ECHONET Lite Web API

セキュリティ関連　検討状況

2024年3月8日
エコーネットコンソーシアム
寺本　圭一

今回ご紹介させていただく内容

本日の主題:

- 宅内のECHONET Lite対応機器などをクラウド上からWeb APIによって操作可能とする標準仕様「ECHONET Lite Web API (ELWA)」の検討WGにて検討中のセキュリティ課題を紹介

エコーネットコンソーシアムでの目標:

- 同Web APIを用いた特定サービス仕様にて、ELWA規格適合性認証試験・認証制度を提供(相互接続性)
- ELWA搭載クラウド(サーバ)および外部アプリ(クライアント)の両方に対して認証取得を可能とする
- セキュリティ自体、試験環境への接続方法として指定するが、ELWA規格適合性認証の対象とはしない

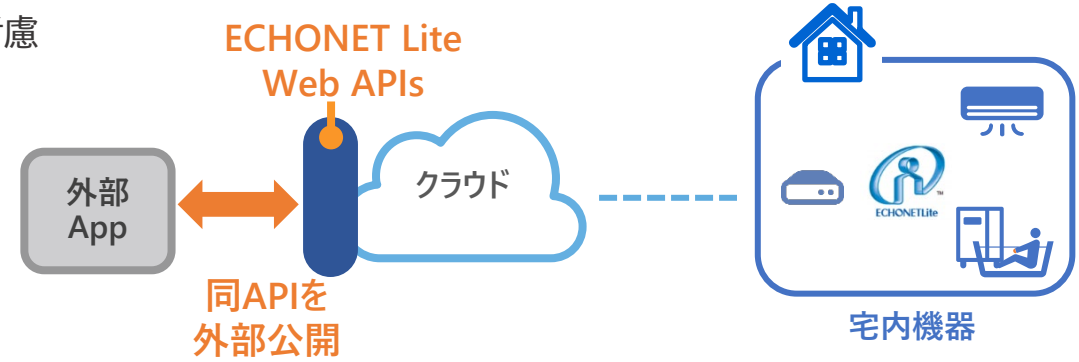
セキュリティ関連仕様の考え方:

- RFCなど既存の標準仕様から当方の想定要件に適合する仕様を選択(OAuth2.0/OIDCベース)
- 既存の商用認可サーバなど実装状況も考慮

現在の検討状況:

- 要件整理中、標準仕様＆既存実装調査中

お悩み相談的な話題も多いかと存じますが
アドバイス＆コメントいただければ幸いです



発表の流れ

1. ECHONET Lite Web APIの概要・背景
2. ECHONET Lite Web APIガイドライン概要
3. WoTとECHONET Lite Web APIの連携
4. Web APIセキュリティ検討（認証・認可）
5. まとめ

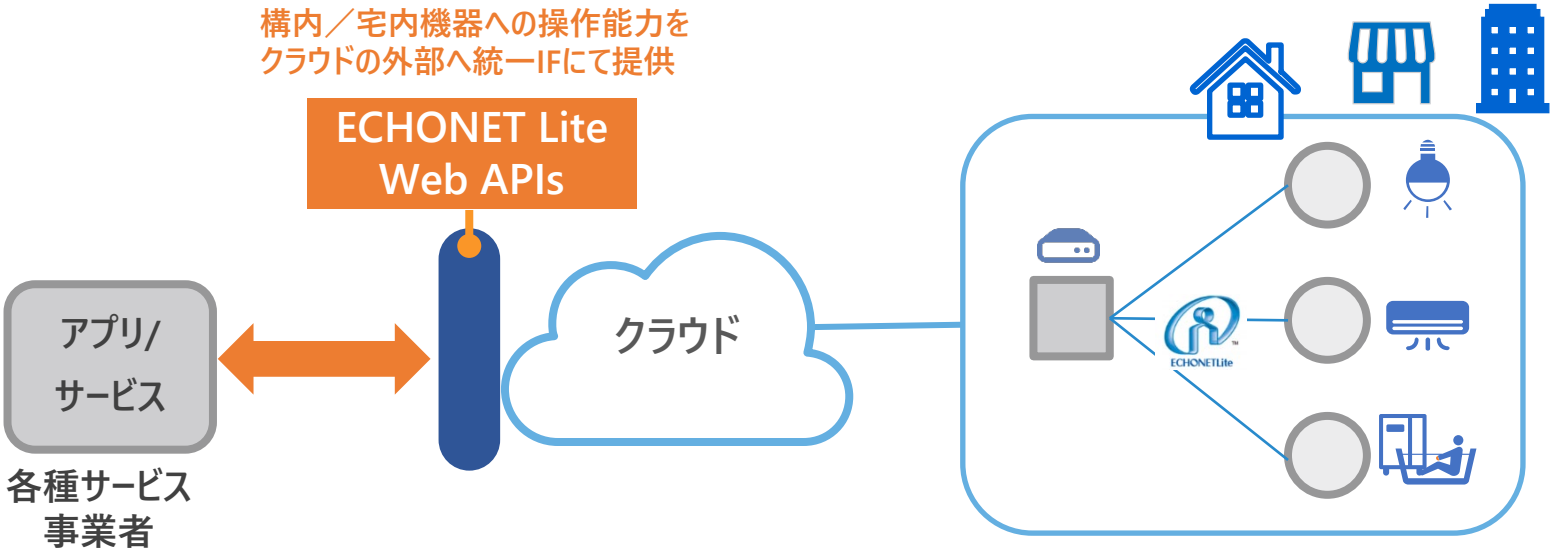
01

ECHONET Lite Web APIの概要･背景

ECHONET Lite Web APIとは？

ECHONET Lite等の機器をクラウドを介して操作可能とするWeb API

構内／宅内機器への操作能力を
クラウドの外部へ統一IFにて提供

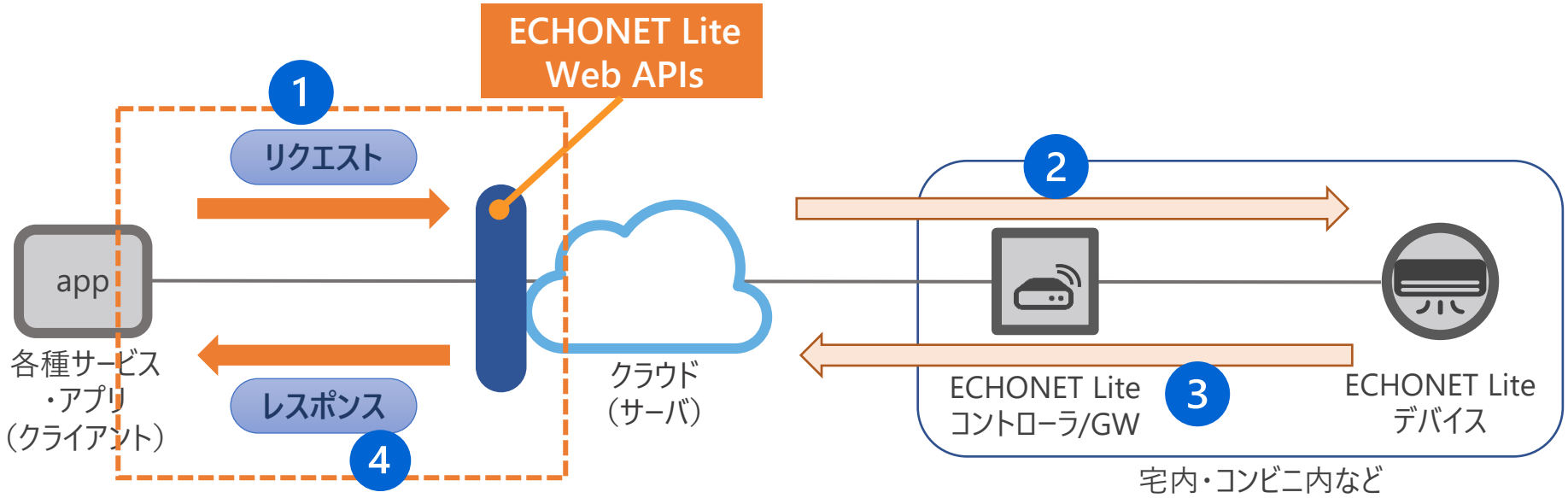


機器：照明、エアコン、給湯器等
（ECHONET Lite以外の機器もマッピング可能であれば対象）

外部連携サービス　クラウド独自サービス　宅内網での制御

ECHONET Lite Web APIの対象範囲

クライアントへサーバが提供する機器操作用Web API(リクエスト、レスポンス手順等)

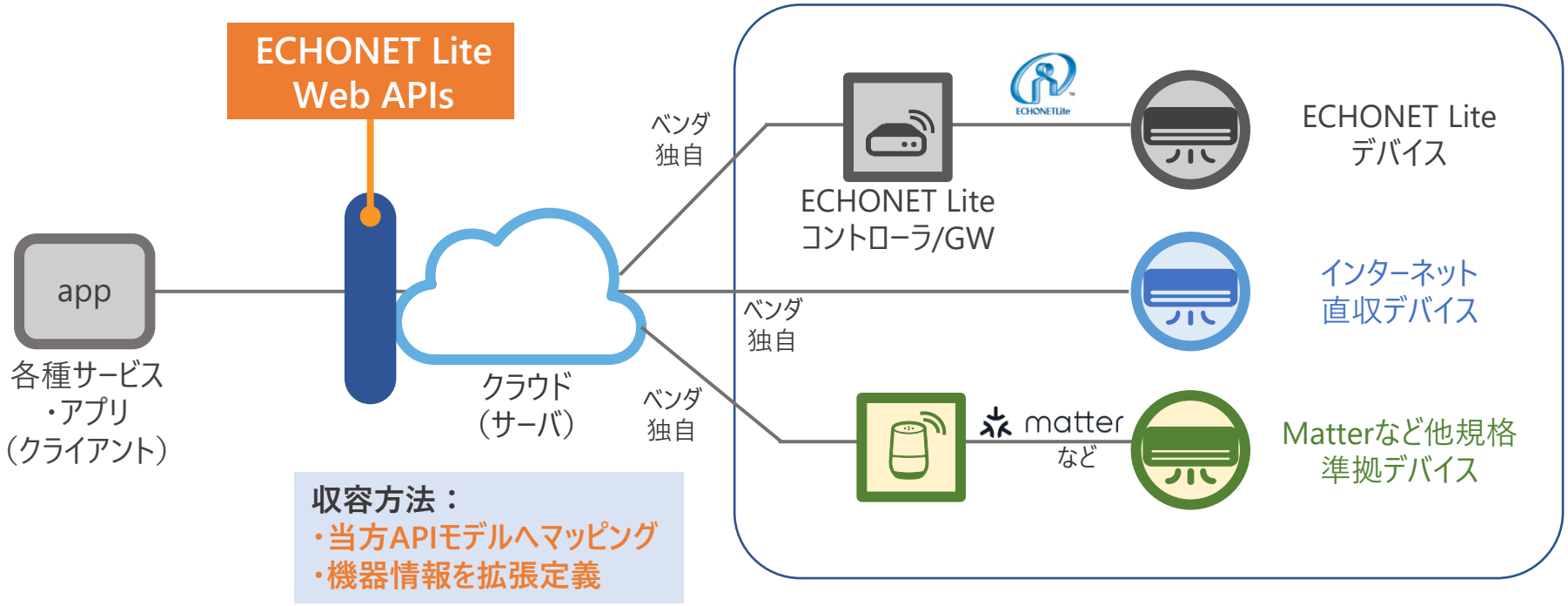


ECHONET Lite Web APIの対象範囲

処理によっては②③が不要なケースもあります

ECHONET Lite Web API：宅内機器のカバー想定範囲

ECHONET Lite機器や他のIoT機器もクラウド収容可能なモデル



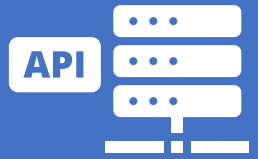
収容方法：
- 当方APIモデルへマッピング
- 機器情報を拡張定義

標準Web API採用によるメリット

サービス事業者

- 複数のWeb API提供者と連携しユーザ拡大
- 機器管理/操作を外部へオフロード可能
- 付加価値サービス注力

標準 Web API提供者



- 顧客へ多彩なサービスメニューを提供可能に（新規事業導出・共創）
- 大量の機器リソース獲得

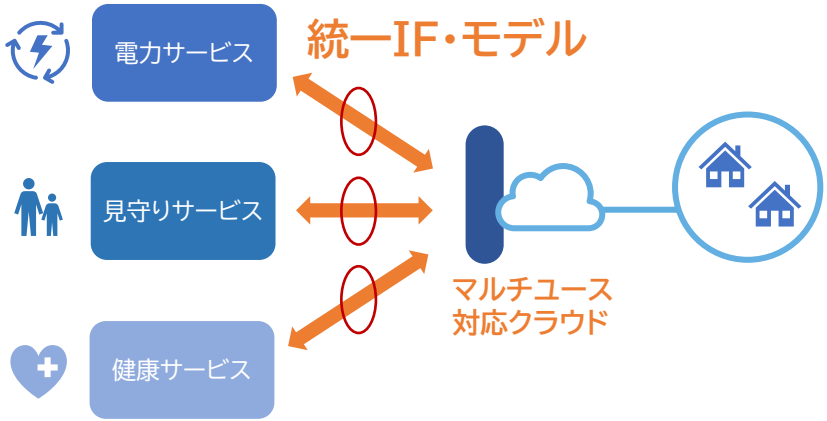
GW/機器ベンダ

- Web API提供者やサービス事業者との連携にてマネタイズ機会増大

統一APIによるメリット

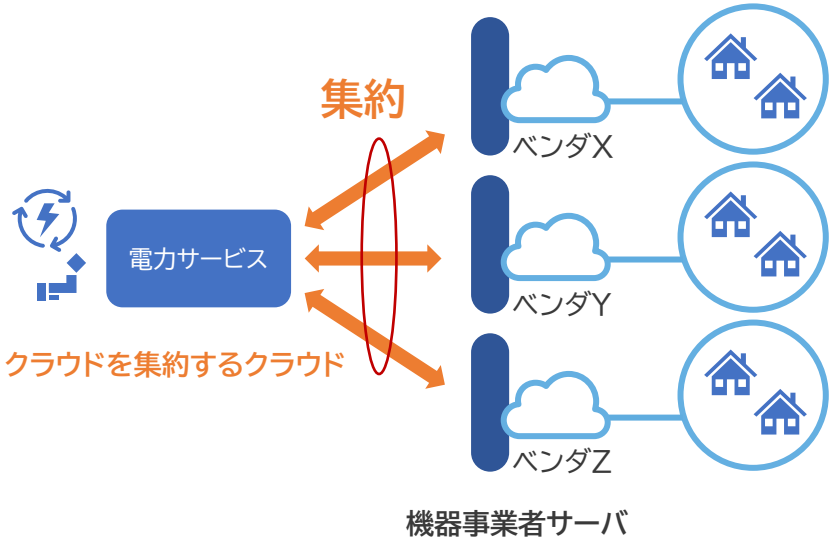
メリット1

各種サービスに対応可能な
統一APIモデルを活用可能



メリット2

複数機器事業者を束ねる
集約クラウドを実現可能



ECHONET Lite Web APIによる異業種サービス連携PF

異業種サービスとの連携を促進するIoTクラウドの標準基盤へ

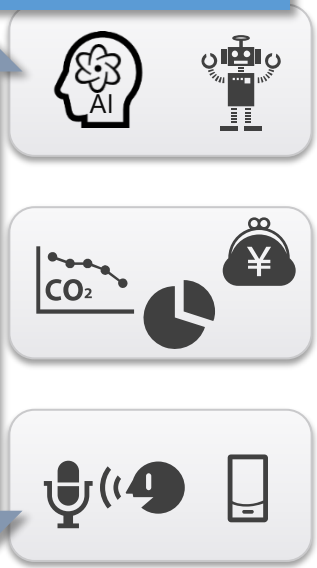
多様な業種をつなげたい

AI
ロボット操作

環境
エネルギー
健康

マルチモーダル
位置情報

[事例]



異業種サービス

クラウドやスマホアプリなど

統一インタフェース提供

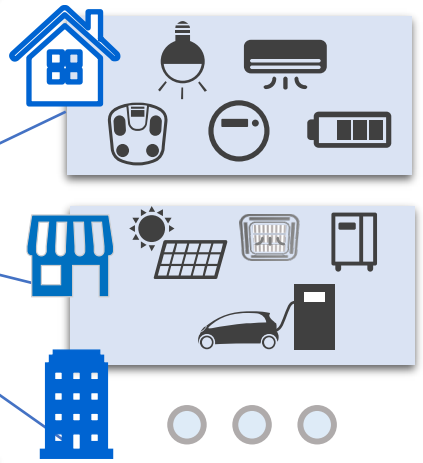
ECHONET Lite Web APIs

・異業種機器の混在・集約を実現
・異業種サービス向け統合API提供

異業種サービス連携PF

多様な機器をつなげたい

ECHONET Lite機器／非準拠機器



宅内、構内（中小ビル、コンビニ等）

IoT機器群

ECHONET Lite Web API ガイドライン提供リソース

随時更新・拡張中

ベース仕様

- API仕様部
- 機器仕様部

サービス仕様

- DR関連サービス仕様
- PCHAデータ連携に関するガイドライン仕様例 *1

参考資料

- Device Description集
- OpenAPI Document

参考書類 *2

- MRA(Machine Readable Appendix)
  機器オブジェクト詳細既定のJSON形式版

会員限定 *3

- 実験クラウド
- 動作確認GUIツール
- 学習用アプリ

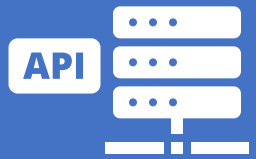
HP (https://echonet.jp)上にて公開中
無印: ダウンロード>Web API>ECHONET Lite Web APIガイドライン
*1: 活動内容>関連する団体との活動>PCHA
*2: ダウンロード>規格書・仕様書など>Machine Readable Appendix (参考)
*3: 会員ページ: 会員トップ>Web API実験クラウド

02

ECHONET Lite Web APIガイドライン

ECHONET Lite Web API ガイドライン：ベース仕様

API仕様部



API仕様や事例紹介

- ユースケース
- Web APIモデルの指針
- ECHONET Lite 仕様のマッピング指針
- 応用サービス

機器仕様部

対応機器毎のJSON定義集

- データ型定義
- ネーミング指針
- 搭載プロパティ方針
- 共通項目（SuperClass）
- 機器毎のDevice Description

いずれもエコーネットコンソーシアムHPにて公開中（https://echonet.jp/web_api）

ECHONET Lite Web APIガイドライン：ラインナップ

ベース仕様

API仕様部

メイン仕様。要望に基づき拡張

機器仕様部

ECHONET Lite Appendixの
更新に伴い改訂（年1回目処）

＋

サービス仕様

DR関連サービス仕様 — JEMA連携

PCHAデータ連携に関するガイドライン ECHONET Lite Web API仕様例 — PCHA連携

JEITA連携

＜外部団体らとの協議に伴い
連携サービス追加・拡大中＞

各種業界団体
とコラボ

基本機能：devices サービス リソース操作 俯瞰

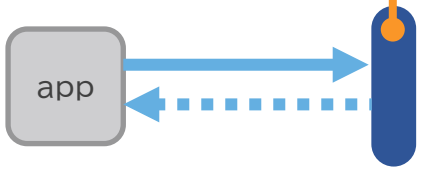
リクエスト（/elapi/v1省略）

ECHONET Lite Web APIs

レスポンス例

機器一覧取得

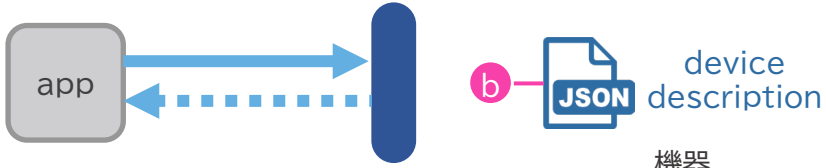
GET /devices




```
{ "devices": ["a", "b", "c"] }
```

機器情報取得

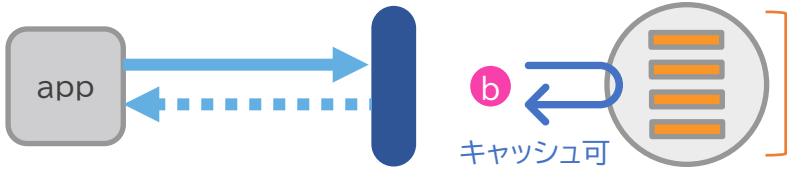
GET /devices/<id>




```
{
  "properties": ... ,
  "actions": ...
}
```

全プロパティリソース取得

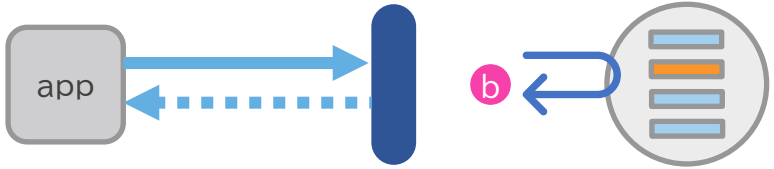
GET /devices/<id>/properties




```
{
  "propA": "v1",
  "propB": "v2",
   ...
}
```

指定プロパティリソース取得

GET /devices/<id>/properties/propA

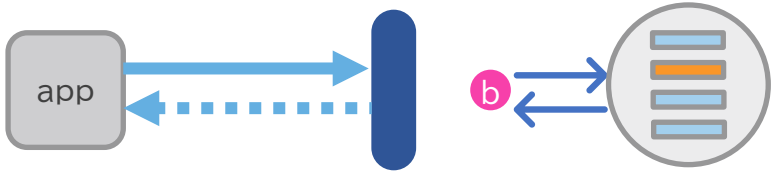



```
{  "propA": 1 }
```

指定プロパティリソース設定

PUT /devices/<id>/properties/propA

リクエストボディ例 { "propA": 10 }

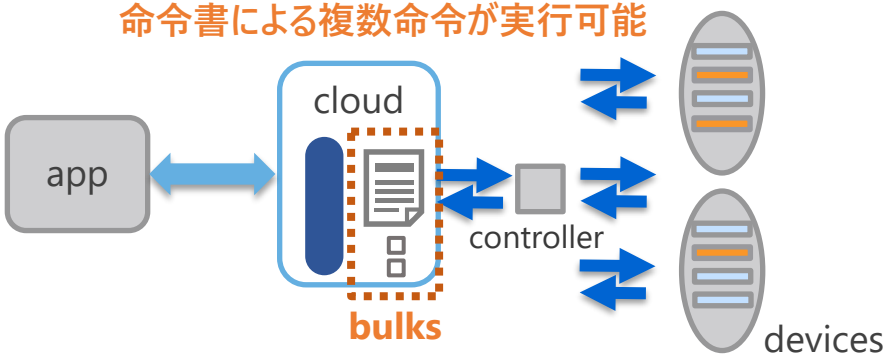



```
{  "propA": 10  }
```

API仕様部：応用ユースケース

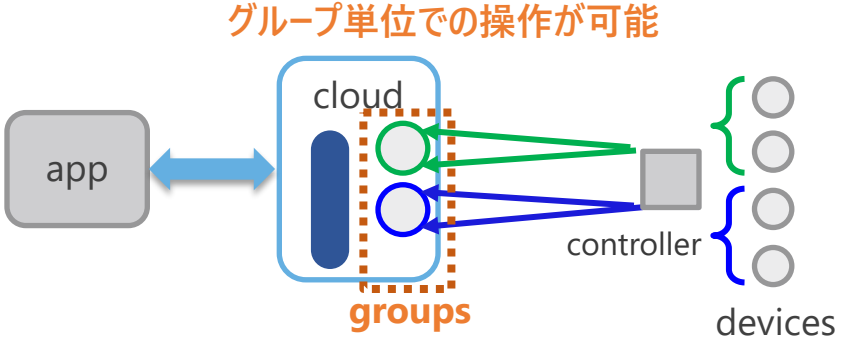
1 複数命令の一括指示

任意の機器、任意のプロパティを対象とした**コマンドを複数列挙した命令セット**を作成することができます。この命令セットを用いて、一括で操作・指示することが可能となります。非同期・同期実行の指定も可能です。



2 機器のグルーピング

複数の**機器をまとめてグループ化**します。
グループ化した対象は、リソースの分類・整理や、同種の命令を受け付けるなど、グループ化により規定される動作も可能となります。仮想機器化にも利用できます。

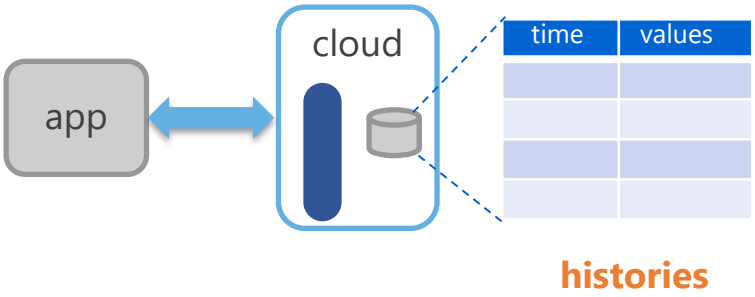


API仕様部：応用ユースケース

3 履歴データの蓄積・検索

指定した**機器に関する取得値を時刻とともに**記録し、取得可能にします。取得したい値や時刻の範囲などを指定し、アプリ側でグラフ表示や各種データ加工などに利用できます。

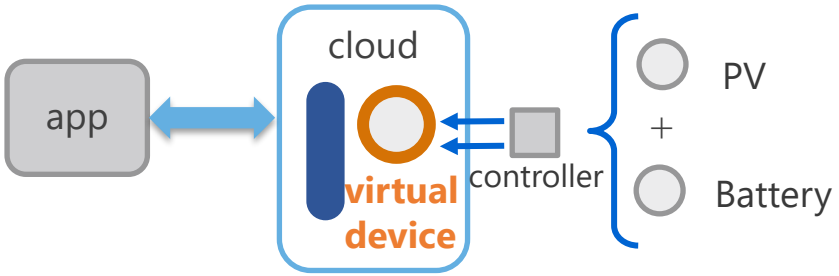
機器の稼働記録・計測値など検索が可能



4 仮想的な機器を定義可能

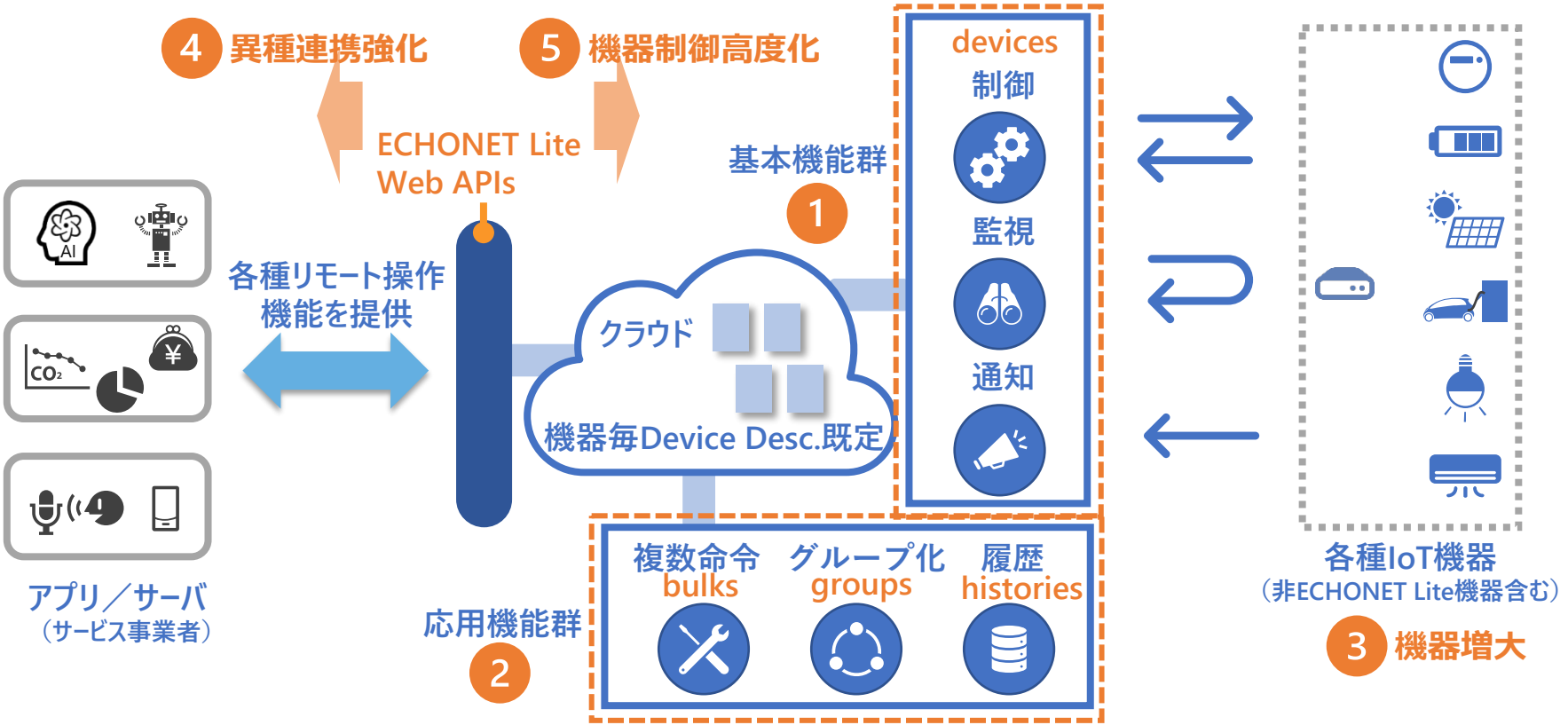
2種類以上の機器を組み合わせたり、クラウド上で任意の機能を追加することで、あらたな機能を持つ**独自機器を定義・操作**できます。仕様外の任意の機器やプロパティを定義できます。

複数機器＋機能を結合した仮想VPP機器など生成できる



ECHONET Lite Web API：機能構成(現状)

ECHONET Lite Web API ＝ 基本機能群 ＋ 応用機能群



機器仕様部：V.1.6.0 改訂ポイント（策定中）

全50機種。現在、ECHONET Lite Appendix Rel.Rに対応中

V.1.5.0 時点のサポート機器　★ AIF対応機器

家庭用エアコン ★	住宅用太陽光発電 ★	拡張照明システム ★	温度センサ	水流量メータ
換気扇	床暖房	冷凍冷蔵庫	電力量センサ	人体検知センサ
空気清浄機	燃料電池 ★	クッキングヒータ	電流センサ	風呂沸き上がりセンサ
業務用パッケージエアコン室内機 ★	蓄電池 ★	炊飯器	冷温水熱源機	ガスメータ
業務用パッケージエアコン室外機 ★	電気自動車充放電器 ★	業務用ショーケース ★	電力量メータ	スマート電力量サブメータ
電動雨戸・シャッター	低圧スマート電力メータ ★	業務用ショーケース室外機 ★	分電盤メータリング	照明システム
電気温水器 ★	高圧スマート電力メータ ★	スイッチ（JEM-A/HA端子対応）	空調換気扇	非常ボタン
電気錠	一般照明 ★	コントローラ	テレビ	照度センサ
瞬間式給湯器 ★	単機能照明 ★	ハイブリッド給湯機	CO2センサ	VOCセンサ
浴室暖房乾燥機	電気自動車充電器 ★	洗濯乾燥機	温度センサ	分散型電源電力量メータ

＋

V.1.6.0 対応

新規追加

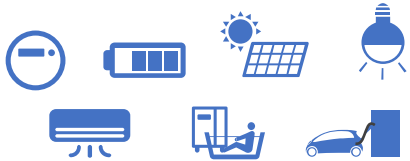
- 双方向対応高圧スマート電力量メータ
- 周波数制御クラス
- マルチ入力PCS
- 防犯センサ
- 電動ブラインド・日よけ

改訂

- 低圧スマート電力量メータ
- 機器オブジェクトスーパークラス
- 住宅用太陽光発電
- 蓄電池
- 電気自動車充放電器
- 分散型電源電力量メータクラス

ECHONET Lite Web APIのお薦めポイント

ECHONET Lite Web API の強み



既製品多数に対応

市場展開中のECHONET Lite製品(機器クラス)を概ねカバー。機器機能をクラウド経由で活用できます

リアルタイム制御・監視

ECHONET Lite機器へのコマンドをクラウド経由で呼び出すAPIを提供。非ECHONET Lite機器も対応可能です

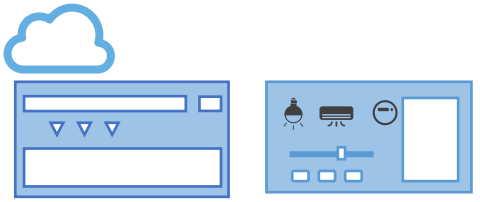
- Restful API
- 機器用スキーマ(JSON形式)
- OpenAPI

統一的なWeb APIモデル

Device Descriptionと呼ぶ機器用スキーマ定義により、機械可読可能な形式でAPI仕様を設計できます

応用API機能も充実

複数コマンド一括操作や機器グルーピング操作、履歴データの検索など高度なAPIを提供します



実験ｸﾗｳﾄﾞ・豊富ﾂｰﾙ群

会員向けに実験環境や、開発用・初心者用ツールなど提供。仕様理解が進み、開発の参考となります

ECHONET Lite Web APIのビジネス適用・展開

標準WebAPI活用による好循環サイクルにより普及促進・ビジネス連携機会増大へ

技術共有の視点

- 各種API仕様書は一般に公開
- ECHONET Lite仕様(MRA)、ECHONET Lite Web API仕様(Device Description、OpenAPI形式)もデータ公開中
- 非ECHONET Lite対応機器も収容可能なモデルを提供

業界連携の視点

- PCHA, W3C, JEMA, JEITAらと連携
- ヘルスケア、エネルギーなど複数業種展開中

顧客・ビジネスの視点

- IoT機器を製品として持たないサービス事業者によるビジネス参入が可能に
- サービス事業者、API搭載クラウド事業者との間でビジネス連携機会・共創が広がる

学習/成長の視点

- 実験クラウド、初心者用ツール、開発支援ツールなど展開(会員向け)
- ECHONET IoTマスター制度の対応教育機関にて本API教育提供(一般向け)

ECHONET Lite Web APIガイドライン(一般公開) https://echonet.jp/web_api_guideline/

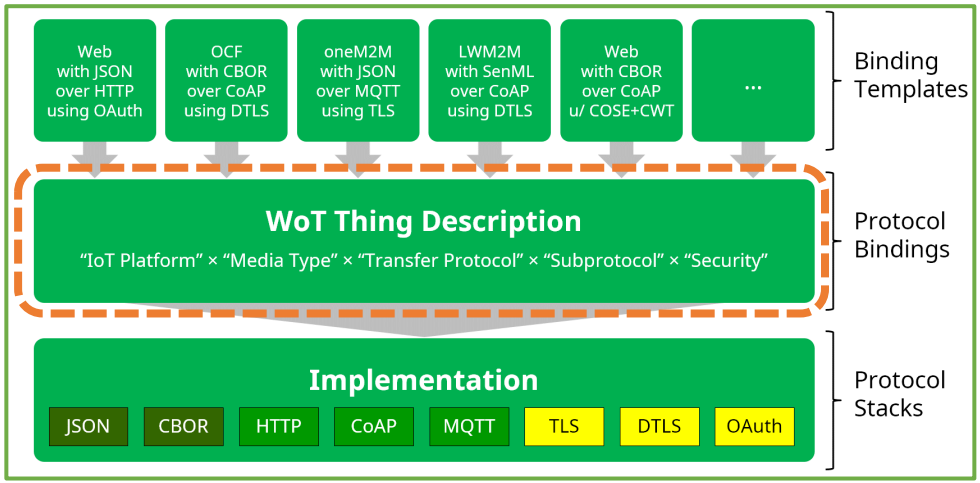
03

WoTとECHONET Lite Web APIの関係

ECHONET Lite Web APIの機器モデル

Webの標準化団体W3Cで策定中のWoTモデルを参考に定義（W3Cと提携中）

- WoTのThing Descriptionを簡素化した機器定義用Device Descriptionを設計



簡素化

機器用スキーマ定義

Device Description

基本的にHTTP(S)のみ採用

HTTP(S)

WoTモデル

ECHONET Lite Web APIモデル

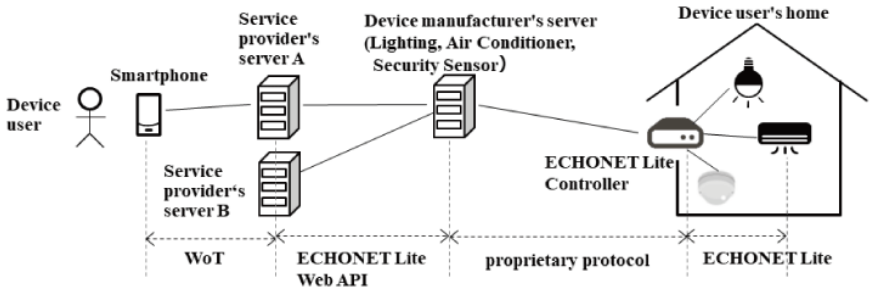
WoT Binding Templates (https://www.w3.org/TR/wot-architecture/)より
W3C：World Wide Web Consortium、WoT：Web of Things

W3C WoTとの連携活動

ユースケース登録、プラグフェスト参加などにて連携

家電連携ユースケース

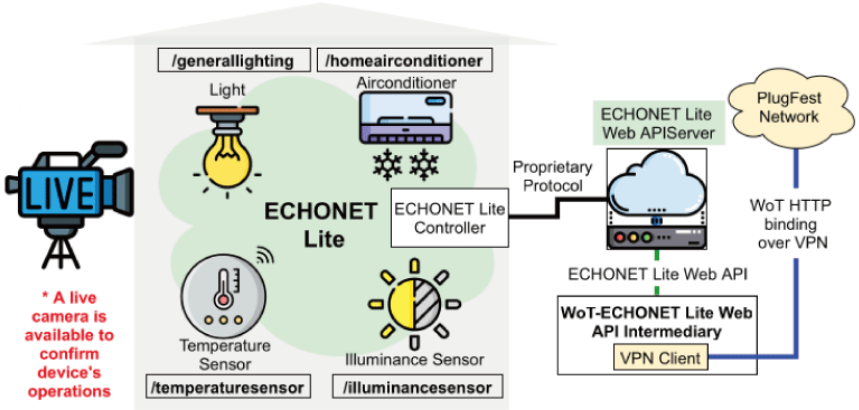
外出直後や帰宅直前に家庭内の複数機器を一斉に制御するスマートホーム関連操作モデル



Web of Things (WoT): Use Cases and Requirements

WoT Plugfest

「WoT Sept 2021 Plugfest/Testfest」において、本APIとWoT との連携動作を確認



WoT Sept 2021 Plugfest/Testfest

04

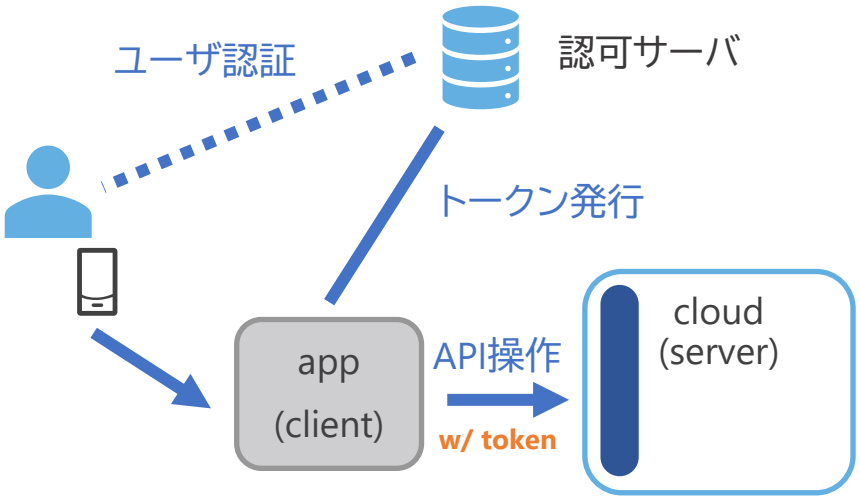
Web APIセキュリティ検討（認証・認可）

現行のELWA API仕様部における認証・認可の扱い

OpenID ConnectやOAuth2.0を使用した、典型事例を紹介

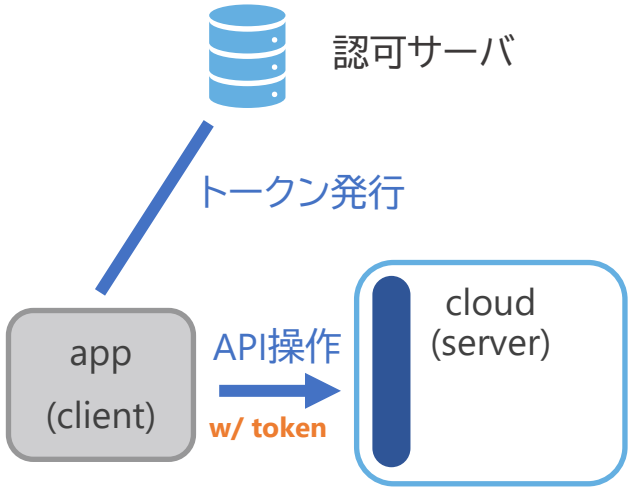
※会員向け実験クラウドでは
APIキーを使用

スマホアプリなど向け



Authorization Codeによる実現例

サーバ・サーバ間向け



Client Credentialsによる実現例

ELWAにおける認証・認可関連の規定方針

前提：サービス要件によって適切な方式は異なる

ベース仕様

API仕様部

セキュリティや通知機構などは、開発者が自由に選択できる余地を残し、柔軟な仕様とする

機器仕様部

＋

サービス仕様

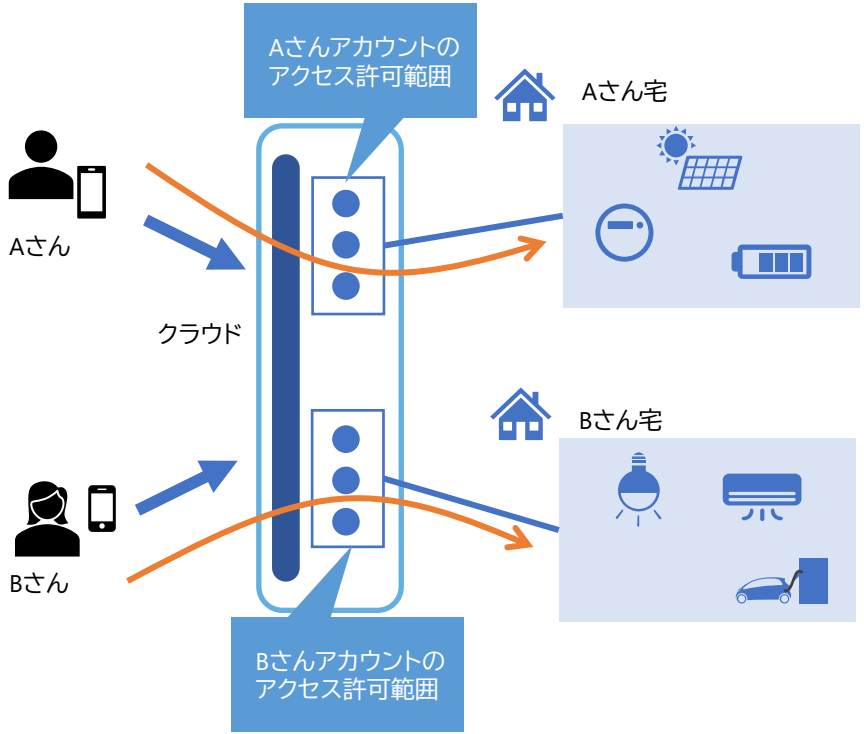
XXXサービス仕様
- ＋API集α
- セキュリティ方式A

YYYサービス仕様
- ＋API集β
- セキュリティ方式B
- 通知機構X

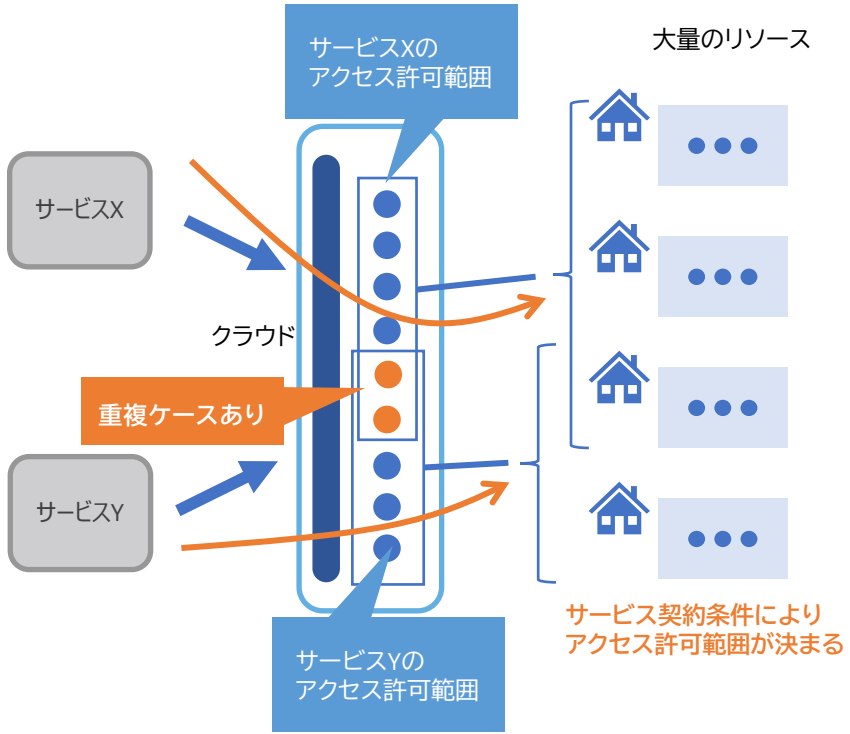
※現状、セキュリティ方式などは検討中

- 明確に採用すべきセキュリティ方式などを規定するのは「サービス仕様」にて
- 要件によっては各サービス間で共通仕様となるケースも想定される

マルチクライアント環境でのアクセス許可範囲の考え方



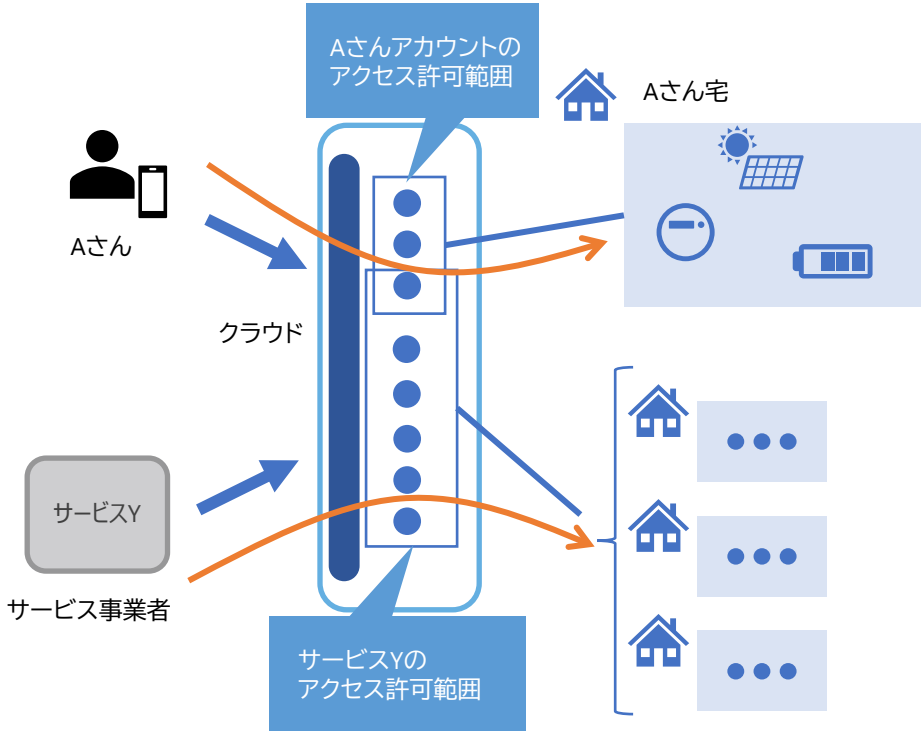
ユーザ毎に自宅機器操作を行うケース



B2Bにてサービス提供するケース

いずれのケースもユーザやクライアントの識別が必要

ユーザとクライアント(サービス)の混在ケースでは？



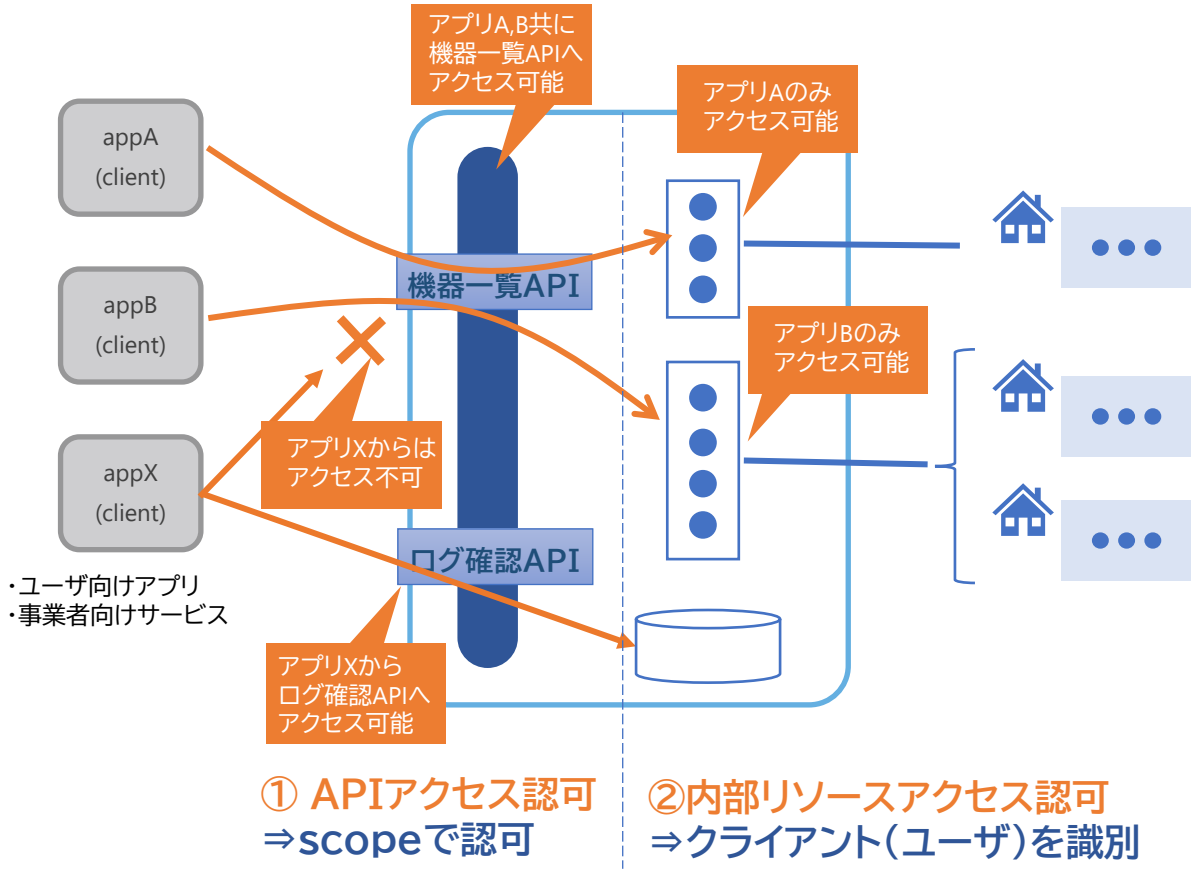
Authorization CodeもClient Credentialsも
リソースサーバへのAPI呼び出しは下記ヘッダを伴う

Authorization: Bearer <access_token>

ただし、アクセストークンをもとに、サーバ側
にてクライアントを識別できる仕組みが必要

ユーザ単位＆サービス単位での操作を両立するケース

APIアクセス認可と内部リソースアクセス認可の関係



①APIアクセス認可の実現:
OAuth2.0／OIDCでは、
アクセストークンを認可サーバへ
要求するとき、scope指定が可能

例)
- read, write
- admin, user

②内部リソースアクセス認可の実現:
アクセストークンは基本的にAPI
利用の可否のみにしか使用しない。
拡張機能の仕組みが必要となる。

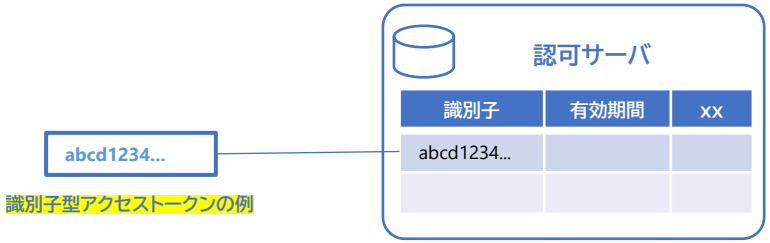
例)
- イントロスペクション
- JWT
- 実装でカバー(一体型)

参考）アクセストークンについて

大きく識別子型と内包型に分類される

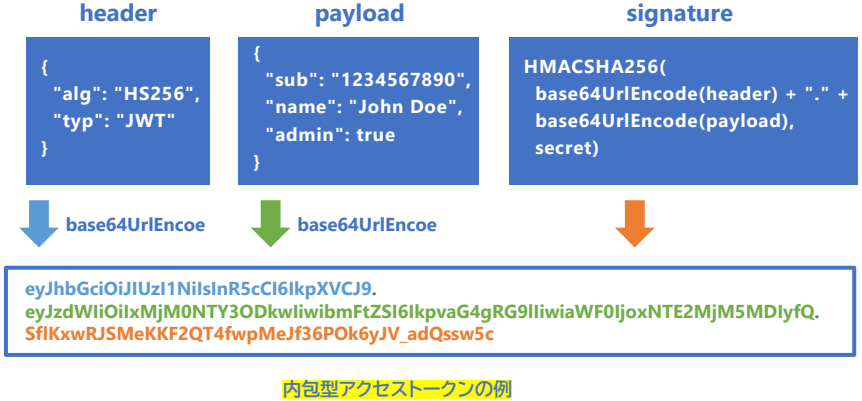
識別子型（Identifier-based）:

- 一般的にはランダムな文字列(一意に識別可能)。
- アクセストークンに紐付く情報を認可サーバーのデータベース内に保存し、リソースサーバーがこの識別子を使って認可情報を取得可能にする。



内包型（Self-contained）:

- アクセストークンに紐付く情報をアクセストークン自体の中に埋め込む。一般的にはJWT(JSON Web Token)形式が採用される。
- アクセストークンは自己完結しており、リソースサーバーはアクセストークンの中身を読むことで認可情報を取得。

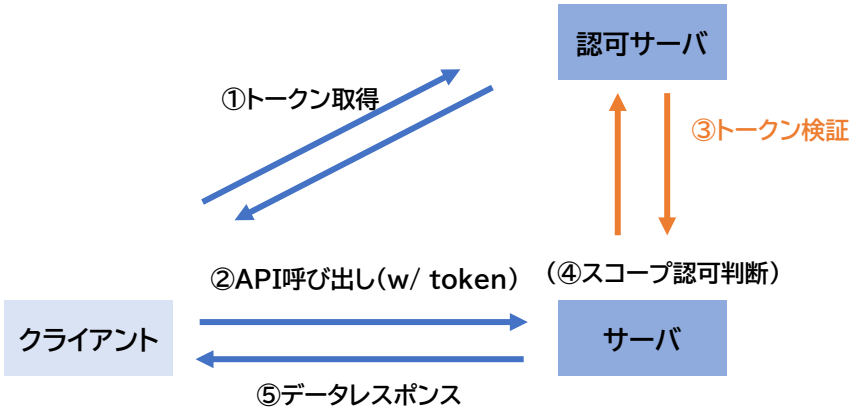


ハイブリッド型:

- 内包型アクセストークンを生成しつつ、それに付随するデータを認可サーバーのデータベース内に持つ。

内部リソースへのアクセス認可①:イントロスペクションの場合

アクセストークン(識別子型)を認可サーバへ送付し、トークンの有効性を確認



アクセストークンは任意の文字列(識別子型)でOK

③ リソースサーバから認可サーバのトークン確認用(イントロスペクション)エンドポイントへ送付し、アクセストークンの有効性を確認

ただし、応答はactiveのみ必須

```
{
  "active": true,
  "client_id": "app",
  "exp": 1674376730,
  "iat": 1674376130,
  "iss": "https://xxx.com",
  "token_type": "Bearer"
}
```

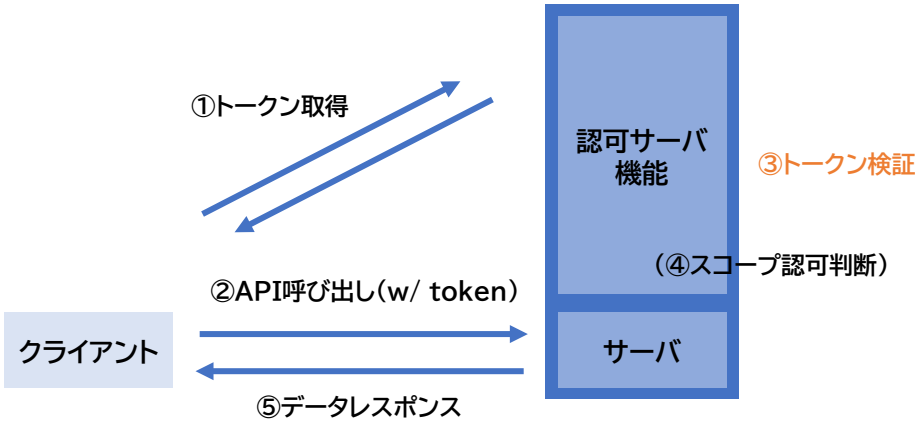
クライアントやユーザに関するクレームを必須化できれば識別可能

RFC7662: OAuth 2.0 Token Introspection

ちなみに、OAuth2.0を必須としているFHIRではイントロスペクションが必須。応答では、active以外にscope, client_id, expが必須。iss, subが推奨。
Token Introspection - SMART App Launch v2.2.0-ballot (fhir.org)

内部リソースへのアクセス認可②:認可サーバー体型の場合

アクセストークンをサーバ上の認可サーバ機能で検証



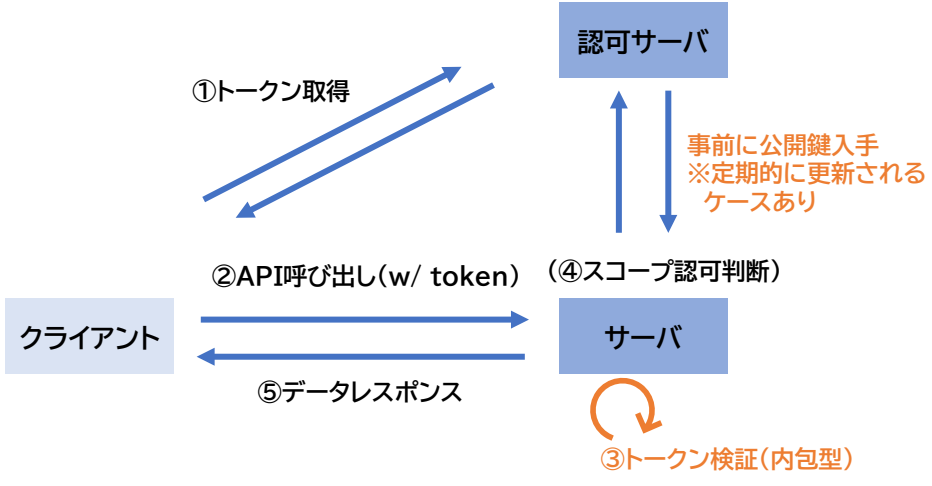
アクセストークンは任意の文字列(識別子型)でOK

③ リソースサーバからサーバ内部で認可サーバ機能を呼び出しアクセストークンの有効性を確認

どのクライアントにトークンを発行したか認識しているため、問題なくクライアントやユーザを識別可能

内部リソースへのアクセス認可③:JWTを用いた場合

アクセストークンをリソースサーバで検証



アクセストークンはJWT形式(内包型)

リソースサーバは事前に認可サーバが公開する公開鍵を入手

JWTの署名確認・解析によりクライアント情報を入手

RFC 9068:JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokensにて、クライアントやユーザ情報のクレームは必須化されている

RFC 9068でのリクエスト例

```
GET /as/authorization.oauth2?response_type=code
    &client_id=s6BhdRkqt3
    &state=xyz
    &scope=openid%20profile%20reademail
    &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
    &resource=https%3A%2F%2Frs.example.com%2F HTTP/1.1
Host: authorization-server.example.com
```

RFC 9068でのJWT例

```
{
   "typ": "at+JWT",
   "alg": "RS256",
   "kid": "RjEwOwOA"
}

{
   "iss": "https://authorization-server.example.com/",
   "sub": "5ba552d67",
   "aud": "https://rs.example.com/",
   "exp": 1639528912,
   "iat": 1618354090,
   "jti": "dbe39bf3a3ba4238a513f51d6e1691c4",
   "client_id": "s6BhdRkqt3",
   "scope": "openid profile reademail"
}
```

ELWA搭載リソースサーバでの内部リソースアクセス許可

認可サーバとリソースサーバ間にてクライアント・ユーザ情報の適切なやりとり必要

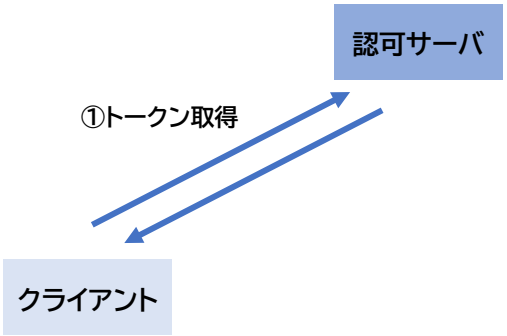
①～③いずれも対応可能だが、

- 認可サーバが外部にある場合：
  - イントロスペクション、JWTを活用。クライアントやユーザ情報を含むクレームを必須化が課題
  - キャッシュ利用による効率化、適切な無効化処理など考慮
- 認可サーバ機能をサーバ内に搭載する場合：
  - アクセストークンからクライアントやユーザ情報を導出する機能を搭載

- いずれのケースもクライアントは単にアクセストークンをヘッダ指定にてAPI呼び出しするのみ
- サーバはこのリクエストを受信後、クライアントやユーザ情報に基づきアクセス認可されているリソースへの操作後、適切に値を返却する
- クライアント情報（client_id）、ユーザ情報（sub）を一緒に扱う場合は、処理方針が必要

参考)クライアント側の課題①

認可サーバへのクライアント認証方式は複数バリエーション存在



client_secret_basic: ベーシック認証(RFC 7617)使用

```
POST <トークンエンドポイント> HTTP/1.1
Host: <認可サーバ>
Authorization: Basic base64化(<クライアントID>:<クライアントシークレット>)
Content-Type: application/x-www-form-urlencoded

#リクエストパラメータは省略
```

client_secret_post: リクエストパラメータ使用

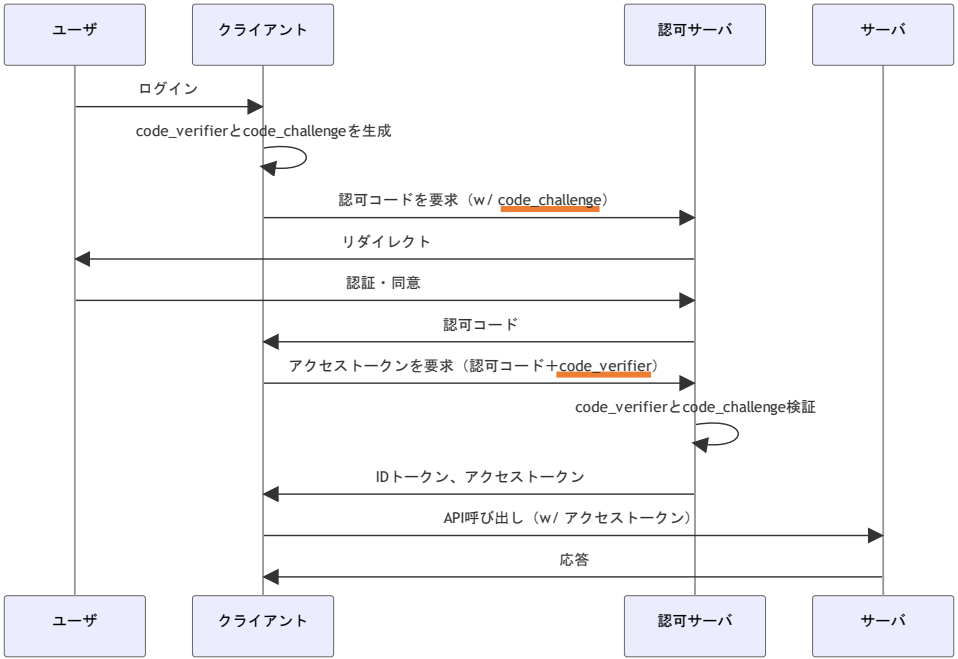
```
POST <トークンエンドポイント> HTTP/1.1
Host: <認可サーバ>
Content-Type: application/x-www-form-urlencoded

client_id=<クライアントID>
client_secret=<クライアントシークレット>
#他のリクエストパラメータは省略
```

トークンエンドポイントの
クエリーパラメータに
指定するケースも

- 他にも、client_secret_jwt、private_key_jwt、tls_client_auth、self_signed_tls_client_authといった方式あり
- Financial-grade API(FAPI)のセキュリティプロファイルによっては許可される方式に制限がある

参考)クライアント側の課題②

Authorization Codeでのパラメータに複数バリエーションあり



PKCE利用例

state
RFC 6749 - The OAuth 2.0 Authorization Framework
OAuthのCSRF対策

PKCE
RFC7636 - Proof Key for Code Exchange by OAuth Public Clients)
認可コード横取り攻撃対策

nonce
OpenID Connect Core 1.0
リプレイ攻撃の対策

05

まとめ

まとめ

ELWAにて検討中のAPI＆内部リソースへのアクセス認可方式の検討状況について紹介

- **ECHONET Lite Web API（ELWA）では、クライアントやユーザの権限に基づいたリソース管理を想定**
  - 同一APIを呼び出す場合でも、呼び出し元が異なれば返却リソースは異なりうる
- **上記要件に基づき、認証・認可モデルを考察**
  - OAuth2.0, OIDCを活用
- **ELWA搭載のリソースサーバの呼び出しはアクセストークンベース**
  - API認可は、scopeなど適切に利用
  - 内部リソースアクセス認可は、イントロスペクション／JWT ／認可サーバー体型より選択し実装
- **課題・今後の検討**
  - クライアント認証方式やPKCEなどのパラメータの選択？を検討
  - 既存認可サーバの対応状況の整理、共通仕様の絞り込みについて検討