LoyalNest App Features

Overview

This document outlines the feature set for the LoyalNest Shopify app, targeting small (100–5,000 customers), medium (5,000–50,000 customers), and Shopify Plus merchants (50,000+ customers, 10,000 orders/hour). The **Must Have** features (Phase 1, MVP) align with user stories (US-CW1–CW15, US-MD1–MD18, US-AM1–AM13, US-BI1–BI5), wireframes, and the LoyalNest App Feature Analysis. The app uses a microservices architecture (`rfm-service`, `users-service`, `roles-service`, `AdminCore`, `AdminFeatures`, `Points`, `Referrals`, `Auth`, `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 Black Friday surges (10,000 orders/hour), Shopify Plus compatibility (40 req/s API limits), GDPR/CCPA compliance (AES-256 encryption, 90-day Backblaze B2 backups), multilingual support (`en`, `es`, `fr`, `de`, `pt`, `ja` in Phases 2–5; `ru`, `it`, `nl`, `pl`, `tr`, `fa`, `zh-CN`, `vi`, `id`, `cs`, `ar`(RTL), `ko`, `uk`, `hu`, `sv`, `he`(RTL) in Phase 6), and WCAG 2.1 compliance (Lighthouse CI: 90+ for ARIA, LCP, FID, CLS). 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`).

MUST HAVE FEATURES

Essential for core functionality, user engagement, Shopify App Store compliance, and scalability for 100–5,000 customers.

1. Points Program (Phase 1)

- **Goal**: Enable customers to earn and redeem points to drive loyalty. Success metric: 90%+ successful point awards within 1s, 85%+ redemption rate, 80%+ widget engagement.
- **Earning Actions**: Purchases (10 points/$), account creation (200 points), newsletter signups (100 points), reviews (100 points), birthdays (200 points). Adjusted by RFM multipliers (`program_settings.rfm_thresholds`, e.g., 1.5x for Champions, time-weighted recency: R5 ≤14 days for subscriptions).
- **Redemption Options**: Discounts ($5 off for 500 points), free shipping (1000 points), free products (1500 points), coupons at checkout via Shopify Checkout UI Extensions (Rust/Wasm). Supports multi-currency discounts (Phase 6).
- **Points Adjustments**: Deductions for order cancellations/refunds via `orders/cancel` webhook with Redis idempotency key, logged in `points_transactions` and `audit_logs`.
- **Point Expiry Notifications**: Notify customers 30 days before expiry via email/SMS (Klaviyo/Postscript, 3 retries, `en`, `es`, `fr`, `ar` with RTL). Tracks via PostHog (`expiry_notification_sent`, 90%+ delivery rate).
- **Rate Limit Alerts**: Notify merchants via Slack/email (AWS SNS) when approaching Shopify API limits (2 req/s REST, 40 req/s Plus) with circuit breakers and exponential backoff (3 retries, 500ms base delay). Tracks via PostHog (`rate_limit_alerted`).
- **Customization**: Customizable rewards panel, launcher button, points currency (e.g., Stars) with Polaris-compliant UI, i18next support (`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 for styling.

- **Scalability**: Handles 10,000 orders/hour (Black Friday) via Redis Streams (`points:{customer_id}`, `expiry:{customer_id}`), Bull queues for async processing, PostgreSQL partitioning by `merchant_id`. Uses circuit breakers and Chaos Mesh for resilience.
- **Database Design**:
  - **Table**: `points_transactions` (partitioned by `merchant_id`)
    - `transaction_id` (text, PK, NOT NULL): Unique ID.
    - `customer_id` (text, FK → `customers`, 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.
    - `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**: `customers`
    - `points_balance` (integer, NOT NULL, DEFAULT 0): Current balance.
    - `total_points_earned` (integer, NOT NULL, DEFAULT 0): Cumulative points.
    - `rfm_score` (jsonb): e.g., `{"recency": 5, "frequency": 3, "monetary": 4, "score": 4.2}`.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `points_earned`, `points_adjusted`, `expiry_notification_sent`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"points": 100, "source": "order"}`.
  - **Indexes**: `idx_points_transactions_customer_id` (btree: `customer_id`, `created_at`), `idx_customers_rfm_score` (gin: `rfm_score`), `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **POST** `/v1/api/points/earn` (REST) | gRPC `/points.v1/PointsService/EarnPoints`
    - **Input**: `{ customer_id: string, order_id: string, action_type: string, locale: string }`
    - **Output**: `{ status: string, transaction_id: string, points: number, error: { code: string, message: string } | null }`
    - **Flow**: Validate order via GraphQL Admin API (Rust/Wasm), apply RFM multiplier (`program_settings.rfm_thresholds`), insert into `points_transactions`, update `customers.points_balance`, cache in Redis Streams (`points:{customer_id}`), notify via Klaviyo/Postscript (3 retries, exponential backoff), log in `audit_logs`, track via PostHog (`points_earned`, 20%+ redemption rate).
  - **POST** `/v1/api/rewards/redeem` (REST) | gRPC `/points.v1/PointsService/RedeemReward`
    - **Input**: `{ customer_id: string, reward_id: string, locale: string }`
    - **Output**: `{ status: string, redemption_id: string, discount_code: string, error: { code: string, message: string } | null }`
    - **Flow**: Validate `points_balance` ≥ `rewards.points_cost`, create discount via GraphQL Admin API (Rust/Wasm, 40 req/s for Plus), insert into `reward_redemptions` (AES-256 encrypted `discount_code`), deduct points, cache in Redis Streams (`points:

`{customer_id}`), log in `audit_logs`, track via PostHog (`points_redeemed`, 15%+ redemption rate for Plus).

- **GET** `/v1/api/points/rate-limits` (REST) | gRPC `/admin.v1/AdminService/GetRateLimitStatus`
  - **Input**: `{ merchant_id: string }`
  - **Output**: `{ status: string, rate_limit: { rest: number, graphql: number }, error: { code: string, message: string } | null }`
  - **Flow**: Query `api_logs` for rate limit usage, cache in Redis Streams (`rate_limit:{merchant_id}`), notify via AWS SNS if near threshold (95% of limit), log in `audit_logs`, track via PostHog (`rate_limit_alerted`).

- **GraphQL Query Examples**:
  - **Query: Validate Order for Points Earning**
    - **Purpose**: Fetches order details from Shopify to validate before awarding points, used in `/v1/api/points/earn`.
    - **Query**:

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

    - **Variables**: `{ "id": "gid://shopify/Order/123456789" }`
    - **Use Case**: Merchant Dashboard uses this to confirm order validity and calculate points (e.g., 10 points/$). The response feeds into `points_transactions` and applies RFM multipliers.
  - **Query: Create Discount for Redemption**
    - **Purpose**: Creates a discount code 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 $5 Off",
    "code": "LN5OFF",
    "customerGets": {
      "value": {
        "discountAmount": {
          "amount": 5.0,
          "currencyCode": "USD"
        }
      }
    },
    "appliesOncePerCustomer": true
  }
}
```

- **Use Case**: Merchant Dashboard applies discounts at checkout via Shopify Checkout UI Extensions, storing the code in `reward_redemptions` (AES-256 encrypted).

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

2. Referral Program (Phase 1)

- **Goal**: Drive customer acquisition via referrals. Success metric: 7%+ referral conversion rate (SMS), 3%+ (email), 80%+ dashboard interaction rate.
- **Sharing Options**: Email (Klaviyo), SMS (Postscript), social media (Facebook, Instagram). Generates unique `referral_code` via Storefront API, supports merchant referral program (Phase 5, e.g., "Refer a merchant, get 1 month free").
- **Rewards**: Points (50 for referrer/referee) or discounts (10% off) issued via GraphQL Admin API (Rust/Wasm). Supports multi-tier referrals (Phase 3).
- **Referral Link Analytics**: Tracks clicks, conversions, and merchant referrals in Merchant Dashboard using Polaris `DataTable` and Chart.js, showing CTR and conversion rate.

- **Dedicated Referral Page**: Displays incentives for referrer, friend, and merchant referrals (Phase 5), localized (`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.
- **Async Processing**: Queued notifications via Bull (monitored in `QueuesPage.tsx` with Chart.js), cached in Redis Streams (`referral:{referral_code}`, `queues:{merchant_id}`).
- **Scalability**: Handles 1,000 concurrent shares with circuit breakers, Chaos Mesh testing, and PostgreSQL partitioning.
- **Database Design**:
  - **Table**: `referral_links` (partitioned by `merchant_id`)
    - `referral_link_id` (text, PK, NOT NULL): Unique ID.
    - `advocate_customer_id` (text, FK → `customers`, 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 → `customers`, NOT NULL): Advocate.
    - `friend_customer_id` (text, FK → `customers`, NOT NULL): Friend.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `reward_id` (text, FK → `rewards`): Reward.
    - `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')): Action type.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `merchant_referrals` (Phase 5, partitioned by `merchant_id`)
    - `merchant_referral_id` (text, PK, NOT NULL): Unique ID.
    - `advocate_merchant_id` (text, FK → `merchants`, NOT NULL): Referring merchant.
    - `referred_merchant_id` (text, FK → `merchants`, NOT NULL): Referred merchant.
    - `reward_id` (text, FK → `rewards`): Reward (e.g., 1 month free).
    - `status` (text, CHECK IN ('pending', 'completed', 'expired')): Status.
    - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `referral_created`, `referral_completed`, `merchant_referral_created`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"referral_code": "REF123", "merchant_referral_id": "MER123"}`.
  - **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_merchant_referrals_advocate_merchant_id` (btree: `advocate_merchant_id`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **POST** `/v1/api/referrals/create` (REST) | gRPC `/referrals.v1/ReferralService/CreateReferral`
        - **Input**: `{ advocate_customer_id: string, merchant_referral_id: string | null, locale: string }`
        - **Output**: `{ status: string, referral_code: string, error: { code: string, message: string } | null }`
        - **Flow**: Validate input, insert into `referral_links` (link `merchant_referral_id` for Phase 5), cache in Redis Streams (`referral:{referral_code}`), notify via Klaviyo/Postscript (3 retries, `en`, `es`, `fr`, `ar` with RTL), log in `audit_logs`, track via PostHog (`referral_created`, 7%+ SMS conversion).
    - **POST** `/v1/api/referrals/complete` (REST) | gRPC `/referrals.v1/ReferralService/CompleteReferral`
        - **Input**: `{ referral_code: string, friend_customer_id: string, locale: string }`
        - **Output**: `{ status: string, referral_id: string, error: { code: string, message: string } | null }`
        - **Flow**: Verify `customers/create` webhook (HMAC validated, Redis idempotency key), validate `referral_code`, insert into `referrals`, award points via `/points.v1/PointsService/EarnPoints`, notify via Klaviyo/Postscript, cache in Redis Streams, log in `audit_logs`, track via PostHog (`referral_completed`, 7%+ conversion).
    - **GET** `/v1/api/referrals/analytics` (REST) | gRPC `/analytics.v1/AnalyticsService/GetReferralAnalytics`
        - **Input**: `{ merchant_id: string, referral_code: string | null, merchant_referral_id: string | null, date_range: { start: string, end: string } }`
        - **Output**: `{ status: string, analytics: { clicks: number, conversions: number, ctr: number, merchant_referrals: number }, error: { code: string, message: string } | null }`
        - **Flow**: Query `referral_links`, `referral_events`, `merchant_referrals`, cache in Redis Streams (`referral_analytics:{referral_code}`), log in `audit_logs`, track via PostHog (`referral_analytics_viewed`, 80%+ view rate).
- **GraphQL Query Examples**:
    - **Query: Generate Referral Code**
        - **Purpose**: Creates a unique referral code via Shopify Storefront API for sharing, used in `/v1/api/referrals/create`.
        - **Query**:

```
mutation CreateCustomerAccessToken($input:
CustomerAccessTokenCreateInput!) {
  customerAccessTokenCreate(input: $input) {
    customerAccessToken {
      accessToken
```

```
    }
    userErrors {
      field
      message
    }
  }
}
```

- **Variables**:

```
{
  "input": {
    "email": "customer@example.com",
    "password": "securepassword"
  }
}
```

- **Use Case**: Merchant Dashboard generates a `referral_code` stored in `referral_links`, used for tracking in Customer Widget and analytics.

- **Query: Award Referral Discount**
  - **Purpose**: Creates a discount for a completed referral, used in `/v1/api/referrals/complete`.
  - **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 10% Off",
    "code": "REF10OFF",
    "customerGets": {
      "value": {
        "percentage": 0.1
      }
    },
    "appliesOncePerCustomer": true
  }
}
```

    - **Use Case**: Merchant Dashboard issues a 10% discount to referrer/referee, stored in `referrals.reward_id`, displayed in Customer Widget.
- **Service**: Referrals Service (gRPC: `/referrals.v1/*`, Dockerized).

3. On-Site Content (Phase 1)

- **Goal**: Enhance visibility and engagement via widgets. Success metric: 85%+ widget interaction rate, 10%+ nudge conversion rate, Lighthouse CI score 90+.
- **Widgets**: SEO-friendly loyalty page, rewards panel, launcher button, checkout integration, points display on product pages via Storefront API, sticky bar (Phase 2, US-CW14), post-purchase widget (Phase 2, US-CW15). Supports Theme App Extensions (Phase 5).
- **Nudges**: Post-purchase prompts, email capture popups, RFM-based nudges (e.g., “Stay Active!” for At-Risk) via `rfm-service` (`nudges`, `nudge_events`). Supports A/B testing (variants in `nudges.variants` JSONB, tracked via PostHog: `nudge_variant_viewed`, `nudge_variant_clicked`, 10%+ click-through). Localized via 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)).
- **Accessibility**: ARIA labels (e.g., `aria-label="Join loyalty program"`), keyboard navigation, screen reader support, RTL for `ar`, `he`, WCAG 2.1 compliance, Lighthouse CI scores (90+ for ARIA, LCP, FID, CLS).
- **Scalability**: Renders <1s via Redis Streams caching (`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., `{"loyalty_page": {"en": {...}, "es": {...}, "ar": {...}}, "popup": {...}, "sticky_bar": {...}}`.
        - `ab_tests` (jsonb): e.g., `{"launcher_button": {"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')), `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_started`, `rfm_nudge_viewed`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"type": "loyalty_page", "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` (REST) | gRPC `/frontend.v1/FrontendService/UpdateContent`
        - **Input**: `{ merchant_id: string, branding: { loyalty_page: object, popup: object, sticky_bar: object, post_purchase: object }, ab_tests: object, locale: string }`
        - **Output**: `{ status: string, preview: object, error: { code: string, message: string } | null }`
        - **Flow**: Validate inputs, 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_started`, 10%+ click-through).
    - **GET** `/api/v1/rfm/nudges` (REST) | gRPC `/rfm.v1/RFMService/GetNudges`
        - **Input**: `{ merchant_id: string, customer_id: string, locale: string }`
        - **Output**: `{ status: string, nudges: [{ nudge_id: string, type: string, title: object, description: object, variants: object }], error: { code: string, message: string } | null }`
        - **Flow**: Query `nudges`, `nudge_events`, cache in Redis Streams (`rfm:nudge:{customer_id}`), use i18next, log in `audit_logs`, track via PostHog (`rfm_nudge_viewed`, 10%+ conversion).
- **GraphQL Query Examples**:
    - **Query: Fetch Customer Data for Widget Display**
        - **Purpose**: Retrieves customer points balance for display in the rewards panel or loyalty page via Storefront API.
        - **Query**:

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

        - **Variables**: `{ "id": "gid://shopify/Customer/987654321" }`

        - **Use Case**: Customer Widget displays points balance from `customers.points_balance`, fetched via Storefront API for real-time updates.
- **Service**: Frontend Service (gRPC: `/frontend.v1/*`, Dockerized).

4. Integrations (Phase 1)

- **Goal**: Seamlessly connect with Shopify and third-party tools. Success metric: 99%+ sync accuracy, 90%+ notification delivery rate, 95%+ integration uptime.
- **Shopify**: OAuth, webhooks (`orders/create`, `orders/cancel`, `customers/data_request`, `customers/redact`) with HMAC validation and Redis idempotency keys. Triggers RFM updates in `rfm-service` (`/rfm.v1/RFMService/PreviewRFMSegments`) for `orders/create`. POS for online/in-store rewards (10 points/$). Handles rate limits (2 req/s REST, 40 req/s Plus) with circuit breakers and Bull queues (`rate_limit_queue:{merchant_id}`).
- **Email/SMS**: Klaviyo, Postscript for notifications (points, referrals, RFM nudges via `rfm-service`: `email_templates`, `email_events`) with 3 retries, AES-256 encryption, exponential backoff, and queue monitoring via `QueuesPage.tsx` (Chart.js).
- **Reviews**: Yotpo, Judge.me for points-for-reviews, integrated via GraphQL Admin API (Rust/Wasm).
- **Webhook Retry Dashboard**: Monitor and retry failed webhooks (e.g., `orders/create`, `customers/redact`) in Admin Module with real-time log streaming (WebSocket, Loki + Grafana).
- **Shopify Flow**: Templates for automation (Phase 5, e.g., “Order Cancelled → Adjust Points”).
- **Scalability**: Handles 10,000 orders/hour with PostgreSQL partitioning, Redis Streams (`order:{order_id}`, `rfm:burst:{merchant_id}`), and Chaos Mesh testing.
- **Database Design**:
    - **Table**: `api_logs`
        - `log_id` (text, PK, NOT NULL): Log ID.
        - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
        - `endpoint` (text, NOT NULL): e.g., `orders/create`.
        - `payload` (jsonb): Webhook payload.
        - `retry_count` (integer, DEFAULT 0): Retry attempts.
        - `status` (text, CHECK IN ('pending', 'success', 'failed')): Status.
        - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `shopify_sync`, `webhook_retry`, `integration_configured`, `rfm_updated`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"endpoint": "orders/create", "platform": "klaviyo"}`.
    - **Indexes**: `idx_api_logs_merchant_id_endpoint` (btree: `merchant_id`, `endpoint`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **POST** `/v1/api/shopify/webhook` (REST) | gRPC `/admin.v1/AdminCoreService/HandleShopifyWebhook`
        - **Input**: `{ merchant_id: string, endpoint: string, payload: object }`
        - **Output**: `{ status: string, error: { code: string, message: string } | null }`
        - **Flow**: Validate HMAC with Redis idempotency key, process webhook, update `users`, `points_transactions`, trigger RFM updates via

`/rfm.v1/RFMService/PreviewRFMSegments`, cache in Redis Streams (`order:{order_id}`), log in `api_logs`, `audit_logs`, track via PostHog (`shopify_sync`, 99%+ accuracy).

- **GET** `/v1/api/webhooks/retries` (REST) | gRPC `/admin.v1/AdminFeaturesService/GetWebhookRetries`
  - **Input**: `{ merchant_id: string, endpoint: string, status: string }`
  - **Output**: `{ status: string, retries: [{ log_id: string, endpoint: string, retry_count: number, status: string }], error: { code: string, message: string } | null }`
  - **Flow**: Query `api_logs`, cache in Redis Streams (`webhook:{merchant_id}`), stream via WebSocket, log in `audit_logs`, track via PostHog (`webhook_retry_viewed`, 80%+ view rate).

- **GraphQL Query Examples**:
  - **Query: Fetch Webhook Status**
    - **Purpose**: Retrieves webhook subscription details to monitor integration health, used in `/v1/api/webhooks/retries`.
    - **Query**:

```
query GetWebhookSubscriptions {
  webhookSubscriptions(first: 10) {
    edges {
      node {
        id
        topic
        endpoint {
          ... on WebhookHttpEndpoint {
            callbackUrl
          }
        }
        status
      }
    }
  }
}
```

    - **Use Case**: Admin Module displays webhook status (e.g., `orders/create`) in the Webhook Retry Dashboard, streamed via WebSocket for real-time monitoring.
- **Service**: Admin Service (gRPC: `/admin.v1/*`, Dockerized).

5. Analytics (Phase 1)

- **Goal**: Provide actionable loyalty insights with RFM segmentation. Success metric: 80%+ dashboard interaction rate, <1s latency for real-time updates.
- **Reports**: Customer engagement (points issued/redeemed), referral ROI, retention metrics, sales attribution via GraphQL Admin API (Rust/Wasm).
- **RFM Analytics**: Managed by `rfm-service`. Recency (≤7 to >90 days), Frequency (1 to >10 orders), Monetary (<0.5x to >5x AOV) with static thresholds (Phase 1) and time-weighted recency (Phase 3, e.g., R5 ≤14 days for subscriptions). Segments stored in `customer_segments`, deltas in

`rfm_segment_deltas`, history in `rfm_score_history`, benchmarks in `rfm_benchmarks`, and materialized view `rfm_segment_counts` (refreshed daily, `0 1 * * *`). Supports multi-segment membership (e.g., "VIP" and "At-Risk High-Value" in `customer_segments.segment_ids` JSONB).

- **Smart Nudges**: Triggered for At-Risk customers (R1–2, F1–2, M1–2) via `nudges` and `nudge_events`, sent through Klaviyo/Postscript/AWS SES with A/B testing (variants in `nudges.variants` JSONB).
- **Real-Time Dashboard Updates**: Stream metrics (points issued, redemptions, RFM segments) via WebSocket (`/api/v1/rfm/stream`) and Chart.js in Polaris `Card`, cached in Redis Streams (`rfm:metrics:{merchant_id}:stream`).
- **PostHog Tracking**: Events (`points_earned`, `redeem_clicked`, `rfm_nudge_viewed`, `rfm_segment_filtered`, `rfm_benchmarks_viewed`) for usage insights.
- **Scalability**: Handles 5,000 customers with Redis Streams (`rfm:preview:{merchant_id}`, `rfm:burst:{merchant_id}`), Bull queues (`rate_limit_queue:{merchant_id}`), PostgreSQL partitioning by `merchant_id`, and circuit breakers.
- **Database Design**:
  - **Table**: `users` (users-service)
    - `id` (text, PK, NOT NULL): Unique ID.
    - `email` (text, AES-256 ENCRYPTED, NOT NULL): Email.
    - `first_name`, `last_name` (text): Customer name.
    - `rfm_score` (jsonb, AES-256 ENCRYPTED): e.g., `{"recency": 5, "frequency": 3, "monetary": 4, "score": 4.2}`.
    - `metadata` (jsonb): e.g., `{"lifecycle_stage": "repeat_buyer"}`.
  - **Table**: `customer_segments` (rfm-service, partitioned by `merchant_id`)
    - `segment_id` (text, PK, NOT NULL): Segment ID.
    - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
    - `rules` (jsonb, NOT NULL): e.g., `{"recency": ">=4", "frequency": ">=3"}`.
    - `name` (jsonb, NOT NULL): e.g., `{"en": "Champions", "es": "Campeones", "ar": "الأبطال"}`.
    - `segment_ids` (jsonb ARRAY): e.g., `["Champions", "VIP"]`.
  - **Table**: `rfm_segment_deltas` (rfm-service, partitioned by `merchant_id`)
    - `merchant_id`, `customer_id`, `segment_change` (jsonb), `updated_at`: Tracks segment changes.
  - **Table**: `rfm_score_history` (rfm-service, partitioned by `merchant_id`)
    - `history_id`, `customer_id`, `merchant_id`, `rfm_score` (jsonb), `created_at`: Score history.
  - **Table**: `rfm_benchmarks` (rfm-service)
    - `benchmark_id`, `merchant_size`, `aov_range`, `segment_name`, `customer_percentage`, `avg_rfm_score`, `last_updated`: Industry benchmarks.
  - **Table**: `nudges` (rfm-service, partitioned by `merchant_id`)
    - `nudge_id`, `merchant_id`, `type` (CHECK IN ('at-risk', 'loyal', 'new', 'inactive', 'tier_dropped')), `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**: `email_templates` (rfm-service, partitioned by `merchant_id`)
    - `template_id`, `merchant_id`, `type` (CHECK IN ('tier_change', 'nudge')), `subject` (jsonb), `body` (jsonb), `fallback_language`: Notification templates.

    - **Table**: `email_events` (rfm-service, partitioned by `merchant_id`)
        - `event_id`, `merchant_id`, `event_type` (CHECK IN ('sent', 'failed')), `recipient_email` (AES-256 ENCRYPTED): Notification events.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `analytics_viewed`, `rfm_updated`, `rfm_nudge_viewed`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"segment_name": "Champions", "customer_count": 50}`.
    - **Materialized View**: `rfm_segment_counts`
        - `merchant_id`, `segment_name`, `customer_count`: Refreshed daily (`0 1 * * *`).
    - **Indexes**: `idx_users_rfm_score` (gin: `rfm_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_merchant_size` (btree: `merchant_size`), `idx_nudges_merchant_id` (btree: `merchant_id`), `idx_nudge_events_customer_id` (btree: `customer_id`), `idx_email_templates_merchant_id` (btree: `merchant_id`), `idx_email_events_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/analytics` (REST) | gRPC `/rfm.v1/RFMService/GetAnalytics`
        - **Input**: `{ merchant_id: string, locale: string }`
        - **Output**: `{ status: string, metrics: { members: number, points_issued: number, referral_roi: number, segment_counts: object }, segments: array, error: { code: string, message: string } | null }`
        - **Flow**: Query `rfm_segment_counts`, `customer_segments`, generate Chart.js data (bar, line, scatter types), cache in Redis Streams (`rfm:preview:{merchant_id}`), use i18next, log in `audit_logs`, track via PostHog (`rfm_segment_filtered`, `rfm_benchmarks_viewed`, 80%+ view rate).
    - **POST** `/api/v1/rfm/config` (REST) | gRPC `/rfm.v1/RFMService/UpdateThresholds`
        - **Input**: `{ merchant_id: string, thresholds: { recency: object, frequency: object, monetary: object, recency_decay: string }, segments: array, locale: string }`
        - **Output**: `{ status: string, preview: object, error: { code: string, message: string } | null }`
        - **Flow**: Validate thresholds, update `program_settings.rfm_thresholds`, trigger preview via `/rfm.v1/RFMService/PreviewRFMSegments`, cache in Redis Streams (`rfm:preview:{merchant_id}`), log in `audit_logs`, track via PostHog (`rfm_config_saved`).
    - **WebSocket** `/api/v1/rfm/stream` | gRPC `/rfm.v1/RFMService/StreamMetrics`
        - **Input**: `{ merchant_id: string, metrics: array }`
        - **Output**: Stream `{ metrics: { segment_counts: object, nudge_events: object } }`
        - **Flow**: Stream data from `rfm_segment_counts`, `nudge_events`, cache in Redis Streams (`rfm:metrics:{merchant_id}:stream`), log in `audit_logs`, track via PostHog (`rfm_nudge_viewed`, <1s latency).
- **GraphQL Query Examples**:
    - **Query: Fetch Customer Segments for Analytics**

    - **Purpose**: Retrieves customer segment data for Merchant Dashboard analytics, used in `/api/v1/rfm/analytics`.
    - **Query**:

```
query GetCustomerSegments($merchantId: String!) {
  customers(first: 100, query: $merchantId) {
    edges {
      node {
        id
        metafield(namespace: "loyalnest", key: "rfm_score") {
          value
        }
      }
    }
  }
}
```

    - **Variables**: `{ "merchantId": "merchant_123" }`
    - **Use Case**: Merchant Dashboard visualizes RFM segments (e.g., Champions) using Chart.js, sourced from `customer_segments` and `rfm_score`.
- **Service**: RFM Service (gRPC: `/rfm.v1/*`, Dockerized), Analytics Service (gRPC: `/analytics.v1/*`, Dockerized for non-RFM metrics).

6. GDPR Compliance (Phase 1)

- **Goal**: Ensure compliance with GDPR/CCPA. Success metric: 100% request completion within 72 hours, 90%+ consent tracking accuracy.
- **Webhooks**: Handle `customers/data_request`, `customers/redact` webhooks with HMAC validation and Redis idempotency keys.
- **Data Export/Redaction**: UI in Admin Module to process requests, encrypt PII (`customers.email`, `rfm_score`) with AES-256 via pgcrypto. Supports async processing for 50,000+ customers.
- **Customer Consent Tracking**: Log explicit consent (newsletter, RFM tracking, referrals) in `consents` table, displayed in Customer Widget (US-CW8).
- **Data Import**: Import from Smile.io, LoyaltyLion via CSV (max 10MB, fields: `customer_id`, `email`, `points`, `rfm_score`), validate unique emails, process async via Bull queue.
- **Database Design**:
    - **Table**: `gdpr_requests` (partitioned by `merchant_id`)
        - `request_id` (text, PK, NOT NULL): Request ID.
        - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
        - `customer_id` (text, FK → `customers`): Customer.
        - `request_type` (text, CHECK IN ('data_request', 'redact')): Type.
        - `status` (text, CHECK IN ('pending', 'completed', 'failed')): Status.
        - `retention_expires_at` (timestamp(3)): 90-day retention.
        - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
    - **Table**: `consents` (partitioned by `merchant_id`)
        - `consent_id` (text, PK, NOT NULL): Consent ID.
        - `customer_id` (text, FK → `customers`, NOT NULL): Customer.

        - `merchant_id` (text, FK → `merchants`, NOT NULL): Merchant.
        - `type` (text, CHECK IN ('newsletter', 'rfm_tracking', 'referrals')): Consent type.
        - `status` (text, CHECK IN ('granted', 'revoked')): Consent status.
        - `created_at` (timestamp(3), DEFAULT CURRENT_TIMESTAMP): Timestamp.
    - **Table**: `customers`
        - `email` (text, AES-256 ENCRYPTED, NOT NULL): Email.
        - `rfm_score` (jsonb, AES-256 ENCRYPTED): e.g., `{"recency": 5, "frequency": 3, "monetary": 4}`.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `gdpr_request_submitted`, `gdpr_processed`, `consent_granted`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"request_type": "redact", "consent_type": "newsletter"}`.
    - **Indexes**: `idx_gdpr_requests_merchant_id_request_type` (btree: `merchant_id`, `request_type`), `idx_consents_customer_id_type` (btree: `customer_id`, `type`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **POST** `/v1/api/gdpr/request` (REST) | gRPC `/admin.v1/AdminService/SubmitGDPRRequest`
        - **Input**: `{ customer_id: string, request_type: string, locale: string }`
        - **Output**: `{ status: string, request_id: string, error: { code: string, message: string } | null }`
        - **Flow**: Validate `customer_id`, insert into `gdpr_requests`, notify 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)), cache in Redis Streams (`gdpr:{customer_id}`), log in `audit_logs`, track via PostHog (`gdpr_request_submitted`, 90%+ success rate).
    - **POST** `/v1/api/consents` (REST) | gRPC `/admin.v1/AdminService/UpdateConsent`
        - **Input**: `{ customer_id: string, type: string, status: string, locale: string }`
        - **Output**: `{ status: string, consent_id: string, error: { code: string, message: string } | null }`
        - **Flow**: Validate `customer_id`, insert into `consents`, cache in Redis Streams (`consent:{customer_id}`), log in `audit_logs`, track via PostHog (`consent_granted`, 90%+ accuracy).
- **GraphQL Query Examples**:
    - **Query: Handle GDPR Data Request**
        - **Purpose**: Fetches customer data for GDPR export, used in `/v1/api/gdpr/request`.
        - **Query**:

```
query GetCustomerData($id: ID!) {
  customer(id: $id) {
    id
    email
    firstName
    lastName
    metafield(namespace: "loyalnest", key: "rfm_score") {
```

```
        value
      }
    }
  }
```

    - **Variables**: `{ "id": "gid://shopify/Customer/987654321" }`
    - **Use Case**: Admin Module processes GDPR `data_request`, exporting encrypted `customers.email` and `rfm_score` for compliance.
- **Service**: Admin Service (gRPC: `/admin.v1/*`, Dockerized).

7. Security (Phase 1)

- **Goal**: Secure access and data. Success metric: 100% secure authentication, zero data breaches, 95%+ anomaly detection rate.
- **Authentication**: Shopify OAuth for Merchant Dashboard/Customer Widget, RBAC with MFA via Auth0 and `roles-service` (`roles` table, gRPC: `/roles.v1/RolesService/GetPermissions`) for Admin Module (`admin:full`, `admin:analytics`, `admin:support`).
- **Encryption**: AES-256 (pgcrypto) for `users.email`, `users.rfm_score` (users-service), `merchants.api_token`, `reward_redemptions.discount_code`, `email_events.recipient_email` (rfm-service).
- **IP Whitelisting**: Restrict Admin Module access to trusted IPs in `merchants.ip_whitelist`.
- **Webhook Idempotency**: Use Redis keys (`webhook:{merchant_id}:{event_id}`) to prevent duplicate processing.
- **Anomaly Detection**: Detect unusual activity (e.g., >100 points adjustments/hour) with AWS SNS alerts (Slack/email).
- **Error Handling**: Returns 400 (invalid input), 401 (unauthorized), 429 (rate limits), `RFM_CONFIG_INVALID`, `EXPORT_FAILED`, `NUDGE_LIMIT_EXCEEDED` with transaction rollbacks, logged in Sentry and `audit_logs`.
- **Security Testing**: OWASP ZAP for penetration testing, Chaos Mesh for resilience testing.
- **Database Design**:
  - **Table**: `merchants`
    - `merchant_id` (text, PK, NOT NULL): Unique ID.
    - `api_token` (text, AES-256 ENCRYPTED, NOT NULL): Shopify token.
    - `ip_whitelist` (jsonb): e.g., `["192.168.1.1", "10.0.0.1"]`.
  - **Table**: `roles` (roles-service)
    - `role_id` (text, PK, NOT NULL): Unique ID.
    - `role_name` (text, UNIQUE, NOT NULL): Role name.
    - `permissions` (jsonb): e.g., `["admin:full", "admin:analytics"]`.
  - **Table**: `users` (users-service)
    - `id` (text, PK, NOT NULL): Unique ID.
    - `email` (text, AES-256 ENCRYPTED, NOT NULL): Email.
    - `first_name`, `last_name` (text): Customer name.
    - `rfm_score` (jsonb, AES-256 ENCRYPTED): e.g., `{"recency": 5, "frequency": 3, "monetary": 4}`.
    - `metadata` (jsonb): e.g., `{"lifecycle_stage": "repeat_buyer"}`.
  - **Table**: `admin_users`
    - `admin_id` (text, PK, NOT NULL): Unique ID.

        - `username` (text, UNIQUE, NOT NULL): Username.
        - `password_hash` (text, AES-256 ENCRYPTED, NOT NULL): Bcrypt hash.
        - `metadata` (jsonb, AES-256 ENCRYPTED): e.g., `{"role": "admin:full", "mfa_enabled": true}`.
    - **Table**: `audit_logs`
        - `action` (text, NOT NULL): e.g., `auth_failed`, `access_granted`, `ip_rejected`, `anomaly_detected`, `rfm_config_updated`.
        - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
        - `metadata` (jsonb): e.g., `{"role": "admin:full", "ip": "192.168.1.1"}`.
    - **Indexes**: `idx_merchants_api_token` (btree: `api_token`), `idx_roles_role_name` (btree: `role_name`), `idx_users_email` (btree: `email`), `idx_admin_users_username` (btree: `username`), `idx_audit_logs_action` (btree: `action`).
    - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
    - **POST** `/v1/api/auth/login` (REST) | gRPC `/auth.v1/AuthService/Login`
        - **Input**: `{ shopify_domain: string, token: string, ip: string }`
        - **Output**: `{ status: string, access_token: string, error: { code: string, message: string } | null }`
        - **Flow**: Validate OAuth token, check `ip_whitelist`, verify MFA via Auth0, generate JWT, cache in Redis Streams (`auth:{merchant_id}`), log in `audit_logs`, track via PostHog (`login_success`, 100% secure authentication).
- **GraphQL Query Examples**:
    - **Query: Verify Shopify OAuth Token**
        - **Purpose**: Validates merchant's Shopify OAuth token for secure access, used in `/v1/api/auth/login`.
        - **Query**:

```
query GetShopDetails {
  shop {
    id
    name
    email
    domain
  }
}
```

        - **Use Case**: Merchant Dashboard authenticates via Shopify OAuth, storing `api_token` in `merchants` (AES-256 encrypted) for secure access to Admin Module.
    - **Query: Fetch Admin User Permissions**
        - **Purpose**: Retrieves admin user permissions for RBAC enforcement, used in `/roles.v1/RolesService/GetPermissions`.
        - **Query**:

```
query GetAdminUser($id: ID!) {
  user(id: $id) {
    id
```

```
      metafield(namespace: "loyalnest", key: "permissions") {
        value
      }
    }
  }
```

    - **Variables**: `{ "id": "gid://shopify/User/123456789" }`
    - **Use Case**: Admin Module enforces RBAC by checking `roles.permissions` against `admin_users.metadata`, ensuring only authorized access (e.g., `admin:full`, `admin:analytics`).
  - **Query: Check Webhook Status for Idempotency**
    - **Purpose**: Verifies webhook subscription status to ensure idempotent processing, used in webhook handling.
    - **Query**:

```
query GetWebhookSubscriptions {
  webhookSubscriptions(first: 10) {
    edges {
      node {
        id
        topic
        endpoint {
          ... on WebhookHttpEndpoint {
            callbackUrl
          }
        }
        status
      }
    }
  }
}
```

    - **Use Case**: Admin Module validates webhook subscriptions (e.g., `orders/create`) to prevent duplicate processing, using Redis keys (`webhook:{merchant_id}:{event_id}`) for idempotency.
- **Service**: Auth Service (gRPC: `/auth.v1/*`, Dockerized).

8. Testing and Monitoring (Phase 1)

- **Goal**: Ensure reliability and performance. Success metric: 99%+ uptime, <1s alert latency, 80%+ test coverage, 95%+ Redis cache hit rate.
- **Automated Testing**: Jest (unit/integration for `PointsService`, `ReferralService`, `rfm-service` covering RFM calculations, nudges, `rfm_segment_deltas`), 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., `rfm-service`, Redis, PostgreSQL downtime) using Chaos Mesh, ensuring circuit breakers and DLQs maintain 99%+ uptime.

- **Monitoring Metrics**: API latency (<1s for `rfm-service`, `PointsService`), Redis cache hits (>95%), Bull queue delays (<5s), error rates (<1%) via Prometheus/Grafana. Error tracking with Sentry (`rfm_calculation_failed`, `export_failed`), centralized logging with Loki, AWS SNS alerts for rate limits.
- **Developer Tools**: `dev.sh` script for mock data, RFM simulation, and merchant referral testing, enhanced with Grok AI and GitHub Copilot/Cursor.
- **Database Design**:
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `system_alert`, `chaos_test_failed`, `rate_limit_alerted`, `rfm_calculation_failed`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"error_rate": 0.01, "chaos_type": "redis_downtime"}`.
  - **Indexes**: `idx_audit_logs_action` (btree: `action`).
  - **Backup Retention**: 90 days in Backblaze B2, encrypted with AES-256.
- **API Sketch**:
  - **GET** `/v1/api/monitoring` (REST) | gRPC `/admin.v1/AdminFeaturesService/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: Fetch Shop Order Metrics for Load Testing**
    - **Purpose**: Retrieves recent order data to monitor system performance during load testing, used in `/v1/api/monitoring`.
    - **Query**:

```
query GetShopOrders($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 (10,000 orders/hour) during k6 load tests, feeding into Prometheus/Grafana for latency and error rate metrics.
    - **Query: Check API Usage for Rate Limit Monitoring**
      - **Purpose**: Retrieves API usage data to track Shopify API rate limits, used in `/v1/api/monitoring`.
      - **Query**:

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

      - **Use Case**: Admin Module monitors API usage (2 req/s REST, 40 req/s GraphQL for Plus) to trigger AWS SNS alerts when nearing rate limits, logged in `audit_logs` and tracked via PostHog (`rate_limit_alerted`).
- **Service**: Admin Service (gRPC: `/admin.v1/*`, Dockerized).

9. Admin Module (Phase 1)

- **Goal**: Provide tools for platform management. Success metric: 90%+ task completion rate, <1s latency for real-time logs.
- **Features**: Overview (metrics, RFM segments via `rfm-service`, merchant timelines via Chart.js), Merchants (search, plan management, undo actions), Admin Users (RBAC via `roles-service`, MFA via Auth0), Logs (API/audit streaming via WebSocket, Loki + Grafana), Queue Monitoring (`QueuesPage.tsx` with Chart.js for `rfm-service` queues), Integration Health (Shopify, Klaviyo, Postscript).
- **RFM Configuration**: Configure thresholds, segments, and nudges via `RFMConfigPage.tsx`, calling `rfm-service` (`/rfm.v1/RFMService/UpdateThresholds`, `/rfm.v1/RFMService/GetNudges`). Supports persona-based defaults (e.g., “Pet Store”), A/B testing, and onboarding progress (`setup_tasks`).
- **Merchant Onboarding Wizard**: Gamified setup (points program, branding, RFM config, Slack community “LoyalNest Collective” join) with Polaris `ProgressBar`, badges, and PostHog tracking

(`onboarding_step_completed`, 80%+ completion).

- **Automated Email Flows**: Points updates, referral confirmations, GDPR notifications, RFM nudges via Klaviyo/Postscript/AWS SES (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)).
- **Database Design**:
  - **Table**: `merchants`
    - `merchant_id` (text, PK, NOT NULL): Unique ID.
    - `shopify_domain` (text, UNIQUE, NOT NULL): Shopify domain.
    - `plan_id` (text, FK → `plans`): Plan (e.g., Free: 300 orders, $29/mo: 500 orders, $99/mo: 1500 orders).
    - `onboarding_status` (jsonb): e.g., `{"steps_completed": ["points_config", "branding", "rfm_config"], "badge": "Setup Pro"}`.
    - `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**: `admin_users`
    - `admin_id` (text, PK, NOT NULL): Unique ID.
    - `username` (text, UNIQUE, NOT NULL): Username.
    - `password_hash` (text, AES-256 ENCRYPTED, NOT NULL): Bcrypt hash.
    - `metadata` (jsonb, AES-256 ENCRYPTED): e.g., `{"role": "admin:full", "mfa_enabled": true}`.
  - **Table**: `setup_tasks` (AdminFeatures)
    - `merchant_id`, `task_name`, `status`, `completed_at`: Tracks onboarding progress.
  - **Table**: `audit_logs`
    - `action` (text, NOT NULL): e.g., `merchant_added`, `onboarding_step_completed`, `rfm_config_updated`.
    - `actor_id` (text, FK → `admin_users` | NULL): Admin user.
    - `metadata` (jsonb): e.g., `{"plan_id": "basic", "step": "rfm_config"}`.
  - **Indexes**: `idx_merchants_shopify_domain` (btree: `shopify_domain`), `idx_admin_users_username` (btree: `username`), `idx_setup_tasks_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** `/admin/v1/merchants` (REST) | gRPC `/admin.v1/AdminCoreService/ListMerchants`
    - **Input**: `{ page: number, per_page: number, search: string }`
    - **Output**: `{ status: string, merchants: array, total: number, error: { code: string, message: string } | null }`
    - **Flow**: Query `merchants`, enforce RBAC via `/roles.v1/RolesService/GetPermissions`, cache in Redis Streams (`merchants:{page}`), log in `audit_logs`, track via PostHog (`merchants_listed`, 80%+ view rate).
  - **POST** `/admin/v1/rfm/config` (REST) | gRPC `/rfm.v1/RFMService/UpdateThresholds`
    - **Input**: `{ merchant_id: string, thresholds: object, segments: array, locale: string }`

        - **Output**: `{ status: string, preview: object, error: { code: string, message: string } | null }`
        - **Flow**: Update `program_settings.rfm_thresholds`, trigger preview via `/rfm.v1/RFMService/PreviewRFMSegments`, cache in Redis Streams (`rfm:preview:{merchant_id}`), log in `audit_logs`, track via PostHog (`rfm_config_saved`, 80%+ completion).
    - **WebSocket** `/admin/v1/logs` | gRPC `/admin.v1/AdminFeaturesService/StreamLogs`
        - **Input**: `{ merchant_id: string, actions: array }`
        - **Output**: Stream `{ logs: [{ action: string, metadata: object, created_at: string }] }`
        - **Flow**: Stream `api_logs`, `audit_logs` via WebSocket, cache in Redis Streams (`logs:{merchant_id}`), enforce RBAC, log in `audit_logs`, track via PostHog (`logs_viewed`, <1s latency).
- **GraphQL Query Examples**:
    - **Query: Fetch Merchant Details for Management**
        - **Purpose**: Retrieves merchant details for the Admin Module's merchant management interface, used in `/admin/v1/merchants`.
        - **Query**:

```
query GetMerchants($first: Int, $query: String) {
  shops(first: $first, query: $query) {
    edges {
      node {
        id
        name
        domain
        plan {
          displayName
        }
        metafield(namespace: "loyalnest", key:
"onboarding_status") {
          value
        }
      }
    }
  }
}
```

        - **Variables**: `{ "first": 10, "query": "domain:*.myshopify.com" }`
        - **Use Case**: Admin Module displays merchant list with `onboarding_status` from `merchants` table, enabling plan management and search functionality.
    - **Query: Update RFM Configuration**
        - **Purpose**: Updates merchant's RFM thresholds, used in `/admin/v1/rfm/config`.
        - **Query**:

```
mutation UpdateMetafield($input: MetafieldInput!) {
  metafieldStorefrontVisibilityCreate(input: {
```

```
      namespace: "loyalnest"
      key: "rfm_thresholds"
      ownerType: SHOP
    }) {
      metafieldStorefrontVisibility {
        id
      }
      userErrors {
        field
        message
      }
    }
    metafieldsSet(input: [$input]) {
      metafields {
        id
        namespace
        key
        value
      }
      userErrors {
        field
        message
      }
    }
}
```

- **Variables**:

```
{
  "input": {
    "namespace": "loyalnest",
    "key": "rfm_thresholds",
    "value": "{\"recency\": {\"R5\": 14, \"R4\": 30},
\"frequency\": {\"F5\": 5, \"F4\": 3}, \"monetary\": {\"M5\": 500,
\"M4\": 200}}",
    "ownerId": "gid://shopify/Shop/123456789",
    "type": "json"
  }
}
```

- **Use Case**: Admin Module updates `program_settings.rfm_thresholds` for RFM configuration, stored as a Shopify metafield and cached in Redis Streams (`rfm:preview:{merchant_id}`).

- **Query: Check Integration Health**
  - **Purpose**: Verifies the status of third-party integrations (e.g., Klaviyo, Postscript), used in Integration Health monitoring.
  - **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 (Shopify, Klaviyo, Postscript) by checking `appInstallations` status, logged in `audit_logs` for real-time updates.
- **Service**: Admin Service (gRPC: `/admin.v1/*`, Dockerized), RFM Service (gRPC: `/rfm.v1/*`, Dockerized).