OAuth Client Management

Creating OAuth Clients

Step-by-Step: Create an OAuth Client

Access Path:

Admin → Integrations → OAuth → Add client

OR

Menu → IT and Integrations → OAuth → Add client

Procedure:

1. Click "Add Client" Button
   └─ "Add New Client" page appears

2. Enter Application Name (Required)
   ├─ Your application's display name
   ├─ Example: "Salesforce Integration Service"
   ├─ Visible in admin dashboard
   └─ Visible in authorization screens

3. Enter Description (Optional)
   ├─ What the application does
   ├─ Example: "Syncs Salesforce contacts to Genesys every 30 minutes"
   ├─ Helpful for documentation
   └─ Visible in admin UI

4. Select Grant Type(s)
   ├─ Choose one or more:
   │ ├─ Client Credentials (service-to-service)
   │ ├─ Code Authorization / PKCE (web/mobile apps)
   │ ├─ Token Implicit Grant (DEPRECATED - don't use)

```
   │  └─ SAML2 Bearer (enterprise SSO)
   │
   ├─ Can select multiple grant types
   ├─ Each grant type has specific settings
   └─ Note: Cannot select Implicit after March 2026

5. Click "Next" Button
   └─ Proceed to grant-specific configuration

6. Grant-Specific Configuration:

   For Client Credentials:
   ├─ Assign Roles (Required)
   │  ├─ Select minimum roles needed
   │  ├─ Available roles listed
   │  ├─ Application will have these permissions
   │  └─ Note: Must have roles in your profile to assign
   │
   ├─ Assign Divisions
   │  ├─ Required for role assignment
   │  ├─ Roles scoped to divisions
   │  ├─ Default: Home Division
   │  └─ Update to appropriate divisions
   │
   └─ Token Duration
      ├─ Configurable: 300-172,800 seconds
      ├─ Default: 3600 seconds (1 hour)
      ├─ SCIM special: up to 450 days
      └─ Your choice based on use case

   For Authorization Code / PKCE:
   ├─ Token Duration
   │  ├─ Configurable: 300-172,800 seconds
   │  ├─ Default: 3600 seconds (1 hour)
   │  └─ Recommended: 18 hours (64,800 seconds)
   │
   ├─ Authorized Redirect URIs (Required)
   │  ├─ Up to 125 URIs
   │  ├─ One per line
   │  ├─ Must use HTTPS
```

```
   │  ├─ Example: https://yourapp.com/callback
   │  ├─ Example: https://yourapps.com/auth
   │  ├─ Example: http://localhost:3000/callback (dev)
   │  ├─ Loopback: http://localhost/ (any port)
   │  └─ MUST be exact match (case-sensitive)
   │
   └─ Scopes (Required)
      ├─ Click "Scope" button
      ├─ Select minimum scopes needed
      ├─ Space-separated in requests
      ├─ Example: "conversations:readonly users:readonly"
      └─ Recommended: Least privilege principle

7. Click "Next" Button
   └─ Review configuration

8. Review & Save
   ├─ Verify all settings
   ├─ Check OAuth name
   ├─ Confirm grant type(s)
   ├─ Verify scopes correct
   └─ Click "Save"

9. Client Created Successfully
   ├─ System generates Client ID
   ├─ System generates Client Secret
   ├─ IMPORTANT: Copy secret immediately!
   └─ Secret cannot be retrieved later

10. Display Confirmation
   ├─ Client Name: Your application name
   ├─ Client ID: Unique identifier (public)
   ├─ Grant Types: Selected methods
   ├─ Scopes: Requested permissions
   ├─ Created By: Your username
   ├─ Created Date: Timestamp
   └─ Client Secret: (hidden after creation)

11. Finish
   └─ Client ready to use
```

Typical Client Example:

Name: Salesforce Contact Sync
Grant Type: Client Credentials
Roles: External Contact Manager, Agent
Divisions: Home Division
Token Duration: 86400 (1 day)
Scopes: externalcontacts:manage
Created: 2026-03-13

Client Secret Management (Critical - March 2026 Change)

⚠ IMPORTANT SECURITY UPDATE (March 2026)

Previous Behavior:
├─ Client secret visible in admin UI anytime
├─ Could view secret for existing clients
├─ Increased exposure risk
└─ Security concern

Current Behavior (March 2026):
├─ Client secret only shown at creation
├─ Cannot view secret after creation
├─ Must copy immediately when created
├─ Cannot retrieve via API
└─ Enhanced security

Action Required at Client Creation:

1. When Client Created:
   ├─ Page displays client secret
   ├─ Only time you see it
   └─ Immediately copy and store

2. Copy Secret:
   ├─ Click "Copy" button
   ├─ OR manually select and copy
   └─ Keep safe!

3. Store Secret Securely:
   ├─ Secure vault (recommended)
   ├─ Environment variables
   ├─ NOT in code
   ├─ NOT in git repository
   ├─ NOT in logs
   └─ Encrypted storage

4. Acknowledge:
   ├─ Checkbox: "I have copied and stored the secret"
   ├─ Verify before checking
   └─ Only way to proceed

5. If You Lose It:
   ├─ Cannot retrieve from UI
   ├─ Click "Generate new secret"
   ├─ New secret replaces old
   ├─ Old secret no longer works
   └─ Update application immediately

Secure Storage Solutions:

HashiCorp Vault:
├─ Enterprise secret management
├─ Encryption, rotation, audit
├─ Recommended: Production environments
└─ Access via API

AWS Secrets Manager:
├─ AWS-native secret storage
├─ Encryption, rotation, audit
├─ Recommended: AWS environments
└─ IAM-based access control

Azure Key Vault:

├─ Azure-native solution
├─ Encryption, versioning
├─ Recommended: Azure environments
└─ RBAC access control

Environment Variables (Dev Only):
├─ .env file (development)
├─ Never commit to git
├─ NOT for production
├─ Simple for local development
└─ Use .gitignore

Docker Secrets:
├─ Container orchestration
├─ Swarm/Kubernetes
├─ Encrypted storage
└─ Recommended: Container deployments

Never Store:
- In application code
- In git repository
- In version control
- In logs
- In configuration files
- In plain text
- In comments
- In documentation

OAuth Client Security

Security Best Practices:

Secret Rotation:

Timeline:
├─ Rotate monthly minimum
├─ Before employee departures

├─ After any exposure
├─ If suspected compromise
└─ Automated via CI/CD

Process:

1. Generate New Secret:
   ├─ Click "Generate new secret"
   ├─ Confirm action
   └─ New secret displayed once

2. Update Application:
   ├─ Update configuration
   ├─ Dual-use period: old + new both work
   ├─ Monitor for errors
   └─ Verify new secret works

3. Verify Working:
   ├─ Test authentication
   ├─ Check logs for success
   ├─ No 401 errors
   └─ All requests working

4. Remove Old Secret:
   ├─ Old automatically stops working
   ├─ After brief dual-use period
   ├─ No action needed
   └─ Can't revert to old

Audit Logging:

Log These Events:
├─ OAuth client created
├─ Secret regenerated
├─ Scopes added/removed
├─ Roles assigned/removed
├─ Client deleted
├─ Access failures
└─ Unusual activity

What to Log:
├─ Timestamp
├─ Admin user
├─ Action taken
├─ Client affected
├─ Result (success/failure)
└─ NOT the secret itself

Monitoring:

Monitor For:
├─ Unexpected client creation
├─ Secret rotation outside schedule
├─ Failed authentication attempts
├─ Unusual scope usage
├─ Access pattern changes
├─ Clients not used regularly
└─ Suspicious activity

Permissions:

Who Can Create Clients?
├─ Admin users with "oauth:client:add" permission
├─ Typically: Super Admin, OAuth Admin role
└─ Verify in your organization

Who Can View Clients?
├─ Admin users
├─ Users with "oauth:client:view" permission
└─ Consider minimizing access

Who Can Delete Clients?
├─ Admin users
├─ OAuth Admin role
└─ Approval process recommended

Compliance Requirements:

HIPAA:
├─ Requires MFA for admin access

├─ 15-minute idle timeout enforced
├─ Audit logging required
└─ Client rotation documented

PCI-DSS:
├─ Secure secret storage required
├─ No hardcoded secrets
├─ Regular rotation required
└─ Access control required

SOC 2:
├─ Audit trails for all changes
├─ Access control enforcement
├─ Change management process
└─ Incident response documented

GDPR:
├─ Data processing log required
├─ Right to access supported via API
├─ Data retention policies
└─ Deletion/export capabilities

OAuth Client Lifecycle

Stages:

1. Creation
   ├─ Admin creates client
   ├─ Scopes assigned
   ├─ Roles assigned (if Client Credentials)
   ├─ Secret generated
   └─ Client ID provided

2. Configuration
   ├─ Redirect URIs registered (if Auth Code)
   ├─ Scopes finalized
   ├─ Token duration set

└─ Roles/divisions confirmed

3. Usage
   ├─ Applications authenticate with client
   ├─ Users authorize app (if Code grant)
   ├─ Tokens issued and used
   ├─ API calls made
   └─ Regular operations

4. Monitoring
   ├─ Track usage patterns
   ├─ Monitor authentication success
   ├─ Alert on anomalies
   ├─ Quarterly scope review
   └─ Annual security audit

5. Maintenance
   ├─ Rotate secrets monthly
   ├─ Update scopes as needed
   ├─ Verify still in use
   ├─ Remove unused clients
   └─ Document changes

6. Deactivation
   ├─ Determine if still needed
   ├─ Plan removal
   ├─ Notify application owners
   ├─ Provide replacement if needed
   └─ Allow migration period

7. Deletion
   ├─ Remove old client
   ├─ Tokens immediately invalid
   ├─ Applications get 401 errors
   ├─ Audit log records deletion
   └─ Cannot be recovered

Timeline:

Creation → Configuration → Usage → Monitoring → Maintenance → Deactivation → Deletion

0 days 1 day Day 1+ Day 30+ Day 30+ Day 365+ Day 380+

Common OAuth Client Configurations

Configuration 1: Web Application (Node.js + React)

Grant Type: Authorization Code

Redirect URIs:

├─ https://yourapp.com/callback

├─ https://yourapp.com/auth

└─ http://localhost:3000/callback (dev)

Scopes:

├─ conversations:readonly

├─ users:readonly

└─ presence:manage

Token Duration: 3600 (1 hour)

Use Case:

├─ User logs in via browser

├─ Backend handles token exchange

├─ Backend stores refresh token

└─ User can revoke access anytime

Configuration 2: Service Integration (Salesforce Sync)

Grant Type: Client Credentials

Roles:

├─ External Contact Manager

└─ Scheduler (if needed)

Divisions: Home Division

Scopes:
├─ externalcontacts:manage
├─ users:readonly
└─ scheduling:readonly

Token Duration: 86400 (1 day)

Use Case:
├─ Automated sync service
├─ No user interaction
├─ Runs on schedule
└─ Application acts as itself

Configuration 3: Mobile App with Backend

Grant Type: Authorization Code + PKCE
Redirect URIs:
├─ myapp://oauth/callback
├─ https://myapp.com/callback
└─ http://localhost:3000/callback (dev)

Scopes:
├─ conversations:readonly
├─ conversations:call:control
├─ users:readonly
└─ presence:manage

Token Duration: 3600 (1 hour)

Use Case:
├─ Mobile app user login
├─ PKCE for public client security
├─ Backend stores refresh token
└─ App cannot store client_secret

Configuration 4: Single-Page Application (SPA)

Grant Type: Authorization Code + PKCE

Redirect URIs:
├─ https://yourapp.com/
├─ https://yourapp.com/callback
└─ http://localhost:3000/ (dev)

Scopes:
├─ conversations:readonly
├─ users:readonly
└─ presence:manage

Token Duration: 3600 (1 hour)

Use Case:
├─ JavaScript SPA
├─ No backend (serverless)
├─ PKCE prevents code interception
├─ Tokens stored in memory only

Configuration 5: Bot/Virtual Agent

Grant Type: Client Credentials
Roles:
├─ Agent
└─ Virtual Agent (if available)

Divisions: Home Division

Scopes:
├─ conversations:readonly
├─ conversations:external:contact:add
├─ knowledge:readonly
└─ users:readonly

Token Duration: 86400 (1 day)

Use Case:
├─ Chatbot integration
├─ No user context
├─ Reads knowledge base

└─ Fully automated

Configuration 6: Admin Tool/Dashboard

Grant Type: Authorization Code
Redirect URIs:
├─ https://admin.yourcompany.com/callback
└─ http://localhost:8080/callback (dev)

Scopes:
├─ users:manage
├─ roles:manage
├─ oauth:client:manage
└─ admin:org:manage

Token Duration: 3600 (1 hour)

Use Case:
├─ Admin portal
├─ Full management capabilities
├─ User authentication
└─ Administrative access control

Troubleshooting OAuth Clients

Problem: Client Creation Fails

Check:
├─ Permission: Do you have oauth:client:add?
├─ Roles: Do you have the roles you're assigning?
├─ Scopes: Are scope names spelled correctly?
└─ Divisions: Do divisions exist?

Solution:
├─ Request oauth:client:add permission
├─ Use roles you have assigned

├─ Check scope names in API Explorer
└─ Use existing divisions

Problem: Client Secret Lost/Forgotten

Cannot Retrieve:
├─ Old secret: Cannot view (security feature)
├─ No recovery method
├─ No admin override
└─ Design for security

Solution:
├─ Generate new secret
├─ Update application
├─ Delete old client if not needed
└─ Document in future

Steps:
1. Click "Generate new secret"
2. Copy new secret
3. Update application config
4. Test authentication
5. Verify working

Problem: 401 Unauthorized on API Calls

Possible Causes:
├─ Token expired: Get fresh token
├─ Token revoked: Authenticate again
├─ Client credentials wrong: Check spelling
├─ Client no longer exists: Recreate
└─ Secret wrong: Regenerate

Troubleshooting:
1. Check token hasn't expired
2. Try authenticating again (fresh token)
3. Verify client_id and client_secret spelling
4. Check client exists in Admin UI

5. If regenerated secret, update application
6. Monitor logs for pattern

Problem: Scopes Not Working

User Still Can't Access API:

Causes:
├─ Wrong scope name
├─ User missing permission
├─ Token doesn't have scope
├─ Both scope and permission needed
└─ Other 403 reason

Check:
1. API Explorer: What scope required?
2. Token: Does it have that scope?
3. User: Does user have role permission?
4. Both: Scope AND permission needed
5. 403 error: One or both missing

Problem: Application Not Working After Update

Something Changed:

Check:
├─ Secret regenerated? Update app config
├─ Scopes changed? Users must re-authorize
├─ Roles changed? Token might lack permission
├─ Token duration changed? Should still work
└─ Redirect URI changed? Registration needed

Fix:
├─ If secret: Update app immediately
├─ If scopes: Get new token (user re-auth)
├─ If roles: Assign correct roles
└─ If URI: Re-register in Admin UI

Best Practices: OAuth Client Management

Security:

☐ Store secrets in secure vault

☐ Rotate monthly minimum

☐ Never commit to git

☐ Use environment variables

☐ Encrypt at rest and in transit

☐ Audit all changes

☐ Monitor for anomalies

☐ Principle of least privilege

Operations:

☐ Document each client's purpose

☐ Keep contact info for app owner

☐ Review quarterly

☐ Remove unused clients

☐ Plan secret rotation schedule

☐ Test after any changes

☐ Verify working regularly

☐ Monitor token usage

Compliance:

☐ Meet HIPAA requirements

☐ Meet PCI-DSS requirements

☐ Meet SOC 2 requirements

☐ Document compliance steps

☐ Keep audit logs

☐ Review permissions quarterly

☐ Maintain change log

☐ Plan for incidents

Key Takeaways: Chapter 6

- **Admin-Only Access** - Only admins can create OAuth clients
- **Secret View-Once** - Client secret only shown at creation (March 2026 change)
- **Secure Storage Required** - Use vault, not code/git/environment
- **Monthly Rotation** - Standard practice for secret updates
- **Dual Grant Types** - Can select multiple grant types per client
- **Role & Scope** - Both required for Client Credentials
- **Redirect URIs** - Must be exact match (case-sensitive)
- **Audit Everything** - Log client creation, secret rotation, deletions

---

Interview Prep: OAuth Client Management

Question	Answer
Where create OAuth client?	Admin → Integrations → OAuth → Add client
Who can create?	Users with oauth:client:add permission
Client secret visibility?	Only shown once at creation (March 2026 change)
If secret lost?	Generate new secret, update application
Secret storage?	Secure vault (HashiCorp, AWS, Azure)
Secret rotation?	Monthly minimum, before departures, after exposure
Redirect URIs?	Up to 125, must be HTTPS, exact match required
Scopes required?	Yes, principle of least privilege
Token duration?	Default 1 hour, configurable 300-172,800 seconds
Client deletion?	Immediate, tokens invalid, can't recover

Document Version

Chapter: 6 of 8
Last Updated: March 2026
Status: Current with OAuth 2.0 standards
Scope: Client creation, management, security

Revision #1
Created 14 March 2026 19:33:01 by Cesar Gzz
Updated 14 March 2026 19:33:16 by Cesar Gzz