API Architecture

Genesys Cloud APIs & Platform Integration Documentation

Study Notes

Topic	Description
Platform API	RESTful API for all Genesys Cloud operations
OAuth 2.0	Industry-standard authentication framework
Grant Types	Authorization Code, Client Credentials, PKCE
API Scopes	Granular permission control for applications
OAuth Clients	Application credentials and configuration
API Documentation	Developer Center with SDKs and examples
API Rate Limiting	Throttling and quota management
Web Services	Real-time event subscriptions
Integration Architecture	Multi-system connectivity patterns

Navigation

Admin → Integrations → OAuth Developer Portal: developer.genesys.cloud API Docs: https://developer.genesys.cloud/apis/rest

---

Genesys Cloud Platform API Overview

The Genesys Cloud Platform API is a comprehensive RESTful API that enables developers to programmatically manage all aspects of the Genesys Cloud environment. It provides secure, scalable access to contact center configuration, operations, analytics, and workforce management data.

API Capabilities

Platform API Scope:

Organization & Administration:
├─ Users and credentials
├─ Roles and permissions
├─ Divisions and groups
├─ Organizations and settings
└─ Custom attributes

Contact Center Configuration:
├─ Queues and routing
├─ Skills and languages
├─ Wrap-up codes and activities
├─ Schedules and schedules groups
└─ Agents and teams

Interactions & Communications:
├─ Conversations (calls, chats, emails)
├─ Call recordings and transcripts
├─ Presence and status
├─ Notifications and events
└─ Messaging and channels

Workforce Management:
├─ Forecasts and schedules
├─ Time-off requests

├─ Shift trades and exchanges
├─ Adherence and performance
└─ Capacity planning

Analytics & Reporting:
├─ Conversation analytics
├─ Performance metrics
├─ Quality evaluations
├─ Speech analytics
└─ Custom reports

Knowledge & Collaboration:
├─ Knowledge articles
├─ Canned responses
├─ Assistants and bots
└─ Bot flows

External Integration:
├─ External contacts and organizations
├─ Third-party CRM sync
├─ Data tables and imports
└─ Webhooks and subscriptions

API Architecture

REST API Design:

Base URL: https://api.[region].mypurecloud.com/api/v2

HTTP Methods:
├─ GET: Retrieve resources
├─ POST: Create resources
├─ PUT: Replace entire resource
├─ PATCH: Partial updates
└─ DELETE: Remove resources

Response Format: JSON

Authentication: OAuth 2.0 Bearer Token

Versioning: /v2/ in URL path

Rate Limiting: Requests per minute (varies by grant type)

Pagination: Cursor-based or offset-based

Error Handling: HTTP status codes + error objects

OAuth 2.0 Authentication

Genesys Cloud uses OAuth 2.0, an industry-standard authorization framework that enables secure, delegated access to resources without exposing credentials.

OAuth 2.0 Concepts

OAuth 2.0 Terminology:

Resource Owner:
├─ The user who owns the data
├─ Genesys Cloud user account
└─ Grants permission to applications

Client (Application):
├─ The application requesting access
├─ Mobile app, web app, service, bot
├─ Registered as OAuth client
└─ Has Client ID and Client Secret

Authorization Server:
├─ Genesys Cloud authentication service
├─ Validates credentials
├─ Issues access tokens
└─ Manages token lifecycle

Resource Server:

├─ Genesys Cloud Platform API
├─ Protects API resources
├─ Validates access tokens
└─ Returns authorized data

Access Token:
├─ Short-lived credential (1 hour typical)
├─ Proves authorized access
├─ Sent with every API request
├─ In Authorization header
└─ Bearer token format

Refresh Token:
├─ Long-lived credential (up to 450 days)
├─ Used to obtain new access token
├─ Never expires unless revoked
└─ Not sent with API requests

Scopes:
├─ Granular permissions
├─ Limit application access
├─ Principle of least privilege
├─ Combined as space-separated list
└─ Examples: "conversations:readonly" "users:manage"

OAuth 2.0 Grant Types

1. Authorization Code Grant (Most Secure - Recommended)

Use Case: Web applications, desktop apps, mobile apps with backend

Flow:
1. User clicks "Login with Genesys"
2. Redirected to:
   https://login.mypurecloud.com/oauth/authorize
   ?client_id=YOUR_CLIENT_ID
   &response_type=code
   &redirect_uri=https://yourapp.com/callback
   &scope=conversations:readonly+users:readonly

3. User enters Genesys Cloud credentials

4. User grants permission to application

5. Genesys redirects to callback with authorization code:
   https://yourapp.com/callback?code=AUTH_CODE

6. Backend exchanges code for token:
   POST /oauth/token
   Content-Type: application/x-www-form-urlencoded
   Authorization: Basic BASE64(client_id:client_secret)

   grant_type=authorization_code
   &code=AUTH_CODE
   &redirect_uri=https://yourapp.com/callback

7. Response:
   {
    "access_token": "abc123...",
    "token_type": "bearer",
    "expires_in": 86400,
    "refresh_token": "xyz789..."
   }

8. Backend stores token securely (NOT in browser)

9. Backend uses token to make API calls:
   GET /api/v2/users/me
   Authorization: Bearer abc123...

Advantages:
├─ Credentials never shared with app
├─ Authorization code only for callback URL
├─ Backend exchange provides server-to-server security
├─ Refresh token enables long-lived access
└─ Aligns with OAuth 2.0 best practices

Requirements:
├─ Backend server

├─ Secure token storage
├─ HTTPS only
└─ Registered redirect URI

2. Client Credentials Grant (Service-to-Service)

Use Case: Service integrations, non-user applications, scheduled jobs, background tasks

Flow:
1. Application (no user) requests token:

```
POST /oauth/token
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
&client_id=YOUR_CLIENT_ID
&client_secret=YOUR_CLIENT_SECRET
&scope=conversations:readonly+users:manage
```

2. Response:

```
{
  "access_token": "abc123...",
  "token_type": "bearer",
  "expires_in": 86400
}
```

3. Application uses token immediately:

```
POST /api/v2/users
Authorization: Bearer abc123...
Content-Type: application/json

{
  "email": "newuser@company.com",
  "name": "John Doe"
}
```

Advantages:
├─ Single-step process
├─ No user involvement needed
├─ Ideal for backend services
├─ Simpler for automation

└ No refresh token needed (get new one as needed)

Limitations:
├ No user context (cannot access /me endpoints)
├ Application acts as itself, not user
├ Full permissions assigned to credentials
└ Requires secure secret storage

Common Uses:
├ Integration service (syncing Salesforce data)
├ Scheduled batch jobs (daily reports)
├ Outbound campaign system
├ Analytics aggregator
├ CRM synchronization
└ Webhook handlers

3. Authorization Code with PKCE (Enhanced Security)

Use Case: Single-Page Apps (SPAs), mobile apps, public clients

Why PKCE?
├ Public clients cannot securely store client_secret
├ Authorization code can be intercepted
├ PKCE prevents code exchange without proof
└ Recommended for all public clients

Flow:
1. Client generates random strings:
   code_verifier = random string (43-128 chars)
   code_challenge = BASE64(SHA256(code_verifier))

2. Redirect to authorization:
   https://login.mypurecloud.com/oauth/authorize
   ?client_id=YOUR_CLIENT_ID
   &response_type=code
   &redirect_uri=https://yourapp.com/callback
   &scope=conversations:readonly
   &code_challenge=HASH
   &code_challenge_method=S256

3. User authenticates and grants permission

4. Receives authorization code in callback:
   https://yourapp.com/callback?code=AUTH_CODE

5. Exchange code with PKCE proof:
   POST /oauth/token
   Content-Type: application/x-www-form-urlencoded

   grant_type=authorization_code
   &code=AUTH_CODE
   &client_id=YOUR_CLIENT_ID
   &redirect_uri=https://yourapp.com/callback
   &code_verifier=ORIGINAL_RANDOM_STRING

6. Token returned same as Authorization Code

Advantages:
├─ No client_secret needed
├─ Resistant to code interception attacks
├─ Proof of authorization ownership
├─ Modern OAuth 2.0 standard
└─ Works for public clients

Security:
├─ Only original code_verifier can exchange auth code
├─ Intercepted code cannot be used without verifier
├─ Verifier never transmitted over network
└─ Hash prevents code_verifier exposure

Important: Token Implicit Grant Deprecation
├─ Deprecated: March 2026
├─ Removed: March 2027
├─ Migration: Use Authorization Code + PKCE
└─ Action needed for existing embedded clients

4. Implicit Grant (DEPRECATED - DO NOT USE)

Status: DEPRECATED as of March 2026

⚠ Security Issues:
├─ Access token exposed in URL
├─ Browser history vulnerability
├─ No server-side security
├─ No refresh token support
└─ Vulnerable to token interception

Migration Path:
├─ Replace with Authorization Code + PKCE
├─ Deadline: May 2027 (existing clients)
└─ No new clients permitted after March 2026

DO NOT use for new applications.

OAuth Client Management

Creating an OAuth Client

Step-by-Step:

1. Navigate to Admin → Integrations → OAuth

2. Click "Add Client"

3. Configure Client:
   ├─ App Name: Your application name
   ├─ Description: What the app does
   ├─ Grant Type(s): Select appropriate type
   │ ├─ Authorization Code (with or without PKCE)
   │ └─ Client Credentials
   ├─ Authorized Redirect URIs:
   │ └─ https://yourapp.com/callback (one per line)
   ├─ Scopes: Select minimum needed
   └─ Roles: Select appropriate roles

4. System displays:

├─ Client ID (can be displayed anytime)
├─ Client Secret (ONLY shown at creation!)
├─ Creation metadata
└─ Modification history

⚠ CRITICAL: Copy Client Secret immediately!
├─ Secret not displayed again
├─ Cannot be recovered from API
├─ Store securely (never in code/git)
├─ Rotate regularly (monthly)

5. Example values:
Client ID: 1a2b3c4d-5e6f-7g8h-9i0j-1k2l3m4n5o6p
Client Secret: abc123...xyz789 (shown once!)

OAuth Client Security Best Practices

Credential Management:

Client ID:
├─ Public (can be shared)
├─ Used in browser/mobile apps
└─ Identifies application

Client Secret:
├─ CONFIDENTIAL (never share)
├─ Server-side only
├─ Store in secure vault
├─ Never commit to git
├─ Rotate regularly
└─ Regenerate if compromised

Access Token:
├─ Short-lived (1 hour)
├─ Sent with every request
├─ HTTP Authorization header only
├─ Never expose in browser
└─ Delete when done

Refresh Token:
├─ Long-lived (up to 450 days)
├─ Never sent with API requests
├─ Used to get new access token
├─ Server-side only
└─ Rotate and track

Secret Storage (Recommended):
├─ HashiCorp Vault
├─ AWS Secrets Manager
├─ Azure Key Vault
├─ Environment variables (dev only)
└─ Docker secrets (container orchestration)

Secret Rotation:
├─ Monthly minimum
├─ Before employee departures
├─ After exposure (immediately)
├─ Automated via CI/CD
└─ No application downtime needed (dual-use temporary period)

IP Whitelisting:
├─ For sensitive integrations
├─ Restrict which IPs can exchange code
├─ Requires AppxConnect IP for Salesforce
├─ Example: 177.104.46.240:443

OAuth Scopes

OAuth scopes provide granular permission control, limiting what applications can do with user data.

Scope Format

Scope Naming Convention: resource:action:scope

Examples:

```
├─ conversations:readonly       (view conversations)
├─ conversations:call:add       (make calls)
├─ users:manage                 (create/modify users)
├─ telephony:device:manage      (manage phones)
├─ routing:queue:view           (view queues)
├─ analytics:conversationDetail:view  (view call recordings)
└─ conversations:call:control   (transfer, hold, etc.)

Combining Scopes:
├─ Space-separated in requests
├─ Example: "conversations:readonly users:readonly"
├─ Authorization Code prompt shows all scopes
└─ User grants all or none

Scope Enforcement:
├─ If token lacks required scope: 403 Forbidden
├─ Each API endpoint requires specific scope
├─ Token scope limits apply, not user permissions
├─ User permissions and scopes both required (AND)
```

Common Scope Requirements

```
Conversation Management:
├─ conversations:readonly       (GET calls, emails, chats)
├─ conversations:call:add       (Place outbound calls)
├─ conversations:call:control   (Transfer, hold, disconnect)
├─ recordings:view              (Access recordings)
└─ recordings:download          (Download recording files)

User Management:
├─ users:readonly               (Read user info)
├─ users:manage                 (Create/update/delete users)
├─ authorization:grant:readonly (View user permissions)
└─ authorization:grant:manage   (Modify user permissions)

Contact Center Configuration:
├─ routing:queue:view           (View queue config)
├─ routing:queue:manage         (Modify queue config)
├─ directory:organization:view  (View org structure)
```

```
└─ telephony:phone:manage       (Manage phones)

Workforce Management:
├─ forecasting:readonly         (View forecasts)
├─ forecasting:manage           (Create/edit forecasts)
├─ scheduling:readonly          (View schedules)
├─ scheduling:manage            (Create/edit schedules)
├─ timeoff:view                 (View time-off)
└─ timeoff:manage               (Approve time-off)

Analytics & Reports:
├─ analytics:conversationDetail (View detailed analytics)
├─ analytics:aggregate:view     (View aggregated data)
├─ quality:evaluation:view      (View quality evaluations)
└─ speechanalytics:data:view    (Speech analytics)

Integrations:
├─ webhooks:manage              (Create/manage webhooks)
├─ scim:write                   (SCIM provisioning)
└─ externalcontacts:manage      (Sync external contacts)
```

API Authentication Flow

Complete Request Cycle

```
1. Authenticate (get access token)
   ├─ Authorization Code flow:
   │  ├─ User grants permission
   │  ├─ Get authorization code
   │  ├─ Exchange code + secret for token
   │  └─ Token valid for 1 hour
   │
   └─ Client Credentials flow:
      ├─ Direct credential exchange
      ├─ Token valid for 1 hour
      └─ No user involved
```

2. Use Access Token

GET /api/v2/users/me

Host: api.mypurecloud.com

Authorization: Bearer abc123xyz789

Content-Type: application/json

3. Response

HTTP/1.1 200 OK

Content-Type: application/json

{

"id": "user-123",

"name": "John Doe",

"email": "john@example.com",

"active": true

}

4. Handle Token Expiration

├─ Token expires after 1 hour

├─ API returns 401 Unauthorized

├─ Use refresh token to get new token

└─ Automatically retry request

5. Refresh Access Token

POST /oauth/token

Content-Type: application/x-www-form-urlencoded

Authorization: Basic BASE64(client_id:client_secret)

grant_type=refresh_token

&refresh_token=xyz789...

6. New Token Response

{

"access_token": "new_token_abc456...",

"token_type": "bearer",

"expires_in": 86400,

"refresh_token": "new_refresh_xyz890..."

}

7. Continue with new token
   ├─ Use new access token immediately
   ├─ Store new refresh token
   └─ Repeat cycle

Error Handling:
├─ 400 Bad Request: Invalid request format
├─ 401 Unauthorized: Invalid/expired credentials
├─ 403 Forbidden: Valid token, insufficient permissions
├─ 429 Too Many Requests: Rate limit exceeded
├─ 500 Server Error: Genesys Cloud issue
└─ 503 Service Unavailable: Maintenance mode

Token Lifecycle Management

Token Management Best Practices:

In-Memory Tokens:
├─ Store in secure memory (not localStorage)
├─ Clear when tab/window closes
├─ No persistence across sessions
└─ Suitable for SPAs with redirect flow

Backend Token Storage:
├─ Encrypt at rest
├─ Secure from SQL injection
├─ Never log token values
├─ Implement token rotation
└─ Use token expiration middleware

Refresh Token Handling:
├─ Store separately from access token
├─ Longer expiration (up to 450 days)
├─ Use to silently refresh without user action
├─ Rotate on every refresh (optional but recommended)
└─ Implement cleanup for expired tokens

Automatic Refresh:

├─ Refresh 5 minutes before expiration
├─ Detect 401 and refresh + retry
├─ Queue requests during refresh
└─ Handle race conditions (parallel requests)

Example Implementation (Node.js):

const expiresAt = Date.now() + (token.expires_in * 1000); if (expiresAt - Date.now() < 5 * 60 * 1000) { // Refresh token before expiration token = await refreshAccessToken(); }

Token Revocation:
├─ DELETE /oauth/sessions/me (revoke current token)
├─ User logout revokes all tokens
├─ No manual revocation needed normally
└─ Tokens expire automatically

API Rate Limiting & Quotas

Rate Limits (Requests Per Minute):

Grant Type:
├─ Authorization Code: 60 requests/minute per user
├─ Client Credentials: 60 requests/minute per organization
├─ SCIM: 120 requests/minute
└─ Higher limits available for enterprise customers

Example:
├─ 10 API calls @ 500ms each = uses 10 requests
├─ 60 requests/min = 1 request/second sustained
└─ 10 concurrent parallel = 10 requests counted

Handling Rate Limits:

Response Header:
X-Rate-Limit-Limit: 60
X-Rate-Limit-Remaining: 42
X-Rate-Limit-Reset: 1234567890

When Limit Exceeded:

HTTP 429 Too Many Requests

Retry-After: 60 (seconds to wait)

Implementation:

├─ Monitor X-Rate-Limit-Remaining

├─ Implement exponential backoff

├─ Queue requests if approaching limit

├─ Cache results when possible

└─ Batch operations (POST /conversations/batch)

Optimization:

├─ Use pagination (pageSize parameter)

├─ Filter results server-side (not client-side)

├─ Request only needed fields

├─ Implement caching layer

└─ Batch API calls where possible

Real-World Integration Example

Service Integration Pattern

Use Case: Sync Salesforce contacts with Genesys every hour

Architecture:

```
┌─────────────────────────────────────────────────┐
│ Node.js Service (runs hourly)                   │
├─────────────────────────────────────────────────┤
│ 1. Authenticate with Genesys (Client Credentials) │
│ 2. Fetch Salesforce contacts (Salesforce SDK)   │
│ 3. Compare with existing external contacts      │
│ 4. POST new contacts to Genesys API             │
│ 5. PATCH updates to existing contacts           │
│ 6. Log results and notify on error              │
└─────────────────────────────────────────────────┘
```

Implementation:

```
const axios = require('axios');
const salesforce = require('jsforce');

async function syncContacts() {
  try {
    // Step 1: Get Genesys access token
    const tokenResponse = await axios.post(
      'https://login.mypurecloud.com/oauth/token',
      {
        grant_type: 'client_credentials',
        client_id: process.env.GENESYS_CLIENT_ID,
        client_secret: process.env.GENESYS_CLIENT_SECRET,
        scope: 'externalcontacts:manage'
      }
    );

    const accessToken = tokenResponse.data.access_token;

    // Step 2: Get Salesforce contacts
    const conn = new salesforce.Connection({
      oauth2: { clientId, clientSecret, redirectUri }
    });

    const records = await conn.query(
      "SELECT Id, FirstName, LastName, Email, Phone FROM Contact"
    );

    // Step 3-4: Create/update in Genesys
    for (const contact of records.records) {
      const genesysContact = {
        firstName: contact.FirstName,
        lastName: contact.LastName,
        email: contact.Email,
        phone: contact.Phone,
        externalId: contact.Id // Link to Salesforce
      };
```

```
      try {
        await axios.post(
          'https://api.mypurecloud.com/api/v2/externalcontacts/contacts',
          genesysContact,
          {
            headers: {
              'Authorization': `Bearer ${accessToken}`,
              'Content-Type': 'application/json'
            }
          }
        );
      } catch (error) {
        if (error.response.status === 409) {
          // Contact exists, update instead
          await axios.patch(
            `https://api.mypurecloud.com/api/v2/externalcontacts/contacts/${contact.Id}`,
            genesysContact,
            { headers: { 'Authorization': `Bearer ${accessToken}` } }
          );
        }
      }
    }

    console.log(`Synced ${records.records.length} contacts`);
  } catch (error) {
    console.error('Sync failed:', error);
    // Send alert notification
  }
}

// Run hourly via cron job
schedule.scheduleJob('0 * * * *', syncContacts);
```

Best Practices

Security

- **Never expose client_secret** - Store in secure vault only
- **Use HTTPS** - Always for all API calls
- **Implement token rotation** - Monthly minimum
- **Validate SSL certificates** - Prevent man-in-the-middle
- **Log API calls** - For auditing and debugging (exclude tokens)
- **Scope principle of least privilege** - Grant only needed permissions

Performance

- **Implement caching** - Avoid repeated API calls
- **Use pagination** - Limit data per request
- **Batch operations** - Combine multiple updates
- **Monitor rate limits** - Respect API quotas
- **Async processing** - Handle large datasets asynchronously
- **Connection pooling** - Reuse HTTP connections

Reliability

- **Implement retry logic** - Handle temporary failures
- **Exponential backoff** - Avoid overwhelming API
- **Error handling** - Catch and log all exceptions
- **Health checks** - Monitor API availability
- **Fallback options** - Graceful degradation
- **Monitoring & alerts** - Know when things fail

Interview Cheat Sheet

Question	Answer
What's OAuth 2.0?	Industry-standard authorization framework
Grant types?	Authorization Code, Client Credentials, PKCE
When use Client Credentials?	Service-to-service, no user involved
When use Authorization Code?	Web/mobile apps with backend
What's PKCE?	Enhanced security for public clients
Scope purpose?	Granular permission control
Token lifetime?	1 hour (access), up to 450 days (refresh)
Rate limit?	60 requests/minute per credential

Question	Answer
How refresh token?	POST /oauth/token with grant_type
API base URL?	https://api.[region].mypurecloud.com/api/v2
Secret security?	Never expose, store in vault only
401 response?	Invalid/expired token, refresh required
403 response?	Valid token, missing scope/permission
429 response?	Rate limit exceeded, implement backoff
Token header?	Authorization: Bearer {access_token}

Key Takeaways

- **OAuth 2.0 Standard** - Industry-best secure authentication
- **Multiple Grant Types** - Choose based on application type
- **Scopes Control Access** - Granular permission delegation
- **Token Lifecycle** - 1-hour access, refresh tokens for persistence
- **Rate Limiting** - 60 requests/minute, monitor and respect
- **Security Critical** - Protect client secrets and access tokens
- **Error Handling** - Implement retry logic and proper error responses
- **Automation Ready** - APIs enable complete workflow automation
- **Deprecation Path** - Implicit Grant removed, migrate to PKCE
- **Enterprise Scale** - Supports millions of API calls daily

---

Additional Resources

Official Documentation

- Developer Center: https://developer.genesys.cloud/
- API Reference: https://developer.genesys.cloud/apis/rest/
- OAuth Guide: https://developer.genesys.cloud/authorization/platform-auth/
- API Explorer: https://developer.genesys.cloud/api/rest/

Support & Tools

- Genesys Community: https://community.genesys.com

- SDK Libraries: Node.js, Java, Python, Go, C#
- Postman Collection: API testing and documentation
- Technical Support: https://support.genesys.com

Document Version Info

Last Updated: March 2026
Source: Genesys Cloud Developer Documentation
Validated: Current with March 2026 releases
Version: 1.0

Revision #2
Created 14 March 2026 17:45:01 by Cesar Gzz
Updated 14 March 2026 19:35:03 by Cesar Gzz