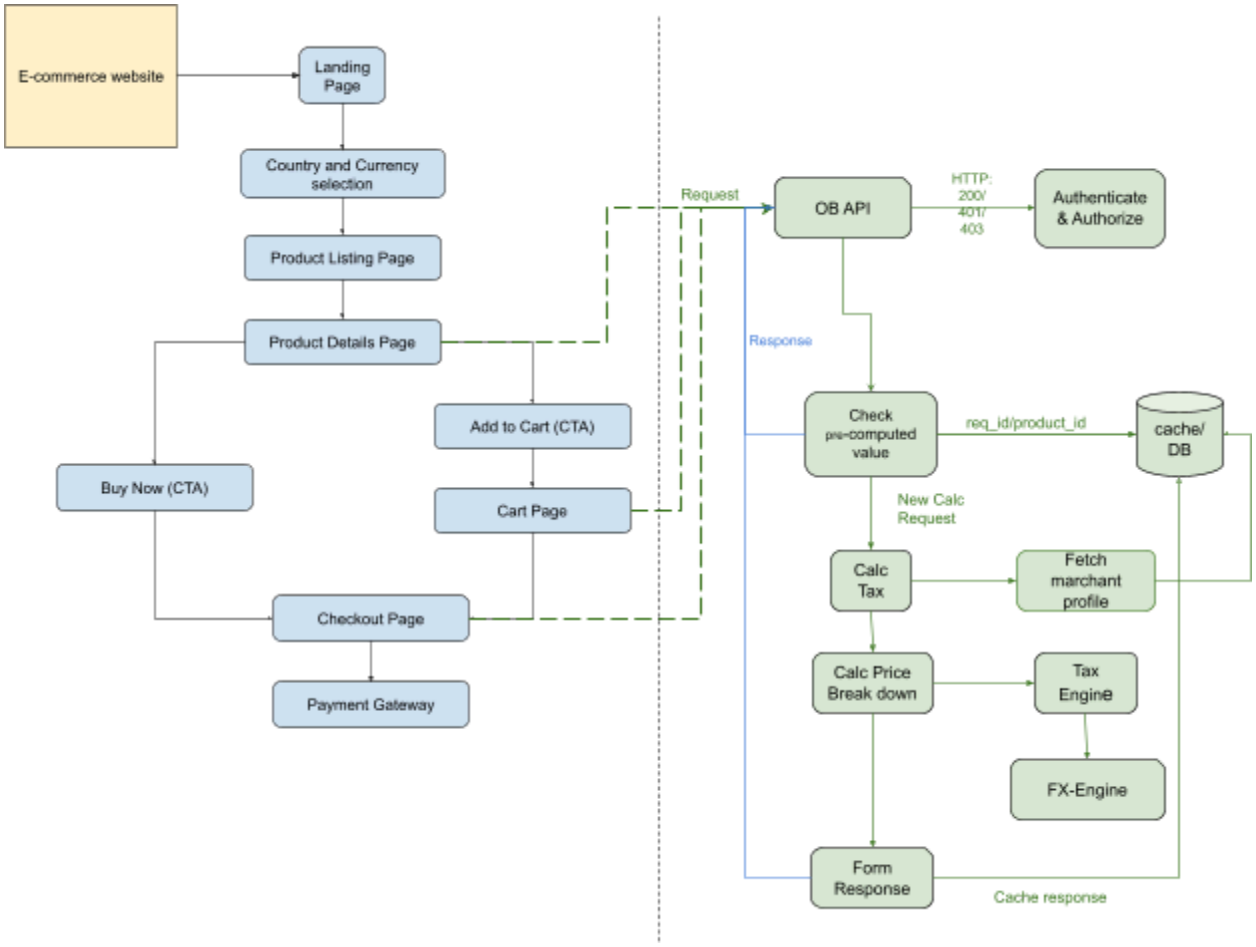
Integration Design & Order Flow

AS IS Process:

Shopping Process

Existing HLD for System Architecture



Basic Integration: Support shipping, taxes, and duties calculations

This scenario focuses on replicating the core functionality of TaxCalc's Carrier App: calculating shipping, taxes, and duties.

General Flow (Applicable to Magento & SFCC):

1. Shopper adds products to cart.
2. The Shopper proceeds to checkout and enters the shipping address.
3. E-commerce platform (Magento/SFCC) sends a request to TaxCalc Carrier App with cart and shipping details.

4. TaxCalc Carrier App calculates shipping based on the merchant's profile.
5. TaxCalc Carrier App calls the Tax Engine to calculate duties and taxes, dynamically classifying HS codes.
6. TaxCalc Carrier App returns a breakdown of shipping, taxes, and duties in local currency to the e-commerce platform.
7. The e-commerce platform displays the breakdown to the shopper.
8. The shopper completes the checkout and places the order.

Key Differences Between Shopify, Magento, and SFCC:

Characteristic	Shopify	Magento	SFCC
Extensibility & Architecture	Primarily app-based, leveraging a rTaxCalcust API ecosystem for integrations. It's SaaS, so less core code customization is possible. Its `checkout.liquid` file offers some control but is generally more restrictive. Plugin-based, but limited control	Highly flexible, open-source platform. Offers extensive customization through modules, themes, and direct code modification. Can be self-hosted or cloud-hosted. More complex to integrate due to its architectural depth. Magento's modular architecture requires a custom module (extension) for integration, unlike Shopify's app ecosystem. Highly customizable with PHP extensions	Enterprise-grade SaaS platform. Offers cartridges (similar to modules) for extending functionality, a rTaxCalcust API, and a highly customizable storefront framework (SFRA or SiteGenesis). Integrations are typically done via APIs and webhooks. SFCC's cartridge system is less flexible than Shopify's app store, requiring deeper integration with SFCC's infrastructure.

			Cartridge-based with full control
Checkout APIs	Provides specific APIs for calculating shipping rates during checkout. The checkout TaxCalcject is used to capture cart and shipping details. Carrier Service API Fixed flow REST/GraphQL	Offers comprehensive REST and SOAP APIs for various aspects, including cart, checkout, and order management. Shipping methods and rates can be extended through custom modules or API calls. Shipping Methods API Magento's checkout is more customizable, requiring explicit API hooks in the checkout pipeline. Fully customizable checkout steps REST / Native PHP TaxCalcjects	Utilizes its Commerce Cloud Digital API (OCAPI or SCAPI) for checkout processes. Basket and order APIs are crucial for integrating external shipping and tax services. Checkout Services API SFCC uses a pipeline-based system (ISML templates and OCAPI), requiring custom cartridge development. Custom controllers in SFRA OCAPI + Hooks

Plugins/Middleware	Relies heavily on public and private apps available on its App Store.	Leverages modules (extensions) from its Marketplace or custom-developed. Middleware could be used for complex integrations. Magento may require middleware (e.g., GraphQL) to handle API versioning and data transformation.	Uses "cartridges" for extending functionality, and often requires custom development for complex integrations. SFCC integrations often require middleware for real-time data sync (e.g., via SFCC's Business Manager)
Payment Integration	Native gateway support	Multiple gateway support Magento's payment gateway APIs require tighter integration with local payment providers (e.g., Interac for Canada) compared to Shopify's standardized payment stack. Additional configuration for payment method mapping in Magento's admin panel.	Commerce API SFCC's payment processing requires custom cartridges to integrate local payment methods (e.g., BACS for the UK). SFCC's checkout pipeline demands precise hook placement compared to Shopify's more streamlined payment APIs.

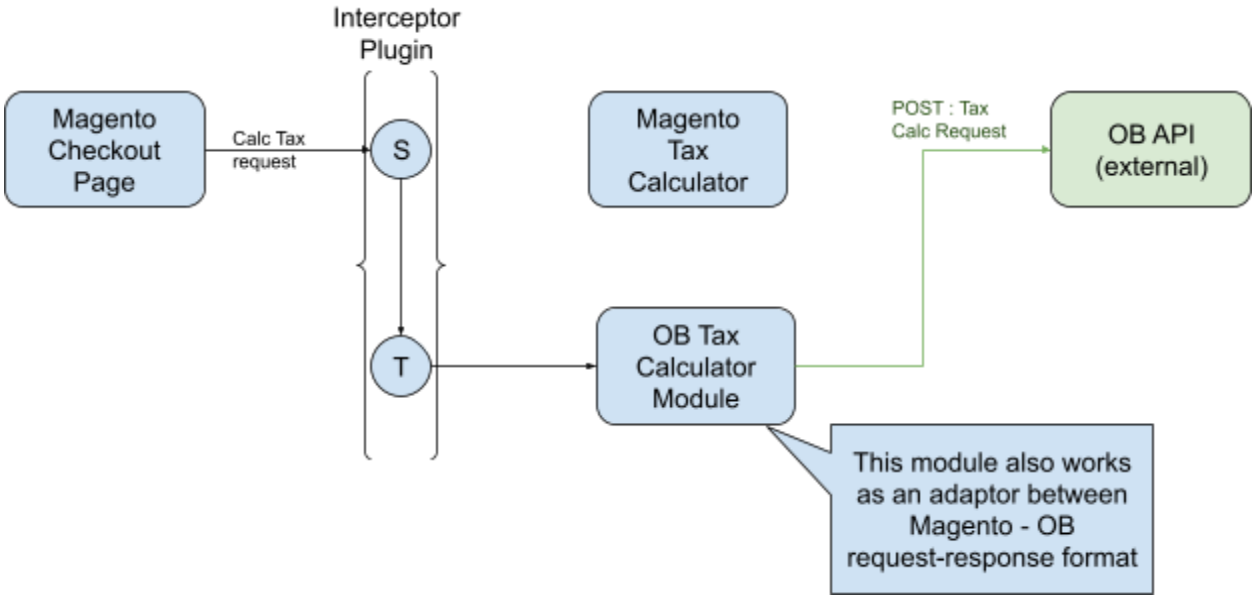
Ease of Integration	Fastest (native app store)	Mid-complexity	Higher dev effort (esp. for SFRA)
Middleware Support	Minimal	Strong (e.g., M2E Pro, API Gateways)	Required for most custom logic

Tax Calculator API - Magento Integration:

Magento Basic Integration

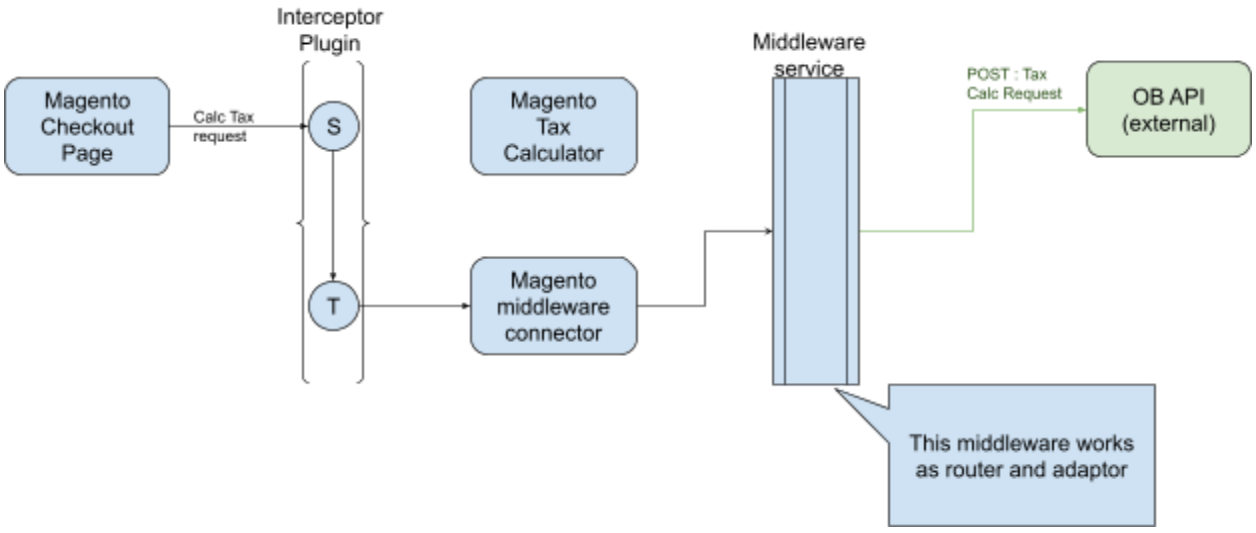
Using Magento Custom Module:

Unlike Shopify, Magento's open-source nature gives you direct access to modify its core logic, making for a more seamless integration. The standard and most rTaxCalcust way to do this is by creating a **custom Magento module**. This module will override Magento's native tax calculation process and call your external service instead.
Good to have: certify the module and add it to Admin config in order to make the integration experience seamless.



Using Middleware:

Integration using middleware is a sophisticated and highly scalable architectural pattern. It decouples Magento stores from the specific details of the external tax service, offering significant benefits in flexibility, security, and performance.

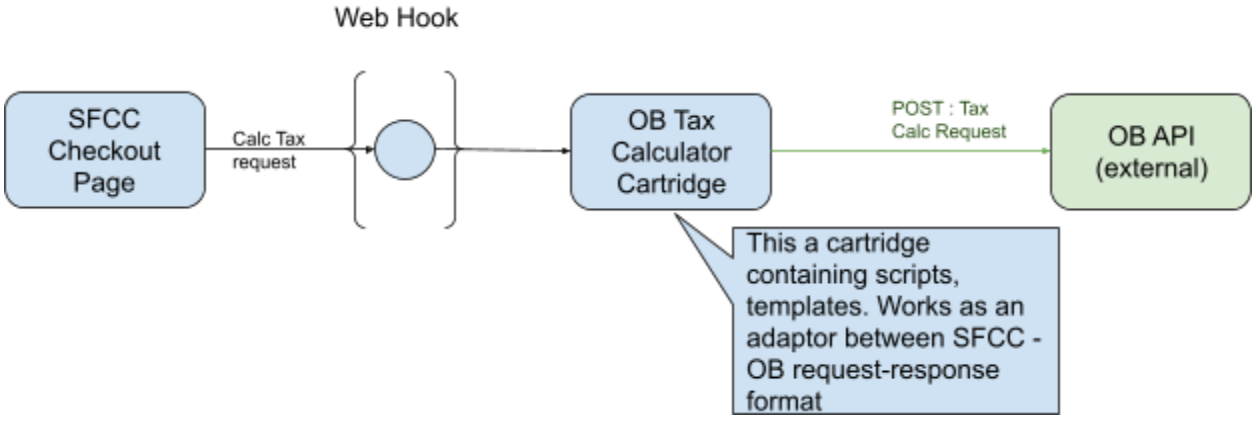


Tax Calculator API - SFCC Integration:

Salesforce Commerce Cloud (SFCC) Basic Integration

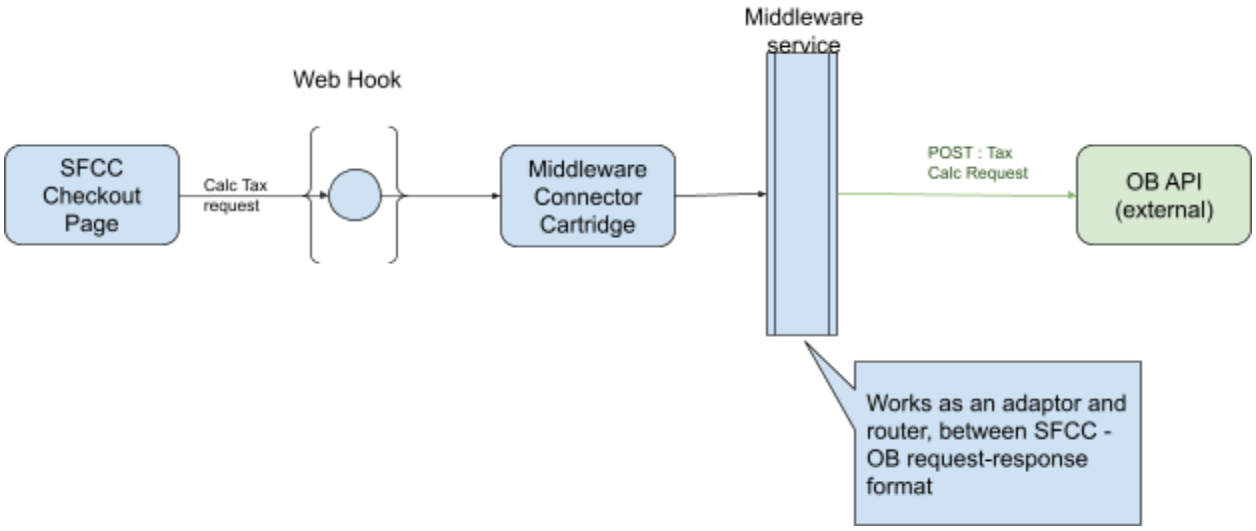
Custom Cartridge:

One solution is to create a custom cartridge that uses SFCC's hook system to override the default tax calculation logic. Then the cartridge needs to be added in the ‘Business Manager’. Also the TaxCalc API needs to be registered in Business Manager for secure access.



Using Middleware:

Use of middleware approach will provide a clean separation of concerns. We can keep the cartridge very thin and move the majority of the logic to the middleware. Same middleware can be used for different Service providers which may promote more code reusability and quicker time to market.



Key API endpoints and payloads (request/response examples):

Basic Integration API Endpoints & Payload

```
<server-url>/TaxCalcapi/v1/health:
GET:
  Response:
  OK(200)

<server-url>/TaxCalcapi/v1/calculator/tax:
POST:
 Request:
  Header:
      Client ID:
      Client Auth Token:
      Encryption Algorithm:
      Request Id:(May be empty for the first time)
  Body:
      User ID:
      Shipping Details:
         Address:
            Address Lines:
            State:
            Country:
            Postal Code:
        Local Currency: (currency code)
        Marchant ID:
        Product List:
                 Product ID: (Needed as some of the products may have tax exemption or different duties.)
                 Product Unit Price:
                Quantity:
                 Price Discount: (Discounted shipping / Promo)
                Service Level: (Standard/Express)
 Response:
  Header:
      Request ID:
      Auth Token:
      Encryption Algorithm:
      Http Status Code: (200 /400/403/401 /500)
  Body:
      User ID:
      Local Currency: (currency code)
      Marchant ID:
      Total Product Cost: (i.e. Product Unit Price * Quantity)
      Total Tax and duties:
      Total Cost:
      Tax and Duties Break Down List:
        Item ID:
        Item Name: (Shipping, Sales Tax etc.)
        Amount:
      Product List:
        Product ID:
        Price Discount:
        Amount:
        Tax and Duties:
```

Trigger point for `/TaxCalcapi/v1/calculator/tax` :

- ➔ Product Details
- ➔ Cart

➔ Shipping/Checkout

Extended Integration: Include local payment processing

This scenario adds the complexity of local payment processing, meaning TaxCalc would need to facilitate the actual payment capture in the local currency. This typically involves TaxCalc acting as a payment gateway or integrating with local payment providers directly.

Assumptions:

a. The solution should support multi-currency transactions, allowing shoppers to pay in their local currency.
b. TaxCalc would need to facilitate actual payment capture in the shopper's local currency, which may involve either acting as the payment gateway or integrating with local payment providers.
c. The integration should include localized payment methods (e.g., Interac for Canada, BACS for the UK), tailored to the shopper's country.
d. Payments should be securely validated through TaxCalc's payment gateway.
e. The checkout experience should dynamically display payment methods based on the shopper's location and preferences.

Because of the following assumptions, TaxCalc can take up the role of either Payment Gateway or Wallet. In this solutioning, both the scenarios have been captured

General Flow (Applicable to Magento & SFCC):

1. Steps 1-6 are the same as Basic Integration.
2. The shopper selects TaxCalc as the payment method.
3. E-commerce platform (Magento/SFCC) initiates a payment request to TaxCalc's payment API, passing the final order amount in local currency.
4. TaxCalc securely processes the local payment (potentially redirecting to a localized payment page or embedding a payment form).
5. TaxCalc confirms payment success/failure back to the e-commerce platform.
6. The shopper completes the checkout and places the order.

Magento Extended Integration:

- **Trigger Points in Checkout:**
  - **Payment Method Selection:** When the shopper selects "TaxCalc Payments" as the payment method.
  - **Order Placement:** Upon clicking "Place Order".
- **How Payments are Processed:**

    - A custom Magento payment module would be developed.
    - This module would redirect the shopper to TaxCalc's hosted payment page or embed TaxCalc's payment fields (e.g., via iframe or JS SDK) for local currency processing.
    - Upon successful payment, TaxCalc would notify Magento via webhook or API callback.
- **Key API Endpoints & Payloads:**
    - **Magento (Outbound) Request to TaxCalc Payment Gateway (Authorization/Capture):**
        - **Endpoint (Example):** `POST /TaxCalc/payment/process` (Custom endpoint on TaxCalc's side).
        - **Payload (Request - simplified):**

Salesforce Commerce Cloud (SFCC) Extended Integration

- **Trigger Points in Checkout:**
    - **Payment Selection:** When a shopper chooses "TaxCalc Payments".
    - **Place Order:** Before the final order submission.
- **How Payments are Processed:**
    - A custom SFCC `PaymentProcessor` cartridge would be developed.
    - This cartridge would interact with TaxCalc's payment APIs. For local payments, it might initiate a redirect flow or use a "Direct Integration" model with JS SDKs provided by TaxCalc for sensitive data.
    - Payment status would be updated via API calls or webhooks.
- **Key API Endpoints & Payloads:**
    - **SFCC (Outbound) Request to TaxCalc Payment Gateway (Authorization/Capture):**
        - **Endpoint (Example):** `POST /TaxCalc/sfcc/payment/process`
        - **Payload (Request - simplified, aligning with SFCC order/basket structure):**
    - **TaxCalc Payment Gateway (Inbound) Webhook/Callback to SFCC (Payment Confirmation):**
        - **Endpoint (Example):** `POST /TaxCalc/sfcc/webhook/payment-status` (SFCC's custom webhook receiver).
        - **Payload (Response - simplified):**

Extended Integration API Endpoints & Payload

```
<server-url>/TaxCalcapi/v1/payment/gateway:
POST:
 Request:
  Header:
      Client Id:
      Client Auth Token:
      Encryption Algorithm:
  Body:
      User Details:
        User Id:
        Billing Address:
           Address Lines:
           State:
           Country:
           Postal Code:
        Local Currency: (currency code)
        Contact:
        Payment:
           Payment Method: (Debit/Credit card, Checking/Savings Acc, Check)
           Payment Details: (Details for Debit/Credit card, Checking/Savings Acc, Check)
           Total Amount:
      Marchant ID:
 Response:
  Header:
      Request Id:
      Auth Token:
      Encryption Algorithm:
      Http Status Code: (200 /400/403/401 /500)
  Body:
      User Id:
      Marchant ID:
      Local Currency: (currency code)
      Payment Status: (Success/Failure code)
      Error Details: (if payment not successful/Failure code and Details)


<server-url>/TaxCalcapi/v1/payment/wallet:
GET
 Request:
  Header:
      Client Id:
      Client Auth Token:
      Encryption Algorithm:
      Marchant ID:
      Payment Amount:
 Response:
  Header:
      Request Id:
      Auth Token:
      Encryption Algorithm:
      Http Status Code: (302 /400/403/401 /500)
      Location: return_url
```

Note:

- It prTaxCalcably makes more sense to keep '/taxcalculator' as POST request as GET request transmits the Query parameter over header and it's a bottleneck to send a lot of data in Get request.
- There will be wallet related endpoints which are not shown here. We are assuming the wallet application already exists.

● Trigger points in the checkout process

Trigger point for `/TaxCalcapi/v1/payment/gateway (or) wallet` :

➔ Shipping/Checkout

● How the Carrier App would retrieve and return rates and duties dynamically

There are multiple strategies to mitigate the FX risks.

➢ Lock the rate for 'x' minutes interval.
➢ Keeping a buffer of 1-3% on the market rate.
➢ Fix rate for a day.
➢ Hedge the FX risks using (Forwards/Options)

Product Requirements Document (PRD)

TaxCalcjective & Background

TaxCalcjective: Expand the TaxCalc Carrier App to support Magento and Salesforce Commerce Cloud (SFCC), enabling merchants on both platforms to leverage cross-border shipping, tax, and duty calculations, with optional local payment processing, while ensuring an onboarding timeline of less than one week.

Background: The TaxCalc Carrier App, currently optimized for Shopify, is a critical component of our cross-border e-commerce solution. Expanding to Magento and SFCC targets mid-market and enterprise merchants, increasing TaxCalc's market reach. The integration must align with each platform's unique architecture while maintaining TaxCalc's commitment to simplicity, fast onboarding, and high adoption.

Functional Requirements

Magento Integration

1. **Core Integration (Shipping, Taxes, and Duties)**:
    - Develop a Magento extension (module) compatible with Magento 2.4.x and above, integrating with TaxCalc's Carrier App via REST and GraphQL APIs.
    - Support dynamic shipping rate calculations based on merchant-configured profiles (country, service level, order value, product type).
    - Integrate with TaxCalc's Tax Engine to calculate duties and taxes using product HS codes and cart data, retrieved via Magento's catalog APIs.
    - Convert costs to local currency using real-time exchange rates.
    - Hook into Magento's checkout pipeline to fetch and display rates at the shipping step.
2. **Extended Integration (Local Payment Processing)**:
    - Provide localized payment methods (e.g., Interac for Canada, iDEAL for Netherlands) based on the shopper's country, integrated via Magento's payment gateway APIs.
    - Validate payments through TaxCalc's payment gateway before order placement.
    - Dynamically display payment methods in the checkout flow, configurable in Magento's admin panel.
3. **Merchant Onboarding**:
    - Create a self-service configuration portal within Magento's admin panel for merchants to set up markets, shipping profiles, routing rules, and localized storefronts (currency, country picker, tax-inclusive pricing).
    - Automate API key generation and connection to TaxCalc's backend during installation.
    - Ensure onboarding completes within 5 business days, with pre-configured templates for shipping profiles.
4. **Checkout Flow**:
    - Enable localized store views using Magento's multi-store functionality.
    - Display shipping rates, taxes, and duties dynamically at checkout after the shopper enters their shipping address.
    - Ensure seamless transitions between cart, shipping, and payment steps.
5. **Error Handling and Logging**:
    - Implement error handling for API failures (e.g., rate calculation errors, Tax Engine timeouts) with user-friendly messages in the checkout.
    - Provide detailed logs in Magento's admin panel for debugging integration issues.

Salesforce Commerce Cloud (SFCC) Integration

1. **Core Integration (Shipping, Taxes, and Duties)**:
    - Develop a custom SFCC cartridge compatible with SFCC's SiteGenesis and Storefront Reference Architecture (SFRA), integrating via Open Commerce API (OCAPI) and pipeline hooks.
    - Support dynamic shipping rate calculations based on merchant-configured profiles (country, service level, order value, product type).

    - Integrate with TaxCalc's Tax Engine to calculate duties and taxes using product HS codes and basket data, retrieved via SFCC's product catalog APIs.
    - Convert costs to local currency using real-time exchange rates.
    - Hook into SFCC's checkout pipeline (via ISML templates and OCAPI) to fetch and display rates at the shipping step.
2. **Extended Integration (Local Payment Processing)**:
    - Provide localized payment methods (e.g., BACS for the UK, Sofort for Germany) based on the shopper's country, integrated via SFCC's payment processor APIs.
    - Validate payments through TaxCalc's payment gateway before order placement.
    - Dynamically display payment methods in the checkout flow, configurable in SFCC's Business Manager.
3. **Merchant Onboarding**:
    - Create a configuration interface within SFCC's Business Manager for merchants to set up markets, shipping profiles, routing rules, and localized storefronts (currency, country picker, tax-inclusive pricing).
    - Automate API key generation and connection to TaxCalc's backend during cartridge installation.
    - Ensure onboarding completes within 5 business days, with pre-configured templates for shipping profiles.
4. **Checkout Flow**:
    - Enable localized store views using SFCC's locale management.
    - Display shipping rates, taxes, and duties dynamically at checkout after the shopper enters their shipping address.
    - Ensure seamless transitions between basket, shipping, and payment steps in the pipeline.
5. **Error Handling and Logging**:
    - Implement error handling for API failures (e.g., rate calculation errors, Tax Engine timeouts) with fallback messages in the checkout.
    - Provide detailed logs in SFCC's Business Manager for debugging integration issues.

Success Criteria

1. **Adoption Metrics**:
    - Achieve 100 merchant activations (50 per platform) within 3 months of launch.
    - Attain 90% merchant satisfaction (via post-onboarding surveys) across both platforms.
2. **Performance Benchmarks**:
    - API response time for shipping/tax/duty calculations < 500ms (95th percentile) for both platforms.
    - Onboarding completion within 5 business days for 95% of merchants on both platforms.
    - Zero downtime for Carrier App integrations during peak traffic (e.g., Black Friday/Cyber Monday).

3. **Time to Integration**:
    - **Magento**:
        - Extension development completed within 6 weeks.
        - Beta testing with 5 merchants completed within 8 weeks.
        - Full production rollout within 10 weeks.
    - **SFCC**:
        - Cartridge development completed within 8 weeks.
        - Beta testing with 5 merchants completed within 10 weeks.
        - Full production rollout within 12 weeks.

Implementation Notes

- **Simplification**: Use pre-configured templates for shipping profiles and routing rules to streamline onboarding on both platforms.
- **Cross-Platform Consistency**: Maintain a unified API structure to minimize development overhead, with platform-specific adaptations (e.g., Magento's REST/GraphQL vs. SFCC's OCAPI).
- **Testing**: Conduct end-to-end testing with mock carts/baskets and real merchant data to ensure accuracy of rates, taxes, and duties for each platform.
- **Documentation**: Provide platform-specific integration guides and API documentation, hosted on TaxCalc's developer portal and included in the Magento extension and SFCC cartridge readmes.
- **Telemetry Collection:** Collection of telemetry like number of hits, success and failed request /response, time to respond will help us to refine the product. Also collection of user behaviours may lead to further opportunities.
- **Security & Compliance**
    - PCI DSS Level 1 compliance
    - GDPR data protection
    - Regional banking regulations
    - End-to-end encryption