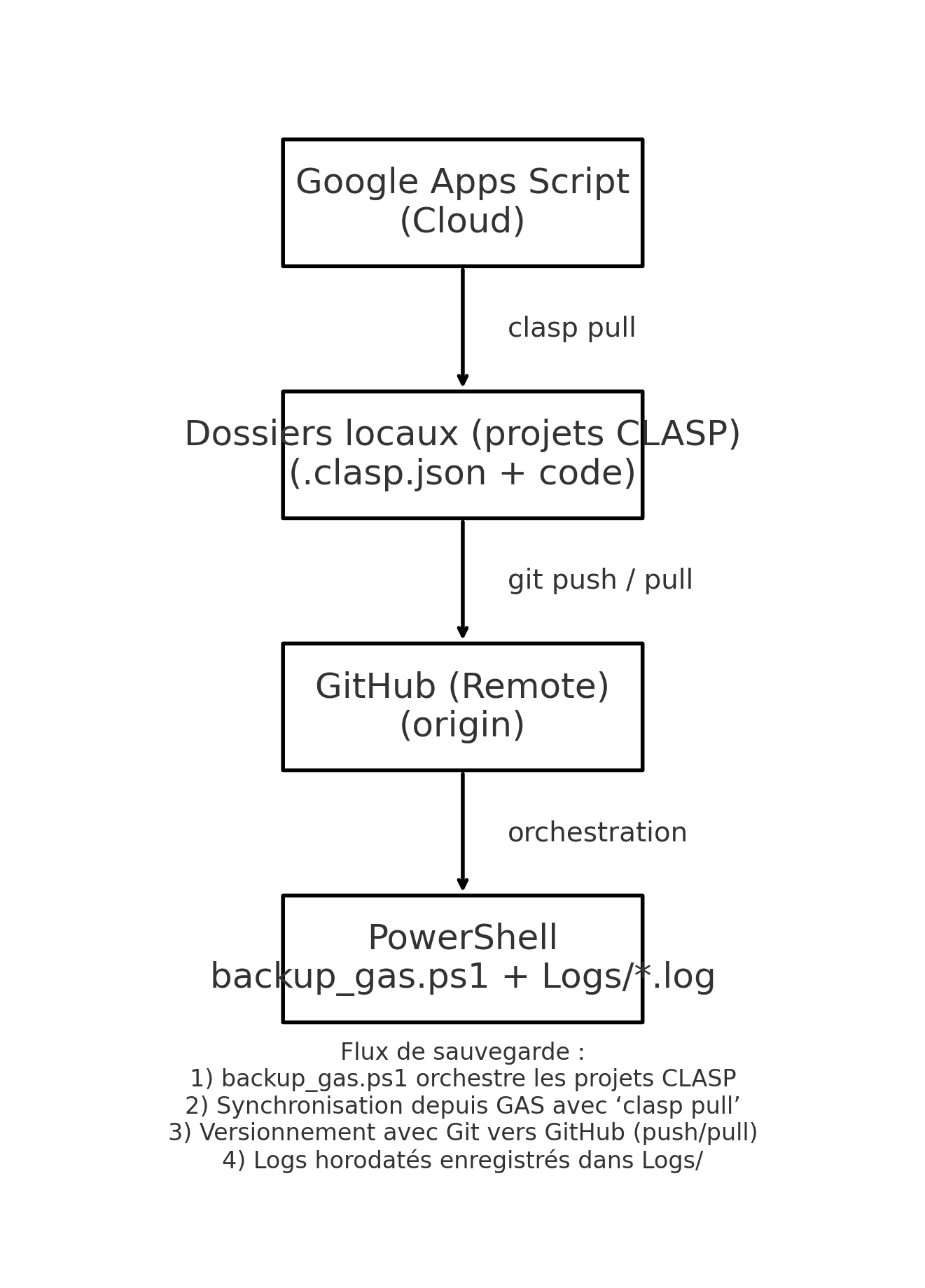
• Multi‑repos : un dépôt par projet Apps Script (le script PowerShell peut itérer sur une liste de répertoires absolus).

• Intégration CI : exécuter le script sur un runner auto‑hébergé selon une planification centralisée.

• Sécurité : stocker des secrets (PAT/SSH) dans le Gestionnaire d’informations d’identification Windows.

• Logs : rotation hebdomadaire, compression (.zip) et archivage sur NAS si souhaité.

**Annexe – Schéma de flux vertical (lisible)**



*Annexe ajoutée le 24/08/2025 à 01:37*