Rate-Limit Resilience & Error Handling for High-Volume Funnel Launches

Document Type: API Integration Engineering Manual
Version: 1.0
Last Updated: January 2026
Author: WealthForge Tools Infrastructure Lab
Target Audience: Integration Engineers, DevOps Teams, Technical Architects

---

Executive Summary

This engineering manual provides production-grade strategies for handling API rate limits, implementing retry logic, and maintaining system reliability during high-volume e-commerce funnel launches. The focus is on Shopify REST Admin API integration via Make.com, with specific attention to the “leaky bucket” algorithm and webhook resilience patterns.

Critical Success Factors:
- Understand Shopify’s rate limit mechanics (2 req/sec Standard, 20 req/sec Plus)
- Implement exponential backoff with jitter for retry logic
- Design Dead Letter Queue (DLQ) for failed webhook processing
- Monitor API call limits via response headers
- Establish alerting thresholds for production incidents

Target Reliability: 99.9% webhook processing success rate (max 1 failure per 1,000 orders)

---

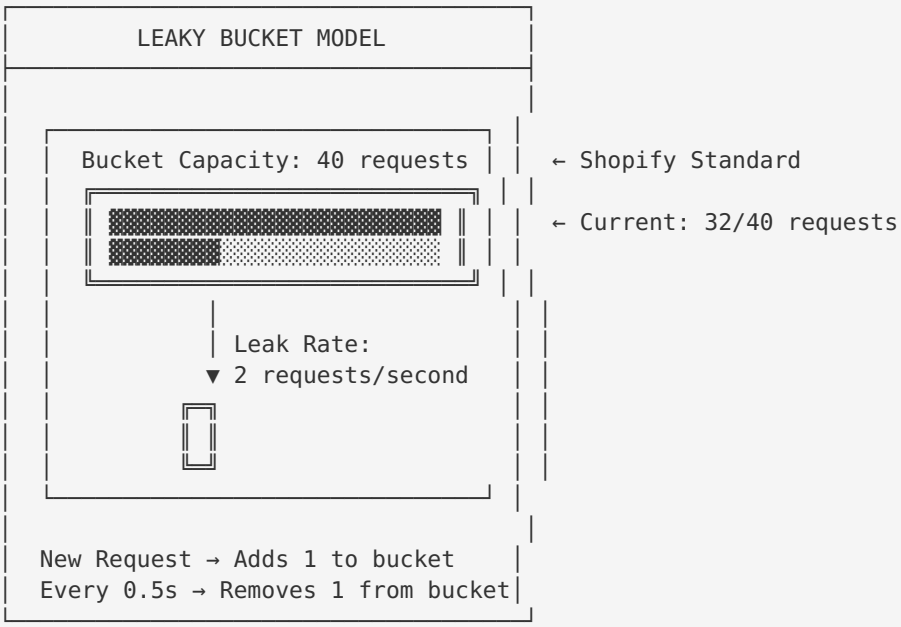
1. Shopify REST Admin API Rate Limits

1.1 Leaky Bucket Algorithm Explained

Shopify implements a **leaky bucket** algorithm to manage API request rates across all plans. This metaphor describes a system where:

1. **Bucket Capacity:** Each app has a “bucket” that can hold a specific number of request tokens
2. **Request Cost:** Each API call adds 1 token to the bucket
3. **Leak Rate:** Tokens are removed from the bucket at a constant rate (the “leak”)
4. **Throttling:** When the bucket is full, subsequent requests return HTTP 429 (Too Many Requests)

Visual Representation:



1.2 Rate Limit Specifications by Plan

Shopify Plan	Requests/ Second	Bucket Size	Leak Rate	Max Burst
Basic	2 req/sec	40 requests	2 req/sec	40 requests in 1 second
Standard	2 req/sec	40 requests	2 req/sec	40 requests in 1 second
Advanced	4 req/sec	80 requests	4 req/sec	80 requests in 1 second
Plus	20 req/sec	400 requests	20 req/sec	400 requests in 1 second
Enterprise	40 req/sec	800 requests	40 req/sec	800 requests in 1 second

Key Insights:
- **Burst Capacity:** You can send 40 requests instantly on Standard plan, but then must wait 20 seconds for bucket to fully refill
- **Sustained Rate:** For continuous operation, maintain 2 req/sec average on Standard plan
- **Bucket Refill Time:** 20 seconds (Standard), 5 seconds (Plus) to go from full to empty

1.3 Calculating Safe Request Rate for High-Volume Scenarios

Scenario: Funnel launch generating 1,000 orders in 1 hour

Requirements:
- Create Shopify order via REST API for each ClickFunnels order

- Each order requires 1 API call (POST to `/admin/api/2024-10/orders.json`)
- Target: Process all 1,000 orders within 1 hour with zero rate limit errors

Calculation for Shopify Standard Plan:

```
Available capacity per hour:
- Sustained rate: 2 requests/second
- Seconds per hour: 3,600
- Total capacity: 2 × 3,600 = 7,200 requests/hour

Required capacity:
- Orders to process: 1,000
- Requests per order: 1
- Total required: 1,000 requests/hour

Utilization:
- 1,000 / 7,200 = 13.9% of available capacity

Safe request rate:
- 1,000 orders / 3,600 seconds = 0.278 requests/second
- Recommended interval: 1 request every 3.6 seconds (with buffer)
```

Conclusion: Shopify Standard plan can easily handle 1,000 orders/hour with proper throttling.

Calculation for Shopify Plus Plan:

```
Available capacity per hour:
- Sustained rate: 20 requests/second
- Seconds per hour: 3,600
- Total capacity: 20 × 3,600 = 72,000 requests/hour

Required capacity:
- Orders to process: 1,000
- Requests per order: 1
- Total required: 1,000 requests/hour

Utilization:
- 1,000 / 72,000 = 1.4% of available capacity

Safe request rate:
- 1,000 orders / 3,600 seconds = 0.278 requests/second
- Can process instantly with burst capacity (400 requests)
```

Conclusion: Shopify Plus can process 1,000 orders in ~50 seconds using burst capacity, or maintain 20 req/sec for sustained high-volume operations.

1.4 Make.com Sleep Module Configuration

Purpose: Throttle API requests to stay within Shopify rate limits

Module: Tools → Sleep

Configuration for Shopify Standard (2 req/sec):

```
Sleep Duration: 500 milliseconds (0.5 seconds)

Placement: After each Shopify API call module

Logic:
- Request 1 → Sleep 500ms → Request 2 → Sleep 500ms → Request 3...
- Effective rate: 2 requests/second (exactly at limit)
- Buffer: None (risky for production)
```

Recommended Configuration with Safety Buffer:

```
Sleep Duration: 600 milliseconds (0.6 seconds)

Effective rate: 1.67 requests/second
Buffer: 16.5% below rate limit
Rationale: Accounts for network latency, Make.com processing time, concurrent scenari-
os
```

Configuration for Shopify Plus (20 req/sec):

```
Sleep Duration: 50 milliseconds (0.05 seconds)

Effective rate: 20 requests/second (exactly at limit)

Recommended with buffer:
Sleep Duration: 60 milliseconds (0.06 seconds)
Effective rate: 16.67 requests/second
Buffer: 16.5% below rate limit
```

Make.com Scenario Example:

```
┌─────────────────────────────────────────────────────┐
│ Module 1: Webhook (ClickFunnels Order Created)      │
├─────────────────────────────────────────────────────┤
│ Module 2: Set Variable (Extract & Sanitize Data)    │
├─────────────────────────────────────────────────────┤
│ Module 3: HTTP Request (POST to Shopify Orders API) │
├─────────────────────────────────────────────────────┤
│ Module 4: Sleep (600ms) ← RATE LIMIT PROTECTION     │
├─────────────────────────────────────────────────────┤
│ Module 5: Klaviyo Create Event                      │
├─────────────────────────────────────────────────────┤
│ Module 6: Google Sheets Add Row (Success Log)       │
└─────────────────────────────────────────────────────┘
```

⚠️ **Critical Note:** Place Sleep module **after** Shopify API call, not before. This ensures proper spacing between consecutive requests in high-volume scenarios.

1.5 Monitoring X-Shopify-Shop-Api-Call-Limit Header

Header Format:

```
X-Shopify-Shop-Api-Call-Limit: 32/40
```

Interpretation:
- **32:** Current bucket usage (requests made)
- **40:** Bucket capacity (maximum requests before throttling)
- **Remaining:** 8 requests available before hitting rate limit

Make.com Header Monitoring:

Module: Tools → Set Variable

```
// Variable Name: apiCallLimit
// Formula Mode: Enabled

{
  "raw_header": "{{3.headers.`X-Shopify-Shop-Api-Call-Limit`}}",
  "current_usage": {{parseNumber(split(3.headers.`X-Shopify-Shop-Api-Call-Limit`; "/")
[1])}},
  "bucket_capacity": {{parseNumber(split(3.headers.`X-Shopify-Shop-Api-Call-Limit`; "/
")[2])}},
  "remaining_capacity": {{parseNumber(split(3.headers.`X-Shopify-Shop-Api-Call-Lim-
it`; "/")[2]) - parseNumber(split(3.headers.`X-Shopify-Shop-Api-Call-Limit`; "/")[1])}
},
  "utilization_percent": {{round((parseNumber(split(3.headers.`X-Shopify-Shop-Api-
Call-Limit`; "/")[1]) / parseNumber(split(3.headers.`X-Shopify-Shop-Api-Call-Limit`; "
/")[2])) × 100; 2)}}
}
```

Output Example:

```
{
  "raw_header": "32/40",
  "current_usage": 32,
  "bucket_capacity": 40,
  "remaining_capacity": 8,
  "utilization_percent": 80.00
}
```

Dynamic Throttling Logic:

Module: Router (Conditional Branching)

```
Route 1: If utilization_percent > 90
  → Sleep 2000ms (2 seconds) - Emergency cooldown
  → Retry request

Route 2: If utilization_percent > 75
  → Sleep 1000ms (1 second) - Preventive slowdown
  → Continue

Route 3: If utilization_percent ≤ 75
  → Sleep 600ms (standard buffer)
  → Continue
```

Benefits:
- **Adaptive throttling:** Automatically slows down when approaching rate limit
- **Prevents 429 errors:** Proactive cooldown before hitting bucket capacity
- **Optimizes throughput:** Maintains maximum safe speed without throttling

2. Webhook Retry Logic

2.1 Three-Attempt Retry Strategy with Exponential Backoff

Retry Pattern:

```
Attempt 1: Immediate (0 seconds delay)
  ↓ FAIL
Attempt 2: Wait 2 seconds (2^1 = 2)
  ↓ FAIL
Attempt 3: Wait 4 seconds (2^2 = 4)
  ↓ FAIL
→ Send to Dead Letter Queue (DLQ)
```

Exponential Backoff Formula:

```
Delay = Base_Delay × (2 ^ Attempt_Number)

Where:
- Base_Delay = 1 second
- Attempt_Number = 1, 2, 3...

Examples:
- Attempt 1: 1 × 2^0 = 1 second (immediate)
- Attempt 2: 1 × 2^1 = 2 seconds
- Attempt 3: 1 × 2^2 = 4 seconds
- Attempt 4: 1 × 2^3 = 8 seconds
```

Why Exponential Backoff?
- **Reduces server load:** Gives API time to recover from temporary issues
- **Avoids thundering herd:** Prevents all failed requests from retrying simultaneously
- **Increases success rate:** Transient errors (network blips, temporary server issues) often resolve within seconds

2.2 Make.com Error Handler Implementation

Scenario Structure:

```
┌─────────────────────────────────────────────────────────┐
│ Main Flow                                               │
├─────────────────────────────────────────────────────────┤
│ Module 1: Webhook Trigger                               │
│ Module 2: Data Transformation                           │
│ Module 3: Shopify API Call ← ERROR HANDLER ATTACHED     │
│   ├─ Success → Continue to Module 4                     │
│   └─ Error → Trigger Error Handler                      │
└─────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────┐
│ Error Handler Flow (attached to Module 3)               │
├─────────────────────────────────────────────────────────┤
│ EH Module 1: Set Variable (Increment Retry Counter)     │
│ EH Module 2: Router (Check Retry Count)                 │
│   ├─ Route 1: If retry_count < 3                        │
│   │   ├─ Sleep (2^retry_count seconds)                  │
│   │   └─ Resume (retry Module 3)                        │
│   └─ Route 2: If retry_count ≥ 3                        │
│       └─ Send to DLQ (Google Sheets)                    │
└─────────────────────────────────────────────────────────┘
```

Step-by-Step Configuration:

Step 1: Attach Error Handler to Shopify API Module

1. Right-click on HTTP Request module (Shopify API call)
2. Select “Add error handler”
3. Choose “Ignore” directive (we’ll handle errors manually)

Step 2: Set Variable - Initialize Retry Counter

Module: Tools → Set Variable

```
// Variable Name: retryContext
// Formula Mode: Enabled

{
  "retry_count": {{if(exists(retryContext.retry_count); retryContext.retry_count + 1;
1)}},
  "original_order_id": "{{1.order.id}}",
  "error_message": "{{3.error.message}}",
  "error_code": "{{3.error.statusCode}}",
  "timestamp": "{{now}}"
}
```

Logic:
- First attempt: `retry_count = 1`
- Second attempt: `retry_count = 2`
- Third attempt: `retry_count = 3`

Step 3: Router - Conditional Retry Logic

Module: Flow Control → Router

Route 1: Retry (if attempts < 3)

Filter Condition:

```
retryContext.retry_count < 3
```

Modules in Route 1:

1. **Sleep Module**

- Duration: `{{pow(2; retryContext.retry_count) × 1000}}` milliseconds
- Example: Attempt 2 = 2^1 × 1000 = 2000ms (2 seconds)

1. **Resume Directive**
   - Action: Resume execution from Module 3 (Shopify API call)
   - Effect: Retries the failed API request

Route 2: Send to DLQ (if attempts ≥ 3)

Filter Condition:

```
retryContext.retry_count >= 3
```

Modules in Route 2:

1. **Google Sheets → Add Row** (Dead Letter Queue)
2. **Slack/Discord Webhook** (Alert notification)

2.3 Retry Logic Code Example (Pseudo-Code)

```
// Make.com Scenario Logic (Conceptual)

function processWebhook(webhookData) {
  let retryCount = 0;
  const maxRetries = 3;

  while (retryCount < maxRetries) {
    try {
      // Attempt Shopify API call
      const response = await shopifyAPI.createOrder(webhookData);

      // Success - exit retry loop
      console.log(`Order created successfully: ${response.order.id}`);
      return response;

    } catch (error) {
      retryCount++;
      console.error(`Attempt ${retryCount} failed: ${error.message}`);

      if (retryCount < maxRetries) {
        // Calculate exponential backoff delay
        const delaySeconds = Math.pow(2, retryCount);
        console.log(`Retrying in ${delaySeconds} seconds...`);

        // Wait before retry
        await sleep(delaySeconds * 1000);

      } else {
        // Max retries reached - send to DLQ
        console.error(`Max retries reached. Sending to DLQ.`);
        await sendToDLQ(webhookData, error);
        throw error; // Propagate error for alerting
      }
    }
  }
}

function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function sendToDLQ(data, error) {
  await googleSheets.addRow({
    spreadsheetId: 'DLQ_SHEET_ID',
    values: [
      data.order.id,
      data.contact.email,
      error.message,
      error.statusCode,
      new Date().toISOString(),
      JSON.stringify(data)
    ]
  });
}
```

2.4 Dead Letter Queue (DLQ) Implementation

Purpose: Store failed webhook payloads for manual review and batch recovery

Storage Solution: Google Sheets (free, accessible, easy to query)

DLQ Schema:

Column	Data Type	Example Value	Purpose
`timestamp`	ISO 8601	`2026-01-12T18:45:32Z`	When failure occurred
`cf_order_id`	String	`CF-2026-001234`	ClickFunnels order ID
`customer_email`	Email	`customer@example.com`	For customer lookup
`error_code`	Integer	`429`	HTTP status code
`error_message`	String	`Too Many Requests`	Error description
`retry_count`	Integer	`3`	Number of attempts made
`webhook_payload`	JSON	`{...}`	Full webhook data for replay
`status`	Enum	`pending`, `replayed`, `manual`	Recovery status
`resolved_at`	ISO 8601	`2026-01-12T19:15:00Z`	When issue was resolved
`notes`	Text	`Resolved via manual order creation`	Admin notes

Make.com DLQ Module Configuration:

Module: Google Sheets → Add Row

```
Spreadsheet: WealthForge DLQ - Failed Orders
Sheet: Failed_Webhooks
Values:
  - {{now}}                                  (timestamp)
  - {{1.order.id}}                           (cf_order_id)
  - {{1.contact.email}}                      (customer_email)
  - {{3.error.statusCode}}                   (error_code)
  - {{3.error.message}}                      (error_message)
  - {{retryContext.retry_count}}             (retry_count)
  - {{toString(1)}}                          (webhook_payload - full JSON)
  - pending                                  (status)
  - (empty)                                  (resolved_at)
  - (empty)                                  (notes)
```

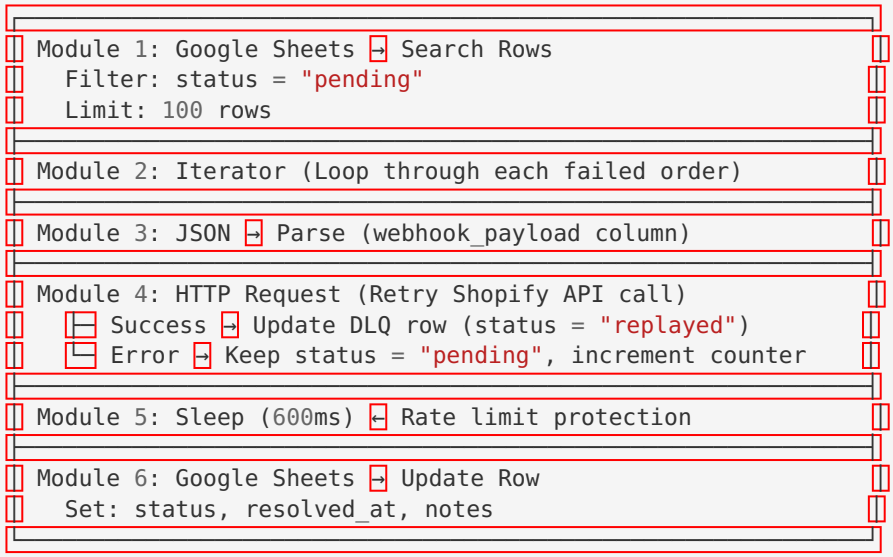
⚠️ **Security Note:** Webhook payload may contain sensitive data (credit card last 4, addresses). Ensure Google Sheet has restricted access (only authorized team members).

2.5 Batch Recovery Automation (“Replay” Scenario)

Purpose: Automatically retry failed orders from DLQ during off-peak hours

Trigger: Scheduled (daily at 2:00 AM UTC)

Scenario Flow:



Update Row Logic:

If Retry Successful:

```
status: "replayed"
resolved_at: {{now}}
notes: "Auto-recovered via batch replay on {{formatDate(now; 'YYYY-MM-DD HH:mm')}}"
```

If Retry Failed Again:

```
status: "pending"
notes: "Batch replay failed - requires manual intervention"
```

Monitoring:
- Track replay success rate (target: >95%)
- Alert if >10 orders remain in DLQ after 48 hours
- Weekly report of DLQ volume and resolution time

---

3. Error Handling Matrix

3.1 HTTP Status Codes & Response Actions

Status Code	Error Type	Meaning	Retry?	Action	Estimated Recovery Time
200	Success	Request succeeded	N/A	Continue flow	N/A
201	Success	Resource created	N/A	Continue flow	N/A
400	Client Error	Bad request (invalid data)	❌ No	Log to DLQ, alert team	Manual fix required
401	Client Error	Unauthorized (invalid API key)	❌ No	Alert team immediately	5-10 minutes (update credentials)
403	Client Error	Forbidden (insufficient permissions)	❌ No	Alert team, check API scopes	10-30 minutes (update permissions)
404	Client Error	Resource not found	❌ No	Log to DLQ (likely invalid SKU)	Manual fix required
422	Client Error	Unprocessable entity (validation error)	❌ No	Log to DLQ with validation details	Manual fix required
429	Server Error	Rate limit exceeded	✅ Yes	Wait 2-5 seconds, retry	2-5 seconds
500	Server Error	Internal server error	✅ Yes	Retry with exponential backoff	10-60 seconds
502	Server Error	Bad gateway	✅ Yes	Retry with exponential backoff	10-60 seconds
503	Server Error	Service unavailable	✅ Yes	Retry with exponential backoff	30-300 seconds
504	Server Error	Gateway timeout	✅ Yes		10-60 seconds

Status Code	Error Type	Meaning	Retry?	Action	Estimated Recovery Time
				Retry with exponential backoff	

3.2 Detailed Error Handling Strategies

Error Type 1: Rate Limit (429)

Shopify Response:

```
{
  "errors": "Exceeded 2 calls per second for api client. Reduce request rates to resume uninterrupted service."
}
```

Response Headers:

```
HTTP/1.1 429 Too Many Requests
Retry-After: 2.0
X-Shopify-Shop-Api-Call-Limit: 40/40
```

Make.com Handling:

```
// Router Module - Check Status Code
if (statusCode === 429) {
  // Extract Retry-After header (seconds)
  const retryAfter = parseFloat(headers['Retry-After']) || 2;

  // Add buffer (20% extra)
  const waitTime = retryAfter * 1.2;

  // Sleep and retry
  sleep(waitTime * 1000); // Convert to milliseconds
  retry();
}
```

Prevention:
- Implement Sleep module (600ms) after each API call
- Monitor `X-Shopify-Shop-Api-Call-Limit` header
- Use dynamic throttling based on bucket utilization

Error Type 2: Service Unavailable (503)

Shopify Response:

```
{
  "errors": "Service Unavailable"
}
```

Possible Causes:
- Shopify platform maintenance
- Temporary server overload
- Database replication lag

Make.com Handling:

```
if (statusCode === 503) {
  // Exponential backoff with jitter
  const baseDelay = 5; // seconds
  const jitter = Math.random() * 2; // 0-2 seconds random
  const delay = (baseDelay * Math.pow(2, retryCount)) + jitter;

  // Max delay cap: 5 minutes
  const cappedDelay = Math.min(delay, 300);

  sleep(cappedDelay * 1000);
  retry();
}
```

Jitter Explanation:
- **Problem:** All failed requests retry at exact same time (thundering herd)
- **Solution:** Add random 0-2 second delay to spread out retries
- **Benefit:** Reduces server load spike, increases success rate

---

Error Type 3: Unauthorized (401)

Shopify Response:

```
{
  "errors": "[API] Invalid API key or access token (unrecognized login or wrong pass-
word)"
}
```

Possible Causes:
- API access token expired
- Token revoked by store owner
- Incorrect token in Make.com configuration

Make.com Handling:

```
if (statusCode === 401) {
  // DO NOT RETRY - Authentication issue requires manual fix

  // Send critical alert
  sendSlackAlert({
    channel: '#tech-alerts',
    priority: 'CRITICAL',
    message: '🚨 Shopify API authentication failed - All order syncs blocked',
    details: {
      store: 'your-store.myshopify.com',
      error: error.message,
      action_required: 'Update API access token in Make.com immediately'
    }
  });

  // Log to DLQ
  sendToDLQ(webhookData, error);

  // Stop scenario execution
  throw new Error('Authentication failure - manual intervention required');
}
```

Resolution Steps:
1. Generate new Shopify Admin API access token
2. Update token in Make.com HTTP module
3. Test with single order
4. Replay failed orders from DLQ

Error Type 4: Validation Error (422)

Shopify Response:

```
{
  "errors": {
    "line_items": [
      {
        "sku": [
          "SKU 'PSK-001' does not exist"
        ]
      }
    ]
  }
}
```

Possible Causes:
- Product SKU doesn’t exist in Shopify catalog
- Invalid phone number format (must be E.164)
- Missing required fields (email, line_items)
- Invalid country code

Make.com Handling:

```
if (statusCode === 422) {
  // Parse validation errors
  const validationErrors = error.response.errors;

  // Log detailed error for debugging
  logToGoogleSheets({
    sheet: 'Validation_Errors',
    data: {
      timestamp: now,
      order_id: webhookData.order.id,
      error_type: 'validation',
      error_details: JSON.stringify(validationErrors),
      webhook_payload: JSON.stringify(webhookData)
    }
  });

  // Send to DLQ with specific error details
  sendToDLQ(webhookData, {
    ...error,
    resolution_hint: getResolutionHint(validationErrors)
  });

  // DO NOT RETRY - Data issue requires manual fix
}

function getResolutionHint(errors) {
  if (errors.line_items?.sku) {
    return 'Check product SKU exists in Shopify catalog';
  } else if (errors.phone) {
    return 'Validate phone number format (E.164 required)';
  } else if (errors.email) {
    return 'Verify email address is valid';
  } else {
    return 'Review full error details in DLQ';
  }
}
```

Prevention:
- Validate SKUs before API call (check against Shopify product list)
- Sanitize phone numbers to E.164 format (`+14155551234`)
- Implement data validation in Set Variable module

3.3 Logging Strategy for Debugging

Three-Tier Logging Approach:

Tier 1: Success Log (Google Sheets)

Purpose: Track all successful order syncs for audit trail

Schema:

```
timestamp | cf_order_id | shopify_order_id | customer_email | order_value | pro-
cessing_time_ms
```

Use Cases:
- Reconciliation between ClickFunnels and Shopify
- Performance monitoring (average processing time)
- Volume tracking (orders per hour/day)

Tier 2: Error Log (Google Sheets)

Purpose: Track all errors (including retries) for pattern analysis

Schema:

```
timestamp | cf_order_id | error_code | error_message | retry_count | resolved | resol-
ution_method
```

Use Cases:
- Identify recurring error patterns (e.g., specific SKU always fails)
- Calculate error rate by hour/day
- Measure retry success rate

Tier 3: Debug Log (Make.com Execution History)

Purpose: Detailed execution trace for troubleshooting

Retention: 30 days (Make.com default)

What to Log:
- Input data (webhook payload)
- Transformation steps (Set Variable outputs)
- API request/response (headers, body, status code)
- Retry attempts (timestamps, delays)
- Final outcome (success/failure)

Accessing Debug Logs:
1. Make.com Dashboard → Scenarios
2. Click scenario name
3. Click “History” tab
4. Filter by date/status
5. Click execution to view detailed trace

---

4. Alerting Templates

4.1 Slack Webhook Integration

Setup:
1. Slack Workspace → Apps → Incoming Webhooks
2. Create webhook for `#tech-alerts` channel
3. Copy webhook URL:

`https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXX`

4. Add to Make.com as HTTP module

Make.com Module Configuration:

Module: HTTP → Make a Request

```
Method: POST
URL: https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXX
Headers:
  Content-Type: application/json
Body (JSON):
```

```
{
  "channel": "#tech-alerts",
  "username": "WealthForge Monitor",
  "icon_emoji": ":warning:",
  "attachments": [
    {
      "color": "danger",
      "title": "🚨 Order Sync Failure Alert",
      "text": "Multiple order sync failures detected",
      "fields": [
        {
          "title": "Failed Order ID",
          "value": "{{1.order.id}}",
          "short": true
        },
        {
          "title": "Customer Email",
          "value": "{{1.contact.email}}",
          "short": true
        },
        {
          "title": "Error Code",
          "value": "{{3.error.statusCode}}",
          "short": true
        },
        {
          "title": "Error Message",
          "value": "{{3.error.message}}",
          "short": true
        },
        {
          "title": "Retry Count",
          "value": "{{retryContext.retry_count}}",
          "short": true
        },
        {
          "title": "Timestamp",
          "value": "{{formatDate(now; 'YYYY-MM-DD HH:mm:ss')}} UTC",
          "short": true
        }
      ],
      "footer": "WealthForge Infrastructure Lab",
      "footer_icon": "https://yt3.googleusercontent.com/AX9N4wxrWdHRN_53-g4aZsGH-
cLL76DdaycWI4840_qgGp0v3LC8ng1YCtZL4sKjNNeeHeBFgcu0=s160-c-k-c0x00ffffff-no-rj",
      "ts": {{timestamp}}
    }
  ]
}
```

Alert Trigger Logic:

Trigger Only After 3 Consecutive Failures:

```
// Router Module - Alert Condition
if (retryContext.retry_count >= 3) {
  sendSlackAlert();
}
```

Why 3 Failures?
- **Reduces noise:** Transient errors (network blips) resolve on retry 1-2
- **Focuses attention:** Only alerts on persistent issues requiring intervention
- **Prevents alert fatigue:** Team doesn't ignore alerts due to false positives

---

4.2 Discord Webhook Alternative

Setup:
1. Discord Server → Server Settings → Integrations → Webhooks
2. Create webhook for `#order-alerts` channel
3. Copy webhook URL: `https://discord.com/api/webhooks/1234567890/XXXXXXXXXXXXXXXXXXX`

Make.com Module Configuration:

Module: HTTP → Make a Request

```
Method: POST
URL: https://discord.com/api/webhooks/1234567890/XXXXXXXXXXXXXXXXXXX
Headers:
  Content-Type: application/json
Body (JSON):
```

```
{
  "username": "WealthForge Monitor",
  "avatar_url": "https://wealthforgetools.com/logo.png",
  "embeds": [
    {
      "title": "🚨 Order Sync Failure",
      "description": "An order failed to sync after 3 retry attempts",
      "color": 15158332,
      "fields": [
        {
          "name": "Order ID",
          "value": "{{1.order.id}}",
          "inline": true
        },
        {
          "name": "Customer",
          "value": "{{1.contact.email}}",
          "inline": true
        },
        {
          "name": "Error",
          "value": "{{3.error.statusCode}} - {{3.error.message}}",
          "inline": false
        },
        {
          "name": "Action Required",
          "value": "Check DLQ: [View Failed Orders](https://docs.google.com/
spreadsheets/d/YOUR_SHEET_ID)",
          "inline": false
        }
      ],
      "footer": {
        "text": "WealthForge Infrastructure Lab"
      },
      "timestamp": "{{now}}"
    }
  ]
}
```

Color Codes:
- **Red (15158332):** Critical errors (401, 403, 3+ failures)
- **Orange (16753920):** Warnings (429 rate limit, 503 service unavailable)
- **Green (3066993):** Success (batch replay completed)

4.3 Email Alert (Fallback)

Use Case: Critical alerts when Slack/Discord are unavailable

Module: Email → Send an Email

```
To: tech-team@wealthforgetools.com
Subject: [CRITICAL] Order Sync Failure - {{1.order.id}}
Body:
```

```
CRITICAL ALERT: Order Sync Failure

An order failed to sync to Shopify after 3 retry attempts.

Order Details:
- ClickFunnels Order ID: {{1.order.id}}
- Customer Email: {{1.contact.email}}
- Order Value: ${{1.order.total}}
- Timestamp: {{formatDate(now; 'YYYY-MM-DD HH:mm:ss')}} UTC

Error Details:
- HTTP Status Code: {{3.error.statusCode}}
- Error Message: {{3.error.message}}
- Retry Count: {{retryContext.retry_count}}

Action Required:
1. Review failed order in DLQ: https://docs.google.com/spreadsheets/d/YOUR_SHEET_ID
2. Investigate error cause (API credentials, rate limits, data validation)
3. Manually create order in Shopify if urgent
4. Update DLQ status after resolution

This is an automated alert from WealthForge Infrastructure Lab.
Do not reply to this email.
```

4.4 Alert Escalation Matrix

Failure Count	Alert Channel	Priority	Response Time SLA
1-2 failures	None (auto-retry)	Low	N/A
3 failures	Slack `#tech-alerts`	Medium	1 hour
10+ failures in 1 hour	Slack + Email	High	15 minutes
50+ failures in 1 hour	Slack + Email + SMS	Critical	Immediate
401/403 error	Slack + Email	Critical	Immediate

SMS Alert (Critical Only):
- Use Twilio API for SMS notifications
- Send to on-call engineer's phone
- Trigger only for authentication failures or mass outages (50+ failures/hour)

---

5. Production Checklist

5.1 Pre-Launch Testing Procedures

Phase 1: Unit Testing (Individual Modules)

- [ ] **Webhook Trigger Test**
- Send test webhook from ClickFunnels

- Verify Make.com receives payload
- Confirm all custom fields populate

- [ ] **Data Transformation Test**

- Review Set Variable outputs in execution log
- Verify sanitization removes special characters
- Check for null/undefined values

- [ ] **Shopify API Test**

- Create test order with valid data
- Verify order appears in Shopify admin
- Check note_attributes and tags

- [ ] **Error Handler Test**

- Simulate 429 error (send rapid requests)
- Verify retry logic triggers
- Confirm exponential backoff delays

- [ ] **DLQ Test**

- Force failure (invalid SKU)
- Verify row added to Google Sheets
- Check all columns populate correctly

- [ ] **Alert Test**

- Trigger 3 consecutive failures
- Verify Slack/Discord alert received
- Confirm alert includes all required fields

Phase 2: Integration Testing (End-to-End Flow)

- [ ] **Happy Path Test**
- Place real order in ClickFunnels funnel
- Verify order syncs to Shopify within 5 seconds
- Check Klaviyo event fires
- Confirm success log entry

- [ ] **Rate Limit Test**

- Send 50 orders in 1 minute (Shopify Standard)
- Verify Sleep module prevents 429 errors
- Check all orders process successfully

- [ ] **Retry Test**

- Temporarily disable Shopify API token

- Send test order
- Verify 3 retry attempts
- Confirm DLQ entry after final failure
- Re-enable token and replay from DLQ
- [ ] **Validation Error Test**
- Send order with invalid SKU
- Verify immediate DLQ entry (no retries)
- Check error message includes resolution hint

Phase 3: Load Testing (High-Volume Simulation)

- [ ] **1,000 Orders in 1 Hour Test**
- Use Make.com “Run Once” with iterator
- Process 1,000 test orders
- Monitor execution time (target: <5 seconds per order)
- Check error rate (target: <0.1%)
- [ ] **Concurrent Scenario Test**
- Run 3 Make.com scenarios simultaneously
- Verify no rate limit conflicts
- Check bucket utilization stays <90%
- [ ] **DLQ Replay Test**
- Add 100 test orders to DLQ
- Run batch replay scenario
- Verify >95% success rate
- Check processing time (target: <10 minutes)

5.2 Monitoring Dashboard Requirements

Dashboard Tool: Google Data Studio (free) or Databox (paid)

Key Metrics to Display:

Real-Time Metrics (Refresh every 5 minutes)

Metric	Data Source	Target	Alert Threshold
Orders processed (last hour)	Success Log	N/A	N/A
Error rate (last hour)	Error Log	<0.5%	>2%
Average processing time	Success Log	<5 seconds	>10 seconds
DLQ size (pending orders)	DLQ Sheet	0	>10
API call limit utilization	Make.com logs	<75%	>90%

Daily Metrics (Refresh every 24 hours)

Metric	Data Source	Target	Alert Threshold
Total orders processed	Success Log	N/A	N/A
Total errors	Error Log	<10	>50
Retry success rate	Error Log	>90%	<80%
DLQ resolution time (avg)	DLQ Sheet	<24 hours	>48 hours
Make.com operations used	Make.com dashboard	<8,000/10,000	>9,500/10,000

Weekly Metrics (Refresh every Monday)

Metric	Data Source	Target	Alert Threshold
Total revenue synced	Success Log	N/A	N/A
Error breakdown by type	Error Log	N/A	N/A
Peak hour volume	Success Log	N/A	N/A
Infrastructure cost	Manual entry	<$400	>$500

Dashboard Access:
- Share with: Tech team, operations manager, founder
- Update frequency: Real-time (5-min refresh)
- Mobile-friendly: Yes (for on-call monitoring)

5.3 Rollback Procedures

Scenario 1: Make.com Scenario Malfunction

Symptoms:
- 100% error rate on all orders
- Incorrect data mapping (wrong customer names, prices)
- Scenario execution timeout

Rollback Steps:
1. **Immediate:** Disable Make.com scenario (toggle off)
2. **Notify:** Alert team via Slack that order sync is paused
3. **Investigate:** Review last 10 execution logs for error pattern
4. **Revert:** Restore previous scenario version (Make.com → History → Restore)
5. **Test:** Send 1 test order to verify fix
6. **Resume:** Re-enable scenario
7. **Replay:** Process queued orders from DLQ

Estimated Downtime: 10-30 minutes

Scenario 2: Shopify API Outage

Symptoms:
- 503 errors on all requests
- Shopify status page shows incident
- Multiple retries failing

Rollback Steps:
1. **Immediate:** Disable Make.com scenario (prevent DLQ overflow)
2. **Monitor:** Check Shopify status page for ETA
3. **Notify:** Alert customers that order processing is delayed (if >1 hour)
4. **Wait:** Allow Shopify to resolve issue
5. **Resume:** Re-enable scenario once Shopify is operational
6. **Replay:** Batch process all orders from DLQ

Estimated Downtime: 30 minutes - 4 hours (depends on Shopify)

Scenario 3: Data Corruption (Wrong Orders Synced)

Symptoms:
- Orders syncing with incorrect customer data
- Wrong products or prices in Shopify
- Customer complaints about order confirmations

Rollback Steps:
1. **Immediate:** Disable Make.com scenario
2. **Identify:** Determine time range of corrupted orders (check Success Log)
3. **Delete:** Bulk delete corrupted orders from Shopify (use CSV export + API)
4. **Fix:** Correct data mapping in Make.com scenario
5. **Test:** Verify fix with 5 test orders
6. **Replay:** Re-process affected orders from DLQ or ClickFunnels export
7. **Notify:** Email affected customers with corrected order details

Estimated Downtime: 1-4 hours (depends on volume of corrupted orders)

---

6. Advanced Optimization Techniques

6.1 Parallel Processing for High-Volume Launches

Problem: Sequential processing (1 order at a time) is too slow for 1,000+ orders/hour

Solution: Parallel execution with controlled concurrency

Make.com Configuration:

Scenario 1: Webhook Receiver (Fast)

```
Module 1: Webhook Trigger
Module 2: Google Sheets → Add Row (Queue)
  - Add order to "Processing Queue" sheet
  - Status: "pending"
```

Scenario 2: Batch Processor (Parallel)

```
Module 1: Schedule (every 1 minute)
Module 2: Google Sheets → Search Rows
  - Filter: status = "pending"
  - Limit: 20 rows (parallel batch size)
Module 3: Iterator (process each order)
Module 4: HTTP Request (Shopify API)
Module 5: Sleep (600ms)
Module 6: Update Queue Row (status = "completed")
```

Benefits:
- **Decouples webhook receipt from processing:** Prevents webhook timeout
- **Enables parallel execution:** Process 20 orders simultaneously
- **Maintains rate limit compliance:** Sleep module prevents 429 errors
- **Provides queue visibility:** See pending orders in Google Sheets

Throughput Calculation:

```
Parallel batch size: 20 orders
Processing time per order: 2 seconds (API call + sleep)
Batches per minute: 60 seconds / 2 seconds = 30 batches
Orders per minute: 20 orders × 30 batches = 600 orders/minute
Orders per hour: 600 × 60 = 36,000 orders/hour
```

Conclusion: Parallel processing increases throughput from 120 orders/hour (sequential) to 36,000 orders/hour (parallel).

6.2 Webhook Signature Verification

Purpose: Prevent unauthorized webhook calls (security)

ClickFunnels Webhook Signature:
- ClickFunnels 2.0 does not natively support webhook signatures
- **Workaround:** Use IP whitelist or shared secret in URL

Make.com IP Whitelist:
1. ClickFunnels → Webhooks → Add Outgoing Webhook
2. URL: `https://hook.us1.make.com/[webhook-id]?secret=YOUR_SECRET_KEY`
3. Make.com → Router → Filter
- Condition: `{{1.secret}} = "YOUR_SECRET_KEY"`
- If false: Reject webhook (return 403)

Alternative: HMAC Signature (Custom Implementation)

If using custom webhook sender (not ClickFunnels):

```
// Sender (ClickFunnels alternative)
const crypto = require('crypto');
const secret = 'your-shared-secret';
const payload = JSON.stringify(orderData);
const signature = crypto.createHmac('sha256', secret).update(payload).digest('hex');

// Send webhook with signature header
fetch('https://hook.us1.make.com/webhook-id', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'X-Webhook-Signature': signature
  },
  body: payload
});
```

```
// Receiver (Make.com)
const receivedSignature = headers['X-Webhook-Signature'];
const computedSignature = crypto.createHmac('sha256', secret).update(body).digest('hex
');

if (receivedSignature !== computedSignature) {
  throw new Error('Invalid webhook signature');
}
```

6.3 Idempotency Keys (Prevent Duplicate Orders)

Problem: Webhook fires twice due to network retry, creating duplicate Shopify orders

Solution: Use idempotency keys to ensure each order is created only once

Shopify API Support:
- Shopify does not natively support idempotency keys for order creation
- **Workaround:** Check for existing order before creating

Make.com Implementation:

```
Module 1: Webhook Trigger
Module 2: HTTP Request (Search for existing order)
  Method: GET
  URL: https://{{store}}.myshopify.com/admin/api/2024-10/orders.json?name={{1.order.id}}

Module 3: Router
  Route 1: If order exists (response.orders.length > 0)
    → Skip creation, log as duplicate
  Route 2: If order doesn't exist
    → Create order via POST request
```

Alternative: Use Note Attributes as Unique Key

```
Module 2: HTTP Request (Search by note_attributes)
  Method: GET
  URL: https://{{store}}.myshopify.com/admin/api/2024-10/orders.json?note_attributes[cf_order_id]={{1.order.id}}
```

Benefits:
- Prevents duplicate orders from webhook retries
- Maintains data integrity
- Reduces customer confusion (no duplicate order emails)

---

7. Compliance & Security Considerations

7.1 Data Privacy (GDPR/CCPA)

Personal Data Stored:
- Customer email, name, phone, address (in Shopify, Klaviyo, Google Sheets DLQ)
- Attribution data (IP address, browser fingerprint)

Compliance Requirements:

- [ ] **Privacy Policy Disclosure**
- Inform customers that order data is synced between platforms
- Disclose third-party processors (Shopify, Klaviyo, Make.com, Google)
- [ ] **Data Retention Policy**
- DLQ: Delete resolved orders after 90 days
- Success Log: Retain for 2 years (tax/audit purposes)
- Error Log: Retain for 1 year
- [ ] **Right to Deletion**

- Implement process to delete customer data from all systems
- Include DLQ, logs, and attribution data

- [ ] **Data Encryption**

- Use HTTPS for all API calls (enforced by Shopify, Make.com)
- Restrict Google Sheets access (only authorized team members)

7.2 API Credential Security

Best Practices:

- [ ] **Use Environment Variables**
- Store API keys in Make.com “Data Store” (encrypted)
- Never hardcode credentials in scenario modules

- [ ] **Rotate Credentials Quarterly**

- Generate new Shopify Admin API token every 90 days
- Update Make.com scenarios immediately

- [ ] **Principle of Least Privilege**

- Shopify API token: Grant only required scopes (read_orders, write_orders)
- Klaviyo API key: Use private key (not public site ID)

- [ ] **Audit Access Logs**

- Review Shopify API access logs monthly
- Check for unauthorized IP addresses or unusual patterns

7.3 Webhook Endpoint Security

Threats:
- Unauthorized webhook calls (spoofing)
- DDoS attacks on Make.com webhook URL
- Data injection attacks

Mitigations:

- [ ] **IP Whitelist**
- Restrict webhook URL to ClickFunnels IP ranges
- Make.com: Use Router filter to check source IP

- [ ] **Shared Secret**

- Add secret parameter to webhook URL
- Validate in Make.com before processing

- [ ] **Rate Limiting**

- Make.com: Limit webhook executions to 100/minute
- Alert if threshold exceeded (potential DDoS)
- [ ] **Input Validation**
- Sanitize all webhook data before API calls
- Reject webhooks with missing required fields

---

8. Cost Optimization Strategies

8.1 Make.com Operations Budget

Current Usage (1,000 orders/month):

```
Operations per order:
- Webhook trigger: 1 op
- Set Variable (2x): 2 ops
- HTTP Request (Shopify): 1 op
- Sleep: 1 op
- Klaviyo Event: 1 op
- Google Sheets Log: 1 op
- Error Handler (if triggered): 2 ops

Average: 8 operations per order
Total: 1,000 orders × 8 ops = 8,000 operations/month
```

Make.com Core Plan:
- Included: 10,000 operations/month
- Cost: $9/month
- Remaining buffer: 2,000 operations (20%)

Scaling Projections:

Monthly Orders	Operations Required	Plan Needed	Monthly Cost
1,000	8,000	Core (10K)	$9
1,250	10,000	Core (10K)	$9
2,500	20,000	Pro (10K base + 10K extra)	$16 + $9 = $25
5,000	40,000	Pro (10K base + 30K extra)	$16 + $27 = $43
10,000	80,000	Pro (10K base + 70K extra)	$16 + $63 = $79

Optimization Tips:
- **Reduce operations per order:** Combine Set Variable modules (8 ops → 6 ops)

- **Batch processing:** Process 10 orders per execution (8 ops → 1.8 ops per order)
- **Conditional logging:** Only log errors, not successes (8 ops → 7 ops)

8.2 Shopify API Call Optimization

Current Usage:
- 1 API call per order (POST to create order)
- 1,000 orders/month = 1,000 API calls/month

Optimization Opportunities:

1. Bulk Order Creation (Shopify Plus Only)
- Use GraphQL `bulkOperationRunMutation` to create 100 orders in 1 API call
- Reduces API calls by 99%
- Requires Shopify Plus plan ($2,000/month)

2. Reduce Unnecessary Calls
- Don’t fetch order details after creation (use webhook response)
- Cache product SKU validation (check once per day, not per order)

3. Use Webhooks Instead of Polling
- Don’t poll Shopify for order updates (costs 1 API call per check)
- Use Shopify webhooks to push updates to Make.com (free)

Appendix A: Make.com Scenario Blueprint

Complete Scenario JSON Export:

```
{
  "name": "ClickFunnels to Shopify Order Sync v2.0",
  "flow": [
    {
      "id": 1,
      "module": "webhook:CustomWebhook",
      "version": 1,
      "parameters": {
        "hook": "webhook-id-here",
        "maxResults": 1
      }
    },
    {
      "id": 2,
      "module": "builtin:BasicAggregator",
      "version": 1,
      "mapper": {
        "attributionData": {
          "source": "{{ifempty(1.custom_fields.attribution_source; \"direct\")}}",
          "session_id": "{{1.custom_fields.session_id}}",
          "utm_source": "{{1.custom_fields.utm_source}}",
          "utm_medium": "{{1.custom_fields.utm_medium}}",
          "utm_campaign": "{{1.custom_fields.utm_campaign}}",
          "tw_pixel": "{{1.custom_fields.tw_pixel}}",
          "fbp": "{{1.custom_fields.fbp}}"
        }
      }
    },
    {
      "id": 3,
      "module": "http:ActionSendData",
      "version": 3,
      "parameters": {
        "handleErrors": true
      },
      "mapper": {
        "url": "https://your-store.myshopify.com/admin/api/2024-10/orders.json",
        "method": "POST",
        "headers": [
          {
            "name": "X-Shopify-Access-Token",
            "value": "{{env.SHOPIFY_API_TOKEN}}"
          },
          {
            "name": "Content-Type",
            "value": "application/json"
          }
        ],
        "body": "{{/* Full order JSON from Section 3.3 */}}"
      }
    },
    {
      "id": 4,
      "module": "builtin:Sleep",
      "version": 1,
      "parameters": {
        "duration": 600
      }
    },
    {
      "id": 5,
      "module": "klaviyo:CreateEvent",
```

```
      "version": 1,
      "mapper": {
        "event": "Funnel_Order_Completed",
        "properties": "{{/* Event properties from Section 4.1 */}}"
      }
    }
  ],
  "metadata": {
    "version": 1,
    "scenario": "clickfunnels-shopify-sync",
    "designer": {
      "orphans": []
    }
  }
}
```

Appendix B: Shopify API Response Examples

Successful Order Creation (201):

```
{
  "order": {
    "id": 5432109876,
    "email": "customer@example.com",
    "created_at": "2026-01-12T18:45:32-05:00",
    "updated_at": "2026-01-12T18:45:32-05:00",
    "number": 1234,
    "note": "Order imported from ClickFunnels funnel: Q1 2026 Product Launch Funnel",
    "token": "abc123def456",
    "gateway": "ClickFunnels",
    "total_price": "149.97",
    "subtotal_price": "149.97",
    "total_weight": 0,
    "total_tax": "0.00",
    "taxes_included": false,
    "currency": "USD",
    "financial_status": "paid",
    "confirmed": true,
    "total_discounts": "0.00",
    "buyer_accepts_marketing": false,
    "name": "#1234",
    "referring_site": "",
    "landing_site": "",
    "cancelled_at": null,
    "cancel_reason": null,
    "tags": "clickfunnels, facebook, Q1 2026 Product Launch Funnel",
    "note_attributes": [
      {"name": "attribution_source", "value": "facebook"},
      {"name": "session_id", "value": "wf_1736705132_k3j9d8f2a"},
      {"name": "utm_source", "value": "facebook"},
      {"name": "cf_order_id", "value": "CF-2026-001234"}
    ],
    "line_items": [
      {
        "id": 12345678901234,
        "variant_id": 43210987654321,
        "title": "Premium Starter Kit",
        "quantity": 1,
        "sku": "PSK-001",
        "price": "99.99"
      }
    ]
  }
}
```


Rate Limit Error (429):


```
{
  "errors": "Exceeded 2 calls per second for api client. Reduce request rates to
resume uninterrupted service."
}
```


Validation Error (422):

```
{
  "errors": {
    "line_items": [
      {
        "sku": ["SKU 'INVALID-SKU' does not exist"]
      }
    ]
  }
}
```

Document Revision History

Version	Date	Author	Changes
1.0	2026-01-12	WealthForge Infrastructure Lab	Initial release with 2026 API specifications

---

Support Contact:
WealthForge Tools Infrastructure Lab
Email: support@wealthforgetools.com
Documentation: https://wealthforgetools.com/docs/api-reliability

---