**Bluehost Architecture for Pet Store**

**Objective:**

The objective of the document is to provide overview of the requirement and architecture that is being proposed to develop the solution based on the requirements.

**Requirements Overview:**

The basic requirement of an application is to provide the following services.

* Storing location data from cellular transmitters attached to pet collars and collected by units installed on cell towers.
* Returning historical location data for a given pet (basic users can see the last 24 hours while premium users can see the last 30 days of data).
* Searching for and returning contact information for opted-in owners of other pets within a 5-mile radius. [Note: This is premium functionality.]

Deliverables Requirement:

1. An architecture design of service(s). Documentation with visual examples encouraged.

2. A mock API service prototype. Include all source code, and it should be something that could

be run/interacted with locally by start-up engineering reps.

**Architecture Overview:**

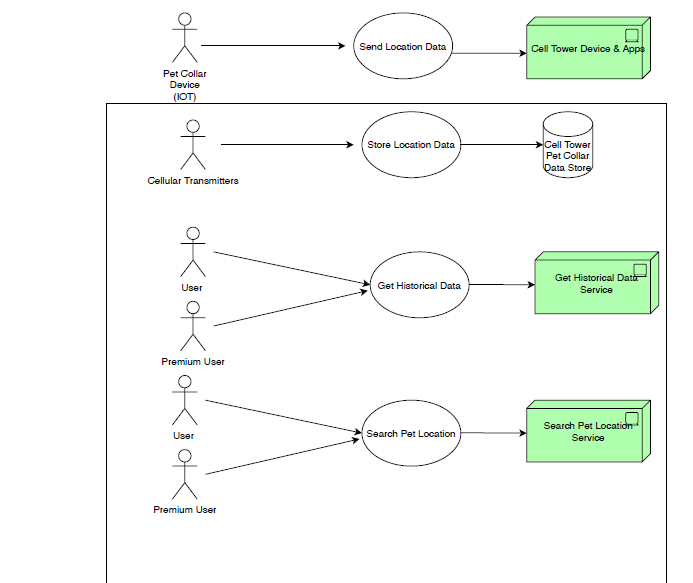
This section provides details of the architecture proposed to support the above requirement. It also covers the assumption that is being made to define the proposed architecture.

**Use Case View:**

The below diagram represents the use-case view of the requirements. The key use-cases are

1. Store Pet Locations
2. Get Historical Locations
3. Get Pet Owner Contact

In real-world, the “store Pet Locaitions” use-case, would be a IoT device integration. The Pet Collar devices emits the “location” data to cell tower and the IOT receiver application will receive the “location” data and stores the location data by invoking an API.



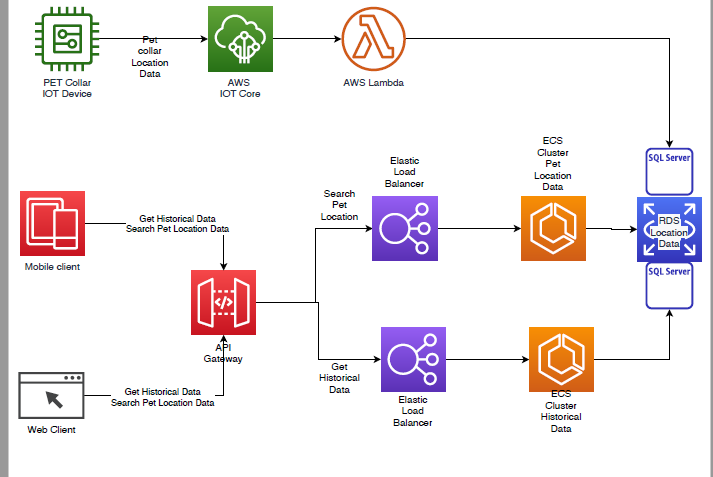
**Approach for Candidate Architecture:**

API First Approach design has been considered to provide the solution. As part of architecture development, it is necessary to design the “API Service Contract” (which I have created using Swagger) and it demonstrates how the API definition needs to be created.

Next step is choosing the right architecture for the services which should support highly available, horizontal scalable, secure and highly performance solution. In order to support these basic non-functional requirements, I have chosen to design the services using microservices architecture.

|  |  |
| --- | --- |
| Capability | Platform |
| To provide swift delivery to market with less CapEx | Cloud Platform (like AWS) has been chosen which provides the capability needed |
| To Provide API Capabilities, deliver service to multi channel like Web, Mobile | AWS API Gateway |
| To provide microservices capability | Docker, Kubernetes |
| To Provide highly scalable architecture | EKS has been chosen as Container Platform |
| Storing Pet Location and Owner Data (Needed for real-time operations) | For operational purpose it is recommended to have RDBMS. As we need to provide highly available, RDS MSSQL (or similar) is recommended |
| For API Service Implementation | Java, SpringBoot, Spring etc. is considered (assumption is that dev team is more comfortable with this open source platform) |

**Candidate Architecture:**



API Gateway – API Gateway component will expose the API to the mobile application/web client.

API implementation will have the actual implementation of the services.

**Assumptions:**

This section details the assumptions that have been made to arrive at the architecture.

* The solution requirement is to provide only the mockup API service definition and dummy implementation of the API.
* It is sufficient to only provide the mockup API definition and sample code to host the API and execute to produce results of the services.
* 3 API’s have been designed to support the functionality and assumptions I have considered.
  + Store Location Service: I have included only sample data model to show the location details that needs to be stored. (As in real time this would be a IoT invoked service and have not optimized to show the details)

**Alternate Architecture Considered:**

Architecture1 (Outlined above):

Pros:

Cons:

Alternate Architecture 1:

Pros:

Cons:

Alternate Architecture 2:

Pros:

Cons:

Architecture Decision and Recommendations:

For the purpose of proving the coding skills, I have created the following

* API Definition – (Defining using OpenAPI standard)
* Spring/SpringBott Code using DockerHub Editor and Code Generator
* Derby DB – Used as a temporary storage to prove that the data gets stored in backend.
  + Accessing data from Derby was developed using Java/JDBC code
* Git to checkout the code from GitHub
* Maven is used to build the package

**Service – Pet Collar – Run Book**