LoyalNest App Features - Should Have

Overview

This document outlines **Should Have** features for the LoyalNest Shopify app, targeting medium to Shopify Plus merchants (5,000–50,000 customers) in Phase 3. It enhances the MVP from `features_1_must_have.md`, aligning with user stories (US-CW1–CW15, US-MD1–MD18, US-AM1–AM13, US-BI1–BI5), wireframes, and LoyalNest App Feature Analysis. The app uses microservices (`rfm-service`, `users-service`, `roles-service`, `Points`, `Referrals`, `Analytics`, `AdminCore`, `AdminFeatures`, `Frontend`) with NestJS/TypeORM, gRPC, Rust/Wasm Shopify Functions, PostgreSQL partitioning, Redis Streams, Bull queues, Kafka, and monitoring via Prometheus/Grafana, Sentry, Loki, and PostHog. It supports 10,000 orders/hour, Shopify Plus API limits (40 req/s), GDPR/CCPA compliance (AES-256 encryption, 90-day Backblaze B2 backups), multilingual support (`en`, `es`, `fr`, `de`, `pt`, `ja`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)), and WCAG 2.1 compliance (Lighthouse CI: 90+). The implementation aligns with a 39.5-week TVP timeline (ending February 17, 2026) and $97,012.50 budget, using Docker Compose (`artifact_id: 16fc7997-8a42-433e-a159-d8dad32a231f`) and SQL schemas (`artifact_id: 6f83061c-0a09-404f-8ca1-81a7aa15c25e`).

SHOULD HAVE FEATURES

1. Points Program (Phase 3)

- **Goal**: Expand earning and redemption options. Success metric: 90%+ adoption of new actions, 85%+ redemption rate, 15%+ multi-store sync adoption (Phase 5), 90%+ multi-currency adoption.
- **Earning Actions**: Social follows (50 points), goal spend ($100 for 200 points), referrals (50 points), merchant referrals (Phase 5, e.g., 500 points).
- **Dynamic Point Multipliers**: Real-time multipliers (e.g., 2x for first purchase within 24 hours, 1.5x for Champions via `rfm-service /rfm.v1/RFMService/GetSegmentCounts`) calculated using Rust/Wasm Shopify Functions, logged in `points_transactions`.
- **Redemption Options**: Cashback, custom incentives (e.g., exclusive products) via GraphQL Admin API (40 req/s for Plus). Supports multi-store point sharing (Phase 5, US-CW6).
- **Multi-Currency Support**: Supports multi-currency discounts (e.g., USD, EUR, CAD) via Shopify's multi-currency API, applied at checkout using Shopify Checkout UI Extensions (Rust/Wasm). Stores currency in `points_transactions.currency`.
- **Customization**: Fully customizable rewards page, no-code Sticky Bar (US-CW14), post-purchase widget (US-CW15), advanced branding with Polaris-compliant UI, i18next (`en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)), and Tailwind CSS.
- **Scalability**: Supports 50,000+ customers via Redis Streams (`points:{customer_id}`, `multiplier:{customer_id}`, `currency:{customer_id}`), Bull queues, PostgreSQL partitioning, circuit breakers, and Chaos Mesh testing for Black Friday surges (10,000 orders/hour).
- **Database Design**:
  - **Table**: `users` (users-service)
    - `id` (text, PK, NOT NULL): Unique ID.
    - `email` (text, AES-256 ENCRYPTED, NOT NULL): Customer email.
    - `rfm_score` (jsonb, AES-256 ENCRYPTED): e.g., `{"recency": 5, "frequency": 3, "monetary": 4, "score": 4.2}`.
    - `churn_score` (numeric(10,2), CHECK BETWEEN 0 AND 1): Churn probability.

        - `lifecycle_stage` (text, CHECK IN ('new_lead', 'repeat_buyer', 'churned', 'vip')): Lifecycle stage.
    - **Table**: `program_settings`
        - `merchant_id` (text, PK, FK → `merchants`, NOT NULL): Merchant.
        - `dynamic_multipliers` (jsonb): e.g., `{"first_purchase_24h": 2.0, "rfm_champion": 1.5}`.
        - `multi_store_config` (jsonb, Phase 5): e.g., `{"shared_stores": ["store1.myshopify.com", "store2.myshopify.com"]}`.
        - `multi_currency_config` (jsonb): e.g., `{"supported_currencies": ["USD", "EUR", "CAD"]}`.
    - **Table**: `points_transactions` (partitioned by `merchant_id`)
        - `transaction_id` (text, PK, NOT NULL): Unique ID.
        - `customer_id` (text, FK → `users`, NOT NULL): Customer.
        - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
        - `type` (text, CHECK IN ('earn', 'redeem', 'expire', 'adjust', 'import', 'referral', 'campaign')): Action type.
        - `points` (integer, CHECK >= 0): Points awarded.
        - `currency` (text): e.g., `USD`, `EUR`, `CAD`.
        - `source` (text): Source (e.g., "order", "rfm_reward").
        - `order_id` (text): Shopify order ID.
        - `expiry_date` (timestamp(3)): Expiry timestamp.
        - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `multiplier_applied`, `multi_store_sync`, `multi_currency_applied`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"multiplier": 2.0, "action": "first_purchase", "currency": "USD"}`.
    - **Indexes**: `idx_users_email` (btree: `email`), `idx_program_settings_merchant_id` (btree: `merchant_id`), `idx_points_transactions_customer_id` (btree: `customer_id`, `created_at`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **PUT** `/v1/api/points-program` (REST) | gRPC `/admin.v1/AdminService/UpdatePointsProgram`
        - **Input**: `{ merchant_id: string, config: { purchase: number, signup: number, social_follow: number, referral: number }, dynamic_multipliers: object, branding: { points_currency_singular: string }, multi_store_config: object, multi_currency_config: { supported_currencies: array }, locale: string }`
        - **Output**: `{ status: string, error: { code: string, message: string } | null }`
        - **Flow**: Update `program_settings`, cache in Redis Streams (`program:{merchant_id}`, `multiplier:{merchant_id}`, `currency:{merchant_id}`), log in `audit_logs`, track via PostHog (`points_config_updated`, 80%+ usage, `multi_currency_configured`, 90%+ adoption).

  - **POST** `/v1/api/points/calculate` (REST) | gRPC `/points.v1/PointsService/CalculateDynamicPoints`
    - **Input**: `{ customer_id: string, action_type: string, order_id: string, currency: string, locale: string }`
    - **Output**: `{ status: string, points: number, multiplier: number, currency: string, error: { code: string, message: string } | null }`
    - **Flow**: Fetch RFM segment via `/rfm.v1/RFMService/GetSegmentCounts`, calculate multiplier via Rust/Wasm, validate currency via Shopify's multi-currency API, cache in Redis Streams (`multiplier:{customer_id}`, `currency:{customer_id}`), log in `audit_logs`, track via PostHog (`multiplier_applied`, 90%+ adoption, `multi_currency_applied`, 90%+ adoption).
  - **POST** `/v1/api/points/sync` (Phase 5) | gRPC `/points.v1/PointsService/SyncPoints`
    - **Input**: `{ customer_id: string, store_ids: array, locale: string }`
    - **Output**: `{ status: string, points_balance: number, error: { code: string, message: string } | null }`
    - **Flow**: Sync `points_balance` across stores, update `points_transactions`, cache in Redis Streams (`points:{customer_id}`), log in `audit_logs`, track via PostHog (`multi_store_sync`, 15%+ adoption).
- **GraphQL Query Examples**:
  - **Query: Fetch Multi-Currency Order Details**
    - **Purpose**: Retrieves order details with currency information to apply multi-currency discounts, used in `/v1/api/points/calculate`.
    - **Query**:

```
query GetOrderDetails($id: ID!) {
  order(id: $id) {
    id
    totalPriceSet {
      shopMoney {
        amount
        currencyCode
      }
      presentmentMoney {
        amount
        currencyCode
      }
    }
    customer {
      id
    }
    createdAt
  }
}
```

    - **Variables**: `{ "id": "gid://shopify/Order/123456789" }`
    - **Use Case**: Merchant Dashboard validates order currency for points calculation, storing `currency` in `points_transactions` for multi-currency support.

- **Query: Create Cashback Reward**
  - **Purpose**: Creates a cashback discount for point redemption, used in `/v1/api/rewards/redeem`.
  - **Query**:

```
mutation CreateDiscount($input: DiscountCodeBasicInput!) {
  discountCodeBasicCreate(basicCodeDiscount: $input) {
    codeDiscountNode {
      id
      codeDiscount {
        ... on DiscountCodeBasic {
          title
          codes(first: 1) {
            nodes {
              code
            }
          }
        }
      }
    }
    userErrors {
      field
      message
    }
  }
}
```

  - **Variables**:

```
{
  "input": {
    "title": "LoyalNest $10 Cashback",
    "code": "LN10CASH",
    "customerGets": {
      "value": {
        "discountAmount": {
          "amount": 10.0,
          "currencyCode": "USD"
        }
      }
    },
    "appliesOncePerCustomer": true
  }
}
```

  - **Use Case**: Merchant Dashboard issues cashback rewards via Shopify Checkout UI Extensions, stored in `reward_redemptions` (AES-256 encrypted).

- **Service**: Points Service (gRPC: `/points.v1/*`, Dockerized).

2. Referral Program (Phase 3)

- **Goal**: Enhance referral engagement. Success metric: 10%+ referral conversion rate (SMS), 5%+ (email), 85%+ dashboard interaction rate.
- **Multi-Tier Referrals**: Support tiered rewards (e.g., 100 points for 1st referral, 200 for 2nd) via `referrals.tier_level`, configurable in Merchant Dashboard.
- **Custom Referral Incentives**: Custom rewards (e.g., exclusive products) via GraphQL Admin API, integrated with Klaviyo/Postscript for notifications (3 retries, `en`, `es`, `fr`, `ar`(RTL)).
- **Advanced Analytics**: Track multi-tier referral performance, A/B test incentives via `referral_events.variants` (JSONB), visualized in Polaris `DataTable` and Chart.js (CTR, conversion rate).
- **Scalability**: Supports 50,000+ customers with Redis Streams (`referral:{referral_code}`, `multi_tier:{referral_code}`), Bull queues, PostgreSQL partitioning, and Chaos Mesh testing.
- **Database Design**:
  - **Table**: `referral_links` (partitioned by `merchant_id`)
    - `referral_link_id` (text, PK, NOT NULL): Unique ID.
    - `advocate_customer_id` (text, FK → `users`, NOT NULL): Advocate.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `referral_code` (text, UNIQUE, NOT NULL): Unique code.
    - `click_count` (integer, DEFAULT 0): Number of clicks.
    - `merchant_referral_id` (text, FK → `merchant_referrals` | NULL): Merchant referral (Phase 5).
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `referrals` (partitioned by `merchant_id`)
    - `referral_id` (text, PK, NOT NULL): Unique ID.
    - `advocate_customer_id` (text, FK → `users`, NOT NULL): Advocate.
    - `friend_customer_id` (text, FK → `users`, NOT NULL): Friend.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `reward_id` (text, FK → `rewards`): Reward.
    - `tier_level` (integer, DEFAULT 1): Referral tier.
    - `status` (text, CHECK IN ('pending', 'completed', 'expired')): Status.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `referral_events` (partitioned by `merchant_id`)
    - `event_id` (text, PK, NOT NULL): Event ID.
    - `referral_link_id` (text, FK → `referral_links`, NOT NULL): Link.
    - `action` (text, CHECK IN ('click', 'conversion', 'view')): Action type.
    - `variants` (jsonb): e.g., `{"incentive": "exclusive_product"}`.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `multi_tier_referral_created`, `referral_incentive_updated`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"tier_level": 2, "incentive": "exclusive_product"}`.
  - **Indexes**: `idx_referral_links_referral_code` (btree: `referral_code`), `idx_referrals_merchant_id` (btree: `merchant_id`), `idx_referral_events_referral_link_id` (btree: `referral_link_id`), `idx_audit_logs_action` (btree: `action`).

  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **PUT** `/v1/api/referrals/config` (REST) | gRPC `/referrals.v1/ReferralService/UpdateReferralConfig`
    - **Input**: `{ merchant_id: string, tiers: array, incentives: object, locale: string }`
    - **Output**: `{ status: string, error: { code: string, message: string } | null }`
    - **Flow**: Update `program_settings.referral_config`, cache in Redis Streams (`referral_config:{merchant_id}`), log in `audit_logs`, track via PostHog (`referral_config_updated`, 85%+ usage).
  - **POST** `/v1/api/referrals/tier` (REST) | gRPC `/referrals.v1/ReferralService/AssignTierReward`
    - **Input**: `{ referral_id: string, tier_level: number, locale: string }`
    - **Output**: `{ status: string, reward_id: string, error: { code: string, message: string } | null }`
    - **Flow**: Update `referrals.tier_level`, award points via `/points.v1/PointsService/EarnPoints`, notify via Klaviyo/Postscript, cache in Redis Streams (`multi_tier:{referral_code}`), log in `audit_logs`, track via PostHog (`multi_tier_referral_created`, 10%+ conversion).
- **GraphQL Query Examples**:
  - **Query: Create Multi-Tier Referral Reward**
    - **Purpose**: Creates a custom reward for a multi-tier referral, used in `/v1/api/referrals/tier`.
    - **Query**:

```
mutation CreateDiscount($input: DiscountCodeBasicInput!) {
  discountCodeBasicCreate(basicCodeDiscount: $input) {
    codeDiscountNode {
      id
      codeDiscount {
        ... on DiscountCodeBasic {
          title
          codes(first: 1) {
            nodes {
              code
            }
          }
        }
      }
    }
    userErrors {
      field
      message
    }
  }
}
```

        - **Variables**:

          ```
          {
            "input": {
              "title": "Referral Tier 2 Reward",
              "code": "REF2TIER",
              "customerGets": {
                "value": {
                  "percentage": 0.15
                }
              },
              "appliesOncePerCustomer": true
            }
          }
          ```

        - **Use Case**: Merchant Dashboard issues tiered referral rewards (e.g., 15% off for 2nd referral), stored in `referrals.reward_id`.
    - **Query: Track Referral Analytics**
        - **Purpose**: Retrieves referral event data for advanced analytics, used in `/v1/api/referrals/analytics`.
        - **Query**:

          ```
          query GetReferralAnalytics($merchantId: String!) {
            orders(first: 100, query: $merchantId) {
              edges {
                node {
                  id
                  customer {
                    id
                    metafield(namespace: "loyalnest", key: "referral_code")
          {
                      value
                    }
                  }
                }
              }
            }
          }
          ```

        - **Variables**: `{ "merchantId": "merchant_123" }`
        - **Use Case**: Merchant Dashboard tracks referral conversions, feeding into `referral_events` for Chart.js visualization.
- **Service**: Referrals Service (gRPC: `/referrals.v1/*`, Dockerized).

3. VIP Tiers (Phase 3)

- **Goal**: Reward loyal customers with tiered benefits and ensure accurate configuration. Success metric: 15%+ tier progression rate, 90%+ adoption of preview mode.

- **Features**: Tiered rewards (Bronze, Silver, Gold) based on points, spend, or RFM segments via `rfm-service` (`/rfm.v1/RFMService/GetSegmentCounts`), configured via no-code dashboard (US-MD2). Displays progress in Customer Widget (US-CW7) with Polaris `ProgressBar`. Supports Shopify Flow triggers (e.g., "Tier Upgraded → Notify Customer").
- **VIP Preview Mode**: Simulates tier assignments for sample customers using `users.rfm_score`, visualized with Chart.js (bar type) via `/rfm.v1/RFMService/GetRFMVisualizations`.
- **Scalability**: Handles 50,000+ customers with Redis Streams (`tiers:{customer_id}`, `rfm:preview:{merchant_id}`), Bull queues, PostgreSQL partitioning, and circuit breakers.
- **Database Design**:
  - **Table**: `users` (users-service)
    - `id` (text, PK, NOT NULL): Unique ID.
    - `rfm_score` (jsonb, AES-256 ENCRYPTED): e.g., `{"recency": 5, "frequency": 3, "monetary": 4, "score": 4.2}`.
  - **Table**: `vip_tiers` (partitioned by `merchant_id`)
    - `tier_id` (text, PK, NOT NULL): Unique ID.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `name` (jsonb, CHECK ?| ARRAY['en', 'es', 'de', 'ja', 'fr', 'pt', 'ru', 'it', 'nl', 'pl', 'tr', 'fa', 'zh-CN', 'vi', 'id', 'cs', 'ar', 'ko', 'uk', 'hu', 'sv', 'he']): e.g., `{"en": "Gold", "ar": "..."}`.
    - `rules` (jsonb): e.g., `{"points": ">=1000", "rfm_score": ">=4"}`.
    - `benefits` (jsonb): e.g., `{"discount": "10%", "free_shipping": true}`.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `tier_assigned`, `preview_simulated`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"tier_id": "gold", "customer_count": 50}`.
  - **Indexes**: `idx_users_rfm_score` (gin: `rfm_score`), `idx_vip_tiers_merchant_id` (btree: `merchant_id`), `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **GET** `/v1/api/vip/preview` (REST) | gRPC `/rfm.v1/RFMService/PreviewVIPTier`
    - **Input**: `{ merchant_id: string, sample_customers: array, locale: string }`
    - **Output**: `{ status: string, preview: [{ customer_id: string, tier_id: string, tier_name: string }], error: { code: string, message: string } | null }`
    - **Flow**: Apply `vip_tiers.rules` to `users.rfm_score`, generate Chart.js data (bar type), cache in Redis Streams (`rfm:preview:{merchant_id}`), log in `audit_logs`, track via PostHog (`preview_simulated`, 90%+ adoption).
- **GraphQL Query Examples**:
  - **Query: Fetch Customer RFM Data for Tier Assignment**
    - **Purpose**: Retrieves customer RFM scores to determine VIP tier eligibility, used in `/rfm.v1/RFMService/GetSegmentCounts`.
    - **Query**:

```
query GetCustomerRFM($ids: [ID!]!) {
  customers(ids: $ids) {
    id
    metafield(namespace: "loyalnest", key: "rfm_score") {
      value
    }
    ordersCount
    totalSpent
  }
}
```

- **Variables**: `{ "ids": ["gid://shopify/Customer/987654321", "gid://shopify/Customer/123456789"] }`
- **Use Case**: Merchant Dashboard applies `vip_tiers.rules` to `users.rfm_score` for tier assignment, visualized in Customer Widget with Polaris `ProgressBar`.

- **Query: Update VIP Tier Configuration**
  - **Purpose**: Updates VIP tier rules and benefits, used in no-code dashboard for `/v1/api/vip/preview`.
  - **Query**:

```
mutation UpdateMetafield($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
  }
}
```

  - **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "vip_tiers",
    "value": "{\"tiers\": [{\"id\": \"gold\", \"name\": {\"en\": \"Gold\"}, \"rules\": {\"points\": \">=1000\", \"rfm_score\": \">=4\"}, \"benefits\": {\"discount\": \"10%\", \"free_shipping\": true}}]}",
    "ownerId": "gid://shopify/Shop/123456789",
    "type": "json"
  }
}
```

    - **Use Case**: No-code dashboard updates `vip_tiers` table with tier rules and benefits, cached in Redis Streams (`rfm:preview:{merchant_id}`) for preview mode.
  - **Query: Create Shopify Flow Trigger for Tier Upgrade**
    - **Purpose**: Sets up a Shopify Flow trigger for tier upgrades, used in `/rfm.v1/RFMService/NotifyTierUpgrade`.
    - **Query**:

```
mutation CreateWebhook($input: WebhookSubscriptionInput!) {
  webhookSubscriptionCreate(input: $input) {
    webhookSubscription {
      id
      topic
      endpoint {
        ... on WebhookHttpEndpoint {
          callbackUrl
        }
      }
    }
    userErrors {
      field
      message
    }
  }
}
```

    - **Variables**:

```
{
  "input": {
    "topic": "CUSTOMERS_UPDATE",
    "endpoint": {
      "callbackUrl": "https://loyalnest.com/webhooks/tier-upgrade"
    }
  }
}
```

    - **Use Case**: Admin Module configures Shopify Flow to trigger notifications (via Klaviyo/Postscript) when customers upgrade tiers, logged in `audit_logs` as `tier_assigned`.
- **Service**: RFM Service (gRPC: `/rfm.v1/*`, Dockerized).

4. On-Site Content (Phase 3)

- **Goal**: Improve engagement with advanced widgets. Success metric: 90%+ widget interaction rate, 15%+ nudge conversion rate, Lighthouse CI score 95+.
- **Advanced Widgets**: No-code Sticky Bar (US-CW14), post-purchase widget (US-CW15), personalized banners based on RFM segments (`rfm-service`, `nudges`), and Theme App Extensions (Phase 5).

Supports A/B testing via `program_settings.ab_tests` (JSONB).
- **Dynamic Nudges**: Personalized nudges for RFM segments (e.g., "VIP Exclusive Offer" for Champions) via `rfm-service`, delivered through Klaviyo/Postscript/AWS SES (3 retries, `en`, `es`, `fr`, `ar`(RTL)).
- **Accessibility**: Enhanced ARIA labels, keyboard navigation, screen reader support, RTL for `ar`, `he`, WCAG 2.1 compliance, Lighthouse CI scores (95+ for ARIA, LCP, FID, CLS).
- **Scalability**: Renders <1s via Redis Streams (`content:{merchant_id}:{locale}`, `rfm:nudge:{customer_id}`), supports 10,000 orders/hour with circuit breakers and Chaos Mesh testing.
- **Database Design**:
  - **Table**: `program_settings`
    - `merchant_id` (text, PK, FK → `merchants`, NOT NULL): Merchant.
    - `branding` (jsonb, CHECK ?| ARRAY['en', `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`, `ko`, `uk`, `hu`, `sv`, `he`]): e.g., `{"sticky_bar": {"en": {...}, "es": {...}, "ar": {...}}, "post_purchase": {...}}`.
    - `ab_tests` (jsonb): e.g., `{"sticky_bar": {"variant_a": {"color": "blue"}, "variant_b": {"color": "green"}}}`.
  - **Table**: `nudges` (rfm-service, partitioned by `merchant_id`)
    - `nudge_id`, `merchant_id`, `type` (CHECK IN ('at-risk', 'loyal', 'new', 'inactive', 'tier_dropped', 'vip')), `title` (jsonb), `description` (jsonb), `is_enabled`, `variants` (jsonb): Nudge configurations.
  - **Table**: `nudge_events` (rfm-service, partitioned by `merchant_id`)
    - `event_id`, `customer_id`, `nudge_id`, `action` (CHECK IN ('view', 'click', 'dismiss')), `created_at`: Nudge interactions.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `content_updated`, `ab_test_updated`, `rfm_nudge_viewed`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"type": "sticky_bar", "variant": "blue"}`.
  - **Indexes**: `idx_program_settings_merchant_id` (btree: `merchant_id`), `idx_nudges_merchant_id` (btree: `merchant_id`), `idx_nudge_events_customer_id` (btree: `customer_id`), `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **PUT** `/v1/api/content/advanced` (REST) | gRPC `/frontend.v1/FrontendService/UpdateAdvancedContent`
    - **Input**: `{ merchant_id: string, branding: { sticky_bar: object, post_purchase: object, banners: object }, ab_tests: object, locale: string }`
    - **Output**: `{ status: string, preview: object, error: { code: string, message: string } | null }`
    - **Flow**: Update `program_settings.branding`, `program_settings.ab_tests`, cache in Redis Streams (`content:{merchant_id}:{locale}`, `ab_test:{merchant_id}`), log in `audit_logs`, track via PostHog (`content_updated`, `ab_test_updated`, 15%+ conversion).
  - **POST** `/api/v1/rfm/nudges/personalized` (REST) | gRPC `/rfm.v1/RFMService/CreatePersonalizedNudge`
    - **Input**: `{ merchant_id: string, customer_id: string, segment_id: string, locale: string }`

        - **Output**: `{ status: string, nudge_id: string, error: { code: string, message: string } | null }`
        - **Flow**: Create nudge in `nudges`, cache in Redis Streams (`rfm:nudge:{customer_id}`), notify via Klaviyo/Postscript, log in `audit_logs`, track via PostHog (`rfm_nudge_viewed`, 15%+ conversion).
- **GraphQL Query Examples**:
    - **Query: Fetch Customer Segment for Personalized Nudge**
        - **Purpose**: Retrieves RFM segment for dynamic nudge personalization, used in `/api/v1/rfm/nudges/personalized`.
        - **Query**:

```
query GetCustomerSegment($id: ID!) {
  customer(id: $id) {
    id
    metafield(namespace: "loyalnest", key: "rfm_score") {
      value
    }
  }
}
```

        - **Variables**: `{ "id": "gid://shopify/Customer/987654321" }`
        - **Use Case**: Customer Widget displays personalized nudges (e.g., "VIP Exclusive Offer") based on `rfm_score` from `users`.
- **Service**: Frontend Service (gRPC: `/frontend.v1/*`, Dockerized).

5. Bonus Campaigns (Phase 3)

- **Goal**: Drive urgency with time-sensitive campaigns and data-driven suggestions. Success metric: 20%+ engagement rate, 15%+ redemption rate for Plus merchants, 80%+ campaign suggestion adoption.
- **Types**: Time-sensitive promotions, goal spend ($100 for 200 points), points multipliers (2x), RFM-based campaigns (e.g., Champions only) via `rfm-service`.
- **Geo-Targeted Campaigns**: Region-specific promotions (e.g., 2x points in EU) based on `users.location`, logged in `bonus_campaigns.conditions`.
- **Predictive Campaign Suggestions**: Suggest campaigns based on `rfm_segment_deltas`, `users.churn_score` via `/rfm.v1/RFMService/SuggestCampaign`, displayed in Merchant Dashboard with Polaris `Card`.
- **Conditions**: Scheduled/automated via no-code dashboard (US-MD2), RFM-based eligibility via `rfm-service` (`/rfm.v1/RFMService/GetSegmentCounts`), supports Shopify Flow templates (Phase 5, e.g., "Campaign Started → Notify Customers").
- **Scalability**: Handles 50,000+ customers with Redis Streams (`campaign:{campaign_id}:{region}`, `rfm:suggestion:{merchant_id}`), Bull queues, and circuit breakers.
- **Database Design**:
    - **Table**: `users` (users-service)
        - `id` (text, PK, NOT NULL): Unique ID.
        - `location` (jsonb): e.g., `{"region": "EU"}`.
    - **Table**: `bonus_campaigns` (partitioned by `merchant_id`)

        - `campaign_id` (text, PK, NOT NULL): Unique ID.
        - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
        - `type` (text, CHECK IN ('discount', 'double_points', 'goal_spend')): Type.
        - `multiplier` (numeric(10,2), CHECK >= 1): Multiplier.
        - `start_date`, `end_date` (timestamp(3)): Dates.
        - `conditions` (jsonb): e.g., `{"rfm_segment": "Champions", "region": "EU"}`.
    - **Table**: `campaign_suggestions` (partitioned by `merchant_id`)
        - `suggestion_id` (text, PK, NOT NULL): Unique ID.
        - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
        - `type` (text, CHECK IN ('discount', 'double_points', 'goal_spend')): Type.
        - `conditions` (jsonb): e.g., `{"rfm_segment": "At-Risk", "churn_score": ">0.7"}`.
        - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `campaign_created`, `campaign_discount_applied`, `campaign_suggested`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"region": "EU", "campaign_id": "CAMP123", "suggestion_id": "SUG123"}`.
    - **Indexes**: `idx_bonus_campaigns_merchant_id_type` (btree: `merchant_id`, `type`), `idx_campaign_suggestions_merchant_id` (btree: `merchant_id`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **POST** `/v1/api/campaigns` (REST) | gRPC `/points.v1/PointsService/CreateCampaign`
        - **Input**: `{ merchant_id: string, type: string, multiplier: number, dates: { start: string, end: string }, conditions: { rfm_segment: string, region: string }, locale: string }`
        - **Output**: `{ status: string, campaign_id: string, error: { code: string, message: string } | null }`
        - **Flow**: Insert into `bonus_campaigns`, notify via Klaviyo/Postscript (3 retries, `en`, `es`, `fr`, `ar`, `he` with RTL), cache in Redis Streams (`campaign:{campaign_id}:{region}`), log in `audit_logs`, track via PostHog (`campaign_created`, 20%+ engagement).
    - **GET** `/v1/api/campaigns/suggest` (REST) | gRPC `/rfm.v1/RFMService/SuggestCampaign`
        - **Input**: `{ merchant_id: string, locale: string }`
        - **Output**: `{ status: string, suggestions: [{ suggestion_id: string, type: string, conditions: object }], error: { code: string, message: string } | null }`
        - **Flow**: Query `rfm_segment_deltas`, `users.churn_score`, generate suggestions via Rust/ML (xAI API), insert into `campaign_suggestions`, cache in Redis Streams (`rfm:suggestion:{merchant_id}`), log in `audit_logs`, track via PostHog (`campaign_suggested`, 80%+ adoption).
- **GraphQL Query Examples**:
    - **Query: Fetch Customers for RFM-Based Campaign Eligibility**
        - **Purpose**: Retrieves customer data to determine eligibility for RFM-based campaigns, used in `/rfm.v1/RFMService/GetSegmentCounts`.
        - **Query**:

```
query GetCustomersForCampaign($first: Int, $query: String) {
  customers(first: $first, query: $query) {
    edges {
      node {
        id
        metafield(namespace: "loyalnest", key: "rfm_score") {
          value
        }
        metafield(namespace: "loyalnest", key: "churn_score") {
          value
        }
      }
    }
  }
}
```

- **Variables**: `{ "first": 100, "query": "tag:Champions" }`
- **Use Case**: Merchant Dashboard identifies eligible customers (e.g., Champions) for RFM-based campaigns, using `users.rfm_score` and `users.churn_score` to apply `bonus_campaigns.conditions`.

- **Query: Create Campaign Discount**
  - **Purpose**: Creates a discount for a bonus campaign, used in `/points.v1/PointsService/CreateCampaign`.
  - **Query**:

```
mutation CreateDiscount($input: DiscountCodeBasicInput!) {
  discountCodeBasicCreate(basicCodeDiscount: $input) {
    codeDiscountNode {
      id
      codeDiscount {
        ... on DiscountCodeBasic {
          title
          codes(first: 1) {
            nodes {
              code
            }
          }
        }
      }
    }
    userErrors {
      field
      message
    }
  }
}
```

  - **Variables**:

```
{
  "input": {
    "title": "EU Double Points Campaign",
    "code": "EU2XPOINTS",
    "customerGets": {
      "value": {
        "percentage": 0.0
      }
    },
    "appliesOncePerCustomer": false
  }
}
```

    - **Use Case**: Merchant Dashboard creates discounts for geo-targeted campaigns (e.g., 2x points in EU), stored in `bonus_campaigns` and applied via Shopify Checkout UI Extensions.
  - **Query: Fetch Customer Location for Geo-Targeted Campaigns**
    - **Purpose**: Retrieves customer location data for region-specific campaigns, used in `/points.v1/PointsService/CreateCampaign`.
    - **Query**:

```
query GetCustomerLocations($first: Int, $query: String) {
  customers(first: $first, query: $query) {
    edges {
      node {
        id
        defaultAddress {
          countryCodeV2
        }
      }
    }
  }
}
```

    - **Variables**: `{ "first": 100, "query": "country:EU" }`
    - **Use Case**: Merchant Dashboard targets campaigns to specific regions (e.g., EU) using `users.location`, logged in `bonus_campaigns.conditions` and cached in Redis Streams (`campaign:{campaign_id}:EU`).
- **Service**: RFM Service (gRPC: `/rfm.v1/*`, Dockerized), Points Service (gRPC: `/points.v1/*`, Dockerized).

6. Analytics (Phase 3)

- **Goal**: Enhance analytics for campaign optimization and ROI tracking. Success metric: 80%+ dashboard interaction rate, <1s latency for advanced analytics, 80%+ ROI dashboard view rate.
- **Reports**: ROI dashboard (revenue influenced, referral contribution %, VIP vs. non-VIP purchase behavior) via `rfm-service`, comparisons with industry benchmarks (`rfm_benchmarks`), advanced retention analytics, RFM segment distribution (Chart.js, bar type).

- **Predictive Churn Models**: Use lightweight ML (logistic regression in Rust, integrated via xAI API: https://x.ai/api) in `rfm-service` to calculate `users.churn_score`, suggest nudges for at-risk customers.
- **Insights**: Real-time RFM data (time-weighted recency, lifecycle stages: new lead, repeat buyer, churned, vip), behavioral segments (e.g., churn risk, VIP), A/B testing for nudges.
- **Loyalty ROI Dashboard**: Dedicated dashboard showing revenue influenced by loyalty program, referral contribution %, and VIP vs. non-VIP purchase behavior, visualized with Chart.js (bar and line types) via `/rfm.v1/RFMService/GetRFMVisualizations`.
- **Scalability**: Handles 50,000+ customers with Redis Streams (`rfm:analytics:{merchant_id}`, `rfm:churn:{customer_id}`, `rfm:roi:{merchant_id}`), PostgreSQL partitioning, and circuit breakers.
- **Database Design**:
  - **Table**: `users` (users-service)
    - `id` (text, PK, NOT NULL): Unique ID.
    - `email` (text, AES-256 ENCRYPTED, NOT NULL): Email.
    - `rfm_score` (jsonb, AES-256 ENCRYPTED): e.g., `{"recency": 5, "frequency": 3, "monetary": 4, "score": 4.2}`.
    - `churn_score` (numeric(10,2), CHECK BETWEEN 0 AND 1): Churn probability.
    - `lifecycle_stage` (text, CHECK IN ('new_lead', 'repeat_buyer', 'churned', 'vip')): Lifecycle stage.
  - **Table**: `customer_segments` (rfm-service, partitioned by `merchant_id`)
    - `segment_id` (text, PK, NOT NULL): Unique ID.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `rules` (jsonb): e.g., `{"recency": "<=30", "frequency": ">=5"}`.
    - `name` (jsonb, CHECK ?| ARRAY['en', 'es', 'de', 'ja', 'fr', 'pt', 'ru', 'it', 'nl', 'pl', 'tr', 'fa', 'zh-CN', 'vi', 'id', 'cs', 'ar', 'ko', 'uk', 'hu', 'sv', 'he']): Segment name.
    - `segment_ids` (jsonb ARRAY): List of customer IDs.
  - **Table**: `rfm_segment_deltas` (rfm-service, partitioned by `merchant_id`)
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `customer_id` (text, FK → `users`, NOT NULL): Customer.
    - `segment_change` (jsonb): e.g., `{"from": "At-Risk", "to": "Champions"}`.
    - `updated_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `rfm_score_history` (rfm-service, partitioned by `merchant_id`)
    - `history_id` (text, PK, NOT NULL): Unique ID.
    - `customer_id` (text, FK → `users`, NOT NULL): Customer.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `rfm_score` (jsonb): e.g., `{"recency": 5, "frequency": 3, "monetary": 4, "score": 4.2}`.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `rfm_benchmarks` (rfm-service)
    - `benchmark_id` (text, PK, NOT NULL): Unique ID.
    - `industry` (text, NOT NULL): e.g., "Pet Store", "Electronics".
    - `segment_name` (text, NOT NULL): e.g., "Champions".
    - `thresholds` (jsonb): e.g., `{"recency": "<=30", "frequency": ">=5"}`.
    - `customer_percentage` (numeric(10,2)): Percentage of customers.
    - `avg_rfm_score` (numeric(10,2)): Average RFM score.
    - `last_updated` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.

    - **Table**: `roi_metrics` (rfm-service, partitioned by `merchant_id`)
        - `metric_id` (text, PK, NOT NULL): Unique ID.
        - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
        - `revenue_influenced` (numeric(10,2)): Revenue from loyalty.
        - `referral_contribution` (numeric(10,2)): Referral-driven revenue %.
        - `vip_purchase_share` (numeric(10,2)): VIP vs. non-VIP purchase %.
        - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `churn_predicted`, `rfm_updated`, `roi_viewed`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"churn_score": 0.75, "segment_name": "At-Risk", "revenue_influenced": 5000}`.
    - **Materialized View**: `rfm_segment_counts` (rfm-service)
        - `merchant_id`, `segment_name`, `customer_count`: Refreshed daily (`0 1 * * *`).
    - **Indexes**: `idx_users_churn_score` (btree: `churn_score`), `idx_customer_segments_rules` (gin: `rules`), `idx_rfm_segment_deltas_merchant_id` (btree: `merchant_id`, `updated_at`), `idx_rfm_score_history_customer_id` (btree: `customer_id`, `created_at`), `idx_rfm_benchmarks_industry` (btree: `industry`), `idx_roi_metrics_merchant_id` (btree: `merchant_id`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **GET** `/api/v1/rfm/advanced` (REST) | gRPC `/rfm.v1/RFMService/GetAnalytics`
        - **Input**: `{ merchant_id: string, metrics: array, locale: string }`
        - **Output**: `{ status: string, metrics: { roi: { revenue_influenced: number, referral_contribution: number, vip_purchase_share: number }, retention_rate: number, churn_risk: array, segment_counts: object }, segments: array, error: { code: string, message: string } | null }`
        - **Flow**: Query `rfm_segment_counts`, `roi_metrics`, `users.churn_score`, generate Chart.js data (bar, line types), cache in Redis Streams (`rfm:analytics:{merchant_id}`), log in `audit_logs`, track via PostHog (`analytics_viewed`, `roi_viewed`, 80%+ view rate).
    - **POST** `/api/v1/rfm/churn` (REST) | gRPC `/rfm.v1/RFMService/PredictChurn`
        - **Input**: `{ merchant_id: string, customer_ids: array, locale: string }`
        - **Output**: `{ status: string, predictions: [{ customer_id: string, churn_score: number, lifecycle_stage: string }], error: { code: string, message: string } | null }`
        - **Flow**: Calculate churn via Rust/ML (xAI API: https://x.ai/api), update `users.churn_score`, `users.lifecycle_stage`, cache in Redis Streams (`rfm:churn:{customer_id}`), log in `audit_logs`, track via PostHog (`churn_predicted`, 80%+ accuracy).
    - **WebSocket** `/api/v1/rfm/advanced/stream` | gRPC `/rfm.v1/RFMService/StreamMetrics`
        - **Input**: `{ merchant_id: string, metrics: array }`
        - **Output**: Stream `{ metrics: { segment_counts: object, roi_metrics: object } }`
        - **Flow**: Stream from `rfm_segment_counts`, `roi_metrics`, cache in Redis Streams (`rfm:metrics:{merchant_id}:stream`), log in `audit_logs`, track via PostHog (`analytics_streamed`, <1s latency).
- **GraphQL Query Examples**:

- **Query: Fetch Customer Order Data for ROI Calculation**
  - **Purpose**: Retrieves customer order data to calculate revenue influenced by the loyalty program, used in `/api/v1/rfm/advanced`.
  - **Query**:

```
query GetCustomerOrders($first: Int, $query: String) {
  customers(first: $first, query: $query) {
    edges {
      node {
        id
        orders(first: 100) {
          edges {
            node {
              id
              totalPriceSet {
                shopMoney {
                  amount
                  currencyCode
                }
              }
              createdAt
              metafield(namespace: "loyalnest", key:
"referral_code") {
                value
              }
            }
          }
        }
        metafield(namespace: "loyalnest", key: "lifecycle_stage")
{
          value
        }
      }
    }
  }
}
```

  - **Variables**: `{ "first": 50, "query": "tag:vip" }`
  - **Use Case**: Merchant Dashboard calculates `roi_metrics.revenue_influenced` and `referral_contribution` by analyzing orders linked to loyalty program activities, visualized in Chart.js (bar type) for the Loyalty ROI Dashboard.
- **Query: Fetch RFM Segment Distribution**
  - **Purpose**: Retrieves customer segment counts for RFM distribution visualization, used in `/rfm.v1/RFMService/GetAnalytics`.
  - **Query**:

```
query GetCustomerSegments($first: Int, $query: String) {
  customers(first: $first, query: $query) {
    edges {
      node {
        id
        metafield(namespace: "loyalnest", key: "rfm_segment") {
          value
        }
      }
    }
  }
}
```

- **Variables**: `{ "first": 100, "query": "tag:Champions OR tag:At-Risk" }`
- **Use Case**: Merchant Dashboard generates RFM segment distribution (e.g., Champions, At-Risk) for Chart.js visualization (bar type), sourced from `rfm_segment_counts` and cached in Redis Streams (`rfm:analytics:{merchant_id}`).

- **Query: Fetch Churn Predictions**
  - **Purpose**: Retrieves customer churn scores for predictive analytics, used in `/api/v1/rfm/churn`.
  - **Query**:

```
query GetCustomerChurn($first: Int, $query: String) {
  customers(first: $first, query: $query) {
    edges {
      node {
        id
        metafield(namespace: "loyalnest", key: "churn_score") {
          value
        }
        metafield(namespace: "loyalnest", key: "lifecycle_stage")
{
          value
        }
      }
    }
  }
}
```

  - **Variables**: `{ "first": 50, "query": "churn_score:>0.5" }`
  - **Use Case**: Merchant Dashboard uses `users.churn_score` and `lifecycle_stage` to suggest nudges for at-risk customers, powered by Rust/ML via xAI API, cached in Redis Streams (`rfm:churn:{customer_id}`).

- **Service**: RFM Service (gRPC: `/rfm.v1/*`, Dockerized), Analytics Service (gRPC: `/analytics.v1/*`, Dockerized for non-RFM metrics).

7. Integrations (Phase 3)

- **Goal**: Broaden compatibility with advanced tools. Success metric: 95%+ integration success rate, 99%+ uptime, 90%+ adoption for Shopify Flow templates, 90%+ Klaviyo VIP tier sync adoption.

- **Email/SMS**: Klaviyo, Mailchimp, Yotpo Email & SMS, Postscript for personalized campaigns (3 retries, exponential backoff, queue monitoring via `QueuesPage.tsx` with Chart.js). Supports RFM-based personalization (e.g., "Champions: 2x points") via `rfm-service` (`email_templates`, `email_events`, `/rfm.v1/RFMService/GetNudges`), Klaviyo VIP tier sync (`loyalnest_vip_tier` profile property), and multilingual notifications (`en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)).
- **Others**: Shopify Plus (multi-store sync, 40 req/s), ReCharge (subscriptions, points for recurring orders), Gorgias (support tickets, priority for VIP tiers), Shopify Flow (automation templates, e.g., "Points Earned → Notify Customer"), Zapier for custom workflows (e.g., sync points to CRM).
- **Integration Health Checks**: Monitor status (e.g., "Shopify: OK", "Klaviyo: Error") in Admin Module (US-AM8) with real-time updates via WebSocket, visualized with Chart.js (line type for uptime trends). Alerts via AWS SNS (Slack/email) for failures.
- **Scalability**: Handles 50,000+ customers with Redis Streams (`integration:{merchant_id}`, `klaviyo_vip:{merchant_id}`), PostgreSQL partitioning by `merchant_id`, circuit breakers, and Chaos Mesh testing for Black Friday surges (10,000 orders/hour).
- **Database Design**:
  - **Table**: `integrations` (partitioned by `merchant_id`)
    - `integration_id` (text, PK, NOT NULL): Unique ID.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `platform` (text, CHECK IN ('shopify', 'klaviyo', 'mailchimp', 'yotpo', 'postscript', 'recharge', 'gorgias', 'shopify_flow', 'zapier')): Platform.
    - `settings` (jsonb): e.g., `{"zapier_webhook_url": "https://hooks.zapier.com/...", "klaviyo_vip_sync": true, "shopify_flow_template": "points_earned"}`.
    - `status` (text, CHECK IN ('ok', 'error')): Health status.
    - `last_checked_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Last health check.
  - **Table**: `email_templates` (rfm-service, partitioned by `merchant_id`)
    - `template_id` (text, PK, NOT NULL): Unique ID.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `type` (text, CHECK IN ('points_earned', 'referral_completed', 'tier_upgraded')): Template type.
    - `subject` (jsonb, CHECK ?| ARRAY['en', 'es', 'de', 'ja', 'fr', 'pt', 'ru', 'it', 'nl', 'pl', 'tr', 'fa', 'zh-CN', 'vi', 'id', 'cs', 'ar', 'ko', 'uk', 'hu', 'sv', 'he']): e.g., `{"en": "You Earned Points!"}`.
    - `body` (jsonb, CHECK ?| ARRAY['en', 'es', 'de', 'ja', 'fr', 'pt', 'ru', 'it', 'nl', 'pl', 'tr', 'fa', 'zh-CN', 'vi', 'id', 'cs', 'ar', 'ko', 'uk', 'hu', 'sv', 'he']): e.g., `{"en": "Earned {points} points!"}`.
    - `fallback_language` (text, DEFAULT 'en'): Fallback locale.
  - **Table**: `email_events` (rfm-service, partitioned by `merchant_id`)
    - `event_id` (text, PK, NOT NULL): Unique ID.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `event_type` (text, CHECK IN ('points_earned', 'referral_completed', 'tier_upgraded')): Event type.
    - `recipient_email` (text, AES-256 ENCRYPTED, NOT NULL): Recipient.
    - `template_id` (text, FK → `email_templates`): Template used.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `integration_configured`, `integration_health_checked`, `klaviyo_vip_synced`.

    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"platform": "klaviyo", "status": "ok", "template": "points_earned"}`.
  - **Indexes**: `idx_integrations_merchant_id_platform` (btree: `merchant_id`, `platform`), `idx_email_templates_merchant_id_type` (btree: `merchant_id`, `type`), `idx_email_events_merchant_id` (btree: `merchant_id`, `created_at`), `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **POST** `/v1/api/integrations` (REST) | gRPC `/admin.v1/AdminService/ConfigureIntegration`
    - **Input**: `{ merchant_id: string, type: string, settings: { klaviyo_vip_sync: boolean, zapier_webhook_url: string }, locale: string }`
    - **Output**: `{ status: string, integration_id: string, error: { code: string, message: string } | null }`
    - **Flow**: Validate settings, insert into `integrations`, test connection (e.g., Klaviyo VIP sync, Zapier webhook ping), cache in Redis Streams (`integration:{merchant_id}`, `klaviyo_vip:{merchant_id}`), log in `audit_logs`, track via PostHog (`integration_configured`, `klaviyo_vip_synced`, 90%+ adoption).
  - **POST** `/v1/api/integrations/zapier` (REST) | gRPC `/admin.v1/AdminService/ConfigureWebhook`
    - **Input**: `{ merchant_id: string, webhook_url: string, events: array, locale: string }`
    - **Output**: `{ status: string, webhook_id: string, error: { code: string, message: string } | null }`
    - **Flow**: Validate webhook URL, update `integrations`, cache in Redis Streams (`zapier:{merchant_id}`), log in `audit_logs`, track via PostHog (`zapier_configured`, 90%+ adoption).
  - **GET** `/v1/api/integrations/health` (REST) | gRPC `/admin.v1/AdminService/CheckIntegrationHealth`
    - **Input**: `{ merchant_id: string, platform: string }`
    - **Output**: `{ status: string, health: { platform: string, status: string, last_checked: string }, error: { code: string, message: string } | null }`
    - **Flow**: Query `integrations`, perform health check, stream via WebSocket, cache in Redis Streams (`health:{merchant_id}:{platform}`), log in `audit_logs`, track via PostHog (`integration_health_checked`, 99%+ uptime).
- **GraphQL Query Examples**:
  - **Query: Sync Klaviyo VIP Tier**
    - **Purpose**: Updates customer VIP tier in Klaviyo for personalized campaigns, used in `/v1/api/integrations`.
    - **Query**:

```
mutation UpdateCustomerMetafield($input: CustomerInput!) {
  customerUpdate(input: $input) {
    customer {
      id
      metafield(namespace: "loyalnest", key: "vip_tier") {
        value
      }
    }
    userErrors {
      field
      message
    }
  }
}
```

- **Variables**:

```
{
  "input": {
    "id": "gid://shopify/Customer/987654321",
    "metafields": [
      {
        "namespace": "loyalnest",
        "key": "vip_tier",
        "value": "Gold",
        "type": "string"
      }
    ]
  }
}
```

- **Use Case**: Admin Module syncs `vip_tiers.name` to Klaviyo's `loyalnest_vip_tier` profile property, enabling RFM-based personalized campaigns (e.g., "Champions: 2x points").

- **Query: Configure Shopify Flow Template**
  - **Purpose**: Creates a Shopify Flow trigger for automation templates, used in `/v1/api/integrations`.
  - **Query**:

```
mutation CreateWebhook($input: WebhookSubscriptionInput!) {
  webhookSubscriptionCreate(input: $input) {
    webhookSubscription {
      id
      topic
      endpoint {
        ... on WebhookHttpEndpoint {
          callbackUrl
        }
      }
    }
    userErrors {
      field
      message
    }
  }
}
```

- **Variables**:

```
{
  "input": {
    "topic": "CUSTOMERS_UPDATE",
    "endpoint": {
      "callbackUrl":
"https://loyalnest.com/webhooks/points_earned"
    }
  }
}
```

- **Use Case**: Admin Module sets up Shopify Flow templates (e.g., "Points Earned → Notify Customer"), stored in `integrations.settings` and tracked via PostHog (`shopify_flow_configured`, 90%+ adoption).

- **Query: Check Integration Health Status**
  - **Purpose**: Retrieves app installation status for integration health checks, used in `/v1/api/integrations/health`.
  - **Query**:

```
query GetAppInstallations($first: Int) {
  appInstallations(first: $first) {
    edges {
      node {
        id
        app {
          id
          title
        }
        active
      }
    }
  }
}
```

  - **Variables**: `{ "first": 10 }`
  - **Use Case**: Admin Module monitors integration health (e.g., Klaviyo, Postscript) by checking `appInstallations.active`, visualized with Chart.js (line type) and streamed via WebSocket, logged in `audit_logs`.

- **Service**: Admin Service (gRPC: `/admin.v1/*`, Dockerized).

8. Testing and Monitoring (Phase 3)

- **Goal**: Ensure reliability and performance with developer-friendly tools. Success metric: 99%+ uptime, <1s alert latency, 80%+ test coverage, 95%+ Redis cache hit rate, 90%+ sandbox adoption.
- **Automated Testing**: Jest (unit/integration tests for `PointsService`, `ReferralService`, `rfm-service`, e.g., churn prediction, RFM calculations), Cypress (E2E for Customer Widget, Merchant Dashboard, Admin Module), cargo test (Rust/Wasm Shopify Functions), k6 (load testing for 10,000 orders/hour, including `rfm-service`).
- **Chaos Testing**: Simulate failures (e.g., Redis downtime, `rfm-service` latency, pod crashes) using Chaos Mesh in Kubernetes to ensure resilience (99%+ uptime).
- **Staging Sandbox**: Developer mode with dummy customer data for testing configurations (e.g., points, VIP tiers, campaigns), integrated with `dev.sh` script and Kubernetes namespaces.
- **Monitoring Metrics**: API latency (<1s), Redis cache hits (>95%), Bull queue delays (<5s), error rates (<1%) via Prometheus/Grafana. Error tracking with Sentry, centralized logging with Loki. Rate limit monitoring with AWS SNS alerts.
- **Developer Tools**: `dev.sh` script for mock data, RFM simulation, merchant referral testing, and sandbox mode, enhanced with AI tools (Grok for edge cases, Copilot for code, Cursor for tests).
- **Database Design**:
  - **Table**: `sandbox_configs` (partitioned by `merchant_id`)
    - `sandbox_id` (text, PK, NOT NULL): Unique ID.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `dummy_data` (jsonb): e.g., `{"users": [{"id": "C123", "points_balance": 500}]}`.
    - `status` (text, CHECK IN ('active', 'expired')): Status.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `system_alert`, `chaos_test_failed`, `rate_limit_alerted`, `sandbox_enabled`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"error_rate": 0.01, "chaos_type": "redis_downtime", "sandbox_id": "SAND123"}`.
  - **Indexes**: `idx_sandbox_configs_merchant_id` (btree: `merchant_id`), `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **POST** `/v1/api/admin/sandbox` (REST) | gRPC `/admin.v1/AdminService/EnableSandbox`
    - **Input**: `{ merchant_id: string, dummy_data: object, locale: string }`
    - **Output**: `{ status: string, sandbox_id: string, error: { code: string, message: string } | null }`
    - **Flow**: Create temporary `merchants` record with dummy `users`, `points_transactions`, insert into `sandbox_configs`, isolate via Kubernetes namespaces, cache in Redis Streams (`sandbox:{merchant_id}`), log in `audit_logs`, track via PostHog (`sandbox_enabled`, 90%+ adoption).
  - **GET** `/v1/api/monitoring` (REST) | gRPC `/admin.v1/AdminService/GetMonitoringMetrics`
    - **Input**: `{ service: string, time_range: string }`
    - **Output**: `{ status: string, metrics: { latency: number, error_rate: number, cache_hits: number, queue_delays: number, chaos_results: object }, error: { code: string, message: string } | null }`

    - **Flow**: Query Prometheus, cache in Redis Streams (`metrics:{service}`), enforce RBAC (`admin:full`, `admin:analytics`), log in `audit_logs`, track via PostHog (`monitoring_viewed`, 80%+ view rate).
- **GraphQL Query Examples**:
  - **Query: Create Sandbox Customers**
    - **Purpose**: Creates dummy customer data for sandbox testing, used in `/v1/api/admin/sandbox`.
    - **Query**:

```
mutation CreateCustomers($input: [CustomerInput!]!) {
  customerCreateBulk(inputs: $input) {
    customers {
      id
      email
      metafield(namespace: "loyalnest", key: "points_balance") {
        value
      }
    }
    userErrors {
      field
      message
    }
  }
}
```

    - **Variables**:

```json
{
  "input": [
    {
      "email": "dummy1@sandbox.com",
      "metafields": [
        {
          "namespace": "loyalnest",
          "key": "points_balance",
          "value": "500",
          "type": "number_integer"
        }
      ]
    },
    {
      "email": "dummy2@sandbox.com",
      "metafields": [
        {
          "namespace": "loyalnest",
          "key": "points_balance",
          "value": "1000",
          "type": "number_integer"
        }
        ]
      }
    ]
  }
```

- **Use Case**: Admin Module creates dummy customers in `sandbox_configs.dummy_data` for testing points, VIP tiers, and campaigns, isolated in Kubernetes namespaces and tracked via PostHog (`sandbox_enabled`).

- **Query: Fetch Monitoring Metrics for Load Testing**
  - **Purpose**: Retrieves recent order data to monitor system performance during load testing, used in `/v1/api/monitoring`.
  - **Query**:

```
query GetRecentOrders($first: Int, $after: String) {
  orders(first: $first, after: $after) {
    edges {
      node {
        id
        createdAt
        totalPriceSet {
          shopMoney {
            amount
            currencyCode
          }
        }
      }
    }
    pageInfo {
      hasNextPage
      endCursor
    }
  }
}
```

  - **Variables**: `{ "first": 10, "after": null }`
  - **Use Case**: Admin Module monitors order processing rates during k6 load tests (10,000 orders/hour), feeding into Prometheus/Grafana for latency and error rate metrics, cached in Redis Streams (`metrics:orders`).

- **Query: Check API Rate Limit Usage**
  - **Purpose**: Retrieves Shopify API usage to monitor rate limits, used in `/v1/api/monitoring`.
  - **Query**:

```
query GetAppApiUsage {
  app {
    id
    apiUsage {
        graphql {
          currentUsage
          limit
        }
        rest {
          currentUsage
          limit
        }
      }
    }
  }
```

- **Use Case**: Admin Module tracks API usage (40 req/s for Shopify Plus) to trigger AWS SNS alerts for rate limits, logged in `audit_logs` and tracked via PostHog (`rate_limit_alerted`).

- **Service**: Admin Service (gRPC: `/admin.v1/*`, Dockerized).

9. Multilingual Support (Phase 3)

- **Goal**: Support multi-region stores. Success metric: 80%+ adoption of localized widgets, 90%+ translation accuracy, Lighthouse CI score 90+.
- **Implementation**: Uses i18next for widget localization (`en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)), JSONB fields in `email_templates.body`, `nudges.title`, `program_settings.branding`. Persists user choice in `localStorage`, respects `Accept-Language` headers, supports RTL for `ar`, `he` with WCAG 2.1 compliance (ARIA labels, keyboard navigation).
- **Translation Management**: No-code dashboard for merchants to customize translations (US-MD2), validated for accuracy via automated checks (e.g., missing keys, RTL alignment).
- **Scalability**: Handles 50,000+ customers with Redis Streams (`config:{merchant_id}:{locale}`), <1s widget rendering.
- **Database Design**:
  - **Table**: `merchants`
    - `language` (jsonb, CHECK ?| ARRAY['en', 'es', 'de', 'ja', 'fr', 'pt', 'ru', 'it', 'nl', 'pl', 'tr', 'fa', 'zh-CN', 'vi', 'id', 'cs', 'ar', 'ko', 'uk', 'hu', 'sv', 'he']): e.g., `{"default": "en", "supported": ["en", "es", "de", "ja", "fr", "pt", "ru", "it", "nl", "pl", "tr", "fa", "zh-CN", "vi", "id", "cs", "ar", "ko", "uk", "hu", "sv", "he"], "rtl": ["ar", "he"]}`.
  - **Table**: `email_templates` (rfm-service)
    - `template_id` (text, PK, NOT NULL): Unique ID.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `type` (text, CHECK IN ('points_earned', 'referral_completed', 'tier_upgraded')): Template type.
    - `subject` (jsonb, CHECK ?| ARRAY['en', 'es', 'de', 'ja', 'fr', 'pt', 'ru', 'it', 'nl', 'pl', 'tr', 'fa', 'zh-CN', 'vi', 'id', 'cs', 'ar', 'ko', 'uk', 'hu', 'sv', 'he']): e.g., `{"en": "You Earned Points!"}`.
    - `body` (jsonb, CHECK ?| ARRAY['en', 'es', 'de', 'ja', 'fr', 'pt', 'ru', 'it', 'nl', 'pl', 'tr', 'fa', 'zh-CN', 'vi', 'id', 'cs', 'ar', 'ko', 'uk', 'hu', 'sv', 'he']): e.g., `{"en": "Earned {points} points!"}`.
    - `fallback_language` (text, DEFAULT 'en'): Fallback locale.
  - **Table**: `audit_logs`

        - `action` (text, NOT NULL): e.g., `language_updated`, `translation_validated`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"locale": "ar", "rtl": true}`.
    - **Indexes**: `idx_merchants_language` (gin: `language`), `idx_email_templates_merchant_id_type` (btree: `merchant_id`, `type`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **GET** `/v1/api/widget/config` (REST) | gRPC `/frontend.v1/FrontendService/GetWidgetConfig`
        - **Input**: `{ merchant_id: string, locale: string }`
        - **Output**: `{ status: string, language: string, translations: { points_label: string, rewards: object }, rtl: boolean, error: { code: string, message: string } | null }`
        - **Flow**: Query `merchants.language`, `email_templates`, cache in Redis Streams (`config:{merchant_id}:{locale}`), log in `audit_logs`, track via PostHog (`language_selected`, 80%+ adoption).
    - **PUT** `/v1/api/translations` (REST) | gRPC `/frontend.v1/FrontendService/UpdateTranslations`
        - **Input**: `{ merchant_id: string, translations: object, locale: string }`
        - **Output**: `{ status: string, error: { code: string, message: string } | null }`
        - **Flow**: Validate translations, update `program_settings.branding`, `email_templates`, cache in Redis Streams (`translations:{merchant_id}:{locale}`), log in `audit_logs`, track via PostHog (`translation_updated`, 90%+ accuracy).
- **GraphQL Query Examples**:
    - **Query: Fetch Merchant Language Settings**
        - **Purpose**: Retrieves merchant's language settings for widget localization, used in `/v1/api/widget/config`.
        - **Query**:

```
query GetMerchantLanguage($id: ID!) {
  shop(id: $id) {
    id
    metafield(namespace: "loyalnest", key: "language") {
      value
    }
  }
}
```

        - **Variables**: `{ "id": "gid://shopify/Shop/123456789" }`
        - **Use Case**: Customer Widget fetches `merchants.language` to apply i18next localization (e.g., `en`, `ar` with RTL), cached in Redis Streams (`config:{merchant_id}:{locale}`) and respecting `Accept-Language` headers.
    - **Query: Update Translation for Email Template**
        - **Purpose**: Updates multilingual email template content, used in `/v1/api/translations`.

    - **Query**:

      ```
      mutation UpdateMetafield($input: MetafieldInput!) {
        metafieldsSet(input: [$input]) {
          metafields {
            id
            namespace
            key
            value
          }
          userErrors {
            field
            message
          }
        }
      }
      ```

    - **Variables**:

      ```
      {
        "input": {
          "namespace": "loyalnest",
          "key": "email_template_points_earned",
          "value": "{\"en\": \"You Earned {points} Points!\", \"ar\": \"لقد ربحت {points} نقاط!\", \"rtl\": true}",
          "ownerId": "gid://shopify/Shop/123456789",
          "type": "json"
        }
      }
      ```

    - **Use Case**: Merchant Dashboard updates `email_templates.body` for multilingual notifications (e.g., `points_earned` in `en`, `ar`), validated for RTL alignment and cached in Redis Streams (`translations:{merchant_id}:{locale}`).
  - **Query: Fetch Customer Language Preference**
    - **Purpose**: Retrieves customer's preferred language for personalized notifications, used in `/v1/api/widget/config`.
    - **Query**:

      ```
      query GetCustomerLanguage($id: ID!) {
        customer(id: $id) {
          id
          metafield(namespace: "loyalnest", key: "preferred_language") {
            value
          }
        }
      }
      ```

        - **Variables**: `{ "id": "gid://shopify/Customer/987654321" }`
        - **Use Case**: Customer Widget delivers localized content based on `preferred_language`, stored in `localStorage` and aligned with `merchants.language`, tracked via PostHog (`language_selected`, 80%+ adoption).
- **Service**: Frontend Service (gRPC: `/frontend.v1/*`, Dockerized).

10. RFM Nudges (Phase 3)

- **Goal**: Encourage engagement with RFM-based nudges. Success metric: 10%+ interaction rate, 15%+ conversion rate for nudges.
- **Features**: Displays smart nudges (e.g., “Stay Active!” for At-Risk, “Welcome Back!” for Inactive) in Customer Widget via Storefront API (US-CW10), logs interactions in `nudge_events`, supports `en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL) via i18next. A/B tests nudge variants (e.g., text, timing) via `nudges.variants`.
- **Personalized Nudge Content**: Uses `users` data (e.g., “Hi {first_name}, earn 50 points!”) with RFM segments (Champions, At-Risk, New, Inactive) and lifecycle stages (new lead, repeat buyer, churned) via `rfm-service`.
- **Scalability**: Handles 50,000+ customers with Redis Streams (`rfm:nudge:{customer_id}`), PostgreSQL partitioning, and circuit breakers.
- **Database Design**:
    - **Table**: `nudges` (rfm-service, partitioned by `merchant_id`)
        - `nudge_id` (text, PK, NOT NULL): Unique ID.
        - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
        - `type` (text, CHECK IN ('at-risk', 'loyal', 'new', 'inactive', 'churned')): Nudge type.
        - `title` (jsonb, CHECK ?| ARRAY['en', 'es', 'de', 'ja', 'fr', 'pt', 'ru', 'it', 'nl', 'pl', 'tr', 'fa', 'zh-CN', 'vi', 'id', 'cs', 'ar', 'ko', 'uk', 'hu', 'sv', 'he']): e.g., `{"en": "Stay Active", "ar": "..."}`.
        - `personalized_content` (jsonb, CHECK ?| ARRAY['en', 'es', 'de', 'ja', 'fr', 'pt', 'ru', 'it', 'nl', 'pl', 'tr', 'fa', 'zh-CN', 'vi', 'id', 'cs', 'ar', 'ko', 'uk', 'hu', 'sv', 'he']): e.g., `{"en": "Hi {first_name}, earn 50 points!"}`.
        - `conditions` (jsonb): e.g., `{"rfm_segment": "At-Risk", "lifecycle_stage": "churned"}`.
        - `variants` (jsonb): e.g., `{"A": {"title": "Stay Active"}, "B": {"title": "Keep Shopping"}}`.
    - **Table**: `nudge_events` (rfm-service, partitioned by `merchant_id`)
        - `event_id` (text, PK, NOT NULL): Interaction ID.
        - `customer_id` (text, FK → `users`, NOT NULL): Customer.
        - `nudge_id` (text, FK → `nudges`, NOT NULL): Nudge.
        - `action` (text, CHECK IN ('view', 'click', 'dismiss')): Action.
        - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `rfm_nudge_personalized`, `nudge_ab_tested`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"nudge_id": "nudge1", "customer_name": "John", "variant": "A"}`.
    - **Indexes**: `idx_nudges_merchant_id_type` (btree: `merchant_id`, `type`), `idx_nudge_events_customer_id` (btree: `customer_id`, `created_at`), `idx_audit_logs_action` (btree: `action`).

  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **GET** `/api/v1/rfm/nudges` (REST) | gRPC `/rfm.v1/RFMService/GetNudges`
    - **Input**: `{ customer_id: string, locale: string }`
    - **Output**: `{ status: string, nudges: [{ nudge_id: string, title: string, personalized_content: string, variant: string }], error: { code: string, message: string } | null }`
    - **Flow**: Query `nudges` based on `users.rfm_score`, `users.lifecycle_stage`, render personalized content, cache in Redis Streams (`rfm:nudge:{customer_id}`), log in `audit_logs`, track via PostHog (`rfm_nudge_viewed`, 10%+ interaction rate).
  - **POST** `/api/v1/rfm/nudges/ab-test` (REST) | gRPC `/rfm.v1/RFMService/StartNudgeABTest`
    - **Input**: `{ merchant_id: string, nudge_id: string, variants: array, locale: string }`
    - **Output**: `{ status: string, test_id: string, error: { code: string, message: string } | null }`
    - **Flow**: Update `nudges.variants`, cache in Redis Streams (`rfm:ab_test:{merchant_id}:{nudge_id}`), log in `audit_logs`, track via PostHog (`nudge_ab_tested`, 15%+ conversion rate).
- **GraphQL Query Examples**:
  - **Query: Fetch Customer Data for Nudge Personalization**
    - **Purpose**: Retrieves customer RFM and lifecycle data to personalize nudges, used in `/api/v1/rfm/nudges`.
    - **Query**:

```
query GetCustomerNudgeData($id: ID!) {
  customer(id: $id) {
    id
    firstName
    metafield(namespace: "loyalnest", key: "rfm_score") {
      value
    }
    metafield(namespace: "loyalnest", key: "lifecycle_stage") {
      value
    }
  }
}
```

    - **Variables**: `{ "id": "gid://shopify/Customer/987654321" }`
    - **Use Case**: Customer Widget personalizes nudges (e.g., “Hi {first_name}, earn 50 points!”) based on `users.rfm_score` and `lifecycle_stage`, rendered via Storefront API and cached in Redis Streams (`rfm:nudge:{customer_id}`).
  - **Query: Log Nudge Interaction**
    - **Purpose**: Logs customer interactions with nudges for analytics, used in `/api/v1/rfm/nudges`.
    - **Query**:

```
mutation CreateNudgeEvent($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
  }
}
```

- **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "nudge_event",
    "value": "{\"nudge_id\": \"nudge1\", \"action\": \"click\",
\"created_at\": \"2025-07-31T12:44:00Z\"}",
    "ownerId": "gid://shopify/Customer/987654321",
    "type": "json"
  }
}
```

- **Use Case**: Customer Widget logs nudge interactions (e.g., view, click, dismiss) in `nudge_events`, tracked via PostHog (`rfm_nudge_viewed`, 10%+ interaction rate) and stored for analytics.

- **Query: Configure A/B Test for Nudge Variants**
  - **Purpose**: Sets up A/B test variants for nudges, used in `/api/v1/rfm/nudges/ab-test`.
  - **Query**:

```
mutation UpdateNudgeVariants($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
    }
  }
```

- **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "nudge_variants",
    "value": "{\"nudge_id\": \"nudge1\", \"variants\": {\"A\":
{\"title\": \"Stay Active\"}, \"B\": {\"title\": \"Keep
Shopping\"}}}",
    "ownerId": "gid://shopify/Shop/123456789",
    "type": "json"
  }
}
```

- **Use Case**: Merchant Dashboard configures A/B test variants for nudges (e.g., "Stay Active" vs. "Keep Shopping"), stored in `nudges.variants`, cached in Redis Streams (`rfm:ab_test:{merchant_id}:{nudge_id}`), and tracked via PostHog (`nudge_ab_tested`, 15%+ conversion rate).

- **Service**: RFM Service (gRPC: `/rfm.v1/*`, Dockerized).

11. Gamification (Phase 3)

- **Goal**: Motivate customers with badges and leaderboards. Success metric: 15%+ engagement rate, 80%+ leaderboard view rate.
- **Features**: Awards badges for actions (purchases, referrals, social follows, merchant referrals in Phase 5) via no-code dashboard (US-MD2), displays ranks in Redis sorted sets (`leaderboard:{merchant_id}:{cycle}`), visualized in Customer Widget with Chart.js (bar type). Notifies via Klaviyo/Postscript (3 retries, `en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)).
- **Seasonal Leaderboards**: Monthly reset cycles (e.g., `2025-07`) to encourage recurring engagement, displayed in Customer Widget (US-CW11).
- **Scalability**: Handles 50,000+ customers with Redis Streams (`badge:{customer_id}`, `leaderboard:{merchant_id}:{cycle}`), PostgreSQL partitioning, and circuit breakers.
- **Database Design**:
  - **Table**: `gamification_achievements` (partitioned by `merchant_id`)
    - `achievement_id` (text, PK, NOT NULL): Unique ID.
    - `customer_id` (text, FK → `users`, NOT NULL): Customer.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `badge` (jsonb, CHECK ?| ARRAY['en', 'es', 'de', 'ja', 'fr', 'pt', 'ru', 'it', 'nl', 'pl', 'tr', 'fa', 'zh-CN', 'vi', 'id', 'cs', 'ar', 'ko', 'uk', 'hu', 'sv', 'he']): e.g., `{"en": "Loyal Customer", "ar": "..."}`.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `leaderboards` (partitioned by `merchant_id`)
    - `leaderboard_id` (text, PK, NOT NULL): Unique ID.

        - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
        - `customer_id` (text, FK → `users`, NOT NULL): Customer.
        - `score` (integer, CHECK >= 0): Points-based score.
        - `cycle` (text, NOT NULL): e.g., `2025-07`.
        - `updated_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `badge_earned`, `leaderboard_reset`, `leaderboard_viewed`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"cycle": "2025-07", "badge": "Loyal Customer"}`.
    - **Indexes**: `idx_gamification_achievements_customer_id` (btree: `customer_id`, `created_at`), `idx_leaderboards_merchant_id_cycle` (btree: `merchant_id`, `cycle`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **POST** `/v1/api/gamification/action` (REST) | gRPC `/analytics.v1/AnalyticsService/AwardBadge`
        - **Input**: `{ customer_id: string, action_type: string, locale: string }`
        - **Output**: `{ status: string, achievement_id: string, badge: string, error: { code: string, message: string } | null }`
        - **Flow**: Insert into `gamification_achievements`, notify via Klaviyo/Postscript, cache in Redis Streams (`badge:{customer_id}`), log in `audit_logs`, track via PostHog (`badge_earned`, 15%+ engagement).
    - **POST** `/v1/api/leaderboards/reset` (REST) | gRPC `/points.v1/PointsService/ResetLeaderboard`
        - **Input**: `{ merchant_id: string, cycle: string, locale: string }`
        - **Output**: `{ status: string, leaderboard_id: string, error: { code: string, message: string } | null }`
        - **Flow**: Reset Redis sorted set (`leaderboard:{merchant_id}:{cycle}`), update `leaderboards`, log in `audit_logs`, track via PostHog (`leaderboard_reset`, 80%+ view rate).
    - **GET** `/v1/api/leaderboards` (REST) | gRPC `/analytics.v1/AnalyticsService/GetLeaderboard`
        - **Input**: `{ merchant_id: string, cycle: string, locale: string }`
        - **Output**: `{ status: string, rankings: [{ customer_id: string, score: number, rank: number }], error: { code: string, message: string } | null }`
        - **Flow**: Query Redis sorted set (`leaderboard:{merchant_id}:{cycle}`), generate Chart.js data, cache in Redis Streams (`leaderboard:{merchant_id}:{cycle}`), log in `audit_logs`, track via PostHog (`leaderboard_viewed`, 80%+ view rate).
- **GraphQL Query Examples**:
    - **Query: Award Badge to Customer**
        - **Purpose**: Assigns a badge to a customer for a specific action, used in `/v1/api/gamification/action`.
        - **Query**:

```
mutation AwardBadge($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
  }
}
```

- **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "badge_earned",
    "value": "{\"badge\": {\"en\": \"Loyal Customer\", \"ar\":
\"عميل مخلص\"}, \"created_at\": \"2025-07-31T12:46:00Z\"}",
    "ownerId": "gid://shopify/Customer/987654321",
    "type": "json"
  }
}
```

- **Use Case**: Customer Widget awards badges (e.g., “Loyal Customer”) for actions like purchases, stored in `gamification_achievements`, notified via Klaviyo/Postscript, and tracked via PostHog (`badge_earned`, 15%+ engagement).

- **Query: Fetch Leaderboard Rankings**
  - **Purpose**: Retrieves leaderboard rankings for a specific cycle, used in `/v1/api/leaderboards`.
  - **Query**:

```
query GetLeaderboard($first: Int, $query: String) {
  customers(first: $first, query: $query) {
    edges {
      node {
        id
        metafield(namespace: "loyalnest", key:
"leaderboard_score") {
          value
        }
      }
    }
  }
}
```

- **Variables**: `{ "first": 10, "query": "tag:leaderboard_2025-07" }`
- **Use Case**: Customer Widget displays top ranks from `leaderboards` for the `2025-07` cycle, visualized with Chart.js (bar type), cached in Redis Streams (`leaderboard:{merchant_id}:{cycle}`), and tracked via PostHog (`leaderboard_viewed`, 80%+ view rate).

- **Query: Reset Seasonal Leaderboard**
  - **Purpose**: Resets leaderboard scores for a new cycle, used in `/v1/api/leaderboards/reset`.
  - **Query**:

```
mutation ResetLeaderboard($input: MetafieldInput!) {
  metafieldsSet(input: [$input]) {
    metafields {
      id
      namespace
      key
      value
    }
    userErrors {
      field
      message
    }
  }
}
```

  - **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "leaderboard_cycle",
    "value": "{\"cycle\": \"2025-08\", \"reset_at\": \"2025-07-31T23:59:59Z\"}",
    "ownerId": "gid://shopify/Shop/123456789",
    "type": "json"
  }
}
```

  - **Use Case**: Merchant Dashboard resets `leaderboards` for a new monthly cycle (e.g., `2025-08`), clears Redis sorted set (`leaderboard:{merchant_id}:{cycle}`), logs in `audit_logs`, and tracks via PostHog (`leaderboard_reset`, 80%+ view rate).

- **Service**: Analytics Service (gRPC: `/analytics.v1/*`, Dockerized).

12. Customer Data Import (Phase 3)

- **Goal**: Initialize loyalty programs with imported data. Success metric: 95%+ import success for 50,000+ records under 5 minutes, 90%+ data validation accuracy.
- **Features**: Supports CSV imports (max 10MB, fields: `id`, `email`, `points`, `rfm_score`, `lifecycle_stage`) for `users-service`, encrypts PII (`email`, `rfm_score`) with AES-256 via pgcrypto, processes async via Bull queue (monitored in `QueuesPage.tsx` with Chart.js). Validates unique emails, checks RFM data format, notifies via Klaviyo/Postscript (3 retries, `en`, `es`, `de`, `ja`, `fr`, `pt`, `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL)). Triggers RFM updates via `/rfm.v1/RFMService/PreviewRFMSegments`.
- **Sources**: Smile.io, LoyaltyLion, custom CSVs, Shopify Customer Export API.
- **Scalability**: Handles 50,000+ records with Redis Streams (`import:{merchant_id}`), PostgreSQL partitioning, and circuit breakers.
- **Database Design**:
  - **Table**: `users` (users-service)
    - `id` (text, PK, NOT NULL): Unique ID.
    - `email` (text, AES-256 ENCRYPTED, NOT NULL): Email.
    - `rfm_score` (jsonb, AES-256 ENCRYPTED): e.g., `{"recency": 5, "frequency": 3, "monetary": 4}`.
    - `churn_score` (numeric(10,2), CHECK BETWEEN 0 AND 1): Churn probability.
    - `lifecycle_stage` (text, CHECK IN ('new_lead', 'repeat_buyer', 'churned', 'vip')): Lifecycle stage.
  - **Table**: `data_imports` (partitioned by `merchant_id`)
    - `import_id` (text, PK, NOT NULL): Unique ID.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `source` (text, CHECK IN ('smile_io', 'loyaltylion', 'shopify', 'custom')): Source.
    - `status` (text, CHECK IN ('pending', 'processing', 'completed', 'failed')): Status.
    - `record_count` (integer, CHECK >= 0): Number of records.
    - `error_log` (jsonb): e.g., `{"row": 10, "error": "Duplicate email"}`.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `data_import_started`, `data_import_failed`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"source": "smile_io", "record_count": 5000}`.
  - **Indexes**: `idx_users_email` (btree: `email`), `idx_data_imports_merchant_id_status` (btree: `merchant_id`, `status`), `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **POST** `/v1/api/data/import` (REST) | gRPC `/admin.v1/AdminService/ImportCustomerData`
    - **Input**: `{ merchant_id: string, source: string, file_url: string, locale: string }`
    - **Output**: `{ status: string, import_id: string, error: { code: string, message: string } | null }`
    - **Flow**: Validate CSV (max 10MB, unique emails), enqueue in Bull, insert into `data_imports`, update `users`, trigger `/rfm.v1/RFMService/PreviewRFMSegments`, cache in Redis Streams (`import:{merchant_id}`), notify via Klaviyo/Postscript, log in `audit_logs`, track via PostHog (`data_import_started`, 95%+ success rate).

- **GET** `/v1/api/data/import/status` (REST) | gRPC `/admin.v1/AdminService/GetImportStatus`
  - **Input**: `{ merchant_id: string, import_id: string }`
  - **Output**: `{ status: string, import: { status: string, record_count: number, error_log: object }, error: { code: string, message: string } | null }`
  - **Flow**: Query `data_imports`, cache in Redis Streams (`import:{merchant_id}:{import_id}`), log in `audit_logs`, track via PostHog (`data_import_status_viewed`, 80%+ view rate).

- **GraphQL Query Examples**:
  - **Query: Import Customer Data**
    - **Purpose**: Imports customer data into Shopify for loyalty program initialization, used in `/v1/api/data/import`.
    - **Query**:

```
mutation CreateCustomers($input: [CustomerInput!]!) {
  customerCreateBulk(inputs: $input) {
    customers {
      id
      email
      metafield(namespace: "loyalnest", key: "rfm_score") {
        value
      }
      metafield(namespace: "loyalnest", key: "lifecycle_stage") {
        value
      }
    }
    userErrors {
      field
      message
    }
  }
}
```

- **Variables**:

```
{
  "input": [
    {
      "email": "customer1@example.com",
      "metafields": [
        {
          "namespace": "loyalnest",
          "key": "rfm_score",
          "value": "{\"recency\": 5, \"frequency\": 3,
\"monetary\": 4}",
          "type": "json"
        },
        {
          "namespace": "loyalnest",
          "key": "lifecycle_stage",
          "value": "repeat_buyer",
          "type": "string"
        }
      ]
    }
  ]
}
```

        - **Use Case**: Admin Module imports customer data from sources like Smile.io or Shopify Customer Export API, validates unique emails, encrypts PII (`email`, `rfm_score`) with AES-256, and stores in `users`, tracked via PostHog (`data_import_started`, 95%+ success rate).
    - **Query: Fetch Import Status**
        - **Purpose**: Retrieves the status of a data import job, used in `/v1/api/data/import/status`.
        - **Query**:

```
query GetImportStatus($id: ID!) {
  shop(id: $id) {
    id
    metafield(namespace: "loyalnest", key: "import_status") {
      value
    }
  }
}
```

        - **Variables**: `{ "id": "gid://shopify/Shop/123456789" }`
        - **Use Case**: Merchant Dashboard checks `data_imports` status (e.g., `completed`, `failed`) and `error_log` for issues like duplicate emails, visualized in `QueuesPage.tsx` with Chart.js, cached in Redis Streams (`import:{merchant_id}:{import_id}`).
    - **Query: Trigger RFM Segment Preview**
        - **Purpose**: Previews RFM segments after data import, used in `/rfm.v1/RFMService/PreviewRFMSegments`.
        - **Query**:

```
query GetCustomersForRFM($first: Int, $query: String) {
  customers(first: $first, query: $query) {
    edges {
      node {
        id
        metafield(namespace: "loyalnest", key: "rfm_score") {
          value
        }
        metafield(namespace: "loyalnest", key: "lifecycle_stage")
{
```

```
              value
            }
          }
        }
      }
    }
```

- **Variables**: `{ "first": 100, "query": "tag:imported" }`
- **Use Case**: Admin Module triggers RFM segment updates post-import, using `users.rfm_score` and `lifecycle_stage` to preview segments, cached in Redis Streams (`import:{merchant_id}`), and tracked via PostHog (`data_import_started`).

- **Service**: Admin Service (gRPC: `/admin.v1/*`, Dockerized).