

OAuth 2.0 und OpenID Connect mit ADFS Application Groups

ADFS · Moderne Protokolle

OAuth 2.0 und OpenID Connect mit Application Groups

Client-ID, Redirect URI, Scopes – drei Felder, an denen jede zweite Anbindung erst einmal scheitert.

boddenberg.de



WISSEN

Grundlagen, Architektur und alle Praxisbeiträge rund um ADFS an einem Ort.

> [Active Directory Federation Services](#)

BERATUNG

Bestehende Application Groups prüfen, aufräumen und den Umzug moderner Apps nach Entra ID planen.

> [Consulting zu ADFS](#)

SCHULUNG

OAuth im Labor: Code abfangen, Token dekodieren, Redirect URI absichtlich verbiegen.

> [ADFS-Schulungen](#)

Irgendwann steht jemand aus der Entwicklung in deiner Tür, Laptop unter dem Arm, und sagt den Satz, der seit einigen Jahren den Satz „Wir brauchen da mal SAML“ abgelöst hat: „Wir machen OpenID Connect. Ich brauche nur eine Client-ID, ein Secret und die Discovery-URL.“ Das klingt nach fünf Minuten. In der Praxis sind es eher fünf Minuten plus zwei Tage, in denen ihr gemeinsam herausfindet, warum ADFS die Redirect URI ablehnt, warum im ID Token der Benutzername fehlt und warum die API jedes Token mit „invalid audience“ zurückweist.

Dieser Beitrag nimmt dir diese zwei Tage ab. Er zeigt, wie ADFS moderne Protokolle über Application Groups abbildet, wann du eine native Anwendung, eine Server-Anwendung oder eine Web-API anlegst, wie Redirect URIs, Scopes und Berechtigungen zusammenspielen, wie lange welches Token lebt – und wie der Authorization-Code-Flow mit ADFS tatsächlich abläuft. Am Ende steht eine ehrliche Einordnung, wann du so etwas heute nicht mehr auf ADFS neu bauen solltest. Grundlagen zu Farm, Zertifikaten und Vertrauensstellungen setzen wir voraus; sie stehen auf der Übersichtsseite [Active Directory Federation Services](#).

FAKTEN — OAuth und OpenID Connect auf ADFS in fünf Sätzen

Application Groups gibt es seit ADFS unter Windows Server 2016. Sie sind der Container, in dem Clients und Web-APIs für OAuth 2.0 und OpenID Connect registriert werden.

Unter Windows Server 2012 R2 gab es nur einen schmalen OAuth-Unterbau mit Add-AdfsClient – so stand es früher in jeder Anleitung, und so sollte es heute nirgends mehr stehen.

PKCE für den Authorization-Code-Flow unterstützt ADFS seit Windows Server 2019, mit den Methoden plain und S256.

ADFS stellt Access Token, ID Token und Refresh Token aus; Access und ID Token sind JWTs, das Refresh Token ist für den Client undurchsichtig.

Die Protokoll-Metadaten liegen im Beispiel unter <https://sts.contoso.de/adfs/.well-known/openid-configuration>.

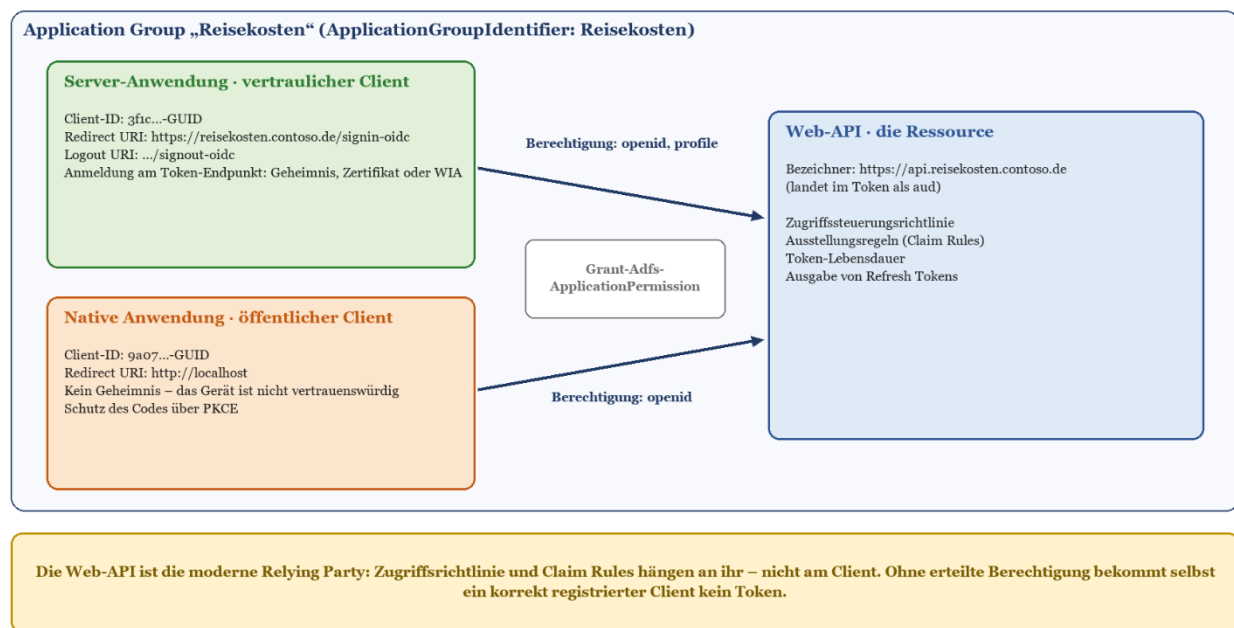
Application Groups: der Container für moderne Anwendungen

Wer ADFS aus der SAML- und WS-Federation-Welt kennt, denkt in Relying Party Trusts: eine Anwendung, ein Trust, ein Satz Claim Rules. OAuth und OpenID Connect passen in dieses Bild nur halb. Hier gibt es nicht nur die Anwendung, die ein Token konsumiert, sondern auch den Client, der es abholt – und beide können verschiedene Programme auf verschiedenen Maschinen sein. Die Web-App holt das Token, die REST-API prüft es. Oder die Mobile App holt es, und drei APIs im Rechenzentrum nehmen es entgegen.

Genau dafür hat Microsoft mit ADFS 2016 die Application Groups eingeführt. Eine Application Group ist zunächst nur ein Container mit Namen und Bezeichner. Darin legst du Rollen an: Clients, die Tokens anfordern, und Web-APIs, die Tokens entgegennehmen. Was ein Client bei einer API darf, regelst du über Berechtigungen mit Scopes. Und weil Microsoft Konsistenz liebt, wenn sie gerade niemand erwartet, heißt die Web-API in der Dokumentation an einer Stelle ausdrücklich „die neue Darstellung der Relying Party“.

Skizze 1: Was in einer Application Group steckt

Ein Container, zwei Sorten Bewohner: Clients holen Tokens, Web-APIs nehmen sie entgegen



Skizze 1: Clients holen Tokens, die Web-API nimmt sie entgegen – verbunden über Berechtigungen mit Scopes.

Die Vorlagen im Assistenten

Die ADFS-Verwaltung bietet beim Anlegen einer Application Group Vorlagen an. Sie sind nichts anderes als vorgefertigte Kombinationen der drei Rollen. Die Bezeichnungen in der deutschen Konsole weichen je nach Sprachpaket leicht ab, das Prinzip nicht.

Vorlage (sinngemäß)	Was angelegt wird	Typischer Einsatz
Native Anwendung mit Zugriff auf eine Web-API	Native Anwendung + Web-API + Berechtigung	Desktop- oder Mobile App, die ein eigenes Backend aufruft
Server-Anwendung mit Zugriff auf eine Web-API	Server-Anwendung + Web-API + Berechtigung	Webanwendung mit Backend, die Benutzer anmeldet und eine API aufruft
Webbrowser mit Zugriff auf eine Webanwendung	Client und Web-API, die eng zusammengehören	Webanwendung, die nur Benutzer anmelden will und keine weitere API braucht

Eigenständige native Anwendung	nur die Client-Rolle	Client, der Zugriff auf eine bestehende Web-API bekommt
Eigenständige Server-Anwendung	nur die Client-Rolle	Dienst oder Batch-Job, der eine bestehende API aufruft
Eigenständige Web-API	nur die Ressource	API, auf die mehrere Clients aus anderen Gruppen zugreifen

Tabelle 1: Vorlagen für Application Groups – alles Kombinationen derselben drei Bausteine

WICHTIG – Web-APIs stehen nicht in der Liste der Relying Party Trusts

Web-APIs aus Application Groups verwaltest du unter „Anwendungsgruppen“ und mit eigenen Cmdlets wie `Get-AdfsWebApiApplication` – nicht in der Liste der Vertrauensstellungen der vertrauenden Seite. Wer vor einer Migration nur die Relying Party Trusts zählt, übersieht die modernen Anwendungen komplett. Wie eine vollständige Bestandsaufnahme aussieht, steht in [Relying Parties inventarisieren – die Bestandsaufnahme vor der Entra-Migration](#).

```
# Alle Application Groups mit ihren Bausteinen auflisten
Get-AdfsApplicationGroup | Select-Object Name, ApplicationGroupIdentifier, Enabled
Get-AdfsNativeClientApplication | Select-Object Name, Identifier, RedirectUri
Get-AdfsServerApplication | Select-Object Name, Identifier, RedirectUri, ADUserPrincipalName
Get-AdfsWebApiApplication | Select-Object Name, Identifier, AccessControlPolicyName,
TokenLifetime
Get-AdfsApplicationPermission | Select-Object ClientRoleIdentifier, ServerRoleIdentifier,
ScopeNames
```

Listing 1: Bestandsaufnahme – was auf deiner Farm an OAuth- und OpenID-Connect-Anwendungen existiert

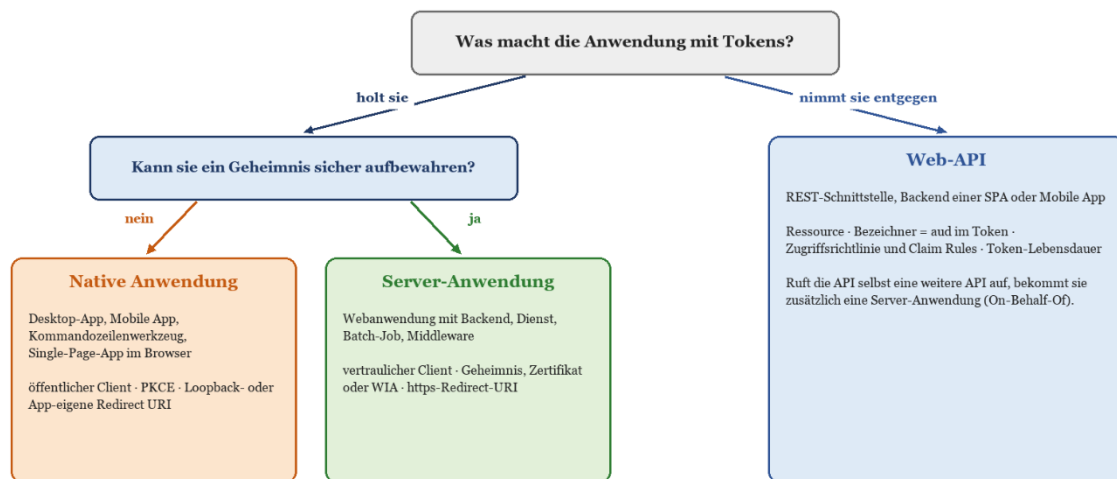
Native Anwendung, Server-Anwendung oder Web-API – wer ist hier wer?

Die häufigste Fehlentscheidung passiert in der ersten Minute: Jemand wählt den Anwendungstyp nach dem Bauchgefühl, und danach passt nichts mehr zusammen. Die Single-Page-App bekommt ein Geheimnis, das dann im JavaScript-Bundle für jeden Besucher lesbar ist. Oder die Webanwendung wird als native Anwendung registriert, weil „die Entwicklerin hat gesagt, sie braucht kein Secret“. Beides funktioniert technisch – in etwa so, wie ein Haustürschlüssel unter der Fußmatte technisch funktioniert.

Die Unterscheidung folgt der OAuth-Spezifikation: öffentliche Clients können kein Geheimnis sicher aufbewahren, vertrauliche Clients können es. ADFS nennt die einen native Anwendungen und die anderen Server-Anwendungen. Die Web-API ist kein Client, sondern die Ressource, für die das Access Token ausgestellt wird.

Skizze 3: Welcher Anwendungstyp ist der richtige?

Die Frage ist nicht, was die App kann – sondern wo ihr Code läuft und ob sie ein Geheimnis hüten kann



Faustregel: Alles, was du einem Benutzer auf den Rechner, das Handy oder in den Browser lieferst, hat kein Geheimnis. Es hat höchstens eines, das alle kennen.

Skizze 3: Die Entscheidung hängt daran, wo der Code läuft – nicht daran, wie modern die App aussieht.

Merkmal	Native Anwendung	Server-Anwendung	Web-API
OAuth-Rolle	öffentlicher Client	vertraulicher Client	Ressource
Geheimnis	keines	Geheimnis, Zertifikat (private_key_jwt) oder WIA	nicht relevant
Redirect URI	Pflicht, oft Loopback oder App-Schema	Pflicht bei Benutzeranmeldung, https	keine
Schutz des Codes	PKCE	Client-Authentifizierung, PKCE zusätzlich möglich	–
Hier hängen	nur Client-Daten	nur Client-Daten	Zugriffsrichtlinie, Claim Rules, Lebensdauer
Beispiele	Desktop-App, Mobile App, SPA, CLI	ASP.NET-Webanwendung, Dienst, Batch-Job	REST-Backend, Middleware
Cmdlet	Add-AdfsNativeClientApplication	Add-AdfsServerApplication	Add-AdfsWebApiApplication

Tabelle 2: Die drei Rollen einer Application Group im Vergleich

Wie sich die Server-Anwendung am Token-Endpunkt ausweist

Eine Server-Anwendung muss sich am Token-Endpunkt authentifizieren, bevor sie den Code gegen Tokens tauschen darf. ADFS bietet dafür drei Wege. Das gemeinsame Geheimnis ist der bequemste und der am häufigsten vergessene: Es steht irgendwann in einer appsettings.json, die in einem Repository landet, das „nur intern“ ist. Die Zertifikatsvariante ist sauberer, weil nur der öffentliche Schlüssel bei ADFS liegt. Windows-integrierte Authentifizierung über ein AD-Dienstkonto ist elegant für Dienste im eigenen Netz, aber wenig portabel.

Verfahren	Parameter	Stärken	Schwächen
Gemeinsames Geheimnis	-GenerateClientSecret	schnell eingerichtet, jede Bibliothek kann es	liegt im Klartext in der Konfiguration, kein Ablaufdatum, das sich selbst meldet
Zertifikat (private_key_jwt)	-JWTSigningCertificate oder -JWKSUri	privater Schlüssel verlässt den App-Server nicht	Zertifikat muss erneuert und nachgetragen werden

Windows-integriert (WIA)	-ADUserPrincipalName	kein Geheimnis in Dateien, Kerberos im LAN	nur im eigenen AD, bei einer späteren Migration ohne direktes Gegenstück
---------------------------------	----------------------	--	--

Tabelle 3: Client-Authentifizierung der Server-Anwendung

WARNUNG — Das generierte Geheimnis siehst du genau einmal

Erzeugt ADFS das Geheimnis, zeigt der Assistent oder die Cmdlet-Ausgabe es einmal an. Danach gibt es keinen Weg, es wieder auszulesen – nur ein neues mit Set-AdfsServerApplication - ResetClientSecret. Das ist kein Mangel, sondern Absicht. Lege das Geheimnis direkt im Passwort-Tresor ab und nicht im Chatverlauf mit der Entwicklung.

Die Web-API: hier wohnen Richtlinie und Claims

Der Bezeichner der Web-API ist der wichtigste Wert der ganzen Konfiguration. Er landet im Access Token als aud, und die API prüft genau diesen Wert. Steht im Code der API „https://api.reisekosten.contoso.de/“ mit Schrägstrich und bei ADFS ohne, lehnt die API jedes Token ab – mit einer Fehlermeldung, die selten „Schrägstrich“ enthält. Zugriffsrichtlinie, Ausstellungsregeln und Token-Lebensdauer hängen ebenfalls an der Web-API. Die Regelsprache ist dieselbe wie bei SAML, die Details stehen im [ADFS Claim Rules Leitfaden](#); wer den Zugriff auf bestimmte Gruppen begrenzen will, findet das Vorgehen in [Access Control Policies in ADFS – Zugriffsregeln ohne Claim-Rule-Akrobatik](#).

```
# 1. Container anlegen
New-AdfsApplicationGroup -Name "Reisekosten" -ApplicationGroupIdentifier "Reisekosten"

# 2. Server-Anwendung (die Web-App) mit generiertem Geheimnis
$clientId = [guid]::NewGuid().ToString()
Add-AdfsServerApplication -ApplicationGroupIdentifier "Reisekosten" `
    -Name "Reisekosten - Web" -Identifier $clientId `
    -RedirectUri "https://reisekosten.contoso.de/signin-oidc" `
    -LogoutUri "https://reisekosten.contoso.de/signout-oidc" `
    -GenerateClientSecret -PassThru

# Das Geheimnis steht in der Ausgabe - sofort in den Tresor damit.

# 3. Web-API mit Claims, Zugriffsrichtlinie und kurzer Token-Lebensdauer (Minuten)
# Die Ausstellungsregeln (UPN und gefilterte Gruppen aus dem AD) liegen versioniert in einer Datei.
Add-AdfsWebApiApplication -ApplicationGroupIdentifier "Reisekosten" `
    -Name "Reisekosten - API" -Identifier "https://api.reisekosten.contoso.de" `
    -AccessControlPolicyName "Reisekosten-Benutzer" `
    -IssuanceTransformRulesFile "C:\ADFS\Regeln\reisekosten-api.txt" -TokenLifetime 30

# 4. Berechtigung: dieser Client darf mit diesen Scopes an diese API
Grant-AdfsApplicationPermission -ClientRoleIdentifier $clientId `
    -ServerRoleIdentifier "https://api.reisekosten.contoso.de" `
    -ScopeNames "openid", "profile"
```

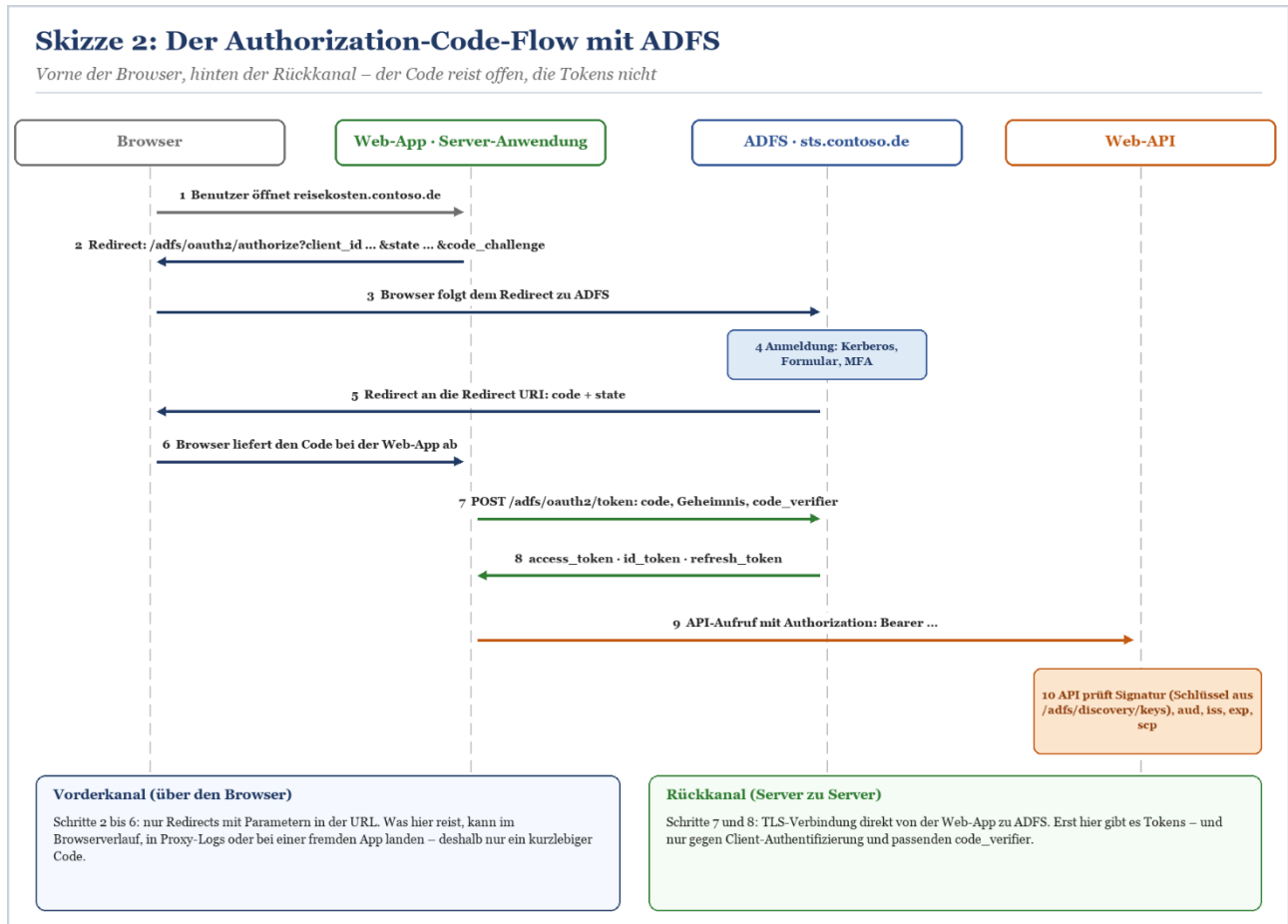
Listing 2: Eine Application Group für Web-App und API, komplett per PowerShell. Die Zugriffsrichtlinie „Reisekosten-Benutzer“ muss in der Farm bereits existieren.

TIPP — PowerShell statt Assistent – zumindest für die Dokumentation

Der Assistent ist für den ersten Versuch angenehm. Für alles, was produktiv geht, lohnt das Skript: Es ist wiederholbar, landet in der Versionsverwaltung und erklärt in einem Jahr der Kollegin aus dem Bereitschaftsdienst, warum die API genau diese Claims bekommt. Der Assistent erklärt gar nichts, er klickt nur.

Der Authorization-Code-Flow mit ADFS, Schritt für Schritt

Der Authorization-Code-Flow ist der Standardweg für alles, was einen Benutzer anmeldet – Webanwendungen wie native Apps. Sein Trick: Über den Browser reist nur ein kurzlebiger Code, die eigentlichen Tokens holt die Anwendung über einen direkten Rückkanal ab. Wer den Code abfängt, hat ohne Client-Authentifizierung oder `code_verifier` nichts davon außer einem kurzen Gefühl von Wichtigkeit.



Skizze 2: Der Authorization-Code-Flow mit ADFS – Code über den Browser, Tokens über den Rückkanal.

Im Beispiel sieht der Ablauf so aus: Die Web-App leitet den Browser an `https://sts.contoso.de/adfs/oauth2/authorize` weiter und übergibt dabei `client_id`, `redirect_uri`, `response_type=code`, die gewünschten Scopes, einen zufälligen state-Wert und bei PKCE die `code_challenge`. ADFS prüft Client-ID und Redirect URI, ermittelt die Ressource, prüft Berechtigung und Scopes, meldet den Benutzer an – im LAN per Kerberos, von außen per Formular und gegebenenfalls MFA – und schickt den Browser mit code und state an die Redirect URI zurück. Die Web-App tauscht den Code am Token-Endpoint gegen die Tokens und ruft mit dem Access Token die API auf.

Endpunkt (Beispiel sts.contoso.de)	Zweck
<code>/adfs/.well-known/openid-configuration</code>	Metadaten: alle Endpunkte, Issuer, unterstützte Verfahren – das ist die „Discovery-URL“
<code>/adfs/oauth2/authorize</code>	Start der Anmeldung, liefert den Autorisierungscode
<code>/adfs/oauth2/token</code>	tauscht Code oder Refresh Token gegen Tokens, Client-Authentifizierung hier
<code>/adfs/discovery/keys</code>	öffentliche Schlüssel, mit denen die API die Signatur prüft

/adfs/userinfo	liefert Angaben zum angemeldeten Benutzer
/adfs/oauth2/devicecode	Device-Code-Flow für Geräte ohne vernünftige Tastatur
/adfs/oauth2/logout	Abmeldung

Tabelle 4: Die OAuth- und OpenID-Connect-Endpunkte einer ADFS-Farm. Verbindlich sind immer die Werte aus deinem eigenen Discovery-Dokument.

FAKTEN — Signiert mit dem Token-Signing-Zertifikat

Die JWTs signiert ADFS mit demselben Token-Signing-Zertifikat wie die SAML-Assertions; der öffentliche Teil steht im keys-Endpunkt. APIs, die die Schlüssel regelmäßig aus den Metadaten nachladen, überstehen einen Zertifikatswechsel unbemerkt. APIs mit fest eingetragenem Zertifikat fallen am Tag nach dem Rollover um. Wie du den Wechsel planst, zeigt [ADFS Token-Signing-Zertifikat erneuern – mit und ohne AutoCertificateRollover](#).

Redirect URIs: Zeichen für Zeichen

ADFS vergleicht die redirect_uri aus der Anfrage exakt mit den registrierten Werten – inklusive Groß- und Kleinschreibung im Pfad, Port und abschließendem Schrägstrich. Die Cmdlet-Dokumentation sagt es ausdrücklich: auch der Trailing Slash muss passen. Es gibt keine Toleranz für „ungefähr richtig“, und das ist gut so: Eine zu großzügige Redirect URI ist die bequemste Methode, einem Angreifer Codes zuzustellen.

- Registriere jede Umgebung einzeln: Test, Abnahme und Produktion haben eigene Adressen, also eigene Einträge.
- Verwende https für Server-Anwendungen. Ausnahme ist die Loopback-Adresse http://localhost für native Desktop-Apps, die den Systembrowser verwenden.
- Für Windows-Store-Apps über den Web Authentication Broker sieht ADFS das Schema ms-app:// vor.
- Set-AdfsServerApplication -RedirectUri ersetzt die vorhandene Liste. Wer nur einen Eintrag ergänzen will, übergibt die alte Liste plus den neuen Wert.
- Räume auf: Redirect URIs auf Entwicklerrechner oder abgeschaltete Testserver haben in Produktion nichts verloren.

Scopes und Berechtigungen

Ein Scope ist bei ADFS zweierlei: ein Wert, den der Client in der Anfrage mitschickt, und eine Berechtigung, die du für das Paar aus Client und Web-API erteilt hast. Nur wenn beides zusammenpasst, gibt es Tokens. Schickt der Client einen Scope, den du nicht erteilt hast, bricht ADFS ab. Erteilst du einen Scope, den der Client nicht anfordert, passiert schlicht nichts – das ID Token gibt es zum Beispiel nur, wenn openid in der Anfrage steht.

Scope	Wirkung
openid	aktiviert OpenID Connect, ADFS stellt ein ID Token aus
profile	profilbezogene Claims zum angemeldeten Benutzer
email	E-Mail-Claim zum angemeldeten Benutzer
allatclaims	übernimmt die Claims des Access Tokens zusätzlich ins ID Token
user_impersonation	nötig für On-Behalf-Of, wenn eine API im Namen des Benutzers eine weitere API aufruft
aza	für Broker-Clients nach den OAuth-Erweiterungen von Microsoft, liefert ein Primary Refresh Token
logon_cert	Anmeldezertifikate für bestimmte Windows-Szenarien statt eines Access Tokens
vpn_cert	VPN-Zertifikate für EAP-TLS – laut Microsoft nicht mehr unterstützt

Tabelle 5: Die eingebauten Scopes von ADFS. Eigene Beschreibungen verwaltest du mit Get-AdfsScopeDescription und Add-AdfsScopeDescription.

WARNUNG — Ohne Ressource landest du bei urn:microsoft:userinfo

Gibt der Client weder einen resource-Parameter noch eine Ressourcen-URL im Scope mit, verwendet ADFS die Standardressource urn:microsoft:userinfo. Für die lassen sich weder MFA-Vorgaben noch Ausstellungs- oder Autorisierungsregeln konfigurieren. Das Ergebnis: Die Anmeldung klappt, aber deine sorgfältig gebaute Zugriffsrichtlinie an der Web-API greift nie. MSAL schickt die Ressource als Präfix im Scope, etwa https://api.reisekosten.contoso.de/openid – das ist richtig so und kein Tippfehler.

```
# Welche Scopes darf welcher Client bei welcher API?
Get-AdfsApplicationPermission |
  Where-Object ServerRoleIdentifier -eq "https://api.reisekosten.contoso.de" |
  Select-Object ClientRoleIdentifier, ScopeNames

# Scope ergänzen, wenn die Berechtigung schon besteht
$perm = Get-AdfsApplicationPermission |
  Where-Object { $_.ClientRoleIdentifier -eq $clientId -and
    $_.ServerRoleIdentifier -eq "https://api.reisekosten.contoso.de" }
Set-AdfsApplicationPermission -TargetIdentifier $perm.ObjectIdIdentifier -AddScope "email"
```

Listing 3: Berechtigungen prüfen und erweitern

PKCE: Pflicht für jeden öffentlichen Client

Eine native Anwendung hat kein Geheimnis. Fängt eine fremde App auf demselben Gerät den Autorisierungscode ab, könnte sie ihn ohne weiteren Schutz einfach selbst eintauschen. PKCE schließt diese Lücke: Der Client erzeugt pro Anmeldung einen zufälligen code_verifier, schickt dessen Hash als code_challenge an den authorize-Endpunkt und legt den Verifier erst beim Token-Tausch vor. ADFS unterstützt das seit Windows Server 2019, sowohl mit plain als auch mit S256. Nimm S256 – plain ist für Clients gedacht, die keinen Hash rechnen können, und solche Clients solltest du 2026 nicht mehr haben.

FAKTEN — Auf welchem Stand ist deine Farm?

PKCE, die Unterstützung für CORS-Antworten an Single-Page-Apps und weitere OAuth-Details kamen mit ADFS unter Windows Server 2019. Läuft deine Farm noch auf einem älteren Farm Behavior Level, gelten die neueren Funktionen nicht. Wie du das Level ohne Ausfall anhebst, steht in [ADFS-Farm upgraden – Farm Behavior Level anheben ohne Ausfall](#).

Wenn es klemmt: die typischen Fehlerbilder

Fehlerbild	Wahrscheinliche Ursache	Was du prüfst
ADFS zeigt sofort eine Fehlerseite, ohne Anmeldemaske	Client-ID unbekannt oder Redirect URI nicht registriert	Get-AdfsServerApplication / Get-AdfsNativeClientApplication, Redirect URI Zeichen für Zeichen
Anmeldung klappt, dann Abbruch mit Hinweis auf Scope oder Berechtigung	Scope nicht erteilt oder Client hat keine Berechtigung für die Ressource	Get-AdfsApplicationPermission, Scopes in der Anfrage
Kein ID Token in der Antwort	openid fehlt in der Anfrage	Scope-Parameter des Clients
Claims fehlen im ID Token	Claims stehen nur im Access Token	allatclaims erteilen und anfordern
API meldet „invalid audience“	aud im Token passt nicht zur Erwartung der API	Bezeichner der Web-API, resource-Parameter, Schrägstrich
API meldet ungültige Signatur	API kennt das aktuelle Token-Signing-Zertifikat nicht	Schlüssel-Nachladen der API aus den Metadaten
Token-Tausch schlägt	falsches oder neu erzeugtes	Konfiguration der App, ggf. -

mit invalid_client fehlt	Geheimnis, falsche Authentifizierungsmethode	ResetClientSecret
Zugriffsrichtlinie greift nicht	Ressource fehlt, ADFS verwendet urn:microsoft:userinfo	resource-Parameter oder Ressourcen-Präfix im Scope

Tabelle 6: Fehlerbilder bei OAuth und OpenID Connect auf ADFS

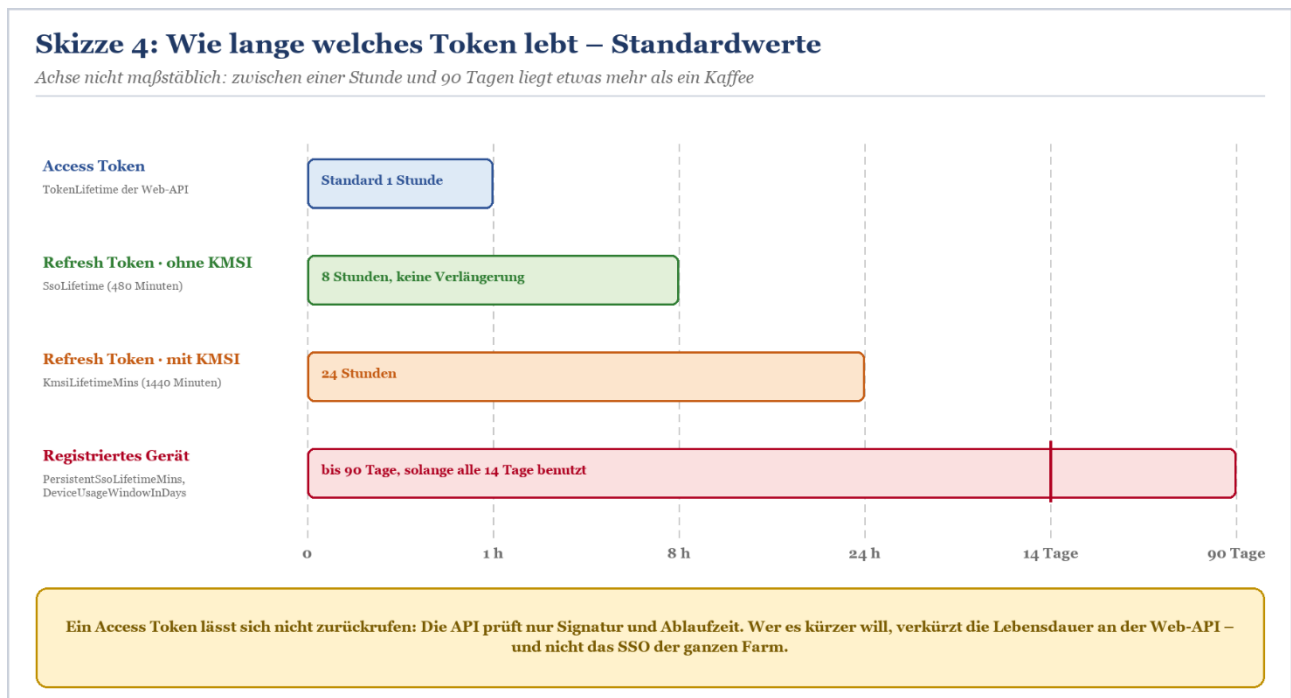
Für die Details sind das Admin-Log und bei Bedarf das Debug-Tracing auf den ADFS-Servern die erste Adresse; die Fehlerseite von ADFS nennt eine Aktivitäts-ID, mit der du den passenden Eintrag findest. Wie das geht, beschreibt [ADFS-Ereignisprotokolle lesen – Admin-Log, Debug-Tracing und Auditing](#). Viele Ursachen sind aber gar nicht OAuth-spezifisch, sondern betreffen Kerberos, Zertifikate oder den Proxy – die Sammlung dazu steht in [ADFS-Anmeldung schlägt fehl – die häufigsten Ursachen und ihre Lösung](#).

TIPP – Ein dekodiertes Token beendet jede Diskussion

Lass dir von der Entwicklung ein Access Token geben und dekodiere es lokal, etwa mit einem kleinen PowerShell-Einzeiler über Base64 oder dem Debugger der Entwicklungsumgebung. Prüfe aud, iss, exp, scp und appid. Kopiere Produktionstokens nicht in fremde Webseiten: Ein gültiges Bearer Token ist bis zum Ablauf ein Generalschlüssel für die API.

Token-Lebensdauer: wer wie lange gültig ist

Bei der Lebensdauer verwechseln die meisten zwei Dinge: das Access Token, das die API sieht, und das Refresh Token, mit dem der Client sich neue Access Tokens holt. Das eine regelst du an der Web-API, das andere hängt an den SSO-Einstellungen der gesamten Farm. Wer den Unterschied ignoriert, stellt am Ende die halbe Farm um, um eine einzige App zu ändern – und wundert sich dann über die Anrufe aus der Buchhaltung.



Skizze 4: Access Tokens leben kurz, Refresh Tokens je nach SSO-Art von Stunden bis Wochen.

Token oder Sitzung	Gesteuert über	Standard laut Microsoft
Access Token	-TokenLifetime an der Web-API (Minuten)	1 Stunde
Refresh Token, Gerät nicht	SsoLifetime (Farm)	480 Minuten, also 8 Stunden

registriert, ohne KMSI		
Refresh Token mit „Angemeldet bleiben“ (KMSI)	EnableKmsi und KmsiLifetimeMins (Farm)	KMSI aus; wenn aktiv, 1440 Minuten, also 24 Stunden
Refresh Token, registriertes Gerät	PersistentSsoLifetimeMins und DeviceUsageWindowInDays (Farm)	bis 90 Tage, solange das Gerät mindestens alle 14 Tage genutzt wird
Ausgabe von Refresh Tokens überhaupt	-IssueOAuthRefreshTokensTo an der Web-API	Werte NoDevice, WorkplaceJoinedDevices, AllDevices

Tabelle 7: Lebensdauern und ihre Stellschrauben

```
# Die Farmwerte, die über Refresh Tokens entscheiden
Get-AdfsProperties | Select-Object SsoLifetime, EnableKmsi, KmsiLifetimeMins,
    EnablePersistentSso, PersistentSsoLifetimeMins, DeviceUsageWindowInDays

# Die Werte an der Web-API
Get-AdfsWebApiApplication | Where-Object Name -eq "Reisekosten - API" |
    Select-Object Name, TokenLifetime, IssueOAuthRefreshTokensTo, RefreshTokenProtectionEnabled

# Access Token der API auf 30 Minuten verkürzen, Refresh Tokens nur für registrierte Geräte
Set-AdfsWebApiApplication -TargetIdentifier "https://api.reisekosten.contoso.de" `
    -TokenLifetime 30 -IssueOAuthRefreshTokensTo WorkplaceJoinedDevices
```

Listing 4: Lebensdauern auslesen und an einer einzelnen API anpassen

WARNUNG — Ein Access Token lässt sich nicht zurückholen

Die API prüft ein JWT lokal: Signatur, Audience, Ablaufzeit. Sperrst du ein Konto im AD, bleibt ein bereits ausgestelltes Access Token bis zu seinem Ablauf gültig. Neue Tokens über den Refresh Token gibt es nicht mehr, das laufende aber schon. Eine Stunde ist für die meisten Fachanwendungen vertretbar; für Anwendungen mit sensiblen Daten darf es weniger sein. Mehr zur Absicherung der Farm steht in [ADFS-Sicherheit: Angriffe, Härtung, Golden SAML und MFA](#).

Abmelden gehört auch dazu

OpenID Connect kennt neben der Anmeldung auch die Abmeldung. ADFS unterstützt Single Log-out für OpenID Connect: Meldet sich der Benutzer ab, ruft ADFS die registrierten Logout URIs der beteiligten Clients in einem unsichtbaren iframe auf. Damit das funktioniert, braucht jeder Client eine Logout URI, die absolut ist und kein Fragment enthält. Fehlt sie, meldet die App den Benutzer lokal ab, während seine ADFS-Sitzung munter weiterlebt – und der nächste Klick auf „Anmelden“ ihn ohne Passwortabfrage wieder hineinlässt. Am Kiosk-PC im Empfangsbereich ist das ein Erlebnis, das man nur einmal haben möchte.

Ehrliche Einordnung: Wann du das nicht mehr auf ADFS neu bauen solltest

ADFS ist nicht tot. Viele Farmen laufen seit über zehn Jahren zuverlässig, und eine Application Group, die heute stabil funktioniert, musst du nicht aus Prinzip morgen umziehen. Für neue Anwendungen sieht die Lage aber anders aus, und das sagt nicht nur ein Berater, der gern Migrationsprojekte macht: In der ADFS-Entwicklerdokumentation zu den OAuth-Flows steht bei praktisch jedem Flow der Hinweis, Microsoft empfehle dringend, nach Entra ID zu migrieren, statt auf eine neuere ADFS-Version zu aktualisieren. Wer heute eine neue moderne App auf ADFS baut, baut also bewusst gegen die Richtung des Herstellers.

Skizze 5: Neu auf ADFS bauen – oder lieber nicht?

Eine Ampel für neue OAuth- und OpenID-Connect-Anbindungen



Microsoft selbst empfiehlt in der ADFS-Entwicklerdokumentation, statt eines Upgrades auf eine neuere ADFS-Version nach Entra ID zu migrieren. Für neue Anwendungen ist das ein ziemlich deutlicher Wink mit dem Zaunpfahl.

Skizze 5: Bestehendes sauber betreiben, Neues nur mit gutem Grund – und manches gar nicht mehr.

Anforderung der neuen Anwendung	ADFS	Entra ID
Benutzeranmeldung per OpenID Connect im LAN	ja, mit Kerberos-SSO	ja
Conditional Access mit Gerätezustand und Risiko	nur über Claim Rules und Zugriffsrichtlinien, ohne Risikobewertung	ja, zentral für alle Apps
Zugriff auf Microsoft Graph und andere Cloud-APIs	nein	ja
Partner und Gäste ohne eigene Vertrauensstellung	nur mit Claims Provider Trusts	ja, über B2B-Zusammenarbeit
Self-Service für App-Registrierungen durch die Entwicklung	nein, alles über die ADFS-Administration	ja, mit Rollen und Zustimmung
Dienste ohne Geheimnis in Konfigurationsdateien	WIA im eigenen AD	Managed Identities in Azure
Betrieb, Patchen, Zertifikatswechsel	keine Aufgabe	Aufgabe von Microsoft

Tabelle 8: Moderne Anforderungen und wer sie abdeckt

Wo ADFS für neue Apps noch vertretbar ist

Es gibt sie, die Ausnahmen: Netzbereiche ohne Cloud-Anbindung, eine Fachanwendung, die ausschließlich im LAN läuft und dort per Kerberos anmeldet, oder ein Übergangsprojekt, dessen Umzug in den nächsten Monaten ohnehin kommt. Dann gilt: Begründung, Ablaufdatum und Umzugsweg gehören ins Ticket, bevor die Application Group angelegt wird – nicht in die Erinnerung des Kollegen, der in zwei Jahren das Haus verlassen hat.

Wenn schon ADFS, dann portabel bauen

- Die Anwendung liest ihre Konfiguration aus dem Discovery-Dokument und lädt Schlüssel automatisch nach. Ein Wechsel des Identity Providers ist dann eine Änderung von Authority, Client-ID und Geheimnis.
- MSAL als Client-Bibliothek, Standard-Claims statt Sonderkonstruktionen. Wer sich auf allatclaims oder exotische Claim-Typen verlässt, muss beim Umzug nacharbeiten.
- Kein Implicit Flow und kein Passwort-Flow für neue Apps, auch wenn ADFS beide anbietet. Authorization Code mit PKCE ist der Weg, der auch in Entra ID der empfohlene ist.

- Zugriffsregeln als einfache Gruppenbedingungen formulieren. Die lassen sich später sauber in Conditional Access übersetzen – wie, zeigt [Claim Rules nach Conditional Access übersetzen](#).

Zum Vergleich: In Entra ID ist dieselbe Web-App eine App-Registrierung, die du per Microsoft Graph PowerShell genauso skripten kannst wie die Application Group auf ADFS. Die früher allgegenwärtigen Module MSONline und AzureAD tauchen in alten Anleitungen noch auf; der aktuelle Weg ist Microsoft Graph PowerShell.

```
Connect-MgGraph -Scopes "Application.ReadWrite.All"

$app = New-MgApplication -DisplayName "Reisekosten" -SignInAudience "AzureADMyOrg" `
    -Web @{ RedirectUri = @"https://reisekosten.contoso.de/signin-oidc" }

New-MgServicePrincipal -AppId $app.AppId
Add-MgApplicationPassword -ApplicationId $app.Id -PasswordCredential @{ DisplayName = "Web-App"
}
```

Listing 5: Dieselbe Web-App als App-Registrierung in Entra ID

FAKTEN — Der Umzug folgt dem Inventar

Für die Migration bestehender Anwendungen gilt dieselbe Reihenfolge wie bei SAML: erst inventarisieren, dann Claims und Zugriffsregeln übersetzen, dann Anwendung für Anwendung umstellen. Die SAML-Seite beschreibt [SAML-Anwendungen von ADFS nach Entra ID migrieren](#); viele Schritte gelten für OpenID-Connect-Apps sinngemäß. Den Überblick zur Zielplattform findest du auf der Seite [Microsoft Entra ID](#).

Fazit: Drei Rollen, ein paar Zeichenketten und eine klare Richtung

OAuth 2.0 und OpenID Connect auf ADFS sind kein Hexenwerk, sobald das Modell sitzt: Eine Application Group enthält Clients und Web-APIs, Clients sind öffentlich oder vertraulich, Richtlinien und Claims hängen an der Web-API, und Berechtigungen mit Scopes verbinden das eine mit dem anderen. Die meisten Fehler entstehen nicht im Protokoll, sondern in Zeichenketten – einem Schrägstrich in der Redirect URI, einem fehlenden openid, einer Ressource, die der Client nie mitgeschickt hat.

Was bereits läuft, betreibst du sauber weiter: Inventar pflegen, Geheimnisse rotieren, Lebensdauern bewusst setzen, Logout URIs eintragen. Neue moderne Anwendungen baust du dagegen nur noch mit gutem Grund auf ADFS – und dann so portabel, dass der spätere Umzug nach Entra ID ein Konfigurationswechsel ist und kein Projekt. Wenn du deine bestehenden Application Groups prüfen oder den Umzug planen willst, unterstützen wir dich im [Consulting zu ADFS \(Active Directory Federation Services\)](#). Wer den Flow lieber einmal im Labor zerlegt, ist in den [ADFS-Schulungen](#) richtig. Und die Arbeit mit Application Groups, Claims und Tokens vertieft das Buch – mehr dazu unter [ADFS in der Praxis – das Buch im Detail](#).

WEITER — Passend zum Thema

- › [ADFS Claim Rules Leitfaden](#) – die Regelsprache hinter den Claims im Access Token
- › [Access Control Policies in ADFS – Zugriffsregeln ohne Claim-Rule-Akrobatik](#) – Zugriff auf die Web-API sauber einschränken
- › [SAML-Anwendung an ADFS anbinden – Praxisbeispiel SaaS](#) – wenn die Anwendung doch lieber SAML spricht
- › [ADFS Token-Signing-Zertifikat erneuern – mit und ohne AutoCertificateRollover](#) – damit APIs den Zertifikatswechsel überstehen
- › [Relying Parties inventarisieren – die Bestandsaufnahme vor der Entra-Migration](#) – Application

Groups gehören ins Inventar

› [Claim Rules nach Conditional Access übersetzen](#) – Zugriffsregeln für die Zeit nach ADFS

FAQ: OpenID Connect und OAuth mit ADFS Application Groups

Ab welcher Version unterstützt ADFS OpenID Connect?

Mit ADFS unter Windows Server 2016, zusammen mit den Application Groups. PKCE kam mit Windows Server 2019 dazu.

Was ist eine ADFS Application Group?

Ein Container, in dem du Clients – native Anwendungen und Server-Anwendungen – sowie Web-APIs für OAuth 2.0 und OpenID Connect registrierst und über Berechtigungen mit Scopes verbindest.

Native Anwendung oder Server-Anwendung – was nehme ich?

Kann die App ein Geheimnis sicher auf einem Server aufbewahren, ist sie eine Server-Anwendung. Läuft der Code auf dem Gerät oder im Browser des Benutzers, ist sie eine native Anwendung und nutzt PKCE.

Wo finde ich die Discovery-URL meiner ADFS-Farm?

Unter /adfs/.well-known/openid-configuration auf dem Namen deines Federation Service, im Beispiel auf sts.contoso.de. Dort stehen alle Endpunkte und der Issuer.

Warum lehnt ADFS meine Redirect URI ab?

Weil sie nicht exakt einem registrierten Wert entspricht. ADFS vergleicht Zeichen für Zeichen, auch Port und abschließenden Schrägstrich.

Warum enthält mein ID Token nicht die Claims aus meinen Regeln?

Die Ausstellungsregeln der Web-API wirken auf das Access Token. Damit sie auch im ID Token landen, erteilst du dem Client den Scope allatclaims und forderst ihn in der Anfrage an.

Wie lange ist ein ADFS Access Token gültig?

Standardmäßig eine Stunde. Du änderst das pro Web-API mit Set-AdfsWebApiApplication - TokenLifetime in Minuten.

Wie verlängere ich die Gültigkeit von Refresh Tokens?

Über die SSO-Einstellungen der Farm: SsoLifetime, KMSI mit KmsiLifetimeMins oder Persistent SSO für registrierte Geräte. Diese Werte wirken auf alle Anwendungen, also mit Bedacht ändern.

Wie erzeuge ich ein neues Client Secret?

Mit Set-AdfsServerApplication und dem Schalter -ResetClientSecret. Das alte Geheimnis ist danach ungültig, die Anwendung braucht sofort den neuen Wert.

Sollte ich neue OpenID-Connect-Apps noch auf ADFS anbinden?

Nur mit gutem Grund, etwa in Netzen ohne Cloud-Anbindung. Microsoft empfiehlt in der ADFS-Dokumentation selbst, nach Entra ID zu migrieren. Bestehende Application Groups kannst du dagegen sauber weiterbetreiben.