オージス総研

# オープンAPIを支える API公開プロセスと管理の勘所

株式会社オージス総研事業開発本部
クラウドインテグレーションサービス部
齋藤 伸也（Saito_Shinya@ogis-ri.co.jp）

# 自己紹介

- 齋藤伸也 (Saito_Shinya@ogis-ri.co.jp)
- 株式会社オージス総研
  - クラウドインテグレーションサービス部所属
- インテグレーションアーキテクト / APIテクニカルコンサルタント
- システム連携やAPI構築プロジェクトに参画

オージス総研

# API公開プロセス

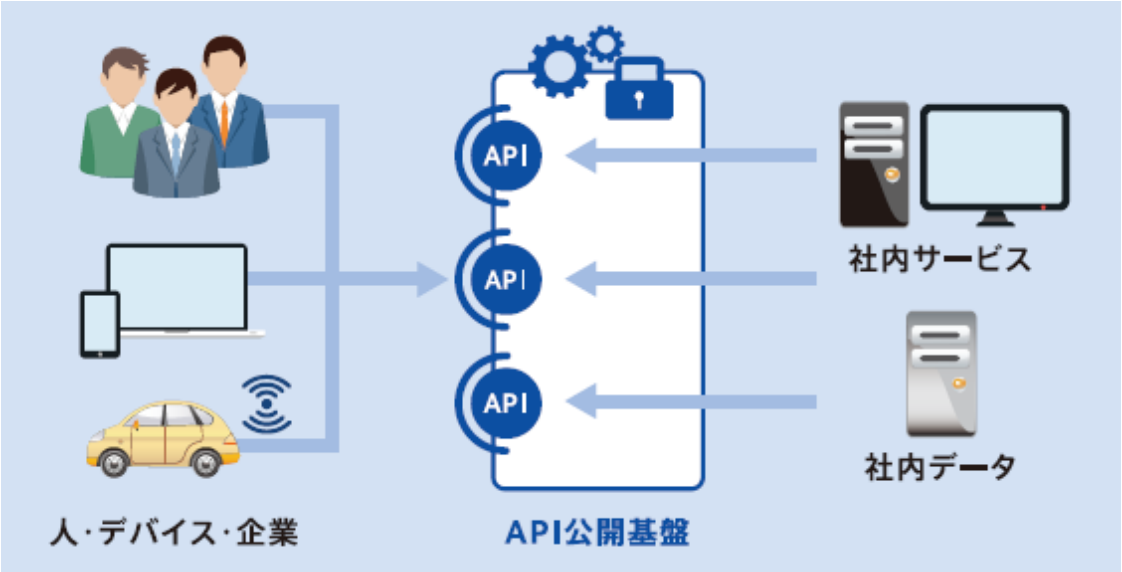
# API公開を成功させるには変化への素早い対応が重要

オージス総研

- デジタルビジネスは変化が激しく予測が難しい

- APIに求められる変化

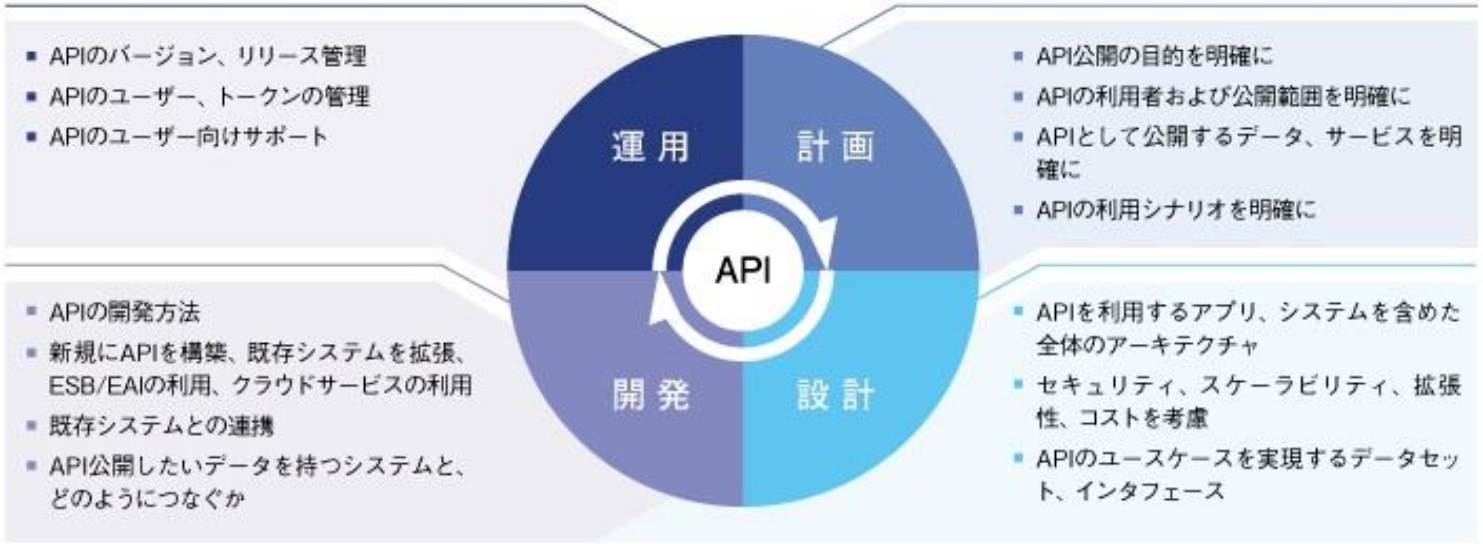
  - 既存のデータやサービスを拡張
  - 新しいデバイスや新しいビジネスパートナーの増加



# API公開のライフサイクル

- API公開は、一度きりの取り組みではない
- デジタルビジネスの成長、変化にあわせAPIを改修し、バージョンアップすることが必要

⇒ **ライフサイクルを素早く、しっかり回していくこと**が重要



# API公開のライフサイクル:計画

オージス総研

## ◘ API公開の計画で重要になるポイント

- API公開の目的を明確にする
- APIの利用者および公開範囲を明確にする
- APIとして公開するデータ、サービスを明確にする
- APIの利用シナリオを明確にする

API公開範囲の種類について

**プライベート**

メリット： 様々なクライアントから共通的に利用可能なモジュールを提供できる。

例: モバイル向けのバックエンドAPI

**パートナー**

メリット: パートナーとの新規協業、立ち上げの迅速化ができる。

例: 取引先、代理店向けのカタログAPI

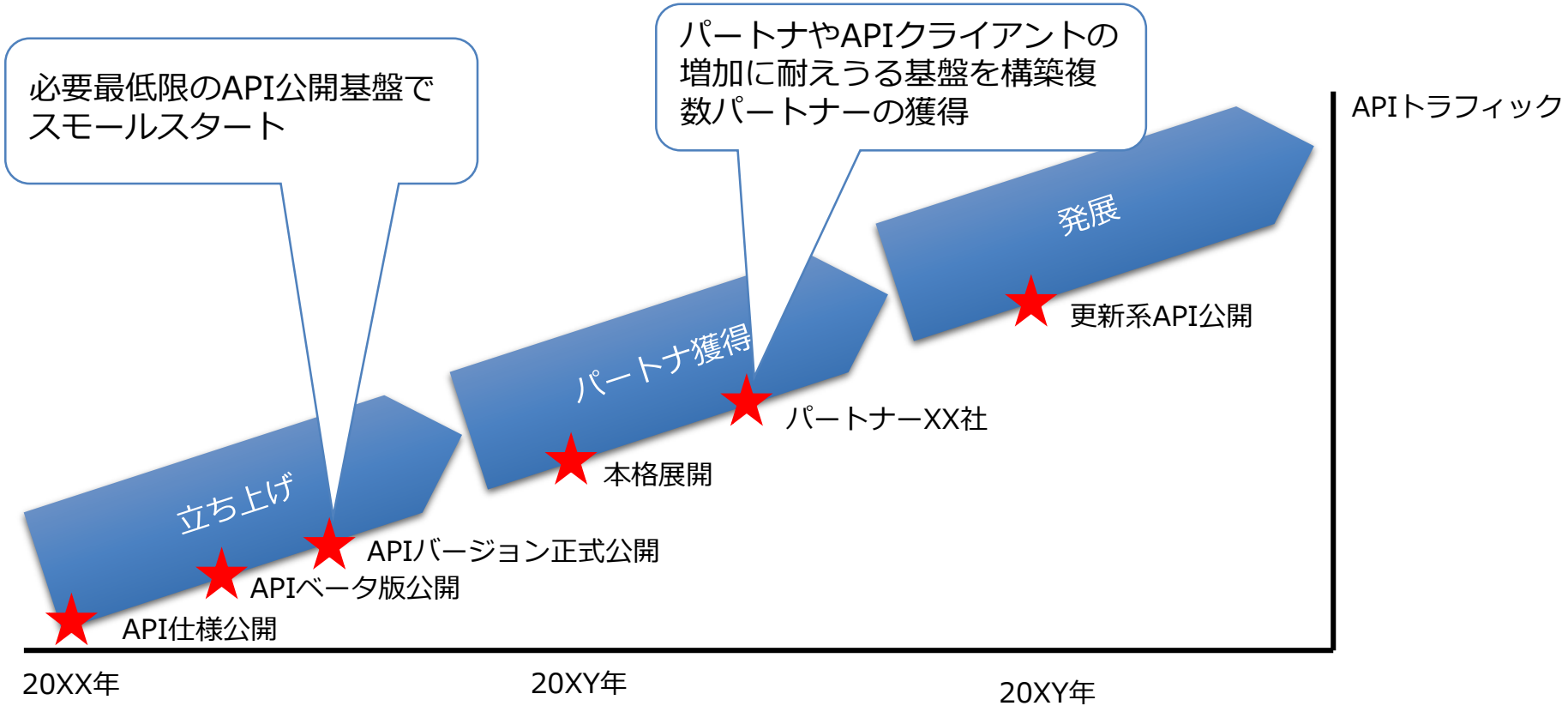
**パブリック**

メリット: ビジネスをプラットフォーム化することを実現できる。

例: オープンに公開されているMap API

# 例：製造業様API公開ロードマップ

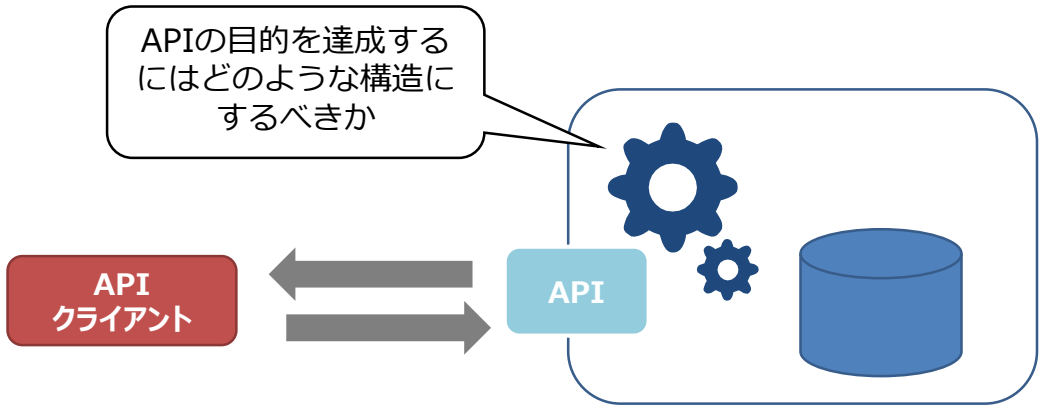
- IoT対応した製品を中心としたエコシステムを構築することを上位目標にすえ、APIをエコシステム拡大の手段としてロードマップを作成



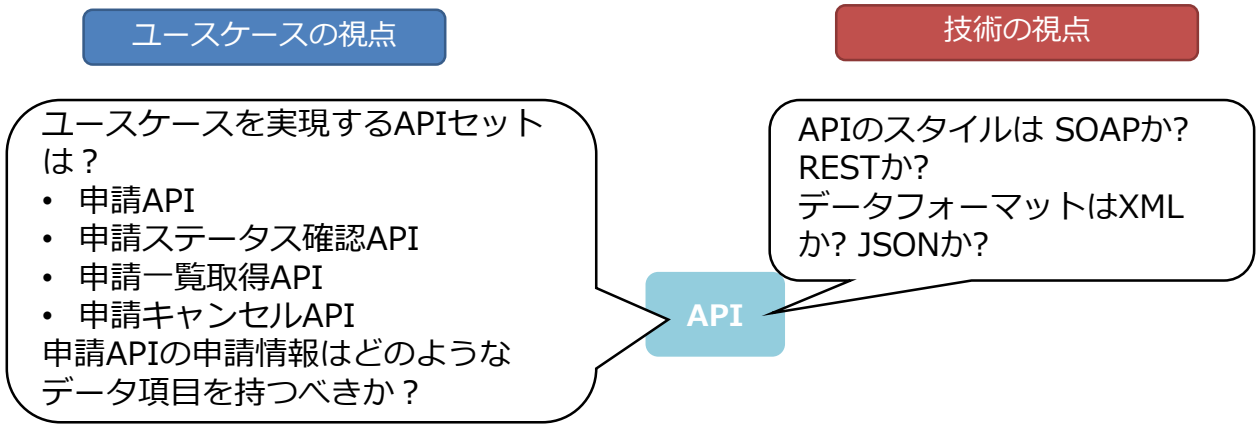
# API公開のライフサイクル:設計

- API公開の設計で重要になるポイント
  - APIを利用するアプリ、システムを含めた全体のアーキテクチャ
    → セキュリティ、スケーラビリティ、拡張性、コストを考慮する
  - APIのユースケースを実現するデータセット、インタフェース
    → ユーザ視点のデータセット、標準的なAPIスタイルなどユーザの利用しやすさを考慮する

アーキテクチャ設計




インタフェース設計

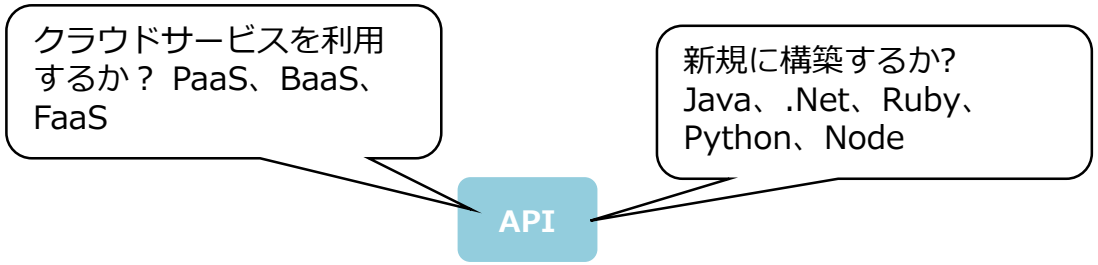


# API公開のライフサイクル:開発

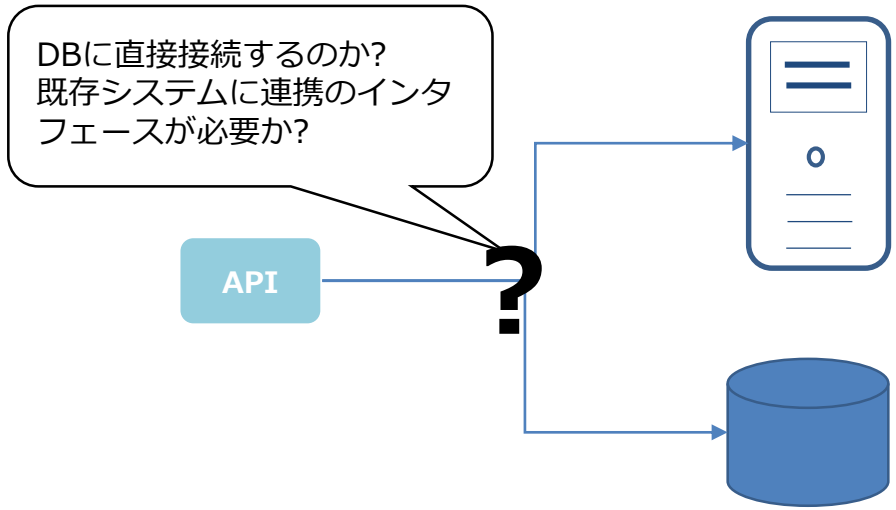
オージス総研

- API公開の開発で重要になるポイント
  - APIの開発方法
    - →新規にAPIを構築、既存システムを拡張、ESB/EAIなどの連携ミドルウェアの利用、クラウドサービスの利用
  - 既存システムとの連携
    - →API公開したいデータを持つシステムと、どのようにつなぐか

何をつかって開発する？




どうやって連携する？



# API公開のライフサイクル:運用

オージス総研

- API公開の運用で重要になるポイント
  - APIのバージョン、リリース管理
  - APIのユーザ、契約管理
  - APIのユーザ向けサポート
  - APIの監視、障害対応

APIの機能追加やデータ項目変更などの管理する

| | バージョン | ライフサイクル |
|---|---|---|
| ユーザプロファイル変更API | 1.2 | 公開中 |
| サービス取得API | 0.1 | 開発中 |

APIユーザや管理
APIを利用するために
トークンの管理

ユーザ

契約やAPI利用の課金情報の管理

API利用契約

APIの使い方を理解するためのドキュメント、SDK


開発者

# 事例：API公開・運用ユースケース

オージス総研

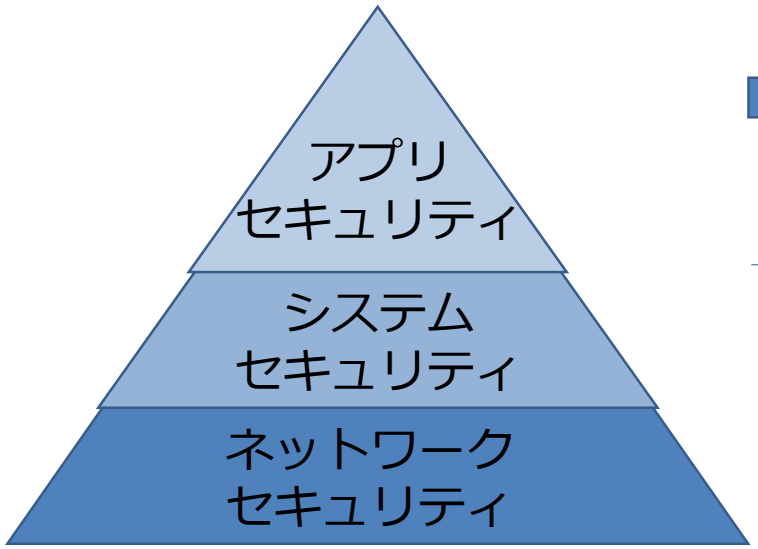
| No | ユースケース | 開発・運用切り分け | | オペレーションマニュアル作成対象 |
|---|---|---|---|---|
| | | 開発 | 運用 | |
| 1 | 公開APIの開発環境を作成する | | ○ | ○ |
| 2 | 公開APIの本番環境を作成する | | ○ | ○ |
| 3 | 内部APIをリリースする（バックエンドAPIのプロキシー） | ○ | | |
| 4 | 公開APIを新規作成する（パラメータ変換処理等を含む） | ○ | | |
| 5 | 公開APIを更新する（パラメータ変換処理等を含む） | ○ | | |
| 6 | 公開APIをリリースする | | ○ | ○ |
| 7 | 公開APIを利用会社に通知する | | ○ | ○ |
| 8 | 公開APIの利用状況を確認する | | ○ | ○ |
| 9 | 公開エンドポイントを死活監視する | | ○ | |
| 10 | 公開APIのパフォーマンスを監視する | | ○ | |
| 11 | ユーザ(APIキーも含む)・ロールを追加・変更する | | ○ | ○ |
| 13 | 障害対応を行う（ログ取得、サポート問合せ） | | ○ | ○ |
| 13 | 設定情報をバックアップする | | ○ | ○ |

オージス総研

# APIのセキュリティ

# APIセキュリティの考え方

- APIのセキュリティは、Webアプリケーションのセキュリティをベースに考えAPI固有の要素に対応する。




Webアプリケーションと共通

- 暗号化
  - HTTPS
- URLパス名チェック
  - ディレクトリ、トラバーサル

など

API固有

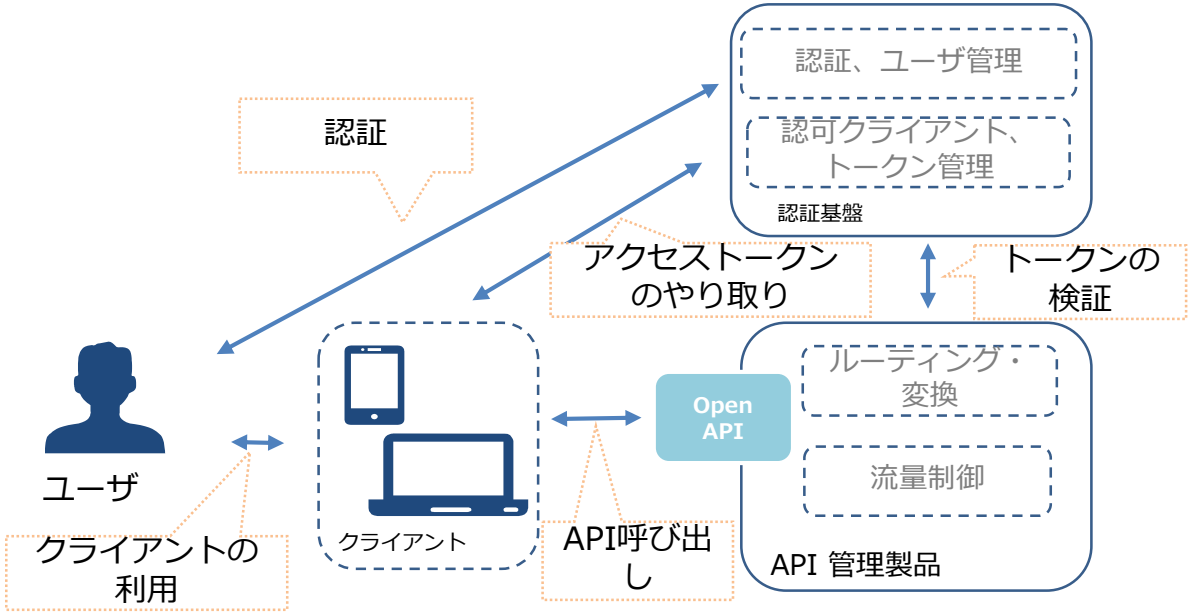
- データバリデーション
  - インジェクション対策、JSON、XMLに関する攻撃対策
- アクセス制御
  - OAuth2.0, OIDC、FAPI

など

ネットワークやOSレベルのセキュリティの考え方は、従来のWebアプリケーションとほぼ同様

# オープンAPIで考慮すべきセキュリティ標準

- FISC 安全対策基準、PCI DSS 、ISO27001 等の金融機関によく活用されているセキュリティ関連基準の考慮
- OAuth2.0, OpenId Connect, Financial-grade APIなどAPI特有のセキュリティ標準への対応
- 本人確認(KYC)、様々な認証への対応、決済指示API時の追加認証対応



# APIのアクセス制御

オージス総研

## ◘ APIによって異なるセキュリティレベル

### 参照系API

- 株価・為替相場情報参照
- 店舗・ATM所在地
- 金利・手数料照会
- 店頭混雑状況照会
- 匿名加工・分析情報
- ポイント照会
- カード請求額照会
- 口座情報参照
- 口座残高照会
- 入室金明細照会
- KYC・AML関連情報
- 営業機密データ、等

出典:金融庁 金融制度WG第3回資料

低 機密性・秘匿性 高

### 更新系API

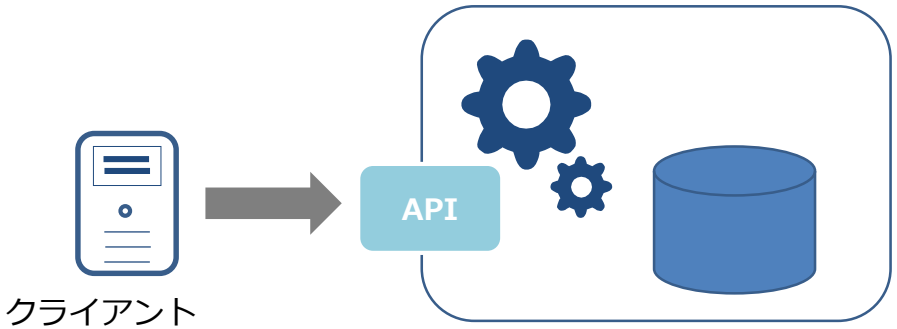
- 来店予約
- ローンシュミレーション
- 口座開設
- 各種届出
- 株式売買指図
- 投信購入指図
- 保険商品購入指図
- 資金移動
  - 振込・振替指図
  - 口座振替、等

# APIのアクセス制御

- 提供しているAPIは、クライアントに対してアクセス制御したいのか、リソースオーナー(エンドユーザー)に対してアクセス制御したいのか。

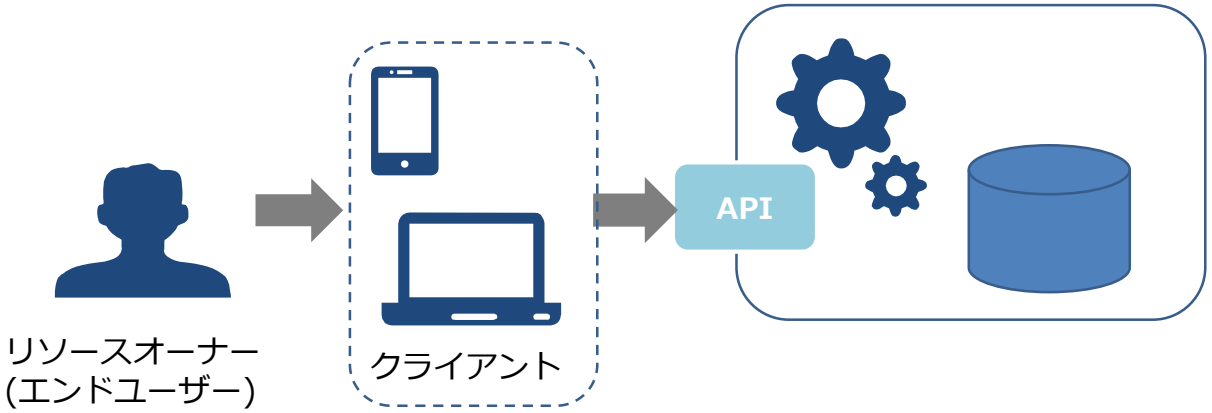
## クライアントに対するアクセス制御

ユーザに依存しない特定クライアント向けのデータや機能
例えば、取引先のシステムに提供する、カタログ情報参照API




## エンドユーザーに対するアクセス制御

ユーザーが承認したデータや機能
例えば、サードパーティアプリに提供する、口座情報参照API

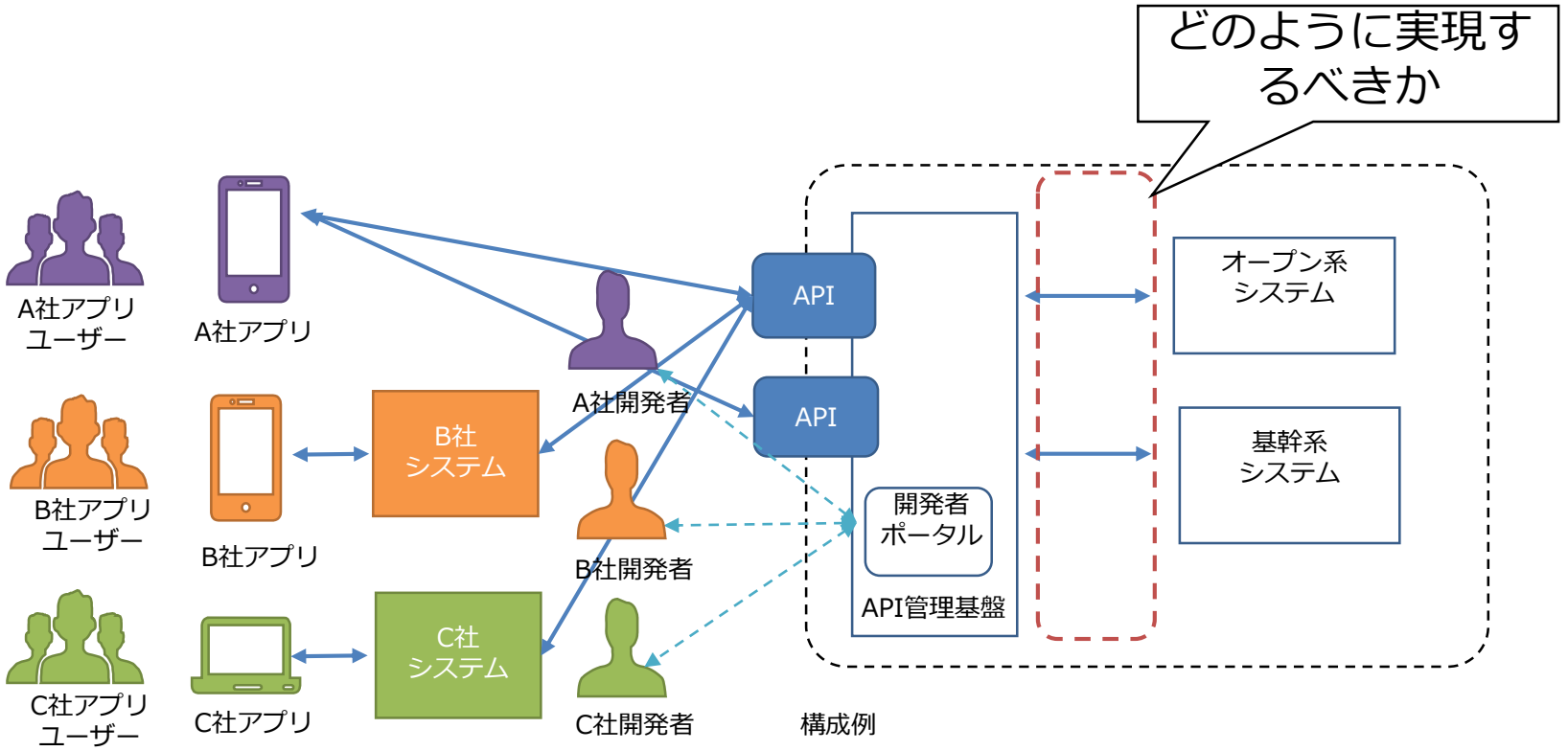


オージス総研

# 既存システムとの連携

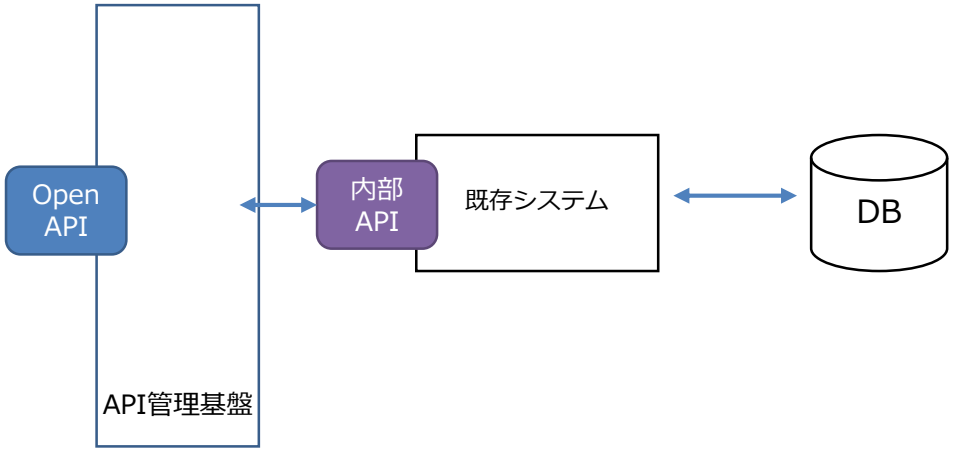
# 既存システムとの連携

- APIが提供するデータや機能の元となるシステムや連携する手段が異なる
  - システムの種類
  - 更新処理の有無
  - リアルタイム性



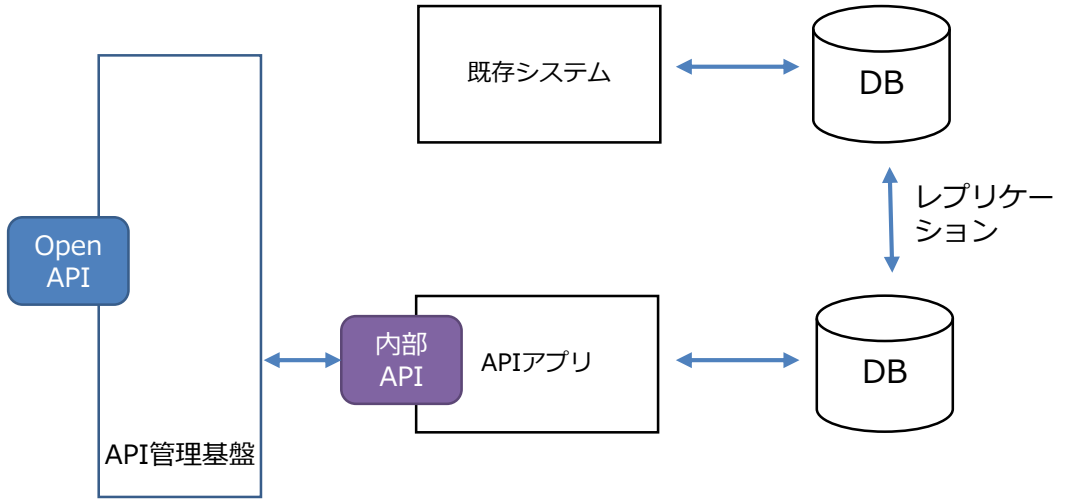
# 連携パターン：既存システムの拡張

- 既存システムを拡張し、内部APIを作成する
- 既存の既存システムと
  - 開発、管理組織が一緒
  - SLAが一緒
  - 変更サイクルが一緒
- 規模が大きくないWebアプリで使われることが多い



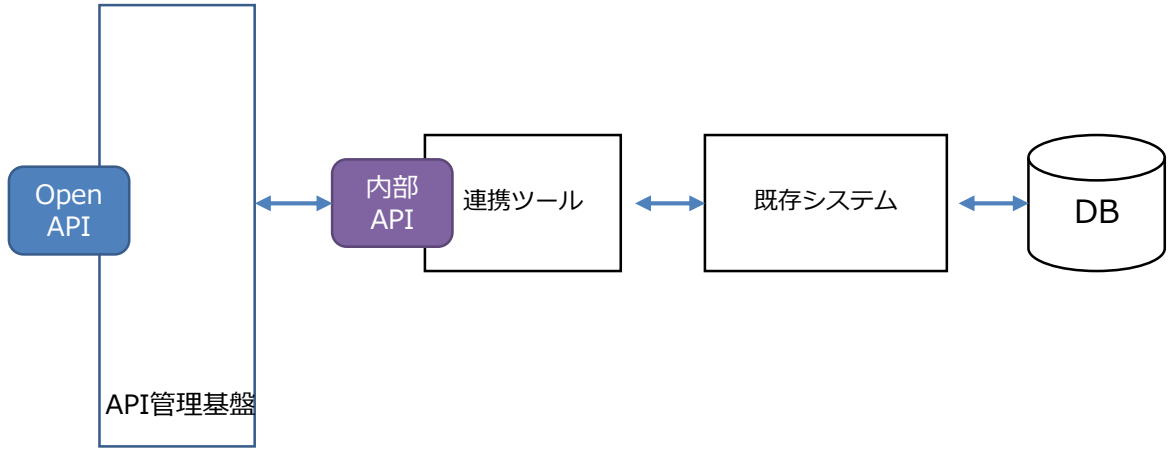
# 連携パターン：データコピー

- 既存システムのデータをコピーして、APIを新規開発
- 既存のシステムと
  - 開発組織が異なる
  - SLAが異なる
  - 変更サイクルが異なる
- 既存システムへの影響を最小化したい
- DB同期のタイムラグが許容できる場合
- DB間の双方向の同期が必要ない場合
  - 参照系APIのみが利用する



# 連携パターン：連携ツールの利用

- 連携用のツールを利用し、内部APIを作成する
- 管理組織が異なるため
  - 既存のAPIを修正できない
  - データ・ベースを共有できない
- 既存システムへの影響を少なく、更新系処理があるAPIの場合



オージス総研

# 利用者を意識したAPI

# DXを意識したAPI

- DX: Developer Experience （開発体験）を意識したAPIとは、あるAPIを開発者が「気持ちよく開発・保守できるかどうか」を示す

- DXが優れたAPIとは

  - 理解しやすくシンプル
  - トラブルシュートが容易である
  - 変更管理が明確である
  - 色々試すことができる

- DXが優れたAPIはサポートのコストを低減する

  - 開発者がAPIの使い方に迷うことがないため問い合わせが減る
  - トラブルシュートのとき何をすべきかわかるため、適切な対処ができAPI提供側のシステム負荷を減らすことができる

# 標準的なAPI仕様

- APIを利用する開発者に向けて分かりやすい標準的な仕様にすることが重要になる。独自仕様は開発者の混乱を招き、API利用が進まない原因になる
- REST/JSONスタイルが採用されることが多い
  - RESTスタイルのメリットは一貫性と予測可能性(URIとHTTPメソッドからある程度何ができるAPIなのか類推できる)

REST/JSON スタイル

**REST: 基本的な考えはHTTPの原理。URIはリソースを表す名詞**

GET http://domain/api/items 商品一覧取得
POST http://domain/api/items 商品登録

**JSON: シンプルで相互運用性が高い**

```
{"user" : [
  { "name" : "saito", "age" : "32" },
  { "name" : "yamada"  , "age" : "25" },
  { "name" : "kimura" , "age" : "41" }
]}
```

技術的な標準仕様

- OpenAPI Specification

業界向け標準仕様

- [金融] Fintech共通API、Open Banking Standard
- [スマートハウス] HEMSデータ利活用業者間API標準

# API仕様の定義

オージス総研

## Open API Specification

- デファクトスタンダード
- バージョン2はSwagger
- バージョン3が最新
- 様々なAPI管理製品/サービスがサポートしている
- この定義を元にAPI仕様のドキュメントを作成することができる

```
---
swagger: "2.0"
info:
  version: "1.0.0"
  title: "Swagger Petstore"
  description: “A sample API that uses a petstore as an example to …”
  termsOfService: "http://swagger.io/terms/"
  contact:
    name: "Swagger API Team"
  license:
    name: "MIT"
host: "petstore.swagger.io"
basePath: "/api"
schemes:
  - "http"
consumes:
  - "application/json"
produces:
  - "application/json"
paths:
  /pets:
    get:
      description: "Returns all pets from the system that the user has …"
      produces:
        - "application/json"
      responses:
        "200":
          description: "A list of pets."
          schema:
            type: "array"
            items:
              $ref: "#/definitions/Pet"
definitions:
  Pet:
    type: "object"
    required:
      - "id"
      - "name"
    properties:
```

https://github.com/OAI/OpenAPI-Specification/blob/master/examples/v2.0/yaml/petstore-minimal.yaml

# APIのエラー通知について

- APIを利用する開発者からくる問い合わせはAPIのエラーに関するものが多い
- APIドキュメントを充実させることも大事だが、APIのエラーレスポンスを工夫することが重要

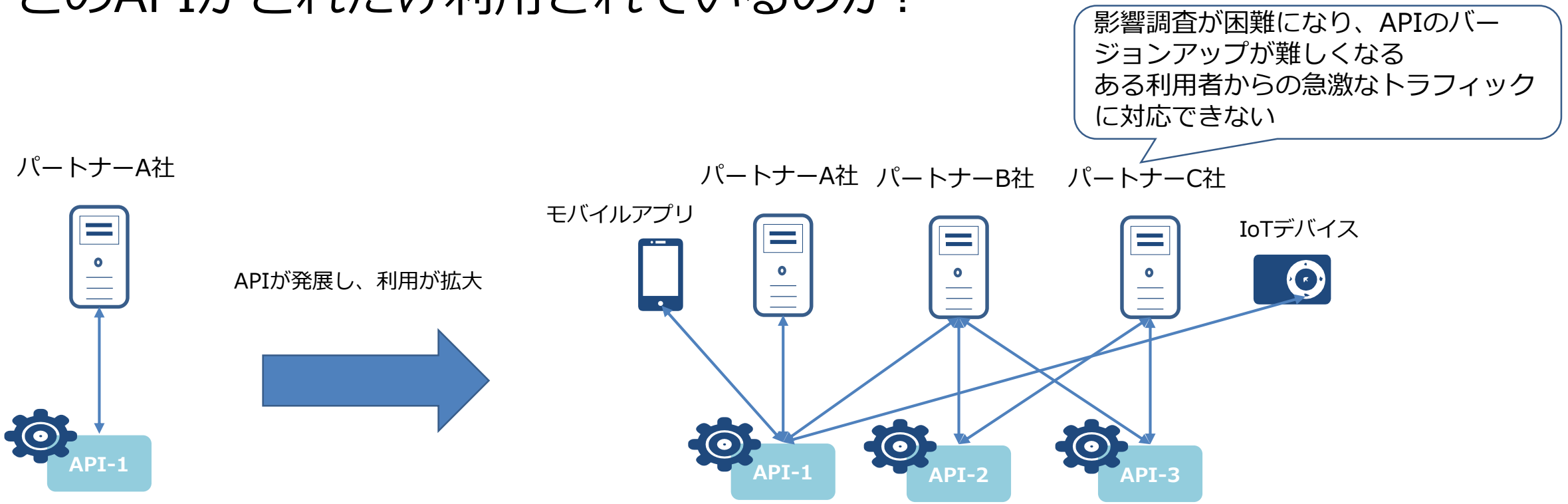
  - HTTPのレスポンスコードを適切なものを利用する
  - 何が起こっているのかAPI利用者が理解できる
  - 次にAPI利用者が何をするべきかわかる
  - 内部構造は隠蔽する
  - 一貫性があり、シンプルである

オージス総研

# API管理の重要性

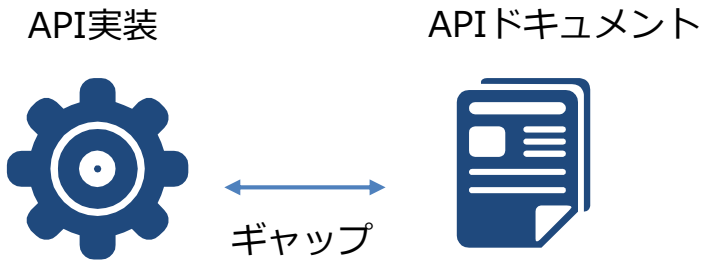
# APIの数が増え、利用が増加した場合の管理

- APIは誰(どのAPIクライアント)から利用されているのか？
- どのAPIがどれだけ利用されているのか?



# API利用者向けドキュメントの提供

- APIを利用する開発者向けにAPIの仕様をドキュメントとして提供しなければならない。
- WordやHTMLなどの静的なドキュメントでは、変化の多いAPI開発に追従させなければならないが、APIの実装と仕様の乖離が発生してしまうことがある。



# API公開のライフサイクルを支援するAPI管理製品

オージス総研

- API公開のライフサイクルの中の設計、開発、運用をより効果的に実践できる。

運用を効率的に実行できる

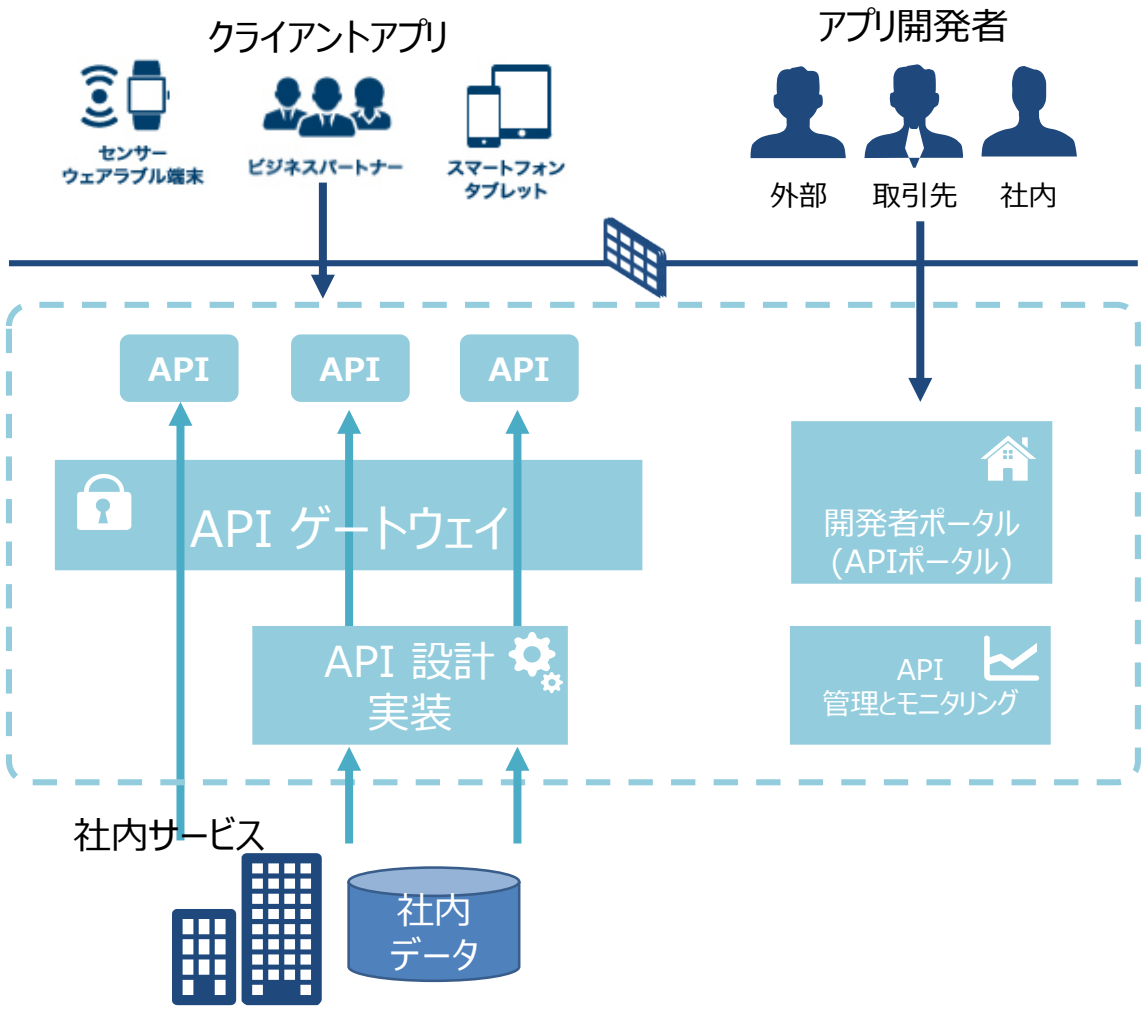

- API公開の目的を明確に
- APIの利用者および公開範囲を明確に
- APIとして公開するデータ、サービスを明確に
- APIの利用シナリオを明確に

高速にAPIの開発行うことができる

設計のベストプラクティスを適用できる

# API管理製品とは

- API管理製品：個々のAPI実装から、共通に管理すべき機能・非機能要件を分離
- 公開しているAPIの全貌を一括して管理可能
  - API管理製品の標準的な構成
    - → APIゲートウェイ
    - → API管理機能（認証・認可や流量制御設定等）とモニタリング（ログ、アクセス統計情報等）
    - → API設計・実装
    - → 開発者ポータル
- 代表的なソフトウェア/サービス
  - パッケージ製品
    - →IBM API Connect、Apigee Edge、 Mulesoft Anypoint Platform
  - クラウドサービス
    - →Amazon API Gateway、 Azure API Management
  - OSS(オープンソースソフトウェア)
    - →Kong、Zuul



# API管理製品採用のポイント

オージス総研

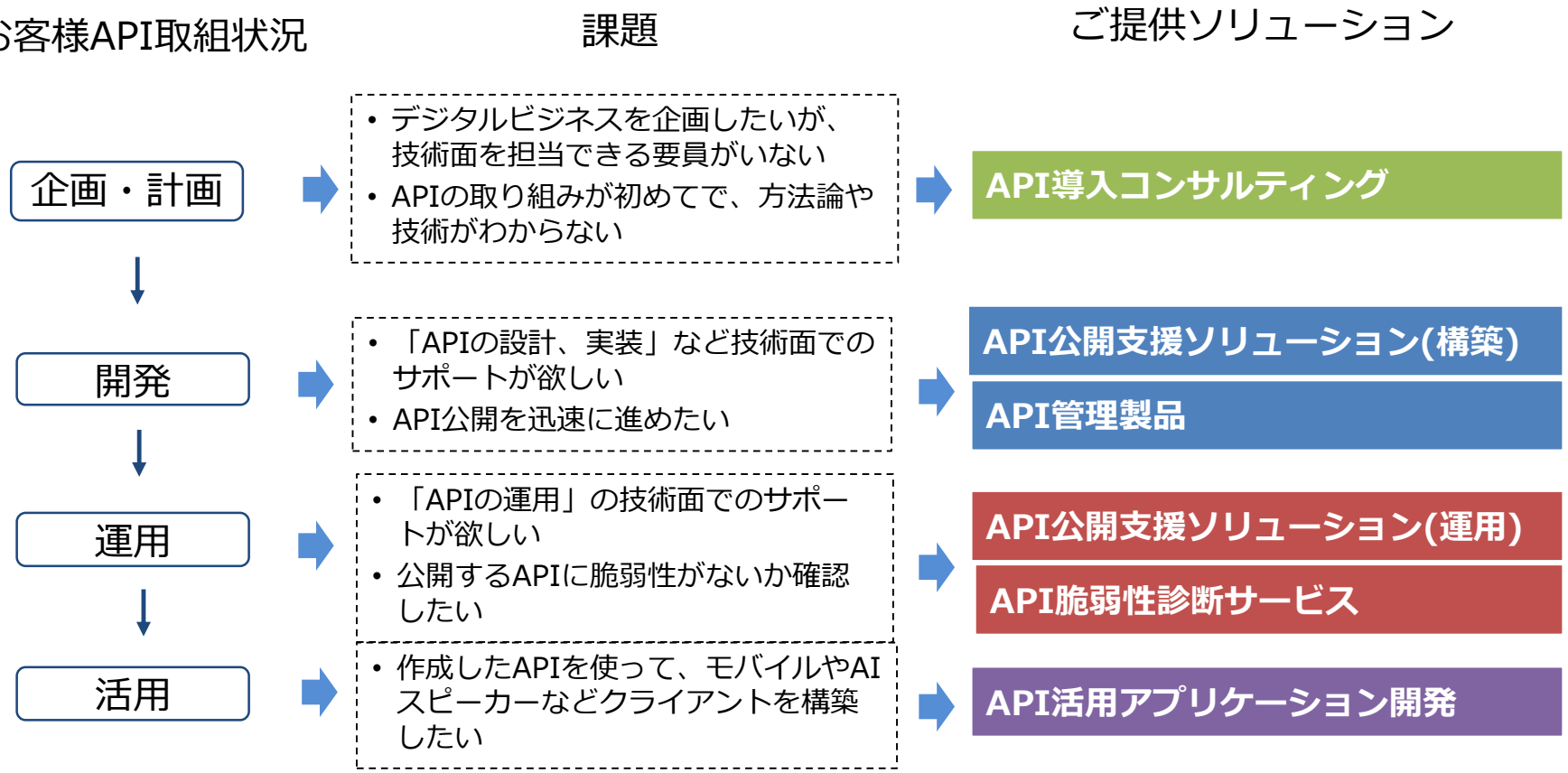
- API管理製品の採用可否の判断(目安)
  - APIのクライアントが３以上
  - APIの数が30以上
  - APIのグループ・カテゴリが3以上
  - APIのセキュリティモデルが2以上(特にOAuth2が含まれる場合)
- 製品選定のポイント
  - ゲートウェイで必要最小限のアクセス管理をするか？
  - API管理製品（スイート）でライフサイクル全体を管理するか？
  - クラウドサービスかオンプレか？
  - 社内既存システムとの連携の重要度は？
  - API実装フレームワークを製品で共通化するか？

オージス総研

# オージス総研のAPI公開支援ソリューション

# オージス総研APIソリューションの全体像

- APIの技術調査・検討から運用までのフルカバー
- API利用者のニーズを捉えたビジネスの継続的な進化を実現する



# ご支援内容

- ロードマップの策定支援
  - ✓ APIの成長を考慮した長期的なロードマップを策定します
- API管理要件の定義
  - ✓ クライアントや開発者の管理、APIドキュメントの公開、APIのバージョンや互換性に関する方針などを定義します
  - ✓ 可用性、性能・拡張性、運用・保守性、セキュリティなどのAPIに対する非機能要件を定義します
- 初期システムアーキテクチャ策定
  - ✓ ネットワークやサーバー構成、認証、認可の仕組みなど、要件を実現するために必要な初期システムアーキテクチャ案を決定します
  - ✓ API管理製品の導入の要否やAPI管理製品の選定を行います
- API導入PoC
  - ✓ APIの導入の概念検証を支援します
  - ✓ 実践を通じて様々なAPI導入に対する様々な知見を得ることができます

# オージス総研のAPI公開支援ソリューションの価値

- 様々なAPI案件を積み重ね得られたAPI公開プロセスのノウハウとプロセスを効率的に実践できるAPI管理製品により、お客様のAPI公開を実現いたします。


API公開プロセス