Table of Contents

[概要: グループ15『習い事Prime』技術要素整理（PR/FAQ準拠） 1](#_Toc207855061)

[スコープ（PR/FAQ準拠のサービス機能） 2](#_Toc207855062)

[6分野の技術要素 3](#_Toc207855063)

[データモデル（PR/FAQ準拠） 7](#_Toc207855064)

[アーキテクチャ構成: 機能別フロー 11](#_Toc207855065)

[アーキテクチャ構成: 全体フロー 13](#_Toc207855066)

## 概要: グループ15『習い事Prime』技術要素整理（PR/FAQ準拠）

本ドキュメントは、PR/FAQで定義されたサービス機能を実現するための技術要素を6分野で整理したものです。最初に概要として、全体アーキテクチャ構成図を載せておく。ダイアグラム, 概略図

AI 生成コンテンツは誤りを含む可能性があります。

## スコープ（PR/FAQ準拠のサービス機能）

* **アカウント/認証**: 親子それぞれのアカウント作成、家族連携、ログイン/ログアウト（保護者/子/教室/管理者ロール）。
* **検索/閲覧**: 教室・先生の検索、一覧/詳細、画像表示、エリア・ジャンル絞り込み。検索・口コミ閲覧は無料。
* **サブスク/チケット**: 月額プラン課金、毎月のチケット付与、残高表示。
* **予約**: チケットを使用した体験レッスン予約、予約確認通知（メール）。一部教室では「先生を指定した予約」が可能。
* **受講確認**: 教室側による出席確定、ユーザー側の履歴確認。
* **口コミ**: 保護者用/子ども用の別枠口コミ投稿・閲覧。施設（教室）と指導者（先生）それぞれを評価可能。同一家族の口コミを並べて表示。
* **いいね/共有**: 親子が付けた「いいね」を家族内で共有・比較できるビューを提供。
* **メッセージ**: 予約後に教室側とユーザーがやり取りできるメッセージ（スレッド）機能。
* **ポイント/紹介**: 口コミ投稿でポイント付与、紹介（リファラル）機能。一定ポイントでチケットに交換可能な仕組み。
* **決済/通知**: Stripeによる月額課金/カード管理、適切タイミングでのメール/通知配信。

（将来/任意）例: AI推薦、決済の複数通貨対応、ネイティブアプリ など。

## 6分野の技術要素

### Database

* **リレーショナルDB（推奨: Amazon RDS for PostgreSQL + RDS Proxy + PostGIS）**
  + 理由: 予約/チケット消費の整合性、親子/世帯の参照整合性、地理検索（PostGIS）、メッセージ/ポイントの履歴性
  + スキーマ詳細は末尾の「データモデル（PR/FAQ準拠）」を参照
  + 特徴（CS観点のタグ）:
    - PostgreSQL: リレーショナルデータベース, ACID, スキーマ, 参照整合性, SQL, インデックス
    - RDS Proxy: 接続プーリング, コネクション管理, 可用性, スケーラビリティ, マネージド
    - PostGIS: 地理空間拡張, 空間インデックス(GiST), 近傍検索, 半径検索, 距離計測

### Security

* **認証/認可**: Amazon Cognito（ユーザープール、グループ/カスタムクレームで親/子/教室/管理者ロール）。JWTをAPI層で検証
* **通信/暗号化**: ACMのTLS証明書、RDS暗号化（KMS）、S3バケット暗号化、Secrets ManagerでDB資格情報管理
* **ネットワーク保護**: AWS WAF（ボット/レート制御）、AWS Shield（DDoS）、最小権限IAM
* **コンテンツ健全性**: 口コミは「予約/受講実績ユーザー」のみ投稿可。通報フラグ＋管理者レビュー（初期は手動対応）。将来的な自動モデレーション（Comprehend/rekognition）は任意
* **公開/非公開制御**: 検索/口コミ閲覧はパブリック（レート制御あり）、予約/投稿/メッセージは認証必須。メッセージは会話参加者のみアクセス可
* **決済保護**: Stripe Webhook署名検証（Stripe-Signature）、Idempotency-Key運用、PCI DSSはStripeに委譲

### Network

* **配信/エッジ**: Amazon CloudFront（S3フロントエンド配信、画像配信、HTTP/2, HTTP/3）
* **APIエントリ**: Amazon API Gateway（REST + WebSocket）。バックエンドはAWS Lambda
* **VPC設計**: RDSはプライベートサブネット、LambdaはVPC内からRDS Proxy経由で接続
* **DNS**: Route 53（独自ドメイン。ACMと統合）
* **決済Webhooks**: POST /webhooks/stripe（API Gateway → Lambda）。署名検証、再送/順不同に耐える冪等設計

### Application Development

* **フロントエンド**: Next.js（静的出力）
  + 認証: Cognito Hosted UI/Amplify Auth、またはOAuthフロー直接実装
  + UI: 検索→詳細（教室/先生）→予約→メッセージ→口コミ投稿の導線。家族内の「いいね」共有ビュー、ポイント残高/交換。
* **バックエンド**: AWS Lambda（Node.js 20）+ API Gateway（REST JSON + WebSocket）
  + 主要API（例）:
    - POST /auth/callback（Cognito連携。初回プロビジョニング）
    - GET /schools / GET /schools/{id} / GET /instructors/{id}
    - GET /classes（エリア/ジャンル/日付で検索）
    - GET /me/tickets / POST /bookings / GET /me/bookings / POST /bookings/{id}/cancel（lesson\_schedules.instructor\_id を考慮）
    - POST /reviews/parent / POST /reviews/child（target\_type/target\_id 指定） / GET /schools/{id}/reviews
    - POST /attendances/{bookingId}（教室権限）
    - いいね: POST /schools/{id}/like / DELETE /schools/{id}/like / GET /families/{id}/likes
    - メッセージ（REST）: POST /conversations / GET /conversations / GET /conversations/{id}/messages / POST /conversations/{id}/messages
    - メッセージ（WebSocket）: 接続/切断/送受信ハンドラ（会話参加者のみ配信）
    - ポイント: GET /me/points / POST /points/redeem（ポイント→チケット交換）
    - 紹介: POST /referrals（招待コード発行）/ POST /referrals/accept / GET /me/referrals
    - 決済(Stripe): POST /billing/checkout-session（Checkoutセッション作成）/ POST /billing/portal-session（顧客ポータル）/ POST /webhooks/stripe（Webhook受信）
  + 入力検証: Zod/Yup。レート制御: API Gateway + WAF
  + トランザクション: チケット残高消費と予約作成を同一トランザクションで実装
  + 非同期処理: EventBridgeで月次チケット付与、口コミ投稿イベントに応じたポイント付与
  + 決済フロー（Stripe）:
    - 顧客作成: 初回購入でStripe Customer作成→families.stripe\_customer\_id保存
    - 購入: フロント→/billing/checkout-session→Stripe Checkout（サブスク）
    - 成立: Webhook checkout.session.completed→subscriptions作成/更新、初回チケット付与
    - 更新: invoice.payment\_succeeded→次期チケット付与、invoice.payment\_failed→statusをon\_hold等に変更
    - 解約: customer.subscription.deleted→契約停止・残チケット処理
    - 冪等: payments.stripe\_payment\_intent\_id と webhook\_events.stripe\_event\_id で重複防止
  + メール/通知タイミング:
    - 予約作成: 確認メール（保護者/教室）
    - レッスン前日: リマインド（EventBridge Scheduler）
    - 予約キャンセル: キャンセル通知
    - 受講後: 口コミ依頼
    - メッセージ受信: 既読前はWebSocketリアルタイム通知、未読が一定時間超でメール
    - サブスク開始/更新/失敗: 成功/失敗メール、カード更新依頼
    - チケット残量閾値: お知らせメール
    - 紹介/ポイント: 成立・交換通知

### Infrastructure

* **IaC**: AWS CDK（TypeScript）。API Gateway(REST/WebSocket), Lambda, RDS, S3, CloudFront, Cognito, WAF, EventBridge, SES, SNS をコード化
* **CI/CD**:
  + バックエンド: GitHub Actions → CDKデプロイ → Lambda更新
  + フロントエンド: GitHub Actions → S3静的サイト配信 → CloudFront無効化
* **監視/運用**: CloudWatch Logs/Alarms, X-Ray（トレース）, Health Dashboard
* **可観測性**: 主要メトリクス（予約成功率、API 4xx/5xx、レイテンシ、DB接続数、チケット消費失敗率、WebSocket接続数/エラー、ポイント付与イベント処理遅延）
* **通知基盤**: Amazon SES（メール）, Amazon SNS（トピック配信/モバイルプッシュ連携）, Amazon SQS（DLQ/再試行）
* **決済/通知運用**: Secrets Manager（STRIPE\_API\_KEY/STRIPE\_WEBHOOK\_SECRET）、SESドメイン認証、Webhook LambdaのDLQ、EventBridge Scheduler（リマインド/ダイジェスト）

### Storage

* **オブジェクトストレージ**: Amazon S3（教室/先生画像、口コミ画像）
  + アップロードはフロント→PUTプレサインドURL（サイズ/拡張子制限）
  + 配信はCloudFront。キャッシュ無効化は運用フローで最小化

## データモデル（PR/FAQ準拠）

* 概要スキーマ（詳細は実装時にマイグレーションで管理）
* **users**: id, type(parent|child|school\_owner|admin), email, name, created\_at
  + 役割: 利用者（保護者/子/教室オーナー/管理者）
  + 列の意味: id(ユーザーID), type(ロール), email(連絡先), name(表示名), created\_at(作成時刻)
  + 主な制約: PK(id), UNIQUE(email)
* **families**: id, parent\_user\_id, stripe\_customer\_id
  + 役割: 家族（世帯）
  + 列の意味: id(家族ID), parent\_user\_id(保護者ユーザーID), stripe\_customer\_id(Stripe顧客ID)
  + 主な制約: PK(id), FK(parent\_user\_id→users.id), UNIQUE(stripe\_customer\_id)
* **family\_members**: family\_id, child\_user\_id
  + 役割: 家族と子ユーザーの関連
  + 列の意味: family\_id(家族ID), child\_user\_id(子ユーザーID)
  + 主な制約: PK(family\_id, child\_user\_id), FK(family\_id→families.id), FK(child\_user\_id→users.id)
* **schools**: id, name, area, category, description, image\_key, location
  + 役割: 教室（施設）
  + 列の意味: id(教室ID), name(名称), area(エリア), category(ジャンル), description(紹介文), image\_key(S3キー), location(位置Point)
  + 主な制約: PK(id)
* **instructors**: id, school\_id, name, profile, image\_key
  + 役割: 指導者（先生）
  + 列の意味: id(先生ID), school\_id(所属教室), name(氏名), profile(紹介), image\_key(S3キー)
  + 主な制約: PK(id), FK(school\_id→schools.id)
* **classes**: id, school\_id, title, capacity, duration\_min
  + 役割: クラス（講座の種類）
  + 列の意味: id(クラスID), school\_id(教室ID), title(タイトル), capacity(定員), duration\_min(所要分)
  + 主な制約: PK(id), FK(school\_id→schools.id)
* **lesson\_schedules**: id, class\_id, instructor\_id, start\_at, end\_at
  + 役割: 開催スケジュール（日時枠）
  + 列の意味: id(枠ID), class\_id(クラスID), instructor\_id(担当先生ID), start\_at(開始), end\_at(終了)
  + 主な制約: PK(id), FK(class\_id→classes.id), FK(instructor\_id→instructors.id), UNIQUE(class\_id, start\_at)
* **subscriptions**: id, family\_id, plan, status, current\_period\_start, current\_period\_end, stripe\_subscription\_id
  + 役割: 月額契約
  + 列の意味: id(契約ID), family\_id(家族ID), plan(プラン), status(状態), current\_period\_start(期間開始), current\_period\_end(期間終了), stripe\_subscription\_id(Stripe契約ID)
  + 主な制約: PK(id), FK(family\_id→families.id), UNIQUE(stripe\_subscription\_id)
* **payments**: id, family\_id, stripe\_payment\_intent\_id, amount, currency, status, created\_at, meta
  + 役割: 個別決済（初回/更新/失敗）
  + 列の意味: id(決済ID), family\_id(家族ID), stripe\_payment\_intent\_id(Stripe PI), amount(金額), currency(通貨), status(状態), created\_at(発生), meta(付随情報)
  + 主な制約: PK(id), FK(family\_id→families.id), UNIQUE(stripe\_payment\_intent\_id)
* **webhook\_events**: id, stripe\_event\_id, type, payload, received\_at, processed\_at
  + 役割: Stripe Webhookイベント受信
  + 列の意味: id(受信ID), stripe\_event\_id(イベントID), type(種別), payload(生データ), received\_at(受信), processed\_at(処理完了)
  + 主な制約: PK(id), UNIQUE(stripe\_event\_id)
* **ticket\_balances**: id, family\_id, month, balance
  + 役割: 月次チケット残高
  + 列の意味: id(残高ID), family\_id(家族ID), month(対象月), balance(残枚数)
  + 主な制約: PK(id), FK(family\_id→families.id), UNIQUE(family\_id, month)
* **bookings**: id, user\_id(child), schedule\_id, status, consumed\_tickets, created\_at
  + 役割: 子ユーザーの予約
  + 列の意味: id(予約ID), user\_id(子ユーザーID), schedule\_id(スケジュールID), status(状態), consumed\_tickets(消費枚数), created\_at(作成)
  + 主な制約: PK(id), FK(user\_id→users.id), FK(schedule\_id→lesson\_schedules.id), UNIQUE(user\_id, schedule\_id)
* **attendances**: id, booking\_id, status(attended|absent)
  + 役割: 出欠
  + 列の意味: id(出欠ID), booking\_id(予約ID), status(出席/欠席)
  + 主な制約: PK(id), FK(booking\_id→bookings.id)
* **reviews\_parent / reviews\_child**: id, booking\_id, target\_type(school|instructor), target\_id, rating, comment, created\_at
  + 役割: 口コミ（保護者/子ども）
  + 列の意味: id(口コミID), booking\_id(予約ID), target\_type(対象種別), target\_id(対象ID), rating(評点), comment(本文), created\_at(投稿)
  + 主な制約: PK(id), FK(booking\_id→bookings.id), UNIQUE(booking\_id)
* **likes**: id, user\_id, school\_id, created\_at
  + 役割: いいね（お気に入り）
  + 列の意味: id(いいねID), user\_id(ユーザーID), school\_id(教室ID), created\_at(付与時刻)
  + 主な制約: PK(id), FK(user\_id→users.id), FK(school\_id→schools.id), UNIQUE(user\_id, school\_id)
* **conversations**: id, booking\_id, family\_id, school\_id, created\_at
  + 役割: メッセージ会話スレッド
  + 列の意味: id(会話ID), booking\_id(予約ID), family\_id(家族ID), school\_id(教室ID), created\_at(作成)
  + 主な制約: PK(id), FK(booking\_id→bookings.id), FK(family\_id→families.id), FK(school\_id→schools.id)
* **messages**: id, conversation\_id, sender\_user\_id, body, created\_at, read\_at
  + 役割: 会話内メッセージ
  + 列の意味: id(メッセージID), conversation\_id(会話ID), sender\_user\_id(送信者ID), body(本文), created\_at(送信), read\_at(既読)
  + 主な制約: PK(id), FK(conversation\_id→conversations.id), FK(sender\_user\_id→users.id)
* **points\_transactions**: id, family\_id, type(review|referral|redeem), amount, created\_at, meta
  + 役割: ポイント取引履歴
  + 列の意味: id(取引ID), family\_id(家族ID), type(種別), amount(増減), created\_at(記録), meta(付随情報)
  + 主な制約: PK(id), FK(family\_id→families.id)
* **referrals**: id, referrer\_family\_id, code, referred\_family\_id, status(pending|accepted), created\_at
  + 役割: 紹介（リファラル）
  + 列の意味: id(リファラルID), referrer\_family\_id(紹介元家族ID), code(招待コード), referred\_family\_id(紹介先家族ID), status(状態), created\_at(作成)
  + 主な制約: PK(id), FK(referrer\_family\_id→families.id), FK(referred\_family\_id→families.id), UNIQUE(code)

## アーキテクチャ構成: 機能別フロー

* 検索/閲覧（学校・先生・地理）
  + フロント（Next.js）→ API Gateway(REST) → Lambda → RDS(PostgreSQL/PostGIS)
  + キーワードはpg\_trgm+GIN、地理はPostGIS（ST\_DWithin/Distance）。schools\_stats等の集約でランキング
* 画像アップロード/配信
  + フロント→バックエンドでプリサインドURL発行→ S3にPUT → CloudFrontから配信
* 認証/認可
  + Cognito Hosted UIでログイン→JWTをAPIに付与→Authorizer検証→Lambda実行、ロール/属性で権限制御
* 予約（チケット消費）
  + POST /bookings → LambdaがRDSトランザクション：残高検証→二重予約チェック(UNIQUE)→残高減算→予約作成
  + 成功後にSESで確認メール、EventBridge Schedulerに前日リマインド登録
* 出席確定/口コミ依頼
  + 教室側POST /attendances/{bookingId}→RDS更新→受講後に口コミ依頼メール（SES）
* メッセージ（チャット）
  + スレッド/履歴はREST、送受信はWebSocket。会話参加者のみに配信。未読継続でメール通知
* 口コミ（親/子）
  + POST /reviews/(parent|child)→RDSに保存（各UNIQUE(booking\_id)）
* いいね/共有
  + POST /schools/{id}/like→RDS（UNIQUE(user\_id, school\_id)）。家族共有はfamily\_idで集約表示
* ポイント/紹介
  + イベント契機でpoints\_transactions記録。残高は集計ビューで提供
* 決済/サブスク（Stripe）
  + フロント→/billing/checkout-session→Stripe Checkout
  + Webhook(/webhooks/stripe)で checkout.session.completed/invoice.\*を受信→冪等処理→subscriptions/payments更新とチケット付与
  + 失敗時はメールでカード更新依頼
* 通知（メール/プッシュ/SMS）
  + Lambda→SESで送信。ファンアウトが必要な場合はSNSトピック経由。失敗はSQS DLQに退避
* 運用/監視
  + CloudWatch Logs/Alarms、X-Rayトレース。機能別ダッシュボード（検索/予約/メッセージ/決済）

## アーキテクチャ構成: 全体フロー

* 注記: S3はVPCの外にあるリージョンサービスです（バケットはVPC内に存在しません）。フロント配信はCloudFront→S3（OAC/署名URL）で私有化しアクセス。今のところ無いが、将来的に機能拡張するにあたりバックエンドからアクセスする場合はVPCエンドポイント（Gateway）経由でプライベートアクセスする。
* 再掲): 以下はアーキテクチャ構成図である。以下のmermaidコードを忠実に視覚化した図となっている。

ダイアグラム, 概略図

AI 生成コンテンツは誤りを含む可能性があります。

```mermaid

flowchart LR  
 subgraph Client  
 U["Browser/SPA (Next.js)"]  
 end  
  
 %% 非AWSサービス  
 subgraph NonAWS["非AWSサービス"]  
 STRIPE["Stripe"]  
 end  
  
 %% AWSサービス/VPC外（エッジ/グローバル）  
 subgraph GlobalEdge["AWSサービス / VPC外（エッジ/グローバル）"]  
 R53["Route 53 (DNS; Global)"]  
 WAF["AWS WAF (CloudFront scope; Global)"]  
 CF["CloudFront (Edge)"]  
 SHIELD["AWS Shield (Global)"]  
 end  
  
 %% AWSサービス/VPC外（リージョン/マネージド）  
 subgraph RegionalManaged["AWSサービス / VPC外（リージョン/マネージド）"]  
 S3["S3 Private Bucket (静的資産/画像; Regional)"]  
 COG["Amazon Cognito (User Pool: Regional / Hosted UI delivered via Edge)"]  
 SES["Amazon SES (メール; Regional)"]  
 SNS["Amazon SNS (ファンアウト; Regional)"]  
 SQS["Amazon SQS (DLQ/再試行; Regional)"]  
 EVS["EventBridge Scheduler (Regional)"]  
 SM["Secrets Manager (Regional)"]  
 CWL["CloudWatch Logs (Regional)"]  
 CWM["CloudWatch Metrics (Regional)"]  
 ALM["CloudWatch Alarms (Regional)"]  
 XRAY["AWS X-Ray (Regional)"]  
 HLTH["AWS Health (Global/Regional)"]  
 APIR["公開エンドポイント\_API Gateway (REST; Regional)"]  
 APIWS["公開エンドポイント\_API Gateway (WebSocket; Regional)"]  
 end  
  
 %% AWSサービス/VPC内（プライベート）  
 subgraph VPC["AWSサービス / VPC内（プライベート）"]  
 LREST["Lambda (REST ハンドラ)"]  
 LWS["Lambda (WS ハンドラ)"]  
 LWH["Lambda (Webhook ハンドラ)"]  
 LJ["Lambda Jobs (リマインド/集計)"]  
 VPCE\_SM["VPC Endpoint(Interface Endpoint): Secrets Manager"]  
 VPCE\_LOGS["VPC Endpoint(Interface Endpoint): CloudWatch Logs"]  
 VPCE\_MON["VPC Endpoint(Interface Endpoint): CloudWatch Metrics"]  
 VPCE\_SES["VPC Endpoint(Interface Endpoint): Amazon SES"]  
 VPCE\_SNS["VPC Endpoint(Interface Endpoint): Amazon SNS"]  
 VPCE\_SQS["VPC Endpoint(Interface Endpoint): Amazon SQS"]  
 NATGW["NAT Gateway"]  
 RPROXY["RDS Proxy"]  
 RDS["RDS for PostgreSQL (Multi-AZ + PostGIS)"]  
 end  
  
 %% DNS: クエリ/応答（案内のみ）  
 U -->|"DNS query/answer"| R53  
  
 %% フロント配信とAPIエントリ（エッジ→境界→内部）  
 SHIELD -->|"DDoS protection"| R53  
 SHIELD -->|"DDoS protection"| CF  
 SHIELD -->|"DDoS protection"| APIR  
 SHIELD -->|"DDoS protection"| APIWS  
  
 R53 --> WAF  
 WAF --> CF  
 CF -->|"OAC/署名"| S3  
 CF --> APIR  
 CF --> APIWS  
  
 %% アップロード  
 U <--> S3  
  
 %% API → Lambda → DB  
 APIR --> LREST  
 APIWS --> LWS  
 LREST --> RPROXY --> RDS  
 LWS --> RPROXY --> RDS  
 %% 可観測性（トレース/メトリクス/アラーム）  
 LREST -->|"Trace"| XRAY  
 LWS -->|"Trace"| XRAY  
 LWH -->|"Trace"| XRAY  
 LJ -->|"Trace"| XRAY  
 LREST -->|"EMF/PutMetricData"| CWM  
 LWS -->|"EMF/PutMetricData"| CWM  
 LWH -->|"EMF/PutMetricData"| CWM  
 LJ -->|"EMF/PutMetricData"| CWM  
 LREST -->|"stdout"| CWL  
 LWS -->|"stdout"| CWL  
 LWH -->|"stdout"| CWL  
 LJ -->|"stdout"| CWL  
 CWM --> ALM  
 ALM --> SNS  
 HLTH --> SNS  
  
 %% VPC内からのプライベート連携（Interface/Gateway Endpoints 経由）  
 LREST --> VPCE\_SM --> SM  
 LWS --> VPCE\_SM --> SM  
 LWH --> VPCE\_SM --> SM  
 LREST --> VPCE\_LOGS --> CWL  
 LWS --> VPCE\_LOGS --> CWL  
 LREST --> VPCE\_MON --> CWM  
 LWS --> VPCE\_MON --> CWM  
  
 %% 認証（Cognito: Hosted UIはEdge配信、User PoolはRegional）  
 U -->|"Auth (OIDC/OAuth2)"| COG  
  
 %% 通知/ファンアウト（外向き）  
 LREST --> VPCE\_SES --> SES  
 LREST --> VPCE\_SNS --> SNS  
 SES --> VPCE\_SQS --> SQS  
 SNS --> VPCE\_SQS --> SQS  
  
 %% 決済（外部）とWebhook受信  
 LREST --> NATGW --> STRIPE  
 STRIPE -->|"Webhook"| APIR  
 APIR --> LWH --> RPROXY --> RDS  
 STRIPE <--> U   
  
 %% バッチ/リマインド  
 EVS --> LJ --> VPCE\_SES --> SES  
 EVS --> LJ --> VPCE\_SNS --> SNS

```

* 外部サービス
  + Stripe（外部決済）
* VPC外（エッジ/グローバル）
  + Route 53（独自ドメイン; Global DNS）
  + CloudFront（エッジ配信: フロント/画像配信）
  + AWS WAF（CloudFrontスコープ; エッジ評価）
* VPC外（リージョン/マネージド）
  + Amazon S3（オブジェクトストレージ; Regional）
  + Amazon Cognito（User Pool: Regional / Hosted UI: Edge配信）
  + Amazon SES / SNS / SQS（メール / 通知 / DLQ; Regional）
  + Amazon EventBridge Scheduler（Regional）
  + AWS Secrets Manager（シークレット管理; Regional）
  + Amazon CloudWatch Logs / Metrics（監視; Regional）
* VPC内（プライベート）
  + サブネット: アプリ用プライベートサブネット、DBサブネット（別AZ）
  + セキュリティ: セキュリティグループ、NACL、VPCエンドポイント（Secrets Manager/CloudWatch Logs/CloudWatch Metrics/SES/SNS/SQS）
  + 送信経路: NAT Gateway（外部SaaS: Stripe 等へのアウトバウンド）
  + RDS for PostgreSQL（マルチAZ, PostGIS）+ RDS Proxy
  + Lambda（VPC接続: REST/メッセージWS/決済Webhook/ジョブ(バッチ処理など)）
* 監視/運用の組み込み
  + トレース: Lambda → X-Ray（SDK/統合）
  + ログ/メトリクス: Lambda 標準出力 → CloudWatch Logs、EMF/PutMetricData → CloudWatch Metrics
  + アラーム/通知: CloudWatch Alarms → SNS（運用通知/ページャ連携）
  + プラットフォーム状況: AWS Health（イベント通知/メンテ情報）
* 公開エンドポイント（VPC外）
  + AWS Shield: DDoS緩和
* データフロー（高レベル）

1. DNS解決 → 静的配信（フロント/画像閲覧）/API 呼び出し（REST/WS）: クライアント → Route 53（A/AAAA/CNAME）→ クライアント→ WAF → CloudFront → S3/[API Gateway...]
2. API 呼び出し（REST/WS）:

1.1) DBアクセス: クライアント → Route 53 → クライアント → WAF → CloudFront → API Gateway → Lambda → RDS Proxy → RDS

1. 認証（Cognito）: ブラウザ → Cognito Hosted UI（Cognitoドメイン; 自社WAFは経由しない）→ JWT 取得 → 以降の API は 1) 経路
2. 画像 （0) の拡張）:
   * アップロード:
   1. リクエスト: ブラウザ → ROUTE53 → CF API Gateway → Lambda「fileName / contentType / 期待サイズ などを送る」
   2. プリサインドURL発行: Lambda が S3 SDK で プリサインドURL を生成「例：有効期限300秒、キーuploads/user123/abc.jpg、Content-Type固定、最大サイズ制約、ACL禁止 等」
   3. 直PUT: ブラウザ → S3（プリサインドURL; CloudFront/WAF は経由しない）
   * 配信（閲覧）: ブラウザ → WAF → CloudFront → S3（1 と同一路）
3. 決済（1) の拡張）:
   * Checkout Session 作成(API) 「目的：フロントから『決済したい』→ サーバーが Stripe の Checkout Session を作成して URL/ID を返す」: ブラウザ → Route53(DNS) → Shield → CloudFront(WAFスコープ) → API Gateway(Regional) → Lambda(LREST) → VPCE\_SM → Secrets Manager（Stripe鍵取得） → Lambda(LREST) →（VPC内なら NAT →）Internet → Stripe（ブラウザに checkout\_url / session\_id を返却①） 注意: 金額はサーバー側で再計算／Idempotency-Key 付与
   * ユーザーリダイレクト/完了: ブラウザ ↔ Stripe(Hosted)（① で返却された checkout\_url に遷移・カード入力・決済実行）→ success\_url / cancel\_url にリダイレクト（ブラウザ → Route53 → Shield → CloudFront → フロント） 任意（推奨）: 検証APIで暫定確認 → ブラウザ → Route53 → Shield → CloudFront(WAF) → API Gateway → Lambda(LREST) → Stripe（Session/PaymentIntent 再取得） 注意: 最終確定は Webhook 側で行う
   * Webhook 受信: Stripe → Shield → API Gateway(Regional; 本システムではCFにのみWAFをアタッチしているが、ベストプラクティスとしては、このエンドポイントにも WAF を“直接”付与することを強く勧める) → Lambda(LWH) → VPCE\_SM → Secrets Manager（endpoint secret 取得） → Lambda(LWH) 署名検証 → RDS Proxy → RDS（冪等更新） →（必要に応じて SNS/SQS へファンアウト）→ 2xx 即返 注意: event.id の冪等記録／重い処理は非同期（SQS/DLQ）／CloudFrontは挟まない
4. 通知（内部アウトバウンド）:
   * Lambda(LREST) → VPCE\_SES → SES（メール）→ SQS（DLQ）
   * Lambda(LREST) → VPCE\_SNS → SNS（ファンアウト）→ SQS（DLQ）
   * 外向き通信のため WAF/CloudFront は関与しない
5. バッチ/リマインド（内部イベント）:
   * EventBridge Scheduler → Lambda（ジョブ）→ VPCE\_SES → SES
   * EventBridge Scheduler → Lambda（ジョブ）→ VPCE\_SNS → SNS
   * 必要に応じて内部で API（1) 経路）や RDS を呼び出し

* 運用/可観測性
  + CloudWatch Logs/Alarms, X-Ray, Health Dashboard, Secrets Manager（Stripe鍵/DB資格情報）