Salesforce Updater with Google Cloud Integration

A comprehensive JavaScript solution for bulk updating Salesforce records with Google Cloud integration, featuring automatic CSV export and robust error handling.

🚀 Features

- **OAuth2 Authentication** - Secure Salesforce authentication with session management
- **Bulk Record Updates** - Process large batches of record updates efficiently
- **Rate Limit Handling** - Configurable batching and delays to respect API limits
- **CSV Export** - Automatic export of update results and summaries
- **Google Cloud Integration** - Deploy as Cloud Functions with Secret Manager support
- **Error Resilience** - Comprehensive error handling and retry logic
- **Mixed Object Support** - Update different Salesforce object types in one operation

📦 Modules

Core Modules

- `salesforce-auth.js` - Handles Salesforce OAuth2 authentication
- `salesforce-updater.js` - Manages bulk record updates and processing
- `csv-exporter.js` - Exports results to CSV format with various options
- `main-example.js` - Main orchestrator class with complete workflow

Deployment Modules

- `cloud-function.js` - Google Cloud Function implementation
- `package.json` - Dependencies and scripts for Google Cloud deployment

🛠️ Setup

Prerequisites

1. **Node.js 18+** installed on your system
2. **Google Cloud SDK** installed and authenticated
3. **Salesforce Connected App** configured with OAuth2

Installation

```bash
# Clone or download the modules
git clone <your-repo-url>
cd salesforce-updater-gcp

# Install dependencies
npm install

# Set up environment variables (see Configuration section)
cp .env.example .env
# Edit .env with your credentials
```

Salesforce Connected App Setup

1. In Salesforce Setup, go to **App Manager**
2. Click **New Connected App**
3. Fill in basic information
4. Enable OAuth Settings:
   - Selected OAuth Scopes: `Full access (full)` or specific scopes
   - Callback URL: `http://localhost:3000/oauth/callback` (or your domain)
5. Save and note the **Consumer Key** (Client ID) and **Consumer Secret**

⚙️ Configuration

Environment Variables

```bash
```

```
# Salesforce Configuration
SF_CLIENT_ID=your_salesforce_consumer_key
SF_CLIENT_SECRET=your_salesforce_consumer_secret
SF_USERNAME=your_salesforce_username
SF_PASSWORD=your_salesforce_password
SF_SECURITY_TOKEN=your_salesforce_security_token
SF_LOGIN_URL=https://login.salesforce.com  # or https://test.salesforce.com for sandbox

# Google Cloud Configuration
GOOGLE_CLOUD_PROJECT=your-gcp-project-id
STORAGE_BUCKET=your-storage-bucket-name
```

Google Cloud Secret Manager (Recommended)

Store Salesforce credentials securely in Secret Manager:

bash

```bash
# Create secret with JSON format
echo '{
  "clientId": "your_client_id",
  "clientSecret": "your_client_secret",
  "username": "your_username",
  "password": "your_password",
  "securityToken": "your_security_token",
  "loginUrl": "https://login.salesforce.com"
}' | gcloud secrets create salesforce-credentials --data-file=-
```

📖 Usage Examples

Basic Local Usage

javascript

```javascript
const SalesforceUpdateOrchestrator = require('./main-example');

async function updateAccounts() {
  // Initialize orchestrator
  const orchestrator = new SalesforceUpdateOrchestrator();

  // Configuration
  const config = {
    clientId: process.env.SF_CLIENT_ID,
    clientSecret: process.env.SF_CLIENT_SECRET,
    username: process.env.SF_USERNAME,
    password: process.env.SF_PASSWORD,
    securityToken: process.env.SF_SECURITY_TOKEN
  };

  await orchestrator.initialize(config);

  // Key-value pairs approach
  const updates = {
    '0031234567890ABC': {
      Name: 'Updated Account Name',
      Phone: '555-1234',
      BillingCity: 'San Francisco'
    },
    '0031234567890DEF': {
      Name: 'Another Updated Account',
      Phone: '555-5678',
      BillingCity: 'New York'
    }
  };

  const results = await orchestrator.executeUpdateWorkflow(
    updates,
    'Account',
    {
      batchSize: 5,
      delayMs: 100,
      exportOptions: {
        outputDir: './exports',
        baseFilename: 'account_updates'
      }
    }
  );
  console.log(`Updated ${results.updateResults.successful} records successfully`);
}

updateAccounts().catch(console.error);
```

Array Format with Mixed Objects

```javascript
// Update different object types in one batch
const mixedUpdates = [
  {
    objectType: 'Account',
    recordId: '0031234567890ABC',
    updateData: { Name: 'Updated Account' },
    metadata: { source: 'crm_migration' }
  },
  {
    objectType: 'Contact',
    recordId: '0031234567890DEF',
    updateData: { Email: 'new.email@example.com' },
    metadata: { source: 'email_update' }
  }
];

await orchestrator.executeUpdateWorkflow(mixedUpdates);
```

Using Individual Modules

```javascript
```

```
const SalesforceUpdater = require('./salesforce-updater');
const CSVExporter = require('./csv-exporter');

// Initialize updater
const updater = new SalesforceUpdater(config);
await updater.initialize();

// Process updates
const results = await updater.processKeyValueUpdates('Account', updates);

// Export to CSV
const exportResults = await CSVExporter.exportUpdateResults(results, {
  outputDir: './exports',
  separateFiles: true
});
```

Google Cloud Deployment

Deploy as Cloud Function

```bash
# Deploy HTTP-triggered function
gcloud functions deploy salesforce-updater \
  --runtime nodejs18 \
  --trigger-http \
  --allow-unauthenticated \
  --memory 512MB \
  --timeout 540s \
  --set-env-vars GOOGLE_CLOUD_PROJECT=your-project-id,STORAGE_BUCKET=your-bucket

# Deploy with authentication required
gcloud functions deploy salesforce-updater \
  --runtime nodejs18 \
  --trigger-http \
  --memory 512MB \
  --timeout 540s
```

Call Cloud Function

```bash
```

```
# Example POST request to Cloud Function
curl -X POST https://your-region-your-project.cloudfunctions.net/salesforce-updater \
  -H "Content-Type: application/json" \
  -d '{
    "updates": {
      "0031234567890ABC": {"Name": "Updated via Cloud Function"}
    },
    "objectType": "Account",
    "options": {
      "batchSize": 3,
      "bucketName": "your-exports-bucket"
    }
  }'
```

Scheduled Updates with Pub/Sub

```bash
# Create Pub/Sub topic
gcloud pubsub topics create salesforce-updates

# Deploy Pub/Sub triggered function
gcloud functions deploy scheduled-salesforce-update \
  --runtime nodejs18 \
  --trigger-topic salesforce-updates \
  --memory 512MB

# Schedule with Cloud Scheduler
gcloud scheduler jobs create pubsub daily-salesforce-sync \
  --schedule="0 9 * * *" \
  --topic=salesforce-updates \
  --message-body='{"updates": {...}, "objectType": "Account"}'
```

📊 CSV Export Features

The CSV exporter creates detailed reports of all update operations:

Export Files Generated

- **Successful Updates** (`*_successful.csv`): Records that were updated successfully
- **Failed Updates** (`*_failed.csv`): Records that failed with error details
- **All Updates** (`*_all.csv`): Combined report when `separateFiles: false`

- **Summary** (`*_summary.csv`): High-level statistics and file references

Export Options

```javascript
const exportOptions = {
  outputDir: './exports',          // Output directory
  baseFilename: 'sf_updates',      // Base filename
  separateFiles: true,             // Separate success/failure files
  includeTimestamp: true           // Include timestamp in filename
};
```

🔧 Advanced Configuration

Rate Limiting

```javascript
const options = {
  batchSize: 5,        // Records processed concurrently
  delayMs: 100,        // Delay between batches (milliseconds)
};
```

Error Handling

```javascript
// Access detailed error information
const results = await updater.processUpdates(updates);

console.log('Failed updates:', results.failedUpdates);
results.failedUpdates.forEach(failure => {
  console.log(`Record ${failure.recordId} failed:`, failure.error);
});
```

Custom Metadata

```javascript
```

```javascript
// Add tracking metadata to updates
const updates = [
  {
    objectType: 'Account',
    recordId: '001xxx',
    updateData: { Name: 'New Name' },
    metadata: {
      source: 'data_migration',
      batch: 'batch_001',
      priority: 'high'
    }
  }
];
```

🚨 Error Handling & Troubleshooting

Common Issues

1. **Authentication Failures**
   - Verify all credentials are correct
   - Check if security token is current (resets when password changes)
   - Ensure Connected App has proper OAuth scopes
2. **Rate Limiting**
   - Reduce batch size and increase delays
   - Monitor Salesforce API usage in Setup → System Overview
3. **Google Cloud Permissions**
   - Ensure Cloud Function has access to Secret Manager
   - Verify Storage bucket permissions for CSV uploads

Debug Logging

```javascript
// Enable detailed logging
process.env.NODE_ENV = 'development';

// The modules provide detailed console logging for debugging
```

📚 API Reference

SalesforceUpdater Class

Methods

- `initialize()` - Authenticate with Salesforce
- `updateRecord(objectType, recordId, updateData, metadata)` - Update single record
- `processUpdates(updateList, options)` - Process array of updates
- `processKeyValueUpdates(objectType, keyValuePairs, options)` - Process key-value updates
- `getStats()` - Get processing statistics

CSVExporter Class

Methods

- `exportUpdateResults(results, options)` - Export update results to CSV
- `exportCustomData(data, filePath, columns)` - Export custom data
- `arrayToCSV(data, columns)` - Convert array to CSV string

📄 License

MIT License - see LICENSE file for details

🤝 Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add tests for new functionality
5. Submit a pull request

📞 Support

For issues and questions:

1. Check the troubleshooting section
2. Review Salesforce API documentation
3. Check Google Cloud Function logs
4. Open an issue in the repository