ANSA Solutions

On-Premises QuickBooks Integration Agent

Technical Requirements Document - Hybrid Synchronous Architecture

Version: 3.0
Date: October 1, 2025
Project Type: Custom Development
Architecture: Hybrid Two-Component with Synchronous Passthrough

1. Executive Summary

1.1 Project Overview

Develop a two-component on-premises integration system that provides synchronous REST API access to QuickBooks Desktop. The system consists of a main server agent (can run on a separate server) and a lightweight QB shim (runs on QB computer). This hybrid architecture enables server separation while maintaining synchronous request/response with guaranteed delivery.

1.2 Key Design Principles

- **Hybrid Architecture**: Two components - Server Agent (external facing) + QB Shim (QB interface)
- **Passthrough Design**: All QBXML formatting occurs in Make.com
- **Synchronous Communication**: Direct SDK integration provides responses within 15 seconds
- **Guaranteed Delivery**: Transaction logging, retry logic, and idempotency ensure no data loss
- **Server Separation**: Main agent can run on separate server (Windows)
- **Minimal Business Logic**: Pure infrastructure - no QBXML formatting

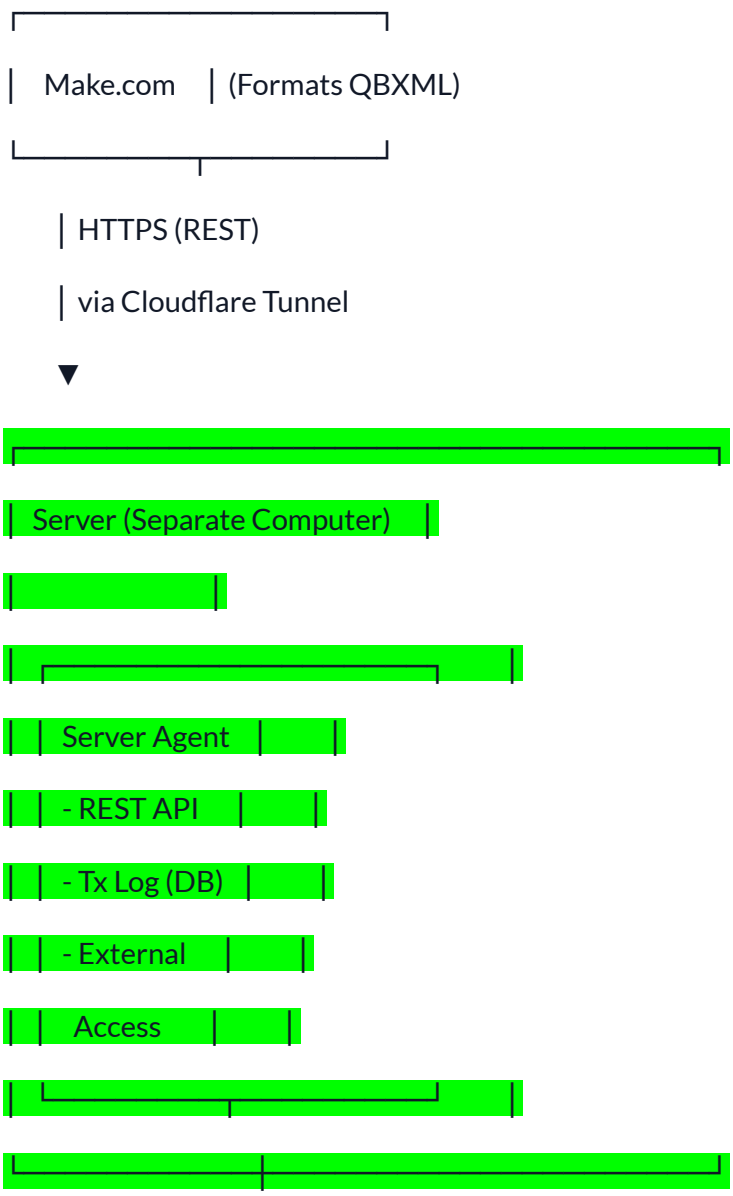
1.3 Success Criteria

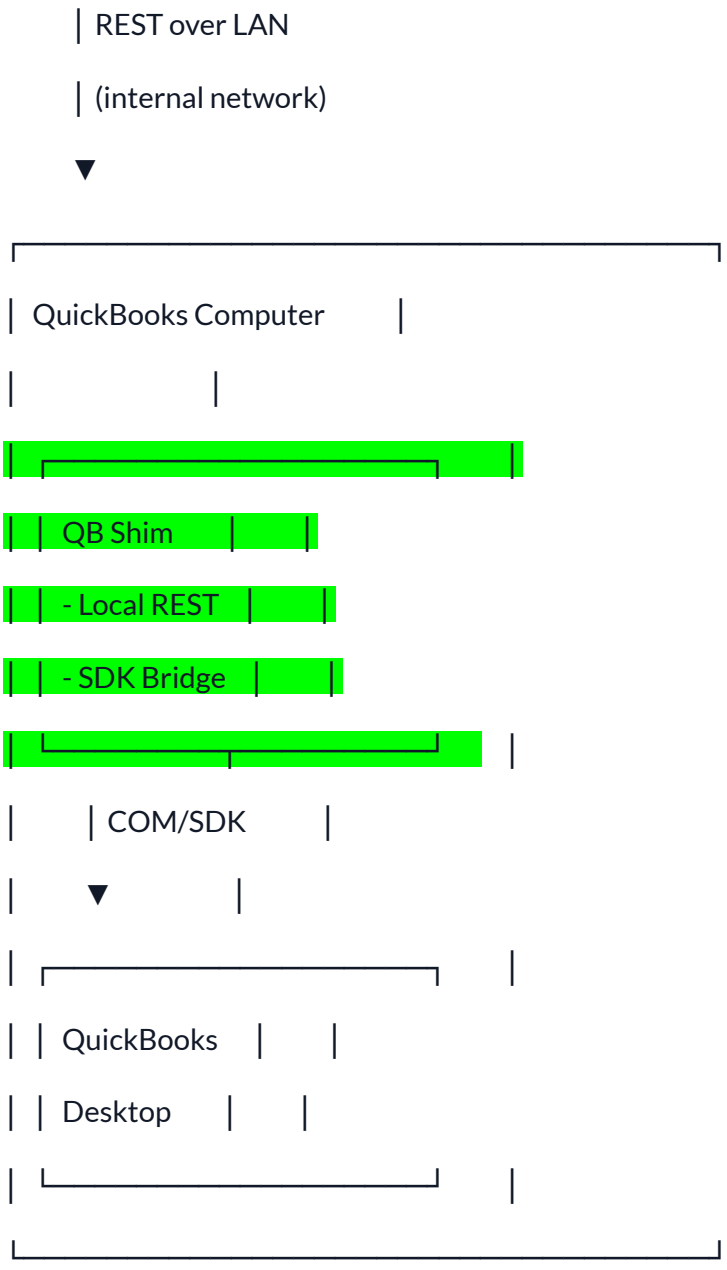
- Accept pre-formatted QBXML from Make.com via REST API
- Forward to QB computer over LAN
- Synchronously transmit to QuickBooks via SDK

- Return QuickBooks response within 15 seconds
- Guarantee delivery with transaction logging
- Support server on separate machine from QuickBooks
- 99.9% uptime during business hours

2. Technical Architecture

2.1 System Components (IN-SCOPE highlighted in green)





2.2 Component Descriptions

2.2.1 Server Agent (Main Component)

Purpose: External-facing REST API with transaction management

Responsibilities:

- Accept QBXML requests from Make.com
- Validate API Key
- Check idempotency (duplicate detection)
- Forward to QB Shim over LAN
- Wait for response from QB Shim
- Log all transactions to database
- Return responses to Make.com
- Handle retry logic

Deployment:

- Can run on separate server
- Windows
- Accessible via Cloudflare Tunnel
- SQLite database for transaction log
- Does NOT require QuickBooks

2.2.2 QB Shim (QuickBooks Interface)

Purpose: Lightweight SDK bridge on QB computer

Responsibilities:

- Accept requests from Server Agent (LAN only)
- Open QuickBooks SDK connection
- Send QBXML to QuickBooks
- Receive QuickBooks response
- Return response to Server Agent
- Minimal logging (errors only)

Deployment:

- MUST run on QB computer
- Windows only (COM/SDK requirement)
- Local REST API (port 5001)
- No database required
- No external access
- Lightweight service

2.3 Technology Stack

Server Agent:

- Python 3.9+
- Flask (REST API framework)
- Flask-CORS (Cross-origin support)
- SQLite (transaction logging)
- python-dotenv (configuration)
- Requests (for calling QB Shim)
- waitress (Windows)

QB Shim:

- Python 3.9+ (64-bit)
- Flask (minimal REST API)
- pywin32 (COM object access)
- Windows only

Network Requirements:

- Server and QB computer on same LAN
- QB computer accessible via IP or hostname
- No special firewall rules (standard LAN traffic)

2.4 Communication Protocol

Server Agent → QB Shim:

```
POST http://[qb-computer]:5001/qbxml
Content-Type: application/json
{
 "qbxml": "<QBXML>...</QBXML>",
 "transaction_id": "tx-abc123"
}
```

QB Shim → Server Agent (Response):

```
{
 "success": true,
 "qbxml_response": "<QBXML>...</QBXML>",
 "processing_time_ms": 2450
}
```

Or Error:

```
{
 "success": false,
 "error": "QuickBooks is not running",
 "error_code": "QB_UNAVAILABLE"
}
```

---

3. Functional Requirements

3.1 Server Agent REST API Endpoints

3.1.1 Process QBXML Request (Synchronous)

Endpoint: `POST /api/qbxml`

Purpose: Accept pre-formatted QBXML, forward to QB Shim, return response synchronously

Request Format Option A (JSON):

```
{
 "qbxml": "<QBXML>...</QBXML>",
 "identifier": "order-12345",
 "idempotency_key": "uuid-or-order-id"
}
```

Request Format Option B (Raw XML):

- Content-Type: `application/xml`
- Body: Raw QBXML string
- Header: `X-Request-ID: order-12345` (optional)
- Header: `X-Idempotency-Key: uuid` (optional)

Processing Flow:

1. Validate API key
2. Check idempotency (duplicate detection)
3. Generate transaction_id
4. Log transaction (status='pending')
5. Forward to QB Shim via REST (with timeout)
6. Receive response from QB Shim

7. Update transaction (status='success' or 'error')
8. Return response to Make.com

Response Success (200):

```
{
 "success": true,
 "identifier": "order-12345",
 "qb_response": "<QBXML>...</QBXML>",
 "processing_time_ms": 3250,
 "transaction_id": "tx-abc123",
 "message": "Transaction completed successfully"
}
```

Response Error (503 - QB Shim Unavailable):

```
{
 "success": false,
 "error": "QuickBooks computer not reachable",
 "error_code": "SHIM_UNAVAILABLE",
 "retry_after_seconds": 60,
 "transaction_id": "tx-abc123"
}
```

Response Error (500 - QuickBooks Error):

```
{
 "success": false,
 "error": "QuickBooks returned an error",
 "error_code": "QB_ERROR",
 "qb_error_message": "Customer 'ABC Corp' not found",
 "qb_error_code": "3100",
 "qb_response": "<QBXML>...</QBXML>",
 "transaction_id": "tx-abc123"
}
```

3.1.2 Health Check

Endpoint: `GET /health`

Checks:

- Server Agent status
- Database connectivity
- QB Shim reachability (calls QB Shim health endpoint)
- QuickBooks status (via QB Shim)

Response (200):


```
{
 "status": "healthy",
 "timestamp": "2025-10-01T12:34:56Z",
 "server_agent": {
  "status": "running",
  "database": "connected"
 },
 "qb_shim": {
  "status": "reachable",
  "url": "http://qb-computer:5001"
 },
 "quickbooks": {
  "status": "connected",
  "company_file": "C:\\QB\\Company.QBW",
  "company_file_open": true
 },
 "transactions_today": 147,
 "last_transaction": "2025-10-01T12:33:21Z"
}
```


Response (503 - Unhealthy):


```
{
 "status": "unhealthy",
 "timestamp": "2025-10-01T12:34:56Z",
 "server_agent": {
  "status": "running",
  "database": "connected"
 },
 "qb_shim": {
  "status": "unreachable",
  "url": "http://qb-computer:5001",
  "error": "Connection timeout"
 },
```

```
 "quickbooks": {
  "status": "unknown"
 }
}
```

3.1.3 Transaction History

Endpoint: `GET /api/transactions`

Query Parameters:

- `limit`: Number of records (default: 100, max: 1000)
- `offset`: Pagination offset
- `status`: Filter by status (success, error, duplicate, pending)
- `since`: ISO timestamp for date range

Response (200):

```
{
 "transactions": [
  {
   "transaction_id": "tx-abc123",
   "identifier": "order-12345",
   "idempotency_key": "uuid-123",
   "timestamp": "2025-10-01T12:33:21Z",
   "status": "success",
   "processing_time_ms": 3250,
   "qbxml_request_size": 1024,
   "qbxml_response_size": 2048
  }
 ],
 "total": 147,
 "limit": 100,
 "offset": 0
}
```

3.1.4 Get Transaction Details

Endpoint: `GET /api/transactions/{transaction_id}`

Response (200):

```
{
  "transaction_id": "tx-abc123",
  "identifier": "order-12345",
  "idempotency_key": "uuid-123",
  "timestamp": "2025-10-01T12:33:21Z",
  "status": "success",
  "processing_time_ms": 3250,
  "qbxml_request": "<QBXML>...</QBXML>",
  "qbxml_response": "<QBXML>...</QBXML>",
  "error_message": null
}
```


3.1.5 Retry Failed Transaction

Endpoint: `POST /api/transactions/{transaction_id}/retry`

Purpose: Retry a failed transaction with same QBXML

Response (200):


```
{
  "success": true,
  "new_transaction_id": "tx-def456",
  "original_transaction_id": "tx-abc123"
}
```


3.2 QB Shim REST API Endpoints

3.2.1 Process QBXML (Internal Only)

Endpoint: `POST /qbxml`

Purpose: Receive QBXML from Server Agent, send to QuickBooks, return response

Security: Only accepts connections from Server Agent IP

Request:


```
{
  "qbxml": "<QBXML>...</QBXML>",
  "transaction_id": "tx-abc123"
}
```

Processing:

1. Validate source IP (must be Server Agent)
2. Open QB SDK connection
3. Begin session
4. Process QBXML request
5. End session
6. Close connection
7. Return response

Response Success (200):

```
{
 "success": true,
 "qbxml_response": "<QBXML>...</QBXML>",
 "processing_time_ms": 2450
}
```

Response Error (503):

```
{
 "success": false,
 "error": "QuickBooks is not running or company file not open",
 "error_code": "QB_UNAVAILABLE"
}
```

Response Error (500):

```
{
 "success": false,
 "error": "QuickBooks returned an error",
 "error_code": "QB_ERROR",
 "qb_error_message": "Customer not found",
 "qb_error_code": "3100",
 "qb_response": "<QBXML>...</QBXML>"
}
```

3.2.2 Health Check (Internal Only)

Endpoint: `GET /health`

Purpose: Report QuickBooks connection status to Server Agent

Response (200):

```
{
 "status": "healthy",
 "timestamp": "2025-10-01T12:34:56Z",
 "quickbooks_connected": true,
 "company_file": "C:\\QB\\Company.QBW",
 "company_file_open": true
}
```

Response (503):

```
{
 "status": "unhealthy",
 "timestamp": "2025-10-01T12:34:56Z",
 "quickbooks_connected": false,
 "error": "Cannot connect to QuickBooks"
}
```

---

4. QuickBooks SDK Integration (QB Shim)

4.1 SDK Connection Management

Connection Lifecycle (per request):

1. OpenConnection(app_id, app_name)
2. BeginSession(company_file, mode=1) # Single-user
3. ProcessRequest(ticket, qbxml)
4. EndSession(ticket)
5. CloseConnection()

Session Management:

- Per-request connections (open/close each time)
- Maximum session duration: 30 seconds
- Auto-cleanup on timeout
- No connection pooling (unnecessary for this volume)

4.2 SDK Requirements

COM Object:

- Component: `QBXMLRP2.RequestProcessor2`
- Or: `QBXMLRP2.RequestProcessor` (fallback)

Connection Parameters:

- Application ID: Empty string `""`
- Application Name: `"QuickBooks Integration Shim"`
- Company File: From configuration
- Open Mode: `1` (single-user mode)
- **IMPORTANT**: Single-user mode locks QuickBooks during requests (2-5 seconds per request)

4.3 QuickBooks Requirements

Permissions:

- QB Shim must run as Windows user with QB access
- User must grant integration permissions (first run)
- Permissions persist after initial grant

Company File:

- Must be open during requests
- Single-user mode required
- QB Shim detects if closed and returns 503

Version Support:

- QuickBooks Desktop Enterprise
- QBXML version 13.0

---

5. Guaranteed Delivery System

5.1 Transaction Logging (Server Agent)

SQLite Database Schema:

```
CREATE TABLE transactions (
  transaction_id TEXT PRIMARY KEY,
  identifier TEXT,
  idempotency_key TEXT UNIQUE,
  timestamp DATETIME,
  status TEXT, -- 'pending', 'success', 'error', 'duplicate'
  processing_time_ms INTEGER,
  qbxml_request TEXT,
  qbxml_response TEXT,
  error_message TEXT,
  error_code TEXT,
  retry_count INTEGER DEFAULT 0,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_idempotency ON transactions(idempotency_key);
CREATE INDEX idx_timestamp ON transactions(timestamp DESC);
CREATE INDEX idx_status ON transactions(status);
```

5.2 Idempotency

Duplicate Detection:

- If `idempotency_key` provided, check database
- If exists and status='success': Return original response (409)
- If exists and status='error': Allow retry
- If exists and status='pending': Return 409 with "in progress"

Key Generation:

- Client provides idempotency_key (recommended)
- Or Server Agent generates from identifier + timestamp

5.3 Retry Logic

Automatic Retries:

- Retry on transient errors (QB Shim unreachable, QB busy)
- Max 3 automatic retries
- Exponential backoff: 1s, 2s, 4s
- Do NOT retry on validation errors or QB business logic errors

Manual Retries:

- API endpoint: `POST /api/transactions/{id}/retry`
- Uses original QBXML from transaction log
- Creates new transaction_id

5.4 Network Error Handling

QB Shim Unreachable:

- Connection timeout: 10 seconds
- Read timeout: 35 seconds (allows 30s for QB + 5s buffer)
- Retry with exponential backoff
- After max retries, return 503 to Make.com

Partial Failures:

- Request sent to QB Shim but no response
- Transaction marked 'pending'
- Operator can manually verify in QB and retry if needed

5.5 Request Guarantees

At-Least-Once Delivery:

- Transaction logged before forwarding to QB Shim
- Status updated after response
- If Server Agent crashes, status remains 'pending'
- Operator can retry pending transactions

Exactly-Once Semantics:

- Achieved via idempotency_key

- Client must provide unique key per logical operation
- Server Agent prevents duplicate processing

---

6. Non-Functional Requirements

6.1 Performance

- **REST API response time** (Server Agent validation): < 100ms
- **End-to-end response time**: < 15 seconds (including network + QB)
- **Network latency** (Server ↔ QB Shim): < 50ms on LAN
- **QuickBooks processing time**: 2-8 seconds typical
- **Concurrent requests**: Server Agent queues and processes serially
- **Throughput**: 300-500 transactions per hour

6.2 Reliability

- **Server Agent uptime**: 99.9% during business hours
- **QB Shim uptime**: Follows QuickBooks availability
- **Crash recovery**: All pending transactions recoverable from log
- **Data persistence**: SQLite database survives crashes
- **Auto-restart**: Both components restart on failure
- **Network resilience**: Automatic retry on transient network errors

6.3 Availability

Server Agent:

- Always available (even if QB is down)
- Returns 503 if QB Shim unreachable
- Health endpoint reports component status

QB Shim:

- Requires QuickBooks to be open
- Returns 503 if QB closed
- Lightweight - minimal resource usage

6.4 Security

Server Agent:

- **Authentication**: API key via `X-API-Key` header (REQUIRED)
- **Transport**: HTTPS via Cloudflare Tunnel (external)
- **Database**: Encrypted at rest (optional, SQLite encryption)
- **Secrets**: API key in .env file with restricted permissions

QB Shim:

- **Network Security**: Only accepts connections from Server Agent IP
- **No External Access**: Only accessible on LAN
- **Authentication**: IP whitelist (Server Agent only)
- **No API Key Needed**: Protected by network isolation

Communication:

- Server Agent ↔ QB Shim: HTTP over LAN (internal network)
- Can optionally use HTTPS with self-signed cert

6.5 Monitoring

Server Agent:

- Structured logging (JSON format)
- Log rotation (max 100MB per file, keep 10 files)
- Metrics: Transaction count, success rate, avg processing time
- QB Shim health checks every 60 seconds

QB Shim:

- Minimal logging (errors only)
- Windows Event Log for critical errors
- No log rotation needed (low volume)

6.6 Maintainability

- **Server Agent**: Clean code structure, comprehensive docs
- **QB Shim**: Simple design (~100 lines), minimal dependencies
- **Configuration**: Environment variables + `.env` files
- **Type Hints**: Python type annotations throughout

- **Error Messages**: Actionable with troubleshooting steps

---

7. Error Handling

7.1 Error Categories

1. Client Errors (4xx):

- `401 Unauthorized`: Missing or invalid API key
- `400 Bad Request`: Invalid QBXML syntax
- `409 Conflict`: Duplicate idempotency_key
- `413 Payload Too Large`: QBXML exceeds size limit

2. Server Agent Errors (5xx):

- `500 Internal Server Error`: Unexpected Server Agent error
- `503 Service Unavailable`: QB Shim unreachable

3. QB Shim Errors (forwarded to Server Agent):

- `503 Service Unavailable`: QuickBooks not running
- `500 Internal Server Error`: QuickBooks returned error

4. Network Errors:

- Connection timeout (10s)
- Read timeout (35s)
- DNS resolution failure
- Network unreachable

7.2 Error Response Format

```
{
 "success": false,
 "error": "Human-readable error message",
 "error_code": "MACHINE_READABLE_CODE",
 "error_details": {
  "component": "qb_shim",
  "qb_error_code": "3100",
```

```
  "qb_error_message": "Customer not found"
 },
 "transaction_id": "tx-abc123",
 "timestamp": "2025-10-01T12:34:56Z",
 "retry_allowed": true
}
```


7.3 Common Error Codes

Code	HTTP	Description	Retry
UNAUTHORIZED	401	Missing or invalid API key	No
INVALID_XML	400	Malformed QBXML	No
DUPLICATE_REQUEST	409	Idempotency key exists	No
SHIM_UNREACHABLE	503	QB Shim not responding	Yes
SHIM_TIMEOUT	504	QB Shim timeout	Yes
QB_UNAVAILABLE	503	QB not running/file closed	Yes
QB_BUSY	503	QB in use by another process	Yes
QB_ERROR	500	QB returned error	Depends
SDK_ERROR	500	SDK connection failed	Yes
NETWORK_ERROR	503	Network communication failure	Yes
INTERNAL_ERROR	500	Unexpected error	Maybe

7.4 Logging Requirements

Server Agent:

- Every REST API request (method, endpoint, status)

- Every QB Shim call (URL, status, duration)
- All errors with stack traces
- Transaction status changes
- Health check results

QB Shim:

- SDK connection events (open, close)
- QuickBooks errors only
- Critical failures to Windows Event Log

Log Format (JSON):

```
{
 "timestamp": "2025-10-01T12:34:56.789Z",
 "level": "INFO",
 "component": "server_agent",
 "message": "Request processed successfully",
 "transaction_id": "tx-abc123",
 "identifier": "order-12345",
 "processing_time_ms": 3250,
 "status": "success"
}
```

---

8. Configuration

8.1 Server Agent Environment Variables

Server Configuration

- SERVER_HOST=0.0.0.0
- SERVER_PORT=5000
- API_KEY=your_secure_api_key_here # REQUIRED

QB Shim Configuration

- QB_SHIM_URL=http://qb-computer:5001
- QB_SHIM_TIMEOUT_SECONDS=35
- QB_SHIM_CONNECT_TIMEOUT_SECONDS=10

Database

- DATABASE_PATH=./transactions.db
- DATABASE_RETENTION_DAYS=30

Performance

- MAX_QBXML_SIZE_MB=10

Logging

- LOG_LEVEL=INFO
- LOG_FILE=./logs/server_agent.log
- LOG_FORMAT=json

Retry Configuration (AUTO-RETRY ENABLED)

- AUTO_RETRY_ENABLED=true
- AUTO_RETRY_MAX_ATTEMPTS=3
- AUTO_RETRY_BACKOFF_SECONDS=1,2,4

8.2 QB Shim Environment Variables

QuickBooks Configuration

- QB_COMPANY_FILE=C:\QuickBooks\Company.QBW
- QB_OPEN_MODE=1 # 1=single-user (required)
- QB_APP_NAME=QuickBooks Integration Shim
- QBXML_VERSION=13.0

Shim Configuration

- SHIM_HOST=0.0.0.0
- SHIM_PORT=5001
- ALLOWED_IP=192.168.1.50 # Server Agent IP

Logging

- LOG_LEVEL=ERROR # Only log errors
- LOG_FILE=./logs/qb_shim.log

8.3 Network Configuration

Server Agent Location:

- Can be on separate server
- Requires network access to QB computer
- Firewall: Allow outbound to QB Shim (port 5001)

QB Shim Location:

- Must be on QB computer
- Firewall: Allow inbound from Server Agent IP (port 5001)
- No external access required

DNS/Hostname:

- QB computer must be reachable by hostname or IP
- Recommend static IP or DHCP reservation
- Update Server Agent config with QB computer address

9. Deployment Requirements

9.1 Installation Package - Server Agent

Deliverables:

1. **Application Code**

   - Python source files
   - requirements.txt with pinned versions
   - .env.example template

2. **Installation Script**

   - install_server.ps1 (Windows)
   - Creates directory structure
   - Installs dependencies
   - Initializes database
   - Configures as system service
   - Tests QB Shim connectivity

3. **System Service**

- Windows: Windows Service wrapper
- Auto-start on boot
- Auto-restart on failure

4. **Cloudflare Tunnel Setup**

- Configuration guide
- Automated setup script
- DNS configuration

Directory Structure:

```
/opt/qb_server/ (or C:\QB_Server\)
├── server_agent.py
├── requirements.txt
├── .env
├── transactions.db
└── logs/
    └── server_agent.log
```

9.2 Installation Package - QB Shim

Deliverables:

1. **Application Code**

- Python source files (qb_shim.py)
- requirements.txt with pinned versions
- .env.example template

2. **Installation Script**

- `install_qb_shim.ps1` (Windows only)
- Creates directory structure
- Installs pywin32
- Tests QB SDK connection
- Configures Windows Service

- Sets IP whitelist

3. **Windows Service**

- Service name: `QBIntegrationShim`
- Run as specific user account
- Auto-start on boot
- Auto-restart on failure

Directory Structure:

```
C:\QB_Shim\
├── qb_shim.py
├── requirements.txt
├── .env
└── logs/
    └── qb_shim.log
```