1　システム構成

1.1　全体構成

本研究で作成するシステム全体のシステム図を 図 1.1 に示す．まず，畑内に設置した赤外線センサノード群で動物の侵入を検知し，そのデータをワイヤレス通信の LoRa 通信を用いてゲートウェイに集約する．ゲートウェイは集約されたデータを LTE 通信を用いて Internet 経由でクラウドサーバにデータを送信する．クラウドサーバではデータの管理やデータの解析，農業従事者への通知および解析結果の表示を行う．本論文ではクラウドサーバの解析アルゴリズム以外を担うため，次節でクラウドサーバの構成と詳細について述べる．

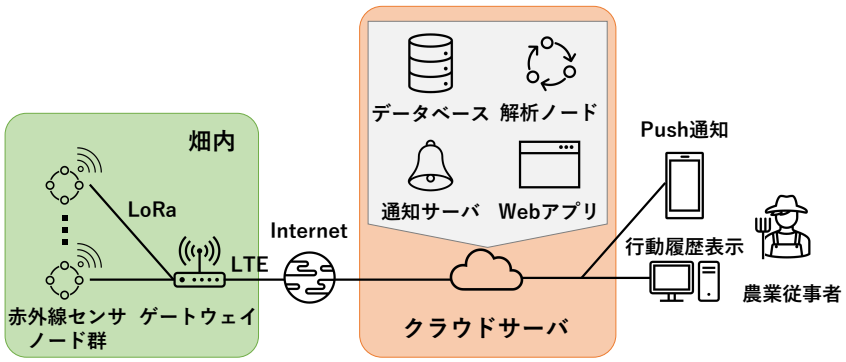

図 1.1: 全体のシステム図

1.2　サーバ構成

本システムのクラウドサーバは 図 1.1 にある通り，以下の 4 つの役割の担い，それぞれの役割が連携しながら処理を行う．

- データベース
- 解析ノード
- 通知サーバ
- Web アプリケーション

クラウドサーバの役割が 4 つあり，それぞれが連携するため処理の流れが複雑になっている．クラウドサーバの処理を整理するために DFD(Data Flow

Diagram) を 図 1.2 と 図 1.3 に示す．クラウドサーバの処理には畑のデータを送ってくるゲートウェイ，本システムに登録して通知や解析結果を受け取る農業従事者，センサデータや解析データ，農業従事者の登録者情報を保存するデータベースが関わっている．ゲートウェイからは畑で検知したセンサデータがクラウドサーバへ一方的に送信される．クラウドサーバでは送信されてきたセンサデータをデータベースに一時的に保存する．保存したセンサデータを用いて解析を行い，通知と解析後データの保存を行う．通知では事前に登録した農業従事者の登録者情報から Push 通知を行う．また，解析後データから行動履歴表示を行う Web アプリケーションを農業従事者へ提供する．

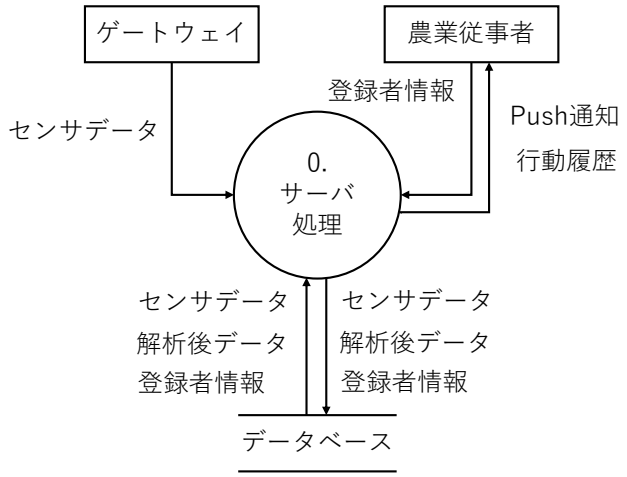

図 1.2: クラウドサーバの DFD(レベル 0)

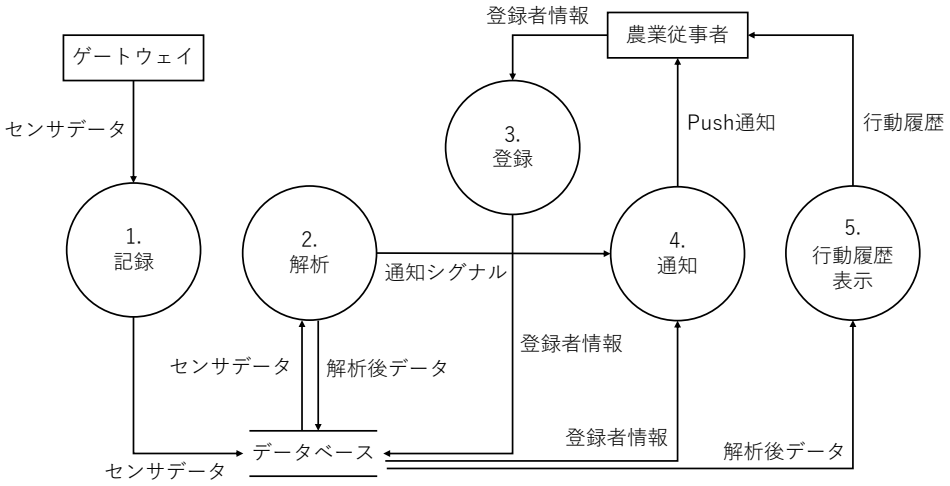

図 1.3: クラウドサーバの DFD(レベル 1)

クラウドサーバの DFD(図 1.2 と 図 1.3) を参考にクラウドサーバ上の構成を決める．本システムのクラウドサーバは文字通り，クラウド上にシステムを設置するため，開発環境 (手元の PC) との OS の違いといった環境差が生じる．そういった環境差を最小限にするために Docker を用いて Docker コンテナ内に各システムを構築する．また，クラウド内の 4 つのシステムを連携させるために Docker Compose を用いる．

Docker Compose を用いて作成するクラウドサーバの内部ネットワークを 図 1.4 に示す．クラウドサーバ内部に作成する Docker コンテナは以下の 6 つである．また，Docker コンテナごとの詳細については後述する．

- リバースプロキシ (Reverse Proxy)
- 通知サーバ (Notification-app)
- Web アプリケーション (Visualizer-app)
- データベース
  - データベースのアクセス制御 (Database-API)
  - データベース本体 (Database)
- 解析ノード (Analyzer-node)

また，Docker コンテナ間などの通信を行うために Docker ネットワークを以下の 3 つ用意する．

- surface
- inside
- db-bus

surface ネットワークでは Internet と接続する Docker コンテナのみ所属させる．inside ネットワークには基本的な Docker コンテナ全て所属させる．db-bus ネットワークにはデータベース管理の処理を担う Docker コンテナのみ所属させる．

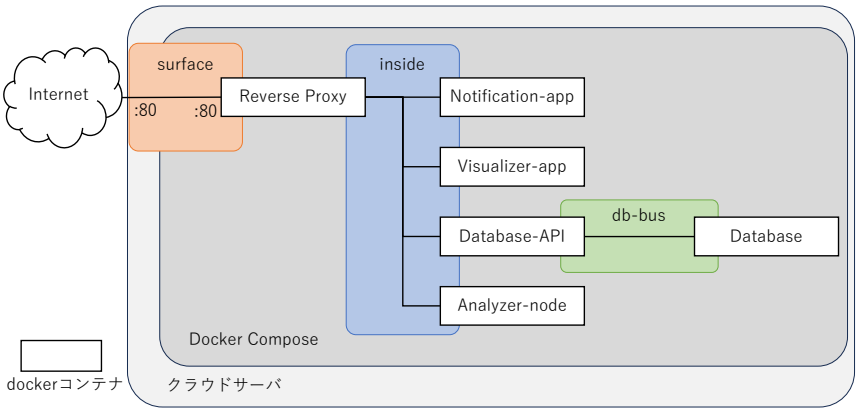

図 1.4: クラウドサーバのネットワーク図

1.2.1 リバースプロキシ (Reverse Proxy)

クラウドサーバには前述の 4 つの役割があり，それぞれについて Docker コンテナが作成される．しかし，クラウドサーバ内に 4 つのサーバを作成するとなると，ポートの衝突などを回避するために Docker コンテナごとに設定を細かく変える必要が出てくる．細かい設定が増えるとクラウドサーバ全体の設定を把握しづらくなり管理保守することが難しくなる．そこでリバースプロキシを用いて Internet とクラウドサーバ内部の Docker コンテナを仲介・制御する．リバースプロキシは外からのアクセスの種類によって内部のどのサーバと通信するかを制御する機能を持ったサーバのことである．リバースプロキシによって，クラウドサーバ内部に存在する複数機能を担う各 Docker コンテナへのアクセスをまとめて管理する．

実装するリバースプロキシの具体的な仕様としては，クラウドサーバにくる全てのデータアクセス (HTTP 通信) を役割ごとに適当な Docker コンテナに通信するため，surface ネットワークと inside ネットワークに所属する．

Nginx の Docker イメージをベースとし，/etc/nginx/nginx.conf を編集することでリバースプロキシサーバとして Docker コンテナを作成する．

リバースプロキシで行う転送処理は以下の通りである．

/notification： 通知サーバ (Notification-app)

/maps： Web アプリケーション (Visualizer-app)

/nodegw (POST)： データベース (Database-API)

1.2.2　通知サーバ (Notification-app)

通知サーバではレベル 1 の DFD 図 1.3 にある「4. 通知」で行うため．Push 通知を用いて農業従事者への通知を実現する．まず，Push 通知はスマートフォンやパソコンなどに搭載されている通知方法の一つで，画面を付けていない状態や別のアプリを利用中といった場面でも通知音やバナーといった方法で即時通知することができる．このような常に通知用の管理画面を開く必要がないことや即時性に着目して，通知を Push 通知を用いて実現することにした．しかし基本的な Push 通知の実装はその端末にあったアプリケーションを作成する必要があり，Android や iOS，Windows，MacOS といった対応させたい OS の数だけアプリケーションを作成する必要がある．内部構造が基本的に同じになるとは言っても OS ごとのアプリケーションを作成するのはコストが高く，バージョンアップといった保守の手間も非常に大きい．

そこで, WebPush というものを活用する．WebPush はブラウザを用いた Push 通知を行う仕組みであり, 一つの Web サーバを作成するだけで対応するブラウザを介して複数種類の端末で Push 通知を実現することができる．また WebPush は基本的なブラウザに対応しているため，簡単に複数種類の端末への Push 通知が実現することが可能である．

したがって，通知サーバでは登録された畑に動物の侵入があったことを Web-Push を用いて農業従事者に通知するサーバを実現する．具体的には WebPush で VAPID を用いてサーバの登録を行い，Node.js の Docker イメージをベースとして WebPush を行うサーバを実現する．

通知サーバはレベル 1 の DFD(図 1.3) においては「4. 通知」を主に担う．そのため「2. 解析」からの通知シグナルとデータベースから登録者情報を取得するために Docker ネットワークで Analyzer-node および Database-API と同じ inside に所属する．

1.2.3　Web アプリケーション (Visualizer-app)

Web アプリケーションではレベル 1 の DFD 図 1.3 にある「3. 登録」と「5. 行動履歴表示」を行う．行動履歴表示では特に侵入した動物の侵入経路の分かりやすさが重要である．そこで Leaflet.js というマップライブラリを用いる．Lealfet.js は地図の拡大縮小やマーカの表示など基本的な地図アプリの機能に加えて，線や多角形領域，ピンといったマーカを地図に投影することができるため採用する．

加えて，Web ページとして JavaScript(Leaflet.js) を利用するため，JavaScript の Node.js を用いることで利用する言語を統一し，管理を容易にする．

また，本システムは複数の農業従事者および畑を対象としてサービスを提供するため，どこの畑で，誰が管理者であるかを登録する必要がある．登録ページについて開発中のページを 図 1.5 に示す．

図 1.5: 開発中の登録ページ

Web アプリケーションの具体的な実装としては，Node.js の Docker イメージをベースとして HTML や CSS，Leaflet.js を含めた JavaScript を用いて実現する．Web アプリケーションはレベル 1 の DFD(図 1.3) においては「3. 登録」と「5. 行動履歴表示」を主に担う．そのためデータベースへの登録者情報の保存および解析後データの取得するため，Docker ネットワークで Database-API と同じ inside に所属する．

1.2.4 データベース (Database, Database-API)

データベースでは MySQL を用いたデータベースコンテナと Node.js を用いたデータベースコンテナ管理用の API コンテナの 2 つで構成する．API コンテナでデータベースアクセスを仲介することで，データベースアクセスの簡易化や誤操作の抑制を実現する．

Docker ネットワークとしては db-bus ネットワークに所属して，他の Docker コンテナからデータベースアクセスを受けるために Database-API コンテナのみ inside ネットワークにも所属する．

1.2.5 データ解析ノード (Analyzer-node)

データ解析ノードでは python イメージを主体としてデータベースに蓄えられる赤外線センサのデータを解析する役割を持つ．本コンテナでは Database-API

コンテナと通信してデータの取得および解析後のデータの保存を行い，通知サーバに通知シグナルを送信する．そのため，Docker ネットワークとしては inside ネットワークに所属する．

解析の詳細に関しては本論文範囲外であるため割愛する．

参考文献

[1] 農林水産省, “野生鳥獣による農作物被害状況の推移”, https://www.maff.go.jp/j/seisan/tyozyu/higai/hogai_zyoukyou/attach/pdf/index-31.pdf, 2024年1月18日参照.

[2] 大澤文孝, 浅井尚, “触って学ぶクラウドインフラ docker 基礎からのコンテナ構築”, 日経BPマーケティング, 2020年.

[3] 掌田津耶乃, “Node.js 超入門”, 株式会社 秀和システム, 2017年.

[4] StrongLoop, IBM, “Express - Node.js web application framework”, https://expressjs.com/, 2024年1月18日参照.

[5] Volodymyr Agafonkin, “Leaflet - a JavaScript library for interactive maps”, https://leafletjs.com, 2024年1月18日参照.

[6] Mozilla Foundation, “プッシュ API - Web API | MDN”, https://developer.mozilla.org/ja/docs/Web/API/Push_API, 2024年1月18日参照.

[7] M. Thomson, E. Damaggio, B. Raymor, Ed., “Generic Event Delivery Using HTTP Push”, https://datatracker.ietf.org/doc/html/rfc8030, RFC8030, December 2016, 2024年1月21日参照.

[8] Mozilla Foundation, “サービスワーカー API - Web API | MDN”, https://developer.mozilla.org/ja/docs/Web/API/Service_Worker_API, 2024年1月18日参照.

[9] M. Thomson, P. Beverloo, “Voluntary Application Server Identification (VAPID)”, https://datatracker.ietf.org/doc/html/rfc8292, RFC8292, November 2017, 2024年1月21日参照.

[10] Nginx, “nginx”, https://nginx.org/en/, 2024年1月18日参照.

[11] Mozilla Foundation, “POST - HTTP | MDN”, https://developer.mozilla.org/ja/docs/Web/HTTP/Methods/POST, 2024年1月22日参照.