BigCommerce Plugin for ChessWorld SKU Management

Technical Setup and Architecture Guide

Version 4.0

Written by:

Rui En Koe

Alex Chan

Ashley Warden

Liangdi Wang

Luqmaan Yurzaa

Wen Cheng Huong

Document History and Version Control

Date (dd/mm/yy)	Version	Author	Description
17/04/2025	1.0	Liangdi Wang	Initial Technical Setup and Architecture Guide Draft
17/04/2025	1.0	Alex Chan	Include Ngrok Setup in the document
18/08/2025	3.0	Liangdi Wang	Fill out deployment process section
21/09/2025	3.0	Liangdi Wang	Complete technical architecture, key implementation details, and testing and debugging sections

Contents

1. Introduction

1.1 Purpose of this Document

This document serves as a concise technical guide for setting up and understanding the architecture of the BigCommerce Plugin for ChessWorld SKU Management. It is primarily intended for:

- New team members joining the development effort
- Existing team members who need a reference for setup procedures
- Technical documentation for project handover

This guide focuses specifically on the technical implementation details and development environment setup.

1.2 Project Overview

The SKU Management Plugin to be developed addresses BigCommerce's limitation in handling bundled products with accurate inventory tracking. The plugin will enable:

- Creation of bundled products composed of multiple individual SKUs
- Independent tracking of component SKUs across all bundles
- Prevention of overselling when shared SKUs are used in multiple products
- Real-time inventory synchronization between bundles and individual items

This technical implementation plans to leverage BigCommerce's REST Catalog API for product and SKU management, along with Webhooks for real-time inventory updates when sales occur. The solution will be built using Next.js and React for the frontend interface, with a PostgreSQL database to store supplementary data not maintained in BigCommerce's system.

2. Development Environment Setup

Here is a guide to set up the local environment, but you can also look at the quickstart guide by BigCommerce, as well as the README in our repo.

2.1 Prerequisites

Before setting up the development environment for the BigCommerce Plugin, ensure you have the following:

- Node.js and npm: Node.js version 22+ and npm version 10+
- Developer Portal Account: Required to register the app and accessing sandbox store
- ngrok: Used to expose your local development server to the internet
- Docker: To run local database in a container
- Git: For version control and accessing the repository
- An IDE (Integrated development environment), i.e. VS Code

2.2 Ngrok Setup

1. Create an Ngrok Account and find and save your Authtoken
2. Download and install Ngrok and authenticate your Ngrok agent:
   ngrok config add-authtoken <TOKEN>
3. Set up ngrok to create a secure tunnel: (port 3000 is default for Next.js)
   ngrok http 3000
   a. Note the HTTPS URL provided by ngrok (e.g., https://12345.ngrok-free.app) as you'll need it for the next steps.

2.3 BigCommerce Developer Account Setup

1. Sign up for a Developer account, and sign in
2. Go to https://manage.bigcommerce.com/ and create a store, this will be a sandbox store where you can install your plugin and test it during development
3. Go to https://devtools.bigcommerce.com/my/apps and create an app (plugin)
4. Provide a name for the app (e.g., "ChessWorld SKU Manager")
5. Navigate to the Technical tab and configure the following (using URL from Ngrok earlier):
   a. Auth Callback URL: https://{ngrok_url}/api/auth
   b. Load Callback URL: https://{ngrok_url}/api/load
   c. Uninstall Callback URL: https://{ngrok_url}/api/uninstall
   Eg. https://12345.ngrok-free.app/api/auth

6. Under OAuth scopes, set the Products scope to "modify" as the plugin will need to manage product inventory
7. Click "Update & Close"
8. Click "View Client ID" to access the app's client ID and client secret for the next steps

2.4 Local Development Configuration

1. Navigate to your desired directory and clone the repository
   ```
   git clone https://github.com/VictoriaKoe/BigCommerce-Plugin.git
   cd BigCommerce-Plugin
   ```
2. Install dependencies:
   ```
   npm install
   ```
3. Create a .env file by copying the .env-sample file
4. Configure the .env file with the following:
   a. Replace CLIENT_ID and CLIENT_SECRET with values from the Developer Portal https://devtools.bigcommerce.com/my/apps
   b. Update AUTH_CALLBACK with your ngrok URL (e.g., https://12345.ngrok-free.app/api/auth)
   c. Set a secure JWT_KEY (at least 32 random characters)
   d. Set DB_TYPE to postgresql for our implementation
   e. Configure the PostgreSQL database connection parameters
      i. POSTGRES_URL=postgres://postgres:postgres@db.localtest.me:5432/main
5. Start the local PostgreSQL database:
   (make sure you have started Docker engine)
   ```
   npm run db:start
   ```
6. Set up the initial database:
   ```
   npm run db:setup
   ```
7. Start the development server:
   ```
   npm run dev
   ```
8. Install the app on your BigCommerce sandbox store:
   a. Sign in to your store - click control panel at https://manage.bigcommerce.com/
   b. Navigate to Apps > My Apps > My Draft Apps
   c. Find and install your app
   d. You should see an "Authorization Successful" message if everything is configured correctly

2.5 Common Setup Issues and Solutions

- ngrok tunnel expiration: Free ngrok tunnels expire after 2 hours. If this happens, restart ngrok and update the callback URLs in both the Developer Portal and your .env file.
- Database connection errors: Ensure your PostgreSQL server is running and the credentials in the .env file are correct. Verify network access if the database is hosted remotely.
- Authentication failures: Double-check that the CLIENT_ID and CLIENT_SECRET in your .env file match exactly with those in the Developer Portal.
- Callback URL errors: Ensure the ngrok URL is correctly formatted and includes the /api/auth path. The URL in your .env file must match the one registered in the Developer Portal.
- JWT errors: Make sure your JWT_KEY is at least 32 characters long to support HS256 encryption.

3. Technical Architecture

3.1 Technology Stack Overview

3.1.1 Next.js Framework

The application uses Next.js as the primary framework, providing:

- **API Routes**: Server-side endpoints in `/pages/api/` for BigCommerce integration
- **React Pages**: Frontend components in `/pages/` for the user interface
- **Static Generation**: Pre-built pages for better performance
- **File-based Routing**: Automatic routing based on file structure

3.1.2 React Components

React components serve two main UI contexts in this BigCommerce app:

- **Plugin Pages**: Full-page interfaces accessible through the BigCommerce admin navigation (product management, bundle listing)
- **App Extensions**: Side panel components that overlay the existing BigCommerce UI (product detail pages, allowing bundle configuration without leaving the native interface)
- **Shared Components**: Reusable form elements, tables, and controls used across both contexts for consistent user experience

3.1.3 Database Structure

PostgreSQL database with core authentication and session tables:

- **users**: Global user data (id, email, username)
- **stores**: Store credentials and OAuth scopes
- **store_users**: Multi-store user permissions

Note: Bundle configuration data is stored entirely in BigCommerce metafields rather than local database tables, eliminating the need for separate `bundles` and `bundle_links` tables.

3.2 BigCommerce Integration

3.2.1 REST Catalog API Usage

The app integrates with BigCommerce's REST API for:

- **Product Management**: Reading product data, SKUs, and inventory levels
- **Metafields**: Storing bundle configuration data on products
- **Inventory Updates**: Synchronizing stock levels between bundles and components
- **Variant Support**: Handling both product-level and variant-level bundles

3.2.2 Webhook Implementation

Automated webhooks handle real-time updates:

- **Order Created**: Updates bundle inventory when orders are placed
- **Product Updated**: Synchronizes changes to component products
- **Automatic Setup**: Webhooks are created during app installation
- **Error Handling**: Graceful fallback when webhook creation fails

3.2.3 Authentication Flow

OAuth 2.0 implementation with JWT tokens:

- **Installation**: BigCommerce redirects to `/api/auth` for initial setup
- **Session Management**: JWT tokens encode store context and user permissions
- **Load Verification**: `/api/load` validates app access on each session
- **Context Passing**: Authentication context is passed via URL parameters using the pattern `?context=${encodeURIComponent(context)}` throughout the application
- **Multi-store Support**: Users can access multiple stores with appropriate permissions

4. Key Implementation Details

4.1 Bundle Management Logic

Bundle products are identified and managed through BigCommerce metafields:

- **Bundle Detection**: Products with `is_bundle=true` metafield in the `bundle` namespace
- **Component Linking**: `linked_product_ids` metafield stores array of component product IDs
- **Quantity Mapping**: `linked_product_quantities` metafield defines quantities for each component
- **SKU Prefixing**: Bundle products automatically get "BUN-" prefix added to their SKU
- **Variant Support**: Both product-level and variant-level bundles are supported

4.2 Inventory Synchronization

Real-time inventory management ensures accurate stock levels:

- **Order Processing**: Webhooks trigger when orders are created, updating component inventory
- **Bundle Stock Calculation**: Bundle inventory = minimum(component_stock / required_quantity)
- **Oversell Prevention**: Stock validation prevents selling bundles when components are insufficient
- **Automatic Updates**: Component stock changes automatically recalculate bundle availability
- **Multi-bundle Support**: Single components can be used across multiple bundles

4.3 Database Schema and Relationships

Core database tables and their relationships:

Authentication Tables:

- `users` → `store_users` (one-to-many): Users can access multiple stores
- `stores` ← `store_users` (one-to-many): Stores can have multiple users

Data Flow:

1. Product data stored in BigCommerce via REST API
2. Bundle configuration stored entirely in BigCommerce metafields
3. PostgreSQL database only handles authentication and session management
4. Webhooks synchronize inventory changes between BigCommerce products

5. Testing and Debugging

5.1 Local Testing Procedures

Bundle Creation Testing:

1. Navigate to the product management interface
2. Select a product and toggle "Is this product a bundle?"
3. Add component products with specified quantities
4. Verify bundle inventory calculation matches expected values
5. Test bundle creation with both products and variants

Inventory Synchronization Testing:

1. Use the simulate sale feature at `/test/simulate-sale`
2. Create test orders through BigCommerce admin
3. Verify component inventory decreases correctly
4. Check bundle availability updates automatically
5. Test oversell prevention when components are insufficient

API Endpoint Testing:

- `/api/bundles/list` - Verify bundle and product data retrieval
- `/api/bundles/[id]` - Test individual bundle details and updates
- `/api/products/list` - Check product filtering and inventory data
- `/api/webhooks/orders` - Validate order processing webhook

5.2 Debugging Common Issues

Authentication Problems:

- Check JWT token validity and expiration
- Verify store hash and user permissions in database
- Ensure OAuth scopes include required permissions
- Review callback URL configuration in Developer Portal

Inventory Sync Issues:

- Check webhook delivery in BigCommerce admin
- Verify metafield data format and namespace
- Review database bundle_links relationships
- Test manual inventory updates via API

Database Connection Errors:

- Confirm PostgreSQL service is running (Docker)
- Check connection string format and credentials
- Verify network connectivity to database
- Review Neon proxy configuration for local development

Bundle Configuration Problems:

- Validate metafield JSON structure
- Check product ID references are correct
- Verify component products have inventory tracking enabled
- Review SKU prefix application ("BUN-")

6. Deployment Process

6.1 Preparing for Production

Before deploying the plugin to a production environment, several steps were required to ensure stability and compatibility:

- Code Quality: Fixed build and linting issues to ensure the app compiled cleanly and passed checks.
- Hosting: Selected Vercel for deployment due to its seamless integration with Next.js and free hosting tier.
- Database Migration: Migrated from MySQL to PostgreSQL (Neon Postgres on Vercel) for simplicity and free-tier hosting support. Updated environment variables and connection logic accordingly.
- Authentication Context: Adjusted the app extension to correctly pass authentication context to ensure proper session handling in production.
- Webhook Automation: Configured the app to automatically create required webhooks during setup, removing the need for manual commands.
- Environment Variables: Updated callback URLs in the BigCommerce Developer Portal to point to the deployed Vercel domain.
- Debugging: Used Vercel logs to identify and resolve runtime issues during deployment.

At this stage, the app is fully deployed and functional in a production-like environment, though not yet published to the BigCommerce App Marketplace.

6.2 BigCommerce App Store Submission

Once the app is production-ready, it can be submitted for review and listing on the BigCommerce App Marketplace. The process involves:

1. App Information: Provide technical details (callback URLs, multiple user support, API credentials).
2. OAuth Scopes: Configure only the required scopes (e.g., Products: Modify).
3. Listing Information: Add marketplace details such as app name, summary, description, support contact, and media assets (logos, screenshots, videos).
4. Testing Instructions: Supply clear steps for reviewers to test the app.
5. Legal Requirements: Include privacy policy and terms of service.
6. Submission: Submit the app for review and pay the listing fee.

For full details, refer to the official BigCommerce documentation:
https://developer.bigcommerce.com/docs/integrations/apps/guide/publishing