

PowerShell-Lösung für das Senden einer E-Mail über dein GMail-Konto mit OAuth2

- einen einmaligen Login-Schritt
- einen dauerhaften Zugang via refresh_token
- komplette Gmail API Integration via PowerShell

All-in-One PowerShell-Skript

(am Ende)

Das Prinzip in 3 Schritten

1. Einmaliger Login via Google (Code eintippen)

- Du bekommst access_token + refresh_token
- Speichere beide lokal (z. B. in token.json oder settings.ini)

2. Ab dann: automatisches Refresh beim Start

- Tool prüft, ob access_token fehlt oder abgelaufen ist
- Holt mit dem gespeicherten refresh_token ein neues access_token

3. Danach sofort Mail versenden

- Kein Login, kein Code-Eingeben, kein Aufwand

1. refresh.ps1

- Ruft <https://oauth2.googleapis.com/token> auf
- Nutzt refresh_token, um access_token neu zu holen
- Speichert beides lokal

2. sendmail.ps1

- Liest access_token aus Datei
- Kodiert MIME-Mail
- Sendet sie via Gmail-API

3. send.ps1 (Master-Script)

powershell

```
.\refresh.ps1 # Holt neues Access Token, wenn nötig .\sendmail.ps1 # Sendet E-Mail mit aktuellem Access Token
```

Du kannst dieses send.ps1 jederzeit direkt starten –
oder aus Xbase++ / Harbour / Taskplaner aufrufen –
und es wird zuverlässig Mails über dein GMail-Konto senden, auch in einem Jahr noch.

Automatisch access_token aktualisieren per refresh_token

Funktionsprinzip:

[Start Programm]

- Lies gespeicherten refresh_token
- POST an Google zum Erneuern des access_token
- access_token speichern
- E-Mail per API senden

send-gmail.ps1

Du brauchst dafür nur ein gültiges **Access Token** (z. B. via vorherigem Login mit refresh_token)
Das Token trägst du **einmal oben ein**, und das Skript erledigt den Rest.

PowerShell: Gmail API Mailversand mit OAuth2

```
# =====  
# PowerShell: Gmail API Mailversand mit OAuth2  
# =====  
# Voraussetzung:  
# - Du hast bereits ein gültiges access_token  
# - Du hast Gmail API in der Google Cloud Console aktiviert  
# - Dein Google-Konto hat Zugriff über den OAuth2-Flow erlaubt  
  
# 📄 Access Token einfügen (am besten per refresh.ps1 vorher holen)  
$token = "ya29.a0AfH6SMC..." # <-- hier dein gültiges access_token eintragen  
  
# 📧 E-Mail-Inhalt definieren  
$to = "empfaenger@example.com"  
$subject = "Testmail per PowerShell"  
$body = "Hallo! Diese E-Mail wurde mit der Gmail API + PowerShell gesendet."  
  
# 📧 1. MIME-Format vorbereiten (so wie eine echte E-Mail)  
$msg = "From: me`r`nTo: $to`r`nSubject: $subject`r`nContent-Type: text/plain; charset=UTF-8`r`n`r`n$body"  
  
# 📧 2. In Base64url kodieren (so verlangt es die Gmail API)  
# - UTF8 → Bytes → Base64 → URL-safe Format  
$bytes = [System.Text.Encoding]::UTF8.GetBytes($msg)  
$base64 = [System.Convert]::ToBase64String($bytes) `br/>    -replace '+', '-' `br/>    -replace '/', '_' `br/>    -replace '=' # Padding entfernen  
  
# 📄 3. JSON-Body für Gmail API vorbereiten  
$json = @{ raw = $base64 } | ConvertTo-Json -Compress  
  
# 🌐 4. E-Mail per HTTP POST an Gmail API senden  
$response = Invoke-RestMethod `br/>    -Uri "https://gmail.googleapis.com/gmail/v1/users/me/messages/send" `br/>    -Method POST `br/>    -Headers @{ Authorization = "Bearer $token" } `br/>    -ContentType "application/json" `br/>    -Body $json  
  
# ✅ 5. Rückmeldung ausgeben  
Write-Host "Antwort von Gmail API:"  
$response
```

Wie du es startest:

1. Speichere es z. B. als **send-gmail.ps1**
2. Öffne PowerShell (als Benutzer reicht)
3. Führe aus:

PowerShell -ExecutionPolicy Bypass -File .\send-gmail.ps1

refresh-token.ps1

- 🔑 **Du brauchst nur den einmal gespeicherten refresh_token**
- ✅ Danach kannst du das Skript beliebig oft aufrufen – und bekommst jedes Mal ein frisches access_token

PowerShell: Access Token mit refresh_token erneuern

Wie kommt man an den refresh_token?

🔄 Nur einmal nötig:

1. Du baust dir eine Google-Login-URL:

https://accounts.google.com/o/oauth2/v2/auth?client_id=...&response_type=code&scope=...&access_type=offline

- Du öffnest diese URL im Browser und meldest dich mit deinem Google-Konto an.
- Nach der Einwilligung bekommst du **einen Code**.
- Du tauschst diesen Code **einmalig** gegen:
 - ein access_token (gültig für ~1 Stunde)
 - ein refresh_token (**dauerhaft gültig!**)

Das geht z. B. mit PowerShell:

```
Invoke-RestMethod -Uri "https://oauth2.googleapis.com/token" -Method POST -Body @{
  code      = "CODE_VON_GOOGLE"
  client_id = "DEINE_CLIENT_ID"
  client_secret = "DEIN_CLIENT_SECRET"
  redirect_uri = "http://localhost"
  grant_type = "authorization_code"
}
```

In der Antwort steht dann z. B.:

```
{
  "access_token": "ya29.a0AfH6SMC...",
  "expires_in": 3599,
  "refresh_token": "1//0gA4gXabcDEFghiJKL...",
  ...
}
```

Dieser refresh_token wird gespeichert und ab dann immer wieder verwendet.

💡 Warum steht da \$refresh_token = "1//0gA..." im Script?

Weil du dort **deinen echten, einmal erhaltenen refresh_token eintragen musst**.
Dieser bleibt **gültig, solange du ihn nicht widerrufst** oder das Google-Konto verändert wird.

💡 Best Practice:

- Du führst **einmal** den Auth-Flow durch (im Dev-Modus oder Setup)
- Du kopierst den refresh_token in dein refresh-token.ps1
- Danach brauchst du **nie wieder etwas manuell einzugeben**

Zusammenfassung

Begriff	Gültigkeit	Zweck
access_token	1 Stunde	Authentifizierung für API-Zugriff
refresh_token	dauerhaft	Zugang zum Holen neuer access_token

PowerShell-Skript, mit dem du **einmalig** den Google-Login durchführst, den **Authorization Code** eintippst – und danach automatisch deinen **access_token** + **refresh_token erhältst** und in einer Datei speicherst.

authorize.ps1 – Der einmalige Login-Schritt mit Token-Speicherung

```
# =====
# PowerShell: Gmail API – Initiale Autorisierung und Refresh Token holen
# =====

# 🛠 Konfiguration – bitte mit deinen Werten ersetzen
$client_id = "DEINE_CLIENT_ID"
$client_secret = "DEIN_CLIENT_SECRET"
$redirect_uri = "http://localhost" # Muss mit Google Console übereinstimmen

# 🌐 Schritt 1: URL bauen für Login im Browser
$scope = "https://www.googleapis.com/auth/gmail.send"
$response_type = "code"
$auth_url = "https://accounts.google.com/o/oauth2/v2/auth?
client_id=$client_id&redirect_uri=$redirect_uri&response_type=$response_type&scope=$scope&access_type=offline&prompt=consent"

Write-Host "👉 Öffne im Browser folgenden Link und logge dich mit deinem Google-Konto ein:"
Write-Host ""
Write-Host $auth_url
Write-Host ""

# 📄 Schritt 2: Benutzer gibt Authorization Code ein
$code = Read-Host "❓ Bitte den Autorisierungscode von Google hier einfügen"




# 📄 Schritt 3: Token von Google holen (access + refresh)
$response = Invoke-RestMethod `
    -Uri "https://oauth2.googleapis.com/token" `
    -Method POST `
    -Body @{
        code = $code
        client_id = $client_id
        client_secret = $client_secret
        redirect_uri = $redirect_uri
        grant_type = "authorization_code"
    } `
    -ContentType "application/x-www-form-urlencoded"

# 📄 Schritt 4: Tokens speichern
if ($response.access_token) {
    $tokenData = @{}
    $tokenData.access_token = $response.access_token
    $tokenData.refresh_token = $response.refresh_token
    $tokenData.expires_in = $response.expires_in
    $tokenData.token_type = $response.token_type
    $tokenData.obtained_at = (Get-Date).ToString("s")
}

$tokenData | ConvertTo-Json | Set-Content -Encoding UTF8 "token.json"

Write-Host "`n✅ Erfolgreich! Tokens wurden in 'token.json' gespeichert."
Write-Host "📄 Ab jetzt kannst du mit 'refresh-token.ps1' neue access_token abrufen."
} else {
    Write-Host "`n❌ Fehler beim Abrufen der Tokens!"
    $response | Format-List
}
```

Danach läuft alles automatisch:

1.  **authorize.ps1 einmalig ausführen**
 - Du bekommst die Datei token.json mit deinem refresh_token
2.  **refresh-token.ps1 bei jedem Start**
 - Holt automatisch ein frisches access_token mit dem gespeicherten refresh_token
3.  **send-gmail.ps1 verwendet das aktuelle Token aus token.json**

Inhalt von token.json (Beispiel):

```
{
  "access_token": "ya29.a0AfH6SM...",
  "refresh_token": "1//04aBCDXYZ...",
  "expires_in": 3599,
  "token_type": "Bearer",
  "obtained_at": "2025-03-27T14:55:23"
}
```

```
# =====
# PowerShell: Access Token mit refresh_token erneuern
# =====

# 📁 1. Konfigurationsdaten festlegen
# Diese Werte stammen aus deiner Google OAuth2-App
$client_id = "DEINE_CLIENT_ID"
$client_secret = "DEIN_CLIENT_SECRET"
$refresh_token = "1//04abcdEFGH..." # <--- Hier deinen echten refresh_token eintragen

# 📦 2. Daten für die POST-Anfrage zusammenstellen
$body = @{
  client_id = $client_id
  client_secret = $client_secret
  refresh_token = $refresh_token
  grant_type = "refresh_token"
}

# 🌐 3. Token bei Google anfragen
$response = Invoke-RestMethod `
  -Uri "https://oauth2.googleapis.com/token" `
  -Method POST `
  -Body $body `
  -ContentType "application/x-www-form-urlencoded"

# ✅ 4. Neues access_token speichern (z. B. in JSON-Datei)
if ($response.access_token) {
  $tokenData = @{
    access_token = $response.access_token
    expires_in = $response.expires_in
    token_type = $response.token_type
    obtained_at = (Get-Date).ToString("s")
  }
}

# In Datei speichern (z. B. token.json)
$tokenData | ConvertTo-Json | Set-Content -Encoding UTF8 "token.json"
```

```
Write-Host "Neues Access Token erfolgreich gespeichert!"
} else {
    Write-Host "Fehler beim Abrufen des Tokens:"
    $response | Format-List
}
```

Was du bekommst (nach dem Aufruf):

Eine Datei token.json mit z. B.:

```
{
  "access_token": "ya29.a0AfH6S...",
  "expires_in": 3599,
  "token_type": "Bearer",
  "obtained_at": "2025-03-27T15:42:17"
}
```

Wie du es nutzt:

1. refresh-token.ps1 ausführen:

```
PowerShell -ExecutionPolicy Bypass -File .\refresh-token.ps1
```

2. In deinem send-gmail.ps1:

Ersetze:




```
$token = "ya29.a0AfH6SMC..."
```

durch:

```
# Token aus Datei laden $token = (Get-Content "token.json" | ConvertFrom-Json).access_token
```

Ergebnis: Ein dauerhaftes Mailversand-System

Du kannst jetzt:

-  Beim Start automatisch refresh-token.ps1 ausführen
-  Danach send-gmail.ps1 ausführen
-  Oder beides in einem Master-Skript wie run.ps1 kombinieren:

```
.\refresh-token.ps1
```

```
.\send-gmail.ps1
```

Dann brauchst du **nicht wieder manuell einloggen** oder einen Code eingeben.

Wo kommt Betreff & Nachricht hin?

Die eigentliche E-Mail erstellst und sendest du mit dem separaten Skript:

send-gmail.ps1

Dort stehen z. B. folgende Zeilen:

```
# 📧 E-Mail-Inhalt definieren
$to    = "empfaenger@example.com"
$subject = "Testmail per PowerShell"
$body   = "Hallo! Diese E-Mail wurde mit der Gmail API + PowerShell gesendet."
```

send-gmail-full.ps1 – Alles in einem Skript

✅ So verwendest du das Skript:

1. **Speichere es als** send-gmail-full.ps1
2. **Passe oben** client_id, client_secret, redirect_uri **an**
3. **Setze \$to, \$subject, \$body nach Wunsch**
4. Führe es aus:

```
PowerShell -ExecutionPolicy Bypass -File .\send-gmail-full.ps1
```

Vorteile:

- Alles in einem Script
- Automatische Speicherung in token.json
- Funktioniert dauerhaft, keine erneute Anmeldung nötig
- Einfach in Xbase++, Harbour, oder geplanter Task integrierbar

```
# =====
# PowerShell All-in-One: Gmail API E-Mailversand
# =====
# 1. Führt bei Bedarf erstmaligen Login durch (Code eingeben)
# 2. Holt automatisch ein gültiges access_token via refresh_token
# 3. Sendet E-Mail über Gmail API
# =====

# 🛠 Konfiguration: einmal eintragen
$client_id  = "DEINE_CLIENT_ID"
$client_secret = "DEIN_CLIENT_SECRET"
$redirect_uri = "http://localhost" # Muss mit Google Console übereinstimmen

# 📧 Deine E-Mail-Daten
$to    = "empfaenger@example.com"
$subject = "Testmail über Gmail API"
$body   = "Hallo! Diese Nachricht wurde mit PowerShell und der Gmail API gesendet."


# 📁 Speicherdatei für Tokens
$tokenFile = "token.json"

# 🧠 Hilfsfunktion: Token aus Datei laden
function Load-Token {
    if (Test-Path $tokenFile) {
        return Get-Content $tokenFile | ConvertFrom-Json
    }
    return $null
}

# 📁 Hilfsfunktion: Token in Datei speichern
function Save-Token($data) {
```

```

$data | ConvertTo-Json | Set-Content -Encoding UTF8 $tokenFile
}

#  Funktion: Access Token automatisch holen (neu oder via refresh_token)
function Get-AccessToken {
    $tokenData = Load-Token

    # Fall 1: Kein refresh_token vorhanden → Login notwendig
    if (-not $tokenData -or -not $tokenData.refresh_token) {
        Write-Host "🔴 Kein Token gefunden. Starte erstmalige Autorisierung..."

        $scope = "https://www.googleapis.com/auth/gmail.send"
        $auth_url = "https://accounts.google.com/o/oauth2/v2/auth?
client_id=$client_id&redirect_uri=$redirect_uri&response_type=code&scope=$scope&access_type=offline&prompt=consent"

        Write-Host "`n👉 Öffne im Browser diesen Link und logge dich ein:"
        Write-Host $auth_url
        Write-Host ""

        $code = Read-Host "🔴 Bitte den Autorisierungscode von Google hier einfügen"

        # Tausche Code gegen access_token und refresh_token
        $response = Invoke-RestMethod `
            -Uri "https://oauth2.googleapis.com/token" `
            -Method POST `
            -Body @{
                code      = $code
                client_id = $client_id
                client_secret = $client_secret
                redirect_uri = $redirect_uri
                grant_type = "authorization_code"
            } `
            -ContentType "application/x-www-form-urlencoded"

        if ($response.access_token) {
            $tokenData = @{
                access_token = $response.access_token
                refresh_token = $response.refresh_token
                expires_in    = $response.expires_in
                obtained_at   = (Get-Date).ToString("s")
            }
            Save-Token $tokenData
            Write-Host "`n🟢 Erstes Login erfolgreich. Token gespeichert."
        } else {
            Write-Host "`n🔴 Fehler beim Abrufen des Tokens."
            exit
        }
    }

    # Fall 2: Es gibt bereits ein refresh_token → Access Token erneuern
    $refresh_token = $tokenData.refresh_token

    $response = Invoke-RestMethod `
        -Uri "https://oauth2.googleapis.com/token" `
        -Method POST `
        -Body @{
            client_id      = $client_id
            client_secret = $client_secret
            refresh_token = $refresh_token
            grant_type      = "refresh_token"
        } `
        -ContentType "application/x-www-form-urlencoded"

    if ($response.access_token) {
        $tokenData.access_token = $response.access_token
        $tokenData.obtained_at = (Get-Date).ToString("s")
        Save-Token $tokenData
        return $response.access_token
    } else {
        Write-Host "🔴 Fehler beim Refresh des Tokens."
        exit
    }
}

```

```

}
}

# ✉ Funktion: E-Mail senden über Gmail API
function Send-GmailMail($access_token, $to, $subject, $body) {
    # MIME-Format vorbereiten
    $msg = "From: me`r`nTo: $to`r`nSubject: $subject`r`nContent-Type: text/plain; charset=UTF-8`r`n`r`n$body"
    $bytes = [System.Text.Encoding]::UTF8.GetBytes($msg)
    $base64 = [System.Convert]::ToBase64String($bytes) `
        -replace '\+', '_`' `
        -replace '/', '_`' `
        -replace '='

    $json = @{ raw = $base64 } | ConvertTo-Json -Compress

    $response = Invoke-RestMethod `
        -Uri "https://gmail.googleapis.com/gmail/v1/users/me/messages/send" `
        -Method POST `
        -Headers @{ Authorization = "Bearer $access_token" } `
        -ContentType "application/json" `
        -Body $json

    Write-Host "`n 📧 E-Mail wurde gesendet!"
}

# ▶ Ablauf starten
$access_token = Get-AccessToken
Send-GmailMail -access_token $access_token -to $to -subject $subject -body $body

```

📌 Hier nochmal zur Klarstellung:

🔑 Du brauchst diesen Schritt nur EINMAL:

1. Du baust die URL:

```

https://accounts.google.com/o/oauth2/v2/auth?
client_id=DEINE_CLIENT_ID
&redirect_uri=http://localhost
&response_type=code
&scope=https://www.googleapis.com/auth/gmail.send
&access_type=offline
&prompt=consent

```

2. Du öffnest diesen Link im Browser → Google-Login + Zustimmung

3. Google zeigt dir einen einmaligen code an (z. B. 4/0AY0e-g7xXK...)

4. Du tauscht diesen code gegen Tokens:

```

Invoke-RestMethod -Uri "https://oauth2.googleapis.com/token" -Method POST -Body @{
    code      = "HIER_DEIN_CODE"
    client_id  = "DEINE_CLIENT_ID"
    client_secret = "DEIN_CLIENT_SECRET"
    redirect_uri = "http://localhost"
    grant_type = "authorization_code"
} -ContentType "application/x-www-form-urlencoded"

```

5. Die Antwort enthält:

```

{
  "access_token": "ya29.a0AfH6SMC...",
  "expires_in": 3599,
  "refresh_token": "1//0gA4gXabcDEFghiJKL...",
  ...
}

```

✅ Wichtig:

- access_token ist nur **ca. 1 Stunde gültig**
- refresh_token ist **dauerhaft gültig** (wird nur **beim allerersten Login** zurückgegeben!)
- Diesen refresh_token **musst du speichern** (z. B. token.json)

Ab dann brauchst du **nie wieder diesen Code oder Browser**, sondern kannst bei jedem Start automatisch ein neues access_token holen und **dauerhaft E-Mails senden**.

💡 Tipp: In der Login-URL ist **access_type=offline entscheidend**, sonst bekommst du keinen refresh_token!

Paket-Inhalt

✅ Datei 1: authorize.bat

→ Führt die PowerShell-Datei zum erstmaligen Login aus.

```
@echo off PowerShell -ExecutionPolicy Bypass -File authorize.ps1 pause
```

✅ Datei 2: authorize.ps1

→ Öffnet den Browser-Link, fragt den Code ab, speichert access_token + refresh_token in token.json.

```
# =====
# PowerShell: Gmail API – Erste Anmeldung und Tokens holen
# =====

# 🛠 Konfiguration (deine Google-Daten hier eintragen)
$client_id = "DEINE_CLIENT_ID"
$client_secret = "DEIN_CLIENT_SECRET"
$redirect_uri = "http://localhost" # Muss mit deiner Google OAuth2-App übereinstimmen

# 🌐 Login-Link erzeugen
$scope = "https://www.googleapis.com/auth/gmail.send"
$auth_url = "https://accounts.google.com/o/oauth2/v2/auth?
client_id=$client_id&redirect_uri=$redirect_uri&response_type=code&scope=$scope&access_type=offline&prompt=consent"

Write-Host "`n 🖱 Öffne im Browser diesen Link und logge dich mit deinem Google-Konto ein:"
Start-Process $auth_url

# 📄 Benutzer gibt den Autorisierungscode ein
$code = Read-Host "`n ? Bitte den Autorisierungscode von Google hier eingeben"

# 🔄 Code gegen Tokens eintauschen
$response = Invoke-RestMethod `
-Uri "https://oauth2.googleapis.com/token" `
-Method POST `
-Body @{
    code = $code
    client_id = $client_id
    client_secret = $client_secret
    redirect_uri = $redirect_uri
    grant_type = "authorization_code"
} `
-ContentType "application/x-www-form-urlencoded"

# 💾 Tokens speichern
if ($response.access_token) {
    $tokenData = @{}
    $tokenData.access_token = $response.access_token
    $tokenData.refresh_token = $response.refresh_token
    $tokenData.expires_in = $response.expires_in
    $tokenData.token_type = $response.token_type
```

```

    obtained_at = (Get-Date).ToString("s")
}

$tokenData | ConvertTo-Json | Set-Content -Encoding UTF8 "token.json"

Write-Host "`n✅ Tokens wurden erfolgreich in 'token.json' gespeichert."
} else {
    Write-Host "`n❌ Fehler beim Abrufen der Tokens:"
    $response | Format-List
}

```

📦 So verwendest du das Paket:

1. Lege beide Dateien (authorize.bat und authorize.ps1) in einen Ordner.
2. Trage deine Werte in authorize.ps1 ein:
 - DEINE_CLIENT_ID
 - DEINE_CLIENT_SECRET
 - redirect_uri (z. B. http://localhost)
3. Doppelklick auf authorize.bat.
4. Der Browser öffnet sich → du loggst dich ein.
5. Du bekommst einen Code → diesen ins PowerShell-Fenster kopieren.
6. Das Skript speichert die Tokens in token.json.

✅ Danach:

Du kannst dein send-gmail.ps1 oder ein automatisches Script wie refresh-token.ps1 verwenden – dauerhaft, ohne Browser oder Interaktion.

Hier ist dein ZIP-Starterpaket für den einmaligen Google-Login und das Speichern des refresh_token:

📄 Gmail OAuth2 Starterpaket herunterladen (authorize.bat + authorize.ps1)

👉 Trage deine client_id und client_secret in die Datei authorize.ps1 ein, starte authorize.bat – und schon bekommst du deine Token als token.json.

Hier ist dein aktualisiertes ZIP-Paket mit:

- ✅ authorize.bat
- ✅ authorize.ps1 → für den einmaligen Google-Login
- ✅ send-gmail.ps1 → zum E-Mail-Versand mit dem gespeicherten Token

📄 Gmail OAuth2 Starterpaket herunterladen (inkl. Mailversand)

👉 Nach dem ersten Login kannst du send-gmail.ps1 beliebig oft aufrufen, um E-Mails über dein GMail-Konto zu senden – ganz ohne erneute Anmeldung.


Hier ist dein aktualisiertes ZIP-Paket – jetzt zusätzlich mit:

📄 send-gmail-html-attachment.ps1
→ für den Versand von **HTML-E-Mails mit PDF-Anhang** über die Gmail API.

📄 Gmail OAuth2 Starterpaket herunterladen (inkl. HTML + Anhang)

👉 Lege eine Datei mit dem Namen `testdatei.pdf` in den Ordner – oder ändere den Dateinamen im Skript.
Dann einfach per Doppelklick aufrufen oder per Task starten – die Mail wird mit HTML + Anhang versendet.

Dein ZIP-Paket enthält jetzt zusätzlich:

 `send-gmail-multi-attachment.ps1`

→ für den Versand von HTML-E-Mails mit **mehreren Anhängen** (PDF, JPG, DOCX, PNG ...).


▼ Gmail OAuth2 Starterpaket herunterladen (inkl. Multi-Attachment)

👉 Du kannst in der Datei einfach weitere Pfade in die Zeile eintragen:


```
$attachments = @("anhang1.pdf", "anhang2.jpg", "anhang3.docx")
```

– und das Skript erledigt den Rest: MIME, Base64, API-Aufruf.

Dein ZIP-Paket enthält jetzt auch:

 `send-gmail-inline-template.ps1`

→ für HTML-Mails mit **eingebetteten Bildern** (inline via `cid:image1`, `cid:image2` etc.)


 `template.html`

→ eine Beispiel-HTML-Vorlage mit Platzhaltern für Bilder

▼ Gmail OAuth2 Starterpaket herunterladen (mit HTML-Vorlage & Bildauswahl)

👉 Wenn du das Skript ausführst, öffnet sich ein Dateidialog. Du wählst 1–n Bilder aus, und sie werden in die HTML-Mail eingebettet – ideal für **individuelle Designs, Logos, Fotos**.


Hier ist dein aktualisiertes PowerShell-Starterpaket mit erweiterten Funktionen:

 `send-gmail-dynamic.ps1`

→ für personalisierte HTML-Mails mit Platzhalter `{{name}}`, Empfängerliste aus `.csv`, **und automatisch generiertem PDF-Anhang aus Word**

 `empfaenger.csv`

→ Beispiel-Empfängerliste mit `name,email`


 `template.html`

→ HTML-Vorlage mit dem Platzhalter `{{name}}`

📎 Du brauchst nur noch eine Datei `beispielbrief.docx` im Ordner – daraus wird automatisch ein PDF für den Anhang erstellt.

▼ Gmail OAuth2 Starterpaket herunterladen (mit Platzhaltern, CSV & Word zu PDF)

✅ Fertig! Hier ist dein finales Starterpaket mit grafischer Oberfläche (GUI) für den Gmail-Versand – jetzt mit allem drin:

 `send-gmail-gui.ps1`


→ Eine PowerShell-GUI, die folgendes kann:

- Empfängerdaten aus `.csv` laden
- Eine Word-Vorlage automatisch mit Platzhaltern füllen (`{{name}}`, `{{adresse}}`, ...)
- Das personalisierte Word-Dokument **automatisch in PDF umwandeln**

- Eine HTML-Mail mit Platzhaltern verschicken
- Den PDF-Brief als **Anhang** beilegen
- **Alles automatisch & in einem Klick!**

 Beispiel-Dateien:

- empfaenger.csv
- template.html

 Du brauchst nur noch eine Word-Vorlage briefvorlage.docx mit Platzhaltern – der Rest läuft automatisch.

Wenn du aus dem PowerShell-Skript eine **EXE machen möchtest**, reden wir über ein sogenanntes:

✓ Self-contained PowerShell-Tool als .exe

Das bedeutet:

- Du wandelst dein .ps1-Skript in eine eigenständig ausführbare .exe-Datei um.
- Die EXE kann **wie ein normales Windows-Programm doppelt geklickt** oder per Autostart/Taskplaner ausgeführt werden.
- Nutzer brauchen **kein PowerShell-Fenster**, keinen Editor, keine Konsole.

🔧 Wie kann man das machen?

Variante 1: Mit [PS2EXE \(Open Source\)](#)

Ein Tool, das dein .ps1 zu .exe kompiliert.


powershell

```
Install-Module -Name ps2exe -Scope CurrentUser
```

Dann z. B.:

powershell

```
Invoke-ps2exe .\send-gmail-gui.ps1 .\send-gmail-gui.exe -noConsole -icon myicon.ico
```

 Ergebnis:

- send-gmail-gui.exe ist eine voll funktionsfähige Windows-Anwendung.
- Optional mit eigenem Icon.
- PowerShell ist zwar im Hintergrund erforderlich, aber **versteckt**.

Variante 2: Mit ISESteroids, PowerShell Studio oder Advanced Installer

Diese Tools bieten kommerzielle Funktionen, um PS-Skripte grafisch als Setup oder EXE zu bauen – z. B. mit Installer, Icons, Manifest etc.

🧩 Was steckt in der EXE?

Die .exe enthält:

Bestandteil	Beschreibung
Dein Skript (send-gmail-gui.ps1)	GUI, Logik, Mailversand





Bestandteil	Beschreibung
Eingebettete Ressourcen	(optional) Icon, Vorlage, HTML-Text, CSV, z. B. als Temp-Dateien
Laufzeit-Anforderung	PowerShell 5+ auf Windows (meist vorinstalliert)

Und wie funktioniert das?

Du kannst die .exe z. B.:

- In ein Desktop-Symbol legen
- Automatisch bei PC-Start oder RDP-Login starten
- In Xbase++ oder Harbour per RUN oder Shell aufrufen

Zusammenfassung: .exe = Benutzerfreundliches Frontend

Vorteil	Was du bekommst
 Doppelklick-fähig	Keine Kommandozeile nötig
 Passwort/Token im Hintergrund	Kein Code für User sichtbar
 GUI bleibt gleich	Deine PS-Logik lebt in der EXE weiter
 Weiterverteilbar	Für andere Mitarbeiter/Kunden

 **Ja – das ist einer der größten Vorteile einer .exe:**

 Die Variablen `$client_id` und `$client_secret` sind dann nicht mehr sichtbar für den Endbenutzer.

Warum?

Wenn du dein PowerShell-Skript als .ps1 weitergibst:

- kann es **jeder mit Notepad öffnen**
- alle Variablen (inkl. `client_secret`) sind **im Klartext lesbar**

Wenn du es in eine .exe kompilierst (z. B. mit `ps2exe`):



- ist der gesamte Inhalt in der .exe verpackt
- **keine einfache Möglichkeit**, den Code oder die Secrets zu sehen
- **kein Rechtsklick → Öffnen mit Editor**

ABER (fairerweise):

Technisch gesehen:

- ist die .exe **nicht vollständig verschlüsselt**
- mit Tools wie **ILSpy, Resource Hacker oder speziellen Extractoren** kann man theoretisch *versuchen*, die Originalzeichenfolgen auszulesen
- **aber für normale Anwender und Mitarbeiter ist das praktisch ausgeschlossen**

Wie sicher ist das?

Sicherheitsniveau	Zugriff auf <code>client_secret</code> möglich?
.ps1 offen im Ordner	 JA, sofort sichtbar
.ps1 mit Passwortschutz	 Nur bedingt

Sicherheitsniveau	Zugriff auf client_secret möglich?
.exe mit ps2exe	✓ KAUM sichtbar, praktisch geschützt
.exe + obfuscation	✓ ✓ Sehr gut geschützt

✖ Wenn du's maximal sicher willst:

- Speichere client_id + client_secret nicht direkt im Skript
- Lade sie **extern (verschlüsselt)** z. B. aus:
 - client.json + Protect-CmsMessage
 - Registry
 - Umgebungsvariablen
- Oder verschlüssele sie im Skript und entschlüssele zur Laufzeit

Brauch ich bei der EXE Version auch Powershell nicht freischalten ?

ChatGPT:

Super wichtige Frage – und hier die **klare Antwort**:

- ✓ Bei der .exe-Version brauchst du PowerShell *nicht explizit freischalten*
- ✗ Du brauchst **keine Änderung der ExecutionPolicy**, kein Set-ExecutionPolicy, kein Admin-Recht.

🧠 Warum das so ist:

🔧 Bei normalen .ps1-Skripten:

- Windows überprüft die **Ausführungsrichtlinie (ExecutionPolicy)**, z. B.:
 - Restricted (Standard): keine Skripte erlaubt
 - RemoteSigned: nur lokal erlaubt
 - AllSigned, Bypass, etc.

➡ Deshalb funktioniert dein Skript oft **nur mit**:

powershell

Kopieren Bearbeiten

PowerShell -ExecutionPolicy Bypass -File dein_script.ps1

● Bei .exe, die mit ps2exe erstellt wurden:

- Das Skript ist **in die EXE eingebettet**
- Es wird **intern per C#-Host** ausgeführt
- Die **ExecutionPolicy greift nicht mehr** (sie wird umgangen)

✓ Was das konkret für dich bedeutet:

Situation	.ps1 notwendig	.exe notwendig
PowerShell-Sicherheit aktiv	✗ blockiert	✓ läuft sofort
Kein Admin-Recht verfügbar	✗ Problematisch	✓ funktioniert
User hat keine Ahnung von PS	✗ riskant	✓ einfach klickbar
Weitergabe an Kunden/Hotel	✗ eher ungeeignet	✓ voll geeignet



Fazit:

Die EXE ist nicht nur hübscher und benutzerfreundlicher – sie umgeht auch alle ExecutionPolicy-Probleme, weil sie intern wie ein Mini-Programm läuft.

Absolut richtig – und das ist der **nächste logische Schritt**:

➔ Du brauchst zuerst deine `client_id` und `client_secret` von Google, damit dein Tool überhaupt Zugriff auf die Gmail API bekommt.



So bekommst du `client_id` und `client_secret` – Schritt für Schritt



1. Gehe zur Google Cloud Console

Am besten mit dem Google-Konto, über das du später die E-Mails verschicken willst.



2. Projekt erstellen

- Oben links: Menü öffnen → "Projekt erstellen"
- Name z. B.: Gmail-Mailversand
- Erstellen klicken



3. Gmail API aktivieren

- Im Dashboard: „APIs & Dienste“ → „Bibliothek“
- Suche: Gmail API → aktivieren



4. OAuth2-Zugangsdaten erstellen

- Gehe zu „APIs & Dienste“ → „Zugangsdaten“
- Klicke auf „+ Zugangsdaten erstellen“ → „OAuth-Client-ID“
- Wähle **Anwendungstyp: Desktop-App**
- Name z. B.: GmailSender
- Erstellen klicken



Jetzt bekommst du:

- `client_id` (lange Zeichenkette)
- `client_secret`



Am besten klickst du auf „JSON herunterladen“ und speicherst die Datei als `credentials.json`



5. In dein Skript einfügen

Im PowerShell-Skript:

```
$client_id = "DEINE_CLIENT_ID"
$client_secret = "DEIN_CLIENT_SECRET"
```

Oder du lässt das automatisch aus der JSON-Datei einlesen – das kann ich dir auch einbauen.

Hinweis zur Sicherheit:

Wenn du mit einem bestimmten GMail-Konto sendest (z. B. hotel@example.com), sollte dieses Konto die API-Zustimmung einmalig erteilen. Danach kannst du es dauerhaft mit dem `refresh_token` nutzen – wie in deinem Tool vorbereitet.

Ja, du kannst die .exe ganz einfach selbst erstellen!

Alles, was du brauchst, ist ein Tool namens **PS2EXE**, das kostenlos ist und PowerShell-Skripte in .exe-Dateien umwandelt.

So machst du aus deinem PowerShell-Skript eine EXE

1. PS2EXE installieren

Einmalig in PowerShell (als Benutzer, kein Admin nötig):

```
powershell
```

```
Install-Module -Name ps2exe -Scope CurrentUser
```

Wenn du gefragt wirst, ob du NuGet und das Repository akzeptierst → einfach mit J (Ja) bestätigen.

2. EXE erzeugen

Sagen wir, du hast dein Skript z. B. als:

```
send-gmail-gui.ps1
```

Dann erzeugst du deine EXE so:

```
Invoke-PS2EXE .\send-gmail-gui.ps1 .\send-gmail-gui.exe -noConsole
```

Optionale Parameter:

- `-noConsole` → versteckt das schwarze Fenster
- `-icon myicon.ico` → eigenes Icon verwenden (optional)
- `-title "Gmail Versand"` → Fenstername setzen
- `-requireAdmin` → nur wenn unbedingt nötig (eher vermeiden)

3. EXE testen

Die EXE-Datei ist danach komplett selbstständig ausführbar:

- kein PowerShell-Fenster
- kein Skript sichtbar
- Doppelklick → läuft sofort
- kann aus anderen Programmen (z. B. Xbase++, Harbour) gestartet werden

Was steckt drin in der EXE?

- Dein gesamtes .ps1-Skript (komprimiert eingebettet)





- Optional: Ressourcen wie Icons oder Pfade
- Die EXE nutzt im Hintergrund powershell.exe, aber **ohne dass der Benutzer das merkt**

Sicherheit


Wie vorhin besprochen:

- Dein client_secret ist **nicht sichtbar** wie in einem .ps1
- Normale Benutzer können die EXE **nicht einfach öffnen oder bearbeiten**
- Nur mit Spezialtools könnte man theoretisch versuchen, Inhalte zu extrahieren (vergleichbar mit jeder Software)


Fazit:

Vorteil	.exe selbst erstellen bringt dir...
 Kontrolle	Du bestimmst, was drin ist
 Weitergabe	An Kollegen, Kunden, Mitarbeiter
 Schutz vor Skript-Einblick	Kein Klartext sichtbar
 Keine ExecutionPolicy-Probleme	Läuft überall

 Hier ist dein aktualisiertes ZIP-Paket mit einem neuen **Build-Skript**, das deine send-gmail-gui.ps1 automatisch in eine .exe umwandelt:

 **build-exe.ps1**

→ Aufruf von PS2EXE zum Erzeugen einer send-gmail-gui.exe, inkl. Fenster-Titel, optionalem Icon, und ohne PowerShell-Konsole.

 Gmail OAuth2 Starterpaket herunterladen (inkl. EXE-Build-Skript)

So verwendest du das Build-Skript:

1. Öffne PowerShell
2. Navigiere zum Ordner mit dem ZIP-Inhalt
3. Gib ein:

```
Install-Module ps2exe -Scope CurrentUser # nur einmal nötig .\build-exe.ps1
```

Fertig! 🎉 Deine EXE ist einsatzbereit.

Wenn du willst, kann ich auch ein Icon hinzufügen oder einen Installer für Endanwender bauen. Sag einfach Bescheid!

